

# 长链剖分

## 算法简介

一种可以在线性时间复杂度维护子树中仅限深度有关的信息的算法，主要用于一些特殊的 \$text{dp}\$

## 算法思想

类似重链剖分，将儿子分为重儿子和轻儿子，重儿子组成的链构成长链

但不同的是重链剖分重儿子是子树结点最多的结点，长链剖分长儿子是子树深度最大的结点

长链剖分有两个重要性质

性质一：所有长链长度和为 \$n\$

显然所有点均属于且仅属于一条长链，所以性质一成立

性质二：长链剖分的一个重要性质是一个结点 \$x\$ 的 \$k\$ 级祖先所在的长链长度一定 \$\geq k\$

考虑结点 \$x\$ 和它的 \$k\$ 级祖先 \$y\$ 如果 \$x\$ 和 \$y\$ 属于同一条长链，该情况下性质二成立

如果 \$x\$ 和 \$y\$ 不属于同一条长链，知 \$y\$ 的重儿子子树的深度一定大于 \$x\$ 的深度，该情况下性质二也成立

## 算法应用

### 树上 \$k\$ 级祖先

#### 洛谷p5903

先一遍 dfs 处理出每个结点的深度、父结点、重儿子，同时用倍增法 \$O(\log n)\$ 时间处理出每个结点的 \$2^k\$ 级祖先

第二遍 dfs 处理出每个结点所在长链的起点 \$x\$

对每个长链的起点 \$x\$ 暴力处理出 \$x\$ 上下 \$len(x)\$ 个结点，其中 \$len(x)\$ 表示 \$x\$ 所在长链的长度

由于所有长链长度之和为 \$n\$ 所以该暴力处理的时间复杂度为 \$O(n)\$

对每次查询结点 \$x\$ 的 \$k\$ 级祖先，记 \$h\_k = \lfloor \log\_2 k \rfloor\$ \$tk = k - 2^{h\_k}\$

先从 \$x\$ 向上跳 \$2^{h\_k}\$ 级到 \$y\$ 根据长链剖分性质，知 \$y\$ 所在长链长度必然 \$\geq 2^{h\_k}\$

此时还需向上跳 \$tk\$ 级，\$tk < 2^{h\_k}\$ 所以 \$x\$ 的 \$k\$ 级祖先一定在 \$y\$ 所在长链起点的上下 \$len(x)\$ 级结点范围内

因此在从  $y$  跳到  $y$  所在长链起点，最后便可以  $O(\log n)$  访问目标结点

时间复杂度  $O(\log n)$

```
const int MAXN=5e5+5,MAXM=21;
unsigned int s;
inline unsigned int get(unsigned int x){
    x^=x<<13;
    x^=x>>17;
    x^=x<<5;
    return s=x;
}
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],m;
void Insert(int u,int v){
    edge[++m].to=v;
    edge[m].next=head[u];
    head[u]=m;
}
int d[MAXN],fa[MAXN][MAXM],log_2[MAXN];
int h_son[MAXN],mson[MAXN],p[MAXN];
vector<int> Node_up[MAXN],Node_down[MAXN];
void get_log2(){
    log_2[0]=-1;
    for(i,1,MAXN)
        log_2[i]=log_2[i>>1]+1;
}
void dfs_1(int u,int depth){
    mson[u]=d[u]=depth;
    _rep(i,1,log_2[d[u]])
        fa[u][i]=fa[fa[u][i-1]][i-1];
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs_1(v,depth+1);
        if(mson[v]>mson[u]){
            h_son[u]=v;
            mson[u]=mson[v];
        }
    }
}
void dfs_2(int u,int top){
    p[u]=top;
    if(u==top){
        for(int i=0, pos=u; i<=mson[u]-d[u]; pos=fa[pos][0], i++)
            Node_up[u].push_back(pos);
        for(int i=0, pos=u; i<=mson[u]-d[u]; pos=h_son[pos], i++)
            Node_down[u].push_back(pos);
    }
}
```

```

}
if(h_son[u])
dfs_2(h_son[u],top);
for(int i=head[u];i;i=edge[i].next){
    int v=edge[i].to;
    if(v==h_son[u])
        continue;
    dfs_2(v,v);
}
int query(int u,int k){
    if(!k) return u;
    u=fa[u][log_2[k]],k-=1<<log_2[k];
    k-=d[u]-d[p[u]],u=p[u];
    return k>=0?Node_up[u][k]:Node_down[u][-k];
}
int main()
{
    int n,q,root,x,y;
    cin>>n>>q>>s;
    _rep(i,1,n){
        fa[i][0]=read_int();
        if(fa[i][0])
            Insert(fa[i][0],i);
        else
            root=i;
    }
    get_log2();
    dfs_1(root,0);
    dfs_2(root,root);
    long long tot=0,last=0;
    int u,k;
    _rep(i,1,q){
        u=(get(s)^last)%n+1;
        k=(get(s)^last)%(d[u]+1);
        last=query(u,k);
        tot^=last*i;
    }
    enter(tot);
    return 0;
}

```

## 合并信息

[洛谷p5904](#)

给定一棵  $n$  个结点的树，问有多少个无序点对  $(i,j,k)$  满足  $i,j,k$  两两间距离相等

树形 $\text{dp}$ 状态转移方程可以参考这份博客[详情](#)

利用指针位移使得父结点以  $O(1)$  时间继承重儿子信息，暴力继承轻儿子信息

设当前结点为  $u$  的儿子结点为  $x$  每次暴力继承时间复杂度为  $O(\text{left}(\text{text}\{\text{len}\})\text{left}(x)\text{right})$

总时间复杂度为  $\sum_u \left( \left( \sum_x \text{left}(\text{text}\{\text{len}\})\text{left}(x)\text{right} \right) \text{right} - \text{left}(\text{text}\{\text{len}\})\text{left}(u)\text{right} + 1 \right)$

每个结点仅在  $\sum_u \left( \left( \sum_x \text{left}(\text{text}\{\text{len}\})\text{left}(x)\text{right} \right) \text{right} \right)$  作为它的父结点的子结点出现一次

所以有  $\sum_u \left( \left( \sum_x \text{left}(\text{text}\{\text{len}\})\text{left}(x)\text{right} \right) \text{right} \right) = \sum_u \text{left}(\text{text}\{\text{len}\})\text{left}(u)\text{right} - \text{left}(\text{text}\{\text{len}\})(\text{root})$

所以总时间复杂度为  $O(n\text{right})$

```
const int MAXN=5e5+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],m;
void Insert(int u,int v){
    edge[++m].to=v;
    edge[m].next=head[u];
    head[u]=m;
}
int d[MAXN],Son[MAXN],len[MAXN];
LL *f[MAXN],*g[MAXN],dp[MAXN<<2],*pos=dp,ans;
void Malloc(int u){
    f[u]=pos,pos+=len[u]<<1;
    g[u]=pos,pos+=len[u]<<1;
}
void dfs_1(int u,int depth,int fa){
    d[u]=depth;Son[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
            continue;
        dfs_1(v,depth+1,u);
        if(len[v]>len[Son[u]])
            Son[u]=v;
    }
    len[u]=len[Son[u]]+1;
}
void dfs_2(int u,int fa){
    if(Son[u]){
        f[Son[u]]=f[u]+1,g[Son[u]]=g[u]-1;
        dfs_2(Son[u],u);
    }
    f[u][0]=1,ans+=g[u][0];
```

```
for(int i=head[u];i;i=edge[i].next){  
    int v=edge[i].to;  
    if(v==fa||v==Son[u])  
        continue;  
    Malloc(v);  
    dfs_2(v,u);  
    _for(i,0,len[v]){
        if(i) ans+=f[u][i-1]*g[v][i];
        ans+=g[u][i+1]*f[v][i];
    }  
    _for(i,0,len[v]){
        g[u][i+1]+=f[u][i+1]*f[v][i];
        if(i) g[u][i-1]+=g[v][i];
        f[u][i+1]+=f[v][i];
    }
}  
}  
int main()  
{  
    int n=read_int(),u,v,root=1;  
    _for(i,1,n){
        u=read_int();v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    dfs_1(root,0,0);
    Malloc(root);
    dfs_2(root,0);
    enter(ans);
    return 0;
}
```

From:  
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:  
[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E9%95%BF%E9%93%BE%E5%89%96%E5%88%86](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E9%95%BF%E9%93%BE%E5%89%96%E5%88%86)

Last update: 2020/07/27 22:54

