

## 华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析

年级：19 级

上机实践成绩：

指导教师：金澈清

姓名：龚敬洋

上机实践名称：路径规划

学号：

上机实践日期：

10195501436

2020/12/25

上机实践编号：No. 12

组号：1-436

## 一、目的

1. 熟悉算法设计的基本思想
2. 掌握最小生成树算法的思路

## 二、内容与设计思想

川西风光几枚，以下图片是川西路线图。张三是旅游爱好者，他从成都出发自驾到西藏江达。



- 1) 从成都到江达的最短自驾路线是什么？可以用 Dijkstra 算法来求解。
- 2) 张三把理塘列为必游之地。怎么规划路线，使得总行程最短？
- 3) 张三觉得理塘风景很美，道孚也不错，两个地方如果能够去一个地方的话就心满意足了。应该怎么安排行程使得总行程最短？
- 4) 张三在规划线路的时候，发现不同路况行驶速度不一样。地图中粗的路径表示平均时速可以达到 80 公里每小时，而细的路径表示平均时速仅仅有每小时 60 公里每小时。那么用时最短的路径是哪一条？
- 5) （思考题）考虑到 Dijkstra 算法仅仅从一段开始寻找路径，效率不高。李教授想到一个高招，就是同时从出发地和目的地进行搜索，扩展搜索节点，然后两个方向扩展的路径会在中途相遇，则拼接起来的路径就是最短路径。如何实现李教授这个想法？

## 三、使用环境

推荐使用 C/C++ 集成编译环境。

## 四、实验过程

### 1. 编写相关实验代码

#### (1) 从成都到江达的最短路

```
1. #include <iostream>
2. #include <fstream>
3. #include <stack>
4. #include <cstring>
5. #include <map>
6. #define SUP 100000005
7. using namespace std;
8. int graph[50][50], d[50], visit[50] = {0}, pred[50] = {0};
9. //Procedure of finding the shortest path
10. void find_path(int from, int to, int n){
11.     int flag = 0;
12.     for(int i = 0; i < n; i++) d[i] = SUP;
13.     for(int i = 0; i < n; i++) visit[i] = 0;
14.     d[from] = 0;
15.     while (!flag){
16.         int min_dist = SUP, min_idx = -1;
17.         for(int i = 0; i < n; i++){
18.             if (visit[i] == 0 && d[i] < min_dist){
19.                 min_dist = d[i];
20.                 min_idx = i;
21.             }
22.         }
23.         visit[min_idx] = 1;
24.         if(min_idx == to) break;
25.         for(int i = 0; i < n; i++){
26.             if (visit[i] == 0 && graph[min_idx][i] != SUP){
27.                 if(d[i] > d[min_idx] + graph[min_idx][i]) {
28.                     d[i] = d[min_idx] + graph[min_idx][i];
29.                     pred[i] = min_idx;
30.                 }
31.             }
32.         }
33.         flag = 1;
34.         for (int i = 0; i < n; i++) {
35.             if (visit[i] == 0){
36.                 flag = 0;
37.                 break;
38.             }
39.         }
40.     }
41. }
42. int main() {
43.     ifstream fin("data.txt");
44.     stack<int> path;
45.     map<string, int> location_index;
46.     map<int, string> inv_location_index;
47.     int n = 0, start, end, flag = 0;
48.     string start_city, end_city;
49.     for(int i = 0; i < 50; i++)
50.         for(int j = 0; j < 50; j++) {
51.             graph[i][j] = SUP;
52.             graph[j][i] = SUP;
53.         }
54.     //Read in data & build the graph
55.     while(!fin.eof()){
56.         string from, to;
```

```

57.     int from_index, to_index, distance, speed;
58.     fin>>from>>to>>distance>>speed;
59.     if (location_index.find(from) == location_index.end()){
60.         from_index = n;
61.         location_index[from] = from_index;
62.         inv_location_index[from_index] = from;
63.         n++;
64.     }
65.     else{
66.         from_index = location_index[from];
67.     }
68.     if (location_index.find(to) == location_index.end()){
69.         to_index = n;
70.         location_index[to] = to_index;
71.         inv_location_index[to_index] = to;
72.         n++;
73.     }
74.     else{
75.         to_index = location_index[to];
76.     }
77.     graph[from_index][to_index] = distance;
78.     graph[to_index][from_index] = distance;
79. }
80. start_city = "成都";
81. end_city = "江达";
82. start = location_index[start_city];
83. end = location_index[end_city];
84. find_path(start, end, n);
85. //Recall the path
86. int cur = end;
87. path.push(end);
88. while(cur != start){
89.     path.push(pred[cur]);
90.     cur = pred[cur];
91. }
92. cout<<"路径: ";
93. while(!path.empty()) {
94.     cout << inv_location_index[path.top()] << " ";
95.     path.pop();
96. }
97. cout<<endl;
98. cout<<"Total: "<<d[end]<<"km"<<endl;
99. fin.close();
100.    return 0;
101. }

```

(2) 从成都经过理塘再到江达的最短路 ( 省略号部分与(1)代码一致 )

```

1. ...
2. int main() {
3.     ...
4.     int tot_dist = 0;
5.     ...
6.     int n = 0, start, mid, end;
7.     string start_city, mid_city, end_city;
8.     ...
9.     start_city = "成都";
10.    mid_city = "理塘";
11.    end_city = "江达";
12.    start = location_index[start_city];
13.    mid = location_index[mid_city];
14.    end = location_index[end_city];
15.    //Find path from middle point to end point

```

```
16.     find_path(mid, end, n);
17.     int cur = end;
18.     path.push(end);
19.     while(cur != mid){
20.         path.push(pred[cur]);
21.         cur = pred[cur];
22.     }
23.     tot_dist += d[end];
24.     //Find path from start point to middle point
25.     find_path(start, mid, n);
26.     cur = pred[mid];
27.     path.push(pred[mid]);
28.     while(cur != start){
29.         path.push(pred[cur]);
30.         cur = pred[cur];
31.     }
32.     ...
33.     tot_dist += d[mid];
34.     cout<<endl;
35.     cout<<"Total: "<<tot_dist<<"km"<<endl;
36.     ...
37. }
```

(3) 从成都经过理塘或道孚再到江达的最短路（省略号部分与(1)代码一致）

```
1. ...
2. int main() {
3.     ...
4.     stack<int> path_1, path_2;
5.     int tot_dist_1 = 0, tot_dist_2 = 0;
6.     ...
7.     int n = 0, start, mid_1, mid_2, end;
8.     string start_city, mid_city_1, mid_city_2, end_city;
9.     ...
10.    start_city = "成都";
11.    mid_city_1 = "理塘";
12.    mid_city_2 = "道孚";
13.    end_city = "江达";
14.    start = location_index[start_city];
15.    mid_1 = location_index[mid_city_1];
16.    mid_2 = location_index[mid_city_2];
17.    end = location_index[end_city];
18.    //Find path 1
19.    find_path(mid_1, end, n);
20.    int cur = end;
21.    path_1.push(end);
22.    while(cur != mid_1){
23.        path_1.push(pred[cur]);
24.        cur = pred[cur];
25.    }
26.    tot_dist_1 += d[end];
27.    find_path(start, mid_1, n);
28.    cur = pred[mid_1];
29.    path_1.push(pred[mid_1]);
30.    while(cur != start){
31.        path_1.push(pred[cur]);
32.        cur = pred[cur];
33.    }
34.    tot_dist_1 += d[mid_1];
35.    //Find path 2
36.    find_path(mid_2, end, n);
37.    cur = end;
38.    path_2.push(end);
39.    while(cur != mid_2){
```

```

40.     path_2.push(pred[cur]);
41.     cur = pred[cur];
42. }
43. tot_dist_2 += d[end];
44. find_path(start, mid_2, n);
45. cur = pred[mid_2];
46. path_2.push(pred[mid_2]);
47. while(cur != start){
48.     path_2.push(pred[cur]);
49.     cur = pred[cur];
50. }
51. tot_dist_2 += d[mid_2];
52. //Compare and print the smaller one
53. if(tot_dist_1 <= tot_dist_2){
54.     cout<<"途径: 理塘"<<endl;
55.     cout<<"路径: ";
56.     while(!path_1.empty()){
57.         int p = path_1.top();
58.         cout<<inv_location_index[p]<<" ";
59.         path_1.pop();
60.     }
61.     cout<<endl;
62.     cout<<"Total: "<<tot_dist_1<<"km"<<endl;
63. }
64. else{
65.     cout<<"途径: 道孚"<<endl;
66.     cout<<"路径: ";
67.     while(!path_2.empty()){
68.         int p = path_2.top();
69.         cout<<inv_location_index[p]<<" ";
70.         path_2.pop();
71.     }
72.     cout<<endl;
73.     cout<<"Total: "<<tot_dist_2<<"km"<<endl;
74. }
75. ...
76. }

```

#### (4) 有时速限制时的最短路 ( 省略号部分与(1)代码一致 )

```

1. ...
2. double graph[50][50], d[50];
3. int graph_dist[50][50], visit[50] = {0}, pred[50] = {0};
4. void find_path(int from, int to, int n){
5.     ...
6.     while (!flag){
7.         double min_dist = SUP;
8.         ...
9.     }
10. }
11. int main() {
12.     ...
13.     int n = 0, start, end, flag = 0, tot_dist = 0;
14.     ...
15.     while(!fin.eof()){
16.         ...
17.         graph[from_index][to_index] = (double)distance / speed;
18.         graph[to_index][from_index] = (double)distance / speed;
19.         graph_dist[from_index][to_index] = distance;
20.         graph_dist[to_index][from_index] = distance;
21.     }
22.     ...
23.     while(cur != start){
24.         path.push(pred[cur]);

```

```

25.         tot_dist += graph_dist[cur][pred[cur]];
26.         cur = pred[cur];
27.     }
28.     ...
29.     cout<<"Total: "<<tot_dist<<"km"<<endl;
30.     ...
31. }

```

## (5) 分别从两边开始搜索的最短路

```

1. #include <iostream>
2. #include <fstream>
3. #include <stack>
4. #include <cstring>
5. #include <map>
6. #define SUP 100000005
7. using namespace std;
8. int graph[50][50], d1[50], d2[50], visit_1[50], visit_2[50], pred_1[50] = {0}, pred_2[50]
   = {0}, inter_idx;
9. void find_path(int from, int to, int n){
10.     int flag = 0;
11.     for(int i = 0; i < n; i++) d1[i] = SUP;
12.     for(int i = 0; i < n; i++) d2[i] = SUP;
13.     for(int i = 0; i < n; i++) visit_1[i] = 0;
14.     for(int i = 0; i < n; i++) visit_2[i] = 0;
15.     d1[from] = 0;
16.     d2[to] = 0;
17.     while (!flag){
18.         int min_dist = SUP, min_idx_1 = -1, min_idx_2 = -1;
19.         for(int i = 0; i < n; i++){
20.             if (visit_1[i] == 0 && d1[i] < min_dist){
21.                 min_dist = d1[i];
22.                 min_idx_1 = i;
23.             }
24.         }
25.         visit_1[min_idx_1] = 1;
26.         min_dist = SUP;
27.         for(int i = 0; i < n; i++){
28.             if (visit_2[i] == 0 && d2[i] < min_dist){
29.                 min_dist = d2[i];
30.                 min_idx_2 = i;
31.             }
32.         }
33.         visit_2[min_idx_2] = 1;
34.         //If one searching route meet the other, record the intersection point and return
35.         if(visit_1[min_idx_2] == 1 || visit_2[min_idx_1] == 1) {
36.             inter_idx = visit_1[min_idx_2] ? min_idx_2 : min_idx_1;
37.             break;
38.         }
39.         if(min_idx_1 == to || min_idx_2 == from) break;
40.         //Update shortest path from start point
41.         for(int i = 0; i < n; i++){
42.             if (visit_1[i] == 0 && graph[min_idx_1][i] != SUP){
43.                 if(d1[i] > d1[min_idx_1] + graph[min_idx_1][i]) {
44.                     d1[i] = d1[min_idx_1] + graph[min_idx_1][i];
45.                     pred_1[i] = min_idx_1;
46.                 }
47.             }
48.         }
49.         //Update shortest path from end point
50.         for(int i = 0; i < n; i++){
51.             if (visit_2[i] == 0 && graph[min_idx_2][i] != SUP){
52.                 if(d2[i] > d2[min_idx_2] + graph[min_idx_2][i]) {
53.                     d2[i] = d2[min_idx_2] + graph[min_idx_2][i];

```

```
54.         pred_2[i] = min_idx_2;
55.     }
56. }
57. }
58. flag = 1;
59. for (int i = 0; i < n; i++) {
60.     if (visit_1[i] == 0 || visit_2[i] == 0){
61.         flag = 0;
62.         break;
63.     }
64. }
65. }
66. }
67. int main() {
68.     ifstream fin("data.txt");
69.     stack<int> path;
70.     map<string, int> location_index;
71.     map<int, string> inv_location_index;
72.     int n = 0, start, end, flag = 0, tot_dist = 0;
73.     string start_city, end_city;
74.     for(int i = 0; i < 50; i++)
75.         for(int j = 0; j < 50; j++) {
76.             graph[i][j] = SUP;
77.             graph[j][i] = SUP;
78.         }
79.     while(!fin.eof()){
80.         string from, to;
81.         int from_index, to_index, distance, speed;
82.         fin>>from>>to>>distance>>speed;
83.         if (location_index.find(from) == location_index.end()){
84.             from_index = n;
85.             location_index[from] = from_index;
86.             inv_location_index[from_index] = from;
87.             n++;
88.         }
89.         else{
90.             from_index = location_index[from];
91.         }
92.         if (location_index.find(to) == location_index.end()){
93.             to_index = n;
94.             location_index[to] = to_index;
95.             inv_location_index[to_index] = to;
96.             n++;
97.         }
98.         else{
99.             to_index = location_index[to];
100.        }
101.        graph[from_index][to_index] = distance;
102.        graph[to_index][from_index] = distance;
103.    }
104.    start_city = "成都";
105.    end_city = "江达";
106.    start = location_index[start_city];
107.    end = location_index[end_city];
108.    find_path(start, end, n);
109.    int cur = inter_idx;
110.    path.push(inter_idx);
111.    while(cur != start){
112.        path.push(pred_1[cur]);
113.        cur = pred_1[cur];
114.    }
115.    tot_dist += d1[inter_idx];
116.    cout<<"路径: ";
117.    while(!path.empty()) {
118.        cout << inv_location_index[path.top()] << " ";
```

```

119.         path.pop();
120.     }
121.     cur = inter_idx;
122.     while(cur != end){
123.         cout<<inv_location_index[pred_2[cur]]<<" ";
124.         cur = pred_2[cur];
125.     }
126.     tot_dist += d2[inter_idx];
127.     cout<<endl;
128.     cout<<"Total: "<<tot_dist<<"km"<<endl;
129.     fin.close();
130.     return 0;
131. }

```

## 2. 写出算法的思路。

### (1) 从成都到江达的最短路

使用 Dijkstra 算法思想，每次从未完成点集中选择离源点距离最短的加入已完成点集，并更新与其相邻所有点到源点的最短距离及对应的前继节点编号，最终即可得到源点到目标点的最短路径。

### (2) 从成都经过理塘再到江达的最短路

使用(1)的算法先找出成都到理塘的最短路径，再以理塘为起点找出到江达的最短路径，拼在一起即为满足要求的最短路。

### (3) 从成都经过理塘或道孚再到江达的最短路

使用(2)的方法先找出成都经过理塘到江达的最短路径，再找出成都经过道孚到江达的最短路径，比较两者路径长度选择更短的即可。

### (4) 有时速限制时的最短路

在建图时，将最短行驶时间（路程/时速限制）作为每条边的权值，再使用(1)的算法搜索即可。

### (5) 分别从两边开始搜索的最短路

使用(1)的算法，分别从两边开始遍历图。当其中一个遍历过程遍历到另一个的路径上时，记录相遇节点并停止遍历。再分别回溯即可拼接成完整的最短路径。

## 五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

旅行路径规划问题本质上即为单源最短路问题，可使用 Dijkstra 算法在 $O(E^2)$ 的时间内找到最短路。

第(1)(3)(4)(5)问的最短路径结果如下：



第(2)的最短路径结果如下：