Crack a Hack

Maximum Disjoint Subtree Product

Course: Algorithmic Problem Solving

Course Code: 17ECSE309

By :

Name: Akash Pujar

USN: 01FE15BCS015

Problem:

We define an unrooted tree, T, with the following properties:

- T is a connected graph with nodes connected by n-1 edges.
- Each node has a distinct ID number from 1 to n, and each node ID i has a value w(i).

We define a subtree of T to be a connected part of T. Two subtrees T(a) and T(b) are disjoint if they don't contain any common nodes.

Sum of a subtree is the sum of the w(i) values for each node i belonging to the subtree.

Given the configuration of tree T, find and print the maximum possible product of the sums of two disjoint subtrees in T(i.e. sum[t(a)].sum[t(b)]).

Code(C++):

```
#include <bits/stdc++.h>
using namespace std;
const int N=300009;
int n,a[N];
vector <int> v[N];
int d[N],da[N];
```

```
int p[N];
long long ans;

void dfs1(int x,int fr){
    d[x]=a[x];
    da[x]=0;
    int A=0,B=0;
    for (int i=0;i<v[x].size();i++)
    if (v[x][i]!=fr){
        dfs1(v[x][i],x);
        if (d[v[x][i]]>0)
            d[x]+=d[v[x][i]];
        da[x]=max(da[x],da[v[x][i]]);
        if (B<da[v[x][i]]){
            B=da[v[x][i]];
            if (B>A) swap(B,A);
        }
    }
    ans=max(ans,1ll*A*B);
    d[x]=max(d[x],0);
    da[x]=max(da[x],d[x]);
}
```

```
void dfs2(int x,int fr){
    ans=max(ans,1ll*da[x]*p[x]);
    for (int i=0;i<v[x].size();i++)
    if (v[x][i]!=fr){
        p[v[x][i]]=max(0,p[x]+d[x]-d[v[x][i]]);
        dfs2(v[x][i],x);
    }
}

void func(){
    dfs1(1,1);
    dfs2(1,1);
    for (int i=1;i<=n;i++)
        d[i]=da[i]=p[i]=0,
        a[i]*=-1;
    dfs1(1,1);
    dfs2(1,1);
    cout<<ans<<endl;
}

int main() {
    cin>>n;
    for (int i=1;i<=n;i++)
        cin>>a[i];
```

```
    for (int i=1;i<n;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    func();
    return 0;
}
```

Explanation:

The idea is to split tree into two sub-tree and find biggest sum subtree in each. Here we calculate biggest sum subtree for directed edge(a->b).

The biggest sum for an edge means find subtree with maximum sum where node b is included and node a is not. We root tree and do this for the edges(parent->child) in one DFS. And for edges(child->parent) we need another DFS. In first DFS we are calculating max value in whole subtree to consider all subtrees in the process of calculating the answer. This process is done twice because we need

min not max, because product of two negative numbers may be greater than two positive numbers.

Time Complexity: O(n)

References:

1) https://www.topcoder.com/community/data-science/data-science-tutorials/disjoint-set-data-structures/
2) http://users.cecs.anu.edu.au/~Alistair.Rendell/Teaching/apac_comp3600/module3/disjoint_sets.xhtml