同济大学

离散数学说明文档

--最小生成树

姓　　名：胡正华

学　　号：2353741

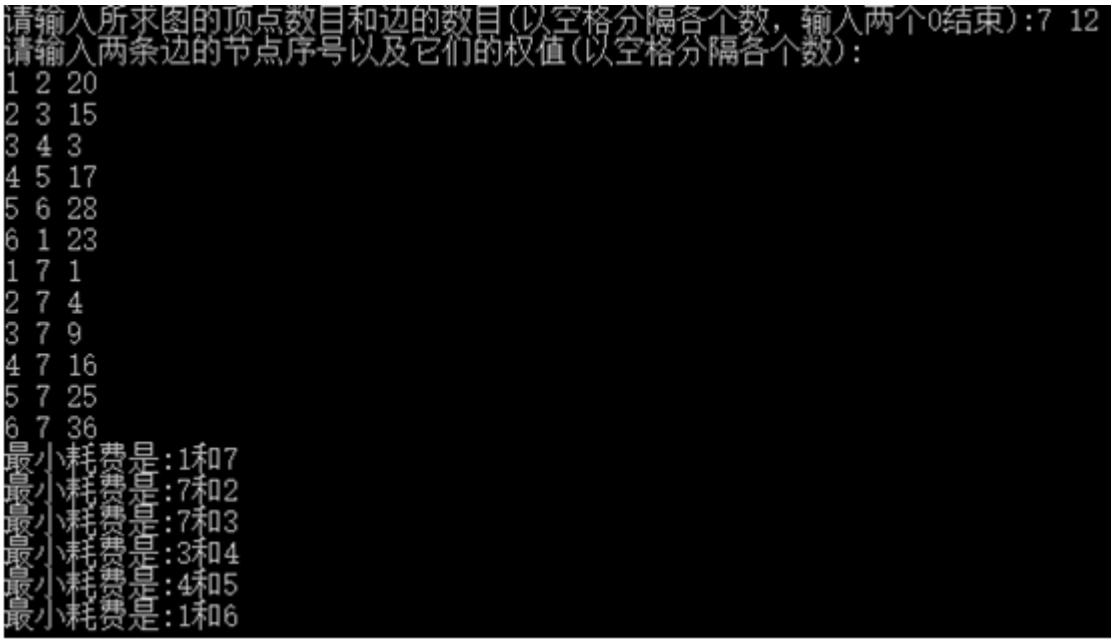
任课教师：李　冰

1. 题目简介

1.1 题目要求

用Prim算法求解给出图的最小生成树。

1.2 题目样例



2. 解题思路

本实验的目的是通过 Prim 算法实现加权无向图的最小生成树，从而找到一种连接所有顶点且总权值最小的边集合。解题思路如下：

2.1 问题分析

一个加权无向图由顶点集合 V 和边集合 E 组成，每条边 e=(u,v,w)e = (u, v, w)e=(u,v,w) 表示顶点 u 和 v 之间的连接，权值为 w。最小生成树的目标是找到一个包含 |V|−1|V| - 1|V|−1 条边的子图，使其连接所有顶点且权值总和最小。

Prim 算法是一种贪心算法，它通过逐步扩展已选中的顶点集，不断选择与当前顶点集权值最小的边，直至构建出完整的最小生成树。

2.2 算法设计

2.2.1 初始化：

从图中任选一个顶点作为起点（本实验默认为顶点 1）。

初始化一个数组 key 用于存储每个顶点到已选中顶点集的最小权值；key[start] = 0 表示起点权值为 0。

用数组 p 记录每个顶点的父节点，用于输出生成树的结构。

2.2.2 执行步骤：

在所有未加入生成树的顶点中，找到与已选顶点集权值最小的顶点，将其加入生成树。

更新其他未加入顶点的最小权值。如果某个顶点通过新加入的顶点可以得到更小的权值，则更新该顶点的权值及其父节点。

重复上述步骤，直至所有顶点都加入生成树。

2.3 算法实现

本实验采用邻接矩阵存储图，矩阵元素 tb[i][j]tb[i][j]tb[i][j] 表示顶点 iii 和 jjj 之间的权值，若两顶点之间无边则权值为无穷大（使用 INT_MAX 表示）。

通过三重循环实现 Prim 算法：

（1）外层循环控制生成树的构建过程，最多执行 |V|-1|V| - 1|V|-1 次。

（3） 内层循环找到当前未加入生成树且与已选顶点集权值最小的顶点。

（3） 更新每个未加入顶点的最小权值和父节点。

2.4复杂度分析

使用邻接矩阵存储图时，Prim 算法的时间复杂度为 O(n^2)，其中 n 为图的顶点数。这是因为每次选出权值最小的顶点需要遍历所有顶点，更新邻接矩阵的权值也需要遍历所有顶点。

3. 代码实现

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

#define N 100
int p[N], key[N], tb[N][N];

void prim(int start, int n) {
    int i, j;
    int minCost;
    for (i = 1; i <= n; i++) {
        p[i] = start;         // 初始所有顶点的父节点为起点
        key[i] = tb[start][i]; // 起点到其他顶点的权值
    }
    key[start] = 0; // 起点自身的权值设为0，标记为已加入MST
```

```
    for (i = 1; i < n; i++) { // 循环n-1次，选出n-1条边
        int min = INT_MAX;
        int v = -1;

        // 找到未加入MST且权值最小的顶点
        for (j = 1; j <= n; j++) {
            if (key[j] > 0 && key[j] < min) {
                v = j;
                min = key[j];
            }
        }

        if (v == -1) break; // 如果找不到有效的顶点，结束循环
        cout << "最小耗费是: " << p[v] << "和" << v << endl;
        key[v] = 0; // 标记顶点v已加入MST

        // 更新其他顶点到MST的权值
        for (j = 1; j <= n; j++) {
            if (tb[v][j] < key[j]) {
                p[j] = v;         // 更新父节点为v
                key[j] = tb[v][j]; // 更新权值
            }
        }
    }
}
```

```
int main() {
    int n, m;
    int i, j;
    int u, v, w;

    cout << "请输入所求图的顶点数目和边的数目(以空格分隔，输入两个0结束):";
    while (cin >> n >> m) {
        if (m != 0 && n != 0) {
            // 初始化邻接矩阵
            for (i = 1; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                    tb[i][j] = INT_MAX; // 初始权值为无穷大
                }
            }

            cout << "请输入两条边的节点序号以及它们的权值(以空格分隔各个数):" << endl;
            for (i = 0; i < m; i++) {
                cin >> u >> v >> w;
                tb[u][v] = tb[v][u] = w; // 无向图，边对称
            }

            prim(1, n); // 从顶点1开始Prim算法
            cout << endl;
            cout << "请输入所求图的顶点数目和边的数目(以空格分隔，输入两个0结束):" <<
endl;   }
        else if (n == 0 && m == 0) {
            return 0; // 输入0 0 结束程序
        }
    }
    return 0;
}
```

4. 心得体会

通过本次实验，我深入学习并实践了Prim算法的原理和实现方法，进一步加深了对最小生成树问题的理解。以下是我的几点心得体会：

1. **算法的直观性与贪心策略**
   在实现 Prim 算法的过程中，我深刻体会到贪心策略在解决最小生成树问题中的高效性。每次选择权值最小的边虽然是局部最优，但最终能够保证全局最优解，这体现了贪心算法的核心思想。通过编写代码，我进一步理解了该策略如何逐步扩展当前生成树，确保构建完整的最小生成树。
2. **邻接矩阵的使用与空间效率**
   本实验采用邻接矩阵表示图结构，在小规模图的情况下非常方便。通过邻接矩阵的操作，我能够快速访问任意两点之间的权值。但是，我也认识到邻接矩阵的空间复杂度

为 O(n^2)，当图的顶点数目较大且稀疏时，可能会导致较大的内存浪费。实际工程中，对于稀疏图可以考虑使用邻接表优化。

3. **复杂度分析的必要性**

   在分析 Prim 算法的时间复杂度时，我了解到其基于邻接矩阵实现的复杂度为 O(n^2)。当顶点数较小时，这种实现方法性能良好，但随着规模增大，其效率会显著降低。通过对复杂度的分析，我也认识到优化算法和选择合适的数据结构的重要性，例如利用优先队列实现的 Prim 算法能够将复杂度降低到 O(E \log V)。

总的来说，本次实验帮助我强化了对算法思想的掌握，提升了代码实现能力和调试能力，同时让我更加重视理论与实践相结合的重要性。这次实验让我受益匪浅，我也将在今后的学习中继续探索更高效的算法和数据结构，为解决复杂问题积累更多的经验和技能。