国际大学生程序设计竞赛试题与算法分析(三)

动态规划及其应用——最短路问题

郭嵩山*，陈明睿**

(中山大学信息科技学院计算机科学系，广州 510275)

动态规划实际上是研究一类最优化问题的方法，在经济、工程技术、企业管理、工农业生产及军事等领域中都有广泛的应用。近年来，在ACM/ICPC中，使用动态规划（或部分应用动态规划思维)求解的题不仅常见，而且形式也多种多样。而在与此相近的各类信息学竞赛中，应用动态规划解题已经成为一种趋势，这和动态规划的优势不无关系。

与其说动态规划是一种算法，不如说是一种思维方法来得贴切。因为动态规划没有固定的框架，即便是应用到同一道题上，也可以建立多种形式的求解算法。许多隐式图上的算法，例如求单源最短路径的Dijkstra算法、广度优先搜索算法，都渗透着动态规划的思想。还有许多数学问题，表面上看起来与动态规划风马牛不相及，但是其求解思想与动态规划是完全一致的。正因为如此，今后我们还将会分几个专题，对动态规划的应用分门别类地进行介绍。

动态规划的概念和基础

什么是动态规划？动态规划是运筹学的一个分支。1951年美国数学家R. Bellman等人，根据一类多阶段问题的特点，把多阶段决策问题变换为一系列互相联系的单阶段问题，然后逐个加以解决。一些静态模型，只要人为地引进“时间”因素，分成时段，就可以转化成多阶段的动态模型，用动态规划方法去处理。与此同时，他提出了解决这类问题的“最优化原理”(Principle of optimality)。

“一个过程的最优决策具有这样的性质：即无论其初始状态和初始决策如何，其今后诸策略对以第一个决策所形成的状态作为初始状态的过程而言，必须构成最优策略”。简言之，一个最优策略的子策略，对于它的初态和终态而言也必是最优的。

这个“最优化原理”如果用数学化一点的语言来描述的话，就是：假设为了解决某一优化问题，需要依次作出n个决策$D_1, D_2, \cdots, D_n$，如若这个决策序列是最优的，对于任何一个整数k，$1<k<n$，不论前面k个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策D_{k+1}，$D_{k+2}, \cdots, D_n$也是最优的。

最优化原理是动态规划理论的基础。动态规划实际上是一种寻找组合最优化问题的最优解的方法，它将一个待求的最优化问题分解成几个子问题，先寻找子问题的局部最优解，并用逆推公式把原问题的最优解与子问题的局部最优解联系起来，最后获得所求问题的最优解。最优化原理说起来很简单，如何灵活应用它则又是另一回事。它用途很广，由于运用最优化原理来解决的问题本身并没有固定不变的算法，需要不断地探索其规律并进行归纳和总结。因此，利用它来解决问题的本身也就是一种创造。我们下面希望通过讲解动态规划的

* 1999~2000ACM/ICPC 亚洲(上海)赛区中山大学二队教练

**1999~2000ACM/ICPC 亚洲(上海)赛区中山大学二队主力队员

一个重要应用——最短路问题，来帮助读者初步掌握它的技巧。

最短路问题

所谓最短路问题是指给定起点及终点，并知道由起点到终点的各种可能的路径，问题是要找一条由起点到终点的最短的路，即长度最短的路。需要指出的是最短路问题中的“长度”可以是通常意义下的距离，也可以是运输的时间或者运输费用等等。而且，有些与运输根本没有关系的问题也可以化为求最短路的模型，例如求关键路径（实际上是AOE网中的最长路）问题等。

由起点A到终点E的路线图如图1所示，连接两地之间的路的长度用连线上的数字表示，求由起点A到终点E的最短路。

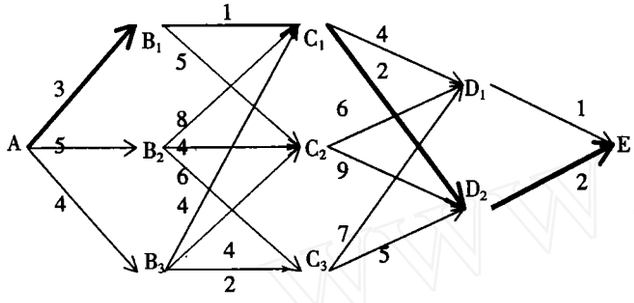


图1

一般地，求一个具有n个结点的图的最短路，如果使用穷举法，其运算量是n的指数函数。当n比较大，同时图的深度较深时，这个算法在时间效率上是不可取的。如果用最优化原理来思考这个问题，我们可以注意到最短路有这样一个特性，即如果最短路的第k站通过P_k，则这一最短路在由P_k出发到达终点的那一部分路径，对于起点为P_k到终点所有可能的路径来说，必定也是长度最短的。

引入几个记号。记$f[P_k]$表示由点P_k到E的最短路的长度，例如，f[A]表示由A到E的最短路的长度，$f[B_1]$表示由B_1到E的最短路的长度。$x(P_k)$表示从点P_k出发到下一步所选的点P_{k+1}的集合。$d[P_k,P_{k+1}]$表示点P_k到下一步所选的点P_{k+1}的长度，例如，$d[A,B_1]$表示由A到B_1的距离，即$d[A,B_1]=3$。则最短路这一特性可描述为

$$f[P_k]=\min\{d[P_k,P_{k+1}]+f[P_{k+1}] \mid P_{k+1}\in x(P_k)\}$$

显然有$f[E]=0$，而$f[P_k]$ ($P_k\neq E$)是未知的，我们将此作为初始的已知条件。根据最短路这一特性，我们很容易可以得到一个递归形式的算法来求解最短路问题：

```
procedure ShortestDistance(Pk)
//求由 Pk 到 E 的最短路的长度
begin
  if f[Pk]已求出 then
return(f[Pk])
  else
begin
对所有 Pk+1∈x(Pk) do ShortestDistance(Pk+1);
f[Pk] := min{d[Pk,Pk+1]+f[Pk+1] | Pk+1∈x(Pk)};
  end;
end;
```

调用ShortestDistance (A)，求出的最短路为$A\blacktriangleright B_1\blacktriangleright C_1\blacktriangleright D_2\blacktriangleright E$，长度为8，如图1中粗线所示。

我们实际上是将求解A到E的最短路问题(f[A])分解成结构相同的若干个子问题($f[B_1]$, $f[B_2]$, …, f[E])，这些子问题之间可能有重叠，在求解的过程中，如果某个需要求解的子问题已经解出来了，就直接取其结果；如果还没有解出来，就递归进行求解，然后通过这些子问题的解来求出原问题的解。其结果，我们不仅得到了原问题的解，而且得到了一连串子问题的解。动态规划之所以比穷举法高效的原因，就在于它对每个子问题只求解一次。

与穷举法相比，动态规划的方法有两个明显的优点：

(1)大大减少了计算量；

(2)丰富了计算结果。

从上面例子的求解结果中，我们得到的不仅是由起点A出发到终点E的最短路，而且还得到了从所有各中间点到终点的最短路，这对许多实际问题来讲是很有用的。

你也许会想，动态规划是不是分而治之呢？其实，虽然在分析许多问题时，我们都使用了这种将大问题分解成小问题的思路，但是动态规划不是分治法，这一点我们将在以后的专题中加以分析。

另一种表示形式：递推

上面我们用递归的形式描述了最短路问题的求解方法。既然每个子问题只求解一次，我们是不是可以确定出一个求解子问题顺序，用递推的方法来求解最短路问题呢？其实这样的顺序是很容易确定的，例如E, D_1, D_2, C_1, C_2, C_3, B_1, B_2, B_3, A。事实

上，如果把这个路线图看成是一个有向带权图的话，它的任一个拓扑序列的逆序都可以作为求解子问题顺序的序列。于是我们就得到了一个求最短路问题的递推形式的算法：

令 $S=(S_1,S_2,\cdots,S_n)$是某个求解子问题顺序的序列，而且 S_1 且已经解出：

```
for i := 2 to n do
  f[Si] := min{d[Si,Sj]+f[Sj] | Sj ∈ x(Si)};
```

我们现在用手工来计算一下，你就会对整个递推过程有更加清晰的理解了：

f[E]=0(终态，已知)
$f[D_1]=1+f[E]=1$
$f[D_2]=2+f[E]=2$
$f[C_1]=\min\{4+f[D_1],2+f[D_2]\}=\min\{4+1,2+2\}=4$
$f[C_2]=\min\{6+f[D_1],9+f[D_2]\}=\min\{6+1,9+2\}=7$
$f[C_3]=\min\{7+f[D_1],5+f[D_2]\}=\min\{7+1,5+2\}=7$
$f[B_1]=\min\{1+f[C_1],5+f[C_2]\}=\min\{1+4,5+7\}=5$
$f[B_2]=\min\{8+f[C_1],4+f[C_2],6+f[C_3]\}=\min\{8+4,4+7,6+7\}=11$
$f[B_3]=\min\{4+f[C_1],4+f[C_2],2+f[C_3]\}=\min\{4+4,4+7,2+7\}=8$
$f[A]=\min\{3+f[C_1],5+f[C_2],4+f[C_3]\}=\min\{3+5,5+11,4+8\}=8$

现在的问题是，对任意给定的路线图，如何定出这个求解子问题顺序的序列呢？拓扑排序是其中的一种考虑。但是我们不要忘记，即使这个图中含有长度为正数的回路，问题还是有解的，不过这个图就不能进行拓扑排序了。另外，如果图中含有起点可达的负权回路，算法应该宣告原问题无解，但是这个算法并不能很好地解决这个问题。

正反思维，多向求解

要从这个困局之中摆脱出来，不妨用逆向思维，从另一个角度来看这个问题。

之前的分析是从终点开始，从后向前逐步递推求出各点到 E 的最短路径，最后求得从 A 到 E 的最短路径。其实，最短路还有另一个性质：从起点到终点的最短路也是起点到该路径上各点的最短路径。从这个性质出发，我们可以从起点出发，递推求出其他点的最短路的长度。

现在我们改记 $f[P_k]$ 表示由 A 到点 P_k 的最短路的长度，例如，f[E]表示由 A 到 E 的最短路的长度，$f[B_1]$ 表示由 A 到 B_1 的最短路的长度。显然有 f[A]=0，而 $f[P_k]$ $(P_k\neq A)$是未知的，就记为 $f[P_k]=+\infty$。我们将此作为初始的已知条件。根据最短路这一特性，我们可以得到一个由 $f[P_k]$修正 $f[P_{k+1}](P_{k+1}\in x(P_k))$的公式：

$$f[P_{k+1}]=\min\{f[P_{k+1}],f[P_k]+d[P_k,P_{k+1}]\mid P_{k+1}\in x(P_k)\}$$

当所有的 $f[P_k]$都不能再改进的时候，f[E]中存放的就是由点 A 到点 E 的最短路的长度。这个算法可以写成一个反复迭代的过程，直到整个“系统”稳定下来的时候，迭代结束。我们可以看作这个迭代过程收敛于问题的最优解。

```
repeat
NoChange := true;
  对所有的点 Pk do
对所有的点 Pk+1 ∈ x(Pk) do
  if f[Pk+1] > f[Pk]+d[Pk,Pk+1] then
begin
f[Pk+1] := f[Pk]+d[Pk,Pk+1];
NoChange:=false;
            end;
until NoChange;
```

这个算法的好处在于它的形式很简单，但是它的效率较低，而且还有一个缺陷：当图中含有起点可达的负权回路时，这个算法会陷入死循环。其实，当图中含有起点可达的负权回路时，最短路问题是无解的。所以当图中含有负权边的时候，在使用这个算法之前，要进行额外的检查。

求最短路的 Dijkstra 算法

有没有能克服上面两个问题的算法呢？答案是肯定的。首先不妨假设图 G=(V,E)中不含负权回路，则 $f[v_0]=\min\{f[v_k]\mid v_k\in V\}$一定是从起点到 v_0 的最短路的长度，因为 $f[v_0]$不可能继续改进了。然后对所有在 $x(v_0)$中的点进行改进，则下一个肯定已经求出最短路的必定是 $f[v_1]=\min\{f[v_k]\mid v_k\in V-\{v_0\}\}$，如此下去，可以肯定的说，经过 n 次迭代，一定能求出所有点的最短路长度。

下面我们完整地用伪码描述这个算法：

```
f[A] := 0,f[v] := +∞ (v≠A)
S := { };
for i := 1 to n do
begin
  f[v] = min{f[vk] | vk ∈ V-S};
  对所有 u ∈ x(v) do f[u] := min{f[u],f[v]+d[v,u]};
  S := S ∪ {v};
end;
```

这就是求单源最短路径的 Dijkstra 算法。

现在我们来考虑图 G 中含有负权回路的情况。如果在算法执行的过程中存在某个 u∈ x(v)且 u∈S 且 f[v]+d[v,u] ∈ f[u],则图 G 中一定含有负权回路,这时算法应终止并返回原问题无解。

如果我们只需要求出从 A 到 E 的最短路,则上面的算法可改写为(我们把判断负权回路也考虑进去):

```
f[A] := 0,f[v] := +∞ (v≠A);
S := { };
repeat
   f[v] = min{f[vk] | vk ∈ V-S};
   对所有 u∈x(v) do
if f[v]+d[v,u] ∈ f[u] then
begin
   if u ∈ S then
return 问题无解;
f[u] := f[v]+d[v,u];
end;
   S := S ∪ {v};
until f[v]=+∞ or v=E;
return f[E];
```

Dijkstra 算法与广度优先搜索

如果我们再换一个角度,从图的搜索来重新审视这个算法,就会发现我们生成了一棵以结点 A 为根的搜索树,在扩展结点的时候,我们每次总是挑选深度最小的结点进行扩展,这种做法其实就是图的广度优先搜索。换而言之,图的广度优先搜索的思想本质与动态规划是一致的。下面我们从这个角度再来重写求解最短路的算法,以加深读者对动态规划的认识:

```
f[A] := 0,f[v] := +∞ (v≠A);
CLOSED 表初始为空,OPEN 表初始为结点 A;
repeat
   从 OPEN 表中选择 f[v]值最小的结点 v 放入 CLOSED 表;
   对所有 u∈x(v) do
if f[v]+d[v,u] ∈ f[u] then
begin
   if u 在 OPEN 表中 then
return 问题无解;
f[u] := f[v]+d[v,u];
 //扩展出结点 u
将结点 u 放入 OPEN 表;
end;
until OPEN 表空 or v=E;
return f[E];
```

从上面的分析可以看出,我们从已知的初始状态出发,利用最优化原理,一步一步向未知的目标状态推进,直到目标状态解决。从已知推广到未知,这就是动态规划思维方法的精髓。

算法的时间复杂度分析

我们来简单分析一下算法的时间复杂度。一般情况下,Dijkstra 算法的时间复杂度是 $O(n^2)$,但是当图 G 是一个稀疏图时,特别地,当所有顶点的度数都有上界 C(C 是常数),如果数据结构组织得当(例如,采用堆来存放 OPEN 表),可以将 Dijkstra 算法的时间复杂度降为 $O(n\log_2 n)$。当 n 特别大的时候,这一点就显得至关重要了。我们将在以后的专题中进行分析。

本期将刊登 3 题应用最短路问题求解的例题,供读者分析和学习。希望读者能够从题目的分析中领会这一点:应用动态规划解题是富于技巧和创造性的,没有固定的模式可套;题目出现的形式多种多样,而且大部分表面上与动态规划看不出直接的联系,只有在充分把握其思想精髓的前提下大胆联想,才能达到得心应手,灵活运用的境界。

第一题:相邻项序列

问题描述:

对于一个 N * N(N≤100)的正整数矩阵 M,存在从 M[A1,B1]开始到 M[A2,B2]结束的相邻项序列。两个项 M[I,J]和 M[K,L]相邻的条件是指满足如下情况之一:(1) I=K±1 和 J=L (2) I=K 和 J=L±1

任务:从文件 1 中输入矩阵 M,从文件 2 中输入 K(K<=4)组 M[A1,B1]和 M[A2,B2]的值,对于每一组 M[A1,B1]和 M[A2,B2],求一相邻项序列,使得相邻项之差的绝对值之和为最小。

输入格式:

从键盘输入数据文件 1 及数据文件 2 的文件名。

输入数据文件 1 格式如下:

```
4              ———    表示矩阵 M 的大小
1 9  6 12      ———    每行有 M 个数据,共 M 行
8 7  3  5
5 9 11 11
7 3  2  6
```

输入数据文件2格式如下：

```
2          ———表示有K组数据
4 1 1 4    ———表示A1,B1及A2,B2的值,共N行
2 2 3 4
```

输出格式：

输出到数据文件，文件名由键盘输入：

```
1 17          ——— 表示第1组数据相邻项之差的
                   绝对值之和的最小值为17
7 5 8 7 9 6 12 ——— 表示第1组数据的相邻序列
2 4           ——— 表示第2组数据相邻项之差的
                   绝对值之和的最小值为4
7 9 11 11     ——— 表示第2组数据的相邻序列
```

算法分析：

本题若将相邻的两个数看作是两个顶点，两个数差的绝对值作为权，则问题便转化成图论中的求两顶点间的最短路问题。

对于有向图D=(V,A)，弧a=(Vi,Vj)，相应地有权W(a)=Wij，对有向图D中两个顶点Vs、Vt，设P是D中从Vs到Vt的一条路，定义路P的权是P中所有弧的权之和，记作W(P)，所谓最短路问题就是在所有从Vs到Vt的路中，找出一条权最短的路，即求一条从Vs到Vt的路Po，令：

$$W(Po)=\min_{P} W(P)$$

对于所有Wij>=0，即所有的权为非负值时，求最短路通常使用标号法。

所谓标号法的基本思想是从Vs出发，逐步地向外探寻最短路。在执行过程中，与每个点对应，记录下一个数(称为这个点的标号)，它或者表示从Vs到该点的最短路的权(称为P标号)，或者表示这个权的上界(称为T标号)，具体做法是每扩展一步，就将一个具T标号的点改为具P标号的点，从而使有向图D中具P标号的顶点数增多一个。如此一步步执行下去，就可求出从Vs到各点的最短路。

为简便起见，我们可以称从M[I,J]到M[K,L]的相邻项之差的绝对值之和最小的相邻项序列为从M[I,J]到M[K,L]的“最优序列”。这样便可用标号法来求得从起点M [A1,B1] 到矩阵M的其它各项的“最优序列”。

设SUM[I,J]为从M[A1,B1]到M[I,J]的“最优序列”的相邻项的绝对值之和。有：

SUM[I,J] = min SUM[K,L]+abs(M[I,J]-M[K,L])
I=K±1 且 J=L,或 I=K 且 J=L±1

通过这一式子，可以利用标号法来求得从M[A1,B1]到M[A2,B2]的“最优序列”。

在标号法中，每一次扩展都要寻找一个项M[I,J],其中：

①M[I,J]是未改为P标号的点；

②SUM[I,J] = min SUM[K,L] K,L∈[1,N],且M[K,L]是未改为P标号的点。

这一步若采用二重循环来求则非常耗时。所以考虑采用一个队列来存储矩阵中待扩展的项，使得该队列的各项的SUM值是由小到大排列的。扩展时，只要移动队列的首指针即可；生成的新的待扩展的项，可以将其插入到队列中的适当位置，使插入后队列的各项的SUM值仍是从小到大排列。为了较方便地插入新的待扩展的项，采用指针结构来存储这个队列。

具体算法描述如下：

定义一个POINT类型来记录待扩展的项：

```
point=^xy;
xy=record
 x,y:byte;
 next:point;
end;
```

其中X,Y为该项在矩阵M中的坐标，NEXT为指针类型，以记录其在队列中的后继结点。

(1)置SUM[I,J]为∞(I<>A1或J<>B1)，置SUM[A1,B1]为0；

(2)首指针F后移一位；

(3)若F^.X=B2且F^.Y=A2，则已找到从M[A1,B1]到M[A2,B2]的“最优序列”，反向链接、打印所经的路径并转(6)，否则继续(4)；

(4)从M[F^.Y,F^.X]向上下左右四个方向扩展，现设向M[NODE1^.Y,NODE1^.X](NODE1^.Y=F^.Y±1且NODE1^.X=F^.X，或NODE1^.Y=F^.Y且NODE^.X=F^.X±1)扩展：

①SUM1:=SUM[F^.Y,F^.X]+abs(M[F^.Y,F^.X]-M[NODE1^.Y,NODE1^.X])；

②若SUM1<SUM[NODE1^.Y,NODE1^.X]，则继续③，否则转(4)；

③SUM[NODE1^.Y,NODE1^.X]:=SUM1；

④M[F^.Y,F^.X]是M[NODE1^.Y,NODE1^.X]的前趋项，记下从M[NODE1^.Y, NODE1^.X]到M[F^.Y,F^.X]的方向，以便打印时反向链接所经路径；

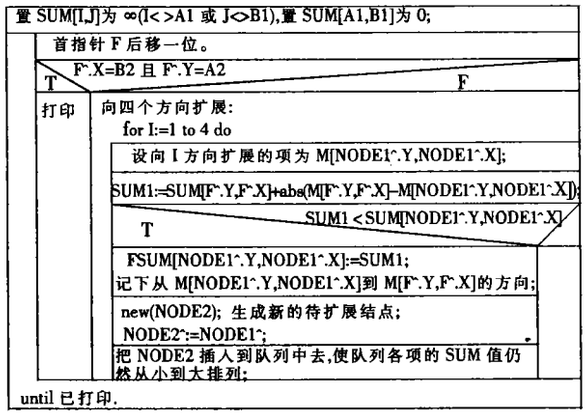
⑤生成一新待扩展结点NODE2，使得NODE2^=NODE1^，并把NODE2插入到队列中，使得

插入后队列的各项的SUM值仍是从小到大排列;

(5)转(2);

(6)结束。

用N-S框图描述如下:



对时间、空间复杂度的分析:

本算法的时间复杂度约为O(N^4)。由于对矩阵中的某一项,当其SUM值每改变一次,就会在队列中生成一个新的结点,而原来的代表该项的结点就荒废不用。所以,本算法比较浪费空间,但如果改为对原结点进行修改再重新放入队列,尽管节省了空间,但耗时较多,显得不合算了。本算法是用空间来换取时间。

第二题:猫捉老鼠

问题描述:

有如下图形一个笼子,笼子中方格边长为1,共有N*M(N≤100,M≤100)个方格,其中打"*"的方格放有障碍物,不可进入。现在笼中有一只猫和一只老鼠,猫的初始位置已知,老鼠在R(R≤200)秒内的运动轨迹已知。老鼠每秒走一步,猫每秒走S步。所谓一步是指在笼子中从方格A(I,J)运动到方格B(K,L)且指满足如下情况之一:(1) I=K±1 和 J=L (2) I=K 和 J=L±1 (左上角方格坐标为(1,1),右下角方格坐标为(N,M))。

		*	
*		*	

任务:判断猫在R秒内能否捉住老鼠。如果能捉住老鼠,则先输出YES,再输出最短时间及运动轨迹。如果不能捉住老鼠,则输出NO。所谓猫在T秒内捉住老鼠是指猫鼠在T秒末的相距步数不超过S步。

输入格式:

从键盘输入数据文件的文件名

输入数据文件格式如下:

```
3 4          ———表示笼子大小 N=3,M=4
0 0 1 0      ———表示每行有 M 个数据,共 N 行
1 0 1 0      ———0 表示无障碍,1 表示有障碍
0 0 0 0
2            ——— 表示猫每秒走 S 步
1 1          ——— 表示猫的初始位置
3            ——— 表示老鼠走了 R 秒
3 4          ——— 表示老鼠初始位置
3 3          ——— 表示老鼠第 1 秒末的位置
3 2          ———表示老鼠第 2 秒末的位置
3 1          ———表示老鼠第 R 秒末的位置
```

输出格式:

输出到数据文件,文件名由键盘输入。

```
YES          ———表示能捉住老鼠(否则输出 NO)
1            ———表示能在最短时间 1 秒内捉住老鼠
1 1          ———表示猫初始位置
2 2          ———表示猫第 1 秒末的位置
```

算法分析:

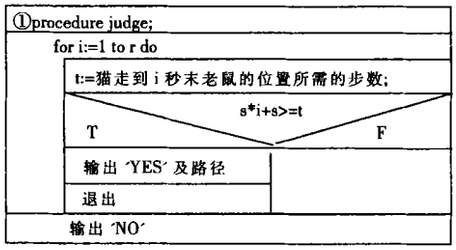
本题思路十分简单,将问题转化成求最短路径问题。也就是说,对每一个位置,分别求出猫和老鼠到该位置的最短路径,并分别计算出猫和老鼠最短沿路径到达该位置所需的最短时间,通过时间的比较就可以判断出猫是否能捉住老鼠。

在实现上,首先用标号法求出猫到达任意位置所需的步数(标号法请参见上一题相邻项序列的分析),然后判断猫能否在规定时间内走完捉鼠所需的步数。

时间、空间复杂度均为O(nm),最坏情况下运算100*100=1万次。

用N-S框图描述如下:

主程序
初始化
计算猫能到达的位置及其步数
判断及输出结果①



第三题:旅游预算

问题描述:

一个旅行社需要估算乘汽车从某城市到另一城市的最小费用,沿路有若干加油站,每个加油站收费不一定相同。

旅游预算有如下规则:

(1)若油箱的油过半,不停车加油,除非油箱中的油不可支持到下一站;

(2)每次加油时都加满;

(3)在一个加油站加油时,司机要花费 2 元买东西吃;

(4)司机不必为其他意情况而准备额外的油;

(5)汽车开出时在起点加满油箱;

(6)计算精确到分(1 元=100 分)。

编写程序估计实际行驶在某路线所需的最小费用。

输入格式:

从当前目录下的文本文件"route.dat"读入数据。

按以下格式输入若干旅行路线的情况:

第一行为起点到终点的距离(实数);

第二行为三个的实数,后跟一个整数,每两个数据间用一个空格分隔。

其中第一个数为汽车油箱的容量(升),第二个数是每升汽油行驶的公里数,第三个数是在起点加满油箱所需的费用,第四个数是加油站的数量(≤50)、接下去的每行包括两个实数,每个数据之间用一个空格分隔,其中第一个数是该加油站离起点的距离,第二个数是该加油站每升汽油的价格(元/升)。加油站按它们与起点的距离升序排列。所有的输入都一定有解。

输出格式:

答案输出到当前目录下的文本文件"route.out"中。

该文件包含两行。第一行为一个实数和一个整数,实数为旅行的最小费用,以元为单位,精确到分,整数表示途中加油的站的总数 N。第二行是 N 个整数,表示 N 个加油的站的编号,按升序排列。数据间用一个空格分隔,此外没有多余的空格。

输入输出举例:

输入文件:route.dat

输出文件:route.out

```
516.3
15.7 22.1 20.87 3        38.09 1
125.4 1.259              2
297.9 1.129
345.2 0.999
```

算法分析:

本题的求解与求有向图的最短路径比较相似,也是选取两点间的直接连线费用和经过某个中间结点进行转折费用中的最小者。因此,求总费用的最小值可以分解为求若干分段的费用最小值,它满足优化原则,所以本题可以采用动态规划的方法进行求解。

```
for interval:=1 to n-1 do
 for start:=0 to n-1 do
  if start+interval<=n then
  begin
   stop:=interval+start;
   if 从 start 能直接到 stop then
    begin
     lest[start,stop]:=从 start 能直接到 stop 的费用;
     nextstop[start,stop]:=stop;
    end;{求出直接从 start 到 stop 的费用}
   for next:=start+1 to stop-1 do
   if 从 start 能直接到 next then
    begin
     cost:=从 start 到 next 的费用+从 next 到 stop 的费用;
     if cost<lest[start,stop] then
      begin
       lest[start,stop]:=cost;
       nextstop[start,stop]:=next;
      end;
     end;{中间经过 next 是否能减少费用}
  end;{动态规划的求解过程}
```

然后,从各油站中找到能直接到终点,而且从起点到该站的 lest 值最小的油站 Stop。则该旅程的最小费用为该 lest 值加上在起点的费用。然后利用下面的迭代式就可以求出,中途所停的油站:

start:=0;start:=nextstop[start,stop];(直到 start=stop)

(收稿日期:2000-04-10)