动态规划

关健字：阶段 状态 决策 函数递推式

摘要：

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。动态规划思想近来在各类型信息学竞赛中频繁出现，它的应用也越来越受人重视。本文就是讨论如何运用动态规划的思想设计出有效的数学模型来解决问题。

一 动态规划问题的数学描述

我们先来看一个简单的多阶段决策问题。

[例 1]现有一张地图，各结点代表城市，两结点间连线代表道路，线上数字表示城市间的距离。如图 1 所示，试找出从结点 1 到结点 10 的最短路径。

第一阶段 第二阶段 第三阶段　第四阶段　第五阶段

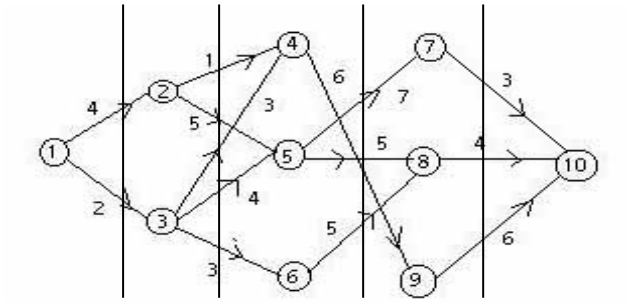

图 1

本问题的解决可采用一般的穷举法，即把从结点 1 至结点 10 的所有道路列举出来，计算其长度，再进行比较，找出最小的一条。虽然问题能解决，但采用这种方法，当结点数增加，其运算量将成指数级增长，故而效率是很低的。

分析图 1 可知，各结点的排列特征：

(1) 可将各结点分为 5 个阶段；

(2) 每个阶段上的结点只跟相邻阶段的结点相连，不会出现跨阶段或同阶段结点相连的情况，如不会出现结点 1 与结点 4 连、结点 4 与结点 5 连的情况。

(3) 除起点 1 和终点 10 外，其它各阶段的结点既是上一阶段的终点，又是下一阶段的起点。例如第三阶段的结点 4、5、6，它即是上一阶段结点 2、3 中某结点的终点，又是下一阶段结点 7、8、9 中某结点的起点。

根据如上特征，若对于第三阶段的结点 5，选择 1-2-5 和 1-3-5 这两条路径，后者的费用要小于前者。那么考虑一下，假设在所求的结点 1 到结点 10 最短路径中要经过结点 5，那我们在结点 1 到结点 5 之间会取那条路径呢？显然，无论从结点 5 出发以后的走法如何，前面选择 1-3-5 这条路都总是会优于 1-2-5 的。也就是说，当某阶段结点一定时，后面各阶段路线的发展不受这点以前各阶段的影响。反之，到该点的最优决策也不受该点以后的发展影响。

由此，我们可以把原题所求分割成几个小问题，从阶段 1 开始，往后依次求出结点 1 到阶段 2、3、4、5 各结点的最短距离，最终得出答案。在计算过程中，到某阶段上一结点的决策，只依赖于上一阶段的计算结果，与其它无关。例如，已求得从结点 1 到结点 5 的最优值是 6，到结点 6 的最优值是 5，那么要求到下一阶段的结点 8 的最优值，只须比较 min{6+5,5+5}即可。这样，运用动态规划思想大大节省了计算量。可以看出，动态规划是解决此类多阶段决策问题的一种有效方法。

二 动态规划中的主要概念，名词术语

1 阶段：把问题分成几个相互联系的有顺序的几个环节，这些环节即称为阶段。

2 状态：某一阶段的出发位置称为状态。通常一个阶段包含若干状态。如图 1 中，阶段 3 就有三个状态结点 4、5、6。

3 决策：从某阶段的一个状态演变到下一个阶段某状态的选择。

4 策略：由开始到终点的全过程中，由每段决策组成的决策序列称为全过程策略，简称策略。

5 状态转移方程：前一阶段的终点就是后一阶段的起点，前一阶段的决策选择导出了后一阶段的状态，这种关系描述了由 k 阶段到 k+1 阶段状态的演变规律，称为状态转移方程。

6 目标函数与最优化概念：目标函数是衡量多阶段决策过程优劣的准则。最优化概念是在一定条件下找到一个途径，经过按题目具体性质所确定的运算以后，使全过程的总效益达到最优。

三 运用动态规划需符合的条件

任何思想方法都有一定的局限性，超出了特定条件，它就失去了作用。同理，动态规划也并不是万能的。那么使用动态规划必须符合什么条件呢？必须满足最优化原理和无后效性。

1 最优化原理

最优化原理可这样阐述：一个最优化策略具有这样的性质，不论过去状

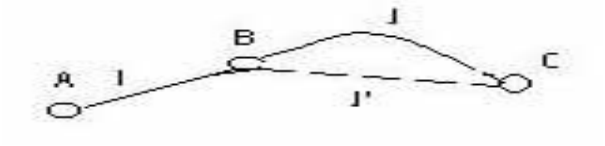

图 2

态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的。

如图 2 中，若路线 I 和 J 是 A 到 C 的最优路径，则根据最优化原理，路线 J 必是从 B 到 C 的最优路线。这可用反证法证明：假设有另一路径 J′是 B 到 C 的最优路径，则 A 到 C 的路线取 I 和 J′比 I 和 J 更优，这与原名题矛盾。从而证明 J′必是 B 到 C 的最优路径。

最优化原理是动态规划的基础，任何问题，如果失去了最优化原理的支持，就不可能用动态规划方法计算。

2 无后效性

“过去的步骤只能通过当前状态影响未来的发展，当前的状态是历史的总结”。这条特征说明动态规划只适用于解决当前决策与过去状态无关的问题。状态，出现在策略任何一个位置，它的地位相同，都可实施同样策略，这就是无后效性的内涵。

由上可知，最优化原理，无后效性，是动态规划必须符合的两个条件。

四 动态规划的计算方法

对于一道题，怎样具体运用动态规划方法呢?

（1） 首先，分析题意，考察此题是否满足最优化原理与无后效性两个条件。

（2） 接着，确定题中的阶段，状态，及约束条件。

（3） 推导出各阶段状态间的函数基本方程，进行计算。

具体求解则有多种方法。

1 前向与后向动态规划法

所谓前向与后向，指的是从起点出发，层层递推，直到终点，或从终点出发，逆向求解。这两种方法本质上是一样的，具体解题时，可根据实际情况来选用。

[例 2] 排队买票

问题描述：一场演唱会即将举行。现有 N（O〈N〈=200）个歌迷排队买票，一个人买一张，而售票处规定，一个人每次最多只能买两张票。假设第 I 位歌迷买一张票需要时间 Ti（1〈=I〈=n），队伍中相邻的两位歌迷（第 j 个人和第 j+1 个人）也可以由其中一个人买两张票，而另一位就可以不用排队了，则这两位歌迷买两张票的时间变为 Rj，假如 Rj〈Tj+Tj+1，则这样做就可以缩短后面歌迷等待的时间，加快整个售票的进程。现给出 N，Tj 和 Rj，求使每个人都买到票的最短时间和方法。

解决问题：本题的阶段十分明显，只要按排队的先后顺序划分即可。而买票

的方式只有两种，要么一人买一张，要么一人买两张，整个过程呈线性排列。要使前 I 个人买票时间最短，只需考虑前 I 个人的买票方式，与队列以后的人无关。而且显而易见，在最优策略中，任意 m 个连续的决策也肯定是最优的。这样，问题就符合了最优化原理及无后效性，能运用动态规划。那如何构造函数递推式呢？可以以到每个人为止所需的最短时间为状态值，设为 f（i），于是有 f（i）=min{f（i-1）+Ti，f（i-2）+Ri-1}，起步时 f（0）=0，f（1）=T1。如此从前往后，只需遍历一次即可。

上面的分析是从前往后进行的。其实倒过来逆推也一样。设 f（I）表示当票卖到第 I 个人时，最少还需多少时间才能卖完。则函数递推式为 f（I）=min{f（I+1）+Ti，f（I+2）+Ri}，从后往前逆推，起步时 f（n）=Tn，f（n+1）=0。

采用前向还是后向动态规划，要看实际情况而定，哪一种直观、简便，就运用哪一种。

2 具有隐含阶段的问题（即阶段不明显）

动态规划的一个重要环节是阶段划分，可有些题目无明显阶段，但也符合最优化原理，怎么解决呢？下面来看一道例题。

[例 3] 最小费用问题

问题描述：已知从 A 到 J 的路线及费用如图 3，求从 A 到 J 的最小费用路线。

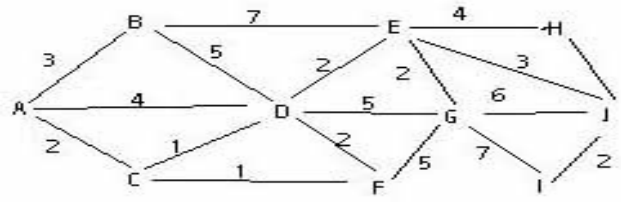

图 3

解决问题：本问题没有明显的阶段划分，各点间没有一定的先后次序，不能按照最少步数来决定顺序，如从 A 到 D 走捷径需 4，但 A-C-D 只需 3，更优。看来图中出现回路，不能实施动态规划。其实不然。细想一下，从 A 到 J 的最优策略，它每一部分也是最优的（可以用前述的反证法来证明），换言之，本题也具有最优化性质，只是阶段不明显而已。

对于这类问题，我们可以换个角度分析，构造算法。比较一下前面所讲的前向与后向动态规划法，都是以某个状态为终点，寻找到达次点的路径，然后比较优劣，确定此状态最优值。可是，本题阶段不明显，各状态之间的道路会出现嵌套，故此法不能使用。变一下角度，每次都以某个状态为起点，遍历由它引申出去的路径，等所有已知状态都扩展完了，再来比较所有新状态，把值最小的那个状态确定下来，其它的不动。如图 3，先从 A 出发，找到 3 个结点 B，D，C，费用为 F（B）=3，F（D）=4，F（C）=2。因为 F（B），F（D）都大于 F（C），那么可以确定：不可能再有路线从 B 或 D 出发到 C，比 A-C 更优。这样 F（C）的最优值便确定了。可是，有没有路线从 C 出发到 B 或 D，比 A-B 或 A-D 更优呢？还不清楚。继续下去，因为 A 扩展完了，只有从 C 开始，得到 A-C-D=3，A-C-F=3，于是 F（D）的值被刷新了，

等于3。现在，有F（B）=F（D）=F（F）=3，于是，三点的最优值都确定下来了。然后以分别以三个点为起点，继续找。以次类推，直到J点的最优值确定为止。

细心观察，其实本题的隐含阶段就是以各结点的最优值的大小来划分的，上述过程就是按最优值从小到大前向动态规划。人们习惯上把此题归入到图论范畴中，并将上述方法称为标号法。

五 动态规划的应用

上文叙述了动态规划的几种解法，下面我们通过例题来加深了解。

[例4] 复制书稿（BOOKS）

问题描述：假设有M本书（编号为1，2，…M），想将每本复制一份，M本书的页数可能不同（分别是P1，P2，…PM）。任务时将这M本书分给K个抄写员（K〈=M〉，每本书只能分配给一个抄写员进行复制，而每个抄写员所分配到的书必须是连续顺序的。

意思是说，存在一个连续升序数列0=bo〈b1〈b2〈…<bk-1 <bk=m，这样，第I号抄写员得到的书稿是从bi-1+1到第bi本书。复制工作是同时开始进行的，并且每个抄写员复制的速度都是一样的。所以，复制完所有书稿所需时间取决于分配得到最多工作的那个抄写员的复制时间。试找一个最优分配方案，使分配给每一个抄写员的页数的最大值尽可能小（如存在多个最优方案，只输出其中一种）。

解决问题：该题中M本书是顺序排列的，K个抄写员选择数也是顺序且连续的。不管以书的编号，还是以抄写员标号作为参变量划分阶段，都符合策略的最优化原理和无后效性。考虑到K〈=M，以抄写员编号来划分会方便些。

设F（I，J）为前I个抄写员复制前J本书的最小“页数最大数”。于是便有F（I，J）=MIN{ F（I-1，V），T（V+1，J）}　（1〈=I〈=K，I〈=J〈=M-K+I，I-1〈=V〈=J-1〉。其中T（V+1，J）表示从第V+1本书到第J本书的页数和。起步时F（1，1）=P1。

观察函数递推式，发现F（I）阶段只依赖于F（I-1）阶段的状态值，编程时可令数组F的范围为（0…1,1…M），便于缩小空间复杂度。

[例5] 多米诺骨牌（DOMINO）

问题描述：有一种多米诺骨牌是平面的，其正面被分成上下两部分，每一部分的表面或者为空，或者被标上1至6个点。现有一行排列在桌面上：

顶行骨牌的点数之和为6+1+1+1=9；底行骨牌点数之和为1+5+3+2=11。顶行和底行的差值是2。这个差值是两行点数之和的差的绝对值。每个多米诺骨牌都可以上下倒置转换，即上部变为下部，下部变为上部。

现在的任务是，以最少的翻转次数，使得顶行和底行之间的差值最小。对于上面这个例子，我们只需翻转最后一个骨牌，就可以使得顶行和底行的差值为0，所以例子的答案为1。

解决问题：例子的上下部分之差是6+1+1+1-（1+5+3+2）=（6-1）+（1-5）+（1-3）+（1-2）=-2，而翻转最后一个骨牌后，上下之差变

为（6-1）+（1-5）+（1-3）+（2-1）=0。由此看出，一个骨牌对翻转策略造成影响的是上下两数之差，骨牌上的数则是次要的了。这么一来，便把骨牌的放置状态由 8 个数字变为 4 个：5　-4　-2　-1，翻转时只需取该位数字的相反数就行了。

在本题中，因为各骨牌的翻转顺序没有限定，所以不能按骨牌编号作为阶段来划分。怎么办呢？考虑到隐含阶段类型的问题可以按状态最优值的大小来划分阶段。于是，我们以骨牌序列上下两部分的差值 I 作为状态，把达到这一状态的翻转步数作为状态值，记为 f（I）。便有 f（I）=min{f（I+j）+1}（-12〈=j<=12,j 为偶数，且要求当前状态有差值为 j/2 的骨牌）。这里，I 不是无限增大或减小，其范围取决于初始骨牌序列的数字差的和的大小。

具体动态规划时，如例题，我们以 f（-2）=0 起步，根据骨牌状态，进行一次翻转，可得到 f（-12）=1，f（6）=1，f（2）=1，f（0）=1，由于出现了 f（0），因此程序便可以结束，否则将根据四个新状态继续扩展，直至出现 f（0）或者无法生成新状态为止。

注意：在各状态，除记录最少步数外，还需记录到达这一状态时各骨牌的放置情况；而当到达某一状态发现已记录有一种翻转策略时，则取步数较小的一种。

六 小结

动态规划是一种高效算法。若能成功运用，必会使程序效率，无论时间还是空间，都有着质的飞跃。因此，我们视其为算法思想中的一项重要内容，有熟练掌握，深入研究的必要。希望本文能给大家一点启发。

[附录]

例 4：
输入格式：
文件的第一行是两个整数 m 和 k (1〈=k〈=m〈=500)。
第二行有 m 个整数 P1，P2，…，Pm，这 m 个整数均为正整数且都不超过 1000000。每两个整数之间用空格分开。

输出格式：
文件有 k 行，每行有两个正整数。整数之间用空格分开。
第 I 行的两个整数 ai 和 bi，表示第 I 号抄写员所分配得到的书稿的起始编号与终止编号。

程序如下：

```
program books;
type tp=array[1..500] of integer;
```

```
    tc=array[1..500] of longint;
var c:array[1..500] of ^tp;
   {记录路径}
f:array[0..1] of tc;
{状态值}
t:tc;
{书本页数和}
cc:tp;
{链接路径}
   i,j,v,k,m,x,y,min,p1,p2:longint;
   f1,f2:text;
procedure init;
{输入部分}
begin
  assign(f1,'books.dat');
  reset(f1);
  assign(f2,'books.out');
  rewrite(f2);
  readln(f1,m,k);
  if k=1 then
  begin
    writeln(f2,1,' ',m);
    close(f2);
    halt;
  end;
  {当 k=1 时,作特殊处理}
  for i:=1 to m do
  begin
    read(f1,j);
    if i=1 then t[1]:=j else
      t[i]:=t[i-1]+j;
    {累加页数}
  end;
  for i:=1 to k do
    new(c[i]);
end;
procedure main;
{主过程}
begin
  p1:=1;f[1]:=t;
  {起步状态}
  for i:=2 to k-1 do
  {利用函数递推式计算}
  begin
```

```
    p2:=p1;p1:=1-p2;
    for j:=i to m-k+i do
    begin
      min:=maxlongint;y:=0;
      for v:=i-1 to j-1 do
      begin
        if f[p2,v]>t[j]-t[v] then x:=f[p2,v] else x:=t[j]-t[v];
        if x<min then begin min:=x;y:=v;end;
      end;
      f[p1,j]:=min;
      c[i]^[j]:=y;
    end;
  end;
  p2:=p1;p1:=1-p2;
  min:=maxlongint;y:=0;
  for i:=k-1 to m-1 do
  begin
    if f[p2,i]>t[m]-t[i] then x:=f[p2,i] else x:=t[m]-t[i];
    if x<min then begin min:=x;y:=i;end;
  end;
  {最后找出最优分配方案}
  for i:=k-1 downto 1 do
  begin
    cc[i]:=y;
    y:=c[i]^[y];
  end;
  {链接路径}
  writeln(f2,1,' ',cc[1]);
  for j:=2 to k-1 do
  writeln(f2,cc[j-1]+1,' ',cc[j]);
  writeln(f2,cc[k-1]+1,' ',m);
  close(f2);
  {打印}
end;
begin
  init;
  main;
end.
```

例 5：

输入格式：

文件的第一行是一个整数 n（1〈=n〈=1000〉，表示有 n 个多米诺骨牌在桌面上排成一行。接下来共有 n 行，每行包含两个整数 a、b（0〈=a、b〈=6，中间用空格分开〉。第 I+1 行的 a、b 分别表示第 I 个多米诺骨牌的上部与下部的点

数（0 表示空）。

输出格式:

只有一个整数在文件的第一行。这个整数表示翻动骨牌的最少次数，从而使得顶行和底行的差值最小。

程序如下:

```
program domino;
type tp=array[1..6] of integer;
var t:array[1..6000] of ^tp;
   {记录骨牌摆放状态}
f:array[-6000..6000] of integer;
{记录达到某个差值的最少步数}
l:array[1..6000] of integer;
{扩展队列}
tt:tp;
   i,j,n,m,x,y,ft,re:integer;
   f1,f2:text;
procedure init;
{程序初始化}
begin
  assign(f1,'domino.dat');
  reset(f1);
  assign(f2,'domino.out');
  rewrite(f2);
  m:=0;
  ft:=0;re:=1;new(t[1]);
  fillchar(t[1]^,sizeof(t[1]^),0);
  fillchar(f,sizeof(f),0);
  fillchar(tt,sizeof(tt),0);
  readln(f1,n);
  for i:=1 to n do
  begin
    readln(f1,x,y);
    if x<>y then
    begin
      x:=x-y;
      inc(m,x);
      inc(tt[abs(x)]);
      if x>0 then inc(t[1]^[x]);
    end;
  end;
  if m=0 then
```

```
  begin
    writeln(f2,0);
    close(f2);
    halt;
  end;
  {处理步数为零的情况}
  l[1]:=m;
  f[m]:=1;
end;
procedure main;
{主过程}
begin
  repeat
for ft:=ft+1 to re do
{以步数为阶段扩展状态}
    begin
      x:=l[ft];
      for i:=1 to 6 do
      {不同差值的六种情况}
      begin
        if x<6 then
        if (t[ft]^[i]<tt[i])and(f[x+i*2]=0) then
        begin
          inc(re);l[re]:=x+i*2;
          f[l[re]]:=f[x]+1;
          if l[re]=0 then
          {找到解便打印}
          begin
            writeln(f2,f[l[re]]-1);
            close(f2);
            halt;
          end;
          new(t[re]);
          t[re]^:=t[ft]^;
          inc(t[re]^[i]);
        end;
        {差值增加}
        if x>-6 then
        if (t[ft]^[i]>0)and(f[x-i*2]=0) then
        begin
          inc(re);l[re]:=x-i*2;
          f[l[re]]:=f[x]+1;
          if l[re]=0 then
          {找到解便打印}
```

```
          begin
            writeln(f2,f[l[re]]-1);
            close(f2);
            halt;
          end;
          new(t[re]);
          t[re]^:=t[ft]^;
          dec(t[re]^[i]);
        end;
        {差值减少}
      end;
    end;
  until ft=re;
  for i:=1 to 6 do
  if (f[i]>0)or(f[-i]>0) then
  begin
    if (f[-i]>0)and((f[i]=0)or(f[-i]<f[i])) then x:=f[-i]
      else x:=f[i];
    writeln(f2,x-1);
    i:=6;
  end;
  close(f2);
  {骨牌上下两行点数之和的绝对值不为零}
end;
begin
  init;
  main;
end.
```

参考数目:

1　《运筹学----规划及网络》 王永县 编著 清华大学出版社

2　《组合数学的算法与程序设计》 王建德 编著 清华大学出版社

3 《中学信息学奥林匹克习题解析》张欣研 李冬梅 北京大学出版社