DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Gurnoor Oberoi **UID: 22BCS15716**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/B**
Semester: 5th

➢ DAY-5 [24-12-2024]

1. Searching a Number *(Very Easy)*

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
using namespace std;
int findFirstOccurrence(vector<int>& arr, int k) {
    for (int i = 1; i < arr.size()+1; i++) {
        if (arr[i] == k) {
            return i;
        }
    }
    return -1;
}
int main() {
    int n, k;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the elements of the array: ";
    for (int i = 1; i < n+1; i++) {
        cin >> arr[i];
    }
    cout << "Enter the element to search for: ";
    cin >> k;
```

```
    int result = findFirstOccurrence(arr, k);
    if (result != -1) {
        cout << "The first occurrence of " << k << " is at index: " << result << endl;
    } else {
        cout << "Element " << k << " is not present in the array." << endl;
    }
    return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the array: 9 7 16 16 4
Enter the element to search for: 16
The first occurrence of 16 is at index: 3
```

2. Minimum Number of Moves to Seat Everyone *(Easy)*

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.

Implementation/Code:

```
#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 totalMoves = 0;
    for (int i = 0; i < seats.size(); i++) {
        totalMoves += abs(seats[i] - students[i]);
    }
```

```
    return totalMoves;
}
int main() {
    int n;
    cout << "Enter the number of seats and students: ";
    cin >> n;
    vector<int> seats(n), students(n);
    cout << "Enter the positions of seats: ";
    for (int i = 0; i < n; i++) {
        cin >> seats[i];
    }
    cout << "Enter the positions of students: ";
    for (int i = 0; i < n; i++) {
        cin >> students[i];
    }
    int result = minMovesToSeat(seats, students);
    cout << "The minimum number of moves required is: " << result << endl;
    return 0;
}
```

Output:

```
Enter the number of seats and students: 3
Enter the positions of seats: 3 1 5
Enter the positions of students: 2 7 4
The minimum number of moves required is: 4
```

3. Search in 2D Matrix *(Medium)*

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
```

```
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int row = matrix.size();
        int col = matrix[0].size();
        int s = 0;
        int e = row * col - 1;
        while (s <= e) {
            int m = s + (e - s) / 2;
            int element = matrix[m / col][m % col];
            if (target == element) {
                return true;
            } else if (target < element) {
                e = m - 1;
            } else {
                s = m + 1;
            }
        }
        return false;
    }
};
int main() {
    int rows, cols, target;
    cout << "Enter the number of rows: ";
    cin >> rows;
    cout << "Enter the number of columns: ";
    cin >> cols;
    vector<vector<int>> matrix(rows, vector<int>(cols));
    cout << "Enter the elements of the matrix row by row:\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    cout << "Enter the target value: ";
    cin >> target;
    Solution solution;
    bool result = solution.searchMatrix(matrix, target);\
    if (result) {
        cout << "Target is in the matrix." << endl;
```

```
    } else {
        cout << "Target is not in the matrix." << endl;
    }
    return 0;
}
```

Output:

```
Enter the number of rows: 4
Enter the number of columns: 3
Enter the elements of the matrix row by row:
1 3 5 7
10 11 16 20
23 30 34 60
Enter the target value: 3
Target is in the matrix.
```

4. Sort Items by Groups Respecting Dependencies *(Hard)*

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
#include <unordered_set>
using namespace std;
class Solution {
public:
    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<vector<int>> groupGraph(m);
        vector<int> groupIndegree(m, 0);
        vector<vector<int>> itemGraph(n);
        vector<int> itemIndegree(n, 0);
        unordered_map<int, unordered_set<int>> groupDependencies;

        for (int i = 0; i < n; ++i) {
            for (int before : beforeItems[i]) {
                itemGraph[before].push_back(i);
                ++itemIndegree[i];
                if (group[before] != group[i] &&
!groupDependencies[group[before]].count(group[i])) {
                    groupGraph[group[before]].push_back(group[i]);
                    ++groupIndegree[group[i]];
                    groupDependencies[group[before]].insert(group[i]);
                }
            }
        }
        vector<int> sortedGroups = topologicalSort(m, groupGraph, groupIndegree);
        if (sortedGroups.empty()) return {};
        vector<int> sortedItems = topologicalSort(n, itemGraph, itemIndegree);
        if (sortedItems.empty()) return {};
        unordered_map<int, vector<int>> groupToItems;
        for (int item : sortedItems) {
            groupToItems[group[item]].push_back(item);
        }
        vector<int> result;
        for (int g : sortedGroups) {
            result.insert(result.end(), groupToItems[g].begin(), groupToItems[g].end());
        }
        return result;
    }
private:
    vector<int> topologicalSort(int n, const vector<vector<int>>& graph, vector<int>&
indegree) {
        queue<int> q;
```

```
        for (int i = 0; i < n; ++i) {
            if (indegree[i] == 0) q.push(i);
        }
        vector<int> sorted;
        while (!q.empty()) {
            int curr = q.front();
            q.pop();
            sorted.push_back(curr);
            for (int neighbor : graph[curr]) {
                if (--indegree[neighbor] == 0) {
                    q.push(neighbor);
                }
            }
        }
        return sorted.size() == n ? sorted : vector<int>{};
    }
};
int main() {
    Solution solution;
    int n = 8;
    int m = 2;
    vector<int> group = {-1, 0, 0, 1, 1, -1, 0, -1};
    vector<vector<int>> beforeItems = {
        {},
        {6},
        {5},
        {3, 6},
        {},
        {},
        {}
    };
    vector<int> result = solution.sortItems(n, m, group, beforeItems);
    if (result.empty()) {
        cout << "No valid ordering exists." << endl;
    } else {
        cout << "Valid ordering: ";
        for (int item : result) {
            cout << item << " ";
```

```
        }
        cout << endl;
    }
    return 0;
}
```

Output:

```
Valid ordering: 0 5 7 6 2 1 3 4
```

5. Find Minimum in Rotated Sorted Array II *(Very Hard)*

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.

Implementation/Code:

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
class Solution {
public:
    int findMin(vector<int>& nums) {
        int low = 0, high = nums.size() - 1;
        int minVal = INT_MAX;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] < minVal) {
                minVal = nums[mid];
            }
            if (nums[low] == nums[mid] && nums[mid] == nums[high]) {
                low++;
                high--;
            }
```

```
            else if (nums[low] <= nums[mid]) {
                minVal = min(minVal, nums[low]);
                low = mid + 1;
            }
            else {
                high = mid - 1;
            }
        }
        return minVal;
    }
};
int main() {
    Solution sol;
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter the elements of the rotated sorted array: ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }
    int result = sol.findMin(nums);
    cout << "The minimum element in the array is: " << result << endl;
    return 0;
}
```

Output:

```
Enter the number of elements in the array: 3
Enter the elements of the rotated sorted array: 1 3 5
The minimum element in the array is: 1
```