DOMAIN WINTER CAMP WORKSHEET

DAY-5 (24/12/2024)

Student Name :- Pratham Kapoor **University ID :- 22BCS10732**
Branch :- B.E. (C.S.E.) **Section/Group :- FL_ 603-A**

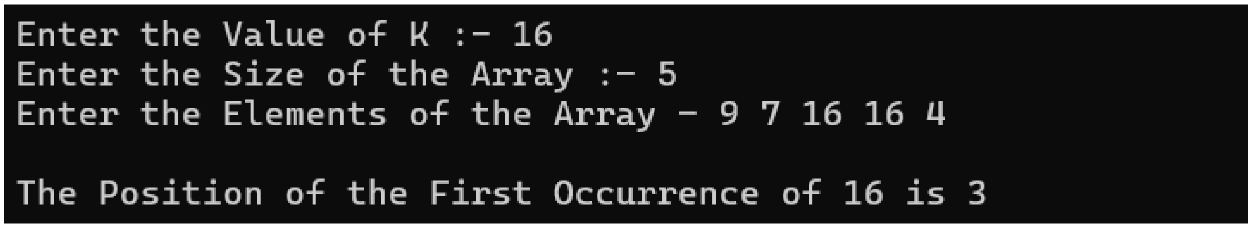
Problem-1 :- Given an integer k and array arr. Your task is to return position of the first occurrence of k in the given array and if element k is not present in the array then return -1. Note :- 1-based indexing is followed here. **(Very Easy)**

Source Code

```
#include<iostream>
#include<vector>
using namespace std;

int findFirstOccurrence(int k, vector<int>& arr)
{
   for(int i=0; i<arr.size(); i++)
  {
      if(arr[i]==k) {
         return i+1;
      }
   }
   return -1;
}

int main()
{
   int k, n;
   cout << "Enter the Value of K :- ";
```

```
    cin >> k;
    cout << "Enter the Size of the Array :- ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter the Elements of the Array - ";
    for(int i=0; i<n; i++)
   {
        cin >> arr[i];
    }

    int result = findFirstOccurrence(k, arr);
    cout << "\nThe Position of the First Occurrence of " << k <<" is " << result
<< endl;
    return 0;
}
```

Output



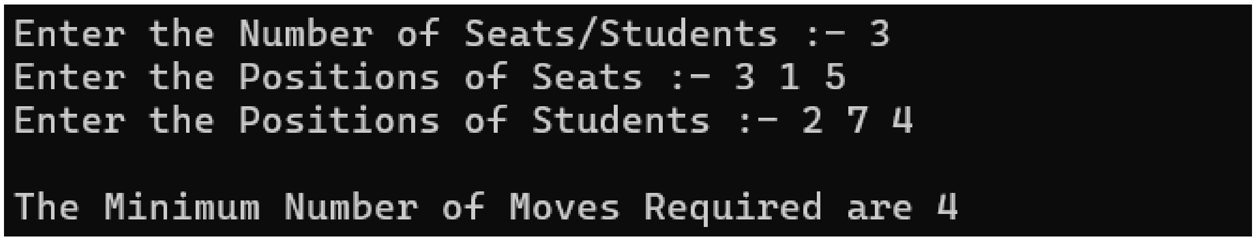
```
Enter the Value of K :- 16
Enter the Size of the Array :- 5
Enter the Elements of the Array - 9 7 16 16 4

The Position of the First Occurrence of 16 is 3
```

Problem-2 :- 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. **(Easy)**

Source 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 moves = 0;

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

    return moves;
}

int main() {
    int n;
    cout << "Enter the Number of Seats/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 << "\nThe Minimum Number of Moves Required are " << result <<
endl;

    return 0;
}
```

Output



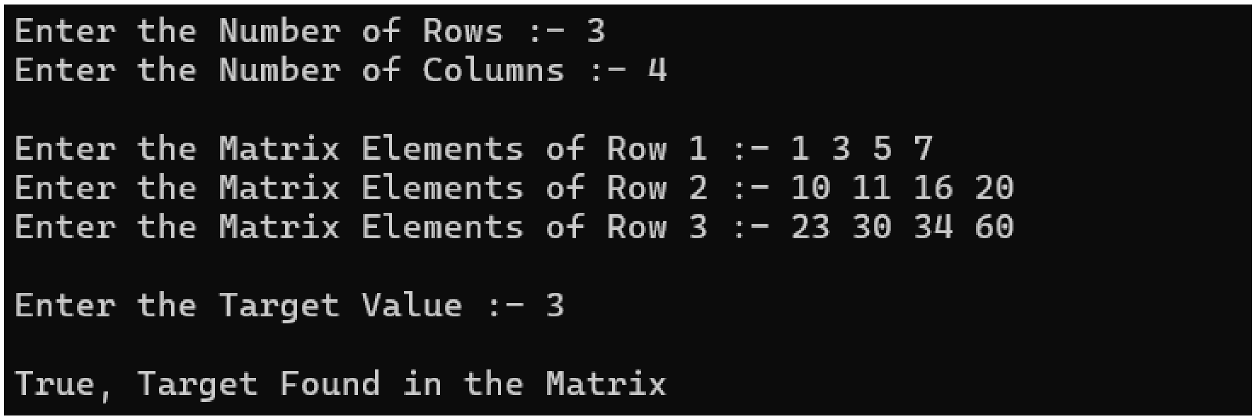
```
Enter the Number of Seats/Students :- 3
Enter the Positions of Seats :- 3 1 5
Enter the Positions of Students :- 2 7 4

The Minimum Number of Moves Required are 4
```

Problem-3 :- 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. **(Medium)**

Source Code

```
#include<iostream>
#include<vector>
using namespace std;

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size();
    int n = matrix[0].size();
    int low = 0, high = m * n - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;
        int midValue = matrix[mid / n][mid % n];

        if (midValue == target) {
            return true;
        } else if (midValue < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return false;
}

int main() {
    int m, n, target;

    cout << "Enter the Number of Rows :- ";
    cin >> m;
    cout << "Enter the Number of Columns :- ";
    cin >> n;
    cout<<endl;
    vector<vector<int>> matrix(m, vector<int>(n));
    for (int i = 0; i < m; i++) {
        cout << "Enter the Matrix Elements of Row " << i + 1 << " :- ";
        for (int j = 0; j < n; j++) {
            cin >> matrix[i][j];
        }
    }

    cout << "\nEnter the Target Value :- ";
    cin >> target;

    if (searchMatrix(matrix, target)) {
        cout << "\nTrue, Target Found in the Matrix" << endl;
    } else {
        cout << "\nFalse, Target Not Found in the Matrix" << endl;
    }
    return 0;
}
```

Output



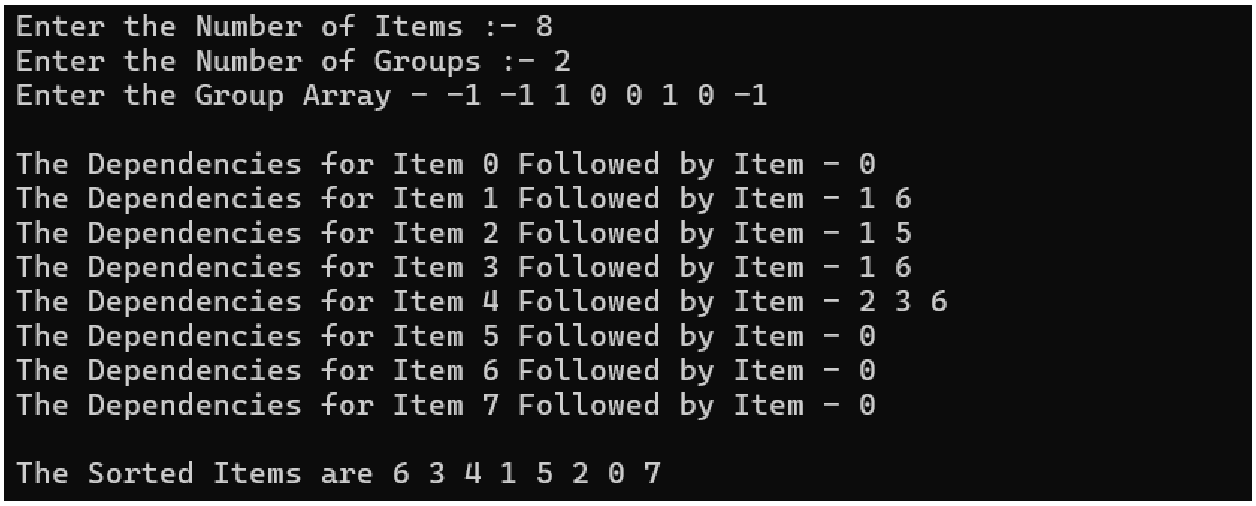
```
Enter the Number of Rows :- 3
Enter the Number of Columns :- 4

Enter the Matrix Elements of Row 1 :- 1 3 5 7
Enter the Matrix Elements of Row 2 :- 10 11 16 20
Enter the Matrix Elements of Row 3 :- 23 30 34 60

Enter the Target Value :- 3

True, Target Found in the Matrix
```

Problem-4 :- 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. **(Hard)**

Source Code

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

vector<int> TopologicalSort(const unordered_map<int, vector<int>>& Graph, vector<int>& InDegree, int Nodes) {
    queue<int> Q;
    vector<int> SortedOrder;

    for (int I = 0; I < Nodes; ++I) {
        if (InDegree[I] == 0) {
            Q.push(I);
        }
    }

    while (!Q.empty()) {
        int Node = Q.front();
        Q.pop();
        SortedOrder.push_back(Node);

        if (Graph.find(Node) != Graph.end()) {
            for (int Neighbor : Graph.at(Node)) {
                InDegree[Neighbor]--;
                if (InDegree[Neighbor] == 0) {
                    Q.push(Neighbor);
            }
         }
      }
   }

   if (SortedOrder.size() == Nodes) {
      return SortedOrder;
   }
   return {};
}
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++;
      }
   }
   unordered_map<int, vector<int>> ItemGraph, GroupGraph;
   vector<int> ItemInDegree(N, 0), GroupInDegree(M, 0);
   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(GroupGraph, GroupInDegree,
M);
   if (GroupOrder.empty()) {
      return {};
   }
   vector<int> ItemOrder = TopologicalSort(ItemGraph, ItemInDegree, N);
   if (ItemOrder.empty()) {
      return {};
   }
   unordered_map<int, vector<int>> GroupToItems;
   for (int Item : ItemOrder) {
      GroupToItems[Group[Item]].push_back(Item);
   }
   vector<int> Result;
   for (int Grp : GroupOrder) {
      if (GroupToItems.find(Grp) != GroupToItems.end()) {
         Result.insert(Result.end(),                GroupToItems[Grp].begin(),
GroupToItems[Grp].end());
      }
   }
   return Result;
}

int main()
{
   int N, M;
   cout << "Enter the Number of Items :- ";
   cin >> N;
   cout << "Enter the Number of Groups :- ";
   cin >> M;

   vector<int> Group(N);
   cout << "Enter the Group Array - ";
   for (int I = 0; I < N; ++I)
  {
      cin >> Group[I];
   }
   vector<vector<int>> BeforeItems(N);
   cout <<  endl;
   for (int I = 0; I < N; ++I)
   {
      int Count;
      cout << "The Dependencies for Item " << I << " Followed by Item - ";
      cin >> Count;
      BeforeItems[I].resize(Count);
      for (int J = 0; J < Count; ++J)
     {
         cin >> BeforeItems[I][J];
      }
   }

   vector<int> Result = SortItems(N, M, Group, BeforeItems);

   if (Result.empty())
  {
      cout << "\nNo Valid Ordering Exists." << endl;
   } else {
      cout << "\nThe Sorted Items are ";
      for (int Item : Result)
     {
         cout << Item << " ";
      }
      cout << endl;
   }

   return 0;
}
```

Output



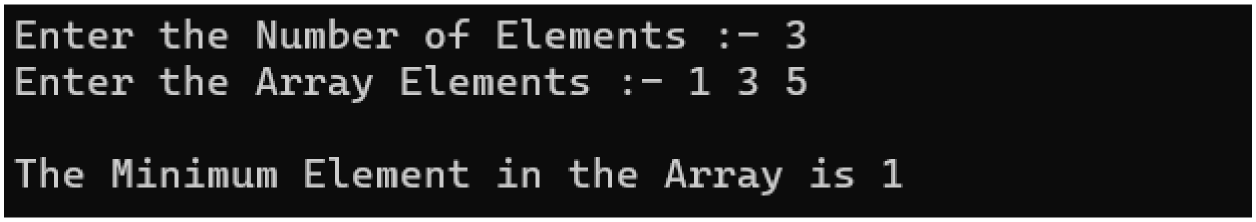
```
Enter the Number of Items :- 8
Enter the Number of Groups :- 2
Enter the Group Array - -1 -1 1 0 0 1 0 -1

The Dependencies for Item 0 Followed by Item - 0
The Dependencies for Item 1 Followed by Item - 1 6
The Dependencies for Item 2 Followed by Item - 1 5
The Dependencies for Item 3 Followed by Item - 1 6
The Dependencies for Item 4 Followed by Item - 2 3 6
The Dependencies for Item 5 Followed by Item - 0
The Dependencies for Item 6 Followed by Item - 0
The Dependencies for Item 7 Followed by Item - 0

The Sorted Items are 6 3 4 1 5 2 0 7
```

Problem-5 :- 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 array [a[n-1], a[0], a[1], a[2], ..., a[n-2]]. Given sorted rotated array nums that may contain duplicates, return minimum element of this array. You must decrease the overall operation steps as much as possible. **(Very Hard)**

Source Code

```
#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]) {
            left = mid + 1;
        } else if (nums[mid] < nums[right]) {
            right = mid;
        } else {
            right--;
        }
    }
    return nums[left];
}

int main() {
    int n;
    cout << "Enter the Number of Elements :- ";
    cin >> n;

    vector<int> nums(n);
    cout << "Enter the Array Elements :- ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    int minElement = findMin(nums);
    cout << "\nThe Minimum Element in the Array is " << minElement << endl;

    return 0; }
```

Output



```
Enter the Number of Elements :- 3
Enter the Array Elements :- 1 3 5

The Minimum Element in the Array is 1
```