《离散数学》课程实验报告 4 最小生成树

2254321 陆诚彬

1. 实验用例

如下图所示的赋权图表示某七个城市，预先计算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。

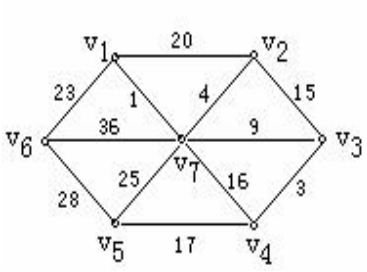

图 1 七个城市赋权图

2. 实验内容

本实验的目的是实现一个程序，用于构建一个最小生成树。最小生成树是一种在不连通图中找到可以连接所有顶点且总权重最小的边集合的算法。该程序将适用于表示城市间通信道路的赋权图。本实验通过实现 Kruskal 算法来寻找最小生成树，从而达到在保持城市间通信的同时，使总造价最小的目的。

3. 解题思路

Kruskal 算法是求解最小生成树的经典方法。它的基本思想是：

1) 首先将图中的所有边按照权重从小到大排序。
2) 初始化一个空的最小生成树。
3) 遍历排序后的边，对于每一条边，检查它加入最小生成树后是否会形成回路。如果不会，则将其加入最小生成树；否则，丢弃该边。
4) 当所有的顶点都被包含在最小生成树中时，算法结束。

本实验中实现了一个简单的 Kruskal 算法。程序首先读取顶点数和边数，然后读取每条边及其权重。使用并查集数据结构来检查加入新边是否会在生成树中形成回路，并据此来决定是否加入该边。

4. 数据结构设计

Edge 结构体：用于表示图中的一条边，包含起点 u、终点 v 和权重 weight。重载 < 操作符以便能够根据权重对边进行排序。

```
// 定义边结构
struct Edge {
    int u, v, weight;
    bool operator<(const Edge& e) const {
        return weight < e.weight;
    }
};
```

数组 parent：一个整型数组，用作并查集的实现。数组的索引表示顶点，值表示该顶点的父顶点。并查集用于快速判断两个顶点是否属于同一集合，从而检测加入新边是否会形成回路。

主函数中的逻辑：首先读取顶点和边的信息，然后对所有边按权重排序。遍历每条边，使用并查集检查是否可以加入该边而不形成回路。如果可以，则加入该边，并更新并查集和总权重。

5. 项目实现

1) 定义边结构和集合操作函数：

创建一个 Edge 结构，包含两个顶点 u 和 v 以及边的权重 weight。
定义 findSet 函数，用于查找一个元素所在的集合的代表。
定义 unionSet 函数，用于合并两个集合。

2) 输入顶点和边信息：

从用户那里获取顶点数（n）和边数（m）。
创建一个 Edge 类型的向量来存储所有边及其权重。

3) 初始化集合：

初始化一个数组 parent，用来代表每个顶点所在集合的代表。初始时，每个顶点自己就是自己集合的代表。

4) 对边进行排序并选择边构建 MST：

使用 std::sort 对所有边按权重进行排序。
遍历排序后的边，使用 findSet 来检查当前边的两个顶点是否属于同一个集合。如果不是，说明这条边不会产生循环，可以添加到 MST 中，并使用 unionSet 合并两个集合。

5) 输出结果：

在选择边的过程中累加其权重，以计算最小生成树的总权重。
输出每条被选中的边和最终的总权重。

6. 设计小结

在本实验中，成功实现了 Kruskal 算法，用于构建最小生成树（MST）。这个算法在处理图论问题，尤其是在网络设计（如城市通信网络）中寻找最小成本路径时具有重要意义。我们的实现重点放在了算法的核心部分：边的结构定义、并查集的实现以及如何有效地选择构成最小生成树的边。

通过对边进行排序和逐一检查，确保了在构建最小生成树的过程中，每次添加的都是成本最低的边，同时避免了回路的生成。这个方法不仅提高了算法的效率，也确保了总成本的最小化。整个过程的实现强调了数据结构（如边的数组和并查集）在算法设计中的重要性。

7. 实验心得

通过这次实验，我深刻理解了最小生成树和 Kruskal 算法的概念和重要性。在实际编程过程中，我学会了如何将理论应用到实际的问题解决中，特别是如何运用数据结构来优化算法的执行效率。

此外，我也认识到了代码的结构和可读性的重要性。在编写程序的过程中，我努力保持代码的整洁和组织性，确保算法的每个步骤都清晰明了。这不仅有助于调试过程中的问题解决，也使得代码更容易被他人理解和维护。

总的来说，这个项目不仅加深了我对图论和算法设计的理解，也提高了我的编程技能和问题解决能力。通过亲自实现复杂的算法，我感到自己在作为一名计算机科学家的道路上又向前迈进了一步。

8. 项目源代码

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

#define N 100 // 定义最大顶点数

// 定义边结构
struct Edge
{
    int u, v, weight; // 边的两个顶点 u, v 和边的权重 weight
    // 重载 < 操作符，便于后续按权重对边进行排序
    bool operator<(const Edge &e) const
    {
        return weight < e.weight;
    }
};

int parent[N]; // 并查集数组，用于维护顶点的集合信息

// 查找集合的代表
int findSet(int i)
{
    // 路径压缩优化：直接将 i 的父节点设置为集合代表
    if (i != parent[i])
        parent[i] = findSet(parent[i]);
    return parent[i];
}

// 合并两个集合
void unionSet(int u, int v)
{
    // 将一个集合的代表指向另一个集合的代表
    parent[findSet(u)] = findSet(v);
}

int main()
{
    int n, m; // n 代表顶点数，m 代表边数
    cout << "请输入顶点数和边数: ";
    cin >> n >> m;

    vector<Edge> edges(m); // 存储所有边的向量
    cout << "请输入边和它们的权重（格式：起点 终点 权重）: \n";

    // 读取边的信息
    for (int i = 0; i < m; ++i)
    {
        cin >> edges[i].u >> edges[i].v >> edges[i].weight;
    }

    // 初始化并查集，每个顶点初始时都是自己的代表
    for (int i = 1; i <= n; ++i)
    {
        parent[i] = i;
```

```
    }

    // 对边按权重进行排序
    sort(edges.begin(), edges.end());

    int totalWeight = 0; // 记录最小生成树的总权重
    // 遍历所有边，构建最小生成树
    for (const auto &e : edges)
    {
        // 如果两个顶点属于不同集合，则选择这条边
        if (findSet(e.u) != findSet(e.v))
        {
            cout << "选择边: " << e.u << " - " << e.v << " (权重: " << e.weight << ")\n";
            totalWeight += e.weight;
            unionSet(e.u, e.v); // 合并两个集合
        }
    }

    // 输出最小生成树的总权重
    cout << "最小生成树的总权重: " << totalWeight << endl;

    system("pause");

    return 0;
}
```