2015级计算机科学专业**“算法设计与分析”**综合大作业

2017~2018学年第1学期

2017年10月25日

**总体要求**

本次综合大作业（上机实现题目）是为了配合“**算法设计与分析**”课程的讲授而设置的，目的在于培养学生理论联系实际的问题求解能力。综合大作业题目共两道，依据不同题目的要求采用“动态规划”或“回溯法”算法求解。每题50分。以小组为单位检查，自由组合（可跨越班级），每组人数一般不少于4人、但不超过5人。每组提交1份**课程报告**，检查时每组至少须有1人做成果展示演讲（答辩），在课堂上就题目的设计思想、实现方法的正确性等进行说明，全组成员一起参与答辩，回答教授及同学的提问与质疑。

对于“**动态规划**”问题，课程报告中必须对**最优值函数**和**标记函数**的含义进行详细说明，给出子问题计算中所使用的**最优值函数**和**标记函数**的**递推关系**（**一般采用递归式加以描述**）和**初值**（边界条件）。对于“**回溯法**”问题，需要给出**限界函数**的设计。每一道题目均需给出所设计算法的**时间复杂度**分析。

**每组需自备2、3个相关问题的实例，报告中以实例中的数据描述算法的执行步骤**。

**B、TSP问题**

所谓TSP问题是指旅行商要去n个城市推销商品，其中每个城市到达且仅到达一次，并且要求所走的路程最短（该问题又称货郎担问题、邮递员问题、售货员问题等）。TSP问题最容易想到、也肯定能得到最优解的算法是穷举法，即考察所有可能的行走线路，从中选出最佳的一条。但是用穷举法求解TSP问题的时间复杂性为O(n!)，属于NP问题。请用数学语言对该TSP问题加以抽象，在此基础上给出**动态规划**求解该问题的递推公式。要求对所给公式中的符号意义加以详细说明，并简述算法求解步骤。用一种你熟悉的程序设计语言加以实现。

**TSP问题**

一、问题描述

所谓TSP问题是指旅行商要去n个城市推销商品，其中每个城市到达且仅到达一次，并且要求所走的路程最短（该问题又称货郎担问题、邮递员问题、售货员问题等）。

TSP问题是最基本的路线问题，是在寻求旅行者由起点出发，通过所有给定的需求点之后，最后回到原点的最小路径成本。

二、动态规划可行性分析

设v1,v2,v3......vn，v1是从v1出发后不重复的经过所有节点再次回到v1的路径中，路径长度最短的回路。将问题分解后变成，v1到vx的路径已经求出，那么问题将变为求vx经过v2，v3...vx-1，vx+1...vn再回到v1的最短路径(问题最优解）。再往下看，问题又将分解成，vx到vy的路径已经求出，那么问题将变为求vy经过v2，v3...vy-1,vy+1...vn再回到v1的最短路径（问题最优解）。以此类推，TSP问题可转化成求多个分问题最短路径的问题，具有最优子结构性质。因此可以用动态规划来求解。

三、问题分析

1、本问题可抽象为：在有向带权的图中寻找从某一点出发，经过所有结点，最终回到出发结点的权值最小且最短路径。

2、规范变量：

权值矩阵 ：Wn \* n；

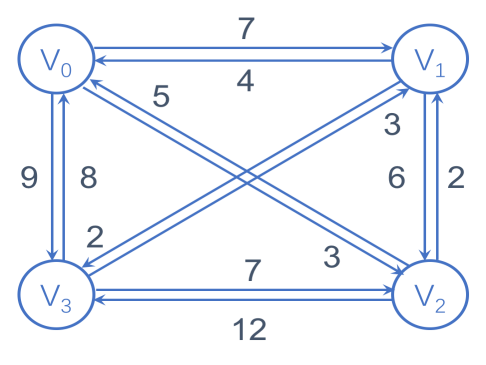
动态规划路径矩阵：Dn \* 2n-1；

最优路径：path；

最优路径总时间：value；

3、实例分析与递归关系推导

（1）问题的图形化表示：



图表1TSP问题4个节点的图形化实例

（2）实例的权值矩阵：

为了在程序中可以调用，我们将图形化实例用邻接矩阵表示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V0 | V1 | V2 | V3 |
| V0 | 0 | 7 | 3 | 9 |
| V1 | 4 | 0 | 6 | 2 |
| V2 | 5 | 2 | 0 | 12 |
| V3 | 8 | 3 | 7 | 0 |

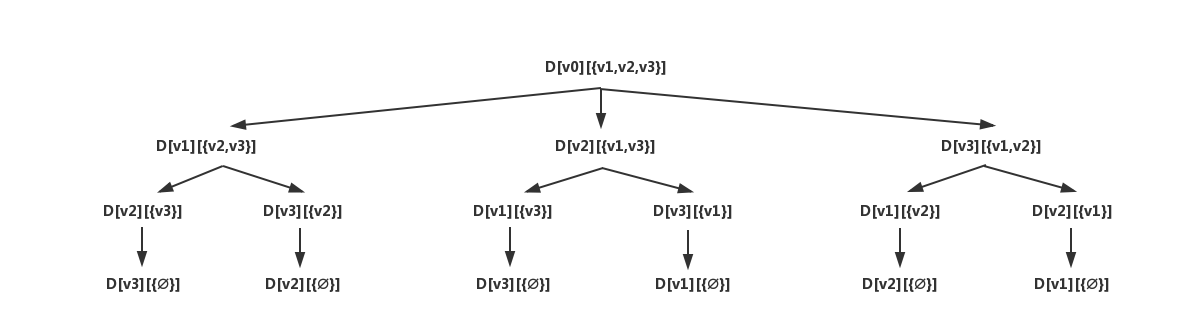
图表2 TSP问题图形的权值矩阵

（3）问题分解：

根据题目要求，将原问题逐步分解成多层次的多个子问题，以此降低求解难度，时间复杂度，以及空间开销。

自顶向下分析问题；

自底向上求解问题。



图表3 TSP问题分解示意图

（4）动态规划路径矩阵 Dn x 2n-1：

根据问题分解中的思路，建立Dn x 2n-1矩阵

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Φ | {v1} | {v2} | {v3} | {v1,v2} | {v1,v3} | {v2,v3} | {v1,v2,v3} |
| V0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 15 |
| V1 | 4 | 0 | 11 | 10 | 0 | 14 | 0 | 0 |
| V2 | 5 | 6 | 0 | 20 | 0 | 0 | 12 | 0 |
| V3 | 8 | 7 | 12 | 0 | 13 | 0 | 0 | 0 |

图表4 TSP问题动态规划矩阵

为了在程序中可以调用，我们将集合用二进制表示:

Φ:0000 ->0

{v1}:0001 ->1

{v2}:0010 ->2

{v3}:0100 ->4

{v1,v2}:0011 ->3

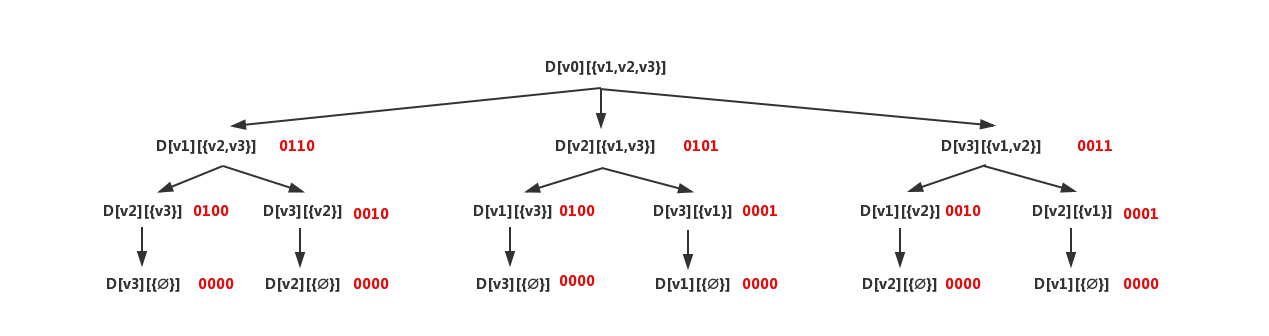
{v1,v3}:0101 ->5

{v2,v3}:0110 ->6

{v1,v2,v3}:0111 ->7

将集合转化为二进制表示后，我们可以进一步将二进制转化为对应的十进制，以便于程序运行中循环与递归调用。

但是在转化为十进制后，我们会发现集合的顺序被打乱（如方框中圈出的部分），原本保持的自底向上的求解顺序被打破。



图表5 TSP问题分解的二进制表示

仔细分析被打乱的部分后发现，按照十进制顺序求解问题时，被提前的集合只会用到已经求出的子问题的结果。求解顺序变为每个支路中的自底向上，和之路与之路之间的并列求解。因此可以按照十进制顺序的求解顺序。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0000 | 0001 | 0010 | 0100 | 0011 | 0101 | 0110 | 0111 |
| V0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 15 |
| V1 | 4 | 0 | 11 | 10 | 0 | 14 | 0 | 0 |
| V2 | 5 | 6 | 0 | 20 | 0 | 0 | 12 | 0 |
| V3 | 8 | 7 | 12 | 0 | 13 | 0 | 0 | 0 |

图表6 TSP问题的二进制表示矩阵

（5）递归公式推导：

四、代码实现

1、代码实现（未改进版）：

在第一版代码实现中，我们将递归矩阵D的二进制表达设置成三位数码表示。

#include<iostream>

#include<stdlib.h>

#include<bitset>//使用二进制bitset标准库

usingnamespacestd;

intresultCompare = 0;//最优子结构求解的中间比较变量

intDMatrix(intVrow, intVcolumn, intmode, intW[4][4],intD[4][8]) {

intvisitCity[3];//三位二进制的城市表示

intcnt=0;

int result=0;

bitset<3> t;//十进制转换成二进制

t =Vcolumn;

stringstr = t.to\_string();//二进制表示

if (Vcolumn == 0) { //如果是递归矩阵的第零列（Φ），直接从权值矩阵读取结果

resultCompare = W[Vrow][0];

}

else { //计算D矩阵值

for (intcityj = 0; cityj< (mode - 1); cityj++) { //最优值计算

intcitybit = 0;

if (str[cityj] == '1') { //分别计算每种情况（每个支路）的值，并取最优值

cnt = cnt + 1;

str[cityj] = '0';//D[][visitcity新]

for (intbitcnt = 0; bitcnt<mode - 1; bitcnt++) {

if (str[bitcnt] == '1') {

citybit = citybit + pow(2, mode - 2 - bitcnt);

//计算result存在D矩阵第几列

}

}

if (Vrow == (mode - 1 - cityj)) {

//行表示的城市若与列中的城市相同，则递归矩阵值，直接设为0

（原因：在未遍历完所有城市之前，不能形成回路）

resultCompare = 0;

break;

}

else {

result = W[Vrow][mode - 1 - cityj] + D[mode - 1 - cityj][citybit];

//利用最优子结构性质计算递归矩阵

if (cnt == 1)

resultCompare = result;

else {

if (result <resultCompare)

resultCompare = result;//最优值比较

}

}

str[cityj] = '1';

}

}

}

returnresultCompare;//DMatrix矩阵返回值为最优值

}

int main() {

intmode,modeSize;

mode = 4;//城市总个数

modeSize = pow(2 ,mode-1);//递归矩阵列数计算

int W[4][4] = { { 0,3,6,7 },{ 5,0,2,3 },{ 6,4,0,2 },{ 3,7,5,0 } };//输入权值矩阵

int D[4][8];

//D递归矩阵读入第一列

for (intzerocolumn\_row = 0; zerocolumn\_row< mode; zerocolumn\_row++) {

//初始化第一行,减少循环次数

D[zerocolumn\_row][0] = W[zerocolumn\_row][0];

}

//D递归矩阵计算

for (intcolumni = 1; columni< (modeSize - 1); columni++) {

for (introwi = 1; rowi< mode; rowi++) {

D[rowi][columni] = DMatrix(rowi, columni, mode, W, D);

}

}

//D递归矩阵计算最后一列

D[0][modeSize - 1] = DMatrix(0, modeSize - 1, mode, W, D);

cout<<"最短路径为："<<D[0][modeSize - 1]<<"\n";

}

2、代码改进

改进算法的思想为，在前一个算法的基础上，逐列生成矩阵D，在循环操作列索引的同时，定位出行索引的值，因此在一个for循环下即可同时得到该列的行索引以及其单元格中的数值。如此操作，比前一个算法省去一重循环，时间复杂度上有所减小。

下面详细说明算法改进的细节：

1. 逐列生成矩阵D

for(int row = 0; row<8;row++){

calculateD(row,mode,W,D);

}

1. 循环操作列索引

for (intcityID = 0; cityID< mode; cityID++)

1. 确定行索引，并求值

if(row == 0){

D[row][cityID+1] = W[cityID+1][0];

//cout<<D[row][cityID+1]<<" ";

}

else if(str[cityID] == '1'){

D[row][cityID] = 0;

}

else {

int result = 100;

for(intpopID = 1;popID<mode;popID++){

intpopValue = 0;

intafterPopDecimal = 0;

intresultTemp = 0;

if(str[popID]=='1'){

str[popID] = '0';

popValue = pow(2, mode-popID-1);

afterPopDecimal = row - popValue;

resultTemp = W[cityID][popID] + D[afterPopDecimal][popID];

str[popID] = '1';

if(resultTemp<result){

result = resultTemp;

}

}

}

D[row][cityID] = result;

}

五、计算复杂度分析

* 第一个算法，逐个单元生成矩阵D。计算每个单元格的值时，再运用两个for循环操作列索引。求得计算复杂度为*O*().
* 改进后的算法，逐列生成矩阵D。在循环操作列索引的同时得到行索引，以及其单元格所对应的数值，此处省去一个for循环。求得计算复杂度为*O*().

下面以一个15个节点的列子对两个算法的运行时间进行比较：

W[15][15] = {

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}};

令权值矩阵为15×15的矩阵，分别用两个算法求其最短路径长度：

第一个算法的运行时间为：938ms

D:\张洪滨\北工大\算法\1.png

改进后的算法运行时间为：110 ms

D:\张洪滨\北工大\算法\2.png