信奥算法

快快编程1720

此题是**无根树**
还是**有根树**?

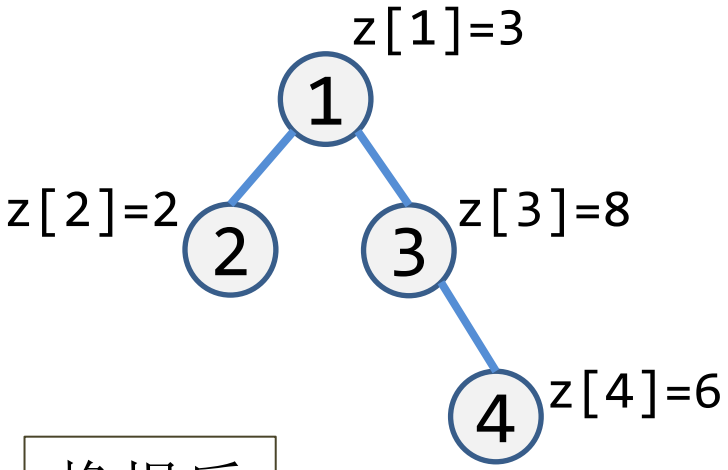
此题父子即邻居
"有根"理解贴合题意
"无根"/"换根"处理结果也一致

请同学核对题目大意
不能遗漏核心要点
1720

俄狄浦斯：已知一棵树有n个节点以及n-1个节点的父节点，i号节点对应正整数战斗力z[i]。要选出若干个节点组合，但是父子不可以同时被选中，求选中的节点总战斗力最大是多少。

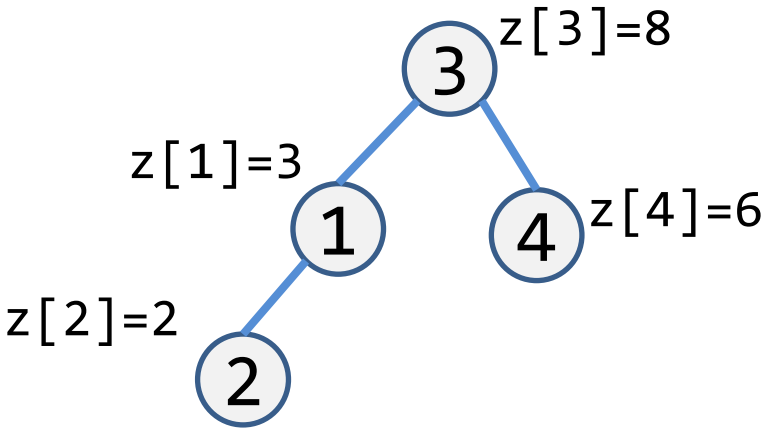
树上**最优**独立集问题

树上最优独立集



求左图的最优独立集

换根后等效



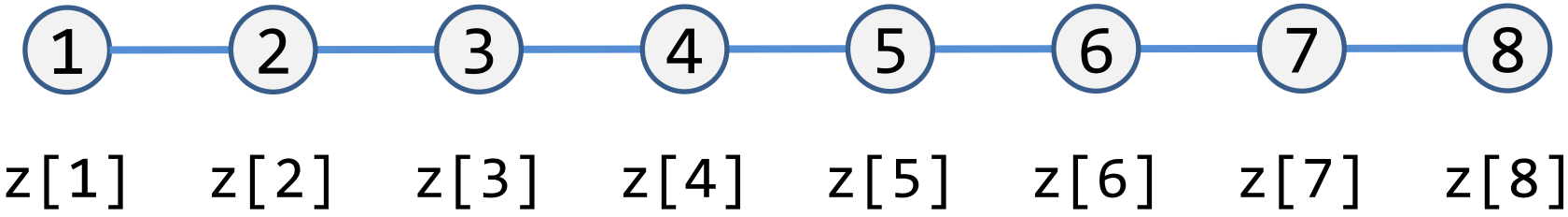
正确算法是什么？

树的简化

简化

启发
灵感

树上问题可以先简化为链状
链状问题解决后再推广到树形



链上最优独立集
如何求解?

动态
规划

复习
老题

打游戏

简化

启发
灵感

小明沉溺于打手机游戏无法自拔，严重影响了他的身体健康。妈妈虽然仍然允许他适度地打游戏，但是加了一条限制条件：不能连续两天都打游戏。
输入第一行是一个正整数n代表天数，第二行为n个正整数代表这连续n天每天可以打游戏的小时数。输出一个正整数，代表小明这些天最多能打多少小时时间的游戏。n<=100

输入样例：	输出样例：
4 3 1 1 2	5

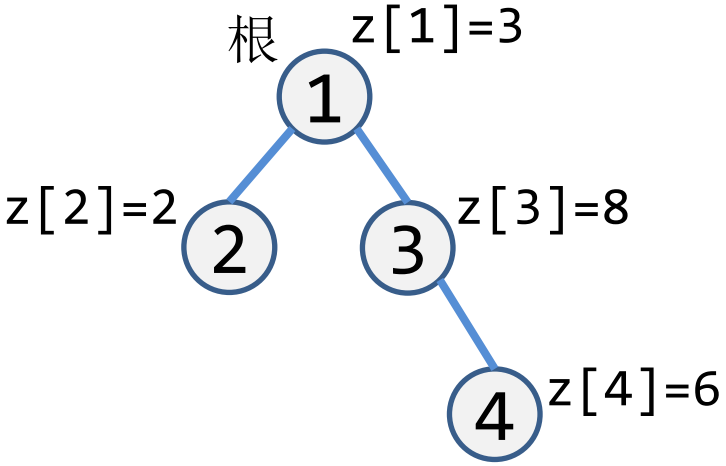
动态
规划

邀你玩游戏
你拒绝

树上最优独立集

动态规划

设计状态	f[u]代表u为根的子树的最优独立集总和
	但状态转移不方便，因为不清楚u是否选中



增加一维信息

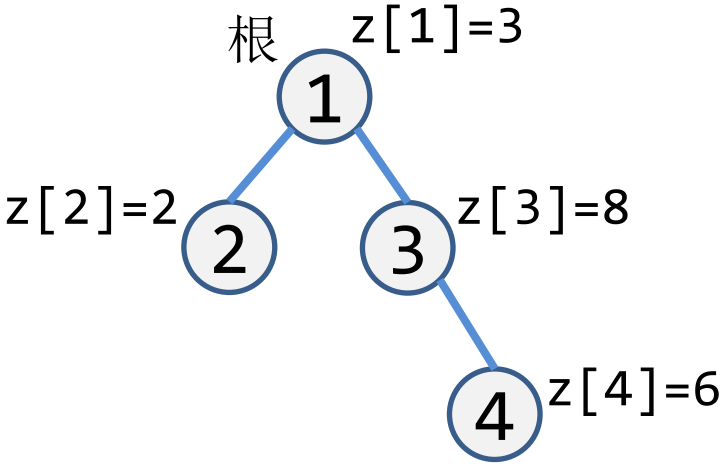
具体化状态

树上最优独立集

动态规划

设计状态	f[u][0]:不选u时,u为根的子树的最优独立集总和
	f[u][1]:选中u时,u为根的子树的最优独立集总和

原题的核心**决策**就是每个节点选不选



增加一维信息

具体化状态

将决策状态化

树上最优独立集

动态规划

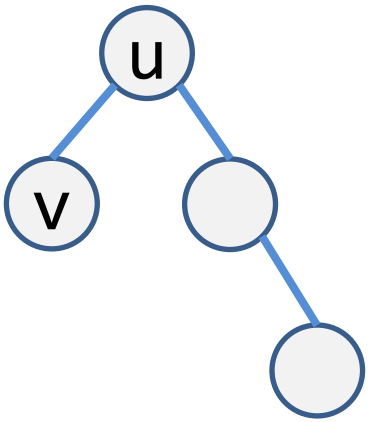
状态转移方程

如果选中u	u的儿子都不能选

$$f[u][1] = z[u] + \sum_{v是u儿子} f[v][0]$$

如果不选u	u的儿子选不选都行

$$f[u][0] = \sum_{v是u儿子} \max\{f[v][0], f[v][1]\}$$



树形DP常见转移方程
父子间依赖关系

代码1

邻接表

无根树

```
6  void dfs(int u,int fa){
7      f[u][1]=z[u];
8      f[u][0]=0;
9      for(int i=0;i<es[u].size();i++){
10         int v=es[u][i];
11         if(v==fa)continue;
12         dfs(v,u);
13         f[u][1]+=
14         f[u][0]+=max(f[v][0],f[v][1]);
15     }
16 }

28     dfs(1,0);
29     cout<<max(f[1][0],f[1][1])<<endl;
```

```
19      cin>>n;
20      for(int i=1;i<=n;i++)cin>>z[i];
21      for(int i=1;i<=n-1;i++){
22          int v,u;
23          cin>>v>>u;
24          p[v]=u;
25          son[u].push_back(v);
26      }
```

儿子列表是 邻接表的简化版本	son[u]里记录u的儿子节点 不包含u的父节点

使用 场景	父子关系明确
	例如:输入为p[u]信息

缺点
DFS不可以从 任意点开始
如何识别 唯一根节点?

代码2

儿子表

有根树

```
6  void dfs(int u){
7      f[u][1]=z[u];
8      f[u][0]=0;
9      for(int i=0;i<son[u].size();i++){
10         int v=son[u][i];
11         dfs(v);
12         f[u][1]+=f[v][0];
13         f[u][0]+=max(f[v][0],f[v][1]);
14     }
15 }
```

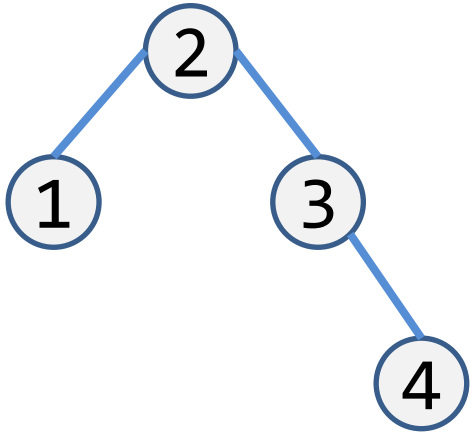
```
27     int rt=0;
28     for(int i=1;i<=n;i++)if(!p[i]){
29         rt=i; break;
30     }
31     dfs(rt);
32     cout<<                                <<endl;
```

快快编程861

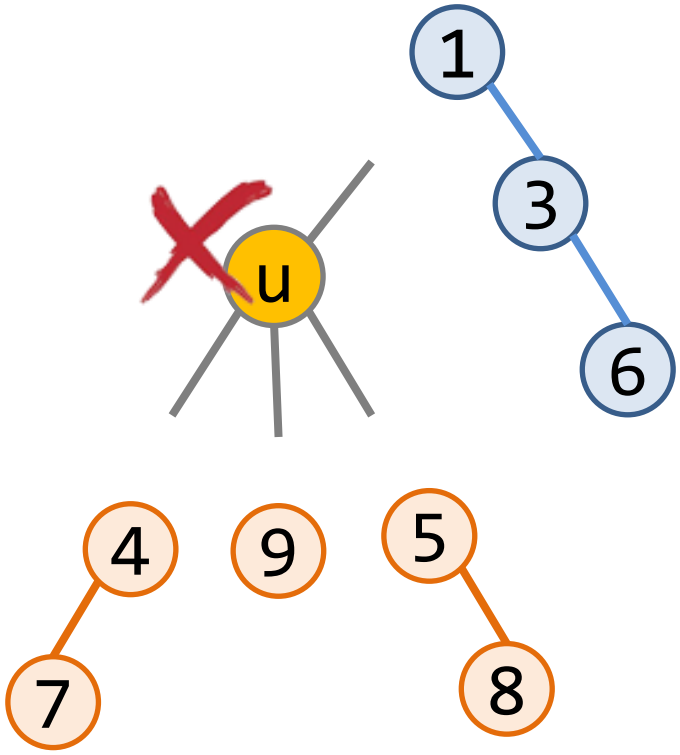
树的重心

树的重心也叫质心

树的重心可能有2个



用重心作为根节点
子树大小"最平衡"



枚举删除节点u

剩下连通分支最大节点数是几?

删除节点u后，连通分支共2类

u的儿子们带头	u的父亲带头

以u为根的子树内部	u的儿子们自立门户

$$\max_{v\text{是}u\text{儿子}}\{sz[v]\}$$

以u的儿子为根的子树大小求最大值

以u为根的子树外部	这块大小为多少?

$$n - sz[u]$$

补集转换
正难则反

树的重心

算法步骤

通过DFS过程计算出sz[]

sz[u]代表以u为根的子树大小

枚举删除节点u

计算$f[u]$: u儿子为根的**最大**子树的大小

$$f[u] = \max_{v\text{是}u\text{儿子}}\{sz[v]\}$$

计算$g[u]$:删除u后**最大**分支的大小

$$g[u] = max\{f[u], n - sz[u]\}$$

求$g[]$**最小**值的编号和大小

树的重心

通过DFS过程计算出sz[],p[]

```
6  void dfs(int u,int fa){
7      p[u]=fa;
8      sz[u]=1;
9      for(int i=0;i<es[u].size();++i){
10         int v=es[u][i];
11         if(v==fa)
12         dfs(v,u);
13
14     }
15 }
```

邻居是父亲怎么办

如何修改sz[u]

补全程序

树的重心

```
19      cin>>n;
20      for(int i=1;i<=n-1;++i){
21          int x,y;
22          cin>>x>>y;
23          es[x].push_back(y);
24          es[y].push_back(x);
25      }
26      dfs(1,0);
27      for(int u=1;u<=n;++u)
28          for(int i=0;i<es[u].size();++i){
29              int v=es[u][i];
30              
31              f[u]=
32          }
```

补全程序

树的重心

输出2个数：删除几号节点；剩下连通分支最大节点数至少是几。

如有多个相同方案，输出节点编号最小的

```
33      for(int u=1;u<=n;++u)
34          g[u]=
```

```
35      int id=min_element(g+1,g+1+n)-g;
36      cout<<   <<" "<<      <<endl;
```


补全程序

查错方法

树上递推/DP的查错方法

打印邻接表es[][]，确保输入和储存正确

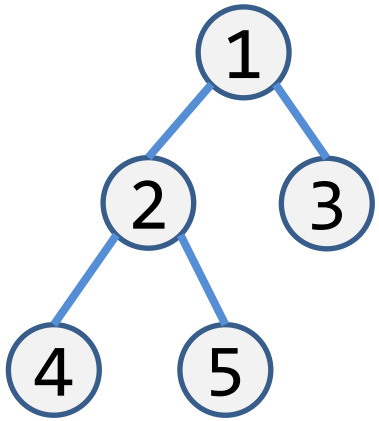
打印所有数组，对照手算数据

如p[],sz[],f[],g[]

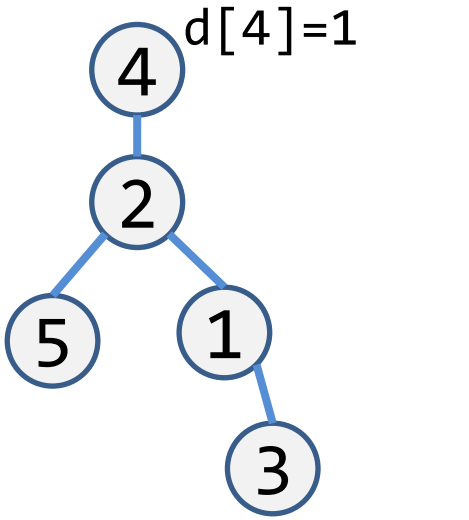
快快编程763

暴力DFS求树上路径

路径长度	共T次询问：求dst(x,y) 即x号节点和y号节点之间的路径有几条边?
x作为根	每次从x为起点DFS，直到遇到y
	顺路求出遇到节点的深度d[]
	x到y的路径长度 = d[y]-d[x]



要计算
dst(4,3)



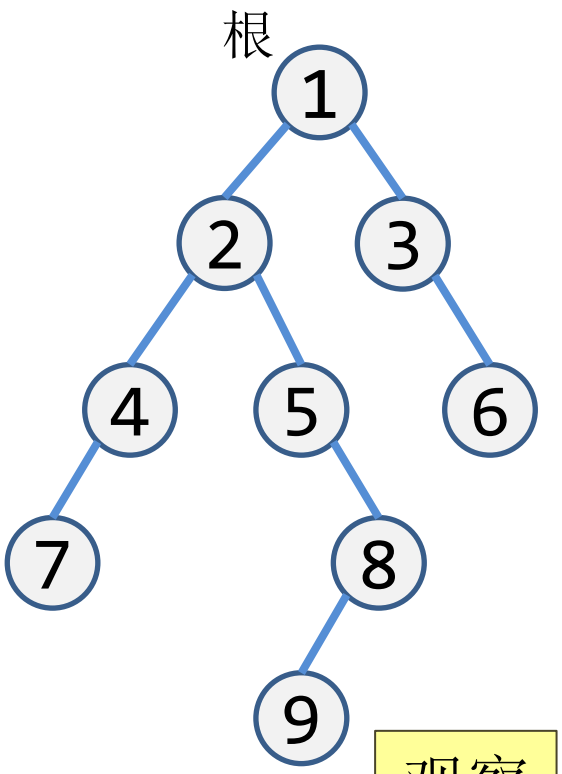
单次DFS 复杂度O(n)
总复杂度 O(Tn)
思考：时间 浪费在哪?

树上路径 + 深度

手算

启发灵感

d[u]代表节点u的深度



1到7的路径长度	d[7]-d[1]
1到5的路径长度	d[5]-d[1]
根到u的路径长度	d[u]-d[根]
6到7的路径长度	d[6]-d[1]+d[7]-d[1]
9到7的路径长度	d[9]-d[2]+d[7]-d[2]
4到8的路径长度	d[4]-d[2]+d[8]-d[2]

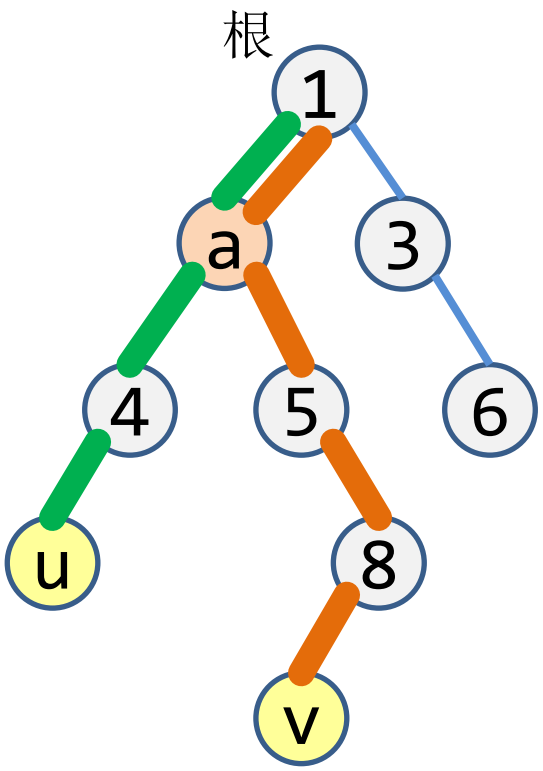
观察发现规律

路径必须经过u和v的**最近公共祖先**节点a

u到v的路径长度	d[u]+d[v]-2*d[a]

树上路径 + 深度 + 最近公共祖先

树上两点u和v之间的路径唯一确定



在u和v的路径中a的深度最小

a的所有祖先节点也都是u和v的祖先

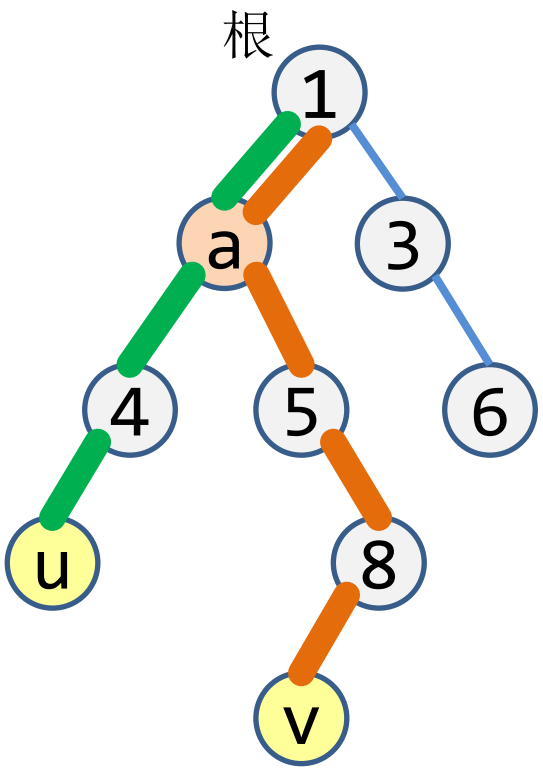
除了a以外，u和v的其他祖先不在路径上

Lowest Common Ancestor 简称**LCA**

路径必须经过u和v的
最近公共祖先节点LCA(u,v)

u到v的路径长度	d[u]+d[v]-2*d[LCA(u,v)]

最近公共祖先 LCA



求LCA是树上重要基本功

暴力算法

从u开始往上爬：u,4,a,1

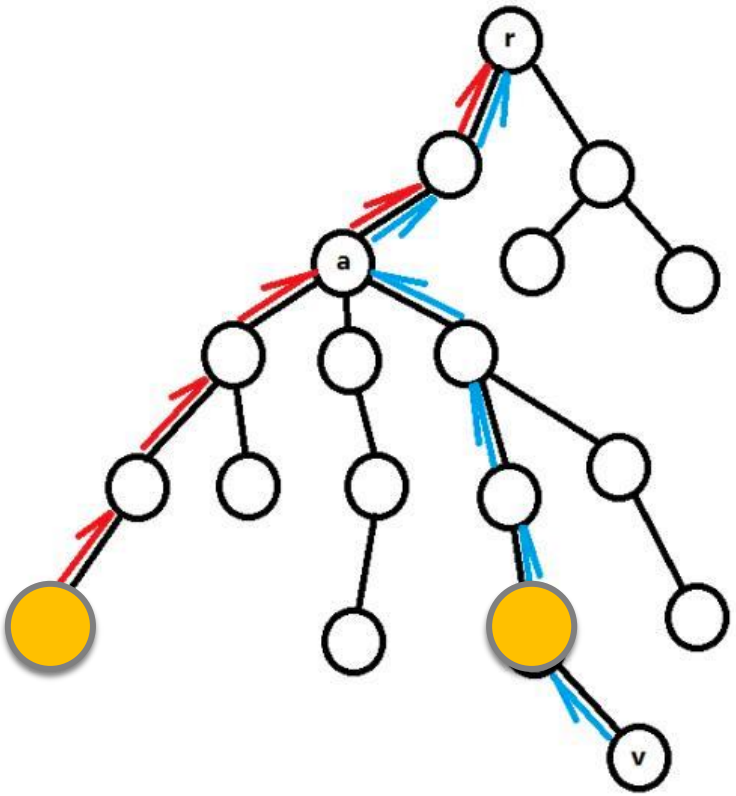
从v开始往上爬：v,8,5,a,1

求这两个序列的公共元素

最长公共后缀问题，可以二分查找

但求到根的整条序列太浪费时间了

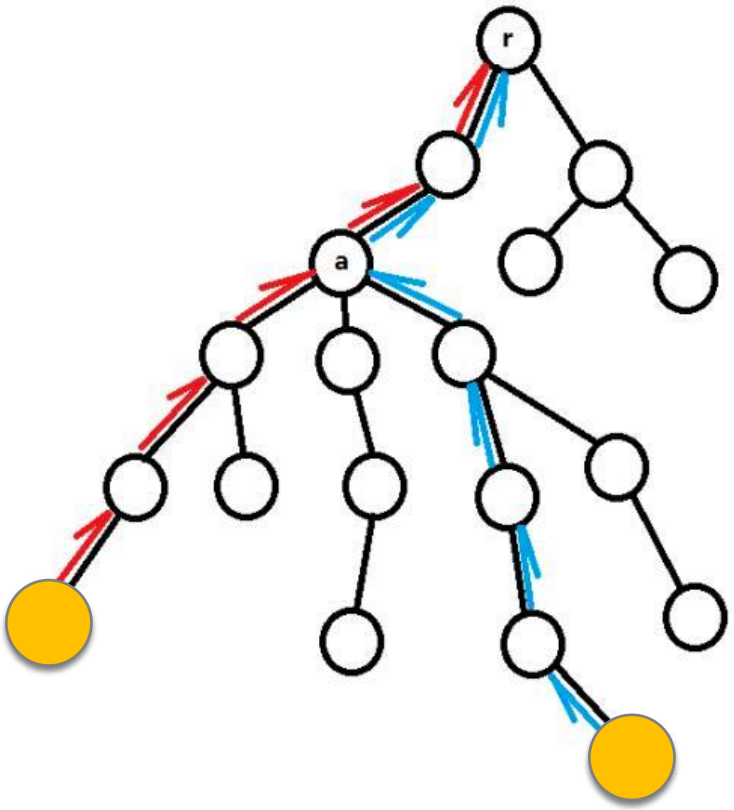
暴力LCA： 同步爬树



若u和v深度相同
d[u]==d[v]

从u和v同时向上爬
直到相遇

暴力LCA： 等候同伴



若u和v深度不同
d[u]!=d[v]

从较深节点先出发
直到两者深度相同

然后再同时向上爬
直到相遇

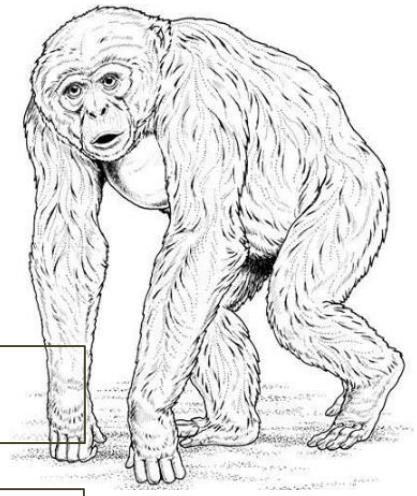
LCA + 树上路径

```
14 int lca(int u,int v){
15     while (d[u]>d[v]) u=p[u];
16     while (d[v]>d[u]) v=p[v];
17     while (u!=v) u=p[u],v=p[v];
18     return u;
19 }
```

lca函数独立封装

```
21 int dst(int u,int v){
22     return d[u]+d[v]-2*d[lca(u,v)];
23 }
```

LCA小结：暴力爬树



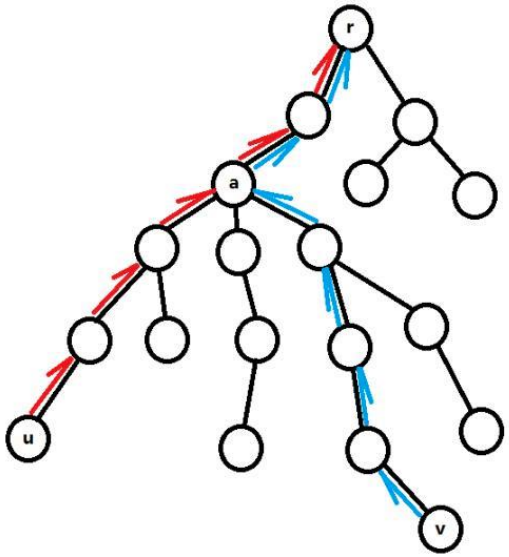
Lowest Common Ancestor 简称**LCA**

路径必须经过u和v的
最近公共祖先节点LCA(u,v)

预计算	先用DFS过程计算节点深度d[]
LCA(u,v)查询	等待同伴再齐头并进

dst(u,v)查询	d[u]+d[v]-2*d[LCA(u,v)]

树形问题建模



树的重心	利用子树大小sz[]

最近公共祖先LCA	利用节点深度d[]

太戈编程

1720
861
763

拓展题	1651,662,547