提高组300分试题 第一组题解

猴猴吃苹果

题解

直接说具体做法，做法中蕴含的意义，请大家自行理解。

1、对整棵树做DFS，并记录节点深度。

2、按照节点深度及编号对节点排序，深度越大排序越靠前，深度相同时，编号小的排在前面。

3、按照排好序的节点顺序，逐个处理节点，从叶子结点向上走到父节点，并对已走过的节点标注为已访问。直到走到某个已访问的节点或者根节点后停止，记录走过的步数。

4、按照走过的步数及编号，对节点做第二次排序，得到的结果即为答案。

排序可以使用基数排序，因此复杂度为O(n)。

标准代码

c++

```
#include <bits/stdc++.h>
using namespace std;

int N, K, L, t;
int parent[50000];
int length[50000];
vector<vector<int> > tree;
vector<vector<int> > res;
vector<vector<int> > tmp;
int main()
{
    cin >> N >> K;
    tree.resize(N); tmp.resize(N); res.resize(N);
    for(int i = 1; i < N; ++i) {
        scanf("%d", &t);
        tree[t].push_back(i);
        tree[i].push_back(t);
    }
    vector<int> st;
    st.push_back(K);
    parent[K] = K;
    length[K] = 0;
    for(; !st.empty(); ) {
```

```
            t = st.back();
            st.pop_back();
            for(int i = 0; i < tree[t].size(); ++i) {
                if(tree[t][i] != parent[t]) {
                    parent[tree[t][i]] = t;
                    length[tree[t][i]] = length[t] + 1;
                    st.push_back(tree[t][i]);
                }
            }
        }
        for(int i = 0; i < N; ++i) {
            if(tree[i].size() == 1) {
                res[length[i]].push_back(i);
            }
        }
        for(int i = N - 1; i > 0; --i) {
            for(int j = 0; j < res[i].size(); ++j) {
                int len = 0;
                int cur = res[i][j];
                for(; length[cur] != 0; ++len) {
                    length[cur] = 0;
                    cur = parent[cur];
                }
                length[res[i][j]] = len;
            }
            res[i].clear();
        }
        for(int i = 0; i < N; ++i) {
            if(tree[i].size() == 1) {
                res[length[i]].push_back(i);
            }
        }
        printf("%d\n", K);
        for(int i = N - 1; i > 0; --i) {
            for(int j = 0; j < res[i].size(); ++j) {
                printf("%d\n", res[i][j]);
            }
        }
            return 0;
}
```

猴猴吃香蕉

题解

类似于背包的DP，以乘积为状态。先把等选数字里面不是K约数的去掉。然后找出K的约数，进行离散化。然后dp[i][j]表示前i个数字乘积为j的状态。 $Dp[i+1][j*a[i+1]]+=dp[i][j]$.
$Dp[i+1][j]+=dp[i][j]$; 总的复杂度是 $O(n*d(k)*log(d(k)))$ $D(k)$ 表示 k 的因子数目。多一个 log 是因为离散化了，对应下标的时候要二分查找。

标准代码

c++

```
#include <stdio.h>
#include <map>
using namespace std;
typedef map< int, int > MyMap;
#define MAXN 1000
#define MAXSQRTK 10000
#define P 1000000007

void get_factor( int k, MyMap& f ) {
    for( int i = 1; i * i <= k; ++i ) {
        if( k % i == 0 ) {
            f[i] = f[k/i] = 0;
        }
    }
}

int main() {

    int t, i, n, k, y;
    MyMap  factor;
    MyMap::reverse_iterator it;

    for( scanf( "%d", &t ); t--; ) {
        scanf( "%d%d", &n, &k );
        factor.clear();
        get_factor( k, factor );
        factor[1] = 1;

        for( i = 0; i < n; ++i ) {
            scanf( "%d", &y );
            if( k % y != 0 ) continue;
            for( it = factor.rbegin(); it != factor.rend(); ++it ) {
                if( it->first >= y && it->first % y == 0 ) {
                    it->second = ( it->second + factor[it->first/y] ) % P;
                }
            }
        }
        printf( "%d\n", factor[k] );
    }

    return 0;
}
```

猴猴的比赛

题解

我们可以通过dfs序对第一棵树进行编号，得到每个节点本身的编号以及子树的编号范围（L_i，R_i）。 用树状数组来维护后面的计算。 在DFS处理第二棵树时，我们每处理完一个节点，就将该节点在第一棵树中的对应编号设为1，处理完所有子节点后，查询编号区间 (L_i, R_i) 的区间和，即当前子树下，有多少个节点在树1中也是当前节点的子孙节点。维护计数即可。 复杂度 $nlog(n)$ 。

标准代码

c++

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
#define N 500000 + 10

int t,n;
vector<int> a[N], b[N];
int cnt=1, num2[N], out[N];
int mk[N], d[2 * N];
int res[N];
int ww=0;
void DFS2(int u, int prev)
        {
                num2[u]=cnt++;
                for (int i=0; i < b[u].size(); i++)
                        {
                int v = b[u][i];
                            if (v == prev) continue;
                                if (num2[v] == 0)
                                        DFS2(v, u);
                        }
        out[u]=cnt++;
        }
void update(int x, int val)
{
        while (x <= 2 * n)
            d[x]+=val, x+=x & (-x);
}
int get(int x)
{
        int rs=0;
        while (x > 0)
            rs+=d[x], x-=x & (-x);
        return rs;
}
void DFS1(int u, int prev)
    {
        res[u]=get(num2[u]);
        update(num2[u],1);
        update(out[u],-1);
```

```
        for (int i=0; i < a[u].size(); i++)
            {
                int v=a[u][i];
                if (v == prev) continue;
                DFS1(v, u);
            }
        update(num2[u],-1);
        update(out[u],1);
    }
int main()
{
        scanf("%d", &n);
        for (int i=1; i <= n; i++)
        {
                a[i].clear();
                b[i].clear();
                num2[i]=out[i] =d[i] =res[i]=0;
                cnt = 1;
        }
        for (int i=1; i < n; i++)
        {
                int u,v;
                scanf("%d%d", &u, &v);
                a[u].push_back(v);
                a[v].push_back(u);
        }
        for (int i=1; i < n; i++)
        {
                int u,v;
                scanf("%d%d", &u, &v);
                b[u].push_back(v);
                b[v].push_back(u);
        }
        DFS2(1, 0);
        DFS1(1, 0);
        long long ans = 0;
        for (int i=1; i <= n; i++) {
                ans += res[i];
        }
        printf("%lld\n", ans);
        return 0;
}
```