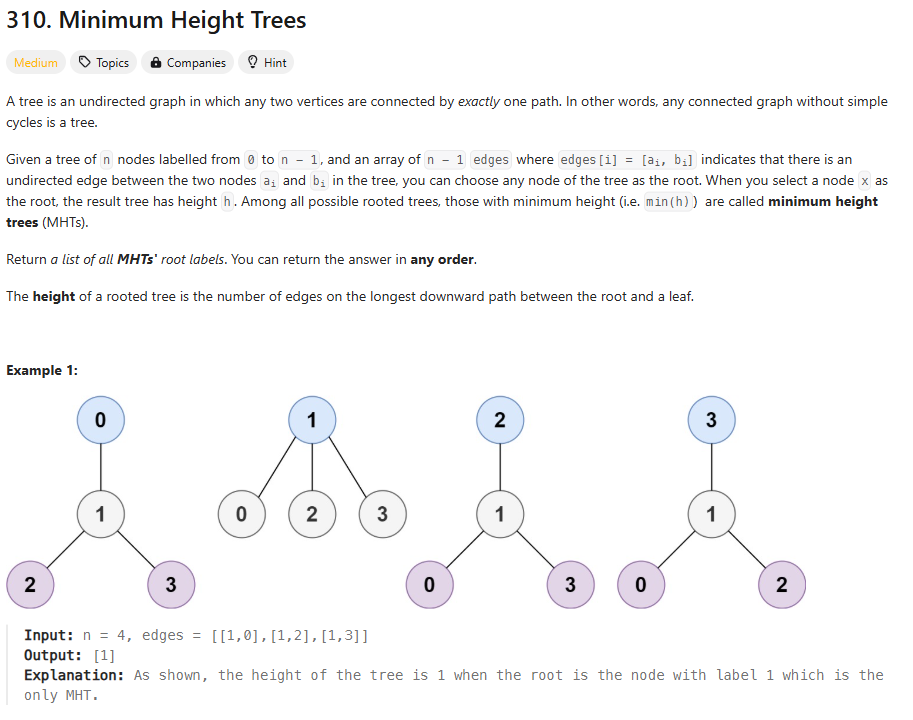
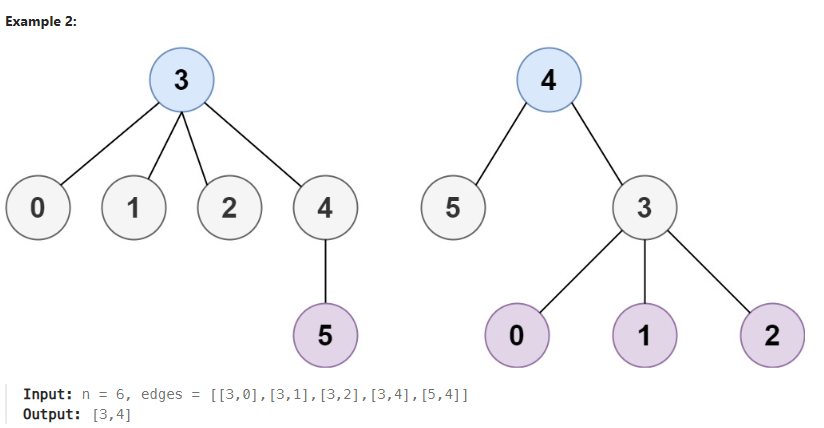
LAB 7: (LEETCODE)



Code:

#include <stdlib.h>

#define MAX\_N 20000

int\* findMinHeightTrees(int n, int\*\* edges, int edgesSize, int\* edgesColSize, int\* returnSize) {

    if (n == 1) {

        int\* res = (int\*)malloc(sizeof(int));

        res[0] = 0;

        \*returnSize = 1;

        return res;

    }

    int\* graph[MAX\_N];

    int graphSize[MAX\_N] = {0};

    int degree[MAX\_N] = {0};

    for (int i = 0; i < n; i++) {

        graph[i] = (int\*)malloc(sizeof(int) \* MAX\_N);

    }

    for (int i = 0; i < edgesSize; i++) {

        int u = edges[i][0];

        int v = edges[i][1];

        graph[u][graphSize[u]++] = v;

        graph[v][graphSize[v]++] = u;

        degree[u]++;

        degree[v]++;

    }

    int\* leaves = (int\*)malloc(sizeof(int) \* n);

    int front = 0, back = 0;

    for (int i = 0; i < n; i++) {

        if (degree[i] == 1) {

            leaves[back++] = i;

        }

    }

    int remainingNodes = n;

    while (remainingNodes > 2) {

        int leavesCount = back - front;

        remainingNodes -= leavesCount;

        for (int i = 0; i < leavesCount; i++) {

            int leaf = leaves[front++];

            for (int j = 0; j < graphSize[leaf]; j++) {

                int neighbor = graph[leaf][j];

                degree[neighbor]--;

                if (degree[neighbor] == 1) {

                    leaves[back++] = neighbor;

                }

            }

        }

    }

    \*returnSize = back - front;

    int\* result = (int\*)malloc(sizeof(int) \* (\*returnSize));

    for (int i = 0; i < \*returnSize; i++) {

        result[i] = leaves[front++];

    }

    for (int i = 0; i < n; i++) {

        free(graph[i]);

    }

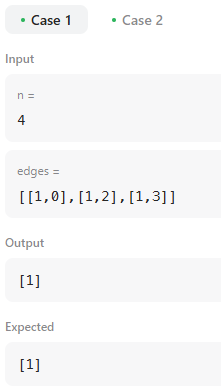
    free(leaves);

    return result;

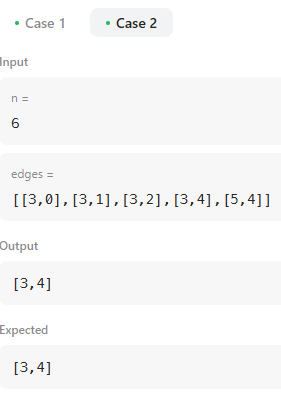
}

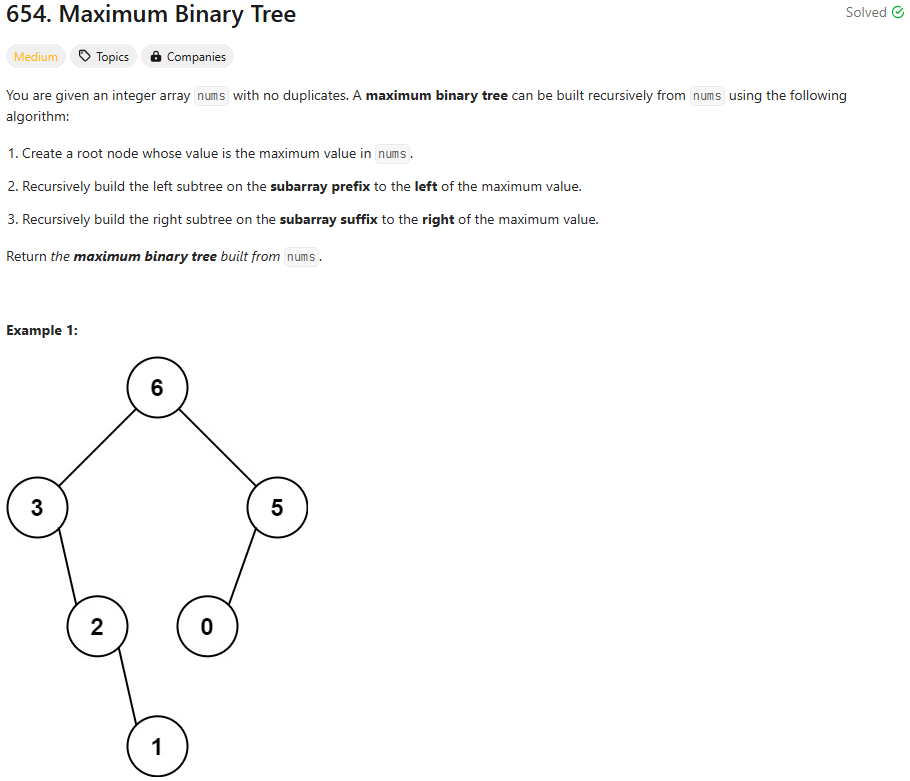
Output:

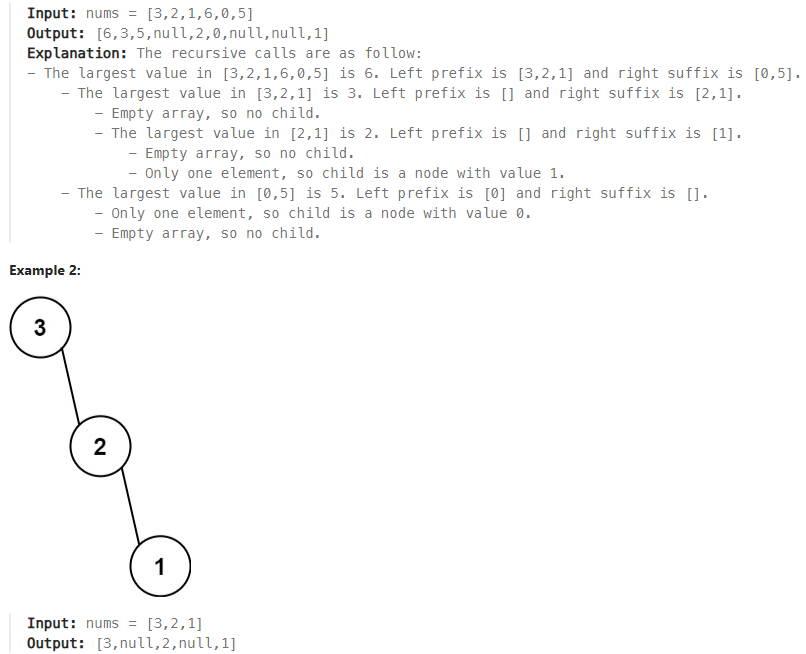
Case 1:



Case 2:







Code:

struct TreeNode\* constructMaximumBinaryTree(int\* nums, int numsSize){

    int max = -1, idx = -1;

    for (int i = 0; i < numsSize; i++){

        if (nums[i] > max){

            max = nums[i];

            idx = i;

        }

    }

    int size1 = idx;

    int size2 = numsSize - idx - 1;

    int \* pref = nums;

    int \* suff = nums + idx + 1;

    struct TreeNode \* root = malloc(sizeof(struct TreeNode));

    root->val = max;

    if (size1 != 0){

        root->left = constructMaximumBinaryTree(pref, size1);

    }

    else root->left = NULL;

    if (size2 != 0){

        root->right = constructMaximumBinaryTree(suff, size2);

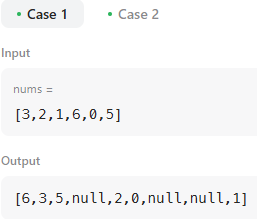
    }

    else root->right = NULL;

    return root;

}

Output:



Case 2:

