树

#基本概念

（1）树：树是一个有着有限个节点的集合

（2）度：即为每个节点之后对应的子树数量。

（3）树的度：即为一棵树的度的最大值

（4）叶子结点：没有子节点的节点是叶子节点，度为零

（5）空树：没有节点的树

（6）在树中，每个节点有且仅有一个前驱（除了根节点）

（7）左孩子：一个节点对应的左子节点

（8）右孩子：一个节点对应的右子节点

（9）节点的深度：从上往下数数，从一开始数

（10）节点的高度：从下往上数，从一开始数

(11) 满足树的必要条件：有且仅有一个根节点，且其余节点可分为m个互不相交的分支

#树的一些性质

（1）节点数=度的总数+1

（2）度为n 的树与n叉树：
前者意为至少有一个节点的度对应的是N且这个树的度就是n;
后者是这个树的最大度<=n,树甚至可以是空树。

（3）度为i的树的第m层最多只有i^(m-1)个节点

（4）用等比数列可求一个高度是h的n叉树最多有多少个节点

（5）一个高度是h的n叉树最少有H个节点；一个高度是h的度为n的树最少有h+n-1个节点

#二叉树

（1）二叉树的特点：每个节点最多只有两个子树，而且计较顺序

（2）二叉查找树：对任何一个节点，左孩子的值<根节点值<右孩子的值

（3）平衡二叉树：对于树上任何一个节点的左子树与右子树高度之差不超过1
平衡因子为左子树高度减去右子树高度

（4）二叉树的性质（*重点*）

- 对于二叉树第i层最多有2^(i-1)个节点，最少有一个节点
- 深度为k的树最多有2^(k-1)个节点（可以用等比数列推出来），最少有k个节点
- 对于一棵二叉树，若度数为0的节点数是n0,如果度数为2的节点数是n2,那么满足n0=n2+1；（总边数都等于节点数-1）
- 有k层且节点个数等于2^k-1的树是满二叉树，满二叉树在同深度二叉树中节点数，叶子节点数最多
- 深度为k且有N个节点的树，且每个节点的编号对应于完全二叉树中对应节点的编号，则这种树是完全二叉树。从满二叉树中从最后一个节点开始，一一去掉连续的节点，得到的就是完全二叉树。对于完全二叉树，如果右节点的层数为i，那么左节点的层数为i或i+1.所有叶子节点都在倒数第一或第二层。

（5）二叉树的存储结构：

- 顺序存储：依次按照完全二叉树的顺序的编号存放节点，如果是没有对应的节点要存入空，而不是什么也不做。缺点是大量的空点位置存入了空点，空间浪费。适合存储完全二叉树和满二叉树
- 链式存储：也不赘述，值得注意的是有n个节点的链式二叉树，有n+1个空指针域

（6）二叉树的遍历：总而言之有三种方法，先序后序中序。

- 递归遍历：最简单的遍历方式
- 栈式遍历：依次入栈
- 层次遍历：bfs，遇见根节点则将其入队，如果有左右孩子再将左右孩子入队，并输出该节点的值。循环一直持续到队列中的元素数量为零为止。
- 以上所有遍历方式的时间空间复杂度都是o(n)

（7）根据遍历序列判断树：如果树中每个节点的值都是不一样的，则该树的先序后序中序遍历结果是不一样的。
只要知道中序再任意来一个，就可以确定树的内容。
第07周12--5.5 遍历二叉树和线索二叉树2--由遍历序列确定二叉树_哔哩哔哩_bilibili

三种递归式遍历算法的空间复杂度是：o（n）；时间复杂度是o(n)；
其他还可以通过栈实现非递归的遍历，

（8）线索二叉树：
依照某种要求的遍历序列，如果一个树的节点的左子树指针为空，则让他指向前驱；如果右子树指针为空，则让他指向后继。注意这里的前驱后继是在这个序列中的顺序来说的。

> 注意树和二叉树是两个截然不同的东西！树不用区分左右子树，而二叉树需要！

#二叉查找树

#哈夫曼树

（1）定义：构造出来的WPL(带权路径最小的树)，因此也叫最优二叉树。

（2）权：将某个特定的值赋给树中的节点，这个值叫做节点的权。

> 因此我们可以知道，哈夫曼树与最小生成树和最短路径问题的区别在于哈夫曼树依照点的权值做出判断，而不是路的权值。

节点的带权路径长度为根节点到该节点的路径乘以该节点的权值。
树的带权路径长度是根节点到所有叶子节点的带权路径长度之和。

（3）哈夫曼树的构造方法：将每个节点作为一个单独的树之后：

- 选出其中根节点权值最小的两棵树
- 将两棵树取出来，并构造一个节点等于其根节点权值之和的节点作为其新根，两棵树分别作为左右子树
- 在原节点集合中删除这两棵树，并将新树加入其中
- 重复操作直至只剩一棵树，这棵树就是哈夫曼树

（4）哈夫曼树构造算法：若有N个节点，则需要2* N-1个节点的空间。
具体模板在这：
P1090 [NOIP2004 提高组] 合并果子 / [USACO06NOV] Fence Repair G - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

#生成树
主要关注生成树算法。
（1）prim 算法：复杂度为$O(n^2)$
原题：858. Prim算法求最小生成树 - AcWing题库

```
# include<bits/stdc++.h>
# include<algorithm>
typedef long long ll;
using namespace std;
ll n,m;
bool istrue[2030];
int dis[2030];
int p[2030][2030];
int prim()
{
        int res=0;
        memset(dis,0x3f,sizeof dis);
        for(int i=0;i<n;i++)
        {
                int t=-1;
                for(int j=1;j<=n;j++)
                {
                        if(!istrue[j]&&(t==-1||dis[j]<dis[t]))
```

```
                        j=t;
                }
                if(i&&dis[t]==0x3f3f3f3f)
                {
                        return 114514;
                }
                if(i)
                res+=dis[t];
                for(int j=1;j<=n;j++)
                {
                        dis[j]=min(dis[j],p[t][j]);
                }
                istrue[t]=true;

                }
                return res;
}
int main (void)
{
        cin>>n>>m;
        for(int i=0;i<m;i++)
        {
                int a,b,c;
                cin>>a>>b>>c;
                p[a][b]=p[b][a]=min(p[a][b],c);

        }
        int t=prim();
        if(t=114514)
        {
                cout<<"impossible"<<endl;
                return 0;
        }
        cout<<t<<endl;
        return 0;

 }
```

（2）k算法

比较建议使用k算法

输入和上一题完全一样

```cpp
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010, M = 200010, INF = 0x3f3f3f3f;

int n, m;
int p[N];

struct Edge
{
    int a, b, w;

    bool operator< (const Edge &W)const
    {
        return w < W.w;
    }
}edges[M];

int find(int x)
{
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int kruskal()
{
    sort(edges, edges + m);

    for (int i = 1; i <= n; i ++ ) p[i] = i;    // 初始化并查集

    int res = 0, cnt = 0;
    for (int i = 0; i < m; i ++ )
    {
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        a = find(a), b = find(b);
        if (a != b)
        {
            p[a] = b;
            res += w;
```

```
            cnt ++ ;
        }
    }

    if (cnt < n - 1) return INF;
    return res;
}

int main()
{
    scanf("%d%d", &n, &m);

    for (int i = 0; i < m; i ++ )
    {
        int a, b, w;
        scanf("%d%d%d", &a, &b, &w);
        edges[i] = {a, b, w};
    }

    int t = kruskal();

    if (t == INF) puts("impossible");
    else printf("%d\n", t);

    return 0;
}
```