**Day-5**

**Searching and Sorting**

**Student Name: Kamalpreet Singh UID: 22BCS11720**

**Branch: BE - CSE Section/Group: FL\_IOT-603-A**

**Semester: 5th Date of Performance:24-12-24**

**Problem 1 -Searching a Number**

**Aim- Given an integer k and array arr. Your task is to return the position of the first occurrence of k in the given array and if element k is not present in the array then return -1.**

**Note: 1-based indexing is followed here.**

**Solution-**#include <iostream>

using namespace std;

int findPosition(int arr[], int n, int k) {

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

if (arr[i] == k)

return i + 1;

}

return -1;

}

int main() {

int n, k;

cin >> n >> k;

int arr[n];

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

cin >> arr[i];

}

cout << findPosition(arr, n, k) << endl;

return 0;

}

**Output-**



**Problem 2 - Minimum Number of Moves to Seat Everyone**

**Aim - There are n availabe seats and n students standing in a room. You are given an array seats of length n, where seats[i] is the position of the ith seat. You are also given the array students of length n, where students[j] is the position of the jth student.**

**You may perform the following move any number of times:**

**Increase or decrease the position of the ith student by 1 (i.e., moving the ith student from position x to x + 1 or x - 1)**

**Return the minimum number of moves required to move each student to a seat such that no two students are in the same seat.**

**Note that there may be multiple seats or students in the same position at the beginning.**

**Solution-**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int minMovesToSeat(vector<int>& seats, vector<int>& students) {

sort(seats.begin(), seats.end());

sort(students.begin(), students.end());

int moves = 0;

for (int i = 0; i < seats.size(); i++) {

moves += abs(seats[i] - students[i]);

}

return moves;

}

int main() {

int n;

cin >> n;

vector<int> seats(n), students(n);

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

cin >> seats[i];

}

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

cin >> students[i];

}

cout << minMovesToSeat(seats, students) << endl;

return 0;

}

**Output-**



#### **Problem 3- Search in 2D Matrix.**

**Aim - You are given an m x n integer matrix matrix with the following two properties:**

**Each row is sorted in non-decreasing order.**

**The first integer of each row is greater than the last integer of the previous row.**

**Given an integer target, return true if target is in matrix or false otherwise.**

**You must write a solution in O(log(m \* n)) time complexity.**

**Solution-**

#include <iostream>

#include <vector>

using namespace std;

bool searchMatrix(vector<vector<int>>& matrix, int target) {

int m = matrix.size(), n = matrix[0].size();

int left = 0, right = m \* n - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int midValue = matrix[mid / n][mid % n];

if (midValue == target)

return true;

else if (midValue < target)

left = mid + 1;

else

right = mid - 1;

}

return false;

}

int main() {

int m, n, target;

cin >> m >> n >> target;

vector<vector<int>> matrix(m, vector<int>(n));

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

for (int j = 0; j < n; j++) {

cin >> matrix[i][j];

}

}

cout << (searchMatrix(matrix, target) ? "true" : "false") << endl;

return 0;

}

**Output:**



**Problem 4: Sort Items by Groups Respecting Dependencies**

**Aim- There are n items each belonging to zero or one of m groups where group[i] is the group that the i-th item belongs to and it's equal to -1 if the i-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.**

**Return a sorted list of the items such that:**

**The items that belong to the same group are next to each other in the sorted list.**

**There are some relations between these items where beforeItems[i] is a list containing all the items that should come before the i-th item in the sorted array (to the left of the i-th item).**

**Return any solution if there is more than one solution and return an empty list if there is no solution.**

**Solution-**

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_map>

#include <algorithm>

using namespace std;

vector<int> topologicalSort(int n, vector<int>& indegree, vector<vector<int>>& graph) {

queue<int> q;

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

if (indegree[i] == 0) q.push(i);

}

vector<int> order;

while (!q.empty()) {

int curr = q.front();

q.pop();

order.push\_back(curr);

for (int neighbor : graph[curr]) {

if (--indegree[neighbor] == 0) {

q.push(neighbor);

}

}

}

return order.size() == n ? order : vector<int>();

}

vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>& beforeItems) {

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

if (group[i] == -1) group[i] = m++;

}

vector<int> itemIndegree(n, 0), groupIndegree(m, 0);

vector<vector<int>> itemGraph(n), groupGraph(m);

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

for (int before : beforeItems[i]) {

itemGraph[before].push\_back(i);

itemIndegree[i]++;

if (group[before] != group[i]) {

groupGraph[group[before]].push\_back(group[i]);

groupIndegree[group[i]]++;

}

}

}

vector<int> groupOrder = topologicalSort(m, groupIndegree, groupGraph);

if (groupOrder.empty()) return {};

vector<int> itemOrder = topologicalSort(n, itemIndegree, itemGraph);

if (itemOrder.empty()) return {};

unordered\_map<int, vector<int>> groupedItems;

for (int item : itemOrder) {

groupedItems[group[item]].push\_back(item);

}

vector<int> result;

for (int grp : groupOrder) {

result.insert(result.end(), groupedItems[grp].begin(), groupedItems[grp].end());

}

return result;

}

int main() {

int n, m;

cin >> n >> m;

vector<int> group(n);

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

cin >> group[i];

}

vector<vector<int>> beforeItems(n);

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

int k;

cin >> k;

beforeItems[i].resize(k);

for (int j = 0; j < k; j++) {

cin >> beforeItems[i][j];

}

}

vector<int> result = sortItems(n, m, group, beforeItems);

if (result.empty()) {

cout << "[]" << endl;

} else {

for (int item : result) {

cout << item << " ";

}

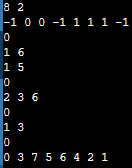
cout << endl;

}

return 0;

}

**Output-**



**Problem 5: Find Minimum in Rotated Sorted Array II.**

**Aim- Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,4,4,5,6,7] might become:**

**[4,5,6,7,0,1,4] if it was rotated 4 times.**

**[0,1,4,4,5,6,7] if it was rotated 7 times.**

**Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].**

**Given the sorted rotated array nums that may contain duplicates, return the minimum element of this array.**

#### **You must decrease the overall operation steps as much as possible.**

## Solution:

#include <iostream>

#include <vector>

using namespace std;

int findMin(vector<int>& nums) {

int left = 0, right = nums.size() - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (nums[mid] < nums[right]) {

right = mid;

} else if (nums[mid] > nums[right]) {

left = mid + 1;

} else {

right--;

}

}

return nums[left];

}

int main() {

int n;

cin >> n;

vector<int> nums(n);

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

cin >> nums[i];

}

cout << findMin(nums) << endl;

return 0;

}

**Output-**

