LCA and some other topics on tree

Ran Xiao
The Affiliated High School of Peking University
2020.5
xiaoran@i.pkuschool.edu.cn

Contents

- 1. Lowest Common Ancestor
  - concept
  - binary lifting method
  - other methods
- 2. Some other tricks on tree
  - difference array & partial sum on tree
  - in-out difference on tree
- 3. Tasks

Notes

- u,v: vertex/node on tree
- rt: the root of the tree
- $fa(u)$: the father of u
- $son(u)$: sons of u
- $sub(u)$: all vertex in subtree of u (include u)
- $path(u,v)$: all vertex on the simple path from u to v
- $anc(u)$: all vertex $v \in path(u,rt)$ (include u)
- $lca(u,v)$: lowest common ancestor of u and v
- $dist(u,v)$: number of edges in the path from u to v

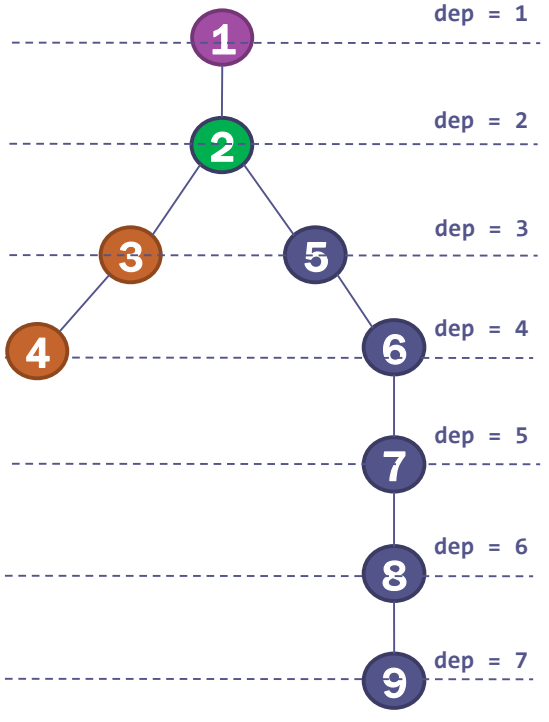
- dep[u] = $dist(u,rt) + 1$
- dfn[u] = the dfs order of u
- anc[u][i] = the 2^i ancestor of u

Lowest Common Ancestor

1. concept
2. binary lifting method
3. other methods

Lowest Common Ancestor(lca)

- **Given a rooted tree and 2 nodes *u* and *v*:**
  - *anc(u)*
    - all vertexs $v \in path(u,rt)$
  - *Common Ancestors of (u,v)*
    - intersection of *anc(u)* and *anc(v)*
  - *lca(u,v)*
    - the vertex with the maximum depth among all the common ancestors of (*u*,*v*)

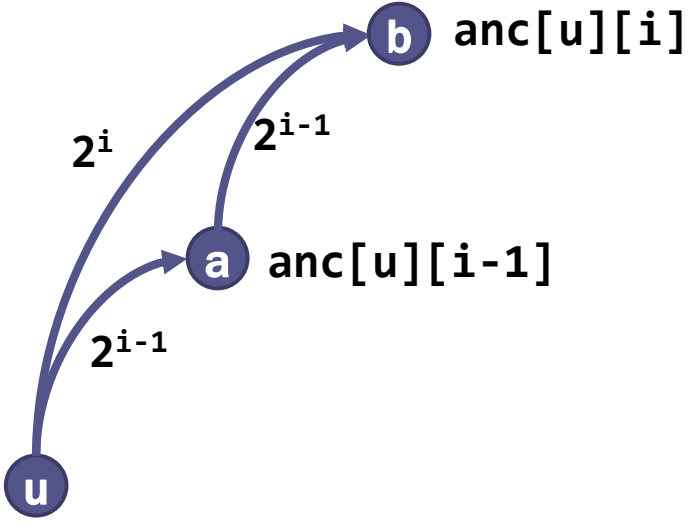


anc(4) = {1,2,3,4}
anc(9) = {1,2,5,6,7,8,9}
common anc. of (4,9) = {1,2}
lca(4,9) = 2

[SPOJ] LCA: https://www.spoj.com/problems/LCA/

binary lifting method

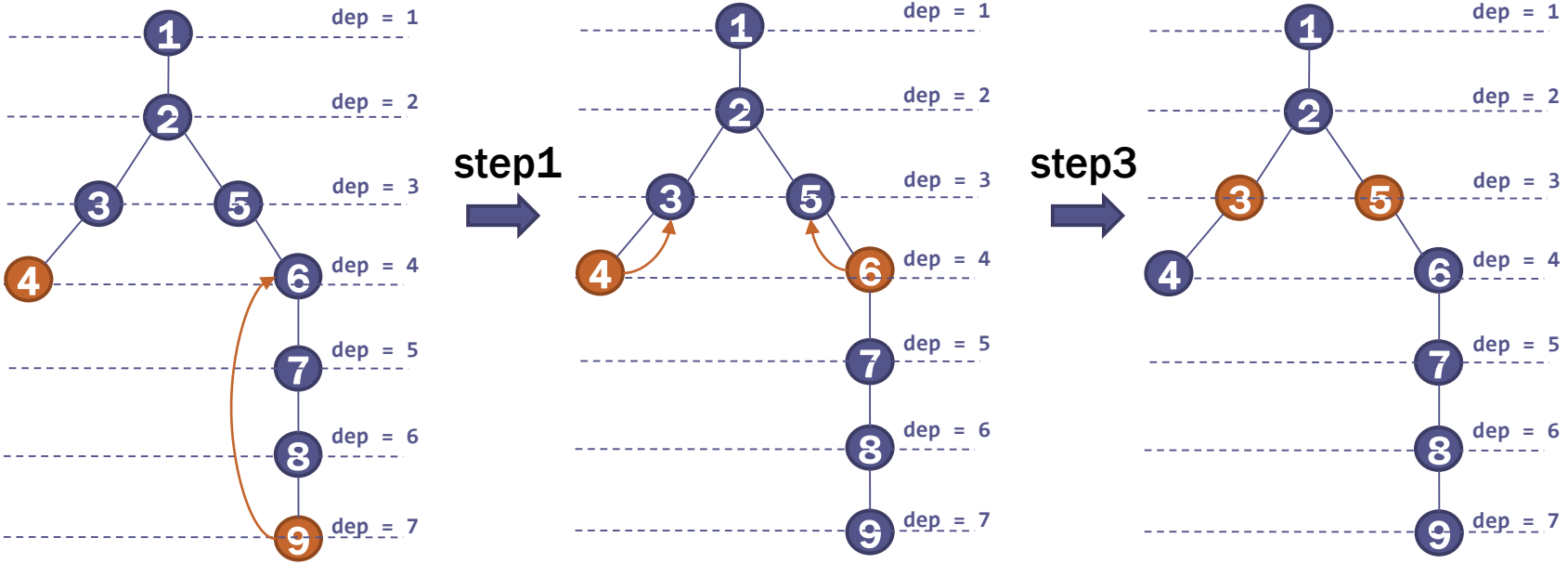
- Pre-processing in O(nlogn)
- anc[u][i] = the 2^i-th ancestor of u
  - anc[u][0] = fa[u]
  - anc[u][i] = anc[anc[u][i-1]][i-1]
- DP: anc[v][·] for all $v \in anc(u)$ must be known before calculating anc[u][·] -> dfs order
- These information allow us to jump from u to any $v \in anc(u)$ in O(logn) time.




```
vector<int> adj[MAXN];
int dfn[MAXN], timer = 0;
int dep[MAXN], anc[MAXN][21];

void dfs(int u){
    dfn[u] = ++timer;

    for(int j=1;j<=20;j++){
        anc[u][j] = anc[anc[u][j-1]][j-1];
    }

    int v;
    for(int k=0;k<adj[u].size();k++){
        v = adj[u][k];
        if(dfn[v]) continue;
        dep[v] = dep[u] + 1;
        anc[v][0] = u;
        dfs(v);
    }
}
```

binary lifting method

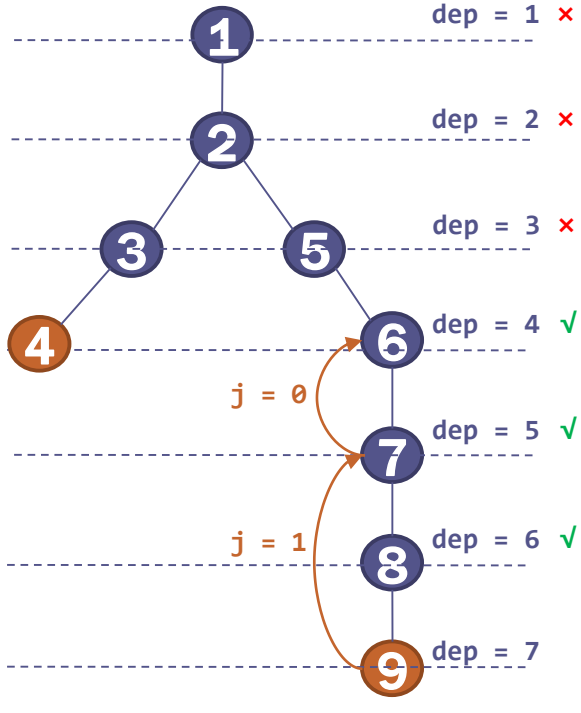
- step1: jump to same level
- step2: check if u and v on same chain
- step3: jump simultaneously to the nearest level under lca



step1

```
if(dep[u] < dep[v]) swap(u,v);
//step1 jump to same level
for(int j=20;j>=0;j--){
    if(dep[anc[u][j]] >= dep[v]){
        u = anc[u][j];
    }
}
```

- v = 4, dep[v] = 4
- u = 9, dep[u] = 9
- j = 1:
  - anc[u][1] = 7, dep[7] = 5
  - check dep[anc[u][j]]>=dep[v] -> √
  - u = anc[u][1] = 7
- j = 0:
  - anc[u][0] = 6, dep[6] = 4
  - check dep[anc[u][j]]>=dep[v] -> √
  - u = anc[u][0] = 6



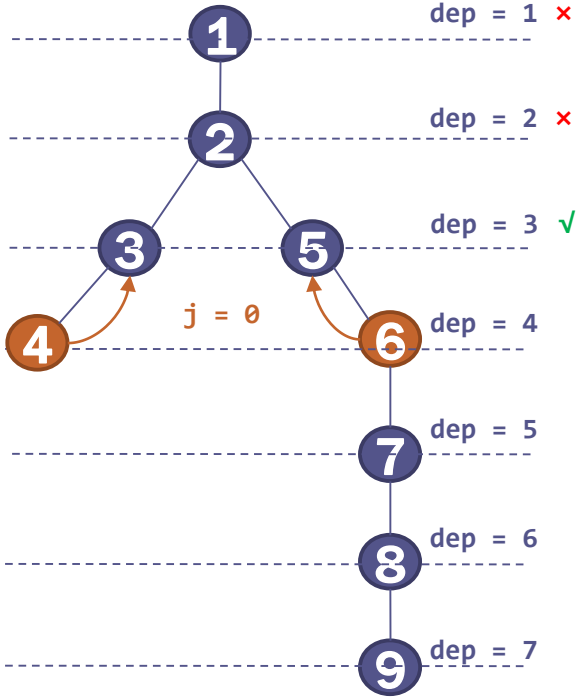
×/√ dep[anc[u][j]]>=dep[v]

step3

```
//step3 jump to the nearest level under lca
for(int j=20;j>=0;j--){
    if(anc[u][j] != anc[v][j]){
        u = anc[u][j];
        v = anc[v][j];
    }
}
```

- v = 4
- u = 6
- j = 0:
  - anc[u][0] = 5
  - check anc[u][j]!=anc[v][j] -> √
  - u = anc[u][0] = 5

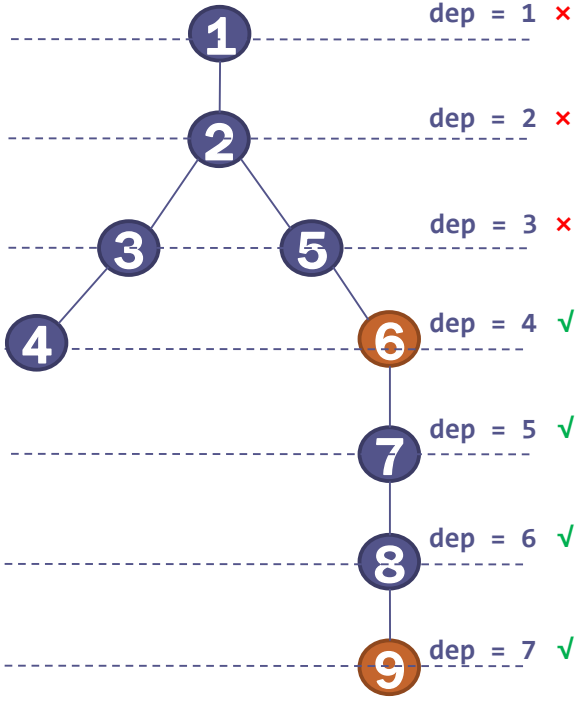
- lca = anc[u][0] = anc[v][0] = 2



×/√ anc[u][j]!=anc[v][j]

binary lifting method - code

```
int lca(int u, int v){
    if(dep[u] < dep[v]) swap(u,v);
    //step1 jump to same level
    for(int j=20;j>=0;j--){
        if(dep[anc[u][j]] >= dep[v]){
            u = anc[u][j];
        }
    }
    //step2 on same chain
    if(u == v) return u;

    //step3 jump to the nearest level under lca
    for(int j=20;j>=0;j--){
        if(anc[u][j] != anc[v][j]){
            u = anc[u][j];
            v = anc[v][j];
        }
    }
    return anc[u][0];
}
```



Why need step2?

LCA - Other methods

- **Heavy-Light Decomposition**
  - any path can be decomposed to ~log(N) heavy chains
  - O(n)-O(logn)

- **Reduction to RMQ**
  - Euler Tour: write down the depth while each time you enter a vertex, total len = 2(n-1)
  - lca(u,v) = minimum element in interval [first[u], first[v]]
    - where first[u] is the position of first occurrence of u in Euler Tour
  - Sparse Table: O(nlogn)-O(1)
  - Restricted(+/-1) RMQ: so-called *Farach-Colton and Bender Algorithm*[2], O(n)-O(1)

- *Tarjan's Algorithm*[1]
  - offline
  - Dfs and DSU
  - O(nα(n))

[1] Dov Harel, Robert Endre Tarjan, *Fast Algorithms for Finding Nearest Common Ancestors*, SIAM J. Comput. 13(2): 338-355 (1984)
[2] Michael A. Bender, Martin Farach-Colton, *The LCA Problem Revisited*, LATIN 2000: 88-94

Some other tricks on tree

1. difference array & partial sum on tree
2. in-out difference on tree

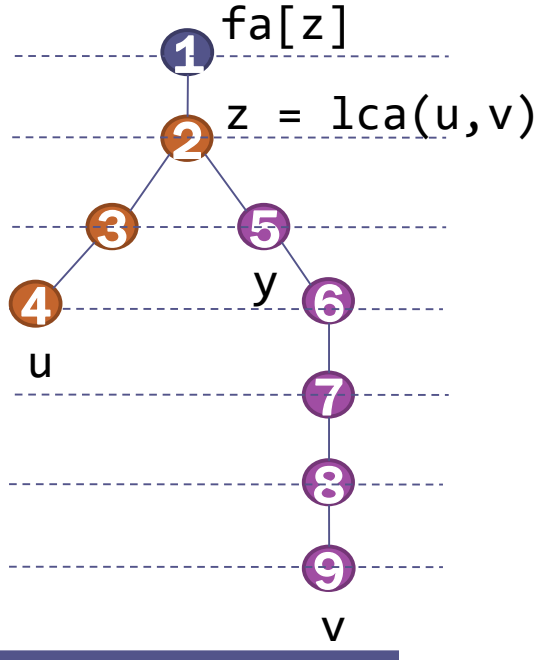
difference array & partial sum

- **Problem:**
  - giving a tree with weight on vertex a[1…n](intially all zero), m updates:
    - (u v x): add x on *path(u,v)*
  - output the a[u] after all queries

  - n = number of vertex
  - m = number of query
  - n,m ~ 1e5
  - offline/online

difference array & partial sum

- define the difference array on tree:
  - d[u] = a[u] – Σa[son(u)]
- adding x on *path(u,v)* can be decomposed by adding on 2 chains:
  - add x on *path(u,z)*, *where z = lca(u,v)*
  - add x on *path(v,y)*, where *y* is both *son(z)* and *anc(v)*
- add x on chain(u,z), where z∈*anc(u)*
  - d[u] += x
  - d[fa[z]] -= x
- ask a[u]
  - a[u] = Σd[sub(u)]



difference array	on sequence	on tree
definition	d[1] = a[1] d[i] = a[i] – a[i-1]	d[leaf] = a[leaf] d[i] = a[i] – Σa[son(i)]
query a[i]	a[i] = Σd[1…i]	a[i] = Σd[sub(i)]

- **offline: O(nlogn)**

update d[] in O(logn) time per query, dfs traversal processing sum for all subtrees

```
//offline O(nlogn) - O(1)
int update(int u, int v, int x){//O(logn)
    int z = lca(u,v);
    d[u] += x;
    d[v] += x;
    d[z] -= x;
    d[fa[z]] -= x;
}

void dfs(int u){//O(n)
    dfn[u] = ++timer;

    int v;
    for(int k=0;k<adj[u].size();k++){
        v = adj[u][k];
        if(dfn[v]) continue;
        dfs(v);
        d[u] += d[v];
    }
    ans[u] = d[u];//a[u] = Σd[sub(u)]
}
```

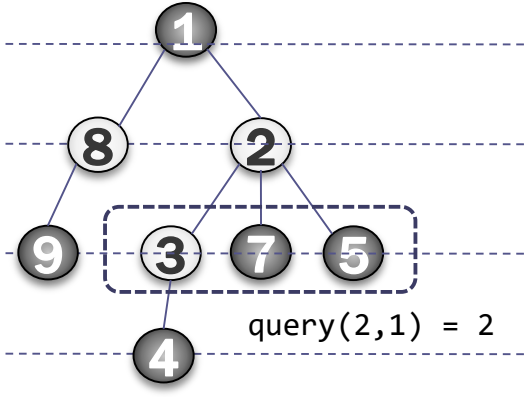
- **online: O(nlogn)**

the dfs order of sub(u) form a continuous interval, use a BIT with range sum function

```
//online O(nlogn) - O(logn)
int lowbit(int x){
    return x & (-x);
}

int change(int u, int x){
    for(int i = u;i<=n;i+=lowbit(i)){
        fw[i] += x;
    }
}

int prefix_sum(int x){
    int ans = 0;
    for(int i = x;i>0;i-=lowbit(i)){
        ans += fw[i];
    }
    return ans;
}

int update(int u, int v, int x){//O(logn)
    int z = lca(u,v);
    change(dfn[u], x);
    change(dfn[v], x);
    change(dfn[z], -x);
    change(dfn[fa[z]], -x);
}

int query(int u){//O(logn)
    return prefix_sum(dfn[u]+sz[u]-1) - prefix_sum(dfn[u]-1);
}
```

in-out difference on tree

- **Problem:**
  - giving a rooted tree, vertex is white or black. m queries:
    - *query(u,k)*: output the number of black vertex $v \in sub(u)$ and $dist(u,v)=k$
  - n = number of vertex
  - m = number of query
  - n,m ~ 1e5
  - Offline/online

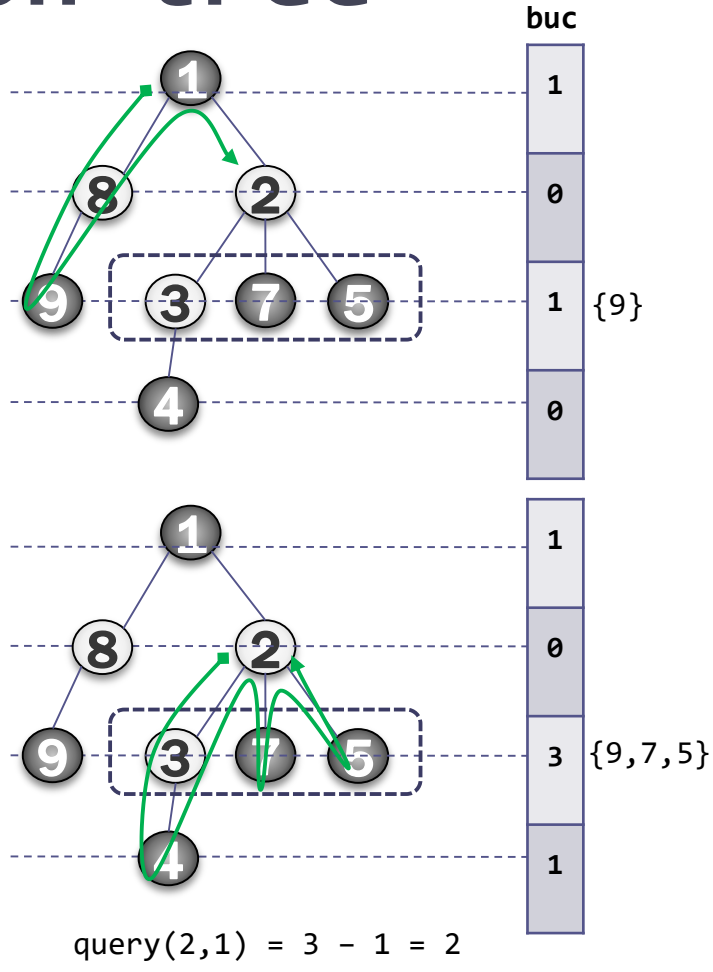


two constrains:

1. $v \in sub(u)$
2. $dep[v] = dep[u] + k$

in-out difference on tree

- Run dfs traversal and maintain a bucket array buc[1…n] for each depth. Each time entering a black vertex v, record it in buc[dep[v]].
- *query(u,k)*: check buc[dep[u]+k] whenever you enter as well as leave u, the difference is affected only by those black vertex in *sub(u)*.
  - enter u: ans0 = buc[dep[u]+k]
  - leave u: ans1 = buc[dep[u]+k]
  - ans[u] = ans1 – ans0
- O(n)
- online?
  - persistent segment tree



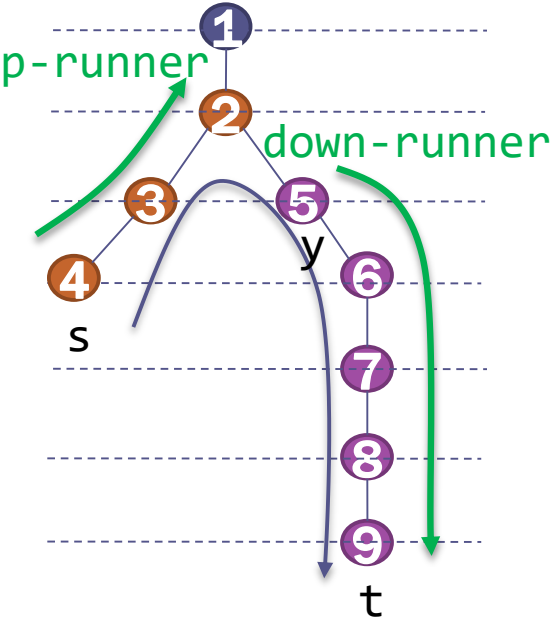
War Story: Tiantian loves running

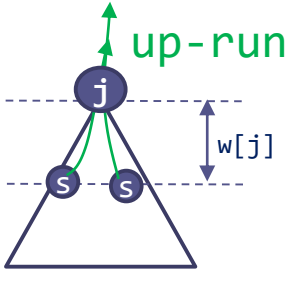
- NOIP2016 Day1T2
- Giving a tree with n vertex. m runners run from s[i] to t[i](i=1…m). All runners depart at t=0 and run at the speed of 1 edge per second.
- n observers at each node with observation time W[j](j=1…n). The observer *j* will recode the number of runners that just happened to reach j at t = W[j].
- Output the answer for each observer.
- n,m = 3e5

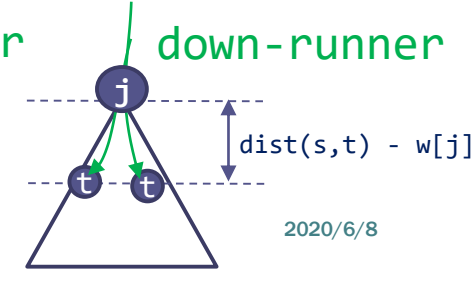
[loj2359] Tiantian loves running
https://loj.ac/problem/2359

War Story: Tiantian loves running

- **A runner from *s* to *t* can be decomposed into 2 runners:**
  - a up-runner runs from *s* to *z* = *lca(u,v)*
  - a down-runner runs from *y* to *t*, where *y* is both *son(z)* and *anc(t)*
- **For observer *j* with w[j]:**
  - a up-runner *i* can be recorded if:
    - *s[i]∈sub(j) and t[i]∉sub(j)*
    - dep[s[i]] – dep[j] = w[j]
    - add i to buc[dep[s[i]]] when enter s[i]
    - query j at buc[w[j]+dep[j]]
    - reomve i from buc when leave z
  - a down-runner *i* can be recorded if:
    - *s[i]∉sub(j) and t[i]∈sub(j)*
    - dist(s[i],t[i]) – (dep[t[i]] – dep[j]) = w[j]
    - add i to buc[dist(s[i],t[i]) - dep[t[i]]] when enter s[i]
    - query j at buc[w[j] - dep[j]]
    - reomve i from buc when leave y
- **use in-out difference on tree: O(nlogn), O(n) if using O(n) lca**







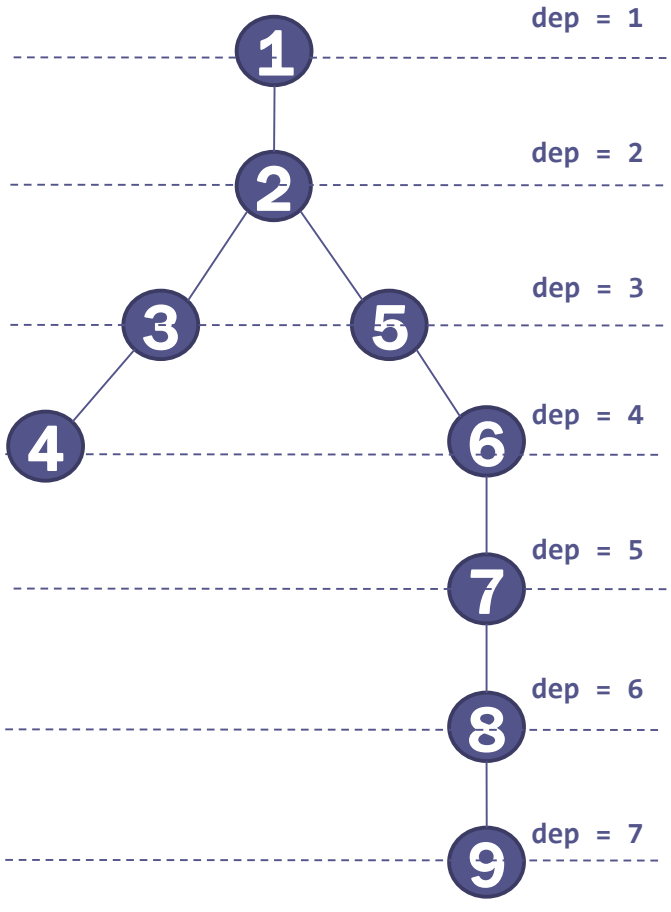
Tasks

- Task 1-3

lca(1,4) = ?
  - A. 1
  - B. 2
  - C. 3
  - D. 4

lca(3,5) = ?
  - A. 2
  - B. 1
  - C. 3
  - D. 5

lca(2,9) = ?
  - A. 1
  - B. 2
  - C. 5
  - D. 6



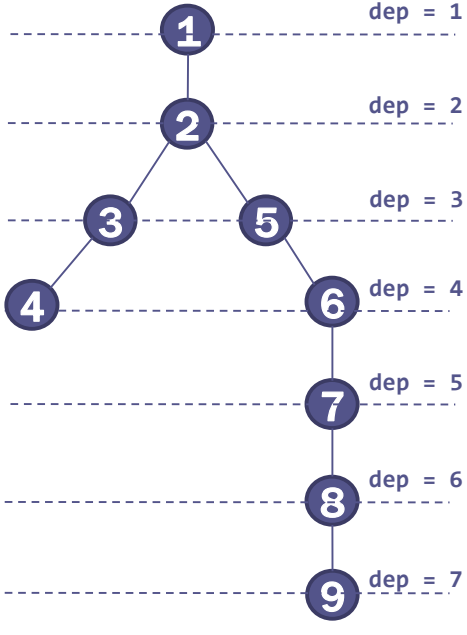
- Task 4-7

anc[9][0] = ?
- A. 9
- B. 8
- C. 7
- D. 6

anc[9][1] = ?
- A. 9
- B. 8
- C. 7
- D. 6

anc[9][2] = ?
- A. 1
- B. 2
- C. 5
- D. 6

anc[9][3] = ?
- A. 0
- B. 1
- C. 5
- D. 6




```
vector<int> adj[MAXN];
int dfn[MAXN], timer = 0;
int dep[MAXN], anc[MAXN][21];

void dfs(int u){
    dfn[u] = ++timer;

    for(int j=1;j<=20;j++){
        anc[u][j] = anc[anc[u][j-1]][j-1];
    }

    int v;
    for(int k=0;k<adj[u].size();k++){
        v = adj[u][k];
        if(dfn[v]) continue;
        dep[v] = dep[u] + 1;
        anc[v][0] = u;
        dfs(v);
    }
}
```

- Task 8

In lca problem, what is the time complexity of: binary lifting, RMQ(Sparse table), and Tarjan's method:

  - A. O(nlogn)-O(logn), O(nlogn)-O(1), O(nα(n))
  - B. O(nlogn)-O(1), O(nlogn)-O(logn), O(nα(n))
  - C. O(nlogn)-O(1), O(nlogn)-O(1), O(n)
  - C. O(nlogn)-O(logn), O(nlogn)-O(logn), O(n)

- Task 9

What strategies/ideas are used in binary lifting method (select 2 options):

  - A. Greedy
  - B. Dynamic Programming
  - C. Divide and Conquer(Binary search)
  - C. Brute Force

```
void dfs(int u){
    dfn[u] = ++timer;

    for(int j=1;j<=20;j++){
        ___?___
        anc[u][j] = anc[anc[u][j-1]][j-1];
    }

    int v;
    for(int k=0;k<adj[u].size();k++){
        v = adj[u][k];
        if(dfn[v]) continue;
        dep[v] = dep[u] + 1;
        anc[v][0] = u;
        dfs(v);
    }
}
```

- **Task 10**

In order to get faster, which if-statement can be put on “?”:

  - A. if(dep[u] <= (1<<j)) continue;
  - B. if(dep[u] < (1<<j)) continue;
  - A. if(dep[u] <= j) continue;
  - B. if(dep[u] < j) continue;

- **Task 11**

Giving a rooted tree with black/white vertex, *query(u,k)*: output the number of black vertex $v \in sub(u)$ and $dist(u,v)=k$. (same problem on page 16)

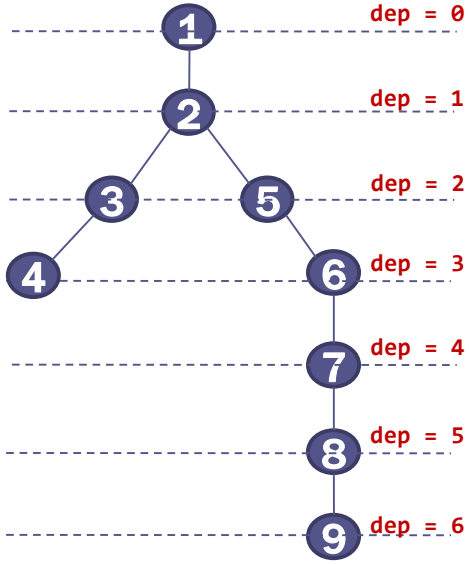
If you use persisitent segment tree to solve the *online version* of this problem, what is the time and space complexity?

  - A. O(nlogn), O(n)
  - B. O(n), O(n)
  - C. $O(n\log^2 n)$, O(nlogn)
  - A. O(nlogn), O(nlogn)

- **Task 12**

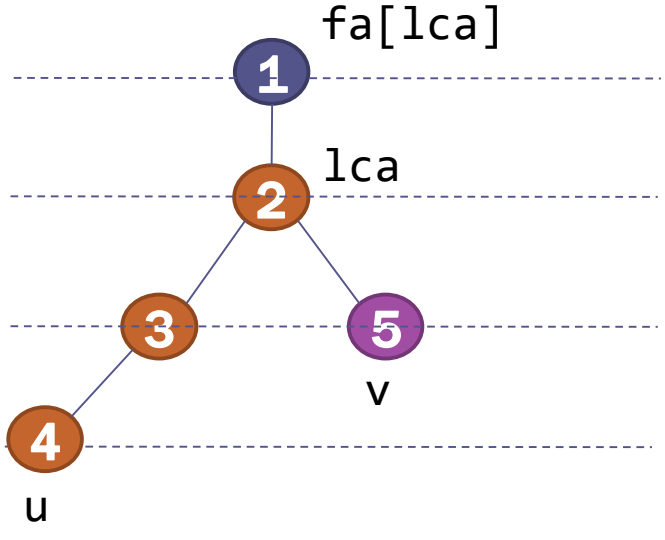
Try to hack the program using binary lifting method with dep[rt]=0. Construct a set of input(u,v) to make the program go wrong.

- A. u=3, v=5
- B. u=1, v=5
- C. u=2, v=5
- D. u=4, v=3




```
vector<int> adj[MAXN];
int dfn[MAXN], timer = 0;
int dep[MAXN], anc[MAXN][21];

void dfs(int u){
    dfn[u] = ++timer;

    for(int j=1;j<=20;j++){
        anc[u][j] = anc[anc[u][j-1]][j-1];
    }

    int v;
    for(int k=0;k<adj[u].size();k++){
        v = adj[u][k];
        if(dfn[v]) continue;
        dep[v] = dep[u] + 1;
        anc[v][0] = u;
        dfs(v);
    }
}

int lca(int u, int v){
    if(dep[u] < dep[v]) swap(u,v);
    for(int j=20;j>=0;j--){
        if(dep[anc[u][j]] >= dep[v]){
            u = anc[u][j];
        }
    }
    if(u == v) return u;
    for(int j=20;j>=0;j--){
        if(anc[u][j] != anc[v][j]){
            u = anc[u][j];
            v = anc[v][j];
        }
    }
    return anc[u][0];
}
```

- Task 13
- Using the difference array d[], after +1 operation on path(4,5), the d[1-5] will become:
  - A. [0,0,0,1,1]
  - B. [-1,-1,0,1,1]
  - C. [0,-2,0,1,1]
  - D. [-2,0,0,1,1]



- **Answer for task 1-13**
  - A/A/B
  - B/C/C/A
  - A/BC
  - A/D
  - B
  - B

- **Task 14-20**
  - [SPOJ] LCA
    - https://www.spoj.com/problems/LCA/
  - [CF191C] Fools and Roads
    - https://codeforces.com/problemset/problem/191/C
  - [loj2359] Tiantian loves running
    - https://loj.ac/problem/2359

  - [CF980E] The Number Games
  - [CF519E] A and B and Lecture Rooms
  - [CF832D] Misha, Grisha and Underground
  - [CF1110F] Nearest Leaf
  - [CF1076E] Vasya and a Tree
  - [CF739B] Alyona and a tree