第十五章 动态规划

第一节 动态规划的基础

一．动态规划简介

动态规划算法（Dynamic Programming，简称 DP）是信息学奥赛中重点考察的基本算法，每年的各类比赛中经常会有动态规划的题目出现，并且频率相当高。所以参赛选手们必须准确、熟练地掌握这一算法。

动态规划算法是解决“多阶段决策问题”的一种高效算法，它对每个出现的问题只求解一次，并将其结果保存在一张表中，以后再次遇到相同的问题时，直接从表中索取答案，避免重复计算。正是这种“不做无用功”的求解模式，大大提高了程序的效率。动态规划算法常用于解决统计类问题（统计方案总数）和最优值问题（最大值或最小值），尤其普遍用于最优化问题。

本章主要介绍了动态规划的基本概念、动态规划的问题特征以及求解方法和技巧心得，并通过分析一些典型的题目来说明动态规划类问题的基本解题思路。

二．动态规划的基本概念

下面，我们通过分析一个实例来对动态规划算法有一个初步的认识，并了解阶段、状态、决策等基本概念。

最短路线问题

如图 15-1 所示，宇宙中存在着大大小小的星球，星球之间存在着纵横交错的飞船通道。梦佳背负着巨大的使命，需要尽快从星球 A 赶到星球 E 执行任务。请你帮助她找到一条最短的路线。

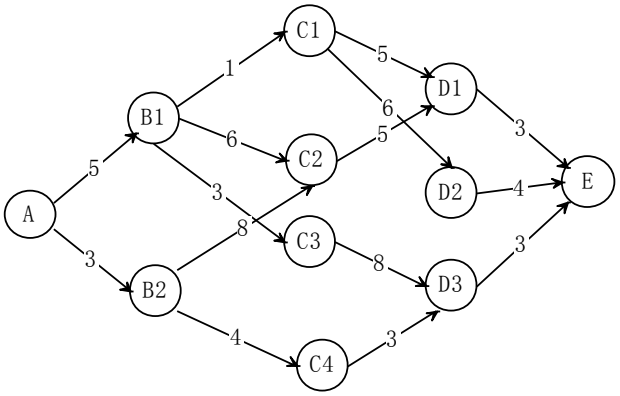

图 15-1 最短路线图

问题分析：

经过观察，容易发现本问题有一明显的特点，就是在从星球 A 到星球 E 的路径上，结点可按从左向右的顺序分为 5 部分：

第一部分：A

第二部分：B1，B2

第三部分：C1，C2，C3，C4

第四部分：D1，D2，D3

第五部分：E

显而易见，想尽快到达星球 E，必须始终从左到右单向（也就是按照第一部分到第五部分依次增加的顺序）前进，如果出现从右向左前进的路线，结果必然不是最优的。

既然只能从左向右单向前进，那么从 A 点到其他任意一点 X 的最短距离，必然经过且仅经过 X 左侧的点（设为 Y）。因此，如果能提前求得由 A 到 Y 的最短距离，那么枚举 X 的所有前驱点 Y，就可以计算出 A 到 X 的最短距离。所以，按照空间顺序从左到右的顺序划分阶段，我们就可以按阶段一步步求得从 A 到 E 的最短路径了。

算法描述 ：

定义 f[i]为 A 到 i 点的最短距离，dis[i,j]为结点 i 与结点 j 之间的距离。

我们从第一部分开始从左向右依次求 A 到其他各点的最短距离，每一部分看成一个阶段，每到一个新的阶段就把相应结点的 f[i]值求出来，直到求完第五阶段结点 E 的 f[E]，f[E]也就是我们希望得到的答案。

按照上面的思路，下面我们手动模拟，求解点 A 到点 E 的最短距离。

第一阶段：f[A] = 0;

第二阶段：f[B1] = dis[A,B1] = 5, f[B2] = dis[A,B2] = 3;

第三阶段：f[C1] = f[B1]+dis[B1,C1] = 6,

f[C2] = min{f[B1]+dis[B1,C2] , f[B2]+dis[B2,C2]}= min{5+6,3+8} = 11,

f[C3] = f[B1]+dis[B1,C3] = 5+3 = 8,

f[C4] = f[B2]+dis[B2,C4] = 3+4 = 7;

第四阶段：f[D1] = min{f[C1]+dis[C1,D1] , f[C2]+dis[C2,D1]} = min{6+5,11+5} =11,

f[D2] = f[C1]+dis[C1,D2] = 6+6 = 12,

f[D3] = min{f[C3]+dis[C3,D3], f[C4]+dis[C4,D3]} = min{8+8,7+3} = 10;

第五阶段：f[E] = min{f[D1]+dis[D1,E], f[D2]+dis[D2,E) ,f[D3]+dis[D3,E]}

= min{11+3, 12+4, 10+3} = 13;

最短路径：A-->B2-->C4-->D3-->E

最短距离：13

通过上述求解过程，我们可以看出，在本例中，对每一个出现的问题（f[i]）只求解一次，不做重复计算，以后再遇到时直接使用，这就是动态规划算法高效的原因，更是学习动态规划算法必须深刻理解的一点。

解决了上述最短路径问题，下面我们了解一下动态规划中常见的几个概念：

1．阶段

阶段是为了方便解决问题人为划分的。把问题的求解过程恰当地按一定顺序分成若干个相互联系的阶段，从而按相对应的次序去求解问题。

阶段一般依据时间或者空间划分，上面的例子就是按照空间来划分阶段的，一共 5 个阶段，A 点是第一阶段，E 点是第五阶段。在生活中有许多划分阶段的例子，如计算机的使用，可按照时间顺序划分为打开电源开关，启动计算机、正常使用计算机、用完后关机等几个阶段。

阶段划分是动态规划中最重要的环节，阶段划分不当或者错误，就会导致题目难度剧增或得出错误的结果。因此，正确划分阶段是解决动态规划问题的重中之重。另外，如果一个问题无法划分合适的阶段，那它或许就无法使用动态规划算法解决。

2．状态

通常一个阶段有多个状态，状态可以用一组数来描述。在定义状态时，要做到“不多不少恰恰好”。冗余的状态会影响程序的效率，状态残缺会导致答案错误，所以如何设计状态是近年 DP 问题考察的重点。动态规划的优化方法之一就是基于减少状态数量这一基本思路来达到降低程序复杂度，减少程序运行时间的目的。

3．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（行为）称为决策。如在图 15-1 中，由 C1 点到达 E 点有两种可选决策（C1-->D1-->E 和 C1-->D2-->E），每一阶段采取的决策都是从当前状态选择一条最短路到下一阶段。

三．动态规划问题的特征

动态规划有三大重要特征：“重叠子问题”、“最优子结构（最优化原理）”和“无后效性原则”。

1．重叠子问题

在解决整个问题前，需要先解决其子问题，要解决这些子问题，又需要先解决它们的子子问题，而这些子问题又非相互独立，它们重叠交织在一起，我们把这些重复的子问题称为重叠子问题。在图 15-1 中所示的问题中，想要求解由 A 到 E 的最短距离，需要知道由 A 到 D1、由 A 到 D2 的最短距离，而在解决这两个子问题时，又都需要知道由 A 到 C1 的最短距离，所以 A 到 C1 的最短距离就是一个重叠子问题。

动态规划算法正是利用了这种重叠子问题的性质，对每一个子问题只求解一次，而后将其解保存在一个表中，当再次碰到同样的问题时直接查表就可得到答案，不需重复计算。查表的时间是常数，远小于重新计算所花费的时间，动态规划就是利用这一特性，节省程序运行时间，大大提高了程序效率。

2．最优子结构（最优化原理）

把一个大问题划分成多个子问题，这些子问题的最优解在求解当前问题之前已经求出。如果这个大问题的最优解中包含了子问题的最优解，则称该问题具有最优子结构，也称最优化原理。在图 15-1 的例子中，A-->B2-->C4-->D3-->E 是由 A 到 E 的诸多路径中最短的一条路径，同时 A-->B2-->C4-->D3 也是由 A 到 D3 的诸多路径中最短的一条，这就满足了最优子结构原理：当前问题的最优解包含子问题的最优解。

3．无后效性原则

无后效性原则是指已经求得的状态，不受未求状态的影响。

在图 15-1 的问题中，一旦求得了某两点之间的最短距离，后面阶段的其他点就不会对他产生影响。比如一旦求得 A 到 C3 的最短距离，其值它不会受 D2 或 E 点的影响，若某值受后来值的影响，则称其存在后效性。

无后效性是运用动态规划算法的前提，因为只有无后效性的问题才能顺利划分阶段。如果无法划分阶段，自然不能使用动态规划算法了。所以，在使用动态规划算法前，必须谨慎判断所求问题是否具有后效性。

第二节 动态规划的解题步骤

动态规划问题的求解过程可分为以下二个步骤：

一．分析问题，建立模型

1．构造问题所对应的过程。

2．判断问题是否具有重叠子问题、最优子结构性质，是否符合无后效性原则。若不满足条件，则不能使用动态规划算法。

3．为问题划分阶段，设计状态。

4．根据状态建立状态转移方程（递推公式）。

5．分析时空复杂度是否满足要求。

6．如果（5）不满足要求，则对其进行优化，重写方程。

7．找出问题的边界，预处理，设定目标状态。

对于图 15-1 中求 A 点到 E 点之间的最短路问题，很明显具有重叠子问题、最优子结构性质，而且满足无后效性原则，所以可以利用动态规划解决。

按照空间划分阶段后，定义 f[i]为 A 到 i 点的最短距离，于是建立状态转移方程：

f[i]:= min(f[k]+dis[k,i])　（条件：k 是 i 的前驱）

预处理：f[A]:=0; f[其他的点]:=inf（正无穷）

目标状态: f[E]

完成上述准备工作之后，就可以求解了。

二．算法实现

1．记忆化搜索

记忆化搜索是一种直接求解的方法。在递归求解时，先判断该子问题是否求过，如果求过，就直接调用之前求过并记录下来的解，否则就递归的求解该子问题，求解完后并保存这个解，以备后用。

树型结构的问题一般多采用记忆化搜索。

2．递推

先求小的子子问题，再计算大的子问题，最终把目标问题解决。常见的动态规划问题基本上都是采用递推的方式实现。

关键：在求某个子问题时，他需要用到的子子问题都必须是已经求解过了的，否则有可能出错。

在使用递推求解问题时，根据题目的特点选择顺推还是倒推。

假设用 k 表示阶段，U 表示状态，X 表示决策，则：

顺推：

```
f[U1]=初始值；
for k←2 to n  do                    //枚举阶段
  for U 取遍所有状态 do             //枚举状态
    for X 取遍所有决策 do           //枚举决策
      f[Uk]=opt{f[Uk-1]+L[Uk-1,Xk-1]}}；//L[Uk-1,Xk-1]}：状态 Uk-1 通过策略 Xk-1 到达
```

状态 U_k 的费用

输出目标：f[U_n]。

倒推：

f[U_n]=初始值；

for k←n-1 downto 1 do //枚举阶段

for U 取遍所有状态 do //枚举状态

for X 取遍所有决策 do //枚举决策

f[U_k]=opt{f[U_{k+1}]+L[U_k,X_k]}；//L[U_k,X_k]}：状态 U_k 通过策略 X_k 到达状态 U_{k+1} 的费用

输出目标：f[U_1]。

采取什么样的方法实现好，要根据具体问题的结构特点进行选择。

第三节 动态规划的基本模型和常见方程

动态规划有多种多样的题目，但通常按照状态可分为以下几类：线性动态规划、坐标类动态规划、区间动态规划、背包资源类动态规划、树型动态规划等，每种模型都有其独特的特征。

下面我们通过分类解决一些典型的题目来掌握动态规划算法的一般求解过程。

一．线性模型

线性动态规划的状态通常是一维的（f[i]），第 i 个元素的最优值只与前 i-1 个元素的最优值或第 i+1 个元素之后的最优值有关。经典的线性 DP 题目有最长上升子序列、最大连续子序列和、最长公共子序列等。

1．最长上升子序列模型

例 15-1 导弹拦截（bumb）

【问题描述】

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统，但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都要求高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段，不能拦截所有的导弹，输入敌国导弹依次飞来的高度（雷达给出的高度数据是不大于30000 的正整数），请聪明的你帮忙计算这套系统最多能拦截多少导弹。

【输入】

第一行：n，敌国导弹的数量。

第二行：n个整数，用空格分隔，依次是敌国导弹的高度h(h<30000)。

【输出】

该拦截系统最多拦截敌国导弹的数量。

【样例输入】

8

2 7 1 9 10 1 2 3

【样例输出】

4

【样例说明】

该系统拦截的4发导弹的高度依次是：2 7 9 10。

【数据规模】

100%的数据：n<=1000。

问题分析：

经过分析，容易发现该问题的本质就是寻找导弹高度的最长上升子序列，英文缩写为LIS(Longest Increasing Subsequence)。

定义f[i]表示：从前向后，以h[i]为最后一个元素的最长上升子序列的长度。

想要求解f[i],思路是在所有满足条件(1<=j<=i-1 且 h[j]<h[i])的f[j]中取最优值。这样f[i]的取值就只与前i-1个数有关，所以可以枚举前i-1个状态进行状态转移。

DP方程也就可以写成：

f[i]:=max{f[j]}+1 (1<=j<=i-1 且 h[j]<h[i])

最终答案ans=max{f[i]} (1<=i<=n)。

时间复杂度O(n^2)，满足题目的要求。

参考程序：

```
const
  maxn=1000;
  fin='bumb.in';
  fout='bumb.out';
var
  h:array[1..maxn] of longint;
  f:array[1..maxn]of longint;
  n,ans,i,j:longint;
begin
  assign(input,fin); reset(input);
  assign(output,fout); rewrite(output);
  readln(n);
  for i:=1 to n do read(h[i]);
  fillchar(f,sizeof(f),0);
  f[1]:=1;
  for i:=2 to n do      //从前向后顺推求解
    begin
      for j:=1 to i-1 do
        if (h[j]<h[i]) and (f[j]>f[i]) then f[i]:=f[j];
      f[i]:=f[i]+1;
    end;
  ans:=1;
  for i:=2 to n do
    if f[i]>ans then ans:=f[i];
  writeln(ans);
```

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

知识拓展：

（1）DP的顺推与倒推

上述的参考程序使用的是顺推DP，从左向右寻找递增序列。相反地，倒推DP同样也可解决这个问题：从右向左寻找递减序列，完全等同于从左向右寻找递增序列。下面给出部分参考代码：

```
f[n]:=1;
for i:=n-1 downto 1 do
  begin
    for j:=i+1 to n do
      if (h[j])>h[i])and(f[j]>f[i]) then f[i]:=f[j];
    inc(f[i]);
  end;
ans:=f[1];
for  i:=2 to n do if f[i]>ans then ans:=f[i];
writeln(ans);
```

（2）其他最长子序列问题

除了本题的最长上升子序列问题，还经常会遇到如：最长下降子序列；最长不上升子序列；最长不下降子序列等一系列类似问题。算法与求LIS类似。

例 15-2 合唱队形（chorus）

【问题描述】

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 $T_1,T_2,\cdots\cdots,T_k$,则他们的身高满足 $T_1<T_2<\cdots\cdots T_i$，$T_i>T_{i+1}>\cdots\cdots>T_k$ (1<=i<=K)。

你的任务是，已知有 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】

第一行是一个整数 N(2<=N<=1000),表示同学的总数。

第二行有 N 个整数,用空格分隔,第 i 个整数 T_i(130<=T_i<=230)是第 i 位同学的身高(厘米)。

【输出】

一个整数，表示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。

【样例输入】

```
8
186 186 150 200 160 130 197 220
```

【样例输出】

```
4
```

问题分析：

计算最少需要几位同学出列，可转化为计算最多能留下多少同学。将所有合唱队形同学排为一列，在二维坐标系中根据身高描点连线，图形就像一座山峰。

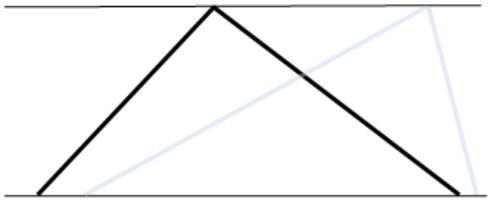

图 15-2 队形图

如上图所示，队列左边的身高依次上升，右边依次下降，中间有一个最高点，而这个最高点的左侧是最长上升子序列，右侧是最长下降子序列。满足这一条件的合唱队形就是最优合唱队形。

依据上述分析，对整个身高序列分别求一次最长上升子序列(f[i])和一次最长下降子序列(g[i])，通过枚举最高点可以求出合唱队列的最多保留人数是：

ans=max{f[i]+g[i]-1} (1<=i<=n)

从而求得最少的出列人数是 n- ans。

参考程序：

```
const
  maxn=1001;
  fin='chorus.in';
  fout='chorus.out';
var
  a:array[1..maxn] of longint;
  f,g:array[1..maxn] of longint;
  n,ans:longint;
  procedure init;
    var i:longint;
    begin
      assign(input,fin); reset(input);
      readln(n);
      for i:=1 to n do read(a[i]);
      close(input);
    end;
  procedure dp;
    var i,j:longint;
    begin
      f[1]:=1;
```

```
      for i:=2 to n do
        begin
          f[i]:=0;
          for j:=1 to i-1 do
            if (a[j]<a[i])and(f[j]>f[i]) then f[i]:=f[j];
          inc(f[i]);
        end;
      g[n]:=1;
      for i:=n-1 downto 1 do
        begin
          g[i]:=0;
          for j:=i+1 to n do
           if (a[j]<a[i])and(g[j]>g[i]) then g[i]:=g[j];
          inc(g[i]);
        end;
      ans:=0;
      for i:=1 to n do
       if f[i]+g[i]-1>ans then ans:=f[i]+g[i]-1;
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(n-ans);
      close(output);
    end;
begin
  init;
  dp;
  print;
end.
```

例 15-3 上帝选人（select）

【问题描述】

世界上的人都有智商和情商。我们用两个数字来表示人的智商和情商，数字大就代表其相应的属性（智商或情商）高。现在你面前有 N 个人，这 N 个人的智商和情商均已知。

请你选择出尽量多的人，满足选出的人中不存在任意两人 i 和 j，i 的智商大于 j 的智商而 i 的情商小于 j 的情商。

【输入】

第一行一个正整数 N，表示人的数量。

第二行至第 N+1 行，每行两个正整数，分别表示每个人的智商和情商。

【输出】

仅一行，为最多选出的人的个数。

【样例输入】

3

100 100

110 80

120 90

【样例输出】

2

【数据规模】

对于 100%的数据，N<=1000。

问题分析：

将原题中的“不存在任意两人 i 和 j,i 的智商大于 j 的智商而 i 的情商小于 j 的情商。”转化为等价问题：

用 iq 表示智商，eq 表示情商，在所有选出的人中，如果 iq[i] > iq[j]，则 eq[i] >= eq[j] 恒成立。

这样，也就找到了问题的解决方法：

先将所有人的iq（或者eq）递增排序，然后求eq（或者iq）的最长非递减子序列长度。

参考程序：

```
const
  fin='select.in';
  fout='seledt.out';
type
  node=record
    iq,eq:longint;
    end;
var
  a:array[0..1001] of node;
  f:array[0..1001] of longint;
  n,len:longint;
  procedure init;
    var i:longint;
    begin
      assign(input,fin);reset(input);
      readln(n);
      for i:=1 to n do readln(a[i].iq,a[i].eq);
    end;
  procedure qsort(l,r:longint);
    var i,j,x,y:longint;
    begin
      i:=l;j:=r;x:=a[l].iq;y:=a[l].eq;
```

```
    while i<j do
      begin
        while ((a[j].iq>x)or((a[j].iq=x)
              and(a[j].eq>y)))and(i<j) do dec(j);
        if i<j then begin a[i]:=a[j]; inc(i);end;
        while ((a[i].iq<x)or((a[i].iq=x)
              and(a[i].eq<y)))and(i<j) do inc(i);
        if i<j then begin a[j]:=a[i]; dec(j);end;
      end;
    a[i].iq:=x;a[i].eq:=y;
    if i-1>l then qsort(l,i-1);
    if i+1<r then qsort(i+1,r);
  end;
function max(a,b:longint):longint;
  begin
    if a>b then exit(a);exit(b);
  end;
procedure dp;
  var i,j:integer;
  begin
    len:=0;
    for i:=1 to n do
      begin
        for j:=1 to i-1 do
          if (a[j].eq <= a[i].eq) and (f[j]>f[i]) then f[i]:=f[j];
        inc(f[i]);
        if f[i]>len then len:=f[i];
      end;
  end;
procedure print;
  begin
    assign(output,fout);rewrite(output);
    writeln(len);
    close(output);
  end;
BEGIN
  init;
  qsort(1,n);
  dp;
  print;
```

END.

例 15-4 最大递增子序列和（sum）

【问题描述】

给定长度为n的正整数序列$a_1, a_2, \cdots, a_n$。

令$sum=a_{b1}+a_{b2}+\cdots+a_{bm}$，并且满足：$a_{b1}<a_{b2}<\cdots<a_{bm}$；b1<b2<…<bm；1<=m<=n。

求最大的sum。

【输入】

第一行，n，表示给定序列中整数的个数。

第二行，n个用空格隔开的正整数。

【输出】

最大的sum。

【样例输入】

6

2 4 1 20 5 6

【样例输出】

26

【样例说明】

sum=2+4+20=26。

【数据规模】

100%的数据：n<=1000, $0<a_i<=10^9$。

问题分析：

本题是LIS的简单变形：由求最大个数变为求最大和，除了求解的目标状态不同，其他参数（转移条件、方程定义）与LIS基本一样。

定义f[i]表示以第i个数a[i]为最后一个数的最大递增子序列和，则状态转移方程可以写为：

f[i]:=max{f[j]}+a[i]　(1<=j<i<=n)and(a[j]<a[i])

目标：ans=max{f[i]}　(1<=i<=n)。

参考程序：

```
const
  maxn=1000;
  fin='sum.in';
  fout='sum.out';
var
  a:array[0..maxn] of longint;
  f:array[0..maxn] of longint;
  n,ans:longint;
  procedure init;
    var i:longint;
    begin
      assign(input,fin); reset(input);
```

```
      readln(n);
      for i:=1 to n do read(a[i]);
      close(input);
    end;
  function max(a,b:longint):longint;
    begin if a>b then exit(a) else exit(b); end;
  procedure dp;
    var i,j:longint;
    begin
      fillchar(f,sizeof(f),0);
      ans:=0; a[0]:=-1;
      for i:=1 to n do
        for j:=0 to i-1 do
          if (a[j]<a[i]) then
          f[i]:=max(f[i],int64(f[j])+a[i]);
      for i:=1 to n do if f[i]>ans then ans:=f[i];
      writeln(ans);
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(ans);
      close(output);
    end;
begin
  init;
  dp;
  print;
end.
```

2．最大连续子序列和模型

例 15-5 最大连续子序列和（maxsub）

【问题描述】

对于给定的一个有序序列，求出其最大连续子序列的和。

【输入】

第一行：n。

第二行：n 个整数（-3000，3000）。

【输出】

最大连续子序列的和。

【样例输入】

7

-6 4 -1 3 2 -3 2

【样例输出】

8

【样例说明】

最大连续子序列是4 -1 3 2，和为8。

【数据规模】

n<=1000000。

问题分析：

根据题目中 n<=1000000 的范围限制，要想解决此题，必须寻找时间复杂度为 O(n)的有效算法。容易想到采用动态规划的方法来解决该问题。

用 a[i]保存序列。

定义 f[i]：以 a[i]为连续子序列右边界（连续子序列的最右一个元素）的最大连续子序列的和。

容易得出方程：

f[i]=max{f[i-1]+a[i],a[i]} (1<=i<=n)

初始： f[1]=a[1]。

目标： max{f[i]} (1<=i<=n)。

参考程序：

```
//求最大连续子序列的和
const
  maxn=10000;
  fin='maxsub.in';
  fout='maxsub.out';
var
  a:array[1..maxn] of longint;
  f:array[1..maxn] of longint;
  n,i:longint;
  ans:longint;
  procedure init;
    begin
      assign(input,fin); reset(input);
      readln(n);
      for i:=1 to n do read(a[i]);
      close(input);
    end;
  function max(a,b:longint):longint;
    begin max:=a; if b>max then max:=b; end;
  procedure dp;
    begin
      f[1]:=a[1];
```

```
      for i:=2 to n do
        f[i]:=max(f[i-1]+a[i],a[i]);
      ans:=f[1];
      for i:=2 to n do if f[i]>ans then ans:=f[i];
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(ans);
      close(output);
    end;
begin
  init;
  dp;
  print;
end.
```

3．最长公共子序列模型

最长公共子序列也称作最长公共子串，英文缩写为 LCS（Longest Common Subsequence）。

其定义是：一个序列 S ，如果分别是两个或多个已知序列的子序列，且是所有符合此条件序列中最长的，则 S 称为已知序列的最长公共子序列(不要求一定连续)。

例 15-6 最长公共字符子序列（lcs）

【问题描述】

给定两个字符序列:

$X = \{ x_1 , x_2 , \dots , x_m \}$

$Y = \{ y_1 , y_2 , \dots , y_n \}$

求 X 和 Y 的一个最长公共子序列长度。

举例：

$X = \{ a , b , c , b , d , a , b \}$

$Y = \{ b , d , c , a , b , a \}$

其中一个最长公共子序列为：$LCS = \{ b , c , b , a \}$，长度是 4。LCS 可能不止一个。

【输入】

字符串 X。

字符串 Y。

【输出】

最长公共子串的长度。

【样例输入】

```
abcbdab
bdcaba
```

【样例输出】

```
4
```

【数据规模】

100%的数据：X 和 Y 的长度<=1000。

问题分析：

考虑：

$X = \{ x_1 , \ldots , x_{i-1} , x_i \}$

$Y = \{ y_1 , \ldots , y_{j-1} , y_j \}$

定义 f[i,j]为 X 的前 i 个字符和 Y 的前 j 个字符中最大公共子序列的长度。

当 $x_i = y_j$ 时：f[i,j] = f[i-1,j-1] + 1；

当 $x_i <> y_j$时：f[i,j] = max { f[i,j-1]； f[i-1,j] }

本题还有一个需要注意的地方是不要忽略字符串的长度限制，在定义字符串类型时要用 ansistring 类型，不能使用 string 类型；当然本题也可采用字符类型实现。

参考程序：

```
const
  maxn=1000;
  fin='lcs.in';
  fout='lcs.out';
var
  f:array[0..maxn,0..maxn] of longint;
  x,y:ansistring;
  n,m:longint;
  procedure init;
    begin
      assign(input,fin); reset(input);
      readln(x);
      readln(y);
      n:=length(x);
      m:=length(y);
      close(input);
    end;
  function max(a,b:longint):longint;
    begin if a>b then exit(a) else exit(b); end;
  procedure main;
    var i,j:longint;
    begin
      for i:=1 to n do
        for j:=1 to m do
          begin
            f[i,j]:=0;
            if x[i]=y[j] then f[i,j]:=f[i-1,j-1]+1
            else f[i,j]:=max(f[i-1,j],f[i,j-1]);
```

```
          end;
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(f[n,m]);
      close(output);
    end;
begin
  init;
  main;
  print;
end.
```

例 15-7　字符串转换（change）

【问题描述】

设A和B是两个字符串。我们可以通过下面的三种字符操作将字符串A转换为字符串B。对串A的字符操作包括以下三种：

(1)删除一个字符；

(2)插入一个字符；

(3)将一个字符改为另一个字符。

对于给定的字符串A和B，要求用最少的操作步数将A串转换为B串。

【输入】

第一行：A串。

第二行：B串。

【输出】

将A串转换为B串所用的最少步数。

【样例输入】

```
acdef
abcde
```

【样例输出】

```
2
```

【数据规模】

A和B串的长度不超过3000。

问题分析：

设f[i,j]为将A的前i个字符变成B的前j个字符所用的最少操作步数。如：

```
A=‘acdef’
B=‘abcde’
i=length(A)
j=length(B)
```

从后向前依次比较分析，f[i,j]有以下三种情况：

1)删除 A 中的最后一个字符

问题变为将 A 中前 i-1 个字符转换为 B 中的前 j 个字符。

则：f[i,j]＝f[i-1,j]+1

2)在 A 的最后插入一个字符

插入后使 a[i+1]=b[j],问题变为将 A 中的前 i 个字符转换为 B 中的前 j-1 个字符。

则：f[i,j]＝f[i,j-1]+1

3)将 A 中的一个字符转换为另一个字符。

如果 a[i]=b[j],则 f[i,j]＝f[i-1,j-1]；

如果 a[i]<>b[j],将 a[i]换成 b[j]；则 f[i,j]＝f[i-1,j-1]+1。

综上所述，得出方程：

```
f[i,j]＝min{
          f[i-1,j]+1,
          f[i,j-1]+1,
          f[i-1,j-1]    (a[i]=b[j]),
          f[i-1,j-1]+1    (a[i]<>b[j]) }
```

令：

```
n=length(A);
m=length(B);
```

初始条件：

f[i,0]=i （0<=i<=n），删除 A 中的 i 个字符使 A 变成 B；

f[0,j]=j （0<=j<=m），在 A 中插入 j 个字符使 A 变成 B。

目标：f[n,m]。

参考程序：

```
const
  maxn=3000;
  fin='change.in';
  fout='change.out';
var
  a,b:ansistring;
  f:array[0..maxn,0..maxn] of longint;
  n,m:longint;
  procedure init;
    begin
      assign(input,fin); reset(input);
      readln(a);
      readln(b);
      n:=length(a);
      m:=length(b);
      close(input);
    end;
```

```
  function min(a,b:longint):longint;
    begin if a<b then exit(a) else exit(b); end;
  procedure main;
    var i,j:longint;
    begin
      for i:=0 to n do f[i,0]:=i;
      for i:=0 to m do f[0,i]:=i;
      for i:=1 to n do
        for j:=1 to m do
          if a[i]=b[j] then f[i,j]:=f[i-1,j-1]
          else f[i,j]:=min(min(f[i-1,j],f[i,j-1]),f[i-1,j-1])+1;
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(f[n,m]);
      close(output);
    end;
begin
  init;
  main;
  print;
end.
```

二．坐标类模型

坐标类动态规划是比较简单的一类动态规划问题，此类问题常以二维空间坐标系为模板展开，因此状态转移方程也在坐标系中寻找，一般定义 f[i,j]表示坐标位置(i,j)的状态。

例 15-8　数字三角形（triangle）

【题目描述】

有一个数字三角形，由 R 行组成，第 i 行有 i 个自然数。

起初你在第一行唯一的数字上，每次可以向左下走或右下走，走到最后一行。你所经过的数字之和是你的分值，求你最多可以得到多少分。

如下数字三角形：

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

从 7 开始出发，最大得分是 30，路线依次是：7 3 8 7 5。

【输入】

第一行一个整数 R。

接下来 R 行，第 i 行 i 个整数，表示数字三角形中的数。

【输出】

得到的最大分值。

【样例输入】

5

7

3 8

8 1 0

2 7 4 4

4 5 2 6 5

【样例输出】

30

【数据规模】

R <= 1000，数字三角形中的所有数字不超过 100。

题目分析：

对于这道题目，朴素做法是搜索尝试每一条道路，记录其分值，取最大值输出。但是无法通过 100%的数据，原因是搜索到（i,j）点时的最优值未做记录，导致以后再搜（i,j）点时做了重复的操作。为此，我们需要记录搜索到某一个点时的最优值，以备后用。

在这里，我们介绍两种方法：记忆化搜索和递推。

定义 f[i,j]为从(i,j)位置走到最后一行的最大得分，容易得出方程：

f[i,j]:=max{f[i+1,j],f[i+1,j+1]}+a[i,j]

初始值：f[n,i]=a[n,i]。

目标：f[1,1]（所求的最大得分）。

1．记忆化搜索：

参考程序:

```
const
  maxn=100;
  fin='triangle.in';
  fout='triangle.out';
var
  n:longint;
  a,f:array[0..100,0..100] of longint;
  procedure init;
    var i,j:longint;
    begin
      assign(input,fin); reset(input);
      fillchar(a,sizeof(a),0);
      fillchar(f,sizeof(f),$ff);//-1
```

```
      readln(n);
      for i:=1 to n do
        for j:=1 to i do  read(a[i,j]);
      close(input);
    end;
  function max(a,b:longint):longint;
    begin if a>b then exit(a) else exit(b);end;
  procedure dfs(i,j:integer);
    begin
      if i=n then f[i,j]:=a[i,j];
      if i>=n then exit;
      if f[i,j]>=0 then exit;
      dfs(i+1,j);
      dfs(i+1,j+1);
      f[i,j]:=max(f[i+1,j],f[i+1,j+1])+a[i,j];
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(f[1,1]);
      close(output);
    end;
Begin
   init;
   dfs(1,1);
   print;
End.
```

2．递推：

参考程序：

```
const
  maxn=100;
  fin='triangle.in';
  fout='triangle.out';
var
  n,ans:longint;
  a,f:array[0..100,0..100] of longint;
  procedure init;
    var i,j:longint;
    begin
      assign(input,fin); reset(input);
```

```
    fillchar(a,sizeof(a),0);
    readln(n);
    for i:=1 to n do
      for j:=1 to i do  read(a[i,j]);
    close(input);
  end;
function max(a,b:longint):longint;
  begin if a>b then exit(a) else exit(b); end;
procedure dp;  //顺推
  var i,j:longint;
  begin
    for i:=1 to n do f[n,i]:=a[n,i];
    for i:=n-1 downto 1 do
      for j:=1 to i do
        f[i,j]:=max(f[i+1,j],f[i+1,j+1])+a[i,j];
    ans:=f[1,1];
  end;
procedure print;
  begin
    assign(output,fout);rewrite(output);
    writeln(ans);
    close(output);
  end;
begin
  init;
  dp;
  print;
end.
```

例 15-9 马拦过河卒（horse）

【问题描述】

棋盘上 A 点有一个过河卒，需要走到目标 B 点。卒行走的规则：可以向下、或者向右。同时在棋盘上 C 点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。

棋盘用坐标表示，A 点(0, 0)、B 点(n, m)(n, m 为不超过 15 的整数)，同样马的位置坐标是需要给出的。

现在要求你计算卒从 A 点能够到达 B 点的路径的条数。

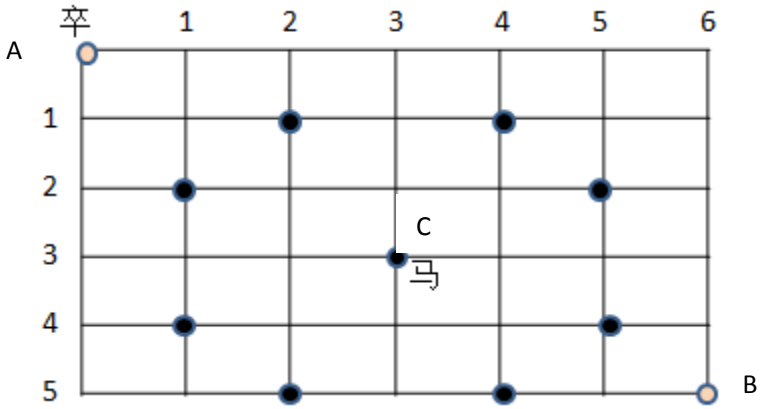

图 15-3 样例图

【输入】

一行四个数，分别表示 B 点坐标和马的坐标。

【输出】

一个数，表示卒从 A 点能够到达 B 点的路径的条数。

【样例输入】

6 6 3 3

【样例输出】

6

问题分析：

本题中比较重要的一句就是“卒行走的规则：可以向下、或者向右。”,这是解决本题的关键。

卒只可向下或向右走，所以到达任何一个点只能由它上方或左方过来，因此，在计算到达某点的路径总数时，需要先计算到达其上方和左方点的路径总数，两者之和就是到达此点的路径总条数了。

设 f[i,j]表示从 A 到达[i,j]点的路径总数，则有方程：

f[i,j]= f[i-1,j]+f[i,j-1]　　(1<=i<=n,1<=j<=m)

特别的，对于马能控制到的点，到达其路径总数应该为 0，所以在程序实现时要注意特别判断。

参考程序 ：

```
const
  fin='horse.in';
  fout='horse.out';
var
  n,m:longint;
  f:array[-2..15,-2..15] of longint;
  can:array[-2..17,-2..17] of boolean;
  x,y:longint;
  procedure init;
```

```
    var i,j:longint;
    begin
      assign(input,fin); reset(input);
      readln(n,m,x,y);
      close(input);
      fillchar(can,sizeof(can),true);
      can[x,y]:=false;
      can[x+2,y+1]:=false; can[x+2,y-1]:=false;
      can[x+1,y+2]:=false; can[x+1,y-2]:=false;
      can[x-1,y-2]:=false; can[x-1,y+2]:=false;
      can[x-2,y+1]:=false; can[x-2,y-1]:=false;
    end;
  procedure dp;
    var i,j:longint;
    begin
      fillchar(f,sizeof(f),0);
      f[0,0]:=1;
      can[0,0]:=false;
      for i:=0 to n do
        for j:=0 to m do
          if  can[i,j] then
          f[i,j]:=f[i-1,j]+f[i,j-1];
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(f[n,m]);
      close(output);
    end;
Begin
  init;
  dp;
  print;
End.
```

例 15-10 公共汽车（bus）

【问题描述】

一个城市的道路构成了像棋盘那样的网状，南北向的路有n条，并由西向东从1标记到n,东西向的路有m条，并从南向北从1标记到m。每一个交叉点代表一个路口，有的路口有正在等车的乘客。一辆公共汽车将从(1,1)点驶到（n,m）点，车只能向东或者向北开。

写一个程序，告诉司机怎么走能接到最多的乘客。

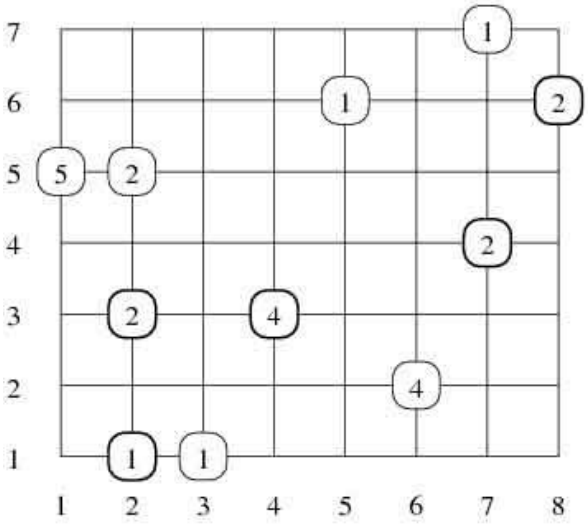

图 15-4 道路图

【输入】

第一行是n,m,和k,其中k是有乘客的路口的个数。

以下k行是有乘客的路口的坐标和乘客的数量。

已知每个路口的乘客数量不超过1000000。

【输出】

汽车能接到的最多的乘客数量。

【样例输入】

8 7 11

4 3 4

6 2 4

2 3 2

5 6 1

2 5 2

1 5 5

2 1 1

3 1 1

7 7 1

7 4 2

8 6 2

【样例输出】

11

【数据规模】

1 <= n <= 10^3, 1 <= m <= 10^3, 1 <= k <= 10^3。

问题分析：

用 a[i,j]表示坐标 (i,j)位置的人数， f[i,j]表示汽车从位置（1,1）走到位置（i，

j）能接的最多人数,则方程：

f[i,j]:=max{f[i-1,j],f[i,j-1]}+a[i,j]　（1<=i<=n,1<=j<=m）

目标：f[n,m]。

参考程序：

```
const
  maxn=1000;
  fin='bus.in';
  fout='bus.out';
var
  a:array[1..maxn,1..maxn] of longint;
  f:array[0..maxn,0..maxn] of longint;
  n,m,k:longint;
  procedure init;
    var i,j,x,y,z:longint;
    begin
      assign(input,fin); reset(input);
      readln(n,m,k);
      for i:=1 to k do
        begin
          readln(x,y,z);
          a[x,y]:=z;
        end;
      close(input);
    end;
  function max(a,b:longint):longint;
    begin if a>b then exit(a) else exit(b); end;
  procedure dp;
    var i,j:longint;
    begin
      fillchar(f,sizeof(f),0);
      for i:=1 to n do
        for j:=1 to m do
          f[i,j]:=max(f[i-1,j],f[i,j-1])+a[i,j];
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(f[n,m]);
      close(output);
    end;
```

```
begin
  init;
  dp;
  print;
end.
```

例 15-11 最大正方形（matrix）

【题目描述】

给定一个 R 行 C 列的 01 矩阵，求一个最大的正方形全 1 子矩阵，并输出该最大正方形子矩阵的面积。

【输入】

第一行给出两个正整数 R,C，表示矩阵有 R 行 C 列。

接下来 R 行 C 列给出这个 01 矩阵，行内相邻两元素用一个空格隔开。

【输出】

最大正方形全 1 子矩阵的的面积。

【样例输入】

```
5 8
0 0 0 1 1 1 0 1
1 1 0 1 1 1 1 1
0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0
1 1 1 0 1 1 0 1
```

【样例输出】

```
9
```

【数据规模】

20%的数据，R,C<=20；

40%的数据，R,C<=100；

100%的数据，R,C<=1000。

问题分析：

定义 f[i,j]为以(i,j)为右下角顶点的最大正方形边长。

当 a[i,j]=1 时，f[i,j] =min{ f[i,j-1],
                          f[i-1,j],
                          f[i-1,j-1] }+1

最大正方形边长：ans=max{f[i,j]} (1<=i<=R，1<=j<=C)。

最大正方形面积：ans*ans。

参考程序：

```
const
  maxn=1000;
  fin='matrix.in';
  fout='matrix.out';
var
```

```
  a:array[0..maxn+1,0..maxn+1] of longint;
  f:array[0..maxn+1,0..maxn+1] of longint;
  n,m,i,j,x,ans:Longint;
  procedure init;
    begin
      assign(input,fin); reset(input);
      readln(n,m);
      fillchar(a,sizeof(a),0);
      for i:=1 to n do
        for j:=1 to m do  read(a[i,j]);
      close(input);
    end;
  function min(a,b:longint):longint;
    begin if a>b then exit(b) else exit(a); end;
  procedure main;
    begin
      fillchar(f,sizeof(f),0);
      for i:=1 to n do
        for j:=1 to m do if a[i,j]=1 then
          f[i,j]:=min(min(f[i,j-1],f[i-1,j]),f[i-1,j-1])+1;
      ans:=0;
      for i:=1 to n do
        for j:=1 to m do
          if f[i,j]>ans then ans:=f[i,j];
      ans:=ans*ans;
    end;
  procedure print;
    begin
      assign(output,fout); rewrite(output);
      writeln(ans); close(output);
    end;
begin
  init;
  main;
  print;
end.
```

三．区间模型

区间型动态规划是线性动态规划的拓展，它将区间长度作为阶段，长区间的最优值依赖于短区间的最优值。区间型动态规划的典型应用是石子合并。

例 15-12 线性石子合并（stone）

【问题描述】

设有 N 堆石子排成一排，其编号为 1，2，3，…，N（N<=300）。每堆石子有一定的数量，可以用一个整数来描述，现在要将这 N 堆石子合并成为一堆，每次只能合并相邻的两堆，合并的代价为这两堆石子的数量之和，合并时由于选择的顺序不同，合并的总代价也不相同。

问题是：找出一种合理的合并方法，使总的代价最小，输出最小代价。

【输入】

第一行一个数 N 表示石子的堆数。

第二行 N 个数，表示每堆石子的数量（不超过 1000）。

【输出】

合并的最小代价。

【样例输入】

4

1 3 5 2

【样例输出】

22

问题分析：

第一次看到此题，很多同学会采取贪心的方法，每次选相邻石子和最小的两堆进行合并。

对于样例 1 3 5 2：

第一次合并 1、2 堆，代价为 4，得到 4 5 2；

第二次合并 2、3 堆，代价为 7，得到 4 7；

第三次合并最后的两堆得到 11。

3 次合并后总代价为 4+7+11=22。

恰好与样例答案一样，这造就了一种用贪心策略即可解决本题的假象，导致很多同学使用贪心策略。事实上，这种算法是存在反例的。

例如 4 堆石子，分别为 7 4 4 7，采用贪心算法（每次选相邻石子和最小的两堆进行合并）得到的最小得分是 45，而事实上正确的最小得分是 44，所以本题采用贪心算法是错误的。

从合并的最后一步可以看出，合并第 1 堆至第 n 堆，可以分为{合并 1 到 K 堆，合并 k+1 到 N 堆}两部分，最优答案就可以枚举 k 获得，而这两互不相干的步骤又可以再次从中间断开继续由更小的合并方案得来。一个大合并区间由两个小合并区间得来，小合并区间又由两个更小的区间得来，因此，我们可以肯定本题具有重叠子问题和最优子结构性质。

所以，本题可以使用动态规划算法。

考虑状态的定义，既然合并的区间有起点 i 和终点 j，就定义 f[i,j]表示将 i 至 j 区间合并为一堆的最小代价。

由上面的分析可以得到状态转移方程：

f[i,j]:={f[i,k]+f[k+1,j]}+sum[i,j] （i<=k<=j-1）

其中 sum[i,j]表示 a[i]+a[i+1]+…+a[j-1]+a[j]，而 k 的枚举范围需要根据边界判断，左边界：f[i,k]=f[i,i]，此时 k=i；右边界：f[k+1,j]=f[j,j]，此时 k=j-1 所以 i<=k<=j-1。

此题与前面讲到的例题不同，之前的例题 i 与 j 的枚举顺序都十分清晰，但本题却不明

显。

这里介绍两种求解方法：

方法 1：记忆化搜索

参考程序：

```
const
  maxn=100;
  fin='stone.in';
  fout='stone.out';
var
  a:array[1..maxn]of longint;
  sum,f:array[1..maxn,1..maxn]of longint;
  n:longint;
  procedure init;
    var i,j:longint;
    begin
      assign(input,fin);
      reset(input);
      readln(n);
      for i:=1 to n do read(a[i]);
      for i:=1 to n do
        begin
          sum[i,i]:=a[i];
          for j:=i+1 to n do sum[i,j]:=sum[i,j-1]+a[j];
        end;
      close(input);
      fillchar(f,sizeof(f),0);
    end;
  function min(a,b:longint):longint;
    begin if a<b then exit(a) else exit(b); end;
  function dfs(i,j:longint):longint;
    var k:longint;
    begin
      if i=j then exit(0);
      if f[i,j]>0 then exit(f[i,j]);
      f[i,j]:=maxlongint;
      for k:=i to j-1 do
        f[i,j]:=min(f[i,j],dfs(i,k)+dfs(k+1,j)+sum[i,j]);
      exit(f[i,j]);
    end;
  procedure print;
```

```
    begin
      assign(output,fout); rewrite(output);
      writeln(f[1,n]);
      close(output);
    end;
begin
  init;
  dfs(1,n);
  print;
end.
```

方法 2：以区间长度为阶段的 DP

参考程序：

```
const
  maxn=100;
  fin='stone.in';
  fout='stone.out';
var
  a:array[1..maxn]of integer;
  sum,f:array[1..maxn,1..maxn]of longint;
  n:integer;
  procedure init;
    var i,j:longint;
    begin
      assign(input,fin);
      reset(input);
      readln(n);
      for i:=1 to n do read(a[i]);
      for i:=1 to n do
        begin
          sum[i,i]:=a[i];
          for j:=i+1 to n do sum[i,j]:=sum[i,j-1]+a[j];
        end;
      close(input);
    end;
  procedure dpwork;
    var i,j,p,k:longint;
    begin
      for i:=1 to n do f[i,i]:=0;
      for p:=1 to n-1 do
        for i:=1 to n-p do
```

```
            begin
              j:=i+p;
              f[i,j]:=maxlongint;
              for k:=i to j-1 do
                if f[i,j]>f[i,k]+f[k+1,j]
                then f[i,j]:=f[i,k]+f[k+1,j];
              f[i,j]:=f[i,j]+sum[i,j];
            end;
      end;
    procedure print;
      begin
        assign(output,fout);
        rewrite(output);
        writeln(f[1,n]);
        close(output);
      end;
  begin
    init;
    dpwork;
    print;
  end.
```

方法 1 与方法 2 得出的答案是一样的，方法 1 利用的记忆化搜索，在第一次求解 f[i,j] 时递归求解，之后的求解直接读表，不再递归，提高了算法效率。

方法 2 是常规的递推 DP，以区间长度 p 为阶段，i 是区间的左端点，区间右端点 j 由左端点和区间长度 p 计算得出，并不枚举；而大区间答案需要小区间的支持，以区间长度递增枚举。

思维延伸：环形石子合并

若本例将“有 N 堆石子排成一排”改为“有 N 堆石子围成一圈”，题目如何解决？

一种方法是将环形展开呈直线形即可：

原模型下标：1 2 3 4... n-1 n。

新模型下标：1 2 3 4... n-1 n 1 2 3 4 ... n-1。

对应石子数量：a[1],a[2],...,a[n],a[n+1],...,a[2n-1] （满足 a[n+i]=a[i]）。

DP 方程与例题一样，只是最终求解时枚举区间的两端点即可：

ans=min{f[1,n],f[2,n+1],...,f[n,2n-1]}

例 15-13 加分二叉树（tree）

【问题描述】

设一个 n 个结点的二叉树 tree 的中序遍历为（1,2,3,…,n），其中数字 1,2,3,…,n 为结点编号。每个结点都有一个分数（均为正整数），记第 i 个结点的分数为 d_i，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，叶子的加分就是叶结点本身的分数，不考虑它的空子树。

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】

第1行：一个整数n（n＜30），为结点个数。

第2行：n个用空格隔开的整数，为每个结点的分数（分数＜100）。

【输出】

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。

【样例输入】

5

5 7 1 2 10

【样例输入】

145

3 1 2 4 5

问题分析：

1.最高加分

设f[i,j]为中序遍历结果序列为i..j时的最大加分。

枚举根结点k：i<=k<=j，得到不同的二叉树:

左子树中序遍历结果是（i,…,k-1），最大加分是f[i,k-1]；

右子树中序遍历结果是（k+1,…,j），最大加分是f[k+1,j]。

得到方程：

f[i,j]=max{f[i,k-1]*f[k+1,j]+a[k]}　（i+1<=k<=j-1）

初始化：

f[i,j]=1；

f[i,i]=a[i]。

目标：f[1,n]}。

2.输出先序遍历序列

任意中序遍历结果为(i..j)的二叉树中，一旦确定其根结点k，二叉树就唯一确定了。

在求f[i，j]的同时，记下(i..j)的根k，用root[i,j]=k。

求得了所有的root[i,j]后，递归调用root[1,n]即可输出先序遍历序列。

参考程序：

```
const
  maxn=30;
  fin='tree.in';
  fout='tree.out';
var
  a:array[1..maxn] of integer;
```

```
root:array[1..maxn,1..maxn] of integer;       //保存根结点
f:array[0..maxn,0..maxn]of longword;
n:integer;
procedure init;
  var i:integer;
  begin
    assign(input,fin);
    reset(input);
    readln(n);
    for i:=1 to n do read(a[i]);
    close(input);
  end;
procedure dp;
  var i,j,p,k:integer;
  begin
    fillchar(f,sizeof(f),0);
    for i:=0 to n do
      for j:=0 to n do f[i,j]:=1;
    for i:=1 to n do
      begin f[i,i]:=a[i];root[i,i]:=i;end;
    for p:=1 to n-1 do
      for i:=1 to n-p do
        begin
          j:=i+p;
          f[i,j]:=0;
          for k:=i to j do                          //枚举根接点
            if f[i,j]<f[i,k-1]*f[k+1,j]+f[k,k] then
              begin
                f[i,j]:=f[i,k-1]*f[k+1,j]+f[k,k];
                root[i,j]:=k;
              end;
        end;
  end;
procedure preorder(i,j:integer);
  begin
    if i<=j then
      begin
        write(root[i,j],' ');
        preorder(i,root[i,j]-1);
        preorder(root[i,j]+1,j);
```

```
        end;
    end;
procedure print;
  begin
    assign(output,fout);rewrite(output);
    writeln(f[1,n]);
    preorder(1,n);
    close(output);
  end;
BEGIN
  init;
  dp;
  print;
END.
```

例 15-14 括号序列（brack）

【问题描述】

我们用以下规则定义一个合法的括号序列：

（1）空序列是合法的；

（2）假如 S 是一个合法的序列，则 (S) 和[S]都是合法的；

（3）假如 A 和 B 都是合法的，那么 AB 和 BA 也是合法的。

如：

以下是合法的括号序列：

(), [], (()), ([]), ()[], ()[()]

以下是不合法的括号序列：

(, [,],)(, ([)], ([()

现在给定一些由'(', ')', '[', , ']'构成的括号序列 ，请添加尽量少的括号，得到一个合法的括号序列。

【输入】

输入括号序列 s。含最多 100 个字符（四种字符： '(', ')', '[' , ']' ），都放在一行，中间没有其他多余字符。

【输出:】

使括号序列 s 成为合法序列需要添加最少的括号数量。

【样例输入】

([()

【样例输入】

2

【样例说明】

最少好添加 2 个括号可以得到合法的序列，如：()[()]或([()])或()[]()。

问题分析：

本题是比较典型的区间 DP 问题，区间 DP 的状态转移常从最后一步向前考虑。

设序列 S=$S_iS_{i+1}..S_{j-1}S_j$最少添加 f[i,j]个括号能变成规则的括号序列。
根据不同结构，可以分为以下 4 种不同情况来处理：
1）S 形如（S′）或[S′]：
只需把 S′变规则即可,则 f[i,j]= f[i+1,j-1]。
2）S 形如（S′或[S′：
先把 S′化为规则的，右边加“)”或“]”即可，则 f[i,j]= f[i+1,j]+1。
3）S 形如 S′）或 S′]：
先把 S′化为规则的，左边加“(”或“[”即可，则 f[i,j]= f[i,j-1]+1
4）把长度大于 1 的序列 $S_iS_{i+1}..S_{j-1}S_j$分为两部分:
$S_i..S_k$，$S_{k+1}..S_j$，分别化为规则序列，则
f[i,j]=f[i,k]+f[k+1,j] ； i<=k<=j-1;
上述 4 种情况取最小值即可。
动态规划方程：

```
f[i,j]:=min{    f[i+1,j-1]
                    ((s[i]='(')and(s[j]=')')) or ((s[i]='[')and(s[j]=']')) ,
           f[i+1,j]+1          (s[i]='(')or(s[i]='['),
           f[i,j-1]+1          (s[j]=')')or(s[j]=']'),
           f[i,k]+f[k+1,j]     (i<=k<=j-1)
           }
```

初始化：f[i,i]=1。
目标：f[1,length(s)]。
参考程序：

```
const
  maxn=100;
  fin='brack.in';
  fout='brack.out';
  k1='(';k2=')';k3='[';k4=']';
var
  f:array[0..maxn+1,0..maxn+1] of longint;
  s:string;
  n:integer;
  procedure init;
    begin
      assign(input,fin);reset(input);
      readln(s);
      n:=length(s);
      close(input);
    end;
  function min(a,b:longint):longint;
     begin
```

```
        if a<b then exit(a);exit(b);
      end;
  procedure dp;
    var i,j,p,k:longint;
    begin
      for i:=1 to n do f[i,i]:=1;
      for p:=1 to n-1 do
        for i:=1 to n-p do
          begin
            j:=i+p;
            f[i,j]:=maxlongint;
            if (s[i]=k1) or (s[i]=k3) then f[i,j]:=min(f[i,j],f[i+1,j]+1);
            if (s[j]=k2) or (s[j]=k4) then f[i,j]:=min(f[i,j],f[i,j-1]+1);
            if ((s[i]=k1)and(s[j]=k2))or((s[i]=k3)and(s[j]=k4))
                then f[i,j]:=min(f[i,j],f[i+1,j-1]);
            for k:=i to j-1 do
              f[i,j]:=min(f[i,j],f[i,k]+f[k+1,j]);
          end;
    end;
  procedure print;
    begin
      assign(output,fout);rewrite(output);
      writeln(f[1,n]);
      close(output);
    end;
Begin
  init;
  dp;
  print;
End.
```

四．背包模型

背包模型有很多，这里只讲最简单的 0-1 背包（1 件物品要么选，要么不选，不能只选一部分）基本模型。

例 15-5 0-1 背包问题（bag）

【问题描述】

设有 n 种物品，每种物品有一个重量及一个价值，同时有一个背包，最大载重量为 m，今从 n 种物品中选取若干件，使其总重量的和不超过 m，而价值的和最大。n<=100,m<1000。

【输入】

第一行两个数：物品总数 n，背包载重量 m，两个数用空格分隔。

第二行 n 个数,为 n 种物品重量 w_i(w_i<1000)，两个数用空格分隔。

第三行 n 个数,为 n 种物品价值 v_i(v_i<1000)，两个数用空格分隔。

【输出】

背包能装物品的最大总价值。

【样例输入】

4 10

3 4 5 7

7 15 20 25

【样例输入】

35

问题分析：

用 f[i,j]表示从第 1 件到第 i 件物品中选择若干件装入载重量为 j 的背包中获得的最大价值。

考察第 i 件物品有选和不选两种情况：

1) f[i-1,j]:不选第 i 件物品获得的价值。

2) f[i-1,j-w[i]]+v[i]：选第 i 件获得的价值。

条件：j>=w[i]，即背包的剩余空间还能装下物品 i。

得到方程：

f[i,j]=max{f[i-1,j]，f[i-1,j-w[i]]+v[i] （j>=w[i]）}

(1<=i<=n,1<=j<=m)

初始值： f[1,j]=v[1] (j>=w[1]);

f[1,j]=0 (j<w[1])。

目标： f[n,m]。

参考程序：

```
const
  maxn=100;
 maxw=1000;
  fin='bag.in';
  fout='bag.out';
var
  f:array[0..maxn,0..maxw]of longint;
  w,v:array[1..maxn]of integer;
  i,j,k,l,m,n:integer;
  procedure init;
    begin
      assign(input,fin);reset(input);
      readln(n,m);
      for i:=1 to n do read(w[i]);
      for i:=1 to n do read(v[i]);
      close(input);
```

```
  end;
procedure work;
  begin
    fillchar(f,sizeof(f),0);
    for i:=1 to n do
      for j:=1 to m do
        begin
          f[i,j]:=f[i-1,j];
          if (j>=w[i]) and (f[i,j]<f[i-1,j-w[i]]+v[i])
          then f[i,j]:=f[i-1,j-w[i]]+v[i];
        end;
  end;
procedure print;
  begin
    assign(output,fout); rewrite(output);
    writeln(f[n,m]);
    close(output);
  end;
begin
  init;
  work;
  print;
end.
```

五．树型 DP

例 15-16 拾金子（gold）

【问题描述】

古老的传说中有一个古老的游戏，游戏的名字叫拾金子。

游戏的规则如下：

有一树型道路，树中每一结点都有一个标号，同时有一块标有质量的金子，游戏者从最上边根结点出发，遍历若干结点，每经过一个结点都必须拿走该结点的金子。现在规定游戏者拿走的金子数目是有限的，问怎样走才能使得到的金子质量最大？

具体问题：

一棵有 n 个结点的树（结点标号是 1..n），从中找 m 个点，使这些点连通且包含根结点，并使得所有点的权值（金子质量）和最大。（如下图中如果包含 10 号结点，则必须包含 9 号和 1 号结点，因为到达 10 时必须经过 9 和 1 结点）。所求的一定是含有 m 个结点(包含根)的最大连通分支。

【输入】

第一行： n, m。

以下 n 行,每行依次是 n 个结点的：标号、父结点、金子质量。父亲结点为 0 的结点是

根结点。

【输出】

得到的金子最大质量。

【样例输入】

10 5
2 9 9
1 9 1
4 2 1
5 2 1
9 0 1
6 9 1
10 1 20
3 2 1
8 6 6
7 6 4

【样例输入】

32

【数据规模】

40% 的数据：1<=n<=10；

100%的数据：1<=n<=100，1<=m<=n，1<=金子重量<=1000。

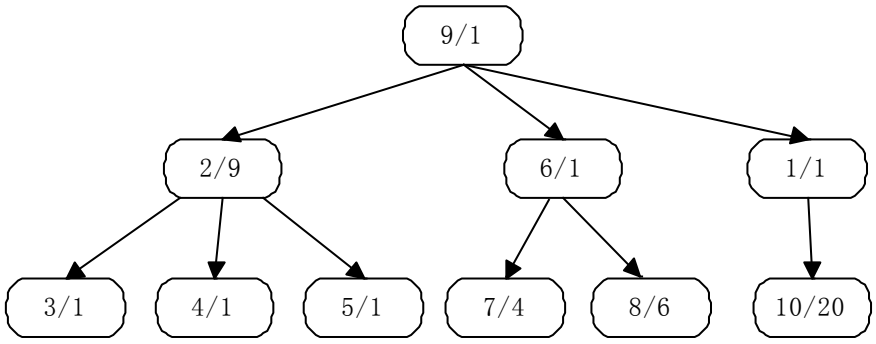

图 15-5 结点中第一个数是结点标号，第二个数是金子重量

问题分析：

本题的结构模型是一棵树，应用在树结构上的动态规划也称为树型动规，是常见的一类动态规划题目。本题目就是求含有树根的共有 m 个结点的最大连通分支。

解决树型动规问题时一般是把多叉树转化为二叉树，以便减少编程的复杂度，但要搞清楚转化后的二叉树和原来多叉树的结点对应关系。

注意：本题所求的二叉树不一定是连通的。

定义 f[i,y]是以 i 为根的子树，含有 y 个结点的最优值。

容易得出方程：

f[i,y]=max{

f[r[i],y], //不选结点 i，只能从右子树（对应原多叉树 i 的兄弟）中选 y 个结点

f[l[i],k]+a[i]+f[r[i],y-k-1] （0<=k<=y-1）//选结点 i，左子树选 k 个结点

}

参考程序：

```
const
  maxn=100;
  fin='gold.in';
  fout='gold.out';
var
  left,right:array[0..maxn] of longint;
  a:array[1..maxn] of longint;
  f:array[0..maxn,0..maxn] of longint;
  n,m,root:longint;
  procedure init;
    var i,j,k:longint;
    begin
      assign(input,fin); reset(input);
      fillchar(f,sizeof(f),0);
      readln(n,m);
      for k:=1 to n do
        begin
          read(i,j);readln(a[i]);
          if j=0 then root:=i else
            if left[j]=0 then left[j]:=i
            else
              begin
                right[i]:=left[j];
                left[j]:=i;
              end;
        end;
      close(input);
    end;
  function max(a,b:longint):longint;
    begin if a>b then exit(a) else exit(b); end;
  procedure tdp(v,x:longint);
    var  i:longint;
    begin
      if (v=0)or(x=0) then exit;
      if f[v,x]>0 then exit;
```

```
      tdp(right[v],x);
      f[v,x]:=f[right[v],x];
      for i:=0 to x-1 do
        begin
          tdp(left[v],i);
         tdp(right[v],x-i-1);
          f[v,x]:=max(f[v,x],f[left[v],i]+a[v]+f[right[v],x-i-1]);
        end;
    end;
  procedure main;
    begin
      assign(output,fout); rewrite(output);
      tdp(root,m);
      writeln(f[root,m]);
      close(output);
    end;
begin
  init;
  main;
end.
```