|  |  |
| --- | --- |
| 姓名 | 单禹嘉 |
| 学号 | 2023215177 |

|  |  |
| --- | --- |
| 实验成绩 |  |

华中师范大学计算机学院

实验报告书

课程名称：数据结构

主讲教师：沈显君

课程编号：

班级：2306

1. 实验目的

目的：

用图实现管道铺设的最佳方案要在某个城市的n个居民区之间铺设煤气管道，则在这n个居民区之间只要铺设n-1条管道即可。假设任意两个居民区之间都可以假设管道，但由于地理环境的不同，所需经费不同。选择最优的施工方案使得中投资尽可能少

1. 实验要求

1）选择合理的数据类型表示煤气管道；

2）完整图的基本操作；

3） 对算法进行算法分析

1. 实验环境

VSCode

1. 系统描述

• 数据结构设计：

1. 停车场（栈）： 存放已进入停车场的车辆，容量有限。

2. 便道（队列）： 存放等待进入停车场的车辆，容量无限。

• 类设计：

• Car：表示车辆，记录车牌号和进入停车场的时间。

• Stack：实现停车场的栈结构，支持车辆进出。

• Queue：实现便道的队列结构，支持车辆排队等待需求分析

1. 集合表示：

• 使用线性链表表示集合，链表中的每个节点存储一个整数。

2. 基本运算：

• 并集运算： 返回两个集合的并集，包含所有不重复的元素。

• 交集运算： 返回两个集合的交集，包含在两个集合中都存在的元素。

• 差集运算： 返回集合 A 相对集合 B 的差集，即只存在于集合 A 而不在集合 B 中的元素。

1. 附录（源代码）与说明

#include <iostream>

#include <vector>

#include <algorithm>

#include <limits>

using namespace std;

// 边结构体

struct Edge {

int u, v, weight;

Edge(int u, int v, int weight) : u(u), v(v), weight(weight) {}

};

// 并查集结构体

class DSU {

public:

vector<int> parent, rank;

DSU(int n) {

parent.resize(n + 1);

rank.resize(n + 1, 0);

for (int i = 1; i <= n; i++) {

parent[i] = i;

}

}

int find(int x) {

if (parent[x] != x) {

parent[x] = find(parent[x]); // 路径压缩

}

return parent[x];

}

void unite(int x, int y) {

int rootX = find(x);

int rootY = find(y);

if (rootX != rootY) {

if (rank[rootX] > rank[rootY]) {

parent[rootY] = rootX;

} else if (rank[rootX] < rank[rootY]) {

parent[rootX] = rootY;

} else {

parent[rootY] = rootX;

rank[rootX]++;

}

}

}

};

// Kruskal 算法

int kruskal(int n, vector<Edge>& edges, vector<Edge>& mst) {

sort(edges.begin(), edges.end(), [](Edge& a, Edge& b) {

return a.weight < b.weight; // 按权重升序排序

});

DSU dsu(n);

int totalWeight = 0;

for (Edge& edge : edges) {

if (dsu.find(edge.u) != dsu.find(edge.v)) {

dsu.unite(edge.u, edge.v);

mst.push\_back(edge);

totalWeight += edge.weight;

if (mst.size() == n - 1) break; // 最小生成树完成

}

}

return totalWeight;

}

// 输入验证函数

bool validateInput(int& n, int& m) {

if (n < 2) {

cout << "居民区数量必须大于等于2，请重新输入。\n";

return false;

}

if (m < n - 1 || m > n \* (n - 1) / 2) {

cout << "边的数量必须在 [" << n - 1 << ", " << n \* (n - 1) / 2 << "] 范围内，请重新输入。\n";

return false;

}

return true;

}

int main() {

int n, m;

while (true) {

cout << "输入居民区数量 n 和管道候选路径数量 m: ";

cin >> n >> m;

if (cin.fail() || !validateInput(n, m)) {

cin.clear(); // 清除错误状态

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // 丢弃输入缓冲区中的内容

continue; // 提示用户重新输入

}

break;

}

vector<Edge> edges;

cout << "输入每条路径的起点、终点和权重 (u v w):\n";

for (int i = 0; i < m; i++) {

int u, v, w;

while (true) {

cout << "路径 " << i + 1 << ": ";

cin >> u >> v >> w;

if (cin.fail() || u <= 0 || v <= 0 || u > n || v > n || w <= 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "输入无效，请确保起点和终点在 1 到 " << n << " 范围内，权重为正数。\n";

continue;

}

edges.emplace\_back(u, v, w);

break;

}

}

vector<Edge> mst;

int totalWeight = kruskal(n, edges, mst);

cout << "\n=== 最小生成树结果 ===\n";

cout << "最小生成树总费用: " << totalWeight << "\n";

cout << "选取的路径:\n";

for (Edge& edge : mst) {

cout << edge.u << " - " << edge.v << " : " << edge.weight << "\n";

}

return 0;

}