

【题目】校园地图

一、设计要求：

针对我们的校园布局，设计一款校园平面图。

基本功能及要求如下：

- 1、提供用户操作的菜单和界面，实现系统的所有功能。
- 2、添加校内地点的信息，例如校门、教学楼、食堂、宿舍、快递点等，每个地点视为一个顶点，具体信息包括：名称、坐标、简介等。
- 3、添加道路信息，可理解为顶点之间的边，也可以理解为顶点在路边。
- 4、以上添加的地点和道路信息可以修改和删除，即具备信息的增删改功能。并且所有信息要求以文件的形式存储（如文本文件），格式自行设计。当下次运行程序时，从文件读取地点和道路信息。
- 5、打印校园地图。
- 6、设计出一套铺设线路最短，但能满足每个地点都能通电的方案。两点之间的距离根据坐标计算。输出铺设电路规划以及每条电路的长度和总长度。
- 7、查询校园地点信息，查询该地点到其他任一地点的最短简单路径及距离，并按照距离从近到远的顺序显示。
- 8、求从某一地点出发经过校园所有地点（地点可以重复）且距离最短的路线；求从某一地点出发经过校园所有地点（地点可以重复）并回到源点的最短路线。

二、思路框架:

本项目的代码思路框架描述如下:

首先,通过 **main()函数**中 **screen()函数**进入用户操作界面,通过用户的交互性选择,进行响应。在此层面,从 TXT 文本文件的直接操作角度,可以直接响应实现校园路径及地点的**添加、修改**等操作。

此外,通过 **Createmap()函数**,从 TXT 文本文件中读取数据,构建稳定的邻接矩阵,其中顶点对应地点,权值表示距离(若无穷大,表示无通路)。在此层面,通过建立的邻接矩阵,可直接响应**打印校园地图**。

最后,通过 **path()函数**,传入邻接矩阵及交互信息,将封装各种**图操作算法**的类实体化,做出对应的相应。

代码思路的设计思路框架可视化如图 1 所示。

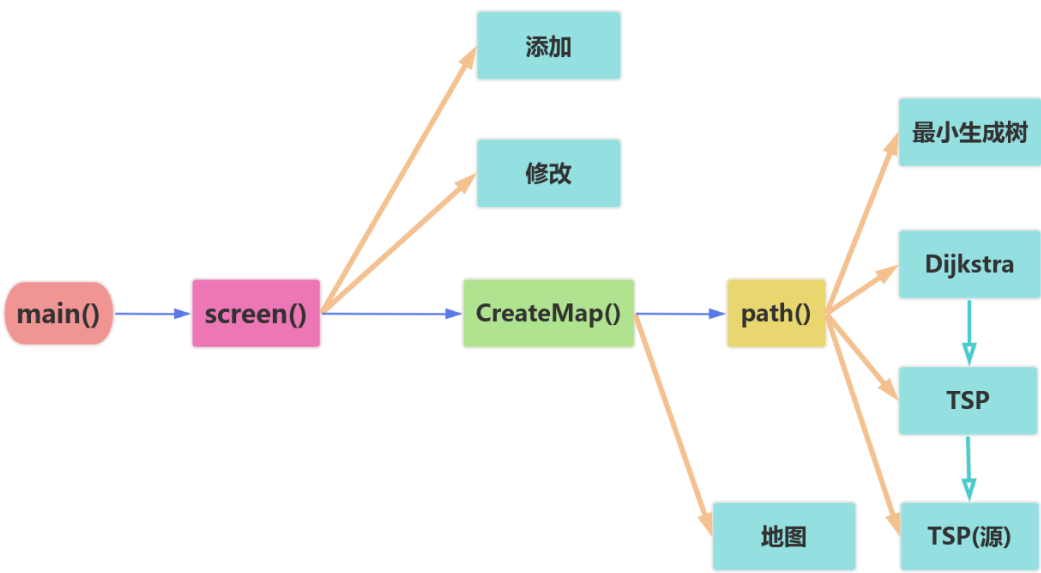


图 1 思路框架

三、数据格式：

本项目使用 TXT 文本文件存储输入的地点及路径信息，具体表现为 Location_Information.txt、Path_Information.txt 2 个文件，分别存储校内地点及路径数据。

(一) 校内地点数据(Location_Information.txt)：

Location_Information.txt 文件包含了校内地点信息，每一行的数据分别表示为：地点名称(字符串)、地点 x 坐标(正整数)、地点 y 坐标(正整数)、地点介绍(字符串)。数据之间用空格隔开，如图 2 所示。

1	校门	0	0	入校之处
2	教楼	3	4	新知之处
3	食堂	6	8	吃饭之处
4	宿舍	5	12	睡觉之处
5	菜鸟	9	10	外卖之处
6	公园	7	8	散步之处
7	操场	17	9	运动之处
8	超市	4	18	用品之处
9	书店	9	14	购书之处
10	厕所	4	16	方便之处
11	旗杆	12	15	升旗之处
12	工训	16	7	训练之处
13	体馆	8	19	室内之处

图 2 校内地点信息

(二) 校内道路数据(Path_Information.txt)：

Path_Information.txt 文件包含了校内地点信息，每一行的数据分别表示为：地点 1(字符串)名称、地点 2(字符串)名称。数据之间用空格隔开，表示两地存在路径，如图 3 所示。

1	校门	教楼
2	食堂	宿舍
3	教楼	食堂
4	校门	食堂
5	校门	宿舍
6	教楼	宿舍
7	菜鸟	校门
8	公园	校门
9	操场	公园
10	旗杆	食堂
11	超市	操场
12	书店	超市
13	工训	宿舍
14	书店	校门
15	厕所	书店
16	体馆	公园
17	旗杆	厕所
18	工训	旗杆
19	体馆	工训

图 3 校内路径信息

四、数据结构：

本项目通过 Createmap()来读取 TXT 文件中读取的校园地点、路径信息，构建邻接矩阵的存储结构，并通过 path()函数实体化 Graph 和 Vertex 类，其中顶点对应地点，权值表示距离(若无穷大，表示无通路)。

（一）数据结构定义：

本项目中存储结构的 c 语言定义为：

```
C
struct Graph{
    int vexnum; // 顶点数量
    int adjMatrix[MVNum][MVNum]; // 邻接矩阵
    bool visi[MVNum]; // 访问标记
};
```

由于图算法基于邻接矩阵操作的高抽象性，独立于实际的地点信息，因此在这里新构 Vertex 来存储交互性信息，为 TSP 问题回溯路径、及交互性地点信息展示提供便利。

```
C
struct Vertex {
    char name[MAX_NAME_LENGTH]; //地点信息
    int ID; //地点标记
};
```

本项目通过 C++语言编写，因此对上述 C 语言结构体进行封装成类，具体如下：

```
C++

class Graph {
private:
    int vexnum, adjMatrix[MVNum][MVNum]; // 顶点数量 邻接矩阵
    bool visi[MVNum];
public:
    Graph(int vexnum) : vexnum(vexnum) { . . . } // 类初始化
    void addEdge(int src, int dest, int weight) { . . . } // 添加边
    // Dijkstra 算法实现
    void Dijkstra(int src, int* prev, int* dist) { . . . }
    int minDistance(int* dist, bool* visited) { . . . } // 寻找最近点
    bool allVisited(int* vis) { . . . } // 检查所有顶点是否都被访问过
    void printShortestPathsFromSource(int begin, int* prev, int* dist,
    string name[]) { . . . } // 输出最短路径
    void printShortestPaths(int source, int destination, int Pr[], int
    size, string name[]) { . . . }
    bool DFS(int v) { . . . } // DFS__连通判断
    // 最小生成树
    void mintree(string name[]) { . . . }
    // 输出推荐路径 1
    void print_recommended_path_1(Vertex v, string na[]) { . . . }
    // 输出推荐路径 2
    void print_recommended_path_2(Vertex v, string na[]) { . . . }
};
```

```
C++

class Vertex {
public:
    string name; int ID; // 地点信息
```

```
Vertex(string name, int ID) : name(name), ID(ID) {} // 地点标记
};
```

（二）数据结构实体化：

本项目首先分别从两个 TXT 文件中读取数据，构建二维数组，并 path() 函数实体化 Graph 类及 Vertex 类，其中 Graph 类中的邻接矩阵的顶点对应地点，权值表示距离(若无穷大，表示无通路)。

具体实现核心代码如下：

C++

```
void CreateMap(int choose)

    ifstream inputFile("Location_Information.txt"); // 读取地点信息
    while (getline(inputFile, line)) {
        istringstream iss(line);
        iss >> name[sum] >> x[sum] >> y[sum] >> info[sum]; sum++;
    }

    ifstream pathFile("Path_Information.txt"); // 读取路径信息
    string pathLine;
    while (getline(pathFile, pathLine)) {
        istringstream iss(pathLine);
        iss >> name1 >> name2;
        int hang = -1, lie = -1;
        for (int k = 0; k < sum; k++) {
            if (name[k] == name1) hang = k;
            if (name[k] == name2) lie = k;
        }
        if (hang != -1 && lie != -1) // 计算距离
            pa[hang][lie] = pa[lie][hang] = round(sqrt((.) * (.) + (.) * (.) ));
    } // 实体化 Graph 及 Vertex 类

    if (choose == 1) PrintMap(name, x, y, sum);
    else if (choose == 4) Path(pa, sum, 4, name);
    else if (choose == 5) Path(pa, sum, 5, name);
```

```
else if (choose == 6) Path(pa, sum, 6, name);  
else Path(pa, sum, 7, name);  
}
```

C++

```
void Path(int pa[MVNum][MVNum], int sum, int choose, string name[]) {  
    Graph graph(sum); //实体化 Graph 类  
    for (int i = 0; i < sum; i++)  
        for (int j = i + 1; j < sum; j++)  
            graph.addEdge(i, j, pa[i][j]);  
    vector<Vertex> vertices; //实体化 Vertex 类  
    for (int i = 0; i < sum; i++)  
        vertices.push_back(Vertex(name[i], i));  
}
```

五、编辑地点、道路信息：

本项目通过 `screen()` 函数，提供的用户操作的菜单和界面如图 4 所示。

并通过其中的 `switch()` 函数，编写相应代码，响应各种操作，其中情景化假定此为人工智能辅助系统，帮助我们进行校园规划。

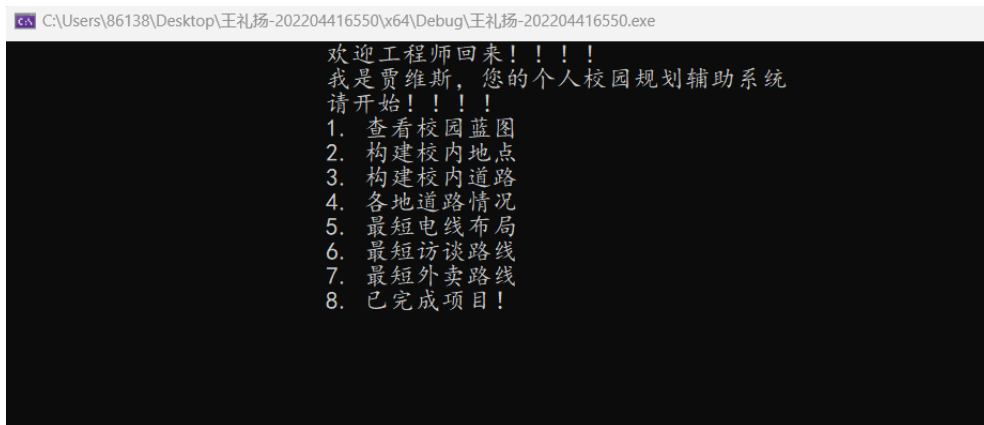


图 4 交互界面

由上图可知，可执行如下功能操作：

输入 1，查看整个校园的地图；

输入 2，进行校园地图的地点信息操作，包括添加、修改(即删除)；

输入 3，进行校园地图的路径信息操作，包括添加、修改(即删除)；

输入 4，可查看从某个起点出发，到其他所有地点的从小到大排序的距离及路径；

输入 5，可看到如何进行所有地点间电线连接，使得成本最低，本质为最小生成树问题；

输入 6，可重复地经过校园所有地点的最短路径及距离的所有起点方案；

输入 7，可重复地经过校园所有地点并返回起点的最短路径及距离的所有起点方案；

输入 8，可以安全退出此人工智能操作系统；

输入其他数字，则会要求重新输入。

(一) 校内地点信息操作：

1. 添加校内地点信息：

本项目直接用 C++标准库中的 ofstream 类，用于创建和操作文件，并通过 ios::app 指定追加模式，使得可以有记忆添加，核心代码如下。

```
C++
void Add_Location()
{
    // 添加校内地点
    outFile.open("Location_Information.txt", ios::app);
    if (outFile.is_open()) {
        outFile << name << " " << x << " " << y << " " << inf << endl;
        cout << "                                添加成功!!! " << endl;}
    else {
        cerr << "                                添加失败!!! " << endl;}
}
```

所得到的结果图如图 5-7 所示。

1	校门	0 0	入校之处
2	教楼	3 4	新知之处
3	食堂	6 8	吃饭之处
4	宿舍	5 12	睡觉之处
5	菜鸟	9 10	外卖之处
6	公园	7 8	散步之处
7	操场	17 9	运动之处
8	超市	4 18	用品之处
9	书店	9 14	购书之处
10	厕所	4 16	方便之处
11	旗杆	12 15	升旗之处
12	工训	16 7	训练之处
13	体馆	8 19	室内之处

图 5 地点信息(操作前)

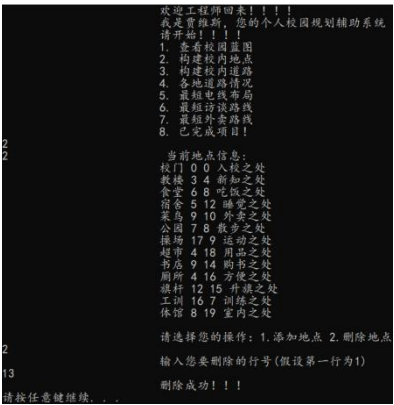


图 6 地点操作(删除)

1	校门	0 0	入校之处
2	教楼	3 4	新知之处
3	食堂	6 8	吃饭之处
4	宿舍	5 12	睡觉之处
5	菜鸟	9 10	外卖之处
6	公园	7 8	散步之处
7	操场	17 9	运动之处
8	超市	4 18	用品之处
9	书店	9 14	购书之处
10	厕所	4 16	方便之处
11	旗杆	12 15	升旗之处
12	工训	16 7	训练之处
13			

图 7 地点信息(操作后)

2. 删除校内地点信息:

项目的修改思路为: 修改某一部分, 即删除整一行数据。
具体实现流程为: 先通过 `getline()` 函数从输入文件流 `inputFile` 中逐行读取数据, 并至数组中, 再对数组进行删除操作, 最后重新写入文件中。

```
C++

void Delete_Location() {
    ifstream inputFile("Location_Information.txt");
    while (getline(inputFile, line)) { // 文件读取
        istringstream iss(line);
        iss >> name[sum] >> x[sum] >> y[sum] >> info[sum];

        sum++;
    }

    for (int i = 1 - 1; i < sum - 1; i++) { // 数组删除操作
        name[i] = name[i + 1]; x[i] = x[i + 1]; y[i] = y[i + 1];
        info[i] = info[i + 1];}

    ofstream outputFile("Location_Information.txt", ios::trunc);
    for (int i = 0; i < sum - 1; i++) { // 清空后 重新写入
        outputFile<<name[i]<<" "<<x[i]<<" "<<y[i]<<" "<<info[i]<<endl;}
}
```

所得到的结果图如 8-10 所示。

1	校门	0 0	入校之处
2	教楼	3 4	新知之处
3	食堂	6 8	吃饭之处
4	宿舍	5 12	睡觉之处
5	菜鸟	9 10	外卖之处
6	公园	7 8	散步之处
7	操场	17 9	运动之处
8	超市	4 18	用品之处
9	书店	9 14	购书之处
10	厕所	4 16	方便之处
11	旗杆	12 15	升旗之处
12	工训	16 7	训练之处
13			

图 8 地点信息(操作前)

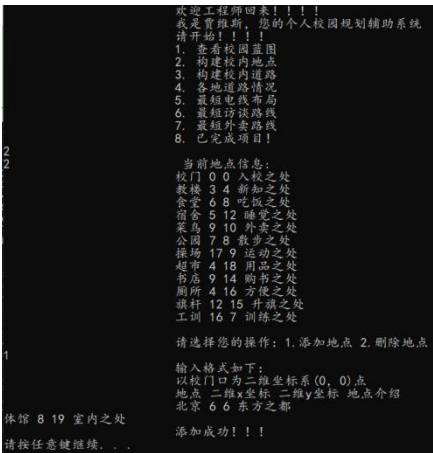


图 9 地点操作(添加)

1	校门	0 0	入校之处
2	教楼	3 4	新知之处
3	食堂	6 8	吃饭之处
4	宿舍	5 12	睡觉之处
5	菜鸟	9 10	外卖之处
6	公园	7 8	散步之处
7	操场	17 9	运动之处
8	超市	4 18	用品之处
9	书店	9 14	购书之处
10	厕所	4 16	方便之处
11	旗杆	12 15	升旗之处
12	工训	16 7	训练之处
13	体育馆	8 19	室内之处
14			

图 10 地点信息(操作后)

(二) 校内路径信息操作：

1. 添加校内路径信息：

具体添加方式如前所示。

```
C++
void Add_Path()
{ // 添加校内路径
    outFile.open("Path_Information.txt", ios::app);
    if (outFile.is_open()) {
        outFile << name1 << " " << name2 << endl;
        cout << "                                添加成功!!! " << endl;}
    else {
        cerr << "                                不能打开文件!!! " << endl;}
}
```

所得到的结果图如图 11-13 所示。

1	校门	教楼
2	食堂	宿舍
3	教楼	食堂
4	校门	食堂
5	校门	宿舍
6	教楼	宿舍
7	菜鸟	校门
8	公园	校门
9	操场	公园
10	旗杆	食堂
11	超市	操场
12	书店	超市
13	工训	宿舍
14	书店	校门
15	厕所	书店
16	体馆	公园
17	旗杆	厕所
18	工训	旗杆
19	体馆	工训

图 11 路径信息
(操作前)

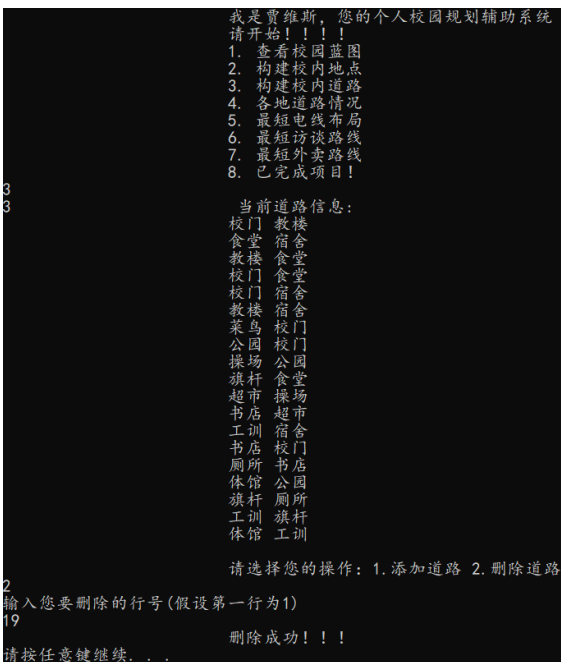


图 12 路径操作(删除)

1	校门	教楼
2	食堂	宿舍
3	教楼	食堂
4	校门	食堂
5	校门	宿舍
6	教楼	宿舍
7	菜鸟	校门
8	公园	校门
9	操场	公园
10	旗杆	食堂
11	超市	操场
12	书店	超市
13	工训	宿舍
14	书店	校门
15	厕所	书店
16	体馆	公园
17	旗杆	厕所
18	工训	旗杆
19		

图 13 路径信息
(操作后)

2. 删除校内路径信息:

如前面所述，项目的修改思路为：修改某一部分，即删除整一行数据。
具体实现流程为：先通过 `getline()` 函数从输入文件流 `inputFile` 中逐行读取数据，并至数组中，再对数组进行删除操作，最后重新写入文件中。

```
C++
void Delete_Path() {
    ifstream inputFile("Path_Information.txt");
    while (getline(inputFile, line)) { // 文件读取
        istringstream iss(line);
        iss >> pa1[sum] >> pa2[sum];sum++;}
    for (int i = 1 - 1; i < sum - 1; i++) { // 数组删除
        pa1[i] = pa1[i + 1]; pa2[i] = pa2[i + 1];}
    ofstream outputFile("Path_Information.txt", ios::trunc);
    for (int i = 0; i < sum - 1; i++) { // 清空后 重新写入
        outputFile << pa1[i] << " " << pa2[i] << endl;}
    cout << "                删除成功!!! " << endl;
}
```

1	校门	教楼
2	食堂	宿舍
3	教楼	食堂
4	校门	食堂
5	校门	宿舍
6	教楼	宿舍
7	菜鸟	校门
8	公园	校门
9	操场	公园
10	旗杆	食堂
11	超市	操场
12	书店	超市
13	工训	宿舍
14	书店	校门
15	厕所	书店
16	体馆	公园
17	旗杆	厕所
18	工训	旗杆
19		

图 14 路径信息
(操作前)

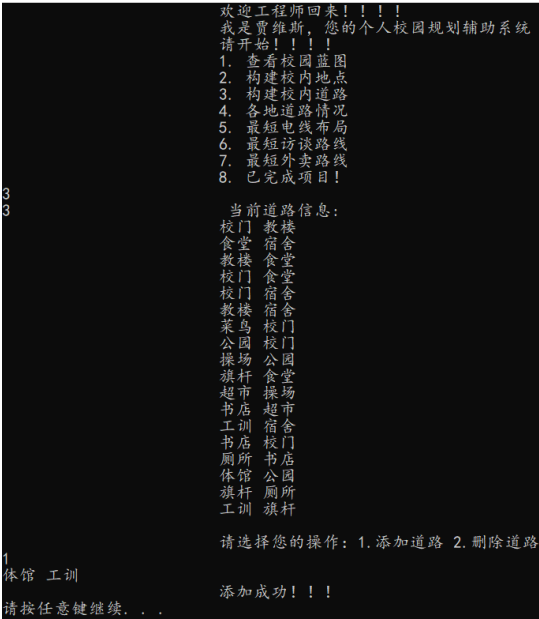


图 15 路径操作(添加)

1	校门	教楼
2	食堂	宿舍
3	教楼	食堂
4	校门	食堂
5	校门	宿舍
6	教楼	宿舍
7	菜鸟	校门
8	公园	校门
9	操场	公园
10	旗杆	食堂
11	超市	操场
12	书店	超市
13	工训	宿舍
14	书店	校门
15	厕所	书店
16	体馆	公园
17	旗杆	厕所
18	工训	旗杆
19	体馆	工训
20		

图 16 路径信息
(操作后)

六、打印校园地图：

本项目通过 CreateMap()函数，从两个 TXT 文件中读取数据构建邻接矩阵，并通过 PrintMap()函数，打印用户交互地图信息，如图 17 所示，直观明了。

同时打印每个地点的相关坐标信息及解释，及连通地点的道路信息，如图 18 所示。

核心代码如下所示：

```
C++

void PrintMap(string name[MVNum], int x[MVNum], int y[MVNum], int sum)
{
    // 交互性地图信息
    for (int i = 0; i <= max_y + 1; ++i) {
        if (i < 10) cout << " " << i << " ";
        else cout << " " << i << " "; }
    for (int i = 0; i <= max_x + 1; ++i) {
        cout << " " << i << " ";
        for (int j = 0; j <= max_y + 1; ++j) {
            cout << loc[i][j];}
        cout << endl;
    }
    // 地点信息
    Precent_Location();
    // 路径信息
    Precent_Path();
}
```

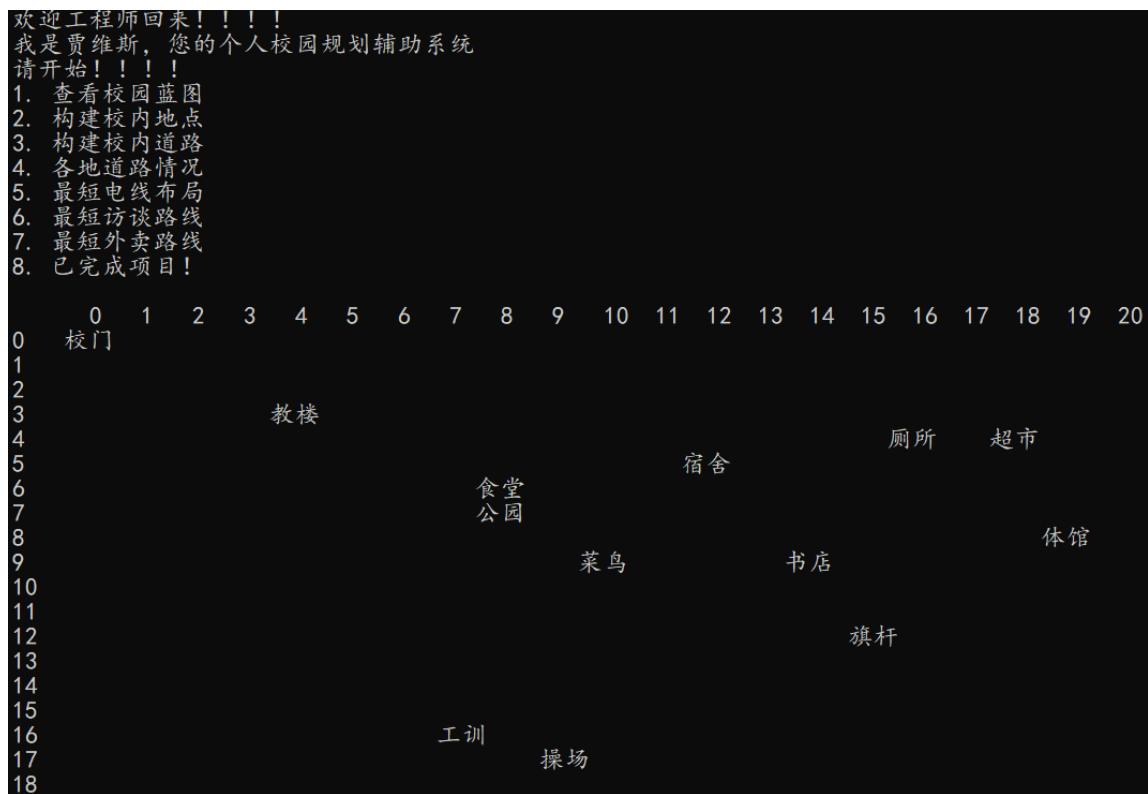


图 17 校园地图打印

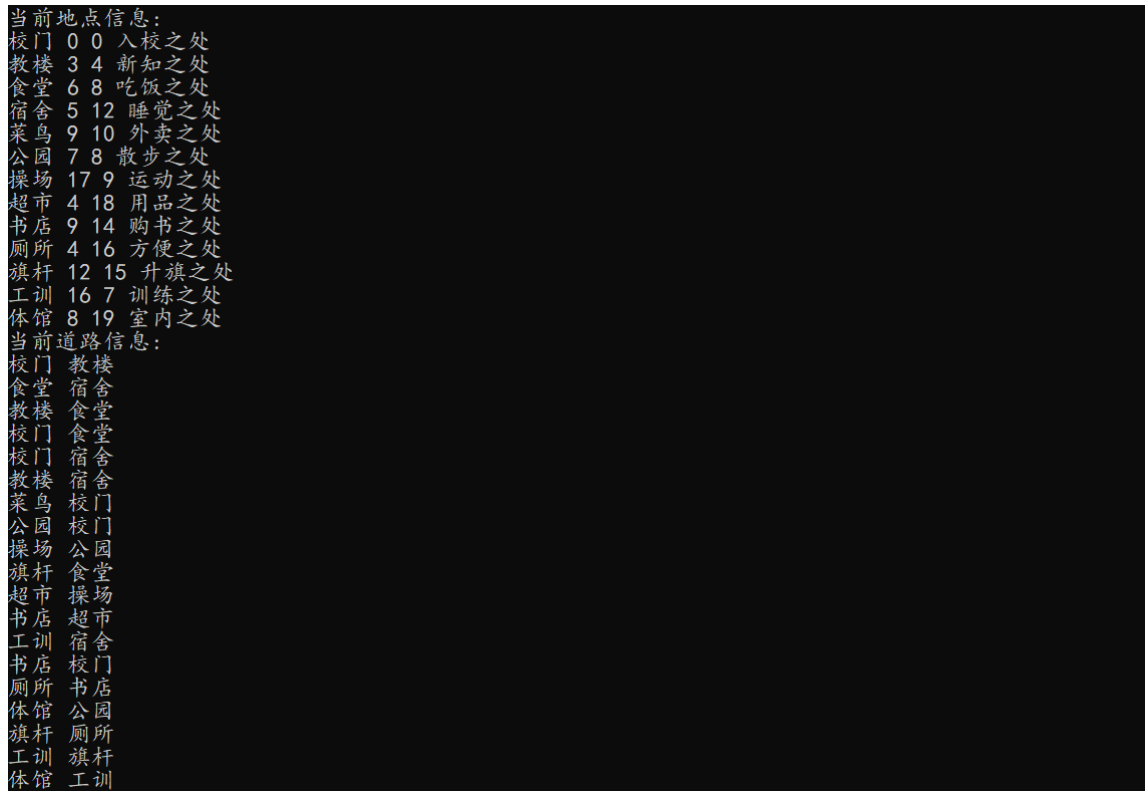


图 18 校园地图打印(续)

七、铺设电路规划：

铺设最短路径线路，本质就是最小生成树问题，而其有两种常见的算法，考虑到地点的有限性，本项目选用 Prim 算法(适用于点稀疏问题)解决此问题。

最小生成树问题，是求解一个无向连通图中的一棵包含所有顶点的树，且树的边的权值之和最小的方案。

（一）算法介绍：

Prim 算法是一种用于求解最小生成树（Minimum Spanning Tree, MST）的贪心算法。算法的基本思想是从一个任意的顶点开始，逐步扩展生成树，每次选择与当前生成树相邻的权值最小的边所连接的顶点，直到所有的顶点都被加入生成树为止。

Prim 算法，有三个辅助数组，介绍如下：

`lowcost[*]`

用于存放当前节点到最小生成树的最短距离

利用记忆性特点，即广度优先搜索算法思想，规避贪心局限，如图 21 所示

`closest[*]`

存放当前节点在最小生成树中的邻接节点

用于输出路径信息

`s[*]`

标记节点是否已加入最小生成树

用于确保连通性

本算法流程核心为：

首先，进行连通性检验，因为非连通图不存在最小生成树。

其次，通过 `lowcost[*]` 记忆回溯能力改进贪心决策，寻找局部最优解。

然后，更新三个辅助数组，并进行循环，直至所有顶点全遍历。

最后，输出路径情况及最短距离。

（二）算法流程：

采用伪代码形式描述算法流程，既避免自然语言描述的冗余、二义性，又规避流程图的非严密性，又能通过通俗易懂的方式，展示核心代码的通俗表现

形式及注释功能,使读者快速准确理解算法流程,有很强的表达能力及抽象能力,又能快速基于此编写、调试代码。伪代码如下:

Algorithm 1: 最小生成树

Input: 交互文本

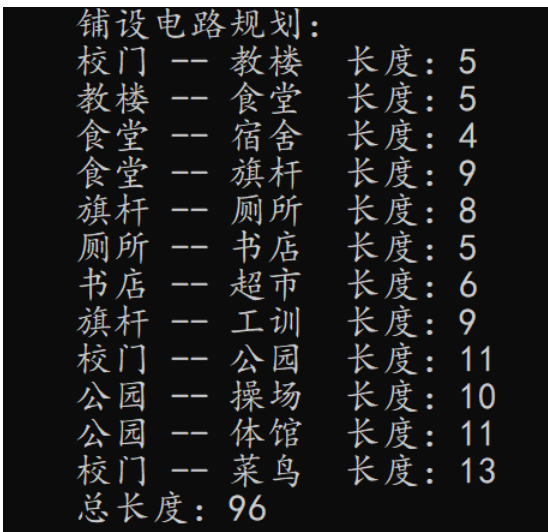
Initialize: lowcost[*] = adjMatrix[0][*], closest[*] = 0, vs[*] = false

```
1: DFS(0); //可行性检验_判断是否为连通图
2: for i: = 0 To vexnum //循环确保连通性
3:     update: minIndex = -1, minDistance = INF
4:     for j: = 0 To vexnum //贪心算法_寻找最小距离节点
5:         if (!s[j] && lowcost[j] < minDistance)
6:             update: minIndex = j, minDistance = lowcost[j]
7:         update: totalLength += minDistance, s[minIndex] = true
8:     for j: = 0 To vexnum //更新最小距离数组 及 节点的邻接节点
9:         if (!s[j] && adjMatrix[minIndex][j] < lowcost[j])
10:            update: lowcost[j] = adjMatrix[minIndex][j], closest[j] = minIndex
```

Output: 最短输电线路

(三) 算法运行结果:

通过在用户交互界面输入 5, 进入最短电线布局, 所得结果如下图 19 所示。



```
铺设电路规划:
校门 -- 教楼    长度: 5
教楼 -- 食堂    长度: 5
食堂 -- 宿舍    长度: 4
食堂 -- 旗杆    长度: 9
旗杆 -- 厕所    长度: 8
厕所 -- 书店    长度: 5
书店 -- 超市    长度: 6
旗杆 -- 工训    长度: 9
校门 -- 公园    长度: 11
公园 -- 操场    长度: 10
公园 -- 体馆    长度: 11
校门 -- 菜鸟    长度: 13
总长度: 96
```

图 19 最小生成树-电路规划

上图不仅展示最短生成树及最短电路铺设的距离,还提供具体连通的道路信息及两两连通的距离,更细节的描述更便于建设规划。

八、查询校园地点信息:

本项目设置的查询校园地点信息功能,首先通过如图 17、图 18 所示,展示校园地图整体概况,并提供图 20 所示,即起点的交互选择界面,由此可输出某一校园地点为起点,按照距离从近到远的顺序显示到其余地点的最短距离及路径。

本功能的代码实现,本质为单源点最短路径 (Single Source Shortest Path, SSSP) 问题,本文采用时间复杂度为 $O(n^2)$ 的 Dijkstra 算法。

(一) 算法介绍:

Dijkstra 算法本质思想为贪心算法思想+bfs 思想,通过 bfd 的记忆回溯数组规避贪心决策的盲目性。

Dijkstra 算法的基本思想是采用贪心策略,逐步扩展最短路径集合,每次选择当前最短路径集合外的一个顶点,并通过这个顶点更新到其他顶点的最短路径。

Dijkstra 算法,有三个辅助数组,介绍如下:

dist[*]

用于存放起点到可到达顶点的最短距离
利用记忆性特点,即广度优先搜索算法思想,规避贪心局限,如图 21 所示

prev[*]

存放当前节点之前的邻接节点
用于回溯输出路径信息

visited[*]

标记节点是否已经访问

算法核心流程如下:

首先,初始化源点至其他顶点的距离为无穷大,至自身距离为 0,并标记所有顶点为未访问。

其次,选取未访问源点的下一目的地。

再次,通过记忆回溯能力改进贪心决策,寻找局部最优解,即选择源点所有邻居顶点或末端点关联顶点中的最短距离,更新三个数组,最终求得源点到下一目的地的最短路径及距离。

然后，循环其次部分，直至所有顶点全遍历。
最后，输出单源起点至所有其他顶点的路径情况及最短距离。

(二) 算法流程:

为了更优的算法流程展现，依旧采用伪代码形式。

Algorithm 2: Dijkstra

Input: 交互文本

Initialize: visited[MVNum] = { false }, prev[*] = -1, dist[i] = INF

1:

update: dist[src] = 0 //辅助数组初始化

2:

for i: = 0 To vexnum - 1 //循环访问所有顶点

2:

update: u = minDistance(dist, visited), visited[u] = true //找最近顶点

3:

for j: = 0 To vexnum //找最近顶点

4:

if (!visited[v]&& dist[u] + adjMatrix[u][v] < dist[v])

5:

update: dist[v] = dist[u] + adjMatrix[u][v], prev[v] = u

6:

printShortestPathsFromSource(int begin, int* prev, int* dist, string name[])

Output: 某地到各地最短距离与路径(递增)

(三) 算法运行结果:

通过在用户交互界面输入 4，进入各地道路情况功能，并输入起点 4，即菜鸟地点，所得结果如下图 19 所示。

```
请输入您的位置:
0. 校门 1. 教学楼 2. 食堂 3. 宿舍 4. 菜鸟 5. 公园 6. 操场 7. 超市 8. 书店 9. 厕所 10. 旗杆 11. 工训 4
目的地: 校门, 路径长度: 13, 路径: 菜鸟 -> 校门
目的地: 教学楼, 路径长度: 18, 路径: 菜鸟 -> 校门 -> 教学楼
目的地: 食堂, 路径长度: 23, 路径: 菜鸟 -> 校门 -> 食堂
目的地: 公园, 路径长度: 24, 路径: 菜鸟 -> 校门 -> 公园
目的地: 宿舍, 路径长度: 26, 路径: 菜鸟 -> 校门 -> 宿舍
目的地: 书店, 路径长度: 30, 路径: 菜鸟 -> 校门 -> 书店
目的地: 旗杆, 路径长度: 32, 路径: 菜鸟 -> 校门 -> 食堂 -> 旗杆
目的地: 操场, 路径长度: 34, 路径: 菜鸟 -> 校门 -> 公园 -> 操场
目的地: 厕所, 路径长度: 35, 路径: 菜鸟 -> 校门 -> 书店 -> 厕所
目的地: 超市, 路径长度: 36, 路径: 菜鸟 -> 校门 -> 书店 -> 超市
目的地: 工训, 路径长度: 38, 路径: 菜鸟 -> 校门 -> 宿舍 -> 工训
```

图 20 Dijkstra-最短路径

此界面输出由源点菜鸟，到所有其他地点的最短路径及距离，并按照从小到大的距离展示。

九、遍历校园：

寻找经过所有地点的最短路径问题，本质是旅行商问题(Traveling Salesman Problem, TSP)，本项目主要解决其中的可重复+是否回源类型的问题，具体介绍为：

旅行商问题，是一类经典的组合优化问题，目标是找到一条路径，使得旅行商可以恰好访问每个给定城市一次，并最终回到起始城市，同时总行程最短。

具体来说，给定一个包含 n 个城市的图，每两个城市之间有一条边，对应着连接它们的距离。旅行商问题要求找到一条路径，使得经过每个城市恰好一次，并且最终回到起始城市，同时使得路径的总长度最小。

旅行商问题是一个 NP 困难问题，意味着没有已知的多项式时间解决方案。因此，通常采用启发式算法、近似算法或者精确算法来解决，而由于本项目涉及的背景为校园地点路径信息规划，由于其量级有限性，因此可采用 Dijkstra 的功能及思想，在有些时间内，准确求解本问题，可重复+是否回源类型的 TSP 问题的问题分析及算法的可行性证明如下：

（一）TSP 问题分析及可行性证明：

1. 贪婪思想应用可行性证明：

① 选择双重性(起点 0，终点 2)：

以校园为应用场景的单源点 TSP 问题，下一路径选择决策在本质上可最终归结为两类，如下两种情况(可视化于图 21)：

- 0—>1—>2 以末端(新起点)到目的地
- 0—>2 以起点到达目的地

通过 `dist[*]` 记忆回溯功能改进贪心决策，即选择源点所有邻居顶点或末端点关联顶点中的最短距离，作为源点到下一目的地的最短路径及距离，确保局部最优解。

② 选择无记忆性:

以校园为应用场景的单源点 TSP 问题, 若起点 与 目的地无直接路径, 必须基于前面的局部最优解, 如下所示路径(可视化于图 22):

0—>1—> 4

由此可得, 下一步的最优解建立在上一步的局部最优解上, 且此局部最优解为确定解, 即全局最优解的最优子结构。

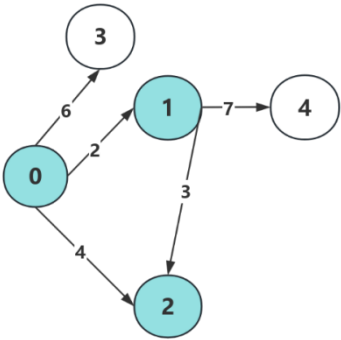


图 23 TSP 决策情况

由(1) (2)可知, 此类型的 TSP 问题, 通过可应用贪心算法得到局部最优解, 由于与问题的独特背景约束, 即局部最优解是全局最优解, 因此可应用贪心策略求解准确解。

2. Dijkstra 应用可行性证明:

可重复+是否回源 TSP 问题, 关键是可重复问题及是否回源两个问题的解决, 解决方案探讨如下:

① 可重复:

利用 Dijkstra 算法, 其中的 dist[*]记忆回溯特性改进贪心策略, 可找到尾端与所有地点最短路径及距离。

当尾端点邻近端点全访问时, 通过 pre[*]回溯记录及容器数据结构的寻头、寻尾、段插入操作, 实现未查找节点的选择, 即可解决可重复顶点问题。

若 1—>2—>3

则下一步为 3→2→1→4

核心代码如下：

C++

```
vector<int> temp;  
for (int j = next_vex; j != path_list.back(); j = prev[j])  
    temp.insert(temp.begin(), j);  
path_list.insert(path_list.end(), temp.begin(), temp.end());
```

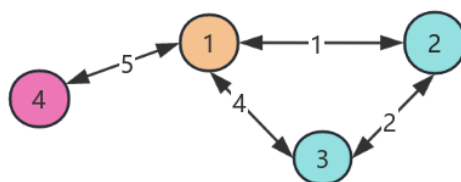


图 24 可重复情况

② 回到源点：

由贪心无记忆性可知，每次抉择及确定的全局最优子结构，因此回到源点，基于可重复不回源点 TSP 算法下的最后端点至起点最短路径情况。

而起点到端点最短路径 = 端点到起点最短路径。

因此再次利用 Dijkstra 算法求得起点到末端点的路径及距离即可解决。

（二）TSP 问题求解：

1. 思路核心流程：

首先，进行连通性检验，因为非连通图不存在 TSP 解。

其次，利用 Dijkstra 记录末端点至所有未访问顶点的最短路径情况及距离

再次，利用改进的贪心决策，寻找局部最优解，即最优子结构

然后，更新路径信息，然后至其次循环，直至所有顶点访问完

最后，若回源点，则再次用 Dijkstra 操作源点与末端点的路径信息，最终输出最短路径信息及距离

2. 算法流程及结果:

① 起点出发经过校园所有地点（地点可以重复）且距离最短的路线:

Algorithm 3: TSP 问题(可重复)

Input: 起点位置

Initialize: totalLength = 0, vis[MVNum] = { 0 }, vis[v.ID] = 1.

```
1: DFS(v.ID); // 判断是否为空
2: while (!allVisited(vis)) do
2:   update: prev[*] = { 0 }, dist[*] = { 0 }, next_vex = -1
3:   while !vis[j] do
4:     next_vex = j;
5:   Dijkstra(path_list.back(), prev, dist); // 记录末端点至其他顶点情况
6:   for j = 0 To vexnum // 寻找局部最优解
7:     if (!vis[j])
8:       next_vex = (next_vex == -1 || dist[j] < dist[next_vex]) ? j : next_vex;
9:   while j != path_list.back() do // 更新路径信息
10:    temp.insert(temp.begin(), j)
11:    j = prev[j]
12:   path_list.insert(path_list.end(), temp.begin(), temp.end());
13:   Update: totalLength += dist[next_vex], vis[next_vex] = 1
14: end while
```

Output: 推荐路径(可重复), 总长度

```
推荐路径起始点 "校门": 校门 → 教学楼 → 食堂 → 宿舍 → 工训 → 旗杆 → 厕所 → 书店 → 超市 → 操场 → 公园 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 126
推荐路径起始点 "教学楼": 教学楼 → 校门 → 食堂 → 宿舍 → 工训 → 旗杆 → 厕所 → 书店 → 超市 → 操场 → 公园 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 131
推荐路径起始点 "食堂": 食堂 → 宿舍 → 教学楼 → 校门 → 公园 → 操场 → 超市 → 书店 → 厕所 → 旗杆 → 工训 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 131
推荐路径起始点 "宿舍": 宿舍 → 食堂 → 教学楼 → 校门 → 公园 → 操场 → 超市 → 书店 → 厕所 → 旗杆 → 工训 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 128
推荐路径起始点 "菜鸟": 菜鸟 → 校门 → 教学楼 → 食堂 → 宿舍 → 工训 → 旗杆 → 厕所 → 书店 → 超市 → 操场 → 公园 → 体育馆
总距离: 104
推荐路径起始点 "公园": 公园 → 操场 → 超市 → 书店 → 厕所 → 旗杆 → 食堂 → 宿舍 → 工训 → 旗杆 → 厕所 → 书店 → 超市 → 书店 → 校门 → 公园 → 体育馆
总距离: 133
推荐路径起始点 "操场": 操场 → 公园 → 校门 → 教学楼 → 食堂 → 宿舍 → 工训 → 旗杆 → 厕所 → 书店 → 超市 → 书店 → 校门 → 公园 → 体育馆
总距离: 146
推荐路径起始点 "超市": 超市 → 书店 → 厕所 → 旗杆 → 食堂 → 宿舍 → 教学楼 → 校门 → 公园 → 操场 → 公园 → 体育馆 → 工训 → 宿舍 → 校门 → 菜鸟
总距离: 139
推荐路径起始点 "书店": 书店 → 厕所 → 旗杆 → 食堂 → 宿舍 → 教学楼 → 校门 → 公园 → 操场 → 超市 → 书店 → 厕所 → 旗杆 → 工训 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 153
推荐路径起始点 "厕所": 厕所 → 书店 → 超市 → 操场 → 公园 → 校门 → 教学楼 → 食堂 → 宿舍 → 工训 → 旗杆 → 工训 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 141
推荐路径起始点 "旗杆": 旗杆 → 厕所 → 书店 → 超市 → 操场 → 公园 → 校门 → 教学楼 → 食堂 → 宿舍 → 工训 → 体育馆 → 公园 → 校门 → 菜鸟
总距离: 131
推荐路径起始点 "工训": 工训 → 旗杆 → 厕所 → 书店 → 超市 → 操场 → 公园 → 校门 → 教学楼 → 食堂 → 宿舍 → 校门 → 菜鸟 → 校门 → 公园 → 体育馆
总距离: 140
推荐路径起始点 "体育馆": 体育馆 → 公园 → 操场 → 超市 → 书店 → 厕所 → 旗杆 → 食堂 → 宿舍 → 教学楼 → 校门 → 菜鸟 → 校门 → 宿舍 → 工训
总距离: 133
```

图 25 TSP 问题(可重复)-运行结果

② 起点出发经过校园所有地点（地点可以重复）并回到源点的最短路线：

Algorithm 4: TSP 问题(可重复回起点)

Input: 起点位置

Initialize: totalLength = 0, vis[MVNum] = { 0 }, vis[v.ID] = 1.

```
1: DFS(v.ID); // 判断是否为空
2: while (!allVisited(vis)) do
2:   update: prev[*] = { 0 }, dist[*] = { 0 }, next_vex = -1
3:   while !vis[j] do
4:     next_vex = j;
5:     Dijkstra(path_list.back(), prev, dist); // 记录末端点至其他顶点情况
6:     for j: = 0 To vexnum // 寻找局部最优解
7:       if (!vis[j])
8:         next_vex = (next_vex == -1 || dist[j] < dist[next_vex]) ? j : next_vex;
9:       while j != path_list.back() do // 更新路径信息
10:        temp.insert(temp.begin(), j)
11:        j = prev[j]
12:      path_list.insert(path_list.end(), temp.begin(), temp.end());
13:      update: totalLength += dist[next_vex], vis[next_vex] = 1
14:    end while
15:  update: prev[*] = { 0 }, dist[*] = { 0 }
16:  while j != path_list.back() do // 返回源点
17:    temp.insert(temp.begin(), j)
18:    j = prev[j]
19:  path_list.insert(path_list.end(), temp.begin(), temp.end());
20:  update: totalLength += dist[v.ID], vis[next_vex] = 1
```

Output: 推荐路径(可重复), 总长度

```

推荐路径开始与结束“校门”：校门->教学楼->食堂->宿舍->工训->旗杆->厕所->书店->超市->操场->公园->体育馆->公园->校门->菜鸟->校门
总距离：139
推荐路径开始与结束“教学楼”：教学楼->校门->食堂->宿舍->工训->旗杆->厕所->书店->超市->操场->公园->体育馆->公园->校门->菜鸟->校门->教
楼
总距离：149
推荐路径开始与结束“食堂”：食堂->宿舍->教学楼->校门->公园->操场->超市->书店->厕所->旗杆->工训->体育馆->公园->校门->菜鸟->校门->食
堂
总距离：154
推荐路径开始与结束“宿舍”：宿舍->食堂->教学楼->校门->公园->操场->超市->书店->厕所->旗杆->工训->体育馆->公园->校门->菜鸟->校门->宿
舍
总距离：154
推荐路径开始与结束“菜鸟”：菜鸟->校门->教学楼->食堂->宿舍->工训->旗杆->厕所->书店->超市->操场->公园->体育馆->公园->校门->菜鸟
总距离：139
推荐路径开始与结束“公园”：公园->操场->超市->书店->厕所->旗杆->食堂->宿舍->教学楼->校门->菜鸟->校门->公园->体育馆->工训->体育馆->公
园
总距离：158
推荐路径开始与结束“操场”：操场->公园->校门->教学楼->食堂->宿舍->工训->旗杆->厕所->书店->超市->书店->校门->菜鸟->校门->公园->体
育馆->公园->操场
总距离：167
推荐路径开始与结束“超市”：超市->书店->厕所->旗杆->食堂->宿舍->教学楼->校门->公园->操场->公园->体育馆->工训->宿舍->校门->菜鸟->校
门->书店->超市
总距离：175
推荐路径开始与结束“书店”：书店->厕所->旗杆->食堂->宿舍->教学楼->校门->公园->操场->超市->书店->厕所->旗杆->工训->体育馆->公园->校
门->菜鸟->校门->书店
总距离：183
推荐路径开始与结束“厕所”：厕所->书店->超市->操场->公园->校门->教学楼->食堂->宿舍->工训->旗杆->工训->体育馆->公园->校门->菜鸟->校
门->书店->厕所
总距离：176
推荐路径开始与结束“旗杆”：旗杆->厕所->书店->超市->操场->公园->校门->教学楼->食堂->宿舍->工训->体育馆->公园->校门->菜鸟->校门->食
堂->旗杆
总距离：163
推荐路径开始与结束“工训”：工训->旗杆->厕所->书店->超市->操场->公园->校门->教学楼->食堂->宿舍->校门->菜鸟->校门->公园->体育馆->工
训
总距离：154
推荐路径开始与结束“体育馆”：体育馆->公园->操场->超市->书店->厕所->旗杆->食堂->宿舍->教学楼->校门->菜鸟->校门->宿舍->工训->体
育馆
总距离：147

```

图 26 TSP 问题(可重复回起点)-运行结果

十、设计总结：

本次数据结构课程设计使我收获颇多，我将从以下四个视角进行整个过程的总结：

（一） 方法论视角：

在完成整个项目过程中，第一次遇到的困难，必然会很多试错，很难一开始就是正确的准确的道路，但是一旦解决这个困难，再回溯过去的种种经历与弯路，可以总结抽象出一套方法论模板，用于下次解决相同及类似的问题。

注 1：以下解决方案为过程，具体代码角度的解决方案以上或代码已给出。

注 2：以下解决方案为目前抽象的最优路径，总结前的过程中必然很多试错。

问题一(微观)：如何正确学习并解决一个新问题？

具体体现：

TSP 问题如何实现可重复？如何实现起点化？

解决方案：

1. 思想上意识到目前所遇到的 90%问题别人都可能遇到并解决过，一定会有正确的解决方案；
2. 通过 Chatgpt 定义问题类型，为 TSP 问题；
3. 通过博客初步了解 TSP 问题，为 NP 问题，有重复与否、回起点与否等问题类型；
4. 通过 github 等源码网站、公众号等博客、及学术论文等从不同层次及角度深刻理解问题及解决方案，并学习补充相应基础理论知识；

5. 通过博客、开源代码网站 [GitHub/paperswithcode](#)、论文等寻找代码 demo;
6. 基于 demo 及问题的分析, 选择合适的数据类型及结构, 设计合适的算法, 并通过 Chatgpt/博客等辅助解决各种调试问题;
7. 最终完成此项目。

问题二（宏观）：如何一步步完成整个项目？

解决方案：

1. 对整个问题进行宏观了解, 构建问题框架及阶段性目标
2. 预估完成每阶段完成的截止时间, 预留变化缓冲时间
3. 每个目标要集中时间与精力完成, 不要零散化, 以结果为导向, 不执拗于时限
4. 及时根据变化更新迭代阶段目标(循环 1), 必要时加班加点完成

（二）知识视角：

1. 从算法框架角度, 学习搜索算法、贪心算法、动态规划递进优化的算法框架, 知晓各种算法框架的优缺点及适用场景。
2. 梳理最短距离常见问题间联系:
单源路径问题:
 BFS 适用无权图(权值为 1);
 Dijkstra 适用带权图_贪心思想。
顶点间的最短路径问题:
 Floyd(无权图、带权图)_动态规划思想。
3. 梳理单源路径问题与最小生成树间相似及区别:
 相似: 均运用 `dist[*]`回溯改善贪心局限, 运用 `pre[*]`回溯路径, 运用 `vis[*]`记录访问信息。
 区别: 单源问题的贪心决策为源点与末端点邻接点距离选择, 而最小生成树为最优子集合与末端点的邻接点选择, 简单说一个为点与点, 一个为集合与点。
4. 分析并证明, 以 Dijkstra 为核心的小数量 TSP 问题及最小生成树问题应用贪心算法策略的可行性及特殊性, 具体证明过程如上所示。
5. 知晓 TSP 为代表的作为 7 大数学猜想的 NP 问题的难点, 并知晓更高量级的 NP 问题需要用粒子群、遗传算法、蚁群、模拟退火等启发算法进行近似求解。
6. 了解到容器相比栈或队列在批量段数据处理方面的便捷性。
7. 学会用头文件及源文件进行多文件项目管理。
8. 学会文本文件的插入、删除等操作。
9. 学会解决栈溢出问题:
 在 “项目 ---- 属性 ---- 配置属性 ---- C/C++ ---- 代码生成 ---- 基本运行时检查:” 设置为默认值;
 在 “项目 ---- 属性 ---- 链接器 ---- 系统 ---- 堆栈保留大小” 设置为 10000000000;
 将该局部变量设置为全局变量;
 优化算法及数据结构, 以减少代码的递归调用深度。

10. 意识到代码注释清晰的重要性，因为长期项目必然会出现遗忘。

(三) 技能视角：

1. 学会项目进程管理：

如何进行项目阶段性拆解？如何进行项目阶段目标更新迭代？如何规划并集中精力完成阶段性目标？如何评估阶段性目标成果？如何反馈阶段性目标最终能实现最终目标？通过以上问题，在整个项目过程中，确保有目标、有反馈、有成果。

2. 提升学习能力：

在此次课程设计中，深入学习了数据结构未讲的扩展知识部分，比如代码层面的最短路径算法、TSP 问题等，通过博客、论文、代码 demo 等不同深度层次及层次，一步步深刻理解问题，并解决问题的过程中，使得我更有能力与信心解决未来遇到的问题。

3. 提升信息检索能力：

在遇到知识范围以外的问题时，寻求求助是必然。因此，此次课程设计使得我扩展平常知识理解及信息搜集渠道 至 GitHub、论文、公众号等以前不知道的地方。

4. 学会修改、调试已有 demo，解决特定问题的能力：

意识到在项目中，很少有情况是从零开始的，更多的时候我们需要基于已有的代码进行修改和拓展，相当于站在巨人的肩膀上，因此学会修改和调试现有的代码是一个非常实用的技能。

(4) 素养视角：

1. 心理素质锻炼，一般在以下情况会出现负面情况：难点投入很多时间却始终解决不了、任务 deadline、畏难。

因此，总结如下解决方案：

首先，意识到你在这种情况下产生的负面情绪很正常；

其次，学会拆解问题，一步一步阶段性小成就，提升自信心；

再次，从情感支持体系上获得安慰，或者纯粹出去逛逛，放松心情；

然后，心情收拾好后，静下心来面对并解决难点；

最后，实在解决不了，学会交流、求助。

2. 逻辑思维能力，通过数据结构的选择与设计，算法问题的分析、理解、设计，锻炼了知识理解，及一步步逻辑推理分析问题的过程中，训练了严谨的习惯。

总之，此次数据结构课程设计，从多个层面提升了我的综合素质与能力。非常感谢自己能抓住此次机会，也感激自己能全身心投入，遇到问题不逃避，并将其做好，才能在其中获得如此多的成长，扩展了自己的能力边界，更好地面对未来新的未知挑战。