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　林继申

学　　号　2250758

指 导 教 师　张　颖

学 院 专 业　软件学院　软件工程

同济大学 TONGJI UNIVERSITY

二〇二三 年 十二 月 十三 日

目录

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 Prim 最小生成树功能

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。

Prim 算法特别适用于本项目，原因如下：

(1)Prim 算法能够确保找到全局最优的最小生成树，即总成本最低的电网构建方案；

(2)相比其他算法如 Kruskal 算法，Prim 算法在处理稠密图时更加高效；

(3)该算法易于与邻接矩阵结合，提高了整个项目的运行效率。

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 MinimumSpanningTree 类的设计

2.2.1 概述

MinimumSpanningTree 类是电网建设造价模拟系统的核心组成部分，专门用于构建和处理最小生成树（MST），从而实现电网建设的成本最优化。该类采用 Prim 算法，适用于模拟城市小区之间的电网连接，以寻找连接所有小区的最低成本方案。它提供了一系列功能，包括电网结构的初始化、权重设置、最小生成树的构建和打印，以及异常处理机制。通过这个类，用户可以有效地模拟和分析给定电网结构的最优连接路径，从而在确保电网互通的前提下，最小化建设成本。

2.2.2 类定义

```
class MinimumSpanningTree {
private:
    int vertex;
    int** graph;
    int* parent;
    int* key;
    bool* mstSet;
public:
```

```
    MinimumSpanningTree(int V);
    ~MinimumSpanningTree();
    int minKey(void);
    void printMST(int startVertex);
    void primMST(int startVertex);
    void setWeight(int src, int dst, int weight);
};
```

2.2.3 私有数据成员

int vertex：顶点数量，代表电网中的小区数

int** graph：二维数组，用于存储电网中各小区间的连接成本

int* parent：父节点数组，记录最小生成树中每个顶点的父顶点

int* key：关键值数组，用于 Prim 算法中选取最小权重边

bool* mstSet：布尔数组，标记顶点是否已包含在最小生成树中

2.2.4 构造函数

MinimumSpanningTree(int V);

构造函数，接受顶点数量作为参数，初始化图结构和相关数组。

2.2.5 析构函数

~MinimumSpanningTree();

析构函数，释放动态分配的内存资源。

2.2.6 公有成员函数

int minKey(void);

返回尚未包含在 MST 中且权值最小的顶点的索引。

void printMST(int startVertex);

打印从指定起始顶点生成的最小生成树。

void primMST(int startVertex);

实现 Prim 算法，从指定的起始顶点生成最小生成树。

void setWeight(int src, int dst, int weight);

设置图中两个顶点间的边权重。

2.3 项目主体架构设计

项目主体架构设计为：

(1)创建电网；

(2)输入电网节点个数；

(3)输入任意两个电网节点之间的距离；

(4)输入建立最小生成树的起始节点；

(5)建立 Prim 最小生成树；

(6)打印 Prim 最小生成树；

(7)退出程序。

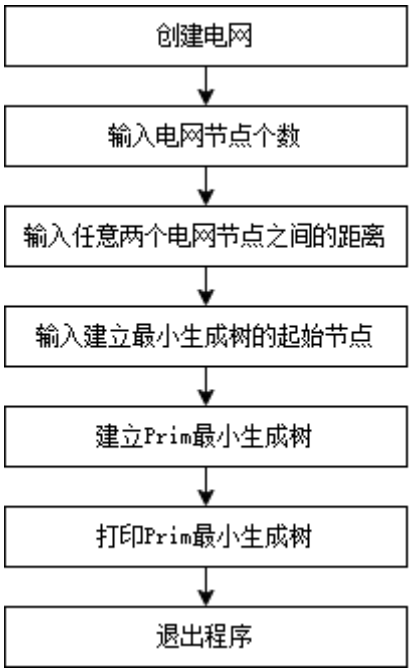

图 2.3.1 项目主体架构设计流程图

3 项目功能实现

3.1 项目主体架构的实现

3.1.1 项目主体架构实现思路

项目主体架构实现思路为：

(1)创建电网：在程序的开头，通过打印欢迎信息向用户介绍系统，为用户创建电网的过程提供指引；

(2)输入电网节点个数：使用 `inputInteger` 函数提示用户输入电网的节点个数。这个函数确保用户输入的是一个有效的整数，并且在指定的范围内。用户输入的数值决定了电网中的小区数量；

(3)输入任意两个电网节点之间的距离：进入一个双层循环，每次循环调用 `inputInteger` 函数来输入两个节点间的距离或成本。这一步骤对于每一对节点

都会重复，直到所有可能的节点对的距离都被输入。距离数据存储在 MinimumSpanningTree 类的实例中；

(4)输入建立最小生成树的起始节点：通过 inputStartVertex 函数获取用户输入的起始节点。这个函数处理用户的输入，确保它是一个有效的起始点；

(5)建立 Prim 最小生成树：调用 MinimumSpanningTree 类的 primMST 方法，使用 Prim 算法从指定的起始节点开始构建最小生成树。该方法负责处理整个图结构，并找出总成本最低的连接方式；

(6)打印 Prim 最小生成树：printMST 方法被调用以打印出构建的最小生成树。这个步骤在屏幕上展示了每个节点间的连接及其相应的成本，为用户提供了一个清晰的视觉表示；

(7)退出程序。

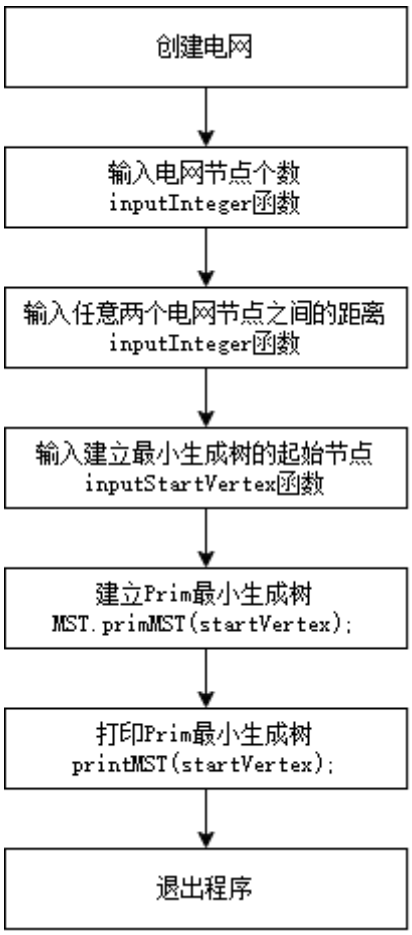

图 3.1.1.1 项目主体架构实现流程图

3.1.2 项目主体架构核心代码

```
int main()
{
    /* System entry prompt */
    std::cout << "+-------------------------------------------------
```

```
-+" << std::endl;
        std::cout << "|                电网建设造价模拟系统                |"
<< std::endl;
        std::cout << "|  Power Grid Construction Cost Simulation System
|" << std::endl;
        std::cout << "+-----------------------------------------------
-+" << std::endl;

        /* Establish power grid */
        std::cout << std::endl << ">>> 请创建电网" << std::endl <<
std::endl;
        int vertex = inputInteger(2, 'Z' - 'A' + 1, "电网节点个数");
        MinimumSpanningTree MST(vertex);
        std::cout << std::endl << ">>> 电网节点 A";
        for (int i = 1; i < vertex; i++)
            std::cout << "、" << static_cast<char>(i + 'A');
        std::cout << " 创建成功" << std::endl;

        /* Input the distance between any two power grid nodes */
        std::cout << std::endl << ">>> 请输入任意两个电网节点之间的距离" <<
std::endl << std::endl;
        for (int i = 0; i < vertex; i++)
            for (int j = i + 1; j < vertex; j++) {
                char tmp[64];
                sprintf(tmp, "电网节点 %c 和 %c 之间的距离", i + 'A', j +
'A');
                MST.setWeight(i, j, inputInteger(1, SHRT_MAX, tmp));
            }

        /* Generate minimum spanning tree using Prim algorithm */
        int startVertex = inputStartVertex(vertex);
        std::cout << ">>> 建立 Prim 最小生成树:" << std::endl << std::endl;
        MST.primMST(startVertex);

        /* Wait for enter to quit */
        std::cout << std::endl << "Press Enter to Quit" << std::endl;
        while (_getch() != '\r')
            continue;

        /* Program ends */
        return 0;
    }
```

3.1.3 项目主体架构示例

图 3.1.3.1 项目主体架构示例

3.2 建立 Prim 最小生成树功能的实现

3.2.1 建立 Prim 最小生成树功能实现思路

建立 Prim 最小生成树功能的函数为 MinimumSpanningTree 类的成员函数 primMST，建立 Prim 最小生成树功能实现的思路为：

(1)初始化关键数据结构：在 primMST 函数开始时，首先初始化关键的数据结构，包括 key 数组、parent 数组和 mstSet 数组。key 数组用于存储到达每个顶点的最小权重边的权重，parent 数组用于记录最小生成树中每个顶点的父顶点,而 mstSet 数组用于标记每个顶点是否已经被包含在当前的最小生成树中；

(2)选择起始顶点：将起始顶点的 key 值设置为 0，以确保它被选为最小生成树的第一个顶点。同时，将其父顶点设置为-1，表示它没有父顶点；

(3)构建最小生成树：对于电网中的每个顶点，执行以下步骤：

①调用 minKey 函数选择一个权重最小且尚未被包含在最小生成树中的顶点；

②将这个顶点标记为已包含在最小生成树中；

③遍历所有顶点，并更新与该顶点相连的顶点的 key 值和 parent 值。如果某个相连顶点尚未包含在最小生成树中且其连接边的权重小于当前的 key 值，则更新该顶点的 key 值和 parent 值；

(4)最小生成树结果输出：使用 printMST 方法打印最终的最小生成树。这一步骤通过遍历 parent 数组来实现，展示从起始顶点到其他所有顶点的最低成本路径。

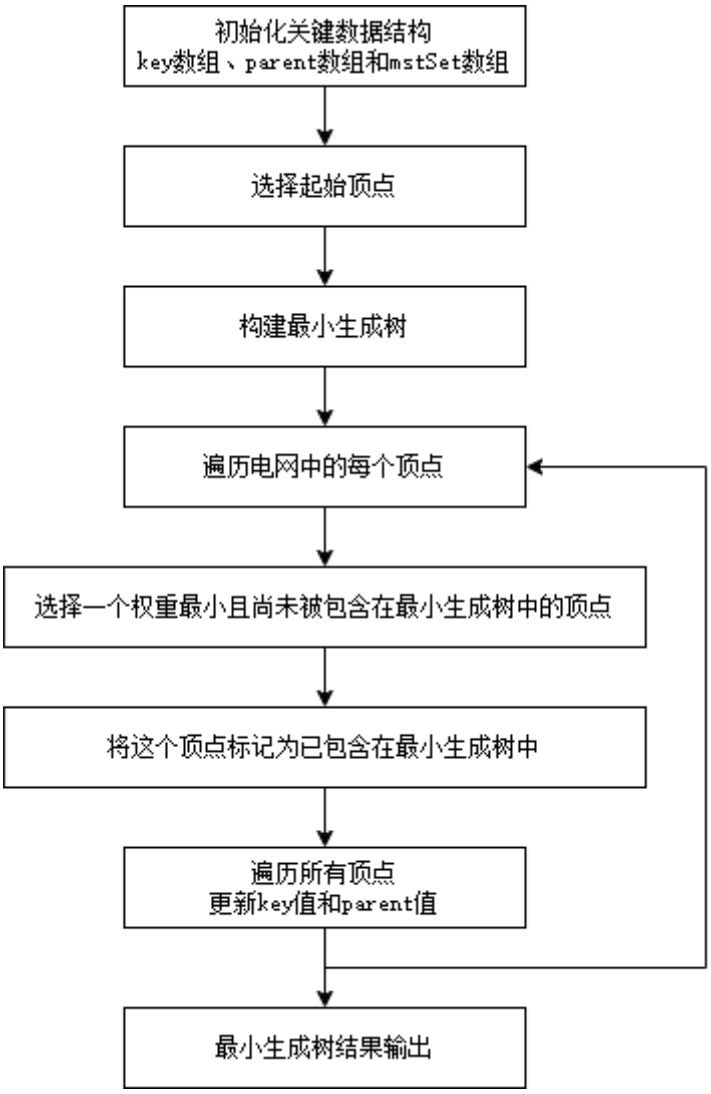

图 3.2.1.1 建立 Prim 最小生成树功能实现流程图

3.2.2 建立 Prim 最小生成树功能核心代码

```
int MinimumSpanningTree::minKey(void)
{
    int minVal = INF, minIndex = -1;
    for (int i = 0; i < vertex; i++) {
        if (!mstSet[i] && key[i] < minVal) {
```

```
            minVal = key[i];
            minIndex = i;
        }
    }
    return minIndex;
}

void MinimumSpanningTree::printMST(int startVertex)
{
    for (int i = startVertex + 1; i < vertex + startVertex; i++)
        std::cout << static_cast<char>(parent[i % vertex] + 'A') << "
--(" << graph[i % vertex][parent[i % vertex]] << ")--> " <<
static_cast<char>(i % vertex + 'A') << std::endl;
}

void MinimumSpanningTree::primMST(int startVertex)
{
    /* Initialize all keys as infinite and mstSet as false */
    for (int i = 0; i < vertex; i++) {
        key[i] = INF;
        mstSet[i] = false;
    }

    /* Initialize the starting vertex in MST */
    key[startVertex] = 0;
    parent[startVertex] = -1;

    /* Prim algorithm */
    for (int count = 0; count < vertex; count++) {
        int u = minKey();
        mstSet[u] = true; // Add the picked vertex to the MST Set
        for (int v = 0; v < vertex; v++)
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
    }

    /* Print the constructed minimum spanning tree (MST) */
    printMST(startVertex);
}
```

3.2.3 建立 Prim 最小生成树功能示例

建立 Prim 最小生成树功能示例见项目主体架构示例（见图 3.1.3.1）。

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 MinimumSpanningTree 类中私有数据成员等的动态内存申请时，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（NULL 或 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."，指出内存分配失败；

(2)调用 exit 函数，返回错误码 MEMORY_ALLOCATION_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

下面是动态内存申请的异常处理的一个代码示例：

```
graph = new(std::nothrow) int* [vertex];
if (graph == NULL) {
    std::cerr << "Error: Memory allocation failed." << std::endl;
    exit(MEMORY_ALLOCATION_ERROR);
}
for (int i = 0; i < vertex; i++) {
    graph[i] = new(std::nothrow) int[vertex];
    if (graph[i] == NULL) {
        std::cerr << "Error: Memory allocation failed." << std::endl;
        exit(MEMORY_ALLOCATION_ERROR);
    }
}
```

3.3.2 输入非法的异常处理

3.3.2.1 电网节点个数和电网节点之间的距离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)
{
    while (true) {
        std::cout << "请输入" << prompt << " [整数范围: " << lowerLimit
<< "~" << upperLimit << "]: ";
        double tempInput;
        std::cin >> tempInput;
        if (std::cin.good() && tempInput == static_cast<int>(tempInput)
&& tempInput >= lowerLimit && tempInput <= upperLimit) {
            std::cin.clear();
```

```
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                return static_cast<int>(tempInput);
            }
            else {
                std::cerr << std::endl << ">>> " << prompt << "输入不合法，
请重新输入" << prompt << "! " << std::endl << std::endl;
                std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            }
        }
    }
```

inputInteger 函数对输入非法的情况进行了处理，代码具体执行逻辑如下：

⑴进入一个无限循环，它会一直运行直到用户提供有效的输入；

⑵用户的输入被读取到 tempInput 变量中，这里采用 double 类型来接收输入以便后续检查；

⑶进行输入验证：std::cin.good()检查输入流的状态是否正常，确保没有发生数据类型输入错误，tempInput==static_cast<int>(tempInput)检查用户输入是否为整数，通过将其转换为整数再比较，tempInput>=lowerLimit 和 tempInput<=upperLimit 确保输入在指定的范围内；

⑷合法输入处理：如果用户提供了合法的输入，函数会清除输入流的错误状态，丢弃输入缓冲区中的任何剩余内容，然后返回转换后的整数值；

⑸非法输入处理：如果用户提供的输入不合法，函数会输出错误消息，清除输入流的错误状态，丢弃输入缓冲区中的内容，并继续循环以等待用户提供合法的输入。

3.3.2.2 建立最小生成树的起始节点输入非法的异常处理

建立最小生成树的起始节点输入非法的异常处理通过如下代码实现：

```
int inputStartVertex(int vertex)
{
    std::cout << std::endl << "请输入建立最小生成树的起始节点 [输入范围:
A~" << static_cast<char>(vertex + 'A' - 1) << "]: ";
    char optn;
    while (true) {
        optn = _getch();
        if (optn == 0 || optn == -32)
            optn = _getch();
        else if (optn >= 'A' && optn <= vertex + 'A' - 1) {
            std::cout << optn << std::endl << std::endl;
            return optn - 'A';
```

```
        }
    }
}
```

这段代码会一直等待用户输入，只有当用户输入有效的起始节点时，才会返回该起始节点的顶点值，表示用户输入的起始节点。如果用户输入无效字符，循环会继续等待用户提供有效的输入。这段代码的具体执行逻辑如下：

(1)显示输入提示信息；

(2)进入一个无限循环，以等待用户输入；

(3)用户输入的字符会被_getch()函数获取。这个函数通常用于在控制台应用程序中获取单个字符而不显示在屏幕上。用户的输入存储在变量 optn 中；

(4)代码检查用户输入是否是有效的起始节点字符或特殊的控制字符。如果用户按下无效的起始节点字符或特殊控制字符，它将不执行下面的操作；

(5)如果用户输入是有效的起始节点字符，它会在屏幕上显示用户输入的起始节点，并返回对应的顶点值；

(6)如果用户输入的不是有效的起始节点字符，代码将保持在循环中，继续等待有效的输入，这确保了只有在用户输入正确的起始节点时才会退出循环。

4 项目测试

4.1 输入电网节点个数功能测试

分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。

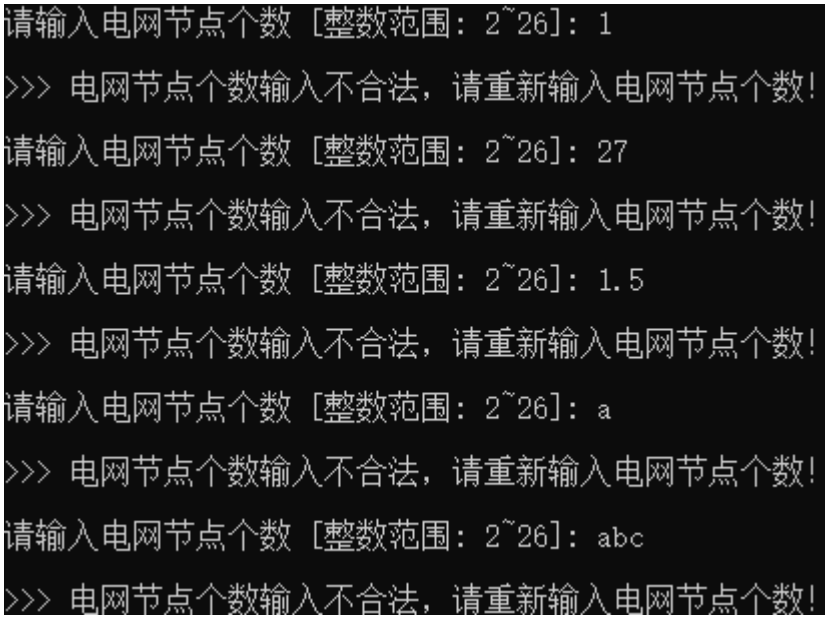

图 4.1.1 输入电网节点个数功能测试（输入非法）

当输入合法时，程序继续运行。

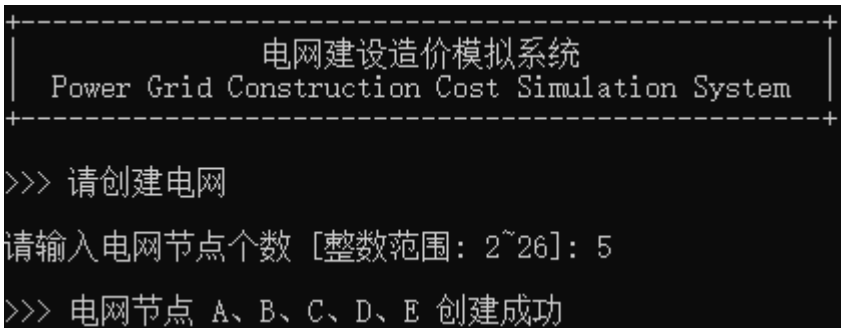

图 4.1.2 输入电网节点个数功能测试（输入合法）

4.2 输入任意两个电网节点之间的距离功能测试

分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。

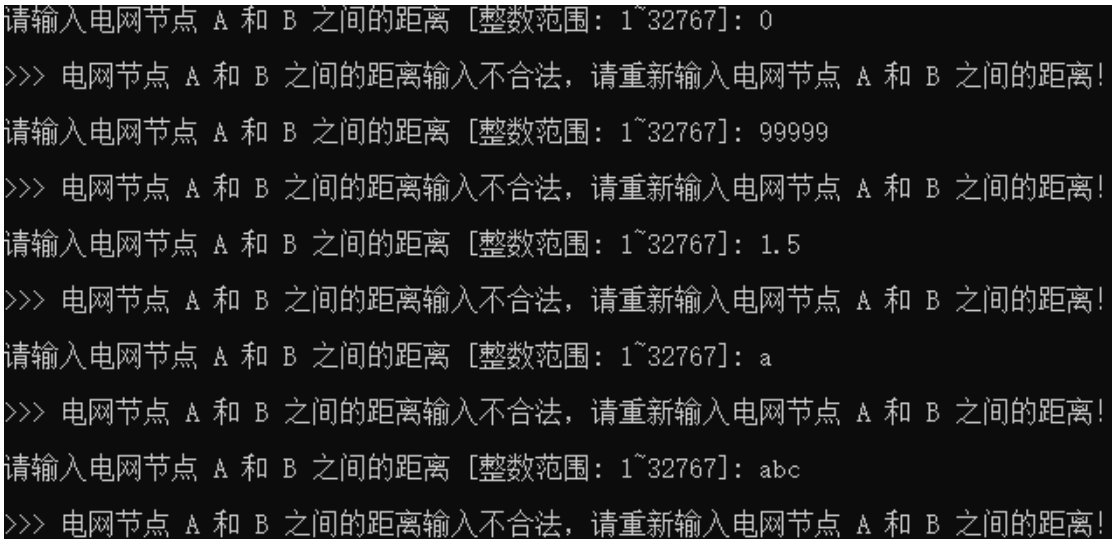

图 4.2.1 输入任意两个电网节点之间的距离功能测试（输入非法）

当输入合法时，程序继续运行。

图 4.2.2 输入任意两个电网节点之间的距离功能测试（输入合法）

4.3 建立 Prim 最小生成树功能测试

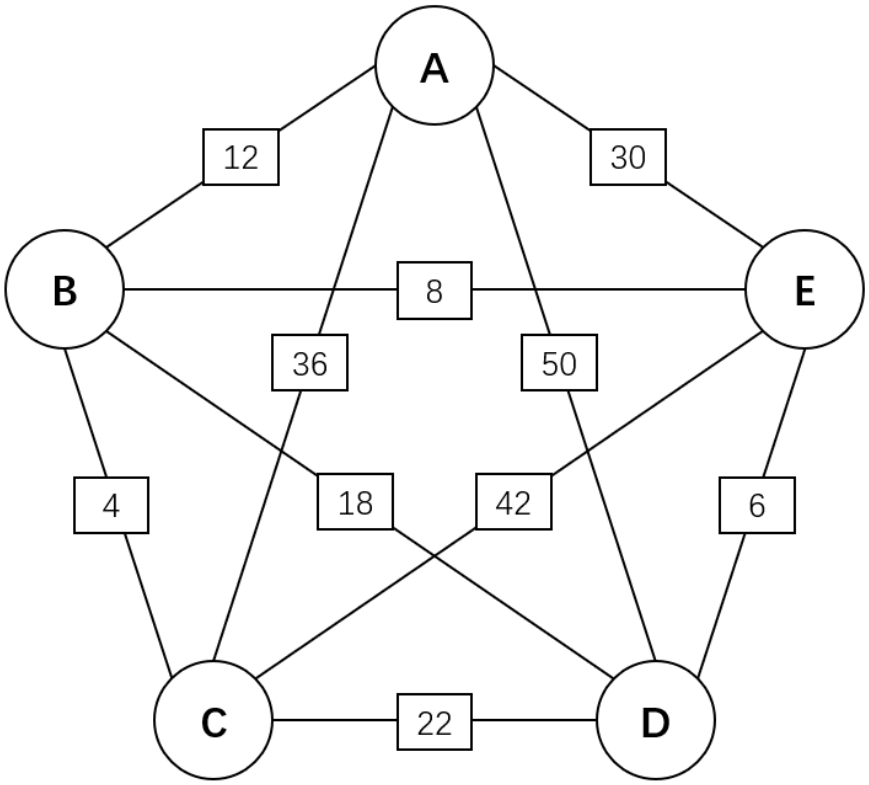

图 4.3.1 电网节点连接模拟图（数字表示线路成本）

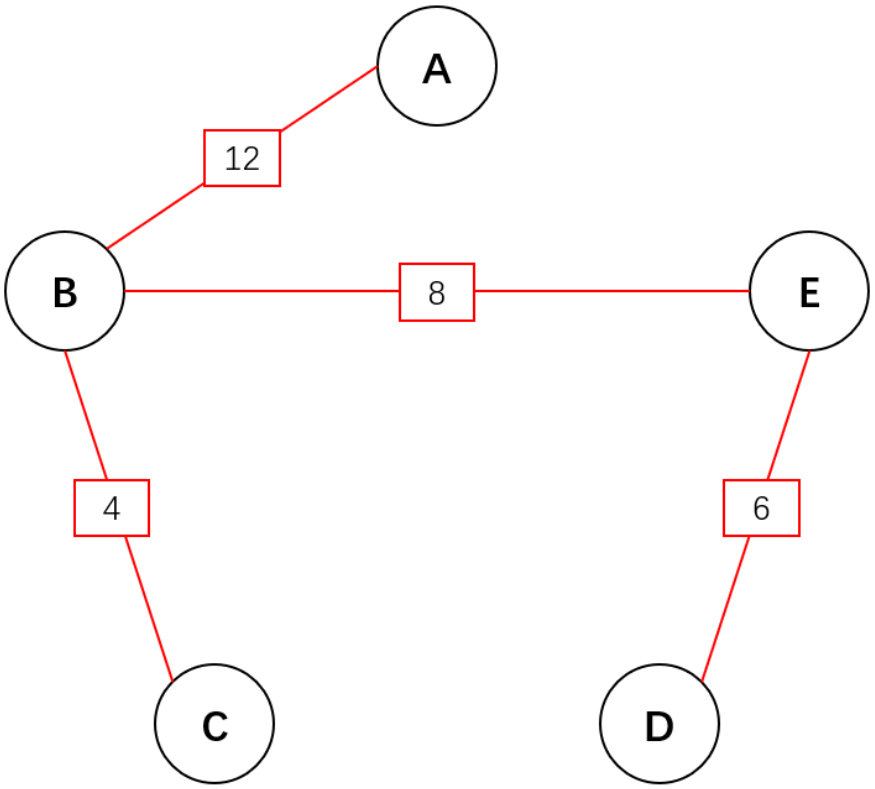

图 4.3.2 Prim 最小生成树示意图

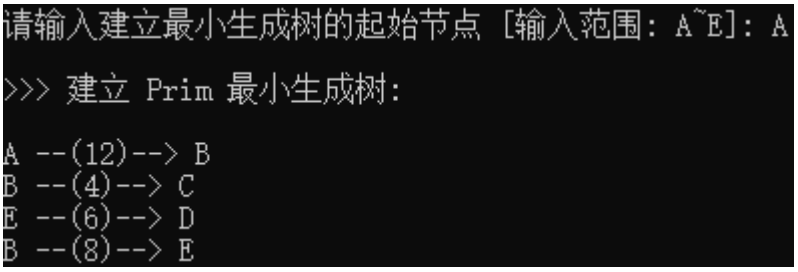

图 4.3.3 建立 Prim 最小生成树功能测试（起始节点为 A）

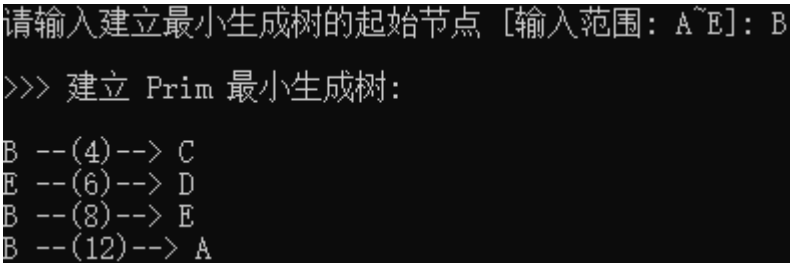

图 4.3.4 建立 Prim 最小生成树功能测试（起始节点为 B）

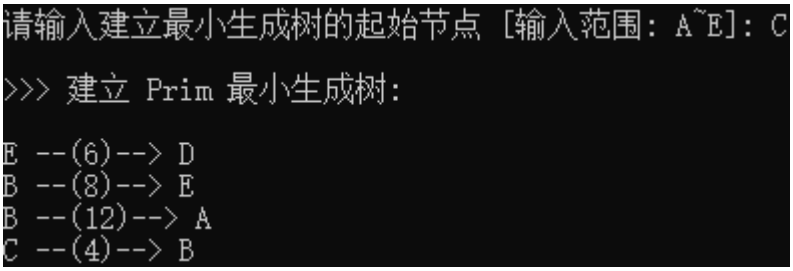

图 4.3.5 建立 Prim 最小生成树功能测试（起始节点为 C）

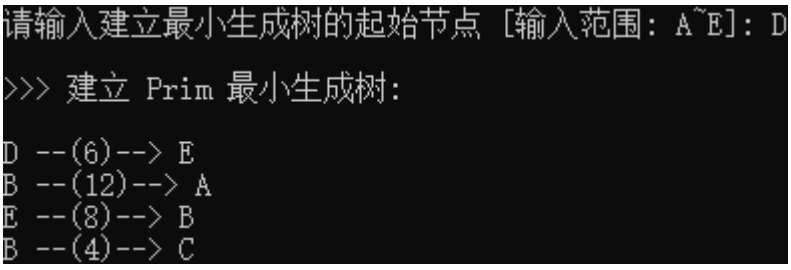

图 4.3.6 建立 Prim 最小生成树功能测试（起始节点为 D）

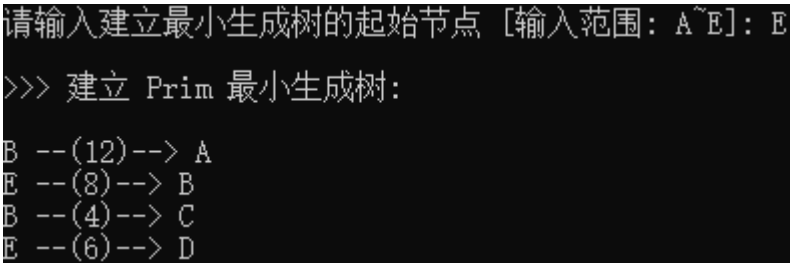

图 4.3.7 建立 Prim 最小生成树功能测试（起始节点为 E）

4.4 退出程序功能测试

为避免直接运行可执行文件在输入完成后会发生闪退的情况，本程序使用如下代码，创建了一个无限循环，等待用户按下回车键，以便退出程序。避免结果输出结束后程序迅速退出的情况。

```
/* Wait for enter to quit */
std::cout << std::endl << "Press Enter to Quit" << std::endl;
while (_getch() != '\r')
    continue;
```

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)

Windows 编译运行环境：本项目适用于 x86 架构和 x64 架构

Linux 系统：CentOS 7 x64

Linux 编译命令：

g++ '/root/桌面/Share_Folder/power_grid_construction_cost_simulation_system.cpp' -o '/root/桌面/Share_Folder/power_grid_construction_cost_simulation_system' -lncurses

Linux 运行命令：

'/root/桌面/Share_Folder/power_grid_construction_cost_simulation_system'

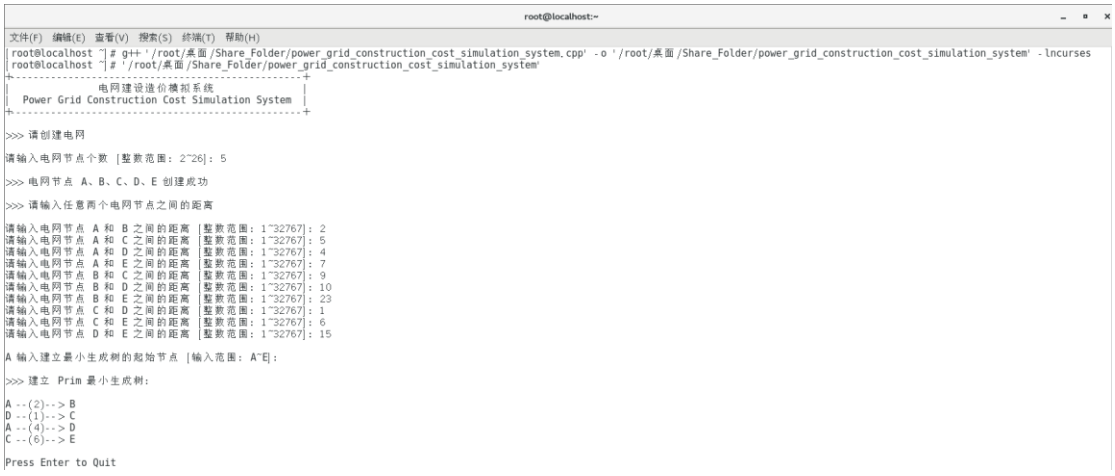

图 5.1 Linux 环境程序运行示例

本项目使用条件编译解决 Windows 系统和 Linux 系统编译环境的差异，示例

代码如下。

```
#ifdef _WIN32
#include <conio.h>
#elif __linux__
#include <ncurses.h>
#endif
```