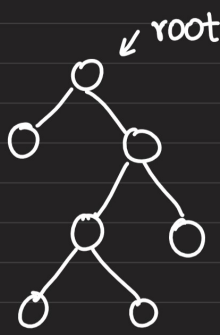
Tree - DP

- In out DP (v. common)
- Knapsack DP
- Ancenstral DP

Tree → unrooted
Tree → root



Q1. Tree → N nodes → rooted at x

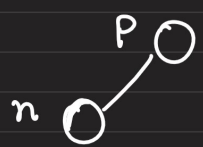
Find

a) Size of subtree of each node

```
vector<vector<lli>> g;
vector<lli> sub_sz;

void dfs(int nn, int pp, int dd)
{
    sub_sz[nn] = 1;
    for(auto v: g[nn]){
        if(v!=pp){
            dfs(v,nn,dd+1);
            sub_sz[nn] += sub_sz[v];
        }
    }
}

void solve()
{
    lli n,x;cin>>n>>x;
    g.resize(n+1); sub_sz.assign(n+1,0);
    for(int i=0;i<n-1;i++){
        int u,v; cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(x,0,0);
    for(int i=1;i<=n;i++) cout<<sub_sz[i]<<' ';
    cout<<'\n';
}
```

$$sz[node] = 1 + \sum_{child \in t[node]} sz[child]$$ } in-dp

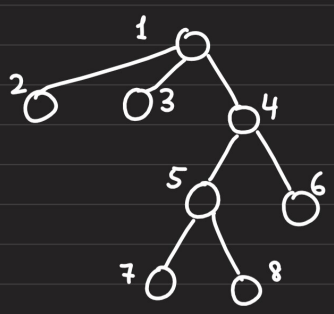
Edge contribution



no. of time this edge is used in simple paths from any one node to another node of the tree.

sub_sz[n] * (n - sub_sz[n]) | depth[n] > depth[p]

Q) Tree

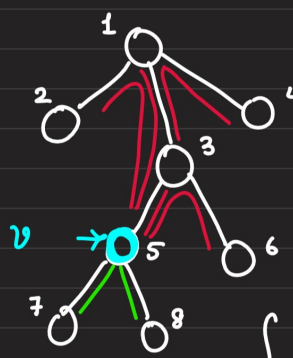


$$Ans[i] = \sum_{x=1}^{N} dist(i, x)$$

For every node, find its sum of distances to every other node

eg: Ans[4] = 1(3) + 2(4) = 11

Soln:



Ans[v] = 3 + 4 + 6 = 13

indp[v] → Green coloured = 2
outdp[v] → Red coloured = 11

ans[v] = indp[v] + outdp[v]

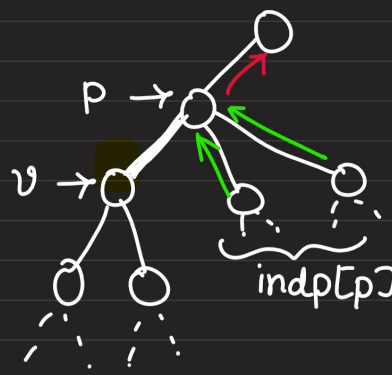
NOTE: Answer does not depend on the root of the tree. So, lets root it at 1.

Transition:

indp[a] = indp[b] + sz(b) +
indp[c] + sz(c) +
indp[d] + sz(d)

one additional path length added

{ Note: indp[leaf_node] = 0 }



outdp[root] = 0

redsum = outdp[p]

indp[p] - (indp[v] + sz(v)) } green sum

$$outDP[v] = outDP[p] + indp[p] - (indp[v] + sz(v)) + \underbrace{(N - sz(v))}_{\text{extra 1 length count}}$$

```cpp
lli n,x;
vector<vector<lli>> g;
vector<lli> sub_sz,indp,outdp;

void indfs(int nn, int pp)
{
    sub_sz[nn] = 1;
    for(auto v: g[nn]){
        if(v!=pp){
            indfs(v,nn);
            sub_sz[nn] += sub_sz[v];
            indp[nn] += (indp[v] + sub_sz[v]);
        }
    }
}

void outdfs(int nn, int pp)
{
    if(pp==0) outdp[nn] = 0;
    else outdp[nn] = outdp[pp] + (indp[pp] - (indp[nn]+sub_sz[nn])) + (n-sub_sz[nn]);
    for(auto v: g[nn])
    {
        if(v!=pp){
            outdfs(v,nn);
        }
    }
}

void solve()
{
    cin>>n;
    g.resize(n+1); sub_sz.assign(n+1,0); indp.assign(n+1,0); outdp.assign(n+1,0);
    for(int i=0;i<n-1;i++){
        int u,v; cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    indfs(1,0);
    outdfs(1,0);
    for(int i=1;i<=n;i++) cout<<indp[i]+outdp[i]<<' ';
    cout<<'\n';
}
```

Ans[i]

Note we need indp to calculate outdp. So, first calculate indp

$$outDP[v] = outDP[p] + indp[p] - (indp[v] + sz(v)) + (N - sz(v))$$

$$\Rightarrow outdp[v] + indp[v] = outdp[p] + indp[p] + N - 2sz(v)$$

$$\Rightarrow Ans[v] = Ans[p] + N - 2\,sz[v]$$

find Answer for root

↳ use it to find all other answers

You are given a tree consisting exactly of n vertices. Each vertex v of this tree has a value a_v assigned to it.

Let $dist(x,y)$ be the distance between the vertices x and y. The distance between the vertices is the number of edges on the simple path between them.

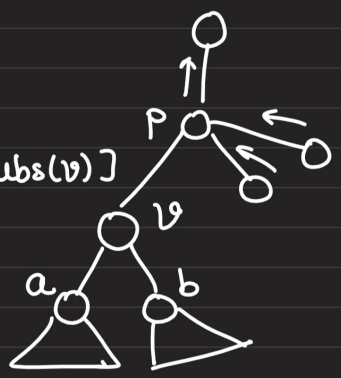
Let's define the cost of the tree as the following value: firstly, let's fix some vertex of the tree. Let it be v. Then the cost of the tree is

$$\sum_{i=1}^{n} dist(i, v) \cdot a_i.$$

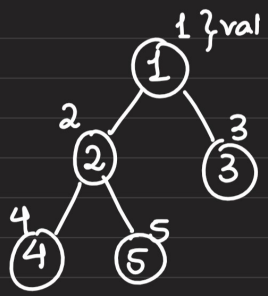
Your task is to calculate the **maximum as well as minimum possible cost** of the tree if you can choose v arbitrarily to calculate the answer for both the case.

$$indp[v] = \sum indp(child) + sub\text{-}sum(child)$$

$$outdp[v] = outdp(p) + indp(p) - [indp[v] + subs(v)] + (total\text{-}sum - sub\_sum(child))$$



Instead of subtree size, we have subtree sum



for 1

Ans [1] = 2 + 3 + 2(4) + 2(5) = 23

Ans [2] = 1 + 4 + 5 + 2(3) = 16

Ans [3] = 1 + 2(2) + 3(4) + 3(5) = 32

Ans [4] = 2 + 2 + 2(5) + 3(3) = 23

Ans [5] = 21

max = 32, min = 16

```cpp
lli n;
vector<vector<lli>> g;
vector<lli> val,sub_sum,indp,outdp;
lli total = 0;

void indfs(int nn, int pp)
{
    sub_sum[nn] = val[nn];
    for(auto v: g[nn]){
        if(v!=pp){
            indfs(v,nn);
            sub_sum[nn] += sub_sum[v];
            indp[nn] += (indp[v] + sub_sum[v]);
        }
    }
}

void outdfs(int nn, int pp)
{
    if(pp==0) outdp[nn] = 0; //for root
    else outdp[nn] = outdp[pp] + (indp[pp] - (indp[nn]+sub_sum[nn])) + (total-sub_sum[nn]);
    for(auto v: g[nn])
    {
        if(v!=pp){
            outdfs(v,nn);
        }
    }
}

void solve()
{
    total = 0;
    cin>>n;
    g.resize(n+1); val.assign(n+1,0); sub_sum.assign(n+1,0); indp.assign(n+1,0); outdp.assign(n+1,0);
    for(int i=1;i<=n;i++) {
        cin>>val[i];
        total += val[i];
    }
    for(int i=0;i<n-1;i++){
        int u,v; cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    indfs(1,0);
    outdfs(1,0);
    lli ma = -1, mi = INF;
    for(int i=1;i<=n;i++)
    {
        lli ans = indp[i]+outdp[i];
        ma = max(ma,ans);
        mi = min(mi,ans);
    }
    cout<<ma<<' '<<mi<<'\n';
    g.clear();
}
int main()
{
    int t; cin>>t; while(t--)
    solve();
}
```

You are given a tree consisting of n nodes.

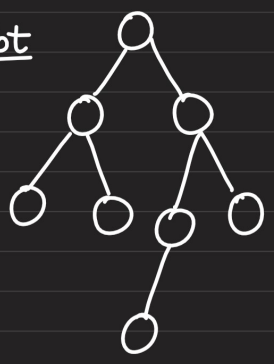
Your task is to determine for each node the maximum distance to another node.

```cpp
using lli = long long int;
using ii = pair<lli,lli>;
#define F first
#define S second

lli n,x;
vector<vector<lli>> g;
vector<lli> par,outdp;
vector<ii> indp; //max and second max indp

void indfs(int nn, int pp)
{
    indp[nn].F = 0;
    par[nn] = pp;
    for(auto v: g[nn]){
        if(v!=pp){
            indfs(v,nn);
            if(indp[v].F + 1 > indp[nn].F){
                indp[nn].S = max(indp[nn].S, indp[nn].F);
                indp[nn].F = indp[v].F + 1;
            }
            else{
                indp[nn].S = max(indp[nn].S, indp[v].F + 1);
            }
        }
    }
}
void outdfs(int nn, int pp)
{
    if(pp==0) outdp[nn] = 0; //for root
    else {
        outdp[nn] = 1 + outdp[pp];
        if(1 + indp[nn].F == indp[pp].F){ // if the max is in the same subtree as that of v
            outdp[nn] = max(outdp[nn],1 + indp[pp].S);
        }else{
            outdp[nn] = max(outdp[nn],1 + indp[pp].F);
        }
    }
    for(auto v: g[nn])
    {
        if(v!=pp){
            outdfs(v,nn);
        }
    }
}
void solve()
{
    cin>>n;
    g.resize(n+1); par.assign(n+1,0); indp.assign(n+1,{-1,-1}); outdp.assign(n+1,0);
    for(int i=0;i<n-1;i++){
        int u,v; cin>>u>>v;
        g[u].push_back(v); g[v].push_back(u);
    }
    indfs(1,0);
    outdfs(1,0);
    for(int i=1;i<=n;i++) cout<<max(indp[i].F,outdp[i])<<' ';
    cout<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    solve();
}
```

Solution 2

Compute a diameter of the tree as described by algorithm 2 here. Once we have a diameter (a, b), output $\max(dist(a, i), dist(b, i))$ for each node i.

Q2
Doubt



N nodes
Val[] → value of all nodes

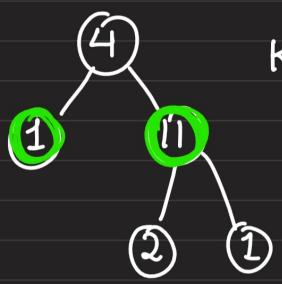
Select exactly K nodes such that

1) No 2 nodes are neighbour

2) $\sum_{j \in \text{selected } K} val[j]$ is maximum

Print that maximum value.

soln: knapsack - type



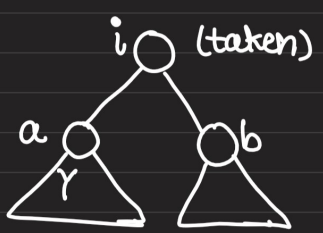
K = 2 } Ans = 12

NOTE:
Ans : does not depend on the root

dp (node, left, take)

We will always take the value of node if take = 1

$$Ans = \max \begin{pmatrix} Dp(1, K, 1), \\ Dp(1, K, 0) \end{pmatrix}$$



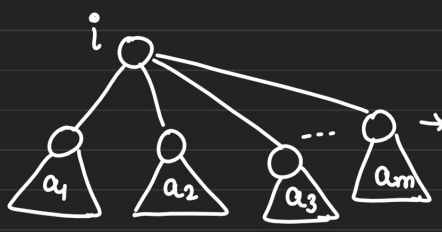
$$dp(i, X, 1) = \underbrace{dp(a, Y, 0) + dp(b, X-1-Y, 0) + val[i]}_{\text{take max along all Ys}}$$

$$dp(i, x, 0) = \max\left(dp(a, Y, 1), dp(a, Y, 0)\right) + \max\left(dp(b, x-Y, 1), dp(b, X-Y, 0)\right)$$

take max along all Ys

↑ node b taken

↑ node b not taken



$Dp(i, x, 0) = Dp(ch1, a_1, 0/1) + Dp(ch2, a_2, 0/1) + \cdots + Dp(ch_m, a_m, 0/1)$

which every is maximum

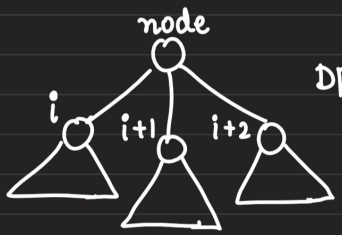
$\sum_{i=1}^{m} a_i = x$

Similar to form 1

for solving Dp, we created Dp2.

$dp2(0, X) \rightarrow dp2(1, X-1)$

$\rightarrow dp2(1, X-2)$



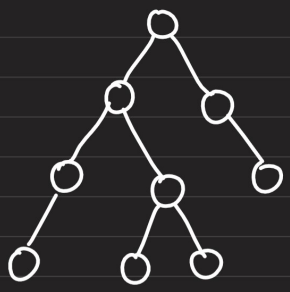
$Dp2(i, left) = \max(Dp2(i+1, left - c) + Dp(i, c, 0/1))$

$0 \le c \le sz(i)$

maximum sum we can get by taking c nodes from i.

$Dp(node, X, 0) = Dp2(0, X)$

Q3



$N \leq 10^5$

Find Ans[], where Ans[i] tells us the height if you the root is i.

You are given a tree consisting of n nodes.

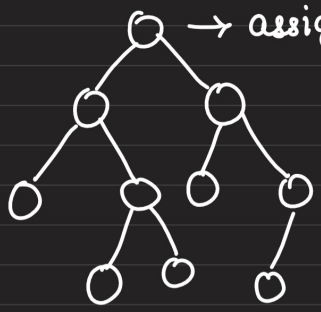
Your task is to determine for each node the maximum distance to another node.

Same problem.

Height with root i is the maximum distance to any other node

Q4 Alphonso Company

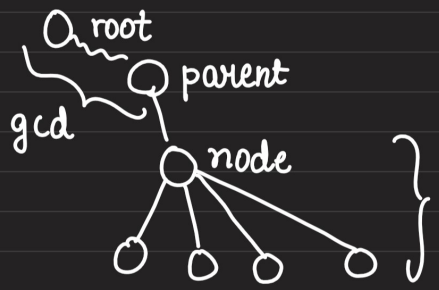
N nodes , M values [1 - M] } $N \leq 10^5$, $M \leq 20$



→ assign each node a value b/w 1-M s.t all paths from root to leaf has gcd == 1.

Count number of valid assignments.

soln:



make all paths from this node to leaf with gcd 1 given the gcd from root to parent

State

dp (node, gcd_till_now)

↳ gcd of path from root to par

↳ count valid assignments.

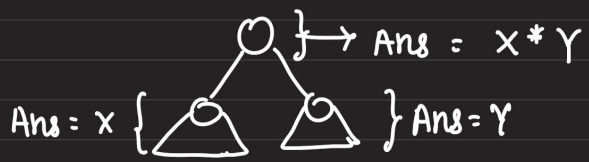
Very similar to form 1

Transitions

dp (node, G1)

↳ loop from [1-M]

↳ dp(child , gcd(chosen, G1))



$$dp(node, G) = \sum_{x=1}^{M} \left(\prod_{ch \in node} dp(ch, gcd(G, x)) \right)$$ } Transition

```cpp
lli n,m;
vector<vector<lli>> g;
vector<vector<lli>> dp;

int dfs(int nn, int pp, int gc)
{
    //Memoization
    if(dp[nn][gc]!=-1) return dp[nn][gc];

    int ans = 0;
    //Base Case
    if(g[nn].size()==1) //only parent is there, so leaf node
    {
        for(int col=1;col<=m;col++){
            if(__gcd(gc,col)==1) ans++;
        }
        return dp[nn][gc] = ans;
    }

    for(int col = 1; col<=m; col++)
    {
        int temp = 1;
        for(auto v: g[nn]){
            if(v!=pp)
            {
                int ans_child = dfs(v,nn,__gcd(gc,col));
                temp *= ans_child;
                temp%=mod;
            }
        }
        ans += temp;
        ans %= mod;
    }
    return dp[nn][gc] = ans;
}
```

```cpp
void solve()
{
    cin>>n>>m;
    g.resize(n+1);
    dp.assign(n+1,vector<lli>(21,-1));
    for(int i=1;i<n;i++)
    {
        int u,v; cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    cout<<dfs(1,0,0)<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    solve();
}
```

TC 1

Passed 20ms

Input: Copy

```
3 2
1 2
1 3
```

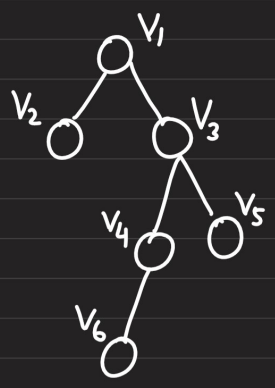
Expected Output: Copy

```
5
```

Received Output: Copy

```
5
```

Q5



N nodes
val[] → value of all nodes

Select some of the nodes such that

1) No 2 nodes are neighbour

2) $\sum_{j \in \text{selected K}} val[j]$ is maximum

Print that maximum value.

Soln: for array

$dp(i) = \max(arr[i] + dp(i+2),\ dp(i+1))$

} Simple 1D DP problem

dp(node, par_taken)
↓
0/1

} If par is taken

$dp(node, 1) = \sum_{child} dp(child, 0)$

If par is not taken

$$dp(node, 0) = \max\left\{ \sum_{child} dp(child, 0),\ val[node] + \sum_{child} dp(child, 1) \right\}$$

```cpp
lli n;
vector<vector<lli>> g,dp;
vector<lli> val;

lli dfs(int nn, int pp, int taken)
{
    //Memoization
    if(dp[nn][taken]!=-1) return dp[nn][taken];

    //Base Case
    if(g[nn].size()==1) //only parent is there, so leaf node
    {
        if(taken) return 0ll;
        else return max(val[nn],0ll);
    }
    lli ans = 0;
    lli temp1 = 0, temp2 = 0;
    for(auto v: g[nn])
    {
        if(v!=pp){
            temp1 += dfs(v,nn,0); //val[node] not taken
            temp2 += dfs(v,nn,1);
        }
    }
    ans = temp1;
    if(!taken) ans = max(ans,val[nn]+temp2);

    return dp[nn][taken] = ans;
}

void solve()
{
    cin>>n;
    g.resize(n+1); val.resize(n+1);
    dp.assign(n+1,vector<lli>(2,-1));

    for(int i=1;i<=n;i++) cin>>val[i];
    for(int i=1;i<n;i++)
    {
        int u,v; cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    cout<<dfs(1,0,0)<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    solve();
}
```

TC 1

Passed 26ms

Input: Copy

```
3
1 2 3
1 2
1 3
```

Expected Output: Copy

```
5
```

Received Output: Copy

```
5
```