Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования «Барнаульский государственный педагогический университет»

Задания и методический блок для педагогов по олимпиаде по информатике для учащихся

Автор:
Каракозов С.Д., докт. пед. наук, профессор,
Петропавловский М.Д., канд. ф.- м. наук, доцент

Барнаул - 2006

В данном пособии мы рассмотрим решение ряда задач и соответствующий теоретический материал.

АНАЛИЗ И ПОСТРОЕНИЕ АЛГОРИТМОВ

Построение и анализ алгоритмов рассмотрим на примере трех сортировок данных:

1.Сортировка выбором
2. Сортировка вставками.
2. Быстрая сортировка.

Сортировка выбором

Алгоритм сортировки выбором можно описать следующим образом: Пусть дан массив $a_1..a_n$. Предположим, что часть массива a: $a_1..a_{j-1}$ уже упорядочена по возрастанию элементов. Из оставшейся части массива выберем наименьший элемент и обменяем его местами с элементом a_i. Это элемент массива добавляем к уже упорядоченной части, а номер i увеличиваем на единицу. И так будем продолжать до тех пор пока не достигнем предпоследнего элемента. По данному словесному алгоритму напишем программу, например, на языке Паскаль.

```
Const N=20;
Var
    A:array[1..N] of integer;
    i,j,k,num,v,minA:integer;
begin
   assign(input, 'ArrInput.txt'); Reset(input);{Связываем входной и выходной }
   assign(output,'ArrOutpu.txt');rewrite(output);{потоки данных с}
   {соответствующими файлами}
   fillchar(A,sizeof(A),0);    num:=0;
   while not eof(input) do
     begin inc(num);read(A[num]);
   end;
    for i:=1 to num-1 do
      begin k:=i; minA:=A[i];
        for j:=i+1 to num do
          if MinA>A[j] then
            begin
              k:=j;MinA:=A[k];
            end;
         v:=A[i];A[i]:=A[k];A[k]:=v;
       end;
 for i:=1 to num do write(A[i]:6);
```

```
  close(input);close(output);
end.
```

Проанализируем данный алгоритм:

1). Массив первоначально упорядочен в порядке убывания элементов.

В этом случае мы производим операцию v:=A[i];:=A[i]:=A[k];A[k]:=v, но i. Если это считать за одну перестановку, то таких перестановок будет произведено n-1. Число сравнений будет $\sum_{j=i}^{n-1} j = (n-1)*(n-1+n-2+..1) = \frac{n^2}{2} - \frac{n}{2}$, то есть можем считать, что число сравнений не более n^2.

2). Массив первоначально упорядочен в порядке возрастания элементов.

Независимо от этого получим те же самые значения для числа обменов и сравнений, так как алгоритм будет одинаково выполняться независимо от расположения элементов в массиве.

Отсюда можно сделать вывод, что время выполнения будет пропорционально $n^2/2-n/2$ или взяв верхнюю границу можно сказать, что время выполнения пропорционально n^2, в таких случаях записывают $T[n]=O(n^2)$. Такие алгоритмы можно использовать для небольших значений n.

Сортировка вставками

Если рассмотреть алгоритм сортировки вставками, то алгоритм написанный на языке Паскаль будет выглядеть следующим образом:

```
 Const N=20;
Var
    A:array[1..N] of integer;
    i,j,k,num,v,key:integer;
begin
   assign(input, 'ArrInput.txt'); Reset(input);
   assign(output,'ArrOutpu.txt');rewrite(output);
   fillchar(A,sizeof(A),0);    num:=0;
   while not eof(input) do
     begin inc(num);read(A[num]);
   end;
    for i:=2 to num do
       begin
         key:=A[j]; i:=j-1;
         while (i>0 and A[j]>key do
           begin
              A[i+1]:=A[i];dec(i);
           end;
          A[i+1]:=key;
        end;
  for i:=1 to num do write(A[i]:6);
  close(input);close(output);
end.
```

Покажем, как в процессе работы этой программы изменяется массив данных
A=5 2 4 6 1 3

2 5 4 6 1 3
2 4 5 6 1 3
1 2 4 5 6 3
1 2 3 4 5 6

Массив просматривается слева направо, если обнаруживается неупорядоченность, то производится вставка в нужное место массива, то есть 2 вставляется перед 5, затем 4 – перед 5 затем 6 –перед 6, затем 1 – перед 2 и, окончательно 3- перед 4.
В худшем случае, когда массив упорядочен по убыванию время сортировки пропорционально n^2 , в лучшем случае (массив упорядочен по возрастанию) время работы пропорционально n. В качестве оценки эффективности данного алгоритма берем $O(n^2)$.

Быстрая сортировка

Существуют и более быстрые сортировки, например: сортировка слиянием и быстрая сортировка эффективность этих сортировок O(n*log(n)). Остановимся на быстрой сортировке.

Быстрая сортировка выполняется следующим образом. Массив A делится на 2 части. Элементы левой части обмениваются с элементами правой части, таким образом, чтобы любой элемент из левой части был не больше любого элемента из правой. После выполнения этого условия данные действия применяются к каждой из двух частей. Данный алгоритм рекурсивный. Быстрая сортировка была разработана Ч. Хоаром в 1962 г. Приведем один из вариантов программы быстрой сортировки.

```
Const N=20;
Type Arr=array[1..N] of integer;
Var
    A:Arr;
    i,j,k,num:integer;
Procedure quick(m,n:integer);
var i,j,midl:integer;
  procedure swap;
  var v:integer;
  begin
    v:=A[i];A[i]:=A[j];A[j]:=v;
  end;
begin
   i:=m;j:=n;midl:=A[(n+m) div 2];
   repeat
     while A[i]<midl do inc(i);
     while A[j]>midl do dec(j);
```

```
    if i<=j then begin swap;inc(i);dec(j);end;
  until i>j;
  if m<j then Quick(m,j);
  if i<n then Quick(i,n);
end;
begin
   assign(input, 'ArrInput.txt'); Reset(input);
   assign(output,'ArrOutpu.txt');rewrite(output);
   fillchar(A,sizeof(A),0);    num:=0;
   while not eof(input) do
      begin inc(num);read(A[num]);
   end;
 quick(1,num);
 for i:=1 to num do write(A[i]:6);
 close(input);close(output);
end.
```

Тестирование разработанных алгоритмов

Помимо оценки эффективности алгоритма необходима и проверка правильности его работы. Алгоритм можно считать правильным, если при любом допустимом (для данной задачи) наборе входных данных он заканчивает работу и выдает набор выходных данных удовлетворяющих требованиям задачи. В этом случае говорят, что алгоритм решает данную вычислительную задачу. То есть для оценки правильности работы алгоритма необходимо составить набор тестов, которые бы позволяли оценить правильность работы алгоритма. К таким тестам можно отнести:

1. Тесты проверяющие, работает ли программа вообще.
2. Выдает ответ программа в вырожденных случаях, то есть при данном наборе данных решение не существует и программа должна сообщить об этом. Или сообщить, что вы вводите недопустимый набор данных и т.д.
3. Тесты позволяющие проверить работу программы в граничных случаях: переполнение памяти, выход за границы массива, выход за пределы какого-либо типа данных и т.д.
4. Тесты проверяющие работу программы в целом:
   тесты проверяющие работу всех ветвей логической схемы программы.
5. Если программа выполняет задачу не точно, а приближенно, то необходимы тесты для проверки правильности приближенных вычислений.
6. Случайные тест: наборы входных данных сгенерированных случайным образом.

ПЕРЕБОР И МЕТОДЫ ЕГО СОКРАЩЕНИЯ.

Существует достаточно много задач, которые требуют полного перебора всех комбинаций данных. Такие задачи называют NP полными, так как нельзя их решить за полиномиальное время ($T[n]=O(n^k)$, где k – целое число). Такие задачи часто бывают практически неразрешимы (требуют значительных машинных ресурсов - память, время исполнения) и могут быть решены, если возможен приближенный вариант решения или имеются какие-либо ограничения. Существуют различные методы сокращения перебора, конечно, если сокращение перебора возможно. Рассмотрим некоторые методы и приведем решения задач.

Перебор вариантов

Рассмотрим две задачи, которые можно отнести к NP полным задачам, но даже при простом переборе просмотр всех вариантов не обязателен.

1. Подсчитать количество различных сочетаний заданной длины (не более 40 символов), составленных из букв заданного слова. Буквы могут повторяться не более того количества раз, сколько раз они повторяются в заданном слове.

Входные данные: Первая строка содержит исходное слово, вторая строка число К.

Выходные данные: Число.

Ограничение по времени тестирования: по 1 секунде на один тест.

Пример:

Входные данные	Выходные данные
Миссисипи 3	12

Это задача, где перебираются все возможные сочетания слов длины k символов, получаемых из слова длины n символов (n>k). Так как в исходном слове есть повторяющиеся буквы, то возникают и повторяющиеся сочетания, которые отбрасываются. Данный алгоритм является рекурсивным. На все сочетания заводится массив повторений, каждый элемент, которого должен быть равен 1.

Решение:

```
Var i,j,n,n1,m,k,p,min:integer;
```

```
   s,s1:string;kol:longint;
   a,last,pov,pov1:array[1..255] of integer;
   f1,f2:text;
   fin,fout:string;
Procedure CNK(i,L,R:integer);
 Var j,p:integer;
 begin
  for j:=L to R do
   begin
      a[i]:=j;
      if i<k then
            begin
              if pov[j]>1 then p:=0 else p:=1;
              if p=0 then dec(pov[j]); CNK(i+1,j+p,last[i+1]);
              if p=0 then inc(pov[j]);
            end   else inc(kol);
   end;
 end;
BEGIN fin:='';fout:='';
 assign(f1,fin);assign(f2,fout);
 reset(f1);rewrite(f2);
 readln(f1,s);readln(f1,k);close(f1);
 n:=length(s);
  s1:='';
  for i:=1 to n do if pos(s[i],s1)=0 then s1:=s1+s[i];
  n1:=length(s1);
  for i:=1 to n1 do
   begin
     kol:=0;
     for j:=1 to n do if s1[i]=s[j] then kol:=kol+1;
     pov[i]:=kol;
   end;
  pov1:=pov;m:=n1;
  for i:=k downto 1 do
   begin
     last[i]:=m;
     dec(pov1[m]);if pov1[m]=0 then dec(m);
   end;
  kol:=0;CNK(1,1,last[1]);
  WRITELN(f2,kol);close(f2);
END.
```

2. Один из режимов ввода SMS-сообщений в мобильном телефоне требует нажатия по одному разу кнопок, содержащих буквы слова, которое надо ввести.

Например, чтобы ввести слово PHONE, нужно нажать один раз кнопку 7 (на ней находится буква P), затем один раз кнопку 4 (на ней находится буква H), затем два раза кнопку 6 (на ней находятся буквы O и N), и, наконец, один раз кнопку 3 (на ней находится буква E).
Поскольку одна кнопка мобильного телефона соответствует более чем одной букве, то и полная последовательность нажатий может соответствовать более чем одному слову. Вам требуется реализовать часть программного обеспечения мобильного телефона — по заданной последовательности нажатий клавиш выделить из словаря телефона слова, которые могли подразумеваться пользователем при нажатии такой последовательности клавиш.

Раскладка клавиатуры мобильного телефона:
2 – ABC, 3 – DEF, 4 – GHI, 5 – JKL, 6 – MNO, 7 – PQRS, 8 – TUV, 9 – WXYZ
Входные данные: первая строка содержит последовательность цифр, соответствующая последовательности нажатий кнопок (не более 20). Во второй строке содержится число слов в словаре (не более 10 000). Начиная с третьей строки идёт словарь по одному слову в строке, в лексикографически возрастающем порядке. Длина каждого слова не превосходит 20. Слова состоят из заглавных латинских букв.
Выходные данные: содержится набор слов словаря, по одному в строке, подходящих под заданную последовательность нажатий кнопок по одному слову в строке, в лексикографически возрастающем порядке. В случае, если такого слова нет – вывести NO
Ограничение по времени тестирования: по 1 секунде на один тест.
Пример

Входные данные	Выходные данные
74663 3 CELL MOBILE PHONE	PHONE

В данной задаче осуществляется сравнение получаемого слова, при нажатии клавиш мобильного телефона, со словами представленными в словаре.

Решение:

```
const
  key: array['A'..'Z']of char=(
    '2','2','2',
    '3','3','3',
    '4','4','4',
    '5','5','5',
    '6','6','6',
    '7','7','7','7',
```

```
    '8','8','8',
    '9','9','9','9'
  );
var
  s,ss,digits:string;
  i,j,n:longint;
begin
  readln(digits);
  readln(n);
  for i:=1 to n do begin
    readln(s);
    if length(s)=length(digits) then begin
      ss:=s;
      for j:=1 to length(s) do
        s[j]:=key[s[j]];
      if s=digits then
        writeln(ss);
    end;
  end;
end.
```

Перебор с возвратом

Данный метод рассмотрим на примере широко известной задачи, которая формулируется следующим образом:

Дана шахматная доска размером n*n, необходимо обойти шахматным конем доску так, чтобы конь побывал на всех клетках по одному разу.

В данной задаче, при обходе конем шахматной доски, может возникнуть ситуация когда коню станет некуда ходить: все ходы, которые разрешены приводят к попаданию коня либо за пределы доски, либо на клетки на которых уже конь бывал ранее. При возникновении такой ситуации необходимо отступить на ход назад и проверить есть ли еще неиспользованные ходы с данной клетки. Если неиспользованные ходы есть, то пробуем выполнить первый неиспользованный ход, если нет, то отступим на еще ход назад и так далее. С каждой клетки конь может выполнить восемь ходов. При каждом ходе одна из координат изменяется на ±1, другая на ±2. В ниже приведенной программе эти приращения координат заданы массивами A=[2,1,-1,-2,-2,-1, 1, 2] (изменение координаты X) и B=[1,2, 2, 1,-1,-2,-2,-1] (изменение координаты Y). При выполнении хода X:=X+A[k], Y:=Y+B[k], где k – номер варианта хода коня с клетки (X,Y). Варианты ходов начинаются с k=1. При попадании за пределы доски или на клетку, где конь уже побывал, номер варианта увеличивается на единицу. Если при проверке окажется, что уже использовался последний вариант, то необходимо отступить на один ход. Задача имеет решение, если N>4.

Время исполнения зависит от размеров доски и клетки, с которой начинается обход.

```
Const
   n=8;{размер шахматной доски n*n}
   n1=8;{максимальное число вариантов хода конем с данной клетки,
    номер хода нумеруется от 1 до 8 по часовой стрелки.}
   Nsqr=n*n;
   a:array[1..n1] of {При ходе хонем происходит приращение номера строки}
   integer=(2,1,-1,-2,-2,-1, 1, 2);
   b:array[1..n1] of {и приращение номера столбца}
   integer=(1,2, 2, 1,-1,-2,-2,-1);
Type
  delta=array[1..n] of integer;
  Desk =array[1..n] of delta;
Var
  d:desk;
  q:boolean;
Procedure Try(i,x,y:integer;var q:boolean);
Var
  u,v,k:integer;
  q1   :boolean;
BEGIN
  k:=0;
  Repeat
    inc(k);q1:=false;u:=x+a[k];v:=y+b[k];
    if
    (1<=u) and (u<=n) and
    (1<=v) and (v<=n) and (d[u][v]=0)
       then
         begin
           d[u][v]:=i;
           if (i<Nsqr)
             then
               begin
                 Try(i+1,u,v,q1);
                 if not q1 then d[u][v]:=0;
               end
             else q1:=true;
         end;
  Until q1 or (k=n1);
  q:=q1;
END;(* Try *)
Procedure KnightsTour;
Var
```

```
  i,j:integer;
BEGIN
  fillChar(d,sizeof(d),0);
  i:=1;j:=1;
  d[i][j]:=1;
  Try(2,i,j,q);
  if q then Writeln('ok')  else  Writeln('end');
  Writeln;
  for i:=1 to n do
    begin
      for j:=1 to n do  Write(d[i][j]:4);
      Writeln;Writeln;
    end;
END;
BEGIN
  KnightsTour;
  Readln;
END.
```

Перебор отсечением и склеиванием ветвей.

Идея метода отсечения ветвей заключается в следующем: из возможных вариантов перебора заранее исключаются те, которые очевидно не дадут решения. Так, например, требуется расставить N ферзей на шахматной доске (размер N*N), так чтобы ни один ферзь не находился под боем. Очевидно, что на каждой вертикали, горизонтали и диагонали должен находится один ферзь. Учет этого значительно сокращает перебор.

Идея метода склеивания ветвей заключатся в том, что если решение имеет симметричные или одинаковые части, то находится только часть решений. Рассмотрим ту же самую задачу про расстановку ферзей, а также будем считать, что число ферзей четное. Если ферзь номер i находится на горизонтали номер j, то этот факт записывается следующим образом:x[i]=j. Номер ферзя это номер вертикали. Легко видеть, что любая расстановка ферзей имеет симметричную относительно середины вектора x. Поэтому можно выполнить расстановки для x[1]<=N div 2, половину можно получить, если x[i]:=N+1-x[i]

1.

Пример программы:

```
Const
  N=8;
  Num=4;
Type
  YesHor =array[1..N] of boolean; {занятость горизонтали, где
  индекс массива номер горизонтали}
  YesD1  =array[-(N-1)..(N-1)] of boolean;{номер диагонали определяется
```

```
  разностью индексов квадратной матрицы i-j,диагонали параллельные
  главной диагонали матрицы}
  YesD2  =array[2..2*N] of boolean;{номер диагонали определяется суммой
  индексов i+j, диагонали параллельные вспомогательной}
  ferz   =array[1..N] of integer;{индекс -номер ферзя ,
                          то есть номер вертикали}
Var
  x     :ferz;
  a     :yeshor;
  b     :yesD1;
  c     :YesD2;
Procedure Print;
var
  k:integer;
BEGIN
  for k:=1 to N do Write(x[k],'..');
  write('          ');
  for k:=1 to N do Write(N+1-x[k],'..');{+}
  Writeln;{+}
END;
Procedure Try(i:integer);
Var
  j:integer;
BEGIN
  for j:=1 to N do
   if a[j] and b[i-j] and c[i+j] and (x[1]<=Num) then {+}
   {если клетка не бьется, то}
     begin
       x[i]:=j;{то ферзю i,устанавливается номер горизонтали j}
       a[j]:=false;b[i-j]:=false;c[i+j]:=false;{соответсвующие горизонталь
       , и две вертикали становятся занятыми}
       if i<n then try(i+1) else print;{если это не последний ферьз,
       то пытаемся поставить следущий ферьз, иначе распечатка варианта}
       a[j]:=true;b[i-j]:=true;c[i+j]:=true;
       {если нет не битого поля для данного ферзя, то предыдущая
        занятая позиция освобождается(отход назад) и все повторяется}
     end;
END;
Procedure Init;
BEGIN
  FillChar(a,sizeof(a),1);FillChar(b,sizeof(b),1);FillChar(c,sizeof(c),1);
  {Все элементы логических массивов a,b,c становятся TRUE , то есть
   клетки доски не находятся под боем }
  FillChar(x,sizeof(x),0);{все ферзи вне доски , то есть все x[i]=0}
END;
```

```
BEGIN
  assign(output,'foutput.txt');rewrite(output);
  Init;{устанавливаем все первоначальные данные}
  try(1);{выполняем расстановку}
  close(output);
END.
```

Программа расстановки ферзей рекурсивная,
Если в программе условие:
if a[j] and b[i-j] and c[i+j] and (x[1]<=Num) then
заменить на:
if a[j] and b[i-j] and c[i+j] then, а в процедуре print оставить:
for k:=1 to N do Write(x[k],'..');writeln;
то мы получим программу без склеивания ветвей.

2. На шахматной доске N*N (1<=N<=10) расположить N пешек таким образом, чтобы в каждой строке, каждом столбце, на главной и побочной диагоналях находилось ровно по одной пешке. Составьте алгоритм подсчета количество всех комбинаций.
Входные данные: число N.
Выходные данные: количество всех комбинаций.
Ограничение по времени тестирования: по 1 секунде на один тест.
Пример

Вход	Выход
4	8

Данная задача очень похожа на задачу о расстановке N ферзей на шахматной доске размером N*N. Алгоритм решения рекурсивный. За каждой пешкой закреплена, например, своя вертикаль по которой пешка может перемещаться. Количество пешек на главной диагонали задается параметром d1, количество пешек на вспомогательной диагонали – d2 . Пешка находится на главной диагонали, если a[k]=k и пешка находится на вспомогательной диагонали, если a[k]=N-1+k ;

Решение:

```
{В квадратах n*n.      .n<10 }
Type mas=array[1..10] of integer;
Var
  a:mas;
  i,n,d1,d2:integer;
  kol:longint;
  f1,f2:text;
  fin,fout:string;
```

```
Procedure PERES(k:integer);
  Var j,x:integer;
  begin
    for j:=k to n do
      begin
        if a[k]=k then inc(d1);
        if a[k]=n+1-k then inc(d2);
          if (d1<2)and(d2<2)and(k<n) then PERES(k+1)
        else if d1*d2=1 then inc(kol);
        if a[k]=k then dec(d1);
        if a[k]=n+1-k then dec(d2);
          x:=a[k];
          for i:=k to n-1 do a[i]:=a[i+1];
        a[n]:=x;
      end;
  end;
   BEGIN
   fin:='';fout:='';
   assign(f1,fin);assign(f2,fout);
   reset(f1);rewrite(f2);
   readln(f1,n);close(f1);
     kol:=0;
     for i:=1 to n do a[i]:=i;
     PERES(1);
     writeln(f2,kol);close(f2);
   END.
```

Динамическое программирование

Динамическое программирование решает задачу, разбивая ее на подзадачи. В дальнейшем решение подзадач объединяется. Динамическое программирование применяется тогда, когда подзадачи не являются независимыми. При решении задач методом динамического программирования, каждая подзадача решается только один раз и ответ запоминается в специальной таблице. В дальнейшем это входит в решение основной задачи.

Динамическое программирование часто используется для решения задач оптимизации. У задач оптимизации много различных решений, при оптимизации необходимо выбрать лучшее, то есть какой – то параметр при оптимальном решении должен быть максимальным или минимальным в зависимости от условия задачи.

В общем случае алгоритм динамического программирования можно построить следующим образом:

1. Описывается строение оптимальных решений.
2. Выписывается рекуррентное соотношение, связывающее оптимальные значения параметра для подзадач.

3. Двигаясь снизу вверх, вычисляется оптимальное значение для подзадач.
4. На основе полученной информации строится оптимальное решение.

Для иллюстрации вышесказанного приведем примеры решения ряда задач:

1. Необходимо расставить скобки в выражении для произведения N матриц A1*A2*…*An, так чтобы число перемножений было минимальным.

Естественно, что число столбцов первой матрицы равно числу строк второй, число столбцов второй матрицы равно числу строк третьей и т. д.
Разместим данные о строках и столбцах перемножаемых матриц в массиве $p=[p_1,p_2,p_3,..,p_n, p_{n+1}]$, где p_1 – число строк 1- ой матрицы, p_2 – число столбцов 1-ой матрицы и этот же элемент есть число строк 2- ой, и т. д. Последний элемент p_{n+1} – число столбцов n – ой матрицы. Пусть p=[30,35,15,5,10,20,25].
Подсчитаем число требуемых перемножений A1*A2*A3 в двух вариантах:

1. A1*(A2*A3).
2. (A1*A2)*A3

Введем обозначение NA которое будет использоваться для числа перемножений, например NA(A1*(A2*A3)) – есть число перемножений необходимое при вычислении A1*(A2*A3). Очевидно, что NA(A1*(A2*A3))= NA(A2*A3)+NA(A1*Ar), где Ar=A2*A3.
Тогда NA(A1*(A2*A3))=35*15*5+30*35*5=7875.
NA((A1*A2)*A3)= 30*35*15+30*15*5=18000
Как видим от порядка выполнения операций, определяемого скобками в выражении, зависит в, то число перемножений, которое мы должны произвести, чтобы перемножить несколько матриц, хотя произведение нескольких матриц от расстановки скобок не зависит. Другими словами мы можем найти произведения нескольких матриц за разное время, которое зависит от порядка перемножения, то есть от расстановки скобок. Минимальным это время будет тогда, когда число перемножений будет минимально. Окончательно можно записать $NA(A_1*A_2*..*A_k,A_{k+1})= \min(NA(A_1*A_2*..*A_k))+\min(NA(A_k,A_{k+1}))$. Каждое минимальное значение числа перемножений вычисляется только один раз и заносится в массив, matr, где элемент matr[i,j] – минимальное число перемножений, требующееся для нахождения произведений матриц $A_i*A_{i+1}*..*A_j$
Число k (место, где выражение для произведения разделяется), записывается в матрицу cost, cost[i,j]:=k.

Для данного p=[30,35,15,5,10,20,25] приведем полученные matr и cost.

$$Matr=\begin{matrix} 0 & 15750 & 7875 & 9375 & 11875 & 15125 \\ 0 & 0 & 2625 & 4375 & 7125 & 10500 \\ 0 & 0 & 0 & 750 & 2500 & 5375 \\ 0 & 0 & 0 & 0 & 1000 & 3500 \\ 0 & 0 & 0 & 0 & 0 & 5000 \end{matrix}$$

0 0 0 0 0 0

0 1 1 3 3 3
0 0 2 3 3 3
0 0 0 3 3 3
Cost= 0 0 0 0 4 5
0 0 0 0 0 5
0 0 0 0 0 0

Из таблиц для этих матриц видно, что элементы лежащие на главной диагонали и ниже нее равны нулю. Окончательный результат есть Matr[1,6]=15125, все остальные элементы промежуточные результаты, которые были использованы для получения основного результат.

По матрице cost расставляем скобки: cost[1,6]=3, последнее означает означает, что A1*A2*A3*A4*A5*A6 вычисляется как (A1*A2*A3)*(A4*A5*A6). Далее смотрим cost[1,3]=1, то есть (A1*A2*A3)=(A1*(A2*A3)). Смотрим cost[4,6]=5, то есть (A4*A5*A6)= (A4*(A5*A6)). И окончательно имеем: (A1*(A2*A3))* (A4*(A5*A6)).

Пример программы:

```
const NP=20;
type
  RowCol=array[1..NP] of byte;
 var
 p:RowCol; {массив, где записаны число строк и столбцов всех матриц участвующих в перемножении. p[1],p[2]- число строк и число столбцов 1-ой матрицы, а p[2],p[3] - 2-ой матрицы , p[3],p[4] – 3-й}
 matr:array[1..NP,1..NP] of longint; {матрица, где записаны минимальные числа перемножений}
 cost:array[1..NP,1..NP] of byte; {матрица, где записаны положения скобок}
 num,ni,nj:byte; {num – число перемножаемых матриц}
 q:longint;
 D,Sb,g,b:string; {D – строка, где записан окончательный результат, то есть выражение с расставленными скобками}
 br,ce:string;
Procedure Init;
var i,t:byte;
BEGIN
  fillchar(p,sizeof(p),0);fillchar(matr,sizeof(matr),0);
  fillchar(cost,sizeof(cost),0);{обнуляем массивы}
  assign(input,'Mp.txt');reset(input);
  num:=0;t:=0;
  while not eoln(input) do
    begin
      inc(num);
```

```
      read(p[num]);
    end;
  close(input);dec(num);{число элементов массива p на 1 меньше числа
перемножаемых матриц}
  g:='A'; D:=' ';
  for i:=1 to num do
    begin
      str(i,b);D:=D+g+b;
    end;D:=D+' ';
    br:='(';ce:=')';
END;
Procedure MCOG;
var i,j,L,k:byte;
BEGIN
  for L:=2 to num do
    begin
      for i:=1 to num-L+1 do
        begin
          j:=i+L-1; matr[i,j]:=maxlongint;
          for k:=i to j-1 do
            begin
              q:=matr[i,k]+matr[k+1,j]+p[i]*p[k+1]*p[j+1];
              if q<matr[i,j] then begin matr[i,j]:=q;cost[i,j]:=k;end;
              {Записано минимальное значение числа перемножений, причем с
испоьзованием ранее полученных результатов}
            end;
        end;
    end;
END;
Procedure bracets(i,j:byte);{Процедура расставления скобок}
var k:byte;
BEGIN
   if i<j then
    begin
      k:=cost[i,j];
      bracets(i,k);
      str(i,b);Sb:=g+b;ni:=pos(Sb,D);insert(br,D,ni);
      bracets(k+1,j);
      if j<num then
        begin
          str(j+1,b);Sb:=g+b;nj:=pos(Sb,D);insert(ce,D,nj);
        end
       else
        begin
           insert(ce,D,length(D));
```

```
        end;
    end;
END;

Procedure wr;{Процедура записи результатов в файлы}
var i,j:byte;
BEGIN
  assign(output,'mm.txt');rewrite(output);
  for i:=1 to num do
    begin
      for j:=1 to num do
        begin
          write(matr[i,j]:6);
        end;writeln;
   end;close(output);
  assign(output,'fs.txt');rewrite(output);
  for i:=1 to num do
    begin
      for j:=1 to num do
        begin
          write(cost[i,j]:6);
        end;writeln;
    end;close(output);
  assign(output,'fd.txt');rewrite(output);
  writeln(D);
  close(output);
END;
BEGIN
  init;
  mcog;
  bracets(1,num);delete(D,1,2);delete(D,length(D)-1,2);
  wr;
END.
```

Приведем формулировку следующей задачи:

2. В арифметическом выражении, операндами которого являются целые числа, а операциями – бинарные операции “+”, “*” , расставить скобки так, чтобы результат оказался максимальным.

Пусть во входном файле записано: 12*22+11*14*12+-11+124*12

Результатом выполнения программы будет:

12*((22+11)*(14*((12+([-11]+124))*12)))

Пример программы

```
Const NN=35;
Type pt=array[1..4] of byte;
```

```
var r_max,r_min,n,m,t,j,k,i:longint;
  a:array[1..NN,1..NN] of longint;
  op:array[1..NN] of char;
  b:array[1..NN,1..NN] of pt;
  b_max,b_min:pt;

Procedure Init; {В данной процедуре инициализируются массивы и
считываются входные данные в массив a – операнды, в массив op- знаки
операций}
var ch:char;
    flgmin:boolean; {flgmin=true, если операнд отрицательнвый}
BEGIN
  Assign(input,'input.txt');reset(input);t:=0;n:=0;flgmin:=false;
   fillchar(a,sizeof(a),0);
   fillchar(b,sizeof(b),0);
   fillchar(op,sizeof(op), ' ');
  while not eoln(input) do
    begin
      read(ch);
      case ch of
      '0'..'9':t:=t*10+(ord(ch)-48);
'*','+':begin
        inc(n);op[n]:=ch;
        if flgmin then a[n,n]:=-t else a[n,n]:=t;flgmin:=false;t:=0;
        end;
    '-':flgmin:=true;
      end;
    end;  inc(n);
    if flgmin then t:=-t;a[n,n]:=t;
    close(input);
END;
procedure wra; {Данная процедура использовалась при отладке}
var i,j:byte;
BEGIN
  assign(output,'outputA.txt');rewrite(output);
  for i:=1 to n do
    begin
      for j:=1 to n do write(a[i,j]:5);
      writeln;
    end;
  close(Output);
END;
Procedure Run;
  Procedure fillmax(a,b,c,d:longint);
```

```
  Begin {b_max –используются при расстановке скобок: Первые два элемента
связаны с открывающими скобками, вторые два – с закрывающими }
    b_max[1]:=a;b_max[2]:=b;b_max[3]:=c;b_max[3]:=d;
  end;
  Procedure fillmin(a,b,c,d:longint);
  Begin {то же самое, что и для fillmax}
    b_min[1]:=a;b_min[2]:=b;b_min[3]:=c;b_min[3]:=d;
  end;
BEGIN{Run}
  for m:=1 to n-1 do
   for i:=1 to n-m do
    begin{m,i}
     j:=i+m;r_max:=-maxlongint;r_min:=maxlongint;
    { maxlongint = 2147483647 заменяет знак бесконечности }
   for k:=i to j-1 do
      case  op[k] of
      '+':
        begin{'+'}
          if r_max<a[i,k]+a[k+1,j] then begin fillmax(i,k,k+1,j);
          r_max:=a[i,k]+a[k+1,j];end;
          if r_min>a[k,i]+a[j,k+1] then begin fillmin(k,i,j,k+1);
          r_min:=a[k,i]+a[j,k+1];end;
        end;{'+'}
      '*':
      begin{'*'}
        if r_max<a[i,k]*a[k+1,j] then begin fillmax(i,k,k+1,j);
        r_max:=a[i,k]*a[k+1,j];end;
        if r_max<a[k,i]*a[j,k+1] then begin fillmax(k,i,j,k+1);
        r_max:=a[k,i]*a[j,k+1];end;
        if r_max<a[i,k]*a[j,k+1] then begin fillmax(i,k,j,k+1);
        r_max:=a[i,k]*a[j,k+1];end;
        if r_max<a[k,i]*a[k+1,j] then begin fillmax(k,i,k+1,j);
        r_max:=a[k,i]*a[k+1,j];end;
        if r_min>a[i,k]*a[k+1,j] then begin fillmin(i,k,k+1,j);
        r_min:=a[i,k]*a[k+1,j];end;
        if r_min>a[k,i]*a[j,k+1] then begin fillmin(k,i,j,k+1);
        r_min:=a[k,i]*a[j,k+1];end;
        if r_min>a[i,k]*a[j,k+1] then begin fillmin(i,k,j,k+1);
        r_min:=a[i,k]*a[j,k+1];end;
        if r_min<a[k,i]*a[k+1,j] then begin fillmin(k,i,k+1,j);
        r_min:=a[k,i]*a[k+1,j];end;
      end;{'*'}
      end;{case}
     a[i,j]:=r_max;a[j,i]:=r_min;b[i,j]:=b_max;b[j,i]:=b_min;
    end{m,i};
```

```
END; {Run}
Procedure rec(x,y:byte);{Процедура расставляет скобки в выражении, по
массивам a и b }
BEGIN
  if b[x,y,1]=b[x,y,2] then
    begin
      if a[b[x,y,1],b[x,y,2]]<0 then write('[');
      write(f2,a[b[x,y,1],b[x,y,2]]);
      if a[b[x,y,1],b[x,y,2]]<0 then write(']');
    end else
    begin write('(');rec(b[x,y,1],b[x,y,2]);write(')'); end;
  if b[x,y,1]<b[x,y,2] then write(op[b[x,y,2]]) else write(op[b[x,y,1]]);
  if b[x,y,3]=b[x,y,4] then
    begin
      if a[b[x,y,3],b[x,y,4]]<0 then write('[');
      write(a[b[x,y,3],b[x,y,4]]);
      if a[b[x,y,3],b[x,y,4]]<0 then write(']');
    end else
    begin write('(');rec(b[x,y,3],b[x,y,4]);write(')'); end;
END;{Rec}
BEGIN
  ClrScr;
  init;
  wra;
  Run;
  assign(output,‘output.txt');rewrite(output);
  writeln(a[1,n]);rec(1,n);close(output);
END.
```

3. За билетами на премьеру нового мюзикла выстроилась очередь из N человек, каждый из которых хочет купить 1 билет. На всю очередь работала только одна касса, поэтому продажа билетов шла очень медленно, приводя «постояльцев» очереди в отчаяние. Самые сообразительные быстро заметили, что, как правило, несколько билетов в одни руки кассир продаёт быстрее, чем когда эти же билеты продаются по одному. Поэтому они предложили нескольким подряд стоящим людям отдавать деньги первому из них, чтобы он купил билеты на всех.

Однако для борьбы со спекулянтами кассир продавала не более 3-х билетов в одни руки, поэтому договориться таким образом между собой могли лишь 2 или 3 подряд стоящих человека.

Известно, что на продажу i-му человеку из очереди одного билета кассир тратит A_i секунд, на продажу двух билетов — B_i секунд, трех билетов — C_i секунд. Напишите программу, которая подсчитает минимальное время, за которое могли быть обслужены все покупатели.

Обратите внимание, что билеты на группу объединившихся людей всегда покупает первый из них. Также никто в целях ускорения не покупает лишних билетов (то есть билетов, которые никому не нужны).

Входные данные: Первая строка содержит число N, следующие N строк содержат натуральные числа A_i, B_i, C_i не превосходящие 3600.

Выходные данные: Число.

Ограничение по времени тестирования: по 1 секунде на один тест.

Пример:

Входные данные	Выходные данные
5 5 10 15 2 10 15 5 5 5 20 20 1 20 1 1	12

В данной задаче на каждом шаге выбирается минимальный элемент из трех элементов, по схеме d[i]:=min(d[i-1]+a[i], d[i-2] + b[i-1], d[i-3] + c[i-2]), где a[i], b[i], c[i] –время необходимое для продажи одного, двух и трех билетов в одни руки,

Решение:

```
const
  MaxN = 5000;
var
  n : word; {число человек в очереди}
  a, b, c : array [0 .. MaxN] of integer;
  d : array [0 .. MaxN] of longint;
  i : integer;

function min(a,b : longint) : longint;
begin
   if a<b then min:=a else min:=b;
end;

begin

   readln(n);
   for i := 1 to n do
      read(a[i], b[i], c[i]);

   d[0] := 0;
   d[1] := a[1];
   d[2] := min(a[1]+a[2], b[1]);
```

```
    for i := 3 to n do
       d[i] := min( d[i-1] + a[i], min( d[i-2] + b[i-1], d[i-3] + c[i-2] ) );

    writeln(d[n]);
end.
```

4. В некоторой географической области все поселки располагались на одной прямой дороге. Возникла необходимость построить в поселках почтовые отделения, причем не обязательно во всех поселках. Дорога представлена в виде числовой оси, позиция каждого поселка на дороге зафиксирована целым числом - координатой на оси. Все координаты различны. Почтовое отделение в поселке имеет ту же координату, что и сам поселок. Необходимо написать программу, которая, учитывая расположение поселков и количество строящихся почтовых отделений, выбирает местоположение почтового отделения таким образом, чтобы общая сумма всех расстояний между поселками и, ближайшими к ним, почтовыми отделениями была минимальна.

Входные данные: Первая строка содержит два целых числа:
Первое - количество поселков N (1<=N<=300)
Второе – количество почтовых отделений P (1<=P<=30, P<=N)
Вторая строка содержит отсортированные по возрастанию целые числа X, разделенные пробелом - координаты поселков(1<=X<=10000)

Выходные данные: Первая строка – содержит сумму расстояний между поселками и ближайшими к ним почтовыми отделениями. Вторая строка содержит P целых чисел, записанных через пробел в порядке возрастания – координаты построенных почтовых отделений.

Пример:

Входные данные	Выходные данные
10 5 1 2 3 6 7 9 11 22 44 50	9 2 7 22 44 50

Как видно из текста программы приводимой в качестве решения. Данная задача была придумана и решена программистом Шао Зенг (Shao Zheng) в 1999 г. .

Решение:

Ограничение по времени тестирования: по 3 секунде на один тест.

```
{$APPTYPE CONSOLE}
Program Post;
{
IOI2000 Sample Program
```

```
Day 2
Task: Post
Programmer: Shao Zheng
Email: shaoz@sina.com
Date:2000.09.23
Algorithm: Dynamic Programming, O(N^2)
}
Const
  InputFileName='';
  OutputFileName='';
  MaxOfficeCount=30;
  MaxVillageCount=420;

Var
  Fi,Fo:Text;
  OfficeCount,VillageCount:integer;
  Position:array[1..MaxVillageCount]of integer;

Procedure Init;
var vc:integer;
begin
  assign(Fi,InputFileName);
  assign(Fo,OutputFileName);
  reset(Fi);
  read(Fi,VillageCount,OfficeCount);
  for vc:=1 to VillageCount do read(Fi,Position[vc]);
  close(fi);
end;

Function GetCost(left,right:integer):Longint;
var
  mean:integer;
  v:integer;
  result1:longint;
begin
  mean:=Position[(left+right)div 2];
  result1:=0;
  for v:=left to right do
    inc(result1,abs(mean-Position[v]));
  GetCost:=result1;
end;

Type
  TBackWay=array[1..MaxOfficeCount,1..MaxVillageCount]of integer;
Var
```

```
  Cost:array[1..MaxOfficeCount,1..MaxVillageCount]of longint;
  BackWay:^TBackWay;

Procedure Process;
var o,v,i:integer;
  temp:Longint;
begin
  for o:=1 to OfficeCount do
   for v:=1 to VillageCount do
    begin
      Cost[o,v]:=MaxLongint;
      BackWay^[o,v]:=-1;
    end;
  for v:=1 to VillageCount do Cost[1,v]:=GetCost(1,v);
  for o:=2 to OfficeCount do
   for v:=o to VillageCount do
    for i:=o-1 to v-1 do
     begin
       temp:=Cost[o-1,i]+GetCost(i+1,v);
       if temp<Cost[o,v] then
         begin Cost[o,v]:=temp;
               BackWay^[o,v]:=i;
         end;
     end;
end;

Procedure Print;
var officelist:array[1..MaxOfficeCount]of integer;
  v,o:integer;
begin
  rewrite(Fo);
  writeln(Fo,Cost[OfficeCount,VillageCount]);
  v:=VillageCount;
  for o:=OfficeCount downto 1 do
    begin
      if o=1 then officelist[o]:=(v+1)div 2 else
        officelist[o]:=(v+BackWay^[o,v]+1)div 2;
      v:=BackWay^[o,v];
    end;
  write(Fo,Position[officelist[1]]);
  for o:=2 to OfficeCount do
    write(Fo,' ',Position[officelist[o]]);
  writeln(fo);
  close(Fo);
end;
```

```
{Main Program}
BEGIN
  new(BackWay);
  Init;
  Process;
  Print;
END.
```

Жадные алгоритмы

Одним из методов оптимизации являются, так называемые, жадные алгоритмы. Жадные алгоритмы используются в тех случаях, когда на каждом шаге алгоритма можно оптимизировать только решение данного шага, не рассматривая оптимальность конечного результата, то есть последовательность оптимальных шагов дает оптимальный конечный результат.

1. На складе имеются различные предметы, каждый из которых имеет вес P[i] и стоимость S[i]. Экспедитору выделили на один рейс грузовой автомобиль с грузоподъемностью M, с условием привести в магазин товаров с наибольшей общей стоимостью. Составить алгоритм решения данной задачи.

Алгоритм решения данной задачи:

При решении данной задачи необходимо определить удельную стоимость данного предмета, то есть h[i]=S[i]/P[i], записать все h[i] в массив, затем отсортировать этот массив по убыванию. В таком же порядке произвести перестановки в массивах P[i] и S[i]. После указанных действий необходимо начать загрузку автомобиля предметами с наибольшей удельной стоимостью. Если груз превысит грузоподъемность автомобиля, то необходимо удалить последний предмет и начать помещать (из оставшихся) предметы подходящего веса но с наибольшей удельной стоимостью.

ГРАФЫ

Граф – математическая структура, используемая при исследовании связей между объектами. Вершины – объекты. ребра (дуги) –связи между объектами. Графы широко используются в программировании при решении задач связанных с нахождением оптимальных маршрутов, минимальной стоимости перевозок и т.д. Дадим некоторые определения:
Ориентированный граф это пара (V,E), где V- конечное множество, а E – Бинарное отношение на V, то есть подмножество множества V×V. V- множество вершин, E – множество ребер. Графически вершины изображаются, например, кружками или квадратиками, а ребра в ориентированном графе стрелками

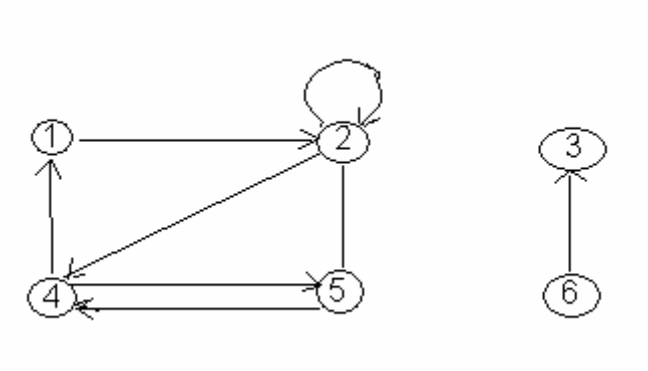


Рис. 1

Ребро соединяющее вершину саму с собой называется ребром циклом

Неориентированный граф. В неориентированном графе ребра это набор неупорядоченных пар вершин (u,v). $u, v \in V$ и u ≠ v. Неориентированный граф не может содержать ребер циклов. В неориентированном графе - (u,v) и (v,u) – это одно и тоже ребро.

Про ребро неориентированного графа говорят, что оно инциндентно вершинам.

Если в графе G имеется ребро (u,v), то говорят, что u смежна с вершиной v.

Степенью вершины в неориентированном графе называется число инцидентных ей ребер.

Путь длины к – это число ребер, которое соединяет вершину v_0 с вершиной v_k., вершина v_0- начало пути, v_k- конец пути. Если существует путь из u в v, то говорят, что вершина v достижима из v. Путь называется простым, если все вершины, через которые он проходит, разные.

Подпуть пути p =(v_0,..,v_k) получится, если мы возьмем некоторое количество подряд вершин этого пути.

Циклом ориентированного графа называется путь в котором конечная и начальная вершины совпадают и который содержит хотя бы одно ребро.

Цикл называется простым, если в нем нет одинаковых вершин (кроме первой и последней)

Ориентированный граф не содержащий циклов называется простым.

В неориентированном графе путь p=(v_0,..,v_k) называется простым циклом, если k≥3 и v_0= v_k и все вершины принадлежащие пути, кроме v_0, v_k, различны.

Граф, в котором нет циклов, называется ациклическим.

Неориентированный граф называется связным, если для любой пар вершин есть путь из одной в другую.

Ориентированный граф называется сильно связным, если из его любой вершины достижима (по ориентированным путям) любая другая.

Два графа G=(v,u) и Gp=(vp,up) называются изоморфными, если существует взаимно однозначное соответствие f: V→Vp между множествами их верши, при котором ребрам одного графа соответствуют ребра другого,

$(u,v) \in E$ тогда и только тогда, когда $(f(u), f(v)) \in Ep$

Можно сказать, что изоморфные графы это одни и тот же граф, в котором вершины названы по-разному.

Граф Gh(Vh,Eh) называется подграфом графа G(V,E), если $Eh \subseteq E$ и $Vh \subseteq V$

Некоторые виды графов имеют специальные названия:

Полным графом называют неориентированный граф, содержащий все возможные для данного множества вершин (две любые вершины - смежны)

Неориентированный граф называют двудольным, если множество вершин V можно разбить на две части V1 и V2, таким образом, что концы любого ребра оказываются в разных частях.

Ациклический неориентированный граф называют лесом.

Связный ациклический неориентированный граф называют деревом без корня

Мультиграф – неориентированный граф, две вершины, которого могут соединяться не одним, а несколькими ребрами.

Графы обычно задают либо списком смежных вершин, либо матрицами смежности:

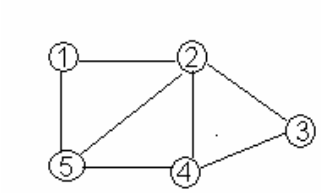


Рис.2

Неориентированный граф (рис.2) может быть представлен следующей таблицей (матрицей):

0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 1
1 1 0 1 0

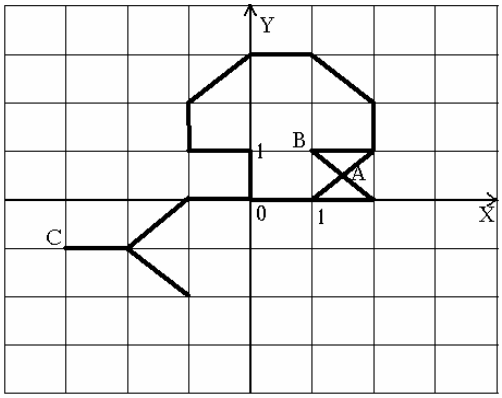
Для ориентированных графов практически то же самое, только матрица уже будет не симметричной. Если в ориетированном графе имеются ребра циклы, то в клетке (i,i) , будет поставлена 1.Если пути имеют разный вес, например разное время горения, то вместо 1 в матрице ставятся соответствующие веса.

Рассмотрим следующие задачи:

1. На клеточном поле введена система координат так, что центр координат находится в точке пересечения линий сетки и оси направлены вдоль линий сетки.

На этом поле выложили связную фигуру, состоящую из спичек. Использовались спички двух типов:

- Спички длины 1 выкладывались по сторонам клеток.
- Спички длины $\sqrt{2}$ выкладывались по диагоналям клеток.



Ребенок хочет сжечь фигуру. При этом он может поджечь ее в одной точке, имеющей целочисленные координаты (например, в точке A на рисунке поджигать фигуру нельзя, а в точках B и C — можно).

Известно, что огонь распространяется вдоль спички равномерно (но по каждой спичке — со своей скоростью). Спичка может гореть в нескольких местах (например, когда она загорается с двух концов; или когда в середине диагональной спички огонь перекидывается с одной спички на другую — огонь расползается по вновь подожженной спичке в обе стороны).

Напишите программу, которая определит, минимальное время сгорания фигуры.

Входные данные: число N — количество спичек ($1 \le N \le 40$). Затем идет N пятерок чисел вида X_1, Y_1, X_2, Y_2, T, задающих координаты концов спички и время ее сгорания при условии, что она будет подожжена с одного конца (гарантируется, что каждая спичка имеет длину 1 или $\sqrt{2}$, все спички образуют связную фигуру, и положение никаких двух спичек не совпадает). Все координаты — целые числа, по модулю не превышающие 200, время сгорания — натуральное число, не превышающее 10^7.

Выходные данные: Выведите наименьшее время, за которое фигура сгорит. Время должно быть выведено с точностью до 2-х знаков после десятичной точки.

Ограничение по времени тестирования: по 1 секунде на один тест.

Примеры:

Входные данные	Выходные данные
1 0 0 1 1 1	1.00
3 1 1 1 2 10 1 2 2 2 10 1 1 2 2 50	35.00

Решение данной задачи прокомментировано.

Решение:

```
program Matches;

Const
  TaskID='f';
  InFile=TaskID+'.in';
  OutFile=TaskID+'.out';

Const
  MaxN=42; { Ограничение на N }
  MaxG=2*MaxN+1; { Ограничение на число вершин в графе }
  Infinity=MaxLongInt; { "Бесконечное" расстояние }

Var
  N:Integer;
  Match:Array[1..MaxN]Of Record  { Входные     }
    X1,Y1,X2,Y2:Integer;         {      данные }
    Time:LongInt;                {Время горения }
  End;

  NG:Integer;
  Vertex:Array[1..MaxG]Of Record                 {Вершины}
    X,Y:Integer;
   End;
  Edge,Distance:Array[1..MaxG,1..MaxG]Of LongInt;{ Ребра – их веса     }

  Res:Extended;      { Минимальное время сгорания }
  ResX,ResY:Integer; { Оптимальная точка поджога  }

Procedure Load;
Var
  I:Integer;
Begin
{  Assign(Input,InFile);
  ReSet(Input);}
  Read(N);
  For I:=1 To N Do
    With Match[I] Do
      Read(X1,Y1,X2,Y2,Time);
{  Close(Input);}
End;

Function GetVertex(VX,VY:Integer):Integer;
```

```
{ Функция, возвращающая номер вершины с заданными координатами.
  При отсутствии нужной вершины она создаётся }
Var
 I:Integer;
Begin
 For I:=1 To NG Do
  With Vertex[I] Do
   If (X=VX) And (Y=VY) Then Begin
    GetVertex:=I;
    Exit;
   End;

 Inc(NG); { Если нужная вершина не найдена }
 With Vertex[NG] Do Begin
  X:=VX;
  Y:=VY;
  For I:=1 To NG-1 Do Begin
   Edge[I,NG]:=Infinity;
   Edge[NG,I]:=Infinity;
  End;
  Edge[NG,NG]:=0;
 End;
 GetVertex:=NG;
End;

Procedure AddEdge(X1,Y1,X2,Y2:Integer; Time:Longint);
 { Функция, добавляющая ребро между двумя точками }
Var
 A,B:Integer;
Begin
 A:=GetVertex(X1,Y1);
 B:=GetVertex(X2,Y2);
 Edge[A,B]:=Time;
 Edge[B,A]:=Time;
End;

Procedure BuildGraph; { Процедура построения графа }
Var
 I:Integer;
Begin
 NG:=0;
 For I:=1 To N Do
  With Match[I] Do Begin
   AddEdge(X1*2,Y1*2,X1+X2,Y1+Y2,Time);
```

```
          AddEdge(X1+X2,Y1+Y2,X2*2,Y2*2,Time);
        End;
      End;

      Procedure FindShortestPaths;
      Var
        K,I,J:Integer;
      Begin
        Distance:=Edge;
        For K:=1 To NG Do
          For I:=1 To NG Do If Distance[I,K]<Infinity Then
            For J:=1 To NG Do If Distance[K,J]<Infinity Then
              If Distance[I,K]+Distance[K,J]<Distance[I,J] Then
                Distance[I,J]:=Distance[I,K]+Distance[K,J];
      End;

      Function BurnAt(At:Integer):Extended;
        { Функция, вычисляющая время сгорания при поджоге в точке At }
      Var
        I,J:Integer;
        Cur,ThisEdge:Extended;
      Begin
        Cur:=0;
        For I:=1 To NG Do If Distance[At,I]>Cur Then Cur:=Distance[At,I];
        For I:=1 To NG Do
          For J:=I+1 To NG Do If Edge[I,J]<Infinity Then Begin
            If (Distance[At,I]<Distance[At,J]+Edge[I,J]) And
              (Distance[At,J]<Distance[At,I]+Edge[I,J]) Then Begin
              If Distance[At,I]<Distance[At,J] Then
                ThisEdge:=Distance[At,J]+(Edge[I,J]-(Distance[At,J]-
Distance[At,I]))/2
              Else
                ThisEdge:=Distance[At,I]+(Edge[I,J]-(Distance[At,I]-
Distance[At,J]))/2;
              If ThisEdge>Cur Then Cur:=ThisEdge;
            End;
          End;
        BurnAt:=Cur;
      End;

      Procedure Solve; {Решение}
      Var
        I:Integer;
        Cur:Extended;
      Begin
```

```
  Res:=Infinity;
  For I:=1 To NG Do
    With Vertex[I] Do
      If Not Odd(X) And Not Odd(Y) Then Begin
        Cur:=BurnAt(I);
        If Cur<Res Then Begin
          Res:=Cur;
          ResX:=X Div 2;
          ResY:=Y Div 2;
        End;
      End;
End;

Procedure Save;
Begin
{  Assign(Output,OutFile);
  ReWrite(Output);
  WriteLn(ResX,' ',ResY);}
  WriteLn(Res/2:0:2);
{  Close(Output);}
End;

Begin
  Load;
  BuildGraph;
  FindShortestPaths;
  Solve;
  Save;
End.
```

2. Имеется некоторый город M, который связан маршрутами с городами A_1, A_2,…, A_m. Пусть, согласно расписанию, маршрут MA_iM обслуживается в интервале времени $[a_i, b_i]$. Другими словами, a_i – ‘это тот момент, начиная с которого самолет связан с маршрутом MA_iM, а b_i – тот момент, когда эта связь прекращается. Таким образом, задано m временных интервалов $[a_1, b_1]$, …, $[a_m, b_m]$. Укажите минимальное число самолетов, достаточное для обслуживания всех рейсов в течение суток.

Входные данные: Первая строка содержит целое число M – число городов (1<=M<=100). Последующие M строк содержат пары вещественных чисел a_i b_i, задающих расписание рейсов.

Выходные данные: Число

Ограничение по времени тестирования: по 1 секунде на один тест.

Пример:

Входные данные	Выходные данные
8 1.48 7.53 17.79 18.59 16.50 21.46 7.72 13.74 17.02 22.33 6.80 23.97 2.32 5.15 1.50 13.79	4

Идея решения данной задачи заключается в следующем: Расписанию рейсов сопоставляется граф, Граф представляется в виде списка смежных вершин следующим образом: Начало и конец каждого рейса заносятся в запись, содержащих два вещественных поля и два поля указателя. Из записей создается двухсвязный список, который сортируется по времени вылета. Берется рейс с самым ранним временем вылета (первый элемент в списке), затем перемещаясь по списку от головы к концу находят запись, в которой время вылета не перекрывается с временем прилета записанном в первом элементе. Далее то же повторяется с этой записью. Таким образом набираются рейсы для одного самолета. Далее, записи, выбранные для первого самолета, удаляются из списка и по такому же алгоритму набираются рейсы для следующих самолетов.

Решение:

```
type
 uk=^zp;
 zp=record
     a,b:real;
     pr,lv:uk;
    end;
var a,b:uk;
m,i:integer;
{--------------}
procedure sort(var h:uk);
var p,t,mak:uk;
begin
 p:=nil;
 while h<>nil do
 begin
  t:=h^.pr;
  mak:=h;
  while t<>nil do
   begin
```

```
    if t^.a>mak^.a then
     mak:=t;
    t:=t^.pr;
   end;
   if mak=h then
    h:=h^.pr;
   if mak^.pr<>nil then
    mak^.pr^.lv:=mak^.lv;
   if mak^.lv<>nil then
   mak^.lv^.pr:=mak^.pr;
   mak^.lv:=nil;
   mak^.pr:=nil;
   if p<>nil then
    begin
     mak^.pr:=p;
     p^.lv:=mak;
    end;
     p:=mak;
 end;
 h:=p;
end;
{--------------}
function count(h:uk):integer;
var b:real;
t,s:uk;
c:integer;
begin
 c:=0;
 while h<>nil do
  begin
   b:=h^.b;
   inc(c);
   t:=h;
   h:=h^.pr;
   dispose(t);
   t:=h;
   while t<>nil do
    begin
     while (t<>nil)and(t^.a<b) do
      t:=t^.pr;
     if t<> nil then
      begin
       b:=t^.b;
       s:=t^.pr;
       t^.lv^.pr:=s;
```

```
       if t=h then
        h:=h^.pr;
       if t^.lv<> nil then
        t^.lv^.pr:=t^.pr;
       if t^.pr<>nil then
        t^.pr^.lv:=t^.lv;
       dispose(t);
       t:=s;
      end;
    end;
  end;
  count:=c;
end;
{--------------}
Begin

 read(m);
 b:=nil;
 for i:=1 to m do
  begin
   new(a);
   a^.lv:=nil;
   a^.pr:=nil;
   read(a^.a,a^.b);
   if b<>nil then
    begin
     a^.pr:=b;
     b^.lv:=a;
    end;
   b:=a;
  end;
 sort(b);
 writeln(count(b));
end.
```

ВЫЧИСЛИТЕЛЬНЫЕ ЗАДАЧИ

1. Числа Фибоначчи F1, F2, ... определяются начальными значениями и соотношением: F1=1, F2=2, F(N)=F(N-1)+F(N-2). Рассмотрим систему счисления с двумя цифрами 0 и 1, в которой в отличие от двоичной системы весами являются не степени двойки 1, 2, 4, 8, 16, ..., а числа Фибоначчи 1, 2, 3, 5, 8, 13, ... В этой системе счисления каждое положительное целое число

единственным способом представляется в виде строки из нулей и единиц, в которой нет двух единиц, стоящих рядом.
Даны две строки, представляющие числа A и B в данной системе счисления.
Требуется написать программу, которая находит строку, представляющую число A+B.
Например, исходные строки "10101" и "100" представляют числа 8+3+1=12 и 3. Ответом является строка "100010", представляющая число 13+2=15=12+3.

Входные данные: В первой строке представление числа A, а во второй - B. Количество знаков в каждом из представлений не превышает 255.

Выходные данные: Найденное представление суммы A+B.

Ограничение по времени тестирования: по 1 сек. на тест

Пример:

Входные данные	Выходные данные
10101 100	100010

Данная задача решается следующим образом:

Считываются две строки, состоящих и 0 и 1, затем эти строки с помощью функции perevod преобразуются в числа (десятичная система). Эти десятичные числа складываются, после чего результат преобразуется в строку нулей и единиц с помощью функции perevod2.

Решение:

```
var s1,s2,s:string;
a,b:real;
i,n,j:integer;
{***********}
function perevod(s:string):real;
var
f1,f2,f:real;
i:integer;
r:real;
begin
 r:=0;
 f1:=1;
 f2:=1;
 for i:=length(s) downto 1 do
  begin
   r:=r+(ord(s[i])-48)*f1;
   f:=f1;
   f1:=f2+f1;
   f2:=f;
  end;
 perevod:=r;
```

```
end;
{************}
function perevod2(a:real):string;
var f1,f2,f:real;
n:byte;
r:string;
begin
 r:='';
 for n:=1 to 255 do
  r:=r+'0';
 while a>0 do
  begin
   f1:=1;
   f2:=1;
   n:=1;
   while f1<=a do
    begin
     f:=f1;
     f1:=f2+f1;
     f2:=f;
     n:=n+1;
    end;
    a:=a-f2;
    n:=n-1;
    r[255-n+1]:='1';
  end;
  n:=1;
  while r[n]='0' do n:=n+1;
  delete(r,1,n-1);
  perevod2:=r;
end;
{************}
begin
 readln(s1);
 readln(s2);
 a:=perevod(s1);
 b:=perevod(s2);
 writeln(perevod2(a+b));
end.
```

2. Дан вещественный массив X: $\{X_0,X_1,\ldots X_{N-1}\}$, имеющий N элементов ($N=2048=2^{11}$). Элементы массива Y вычисляются по формуле:

$Y_k = \sum_{m=0}^{N-1} X_m \cdot \sin(\frac{2 \cdot \pi}{N} \cdot k \cdot m)$. Составьте алгоритм[1] вычисления всех $Y_{k,}$

Входные данные: последовательность элементов $X_{k,}$ разделенных пробелом.
Выходные данные: последовательность элементов Y_k (с точностью до 3-х знаков после десятичной точки), разделенных пробелом.
Ограничение по времени тестирования: по 1 сек. на тест
Алгоритм решения данной задачи:

Подобные задачи при большом объеме данных требуют значительных временных затрат, поэтому необходимо искать пути по сокращению времени выполнения таких задач. Отметим, что выражение стоящее под знаком синуса принимает дискретный ряд значений, так как период синуса с данным аргументом равен $2 \cdot \pi$, то значения $\sin(\frac{2 \cdot \pi}{N} \cdot k) = \sin(\frac{2 \cdot \pi}{N} \cdot (k + N)$ в точности равны. Поэтому вместо того, чтобы вычислять синус N^2 раз (значительные временные затраты), достаточно вычислить значения синуса только N раз и записать в массив. Если значения аргумента выходят за пределы периода, то аргумент надо приводить к значению лежащему внутри периода. Выполнение подобных действий сокращает время вычислений в десятки раз. Для дальнейшей оптимизации следует учесть, что умножение вещественных чисел требует гораздо больших затрат машинного времени, чем операция сложения, поэтому временные затраты будут меньше, если вычислить $(a + b) * c$ вместо $a * c + b * c$. В данном алгоритме это учитывается при вычислении $X_k * \sin(\frac{2\pi m}{N}) + X_{N-m} * \sin(\frac{2\pi(N-m)}{N}) = (X_m - X_{n-m}) \sin(\frac{2\pi m}{N})$
Данные действия помимо сокращения времени вычисления позволяют сократить и интервал перебора индекса m от 0 до N div 2. Далее можно отметить, что $Y_k = -Y_{N-k}$, Последнее позволяет вычислять только N div 2 значений . Y_k Вторая половина массива Y при копировании первой половины массива в обратном порядке и изменении знака копируемых элементов.

ХЕШ-ФУНКЦИИ (Хеширование)

В ряде случаев когда, число данных значительно, становиться невозможной их запись в оперативной памяти ЭВМ, то есть мы не можем пользоваться массивами. В этих случаях создают динамические, которые называют хеш – таблицами. В эти таблицах создаются указатели на часто использующиеся данные. Элементы хеш – таблиц являются значениями некоторой функции, которая называется хеш – функцией. От того, как вы зададите хеш – функцию зависит, как быстро вы будете находит часто использующиеся данные. Мы рассмотрим с вами способ построения хеш – функции, который называется методом деления с остатком.

[1] Если при вычислениях вы не укладываетесь в ограничения повремени, то постарайтесь придумать способ оптимизации процесса вычислений.

Этот метод состоит в том, что ключу k ставится в соответствие остаток от деления k, где m – число возможных хеш – значении. Этот факт записывается следующим образом: h(k)= k mod m. Некоторых значений m следует избегать, например если $m=2^p$, то h(k) – это просто p младших битов, числа k, если нет уверенности, что все комбинации младших битов ключа будут встречаться с одинаковой частотой, то степень двойки в качестве ключа m не выбирают. Хорошие результаты получаются, если в качестве m выбирают простое число далеко отстоящее от степени двойки.

Далее рассмотрим следующую задачу:

Дан текст. Необходимо определить K - количество повторений наиболее часто встречающегося слова. Слова могут состоять только из букв заданного алфавита. Разделителем слов могут быть любые символы не входящие в заданный алфавит. Алфавит задается строкой не превышающей 255 символов.

Входные данные: Первая строка содержит все символы алфавита. Последующие строки содержат текст, состоящий из слов указанного алфавита и разделителей (символов не входящих в алфавит).

Примечание: 1. Все слова записаны строчными символами. 2. Длина слова не превышает 20 символов. 3. Количество повторений одного слова не превышает 65 тыс.

Выходные данные: K.

Ограничение по времени: по 1 секунд на один тест.

Пример:

Вход	Выход
Абвгдежзийклмнопрстуфхцчшщъыьэюяё мама мыла раму. Мама мыла папу.	2

В данной задаче осуществляется перебор символов текста и символов алфавита, сравнение этих символов позволяет выделить символы разделители и затем выделить слова. Слова и число их повторений заносятся в хеш – таблицу (число повторений - поле count). По хеш – таблице и определяется максимальное число повторений.

Решение:

```
type
  { элемент хеш-таблицы }
  PHTItem = ^THTItem;
  THTItem = record
    key: string[20];
    count: Word;
    next: PHTItem;
  end;

const
  HashTableSize = 701;  { размер хеш-таблицы }
```

```
  PNCount = 53;         { массив простых чисел из отрезка [0..255] }
  PrimeNumbers: array[1..PNCount] of byte =
    (3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,
     89,97,101,103,107,109,113,127,131,137,139,149,151,157,163,167,
     173,179,181,191,193,197,199,211,223,227,229,233,239,241,251);

var
  { сама хеш-таблица }
  HashTable: array[0..HashTableSize-1] of PHTItem;

  alf: set of char;
  s,key: string;
  L,i,fi,HashValue,MaxCount: Word;
  p: PHTItem;


procedure InitHashTable;
var
  i: word;
begin
  for i:=0 to HashTableSize-1 do
    HashTable[i]:=Nil;
end;

{ хеш-функция }
function HashFunc(const s: string): word;
var
  i,L: integer;
  r: Word;
begin
  L:=Length(s);
  r:=0;
  for i:=1 to L do
    r:=(r+PrimeNumbers[i]*ord(s[i])) mod HashTableSize;
  HashFunc:=r;
end;

begin
  alf:=[];
  MaxCount:=0;
  InitHashTable;
  { чтение алфавита }
  Readln(s);
  L:=Length(s);
  for i:=1 to L do
```

```
  alf:=alf+[s[i]];

 { чтение текста: по строкам и выделение в них слов }
 while not Eof do begin  { цикл по строкам входного файла }
  readln(s);
  L:=Length(s);
  i:=1;
  while i<L do begin { цикл по словам строки }
   { пропуск пробельных символов }
   while i<=L do begin
    if s[i] in alf then Break;
    Inc(i);
   end;
   if i>L then Break; { строка кончилась }
   { чтение очередного слова слова }
   fi:=i;
   while i<=L do begin
    if not(s[i] in alf) then Break;
    Inc(i);
   end;
   key:=copy(s,fi,i-fi);
   // поиск слова в таблице
   HashValue:=HashFunc(key);
   p:=Hashtable[HashValue];
   while p<>Nil do begin
    if p^.key = key then Break;
    p:=p^.next;
   end;
   // если поиск был неудачным - добавить новый элемент
   if p=Nil then begin
    new(p);
    p^.key:=key;
    p^.count:=1;
    p^.next:=Hashtable[HashValue];
    Hashtable[HashValue]:=p;
   end else begin
    Inc(p^.count);
   end;
   if p^.count > MaxCount then MaxCount:=p^.count;
  end;
 end;
 WriteLn(MaxCount);
end.
```

МНОЖЕСТВА

В задаче, которая будет рассмотрена ниже, используется множество в, которое быть занесено 2000000 целых неотрицательных чисел. Каждому числу соответствует свой бит, если число принадлежит множеству, то соответствующий бит равен 1, иначе 0. Эти биты связаны с элементами массива размерностью 2000000 div 8. Первые 8 чисел это биты первого элемента массива, причем число 1 это младший бит, а число 8 старший бит первого элемента массива. Следующие восемь чисел уже относятся ко второму элементу массива. По программе будет видно, как записывается принадлежность данного числа к множеству, как заносится данное число в множество. Поэтому если встречаются одинаковые числа, то они входят в множество только один раз, то есть это один и тот же бит. Поэтому, чтобы сосчитать различные, необходимо сосчитать все биты значения, которых 1.

Дана последовательность из N целых неотрицательных чисел C1,C2, … , Cn (1<N<=1 000 000; 0<=Ci<=2 000 000). Посчитать K – количество различных чисел в данной последовательности.
Входные данные: в первой строке записано число N, далее следует N строк, содержащих числа C1,C2, … ,Cn по одному в каждой строке.
Выходные данные: K.
Ограничение по времени: по 3 секунды на один тест.
Пример:

Вход	Выход
10 1 2 3 4 5 2 3 1 4 0	6

Решение:

```
const
  max_num = 2000000;
  asize = max_num div 8;
var
  n,c,i,ByteNo,counter: longint;
  a: array[0..asize] of byte;
  k: byte;
begin
```

```
  for i:=0 to asize do a[i]:=0;
  read(n);
  counter:=0;
  for i:=1 to n do begin
    read(c);
    ByteNo:=c div 8; {Определяется номер байта}
    k:=(1 shl (c mod 8)); {Бит вдвигается влево на число c mod 8  }
    if (a[ByteNo] and k)=0 then begin
{Проверка принадлежности к множеству}
      Inc(counter);
      a[ByteNo]:=a[ByteNo] or k;
    end;
  end;
  write(counter);
end.
```

ЛИТЕРАТУРА

1. Столяр С. Алгоритмы: СПб.: ЦПО “Информатизация образования”
2. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ. Москва: МЦНМО, 1999
3. Крючкова Е. Теория алгоритмов и формальных языков.: Барнаул: АлтГТУ, 2000.
4. Черкасова П. Компьютер и графы. СПб.: “Информатизация образования”, 2000
5. Шень А. Программирование: теоремы и задачи. М.: МЦНМО, 1995
6. Андреева Е. Принципы проверки проверки учебных и олимпиадных задач по информатике. // Информатика N34, 2001
7. Сборник “Компьютор и задачи выбора”. М.: “Наука”, 1989
8. Юркин А. Практикум по программированию. Барнаул: АГУ, 1999