

**TONGJI UNIVERSITY**

数据结构课程设计

项目名称 电网建设造价模拟系统

学 院 计算机科学与技术学院

专 业 软件工程

学生姓名 杨瑞晨

学 号 2351050

指导教师 张颖

日 期 2024 年 12 月 4 日

|  |  |  |
| --- | --- | --- |
|  | 目 录 |  |
| [1](#_bookmark0) | [项目分析](#_bookmark0) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
|  | [1.1 项目背景分析](#_bookmark1)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
|  | [1.2 项目功能分析](#_bookmark2)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
|  | [1.2.1 功能要求](#_bookmark3)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
|  | [1.2.2 输入要求](#_bookmark4)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
|  | [1.2.3 输出要求](#_bookmark5)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
|  | [1.2.4 项目实例](#_bookmark6)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 1 |
| [2](#_bookmark7) | [项目设计](#_bookmark7) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.1 数据结构设计](#_bookmark8)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.1.1 边（Edge）](#_bookmark9) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.1.2 点（Vertex）](#_bookmark10) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.1.3 图（Graph）](#_bookmark11) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.2 类设计](#_bookmark12) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.2.1 Edge 类](#_bookmark13)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 2 |
|  | [2.2.2 Vertex 类](#_bookmark14) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 3 |
|  | [2.2.3 Graph 类](#_bookmark15)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 3 |
| [3](#_bookmark16) | [项目实现](#_bookmark16) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 5 |
|  | [3.1 流程表示](#_bookmark17)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 5 |
|  | [3.2 功能实现](#_bookmark18)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 5 |
|  | [3.2.1 初始化顶点（initVertexes）](#_bookmark19)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 5 |
|  | [3.2.2 初始化边（initEdges）](#_bookmark20) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 6 |
|  | [3.2.3 构建最小生成树（kruskalMST）](#_bookmark21) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 7 |
|  | [3.2.4 打印最小生成树（printMST）](#_bookmark22) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 8 |
|  | [3.3 main 函数](#_bookmark23) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 8 |
| [4](#_bookmark24) | [项目测试](#_bookmark24) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 10 |
|  | [4.1 输入有效数据](#_bookmark25)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 10 |
|  | [4.1.1 正常测试](#_bookmark26)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 10 |
|  | [4.1.2 边界测试](#_bookmark27)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 11 |
|  | [4.2 健壮性测试](#_bookmark28) · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 11 |
| [5](#_bookmark29) | [项目心得与体会](#_bookmark29)· · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · | 12 |

I

# 项目分析

* 1. 项目背景分析

在网络设计和连接问题中，最小生成树是一个关键概念，它涉及到如何在加权连通图中找到一棵连接所有顶点的树，且这棵树的总权重（即所有边的权重之和）最小。这样的树可以被视为网络布线成本的最小化，其中顶点代表网络节点（如城市、计算机等），边的权重代表连接这些节点的成本（如距离、电缆长度等）。最小生成树问题在多个领域都有广泛的应用，包括网络设计、电路设计、供水系统规划等。

* 1. 项目功能分析
     1. 功能要求

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。

项目功能要求：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) 条线路，选择其中的 n-1 条使总的耗费最少。

2

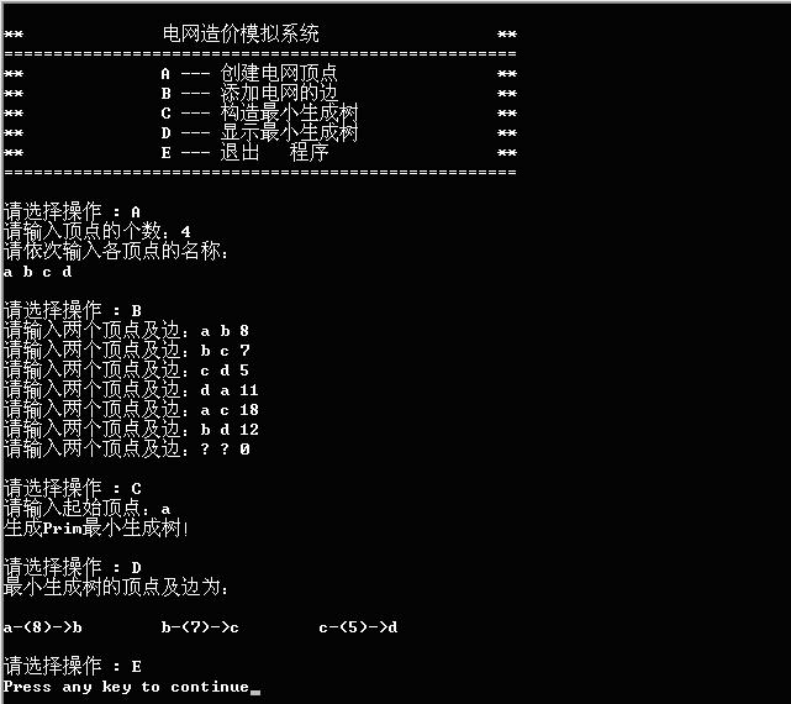
* + 1. 输入要求

输入对应的操作构建电网造价模拟系统。包括顶点个数 n，以及 n 个顶点之间的边的权重。

* + 1. 输出要求

输出最小生成树的顶点和边的信息。

* + 1. 项目实例



# 项目设计

* 1. 数据结构设计

根据题目的要求，采用 Kruskal 算法生成最小生成树，确立了以下数据结构：

* + 1. 边（Edge）

边是构成图的基本元素之一，包含起点、终点和权重三个属性。边的权重表示该边的成本或长度。（1） 顶点连接表示：在电网造价模拟中，边用于表示两个顶点之间的连接，因此需要记录每个边的起点和终点。（2） 权重存储：每条边都有一个权重，代表边的成本或长度，因此边结构需要存储权重信息。（3） 排序需求：Kruskal 算法要求边按权重排序，因此边结构需要支持排序操作。

* + 1. 点（Vertex）

点代表图中的顶点，包含名称和祖先两个属性。名称用于标识顶点，祖先用于在并查集中追踪顶点的祖先。（1） 顶点标识：顶点需要一个唯一的标识，名称是一个直观的选择。（2） 并查集支持：在 Kruskal 算法中，使用并查集来检测环，并合并集合。因此，顶点结构需要记录每个顶点在并查集中的祖先。

* + 1. 图（Graph）

图是由顶点和边构成的数据结构，包含顶点数、边数、顶点数组和边数组。图还包含一个指向最小生成树的指针。（1） 顶点和边的存储：图需要存储所有的顶点和边，因此需要有相应的数组来存储。（2） 最小生成树：图需要一个额外的图来存储最小生成树的结果。（3） 快速排序：图中需要对边进行排序，因此需要实现快速排序算法。

* 1. 类设计
     1. Edge 类

Edge 类是边的实现，存储边的起点、终点和权重，并且支持边的比较操作，以便在算法实现中进行排序。

1

*//* 边

**class Edge**

{

**public**:

**int** start, end; *//* 顶点

**double** weight; *//* 权重

**bool operator**<(**const** Edge &other) **const** { **return** weight < other.weight; } **bool operator**>(**const** Edge &other) **const** { **return** weight > other.weight; } Edge() {}

2

3

4

5

6

7

8

9

10

Edge(**int** u, **int** v, **double** weight) : start(u), end(v), weight(weight) {}

};

11

* + 1. Vertex 类

设计该类是为了表示图中的顶点并管理与顶点相关的信息。它提供了顶点名称的存储和并查集中祖先的跟踪，这对于在图形算法中处理顶点至关重要。

1

*//* 点

**struct Vertex**

{

**public**:

**char** name[STRING\_SIZE]; *//* 名称

**int** ancestor; *//* 记录在并查集中的祖先

Vertex() {}

Vertex(**char** \*name, **int** ancestor) : ancestor(ancestor) { strcpy(**this**->name, name); }

};

2

3

4

5

6

7

8

9

* + 1. Graph 类

核心类，负责整个电网构造项目的图形表示和算法实现。它定义了初始化图形的方法以及构建和打印最小生成树的方法。

1

*//* 图

**class Graph**

{

**template** <**class T**>

**friend void** quickSort(T \*arr, **int** low, **int** high);

**private**:

**int** V; *//* 顶点数

**int** E; *//* 边数 Vertex \*vertexes; *//* 顶点数组 Edge \*edges; *//* 边数组 Graph \*mst; *//* 最小生成树

*//* 根据名称查找顶点

**int** findVertex(**char** \*name)

{

**for** (**int** i = 0; i < V; ++i) { **if** (strcmp(name, vertexes[i].name) == 0) **return** i; }

**return** -1;

}

*//* 在并查集中查找顶点祖先

**int** findAncestor(**int** i)

{

**if** (vertexes[i].ancestor != i) { vertexes[i].ancestor =

*↩*→ findAncestor(vertexes[i].ancestor); } *//* 路径压缩

**return** vertexes[i].ancestor;

}

*//* 重置图

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27 **void** reset() {

28 V = 0, E = 0;

29 **if** (vertexes) **delete**[] vertexes;

30 **if** (edges) **delete**[] edges;

31 **if** (mst) **delete** mst;

32 vertexes = **nullptr**;

33 edges = **nullptr**;

34 }

35

36 *//* 根据名称快速排序边

37 **void** quickSortEdgesByName(Edge edges[], **int** left, **int** right, Vertex vertexes[])

38 {

39 **if** (left >= right) **return**;

40

41 **int** i = left, j = right;

42 Edge pivot = edges[left];

43

44 **while** (i < j)

45 {

46 *//* 从右向左，找到起点名称小于基准的边

47 **while** (i < j && strcmp(vertexes[edges[j].start].name, vertexes[pivot.start].name)

*↩*→ >= 0) j--;

48 **if** (i < j) edges[i++] = edges[j];

49

50 *//* 从左向右，找到起点名称大于基准的边

51 **while** (i < j && strcmp(vertexes[edges[i].start].name, vertexes[pivot.start].name)

*↩*→ <= 0) i++;

52

53 }

54

**if** (i < j) edges[j--] = edges[i];

55

56

57

58

59 }

60 **public**:

edges[i] = pivot;

quickSortEdgesByName(edges, left, i - 1, vertexes); quickSortEdgesByName(edges, i + 1, right, vertexes);

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81 };

Graph(**int** V = 0, **int** E = 0) : V(V), E(E)

{

edges = **nullptr**; vertexes = **nullptr**; mst = **nullptr**;

edges = **new** Edge[E]; vertexes = **new** Vertex[V];

}

~Graph()

{

**delete**[] vertexes; **delete**[] edges; **delete** mst;

}

**void** printMST(); *//* 打印最小生成树

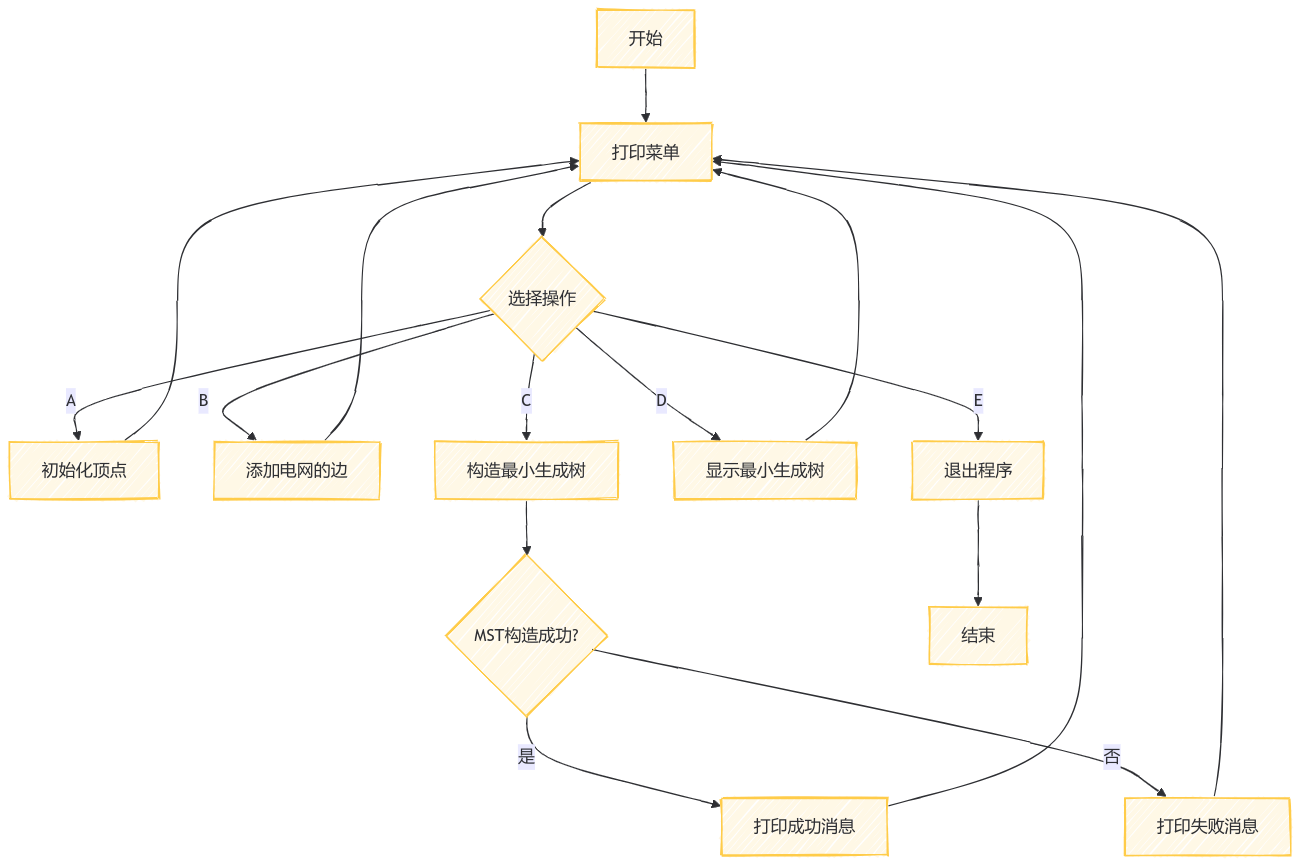
**bool** kruskalMST(); *// Kruskal* 算法构造最小生成树

**void** initVertexes(); *//* 初始化顶点

**void** initEdges(); *//* 初始化边

# 项目实现

* 1. 流程表示



* 1. 功能实现
     1. 初始化顶点（initVertexes）
        + 用户输入顶点数量：程序首先要求用户输入顶点的数量，并进行错误检查（非负整数）。
        + 动态分配顶点数组：根据用户输入的数量，动态分配一个顶点数组。
        + 用户输入顶点名称：程序要求用户依次输入每个顶点的名称，并检查是否有重复的顶点名称。
        + 顶点初始化：程序将用户输入的顶点名称和索引存储到顶点数组中。

1

**void** initVertexes()

{

reset();

cout << " 请输入顶点的个数：";

**while** (true)

{

cin >> V;

**if** (cin.fail() || V < 2)

{

cerr << " 输入错误，请重新输入！" << endl; cin.clear(); cin.ignore(numeric\_limits<streamsize>::max(), '\n'); **continue**;

}

**else break**;

}

vertexes = **new** Vertex[V];

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

|  |  |  |
| --- | --- | --- |
| 20 |  | cout << " 请依次输入各顶点的名称：" << endl; |
| 21 |  | **for** (**int** i = 0; i < V; ++i) |
| 22  23 |  | {  **char** name[STRING\_SIZE]; |
| 24 |  | cout << " 第 " << i + 1 << " 个顶点："; |
| 25 |  | cin >> name; |
| 26 |  | **if** (findVertex(name) != -1) |
| 27  28 |  | {  cerr << " 顶点名称重复，请重新输入！" << endl; |
| 29 |  | --i; |
| 30 |  | } |
| 31 |  | **else** vertexes[i] = Vertex(name, i); |
| 32  33 | } | } |

* + 1. 初始化边（initEdges）
       - 用户输入边的数量：程序要求用户输入边的数量，并进行错误检查（边数应在一定范围内）。
       - 动态分配边数组：根据用户输入的数量，动态分配一个边数组。
       - 用户输入边的详细信息：程序要求用户依次输入每条边的起点、终点和权重，并进行错误检查。
       - 边初始化：程序将用户输入的边的起点、终点和权重存储到边数组中。

|  |  |
| --- | --- |
| 1 | **void** initEdges() |
| 2 | { |
| 3 | **delete**[] edges; |
| 4 |  |
| 5 | cout << " 请输入边的个数："; |
| 6 | **while** (true) |
| 7 | { |
| 8 | cin >> E; |
| 9 | **if** (cin.fail() || E < V - 1 || E > V \* (V - 1) / 2) |
| 10 | { |
| 11 | cerr << " 输入错误，请重新输入" << endl; |
| 12 | cin.clear(); |
| 13 | cin.ignore(numeric\_limits<streamsize>::max(), '\n'); |
| 14 | **continue**; |
| 15 | } |
| 16 | **else break**; |
| 17 | } |
| 18 | edges = **new** Edge[E]; |
| 19 |  |
| 20 | cout << " 请依次输入两个顶点及边：" << endl; |
| 21 | **for** (**int** i = 0; i < E; ++i) |
| 22 | { |
| 23 | **char** u[STRING\_SIZE], v[STRING\_SIZE]; |
| 24 | **double** weight; |
| 25 | **while** (true) |
| 26 | { |
| 27 | cin >> u >> v; |
| 28 | **if** (findVertex(u) == -1 || findVertex(v) == -1) |
| 29 | { |

30

cerr << " 未找到顶点，请重新输入！" << endl; cin.clear(); cin.ignore(numeric\_limits<streamsize>::max(), '\n'); **continue**;

}

**else break**;

}

**while** (true)

{

cin >> weight;

**if** (cin.fail() || weight < 0)

{

cerr << " 权重输入错误，请重新输入！" << endl; cin.clear(); cin.ignore(numeric\_limits<streamsize>::max(), '\n'); **continue**;

}

**else break**;

}

edges[i] = Edge(findVertex(u), findVertex(v), weight);

}

}

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

* + 1. 构建最小生成树（kruskalMST）
       - 初始化最小生成树：程序创建一个新的 Graph 对象来存储最小生成树。
       - 排序边：程序将所有的边按照权重从小到大排序。
       - 遍历边并构建 MST：程序遍历排序后的边，使用并查集来检测环，并逐步构建最小生成树。
       - 检查是否完成：当选择的边数等于顶点数减一时，最小生成树构建完成。

1

*//* 快速排序模版函数

**template** <**class T**>

**void** quickSort(T \*arr, **int** low, **int** high)

{

**if** (low > high) **return**;

T mid = arr[(low + high) / 2];

**int** i = low, j = high;

**while** (i < j) {

**while** (arr[i] < mid) ++i; **while** (arr[j] > mid) --j; **if** (i <= j) {

swap(arr[i], arr[j]);

++i;

--j;

}

}

**if** (low < j) quickSort(arr, low, j);

**if** (i < high) quickSort(arr, i, high);

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

quickSort 函数：模板函数，用于对任意类型的数组进行快速排序。

1

*//* 快速排序模版函数

**template** <**class T**>

**void** quickSort(T \*arr, **int** low, **int** high)

{

**if** (low > high) **return**;

T mid = arr[(low + high) / 2];

**int** i = low, j = high;

**while** (i < j) {

**while** (arr[i] < mid) ++i; **while** (arr[j] > mid) --j; **if** (i <= j) {

swap(arr[i], arr[j]);

++i;

--j;

}

}

**if** (low < j) quickSort(arr, low, j);

**if** (i < high) quickSort(arr, i, high);

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

* + 1. 打印最小生成树（printMST）
       - 检查 MST 是否已构造：在显示最小生成树之前，先检查是否已经成功构造了最小生成树。
       - 排序 MST 的边：为了更清晰地显示最小生成树，将 MST 中的边按照顶点名称排序。
       - 打印 MST 的边：遍历 MST 中的每条边，并打印出起点、权重和终点。

1

**void** printMST()

{

**if** (mst == **nullptr**)

{

cerr << " 请先构造最小生成树！" << endl;

**return**;

}

quickSortEdgesByName(mst->edges, 0, mst->E - 1, vertexes); cout << " 最小生成树的顶点及边如下：" << endl;

**for** (**int** i = 0; i < mst->E; ++i)

{

cout << vertexes[mst->edges[i].start].name << " - (" << mst->edges[i].weight << ")

*↩*→ -> " << vertexes[mst->edges[i].end].name << endl;

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

* 1. main 函数

根据用户选择调用相关函数。

* 初始化顶点：程序首先调用 initVertexes 函数来初始化顶点。
* 初始化边：程序调用 initEdges 函数来初始化边。
* 构建最小生成树：程序调用 kruskalMST 函数来构建最小生成树。
* 打印最小生成树：程序调用 printMST 函数来打印最小生成树。

1

**void** printMST()

{

**if** (mst == **nullptr**)

{

cerr << " 请先构造最小生成树！" << endl;

**return**;

}

quickSortEdgesByName(mst->edges, 0, mst->E - 1, vertexes); cout << " 最小生成树的顶点及边如下：" << endl;

**for** (**int** i = 0; i < mst->E; ++i)

{

cout << vertexes[mst->edges[i].start].name << " - (" << mst->edges[i].weight << ")

*↩*→ -> " << vertexes[mst->edges[i].end].name << endl;

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

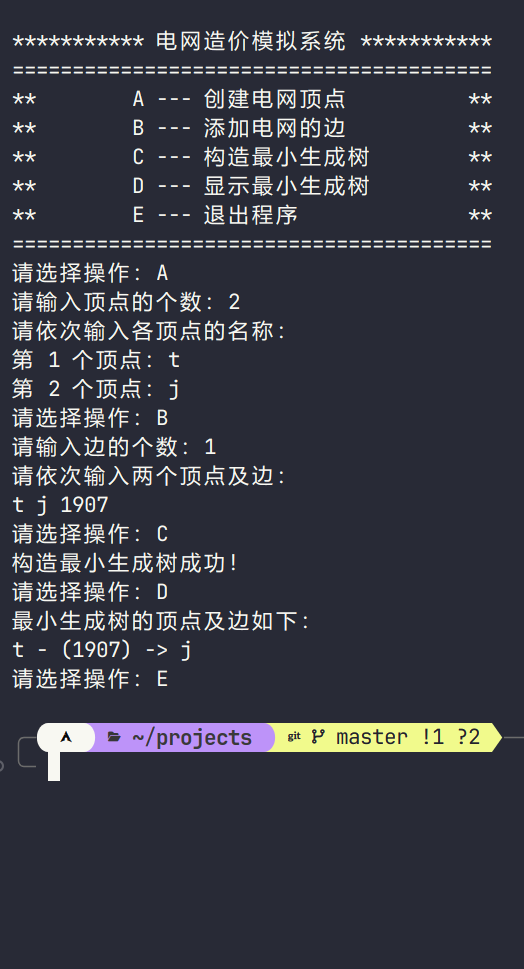
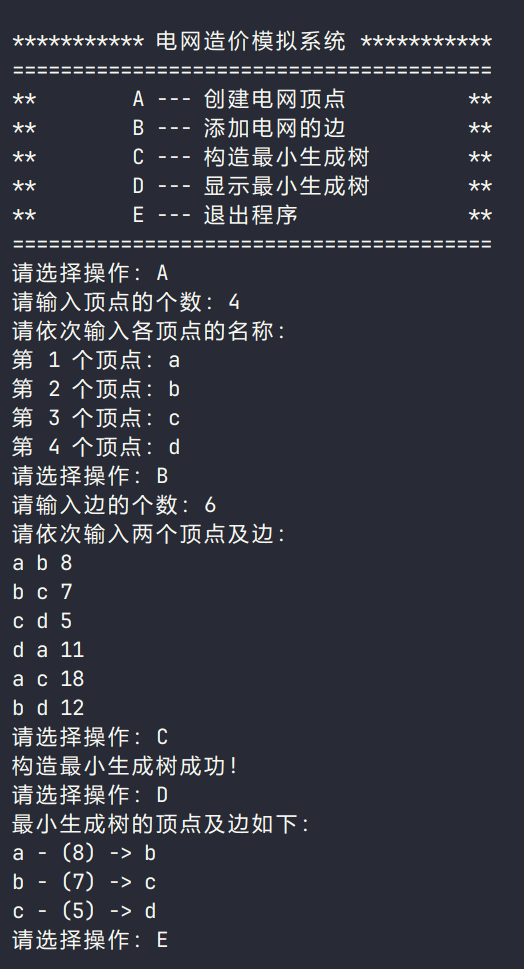
15

16

17

# 项目测试

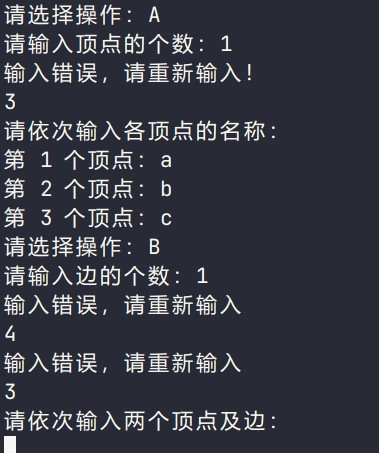
* 1. 输入有效数据
     1. 正常测试



* + 1. 边界测试

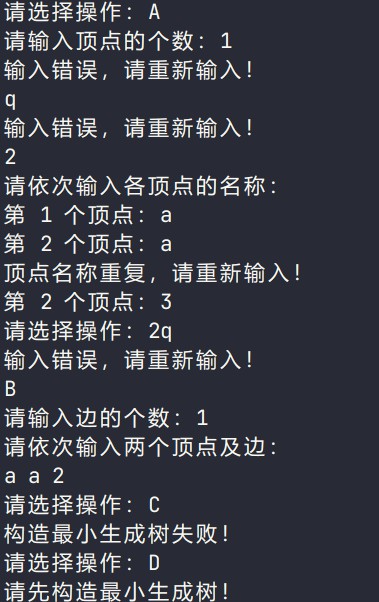
点的数量 𝑛 需要 ≥ 2，边的数量需要 ≥ 𝑛 − 1 且 ≤ 𝑛(𝑛−1) 。

2



* 1. 健壮性测试

输入非法字符或负数，程序提示输入错误，重新输入。



# 项目心得与体会

通过本项目的深入开发与实践，我获得了许多宝贵的经验和深刻的认识，这些不仅增强了我的专业知识，也提升了我的实践技能。

在实现电网造价模拟系统的过程中，我深刻体会到了数据结构的重要性。通过设计和实现 Edge、 Vertex 和 Graph 类，我不仅复习了基本的数据结构知识，如数组、结构体等，还学会了如何将这些数据结构应用于解决实际问题；Kruskal 算法的实现让我对贪心算法有了更深的理解。在实际操作中，我学会了如何对算法进行优化，比如通过快速排序模板函数来提高边的排序效率，以及如何使用并查集来高效地处理元素的合并和查找操作；在编写和调试代码的过程中，我的编程能力得到了显著提升。我学会了如何编写清晰、高效且易于维护的代码，并且掌握了一些调试技巧，这些都对我未来的编程工作大有裨益。

总的来说，这个项目不仅让我掌握了数据结构和算法的具体应用，还提升了我的编程能力、问题解决能力和系统设计能力。这些经验和技能将为我未来的学习和工作奠定坚实的基础。