动态规划

【基础知识】

一．基本概念

动态程序设计（动态规划）是解决多阶段决策最优化问题的一种思想方法。所谓“动态”，指的是在问题的多阶段决策中，按某一顺序，根据每一步所选决策的不同，将随即引起状态的转移，最终在变化的状态中产生一个决策序列。动态规划就是为了使产生的决策序列在符合某种条件下达到最优。

动态程序设计是一种重要的程序设计思想，具有广泛的应用价值。使用动态规划思想来设计算法，对于不少问题往往具有高时效，因而，对于能够使用动态程序设计思想来解决的问题，使用动态规划是比较明智的选择。

二．基本原理

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”。

无后效性原则：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。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。

三．动态程序设计的优化方法

线性模型：f(i)=optimal{f(j)+w(i,j)}

树状模型：dfs 遍历，儿子兄弟表示法

二维模型 1：

$$f(i,j)=optimal\begin{cases} f(i-1,j)+1, & \text{条件 1} \\ f(i,j+1)+1, & \text{条件 2} \\ f(i-1,j+1) & \text{条件 3} \end{cases}$$

二维模型 2：f(i,j)=optimal{f(k,j-1)+w(k+1,i)}

集合模型：状态是集合，我们往往可以用二进制或者三进制数来表示。

……

四．动态规划解题的步骤

1、确定问题的研究对象，即状态。选定的状态必须满足如下两点：

⑴状态必须完全描述出事物的性质，两个不同事物的状态是不同的;

⑵必须存在状态与状态之间的“转移方程”，以便我们可以由初始状态逐渐转化为目标状态。

2、划分阶段，确定阶段之间的状态转移方程。

3、考察此问题可否用动态程序设计方法解决，即问题是否具备最优子结构和无后效性的特征。

4、如果发现问题目前不能用动态程序设计方法解决，则调整阶段的划分和状态的定义，使其具备最优子结构和无后效性的特征。

【典型题解析】

【例 1】金明的预算方案

金明在做预算，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的。如

清北学堂

果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有 0 个、1 个或 2 个附件。附件不再有从属于自己的附件。金明想买的东西很多，肯定会超过妈妈限定的 N 元。于是，他把每件物品规定了一个重要度，分为 5 等：用整数 1~5 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是 10 元的整数倍）。他希望在不超过 N 元（可以等于 N 元）的前提下，使每件物品的价格与重要度的乘积的总和最大。设第 j 件物品的价格为 v[j]，重要度为 w[j]，共选中了 k 件物品，编号依次为 j1，j2，……，jk，则所求的总和为：v[j1]*w[j1]+v[j2]*w[j2]+ …+v[jk]*w[jk]。（其中*为乘号） 请你帮助金明设计一个满足要求的购物单。

【分析】

把每个主件和属于它的附件组合在一起，构成属于这个主件的不同方案。

例如　800 2 0

400 3 1

100 5 1

该主件有两个附件，则它的不同方案有(1)(1,2)(1,3)(1,2,3)

由于每个主件至多有 2 个附件，所以至多有 4 个方案

考虑深度优先搜索：

Void search(int Money, int k)

//Money 表示还剩多少钱，k 表示当前枚举到第几个主件

在搜索内部枚举使用该主件的第几个方案

该算法效率不高，原因是存在着大量的重复计算：对于同样的(Money, k)，我们可能要进行多次计算；而对于当前的搜索状态(Money, k)，接下来的搜索结果与之前的枚举无关。我们考虑用数组 f [Money][k]从(Money, k)开始搜索得到的最优值。冗余计算没有了，算法效率大大提高。这就是动态规划的一种表现形式。算法的时间复杂度为 O(Nm)

【例 2】机器分配问题：

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。

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。

【分析】

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]} (1<=I<=N,1<=J<=M,0<=K<=J)

初始值: F(0,0)=0

时间复杂度 O(N*M2)

【例 3】住宿分配：

有 N 个男人，M 个女人，其中有 C 对夫妇要住房。现在有 P 个房子。每个房子有个费用 Ci 和床位 Bi。住房有以下要求:

1.每个房子住的人数不能超过 Bi

2.一个房间住了夫妇，不能再住其他人。

3.不考虑夫妇情况下：一个房间住了男人后，不能再住女人。对女人也是一样。

问最少的费用。(n<500,m<500,P<500,Bi<=5)。

【分析】

事实上我们可以发现按照第 2 条住房的夫妇数不会超过 1。如果有 2 对夫妇那么可以交换一下，在同样代价下反而能住更多人。所以我们可以枚举夫妇数，这样就去掉了夫妇的这个限制。

设 F[i,j,k]表示当前 1-i 这些房间安排好后，还剩下 j 男 k 女的最小费用。

很容易写出状态转移方程：

F[i,j,k]=Min{f[i-1,j,k],f[i-1,j+b[i],k]+c[i],

f[i-1,j,k+b[i]]+c[i]}

但是这个动态规划的时间复杂度太高了。如果人数很多的话(k>size,l>size) ，那么肯定无论如何 Ci/Bi 最小的那些房间肯定会被选到。于是我们可以贪心在 k>size,l>size 的时候，给他们安排 Ci/Bi 最小的房间。然后再进行动态规划。由于 Bi<=5.所以 size=20 就够了。这样时间复杂度就很低了。

【例 4】Robots：

在一个 n*m 的棋盘内，一些格子里有垃圾要拾捡。现在有一个能捡垃圾的机器人从左上格子里出发，每次只能向右或者向下走。每次他到达一个点，就会自动把这个点内的垃圾拾掉。

问：最多能拾多少垃圾。在最多的情况下，有多少种方案？请举出一种方案来。

数据范围:n<=100,m<=100

	1	2	3	4	5	6	7
1		G		G			
2				G		G	
3							
4				G			G
5							
6						G	

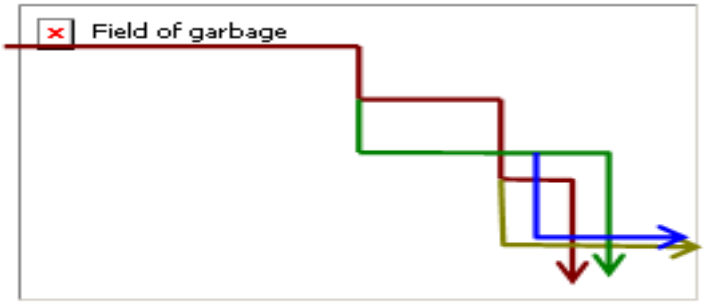


Figure 2: Four sample solutions

【分析】

最多能拾 5 块。有 4 种方法。因为只能向右或者向下走。也就是说不能走回头路。于是考虑动态规划。

设 F[i,j]表示从(1,1)点开始走到(i,j)的时候，最多捡了多少垃圾。G[i,j]表示在 f[i,j]最大的时候，有多少种方案。C[i,j]=1 表示(i,j)点有垃圾。C[i,j]=0 表示没有。根据(i,j)只能从(i-1,j)或者(i,j-1)走过来。于是 f[i,j]=Max{f[i-1,j],f[i,j-1]}+c[i,j].

g[i,j]=g[i-1,j]*k+g[i,j-1]*L。

如果 f[i-1,j]+c[i,j]=f[i,j]，则 K=1 否则 K=0。

如果 f[i,j-1]+c[i,j]=f[i,j]，则 L=1 否则 L=0。

于是我们得到第 1，2 问的答案。对于第 3 问，我们只要简单得在动态规划的时候记录一个决策即从哪个方向走过来的就可以了。时间复杂度 O(nm)。

【例 5】两种颜色的马

农夫 Ion 放完马以后，需要把马儿关回马厩。为了做好这件事，Ion 让马排成一行跟着他入马厩。他想出了一个就近入厩的办法：让前 P1 匹马进入第一个马厩， 然后的 P2 匹马进入第二个马厩，如此类推。而且，他不想让任何一个马厩（共 k 个）留空，还有所有的马都进入马厩。

已知 Ion 只有黑色和白色两种颜色的马，然而并不是所有的马都能相处融洽。假如有 i 匹黑马和 j 匹白马同在一个马厩，那么它们之间的不愉快系数为 i*j。马厩总的不愉快系数等于 k 个马厩的不愉快系数之和。

请帮忙把 N 匹马按顺序放入 k 个马厩中（即求一种 P1,P2…的安排方案），使得总的不愉快系数最小。

【分析】

设

f(i,j)表示将前 i 匹马放入前 j 个马厩，得到的最小不愉快系数。

w(i,j)表示将第 i 至第 j 匹马放入同一个马厩所得到的不愉快系数。

f(i,j)=min{f(k, j-1)+w(k+1,i)}　其中 j<=k<i)。

该算法的时间复杂度为 O(n3)

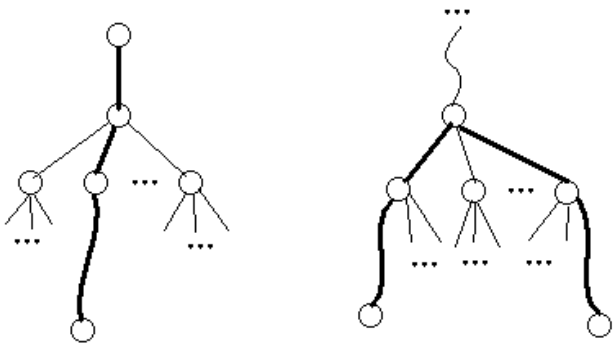
【例 6】树的最长链问题：

给你一棵树 T=(V,E,W)，其中 V 表示顶点集且|V|=n，E 表示边集。如果<v,u>属于 E，则 W<v,u>表示<v,u>的长度。

求两点 v,u，使得它们之间的路径总长度最长。你只需要输出这个最长长度即可。

【分析】

一棵有根树的最长链，可能出现如下图的两种情况：



设 dep(i)表示以节点 i 为根的子树的最大深度。

F(i)表示以节点 i 为根的子树中，包含节点 i 的最长链长度。

我们有，

dep(i)=max{dep(j)+w(i,j)}，其中 j 是 i 的子节点。

F(i)=max{dep(i), dep(j)+w(i,j)+dep(k)+w(i,k)}，其中 j,k 是 i 的子节点，且 j<>k。

不难发现，我们的状态转移方程是按照从下至上的顺序计算的。

做一遍 DFS 遍历，在回朔的时候分别计算 dep 和 F 的值。

关于 F 值的计算：由于节点 j 和 k 之间没有关联，所以我们只需要选择两个(dep(j)+w(i,j))最大的子节点进行累加即可。

整个算法的复杂度为 O(|V|+|E|)=O(n)。

【例 7】采药问题：

辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗？

【输入文件】输入文件 medic.in 的第一行有两个整数 T（1 ≤ T ≤ 1000）和 M（1 ≤ M ≤ 100），用一个空格隔开，T 代表总共能够用来采药的时间，M 代表山洞里的草药的数目。接下来的 M 行每行包括两个在 1 到 100 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

【输出文件】输出文件 medic.out 包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

【分析】

设 a[j]为 j 时间内可采到的草药的最大总价值。如果第 i 株草药的采摘时间为 x、价值为 y 的话，则开始采摘第 i 株草药的时刻 j 的可能范围为[0，t-x]，采完第 i 株草药的时刻为 j+x。在该时间内内可采到的草药的最大总价值 a[j+x]= $\max_{0\le j\le t-x}\{a[j]+y\,|\,a[j]\ge 0\}$

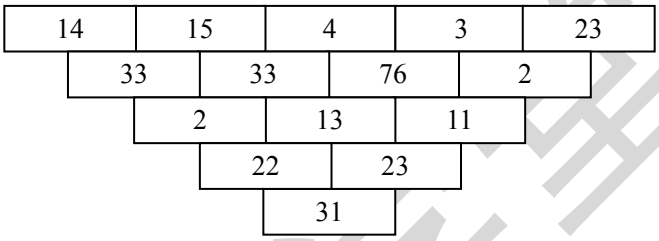
显然，在 t 时间内可以采到的草药的最大总价值 ans= $\max_{0\le i\le t}\{a[i]\}$

```
var a:array[0..1000]of longint; {a[j]为 j 时间内可采到的草药的最大总价值
a[j+x]=                                                    }
    t,m,i,j,x,y,ans:longint;
begin
 assign(input,'medic.in');assign(output,'medic.out');
 reset(input);rewrite(output);
 readln(t,m); {输入总共能够用来采药的时间和山洞里的草药数}
 for i:=1 to t do a[i]:=-1;
 a[0]:=0;
 for i:=1 to m do{逐株采摘草药}
  begin
   readln(x,y);{输入采摘第 i 株草药的时间和价值}
   for j:=t-x downto 0 do{枚举开始采摘第 i 株草药的时刻}
    if (a[j]>=0)and(a[j+x]<a[j]+y) then a[j+x]:=a[j]+y
```

```
  end;{ for }
 ans:=-1; {在 t 时间内可采到的草药的最大总价值 ans=                    }
 for i:=0 to t do if ans<a[i] then ans:=a[i];
 writeln(ans);{输出 t 时间内可采到的草药的最大总价值}
 close(input);close(output){关闭输入文件和输出文件}
end.
```

【例 8】打方块

有一面墙如图放置，一共 n 层，从上往下第 i 层有 n-i+1 个砖块。现在需要打掉 m 块砖。我们把层从上往下编号，每一层从左往右编号，那么，如果要打掉第 i 层的第 j 块砖，必须打掉第 i-1 层的第 j 和第 j+1 块砖。每一块砖有一个价值。现在请你安排一种方案，使得打掉的价值总和最大。



【分析】

首先我们把墙面拉成一个直角三角形的样子，便于研究。如图：

14	15	4	3	23
33	33	76	2	
2	13	11		
22	23			
31				

这样，我们可以得出下面的结论：

每一列必须打掉从最上面开始由上至下的连续若干个砖块;

如果某一列需要打掉连续的 p 个砖块，那么它右边的那一列至少打掉 p-1 个砖块。

通过上面的分析，启发我们用动态规划解决这个题目：

阶段：我们按每一列从右到左分阶段（为了便于分析计算，我们将列重新从右至左编号）

状态：某一列取多少个连续的砖块+前面一共取多少砖块。

令：

F (i, j, k)为右数第 i 列打掉连续 j 个砖块，而且前 i 列共打掉 k 个砖块的得到的最大价值。

转移方程如下：

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

$$j-1\le p\le \min\{i-1,k-j\}$$

其中 s (i, j)表示第 i 列前 j 个砖块的价值和。
为了便于决策和优化，我们令

$$G(i,j,k)=\max\{f(i,p,k)\mid j\le p\le \min\{i,k\}\}$$

则转移方程简化为

$$f(i,j,k)=G(i-1,j-1,k-j)+s(i,j)$$

边界：f(0,0,0)=0；f (i, 0, k)=G(i-1, 0, k)
答案就是 max {f (i, j, m)}(1<=i<=n，0<=j<=i)

时间复杂度分析：
计算 s (i, j)的复杂度为 O(n2)
G (i, j, k)可以这样计算：
G(I, j, k)=max {f (i, j, k) , G (i, j+1, k)}
所以计算 f 和 G 数组的复杂度都为 O(mn2)。
算法的总时间复杂度为 O(mn2)。

【例 9】快餐问题

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: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。第二行为 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:每天套餐的最大产量。

【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<=100,0<=k1<=k<=100,
且(j-j1)*p1+(k-k1)*p2<=T[i]---第 i 条生产线的生产时间)
r[i,j-j1,k-k1]=(T[i]-(j-j1)*p1+(k-k1)*p2) div p3 ;
此算法的时间复杂度为 O(N*1004),

优化：

在本题中，可以在动态规划方法中加入了贪心算法思想：即首先计算出每天生产套数的上限值（由 A,B,C 计算，即 min｛100 div A，100 div B，100 div c｝），接着，用贪心法计算出这 N 条流水线可以生产的套数，并与上限比较，大于则输出上限值并退出，否则再调用动态规划；同时，在运行动态规划的过程中，也可以每完成一阶段工作便与上限值进行比较，这样以来，便可望在动态规划完成前提前结束程序。其算法设计为：

S1：读入数据。

S2：贪心求上限并计算出一可行解，判断是否需进行下一步。

S3：动态规划求解。

S4：输出。

其他优化方法：

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.减少循环次数：在计算每一阶段状态过程中无疑要用到 4 重循环，我们可以这样修改每一重循环的起始值和终数，其中 q1,q2 为 A，B 上限值。

原起始值	改进后的起始值
for i:=1 to n do	for i:=1 to n do
for j:=0 to tot[i] div p1 do	for j:=0 to min(q1,tot[i] div p1) do
for k:=0 to (tot[i]-p1*j) div p2 do	for k:=0 to min(q2,(tot[i]-p1*j)div p2) do
for j1:=0 to j do	for j1 := max(0,j-t[i] div p1) to min(j,tot[i-1] div p1) do
for k1 := 0 to k do	for k1:=max(0,k-(t[i]-(j-j1)*p1) div p2) to min(k,(tot[i-1]-p1*j1)div p2) do

【例 10】合唱队形

N 位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的 K 位同学排成合唱队形。

合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 1, 2, …, K，他们的身高分别为 T1, T2, …, TK，则他们的身高满足 T1 < T2 < … < Ti , Ti > Ti+1 > … > TK (1≤i ≤K)。

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

【输入文件】输入文件 chorus.in 的第一行是一个整数 N（2 ≤ N ≤ 100），表示同学的总数。第一行有 n 个整数，用空格分隔，第 i 个整数 Ti（130 ≤ Ti ≤ 230）是第 i 位同学的身高（厘米）。

【输出文件】输出文件 chorus.out 包括一行，这一行只包含一个整数，就是最少需要几位同学出列。

【分析】

解法 1：动态程序设计方法

我们按照由左而右和由右而左的顺序，将 n 个同学的身高排成数列。如何分别在这两个数列中寻求递增的、未必连续的最长子序列，就成为问题的关键。设

a 为身高序列，其中 a[i]为同学 i 的身高；

b 为由左而右身高递增的人数序列，其中 b[i]为同学 1‥同学 i 间（包括同学 i）身高满

足递增顺序的最多人数。显然

b[i]= $\max_{1\le j\le i-1}${b[j]|同学 j 的身高<同学 i 的身高}+1；

c 为由右而左身高递增的人数序列，其中 c[i]为同学 n‥同学 i 间（包括同学 i）身高满足递增顺序的最多人数。显然

c[i]= $\max_{i+1\le j\le n}$ {c[j]|同学 j 的身高<同学 i 的身高}+1；

由上述状态转移方程可知，计算合唱队形的问题具备了最优子结构性质(要使 b[i]和 c[i]最大，子问题的解 b[j]和 c[k]必须最大(1≤j≤i-1 ，i+1≤k≤n))和重迭子问题的性质(为求得 b[i]和 c[i]，必须一一查阅子问题的解 b[1]‥b[i-1]和 c[i+1]‥c[n])，因此可采用动态程序设计的方法求解。

显然，合唱队的人数为 $\max_{1\le i\le n}\{b[i]+c[i]\}-1$ (公式中同学 i 被重复计算，因此减 1)，n 减去合唱队人数即为解。

程序如下:

```
readln(n); {读学生数}
  for i:=1 to n do read(a[i]); {读每个学生的身高}
  fillchar(b,sizeof(b),0);fillchar(c,sizeof(c),0);{身高满足递增顺序的两个队列初始化}
  for i:=1 to n do{按照由左而右的顺序计算 b 序列}
   begin
    b[i]:=1;
    for j:=1 to i-1 do if (a[i]>a[j])and(b[j]+1>b[i]) then b[i]:=b[j]+1;
   end;{for}
  for i:=n downto 1 do{按照由右而左的顺序计算 c 序列}
   begin
    c[i]:=1;
        for j:=i+1 to n do if (a[j]<a[i])and(c[j]+1>c[i])then c[i]:=c[j]+1;
    end;{for}
  max:=0;{计算合唱队的人数 max（其中 1 人被重复计算）}
  for i:=1 to n do if b[i]+c[i]>max then max:=b[i]+c[i];
  writeln(n-max+1); {输出出列人数}
```

这个算法的时间复杂度为 O(n2)

解法 2：二分法

设 x 为当前身高满足递增顺序的队列，其中 x[i]为第 i 高的队员身高；a 序列、b 序列和 c 序列的定义如解法 1。

设 x[i]的初始值为∞(1≤i≤n)，b[0]为 0。我们从同学 1 出发，按照由左而右的顺序计算身高递增的人数序列 b。在计算 b[i]时，通过二分法找出区间 x[1]‥x[i]中身高矮于同学 i 的元素个数 min（x 区间的左右指针为 min 和 max，中间指针为 mid）

在计算出 b 序列后，再采用类似方法由右而左计算身高递增的人数序列 c。最后得出出列人数为

n- $\max_{1\le i\le n}\{b[i]+c[i]\}-1$ (公式中同学 i 被重复计算，因此减 1)。

程序如下：

```
readln(n);{读学生数}
  for i:=1 to n do read(a[i]);{读每个学生的身高}
```

```
for i:=1 to n do x[i]:=maxlongint;{身高满足递增顺序的队列初始化}
b[0]:=0;
for i:=1 to n do{由左而右计算 b 序列}
begin
  min:=0;max:=i;{设 x 区间的左右指针}
  while min<max-1 do{若 x 区间存在，则通过二分法计算同学 i 前身高满足递增顺序的最多人数 min}
  begin
    mid:=(min+max) div 2;{计算中间指针}
    if x[mid]<a[i]
    then min:=mid{搜索右区间}
    else max:=mid{搜索左区间}
  end;{while}
  b[i]:=min+1;x[min+1]:=a[i]{同学 1‥同学 i 间（包括同学 i）最多有 min+1 个同学可排成身高递增的队列}
end;{for}

for i:=1 to n do x[i]:=maxlongint; {身高满足递增顺序的队列初始化}
c[0]:=0;
for i:=n downto 1 do{由右而左计算 c 序列}
begin
  min:=0;max:=i;
  while min<max-1 do
  begin
    mid:=(min+max) div 2;  设 x 区间的左右指针}
    if x[mid]<a[i]
    then min:=mid{搜索右区间}
    else max:=mid{搜索左区间}
  end;{while}
  c[i]:=min+1;x[min+1]:=a[i] {同学 n‥同学 i 间（包括同学 i）最多有 min+1 个同学可排成身高递增的队列}
end;{max}
max:=0;{计算合唱队的人数 max（其中 1 人被重复计算）}
for i:=1 to n do if b[i]+c[i]>max then max:=b[i]+c[i];
writeln(n+1-max);{输出出列人数}
```

第二种解法的算法的时间复杂度为 O(n*log n)

【例 11】加分二叉树

设一个 n 个节点的二叉树 tree 的中序遍历为（l,2,3,…,n），其中数字 1,2,3,…,n 为节点编号。每个节点都有一个分数（均为正整数），记第 j 个节点的分数为 di，tree 及它的每个子树都有一个加分，任一棵子树 subtree（也包含 tree 本身）的加分计算方法如下：

subtree 的左子树的加分× subtree 的右子树的加分＋subtree 的根的分数。若某个子树为空，规定其加分为 1，叶子的加分就是叶节点本身的分数。不考虑它的空子树。

清北学堂

试求一棵符合中序遍历为（1,2,3,…,n）且加分最高的二叉树 tree。要求输出；

（1）tree 的最高加分

（2）tree 的前序遍历

【输入格式】第 1 行：一个整数 n（n＜30），为节点个数；

第 2 行：n 个用空格隔开的整数，为每个节点的分数（分数＜100）。

【输出格式】第 1 行：一个整数，为最高加分（结果不会超过 4,000,000,000）；第 2 行：n 个用空格隔开的整数，为该树的前序遍历。

【分析】

解法 1：采用动态程序设计方法计算最大分值

设 f[i,j]——顶点 i…顶点 j 所组成的子树的最大分值。若 f[i,j]=-1，则表明最大分值尚未计算出。

way[i,j]——顶点 i…顶点 j 所组成的子树达到最大分值时的根编号。i=j 时，way[i,i]:=I。

由于问题没有明显的阶段特征，而是呈非线性的树形结构，因此我们采用后序遍历的顺序计算状态转移方程。

```
function search(l,r : integer) : int64;{递归计算 f[l,r]}
  var
    i : integer; now : int64;{当前分值}
begin
if l>r then search:=1
     else begin
          if f[l,r]=flag then{若尚未计算出顶点 l‥顶点 r 对应子树的最高分值}
            for i:=l to r do begin {枚举每一个可能的子根 i}
            now:=search(l,i-1)*search(i+1,r)+f[i,i];{计算以 i 为根的子树的分值}
            if now>f[l,r] then{若该分值为目前最高，则记入状态转移方程，并记下子根}
            begin    f[l,r]:=now;way[l,r]:=i; end;{then}
              end;{for}
                 search:=f[l,r];{返回顶点 l‥顶点 r 对应子树的最高分值}
            end;{else}
end;{ search }
```

显然主程序可以通过递归调用 search(1,n)计算最高分值。算法的时间复杂度为 O(n3)

解法 2：输出加分二叉树的前序遍历

```
procedure writeout(l,r : integer);{前序遍历顶点 l‥顶点 r 对应的子树}
      begin
           if l>r then exit;
           if firstwrite then firstwrite:=false
           else write(' ');{顶点间空格分开}
           write(way[l,r]);{输出子根}
           writeout(l,way[l,r]-1);{  前序遍历左子树}
           writeout(way[l,r]+1,r);{前序遍历右子树}
      end;{writeout}
```

主程序：

```
read(n);{读顶点数}
for i:=1 to n do{状态转移方程初始化}
    for j:=i to n do f[i,j]:=-1;
for i:=1 to n do
    begin
      read(temp);{读顶点 i 的分值}
      f[i,i]:=temp;way[i,i]:=i;{顶点 i 单独成一棵子树}
    end;{for}
writeln(search(1,n));{计算和输出最高加分}
firstwrite:=true;{设立首顶点标志}
writeout(1,n);{前序遍历二叉树}
writeln;
```

【例 12】过河问题

在河上有一座独木桥，一只青蛙想沿着独木桥从河的一侧跳到另一侧。在桥上有一些石子，青蛙很讨厌踩在这些石子上。由于桥的长度和青蛙一次跳过的距离都是正整数，我们可以把独木桥上青蛙可能到达的点看成数轴上的一串整点：0，1，……，L（其中 L 是桥的长度）。坐标为 0 的点表示桥的起点，坐标为 L 的点表示桥的终点。青蛙从桥的起点开始，不停的向终点方向跳跃。一次跳跃的距离是 S 到 T 之间的任意正整数（包括 S,T）。当青蛙跳到或跳过坐标为 L 的点时，就算青蛙已经跳出了独木桥。

题目给出独木桥的长度 L，青蛙跳跃的距离范围 S,T，桥上石子的位置。你的任务是确定青蛙要想过河，最少需要踩到的石子数。

【输入文件】

输入文件 river.in 的第一行有一个正整数 L（1 <= L <= 109），表示独木桥的长度。第二行有三个正整数 S，T，M，分别表示青蛙一次跳跃的最小距离，最大距离，及桥上石子的个数，其中 1 <= S <= T <= 10，1 <= M <= 100。第三行有 M 个不同的正整数分别表示这 M 个石子在数轴上的位置（数据保证桥的起点和终点处没有石子）。所有相邻的整数之间用一个空格隔开。

【输出文件】

输出文件 river.out 只包括一个整数，表示青蛙过河最少需要踩到的石子数。

【样例输入】

10
2 3 5
2 3 5 6 7

【样例输出】

2

【数据规模】

对于 30%的数据，L <= 10000；
对于全部的数据，L <= 109。

【分析】

由于不能往回跳，很容易想到用动态规划解决这个题目。

设 f(i)表示跳到第 i 个点需要踩到的最少石子数， f(0)=0

$$f(i)=\min\begin{cases} f(i-k) & s<=k<=T,\text{当第i点没有石子} \\ f(i-k)+1 & s<=k<=T,\text{当第i点有石子} \end{cases}$$

所求的结果 Ans=min{f(L),f(L+1),……,f(L+T-1)}

这个算法的时间复杂度是 O(L*(T-S)),但本题的 L 高达 109,根本无法承受。

我们先来考虑这样一个问题：长度为 k 的一段没有石子的独木桥，判断是否存在一种跳法从一端正好跳到另一端。

若 S<T，事实上对于某个可跳步长区间[S,T]，必然存在一个 MaxK 使得任何 k>=MaxK，都可以从一端正好跳到另一端。题设中 1<=S<=T<=10，经过简单推导或者程序验证就可以发现，取 MaxK=100 就能满足所有区间。

S=T 时：

这时候由于每一步只能按固定步长跳，所以若第 i 个位置上有石子并且 i mod S=0 那么这个石子就一定要被踩到。这是我们只需要统计石子的位置中哪些是 S 的倍数即可。复杂度 O(M)

S<T 时：

首先我们作如下处理：若存在某两个相邻石子之间的空白区域长度〉MaxK+2*T，我们就将这段区域缩短成长度为 MaxK+2*T。可以证明处理之后的最优值和原先的最优值相同。

如上图所示，白色点表示连续的一长段长度大于 MaxK+2*T 的空地，黑色点表示石子。设原先的最优解中，白色段的第一个被跳到的位置 a，最后一个被跳到的位置是 b，则在做压缩处理之后，a 和 b 的距离仍然大于 MaxK，由上面的结论可知，必存在一种跳法从 a 正好跳到 b，而且不经过任何石子。

所以原来的最优解必然在处理之后的最优解解集中。

经过这样的压缩处理，独木桥的长度 L’最多为(M+1)*(MaxK+2*T)+M，大约 12000 左右。压缩之后再用先前的动态规划求解，复杂度就简化成了 O(L’*(T-S))，已经可以在时限内出解了。

这样本题就得到了解决。