目录

1 最小生成树

1.1 应用场景

在一个具有 n 个节点的加权连通图中，选择 n-1 条边使得图中任意两两节点联通，并且要保证选取的 n-1 个边的边权和最小。

1.2 问题定义

最小生成树: 在一个加权连通图中，最小生成树包含原图中的所有 n 个结点且权值和最小，并且有保持图连通的最少的边（说的就是不会成环）。

1.3 kruskal 算法

1.3.1 主要流程

1. 第一步定义边，并对边进行排序

   (a) 定义边

```
const int maxn = 10010;
struct edge{
   int st;//边的一个端点
   int ed;//边的另一个端点
   int w;//边的权重
};
edge E[maxn];
//读取边信息
for (int i= 0 ;i < n;i++){
    scanf("%d%d%d",&E[i].st,&E[i].ed,&E[i].w);
}
```

(b) 根据权值对边进行排序

```
#include <algorithm>
bool cmp(edge a, edge b){
    return a.w < b.w;
}
sort(E,E+n,cmp);
```

2. 第二步并查集模板 通过使用并查集来判断两个节点是否联通

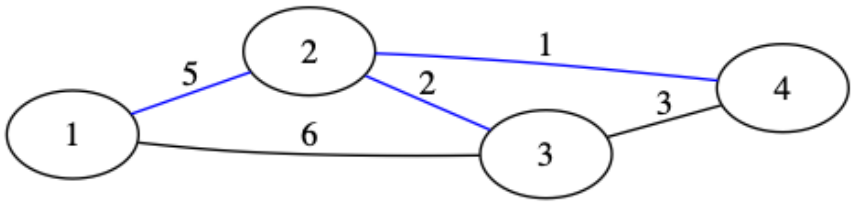
```
int fa[maxn];
void init(int n){
    for (int i = 0 ;i<= n ;i++)
        fa[i] = i;
}
int Find(int v){
    if (v == fa[v])
        return v;
    fa[v] = Find(fa[v]);
    return fa[v];
}
```

```
void update(int u, int v){
    int fu = Find(u);
    int fv = Find(v);
    fa[fu] = fv;
}
```

3. 第三步按照权重从小到大选边

```
int ans = 0;
init(n);
for (int i = 0 ;i< n;i ++){
   int u = E[i].st;
   int v = E[i].ed;
   int w = E[i].w;
   if (Find(u)!=Find(v)){
       update(u,v);
       ans += w;
   }
}
```

1.3.2 举例



1.4 prime 算法

1.4.1 主要流程

1. 设计并存储图

(a) 邻接矩阵

```
const int maxn = 1010;
int edge[maxn][maxn];
//edge[i][j]表示节点i与节点j之间的距离
//无向图中edge[i][j] == edge[j][i]
//有向图中不能保证edge[i][j] == edge[j][i]
```

(b) 邻接表

i. STL 之 Vector

```
//头文件
#include <vector>
//定义
vector <变量类型> 变量名;
vector <int> num;
//清空
num.clear();
//添加
num.push_back(5);
//使用
    //查看vector元素个数
    num.size();
    //查看第i个元素
    num[i];
```

ii. 通过 vector 实现邻接表

```
vector <int> node[maxn];
//node[i]表示节点i和哪些节点相连
//输入节点i与节点j相连
node[i].push_back(j);
node[j].push_back(i);
//查找k节点指向的所有节点
for (int i = 0 ;i< node[k].size() ;i++){
```

```
            int nextId = node[k][i];
        }
```

iii. 图示

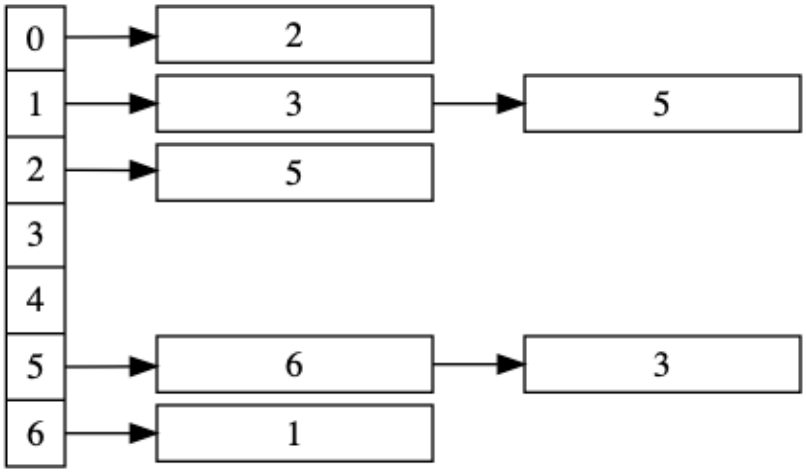


图 1: 邻接表图示

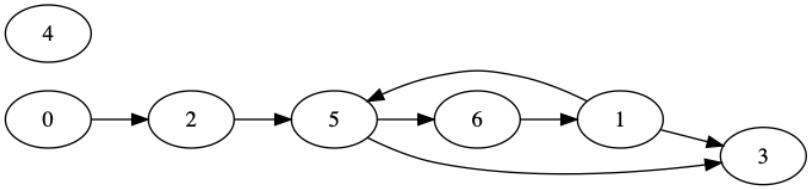


图 2: 邻接表对应的图结构

2. 符号定义

```
const int maxn = 1010;
//判断某个节点不是在树集合中
bool isTreeNode[maxn];
//用于存放某一个节点距离树的距离
int dist[maxn];
```

```
//存放边信息的结构体
struct edge{
    int nextId;
    int dist;
};
```

3. 根据当前节点更新 dist

```
void update(int addNode, int dist[]){
    for (int i = 0 ;i<node[addNode].size(); i++){
        int nextId = node[addNode][i].nextId;
        int dists = node[addNode][i].dist;
        dist[nextId] = dist[nextId]==-1?dists:min(dists,dist[nextId]);
    }
}
```

4. 初始化

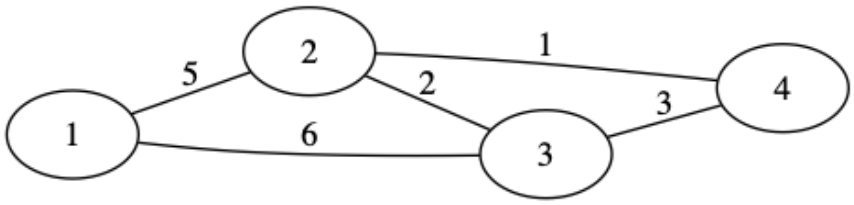
```
vector <edge> node[maxn];
void init(int sNode){
    //读取边信息
    for (int i = 0 ;i< m ;i++){
       int u,v,d;
       scanf("%d%d%d",&u,&v,&d);
       edge tmp;
       tmp.dist=d; tmp.nextId=v;
       node[u].push_back(tmp);
       tmp.nextId = u;
       node[v].push_back(tmp);
    }
    //Prime算法初始化
    memset(isTreeNode,0,sizeof isTreeNode);
```

```
    memset(dist,-1,sizeof dist);
    isTreeNode[sNode] = true;
    update(sNode,dist);
}
```

5. 添加节点

```
int prime(){
    int ans = 0;
    for (int i = 0 ;i< n-1 ;i++){
        int addNode = -1;
        for (int j = 0 ;j< n ;j++)
            if (isTreeNode[j]==false){
                if (addNode == -1 || dist[addNode]< dist[j])
                    addNode = j;
            }
        isTreeNode[addNode] = true;
        ans += dist[addNode];
        update(addNode,dist);
    }
    return ans;
}
```

1.4.2 举例



- step 1:
  - 添加节点 1，树节点集合变为 {1}

  - 使用新添加节点更新其它节点距离树的距离

1	2	3	4
0	5	6	INF

  - 添加边边权为 0

- step 2:
  - 添加节点 2，树节点集合变为 {1,2}
  - 使用新添加节点更新其它节点距离树的距离

1	2	3	4
0	0	2	1

  - 添加边边权为 5

- step 3
  - 添加节点 4，树节点集合变为 {1,2,4}
  - 使用新添加节点更新其它节点距离树的距离

1	2	3	4
0	0	2	0

  - 添加边边权为 1

- step 4
  - 添加节点 3，树节点集合变为 {1,2,3,4}
  - 使用新添加节点更新其它节点距离树的距离

1	2	3	4
0	0	0	0

  - 添加边边权为 2

1.5 相关题目

1.5.1 POJ 1251

1.5.2 POJ 2421

1.5.3 ZOJ 1586

1.5.4 POJ 2349