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**：10195501436	**上机实践日期**：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

```
#include <iostream>
#include <fstream>
#include <stack>
#include <cstring>
#include <map>
#define SUP 100000005
using namespace std;
int graph[50][50], d[50], visit[50] = {0}, pred[50] = {0};
//Procedure of finding the shortest path
void find_path(int from, int to, int n){
    int flag = 0;
    for(int i = 0; i < n; i++) d[i] = SUP;
    for(int i = 0; i < n; i++) visit[i] = 0;
    d[from] = 0;
    while (!flag){
        int min_dist = SUP, min_idx = -1;
        for(int i = 0; i < n; i++){
            if (visit[i] == 0 && d[i] < min_dist){
                min_dist = d[i];
                min_idx = i;
            }
        }
        visit[min_idx] = 1;
        if(min_idx == to) break;
        for(int i = 0; i < n; i++){
            if (visit[i] == 0 && graph[min_idx][i] != SUP){
                if(d[i] > d[min_idx] + graph[min_idx][i]) {
                    d[i] = d[min_idx] + graph[min_idx][i];
                    pred[i] = min_idx;
                }
            }
        }
        flag = 1;
        for (int i = 0; i < n; i++) {
            if (visit[i] == 0){
                flag = 0;
                break;
            }
        }
    }
}
int main() {
    ifstream fin("data.txt");
    stack<int> path;
    map<string, int> location_index;
    map<int, string> inv_location_index;
    int n = 0, start, end, flag = 0;
    string start_city, end_city;
    for(int i = 0; i < 50; i++)
        for(int j = 0; j < 50; j++) {
            graph[i][j] = SUP;
            graph[j][i] = SUP;
        }
    //Read in data & build the graph
    while(!fin.eof()){
        string from, to;
```

```
        int from_index, to_index, distance, speed;
        fin>>from>>to>>distance>>speed;
        if (location_index.find(from) == location_index.end()){
            from_index = n;
            location_index[from] = from_index;
            inv_location_index[from_index] = from;
            n++;
        }
        else{
            from_index = location_index[from];
        }
        if (location_index.find(to) == location_index.end()){
            to_index = n;
            location_index[to] = to_index;
            inv_location_index[to_index] = to;
            n++;
        }
        else{
            to_index = location_index[to];
        }
        graph[from_index][to_index] = distance;
        graph[to_index][from_index] = distance;
    }
    start_city = "成都";
    end_city = "江达";
    start = location_index[start_city];
    end = location_index[end_city];
    find_path(start, end, n);
    //Recall the path
    int cur = end;
    path.push(end);
    while(cur != start){
        path.push(pred[cur]);
        cur = pred[cur];
    }
    cout<<"路径: ";
    while(!path.empty()) {
        cout << inv_location_index[path.top()] << " ";
        path.pop();
    }
    cout<<endl;
    cout<<"Total: "<<d[end]<<"km"<<endl;
    fin.close();
          return 0;
      }
```

(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）

```
...
int main() {
    ...
    int tot_dist = 0;
    ...
    int n = 0, start, mid, end;
    string start_city, mid_city, end_city;
    ...
    start_city = "成都";
    mid_city = "理塘";
    end_city = "江达";
    start = location_index[start_city];
    mid = location_index[mid_city];
    end = location_index[end_city];
    //Find path from middle point to end point
```

```
    find_path(mid, end, n);
    int cur = end;
    path.push(end);
    while(cur != mid){
        path.push(pred[cur]);
        cur = pred[cur];
    }
    tot_dist += d[end];
    //Find path from start point to middle point
    find_path(start, mid, n);
    cur = pred[mid];
    path.push(pred[mid]);
    while(cur != start){
        path.push(pred[cur]);
        cur = pred[cur];
    }
    ...
    tot_dist += d[mid];
    cout<<endl;
    cout<<"Total: "<<tot_dist<<"km"<<endl;
    ...
}
```

(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）

```
...
int main() {
    ...
    stack<int> path_1, path_2;
    int tot_dist_1 = 0, tot_dist_2 = 0;
    ...
    int n = 0, start, mid_1, mid_2, end;
    string start_city, mid_city_1, mid_city_2, end_city;
    ...
    start_city = "成都";
    mid_city_1 = "理塘";
    mid_city_2 = "道孚";
    end_city = "江达";
    start = location_index[start_city];
    mid_1 = location_index[mid_city_1];
    mid_2 = location_index[mid_city_2];
    end = location_index[end_city];
    //Find path 1
    find_path(mid_1, end, n);
    int cur = end;
    path_1.push(end);
    while(cur != mid_1){
        path_1.push(pred[cur]);
        cur = pred[cur];
    }
    tot_dist_1 += d[end];
    find_path(start, mid_1, n);
    cur = pred[mid_1];
    path_1.push(pred[mid_1]);
    while(cur != start){
        path_1.push(pred[cur]);
        cur = pred[cur];
    }
    tot_dist_1 += d[mid_1];
    //Find path 2
    find_path(mid_2, end, n);
    cur = end;
    path_2.push(end);
    while(cur != mid_2){
```

```
        path_2.push(pred[cur]);
        cur = pred[cur];
    }
    tot_dist_2 += d[end];
    find_path(start, mid_2, n);
    cur = pred[mid_2];
    path_2.push(pred[mid_2]);
    while(cur != start){
        path_2.push(pred[cur]);
        cur = pred[cur];
    }
    tot_dist_2 += d[mid_2];
    //Compare and print the smaller one
    if(tot_dist_1 <= tot_dist_2){
        cout<<"途径：理塘"<<endl;
        cout<<"路径：";
        while(!path_1.empty()){
            int p = path_1.top();
            cout<<inv_location_index[p]<<" ";
            path_1.pop();
        }
        cout<<endl;
        cout<<"Total: "<<tot_dist_1<<"km"<<endl;
    }
    else{
        cout<<"途径：道孚"<<endl;
        cout<<"路径：";
        while(!path_2.empty()){
            int p = path_2.top();
            cout<<inv_location_index[p]<<" ";
            path_2.pop();
        }
        cout<<endl;
        cout<<"Total: "<<tot_dist_2<<"km"<<endl;
    }
    ...
}
```

(4) 有时速限制时的最短路（省略号部分与(1)代码一致）

```
...
double graph[50][50], d[50];
int graph_dist[50][50], visit[50] = {0}, pred[50] = {0};
void find_path(int from, int to, int n){
    ...
    while (!flag){
        double min_dist = SUP;
        ...
    }
}
int main() {
    ...
    int n = 0, start, end, flag = 0, tot_dist = 0;
    ...
    while(!fin.eof()){
        ...
        graph[from_index][to_index] = (double)distance / speed;
        graph[to_index][from_index] = (double)distance / speed;
        graph_dist[from_index][to_index] = distance;
        graph_dist[to_index][from_index] = distance;
    }
    ...
    while(cur != start){
        path.push(pred[cur]);
```

```
        tot_dist += graph_dist[cur][pred[cur]];
        cur = pred[cur];
    }
    ...
    cout<<"Total: "<<tot_dist<<"km"<<endl;
    ...
}
```

(5) 分别从两边开始搜索的最短路

```
#include <iostream>
#include <fstream>
#include <stack>
#include <cstring>
#include <map>
#define SUP 100000005
using namespace std;
int graph[50][50], d1[50], d2[50], visit_1[50], visit_2[50], pred_1[50] = {0}, pred_2[50] = {0}, inter_idx;
void find_path(int from, int to, int n){
    int flag = 0;
    for(int i = 0; i < n; i++) d1[i] = SUP;
    for(int i = 0; i < n; i++) d2[i] = SUP;
    for(int i = 0; i < n; i++) visit_1[i] = 0;
    for(int i = 0; i < n; i++) visit_2[i] = 0;
    d1[from] = 0;
    d2[to] = 0;
    while (!flag){
        int min_dist = SUP, min_idx_1 = -1, min_idx_2 = -1;
        for(int i = 0; i < n; i++){
            if (visit_1[i] == 0 && d1[i] < min_dist){
                min_dist = d1[i];
                min_idx_1 = i;
            }
        }
        visit_1[min_idx_1] = 1;
        min_dist = SUP;
        for(int i = 0; i < n; i++){
            if (visit_2[i] == 0 && d2[i] < min_dist){
                min_dist = d2[i];
                min_idx_2 = i;
            }
        }
        visit_2[min_idx_2] = 1;
        //If one searching route meet the other, record the intersection point and return
        if(visit_1[min_idx_2] == 1 || visit_2[min_idx_1] == 1) {
            inter_idx = visit_1[min_idx_2] ? min_idx_2 : min_idx_1;
            break;
        }
        if(min_idx_1 == to || min_idx_2 == from) break;
        //Update shortest path from start point
        for(int i = 0; i < n; i++){
            if (visit_1[i] == 0 && graph[min_idx_1][i] != SUP){
                if(d1[i] > d1[min_idx_1] + graph[min_idx_1][i]) {
                    d1[i] = d1[min_idx_1] + graph[min_idx_1][i];
                    pred_1[i] = min_idx_1;
                }
            }
        }
        //Update shortest path from end point
        for(int i = 0; i < n; i++){
            if (visit_2[i] == 0 && graph[min_idx_2][i] != SUP){
                if(d2[i] > d1[min_idx_2] + graph[min_idx_2][i]) {
                    d2[i] = d1[min_idx_2] + graph[min_idx_2][i];
```

```
                    pred_2[i] = min_idx_2;
                }
            }
        }
        flag = 1;
        for (int i = 0; i < n; i++) {
            if (visit_1[i] == 0 || visit_2[i] == 0){
                flag = 0;
                break;
            }
        }
    }
}
int main() {
    ifstream fin("data.txt");
    stack<int> path;
    map<string, int> location_index;
    map<int, string> inv_location_index;
    int n = 0, start, end, flag = 0, tot_dist = 0;
    string start_city, end_city;
    for(int i = 0; i < 50; i++)
        for(int j = 0; j < 50; j++) {
            graph[i][j] = SUP;
            graph[j][i] = SUP;
        }
    while(!fin.eof()){
        string from, to;
        int from_index, to_index, distance, speed;
        fin>>from>>to>>distance>>speed;
        if (location_index.find(from) == location_index.end()){
            from_index = n;
            location_index[from] = from_index;
            inv_location_index[from_index] = from;
            n++;
        }
        else{
            from_index = location_index[from];
        }
        if (location_index.find(to) == location_index.end()){
            to_index = n;
            location_index[to] = to_index;
            inv_location_index[to_index] = to;
            n++;
        }
        else{
            to_index = location_index[to];
               }
               graph[from_index][to_index] = distance;
               graph[to_index][from_index] = distance;
           }
           start_city = "成都";
           end_city = "江达";
           start = location_index[start_city];
           end = location_index[end_city];
           find_path(start, end, n);
           int cur = inter_idx;
           path.push(inter_idx);
           while(cur != start){
               path.push(pred_1[cur]);
               cur = pred_1[cur];
           }
           tot_dist += d1[inter_idx];
       cout<<"路径: ";
           while(!path.empty()) {
               cout << inv_location_index[path.top()] << " ";
```

```
            path.pop();
        }
        cur = inter_idx;
        while(cur != end){
            cout<<inv_location_index[pred_2[cur]]<<" ";
            cur = pred_2[cur];
        }
        tot_dist += d2[inter_idx];
        cout<<endl;
        cout<<"Total: "<<tot_dist<<"km"<<endl;
        fin.close();
        return 0;
    }
```

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

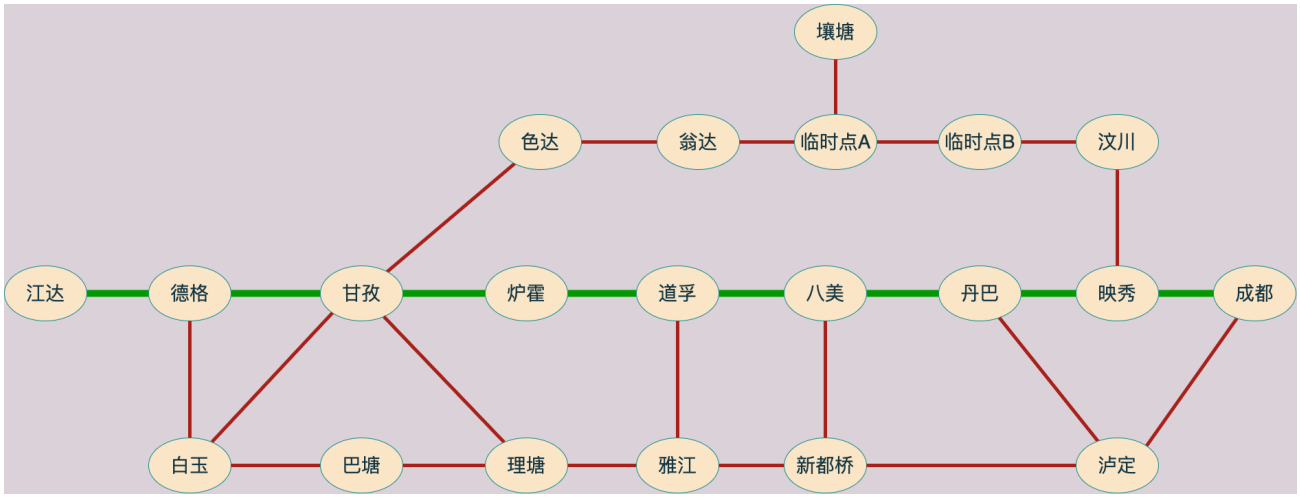
使用(1)的算法，分别从两边开始遍历图。当其中一个遍历过程遍历到另一个的路径上时，记录相遇节点并停止遍历。再分别回溯即可拼接成完整的最短路径。

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：