动态规划基本原理

近年来，涉及动态规划的各种竞赛题越来越多，每一年的 NOI 几乎都至少有一道题目需要用动态规划的方法来解决；而竞赛对选手运用动态规划知识的要求也越来越高，已经不再停留于简单的递推和建模上了。

要了解动态规划的概念，首先要知道什么是多阶段决策问题。

一、多阶段决策问题

如果一类活动过程可以分为若干个互相联系的阶段，在每一个阶段都需作出决策(采取措施)，一个阶段的决策确定以后，常常影响到下一个阶段的决策，从而就完全确定了一个过程的活动路线，则称它为多阶段决策问题。

各个阶段的决策构成一个决策序列，称为一个策略。每一个阶段都有若干个决策可供选择，因而就有许多策略供我们选取，对应于一个策略可以确定活动的效果，这个效果可以用数量来确定。策略不同，效果也不同，多阶段决策问题，就是要在可以选择的那些策略中间，选取一个最优策略，使在预定的标准下达到最好的效果.

让我们先来看下面的例子：如图所示的是一个带权有向的多段图，要求从 A 到 D 的最短路径的长度(下

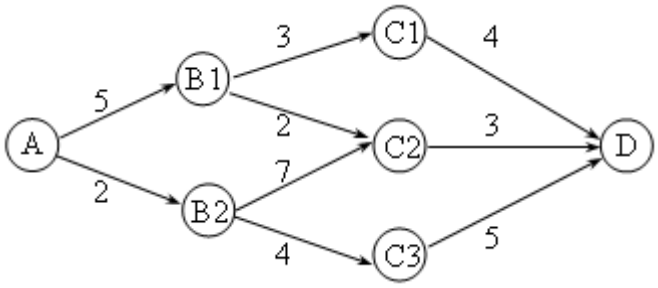


图 4-1 带权有向多段图

面简称最短距离)。

我们可以搜索，枚举图中的每条路径，但当图的规模大起来时，搜索的效率显然不可能尽人意。让我们来试用动态规划的思路分析这道题：从图中可以看到，A 点要到达 D 点必然要经过 B1 和 B2 中的一个，所以 A 到 D 的最短距离必然等于 B1 到 D 的最短距离加上 5，或是 B2 到 D 的最短距离加上 2。同样的，B1 到 D 的最短距离必然等于 C1 到 D 的最短距离加上 3 或是 C2 到 D 的最短距离加上 2，……。

我们设 G[i]为点 i 到点 D 的距离，显然 G[C1]=4，G[C2]=3，G[C3]=5，根据上面的分析，有：

G[B1]=min{G[C1]+3，G[C2]+2}=5，

G[B2]=min{G[C2]+7，G[C3]+4}=9，

再就有 G[A]=min{G[B1]+5，G[B2]+2}=10，

所以 A 到 D 的最短距离是 10，最短路径是 A→B1→C2→D。

二、动态规划的术语

1．阶段

把所给求解问题的过程恰当地分成若干个相互联系的阶段，以便于求解，过程不同，阶段数就可能不同．描述阶段的变量称为阶段变量。在多数情况下，阶段变量是离散的，用 k 表示。此外，也有阶段变量是连续的情形。如果过程可以在任何时刻作出决策，且在任意两个不同的时刻之间允许有无穷多个决策时，

阶段变量就是连续的。

在前面的例子中，第一个阶段就是点 A，而第二个阶段就是点 A 到点 B，第三个阶段是点 B 到点 C，而第四个阶段是点 C 到点 D。

2．状态

状态表示每个阶段开始面临的自然状况或客观条件，它不以人们的主观意志为转移，也称为不可控因素。在上面的例子中状态就是某阶段的出发位置，它既是该阶段某路的起点，同时又是前一阶段某支路的终点。

在前面的例子中，第一个阶段有一个状态即 A，而第二个阶段有两个状态 B1 和 B2，第三个阶段是三个状态 C1，C2 和 C3，而第四个阶段又是一个状态 D。

过程的状态通常可以用一个或一组数来描述，称为状态变量。一般，状态是离散的，但有时为了方便也将状态取成连续的。当然，在现实生活中，由于变量形式的限制，所有的状态都是离散的，但从分析的观点，有时将状态作为连续的处理将会有很大的好处。此外，状态可以有多个分量(多维情形)，因而用向量来代表；而且在每个阶段的状态维数可以不同。

当过程按所有可能不同的方式发展时，过程各段的状态变量将在某一确定的范围内取值。状态变量取值的集合称为状态集合。

3．无后效性

我们要求状态具有下面的性质：如果给定某一阶段的状态，则在这一阶段以后过程的发展不受这阶段以前各段状态的影响，所有各阶段都确定时，整个过程也就确定了。换句话说，过程的每一次实现可以用一个状态序列表示，在前面的例子中每阶段的状态是该线路的始点，确定了这些点的序列，整个线路也就完全确定。从某一阶段以后的线路开始，当这段的始点给定时，不受以前线路（所通过的点）的影响。状态的这个性质意味着过程的历史只能通过当前的状态去影响它的未来的发展，这个性质称为无后效性。

4．决策

一个阶段的状态给定以后，从该状态演变到下一阶段某个状态的一种选择（行动）称为决策。在最优控制中，也称为控制。在许多问题中，决策可以自然而然地表示为一个数或一组数。不同的决策对应着不同的数值。描述决策的变量称决策变量，因状态满足无后效性，故在每个阶段选择决策时只需考虑当前的状态而无须考虑过程的历史。

决策变量的范围称为允许决策集合。

5．策略

由每个阶段的决策组成的序列称为策略。对于每一个实际的多阶段决策过程，可供选取的策略有一定的范围限制，这个范围称为允许策略集合。允许策略集合中达到最优效果的策略称为最优策略。

给定 k 阶段状态变量 x(k)的值后，如果这一阶段的决策变量一经确定，第 k+1 阶段的状态变量 x(k+1)也就完全确定，即 x(k+1)的值随 x(k)和第 k 阶段的决策 u(k)的值变化而变化，那么可以把这一关系看成(x(k)，u(k))与 x(k+1)确定的对应关系，用 x(k+1)=Tk(x(k),u(k))表示。这是从 k 阶段到 k+1 阶段的状态转移规律，称为状态转移方程。

6．最优性原理

作为整个过程的最优策略，它满足：相对前面决策所形成的状态而言，余下的子策略必然构成“最优子策略”。

最优性原理实际上是要求问题的最优策略的子策略也是最优。让我们通过对前面的例子再分析来具体说明这一点：从 A 到 D，我们知道，最短路径是 A→B1→C2→D，这些点的选择构成了这个例子的最优策略，根据最优性原理，这个策略的每个子策略应是最优：A→B1→C2 是 A 到 C2 的最短路径，B1→C2→D 也是 B1 到 D 的最短路径……事实正是如此，因此我们认为这个例子满足最优性原理的要求。

下面我们列举历年国内国际竞赛的一些典型动态规划问题加以分析。

机器分配（HNOI’95）

一、问题描述

总公司拥有高效生产设备 M 台，准备分给下属的 N 个公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这 M 台设备才能使国家得到的盈利最大？求出最大盈利值。其中 M《=15，N〈=10。分配原则：每个公司有权获得任意数目的设备，但总台数不得超过总设备数 M。保存数据的文件名从键盘输入。

数据文件格式为：第一行保存两个数，第一个数是设备台数 M，第二个数是分公司数 N。接下来是一个 M*N 的矩阵，表明了第 I 个公司分配 J 台机器的盈利。

二、分析

这是一个典型的动态规划试题。用机器数来做状态，数组 F[I，J]表示前 I 个公司分配 J 台机器的最大盈利。则状态转移方程为

F[I，J]=Max{F[I-1，K] + Value[I，J-K]} （0〈=K〈=J〉

三、参考程序：

```
program hnoi_95_4;
  var
    a , b , c    : array[0..15,0..15] of longint;
    d    : array[0..15] of longint;
    n , m : longint;
  procedure init;
    var
      fi          : text;
      I , j , k    : integer;
    Begin
      Assign(fi,’input.txt’); Reset(fi);
      Readln(fi,m,n);
      For I:= 0 to m do
        Begin
          Read(fi,j);
           For k := 1 to n do
               Read(fi,a[k,I]);
          Readln(fi);
        End;
      Close(fi);
    End;
```

```
Procedure dynamic;
    Var I , j , k : longint;
        Begin
            For I := 1 to n do
                For j := 0 to m do
                    For k := 0 to j do
                        If b[I-1,k] + a[I,j-k] > b[I,j] then begin
                            B[I,j] := b[I-1,k] + a[I,j-k];
                        End;
                Writeln(fo,b[n,m]);
            End;
Begin
    Init;
    Dynamic;
End.
```

最长不下降序列（HNOI'97）

一、问题描述

设有整数序列 b1,b2,b3,…,bm，若存在 i1<i2<i3<…<in，且 bi1<bi2<bi3<…<bin，则称 b1,b2,b3,…,bm 中有长度为 n 的不下降序列 bi1,bi2,bi3,…,bin。求序列 b1,b2,b3,…,bm 中所有长度(n)最大不下降子序列

输入：整数序列

输出：最大长度 n 和所有长度为 n 的序列个数 Total

二、分析

此题分为两个部分：求最大不下降序列长度和序列个数。

首先我们考虑求最大长度。设 F(i)为前 I 个数中的最大不下降序列长度。由题意不难得知，要求 F(i)，需要求得 F(1)—F(I-1)，然后选择一个最大的 F(j) (j<I, bi>bj)，那么前 I 个数中最大不下降序列便是前 j 个数中最大不下降序列后添加 bi 而得。可见此任务满足最优子结构，可以用动态规划解决。

通过上面的分析可得状态转移方程如下：

F(i)=max{F(j)+1}(j<I , bj<bi)

边界为 F(1)=1

显然只需要求序列个数即可。很容易想到下面的方法：

设 Total(I)为前 I 个数中最长不下降序列的个数。

那么，要求 Total(i)，只需找到所有的 Total(j)满足 j<I，bj<bi 且前 j 个数最大不下降序列长度为前 I 个数中最大不下降序列长度+1，并把他们累加起来。即

Total(i)=∑Total(j) (j<I, bj<bi, F(i)=F(j)-1)

在求所有的 Total(i)(F(i)=max)的累加和即可。

但这样考虑有一个致命的错误，如

1 2 2

这样一个序列，最大不下降序列长度为 2。但如果按上述方法计算序列 1 2 会算两次。因此，我们对算法进行改进：

对原序列按 b 从小到大（当 bi=bj 时按 F 从大到小）排序，增设 Order(i)记录新序列中的 I 个数在原序列中的位置。可见，当求 Total(i)时，当 F(j)=F(j+1) , b(j)=b(j+1)且 Order(j+1)<Order(i)时，便不累加 Total(j)。这样就避免了重复。

综合看来，上述算法的时间复杂度为 O(n^2)，空间复杂度为 O(n)，都是可行的。

三、参考程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+}
{$M 65520,0,655360}
Program HNOI97_1;
  const finp          ='input.txt';
        fout          ='output.txt';
        maxN          =1000;
var   n,len           :integer;    {len 记录最大长度}
      tot             :longint;    {tot  记录序列个数}
      f,b             :array[1..maxN]of integer;

Procedure init;    {输入}
 var f    :text;
 begin
   assign(f,finp);reset(f);
   n:=0;
   while not eoln(f) do begin
     inc(n);
     read(f,b[n])
   end;
   close(f)
 end;

Procedure main;
 var i,j,t :integer;
     order :array[1..maxN]of integer;
     total :array[1..maxN]of longint;
 begin
   f[1]:=1;len:=1;        {求解最大不下降序列长度}
   for i:=2 to n do begin
     f[i]:=1;
     for j:=1 to i-1 do
       if (b[i]>b[j])and(f[i]<f[j]+1) then
         f[i]:=f[j]+1;
     if f[i]>len then len:=f[i]   {记录最大值}
   end;
  for i:=1 to n do order[i]:=i;
  for i:=1 to n do                  {排序}
    for j:=i+1 to n do
       if (b[i]>b[j])or(b[i]=b[j])and(f[i]>f[j]) then begin
         t:=b[i];b[i]:=b[j];b[j]:=t;
         t:=f[i];f[i]:=f[j];f[j]:=t;
         t:=order[i];order[i]:=order[j];order[j]:=t
```

```
        end;
    tot:=0;                    {计算序列个数}
    fillchar(total,sizeof(total),0);
     for i:=1 to n do begin
      if f[i]=1 then total[i]:=1
        else
          for j:=i-1 downto 1 do
            if (f[j]=f[i]-1)and(b[j]<b[i])and(order[j]<order[i]) then
              if (b[j+1]<>b[j])or(f[j+1]<>f[j])or(order[j+1]>=order[i]) then
                inc(total[i],total[j]);
      if (f[i]=len)and(b[i]<>b[i+1]) then        {记录累加最终值}
        tot:=tot+total[i]
    end;
 end;

Procedure out;      {输出}
 begin
    assign(output,fout);rewrite(output);
    writeln(len);
    writeln(tot);
    close(output)
 end;

Begin
 init;
 main;
 out
End.
```

凸多边形三角划分（HNOI'97）

一、试题描述

给定一个具有 N（N<50）个顶点（从 1 到 N 编号）的凸多边形，每个顶点的权均已知。问如何把这个凸多边形划分成 N-2 个互不相交的三角形，使得这些三角形顶点的权的乘积之和最小？

输入文件：第一行 顶点数 N
　　第二行 N 个顶点（从 1 到 N）的权值

输出格式：最小的和的值
　　各三角形组成的方式

输入示例：5
122　123　245　231

输出示例：The　minimum　is ： 12214884
　　The　formation　of 3 triangle:
　　3 4 5, 1 5 3, 1 2 3

二、试题分析

这是一道很典型的动态规划问题。设 F[I,J]（I<J）表示从顶点 I 到顶点 J 的凸多边形三角剖分后所得到的最大乘积，我们可以得到下面的动态转移方程：

F[I,J]=Min{F[I,K]+F[K,J]+S[I]*S[J]*S[K]}　　(I<K<J)

目标状态为：F[1,N]

但我们可以发现,由于这里为乘积之和，在输入数据较大时有可能超过长整形甚至实形的范围，所以我们还需用高精度计算，但这是大家的基本功，程序中就没有写了，请读者自行完成。

三、参考程序

```
Var S             :Array[1..50] Of Integer;
    F             :Array[1..50,1..50] Of Comp;
    D            :Array[1..50,1..50] Of Byte;
    N             :Integer;
Procedure Init;    (*输入数据*)
Var I             :Integer;
Begin
  Readln(N);
  For I:=1 To N Do Read(S[I]);
End;
Procedure Dynamic;    (*动态规划*)
Var I,J,K         :Integer;
Begin
  For I:=1 To N Do
    For J:=I+1 To N Do F[I,J]:=Maxlongint;  (*赋初始值*)
  For I:=N-2 Downto 1 Do
    For J:=I+2 To N Do
      For K:=I+1 To J-1 Do
        If (F[I,J]>F[I,K]+F[K,J]+S[I]*S[J]*S[K]) Then
          Begin
            F[I,J]:=F[I,K]+F[K,J]+S[I]*S[J]*S[K];
            D[I,J]:=K;      (*记录父节点*)
          End;
End;
Procedure Print(I,J:Integer);  (*输出每个三角形*)
Begin
  If J=I+1 Then Exit;
  Write(',',I,' ',J,' ',D[I,J]);
  Out(I,D[I,J]);
  Out(D[I,J],J);
End;
Procedure Out;  (*输出信息*)
Begin
  Assign(Output,'Output.Txt'); Rewrite(Output);
  Writeln('The minimum is :',F[1,N]:0:0);
  Writeln('The formation of ',N-2,' triangle:');
```

```
    Write(1,' ',N,' 'D[1,N]);
    Out(1,D[1,N]);
    Out(D[1,N],N);
    Close(Output);
  End;
  Begin  (主程序)
    Init;
    Dynamic;
    Out;
  End.
```

系统可靠性（HNOI'98）

一、问题描述：

给定一些系统备用件的单价 Ck，以及当用 Mk 个此备用件时部件的正常工作概率 Pk（Mk），总费用上限 C。求系统可能的最高可靠性。

二、算法分析

1．证明这个问题符合最优化原理。可以用反证法证明之。假设用 money 的资金购买了前 I 项备用件，得到了最高的系统可靠性，而且又存在如下情况：对于备用件 I，设要买 Ki 个，则可用的资金还剩余 money – Ci*Ki，用这些钱购买前（I-1）项备用件，如果存在一种前（I-1）种备用件的购买方案得到的系统可靠性比当前得到的要高，那么这个新的方案会使得整个前 I 项备用件组成的系统可靠性比原先的更高，与原假设矛盾。所以可以证明这个问题符合最优化原理。

2．证明这个问题符合无后效性原则。

3．综上所述，本题适合于用动态规划求解。

4．递推方程及边界条件：

F[I,money] := max { F[I-1,money – C[I]*K[I]] } (0<=K[I]<= C div Ci)

三、参考程序

```
{$Q-,R-,S-}
{$M 16384,0,655360}
Program system_dependability;
  const finp='input.txt';
        fout='output.txt';
        maxm=3000;
  var f,p:array[0..maxm] of real;
      max,v:double;
      c,co,e,i,j,k,m,n:integer;
  procedure print;
    var output:text;
    begin
      assign(output,fout); rewrite(output);
      writeln(f[c]:1:4);
      close(output);
    end;
```

```
Begin
  assign(input,finp); reset(input);
  readln(input,n,c);
  for i:=0 to c do f[i]:=1;
  for i:=1 to n do begin
    read(input,co); m:=c div co;
    for e:=0 to m do read(input,p[e]);
    for j:=c downto 0 do begin
      m:=j div co; max:=0;
      for k:=0 to m do begin
        v:=f[j-k*co]*p[k];
        if v>max then max:=v;
      end;
      f[j]:=max;
    end;
  end;
  close(input);
  print
End.
```

快餐问题（HNOI'99）

一、问题描述

Peter 最近在 R 市开了一家快餐店，为了招揽顾客，该快餐店准备推出一种套餐，该套餐由 A 个汉堡，B 个薯条和 C 个饮料组成。价格便宜。为了提高产量，Peter 从著名的麦当劳公司引进了 N 条生产线。所有的生产线都可以生产汉堡，薯条和饮料，由于每条生产线每天所能提供的生产时间是有限的、不同的，而汉堡，薯条和饮料的单位生产时间又不同。这使得 Peter 很为难，不知道如何安排生产才能使一天中生产的套餐产量最大。请你编一程序，计算一天中套餐的最大生产量。为简单起见，假设汉堡、薯条和饮料的日产量不超过 100 个。

输入：

输入文件共四行。第一行为三个不超过 100 的正整数 A、B、C 中间以一个空格分开。第三行为 3 个不超过 100 的正整数 p1,p2,p3 分别为汉堡，薯条和饮料的单位生产耗时。中间以一个空格分开。第三行为 N(0<=0<=10)，第四行为 N 个不超过 10000 的正整数，分别为各条生产流水线每天提供的生产时间，中间以一个空格分开。

输出：

输出文件仅一行，即每天套餐的最大产量。

输入输出示例：

Input2.txt

2 2 2

1 2 2

2

6 6

output.txt

1

二、分析

本题是一个非常典型的资源分配问题。由于每条生产线的生产是相互独立，不互相影响的，所以此题可以以生产线为阶段用动态规划求解。

状态表示：　用 p[I,j,k]表示前 I 条生产线生产 j 个汉堡，k 个薯条的情况下最多可生产饮料的个数。

用 r[I,j,k]表示第 I 条生产线生产 j 个汉堡，k 个薯条的情况下最多可生产饮料的个数。

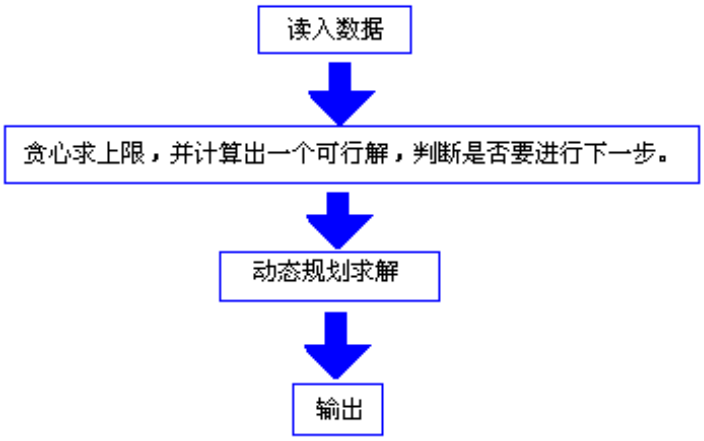
态转移方程 ：　p[I,j,k] = Max{p[I-1,j1,k1]+r[I,j-j1,k-k1]}

(0<=j1<=j,0<=k1<=k,且（j-j1）*p1+(k-k1)*p2<= 第 I 条生产线的生产时间)

但这样的算法是非常粗糙的，稍加分析可以发现，此算法的时间复杂度为 $O(N*100^4)$,当 N=10 的时候，时间复杂度将达到 $10*100^4=10^9$，这是根本无法承受的。

于是，我们必须对算法进行进一步的优化，经仔细观察可以发现：这道题中存在着一个上限值（Min{100 div A, 100 div B, 100 div C}）,这是一个很好的判断条件，他可以帮我们尽早地求出最优解。为什么这样说呢？因为，在动态规划开始前，我们可以先用贪心法算出 N 条生产线可以生产的套数，如果大于上限值则可以直接输出上限值并退出。否则再调用动态规划，而在动态规划求解的过程中，也可以在每完成一阶段工作便和上限值进行比较，若大于等于上限值就退出，这样一来，就可以尽早的求出解并结束程序。

具体的算法流程为：



三、小结

动态规划虽然是种高效的算法，但不加优化的话，有可能在时间和空间上仍然有问题，因此，在做题时要充分发掘题目中的隐含条件，并充分利用已知条件，这样才能令程序更快，更优。

四．对程序优化的进一步讨论:

本题在具体实现中还有一些优化的余地，在此提出，以供参考：

（1）　存储结构：由于每一阶段状态只与上一阶段状态有关，所以我们可以只用两个 100*100 的数组滚动实现。但考虑到滚动是有大量的赋值，可以改进为动态数组，每次交换时只需交换指针即可，这样比原来数组间赋值要快。

（2）　减少循环：在计算每一阶段状态过程中无疑要用到 4 重循环，其实这当中有很多是可以省掉的，我们可以这样修改每一重循环的起始值：

原起始值：

```
for I := 1 to n do begin
  for j := 0 to tot[I] div p1 do
    for k := 0 to (tot[I]-p1*j) div p2 do
      for j1:=0 to j do

      for k1 := 0 to k do
```

改进后的起始值：

```
for I := 1 to n do begin
 for j := 0 to min(q1,tot[I] div p1) do
   for k := 0 to min(q2,(tot[I]-p1*j) div p2) do
     for j1 := max(0,j-t[I] div p1) to
                 min(j,tot[I-1] div p1) do
     for k1 := max(0,k-(t[I]-(j-j1)*p1) div p2)
```

```
                                                    to min(k,(tot[I-1]-p1*j1)div p2) do
{ 注：具体变量用途请参考程序  }
```

五、参考程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R+,S+,T-,V+,X+,Y+}
{$M 65520,0,655360}

program FastFood;
 const
   input          = 'input2.txt';
   output         = 'output2.txt';
 var
   f              : text;
   r,p,pp         : array[0..100,0..100] of integer;        {pp:滚动数组中存放前一阶段的数组}
   t,tot,tt       : array[0..10] of longint;    {tt:辅助数组；t:每条生产线的生产线时间；
                                                  tot:1-I 条生产线生产时间的总和}
   n,a,b,c,p1,p2,p3      : longint;      {a,b,c:汉堡，薯条，饮料的个数；
                                          p1,p2,p3 汉堡，薯条，饮料的生产单位耗时}

 procedure init;
  var
    i                                     : integer;
  begin
    assign(f,input);
    reset (f);
    readln(f,a,b,c);
    readlN(f,p1,p2,p3);
    readln(f,n);
    for i := 1 to n do read(f,t[i]);
    close (f);
    if n=0 then begin   { 特殊情况处理 }
        assign(f,output);
        rewrite(f);
        writeln(f,0);
        close (f);
        halt;
    end;
  end;

 function   min(i,j : longint) : longint;     {求两数中较小的一个}
  begin
        if i>j then min := j else min := i;
  end;

 function   max(i,j : longint) : longint;     {求两数中较大的一个}
```

```
 begin
       if i>j then max := i else max := j;
 end;

procedure main;
 var
   q,q1,q2,i,j,k,j1,k1                    : longint;   {q:上限值；q1,q2 : A,B 的上限值}
 begin
   q := min( 100 div a,min( 100 div b, 100 div c ) ); {求上限值}
   q1 := q*a;
   q2 := q*b;
   tt := t;
   i := 1;
   j := q1*p1;
   while (j>0) and (i<=n) do     { 用贪心法求出 N 条生产线可以生产的套数（可行解）}
         if j>tt[i] then begin
            dec(j,tt[i]);   tt[i] := 0; inc(i);
         end
         else begin dec(tt[i],j); j := 0; end;
   if j=0 then begin
      j := q2*p2;
      while (j>0) and (i<=n) do
            if j>tt[i] then begin
               dec(j,tt[i]) ;tt[i] := 0; inc(i);
            end
            else begin dec(tt[i],j); j := 0; end;
      if j=0 then begin   {如果可行，直接输出上限值}
         assign(f,output);
         rewrite(f);
         writeln(f,q);
         close (f);
         halt;
      end;
   end;
   tot[0] := 0;
   for i := 1 to n do tot[i] := tot[i-1] + t[i];
   if tot[n] div (a*p1+b*p2+c*p3)<q then begin   {否则修改上限值}
      q := tot[n] div (a*p1+b*p2+c*p3);
      q1 := q*a;
      q2 := q*b;
   end;
   for i := 1 to n do begin
       for j := 0 to min(q1,t[i] div p1) do
           for k := 0 to min(q2,(t[i]-p1*j) div p2) do
```

```
                r[j,k] := (t[i]-p1*j-p2*k) div p3;
        fillchar(p,sizeof(p),0);                    {数组清零}
        for j := 0 to min(q1,tot[i] div p1) do
            for k := 0 to min(q2,(tot[i]-p1*j) div p2) do
              for j1 := max(0,j-t[i] div p1) to min(j,tot[i-1] div p1) do
                for k1 := max(0,k-(t[i]-(j-j1)*p1) div p2) to min(k,(tot[i-1]-p1*j1) div p2) do
                    if p[j,k] < pp[j1,k1] + r[j-j1,k-k1] then
                        p[j,k] := pp[j1,k1] + r[j-j1,k-k1];
            if p[q*a,q*b]>=q*c then begin
            {如果在此阶段产生了不小于上限值的解，则之际输出上限值，并直接退出}
            assign(f,output);
            rewrite(f);
            writeln(f,q);
            close (f);
            halt;
        end;
        pp := p;
        for j := 0 to min(100,tot[i] div p1) do
            for k := 0 to min(100,(tot[i]-p1*j) div p2) do
                if p[j,k] > 100 then p[j,k] := 100;
    end;
    { out }
    k1 := 0;                    { 输出最优值  }
    for j := 0 to 100 do
        if (j div a > k1) then
            for k := 0 to 100 do
                if (k div b > k1) and (p[j,k] div c > k1 ) then
                    k1 := min( min( j div a, k div b), p[j,k] div c );
    assign(f,output);
    rewrite(f);
    writeln(f,k1);
    close (f);
  end;

 begin
   init;
   main;
 end.
```

求函数最大值(CTSC'95)

一、问题描述

已知三个函数A,B,C值如下表所示。自变量取值为0-10的整数。请用动态规划的方法求出一组x,y,z。使得A(x)+B(y)+C(z)为最大,并且满足x*x+y*y+z*z<N，N由键盘输入。

X	0	1	2	3	4	5	6	7	8	9	10

A(x)	2	4	7	11	13	15	18	22	18	15	11
B(x)	5	10	15	20	24	18	12	9	5	3	1
C(x)	8	12	17	22	19	16	14	11	9	7	4

二、思路分析

本题已经说明了算法是动态规划，我们接下来要做的就是求他的状态转移方程。稍加分析可得到状态转移方程：

F[I,j]=max(f[i-1,j-x*x]+p[I,x]);

其中，A(X)=p[1,x]，B(X)=P[2,X]，C(X)=P[3,X];

F[I,J]表示前 I 个函数的自变量平方和为 j 时函数和的最大值。易证，此规划方程满足无后效性。得到函数最大值后，我们再用自底向上追溯的方法求出 X,Y,Z,的值,具体实现请看程序。

三、参考程序

```
program CTSC_95_4;
const
  f   : array[1..3,0..10] of integer =                    {函数}
  ((2,  4,  7, 11, 13, 15, 18, 22, 18, 15, 11),
   (5, 10, 15, 20, 24, 18, 12,  9,  5,  3,  1),
   (8, 12, 17, 22, 19, 16, 14, 11,  9,  7,  4));
var
  s   : array[1..3,0..300] of longint;   {状态 S[I,J]表示前 I 个函数自变量值的平方和为 j 时函数和的最大值}
  pre: array[1..3,0..300] of integer;   {前驱数组，保存该状态是由上阶段哪个状态得到的}
  n   : longint;

procedure main;
var
  i,j,k,p,x,y,z : integer;
begin
  write('n=');   readlN(n);
  if n>301 then n:=301;   {如果 N>301(10*10+10*10+10*10)则 N=301}
  fillchar(s,sizeof(s),0);
  fillchar(pre,sizeof(pre),0);
  for i := 0 to 10 do s[1,i*i] := f[1,i];
  for i := 1 to 2 do
      for j := 0 to n-1 do begin
          if s[i,j] > s[i+1,j] then
              begin s[i+1,j] := s[i,j]; pre[i+1,j] := j; end;
          for k := 0 to 10 do
              if s[i,j]+f[i+1,k]>s[i+1,j+k*k] then begin
                  p := j+k*k;
                  if p>=n then break;
                  pre[i+1,p] := j;
                  s[i+1,p] := s[i,j]+f[i+1,k];
              end;
      end;
```

```
  p := 0; k := 0;
  for i := 0 to n-1 do
      if s[3,i]>k then begin
          k := s[3,i]; p := i;
      end;
  z := trunc(sqrt(p-pre[3,p])); p := pre[3,p];   {求出 Z}
  y := trunc(sqrt(p-pre[2,p])); p := pre[2,p];   {求出 Y}
  x := trunc(sqrt(p)); {求出 X}
  writeln('Max=',k);
  writeln('A=',x,',','B=',y,',','C=',z);
end;

begin
  main;
end.
```

石子合并（NOI'95）

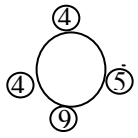
一、问题描述

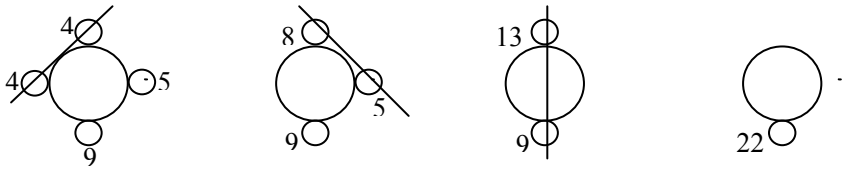
在一园形操场四周摆放 N 堆石子(N≤100),现要将石子有次序地合并成一堆.规定每次只能选相临的两堆合并成一堆,并将新的一堆的石子数,记为该次合并的得分.

编一程序,由文件读入堆数 N 及每堆石子数(≤20),

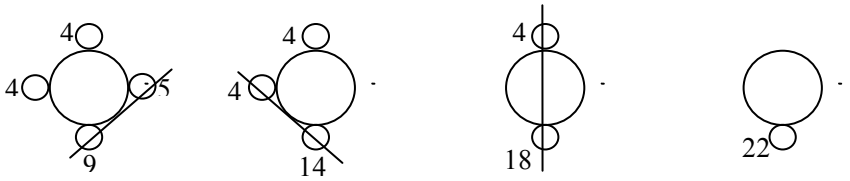
(1)选择一种合并石子的方案,使得做 N-1 次合并,得分的总和最少

(2) 选择一种合并石子的方案,使得做 N-1 次合并,得分的总和最大





总得分=8+13+22=43



总得分=14+18+22=54

输入数据:

文件名由键盘输入,该文件内容为:

第一行为石子堆数 N;

第二行为每堆石子数,每两个数之间用一空格分隔.

输出数据 :

输出文件名为 OUTPUT.TXT

从第 1 至第 N 行为得分最小的合并方案. 第 N+1 行为空行.从 N+2 到 2N+1 行是得分最大的合并方案.

每种合并方案用 N 行表示,其中第 i 行(1≤i≤N)表示第 i 次合并前各堆的石子数(依顺时钟次序输出,哪一堆先输出均可). 要求将待合并的两堆石子数以相应的负数表示,以便识别,参见 MODEL2.TXT

参考输入文件 EXAMPLE2.TXT

```
4
4 5 9 4
```

参考输出文件 MODEL2.TXT

```
-4 5 9 -4
-8 -5 9
-9 -13
-22

4 -5 -9 4
4 -14 -4
-4 –18
-22
```

二、分析

看到本题，容易想到使用贪心法，即每次选取相邻最大或最小的两堆石子合并。

然而这样做对不对呢？看一个例子。

N=5　石子数分别为 3 4 6 5 4 2。

用贪心法的合并过程如下：

第一次 [3 4] 6 5 4 [2] 得分 5

第二次 [5 4] 6 5 4 得分 9

第三次 9 6 [5 4] 得分 9

第四次 [9 6] 9 得分 15

第五次 [15 9] 得分 24

第六次 24

总分：62

然而仔细琢磨后，发现更好的方案：

第一次 [3 4] 6 5 4 2 得分 7

第二次 [7 6] 5 4 2 得分 13

第三次 13 5 [4 2] 得分 6

第四次 13 [5 6] 得分 11

第五次 [13 11] 得分 24

第六次 24

总分：61

显然，贪心法是错误的。

为什么？

显然，贪心只能导致局部的最优，而局部最优并不导致全局最优。

仔细分析后，我们发现此题可用动态规划进行解决。

我们用 data[I,j]表示将从第 i 颗石子开始的接下来 j 颗石子合并所得的分值，

max[i,j]表示将从第 i 颗石子开始的接下来 j 颗石子合并可能的最大值，那么：

max[I,j] = max(max[i, k] + max[i + k, j – k] + data[I,k] + data[I + k, j – k]) (2<=k<=j)

max[i,1] = 0

同样的，我们用 min[i,j]表示将第从第 i 颗石子开始的接下来 j 颗石子合并所得的最小值，可以得到类似的方程：

min[I,j] = min(min[i, k] + min[i + k, j – k] + data[I,k] + data[I + k, j – k]) (0<=k<=j)

min[I,0] = 0

这样，我们完美地解决了这道题。空间复杂度为 $O(n^2)$，时间复杂度也是 $O(n^2)$。

三、小结

通过解决这道题，我们应认识到，对于任何一道题，都不能被其表象所迷惑，应认清问题的实质，从而采取有效的算法解决。

游览街区（NOI'97）

一、问题描述

某旅游区的街道成网格状（见图例）。其中东西向的街道都是旅游街，南北向的街道都是林荫道。由于游客众多，旅游街被规定为单行道，游客在旅游街上只能从西向东走，在林荫道上可以从南向北走，也可从北向南走。

阿隆想到这个旅游区游玩。他的好友阿福给了他一些建议，用分值表示所有旅游街相邻两个路口之间值得游览的程度，分值是从-100 到 100 的整数，所有林荫道不打分。所有分值不可能全是负分。

例如下图是被打过分的某旅游区的街道图：

北

	-50	-47	36	-30	-23
东	17	-19	-34	-13	-8
	-42	-3	-43	34	-45

西

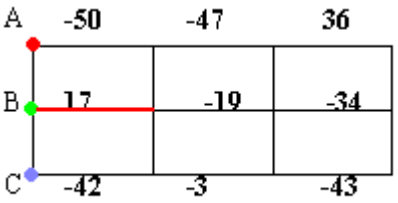
南

阿隆可以从任一个路口开始游览，在任一个路口结束游览。请你写一个程序，帮助阿隆找一条最佳的游览路线，使得这条路线的所有分值总和最大。

二、分析

由于本题网格的规模很大（最大达到 100*20001），所以用一般方法将网格直接记录下来不是很好。于是我们将问题转化以下：

由于南北向可以任意走，而东西向只能从西向东走，那么我们在选择路径的时候一定会**选一条分值最大的格线**向东走。如图右图，无论在 A、B、C 中的那一点，都能通过南北向的移动而选择分度值最大的格线 (分值为 17)。对于这一点的证明如下：



如果有一条最大分值和路经，而其中某段 P(I,j)—P(I,j+1)的分值比与之并列的格线 P(I,k)小，那么当到达点 P(I,j)时，必可通过南北向移动修改 P(I,j)—>P(I,k) —>P(I,k+1)—>P(I,j+1)而得到一条新路经比原路经分值和更大。所以原路经不是最大分值和路径。

因此我们就可以得到这样个数列：

设 F(i)为第 I 列所有格线中最大分值。

由题可知，最大分值即为： Max{F[I]+F[I+1]+…+F[j]}(1<=I<=j<=n)

所以我们将问题转化为：**求数列中连续最大和问题**。

对于求连续最大和问题，很容易想到用动态规划。

设 G(i)为以第 I 个数结尾的连续最大和。由于当 G(I-1)>0 时，G(i)就是 G(I-1)基础上添加一个 F(i)；而如果 G(I-1)<=0，则 G(i)只能取 F(i)本身，否则就不是最大和。可得到如下递推方程：

G(i)=Max{G(I-1)+F(i),F(i)}(n>=I>1)

边界为 G(0)=0

三、参考程序

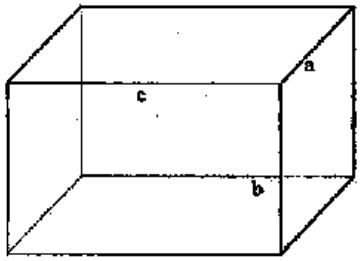
```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 16384,0,655360}
Program Tour;

const finp          ='input.txt';
      fout          ='output.txt';
      maxN          =20001;

var   map           :array[1..maxN]of shortint;          {记录 F 表}
      m,n           :integer;
      buffer        :array[1..40960]of char;
      s             :longint;

procedure init;          {输入}
 var i,j   :integer;
     x     :shortint;
 begin
   assign(input,finp);reset(input);
   Settextbuf(input,buffer);
   readln(m,n);
   fillchar(map,sizeof(map),$80);
   for i:=1 to m do begin
     for j:=1 to n-1 do begin
       read(x);
       if x>map[j] then map[j]:=x
     end;
     readln
   end;
   close(input)
 end;

procedure main;
 var t     :longint;
     i     :integer;
 begin
   s:=0;t:=0;
   for i:=1 to n-1 do begin
     if s+map[i]>map[i] then s:=s+map[i]
                        else s:=map[i];          {状态转移方程}
```

```
        if s>t then t:=s                    {记录最优值}
      end
  end;
?procedure out;      {输出}
procedure out;      {输出}
  begin
    assign(output,fout);rewrite(output);
    writeln(s);
    close(output)
  end;

Begin        {主程序}
  init;
  main;
  out
End.
```

积木游戏（NOI'97）

一、问题描述

一种积木游戏，游戏者有 N 块编号依次为 1，2，…，N 的长方体积木。第 I 块积木通过同一顶点三条边的长度分别为 ai,bi,ci（i=1,2,...,N），如图 1 所示：



游戏规则如下：

1 从 N 块积木中选出若干块，并将他们摞成 M（1<= M <= N）根柱子，编号依次为 1,2,...,M，要求第 k 根柱子的任意一块积木的编号都必须大于第 K-1 根柱子任意一块积木的编号（2<=K<=M）。

2 对于每一根柱子，一定要满足下面三个条件：

除最顶上的一块积木外，任意一块积木的上表面同且仅同另一块积木的下表面接触；

对于任意两块上下表面相接触的积木，若 m,n 是下面一块积木接触面的两条边（m>=n），x,y 是上面一块积木接触面的两条边（x>=y），则一定满足 m.>=x 和 n>=y；

下面的积木的编号要小于上面的积木的编号。

请你编一程序，寻找一种游戏方案，使得所有能摞成的 M 根柱子的高度之和最大。

输入数据：

文件的第一行是两个正整数 N 和 M（1<= M <= N <=100），分别表示积木总数和要求摞成的柱子数。这两个数之间用一个空格符隔开。接下来的 N 行是编号从 1 到 N 个积木的尺寸，每行有三个 1 至 500 之间的整数，分别表示该积木三条边的长度。同一行相邻两个数之间用一个空格符隔开。

输出数据：

文件只有一行，是一个整数，表示所求得的游戏方案中 M 根柱子的高度之和。

二、分析

由于题目中条件的限制：（1）积木已经编了号；（2）后面的柱子中的积木编号必须比前面的柱子中的积木编号大：（3）对于同一根柱子，上面的积木的编号必须大于下面的积木的编号，因此使得这道题的无后效性很明显，因为对于第 I 块积木，它的状态只取决于前 I-1 块积木，与后面的积木无关。

这样我们很自然地想到了动态规划。下面我们来讨论状态转移方程。由于一块积木可以任意翻转，因此它的上表面有三种情况，对应的三个面的长和宽分别为：a 和 b, a 和 c, b 和 c.。设（1） f[I,j,k]表示以第 I 块积木的第 k 面为第 j 根柱子的顶面的最优方案的高度总合；（2）block[I,k] 记录每个积木的三条边信息（block[I,4]:=block[I,1]; block[I,5]:=block[I,2]）。其中 block[I,j+2]表示当把第 I 块积木的第 j 面朝上时所对应的高，即所增加的高度；（3）can[I,k,p,kc]表示第 I 块积木以其第 k 面朝上，第 p 块积木以第 kc 面朝上时，能否将积木 I 放在积木 p 的上面。1 表示能，0 表示不能。对于 f[i,j,k], 有两种可能：（1）除去第 I 块积木，第 j 根柱子的最上面的积木为编号为 p 的，若第 p 块积木以第 kc 面朝上，必须保证当第 I 块积木以 k 面朝上时能够被放在上面，即 can[I,k,p,kc]=1；（2）第 I 块积木是第 j 根柱子的第一块积木，第 j-1 根柱子的最上面为第 p 块积木，则此时第 p 块积木可以以任意一面朝上。则有：

$$f[i,j,k]=\max\begin{cases}\max\{f[p,j,kc]+block[i,j+2]\}\ (1<=p<=i-1,1<=kc<=3,\\ \qquad\qquad\qquad\qquad\qquad\qquad 且 can[i,k,p,kc]=1)\\ \max\{f[p,j-1,kc]+block[i,j+2]\}\ (1<=p<=i-1,1<=kc<=3)\end{cases}$$

边界条件：

f[1,1,1]:=block[1,1,3]; f[1,1,2]:=block[1,1,4]; f[1,1,3]:=block[1,1,5];

f[I,0,k]:=0; (1<= I <= n, 1<= k <= 3);

此算法主要需要存储 block，can 和 f 数组，分别需要 O(n), O(n^2)和 O(n*m)，总和约为 120K。时间复杂度为 O(n^2*m),约为 10^6.

三、参考程序

```
{$A-,B+,D+,E-,F-,G+,I-,L-,N-,O-,P-,Q-,R-,S-,T-,V+,X+,Y+}
{$M 16384,0,655360}

program lcy;
const
  name1                    = 'game.txt';
  name2                    = 'game.out';
  maxn                     = 100;
  maxm                      = 100;

type
  arrtype                 = array [1..maxn,1..3] of byte;
  ftype                    = array [0..maxm,1..3] of longint;

var
  fi,fo                   : text;
  n,m                      : byte;
  can                      : array [1..maxn,1..3] of ^arrtype;
```

```
  block                          : array [1..maxn,1..5] of integer;
  f                              : array [1..maxn] of ^ftype;

procedure sort( w: byte);
var
  i,j,p      : byte;
  tmp        : integer;

begin
  for i:=1 to 2 do
    begin
      p:=i;
      for j:=i+1 to 3 do if block[w,j]<block[w,p] then p:=j;
      tmp:=block[w,i]; block[w,i]:=block[w,p]; block[w,p]:=tmp;
    end;
end;

procedure init;
var
  i            : byte;
begin
  assign(fi,name1); reset(fi);
  assign(fo,name2); rewrite(fo);
  readln(fi,n,m);
  for i:=1 to n do
    begin
      readln(fi,block[i,1],block[i,2],block[i,3]);
      sort(i);   {将三边从大到小排序，便于后面的比较}
      block[i,4]:=block[i,1]; block[i,5]:=block[i,2];
    end;
  close(fi);
end;

procedure cal_can;
var
  i,j,k,p        : byte;

begin
  for i:=1 to n do
    for j:=1 to 3 do
      begin
        new(can[i,j]);
        fillchar(can[i,j]^,sizeof(can[i,j]^),0);
      end;
  for i:=2 to n do
```

```
    for k:=1 to 3 do
      for j:=1 to i-1 do
        for p:=1 to 3 do
          if (block[i,k]<=block[j,p]) and (block[i,k+1]<=block[j,p+1]) then
            can[i,k]^[j,p]:=1;
end;

procedure work;
var
  i,j,k,p,up,down,q          : byte;

begin
  cal_can; （预处理，求 can 数组）
  for i:=1 to n do
    begin
      new(f[i]);
      fillchar(f[i]^,sizeof(f[i]^),0);
    end;
  for i:=1 to 3 do f[1]^[1,i]:=block[1,i+2];

  for i:=2 to n do
    begin
      if i>m then up:=m else up:=i;
      if m+i-n<1 then down:=1 else down:=m+i-n;
      for j:=down to up do
        for k:=1 to 3 do
          for q:=1 to i-1 do
            for p:=1 to 3 do
              begin
                if (can[i,k]^[q,p]=1) and (f[q]^[j,p]<>0) and
                   (f[q]^[j,p]+block[i,k+2]>f[i]^[j,k]) then （第一种可能）
                     f[i]^[j,k]:=f[q]^[j,p]+block[i,k+2];
                if (f[q]^[j-1,p]<>0) and  (f[q]^[j-1,p]+block[i,k+2]>f[i]^[j,k]) then
                        （第二种可能）
                      f[i]^[j,k]:=f[q]^[j-1,p]+block[i,k+2];
              end;
    end;
end;

procedure out;
var
  i            : byte;
  max          : longint;

begin
```

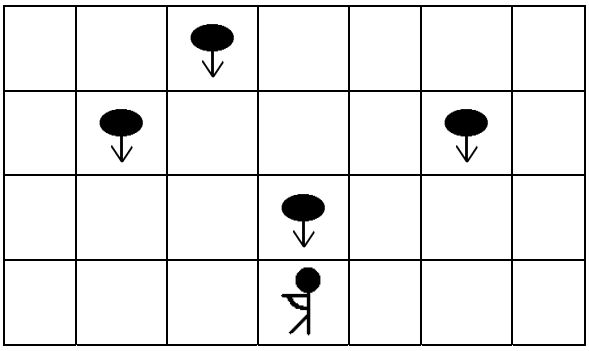
```
  max:=0;
  for i:=1 to 3 do if f[n]^[m,i]>max then max:=f[n]^[m,i];
  writeln(fo,max);
  close(fo);
end;

begin
  init;
  work;
  out;
end.
```

免费馅饼（NOI'98）

一、问题描述

SERKOI 最新推出了一种叫做“免费馅饼”的游戏。

游戏在一个舞台上进行。舞台的宽度为 W 格，天幕的高度为 H 格，游戏者占一格。开始时，游戏者站在舞台的正中央，手里拿着一个托盘（如图一）。



图一

游戏开始后，从舞台天幕顶端的格子中不断出现馅饼并垂直下落。游戏者左右移动去接馅饼。游戏者每秒可以向左或右移动一格或两格，也可以站在愿地不动。

馅饼有很多种，游戏者事先根据自己的口味，对各种馅饼依次打了分。同时在 8-308 电脑的遥控下，各种馅饼下落的速度也是不一样的，下落速度以格/秒为单位。当馅饼在某一秒末恰好到达游戏者所在的格子中，游戏者就收集到了这块馅饼。

写一个程序，帮助我们的游戏者收集馅饼，使得收集的馅饼的分数之和最大。

输入数据：输入文件的第一行是用空格分开的两个正整数，分别给出了舞台的宽度 W（1~99 之间的奇数）和高度 H（1 ～100 之间的整数）。

接下来依馅饼的初始下落时间顺序给出了一块馅饼信息。由四个正整数组成，分别表示了馅饼的初始下落时刻（0～1000 秒），水平位置、下落速度（1～100）以及分值。游戏开始时刻为 0。从 1 开始自左向右依次对水平方向的每格编号。

输出数据：输出文件的第一行给出了一个正整数，表示你的程序所收集到的最大分数之和。

其后的每一行依时间顺序给出了游戏者每秒的决策。输出 0 表示原地不动。1 或 2 表示向右移动一步

或两步、-1 或-2 表示向坐移动一步或两步。输出应持续到游戏者收集完他要收集的最后一块馅饼为止。

三、分析

首先，我们将问题转化。我们将问题中的馅饼信息用一个表格存储。表格的第 I 行第 J 列表示的是游戏者在第 I 秒到达第 J 列所能取得的分值。那么问题便变成了一个类似数字三角形的问题：从表格的第一行开始往下走，每次只能向左或右移动一格或两格，或不移动走到下一行。怎样走才能得到最大的分值。

因此，我们很容易想到用动态规划求解。

F[I,J]表示游戏进行到第 I 秒，这时游戏者站在第 J 列时所能得到的最大分值。那么动态转移方程为：

F[I,J] = Max { F[I-1,K] } + 馅饼的分值 (J-2<=K<=J+2)

另外，由于本题表格的规模会很大，我们可以每次处理 100 行，这样就可以解决空间上的问题。

四、参考程序

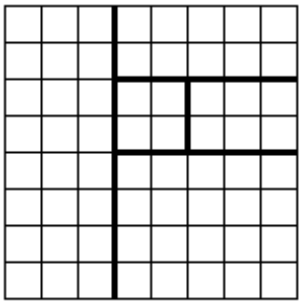
```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R+,S+,T-,V+,X+,Y+}
{$M 65520,0,655360}
Program Ex;
Type Node               =Array [1..99] Of Shortint;
     Node2              =Array [1..99] Of Longint;
     Node3              =Array [1..99] Of Longint;
Var P                   :Array [1..1100] Of ^Node;
    A                   :Array [1..1100] Of ^Node2;
    Now                 :Node3;
    Last                :Node3;
    H,Wi                :Integer;
    Max                 :Integer;
    S                   :Longint;
    Time,X              :Integer;
Procedure Init;
Var U,V,W,N             :Integer;
    I,J,K               :Integer;
Begin
  Max:=-Maxint;
  Assign(Input,'Input.txt');  Reset(Input);
  Readln(Wi,H);
  Fillchar(Last,Sizeof(Last),$FF);
  Last[Wi Div 2+1]:=0;
  For U:=1 To 100 Do Begin
    New(A[U]);
    Fillchar(A[U]^,Sizeof(A[U]^),0);
  End;
  Readln(U,V,W,N);
  I:=1;
  Repeat
    While (U=I) Or (U=0) Do Begin
      If (H-1) Div W=(H-1)/W Then Begin
        Inc(A[U+(H-1) Div W]^[V],N);
```

```
        If U+(H-1) Div W>Max Then Max:=U+(H-1) Div W;
      End;
      If SeekEof(Input) Then Break;
      Readln(U,V,W,N);
    End;
    Fillchar(Now,Sizeof(Now),$FF);
    New(P[I]);
    Fillchar(P[I]^,Sizeof(P[I]^),0);
    For J:=1 To Wi Do Begin
      For K:=-2 To 2 Do
        If (J-K>0) And (J-K<=Wi) Then
          If Last[J-K]>Now[J] Then Begin
            Now[J]:=Last[J-K];
            P[I]^[J]:=K;
          End;
      If Now[J]>-1 Then Begin
        Now[J]:=Now[J]+A[I]^[J];
        If Now[J]>S Then Begin
          S:=Now[J];
          Time:=I;
          X:=J;
        End;
      End;
    End;
    Last:=Now;
    A[I+100]:=A[I];
    Fillchar(A[I+100]^,Sizeof(A[I+100]^),0);
    Inc(I);
  Until (I>Max) And Seekeof(Input);
  Close(Input);
End;
Procedure Out(T,M       :Integer);
Begin
  If T>1 Then Out(T-1,M-P[T]^[M]);
  Writeln(P[T]^[M]);
End;
Begin
  Init;
  Assign(Output,'Output.txt'); Rewrite(Output);
  Writeln(S);
  Out(Time,X);
  Close(Output);
End.
```

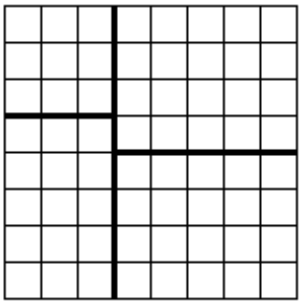
棋盘分割（NOI'99）

一、问题描述

将一个 8 × 8 的棋盘进行如下分割：将原棋盘割下一块矩形棋盘并使剩下部分也是矩形，再将剩下的部分继续如此分割，这样割了(n-1)次后，连同最后剩下的矩形棋盘共有 n 块矩形棋盘。(每次切割都只能沿着棋盘格子的边进行)



允许的分割方案



不允许的分割方案

原棋盘上每一格有一个分值，一块矩形棋盘的总分为其所含各格分值之和。现在需要把棋盘按上述规则分割成 n 块矩形棋盘，并使各矩形棋盘总分的均方差最小。

均方差 $\sigma=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n}}$，其中平均值 $\overline{x}=\frac{\sum_{i=1}^{n}x_i}{n}$，$x_i$ 为第 i 块矩形棋盘的总分。

请编程对给出的棋盘及 n，求出 σ 的最小值。

输入

第 1 行为一个整数 n(1<n<15)。

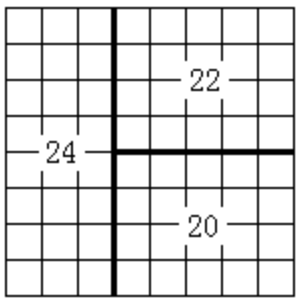
第 2 行至第 9 行每行为 8 个小于 100 的非负整数，表示棋盘上相应格子的分值。每行相邻两数之间用一个空格分隔。

输出

仅一个数，为 σ（四舍五入精确到小数点后三位）。

样例输入

```
3
1 1 1 1 1 1 1 3
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 0 3
```



样例输出

1.633

二、初步分析

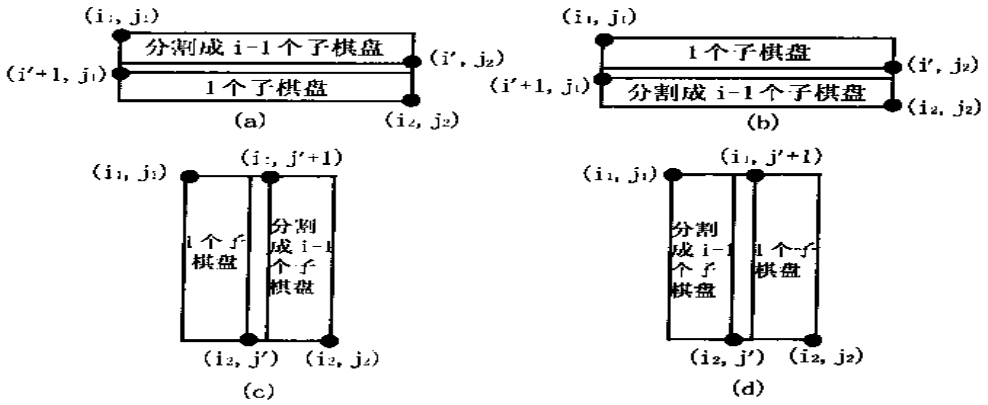
本题的实质是求一种最优的棋盘分割方式，使得每块棋盘与平均值的差值之和最小。

首先我们必须明确几个条件（关键字），这样对我们理解题意进而采取正确的算法解决问题大有帮助：

- 均方差 $\sigma=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n}}$ ：在实际编程的过程中，由于 n 是定值，实际只需求（X_i-X）的和的值作为参数，以此简化程序的计算及存储的空间。
- 本题提供固定的 8 × 8 棋盘，并不算大，这是本题有解的重要条件，并且给我们一个暗示：以棋盘格子为存储单位，将有很大的自由度。

于是我们开始考虑算法：对比八皇后问题的复杂度，我们不难看出这道题需要搜索更多的内容，在时间上搜索算法实不可取；因此，只能使用动态规划实现本题。经过分析，不难发现本题符合最优化原理：即若第 i 次分割为最佳分割，则第 i-1 次分割为且必为最佳；定义函数 F[i,j][i',j']为[i,j]、[i',j']分别为左上、右下角的棋盘的最优值，F_0[i,j][i',j']为[i,j]、[i',j']分别为左上、右下角的棋盘值，探寻函数 F[i,j][i',j']的动态转移方程。

下面分析分割方式。当我们进行第 i 次分割时不外乎以下四种方式：



（图 3.4－3）

逐一进行分析：（图 3.4-3）

1. 横割方式：
   a) 第 i 次切割将棋盘分为上 i-1 个棋盘和下 1 个棋盘（图（a））
      A1=F_0[i1,j1][i',j2]+F[i'+1,j1][i2,j2]
   b) 第 i 次切割将棋盘分为下 i-1 个棋盘和上 1 个棋盘（图（b））
      A2=F[i1,j1][i',j2]+F_0[i'+1,j1][i2,j2]
2. 竖割方式：
   c) 第 i 次切割将棋盘分为右 i-1 个棋盘和左 1 个棋盘（图（c））
      A3=F[i1,j1][i2,j']+F_0[i1,j'+1][i2,j2]
   d) 第 i 次切割将棋盘分为左 i-1 个棋盘和右 1 个棋盘（图（d））
      A3=F_0 [i1,j1][i2,j']+F [i1,j'+1][i2,j2]

状态转移方程为

F[i1, j1][i2, j2]=min{A1, A2, A3, A4}

(1<=i1, j1<=8, i1<=i2<=8, j1<=j2<=8, 2<=k<=n)

其中 k 代表分割的棋盘数，单调递增，因此第 k 次分割只与 k-1 次的结果有关，所以每做完第 k 次对棋盘的规划 $F_0 \leftarrow F$。由此节省下许多空间。

三、程序实现

下面我们讨论程序实现的具体步骤与代码的优化。

首先在读入过程段我们进行准备工作，累加计算出 F_0 并统计出棋盘每个格子值之和 S 来计算平均数 Average。

```
s←0;
for i:=1 to 8 do
    for j:=1 to 8 do begin
        read(f[i,j][i,j]); s←s+f[i,j][i,j];    {读入棋盘每个格子的值，并统计其和}
        for i1:=1 to i do                      {枚举左上方坐标 i1,j1}
            for j1:=1 to j do
                for i2:=i to 8 do
                    for j2:=j to 8 do          {枚举右上方坐标 i2,j2}
                        if   (i1<>i) or (j1<>j) or (i2<>i) or (j2<>j)
                            the f[i1,j1][i2,j2]←f[i1,j1][i2,j2]+f[i,j][i,j];
    end;
```

在套用循环算出 F_0[i1,j1][i2,j2]的值，此处不再赘述。

然后用动态规划求解：

```
for i:=2 to n do begin                         {分阶段，第 i 次分割}
    for i1:=1 to 8 do
     for j1:=1 to 8 do
       for i2:=i1 to 8 do
        for j2:=j1 to 8 do begin               {确定坐上、右下角坐标}
            F[i1,j1][i2,j2]←max;
            for i':=i1 to i2-1 do begin
                计算 A1,A2；
                F[i1,j1][i2,j2]←min{A1,A2};
            end;
            for i':=i1 to i2-1 do begin
                计算 A3,A4；
                F[i1,j1][i2,j2]←min{F[i1,j1][i2,j2],A3,A4};
            end;
        end;
    F0←F;
end;
```

显然问题的解为 $\sqrt{\frac{f_1[1,1]+[8,8]}{n}}$。

三、小结

本题是极有代表性的动态规划题型，较之 NOI99 的其他题目算是比较简单的。此题的思路简单而明了，没有太多限制条件让人梳理不清，空间的自由度很大，唯一的限制便是运行时间。

所谓窥一斑可见全豹，从本题的思考过程中，我们不难总结出应用动态规划算法的一般思路及步骤：

- 确定算法，整体估计可行度。一般从两方面着手：时空复杂度和最优化原理。

- 建立模型，考虑可能出现的情况。
- 建立动态转移方程。
- 程序实现及代码优化。

钉子和小球（NOI'99）

一、问题描述

有一个三角形木板,竖直立放，上面钉着 n(n+1)/2 颗钉子，还有(n+1)个格子（当 n=5 时如图 1）。每颗钉子和周围的钉子的距离都等于 d，每个格子的宽度也都等于 d，且除了最左端和最右端的格子外每个格子都正对着最下面一排钉子的间隙。

让一个直径略小于 d 的小球中心正对着最上面的钉子在板上自由滚落，小球每碰到一个钉子都可能落向左边或右边（概率各 1/2），且球的中心还会正对着下一颗将要碰上的钉子。例如图 2 就是小球一条可能的路径。

我们知道小球落在第 i 个格子中的**概率** $p_i=C_n^i/2^n=\frac{n!}{i!(n-i)!}\Big/2^n$，其中 i 为格子的编号，从左至右依次为 0,1,...,n。

现在的问题是计算拔掉某些钉子后，小球落在编号为 m 的格子中的概率 p_m。假定最下面一排钉子不会被拔掉。例如图 3 是某些钉子被拔掉后小球一条可能的路径。

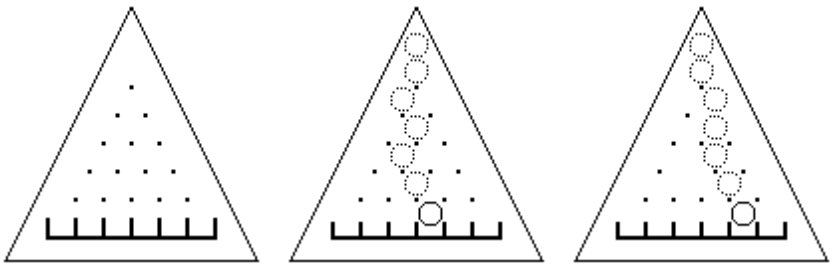


图 1　　*图 2*　　*图 3*

输入

第 1 行为整数 n（2<=n<=50）和 m（0<=m<=n）。

以下 n 行依次为木板上从上至下 n 行钉子的信息，每行中'*'表示钉子还在，'.'表示钉子被拔去，注意在这 n 行中空格符可能出现在任何位置。

输出

仅一行，是一个既约分数(0 写成 0/1)，为小球落在编号为 m 的格子中的概率 p_m

既约分数的定义：A/B 是既约分数，当且仅当 A、B 为正整数且 A 和 B 没有大于 1 的公因子。

样例输入

5 2

*

* .

＊ ＊ ＊
＊ ． ＊ ＊
＊ ＊ ＊ ＊ ＊
样例输出
7/16

二、分析

为了方便起见，可以将原问题看作把 2^n 个小球从顶端放入，每个小球到达底部的路径不同，求第 m 个格子中小球数与总数的比值。

设三角形有 n 行，第 i 行（1<=i<=n）有 i 个铁钉位置，其编号为 0..i-1；第 n+1 行有 n+1 个铁钉位置，排成 n+1 个格子，编号为 0..n。设经过位置（i,j）的小球个数为 $P_{i,j}$，则落入格子 m 的小球个数为 $P_{n+1,m}$，问题要求的是 $\frac{P_{n+1,m}}{2^n}$。

假设位置(i,j)有铁钉，则各有 $\frac{P_{i,j}}{2}$ 个小球落入位置（i+1,j）和位置(i+1,j+1)；否则 $P_{i,j}$ 个小球将全部落入位置(i+2,j+1)。

可得如下算法：

$P_{1,0} \leftarrow 2^n$;

for i←1 to n do
for j←1 to n do if 位置(i,j)有钉子 then

{ $P_{i+1,j} \leftarrow P_{i+1,j} + \frac{P_{i,j}}{2}$;

$P_{i+1,j+1} \leftarrow P_{i+1,j+1} + \frac{P_{i,j}}{2}$;

} else $P_{i+2,j+1} \leftarrow P_{i+2,j+1} + P_{i,j}$;

问题求的是既约分数，因为分母为 2 的次幂，因此可把分子、分母同时约去 2 的因子，直至分子不能整除 2。

三、参考程序

```
{$N+,R-,S-}
Program Noi99_Ball;
Const Fn='Ball.In';
      On='';
Var Tot        :Comp;
    N,M        :Integer;
    G          :Array[1..50,0..50] Of Char;
    P          :Array[1..51,0..50] of Comp;
```

```
    F,O         :Text;

Procedure Init;
Var I,J         :Integer;
    Ch          :Char;
Begin
    Assign(F,Fn); ReSet(F);
    ReadLn(f,n,m);
    For I:=1 To n Do Begin
        J:=0;
        Repeat
            Read(F,Ch);
            If Ch<>' ' Then Begin
                G[i,j]:=Ch; Inc(J);
            End;
        Until J=I;      {读入 i-1 个非空格的字符}
        ReadLn(F);
    End;
    Close(F);
End;

Procedure Out;
Var U,V         :Comp;
Begin
    If P[N+1,M]=0 Then Begin
        V:=0; U:=1;
    End Else Begin
        V:=P[N+1,M];
        U:=Tot;
        While (Frac(V/2)<0.1) Or(Frac(V/2)>0.9) Do Begin
            V:=V/2; U:=U/2;
        End;      {约分}
    End;
    Assign(O,On); ReWrite(O);
    WriteLn(O,V:0:0,'/',U:0:0);
    Close(O);
End;

Procedure Main;
Var I,J         :Integer;
Begin
    Tot:=1;
    For I:=1 To N Do Tot:=Tot*2;
    P[1,0]:=Tot;
    For I:=1 To N Do
```

```
            For J:=0 To I-1 Do
                If G[I,J]='*' Then Begin
                    P[I+1,J]:=P[I+1,J]+P[I,J]/2;
                    P[I+1,J+1]:=P[I+1,J+1]+P[I,J]/2;
                End Else
                    P[I+2,J+1]:=P[I+2,J+1]+P[I,J];
End;

Begin
      Init;
      Main;
      Out;
End.
```

Subset（NOI'99）

一、问题描述

众所周知，我们可以通过直角坐标系把平面上的任何一个点 P 用一个有序数对(x,y)来唯一表示，如果 x,y 都是整数，我们就把点 P 称为整点，否则点 P 称为非整点。我们把平面上所有整点构成的集合记为 W。

定义 1 两个整点 $P_1(x_1,y_1),P_2(x_2,y_2)$，若 $|x_1-x_2|+|y_1-y_2|=1$，则称 **P_1,P_2相邻**，记作 **$P_1\sim P_2$**，否则称 P_1,P_2 不相邻。

定义 2 设点集 S 是 W 的一个有限子集，即 $S=\{P_1,P_2,\ldots,P_n\}(n>=1)$，其中 $P_i(1<=i<=n)$属于 W，我们把 S 称为**整点集**。

定义 3 设 S 是一个整点集，若点 R,T 属于 S，且存在一个有限的点序列 $Q_1,Q_2,\ldots,Q_k$满足:

Q_i属于 S（$1<=i<=k$）；

$Q_1=R,Q_k=T$;

$Q_i\sim Q_{i+1}(1<=i<=k-1)$，即 Q_i与 Q_{i+1}相邻;

对于任何 $1<=i<j<=k$ 有 $Q_i\neq Q_j$;

我们则称点 R 与点 T 在整点集 S 上**连通**，把点序列 $Q_1,Q_2,\ldots,Q_k$称为整点集 S 中连接点 R 与点 T 的**一条道路**。

定义 4 若整点集 V 满足：对于 V 中的任何两个整点，V 中有且仅有一条连接这两点的道路，则 V 称为**单整点集**。

定义 5 对于平面上的每一个整点，我们可以赋予它一个整数，作为该点的权，于是我们把一个整点集中所有点的权的总和称为该**整点集的权和**。

我们希望对于给定的一个单整点集 V，求出一个 V 的最优连通子集 B，满足：

B 是 V 的子集

对于 B 中的任何两个整点，在 B 中连通；

B 是满足条件(1)和(2)的所有整点集中权和最大的。

二、分析

分析所给条件，建立数学模型。

试题的描述有一大篇，粗看很难抓住重点。于是我们逐条来看每个定义的具体意义。定义一讲述的是相邻的概念，在整点直角坐标系中，点通过相邻来同其他的点产生联系，且每一个点最多同它的四周的四个点相连，这是我们得到的一个性质。定义二规定整点集是有限个点的集合。定义三给连通下了定义。同

图的连通相似，两个点相通，也是通过一系列不重复的相邻的点的序列来连接。这启发我们是否能够用图来描述整个整点集之间的关系。定义四是说任意两个点之间有且仅有一条道路连接。这又同树的特点是相同。定义五基本就是为问题服务了，引出点权值的概念。对题目中的几个定义有了一定的了解之后，我们就感觉到这个所谓的单整点集，可以转化成我们熟悉的无环图。每个整点对应一个顶点，而两个相邻的顶点之间连上一条边。于是，我们得到了一个类似于无根树的图，且每个顶点的度数还不超过 4。对于这道题目的规模 N 不大于 1000，只需要一个两层循环就完成建图的过程。

算法如下：

```
For i := 1 to n do begin
  g[i, 0] := 0;
  For j := 1 to n do
    If abs(x[i] - x[j]) + abs(y[i] - y[j]) = 1 then begin
      Inc(g[i, 0]);
      g[i, g[i, 0]] := j;
    end;
end;
```

搜索图中权值和最大的子树。

再看看问题，要求一个属于给定的单整点集的一个权值和最大的子集。很容易想到搜索算法，从每一个点出发搜索一遍，每次搜索得到一棵树，其中包括所有的 N 个顶点。但是如果该树的某一子树的权值和为负数，那么它是不被记入总的权值和的（加入之后使得总权值和变晓）。通过递归过程 Search(Root, Direction)来实现，其中两个参数表示搜索的为以 Root 的第 Direction 个孩子为根的子树。把搜索得到的最大子树权值和保存在一个二维数组 F[Root, Direction]中。当然，若权值和小于 0，则 F[Root, Direction]为 0。

具体过程如下：

```
Procedure Search (Root, Direction: integer);
Begin
    Now := g[Root, direction]; {Now 即为当前所在的顶点}
    temp := value[Now];      {保存以 Now 为根的子树的最大权值和}
    for p := 1 to g[Now, 0] do
     if g[Now, p] <> Root then begin     {避免重复计算父亲节点的值}
     Search (Now, p);                    {搜索 Now 的第 P 个子树}
     if F[Now, p] > 0 then inc(temp, F[Now, p]);   {累加}
    end;
    if temp > 0 then F[Root, Direction] := temp
    else F[Root, Direction] := 0;            {赋值}
end;
```

减少重复计算，运用动态规划解决问题

通过对上面算法的分析，我们知道每次以不同的顶点作为根结点开始搜索，都要重新把图中的每一个点，以及以其为根的子树重新搜一遍。这样就产生了重叠的子问题。同时对于这道问题来说，每两个点之间只有一条道路。由此可知，如果我们来到了一个顶点，我们就不可能通过它的孩子节点再回到这个点的父辈或祖先顶点了。这表明此题具有无后效性。

于是我们找到实现动态规划的理论基础。对于重叠的子问题，我们就只需要计算一次，以后，只要直接调用结果。结合本题，就是以每个点为根结点的子树的最大权值。为了不重复的遍历顶点，同时要把它的父亲节点信息也要保留下来。在计算最大权值和时不把父亲节点的结果包含在内。

其实，上面的搜索算法，我之所以写得不象一般的搜索算法，而是用二维关系来确定一个顶点，为的就是容易在搜索的基础上改进成动态规划。具体实现为：

递归调用 Search 之前给 F[Root, Direction]数组赋初值-1。

递归调用 Search 时，若 F[Root, Direction] = -1 则继续。

若 F[Root, Direction]不为-1，则退出。

这一步只需在过程开始时判断一下就可以了。

主程序如下：

```
For i := 1 to n do
  For j := 1 to 4 do f[i, j] := -1;
ans := 0;
For i := 1 to n do begin
  s := value[i];
  For j := 1 to g[i, 0] do begin
    Search(i, j);
    If f[i, j] > 0 then inc(s, f[i, j]);
  End;
  If s > ans then ans := s;
End
Writeln(Ans);
```

三、小结

这道题说明了搜索通过一定的改造是可以变为动态规划。关键是找到两者之间的联系和区别。一般来说，搜索的不同对象可以转化为动态规划中的各个状态，对于同一个对象，我们可以通过查找已经计算出来的值来很快得到想要的答案。但是，这种记忆型的动态规划要满足无后效性。例如，本题如果只用当前所在点的编号来作为状态，当然搜索是能够做出来的，但是动态规划就不满足无后效性了。我们只有增加维数才能得到动态规划的做法。最后得出的空间复杂度为 $O(4*N)$,时间复杂度由于是递归实现，比较难以分析，大概是不到或是 $O(N^2)$。

四、参考程序

```
program _subset;
const
    name1 = 'input.txt';
    name2 = 'output.txt';
    maxn  = 1000;
var
    f       : array[1 .. maxn, 1 .. 4] of longint;
    g       : array[1 .. maxn, 0 .. 4] of integer;
    value   : array[1 .. maxn] of longint;
    n       : integer;
    ans     : longint;

procedure init;
var x, y    : array[1 .. maxn] of longint;
    i,j     : integer;
begin
    assign(input, name1);
    reset(input);
```

```
    readln(n);
    fillchar(x, sizeof(x), 0);
    fillchar(y, sizeof(y), 0);
    for i := 1 to n do readln(x[i], y[i], value[i]);
    for i := 1 to n do begin
      g[i, 0] := 0;
      for j := 1 to n do
        if abs(x[i] - x[j]) + abs(y[i] - y[j]) = 1 then begin
          inc(g[i, 0]);
          g[i, g[i, 0]] := j;
        end;
    end;
    close(input);
end;

procedure search( pre, direction : integer);
var now,p   : integer;
    temp    : longint;
begin
    if f[pre, direction] <> -1 then exit;

    now := g[pre, direction];
    temp := value[now];
    for p := 1 to g[now, 0] do
      if g[now, p] <> pre then begin
        search(now, p);
        if f[now, p] > 0 then inc(temp, f[now, p]);
      end;
    if temp > 0 then f[pre, direction] := temp
    else f[pre, direction] := 0;
end;

procedure main;
var i,j : integer;
    s   : longint;
begin
    for i := 1 to n do
      for j := 1 to 4 do f[i, j] := -1;
    ans := 0;
    for i := 1 to n do begin
      s := value[i];
      for j := 1 to g[i, 0] do begin
        search(i, j);
        if f[i, j] > 0 then inc(s, f[i, j]);
      end;
```

```
        if s > ans then ans := s;
      end;
  end;

  procedure print;
  begin
    assign(output, name2);
    rewrite(output);
    writeln(ans);
    close(output);
  end;

  begin
    init;
    main;
    print;
  end.
```

陨石的秘密（NOI'2001）

一、问题描述

公元 11380 年，一颗巨大的陨石坠落在南极。于是，灾难降临了，地球上出现了一系列反常的现象。当人们焦急万分的时候，一支中国科学家组成的南极考察队赶到了出事地点。经过一番侦察，科学家们发现陨石上刻有若干行密文，每一行都包含 5 个整数：

1 1 1 1 6
0 0 6 3 57
8 0 11 3 2845

著名的科学家 SS 发现，这些密文实际上是一种复杂运算的结果。为了便于大家理解这种运算，他定义了一种 SS 表达式：

1. SS 表达式是仅由‘{’，‘}’，‘[’，‘]’，‘(’，‘)’组成的字符串。
2. 一个空串是 SS 表达式。
3. 如果 A 是 SS 表达式，且 A 中不含字符‘{’，‘}’，‘[’，‘]’，则(A)是 SS 表达式。
4. 如果 A 是 SS 表达式，且 A 中不含字符‘{’，‘}’，则[A]是 SS 表达式。
5. 如果 A 是 SS 表达式，则{A}是 SS 表达式。
6. 如果 A 和 B 都是 SS 表达式，则 AB 也是 SS 表达式。

例如

()(())[]
{()[()]}
{{[[(())]]}}

都是 SS 表达式。

而

()([])()
[()

不是 SS 表达式。

一个 SS 表达式 E 的深度 D(E)定义如下：

$$D(E)=\begin{cases}0, & \text{如果}E\text{是空串}\\ D(A)+1, & \text{如果}E=(A)\text{或者}E=[A]\text{或者}E=\{A\},\text{其中}A\text{是}SS\text{表达式}\\ \max(D(A),D(B)), & \text{如果}E=AB,\text{其中A,B是}SS\text{表达式。}\end{cases}$$

例如()｛()｝[]的深度为 2。

密文中的复杂运算是这样进行的：

设密文中每行前 4 个数依次为 L1，L2，L3，D，求出所有深度为 D，含有 L1 对{}，L2 对[]，L3 对()的 SS 串的个数，并用这个数对当前的年份 11380 求余数，这个余数就是密文中每行的第 5 个数，我们称之为“神秘数”。

密文中某些行的第五个数已经模糊不清，而这些数字正是揭开陨石秘密的钥匙。现在科学家们聘请你来计算这个神秘数。

输入文件（secret.in）

共一行，4 个整数 L1，L2，L3，D。相邻两个数之间用一个空格分隔。

（0≤L1≤10，0≤L2≤10，0≤L3≤10，0≤D≤30）

输出文件（secret.out）

共一行，包含一个整数，即神秘数。

输入样例

1 1 1 2

输出样例

8

二、分析解答

这是一个典型的计数问题。

动态规划的一个重要应用就是组合计数—如鱼得水，具有编程简单、时空复杂度低等优点。我们自然想到：是否本题也可以用动态规划来解决呢？

条件的简化

题目对于什么是 SS 表达式做了大量的定义，一系列的条件让我们如坠雾中。为了看清 SS 表达式的本质，有必要对条件进行简化。

条件 1 描述了 SS 表达式的元素。

条件 3、4、5 实际上对于()、[]、{}的嵌套顺序做了限制，即()内不能嵌套[]、{}，[]内不能潜逃{}。概括起来是两点：SS 表达式中括号要配对；{}、[]、() 从外到内依次嵌套。

状态的表示

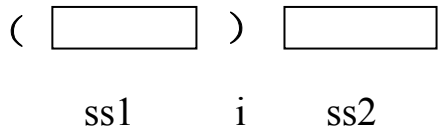
这是动态规划过程中首先要解决的一个问题。本题的条件看似错综复杂，状态不易提炼出来，实际上，题目本身已经为我们提供了一个很好的状态表示法。

对于一个表达式来说，它含有的元素是“(”，“)”，“[”，“]”，“{”，“}”，此外，定义了深度这一概念。最简单的一种想法是：按照题目的所求，直接把{}的对数 11、[]的对数 12、()的对数 13 以及深度 d 作为状态表示的组成部分，即用(11,12,13,d)这样一个四元组来确定一个状态。令 F(11,12,13,d)表示这样一个状态所对应的神秘数，于是 F(L1,L2,L3,D)对应问题答案。此外，我们令 G(11,12,13,d)表示含有 11 个{}，12 个[]，13 个()，深度不大于 d 的表达式个数。显然，F(11,12,13,d)=G(11,12,13,d)-G(11,12,13,d-1)。于是求解 F 的问题，可以转化为求解 G 的问题。

状态转移方程的建立

设当前的状态为（11,12,13,d），根据表达式的第一位的值，分如下三种情况：

情况一：第一位是“(”，与其配对的“)”位于第 i 位。设$G_1(l1,l2,l3,d)$表示这种情况下的总数，G_2、G_3类似定义。



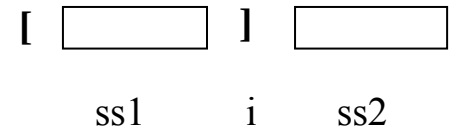
()将整个表达式分成两部分(图中的 ss1 和 ss2)。根据乘法原理，我们只需对两部分分别计数，然后乘起来即为结果。

我们设 ss1 中含有 x 对{}，y 对[]，z 对()。因为 ss1 外层已经由一对()括起来，故其内部不可再含[]、{}，因此 x=0，y=0，且 ss1 的深度不可超过 d-1，ss1 的数目为 G(x,y,z,d-1)=G(0,0,z,d-1)。ss2 中含有 l1-x=l1 个{}，l2-y=l2 个[]，l3-z-1 个()，深度不可超过 d，ss2 的数目为 G(l1,l2,l3-z-1,d)。据此，我们写出下面这个式子：

$$G_1(l1,l2,l3,d)=\sum_{z=0}^{l3-1}G(0,0,z,d-1)*G(l1,l2,l3-z-1,d)$$

情况一计算的复杂度为 O(n^5)。

情况二：第一位是“[”，与其配对的“]”位于第 i 位。

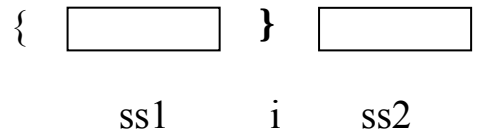


与情况一类似可得出

$$G_2(l1,l2,l3,d)=\sum_{\substack{y<=l2-1,\\ z<=l3}}G(0,y,z,d-1)*G(l1,l2-y-1,l3-z,d)$$

计算复杂度为 O(n^6)。

情况三：第一位是“{”，与其配对的“}”位于第 i 位。



有如下式子：

$$G_3(l1,l2,l3,d)=\sum_{\substack{x<=l1-1\\ y<=l2,\\ z<=l3}}G(x,y,z,d-1)*G(l1-x-1,l2-y,l3-z,d)$$

这一部复杂度为 O(n^7)。

综合上述三种情况：

$$G(l1,l2,l3,d)=G_1(l1,l2,l3,d)+G_2(l1,l2,l3,d)+G_3(l1,l2,l3,d)$$

三、小结

本题的时间复杂度为O(n^7)，在规定时间内完全可以出解。空间上，若采用滚动数组的方式，可将空间复杂度为 n^3，保存也不成问题。本题的难点在于动态规划时情况的划分及处理，当需要建立复杂的状态转移方程时，我们也要保持冷静、抓住要害。

四、参考程序

```
program Secret;

  const
    finp        = 'secret.in';
    fout        = 'secret.out';
    year        = 11380;

  var
    f           : array[0 .. 11 , 0 .. 11 , 0 .. 11 , -1 .. 31] of integer;
    a1 , a2 , a3 , dep : integer;

  procedure calc;
    var
      s         : integer;
      d , l1 , l2 , l3 ,
      x , y , z : integer;
    begin
      fillword(f , sizeof(f) shr 1 , 0);
      for d := 0 to dep do
        f[0,0,0,d] := 1;
      for d := 1 to dep do
        for l1 := 0 to a1 do
          for l2 := 0 to a2 do
            for l3 := 0 to a3 do
             if (l1 > 0) or (l2 > 0) or (l3 > 0) then begin
              s := 0;
              for x := 0 to l1 - 1 do
                for y := 0 to l2 do
                  for z := 0 to l3 do
                    s := (s + f[x,y,z,d-1] * f[l1-x-1,l2-y,l3-z,d]) mod year;
              for y := 0 to l2 - 1 do
                for z := 0 to l3 do
                  s := (s + f[0,y,z,d-1] * f[l1,l2-y-1,l3-z,d]) mod year;
              for z := 0 to l3 - 1 do
                s := (s + f[0,0,z,d-1] * f[l1,l2,l3-z-1,d]) mod year;
              f[l1,l2,l3,d] := s;
```

```
            end;
    end;

  procedure init;
    begin
      assign(input , finp);
      reset(input);
      readln(a1 , a2 , a3 , dep);
      close(input);
    end;

  procedure print;
    var
      left : longint;
    begin
      assign(output , fout);
      rewrite(output);
      left := f[a1,a2,a3,dep] - f[a1,a2,a3,dep-1];
      if left < 0 then left := left + year;
      writeln(left);
      close(output);
    end;

  begin
    init;
    calc;
    print;
    end.
```

商店购物（IOI'95）

一、问题描述

某商店中每种商品都有一个价格。例如，一朵花的价格是 2 ICU(ICU 是信息学竞赛的货币的单位）；一个花瓶的价格是 5 ICU。为了吸引更多的顾客，商店提供了特殊优惠价。

特殊优惠商品是把一种或几种商品分成一组。并降价销售。例如:3 朵花的价格不是 6 而是 5 ICU ;2 个花瓶加 1 朵花是 10 ICU 不是 12 ICU。

编一个程序，计算某个顾客所购商品应付的费用。要充分利用优惠价以使顾客付款最小。请注意，你不能变更顾客所购商品的种类及数量，即使增加某些商品会使付款总数减小也不允许你作出任何变更。假定各种商品价格用优惠价如上所述，并且某顾客购买物品为:3 朵花和 2 个花瓶。那么顾客应付款为 14 ICU 因为:

1 朵花加 2 个花瓶: 优惠价:10 ICU
2 朵花　　正常价: 4 ICU

输入数据

用两个文件表示输入数据。第一个文件 INPUT. TXT 描述顾客所购物品（放在购物筐中）;第二个文件描述商店提供的优惠商品及价格（文件名为 OFFER. TXT)。 两个文件中都只用整数。

第一个文件 INPUT．TXT 的格式为:第一行是一个数字 B（0≤B≤5），表示所购商品种类数。下面共 B 行，每行中含 3 个数 C，K，P。C 代表商品的编码（每种商品有一个唯一的编码），1≤C≤999。K 代表该种商品购买总数，1≤K≤5。P 是该种商品的正常单价（每件商品的价格），1≤P≤999。请注意，购物筐中最多可放 5*5＝25 件商品。

第二个文件 OFFER．TXT 的格式为:第一行是一个数字 S（0≤S≤99），表示共有 S 种优惠。下面共 S 行，每一行描述一种优惠商品的组合中商品的种类。下面接着是几个数字对（C，K），其中 C 代表商品编码，1≤C≤9 99。K 代表该种商品在此组合中的数量，1≤K≤5。本行最后一个数字 P（1≤ P≤9999）代表此商品组合的优惠价。当然， 优惠价要低于该组合中商品正常价之总和。

输出数据

在输出文件 OUTPUT．TXT 中写 一个数字（占一行）， 该数字表示顾客所购商品（输入文件指明所购商品）应付的最低货款。

二、分析

初看这道题目，我的感觉是似曾相识,同我们做的背包问题差不多。只是背包问题是给定容量，求最大价值的东西。而这道题目是给定所放的东西，求最小的费用（对应背包问题为最小的容量）。恰好是一个求最值的“逆问题”。背包问题是经典的动态规划问题，那么这道题呢？

由于动态规划要满足无后效性和最优化原理，所以我们来分析此题是否满足以上两点。先来状态表示的方法，商品不超过 5 种，而每种采购的数量又不超过 5，那么用一个五元组来表示第 I 种商品买 A_I 的最小费用。:

F（A_1，A_2，A_3，A_4，A_5）　　（1）

考虑这个状态的由来，当然，我们不用优惠商品也可以买，显然这样不是最优。于是如果我们能够使用第 I 条商品组合的话，状态就便为了：

F（A_1-S_{I1}，A_2-S_{I2}，A_3-S_{I3}，A_4-S_{I4}，A_5-S_{I5}）　（2）

这样的话，状态 1 的费用为状态 2 的费用加上 S_I 的费用，而状态 2 的费用必须最低（很显然，用反证法即可），同时，我们也不管状态 2 是如何来的（因为每一个优惠商品组合的使用是没有限制的），所以本题既满足无后效性，又符合最优化原理，同时还有大量重叠子问题产生，动态规划解决此题是最好不过了。

通过对问题的分析，我们知道了状态的表示和转移的基本方法，我们很容易得到一个状态转移方程：

F [a, b, c, d, e] = Min {F [a-S_1, b-S_2, c-S_3, d-S_4, e-S_5] + SaleCost [S]}

初始条件为：

F [a, b, c, d, e] = Cost [1]*a+Cost [2]*b+Cost [3]*c+Cost [4]*d+Cost [5]*e

即不用优惠的购买费用。

三、小结

这道题还是相对较简单的，毕竟事过境迁，这已经是七八年前的题目了。时间复杂度也可以接受，为 O（6^5*99）≈10^6，但常数项的影响因为商品总数小而比较突出。空间复杂度为 O（6^5），根本不是问题。具体实现时，由于输入的数据有一部分是没有意义的，比如商品中包含不需要的物品，我们可以在输入时剔除掉，以提高程序的效率。对于数据中，商品总数不足 5 种的，可以把不买的那几种看成是购买 0 件，以统一操作。

四、参考程序

```
program _shop;
const
      name1 = 'input.txt';
      name2 = 'output.txt';
```

```
    name3 = 'offer.txt';
type
    saletype = array[1 .. 5] of byte;
var
  f                 : array[0 .. 5, 0 .. 5, 0 .. 5, 0 .. 5, 0 .. 5] of word;
  costs             : array[0 .. 5] of integer;
  code              : array[1 .. 999] of byte;
  sale              : array[1 .. 99] of saletype;
  paysale           : array[1 .. 99] of integer;
  check             : array[1 .. 99] of boolean;
  st,ed             : array[0 .. 5] of byte;
  s,b               : integer;

procedure init;
var i,cc,kk,pp,j            : word;
begin
    assign(input, name1);       reset(input);
    fillchar(f, sizeof(f), 0);
    fillchar(code, sizeof(code), 0);
    fillchar(sale, sizeof(sale), 0);
    fillchar(costs, sizeof(costs), 0);
    fillchar(st, sizeof(st), 0);
    fillchar(ed, sizeof(ed), 0);
    fillchar(check, sizeof(check), true);     {初始化}
    readln(b);
    for i := 1 to b do begin
      readln(cc, kk, pp);
      code[cc] := i;
      ed[i] := kk;                            {ed[i]表示第 I 种商品购买的数量}
      costs[i] := pp;
    end;
    close(input);
    assign(input, name3);     reset(input);
    readln(s);
    for i := 1 to s do begin
      read(cc);
      for j := 1 to cc do begin
        read(kk, pp);
        if code[kk] = 0 then begin check[i] := false; break; end;
            {显然不行的组合就不用计算了}
        sale[i, code[kk]] := pp;
      end;
      readln(cc);
      paysale[i] := cc;
    end;
```

```
    close(input);
end;

procedure main;
var j,k   : integer;
    i     : array[1 .. 5] of byte;
    q,t   : word;
    can   : boolean;
begin
    for i[1] := st[1] to ed[1] do
      for i[2] := st[2] to ed[2] do
        for i[3] := st[3] to ed[3] do
          for i[4] := st[4] to ed[4] do
            for i[5] := st[5] to ed[5] do begin          {枚举每个状态}
              q := 0;
              for j := 1 to 5 do inc(q, costs[j] * i[j]);   {初始值为不用任何优惠}
              for j := 1 to s do                          {枚举每个优惠商品组合}
                if check[j] then begin
                  can := true;
                  for k := 1 to 5 do
                    if i[k] < sale[j, k] then can := false; {是否适用当前组合}
                  if can then begin
                    t := paysale[j] +
                         f[
                           i[1] - sale[j, 1], i[2] - sale[j, 2],
                           i[3] - sale[j, 3], i[4] - sale[j, 4],
                           i[5] - sale[j, 5]
                         ];
                    if t < q then q := t;                 {如果更优则更新}
                  end;
                end;
              f[i[1], i[2], i[3], i[4], i[5]] := q;       {赋值}
            end;
end;

procedure print;   {输出}
begin
    assign(output, name2);
    rewrite(output);
    writeln(f[ed[1], ed[2], ed[3], ed[4], ed[5]]);
    close(output);
end;

begin
    init;
```

```
main;
print;
end.
```

添括号问题（NOI'96）

一、试题

有一个由数字 1，2，... ，9 组成的数字串（长度不超过 200），问如何将 M(M<=20)个加号("+")插入到这个数字串中，使所形成的算术表达式的值最小。请编一个程序解决这个问题。

注意：

加号不能加在数字串的最前面或最末尾，也不应有两个或两个以上的加号相邻。

M 保证小于数字串的长度。

例如：数字串 79846，若需要加入两个加号，则最佳方案为 79+8+46，算术表达式的值 133。

[输入格式]

从键盘读入输入文件名。数字串在输入文件的第一行行首（数字串中间无空格且不折行），M的值在输入文件的第二行行首。

[输出格式]

在屏幕上输出所求得的最小和的精确值。

输入示例

82363983742

3

输出示例

2170

二、分析

这道题目是经典的动态规划的题目.因此肯定采用动态规划.

考虑到数据的规模超过了长整型,我们注意在解题过程中采用高精度算法.

规划方程:F[I,J] = MIN { F[I-1,K] + NUM[K+1,J] } (I-1<=K<=J-1)

边界值:F[0,I] := NUM[1,I]　;

F[I，J]表示前 J 个数字中添上 I 个加号后得到的最小值。

NUM[I，J]表示数字串第 I 位到第 J 位的数

程序需要的空间约为 20 * 200 * 200.显然难以承受。

但是，还能更优。每一步，我们都只与上一步有关。因此可以采用滚动数组，这样，复杂度就降到了 2 * 200 * 200 左右了。

程序的时间效率约为 20 * 200 * 200.时间上根本不成问题。

这道题目看起来很容易,但是如果在编程时不多加细心，容易的问题也难免会有这样那样的错误。因此,越是简单的题目越要细心，不能粗枝大叶。

三、参考程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+,Y+}
{$M 16384,0,655360}
```

```
program Noi96_4;
  const
    filein      ='input.txt';
    fileou      ='output.txt';
  type
    stype       =string[202];
  var
    f1 , f2     :array[1..200] of ^stype;{ 滚动数组 }
    st          :string;{数字串}
    fi,  fo     :text;
    n , m       :integer;{字串长度和加号数目}
  procedure init;
    var
      i , j , k : integer;
    begin
      assign(fi,filein);
      assign(fo,fileou);
      reset(fi);
      rewrite(fo);
      readln(fi,st);
      n := length(st);
      readln(fi,m);
      close(fi);
      for i := 1 to 200 do
      begin new(f1[i]);fillchar(f1[i]^,sizeof(f1[i]^),'0');end;
      for i := 1 to 200 do
      begin new(f2[i]);fillchar(f1[i]^,sizeof(f1[i]^),'0');end;
      for i := 1 to n do
        f1[i]^ := copy(st,1,i);
    end;
  function big(st1,st2:string):boolean;{判断两个字符串的大小}
    begin
      big := ( length(st1)>length(st2) ) or
             ( length(st1)=length(st2) ) and ( st1 > st2 );
    end;
  procedure sum(st1,st2:string;var st3:string);{高精度加法}
    var
      i , j , k  , l , o , p : integer;
    begin
      st3 := '';
      while length(st1)<length(st2) do insert('0',st1,1);
      while length(st2)<length(st1) do insert('0',st2,1);
      p := 0;
      for o := length(st1) downto 1 do
        begin
```

```
          val(st1[o],i,l);
          val(st2[o],j,l);
          k := (i+j+p) mod 10;
          p := (i+j+p) div 10;
          st3 := chr(k+48) + st3;
        end;
      if p>0 then st3 := chr(p+48) + st3;
      while (st3[1]='0') and (length(st3)>1) do delete(st3,1,1);
    end;
  procedure dynamic;{动态规划}
    var
     i  , j , k , l , o , p  : integer;
     st1 , st2 , st3         : string;
    begin
     for i := 1 to m do
      begin
       for j := 1 to 200 do
       fillchar(f2[j]^,sizeof(f2[j]^),'0');
       for j := i+1 to n-m+i do
         for k := i to j-1 do
           if (k >= i) and ( k < j ) and  (j-k<=101) then
             begin
               sum(f1[k]^,copy(st,k+1,j-k),st3);
               if big(f2[j]^,st3) or (f2[j]^='0') then
                 f2[j]^ := st3;
             end;
       for j := 1 to 200 do
       f1[j]^ := f2[j]^;
      end;
    end;
  begin
    init;
    dynamic;
    writeln(fo,f1[n]^);
    close(fo);
  end.
```

最长前缀（IOI'96）

一、问题描述

一些生物体的复杂结构可以用其基元的序列表示，而一个基元用一个大写英文字符串表示。生物学家的一个问题就是一个这样的长序列分解为基元（字符串）的序列。对于给定的基元集合 P，如果可以从中选出 N 个基元 P1，P2，P3，...,Pn，将它们各自对应的字符串依次连接后得到一个字符串 S，称 S 可以由基元集合 P 构成。在从 P 中挑选基元时，一个基元可以使用多次，也可不用。例如，序列 ABABACABAAB 可以由基元集合{A，AB，BA，CA，BBC} 构成。

字符串的前 K 个字符为该字符串的前缀，其长度为 K。请写一个程序，对于输入的基元集合 P 和字符串 T，求出一个可以由基元集合 P 构成的字符串 T 的前缀，要求该前缀的长度尽可能长，输出其长度。

输入数据：有两个输入文件 INPUT.TXT，DATA.TXT

INPUT.TXT 的第一行是基元集合 P 中的基元数目 N（1<=N<=100），随后有 2N 行，每两行描述一个基元，第一行为该基元的长度 L（1<=L<=20）。随后一行是一个长度为 L 的大写英文字符串，表示该基元。每个基元互不相同。

DATA.TXT 描述要处理的字符串 T，每一行行首有一个大写字母，最后一行是一个字符'.'，表示字符串结束。T 的长度最小为 1，最大不超过 500000。

输出数据：OUTPUT.TXT。只有一行，一个数字，表示可以由 P 构成的 T 的最长前缀的长度。

示例：

INPUT.TXT	DATA.TXT	OUTPUT.TXT
5	A	11
1	B	
A	A	
2	B	
AB	A	
3	C	
BBC	A	
2	B	
CA	A	
2	A	
BA	B	
C	B	

二、分析

本题可以简述为：从一个集合里选出若干个基元，使其组成字符串 T 的前缀，求最长前缀的长度.

对于 T 的每个字符，其状态可分为两种：

在此之前的所有字符（包括本身）可匹配(true)、不可匹配(false)。(可匹配是指可以由集合里的基元组成)

用 F_i 表示第 i 个字符的状态，$find_{a,b}$ 表示由第 a 至 b 位的字符组成的子串是否存在于集合中，则：

$F_i = F_i$ or（F_k and $find_{k+1,i}$） (k=0..i-1)

初始条件：

$$F_i = \begin{cases} \text{true} & (i=0) \\ \text{false} & (i \neq 0) \end{cases}$$

由于 T 的长度最大达 500000，无法存放所有状态，但集合里基元长度不超过 20，因此可只保留当前 20 位字符与状态。当 20 位字符都不可以匹配时，停止运算，最后一个状态为 true 的字符的位置，即为所求。

为了便于操作，可用字符串表示状态，‘0’表 false、‘1’表 true.

为了便于查找，可将基元按长度存储。

形如：s[i,j]，表长度为 i 的第 j 个基元。

亦可采用树的结构存储基元，构造一种多叉树（最多 26 叉），查找时顺着相应字母，定位到相应分支。

这样速度要快些，但程序更复杂。大家可以比较一下。

按树结构算，时间复杂度为 0(500000*L*L)，勉强可以承受。

三、小结

本题中，当前状态的确定只与前面的状态有关，可找出递推式。

充分利用基元长度不超过 20 这一条件。

四、参考程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+}
{$M 16384,0,655360}
program ioi96_5;
const fn = 'input.txt';
      dn = 'data.txt';
type tree = ^node;
     node = record
            a          :boolean;
            l          :array['A'..'Z'] of tree;
     end;
var
    root          :tree;
    n,tot,l       :longint;
    f,o           :text;
    tm            :real;

procedure init;
var i,len         :integer;
    str           :string[20];
    ch            :char;
  procedure get_tree (p :byte; var x :tree);
  var i                       :char;
  begin
       if p=len+1 then begin
          x^.a:=true;   exit;
       end;
       if x^.l[str[p]]=nil then begin
          new(x^.l[str[p]]);
          x^.l[str[p]]^.a:=false;
          for i:='A' to 'Z' do
              x^.l[str[p]]^.l[i]:=nil;
       end;
       get_tree(p+1,x^.l[str[p]]);
  end;

begin
     tm:=meml[$40:$6c];
```

```
        assign(f,fn);
        reset(f);
        readln(f,n);
        new(root);
        for ch:='A' to 'Z' do root^.l[ch]:=nil;
        for i:=1 to n do begin
              readln(f,len);
              if len>l then l:=len;
              readln(f,str);
              get_tree(1,root);
        end;
        close(f);
end;

function find(str :string) :boolean;
var i                    :integer;
      x                    :tree;
begin
        x:=root;
        find:=false;
        for i:=1 to length(str) do begin
              if x^.l[str[i]]=nil then exit;
              x:=x^.l[str[i]];
        end;
        if x^.a then find:=true;
end;

procedure out;
begin
        assign(o,on);
        rewrite(o);
        writeln(o,tot);
        close(o);
        writeln((meml[$40:$6c]-tm)/18.2:0:3);
        halt;
end;

procedure main;
var st1,st2,st3 :string;
      ch               :char;
      i                :integer;
begin
        assign(f,dn);
        reset(f);
        readln(f,ch);
```

```
    st2:='#'; st3:='1';
    n:=0;
    while ch<>'.' do begin
          inc(n);      st1:='';
          st2:=st2+ch;
          for i:=length(st2) downto length(st2)-l+1 do begin
              st1:=st2[i]+st1;
              if (find(st1))and(st3[i-1]='1') then begin
                  st3:=st3+'1';
                  tot:=n;
                  break;
              end;
          end;
          if length(st2)>length(st3) then st3:=st3+'0';
          if length(st2)>l then begin
              delete(st2,1,1);
              delete(st3,1,1);
          end;
          if n-tot>=l then out;
          readln(f,ch);
    end;
    close(f); out;
end;

begin
init;
main
end.
```

多边形（IOI’98）

一、问题描述

多角形是一个单人玩的游戏，开始时有一个 N 个顶点的多边形。如图 1，这里 **N**=4。每个顶点有一个整数标记，每条边上有一个“+”号或“*”号。边从 1 编号到 N

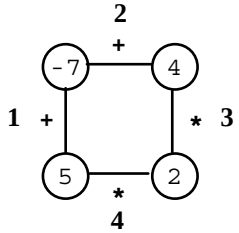


Figure 1.一个多边形的图示

第一步，一条边被拿走；随后各步包括如下：

- 选择一条边 E 和连接着 E 的两个顶点 V_1 和 V_2；
- 得到一个新的顶点，标记为 V1 与 V2 通过边 E 上的运算符运算的结果。

最后，游戏中没有边，游戏的得分为仅剩余的一个顶点的值。

游戏样例

如图 1 所示的多边形。游戏者开始时拿走第 3 边。结果如图 2 所示。

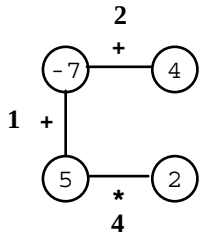


Figure 2. 拿走第 3 边

之后，他拿走边 1

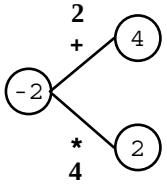


Figure 3. 拿走边 1

然后是边 **4**,

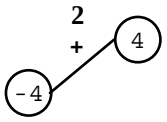


Figure 4. 拿走边 4

最后是边 2，得分为 0。

Figure 5. 拿走边 2

任务

写一个程序，对于给定一个多边形，计算出可能的最高得分，并且列出得到这个分数的过程。

输入数据

文件 POLYGON.IN 描述一个 N 个顶点的多边形。其中包含两行。第一行为整数 N。第二行包含边 1 到边 N 的标记符号，之间插入点的标记值。(边 1 与边 2 之间为顶点 1，边 2 与边 3 之间为顶点 2，以此类推。边 N 与边 1 之间为顶点 N）所示数据之间有一个空格。一条边的标记为字母“t”（代表+)或者字母“x”(代表*)。

样例输入：

```
4
t -7 t 4 x 2 x 5
```

如图 1 所示的多边形的输入文件。

输出数据

在文件 POLYGON.OUT 的第一行是最高分，第二行列出得到这个分数的所有可能中第一步拿走的边的序

号。边的序号必须是递增的，且

之间有一个空格隔开。d by one space.

Sample Output:

```
33
1 2
```

以上是图 1 所示的例子中的结果。

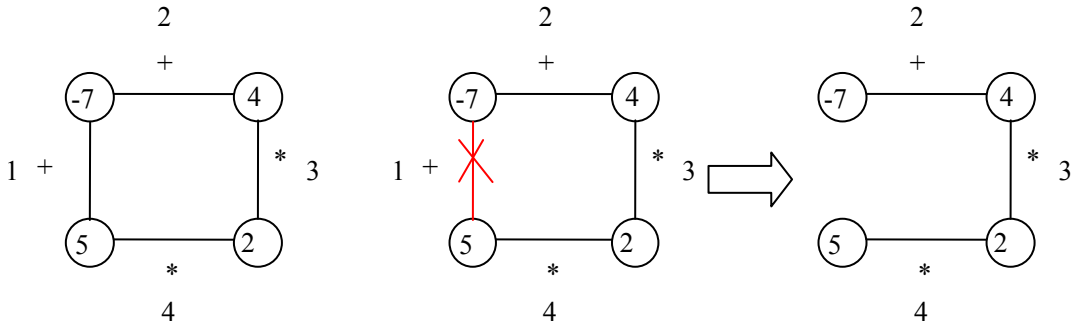
限制：

3 ≤ N ≤ 50

顶点的标记值在范围[-32768,32767]内。

二、分析

Polygon 是一个由 n 个（n<=50）顶点构成的凸多边形。多边形相邻的点之间有一条边，边上为一个操作符，顶点上则为一个数，见图一。现在我们可以任意删除多边形中的一条边。比如我们删除图一多边形中的 1 边（图二，图三）。



图一　　图二　　图三

将边删除后，多边形变成了一条“线”（图四）。

图四　　图五

我们在这条“线”当中继续删边，并且每次删边都使被删边两旁的点按边上的操作符合并，图五。这样进行了 n-1 次删边操作后，“线” 变成了一个点。我们的目的，就是安排删边的顺序，使最后的点上的数尽可能的大。

拿到题目之后，我们马上可以想到用枚举法——枚举删边的先后顺序。但边数最大可以达到 50，枚举的复杂将会有 50!。因此枚举算法马上被排除了。

对最优化问题的求解，我们往往可以使用动态规划来解决。这道题是不是可以使用动态规划呢？

我们先对题目进行一些变化——原题中顶点上的数可以为负数，现在我们规定这个数一定大于等于 0；原题中边可以为乘号，现在我们规定只能为加号。

题意改变后，我们想到了什么？对！“石子合并”。

我们先枚举第一次删掉的边，然后再对每种状态进行动态规划求最大值：

用 f(I,j)表示从 j 开始，进行 i 次删边操作所能得到的最大值，num(i)表示第 i 个顶点上的数，那么：

$$f(i,j)=\max\sum_{k=1}^{i=1}(f(k,i)+f(j-k,i+k))$$

$$f(1,i)=num(i)$$

现在我们来考虑加入乘号后的情况。

由于所有的顶点上的数都为非负数，因此即使有了乘法，函数 f 的无后效性并不被破坏。我们可以在前一方程的基础上进行改进：（opr(i)表示第 i 条边上的操作符）

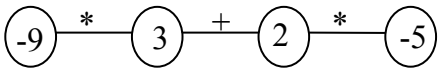
$$Act(x1,y1,x2,y2)=\begin{cases}\text{当 opr(x2-1)= +时}\ \ f(x1,y1)+f(x2,y2)\\ \text{当 opr(x2-1)= -时}\ \ f(x1,y1)*f(x2,y2)\end{cases}$$

$$f(i,j)=\max\sum_{k=1}^{i-1}act(k,i,j-k,i+k)$$

$$f(1,i)=num(i)$$

最后，我们允许顶点上出现负数。以前的方程还是不适用呢？

我们来看一个例子（图六）。



图六

这个例子的最优解应该是（3+2）*（-9）*（-5）=250，然而如果沿用以前的方程，得出的解将是（（-10）*3+2）*（-5）=140。为什么？

我们发现，两个负数的积为正数；这两个负数越小，它们的积越大。我们从前的方程，只是尽量使得局部解最大，而从来没有想过负数的积为正数这个问题。

我们引入函数 fmin 和 fmax 来解决这个问题。fmax(I,j) 表示从 j 开始，进行 i 次删边操作所能得到的最大值，fmin(I,j) 表示从 j 开始，进行 i 次删边操作所能得到的最小值。

$$f\ \max(i,j)=\max\sum_{k=1}^{i=1}act\max(k,i,j-k,i+k)$$

$$f\ \min(i,j)=\min\sum_{k=1}^{i-1}act\min(k,i,j-k,i+k)$$

$$f(1,i)=num(i)$$

函数 actmax 与 actmin 的构造是十分关键的。

首先讨论 actmax(x1,y1,x2,y2)的构造：

当 opr(x2-1)=+时，毫无疑问，actmax=fmax(x1,y1)+fmax(x2,y2)

当 opr(x2-1)=*时，

actmax=max(fmax(x1,y1)*fmax(x2,y2),fmin(x1,y1)*fmin(x2,y2))

接下来讨论 actmin(x1,y1,x2,y2)的构造：

当 opr(x2-1)=+时，actmin=fmin(x1,y1)+fmin(x2,y2)

当 opr(x2-1)=*时，

actmin=min(fmax(x1,y1)*fmin(x2,y2),fmin(x1,y1)*fmax(x2,y2))

到此为止，整个问题圆满解决了。算法的空间复杂度为 n^2，算法时间复杂度为 n^4（先要枚举每一条边，然后再用复杂度为 n^3 的动态规划解决），对于竞赛给出的测试数据，全部一秒内出解。

三、小结

我们采用类比思想，先将问题简单化，与曾经做过的“石子合并”进行比较，从而一步步推出本题的解法。这种思想，竞赛中是经常用到的。

花店橱窗布置（IOI’99）

一、问题描述

假设你想以最美观的方式布置花店的橱窗。你有 F 束花，每束花的品种都不一样，同时，你至少有同

样数量的花瓶，被按顺序摆成一行。花瓶的位置是固定的，并从左至右，从 1 至 V 顺序编号，V 是花瓶的数目，编号为 1 的花瓶在最左边，编号为 V 的花瓶在最右边。花束则可以移动，并且每束花用 1 至 F 的整数唯一标识。标识花束的整数决定了花束在花瓶中排列的顺序，即如果 I＜j，则花束 I 必须放在花束 j 左边的花瓶中。

例如，假设杜鹃花的标识数为 1，秋海棠的标识数为 2，康乃馨的标识数为 3，所有的花束在放入花瓶时必须保持其标识数的顺序，即：杜鹃花必须放在秋海棠左边的花瓶中，秋海棠必须入在康乃馨左边的花瓶中，如果花瓶的数目大于花束的数目，则多余的花瓶必须空置，每个花瓶中只能放一束花。

每一个花瓶的形状和颜色也不相同。因此，当各个花瓶中放入不同的花束时，会产生不同的美学效果，并以美学值（一个整数）来表示，空置花瓶的美学值为零。在上述例子中，花瓶与花束的不同搭配所具有的美学值，可以用下面式样的表格来表示。

		花瓶				
		1	2	3	4	5
花束	1、杜鹃花	7	23	-5	-24	16
	2、秋海棠	5	21	-4	10	23
	3、康乃馨	-21	5	-4	-20	20

例如，根据上表，杜鹃花放在花瓶 2 中，会显得非常好看；但若放在花瓶 4 中则显得很难看。

为取得最佳美学效果，你必须在保持花束顺序的前提下，使花束的摆放取得最大的美学值。如果具有最大美学值的摆放方式不止一种，则其中任何一种摆放方式都可以接受，但你只右输出其中一种摆放方式。

假设条件（Asumption）

1≤F≤100，其中 F 为花束的数量，花束编号从 1 至 F。

F≤V≤100，其中 V 是花瓶的数量。

-50≤Aij≤50，其中 Aij 是花束 i 在花瓶 j 中时的美学值。

输入（Input）

输入文件是正文文件（text file），文件名是 flower.inp。

第一行包含两个数：F、V。

随后的 F 行中，每行包含 V 个整数，Aij 即为输入文件中第（i+1）行中的第 j 个数。

（4）输出（Input）

输出文件必须是名为 flower.out 的正文文件，文件应包含两行：

第一行是程序所产生摆放方式的美学值。

第二行必须用 F 个数表示摆放方式，即该行的第 K 个数表示花束 K 所在的花瓶的编号。

（5）例子

flower.inp：

3　5

7　23　—5　—24　16

5　21　—4　10　23

—21　5　—4　—20　20

flower.out：

53

4　5

（6）评分

程序必须在 2 秒钟内运动完毕。

在每个测试点中，完全正确者才能得分。

二、分析

《花店橱窗布置问题》讲的是：在给定花束顺序的前提下，如何将花束插入到花瓶中，才能产生最大美学值。

下面我们分析一下该题的数学模型。

三、数学模型的建立

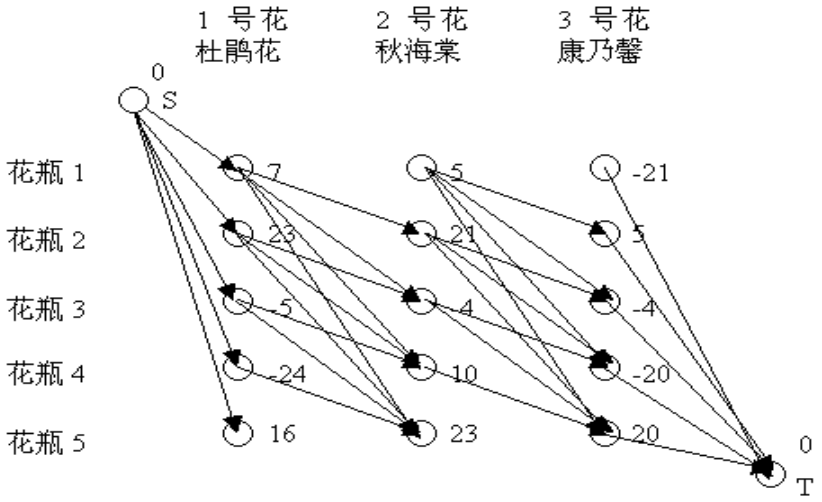
考虑通过有向图 G 来建立数学模型。假设有 F 束花，V 个花瓶，我们将第 i 束花插入第 j 号花瓶看成一个点(i,j)，点(i,j)具有一个权值，就是第 i 束花插在第 j 号花瓶中所产生的美学值 A(i,j)。

为了体现出花束摆放的顺序，从(i,j)向点(i+1,k)做有向弧，其中 k>j。

增加源点 S=(0,0)和汇点 T=(F+1,V+1)。点 S 向点(1,k) 做有向弧，点(F,k) 向点 T 做有向弧，其中 1≤k≤V。S 和 T 的权均为 0。

以问题给出的示例为例，将问题抽象成图如下：

样例中的数据		花瓶				
		1	2	3	4	5
花束	杜鹃花	7	23	-5	-24	16
	秋海棠	5	21	-4	10	23
	康乃馨	-21	5	-4	-20	20



设 f 为图 G 中 S 到 T 的有向路径集合，g 为满足题设的花束摆放集合。下面，我们建立 f→g 的一一映射：由图 G 的构造可知，对于任何一条从 S 到 T 且长度为 k 的有向路径 $P=(i_0,j_0)\rightarrow(i_1,j_1)\rightarrow\ldots(i_k,j_k)$，有 $i_x=i_{x-1}+1$，$j_x>j_{x-1}(1\leqslant x\leqslant k)$。而$(i_0,j_0)=S=(0,0)$，所以 $i_0=0$。又 $i_x=i_{x-1}+1$，$(i_k,j_k)=T=(F+1,V+1)$，所以 $i_x=x$，$k=F+1$。我们把第 $x(1\leqslant x\leqslant F)$束花放在第 j_x 号花瓶中，由于 $j_x>j_{x-1}$，该方案显然符合题目的条件。对于任意的一个满足题设的摆放方案，我们都可以类似的从图 G 中找到一条从 S 到 T 的有向路径 P 与之对应。另外，路径 P 上各顶点的权值之和为$\Sigma_{(x=1..F)}A(x,j_x)$，正是该路径对应方案所产生的美学值。

注意到图 G 中没有环，这一点可以用反证法证明：若图 G 中有一条有向圈 $P=(i_0,j_0)\rightarrow(i_1,j_1)\rightarrow\ldots\rightarrow(i_k,j_k)$，其中$(i_0,j_0)=(i_k,j_k)$，而 $i_0<i_1<\ldots i_k$，这与假设 $i_0=i_k$ 矛盾。

根据上面的分析，如果我们将顶点上的权值看做是引向该顶点的弧的长度，则该问题的实质是求一个有向无环图的最长路径。

四、动态规划求解模型

求有向无环图的最长路径，一个比较好的算法是动态规划。我们按照花束来分阶段。设 P[i,j]表示从原

点 S 到顶点(i,j)的最长路径的长度。由图 G 的构造可知，只有(i-1,0),(i-1,1),…,(i-1,j-1)到(i,j)有有向弧，且弧长都是 A(i,j)。也就是说，从 S 到(i,j)的最长路径长度，必然是由 S 到(i,k)的最长路径长度加上 A(i,j)所得，其中 0≤k≤j-1。因此，我们有动态规划方程：

P[0,0]=0

P[i,0]=-∞(1≤i≤F+1)

$P[i,j]=Max_{(0\leqslant k\leqslant j-1)}\{P[i-1,k]\}+A(i,j)(1\leqslant i\leqslant F+1,1\leqslant j\leqslant V+1)$　　①

最大美学值即为 P[F+1,V+1]。

该算法的时间复杂度为 $O(FV^2)$，仍然存在优化的余地。

设 $Q[i,j]=Max_{(0\leqslant k\leqslant j)}\{P[i,k]\}$，代入①式，得

P[i,j]=Q[i-1,j-1]+A(i,j)　　②

而 Q[i,j]=Max{Q[i,j-1],P[i,j]}　　③

将②代入③，得

Q[i,j]=Max{Q[i,j-1],Q[i-1,j-1]+A(i,j)}

这样，我们有改进后的动态规划方程：

Q[0,0]:=0

Q[i,0]:=-∞(1≤i≤F+1)

Q[i,j]:=Max{ Q[i,j-1],Q[i-1,j-1]+A(i,j)}(1≤i≤F,1≤j≤V)

最大美学值即为 Q[F,V]。

改进后的算法时间复杂度和空间复杂度都是 O(FV)。由于 1≤F,V≤100，这样的复杂度是可以接受的。

五、小结

上述动态规划方程是在有向图无环 G 的基础上得到的。如果设 Q_{ij} 表示前 i 束花放在前 j 号花瓶中所得到的最大美学值，同样可以得到上面的规划方程，而且同样容易理解。

选课（CTSC'98）

一、问题描述

大学里实行学分。每门课程都有一定的学分，学生只要选修了这门课并考核通过就能获得相应的学分。学生最后的学分是他选修的各门课的学分的总和。

每个学生都要选择规定数量的课程。其中有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其它的一些课程的基础上才能选修。例如，《数据结构》必须在选修了《高级语言程序设计》之后才能选修。我们称《高级语言程序设计》是《数据结构》的先修课。每门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。为便于表述每门课都有一个课号，课号依次为 1，2，3，……。下面举例说明

课号	先修课号	学分
1	无	1
2	1	1
3	2	3
4	无	3
5	2	4

上例中 1 是 2 的先修课，即如果要选修 2，则 1 必定已被选过。同样，如果要选修 3，那么 1 和 2 都一定已被选修过。

学生不可能学完大学所开设的所有课程，因此必须在入学时选定自己要学的课程。每个学生可选课程的总数是给定的。现在请你找出一种选课方案，使得你能得到学分最多，并且必须满足先修课优先的原则。

假定课程之间不存在时间上的冲突。

输入

输入文件的第一行包括两个正整数 M、N（中间用一个空格隔开）其中 M 表示待选课程总数（1≤M≤1000)，N 表示学生可以选的课程总数（1≤N≤M)。

以下 M 行每行代表一门课，课号依次为 1，2……M。每行有两个数（用一个空格隔开），第一个数为这门课的先修课的课号（若不存在先修课则该项为 0），第二个数为这门课的学分。学分是不超过 10 的正整数。

输出

输出文件第一行只有一个数，即实际所选课程的学分总数。以下 N 行每行有一个数，表示学生所选课程的课号。

输入输出示例

INPUT.TXT	OUTPUT.TXT
7　4	13
2　2	2
0　1	6
0　4	7
2　1	3
7　1	
7　6	
2　2	

二、分析

本题看上去不太好动手。这是一道求最优解的问题，如果用搜索解题，那规模未免过于庞大；用动态规划，本题数据之间的关系是树形，和我们往常见到线性的数据关系不一样。

怎么办？我们先从一些简单的数据入手分析。如表 1 所示的输入数据，我们可将它看为由两棵树组成的森林，如图 1 所示。

我们添加一个顶点 0，并且在每棵树的顶点与 0 之间连一条边使森林成为一棵树，如图 2。

我们发现，我们可以选取某一个点 k 的条件只是它的父节点已经被选取或者它自己为根节点；而且我

7 4
2 2
0 1
0 4
2 1
7 1
7 6
2 2

表 1

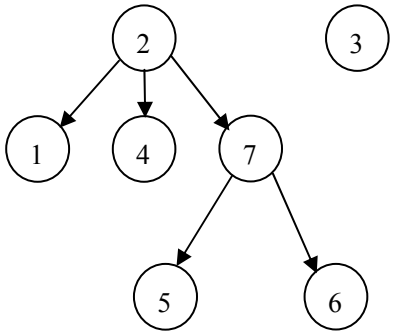


图 1

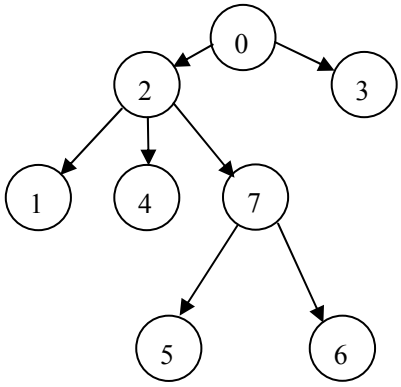


图 2

们不论如何取 k 的子孙节点，都不会影响到它父节点的选取情况，这满足无后效性原则。于是我们猜测，是不是可以以节点为阶段，进行动态规划呢？

我们用函数 f(I,j)表示以第 i 个节点为父节点，取 j 个子节点的最佳代价，则：

$f(i,j)=\max\sum_{ch1n=1}^{j}\sum_{ch2n=1}^{j-ch1n}\sum_{ch3n=1}^{j-ch1n-ch2n}\cdots\sum_{chin}^{j-ch1n-ch2n-\cdots ch(i-1)n}(f(ch1,ch1n)+f(ch2,ch2n)+\cdots+f(chi,chin))$ 可是如此规划，其效率与搜索毫无差别，虽然我们可以再次用动态规划来使它的复杂度变为平方级，但显得过于麻烦。

我们继续将树改造：原本是多叉树，我们将它转变为二叉树。如果两节点 a,b 同为兄弟，则将 b 设为 a 的右节点；如果节点 b 是节点 a 的儿子，则将节点 b 设为节点 a 的左节点。树改造完成后如图 3。

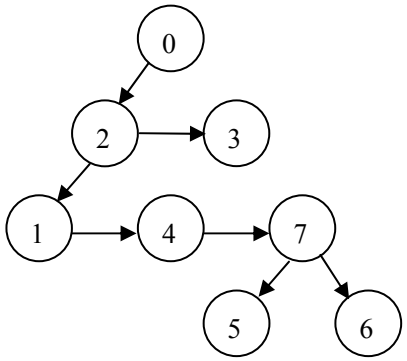


图 3

我们用函数 f(I,j)表示以第 i 个节点为父节点，取 j 个子节点的最佳代价，这和前一个函数表示的意义是一致的，但方程有了很大的改变：

$f(i,j)=\max\sum_{lcn=1}^{j}(f(leftc,lcn)+f(rightc,j-lcn))$ 这个方程的时间复杂度最大为 n3，算十分优秀了。

在具体实现这道题时，我们可以自顶而下，用递归进行树的遍历求解；在空间方面，必须特别注意。因为如果保存每一个 f(I,j)，bp 下是不可能的。我们必须用多少开多少，这样刚好可以过关。（具体请参见程序）

三、程序

```
{$Q-,R-,S-}
{$M 65520,0,655360}
const maxn = 1001;
      infile = 'input.txt';
      outfile = 'output.txt';
type flink = ^ftype;
     fnode = record
               ln, value : integer;
             end;
     ftype = array[0..maxn] of fnode;
     listtype = array[0..maxn] of integer;
     treenode = record
                  l, r, cost : integer;
                  f : flink;
                end;
     treetype = array[-1..maxn] of treenode;

var
    tree : treetype;
    m, n : integer;

procedure init;
var
    lastc : listtype;
    fa, ch : integer;
begin
```

```
  assign(input, infile);reset(input);
  fillchar(lastc, sizeof(lastc), 0);
  readln(m, n);tree[0].l := -1;tree[0].r := -1;
  for ch := 0 to m do with tree[ch] do begin
    l := -1;r := -1;
  end;
  for ch := 1 to m do begin
    readln(fa, tree[ch].cost);
    if lastc[fa] = 0  then begin
      lastc[fa] := ch;
      tree[fa].l := ch;
    end else begin
      tree[lastc[fa]].r := ch;
      lastc[fa] := ch;
    end;
  end;
  getmem(tree[-1].f, 8);
  tree[-1].f^[1].value := 0;
  tree[-1].f^[0].value := 0;
end;

function getf(node : integer) : integer;
var
    ln, rn, i, j, st, tar, max, maxj, p : integer;
begin
  getf := 0;
  if node = -1 then exit;
  ln := getf(tree[node].l);
  rn := getf(tree[node].r);
  inc(ln);
  getmem(tree[node].f, (ln + rn + 1) * 4);
  fillchar(tree[node].f^, (ln + rn + 1) * 4, 0);
  for i := 1 to ln + rn do begin
    max := 0;maxj := 0;
    st := i - rn;if st < 0 then st := 0;
    tar := ln;if i < tar then tar := i;
    for j := st to tar do begin
      p := tree[tree[node].l].f^[j].value +
           tree[tree[node].r].f^[i- j].value;
      if j > 0 then begin
          p:=tree[tree[node].l].f^[j-1].value +
             tree[tree[node].r].f^[i-j].value;
          p := p + tree[node].cost;
      end;
      if p > max then begin
```

```
          max := p;maxj := j;
        end;
      end;
      tree[node].f^[i].value := max;
      tree[node].f^[i].ln := maxj;
    end;
    getf := ln + rn;
  end;

  procedure out;
    procedure printtree(node, noden : integer);
    begin
      if noden <= 0 then exit;
      if (tree[node].f^[noden].ln > 0)and(node <> 0) then writeln(node);
      printtree(tree[node].l, tree[node].f^[noden].ln - 1);
      printtree(tree[node].r, noden - tree[node].f^[noden].ln);
    end;

  begin
    assign(output, outfile);rewrite(output);
    inc(n);
    writeln(tree[0].f^[n].value);
    if n <> 1 then printtree(0, n);
  end;
  begin
    init;
    getf(0);
    out;
  end.
```

拯救大兵瑞恩（CTSC'99）

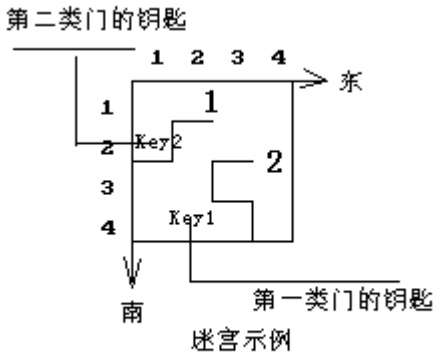
一、问题描述

1944 年，特种兵麦克接到国防部的命令，要求立即赶赴太平洋上的一个孤岛，营救被敌军俘虏的大兵瑞恩。瑞恩被关押在一个迷宫里，迷宫地形复杂，但是幸好麦克得到了迷宫的地形图。

迷宫的外形是一个长方形，其在南北方向被划分为 N 行，在东西方向被划分为 M 列，于是整个迷宫被划分为 N*M 个单元。我们用一个有序数对（单元的行号，单元的列号）来表示单元位置。南北或东西方向相邻的两个单元之间可以互通，或者存在一扇锁着的门，又或者存在一堵不可逾越的墙。迷宫中有一些单元存放着钥匙，并且所有的门被分为 P 类，打开同一类的门的钥匙相同，打开不同类的门的钥匙不同。

大兵瑞恩被关押在迷宫的东南角，即（N,M）单元里，并已经昏迷。迷宫只有一个入口，在西北角，也就是说，麦克可以直接进入(1,1)单元。另外，麦克从一个单元移动到另一个相邻单元的时间为 1，拿取所在单元的钥匙的时间以及用钥匙开门的时间忽略不计。

你的任务是帮助麦克以最快的方式抵达瑞恩所在单元，营救大兵瑞恩。



迷宫示例

输入：

第一行是三个整数，依次表示 N,M,P 的值；

第二行是一个整数 K，表示迷宫中门和墙的总个数；

第 I+2 行（1<=I<=K），有 5 个整数，依次为 Xi1,Yi1,Xi2,Yi2,Gi：

当 Gi>=1 时，表示(Xi1,Yi1)单元与(Xi2,Yi2)单元之间有一扇第 Gi 类的门，当 Gi=0 时，表示(Xi1,Yi1)单元与(Xi2,Yi2)单元之间有一堵不可逾越的墙；

（其中，|Xi1-Xi2|+|Yi1-Yi2|=1，0<=Gi<=P）

第 K+3 行是一个整数 S，表示迷宫中存放的钥匙总数；

第 K+3+J 行(1<=J<=S)，有 3 个整数，依次为 Xi1,Yi1,Qi：表示第 J 把钥匙存放在(Xi1,Yi1)单元里，并且第 J 把钥匙是用来开启第 Qi 类门的。(其中 1<=Qi<=P)

注意：输入数据中同一行各相邻整数之间用一个空格分隔。

输出：

输出文件只包含一个整数 T，表示麦克营救到大兵瑞恩的最短时间的值，若不存在可行的营救方案则输出-1。

输入输出示例：

输入文件

```
4 4 9
9
1 2 1 3 2
1 2 2 2 0
2 1 2 2 0
2 1 3 1 0
2 3 3 3 0
2 4 3 4 1
3 2 3 3 0
3 3 4 3 0
4 3 4 4 0
2
2 1 2
4 2 1
```

输出文件

```
14
```

参数设定：

3<=N,M<=15;

1<=P<=10;

二、分析

该问题可以简述为，在一 N 行 M 列的迷宫中，寻找一条从起点（1，1）到（n , m）的最短路径。要是该问题没有设置门，则很容易解决。当有门和钥匙存在时，问题的关键就是要确定到哪些地点去拿钥匙和拿钥匙的顺序。因此我们可以将起点，终点和存放了钥匙的单元格抽象成点，问题转变成怎样选择经过这些点的路径，得到最优解。

对于节点的描述，即表示我们手上有哪些钥匙，由于 P〈=10，所以我们可以用一个 p 位的二进制数 s 来表示，若 s 的第 I 位为 0，表示还没有这种钥匙；相应的，若其第 I 位为 1，则表示已经有了。这样最多有 2^p 种状态。

当我们手中的钥匙数增多时，s 的值也在不断地增加，这表示我们可以按 s 从小到大的次序进行规划，因为当前的状态只会扩展出 s 值更大的状态，而不会涉及到 s 值更小的状态，即可保证无后效性。

设 F[State]^[I,j]表示从（1，1）到达（I,j），钥匙状态为 State 时，所需的最少时间，在扩展新的状态时，首先要以（I,j）为起点进行一次宽度优先搜索，确定在当前拥有这些钥匙的状态下，从（I,j）到其他各单元格的最短距离 d[x,y]。如果单元格（x,y）有第 k 种钥匙，那么修改最优值：

(1) if f[State]^[I,j]+d[x,y]<f[State]^[x,y] then f[State]^[x,y]:= F[State]^[I,j]+d[x,y];

(2) if (state and 1 shl (k-1)=0) and (f[State]^[I,j]+d[x,y] < f[state+1 shl (k-1)]) then f[state+1 shl (k-1)]:= f[State]^[I,j]+d[x,y];

这种算法的空间复杂度为：2^maxp*maxn*maxm, 约为 230K，而时间复杂度为 0（2^p*n^2*m^2）.

三 源程序

```
{$A+,B-,D+,E+,F-,G-,I-,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+,Y+}
{$M 16384,0,655360}

program lcy;
const
  name1                    = 'rescue.in';
  name2                    = 'rescue.out';
  maxn                     = 15;
  maxm                     = 15;
  maxp                     = 20;
  del                      : array [1..4,1..2] of shortint = ((-1,0),(0,1),(0,-1),(1,0));
  two                      : array [0..10] of integer = (0,1,2,4,8,16,32,64,128,256,512);

type
  ftype                    = array [1..maxn,1..maxm] of word;
  keytype                  = record
                               x,y, （位置）
                               kind （存放的钥匙种类） : byte;
                             end;
var
  fi,fo                    : text;
  n,m,p,keyn               : byte;
```

```
  g                              : array [1..maxn,1..maxm,1..4] of shortint; （对于每个单元格保存它到四个相
                                   邻的格子的门说通道的情况），
  key                            : array [0..maxn*maxm] of keytype;
  f                              : array [0..1023] of ^ftype;
  hold                           : array [0..maxp] of boolean;
  ans                            : word;
  dis                            : array [1..maxn,1..maxm] of word;

procedure init;
var
  i,k                  : integer;
  x1,y1,x2,y2,gi,t : byte;

begin
  assign(fi,name1); reset(fi);
  assign(fo,name2); rewrite(fo);
  fillchar(g,sizeof(g),0);
  readln(fi,n,m,p);
  readln(fi,k);
  for i:=1 to k do
    begin
      readln(fi,x1,y1,x2,y2,gi);
      if (x1>x2) or (x1=x2) and (y1>y2) then
        begin
          t:=x1; x1:=x2; x2:=t;
          t:=y1; y1:=y2; y2:=t;
        end;
      if x1=x2 then
        if gi=0 then begin g[x1,y1,2]:=-1; g[x2,y2,3]:=-1; end
          else begin g[x1,y1,2]:=gi; g[x2,y2,3]:=gi; end
        else if gi=0 then begin g[x1,y1,4]:=-1; g[x2,y2,1]:=-1; end
           else begin g[x1,y1,4]:=gi; g[x2,y2,1]:=gi; end;
    end;
  readln(fi,keyn);
  for i:=1 to keyn do
    with key[i] do readln(fi,x,y,kind);
  key[0].x:=1; key[0].y:=1; key[0].kind:=0;
  close(fi);
end;

procedure shortdis(stx,sty : byte; base:word);
var
  que                : array [1..maxn*maxm,1..2] of byte;
  op,cl,x,y,d,i,
  xx,yy              : byte;
```

```
begin
  fillchar(que,sizeof(que),0);
  fillchar(dis,sizeof(dis),$ff);
  dis[stx,sty]:=base;
  op:=1; cl:=0;
  que[1,1]:=stx; que[1,2]:=sty;
  while cl<op do
    begin
      inc(cl);
      x:=que[cl,1]; y:=que[cl,2];
      d:=dis[x,y];
      for i:=1 to 4 do
        begin
          xx:=x+del[i,1]; yy:=y+del[i,2];
          if (xx in [1..n]) and (yy in [1..m])
            and (g[x,y,i]>=0) and (hold[g[x,y,i]]) and (d+1<dis[xx,yy]) then
              begin
                inc(op);
                que[op,1]:=xx; que[op,2]:=yy;
                dis[xx,yy]:=d+1;
              end;
        end;
    end;
end;

procedure work;
var
  i,news          : word;
  j,x,y,x2,y2,q   : byte;

begin
  ans:=$ffff;
  fillchar(f,sizeof(f),0);
  new(f[0]);
  fillchar(f[0]^,sizeof(f[0]^),$ff);
  f[0]^[1,1]:=0;
  for i:=0 to two[p+1]-1 do
    if f[i]<>nil then
      begin
        fillchar(hold,sizeof(hold),false);
        hold[0]:=true; （我们规定没有门即第 0 种门，这便于以后做统一处理）
        for j:=1 to p do   hold[j]:=(i and two[j]>0); （将 I 转换成其对应的每种钥匙是否已经拥有）
        for j:=0 to keyn do
          with key[j] do
```

```
            if (f[i]^[x,y]<ans) {若此时所花费的时间就已经超过当前最优解，那么这个状态不可能得出比
                                当前最优解更优的解}
               and (f[i]^[x,y]+n-x+m-y<ans) （从（x,y）到（n,m），最少也还需要 n-x+m-y 个单位时间）
                                then
            begin
              shortdis(x,y,f[i]^[x,y]); （宽度优先搜索，求出从（x,y）到其他各点
                                需要的最短时间）
              if dis[n,m]<ans then ans:=dis[n,m];
              for q:=1 to keyn do
                with key[q] do
                  if not hold[kind] and (dis[x,y]<f[i+two[kind]]^[x,y]) then
                    begin
                      news:=i+two[kind];
                      if f[news]=nil then
                        begin
                          new(f[news]); fillchar(f[news]^,sizeof(f[news]^),$ff);
                        end;
                      f[news]^[x,y]:=dis[x,y];
                    end;
            end;
        end;
end;

procedure out;
begin
  if ans=$ffff then writeln(fo,'-1') else writeln(fo,ans);
  close(fo);
end;

begin
  init;
  work;
  out;
end.
```

补丁 VS 错误（CSTS’99）

一、问题描述

错误就是人们所说的 Bug。用户在使用软件时总是希望其错误越少越好，最好是没有错误的。但是推出一个没有错误的软件几乎不可能，所以很多软件公司都在疯狂地发放补丁（有时这种补丁甚至是收费的）。T 公司就是其中之一。

上个月，T 公司推出了一个新的字处理软件，随后发放了一批补丁。最近 T 公司发现其发放的补丁有致命的问题，那就是一个补丁在排除某些错误的同时，往往会加入另一些错误。

此字处理软件中只可能出现 n 个特定的错误，这 n 个错误是由软件本身决定的。T 公司目前共发放了 m 个补丁，对于每个补丁，都有特定的适用环境，某个补丁只有当软件中包含某些错误而同时又不包含另

一些错误时才可以使用。如果它被使用，它将修复某些错误而同时加入某些错误。另外，使用每个补丁都要耗一定的时间（即补丁程序的运行时间）

更准确地说明：

设此字处理软件中可能出现的 n 个错误为集合 B={b1,b2,…,bn}中的元素，T 公司目前共发放了 m 个补丁：P1,P2,…,Pm。对于每一个补丁 Pi，都有特定的适用环境，某个补丁只有当软件中包含某些错误而同时又不包含另一些错误时才可以使用，为了说明清楚，设错误集合：Bi+,Bi-，当软件包含了 Bi+中的所有错误，而没有包含 Bi-中的任何错误时，补丁 Pi 才可以被使用，否则不能使用，显然 Bi+,Bi-的交集为空。补丁 Pi 将修复某些错误而同时加入某些错误，设错误集合 Fi+,Fi-，使用过补丁 Pi 之后，Fi-中的任何错误都不会在软件中出现，而软件将包含 Fi+中的所有错误，同样 Fi-,Fi+交集为空。另外，使用每个补丁都要耗一定的时间（即补丁程序的运行时间）。

现在 T 公司的问题很简单，其字处理软件的初始版本不幸地包含了集合 B 中的全部 n 个错误，有没有可能通过使用这些补丁（任意顺序地使用，一个补丁可使用多次），使此字处理软件成为一个没有错误的软件。如果可能，希望找到总耗时最少的方案。

二、问题分析

若将题意转换一下，将每种错误集合状态看作顶点，如果错误集合 A 可以通过某个补丁转换成错误集合 B，就连一条有向边 A→B，其权值为补丁时间。这样就将原题转换成求有向图的最短路径问题。所以可以用 Dijkstra 的最短路径算法，其状态转移方程为：

F[I]=Min{F[j]+Time[I,j]}(1<=j<=n)

但是由于此题的顶点数可达到 2^n ，上述算法的时空复杂度均无法忍受。必须充分利用信息对其进行优化。

性质：根据初始点和边的规则（即补丁），可以确定终端点。

由于实际上并不会用到 2^n 个点，那么我们无需全部保存，只需将有关联的顶点保存下来，对这些点进行上述算法。方法是：从[1..n]开始扩展所需顶点，用一个有序表P（按耗时从小到大排序）记录已扩展顶点，取当前耗时最小的顶点（显然它的耗时为到这个顶点的最小耗时）根据补丁规则扩展顶点，并记录最小值添加到P。直到耗时最小顶点为[]为止。

可见，有序表P须用链式存储结构。优化后的算法实质就是广度优先搜索。其时间复杂度为$O(2^n *m)$，由于n<=20,m<=100，所以是可行的。

三、参考程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}

program Bugs;
  const finp          ='input.txt';
        fout          ='output.txt';
        maxm          =100;
        maxn          =20;

type    set1          =set of 1..maxn;
        Tlist         =^TNode;
        Tnode         =record
```

```
                    tt    :longint;
                    bug :set1;
                    next:Tlist
                  end;

var     n,m             :integer;
        b0,b1,f0,f1:array[1..maxm]of set1;      {b0[I]——Bi-，b1[I]——Bi+，
                                                 f0[I]——Fi-，f1[I]——Fi+}
        p               :array[1..maxm]of longint; {补丁时间}
        head,tail   :Tlist;   {有序表头、表尾}

procedure init;   {输入}
  var i,j :integer;
      ch    :char;
  begin
    assign(input,finp);reset(input);
    readln(n,m);
    for i:=1 to m do begin
      read(p[i]);
      read(ch);writeln(p[i]);
      b0[i]:=[];b1[i]:=[];
      for j:=1 to n do begin
        read(ch);
        case ch of
          '+':b1[i]:=b1[i]+[j];
          '-':b0[i]:=b0[i]+[j]
       end
      end;
      read(ch);
      f0[i]:=[];f1[i]:=[];
      for j:=1 to n do begin
        read(ch);
        case ch of
          '+':f1[i]:=f1[i]+[j];
          '-':f0[i]:=f0[i]+[j]
        end
      end;
      readln
   end;
    close(input)
  end;

function ok(j:integer):boolean;
  begin
    if (b1[j]*head^.bug=b1[j])and(b0[j]*head^.bug=[])      {判断补丁能否使用}
```

```
        then ok:=true
        else ok:=false
  end;

procedure produce(j:integer);
   var r,q,q1                          :Tlist;
  begin
     new(r);        {扩展新结点}
     r^.tt:=head^.tt+p[j];
     r^.bug:=head^.bug-f0[j]+f1[j];
     q:=head;
     q1:=q;
     while q<>nil do begin
        if q^.tt>r^.tt then break;
        if q^.bug=r^.bug then begin
           dispose(r);exit
        end;
        q1:=q;q:=q^.next;
     end;
     q1^.next:=r;r^.next:=q;        {记录最优值，添加进表}
     q1:=r;
     while q<>nil do begin
       if q^.bug=r^.bug then begin
           q1^.next:=q^.next;dispose(q);        {删除不优值}
           q:=q1
        end;
        q1:=q;q:=q^.next
     end
  end;

Procedure main;
  var j      :integer;
        r      :Tlist;
  begin
     new(head);new(tail);
     head^.next:=tail;
     tail^.tt:=0;tail^.bug:=[1..m];tail^.next:=nil;     {建立第一个顶点}
     repeat
        r:=head;
        head:=head^.next;
        dispose(r);
       if head^.bug=[] then break;
        for j:=1 to m do       {扩展}
           if ok(j) then
              produce(j);
```

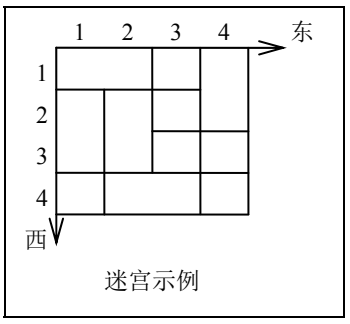
```
    until head^.next=nil;
    assign(output,fout);rewrite(output);
    if head^.bug=[] then writeln(head^.tt)          {输出最优值}
                    else writeln(0);                {输出无解}
    close(output)
 end;

Begin   {主程序}
 init;
 main
End.
```

迷宫改造（WC'99）

一、问题描述

A娱乐公司最近获得了一些古希腊迷宫的拥有权，为了使这些古典式迷宫能够吸引更多的游客，A公司计划对这些迷宫进行合理的改造。任务是根据所给的一个迷宫，针对公司的要求，在原有迷宫的基础上设计出一个最佳的新式迷宫。

迷宫外形是一个长方形，如下图所示。



迷宫示例

在南北方向被划分为 N（3<=N<=20）行，在东西方向被划分为M（3<=M<=20）列，于是整个迷宫被划分为N*M个单元。用一个有序数队（单元的行号，单元的列号）来表示位置。南北或东西方向相邻的两个单元之间存在一堵墙或者一扇门，墙是不可逾越的，而门是双向的且可以任意通过。出于保护文物的目的，A公司决定只适当地将墙改置为门，而不进行其他改造，并且要求新迷宫是最佳的，即新置的门的总数要最少。

首先需要设计出一个最佳迷宫，使游客可以在改造后的新式迷宫的任一单元出发，到达其他任何单元。这样的最佳迷宫称为第一类最佳迷宫。

另外，A公司计划推出一个面向家庭的迷宫游戏，游戏规则如下：

假定有P（1<=P<=3）个家庭成员，他们分别从P个指定的起点出发，要求他们只能向南或向东移动，分别到达P个指定的终点。

针对上述游戏规则，需要设计一个最佳迷宫，使这样的游戏是可行的，即所有家庭成员可从各自的起点出发依照游戏规则到达各自的终点。这样的迷宫称为第二类最佳迷宫。

输入文件

第一行是两个整数,依次表示N,M的值;(3<=n,m<=20)

第二行是一个整数K,表示原迷宫中门的总个数;

第I+2行(1<=I<=K),有4个整数,依次为Xi1,Yi1,Xi2,Yi2,表示第Xi1行Yi1列的单元与第Xi2行Yi2列的单元之间有一扇门.(其中|Xi1- Yi1|+ |Xi2 -Yi2|=1)

第I+3行是一个整数,表示P的值;(1<=p<=3)

第K+3+J行(1<=J<=P),有4个整数Xj1,Yj1,Xj2,Yj2,分别表示第J个家庭成员出发的起点位置(Xj1,Yj1)和要到达的终点位置(Xj2,Yj2).(其中Xj1<= Xj2, Yj1<= Yj2)　　(Xj1,Yj1)<>(Xj2,Yj2))

注意:输入数据中同一行各相邻整数之间用一个空格分隔

输出文件

共 P+2 行

第一行是一个整数,表示你所设计的第一最佳迷宫中,新置的门的个数;

第二是一个整数,表示你所示的第二类最佳迷宫,新置的门的个数;

以下还有 P 行,分别表示各家庭成员从第二类最佳迷宫中起点到终点的一条可行路径,其中:　第 I+2 行(1<=I<=P)表示第 I 个家庭成员的一条可行路径(包括起点),该行只有一个由大写字母 S 和大写字母 E 组成的字符串,第 J 位字符表示他在路径上的第 J 个单元的移动方向:大写字母 S 表示向南,大写字母 E 表示向东.

样本输入文件

4　4

5

1 1 1 2

1 4 2 4

2 1 3 1

2 2 3 2

4 2 4 3

3

2 1 4 3

1 2 4 2

3 1 4 4

样本输出文件

10

4

SESE

SSS

ESEE

二、分析

任务 1 比较简单，把迷宫看成一个图，两个单元间无墙则连一条边，解决此任务只需求出图中连通分量的个数，新置的门数即为连通分量数－1。求连通分量也不一定要构图，因为规模不大，用简单的广搜或者深搜也能求出。时间复杂度大约为 O（n*m），不是很大。

算法如下：

```
proc mission1;
[repeat
    找到一个为到达过的单元；
    搜索所有与该单元相连通的单元并全部置到达过的标志；
until 所有单元都到达过；
]
```

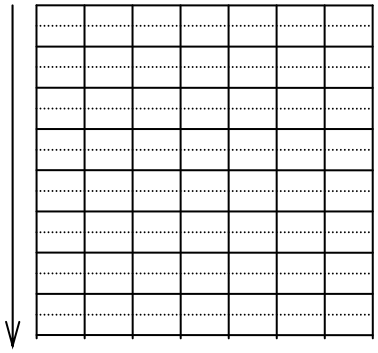
对于任务 2

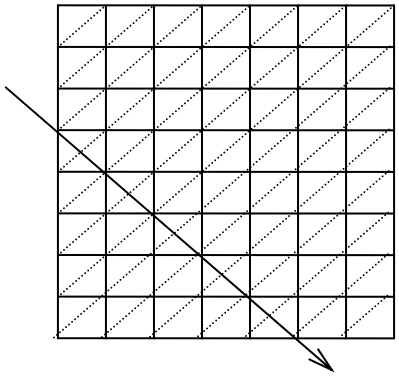
1.搜索。数据较小时，可能还是可以出解的，但对于大数据甚至只需稍大就毫无办法。假设只有一个人，当 N=20,M=20,起点为(1,1),终点为(20,20)时，需走 39 步，在大多数情况下每步有两种选择，搜索量至少就有 2^{20}，人增多时指数以倍数增长。搜索固然是不行的。

2.贪心。贪心有多种贪心法则，速度都很快，但肯定不能全对。

3.动态规划。这里所说的动态规划，是说那种较容易想到的六维动态规划，用六个坐标

（x1,y1,x2,y2,x3,y3）表示三个人分别走到上述三个坐标所需要的最小费用（划分方式一）。由于它所需要的空间就有 20^6=1600000，连动态数组都存不了。所以，它也是行不通的。





联想做过的《方格取数》一题，便想到用对角线划分阶段，把 3 人看成站成一排（划分方式二），这样状态表示降为（x,y1,y2,y3）四维，x 表示他们所处的阶段数，y1,y2,y3 分别表是 3 人在该阶段的第几个。空间复杂度为 O（(n+m-1）*m*m*m），最大为 39*20*20*20=312000，仍然存不下。于是又在原有的优化基础上继续优化，想到只要将 3 人看成一排便能降为 4 维，如果以行或者列为阶段，状态仍然表示为（x,y1,y2,y3），x 表示 3 人所处的行号，空间复杂度又可降为原来的一半，为 O（n*m*m*m），最大为 20*20*20*20=160000，动态数组是可以存下的。状态和阶段定好后，状态转移方程便容易写出了：令函数 Make（a,b,y1,y2,y3,d1,d2,d3）表示从状态 f [a,y1,y2,y3]到状态 f [b,d1,d2,d3]所需要添加的门的张数。

F[a,y1,y2,y3]=min｛F[a,y1-1,y2,y3]+Make（a,a,y1-1,y2,y3,y1,y2,y3），
F[a,y1,y2-1,y3]+Make（a,a,y1,y2-1,y3,y1,y2,y3），
F[a,y1,y2,y3-1]+Make（a,a,y1,y2,y3-1,y1,y2,y3），
F[a-1,y1,y2,y3]+Make（a-1,a,y1,y2,y3,y1,y2,y3），
F[a,y1-1,y2-1,y3]+Make（a,a,y1-1,y2-1,y3,y1,y2,y3），
F[a,y1-1,y2,y3-1]+Make（a,a,y1-1,y2,y3-1,y1,y2,y3），
F[a,y1,y2-1,y3-1]+Make（a,a,y1,y2-1,y3-1,y1,y2,y3），
F[a,y1-1,y2-1,y3-1]+Make（a,a,y1-1,y2-1,y3-1,y1,y2,y3）｝;

最后采用倒推的方法求出人的移动路线。

三、参考程序

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R-,S-,T-,V+,X+,Y+}
{$M 16384,0,655360}

Program Maze;
Const
      sti          ='Maze.in';
      sto          ='Maze.out';
      d            :array[0..3,1..2] of -1..1
                   =((1,0),(0,1),(-1,0),(0,-1));
      maxn         =21;
      nn           =255;
Type
     arr           =array[0..maxn,0..maxn,0..maxn] of byte;
Var
```

```
    fi,fo           :text;
    n,m,p,k         :integer;
    q,s,t,w         :array[1..3] of byte;
    st              :array[1..3] of string;
    f               :array[1..maxn] of ^arr;
    start,last      :array[1..3,1..2] of byte;
    a               :array[0..maxn,0..maxn,0..3] of byte;

Procedure init;
Var
    i,j             :integer;
    x1,y1,x2,y2     :integer;
Begin
      fillchar(a,sizeof(a),1);
      assign(fi,sti);
      reset(fi);
      readln(fi,n,m);
      readln(fi,k);
      for i:=1 to k do
      Begin
        readln(fi,x1,y1,x2,y2);
        for j:=0 to 3 do
        Begin
          if (x1+d[j,1]=x2) and (y1+d[j,2]=y2) then a[x1,y1,j]:=0;
          if (x1-d[j,1]=x2) and (y1+d[j,2]=x2) then a[x2,y2,j]:=0;
        End;
      End;
      readln(fi,p);
      for i:=1 to p do
      readln(fi,start[i,1],start[i,2],last[i,1],last[i,2]);
      close(fi);
End;

Function test(i,j:byte):boolean;
Var
    l               :byte;
Begin
      test:=false;
      for l:=1 to i do
      if q[l]=j
        then exit;
      test:=true;
End;

Procedure next1(i,j1,j2,j3:byte);
```

```
Var
    j,l,s,e,c,o    :word;
    z1,z2              :boolean;
Begin
       for j:=0 to w[1] do
       Begin
          s:=0;
          if j=1
             then Begin inc(s); q[s]:=j1; End;
          for c:=0 to w[2] do
          Begin
             z1:=false;
             if (c=1) and test(s,j2)
                then Begin inc(s); q[s]:=j2; z1:=true; End;
             for l:=0 to w[3] do
             Begin
                z2:=false;
                if (l=1) and test(s,j3)
                   then Begin inc(s); q[s]:=j3; z2:=true; End;
                e:=f[i]^[j1,j2,j3];
                for o:=1 to s do
                inc(e,a[i,q[o],1]);
                if e<f[i]^[j1+j,j2+c,j3+l]
                   then f[i]^[j1+j,j2+c,j3+l]:=e;
                if z2 then dec(s);
             End;
             if z1 then dec(s);
          End;
       End;
End;

Procedure next2(i,j1,j2,j3:byte);
Begin
       if j1=j2 then
          if j1=j3 then f[i+1]^[j1,j2,j3]:=f[i]^[j1,j2,j3]+a[i,j1,0]
             else f[i+1]^[j1,j2,j3]:=f[i]^[j1,j2,j3]+a[i,j1,0]+a[i,j3,0]
      else
        if j1=j3 then f[i+1]^[j1,j2,j3]:=f[i]^[j1,j2,j3]+a[i,j1,0]+a[i,j2,0]
           else
             if j2=j3 then f[i+1]^[j1,j2,j3]:=f[i]^[j1,j2,j3]+a[i,j1,0]+a[i,j2,0]
                else f[i+1]^[j1,j2,j3]:=f[i]^[j1,j2,j3]+a[i,j1,0]+a[i,j2,0]+a[i,j3,0];
End;

Procedure main;
Var
```

```
    i,j,j1,j2,j3 :integer;
Begin
  for i:=1 to n+1 do Begin
new(f[i]);
      fillchar(f[i]^,sizeof(f[i]^),nn);
  End;
f[1]^[0,0,0]:=0;
  for i:=1 to n do Begin
   if i=start[1,1] then Begin
    for j2:=0 to m do
      for j3:=0 to m do
       if f[i]^[0,j2,j3]<>nn then f[i]^[start[1,2],j2,j3]:=f[i]^[0,j2,j3];
      s[1]:=1; t[1]:=m; w[1]:=1;
   End;
   if i=start[2,1] then Begin
     for j1:=0 to m do
        for j3:=0 to m do
            if f[i]^[j1,0,j3]<>nn then f[i]^[j1,start[2,2],j3]:=f[i]^[j1,0,j3];
        s[2]:=1; t[2]:=m; w[2]:=1;
     End;
     if i=start[3,1] then Begin
       for j1:=0 to m do
         for j2:=0 to m do
            if f[i]^[j1,j2,0]<>nn then f[i]^[j1,j2,start[3,2]]:=f[i]^[j1,j2,0];
          s[3]:=1; t[3]:=m; w[3]:=1;
     End;
     for j1:=s[1] to t[1] do
      for j2:=s[2] to t[2] do
        for j3:=s[3] to t[3] do
         if f[i]^[j1,j2,j3]<>nn then next1(i,j1,j2,j3);
     if i=last[1,1] then Begin
       for j2:=0 to m do
         for j3:=0 to m do
           if f[i]^[last[1,2],j2,j3]<>nn then f[i]^[0,j2,j3]:=f[i]^[last[1,2],j2,j3];
       s[1]:=0; t[1]:=0; w[1]:=0;
     End;
     if i=last[2,1] then Begin
       for j1:=0 to m do
        for j3:=0 to m do
            if f[i]^[j1,last[2,2],j3]<>nn then f[i]^[j1,0,j3]:=f[i]^[j1,last[1,2],j3];
       s[2]:=0; t[2]:=0; w[2]:=0;
     End;
     if i=last[3,1] then Begin
       for j1:=0 to m do
         for j2:=0 to m do
```

```
            if f[i]^[j1,j2,last[3,2]]<>nn then f[i]^[j1,j2,0]:=f[i]^[j1,j2,last[1,2]];
          s[3]:=0; t[3]:=0; w[3]:=0;
      End;
      for j1:=s[1] to t[1] do
       for j2:=s[2] to t[2] do
          for j3:=s[3] to t[3] do
           if f[i]^[j1,j2,j3]<>nn then next2(i,j1,j2,j3);
  End;
End;

Procedure pred(Var i,j1,j2,j3:integer);
Var
   j,c,l,s,e,o    :integer;
   z1,z2              :boolean;
Begin
      for j:=w[1] downto 0 do
        if (j1>1) or (j<1) then Begin
          s:=0;
          if j=1 then Begin inc(s); q[s]:=j1; End;
          for c:=w[2] downto 0 do
              if (j2>1) or (c<1) then Begin
                  z1:=false;
                  if (k=1) and test(s,j2) then Begin
                 inc(s); q[s]:=j2; z1:=true;
              End;
          for l:=w[3] downto 0 do
              if (j3>1) or (l<1) then Begin
                 if (j=0) and (c=0) and (l=0) then break;
                 z2:=false;
                 if (l=1) and test(s,j3) then Begin
                     inc(s); q[s]:=j3; z2:=true;
                 End;
                 e:=f[i]^[j1-j,j2-c,j3-l];
                 if e=f[i]^[j1,j2,j3] then Begin
                     dec(j1,j); if j=1 then st[1]:='E'+st[1];
                     dec(j2,c); if c=1 then st[2]:='E'+st[2];
                     dec(j3,l); if l=1 then st[3]:='E'+st[3];
                     exit;
                 End;
                 if z2 then dec(s);
             End;
             if z1 then dec(s);
     End;
End;
      if (i=start[1,1]) and (j1=start[1,2]) then Begin j1:=0; w[1]:=0; End;
```

```
        if (i=start[2,1]) and (j2=start[2,2]) then Begin j2:=0; w[2]:=0; End;
        if (i=start[3,1]) and (j3=start[3,2]) then Begin j3:=0; w[3]:=0; End;
        if j1=j2 then
          if j1=j3 then s:=a[i-1,j1,0]else s:=a[i-1,j1,0]+a[i-1,j3,0]
        else if j1=j3 then s:=a[i-1,j1,0]+a[i-1,j2,0] else
                if j2=j3 then s:=a[i-1,j1,0]+a[i-1,j2,0] else
                  s:=a[i-1,j1,0]+a[i-1,j2,0]+a[i-1,j3,0];
      if f[i-1]^[j1,j2,j3]+s=f[i]^[j1,j2,j3]    then Begin
        dec(i);
        if j1<>0 then st[1]:='S'+st[1];
        if j2<>0 then st[2]:='S'+st[2];
        if j3<>0 then st[3]:='S'+st[3];
      End;
End;

Procedure print;
Var
    i,j1,j2,j3      :integer;
Begin
        i:=n; j1:=0; j2:=0; j3:=0;
        w[1]:=0; w[2]:=0; w[3]:=0;
        while (i>1) or (j1<>0) or (j2<>0) or (j3<>0) do
        Begin
          if (i=last[1,1]) and (j1=0) then Begin j1:=last[1,2]; w[1]:=1; End;
          if (i=last[2,1]) and (j2=0) then Begin j2:=last[2,2]; w[2]:=1; End;
          if (i=last[3,1]) and (j3=0) then Begin j3:=last[3,2]; w[3]:=1; End;
          pred(i,j1,j2,j3);
        End;
        assign(fo,sto);
        rewrite(fo);
        writeln(fo,f[n]^[0,0,0]);
        for i:=1 to 3 do
        writeln(fo,st[i]);
        close(fo);
End;

Begin
      init;
      main;
      print;
End.
```

奶牛浴场（WC'2002）

一、问题描述

由于 john 建造了牛场围栏，激起了奶牛的愤怒，奶牛的产奶量急剧减少。为了讨好奶牛，john 决定在牛场中建造一个大型浴场。但是 john 的奶牛有一个奇怪的习惯，每头奶牛都必须在牛场中的一个固定位置产奶，而奶牛显然不能在浴场中产奶，于是，john 希望所建造的浴场不能覆盖这些产奶点。这回，他又要求助于 clevow 了。你能帮助 clevow 吗？

John 的牛场和规划的浴场都是矩形。浴场要完全位于牛场之内。并且浴场的轮廓要与牛场的轮廓平行或者重合。浴场不能覆盖任何产奶点，但是产奶点可以位于浴场的轮廓上。

Clevow 当然希望浴场的面积尽可能大了，所以你的任务就是帮助她计算浴场的最大面积。

输入文件

输入文件的第一行包含两个整数 L 和 W，分别表示牛场的长和宽。文件的第二行包含一个整数 N，表示产奶点的数量。以下 N 行每行包含两个整数 X 和 Y，表示一个产奶点的坐标。所有产奶点都位于牛场内，即：0<=X<=L，0<=Y<=W。

输出文件

输出文件仅有一行，包含一个整数 S，表示浴场的最大面积

输入输出样例

HAPPY.IN

10 10

4

1 1

9 1

1 0

9 9

HAPPY.OUT

80

二、初步分析

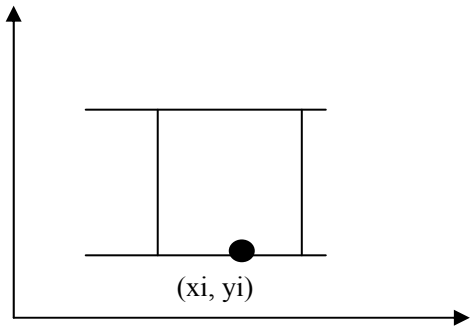
设这 n 个点的坐标为(x_1, y_1)，(x_2, y_2)，…，(x_n, y_n)，$y_1 <= y_2 <= \ldots <= y_n$。

稍加分析可知：浴场的上下左右四条边界要不同牛场的边界重合、要不就经过某一个产奶点。为了简化问题，我们假设：

所求矩形的下边界必须经过产奶点。

所有的 $y_i(1 <= i <= n)$ 互不相同。

考虑取下边界 $y = y_i$ 时，矩形的最大面积。（如下图）



设左边界是 l，右边界是 r（l <= r），无论上边界取何值，都有：

(l <= x_i) 且 (r >= x_i)　　(*)

若不然，不妨设 l < xi，r < xi（l > xi，r > xi 的情况类似可证）。令 $y_0 = 0$，那么可以让左、右、上边界都保持不变，将下边界变为 $y = y_{i-1}$，则新得到的矩形比当前的矩形面积要大。故(*)式满足。

同时矩形不能包含任何产奶点，设上边界为 y = yj，那么：

l = max{x_p, 0}　　(i < p < j,　x_p <= x_i)

r = min{x_p, w}　　(i < p < j,　x_p >= x_i)

据此，我们就可以很轻松的设计出算法：

```
for i := 1 to n do begin
  l ← 0; r ← w;
  for j := i + 1 to n do begin
    if (y[j] - y[i]) * (r - l) > 最优值 then 更新;
    if x[j] <= x[i] then l← max{l, x[j]};
    if x[j] >= x[i] then r ← min{r, x[j]}
  end;
  if (h - y[i]) * (r - l) > 最优值 then 更新    {上边界与牛场上边界重合}
end;
输出最优值
```

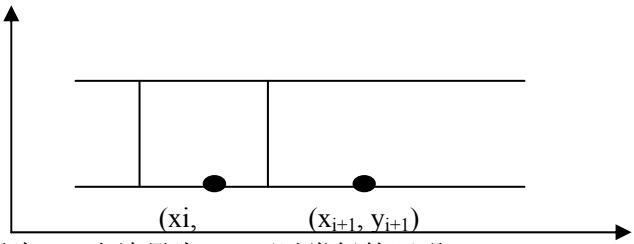
三、算法完善

上述算法的前提条件有两个：

矩形下边界经过产奶点。

所有的 y_i(1 <=i <= n)互不相同。

首先考虑 yi 可能相同的情况。取下边界为 y = yi。



设左边界为 l，右边界为 r，可以类似的证明：

(l, yi)与(r, yi)之间的线段上，至少有一个产奶点。

因此对于所有纵坐标为 yi 的点，只需每个单独处理一遍即可。修正算法如下：

```
for i := 1 to n do begin
  l ← 0; r ← w;
  for j := i + 1 to n do
    if y[j]≠y[i] then begin
      if (y[j] - y[i]) * (r - l) > 最优值 then 更新;
      if x[j] <= x[i] then l← max{l, x[j]};
      if x[j] >= x[i] then r ← min{r, x[j]}
    end;
  if (h - y[i]) * (r - l) > 最优值 then 更新
end;
输出最优值
```

至于矩形下边界与牛场边界重合的情况，使用插入排序，简单判断即可，这里不赘述。

整个算法的时间复杂度是 O(n^2)。

四、总结

通过解决本题，我们获得了一些有益的经验：

化繁为简。即提出一些假设将问题条件简化。

全面的考虑问题。各种情况必须一网打尽，不能遗漏。

五、参考程序

```
const
  fin = 'happy.in';
  fout = 'happy.out';
  maxn = 5100;

var
  n : integer;
  x, y : array[0 .. maxn] of integer;
  w, h : integer;
  result : longint;

procedure getinfo; {读入数据}
var
  i : integer;
begin
  assign(input, fin); reset(input);
  readln(w, h);
  readln(n);
  for i := 1 to n do readln(x[i], y[i]);
  close(input);
end;

procedure print; {输出}
begin
  assign(output, fout); rewrite(output);
  writeln(result);
  close(output);
end;

procedure sort;   {按照纵坐标排序}
var
  i : integer;

procedure swap(var x, y : integer);
var
  t : integer;
begin
  t := x; x := y; y := t;
end;
```

```
procedure sift(s, t : integer);
var
  i, j : integer;
begin
  i := s; j := i + i;
  while j <= t do begin
    if (j < t) and (y[j + 1] > y[j]) then inc(j);
    if y[i] >= y[j] then break else begin
      swap(y[i], y[j]); swap(x[i], x[j]); i := j; j := i + i;
    end;
  end;
end;

begin
  for i := n shr 1 downto 1 do sift(i, n);
  for i := n downto 2 do begin
    swap(x[1], x[i]); swap(y[1], y[i]);
    sift(1, i - 1)
  end;
end;

procedure initresult;  {下边界与牛场边界重合}
var
  maxw : longint;
  p, i, l, t : integer;
  d : array[0 .. maxn] of integer;
begin
  fillchar(d, sizeof(d), 0);
  result := 0;
  l := 2; d[1] := 0; d[2] := w; maxw := w;
  for t := 1 to n do begin
    if maxw * y[t] > result then result := maxw * y[t];
    p := 1; while d[p] < x[t] do inc(p);
    for i := l + 1 downto p + 1 do d[i] := d[i - 1];
    d[p] := x[t]; inc(l);
    maxw := 0;
    for i := l downto 2 do if d[i] - d[i - 1] > maxw then maxw := d[i] - d[i - 1];
  end;
  if maxw * h > result then result := maxw * h;
end;

procedure main;  {下边界不与牛场边界重合}
var
  l, r : longint;
  s, i, j : integer;
```

```
begin
  for i := 1 to n do begin
    l := 0; r := w; s := i + 1;
    while (s <= n) and (y[s] = y[i]) do inc(s);
    for j := s to n do begin
      if (r - l) * (y[j] - y[i]) > result then result := (r - l) * (y[j] - y[i]);
      if (x[j] <= x[i]) and (x[j] > l) then l := x[j];
      if (x[j] >= x[i]) and (x[j] < r) then r := x[j];
    end;
    if (r - l) * (h - y[i]) > result then result := (r - l) * (h - y[i]);
  end;
end;

begin {主过程}
  getinfo;
  sort;
  initresult;
  main;
  print;
end.
```

HPC (WC'2001)

一、问题描述

现在有一项时间紧迫的工程计算任务要交给你——国家高性能并行计算机的主管工程师——来完成。为了尽可能充分发挥并行计算机的优势，我们的计算任务应当划分成若干个小的子任务。

这项大型计算任务包括A和B两个互不相关的较小的计算任务。为了充分发挥并行计算机的运算能力，这些任务需要进行分解。研究发现，A 和 B 都可以各自划分成很多较小的子任务，所有的 A 类子任务的工作量都是一样的，所有的 B 类子任务也是如此（A 和 B 类的子任务的工作量不一定相同）。A 和 B 两个计算任务之间，以及各子任务之间都没有执行顺序上的要求。

这台超级计算机拥有 p 个计算节点，每个节点都包括一个串行处理器、本地主存和高速 cache。然而由于常年使用和不连贯的升级，各个计算节点的计算能力并不对称。一个节点的计算能力包括如下几个方面：

就本任务来说，每个节点都有三种工作状态：待机、A 类和 B 类。其中，A 类状态下执行 A 类任务；B 类状态下执行 B 类任务；待机状态下不执行计算。所有的处理器在开始工作之前都处于待机状态，而从其它的状态转入 A 或 B 的工作状态（包括 A 和 B 之间相互转换），都要花费一定的启动时间。对于不同的处理节点，这个时间不一定相同。用两个正整数 t_i^A 和 t_i^B (i=1,2,…,p)分别表示节点 i 转入工作状态 A 和工作状态 B 的启动时间（单位：ns）。

一个节点在连续处理同一类任务的时候，执行时间——不含状态转换的时间——随任务量（这一类子任务的数目）的平方增长，即：

若节点 i 连续处理 x 个 A 类子任务，则对应的执行时间为

$$t = k_i^A x^2$$

类似的，若节点 i 连续处理 x 个 B 类子任务，对应的执行时间为：

$$t = k_i^B x^2$$

其中，k_i^A 和 k_i^A 是系数，单位是 ns。i=1,2,…,p。

任务分配必须在所有计算开始之前完成，所谓任务分配，即给每个计算节点设置一个任务队列，队列由一串 A 类和 B 类子任务组成。两类子任务可以交错排列。

计算开始后，各计算节点分别从各自的子任务队列中顺序读取计算任务并执行，队列中连续的同类子任务将由该计算节点一次性读出，队列中一串连续的同类子任务不能被分成两部分执行。

现在需要你编写程序，给这 p 个节点安排计算任务，使得这个工程计算任务能够尽早完成。假定任务安排好后不再变动，而且所有的节点都同时开始运行，任务安排的目标是使最后结束计算的节点的完成时间尽可能早。

输入文件的第一行是对计算任务的描述，包括两个正整数 n_A 和 n_B，分别是 A 类和 B 类子任务的数目，两个整数之间由一个空格隔开。

文件的后面部分是对此计算机的描述：

文件第二行是一个整数 p，即计算节点的数目。

随后连续的 p 行按顺序分别描述各个节点的信息，第 i 个节点由第 i+2 行描述，该行包括下述四个正整数（相邻两个整数之间有一个空格）：

$$t_i^A \quad t_i^B \quad k_i^A \quad k_i^B$$

输出文件名是 hpc.out。其中只有一行，包含有一个正整数，即从各节点开始计算到任务完成所用的时间。

二、分析

超级计算机一共有 p 个节点，但这些节点是可以并行运作的，也就是说独立工作而可以不受影响。因此，我们可以先考虑一个节点，即 p = 1 的情况。

对于这个节点，设其转入工作状态 A 和工作状态 B 的启动时间分别为 t_a 和 t_b，处理连续任务所需的时间系数分别为 k_a 和 k_b。如果要处理 a 个 A 类任务，b 个 B 类任务，可以分两种情况讨论：

1. 最后处理任务 A

由于最后处理的是任务 A，故可设最后连续处理了 x 个 A 类任务。而我们实际上需要处理 a 个 A 类任务，b 个 B 类任务，因此，在处理这 x 个 A 任务之前，我们必须先完成一个子任务，它包括 a – x 个 A 类任务和 b 个 B 类任务，且最后处理的必须是 B 类任务。显然，这个子任务也必须是采用最优的方案。

如图 4-2，最后连续处理了 x 个 A 类任务的时间是 $k_a x^2$

节点转入工作状态 A 的时间是 t_a

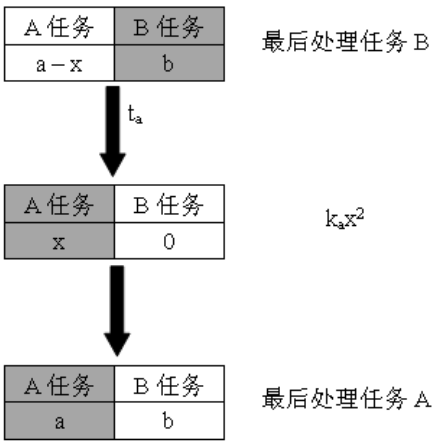


图 4-2 后处理 A 任务序列图

因此，在假定最后连续处理 x 个 A 类任务的前提下，处理 a 个 A 类任务，b 个 B 类任务，所需的最短时间，就是处理 a – x 个 A 类任务，b 个 B 类任务，且最后处理的是 B 类任务所需的最短时间 $+ k_a x^2 + t_a$

下面我们就来看看最后处理的是 B 类任务的情况。

2. 最后处理任务 B

由于最后处理的是任务 B，故可设最后连续处理了 x 个 B 类任务。而我们实际上需要处理 a 个 A 类任务，b 个 B 类任务，因此，在处理这 x 个 B 任务之前，我们必须先完成一个子任务，它包括 a 个 A 类任务和 b – x 个 B 类任务，且最后处理的必须是 A 类任务。显然，这个子任务也必须是采用最优的方案。

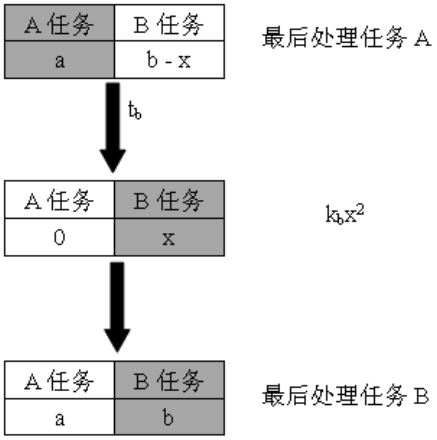


图 4-3 后处理 B 任务序列图

如图 4-3，最后连续处理了 x 个 B 类任务的时间是 k_bx^2 。节点转入工作状态 B 的时间是 t_b。因此，在假定最后连续处理 x 个 B 类任务的前提下，处理 a 个 A 类任务，b 个 B 类任务，所需的最短时间，就是处理 a 个 A 类任务，b – x 个 B 类任务，且最后处理的是 A 类任务所需的最短时间 $+ k_bx^2 + t_b$

如果用 $g_a(a, b)$表示处理 a 个 A 类任务，b 个 B 类任务，且最后处理的是 A 类任务所需的最短时间，$g_b(a, b)$ 表示处理 a 个 A 类任务，b 个 B 类任务，且最后处理的是 B 类任务所需的最短时间，根据上面的讨论，可以得到：

$$g_a(a, b) = \min_{(1\le x\le a)}\{g_b(a - x, b) + k_ax^2\} + t_a$$
$$g_b(a, b) = \min_{(1\le x\le b)}\{g_a(a, b - x) + k_bx^2\} + t_b$$

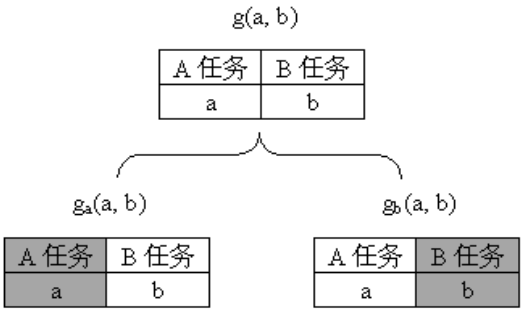


图 4-4 一个节点的情况

如果用 g(a, b)表示处理 a 个 A 类任务，b 个 B 类任务所需的最短时间，这个问题可以分解为最后处理任务 A 或最后处理任务 B 两种情况（如图 4-4）。因此，

$g(a, b) = \min\{g_a(a, b), g_b(a, b)\}$

这样，我们就完成 p = 1 的分析。

当 p > 1 时，可以这样考虑：

如果要处理 a 个 A 类任务，b 个 B 类任务，则设第一个节点负责了 i 个任务 A 和 j 个任务 B，则剩下的 a – i 个任务 A 和 b – j 个任务 B 都必须由后 p – 1 个节点完成。设 f(p, a, b)表示在后 p 个节点上处理 a 个 A 类任务，b 个 B 类任务所需的最少时间（如图 4-5）。根据上面的分析，有：

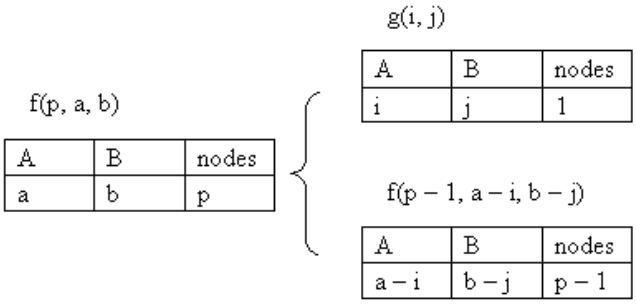


图 4-5 多个节点的情况

$f(p, a, b) = \min_{(0\le i\le a,\ 0\le j\le b)}\{\max\{g(i, j), f(p - 1, a - i, b - j)\}\}$

这样就可以处理 p > 1 的情况了。

由于计算 f(p)时，只需用到 f(p – 1)的结果，故可以使用滚动数组。这样，计算过程中，只需保存 g_a, g_b, g, f(p – 1), f(p)这 5 个大小为 n^2 的数组，故空间复杂度是 $O(n^2)$。

计算数组 g 的复杂度为 $O(n^3)$，一共有 p 个节点，故时间复杂度是 $O(pn^3)$。计算数组 f 的复杂度为 $O(pn^4)$。所以，总的时间复杂度为 $O(pn^4)$。

三、参考程序：

```
#include <fstream.h>
```

```
#include <values.h>
#include <mem.h>

const char * const finp = "hpc.in";
const char * const fout = "hpc.out";
const int maxn = 64;
const int maxp = 32;

struct Tnode {
   int ta, tb, ka, kb;
};

typedef long Tmap[maxn][maxn];

int na, nb, p;
Tmap now, f;
Tnode info[maxp];

void init()
{
   ifstream inp(finp);
   inp >> na >> nb; inp >> p;
   for (int i = 0; i < p; i ++)
      inp >> info[i].ta >> info[i].tb >> info[i].ka >> info[i].kb;
   inp.close();
}

void calc(const int &p, const long &max)
{
   long da[maxn], db[maxn], x;
   for (x = 0; x <= na; x ++) da[x] = x * x * info[p].ka + info[p].ta;
   for (x = 0; x <= nb; x ++) db[x] = x * x * info[p].kb + info[p].tb;
   Tmap a, b;
   a[0][0] = b[0][0] = f[0][0] = 0;
   for (int i = 0; i <= na; i ++)
      for (int j = 0; j <= nb; j ++)
         if ((i + j) != 0) {
            int k;
            x = max; k = 1;
            while ((k <= i) && (da[k] < x)) {
               long q = b[i - k][j] + da[k];
               if (q < x) x = q;
               k ++;
            }
            a[i][j] = x;
```

```
            x = max; k = 1;
            while ((k <= j) && (db[k] < x)) {
               long q = a[i][j - k] + db[k];
               if (q < x) x = q;
               k ++;
            }
            b[i][j] = x;
            f[i][j] = a[i][j] < b[i][j] ? a[i][j] : b[i][j];
         }
}

void start()
{
   calc(0, MAXLONG >> 1); memcpy(now, f, sizeof(now));
   for (int k = 1; k < p; k ++) {
      Tmap pre; memcpy(pre, now, sizeof(pre));
      calc(k, now[na][nb]);
      for (int i = 0; i <= na; i ++)
         for (int j = 0; j <= nb; j ++)
            if (f[i][j] < now[na][nb]) {
               long x = f[i][j];
               for (int a = i; a <= na; a ++)
                  for (int b = j; b <= nb; b ++)
                     if ((x < now[a][b]) && (pre[a - i][b - j] < now[a][b]))
                        now[a][b] = x < pre[a - i][b - j] ? pre[a - i][b - j] : x;
            }
   }
}

void print()
{
   ofstream out(fout);
   out << now[na][nb] << endl;
   out.close();
}

void main()
{
   init();
   start();
   print();
}
```

四、总结

在这个例子中，我们用 f 来表示目标问题。在求 f 之前，先要求出 g，而在求 g 的时候，也采用了动

态规划。我们称之为多次动态规划。很多题目中，动态规划并不是单一出现的，但有些例子中，这种多层次的结构并不明显，下一节将讨论这个问题。

交叉匹配 (WC'2001 练习题)

一、问题描述

现有两行正整数。如果第一行中有一个数和第二行中的一个数相同，都为 r，则我们可以将这两个数用线段连起来。称这条线段为 r一匹配线段。例如下图就显示了一条 3 匹配线段和一条 2 匹配线段。

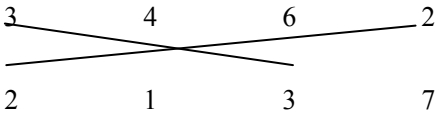


表 4-1

我们想要对于给定的输入，找到画出最多的匹配线段的方式，使得：

1. 每条 a 匹配线段恰好和一条 b 匹配线段相交，且 a≠b，a, b 指代任何值，并非特定值。
2. 不存在两条线段都从一个数出发。

写一个程序，对于给定的输入数据，计算出匹配线段的最多个数。

二、分析

这是一个多次动态规划问题。

设这两行数分别保存在 a[n]和 b[m]中。用 f(i, j)表示如下图所示的两行数据，可以画出的最多的匹配线段数。

a[1]	a[2]	…	A[i]
b[1]	b[2]	…	B[j]

表 4-2

对于这个状态的求解，可以分如下几种情况讨论：

1. 没有从 a[i]出发的匹配线段。如下图，这种情况的解即为 f(i – 1, j)

a[1]	a[2]	…	a[i – 1]	a[i]
b[1]	b[2]	…	b[j – 1]	b[j]

表 4-3

2. 没有从 b[j]出发的匹配线段。如下图，这种情况的解即为 f(i, j – 1)

a[1]	a[2]	…	a[i – 1]	a[i]
b[1]	b[2]	…	b[j – 1]	b[j]

表 4-4

3. a[i]和 b[j]处，各引出一条匹配线段，这时必须 a[i]≠b[j]。

设 a[i] = b[v]，b[j] = a[u]。从 a[i]向 b[v]，b[j]向 a[u]各引一条匹配线段，这两条线段必然相交。这种情况的解即为 f(u – 1, v – 1) + 2

a[1]	a[2]	…	a[u – 1]	a[u]	…	a[i]

b[1]	b[2]	…	b[v – 1]	b[v]	…	b[j]

表 4-5

显然，f(i, j)就是上面三种情况中的最优值。因此，我们可以列出状态转移方程：

$$f(i,j)=\max\begin{cases} f(i-1,j) \\ f(i,j-1) \\ f(u-1,v-1)+2 \quad a[i]\neq b[j] \end{cases}$$

其中

$a[i]=b[v]$且$1\le v<j$

$b[j]=a[u]$且$1\le u<i$

该算法的复杂度看上去是 $O((nm)^2)$，因为一共有 i, j, u, v 这四个变量需要循环。这个复杂度的算法在时间上是无法承受的。

从下图可以看出，如果存在 a[u’] = b[j]， 且 u < u’ < i，则用 b[j]向 a[u’]的匹配线段，代替 b[j]向 a[u]的匹配线段，所得到的解不会变差。因此，u 的选择是越靠后越好。同理，v 的选择也是越靠后越好。

a[1]	…	a[u]	…	a[u’]	…	a[i]
B[1]	…	b[v]	…			b[j]

表 4-6

由此，可以得到状态转移方程：

$$f(i,j)=\max\begin{cases} f(i-1,j) \\ f(i,j-1) \\ f(u-1,v-1)+2 \quad a[i]\neq b[j] \end{cases}$$

其中

$a[i]=b[v]$,且不存在v',使得$v<v'<j, a[i]=b[v']$

$b[j]=a[u]$,且不存在u',使得$u<u'<i, b[j]=a[u']$

我们可以看到，对于确定的 i 和 j，相应的 u 和 v 的值也是确定的，这些值可以预先求出。而求法也是利用动态规划。设相应 u 和 v 的值分别为 u[i, j]和 v[i, j]。易知，

$$u(i,j)=\begin{cases} u(i-1,j) & a[i-1]\neq b[j] \\ i-1 & a[i-1]=b[j] \end{cases}$$

$$v(i,j)=\begin{cases} v(i,j-1) & b[j-1]\neq a[i] \\ j-1 & b[j-1]=a[i] \end{cases}$$

这样，原动态转移方程可变为：

$$f(i,j)=\max\begin{cases} f(i-1,j) \\ f(i,j-1) \\ f(u(i,j)-1,v(i,j)-1)+2 \quad a[i]\neq b[j] \end{cases}$$

该算法需要保存 f，u，v 等数组，空间复杂度是 O(nm)。由规划方程可知，计算 f，u，v 的时间复杂度是 O(nm)，因此总的时间复杂度也还是 O(nm)。

在这个例子中，我们很顺利地得到了一个动态转移方程，但这个算法的时间复杂度却无法让人接受。进一步的分析表明，决策变量的取值有很强的约束条件，于是通过第二次动态规划独立的求出决策变量的取值，从而提高了算法的效率。

三、参考程序：

```
const
  max = 100;

var
  k, p, q, n, m: integer;
  u, v, f: array[- 1 .. max, - 1 .. max] of integer;
  a, b: array[0 .. max] of integer;
begin
  assign(input, 'input.txt'); reset(input);
  readln(n, m);
  for k := 1 to n do read(a[k]);
  for k := 1 to m do read(b[k]);
  close(input);
  for p := 1 to n do
    for q := 1 to m do begin
      if a[p - 1] = b[q] then u[p, q] := p - 1 else u[p, q] := u[p - 1, q];
      if a[p] = b[q - 1] then v[p, q] := q - 1 else v[p, q] := v[p, q - 1];
 END;
  for p := 1 to n do
    for q := 1 to m do begin
      if a[p] = b[q] then k := 0 else k := f[u[p, q] - 1, v[p, q] - 1] + 2;
      if f[p - 1, q] > k then k := f[p - 1, q];
      if f[p, q - 1] > k then k := f[p, q - 1];
      f[p, q] := k;
    end;
  writeln(f[n, m]);
end.
```

Codes (IOI‘99)

动态规划的时间效率一般很高，但却需要大量的空间。虽然如此，动态规划算法同样适用于数据量很大的题目。下面的这道题目就是一个例子。

一、问题描述

给定一个码字集合（set of code words），和一个文本（text），码字（code words）以一种独特方式埋藏（embed）到文本之中。

码字（code word）与文本（text）都是由大写和小写的英语字母构成的字符序列（sequence）。注意，这里的大小写是敏感的，码字的长度（length）就是它所包含的字符数目，比如码字“ALL”的长度为 3。

在给定的文本中，构成码字的字母不一定要连续出现。比如，在任何一个形如“AuLvL”的序列中，

我们说码字“ALL”在该序列中出现（occur），这里的 u 和 v 可以是任意字符序列，但也可能是空序列（即没有字母的序列），这时，我们称序列“AuLvL”是码字“ALL”的一个“覆盖序列”（covering sequence）。通常，一个码字的覆盖序列可以定义为文本的一个子序列，其首，尾字母与该码字的首，尾字母一致，并且在删除该子序列中某些字母之后，可以得到该码字，（当然，该子序列也可能就是该码字）。需要注意的是，同一个码字可能在一个或者多个覆盖序列中出现，也可能根本不出现；同时一个覆盖序列也可能同时是多个码字的覆盖序列。

文本中的所有字母由左到右从 1 开始编号，该编号就是该字母在文本中的位置，一个覆盖序列在文本中的位置，由其首字母和尾字母的位置（即起始位置和终止位置）确定，给定两个覆盖序列 c_1 和 c_2，如果 c_1 的起始位置大于（＞）c_2 的终止位置，或者 c_2 的起始位置大于（＞）c_1 的终止位置，我们就称它们是“不重叠的”（do not overlap）。否则我们称它们是“重叠的”。

为了从文本中抽取隐藏的信息，你的任务是生成一个“答案”（solution）。答案由若干“项目（item）”构成，每个项目包括一个码字，以及该码字的一个覆盖序列，同时还要满足下列条件：

所有项目中的覆盖序列互相没有重叠；

每个项目中的覆盖序列长度不超过（≤）1000；

各项目中码字的长度之和达到最大，（换而言之，每个项目对这个和的贡献是其对应码字的长度）。

注意，同一文本的“答案”可能有多个，这种情况下你只需给出其中任何一个“答案”

即可。

（2）假设条件

1≤N≤100，N 是码字的总数。

每个码字的最大长度为 100 个字符。

文本的最小长度为 1，最大长度为 1000000。

给定的文本满足下面约束条件：

对于文本中的任何位置，包含该位置的“右侧最小”覆盖序列总数不超过（≤）2500；

“右侧最小”覆盖序列的总数不超过（≤）10000。

给定一个码字 W，以及它的一个覆盖序列 C，如果 C 的任何前缀（prefix）都不是 W 的覆盖序列，那么就称 C 是 W 的“右侧最小“覆盖序列。例如，对于码字“ALL”，“AAALAL”是它的“右侧最小”覆盖序列，然而虽然“ AAALALAL”是它的一个覆盖序列，但却不是“右侧最小”覆盖序列。

（3）输入

有两个文本格式的输入文件：word.inp 和 text.inp。其中的 word.inp 记录了一个码字的列表，而 text.inp 则记录了文本。

words.inp 文件的首行是数值 N.接下来的 N 行分别为一个码字，每个码字是若干字母组成的序列，字母之间没有空格，根据其在文件 words.inp 中出现的次序，对所有码字从整数 1 到 N 进行编号。

文件 text.inp 是一个字母序列，序列最后接有一个行结束符（end-of-line）和一个文件结束符（end-of-file）。该文件不包含空格符。

对使用 PASCAL 解题者的建议出于效率上的考虑，建议将输入文件的类型声明为“text”，而不是“typed”类型。

输出

输出为一个名为 codes.out 的 text 文件。

第一行是你得到的“答案”的总和值。

接下来的各行，分别记录“答案”中的各个“项目”（item）：每行为三个整数 I，s 和 e，其中 I 是码字的编号，s 和 e 分别是该码字所对应的覆盖序列的首尾位置，除首行外，其它各行对应哪个项目无关紧要。

例子

words.inp：

```
4
RuN
RaBbit
HoBbit
Stop
```

codes.out：

```
12
2    9    21
1    4    7
1    24    28
```

text.inp：

```
StXRuYNvRuHoaBbvizXztNwRRuuNNP
```

在上例中，抽取出来的隐藏信息为“RuN　RaBbit　RuN”（如上图下划线所示）。（注意：该题还有另一个“答案”为 RuN　HoBbit　RuN）。请特别注意：上述信息并未直接呈现在输出文件中。

二、分析

题目给出了一个字码集合和一个文本，要求生成一个结果。这里，结果是由若干满足约束条件(见原题)的项构成。因此，求出结果之前，应该先找到所有的项。我们可以先对组成结果的项进行一些限制。

给定一个字码，以及它的一个覆盖序列 c，若 c 的任何连续子序列 p 都不是 w 的覆盖序列，则称 c 是 w 的最简覆盖序列。例如，’ALAL’和’AAALAL’都是’ALL’的一个覆盖序列，其中’ALAL’是’ALL’的最简覆盖序列，但’AAALAL’却不是’ALL’的最简覆盖序列。显然，一个字码的最简覆盖序列必然是它的右侧最小覆盖序列，反之则不然。若一个项的覆盖序列是其对应字码的最简覆盖序列，则称之为最简项。

设项 i 由字码 w 和覆盖序列 c 所构成，如果 c 不是 w 的最简覆盖序列，显然可以找到 c 的一个子序列 p 为 w 的最简覆盖序列。因此，若结果含有非最简项 i，我们可以将 i 用对应的最简项代替，且仍然满足约束条件，结果的总和值也不变。因此，结果总可以由若干最简项构成。这样，我们只要从文本中找到所有最简项即可。

由题设，最简项的总数不会超过 10000，因此，可以用一个长为 10000 的线性表 Items 保存找到的所有项：

```
TItem     =record(项类型)
      which     :Byte;{字码编号}
    st,ed     :Longint{覆盖序列的首尾位置}
end;
Items      :array[1..10000] of TItem;
m          :Integer;{文本中项的总数}
```

由于文本可能达到 1M，我们最好只读一遍文件便求出 Items。下面先定义一些变量：

Len[i]表示第 i 个字码的长度。

Words[i,1] .. Words[j]表示第 i 个字码的前 j 个字符所构成的子串。

Match[i,j]表示在当前已经读过的文本中，Words[i,1..j]对应的最简覆盖序列的首位置。若存在多个这样的覆盖序列，则 Match[i,j]的值为首位置最大的一个；若不存在这样的覆盖序列，则 Match[i,j]为-∞。

例如，若 Words[1]='abcd'，当前所读到的文本是'aubcavbc'，因为子串'abc'(即 Words[1,1..3])在当前所读到的文本中，最简覆盖序列有'aubc'和'avbc'，它们的首位置分别是 1 和 5，因此 Match[1,3]=5。

如果当前读到了文本的第 Index 个字符 Ch，而第 i 个字码的第 j 个字符也是 Ch，则有下面 3 种情况：

若 j=1，这时令 Match[i,1]:=Index；

若 j>1，则若 Index-Match[i,j-1]+1≤1000(项中的覆盖序列长度不超过 1000)，由于 Words[i,1..j]和 Words[i,1..j-1]在当前所读到的文本中，最简覆盖序列的最大首位置应该是相同的，因此有 Match[i,j]:=Match[i,j-1]。为了不产生非最简项，令 Match[i,j-1]:=-∞；

若 j=Len[i]且 Index-Match[i,j]+1≤1000，则产生了一个项，且该项 which:=i;st:=Match[i,j];ed:=Index。同时，令 Match[i,j]=-∞。

为了便于实现上述算法，我们建立一个字符表记录每个字符在字码集合中的位置，由于字码最多共有 100*100 个字符，因此该表的长度不超过 10000。数组 Chs 的定义如下：

```
TChar    =record
    Ch        :Char;{字符}
    i,j       :Byte;{该字符所在字码编号和在该字码中的位置}
end;
Chs      :array[1..10000] of TChar;
```

为了便于检索，我们将 Chs 以关键字 Ch 按照字典顺序排序。由于在第二种情况中，Match[i,j]利用了 Match[i,j-1]的信息。因此，若关键字 Ch 相同，则以关键字 j 按照降序排列。

设 Start[x]表示排序后 Chs 中第一个 Ch 值为 x 的元素的位置，则字符 x 在字码集合中出现的位置被记录在 Chs 的 Start[x]到 Start[Succ(x)]-1 之中。这样，找所有的项的算法为：

```
读入 Words.inp，并计算数组 Chs 和 Start
Index:=0; {文件位置指针初始化}
Match[1..100,1..100]:=-∞;{ Match 数组初始化}
While not Eoln do begin{文件没有结束}
    Read(Ch);{读入一个字符}
    Index:=Index+1;{后移文件位置指针}
    For p:=Start[Ch] to Start[Succ(Ch)]-1 do{处理 Chs 中所有 Ch 值等于 x 的元素}
        if Chs[p].j=1 then{初始}
          Match[Chs[p].i,1]:=Index;
        if Index-Match[Chs[p].i,Chs[p].j-1]+1≤1000 then{递推}
          Match[Chs[p].i,Chs[p].j]:=Match[Chs[p].i,Chs[p].j-1];
          Match[Chs[p].i,Chs[p].j-1]:=-∞
        if (Chs[p].j=Len) and (Match[Chs[p].i,Chs[p].j])>0) then {找到一个项}
          Match[Chs[p].i,Chs[p].j]:=-∞;
          NewItem.Which:=Chs[p].i;
          NewItem.st:=Match[Chs[p].i,Chs[I].j];
          NewItem.ed:=Index;
          AddItem(NewItem);{将 NewItem 添到 Items 数组末尾}
```

该算法结束后，Items 中保存了所有的项，且项根据对应的覆盖序列的尾位置按照非降序排列。

通过动态规划求出结果。

我们通过建图来进一步抽象该问题。根据求出的 Items 和 m，建立有向带权图 G=(V,E)：顶点 $V=\{V_0,V_1,\ldots,V_m,V_{m+1}\}$，其中 $V_1,V_2,\ldots,V_m$ 代表 i 个项。V_0 和 V_{m+1} 分别是源点和汇点，并设 Items[0].ed:=0;Items[m+1].st:=∞。

若 Items[i].ed<Items[j].st，则在 V_i 与 V_j 之间连一条弧，方向从 V_i 至 V_j，弧的权为 Items[j]所对应的字码的长度。其中，设 Items[0]和 Items[m+1]所对应的字码的长度均为 0。

若 V_i 到 V_j 有弧，则必有 i<j。若不然，设 i≥j 且 V_i 到 V_j 有弧，则由图 G 的定义，有 Items[i].ed<Items[j].st。而 Items 数组已经根据关键字 ed 按照非降序排列，又 i≥j，因此 Items[i].ed≥Items[j].ed≥Items[j].st，这与 Items[i].ed<Items[j].st 矛盾。

根据这个结论，易证图 G 是有向无环图。而结果可以看作是一条 V_0 到 V_{m+1} 的最长路径，路径中经过的顶点的编号即为结果包含的项的编号。因此，该问题的实质是求有向无环图的最长路径，可以采用动态规划求解。

设 F[i]表示 V_0 到 Vi 的最长路径的长度。很容易得出状态转移方程：

F[i]:=Max{F[j]}+Len[Items[i].Which]，（0≤j≤i-1 且 Vj 到 Vi 有弧）

初始条件 F[0]:=0

结果的总和值即为 F[m+1]。

该算法的时间复杂度为 $O(m^2)$，而 m 最大为 10000，将会超时。因此，必须对上述算法进行优化。

我们先看一个具体的例子。如 m=6，Items 数组内容如下表(这里略去了项所对应字码的编号)：

I	0	1	2	3	4	5	6	7
Items[i].st	0	1	2	3	6	4	8	∞
Items[i].ed	0	4	5	6	7	8	8	∞

表 4-7

相应的图 G 如图 4-6(这里略去了弧的长度)：

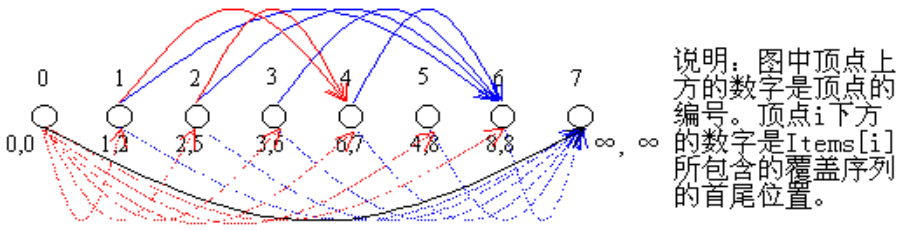


图 4-6　相对应的图 G

注意到图 G 中有一个特殊的性质：若 i<j 且 V_j 到 V_k 有弧，则 V_i 到 V_k 有弧。例如上图中，V_4 到 V_6 有弧，则 V_0,V_1,V_2,V_3 到 V_6 都有弧。

证明：由于 V_j 到 V_k 有弧，根据图 G 的定义，有 Items[j].ed<Items[k].st。又 i<j，所以 Items[i].ed≤Items[j].ed，因此 Items[i].ed<Items[j].ed<Items[k].st，这就说明 V_i 到 V_k 有弧。

设 V_i 是到 V_k 有弧且编号 i 最大的顶点(如上图中，V_2 是到 V_4 有弧且编号最大的顶点)。由上面的讨论可知，所有编号不大于 i 的顶点到 V_k 都有弧。因此，有 $F[k]=Max_{(0\le p\le i)}\{F[p]\}+Len[Items[i].Which]$。我们令 $Q[i]=Max_{(0\le p\le i)}\{F[p]\}$，代入 F 的递推关系式，得 F[k]=Q[i]+Len[Items[k].Which]。而 Q[i]可以通过递推关系式 Q[i]=Max{Q[i-1],F[i]}求出。将 F 的递推关系式代入 Q 的递推关系式，我们可以得到新的动态规划方程：

设 Q[k]表示 V_0 到 $V_p(0\le p\le k)$最长路径长度的最大值，V_i 是到 V_k 有弧且编号 i 最大的点，则:

Q[k]:=Max{Q[k-1],Q[i]+Len[Items[k].Which]}

初始条件 Q[0]:=0

结果的总和值即为 Q[m]。

该算法需要寻找 i，在一般情况下，i 与 k 的值十分接近，因此算法的效率要比一般动态规划高得多。题目中要求保存路径，我们在动态规划时记录决策值即可。具体程序如下：

```
Items[0].Ed:=0;{建立原点 S}
Q[0]:=0;Max[0]:=0;{赋初值}
for k:=1 to M do begin
  i:=k-1;while Items[k].St<=Items[i].Ed do Dec(i);{寻找 i}
  Q[k]:=Q[i]+Len[Items[k].Which];
  Max[k]:=k;{Max[k]是辅助数组，用于求决策值}
  if Q[k-1]>Q[k] then begin
    Q[k]:=Q[k-1];
    Max[k]:=Max[k-1]
  end;
  Father[k]:=Max[i]{Father[k]是 Q[k]的决策值}
end;
```

该题需找到所有的项，这与 IOI96 的 Prefix 有些类似，都是大量数据的统计问题。处理这类题目的基本原则是牺牲空间，尽量只读一次文件便完成统计工作。

另外，我们是基于图 G 与一般有向无环图在结构上的特殊性质，优化生成结果的算法，由此可以看出用有向图来分析动态规划算法的优化的好处。

三、参考程序：

```
program IOI99_Codes;

const
  MaxWords      =100;{字码的总数不超过 100}
  MaxWordLen    =100;{字码的长度不超过 100}
  MaxItems      =10000;{项的总数不超过 10000}
  MaxItemLen    =1000;{项中覆盖序列的长度不超过 1000}
  BufSize       =16384;{文件缓冲大小}
  MaxChar       =52;{52 个英文字母(包括大小写)}
  FWords        ='Words.inp';
  FText         ='Text.inp';
  FOut          ='Codes.Out';

type
  PItem         =^TItem;
  TItem         =record{项类型}
                 Which   :Byte;{字码编号}
                 Last    :Integer;{动态规划中的决策值,用于输出方案}
                 St,Ed   :Longint{覆盖序列的首位位置}
  end;

  TChar         =record{字符类型}
                 Which   :Byte;{字符所在字码的编号}
```

```
                Pos     :Byte{字符在字码中的位置}
  end;

var
  Items          :array[0..MaxItems] of PItem;{保存找到的所有项}
  M              :Integer;{项的总数}
  CharToInt      :array[Char] of Integer;{为了处理方便，将 52 个英文字符与 1..52 建立对应关系}
  NChar          :Integer;{字符总数}
  Chs            :array[1..MaxWords*MaxWordLen] of TChar;{字符表}
  Len            :array[1..MaxWords] of Integer;{Len[i]表示第 i 个字码的长度}
  Start          :array[1..MaxChar+1] of Integer;{索引表}

procedure GetWords;{读入 Words.inp}
  var
    Words        :array[1..MaxWords] of string[MaxWordLen];{字码集合}
    Count        :array[1..MaxChar] of Integer;
    N,i,j,k      :Integer;

    Ch           :Char;
  begin
    Assign(Input,FWords);Reset(Input);
    FillChar(Count,Sizeof(Count),0);
    NChar:=0;
    Readln(N);
    for i:=1 to N do begin
      Readln(Words[i]);
      Len[i]:=Length(Words[i]);
      for j:=1 to Len[i] do begin
        Ch:=Words[i,j];
        if CharToInt[Ch]=0 then begin{建立字符与数字的对应关系}
          Inc(NChar);
          CharToInt[Ch]:=NChar
        end;
        Inc(Count[CharToInt[Ch]])
      end
    end;
    Start[1]:=1;
    for i:=2 to NChar+1 do Start[i]:=Start[i-1]+Count[i-1];{计算 Start 数组}
    for i:=1 to NChar do Count[i]:=Start[i]-1;
    for i:=1 to N do{计算 Chs 数组}
      for j:=Len[i] downto 1 do begin
        Ch:=Words[i,j];
        k:=CharToInt[Ch];
        Inc(Count[k]);
        Chs[Count[k]].Which:=i;
```

```
        Chs[Count[k]].Pos:=j
      end;
    Close(Input)
  end;

procedure GetText;{读入 Text.inp,并生成所有的项}
  var
    Match          :array[1..MaxWords,0..MaxWordLen] of Longint;
    Buf            :array[1..BufSize] of Char;{文件缓冲}
    i,p,pos        :Integer;{循环变量}
    Index          :Longint;{文件位置指针}
    Ch             :Char;{当前所读到的字符}
    f              :file;{文件类型}
  begin
    for i:=1 to MaxWords do begin{初始化}
      Match[i,0]:=MaxLongint;
      for p:=1 to MaxWordLen do Match[i,p]:=-MaxItemLen
    end;
    M:=0;Index:=0;pos:=0;{初始化}
    Assign(f,FText);Reset(f,1);
    BlockRead(f,Buf,BufSize,p);{采用 BlockRead 加快文件处理的速度}
    repeat
      if pos=BufSize then begin
        BlockRead(f,Buf,BufSize,p);
        pos:=0
      end;
      Inc(pos);
      Ch:=Buf[pos];
      if Ch=#13 then Break;{读到换行符，退出处理}
      Inc(Index);
      p:=CharToInt[Ch];
      if p=0 then Continue;{字码中没有出现这个字符}
      for i:=Start[p] to Start[p+1]-1 do{Match 数组的更新}
        with Chs[i] do
          if Index-Match[Which,Pos-1]<MaxItemLen then begin
            if Pos>1 then begin
              Match[Which,Pos]:=Match[Which,Pos-1];
              Match[Which,Pos-1]:=-MaxItemLen
            end else Match[Which,1]:=Index;
            if Pos=Len[Which] then begin{产生一个项}
              Inc(M);
              New(Items[M]);
              Items[M]^.Which:=Which;
              Items[M]^.Last:=0;
              Items[M]^.St:=Match[Which,Pos];
```

```
                    Items[M]^.Ed:=Index;
                    Match[Which,Pos]:=-MaxItemLen
                  end
                end
    until False;
    Close(f)
  end;

procedure Print;{动态规划并输出}
  var
    F                :array[0..MaxItems] of LongInt;
    Max              :array[0..MaxItems] of Integer;
    i,k              :Integer;
    Start            :Longint;
  begin
    New(Items[0]);FillChar(Items[0]^,Sizeof(Items[0]^),0);
    F[0]:=0;Max[0]:=0;
    for i:=1 to M do begin{规划}
      k:=i-1;Start:=Items[i]^.St;
      while Start<=Items[k]^.Ed do Dec(k);
      Items[i]^.Last:=Max[k];
      F[i]:=F[k]+Len[Items[i]^.Which];
      Max[i]:=i;
      if F[i-1]>F[i] then begin
        F[i]:=F[i-1];
        Max[i]:=Max[i-1]
      end
    end;
    Assign(Output,Fout);ReWrite(Output);{输出}
    Writeln(F[M]);
    i:=Max[M];
    while i<>0 do
      with Items[i]^ do begin
        Writeln(Which,' ',St,' ',Ed);
        i:=Items[i]^.Last
      end;
    Close(Output)
  end;

begin
  GetWords;
  GetText;
  Print
end.
```

快乐的蜜月（CTSC 2000）

动态规划常用来计算最优解，其实，要求次优解，第三优解，甚至第 k 优，动态规划同样有它的用武之地。

一、问题描述

位于某个旅游胜地的一家宾馆里，有一个房间是总统套房。由于总统套房价格昂贵，因此常常无人光临。宾馆的经理为了创收，决定将总统套房改建为专门为新婚夫妇服务的蜜月房。宾馆经理不仅大幅度降低了蜜月房的价位，而且还对不同身份的顾客制定了不同的价位，以吸引不同身份、不同消费水平的游客。比如对于来订蜜月房的国内来宾、海外旅客、港澳台同胞等，区别收取费用。

宾馆经理的举措获得了不同凡响的效果。由于蜜月房环境幽雅，服务周到，因此生意红火。宾馆经理在每年年底都会收到第二年的所有蜜月房预订单。每张预订单包括以下几个必要的信息：到达日期、离去日期和顾客身份。

由于宾馆只有一间蜜月房，只能同时接待一对新婚夫妇。因此并不是所有的预订要求都能得到满足。当一些预订要求在时间上发生了重叠的时候，我们就称这些预订要求发生了冲突。

对于那些不与任何其他预订要求发生冲突的预订单，必然会被接受，因为这对宾馆和顾客双方面来说都是件好事。而对于发生冲突的预订要求，宾馆经理则必须拒绝其中的一部分，以保证宾馆有秩序地运转。显然，对于同一时间内发生冲突的预定要求，宾馆经理最多只能接受其中的一个。经理也有可能拒绝同一时间段内的所有预定要求，因为这样可以避免顾客间发生争执。经理在做出决策后，需要将整个计划公布于众，以示公平。这是一个必须慎重的决定，因为它牵涉到诸多方面的因素。经理首先考虑的当然是利润问题。他必然希望获得尽可能多的收入。可是宾馆在获得经济效益的同时，同时也应该兼顾到社会效益，不能太惟利是图，还必须照顾到顾客们的感情。如果宾馆经理单从最大获利角度出发来决定接受或拒绝顾客的预订要求的话，就会引起人们的不满。经理有一个学过市场营销学的顾问。顾问告诉经理，可以采取一种折中的做法，放弃牟利最大的方案，而采纳获利第 k 大的方案。他还通过精确的市场分析，找到了 k 的最佳取值点，告诉了宾馆经理。

现在请你帮助宾馆经理，从一大堆预订要求中，在上述原则下寻找到获利第 k 大的方案。宾馆经理将根据此方案来决定接受和拒绝哪些预订要求。

当然，可能有若干种方案的获利是一样大的。这时候，它们同属于获利第 i 大的方案而不区分看待。例如，假如有 3 种方案的收入同时为 3，有 2 种方案的收入为 2，则收入为 3 的方案都属于获利最大，收入为 2 的方案都属于获利第二大。依次类推。

假设所有的住、离店登记都在中午 12 点进行。

输入文件的第一行是两个数，k（1<=k<=100）和 t(1<=t<=100)。其中 k 表示需要选择获利第 k 大的方案；t 表示顾客的身份共划分为 t 类。

第二行是一个数 y，表示下一年的年份。

第三行是一个数 r(0<=r<=20000)，表示共有 r 个预订要求。

以下 r 行每行是一个预订要求，格式为：

m1/d1 TO m2/d2 id;

其中 m1/d1 和 m2/d2 分别表示到达和离去日期。id 是一个整数（1<=id<=t），用来标识预订顾客的身份。

最后 t 行每行为一个整数 P_i(1<=i<=t，1<=P_i<=32767)，表示蜜月房对于身份代号为 i 的顾客的日收费标准。

例：某对顾客于 6 月 1 日到达，6 月 3 日离去，对他们的日收费标准为 m 元/天，则他们共住店两天，需付钱 2m 元。

输出文件仅包含一个整数 p，表示在获利第 k 大的方案下，宾馆的年度总收入额。如果获利第 k 大的方案不存在，则输出-1。

二、分析

我们可以把一个预定要求看作数轴上的一条线段。线段的两个端点分别是该线段所代表的预定要求的起始时间和结束时间，不同的预定要求有不同的价格，因此，我们还要给每条线段赋一个权值，它表示该预定要求的收益。这样，一种方案就可以看成是一组没有重叠的线段的集合，一种方案的总收益就是这些被选线段的权值和。题目所要求的就是第 k 大的权值和。

这道题目要求第 k 优值，下面我们就先来分析 k = 1 的情况。

当 k = 1 时，实际上就是要求最优值。从问题抽象后的模型来看，k=1 的情形十分类似于 IOI99 的 Codes。从问题的本质上来说，这两道题目完全相同：它们都是要求一个集合，集合中的元素可以用数轴上一条带权值的线段表示，要求集合中的线段互不重叠，且线段权值和最大。从问题的规模上来看，两道题目略有不同：Codes 线段的总数在 10000 以内，线段端点的取值可达 1000000。而本题线段的总数可达到 20000，线段端点最多在 366 以内。

Codes 我们采用的是动态规划，这道题目当然也可以同样的方法解决。在 Codes 中，我们是按线段来划分阶段的，而这道题目线段数比 Codes 多了一倍，高达 20000，而线段的端点取值范围却远远小于 Codes，

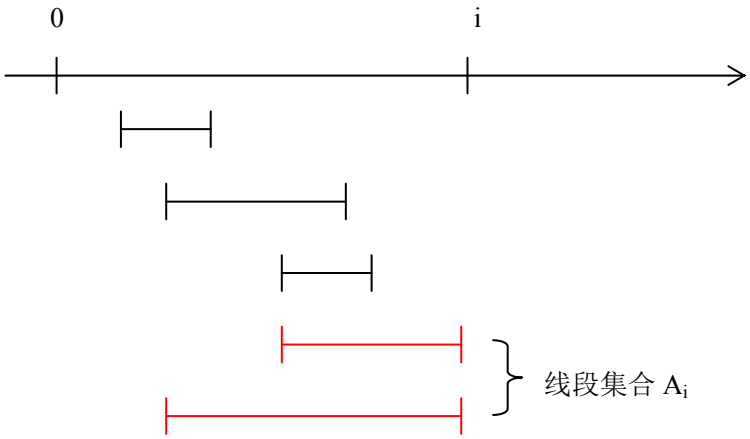


图 4-7 线段集合

只有 366，因此我们可以按线段的端点来划分阶段。如上图，我们用 F[i]表示前 i 天可以达到的最大收益，用集合 A_i 表示右端点在第 i 天的线段的集合，则 F[i]的值或者不选 A_i 中的任意一条线段，即 F[i] = F[i - 1]，或者选择 Ai 中的某一条线段（这是因为 A_i 中的线段右端点相同，所以不可能从中选择多条线段）。不妨设选择线段 x，且 x 的左端点为 s，权值为 v。显然，当 F[i]最优时，前 s 天的收益也必然达到最大。因此，这时有 F[i] = F[s] + v。

这样，我们就得到了状态转移方程：

$F[i] = \min\{F[i-1], \min\{F[x.s] + x.v\}\}$

其中线段 $x \in A_i$，x.s 表示线段的左端点，x.v 表示线段的权值。

下面来看 k≠1 的情况。容易看出，前 i 天收益第 k 大的方案中，前 j 天(j < i)的收益也必然是前 k 大的。因此，k≠1 的情况和 k = 1 的情况可以类似的处理：如果我们要求最后的结果为第 k 大，只需在每一个阶段中都保留前 k 大的结果就可以了。即用 F[i, j]表示前 i 天可以达到的第 j 大收益。在计算 F[i]的过程中，每次处理一条属于 A_i 的线段 x 时，可以建立一个临时数组 Tmp，其中 Tmp[p]表示如果选择待处理的线段 x，前 i 天可以达到的第 p 大的效益，显然我们有 Tmp[p] = F[x.s, p] + x.v (1≤p≤K)。再将 Tmp 与 F[i - 1]归并取前 k 大即可。

下面分析算法的复杂度。空间上，F 数组的每个元素都是长整型的，因此要 367 * 100 * 4 = 144K 的空间，另外线段集合 A 中最多有 20000 个元素，每个元素大约 8 个字节，因此一共需要空间 20000 * 8 = 156K。因此，本题一共需要空间大约 144 + 156 = 300K。完全可以承受。

时间上，虽然我们是按天来划分阶段，但实际上对每条线段都处理了一次，而对于每一条线段的处理，实际上是一个求 Tmp 数组和归并排序的过程，这两个算法的时间复杂度都是 k，因此，总时间复杂度为 20000

* 100 = 200 万。也是可以承受的。

三、参考程序：

```
#include <stdio.h>

const char * Finp = "honey.in";
const char * Fout = "honey.out";
const MaxDays = 366;
const MaxN = 100;

struct TNode
{
  int x, id;
  TNode * Next;
};

TNode * Data[MaxDays + 1];
long * F[MaxDays + 1];
int Count[MaxDays + 1];
long Cost[MaxDays + 1];
int k;

void Insert(int st, int ed, int id)
{
  TNode * Tmp = new TNode;
  Tmp -> x = st;
  Tmp -> id = id;
  Tmp -> Next = Data[ed];
  Data[ed] = Tmp;
}

void GetInfo(void)
{
  FILE * f;
  int n, x, i, st, ed, m, d, _m, _d, id;
  int Days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  int Date[12][31];

  f = fopen(Finp, "r");
  fscanf(f, "%d %d \n", &k, &n);
  fscanf(f, "%d \n", &x);
  Days[1] += ((x % 4 == 0) && (x % 100 != 0) || (x % 400 == 0));
  x = 0;
  for (m = 0; m < 12; m ++) for (d = 0; d < Days[m]; d ++) Date[m][d] = ++ x;
  fscanf(f, "%d \n", &x);
```

```
    for (i = 0; i < x; i ++)
    {
      fscanf(f, "%d / %d TO %d / %d %d \n", &m, &d, &_m, &_d, &id);
      st = Date[m - 1][d - 1];
      ed = Date[_m - 1][_d - 1];
      st --;
      ed --;
      Insert(st, ed, id);
    }
    for (i = 1; i <= n; i ++) fscanf(f, "%d \n", &Cost[i]);
    fclose(f);
}

void Print(void)
{
    FILE * f;
    long Result;
    Result = Count[MaxDays] >= k ? F[MaxDays][k] : - 1;
    f = fopen(Fout, "w");
    fprintf(f, "%ld \n", Result);
    fclose(f);
}

void Make(void)
{
    int i, p, q, l, lp, lq, pp;
    long v, x, y;
    long * Tmp, * pa, * pb, * pr;
    long Result[MaxN + 1];
    TNode * point;

    for (i = 0; i <= MaxDays; i ++) F[i] = new long[MaxN + 1];
    F[0][1] = 0;
    Count[0] = 1;
    for (i = 1; i <= MaxDays; i ++)
    {
      Count[i] = Count[i - 1];
      Tmp = F[i];
      for (pp = 1; pp <= Count[i]; pp ++) Tmp[pp] = F[i - 1][pp];
      point = Data[i];
      while (point)
      {
        v = Cost[point -> id] * (i - point -> x);
        p = 1;
        lp = Count[point -> x];
```

```
    pa = &F[point -> x][p];
    q = 1;
    lq = Count[i];
    pb = &F[i][q];
    l = 0;
    pr = Result;
    while ((p <= lp) && (q <= lq) && (l < k))
    {
l ++;
pr ++;
x = * pa + v;
y = * pb;
if (x > y)
{
    * pr = x;
    p ++;
    pa ++;
} else
{
    * pr = y;
    q ++;
    pb ++;
    if (x == y)
    {
        p ++;
        pa ++;
    }
}
    }
    while ((p <= lp) && (l < k))
    {
l ++;
pr ++;
* pr = * pa + v;
p ++;
pa ++;
    }
    while ((q <= lq) && (l < k))
    {
l ++;
pr ++;
* pr = * pb;
q ++;
pb ++;
    }
```

```
            Count[i] = l;
            for (pp = 1; pp <= l; pp ++) Tmp[pp] = Result[pp];
            point = point -> Next;
          }
        }
    }

    void Done(void)
    {
      for (int i = 0; i <= MaxDays; i ++)
      {
        delete F[i];
        TNode * p = Data[i], * q;
        while (p)
        {
          q = p -> Next;
          delete p;
          p = q;
        }
      }
  }

    void main(void)
    {
      GetInfo();
      Make();
      Print();
      Done();
    }
```

Integer (HNOI 2000)

动态规划不光可以应用在最优化问题中，动态规划中流露出来的递归和分类的思想，同样可以用于组合记数的问题当中。例如下面的这个问题：

一、问题描述

整数分划。把一个正整数 N 表示成如下表达式的一系列正整数的和，叫做 N 的一个分划。
某个正整数 N 的不同表达式的个数称做整数 N 的分划数。编程计算指定正整数的分划数。

$$N = n_1 + n_2 + ... + n_k$$
$$n_i \geq 1, i = 1,2,....., k, k \geq 1$$
$$1 \leq n_1 \leq n_2 \leq ..... \leq n_k$$

输入：输入文件仅为一行，是指定的正整数 N，N<100。
输出：输出文件为一行，输出指定的正整数 N 的分划数。
输入输出示例：
INPUT.TXT

5

OUTPUT.TXT

7

二、分析

这道题可以采用动态规划或使用母函数求解。为了理清思路，我们先看一看 5 的分划。

5=1+1+1+1+1=1+1+1+2=1+1+3=1+2+2=1+4=2+3=5

组合计数中常常用到的方法是分类计数，再通过乘法原理或加法原理求出最后的结果，而动态规划的关键也在于分类。对于分划方案的统计，通过不同的分类可以得到不同的统计方法。

方法 1：

我们把 5 的分划按长度分成 5 类

5=1+1+1+1+1

5=1+1+1+2

5=1+1+3=1+2+2

5=1+4=2+3

5=5

定理 1：设将 n 分成 k 个数的和的方案数为 F(n,k)，F(n,k)满足递推关系

F(n+k,k)=F(n,1)+F(n,2)+…+F(n,k)

其中：F(n,1)=1，F(n,n)=1。

证明：我们考虑 n 分成至多 k 个部分的和的方案总数为

F(n,1)+F(n,2)+…+F(n,k)。

n 分成至多 k 个部分的和的方案可表示为 n=0+0+…+0+x1+x2+…+xm，这个和式包含 k 项，且 1≤x1≤x2≤…≤xm(1≤m≤k)。我们将其映射到 n+k 分成至多 k 个部分的和的方案。

n+k=1+1+…+1+(x1+1)+(x2+1)+…+(xm+1)。

易证，这是一个一一映射。于是定理 1 成立。

正整数 n 的分划数=F(n,1)+F(n,2)+…+F(n,n)。因此，我们只要按照定理 1 的递推关系式求出 F，再输出 F(n,1)+F(n,2)+…+F(n,n)即可。

方法 2：

按分划方案中最大数的值分 5 类

5=1+1+1+1+1

5=1+1+1+2=1+2+2

5=1+1+3=2+3

5=1+4

5=5

第一类可以看作是将 4 分成最大数不大于 1 的方案数。

第三类可以看作是将 2 分成最大数不大于 3 的方案数。

若设 F(n,k)表示将 n 分成若干个不大于 k 的数的和的方案数，有

F(n,k)=F(n-1,1)+F(n-2,2)+…+F(n-k,k)

正整数 n 的分划数=F(n,n)。因此，我们只要按照递推关系式求出 F，再输出 F(n,n)即可。

方法 3：

将 5 的分划方案写成乘法形式：

5=1*5=1*3+2=1*2+3=1+2*2=1+4=2+3=5

我们联想到 $x^5=(x)^5=(x)^3(x^2)=(x)^2(x^3)=(x)(x^2)^2=(x)(x^4)=(x^2)(x^3)=(x^5)$

考虑 $(1+(x)+(x)^2+\ldots)(1+(x^2)+(x^2)^2+\ldots)\ldots(1+(x^n)+(x^n)^2+\ldots)$ 展开后中 x^n 的系数，即为所求。

以上的方法的复杂度均为 $O(N^3)$。

三、参考程序

```
const
  InputFileName = 'Integer.In';
  OutputFileName = 'Integer.Out';
var
  f: array [0 .. 100, 0 .. 100] of Longint;
  n: Integer;
  TestFile: Text;
procedure Init;{读入数据}
begin
  Assign(TestFile, InputFileName);
  Reset(TestFile);
  Readln(TestFile, n);
  Close(TestFile);
end;
procedure Main;{主过程}
var
  i, j, k: Byte;
begin
  f[0, 1] := 1;
  for i := 1 to n do
    for j := 1 to i do
      for k := 1 to j do Inc(f[i, j], f[i - j, k]);{递推求解}
end;
procedure Show;{输出}
var
  i: Byte;
  Answer: Longint;
begin
  Answer := 0;
  for i := 1 to n do Inc(Answer, f[n, i]);{计算结果}
  Assign(TestFile, OutputFileName);
  Rewrite(TestFile);
  Writeln(TestFile, Answer);
  Close(TestFile);
end;
begin    {主程序}
  Init;
  Main;
  Show;
end.
```

Bar

动态规划和搜索有着千丝万缕的关系，我们先来看一个例子：

一、问题描述

“条形码”是一种由亮条（light bar）和暗条（dark bar）交替出现，且以暗条起头的符号。每个“条”（bar）都是若干个单位宽。图 1 给出了一个含 4 个“条”的“条形码”，它延续了 1+2+3+1=7 个单位宽。

一般情况下，BC(n,k,m)是一个包含所有由：k 个“条”，总宽度正好为 n 个单位，每个“条”的宽度至多为 m 个单位的性质的“条形码”组成的集合。例如：图 1 的条形码属于 BC(7,4,3),而不属于 BC(7,4,2)。

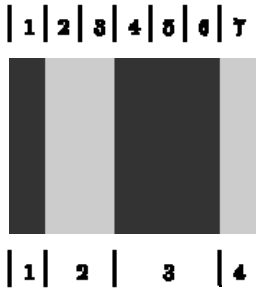


图 1

0: 1000100 | 8: 1100100
1: 1000110 | 9: 1100110
2: 1001000 | 10: 1101000
3: 1001100 | 11: 1101100
4: 1001110 | 12: 1101110
5: 1011000 | 13: 1110010
6: 1011100 | 14: 1110100
7: 1100010 | 15: 1110110

图 2

图 2 显示了集合 BC(7,4,3)中的所有 16 个符号。1 表示暗，0 表示亮。图中的条形码已按字典顺序排列。冒号左边的数字为“条形码”的编号。图 1 中条形码的在 BC(7,4,3)中的编号为 4。

输入：输入文件的第一行为数字 n，k，m（1<=n,k,m<=30）。第二行为数字 s （s<=100）。而后 s 行中，每行为一个如图 1 那样描述的集合 BC(n,k,m)中的一个“条形码”。

输出：在输出文件中第一行为 BC(n.k,m)中“条形码”的个数。而后 s 行中每一行为输入文件中对应“条形码”的编号。

输入输出示例：

Input.txt	Output.txt
7 4 3 5 1001110 1110110 1001100 1001110 1000100	16 4 15 3 4 0

二、分析

题目有两问。容易看出，计数是求序号的基础，因此我们先解决计数问题。原题只给了一个实例，即条形码。为了能用计算机解决该问题，我们必须先建立一个能够很好描述该问题的数学模型。

由于条形码是由黑白相间的且以黑色起头的 k 块组成，每一块最少含有 1 条，最多含有 m 条，k 块合起来为 n 条。因此，一个条形码可以由一个 k 元组(x1,x2,…,xk)表示，且 1≤xi≤m，$\sum(i=1..k)xi=n$。相应地，任意一个满足上述条件的 k 元组唯一表示一个满足条件的条形码。容易证明，所设的 k 元组与条形码满足一一对应的关系。满足条件的条形码的个数即为所设的 k 元组的个数。即方程$\sum(i=1..k)xi=n$，1≤xi≤m 的整数解的个数。

最容易想到的是搜索算法 1：由于 xi 的取值范围已经确定，我们可以穷举所有的 xi 的取值，再检查有多少组解满足$\sum(i=1..k)xi=n$。程序很容易编写，但复杂度却很高,为 m^k，由于 m，k 都可能达到 30，因此

该算法是很低效的。

搜索算法低效的原因是没有很好的利用∑(i=1..k)xi=n 这个约束条件，而只将其作为一个判定条件。最容易想到的改进策略是：如要求方程 x1+x2+x3=4，1≤x1,x2,x3≤2 的整数解的个数。若 x1=1，则方程化为 x2+x3=4-x1=4-1=3，若 x1=2 方程化为 x2+x3=2。原方程的整数解的个数，正是方程 x2+x3=3，x2+x3=2 的整数解的个数的和。这样，我们把含 3 个未知数的方程的整数解个数的问题化为了若干含 2 个未知数的方程的整数解个数的问题。以此类推，最终可以化为求含一个未知数的方程的整数解个数的问题。容易得出，方程 x=n，1≤x≤m 的整数解的个数为：当 1≤n≤m 时，有 1 个解，否则，有 0 个解。

容易写出搜索算法 2：

```
Func Count(k,n):LongInt;{∑(i=1..k)xi=n，1≤xi≤m 的整数解个数}
  begin
    if k=1 then if 1≤n≤m then Count:=1
                          else Count:=0
       else for i:=1 to m do Inc(Count,Count(k-1,n-i)){***}
  end;
```

该程序的时间复杂度仍然为 m^k ，但我们可以通过改进{***}句而将复杂度降低到 O(N)，其中 N 为 Count 函数的返回值，即方程的整数解个数。具体方法是通过修改循环语句的起始值和终止值：for i:=MinI to MaxI do Inc(Count,Count(k-1,n-i));其中，MinI=Max{1,n-m*(k-1)}，MaxI=Min{m,n-k+1}。而方程的整数解个数可以达到 $6*10^8$ 甚至更高。显然，这个程序是不高效的。其原因在于所建立的数学模型的抽象程度不高。

我们将方程的整数解个数的模型进一步抽象为：k 个在 1..m 之间的数的和为 n 的方案数。新的模型与方程的整数解个数的模型似乎没有不同，仅仅是将原模型中未知数 xi 抽象为 k 个数。但事实上，这个并不大的变化可以很大程度上的优化程序：我们用 F(k,n)表示 k 个在 1..m 之间的数的和为 n 的方案数，显然有方程式

F(k,n)=∑(i=1..m)F(k-1,n-i)。

和初始条件：若 i≠0 则 F(0,i)=0，F(0,0)=1。

容易写出动态规划程序：

```
Proc Count;
  begin
    FillChar (F, Sizeof (F), 0);
    F [0,0]: =1;
    for i:=1 to n do
      for j:=1 to k do
        for p:=1 to m do
          if i>=p then Inc(F[i,j],F[i-p,j-1])
  end;
```

动态规划的程序的时间复杂度为 O(n*k*m)≤30*30*30=27000。

动态规划的特点之一是速度快，另一点便是丰富了运算结果。如本题，我们不仅计算出题设条形码的个数，还计算出了所有由 i 块组成，每一块最少含有 1 条，最多含有 m 条，i 块一共为 j 条的条形码的个数(1≤i≤k，1≤j≤n)。而这些信息可以很方便的解决本题的第二问。

要计算一个条形码的编号，可以先统计在字典顺序中比该条形码小的条形码的个数。这是很容易做到的。具体程序如下：

```
Func Index (n, k, p: Integer): LongInt;
  begin
    if k<=1 then Index:=1 else begin
      x:=0;
      if Odd(p) then begin q:=1;Delta:=1 end else begin q:=m;Delta:=-1 end;
```

```
        while l[p]<>q do begin
          if n>=q then Inc(x,F[n-q,k-1]);
          Inc (q,Delta)
        end;
        Inc (x, Index (n-q, k-1, p+1));
        Index: =x
      end
    end;
```

其中 L 数组存放的是所读入的条形码的所对应的 k 元组。如条形码 1001110 对应的 L 数组为 L[1]=1,L[2]=2,L[3]=3,L[4]=1。该过程的时间复杂度为 O(n*k)≤30*30=900

再得到了完美的解答之后，我们再来看看前面的搜索算法。容易看出，搜索算法 2 可改为：

```
Func Count(k,n):LongInt;{∑(i=1..k)xi=n，1≤xi≤m 的整数解个数}
  begin
    if F[k,n]=NULL then
      if k=0 then F[k,n]:=1
            else for i:= Max{1,n-m*(k-1)} to Min{m,n-k+1} do
                 Inc (F [k, n], Count (k-1, n-i));
    Count: =F [k, n]
  end;
```

改进后的算法即为动态规划的递归式写法。此算法可以看作是动态规划算法的改进。因为对决策变量 i 的初始值和终止值的修正，使得其计算次数较递推写法的动态规划更少。也就是说，我们通过对搜索的算法的改进，得到了同样的动态规划的算法。

那么动态规划与搜索的关系究竟是什么呢，我们再来看另外一个问题：

序关系计数问题 (福建试题)

一、问题描述

用关系‘<’和‘=’将 3 个数 A、B 和 C 依次排列有 13 种不同的关系：

A<B<C, A<B=C, A<C<B, A=B<C, A=B=C, A=C<B,

B<A<C, B<A=C, B<C<A, B=C<A,

C<A<B, C<A=B, C<B<A。

编程求出 N 个数依序排列时有多少种关系。

二、分析

〈1〉. 枚举出所有的序关系表达式

我们可以采用回溯法枚举出所有的序关系表达式。N 个数的序关系表达式，是通过 N 个大写字母和连接各字母的 N-1 个关系符号构成。依次枚举每个位置上的大写字母和关系符号，直到确定一个序关系表达式为止。

由于类似于‘A=B’和‘B=A’的序关系表达式是等价的，为此，规定等号前面的大写字母在 ASCII 表中的序号，必须比等号后面的字母序号小。基于这个思想，我们很容易写出解这道题目的回溯算法。

算法 1，计算 N 个数的序关系数。

```
procedure Count(Step,First,Can);
{Step 表示当前确定第 Step 个大写字母；
 First 表示当前大写字母可能取到的最小值；
```

```
Can 是一个集合，集合中的元素是还可以使用的大写字母}
 begin
   if Step=N then begin{确定最后一个字母}
     for i:=First to N do if i in Can then Inc(Total);{Total 为统计的结果}
     Exit
   end;
   for i:=First to N do{枚举当前的大写字母}
     if i in Can then begin{i 可以使用}
       Count(Step+1,i+1,Can-[i]);{添等号}
       Count(Step+1,1,Can-[i]){添小于号}
     end
 end;
```

调用 Count(1,1,[1..N])后，Total 的值就是结果。该算法的时间复杂度是 O(N!)

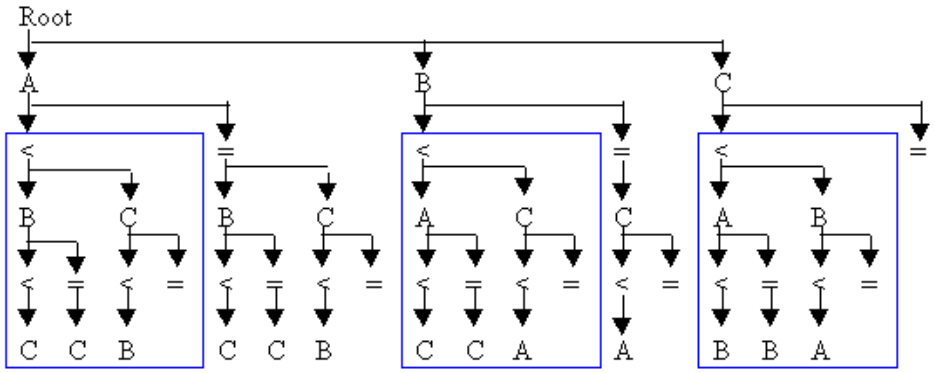


图 4-8 N=3 时的解答树

〈2〉. 粗略利用信息，优化算法 1

算法 1 中存在大量冗余运算。如图 4-8，三个方框内子树的形态是完全一样的。一旦我们知道了其中某一个方框内所产生的序关系数，就可以利用这个信息，直接得到另两个方框内将要产生的序关系数。

显然，在枚举的过程中，若已经确定了前 k 个数，并且下一个关系符号是小于号，这时所能产生的序关系数就是剩下的 N-k 个数所能产生的序关系数。

设 i 个数共有 F[i]种不同的序关系，那么，由上面的讨论可知，在算法 1 中，调用一次 Count(Step+1,1,Can-[i])之后，Total 的增量应该是 F[N-Step]。这个值可以在第一次调用 Count(Step+1,1,Can-[i])时求出。而一旦知道了 F[N-Step]的值，就可以用 Total:=Total+F[N-Step] 代替调用 Count(Step+1,1,Can-[i])。这样，我们可以得到改进后的算法 1-2。

算法 2，计算 N 个数的序关系数。

```
procedure Count(Step,First,Can);
{Step,First,Can 的含义同算法 1}
begin
   if Step=N then begin{确定最后一个字母}
     for i:=First to N do if i in Can then Inc(Total);{Total 为统计的结果}
     Exit
   end;
   for i:=First to N do{枚举当前的大写字母}
     if i in Can then begin{i 可以使用}
       Count(Step+1,i+1,Can-[i]);{添等于号}
```

```
        if F[N-Step]=-1 then begin{第一次调用}
          F[N-Step]:=Total;
          Count(Step+1,1,Can-[i]);{添小于号}
          F[N-Step]:=Total-F[N-Step]{F[N-Step]=Total 的增量}
        end else Total:=Total+F[N-Step]{F[N-Step]已经求出}
      end
    end;
```

开始，将 F[0],F[1],…,F[N-1]初始化为-1

调用 Count(1,1,[1..N])之后，Total 的值就是结果

算法 2 与算法 1 的差别仅限于程序中的粗体部分。

算法 2 就是利用了 F[0],F[1],…,F[N-1]的值，使得在确定添小于号以后，能够避免多余的搜索，尽快地求出所需要的方案数。该算法实质上就是自顶向下记忆化方式的搜索，它的时间复杂度为 $O(2^N)$。同算法 1 相比，效率虽然有所提高，但仍不够理想。

<3>.充分利用信息，进一步优化算法 2

在搜索的过程中，如果确定在第 k 个大写字母之后添加第一个小于号，则可得到下面两条信息：

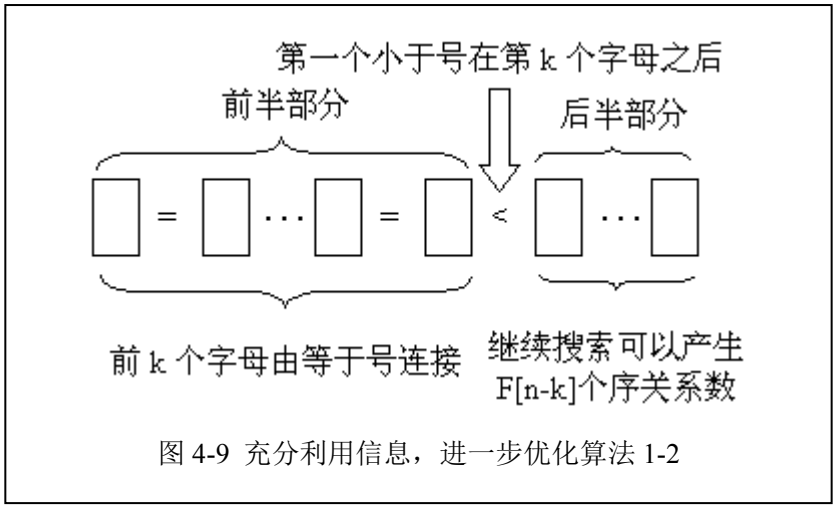


图 4-9 充分利用信息，进一步优化算法 1-2

第一条信息：前 k 个大写字母都是用等号连接的。

第二条信息：在此基础上继续搜索，将产生 F[N-k]个序关系表达式。

如图 4-9 所示，序关系表达式中第一个小于号将整个表达式分成了两个部分。由乘法原理易知，图 4-9 所示的序关系表达式的总数，就是图中前半部分所能产生的序关系数，乘以后半部分所能产生的序关系数。算法 2 实质上利用了第二条信息，直接得到图中后半部分将产生 F[n-k]个序关系数，并通过搜索得到前半部分将产生的序关系数。但如果我们利用第一条信息，就可以推知图中前半部分将产生的序关系数，就是 N 个物体中取 k 个的组合数，即 C_n^k。这样，我们可以得到 F[n] 的递推关系式：

公式1：$F[n]=\sum_{k=1}^{n} C_n^k F[n-k]$，其中 $F[0]=1$

采用公式 1 计算 F[n]的算法记为算法 3，它的时间复杂度是 $O(N^2)$。

<4>.小结

下面是三个算法的性能分析表：

分析项目		算法 1	算法 2	算法 3
理论分析	时间复杂度	O(N!)	$O(2^N)$	$O(N^2)$
	空间复杂度	O(1)	O(N)	O(N)

实际运行情况	N=7	1s	<0.05s	<0.05s
	N=8	10s	<0.05s	<0.05s
	N=15	>10s	0.5s	<0.05s
	N=17	>10s	2s	<0.05s

表 4-8

在优化算法 1 的过程中，我们通过利用 F[0],F[1]…,F[N-1]的信息，得到算法 1-2，时间复杂度也从 O(N!)降到 $O(2^N)$。在算法 2 中，进一步消除冗余运算，就得到了 $O(N^2)$的算法 3。

算法 3 计算 F[n]，体现了动态规划的思想。也就是说，我们通过充分利用信息，提高回溯法的效率，实质上是将搜索转化成了动态规划。

Chain

前面分析 Codes 问题时，已经提到了这个优化思想。下面我们从有向无环图的性质入手，进行动态规划。

一、问题描述

一个词是由至少一个至多 75 个小写英文字母(a,b,…,z)组成的。当在一张一个或多个词组成的表(list)中，每一个词(除第一个)都能由在其前一个词的词尾添加一个或多个字母而得到的话，则称此表为一个词链。

例如:表

i

in

int

integer

为一个含四个词的链,而表

input

integer

不是链.注意:所有仅含一个词的表都是链

输入：你将从文件中读到一张表.。 表中至少有 1 个词,而所有词所含字母个数之总和不超过 2000000 个。文件以“.”结束。其余各行每行含一个词。各行已按字典顺序由小到大排序。文件中的词不会有重复。

输出：一个链的长度是指该链所含词的个数。你的程序应从输入文件中找出最长的链，并把该链写入文件中去。一个词占一行。若有多个最长链，只须输出其中任何一个。

输入输出举例；

下图给出了一个由 7 歌词的输入文件和一个对应的输出文件。输入文件中有一个长度为 4 的链，这里已没有比它更长的链了。

Input.Txt	Output.txt
I If In Input Integer Output .	i in int integer

二、分析

容易看出，这道题可以使用动态规划来解决。但由于题目中的数据可能很大(2M)，无论在时间上还是空间上，简单的动态规划都无法满足要求。因此，必须根据题目的特点和本题动态规划的特殊模型，对动态规划算法进行改进。

我们先看看一般的动态规划的模型：将每一个读入的单词看成一个结点。若单词 i 是单词 j 的前缀并且 i≠j（即单词 j 可以由单词 i 的词尾添上一个或多个字母组成），则在 i，j 之间连一条有向边，方向由 i 至 j。再在图中添加一个始点和终点，始点到任何单词有一条有向边，任何单词到终点有一条有向边，且所有边的权为 1。 则题目即要求从始点到终点的一条最长路径。易证上图是有向无环图。因此，求最长路径可以用动态规划。

由于输入文件已经按照字典顺序排序，我们可以先按照单词读入的次序给单词编号。即第一个单词编号为 1，第二个单词编号为 2，以此类推，且始点编号为 0。为了便于叙述，设编号为 i 的单词为 Wi，且记由始点到 Wi 的最长路径的长度为 Li。

这时，动态规划方程为：

初始：L0=0。

方程：Li=Max{Lj+1|Wj 是 Wi 的前缀，j=0..单词总数}。

目标：Result=Max{Li|i=1..单词总数}。

算法：

```
for i:=1 to n do
   for j:=1 to n do
      if Wi 是 Wj 的前缀 and Li<Lj+1 then Li:=Lj+1
```

空间复杂度：n=2M=$2*10^6$。

时间复杂度：$n^2=4*10^{12}$。

为了优化方程，首先引出以下定理：

定理 1：若 Wi 是 Wj 的前缀，且 i≠j 则 i<j。

证明：根据字典顺序的定义易证（此略）。

根据定理 1，原动态规划方程可改进为：Li=Max{Lj+1| Wj 是 Wi 的前缀，j=0..i-1}。

定理 2：若 Wi 是 Wk 的前缀，Wj 是 Wk 的前缀，且 j>i，则 Wi 是 Wj 的前缀。

证明：根据前缀的定义易得，Wi，Wj 都是 Wk 尾部删去某些字符而成，j>i 则说明 Wj 的长度比 Wi 的大，故 Wi 是 Wj 的前缀。

定理 3：若 Wi 是 Wk 的前缀，且不存在 j，j>i 且 Wj 是 Wk 的前缀，则 Lk=Li+1。

证明：若 Lk=Lj+1，且 Wj 是 Wk 的前缀，j<i，则由定理 2，Wj 是 Wi 的前缀。所以 Li>Lj，因此，Li+1>Lj+1=Lk，这与 Lk=Max{Li+1| Wi 是 Wk 的前缀，i=0..k-1}矛盾。

定理 3 说明，原动态规划方程可改进为：Li=Lj+1，j 为满足 Wj 是 Wi 的最长前缀。

定理 4：若 i<j<k，且 Wi 不是 Wj 的前缀，则 Wi 不是 Wk 的前缀。

证明：若 Wi 是 Wk 的前缀，根据字典顺序的定义，因为 Wi<Wj<Wk，所以 Wi 是 Wj 的前缀，这与前提 Wi 不是 Wj 的前缀相矛盾。因此，Wi 不是 Wk 的前缀。

由定理 1 和定理 4 可得定理 5：Wk 的前缀必定是 Wk-1 的前缀，或 Wk-1。

综合定理 3 和定理 5，我们可以得出优化后的动态规划方程：

若 W_{i-1} 是 W_i 的前缀，则 $L_i=L_{i-1}+1$，否则 $L_i=L_j+1$，j 满足 W_j 是 W_{i-1} 的最长的前缀。

由于一个单词的长度最大为 75 个字符，因此前缀最多只可能有 75 个。这时，可以建立一个可能前缀集合来完成整个规划过程。具体算法如下：

```
begin
   可能前缀集合:={W0};
   L0=0;
   while not 结束 do begin
      Readln (Wi);
```

```
    j:=Select; {j 满足 Wj 是可能前缀集合中最长的 Wi 的前缀}
    Li=Lj+1;
    删除可能前缀集合中所有不是 Wi 前缀的单词;
    在可能前缀集合中添入 Wi
  end
end.
```

改进后的算法时空复杂度都大大的降低。但在实现的过程当中，仍然存在优化的余地。特别是对于“可能前缀集合”的数据结构的设计上。若简单的定义数组=array[1..75] of string[75];无论是添加，删除元素还是寻找最长的前缀都不方便，空间也很浪费。由定理 2，“可能前缀集合”中任意两个不同的单词都存在前缀关系，任意一个非最长的单词都是最长单词的前缀。因此，我们只要保存最长的单词（即 Wi），就可以通过标志分点的方法来表示“可能前缀集合”了。这时，时间复杂度降为 O(N)，N 为文件的的大小≤10^6，空间复杂度降为 O(75)=O(1)。

由于题目要求输出所有解，因此我们可以采取两遍读文件的方法解决。第一遍找到最长词链的长度，第二遍再输出所有解。也可以使用 Rewrite 过程解决这个问题。具体程序见算法部分。

三、参考程序

```
TNode=record
  Ch: Char; L: Integer
end;

TSet=array[0..75] of TNode;{可能前缀集合}

begin
  l=0;{可能前缀集合中单词的最大长度}
  MaxL=0;{最长的词链的长度}
  Readln(Str);{读入一个单词}
  while Str≠'.' do begin{没有结束}
    i:=1;
    if l>Length(Str) then l:=Length(Str);
    while (i≤l) and (Str[i]=List[i].Ch) do Inc(i);
    x:=List[i-1].L;
    while i≤Length(Str) do begin{将 Str 加入可能前缀集合中}
      List [i]. Ch: =Str [i]; List [i]. L: =x; Inc (i)
    end;
    l:=i-1;List[l].L:=x+1;{优化后的动态规划方程}
    if List[l].L>MaxL then begin{找到更长的词链}
      ReWrite (Output);
      输出词链
    end else if List[l].L=MaxL then 输出词链;{找到另一个解}
    Readln(Str){读入下一个单词}
  end
end.
```

四、总结

本题除了用动态规划外，还可以通过建树的方法分析，即用一颗多叉树来存储所有的单词，并通过标

记的方法找到最长链。根据输入文件已经按照字典顺序排序的条件，建树的过程和找最长链的过程都可以优化，而且多叉树也可以压缩为一个线性表。分析后的程序和动态规划的是一样的，只是对程序的解释不同。

分析中所用到的 5 条定理，都是从题目的条件入手，根据字典顺序和词链的定义得出的。表面上是对题目的分析，实际上是对动态规划所建立的有向无环图的特点的分析。而所有的优化，都是根据有向无环图（即题目的数学模型）的特殊性而得到的。

根据题目的条件，挖掘出有向无环图的特殊性质，从而优化算法。最终，时间复杂度降为 n，空间复杂度降为常数。这说明，动态规划相对于搜索来说，的确对问题的数学模型有了更精确的刻画，但并不代表不能再优化。对于很多经典的动态规划模型，特别是用有向无环图的最长路径描述的问题，有时仍然十分粗糙。这时就必须进一步的挖掘出模型的特殊性质，优化动态规划方程，从而达到优化算法的目的。

Land (IOI'99)

搜索中有剪枝。动态规划中同样可以进行剪枝，下面看一个问题。

一、问题描述

D 城的居民正在寻找一块场地修建机场跑道。当地的地图是现成的，整块土地为矩形，在地图上被规则地细划为若干单位正方形，每个正方形由一个（x，y）坐标表示，其中 x 是水平坐标，y 是垂直坐标，每个正方形的高度也被标明在地图上。

你的任务是帮助他们找到其中的一个面积最大（即其中包含的正方形数目最大）的矩形区域，该区域满足以下条件：

在该区域中，最高的正方形和最低的正方形在高度上差别不超过一个给定的阈值 C；

该区域的宽度（沿东西方向的正方形数目）不超过 100。

这样的最优解可能不止一个，而你只须找到一个即可。

假设条件

1≤U≤700，1≤V≤700，其中 U 为地图长度（即东西方向上的正方形数目），V 为地图宽度（即南北方向上的正方形数目）。

O≤C≤10

—30，000≤H≤30000，其中的整数，是坐标为 H_{xy} 是坐标为（x，y）处正方形的高度，1≤x≤U，1≤y≤V。

地图的坐标方向为：坐标（1，1）表示地图上的西南角，而（U，V）表示东北角。

输入

输入是一个名为 land.inp 的文本文件：

首行包括三个整数：U，V 和 C。

接下来的 V 行记录了各正方形处的高度，也就是说 H_{xy}，将出现在第（V—y+2）行的第 x 个位置上。

输出

输出是一个名为 land.out 的文本文件，内容只有一行，该行记录了找到的矩形区域的信息：X_{min}，Y_{min}，X_{max}，Y_{max}，其中（X_{min}，Y_{min}）是矩形区域的西南角坐标，（X_{max}，Y_{max}）是矩形区域的东北角坐标。

二、分析

《机场跑道》的核心问题是：在一个 U*V 的数字矩阵中，找到一个宽度不超过 100 且面积最大的子矩阵，使得该子矩阵中最大数与最小数的差不超过一个给定的数值 C。

矩阵(地图)的读入

由于 U,V≤700，矩阵的每个元素都需要 2 个字节。若一次性读入整个矩阵，共需空间 700*700*2=780KB，显然不能承受。由于题目中规定待求子矩阵宽度不超过 100，这说明我们可以每次只处理矩阵的一部分，通过多次读文件来降低空间上的需求。对于一个 700*700 的矩阵，若每次读入宽度为 300 的矩阵，分 3 次就可以读完。这样，空间需求为 700*300*2=420KB，可以承受。

子矩阵的表示

这里采用 4 元组(x,y,w,h)表示一个西南角坐标为(x,y)、宽 w、高 h 的矩阵。

找出满足要求的最大子矩阵

降维

由于矩阵是 2 维的，处理起来并不方便。对于这种多维的问题，一个基本的策略就是降维。

我们先确定子矩阵的 x 和 w，并在该前提下，设 Max[y]和 Min[y]分别表示第 y 行(即子矩阵(x,y,w,1))

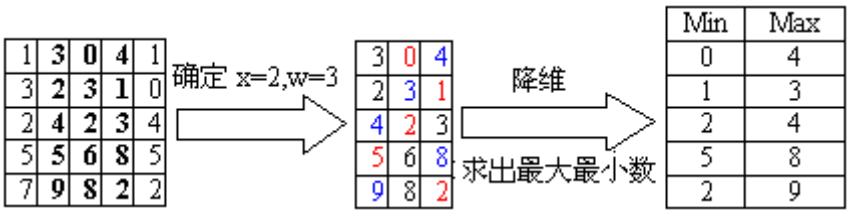


图 4-10 降维

的最大数和最小数(见图 4-10)，则问题变成了：在 Max 和 Min 中，找到一个最长的连续的区间[y,y+h-1]，使得区间内的最大数与最小数的差(即最大高度差)不超过给定的 C。这是一个一维的问题，它的实质是在确定矩阵的 x 和 w 的前提下，确定 y 和 h。

1. 一维问题的求解

我们首先穷举纵坐标 y，再让高度 h 从 1 起递增，直到 y+h-1=V(达到边界)或者 High[h]-Low[h]>C(高度差超过给定值)。High[h]和 Low[h]分别表示在区间[y,y+h-1]中的最大数和最小数，显然有 High[h]:=max{High[h-1],Max[h]},Low[h]:=min{Low[h-1],Min[h]}和初始条件 High[0]:=-∞,Low[0]:=∞。

该算法的最大时间复杂度约为 700^3*100，若不加优化，必然会超时。

2. 算法的优化

上述算法的基本思路是通过 4 重循环，分别确定子矩阵的 x,w,y,h。设已找到的最大子矩阵面积为 MaxS，如果在某层循环中发现，根据已经确定的子矩阵的属性，可以断定继续循环得不出面积比 MaxS 更大的子矩阵，则没有必要再循环下去了。我们可以根据这个思想进行优化剪枝。下面结合程序框图，设计优化的方法。

```
{设矩阵(地图)已经被一次性读入内存}
For x:=1 to U do{确定待处理矩阵的西南角横坐标}
    MaxW:=min(U-x+1,100);{MaxW 是待处理的子矩阵可能达到的最大宽度}
    MaxH[0]:=V;{MaxH[i]是待处理的子矩阵宽度等于 i 时达到的最大高度}
    for w:=1 to MaxW do{确定待处理矩阵的宽度}
        if MaxH[w-1]*MaxW≤MaxS then Break;{剪枝条件 1}
        计算数组 Max 和 Min;{降维}
        if MaxH[w-1]*w≤MaxS then {剪枝条件 2}
            MaxH[w]:=MaxH[w-1];
```

```
            Continue
        MaxH[w]:=0;
        for y:=1 to h do{确定待处理矩阵的西南角纵坐标}
            if (Max[y]<Max[y-1]) or (Min[y]>Min[y-1]) then{剪枝条件 3}
                for h:=1 to V-y+1 do{确定待处理矩阵的高度}
                    计算 High[h]和 Low[h];
                    If High[h]-Low[h]>C then{最大高度差超过 C}
                        Dec(h)
                        Break
                if w*h>MaxS then Update(x,y,w,h);{更新已找到的最大子矩阵}
                if h>MaxH[w] then MaxH[w]:=h{更新 MaxH[w]}
```

下面是 3 个剪枝条件的说明。

剪枝条件 1

如果已经确定待处理子矩阵的西南角横坐标是 x，那么随着宽度 w 的增加，显然，MaxH[w]不会变大(如图 4-11)。因此，在确定 w 的循环中，若 MaxH[w-1]*MaxW≤MaxS，则说明继续循环已经不可能找到面积比 MaxS 更大的子矩阵。

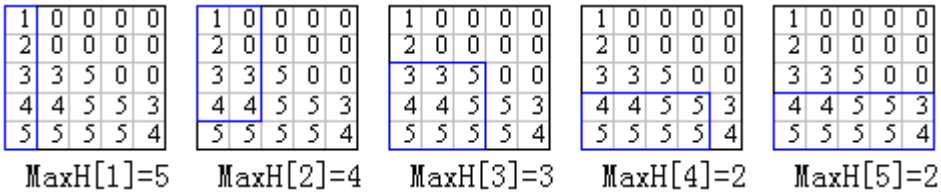


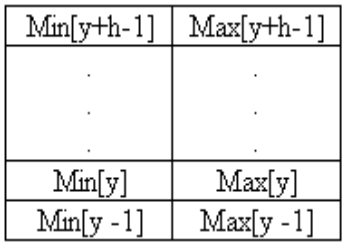
图 4-11 确定了 x=1 后，随着 w 的增加，MaxH[w]不会变大(图中 C=4)

剪枝条件 2

如果已经确定待处理子矩阵的 x 和 w，显然，它所能够达到的最大高度 MaxH[w]必然不会超过 MaxH[w-1]。因此，若 w*MaxH[w-1]≤MaxS，则说明此时一维问题的处理完全没有必要。

剪枝条件 3

剪枝条件 3 是说，在一维问题的处理中，如果待处理子矩阵的西南角纵坐标是 y，且 Min[y]≤Min[y-1]≤Max[y-1]≤Max[y]，则没有必要进行 h 的循环(如图 4-12)。



图中，Min[y]≤Min[y-1]≤Max[y-1]≤Max[y]。
如果区间[y,y+h-1]中最大数与最小数的差不超过数值 C，
显然，区间[y-1,y+h-1]中最大数与最小数的差也不超过数值 C，
且区间[y-1,y+h-1]比区间[y,y+h-1]所代表矩形的高度（面积）大。

图 4-12 剪枝条件 3

通过 3 个剪枝条件，算法的效率大大提高。

以题目的示例数据为例，下表列出了算法在在利用不同的剪枝条件的情况下，最里层循环语句的执行次数。

剪枝条件 1	√	×	√	×	√	×	√	×
剪枝条件 2	√	√	√	√	×	×	×	×
剪枝条件 3	√	√	×	×	√	√	×	×
循环次数	738	738	930	930	1554	1910	1998	2530

表 4-9

三、参考程序

```
{$R-,S-,Q-,I-}
program IOI99_Land;
const
  MaxV              =700;{一次读入的矩阵最大的高度}
  MaxU              =300;{一次读入的矩阵最大的宽度}
  MaxW              :Integer=100;{子矩阵的最大宽度}
  BufSize           =32768;{文件缓冲大小}
  Fin               ='Land.inp';
  Fout              ='Land.out';

type
  TSquare           =record{子矩形}
                     MaxS      :Longint;{面积}
                     xx,yy     :Integer;{西南角坐标}
                     ww,hh     :Integer{宽度和高度}
  end;

  TLine             =array[1..MaxV] of Integer;

var
  Buf               :array[0..BufSize-1] of Char;{文件缓冲}
  Map               :array[1..MaxU] of ^TLine;{地图}
  Best              :TSquare;{最大子矩形}
  U,V,C             :Integer;{子矩阵的宽度,高度和最大允许高度差}

procedure Update(S:Longint;x,y,w,h:Integer);{更新最大子矩形}
  begin
    with Best do begin
      if S<=MaxS then Exit;
      MaxS:=S;
      xx:=x;yy:=y;ww:=w;hh:=h
    end
  end;

procedure GetMap(Head,Tail:Integer);{读入西南角横坐标在[Head,Tail]间的矩阵}
  var
    x,y,k           :Integer;
  begin
    Reset(Input);
    Readln(k,k,k);
    for y:=V downto 1 do begin
      for x:=1 to Head-1 do Read(k);
      for x:=1 to Tail-Head+1 do Read(Map[x]^[y]);
      Readln
```

```
    end
  end;

procedure Main;{找到满足要求的最大面积的子矩阵}
  var
    Head,Tail,High,Low    :Integer;
    MinW,MaxH             :Longint;
    x,y,w,h               :Integer;
    Min,Max               :array[0..MaxV] of Integer;
    Tmp                   :TLine;
    Quit                  :Boolean;
  begin
    for x:=1 to MaxU do New(Map[x]);{分配空间}
    Assign(Input,Fin);SetTextBuf(Input,Buf,Sizeof(Buf));Reset(Input);
    Readln(U,V,C);
    Head:=1;Tail:=Head+MaxU-1;Quit:=False;MinW:=MaxW;
    Max[0]:=MaxInt;Min[0]:=-MaxInt;
    repeat
      if Tail>=U then begin
        Quit:=True;
        Tail:=U;MinW:=0
      end;
      GetMap(Head,Tail);
      for x:=Head to Tail-MinW do begin{x 是子矩阵西南角横坐标}
        if x+MaxW-1>Tail then MaxW:=Tail-x+1;
        for y:=1 to V do begin
          Max[y]:=-MaxInt;
          Min[y]:=MaxInt
        end;
        MaxH:=V;
        for w:=1 to MaxW do begin{w 是子矩阵的宽度}
          if MaxH*MaxW<=Best.MaxS then Break;{剪枝条件 1}
          Tmp:=Map[x+w-Head]^;
          for y:=1 to V do begin
            if Tmp[y]>Max[y] then Max[y]:=Tmp[y];
            if Tmp[y]<Min[y] then Min[y]:=Tmp[y]
          end;
          if MaxH*w<=Best.MaxS then Continue;{剪枝条件 2}
          MaxH:=0;
          for y:=1 to V do{y 是子矩阵西南角纵坐标}
            if (Max[y]<Max[y-1]) or (Min[y]>Min[y-1]) then begin{剪枝条件 3}
              High:=-MaxInt;Low:=MaxInt;
              for h:=1 to V-y+1 do begin{h 是子矩阵的高度}
                if Max[y+h-1]>High then High:=Max[y+h-1];
                if Min[y+h-1]<Low then Low:=Min[y+h-1];
```

```
                    if High>Low+C then begin
                      Dec(h);
                      Break
                    end
                  end;
                  if h>MaxH then begin
                    MaxH:=h;
                    Update(MaxH*w,x,y,w,h)
                  end
                end
              end
            end;
            Head:=Tail-MaxW+1;Tail:=Head+MaxU-1
          until Quit;
          Close(Input)
        end;

      procedure Print;
        begin
          Assign(Output,Fout);ReWrite(Output);
          with Best do Writeln(xx,' ',yy,' ',xx+ww-1,' ',yy+hh-1);{输出}
          Close(Output)
        end;

      begin
        Main;
        Print
      end.
```

理想收入

在前面分析序关系计数问题时，动态规划相对于搜索，之所以有着更高的效率，就在于它对信息的利用程度，较搜索对信息的利用程度更高了。而加强对信息的利用程度，不光可以将某些搜索算法改进成动态规划，它同样可以用于进一步优化动态规划算法。下面就举这样一个例子。

一、问题描述

理想收入是指在股票交易中，以 1 元为本金可能获得的最高收入，并且在理想收入中允许有非整数股票买卖。

已知股票在第 i 天每股价格是 V[i]元，1≤i≤M，求 M 天后的理想收入。

二、分析

<1>. 一种动态规划的解法

解这道题目，很容易想到用动态规划。设 F[i]表示在第 i 天收盘时能达到的最高收入，则有 F[i]的递推关系式：

公式2：$F[i]=\max_{(0\le j\le k<i)}\{F[j]/V[k]*V[i]\}$，其中$F[0]=1$，$V[0]=1$

公式 2 的含义是：在第 i 天收盘时能达到的最高的收入，是将第 j 天收盘后的收入，全部用于买入第 k 天的股票，再在第 i 天将所持的股票全部卖出所得的收入。采用公式 2 可以得到算法 1，它的时间复杂度是 $O(M^3)$，空间复杂度是 O(M)。

算法 1

```
F[0]:=1;V[0]:=1;F[1..M]:=0;
for i:=1 to M do
  for j:=0 to i-1 do
    for k:=j to i-1 do
      F[i]:=Max{F[i],F[j]/V[k]*V[i]}
```

〈2〉. 改变状态表示的含义，优化算法 1

改变动态规划中状态表示的含义，是优化动态规划的常用方法。例如这道题目，我们可以采用两种不同的状态表示方法优化算法 1。

方法 1：设 P[i]表示前 i 天能获得的最多股票数，则可列出状态转移方程：

公式3：$P[i]=\max_{(0\le j<i)}\{P[i-1],P[j]*V[j]/V[i]\}$

这是因为前 i 天所能获得的最多股票数，或者是前 i-1 天获得的最多股票数，或者是在第 j 天将前 j 天所能获得的最多的股票全部卖出，再买入第 i 天的股票。显然，前 i-1 天能获得的最多股票数乘以第 i 天的股价，就是第 i 天能达到的最大收入。

方法 2：设 Q[i]表示前 i 天能达到的最大收入，则可列出状态转移方程：

公式4：$Q[i]=\max_{(0\le j<i)}\{Q[i-1],Q[j]/V[j]*V[i]\}$

就是说前 i 天所能达到的最大收入，或者是前 i-1 天所能达到的最大收入，或者是在第 j 天买入股票，再在第 i 天卖出，所能获得的最大收入。

这两种方法的时间复杂度都是 $O(M^2)$。这表明，改变状态表示的含义，在一定程度上提高了算法的效率。但对于这道题目，仅仅改变状态表示的含义，很难进一步优化算法。

〈3〉. 粗略利用信息，优化算法 1

算法 1 粗体部分的功能是确定 F[i]所能达到的最大值。由于 V[i]不变，因此 F[i]达到最大值，当且仅当 F[j]/V[k]达到最大值，其中 0≤j≤k<i。

算法 2-1 是采用二重循环确定 F[j]/V[k]达到的最大值的。但在确定 F[i-1]所能达到的最大值的时候，我们实际上已经求出当 0≤j≤k<i-1 时，F[j]/V[k]所能达到的最大值。如果能充分利用这一信息，就可以更快的确定 F[i] 所能达到的最大值。如图 4-13，要确定粗线下部的最大值，只需比较虚线下部的最大值和灰色部分的最大值即可。

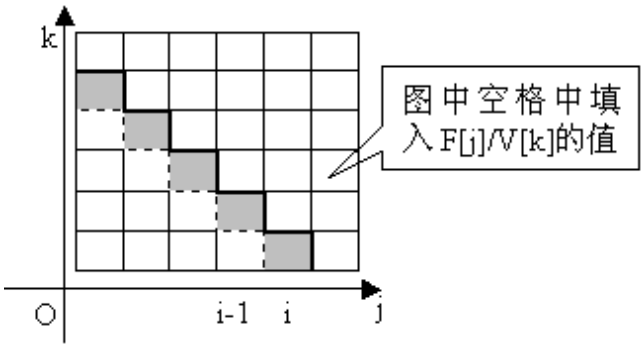


图 4-13 初步利用信息，优化算法 2-1

为了表示出图 4-13 的思想，

设$MaxFV[i]=\max_{(0\le j\le k<i)}\{F[j]/V[k]\}$，则

$$\begin{aligned}MaxFV[i]&=\max_{(0\le j\le k<i)}\{F[j]/V[k]\}\\&=\max\{\max_{(0\le j\le k<i-1)}\{F[j]/V[k]\},\max_{(0\le j\le k,k=i-1)}\{F[j]/V[k]\}\}\\&=\max\{MaxFV[i-1],\max_{(0\le j\le i-1)}\{F[j]/V[i-1]\}\}\\&=\max_{(0\le j<i)}\{MaxFV[i-1],F[j]/V[i-1]\}\end{aligned}$$

由公式2，$$\begin{aligned}F[i]&=\max_{(0\le j\le k<i)}\{F[j]/V[k]*V[i]\}\\&=\max_{(0\le j\le k<i)}\{F[j]/V[k]\}*V[i]\\&=MaxFV[i]*V[i]\end{aligned}$$

这样，我们得到如下递推关系式：

公式5：

$$F[i]=MaxFV[i]*V[i]$$

$$MaxFV[i]=\max_{(0\le j<i)}\{MaxFV[i-1],F[j]/V[i-1]\}$$

从公式 5 中可以看出，在确定 MaxFV[i]时，较充分的利用了确定 MaxFV[i-1]时产生的结果。采用公式 5 可得算法 2，它的时间复杂度为 $O(M^2)$，空间复杂度是 O(M)。

算法 2

```
F[0]:=1;MaxFV[0]:=0;V[0]:=1;
for i:=1 to M do begin
   MaxFV[i]:=MaxFV[i-1];
   for j:=0 to i-1 do
      MaxFV[i]:=Max{MaxFV[i],F[j]/V[i-1]}
   F[i]:=MaxFV[i]*V[i]
end;
```

将公式 5 化简，有

$$\begin{aligned}MaxFV[i]&=\max_{(0\le j<i)}\{MaxFV[i-1],F[j]/V[i-1]\}\\&=\max_{(0\le j<i)}\{MaxFV[i-1],MaxFV[j]*V[j]/V[i-1]\}\end{aligned}$$

在这个公式中，MaxFV[i]可以看作是前 i-1 天所能获得的最多股票数。这种状态表示方法和公式 3 中 P[i]的含义本质上是相同的。这样，我们通过对已知信息的利用，达到了改变动态规划中状态表示含义的

效果。

<4>. 充分利用信息，进一步优化算法 2

在算法 2 中，进一步利用信息，很容易得到时间复杂度为 O(M)的算法。

算法 2 的粗体部分的功能是确定 MaxFV[i]所能达到的最大值。由于 V[i-1]不变，因此 F[j]/V[i-1]达到最大值，当且仅当 F[j]达到最大值，其中 0≤j<i。

算法 2 中内层循环实质上是在确定 MaxFV[i]时，找到 F[0],F[1],…F[i-1]中的最大值。而在确定 MaxFV[i-1]时，我们已经找到了 F[0],F[1],…,F[i-2]中的最大值。如果把这个信息利用起来，就可以更快的确定 F[0],F[1],…F[i-1]中的最大值。

设$MaxF[i]=\max_{(0\le j<i)}\{F[j]\}$，则

$$\begin{aligned}MaxF[i]&=\max_{(0\le j<i)}\{F[j]\}\\&=\max\{\max_{(0\le j<i-1)}\{F[j]\},F[i-1]\}\\&=\max\{MaxF[i-1],F[i-1]\}\end{aligned}$$

$$\begin{aligned}MaxFV[i]&=\max_{(0\le j<i)}\{MaxFV[i-1],F[j]/V[i-1]\}\\&=\max\{MaxFV[i-1],MaxF[i]/V[i-1]\}\end{aligned}$$

这样，我们得到如下递推关系式：

公式6：

$$F[i]=MaxFV[i]*V[i]$$

$$MaxFV[i]=\max\{MaxFV[i-1],MaxF[i]/V[i-1]\}$$

$$MaxF[i]=\max\{MaxF[i-1],F[i-1]\}$$

从公式 6 中可以看出，在确定 MaxF[i]时，充分利用了确定 MaxF[i-1]时所产生的信息。采用公式 6 可得算法 3，它的时间复杂度是 O(M)。公式 6 中的三个递推关系式，当前状态都只与前一个状态有关，因此，空间复杂度可以进一步降到 O(1)[6]。

算法 3

```
F:=1;MaxF:=0;MaxFV:=0;V[0]:=1;
for i:=1 to M do begin
   if F>MaxF then MaxF:=F;
   if MaxF/V[i-1]>MaxFV then MaxFV:=MaxF/V[i-1];
   F:=MaxFV*V[i]
end;
```

公式 6 中 MaxF[i]可以看作是前 i-1 天能达到的最大收入。虽然这种状态表示方法和公式 4 中 Q[i]是类似的，但算法的时间复杂度却从 O(M^2)降到了 O(M)，空间复杂度也从 O(M)降到了 O(1)。这样，我们通过充分利用已知信息，达到了改变状态表示难以达到的优化效果。

三、小结

下面是三个算法的性能分析表：

分析项目		算法 1	算法 2	算法 3
理论分析	时间复杂度	O(M^3)	O(M^2)	O(M)
	空间复杂度	O(M)	O(M)	O(1)
实际运行情况	M=200 的随机数据	4s	0.05s	<0.05s
	M=1000 的随机数据	>60s	1s	<0.05s
	M=2000 的随机数据	>60s	5s	<0.05s

表 4-10

在这道题目中，我们通过避免冗余运算，将时间复杂度由 $O(M^3)$先降到 $O(M^2)$，再进一步降到 O(M)。空间复杂度也从 O(M)降到了 O(1)。

由此可见，充分利用信息，提高动态规划的效率，是非常有效的。此外，充分利用信息并不一定要以牺牲空间为代价，同样可以优化算法的空间复杂度。