

# 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： 马天成

学号： 2020211376

日期： 2022-12-14

目录

一、实验环境 .....2

    1.1 设备规格 .....2

    1.2 操作系统 .....3

    1.3 编程语言&编译器 .....3

    1.4 开发工具 .....3

二、实验内容 .....4

    2.1 实验内容及要求 .....4

    2.2 输入设计 .....4

    2.3 全局变量设计 .....5

    2.4 主函数功能设计 .....5

    2.5 子函数-swap.....5

    2.6 子函数-insert\_print.....6

    2.7 子函数-TSP .....6

    2.8 测试 .....7

三、出现问题及解决.....9

    3.1 下标 .....9

    3.2 思路 .....9

四、总结.....9

    4.1 实现内容 .....9

一、实验环境

1.1 设备规格

设备规格

Legion R7000P2020H

设备名称	PC
处理器	AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz
机带 RAM	16.0 GB (15.9 GB 可用)
设备 ID	370678B7-1FA2-4C35-8BAB-F92D3429406B
产品 ID	00342-35932-44511-AAOEM
系统类型	64 位操作系统, 基于 x64 的处理器
笔和触控	为 10 触摸点提供笔和触控支持

## 1.2 操作系统

### Windows 规格

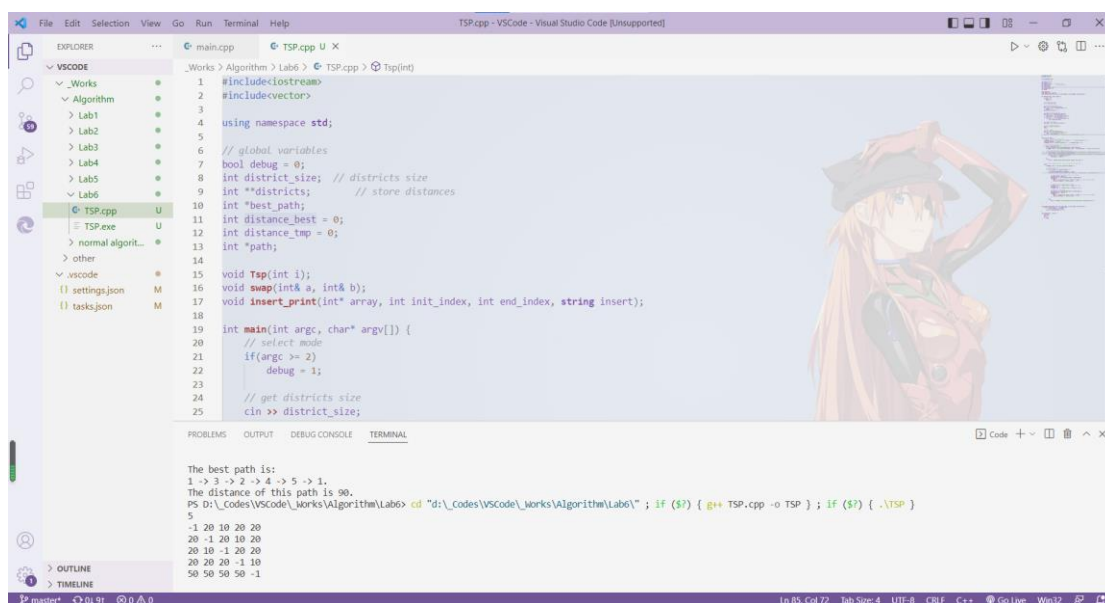
版本	Windows 10 家庭中文版
版本号	21H2
安装日期	2022/4/5
操作系统内部版本	19044.2006
序列号	PF2660CA
体验	Windows Feature Experience Pack 120.2212.4180.0

## 1.3 编程语言&编译器

C++11

```
version : MinGW-W64-builds-4.3.5
user    : nixman
date    : 05.12.2018-10:29:36 AM
```

## 1.4 开发工具



```
File Edit Selection View Go Run Terminal Help
TSP.cpp - VSCode - Visual Studio Code [Unsupported]

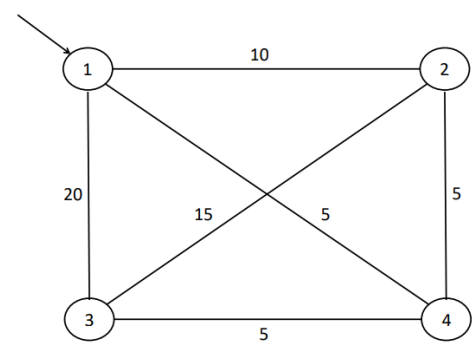
EXPLORER
  _Works
    Algorithm
      Lab1
      Lab2
      Lab3
      Lab4
      Lab5
      Lab6
    TSP.cpp
    TSP.exe
    normal algorithm...
    other
  vscod
  settings.json
  tasks.json

main.cpp TSP.cpp U X
1 #include<iostream>
2 #include<vector>
3
4 using namespace std;
5
6 // global variables
7 bool debug = 0;
8 int district_size; // districts size
9 int **districts; // store distances
10 int *best_path;
11 int distance_best = 0;
12 int distance_tmp = 0;
13 int *path;
14
15 void Tsp(int i);
16 void swap(int& a, int& b);
17 void insert_print(int* array, int init_index, int end_index, string insert);
18
19 int main(int argc, char* argv[]) {
20     // select mode
21     if(argc == 2)
22         debug = 1;
23
24     // get districts size
25     cin >> district_size;
26
27     The best path is:
28     1 -> 3 -> 2 -> 4 -> 5 -> 1.
29     The distance of this path is 90.
30     PS D:\_Codes\VSCode\_works\Algorithm\Lab6> cd "d:\_Codes\VSCode\_works\Algorithm\Lab6" ; if ($?) { g++ TSP.cpp -o TSP } ; if ($?) { .\TSP }
31
32     -1 20 10 20 20
33     20 -1 20 10 20
34     20 10 -1 20 20
35     20 20 20 -1 10
36     50 50 50 -1
```

# 二、实验内容

## 2.1 实验内容及要求

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。采用 C/C++/Java/Python 语言，采用回溯法，使用递归或非递归的方法，求解旅行售货员问题。要求完成以下内容： 1. 设计采用的界限函数（剪枝函数）； 2. 给出回溯过程中对结点采用的剪枝策略。 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； 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。



## 2.2 输入设计

第一行为规模。路径为矩阵。对角线无意义。

```
D:\_Codes\VSCode\_Works\Algorithm\Lab6>TSP.exe debug
5
-1 20 10 20 20
20 -1 20 10 20
20 10 -1 20 20
20 20 20 -1 10
50 50 50 50 -1
```

## 2.3 全局变量设计

```
6 // global variables
7 bool debug = 0;
8 int district_size; // districts size
9 int **districts;    // store distances
10 int *best_path;
11 int distance_best = 0;
12 int distance_tmp = 0;
13 int *path;
```

这个就很简单了。根据英文名称很简单就能猜出。

## 2.4 主函数功能设计

主要功能为读取数据、调用函数处理。带参数会使得进入 debug 模式输出更多信息。

```
19 int main(int argc, char* argv[]) {
20     // select mode
21     if(argc >= 2)
22         debug = 1;
23
24     // get districts size
25     cin >> district_size;
26
27     // initialize first path
28     path = new int[district_size+2];
29     for (int i = 1; i <= district_size; i++)
30         path[i] = i;
31     path[district_size+1] = 1;
32
33     // initialize matrix for dis
34     districts = new int*[district_size+1];
35     for (int i=0; i <= district_size; i++)
36         districts[i] = new int[district_size+1];
37     for (int i=1; i <= district_size; i++)
38         for (int j = 1; j <= district_size; j++)
39             cin >> districts[i][j];
40
41     // allocate best_path
42     best_path = new int[district_size+1];
43
44     // start to find district 2
45     cout << "\nStart to find:\n";
46     Tsp(2);
47
48     // output answer
49     cout << "\nThe best path is:\n";
50     for (int i = 1; i <= district_size; i++)
51         cout << best_path[i] << " -> ";
52     cout << "1.\nThe distance of this path is " << distance_best << ".\n";
53 }
```



## 2.5 子函数-swap

```
120 void swap(int& a, int& b) {
121     int temp;
122     temp = a;
123     a = b;
124     b = temp;
125 }
```

这是最简单的交换函数。

## 2.6 子函数-insert\_print

```
114 void insert_print(int* array, int init_index, int end_index, string insert) {
115     for(int i=init_index; i < end_index; i++)
116         cout << array[i] << insert;
117     cout << array[end_index] << "\n";
118 }
```

这就是简单输出一个数组。主要用来输出路径。

## 2.7 子函数-TSP

```
55 void Tsp(int layer) {
56     // output information of this layer
57     if(debug) cout << "\n----Layer: " << layer << "----\nCurrent path:\n ** ";
58     insert_print(path, 1, layer, " -> ");
59     if(debug) cout << "Current distance:\n ** " << distance_tmp << "\n";
60
61     // has reached the deepest layer, should back to district-1
62     if (layer > district_size) {
63         int last_distance = districts[path[district_size]][path[1]];
64         if(debug) cout << "Has reached the deepest. Add " << last_distance << " back to district-1\n";
65
66         // find the better path, update the best_path
67         if (last_distance > 0 && (distance_tmp+last_distance < distance_best || distance_best == 0)) {
68             distance_best = distance_tmp + last_distance;
69             cout << "Better than best_path: update this one for best_path with distance=" << distance_best << ".\n";
70             for (int i=1; i <= district_size; i++)
71                 best_path[i] = path[i];
72         }
73         else
74             cout << "ACTION: worse than best_path: abandon this one.\n";
75     }
76 }
```

```
77 // not reach the deepest layer. still call new TSP for next layer
78 else {
79     if(debug) cout << "Has not reached the deepest.\n";
80
81     // traverse the layer
82     for (int i=layer; i <= district_size; i++) {
83         // tmp better than best, continue
84         if (districts[path[layer-1]][path[i]] > 0 &&
85             (distance_tmp+districts[path[layer-1]-1][path[i]] < distance_best || distance_best == 0)) {
86
87             // swap & distance_tmp add x[layer-1] -> x[layer]
88             swap(path[layer], path[i]);
89             distance_tmp += districts[path[layer-1]][path[layer]];
90             if(debug) {
91                 cout << "\tSwap before call: layer=" << layer << ", i=" << i << ".\n\t";
92                 insert_print(path, 1, district_size, " -> ");
93             }
94
95             // continue call new TSP for next layer
96             if(debug) cout << "\tCall TSP(" << layer+1 << ")\n";
97             Tsp(layer+1);
98             if(debug) cout << "\tReturn TSP(" << layer+1 << ")\n";
99
100             // restore x and distance
101             distance_tmp -= districts[path[layer-1]][path[layer]];
102             swap(path[layer], path[i]);
103             if(debug) {
104                 cout << "\tSwap after return: layer=" << layer << ", i=" << i << ".\n\t";
105                 insert_print(path, 1, district_size, " -> ");
106             }
107         }
108         else
109             cout << "ACTION: Current distance worse than best_path: pruning this one.\n";
110     }
111 }
112 }
```

这其实就是核心的模块了。我们这里选择递归调用的模板进行编写。  
有一个大的逻辑分支：

```
61 // has reached the deepest layer, should back to district-1
62 > if (layer > district_size) { ...
76
77 // not reach the deepest layer. still call new TSP for next layer
78 > else { ...
```

1. 假如层数走完所有点，走到最后一个点。那就更新当前的距离，函数结束。
2. 假如层数没走完，那就在每一层进行所有的尝试：（由 swap 实现）

因为是排列数，所以根据当前层数来循环向下的数字就可以。记得还原现场。

- 未到达底层-剪枝：

```
81 // traverse the layer
82 for (int i=layer; i <= district_size; i++) {
83     // tmp better than best, continue
84     if (districts[path[layer-1]][path[i]] > 0 &&
85 > (distance_tmp+districts[path[layer-1]-1][path[i]] < distance_best || distance_best == 0)) { ...
108     else
109     |     cout << "ACTION: Current distance worse than best_path: pruning this one.\n";
110 }
```

对于当前尝试的点，假如距离已经超出 best，没必要进行继续了。直接退出函数。

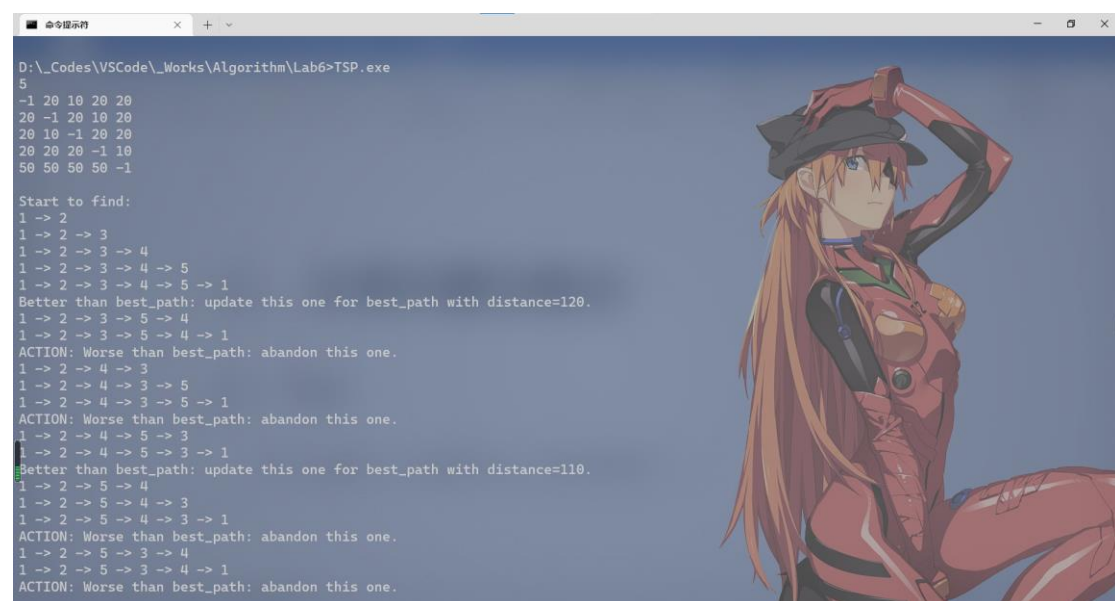
- 到达底层-舍弃：

```
66 // find the better path, update the best_path
67 > if (last_distance > 0 && (distance_tmp+last_distance < distance_best || distance_best == 0)) { ...
73 else
74 |     cout << "ACTION: Worse than best_path: abandon this one.\n";
```

对于当前生成的路径，假如距离大于 best，那么舍弃，不更新 best。

## 2.8 测试

正常模式：



```
D:\_Codes\VSCode\_Works\Algorithm\Lab6>TSP.exe
5
-1 20 10 20 20
20 -1 20 10 20
20 10 -1 20 20
20 20 20 -1 10
50 50 50 50 -1

Start to find:
1 -> 2
1 -> 2 -> 3
1 -> 2 -> 3 -> 4
1 -> 2 -> 3 -> 4 -> 5
1 -> 2 -> 3 -> 4 -> 5 -> 1
Better than best_path: update this one for best_path with distance=120.
1 -> 2 -> 3 -> 5 -> 4
1 -> 2 -> 3 -> 5 -> 4 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 2 -> 4 -> 3
1 -> 2 -> 4 -> 3 -> 5
1 -> 2 -> 4 -> 3 -> 5 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 2 -> 4 -> 5 -> 3
1 -> 2 -> 4 -> 5 -> 3 -> 1
Better than best_path: update this one for best_path with distance=110.
1 -> 2 -> 5 -> 4
1 -> 2 -> 5 -> 4 -> 3
1 -> 2 -> 5 -> 4 -> 3 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 2 -> 5 -> 3 -> 4
1 -> 2 -> 5 -> 3 -> 4 -> 1
ACTION: Worse than best_path: abandon this one.
```



```
命令提示符
1 -> 4 -> 2 -> 3 -> 5 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 4 -> 2 -> 5 -> 3
1 -> 4 -> 2 -> 5 -> 3 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 4 -> 5 -> 2
1 -> 4 -> 5 -> 2 -> 3
ACTION: Current distance worse than best_path: pruning this one.
1 -> 4 -> 5 -> 3 -> 2
1 -> 4 -> 5 -> 3 -> 2 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 5 -> 3
1 -> 5 -> 3 -> 4
1 -> 5 -> 3 -> 4 -> 2
ACTION: Current distance worse than best_path: pruning this one.
1 -> 5 -> 3 -> 2 -> 4
ACTION: Current distance worse than best_path: pruning this one.
1 -> 5 -> 4 -> 3
1 -> 5 -> 4 -> 3 -> 2
1 -> 5 -> 4 -> 3 -> 2 -> 1
ACTION: Worse than best_path: abandon this one.
1 -> 5 -> 4 -> 2 -> 3
ACTION: Current distance worse than best_path: pruning this one.
1 -> 5 -> 2 -> 4
ACTION: Current distance worse than best_path: pruning this one.
1 -> 5 -> 2 -> 3 -> 4
ACTION: Current distance worse than best_path: pruning this one.

The best path is:
1 -> 3 -> 2 -> 4 -> 5 -> 1.
The distance of this path is 90.

D:\_Codes\VSCode\Works\Algorithm\Lab6>TSP.exe debug
```

debug 模式:

```
命令提示符
D:\_Codes\VSCode\Works\Algorithm\Lab6>TSP.exe debug
5
-1 20 10 20 20
20 -1 20 10 20
20 10 -1 20 20
20 20 20 -1 10
50 50 50 50 -1

Start to find:

----Layer: 2----
Current path:
** 1 -> 2
Current distance:
** 0
Has not reached the deepest.
Swap before call: layer=2, i=2.
1 -> 2 -> 3 -> 4 -> 5
Call TSP(3)

----Layer: 3----
Current path:
** 1 -> 2 -> 3
Current distance:
*** 20
Has not reached the deepest.
Swap before call: layer=3, i=3.
1 -> 2 -> 3 -> 4 -> 5
Call TSP(4)

----Layer: 4----
Current path:
** 1 -> 2 -> 3 -> 4
Current distance:
** 70
Has not reached the deepest.
ACTION: Current distance worse than best_path: pruning this one.
Swap before call: layer=4, i=5.
1 -> 5 -> 2 -> 3 -> 4
Call TSP(5)

----Layer: 5----
Current path:
** 1 -> 5 -> 2 -> 3 -> 4
Current distance:
** 90
Has not reached the deepest.
ACTION: Current distance worse than best_path: pruning this one.
Return TSP(5)
Swap after return: layer=4, i=5.
1 -> 5 -> 2 -> 4 -> 3
Return TSP(4)
Swap after return: layer=3, i=5.
1 -> 5 -> 3 -> 4 -> 2
Return TSP(3)
Swap after return: layer=2, i=5.
1 -> 2 -> 3 -> 4 -> 5

The best path is:
1 -> 3 -> 2 -> 4 -> 5 -> 1.
The distance of this path is 90.
```

可以看到 debug 模式中，清楚地看出函数调用关系和当前的路径。



经过一系列的测试，都是没有问题的：

```
D:\_Codes\VSCode\_Works\Algorithm\Lab6>TSP.exe
2
-1 10
10 -1

Start to find:
1 -> 2
1 -> 2 -> 1
Better than best_path: update this one for best_path with distance=20.

The best path is:
1 -> 2 -> 1.
The distance of this path is 20.
```

## 三、出现问题及解决

### 3.1 下标

下标在编写的时候经常出错。但还是解决了。

### 3.2 思路

对整个 swap 以及还原没有很好的认识。但是我将所有的东西输出之后，一切都很明了。

## 四、总结

### 4.1 实现内容

1. 设计采用的界限函数（剪枝函数）；  
在上文中给了很清楚。减去距离已经大于 best 的枝。
2. 给出回溯过程中对结点采用的剪枝策略。  
减去距离已经大于 best 的枝。
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；  
输出的最终结果。

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。

在 debug 模式中，很清楚自己当前的层数和路径长度，以及函数调用关系和 swap 关系。