**2020-2021第二学期**

**网络工程专业2018级**

**《算法数据与分析》课程设计**

**班级：网络工程**

**学号：181124039**

**姓名：张怀周**

**旅行售货员问题**

1. **问题描述**

旅行售货员问题：某售货员要到若干城市去推销商品，已知各城市之间的路程（或旅费）。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程最短（或旅费最少）。

要求：

1. 运用所学设计至少一种算法，并给出运行结果；
2. 对所设计算法的时空效率进行分析；
3. 写出你的心得与体会；

**2.算法分析**

当我们进行递归搜索，搜索到第t层时，我们需要判断一下x[t]所代表的城市是否与上一层x[t-1]所代表的城市有“路”，如果没有的话，需要改变x[t]的值，然后继续上述判断，当出现一个满足条件的x[t]后还要判断当前从1到t-1所走的路程cc加上x[t]与x[t-1]的距离是否小于当前已经记录的最优解（最优解的初始值是一个足够大的数），如果到t的距离比当前最优解还要大的话，那么再以这条路线搜索下去的话回到城市1的路程一定比当前最优解还大，所以我们没有必要对这条路线进行下一步的搜索。最后我们来确定当搜索到叶子结点的时候我们该如何处理？已知搜索到t层时，若t = n，说明已经搜索到了叶子结点，这个时候我们还需做上述所说的两个判断，如果两个判断都通过的话，说明该解比当前最优解还优，那么我们需要将该解记录下来，并记录该解的最优值。

**3.算法描述**

void travel(int t) {

if(t到达第n层即搜索到叶子结点) {

if(城市x[t-1]可以到达城市x[t]，并且城市x[t]可以回到城市1，且此时所走的路程cc加上

x[t-1]与x[t]的距离和x[t]与1的距离小于当前最优值bestc) {

将最优解记录下来;

将最优值记录下来;

}

return;

}

for(int i = t; i < n; i++) {

if(城市x[t-1]能达到城市x[i]即这两个城市间有边，并当前所走的路程cc加上这两个城市的距离

没有比当前最优值bestc大) {

swap(x[i], x[t]);

修改此时所走的路程cc;

进入下一层递归;

恢复原来cc的值;

swap(x[i], x[t]);

}

}

}

**4.算法实现**

**#include <bits/stdc++.h>**

**using namespace std;**

**const int max\_ = 0x3f3f3f;**

**const int NoEdge = -1;**

**int citynum; //城市数**

**int edgenum;//边数**

**int currentcost;//当前路程**

**int bestcost;//最优路程**

**int Graph[100][100];//边距记录**

**int x[100];//记录行走顺序**

**int bestx[100];//记录最优行走顺序**

**//初始化**

**void Initilize()**

**{**

**currentcost = 0;**

**bestcost = max\_;**

**for(int i = 1; i <= citynum; ++i)**

**{**

**x[i] = i;**

**}**

**}**

**void Swap(int &a, int &b)**

**{**

**int temp;**

**temp = a;**

**a = b;**

**b = temp;**

**}**

**void BackTrack(int i) //i代表第i步去的城市而不是代号为i的城市**

**{**

**if(i == citynum)//t到达第n层即搜索到叶子结点**

**{**

**if(Graph[x[i - 1]][x[i]] != NoEdge && Graph[x[i]][x[1]] != NoEdge && (currentcost + Graph[x[i - 1]][x[i]] + Graph[x[i]][x[1]] < bestcost || bestcost == max\_))**

**{**

**//到最后一个城市与出发城市之间有路径且当前总距离比最优值小**

**bestcost = currentcost + Graph[x[i - 1]][x[i]] + Graph[x[i]][x[1]];//记录最优解**

**for(int j = 1; j <= citynum; ++j)//记录最优值**

**bestx[j] = x[j];**

**}**

**}**

**else //没有到达叶子节点**

**{**

**for(int j = i; j <= citynum; ++j)**

**{**

**//城市x[i-1]能达到城市x[j]即这两个城市间有边，并当前所走的路程cc加上这两个城市的距离没有比当前最优值bestc大)**

**if(Graph[x[i - 1]][x[j]] != NoEdge && (currentcost + Graph[x[i - 1]][x[j]] < bestcost || bestcost == max\_))**

**{**

**Swap(x[i], x[j]); //保存要去的第i个城市到x[i]中**

**currentcost += Graph[x[i - 1]][x[i]];//修改此时所走的路程cc**

**BackTrack(i + 1);//搜索下一城市**

**currentcost -= Graph[x[i - 1]][x[i]];//恢复原来cc的值**

**Swap(x[i], x[j]);**

**}**

**}**

**}**

**}**

**int main()**

**{**

**int p1, p2, len;**

**cout<<"输入城市数和边数："<<endl;**

**cin>>citynum>>edgenum;**

**memset(Graph, NoEdge, sizeof(Graph));**

**cout<<"输入两座城市之间的距离（p1 p2 l):"<<endl;**

**for(int i = 1; i <= edgenum; ++i)**

**{**

**cin>>p1>>p2>>len;**

**Graph[p1][p2] = Graph[p2][p1] = len;**

**}**

**Initilize();//初始化**

**BackTrack(2); //回溯函数**

**cout<<"最短路线为："<<bestcost<<endl;**

**cout << "路线为:" << endl;**

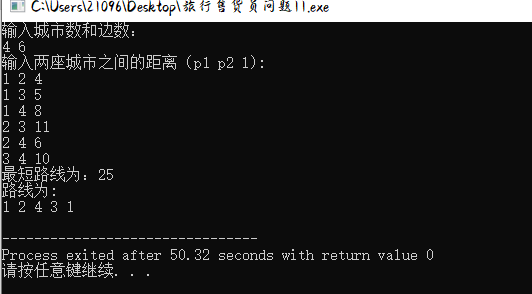
**for(int i = 1; i <= citynum; ++i)**

**cout << bestx[i] << " ";**

**cout << "1" << endl;**

**}**

1. **测试样例**



1. **时空分析**

算法Backtrack在最坏情况下算法的时间复杂度为O(n!)。

7.**个人心得体会**

该算法使用回溯算法问题进行求解，回溯法思路的简单描述是：把问题的解空间转化成了树的结构表示，然后使用深度优先搜索策略进行遍历，遍历所有树的结点，从而找到最优解。