同济大学
TONGJI UNIVERSITY

离散数学

课程实验报告

最小生成树

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：　　李冰

二〇二四年十一月三十日

目录

一 实验目的

通过实验，帮助学生深入理解最小生成树（MST）的定义和最小生成树在实际问题中的应用，如实现网络设计、道路布局、通信网络最优布局等。

学习和掌握常见的最小生成树求解算法，通过编程实现这些算法，帮助学生掌握图的基本操作和数据结构。通过实际编程实现最小生成树算法，培养学生的编程能力和解决实际问题的能力。

实验在，学生将通过使用最小生成树算法解决实际问题，找到最优解，提升学生将实际问题抽象、通过数学和编程来解决问题的能力。

二 实验内容

在城市的基础设施建设中，城市间的通信道路规划是一项重要任务。通信道路的建设不仅需要考虑道路的数量和布局，还需要计算每条道路的建设造价。随着城市的发展，如何在有限的资源和预算下，保证所有城市间互联互通的前提下，设计一条高效且成本最低的通信网络，使得所有城市之间能够保持畅通的通信，成为了城市规划中的一大挑战。

如下图所示的赋权图表示某七个城市，预先给出它们之间的一些直接通信道路造价（单位：万元），试给出一个设计方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算其最小值。

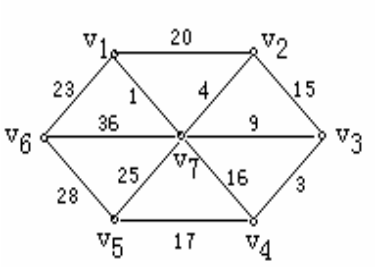


图 1 七个城市赋权图

三 实验环境

编程开发语言：C++

文 件 编 码：UTF-8

集成开发环境：Clion 2024

工　具　集：MinGW 11.0 w64

四 实验原理

为了求解最小代价，使花费的总代价最小，这是数学中经典的求解最小(费用)生成树的算法。

4.1 最小生成树

最小生成树（MST）是图论中的一个重要概念，旨在解决如何连接图中所有节点，且使得边的总权重最小。具体来说，最小生成树是一个包含图中所有顶点的子图，它连接了图中所有的节点，并且总边权和最小。最小生成树的核心特性在于，它不仅能确保图的连通性，还能以最小的成本完成所有节点的连接。

这种特性使得最小生成树在许多实际应用中都具有重要价值。例如，在城市通信网络的设计中，最小生成树帮助选择最优的连接方式，确保所有城市之间能够实现通信，同时使得通信网络的建设成本最小。类似地，在电力、供水、交通等领域，最小生成树也常常用于规划网络布局，以节约资源和减少建设成本。在计算机网络中，最小生成树能够帮助设计高效的路由和数据传输路径，确保网络的稳定运行。

最小生成树的另一个优点是它能够避免冗余连接，保持网络的简洁性。通过选取最小成本的连接路径，最小生成树有效地减少了不必要的资源浪费，并使得网络结构更加高效。此外，最小生成树所构建的网络具有良好的扩展性，可以根据未来需求对网络进行优化和调整。因此，最小生成树不仅在理论研究中具有重要地位，更在实际工程中为许多优化问题提供了解决方案。

4.2 最小生成树的构建

构建最小生成树的准则是确保在连接所有节点的同时，选择的边的总权重最小，并且避免形成环。这个准则的核心目标是通过最少的边和最小的成本完成图的连通。Prim 算法和 Kruskal 算法是两种常用的方式。两者的思路不同，但都遵循着“最小化边权和”这一原则。

Prim 算法从一个起始节点出发，逐步选择与已选节点相连的最小权重边扩展生成树。每次选择的边都使得生成树的结构不断扩大，直到所有节点都被连接。

Kruskal 算法则从所有边中选取权重最小的边，并按权重排序，逐步将边加入生成树中，同时利用并查集来防止环的形成，直到所有顶点都被连接。

两者的核心目标都是通过不断选择权重最小的边来构建最小生成树，但在操作方式上有所不同：Prim 算法是基于节点扩展的方式，而 Kruskal 算法则是基于边的排序与选择。Prim 算法适用于稠密图，而 Kruskal 算法则更适合稀疏图。两者的选择取决于图的特性和实际需求。

五 实验过程

5.1 实验思路

本实验利用 Prim 算法构建图的最小生成树（MST）。该算法属贪心算法，从图中任一顶点起步，持续挑当前权重最小边，把未纳入生成树的顶点与已选顶点相连，直至涵盖所有顶点。

其核心是每次选最小边入树，以此来保证总权重最小。执行时，依贪心策略逐步选边，新顶点加入生成树，同步更新到达其余未加入顶点的最小边，反复操作，同时避免成环，最终形成无环连通子图，即最小生成树。

采用这种算法可以高效达成构建，确保结果最优且准确。

5.2 程序框架设计

程序的框架大致分为几个部分：

获取图的顶点数和边数：使用 InputNum 函数提示用户输入顶点数和边数，进行合法性验证。若输入 0 0，结束程序。

创建图并添加顶点：创建 Graph 类对象，根据顶点数添加顶点。使用 SetDot 检查并添加顶点，更新哈希表。

输入边信息：使用 Input 函数获取边的信息，验证合法性：检查节点范围、无环、权值有效性等。合法则更新邻接矩阵。

构建最小生成树：使用 Prim 算法从指定顶点（如顶点 1）开始，构建最小生成树。通过 low_cost 和 near_vex 数组更新最小边信息，直到所有顶点处理完或失败。

输出最小生成树信息：若构建成功，使用 GetMST 获取最小生成树，Print 输出每条边和总耗费；若失败，输出图可能不连通的错误提示。

循环执行：每次执行后等待用户输入，直到用户选择退出（00）。结束时输出“欢迎下次使用”提示。

5.3 数据结构设计

5.3.1 最小生成树结点（MSTEdgeNode）设计

5.3.1.1 概述

MSTEdgeNode 用于存储最小生成树当前边的起点和终点，以及边的权值。

5.3.1.2 结构体定义

```
template<typename T>
struct MSTEdgeNode {
    T tail,head;
    double key;
    MSTEdgeNode();
    MSTEdgeNode(T src,T end,double key);
};
```

5.3.1.3 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；

5.3.1.4 构造函数

MSTEdgeNode();

构造函数，根据默认信息初始化数据；

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

构造函数，根据传入的具体信息初始化数据；

5.3.2 最小生成树（MinSpanTree）设计

5.3.2.1 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，该类支持插入最小生成树的边等操作。

5.3.2.2 MinSpanTree 类定义

```
template<typename T>
class MinSpanTree {
public:
    MSTEdgeNode<T> *edge_value;
    MinSpanTree(){maxSize = n = 0;edge_value = nullptr;};
    MinSpanTree(int size);
    MinSpanTree(const MinSpanTree<T> &other);
    ~MinSpanTree() { delete[] edge_value; }
    void Initial(int size);
    void MakeEmpty();
    void Insert(const MSTEdgeNode<T> &item);
    bool BuildSuccess() { return n == maxSize; }
    int GetSize()const{return maxSize; }
private:
    int maxSize, n;
};
```

5.3.2.3 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边结点

5.3.2.4 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(const MinSpanTree<T> &other);

拷贝构造函数，用于满足函数返回 MinSpanTree 对象。

~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。

5.3.2.5 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；

int GetSize() const { return maxSize; }

获取最小生成树容量大小。

5.3.3 无向图设计

5.3.3.1 概述

Graph 主要用来存储图的相关信息，包括顶点和边的权值，并根据这些信息借助 Prim 算法构建最小生成树。

5.3.3.2 无向图类定义

```
template<typename T>
class Graph {
public:
    Graph(int sz);
    ~Graph() { MakeEmpty(); }
    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);
    MinSpanTree<T> GetMST()   { return min_span_tree; }
    void InitialEdgeValue();
private:
    int size, n;
```

```
    T *dots;
    double **edge_value;
    MinSpanTree<T> min_span_tree;
    std::unordered_map<T, int> hash_table;
};
```

5.3.3.3 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;顶点数组，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，将 Prim 算法得到的最小生成树存储在其中。

Hash<T,int> hash_table;散列表的一个对象，实现通过顶点映射邻接矩阵的索引。

5.3.3.4 构造函数与析构函数

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。

5.3.3.5 公有成员函数

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);

向无向图中插入顶点；

bool FindDot(const T &dot);

寻找点是否在无向图中，一般用来防止重复插入一个顶点

bool Prim(T start);

使用 Prim 算法构建最小生成树

void InitialEdgeValue();

初始化邻接矩阵；

MinSpanTree<T> GetMST()　{ return min_span_tree; }

返回最小生成树的对象，用于后续格式化输出等操作。

void InitialEdgeValue();

初始化邻接矩阵中边的权值，始、终顶点相同赋值为 0，不同则赋值为最大值 INF；

5.4 程序功能实现

5.4.1 逻辑值输入功能的实现

程序的输入功能主要包含两部分：输入图的顶点数和边数（InputNum 函数）与输入图的边数据（Input 函数）

（一）输入图的顶点数和边数 (InputNum)

1.用户输入顶点数（num_vertices）和边数（num_edges），输入格式要求为两个整数，输入的边数必须在合法范围内：num_edges 不能超过完全图的边数 num_vertices * (num_vertices - 1) / 2。

2.该函数通过循环与条件判断来确保用户输入有效，当输入格式不合法时，即如果输入负数、非整数、超范围的数或字符，程序会提示输入错误，要求用户重新输入。

3.如果用户输入 0 0，则函数返回 false，表示用户希望退出程序。

（二）输入图的边数据 (Input)

1.用户输入每一条边的两个顶点编号和边的权值。

2.然后进行合法性检查，检查顶点编号是否在图的有效范围内、权值是否合理、边是否已存在（防止多次输入相同边）、检查两个顶点是否相同（同一城市通信道路造价为 0，不需要输入），并且验证权值是否在合理范围（0 到 INF 之间）。

3.如果输入不合法，程序会给出错误信息，并提示用户重新输入该条边的信息。

4.当输入合法时，将该边加入无向图中。

5.4.2 异常处理功能

在 Graph 类和 MinSpanTree 类中，为了减少固定数组带来的不便，需要根据具体顶点数量、边的数量来为数组分配动态内存，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），在分配完内存后，代码通过断

言 assert 来检查指针是否为空指针，如果是空指针，则抛出异常信息，一般包含断言失败的表达式，源文件名称，断言失败的代码的行号。方便用户和编程人员快速定位异常发生的语句，便于程序改进。

5.5 核心算法实现

本实验核心算法为 Prim 算法，用于在加权无向图中构建最小生成树，算法的实现过程如下：

1. 初始化辅助数据结构：low_cost[]用于存储从最小生成树（MST）已选顶点到每个其他顶点的最小边的权重；near_vex[]用于记录每个顶点最近的已选顶点的索引，表示如何将该顶点连接到已选集合中。

2. 初始化边权值和访问标记：将 low_cost[i] 设置为起点 start 到其他顶点的边权重。这个值在算法中会不断更新，以保持其为当前未加入 MST 的结点到已经建立的子图的最小边权。near_vex[] 初始化时将每个顶点与起点 start 相连，即都暂时认为起点是所有顶点的最近连接点。

3. 主循环：贪心选择最小的边，在每一轮循环中，寻找未访问的顶点：通过遍历 low_cost[] 数组，找到与已选顶点集合中最近的、未被选中的顶点 v。选中的顶点 v 会与它连接的顶点一起形成子图的一条边。该边的起点是 near_vex[v]，终点是 v，权重是 low_cost[v]。

4.通过遍历图中所有未在子图的顶点，更新它与子图的最小边权；例如当顶点 x 与新加入的顶点之间的边的权重小于它与旧的子图的边权时，将新的边权作为它与子图的边权，同时将新的顶点作为该边的一个端点。

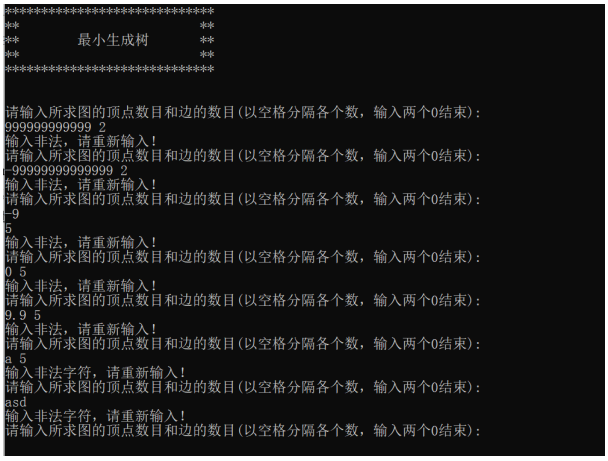
5.继续执行主循环，直到找到子图所有的边。

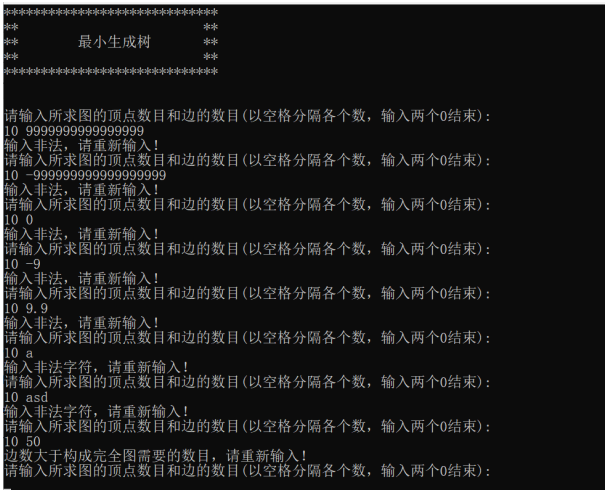
6.检查最小生成树是否构建成功：通过 BuildSuccess()检查最小生成树是否成功构建；如果返回 true，表示最小生成树构建成功，程序可以继续进行；若为 false，提示错误信息，并提示检查图是否连通。

BuildSuccess()通过当前子图的边数是否为图的顶点数减一来判断最小生成树是否生成成功；一个正确的最小生成树一定符合这个条件，通过 Prim 算法生成的子图中，符合这个条件的一定是最小生成树。

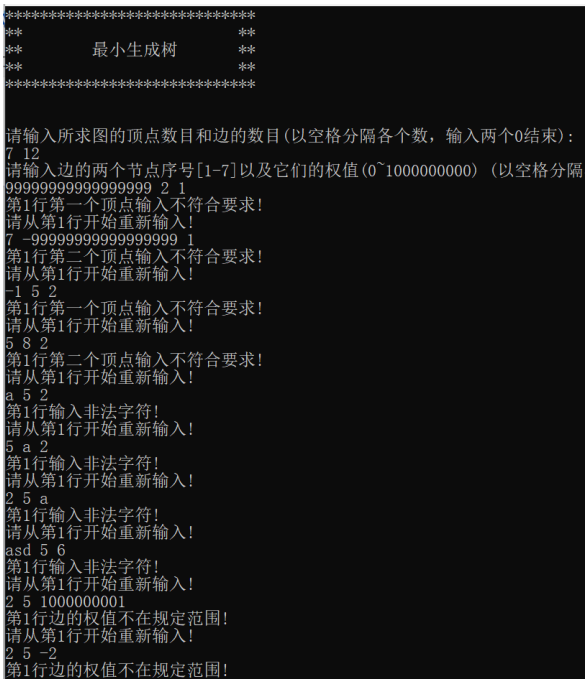
六 实验结果测试

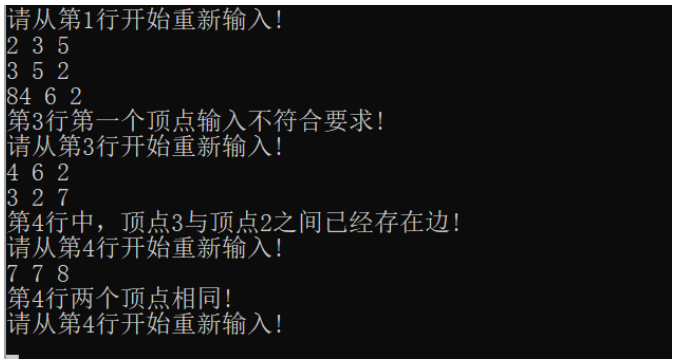
在输入图的顶点和边的数目时，分别输入超 int 类型上限、非正整数、浮点数、字符、字符串，以及在输入边时输入超过构成完全图所需要的边数，程序输入错误提示信息，证明程序对于输入错误进行了处理。





在输入图的边时，输入顶点序号分别输入超 int 类型上限、非正整数、浮点数、字符、字符串，或者重复输入相同的定点对、输入顶点相同以及边的权值小于等于 0，程序输入错误提示信息，证明程序对于输入错误进行了处理。





当输入所有结果无误时，若构成的图不是完全图，则无法构建最小生成树，按下任意键进行下一轮计算。

当输入所有结果无误时，若构成的图不是完全图，则构建并输出最小生成树，按下任意键进行下一轮计算。

```
******************************
**                            **
**          最小生成树        **
**                            **
******************************

请输入所求图的顶点数目和边的数目(以空格分隔各个数，输入两个0结束)：
7 12
请输入边的两个节点序号[1-7]以及它们的权值(0~1000000000) (以空格分隔各个数)
1 2 20
2 3 15
3 4 3
4 5 17
5 6 28
6 1 23
1 7 1
2 7 4
3 7 9
4 7 16
5 7 25
6 7 36
开始构建最小生成树...

最小生成树构建成功!

开始打印最小生成树...
最小耗费是：1和7
最小耗费是：7和2
最小耗费是：7和3
最小耗费是：3和4
最小耗费是：4和5
最小耗费是：1和6
总耗费为：57(万元)


Press any key to continue...
```

当在输入顶点和边的数目时输入两个 0 时，退出程序。

```
******************************
**                            **
**          最小生成树        **
**                            **
******************************

请输入所求图的顶点数目和边的数目(以空格分隔各个数，输入两个0结束)：
0 0

欢迎下次使用!
```

八 实验心得

这次实验让我深入理解了最小生成树（MST）算法，特别是如何通过 Prim 算法解决城市间通信道路的建设问题。在这个实验中，需要在给定的城市图中，设计一条使所有城市互联互通且总建设造价最小的通信网络，这个过程让我在多个方面得到了提升。

首先，通过实际实现 Prim 算法，我对图论中的贪心策略有了更清晰的理解。Prim 算法从一个初始节点出发，不断选择最小权重的边，逐步扩展生成树。实现过程中，我使用了邻接矩阵来表示图，并通过 low_cost 数组来存储当前顶点到每个未访问顶点的最小边权值。每次选择最小的边，加入到最小生成树中，直到构建出完整的最小生成树。这一过程让我理解了如何在图中高效选择最优路径，并且体会到贪心算法在解决实际问题中的优势和局限。

在编程技巧上，本次实验加深了我对 C++ 数据结构的掌握。图的邻接矩阵和哈希表的使用，使我能够快速管理图中的顶点和边。同时为了优化图的构建，我采用了动态内存管理，通过 new 和 delete[] 操作管理内存，避免了内存泄漏

的发生。此外，使用模板类来处理通用的节点类型，提高了程序的灵活性和可复用性。

实验中的用户交互设计也是一个重点。为保证输入数据的合法性，我加入了多项验证功能。无论是顶点编号、边权重的范围，还是避免重复边的出现，都需要在用户输入时进行检查。这一部分增强了我对输入验证与异常处理的理解，也提高了程序的稳定性。尤其是在处理用户输入错误时，能够及时给出清晰的提示，提升了程序的友好性。

更重要的是，实验让我意识到算法与现实问题的紧密联系。城市通信网络的设计问题，实际上就是一个最小生成树问题，它广泛应用于通信网络、交通运输、能源分配等多个领域。在实验中，通过对最小生成树的构建和预算的计算，我学会了如何将抽象的算法应用于实际问题中，确保在有限的资源下，实现最优的设计方案。这不仅增加了我对图论应用的理解，也让我看到了算法在解决实际问题中的巨大潜力。

这次实验不仅提升了我的编程能力，也增强了我对图算法的实际应用能力。

附录 · 源码

```cpp
#include <cassert>
#include <conio.h>
#include <iostream>
#include <limits>
#include <unordered_map>
#include <vector>

#define INF 1000000000

/*************************************************************
* Struct: MSTEdgeNode
* Function: 表示最小生成树中的一条边
*************************************************************/
template<typename T>
struct MSTEdgeNode {
    T tail, head; // 边的起点和终点
    double key; // 边的权重
    MSTEdgeNode(): tail(T()), head(T()), key(0) {}
    MSTEdgeNode(T src, T end, double key): tail(src), head(end), key(key) {}
};

/*************************************************************
* Class:    MinSpanTree
* Function: 管理最小生成树的边集合。
*************************************************************/
template<typename T>
class MinSpanTree {
public:
    MSTEdgeNode<T> *edge_value;
    MinSpanTree(){maxSize = n = 0;edge_value = nullptr;};
    MinSpanTree(int size);
    MinSpanTree(const MinSpanTree<T> &other);
    ~MinSpanTree() { delete[] edge_value; }
    void Initial(int size);
    void MakeEmpty();
    void Insert(const MSTEdgeNode<T> &item);
    bool BuildSuccess() { return n == maxSize; }
    int GetSize() const { return maxSize; }

private:
```

```
    int maxSize, n;
};

/*************************************************************
* Function Name:     MinSpanTree
* Function:          构造函数，存储内存
* Input Parameter:  size - 最小生成树可以存储的边数。
*************************************************************/
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);
}

/*************************************************************
* Function Name:   MinSpanTree<T>::MinSpanTree
* Function:        拷贝构造函数，创建一个新的最小生成树对象，并复制另一个最小生成树
的内容。
* Input Parameter: other - 需要复制的最小生成树对象。
*************************************************************/
template<typename T>
MinSpanTree<T>::MinSpanTree(const MinSpanTree<T> &other)
{
    if (this == &other)
        return;
    maxSize = other.maxSize;
    n = other.n;

    // 分配内存
    edge_value = new MSTEdgeNode<T>[maxSize];
    assert(edge_value != nullptr);

    // 复制数据
    for (int i = 0; i < n; i++) {
        edge_value[i] = other.edge_value[i];
    }
}

/*************************************************************
* Function Name:    Initial
* Function:         初始化最小生成树，设置边数组大小并分配内存。
```

```
* Input Parameter: size - 最小生成树可以存储的边数。
*************************************************************/
template<typename T>
void MinSpanTree<T>::Initial(int size)
{
    assert(maxSize==0);
    maxSize = size - 1;
    n = 0;
    edge_value = new(std::nothrow) MSTEdgeNode<T>[size];
    assert(edge_value!=nullptr);
}

/*************************************************************
* Function Name: MakeEmpty
* Function:      清空最小生成树的边集合，并释放内存。
*************************************************************/
template<typename T>
void MinSpanTree<T>::MakeEmpty()
{
    if (edge_value != nullptr) {
        delete[] edge_value;
        edge_value = nullptr;
    }
    maxSize = n = 0;
}

/*************************************************************
* Function Name:    Insert
* Function:         向最小生成树中插入一条边。
* Input Parameter:  item - 要插入的边。
*************************************************************/
template<typename T>
void MinSpanTree<T>::Insert(const MSTEdgeNode<T> &item)
{
    assert(n< maxSize);
    assert(edge_value!=nullptr);
    edge_value[n] = item;
    n++;
}

/*************************************************************
* Class:    Graph
* Function: 表示加权无向图，并提供构建最小生成树的功能。
*************************************************************/
```

```
template<typename T>
class Graph {
public:
    Graph(int sz);
    ~Graph() { MakeEmpty(); }
    void MakeEmpty();
    void SetEdgeValue(T start, T end, double value);
    void SetDot(const T &dot);
    bool FindDot(const T &dot);
    bool Prim(T start);
    MinSpanTree<T> GetMST()  { return min_span_tree; }
    void InitialEdgeValue();
private:
    int size, n;
    T *dots;
    double **edge_value;
    MinSpanTree<T> min_span_tree;
    std::unordered_map<T, int> hash_table;
};

/*************************************************************
* Function Name:    Graph<T>::Graph
* Function:         Graph 的构造函数，初始化图的结构。
* Input Parameter:  sz - 图中顶点的最大数量。
*************************************************************/
template<typename T>
Graph<T>::Graph(const int sz):size(sz),min_span_tree()
{
    assert(sz>1);
    n = 0;
    dots = new(std::nothrow) T[size];
    assert(dots!=nullptr);
    edge_value = new(std::nothrow) double *[size];
    assert(edge_value!=nullptr);
    for (int i = 0; i < size; i++) {
        edge_value[i] = new(std::nothrow) double[size];
        assert(edge_value[i]!=nullptr);
    }
    // Initialize adjacency matrix
    InitialEdgeValue();

    min_span_tree.Initial(size);
    hash_table=std::unordered_map<T, int>();
}
```

```
/************************************************************
* Function Name: MakeEmpty
* Function:      清空图的所有数据并释放内存。
************************************************************/
template<typename T>
void Graph<T>::MakeEmpty()
{
    if (dots != nullptr) {
        delete[] dots;
        dots = nullptr;
    }
    if (edge_value != nullptr) {
        for (int i = 0; i < size; i++)
            delete[] edge_value[i];
        delete[] edge_value;
        edge_value = nullptr;
    }
    size = n = 0;
}

/************************************************************
* Function Name:    FindDot
* Function:         检查给定的顶点是否已存在于图中。
* Input Parameter: dot - 要检查的顶点。
* Returned Value:  如果顶点存在，返回 true；否则返回 false。
************************************************************/
template<typename T>
bool Graph<T>::FindDot(const T &dot)
{
    for (int i = 0; i < n; i++)
        if (dots[i] == dot)
            return true;
    return false;
}

/************************************************************
* Function Name:   SetDot
* Function:        向图中添加一个顶点。
* Input Parameter: dot - 要添加的顶点。
************************************************************/
template<typename T>
void Graph<T>::SetDot(const T &dot)
{
```

```
    if (FindDot(dot))
        return;
    if (!hash_table.contains(dot))
        hash_table[dot] = n;
    dots[n] = dot;
    n++;
}

/************************************************************
* Function Name:    SetEdgeValue
* Function:         设置图中两顶点之间的边权重。
* Input Parameters: start - 边的起点。
*                   end - 边的终点。
*                   value - 边的权重值。
************************************************************/
template<typename T>
void Graph<T>::SetEdgeValue(T start, T end, double value)
{
    edge_value[hash_table[start]][hash_table[end]] = value;
    edge_value[hash_table[end]][hash_table[start]] = value;
}

/************************************************************
* Function Name:    Prim
* Function:        使用 Prim 算法构建图的最小生成树。
* Input Parameter: start - 最小生成树构建的起点。
* Returned Value:  如果最小生成树成功构建，返回 true；否则返回 false。
************************************************************/
template<typename T>
bool Graph<T>::Prim(T start)
{
    assert(n==size); // 确保图中顶点数与声明的大小一致

    double *low_cost = new(std::nothrow) double[size]; // 存储当前前顶点到每个顶点的最小边权值
    assert(low_cost!=nullptr);
    int *near_vex = new(std::nothrow) int[size]; // 存储与当前顶点连接的最近顶点索引
    assert(near_vex!=nullptr);

    // 初始化辅助数组
    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; // 起点标记为已访问

    // Prim 算法的主循环
    for (int i = 1; i < size; i++) {
        double min = INF;
        int v = -1;

        // 找到当前未访问顶点中与已访问集合最近的顶点
        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)
            break;
        // 如果找到有效的顶点，则将其加入最小生成树
        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; // 标记顶点 v 为已访问

        // 更新与新加入顶点连接的边的权值
        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;
            }
    }

    // 释放辅助数组内存
    delete[] low_cost;
    delete[] near_vex;
    return min_span_tree.BuildSuccess(); // 检查最小生成树是否构建完成
}

/*************************************************************
* Function Name:    InitialEdgeValue
* Function:         初始化邻接矩阵
```

```
* Input Parameter: None
* Returned Value:  None
************************************************************/
template<typename T>
void Graph<T>::InitialEdgeValue()
{
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            edge_value[i][j] = (i == j) ? 0 : INF;
}


/************************************************************
* Function Name: InputNum
* Function:         输入图的顶点数和边数。
*                   验证输入的合法性，确保用户输入的是正整数。
*                   输入 "0 0" 结束函数，返回 false。
* Parameters:       int& num_vertices - 图的顶点数目（输出参数）
*                   int& num_edges - 图的边数目（输出参数）
* Returned Value:  如果用户输入合法且非 "0 0"，返回 `true`；如果输入为 "0 0"，返回
`false`。
************************************************************/
bool InputNum(int &num_vertices,int& num_edges)
{
    double temp_v, temp_e;
    while (true) {
        std::cout << "请输入所求图的顶点数目和边的数目(以空格分隔各个数，输入两个 0
结束):\n";
        std::cin >> temp_v >> temp_e;
        if (std::cin.fail()) {
            std::cout<<"输入非法字符，请重新输入！\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<int>::max(), '\n');
            continue;
        }
        // 如果用户输入了 "0 0"，则结束输入并返回 false
        if (temp_v == 0 && temp_e == 0)
            return false;
        if ( temp_e <= 0 || temp_v <= 0 || temp_v != static_cast<int>(temp_v) ||
temp_e != static_cast<int>(temp_e)) {
            std::cout << "输入非法，请重新输入！\n";
            std::cin.ignore(std::numeric_limits<int>::max(), '\n');
            continue;
        }
```

```
        if (temp_e >temp_v * (temp_v - 1) / 2) {
            std::cout<<"边数大于构成完全图需要的数目，请重新输入！\n";
            std::cin.ignore(std::numeric_limits<int>::max(), '\n');
            continue;
        }
        std::cin.ignore(std::numeric_limits<int>::max(), '\n');// 清除输入流中的无
效数据
        break;
    }
    // 将临时变量的值转换为整数并存储到输出参数中
    num_vertices=static_cast<int>(temp_v);
    num_edges=static_cast<int>(temp_e);
    return true;
}

/*************************************************************
* Function Name: Input
* Function:         从用户输入图的边数据并存储到图中。
*                   验证输入的合法性，确保每条边的节点编号和权值都符合要求。
* Parameters:       Graph<int>& graph - 图对象，存储边的数据。
*                   const int num_vertices - 图的顶点数。
*                   const int num_edges - 图的边数。
*************************************************************/
void Input(Graph<int>& graph, const int num_vertices,const int num_edges)
{
    double start,end;
    double value;
    std::vector<std::vector<bool>> has_set(num_vertices+1,
std::vector<bool>(num_vertices+1, false));// 存储边是否已设置
    bool true_input=true;;// 标记输入是否合法，默认为合法
    std::cout<<"请输入边的两个节点序号[1-"<< num_vertices <<"]以及它们的权值
(0~"<<INF <<") (以空格分隔各个数)\n";
    for (int i = 0; i < num_edges; i++) {
        std::cin >> start >> end >> value;

        // 检查是否输入非法字符
        if (std::cin.fail()) {
            std::cout << "第" << i + 1 << "行输入非法字符!\n";
            true_input=false;
        }
        else if (start <=0||start>num_vertices||start!=static_cast<int>(start)) {
            std::cout << "第" << i + 1 << "行第一个顶点输入不符合要求!\n";
            true_input = false;
        }
```

```
        else if (end <=0||end>num_vertices||end!=static_cast<int>(end)) {
            std::cout<< "第" << i + 1 << "行第二个顶点输入不符合要求!\n";
            true_input = false;
        }
        // 检查两个顶点是否相同，即图中不能有环
        else if (start == end) {
            std::cout<<"第" << i + 1 << "行两个顶点相同!\n";
            true_input = false;
        }
        // 检查边是否已经存在
        else if (has_set[static_cast<int>(start)][static_cast<int>(end)]) {
            std::cout<<"第" << i + 1 << "行中，顶点"<< static_cast<int>(start)<<"与
顶点"<<static_cast<int>(end) <<"之间已经存在边!\n";
            true_input = false;
        }
        else if (value>=INF||value<=0) {
            std::cout<<"第" << i + 1 << "行边的权值不在规定范围!\n";
            true_input = false;
        }
        if (!true_input) {
            std::cout<<"请从第" << i + 1 << "行开始重新输入!\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<int>::max(), '\n');
            true_input=true;
            i--;
            continue;
        }

        // 如果输入合法，则将边的数值设置到图中，并标记这条边已经存在
        graph.SetEdgeValue(static_cast<int>(start), static_cast<int>(end), value);
        has_set[static_cast<int>(start)][static_cast<int>(end)] = true;
        has_set[static_cast<int>(end)][static_cast<int>(start)] = true;

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

/*************************************************************
* Function Name:  Print
* Function:       打印最小生成树中的所有边及其总耗费。
* Parameters:     const MinSpanTree<int>& result - 传入的最小生成树对象。
*************************************************************/
void Print(const MinSpanTree<int>&result)
{
```

```
    double total_cost=0;
    for (int i=0;i<result.GetSize();i++) {
        std::cout<<"最小耗费是: "<<result.edge_value[i].tail<<"和
"<<result.edge_value[i].head<<"\n";
        total_cost+=result.edge_value[i].key;
    }
    std::cout<<"总耗费为: "<<total_cost<<"(万元)\n";
}

/*************************************************************
* Function Name:   main
* Function:        主函数，程序入口，负责执行最小生成树的构建和展示。
* Description:     该函数是程序的主入口，负责循环执行以下功能:
*                  1. 用户输入图的顶点数和边数
*                  2. 输入图的边信息
*                  3. 使用 Prim 算法构建最小生成树（MST）
*                  4. 打印最小生成树的边信息和总耗费
*                  5. 如果图不连通，输出错误信息
*                  6. 提供用户选择是否继续运行程序的功能
*************************************************************/
int main()
{
    // 输出程序标语
    while (true) {
        system("cls");
        std::cout << "*****************************\n" //标语
                  << "**                         **\n"
                  << "**        最小生成树       **\n"
                  << "**                         **\n"
                  << "*****************************\n\n\n";

        int num_vertices, num_edges;
        const bool if_continue = InputNum(num_vertices, num_edges);
        if (!if_continue)
            break;;

        // 创建图对象，初始化顶点
        Graph<int> graph(num_vertices);
        for (int i = 1; i <= num_vertices; i++)
            graph.SetDot(i);

        // 输入图的边的信息（节点序号和权值）
        Input(graph, num_vertices, num_edges);
```

```
        std::cout<<"开始构建最小生成树...\n\n";
        if (graph.Prim(1)) {
            std::cout<<"最小生成树构建成功！\n\n 开始打印最小生成树...\n";

            // 获取最小生成树结果并打印
            MinSpanTree<int> result= graph.GetMST();
            Print(result);
        }
        else {
            std::cout << "最小生成树构建失败，请确保图连通!\n\n";
        }
        std::cout<<"\n\nPress any key to continue...\n";
        _getch();// 等待用户输入，按任意键继续
    }

    std::cout << "\n 欢迎下次使用！\n";
    return 0;

}
```