26th

Trees
문제 풀이

문제

- 트리 순회 BOJ 1991
- 트리의 부모 찾기 BOJ 11725
- 완전 이진 트리 BOJ 9934
- 거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516
- 부동산 다툼 BOJ 20364

트리 순회 BOJ 1991

- 이진 트리가 입력으로 주어진다
- 이진 트리에서 전위 순회, 중위 순회, 후위 순회를 구현하는 문제

트리 순회 BOJ 1991

- 이진 트리를 위한 노드를 먼저 구현하자

```
struct Node {
    char data;
    Node *left;
    Node *right;
};

Node arr[n];
char x, l, r;
int n;
```

트리 순회 BOJ 1991

- 입력에 맞게 트리를 구현

```
cin >> n;
for (int i = 0; i < n; i++) {
    cin >> x >> l >> r;
    arr[i].data = 'A' + i;
    if (l != '.')
        arr[x - 'A'].left = &arr[l - 'A' ];
    if (r != '.')
        arr[x - 'A'].right = &arr[r - 'A'];
}
```

트리 순회 BOJ 1991

- 전위 순회는 본인->왼쪽 자식->오른쪽 자식 순서로 방문

```
void preorder(Node *n) {
    // 본인
    cout << n->data;
    // 왼쪽 자식
    if (n->left != nullptr)
        preorder(n->left);
    // 오른쪽 자식
    if (n->right != nullptr)
        preorder(n->right);
}
```

트리 순회 BOJ 1991

- 중위 순회는 왼쪽 자식-> 본인->오른쪽 자식 순서로 방문

```
void inorder(Node *n) {
    // 왼쪽 자식
    if (n->left != nullptr)
        inorder(n->left);
    // 본인
    cout << n->data;
    // 오른쪽 자식
    if (n->right != nullptr)
        inorder(n->right);
}
```

트리 순회 BOJ 1991

- 후위 순회는 왼쪽 자식 ->오른쪽 자식-> 본인 순서로 방문

```
void postorder(Node *n) {
    // 왼쪽 자식
    if (n->left != nullptr)
        postorder(n->left);
    // 오른쪽 자식
    if (n->right != nullptr)
        postorder(n->right);
    // 본인
        cout << n->data;
}
```

트리 순회 BOJ 1991

- 전위 순회, 중위 순회, 후위 순회 순으로 출력

```
preorder(&arr[0]);
cout << '\n';
inorder(&arr[0]);
cout << '\n';
postorder(&arr[0]);
```

트리의 부모 찾기 BOJ 11725

- 루트가 1번 노드인 트리가 주어진다
- 부모 자식 관계가 주어지지 않고 간선으로 주어질 때, 각 노드의 부모 노드를 구하는 문제
- 1번 노드는 루트이기 때문에 부모 노드가 존재하지 않는다
- 2번 노드부터 부모 노드를 출력한다

트리의 부모 찾기 BOJ 11725

- 트리 노드에 연결된 간선에는 2가지가 존재한다
- 부모로 향하는 간선 1개
- 자식으로 가는 나머지 간선

- 부모 노드로 향하는 간선을 알면 2가지를 구분할 수 있다

트리의 부모 찾기 BOJ 11725

- 부모 노드는 자식 노드보다 깊이가 낮다
- 루트 노드에서 순회를 시작하면 부모 노드를 자식 노드보다 먼저 방문한다
- 인접한 노드 중 먼저 방문한 노드가 부모 노드가 되며, 그 외 노드는 모두 자식 노드가 된다

트리의 부모 찾기 BOJ 11725

```
#define MAX_NODE 100001

vector<int> edges[MAX_NODE];
int parent[MAX_NODE];
bool visited[MAX_NODE];
int n;

int s, e;
cin >> n;
for (int i = 0; i < n - 1; i++) {
    cin >> s >> e;
    edges[s].push_back(e);
    edges[e].push_back(s);
}
```

트리의 부모 찾기 BOJ 11725

```
void dfs(int cur) {
    visited[cur] = true;
    for (auto next : edges[cur]) {
        if (visited[next])
            continue;
        parent[next] = cur;
        dfs(next);
    }
}

dfs(1);
for (int i = 2; i <= n; i++)
    cout << parent[i] << '\n';
```

완전 이진 트리 BOJ 9934

- 완전 이진 트리 구조가 주어진다
- 완전 이진 트리의 높이가 주어지고 순회하는 방법이 주어졌을 때, 각 노드의 번호를 정하는 문제

완전 이진 트리 BOJ 9934

- 완전 이진 트리이기 때문에 높이가 K일 때, 2^K - 1개의 노드를 가지고 있다
- 깊이가 0인 루트에는 1개의 노드, 깊이가 1인 노드들은 2개, 깊이가 2인 노드들은 4개, …, 깊이가 k인 노드들은 2^k개이다
- 이를 이용해 트리를 먼저 만들 수 있다

완전 이진 트리 BOJ 9934

```
struct Node {
    int num;
    Node *left, *right;
};

int k;
Node *root;

cin >> k;
root = new Node();
tree_init(root, 0);
```

완전 이진 트리 BOJ 9934

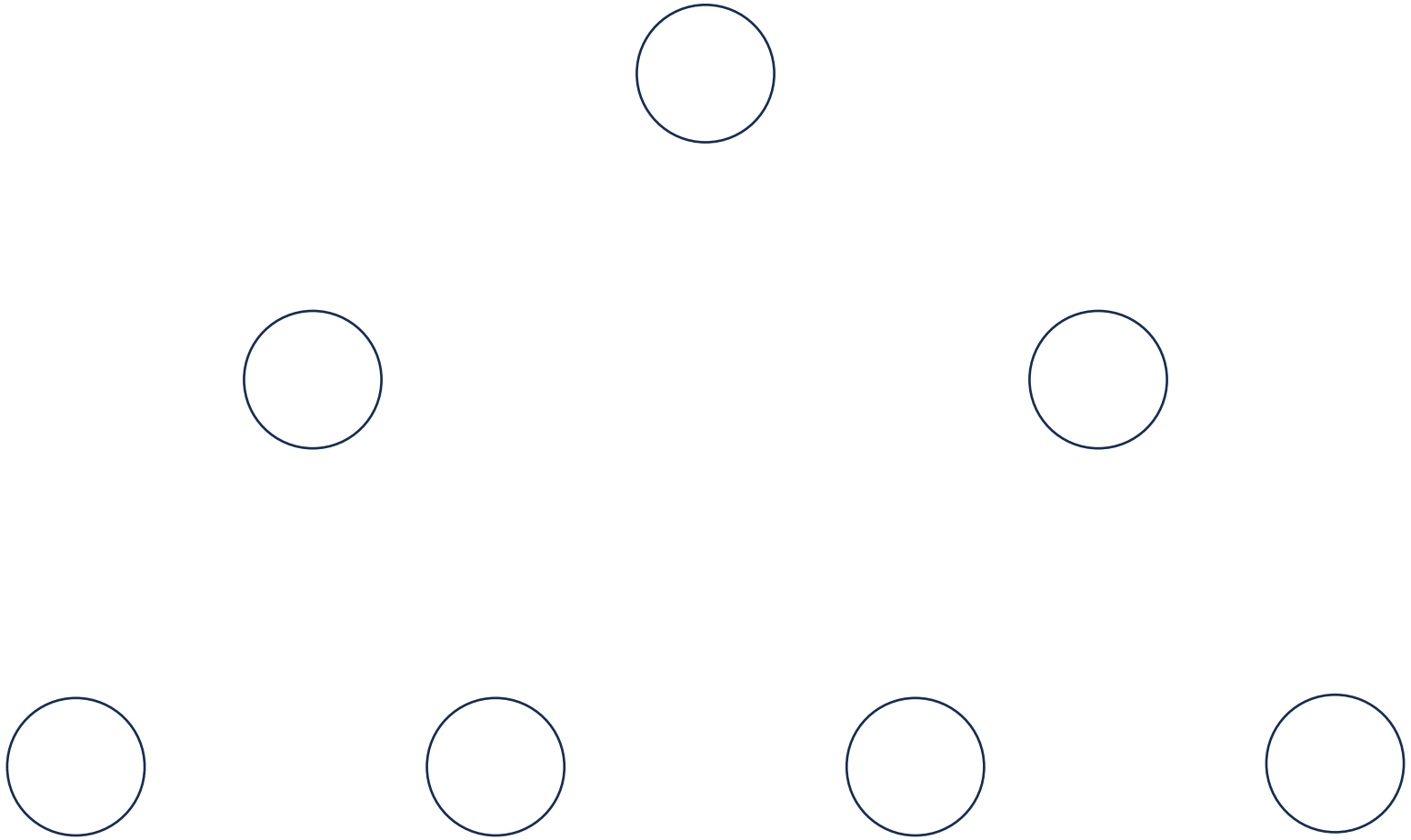
```
void tree_init(Node *cur, int depth) {
    if (depth == k - 1)
        return;

    cur->left = new Node();
    cur->right = new Node();

    tree_init(cur->left, depth + 1);
    tree_init(cur->right, depth + 1);
}
```

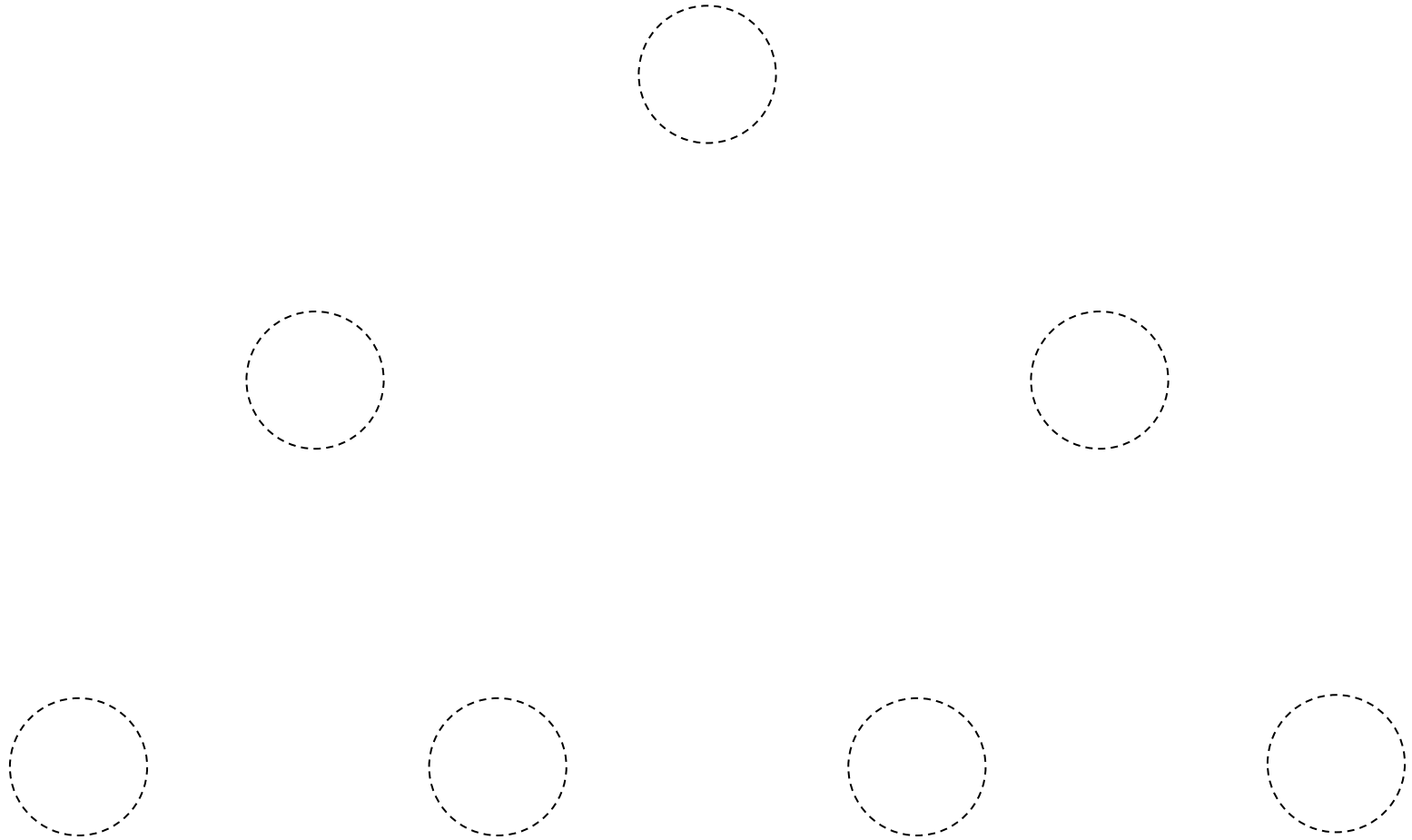
완전 이진 트리 BOJ 9934

- k == 3



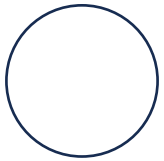
완전 이진 트리 BOJ 9934

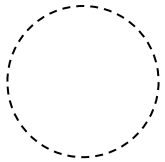
- k == 3

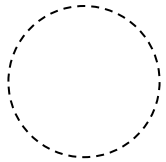


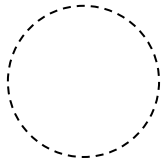
완전 이진 트리 BOJ 9934

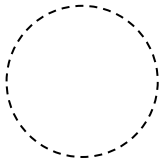
- root = new Node();

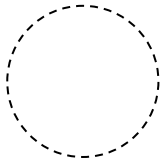


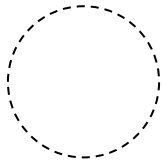






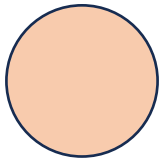


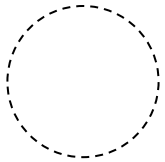


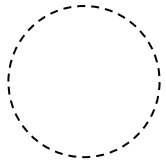


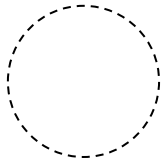
완전 이진 트리 BOJ 9934

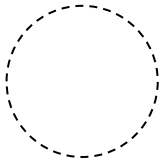
- dfs(root, 0)

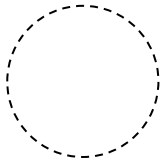


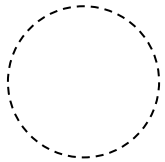






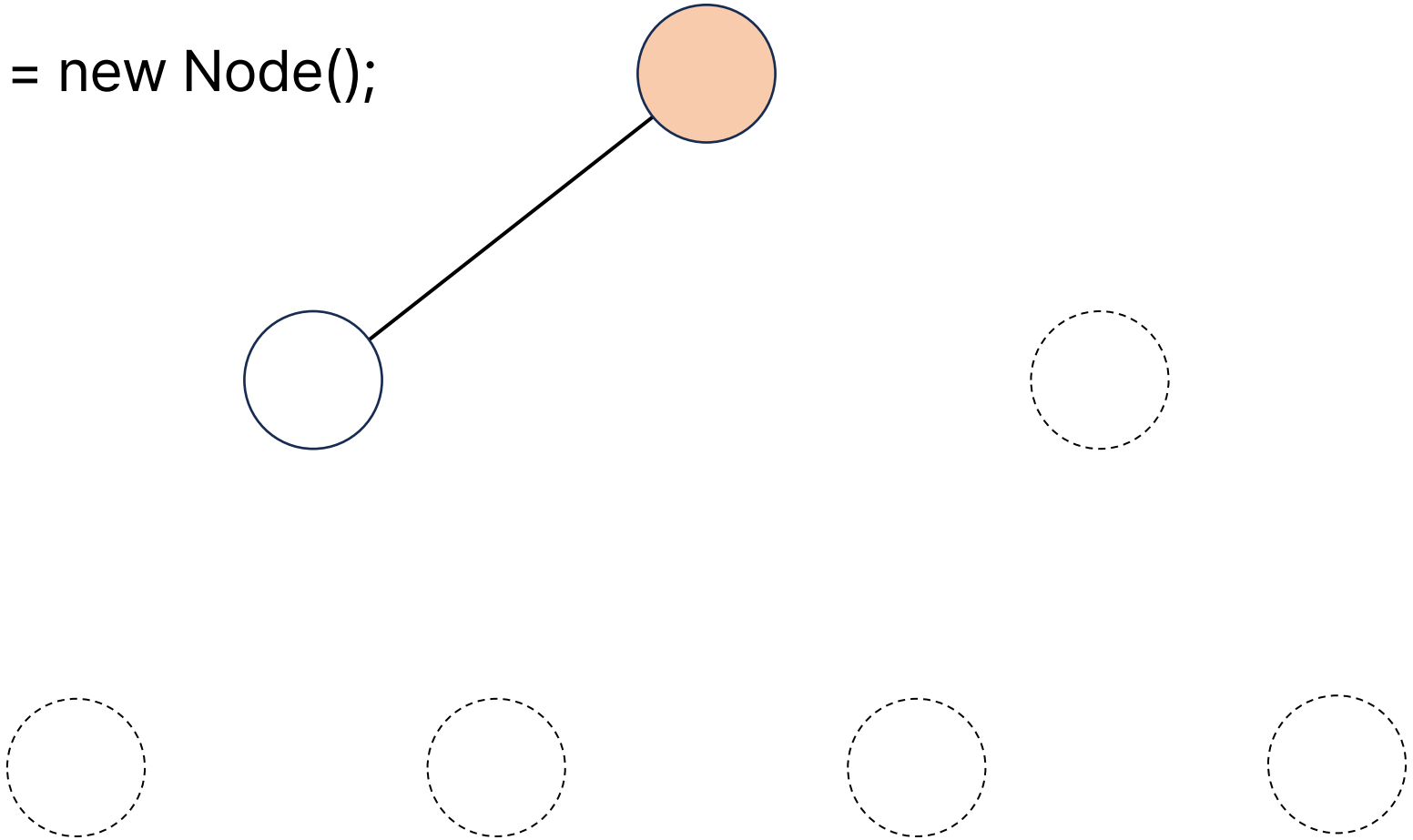






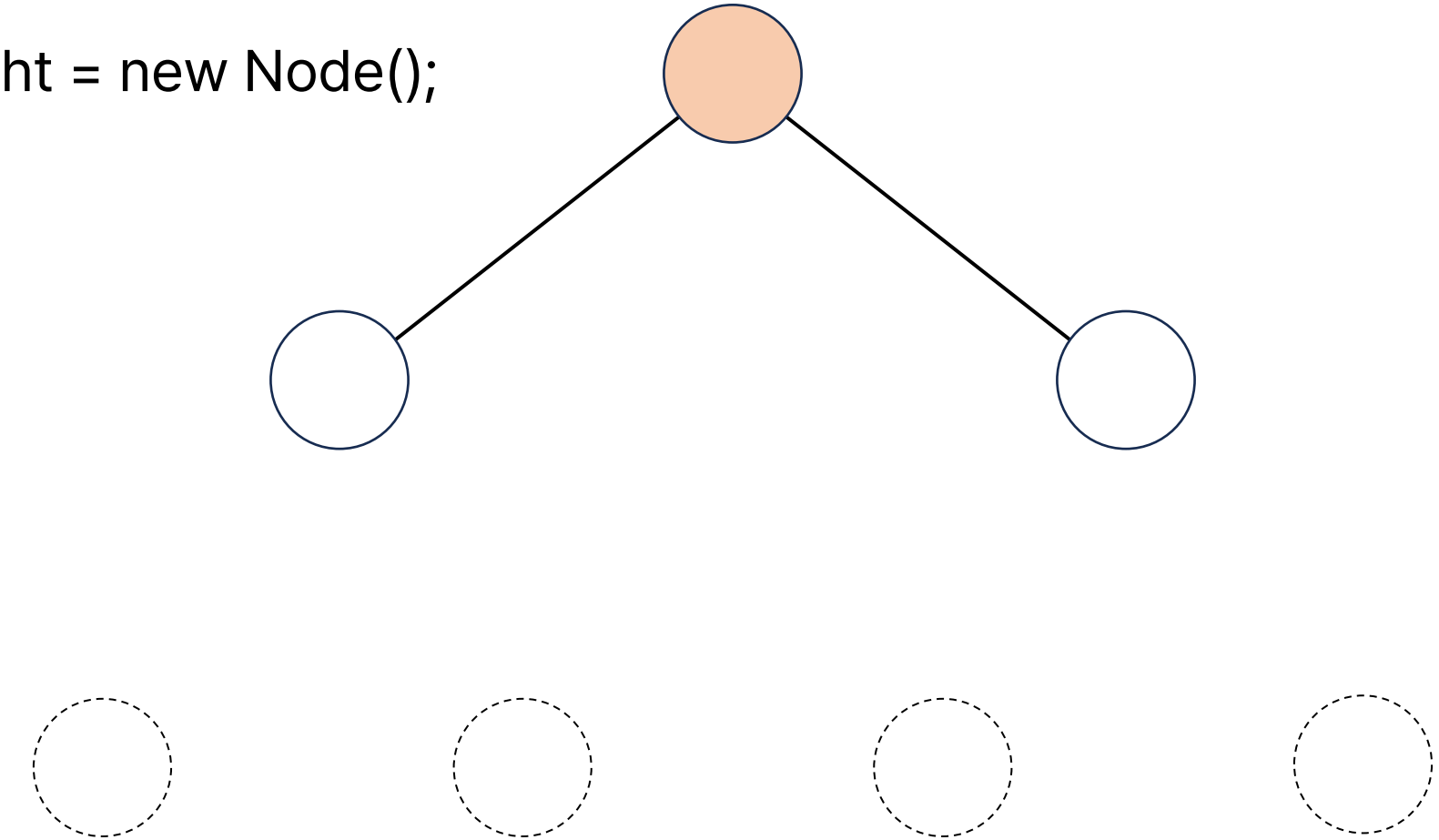
완전 이진 트리 BOJ 9934

- cur->left = new Node();



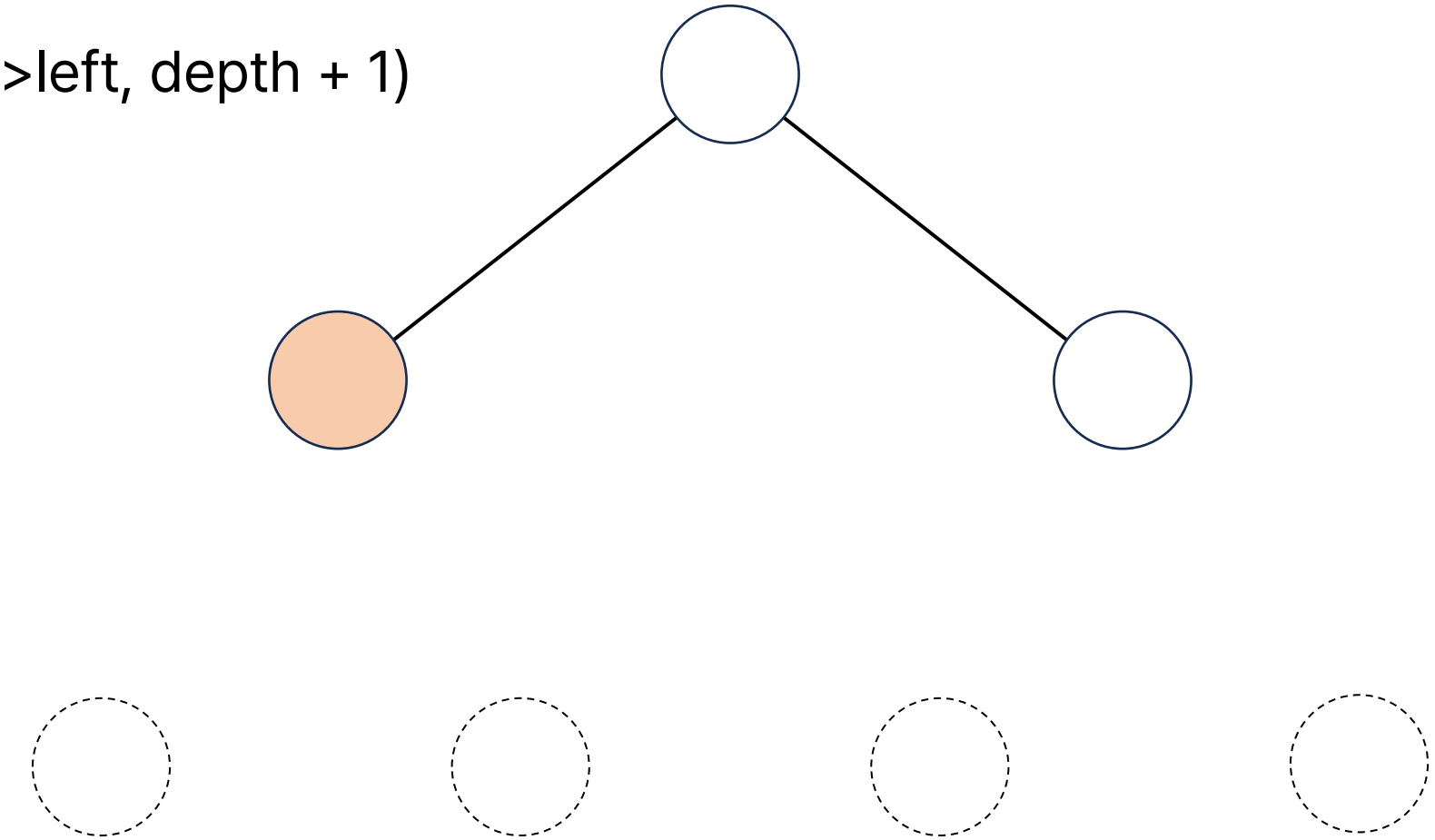
완전 이진 트리 BOJ 9934

- cur->right = new Node();



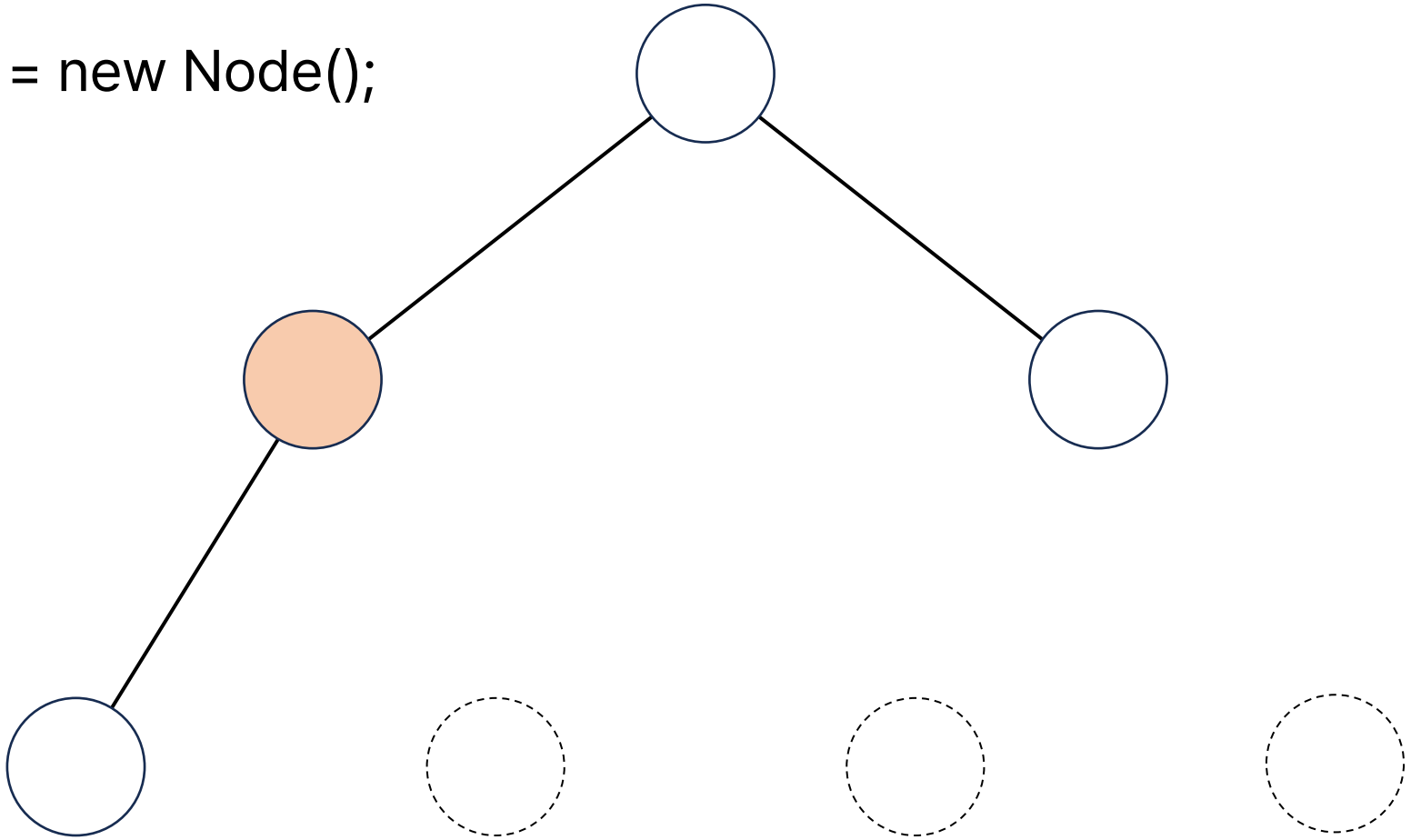
완전 이진 트리 BOJ 9934

- dfs(cur->left, depth + 1)
- depth: 1



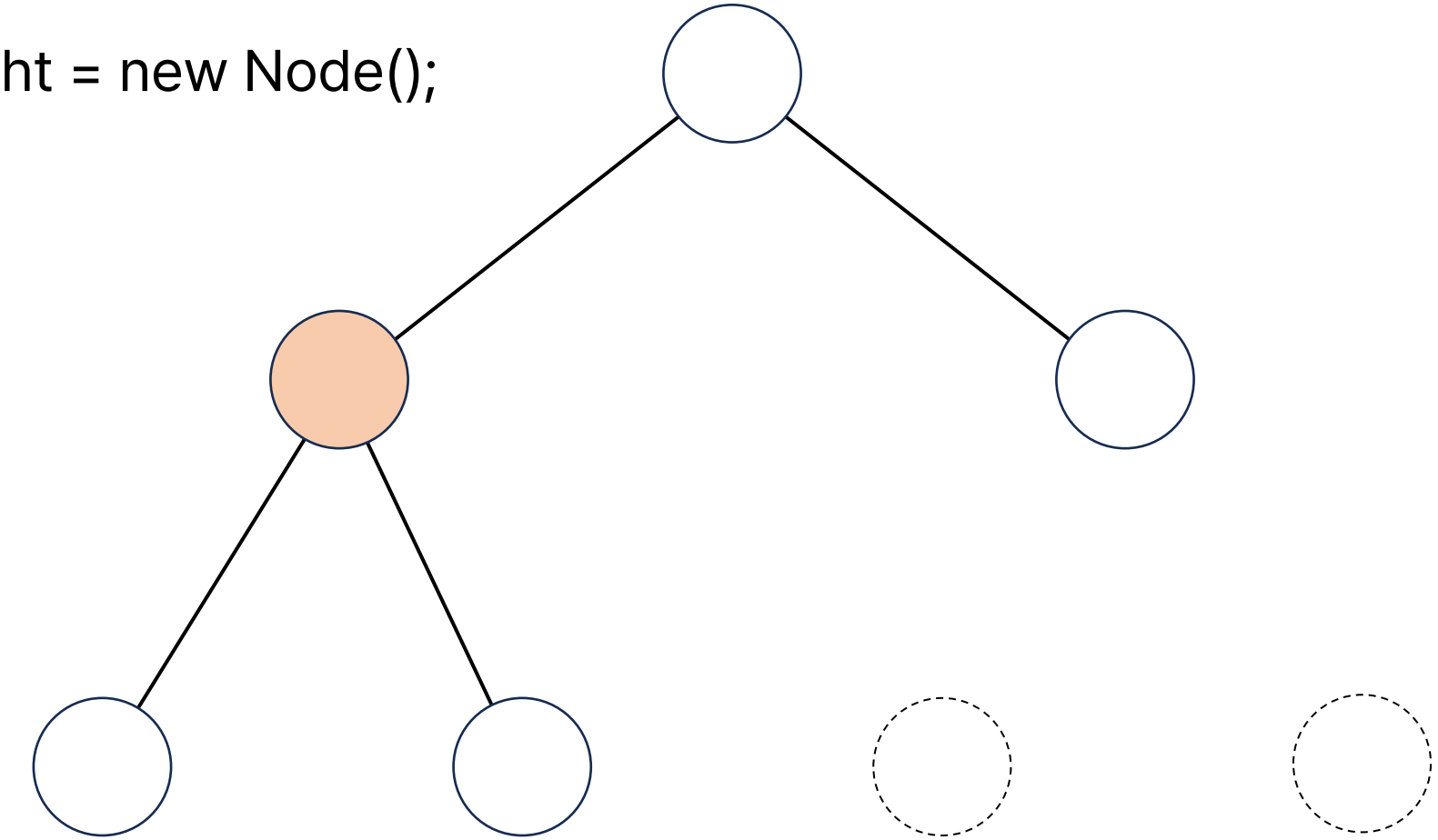
완전 이진 트리 BOJ 9934

- cur->left = new Node();



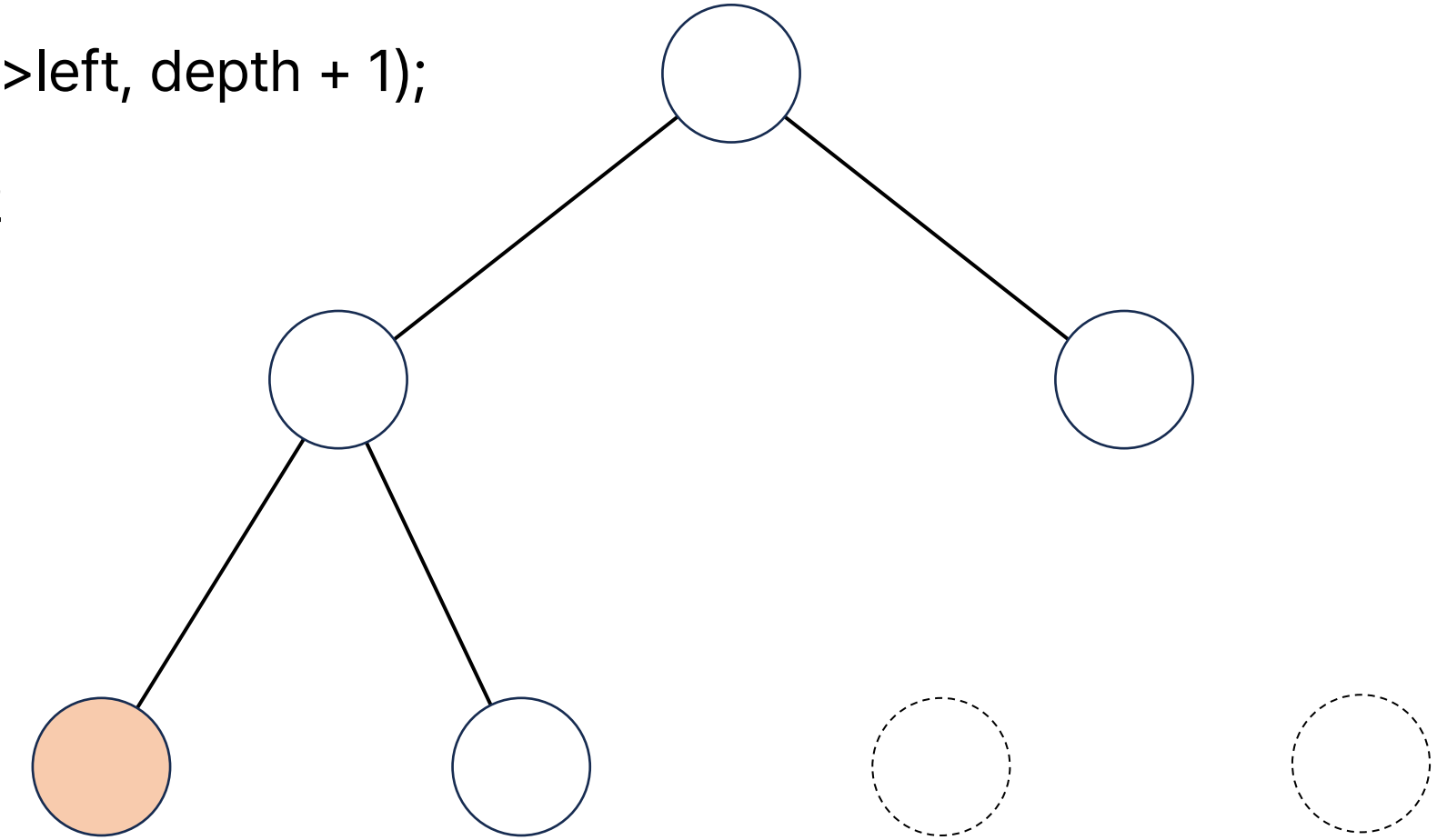
완전 이진 트리 BOJ 9934

- cur->right = new Node();



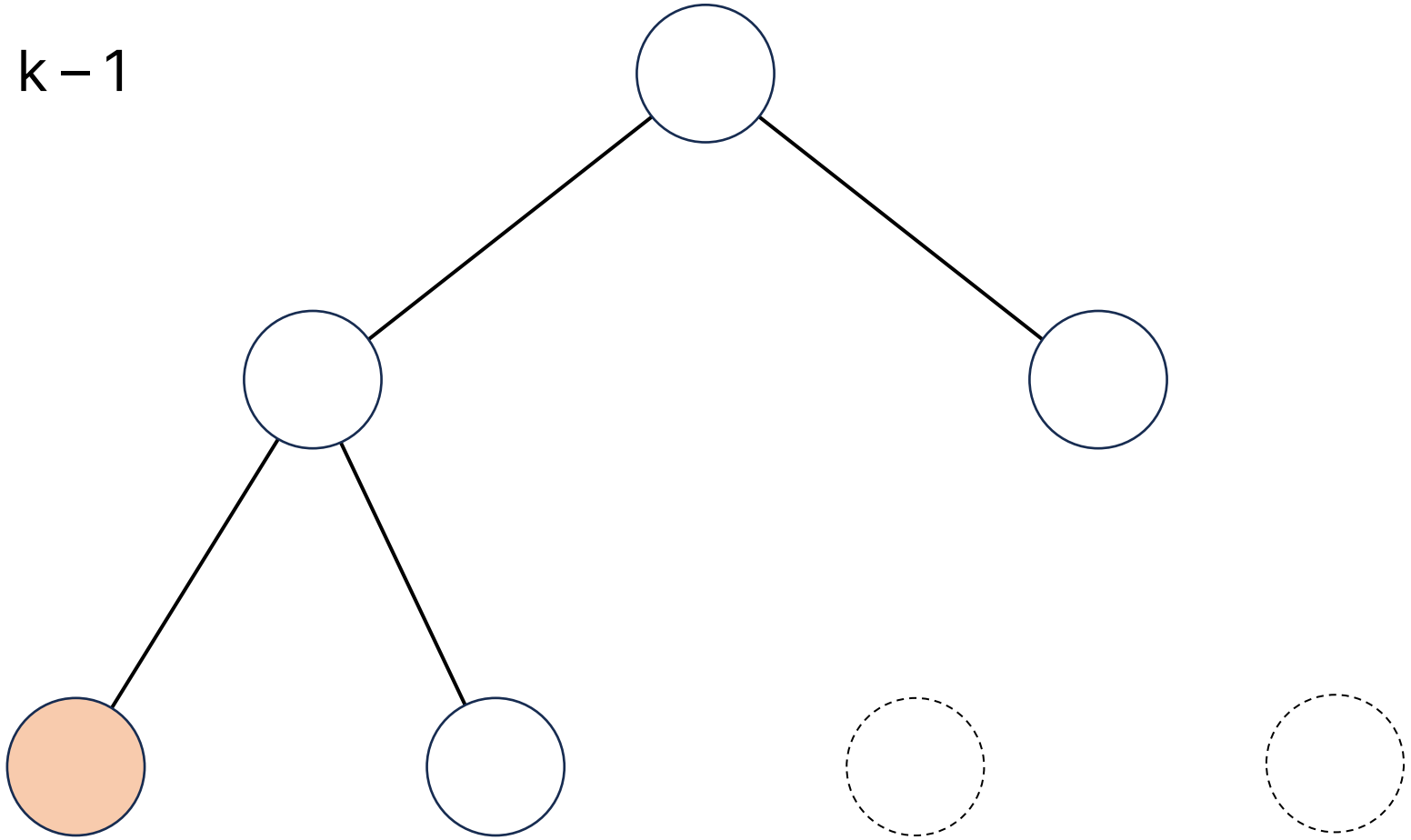
완전 이진 트리 BOJ 9934

- dfs(cur->left, depth + 1);
- depth: 2

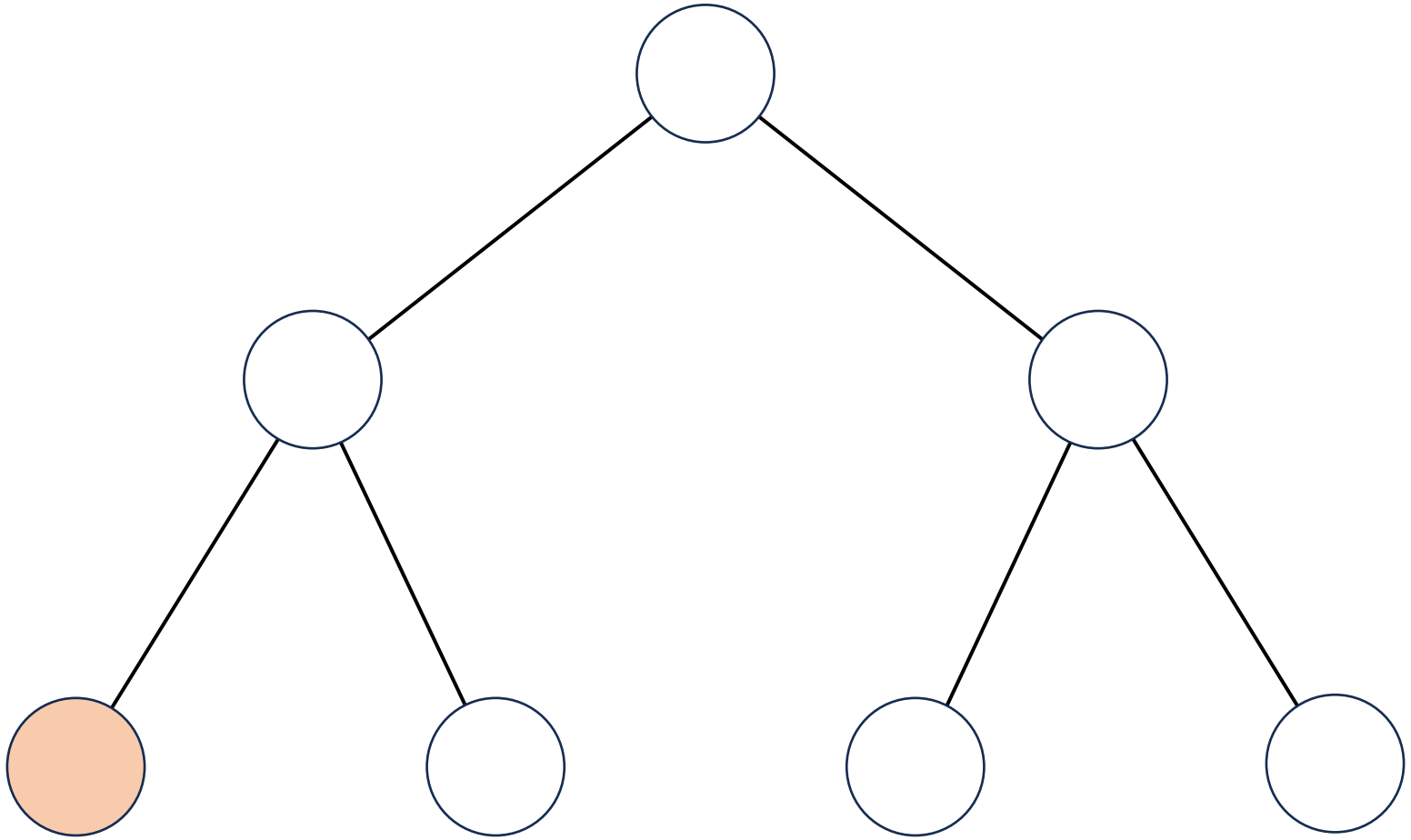


완전 이진 트리 BOJ 9934

- depth == k – 1
- return



완전 이진 트리 BOJ 9934



완전 이진 트리 BOJ 9934

- 트리를 만들었으니 순회를 하며 빌딩 번호를 설정

완전 이진 트리 BOJ 9934

1. 가장 처음에 상근이는 트리의 루트에 있는 빌딩 앞에 서있다.
2. 현재 빌딩의 왼쪽 자식에 있는 빌딩에 아직 들어가지 않았다면, 왼쪽 자식으로 이동한다.
3. 현재 있는 노드가 왼쪽 자식을 가지고 있지 않거나 왼쪽 자식에 있는 빌딩을 이미 들어갔다면, 현재 노드에 있는 빌딩을 들어가고 종이에 번호를 적는다.
4. 현재 빌딩을 이미 들어갔다 온 상태이고, 오른쪽 자식을 가지고 있는 경우에는 오른쪽 자식으로 이동한다.
5. 현재 빌딩과 왼쪽, 오른쪽 자식에 있는 빌딩을 모두 방문했다면, 부모 노드로 이동한다.

완전 이진 트리 BOJ 9934

1. 루트에서 시작
2. 왼쪽으로 이동
3. 본인을 방문
4. 오른쪽으로 이동
5. 종료

완전 이진 트리 BOJ 9934

- 중위 순회를 통하여 탐색했음을 알 수 있다
- 따라서 중위 순회를 구현하고, 중위 순회에 맞춰 빌딩 번호를 입력한다

완전 이진 트리 BOJ 9934

```
void in_order(Node *cur) {
    if (cur->left)
        in_order(cur->left);

    cin >> cur->num;

    if (cur->right)
        in_order(cur->right);
}
```

완전 이진 트리 BOJ 9934

- 출력을 위해 하나의 벡터로 각 값을 모은다
- 왼쪽부터 출력하므로 왼쪽의 값부터 벡터에 넣는다

완전 이진 트리 BOJ 9934

```
void dfs(Node *cur, int depth) {
    v[depth].push_back(cur->num);

    if (depth == k - 1)
        return;

    dfs(cur->left, depth + 1);
    dfs(cur->right, depth + 1);
}

for (int i = 0; i < k; i++) {
    for (auto a : v[i])
        cout << a << ' ‘;
    cout << '\n';
}
```

완전 이진 트리 BOJ 9934

- 또는 이 모든 것을 하나의 함수로 합칠 수 있다

완전 이진 트리 BOJ 9934

```
void tree_init(Node *cur, int depth) {
    if (depth != k - 1) {
        cur->left = new Node();
        tree_init(cur->left, depth + 1);
    }

    cin >> cur->num;
    v[depth].push_back(cur->num);

    if (depth != k - 1) {
        cur->right = new Node();
        tree_init(cur->right, depth + 1);
    }
}
```

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

- 입력으로 트리가 주어진다
- 루트 노드로부터 거리가 k이하인 노드에서 사과를 수확하려고 한다

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

- 루트 노드로부터 거리가 k이하이다
- 깊이가 k 이하인 노드들에서 사과를 모두 수확하면 된다

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

- 트리를 구성하자
- 부모와 자식 관계가 명시되어 있다

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

```
struct Node {
    int num;
    int apple;
    vector<int> child;
};

int n, k;
Node node[100000];
```

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

```
int p, c;
    cin >> n >> k;
for (int i = 0; i < n - 1; i++) {
    cin >> p >> c;
    node[p].child.push_back(c);
}

for (int i = 0; i < n; i++)
    cin >> node[i].apple;
```

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

- 루트 노드에서 순회를 시작해 깊이가 k 이하인 노드들에서 사과를 모두 합하면 된다

거리가 k이하인 트리 노드에서 사과 수확하기 BOJ 25516

```
int ans = 0;

void dfs(int cur, int depth) {
    if (depth > k)
        return;
    ans += node[cur].apple;
    for (auto next : node[cur].child)
        dfs(next, depth + 1);
}

dfs(0, 0);
cout << ans;
```

부동산 다툼 BOJ 20364

- 이진 트리가 주어진다
- 왼쪽 자식의 번호는 부모 번호의 2배 오른쪽 자식은 부모 번호의 2배 + 1이다
- 원하는 땅까지 가면서 점유된 땅이 존재한다면 제일 먼저 만나는 땅을 출력, 점유된 땅을 만나지 않고 원하는 땅까지 가는 경우 0을 출력하고 그 땅을 점유한다

부동산 다툼 BOJ 20364

- 원하는 땅으로 이동하기 위해 자식으로 이동해야한다
- 이동할 때 왼쪽 자식으로 이동할지 오른쪽 자식으로 이동할지 결정해야한다
- 모든 경우의 수를 다 따져보는 것은 비효율적이다

- 이진수로 표기하면 왼쪽 자식으로 이동해야 하는지 오른쪽 자식으로 이동해야 하는지 구분할 수 있다

부동산 다툼 BOJ 20364

- 2배가 되는 것은 2진수에서 왼쪽으로 한 칸 shift 하는 것이다
- 제일 오른쪽 비트은 0으로 채워진다
- 왼쪽 자식으로 이동하는 것은 왼쪽으로 한 칸 shift
- 오른쪽 자식으로 이동하는 것은 왼쪽으로 한 칸 shift한 후 제일 오른쪽 비트를 1로 채우는 것이다

부동산 다툼 BOJ 20364

1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1

부동산 다툼 BOJ 20364

3	0	0	0	0	0	0	1	1
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1

부동산 다툼 BOJ 20364

- 원하는 수가 53 이라면
- 제일 처음엔 루트에서 시작하므로 1이다

1	0	0	0	0	0	0	0	1

53	0	0	1	1	0	1	0	1

부동산 다툼 BOJ 20364

- 1을 shift해서 53을 만들어야 하므로 루트의 1은 53의 제일 앞 1이 된다

1	0	0	0	0	0	0	0	1

53	0	0	1	1	0	1	0	1

부동산 다툼 BOJ 20364

- 따라서 한 칸 내려갔을 때 두 칸이 53 제일 앞에 2칸과 동일해야 하므로 오른쪽 자식으로 이동해야 함을 알 수 있다

3	0	0	0	0	0	0	1	1

53	0	0	1	1	0	1	0	1

부동산 다툼 BOJ 20364

- 그 다음은 0이므로 왼쪽 자식으로 이동한다

6	0	0	0	0	0	1	1	0

53	0	0	1	1	0	1	0	1

부동산 다툼 BOJ 20364

- 이런 식으로 반복해서 내려가는 가면 방문해야하는 번호들을 알 수 있다
- 하지만 제일 앞에 있는 1부터 살펴보는 것은 코드가 복잡해진다

- 더 간단한 방법으로 방문할 번호들을 구할 수 있다

부동산 다툼 BOJ 20364

- 반대로 원하는 땅의 부모는 알기 쉽다
- 정수 나눗셈을 한다면 나머지가 버려진다
- 따라서 왼쪽 자식은 2배 오른쪽 자식은 2배 + 1 되지만 반대로 자식이 부모로 올라갈 때는 기존 번호에 1/2배를 하면 된다
- ex) 1의 왼쪽 자식은 1 * 2 = 2, 오른쪽 자식은 1 * 2 + 1 = 3이다.
- 반대로 2와 3 모두 부모로 올라갈 때는 1/2배 하면 부모를 구할 수 있다

부동산 다툼 BOJ 20364

- 왼쪽 자식과 오른쪽 자식으로 가는 것을 구분하며 내려가며 점유된 땅인지 확인할 수 있다
- 반대로 부모를 찾는 것은 매우 간단하므로 부모로 올라오면서 점유된 땅을 확인하며 제일 루트에서 가까운 땅을 출력하는 방법이 있다
- 따라서 부모로 올라오면서 점유된 땅들을 확인한다

부동산 다툼 BOJ 20364

- 땅을 점유하는지 확인할 visited 배열을 사용

```
int n, q;
bool visited[1 << 20];

cin >> n >> q;
```

부동산 다툼 BOJ 20364

- 루트에서 원하는 땅으로 가기 위해 들려야 하는 땅을 모두 살펴보면서 그 중 방문한 땅이 존재한다면 그 중 가장 루트와 가까운 땅을 저장한다

부동산 다툼 BOJ 20364

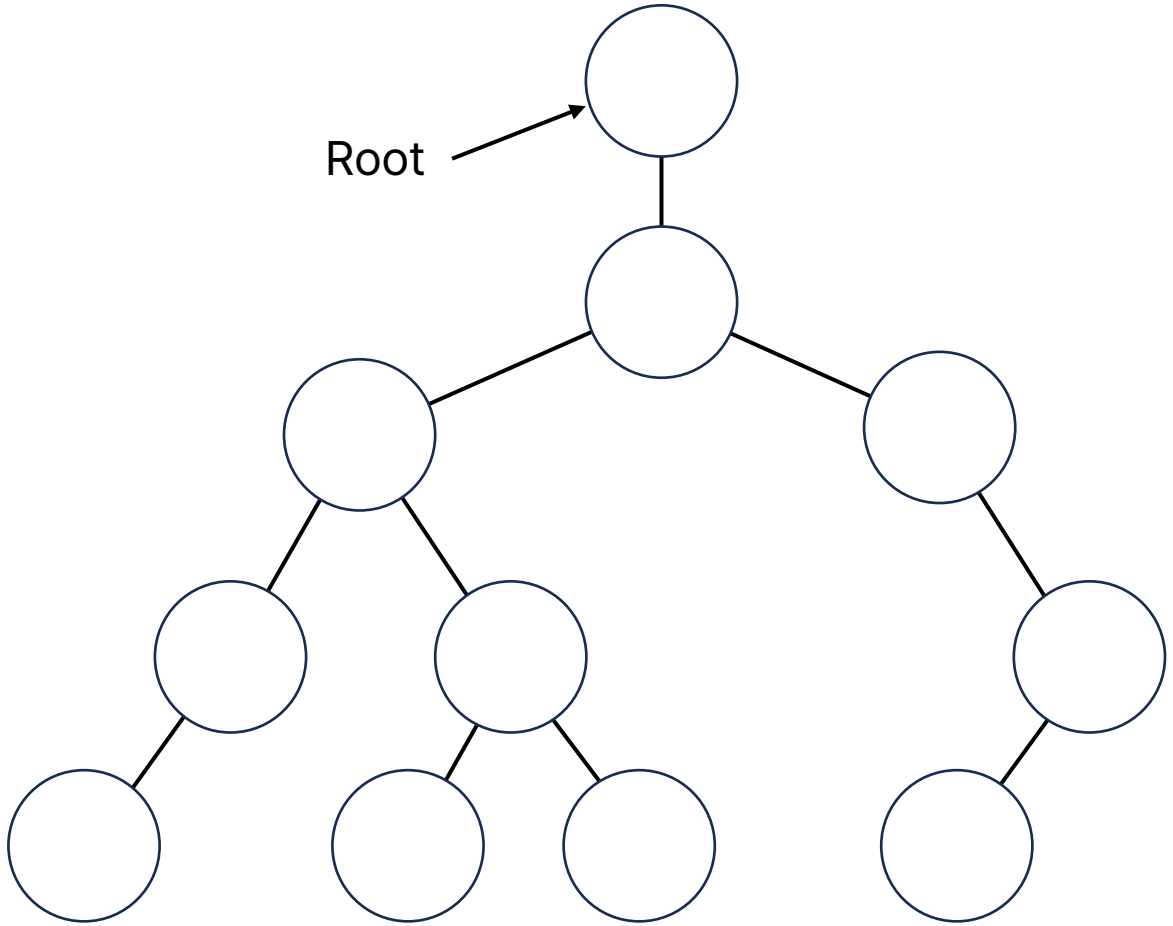
```
int x, tmp;
while (q--) {
    cin >> x;
    tmp = x;
    int ans = 0;
    while (x) {
        if (visited[x])
            ans = x;
        x >>= 1;
        // x /= 2;
}
    if (!ans)
        visited[tmp] = true;
    cout << ans << '\n';
}
```

LCA

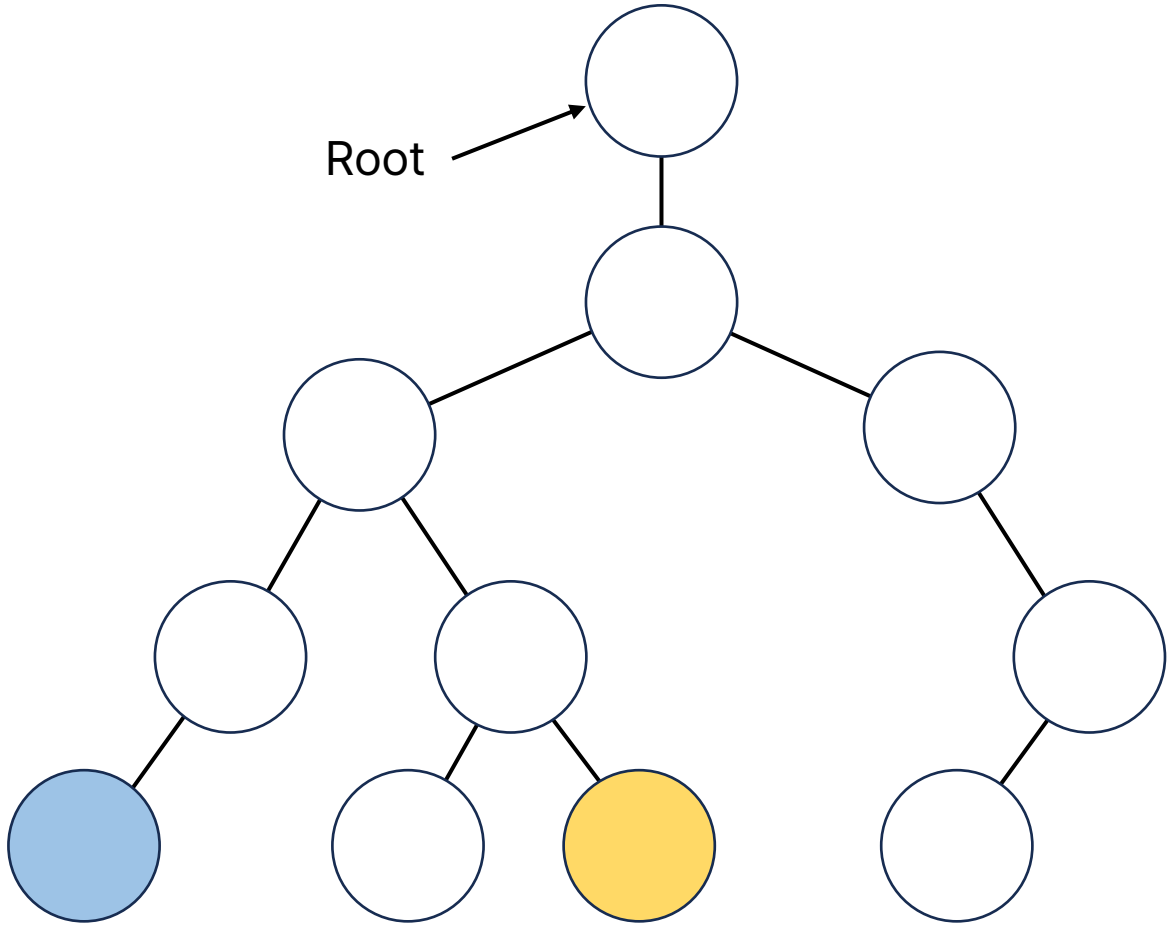
LCA

- Lowest Common Ancestor을 찾는 가장 기본적인 방법은 브루트 포스이다
- a 노드와 b 노드의 공통 조상 중
- 모든 노드를 탐색해야 하므로 노드의 개수가 N개 일 때, $O(N)$의 시간 복잡도를 따른다

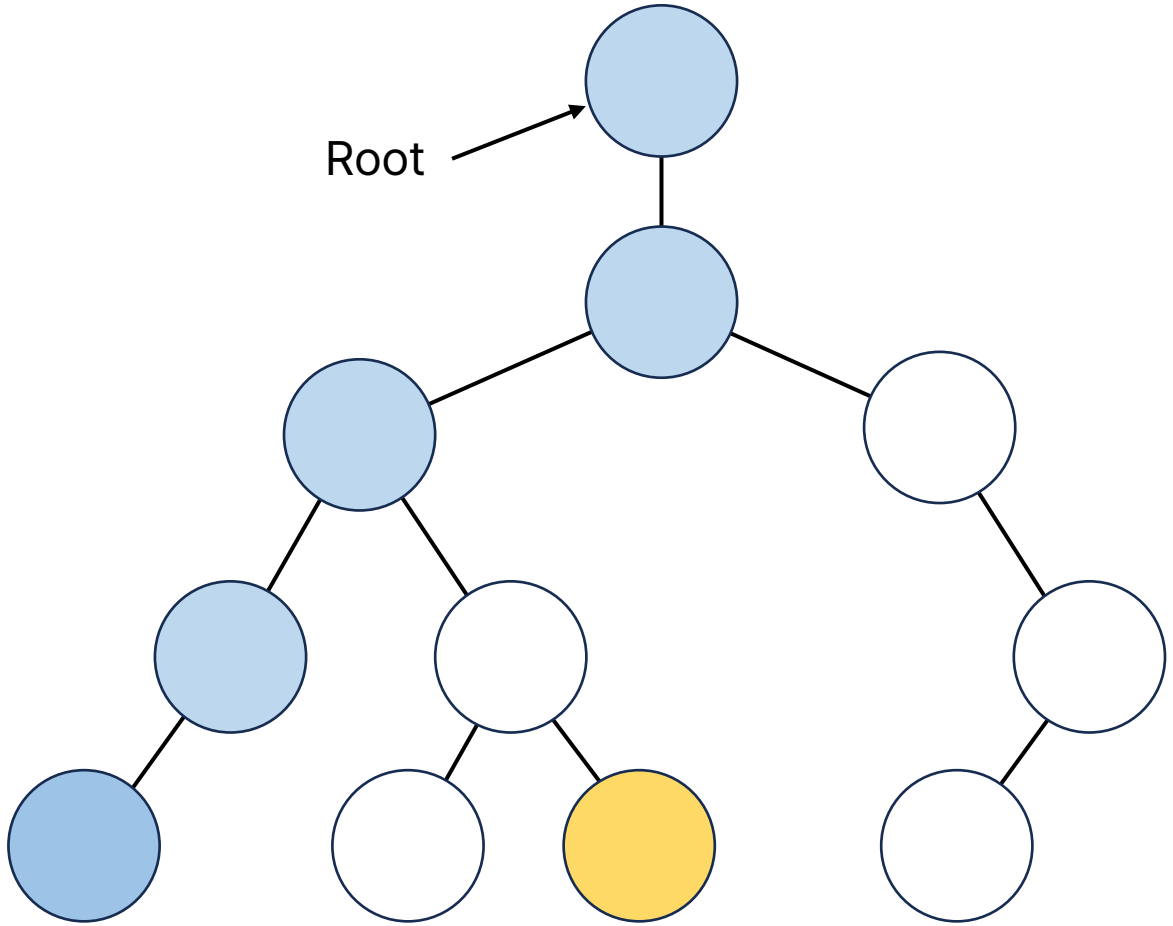
LCA



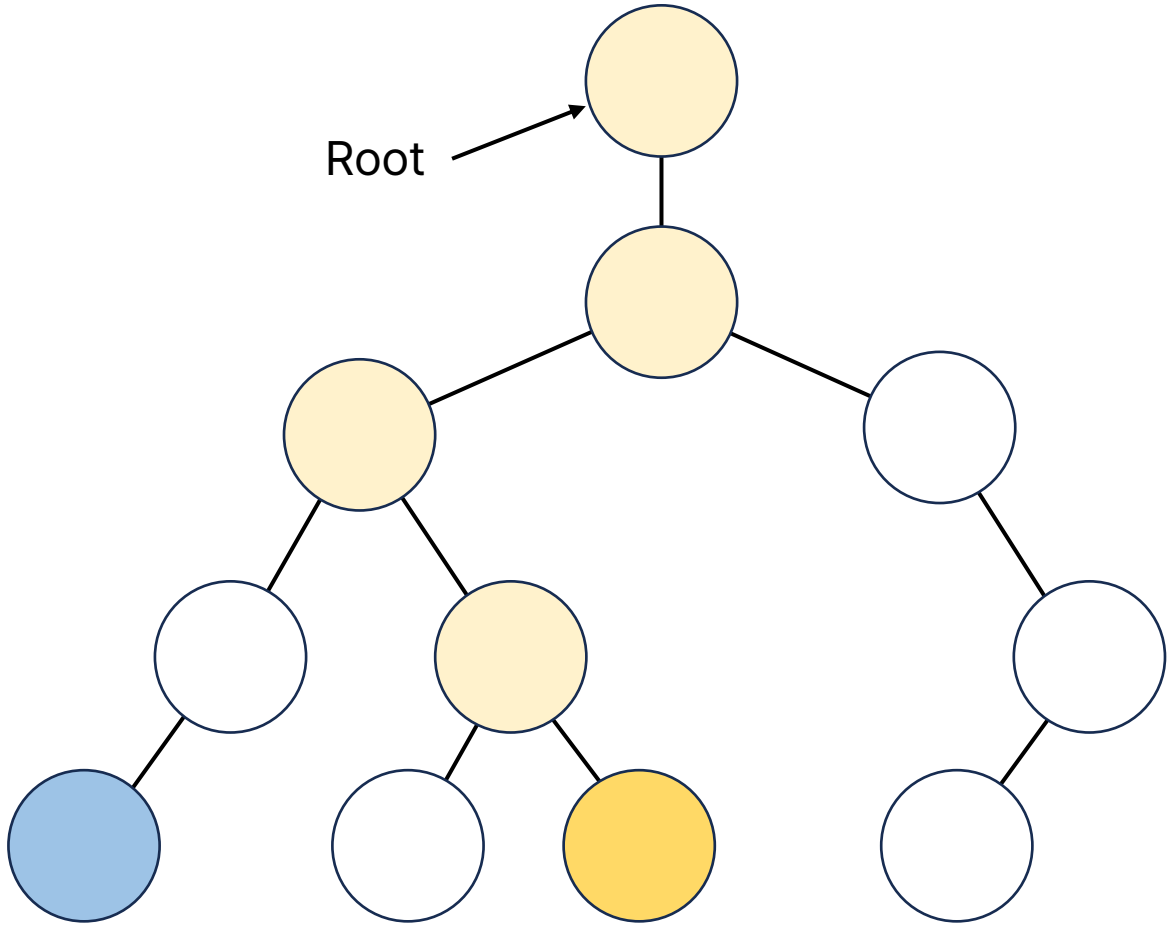
LCA



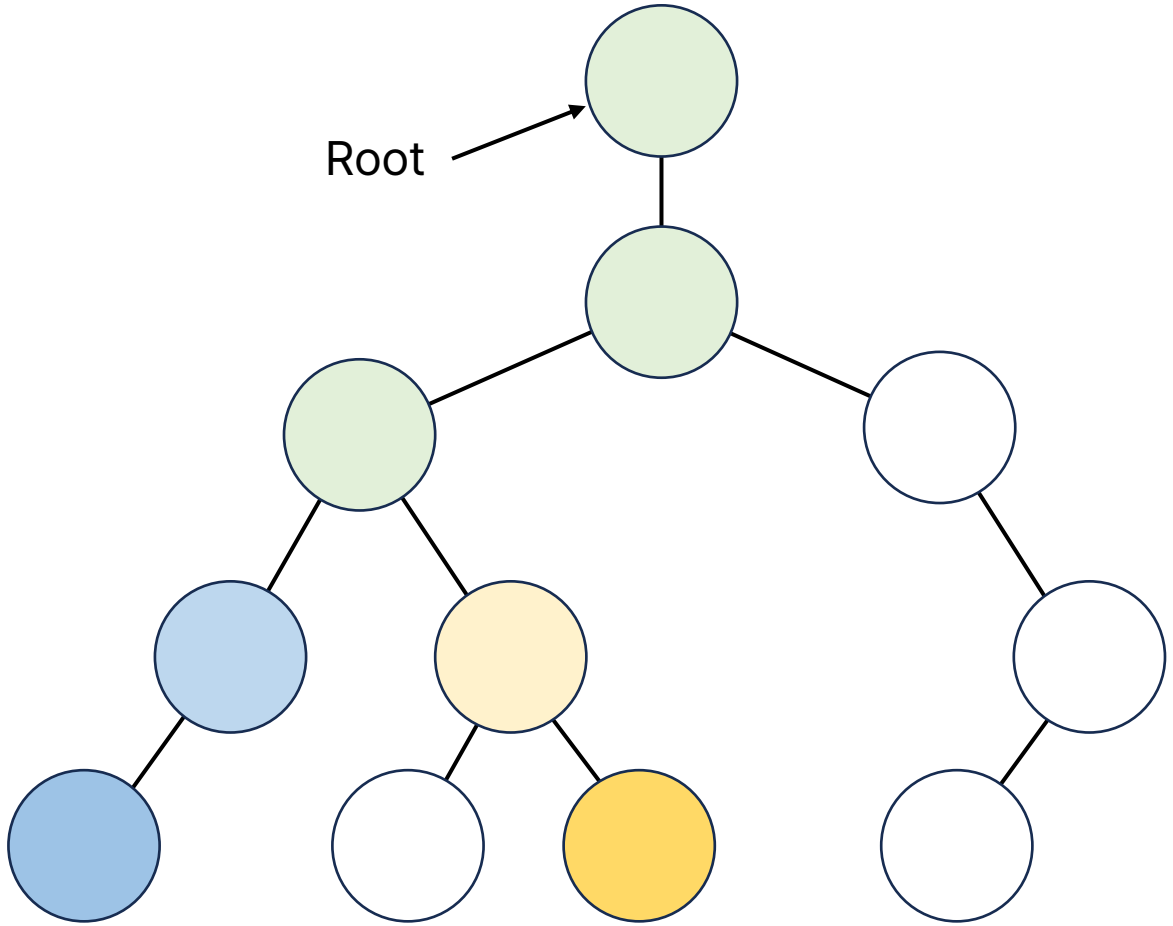
LCA



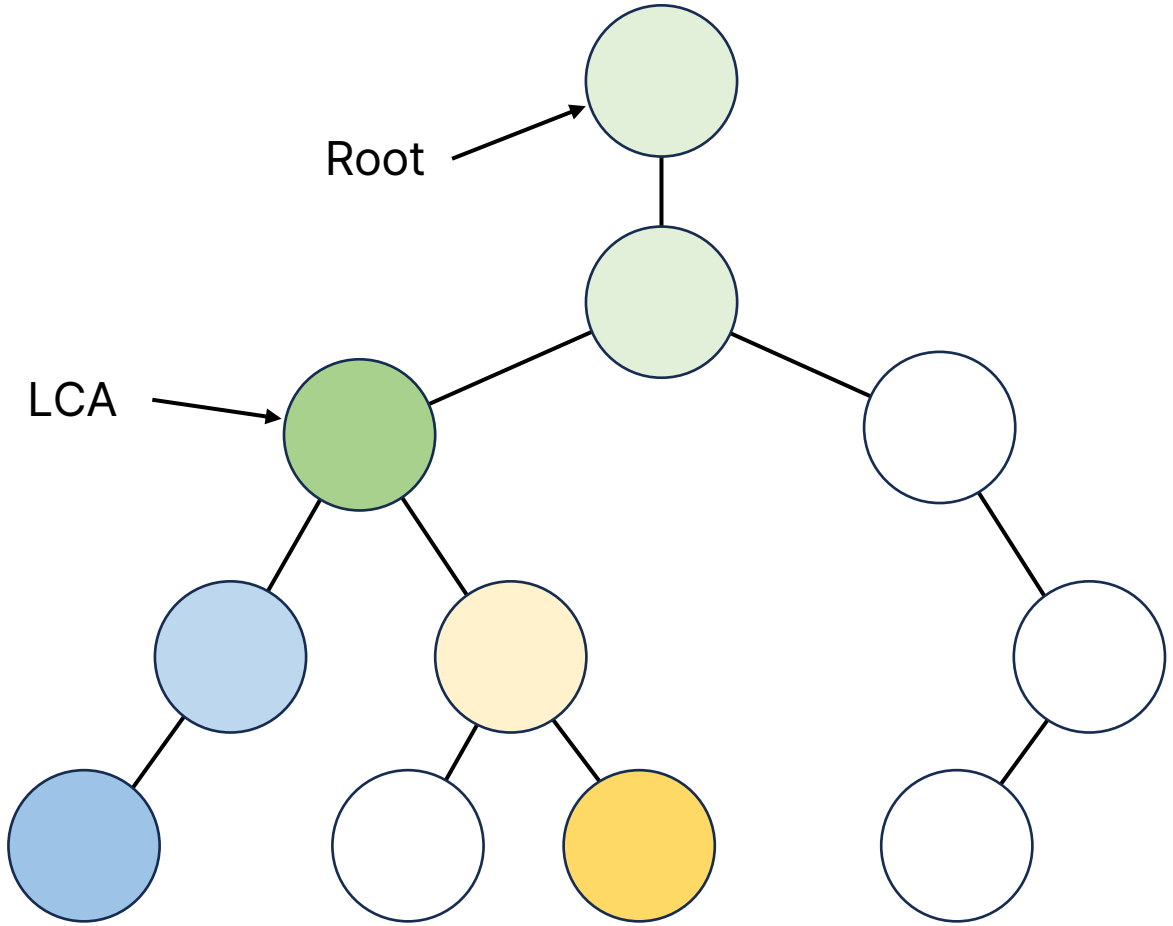
LCA



LCA



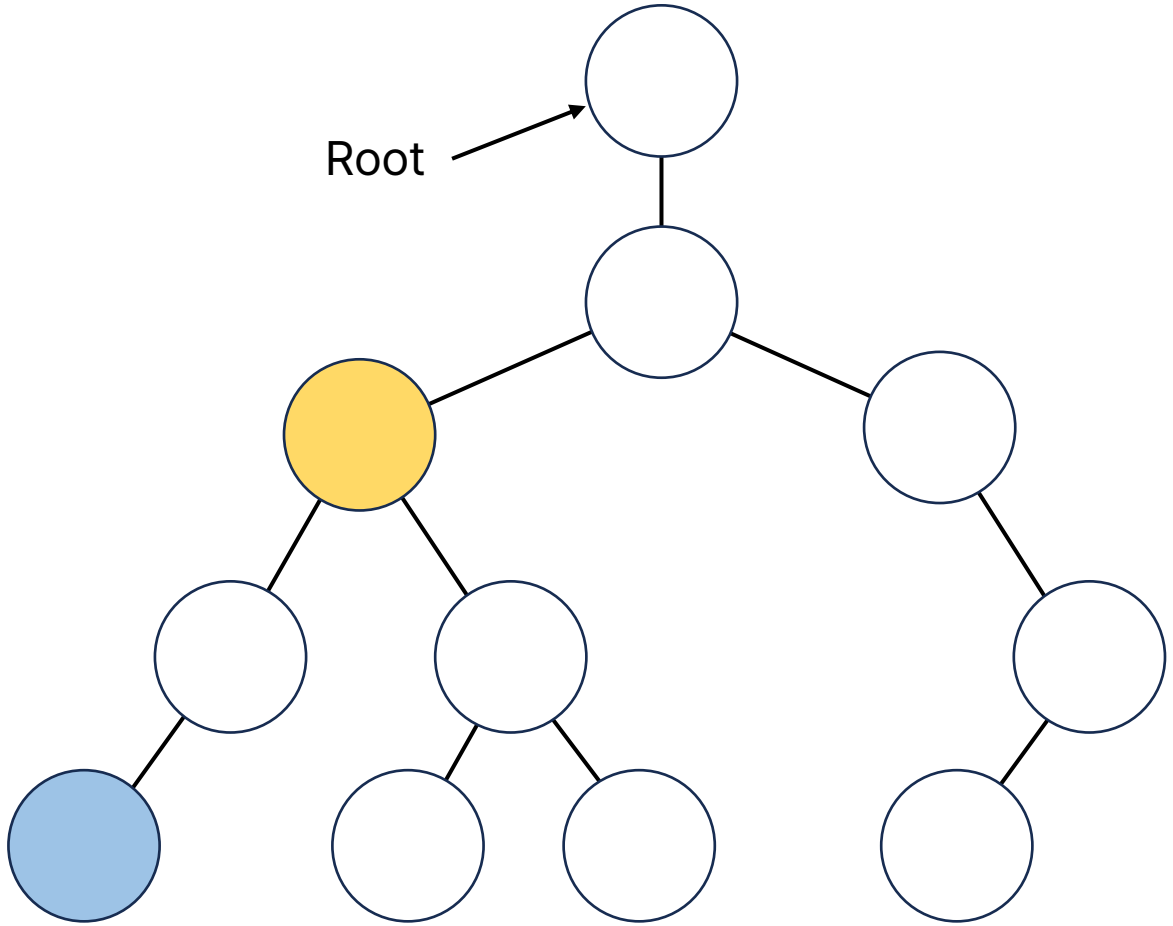
LCA



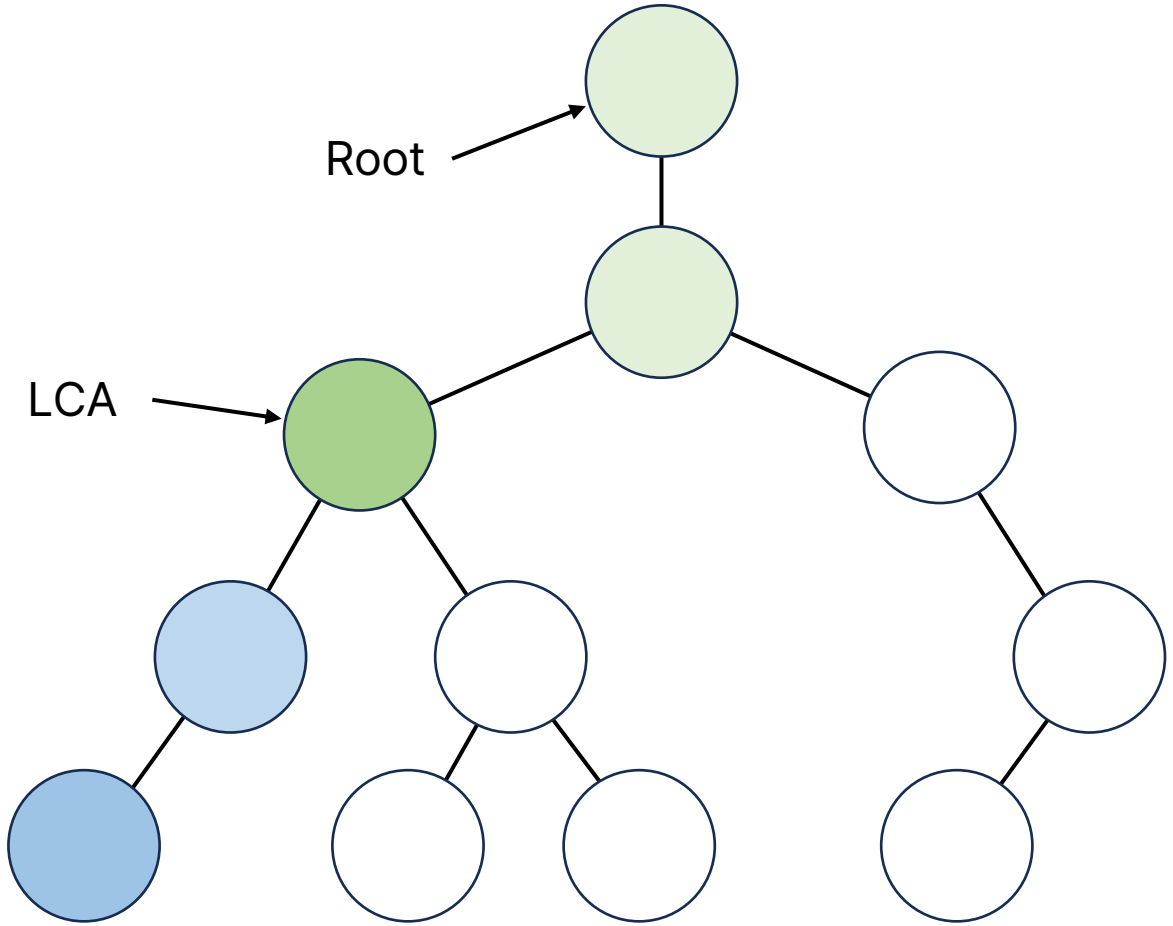
LCA

- 제일 기본적인 LCA 알고리즘을 만들어보자
- LCA를 찾는 첫 과정은 우선 두 노드를 동일한 높이까지 올리는 것이다
- 최소 공통 조상이므로 두 노드가 동일한 높이에 있지 않다면 두 노드 사이에는 최소 공통 조상이 존재하지 않는다

LCA



LCA



LCA

- 두 노드의 공통 조상이므로 LCA는 두 노드와 같은 높이에 있거나 더 높은 곳에 있다
- 만일 두 노드의 높이가 다르다면 둘 중 더 높이 있는 노드의 위치보다 높거나 같은 위치에 LCA노드가 존재하게 된다
- 따라서 높이가 같을 때까지 더 낮은 위치의 노드를 올려야한다

LCA

- 올리기 위해서는 부모 노드가 누구인지, 그리고 깊이는 몇인지 알아야 한다
- 부모와 자식 관계가 주어진 경우에도 깊이 정보를 알기 위해 순회를 해야한다
- 부모와 자식 관계가 주어지지 않은 경우에는 한 번의 순회 동안 두 정보를 모두 알아낼 수 있다

LCA

```
int depth[MAX_NODE], p[MAX_NODE];
vector<int> childs[MAX_NODE];
int parent, child;

for (int i = 0; i < MAX_NODE - 1; i++) {
    cin >> parent >> child;
    childs[parent].push_back(child);
    p[child] = parent;
}

dfs(ROOT_NODE);
```

LCA

```
void dfs(int cur) {
    for (auto child : childs[cur]) {
        depth[child] = depth[cur] + 1;
        dfs(child);
    }
}
```

LCA

```
int depth[MAX_NODE], p[MAX_NODE];
vector<int> edges[MAX_NODE];
int src, dst;
// depth가 정해지지 않은 경우 방문하지 않았음을 알 수 있다
// visited를 사용하지 않아도 된다

for (int i = 1; i <= MAX_NODE; i++)
    depth[i] = -1;

for (int i = 0; i < MAX_NODE - 1; i++) {
    cin >> src >> dst;
    edges[src].push_back(dst);
    edges[dst].push_back(src);
}
```

LCA

```
void dfs(int cur) {
    for (auto child : edges[cur]) {
        if (depth[child] != -1)
            continue;
        depth[child] = depth[cur] + 1;
        p[child] = cur;
        dfs(child);
    }
}
```

LCA

- DFS를 통해 깊이와 부모를 모두 알아냈다
- 그 다음은 깊이가 낮은 노드를 더 높은 위치로 올리는 것이다

LCA

```
int a, b;
// LCA를 구할 두 노드

cin >> a >> b;
if (depth[a] > depth[b]) {
    while (dep[a] > dep[b]) {
        a = p[a];
    }
}
if (dep[a] < dep[b]) {
    while (dep[a] < dep[b]) {
        b = p[b];
    }
}
```

LCA

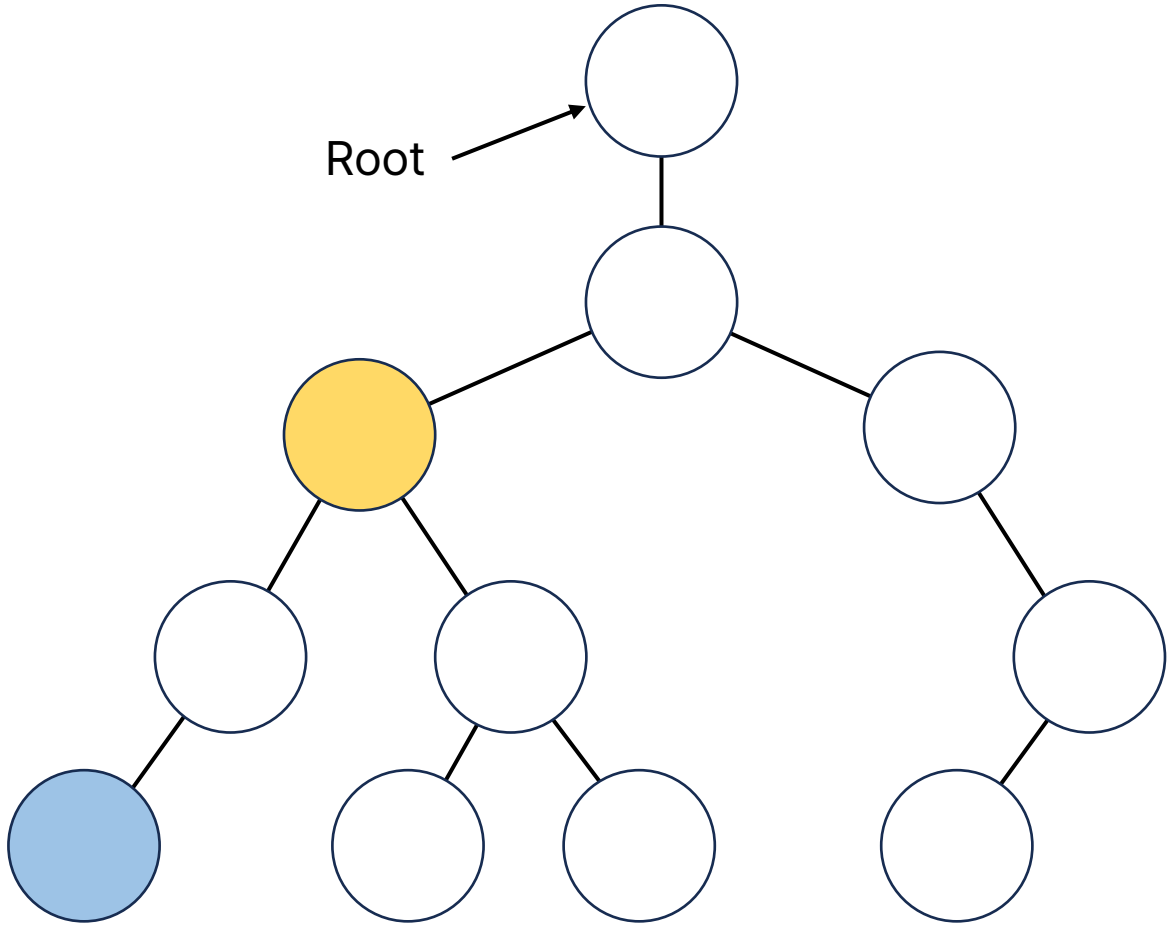
```
int a, b;
// LCA를 구할 두 노드

cin >> a >> b;
// a에 더 낮은 위치의 노드를 넣는다
if (dep[a] < dep[b])
    swap(a, b);
while (dep[a] != dep[b])
    a = p[a];
```

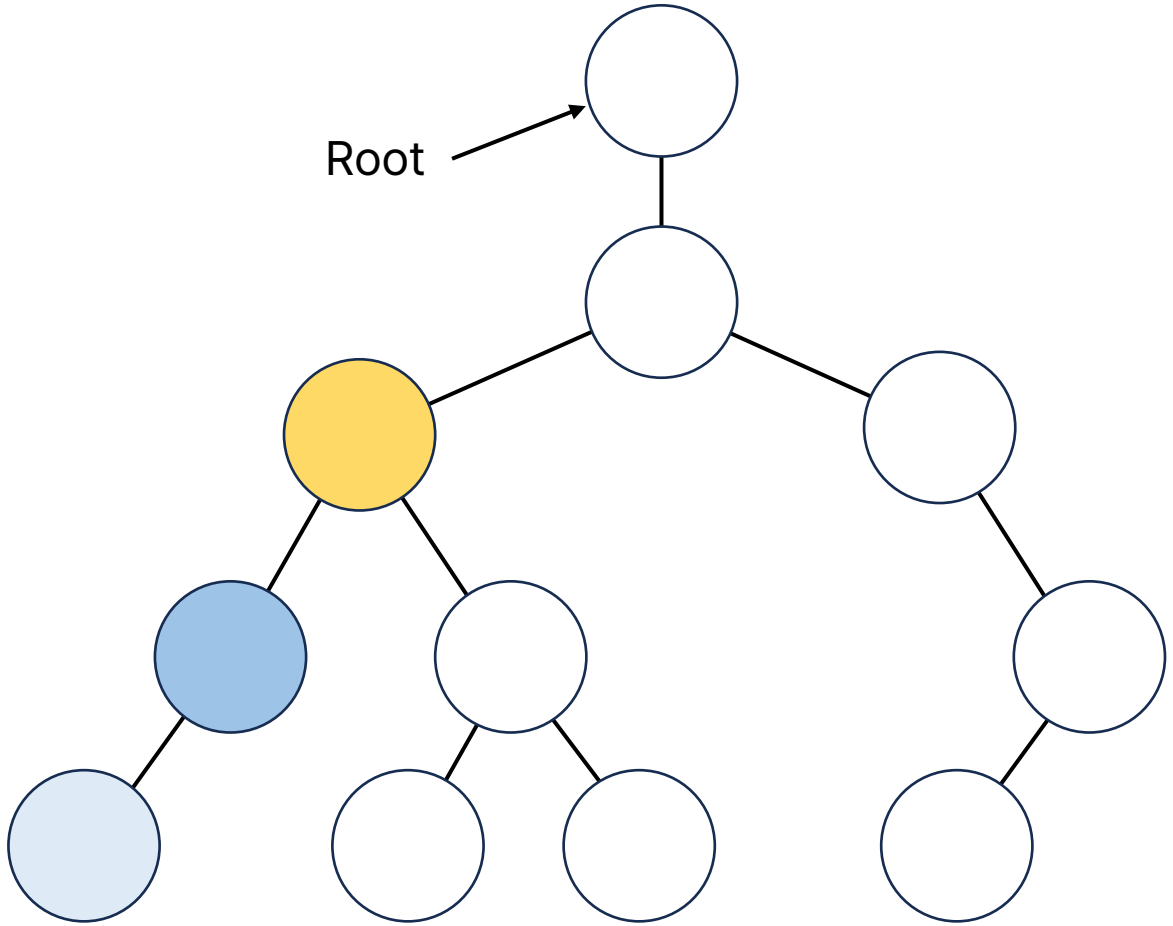
LCA

- 두 노드를 같은 높이까지 올렸을 때, 두 노드가 동일한 노드라면 해당 노드가 LCA이다
- 동일한 노드가 아니라면 동일한 노드가 될 때까지 올린다

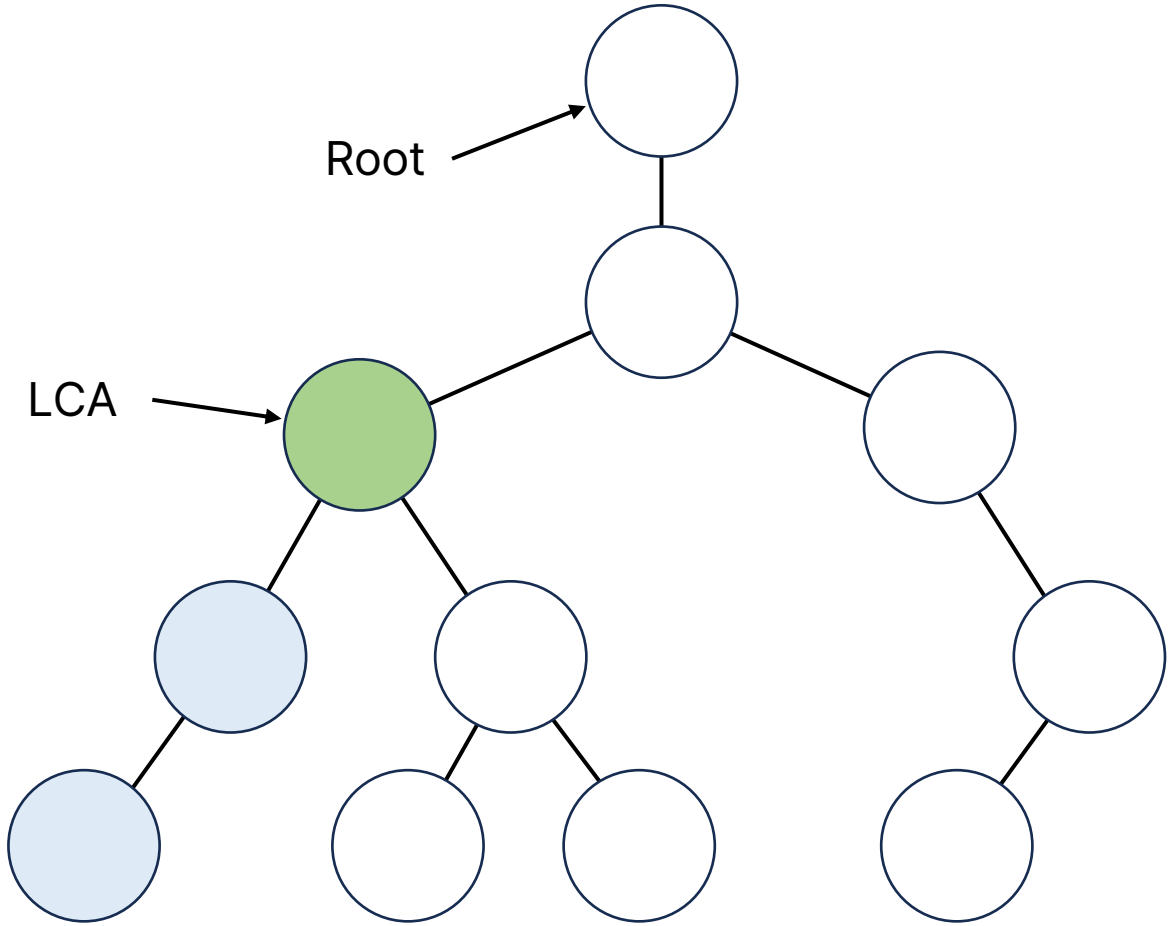
LCA



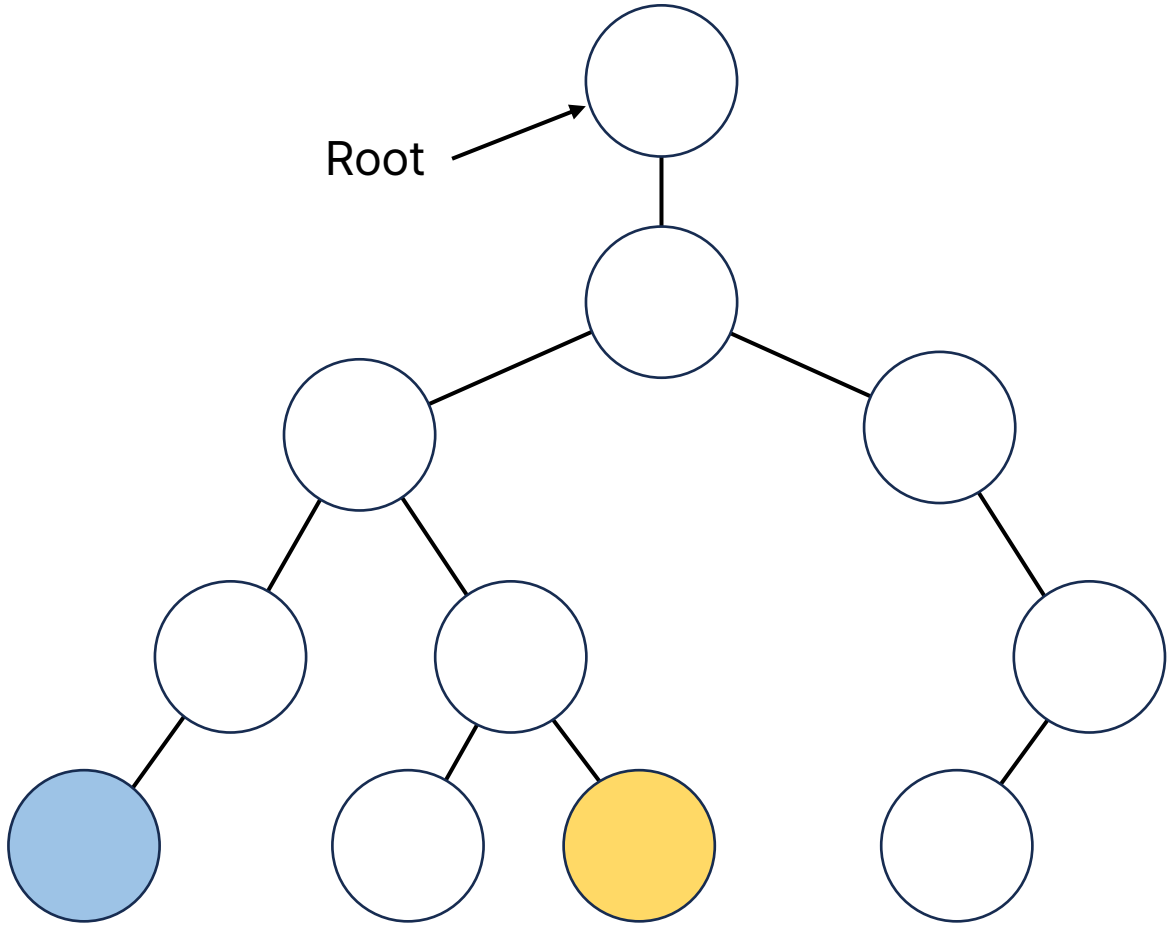
LCA



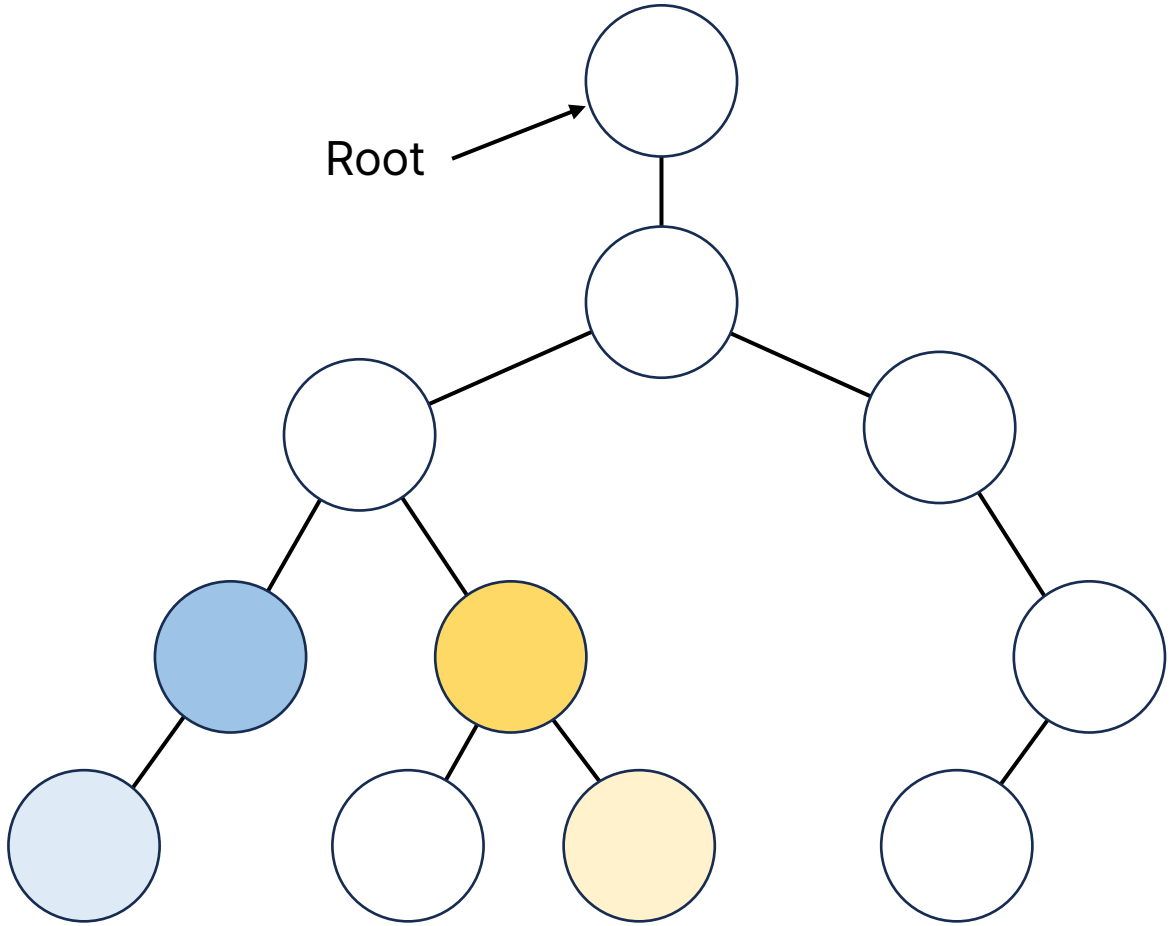
LCA



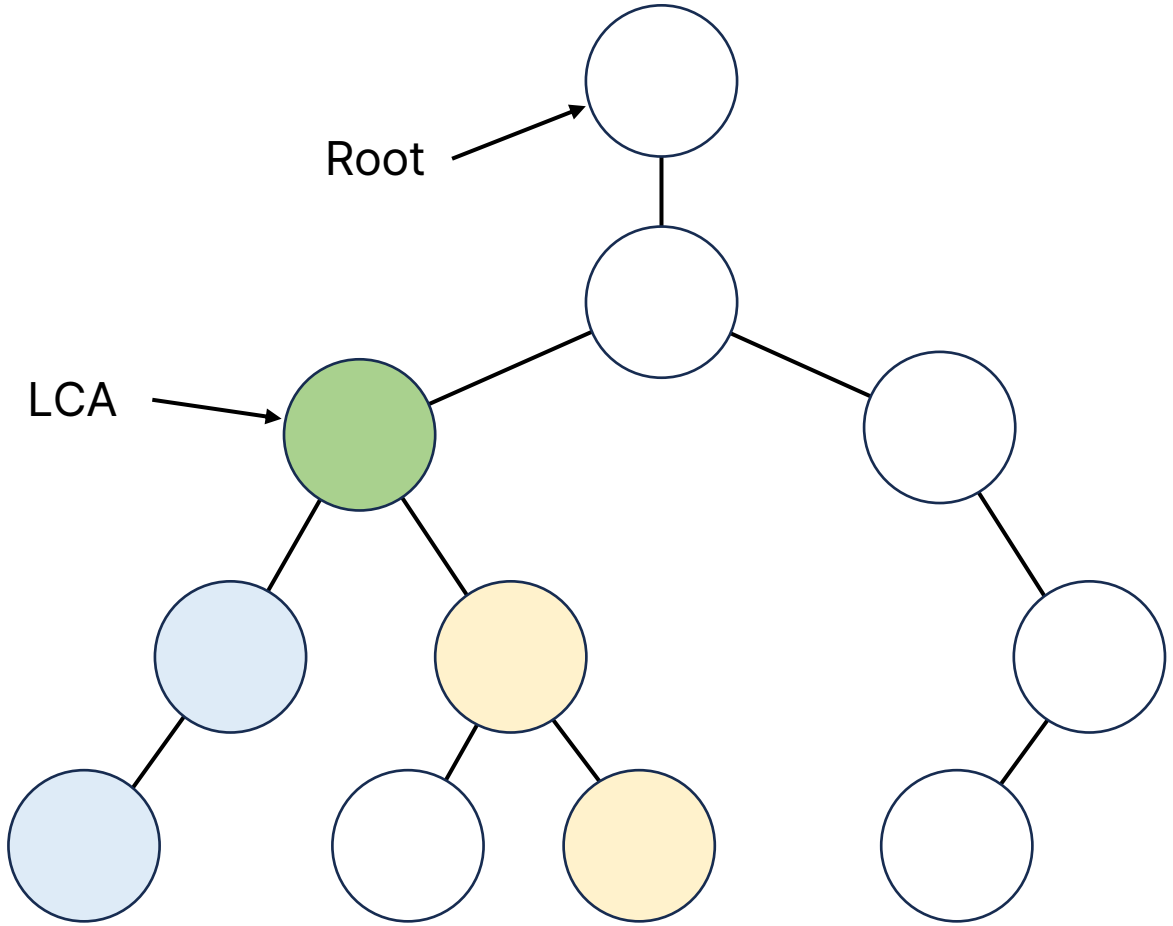
LCA



LCA



LCA



LCA

```
while (a != b) {
    a = p[a];
    b = p[b];
}
cout << a << '\n'; // a = b = LCA
```

문제

- 가장 가까운 공통 조상 BOJ 3548
- LCA BOJ 11437