Batch:T6

Practical No.5

Title of Assignment: Greedy approach

Student Name: Parshwa Herwade

Student PRN: 22510064

1) You are working on the city construction project. You have A houses in the city. You have to divide these houses into B localities such that every locality has at least one house. Also, every house in a locality should have a telephone connection wire with each of the other houses in the locality.You are given integers A and B.

Task: Print the minimum and the maximum number of telephone connections possible if you design the city accordingly.

ANS.

Pseudocode:

```
function minimum_connections(A, B):
    x = A // B  # Minimum houses per locality
    y = A % B   # Extra houses to distribute
    min_connections = (B - y) * (x * (x - 1)) // 2 + y * (x * (x + 1)) // 2
    return min_connections


function maximum_connections(A):
    max_connections = A * (A - 1) // 2
    return max_connections


input A, B
print minimum_connections(A, B)
print maximum_connections(A)
```

Code:

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

int minimum_connections(int A, int B) {
```

```
    int x = A / B;
    int y = A % B;
    int min_connections = (B - y) * (x * (x - 1)) / 2 + y * (x * (x
+ 1)) / 2;
    return min_connections;
}

int maximum_connections(int A) {
    return A * (A - 1) / 2;
}

int main() {
    int A, B;
    cin >> A >> B;
    cout << "Minimum Connections: " << minimum_connections(A, B) <<
endl;
    cout << "Maximum Connections: " << maximum_connections(A) <<
endl;
    return 0;
}
```

Output:

```
PS C:\Users\Parshwa\Desktop\CL(
5 2
Minimum Connections: 4
Maximum Connections: 10
PS C:\Users\Parshwa\Desktop\CL(
7 3
Minimum Connections: 5
Maximum Connections: 21
PS C:\Users\Parshwa\Desktop\CL(
8 4
Minimum Connections: 4
Maximum Connections: 28
PS C:\Users\Parshwa\Desktop\CL(
10 5
Minimum Connections: 5
Maximum Connections: 45
```

Time Complexity:

- Best/Average/Worst: O(1) since the operations are constant-time arithmetic.

Space Complexity:

- O(1) as we only store a few variables.

2)You are working in the Data Consistency team of your company. You are allocated a task as follows: • You have a data stream consisting of an equal number of odd and even numbers. You can make separations in the data stream but the number of odd elements should be equal to the number of even elements in both partitions after separation. Also, if you make a separation between a number x and number y, then the cost of this operation will be |x-y| coins. You are given the following: • An integer N • An array arr • An integer K

Task: Determine the maximum number of separations that can be made in the array by spending no more than K coins.

ANS.

Pseudocode

```
function max_separations(arr, N, K):
  odd_count = 0
  even_count = 0
  separations = 0
  cost = 0

  for i = 0 to N-2:
    if arr[i] is odd:
      odd_count += 1
    else:
      even_count += 1

    if odd_count == even_count:
      cost += abs(arr[i] - arr[i+1])
      if cost <= K:
        separations += 1
      else:
```

            break

    return separations

input N, K, arr[]

print max_separations(arr, N, K)

Code:
```
#include <iostream>

#include <cmath>
using namespace std;

int max_separations(int arr[], int N, int K) {
    int odd_count = 0, even_count = 0, separations = 0, cost = 0;

    for (int i = 0; i < N - 1; i++) {
        if (arr[i] % 2 == 0)
            even_count++;
        else
            odd_count++;

        if (odd_count == even_count) {
            cost += abs(arr[i] - arr[i + 1]);
            if (cost <= K) {
                separations++;
            } else {
                break;
            }
        }
    }
    return separations;
}

int main() {
    int N, K;
    cin >> N >> K;
    int arr[N];
    for (int i = 0; i < N; i++) {
```

```
        cin >> arr[i];
    }
    cout << "Max Separations: " << max_separations(arr, N, K) <<
endl;
    return 0;
}
```

Output:

```
6 5
1 2 3 4 5 6
Max Separations: 2
PS C:\Users\Parshwa\Desktop\CLG
PS C:\Users\Parshwa\Desktop\CLG
tempCodeRunnerFile }
4 1
1 3 5 7
Max Separations: 0
PS C:\Users\Parshwa\Desktop\CLG
tempCodeRunnerFile }
8 4
1 3 2 4 5 7 6 8
Max Separations: 1
PS C:\Users\Parshwa\Desktop\CLG
```

Time Complexity:

- Best/Average/Worst: O(N) since we traverse the array once.

Space Complexity:

- O(1) as we use a fixed number of variables.

3) Bob has an array A of size N, and he is very fond of two integers X and Y. Find the length of the longest subarray, such that it contains exactly X distinct integers and Y exist at least once in the subarray.

Input format

- The first line contains an integer T, which denotes the number of test cases.
- The first line of each test case contains three space separated integers N, X, Y denoting the size of array A, the value of X and Y, respectively.
- The second line of each test case contains N space-separated integers, denoting the elements of array A.

Output format

For each test case, print the length of the largest subarray, such that it contains exactly X distinct integers and Y exist at least once in the subarray in a new line.

ANS.

Pseudocode:

```
function longest_subarray_with_conditions(A, N, X, Y):
    freq_map = empty dictionary
    left = 0
    max_len = 0
    distinct_count = 0
    y_present = false

    for right = 0 to N-1:
        # Add A[right] to the window
        if A[right] not in freq_map or freq_map[A[right]] == 0:
            distinct_count += 1
        freq_map[A[right]] += 1

        # Check if Y is in the current window
        if A[right] == Y:
            y_present = true
```

```
        # Shrink window if distinct_count exceeds X
        while distinct_count > X:
            freq_map[A[left]] -= 1
            if freq_map[A[left]] == 0:
                distinct_count -= 1
            if A[left] == Y:
                y_present = false
            left += 1

        # If the window has exactly X distinct elements and contains Y, update max_len
        if distinct_count == X and y_present:
            max_len = max(max_len, right - left + 1)

    return max_len

# Input handling
input T
for each test case:
    input N, X, Y
    input array A[N]
    result = longest_subarray_with_conditions(A, N, X, Y)
    print result
```

Code:

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

int longest_subarray_with_conditions(const vector<int>& A, int N,
int X, int Y) {
    unordered_map<int, int> freq_map;
    int left = 0, max_len = 0, distinct_count = 0;
```

```
    bool y_present = false;

    for (int right = 0; right < N; ++right) {
        if (freq_map[A[right]] == 0)
            distinct_count++;
        freq_map[A[right]]++;

        if (A[right] == Y)
            y_present = true;

        while (distinct_count > X) {
            freq_map[A[left]]--;
            if (freq_map[A[left]] == 0)
                distinct_count--;
            if (A[left] == Y)
                y_present = false;
            left++;
        }

        if (distinct_count == X && y_present)
            max_len = max(max_len, right - left + 1);
    }

    return max_len;
}

int main() {
    int T;
    cin >> T;

    while (T--) {
        int N, X, Y;
        cin >> N >> X >> Y;
        vector<int> A(N);

        for (int i = 0; i < N; ++i)
            cin >> A[i];

        cout << longest_subarray_with_conditions(A, N, X, Y) <<
endl;
    }
```

```
    return 0;
}
```

Output:

```
2
5 3 2
1 2 3 2 1
5
6 2 3
3 1 2 1 4 3
2
PS C:\Users\Parshwa\Desktop\
1
6 2 3
1 2 3 4 5 3
2
PS C:\Users\Parshwa\Desktop\
1
7 3 5
5 6 5 7 5 8 5
5
PS C:\Users\Parshwa\Desktop\
1
10 4 3
3 1 2 3 4 2 5 3 6 3
6
PS C:\Users\Parshwa\Desktop\
```

Time Complexity:

- **Best/Average/Worst Case**: O(N) per test case due to the sliding window approach where both pointers (left and right) only traverse the array once.
    - Overall complexity for T test cases: O(T * N).

Space Complexity:

- O(N) due to the frequency map that stores counts of up to N elements.

4) The country of Byteland consists of n cities. Between any 2 cities it is possible to have a railway track and a road. Railway tracks are bidirectional, meaning if there exists a

railway track between u and v then you can take a train from u to v as well as from v to u. Similarly, roads are bidirectional, meaning if there exists a route between u and v then you can drive from u to v as well as from v to u. 2 cities, u and v are called railway-connected if it is possible to travel between u and v using railway tracks. 2 cities, u and v are called road-connected if it is possible to travel between u and v using roads.The transportation network is called balanced if for all pairs of cities u, v: u,v are railway-connected if and only if u,v are road-connected. Initially, there are n cities and no roads or railways in Byteland. You will be given q instructions asking you to build either a railway track or a road between some 2 cities. After each instruction, you must report whether the transportation network is balanced.

Input format: The first line of input will contain 2 integers, n and q. q lines will follow. Each line will contain 3 space-separated integers in one of the following formats: 1 u v : build a railway track between u and v 2 u v : build a road between u and v .

Output format You must print q lines. The ith line contains an answer to the question whether the transport network is balanced after the ith instruction. If it is balanced print "YES" (without quotes) otherwise print "NO" (without quotes)

ANS.

Pseudocode

```
function find(parent[], u):
    if parent[u] != u:
        parent[u] = find(parent, parent[u])
    return parent[u]


function union(parent[], rank[], u, v):
    root_u = find(parent, u)
    root_v = find(parent, v)


    if root_u != root_v:
        if rank[root_u] > rank[root_v]:
            parent[root_v] = root_u
        else if rank[root_u] < rank[root_v]:
            parent[root_u] = root_v
        else:
            parent[root_v] = root_u
```

```
      rank[root_u] += 1


function is_balanced(rail_parent[], road_parent[], u, v):
  return find(rail_parent, u) == find(road_parent, u)


input n, q
initialize rail_parent[], road_parent[], rail_rank[], road_rank[]
for each query:
  if type == 1:
    union(rail_parent, rail_rank, u, v)
  else:
    union(road_parent, road_rank, u, v)


  if is_balanced(rail_parent, road_parent, u, v):
    print "YES"
  else:
    print "NO"
```

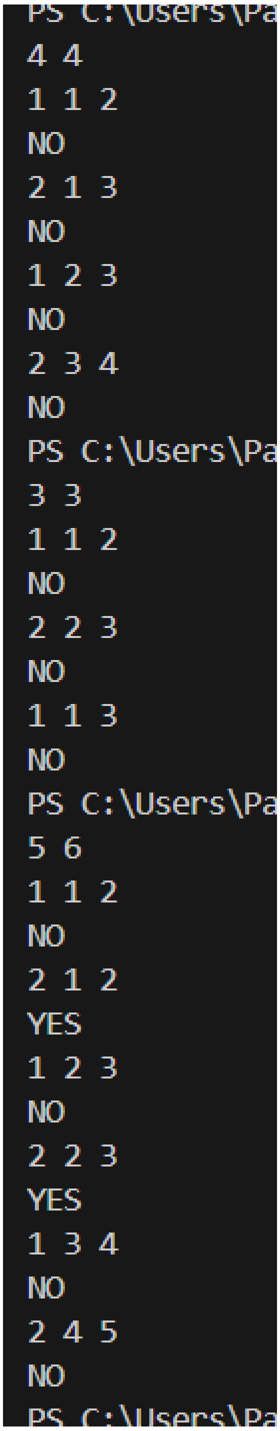
Code:

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

int find(vector<int>& parent, int u) {
    if (parent[u] != u)
        parent[u] = find(parent, parent[u]);
    return parent[u];
}

void union_sets(vector<int>& parent, vector<int>& rank, int u, int
v) {
    int root_u = find(parent, u);
    int root_v = find(parent, v);
```

```
    if (root_u != root_v) {
        if (rank[root_u] > rank[root_v])
            parent[root_v] = root_u;
        else if (rank[root_u] < rank[root_v])
            parent[root_u] = root_v;
        else {
            parent[root_v] = root_u;
            rank[root_u]++;
        }
    }
}

bool is_balanced(vector<int>& rail_parent, vector<int>& road_parent,
int u, int v) {
    return find(rail_parent, u) == find(rail_parent, v) &&
           find(road_parent, u) == find(road_parent, v);
}

int main() {
    int n, q;
    cin >> n >> q;

    vector<int> rail_parent(n + 1), road_parent(n + 1), rail_rank(n
+ 1, 0), road_rank(n + 1, 0);

    for (int i = 1; i <= n; i++) {
        rail_parent[i] = i;
        road_parent[i] = i;
    }

    for (int i = 0; i < q; i++) {
        int type, u, v;
        cin >> type >> u >> v;
        if (type == 1) {
            union_sets(rail_parent, rail_rank, u, v);
        } else {
            union_sets(road_parent, road_rank, u, v);
        }

        if (is_balanced(rail_parent, road_parent, u, v)) {
            cout << "YES" << endl;
```

```
        } else {
            cout << "NO" << endl;
        }
    }
    return 0;
}
```

Output:



```
PS C:\Users\Pa
4 4
1 1 2
NO
2 1 3
NO
1 2 3
NO
2 3 4
NO
PS C:\Users\Pa
3 3
1 1 2
NO
2 2 3
NO
1 1 3
NO
PS C:\Users\Pa
5 6
1 1 2
NO
2 1 2
YES
1 2 3
NO
2 2 3
YES
1 3 4
NO
2 4 5
NO
PS C:\Users\Pa
```

Time Complexity:

- Best/Average/Worst: O(q log n) due to union-find with path compression.

Space Complexity:

- O(n) for storing the parent and rank arrays.