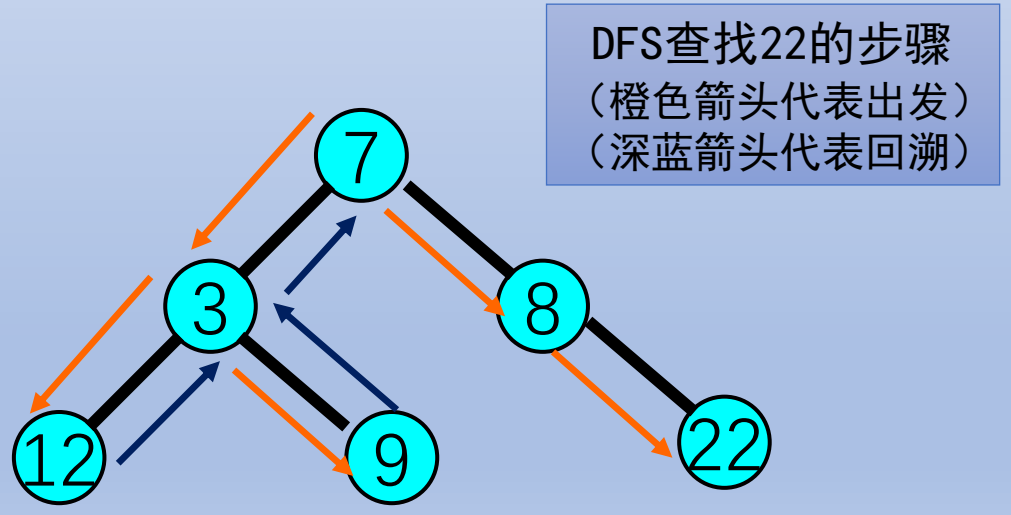
- 聪明
- 乐观
- 勤奋
- 专注
- 妙趣横生
- 坚持就是胜利

Welcome to OI!

知识回顾：

- DFS：深度优先搜索

从问题的某一种可能出发，搜索从这种情况出发所能达到的所有可能，当这一条路走到“尽头”而没找到解时，再倒回上一个步骤，从另一个可能出发，继续搜索.




```
void DFS( Point P ){
    for(所有P的邻接点K){
        if(K未被访问){
            标记K;
            DFS(K);
        }
    }
}
```

DFS序

欧拉序

括号序

design by Lstg

Thanks for Boss He

从DFS到DFS序：

- DFS序，顾名思义，DFS的顺序，记录了每个节点被搜到的时间

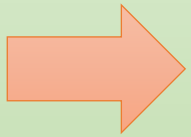
每DFS到一个新的节点，tot都会+1

dfn值更小的节点一定先被访问到

dfn数组实际记录了DFS的顺序

```
void DFS(int x){
    for(int i = 1;i<=n;i++)
        if(g[x][i]) // 如果从x出发能到达i
            if(!mark[i]) { // 如果i没被访问过
                mark[i]=true; // 标记i已访问
                DFS(i); // 递归搜索i
            }
}
```

DFS的一般代码
（某些情况需要还原现场，此处不考虑。）



```
void DFS(int x){
    dfn[x]=++tot; // dfs到当前节点，次序+1
    for(int i = 1;i<=n;i++)
        if(g[x][i]) // 如果从x出发能到达i
            if(!dfn[i]) { // dfn[i]==0表示未被访问过
                DFS(i); // 递归搜索i
            }
}
```

计算DFS序的DFS代码

例题1：

给出一棵根节点为1的树，输入m个询问 (x, y)，如果x是y的祖先，输出YES，否则输出NO。

输入：

第一行输入两个整数n，m，表示节点数和询问数量。（1<=n,m<=200000）

第二行到第n行，每行输入一对整数(a, b)，表示a与b之间有边相连。

第n+1行到第n+m行，每行输入一对整数（x, y）。

输出：

对于每个询问，如果x是y的祖先，输出YES，否则输出NO。

例题1 - 分析：

怎么判断一个点是不是另一个点的祖先？ 问父亲！

```
bool check_ancestor (int x,int y) { // 判断x是不是y的祖先
    if(x==y)return true; // 根据定义，自身可以是自身的祖先节点
    int dad=father[y];
    while(dad>0){
        if(dad==x)return true;
        dad = father[dad];
    }
    return false;
}
```

1<=n,m<=200000

时间复杂度？

O(n)，最坏情况下，会把所有点都问一遍。

总时间复杂度：O(mn)

例题1 - 分析：

怎么判断一个点是不是另一个点的祖先？ **问父亲 - 倍增加速版**

```
bool check_ancestor (int x,int y) { // 判断x是不是y的祖先
    if(x==y || fa[y][0]==x) return true;
    if(dep[x]>dep[y]) return false;
    int u=x, step=dep[y]-dep[x];
    for(i=k; i>=0; i--) if(step&(1<<i)) y=fa[y][i];
    if(x==y)return true;
    else return false;
}
```

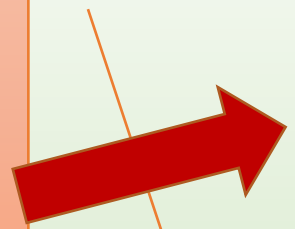
时间复杂度？

O(logN)，总时间复杂度：O(mlogN)

1<=m<=10000000 时，怎么办？

例题1 - 分析：

先赋值，后递归。
father[i]在DFS(i)之前赋值完成
father[i]在i之前就已经被访问过了！



DFS序！

如何优化？

① 优化m？

不可能，因为一定会遍历每一个询问

② 优化n？

提高判断祖先的速度！！！

思考：在DFS建树过程中，father数组是怎么赋值的

```
void DFS(int x) {
    for(int i=1;i<=n;i++)
        if(g[x][i] && father[x]!=i) {
            father[i]=x;
            DFS(i);
        }
}
```

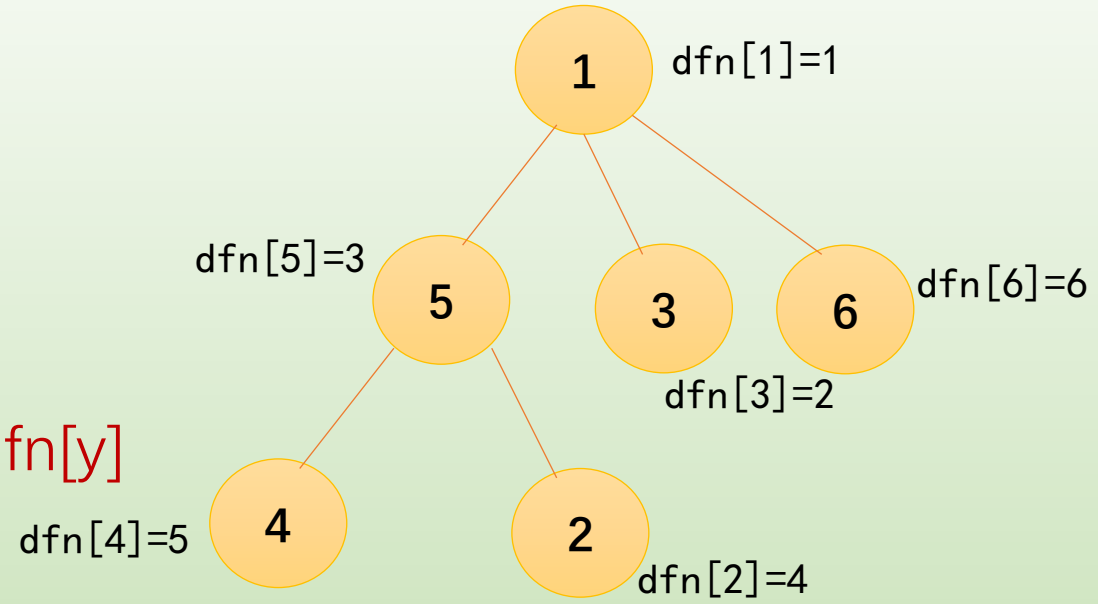
例题1 - 分析：

怎么判断一个点是不是另一个点的祖先？

如果x是y的祖先，那么dfn[x]一定小于等于dfn[y]

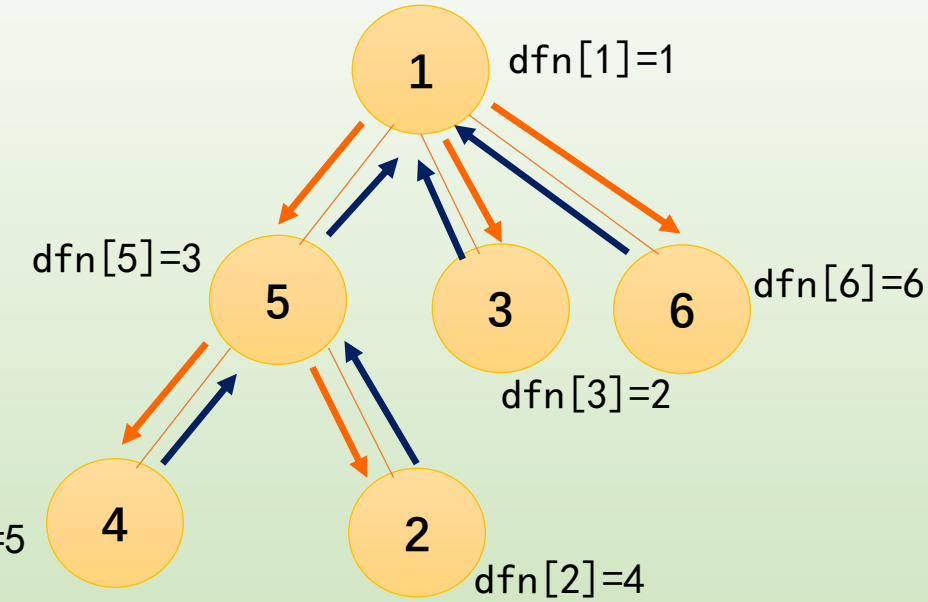
如果dfn[x]小于dfn[y]，那么x一定是y的祖先吗？

思考：在DFS建树过程中，完整的流程是怎么样的？



例题1 - 分析：

怎么判断一个点是不是另一个点的祖先？



如果x是y的祖先，那么dfn[x]一定小于等于dfn[y]

如果dfn[x]小于dfn[y]，那么x一定是y的祖先吗？

DFS：从某一点出发，递归搜索能够到达的点，直到走到“尽头”后，回溯到上一个节点，继续搜索。

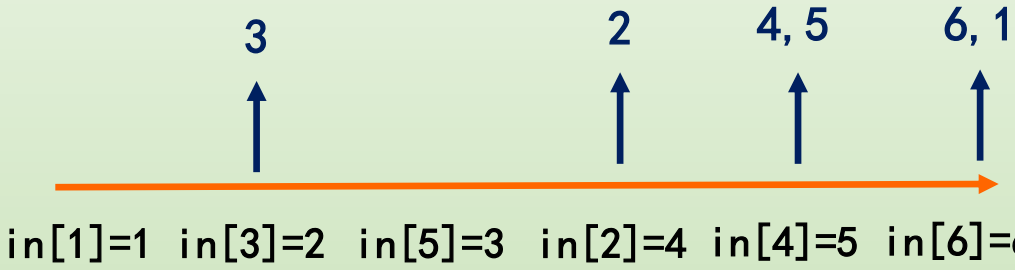
思考：在DFS建树过程中，完整的流程是怎么样的？

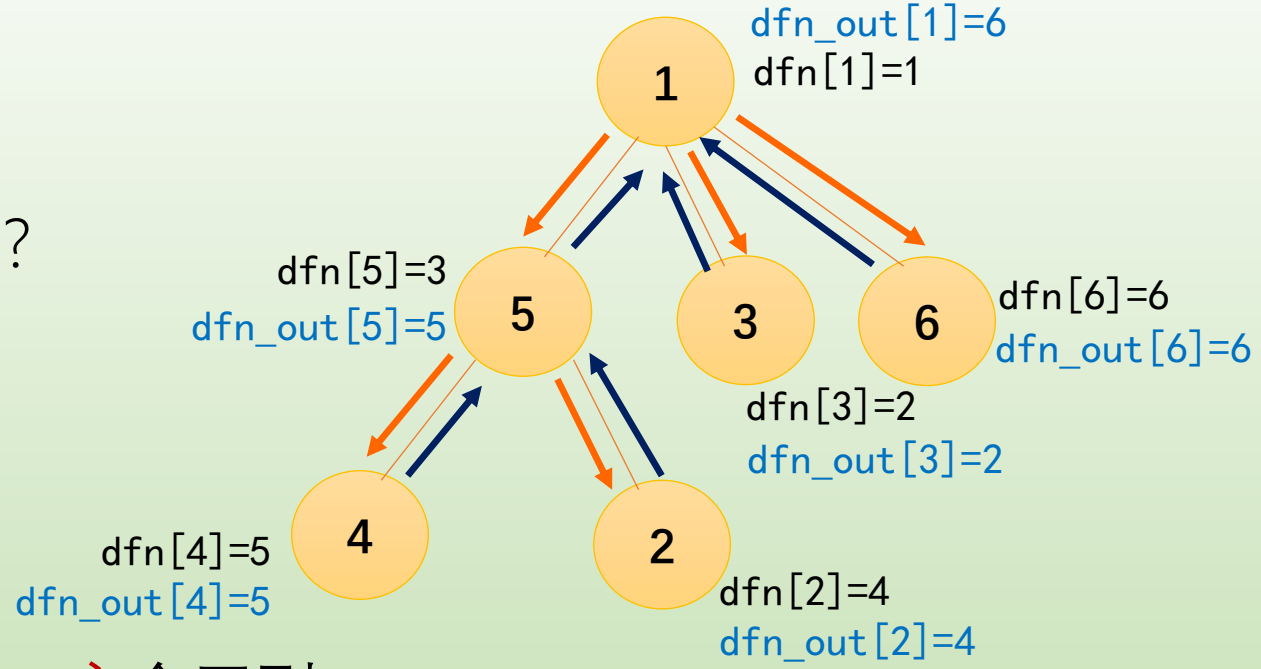
一个点遍历了它所有的子孙节点之后，一定会回到它本身。

此时，它所有的子孙节点都已经有了DFS序。

例题1 - 分析：

怎么判断一个点是不是另一个点的祖先？





一个点遍历了它所有的子孙节点之后，一定会回到它本身。**此时，它所有的子孙节点都已经有了DFS序。**

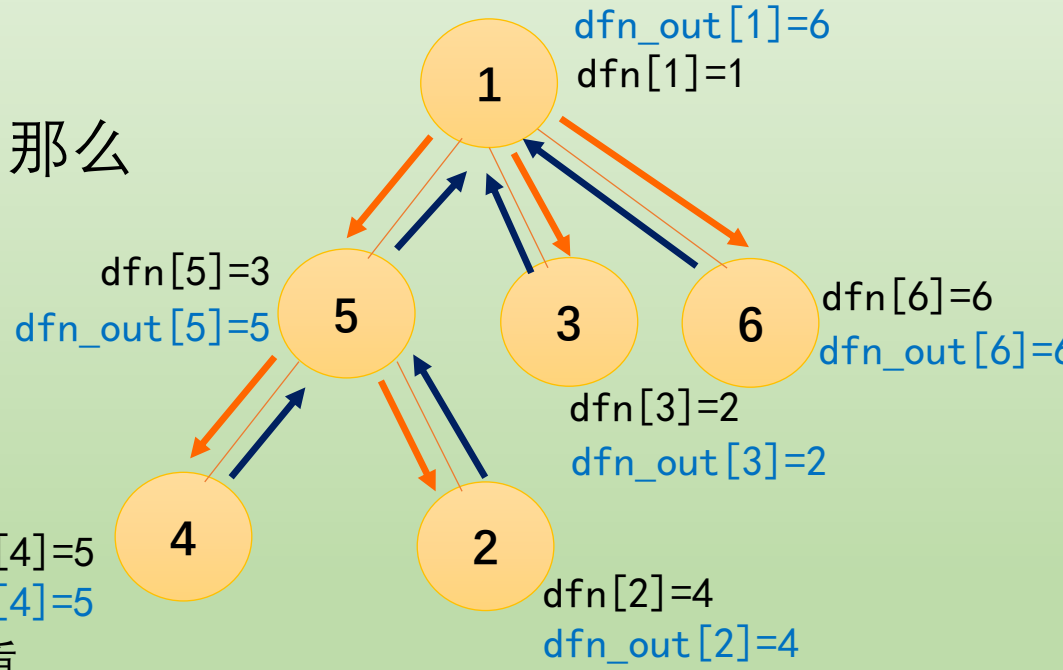
DFS：从某一点出发，递归搜索能够到达的点，直到走到“尽头”后，回溯到上一个节点，继续搜索。

一个节点的子孙节点的DFS序，大于等于它自己的DFS序，小于等于遍历完所有子孙节点后，回溯到自身时，DFS序的最大值。

例题1 - 分析：

一个节点的子孙节点的DFS序，大于等于它自己的DFS序，小于等于遍历完所有子孙节点后，回溯到自身时，DFS序的最大值。

如果有dfn[x]<=dfn[y] && dfn_out[x]>=dfn[y]，那么x一定是y的祖先吗？



证明：如果存在一对点<x,y>，满足dfn[x]<=dfn[y] && dfn_out[x]>=dfn[y]，且y不在x的子树里，则：

① 如果DFS(x)时已经搜索过y，那么有dfn[y]<dfn[x]，矛盾。

② 如果DFS(x)时仍未搜索过y，那么有dfn[y]>dfn_out[x]，矛盾。

实际上，如果y在x的子树里，一定有dfn[x]<=dfn[y]<=dfn_out[y]<=dfn_out[x]，逆命题也成立。

例题1 - 分析：

怎么判断一个点是不是另一个点的祖先？

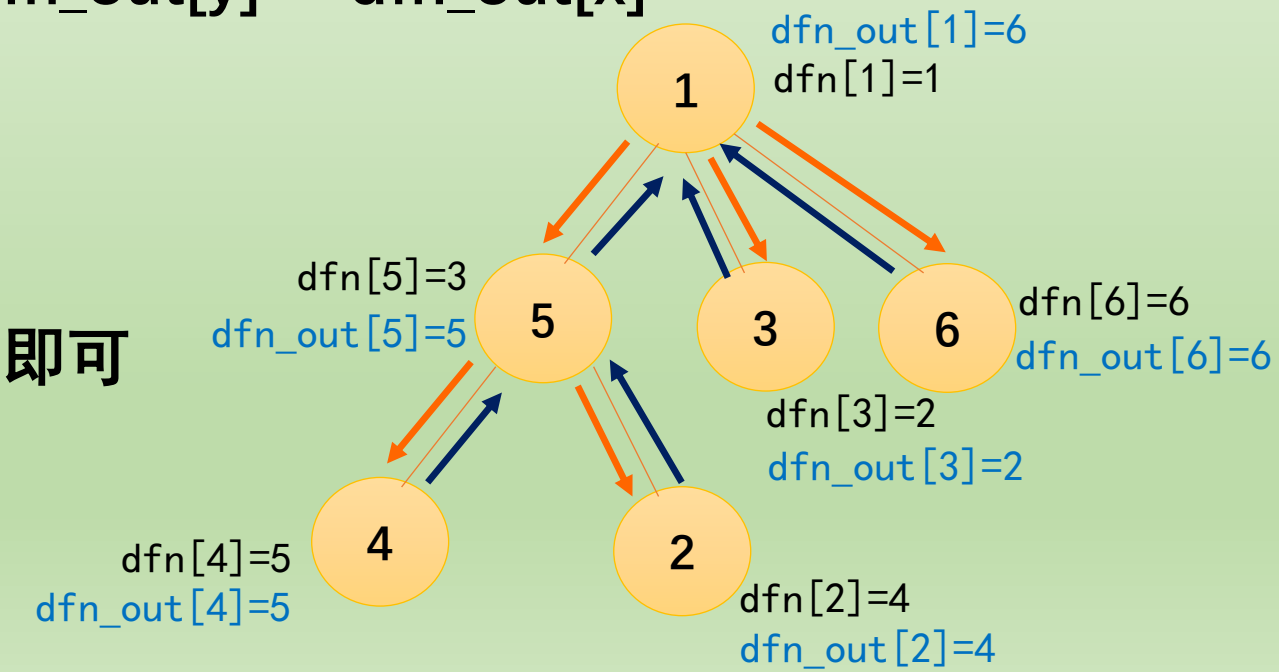
① 求DFS序

② 判断是否有 **dfn[x]<=dfn[y]<=dfn_out[y]<=dfn_out[x]**

时间复杂度？

O(1)，dfs建树时顺便预处理dfs序即可

总时间复杂度：O(n+m)



例题1 - 核心代码：

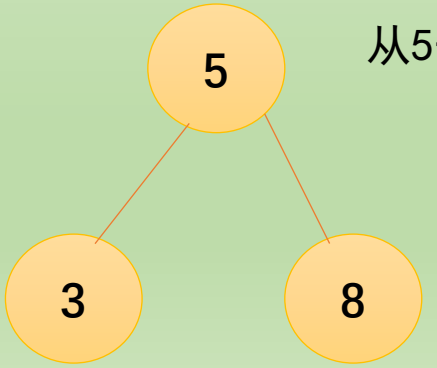
```
int dfn[MAXN]; // 存每个点被dfs搜到的次序
int dfn_out[MAXN]; // 存每个点回溯退出时最大的dfs序
int tot=0; // 用来记录当前的dfs序

void DFS(int x){
    dfn[x]=++tot; // dfs到当前节点，次序+1
    for(int i=1;i<=n;i++)
        if(g[x][i]&&!dfn[i]) //dfn为0则没被访问过
            DFS(i);
    dfn_out[x]=tot; // 已经搜索完所有子树，记录
                    当前最大的dfs序
}
```

使用动态数组或链式向前星存图可使DFS时间复杂度降到O(n), 此处的邻接链表仅为了便于理解

熟练掌握递归思想，理解两个tot值。

从DFS到DFS序：

- DFS序，顾名思义，DFS的顺序，记录了每个节点被搜到的时间
- **一个节点的子孙节点的DFS序，大于等于它自己的DFS序，小于等于遍历完所有子孙节点后，回溯到自身时，DFS序的最大值。**
- DFS序不是唯一的，取决于DFS时具体的搜索顺序



从5开始搜索，求DFS序

[5, 3, 8]？

[5, 8, 3]？

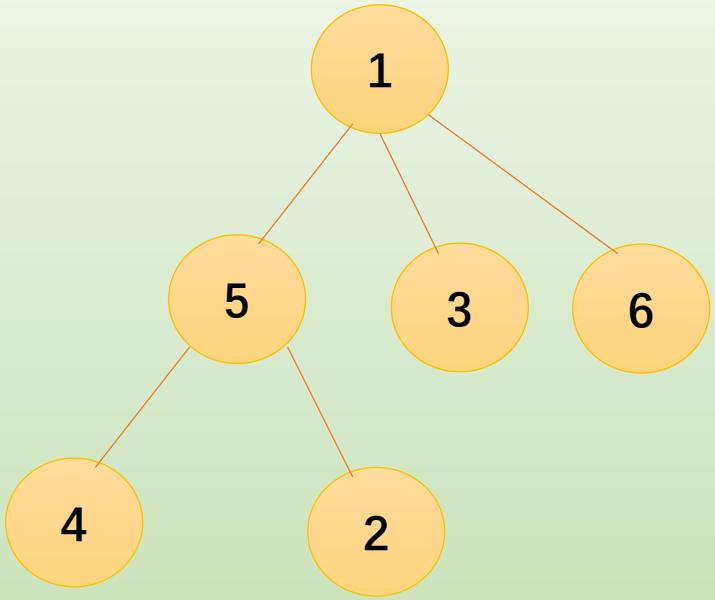
都可以！！

```
void DFS(int x){
    dfn[x]=++tot; // dfs到当前节点，次序+1
    for(int i=1;i<=n;i++)
        if(g[x][i]&&!dfn[i]) //dfn为0则没被访问过
            DFS(i);
    dfn_out[x]=tot; // 已经搜索完所有子树，记录当前最大的dfs序
}
```

计算DFS序的DFS代码（NEW）

热炒热卖：

给出一棵树，有m次询问。对于每次询问x，回答以x为根节点的子树的节点数。



分析：size[x]=dfn_out[x]-dfn[x]+1;

例题3：

给出一棵根节点为1的树，输入m个询问 (x, y) ，对于每个询问，求 (x,y) 的最近公共祖先（LCA）

输入：

第一行输入两个整数n，m，表示节点数和询问数量。（1<=n,m<=2000）

第二行到第n行，每行输入一对整数(a, b)，表示a与b之间有边相连。

第n+1行到第n+m行，每行输入一对整数（x, y）。

输出：

对于每个询问，输出一个整数，表示x和y的最近公共祖先。

注：最近公共祖先 (LCA) 表示满足同时是x和y的祖先且高度最大的一个。

（设root的高度为1）

例题3 - 分析：

怎么判断点u是不是点x的祖先？

```
dfn[u]<=dfn[x]&&dfn_out[u]>=dfn_out[x];
```

怎么判断点u是不是点y的祖先？

```
dfn[u]<=dfn[y]&&dfn_out[u]>=dfn_out[y];
```

怎么判断点u是不是点x和y的公共祖先？

```
dfn[u]<=dfn[x]&&dfn_out[u]>=dfn_out[x]&&dfn[u]<=dfn[y]&&dfn_out[u]>=dfn_out[y];
```

解法：对于每个询问，DFS遍历所有点。

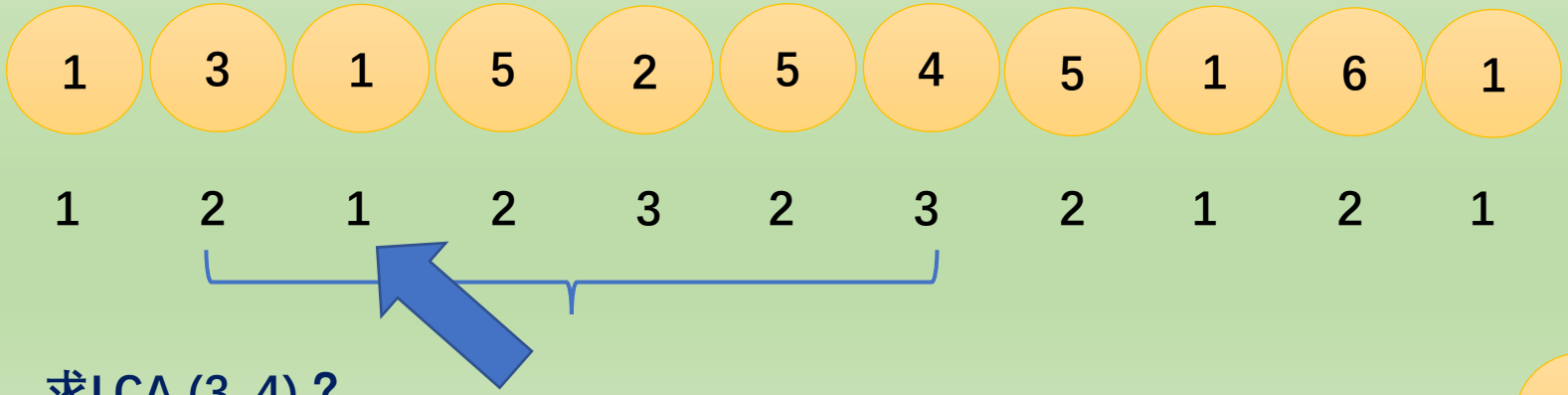
解法（倍增加速版）：只有x一个点往上跳，跳步之前，判断要跳的点是不是y的祖先，不是才跳。

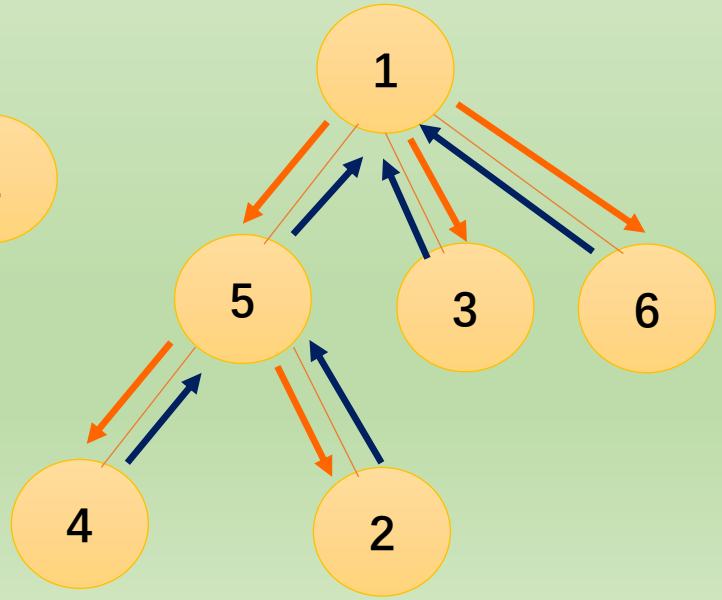
例题3 - 分析：

解法（欧拉序版）：

什么是**欧拉序**？

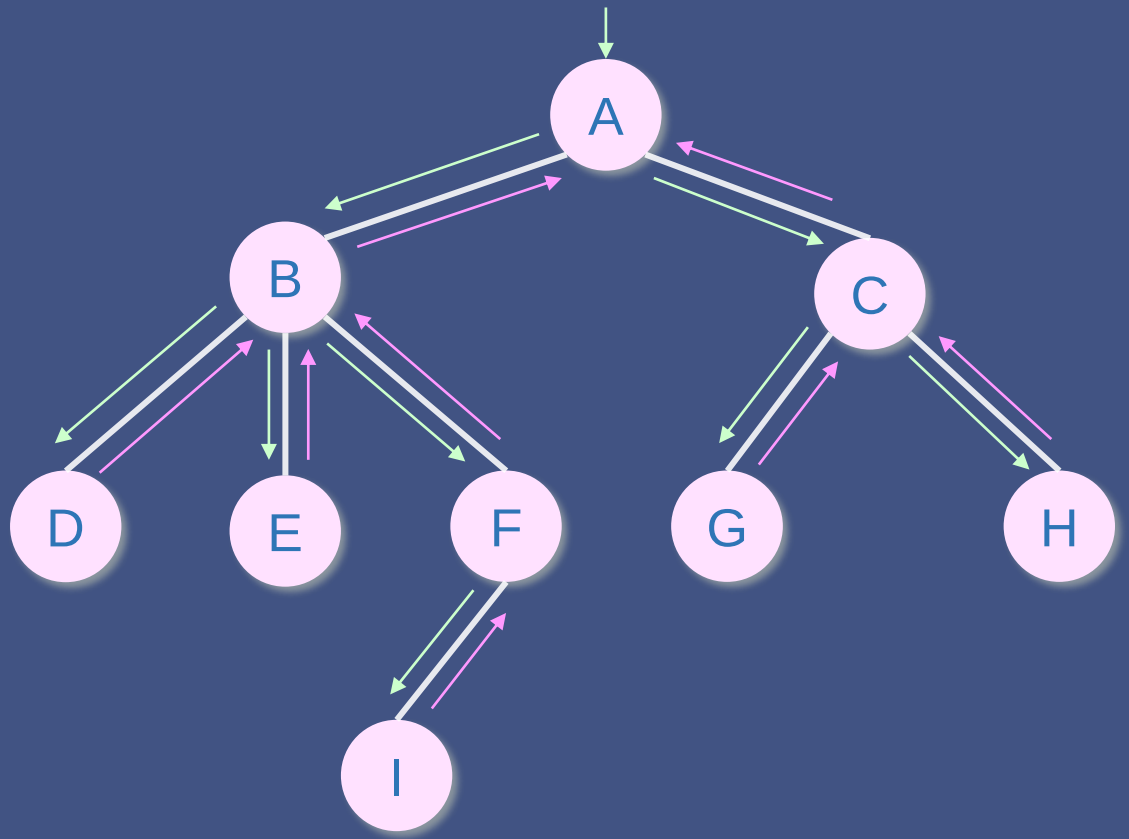
从根结点出发，按dfs的顺序再绕回原点所经过所有点的顺序，这个序列就叫欧拉序。





欧拉序的作用：求LCA

第一种欧拉序：对于点i,每当dfs到i时加进序列，每当dfs回溯到i时加进序列。



求x,y两点的LCA

在欧拉序中，x和y第一次出现的位置之间，一定有它俩的LCA，也就是其中深度最小的一个。

比如，求左图D和C的LCA，在下面欧拉序中，标注出了第一次出现的位置(3和12)，在它们之间，深度最小的是A(深度为1)，那么A就是D和C的LCA。

以深度为关键字，查询区间[3,12]的最小值即可。ST表或者线段树都行。

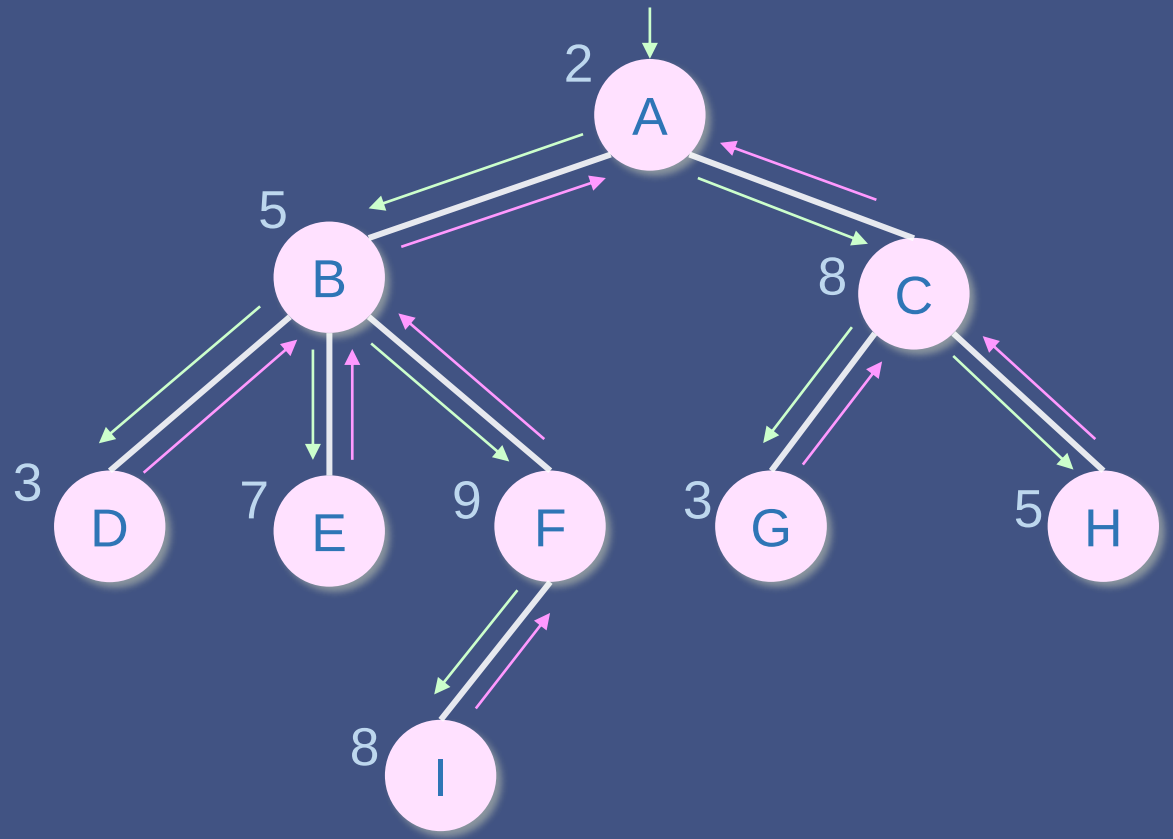
编号：	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
欧拉序：	A	B	D	B	E	B	F	I	F	B	A	C	G	C	H	C	A
深度：	1	2	3	2	3	2	3	4	3	2	1	2	3	2	3	2	1

欧拉序的作用：树上求和1

求某个子树的权值和

方法是：**只记录第一次出现的数的值**，同样的查询某点就只需要查询该点在欧拉序中最后出现的位置的前缀即可减去第一次出现的额位置-1的前缀和即可。

第一种欧拉序：对于点i,每当dfs到i时加进序列，每当dfs回溯到i时加进序列。



以求点B为根的子树中的点权总和为例：

点权数组中，只在每个点第一次出现位置记录了点权，其它地方点权为0。

欧拉序中，B第一次出现的位置为2，最后出现的位置为10。

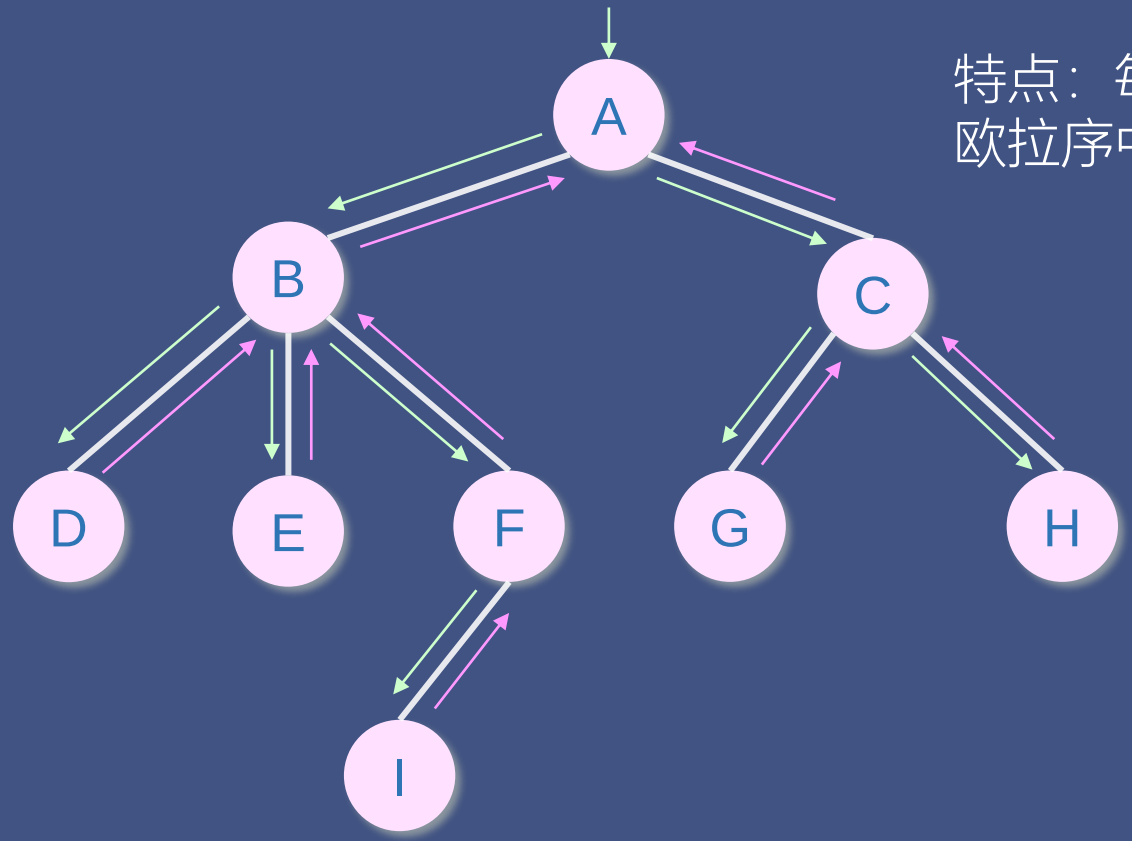
对点权数组求前缀和Sum。

B为根的子树点权总和=Sum[10]-Sum[1]=32

点权数组：	2	5	3	0	7	0	9	8	0	0	0	8	3	0	5	0	0
欧拉序：	A	B	D	B	E	B	F	I	F	B	A	C	G	C	H	C	A
编号：	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

欧拉序的概念1

第一种欧拉序：对于点i,每当dfs到i时加进序列，每当dfs回溯到i时加进序列。



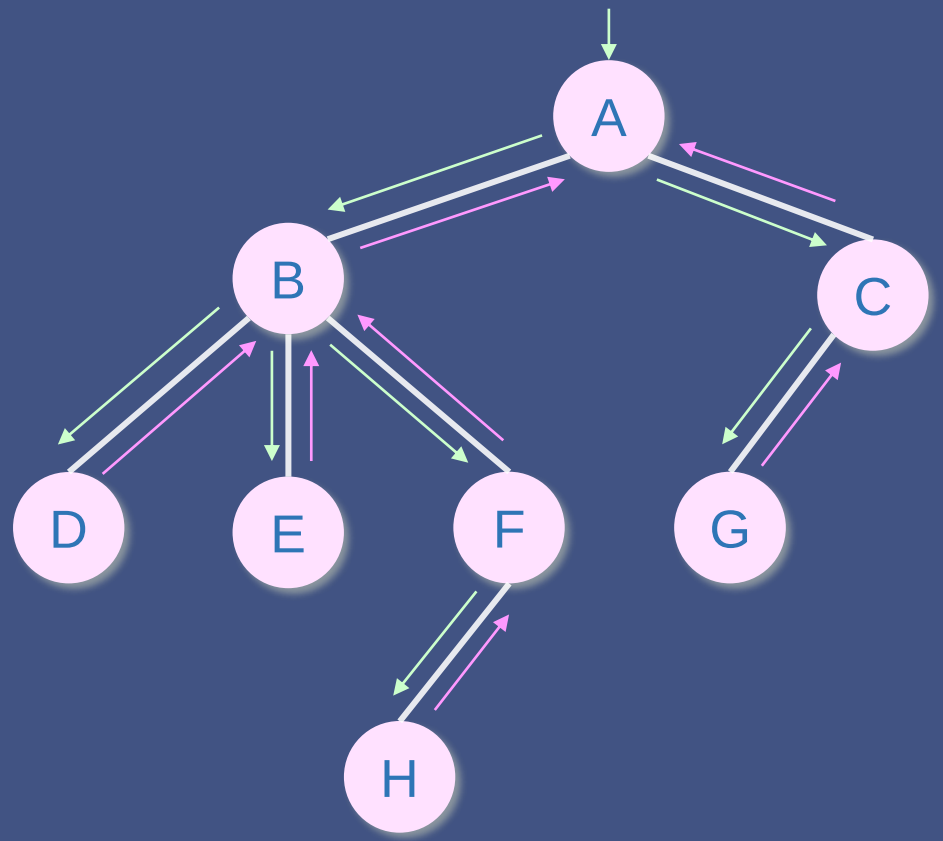
特点：每个点在序列中出现的次数等于该点的度数(根节点除外)
欧拉序中总共有2*n-1个点

```
void Dfs(int u,int fa)
{
    Euler[++Cnt]=u;
    for(int i=Last[u]; i; i=Next[i])
    {
        int v=End[i];
        if(v!=fa){Dfs(v,u);
        Euler[++Cnt]=u;}
    }
}
```

欧拉序：A B D B E B F I F B A C G C H C A

欧拉序的概念2

第二种欧拉序：对于点i,每当dfs到i时加进序列，dfs最后一次回溯到i时加进序列，i在序列中出现两次。
欧拉序中总共有2*n个点




```
void Dfs(int u,int fa)
{
    Euler[++Cnt]=u;
    for(int i=Last[u]; i; i=Next[i])
    {
        int v=End[i];
        if(v!=fa)Dfs(v,u);
    }
    Euler[++Cnt]=u;
}
```

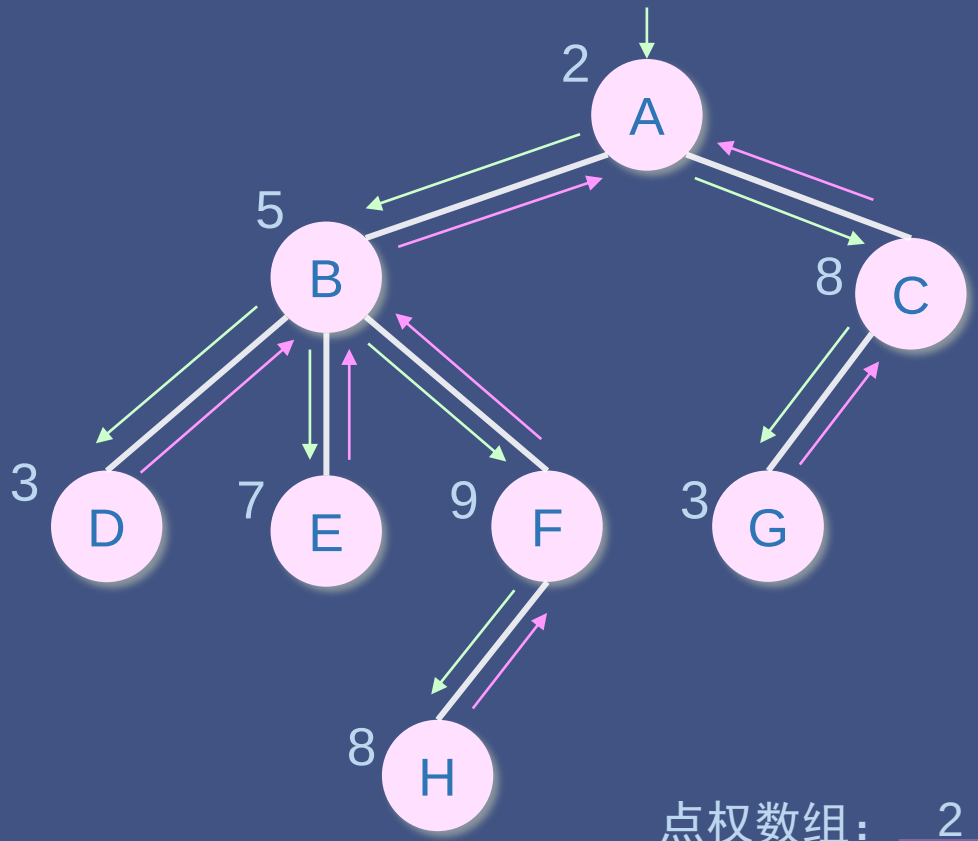
欧拉序： A B D D E E F H H F B C G G C A

欧拉序的作用：树上求和2

求某个点到根节点路径的点权和

方法是：每个点在欧拉序第一次出现处加点权，第二次出现处减点权。查询某点就只需要第一次出现位置的前缀即可。

第二种欧拉序：对于点i,每当dfs到i时加进序列，dfs最后一次回溯到i时加进序列，i在序列中出现两次。



以求点H到根的路径中的点权总和为例：

点权数组中，只在每个点第一次出现位置点权为正，，第二次出现位置的点权为负。

欧拉序中，H第一次出现的位置为8

对点权数组求前缀和Sum。

H到根路径的点权总和=Sum[8]=8+9+5+2=24

点权数组：	2	5	3	-3	7	-7	9	8	-8	-9	-5	8	3	-3	-8	-2
欧拉序：	A	B	D	D	E	E	F	H	H	F	B	C	G	G	C	A
编号：	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

欧拉序的小结

总结1：n个节点的无根树，无论怎样选择根节点以及深度优先搜索的顺序，其得到的欧拉序(第一种)长度一定为 2*n-1 。

总结2：完整的欧拉序是一棵有根树的映射。而对于树上任意一个节点，欧拉序中从它首次出现开始到它最后一次出现为止的子区间，恰好就是以该节点为根的子树的映射 。

```
void Dfs(int u,int fa)
{
    Euler[++Cnt]=u;
    for(int i=Last[u]; i; i=Next[i])
    {
        int v=End[i];
        if(v!=fa){Dfs(v,u);
        Euler[++Cnt]=u;}
    }
}
```

```
void Dfs(int u,int fa)
{
    Euler[++Cnt]=u;
    for(int i=Last[u]; i; i=Next[i])
    {
        int v=End[i];
        if(v!=fa)Dfs(v,u);
    }
    Euler[++Cnt]=u;
}
```

从欧拉序回到DFS序：

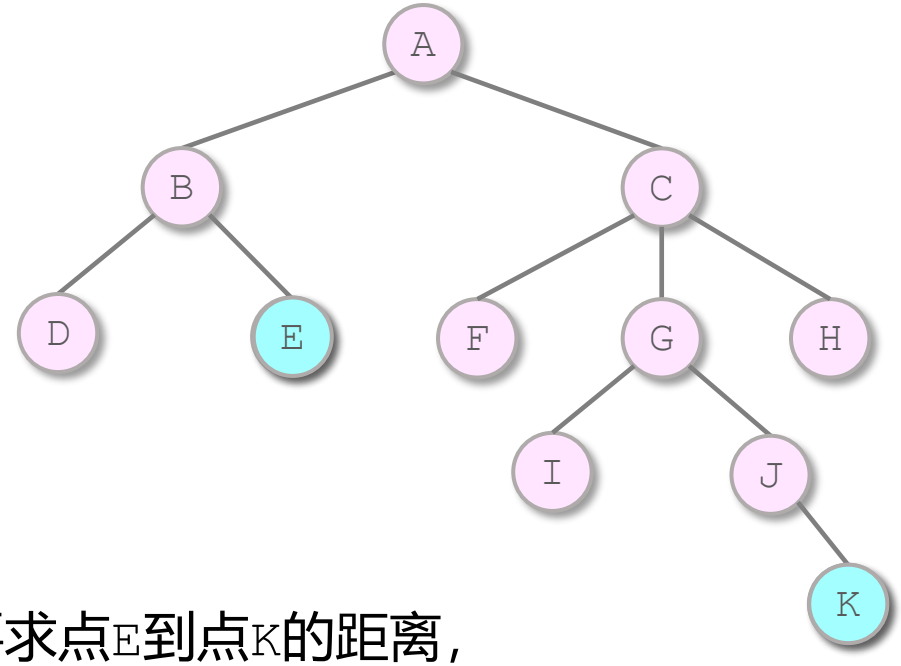
欧拉序：完整的欧拉序是一棵有根树的映射。而对于树上任意一个节点，欧拉序中从它首次出现开始到它最后一次出现为止的子区间，恰好就是以该节点为根的子树的映射 。

DFS序：同样是一棵有根树的映射，**一棵子树上的DFS序是连续的**，以x为根节点的子树的DFS序落在区间[in[x], out[x]]之内。

通过DFS序和欧拉序，我们很容易将子树问题转换成了区间问题。

括号序列与树上距离

dfs整棵树一遍，进入一个节点的时候加上一个左括号和节点编号，离开一个节点的时候加上一个右括号，这就是括号序列。
如左树的括号序：(A(B(D)(E))(C(F)(G(I)(J(K))))(H)))



要求点E到点K的距离，
截取序列中E到K这一子段：E))(C(F)(G(I)(J(K
去除字母：))()()(((
去除已配对的括号：))((((
剩余6个括号，则E到K的距离为6

```
//生成括号序列
void dfs(int u, int fa)
{
    S[++tot]=-1;//-1表示'('
    S[++tot]=u;
    Pos[u]=tot;
    for(int i=Lst[u];i;i=Nxt[i])
    {
      int v=End[i]
      if(v==fa)continue;
      dfs(v,u);
    }
    S[++tot]=-2;//-2表示')'
}
```

Break!

NKOJ1248 慢慢走

每天Farmer John的N头奶牛(1编号1...N)从粮仓走向他的自己的牧场。牧场构成了一棵树，粮仓在1号牧场。恰好有N-1条道路直接连接着牧场，使得牧场之间都恰好有一条路径相连。第i条路连接着Ai，Bi。

奶牛们每人有一个私人牧场Pi。粮仓的门每次只能让一只奶牛离开。耐心的奶牛们会等到他们的前面的朋友们到达了自己的私人牧场后才离开。首先奶牛1离开，前往P1；然后是奶牛2，以此类推。

当奶牛i走向牧场Pi时候，他可能会经过正在吃草的同伴旁。当路过已经有奶牛的牧场时，奶牛i会放慢自己的速度，防止打扰他的朋友。

Farmer John想知道，奶牛们总共要放慢多少次脚步。

1 <= N <= 100000

NKOJ1248 慢慢走

区间修改，单点查询问题

一. 预处理

1.按前序遍历的顺序求出DFS序，即求出每个节点的In[i]和Out[i];

2.对于数组In[ ]，实际是数字1到n由小到大的构成的一个数列。在这个数列中，树中i号点管辖的范围是[In[i],Out[i]]，即In[ ]值在这个区间中的点都属于i为根的子树。

3.我们建立一棵线段树来维护上面的数列，设线段树中节点p代表区间[L,R]，我们维护一个域Cover，记录p号点表示的区间被完全覆盖的次数;

二. 操作

1. 对于第i号牛，设它要去的点是j号点。而j号点管辖的范围是[In[j],Out[j]]。也就是说，后面出场的牛中，只要有牛要去的点在这个区间以内，都会打扰到i号牛吃草。

于是我们把线段树中被区间[In[j],Out[j]]覆盖的点的Cover值+1;

2. 查询i号牛放慢的次数：i号牛要去j号点，对应In[j]，我们在线段树中查询包含In[j]的区间被Cover的总次数即可;

NKOJ4331 树上操作

有一棵点数为 N 的树，以点 1 为根，且树点有边权。然后有 M 个操作，分为三种：
操作 1： 把某个节点 x 的点权增加 a 。
操作 2： 把某个节点 x 为根的子树中所有点的点权都增加 d 。
操作 3： 询问某个节点 x 到根的路径中所有点的点权和。

N,M<=100000 ， 且所有输入数据的绝对值都不会超过 10^6

DFS序+树状数组+差分

NKOJ4331 树上操作

一.通过一次DFS求出每个点的In[i]和Ou[i];
二.处理操作:
操作1：把点x的点权增加a
这会使x的任意后代y到根的距离都增加a，即dfs序在[In[x],Ou[x]]的点到根的距离都增加a;
树状数组1处理：modify1(In[x],a), modify1(Ou[x]+1,-a)即可。

操作2：点x为根的子树的所有点点权都增加d
这会使x的任意后代y到根的距离都增加(dep[y]-dep[x]+1)*d
dep[y]*d+(-dep[x]+1)*d
即看作dfs序在[In[x],Ou[x]]的点到根的距离都会增加(-dep[x]+1)*d;
树状数组1处理：modify1(In[x], (-dep[x]+1)*d), modify1(Out[x]+1, (dep[x]-1)*d);

对于dep[y]*d
即dfs序在[In[x],Ou[x]]的点到根的距离都会增加dep[y]*d
对于任意点y，dep[y]是已知的定值。只需再开一个树状数组记录d
树状数组2处理：modify2(In[x],d), modify2(Ou[x]+1,-d) 即树状数组2里面的数据都需要*dep[]

操作3：求点x到根的距离
ans=getSum2(In[x])*dep[x]+getSum1(In[x])

NKOJ5658 树

题目描述:
给定一棵大小为 n 的有根点权树,支持以下操作:

- 换根
- 修改点权
- 查询子树最小值

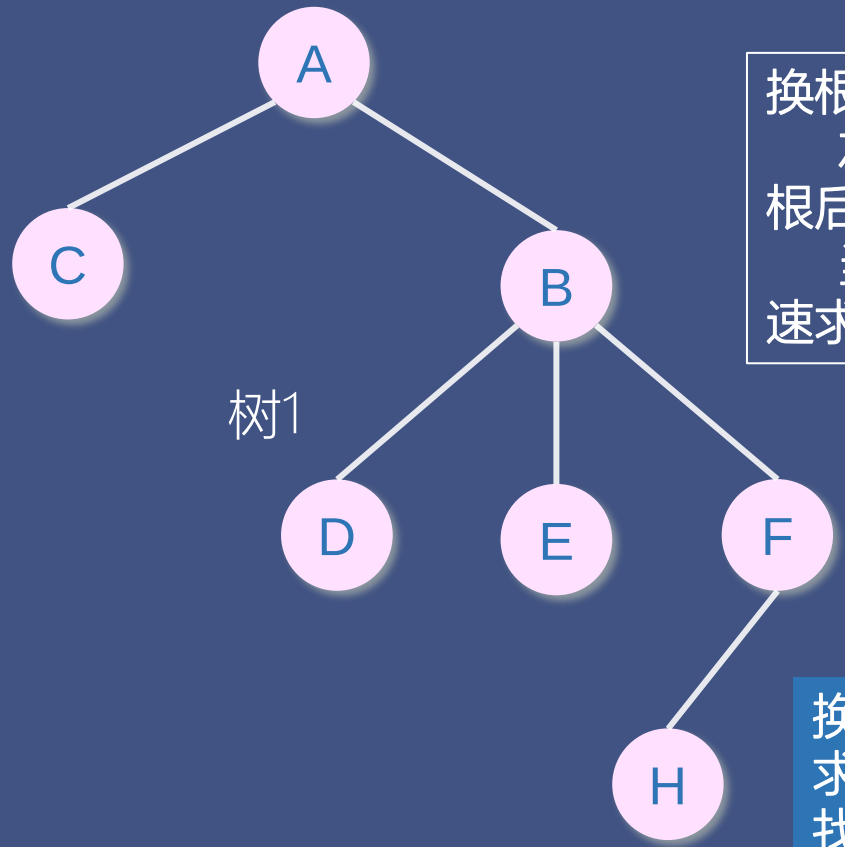
输入格式:
第一行两个整数 n, m ,分别表示树的大小和操作数。
接下来n行,每行两个整数f,v,第i+1行的两个数表示点i的父亲和点i的权。
接下来 m 行,为以下格式中的一种:

- V x y表示把点x的权改为 y
- E x 表示把有根树的根改为点 x
- Q x 表示查询点 x 的子树最小值

数据规模:
1<=n,m<=100000

欧拉序的作用：换根

第一种欧拉序：对于点i,每当dfs到i时加进序列，每当dfs回溯到i时加进序列。

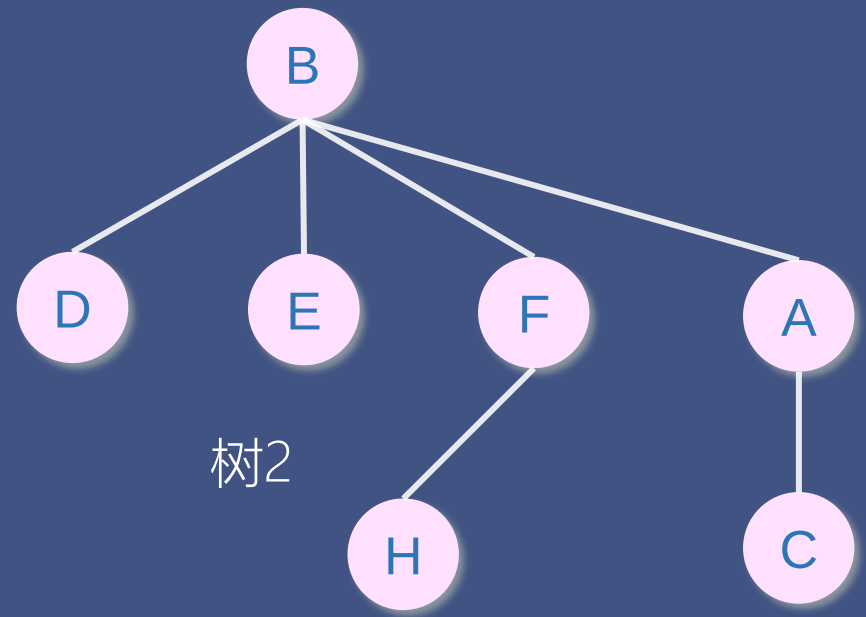


树1

换根：对于一棵无根树

左边树1是以A为根的形态，换根后，得到右边以B为根的树2。

当求出树1的欧拉序后，需要快速求出换根后的欧拉序。



树2

换根后求欧拉序的方法：

求出树1的欧拉序后(绿色)，将其翻倍(黄色，绿色最后位置和黄色开始位置重叠)。

找出B第一次出现的位置，往右数2*n-1步，这2*n-1个字符就是树2的欧拉序

欧拉序：A C A B D B E B F H F B A C A B D B E B F H F B A

Life is short, AC more!

- 聪明
- 乐观
- 勤奋
- 专注
- 妙趣横生
- 坚持就是胜利

See You Tonight!