Tree Algorithms

Rishabh Dhiman

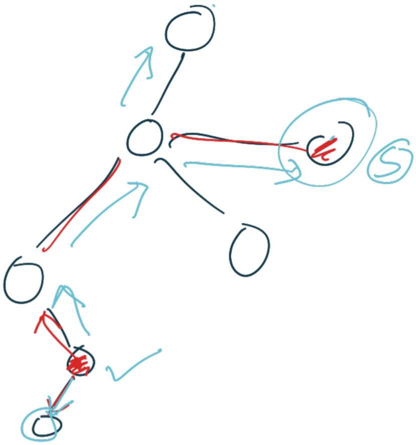
Algorithms and Coding Club
Indian Institute of Technology Delhi

6 December 2021

Distance Queries

Problem

Given a tree T, and two vertices r and s, find the distance between r and s.



Distance Queries

Problem

Given a tree T, and two vertices r and s, find the distance between r and s.

1. Start a DFS at r.
2. Keep track of the current depth.
3. Return this as answer once you reach s.

Distance Queries

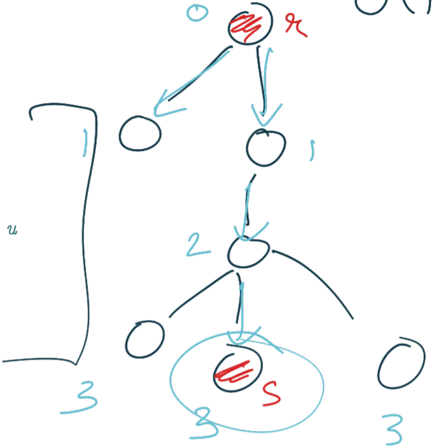
Problem

Given a tree T, and two vertices r and s, find the distance between r and s.

1. Start a DFS at r.
2. Keep track of the current depth.
3. Return this as answer once you reach s.

$O(|V|)$

```
int ans;
void dfs(int u, int depth, int p) {
    if (u == s) {
        ans = depth;
    }
    for (int v : g[u]) { // g[u] stores the neighbours of u
        if (v == p)
            continue;
        dfs(v, depth + 1, u);
    }
}
dfs(r, 0, r);
```



Distance Queries

Problem

Given a tree T rooted at r, answer Q queries. In a query, a vertex s is given, find the distance between r and s.

Distance Queries

Problem

Given a tree T rooted at r, answer Q queries. In a query, a vertex s is given, find the distance between r and s.

1. Start a DFS at r.
2. Keep track of the current depth.
3. Store this depth for each node.
4. Return this stored depth in each query.

Distance Queries

Problem

Given a tree T rooted at r, answer Q queries. In a query, a vertex s is given, find the distance between r and s.

1. Start a DFS at r.
2. Keep track of the current depth.
3. Store this depth for each node.
4. Return this stored depth in each query.

```
int d[N];
void dfs(int u, int depth, int p) {
    d[u] = depth;
    for (int v : g[u]) {
        if (v == p)
            continue;
        dfs(v, depth + 1, u);
    }
}
dfs(r, 0, r);

for (int i = 0; i < q; ++i) {
    int s; cin >> s;
    cout << d[s] << '\n';
}
```

fixed r
and varying s

$O(|V| + Q)$

Distance Queries

Problem

Given a tree T, answer Q queries. In a query, vertices s and t are given, find the distance between s and t.

Q $\quad s \longleftrightarrow t$

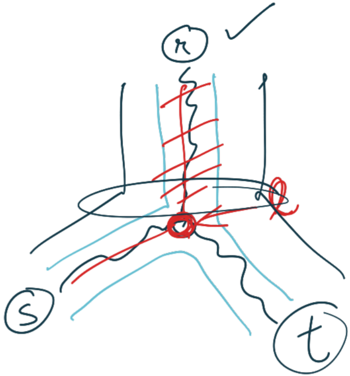
$O(Q|V|)$

$O(|V|^2 + Q)$

$O((|V| + Q) \log |V|)$ ✓

$O(|V| \log |V| + Q)$

r



$d[u]$

$l = LCA(s,t)$

Lowest common ancestor

$$\begin{aligned} dist(s,t) &= d(s,l) + d(l,t) \\ &= d(s,r) - d(l,r) + d(t,r) - d(r,l) \\ &= d(r,s) + d(r,t) - 2\, d(r,l) \end{aligned}$$

Distance Queries

Problem

Given a tree T, answer Q queries. In a query, vertices s and t are given, find the distance between s and t.

1. Arbitrarily root the tree at some vertex r.
2. Compute the distance from r as in the previous case.
3. Output $d(r, s) + d(r, t) - 2d(r, lca(s, t))$.

Distance Queries

Problem

Given a tree T, answer Q queries. In a query, vertices s and t are given, find the distance between s and t.

1. Arbitrarily root the tree at some vertex r.
2. Compute the distance from r as in the previous case.
3. Output $d(r,s) + d(r,t) - 2d(r, lca(s,t))$.

$O(|V| \log |V| + Q \log |V|)$

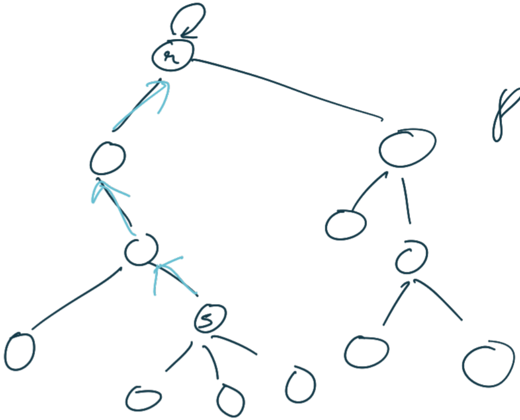
```
dfs(0, 0, 0);
for (int i = 0; i < q; ++i) {
    int s, t; cin >> s >> t;
    cout << d[s] + d[t] - 2 * d[lca(s, t)] << '\n';
}
```

$\log N$

Ancestor Queries

Problem

Given a tree rooted at r, and a vertex s, find the k-th ancestor of s.



parent of r = r

Ancestor Queries

Problem

Given a tree rooted at r, and a vertex s, find the k-th ancestor of s.

1. Start at a DFS at r.
2. For each vertex store its parent.
3. Find the answer by finding the parent of the node k times.

Ancestor Queries

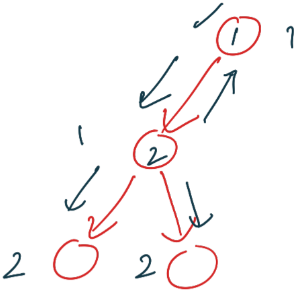
Problem

Given a tree rooted at r, and a vertex s, find the k-th ancestor of s.

1. Start at a DFS at r.
2. For each vertex store its parent.
3. Find the answer by finding the parent of the node k times.

```
int p[N];
void dfs(int u) {
    for (int v : g[u]) {
        if (v == p[u])
            continue;
        p[v] = u;
        dfs(v);
    }
}
p[r] = r;
dfs(r);

for (int i = 0; i < k; ++i) {
    s = p[s];
}
cout << s;
```



$k < |V|$

$|V| - 1$

$s \to p(s)$

$O(|V| + k)$

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, number m is given, find the 2^m-th ancestor of all vertices.

r

2^m-th ancestor of all vertices

$O(Q|V|)$

$O(Q|V|2^m)$

$\Downarrow$

$O(Q|V|m)$

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, number m is given, find the 2^m-th ancestor of all vertices.

The parent relation is a function $p : V \to V$. The query is to find $\underbrace{p \circ p \circ p \circ \cdots \circ p}_{2^m \text{ times}} = p^{2^m}$.

Composition of functions is an associative binary operation.

$p : V \to V$

p^{2^m}

$p^{2^m}(v)$ for all v

p

$\underbrace{p \circ p \circ p \circ \cdots \circ p}_{2^m}$

f, g, h

$(f \circ g) \circ h = f \circ (g \circ h)$

$$\begin{aligned}((f \circ g) \circ h)(x) &= (f \circ g)(h(x)) \\ &= f(g(h(x))) \\ &= f((g \circ h)(x)) \\ &= (f \circ (g \circ h))(x)\end{aligned}$$

$\underline{a}^n$ $\underline{\text{in } \log n}$

A^n for a matrix A

f^n

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, number m is given, find the 2^m-th ancestor of all vertices.

The parent relation is a function $p : V \to V$. The query is to find $\underbrace{p \circ p \circ p \circ \cdots \circ p}_{2^m \text{ times}} = p^{2^m}$.

Composition of functions is an associative binary operation. We can use binary exponentiation!

$$p^{2^m} = p^{2^{m-1}} \circ p^{2^{m-1}}.$$

$$p^{2^m} = p^{2^{m-1}+2^{m-1}} = p^{2^{m-1}} \circ p^{2^{m-1}}$$

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, number m is given, find the 2^m-th ancestor of all vertices.

The parent relation is a function $p : V \to V$. The query is to find $\underbrace{p \circ p \circ p \circ \cdots \circ p}_{2^m \text{ times}} = p^{2^m}$.

Composition of functions is an associative binary operation. We can use binary exponentiation!

$$\underline{p^{2^m} = p^{2^{m-1}} \circ p^{2^{m-1}}}.$$

```
int f[M][N]; // f[m] is the desired answer
p[r] = r; dfs(r);   3
for (int u = 0; u < n; ++u) {
    f[0][u] = p[u];
}
for (int j = 0; j < M - 1; ++j) {
    for (int u = 0; u < n; ++u) {
        f[j + 1][u] = f[j][f[j][u]];
    }
}
```

$f[j] \to p^{2^j}$

$f[j][u] = p^{2^j}(u)$

$f[0][u] = p(u) = p^{2^0}(u)$

$p^{2^{j+1}}(u) = f[j+1][u] = f[j][f[j][u]] = p^{2^j}(p^{2^j}(u))$

$O(|V|)$

$O(|V| m)$ $f[m]$

$O(m|V| + Q|V|)$

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, a vertex s and a number k is given, find the k-th ancestor of s.

$p^k(s)$

$O(m|V|)$

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, a vertex s and a number k is given, find the k-th ancestor of s.

Use binary exponentiation. Let $k = \sum_{i=0}^{m-1} b_i 2^i$,

$$p^k(s) = (p^{b_{m-1}2^{m-1}} \circ \cdots \circ p^{b_2 2^2} \circ p^{b_1 2^1} \circ p^{b_0 2^0})(s).$$

$k = 5 = (101)_2$

$p^k(s) = p^4(p(s)) = f[2][f[0][s]]$

$2^m \approx k < |V| \Rightarrow m < \log_2 |V|$

$O(m|V| + Qm) = O((|V| + Q) \log |V|)$

Ancestor Queries

Problem

Given a tree rooted at r, answer Q queries. In each query, a vertex s and a number k is given, find the k-th ancestor of s.

Use binary exponentiation. Let $k = \sum_{i=0}^{m-1} b_i 2^i$,

$$p^k(s) = (p^{b_{m-1}2^{m-1}} \circ \cdots \circ p^{b_2 2^2} \circ p^{b_1 2^1} \circ p^{b_0 2^0})(s).$$

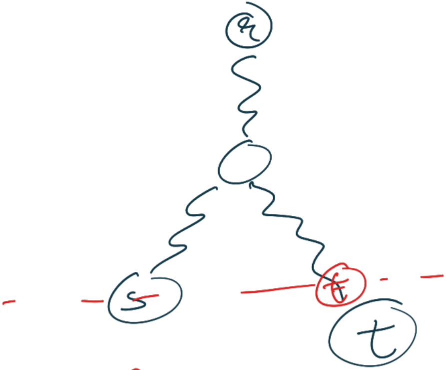
```
int f[M][N]; // compute it as in the previous case
for (int j = 0; j < M; ++j) {
    if (k >> j & 1)
        s = f[j][s];
}
```

$s \to p^{2^j}(s) = f[j][s]$

Ancestor Queries

Problem (LCA)

Given a tree rooted at r, answer Q queries. In each query, vertices s and t are given, find their lowest common ancestor.



t is below s

$t \to p^{d(t)-d(s)}(t)$

now s and t are at the same depth

$\boxed{k = l}$

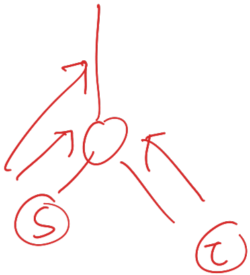
$u \to p^k(s)$ for some k
and $p^l(t)$ for some l

For vertices at the same depth

LCA$(s,t) = p^k(s)$ where k is the min value at which $p^k(s) = p^k(t)$

$Q(k)$ $\quad p^k(s) = p^k(t)$

$\qquad p^{k+1}(s) = p^{k+1}(t)$



$$Q(k) \Rightarrow Q(k+1)$$

Ancestor Queries

Problem (LCA)

Given a tree rooted at r, answer Q queries. In each query, vertices s and t are given, find their lowest common ancestor.

1. Move up s or t such that they both are at the same depth.
2. Binary search to find the smallest k such that $p^k(s) = p^k(t)$.
3. This $p^k(s)$ is the LCA.

Ancestor Queries

Problem (LCA)

Given a tree rooted at r, answer Q queries. In each query, vertices s and t are given, find their lowest common ancestor.

1. Move up s or t such that they both are at the same depth. ✓
2. Binary search to find the smallest k such that $p^k(s) = p^k(t)$.
3. This $p^k(s)$ is the LCA. ✓

$O(\log N)$

```
int f[M][N], d[N]; // compute it as earlier
int ancestor(int s, int k); // returns the k-th ancestor of s
int lca(int s, int t) {
    if (d[s] > d[t]) s = ancestor(s, d[s] - d[t]);
    else t = ancestor(t, d[t] - d[s]);

    int l = -1, r = n;
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (ancestor(s, m) == ancestor(t, m)) r = m;
        else l = m;
    }
    return ancestor(s, r);
}
```

1 $\leftarrow O(\log N)$

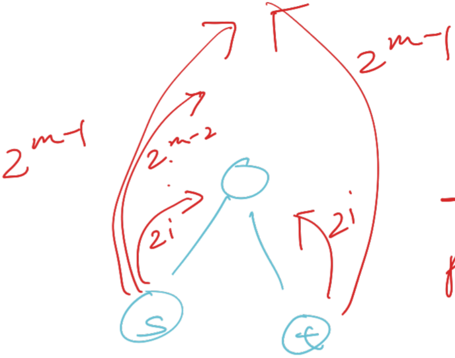
2 $O(\log N \times 2 \log N) = O(\log^2 N)$

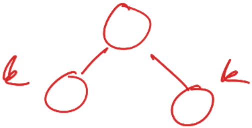
3 $\leftarrow O(\log N)$

Ancestor Queries
Binary Lifting

1. Move up s or t such that they both are at the same depth.
2. If both are now equal, this is the LCA. $s = t$
3. Binary search to find the largest k such that $p^k(s) \neq p^k(t)$.
4. Then $p^{k+1}(s)$ is the LCA.

s





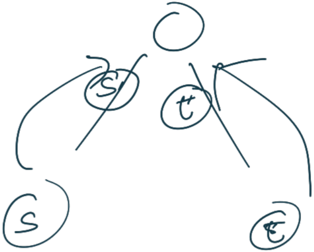
$p^{k}(s) \neq p^{k}(t)$

$p^{k+1}(s) = p^{k+1}(t)$

$p^{2^i}(s) \neq p^{2^i}(t)$ and i is the ~~smallest~~ largest value

is the

such that



$s \to p^{2^i}(s)$

$t \to p^{2^i}(t)$

i will decrease strictly

Ancestor Queries
Binary Lifting

1. Move up s or t such that they both are at the same depth.
2. If both are now equal, this is the LCA.
3. Binary search to find the largest k such that $p^k(s) \neq p^k(t)$.
4. Then $p^{k+1}(s)$ is the LCA.

$O(\log N)$

```
int f[M][N], d[N]; // compute it as earlier
int ancestor(int s, int k); // returns the k-th ancestor of s
int lca(int s, int t) {
    if (d[s] > d[t]) s = ancestor(s, d[s] - d[t]);   ]1
    else t = ancestor(t, d[t] - d[s]);

    if (s == t) return s;   ]2

    for (int j = M - 1; j >= 0; --j) {
        if (f[j][s] != f[j][t]) {
            s = f[j][s];
            t = f[j][t];
        }
    }   ]3
    return f[0][s];   ]4
}
```

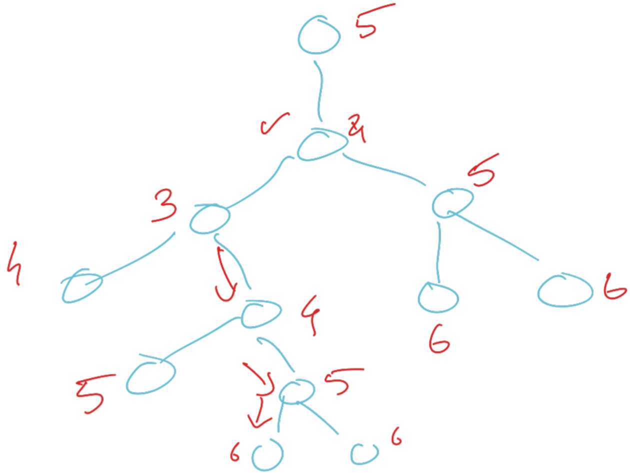
$p^{2^j}(s) \neq p^{2^j}(t)$

move up s and t by 2^j

Tree DP

Problem

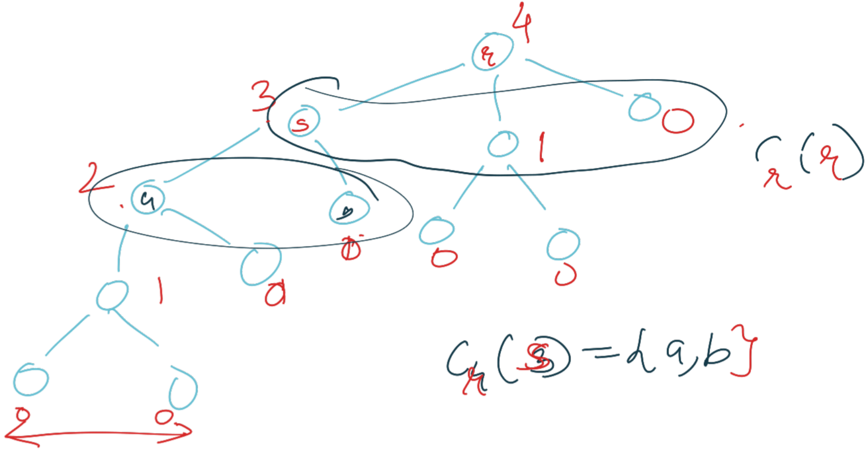
Given a tree T, find the longest path starting at u for all vertices u.



Tree DP

Problem

Given a tree T rooted at r, find the longest path starting at u in the subtree of u.



Tree DP

Problem

Given a tree T rooted at r, find the longest path starting at u in the subtree of u.

$$f_r(u) \stackrel{\text{def}}{=} \text{longest path starting at } u \text{ in the subtree of } u \text{ if we root at } r\text{, and}$$

$$C_r(u) \stackrel{\text{def}}{=} \text{set of children of } u \text{ if we root the tree at } r.$$

Tree DP

Problem

Given a tree T rooted at r, find the longest path starting at u in the subtree of u.

$$f_r(u) \stackrel{\text{def}}{=} \text{longest path starting at } u \text{ in the subtree of } u \text{ if we root at } r \text{, and}$$

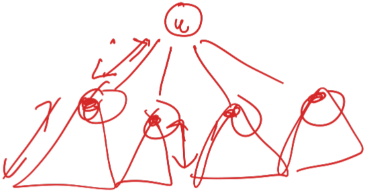
$$C_r(u) \stackrel{\text{def}}{=} \text{set of children of } u \text{ if we root the tree at } r.$$

We get the relation,

$$f_r(u) = \max_{v \in C_r(u)} (1 + f_r(v)),$$

with $f_r(v) = 0$ for a leaf v.

```
int f[N];
void dfs(int u, int p) {
    f[u] = 0;
    for (int v : g[u]) {
        if (v == p)
            continue;
        dfs(v, u);
        f[u] = max(f[u], 1 + f[v]);
    }
}
dfs(r, r);
```



1+

After dfs(u,p) ends

$$f[u] = f_r(u)$$

Tree DP

Problem

Given a tree T and a vertex u, find the longest path starting at u.

Let $h(u)$ = longest path starting at u, we see that

$$h(u) = f_u(u).$$

$d(u)$

Tree DP
Rerooting

Problem

Given a tree T, find the longest path starting at u for all vertices u.

$$O(|V|^2) \Rightarrow O(|V|)$$

f_v ~~&~~ and f_u are very similar

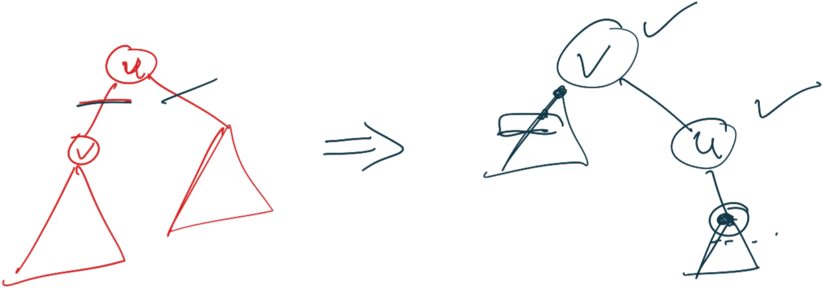
Tree DP
Rerooting

Problem

Given a tree T, find the longest path starting at u for all vertices u.

For $v \in C_u(u)$, f_v and f_u are almost the same, for all $x \notin \{u, v\}$,

$$f_v(x) = f_u(x).$$



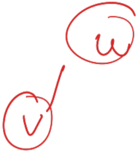
Tree DP
Rerooting

Problem

Given a tree T, find the longest path starting at u for all vertices u.

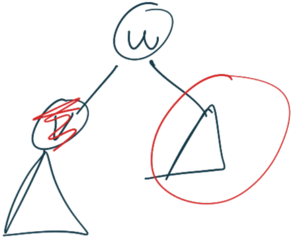
For $v \in C_u(u)$, f_v and f_u are almost the same, for all $x \notin \{u, v\}$,

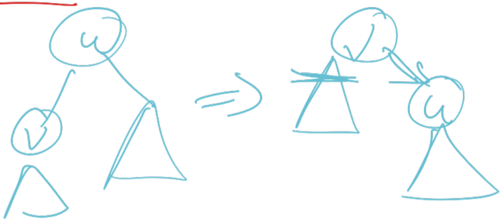
$$f_v(x) = f_u(x).$$



This happens because C_v and C_u are almost the same,

$$\begin{aligned} C_v(u) &= C_u(x) \text{ for } u \notin \{u, v\}, \\ C_v(u) &= C_u(u) \setminus \{v\}, \\ C_v(v) &= C_u(v) \cup \{u\}. \end{aligned}$$





$$
\begin{aligned}
f_v(x) &= f_u(x) \text{ for } x \notin \{u, v\},\\
C_v(u) &= C_u(u) \setminus \{v\},\\
C_v(v) &= C_u(v) \cup \{u\}.
\end{aligned}
$$

We only need to recompute $f_v(v)$ and $f_v(u)$,

$$
\begin{aligned}
f_v(u) &= \max_{x \in C_v(u)} (1 + f_v(x))\\
&= \max_{x \in C_u(u) \setminus \{v\}} (1 + f_v(x))\\
&= \max_{x \in C_u(u) \setminus \{v\}} (1 + f_u(x)).
\end{aligned}
$$

For each v recompute $f_v(u)$ $\quad O(\deg u \times \deg u)$

$\Rightarrow O(\deg^2 u) \rightarrow O((n-1)^2) = O(n^2)$

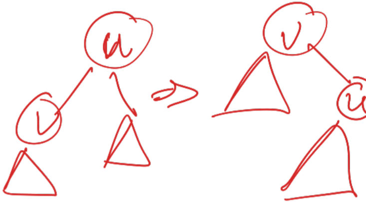
Tree DP
Rerooting

$f_u(x)$ largest

$$f_v(x) = f_u(x) \text{ for } x \notin \{u, v\},$$
$$C_v(u) = C_u(u) \setminus \{v\},$$
$$C_v(v) = C_u(v) \cup \{u\}.$$

We only need to recompute $f_v(v)$ and $f_v(u)$,

$$\begin{aligned} f_v(u) &= \max_{x \in C_v(u)} (1 + f_v(x)) \\ &= \max_{x \in C_u(u) \setminus \{v\}} (1 + f_v(x)) \\ &= \max_{x \in C_u(u) \setminus \{v\}} (1 + f_u(x)). \end{aligned}$$



We consider two cases, if $v = \operatorname{argmax}_{x \in C_u(u)} f_u(x)$ or not, that is if $f_u(x)$ is maximized at v or not. (If there are multiple values at which it is maximized, we arbitrarily pick one to be the argmax.)

1. If it is not the argmax,

$$f_v(u) = \max_{x \in C_u(u) \setminus \{v\}} (1 + f_u(x)) = \max_{x \in C_u(u)} (1 + f_u(x)) = f_u(u).$$

$O(\deg v + O(1))$

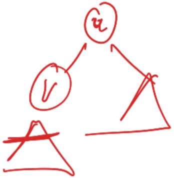
2. If it is the argmax, this happens only once, so we can just recompute $f_v(u)$.

$\to 1 \to O(\deg v)$

Tree DP
Rerooting

$f_u(v)$

$$
\begin{aligned}
f_v(x) &= f_u(x) \text{ for } x \notin \{u, v\}, \\
C_v(u) &= C_u(u) \setminus \{v\}, \\
C_v(v) &= C_u(v) \cup \{u\}.
\end{aligned}
$$



We only need to recompute $f_v(v)$ and $f_v(u)$,

$f_v(v)$

$$
\begin{aligned}
f_v(v) &= \max_{x \in C_v(v)} (1 + f_v(x)) \\
&= \max_{x \in C_u(v) \cup \{u\}} (1 + f_v(x)) \\
&= \max(1 + f_v(u), \max_{x \in C_u(v)} (1 + f_v(x))) \\
&= \max(1 + f_v(u), \max_{x \in C_u(v)} (1 + f_u(x))) \\
&= \max(1 + f_v(u), f_u(v)).
\end{aligned}
$$

$\max(1 + f_v(u),$

$f_u(v$

Tree DP
Rerooting

```
int f[N], h[N];
void reroot(int u, int p) {
    h[u] = f[u]; // at this step, f[x] stores f_u(x)

    int argmax = -1;
    for (auto v : g[u])
        if (f[u] == 1 + f[v])
            argmax = v;

    for (auto v : g[u]) {
        if (v == p) continue;
        int init_fv = f[v], init_fu = f[u];
        if (argmax == v) {
            f[u] = 0;
            for (auto x : g[u])
                if (x != v)
                    f[u] = max(f[u], 1 + f[x]);
        }
        f[v] = max(1 + f[u], f[v]); // now f stores f_v
        reroot(v, u);
        f[v] = init_fv; f[u] = init_fu;
    }
}
dfs(r, r); // this computes f_r and stores it in f
reroot(r, r);
```

f stores f_u

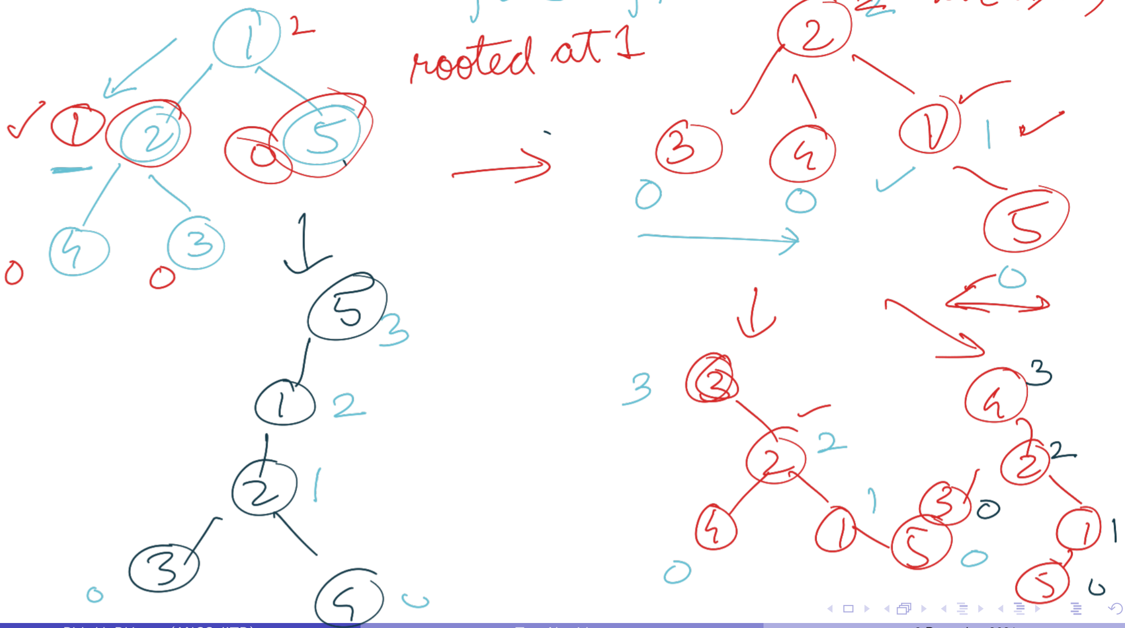
f now stores f_v

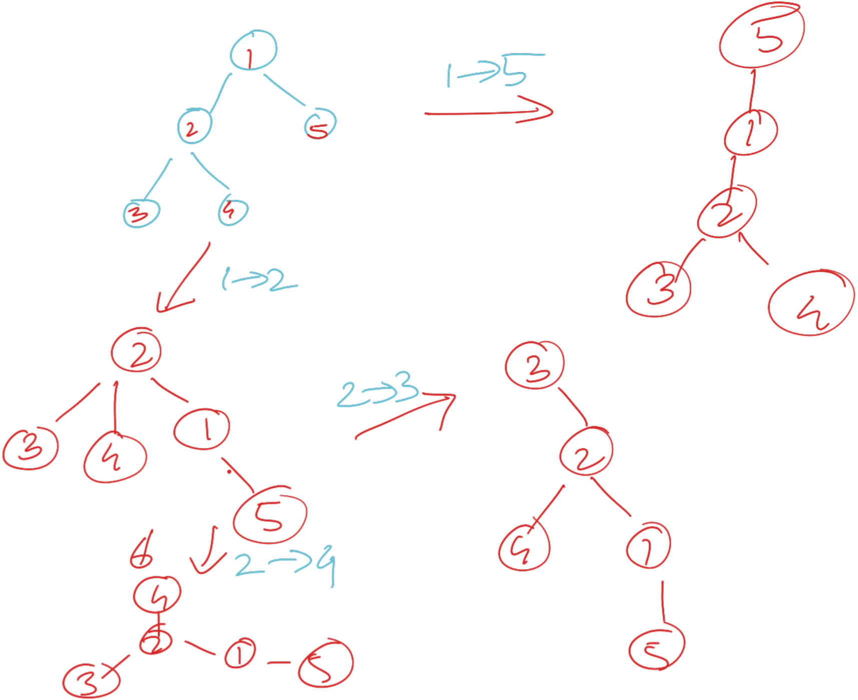
$h(u) = f_u(u)$

$f[u] = f_1$

rooted at 1

$\max(1, 2)$




1→5
1→2
2→3
2→4