Binary Lifting & LCA

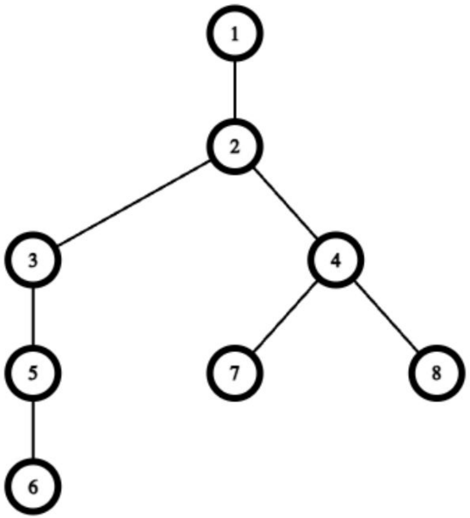
- Raghav Goel

Problem

Given a tree with **N** ($1 <= N <= 10^5$) nodes and **Q** ($1 <= Q <= 10^5$) queries.

Each query contains two integers **u** and **k**, print the **kth ancestor** of node **u**.

Example:

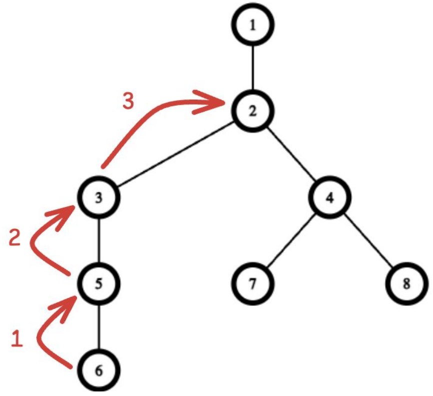



```
query(3, 2) -> 1
query(5, 1) -> 3
query(7, 2) -> 2
```

Straightforward Approach

- All the queries are independent of each other, **so we can solve them independently.**
- For each query, initialize **node = u** and then climb up (i.e. node = parent[node]) k times.

query(6, 3) -> 2



TLE

Time Complexity -> O(∑K) => O(N * Q) in worst case

Space Complexity -> O(N)

Precomputation Intensive Approach

- For each node, precomputate all of its ancestors.
- For each query(u, k), we have already precomputed the kth ancestor of the node u, we can answer the queries in O(1).

Time Complexity -> $O(N^2)$ for precomputation and $O(1)$ per query,
i.e. $\mathbf{O(N^2 + Q)}$ in worst case

Space Complexity -> $O(N^2)$

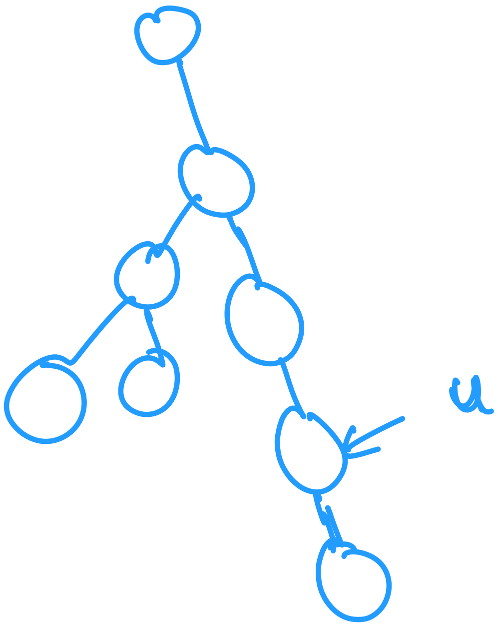
$N \leq 10^3$ ✓
$Q \leq 10^7$

Binary Lifting Approach

- For each node, precomputate its $2^0, 2^1, 2^2, 2^3, \ldots$ ancestors.
- For a query(u, k), decompose k into powers of 2 using binary representation, and use the precomputed ancestors to make the largest possible jumps at each step.

Time Complexity -> O(N * logN) for precomputation and O(∑log(K)) per query, i.e. **O(N * logN + Q * logN)** in the worst case
Space Complexity -> O(N * logN)

Tree



query $\rightarrow$ u & k

k^{th} ancestor of node u

u 1 2 4 8 16 32

$u \rightarrow u_1 \rightarrow u_2 \rightarrow u_3$

$\downarrow$

$\vdots$

15

u 1 2 3 4 5 6 7 8

u: $\underset{2^0}{1}$ $\underset{2^1}{2}$ $\underset{2^2}{4}$ $\underset{2^3}{8}$ $\underset{2^4}{16}$ $\underset{2^5}{32}$

$n \rightarrow 01101101$

27^{th}

i^{th}

11011

$2^4 + 2^3 + 2^1 + 2^0$

$16 + 8 + 2 + 1$

$u \rightarrow u_{16} \rightarrow u_{24} \rightarrow u_{26} \rightarrow u_{27}$

i

2

query(u, k)

$\downarrow$

log k

1023 $\quad$ n

$\downarrow$

$\underbrace{1111111111}_{\text{10 bits}}$

log(n)

[u, 537]

$512 + 16 + 8 + 1$

$2^9 \quad 2^4 \quad 2^3 \quad 2^0$

node = u

node = anc[node][9]

" " [4]

" " [3]

[0]

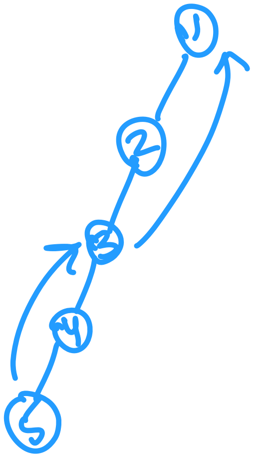
node

ancestor[n][log(n)]

$\boxed{\text{ancestor}[u][k]} \rightarrow 2^k$ th ancestor of node u

(1) ancestor[u][0] → parent of u

(2)



mid

anc[u][k] → $\boxed{\text{anc}[u][k-1]}$

↓

anc[mid][k-1]

dp

```
for (int k=1; k<LOG; k++){        → log(n)
    for (int u=1; u≤n; u++){      → n
        mid = anc[u][k-1]
        anc[u][k] = anc[mid][k-1]
    }
}
```

0-based indexing mid might be -1

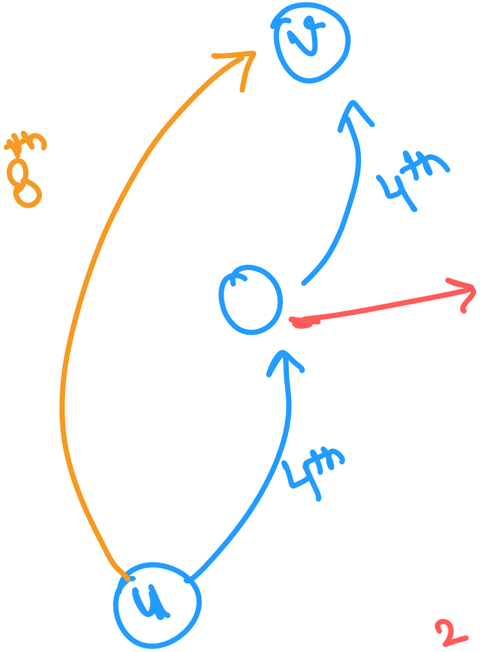
$O(n \log n)$

(u, 2^k) → O(1)

2^{k-1}

$2^0 \quad 2^1 \quad 2^2$

2^3 th



midAncestor

$K = 0 \quad 2^0$ dfs

$2^1 \quad u = 1$ to n

$$\boxed{anc[u][1] = anc[anc[u][0]][0]}$$

2^2

1 to n

mid = anc[u][1]

anc[u][2] = anc[mid][1]

Binary Lifting Approach

Example:

For k = 20, its binary representation is $(10100)_2$, which decomposes into 16 + 4.

1. Jump to the 16th ancestor of u.
2. Then jump to the 4th ancestor of that node.

This allows us to find the k-th ancestor in O(log k) time in the worst case.

Finding Ancestors

```
void calculateAncestors() {
  // ancestor[u][k] -> 2^k-th ancestor of node u.
  for (int k = 1; k < LOG; k++) {
    for (int u = 1; u <= n; u++) {
      int midAncestor = ancestor[u][k - 1];
      int kth_ancestor = ancestor[midAncestor][k - 1];
      ancestor[u][k] = kth_ancestor;
    }
  }
}
```

depends upon ancestor[mid][k-1] which depends upon anc[u][k-1] thus we have k-loop outside

Lift/Jump

```
int lift(int u, int k) {
  for (int j = 0; j < LOG; j++)
    if ((k >> j) & 1) {
      u = ancestor[u][j];
    }
  return u;
}
```

LCA of Two Nodes

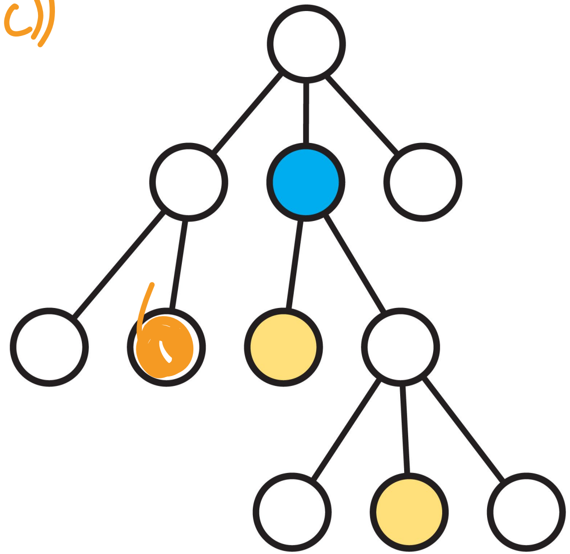
LCA can be for more than 2 nodes also

The **LCA or Lowest Common Ancestor** of two nodes u and v in a tree is the **deepest node that is an ancestor of both *u* and *v*.**

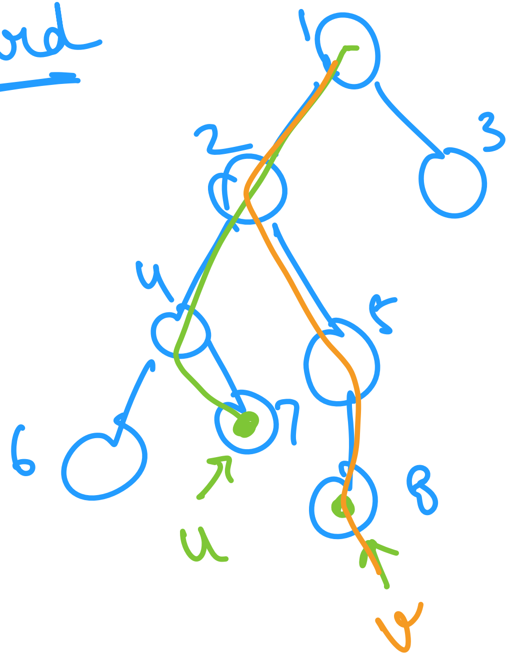
$lca(a,b,c) = lca(a, lca(b,c))$

Example:

The **blue node is the lowest / deepest node** that is a common ancestor for both of the **yellow nodes.**



Straightforward



7 4 2 1

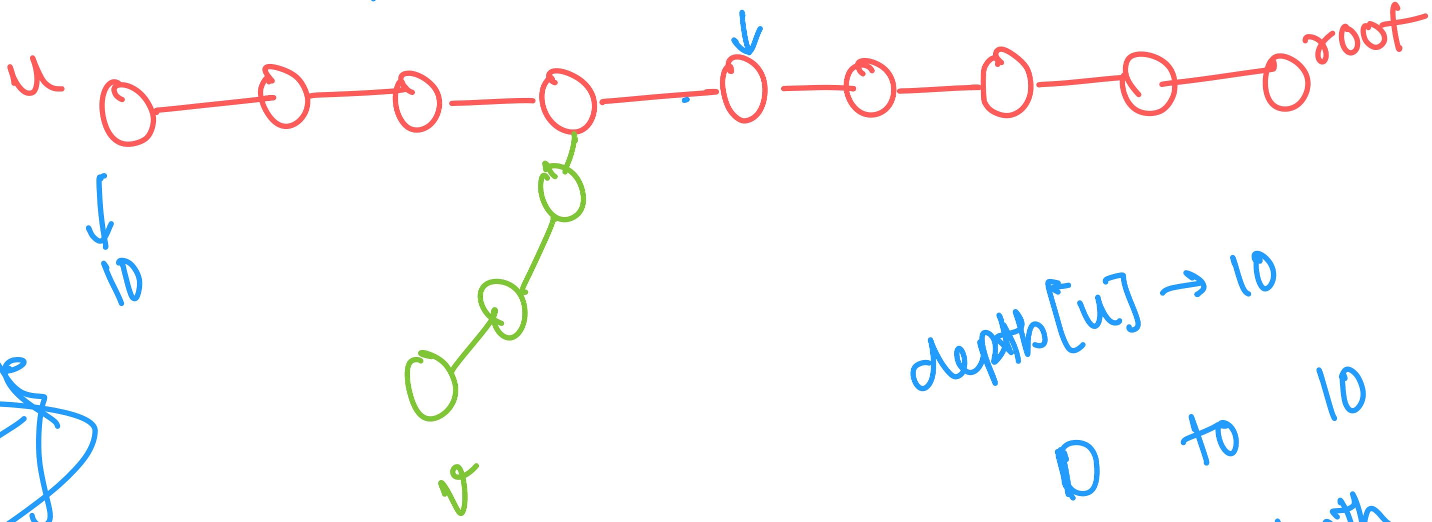
8 5 2 1

$\boxed{O(n)}$

Steps to find LCA

1. If u is an ancestor of v, then LCA is u.
2. If v is an ancestor of u, then LCA is v.
3. Otherwise find the highest ancestor of u that isn't an ancestor of v. The parent of that node will be LCA of u and v.

binary search to find the LCA



dfs ✓

level / depth

size

tin / tout

depth[u] → 10

0 to 10

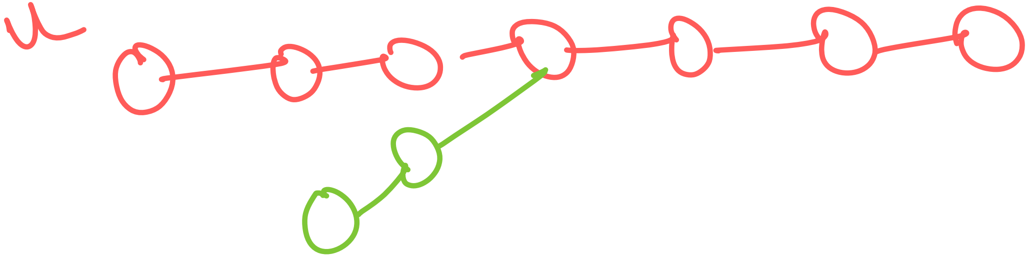
max. depth which is a common ancestor

mid

lift(u, 6)

instead of finding the lowest common ancestor

find the highest uncommon ancestor

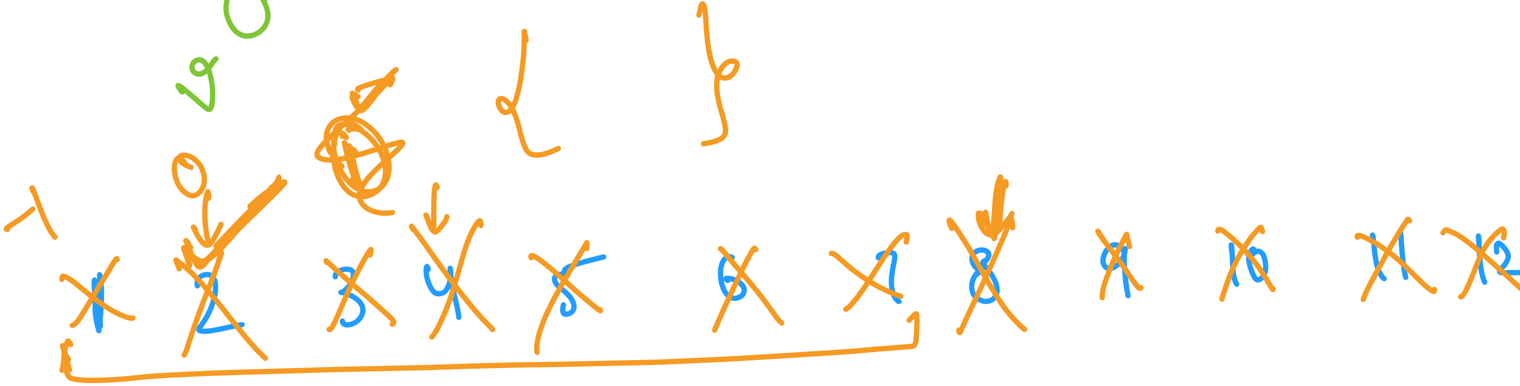


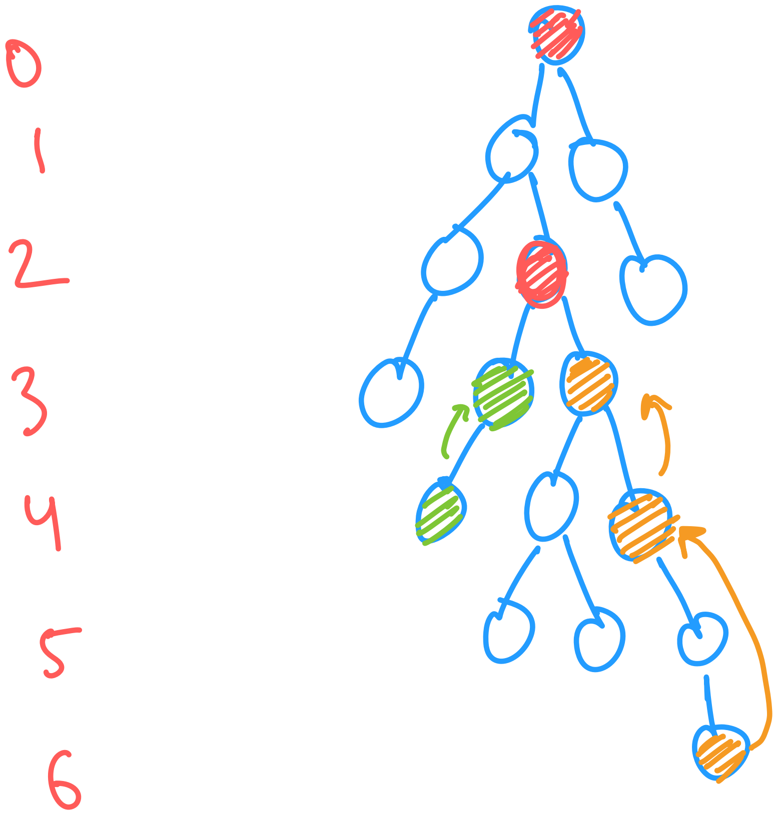
4th

1st, 2nd, 3rd

12

Highest uncommon anc.





$(6 - 4) = 2$

$\log n$

$anc[u][i] == anc[v][i]$

2^2

2^1

2^0

$O(n)$

$2^0 \quad 2^1 \quad 2^2 \quad 2^3 \quad 2^4 \quad 2^5 \quad 2^6 \quad \ldots \quad 2^{10} \quad \cancel{2^{11}}$

1024th

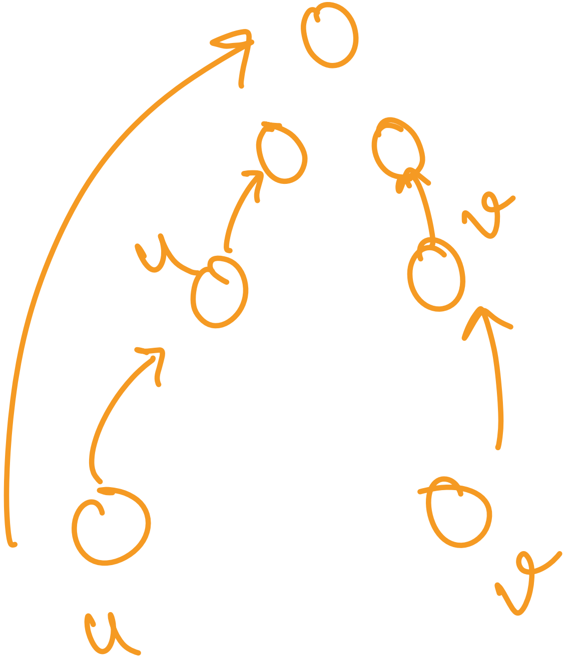
same

$O(\log(n) * \log(n))$

952

1976

3000



~~logn~~

8 4

1 2 3

log n

$2^0 \; - - - \; 2^{10}$

$2^0 \; \cdots\cdots \; 2^9$

$2^0 \; \cdots\cdots \; 2^8$

$\vdots$

$2^0 \; \cdots \; 2^0$

log n

2^{10}

2^9

2^8

Code for LCA

```
int lca(int u, int v) {
  if (depth[u] < depth[v]) swap(u, v);
  if (depth[u] > depth[v]) u = lift(u, depth[u] - depth[v]);
  if (u == v) return u;
  for (int j = LOG - 1; j >= 0; j--) {
    if (ancestor[u][j] != ancestor[v][j]) {
      u = ancestor[u][j];
      v = ancestor[v][j];
    }
  }
  return ancestor[u][0];
}
```

$\log n$

2^j $j \downarrow$

$\log n$

lca

Path (u, v)

Distance Between Two Nodes

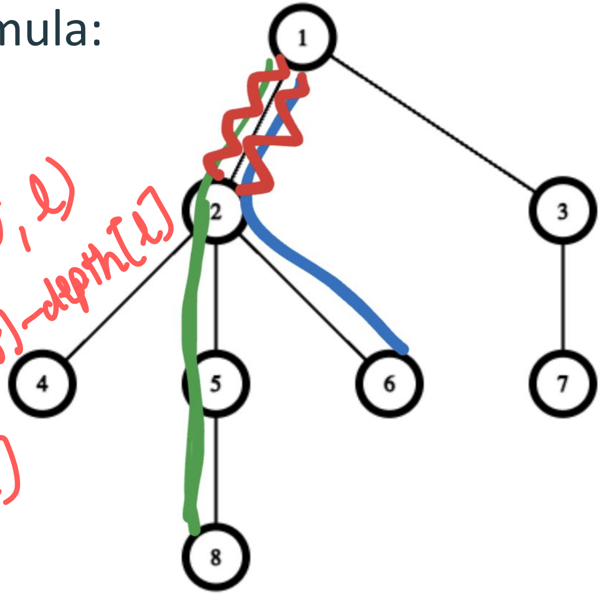
$$distance(u, v) = depth(u) + depth(v) - 2*depth(lca(u,v))$$

Visual Representation of the formula:

Green -> depth(u)

Blue -> depth(v)

Red -> depth(lca(u,v))



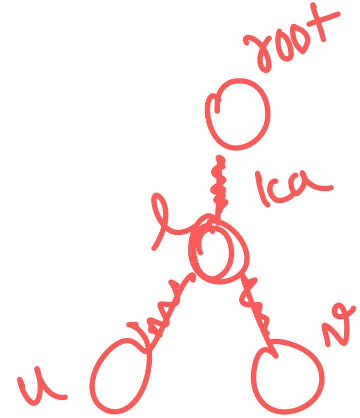
$dist(u,v) = dist(u,l) + dist(v,l)$

$\downarrow$ $depth[u] - depth[l]$

$\downarrow$ $depth[v] - depth[l]$

Sum of Values in a Path

Given a tree with **N** ($1 <= N <= 10^5$) nodes and **Q** ($1 <= Q <= 10^5$) queries, each query consists of two nodes **u** and **v**. For each query, compute the sum of all nodes in the path from **u** to **v**, inclusive.



sum_path(u, v)

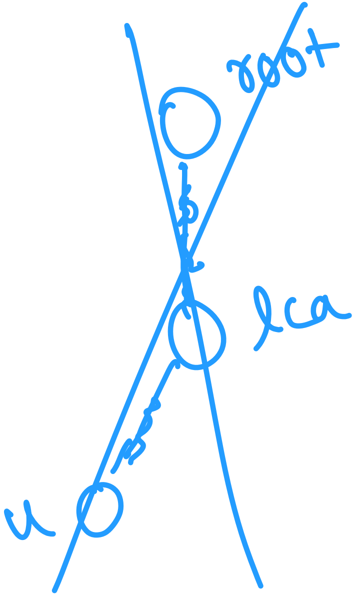
$\rightarrow$ $\boxed{\text{sum\_path}(u, l) + \text{sum\_path}(v, l) - \text{value}[l]}$

$$\text{sum\_path}(u, l) = \text{sum\_from\_root}(u) - \text{sum\_from\_root}(\text{par}[l])$$

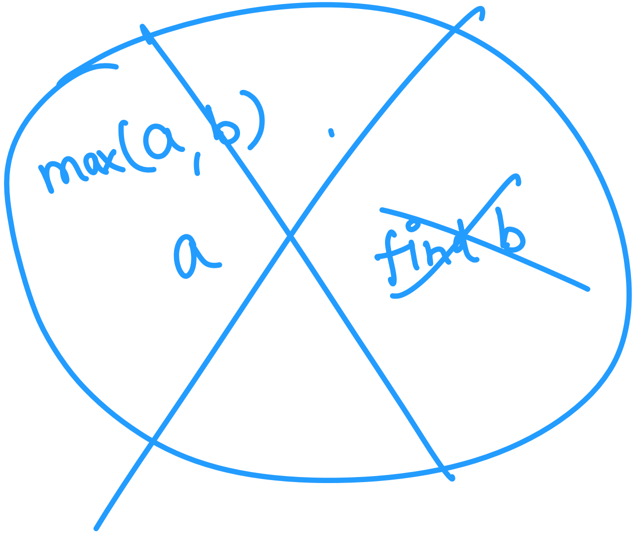
Maximum Value in a Path

Given a tree with **N** ($1 <= N <= 10^5$) nodes and **Q** ($1 <= Q <= 10^5$) queries, each query consists of two nodes **u** and **v**. For each query, compute the maximum value of all nodes in the path from **u** to **v**, inclusive.

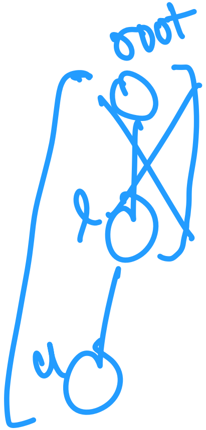
$$\text{max\_path}(u, v) = \max(\underline{\text{max\_path}(u, l)}, \text{max\_path}(v, l))$$



root
lca
u

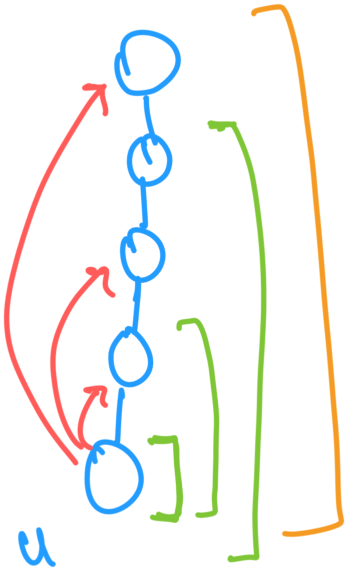


max(a, b)
a
find b



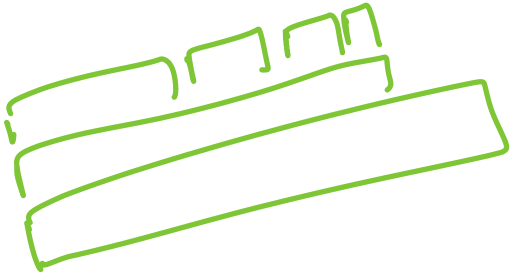
root
l
u

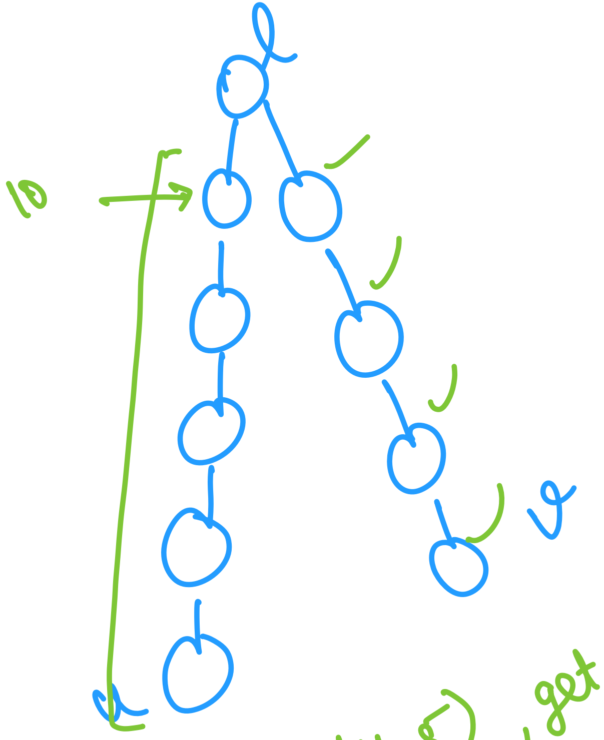
• ~~intuitive~~



max_path(u, l)

depth[u] - depth[l] => 5





$\underline{mx}[u][j] \Longrightarrow$ max. value in from u to 2^j th anc. of u (excluded)

max(get_max(u,5) , get_max(v,4), a[l])