同济大学
TONGJI UNIVERSITY

数据结构课程设计

项目说明文档

电网建设造价模拟系统

姓　　名：　　马小龙

学　　号：　　2353814

学　　院：计算机科学与技术学院（软件学院）

专　　业：　　软件工程

指导教师：　　张颖

二〇二三年十一月二十四日

目录

第 1 章 项目分析

1.1 项目背景分析

随着城市化的快速发展，电网建设在城市基础设施中扮演着至关重要的角色。特别是在现代大城市中，不同小区之间的电力需求日益增加，如何构建高效且经济的电网变得尤为重要。传统的电网建设方法往往侧重于保障电力供应的稳定性和可靠性，但在很多情况下忽视了成本控制和资源优化。这导致了不必要的电网建设投资，甚至可能增加长期运营的维护成本。

城市中的电网结构通常由多个小区组成，而这些小区之间通过电网线路连接，形成了一个复杂的网络系统。在这种网络中，每条线路都有一定的建设成本，如何以最小的总成本连接所有小区，同时确保每个小区之间能够相互供电，是实现电网经济高效构建的核心问题。

为了应对这一挑战，本项目旨在通过建立一个电网建设造价模拟系统，运用最优化算法来寻找最小总成本的电网连接方案。通过模拟不同小区之间的电网连接成本，系统能够帮助规划人员在保证电网连通性的前提下，选择出最优的电网建设方案，最大化降低不必要的开销。这不仅能有效节省建设成本，还能优化电网布局，提升电力资源的分配效率。

1.2 项目需求分析

本项目的主要目标是开发一个电网建设造价模拟系统，旨在帮助用户设计出最经济的电网连接方案。系统需要能够输入多个小区之间的电网连接成本，并通过合适的优化算法计算出最低成本的电网建设方案。用户可以根据该方案进行电网规划，并将其应用于实际的城市电网设计中。

用户能够输入不同小区以及小区之间的电网连接成本，并形成一个网络图结构；系统能够自动运行优化算法，计算出最低造价的电网连接方案，并输出连接的线路及其对应的费用；提供清晰、易操作的用户界面，支持用户查看和修改输入数据；系统能够在出现异常输入时提供相应的错误提示，并保证系统的稳定性。通过满足这些需求，系统能够为电网规划提供一个高效、准确、低成本的解决方案，避免传统电网设计中的冗余开销，帮助实现城市电力资源的优化配置。

1.3 项目功能分析

本项目旨在开发一套高效的电网建设造价模拟系统，以寻找最小成本的电网连接方案。系统支持用户输入小区信息，输入两个小区间的电网造价，生成并打印最优方案。具体方式如下：

1.3.1 创建电网顶点功能

允许根据输入小区数目输入一定数目的小区名称，输入格式为字符串类型。

1.3.2 添加电网的边功能

允许用户输入已有顶点之间的边（权值），要求使每两个顶点之间均有通路，已构成完全图。

1.3.3 构建最小生成树功能

根据已经存在的图，使用 Prim 算法，构建最小生成树，并存储路径。

1.3.4 显示最小生成树功能

将已经构建的最小生成树打印出来，即为最小成本的电网连接方案。

1.3.5 异常处理功能

程序对各种异常进行了基本处理，以提升系统稳定性。

第 2 章 项目设计

2.1 数据结构与算法设计

本项目主要使用了三种数据结构，包括图、string、以及散列表。

（一）使用图作为主要数据结构，用来储存顶点信息和边的信息，并借助 Prim 算法生成最小生成树，主要是由于以下方面原因：

1.图结构适合描述顶点和边的关系；图的顶点可以直接表示电网建设中的小区，每个顶点都能携带小区的相关信息（如名称、编号等）；图的边能够表示小区之间的电网线路，每条边可以附带权重，用于存储电网建设的成本。同时，图作为数据结构，便于扩展功能。

2.在电网建设场景中，小区之间可能有大量潜在的连接方式（边数接近 n×(n-1)/2），这是典型的稠密图，而 Prim 算法适合稠密图；Prim 算法可以通过邻接矩阵实现构建最小生成树，效率较高，能够快速找到最低成本的连通方案。同时，Prim 算法可以直接从任意顶点开始构建最小生成树。

（二）使用 String 来储存小区信息，主要是基于一下方面：

1.使用 string 类型，可以根据直接将小区具体信息作为顶点，避免了编号等操作；

2.string 类型具有良好的可读性和易用性，使得处理和展示小区信息变得直观。在代码中，string 的使用能够增强可维护性，方便后续的调试和修改。

3.string 类型的动态特性使得其在处理长度不确定的文本时非常灵活。小区信息的长度可能各异，string 能够自动管理内存，适应不同长度的内容，避免了固定大小数组带来的限制。

4.string 也能够方便地与其他数据结构进行结合。与散列表的结合使用，可以高效地管理图顶点的信息，确保信息的动态性和易操作性。

（三）使用散列表来存储和查找顶点信息，主要由于以下原因：

1.用户输入的顶点信息具有不确定性，即无法预测其数量，也很难根据输入的信息来将其转化为有序的索引来进行对于邻接矩阵等数组的操作；而是使用散列表，可以将顶点信息与输入顺序相关联（索引），通过某一值查找另外一个，为数组提供了便利；

2.散列表查找效率特别高效，在理想情况下，散列表的查找、插入和删除操作的时间复杂度为 O(1)，能够极大提升顶点信息的存取效率。

3. 散列表的实现和应用相对简单，并且可以灵活扩展功能。

2.2 结构体与类设计

2.2.1 最小生成树结点（MSTEdgeNode）设计

2.2.1.1 概述

MSTEdgeNode 用于储存信息，当前边的起点和终点，以及边的权值。

2.2.1.2 结构体定义

```
template<typename T>
struct MSTEdgeNode {
    T tail,head;
    double key;
    MSTEdgeNode();
    MSTEdgeNode(T src,T end,double key);
    bool operator<=(MSTEdgeNode<T>& R) const;
    bool operator<(MSTEdgeNode<T>& R) const;
    bool operator==(MSTEdgeNode<T>& R) const;
    bool operator!=(MSTEdgeNode<T>& R) const;
    bool operator>(MSTEdgeNode<T>& R) const;
    bool operator>=(MSTEdgeNode<T>& R) const;
};
```

2.2.1.3 数据成员

T tail: 当前边的起点

T head；当前边的终点；

double key; 当前边的权值；

2.2.2.4 构造函数

MSTEdgeNode();

构造函数，根据默认信息初始化数据域；

MSTEdgeNode(T src,T end,double key);

构造函数，根据具体信息初始化数据域；

2.2.1.5 成员函数

bool operator<=(MSTEdgeNode<T>& R) const;

bool operator<(MSTEdgeNode<T>& R) const;

bool operator==(MSTEdgeNode<T>& R) const ；

bool operator!=(MSTEdgeNode<T>& R) cons；

bool operator>(MSTEdgeNode<T>& R) const；

bool operator>=(MSTEdgeNode<T>& R) const；

重载比较运算符，使结构体可以满足各种比较运算；

2.2.2 最小生成树（MinSpanTree）设计

2.2.2.1 概述

该通用模板类 MinSpanTree 主要用于表示最小生成树，用来存储通过 Prim 算法得到的最小生成树顶点，该类支持插入、打印最小生成树等操作。

2.2.2.2 MinSpanTree 类定义

```
template<typename T>
class MinSpanTree {
    public:
    MinSpanTree();
    MinSpanTree(int size);
    ~MinSpanTree();
    void Initial(int size);
    void MakeEmpty();
    void Insert(const MSTEdgeNode<T>& item);
    void PrintEdges() const;
    bool BuildSuccess();
private:
    int maxSize , n;
    MSTEdgeNode<T>* edge_value;
};
```

2.2.2.3 私有数据成员

int maxSize: 最小生成树最大大小（容量）；

int n : 最小生成树当前大小

MSTEdgeNode<T>* edge_value; 用于存储最小生成树的结点

2.2.2.4 构造函数与析构函数

MinSpanTree()；

默认构造函数，创建一个空的最小生成树。

MinSpanTree(int size);

含参构造函数，根据顶点数目设置最小生成树大小（容量），根据大小初始化数据成员；

~MinSpanTree()；

析构函数，释放最小生成树的内存资源，包括所有数组的内存。

2.2.2.5 公有成员函数

void Initial(int size);

重新设定最小生成树大小，初始化数据成员

void MakeEmpty();

清除当前最小生成树数据资源；

void Insert(const MSTEdgeNode<T>& item);

向最小生成树中插入结点，以构建最小生成树;

void PrintEdges() const;

打印最小生成树,以及最小生成树花费；

bool BuildSuccess()；

简单检查最小生成树是否构建成功；

2.2.3 无向图设计

2.2.3.1 概述

Graph 主要用来存储图的相关信息，并根据这些信息借助 Prim 算法构建最小生成树。

2.2.3.2 无向图类定义

```
template<typename T>
class Graph {
public:
    Graph()；
    Graph(int sz);
    ~Graph()；
    void Initial(int sz);
    void MakeEmpty();
    void SetEdgeValue(T start, T end, double value);
    void SetDot(const T &dot);
```

```
        bool FindDot(const T &dot);
        bool Prim(T start);
        void DisplayMST() const；
        void InitialEdgeValue();
        void MTSInitial();
    private:
      int size, n;
      T *dots;
      double **edge_value;
      MinSpanTree<T> min_span_tree;
      Hash<T,int> hash_table;
    };
```

2.2.3.3 私有数据成员

int size; 图的最大大小，即顶点容量；

int n;图的当前大小，即当前顶点数目；

T *dots;用于存储顶点信息

double **edge_value;邻接矩阵，存储边的信息，包括权值及端点；

MinSpanTree<T> min_span_tree;最小生成树的一个对象

Hash<T,int> hash_table;散列表的一个对象

2.2.3.4 构造函数与析构函数

Graph()；

默认构造函数，创建一个空的无向图

Graph(int sz);

含参构造函数，根据顶点数目设置图的大小，根据大小初始化数据成员；

~Graph()；

析构函数，释放最小生成树的内存资源，包括所有数组的内存。

2.2.3.5 公有成员函数

void Initial(int sz);

重新设定无向图的大小，初始化数据成员

void MakeEmpty();

清除当前无向图数据资源；

void SetEdgeValue(T start, T end, double value);

向无向图中添加边（权值）

void SetDot(const T &dot);

向无向图中插入点

bool FindDot(const T &dot);

```
寻找点是否在无向图中，防止重复插入
bool Prim(T start);
使用 Prim 算法构建最小生成树
void DisplayMST() const；
打印最小生成树；
void InitialEdgeValue();
初始化邻接矩阵；
void MTSInitial();
初始化最小生成树
```

2.2.4 String 类设计

2.2.4.1 概述

String 为自定义数据类型，可以用来储存及处理字符串，例如姓名、性别和报考类别。

2.2.4.2 String 类定义

```
class String {
private:
    char* _str;
    size_t _str_len;
    size_t _str_cap;
public:
    static constexpr size_t npos = -1;
    typedef char* iterator;
    typedef const char* const_iterator;
    iterator begin();
    iterator end();
    const_iterator begin() const;
    const_iterator end() const;
    String(const char* str = "");
    String(const String& s);
   ~String();
    const char* C_str ()const;
    size_t Size()const;
    size_t Capacity()const;
    void Swap(String& s);
```

```
    void Reserve(size_t n);
    void Resize(size_t n, char ch = '\0');
    void PushBack(char ch);
    void Append(const char* str);
    void Insert(size_t pos, char ch, size_t n=1);
    void Insert(size_t pos, const char* str);
    void Erase(size_t pos, size_t len = npos);
    size_t Find(char ch, size_t pos = 0) const;
    size_t Find(const char* str, size_t pos = 0) const;
    String Substr(size_t pos = 0, size_t len = npos) const;
    void Clear();
    static int Memcmp(const void *dst,const void *src,size_t n) ;
    static int Strlen(const char* src);
    static char* Strstr(const char* str1, const char* str2);
    String& operator+=(char ch);
    String& operator+=(const char* str);
    bool operator<(const String& s) const;
    bool operator==(const String& s) const;
    bool operator<=(const String& s) const;
    bool operator>(const String& s) const;
    bool operator>=(const String& s) const;
    bool operator!=(const String& s) const;
    char& operator[](size_t pos);
    const char& operator[](size_t pos) const;
};
```

2.2.4.3 私有数据成员

char* _str: 存储字符串的字符数组

size_t _str_len: 当前长度

size_t _str_cap: 当前容量

2.2.4.4 公有数据成员

static constexpr size_t npos = -1: npos 常量表示无效位

2.2.4.5 构造函数与析构函数

String(const char* str = "");

构造函数，默认空字符串

String(const String& s);

拷贝构造函数

~String();

析构函数，释放字符串所占内存

2.2.4.6 公有成员函数

iterator begin();

获取迭代器开始位置。

iterator end();

获取迭代器结束位置。

const_iterator begin() const;

获取常量迭代器开始位置。

const_iterator end() const;

获取常量迭代器结束位置。

const char* C_str ()const;

获取当前字符串。

size_t Size()const;

获取字符串长度。

size_t Capacity()const;

获取当前容量。

void Swap(String& s);

交换字符串内容。

void Reserve(size_t n);

预留容量，扩展字符串容量。

void Resize(size_t n, char ch = '\0');

调整当前字符串大小。

void PushBack(char ch);

在字符串末尾添加字符。

void Append(const char* str);

在字符串末尾添加字符串。

void Insert(size_t pos, char ch, size_t n=1);

在指定位置插入 n 个字符。

void Insert(size_t pos, const char* str);

在指定位置插入字符串。

void Erase(size_t pos, size_t len = npos);

删除指定位置指定长度的字符串。

size_t Find(char ch, size_t pos = 0) const;

在字符串中查找指定字符位置。

size_t Find(const char* str, size_t pos = 0) const;

在字符串查找指定字符串位置。

String Substr(size_t pos = 0, size_t len = npos) const;

提取子字符串。

void Clear();

清空字符串。

static int Memcmp(const void *dst,const void *src,size_t n) ;

检查字符串长度是否相等，根据返回结果进行判断 1 为 dst 大于 src，-1 为 dst 小于 srcdst ，0 为等于 src。

static int Strlen(const char* src);

获取字符串长度。

static char* Strstr(const char* str1, const char* str2);

查找子字符串。

String& operator+=(char ch);

+=运算符重载，将新的字符放在字符串末尾。

String& operator+=(const char* str);

+=运算符重载，将新的字符串放在字符串末尾。

bool operator<(const String& s) const;

小于运算符重载，用于字符串大小比较。

bool operator==(const String& s) const;

等于运算符重载，用于字符串大小比较。

bool operator<=(const String& s) const;

小于等于运算符重载，用于字符串大小比较。

bool operator>(const String& s) const;

大于运算符重载，用于字符串大小比较。

bool operator>=(const String& s) const;

大于等于运算符重载，用于字符串大小比较。

bool operator!=(const String& s) const;

不等于运算符重载，用于字符串大小比较。

char& operator[](size_t pos);

下标运算符重载，用于查找指定位置字符。

const char& operator[](size_t pos) const;

下标运算符重载，用于查找指定位置字符。

String& operator=(const String& s);

赋值运算符重载，用于 String 类的赋值。

2.2.4.7 输入重载

inline std::istream& operator>>(std::istream& input, String& s)；

输入流操作符重载，规范 String 数据类型的输入

2.2.5 散列表类设计

2.2.5.1 概述

散列表主要用来实现索引和顶点信息的相互查找，提高查找效率。

2.2.5.2 散列表（Hash）类定义

```
template<typename Key, typename Value>
class Hash {
private:
    struct Node {
        Key key;
        Value value;
        Node *next;
        Node(const Key &k, const Value &v) : key(k), value(v), next(nullptr) {}
    };
    Node **buckets;
    size_t bucket_count;
    size_t size;
    size_t hash(const char key);
    size_t hash(const int key);
    size_t hash(const long key);
    size_t hash(const long long key);
    size_t hash_float(const float key);
    size_t hash(const double key);
```

```
    size_t hash(const unsigned int key);
    size_t hash(const unsigned long key);
    size_t hash(const unsigned long long key);
    size_t hash(const unsigned char key);
    size_t hash(const mine::String &key);
public:
    Hash(size_t bucket_count = 16);
    ~Hash();
    void make_empty();
    void insert(const Key &key, const Value &value);
    Value *find(const Key &key);
    void erase(const Key &key);
    size_t getSize() const;
    Value &operator[](const Key &key);
};
```

2.2.5.3 构造函数与析构函数

Hash(size_t bucket_count = 16);

构造函数，初始化数据成员，其中，桶的数目默认为 16，通过传入参数不同而变化；

~Hash();

析构函数，释放散列表内存资源，防止内存泄漏；

2.2.5.4 私有数据成员

```
struct Node {
    Key key;
    Value value;
    Node *next;
    Node(const Key &k, const Value &v) : key(k), value(v), next(nullptr) {}
};
```

散列表数据结点，用来存储散列表表的键值对数据，以及下一个结点的指针，通过链表的方式解决哈希冲突；

Node **buckets; 哈希桶数组，用于存储散列表数据结点；

size_t bucket_count; 哈希桶数量

size_t size; 当前存储的键值对数量

2.2.5.5 私有成员函数

size_t hash(const char key);

size_t hash(const int key);

size_t hash(const long key);

size_t hash(const long long key);

size_t hash_float(const float key);

size_t hash(const double key);

size_t hash(const unsigned int key);

size_t hash(const unsigned long key);

size_t hash(const unsigned long long key);

size_t hash(const unsigned char key);

size_t hash(const mine::String &key);

不同数据类型的哈希函数，根据键值计算对应桶的索引。

2.2.5.6 公有成员函数

void make_empty();

清空散列表数据，将其置为初始状态；

void insert(const Key &key, const Value &value);

插入键值对，当键值对存在时，不插入

Value *find(const Key &key);

在散列表中查找值，返回值的地址；若不存在，返回 nullptr；

void erase(const Key &key);

在散列表中删除给定键的键值对

size_t getSize() const;

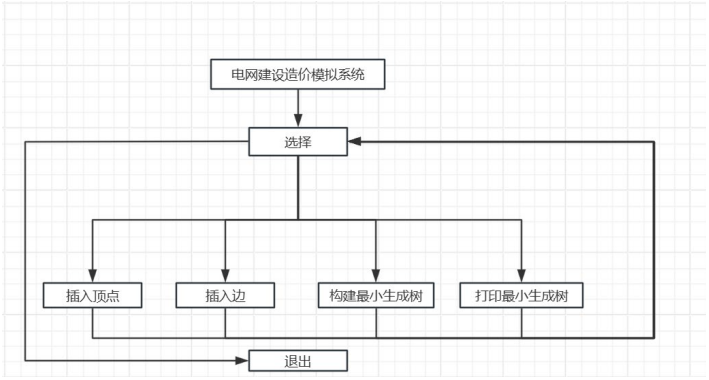
获取散列表大小（键值对数目）

Value &operator[](const Key &key);

重载索引操作符，查找键对应的值，如果键不存在则创建新节点，同时可以修改键对应的值。

2.3 项目框架设计

2.3.1 项目框架流程图



2.3.2 项目框架流程

1. 进入电网建设造价模拟系统，进入一个循环，选择 2~5 操作之一；
2. 输入顶点数目，并根据顶点数目输入顶点信息；
3. 输入任意两个顶点之间的边权值；
4. 构建最小生成树。
5. 显示最小生成树；
6. 若选择“退出”则退出循环与程序。

第 3 章 项目功能实现

3.1 项目主体架构

3.1.1 实现思路

1.进入 PowerGridConstructionCostSimulationSystem 函数以进入电网建设造价模拟系统。

2.进入循环，调用 Select 函数选择要进行地操作后，回到循环，根据选择的操作开始执行不同的函数，包括 CreateVertex 创建电网顶点、AddEdge 添加电网的边、ConstructMST 构建最小生成树、DisplayMST 显示最小生成树；直到选择退出则退出循环。

创建电网顶点：输入电网顶点数目，根据数目输入顶点信息，将顶点插入到图中；

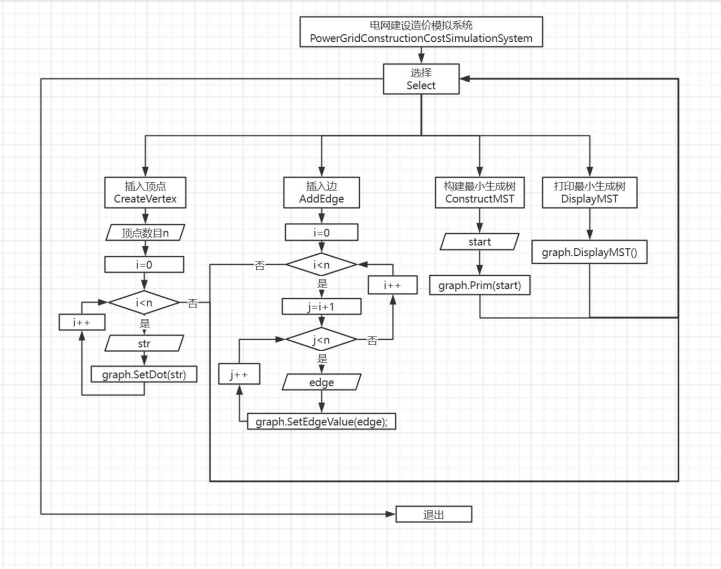
添加电网的边：根据输入的顶点，依次输入每两个顶点之间的边的权值，并插入的图中；

构建最小生成树：通过已有的顶点和边的信息，使用 Prim 算法构建最小生成树；

显示最小生成树：调用图的相关函数，打印最小生成树；

3.退出考试报名系统。

3.1.2 流程图



3.1.3 核心代码

主要逻辑：

```
Graph<mine::String> graph;
bool have_vertex = false,have_edge=false,have_prim=false;
Hash<int,mine::String> vertexes;
int n;
while(true) {
    char selection=Select();
    if(selection=='A')
        CreateVertex(graph,n,vertexes, have_vertex,have_edge,have_prim);
    else if(selection=='B')
        AddEdge(graph, n, vertexes, have_vertex, have_edge, have_prim);
    else if(selection=='C')
        ConstructMST(graph,n, vertexes,have_vertex, have_prim);
    else if(selection=='D')
        DisplayMST(graph,have_prim);
    else
        break;
}
```

创建电网顶点逻辑：

```
void CreateVertex(Graph<mine::String>& graph,int &n,Hash<int,
mine::String>& vertexes, bool& have_vertex,bool& have_edge,bool&
have_prim)
{
    have_prim =have_edge =have_vertex = false;
    double n_temp;
    while(true) {
        std::cout << "请输入顶点个数(至少为 2 个)：";
        std::cin>>n_temp;
        if(std::cin.fail()||n_temp!=static_cast<int>(n_temp)||n_temp<2) {
            std::cout << "输入非法，请重新输入！\n";
            ClearBuffer();
            continue;;
        }
```

```
            ClearBuffer();
            break;
        }
        n=static_cast<int>(n_temp);
        graph.Initial(n);//初始化图
        std::cout<<"请依次输入个顶点的名称：\n";
        for(int i=0;i<n;i++) {
            mine::String str;
            std::cin >> str;
            vertexes[i] = str;
            graph.SetDot(str);
        }
        ClearBuffer();
        have_vertex=true;
    }
```

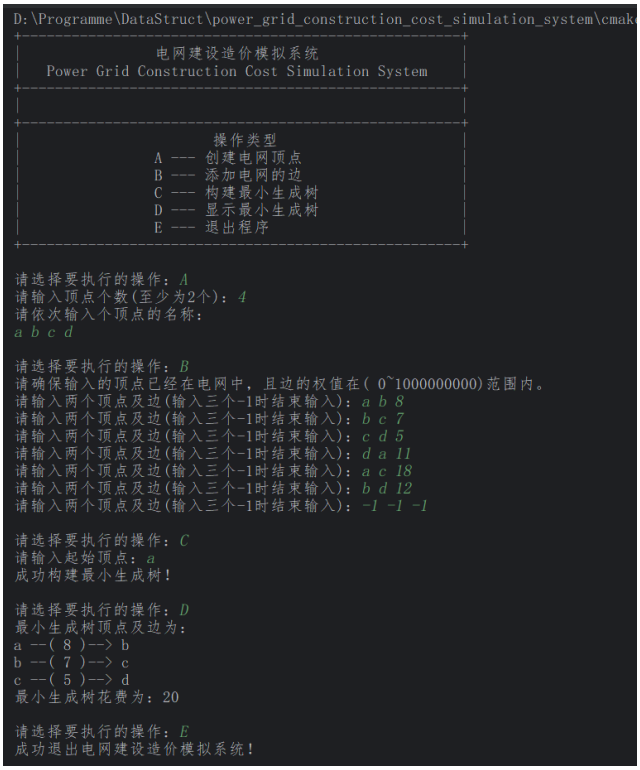
添加电网的边逻辑：

```
    void AddEdge(Graph<mine::String> &graph, const int &n,Hash<mine::String,
int> &vertexes_index, const bool &have_vertex, bool &have_edge, bool
&have_prim)
{
    if(!have_vertex) {
        std::cout << "没有顶点，无法添加边！\n";
        return;
    }
    have_prim = have_edge =false;
    graph.InitialEdgeValue();//初始化边，防止上一次输入产生影响
    bool **has_set=new(std::nothrow) bool*[n];
    assert(has_set!=nullptr);
    for(int i=0;i<n;i++) {
        has_set[i] = new(std::nothrow) bool[n];
        assert(has_set[i]!=nullptr);
    }
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
```

```
                has_set[i][j]=false;
        std::cout<<"请确保输入的顶点已经在电网中，且边的权值在
( 0~"<<INF<<" )范围内。\n";
        while(true) {
            std::cout<<"请输入两个顶点及边(输入三个-1 时结束输入)：";
            mine::String start,end;
            double edge_value;
            std::cin>>start>>end>>edge_value;
            if (start=="-1"&&end=="-1"&&edge_value==-1) {
                ClearBuffer();
                break;
            }
            if (vertexes_index.find(start)==nullptr) {
                std::cout << "第一个顶点不在顶点集合之中!\n";
                ClearBuffer();
                continue;
            }
            if (vertexes_index.find(end)==nullptr) {
                std::cout << "第二个顶点不在顶点集合之中!\n";
                ClearBuffer();
                continue;
            }
            if (std::cin.fail() || edge_value <= 0 || edge_value >= INF) {
                std::cout<<"边输入非法！\n";
                ClearBuffer();
                continue;
            }
            graph.SetEdgeValue(start, end, edge_value);
            if (has_set[vertexes_index[start]][vertexes_index[end]])
                std::cout << "已更新" << start <<"与"<< end << "之间的边！\n";
            has_set[vertexes_index[start]][vertexes_index[end]] = true;
            has_set[vertexes_index[end]][vertexes_index[start]] = true;
        }
        have_edge=true;
```

```
    for(int i=0;i<n;i++)
        delete has_set[i];
    delete has_set;
}
```

3.1.4 示例



3.2 Prim 最小生成树构建功能

3.2.1 实现思路

Prim 算法构建最小成生成树的主要逻辑为：

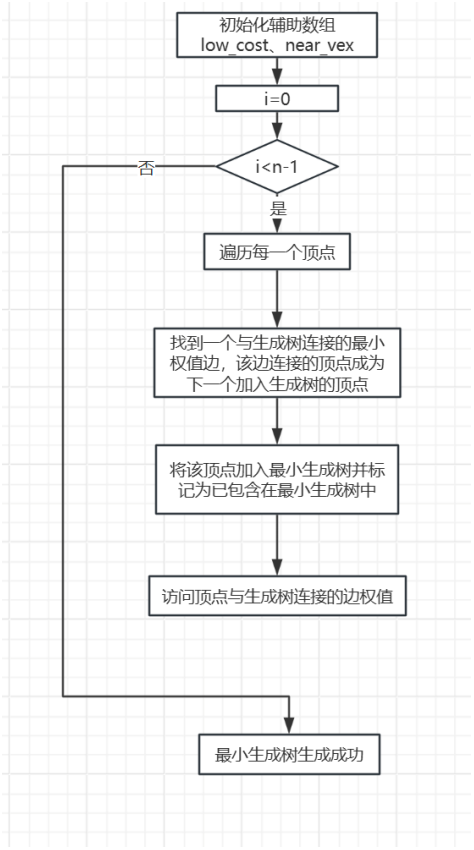
1. 初始状态，起始顶点开始，构建最小生成树（MST）。初始化一个 low_cost 数组：每个元素表示当前生成树中某个顶点到其他顶点的最小边权值；初始化一个 near_vex 数组：记录每个顶点与生成树中顶点的最近连接关系。

2.进入一个循环，执行以下操作，在所有未加入生成树的顶点中，找到一个与生成树连接的最小权值边。该边连接的顶点成为下一个加入生成树的顶点；将选中的顶点加入到最小生成树中，并标记为已访问；更新所有未访问顶

点与生成树连接的边权值，确保 low_cost 数组始终保存当前顶点到生成树中各顶点的最小边权值。对应的 near_vex 数组更新为新的最近连接顶点。

3.当所有顶点都被加入到最小生成树中时，算法结束；理想情况下，生成树包含了图中所有的顶点，并且所有边的权值之和是最小的。

3.2.2 流程图



3.2.3 核心代码

```
template<typename T>
    bool Graph<T>::Prim(T start)
    {
        assert(n==size);
        double* low_cost= new(std::nothrow) double[size];
        assert(low_cost!=nullptr);
        int* near_vex= new(std::nothrow) int[size];
        assert(near_vex!=nullptr);
        for( int i = 0;i < size;i++ ) {
```

```
            low_cost[i] = edge_value[hash_table[start]][i];
            near_vex[i] = hash_table[start];
        }
        near_vex[hash_table[start]] = -1;
        for (int i = 1; i < size; i++) {
            double min = INF;
            int v = -1;
            for (int j= 0; j < size; j++)
                if (near_vex[j] != -1 && low_cost[j] < min) {
                    v = j;
                    min = low_cost[j];
                }
            if (v != -1) {
                MSTEdgeNode<T> temp;
                temp.tail = dots[near_vex[v]];
                temp.head = dots[v];
                temp.key = low_cost[v];
                min_span_tree.Insert(temp);
                near_vex[v] = -1;
                for (int j = 0; j < size; j++)
                    if (near_vex[j] != -1 && edge_value[v][j] < low_cost[j]) {
                        low_cost[j] = edge_value[v][j];
                        near_vex[j] = v;
                    }
            }
        }
        delete[] low_cost;
        delete[] near_vex;
        return min_span_tree.BuildSuccess();
    }
```

3.2.4 示例

示例同 3.1.4

3.3 异常处理功能

3.3.1 输入非法的异常处理

3.3.1.2 操作类型输入非法的异常处理

操作类型输入非法的异常处理通过如下代码实现

```
char Select()
{
    char ch;
    while (true) {
        std::cout << "\n 请选择要执行的操作：";
        std::cin >> ch;
        if (ch < 'A' || ch > 'E') {
            std::cout << "输入非法，请重新输入！\n";
            ClearBuffer();
            continue;;
        }
        ClearBuffer();
        break;
    }
    return ch;
}
```

这段代码的具体执行逻辑如下：

1. 显示提示信息，提示用户进行输入；

2. 进入一个无限循环，以等待用户输入；

3. 用户以字符的方式输入选项，应当输入为 A~E 中的某一个，若输入不规范或不符合范围则清除当前输入状态和缓冲区，开始重新从头开始执行循环；反之，则退出循环，返回选项。

3.3.1.2 顶点数目输入非法的异常处理

在创建电网顶点功能中输入顶点数目时，进入一个 while 无限循环，当输入的顶点数目为大于 1 的正整数，保证电网中至少可以存在一个边时，退出循环，保证接下来的操作无误；否则，清除当前输入状态和缓冲区，从重新输入。

3.3.1.3 起点输入非法的异常处理

在建立最小生成树之前，需要输入起点，起点必须为顶点中的某一个；输入时，进入一个 while 无限循环，开始输入起点，输入完成后，将输入的值与已有定点比较，若存在相同，则退出循环进行下一步操作，反之，需要重新输入。

3.3.2 动态内存申请失败的异常处理

在进行 Hash 类、String 类、MinSpanTreem 类、Graph 类的动态内存申请时，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，对于内存分配失败，通过 assert 断言检查并终止程序，例如：

```
template<typename T>
MinSpanTree<T>::MinSpanTree(int size)
{
    maxSize = size - 1;
    n = 0;
    edge_value = new(std::nothrow) MSTEdgeNode<T>[size];
    assert(edge_value!=nullptr);
}
```

3.3.3 哈希冲突的异常处理

在 Hash 类中，桶的数目是固定的，通过哈希函数计算的桶的索引难免会出现相同的问题，即产生哈希冲突；实现过程中，采用链表的方式解决哈希冲突，即将每一个桶看作一个链表，结点储存当前键值对与下一个结点的地址，产生哈希冲突时，在链表中进行搜索或者插入链表即可。

3.3.4 逻辑顺序异常处理

虽然程序会提供四个选项让用户选择操作，但实际使用时，这四个选项逻辑顺序，只有当前一个现实时，才能进行本次操作；

在操作过程中，使用三个 bool 类型变量来控制操作是否可以执行，例如，当创建电网顶点后 have_vertex 为 true 时，才可以执行添加电网的边；同时，

对于顶点调整、边调整之后，相应的（have_edge、have_prim）、have_prim 也会调整，以保证以新的顶点、边构建和显示最小生成树，而不是旧的。

3.3.5 其他异常处理

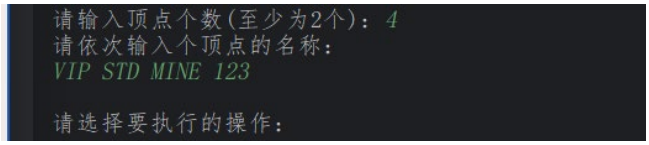
除了以上异常，程序中还存在各种异常，例如越界等需要处理；对于这些异常，根据错误产生逻辑，使用断言基本可以满足需要。

第 4 章 项目测试

4.1 创建电网顶点功能测试

当输入的顶点个数超过类型上限、超过可选范围、浮点数、字符、字符串时，可以验证对输入数据做了错误处理

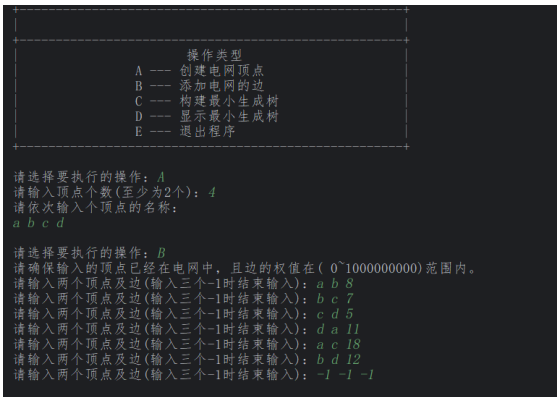
当输入正确时，程序正常运行



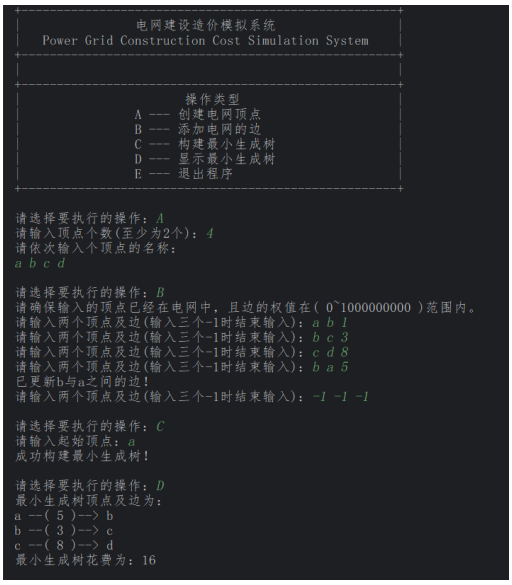
4.2 添加电网的边功能测试

当不创建电网顶点直接添加电网的边时、输入的电网的顶点不存在、顶点相同时、输入的边的权值非法时，提示错误：

当创建电网顶点且输入无误后，该功能正常执行。

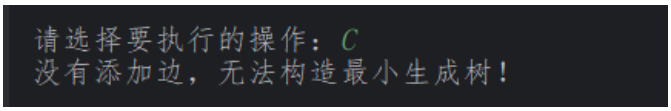


若输入的边已经被建立，则选择新边作为权值。

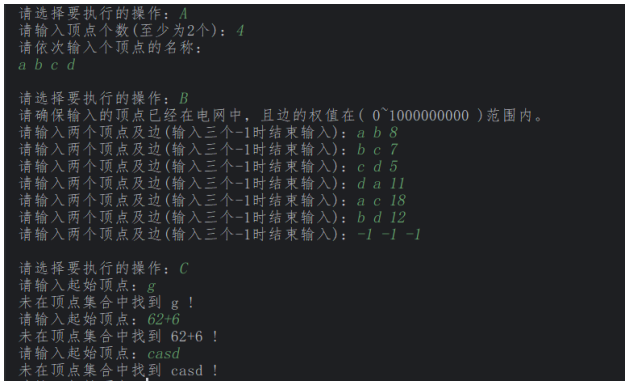


4.3 构建最小生成树功能测试

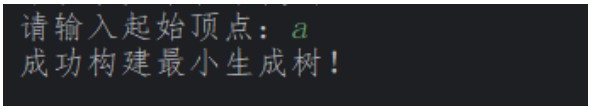
当不添加电网的边直接构建最小生成树时，提示错误：



当输入的起点不在顶点集合时时，提示错误并要求重新输入：

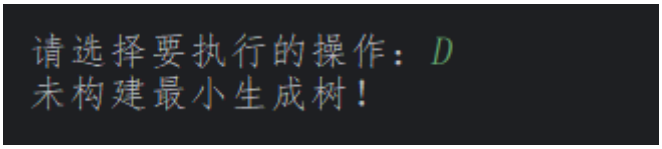


当操作无误时，该功能正常执行。



4.4 显示最小生成树功能测试

当不构建最小生成树直接显示最小生成树时，提示错误：



当构建最小生成树后，该功能正常执行。

```
请选择要执行的操作：A
请输入顶点个数(至少为2个)：3
请依次输入个顶点的名称：
asd fgh jkl

请选择要执行的操作：B
请确保输入的顶点已经在电网中，且边的权值在( 0~1000000000 )范围内。
请输入两个顶点及边(输入三个-1时结束输入)：asd fgh 6
请输入两个顶点及边(输入三个-1时结束输入)：fgh jkl 5
请输入两个顶点及边(输入三个-1时结束输入)：-1 -1 -1

请选择要执行的操作：C
请输入起始顶点：asd
成功构建最小生成树！

请选择要执行的操作：D
最小生成树顶点及边为：
asd --( 6 )--> fgh
fgh --( 5 )--> jkl
最小生成树花费为：11
```

第 5 章 相关说明

5.1 编程语言

本项目全部.cpp 文件以及.h 文件均使用 C++编译完成。

5.2 Windows 环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：CLion 2024

工具集：MinGW 11.0 w64

5.3 Linux 环境

基于 Linux 内核的操作系统发行版：Ubuntu 24.04.1

Linux 命令编译过程为：

1. 定位项目所在文件夹，包括.pp 与.h 文件；具体命令为: cd /home/bruce/programe/ power_grid_construction_cost_simulation_system

2.编译项目，生成可执行文件；具体命令为: g++ -static -o power_grid_construction_cost_simulation_system_linux power_grid_construction_cost_simulation_system.cpp my_hash.h my_graph_mst.h my_string.h;

其中指令含义分别为：

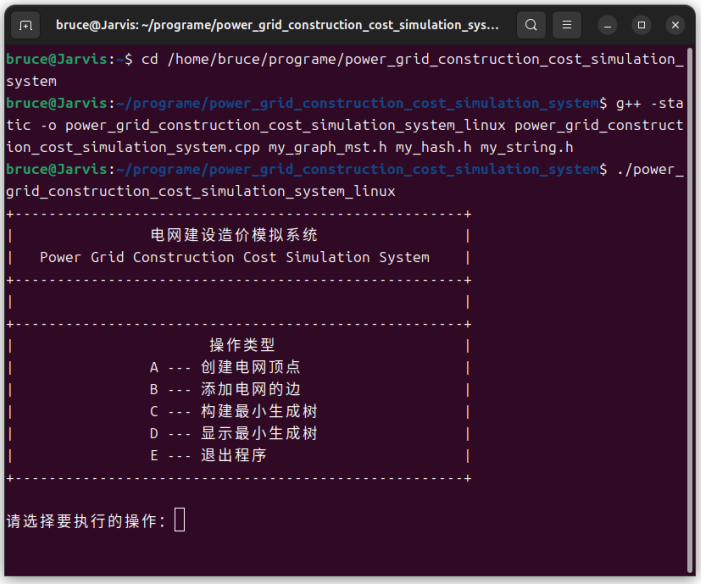
g++: 调用 GNU 的 C++编译器

-static: 使用静态链接而非动态链接，将所有依赖库直接嵌入到可执行文件，文件存储空间变大，但可以单独运行

-o power_grid_construction_cost_simulation_system_linux: -o 表示输出文件选项，power_grid_construction_cost_simulation_system_linux 为可执行文件名

power_grid_construction_cost_simulation_system.cpp my_hash.h my_graph_mst.h my_string.h:编译所需要的文件。

3.运行可执行文件；具体命令为 ./ power_grid_construction_cost_simulation_system_linux


bruce@Jarvis: ~/programe/power_grid_construction_cost_simulation_sys...
bruce@Jarvis:~$ cd /home/bruce/programe/power_grid_construction_cost_simulation_system
bruce@Jarvis:~/programe/power_grid_construction_cost_simulation_system$ g++ -static -o power_grid_construction_cost_simulation_system_linux power_grid_construction_cost_simulation_system.cpp my_graph_mst.h my_hash.h my_string.h
bruce@Jarvis:~/programe/power_grid_construction_cost_simulation_system$ ./power_grid_construction_cost_simulation_system_linux
电网建设造价模拟系统
Power Grid Construction Cost Simulation System
操作类型
A --- 创建电网顶点
B --- 添加电网的边
C --- 构建最小生成树
D --- 显示最小生成树
E --- 退出程序
请选择要执行的操作：