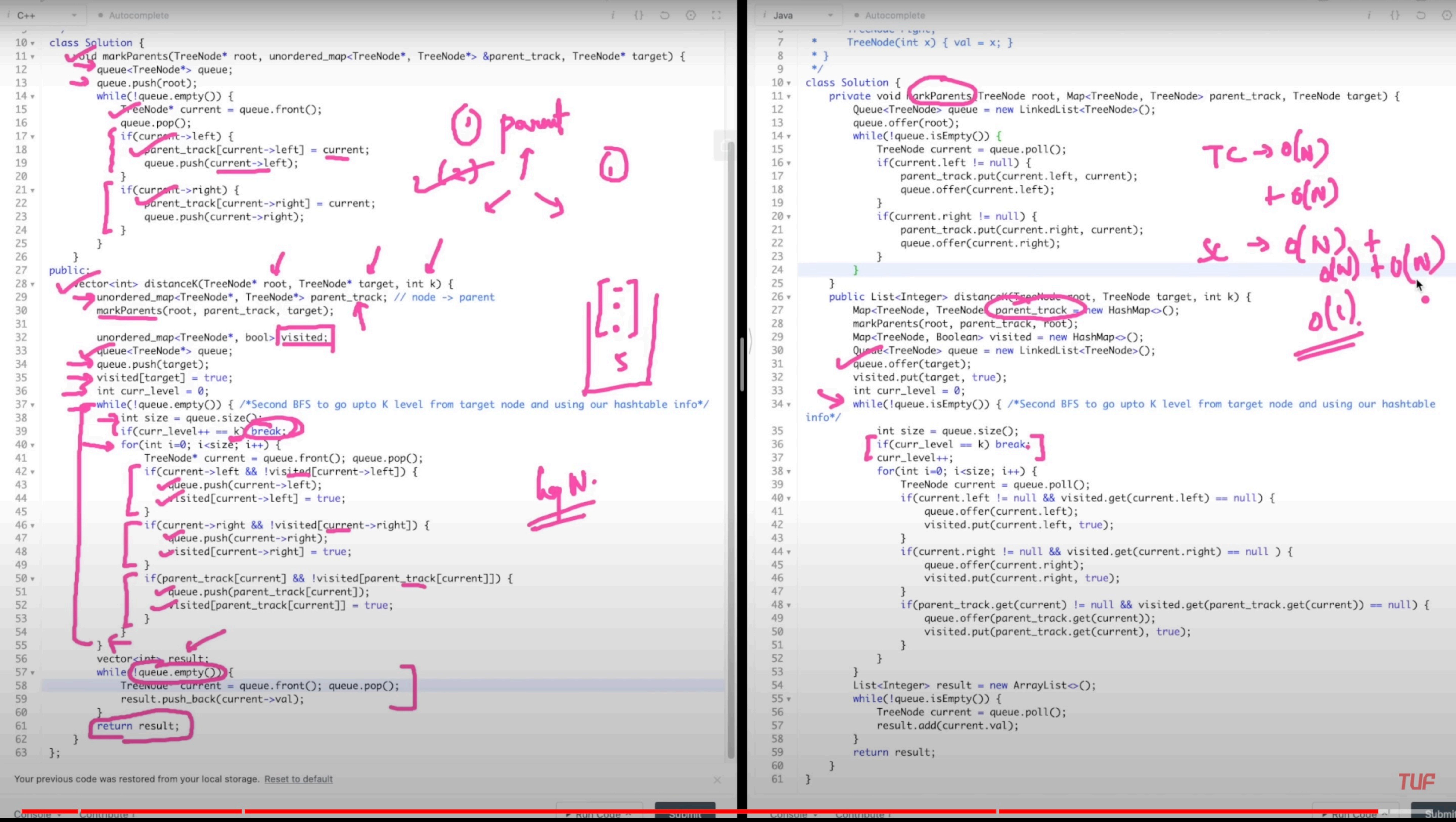
L30. Print all the Nodes at a distance of K in Binary Tree | C++ | Java



```cpp
class Solution {
    void markParents(TreeNode* root, unordered_map<TreeNode*, TreeNode*> &parent_track, TreeNode* target) {
        queue<TreeNode*> queue;
        queue.push(root);
        while(!queue.empty()) {
            TreeNode* current = queue.front();
            queue.pop();
            if(current->left) {
                parent_track[current->left] = current;
                queue.push(current->left);
            }
            if(current->right) {
                parent_track[current->right] = current;
                queue.push(current->right);
            }
        }
    }
public:
    vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
        unordered_map<TreeNode*, TreeNode*> parent_track; // node -> parent
        markParents(root, parent_track, target);

        unordered_map<TreeNode*, bool> visited;
        queue<TreeNode*> queue;
        queue.push(target);
        visited[target] = true;
        int curr_level = 0;
        while(!queue.empty()) { /*Second BFS to go upto K level from target node and using our hashtable info*/
            int size = queue.size();
            if(curr_level++ == k) break;
            for(int i=0; i<size; i++) {
                TreeNode* current = queue.front(); queue.pop();
                if(current->left && !visited[current->left]) {
                    queue.push(current->left);
                    visited[current->left] = true;
                }
                if(current->right && !visited[current->right]) {
                    queue.push(current->right);
                    visited[current->right] = true;
                }
                if(parent_track[current] && !visited[parent_track[current]]) {
                    queue.push(parent_track[current]);
                    visited[parent_track[current]] = true;
                }
            }
        }
        vector<int> result;
        while(!queue.empty()) {
            TreeNode* current = queue.front(); queue.pop();
            result.push_back(current->val);
        }
        return result;
    }
};
```

Your previous code was restored from your local storage. Reset to default

```java
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    private void markParents(TreeNode root, Map<TreeNode, TreeNode> parent_track, TreeNode target) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(root);
        while(!queue.isEmpty()) {
            TreeNode current = queue.poll();
            if(current.left != null) {
                parent_track.put(current.left, current);
                queue.offer(current.left);
            }
            if(current.right != null) {
                parent_track.put(current.right, current);
                queue.offer(current.right);
            }
        }
    }
    public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
        Map<TreeNode, TreeNode> parent_track = new HashMap<>();
        markParents(root, parent_track, root);
        Map<TreeNode, Boolean> visited = new HashMap<>();
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(target);
        visited.put(target, true);
        int curr_level = 0;
        while(!queue.isEmpty()) { /*Second BFS to go upto K level from target node and using our hashtable info*/
            int size = queue.size();
            if(curr_level == k) break;
            curr_level++;
            for(int i=0; i<size; i++) {
                TreeNode current = queue.poll();
                if(current.left != null && visited.get(current.left) == null) {
                    queue.offer(current.left);
                    visited.put(current.left, true);
                }
                if(current.right != null && visited.get(current.right) == null ) {
                    queue.offer(current.right);
                    visited.put(current.right, true);
                }
                if(parent_track.get(current) != null && visited.get(parent_track.get(current)) == null) {
                    queue.offer(parent_track.get(current));
                    visited.put(parent_track.get(current), true);
                }
            }
        }
        List<Integer> result = new ArrayList<>();
        while(!queue.isEmpty()) {
            TreeNode current = queue.poll();
            result.add(current.val);
        }
        return result;
    }
}
```

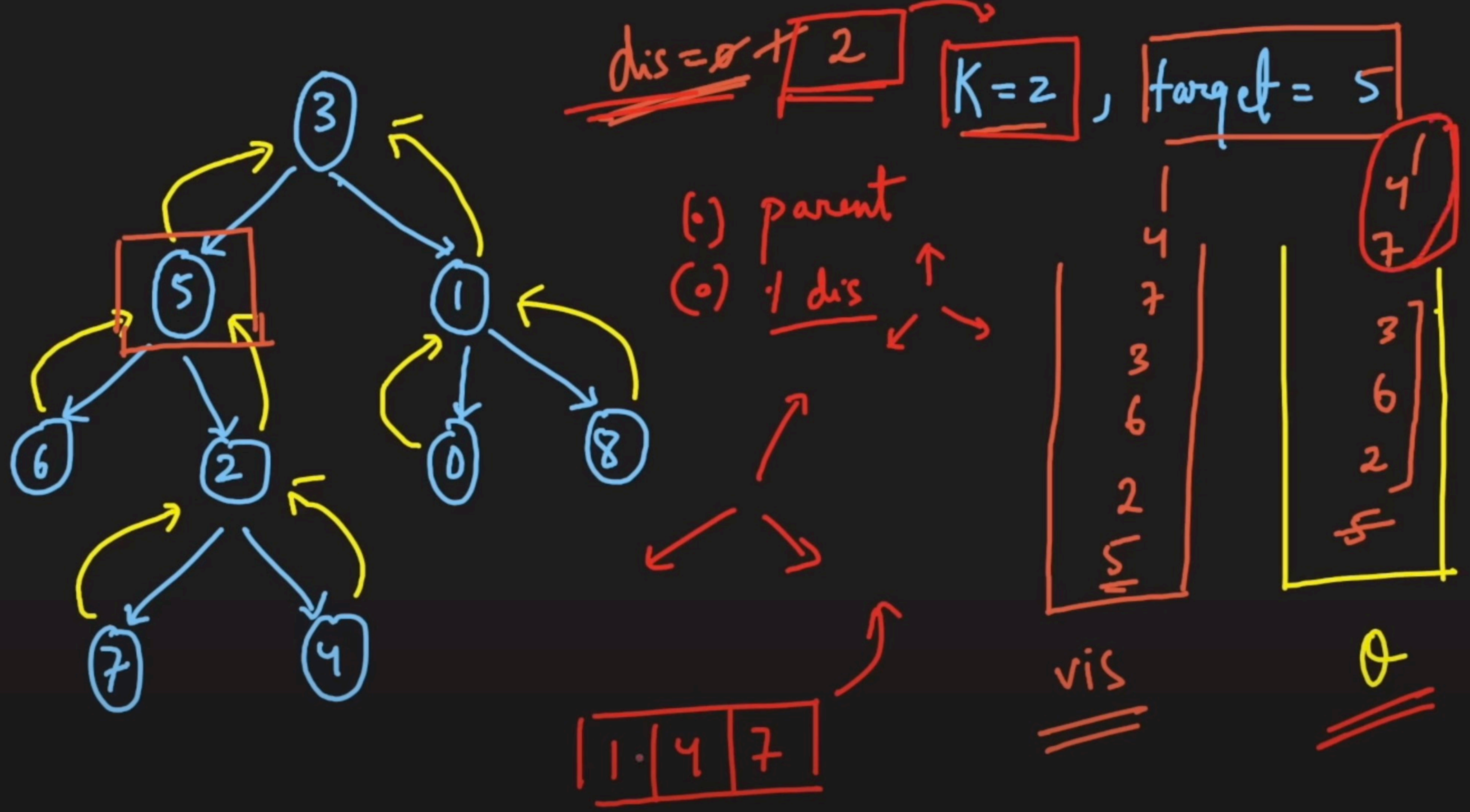
TC → O(N) + O(N)

SC → O(N) + O(N) + O(N)

O(1).

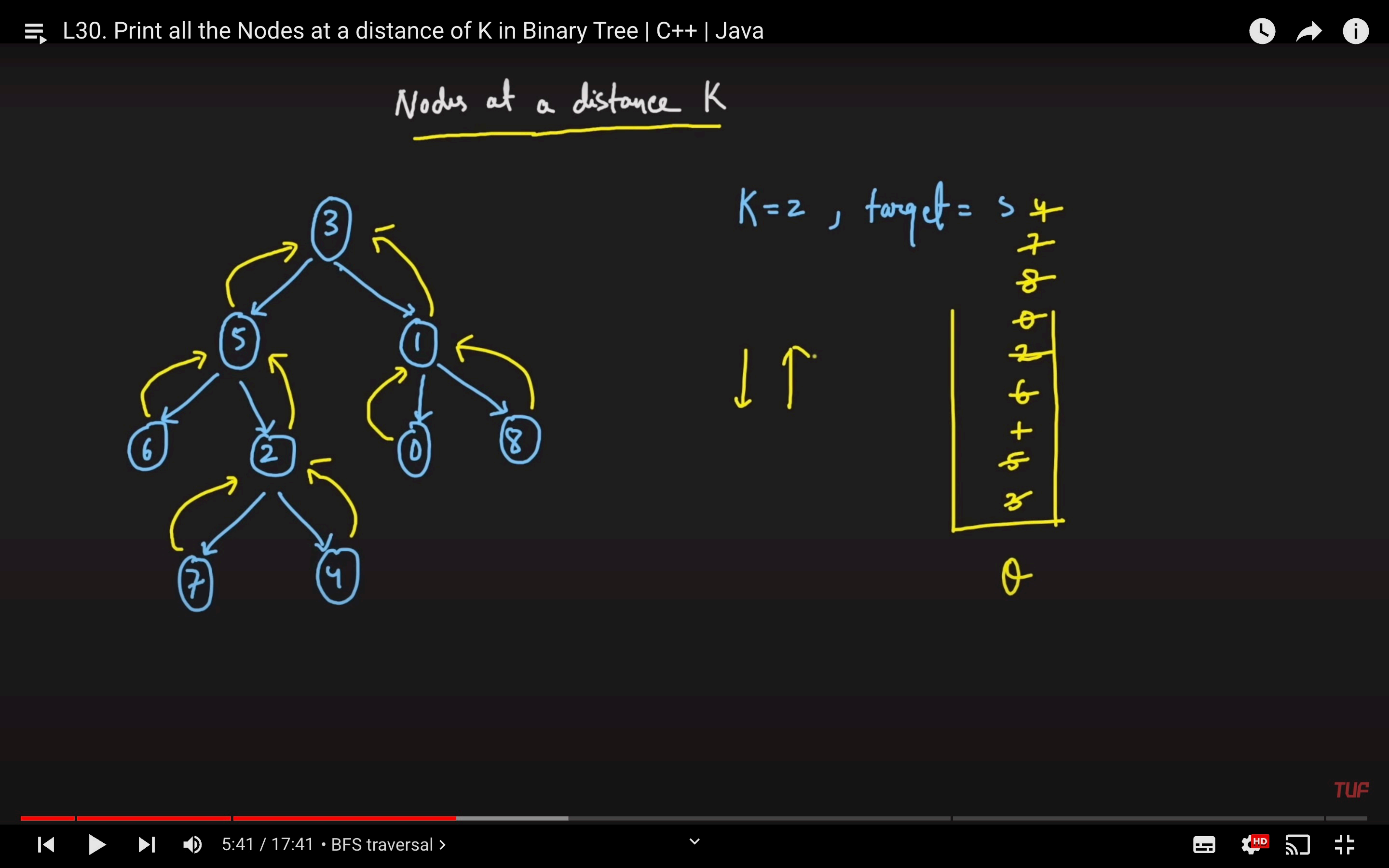
L30. Print all the Nodes at a distance of K in Binary Tree | C++ | Java

Nodes at a distance K



TUF

12:10 / 17:41 • BFS traversal ›



L30. Print all the Nodes at a distance of K in Binary Tree | C++ | Java

Nodes at a distance K

K=2 , target = 5 ~~4~~

~~7~~

~~8~~

~~0~~

~~2~~

~~6~~

~~1~~

~~5~~

~~3~~

0

↓ ↑

5:41 / 17:41 • BFS traversal >