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。

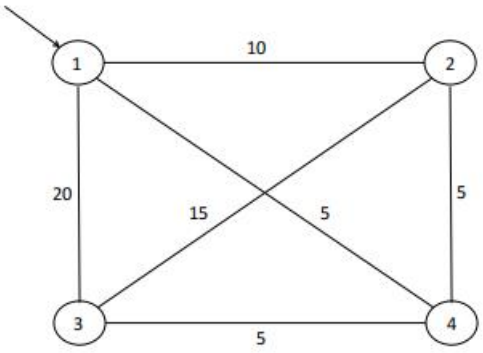


图 1

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：

$$cv + w[x[i-1], x[i]] <= v\_best$$

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：

```
19  unordered_map<int, int> used_map = {
20      //初始化第一个城市
21      {0,1},
22
23      {1,0},
24      {2,0},
25      {3,0},
26  };
```

（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。

```
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;
}
```

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：

```
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];
    }
```

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。**

```
41      for (int u = 0; u < n; u++){
42          // 该城市还没经过
43          if (used_map[u] == 0) {
44              if (cv + w[x[i-1]][u] < v_best) {
45                  x[i] = u;
46
47                  // cout << "x:" ;
48                  // for (int j = 0; j <= i;j++){
49                  //     cout << x[j] + 1 << ' ';
50                  // }
51                  // cout << endl;
52
53                  cv = cv + w[x[i-1]][u];
54
55                  used_map[u] = 1;
56
57                  // cout << "map:" ;
58                  // for (int j = 0; j <= 3;j++){
59                  //     cout << used_map[j] << ' ';
60                  // }
61                  // cout << endl;
62
63                  // cout << "cv:" ;
64                  // cout << cv << endl;
65
66                  TspDFS_1(i+1);
67
68                  // 回溯
69                  x[i] = 0;
70                  cv = cv - w[x[i-1]][u];
71                  used_map[u] = 0;
72
73              }
74          }
75      }
```

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：

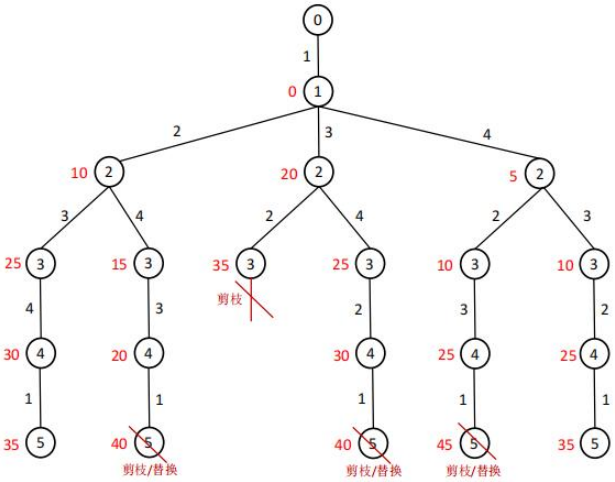
```
29  void TspDFS_1(int i){
30      if (i == n) {
31          cv = cv + w[x[n-1]][0];
32          if(cv < v_best) {
33              v_best = cv;
34              for (int j = 0; j < n;j++){
35                  x_best[j] = x[j];
36              }
37          }
38          cv = cv - w[x[n-1]][0];
39      }
```

```
41      for (int u = 0; u < n; u++){
42          // 该城市还没经过
43          if (used_map[u] == 0) {
44              if (cv + w[x[i-1]][u] < v_best) {
45                  x[i] = u;
46
47                  // cout << "x:" ;
48                  // for (int j = 0; j <= i;j++){
49                  //     cout << x[j] + 1 << ' ';
50                  // }
51                  // cout << endl;
52
53                  cv = cv + w[x[i-1]][u];
54
55                  used_map[u] = 1;
```

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)：

```
x:1 2
x:1 2 3
x:1 2 3 4
x:1 2 4
x:1 2 4 3
x:1 3
x:1 3 4
x:1 3 4 2
x:1 4
x:1 4 2
x:1 4 2 3
x:1 4 3
x:1 4 3 2
v_best:35
x_best:1-2-3-4-1
```

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：

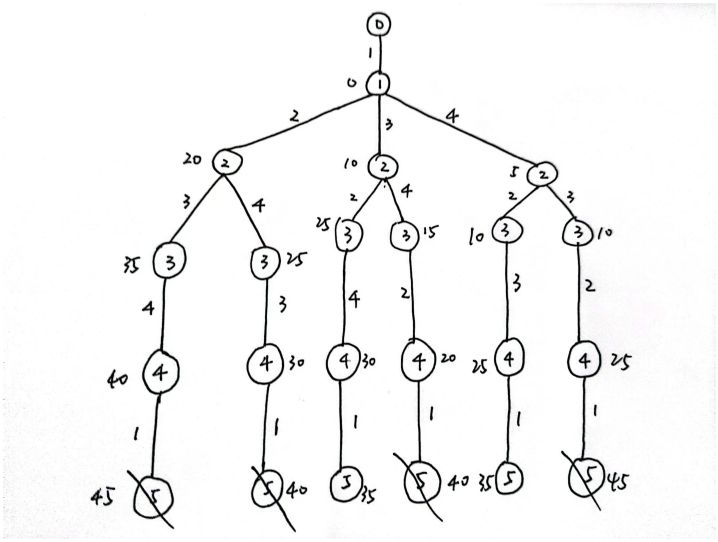
```
x:1 2
x:1 2 3
x:1 2 3 4
x:1 2 4
x:1 2 4 3
x:1 3
x:1 3 2
x:1 3 4
x:1 3 4 2
x:1 4
x:1 4 2
x:1 4 2 3
x:1 4 3
x:1 4 3 2
v_best:35
x_best:1-4-3-2-1
```

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：

```
x:1 2
x:1 2 3
x:1 2 3 4
x:1 2 4
x:1 2 4 3
x:1 3
x:1 3 2
x:1 3 2 4
x:1 3 4
x:1 3 4 2
x:1 4
x:1 4 2
x:1 4 2 3
x:1 4 3
x:1 4 3 2
v_best:35
x_best:1-3-2-4-1
```

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$O((n-1)!)$.但使用剪枝后，计算复杂度将会大大降低。而空间复杂度为$O(n^2)$，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。