GRAPHS

Day 6

(LCA, DP on Trees, Re-rooting technique)

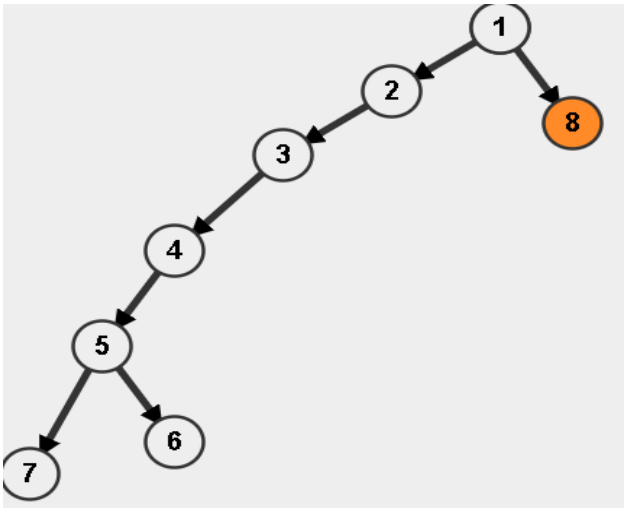
Youtube link :

https://youtu.be/t27lOCeAuUM

Contents:

LCA (Lowest Common Ancestor)

Problem link: SPOJ.com - Problem LCA



4th parent of 7..
already calc..
1.7->parents/ancestors..all values
2.7->(2^2)th parent??,,2^1,2^0

p=node;
loop..k
p=par[p]

O(n)/query..
kth-->binary representation..
k=3,11
k=5,101

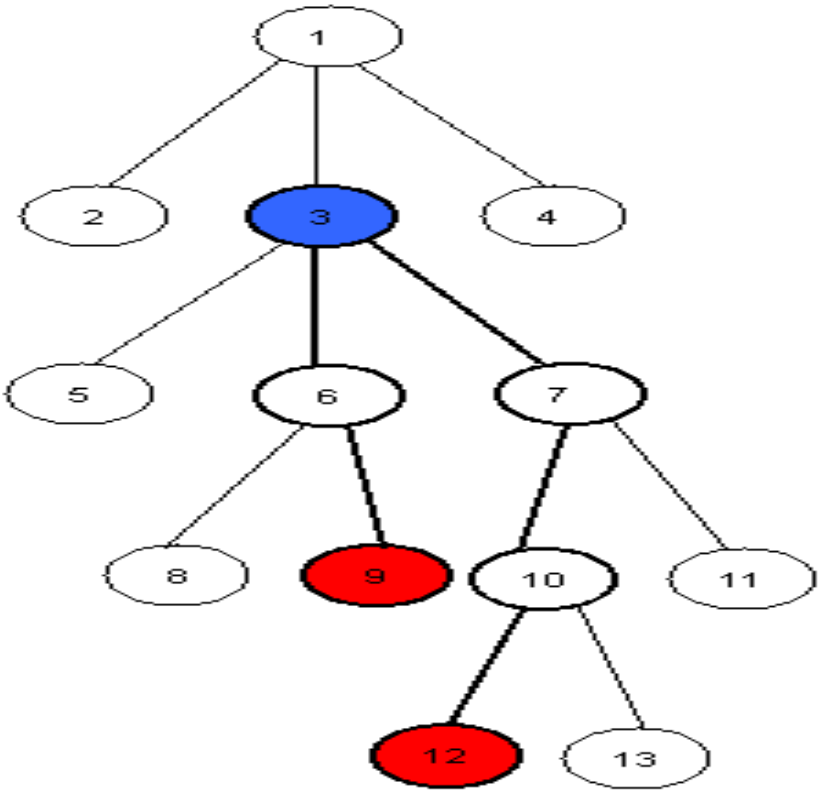
7th-->1st,2nd,4th…,2^x

```
int x=log(n)+1;
// max possible jump required
// to reach a parent..

vector<vector<int> > v; //adjacency list
int par[n][x];
// where par[i][j] ->(2^j)th parent of ith node
int depth[n]; //for depth of node calculation
depth[root(1)]=0;

void dfs(int s,int p)
{
 // s->source node
 // p->parent of s

 par[s][0]=p;
```

```
 for(int j=1;j<=x;j++)
  par[i][j]=par[par[i][j-1]][j-1];

for(int i=0;i<v[s].size();i++)
{
 int ch=v[s][i];
 if(ch!=p)
 {
  depth[ch]=depth[s]+1;
  dfs(ch,s);
 }
}
}

//finding kth parent..
int find_kth(int s,int k)
{
 for(int j=x;j>=0;j--)
 {
  if((1<<j)&k)
  {
   s=par[s][j];
   k-=(1<<j);
  }
 return s;
}
```

```
int lca(int a,int b)
{
  //depth[a]<=depth[b]
if(depth[a]>depth[b])//swap a and b..
{
 int tmp=a;
 a=b;
 b=tmp;
}
int d=depth[b]-depth[a];
b=find_kth(b,d);//dth parent of b

if(a==b)
 return a;

for(int j=x;j>=0;j--)
{
 if(par[a][j]!=par[b][j])
 {
   a=par[a][j];
   b=par[b][j];
 }
}

return par[a][0];
}
```

Time complexity: O(log n) per query

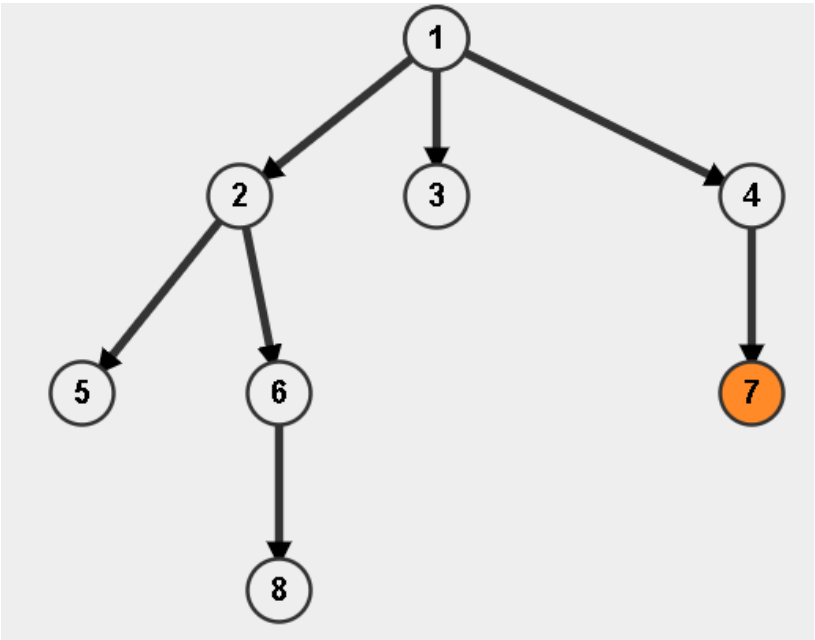


9->9,6,3,1
12->10,7,3,1

Find lca of node x and y ?
Brute Force → O(n) [Will give TLE for q queries]

Distance between two nodes in a tree (Q queries, $1<=Q<=2x10^5$)

Problem link: https://cses.fi/problemset/task/1135
(CSES - Distance Queries)



1-->root..
dist b/w 5 and 8..
l=dist b/w 1 and 2==depth[2]
x=dist b/w 1 and 5==depth[5]
y=dist b/w 1 and 8==depth[8]

x+y=depth[5]+depth[8]-(2*l)//dist b/w 5 and 8

dist(a,b) = depth[a] + depth[b] - 2 * depth[lca(a,b)]

Another Practice Problem on LCA:
CSES - Company Queries II

DP on trees

Q1. Atcoder DP contest - Problem P:
Link: P - Independent Set (atcoder.jp)

tree→ paint with 0 and 1

n nodes

```
const int md=1e9+7;
#define int long long

int dp[n][3]; // initialise all with -1 in main()
dfs(int cur node(s),int prev(0 or 1),int par)
{
 if(dp[n][prev]!=-1)//already calc..
   return dp[n][prev];
int resw=1;//ways to paint children when col of n 0
int resb=1;//ways to paint children when col of n 1

for(int i=0;i<v[s].size();i++)
{
 int ch=v[s][i];
 if(ch!=par)
 {
  resw=(resw%md*dfs(ch,0,s)%md)%md;
  resb=resb*dfs(ch,1,s);
 }
}

if(prev==1) //col of parent of s is ==1
  resb=0;

return dp[s][prev]=(resb+resw);
}
```

```
int32_t main()
{
 int res=dfs(1,2,1);
}
```

CSES - Tree Matching

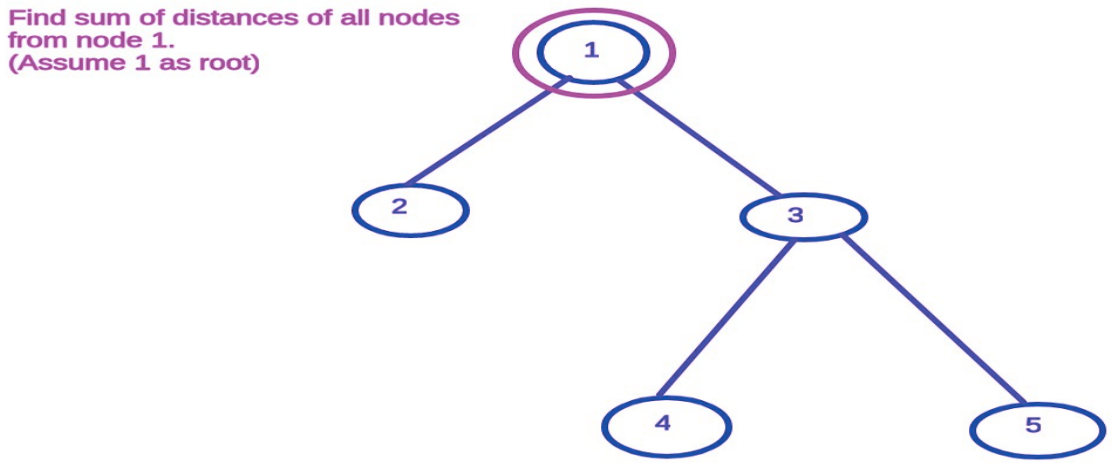
```
dp[n][prev/(0 or 1)]==-1;

int dfs(int n,int prev,int par)
{
 if(dp[n][prev]!=-1)
 return dp[n][prev];

int sum=0;
int res=0;
for(int i=0;i<v[s].size();i++)
{
 int ch=v[s][i];
 if(ch!=par)
 {
  sum+=dfs(ch,0,s);
 }
}
res=sum;
```

```
if(prev!=1)//not picked by parent..
{
for(int i=0;i<v[s].size();i++)
{
 int ch=v[s][i];
 if(ch!=par)
 {
   int tmp=dfs(ch,1,s)+1;
   int tmp2=dfs(ch,0,s);
   res=max(res,sum-tmp2+tmp);
}
return dp[s][prev]=res;
}
```

Rerooting Technique

Q. Given a tree of n nodes, rooted at node 1, find the sum of distances of all nodes from node 1.

Example:



Ans = 1 + 1 + 2 + 2
Let dp[i] = sum of distances from node i to all the nodes in the subtree of node i

dp[node] = dp[child] + cnt[child]
Where cnt[i]= no. of nodes in subtree of node i

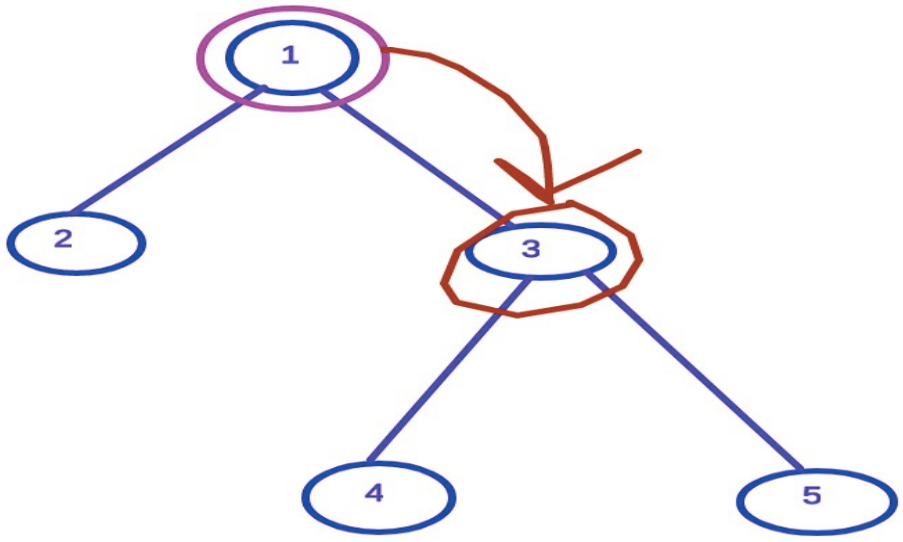
dp[5] =0
dp[3]=1+1=2

```
vector<vector<int> > adj;
vector<int> cnt; // cnt[i] = no. of nodes in subtree of i
vector<int> dp; // dp[i] = sum of distances of all the nodes in subtree of node i from node i
void dfs(int node, int par)
{
  cnt[node]=1;
  dp[node]=0;
  for(auto it: adj[node])
  {
     if(it==par)
        continue;
     dfs(it,node);
     cnt[node]+=cnt[it];
     dp[node]+=dp[child]+cnt[child];
  }
}
// dfs(0,-1) in main()
```

Q 1. Given a tree of n nodes. Your task is to determine for each node the sum of the distances from the node to all other nodes.
Link: https://cses.fi/problemset/task/1133

One approach is simply run dfs() n times (for all n nodes) and solve as above for each root.

Time complexity: $O(n^2)$
We can reduce this overall time complexity to O(n) using re-rooting.
First assume any one node as root and calculate all dp values.
Then, make all the nodes root, one by one, as follows:



When we transfer root from node 1 to node 3 in above graph example, new dp value are calculated as:

dp[1]=dp[1]-dp[3]; // remove contribution of node 3
dp[3]=dp[3]+dp[1]; // add contribution of node 1

In general:
Consider a case in which "**node**" is the current root and "**ch**" is one of it's child, so it would had contributed something in the answer of node. So now we try to make **ch** as the **new root**, so :

1. **First we remove the contribution of ch from node**
2. **Then, node is now a child for ch, so we add the contribution of the subtree with root being node to the answer of ch and now ch has become the new root of the tree.**

But when we need to calculate the answer for say ch2(second child of node), we have to rollback the changes we made (restore the old values of dp).

Thus, we can transfer the root from one node to its child in just O(1).

General Pseduo code used to solve re-rooting problems:

```
void dfs2(int node, int par)
{
```

```
 int t1=dp[node];
  for(auto it: adj[node])
  {
     if(it==par)
        continue;
    int t2=dp[it];
   // transfer root from node to it

dfs(it,node);
// rollback root to node
// (restore old values)
   dp[node]=t1;
   dp[it]=t2;

  }
}
```

Solution of Tree Distances II:

```
vector<vector<int> > adj;
vector<int> cnt; // cnt[i] = no. of nodes in subtree
of i
vector<int> dp; // dp[i] = sum of distances of all
the nodes in subtree of node i from node i
vector<int> ans; // ans[i]= sum of distances of all
nodes from node i when i is a root
void dfs1(int node, int par)
```

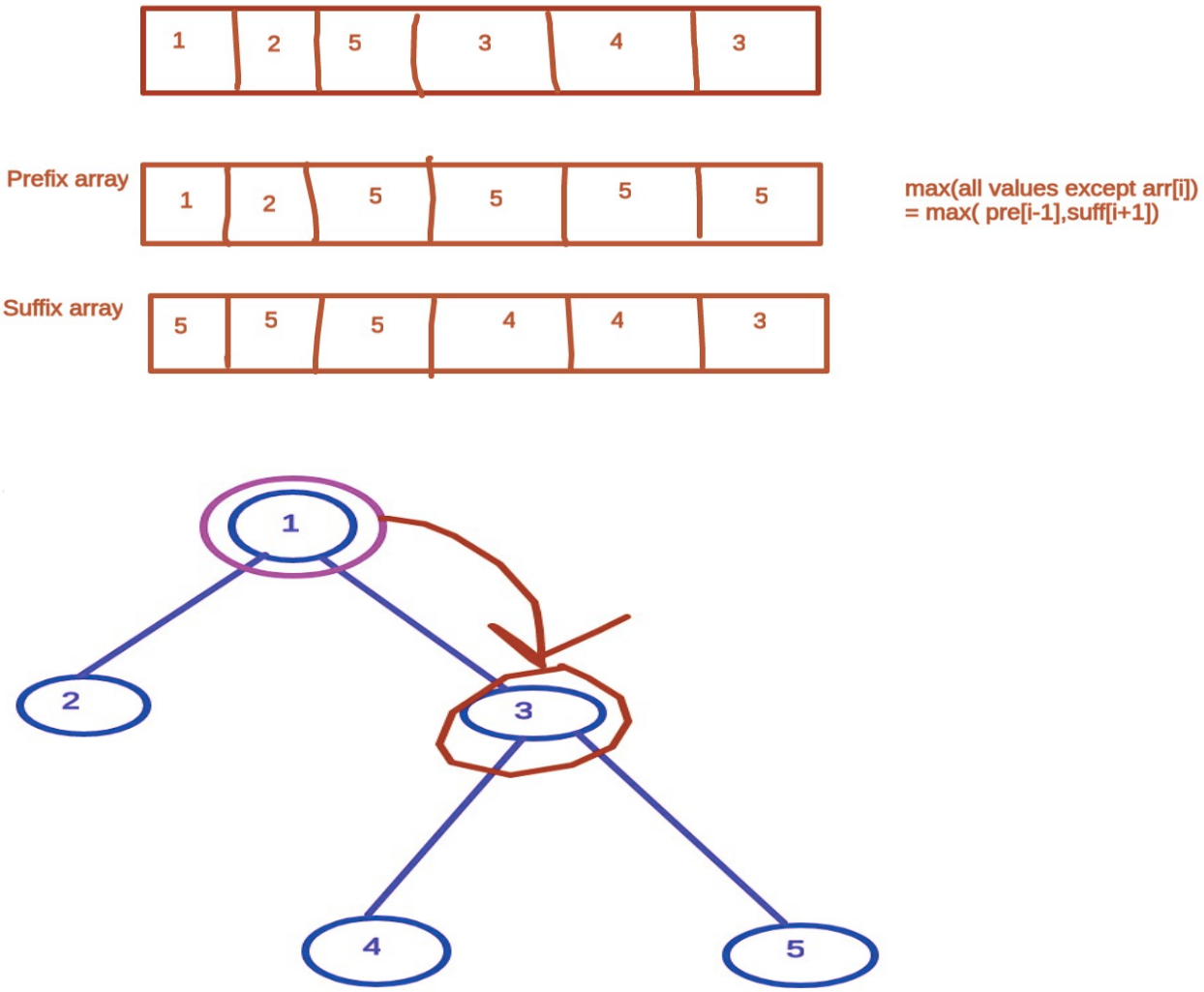
```
{
  cnt[node]=1;
  dp[node]=0;
  for(auto it: adj[node])
  {
     if(it==par)
        continue;
     dfs1(it,node);
     cnt[node]+=cnt[it];
     dp[node]+=dp[child]+cnt[child];
  }
}



int dfs2(int node, int par)
{
ans[node]=dp[node];
int s1=cnt[node];
int t1=dp[node];
   for(auto it: adj[node])
 {
    if(it==par)
      Continue;
int s2=cnt[it];
int t2=dp[it];
// Transfering root from node to it
```

```
    cnt[node]-=cnt[it];
    dp[node]-=(dp[it]+cnt[it]);
    cnt[it]+=cnt[node];
    dp[it]+=dp[node]+cnt[node];

   dfs(it,node);

   // rollback the root to node
   cnt[node]=s1;
   dp[node]=t1;
   cnt[it]=s2;
   dp[it]=t2;
  }
}

// dfs1(0,-1);
// dfs2(0,-1);
```

Q. 2

https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/practice-problems/algorithm/parwal-problem/description/

Solution Approach:



Solution:

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

int n;
vector<vector<int>> adj;
```

```cpp
vector<int> w;
vector<int> dp;
vector<int> ans;

// dp[i] = maximum no. of treats i can get, from
the subtree of node i, if I start from node i

void dfs1(int node, int par)
{
  dp[node]=0;
  for(auto ch: adj[node])
  {
    if(ch==par)
        continue;
    dfs1(ch,node);
    dp[node]=max(dp[node],dp[ch]);
  }
  dp[node]+=w[node];
}

void dfs2(int node, int par)
{
  ans[node]=dp[node];
  int t1=dp[node];
  vector<int> pre,suff;
  // Prefix Maximum and Suffix Maximum array

  for(auto ch: adj[node])
```

```
 {
   if(ch==par)
     continue;
   pre.push_back(dp[ch]) ;
   suff.push_back(dp[ch]) ;
 }

 int sz=pre.size();
 for(int i=1; i<sz; i++)
 {
   pre[i]=max(pre[i],pre[i-1]);
 }
 for(int i=sz-2; i>=0; i--)
 {
   suff[i]=max(suff[i],suff[i+1]);
 }

 int i=0;
 for(auto ch: adj[node])
 {
   if(ch==par)
       continue;
   int t2=dp[ch];


// re-rooting from node to ch
   int l=0, r=0;
   if(i-1>=0)
```

```
      l=pre[i-1];
    if(i+1<sz)
     r=suff[i+1];

    dp[node]=max(l,r)+w[node];
    if(par!=-1)
    {
      dp[node]=max(dp[node], dp[par]+w[node]);
    }
    dp[ch]=max(dp[ch], dp[node]+w[ch]);

    dfs2(ch,node);

    // restoring dp values

    dp[node]=t1;
    dp[ch]=t2;

    i++;
  }

}

int32_t main() {
  int t;
  cin >> t;
  while (t--) {
    cin >> n;
```

```
    w.resize(n);
    for (int i = 0; i < n; i++) {
      cin >> w[i];
    }
    adj.clear();
    adj.resize(n);
    int u, v;
    for (int i = 0; i < n - 1; i++) {
      cin >> u >> v;
      u--;
      v--;
      adj[u].push_back(v);
      adj[v].push_back(u);
    }
    dp.resize(n);
    ans.resize(n);
    dfs1(0,-1); // assuming node 1 as root
    dfs2(0,-1); // re-root to all the nodes, one by
one
    for(int i=0; i<n; i++)
    {
      cout<<ans[i]<<' ';
    }
    cout<<'\n';
  }
  return 0;
}
```

Practice Problems :

1. https://atcoder.jp/contests/dp/tasks/dp_v
2. https://cses.fi/problemset/task/1132

Advanced Level Problem: (Optional)

1. https://cses.fi/problemset/task/1752