项目说明文档

数据结构课程设计

——电网造价模拟系统

作 者 姓 名： 张翔

学 号： 2352985

指 导 教 师： 张颖

学院、 专业： 计算机科学与技术学院 软件工程

同济大学

Tongji University

二〇二四 年 十二 月 七 日

# 1项目分析

## 1.1项目背景分析

随着城市化进程的加快，城市电网的建设成为了一个重要议题。在城市中，小区之间的电网建设需要优化，以确保电力的有效分配和经济效率。传统的电网规划方法往往忽略了成本效益的优化，从而导致不必要的开销。因此，开发一套电网建设造价模拟系统，以找到一种最低成本的电网连接方案，显得尤为重要。

## 1.2项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)设计一个电网建设造价模拟系统，能够输入多个小区间的电网连接成本；

(2)使用算法优化这些连接，以达到总成本最低，同时保证每个小区间的电网连通；

(3)提供友好的用户界面，方便用户输入数据和查看结果；

(4)系统需要具备良好的稳定性和安全性，能够处理非法输入等异常情况。

## 1.3项目功能分析

本项目旨在通过使用Prim算法建立最小生成树，并考虑用户界面设计，实现电网建设造价模拟系统。下面对项目的功能进行详细分析。

### 1.3.1建立最小生成树功能

本项目使用Prim算法实现得到最小生成树。Prim算法是一种用于在带权无向图中构建最小生成树的算法。最小生成树是指覆盖图中所有顶点并使边的总权重最小的树形结构。Prim算法特别适用于稠密图，即大多数顶点彼此之间都有边连接。

通过Prim算法，程序将计算出最小成本的电网构建方案。该算法可以有效处理复杂网络，找到连接所有小区所需的最低成本。

### 1.3.2异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

# 2项目设计

## 2.1数据结构设计

基于项目分析，在使用Prim算法建立最小生成树的过程中，项目采用邻接矩阵的数据结构来表示图，主要基于以下几个考虑：

(1)邻接矩阵适用于稠密图的场景，即图中大部分顶点之间都有边相连，类似于本项目中的城市电网场景；

(2)便于实现Prim算法中的关键操作，如查找最小权重的边；

(3)提高算法的运行效率，尤其是在处理大量节点时；

(4)简化算法的实现，使代码更加易于理解和维护。

## 2.2 MyUndirectedGraph类的设计

### 2.2.1概述

MyUndirectedGraph是一个模板类，用于表示无向图。它支持基本的图操作，如添加顶点和边、查找边的存在等。类中包括存储顶点信息的数组vertice和存储图的邻接矩阵或邻接表的指针 graph。此外，它还提供了一个用于查找顶点索引的辅助函数findVertexIndex。

该类还支持最小生成树（MST）的计算，通过primMST函数实现 Prim 算法来计算以某个顶点为起点的最小生成树，并通过 printMST 输出计算结果。最小生成树的父节点信息保存在 mstParent 数组中，并通过 mstComputed 标志指示是否已计算出最小生成树。getVertexCount和getEdgeCount方法分别返回图中当前的顶点数和边数，addVertex和addEdge方法用于向图中添加顶点和边，findEdge方法用于检查两顶点间是否存在边。

该类适用于处理无向图，并能高效计算和输出最小生成树。

### 2.2.2类定义

template <typename Type>

class MyUndirectedGraph

{

private:

int maxVertices;

int vertexCount;

int edgeCount;

Type\* vertices;

Edge\*\* graph;

int findVertexIndex(const Type& vertex) const;

/\* MST storage \*/

int\* mstParent;

bool mstComputed;

public:

// 无向图的基本操作

MyUndirectedGraph(int \_maxVertices);

~MyUndirectedGraph();

int getVertexCount() { return vertexCount; }

int getEdgeCount() { return edgeCount; }

bool addVertex(const Type& vertex);

bool addEdge(const Type& vertexA, const Type& vertexB, int weight);

bool findEdge(const Type& vertexA, const Type& vertexB);

// 最小生成树

bool isComputed() { return mstComputed; }

void primMST(const Type& vertex);

void printMST();

};

## 2.3 PowerGrid类的实现

### 2.3.1概述

PowerGrid是一个用于模拟城市电网连接的类，内部封装了一个MyUndirectedGraph<char>来表示小区与电网之间的连接。该类提供了创建顶点和边、构建最小生成树以及打印最小生成树的功能。通过 createGridVertices方法，可以动态添加小区节点；createGridEdges方法用于添加小区之间的电网线路及其造价信息；constructMinimumSpanningTree 方法利用最小生成树算法计算总造价最低的电网连接方案；printMinimumSpanningTree 方法用于输出最优连接方案。类的核心操作可以通过 selectOption 方法交互完成，便于动态执行电网规划和分析。

### 2.3.2类定义

class PowerGrid

{

private:

MyUndirectedGraph<char> grid;

public:

PowerGrid(int \_num) : grid(\_num) {}

~PowerGrid() {}

bool selectOption();

void createGridVertices();

void createGridEdges();

void constructMinimumSpanningTree();

void printMinimumSpanningTree();

};

## 2.4项目主题架构设计

|  |
| --- |
|  |

图2.3.1 项目主体架构设计流程图

# 3项目功能实现

## 3.1项目主体架构实现

### 3.1.1项目主体架构实现思路

该程序实现了一个基于命令行的电网造价模拟系统，使用PowerGrid类来管理电网的节点和边并计算最小生成树，实现具体思路如下:

主函数通过清晰的菜单界面提供了五项功能：创建电网节点、添加电网边、构造最小生成树、显示最小生成树，以及退出系统。

首先，创建一个PowerGrid对象，并设置最大支持节点数。

随后，程序进入循环，通过调用selectOption方法根据用户输入执行对应的功能，如创建节点或边、计算最小生成树等。当用户选择退出时，循环终止并打印退出信息。

程序逻辑简洁清晰，通过 PowerGrid 类封装了所有与电网构建和计算相关的操作，确保主函数只需负责菜单展示和用户交互。

### 3.1.2项目主体架构核心代码

int main()

{

std::cout << "+--------------------------------------+" << std::endl;

std::cout << "| 电网造价模拟系统 |" << std::endl;

std::cout << "| Power Grid Cost Simulation System |" << std::endl;

std::cout << "+--------------------------------------+" << std::endl;

std::cout << "| [1] --- 创建电网节点 |" << std::endl;

std::cout << "| [2] --- 添加电网的边 |" << std::endl;

std::cout << "| [3] --- 构造最小生成树 |" << std::endl;

std::cout << "| [4] --- 显示最小生成树 |" << std::endl;

std::cout << "| [5] --- 退出系统 |" << std::endl;

std::cout << "+--------------------------------------+" << std::endl << std::endl;

PowerGrid powergrid(MAX\_VERTICES);

while (powergrid.selectOption());

std::cout << ">>> 已成功退出电网造价模拟系统" << std::endl;

return 0;

}

## 3.2建立最小生成树功能实现

### 3.2.1建立最小生成树功能实现思路

该函数通过实现 Prim算法生成无向图的最小生成树（MST）。它从指定的起始顶点开始，使用布尔数组visited记录哪些顶点已加入MST，使用整数数组minWeight保存从已加入的 MST 顶点到未加入顶点的最小边权值。首先初始化所有顶点的权值为无穷大，并设置起始顶点的权值为 0。在每次迭代中，从未加入MST的顶点中选择权值最小的顶点，将其加入MST，并更新与之直接相连的其他顶点的最小边权值和父节点。循环执行vertexCount次后，若所有顶点都能被访问，则生成了 MST；否则报告图不连通。最后释放动态分配的内存资源并标记MST已计算完成。

### 3.2.2建立最小生成树功能核心代码

template <typename Type>

void MyUndirectedGraph<Type>::primMST(const Type& startVertex)

{

/\* whether the vertex has been added to the MST \*/

bool\* visited = new(std::nothrow) bool[vertexCount];

if (visited == nullptr) {

std::cerr << "Error: Memory allocation failed." << std::endl;

exit(-1);

}

for (int i = 0; i < vertexCount; i++)

visited[i] = false;

/\* The minimum weight edge from each vertex to the MST \*/

int\* minWeight = new (std::nothrow)int[vertexCount];

if (minWeight == nullptr) {

std::cerr << "Error: Memory allocation failed." << std::endl;

exit(-1);

}

/\* Initialize all vertices \*/

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

minWeight[i] = INT\_MAX;

}

/\* Find the starting node \*/

int startIndex = findVertexIndex(startVertex);

if (startIndex == -1) {

std::cerr << "Error: Starting vertex not found in the graph." << std::endl;

/\* Release the dynamic array and exit \*/

delete[] visited;

delete[] minWeight;

return;

}

minWeight[startIndex] = 0;

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

int u = -1;

/\* Find the vertex with the smallest weight that is not currently visited \*/

for (int j = 0; j < vertexCount; j++) {

if (!visited[j] && (u == -1 || minWeight[j] < minWeight[u])) {

u = j;

}

}

if (u == -1) {

std::cerr << "Graph is disconnected; no MST exists." << std::endl;

delete[] visited;

delete[] minWeight;

return;

}

visited[u] = true;

for (int v = 0; v < vertexCount; v++) {

if (graph[u][v].exist && !visited[v] && graph[u][v].weight < minWeight[v]) {

minWeight[v] = graph[u][v].weight;

mstParent[v] = u;

}

}

}

mstComputed = true;

delete[] visited;

delete[] minWeight;

}

## 3.3异常处理功能的实现

### 3.3.1动态内存申请失败的异常处理

在进行动态内存申请时，程序使用new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1)向标准错误流std::cerr输出一条错误消息"Error: Memory allocation failed."；

(2)调用exit函数，返回错误码-1，用于指示内存分配错误，并导致程序退出。

### 3.3.2输入非法的异常处理

程序通过调用inputInteger函数输入电网节点个数和电网节点之间的距离。inputInteger函数用于获取用户输入的整数，同时限制输入必须在指定的范围内，函数的代码如下：

int inputInteger(int lowerLimit, int upperLimit, const char\* prompt)

{

std::cout << ">>> " << "请输入" << prompt << " 整数范围: [" << lowerLimit << "~" << upperLimit << "]: ";

int input;

while (true) {

std::cin >> input;

if (std::cin.good() && input >= lowerLimit && input <= upperLimit) {

std::cin.clear();

std::cin.ignore(INT\_MAX, '\n');

std::cout << std::endl;

return input;

}

else {

std::cerr << ">>> " << prompt << "输入不合法，请重新输入！" << std::endl;

std::cin.clear();

std::cin.ignore(INT\_MAX, '\n');

}

}

}

在createGridVertices函数中，通过输入节点数量的范围检查，确保用户输入在合理范围内（2 至最大节点数）。每次尝试添加节点时，会调用grid.addVertex(vertex)验证节点的合法性。若添加失败（如节点重复或图已满），会提示用户 节点 X 已存在或无法添加！，并通过 i-- 强制重新输入。类似地，在createGridEdges函数中，首先检查是否有足够的节点以构建边。如果没有节点，直接提示用户 请先添加节点！ 并提前退出函数在输入边数量时，使用inputInteger限制其在合理范围内，并逐条添加边。添加过程中，调用 grid.addEdge(vertexA, vertexB, weight) 验证边的合法性，若失败则打印错误信息并重新输入，确保最终的输入无误。

# 4项目测试

## 4.1输入电网节点个数功能测试

|  |
| --- |
|  |

4.1.1输入电网节点个数功能测试示例

## 4.2输入任意两个电网节点之间的距离功能测试

|  |
| --- |
|  |

4.2.1输入电网节点之间的距离功能测试示例

## 4.3建立最小生成树功能测试

|  |
| --- |
|  |

4.3.1建立最小生成树功能测试示例

## 4.4项目退出测试

|  |
| --- |
|  |

4.4.1项目退出功能测试示例

# 5集成开发环境与编译运行环境

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Release模式)