项目说明文档

数据结构课程设计

——电网建设造价模拟系统

作 者 姓 名： 杨宇琨

学 号： 2252843

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 分析

## 1.1 背景分析

随着城市的发展和扩张，城市规模逐渐增大，小区数量也在不断增加。为了满足居民生活和工作的需求，需要建设一个能够实现小区之间相互接通的电网系统。而在建设电网系统时，需要考虑到经济性和效益性，以确保在满足功能需求的同时最大程度地节约资源和降低成本。因此，设计一个能够使总工程造价最低的电网构造方案尤为重要。

## 1.2 功能分析

1. 小区之间的电网接通：确保每个小区都可以通过电网线路与其他小区相互接通，以满足居民生活、商业和工业用电需求。

2. 最小化总工程造价：通过选择合适的电网线路，使得小区之间的电网建设总成本最低，以提高资源利用效率和降低建设成本。

3. 经济代价计算：对每条线路的建设成本进行准确计算，以便在选择电网线路时能够综合考虑经济代价和功能需求。

4. 线路选择算法：设计合适的算法，从可能的n（n-1）/2条线路中选择出n-1条，使得总的耗费最少，即实现最优的电网构造方案。

5. 可行性分析：对于不同规模和结构的城市，需要对电网构造方案的可行性进行分析，以确保方案在实际应用中能够有效实施。

综上所述，项目功能分析主要包括小区间电网接通、总工程造价最低、经济代价计算、线路选择算法和可行性分析等方面，以实现一个经济高效、功能完备的城市电网构造方案。

# 2 设计

## 2.1 数据结构设计

1. 结构体Edge：用于存储边的信息，包括两个节点（u、v）和边的造价（cost）。

2. 数组edges[MAX\_M]：用于存储所有的边，最大数量为MAX\_M。

3. 数组parent[MAX\_N]：用于存储每个节点的父节点信息，用于实现并查集的功能。

struct Edge {

int u, v, cost;

};//Two nodes and a value

Edge edges[MAX\_M]; //Store the edge

int parent[MAX\_N]; //Store the information of the root/father of every node

## 2.2 系统设计

1. 输入模块：通过input函数实现对节点数量n和边的数量m的输入，以及对每条边的节点和造价的输入，确保输入数据的合法性。

2. 初始化模块：通过init函数对并查集的parent数组进行初始化，使每个节点的父节点都是自身。

3. 并查集模块：通过find函数实现查找节点的根节点（即父节点），并通过unite函数实现将两个节点进行合并，即将其中一个节点的父节点指向另一个节点，以实现并查集的功能。

4. 快速排序模块：通过quicksort函数实现对边的造价进行快速排序，以便在后续的Kruskal算法中能够按照造价从小到大进行处理。

5. Kruskal算法模块：通过kruskal函数实现Kruskal算法，包括对边的排序、初始化并查集、遍历边并选择最小生成树的边、计算最小造价等功能。

6. 输出模块：输出最小造价电网的总造价，展示计算结果。

综上所述，系统设计分析包括了输入模块、初始化模块、并查集模块、快速排序模块、Kruskal算法模块和输出模块，能够实现对电网造价计算的功能。

# 3 实现

## 3.1 快速排序功能的实现

### 3.1.1 快速排序功能流程

当进行快速排序时，可以遵循以下流程：

1. 选择基准值：从待排序数组中选择一个基准值（pivot）。选择基准值的方式可以多样化，常见的方式包括选择第一个元素、最后一个元素、中间元素，甚至随机选择一个元素作为基准值。

2. 分区操作：将数组中的其他元素按照与基准值的大小关系分为两部分，一部分小于基准值，一部分大于基准值。分区操作可以通过双指针法或挖坑填数法来实现。具体步骤如下：

- 初始化两个指针：一个指向数组的起始位置，一个指向数组的结束位置。

- 从数组的起始位置开始，向后移动左指针，直到找到一个大于等于基准值的元素。

- 从数组的结束位置开始，向前移动右指针，直到找到一个小于等于基准值的元素。

- 如果左指针小于右指针，交换两个指针所指向的元素，使得大于基准值的元素位于右边，小于基准值的元素位于左边。

- 重复上述步骤，直到左指针大于等于右指针。

3. 递归操作：对分区后的两部分数组分别递归地应用快速排序算法。即对基准值左侧的子数组和右侧的子数组分别重复步骤1和步骤2，直到子数组长度为1或0时，表示子数组已经有序。

4. 合并：经过递归处理后，整个数组就变成了有序的。

这样，整个数组就被分割成了若干个子数组，每个子数组的基准值都处于正确的位置上，从而实现了排序。

3.1.2 快速排序核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:quicksort

Function:To sort the array based on the value

Input Parameters:array two index(low and high)

Return Value:void

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void quicksort(Edge\* edges, int left, int right)

{

//If left index equals to the right index ,return

if (left >= right)

return;

int pivot = edges[left].cost;//Set a pivot

int l = left, r = right;//Create two index

while (l < r) {

while (l < r && edges[r].cost >= pivot)

--r;//Move the right index to the first element which is greater than pivot

while (l < r && edges[l].cost <= pivot)

++l;//Move the left index to the first element which is smaller than pivot

if (l < r)

std::swap(edges[l], edges[r]);//If left index is not equal to the right index swap

}

std::swap(edges[left], edges[l]);//Move the privot to the middle

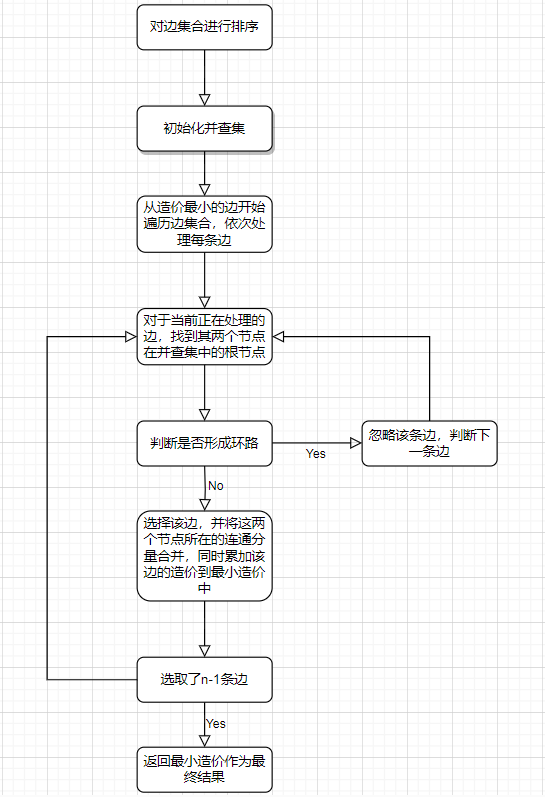
quicksort(edges, left, l - 1);//Go into the left segment

quicksort(edges, l + 1, right);//Go into the right segment

### }

## 3.2 Kruskal功能的实现

### 3.2.1 Kruskal功能流程图



### 3.2.2 Kruskal功能核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:kruskal

Function:To use the kruskal to calculate the minCost

Input Parameters:array two index amount of nodes and amount of edges

Return Value:result

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int kruskal(Edge\* edges, int n, int m)

{

quicksort(edges, 0, m - 1); // Using quicksort

init(n);//Initialize the parent array

int minCost = 0;

for (int i = 0, j = 0; i < m && j < n - 1; ++i) {

int u = edges[i].u;//Temporary u

int v = edges[i].v;//Temporary v

int cost = edges[i].cost;//Temporary cost

//If the root of u and v are not same(Connecting them will not lead to a loop)

if (find(u) != find(v)) {

unite(u, v);//Connect(Make v the father of u)

minCost += cost;//Add the cost

++j;//Add an existing edge(which will not be greater than n(nodes) minus one

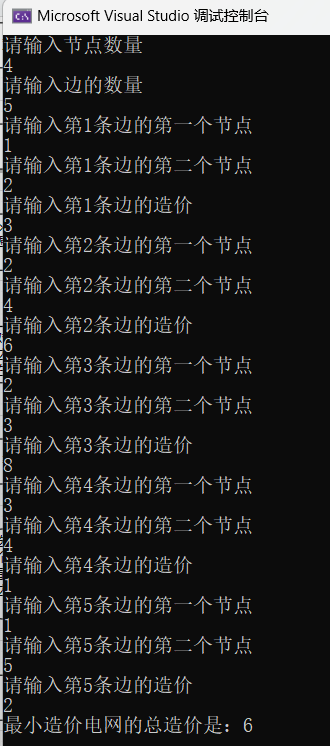
//Due to the rules of Minimum Spanning Tree

}

}

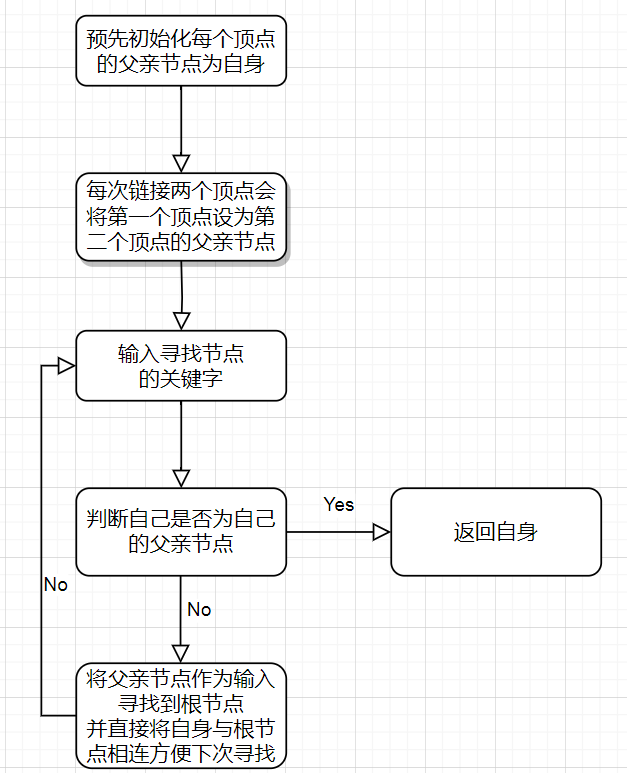
return minCost;

### }3.2.3 Kruskal功能截屏示例



## 3.3 find功能的实现

### 3.3.1 find功能流程图



### 3.3.2 find功能核心代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:find

Function:To find the root of the nodes

Input Parameters:the nodes name(int)

Return Value:the direct father/root of the nodes

Documentation：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int find(int x)

{

if (parent[x] == x)

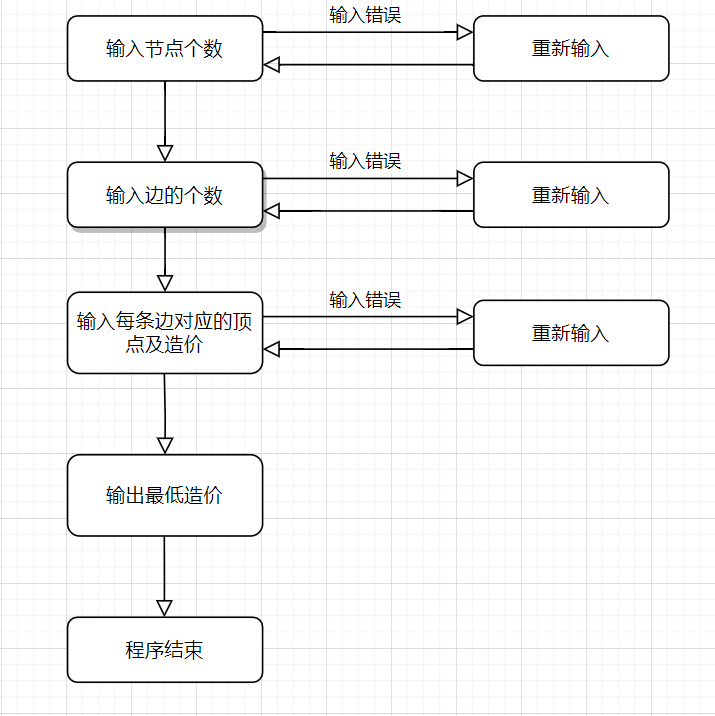
return x;

return parent[x] = find(parent[x]);//Path Compression,quicker when next finding

### }

## 3.4 总体系统的实现

### 3.4.1 总体系统流程图



### 3.4.2 总体系统核心代码

int n, m;//N is the amount of nodes,m is the amount of edge

std::cout << "请输入节点数量" << std::endl;

n = input<int>(0, MAX\_N);

std::cout << "请输入边的数量" << std::endl;

m = input<int>(0, MAX\_M);

//Initialize the edge

for (int i = 0; i < m; ++i) {

std::cout << "请输入第" << i + 1 << "条边的第一个节点" << std::endl;

edges[i].u = input<int>(0, MAX\_N);

std::cout << "请输入第" << i + 1 << "条边的第二个节点" << std::endl;

edges[i].v = input<int>(0, MAX\_N);

std::cout << "请输入第" << i + 1 << "条边的造价" << std::endl;

edges[i].cost = input<int>(0, 1000000);

}

int result = kruskal(edges, n, m);

std::cout << "最小造价电网的总造价是：" << result << std::endl;

### return 0;

### 3.4.3 总体系统截屏示例

