哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：图型结构及其应用

实验题目：最小生成树算法

实验日期：2023年10月26日

班级：2237102

学号：2022111654

姓名：李宸

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 李秀坤 |

**一、实验目的**

本实验主要要求从文件中正确读取图，并以适当形式存储，利用prim算法和kruskal算法建立最小生成树。

1. **实验要求及实验环境**

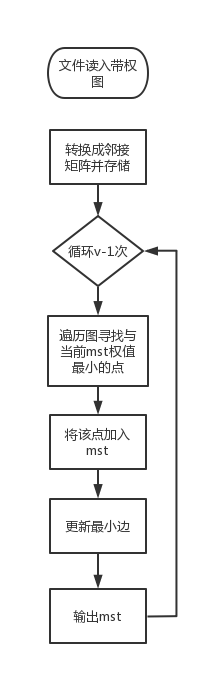
实验要求：

1. 选择并建立加权连通图的存储结构，实现求解加权连通图的Prim算法，并输出连接各顶点的最小生成树。
2. 利用并查集，实现求解加权连通图的Kruskal算法，并输出连接各顶点的最小生成树。
3. 以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点不少于10个，边不少于13个。

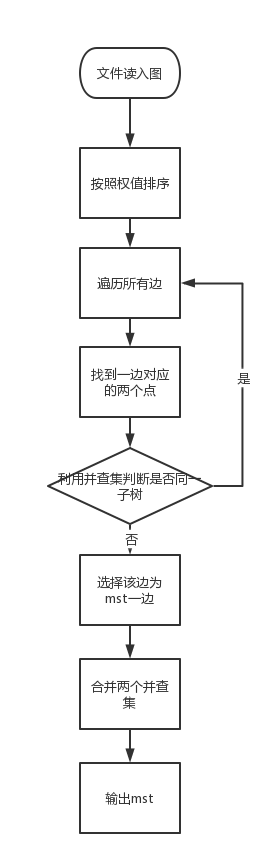
实验环境：win11 vscode1.71.2version

**三、设计思想**（本程序中的用到的所有数据类型的定义，各程序模块之间的调用关系、核心算法的流程图或主要步骤）

1．逻辑设计

Prim整体流程图如下

Kruskal整体流程图如下：



核心算法步骤：

prim算法：

①首先从集合V中任取一顶点(如顶点v0)放入集合U中。这时U={v0}，

边集TE={ }

② 然后找出权值最小的边(u，v) ，且u∈U，v∈(V-U)，将边加入TE，

并将顶点v加入集合U

③ 重复上述操作直到U=V为止。这时TE中有n-1条边，T=(U，TE就

是G的一棵最小生成树

找到连接U和V-U的最短边：

利用MST性质，可以用下述方法构造候选最短边集：对于V-U中的每个顶点，保存从该顶点到U中的各顶点的最短边。

Kruskal算法：

设无向连通网为G＝(V, E)，令G的最小生成树为T＝(U, TE)，其初

态为U＝V，TE＝{ }，

然后，按照边的权值由小到大的顺序，依次考察G的边集E中的各条

边。

若被考察的边连接的是两个不同连通分量，则将此边作为最小生成

树的边加入到T中，同时把两个连通分量连接为一个连通分量；

若被考察的边连接的是同一个连通分量，则舍去此边，以免造成回路，

如此下去，当T中的连通分量个数为1时，此连通分量便为G的一棵

最小生成树。

1. 物理设计

函数表如下：

Prim:

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 功能 |
| primMST | void | 利用prim算法建立最小生成树并输出 |
| minCost | int | 用于找到最小权值的顶点 |
| main | int | 主函数，实现文件读取等 |

Kruskal:

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 功能 |
| find | int | 查找操作，用于查找节点所属的集合 |
| Union | void | 合并操作，用于将两个集合合并为一个集合 |
| cmp | int | 按照边的权重进行升序排序 |
| Kruskal | void | Kruskal算法，用于构建加权连通图的最小生成树 |
| main | int | 主函数，实现文件读取等 |

变量表如下：

Prim:

|  |  |  |
| --- | --- | --- |
| 变量 | 数据类型 | 功能 |
| n | int | 顶点数 |
| edge | int | 边数 |
| file | FILE \* | 进行文件操作 |
| start | int | 起始点 |
| end | int | 终点 |
| weight | int | 权值 |
| graph[n][n] | int | 邻接矩阵 |
| min | int | 最小权值 |
| min\_index | int | 最小权值的点 |
| parent[n] | int | 存储最小生成树的父节点 |
| cost[n] | int | 存储与最小生成树相连的最小权重边的权值 |
| mstSet[n] | int | 记录节点是否包含在最小生成树中 |
| count | int | 记录循环的次数 |

Kruskal:

|  |  |  |
| --- | --- | --- |
| 变量 | 数据类型 | 功能 |
| V | int | 顶点数 |
| E | int | 边数 |
| file | FILE \* | 进行文件操作 |
| edges[E] | Edge | 边的集合 |
| xroot | int | 并查集根节点 |
| yroot | int | 并查集根节点 |
| subsets | Subset\* | 并查集 |
| result[V] | Edge | 最小生成树 |

本实验在kruskal算法中还设计了两个结构体：

1. Edge结构体：定义边的结构体

包含int src, dest, weight;分别表示起点，终点，权重

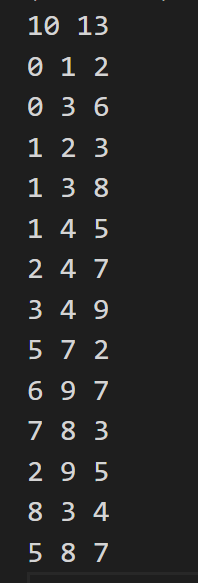
1. Subset结构体：表示哈夫曼树的节点

包含int parent;父节点

int rank;高度

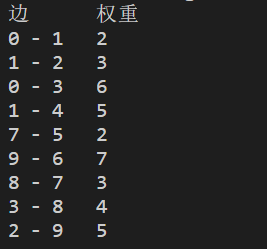
1. **测试结果**

测试数据：（graph.txt）

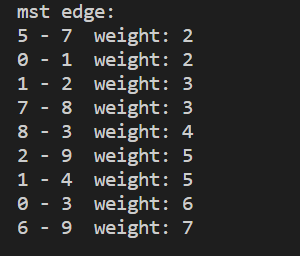


测试结果：(终端输出)

Prim:



Kruakal:



**五、经验体会与不足**

通过本次实验，我充分掌握了prim算法和kruskal算法建立最小生成树的过程，在此基础上可以尝试利用堆结构（优先级队列）改进和优化Prim算法，实现改进和优化的Prim算法，以及设计并实现其他最小生成树算法。例如，管梅谷破圈算法、Sollin（Boruvka）算法。

**六、附录：源代码（带注释）**

**prim算法：**

#include <stdio.h>

#include <limits.h>

#define INF INT\_MAX

int n;//顶点数

// 函数用于找到最小权值的顶点

int minCost(int cost[], int mstSet[]) {

int min = INF, min\_index;

for (int w = 0; w < n; w++) {

if (!mstSet[w] && cost[w] < min) {

min = cost[w];

min\_index = w;

}

}

return min\_index;

}

void primMST(int graph[n][n]) {

int parent[n]; // 存储最小生成树的父节点

int cost[n]; // 存储与最小生成树相连的最小权重边的权值

int mstSet[n]; // 记录节点是否包含在最小生成树中

// 初始化所有顶点的权值为无穷大，mstSet为false

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

cost[i] = INF;

mstSet[i] = 0;

}

// 将第一个顶点作为起始顶点

cost[0] = 0;

parent[0] = -1;

// 循环V-1次的最小生成树

for (int count = 0; count < n - 1; count++) {

int u = minCost(cost, mstSet);

mstSet[u] = 1;

for (int v = 0; v < n; v++) {

if (graph[u][v] && !mstSet[v] && graph[u][v] < cost[v]) {

parent[v] = u;

cost[v] = graph[u][v];

}

}

}

// 打印最小生成树

printf("边\t权重\n");

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

printf("%d - %d\t%d\n", parent[i], i, graph[i][parent[i]]);

}

}

int main() {

FILE \*file = fopen("graph.txt", "r");

int start,end,weight;

int edge;

if (file == NULL) {

printf("failed to open file!\n");

return 1;

}

fscanf(file, "%d", &n);

fscanf(file, "%d", &edge);

int graph[n][n];

// 初始化邻接矩阵

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

for (int j = 0; j < n; j++) {

graph[i][j] = 0;

}

}

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

fscanf(file, "%d %d %d", &start, &end, &weight);

graph[start][end] = weight;

graph[end][start] = weight;

}

primMST(graph);

return 0;

}

**Kruskal算法：**

#include <stdio.h>

#include <stdlib.h>

int V;//顶点数

int E;//边数

// 定义边的结构体

struct Edge {

int src, dest, weight;

};

// 定义并查集的结构体

struct Subset {

int parent; //父节点

int rank; //高度

};

// 查找操作，用于查找节点所属的集合

int find(struct Subset subsets[], int i) {

if (subsets[i].parent != i) {

subsets[i].parent = find(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

// 合并操作，用于将两个集合合并为一个集合

void Union(struct Subset subsets[], int x, int y) {

int xroot = find(subsets, x);

int yroot = find(subsets, y);

if (subsets[xroot].rank < subsets[yroot].rank) {

subsets[xroot].parent = yroot;

} else if (subsets[xroot].rank > subsets[yroot].rank) {

subsets[yroot].parent = xroot;

} else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

// 按照边的权重进行升序排序

int cmp(const void\* a, const void\* b) {

struct Edge\* edge1 = (struct Edge\*)a;

struct Edge\* edge2 = (struct Edge\*)b;

return edge1->weight - edge2->weight;

}

// Kruskal算法，用于构建加权连通图的最小生成树

void Kruskal(struct Edge edges[]) {

//初始化

struct Subset\* subsets = (struct Subset\*)malloc(V \* sizeof(struct Subset));

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

subsets[i].parent = i;

subsets[i].rank = 0;

}

struct Edge result[V];

int e = 0, i = 0;

while (e < V - 1 && i < E) {

struct Edge next\_edge = edges[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

if (x != y) {

result[e++] = next\_edge;

Union(subsets, x, y);

}

}

printf("mst edge:\n");

for (i = 0; i < e; i++) {

printf("%d - %d weight: %d\n", result[i].src, result[i].dest, result[i].weight);

}

}

int main() {

FILE \*file = fopen("graph.txt", "r");

if (file == NULL) {

printf("failed to open file!\n");

return 1;

}

fscanf(file, "%d", &V);

fscanf(file, "%d", &E);

struct Edge edges[E];

for (int i = 0; i < E; i++)

{

fscanf(file, "%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);

}

qsort(edges, E, sizeof(edges[0]), cmp);

Kruskal(edges);

return 0;

}