HACKER RANK

#include <assert.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

} Node;

Node\* createNode(int data) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    newNode->data = data;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

void inOrderTraversal(Node\* root, int\* result, int\* index) {

    if (root == NULL) return;

    inOrderTraversal(root->left, result, index);

    result[(\*index)++] = root->data;

    inOrderTraversal(root->right, result, index);

}

void swapAtLevel(Node\* root, int k, int level) {

    if (root == NULL) return;

    if (level % k == 0) {

        Node\* temp = root->left;

        root->left = root->right;

        root->right = temp;

    }

    swapAtLevel(root->left, k, level + 1);

    swapAtLevel(root->right, k, level + 1);

}

int\*\* swapNodes(int indexes\_rows, int indexes\_columns, int\*\* indexes, int queries\_count, int\* queries, int\* result\_rows, int\* result\_columns) {

    // Build the tree

    Node\*\* nodes = (Node\*\*)malloc((indexes\_rows + 1) \* sizeof(Node\*));

    for (int i = 1; i <= indexes\_rows; i++) {

        nodes[i] = createNode(i);

    }

    for (int i = 0; i < indexes\_rows; i++) {

        int leftIndex = indexes[i][0];

        int rightIndex = indexes[i][1];

        if (leftIndex != -1) nodes[i + 1]->left = nodes[leftIndex];

        if (rightIndex != -1) nodes[i + 1]->right = nodes[rightIndex];

    }

    // Perform swaps and store results

    int\*\* result = (int\*\*)malloc(queries\_count \* sizeof(int\*));

    \*result\_rows = queries\_count;

    \*result\_columns = indexes\_rows;

    for (int i = 0; i < queries\_count; i++) {

        swapAtLevel(nodes[1], queries[i], 1);

        int\* traversalResult = (int\*)malloc(indexes\_rows \* sizeof(int));

        int index = 0;

        inOrderTraversal(nodes[1], traversalResult, &index);

        result[i] = traversalResult;

    }

    free(nodes);

    return result;

}

int main() {

    int n;

    scanf("%d", &n);

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

    for (int i = 0; i < n; i++) {

        indexes[i] = malloc(2 \* sizeof(int));

        scanf("%d %d", &indexes[i][0], &indexes[i][1]);

    }

    int queries\_count;

    scanf("%d", &queries\_count);

    int\* queries = malloc(queries\_count \* sizeof(int));

    for (int i = 0; i < queries\_count; i++) {

        scanf("%d", &queries[i]);

    }

    int result\_rows;

    int result\_columns;

    int\*\* result = swapNodes(n, 2, indexes, queries\_count, queries, &result\_rows, &result\_columns);

    for (int i = 0; i < result\_rows; i++) {

        for (int j = 0; j < result\_columns; j++) {

            printf("%d ", result[i][j]);

        }

        printf("\n");

        free(result[i]); // Free memory allocated for each row

    }

    free(result); // Free memory allocated for the result array

    // Free memory allocated for indexes and queries arrays

    for (int i = 0; i < n; i++) {

        free(indexes[i]);

    }

    free(indexes);

    free(queries);

    return 0;

}

