同济大学
TONGJI UNIVERSITY

《离散数学》课程实验报告

题目 最小生成树

姓　名 林继申

学　号 2250758

学　院 软件学院

专　业 软件工程

教　师 李　冰

二〇二三 年 十二 月 二十二 日

目录

1 实验目的

本次实验的主要目的是理解和实现最小生成树算法，特别是 Kruskal 算法。通过实际编程，学生不仅将深入理解算法的工作原理，还将掌握如何将理论应用于实际问题解决中。本实验旨在加强学生对离散数学中图论部分的理解，并提高他们使用计算机语言（如 C++）解决复杂问题的能力。

2 实验内容

随着城市化进程的加快，路网建设成为了一个重要议题。在城市之间，道路的连接需要优化，以确保交通运行效率。传统的路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，在建立路网的过程中，找到一种最低成本的道路连接方案，显得尤为重要。本实验内容为使用 Kruskal 算法设计和实现一个程序来解决最小生成树问题。

问题描述：如下图所示的路网赋权图表示七个城市之间的一些道路距离，要求给出一个路网设计方案，使得各城市之间既能够保持连接，又使得路网距离之和最小，并计算其最小值。

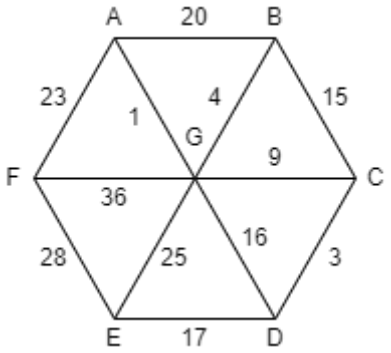

图 2.1 路网赋权图

3 实验环境

程序开发语言：C++

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

编译运行环境：本项目适用于 x86 架构和 x64 架构

4 实验原理

4.1 最小生成树

最小生成树（MST）是图论中的一个重要概念。在一个加权连通图中，生成树是一个无环子图，它包括图中的所有顶点，并且具有最少的边。如果这个树的边具有权重，则最小生成树的总边权重是所有生成树中最小的。

最小生成树具有一些重要的特性。对于一个边权各不相同的连通图，它的最小生成树是唯一的。如果图中有相同权重的边，可能存在多个最小生成树。最小生成树是全连通的，这意味着树中的每个顶点都与其他所有顶点间接或直接相连。作为树的一种，最小生成树不会包含任何环。

最小生成树在多个领域有着广泛的应用。在设计电信网络、计算机网络或任何其他类型的网络时，MST 可以用来最小化网络的总线长度或成本。在数据科学中，MST 可用于构建聚类算法。在物理学和生物学的网络建模中，最小生成树用于模拟自然过程，如水的分布或神经网络的结构。

4.2 构建最小生成树的算法

构建最小生成树的两种著名算法是 Prim 算法和 Kruskal 算法。Prim 算法适用于边密集的图，Kruskal 算法适用于边稀疏的图。

Prim 算法从图的一个顶点开始，逐步增长最小生成树。在每个步骤中，它都添加最小权重的边，这条边连接树与图中的某个未加入树的顶点。这个过程持续进行，直到所有顶点都被包含在树中。

Kruskal 算法则是一种更加全局的方法。它首先将图中的所有边按照权重进行排序，然后从最小的边开始添加到生成树中，但会避免形成环。该过程一直持续到生成树包含了所有的顶点。

5 实验过程

5.1 实验思路

本实验的核心思路是通过实现和应用 Kruskal 算法来构建最小生成树，以此解决一个具体的问题——在给定城市间建设成本最低的路网。实验将从理解最小生成树的概念出发，选择 Kruskal 算法作为构建最小生成树的算法，然后通过编程实践在一个加权连通图中寻找最小生成树。这一过程不仅涉及算法的实现，还

包括对输入数据的处理、结果的验证和效率的分析，旨在深化对离散数学中图论概念的理解，并增强解决实际问题的编程能力。

Kruskal 算法的核心思想如下：

(1)假设图 G 是不连通的，对图的所有边按权值进行非降序排序；

(2)依次考虑排序后的边，如果当前边加入最小生成树 T 不会形成回路，则加入 T，否则，丢弃该边；

(3)持续此过程直至生成树 T 包含所有顶点或所有边都被考虑过；

(4)输出最小生成树 T。

5.2 实验设计

5.2.1 数据结构设计

在使用 Kruskal 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路网场景；

(2)便于实现 Kruskal 算法中的关键操作；

(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。

5.2.2 Edge 结构体的设计

5.2.2.1 概述

Edge 结构体是为图论算法（如最小生成树算法）中表示图的边而设计的。它用于表示图中的一条边，包括边的起点、终点和权重。此外，它通过重载<运算符来比较两条边的权重，使其能够用于需要边权重比较的算法中，例如在实现 Kruskal 算法时对边进行排序。

5.2.2.2 结构体定义

```
struct Edge {
    int src;
    int dst;
    int weight;
    bool operator<(const Edge& other) const { return this->weight < other.weight; }
};
```

5.2.2.3 数据成员

`int src`：边的起点

`int dst`：边的终点

`int weight`：边的权重

5.2.2.4 运算符重载

`bool operator<(const Edge& other) const { return this->weight < other.weight; }`

重载<运算符。它使得可以直接比较两个 `Edge` 实例。比较基于边的权重，如果当前边的权重小于另一条边的权重，则返回 `true`；否则返回 `false`。

5.2.3 MinimumSpanningTree 类的设计

5.2.3.1 概述

`MinimumSpanningTree`类是专为解决图论中的一个关键问题——寻找加权连通图的最小生成树而设计的。此类具体实现了 Kruskal 算法，在图的所有边中选取最小权重的边来构建生成树，同时避免形成任何环，从而达到总权重最小化的目标。它包括了初始化图结构、处理边的权重、执行 Kruskal 算法核心步骤以及提供最终结果的相关方法。此外，类中还包含了对并查集数据结构的实现，用于辅助实现 Kruskal 算法中对边的选择和环的检测。

5.2.3.2 类定义

```
class MinimumSpanningTree {
private:
    int vertex;
    int total;
    int** graph;
    std::vector<Edge> edges;
    std::vector<int> parent;
    std::vector<int> rank;
    int findSet(int i);
    void unionSets(int x, int y);
public:
    MinimumSpanningTree(int V);
    ~MinimumSpanningTree();
    int getTotal(void) const;
    void kruskalMST(void);
```

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

5.2.3.3 私有数据成员

`int vertex`：顶点数量，代表路网中的城市数

`int total`：最小生成树的总权重

`int** graph`：二维数组，用于存储路网中各城市间的道路距离

`std::vector<Edge> edges`：存储图中所有边的向量

`std::vector<int> parent`：并查集中每个元素的父节点

`std::vector<int> rank`：并查集中每个集合的秩（即树的高度）

5.2.3.4 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。

5.2.3.5 析构函数

`~MinimumSpanningTree();`

析构函数，释放动态分配的内存资源。

5.2.3.6 私有成员函数

`int findSet(int i);`

在并查集中查找并返回顶点 `i` 的根节点。

`void unionSets(int x, int y);`

在并查集中合并两个集合，这两个集合分别包含顶点 `x` 和 `y`。

5.2.3.7 公有成员函数

`int getTotal(void) const;`

返回当前最小生成树的总权重。

`void kruskalMST(void);`

使用 Kruskal 算法生成最小生成树并计算总权重。

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

设置图中两个顶点之间边的权重，并将该边添加到边集合中。

5.2.4 程序主体架构设计

程序主体架构设计为：

(1)初始化和创建路网：当程序开始执行时，首先进入 `main` 函数。在这里，程序首先执行初始化操作，包括清除屏幕并显示用户界面。接下来，程序请求用

户输入城市（节点）的数量，这是通过调用 inputInteger 函数实现的。用户输入后，使用这个数值来创建 MinimumSpanningTree 实例，此实例代表整个路网；

(2)输入边的权重：一旦路网初始化完成，程序接着引导用户输入任意两个城市（即图中的节点）之间的距离（权重）。这些权重通过 setWeight 方法设置到 MinimumSpanningTree 的实例中。对于所有城市对，都需要执行这个步骤，以确保图中的所有边都被考虑；

(3)构建最小生成树：输入完所有边的权重后，main 函数调用 kruskalMST 方法来构建最小生成树。此方法内部实现了 Kruskal 算法，它首先排序所有边，然后逐一选择边，构建无环的最小生成树；

(4)输出结果：构建完最小生成树后，程序通过 getTotal 方法获取并输出总权重，展示了构建的最小生成树的总成本。最后，main 函数中包含用户退出程序的选项。一旦用户选择退出，程序将清理所有资源并结束执行。

(5)异常处理：在整个程序中，还包括了对不同情况的异常处理，例如内存分配失败或用户输入无效数据。这确保了程序的健壮性和用户友好性。

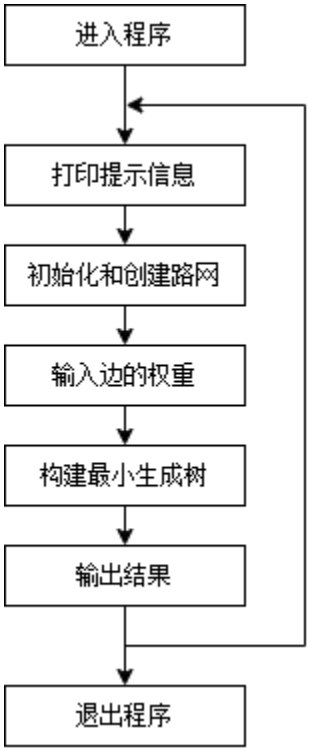

图 5.2.4.1 程序主体架构设计流程图

5.3 程序功能实现

5.3.1 输入城市个数与城市间距离功能的实现

程序通过调用 inputInteger 函数输入城市个数与城市间距离。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。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。

5.3.2 退出程序功能的实现

退出程序的功能是通过在 main 函数内部的一个循环结构实现的。此功能允许用户在完成操作后选择是否退出程序。

程序功能实现思路：

⑴程序使用 do-while 循环来反复执行程序和询问用户是否退出程序。在每次循环的开始，使用 system("cls")清除屏幕，以提供清晰的界面；

⑵在完成一轮操作后，程序会输出询问用户是否退出程序的提示：“是否退出程序［y/n]：”。此时，程序再次调用 inputLogicalValue 函数，这次用于接收用户的退出决定。参数被设置为'n'（不退出）和'y'（退出）；

⑶如果用户输入'y'（对应真值），则 inputLogicalValue 函数返回 true，导致 do-while 循环结束，程序随之退出。如果用户输入'n'（对应假值），inputLogicalValue 函数返回 false，do-while 循环继续，用户可以进行新一轮操作。

5.3.3 动态内存申请失败的异常处理功能的实现

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：

⑴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；

⑵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。

5.4 核心算法实现

5.4.1 并查集相关算法的实现

findSet 是一个并查集算法的核心部分，用于查找给定元素（这里是顶点 i）所属的集合的根节点。这在合并两个集合或者检查两个元素是否位于同一个集合中时非常关键。

当调用 findSet(i)时，算法检查 parent[i]是否等于 i，即该元素是否是其集合的根节点。如果不是根节点，则递归地调用 findSet 在并查集的树结构中向上追溯，直到找到根节点。在查找的过程中，实现了路径压缩，即 parent[i]被更新为其根节点，这有助于加快未来的查找操作。

unionSets 函数用于合并包含两个不同顶点 x 和 y 的两个集合。

首先，使用 findSet 函数找到 x 和 y 的根节点（即集合代表），分别称为 xroot 和 yroot。如果 xroot 和 yroot 相同，意味着 x 和 y 已经在同一个集合中，无需合并。如果 xroot 和 yroot 不同，则需要将两个集合合并。这是通过更新 parent 数组实现的，将一个集合的根节点指向另一个。合并时考虑到了秩（rank）的概念，即树的高度。总是将较小树的根指向较大树的根，以保持树的平衡，优化性能。

5.4.2 Kruskal 算法的实现

kruskalMST 函数实现了 Kruskal 算法，用于在加权图中构建最小生成树。kruskalMST 函数的实现思路如下：

⑴初始化 parent 和 rank 数组，为每个顶点设置父节点为其自身，并将秩置 0；

⑵将所有的边按权重进行排序；

⑶创建一个空的 result 向量来存储最小生成树中的边；

⑷遍历每条边，使用 findSet 检查当前边的两个顶点是否属于不同的集合（即检查是否会形成环）；

⑸如果是，将该边添加到 result 中，并使用 unionSets 合并两个集合；

⑹这个过程一直持续，直到加入了 vertex-1 条边或考虑了所有的边；

⑺通过累加 result 中边的权重，更新 total 来得到最小生成树的总权重；

⑻最后，可以打印或返回最小生成树的边和总权重。

6 实验数据分析

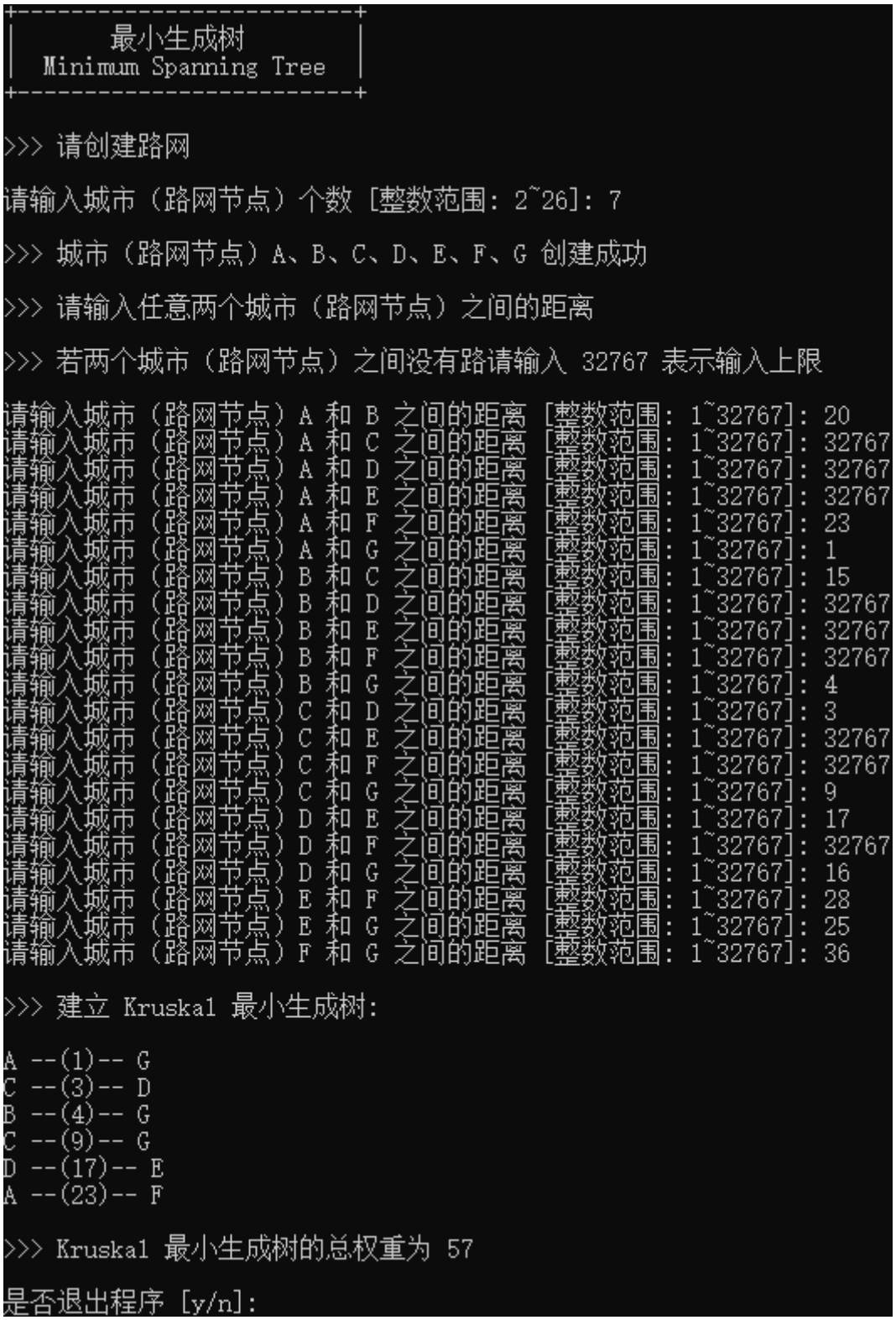

图 6.1 实验数据 1

实验数据分析 1：创建路网并输入城市之间的距离，可以建立 Kruskal 最小

生成树。程序成功建立 Kruskal 最小生成树，并输出 Kruskal 最小生成树及其总权重（见图 2.1 路网赋权图）。

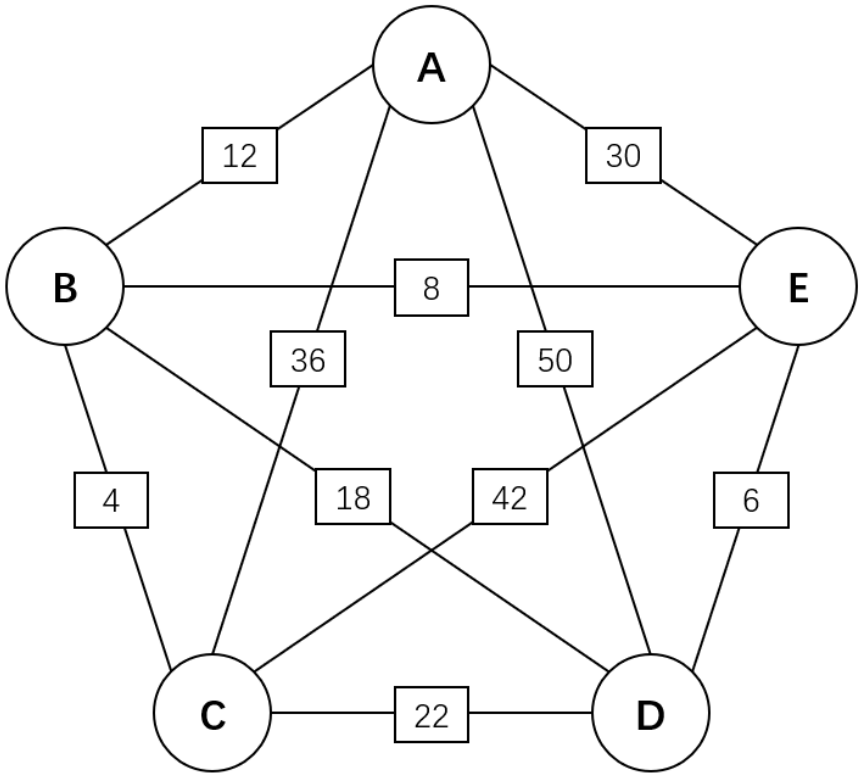

图 6.2 路网节点连接模拟图（数字表示道路距离）

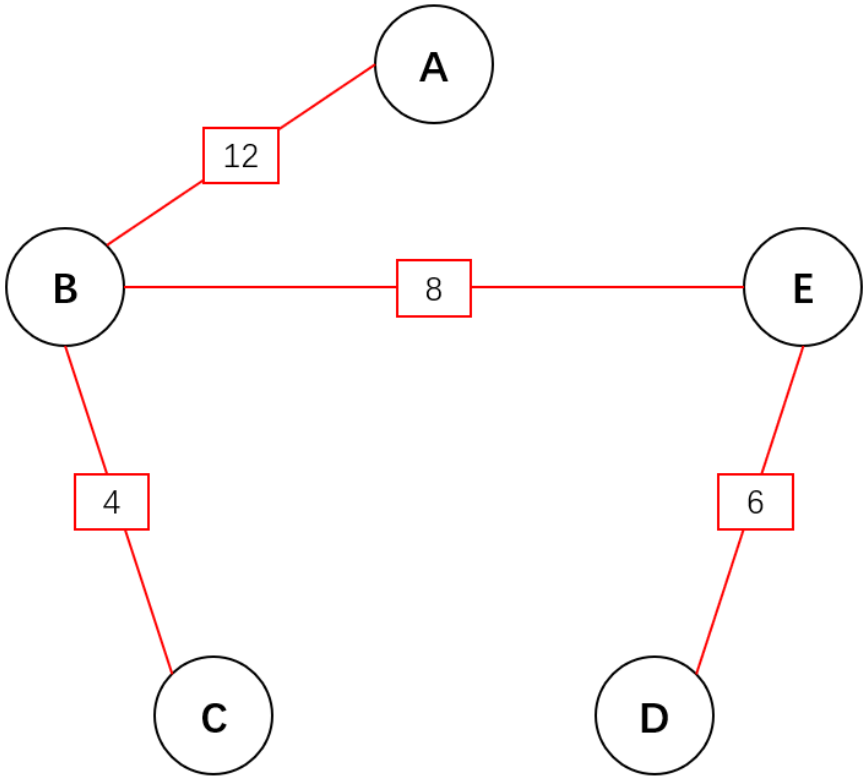

图 6.3 Kruskal 最小生成树示意图

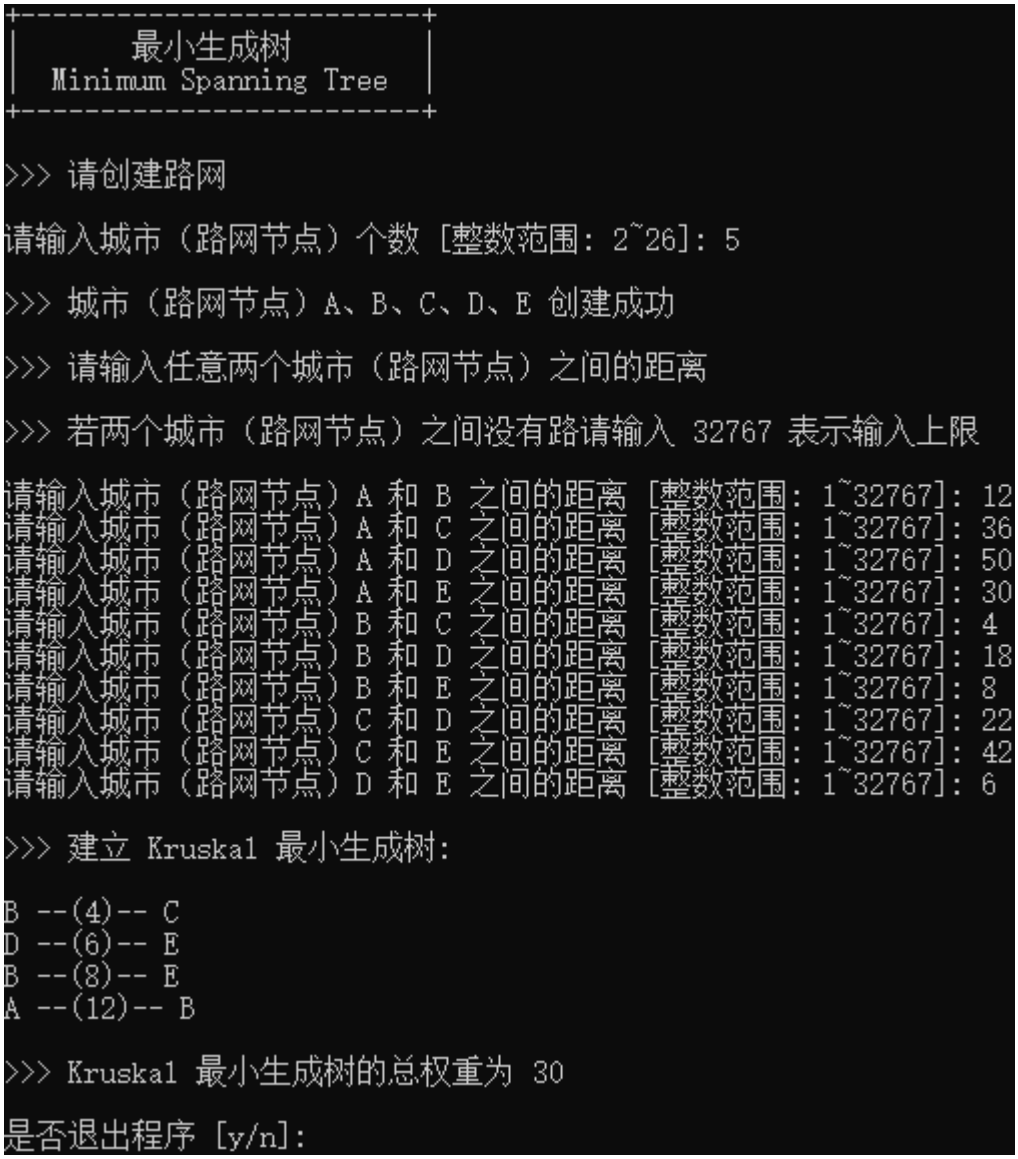

图 6.4 实验数据 2

实验数据分析 2：创建路网并输入城市之间的距离，可以建立 Kruskal 最小生成树。程序成功建立 Kruskal 最小生成树，并输出 Kruskal 最小生成树及其总权重（见图 6.2 路网节点连接模拟图、图 6.3 Kruskal 最小生成树示意图）。

图 6.5 实验数据 3（输入超过上下限的整数或浮点数）

图 6.6 实验数据 3（输入字符或字符串）

实验数据分析 3：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，并要求用户重新输入城市个数。

图 6.7 实验数据 4

实验数据分析 4：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，并要求用户重新输入城市间距离。

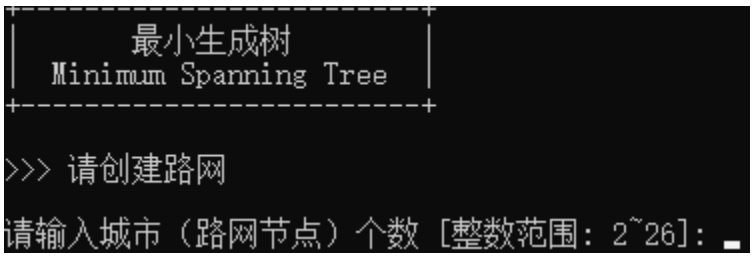

图 6.8 实验数据 5

实验数据分析 5：当输入 n 时，程序清空屏幕并执行下一次最小生成树的运算。

图 6.9 实验数据 6

实验数据分析 6：当输入 y 时，程序退出。

7 实验心得

在完成最小生成树实验后，我深刻体会到了理论与实践结合的重要性。本次实验中，我不仅复习了最小生成树的理论知识，特别是 Kruskal 算法的原理和实现，还通过编程实践将这些理论应用到了实际问题解决中。

实验过程中，我面临的最大挑战是将理论转化为实际的代码。设计数据结构以表示图，理解并实现并查集数据结构来支持 Kruskal 算法，这些都需要深入的理解和细心的编程。这个过程让我更加熟悉了 C++语言，尤其是面向对象编程的思想，如类的使用、数据封装和函数重载等。

通过处理输入数据和异常情况，我认识到了编写健壮且用户友好的程序的重要性。程序中的每一个细节，如输入验证和错误处理，都是确保程序平稳运行的关键。

看到自己的程序能够成功地解决实际问题，计算给定城市的最小生成树总权重，让我感到非常满足。这不仅证明了我的编程能力，也加深了我对离散数学中图论部分的理解。我相信，这次实验的经验将对我未来在计算机科学领域的学习和研究大有裨益。

8 程序源文件

```
/*****************************************************************
 * Project Name:  Assignment_4
 * File Name:     assignment_4.cpp
 * File Function: 最小生成树
 * Author:        Jishen Lin (林继申)
 * Update Date:   2023/12/22
 *****************************************************************/

#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
#include <limits>
#include <conio.h>
#include <vector>
#include <algorithm>
```

```
/* Macro definitions */
#define MEMORY_ALLOCATION_ERROR -1
#define INF INT_MAX

/* Define Edge structure */
struct Edge {
    int src;
    int dst;
    int weight;
    bool operator<(const Edge& other) const { return this->weight <
other.weight; }
};

/* Define MinimumSpanningTree class */
class MinimumSpanningTree {
private:
    int vertex;
    int total;
    int** graph;
    std::vector<Edge> edges;
    std::vector<int> parent;
    std::vector<int> rank;
    int findSet(int i);
    void unionSets(int x, int y);
public:
    MinimumSpanningTree(int V);
    ~MinimumSpanningTree();
    int getTotal(void) const;
    void kruskalMST(void);
    void setWeight(int src, int dst, int weight);
};

/*
 * Function Name:    findSet
 * Function:         Find in Union-Find
 * Input Parameters: int i
 * Notes:            Class external implementation of member functions
 */
int MinimumSpanningTree::findSet(int i)
{
    if (parent[i] != i)
        parent[i] = findSet(parent[i]);
    return parent[i];
}
```

```
/*
 * Function Name:    unionSets
 * Function:         Union in Union-Find
 * Input Parameters: int x
 *                   int y
 * Notes:            Class external implementation of member functions
 */
void MinimumSpanningTree::unionSets(int x, int y)
{
    int xroot = findSet(x), yroot = findSet(y);
    if (rank[xroot] < rank[yroot])
        parent[xroot] = yroot;
    else if (rank[xroot] > rank[yroot])
        parent[yroot] = xroot;
    else {
        parent[yroot] = xroot;
        rank[xroot]++;
    }
}

/*
 * Function Name:    MinimumSpanningTree
 * Function:         Constructed function
 * Input Parameters: int V
 * Notes:            Class external implementation of member functions
 */
MinimumSpanningTree::MinimumSpanningTree(int V)
{
    vertex = V;
    total = 0;
    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);
        }
    }
```

```
    for (int i = 0; i < vertex; i++)
        for (int j = 0; j < vertex; j++)
            graph[i][j] = 0;
}

/*
 * Function Name:    ~MinimumSpanningTree
 * Function:         Destructor
 * Notes:            Class external implementation of member functions
 */
MinimumSpanningTree::~MinimumSpanningTree()
{
    for (int i = 0; i < vertex; i++)
        delete[] graph[i];
    delete[] graph;
}

/*
 * Function Name:    getTotal
 * Function:         Get the total distance
 * Input Parameters: void
 * Return Value:     total distance
 */
int MinimumSpanningTree::getTotal(void) const
{
    return total;
}

/*
 * Function Name:    kruskalMST
 * Function:         Generate minimum spanning tree using Kruskal
algorithm
 * Input Parameters: void
 * Return Value:     void
 */
void MinimumSpanningTree::kruskalMST(void)
{
    /* Initialize parent and rank arrays for Union-Find */
    parent.resize(vertex);
    rank.resize(vertex, 0);
    for (int i = 0; i < vertex; ++i) {
        parent[i] = i;
    }
```

```
    /* Sort all edges in ascending order of their weights */
    std::sort(edges.begin(), edges.end());
    std::vector<Edge> result; // Stores the edges in the MST
    int count = 0; // Keeps track of the number of edges added to the
MST

    /* Iterate through all edges */
    for (const auto& edge : edges) {
        int x = findSet(edge.src); // Find the set of the source vertex
        int y = findSet(edge.dst); // Find the set of the destination
vertex
        if (x != y) {
            result.push_back(edge);
            unionSets(x, y);
            count++;
            if (count == vertex - 1)
                break;
        }
    }

    /* Print the edges in the MST */
    for (const auto& edge : result) {
        std::cout << static_cast<char>(edge.src + 'A') << " --(" <<
edge.weight << ")-- " << static_cast<char>(edge.dst + 'A') <<
std::endl;
        total += edge.weight;
    }
}

/*
 * Function Name:    setWeight
 * Function:         Set the weight of a specific edge
 * Input Parameters: int src
 *                   int dst
 *                   int weight
 * Return Value:     void
 */
void MinimumSpanningTree::setWeight(int src, int dst, int weight)
{
    graph[src][dst] = weight;
    graph[dst][src] = weight;
    edges.push_back({ src, dst, weight });
}
```

```
/*
 * Function Name:    inputInteger
 * Function:         Input an integer
 * Input Parameters: int lowerLimit
 *                   int upperLimit
 *                   const char* prompt
 * Return Value:     an integer
 */
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');
        }
    }
}

/*
 * Function Name:    inputLogicalValue
 * Function:         Input logical value
 * Input Parameters: char falseValue
 *                   char trueValue
 * Return Value:     true / false
 */
bool inputLogicalValue(char falseValue = '0', char trueValue = '1')
{
    while (true) {
        char optn = _getch();
        if (optn == 0 || optn == -32)
```

```
            optn = _getch();
        else if (optn == falseValue || optn == trueValue) {
            std::cout << optn << std::endl << std::endl;
            return optn == falseValue ? false : true;
        }
    }
}

/*
 * Function Name:    main
 * Function:         Main function
 * Return Value:     0
 */
int main()
{
    do {
        /* System entry prompt */
        system("cls");
        std::cout << "+-------------------------+" << std::endl;
        std::cout << "|       最小生成树        |" << std::endl;
        std::cout << "|  Minimum Spanning Tree  |" << std::endl;
        std::cout << "+-------------------------+" << std::endl;

        /* Establish road network */
        std::cout << std::endl << ">>> 请创建路网" << std::endl <<
std::endl;
        int vertex = inputInteger(2, 'Z' - 'A' + 1, "城市（路网节点）个数
");
        MinimumSpanningTree MST(vertex);
        std::cout << std::endl << ">>> 城市（路网节点）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 cities */
        std::cout << std::endl << ">>> 请输入任意两个城市（路网节点）之间的
距离" << std::endl;
        std::cout << std::endl << ">>> 若两个城市（路网节点）之间没有路请输
入 " << SHRT_MAX << " 表示输入上限" << std::endl << std::endl;
        for (int i = 0; i < vertex; i++)
            for (int j = i + 1; j < vertex; j++) {
                char tmp[64];
                sprintf(tmp, "城市（路网节点）%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 Kruskal algorithm */
        std::cout << std::endl << ">>> 建立 Kruskal 最小生成树:" <<
std::endl << std::endl;
        MST.kruskalMST();
        std::cout << std::endl << ">>> Kruskal 最小生成树的总权重为 " <<
MST.getTotal() << std::endl << std::endl;

        /* Whether to exit the program */
        std::cout << "是否退出程序 [y/n]: ";
    } while (!inputLogicalValue('n', 'y'));
    return 0;
}
```