树同构

无根树同构

对每棵树，至多只有两个重心。显然两棵无根树同构等价于以两个无根树的某个重心为根的两个有根树同构。

于是无根树同构问题可以转化为有根树同构问题，故下面只考虑有根树同构问题。

树哈希

算法简介

利用哈希 $O(n)$ 时间判断树同构，不能保证完全正确。

算法思想

构造各种哈希函数，当所有哈希值相同时才认为同构，下面给出两种哈希函数。

$$h_{1,u}=\text{sz}_u+\sum_{v\in \text{son}(u)}\text{sz}_vh_{1,v}$$

$$h_{2,u}=\text{sz}_u+\text{sz}_u\sum_{v\in \text{son}(u)}\text{sz}_vh_{2,v}$$

如果需要更高准确率，可以考虑构造 $f\{1,2\cdots n\}\to \{p_1,p_2\cdots p_n\}$并将上式 sz_u 替换为 p_{sz_u}其中 p 序列可以为随机数表，素数表等等。

算法模板

洛谷p5043

```
const int MAXN=55,Mod=1e9+7;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int sz[MAXN],mson[MAXN],tot_sz,root_sz;
int h1[MAXN],h2[MAXN];
vector<int> rt;
vector<pair<int,int> >root[MAXN];
void find_root(int u,int fa){
```

```
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz)
    root_sz=mson[u];
}
void Hash(int u,int fa){
    sz[u]=1,h1[u]=h2[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        Hash(v,u);
        sz[u]+=sz[v];
        h1[u]=(h1[u]+1LL*h1[v]*sz[v])%Mod;
        h2[u]=(h2[u]+1LL*h2[v]*sz[v])%Mod;
    }
    h1[u]=(h1[u]+sz[u])%Mod;
    h2[u]=(1LL*h2[u]*sz[u]+sz[u])%Mod;
}
bool cmp(int a,int b){
    if(root[a].size()!=root[b].size())
    return false;
    if(root[a].size()==1){
        if(root[a][0]==root[b][0])
        return true;
    }
    else{
        if(root[a][0]==root[b][0]&&root[a][1]==root[b][1])
        return true;
        if(root[a][0]==root[b][1]&&root[a][1]==root[b][0])
        return true;
    }
    return false;
}
int main()
{
    int T=read_int(),n,p;
    _rep(k,1,T){
        n=read_int(),edge_cnt=0;
        _rep(i,1,n)head[i]=0;
        _rep(i,1,n){
            p=read_int();
            if(p)Insert(p,i),Insert(i,p);
```

```
        }
        rt.clear();
        tot_sz=root_sz=n;
        find_root(1,0);
        _rep(i,1,n){
            if(mson[i]==root_sz)
            rt.push_back(i);
        }
        _for(i,0,rt.size()){
            Hash(rt[i],0);
            root[k].push_back(make_pair(h1[rt[i]],h2[rt[i]]));
        }
        _rep(i,1,k){
            if(cmp(i,k)){
                enter(i);
                break;
            }
        }
    }
    return 0;
}
```

最小表示法

算法简介

利用每棵树的最小字典序的括号序列判断树同构。

算法思想

易知两棵树同构等价于两棵树最小字典序的括号序列相同。当然实现的时候不需要真正模拟括号序列，可以考虑使用 01 串。

于是考虑如何快速求出每棵树的最小表示法。

一个比较暴力的算法是先求出所有子树的最小表示法，然后用 Trie 树实现 $O(n)$ 排序，然后顺次拼接，最后最外层添加括号。

上述算法的时间复杂度为 $O(n^2)$□考虑优化算法。

发现可以用名次代替括号序列。考虑根据深度化为树，每个深度为 d 的结点的名次序列由深度为 $d+1$ 的结点的名次构成。

注意名次是指在两棵树中所有深度为 $d+1$ 的结点中的名次，初始叶子结点名次均为 0。

然后对所有深度为 d 的结点的名次序列再次排序，如果使用基数排序，可以使得总时间复杂度为 $O(n)$□

算法练习

习题一

牛客暑期多校(第十场) J 题

题意

给定两棵带标号有根树，保证同构。要求修改两棵树的标号，将两棵树变成完全相同的树，问最小修改次数。

题解

先树哈希求出每棵子树的哈希值，再考虑 dp 定义 $\text{dp}(u_1,u_2)$ 为将以 $u1,u2$ 为根的两棵树匹配时最多的标号相同的位置个数。

对两棵同构的树，暴力考虑枚举所有儿子结点。将所有的同构的儿子 $v1,v2$ 结点连上一条权值为 $\text{dp}(v_1,v_2)$ 的边，然后跑最大二分图匹配。

注意如果根节点标号相同需要将最大匹配结果加一。

最终答案为 $n-\text{dp}(\text{root}_1,\text{root}_2)$ 时间复杂度 $O\left(\sum (\text{son}^2+\text{son}^3)\right)=O(n^3)$

```
const int MAXN=1005,Mod=1e9+7;
namespace KM{
    const int MAXN=505,Inf=1e9;
    int n,w[MAXN][MAXN],lx[MAXN],ly[MAXN],lack[MAXN],p[MAXN];
    int linkx[MAXN],linky[MAXN],visx[MAXN],visy[MAXN];
    void init(int _n){
        n=_n;
        _rep(i,1,n)
        ly[i]=linkx[i]=linky[i]=visx[i]=visy[i]=0;
        _rep(i,1,n)
        _rep(j,1,n)
        w[i][j]=0;
    }
    void augment(int pos){
        int temp;
        while(pos){
            temp=linkx[p[pos]];
            linkx[p[pos]]=pos;
            linky[pos]=p[pos];
            pos=temp;
        }
    }
```

```
    void bfs(int s,int k){
        queue<int>q;
        _rep(i,1,n)
        lack[i]=Inf;
        q.push(s);
        while(true){
            while(!q.empty()){
                int u=q.front();q.pop();
                visx[u]=k;
                _rep(v,1,n){
                    if(visy[v]==k)
                    continue;
                    if(lx[u]+ly[v]-w[u][v]<lack[v]){
                        lack[v]=lx[u]+ly[v]-w[u][v];p[v]=u;
                        if(!lack[v]){
                            visy[v]=k;
                            if(!linky[v])
                            return augment(v);
                            else
                            q.push(linky[v]);
                        }
                    }
                }
            }
            int a=Inf;
            _rep(i,1,n) if(visy[i]!=k)
            a=min(a,lack[i]);
            _rep(i,1,n){
                if(visx[i]==k)lx[i]-=a;
                if(visy[i]==k)ly[i]+=a;
                else lack[i]-=a;
            }
            _rep(i,1,n){
                if(visy[i]!=k&&!lack[i]){
                    visy[i]=k;
                    if(!linky[i])
                    return augment(i);
                    else
                    q.push(linky[i]);
                }
            }
        }
    }
    int get_pair(){
        int k=0;
        _rep(i,1,n){
            lx[i]=-Inf;
            _rep(j,1,n)
            lx[i]=max(lx[i],w[i][j]);
        }
        _rep(i,1,n)
```

```
        bfs(i,i);
        int ans=0;
        _rep(i,1,n)
        ans+=w[linky[i]][i];
        return ans;
    }
}
struct Edge{
    int to,next;
}edge[MAXN<<1];
int n,head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int root[2],sz[MAXN],mson[MAXN],tot_sz,root_sz;
int h1[MAXN],h2[MAXN],dp[MAXN][MAXN];
void Hash(int u){
    sz[u]=1,h1[u]=h2[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        Hash(v);
        sz[u]+=sz[v];
        h1[u]=(h1[u]+1LL*h1[v]*sz[v])%Mod;
        h2[u]=(h2[u]+1LL*h2[v]*sz[v])%Mod;
    }
    h1[u]=(h1[u]+sz[u])%Mod;
    h2[u]=(1LL*h2[u]*sz[u]+sz[u])%Mod;
}
bool cmp(int a,int b){return (h1[a]==h1[b])&&(h2[a]==h2[b]);}
void dfs(int u1,int u2){
    if(~dp[u1][u2])return;
    int cnt=0;
    for(int i=head[u1];i;i=edge[i].next){
        int v1=edge[i].to;
        for(int j=head[u2];j;j=edge[j].next){
            int v2=edge[j].to;
            if(cmp(v1,v2))dfs(v1,v2);
        }
        cnt++;
    }
    KM::init(cnt);
    for(int i=head[u1],pos1=1;i;i=edge[i].next,pos1++){
        int v1=edge[i].to;
        for(int j=head[u2],pos2=1;j;j=edge[j].next,pos2++){
            int v2=edge[j].to;
            if(cmp(v1,v2))
            KM::w[pos1][pos2]=dp[v1][v2];
        }
    }
```

```
    dp[u1][u2]=KM::get_pair()+((u1+n)==u2);
}
int main()
{
    n=read_int();
    _rep(i,1,n){
        int p=read_int();
        if(p)Insert(p,i);
        else root[0]=i;
    }
    Hash(root[0]);
    _rep(i,1,n){
        int p=read_int()+n;
        if(p!=n)Insert(p,i+n);
        else root[1]=i+n;
    }
    Hash(root[1]);
    mem(dp,-1);
    dfs(root[0],root[1]);
    enter(n-dp[root[0]][root[1]]);
    return 0;
}
```