目录

1.项目背景

在图论中，一个有 n 个结点的连通图树是原图的极小连通子图，且包含原图中的所有 n 个结点，并且有保持图连通的最少的边。 最小生成树可以用kruskal（克鲁斯卡尔）算法prim（普里姆）算法求出。

在一给定的无向图G = (V, E) 中，(u, v) 代表连接顶点 u 与顶点 v 的边（即），而 w(u, v) 代表此边的权重，若存在 T 为 E 的子集（即）且为无循环图，使得

的 w(T) 最小，则此 T 为 G 的**最小生成树。**

$$\omega(t)=\sum_{(u,v)\in t}\omega(u,v)$$

最小生成树其实是**最小权重生成树**的简称

而最小生成树有许多重要的应用，比如城市之间铺设光缆，要使总共花费最小，并且任意两个之间都可以通信，这就需要使用最小生成树算法。本程序解决的问题即为光缆问题变形：

> 假设一个城市有n个小区，要实现n个小区之间的电网都能够相互接通，构造这个城市n个小区之间的电网，使总工程造价最低。请设计一个能够满足要求的造价方案。

2.需求分析

> 在每个小区之间都可以设置一条电网线路，都要付出相应的经济代价。n个小区之间最多可以有n（n-1）/2条线路，选择其中的n-1条使总的耗费最少。

将校区抽象为顶点，电线抽象为边，该问题即为求这n个顶点完全图的最小生成树

3.设计思路

使用prim算法来求解最小生成树

Prim算法说明

> **普里姆算法**（Prim算法），图论中的一种算法，可在加权连通图里搜索最小生成树。意即由此算法搜索到的边子集所构成的树中，不但包括了连通图里的所有顶点（Vertex (graph theory)），且其所有边的权值之和亦为最小。该算法于1930年由捷克数学家沃伊捷赫·亚尔尼克（Vojtěch Jarník）发现；并在1957年由美国计算机科学家罗伯特·普里姆（Robert C. Prim）独立发现；1959年，艾兹格·迪科斯彻再次发现了该算法。因此，在某些场合，普里姆算法又被称为DJP算法、亚尔尼克算法或普里姆 - 亚尔尼克算法

算法的描述为：

1. 输入：一个加权连通图，其中顶点集合为V，边集合为E；
2. 初始化：Vnew = {x}，其中x为集合V中的任一节点（起始点），Enew = {},为空；
3. 重复下列操作，直到Vnew = V：
    1. 在集合E中选取权值最小的边，其中u为集合Vnew中的元素，而v不在Vnew集合当中，并且v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；
    2. 将v加入集合Vnew中，将边加入集合Enew中；
4. 输出：使用集合Vnew和Enew来描述所得到的最小生成树

4.核心代码说明

数据结构

邻接顶点

```
//Adjacent vertex
struct CloseNode {
    char adj_vex;
    double low_cost;
};
```

`adj_vex`存储邻接顶点名称

`low_cost`存储Vnew中的顶点到该邻接顶点的最小权值

边

```
struct Arc {
    char node1;
    char node2;
    double value;
};
```

`node1` `node2`用于存储边的两个顶点

`value`存储边的权值

无向图

```
struct Graph {
    vector<char> vexs;
    vector<Arc> arcs;
    double adj_table[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
    int vex_num, arc_num;
};
```

`vexs`为图的顶点集合

`arcs`为图的边集合

`adj_table`为图的邻接矩阵

`vex_num`存储图的顶点数

`arc_num`存储图的边数

函数功能说明

构建无向图

在main函数中声明一个Graph的对象，首先输入顶点信息：

```
#main部分
cout << "请输入顶点个数:";
cin >> graph.vex_num;
cout << "请依次输入各顶点的名称：";
for (int i = 0; i < graph.vex_num; i++) {
  cin >> temp;
  graph.vexs.push_back(temp);
}
break;
```

再通过调用`createUDG`输入边信息，并构造邻接矩阵

```
#main部分
cout << "请输入边的总数：";
cin >> graph.arc_num;
createUDG(graph);
break;

```

```
#createUDG
//Create the undirected graph
void createUDG(Graph& G) {
  //Initialize the adjacent table
  for (int i = 0; i < G.vex_num; i++)
    for (int j = 0; j < G.vex_num; j++)
      G.adj_table[i][j] = MAX;
  //Enter arcs and read the info to arcs and adjacent table
  for (int i = 0; i < G.arc_num; i++) {
    Arc temp;
    int x, y;

    cout << "请输入两个顶点及边：";
    cin >> temp.node1 >> temp.node2 >> temp.value;
    G.arcs.push_back(temp);
    x = getLocation(G, temp.node1);
    y = getLocation(G, temp.node2);
    G.adj_table[x][y] = G.adj_table[y][x] = temp.value;
  }
}
```

生成最小生成树

获取邻接矩阵坐标

通过传入顶点名称返回其在顶点集合中的序号，用于确定

```
//Get the location of a vertex by name;
int getLocation(Graph& G, char name) {
  int location;
  for (int i = 0; i < G.vex_num; i++)
    if (G.vexs[i] == name) {
      location = i;
    }
  return location;
}
```

获取E中权值最小边的顶点

比较Vnew各节点到Enew节点权值最小边集合中的权值，找到最小的一条边，返回该边Enew中的顶点坐标，该操作为Prim算法搜索下一条边的核心算法

```
//Get the location of minimum adjacent vertex
int getMinAdjVex(Graph& G, CloseNode(&close_nodes)[MAX_VERTEX_NUM]) {
  int min = MAX,location;
  for (int i = 0; i < G.vex_num; i++) {
    if (close_nodes[i].low_cost != 0 && close_nodes[i].low_cost < min) {
      min = close_nodes[i].low_cost;
      location = i;
    }
  }
  return location;
}
```

生成最小生成树

Prim算法生成最小生成树，`close_node`邻接顶点集合，每一次接入新的顶点后需要比较新顶点与Enew中边权值后更新该数组，每一次接入新的顶点及是选择该数组中权值最小的顶点，将生成的最小生成树的边存入mst数组

```
//Use Prim algorithm to construct MST
void constructMST(Graph& G, char start,vector<Arc>& mst) {
  CloseNode close_nodes[MAX_VERTEX_NUM];
  int current = getLocation(G, start);

  //Initialize the close nodes
  for (int i = 0; i < G.vex_num; i++) {
    if (i != current) {
      close_nodes[i].adj_vex = start;
      close_nodes[i].low_cost = G.adj_table[current][i];
    }
    else
      close_nodes[i].low_cost = 0;
  }

  //Use recycle to add arc to MST
  for (int i = 1; i < G.vex_num; i++) {
    current = getMinAdjVex(G, close_nodes);
    Arc temp = {
close_nodes[current].adj_vex,G.vexs[current],close_nodes[current].low_cost };
    mst.push_back(temp);
    close_nodes[current].low_cost = 0;
    //Compare the close nodes'value of new node and pre one to refresh the closes
nodes array
    for (int j = 0; j < G.vex_num; j++) {
      if (G.adj_table[current][j] < close_nodes[j].low_cost) {
        close_nodes[j].adj_vex = G.vexs[current];
        close_nodes[j].low_cost = G.adj_table[current][j];
      }
    }
  }
}
```

显示最小生成树

遍历mst数组，输出最小生成树的所有边

```
//Display the MST
void displayMST(vector<Arc>& mst) {
  for (auto i : mst) {
    cout << i.node1 << "-<" << i.value << ">->" << i.node2 << endl;
  }
}
```

函数接口说明

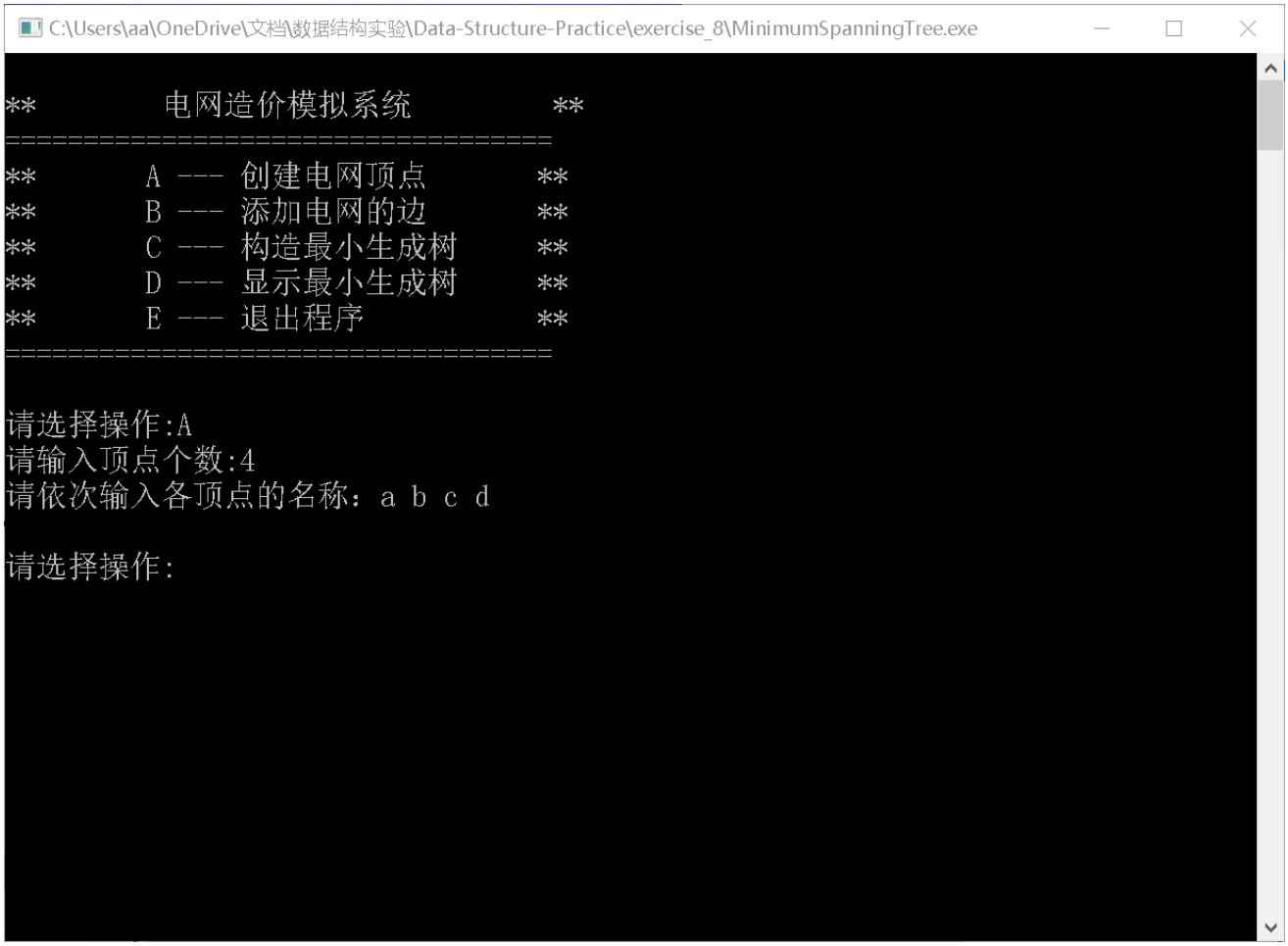
返回值类型	成员函数名	参数	属性	功能
void	createUDG	(Graph& G)	非成员函数	构造无向图
int	getLocation	(Graph& G,char name)	非成员函数	返回顶点在顶点集中坐标
int	getMinAdjVex	(Graph& G,CloseNode(&close_node)[MAX_VERTEX_NUM])	非成员函数	获取E中权值最小边的顶点
void	constructMST	(Graph& G,char start,vector& mst)	非成员函数	构造最小生成树
void	display	(vector& mst)	非成员函数	显示最小生成树

5.使用方法及函数功能演示

双击`MinimumSpanningTree.exe`运行程序，出现功能选择菜单，按照提示进行输入

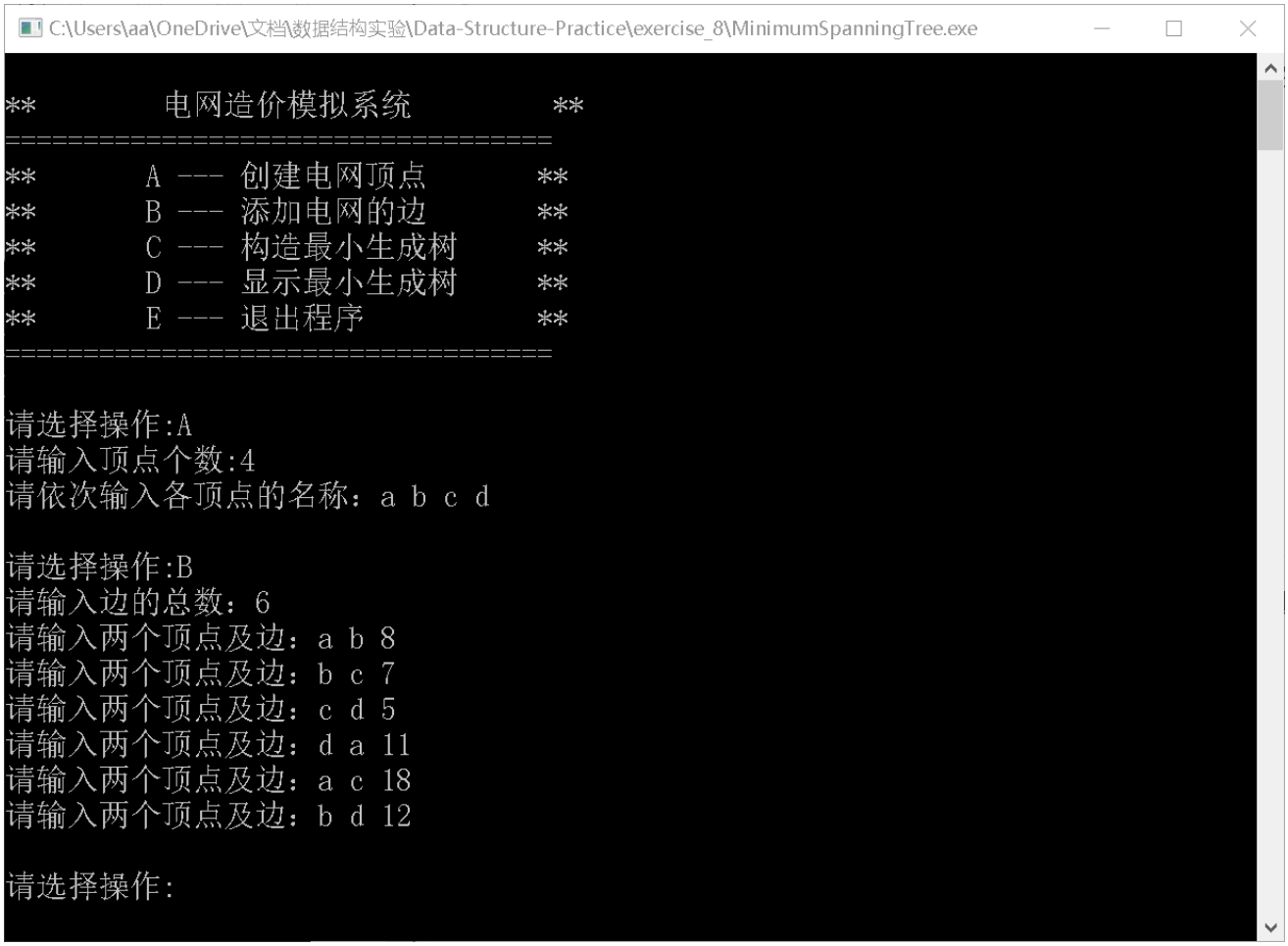
输入顶点

输入A选择该功能，随后先输入顶点个数，再以空格为间隔输入各顶点名称



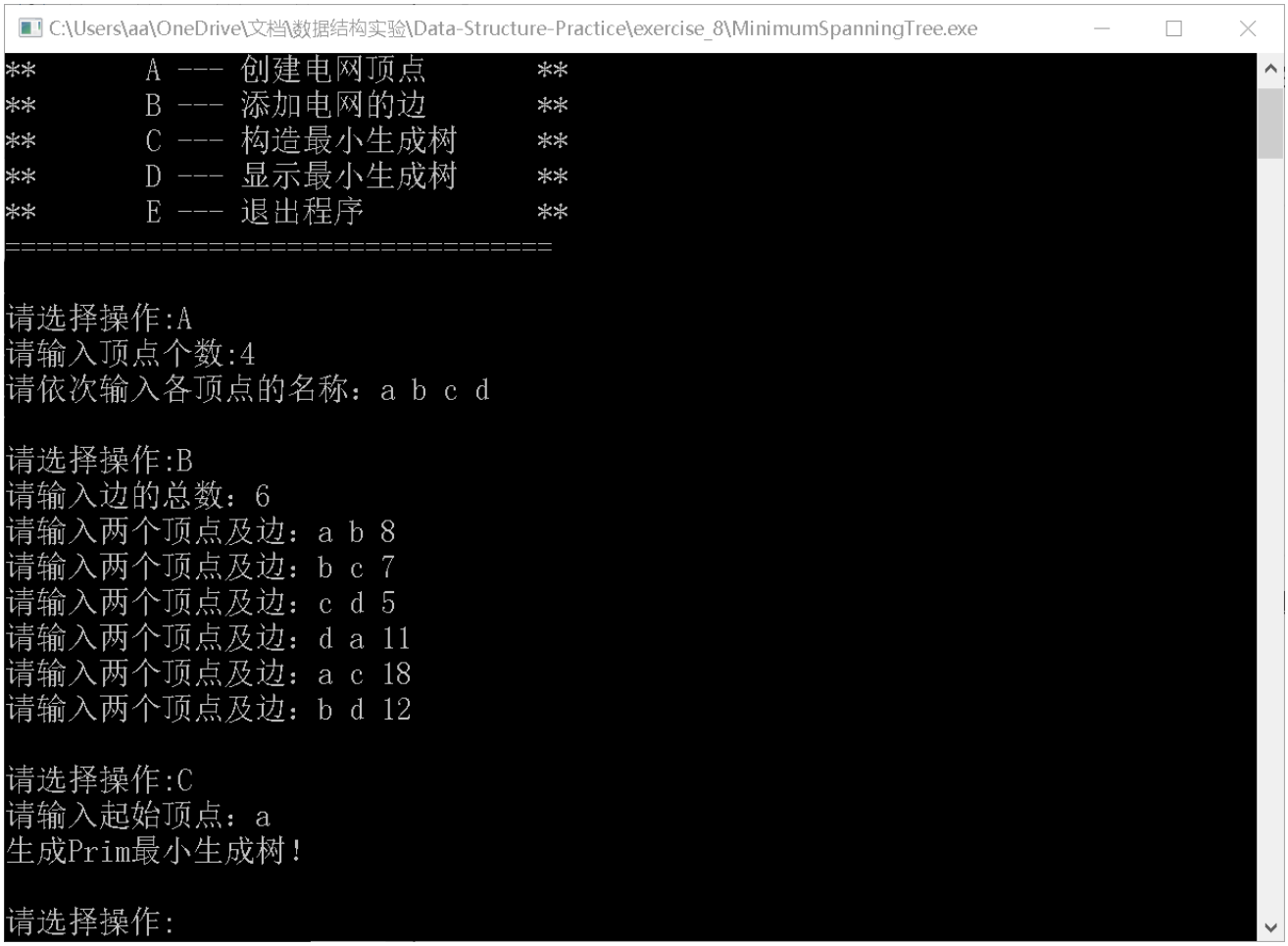
输入边

输入B选择该功能，随后先输入边条数，再逐条输入边，格式为（顶点1 顶点2 权重）



生成最小生成树

输入C选择该功能，在顶点和边都输入完成后才可以生成最小生成树，声称完后系统会提示“生成Prim最小生成树”

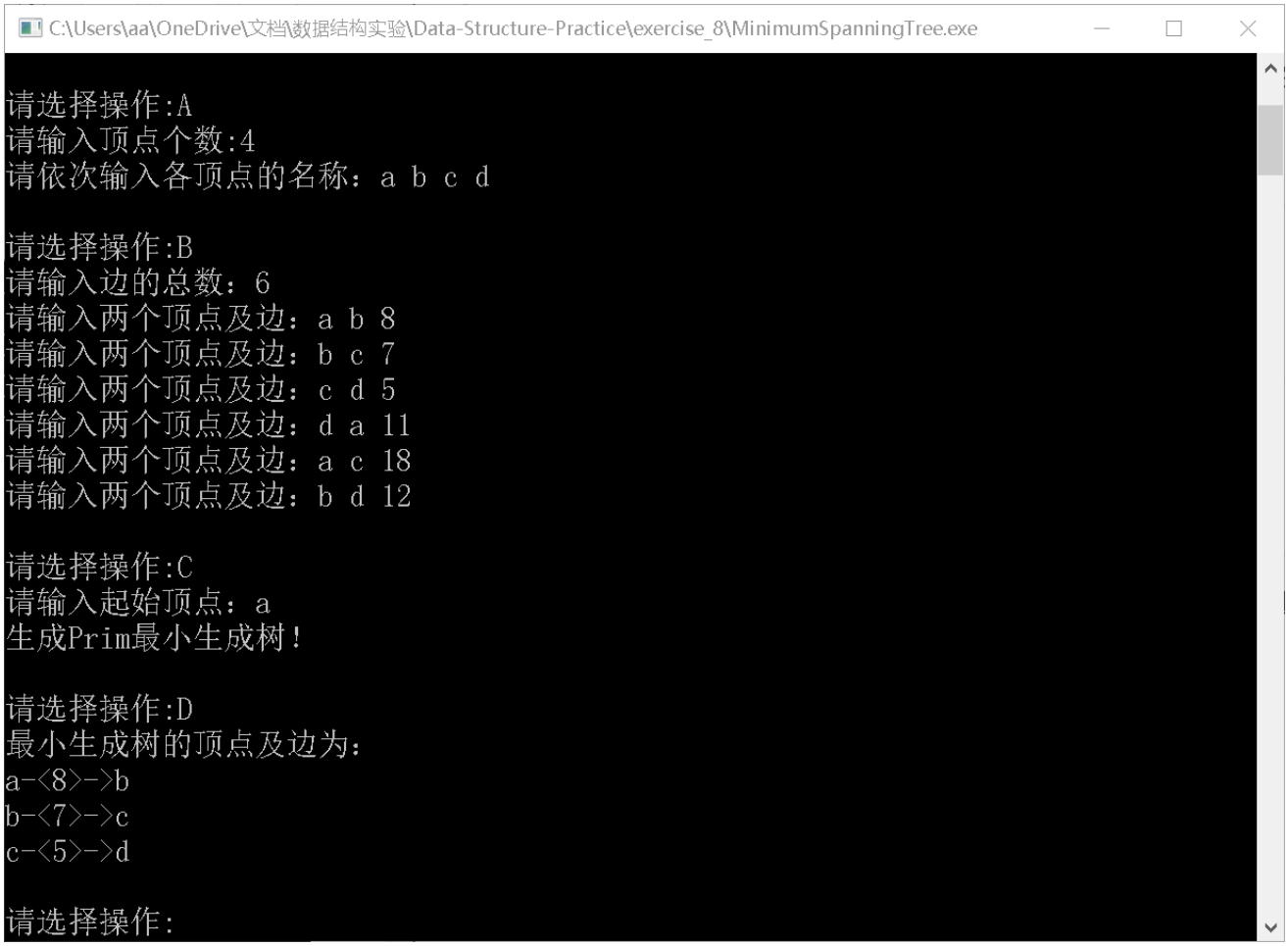


显示最小生成树

输入D显示最小生成树，显示格式为

顶点1 权重 顶点2

系统将依次显示出最小生成树的每一条边



退出程序

输入E退出该程序，若要再次启动则重新运行exe文件