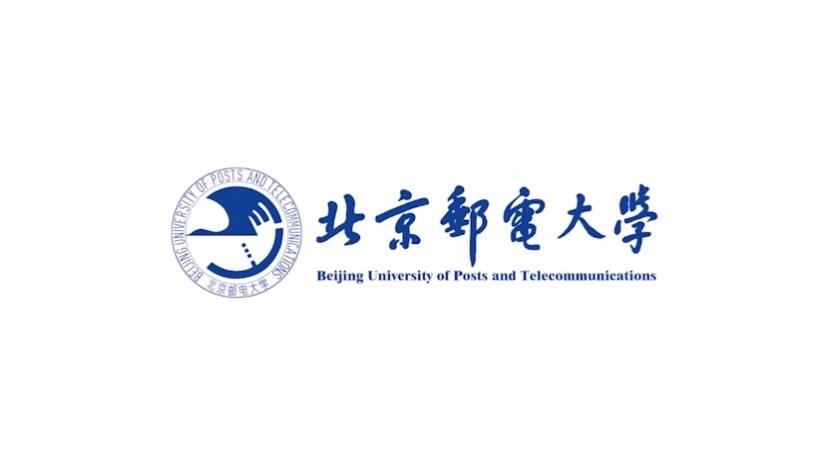
**算法设计与分析实验报告**



实验题目 旅行售货员 TSP 问题的回溯法求解探索

姓名： 陈俊卉

学号： 2020212256

日期： 2022.12.4

# 一、实验环境

(列出实验的操作环境，如操作系统，编程语言版本等，更多信息可视各自实际情况填写)

1. 操作系统：windows 10
2. 编程语言：c++
3. 编程工具：vscode及其组件

# 二、实验内容

具体要求请参照实验指导书中各实验的“实验内容及要求”部分。

(示例：1.描述你对实验算法的设计思路;2.给出算法关键部分的说明以及代码实现截图；3.对测试数据、测试程序(没有要求则非必须)进行说明，如测试覆盖程度，最好最坏平均三种情况等等，并给出测试结果截图等信息)

#### 0、实验要求

如图 1 所示，节点代表城市，节点之间的边代表城市之间的路径。每个城市都有一条进入路径和离开路径，不同的路径将耗费不同的旅费。旅行售货员选择城市1 作为出发城市，途经其他每个城市，要求每个城市必须经过一次，并且只能经过一次，求一条具有最小耗费的路径，该路径从城市 1 出发，经过其余 5 个城市后，最后返回城市 1。

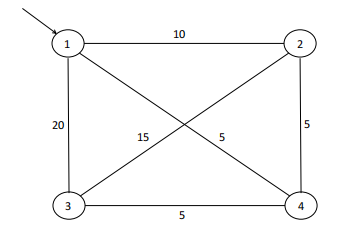


图1

本次实验使用cpp、利用递归回溯法求解。

#### 1、2、设计采用的界限函数，给出回溯过程对结点采用的剪枝策略

###### 剪枝条件1：

当i<n时，若x[i]添加到当前路径后，路径长度已经大于目前的最短路径长度，则剪枝，不再继续搜索。具体公式如下所示：

###### 剪枝条件2：

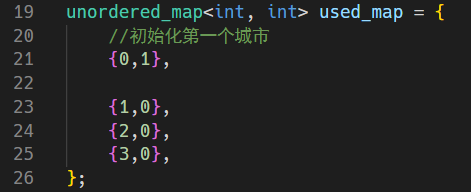
而当遇到i与其子节点之间没有边，因为我们会**在权重矩阵中用\_\_INT\_MAX\_\_（边长不会超过这个数字）表示无边**，所以相加之后一定会大于目前最短路径长度，即会归类到上述界限函数中。但为了进一步加快速度，我们在这一情况也直接剪枝，减少相加这一步骤。

#### 3、编写基于回溯法的算法代码，求出一条最短回路及其长度

采用递归的回溯法解决问题，代码如下所示：

 #include<iostream>  
 #include <unordered\_map>  
 using namespace std;  
 ​  
 ​  
 ​  
 int V[100]; //顶点集合  
 const int n = 4; //顶点个数  
 int w[n][n] = {  
     \_\_INT\_MAX\_\_,10,20,5,  
     10,\_\_INT\_MAX\_\_,15,5,  
     20,15,\_\_INT\_MAX\_\_,5,  
     5,5,5,\_\_INT\_MAX\_\_  
 };  //权重矩阵  
 int x[n]; //路径序列  
 int x\_best[n]; //最短路径序列  
 int cv = 0; //当前最短路径长度  
 int v\_best= \_\_INT\_MAX\_\_; //最短路径长度  
 unordered\_map<int, int> used\_map = {  
     //初始化第一个城市  
    {0,1},  
       
    {1,0},  
    {2,0},  
    {3,0},  
 };   
 ​  
 // 递归回溯  
 void TspDFS\_1(int i){  
     if (i == n) {  
         cv = cv + w[x[n-1]][0];  
         if(cv < v\_best) {  
             v\_best = cv;  
             for (int j = 0; j < n;j++){  
                 x\_best[j] = x[j];  
            }  
        }  
         cv = cv - w[x[n-1]][0];  
    }  
 ​  
     for (int u = 0; u < n; u++){  
         // 该城市还没经过  
         if (used\_map[u] == 0) {  
             if (cv + w[x[i-1]][u] < v\_best) {  
                 x[i] = u;  
 ​  
                 cout << "x:" ;  
                 for (int j = 0; j <= i;j++){  
                     cout << x[j] + 1 << ' ';  
                }  
                 cout << endl;  
 ​  
                 cv = cv + w[x[i-1]][u];  
                   
                 used\_map[u] = 1;  
 ​  
                 // cout << "map:" ;  
                 // for (int j = 0; j <= 3;j++){  
                 //     cout << used\_map[j] << ' ';  
                 // }  
                 // cout << endl;  
 ​  
                 // cout << "cv:" ;  
                 // cout << cv << endl;  
 ​  
                 TspDFS\_1(i+1);  
 ​  
                 // 回溯  
                 x[i] = 0;  
                 cv = cv - w[x[i-1]][u];  
                 used\_map[u] = 0;  
 ​  
            }  
        }  
    }  
 ​  
 }  
 ​  
 ​  
 ​  
 ​  
 int main() {  
     //初始化第一个城市  
     x[0] = 0;  
 ​  
     TspDFS\_1(1);  
 ​  
 ​  
 ​  
     cout << "v\_best:";  
     cout << v\_best << endl;  
     cout << "x\_best:";  
     for (int i = 0; i < n; i++){  
         cout << x\_best[i] + 1 << '-';  
    }  
     cout << 1 << endl;  
 }

为了减小筛选剩余可选城市的运算量，我们使用了一个哈希表来替代遍历：



（下标从0开始，下表为0代表1号城市）

**事实上可以将数组替换为STL中的vector,并直接使用find函数进行查找。但vector扩容的过程需要大量的时间，且本任务的n大小通常是可控的，所以仍使用普通的数组进行求解。**

而因为我们需要固定在1号城市出发，所以我们需要将map中下标0的数置为1，表示从1号城市开始；

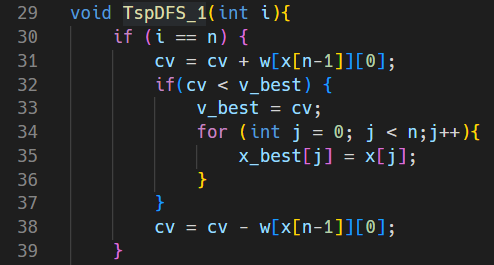
同时，我们需要置x[0]=0,因为路径一定是从1号城市开始；此外，在一开始调用递归函数时，参数i的值为1.因为第0层已经被初始化完成了，如下图所示。



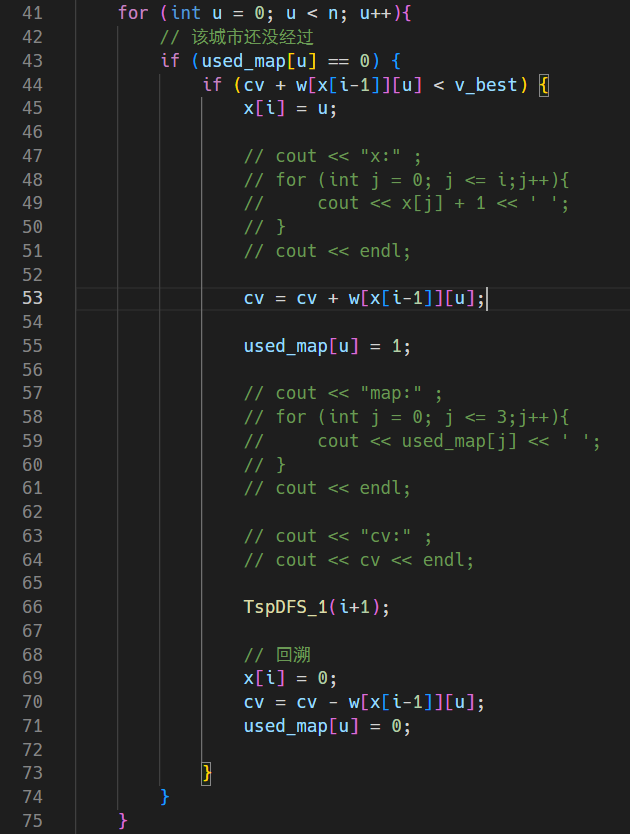
使用递归，首先我们需要结束状态。结束状态是到第n层。到第n层之后，我们需要返回到1号城市。如果此时cv小于最短路径，则更新最短路径序列x\_best和最短路径长度v\_best.

**注意：此后需要回溯！否则最后回到1号城市所累加的距离会一直保留在cv中。老师给出的伪代码在这里并没有回溯，这是有问题的。笔者在实现时改正了这个问题。**

结束状态代码如下所示：

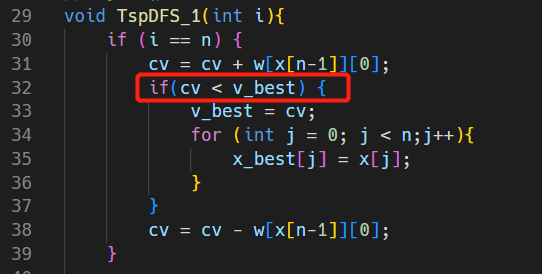


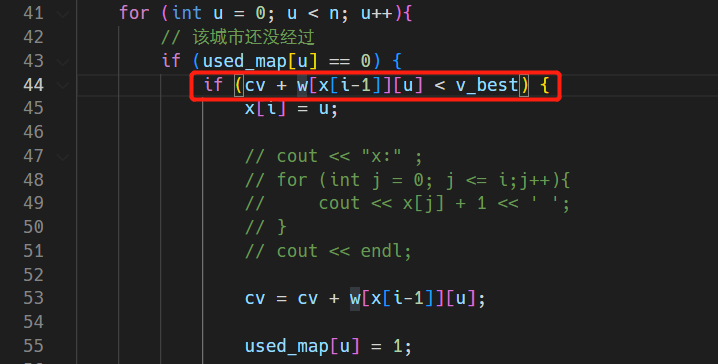
随后对于当前状态，如果不是结束状态，则我们需要找出还没到达过的城市，并一一进行DFS.如上文所述，**我们使用unordered\_map降低了时间复杂度**。此外，我们还使用了剪枝函数：若当前路径长度已经大于最短路径长度了，就不必再继续DFS了.也就是**会过滤掉cv + w[x[i-1]][u] >= v\_best的路径**。而如果小于最短路径长度，则更新x、cv和used\_map，进入DFS的下一步，即TspDFS(i+1).而**TspDFS(i+1)完毕后，我们需要回溯，将x、cv和used\_map的值回调。避免影响到DFS的其他情况。**



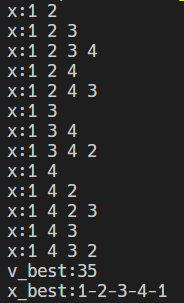
#### 4、画出回溯搜索过程中生成的解空间树，说明发生剪枝的结点，以及树中各个叶结点、非叶结点对应的路径长度

若这两处不取等：

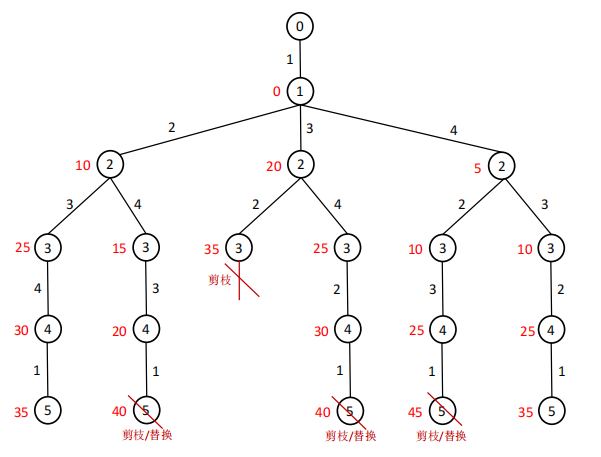




则得到的最短路径序列为1→2→3→4→1.路径长度为35.运行结果为（递归时路径没有打印出最后回到城市1）：

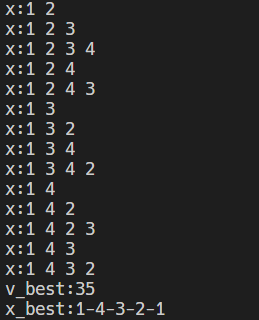


对应的解空间树为：



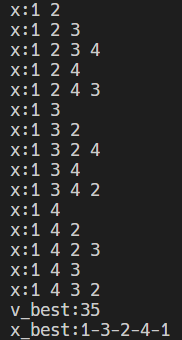
# 发生剪枝的结点、以及树中的路径长度在搜索树中已经标出。

而如果取等，得到的答案则为1→4→3→2→1.对应的剪枝可以从下图的输出看出：

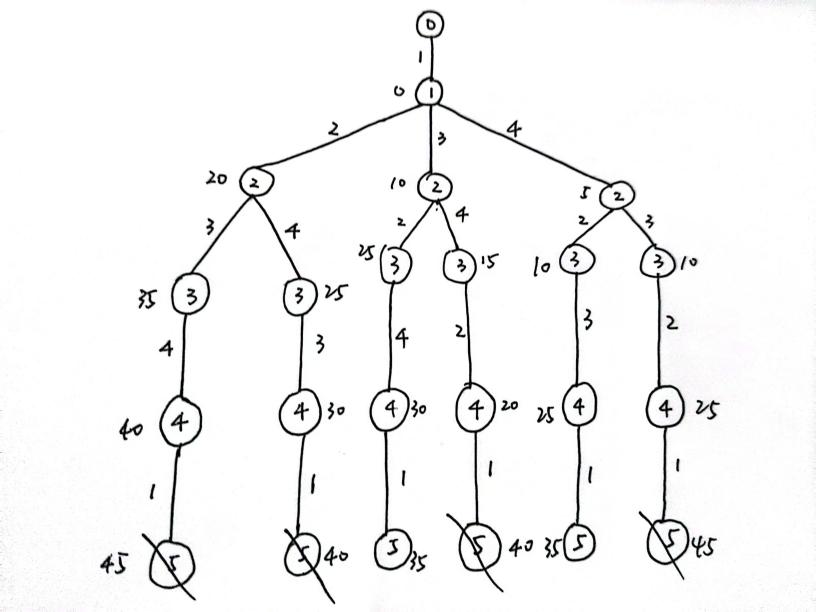


即取的是上面解空间树中最右边的解。

再进行测试以验证正确性：**将序号2、3对调**，则得到的解为：



对应的解空间树为：



而前两个解事实上在一开始并不知道自己会被替换。

# 三、出现问题及解决

(列出你在实验中遇到了哪些问题以及是如何解决的)

1. 递归回溯的伪代码存在问题，在i==n时没有回溯。

解决：如果只是验证查看v\_best和x\_best，是看不出来的。我在验证取等之后是否会出现1→4→3→2→1这个解时发现了这个问题。取等时本该DFS到的路径并没有经过。于是我输出map和cv的值，才发现cv在得到解后就一直多了5.所以据此解决了问题。

# 四、总结

(对所实现算法的总结评价，如时间复杂度，空间复杂度，是否有能够进一步提升的空间，不同实现之间的比较，不同情况下的效率，通过实验对此算法的认识与理解等等)

通过本次实验，我实现了递归回溯法解决TSP问题。

在没有剪枝的情况下，单纯使用该方法的时间复杂度为.但使用剪枝后，计算复杂度将会大大降低。而空间复杂度为，因为存在权重矩阵。

事实上，还能通过非递归的回溯的方法解决，这会进一步节约时间。但实际上，给出的伪代码仍然不足以得到非递归回溯的答案，因为非递归的回溯需要用到额外的空间来存储已经遍历过的情况。但由于时间原因，笔者并没有继续求解得到非递归回溯的答案。