数图的结构，存储，以及遍历

主讲人：谭星语

树的结构

~~树(Tree)是n(n≥0)个结点的有限集。n=0时称为空树。在任意一棵非空树中：(1)有且仅有一个特定的称为根(Root)的结点；(2)当n>1时，其余结点可分为m(m>0)个互不相交的有限集T1、T2、……、Tm，其中每一个集合本身又是一棵树，并且称为根的子树(SubTree)。~~

树是由n（n>=1）个有限结点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。

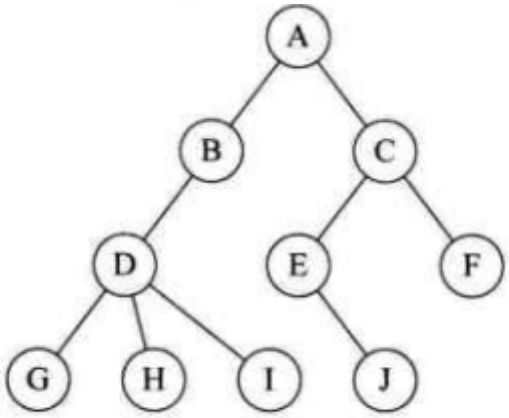


图 6-2-1

树的相关术语：

结点的度: 一个结点含有的子树的个数称为该结点的度；

叶结点:度为0的结点称为叶结点，也可以叫做终端结点；

分支结点：度不为0的结点称为分支结点，也可以叫做非终端结点；

结点的层次：从根结点开始，根结点的层次为1，根的直接后继层次为2，以此类推；

结点的层序编号：将树中的结点，按照从上层到下层，同层从左到右的次序排成一个线性序列，把他们编成连续的自然数；

树的度：树中所有结点的度的最大值；

树的高度(深度)：树中结点的最大层次；

孩子结点：一个结点的直接后继结点称为该结点的孩子结点；

双亲结点(父结点)：一个结点的直接前驱称为该结点的双亲结点；

兄弟结点：同一双亲结点的孩子结点间互称兄弟结点。

二叉树

- 二叉树：每个结点最多含有左，右两个子树的树称为二叉树。
- 满二叉树：除最后一层无任何子节点外，每一层上的所有结点都有两个子结点的二叉树。
- 完全二叉树：树中所含的 n 个结点和满二叉树中编号为 1 至 n 的结点一一对应的二叉树。
- 满二叉树一定是完全二叉树。

树的直径

- 第一次BFS（DFS）从任意顶点出发，设该顶点为t，找到距t最远的顶点为u，则u一定为树的直径中的某一个顶点。
- 第二次BFS(DFS)从u出发，找到距离u最远的顶点v， 则u到v的最短路径就是树的直径

结构

- 孩子数组 动态开点（省空间）/固定点（常用于二叉树，简单）
- father数组记录父亲（见后面并查集的具体存储方法）
- 图存储

动态开点

- 一般对空间有较高要求，如可持续化的线段树，动态开点的线段树，trie树，ac自动机 等，以后学一些高级的数据结构会用到，原理类似于指针，只不过是用标号寻找结点。

```
struct Node{
  int child[MAX_CHILD];//每个节点最多的孩子数
  int val;
}node[maxn];
int ct = 0;
void insert(int &o,int v)
{
  if (!o)
  {
    o = ++ct;
    node[o].val = v;
    for(int i = 0;i<MAX_CHILD;i++)node[o].child[i]=0;
    return;
  }
  ......
}
```

固定点

- 适用于二叉树。每个点只有左孩子，右孩子。从1开始。基础线段树都是这样存点的。

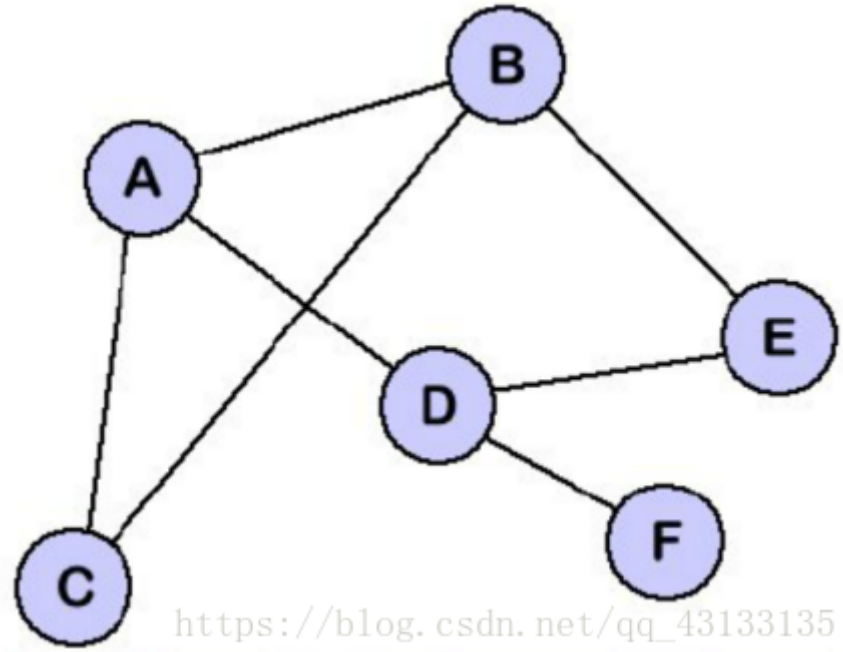
```
#define Lson(x) (x<<1)
#define Rson(x) ((x<<1)|1)
#define Father(x) (x>>1)
int val[maxn];
int main(){
    val[x] = 5; //x结点值为5
    val[Lson(x)] = 6; //x左孩子结点值为6
    val[Rson(x)] = 7; //x右孩子结点值为7
    int lson = Lson(x);
    cout <<Father(lson)<<endl; //输出lson的父节点，即x的值
}
```

图存储（见后）

适用于绝大多数的题，也是最常见的树的存储方法，需要手动给一
个根。
• 邻接矩阵
• 邻接表
• 链式向前星

图

一张图G GG由两个集合组成的二元组，点集V 和边集E 。即G = (V , E) G=(V,E)G=(V,E)。



术语

- 边 就是连接点和点之间的关系，分成有向边和无向边两种。
- 无向图 就是都是无向边的图。

- 有向图 就是有向边组成的图。
- 具体题目中，边和点可能由权值，也就是点权和边权。
- 路径 是一个点到另一个点经过的边的序列。
- 简单路径 指没有经过重复边的路径。
- 环，也叫回路，指的是一个点经过一条简单路径回到自身。

矩阵存图

```
int tu[maxn][maxn];//图

memset(tu,0,sizeof(tu));//初始化图

tu[a][b]=w;//添加一条a到b的边权为w的有向边

tu[a][b]=tu[a][b]=w;//添加一条a到b的边权为w的无向边

//遍历点x连的边
for(int i=1;i<=n;i++)if(i!=x&&tu[x][i]!=0)
{
    printf("点x到点%d,边权为%d的边\n",i,tu[x][i]);
}
```

---

vector存图（邻接表存图）

```
vector<int>v[maxn],g[maxn];//图

for(int i=0;i<maxn;i++)v[i].clear(),g[i].clear();//初始化图

v[a].push_back(b);//添加一条a到b的边权为w的有向边
g[a].push_back(w);

//添加一条a到b的边权为w的无向边
v[a].push_back(b);
g[a].push_back(w);
v[b].push_back(a);
g[b].push_back(w);

//遍历点x的连的出边
for(int i=0;i<v[x].size();i++)
{
    int to=v[x][i],w=g[x][i];
    printf("点x到点%d,边权为%d的边\n",to,w);
}
```

链式前向星

```
struct
{
    int to,w,next;
```

```
}edge[maxm];
int head[maxn],tot;//图
void init()//初始化图
{
    tot=0;
    memset(head,-1,sizeof(head));
}
void addedge(int a,int b,int w)//添加一条a到b边权为w的有向边,无向边就正反各加一次
{
    edge[tot].to=b;
    edge[tot].w=w;
    edge[tot].next=head[a];
    head[a]=tot++;
}

//遍历点x的连的出边
for(int i=head[x];i!=-1;i=edge[i].next)
{
    int to=edge[i].to,w=edge[i].w;
    printf("点x到点%d,边权为%d的边\n",to,w);
}
```