算法设计与分析第二次作业--动态规划 ZY1606104 崔毅峰

一、用动态规划方法手工求解下面的问题：

某工厂调查了解市场情况，估计在今后四个月内，市场对其产品的需求量如下表所示。

时期（月） 需要量（产品单位）

1 2

2 3

3 2

4 4

已知：对每个月来讲，生产一批产品的固定成本费为3 (千元)，若不生产，则为零。每

生产单位产品的成本费为1 （千元)。同时，在任何一个月内，生产能力所允许的最大生产

批量为不超过6 个单位。

又知每单位产品的库存费用为每月0.5 （千元），同时要求在第一个月开始之初， 及

在第四个月末，均无产品库存。

问：在满足上述条件下，该厂应如何安排各个时期的生产与库存，使所花的总成本费用

最低？

要求：写出各种变量、状态转移方程、递推关系式、和详细计算步骤。

1.由题目可知，这是一道典型的生产存储类的动态规划问题。题目要求4个月内，如何安排每个月的生产和存储使得最后的总花费最小，很显然是一个多阶段决策问题。

由于本题中，需求量是给定的，存储量是由生产量和需求量决定的，所以自然状态变量就是生产量。所以按照动态规划求解方法，设状态变量表示第i个月的生产量，表示第i个月的需求量，表示第i个月月初的存储量。状态转移方程如下：

=+- , >=0,0<=<=6,i=1,2,3,4

每个月的决策集合()={ |0<=<=6} i=1,2,3,4

阶段指标为每个月生产成本与存储成本之和：

(,)=

指标函数为：

=

最优指标函数，表示从第i个月到第4个月的总的最小开销：

()= i=4,3,2,1，且()=0

通过逆向计算，最终求解的就是(0)的值。

递推如下：

()=

由前面状态转移关系式得到:

=-+=4-

所以可以从4取到0，即从0变到4

(0)=7

(1)=6.5

(2)=6

(3)=5.5

(4)=2

同理得：

()=

由前面关系式得到:

=-+=2+-

=0时，即=2+，所以可以从2取到6，所以

(0)=min{12,12.5,13,13.5,11}=11 =6

以此类推，

(1)=min{11.5,12,12.5,13,10.5}=10.5 =5

(2)=min{8,11.5,12,12.5,10}=8 =0

(3)= min{8,11.5,12,9.5}=8 =0

(4)= min{8,11.5,9}=8 =0

(5)= min{8,8.5}=8 =0

(6)= min{5}=5 =0

()=

由前面关系式得到:

=-+=3+-

(0)=min{17,17.5,16,17}=11 =5

(1)=min{16.5,17,15.5,16.5,17.5}=10.5 =4

(2)=min{16,16.5,15,16,17,18}=15 =3

(3)= min{12.5,16,14.5,15.5,16.5,17.5,15.5}=12.5 =0

(4)= min{12.5,14,15,16,17,15}=12.5 =0

(5)= min{10.5,14.5,15.5,16.5,14.5}=10.5 =0

(6)= min{11,15,16,14}=11 =0

()=

由前面关系式得到:

=-+=2+-

(0)=min{21,21.5,22,20.5,21.5}=20.5 =5

经过计算可得，倒推得第一个月生产5单位产品，初始存储量为0，第二个月生产0，初始库存量为3，第三个月生产6，初始库存量为0，第四个月生产为0，初始库存量为4，所花开销最小为20.5。

2.

设F[i,s]表示当前节点为i，未访问的节点集合为s的最短路程。决策为选出下一个节点k，所以状态转移方程为：

F[i,s]=，d[i][k]表示i到k的距离

伪代码如下：

TSP(i,S)

//输入:输入起始点i，和未访问过的一个城市集合S,假设输入i为1

//输出:从i出发，经过S中每一个点后，又返回i的最短距离和路径

1. if S=空集合，return （d[i][1],1)

2.否则，对所有S中的元素j，执行TSP(j,S-j)找出使到起点1成立的最小j,记为n0

3.然后从n0出发继续执行TSP(n0,S-n0)，直到到达条件1，路径就是n和TSP(n,S-n)中的path的合，距离也是一样。

4.return (d,path)

时间复杂度：

由于集合的状态个数为2^n-1 个，还需要枚举每个结点，以及每个集合中的元素，因此整个算法的近似复杂度为O(n^2\*2^n)。

程序具体实现代码(参考了网上的部分代码)：

#include<iostream>

using namespace std;

const int n = 6;

const int size = 1 << ( n - 1 );//利用整数位运算表示集合S

const int infinite = 10000;

int D[n][n] = {

{0, 10, 20, 30, 40, 50},//0行

{12, 0, 18, 30, 25, 21},

{23, 19, 0, 5, 10, 15},

{34, 32, 4, 0, 8, 16},

{45, 27, 11, 10, 0, 18},

{56, 22, 16, 20, 12, 0}//5行

};

int printPath( int path[size][n - 1], int final, int s, int step )

{

//递归打印路径

if( step == 1)

cout << "step 1: " << final << endl;

else

{

printPath( path, path[s][final - 2], s - ( 1 << ( 6 - final ) ), step - 1 );

cout << "step " << step << ": " << final << endl;

}

return 0;

}

void tsp()

{

//动态规划求解有向图的TSP问题，其中minDis[][]为上述递推关系式中的dp[][],path[][]用于存放路径

int minDis[size][n - 1];

int path[size][n - 1];

for( int set = 1; set != size; ++ set)

{

for( int city = 2; city != n + 1; ++ city )

minDis[set][city - 2] = infinite;

}

for( int city = 2; city != n + 1; ++ city )

minDis[1 << ( n - city )][city - 2] = D[0][city - 1];

for( int set = 1; set != size-1; ++ set )

{

for( int next = 2; next != n + 1; ++ next )//2~6 选择一个城市

{

if( ! ( ( 1 << ( n - next ) ) & set ) )//若城市next不在集合set中

{

int min = infinite;

int set1 = set + ( 1 << ( n - next ) );//加入集合

for( int city = 2; city != n+1; ++ city )

{

//求从城市1出发经过set中所有城市各一次且仅再到达城市next的最短路径

if( ( 1 << ( n - city ) ) & set )//

{

int tmp = minDis[set][city - 2] + D[city - 1][next - 1];//

if( tmp < min )

{

min = tmp;

path[set1][next - 2] = city;//

}

}

}

minDis[set1][next - 2] = min;//

}

}

}

//求最终再回到城市1的最短行程，即31,011111的集合情况

int minDis\_all = infinite;

int final;

for( int k = 2; k != n + 1; ++ k )

{

if( minDis[size - 1][k - 2] + D[k - 1][0] < minDis\_all )

{

final = k;

minDis\_all = minDis[size - 1][k - 2] + D[k - 1][0];

}

}

//打印最短路径

cout << "The minimun distance is: " << minDis\_all << endl;

cout << "The path is:" << endl;

printPath( path, final, size - 1, n - 1 );

cout << "step 6: back to 1" << endl;

}

int main()

{

tsp();

return 0;

}