数据结构

第四次上机作业实验文档

学号2017211123

班级 2017211301

序号 17

姓名 褚逸豪

## **问题描述**

输入无向图的邻接矩阵，使用前面讲过的任意三种方法求该图的最小代价生成树，并分析各自的时间复杂度。

## 算法思路

kruskal、prim，去边法

## 算法描述

1. Kruskal
   1. 初始每个点都属于编号为自身的独立集合
   2. 将边按照权值升序排序
   3. 从权值最小边开始枚举，如果边连接的两个点不在同一集合，则将两个点所在集合取并，并将该边加入MST
   4. 循环执行步骤c，直到有n-1条边被加入MST
2. Prim
   1. 初始dis[1]=0，其余点dis值为inf，集合V为空集
   2. 取出dis值最小且不在集合V中的元素，将其松弛，更新从他出发可以到达的点的dis值。将这个点加入V
   3. 循环执行步骤b，直到所有的点都被加入集合V
3. 去边法
   1. 将边按照权值降序进行排序
   2. 从权值最大的边开始枚举，如果去掉该边，该边的两条边不能通过其他可用边互相到达，则将该边加入MST，否则真的去掉该边。
   3. 循环执行步骤b，直到有n-1条边被加入MST

## 源程序及驱动程序

编译命令：g++ <文件名.cpp> -std=c++11

#include <iostream>

#include <cstdio>

#include <cstring>

#include <queue>

#include <utility>

#include <algorithm>

#include <vector>

using namespace std;

namespace mst {

const int MXN = 107;

int n;

int mp[MXN][MXN];

int dis[MXN];

struct cmp { bool operator()(const pair<int, int> &a, const pair<int, int> &b) { return a > b; } };

priority\_queue<pair<int, int>, vector<pair<int, int> >, cmp> pq;

struct edge { // 边类型，用于kruskal算法和去边法

int w, u, v;

bool operator<(const edge &b) const {

if (w == b.w) {

if (u == b.u) {

return v < b.v;

}

return u < b.u;

}

return w < b.w;

}

} buf[MXN \* MXN | 7];

int tot = 0;

int fa[MXN];

int getfa(int x) { // 并查集，用于判断连通性

return x == fa[x] ? x : fa[x] = getfa(fa[x]);

}

bool check(int x, int y) {

return getfa(x) == getfa(y);

}

void uni(int x, int y) {

fa[getfa(x)] = getfa(y);

}

int load(int n\_) { // 从标准输入读取邻接矩阵

n = n\_;

memset(mp, 0x3f, sizeof(mp));

memset(dis, 0x3f, sizeof(dis));

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

fa[i] = i;

tot = 0;

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

for (int j = 1; j <= n\_; ++j) {

scanf("%d", &mp[i][j]);

if (mp[i][j] != -1 && i > j) {

buf[tot++] = edge({mp[i][j], i, j});

} else if (mp[i][j] == -1 && i != j) {

mp[i][j] = 0x3f3f3f3f;

}

}

}

}

vector<edge> kru() { // kruskal算法

vector<edge> rtn;

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

fa[i] = i;

sort(buf, buf + tot);

for (int i = 0, z = 1; i < tot, z < n; ++i) {

if (!check(buf[i].u, buf[i].v)) {

rtn.push\_back(buf[i]);

uni(buf[i].u, buf[i].v);

++z;

}

}

return rtn;

}

vector<int> prim() { // 堆优prim算法

vector<int> rtn; // 最后一个存的是mst大小

while (!pq.empty()) pq.pop();

memset(dis, 0x3f, sizeof(dis));

dis[1] = 0;

pq.push(make\_pair(0, 1));

int ans = 0;

while (!pq.empty()) {

pair<int, int> tmp = pq.top();

pq.pop();

if (tmp.first <= dis[tmp.second]) {

rtn.push\_back(tmp.second);

ans += dis[tmp.second];

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

if (i == tmp.second) continue;

if (mp[tmp.second][i] < dis[i]) {

dis[i] = mp[tmp.second][i];

pq.push(make\_pair(dis[i], i));

}

}

}

}

rtn.push\_back(ans);

return rtn;

}

bool tag[MXN];

vector<edge> qubianfa() { // 去边法

vector<edge> rtn;

memset(tag, 0xff, sizeof(tag));

sort(buf, buf + tot);

int ans = 0;

for (int i = tot - 1; i >= 0; --i) {

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

fa[j] = j;

}

tag[i] = false;

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

if (tag[j])

uni(buf[j].u, buf[j].v);

}

if (!check(buf[i].u, buf[i].v)) {

tag[i] = true;

rtn.push\_back(buf[i]);

ans += buf[i].w;

}

}

return rtn;

}

};

using namespace mst;

int main() { // 读入数据

int n;

scanf("%d", &n);

mst::load(n);

auto x = mst::kru();

auto y = mst::prim();

auto z = mst::qubianfa();

int ans = 0;

puts("Kruskal");

for (auto i : x) {

ans += i.w;

printf("%d<-%d->%d\n", i.u, i.w, i.v);

}

printf("sum=%d\n", ans);

puts("Prim");

for (auto i = y.begin(); i != y.cend() - 1; ++i) {

printf("%d ", \*i);

}

printf("\nsum=%d\n", \*(y.cend() - 1));

puts("Qu Bian Fa");

ans = 0;

for (auto i : z) {

ans += i.w;

printf("%d<-%d->%d\n", i.u, i.w, i.v);

}

printf("sum=%d\n", ans);

}

## 测试数据

### Input

4

0 4 9 21

4 0 8 17

9 8 0 16

21 17 16 0

### Output

Kruskal

2<-4->1

3<-8->2

4<-16->3

sum=28

Prim

1 2 3 4

sum=28

Qu Bian Fa

4<-16->3

3<-8->2

2<-4->1

sum=28

### Online Test

<https://ideone.com/lQtVkT>

## 结果分析

三种方法求出来的MST均为预期的正确大小，Kruskal的复杂度为O，Prim复杂度是O，去边法的复杂度为O

## 结论

稠密图使用kruskal算法效率欠佳，但他在稀疏图上表现优秀。

## 心得体会

由于动态维护无向图连通性在我的知识范围之外，所以去边法使用了效率低下的暴力并查集。