数据结构课程设计

项目说明文档

电网建设造价模拟系统

|  |  |
| --- | --- |
| 作者姓名： | 杨烜赫 |
| 学 号： | 2252709 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

1. **题目简介**

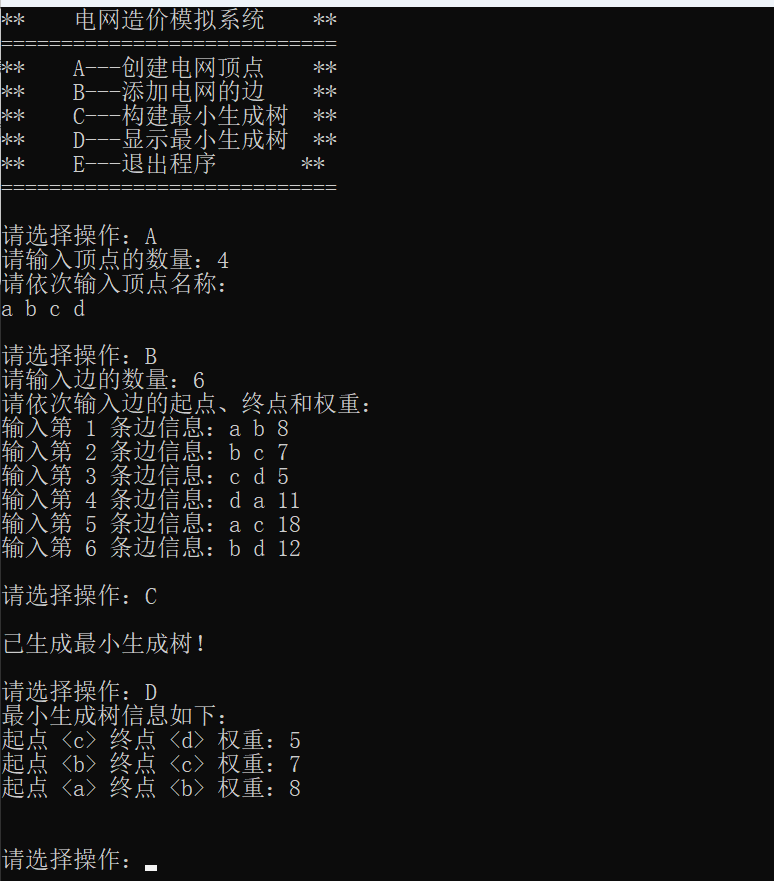
**1.实验背景**

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.实验功能**

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。

**3.项目示例**

****

1. **项目设计**

**1.数据结构设计**

**struct Connection**

表示电网中的一条边，包含起点、终点和边的权重。还包含比较运算符，用于排序。

**struct Node**

表示电网中的一个顶点，包含顶点的标签和在并查集中的根节点。

**2.类结构设计**

**class PowerGrid**

管理整个电网的类，包含电网的顶点、边和最小生成树的构建。

**3.成员与操作设计**

**成员变量**

PowerGrid\* min\_span\_tree：指向最小生成树的指针。

int node\_count, edge\_count：顶点和边的数量。

Connection\* connections：存储边的数组。

Node\* nodes：存储顶点的数组。

**操作方法**

void initialize\_nodes()：初始化电网的顶点。

void initialize\_edges()：初始化电网的边。

bool kruskal\_algorithm()：使用Kruskal算法构建最小生成树。

void print\_min\_span\_tree()：打印最小生成树的信息。

void reset\_graph()：重置电网图，清理内存。

int find\_node(const string& name)：根据名称查找顶点。

int find\_root(int index)：在并查集中查找顶点的根。

**辅助函数**

char get\_menu\_choice()：获取用户菜单选择。

## 4.算法设计

**1. Kruskal算法（用于构建最小生成树）**

目的: 使用Kruskal算法构建电网的最小生成树，以找到最低成本的电网构建方案。实现: 算法首先对所有的边按权重进行排序。然后遍历排序后的边，每次选择权重最小的边，检查这条边是否会形成环（使用并查集）。如果不形成环，则将其加入到最小生成树中。重复此过程直到选出足够的边（顶点数-1）或无更多边可选。

**2. 并查集（用于检测环）**

目的: 在Kruskal算法中使用，并查集用于快速检查添加新边是否会在生成树中形成环。实现: 每个顶点一开始都是自己的根。当考虑加入一条边时，检查这条边的两个顶点是否拥有相同的根。如果是，则这条边会形成环；否则，可以安全地加入这条边，并更新顶点的根信息。

**3. 快速排序（用于排序边）**

目的: 对电网的所有边按权重进行排序，以便Kruskal算法按顺序选择边。

实现: 使用经典的快速排序算法。选择一个枢轴，将所有小于枢轴的元素移动到枢轴的左边，大于枢轴的移动到右边，然后递归地在左右两侧进行同样的操作。

这些算法的设计和实现是电网造价模拟系统的核心，它们使得程序能够有效地处理电网构建问题，并找到成本最低的连接方式。Kruskal算法的应用确保了生成树的最小权重，而快速排序和并查集提高了整体的效率和性能。

1. **项目实现**

**3.1. 顶点初始化**

1. **void** initialize\_nodes() {
2. reset\_graph();
4. cout << "请输入顶点的数量：";
5. **while** (**true**)
6. {
7. cin >> node\_count;
8. **if** (node\_count < 1 || node\_count>1024 || !cin.good())
9. {
10. cin.clear();
11. cin.ignore(1024, '\n');
12. cout << "顶点数量应该在1-1024之间，请重新输入：";
13. **continue**;
15. }
16. **else**
17. {
18. **break**;
19. }
20. }
21. nodes = **new** Node[node\_count];
23. cout << "请依次输入顶点名称：" << endl;
24. **for** (**int** i = 0; i < node\_count; ++i) {
25. string node\_name;
26. cin >> node\_name;
27. **if** (find\_node(node\_name) != -1) {
28. cout << "顶点 " << node\_name << " 已存在，请重新输入：";
29. i--;
30. }
31. **else** {
32. nodes[i].label = node\_name;
33. nodes[i].root = i;
34. }
35. }
36. }

函数 initialize\_nodes 负责初始化电网的顶点，包括设置顶点的数量和名称。首先，函数重置任何现有的图结构，然后提示用户输入顶点的数量，确保输入在有效范围内（1到1024之间）。接着，函数为每个顶点分配内存并逐个读取用户输入的顶点名称，同时检查新输入的顶点名称是否已存在以避免重复。对于每个有效的顶点，函数将其名称存储在 nodes 数组中，并初始化其在并查集中的根节点。这个过程为后续的电网边初始化和最小生成树构建奠定了基础，确保了电网的顶点数据准确无误。

**3.2. 边的初始化功能**

1. // 初始化边
2. **void** initialize\_edges() {
3. **delete**[] connections;
5. cin.clear();
6. cin.ignore(1024, '\n');
8. cout << "请输入边的数量：";
9. **while** (**true**)
10. {
11. cin >> edge\_count;
12. **if** (edge\_count < 1 || edge\_count>node\_count \* (node\_count - 1) / 2 || !cin.good())
13. {
14. cin.clear();
15. cin.ignore(1024, '\n');
16. cout << "边的数量过多，请重新输入：";
17. **continue**;
18. }
19. **else**
20. {
21. **break**;
22. }
23. }
24. connections = **new** Connection[edge\_count];
26. cout << "请依次输入边的起点、终点和权重：" << endl;
27. **for** (**int** i = 0; i < edge\_count; i++) {
28. cout << "输入第 " << i + 1 << " 条边信息：";
29. string start, end;
30. **double** weight;
31. cin >> start >> end;
32. connections[i].start = find\_node(start);
33. connections[i].end = find\_node(end);
35. **if** (connections[i].start == -1 || connections[i].end == -1) {
36. cout << "边的信息输入错误，请重新输入：" << endl;
37. cin.clear();
38. cin.ignore(1024, '\n');
39. i--;
40. **continue**;
41. }
43. cin >> weight;
44. **if** (cin.fail() || weight <= 0) {
45. cout << "权重输入错误，请重新输入：" << endl;
46. cin.clear();
47. cin.ignore(1024, '\n');
48. i--;
49. **continue**;
50. }
51. connections[i].weight = weight;
52. }
53. }

函数 initialize\_edges 负责初始化电网中的边，包括设置边的数量和每条边的具体信息（起点、终点和权重）。首先，函数清除之前可能存在的边信息并读取用户输入的边的数量，确保输入在有效范围内（小于或等于顶点数对的数量）。然后为每条边分配内存，并要求用户依次输入每条边的起点、终点和权重信息。对于每条边，函数通过 find\_node 函数将起点和终点的名称转换为对应的顶点索引，并验证输入的权重是否有效。如果输入的边信息有误（如不存在的顶点名称或无效的权重），函数会提示错误并要求重新输入，直到所有边信息都被正确输入。这个过程为构建电网的最小生成树提供了必要的边信息，是电网模拟的关键组成部分。

**3.3. 构建最小生成树功能实现**

1. // Kruskal算法实现最小生成树
2. **bool** kruskal\_algorithm() {
3. quick\_sort(connections, 0, edge\_count - 1);
5. **delete** min\_span\_tree;
6. min\_span\_tree = nullptr;
8. min\_span\_tree = **new** PowerGrid(node\_count, node\_count - 1);
10. **int** selected\_edges = 0;
11. **for** (**int** i = 0; i < edge\_count; i++) {
12. **int** ancestor1 = find\_root(connections[i].start);
13. **int** ancestor2 = find\_root(connections[i].end);
15. **if** (ancestor1 != ancestor2) {
16. selected\_edges++;
17. nodes[ancestor1].root = ancestor2;
18. min\_span\_tree->connections[selected\_edges - 1] = connections[i];
19. }
20. **if** (selected\_edges == node\_count - 1) {
21. cout << endl;
22. **return** **true**;
23. }
24. }
26. **delete** min\_span\_tree;
27. min\_span\_tree = nullptr;
28. cout << endl;
29. **return** **false**;
30. }

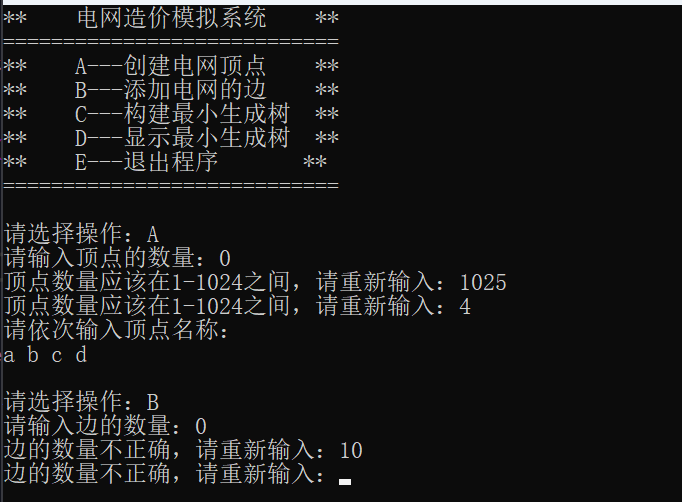
函数 kruskal\_algorithm 实现了Kruskal算法来构建电网的最小生成树，以找到连接所有顶点的最低成本方案。算法首先通过快速排序对所有边按权重进行排序。接着，它遍历排序后的边，每次选择一条边并使用并查集方法检查这条边是否会在已选边中形成环。如果不会形成环，则将该边加入到最小生成树中。这个过程持续进行，直到选取足够的边（顶点数减一）或无更多边可选。如果成功选取了足够的边，算法返回 true 并构建最小生成树；如果无法构建，比如由于不连通等原因，算法清除已构建的树并返回 false。这个过程确保了所构建的最小生成树覆盖了所有顶点，且总权重最小。

**3.4. 快速排序边功能实现**

1. // 快速排序模板函数
2. **template**<**typename** T>
3. **void** quick\_sort(T arr[], **int** left, **int** right) {
4. **if** (left > right) **return**;
5. T pivot = arr[(left + right) / 2];  // 选择中间值作为枢轴
6. **int** i = left, j = right;
7. **while** (i < j) {
8. **while** (arr[i] < pivot) i++;
9. **while** (arr[j] > pivot) j--;
10. **if** (i <= j) {
11. swap(arr[i], arr[j]);  // 交换两个元素
12. i++;
13. j--;
14. }
15. }
16. **if** (left < j) quick\_sort(arr, left, j);  // 对左边部分进行快速排序
17. **if** (i < right) quick\_sort(arr, i, right); // 对右边部分进行快速排序
18. }

Quick\_sort 函数是一个通用的快速排序模板，用于对任何类型 T 的数组进行高效排序。该函数采用分治策略，首先选择数组中间的值作为枢轴，然后将数组分成两部分，一边的元素都小于枢轴，另一边的元素都大于枢轴。通过递归地对数组的这两部分分别进行相同的排序过程，快速排序能够有效地对整个数组进行排序。此函数的关键在于其分割步骤，其中元素交换和递归调用确保了整个数组最终被正确排序。由于其高效性和简洁性，快速排序广泛应用于各类排序需求中，特别是在需要排序大量数据的场合。

**健全性测试：**



**输入验证与错误处理:**

在 initialize\_nodes 和 initialize\_edges 函数中，程序通过循环和条件语句确保用户输入的顶点数量和边数量在有效范围内，并且对错误输入进行了处理，如清除输入缓冲区并提示重新输入。

**内存管理:**

程序在创建新的最小生成树之前，会删除旧的最小生成树（如果存在），避免内存泄漏。

使用 new 和 delete 对象管理内存，确保动态分配的内存得到适当的创建和释放。

**并查集的应用:**

在Kruskal算法中使用并查集方法来检测环，这是一种高效的方法，可以减少不必要的边的添加，并防止生成树中出现环。

边的有效性验证:

在添加边时，程序检查边的起点和终点是否有效，确保所有添加的边都是合法的。

**稳健的排序算法:**

使用快速排序算法对边进行排序，该算法的实现考虑了各种可能的边界情况，例如数组只有一个元素或没有元素。

**程序逻辑的健全性:**

程序中的各种函数逻辑严谨，如在Kruskal算法中正确地实现了最小生成树的构建流程，避免了不必要的逻辑错误

1. **项目创新点**

该电网造价模拟系统的创新之处在于其将Kruskal算法应用于最小生成树的构建，为解决电网设计中的最低成本连接问题提供了一个有效的算法实现。特别地，系统通过结合快速排序和并查集算法，不仅优化了边的选择过程，也高效地避免了在生成树中形成环的可能性。此外，程序使用模板函数实现的快速排序算法增加了其通用性和灵活性。通过交互式的用户输入界面，系统允许用户直观地输入和修改电网的顶点和边，使得该程序不仅是一个算法演示，更是一个实用的电网设计工具。

**五、实验小结**

本项目成功实现了一个电网造价模拟系统，充分展示了Kruskal算法在实际应用中的效果和效率。实验过程中，对输入验证和内存管理的重视提高了程序的健壮性和用户体验。虽然程序在功能实现方面表现出色，但在用户界面和数据处理效率方面还有改进空间，比如引入图形界面和优化大规模数据处理。总体来说，这个项目不仅在技术层面上取得了成功，也为理解和应用复杂算法提供了宝贵的实践经验。