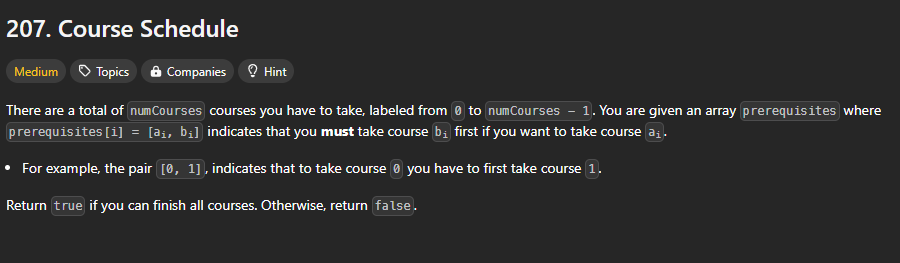
LAB - 8

Leetcode (Topological sorting)



#include <stdio.h>

#include <stdlib.h>

#define MAX\_COURSES 10000

bool canFinish(int numCourses, int\*\* prerequisites, int prerequisitesSize, int\* prerequisitesColSize) {

    int\* inDegree = (int\*)calloc(numCourses, sizeof(int));

    int\*\* graph = (int\*\*)malloc(numCourses \* sizeof(int\*));

    int\* graphSize = (int\*)calloc(numCourses, sizeof(int));

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

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

    }

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

        int course = prerequisites[i][0];

        int prereq = prerequisites[i][1];

        graph[prereq][graphSize[prereq]++] = course;

        inDegree[course]++;

    }

    int\* queue = (int\*)malloc(numCourses \* sizeof(int));

    int front = 0, rear = 0;

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

        if (inDegree[i] == 0) {

            queue[rear++] = i;

        }

    }

    int count = 0;

    while (front < rear) {

        int course = queue[front++];

        count++;

        for (int i = 0; i < graphSize[course]; i++) {

            int neighbor = graph[course][i];

            inDegree[neighbor]--;

            if (inDegree[neighbor] == 0) {

                queue[rear++] = neighbor;

            }

        }

    }

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

        free(graph[i]);

    }

    free(graph);

    free(graphSize);

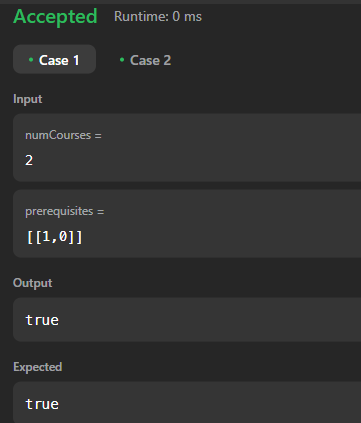
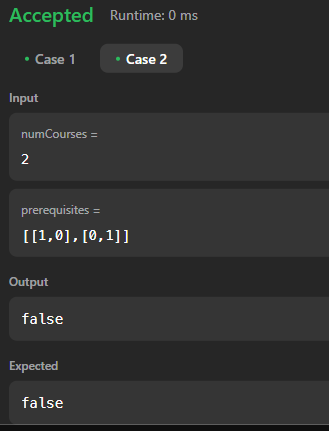
    free(inDegree);

    free(queue);

    return count == numCourses;

}

**OUTPUT-**

** **