Основы информационной культуры

Даулеткулов А.Б.

ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ

Рекомендовано Министерством образования и науки Республики Казахстан для организаций образования

Алматы 2004

ББК 32.973.26-018.1
Д 21

Д 21 Даулеткулов А.Б. Олимпиады по информатике: Учебно-методическое пособие. -Алматы: ИНТ, 2004 - 242 с.

ISBN 9965-9293-5-1

Олимпиады по информатике среди школьников проводятся с середины 80-х годов. С начала 90-х годов основными условиями победы на олимпиадах по информатике принято считать успешную отладку на ЭВМ программ, решающих конкурсные задачи.

Книга ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ состоит из трех частей.

В первой части приведены задачи международных, республиканских и городских олимпиад по информатике среди школьников, а также задачи, использованные при подготовке сборных команд Казахстана по информатике. Рассмотрен ход решения и типичные ошибки.

Во второй части рассматриваются вопросы, связанные с технической, тактической и психологической подготовкой учащихся к соревнованиям.

В третьей части представлены материалы по технологии проведения школьных олимпиад по информатике: отбор конкурсных задач, работа жюри, порядок проведения личных и командных туров, проверка работ.

Рецензенты:

Ускенбаева Р.К. – кандидат технических наук, доцент, заведующая кафедры информатика КАзНТУ им. К.И.Сатпаева.

Мухамбетжанова С.Т. – кандидат педагогических наук, заведующая кабинетом информатики РИПК СО МОН РК.

ББК 32.973.26-018.1

Д240410000
00(05) - 04

ISBN 9965-9293-5-1

ОТ АВТОРА

Предисловие к первому изданию

Все началось весной 1993 года — именно тогда в Республиканском Центре Новых Информационных Технологий Министерства образования Республики Казахстан шло формирование жюри Республиканской олимпиады по информатике среди школьников. Больших трудов стоило мне осуществить свою школьную мечту — участвовать в Республиканской олимпиаде. Так впервые я принял участие в олимпиаде по информатике (как член жюри) и впервые написал две задачи для школьников.

В этом же году я стал заниматься подготовкой сборной команды Казахстана по информатике среди школьников и продолжал писать задачи.

Предлагаемая Вашему вниманию книга — попытка поделиться с Вами теми наблюдениями и мыслями, которые стали результатом участия в городских, республиканских и международных олимпиадах, работы в "Школе юного программиста" и со сборными командами Казахстана среди школьников по информатике.

Я выражаю свою искреннюю признательность своим старшим коллегам: В.М.Котову и И.А.Волкову (Белоруссия), профессору В.А.Каймину (г. Москва), В.В.Харасахалу и Б.К.Накисбекову (г. Алматы); моему другу И. Пашину (г. Темиртау); замечательному человеку, подвижнику олимпиадного движения Э.С.Ратобыльской (Белоруссия); членам жюри, руководителям команд и всем, кто занимается этим прекрасным и увлекательным делом, а так же моим ученикам - членам сборных команд Республики Казахстан 1993, 1994, 1996, 1997 годов. Благодаря вам, друзья, появилась эта книга.

Предисловие ко второму изданию

Прошло более пяти лет после выхода в свет первого издания книги "Олимпиады по информатике". За это время происходило бурное развитие олимпиадного движения. Это связано с развитием компьютерной техники. Резкое повышение скорости вычислений привело к тому, что традиционные задачи могли быть решены с использованием менее эффективных алгоритмов.

Это не могло повлиять и на характер олимпиадных задач. В первую очередь резко увеличилась размерность задач. Во вторых, резко уменьшилось допустимое время работы программы. В третьих, повысилась сложность и круг вопросов рассматриваемых в олимпиадных задачах. Все больше становится задач эвристического характера.

К сожалению, в последнее время олимпиадное движение Республики имеет ряд недостатков. Лакмусовой бумагой неблагополучия в этом являются результаты выступления наших школьников на Всемирных олимпиадах по информатике. Анализ работ и результатов городских, областных, республиканских олимпиад школьников по информатике указывает на низкий уровень подготовки учащихся к олимпиадам по информатике. Отсутствие базовых знаний, умений и навыков. Это, на мой взгляд, обусловлено рядом объективных и субъективных причин(**на IOI-2007 команда Казахстана заняла 3-е общекомандное место. прим. автора**).

Предлагаемое Вашему вниманию второе издание книги "Олимпиады по информатике" несколько отличается от первого издания. Во-первых, в него вошли новые задачи. Во вторых, небольшим изменениям подверглась вторая часть книги, связанная с технической подготовкой. Третья часть книги практически не изменилась, так как опыт прошедших пяти лет показал правильность идей и методологии заложенных в ней.

Автор будет рад если эта книга окажет помощь в подготовке к олимпиадам по информатике и привлечет в олимпиадное движение новых ребят. Замечания и пожелания направляете по адресу Daldijar@rambler.ru. Успехов Вам.

Искренне Ваш Даулеткулов Алдияр.

Часть I Олимпиадные задачи

ВВЕДЕНИЕ

Олимпиада - невидимый накал страстей: взлеты и падения, надежды и разочарования, слезы и радость, поражения и долгожданная победа. Все это крутится вокруг одного - Ее Величества Задачи.

Олимпиадные задачи - это особый мир, в каждой из них (я имею в виду настоящую олимпиадную задачу) есть своя изюминка, которая является ключом к решению. В этом смысле тот, кто решает задачу, подобен сыщику, распутывающему сложное дело. Все улики налицо, но часть из них - ложные, а часть - может привести к решению. Сможете ли Вы как Шерлок Холмс разгадать замысел автора задачи или пойдете по ложному следу?

Перед началом чтения несколько советов.

Советы для учащихся

Совет 1. Не торопитесь читать решение задачи. Попробуйте решить задачу самостоятельно.

Совет 2. Перед решением задачи внимательно прочитайте условие, выявите неясные места, сформулируйте вопросы и запишите их.

Совет 3. Попробуйте придумать тесты для проверки правильности решения.

Совет 4. Больше времени работайте без компьютера, записывайте все свои мысли и алгоритмы решения на бумаге - это поможет Вам разобраться, что и почему Вы упустили из виду в случае неудачи.

Совет 5. Не бегите к компьютеру, сломя голову. Решите задачу на бумаге. Время работы на компьютере должно быть минимальным. Если Вы правильно решили задачу, то ввод программы в компьютер не займет много времени.

Совет 6. Не огорчайтесь, если Вы не смогли полностью решить задачу. Х.Р. Капабланка сказал: “Мало есть выигранных мною партий, которые научили бы меня столько, сколько мои проигрыши”

Совет 7. Не зазнавайтесь, если решили все правильно - у автора всегда найдется “камень за пазухой”.

Советы для преподавателя

Совет 1. Поощряйте ребят, которые работают с бумагой. Старайтесь, чтобы они при решении задач работали за компьютером минимальное время.

Совет 2. Задачи приведены в хронологическом порядке, то есть по времени их использования или написания, поэтому Вы можете рассматривать их в любом порядке. Однако тему “лабиринты” лучше рассматривать в порядке, представленном в книге.

Совет 3. После решения задачи предложите ребятам самим придумать задачи на данную тему.

Совет 4. Смотри Совет 1.

Совет для Всех. Можно не слушать мои советы.

Если Ваша программа успешно прошла все тесты, но Ваше решение задачи отличается от подхода, предложенного в книге, пожалуйста, опишите это решение и пришлите мне.

1.1 Две очень “простые” задачи

С этой задачи я, обычно, начинал первое занятие в “Школе юных программистов”. Она предлагалась на тренировочных сборах сборной команды СССР по информатике.

Задача 1.

Условие задачи: Дана сфера радиуса R, центр которой находится в центре координат. Необходимо определить количество точек, находящихся в сфере, с целочисленными координатами. Если точка лежит на поверхности сферы, то ее надо учитывать.

Как правило, первым ребята предлагают следующий вариант решения :

```
var
R,x,y,z:integer;
S       :longint;
Begin
readln(R);
S:=0;
        for X:=-R to R do
                for Y:=-R to R do
                        for Z:=-R to R do
        if (sqr(x)+sqr(y)+sqr(z))<=sqr(R) then inc(S);
writeln(S);
end.
```

Я просил ребят ввести программу в компьютер и произвести вычисления для нескольких значений R. Например: R=1, R=5, R=10. Проверяю, чтобы программа работала правильно. Затем прошу определить время выполнения счета при значениях: R=100, R=400 , R=1000 и R=10000. Как правило, время выполнения программы при значении R=10000 никто не определял (вычисления велись на 286 компьютерах. **прим. автора**).

Очень часто при проведении олимпиад в условие задачи ставится ограничение времени на выполнение программой расчетов. В настоящее время, на олимпиадах практически всех уровней, ограничение по времени, как правило, составляет 1 - 2 секунды, в крайних случаях — от 10 секунд до одной минуты (на олимпиаде IOI-1999 время выполнения одной из задач составляло 180 секунд. **прим. автора**).

Мало написать правильную программу, необходимо, чтобы она давала результат за приемлемое время.

В проверочных заданиях всегда присутствует тест для проверки эффективности алгоритма.

Задача 2. Задача из учебного пособия “Основы информатики и вычислительной техники” (автор В.А.Каймин)

Условие задачи: Определить площадь треугольника используя координаты его вершин..

В учебнике приведен следующий алгоритм решения данной задачи:

```
  арг     X[1:3]:вещ
          Y[1:3]:вещ
  рез     S:вещ
нач
a:=длина(X[1], Y[1], X[2], Y[2])
b:=длина(X[2], Y[2], X[3], Y[3])
```

```
c:=длина(X[3], Y[3], X[1], Y[1])
p:=(a+b+c)/2
S:=√(p−a)(p−b)(p−c)
кон
```

Присваивание a:=длина(X[1], Y[1], X[2], Y[2]) означает здесь обращение к вспомогательному алгоритму, который вычисляет длины стороны треугольника.

```
арг u1,v1,u2,v2:  вещ
рез  r: вещ
нач
   du:=u1-u2
   dv:=v1-v2
   r:= √du * du + dv * dv
кон
```

Попробуйте написать программу и проверить ее работу со следующими данными:

x1=0,y1=0,x2=10,y2=0,x3=0,y3=10

Площадь треугольника должна быть равна 50. Убедитесь в правильности выполнения программы, а затем запустите ее при значениях:

x1=0,y1=0,x2=2,y2=1,x3=4,y3=3

Площадь треугольника должна быть равна 1. Сравните ответ, полученный с помощью программы. На первый взгляд, расхождение минимальное, но иногда это стоит призового места.

Это случилось на олимпиаде ICI-94 (г. Могилев, Республика Беларусь). Один из участников команды Минского лицея, реально претендовавший на первое место, получил много нулевых оценок при тестировании одной из задач второго тура.

В задаче требовалось определить прямую линию, которая делит два треугольника, заданных на плоскости, на равные по площади части. При определении площадей он использовал формулу Герона, с помощью которой вычислялась площадь треугольника в вышеприведенной программе. Однако из-за не достаточной точности вычисления площадей, его программа в ряде случаев не могла определить положение линии.

Обращайте внимание на точность вычислительных методов, которыми Вы пользуетесь!

1.2 Незнакомый знакомый

После выхода в свет книги “Московские олимпиады по информатике”, отношение к задачам, связанным с лабиринтом, резко ухудшилось, и они на долгое время исчезли из практики олимпиад. Возможно, по этой причине задача на тему лабиринта была полной неожиданностью для участников Республиканской олимпиады по информатике 1994 года.

Ниже приведен текст данной задачи.

ТЕСЕЙ И МИНОТАВР

Республиканская олимпиада 1994г., I-тур

автор Даулеткулов А.

Тесею предстоял опасный бой с Минотавром. Когда отвели Тесея и всех обреченных на растерзание в Лабиринт, Тесей привязал у его входа конец клубка и пошел по запутанным, бесконечным переходам Лабиринта, из которого невозможно было найти выход. Тесей постепенно разматывал клубок, чтобы найти по нитке обратный путь...

Кун “Легенды и мифы древней Греции”

Но, что это? Клубок кончился. Что делать? Тесей решается оставить клубок и преследует Минотавра. Минотавр стремится к своему логову. Если Тесей найдет Минотавра раньше, чем он достигнет своего логова, то Тесей его победит, иначе Минотавр становится непобедимым.

За один ход Тесей и Минотавр могут продвинуться на одно свободное поле вверх-вниз, вправо или влево (движение по диагонали запрещено).

Имея план Лабиринта, положение Тесея и Минотавра, а также путь движения Минотавра к логову, помогите Тесею найти Минотавра (в случае, если они одновременно находятся на одном поле, Тесей найдет Минотавра). После того, как он находит и побеждает Минотавра, выведите его кратчайшим путем из лабиринта.

План лабиринта и путь Минотавра к логову находятся во входном файле "Testx.TXT" (x - номер варианта).

Блок информации входного файла начинается с данных о размерах лабиринта: первая цифра - количество строк, вторая - количество столбцов (см. Пример 1.). Затем следует кодовая информация о лабиринте, код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр 1 и 0 (1 - стена лабиринта, 0 - свободное пространство). После информации о плане лабиринта следует информация о положении Тесея, Минотавра, логова Минотавра, а также путь Минотавра к логову.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла план лабиринта, отображает ее на экране дисплея в виде прямоугольной матрицы N*M (N - количество строк, M - количество столбцов). При этом символ звездочка (*) обозначает стены лабиринта, символ пробел ( ) - свободное пространство, буква (Т) - исходное положение Тесея, буква (М) - исходное положение Минотавра и буква (Л) - логово Минотавра.
3. Считывает пути движения Минотавра по лабиринту к логову и отображает его на экране цифрами от 1 до K. Если Тесей находит Минотавра, то в этом поле ставится символ "X", в противном случае выводится сообщение: "Встреча невозможна".
4. Из точки встречи отображает кратчайший путь к выходу из лабиринта (цифрами от 1 до L).

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. N должно быть не менее 3 и не более 20.
2. M должно быть не менее 5 и не более 40.
3. Имя входного ASCII файла должно быть "Testx.Txt", где x - номер варианта.

Пример 1.

```
 5,7
 1011111
 1000001
 1110011
 1000001
  1110111
2,2          (Тесей)
 4,6          (Минотавр)
 4,2          (Логово)
 4,5         *
 3,6         *
  3,4         * (Путь Минотавра)
```

4,4 *
4,3 *
4,2 *

Решение задачи условно состоит из двух этапов:

1. Нахождение кратчайшего пути до встречи с Минотавром.
2. Нахождение кратчайшего пути к выходу из лабиринта.

Основной трудностью в понимании условия задачи являлось то, что в отличие от традиционного лабиринта, введен дополнительный персонаж. Это, на мой взгляд, и сбило многих участников с толку. Но если внимательно рассмотреть условие задачи, то станет очевидным, что задача практически не отличается от традиционной задачи о «лабиринтах». Минотавра в данной интерпретации можно считать неподвижным. Задача сводится к определению той клетки лабиринта, для которой номер хода Минотавра совпадет с номером хода Тесея.

Для некоторых данных решения не существует. Это может быть в том случае когда Тесей и Минотавр разделены непроходимой стеной. Для получения отрицательного ответа придется решать задачу полностью. Но даже если такой стены нет, задача может не иметь решения. Представьте себе шахматную доску, на которой находятся две фишки. Они могут двигаться на одно поле вверх, вниз, вправо или влево. Оказывается, для того чтобы фишки одновременно оказались на любой из клеток шахматной доски они первоначально должны находиться на клетках одного цвета.

1.3 Начнем с конца

На Всемирной олимпиаде по информатике 1994 года была предложена задача про альпинистов. Приведенную ниже задачу также можно отнести к задачам этого класса.

“ЛОРЕН-ДИТРИХ”

Республиканская олимпиада 1994г., II-тур

автор Даулеткулов А.

- Куда теперь ехать? - с тоской закончил Козлевич

- Куда податься?

Остап помедлил, значительно посмотрел на своего рыжего компаньона и сказал:

... - В Черноморск, ... Бензин ваш - идеи наши.

Козлевич остановил машину и, все еще упираясь, хмуро сказал:

- Бензину мало.

- На пятьдесят километров хватит?

- Хватит на восемьдесят.

- В таком случае, все в порядке. Я вам уже сообщил, что в идеях и мыслях у меня недостатка нет. Ровно через шестьдесят километров вас прямо на дороге будет поджидать большая железная бочка с авиационным бензином. Вам нравится авиационный бензин?

- Нравится, - застенчиво ответил Козлевич.

И. Ильф, Е. Петров “Золотой теленок”

Требуется доехать из города Арбатов до города Черноморск, расстояние между которыми S км. Автомобиль “Лорен-Дитрих” расходует на 1 км. пути C литров бензина и может везти с собой не более M литров бензина.

Предполагается, что одной заправки (M литров) может не хватить на весь путь. Поэтому Остап предлагает Козлевичу двигаться следующим образом: автомобиль заправляется и едет до определен-

ной точки (в дальнейшем - точка дозаправки), оставляет часть бензина для того, чтобы использовать его в дальнейшем и возвращается в необходимую для него точку для дозаправки и т.д.

ПОСТАНОВКА ЗАДАЧИ:

Написать программу, которая выполняет следующее:

1. Вводит с клавиатуры расстояние от Арбатова до Черноморска в км. (S), расход бензина на 1 км. пути (C), максимальное количество бензина (M), которое может везти автомобиль.

Повторить запрос на ввод данных, если они некорректны.

2. Определяет оптимальный режим движения автомобиля из Арбатова в Черноморск. Решение является оптимальным, если:

а) количество бензина, необходимое для переезда, минимально;

б) количество точек дозаправки минимально.

3. Выводит полученные результаты на экран:

а) количество бензина, необходимое для переезда из Арбатова в Черноморск;

б) количество точек дозаправки;

в) положение этих точек (в км. от начала пути).

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. Программа не должна воспринимать входные данные, если S <1 или S>1000, C<1 или C>10,а также M<1 или M>100.

2. Расстояние между пунктами дозаправки - не менее 1 км.

Как многие оптимизационные задачи, эта задача решается с конца. Допустим, что мы доехали до города Черноморска. Тогда, при минимальном расходе горючего, в баке не должно остаться бензина. Попробуем определить точку последней дозаправки машины. Очевидно, она должна находится на расстоянии M/C от конечной точки, т.е. в этой точке у машины должен быть полный бак A(1). От точки последней дозаправки сделаем шаг назад на 1 километр. Определим, какое минимальное количество бензина A(2) необходимо для того, чтобы в точке последней дозаправки в машине был полный бак. При этом надо учесть расходы на транспортировку этого количества бензина и количество поездок, которые должна сделать машина. Опять делаем шаг назад и определяем минимальное количество бензина A(3), необходимое для того, чтобы количество бензина в предпоследней точке было A(2). Эти действия повторяем до тех пор, пока мы не достигнем города Арбатов. Количество бензина, полученное нами, и будет минимальным количеством бензина, которое необходимо для переезда из Арбатова в Черноморск. Теперь необходимо оптимизировать количество точек дозаправки. Для этого надо сгруппировать точки дозаправки, используя как ключ количество переездов между точками дозаправки. Очевидно, в точках, где количество переездов между точками дозаправки изменяется, и надо расположить пункты дозаправки.

При написании программы, пожалуйста, обратите внимание на два особых случая.

1. Расстояние от Арбатова до Черноморска меньше расстояния, которое может проехать машина с полным баком.

2. Количество бензина, которое может везти машина, меньше ее расхода на 1 километр пути.

Примечание: Существует «Вычислительное» решение данной задачи. Оно расмотренно в книге Мартина Гарднера «Математические головоломки и развлечения» и называется задачей о преодолении пустыни. Оказывается сумма ряда вида

$$S = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + ...$$

Может быть сколь угодно большой все зависит от количества слагаемых в этой сумме.

1.4 Старый знакомый

Задача Тесей и Минотавр, как отмечалось, является стандартной задачей про лабиринт. Решение ее строится с применением “поиска в ширину”. Следующая задача возникла как попытка выяснить, что произойдет, если разрешить проходить через стенки лабиринта. Когда количество стенок, которые можно проходить, не ограничено, то задача - стандартная задача о лабиринте. А если количество стенок, которые можно проходить, ограничено?

УЗНИК ЗАМКА ИФ

Отборочный тур, сборная Казахстана 1994г.
автор Даулеткулов А.

- Какое несчастье! - произнес голос.

- Боже мой! Что такое? - спросил Дантес

- Я ошибся, несовершенство моего плана ввело меня в заблуждение, отсутствие циркуля меня погубило, ошибка в одну линию на плане составила пятнадцать футов в действительности, я принял вашу стену за наружную стену крепости!

А. Дюма “Граф Монте-Кристо”

Узник хочет выйти из замка Иф. Он может двигаться по свободным клеткам (коридоры и камеры) или прорывать стены лабиринта, но при этом он теряет скорость продвижения. В замке есть охрана, и встреча с ней не сулит узнику ничего хорошего.

План лабиринта и исходное положение узника находятся во входном файле “TESTx.TXT” (x - номер варианта теста).

Блок информации во входном файле начинается с данных о размерах лабиринта: первое число - количество строк, второе - количество столбцов (Пример 1). Затем следует кодовая информация о лабиринте. Код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр 1 и 0 (1 - стена лабиринта, 0 - свободное пространство), символ “+” (“-”) означает, что охранник находится в этой клетке на нечетном (четном) ходе. После информации о плане лабиринта следует информация о положении узника в лабиринте (первое число - номер строки, второе - номер столбца), количество стенок, которое может прорыть узник (С), скорость прорывания стенок (К - количество ходов, затрачиваемое на прохождение через стену).

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.

2. Считывает из входного файла план лабиринта, отображает его на экране дисплея в виде прямоугольной матрицы NxM (N - количество строк, M - количество столбцов). При этом символ звездочка (*) обозначает стены лабиринта, символ пробел ( ) - свободное пространство, символ (x) - исходное положение узника.

3. Определяет возможность выхода узника из лабиринта.

4. Отображает кратчайший путь выхода из лабиринта (числами от 1 до L i , где 1 - первый ход), а в случае невозможности выхода, выдает сообщение: “Нет решения”.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. $3 < N < 20$, $5 < M < 40$, $0 < C < 10$, $1 < K < 10$.

2. Можно двигаться вверх, вниз, вправо или влево (двигаться по диагонали нельзя).

3. На полях, отмеченных символом “+” (“-”), можно находиться только на четном (нечетном) ходе.

4. Имя входного ASCII файла должно быть “TESTx.TXT” , где x - номер варианта.

Пример 1.

5,7
1011111
1- -0001
1110011
10+0001
1111111
2,5,1,2 (координаты узника, количество стенок, которые он может прорыть, скорость прохождения стенок)

1.5 Идем направо - песнь заводим, налево - сказки говорим

Когда мы рассматривали программу определения площади треугольника методом Герона, то выяснили, что не во всех случаях она дает правильное значение площади. Это связано с вычислением квадратного корня. Однако существуют и другие методы отыскания площади треугольника. Кроме того, эти методы можно применить и к нахождению площади произвольного многоугольника (смотри часть 2).

ЗА ЗОЛОТОМ НА РУЧЕЙ ИНДИАНКИ

автор Даулеткулов А.

Смок равнодушно посмотрел на золото, налил себе кружку кофе и сел. Джой почувствовала что-то недоброе и с беспокойством посмотрела на Смока. Малыш был обижен невниманием товарища.

- Почему ты не радуешься? - спросил он - Ведь тут целое богатство, а ты и посмотреть на него не желаешь.

Прежде чем ответить, Смок отхлебнул глоток кофе.

- Малыш, знаешь ли ты, что наши заявки напоминают Панамский канал ?...

- Ты хочешь сказать, что мы ничего не получим?

- Даже на десять фунтов меньше, чем ничего.

Джек Лондон "Смок Белью и Малыш"

На плоской поверхности установлено N столбов, между которыми натянута проволока. Количество столбов, их расположение на плоскости, номера столбов, которые соединены между собой, даны во входном файле "TESTx.TXT" (x-номер варианта теста).

Входной ASCII файл имеет следующий формат (Пример 1):

N - количество столбов;

X1, Y1 - координаты 1-го столба;

…...

XN, YN - координаты N-го столба;

NL1-NL2 - номера столбов, которые соединены между собой;

.....

NLm-NLh - номера столбов, которые соединены между собой.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла положение столбов и их соединения.
3. Определяет количество замкнутых участков и их площади.
4. Выводит в выходной файл количество замкнутых участков и их площади в порядке убывания (с точностью до целого знака).

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. $1 < N < 1000$.
2. Имя входного ASCII файла должно быть "TESTx.TXT", где x - номер варианта.
3. Имя выходного ASCII файла должно быть "ANSx.TXT", где x - номер варианта.

Пример 1.

5

2,4

4,8
6,7
3,4
1000,1
1-4
2-5
1-5
5-2

1.6 Раскинулось море широко, или о пользе частицы “НЕ”

Попытайтесь нарисовать отрезок прямой линии на экране дисплея. Если отрезок горизонтальный или вертикальный, то он будет выглядеть как отрезок прямой линии. Но если отрезок имеет наклон, то он будет представлять собой ломаную линию.

Когда мы работаем с компьютером, мы должны помнить об одной очень важной проблеме - разрешение экрана конечно. Можно выделить класс задач, основанных на конечном разрешении “компьютерного” холста. Ниже приведена задача из этого класса. Толчком к ее написанию послужила задача на распознавание геометрических фигур, предложенная на олимпиаде ICI-94 (г. Могилев).

ОСТРОВА

Первенство г. Алматы, 1995г., I-тур автор Даулеткулов А.

Вдруг с тихим мелодичным звоном белая кнопка на пульте выдвинулась вперед, наливаясь алым; экран рядом осветился - тонкая красная линия сверху вниз пересекала какие-то смазанные, неясные контуры, какую-то схему.

Карта!...

Дж. Лэрд “Наследство Бар Ригона”

Карта размерностью N строк и M столбцов представляет собой компьютерный холст. На нем изображен участок моря с островами, нарисованными с помощью взаимно непересекающихся контуров. Контур каждого острова нарисован своим символом, который не может повторяться. В качестве символа для заполнения фона выбран символ (*) звездочка (Пример 1).

Карта находится во входном файле “TESTx.TXT” (x - номер варианта теста).

Входной ASCII файл имеет следующий формат:

1- я строка (N)
2- я строка (M)
3- я строка (1-я строка холста)
4- я строка (2-я строка холста)
.....
N+2 - я строка (N - я строка холста)

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла план карты.
3. Определяет количество островов (L) и их площади.
4. Выводит в выходной файл общее количество островов и их площади, отсортированные в порядке убывания площадей.

Формат выходного файла:

количество островов < L >

Символ <изображение символа> - площадь < S1>

.....

Символ <изображение символа> - площадь < SL>

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 3 < N < 10000 , 5 < M < 132.

2. Имя входного ASCII файла должно быть “TESTx.TXT”, где x - номер варианта.

3. Имя выходного ASCII файла должно быть “ANSx.TXT”, где x - номер варианта.

4. Рабочая программа должна быть представлена в виде исполняемого модуля с именем “УXXX.exe” или виде текстового файла (для интерпретатора Бейсик) с именем “УXXX.BAS”, где У - номер класса, XXX - номер участника.

5. Время прохождения одного теста - не более 10 минут.

Пример 1.

```
**********
*1111*****
1****1***
*11111****
**********
```

С данной задачей у автора связано несколько неприятных воспоминаний. В первоначальном варианте задачи разные острова могли обозначаться одинаковыми символами. Один изпредполагаемых тестов как раз и был подготовлен для проверки умения различать эти острова. Однако за час до начала тура, еще раз просматривая условие задачи, я обнаружил следующее - если строго следовать духу задачи, то, при наличии этого условия, появляется побочное решение. Участник может заявить, что каждый единичный символ является отдельным островом (таков масштаб карты!!!), и ему просто надо определить, какие символы используются и приписать им площадь, равную единице.

1.7 Опять вода

Очень часто толчком к написанию задачи является эпизод или отрывок из какой либо книги. Не была исключением и предлагаемая Вам следующая задача.

ТРИ ДОКУМЕНТА

Матчевая встреча сш 28, РСФМШИ, “Школа юного программиста» РЦ НИТ, Алматы, 1995г., автор Даулеткулов А.

Извлеченные из бутылки клочки бумаги были наполовину разъедены морской водой. Из почти расплывшихся строк еле-еле можно было разобрать кое-какие непонятные слова. В течение нескольких минут Гленарван внимательно рассматривал клочки. Он поворачивал их то так, то сяк, разглядывая на свет и изучая малейшие следы уцелевших слов, которые пощадило море...

...- А можно уловить смысл документа? - спросила леди Гленарван.

- Трудно утверждать что-нибудь определенное, дорогая Элен, уцелевшие слова слишком отрывочны.

- А может быть, они дополняют друг друга? - спросил майор.

- Несомненно, - ответил Джон Манглс. - Вряд ли морская вода уничтожила во всех документах одни и те же слова. Сличая сохранившиеся обрывки фраз, мы, в конце концов, доберемся до их смысла.

Жюль Верн “ Дети капитана Гранта”

К Вам попала бутылка с сообщением. Бросивший ее в море, для страховки написал текст сообщения несколько раз. Но листки с текстами оказались сильно подпорчены морской водой: она смыла не только часть букв, но и испортила бумагу. У вас есть N клочков бумаги (разной величины), на которых находится текст одного и того же документа. Часть букв в них стерта. Обрывки документов находятся во входном файле “TESTx.TXT” (X - номер варианта теста). Входной ASCII файл имеет следующий формат (Пример 1):

Z1 (текст на первом клочке);
Z2 (текст на втором клочке);
.....
ZN (текст на N-том клочке).

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла части документа.
3. Восстанавливает исходный текст документа.
4. Выводит в выходной файл текст исходного документа.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 1<N<100.
2. Имя входного ASCII файла должно быть “TESTx.TXT”, где x - номер варианта.
3. Имя выходного ASCII файла должно быть “ANSx.TXT”, где x - номер варианта.

Пример 1.

```
Z1
1995 год,     марта
              каникулы
Z2
од, 25 м
Z3
199
Весенние кан
Исходный текст:     1995 год, 25 марта
                    Весенние каникулы.
```

Задача достаточна проста. Для того, чтобы получить исходный документ, надо наложить части документа добиваясь максимального совпадения. Можно организовать три массива: A1() - промежуточный документ, A2() - сравниваемая часть документа, A3() - результирующий документ.

Начинать рассмотрение лучше с наиболее крупных частей документа, поэтому перед началом сравнения рекомендуется отсортировать их по размеру.

1.8. Кошмарный сон, или все еще о лабиринте

- Мы сами знаем, что она не имеет решения, - сказал Хунта, немедленно ощетиниваясь. - Мы хотим знать, как ее решать.

- К-как-то ты странно рассуждаешь, К-кристо... К-как же искать решение, к-когда его нет? Б-бессмыслица какая-то ...

- Извини, Теодор, но это ты очень странно рассуждаешь. Бессмыслица - искать решение, если оно и так есть. Речь идет о том, как поступить с задачей, которая решения не имеет. Это глубоко принципиальный вопрос, который, как я вижу, тебе, прикладнику, к сожалению, недоступен...

А. Стругацкий, Б. Стругацкий “Понедельник начинается в субботу”

Внимание!!! Следующая задача, наверное, наиболее сложная из всех задач, написанных мною. Я до сих пор не уверен - есть ли у нее общее решение.

Читатель, наверно, уже определил явную любовь автора к задачам, связанным с лабиринтом. Более того, идет молва, что если он (автор) в жюри - жди задачи на лабиринт.

Идея задачи "Подземелье Джафара" была предложена членом сборной команды Казахстана - учеником 10 класса Берлизевым А (сш 28 г.Алматы), во время олимпиады ICI-94,

ПОДЗЕМЕЛЬЕ ДЖАФАРА

Первенство г. Алматы, 1995г., I-тур автор Даулеткулов А.

Каждую ночь Они стоят перед моими глазами... Мои друзья, не вернувшиеся из этого проклятого подземелья...

После отчаянной схватки, мы уничтожили охрану Джафара, но он успел скрыться в своем подземелье. Преследуя его, мы, не задумываясь, бросились за ним. Подземелье представляло собой громадный лабиринт. Мы бежали по его бесконечным запутанным переходам, пытаясь отыскать Джафара. Через некоторое время мы остались вдвоем с Али, остальные мои друзья затерялись где-то в лабиринте...

Тут раздался зловещий смех Джафара, заманившего нас в ловушку. Пол под нами начал проваливаться. Мы побежали с Али, не останавливаясь...

Каждую ночь Они стоят перед моими глазами... Мои друзья, не вернувшиеся из этого проклятого подземелья...

План лабиринта и исходное положение Ваших друзей находятся во входном файле "TESTx.TXT" (x - номер варианта теста).

Блок информации входного файла начинается с данных о размерах лабиринта: первое число - количество строк, второе - количество столбцов (Пример 1). Затем следует кодовая информация о лабиринте. Код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр 1 и 0 (1 - стена лабиринта, 0 - свободное пространство). После информации о плане лабиринта следует информация о количестве (L) и о положении Ваших друзей в лабиринте (первое число - номер строки, второе - номер столбца).

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.

2. Считывает из входного файла план лабиринта, отображает его на экране дисплея в виде прямоугольной матрицы NxM (N - количество строк, M - количество столбцов). При этом символ звездочка (*) обозначает стены лабиринта, символ пробел ( ) - свободное пространство, символ (x) - исходные положения Ваших друзей.

3. Определяет возможность выхода всех друзей из лабиринта.

4. Отображает кратчайшие пути выхода их из лабиринта (числами от 1 до L i , где 1 - первый ход).

5. В случае невозможности выхода хотя бы одного из друзей, выдает сообщение: "Я проиграл".

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. $3 < N < 20$, $5 < M < 40$, $2 < L < 10$.

2. За один ход необходимо продвинуться на одно свободное поле вверх-вниз, направо или налево (двигаться по диагонали или стоять на месте нельзя).

3. Поле, пройденное хотя бы одним человеком, становится непроходимым.

4. Имя входного ASCII файла должно быть "TESTx.TXT" , где x - номер варианта.

Пример 1.

6,7
1011111
1000001
1110011
1000001
1110001
0111101
2 - (количество друзей)
2,5
4,2

Перед решением задачи ответьте на следующий вопрос: **существует ли возможность вывести друзей из лабиринта, если их количество больше, чем количество выходов?** Если вы ответили "нет", то прочитайте внимательно эпиграф к задаче.

1.9 Ямщик, не гони лошадей...

Мы уже сталкивались с использование сортировки для уменьшения числа рассматриваемых вариантов. Предлагаемая Вашему вниманию следующая задача также может служить иллюстрацией применения сортировки для уменьшения числа рассматриваемых вариантов.

РУЧЕЙ МОНО

1-Отборочный тур, сборная Казахстана 1996г.
автор Даулеткулов А.

Смок неторопясь ехал к ручью Моно. Он боялся утомить своих собак до главной гонки. Он изучал тропу и отмечал места, где ему придется менять собак. В этом состязании приняло участие столько людей, что все пространство в сто десять миль было похоже на один сплошной поселок. По всему пути были расставлены собачьи подставы для смены упряжек. Фон Шредер, принимавший участие в состязании исключительно из спортивных видов, имел одинадцать упряжек, то есть мог менять собак каждые десять миль. Аризона Билл удовлетворился восемью упряжками. У Толстого Олафа было семь упряжек - столько же, сколько у Смока. Кроме них, в состязании принимало участие свыше сорока человек. Гонки с призом в миллион долларов даже на золотоносном Севере случаются не каждый день...

Д.Лондон "Смок Белью и Малыш"

К старту готовятся N участников, длина трассы S км, у каждого есть по M(i) упряжек, каждая из которых может бежать с одной из следующих скоростей: V(i, 1) км/час - L (i,1) км ,V(i, 2) км/час - L (i ,2) км,V(i, 3) км/час - L (i,3) км. Эти данные находятся во входном файле "TESTx.TXT" (x - номер варианта теста).

Входной файл имеет следующий формат:

< N > 1-я строка файла (количество участников)
< M1 > 2-я строка файла (количество упряжек у первого участника)
< L11,V11,L12,V12,L13,V13> 3-я строка файла (характеристики первой упряжки)
.....
<Ln1,Vn1,Ln2,Vn2,Ln3,Vn3> K-я строка файла (характеристики последней упряжки последнего участника)
<S> K+1-я строка файла (длина трассы).

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла.
3. Определяет оптимальное использование упряжек на трассе и победителя гонки.
4. Вводит в выходной файл:
- номер победителя (1-я строка файла);
- номер 1-й упряжки, пробег в км, скорость (2-я строка файла);
.....
- номер n -й упряжки, пробег в км, скорость (n+1-я строка файла);
- в случае невозможности доехать до финиша, выдает сообщение: “Нет решения”.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. $10<S<200$, $3 < N < 50$, $1 < M < 15$, $0 < L < 30$, $1<V<30$.
2. Упряжки можно расставлять только через целое количество км.
3. Упряжка может идти только с одной скоростью.
4. Имя входного ASCII файла должно быть “CTESTx.TXT” , выходного ASCII файла “CANSx.TXT”, где x - номер варианта.

1.10. Ох, уж эти лабиринты, или приключения в Королевстве Ном

В книге Гарднера “Математические головоломки и развлечения” приведена одна очень поучительная задача. Дано 6 спичек. Необходимо сложить их так, чтобы они образовали 4 равносторонних треугольника. Решить данную задачу на плоскости нельзя. Для ее решения необходимо выйти в трехмерное пространство. Полученная фигура представляет собой тетраэдр, грани которого и являются равносторонними треугольниками.

Для решения задачи иногда необходимо избавится от стереотипов.

ПРИКЛЮЧЕНИЯ В КОРОЛЕВСТВЕ НОМ

Олимпиада ICI-96, г. Могилев, Республика Беларусь, I-тур, задача 1, автор Даулеткулов А.

- Как же ты уйдешь без моего согласия? - спросил король.

- Очень просто, - ответила Дороти, - Все, что нам надо сделать, так это уйти той же дорогой, которой мы пришли сюда.

- Да что ты говоришь! - насмешливо воскликнул король, - Где же тогда тот ход, по которому вы сюда пришли?

И действительно, прохода нигде не было видно, так как он давно уже закрылся.

Л.Фрэнк Баум “ Приключения в Королевстве Ном”

Дороти и ее друзья находятся в лабиринте. Они пытаются найти выход, но коварный король Ном через определенный промежуток времени (5 ходов) может изменить характер лабиринта. Помогите им выйти из лабиринта.

План лабиринта и его модификации находится во входном файле “TESTx.TXT” (x - номер варианта теста).

Блок информации входного файла начинается с данных о размерах лабиринта: первое число - ширина лабиринта (N - количество строк на уровне), второе - длина лабиринта (M - количество столбцов на уровне), третье - количество различных видов лабиринта (H - количество лабиринтов) (Пример 1). Затем следует кодовая информация о лабиринтах. Код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр 0 и 1 (1- стенки лабиринта, 0 - свободное пространство). Далее следует информация о положении Дороти и ее друзей, и порядок изменения лабиринта.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.

2. Считывает из входного файла план лабиринта и его модификации. Отображает исходный лабиринт на экране дисплея в виде прямоугольной матрицы NxM (N - количество строк, M - количество столбцов). При этом символ звездочка (*) обозначает стены лабиринта, символ пробел ( ) - свободное пространство, символ (x) -исходное положение Дороти и ее друзей.

3. Определяет возможность выхода Дороти из лабиринта.

4.Через каждые 5 ходов отображает новую модификацию лабиринта на экране дисплея, как указано в п2. Символ (x) - положение Дороти и ее друзей в этот момент.

5. В случае невозможности выхода, выдает сообщение: “Нет решения”.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1.10< =N <=20, 10 <= M <=30 , 1< H <=5, максимальное количество ходов 50.

2. Можно двигаться вверх-вниз, вправо или влево (двигаться по диагонали нельзя).

3. Если при смене лабиринта положение Дороти и ее друзей совпало с положением стены, они погибают.

4. Имя входного ASCII файла “ TEST*.TXT”, где * - номер теста.

Пример 1
5,5,5
11111 - лабиринт 1
10000
1101111011
10001
11111 - лабиринт 2
10001
10100
10111
11111
11111 - лабиринт 3
10011
10000
11111
11111
11011 - лабиринт 4
11001
10101
11101
11101
11111-лабиринт 5
11001
10101
10001
11011
2,2
3,3,1.4,5,3,5,4,1,1

Как и многие задачи, связанные с лабиринтами, эта решается с использованием метода поиска в ширину. Так как вид лабиринта изменяется, то в отличие от традиционных лабиринтов, где путник не

возвращается на уже пройденную клетку, в нашем случае возможно посещение одной и той же клетки. Таким образом, после каждого хода путник должен “забыть”, какие клетки он уже посещал, и рассматривать их в числе возможных для движения. Наиболее просто это сделать, если представить лабиринт многоэтажным зданием, где каждый i -ый этаж имеет вид лабиринта на i -ом ходу. В этом случае, возможное положение путника на i-ом ходу проецируется на i+1-ый этаж, и находятся возможные положения путника на i+1 -ом ходу. Эти действия надо повторять до тех пор, пока не найден выход из лабиринта или не закончился лимит ходов.

1.11 Три простые задачи

Ниже приведены задачи, предложенные на областных олимпиадах по информатике 1997 года.

Задача 1.

Областные олимпиады по информатике 1997г., I-тур, задача 1 автор Даулеткулов А.

Дан шар радиуса R, центр которого находится в центре координат. Необходимо определить количество кубиков, полностью находящихся в шаре. Ребро кубика равняется единице. Вершины любого из кубиков совпадают с точками, имеющими целочисленные координаты. Кубик не находится в шаре, если хотя бы одна точка, находящаяся внутри кубика или на его поверхности, лежит за поверхностью шара.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Считывает с клавиатуры значение R.
2. Определяет количество трехмерных квадратов, находящихся в шаре и выводит его на дисплей.

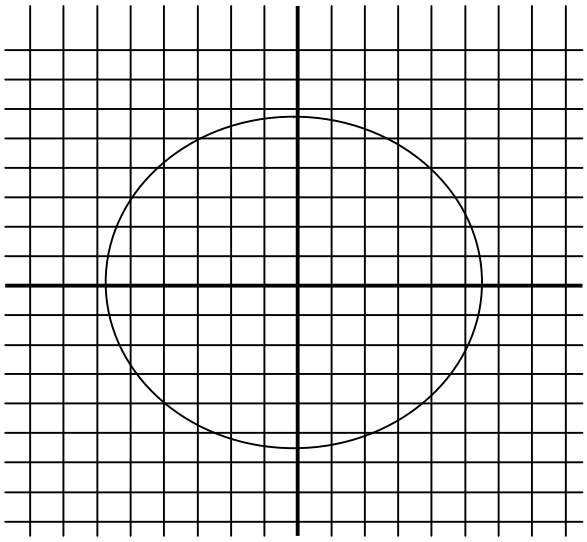
ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 1 <=R <=10000.

Как видно из условия, эта задача очень напоминает первую задачу о количестве точек в сфере. Видимая простота задачи сыграла злую шутку с ребятами. Большинство из них, недолго думая, предложили решение задачи, аналогичное рассмотренной в пп.1.1, и при проведении тестирования с R>=1000 вышли за временные рамки.

Некоторое сокращение времени работы программы достигается учетом симметрии сферы, то есть можно определить, сколько кубиков принадлежит одному из квадрантов сферы, а затем это количество умножается на 8. Однако, как уже отмечалось, этот “лобовой” подход не дает возможности получить решения при больших значениях R за приемлемое время. Главным недостатком “лобового” подхода является полный перебор.

Для того, чтобы понять принцип решения данной задачи, рассмотрим её двухмерный аналог.



Определим, сколько квадратиков лежат внутри правого верхнего квадранта окружности.

Квадрат лежит внутри окружности тогда и только тогда, когда его правый верхний угол находится внутри или на самой окружности. Двигаясь вдоль окружности, мы определяем самые правые квадраты, лежащие внутри окружности. При этом значение координат по x правых верхних углов будут равны количеству квадратов, лежащих левее крайнего квадрата. И они, безусловно, будут находиться внутри окружности. Таким образом, вместо полного перебора всех квадратов мы просматриваем только квадраты, лежащие у границ.

Задача 2.

Областные олимпиады по информатике 1997г., II -тур, задача 1 автор Даулеткулов А.

Для N точек на плоскости, заданных случайным образом, найти три точки, образующих треугольник с наибольшей площадью.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1.Считывает из входного файла координаты точек на плоскости.

2.Определяет треугольник с наибольшей площадью.

3.Выдает на дисплей координаты вершин и площадь треугольника с наибольшей площадью.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1.1<=N<=10000 , -100 <= Хкоорд.<= 100, -100 <= Укоорд.<= 100.

2. Имя входного ASCII файла должно быть “TESTx.TXT”, где x - номер варианта.

Задача относится к “переборным” задачам. Ясно, что при полном переборе всех вариантов, программа не уложится во временные ограничения (см. техническое ограничение N <=10000). Поэтому необходимо уменьшить количество рассматриваемых вариантов. Для этого, на первом этапе решения задачи, рекомендуем построить выпуклый многоугольник, включающий в себя все точки (точки лежат либо внутри, либо являются вершинами многоугольника). На втором этапе - рассматривать только те точки, которые являются вершинами данного многоугольника.

Задача 3.

Областные олимпиады по информатике 1997г., II -тур, задача 2 авторы Усков А., Даулеткулов А.

Шеф отдела МИ-6 секретной службы Ее Величества Дж. Дж. Нервничал:

- Срочно передайте шифрограмму агенту 007. Но не забудьте, что после последней контузии наш Джеймс помнит только четыре арифметических действия.

.....

- 01010101101, свихнуться можно с этой двоичной системой. - Думал Джеймс Бонд, читая текст шифрограммы.

В Вашем распоряжении имеется две шифрограммы и их исходные тексты (длина сообщений L и K соответственно), Вам необходимо выявить ключ и, используя его, расшифровать перехваченное сообщение. Шифрограмма и исходный текст представляют собой последовательность нулей и единиц.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Считывает с клавиатуры 1-й расшифрованный текст сообщения и его зашифрованный вариант.
2. Считывает с клавиатуры 2-й расшифрованный текст сообщения и его зашифрованный вариант.
3. Считывает с клавиатуры зашифрованный текст сообщения.
4. Определяет исходный текст сообщения.
5. Выдает на дисплей исходный текст сообщения.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 1<=L<=20, 1<=K<=20.

Задача достаточно простая. Ребятам надо было придумать как можно больше методов шифрования, использующих четыре арифметических действия. Пробуя их на известных шифрограммах, определить случаи совпадения ключей. Далее применить ключ и действие к зашифрованному сообщению.

1.12 Кое-что о графах

Очень часто на олимпиадах встречаются задачи на тему графов. Отсутствие данных задач (в явном виде) в моем арсенале привело к созданию следующих задач. Хочу напомнить, что задачи с лабиринтами - это тоже задачи на графах.

УТИНЫЕ ИСТОРИИ

Республиканская олимпиада 1997г., II-тур
автор Даулеткулов А.

- Я покажу этому выскочке, кто самый богатый селезень в мире

- Дядя Скрудж, а что нужно сделать? - спросила Поночка.

- Дети мои, мы должны за сорок дней добраться до УтинГрада, где мы сравним свои богатства.

- А если мы опоздаем?

- Тогда все пропало, главное - прибыть не позднее сорока дней.

Имеется N городов, пронумерованных числами от 1 до N. В каждом городе за одно посещение можно взять K единиц сокровищ (K>0). Для каждого города указаны города, в которые можно попасть. Эти данные приведены во входном файле “2TESTx.TXT” (x-номер варианта теста).

Входной ASCII файл имеет следующий формат:

N,M
K1:a12:a13.....:a1n
a21:K2 :a23.....:a2n
.....

an1 an2.....Kn,

где N - количество городов, M - максимальное количество дней, Ki - количество сокровищ, которое можно получить за одно посещение в i-ом городе, aij - наличие связи между городами. (aij =1, если из города i можно попасть в город j, aij =0, если из города i нельзя попасть в город j.)

Скрудж МакДаку и его племянникам необходимо не более, чем за M дней, добраться из СелезеньТауна до УтинГрада. За время путешествия им необходимо собрать максимально возможное количество сокровищ. Переезд из города в город занимает один день. СелезеньТаун обозначен номером 1, а УтинГрад - N.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.

2. Считывает из входного файла данные о сокровищах, количестве городов и соединений между ними.

3. Определяет:

а) оптимальный маршрут движения;

б) количество собранных сокровищ;

в) количество дней, проведенных в пути.

4. Выводит на экран:

а) количество собранных сокровищ;

б) количество дней, проведенных в пути;

в) в случае невозможности достижения УтинГрада за положенное количество дней, выдает сообщение: “Нет решения”.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 2 <= N <= 100, 1<=M<=50.

2. Имя входного ASCII файла должно быть “2TESTx.TXT”, где x - номер варианта.

В этой задаче мы имеем дело с ориентированным графом. Она аналогична задаче “Обезьяна и бананы” (см. Часть 2). Поэтому я рекомендую решать эту задачу с конца.

ЗАВЕЩАНИЕ СТАРОГО АГИ

Республиканская олимпиада 1997, II- тур, задача 1

авторы Тюрюшкин А., Даулеткулов А.

- Дети мои, - выдавил старый султан. - Я скоро умру, и мое огромное государство перейдет к вам. Но я знаю, что вы часто ссоритесь, друг с другом, поэтому я хочу, чтобы вы жили дружно. Для этого я разделил государство так, что каждый город одного из вас не соединен с его же городом...

В стране есть N городов, которые могут быть соединены дорогами. Количество городов и номера городов, которые соединены между собой, даны во входном файле “TESTx.TXT” (x-номер варианта теста).

Входной ASCII файл имеет следующий формат:

N - количество городов;

NL1-NL2 - номера городов, которые соединены между собой;

.....

NLm-NLh - номера городов, которые соединены между собой.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.

2. Считывает из входного файла количество городов и дороги между городами.

3. Определяет, между каким минимальным количеством сыновей можно разделить государство указанным способом.

4. Выводит в выходной файл минимальное количество сыновей и одно из возможных распределений городов между ними.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. $1 < N < 1000$.
2. Имя входного ASCII файла должно быть “TESTx.TXT”, где x - номер варианта.
3. Имя выходного ASCII файла должно быть “ANSx.TXT”, где x - номер варианта.
4. Выходной ASCII файл должен иметь следующий формат:

K - минимальное количество сыновей
N1-M1 - номер города и условный номер сына
.....
Nm-Mk - номер города и условный номер сына.

Вторая задача - задача о правильной раскраске графа.

Раскраска называется правильной, если цвета вершин не совпадают для любых смежных вершин. Граф, для которого существует правильная k-раскраска, называется k-раскрашиваемым. Минимальное число k, при котором граф является k-раскрашиваемым, называется “хроматическим числом” данного графа. Для определения “хроматического числа” используется метод генерации максимальных независимых множеств графа.

1.13 А если идти по диагонали?

В традиционных лабиринтах их план, как правило описывается с помощью матрицы. В следующей задаче план лабиринта описан необычным способом. Помимо этого, путнику в нем разрешается ходить и по диагонали. Эти, на первый взгляд, несущественные отличия приводят к интересным следствиям.

В СТРАНЕ РУДОКОПОВ

Республиканская олимпиада 1997, II-тур, задача 2

автор Даулеткулов А.

...Вдруг перед путниками открылась огромная пещера с каменными стенами и потолком. Она была так велика, что дальний край ее терялся во мраке. Элли в испуге прижалась ко Льву.

- Как тут пусто и страшно! - прошептала она.

А.Волков “Урфин Джюс и его деревянные солдаты”

Путник находится в лабиринте. У него имеется план лабиринта, с помощью которого Вы должны помочь ему выйти из лабиринта. План лабиринта представляет собой тетрадный лист в клетку, на котором отрезками прямых линий обозначены стены лабиринта.

План лабиринта во входном файле “TESTx.TXT” (x - номер варианта теста). Блок информации входного файла начинается с данных о размерах тетрадного листа: первое число - ширина (N - количество столбцов), второе - длина (M - количество строк), третье- количество стенок (K) (Пример 1). Затем следует кодовая информация о лабиринте, которая представляет собой характеристику стенок (координаты начала и конца отрезка). Далее следует информация о положении путника (первая цифра - столбец, вторая - строка).

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.

2. Считывает из входного файла план лабиринта.

3. Определяет возможность выхода путника из лабиринта и его кратчайший путь.

4. В случае невозможности выхода, сразу выдает сообщение: “Нет решения”.

5. Выводит в выходной файл (Пример 2):

а) минимальное количество ходов;

б) путь путника из лабиринта.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1.10<= N <=40, 10 <= M <=40 , 3<= K <=30.

2. Можно двигаться по свободным клеткам вверх-вниз, вправо-влево или по диагонали (клетка свободна, если через нее не проходит стена) (Пример 3) .

3. Левый верхний квадрат на листе имеет координаты (1,1), значения координат концов отрезков целочисленны (начинаются и оканчиваются в центре клеток).

4. Имя входного файла “ TEST*.TXT”, где * - номер теста.

5. Имя выходного файла “ ANS*.TXT”, где * - номер теста.

Пример 1.

10, 10, 5 (N,M,K)

1,1:10,0 (N1,M1:N2,M2)

1,1:0,10

1,10:10,10

10,10:10,6

10,1:10,4

4,5 (Nx,Mx)

Пример 2

6 минимальное кол-во ходов

5,5

6,5

7,5

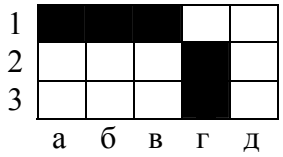
8,5

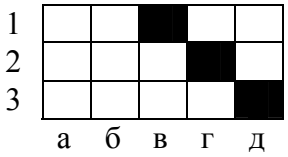
9,5

10,5

Данная задача состоит из двух задач. Первая - построить лабиринт. Вторая - найти кратчайший выход. Если с построением лабиринта у участников соревнований проблем практически не было, то со второй частью в полном объеме никто не справился. Почему?

Рассмотрим два случая





На левом рисунке показан вариант с двумя стенами лабиринта, которые не соединены между собой. Тогда, по условию задачи, из клетки 2в можно пройти в клетку 1г. На правом рисунке имеется стена, идущая по диагонали, поэтому пройти из клетки 2в в клетку 1г нельзя. К сожалению, у всех ребят путник просачивался сквозь стену. Для решения этой проблемы надо было присвоить клеточкам цвета по номеру стенки, проходящей через неё и ввести дополнительное ограничение при определении возможных ходов путника.

1.14 Колобок, Колобок, куда катишься?

Вашему вниманию представляется еще одна задача на тему лабиринта. Она была опробована на двух соревнованиях.

ГОЛОВОЛОМКА Л.Д.УИТТКЕРА

Личное первенство, Задача 2, Вязье-97, Международный компьютерный лагерь, Республика Беларусь.
Первенство РСФМШИ 1997, I- тур, г. Алматы
автор Даулеткулов

Пространственные лабиринты встречаются нечасто. Их изредка используют психологи для исследования процессов обучения животных.

Как правило, головоломка имеет вид коробочки с отверстием в крышке. Бросив туда стальной шарик, вы должны выкатить его через другое отверстие в боковой стенке... Как-то раз Граймз почти год безуспешно бился над разгадкой головоломки Уитткера, но все его усилия так и не привели к успеху

М.Гарднер

Имеется лабиринт, внутри которого помещен шарик. Наклоняя лабиринт на 45 градусов вперед (1), вправо (2), назад (3) или влево (4), необходимо перекатить его в указанное место.

План лабиринта находится во входном файле “BTESTx.TXT” (x - номер варианта теста).

Блок информации входного файла начинается с данных о размерах лабиринта: первое число - ширина лабиринта (N - количество строк), второе - длина лабиринта (M - количество столбцов) (Пример 1). Затем следует кодовая информация о лабиринте. Код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр 0 и 1 (1 - стенки лабиринта, 0 - свободное пространство), цифрой 2 обозначено исходное положение шарика, цифрой 3 - конечное положение.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла план лабиринта.
3. Определяет:

а) минимальное количество действий, которое необходимо выполнить для перевода шарика из начального положения в конечное;

б) путь шарика.

4. Записывает в выходной файл количество действий, порядок действий и положение шарика после каждого действия (Пример 2). Если задача не может быть решена, записывает в выходной файл сообщение: “Нет решения”

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 3<= N <=10, 3 <= M <=10 .
2. Размеры шарика сравнимы с ячейками лабиринта. При наклоне шарик практически мгновенно достигает следующего устойчивого положения.
3. Имя входного файла “ BTEST*.TXT”, где * - номер теста.
4. Имя выходного файла “ ANS*.TXT”, где * - номер теста.

Пример 1

Пример 2

4

8,10	2,2,9
1111111111	3,7,9
1200000001	4,7,9
1010100001	4,7,2
1010000101	
1011110001	
1111100101	
1300000001	
1111111111	

1.15 Ожерелье для любимой, лоскутное одеяло, или верить ли приметам?

“Новое – это хорошо забытое старое”

Народная мудрость

Комбинаторные задачи, в последнее время, редкие гости на олимпиадах по информатике. Поэтому было интересно узнать, как наши ребята умеют решать данные задачи.

ОЖЕРЕЛЬЕ ДЛЯ ЛЮБИМОЙ

Республиканская олимпиада 1998, II-тур, задача 2
автор Даулеткулов А.

У вас имеется N бусинок белого и N черного цвета. Вам необходимо определить, сколько различных ожерелий можно собрать из 2N бусинок.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры число бусинок одного цвета N. 2. Определяет максимальное число различных ожерелий, которые можно собрать из 2N бусинок.

3. Выводит на экран максимальное число различных ожерелий, которые можно собрать из 2N бусинок.

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 1<= N <=20.

2. Необходимо использовать все бусинки.

3. Бусинки имеют сферическую форму и одинаковый размер.

К сожалению, участники республиканской олимпиады по информатике 1998 года не справились с этой задачей. Как правило, ребята использовали полный перебор и поэтому могли получить результативные ответы только для малых значений N.

Существует примета и многие школьники верят в нее: «если тебя спросили, то на следующем занятии уже не спросят, и можно не учить домашнее задание». Очевидно, именно это сыграло злую шутку с участниками 3-го этапа Республиканской олимпиады 1999 года, среди которых были и участники предыдущей Республиканской олимпиады.

ЛОСКУТНОЕ ОДЕЯЛО.

Областные олимпиады 1999, II-тур, задача 1
по мотивам С.В. Голомбо, автор Даулеткулов А.

У Вас имеется неограниченный запас красных и синих лоскутков. Вам необходимо сшить из них одеяло, имеющее форму прямоугольника размером MxN (1<=M<=20, 1<=N<=20, N № M). Лоскутки представляют собой единичные квадраты. Сколько различных одеял можно сшить?

Формат входного файла

M:N

Формат выходного файла

L - число различных одеял, которые можно сшить.

Рабочая программа должна быть представлена в виде исполняемого модуля с именем "XXXX2.exe", где XXXX - шифр участника.

Имя входного файла "2TESTx.txt", где x - номер теста.

Имя выходного файла "2ANSx.txt", где x - номер теста.

Время прохождения одного теста - 30 секунд.

Эти задачи очень похожи друг на друга, единственное отличие -это то, что в первой задаче число бусинок каждого цвета фиксировано, а во второй задаче количество лоскутков различного цвета в одеяле может изменяться. То есть в первом случае мы имеем дело с "Перестановкой с повторением", а во втором – с "Размещением с повторением" (см. часть 2). Используя известные формулы комбинаторики, можно определить общее количество ожерелий и одеял, которые можно получить при заданных условиях. Теперь нам надо учесть симметрию этих объектов, так как при определенных поворотах или переворотах мы будем получать идентичные предметы. Как это сделать? В этом нам поможет английский математик Уильям Бернсайд. В начале 20-го века он предложил следующую общую формулу для решения задач этого класса.

Пусть S – произвольное множество, содержащее конечное число элементов, а G – конечная группа симметрий этого множества, содержащая n операций симметрии g1, g2, …, gn-1, gn. Одной из этих операций должна быть тождественная (единичная) операция. Пусть через C(g) обозначено число элементов множества S, инвариантных относительно операции g . Тогда общее число N элементов S, не эквивалентных по отношению к симметриям группы G, равно

$$N=(C(g1)+C(g2)+\ldots+C(gn))/n$$

Если g1 – тождественная (единичная) операция, то C(g1)=T, то есть C(g1) равно числу элементов S.

Теперь, я думаю, что у Вас не будет больших трудностей с реализацией программы.

1.16 На пыльных дорогах космических трасс останутся наши следы

"ЗВЕЗДНЫЙ ТОРГОВЕЦ"

Республиканская олимпиада 1998, I-тур, задача 1
автор Даулеткулов А.

Пятнадцать минут до нуля. "Атлас" ждал старта. Изъеденныеударами метеоритов, некогда полированные борта космического корабля, тускло отсвечивали в ярком земном свете, заполнявшем небо Луны. Тупой нос устремлен вверх, в пустое пространство. Вакуум окружал его, а под ним простиралась мертвая пемза лунной поверхности.

Доктор Гектор Конвей спросил:

- Как думаешь, успеем ли прибыть на Аврору раньше, чем об открытии узнают эти псы из "Корпорации", Гас?

Командир "Атласа" Огастас Хенри, к которому обратился Конвей, попыхивая трубкой, ответил:

- Старое корыто не подведет. Беспокоюсь, хватит ли у нас горючего. Если что, то придется дозаправляться на других планетах, но "Корпорация" ограничила поставку горючего на независимые планеты.

- Но так мы опоздаем.

- Не волнуйся, мой Лакки - лучший штурман, он составит самый быстрый маршрут.

Звездная карта представляет собой бесконечное поле, разделенное на квадраты. Имеется N планет, пронумерованных числами от 1 до N. Для каждой планеты даны координаты квадрата, в котором она находится. На каждой планете можно взять только Mi единиц топлива. Корабль может нести неограниченное количество топлива. Вам необходимо провести корабль из квадрата, в котором находится исходная планета, в квадрат, где находится конечный пункт. Переход на соседний квадрат считается одним ходом и на него тратится 1 день. Эти данные приведены во входном файле "2TESTx.TXT" (x-номер варианта теста).

Входной ASCII файл имеет следующий формат:

N
X1:Y1:M1
.....
Xi:Yi:Mi
.....
Xn:Yn:Mn
N1:N2

где N - количество планет, Xn:Yn - координаты i -ой планеты, Mi - количество топлива на i -ой планете, N1:N2 - номера исходной и конечной планеты.

Вам необходимо за минимальное количество ходов перелететь с исходной планеты на конечную планету.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла.
2. Считывает из входного файла данные о количестве планет, их координатах и количестве топлива, которым можно заправится.
3. Определяет:

а) оптимальный маршрут движения;

б) количество дней, проведенных в пути.

4. Выводит на экран:

а) количество дней, проведенных в пути;

б) в случае невозможности достижения конечной планеты, выдает сообщение: "Нет решения".

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1.2<= N <=40, 0 <= M <=100 , 0<= Xi <=1000, 0<= Yi <=1000. Все данные - целочисленные.

2. Можно двигаться по клеткам вверх-вниз, вправо-влево или по диагонали, на одно такое передвижение тратится 1 единица горючего. Без топлива корабль двигаться не может.
3. Имя входного ASCII файла " TEST*.TXT", где * - номер теста.
4. Имя выходного ASCII файла " ANS*.TXT", где * - номер теста.

Данную задачу можно отнести к классу задач по нахождению кратчайшего пути между заданными точками на заданной сети. Небольшое отличие состоит в том, что сначала надо эту сеть построить. Поэтому решение задачи разбивается на два этапа:

1. Определить возможные пути движения из начальной точки в конечную точку.
2. Определить кратчайшее расстояние.

На первом этапе, при определении возможных путей движения, надо учитывать суммарный запас топлива и его расход. Помните, что расстояние между планетами определяется как максимальное число из разницы координат по x и по y для этих планет (учет движения по диагонали).

На втором этапе решения задачи используется алгоритм Дейкстры (см. часть 2).

1.17 Еще две задачи

ЛОВИСЬ, РЫБКА, БОЛЬШАЯ И МАЛЕНЬКАЯ

Областные олимпиады 1999, I-тур, задача 1
автор Даулеткулов А.

Рыбак собирается ловить рыбу в озере, которое представляет собой прямоугольник размером MxN (1<=M<=10000, 1<=N<=10000). Начальное положение лодки с рыбаком (0,0). Определить, с какой площади он может ловить рыбу.

ПОДЗАДАЧА 1.

В некоторых районах ему запрещено ловить рыбу.

ПОДЗАДАЧА 2.

В некоторые районы ему запрещено заходить.

Запрещенные районы представляют собой прямоугольники, стороны которого параллельны границам озера. Количество запрещенных районов K (0<=K<=100)

Формат входного файла

M:N:K - длина и ширина озера, кол-во запрещенных областей. x1:y1:L1x:L1y - координаты центра запрещенной области, размеры 1-ой области

……..

xk:yk:Lkx:Lky - координаты центра запрещенной области, размеры k-ой области

Формат выходного файла

S1 - площадь, найденная в подзадаче 1

S2 - площадь, найденная в подзадаче 2

Рабочая программа должна быть представлена в виде исполняемого модуля с именем “XXXX1.exe”, где XXXX - шифр участника.

Имя входного файла “1TESTx.txt”, где x - номер теста.

Имя выходного файла “1ANSx.txt”, где x - номер теста.

Время прохождения одного теста - 30 секунд.

Примечание. Правильность ответа определяется отдельно для подзадачи 1 и подзадачи 2.

Задачи на эту тему часто встречались на олимпиадах разного уровня, включая Всемирные олимпиады по информатике (1993,1998). Основной “хитростью” здесь является то, что запрещенные области могут накладываться друг на друга. Таким образом, площадь, разрешенная для лова рыбы, может быть не равна разнице между площадью озера и суммой площадей запрещенных областей. Более того, в подзадаче 2 могут существовать области, разрешенные для лова рыбы, но рыбак не сможет к ним добраться.

При малых значениях M и N, а также целочисленных значениях координат, задачу можно решить, используя матрицу размерностью MxN. Элемент этой матрицы равен 0, если область разрешена для лова, и равен 1, если область входит в запрещенный район.

Увеличение размерности задачи и отсутствие требования целочисленности координат (см. технические ограничения) делают невозможным применение предложенного выше подхода.

Для решения данной задачи необходимо определить контуры разрешенных для лова рыбы и запрещенных областей.

МАГИЧЕСКИЙ КВАДРАТ

Областные олимпиады 1999, II-тур, задача 2
по М. Гарднеру

Существует большое количество разновидностей “магических” квадратов. Одним из них является следующий квадрат

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Выберем наугад любое число, например 8. Вычеркнем числа, стоящие в одном столбце и в одной строке с ним. Из оставшихся чисел снова выберем произвольное число и повторим операцию вычеркивания. Будем повторять эту операцию до тех пор, пока не останется не вычеркнутых чисел. Теперь, если сложим выбранные нами числа, то для данного квадрата эта сумма будет постоянна (в наше случае 34) и не будет зависеть от порядка выбора чисел.

Определите, является ли квадратная матрица NxN (2<=N<=1000) “магическим” квадратом, если да, то найдите “магическую” сумму.

Формат входного файла

N

a11:a12: .....:a1n:a21:.....:ann

Формат выходного файла

L - “магическая” сумма (равна 0, если квадрат не “магический”

Имя входного файла “3TESTx.txt”, где x - номер теста.

Имя выходного файла “3ANSx.txt”, где x - номер теста.

Время прохождения одного теста - 30 секунд.

Решение этой простой задачи следующее:

Выясним, что представляет собой наш “магический квадрат”?

Можно заметить, что он обладает следующей особенностью: разность между элементами второй строки и первой, третьей строки и первой, четвертой строки и первой, находящихся в одном столбце, постоянна. Выпишем эти числа (для приведения к одному виду найдем также разницу между элементами первой и первой строчки) по краям нашего “магического” квадрата.

	1	2	3	4
0	1	2	3	4
4	5	6	7	8
8	9	10	11	12
12	13	14	15	16

Ясно, что сумма этих чисел (генераторов) и будет “магической” суммой. Так что решение задачи сводилось к определению наличия генераторов.

1.18 Ах вы кони, мои кони.

Эта задача была написана еще в 1996 году, но я не решался ее выставлять на официальных соревнованиях. Но при подготовке кандитатов в сборную Казахстана, для участия во Всемирной олимпиаде по информатике 1999 года, она была извлечена на свет.

Перелом (Кураж)

Сборы сборной Казахстана 1999 г., автор Даулеткулов А.

...- Как бы вы провели скачки на Пуллитцере?...

- Я просмотрел его прошлогоднюю скаковую карточку. Пуллитцер, как правило, приходил третьим, четвертым или шестым. Он скакал первым почти до финиша и сдавал лишь на последнем ферлонге... постараюсь хорошо взять старт и расположиться как можно ближе к канатам или, по крайней мере, чтобы сбоку от меня было не больше одной лошади. Я буду вести скачки не быстро и не медлено, не дальше, чем на два с половиной корпуса от лидера, но постараюсь не вырываться вперед до самого финиша. Думаю, послать лошадь надо не раньше, чем за шестьдесят, а выйти на первое место ярдов за пятнадцать до финишного столба...

...Перед последним барьером я вывел коня на край скаковой дорожки, чтобы у него был ясный обзор, и подбодрил его. Он тут же прибавил шагу и перед препятствием так рано взвился в воздух, что я испугался: не приземлится ли он прямо на барьер. Но я недооценил его силу. Он приземлился в нескольких ярдах по ту сторону и не теряя скорости, понесся к финишу....

«Фаворит»

План трассы, исходное положение лошади и линия финиша находятся во входном файле « ATEST.TXT».

Входной файл имеет следующий формат:

Блок информации начинается с данных о размерах поля, первое число - количество строк, второе - количество столбцов(Пример 1), за которым следует кодовая информация о трассе . Код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр 0,1,2 и 3 (1- граница трассы, 0 - свободное пространство, 2 - барьер, 3 - финиш), цифрой 4 обозначено исходное положение вашей лошади . После информации о трассе следует информация о скоростных характеристиках вашей лошади: первое число - максимальная скорость которую может развить лошадь (V) , второе - максимальное количество ходов (L) которое лошадь может поддерживать максимальную скорость.

Скорость лошади на n - ом ходе равна кXn - Xn-1 к + кYn - Yn-1к

где Xn-1 ,У n-1 - положение лошади после n-1 -го хода;

Xn ,Yn - положение лошади после n -го хода.

На каждом ходу вы можете увеличить (уменьшить) скорость лошади на 1 или оставить её без изменения, а также изменять направление движения лошади.

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая выполняет следующие действия:

1. Вводит с клавиатуры имя входного файла;
2. Считывает из входного файла план трассы;
3. Определяет оптимальный путь для скакуна и минимальное количество ходов необходимое для прохождения дистанции.

4. Записывает в выходной файл, с именем «ans.txt», количество ходов затраченное на прохождение дистанции , а в случае невозможности прохождения дистанции выдает сообщение «Нет решения».

ТЕХНИЧЕСКИЕ ОГРАНИЧЕНИЯ

1. 3< N <21, 5 < M <80 , 1< V <10, 0< L;.
2. Начальная скорость лошади равна 0;
3. Положение лошади после очередного хода не должно совпадать с границей трассы (или выходить за неё) или с лоложением барьера ;
4. Имя входного файла « ATEST.TXT»;
5. Имя входного файла « ANS.TXT»;
6. Рабочая программа должна быть представленна в виде исполняемого модуля с именем «Perelom.exe»/
7. Время прохождения одного теста не более 10 секунд.

Пример1

```
5.10
0 0 0 0 0 0 0 1 1 0
1 1 1 1 1 1 1 0 0 1
4 0 0 2 0 0 0 0 0 3
0 0 0 0 2 0 0 0 1 3
1 1 1 1 1 1 1 1 0 1
4,2
```

1.19 Три не очень простые задачи

Следующие три задачи были предложены мной кандидатам в сборную Казахстана летом 2004 года. Выбор первой задачи был обусловлен тем, что на международных олимпиадах обязательно присутствует задача которая не имеет точного ответа, а в ряде задач ребятам предлагается использовать внешние модули.

Поиск месторождения (Geolog.exe)
(Время тестирования 2 сек)

Геологи ищут месторождение, определяя концентрацию полезного вещества в пробах. Пробы берутся в разных точках отрезка [0, Xк]. В точке Xm с максимальной концентрацией вещества находится месторождение. Зависимость концентрации C(X) вещества от координаты точки - функция, монотонно возрастающая на [0, Xm] и убывающая на [Xm, Xк]. Исследование начинается с вызова функции start, не имеющей аргументов. Результатом работы функции является число Xк. При определении концентрации Вы можете использовать функцию look(x), которая возвращает значение концентрации в точке x. Вам необходимо найти координаты месторождения, в котором Вы обнаружили максимальную концентрацию, вызывая функцию look(x) минимальное число раз. Когда Вы, наконец, определите эти координаты, Вы должны вызвать процедуру finish(*x*), где *x* – координата месторождения.

Разрешается при определении места очередной пробы использовать результаты предыдущих проб.

Ограничения

- 100 <= Xк<=1000000

ВЫХОД

Выходной файл не должен создаваться. Результат решения задачи сообщается вызовом процедуры (функции) finish(*x*).

В исходный файл вашей программы включите следующую строку:

uses anal;

Этот tpu будет содержать следующее:

function start;

function look (x:integer):integer;

procedure finish (x:integer);

Чтобы получить полный балл *A* за тест, нужно, чтобы *x* - количество вызовов функции look, было меньше или равно числу *M*, установленному тестирующей программой. Заметим, что *M* выбирается таким, чтобы быть больше (>), чем минимум. В частности, *M* не зависит от того, какое выбрано направление осмотра. Вы можете получить часть баллов, если число обращений к функции look больше (>) *M,* но меньше (<) удвоенного *M*. Баллы, которые Вы получите, вычисляются путем округления первого десятичного знака того значения, которое определяется по следующей формуле:

A if *x* J *M*

A (2*M* – *x*) /*M* if *M* < *x* < 2*M*

0 if *x* i 2*M*

Если программа работает с нарушением правил, то она оценивается в 0 баллов.

Терминалы (Term.exe)

(Время тестирования 3 сек)

В разных помещениях здания надо установить N (1<=N<=1000)терминалов. Заданы их координаты X[i], Y[i], Z[i], i=1,...,N (0<=x,y,z,<=10000). Определить, в какой точке здания надо установить Сервер, чтобы суммарная длина кабелей связи, идущих от каждого терминала к серверу, была минимальной. Кабели могут проходить только вертикально или горизонтально параллельно стенам. К каждому терминалу должен идти отдельный кабель.

Входной файл (term.in) состоит из N строк. В каждой, через пробел, заданы координаты терминалов по X,Y,Z.

В выходной файл (term.out) выводится, через пробел, четыре целых числа: длина кабелей и координаты точки установки сервера.

Пример

Term.in	Term.out
0 0 0	20 0 0 0
10 0 0	
0 10 0	

Ходом коня (chess.exe) 50 баллов

(Время тестирования 3 сек.)

На шахматном поле находится белый конь. На шахматной доске расположено несколько черных пешек (1<=N<=8). Написать программу, которая определяет за какое минимальное количество ходов белый конь может съесть все черные пешки (пешки неподвижны). Ходы конем осуществляются по правилам шахмат. В случае невозможности съесть все пешки программа выдает сообщение: ‘NO’.

Входной файла (chess.in) содержит две строки. В первой строке заданы координаты белого коня, во второй строке через пробел координаты черных пешек (Буквенные обозначения полей прописные латинские буквы).

Выходной файл должен содержать одно число - минимальное количество ходов.

Пример:

chess.in	chess.out
A1 D3 E3	6
A1 C4 D3	NO

Для решения первой задачи целесообразно было применить метод золотого сечения, а для уменьшения запросов завести дополнительный массив, в котором хранятся данные для уже проанализированных точек. Если с первой половиной задачи практически справились два участника, то они не учли второе положение.

Задача терминалы имеет очень простое решение, так как она разбивается на решение трех идентичных подзадач. То есть расположение сервера и длину кабеля можно находить отдельно для каждой координаты. Для этого координаты (X) терминалов сортируются в порядке не убывания, а затем определяется точка, для которой суммарная длина кабеля по данной координате минимальна. Такие действия проводятся и для двух оставшихся координат. В конце надо просто сложить длину кабеля по трем координатам и вывести значение координат сервера.

Основная сложность в третьей задачи было то, что конь не может съесть некоторые пешки, то есть, пешки которые защищены другими пешками. Размерность задачи не очень велика, поэтому для решения можно воспользоваться обычным перебором с возвратом, а можно использовать принцип решения задачи «приключение в королевстве Ном».

1.20 Сортировка без сортировки

Эта очень простая задача посавила в тупик участников республиканской олимпиады 1999 года.

Рубины Басры

Республиканская олимпиада 1999 года, I –тур, автор А.Даулеткулов

- В моей сокровищнице очень много разных рубинов. Но это самый большой. Он весит 1000 каратов.

- Действительно прекрасный рубин.

- Если тебе дорога жизнь, ты должен к утру отсортировать все рубины по весу.

Помогите Алладину отсортировать рубины. Количество рубинов (N) не менее 3 и не более1 000 000. Веса рубинов находятся в файле 'rubin.in'. В выходной файл ‘rubin.out’ необходимо вывести веса рубинов отсортированные в порядке не убывания массы. Время работы программы 1 секунда.

Пример

rubin.in	rubin.out
100 2 2 5 15	2 2 5 15 100

Участники олимпиады при решении данной задачи использовали стандартные методы сортировки (см. часть 2). Поэтому при больших значениях N решение не укладывалось во временные рамки. Однако, если внимательно прочитать условие задачи, то выяснится, что она имеет очень простое решение. Это связано с наличием ограничения на вес рубинов. Поэтому вместо стандартых методов сортировки можно (и нужно) использовать метод «сортировки подсчетом».

1.21 Еще шестнадцать очень простых задач

Ниже приведены задачи которые были использованы при проведении районных олимпиад в 2001 году в городе Алматы. Время тестирования для каждой задачи 2 секунды. Считаю, что решение таких задач является базовым для учащихся желающих участвовать в олимпиадах по информатике.

Задача 1

Дана последовательность из N (2<=N<=100000) различных чисел. Найти сумму чисел этой последовательности, расположенных между максимальным и минимальным числами (включая оба эти числа). Числа в последовательности лежат в интервале от 1 до 1000000 включительно.

«pos.in» «pos.out»
5 108
1
7
100
3

Задача 2.

Задан ряд последовательных натуральных чисел от n до m (n <m<1000000), из которого удаляют сначала все числа, стоящие на нечетных местах, затем из оставшегося ряда удаляют все числа стоящие на нечетных местах. Эти действия повторяют до тех пор пока не останеться одно число, Определить это число.

Данные n и m, вводятся с клавиатуры, формат ввода :
Input n- 5
Input m- 10
Результат выводится на экран, формат вывода: 8

Задача 3

Даны натуральные числа A и B , обозначающие стороны некоторого прямоугольника. Найти наименьшее количество квадратов, необходимых для его заполнения.

Данные вводятся с клавиатуры, результат выводится на экран.

Пример:

Входные данные	Выходные данные
a = 5	k = 5
b = 6	

Задача 4

Заполнить массив A(n) натуральными числами от 1 до n (3<=n<=10000) так, чтобы любые взятые подряд три его элемента не образовывали монотонно возрастающую или убывающую последовательность. Число n вводится с клавиатуры, результат выводится в файл ‘posled.out’.

Пример:

Входные данные ‘posled.out’
N =3 1 3 2

Задача 5

Дана последовательность из N (2<=N<=100000) различных чисел. Выписать в возрастающем порядке все целые числа находящиеся в интервале от минимального до максимального значения элементов этой последовательности, которые не находятся в данной последоватнльности.

«pos.in» «pos.out»
5 2
1
7
10
3
4
6
8
9

Задача 6

Матрица размерностью NxM (N-количество строк, M -количество столбцов, 2<=N и M<=100000) заполнена числами от 1 до N*M, следующим образом: сначала заполняется первая строка слева на право от 1 до значения M(повозрастанию),затем вторая строка слева на право от M+1 до 2M, и так далее. По введеным параметрам матрицы и числу K,определить в какой строке и в каком столбце стоит данное число.

Исходные данные вводятся с клавиатуры (формат ввода):
Input N- 4
Input M-4
Input K-15
Результат выводится на экран (формат вывода):
Stroka-4
Stolbec-3

Задача 7

Дана последовательность из N (2<=N<=100000) чисел. Найти числа встречающиеся в данной последовательности максимальное количество раз.

«pos.in» «pos.out»
5 1
1
7
1
30
7
34
7

Задача 8

На клетчатом листе бумаги размером NxM (2<=N и M<=200) клеток нарисовано несколько прямоугольников. Каждый прямоугольник состоит из целых клеток, различные прямоугольники не накладываются друг на друга и не соприкасаются. Лист представляет собой матрицу размером NxM,

элемент матрицы равен 1 если клетка пренадлежит прямоугольнику и 0 - если не принадлежит. Определить сколько прямоугольников нарисовано на листке.

«kvadraty.in»	« kvadraty.out»
0001110	3
1001110	
0000000	
1100000	

Задача 9

Даны три числа a, b, c . Необходимо определить, существует ли треугольник с такими длинами сторон. Данные вводятся с клавиатуры. Результат выводится на экран (да-yes, нет -no) .

Исходные данные вводятся с клавиатуры (формат ввода):

Input a- 4

Input b-14

Input c-15

Результат выводится на экран (формат вывода):

yes

Задача 10

Дана последовательность из N (2<=N<=100000) чисел. Найти наибольшее количество одинаковых идущих в нем подряд элементов (если их несколько, то одно из них).

«pos.in»	«pos.out»
5	3 1
1	
1	
1	
30	
7	
7	

Задача 11

Матрица размерностью NxM (N-количество строк, M -количество столбцов, 2<=N и M<=100000) заполнена числами от 1 до N*M, следующим образом: сначала заполняется первая строка слева на право от 1 до значения M(повозрастанию),затем вторая строка слева на право от M+1 до 2M, и так далее. По введеным параметрам матрицы и числу K,определить номера всех клеток, имеющих с ней общую сторону.

Исходные данные вводятся с клавиатуры (формат ввода):

Input N- 4

Input M-4

Input K-15

Результат выводится на экран (формат вывода):

16 14 11

Задача 12

В двенадцати летнем цикле года носят названия животных: мышь, корова, тигл, заяц, улитка, змея, лошадь, овца, обезьяна, курица, собака, свинья. Известно, что 1984 год был годом мыши. Написатиь программу, которая вводит номер года нашей эры с клавиатуры и выводит его название по 12-ти летнему циклу.

Исходные данные вводятся с клавиатуры (формат ввода):
Input Year - 1001
Результат выводится на экран (формат вывода):
Name - корова

Задача 13

Дана последовательность из N (2<=N<=100000) чисел. Найти количество различных чисел в этой последовательности, определить эти числа. Вывести их в порядке уменьшения количества встречаний их в данной последовательности.

«pos.in» «pos.out»
5 1 7 5 30
1
1
1
30
7
7

Задача 14

По окончании массового забега все его участники уложили свои нагрудные номера в один ряд в том порядке, в каком они пересекли финишную черту, оразовав в результате К-значное число. Требуется определить М (М<1000) количество участников забега по известному К. Вслучае невозможности определить количество участников вывести сообщение «net reshenija».

Исходные данные вводятся с клавиатуры (формат ввода):
Input K-15
Результат выводится на экран (формат вывода):
12

Задача 15

С клавиатуры вводится строка символов (длина не привышает 100), Вывести на экран эту строку преобразовав ее по следующему правилу:

- Символы на четных позициях остаются на своих местах;
- Символы на нечетных позициях меняют порядок вывода.

Входные данные Выходные данные ‘posled.out’ (_ -знак пробела)
1234 3214

Задача 16

На плоскости нарисован многоугольник. Определить его площадь. Вершины многоугольника заданы своими координатами (x1 y1),(x2 y2),(x3 y3),…,(xNyN), координаты вершин заданы обходом многоугольника по часовой стрелке. Исходные данные заданы в файле «mnogoyg.in». Результат вывести на экран .

«mnogoyg.in» Результат
0 0 2
1 2
2 0

1.22 Примеры программ.

- Программа! - желчно усмехнувшись, произнес Хунта. - Я не видел твоей программы, Теодор, но я уверен, что она гениальна по сравнению с этим... - он с отвращением подал двумя пальцами Федору Симеоновичу листок со своей задачей. - Полюбуйся, вот образец убожества и ничтожества.

А. Стругацкий, Б. Стругацкий "Понедельник начинается в субботу"

В заключение первой части книги приведены тексты программ задач, рассматриваемых в книге. По просьбе автора, текст программ представлен членами сборных команд Казахстана по информатике 1997 - 2004 годов.

Автор не считает их единственно правильными, более того, надеется, что пытливый читатель сможет улучшить эти программы.

1. Тесей и Минотавр

```
program tesei;
type st=string[2];
var
 hodm,hod:array [1..800,1..2] of byte;
 a,b,c:array [1..20,1..40] of integer;
 d:array [1..20,1..40] of st;
 ntst:string;
 n,m:byte;
 vsi,vsj,ti,tj,mi,mj,li,lj:byte;
 vyhi,vyhj,vshod,mhod:integer;
 vyhod,vstrecha:boolean;
 minhod:integer;

procedure init;
var
 f:text;
 s,s1,s2:string;
 i,j,q,code:integer;
begin
 write("Введите номер теста:");
 readln(ntst);
 assign(f,'test'+ntst+'.txt');
 reset(f);
 readln(f,s);
 q:=pos(",",s);
 s1:=copy(s,1,q-1);
 s2:=copy(s,q+1,length(s)-q);
 val(s1,n,code);
 val(s2,m,code);
 for i:=1 to n do begin
  readln(f,s);
  for j:=1 to m do begin
   val(copy(s,j,1),a[i,j],code);
  end;
```

```
 end;
 readln(f,s);
 q:=pos(“,”,s);
 s1:=copy(s,1,q-1);
 s2:=copy(s,q+1,length(s)-q);
 val(s1,ti,code);
 val(s2,tj,code);
 readln(f,s);
 q:=pos(“,”,s);
 s1:=copy(s,1,q-1);
 s2:=copy(s,q+1,length(s)-q);
 val(s1,mi,code);
 val(s2,mj,code);
 readln(f,s);
 q:=pos(“,”,s);
 s1:=copy(s,1,q-1);
 s2:=copy(s,q+1,length(s)-q);
 val(s1,li,code);
 val(s2,lj,code);
 i:=1;
 while not eof(f) do begin
  readln(f,s);
  q:=pos(“,”,s);
  s1:=copy(s,1,q-1);
  s2:=copy(s,q+1,length(s)-q);
  val(s1,hodm[i,1],code);
  val(s2,hodm[i,2],code);
  inc(i);
 end;
 dec(i);
 mhod:=i;
 close(f);
end;
function change:boolean;
var
 i,j:byte;
 q:boolean;
begin
 q:=false;
for i:=1 to n do begin
  for j:=1 to m do begin
   if b[i,j]<>a[i,j] then begin
    q:=true;
    j:=m;
    i:=n;
   end;
  end;
 end;
 change:=q;
```

```
end;
procedure change_a_to_d(x:byte);
var
 s:string;
 code:integer;
 i,j:byte;
begin
 for i:=1 to n do begin
  for j:=1 to m do begin
   if a[i,j]=1 then d[i,j]:='*  " else if a[i,j]=0 then d[i,j]:="   " else begin
    str(a[i,j]-2,s);
    d[i,j]:=s;
   end;
  end;
 end;
 if x=2 then d[ti,tj]:='T';
 if x=2 then d[mi,mj]:='M';
 if x=2 then d[li,lj]:='Л';
 d[vsi,vsj]:='X';
 for i:=1 to n do begin
  for j:=1 to m do begin
   if length(d[i,j])<2 then d[i,j]:=d[i,j]+' ";
  end;
 end;
end;
procedure done(x:byte);
var
 i,j:byte;
begin
 change_a_to_d(x);
 for i:=1 to n do begin
  for j:=1 to m do begin
   write(d[i,j]);
  end;
  writeln;
 end;
end;
procedure process;
var
 i1,j1,k,i,j:integer;
begin
 c:=a;
 a[ti,tj]:=2;
 k:=2;
 while change do begin
  b:=a;
  for i:=1 to n do begin
   for j:=1 to m do begin
    if a[i,j]=k then begin
```

```
     if ((a[i+1,j]=0) and (i<n)) then a[i+1,j]:=k+1;
     if ((a[i-1,j]=0) and (i>1)) then a[i-1,j]:=k+1;
     if ((a[i,j+1]=0) and (j<m)) then a[i,j+1]:=k+1;
     if ((a[i,j-1]=0) and (j>1)) then a[i,j-1]:=k+1;
    end;
   end;
  end;
  inc(k);
 end;
 vstrecha:=false;
 for i:=1 to mhod do begin
  if ((a[hodm[i,1],hodm[i,2]]-2)=i) then begin
   vsi:=hodm[i,1];
   vsj:=hodm[i,2];
   vstrecha:=true;
   vshod:=i;
   i:=mhod;
  end;
 end;
 a:=c;
 if not vstrecha then begin
  writeln("Встреча невозможна!!!");
halt(1);
 end;
 for i:=1 to mhod do begin
  a[hodm[i,1],hodm[i,2]]:=i+2;
 end;
 done(2);
 b:=c;a:=c;
 a[vsi,vsj]:=2;
 k:=2;
 vyhod:=false;
 while change do begin
  b:=a;
  for i:=1 to n do begin
   for j:=1 to m do begin
    if a[i,j]=k then begin
     if ((a[i+1,j]=0) and (i<n)) then a[i+1,j]:=k+1;
     if ((a[i-1,j]=0) and (i>1)) then a[i-1,j]:=k+1;
     if ((a[i,j+1]=0) and (j<m)) then a[i,j+1]:=k+1;
     if ((a[i,j-1]=0) and (j>1)) then a[i,j-1]:=k+1;
     if ((i+1=n) and (a[i+1,j]<>0) and (a[i+1,j]<>1)) then begin
      vyhi:=n;vyhj:=j;
      j:=m;
      i:=n;
      b:=a;
      vyhod:=true;
     end;
     if ((i-1=1) and (a[i-1,j]<>0) and (a[i-1,j]<>1)) then begin
```

```
     vyhi:=1;vyhj:=j;
     j:=m;
     i:=n;
     b:=a;
     vyhod:=true;
    end;
    if ((j+1=m) and (a[i,j+1]<>0) and (a[i,j+1]<>1)) then begin
     vyhi:=i;vyhj:=m;
     j:=m;
     i:=n;
     b:=a;
     vyhod:=true;
    end;
    if ((j-1=1) and (a[i,j-1]<>0) and (a[i,j-1]<>1)) then begin
     vyhi:=i;vyhj:=1;
     j:=m;
     i:=n;
     b:=a;
     vyhod:=true;
    end;
   end;
  end;
 end;
 inc(k);
end;
writeln;
if not vyhod then begin
 writeln("Выйти невозможно!!!");
 halt(1);
end;
minhod:=k;
b:=a;
a:=c;
a[vyhi,vyhj]:=minhod;
i1:=vyhi;j1:=vyhj;
dec(k);
repeat
for i:=1 to n do begin
 for j:=1 to m do begin
  if b[i,j]=k then begin
   if (((i=i1+1) and (j1=j)) or ((i=i1-1) and (j1=j))
    or ((i=i1) and (j=j1+1)) or ((i=i1) and (j=j1-1))) then begin
     i1:=i;
     j1:=j;
     a[i1,j1]:=k;
     dec(k);
   end;
  end;
 end; end;
```

```
 until k<=2;
end;

BEGIN
 init;
 process;
 done(1);
END.
```

2. Лорен-Дитрих

```
Program Loren;
 Const
  MaxN=1000;

 Type
  TypePole=Array[0..MaxN] of Word;
  TypeBPole=Array[0..MaxN] of Boolean;
 Var
  S,C,M : Integer;
  Pole  : TypePole;
  I     : Word;
  Bp1,Bp2 : TypeBPole;
 Procedure Load;
  Var
   I : Word;
  Begin
   Write("Enter S:");Readln(S);
   Write("Enter M:");Readln(M);
   Write("Enter C:");Readln(C);
   For i:=0 to S do Pole[i]:=MaxInt;
   For i:=0 to S do Bp1[i]:=False;
   Bp2:=Bp1;
  End;
 Procedure Test(S1 : Word; K : Word);
  Var
   I : Byte;
  Begin
   For I:=1 to M div C do
    if (S>=S1+I)  and ((M-2*C*I)>0) and (K-M+C*I>=0) and
       (Pole[S-S1-I]>((K-M+C*I)*M) div (M-2*C*I)+M) then
     Begin
      if S1=M div C then Pole[S-S1]:=(K-M+C*I);
      Pole[S-S1-I]:=((K-M+C*I)*M) div (M-2*C*I)+M;
      Bp1[S-S1-i]:=True;
      if S>S1+I then Test(S1+I,Pole[S-S1-I]) else Bp2:=Bp1;
      Bp1[S-S1-i]:=False;
     End;
  End;
```

```
 Begin
  Load;
  Bp1[S-M div C]:=True;
  Test(M div C,M);
Write(“Otvet :”);
  For i:=1 to S do
   if Bp2[i] then Write(“SB= “,i,” NumB=’,Pole[i],’ “);
  Writeln;
 End.
```

3. Узник замка Иф

```
Program If;
Uses Crt,Dos,Blist,MyProcb;
Const
    MaxC=10;
    MaxK=10;
    MaxN=20;
    MaxM=40;
    MaxTurn=1000;
Type
   TypeKletka=record
       Con  : string[1];
       Step : integer;
       Kirka: integer;
   End;
   TypePole=array[1..MaxM,1..MaxN] of TypeKletka;
Var
  Turn      : Type_Block_List;
  Info      : TypeXY;
  Pole      : TypePole;
N,M,C,K   : integer;
  ManX,ManY : integer;
  Top       : integer;
  OK        : boolean;
Procedure CleanPole(Var Pole : TypePole);
Var
  i,j : integer;
Begin
    for i:=1 to MaxM do
    Begin
       for j:=1 to MaxN do
       Begin
          Pole[i,j].Con:=’’;
          Pole[i,j].Step:=999;
          Pole[i,j].Kirka:=0;
       End;
    End;
End;
```

```
Procedure OutS(x,y : integer);
Begin
   gotoxy(x*3-2,y+N+1);
   write(Pole[x,y].Step:2);
End;
Procedure OutK(x,y : integer);
Begin
   gotoxy(x*3-2+M*3+1,y);
   write(Pole[x,y].Kirka:2);
End;
Procedure OutC(x,y : integer);
Begin
   gotoxy(x,y);
   if Pole[x,y].Con='1'
   then  write({Pole[x,y].Con:2}'*')
   else  Write(" ");
End;

Procedure OutPole;
Var
  i,j : integer;
Begin
   for i:=1 to M do
   Begin
      for j:=1 to N do
      Begin
        OutC(i,j);
      End;
   End;
End;
Procedure Load(Var Pole : TypePole);
Var
  F       : text;
  FileName : string[12];
  Bufer    : string;
  Code     : integer;
  i,j      : integer;
Begin
   write("Введите имя файла : ");
   readln(FileName);
   Assign(F,FileName);
   {$I-}
   Reset(F);
   if IOResult<>0
   then
     Begin
        write("IO ERROR !");
        Halt;
     End;
```

```
   Readln(F,Bufer);
   Val(Copy(Bufer,1,Pos(“,”,Bufer)-1),N,Code);
   Val(Copy(Bufer,Pos(“,”,Bufer)+1,Length(Bufer)),M,Code);
   for j:=1 to N do
   Begin
      Readln(F,Bufer);
      for i:=1 to M do
      Begin
         Pole[i,j].Con:=Bufer[i];
      End;
   End;
   Readln(F,Bufer);
   Val(Copy(Bufer,1,Pos(“,”,Bufer)-1),ManY,Code);
   Delete(Bufer,1,Pos(“,”,Bufer));
   Val(Copy(Bufer,1,Pos(“,”,Bufer)-1),ManX,Code);
Delete(Bufer,1,Pos(“,”,Bufer));
   Val(Copy(Bufer,1,Pos(“,”,Bufer)-1),K,Code);
   Delete(Bufer,1,Pos(“,”,Bufer));
   Val(Copy(Bufer,1,length(Bufer)),C,Code);
   Delete(Bufer,1,Pos(“,”,Bufer));
End;
Procedure  Test(Var Pole : TypePole; x,y  : integer);
Label
   M1,M2,M3,M4;
Var
 Txy:TypeTxy;
 It:Integer;
Begin
   Info.x:=X;
   Info.Y:=y;
   TestLadya(Info,Txy,1,1,N,M);
   for it:=1 to 4 do
    if (Txy[it].X<>0) then
      Begin
         if (Pole[Txy[it].X,Txy[it].Y].Con=’-’) and (Pole[x,y].Step mod 2 = 0) then goto M1;
         if (Pole[TXy[it].X,TXy[it].Y].Con=’+’) and (Pole[x,y].Step mod 2 = 1) then goto M1;
         if (Pole[x,y].Step+1<Pole[TXy[it].X,Txy[it].y].Step)
         then
            Begin
               if (Pole[TXy[it].x,Txy[it].y].Con=’1') and (Pole[x,y].Kirka<=0) then goto M1;
               if Pole[Txy[it].X,Txy[it].y].Con=’1'
               then
                  Begin
     Pole[TXy[it].X,TXy[it].Y].Kirka:=Pole[x,y].Kirka-1;
     Pole[Txy[it].X,Txy[it].y].Step:=Pole[x,y].Step+1+K;
                   PushRight(Turn,@Txy[it],OK);
                  End
               else
                  Begin
```

```
                    Pole[TXy[it].X,Txy[it].y].Kirka:=Pole[x,y].Kirka;
                    Pole[Txy[it].x,Txy[it].y].Step:=Pole[x,y].Step+1;
                    PushRight(Turn,@txy[it],OK);
                  End;
            End
          else
            Begin
              if (Pole[TXy[it].X,TXy[it].y].Con='1') and (Pole[x,y].Kirka<=0) then goto M1;
              if (Pole[Txy[it].x,TXy[it].y].Con='1') and (Pole[x,y].Kirka+1<Pole[TXy[it].X,TXy[it].y].Kirka)
              then
                Begin
      Pole[Txy[it].x,Txy[it].y].Kirka:=Pole[x,y].Kirka-1;
       Pole[Txy[it].x,Txy[it].y].Step:=Pole[x,y].Step+1+K;
                    PushRight(Turn,@txy[it],OK);
                End;
              if (Pole[TXy[it].X,TXy[it].y].Con='0') and (Pole[x,y].Kirka>Pole[txy[it].x,Txy[it].y].Kirka)
              then
                Begin
                  Pole[Txy[it].X,TXy[it].Y].Kirka:=Pole[x,y].Kirka;
                  Pole[TXy[it].X,Txy[it].y].Step:=Pole[x,y].Step+1;
                  PushRight(Turn,@txy[it],OK);
                End
            End;
        M1:
        End;
End;
Procedure BackTrace(Pole : TypePole);
Var
  Min   : integer;
  Info  : TypeXY;
  i,x,y : integer;
Begin
    Min:=Pole[1,1].Step;
    for i:=1 to M do
    Begin
       if Pole[i,1].Step<Min
       then
         Begin
           x:=i; y:=1;
           Min:=Pole[i,1].Step;
         End;
if Pole[i,N].Step<Min
       then
         Begin
           x:=i; y:=N;
           Min:=Pole[i,N].Step;
         End;
    End;
    for i:=1 to N do
```

```
  Begin
     if Pole[1,i].Step<Min
     then
       Begin
          x:=1; y:=i;
          Min:=Pole[1,i].Step;
       End;
     if Pole[M,i].Step<Min
     then
       Begin
          x:=M; y:=i;
          Min:=Pole[M,i].Step;
       End;
  End;
  if Min=Pole[1,1].Step
  then
    Begin
       gotoxy(M+1,N+1);
       write("Нет решения");
       Halt;
    End;
  gotoxy(x,y);
  write(Pole[x,y].Step-1);
  Repeat
      if x<M
      then
        Begin
           if (Pole[x+1,y].Step<Min) and (Pole[x+1,y].Kirka>=Pole[x,y].Kirka)
           then
             Begin
                Info.x:=x+1;
                Info.y:=y;
                Min:=Pole[x+1,y].Step;
             End;
        End;
      if x>1
      then
        Begin
           if (Pole[x-1,y].Step<Min) and
(Pole[x-1,y].Kirka>=Pole[x,y].Kirka)
           then
             Begin
                Info.x:=x-1;
                Info.y:=y;
                Min:=Pole[x-1,y].Step;
             End;
        End;
      if y<N
      then
```

```
          Begin
            if (Pole[x,y+1].Step<Min) and (Pole[x,y+1].Kirka>=Pole[x,y].Kirka)
            then
              Begin
                Info.x:=x;
                Info.y:=y+1;
                Min:=Pole[x,y+1].Step;
              End;
          End;
        if y>1
        then
          Begin
            if (Pole[x,y-1].Step<Min) and
(Pole[x,y-1].Kirka>=Pole[x,y].Kirka)
            then
              Begin
                Info.x:=x;
                Info.y:=y-1;
                Min:=Pole[x,y-1].Step;
              End;
          End;
        gotoxy(Info.x,Info.y);
        write(Pole[Info.x,Info.y].Step-1);
        x:=Info.x;
y:=Info.y;
    Until Min=1;
    gotoxy(x,y);
    write("X");
End;
(*————————PROGRAM————————*)
(*———————————————————————*)
Begin
    ClrScr;
    OK:=true;
    Top:=0;
    OpenList(Turn,SizeOf(TypeXY));
    CleanPole(Pole);
    Load(Pole);
    Info.x:=ManX;
    Info.y:=ManY;
    Pole[ManX,ManY].Kirka:=K;
    Pole[ManX,ManY].Step:=1;
    PushRight(Turn,@Info,OK);
    ClrScr;
    OutPole;
    Repeat
       PopLeft(Turn,@Info,OK);
       Test(Pole,Info.x,Info.y);
    Until OK=False;
```

```
    OutPole;
    BackTrace(Pole);
End.
```

4. За золотом на ручей Индианки

```
Program Ruchei_Za_Zolotom;
 Uses Crt,Dos;
  Const
   MaxN=10;
  Type
   TypeXY=Record
             X,Y : Integer;
          End;
   TypeNXY=Array[1..MaxN] of TypeXY;
   TypeC=Array[1..MaxN,1..MaxN] of Byte;
   TypeAllS=Array[1..MaxN] of Real;
   TypeG=Array[1..MaxN] of Byte;
  Var
   NXY: TypeNXY;
   Name : String;
   AllS : TypeAllS;
   C    : TypeC;
   N    : Byte;
   G    : TypeG;
   NumS : Byte;
  Procedure Load;
   Var
    i,j : Byte;
    i1,i2: Integer;
    IOs  : Integer;
    F    : Text;
    Buf : String[40];
    St  : String[40];
   Begin
    ClrScr;
    Write("Enter FileName: ");Readln(Name);
    Assign(F,Name);
    Reset(F);
    Readln(F,N);
    For I:=1 to N do
     Begin
      Readln(F,Buf);
      St:=Copy(Buf,1,Pos(",",Buf)-1);
      Delete(Buf,1,Pos(",",Buf));
      Val(St,I1,IOs);
      St:=Buf;
      Val(St,I2,IOs);
      NXY[i].X:=I1;
```

```
      NXY[i].Y:=I2;
     End;
    While Not Eof(F) do
     Begin
      Readln(F,Buf);
      St:=Copy(Buf,1,Pos("-",Buf)-1);
      Delete(Buf,1,Pos("-",Buf));
      Val(St,I1,IOs);
St:=Buf;
      Val(St,I2,IOs);
      C[I1,I2]:=1;
      C[I2,I1]:=1;
     End;
    Close(F);
   End;
 Procedure TestOne;
  Var
   Flag : Boolean;
   S: Byte;
   I,J : Byte;
  Begin
   Repeat
    Flag:=True;
    For i:=1 to N do
     Begin
      S:=0;
      For J:=1 to N do S:=S+C[I,J];
      if S=1 then
        Begin
         For J:=1 to N do
          Begin
           C[I,j]:=0;
           C[J,I]:=0;
          End;
         Flag:=False;
        End;
     End;
   Until Flag;
  End;
 Procedure TestAll;
  Var
    IS: Byte;
    S,S1 : Integer;
    I,J,J1 : Byte;
    Flag : Boolean;
  Procedure Test( I : Byte);
   Var
    J : Byte;
    J1 : Byte;
```

```
  Begin
  if  I<>IS then
   Begin
    J1:=1;
    While  (C[I,J1]=0) and (J1<N) do Inc(J1);
    if C[I,J1]=0 then Inc(J1);
    For J:=J1 to N do
     if C[I,J]=1 then
       if (NXY[J].X-NXY[I].X)*(NXY[J1].Y-NXY[I].Y)<
    (NXY[J].Y-NXY[I].Y)*(NXY[J1].X-NXY[I].Y) then J1:=J;
    S:=S+(NXY[I].X-NXY[J1].X)*(NXY[J1].Y+NXY[I].Y);
    if J1<N+1 then
     Begin
      C[I,J1]:=0;
      C[J1,I]:=0;
      Test(J1);
      C[J1,I]:=1;
      C[I,J1]:=1;
     End;
   End
   Else S1:=S;
  End;
  Begin
   For I:=1 to N do G[I]:=I;
   For I:=1 to N-1 do
    For J:=I+1 to N do
     if NXY[G[I]].Y>NXY[G[J]].Y then
       Begin
        IS:=G[I];
        G[I]:=G[J];
        G[J]:=IS;
       End
      Else
     if (NXY[G[I]].Y=NXY[G[J]].Y) and (NXY[G[I]].X<NXY[G[J]].X) then
       Begin
        IS:=G[I];
        G[I]:=G[J];
        G[J]:=IS;
       End;
   For I:=1 to N do
Begin
     J1:=0;IS:=G[I];
     For J:=1 to N do
      If C[IS,J]=1 then
       Begin
        J1:=J;
        J:=N;
       End;
     if J1>0 then
```

```
      Begin
       Flag:=False;
       For J:=J1+1 to N do
        if (C[IS,J]=1) and ((NXY[J].Y-NXY[IS].Y)*(NXY[J1].X-NXY[IS].X)< (NXY[J1].Y-
NXY[IS].Y)*(NXY[J].X-NXY[IS].X)) then
                          Begin
                           Flag:=True;
                           J1:=J;
                          End;
      if Flag then
        Begin
         IS:=G[I];
         C[IS,J1]:=0;
         C[J1,IS]:=0;
         S1:=0;
         S:=(NXY[IS].X-NXY[J1].X)*(NXY[J1].Y+NXY[IS].Y);
         Test(J1);
         TestOne;
         if S1>0 then
          Begin
           Inc(NumS);
           AllS[NumS]:=S1 / 2;
          End;
        End Else
         Begin
          C[IS,J1]:=0;
          C[J1,IS]:=0;
          TestOne;
         End;
       End;
     End;
  End;
 Procedure Save;
  Var
   F : Text;
   Pa : PathStr;
   Di : DirStr;
   Na : NameStr;
   Ex : ExtStr;
   i,J: Integer;
   B  : Real;
  Begin
   FSplit(Name,Di,Na,Ex);
   Delete(Na,1,4);
   Name:=Di+'ANS'+NA+Ex;
   Assign(F,Name);
   Rewrite(F);
   Writeln(F,NumS);
   For I:=1 to NumS-1 do
```

```
    For J:=i+1 to NumS do
     if AllS[I]<AllS[J] then
      Begin
       B:=AllS[I];
       AllS[I]:=AllS[J];
       AllS[J]:=B;
      End;
   For i:=1 to NumS do Writeln(F,AllS[I]);
   Close(F);
  End;
 Begin
  Load;
  TestOne;
  TestAll;
  Save;
 End.
```

5. Острова

```
Program OSTROV;
 Uses Crt,Dos;
 Const
   MaxM=132;
if length(q)<length(w) then begin
  c:=q;
  q:=w;
  w:=c;
 end;
 max:=0;t:=0;z:=0;
 for k:=1 to length(q) do begin
  if z>max then begin
   max:=z;
   t:=r;
  end;
  z:=0;
  for l:=k to length(w)+k-1 do begin
   if (q[l]=w[l-k+1]) and (q[l]<>' ") then begin
    inc(z);
    r:=k;
   end;
   if (q[l]<>w[l-k+1]) and (q[l]<>' ") and (w[l-k+1]<>" ") then begin
    z:=0;r:=0;
    l:=length(w)+k-1;
    qwe:=true;
   end;
  end;
 end;
```

```
 if max<>0 then begin
  k:=t;
  for l:=k to length(w)+k-1 do begin
   if w[l-k+1]<>' ' then q[l]:=w[l-k+1];
  end;
 end;
 if ((max=0) and (qwe)) then qwe:=true else qwe:=false;
 add:=q;
end;
function change:boolean;
var
 j:byte;
 e:boolean;
begin
 e:=false;
 for j:=1 to imax do begin
  if a[j]<>doc[j] then e:=true;
 end;
 change:=e;
end;
begin
 write('Введите номер теста:');
 readln(number);
 assign(f,'test'+number+'.txt');
 a:=doc;
 a[1]:=' ';
 doc[1]:='';
 imax:=1;
 while change do begin
  a:=doc;
  reset(f);
  for i:=1 to 100 do doc[i]:=' ';
  while not eof(f) do begin
   readln(f,s);
   if s[1]='Z' then begin
    if (length(s)=2) and (s[2] in ['0'..'9']) then begin
     readln(f,s);
     i:=1;
    end else
    if (length(s)=3) and (s[2] in ['0'..'9']) and (s[3] in ['0'..'9']) then begin
     readln(f,s);
     i:=1;
    end;
   end;
   doc[i]:=add(doc[i],s);
   j:=1;
   if ((i<imax) and (qwe)) then qwe:=true else qwe:=false;
   while qwe do begin
    doc[j]:=add(doc[j],s);
```

```
    inc(j);
   end;
   if i>imax then imax:=i;
   inc(i);
end;
 end;
 close(f);
end;
procedure done;
var
 i:byte;
 f:text;
begin
 assign(f,'ans'+number+'.txt');
 rewrite(f);
 for i:=1 to imax do begin
  writeln(f,doc[i]);
 end;
 close(f);
end;
BEGIN
 init;
 done;
END.
```

7. Подземелье Джафара

```
Program Djafar;
 Uses Crt,Dos;
  Const
   MaxL=10;
   MaxN=50;
   MaxM=50;
  Type
   TypeOnePole=Record
                 St:Char;
                Num:Byte;
               End;
   TypePole=Array[1..MaxN,1..MaxM] of TypeOnePole;
   TypeOneMan=Record
                 X,Y:Byte;
              End;
   TypeMan=Array[0..MaxL] of TypeOneMan;
  Var
   NumMan,MaxMan : Byte;
   NumSum,MaxSum : Byte;
   Pole,Pole1,PoleS:TypePole;
    Man:TypeMan;
    M,N,L:Word;
```

```
  Name:String;
  NumK:Byte;
  MaxK:Byte;
 Procedure Load;
  Var
    F:Text;
   Buf:String;
    St:String;
   IOs:Integer;
   I,J:Byte;
  Begin
   Write("Enter FileName:");Readln(Name);
   Assign(F,Name);Reset(F);
   Readln(F,Buf);
   St:=Copy(Buf,1,Pos(",",Buf)-1);
   Delete(Buf,1,Pos(",",Buf));
   Val(St,N,IOs);
   St:=Copy(Buf,1,Pos(",",Buf)-1);
   Delete(Buf,1,Pos(",",Buf));
   Val(St,M,IOs);
   St:=Buf;
   Val(St,L,IOs);
   For i:=1 to N do
    Begin
     Readln(F,Buf);
     For j:=1 to M do
       Begin
        Pole[i,j].St:=Buf[j];
        Pole[i,j].Num:=255;
       End;
    End;
   for i:=0 to L-1 do
    Begin
     Readln(F,Buf);
     St:=Copy(Buf,1,Pos(",",Buf)-1);
     Delete(Buf,1,Pos(",",Buf));
     Val(St,Man[i].Y,IOs);
St:=Buf;
     Val(St,Man[i].X,IOs);
     Pole[Man[i].Y,Man[i].X].Num:=1;
     Pole[Man[i].Y,Man[i].X].St:='M';
    End;
   Close(F);
   MaxK:=0;
   MaxMan:=0;
   NumMan:=0;
   MaxSum:=0;
  End;
 Procedure Print;
```

```
 Var
  i,j: Byte;
 Begin
  for i:=1 to N do
   for j:=1 to M do
    Begin
     GotoXY(J,i);
     Write(Pole[i,j].St);
     GotoXY(J*3+M+1,i);
     if Pole[i,j].Num=255 then write("(0)") else Write(Pole[i,j].Num:2,' ");
    End;
  Readln;
 End;
Procedure Test(i:Byte);
 Var
  X,Y:Byte;
 Begin
if (NumK>MaxK) then
  Begin
    MaxK:=NumK;
    NumK:=0;
    NumMan:=0;
    NumSum:=0;
    For X:=0 to L-1 do
     if (Man[X].X=1) or (Man[X].X=M) or
        (Man[X].Y=1) or (Man[X].Y=N) then
        Begin
         Inc(NumMan);
         NumSum:=NumSum+Pole[Man[X].Y,Man[X].X].Num;
        End;
    if (NumMan>MaxMan) or ((NumMan=MaxMan) and (NumSum<MaxSum)) then
     Begin
      MaxMan:=NumMan;
      MaxSum:=NumSum;
      PoleS:=Pole;
     End;
    exit;
  End;
  I:=i mod L;
if (Man[i].X=1) or (Man[i].X=M) or
   (Man[i].Y=1) or (Man[i].Y=N) then
    Begin
     Inc(NumK);
     Test(I+1)
    End
 else
  Begin
     NumK:=0;
     X:=Man[i].X;
```

```
      Y:=Man[i].Y;
      Man[i].X:=X-1;
      Man[i].Y:=Y;
      if (X>1) and (Pole[Y,X-1].St<>'1') then
        Begin
         if (Pole[Y,X-1].Num>=Pole[Y,X].Num+1) then
          Begin
           Pole[Y,X-1].Num:=Pole[Y,X].Num+1;
           Pole[Y,X-1].St:='M';
           Test(i+1);
           Pole[Y,X-1].St:='0';
           Pole[Y,X-1].Num:=255;
          End;
        End;
      Man[i].X:=X+1;
      Man[i].Y:=Y;
      if (X<M) and (Pole[Y,X+1].St<>'1') then
        Begin
         if (Pole[Y,X+1].Num>=Pole[Y,X].Num+1) then
Begin
           Pole[Y,X+1].Num:=Pole[Y,X].Num+1;
           Pole[Y,X+1].St:='M';
           Test(i+1);
           Pole[Y,X+1].St:='0';
           Pole[Y,X+1].Num:=255;
          End;
        End;
      Man[i].X:=X;
      Man[i].Y:=Y-1;
      if (Y>1) and (Pole[Y-1,X].St<>'1') then
        Begin
         if (Pole[Y-1,X].Num>=Pole[Y,X].Num+1) then
          Begin
           Pole[Y-1,X].Num:=Pole[Y,X].Num+1;
           Pole[Y-1,X].St:='M';
           Test(i+1);
           Pole[Y-1,X].St:='0';
           Pole[Y-1,X].Num:=255;
          End;
        End;
      Man[i].X:=X;
      Man[i].Y:=Y+1;
      if (Y<N) and (Pole[Y+1,X].St<>'1') then
        Begin
         if (Pole[Y+1,X].Num>=Pole[Y,X].Num+1) then
          Begin
           Pole[Y+1,X].Num:=Pole[Y,X].Num+1;
           Pole[Y+1,X].St:='M';
           Test(i+1);
```

```
          Pole[Y+1,X].St:='0';
          Pole[Y+1,X].Num:=255;
         End;
       End;
     Man[i].X:=X;
     Man[i].Y:=Y;
   End;
 End;
Begin
 clrscr;
 Load;
 Clrscr;
 Test(0);
 ClrScr;
 Pole:=PoleS;
 TextColor(Red);
 Print;
End.
```

8. Ручей Моно

```
Program Ruchei_Mono;
 Uses Crt,Dos;
  Const
   MaxN=50;
   MaxM=15;
  Type
   TypeM=Array[1..MaxM+1,1..3,1..2] of Byte;
   TypeN=Array[1..MaxN] of Record
          M:TypeM;
        NumM:Byte;
         End;
  Var
      N:TypeN;
    NumN:Byte;
      S:Integer;
   SavH1,
    SavH:Array[1..MaxM,1..2] of Byte;
   L,LS:Byte;
    T,TMin:Real;
   Victory:Boolean;
     Found:Boolean;
   NumU,NumMinU:Byte;
  Procedure Test(K:Byte);
   Var
    i,j:Byte;
    T0:real;
   Begin
    for i:=1 to 3 do
```

```
    Begin
    Inc(NumMinU);
    for j:=N[L].M[K,i,1] downto 1 do
     if ((S-J+1)>0)  then
Begin
      S:=S-J;T0:=J/N[L].M[K,i,2];
      T:=T+T0;
      SavH[K,1]:=J;
      SavH[K,2]:=N[L].M[K,i,2];
      if (S=0) and (T<Tmin) then
       Begin
        Sound(500);
        Tmin:=T;
        SavH1:=SavH;
        Found:=True;
        Delay(100);
        NoSound;
        NumMinU:=NumU;
       End else if (S>0) and (K<N[L].NumM) and (T<Tmin) and (NumU<NumMinU) then Test(K+1);
      T:=T-T0;
      FillChar(SavH[K],2,0);
      S:=S+j;
     End;
     Dec(NumU);
     SavH[K][1]:=0;
     SavH[K][2]:=0;
     if (K<N[L].NumM) then Test(K+1);
   End;
  End;
  Procedure Test1(K:Byte);
   var
    T0:Real;
    i:Byte;
   Begin
    for i:=1 to 3 do
     Begin
      if (N[L].M[K,i,1]>S) then
       Begin
        T0:=S/N[L].M[K,I,2];
        SavH[K][1]:=S;
        SavH[K][2]:=N[L].M[K,i,2];
        if (TMin>T+T0) then
         Begin
          Found:=True;
          Sound(500);
          Tmin:=T+T0;
          SavH1:=SavH;
          Delay(100);
          NoSound;
```

```
          End;
         FillChar(SavH[K],2,0);
        End
       else
        Begin
         S:=S-N[L].M[K,i,1];
         T0:=N[L].M[K,i,1]/N[L].M[K,I,2];
         T:=T+T0;
         SavH[K][1]:=N[L].M[K,i,1];
         SavH[K][2]:=N[L].M[K,i,2];
         if (K<N[L].NumM) and (T<Tmin)  then Test1(K+1);
         T:=T-T0;
         FillChar(SavH[K],2,0);
         S:=S+N[L].M[K,i,1];
        End;
      End;
    End;
  Procedure TestStart;
  Var
   i,j,k,l:Byte;
   Obmen:Array[1..2] of Byte;
  Begin
   for i:=1 to NumN do
    for J:=1 to N[i].NumM do
     Begin
      for L:=1 to 2 do
       for K:=L+1 to 3 do
        if N[i].M[j,l,2]<N[i].M[j,k,2] then
         Begin
          Obmen[2]:=N[i].M[j,k,2];
          Obmen[1]:=N[i].M[j,k,1];
          N[i].M[j,k]:=N[i].M[j,l];
          N[i].M[j,l,1]:=Obmen[1];
          N[i].M[j,l,2]:=Obmen[2];
         End;
     End;
for i:=1 to NumN do
    Begin
     for L:=1 to N[i].NumM-1 do
      for K:=L+1 to N[i].NumM do
       if N[i].M[L,1,2]<N[i].M[K,1,2]  then
         Begin
          N[i].M[N[i].NumM+1]:=N[i].M[l];
          N[i].M[L]:=N[i].M[K];
          N[i].M[K]:=N[i].M[N[i].NumM+1];
         End;
    End;
  End;
```

```
Procedure Load;
 Var
  F:Text;
  Name:String;
  i,j:Byte;
  Buf:String;
  IOs:Integer;
 Begin
  clrscr;
  write("Enter FileName:");Readln(Name);
  write("File Read -");
  Assign(F,Name);
  Reset(F);
  Readln(F,NumN);
  for i:=1 to NumN do
   Begin
    Readln(F,N[i].NumM);
    for j:=1 to N[i].NumM do
     Begin
      Readln(F,Buf);
     Val(Copy(Buf,1,Pos(",",Buf)-1),N[i].M[J,1,1],IOs);
      Delete(Buf,1,Pos(",",Buf));
     Val(Copy(Buf,1,Pos(",",Buf)-1),N[i].M[j,1,2],IOs);
      Delete(Buf,1,Pos(",",Buf));
     Val(Copy(Buf,1,Pos(",",Buf)-1),N[i].M[j,2,1],IOs);
      Delete(Buf,1,Pos(",",Buf));
     Val(Copy(Buf,1,Pos(",",Buf)-1),N[i].M[j,2,2],IOs);
      Delete(Buf,1,Pos(",",Buf));
     Val(Copy(Buf,1,Pos(",",Buf)-1),N[i].M[j,3,1],IOs);
      Delete(Buf,1,Pos(",",Buf));
      Val(Buf,N[i].M[j,3,2],IOs);
     End;
   End;
  Readln(F,S);
  Close(F);
  TestStart;
  Writeln("Ok");
 End;
Procedure Save;
 Var
  I:Byte;
  F:Text;
  Name:String;
 Begin
  clrscr;
  write("Enter FileName:");Readln(Name);
  Assign(F,Name);
  Rewrite(F);
  if Victory then
```

```
     Begin
      writeln(F,LS);
      For I:=1 to N[LS].NumM do
        writeln(F,I,',',SavH1[i][1],',',SavH1[i][2]);
     End else write(F,'Net Reshenia');
   Close(F);
   end;
  Begin
    Load;
    L:=1;
    Ls:=0;
    Tmin:=999;
   Victory:=False;
  for L:=1 to NumN do
   Begin
    NumMinU:=30;
    NumU:=0;
    writeln("NumL=",L);
    FillChar(SavH,SizeOF(SavH),0);
    T:=0;
    Found:=False;
Test1(1);
if Found then
     Begin
      lS:=L;
      Victory:=True;
     End;
   End;
   Save;
  End.
```

9. Приключения в королевстве Ном

```
program doroty;
uses crt;
const
  Wall = -1;
var
  Lab: array[1..50,0..21,0..31] of shortint;
  Lines,Columns,LabNum,L0,C0: byte;

procedure Init;
var
  MyLab: array[1..5,0..21,0..31] of shortint;
  LabOrder: array[1..10] of byte;
  f: text;
  infile,tempstr: string;
  IOError,i,j,l: integer;
begin
```

```
  clrscr;
  FillChar(MyLab,SizeOf(MyLab),$FF);
  readln(infile);
  assign(f,infile);
  reset(f);
  readln(f,tempstr);
  Val(Copy(tempstr,1,Pos(“,”,tempstr)-1),Lines,IOError);
  Delete(tempstr,1,Pos(“,”,tempstr));
  Val(Copy(tempstr,1,Pos(“,”,tempstr)-1),Columns,IOError);
  Delete(tempstr,1,Pos(“,”,tempstr));
  Val(tempstr,LabNum,IOError);
  for i:=1 to LabNum do
    for j:=1 to Lines do
      begin
        readln(f,tempstr);
        for l:=1 to Columns do
          if tempstr[l]=’1' then
            MyLab[i,j,l]:=Wall
          else
            MyLab[i,j,l]:=0;
      end;
  readln(f,tempstr);
  Val(Copy(tempstr,1,Pos(“,”,tempstr)-1),L0,IOError);
  Delete(tempstr,1,Pos(“,”,tempstr));
  Val(tempstr,C0,IOError);
  readln(f,tempstr);
  for i:=1 to 9 do
    begin
      Val(Copy(tempstr,1,Pos(“,”,tempstr)-1),LabOrder[i],IOError);
      Delete(tempstr,1,Pos(“,”,tempstr));
    end;
  Val(tempstr,LabOrder[10],IOError);
  close(f);
  for i:=1 to 10 do
    for j:=1 to 5 do
      Move(MyLab[LabOrder[i]],Lab[(i-1)*5+j],SizeOf(Lab) div 50);
end;

procedure Run;
const
  Turn: array[1..5,1..2] of shortint =
    ((-1,0),(0,1),(1,0),(0,-1),(0,0));
type TQueuePointer = ^TQueueMember;
     TQueueMember = record
       l,c,s: byte;
       Next: TQueuePointer;
     end;
     TPoint = record
       l,c: byte;
```

```
    end;
var
ReadIndex,WriteIndex: TQueuePointer;
  l,c,s,i,StepNum: byte;
  FoundExit: boolean;
  ExitArr: array[1..50] of TPoint;

procedure WriteToQueue(s,l,c:byte);
var
  x: TQueuePointer;
begin
  New(x);
  Lab[s,l,c]:=1;
  x^.l:=l;
  x^.c:=c;
  x^.s:=s;
  x^.Next:=nil;
  if WriteIndex<>nil then
    WriteIndex^.Next:=x
  else
    ReadIndex:=x;
  WriteIndex:=x;
end;
procedure ReadFromQueue(var s,l,c:byte);
var
  x: TQueuePointer;
begin
  l:=ReadIndex^.l;
  c:=ReadIndex^.c;
  s:=ReadIndex^.s;
  x:=ReadIndex;
  ReadIndex:=ReadIndex^.Next;
  if ReadIndex=nil then
    WriteIndex:=nil;
  Dispose(x);
end;
procedure DrawLab(s:byte);
var
  i,j: byte;
begin
  clrscr;
  for i:=1 to Lines do
    for j:=1 to Columns do
      if Lab[s,i,j]=Wall then
        begin
          gotoxy(j*2,i);
          write("*");
        end;
end;
```

```
begin (* procedure Run *)
  DrawLab(1);
  gotoxy(C0*2,L0);
  write("X");
  New(WriteIndex);
  WriteIndex^.s:=0;
  WriteIndex^.l:=L0;
  WriteIndex^.c:=C0;
  WriteIndex^.Next:=nil;
  ReadIndex:=WriteIndex;
  while ReadIndex<>nil do
    begin
      ReadFromQueue(s,l,c);
      if (l in [1,Lines]) or (c in [1,Columns]) then
        begin
        (* save exit and quit *)
          FoundExit:=true;
          StepNum:=s;
          while s>=1 do
            begin
              ExitArr[s].l:=l;
              ExitArr[s].c:=c;
              s:=s-1;
              for i:=1 to 5 do
                if Lab[s,l+Turn[i,1],c+Turn[i,2]]=1 then
                  begin
                    l:=l+Turn[i,1];
                    c:=c+Turn[i,2];
                    break;
                  end;
            end;
          for i:=1 to StepNum do
            begin
              if ((i mod 5)=1) and (i<>1) then
                DrawLab(i);
gotoxy(ExitArr[i].c*2,ExitArr[i].l);
              write(i);
              Delay(2000);
            end;
          halt;
        end;
      if s<>50 then
        for i:=1 to 5 do
          if Lab[s+1,l+Turn[i,1],c+Turn[i,2]]=0 then
            WriteToQueue(s+1,l+Turn[i,1],c+Turn[i,2]);
    end;
end; (* procedure Run *)
```

```
begin
  Init;
  Run;
end.
```

10. Сфера

```
Program Sfer;
var
 k,r:extended;
procedure init;
begin
 write(“Введите радиус:”);
 readln(r);
end;
procedure run;
label qqq;
var
 rk,xk,yk,zk,x1,y1,z:extended;
 x,y:integer;
begin
 k:=0;
 rk:=r*r;
 for x:=1 to round(r) do begin
  for y:=1 to round(r) do begin
   x1:=x;y1:=y;
   xk:=x1*x1;yk:=y1*y1;
   if (rk-xk-yk<0) then goto qqq;
   z:=sqrt(rk-xk-yk);
   z:=round(int(z));
   zk:=z*z;
   if (xk+yk+zk)<=rk then k:=k+z;
qqq:
  end;
 end;
end;
procedure done;
begin
 writeln(“Кол-во кубиков=”,k*8:0:0);
end;
BEGIN
 init;
 run;
 done;
END.
```

11. Треугольник

```
Program treugoln;
```

```
var
 n,m:integer;
 number:string;
 otv:array [1..4] of real;
 a,b:array [1..10000,1..2] of shortint;
procedure init;
var
 i,j:integer;
 k,l:shortint;
 f:text;
begin
 write(“Введите номер теста:”);
 readln(number);
 assign(f,’test’+number+’.txt’);
 reset(f);
 n:=1;
 while not eof(f) do begin
  readln(f,a[n,1],a[n,2]);
  inc(n);
 end;
 dec(n);
 close(f);
for i:=1 to n-1 do begin
  for j:=i+1 to n do begin
   if a[i,2]>a[j,2] then begin
    k:=a[i,1];
    l:=a[i,2];
    a[i,1]:=a[j,1];
    a[i,2]:=a[j,2];
    a[j,1]:=k;
    a[j,2]:=l;
   end;
  end;
 end;
end;
procedure postroenie;
label lb,lb1;
var
 mnog:boolean;
 znak1,znak:integer;
 a1,b1,c1:integer;
 num,max:shortint;
 i1,i,j:integer;
begin
 num:=1;
 max:=a[1,1];
 for i:=2 to n do if max<a[i,1] then begin
  max:=a[i,1];
  num:=i;
```

```
 end;
 b[1]:=a[num];
 j:=2;
 repeat
  mnog:=false;
  for i:=1 to n do begin
    if ((a[i,1]=b[j-2,1]) and (a[i,2]=b[j-2,2])) then goto lb;
    if ((a[i,1]=b[j-1,1]) and (a[i,2]=b[j-1,2])) then goto lb;
    b[j]:=a[i];
    a1:=b[j,2]-b[j-1,2];
    b1:=b[j-1,1]-b[j,1];
    c1:=(-b[j-1,1]*(b[j,2]-b[j-1,2]))+(b[j-1,2]*(b[j,1]-b[j-1,1]));
    znak:=0;znak1:=0;
    mnog:=true;
    for i1:=1 to n do begin
     znak1:=znak;
     if (a1*a[i1,1]+b1*a[i1,2]+c1)>0 then znak:=1;
     if (a1*a[i1,1]+b1*a[i1,2]+c1)<0 then znak:=-1;
     if ((znak1<>znak) and (znak1<>0) and (znak<>0)) then begin
      mnog:=false;
      goto lb;
     end;
    end;
lb:
    if mnog then goto lb1;
  end;
lb1:
  inc(j);
 until ((b[j-1,1]=b[1,1]) and (b[j-1,2]=b[1,2]));
 dec(j,2);
 m:=j;
end;
function pl(i1,i2,i3:integer):real;
var
 q,w:real;
 p:real;
 a2,b2,c2:real;
begin
 a2:=sqrt((b[i1,1]-b[i2,1])*(b[i1,1]-b[i2,1])+(b[i1,2]-b[i2,2])*(b[i1,2]-b[i2,2]));
 b2:=sqrt((b[i2,1]-b[i3,1])*(b[i2,1]-b[i3,1])+(b[i2,2]-b[i3,2])*(b[i2,2]-b[i3,2]));
 c2:=sqrt((b[i3,1]-b[i1,1])*(b[i3,1]-b[i1,1])+(b[i3,2]-b[i1,2])*(b[i3,2]-b[i1,2]));
 p:=(a2+b2+c2)/2;
 w:=p*(p-a2)*(p-b2)*(p-c2);
 q:=sqrt(w);
 pl:=q;
end;
procedure perebor;
var
 s:real;
```

```
i,j,k:integer;
begin
 otv[1]:=1;
 otv[2]:=2;
 otv[3]:=3;
 otv[4]:=pl(round(otv[1]),round(otv[2]),round(otv[3]));
 for i:=1 to m-2 do begin
  for j:=i+1 to m-1 do begin
   for k:=j+1 to m do begin
    s:=pl(i,j,k);
    if s>otv[4] then begin
     otv[4]:=s;
     otv[1]:=i;
     otv[2]:=j;
     otv[3]:=k;
    end;
   end;
  end;
 end;
end;
procedure run;
begin
 postroenie;
 perebor;
end;
procedure done;
var
 f:text;
begin
 assign(f,'ans'+number+'.txt');
 rewrite(f);
 writeln(f,'Площадь - ',otv[4]:8:2);
 writeln(f,'Точки:');
 writeln(f,b[round(otv[1]),1],' ',b[round(otv[1]),2]);
 writeln(f,b[round(otv[2]),1],' ',b[round(otv[2]),2]);
 writeln(f,b[round(otv[3]),1],' ',b[round(otv[3]),2]);
 close(f);
end;
BEGIN
 init;
 run;
 done;
END.
```

12. Утиные истории

```
Program Duck_Tales;
 Uses Crt,Dos;
  Const
```

```
  MaxK=50;
  MaxCity=100;
 Type
  TypeCity=array[1..MaxCity,1..MaxCity] of Byte;
  TypeOneHod=record
     Sum:Word;
     Old:Byte;
    End;
  TypeHod=array[1..MaxK,1..MaxCity] of TypeOneHod;
 Var
  Hod:TypeHod;
  City:TypeCity;
  K,N:Byte;
  NumFile:String;
  Sum,MaxSum,Day1:Integer;
  J1:Integer;
Procedure LoadFile(Var City:TypeCity);
 Var
  F:Text;
  Name:String;
  Bufer:String;
  ios:Integer;
  I,J:Byte;
 Begin
  Clrscr;
  write("Enter File Name:");Readln(Name);
  Assign(F,Name);
  Reset(F);
  delete(Name,1,1);
  while  UpCase(Name[1]) in ["A"..'Z'] do Delete(Name,1,1);
NumFile:=Copy(Name,1,Pos(".",Name)-1);
  Readln(F,Bufer);
  Val(Copy(Bufer,1,Pos(",",Bufer)-1),N,Ios);
  Delete(Bufer,1,Pos(",",Bufer));
  Val(Bufer,K,Ios);
  for I:=1 to N do
   Begin
    Readln(F,Bufer);
    for j:=1 to N-1 do
     Begin
      Val(Copy(Bufer,1,Pos(":",Bufer)-1),City[i,j],Ios);
      Delete(Bufer,1,Pos(":",Bufer));
     End;
    Val(Bufer,City[i,N],Ios);
   End;
  Close(F);
 End;
Procedure TestHod;
 Var
```

```
 I,J:Byte;
 Day:Byte;
Begin
 Fillchar(Hod,SizeOF(Hod),#0);
 For i:=2 to N  do
  if (I<>1) and (City[1,I]=1) then
       Begin
        Hod[1,I].Sum:=City[i,i]+City[1,1];
        Hod[1,i].Old:=1;
       End;
 For Day:=2 to K do
  For I:=1 to N do
   if (Hod[Day-1,I].Sum>0) then
    for j:=1 to N do
     if (I<>J) and (city[i,J]=1) and
(Hod[Day,J].Sum<Hod[Day-1,I].Sum+City[j,j]) then
               Begin
            Hod[Day,J].Sum:=City[j,j]+Hod[Day-1,I].Sum;
           Hod[Day,j].Old:=i;
               End;
End;
Procedure Test1(i1,i2:Byte);
 Var
 i:Byte;
Begin
 Day1:=day1+1;
 if (MaxSum<Sum) and (I1=N) then  MaxSum:=Sum;
 if Day1<=K then
   Begin
    sum:=Sum+City[i1,i1];
    if (MaxSum<Sum) and (I1=N) then  MaxSum:=Sum;
    For i:=i2 to N do
    if (City[i1,i]=1) and (I<>i1) then Test1(i,1);
             Sum:=Sum-City[i1,i1];
  End;
  day1:=Day1-1;
End;
Procedure BackTrace(Hod:TypeHod);
 Var
 F:Text;
 Name:String;
 i,j:Byte;
 Back:array[1..MaxK] of Byte;
 Day:Byte;
Begin
 FillChar(Back,SizeOf(Back),#0);
 Name:='2ANSW'+NumFile+'.Txt';
 Assign(F,Name);
 Rewrite(F);
```

```
  I:=K;
  while (I>0) and(Hod[I,N].Sum=0) and
 (Hod[i,N].Old=0) do I:=I-1;
  if I=0 then
   Begin
    Writeln(F,'-Ґв аҐиҐ-Ёп !!!');
    Close(F);
    Halt;
   End;
  J:=0;
  Day:=i;
  I:=N;
  Repeat
   J:=J+1;
Back[j]:=Hod[Day,I].Old;
   I:=Hod[Day,I].Old;
   Day:=Day-1;
  Until Day=0;
  writeln(F,J);
  writeln(F,Hod[J,N].Sum);
  write("Day=",J,' Sum=',Hod[j,n].Sum);
  for i:=J downto 1 do
   writeln(F,Back[i]);
   writeln(F,N);
  Close(F);
End;
Begin
 LoadFile(City);
 TestHod;
 BackTrace(Hod);
 Day1:=0;
 Sum:=0;
 j1:=0;
  MaxSum:=0;
  Test1(1,1);
End.
```

13. Завешение старого Аги

```
Program Starago_Agi;
 Uses Crt,Dos;
  Const
   MaxN=1000;
  Type
   TypeMat=array[1..MaxN] of Integer;
  Var
   N:Integer;
   NumMan:Integer;
   Mat:TypeMat;
```

```
 Procedure LoadFile(Var Mat:TypeMat);
  Var
   F:Text;
   Name:String;
   Bufer:String;
   N1,N2:Integer;
   i,Ios:Integer;
  Begin
   NumMan:=0;
   Clrscr;
   Write("Enter File Name:");Readln(Name);
   Assign(F,Name);
   Reset(F);
   Readln(F,N);
   For i:=1 to N do  Mat[i]:=1;
   I:=1;
   While Not Eof(F) do
    Begin
     Readln(F,Bufer);
     Val(Copy(Bufer,1,Pos("-",Bufer)-1),N1,Ios);
     Delete(Bufer,1,Pos("-",Bufer));
     Val(Bufer,N2,Ios);
     if I=N1 then
       if Mat[N2]=N1 then Inc(Mat[N2]) else Begin End
      Else
       Begin
        I:=N1;
        if Mat[N2]=Mat[N1] then Inc(Mat[N2]);
        NumMan:=Mat[N2];
       End;
    End;
   Close(F);
  End;
 Procedure SaveFile(Mat:TypeMat);
  Var
   F:Text;
   I:Integer;
   Name:String;
  Begin
   Clrscr;
   write("Enter Out FileName:");Readln(Name);
   Assign(F,Name);
   Rewrite(F);
   Writeln(F,NumMan);
   for i:=1 to N do
    Begin
     write(F,I);
     write(F,'-');
Writeln(F,Mat[i]);
```

```
   End;
  Close(F);
 End;
 Begin
  LoadFile(Mat);
  SaveFile(Mat);
 End.
```

14. В стране Рудокопов

```
Program Zad1;
 Uses Crt,Graph,Dos;
  Const
   MaxN=40;
   MaxM=40;
   MaxK=30;
   MaxList=MaxM*MaxN+100;
 Type
  TypeMat1=array[1..MaxK,0..MaxM,0..MaxN] of Byte;
  TypeMat=Array[0..MaxM,0..MaxN] of Record
                                    Status:Byte;
                                    Mo,No:ShortInt;
                                    Sum:Byte;
                                   End;
  TypeInfo=record
     N,M:ShortInt;
    end;
  TypeList=Array[1..MaxList] of TypeInfo;
  TypeTxy=array[1..8] of TypeInfo;
  TypeTop=Record
     Heap:Word;
     List:TypeList;
   End;
  TypeL=array[1..MaxK] of record
          N1,M1:Integer;
          N2,M2:Integer;
         End;
  Var
   Mat:TypeMat;
   Mat1:TypeMat1;
   N,M,K:INteger;
   L:typel;
   Nman,Mman:Integer; {Начальные координаты }
   NumFile:String;
Procedure PushRIght(Var Top:TypeTop;
                      Info:TypeInfo;
                   Var Ok:Boolean);
  Label Exit;
  Begin
```

```
   if top.Heap=MaxList then
    Begin
     OK:=False;
     goto Exit;
    End;
  OK:=True;
  Inc(Top.Heap);
  Top.List[Top.Heap]:=Info;
 Exit:
End;
Procedure PopLeft(Var Top:TypeTop;
                  Var Info:TypeInfo;
                  Var Ok:Boolean);
 Label Exit;
 Var
  i:Integer;
 Begin
  if Top.Heap=0 then
   Begin
    Ok:=False;
    goto Exit;
   end;
  Ok:=True;
  Info:=Top.List[1];
  For I:=1 to Top.Heap-1 do Top.List[i]:=Top.List[i+1];
  Dec(Top.Heap);
 Exit:
End;
procedure PopRight(Var top:typeTop;
         Var Info:TypeInfo;
         Var Ok:Boolean);
Label Exit;
Begin
 if top.Heap=0 then
  Begin
   Ok:=False;
   goto Exit;
  End;
 Ok:=true;
 info:=top.List[Top.Heap];
 Dec(Top.Heap);
 Exit:
End;
 Procedure OpenList(Var Top:TypeTop);
  Begin
  FillChar(Top,SizeOf(Top),#0);
  End;
Procedure LoadFile;
 Label Exit;
```

```
Var
 F:Text;
 grDriver: Integer;
 grMode: Integer;
 ErrCode: Integer;
 Name:String;
 Bufer:String;
 IOs,i,i1,j:Integer;
Begin
 textcolor(7);
 Clrscr;
 FillChar(Mat,SizeOf(Mat),#0);
 write("Enter File Name:");Readln(Name);
 Assign(F,Name);
 Reset(F);
 delete(Name,1,1);
 while UpCase(Name[1]) in ["A"..'Z'] do Delete(Name,1,1);
 NumFile:=copy(Name,1,Pos(".",Name)-1);
 Readln(F,Bufer);
 Val(Copy(Bufer,1,Pos(",",Bufer)-1),N,Ios);
 Delete(Bufer,1,Pos(",",Bufer));
 Val(Copy(Bufer,1,Pos(",",Bufer)-1),M,Ios);
 Delete(Bufer,1,Pos(",",Bufer));
 Val(Bufer,K,ios);
 if (N<10) or (N>MaxN) or (M<10) or (MaxM<M) or (K<3) or (K>MaxK) then
   Begin
    write("Данные не верны ");
    Halt;
   End;
 for I:=1 to K do
  Begin
   Readln(F,Bufer);
   Val(Copy(Bufer,1,Pos(",",Bufer)-1),L[i].N1,Ios);
   Delete(Bufer,1,Pos(",",Bufer));
   Val(Copy(Bufer,1,Pos(":",Bufer)-1),L[i].M1,Ios);
   Delete(Bufer,1,Pos(":",Bufer));
   Val(Copy(Bufer,1,Pos(",",Bufer)-1),L[i].N2,Ios);
   Delete(Bufer,1,Pos(",",Bufer));
   Val(Bufer,L[i].M2,Ios);
   if (L[i].N1<0) or (L[i].N1>MaxN) or
      (L[i].N2<0) or (L[i].N2>MaxN) or
      (L[i].M1<0) or (L[i].M1>MaxM) or
      (L[i].M2<0) or (L[i].M2>MaxM)  then
   Begin
    write("Данные не верны ");
    Halt;
   End;
  End;
 Readln(F,Bufer);
```

```
  Val(Copy(Bufer,1,Pos(“,”,Bufer)-1),NMan,Ios);
  Delete(Bufer,1,Pos(“,”,Bufer));
  Val(Bufer,MMan,ios);
  grDriver := 9;
  grMode:=2;
  InitGraph(grDriver, grMode,’’);
  ErrCode := GraphResult;
  if ErrCode <> grOk then
    Begin
     write(“Нет специального графического файла “);
     Halt;
    End;
  ClearDevice;
  SetColor(1);
Fillchar(Mat1,SizeOf(Mat1),#0);
  for I1:=1 to K do
   Begin
    SetColor(1);
    Line(L[i1].M1+1,L[i1].N1+1,L[i1].M2+1,L[i1].N2+1);
    for I:=0 to M do
     For J:=0 to N do
       if GetPixel(I+1,j+1)=1 then
         Begin
          Mat1[i1,i,j]:=1; {бвЃ-  }
          Mat[i,j].Status:=1;
         End;
     Setcolor(0);
     Line(L[i1].M1+1,L[i1].N1+1,L[i1].M2+1,L[i1].N2+1);
   End;
Exit:
  Close(F);
  CloseGraph;
End;
Procedure Test;
 Label Start;
 Var
  Txy:typeTxy;
  Xy:typeInfo;
  i,j:Integer;
  i1,flag:Byte;
  top:typeTop;
  Ok:Boolean;
  Txy1:array[1..5] of TypeInfo;
 Procedure TestHod( xy : TypeInfo;
                    var Txy:typeTxy);
  Var
   It:integer;
  Begin
   Txy[1].N:=Xy.N+1;Txy[1].M:=xy.M;
```

```
  Txy[2].N:=Xy.N;Txy[2].M:=xy.M+1;
  Txy[3].N:=Xy.N-1;Txy[3].M:=xy.M;
  Txy[4].N:=Xy.N;Txy[4].M:=xy.M-1;
  Txy[5].N:=Xy.N+1;Txy[5].M:=xy.M+1;
  Txy[6].N:=Xy.N-1;Txy[6].M:=xy.M+1;
  Txy[7].N:=Xy.N-1;Txy[7].M:=xy.M-1;
  Txy[8].N:=Xy.N+1;Txy[8].M:=xy.M-1;
  for it:=1 to 8 do
   if (Txy[it].N<0) or (Txy[it].M<0) or
      (Txy[it].N>N) or (Txy[it].M>M) then
       Begin
        Txy[it].N:=-1;
        Txy[it].M:=-1;
       End;
 End;
Begin
 Xy.N:=Nman;
 Xy.M:=MmaN;
 OpenList(Top);
 Mat[xy.M,Xy.N].Status:=3;
 Mat[Xy.M,Xy.N].Sum:=1;
 for i:=0 to M do
  for j:=0 to N do
   Begin
    gotoxy(i+1,j+1);
    write(Mat[i,j].Status);
   end;
 for i1:=1 to K do
 Begin
  for i:=0 to M do
  for j:=0 to N do
   Begin
    gotoxy(i+1+M+2,j+1);
    write(Mat1[i1,i,j]);
   end;
 delay(100);
End;
Start:
 TestHod(Xy,Txy);
 For i:=1 to 4 do
  if (Txy[i].N>=0) and (Txy[i].M>=0) then
   Begin
   Flag:=1;
   for i1:=1 to K do
    if Mat1[i1,Txy[i].M,Txy[i].N]>0 then Flag:=0;
   if (Flag=1) and (Mat[Txy[i].M,Txy[i].N].Status=0) then
Begin
     Mat[Txy[i].M,Txy[i].N].Mo:=Xy.M;
     Mat[Txy[i].M,Txy[i].N].No:=Xy.N;
```

```
      Mat[txy[i].M,Txy[i].N].Sum:=Mat[xy.M,xy.N].Sum+1;
      Mat[txy[i].M,Txy[i].N].Status:=3;
      PushRight(Top,Txy[i],Ok);
      gotoxy(txy[i].M+1,Txy[i].N+1);
      textcolor(blue);
      write(chr(1));
     End;
    End;
  For i:=1 to 4 do Txy1[i]:=Txy[i];
  txy1[5]:=Txy1[1];
  for i:=1 to 4 do
   if (Txy[i+4].N>=0) and (Txy[i+4].M>=0) then
    Begin
    flag:=1;
    for i1:=1 to K do
     Begin
    if (Txy1[i+1].N>-1) and
       (Txy1[i].M>-1) and
       (Mat1[i1,txy1[i].M,Txy1[i].N]=Mat1[i1,txy1[i+1].M,Txy1[i+1].N]) and
       (Mat1[i1,txy1[i].M,Txy1[i].N]>0) then  Flag:=0;
     End;
    if (Mat[txy[i+4].M,Txy[i+4].N].Status=0) and (Flag=1) then
     Begin
      Mat[Txy[i+4].M,Txy[i+4].N].Mo:=Xy.M;
      Mat[Txy[i+4].M,Txy[i+4].N].No:=Xy.N;
      Mat[txy[i+4].M,Txy[i+4].N].Sum:=Mat[xy.M,xy.N].Sum+1;
      Mat[txy[i+4].M,Txy[i+4].N].Status:=3;
      PushRight(Top,Txy[i+4],Ok);
      textcolor(red);
      gotoxy(txy[i+4].M+1,Txy[i+4].N+1);
      write(chr(1));
     End;
    End;
  PopLeft(top,Xy,Ok);
 if Ok=true then goto Start;
 End;
Procedure BacKTrace;
 Var
  i,j:Integer;
  SumF:Integer;
  Flag:Integer;
  Back,Back1:TypeInfo;
  Top:TypeTop;
  Txy:typeTxy;
  ok:Boolean;
  Name:String;
  F:text;
 Procedure TestHod( xy : TypeInfo;
                 var Txy:typeTxy);
```

```
 Var
  It:integer;
 Begin
  Txy[1].N:=Xy.N+1;Txy[1].M:=xy.M;
  Txy[2].N:=Xy.N-1;Txy[2].M:=xy.M;
  Txy[3].N:=Xy.N;Txy[3].M:=xy.M+1;
  Txy[4].N:=Xy.N;Txy[4].M:=xy.M-1;
  Txy[5].N:=Xy.N+1;Txy[5].M:=xy.M-1;
  Txy[6].N:=Xy.N-1;Txy[6].M:=xy.M-1;
  Txy[7].N:=Xy.N+1;Txy[7].M:=xy.M+1;
  Txy[8].N:=Xy.N-1;Txy[8].M:=xy.M+1;
  for it:=1 to 8 do
    if (Txy[it].N<0) or (Txy[it].M<0) or
      (Txy[it].N>N) or (Txy[it].M>M) then
       Begin
        Txy[it].N:=-1;
        Txy[it].M:=-1;
       End;
 End;
Begin
 SumF:=1000;
 Flag:=0;
 for i:=0 to M do
  Begin
   If (Mat[i,0].Sum<SumF) and (Mat[i,0].Sum<>0) then
    Begin
     flag:=1;
     SumF:=Mat[i,0].Sum;
Back.M:=I;
     Back.N:=0;
    End;
  If (Mat[i,N].Sum<SumF) and (Mat[i,n].Sum<>0)then
    Begin
     flag:=1;
     SumF:=Mat[i,N].Sum;
     Back.M:=I;
     Back.N:=N;
    End;
  End;
 for i:=0 to N do
  Begin
   If (Mat[0,i].Sum<SumF) and (MAt[0,i].Sum<>0) then
    Begin
     flag:=1;
     SumF:=Mat[0,i].Sum;
     Back.M:=0;
     Back.N:=i;
    End;
  If (Mat[M,i].Sum<SumF) and (mat[M,I].SUm<>0) then
```

```
     Begin
      flag:=1;
      SumF:=Mat[M,i].Sum;
      Back.M:=M;
      Back.N:=I;
     End;
   End;
  Name:='3ANS'+NumFile+'.Txt';
  Assign(F,Name);
  Rewrite(F);
  if Flag=0 then
   Begin
    write(Нет решения');
    Close(F);
    Halt;
   End;
  Writeln(F,Mat[Back.M,Back.N].Sum-1);
  OpenList(top);
  PushRight(Top,Back,ok);
  Back1:=Back;
  textcolor(Green);
    gotoxy(Back.M+1,Back.N+1);
    write(chr(3));
  repeat
    Back.N:=Mat[Back1.M,Back1.N].No;
    Back.M:=Mat[Back1.M,Back1.N].Mo;
    Back1:=Back;
    PushRight(top,Back,ok);
    gotoxy(Back.M+1,Back.N+1);
    write(chr(3));
  Until (Back1.N=Nman) and (Back1.M=Mman);

  for i:=Top.Heap-1 Downto 1 do
   Begin
    write(F,Top.List[i].N);
    Write(F,',');
    writeln(f,Top.List[i].M);
   End;
 close(F);
End;
Begin
 LoadFile;
 Test;
 BacKTrace;
End.
```

15. Головоломка Уитикера

```
Program Zad15;
var
 hod:byte;
 vyh:boolean;
 z:string;
 n,m:byte;
 otv:array [0..100,1..3] of byte;
 a,b:array [1..10,1..10] of byte;

procedure init;
var
 s:string;
q,code:integer;
 f:text;
 i,j:byte;
 c:char;
begin
 write(“Введите номер теста:”);
 readln(z);
 assign(f,’test’+z+’.txt’);
 reset(f);
 readln(f,s);
 q:=pos(“,”,s);
 val(copy(s,1,q-1),n,code);
 val(copy(s,q+1,length(s)-q),m,code);
 for i:=1 to n do begin
  for j:=1 to m do begin
   read(f,c);
   s:=c;
   val(s,a[i,j],code);
   if a[i,j]=3 then a[i,j]:=99;
   if a[i,j]=2 then begin
    otv[0,2]:=i;
    otv[0,3]:=j;
   end;
  end;
  read(f,c);
  read(f,c);
 end;
 close(f);
end;

procedure run;
label lb1;
var
 i1,j1,q,i,j:byte;
function change:boolean;
label lb;
var
```

```
 w:boolean;
 i,j:byte;
begin
 w:=false;
 for i:=1 to n do begin
  for j:=1 to m do begin
   if a[i,j]<>b[i,j] then begin
    w:=true;
    goto lb;
   end;
  end;
 end;
lb:
 change:=w;
end;
begin
 vyh:=false;
 q:=2;
 b:=a;
 b[1,1]:=23;
 while ((not vyh) and (change)) do begin
  b:=a;
  for i:=1 to n do begin
   for j:=1 to m do begin
    if a[i,j]=q then begin
     for i1:=i+1 to n do begin
      if a[i1,j]<>0 then begin
       if a[i1,j]=99 then begin
        vyh:=true;
        goto lb1;
       end else
       if a[i1,j]=1 then begin
if ((i1<>i+1) and (a[i1-1,j]=0)) then a[i1-1,j]:=q+1;
        i1:=n;
       end;
      end;
     end;
     for i1:=i-1 downto 1 do begin
      if a[i1,j]<>0 then begin
       if a[i1,j]=99 then begin
        vyh:=true;
        goto lb1;
       end else
       if a[i1,j]=1 then begin
        if ((i1<>i-1) and (a[i1+1,j]=0)) thena[i1+1,j]:=q+1;
        i1:=1;
       end;
      end;
     end;
```

```
    for j1:=j+1 to m do begin
     if a[i,j1]<>0 then begin
      if a[i,j1]=99 then begin
       vyh:=true;
       goto lb1;
      end else
      if a[i,j1]=1 then begin
       if ((j1<>j+1) and (a[i,j1-1]=0)) then a[i,j1-1]:=q+1;
       j1:=m;
      end;
     end;
    end;
    for j1:=j-1 downto 1 do begin
     if a[i,j1]<>0 then begin
      if a[i,j1]=99 then begin
       vyh:=true;
       goto lb1;
      end else
      if a[i,j1]=1 then begin
       if ((j1<>j-1) and (a[i,j1+1]=0)) then a[i,j1+1]:=q+1;
       j1:=1;
      end;
     end;
    end;
   end;
  end;
lb1:
  inc(q);
 end;
 hod:=q;
end;
procedure done;
label lb2,lb3;
var
 w:boolean;
 q,i,j,i1,j1:byte;
 f:text;
begin
 assign(f,'ans'+z+'.txt');
 rewrite(f);
 if not vyh then begin
  writeln(f,'Нет решения');
  close(f);
  halt(1);
 end;
 for i:=1 to n do begin
  for j:=1 to m do begin
   if a[i,j]=hod then a[i,j]:=0;
```

```
 end;
end;
for i:=1 to n do begin
 for j:=1 to m do begin
  if a[i,j]=99 then a[i,j]:=hod;
 end;
end;
q:=hod-1;
while q>=3 do begin
 w:=false;
 for i:=1 to n do begin
  for j:=1 to m do begin
   if a[i,j]=q then begin
    if w then begin
     a[i,j]:=0;
     goto lb2;
    end;
    for i1:=i+1 to n do begin
     if a[i1,j]<>0 then begin
      if a[i1,j]=q+1 then begin
       w:=true;
      end;
      i1:=n;
     end;
    end;
    for i1:=i-1 downto 1 do begin
if a[i1,j]<>0 then begin
      if a[i1,j]=q+1 then begin
       w:=true;
      end;
      i1:=1;
     end;
    end;
    for j1:=j+1 to m do begin
     if a[i,j1]<>0 then begin
      if a[i,j1]=q+1 then begin
       w:=true;
      end;
      j1:=m;
     end;
    end;
    for j1:=j-1 downto 1 do begin
     if a[i,j1]<>0 then begin
      if a[i,j1]=q+1 then begin
       w:=true;
      end;
      j1:=1;
     end;
    end;
```

```
     if not w then a[i,j]:=0;
    end;
lb2:
   end;
  end;
  dec(q);
 end;
 writeln(f,hod-2);
 for q:=1 to hod-2 do begin
  for i:=1 to n do begin
   for j:=1 to m do begin
    if a[i,j]=q+2 then begin
     otv[q,2]:=i;
     otv[q,3]:=j;
     goto lb3;
    end;
   end;
  end;
lb3:
 end;
 for q:=1 to hod-2 do begin
  if otv[q,2]<otv[q-1,2] then otv[q,1]:=1;
  if otv[q,2]>otv[q-1,2] then otv[q,1]:=3;
  if otv[q,3]<otv[q-1,3] then otv[q,1]:=4;
  if otv[q,3]>otv[q-1,3] then otv[q,1]:=2;
 end;
 for q:=1 to hod-2 do begin
  writeln(f,otv[q,1],',',otv[q,2],',',otv[q,3]);
 end;
 close(f);
end;

BEGIN
 init;
 run;
 done;
END.
```

16. Ловись рыбка большая и маленькая

```
{$N+}
program fish;
type restrict = record
      x,y,lx,ly:single;
     end;
     NetPoint = record
       x,y:single;
       IsGoodPoint:boolean;
     end;
```

```
var M,N,S1,S2,BigS: single;
    K,i,num,BytesRead: integer;
    KA: array[1..100] of restrict;
    f: text;
    f1,f2: file;
    TempStr: array[1..1000] of char;
    numstr: string;
   NetArr: array[1..202,1..202] of boolean;
    INetArr,JNetArr: array[1..202] of single;
    MaxI,MaxJ: integer;
function FindS1:single;
var
  i,j: integer;
  x: single;
begin
  x:=0;
  for i:=1 to MaxI-1 do
    for j:=1 to MaxJ-1 do
      if NetArr[i,j]=true then
        x:=x+(INetArr[i+1]-INetArr[i])*(JNetArr[j+1]-JNetArr[j]);
  FindS1:=x;
end;
procedure CheckPoint(i,j:integer);
begin
  NetArr[i,j]:=false;
  BigS:=BigS+(INetArr[i+1]-INetArr[i])*(JNetArr[j+1]-JNetArr[j]);
  if ((j-1) in [1..MaxJ-1]) and NetArr[i,j-1] then
    CheckPoint(i,j-1);
  if ((i+1) in [1..MaxI-1]) and NetArr[i+1,j] then
    CheckPoint(i+1,j);
  if ((j+1) in [1..MaxJ-1]) and NetArr[i,j+1] then
    CheckPoint(i,j+1);
  if ((i-1) in [1..MaxI-1]) and NetArr[i-1,j] then
    CheckPoint(i-1,j);
end;
function FindS2:single;
begin
  BigS:=0;
  if NetArr[1,1] then
    CheckPoint(1,1);
  FindS2:=BigS;
end;
procedure InsertVerLine(x:single;a:integer);
var i,j: integer;
begin
  for i:=MaxI downto a do
    INetArr[i+1]:=INetArr[i];
  INetArr[a]:=x;
  for i:=MaxI-1 downto a-1 do
```

```
    for j:=MaxJ-1 downto 1 do
    NetArr[i+1,j]:=NetArr[i,j];
  MaxI:=MaxI+1;
end;
procedure InsertHorLine(y:real;a:integer);
var i,j: integer;
begin
  for j:=MaxJ downto a do
    JNetArr[j+1]:=JNetArr[j];
  JNetArr[a]:=y;
  for i:=MaxI-1 downto 1 do
    for j:=MaxJ-1 downto a-1 do
    NetArr[i,j+1]:=NetArr[i,j];
  MaxJ:=MaxJ+1;
end;
procedure InsertRect(x,y,lx,ly:real);
var
  i,j,X1Index,X2Index,Y1Index,Y2Index: integer;
begin
  i:=1;
  while (x>INetArr[i]) do  i:=i+1;
  if x<>INetArr[i] then
    InsertVerLine(x,i);
  X1Index:=i;
  while (x+lx>INetArr[i]) do  i:=i+1;
  if x+lx<>INetArr[i] then
    InsertVerLine(x+lx,i);
  X2Index:=i;
  j:=1;
  while (y>JNetArr[j]) do  j:=j+1;
  if y<>JNetArr[j] then
    InsertHorLine(y,j);
  Y1Index:=j;
  while (y+ly>JNetArr[j]) do  j:=j+1;
  if y+ly<>JNetArr[j] then
    InsertHorLine(y+ly,j);
  Y2Index:=j;
  for i:=X1Index to X2Index-1 do
    for j:=Y1Index to Y2Index-1 do
      NetArr[i,j]:=false;
end;
begin (* Main Program *)
{ initialization }
  write("Enter test N:"); readln(num);
  Str(num,numstr);
  assign(f1,'1test'+numstr+'.txt');
  reset(f1,1);
  assign(f2,'mytemp.tmp');
  rewrite(f2,1);
```

```
  while not(eof(f1)) do
    begin
      BlockRead(f1,TempStr,1000*SizeOf(Char),BytesRead);
      for i:=1 to BytesRead do
        if TempStr[i]=':' then
          TempStr[i]:=' ';
      BlockWrite(f2,TempStr,BytesRead);
    end;
  close(f1);
  close(f2);
  assign(f,'mytemp.tmp');
  reset(f);
  readln(f,M,N,K);
  for i:=1 to K do
    begin
      readln(f,KA[i].x,KA[i].y,KA[i].lx,KA[i].ly);
      KA[i].x:=KA[i].x-(KA[i].lx/2);
      KA[i].y:=KA[i].y-(KA[i].ly/2);
    end;
  close(f);
  MaxI:=2; MaxJ:=2;
  INetArr[1]:=0; INetArr[2]:=M;
  JNetArr[1]:=0; JNetArr[2]:=N;
  NetArr[1,1]:=true;
{ run }
  for i:=1 to K do
    InsertRect(KA[i].x,KA[i].y,KA[i].lx,KA[i].ly);
  S1:=FindS1;
  S2:=FindS2;
{ finalization }
  assign(f,'1ans'+numstr+'.txt');
  rewrite(f);
  writeln(f,S1);
  writeln(f,S2);
  close(f);
end.
```

17. Лоскутное одеяло

```
{$N+}
program N26062;
var M,N,i: byte;
    ErrCode: integer;
    f: text;
    S,S1: string;
    x: comp;
function f1(x:word):word;
begin
  if Odd(x) then
```

```
    f1:=x+1
          else
    f1:=x;
end;
function fstepen(a:extended;b:integer):comp;
var x: comp;
    i: integer;
begin
  while (b mod 2 = 0) and (b<>0) do
    begin
      a:=a*a;
      b:=b div 2;
    end;
  x:=1;
  for i:=1 to b do
    x:=x*a;
  fstepen:=x;
end;
{ main program }
{ initialization }
begin
  write("Enter test number: "); readln(S1);
  assign(f,'2test'+S1+'.txt');
  reset(f);
readln(f,S);
  close(f);
  i:=1;
  while (i<=length(S)) and (S[i]<>':') do
    i:=i+1;
  Val(Copy(S,1,i-1),M,ErrCode);
  Val(Copy(S,i+1,length(S)-i),N,ErrCode);
  if (not(M in [1..20])) or (not(M in [1..20])) then
    begin
      writeln("Error!");
      halt;
    end;
  assign(f,'2ans'+S1+'.txt');
  rewrite(f);
{ run }
  x:=(fstepen(2,M*N)+fstepen(2,f1(M*N) div 2)+fstepen(2,(f1(M)*N) div 2)+fstepen(2,(f1(N)*M) div 2))/4;
{ finalization }
  writeln(f,x);
  close(f);
end.
end.
```

18. Перелом

```
Program Perelom;
```

```
Const
  InFile =’Atest.txt’;
  OutFile =’ans.txt’;
  MaxXod        =41;
Var
  N,M,Step :Byte;
  FinI,FinJ,MaxSpeed,MaxStep :Integer;
  Ex :Boolean;
  A :Array [1..MaxXod,1..20,1..79]Of Byte;

Procedure Make(Var Speed,NumMax:Integer;I,J:Integer);
Begin
  Speed:= A[Step,I,J] mod 10;
  NumMax:=A[Step,I,J] div 10;
  If A[Step,I,J]=204 Then
   Begin
     Speed:=0;
     NumMax:=0;
   End;
End;

Procedure OutPutNo;
Var
  F :Text;
Begin
  Assign(F,OutFile);
  ReWrite(F);
  WriteLn(F,’Нет решения !’);
  Close(F);
  Halt(0);
End;

Procedure Recurs(I,J,Speed,NumMax,St,OldI,OldJ : Integer);
Var
  I1,J1,L :Integer;
Begin
  If Step>40 Then OutPutNo;
  If (NumMax>MaxStep)or
    (St>Speed+1) Then Exit;
  If A[1,I,J]=203 Then
    Begin
      Ex:=True;
      A[Step+1,I,J]:=NumMax*10+St;
      FinI:=I;
      FinJ:=J;
    End;
  If Ex then Exit;
  If (Abs(St-Speed)<2)And(St<>0)And((A[1,I,J]<200)or(A[1,I,J]=204)) Then
    Begin
```

```
      If St=MaxSpeed Then
        A[Step+1,I,J]:=(NumMax+1)*10+St
      Else
Case A[1,I,J] Of
    201 : Write(F,'*');
    202 : Write(F,'X');
    203 : Write(F,'0');
    0   : Write(F,' ');
    204 : Write(F,'+');
   End;
 End;
 For Step:=2 to St do
 Begin
  WriteLn(F,'');
  For I:=1 to N do
  Begin
   Writeln(F,'');
   For J:=1 to M do
    Case A[1,I,J] Of
     201 : Write(F,'*');
     202 : Write(F,'X');
     203 : Write(F,'0');
     0  : If (A[Step,I,J]=Step-1)And(Step<>1)
          Then Write(F,'s')
          Else Write(F,' ');
     204: If (A[Step,I,J]=Step-1)And(Step<>1)
          Then Write(F,'s')
          Else Write(F,'+');
    End;
  End;
 End;
 WriteLn(F,' ');
 For I:=1 to N do
 Begin
   Writeln(F,'');
   For J:=1 to M do
    Case A[1,I,J] Of
     201 : Write(F,'*');
     202 : Write(F,'X');
     203 : If A[St+1,I,J]<>0
          Then Write(F,'s')
          Else Write(F,'0');
     0   : Write(F,' ');
     204 : Write(F,'+');
    End;
 End;
 Close(F);
End;
```

```
Procedure Init;
Var
 F            : Text;
 S,S1         : String;
 Code         : Integer;
 I,J          : Integer;
Begin
 Assign(F,InFile);
 Reset(F);
 Readln(F,S);
 S1:=Copy(S,1,Pos(“,”,S)-1);
 Delete(S,1,Pos(“,”,S));
 Val(S1,N,Code);
 Val(S,M,Code);
 For I:=1 to N do
 Begin
  ReadLn(F,S);
  For J:=1 to M do
   Case S[J] of
    “1” :A[1,I,J]:=201;
    “2” :A[1,I,J]:=202;
    “3” :A[1,I,J]:=203;
    “4” :A[1,I,J]:=204;
   End;
 End;
 ReadLn(F,S);
 S1:=Copy(S,1,Pos(“,”,S)-1);
 Delete(S,1,Pos(“,”,S));
 Val(S1,MaxSpeed,Code);
 Val(S,MaxStep,Code);
 Close(F);
End;

Procedure NextStep;
Var
 I,J          : Integer;
Speed,NumMax   : Integer;
Begin
 Inc(Step);
 For I:=1 to N do
  For J:=1 to M do
   If (A[Step,I,J]>0)And(A[Step,I,J]<200)
      or(A[Step,I,J]=204) Then
   Begin
    Make(Speed,NumMax,I,J);
    Recurs(I,J,Speed,NumMax,0,I,J);
    If Ex then Exit;
   End;
End;
```

```
Procedure Done;
Begin
 Step:=0;
 Ex  :=False;
 Repeat
  NextStep;
 Until Ex  ;
End;

Begin
 Init;
 Done;
 MainDone;
End.
```

19. Поиск месторождения

```
{$I-,Q-,R-,S-,D-,L-}
{I+,Q+,R+,S+,D+,L+}
program ProblemFindingTheMaxConcentrationOrJustUsingBinarySearch;
{set Tab space = 2}
uses
        anal;
const
        maxXk = 1000002;

var
        xx : array [0..maxXk] of longint;
  result, xr, xl : longint;

procedure init;
begin
  xl := 0;
  xr := start;
  fillchar(xx, sizeof(xx), 255)
end;

function  look2(x : longint) : longint;
begin
  if xx[x] = -1 then begin
        xx[x] := look(x);
    look2 := xx[x];
  end else look2 := xx[x];
end;

procedure solve;
var
  maxF, x, t, t1 : longint;
```

```
begin
  while xr - xl > 1 do begin
          x := (xr + xl + 1) shr 1;
          t := look2(x);
    if (succ(x) <= xr)and(xx[succ(x)] <> -1)then begin
          t1 := look2(succ(x));
      if t > t1 then xr := x else xl := x;
    end else
    if (pred(x) <= xr)and(xx[pred(x)] <> -1)then begin
          t1 := look2(pred(x));
      if t > t1 then xl := x else xr := x;
    end else
    if succ(x) <= xr then begin
          t1 := look2(succ(x));
      if t > t1 then xr := x else xl := x;
    end else begin
          t1 := look2(pred(x));
if t > t1 then xl := x else xr := x;
    end;
  end;
  x := (xl + xr + 1) shr 1;
  maxF := look2(x);
  result := x;
  if look2(xl) > maxF then begin
          maxF := look2(xl);
    result := xl;
  end;
  if look2(xr) > maxF then begin
          maxF := look2(xr);
    result := xr;
  end;
end;

procedure done;
begin
          finish(result);
end;

begin
          init;
  solve;
  done;
end.
```

21. Терминалы

```
{$I-,Q-,R-,S-,D-,L-}
{I+,Q+,R+,S+,D+,L+}
program ProblemTerminalsOrGreedyAlgorithms;
```

```
{set tab space = 2}
const
  inf = “term.in”;
  ouf = “term.out”;
  maxn = 1002;

type
        tpoint = record
    x, y, z : integer;
  end;
  tmas = array [0..maxn] of integer;

var
  a : array [0..maxn] of tpoint;
  x, y, z : tmas;
  ans : tpoint;
  n, t : integer;
  minr : longint;

procedure init;
begin
        minr := maxlongint div 2 + 1;
end;

procedure readdata;
begin
 assign(input, inf);
  reset(input);
  while not seekeof do begin
        inc(n);
    read(a[n].x, a[n].y, a[n].z);
    x[n] := a[n].x;
    y[n] := a[n].y;
    z[n] := a[n].z;
  end;
  close(input);
end;

function  dist(a, b : tpoint) : longint;
begin
        dist := abs(longint(a.x) - b.x) + abs(longint(a.y) - b.y) + abs(longint(a.z) - b.z);
end;

function  getsum(p : tpoint) : longint;
var
  s : longint;
  i : integer;
begin
        s := 0;
```

```
for i := 1 to n do inc(s, dist(p, a[i]));
  getsum := s;
end;

procedure qsort(var a : tmas);

        procedure sort(l, r : integer);
        var
        i, j : integer;
          x : integer;
        begin
                i := l;
          j := r;
          x := a[(l + r) div 2];
          repeat
                while x > a[i] do inc(i);
            while x < a[j] do dec(j);
            if i <= j then begin
                t := a[i];
              a[i] := a[j];
              a[j] := t;
              inc(i);
              dec(j);
            end;
          until i > j;
          if l < j then sort(l, j);
          if i < r then sort(i, r);
        end;

begin
        sort(1, n);
end;

procedure math;
var
        i : integer;
  cc : tpoint;
  sum : longint;
begin
        qsort(x);
  qsort(y);
  qsort(z);
  for i := 0 to 1 do begin
        cc.x := x[(n + i)shr 1];
        cc.y := y[(n + i)shr 1];
        cc.z := z[(n + i)shr 1];
        sum := getsum(cc);
    if sum < minr then begin
                minr := sum;
```

```
                ans := cc;
        end;
  end;
end;

procedure solve;
begin
  math;
end;

procedure done;
begin
        assign(output, ouf);
  rewrite(output);
  writeln(minr, " ", ans.x, " ", ans.y, " ", ans.z);
  close(output);
end;

begin
        init;
  readdata;
  solve;
  done;
end.
```

22. Ходом коня

```
{$I-,Q-,R-,S-,D-,L-}
{I+,Q+,R+,S+,D+,L+}
{$H-}
program TrickyAndNiceDinamicWithQueueSolutionOfChessProblem;
{set tab space = 2}
const
  inf = "chess.in";
  ouf = "chess.out";
  maxq = 256 * 64;
  dd : array [1..8, 1..2] of shortint = ((2,1),(2,-1),(1,-2),(-1,-2),(-2,-1),
  (-2,1),(-1,2),(1,2));

type
        tint = integer;
  tboard = array [0..9, 0..9] of integer;
  tfig = record
    i, j : shortint;
  end;
  tel = record
        len : integer;
    i, j : shortint;
```

```
    state : byte;
  end;
  tused = array [1..8] of boolean;

var
        d : array [1..8, 1..8, 0..1 shl 8] of integer;
        que : array [0..maxq] of tel;
  p : array [1..8] of tfig;
  def : tboard;
  u : tused;
  si, sj, n : shortint;
  result : integer;
  issol : boolean;
        ql, qr : longint;

procedure init;
var
        i : byte;
begin
        for i := 0 to 9 do begin
        def[i, 0] := -2;
    def[i, 9] := -2;
    def[0, i] := -2;
    def[9, i] := -2;
  end;
  issol := false;
  result := maxint div 2 + 1;
  fillchar(d, sizeof(d), 255);
end;

procedure readdata;
var
  i : shortint;
  r : char;
  s : string;
begin
        assign(input, inf);
  reset(input);
  read(r);
  sj := succ(ord(upcase(r)) - ord("A"));
  readln(si);
  readln(s);
  s := s + " ";
  i := pos(" ", s);
  while i <> 0 do begin
        inc(n);
    p[n].j := succ(ord(upcase(s[1])) - ord("A"));
    p[n].i := ord(s[2]) - 48;
    delete(s, 1, 3);
```

```
        i := pos(" ", s);
  end;
  close(input);
end;

procedure setp(var a : tboard; ind, i, j : tint);
begin
        if a[i, j] = 0 then a[i, j] := ind;
  if i - 1 > 0 then begin
        if j - 1 > 0 then a[i - 1, j - 1] := 9;
    if j + 1 < 9 then a[i - 1, j + 1] := 9;
  end;
end;

function  getstate(u : tused) : byte;
var
        k, state : byte;
begin
        state := 0;
        for k := 1 to n do if u[k] then state := state or (1 shl pred(k));
  getstate := state;
end;

procedure addq(ii, jj : tint; sta : byte; ll : integer);
begin
        inc(qr);
  with que[qr mod maxq] do begin
        i := ii;
    j := jj;
    state := sta;
    len := ll;
  end;
end;

procedure extract(var ii, jj : tint; var ll : integer; var a : tboard; var u : tused);
var
        k : tint;
begin
        a := def;
  fillchar(u, sizeof(u), false);
  with que[ql mod maxq] do begin
        ll := len;
        ii := i;
    jj := j;
    for k := 1 to 8 do if state and (1 shl pred(k)) <> 0 then begin
        setp(a, k, p[k].i, p[k].j);
      u[k] := true;
    end;
  end;
```

```
end;

procedure solve;
var
          clen, ci, cj, l, ni, nj : integer;
  news, curs : byte;
  a : tboard;
begin
  fillchar(a, sizeof(a), 0);
  fillchar(u, sizeof(u), true);
  ql := 0;
  qr := 0;
  ci := si;
  cj := sj;
  clen := 0;
  for l := 1 to n do setp(a, l, p[l].i, p[l].j);
  repeat
    curs := getstate(u);
    if curs = 0 then begin
          issol := true;
      result := clen;
      continue;
    end;
    for l := 1 to 8 do begin
          ni := ci + dd[l, 1];
      nj := cj + dd[l, 2];
      if not((ni >= 1)and(ni <= 8)and(nj >= 1)and(nj <= 8)) then continue;
      case a[ni, nj] of
          0 : begin
          if(d[ni, nj, curs] = -1)or(d[ni, nj, curs] > succ(clen)) then begin
          d[ni, nj, curs] := succ(clen);
          addq(ni, nj, curs, succ(clen));
         end;
        end;
        1..8 : begin
          u[a[ni, nj]] := false;
         news := getstate(u);
          if(d[ni, nj, news] = -1)or(d[ni, nj, news] > succ(clen)) then begin
          d[ni, nj, news] := succ(clen);
          addq(ni, nj, news, succ(clen));
         end;
u[a[ni, nj]] := true;
       end;
      end;
    end;
    inc(ql);
    extract(ci, cj, clen, a, u);
  until (ql > qr) or (issol);
end;
```

```
procedure done;
begin
        assign(output, ouf);
  rewrite(output);
  if issol then writeln(result) else writeln("NO");
  close(output);
end;

begin
        init;
  readdata;
  solve;
  done;
end.
```

23. Рубины Басры

```
Program Rubin;
Const
inf='rubin.in';
outf='rubin.out';
Var
        i,j: integer;
        A: array [1..1000];
begin
assign(input, inf);
  reset(input);
  while not seekeof do
        begin
        read(i);
        inc(a[i]);
        end;
close (input);

assign(output, outf);
reset(output);
for i:=1 to 1000 do
        for j:=1 to a[i] do
                writeln (i);
close (output);
end;
```

Районные олимпиады

Program z1;

```
var Minind,Maxind,k,Min,Max,i,S:longint;
   f  :text;

procedure init;
```

```
begin
i:=0;
assign(f,'pos.in');
reset(f);
read(f,k);
inc(i);
Max:=k;
Min:=k;
Maxind:=i;
Minind:=i;
while not Eof(f) do
 begin
    read(f,k);
    inc(i);
    if Min>k then
       begin
         Minind:=i;
         Min:=k;
       end;
     if Max<k then
       begin
         Maxind:=i;
         Max:=k;
       end;
 end;
 close(f);
end;

Procedure work;
begin
 If Minind>Maxind then
          begin
            k:=Minind;
            Minind:=Maxind;
            Maxind:=k;
          end;
assign(f,'pos.in');
reset(f);
for I:=1 to Minind-1 do
   read(f,k);
S:=0;
For I:=Minind to MaxInd do
   begin
   read(f,k);
   S:=S+k;
   end;
close(f);
end;
```

```
Procedure exi;
begin
assign(f,'pos.out');
rewrite(f);
write(f,S);
close(f);
end;

Begin
    init;
    work;
    exi;
end.
```

Program z2;

```
var k,M,a,b,rez:longint;

procedure init;
begin
write("input k -");
readln(k);
write("input m -");
readln(m);
end;

Procedure work;
begin
B:=1;
while b<=(m-k+1) do
     begin
     a:=b;
     b:=b*2;
     end;
   rez:=a+k-1;
end;

Procedure exi;
begin
 writeln(rez);
end;

Begin
    init;
    work;
    exi;
end.
```

Program z3;

```
Var
 i,Q,W,N,M,Cnt,E,C      : Longint;
{————————————————————————————————————————————}
 Procedure Load;
 Begin
  Write(“a =”);
  ReadLn(N);
  Write(“b =”);
  ReadLn(M);
 End;
{————————————————————————————————————————————}
 Procedure Rec(X,Y:Longint);
 Begin
  If X=Y Then Begin Inc(Cnt);Exit;End Else Begin
    If X>Y Then Begin E:=X;X:=Y;Y:=E;End;
    Inc(Cnt);
    Rec(X,Y-X);
    Exit;
   End;
 End;
{————————————————————————————————————————————}
 Procedure Make;
 Begin
  Cnt:=0;i:=1;
  While i<10 Do Begin
   Inc(i);
   While (N mod i=0)and(M mod i=0) Do Begin
     N:=N div i;
     M:=M div i;
    End;
  End;
  If N>M Then Begin E:=N;N:=M;M:=E;End;
  Q:=N div 2;W:=N-Q;
  Rec(Q,M);
  Rec(W,M);C:=Cnt;Cnt:=0;
Rec(N,M);
  If Cnt>C Then Cnt:=C;
 End;
{————————————————————————————————————————————}
 Procedure Save;
 Var F : Text;
 Begin
  WriteLn(“k =”,Cnt);
 End;
{————————————————————————————————————————————}
Begin
 Load;
 Make;
 Save;
```

End.

Program z4;

```
var i,n,k:longint;
   f: text;

procedure init;
begin
write(“ N -”);
readln(n);
end;

Procedure work;
begin
assign(f,’posled.out’);
rewrite(f);
K:=0;
for i:=1 to n do
   begin
   if (i mod 2)=1 then
      begin
      inc(k);
       write(f,k,’ “);
      end
      else
       write(f,n-k+1,’ “);
   end;
close(f);
end;

Procedure exi;
begin
end;

Begin
    init;
    work;
    exi;
end.
```

Program z5;

```
var k,Min,Max,j,i,S:longint;
   Chisla      :array [1..1000] of integer;
   Flag        :boolean;
   f   :text;

procedure init;
```

```
begin
assign(f,'pos.in');
reset(f);
read(f,k);
Max:=k;
Min:=k;
while not Eof(f) do
 begin
    read(f,k);
     if Min>k then
        Min:=k;
     if Max<k then
        Max:=k;
 end;
 close(f);
end;

Procedure work;
begin
i:=0;
assign(f,'pos.in');
reset(f);
while not Eof(f) do
 begin
    read(f,k);
    if (k>Min) and (k<Max) then
     begin
     Flag:=False;
     for j:=1 to i do
        if Chisla[j]=k then Flag:=True;
     if Flag=False then
        begin
         inc(i);
         Chisla[i]:=k;
         S:=i;
        end;
    end;
 end;
 close(f);
end;

Procedure exi;
begin
assign(f,'pos.out');
rewrite(f);
for i:=Min+1 to Max-1 do
   begin
   Flag:=false;
   for j:=1 to S do
```

```
    if i=Chisla[j] then Flag:=true;
   if Flag=False then writeln(f,i);
   end;
close(f);
end;

Begin
    init;
    work;
    exi;
end.

program z6;

var i,n,k:longint;
   f: text;
procedure init;
begin
write(“ N -”);
readln(n);
end;
Procedure work;
begin
assign(f,’posled.out’);
rewrite(f);
K:=0;
for i:=1 to n do
   begin
   if (i mod 2)=1 then
      begin
      inc(k);
       write(f,k,’ “);
      end
      else
       write(f,n-k+1,’ “);
   end;
close(f);
end;
Procedure exi;
begin
end;

Begin
    init;
    work;
    exi;
end.
```

Program z7;

```
Var
 F      : Text;
 A      : Array [0..13000] Of Longint;
Num,E,i,j,n,max,m : Integer;
Begin
 Assign(f,'pOS.iN');
 Reset(F);
 FillChar(A,siZeoF(A),0);
 While Not Eof(F) Do Begin
   ReadLn(F,Num);If Num>m Then M:=Num;
   Inc(A[Num]);If A[Num]>Max Then Max:=A[Num];
  End;
 Close(F);
 Assign(F,'pos.out');
 ReWrite(F);
 For i:=1 to M Do If A[i]=Max Then WriteLn(F,i);
 Close(F);
End.
```

Program Z8;

```
Var
 F : text;
 i,j,N,M : Byte;
 Cnt : Integer;
 A   : Array [0..202] Of String;
Begin
 Assign(F,'kvadraty.in');
 Reset(F);i:=0;
 While Not Eof(F) Do Begin
   Inc(i);
   ReadLn(F,A[i]);M:=Length(A[i]);
  End;
 Close(F);N:=i;i:=0;
 While (i<=M) Do Begin Inc(i);A[0]:=A[0]+'0';End;
 For i:=0 to N do A[i]:='0'+A[i];
 Cnt:=0;
 For i:=1 to N do
  For j:=2 to M+1 do
   If (A[i,j]='1')and(A[i-1,j]='0')and(A[i,j-1]='0') Then
    Inc(Cnt);
 Assign(F,'kvadraty.Out');
 ReWrite(F);
 WriteLn(F,Cnt);
 Close(F);
End.
```

Program z9;

```
var a,b,c:real;
```

```
   rez:string[3];
procedure init;
begin
write(“input a -”);
readln(a);
write(“input b -”);
readln(b);
write(“input c -”);
readln(c);
end;
Procedure work;
begin
Rez:=’no’;
     if ((a+b)>c) and ((a+c)>b) and ((c+b)>a) then Rez:=’yes’;
end;
Procedure exi;
begin
 writeln(rez);
end;

Begin
     init;
     work;
     exi;
end.
```

Program z10;

```
var m,k,MaxCol,Max,i,Col:longint;
   f   :text;

procedure init;
begin
assign(f,’pos.in’);
eset(f);
read(f,m);
Max:=m;
MaxCol:=1;
Col:=1;
while not Eof(f) do
 begin
    read(f,k);
     if m=k then
       begin
         inc(Col);
           if Col>MaxCol then
              begin
               MaxCol:=Col;
               Max:=k;
```

```
            end;
        end
       else
        Begin
          Col:=1;
          M:=k;
       end;
   end;
 close(f);
end;

Procedure exi;
begin
assign(f,'pos.out');
rewrite(f);
write(f,MaxCol,' ',Max);
close(f);
end;

Begin
     init;
     exi;
end.
```

```
program z11;
var
  j,i,k,m,n :longint;
begin

 readln(n,m);
 readln(k);
 i:=((k-1) div m)+1;
 j:=k-(i-1)*m;
 if i>1 then write(k-m,' ');
 if j<M then write(k+1,' ');
  if i<N then write(k+m,' ');
   if j>1 then write(k-1,' ');
end.

Var
 F : text;
 i,j,N,M,K,L : Byte;
 Max : Integer;
 A   : Array [0..201] Of String;
Begin
 Assign(F,'kvadraty.in');
 Reset(F);i:=0;
```

```
 While Not Eof(F) Do Begin
   Inc(i);
   ReadLn(F,A[i]);M:=Length(A[i]);
  End;
 Close(F);N:=i;i:=0;
 While (i<=M+1) Do Begin Inc(i);A[0]:=A[0]+'0';End;
  While (i<=M+1) Do Begin Inc(i);A[N]:=A[N]+'0';End;
 For i:=0 to N do A[i]:='0'+A[i];
 For i:=0 to N do A[i]:=A[i]+'0';
 Max:=1;
 For i:=1 to N do
  For j:=2 to M+1 do
   If (A[i,j]='1')and(A[i-1,j]='0')and(A[i,j-1]='0') Then
       begin
         K:=0;
         while A[i,j+k]='1' do inc(k);
         L:=0;
while A[i+l,j]='1' do inc(L);
       if Max<L*K then Max:=L*K;
       end;

 Assign(F,'kvadraty.Out');
 ReWrite(F);
 WriteLn(F,Max);
 Close(F);
End.
```

Program z12;

```
var m,k,MaxCol,Max,i,Col:longint;
    f   :text;

procedure init;
begin
assign(f,'pos.in');
reset(f);
read(f,m);
Max:=m;
MaxCol:=1;
Col:=1;
while not Eof(f) do
 begin
    read(f,k);
     if m=k then
       begin
         inc(Col);
           if Col>MaxCol then
              begin
               MaxCol:=Col;
```

```
            Max:=k;
           end;
       end
      else
       Begin
         Col:=1;
         M:=k;
      end;
  end;
 close(f);
end;

Procedure work;
begin
end;

Procedure exi;
begin
assign(f,'pos.out');
rewrite(f);
write(f,MaxCol,' ",Max);
close(f);
end;

Begin
    init;
    work;
    exi;
end.
```

Program z13;

```
var
  Name,Kol      :array[1..15000] of integer;
  i,j,n,k,index,max  :integer;
  f             :text;
  Flag          : boolean;

begin
assign(f,'pos.in');
reset(f);
i:=1;
 readln(f,Name[1]);
Kol[1]:=1;

while not eof(f) do
  begin
  readln(f,k);
    Flag:=False;
   for J:=1 to i do
```

```
       if k=Name[j] then
while A[i+l,j]='1' do inc(L);
      if Max<L*K then Max:=L*K;
      end;

 Assign(F,'kvadraty.Out');
 ReWrite(F);
 WriteLn(F,Max);
 Close(F);
End.
```

Program z12;

```
var m,k,MaxCol,Max,i,Col:longint;
   f   :text;

procedure init;
begin
assign(f,'pos.in');
reset(f);
read(f,m);
Max:=m;
MaxCol:=1;
Col:=1;
while not Eof(f) do
 begin
    read(f,k);
     if m=k then
       begin
         inc(Col);
           if Col>MaxCol then
              begin
               MaxCol:=Col;
               Max:=k;
              end;
       end
      else
       Begin
         Col:=1;
         M:=k;
      end;
   end;
 close(f);
end;

Procedure work;
begin
end;
```

```
Procedure exi;
begin
assign(f,'pos.out');
rewrite(f);
write(f,MaxCol,' ",Max);
close(f);
end;

Begin
    init;
    work;
    exi;
end.
```

Program z13;

```
var
  Name,Kol      :array[1..15000] of integer;
  i,j,n,k,index,max  :integer;
  f             :text;
  Flag          : boolean;

begin
assign(f,'pos.in');
reset(f);
i:=1;
 readln(f,Name[1]);
Kol[1]:=1;

while not eof(f) do
  begin
  readln(f,k);
    Flag:=False;
   for J:=1 to i do
      if k=Name[j] then
for i:=1 to n do
    if (i mod 2)=0 then write(f,Stroka[i]) else write(f,Stroka[n-i+1]);
close(f);

end.
```

Program z16;

```
var
  x,y       :array[1..201] of real;
  i,n           : byte;
  S           : real;
  f           :text;
begin
S:=0;
assign(f,'mnogoyg.in');
```

```
reset(f);
i:=0;
while not eof(f) do
   begin
   inc(i);
   readln(f,x[i],y[i]);
   end;
close(f);
n:=i;
x[n+1]:=x[1];
y[n+1]:=y[1];

for i:=1 to n do
   S:=s+(x[i+1]-x[i])*(y[i+1]+y[i])/2;

writeln(s);
end.
```

Часть II. Подготовка к олимпиаде

ВВЕДЕНИЕ

Вторая часть книги “Олимпиады по информатике” посвящена вопросам, связанным с подготовкой учащихся к соревнованиям.

Готовя участника к олимпиаде, необходимо помнить о триединстве основных компонент:

- тренировка логического мышления;
- работа над техникой реализации;
- психологическая подготовка.

Я считаю, что основное при подготовке ребят - необходимость выработки у них логически-структурного мышления: умение выбрать из условия задачи главное; разбить основную задачу на подзадачи; определить, к какому классу можно их отнести; выбрать наилучший метод (алгоритм) решения.

Данная книга не является учебником по алгоритмизации в полном понимании, вопросы технической подготовки рассматриваются лишь в той мере, в какой они могут быть использованы при решении задач, обычно присутствующих на олимпиадах по информатике. Ряд методов и алгоритмов решения задач рассмотрены менее подробно, другие - более. Это, в основном, связано с анализом работ наших ребят, трудностей, с которыми они сталкиваются, и ошибок, которые они допускают при решении олимпиадных задач.

В процессе психологической подготовки необходимо настроить участника на желание и умение **максимально реализовать свои потенциальные возможности** (заметьте - **не победить**, а максимально реализовать) в экстремальных условиях, которые предъявляет олимпиада.

2.1 Техническая подготовка

2.1.1 О структурном программировании

В чем преимущества структурного подхода к программированию?

1. Можно ожидать, что логика работы программы будет более понятна программирующему.
2. Если Вы пока не знаете, как реализовать какой-либо модуль, всегда можно поставить “умозрительную” машину, которая ничего не делает.
3. Возможность “работать” (программировать) с отдельным модулем.
4. Независимость “умозрительных” машин одного уровня позволяет независимо модифицировать блоки программы без затрагивания других блоков. Как отмечал Э. Дейкстра: “…модификации программ могут быть теперь представлены заменой одной (умозрительной) машины другой подходящей машиной”.
5. Проведение отладки структурированных программ намного легче, чем не структурированных.
6. Можно ожидать, что это оградит нас от ненужной работы. Если нет заварки, наша программа просто завершит работу.
7. “Структурный подход” позволяет проводить целенаправленную подготовку к участию в олимпиадах по информатике. Если модули независимы друг от друга, то можно заранее подготовить машины нижнего уровня, например, блок сортировки, блок поиска в глубину, поиска в ширину, блок обработки входной информации. Отработав их в подготовительный период, на олимпиаде, исходя из задачи, использовать в том или ином сочетании.

Небольшое замечание. Часто, сравнивая языки программирования, говорят, что Паскаль – структурированный язык. А, например, Бейсик – нет. Что один язык лучше другого. Хочется напомнить таким спорщикам следующее: каждый язык программирования наиболее удобен для опре-

деленного круга задач. Например, для задач, связанных с искусственным интеллектом, возможно, более подходит язык Пролог, для задач, связанных с вычислениями, лучше Фортрана вряд ли что можно предложить и т.д. Помните! На “структурированном” Паскале можно написать «неструктурированные» программы. А на “неструктурированном” Бейсике – хорошо «структурированную» программу.

Иногда, сравнивая структурное программирование с объектно-ориентированным и визуальным программированием, представляют последние как более высокую ступень программирования. Это не так. Объектно-ориентированное программирование, как и визуальное программирование, это лишь способы программирования, тогда как структурное программирование – это методология и идеология программирования.

2.1.2 Что день грядущий нам готовит, или о классификации олимпиадных задач.

“Задача очень простая и относится к 1-6 классу школы”

Из ответа на вступительных экзаменах в РЗШ

Часто мне задают вопрос: “Какие задачи встречаются на олимпиадах?”. Что можно об этом сказать? В настоящее время, как правило, задачи включают в себя несколько тем и редко они проявляются в чистом виде. По основной теме, заложенной в задаче, их условно делят на следующие группы:

- вычислительные (арифметические, геометрические);
- связанные с графами;
- сетевого планирования и теории расписаний;
- комбинаторные и вероятностные;
- эвристические и задачи на стратегию;
- линейного программирования;
- связанные с криптологией т.д.

Иногда задачи классифицируют по признаку основных методов, применяемых для их решения:

- арифметические;
- геометрические;
- комбинаторные;
- бинарный поиск;
- сортировка;
- рекурсивные
- поиск в ширину;
- поиск в глубину с возвратом и т.п.

Я, как правило, делю задачи на **вычислительные**, **переборные и все остальные (эвристические).**

Вычислительные задачи - это задачи, связанные с вычислениями по определенным формулам. Например: нахождение точки пересечения прямых, определение площади фигур, нахождение числовой зависимости и т.п. .

Переборные задачи. Многие задачи можно отнести к задачам переборного типа. В самом деле, во многих задачах нам необходимо выбрать из множества возможных вариантов один или несколько вариантов, которые удовлетворяют условию задачи. При этом наиболее полным решением, очевидно, является просмотр всех вариантов и выбор из них нас интересующих. Однако в подавляющем ряде случаев полный перебор занимает такое время, что задача не может быть решена в приемлемое время. Для сокращения времени перебора необходимо каким-то образом ограничить исходное множе-

ство вариантов просмотра. Иными словами, необходимо из множества А (исходное множество вариантов просмотра) выбрать такое подмножество В (рассматриваемых вариантов), что бы В было много меньше множества А.

Например:

1. В задаче со сферой (часть 1) уменьшение числа просматриваемых точек достигается путем:

а) рассмотрения только одного из квадрантов (учет симметрии сферы);

б) рассмотрения точек, лежащих только вблизи границы сферы (если точка с максимальным значением координаты принадлежит сфере, то все точки с меньшей координатой также принадлежат сфере, более того, значение координаты - количество точек, принадлежащих сфере).

2. В задаче "Ручей Моно" (см. часть 1) уменьшение числа просматриваемых участников можно достичь путем отсева тех из них, которые заведомо не смогут достичь финиша.

3. В ряде задач, таких как "Покорение вершины"(3-я Всемирная олимпиада по информатике, Греция, 1991г.) или "Ручей Моно", поиск оптимального решения ведется путем просмотра вариантов, начиная с наилучших к худшим. При получении на каком-либо из этапов решения, которое хуже предыдущего, дальнейшее рассмотрение прекращается. Зачем рассматривать заведомо проигрышные варианты?

4. Первоначально выбирают область, близкую к оптимальному решению. Затем рассматривают варианты решения, близкие к оптимальному.

2.1.3 Алгоритм перебора вариантов в глубину с возвратом

Случай первый. *Представим, что мы находимся внутри лабиринта и нам необходимо исследовать его, т.е. полностью обойти лабиринт*.

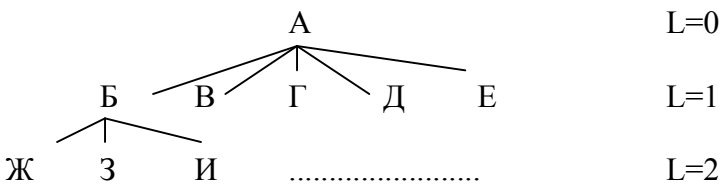
Случай второй. *Знаменитая задача о восьми ферзях* на шахматной доске.

Случай третий. *Дана дорожная сеть между городами. Определить всевозможные пути движения из одного города в другой.*

Эти задачи относятся к задачам переборного типа. В них необходимо определить все возможные варианты, удовлетворяющие условию задачи. Для решения такого класса задач можно использовать алгоритм перебора вариантов в глубину с возвратом.

Суть его заключается в следующем:

Просмотр начинается из начального состояния (вход в лабиринт, пустая доска, начальный город). Для этого состояния определяются возможные варианты, в которые система может перейти (учитывая условия задачи). Если таковых несколько, выбирается один из них. Происходит переход в новое состояние, при этом отмечаем это состояние как просмотренное (гуляя по лабиринту, отмечаем коридоры, в которых Вы побывали, чтобы не заблудиться). Для нового состояния системы, также определяются возможные варианты, в которые она может перейти. В случае, если таких состояний нет, то происходит возврат к предыдущему состоянию.



Попробуем описать данный алгоритм, но сначала введем следующие понятия:

L - глубина (уровень) просмотра;

GL{}- массив возможных вариантов на уровне L.

В начальный момент L=0.

Блок определения GL{}

1. Определяем для данного состояния системы возможные варианты и запишем их в массив GL{}.

2. Переход на блок проверок.

Блок проверок

1. Если GL{}=∅ и L=0, то мы находимся в исходном состоянии, и нет вариантов, которые мы можем просмотреть. Программа переходит на блок завершения работы.

2. Если GL{}=∅ и L≠0, то нет вариантов, которые мы можем просмотреть. Мы должны сделать шаг назад, т.е. вернуться в предыдущее состояние системы.

3. Если GL{}≠∅, то есть варианты, которые мы можем просмотреть. Следовательно, надо сделать шаг вперед.

Блок шаг вперед.

1. Находим первый ненулевой элемент в массиве GL{}, то есть выбирается еще не просмотренный вариант.

2. Увеличиваем L на единицу (L= L+1).

3. Помечаем первый ненулевой элемент в GL-1{} как просмотренный, т.е. обнуляем его.

4. Переход на Блок определения GL{}

Блок шаг назад.

1. Уменьшаем значение L на единицу (L= L -1).

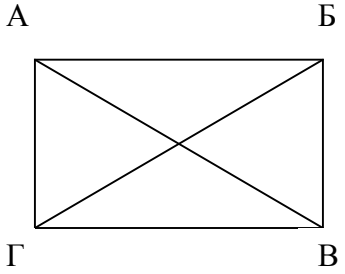
2. Переход на блок проверок.

Блок закончить работу.

Данный блок должен содержать действия по подготовке выходных данных, требуемых по условию задачи.

Попробуем проверить, как будет работать наш алгоритм на примере простой задачи:

Даны четыре города А, Б, В и Г. Они соединены между собой, как показано на рисунке.



Надо определить все возможные пути движения из города А в город В.

БО. Из города А можно попасть в город Г, Б и В. Значит массив - G0{Г,Б,В}

БП. Выполняется условие 3 блока проверок и переход на шаг вперед.

ШВ. L=L+1, G0{0,Б,В}

БО. Из города Г можно попасть в города А, Б и В. Значит массив - G1{А,Б,В}

БП. Выполняется условие 3 блока проверок и переход на шаг вперед.

ШВ. L=L+1, G1{0, Б, В}

БО. Из города А можно попасть в город Г, Б и В. Значит массив G2{Г, Б, В}

Здесь мы прервем трассировку предложенного алгоритма. Обратите внимание, что произошло зацикливание. Что делать? Как обойти эту сложность?

Введем дополнительные массивы: массив N{} - в нем будем хранить путь движения от исходного состояния системы до рассматриваемого и M{} - массив рассмотренных состояний системы.

Внесем изменения в Блоки шаг вперед, шаг назад и введем новый блок - блок обработки цикла.

Блок шаг вперед.

1. Находим первый ненулевой элемент в массиве GL{}, то есть выбирается еще не просмотренный вариант.

2. Определяем, находится ли этот элемент в массиве N{}, если да, то переход на блок обработка цикла.

3. Увеличиваем L на единицу (L= L+1).

4. Помещаем первый ненулевой элемент в GL-1{} в конец массивов N{} и M{}.

5. Помечаем первый ненулевой элемент в GL-1{} как просмотренный, т.е. обнуляем его.

6. Переход на Блок определения GL{}

Блок шаг назад.

1. Уменьшаем значение L на единицу (L= L -1).

2. Удаляем из массива N{} последний ненулевой элемент.

3. Переход на блок проверок.

Блок обработка цикла.

Содержание этого блока зависит от целей программы, в случае с городами нам необходимо просто не попадать в цикл. Тогда действия должны быть следующие:

1. Помечаем первый ненулевой элемент в GL{} как просмотренный, т.е. обнуляем его.

2. Переход на блок проверок.

Однако в ряде случаев в задаче ставится вопрос о нахождении циклов (см. “За золотом на ручей Индианки”), тогда блок обработка цикла будет содержать другие действия.

Попробуйте теперь самостоятельно провести трассировку модифицированной программы для задачи о четырех городах.

Метод трассировки - мощнейший инструмент в достижении понимания логики работы программы, алгоритма.

Для чего нами был введен массив M{}? Если мы имеем систему с несколькими начальными состояниями, то перед рассмотрением каждого нового состояния надо проверить, не находится ли он в M{}. Если да, то зачем нам его рассматривать - мы уже просмотрели его. Тем самым мы сэкономим время и силы.

Примечание: Массив возможных вариантов на уровене L лучше всего организовать как стек, это позволит избежать массовых операций. Обход в этом случае будет вестись от левого края к правому.

2.1.4 Алгоритм перебора вариантов в ширину

В фильме “Волшебная лампа Алладина” есть интересный эпизод. Когда Дух лампы спрашивает у Алладина, что делать с Магрибским колдуном, тот отвечает ему, чтобы колдун шел на все четыре стороны. Дух лампы дословно исполняет пожелание Алладина, и колдун, разделившись на четыре части, идет в четырех направлениях. Попробуем применить этот метод в случае с лабиринтом.

Как и для алгоритма перебора вариантов в глубину с возвратом введем характеристики:

L - глубина (уровень) просмотра;

GL{}- массив возможных вариантов на уровне L.

В начальный момент L=0.

Блок определения GL{}

1. Определяем для данного состояния системы возможные варианты и запишем их в массив GL{}.

2. Помечаем их как просмотренные.

3. Переход на блок проверок.

Блок проверок

1. Если GL{}=□, то нет вариантов, которые мы можем просмотреть. Значит, программа переходит на блок завершения работы.

2. Если GL{}□□, то есть варианты, которые мы можем просмотреть, переход на блок шаг вперед.

Блок шаг вперед

1. Для всех ненулевых элемент в массиве GL{} определяем возможные варианты и запишем их в массив GL+1 {}. При записывании элемента в GL+1{} отмечаем его как просмотренный.

2. Увеличиваем L на единицу (L= L+1).

3. Переход на блок проверок.

Блок закончить работу

Данный блок должен содержать действия по подготовке выходных данных, требуемых по условию задачи.

2.1.5 Кое-что из жизни Графов

- Граф де Ла Фер живет на улице Генего, в гостинице "Карл Великий".

А. Дюма "Двадцать лет спустя"

Граф представляет собой конечное множество вершин и множество упорядоченных и неупорядоченных пар вершин.

Ребро - неупорядоченная пара вершин.

Дуга - упорядоченная пара вершин.

Петля - ребро, ведущее из вершины в эту же вершину.

Смежные вершины - вершины, соединенные ребром.

Смежные ребра - ребра, имеющие общую вершину.

Инцидентными называются ребро и любая из его двух вершин.

Цепь между вершинами A и B - последовательность ребер, соединяющих A и B.

Путь из A в B - последовательность дуг, направленных из A в B.

Неориентированный граф – граф, содержащий только ребра.

Орграф (ориентированный граф) – граф, содержащий только дуги.

Мультиграф - соединение двух вершин несколькими ребрами.

Полный граф - граф, в котором проведены все возможные ребра (при N вершинах N(N-1)/2 ребер).

Простой граф - граф без петель и кратных ребер.

Подграф - часть вершин и все инцидентные им ребра.

Суграф - все вершины и часть инцидентных им ребер.

Связанный граф - это граф, где существует цепь между любой парой вершин.

Взвешенный граф (сети) - когда ребра и дуги имеют вес (длину).

Цикл - цепь из A в A.

Дерево - граф без циклов.

При задании графа используются различные способы описания: матрица инцидентности, матрица смежности, списки связи и перечни ребер. Наиболее часто используют описание в виде матрицы смежности, а так же списка связи и перечня ребер.

Матрица смежности - двумерный массив A размерности NxN.

$$A_{ij} = \begin{cases} l & \text{если вершины } i \text{ и } j \text{ смежны} \\ 0 & \text{если вершины } i \text{ и } j \text{ несмежны} \end{cases}$$

где l - вес ребра.

Для хранения перечня ребер необходим двумерный массив K(в общем случае - размерности Mx3). Каждая строка массива описывает одно ребро (вершина 1, вершина 2, вес ребра).

2.1.6. Жадность не порок, а ...

Рассмотрим первую задачу.

Дана плоская страна и в ней N городов. Необходимо соединить все города телефонной связью так, чтобы общая длина телефонных линий была минимальной.

Вначале, как мы договорились, рассмотрим само условие задачи.

а) Очевидно, что в данной задаче размеры городов малы по сравнению со страной, поэтому будем изображать город точкой. В крайнем случае, точкой будем изображать телефонную станцию, размещенную в городе. Так как страна плоская, то положение i-ой точки (города), очевидно, задается в декартовых координатах (xi , yi), а расстояние между двумя городами будет соответственно равно:

$$d_{ij} = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

б) Так как в задаче подразумевается **транзитивность** (говорится о телефонной связи), то это значит, что если город А связан с городом Б, а Б связан с городом Д, то город А связан с городом Д.

в) Основной вопрос - где могут разветвляться телефонные провода? Для простоты решим, что они соединяются и разветвляются только на телефонных станциях.

Эту задачу называют задачей **Прима-Краскала**.

В терминах теории графов формулировка данной задачи (формулировка Прима) звучит следующим образом:

Дан полный граф с N вершинами, длины ребер определяются по формуле $d_{ij} = \sqrt{(xi - xj)^2 + (yi - yj)^2}$, *где xi , yi - координаты вершин. Найти остовное дерево минимальной длины.*

В формулировке Краскала задача имеет более общий вид:

Дан граф с N вершинами; длины ребер заданы матрицей {dij}, i,j = 1,... n. Найти остовное дерево минимальной длины.

Для решения этих задач можно идти двумя путями:

Путь первый (алгоритм Прима).

1. Выбрать кратчайшее ребро, обозначить его вершины как выбранные.
2. Выбрать кратчайшее ребро, одна из вершин которого является выбранной, а другая принадлежит к оставшейся части сети. Обозначить новую вершину как выбранную.
3. Если есть невыбранные вершины, то перейти на пункт 2, иначе - закончить работу.

Путь второй (алгоритм Краскала).

1. Выбрать самое длинное ребро.
2. Удалить его.
3. Определить, является ли сеть связанной, если да, то перейти на 1.
4. Добавить последнее удаленное ребро и закончить работу.

Оба эти алгоритма относятся к так называемым **“жадным”** алгоритмам. Как правило, применение жадных алгоритмов, то есть принцип на каждом шаге выбирать что-то самое малое (большое) из

возможных, при решении оптимизационных задач может привести к неверным решениям. Однако в нашем случае, что удивительно, алгоритм Прима-Краскала дает точное оптимальное решение.

Хотя оба подхода дают одинаковый результат, но первый, на мой взгляд, более прост в реализации. Формирование остова начинается с произвольной вершины графа. Первым в остовное дерево включается ребро наименьшего веса, выходящее из выбранной вершины. На каждом шаге к дереву добавляется ребро наименьшего веса среди ребер, соединяющих вершины этого дерева с вершинами, пока в дерево не вошедшими. При реализации важно уметь быстро выбирать требуемое ребро минимальное веса. Для того, чтобы на каждом шаге не пересчитывать расстояние от текущего остовного дерева до всех не вошедших в него вершин, эти расстояния удобно хранить в линейном массиве (в программе d), пересчитывая его значения после добавления в остов нового элемента. Для пометки вершин, уже вошедших в остовное дерево, будем использовать булевский массив vert. Следующие две процедуры показывают, как для произвольного графа, вес ребер которого задается с помощью функции a(i,j), а отсутствие ребра между двумя вершинами обозначается весом, равным O (конкретное числовое значение для этого параметра подбирается исходя из условия задачи), построить остовное дерево.

```
procedure calc(i:integer);
{пересчитывает расстояние до остова, i — вершина, включенная в остов последней}
var j:integer;
begin
  for j:=1 to n do
    if not vert[j] then
      if d[j]>a(i,j) then
        begin
          d[j]:=a(i,j);
          res[j]:=i
        end
end;
procedure build;
{строит минимальный остов}
var i,j,imin:integer;
    min:extended;
begin
  fillchar(vert,sizeof(vert),false);
  for i:=1 to n do d[i]:=O;
  vert[1]:=true;
  calc(1);
  for j:=1 to n-1 do
  {остов состоит из n-1 ребра}
    begin
      min:=Ґ;
      for i:=1 to n do
        if not vert[i] then
          if min>d[i] then
            begin
              min:=d[i];
              imin:=i
            end;

vert[imin]:=true;
```

```
    calc(imin);
{в остов вошло ребро imin-res[imin]}
    writeln(imin,' ",res[imin])
  end
end;
```

2.1.7 Эх, дороги...

Трудно представить, хотя и можно, что телефонные провода соединяются и разветвляются не на телефонных станциях, а в чистом поле. Чтобы не напрягать воображение, заменим телефонные провода дорогами. Они уж точно могут разветвляться в чистом поле. В результате получим следующую задачу:

Дана плоская страна и в ней N городов. Необходимо соединить все города кратчайшей системой дорог так, чтобы можно было проехать из каждого города в каждый?

Эта задача носит название "задача Штейнера", а требуемая сеть дорог - "минимальная сеть Штейнера".

Попробуйте, не читая дальше, оценить, насколько изменится метод (алгоритм) решения данной задачи относительно предыдущей (см. 2.1.6. Жадность не порок, а...).

Если что-то сразу не получается, можно и **нужно** действовать постепенно.

а) Рассмотрим вариант, когда N=2 (всего два города). Вы, наверно, уже догадались, что кратчайшей дорожной сетью (минимальной сетью Штейнера) будет прямая, соединяющая эти два города.

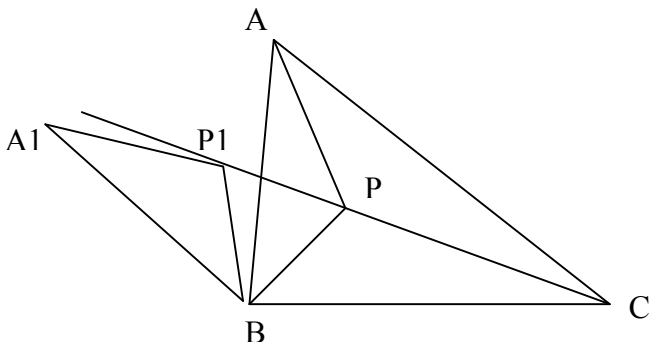
б) Несколько сложнее с тремя городами. Три города можно представить как вершины треугольника, тогда внутри него будет находиться такая точка P (развилка), что сумма длин дорог, соединяющих города, будет минимальна .

Задача Торичелли-Ферма

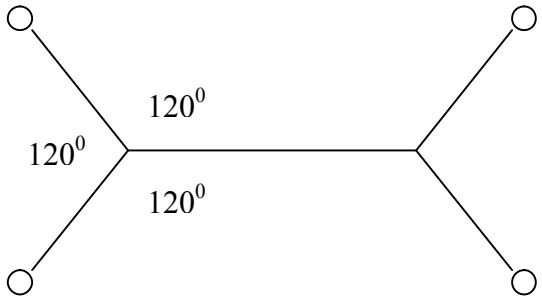
В треугольнике ABC найти такую точку P, чтобы сумма расстояний от P до вершин A, B и C была минимальной.

Для нахождения положения этой точки нам поможет Наполеон Бонапарт. В бытность кадетом артиллерийского училища он предложил следующее решение задачи Торичелли-Ферма:

Пусть P - произвольная точка внутри треугольника. Повернем треугольник ABP на 60^0 против часовой стрелки вокруг вершины B. AP=AP1 и BP=BP1 по построению, BP=PP1 потому, что треугольник BPP1 равносторонний. Сумма расстояний от точки P до A,B,C равна длине ломаной A1P1PC. Если точка P примет такое положение, что ломаная станет прямой, то сумма расстояний станет минимальной. Это произойдет, если углы при точке P будут равны 120^0 .



в) Предположим, что имеется четыре города, расположенные по сторонам единичного квадрата. Тогда минимальная сеть Штейнера будет иметь вид:



К сожалению, когда расположение городов задается произвольной сетью, никакого точного алгоритма задачи Штейнера пока неизвестно. Поэтому прошу извинить меня за попытку ввести Вас в заблуждение.

Как уже отмечалось (см. Часть 1), небольшое различие вусловиях задач может приводить к тому, что одна из них будет решаться достаточно “просто”, а другая - запредельно сложно или вообще не иметь общего решения.

2.1.8 Дороги, которые мы выбираем, или почти по О. Генри

С прокладкой дорог и проводкой телефонных линий мы немного разобрались. Теперь рассмотрим несколько иную задачу:

Дана некоторая произвольная дорожная сеть, необходимо определить кратчайший путь между заданными точками.

Или в терминах теории графов:

Дана сеть N(V,E,W). Найти кратчайшую цепь между vj и vл (vj , vл О *V).*

Конечно, данную задачу можно решать путем полного перебора всех возможных вариантов движения, а затем выбрать из них минимальный. Но, как мы уже знаем, для решения задач такого характера лучше использовать метод поиска в ширину. Мы с Вами уже рассматривали этот метод на примере поиска кратчайшего пути в лабиринте.

Примечание. Лабиринт - это тот же граф, где вершины - клетки, а ребра - возможность перехода из клетки в клетку.

Основное отличие заключается в том, что если время перехода из клетки в клетку одинаково, то в общем случае длина дорог может быть различна.

Таким образом, если мы модернизируем алгоритм поиска в ширину с учетом длины ребер, то получим требуемый нам алгоритм.

Алгоритм использует три массива из n чисел каждый. Первый массив (a) служит для отметки просмотренных вершин. Второй массив (b) содержит расстояния - текущие кратчайшие расстояния от исходной до текущей вершины. Третий массив (c) служит для записи номеров вершин (ck - номер предпоследней вершины на текущем кратчайшем пути от начальной вершины до конечной)

Алгоритм Дейкстры

1. В цикле от 1 до n

Выполнить:

а) Заполним нулями массив a.

б) Заполним числом s (стартовая вершина) массив c.

г) Перенесем s-ю строку матрицы расстояний (D) в массив b.

2.Среди неотмеченных вершин (a[k]=false) найти минимальное значение bk.

Отмечаем эту вершину b[k]=True.

Если bj >bk+djk, то (bj = bk+djk, ck=j).

3. Определить, все ли вершины отмечены, если нет, то перейти на пункт 2. Если все вершины отмечены, то длина пути равна b[k]. Перейти на 4.

4. Выдать кратчайший путь b[k] и номера вершин.

4.1. z = c[k]

4.2. Выдать z.

4.3. z = c[z], Если z = 0, то конец, иначе - на 4.2.

Приведем одну из схем программ, реализующих этот алгоритм (функцию d(i, j) и значение “бесконечности”(O) определять не будем):

```
for i:=1 to N do
  begin
    b[i]:= O;
    a[i]:=false;
  end;
p:=s; l[s]:=0;
a[s]:=true;
f:=true;{стоит ли искать дальше}
while (p<>t) and f do
 begin
  f:=false;
  for j:=1 to N do
   if not a[j] then
    if b[p]+d(p,j)<b[j] then b[j]:=b[p]+d(p,j);
  min:=t;{важно, что a[t]=false}
  for j:=1 to N do
    if (not a[j])and(b[j]<b[min]) then min:=j;
  if b[min]< O then
   begin
    p:=min; a[p]:=true; f:=true
   end
end;
```

Рассмотрим следующую задачу:

В N городах проживают по Ki учащихся которые должны принять участие в олимпиаде по информатике. В каком городе надо провести олимпиаду чтобы затраты на их перевозку были минимальны. Стоимость проезда между городами задана матрицей смежности.

Можно использовать алгоритм Дейкстры для нахождения расстояния между всеми возможными парами городов, но есть простой способ.

Наиболее просто найти кратчайший путь между каждой из пар вершин можно с помощью алгоритма Флойда. Приведем фрагмент программы, реализующий алгоритм Флойдаи печатающий кратчайшие пути между всеми парами вершин графа.

```
procedure way(i,j:integer);
{печатает путь между вершинами i и j}
begin
  if w[i,j]=0 then write(“ “,j)
  {печатаем только вершину j, чтобы избежать повторов}
  else
    begin
```

```
     way(i,w[i,j]); way(w[i,j],j)
   end
end;

begin
 …{заполняем матрицу смежности}
 for k:=1 to N do
  for i:=1 to N do
   for j:=1 to N do
    if a[i,k]+a[k,j]<a[i,j] then
     begin
       a[i,j]:=a[i,k]+a[k,j];
       w[i,j]:=k
     end;
 for i:=1 to N do
  for j:=1 to N do
   begin
     write(i);
     if i<>j then way(i,j);
     writeln
   end
end.
```

2.1.9 Обезьяна и бананы

На шестой всемирной олимпиаде по информатике (Швеция,1994г.) в первом туре была представлена задача под названием “Треугольник”:

На рисунке изображен треугольник из чисел. Напишите программу, которая вычисляет наибольшую сумму чисел, расположенных на пути, начинающемся в верхней точке треугольника и заканчивающемся на основании треугольника.

```
                7
             3     8
          8     1     0
       2     7     4     4
    4     5     2     6     5
```

Каждый шаг на пути может осуществляться вниз по диагонали влево или вниз по диагонали вправо.

Число строк в треугольнике >1 и < 100. Треугольник составлен из целых чисел от 0 до 99.

Условно мы называем эту задачу “Обезьяна и бананы”:

Обезьяна спускается с дерева по правилам, указанным выше. На каждой ветке она может взять определенное количество бананов. Выбрать маршрут, при котором количество бананов будет максимальным.

При применении “жадного” алгоритма для данных, представленных на рисунке, получим цепочку 7-8-1-7-5 (сумма 27).

Если проведем полный перебор вариантов, то оптимальной будет цепочка 7-3-8-7-5 (сумма 30). Однако при решении данной задачи методом полного перебора вариантов путей, необходимое число действий растет примерно как 2 в степени N, что неприемлемо при больших значениях N.

Что делать? Для объяснения принципа решения этой задачи рассмотрим задачу которая является базовой для данного типа задач.

Из города А в город В необходимо построить дорогу минимальной длины. Имеется прямоугольная карта местности, разделенная на квадраты. Для каждого квадрата указана стоимость постройки дороги на этом участке. Надо провести дорогу таким образом, чтобы стоимость строительства ее была минимальна.

A 5	8	1	2
3	6	4	1
12	1	3	5 B

Как видно из рисунка двигаться из города А в город В можно либо вниз, либо вправо (первое условие). Если использовать "жадный" алгоритм, как и в случае задачи "Обезьяна и бананы" можно получить не оптимальное решение. Попробуем решать ее с конца. Если мы находимся в правой нежней клетке, то стоимость строительства будет не менее 5. В эту клетку можно попасть либо с левой клетки, либо с верхней клетки. Для первого случая стоимость будет уже не менее 8, а во втором – 6. Проведем эти действия для нижнего и крайне правого рядов. В результате получим следующую картину.

5	8	1	8 2 ↓
3	6	4	6 1 ↓
21 12	9 → 1	8 → 3	5 5

Теперь посмотрим куда лучше двигаться из клетки 2,3. Если пойдем вниз, то стоимость будет не ниже 12 , а если направо то – 10. Значит лучше двигаться вправо и минимальная стоимость будет для этой клетки 10. Проведем анализ для оставшихся клеток. В результате получим, что минимальная стоимость строительства будет равна 22, стрелки покажут направление движения строительства дороги.

22 → 5	17 → 8	9 → 1	8 2 ↓
18 → 3	15 6 ↓	10 → 4	6 1 ↓
21 → 12	9 → 1	8 → 3	5 5

Посмотрим, как это можно реализовать программно. Массив A- исходный массив, S - массив минимальной стоимости, N1 - массив направления движения.

```
S[n,m]:=A[n,m];
For  j:=m-1 downto 1 do
        begin
        S[n,j]:=S[n,j+1]+A[n,j];
```

```
        N1[n,j]:=1;
        end;
For  i:=n-1 downto 1 do
        begin
        S[i,m]:=S[i+1,m]+A[i,m];
        N1[i,m]:=0;
        end;
For i:=n-1 downto 1 do
   For  j:=m-1 downto 1 do
        If  S[i+1,j]< S[i,j+1] then
begin
S[i,j]:=S[i+1,j]+A[i,j];
N1[i,j]:=0;
end;
else
begin
S[i,j]:=S[i,j+1]+A[i,j];
N1[i,j]:=1;
end;
Writeln(S[1,1]);
```

Данный подход позволяет определить не только общую минимальную стоимость, но для любой клетки наилучшее движение в конечную клетку.

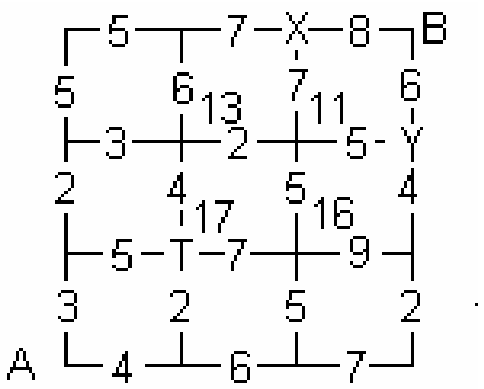
Нашу программу можно упростить. Для этого воспользуемся искусственным приемом. Добавим к нашей таблице строку и столбец.

5	8	1	2	1000
3	6	4	1	1000
12	1	3	5	0
1000	1000	1000	0	0

```
S:=A;
For i:=n downto 1 do
   For  j:=m downto 1 do
        If  S[i+1,j]< S[i,j+1] then
begin
S[i,j]:=S[i+1,j]+A[i,j];
N1[i,j]:=0;
end;
else
begin
S[i,j]:=S[i,j+1]+A[i,j];
N1[i,j]:=1;
end;
Writeln(S[1,1]);
```

Теперь рассмотрим еще одну задачу. Задача о черепашке. Черепашке необходимо попасть из пункта А в пункт В. На каждом углу она может поворачивать только на север или только на восток.

Время движения по каждой улице указано на рисунке. Требуется найти минимальное время, за которое Черепашка может попасть из пункта A в пункт B.



Преобразуем исходную задачу сведя ее к выше изложенной. Будем считать что на перекрестках время передвижения равно 0, а на газонах равна 500. Исходная матрица примет вид

500	500	500	500	500	500	0	0
0	6	0	7	0	8	0	0
5	500	6	500	7	500	6	500
0	3	0	2	0	5	0	500
2	500	4	500	5	500	4	500
0	5	0	7	0	9	0	500
3	500	2	500	5	500	2	500
0	4	0	6	0	7	0	500

И теперь решение данной задачи не должно составить трудности. Попробуйте применить этот подход к нашей задаче " Обезьяна и бананы".

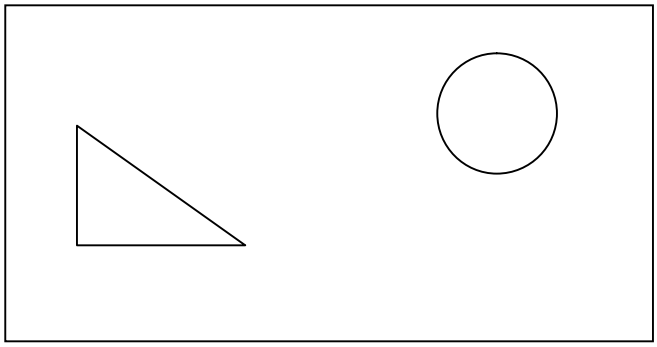
Этот алгоритм можно использовать не только в случае треугольника, но и для всех ориентированных графов. Примером может служить задача "Утиные истории" (см. часть 1). При рассмотрении "с конца" должны просматриваться только те вершины графа, которые смежны с уже просмотренными вершинами. Это позволяет в ряде случаев существенно сократить время просмотра. Если у нас есть ограничение на количество ходов ("Утиные истории"), то дополнительно вводится еще одна характеристика - номер хода.

2.1.10 Чучундра, или "ПО ГАЗОНАМ НЕ ХОДИТЬ !"

Как известно, на плоскости кратчайший путь между двумя точками есть прямолинейный отрезок, соединяющий эти точки. А как изменится задача нахождения кратчайшего пути, если есть препятствия. Такую задачу предложили во втором туре международных соревнований по информатике ICI-98 (г. Могилев). Ниже приведем условие задачи:

На участке прямоугольной формы расположены клумба в виде круга радиуса R и парник в форме многоугольника.

Написать программу, вычисляющую длину кратчайшего пути из левого нижнего угла участка (начало координат) в правый верхний угол участка, при этом путь не должен пересекать ни парник, ни клумбу. Парник и клумба не имеют общих точек и целиком расположены внутри участка.



Данная задача относится к задачам трассировки. Различают два основных класса этих задач:

- когда препятствия непроходимы;
- когда препятствия частично проходимы.

Мы рассмотрим первый класс. Он также условно делится на два вида: строительную трассировку и электронную трассировку. Основное различие их в том, что в первом случае ранее проведенные пути не служат помехой другим путям (т.е. они могут взаимно пересекаться), а во втором пути не могут пересекаться (пример - электронная печатная плата).

Здесь сделаем небольшое отступление и вспомним кое-что из геометрии.

“Вычислительная геометрия – это раздел информатики, изучающей алгоритмы решения геометрических задач”. Задачи связанные с вычислительной геометрией встречаются в компьютерной графике, проектировании технических устройств, архитектурных сооружений и др. Данными в этих задачах являются : множество точек, набор отрезков, различные геометрические фигуры и т.л. В этом разделе будут показаны подходы к решению простейших задач связанных с геометрией на плоскости.

Уравнение прямой.

В декартовых координатах общее уравнение прямой имеет вид:

$$Ax + By + C = 0$$

Уравнение прямой, проходящей через две точки с координатами (x_1,y_1) и (x_2,y_2):

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

Эту формулу, чтобы снять неопределенность при $x_1 = x_2$ или $y_1 = y_2$, часто используют в виде:

$$(x- y_1)(y_2- y_1)- (y- y_1)(x_2- x_1)=0$$

Это выражение можно привести к общему уравнению прямой:

$$Ax + By + C = 0$$

где $A = y_2- y_1$, $B = x_1- x_2$, $C = -x_1(y_2- y_1)+ y_1(x_2- x_1)$.

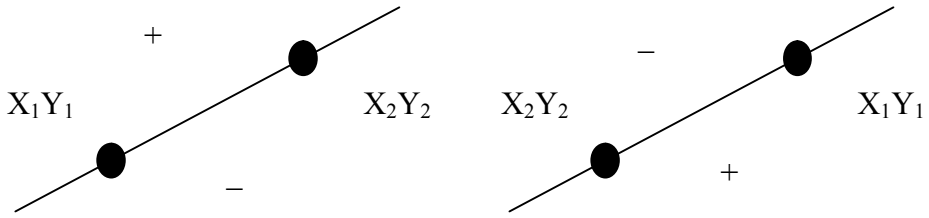
Положение точки относительно прямой.

а) если имеется точка с координатами (x_0 ,y_0) и $(x_0- y_1)(y_2- y_1)- (y_0-y_1)(x_2-x_1)=0$, то точка лежит на прямой. Аналогично и при использовании общего уравнения прямой $Ax_0 + By_0 + C = 0$.

б) если имеется точка с координатами (x_0,y_0) и $Ax_0 + By_0 + C > 0$, то точка лежит выше прямой.

в) если имеется точка с координатами (x_0,y_0) и $Ax_0 + By_0 + C < 0$, то точка лежит ниже прямой.

Примечание: Мы задаем положение прямой парой точек, и если мы посмотрим из точки с координатами (x_1, y_1) в точку (x_2, y_2) то точки лежащие выше прямой будут находиться с левой стороны прямой, а лежащие ниже – с правой.



Положение точек относительно прямой.

Пусть заданы две произвольные точки (x_3,y_3 и x_4,y_4) и прямая линия. Надо определить находятся ли эти точки по одну сторону относительно прямой или по разные.

Из выше изложенного видно, что если для двух точек с координатами (x_i,y_i) и (x_j ,y_j) значения выражений $Ax_i + By_i + C$ и $Ax_j + By_j + C$ имеют одинаковые знаки, то они лежат по одну сторону относительно прямой, если разные знаки - по разные стороны.

Приведем участок текста программы, которая определяет лежат ли точки по одну сторону от прямой или по разные. Условимся, что если они лежат по одну сторону результат работы программы 'YES' иначе 'NO'. Для этого воспользуемся свойством произведения двух чисел.

Знак первого числа	Знак второго числа	Знак результата
Положительное	Положительное	Положительное
Положительное	Отрицательное	Отрицательное
Отрицательное	Отрицательное	Положительное
Отрицательное	Положительное	Отрицательное

```
Rezult:=' NO';
a := y[2]- y[1];
в := x[1]- x[2];
c := -x[1]*(y[2]- y[1])+ y[1]*(x[2]- x[1]);
if  (a*x[3] + b*y[3] + c)* (a*x[4] + b*y[4] + c)>0 then Rezult:='YES';
```

В этом примере мы специально проверяли условие положительности результата, так как это гарантирует то, что точки лежат по разные стороны относительно прямой. Это связано с тем, что одним из возможных вариантов может быть результат, когда произведение равно 0. Это может быть в случае если одна или обе точки лежат на прямой линии, в этом случае можно считать, что они лежат по разные стороны относительно прямой. Если Вы несогласны поменяйте в условии знак '>' на знак '<' и соответственно первоначально назначте Rezult:='YES'.

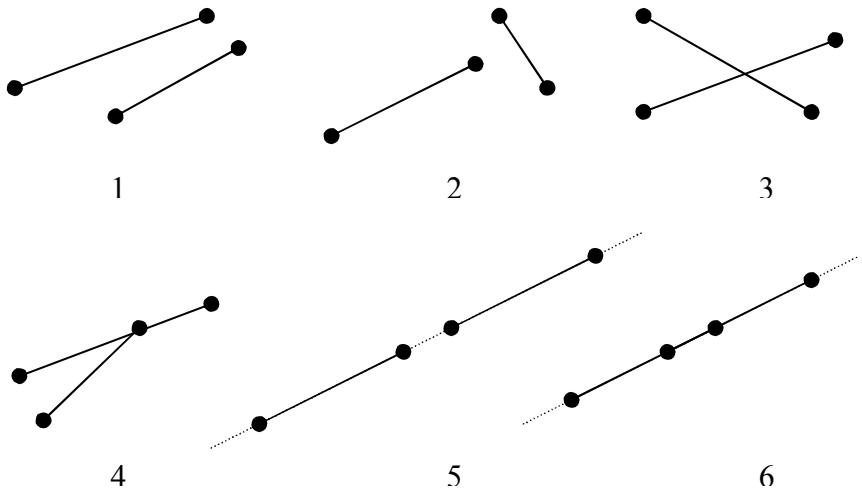
Точка пересечения отрезков.

Пусть нам необходимо определить точку пересечения двух отрезков (отрезки заданы координатами своих концов). Как известно, в евклидовой геометрии, если прямые не параллельны, то они обязательно пересекуться.

Если $A_1x + B_1y + C_1 = 0$ и $A_2x + B_2y + C_2 = 0$ есть уравнения прямых, на которых лежат первый и второй отрезки соответственно, то значения координаты точки пересечения этих прямых будут равны:

$$y = \frac{A_1 C_2 - A_2 C_1}{A_2 B_1 - A_1 B_2} \qquad x = \frac{B_1 C_2 - B_2 C_1}{A_1 B_2 - A_2 B_1}$$

К сожалению, отрезки в отличие от прямых линий имеют конечную длину. Рассмотрим возможные случаи взаимного расположения отрезков.



Обратим внимание на варианты 1 и 2. Можно заметить, что если концы хотя бы одного из отрезков лежат по одну сторону от прямой, проходящей через концы другого отрезка, то отрезки не пересекаются. Поэтому для данных случаев нет смысла искать точку пересечения отрезков (зачем искать черную кошку в черной комнате если ее там нет? **прим. автора**).

Для случаев 3 и 4 – концы каждого из отрезков лежат по разные стороны от прямой линии на которой лежит другой отрезок (посмотрите предыдущий раздел).

Случай 5 и 6. Отрезки лежат на одной прямой. Если их проекции пересекаются (т.е. проекция правого края хотя бы одного из отрезков больше проекции левого края другого отрезка), то они пересекаются. Если проекции отрезков не пересекаются, то и отрезки не пересекаются. Несмотря на то, что отрезки в шестом случае пересекаются, точек пересечения будет бесконечно много. Поэтому нас будут интересовать варианты 3 и 4.

Попробуем реализовать это программно:

```
Flag:=True;
a[1] := y[2]- y[1];
в[1] := x[1]- x[2];
c[1] := -x[1]*(y[2]- y[1])+ y[1]*(x[2]- x[1]);
a[2] := y[4]- y[3];
в[2] := x[3]- x[4];
c[2] := -x[3]*(y[4]- y[3])+ y[3]*(x[4]- x[3]);
{отсеиваем варианты 1 и 2}
if  ((a[1]*x[3]+b[1]*y[3]+c[1])*(a[1]*x[4]+b[1]*y[4]+c[1])>0) or
((a[1]*x[3]+b[1]*y[3]+c[1])*(a[1]*x[4]+b[1]*y[4]+c[1])>0) then Flag:=False;
{отсеиваем варианты 5 и 6}
if  ((a[1]*x[3]+b[1]*y[3]+c[1])=0)and((a[1]*x[4]+b[1]*y[4]+c[1])=0) then Flag:=False;
{проводим вычисление точки пересечения}
if Flag then
   begin
   x:= (b[1]*c[2]-b[2]*c[1])/(a[1]*b[2]-a[2]*b[1]);
   y:= (a[1]*c[2]-a[2]*c[1])/ (a[2]*b[1]-a[1]*b[2]);
   end;
```

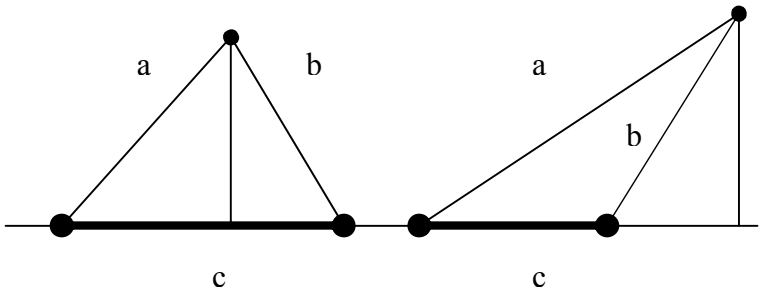
Расстояние между точкой и отрезком.

Расстояние от точки до прямой на плоскости определяется как длина отрезка перпендикуляра, опущенного из точки на прямую. Оно равно:

$$D = \frac{(y_2 - y_1) * x + (x_2 - x_1) * y}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

Для определения расстояния между точкой и отрезком необходимо определить, пересекает ли перпендикуляр, опущенный из точки на прямую, проходящую через концы отрезка, отрезок. Если да, то расстояние находится по формуле, определяющей расстояние от точки до прямой. Если перпендикуляр не пересекает отрезок, то расстояние равно минимальному из расстояний между точкой и одним из концов отрезка.

Для определения пересекает ли перпендикуляр отрезок, поступим следующим образом: рассмотрим треугольник, образованный тремя точками (две точки - концы отрезка, третья точка - искомая точка).



Очевидно, если перпендикуляр пересекает отрезок, то углы при основании треугольника должны быть острыми, а если перпендикуляр не пересекает отрезок, то один из углов при основании тупой. Условием наличия тупого угла является:

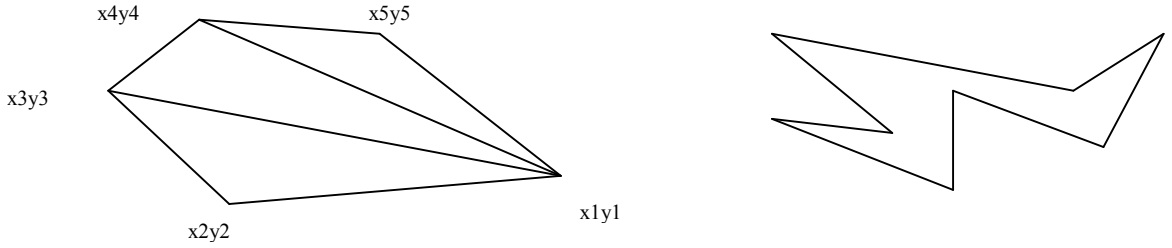
$a^2 > b^2 + c^2$ или $b^2 > a^2 + c^2$,

где c - длина основания (длина отрезка), a и b расстояния от точки до концов отрезка.

Таким образом, если условие не выполняется, то расстояние вычисляется по формуле расстояния от точка до прямой. Если это условие выполняется, то берется минимальное из значений a или b.

Площадь многоугольника

Очень часто в учебниках по программированию, рассматривая вычисление площади многоугольника, используют разбиение его на треугольники. Понятно, что площадь многоугольника будет равна сумме площадей треугольников. А площадь треугольников определяют с помощью формулы Герона.



Недостатки данного способа определения площади многоугольника очевидны. Во-первых, вряде случаев (когда многоугольник является не выпуклым) очень сложно объяснить машине на какие треугольники и как она может разбить исходный многоугольник. Во-вторых, при определении длин сто-

рон, а также в формуле определения площади присутствует действие извлечения квадратного корня, что очень сильно влияет на точность полученного результата.

Предлагаю Вам рассмотреть более простой способ нахождения площади любого много угольника. Он основан на свойствах интеграла по замктутому контуру, для простоты я называю его «принцип кота».

Там днем и ночью кот ученый всеходит по цени кругом.
Идет на право песнь заводит, не лево - сказки говороит.

А.С. Пушкин

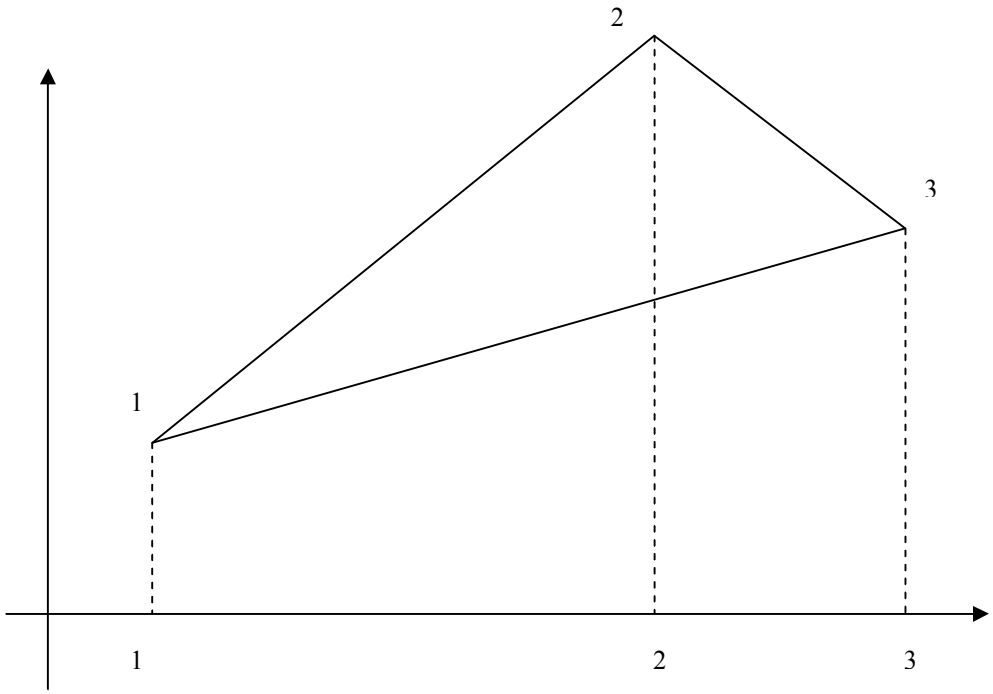
Для примера определим площадь треугольника. Введем несколько ограничений:

1. Координаты вершин треугольника заданы в порядке обхода его по часовой стрелке.
2. Все координаты вершин положительны.

Из рисунка видно, что площадь треугольника будет равна сумме площадей трапеций 1122 и 2233, минус площадь трапеции 1133.

$$S= (x2-x1)(y2+y1)/2 + (x3-x2)(y3+y2)/2 + (x1-x3)(y3+y1)/2$$

Так как значение координаты x3 больше чем x1, последнее слагаемое будет со знаком минус.



Приведем часть программы нахождения площади многоугольника имеющего N сторон (углов).

```
x[N+1]:=x[1];
y[N+1]:=y[1];
S:=0;
For i:=1 to N do
        begin
        S:=S+(x[i+1]-x[i])*(y[i+1]+y[i]);
        end;
S:=S/2;
```

Обратите внимание на первые две строчки. Как и в случае задачи нахождения дороги минимальной длины и стоимости, мы ввели фиктивные данные. Такой прием часто используется при решении задач для упрощения обработки граничных случаев.

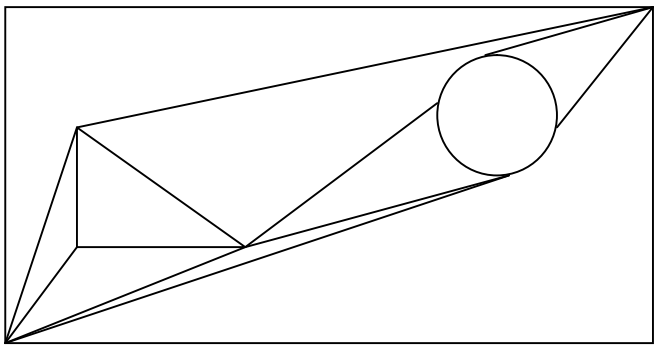
Замечание по ограничению 1. Если координаты вершин треугольника заданы в порядке обхода его против часовой стрелки, то полученная сумма будет отрицательной. Поэтому если Вам неизвестно направления обхода вершин в последней строке напишите:

S:=abs(S/2);

Замечание по ограничению 2. Мы считали, что координаты вершин многоугольника положительны, но как быть если они отрицательны? Для того, чтобы использовать наш подход, воспользуйтесь свойством геометрических фигур не менять своего размера при параллельном переносе.

Рассмотрим, как решается задача, предложенная на олимпиаде ICI-98.

Алгоритм решения данной задачи носит имя Чучундра - так его назвали авторы: Бондарев В.М., Рублинецкий В.И., Сигалов В.Л. Алгоритм назван в честь крысы Чучундры (сказка Р. Киплинга) за способ ее передвижения.



На первом этапе необходимо построить сеть, состоящую из сторон круга, многоугольников и прямолинейных отрезков, соединяющих вершины многоугольников, начальную и конечные точки. Необходимо учитывать только те отрезки, которые не пересекают стороны многоугольника или круга (препятствия непроходимы).

На втором этапе для этой построенной сети находится кратчайший путь с использованием алгоритма Дейкстры.

Для уменьшения количества рассматриваемых вариантов, можно на первом этапе построить выпуклую оболочку для точек начала и конца движения, а также точек вершин треугольника.

Выпуклая оболочка множества *N* точек плоскости

Задача состоит в том, чтобы перечислить все точки, принадлежащие границе выпуклой оболочки заданного множества точек, в порядке ее обхода, например, против часовой стрелки (в некоторых задачах например, Треугольник перечислить только угловые точки). Для эффективного решения этой задачи существует несколько различных алгоритмов Приведем наиболее простую реализацию одного из них — алгоритма Джарвиса.

Перечисление точек искомой границы выпуклого многоугольника начнем с правой нижней точки, которая заведомо принадлежит границе выпуклой оболочки. Сложность данного алгоритма составит $O(kN)$, где k — количество точек в выпуклой оболочке, в худшем случае равное N.

Приведем программу решения данной задачи алгоритмом Джарвиса:

```
type vv = record
x, y: longint;
end;
```

```
var a, b: array[1..100] of vv;
min, m, i, j, k, n: integer;

function vect(a1,a2,b1,b2: vv): longint;
{косое произведение векторов a1a2 и b1b2}
begin
vect := (a2.x - a1.x)*(b2.y - b1.y) .
(b2.x - b1.x)*(a2.y - a1.y)
end;

function dist2(a1,a2: vv): longint;
{квадрат длины вектора a1a2}
begin
dist2 := sqr(a2.x - a1.x) + sqr(a2.y - a1.y)
end;

begin {Main}
readln(n);{количество точек}
for i := 1 to n do
read(a[i].x, a[i].y);
{ищем правую нижнюю точку}
m := 1;
for i := 2 to n do
if a[i].y < a[m].y then m := i else
if (a[i].y = a[m].y) and
(a[i].x > a[m].x) then m := i;
{запишем ее в массив выпуклой оболочки b и
переставим на первое место в массиве a}
b[1] := a[m];
a[m] := a[1];
a[1] := b[1];
k := 1;
min := 2;
repeat
{ищем очередную вершину выпуклой оболочки}
for j := 2 to n do
if (vect(b[k],a[min],b[k],a[j]) < 0) or
((vect(b[k],a[min],b[k],a[j]) = 0) and
(dist2(b[k],a[min]) < dist2(b[k],a[j])))
then min := j;
k := k + 1;
{записана очередная вершина}
b[k] := a[min];
min := 1;
until (b[k].x = b[1].x) and (b[k].y = b[1].y);
{пока ломаная не замкнется}
for j := 1 to k . 1 do {печать результата}
writeln(b[j].x,' ',b[j].y)
end.
```

Существует другой алгоритм решения этой задачи (алгоритм Грэхема) с вычислительной сложностью O(NlogN), основанный на предварительной сортировке точек исходного множества по значению угла в полярной системе координат с центром в одной из точек выпуклой оболочки. То есть наиболее трудоемкой задачей оказывается именно сортировка исходных точек.

```
begin
        readln(n);
for i := 1 to n do
        read(a[i].x, a[i].y);
{ищем правую нижнюю точку}
m := 1;
for i := 2 to n do
if a[i].y < a[m].y then m := i else
if (a[i].y = a[m].y) and
(a[i].x > a[m].x) then m := i;
{запишем ее в массив выпуклой оболочки b и
переставим на первое место в массиве a}
b[1] := a[m];
a[m] := a[1];
a[1] := b[1];
{остальные точки сортируем пузырьком
по значению полярного угла}
for i := n downto 3 do
for j := 2 to i - 1 do
if (vect(a[1],a[j],a[1],a[j + 1]) < 0) or
((vect(a[1],a[j],a[1],a[j + 1]) = 0 ) and
(dist2(a[1],a[j]) > dist2(a[1],a[j + 1])))
then
begin {b[n] . вспомогательная переменная}
b[n] := a[j];
a[j] := a[j + 1];
a[j + 1] := b[n]
end;
{ищем вторую вершину выпуклой оболочки}
i := 2;
while vect(a[1],a[i + 1],a[1],a[2]) = 0 do
i := i + 1;
b[2] := a[i];
b[3] := a[i + 1];
k := 3;
for i := i + 2 to n do
begin
{проверка выпуклости}
while vect(b[k . 1],b[k],b[k],a[i]) <= 0 do
k := k . 1;{удаляем точку из стека}
k := k + 1;
b[k] := a[i] {добавляем точку в стек}
end;
```

```
for j := 1 to k do {печать решения}
writeln(b[j].x,' ',b[j].y)
end.
```

Как правило, отличие алгоритмов построения выпуклой оболочки заключается в певоначальной сортировке исходных точек по какому либо признаку. Понятно, что при увеличении толичества точек время исполнения программы резко возрастает. Например, в задаче «Боинг-747» (международная студенческая олимпиада) надо было определить, с заданной точностью, площадь разлета осколко самолета, при этом количество таких обломков было до 1000000. Координаты осколков находятся в интервале 0<X<1, 0<Y<1. время тестирования 1 секунда. Понятно решая задачу «в лоб» не получится. Необходимо было уменьшить размерность задачи. Уменьшение размерности задачи и количества действий можно было добится используя следующий подход.

1. Организовать массив типа запись:

```
Type Point= Record
maxXpoint: real; {храним реальное значение координаты X для максимального Y}
minXpoint: real; {храним реальное значение координаты X для минимального Y}
maxYpoint: real; { храним максимальное Y для этого элемента массива }
minYpoint: real;  { храним минимальное Y для этого элемента массива }
Flag: byte; {наличие данных в данном элементу массива}
End;
Var
a: array [0..100000] of point;
```

2. Считывая исходные данные:

```
 assign(input,'boing.in');
reset(input);
      while not seekeof do
      begin
      read(x1,y1);
         inde:=round(x1*100000);
      if a[inde].flag=0 then
            begin
               a[inde].minypoint:=y1;
               a[inde].maxypoint:=y1;
               a[inde].minxpoint:=x1;
               a[inde].maxxpoint:=x1;
               a[inde].flag:=1;
            end
            else
            begin
                if y1>a[inde].maxypoint  then
                  begin
                    a[inde].maxypoint:=y1;
                    a[inde].maxxpoint:=x1;
                    a[inde].flag:=2;
                  end;
                if y1<a[inde].minypoint  then
                  begin
                    a[inde].minypoint:=y1;
                    a[inde].minxpoint:=x1;
```

```
                    a[inde].flag:=2;
                  end;

              end;
          end;
    close(input);
```

3. Когда строится оболочка при движении в право рассматриваются точки с максимальными координатами Y, а при обратном ходе с минимальным значениями.

Положение точки относительно выпуклогомногоугольника

Чтобы определить, лежит ли точка внутри выпуклого многоугольника часто рекомендуют следующий подход:

1. Соединить эту точку отрезками с его вершинами.
2. Рассчитать площади получившихся треугольников.
3. Если сумма вычисленных площадей равна площади исходной фигуры (рисунок 3.9. а), то точка лежит внутри, если нет (рисунок 3.9. б) - снаружи.

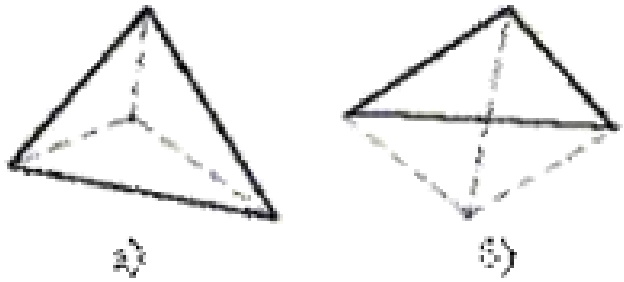


Рисунок 3.9

Этот подход имеет несколько недостатков:

1. Чем больше сторон, тем больше вычислений площадей нам необходимо сделать.
2. Даже если точка лежит внутри многоугольника, из-за погрешностей округления, сумма площадей треугольников и площадь многоугольника могут быть не равны.

Вспомните программу определения выпуклости многоугольника и сравните ее с данной задачей. Оказывается у них много общего. Можно заметить, что если точка лежит внутри выпуклого многоугольника, то она будет находиться правее любой из его сторон (при его обходе сторон по часовой стрелке). Поэтому, можно использовать программу определения выпуклости многоугольника, изменив следующую строку.

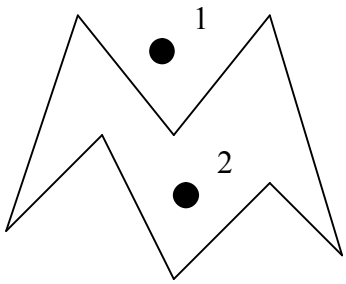
```
if (a*x[i+2]+b*y[i+2]+c)>0 then Rezalt:='NO';
```

на строку

```
if (a*x[0]+b*y[0]+c)>0 then Rezalt:='NO';
```

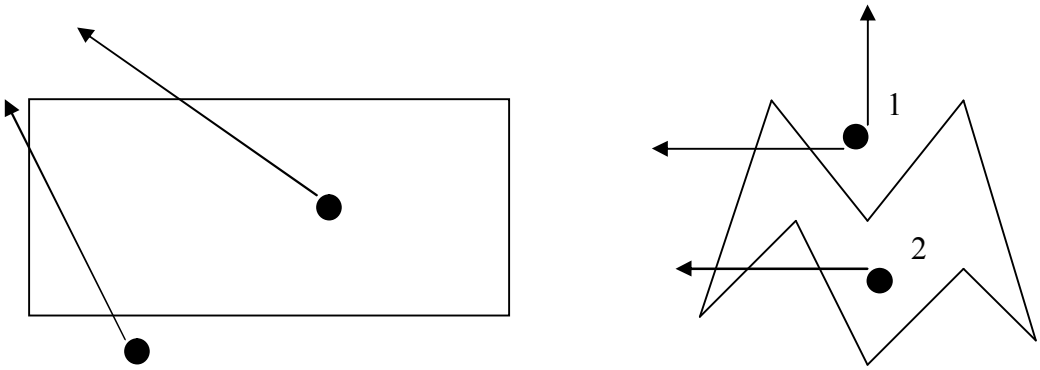
и так как мы не проверяем вершины многоугольника, нет необходимости вводить вторую фиктивную точку.

Если с определением положения точки относительно выпуклого многоугольника как видим проблем нет, то что быдет если многоугольник не выпуклый?



В представленном случае точка 1 находится вне моногоугольника, а точка 2 – внутри. Ясно, что ни сравнение площадей, ни определение положения точки относительно сторон, не поможет нам в решении данного вопроса.

Несмотря в кажущуюся сложность задачи она имеет довольно оригинальное решение. Представим себя героем картины режиссера Вайды «Человек проходит сквозь стену»(Германия 1959). Если находимся внутри прямоугольной комнаты, то двигаясь в любом направлении, что бы выйти их комнаты, мы пересечем оду стенку. Если мы находимся вне комната, то пройдя ее насквозь нам придется пересеч две стенки. В общем случае, если мы пересекаем четное количество стен (или непересекаем ни одну из стенок), то мы находились вне комнаты. Если мы пересекли нечетное количество стенок, то мы находилися внутри комнаты.



Таким образом, задача сводится к определению количеств пересечений сторон многоугольника и отрезка концами которого являются исходная точка и точка лежащая вне многоугольника.

При программной реализации данного алгоритма вторая точка выбирается заведомо находящейся за пределами многоугольника. Это можно сделат прибавив 1 к заначению самой левой (верхней, правой, нижней) координате аершины многоугольника.

2.1.11. Коммивояжер, краски и дырокол

Рассмотрим условия трех задач:

Задача 1.

В стране A есть N городов. Коммивояжер должен выйти из первого города, обойти все города, посетив их по одному разу и вернуться в исходный город. В каком порядке следует проходить города, чтобы замкнутый путь был кратчайшим. Расстояние между городами известны.

Задача 2.

Есть некоторая установка производящая краски. В каждый момент времени она может производить только одну краску. Необходимо произвести N красок. На производство i - той краски затрачивается время ti. Для подготовки оборудования после окончания производства i - той краски и перед началом производства j -той краски необходимо затратить время Cij. Требуется выбрать

такой порядок производства краски, чтобы полное время производства N красок было минимальным.

Задача 3 (Лурье, 1972)

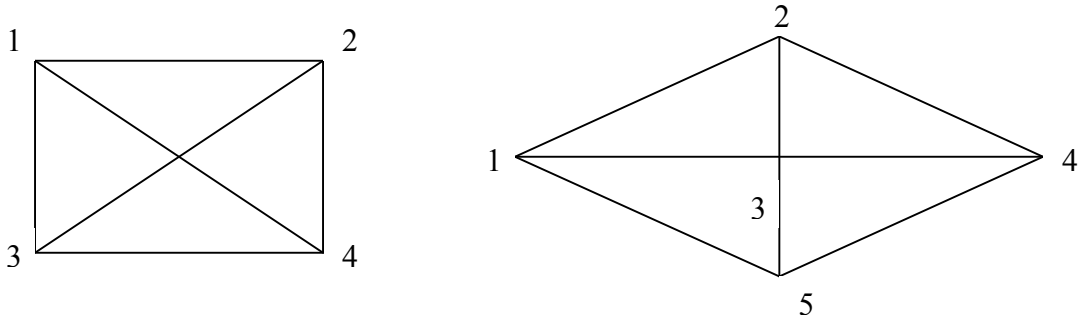
Дыропробивной пресс производит большое число одинаковых панелей - металлических листов, в которых последовательно по одному пробиваются отверстия разной формы и величины.

Операция пробивки j-го отверстия характеризуется четверкой чисел (xj,yj, zj,tj), где xj,yj - координаты отверстия, zj - координаты нужного положения диска с инструментами, tj - время пробивания отверстия.

Производство панелей носит циклический характер; в начале работы и в конце стол находится в исходном положении с координатами x0,y0 при положении диска z0. На каждое действие (перемещение, смена инструмента) затрачивается определенное время. Все действия происходят одновременно, и пробивка дырки происходит сразу после выполнения наиболее длительного из них. Необходимо выбрать такой порядок работы, чтобы полное время производства панели было минимальным.

Эти, казалось бы, разные задачи относятся к задаче о коммивояжере. Рассмотрим несколько подходов к решению задач этого класса. Для начала попробуем использовать "жадный" алгоритм (см. 2.1.4.). Работать будем по принципу "иди в ближайший". Рассмотрим два случая:

а) б)



а) наш алгоритм получит правильное решение;

б) маршрут, выбранный с его помощью (1-2-3-4-1), будет явно не оптимальным.

То есть, применяя "жадный" алгоритм, нет уверенности в том, что мы получим оптимальное решение. Более того, нельзя определить насколько (или во сколько раз) мы ошибаемся.

Если Вы не знаете какого-либо приемлемого алгоритма решения задачи, то я советую воспользоваться методом полного перебора. Ясно, что перебрав все варианты, можно из них выбрать наилучший. Однако для данной задачи количество вариантов очень быстро растет при увеличении n (количество вариантов, которые необходимо просмотреть в несимметричной задаче о коммивояжере с n городами равно (n-1)!). То есть прямой перебор возможен только для небольших значений n. Что делать при больших значениях n? Как Вы помните, при рассмотрении задач, связанных с перебором вариантов, отмечалось, что необходимо сузить множество рассматриваемых вариантов.

Литтл с соавторами (Литтл, 1965) предложил алгоритм решения задачи о коммивояжере с использованием метода ветвей и границ. Общая идея метода такова: надо разделить перебираемые варианты на классы и получить оценки для этих классов, чтобы иметь возможность отбрасывать варианты целыми классами.

Для удобства будем трактовать расстояния между городами как стоимость проезда между ними. Пусть стоимость проезда задана в виде матрицы смежности (в крайнем случае, преобразуем исходныеданные). Если мы изменим для i-го города стоимость проезда из всех городов на одну и ту же величину, то изменится стоимость кратчайшего пути, но сам путь не изменится. То есть при вычитании любой константы из всех элементов любой строки или столбца нашей матрицы, минимальный путь остается минимальным.

Для решения нашей задачи необходимо получить как можно больше нулей в матрице смежности. Для этого из каждой строки вычтем ее минимальный элемент (приведение по строкам), а затем вычтем из каждого столбца его минимальный элемент (приведение по столбцам). Сумма констант приведения по строкам и столбцам определяет оценку снизу для всех туров. Стоимость тура не может быть меньше. Если для приведенной матрицы удастся построить оптимальный тур, то он будет оптимальным и для исходной матрицы. Стоимость тура будет равна сумме стоимости тура для приведенной матрицы и сумме констант приведения.

Далее выбор оптимального тура проводится с помощью перебора вариантов. При выборе очередного хода можно оценить, как изменится общая стоимость тура. Используя эти оценки, можно резко сократить количество рассматриваемых вариантов.

2.1.12 Теория расписаний

Интересный круг задач связан с теорией расписаний. Как правило, они связаны с составлением наилучшего графика работ одним или несколькими исполнителями. Некоторые решаются простыми изящными алгоритмами, другие, иначе как полным перебором, не решить. Перед рассмотрением задач этого класса вспомним кое-что о сортировке.

Сортировка. Как известно, под сортировкой (упорядочением) понимается перераспределение элементов массива в некотором определенном порядке. Основная цель сортировки - облегчить последующий поиск элементов в таком упорядоченном массиве. Примерами упорядоченных объектов являются:

а) размещение книг в хранилищах библиотек;

б) расположение товаров на складах;

в) номера в телефонных справочниках и т.д.

Как правило, сортировка в олимпиадных задачах не является основной темой, однако в ряде случаев ее применение не только сокращает время работы программы, но существенным образом влияет на эффективность ее реализации.

Для примера рассмотрим одну из задач (Котов В.М., Волков И.А., Харитонович А.И.1996):

Имеется 2N чисел. Известно, что их можно разбить на пары таким образом, что произведения чисел в парах равны. Показать эти пары. Все числа натуральные. $1< N< 10000$.

Если мы будем решать ее полным перебором, то, очевидно, можем не уложиться во временные ограничения (постарайтесь оценить, сколько вариантов Вам придется рассмотреть при максимальном возможном значении N). Основная идея задачи состоит в том, чтобы не использовать операцию умножения. Если числа натуральные (т.е. больше 0), то одна из пар должна содержать максимальное и минимальное число. Если их убрать из рассмотрения, то следующую пару можно сформировать таким же образом, выбрав из оставшихся чисел максимальное и минимальное. Таким образом, если мы отсортируем все числа в порядке неубывания, то взяв попарно первое и последнее число, второе и предпоследнее и т.д., получим требуемый результат.

Задание. *Самостоятельно рассмотрите случай, если числа целые. Определите, как изменится алгоритм решения задачи. На какие подзадачи разделится задача.*

Сортировка - хороший пример задачи, которую можно решать с помощью множества различных алгоритмов. Каждый из них имеет и свои достоинства и свои недостатки, поэтому выбирать наилучший алгоритм необходимо, исходя из конкретной постановки задачи.

Существенным критерием выбора нужного метода, среди многих возможных, является его экономичность, т. е. время работы. Хорошей мерой эффективности может служить число необходимых сравнений элементов. Рассмотрим некоторые виды сортировок.

Сортировка выбором. Этот метод сортировки основан на следующих принципах:

1. Выбирается максимальный элемент.

2. Он меняется местами с первым элементом A[1]. На первом месте оказывается максимальный элемент.

3. Дальше рассматривается только неотсортированная часть массива, и этот процесс повторяется с оставшимися (N - 1), (N - 2) элементами и т. д. до тех пор, пока не останется один самый маленький элемент.

```
procedure Linesort(var item: DataArray; count:integer);
   var
   i,j, Index: integer;
x: DataItem;begin
for i:=1 to count-1 do
begin
  index:=i;
    {Поиск максимального элемента}
  for j:=i+1 to n do
          if item[index]< item[j] then index:=j;
    {Обмен максимального элемента с первым рассматриваемым}
  x:= item[i];
  item[i]:= item[index];
  item[index]:=x;
     end;
end;
```

Сортировка обменом. Следующий метод сортировки основан на сравнении двух элементов:

1. Сравниваем первые два элемента. Если первый элемент меньше второго, то меняем их местами.

2. Сравниваем второй и третий, третий и четвертый, ..., предпоследний и последний, при необходимости меняя их местами. Самый маленький окажется на последнем месте.

3. Просматриваем массив с самого начала, уменьшая на единицу количество просматриваемых элементов. Массив будет отсортирован после просмотра, в котором участвуют только первый и второй элементы.

Этот метод широко известен под названием “пузырьковая сортировка”.

```
procedure Bubble(var item: DataArray; count:integer);
            var
              i,j: integer;
              x: DataItem;
  begin
              for i := 2 to count do
              begin
                for j := count downto i do
                  if item[j-1]>item[j] then
                  begin
                    x := item[j-1];
                    item[j-1] := item[j];
                    item[j] := x;
                  end;
               end;
   end; {конец сортировки пузырьковым методом}
```

Можно несколько улучшить этот алгоритм. Можно заметить, что для ряда случаев, когда массив уже отсортирован, программа делает дополнительные просмотры. Что бы избежать этого, мы можем запоминать, были или не были перестановки в процессе некоторого прохода. Если в последнем проходе перестановок не было, то работу можно закончить.

Можно сделать еще одно улучшение, если запоминать не только сам факт, что обмен имел место, но и положение (индекс) последнего обмена. Все пары соседних элементов дальше этого индекса К и уже находятся в требуемом порядке. Поэтому следующий просмотр можно заканчивать на этом индексе, а не идти до заранее определенного индекса N-i.

Будем использовать переменную P (логического типа) для определения, были перестановки или нет, а переменную К - для хранения индекса последнего обмена. Переменная R является границей, на которой заканчивается просмотр.

```
procedure Bubble1(var item: DataArray; count:integer);
var
i: integer;
x: DataItem;
s: Boolean;
begin
    repeat
    s:=true;
for i:=1 to count-1 do
if item[i]< item [i+1] then
     begin
      s:=false;
      x:= item [i];
          item[i]:= item[i+1];
item[i+1]:=x;
      end;
      until s;
end;
```

“Шейкерная” сортировка. При рассмотрении сортировки “методом пузырька”, Вы, очевидно, заметили, что элемент с минимальным значение достигает своего положения за один проход, тогда как элемент с максимальным значением, в общем случае, достигает своего положения только в конце работы алгоритма.

Попробуем после каждого прохода изменять направление просмотра, т.е. если вели просмотр с начала массива и определяли минимальный элемент, то просмотр надо вести с предпоследнего не отсортированного элемента к началу массива, при этом выбирать максимальный элемент.

Такая сортировка называется “Шейкерной”.

```
procedure Shaker(var item: DataArray; count:integer);
        var
           j, k, l, r: integer;
                 x: DataItem;
             begin
               l := 2; r := count; k := count;
               repeat
                 for j := r downto l do
                   if item[j-1]>item[j] then
```

```
        begin    { обмен }
          x := item[j-1];
          item[j-1] := item[j];
          item[j] := x;
          k := j;
        end;

      l := k+1;

      for j := l to r do
        if item[j-1]>item[j] then
        begin   { обмен }
          x := item[j-1];
          item[j-1] := item[j];
          item[j] := x;
          k := j;
        end;
      r := k-1;
    until l>r
  end; { конец челночной сортировки  }
```

Сортировка вставками. Все сортировки, рассмотренные выше, требуют (N! - N)/2 операций сравнения. Используя дихотомический поиск, количество операций сравнения можно сократить.

Будем просматривать элементы массива A, начиная со второго. Каждый новый элемент A[i] будем вставлять на подходящее место в уже упорядоченную совокупность A[1], ...A[i - 1]. Это место определяется последовательными сравнениями элемента A[i] с упорядоченными элементами A[1], ...A[i - 1].

Такой метод сортировки называется сортировкой простыми вставками. Для вставки элемента в нужное место все элементы, стоящие за ним, сдвигаем до позиции, которую занимал вставляемый на данном шаге элемент.

```
procedure Inser(var item: DataArray; count:integer);
    var
      i, l: integer;
      x: DataItem;
    begin
      for i := 2 to count do
      begin
        x := item[i];
        j := i-1;
        while (x<item[j]) and (j>0) do
        begin
          item[j+1] := item[j];
          j := j-1;
        end;
        item[j+1] := x;
end;
  end;  { конец сортировки вставкой }
```

Быстрая сортировки. Метод быстрой сортировки был разработан Ч.Ф.Р.Хоаром и он же дал ему это название. В настоящее время этот метод сортировки считается наилучшим. Он основан на использовании обменного метода сортировки. Это тем более удивительно, если учесть очень низкое быстродействие сортировки пузырьковым методом, который представляет собой простейшую версию обменной сортировки.

В основе быстрой сортировки лежит принцип разбиения. Сначала выбирается некоторое значение в качестве «основы» и затем весь массив разбивается на две части. Одну часть составляют все элементы, равные или большие «основы», а другую часть составляют все элементы меньшего значения, по которому делается разбиение. Этот процесс продолжается для оставшихся частей до тех пор, пока весь массив не будет отсортирован. Например, для массива «fedacb» при использовании в качестве значения разбиения «d» будут получены следующие проходы при выполнении быстрой сортировки:

– исходное состояние: f e d a c b;
– первый проход: d c a d e f;
– второй проход: a b c d e f.

Этот процесс продолжается для каждой части «вса» и def». Фактически этот процесс имеет рекурсивную природу. И действительно, наиболее «аккуратно» быстрая сортировка реализуется посредством рекурсивного алгоритма.

Выбор значения разбиения можно сделать двумя способами. Это значение можно выбирать случайным образом или путем усреднения небольшого числа значений, выбранных из массива. Для оптимальной сортировки требуется выбрать значение, которое будет находиться в точности посередине всех элементов. Однако, для большинства наборов данных это сделать нелегко. Однако, даже в худшем случае, когда выбирается одно из экстремальных значений, быстрая сортировка работает достаточно хорошо.

В приводимом ниже алгоритме быстрой сортировки в качестве значения разбиения используется среднее значение. Хотя такой подход не всегда является наилучшим, он достаточно прост и сортировка будет выполняться правильно.

```
procedure QuickSort(var item: DataArray; count:integer);
procedure qs(l, r: integer; var it: DataArray);
    var
        i, j: integer;
        x, y: DataItem;
            begin
              i:=l; j:=r;
              x:=it[(l+r) div 2];
              repeat
                while it[i]<x do i := i+1;
                while x<it[j] do j := j-1;
                if y<=j then
                begin
                  y := it[i];
                  it[i] := it[j];
                  it[j] := y;
                  i := i+1; j := j-1;
                end;
              until i>j;
              if l<j then qs(l, j, it);
              if l<r then qs(i, r, it)
```

```
        end;
begin
 qs(1, count, item);
end; { конец быстрой сортировки }
```

В данном примере процедура быстрой сортировки обращается к основной процедуре сортировки с именем «qs». Это обеспечивает доступ к данным «item» и «count» при всех вызовах «qs».

Вывод числа операций сравнения и числа операций обмена для быстрой сортировки выходит за рамки данной книги. Можно считать, что число операций сравнения равно n log n, а число операций обмена равно приблизительно n/6 log n. Эти характеристики значительно лучше характеристик рассмотренных ранее сортировок.

Равенство $N = a^x$ можно представить в виде $x = \log_a n$.

Это, например, будет означать, что при сортировке ста элементов методом быстрой сортировки потребуется в среднем 100*2, т.е. 200 операций сравнений, так как log 100 равен 2. Это значение является вполне хорошим по сравнению со значением 990 для сортировки пузырьковым методом.

Однако, следует иметь в виду одно неприятное свойство быстрой сортировки. Если выбираемое для разбиения значение оказывается совпадающим с максимальным значением, то быстрая сортировка превращается в самую медленную с временем выполнения n . Однако, на практике этот случай не встречается.

Необходимо тщательно выбирать метод определения значения разбиения. Часто это значение определяется по фактическим данным, которые сортируются.

2.1.13 Задачи с одним исполнителем, или задача красильщика

Рассмотрим следующую задачу:

Красильщик должен окрасить К предметов. Для окраски j-го предмета требуется время pj. После окраски предмет должен сохнуть bj, после чего он считается готовым. В каком порядке нужно красить предметы, чтобы последний в просушке был готов как можно раньше.

Алгоритм решения этой задачи очень прост - нужно упорядочить работы по убыванию bj (времени просушки), т.е. вначале окрашивать предметы, имеющие максимальное время просушки.

Другая задача из теории расписаний - задача Джексона:

Для каждой работы сопоставляется некоторая величина dj (dj i *0) - директивный срок выполнения работы j. Пусть по предложенному расписанию работа j закончилась в момент времени tj. Величину*

D*j= tj - dj*

назовем отклонением от директивного срока, это отклонение может быть как положительным, так и отрицательным. Как построить расписание выполнения работ, чтобы минимизировать сумму отклонений.

Оказывается, для этого необходимо упорядочить работы по возрастанию их директивных сроков.

Предположим, что выполнение работ до директивного срока не штрафуется, а после директивного срока штрафуется. Как минимизировать число запоздавших работ? Данную задачу поставил Мур, а простой алгоритм ее решения предложил Ходжсон.

Алгоритм Ходжсона.

1. Упорядочить все работы в порядке возрастания их директивных сроков. Если имеется две работы с равными директивными сроками выполнения , то первой ставится работа с меньшим номером.

2. Вводим переменную k. k:=0.

3. k:= k+1. Если работа k отмечена или значение k больше, или равно количеству работ, то полученная перестановка оптимальна.

4. Если время завершения k-ой работы меньше или равно директивному сроку для k-ой работы, то перейти на пункт 3.

5. Если время завершения k-ой работы больше ее директивного срока, то среди работ от 1 до k ищем работу с максимальным временем исполнения и ставим ее в конец списка. Отмечаем эту работу. Переходим на пункт 3.

2.1.14 Задачи с несколькими исполнителями, или время разбрасывать камни, и время собирать камни.

Рассмотрим следующую задачу.

Дана куча из N камней веса p1, p2, ..., pn ; разбросать ее на две кучи максимально равного веса.

В принципе куч может быть не две, а M. Рассматривая эту задачу, Бондарев В.М., Рублинецкий В.И. и Качко Е.Г. считают, что несмотря на простоту формулировки, точно ее решить иначе, чем полным перебором, невозможно. Далее мы попробуем предложить алгоритм решения аналогичной задачи, не используя перебор, а пока рассмотрим еще одну задачу. Она была предложена на олимпиаде ICI-98 (г.Могилев 1998г.):

Есть группа из N учащихся лицея, они должны пройти медосмотр в двух кабинетах А и Б. Время прохождения медосмотра i-го учащегося в этих кабинетах ai и bi соответственно. Чтобы пройти осмотр в кабинете Б, учащийся должен обязательно пройти осмотр вкабинете А. Составить расписание, при котором общее время осмотра минимально.

Эта задача является перефразированной двухоперационной задаче С. Джонсона. Что интересно, она имеет очень простое решение.

Алгоритм (Джонсон). Нахождение оптимального расписания в двухоперационной задаче Джонсона.

1. Среди величин a1 , a2 , ..., an и b1 , b2 , ..., bn выберем минимальное число.

2. Если минимальное число принадлежит массиву a, например, число ai , то i -ю операцию назначить первой из невычеркнутых, в противном случае (принадлежит массиву b) - последней среди невычеркнутых. Вычеркнуть соответствующие ai и bi из массива входных данных

3. Если все a и b вычеркнуты, то конец, иначе - вернуться к 1.

2.1.15 Элементы комбинаторики

Всем Вам, наверно, известна печальная история, описанная Крыловым, о Мартышке, Осле, Козле и косолапом Мишке, которые пытались решить проблему качества игры их квартета путем подбора оптимальной комбинации исполнителей.

Мы с Вами также займемся аналогичными вопросом. Опыт проведения олимпиад показывает, что если в 1993-94 годах практически все участники уверено справлялись с задачами на комбинаторику, то в последнее время многие не знают элементарных формул и понятий.

Комбинаторика – это область математики, в которой изучаются вопросы о том, сколько различных комбинаций, подчиненных тем или иным условиям, можно составить из заданных объектов.

Иногда количество вариантов, которые надо проанализировать при решении комбинаторной задачи, очень велико. В этом случае единственной возможностью получения результата за допустимое время, а не через столетия, является использование компьютера.

Как и любая другая наука, комбинаторика имеет свою терминологию. Основным понятием комбинаторики является комбинаторный объект или соединение. При выборе M элементов из N различных элементов принято говорить, что они образуют соединение из N элементов по M.

В зависимости от того, имеет ли значение порядок элементов в соединении или нет, а также от того, входят в соединение все N элементы или только часть их, различают три вида соединений. Это перестановки, размещения и сочетания.

Перестановки – соединения, каждое из которых содержит N различных элементов, взятых в определенном порядке, называются перестановками из N элементов. Следует отметить, что порядок элементов в перестановке существенен, и в образовании перестановки участвуют все N элементов (M = N).

Пример. Выпишем все перестановки из элементов a, b, c:

abc, acb, bac, bca, cab, cba.

Количество всех способов, которыми можно переставить N различных предметов, расположенных на N различных местах, принято обозначать PN(читается: <число перестановок из N>).

Найдем PN. На первое из имеющихся мест предмет может быть выбран N способами, на второе место – (N - 1) способом, на третье место – (N - 2) способами и т. д. На предпоследнее место предмет выбирается из двух оставшихся, а для последнего места выбор предмета единственен. Общее количество способов будет равно произведению N (N- 1)* (N - 2) ... * 2 * 1. Такое произведение называется факториалом числа N и обозначается N!.

Таким образом, PN = N!.

Иногда требуется не просто подсчитать количество перестановок из n элементов, но и найти каждую из них. Существует несколько алгоритмов генерации всех перестановок из n элементов. Они различаются порядком генерации перестановок.

Рассмотрим один из хорошо известных алгоритмов, в процессе исполнения которого перестановки располагаются лексикографически (в словарном порядке). Это значит, что перестановки сравниваются слева - направо поэлементно и большей из них является та, у которой раньше встретился элемент, больший соответствующего ему элемента во второй перестановке. (Например, если S=(3, 5, 4, 6, 7), a L=(3, 5, 6, 4, 7), то S < L.)

Приведем фрагмент программы, генерирующей всеперестановки в лексикографическом порядке:

```
for i:= 0 to n do
p[i] :=i;
while p[0]=0 do
begin
        for i:= 0 to n do
            p[i] :=p[i];

        j:=N
        while p[j-1] >=p[j] do
        j :=j+1;

        k:=N;
        while p[j-1]>=p[k] do
                k:=k – 1;
        d:=p[j-1];
        p[j-1]:=p[k];
        p[k]:=d;
        for i := j to n do
                r[i] :=p[N-i+j];
        for i: = j to n do
                p[i]: =r[i];
end;
```

Сочетания – соединения, отличающиеся друг от друга по крайней мере одним элементом, каждое из которых содержит M элементов, взятых из N различных элементов, называются сочетаниями (комбинациями или выборками) из N элементов по M. Порядок следования элементов не учитывается.

Пример. Выпишем все сочетания из элементов a, b и c по два:

ab, ac, bc.

Количество способов, которыми можно выбрать M элементов из N, принято обозначать C(N,M) или Cmn(число сочетаний из N по M). Значение величины вычисляется по формуле:

$$Cmn= n!/(m!(n-m)!)$$

Размещения – соединения, отличающиеся друг от друга составом элементов или их порядком, каждое из которых содержит m (m, n) элементов, взятых из n различных элементов, называется размещениями из n элементов по m.

Пример: Имеем четыре элемента: a, b, c и d. Выпишем все размещения этих элементов по два.

ab, ba, ac, ca, ad, da, bc, cb, bd, db, cd, dc

Количество способов, которыми можно выбрать и разместить по m различным местам m из n элементов, обозначают A(n,m) или Amn(число размещений из n по m).

$$Amn=n!/(n-m)!$$

Приведем фрагмент программы, генерирующей все сочетания из n элементов по m . Так же, как и при генерации перестановок, будем работать с номерами элементов:

```
For i:= 0 to m do
        B[i] :=1;
j=100
while j<>0 do
begin
        For i:= 0 to m do
                B[i] :=B[i];
        j:=m;
        while (j>0) and (B[j] >= n +j - m) do j :=j - 1;
        if j<>0 then
                begin
                B[j]:=B[j]+1;
                for k := j +1 to m do
                B[k] :=B[k-1] +1;
            end;
end;
```

Размещения с повторениями. До сих пор рассматривались соединения, в каждое из которых любой из n различных элементов входит один раз. Можно рассматривать соединения с повторениями, т. е. соединения, в каждом из которых любой из n различных элементов может входить более одного раза.

Размещения из n элементов, в каждое из которых входит m элементов, причем один и тот же элемент может повторяться вкаждом размещении любое число раз, но не более чем m, называются размещениями из m элементов по n с повторениями.

Пример. Выпишем размещения с повторениями из двух элементов a, b по три:

aaa, aab, aba, abb, baa, bab, bba, bbb.

Количество размещений из N элементов по M с повторениями обозначают Amn(n). Справедлива формула:

$$Amn(n) = n^{b}$$

Действительно, на каждое из m мест мы можем поместить любой из n элементов.

Опишем алгоритм генерации всех размещений из n элементов по m с повторениями. Пронумеруем элементы от 0 до n-1, и в дальнейшем будем работать не с самими элементами, а с их номерами. Каждому размещению можно сопоставить число в n-ичной системе счисления, состоящее из m цифр.

Приведем фрагмент программы, генерирующей размещения с повторениями :

```
For i:= 0 to m do
A[i] :=0;
while A[m]=0 do
begin
i:=0;
        while A[i]=n – 1 do
    begin
        A[i] :=0;
         Inc(i);
        end;
        Inc(A[i]);
        for i = m-1 downto 0 do
        A[i] :=A[i];
end;
```

Перестановки с повторениями. Перестановки из n предметов, в каждую из которых входят n1 одинаковых предметов одного типа, n2 одинаковых предметов другого типа и т. д. до nk одинаковых предметов k-го типа, где n1+n2+... +nk = n, называются перестановками из n элементов с повторениями.

Пример. Получим перестановки из двух элементов a и b, каждый из которых взят по два раза:

aabb, abab, abba, baab, baba, bbaa.

Число всех таких перестановок с повторениями принято обозначать Pn(n1,n2,... ,nk). Оно может быть найдено по формуле:

$$Pn(n1,n2,\ldots,nk) = n! /(n1!n2!.....nk!)$$

Сочетания с повторениями. Сочетаниями из n элементов по m с повторениями называются соединения, содержащие m элементов (без учета порядка следования), причем любой элемент может входить в соединение некоторое число раз, но не большее m.

Пример. Получим сочетания из элементов a, b по три с повторениями:

aaa, aab, abb, bbb.

Число всех сочетаний из n элементов по m с повторениями принято обозначать Cmn(n). Оно может быть найдено по формуле:

$$Cmn(n) = (m+n-1)!/(m!(n-1)!)$$

2.1.16 Дополнительные вопросы технической подготовки

Темы, рассмотренные во второй части, не перекрывают всех вопросов, связанных с технической подготовкой ребят к соревнованиям. Мы рассмотрели их в связи с тем, что именно задачи, в которых встречаются данные мотивы, наиболее трудны для наших ребят и практически не рассматриваются ими при подготовке к олимпиадам.

Какие вопросы еще должны быть рассмотрены при подготовке к олимпиадам?

Безусловно, первое - это знание языка программирования и интегрированной среды разработчика, в частности, встроенные редактор и компилятор (желательно и отладчик). Необходимо знание опций компилятора - это также может дать определенноепреимущество (скорость выполнения, помехоустойчивость программы и т.п.).

Ребята **обязаны** уметь работать с файлами, организовывать взаимодействие с внешними программами. Отговорка - **это мы не проходили -** не оправдывает незнание и неумение.

Ребята должны уметь работать с различными типами данных, знать основные структуры и их организацию.

Уметь работать с задачами целочисленной арифметики. Иметь понятие о рекуррентных уравнениях и динамическом программировании.

Иметь понятие об оптимизационных задачах и методах их решения.

Как правило, данные, задаваемые при проведения тестирования, корректны в смысле технических ограничений, однако они могут быть логически некорректны или приводить к некорректным результатам. Поэтому важно, чтобы учащиеся могли выявлять эти случаи. Для этого хорошим методом подготовки служит разработка задач самими учащимися.

Обязательно в план занятий необходимо включить численные методы.

2.2 Тактическая подготовка

2.2.1 Десять “золотых” правил

п.1. Учитель всегда прав.
п.2. Если учитель не прав, смотри п.1.

Из правил поведения в классе

В книге “Последний круг” Олимпийский чемпион Петр Болотников в беседе с журналистом, обсуждая тактику проведения бега на длинные дистанции, рассмотрел десять “золотых” правил, предложенных одним из знаменитых стайеров. Вот некоторые из них:

1. Бежать ближе к кромке.
2. Не обгонять соперника на вираже.
3. Начав спуртовать, не снижать скорости до финиша и т.п.

Анализируя эти правила, он пришел к выводу, что часто победа достигалась тогда, когда они (эти правила) нарушались. Например: знаменитая дуэль Владимира Куца и Р. Пири на Мельбурнской олимпиаде, когда В. Куц своими постоянными рывками на дистанции (смотри правило 3) настолько измотал англичанина, что фаворит соревнований не попал даже в призеры. После этой олимпиады считалось, что тактике “рваного” бега нельзя ничего противопоставить. Однако на первенстве СССР Петр Болотников смог победить Владимира Куца, найдя “противоядие” его тактике “рваного бега”.

Нет и не может быть единых правил, безусловно приводящих к победе. В каждом случае поведение (тактика) должно строиться с учетом конкретных условий. Более того, у каждого соревнующегося вырабатывается свой стиль, своя тактика поведения. Это связано, как с его характером (психоло-

гический аспект), так и со степенью подготовленности в данный конкретный момент. Поэтому, не претендуя на истину в последней инстанции, хотелось бы предложить читателям несколько советов, к которым он (читатель) может прислушаться или оставить их без внимания.

2.2.2 Лимит времени

“Еще немного, еще чуть-чуть...”

Слова из песни

Участвуя в олимпиадах по информатике различного уровня, практически всегда я наблюдал следующую картину. После окончания тура обязательно найдется участник или участники, которые будут утверждать, что если бы у них было бы еще пять-десять минут, то они точно бы завершили программу, и она (их программа) прошла бы все тесты. Довольно часто именно в последние двадцать-пятнадцать минут у участников бывают “минуты озарения”, когда им кажется, что они нашли полное решение задачи или лучший вариант оптимизации программы. Такие участники начинают лихорадочно вносить изменения в, может быть, не совсем хорошую, но вполне работоспособную программу. Как результат – полностью неработающая программа! В чем причина и как избавиться от этой “болезни”?

Не кажется ли Вам, что эти пресловутые “пять минут” сродни цейтноту у шахматистов? Посмотрим, что они считают основными причинами появления цейтнотов и как они борются с ними.

Вот что пишет Крогиус Н.В.: “Возникновению цейтнота способствует немало объективных факторов: недостаточная теоретическая подготовка, растренированность, сложность положения на доске и значение спортивного результата данной партии. Однако основная причина цейтнота кроется в свойствах характера шахматиста, прежде всего в снижении волевых качеств и критичности мышления”. “…90% случаев цейтнота получается не от того, что игрок занят был расчетом разветвляющихся вариантов, а просто по причине жалкой нерешительности”, – утверждал гроссмейстер Нимцович А.И..

Что же предлагают шахматисты? Обратимся опять к Крогиус Н.В.: “Профилактика – это всегда конфликт между стремлением как можно глубже проникнуть в тайны позиции (*задачи* – прим. Д.А) и необходимостью считаться с регламентом времени”. Наиболее распространенными из методов профилактики цейтнотов являются следующие:

1. Оставление резервного времени.
2. Введение промежуточных контролей.
3. Обдумывание при ходе противника.
4. Быстрая игра в дебюте.
5. Метод “свертывания”.

Рассмотрим эти методы подробнее, но уже применительно к решению задач.

Оставление резервного времени. Можно рекомендовать оставлять в запасе 10-15 минут, т.е. считать, что на тур отведено не 4 (5) часов, а 3ч. 45минут (4ч. 45минут). Эти резервные минуты можно потратить на правильное оформление программы перед сдачей её членам жюри. Спешка и невнимательность при оформлении программы часто приводит к грустным последствиям. Например: случай 1 - 1996 год, в первом туре на олимпиаде ICI-96 член сборной команды Казахстана А. Тюрюшкин полностью решил задачу, но вместо работающей программы сдал один из неработающих вариантов; случай 2 - 1998 год, Всемирная олимпиада по информатике, первый тур, один из участников команды Казахстана сдает программы, записав их не под именем, указанным жюри, а под другим. В результате итог один – круглый ноль.

Введение промежуточных контролей. Этот прием заключается в примерном планировании времени на определенные действия. Например: 20 минут - на чтение задач и выбор порядка их реше-

ния; 2 часа - на разработку алгоритмов решение задач; 1 час - на реализацию программ на компьютере; 20 минут - на тестирование готовых программ.

Обдумывание при ходе противника. **Если Вы задали вопрос по задаче членам жюри, то не стоит сложа руки ждать, когда придет ответ. Это время Вы с успехом можете использовать для разбора следующей задачи или реализовать стандартные модули.**

Быстрая игра в дебюте. Это связано с хорошей технической и теоретической подготовкой, а так же с наличием "домашних заготовок", то есть использование модулей, отработанных при подготовке к олимпиаде. Чем больше их, тем меньше времени Вы затратите на изобретение "велосипеда". Например, В. Коваленко участвует в олимпиадах по информатике с 1996 года (с седьмого класса), на Республиканской олимпиаде 1998 года при разборе задач заявил, что незачем заучивать стандартные блоки дома, если проще написать их заново непосредственно во время тура. Как Вы думаете, какой результат он показал?

Метод "свертывания". В условиях приближения "цейтнота", когда нет уверенности, что задача может быть решена полностью, она решается частично для простых или особых случаев.

Еще одной причиной непродуктивного использования ограниченного времени на олимпиадах по информатике является следующее. Очень часто, написав программу, участник выясняет, что его программа не работает, что он использовал неверный алгоритм. В этом случае, вместо того, чтобы заново написать программу или неработающий блок, он пытается ее модернизировать с помощью всевозможных "подпорок" (трудно разрушать то, на что затрачено много труда и времени). Это случается, как правило, с ребятами, которые не придерживаются идеологии структурного программирования и "панически" боятся карандаша и бумаги.

В этом случае, рекомендация только одна - больше времени проводите за бумагой, а не за компьютером. **Приступайте к реализации программы на языке программирования только тогда, когда будете уверены, что предложенный вами алгоритм работает так, как надо или, в крайнем случае, если Вы другого ничего придумать не можете.** Исповедуйте структурный стиль программирования (см. п.2.1.1.).

Что делать, если Вы все-таки попали в эту неприятную ситуацию? В спортивном ориентировании есть негласное правило: **если ты заблудился на дистанции или выяснил, что путь, выбранный тобой, неверный, то вернись в исходную точку ("точку привязки").** От этой точки заново начни движение. Попытка продолжать движение по неверному пути приведет только к потере времени и сил.

2.2.3 Сколько задач решать?

На олимпиадах по информатике на один тур предлагается одна или несколько задач. Когда задача одна - вопросов нет, надо решать именно эту задачу. Что делать, если задач несколько?

Случай 1. На международной олимпиаде ICI-98 в каждом туре предлагалось по три задачи. Они имели разную стоимость в баллах, при этом решить все задачи за время, отведенное на тур, было практически невозможно.

Случай 2. Республиканская олимпиада по информатике 1993 год (г. Семипалатинск). На второй тур было предложено две задачи. Участник, лидировавший после первого тура, решил не рисковать и выбрал для решения одну задачу, имеющую максимальную стоимость в баллах. После тестирования, получив максимальный балл, он с удивлением обнаружил, что по результатам 2-х туров он оказался только на третьем месте. Два участника (М. Согрин и А. Джапаров), решавшие во втором туре 2 задачи, хоть и не получили по ним максимальные баллы, тем не менее, в сумме набрали большее количество баллов и заняли первое место.

Можно привести примеры того, когда победителем становился тот, кто из нескольких задач решил одну, но полностью.

Однако анализ результатов Всемирных олимпиад по информатике показывает, что для победы необходимо решать все задачи. Так на Всемирной олимпиаде 1998 года четыре участника показали 100% результат. Поэтому я рекомендую попытаться решить все задачи.

Здесь встает вопрос: в каком порядке их решать? Начинать ли решения с простых (имеющих наименьшую стоимость) или со сложных (имеющих наибольшую стоимость) задач?

2.2.4 Порядок решения задач.

Не плывите по течению, не плывите против течения, а плывите туда, куда Вам надо!
Козьма Прутков

Конечно, выбор количества задач и порядок их решения во многом определяется уровнем технической подготовки учащегося и теми задачами (уровень притязаний), которые он пытается решить на данных соревнованиях (олимпиаде). Однако при равных условиях выбор того или иного порядка решения задач может существенно повлиять на окончательный результат. Это связано с ограничением по времени, которое всегда присутствует на олимпиадах. Неверно выбранный порядок решения приводит к нерациональному использованию и без того ограниченного времени.

Для правильного выбора порядка решения задач необходимо провести оценку сложности задач, а также оценить сложность реализации ее в виде программы. Под сложностью задачи здесь надо понимать не их вычислительную сложность, а сложность в отношении к возможности решения и реализации в виде программы конкретной задачи конкретным участником.

Основным методом подготовки является следующий: ребятам дается одна или несколько задач. За время от 20 до 30 минут ребята должны определить, к какому виду относятся эти задачи, и предложить методы, которые, на их взгляд, можно использовать при решении данных задач. После этого проводится детальный разбор задач.

2.2.5 Логические несоответствия

В последнее время идет тенденция на то, что в тестовых заданиях все входные данные не противоречат техническим ограничениям, поэтому при написании программы не требуется проверка входных данных на корректность, т.е. соответствие их техническим ограничениям. Однако надо помнить, что помимо технической некорректности входных данных, могут присутствовать и логические несоответствия, т. е. при входных данных, отвечающих техническим ограничениям, задача может не иметь решения или числовое решение будет бессмысленным. Приведем несколько примеров. Задача “Тесей и Минотавр”. Если Тесей и Минотавр изначально стоят на полях разной четности (см. часть 1), то ни при каких условиях они не могут оказаться в одной клетке одновременно. Задача “Лорен-Дитрих”. Если количество бензина в баке меньше или равно количеству бензина, которое машина тратит на 1 км. пробега, то задача явно не будет иметь решения. Можно и дальше приводить аналогичные примеры. Такие логические несоответствия лежат на поверхности и их довольно легко выявить. Наиболее трудным является определение скрытых логических противоречий. В качестве примера приведем задачу, предложенную на Республиканской олимпиаде школьников по информатике 1994года в Республике Беларусь.

Арбузы.

В ряд лежат N арбузов, пронумерованных от 1 до N. Нам известно, что:

1) массы первого и N-го арбузов m(1) и m(N) соответственно;

2) масса i-го арбуза m(i) есть среднее арифметическое масс двух соседних арбузов, увеличенное на d:

$$m(i)=d+(m(i-1)+m(i+1))/2;$$

По введенным m(1), m(N), N, d и j найти m(j).

Ограничение: N<200.

Необходимо, в рамках формулировки задачи, предусмотреть проверку корректности данных программы.

Ниже приведены тесты для проверки правильности работы программы

	m1	mN	N	d	j	ответ
1)	1.0	1.0		21	1.0	15 85.0
2)	30.0	210.0		10	-5.0	8 данные некорректны
3)	40.0	58.0		10	-4.0	2 данные некорректны
4)	40.0	1370.0		20	-5.0	13 460.0
5)	1.0	3.625	22	0.9375	14	100.125
6)	50.0	8.5		84	0.25	59 383.5
7)	1.0	1.0		10	0.0	3 1.0

Интерес представляют второй и третий тесты. Оказывается, что при вычислении массы арбуза с этими входными данными в тесте номер 2 вес 8-го арбуза отрицательный, чего не может быть. В третьем тесте, несмотря на то, что вес 2-го арбуза положителен, данные также некорректны. Это связано с тем, что при вычислении массы других арбузов так же получаются отрицательные величины.

2.2.6 Метод исключения

Диалог на экзамене.
Преподаватель: “Кто читал Вам лекции?”
Студент: “....?!”
Преподаватель: “Наводящий вопрос. Вы или Я ?”

Из студенческого фольклора

В качестве примера рассмотрим задачу, предложенную на олимпиаде ICI-94 (г. Могилев, Республика Беларусь):

На бесконечном компьютерном холсте нарисованы контуры геометрических фигур. Каждый контур обозначен каким либо символом. Контуры фигур не пересекаются. Требуется написать такую программу, которая определяет, какие фигуры нарисованы. Фигура может быть треугольником, прямоугольником или окружностью.

Авторский вариант решения данной задачи предусматривал обход фигуры по контуру с анализом угла поворота и был достаточно сложен. Практически никто из ребят не смог решить задачу в общем виде. Однако при тестировании работ были получены интересные результаты. Программа одного из участников очень точно определяла наличие на холсте треугольника. Он (участник) определил, что признаком принадлежности фигуры к треугольнику является наличие определенного сочетания закрашенных квадратиков. Вместе с тем, при обсуждении решения задачи, было выявлено, что окружность тоже имеет свои характерные черты (связанные с симметрией) и также может быть идентифицирована с большой степенью точности.

Учитывая это, можно было бы предложить следующую структуру программы:

1. Ввод данных
2. Определение фигуры
3. Обработка частных случаев (случаи вырожденных фигур)
   если нет, то
   Определение принадлежности фигуры к треугольнику
   если нет, то
   Определение принадлежности фигуры к окружности
   если нет, то

фигура - прямоугольник.
4.Если еще необработанные фигуры?
если да, то на 2.
5. Окончание работы.

Как Вы заметили, здесь был использован структурный подход. При этом использовались две независимые машины. Одна определяла принадлежность фигуры к треугольнику, другая - к окружности.

2.2.7 Табличный вариант

Для увеличения быстродействия программ иногда лучше не вычислять некоторые промежуточные значения, а использовать табличные данные, т.е. заранее рассчитать некоторые необходимые Вам величины, когда Вы не ограничены во времени (во время отладки программы), и использовать их в рабочей программе.

Задача первого тура олимпиады ICI-98:

На шахматной доске размера NxN, 0<N<10 расставить N фишек так, чтобы расстояния между фишками были попарно различными. Мы считаем, что каждая фишка имеет точечные размеры и находится в геометрическом центре того поля, на котором она стоит, а расстояние между двумя фишками измеряется вдоль прямой, проходящей через центры занятых фишками полей.

В ответ на приглашение программы пользователь вводит значение N. После чего программа создаёт текстовый файл {test.dat}, в который записывает в первую строку число N, а затем координаты N клеток, в которых стоят фишки, составляющие решение поставленной задачи - по две координаты (номер строки и номер столбца) в строке, разделённые пробелами. Координаты представляются целыми числами от 1 до N и отсчитываются от левого верхнего угла доски, т.е. координаты левого верхнего угла - (1,1), а правого нижнего - (N,N).

Если решения не существует, программа должна выдать на экран соответствующее сообщение.

Данная задача решалась перебором вариантов. Оказалось, что для N от 2 до 7 решение существует, а при N = 8 задача решения не имеет, более того, если просматривать всевозможные варианты (при N = 8), то есть провести полный перебор, то время работы программы не уложится в контрольное время.

Что можно было сделать в данном случае?

При проверке правильности работы программы после ее написания, нужно было протестировать ее по всем значениям N (согласно техническим ограничениям N <10). В случае, если для каких-либо значений N время работы программы превышало контрольное, в программу нужно было внести дополнительный модуль обработки этих особых случаев.

Ввод N
Если N=8, то выдать в файл “Нет решения”
иначе
Работает основная программа.
Выход

То есть, если на входе N = 8, то без проведения поиска выдать сообщение: “Нет решения”. Такой подход считаю вполне правомерным и моральным.

Помните, что при отладке программы Вы не регламентированы временем.

2.2.8 Что делать, если мы не знаем, как решить задачу, или нам это кажется?

Никто не одолеет тебя, пока ты сам не дашь себя одолеть.

С Томпсон

Случай 1. Ученик знает, как решить задачу, но не может реализовать ее в виде программы.

Случай 2. Ученик не может найти путь к решению задачи или ему это кажется.

И первый и второй случаи связаны с недочетами в технической подготовке ученика в подготовительном периоде.

Если в первом случае можно только пожалеть учащегося, то во втором случае не все еще потеряно.

Предлагаю несколько приемов, которые, возможно, помогут Вам:

Прием 1. Поэтапное решение задачи. Постарайтесь разбить задачу на более простые подзадачи и ищите решение для этих более частных задач. Этот подход мы использовали при решении задач по определению геометрических фигур (см. 2.2.6. Метод исключения).

Прием 2. Рассмотрите самые простые случаи, которые могут быть. Постарайтесь решить их "вручную". То есть, представьте себя машиной (программой). Попытайтесь описать простыми словами те действия, которые необходимо произвести. Усложните задачу, попытайтесь еще раз решить ее, используя только голову, карандаш и бумагу. После того, как Вы продумали варианты решения для самых простых и особых случаев, попытайтесь определить то общее, что их объединяет. Возможно, это подтолкнет Вас к идее общего алгоритма решения задачи. Если такого не случится, то, по крайней мере, у Вас будет несколько независимых "машин", которые, если и не перекрывают все возможные варианты решения задачи, позволят Вам избежать нулевой оценки.

Прием 3. Помните слова С. Томпсона: "В то время, когда все кругом будет казаться тебе мрачным, оставайся твердым, спокойным и добрым, и непременно случится что-нибудь, что приведет все снова в порядок. Выход всегда есть, и мужественное сердце всегда найдет его".

2.2.9 "Золотые" правила, или это должен знать каждый!

Не отдавай приказ, если не уверен, что его выполнят.

Наполеон Бонапарт

Во время участия на олимпиаде, я советую школьникам придерживаться следующего порядка решения олимпиадных задач:

Этап 1. **Подготовительный**

— Внимательно прочитать задачу (все задачи).

— Определить неясные для Вас места в формулировке задачи (задач).

— Вторично прочитать задачу (задачи).

— Правильно сформулировать вопрос к членам жюри по условию задачи.

Примечание: В самом начале тура полезно набить приведенную ниже универсальную заготовку для решения олимпиадной задачи (она представляет из себя работающую программу!!!).

```
var
  i,j,k:longint;
procedure readdata;
begin
  assign(input,'');
  reset(input);
```

```
end;
procedure outdata;
begin
  assign(output,'');
  rewrite(output);
  close(output)
end;
procedure initial;
begin
  {блок инициализации глобальных переменых}
end;
procedure run;
begin
end;
begin
   readdata;
   initial;
   run;
   outdata
end.
```

Далее можно скопировать эту заготовку столько раз, сколько задач предложено на туре и сразу назвать каждый файл так, как это требуется по условиям олимпиады. В результате вам не придется при переходе от решения одной задачи к другой начинать работу с нуля

Этап 2. **Выбор порядка решения задач**

Приблизительно определить сложность задачи (включая и техническую реализацию). Для этого провести анализ каждой задачи, который должен включать следующие пункты:

а) Определить, хотя бы примерно, к какому классу относится задача, вспомнить примеры задач этого класса или аналогичных ему.

б) Предложить возможные пути решения задачи (два-три подхода).

г) Определить, как влияют технические ограничения на использование тех или иных технических приемов и методов.

Этап 3. **Решение задач**

— Еще раз внимательно прочитать условие задач.

— Предложить тесты (5-7) для проверки правильности алгоритма и его эффективности.

— Рассмотреть возможные пути решения задачи (два-три подхода).

— Обратить внимание на технические ограничения.

— Рассмотреть частные и особые случаи.

— Определить случаи логического несоответствия.

— Определить, на какие подзадачи нужно (можно) разбить исходную задачу.

— Словесно описать схему решения задачи (на бумаге).

— Провести трассировку предложенного Вами алгоритма на тестах.

— Определить, в каких случаях алгоритм работает, а при каких условиях работа его некорректна.

— Постараться так модернизировать алгоритм, чтобы проходило как можно больше из тестов, предложенных Вами.

Примечание: Попытаться найти на бумаге (!!!) точное решение, возможно только для малых размерностей. Такой подход зачастую позволяет обнаружить закономерности, которые затем можно попытаться распространить и на общий случай. Отобразить на бумаге принципиально различные

случаи, в том числе и вырожденные. Это поможет при составлении тестов для самопроверки написанной программы.

Этап 4. **Реализация задачи на языке программирования.**

— При реализации обратить внимание на формат входных и выходных данных.

— После написания программы провести тестирование с использованием предложенных Вами тестов.

— Оценить время прохождения тестов (определить, укладывается ли программа в отведенный лимит времени).

Примечанее:

1. Запрограммируйте решение задачи в виде вызовов процедур и функций, которые пока следует описать в виде “заглушек” . Таким образом вы сможете отладить логику вашей программы, которая опять же должна оставаться “работающей”.

2. Затем следует по одной набивать и отлаживать уже описанные процедуры и функции, добиваясь, чтобы свое действие каждая из них выполняла правильно в любом случае. например, поиск максимумов, сортировка массивов, комбинаторные подпрограммы и т.п. должны работать корректно при любых входных параметров, вне зависимости от программного контекста, из которого они будут вызываться.

Этап 5. **Сдача работы**

Перед сдачей работы проверить правильность имени программы (соответствие требованиям жюри) и версии программы, сдаваемой Вами на проверку.

Внимание!!!

Никогда не сдавайте программу, если вы не провели её тестирования. Ваша 100%-я уверенность в ее работоспособности и работоспособность программы - разные вещи.

2.3 Психологическая подготовка

2.3.1 Предметные олимпиады - это спортивные соревнования

Настоящая математическая проблема отличается от олимпиадной задачи только тем, что над первой можно думать в тысячу раз больше.

Б.Н. Делоне

Нередко мне приходилось слышать сетования учителей о том, что их ученик хорошо пишет программы, прекрасно знает язык программирования, а вот на олимпиадах не может показать себя. До настоящего времени, к сожалению, еще бытует мнение, что в предметных олимпиадах могут участвовать все, кто имеет определенные знания по конкретному предмету. Считается, что если он (ученик) много знает и умеет, то это является залогом успеха.

На мой взгляд, это является следствием того, что к предметным олимпиадам относятся без учета фактора соревнования. Участие в предметных олимпиадах любого уровня - это не просто решение задач (или задачи) за определенное время, а работа в сложных, подчас экстремальных условиях при лимите времени. Предметные олимпиады - это спортивные соревнования - вот основная мысль, которую, подчас, не принимают во внимание при подготовке и участии в них. Здесь идет борьба не только интеллектов, но и борьба нервов.

Поэтому успех учащегося или его неуспех, помимо технической подготовки (уровень знаний учащихся сейчас достаточно высок и примерно одинаков), зависит от целого ряда условий, в том числе и от особенностей процессов обучения, воспитания, тренировки и подготовки к соревнованиям. Их необходимо строить не только на основе общих психолого-педагогических закономерностей, но и с учетом индивидуальных свойств нервной системы и темперамента учащегося.

Пример, результаты выступления команды Казахстана на Всемирной олимпиаде по информатике в 1999 году. Когда технически самый слабый член команды завоевывает первую для Казахстана бронзовую медаль, а ребята, которые на тренировках решали и более сложные задачи остались за бортом призеров.

При подготовке к олимпиадам было бы желательно использовать тот богатый опыт, который накоплен спортивными психологами. Так, например, ими отмечено, что лица, отличающиеся слабостью возбудительного процесса, ухудшают результаты в особо ответственных соревнованиях. Мотивация одинаковой силы по-разному влияет на лица с сильной и слабой нервной системой. Чем сильнее нервная система, тем в большей степени сказывается положительное влияние активности мотивации, и наоборот, чем слабее нервная система, тем больше ее отрицательное влияние. Для спортсменов со слабой нервной системой необходимо акцентировать основное внимание не на результатах, а на технически правильном выполнении действий.

Следовательно, чтобы в процессе обучения или тренировки, при подготовке к олимпиадам или непосредственно в ходе их, привести к одинаково высокому уровню деятельность ребят, требуются разные педагогические воздействия для каждого из них - в зависимости от особенностей темперамента.

2.3.2 Соревновательный стресс

Можно говорить, что “ценность” олимпийца в значительной мере определяется его терпимостью к стрессу, т.е. уровнем достижений, которого он способен добиваться в условиях напряженной соревновательной борьбы.

Соревновательный стресс - это психическое состояние, возникающее в трудной ситуации соревнований при обязательном наличии мотивации значительной силы, высокой ответственности за данное выступление.

Трудность. Трудность - одно из обязательных условий, определяющих возникновение стресса. Даже небольшое повышение трудности задания ведет к возникновению стрессового состояния, особенно у лиц со слабой нервной системой.

Угроза. Одной из особенностей психического стресса является возникновение угрозы. Ожидание угрозы понимается как предвосхищение человеком некоторого будущего столкновения с какой-либо опасной для него ситуацией и оценкой ее. Например, ожидание неудачи при решении задачи и реакции на это окружающих (одноклассников, учителей и т.п.). Большая предрасположенность к возникновению угрозы в ответственной и напряженной ситуации определяет большую подверженность лиц со слабой нервной системой отрицательному воздействию стресса.

Примером этого может служить случай произошедший на Республиканской олимпиаде 2000 года. Одна из участниц соревнований, имевшая после первого тура четвертый результат, во втором туре просто не могла заставить себя работать за компьютером, хотя решение задач она знала. Боясь допустить ошибку и реакции родителей, как потом я выяснил, сбежала с тура, в результате ноль баллов и лишь седьмое(!!!) место.

Активность мотивации. Возникновение и динамика стресса связаны с активностью мотивации (активным отношением личности). Этот фактор соревновательного стресса влияет на эффективность деятельности в зависимости от свойства нервной системы - силы возбуждения. Под влиянием высокоактивной мотивации существенно изменяются показатели деятельности (у лиц со слабой и сильной нервной системой - в разных направлениях). Таким образом, для повышения эффективности работы в соревнованиях необходимо дозирование мотивации по силе с учетом типологических свойств нервной системы ребят.

Интересны результаты исследований по влиянию уровня стресса на результаты. Оказывается, что люди высокоактивные достигают наилучших результатов при низком и высоком уровне стресса, а худшие - при среднем уровне. Такая же картина и для низкореактивных людей.

Таким образом:

1. Для достижения высокого результата в условиях выраженного соревновательного стресса олимпийцу необходим оптимум активности отношения. Стимуляция его мотивацией должна быть дозирована с учетом свойств его темперамента.

2. В зависимости от активности отношения личности одни и те же свойства темперамента могут оказывать на успех в деятельности как положительное, так и отрицательное влияние. Тревожность при низкой мотивации повышает общую активность и способствует достижению высокого результата. При высокой мотивации тревожность действует угнетающе.

2.3.3. Управление стрессом

В спортивной практике разработан и применяется целый ряд приемов и методов регулирования уровня стресса. Одни из них используются до начала соревнований, с помощью других корректируется состояние перед началом и в ходе состязаний. Было бы желательно использовать их и в практике подготовки и участия в предметных олимпиадах. Рассмотрим основные приемы регулирования уровня стресса.

Целенаправленное изменение направления мыслей. Этот прием часто называют “отвлечением” или “переключением”. Известно, что тревожное состояние, возникающее накануне соревнования, борьба с волнением отнимают подчас больше нервной энергии, чем сами соревнования. Если отвлечься от навязчивых мыслей, можно сохранить нервный потенциал. Переключение достигается занятием делом, которое полностью или почти полностью занимает мысли (чтение интересной книги, прослушивание музыки и т.п.). **Одним из приемов изменения динамики мышления является акцент в установке на максимально правильное (техническое) выполнение задания, а не на предполагаемый спортивный результат. Забота о том, как лучше достичь победы, а не о том, какой результат покажет, какое место займет учащийся, способствует оптимизации психического состояния.**

Участник команды Казахстана Берлизев А. в первом туре международной олимпиады ICI - 94 (г. Могилев, Республика Беларусь) выступил неудачно и по итогам первого тура занимал одно из последних мест. После подведения итогов первого дня соревнований и анализа действий участников нашей команды, было решено, что основная ошибка - попытка работать на максимальный результат. При этом были допущены явные промахи в технологии решения задач. Мы попросили ребят во втором туре не стремиться показать максимальный результат, а максимально точно придерживаться технологии решения задач. Были обсуждены все этапы написания программ. Более того, в качестве ориентира был подготовлен повременный график (см. тактическая подготовка). Результатом стало то, что из семи участников, получивших наибольшее количество баллов во втором туре, были все наши ребята (пять человек). Команда Казахстана заняла первое место в личном и командном зачете, а Берлизев А. вошел в десятку сильнейших.

Воздействие на внешнее проявление стресса. Состояние стресса, особенно повышенного уровня, отчетливо проявляется внешне: повышенная двигательная активность, суетливость, жестикуляция, мимика и т.п. Если сознательно усилием воли сдерживать эти проявления, то это вызывает усиление процессов торможения в головном мозге, что способствует ослаблению возбуждения и понижению уровня стресса. Наоборот, повышение двигательной активности, усиление жестикуляции способствует повышению уровня возбуждения.

Произвольное переключение внимания на раздражители различного эмоционального значения. Значительное влияние на психическое состояние оказывает окружающая среда (погодные условия, место соревнований, действия участников и т.д.). Часть из них, повышая уровень возбуждения, являются дополнительными стрессорами, часть из них снижают уровень возбуждения. Используя их, можно добиваться оптимального уровня стресса.

Интересный эпизод произошел на третьем этапе республиканской олимпиады по информатике 1998 года. Один из участников команды РСФМШИ (г. Алматы) часто поглядывал на листок бумаги, где крупными буквами было написано: НЕ ВОЛНУЙСЯ. ВСЕ ХОРОШО! Это помогло ему справиться с волнением.

Применение специальных дыхательных упражнений. Как перед началом соревнований, так и в ходе их, для снижения чрезмерного стрессового состояния можно рекомендовать комплекс специальных упражнений на задержку дыхания. Пример упражнений, предложенных Л.Персивалем.

Упражнение 1. Глубокий вдох, задержать дыхание и слегка напрячь мышцы всего тела. Дыхание задержать на 5-6 секунд, затем сделать медленный выдох и расслабить мышцы тела. Повторить упражнение 9-10 раз, стремясь с каждым разом увеличить время задержки дыхания, выдоха и расслабления.

Упражнение 2. Сделать несколько медленных, глубоких и ненапряженных вдохов. При вдохе слегка напрягать все мышцы, при выдохе стремиться полностью расслабится. Выполнять упражнение в течение 2-3 минут.

Среди других способов регулирования уровня стресса можно отметить *внушающее воздействие тренера* и *аутогенную тренировку.*

Большое значение для эффективности педагогического внушения тренера имеет его культура, авторитет, профессиональная интуиция, артистизм и т.п.

Аутогенная тренировка - метод регулирования психического состояния с помощью самовнушения. Один из вариантов этого метода разработан специально для спортивной практики и получил название *психорегулирующей тренировки.*

Проведение тренировочных занятий, моделирующих соревновательные условия, - одна из форм подготовки ребят к работе в условиях стресса, выявление их реакции.

2.3.4 Отбор при подготовке к соревнованиям

Необходимость учета особенностей темперамента наиболее остро проявляется во время подготовки к ответственным соревнованиям. Некоторые типы темперамента, его отдельные свойства и взаимоотношения между ними при соответствующих условиях, не позволяют показывать максимально возможный результат. Тревожность, эмоциональность, импульсивность, синергические отношения между активностью и реактивностью при высокой активной мотивации - все это не способствует достижению успеха в ответственных соревнованиях. Деятельность таких олимпийцев недостаточно надежна. Добиваясь хороших результатов в менее ответственных соревнованиях и тренировках, они, как правило, подвержены срывам на ответственных соревнованиях. Другие ребята именно в условиях ответственных соревнований достигают высоких результатов. Так двукратный победитель Республиканских олимпиад (1993-1994 гг.), победитель Международной олимпиады ICI-94 (г. Могилев, Республика Беларусь), ученик 1-й городской гимназии г. Темиртау Михаил Согрин на школьных, городских и областных олимпиадах занимал, как правило, второе место. Преодоление отрицательного влияния темперамента можно достигнуть путем индивидуализации учебно-тренировочного процесса.

Тренер-педагог обязан досконально изучить особенности своих питомцев, построить их подготовку к соревнованиям с учетом этих знаний таким образом, чтобы их способности смогли раскрыться всесторонне и широко.

Психологические особенности ребят, их темперамент необходимо учитывать и при формировании команды. Хорошие взаимоотношения в команде являются дополнительным условием повышения результатов ребят. Взаимная симпатия, желание помочь товарищам, чувство взаимной ответственности - все это повышает психологический комфорт, способствует более успешному выступлению как команды в целом, так и каждого участника. Если участники команды разобщены, видят в своих партнерах по команде соперников, а не соратников, то можно с уверенностью сказать, что все они выступят ниже своих возможностей. Примером тому может служить выступление сборной команды

Казахстана на ICI-94 (лучший вариант сборной, с которой мне приходилось работать) и выступление команды РСФМШИ на Республиканских олимпиадах по информатике в 1997 и в 1998 годах (очень неудачные выступления).

Часть III Проведение олимпиад по информатике

3.1 Личные соревнования

Олимпиадное движение по информатике развивается стремительно. Постоянно совершенствовались правила и методика оценки результатов. Если на первых порах правильность задачи определялась визуально (по листингу программы), то в настоящее время оценка работоспособности программы осуществляется путем тестирования, что уменьшает субъективность при оценке решений, устраняя "человеческий" фактор. В последнее время на международных олимпиадах тестирование проводится с применением тестирующих программ, которые в автоматическом режиме проверяют работу программ, написанных участниками. Это повышает требования к формату вывода результатов работы программ.

Учитывая опыт участия и проведения международных, республиканских и городских олимпиад по информатике, можно предложить следующие рекомендации:

Рекомендации к составителю задач

Во многом, качество проведения олимпиад по информатике зависит от тех задач, которые будут предложены участникам. Поэтому я полностью разделяю мнение Д.В. Фомина, которое он высказал по поводу олимпиадных задач по математике: "Придумать хорошую олимпиадную задачу совсем непросто. Желательно, чтобы задача была неизвестной, идея решения не должна быть избитой и наскучившей, а формулировка слишком длинной, не вызывающей интереса. Идеальная олимпиадная задача имеет яркую, запоминающуюся формулировку. Сам факт, который требуется доказать, должен удивлять и заинтересовывать, основная идея решения должна быть свежей и неожиданной. При этом желательно, чтобы задача не была чрезвычайно сложной и не давала бы преимущества участникам, знающим многое сверх школьной программы".

Желательно, чтобы задача была построена так, чтобы она давала возможность "многоуровнего" решения, то есть:

- уровень 1 - задача решена для тривиальных случаев;
- уровень 2 - задача решена для частных случаев;
- уровень 3 - задача решена в общем виде;
- уровень 4 - задача решена в общем виде, с использованием эффективного алгоритма.

Для каждой задачи в условии должно быть указано ограничение времени тестирования. Рекомендуемые временные ограничения 1 —2 секунды, исключение составляют задачи связанные с обработкой больших массивов или в считыванием и записью большого количества данных с внешнего носителя.

В последнее время идет тенденция на то, что в тестовых заданиях все входные данные не противоречат условиям задачи (техническим ограничениям), поэтому при написании программы не требуют проверку входных данных на корректность.

На республиканских, международных олимпиадах все входные данные берутся из файлов, и результаты работы программы такжезаписываются в выходной файл. Поэтому рекомендуется дать формат входного и выходного файлов.

Количество задач, предлагаемых на 1 и 2 туры, может быть различным, однако их общая сложность (сложность задач каждого из туров), так как туры равнозначны, должна быть примерно одина-

кова. Желательно на районных (городских) олимпиадах предлагать 2-3 задачи на каждый тур. При использовании нескольких задач, необходимо выполнять следующие требования:

1 Задачи должны быть разнообразными по тематике.

2. Необходимо проследить за разнообразием идей.

3. Туры должны быть тщательно сбалансированы по сложности задач. Необходимо одновременно добиться того, чтобы почти каждый участник олимпиады решил хоть что-то и чтобы наиболее сильным школьникам было над чем поломать голову.

Все предлагаемые задачи должны быть решаемы на «Стандартном» Паскале, Си . Никакие специальные возможности этих сред программирования не должны требоваться.

Рекомендации членам жюри по проведению личных туров

Перед началом соревнований все участники и представители команд должны быть ознакомлены с правилами проведения олимпиады. Особое внимание следует обратить на правила поведения участников во время тура и порядок проверки работ.

Пред началом тура жюри определяет количество задач, а также стоимость каждой задачи в баллах.

При проведении олимпиады по информатике задачи имеют, как правило, алгоритмическую природу. Таким образом, основным является разработка корректного и эффективного алгоритма. Поэтому, в отличие от других предметных олимпиад, возможно предоставление одинаковых задач для учащихся 9— 11-х классов.

Тексты задач с указанием их стоимости (в баллах) должны быть подготовлены для каждого участника.

Тексты задач для данного тура должны находится у компьютера в конверте, который вскрывается участником по стартовому сигналу.

Задание устных вопросов по условию задачи не допускаются.

Переговоры между участниками не допускаются.

Отвечать на вопросы участников по условию задачи, для устранения разночтений, имеет право автор задачи или председатель жюри.

Текст вопросов одного из участников и ответы на них не должны быть читаемы другими участниками. Для этого могут быть использованы конверты участника (с шифром), в которых находятся условия задач.

Рекомендации членам жюри по подготовке тестов

Тесты, по которым проводится проверка работоспособности программ, должны отвечать следующим требованиям:

1. Должны быть полными.

2. Не должны дублировать друг друга.

Полнота тестов предполагает, что они полностью покрывают возможность “многоуровнего” решения, то есть:

вид 1 - тесты для тривиальных случаев;

вид 2 - тесты для частных случаев;

вид 3 - тест для проверки решения задачи в общем виде;

вид 4 - тест для проверки решения задачи в общем виде, с использованием эффективного алгоритма;

вид 5 - тест на отсутствия решения.

В некоторых задачах большое значение играет эффективность. Но и в этих случаях всегда должен присутствовать хотя бы один тест. Такой, чтобы правильная, но неэффективная программа также получила какие-то баллы.

Тесты могут быть подготовлены заранее или во время проведения тура. Для каждого теста определяется его цена в баллах в соответствии с ценой задачи.

Количество тестов и их цены являются секретными и оглашаются только при начале тестирования.

Проверка работ

В последнее время проверка правильности выполнения работы проводится путем тестирования программ, при этом листинг программы не рассматривается. Это связано как с повышением сложности предлагаемых задач, так и с необходимостью уменьшения влияния "человеческого фактора". Более того, вряд ли можно найти специалиста, который бы в совершенстве владел всеми языкамипрограммирования, которые используются на олимпиадах, и смог бы свободно по листингу программы определить ее работоспособность и эффективность. Особенно сложным, при ручной проверке работ, является определения степени (глубины) решения задачи.

Перед проведением тестирования участнику сообщается количество тестов и стоимость каждого теста.

Тестирование программы состоит из запуска нескольких тестов с секретными входными данными. Запуск тестов состоит строго из следующих шагов:

1. Входные данные помещаются в директорию задачи.
2. Запускается (без параметров) тестируемая программа в директории задачи.
3. Выполняются действия, заданные в задачи.
4. Если тестируемая программа завершается "нормально" (с кодом возврата 0) в отведенное ей время, то проверяется ее вывод. В противном случае программа прерывается, и вывод не проверяется.

Баллы за каждый пройденный тест суммируются и образуют окончательную оценку за эту задачу.

Если формат выходного файла не соответствует описанному в условии задачи формату, то тест не засчитывается.

Не допускается «частичное» прохождение теста.

Как правило, тестирование на отсутствие решения проводится последним, при условии, что хотя бы один тест участника получил положительную оценку.

Определение победителя

В отличии от олимпиад по общеобразовательным предметам, проводимым в нашей Республике, определение победителя в олимпиадах по информатике происходит по сумме двух туров. При этом победителем становится участник набравший максимальное количество баллов, вне зависимости от процента от общего количества баллов представленных в задачах.

3.2 Командные соревнования

Одной из интересных форм проведения олимпиад по информатике являются командные соревнования. Они интересны тем, что здесь проявляется умение ребят работать в коллективе.

Более того, я считаю, что именно в командных соревнованиях проявляются лучшие стороны ребят. Они как бы дополняю друг друга. При этом, общение ребят в соревновательной обстановке позволяет более слабым ребятам, наблюдая за работой лидеров команды, поднимать свой уровень. Это позволяет правильно распределить работу между участниками команды.

Командные соревнования имеют две основных формы.

Первая форма. Каждой команде дается для решения 5-8 задач, время на их решение стандартное 4-5 часов. Ребята работают вместе, распределяют, кто какую задачу готовит. При этом можно работать над одной задачей и вместе. Победителем является та команда, которая решит наибольшее количество задач, то есть результат определяется, как и в личных соревнованиях, по количеству про-

шедших тестов и стоимости этих тестов в баллах. Такие соревнования очень полезно проводить в подготовительный период, как одно из средств подготовки непосредственно к олимпиаде.

Вторая форма проведения командных туров заключается в предоставлении команде задачи на разработку программ стратегий. Такая форма проведения командных туров на протяжении последних 5 лет практикуется на традиционных Международных соревнованиях ICI проводимых в Белоруссии. Задания могут даваться ребятам заранее (за один или два дня до начала командного тура), или если задачи не очень сложные - непосредственно в день соревнований. После реализаций программ "стратегий" каждой командой, между ними проводится турнир. Победителем становится та команда, программа "стратегия" которой выигрывает турнир. Как правило, выявление лучшей программы проводится в присутствии ребят.

Вторая форма командного тура более трудоемка для проведения, так как требует наличия программы арбитра, однако более эмоционально окрашена.

ВМЕСТО ЗАКЛЮЧЕНИЯ

Основной целью олимпиад по информатике является выявление талантливой молодежи, привлечение как можно большего числа учащихся к важной для научно-технического прогресса сфере человеческой деятельности, пропаганда лучших достижений в области информатики, вычислительной техники и технологии программирования.

К сожалению, жива еще "традиция" посылать на олимпиады, особенно районного уровня, неподготовленных ребят. Это является следствием "галочной" системы, когда преподавателей- предметников обязывают выставлять участников. Психологическая травма, которую может получить такой ребенок, не только отобьет желание дальнейшего участия в каких-либо предметных олимпиадах, но и может привести к снижению интереса к изучению данного предмета.

Учителя, а особенно "функционеры" от образования, должны понять то, что подготовка олимпийца – трудоемкий и долгий процесс. Отбор ребят и подготовка к олимпиадам должна начинаться задолго до их участия в официальных соревнованиях (с 6-7-го класса). Успеха добьется только тот педагог, который будет работать с ребятами постоянно и целенаправленно. В этом случае участие в предметных олимпиадах повышает интерес, способствует более сознательному и глубокому изучению школьного предмета.

Примером может служить постановка такой работы в 1-й городской гимназии города Темиртау. Здесь разработана специальная методика подготовки олимпийцев. Она начинается с седьмого-восьмого класса. В команду для участия в городских и областных олимпиадах по информатике включаются не только ученики десятых и одиннадцатых классов, но и наиболее сильные ребята из восьмых и девятых классов, которые получают бесценных опыт участия в официальных соревнования. При этом выявляется способность ребят работать в экстремальных условиях. Ребята целеустремленно и настойчиво овладевают знаниями, потому что с ними занимаются энтузиасты своего дела – их учителя. Результат такого плодотворного сотрудничества налицо - в течение пятнадцати лет ученики этой школы регулярно становятся победителями и призерами Республиканских олимпиад по информатике.

Приятно, что этот подход начинает культивироваться и в других школах и регионах нашей Республики, в частности, в Республиканской специализированной физико-математической школе. Бесценный опыт 1-й городской гимназии г.Темиртау еще раз подтверждает истину: НЕТ НЕТАЛАНТЛИВЫХ УЧЕНИКОВ – ЕСТЬ БЕСТАЛАННЫЕ УЧИТЕЛЯ.

ПРИЛОЖЕНИЕ

Типовое положение о проведении олимпиад по информатике

Общие положения

1. Олимпиада по информатике проводится в два тура. Оба тура равноценны. При проведении олимпиады по информатике задачи имеют, как правило, алгоритмическую природу. Таким образом, основным является разработка корректного и эффективного алгоритма. Поэтому, в отличие от других предметных олимпиад, для учащихся 9-х - 11-х классов предложены одинаковые задания.
2. На каждый тур представлено по три задачи. Время, отводимое на каждый тур 4 часа (астрономических).
3. Основными языками программирования являются Pascal, Си и C++.
Использование других языков программирования не допускается
4. Использование дополнительных модулей, если это не оговорено условиями задачи, не допускается.

Порядок проведения тура

1. Тексты задач с указанием их стоимости (в баллах) должны быть подготовлены для каждого участника.
2. Тексты задач для данного тура находится у компьютера. Ознакомление участника с условием задачи начинается только по стартовому сигналу.
3. Отвечать на вопросы участников по условию задачи, для устранения разночтений, имеет право председатель жюри, в течении первого часа с начала тура. Ответы жюри могут быть: “Да”, “Нет”, “Без комментариев”
Задание устных вопросов по условию задачи не допускаются.
Переговоры между участниками не допускаются.
Текст вопросов одного из участников и ответы на них не должны быть читаемы другими участниками.
4. После сигнала об окончании тура участники должны прекратить работу на компьютере.
Изменение в программе после окончания тура не допускается
5. Сдача работ происходит путем переноса текста программы и исполняемого модуля на компьютер (дискету) жюри, для последующей проверки. Отметка о сданных программах делается в специальном протоколе.
Название программы должно соответствовать указанному в условии задачи, отклонение не допускается.
6. За нарушения правил поведения участник, по решению жюри, может быть снят с олимпиады, а его результат аннулируется.

Проверка работ

Проверка работ может осуществляться без участия участников олимпиады. В этом случае, по каждому тесту, если он не засчитан, в рабочий протокол заносится в развернутом виде причина, по которой тест не засчитан. Например: “Превышение времени”, “Неверный результат”, “Нарушение формата вывода”, “Имя программы не соответствует условию” и т.д.

Тестирование программы состоит из запуска нескольких тестов с секретными входными данными. Запуск тестов состоит строго из следующих шагов:
1. Входные данные помещаются в директорию задачи.
2. Запускается (без параметров) тестируемая программа в директории задачи.

3. Выполняются действия, заданные в задачи.
4. Если тестируемая программа завершается “нормально” (с кодом возврата 0) в отведенное ей время, то проверяется ее вывод. В противном случае программа прерывается, и вывод не проверяется.

Баллы за каждый пройденный тест суммируются и образуют окончательную оценку за эту задачу.

Тестирование на отсутствие решения проводится последним, при условии, что хотя бы один тест участника получил положительную оценку.

Для упрощения процесса тестирования жюри может использовать специальную тестирующую программу.

Количество тестов и их цены являются секретными и оглашаются только при начале тестирования.

Тестовый пример, приведенный в тексте задачи, служит только для проверки правильности считывания входных данных из файла и не может быть использован в процессе тестирования, баллы за него не даются.

Если формат выходного файла не соответствует описанному в условии задачи формату, то тест не засчитывается.

Не допускается «частичное» прохождение теста.

Определение призовых мест

На олимпиаде по информатике оба тура равноценны, получение участником нулевой оценки в первом туре не может быть основанием к не допуску его на второй тур. Место участника определяется по сумме баллов набранных им в двух турах. Первое место присуждается участнику, набравшему максимальное количество баллов, без учета процента решения относительно всех задач. При равенстве баллов участники делят соответствующее место.

Темы вопросов для подготовки к олимпиадам по информатике

Техника программирования

1. Основы языка программирования (Паскаль, Си)

Переменные и простейшие типы данных, размеры типов. Линейные программы. Условные операторы. Циклы. Процедуры и функции. Сложные типы данных (массивы, строки, записи, указатели, файлы).

2. Массивы

Одномерные массивы. Двумерные массивы (матрицы). Многомерные массивы.

3. Строки. Элементы лексического и синтаксического разбора

Операции над строками. Лексемы, подсчет лексем различных типов. Выделение чисел из строки.

4. Работа с файлами

Чтение и запись в текстовый файл. Преобразование полученных из файла данных в удобную структуру. Работа с типизированными файлами. Нетипизированные файлы. Буферизация ввода.

5. Рекурсия

Математические функции, задаваемые рекурсивно. Примеры рекурсивных подпрограмм. Проблема остановки рекурсии. Замена рекурсии итерацией.

6. «Длинная» арифметика

Хранения в программе чисел, которые не вмещаются в стандартные типы. Арифметические операции над «длинными» числами. «Длинные» числа с десятичной частью. Извлечение корня с заданной точностью.

7. Хранение информации в динамической памяти.

Хранение набора данных в линейных списках. Вставка в список, удаление из списка, поиск элемента в списке. Двусвязные списки. Понятия структур данных стека, кольца, очереди, дека; реализация их с помощью динамической памяти. Двоичные деревья. Деревья с неопределенным числом потомков. Хранение больших массивов.

Алгоритмы, методы и принципы решения задач

1. Понятие сложности алгоритма.

Определение сложности. Классы задач P и NP. NP-полные задачи.

2. Алгоритмы поиска и сортировки

Поиск элемента в неупорядоченном массиве. Двоичный поиск по ключу в упорядоченном массиве (дихотомия). Поиск методом Фибоначчи. Поиск в упорядоченном n-мерном массиве. Поиск k-го по величине элемента массива. Простые методы сортировки («пузырек», «выборка», «вставка», «подсчет»). Быстрые методы («быстрая», «слиянием», «пирамидальная»), балансировка двоичных деревьев. Сортировка методом черпака.

3. Решение задач методом перебора вариантов

Применение рекурсии для перебора. Генерация сочетаний, размещений, перестановок и булеана множества. Полный перебор. Отсечение вариантов (эвристики). Метод ветвей и границ.

4. Вычислительная геометрия и численные методы

Длина отрезка. Уравнение прямой. Скалярное и векторноепроизведение. Точка пересечения отрезков. Принадлежность точки фигуре на плоскости. Определение выпуклости многоугольника. Площадь многоугольника. Выпуклая оболочка множества точек: алгоритмы Грэхема, Джарвиса, «разделяй и властвуй». Ближайшая пара точек. Метод Гаусса для решения системы линейных уравнений. Нахождение решения уравнения.

5. Принцип динамического программирования

Понятие, применимость. Сравнение с перебором.

6. Жадные алгоритмы

Понятие, применимость. Сравнение с перебором и динамическим программированием.

7. Теория графов. Алгоритмы на графах

Понятие графа. Определения теории графов. Структуры данных для представления графа в программе. Алгоритмы обхода графа (поиски в ширину и глубину). Лабиринт (метод волны). Эйлеров цикл. Кратчайший путь во взвешенном графе (алгоритмы Дейкстры и Минти). Транзитивное замыкание графа (алгоритм Флойда-Уоршилла). Минимальное остовное дерево (алгоритмы Прима и Краскала). Топологическая сортировка графа. Потоки в сетях (алгоритм Форда-Фалкерсона). Паросочетания в двудольном графе (метод удлиняющей цепочки, потоковое решение). Задача о назначениях, назначения на узкое место (венгерский алгоритм). Игры на графах. Раскраска графа. Уложение графа на плоскости. Сильная связность и двусвязность графа. Изоморфизм графов. K-клика. Гамильтонов цикл.

8. Лексический и синтаксический анализ

Задача «Калькулятор». Синтаксические диаграммы. Формы Бэкуса-Наура. Стековая и рекурсивная модель синтаксического разбора. Конечные автоматы. Грамматики.

9. Задачи с «изюминками»

Олимпиады по информатике

1. Правила проведения олимпиад по программированию.
2. Типичные ошибки и отладка программ.
3. Поведение во время соревнования.
4. Методы аутотренинга.

Отрывоки из книги «Мифический человеко-месяц или как создаются программные системы», автор-Фредерик БРУКС (1995).

ЦЕЛОЕ И ЧАСТИ

«Я духов вызывать могу из бездны.
И я могу, и каждый может,
Вопрос лишь, явятся ль на зов они?»
ШЕКСПИР, КОРОЛЬ ГЕНРИХ IV

Среди современных кудесников, как и встарь, встречаются хвастуны: «Я могу писать программы, которые управляют воздушным движением, перехватывают баллистические ракеты, делают переводы по банковским счетам, управляют производственными линиями». На что есть ответ: «И я могу, и каждый может, но будет ли работать то, что ты напишешь?»

Нисходящее проектирование. В очень четкой статье 1971 года Никлаус Вирт формализовал процедуру разработки, годами использовавшуюся лучшими программистами. Более того, его замечания, сделанные в отношении разработки программ, полностью применимы к разработке сложных программных систем. Воплощением этих замечаний является разделение создания систем на проектирование архитектуры, разработку и реализацию. Более того, каждая из задач проектирования архитектуры, разработки и реализации лучше всего может быть решена нисходящими методами.

Вкратце, метод Вирта определяет разработку как последовательность уточняющих шагов. Набрасывается примерное описание задачи и грубый метод решения, позволяющий получить основной результат. Затем определение изучается более пристально, чтобы увидеть, в чем отличие полученного результата от требуемого, и крупные этапы решения разбиваются на более мелкие. Каждое уточнение в определении задачи становится уточнением алгоритма решения и может сопровождаться уточнением представления данных.

В этом процессе выявляются модули решения или данных, дальнейшее уточнение которых может быть продолжено независимо от основной работы. Степень такой модульности определяет гибкость и изменяемость программы.

Вирт считает необходимым использование на каждом шаге нотации как можно более высокого уровня, чтобы выделить понятия и скрыть детали, пока не станет необходимым дальнейшее уточнение.

Правильно осуществляемое нисходящее проектирование позволяет избегать ошибок по нескольким причинам. Во-первых, прозрачность структуры и представления облегчает точную формулировку требований к модулям и их функций. Во-вторых, расчленение и независимость модулей помогают избежать системных ошибок. В-третьих, проект можно тестировать на каждом уточняющем шаге, поэтому тестирование можно начать раньше и на каждом шаге сосредоточиться на подходящем уровне детализации.

Процесс пошагового уточнения не означает, что в случае столкновения с какой-нибудь неожиданно затруднительной деталью не приходится возвращаться назад, отбрасывать самый верхний уровень и начинать все сначала. На практике это часто случается. Но становится значительно легче точно увидеть, когда и почему нужно отбросить весь проект и начать сначала. Многие слабые системы появляются в результате попыток сохранить скверный первоначальный проект путем разного рода косметических заплаток. Нисходящее проектирование уменьшает такой соблазн.

Я убежден, что нисходящее проектирование является важнейшей новой формализацией программирования за десятилетие.

Структурное программирование. Другой важный круг идей для разработки, сокращающих число ошибок в программе, исходит то Дейкстры (Dijkstra) и построен на теоретической структуре Бёма (Boehm) и Джакопини (Jacopini).

В своей основе подход заключается в разработке программ, управляющие структуры которых состоят только из циклов, определяемых такими операторами, как DO WHILE и группами условно выполняемых операторов, ограниченных скобками с использованием операторов условия IF…THEN…ELSE. Бём и Джакопини показывают теоретическую достаточность таких структур. Дейкстра доказывает, что альтернативное неограниченное применение ветвление с помощью GO TO образует структуры, располагающие к появлению логических ошибок.

В основе, несомненно, лежат здравые мысли. При обсуждении сделано много критических замечаний — в частности, большое удобство представляют дополнительные управляющие структуры, такие как n-вариантный переход (так называемый оператор CASE) для различения среди нескольких случаев и аварийный выход (GO TO ABNORMAL END). Кроме того, некоторые догматически избегают всех GO TO , что представляется чрезмерным.

Важной и существенной для создания программ, не содержащих ошибок, является необходимость рассматривать управляющие структуры системы как управляющие структуры, а не как отдельные операторы перехода. Такой образ мысли является большим шагом вперед.

…….

«Пошаговая обработка: наращивать программу, а не строить сразу. Я до сих пор помню испытанный в 1958 году удар, когда впервые услышал, как мой друг говорил о строительстве (building) программ в противоположность написанию (writing). В мгновение он расширил все мое представление о процессе программирования.

Применение метафоры было сильным и точным. Сегодня мы понимаем, что сходство существует между созданием программы и другими строительными процессами, и свободно используем другие элементы метафоры, такие как спецификации (specifications), сборка компонентов (assembly of components), леса (scaffolding).

Метафора строительства пережила свое время. Пора снова вносить изменения. Если, как я считаю, создаваемые сегодня концептуальные структуры слишком сложны, чтобы их можно было точно специфицировать заранее, и слишком сложны, чтобы строить без ошибок, тогда нужен радикально иной подход.

Обратимся к природе и рассмотрим сложность живых созданий, а не безжизненных творений человека. Там мы обнаруживаем конструкции, сложность которых вселяет в нас ужас. Один только мозг настолько сложен, что невозможно составить его схему. Его мощь невозможно повторить, он богат своеобразием, способен к самосохранению и самообновлению. Секрет в том, что мозг растет, а не строится.

Так же должны создаваться наши программные системы. Несколько лет назад Харлан Миллз предложил наращивать программные системы путем пошаговой разработки. *Это значит, что сначала систему надо заставить выполняться, даже если при этом она не делает ничего полезного, кроме вызова некоторого числа фиктивных подпрограмм. Затем она понемногу обрастает мясом, причем подпрограммы, в свою очередь, разрабатываются сначала как вызовы пустых фиктивных подпрограмм, находящихся на уровень ниже*(выделено Д.А).

Настаивая на применении этой технологии разработчиками проектов на моих лабораторных занятиях по программной инженерии, я стал свидетелем поразительных результатов. За последнее десятилетие ничто другое не оказало столь сильного влияния на мою собственную работу и ее эффективность. Этот подход предполагает нисходящее проектирование, поскольку это — нисходящее наращивание программы. Он позволяет легко отслеживать работу в обратном направлении. Он предоставляет возможность раннего создания макетов. Каждая новая функция или возможность работы с более сложными данными или условиями органически вырастают на того, что уже имеется.

Воздействие на моральный дух ошеломительное. Когда есть хотя бы простая работающая система, возрастает энтузиазм. Энергия удваивается, когда на экране появляется картинка из новой графической программной системы, даже если это всего лишь прямоугольник. И на каждой стадии процесса разработки существует работающая система. Я считаю, что за одинаковые сроки команда может нарастить значительно более сложный объект, чем построить.

В больших проектах можно ощутить такие же выгоды, как и в моих маленьких.

Выдающиеся проектировщики. Главная проблема совершенствования искусства программирования заключена, как всегда, в людях.

Мы можем добиваться хороших проектов, следуя хорошим, а не плохим практическим приемам. Хорошим приемам можно обучать. Программисты принадлежат к наиболее интеллектуальной части общества, следовательно, они в состоянии изучать хорошие приемы. Поэтому важнейшим направлением в Соединенных Штатах является распространение хороших современных приемов.

Новые курсы, новые издания, новые организации, такие как Институт инженеров программистов (Software Engineering Institute) — все это вызвано к жизни стремлением повысить уровень наших практических приемов. Это совершенно правильно.

Тем не менее, я считаю, что мы не сможем подняться еще на одну ступеньку выше, действуя в этом направлении. Выбор правильного метода проектирования определяет различия между плохим и хорошим концептуальным проектом, но не между хорошим и выдающимся. Выдающиеся проекты создаются выдающимися проектировщиками. Создание программ является творческим процессом. Крепкая методология может придать силу и освободить творческий ум, но она не может воспламенить или вдохновить того, кто занят нудной работой.

И разница немалая — это как Сальери и Моцарт. Одно исследование за другим показывают, что лучшие проектировщики создают структуры, которые быстрее, меньше по размеру, проще, понятнее и разработаны меньшими усилиями. Различия между выдающимся и средним достигают порядка величины.

Нетрудно проследить, что ряд хороших и полезных программных систем проектировался комиссиями и создавался с помощью проектов, состоявших из многих частей. Но программные системы, вызвавшие восхищение страстных поклонников, являются продуктом одного или небольшого числа выдающихся проектировщиков.

Как растить выдающихся проектировщиков? Место не позволяет обсуждать это пространно, но вот некоторые очевидные шаги:

• Систематически и как можно раньше выявлять первоклассных проектировщиков. Лучшие — не всегда самые опытные.

• Назначить наставника, ответственного за рост перспективного проектировщика и тщательно следить за его карьерой.

• Разработать и осуществлять план служебного роста для каждого перспективного проектировщика, включающий тщательно продуманное обучение у передовых проектировщиков, периоды дополнительного формального обучения, краткосрочные курсы, перемежающиеся с самостоятельным проектированием и назначением на руководящие технические должности.

• Обеспечить возможности для взаимодействия и взаимного стимулирования растущих проектировщиков.

СПИСОК ЛИТЕРАТУРЫ

1. Ахо А.А., Хопкрофт Д.Э., Ульман Д.Д. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

2. Асанов М.О. Дискретная оптимизация. Екатеринбург: УралНаука, 1998.

3. Андреева Е.В. Еще раз о задачах на полный перебор вариантов. “Информатика”, №45, 2000.

4. Бондарев В.М., Рублинецкий В.И., Качко Е.Г. Основы программирования. Харьков. Фолио. 1997.

5. Гарднер М . Математические головоломки и развлечения М.: Мир, 1982.

6. Гордеев Э.Н. Задачи выбора и их решение. В кн.: Компьютер и задачи выбора. М.: Наука, 1989.

7. Гэри М., Джонсон Д. Вычислительные машины и трудноразрешимые задачи. М.: Мир, 1982.

8. Даулеткулов А.Б. Основы программирования на языке Паскаль (алгоритмизация и программирование). Алматы. НИТ. 2004.

9. Кирюхин В.М., Лапунов А.В., Окулов С.М. Задачи по информатике. Международные олимпиады 1989-1996. - М.: “ABF”, 1996.

10. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. М.: МЦНМО, 2000.

11. Котов В.М., Волков И.А., Лапо А.И. Методы алгоритмизации (8 класс). Минск. Народная ясвета.1997.

12. Котов В.М., Волков И.А., Лапо А.И. Методы алгоритмизации (9 класс). Минск. Народная ясвета.1997.

13. Липский В. Комбинаторика для программистов. М. Мир. 1988.

14. Окулов С.М. Геометрические алгоритмы. “Информатика”, №15, 16, 17, 2000.

15. Окулов С.М. 100 задач по информатике. Киров: изд-во ВГПУ, 2000.

16. Препарата Ф., Шеймос М. Вычислительная геометрия: введение. — М.: Мир, 1989.

Учебное издание

Даулеткулов Алдияр Бексеитович

ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ

Учебно - методическое пособие

Лицензия № 14 от 27.06.2003 г., выданная
Министерством образования и науки РК
ТОО «Институт новых технологий» (ИНТ),
г. Алматы, ул. Шевченко 162-ж, офис - 403.
Главный редактор ***Губайдулина З.А.***
Технический редактор ***Мамырбек Г.Б.***
Корректор ***Губайдулина З.А.***
Компьютерная верстка и оформление:
Мрыхина Т.Г.
Подписано к печати 23.07.2004г.
Формат 60x84 1/16. Писчая бумага Kum Lux
Условн.печ.лист 15.2 . Тираж 1300 . Заказ № 24