**Task 4. Java Collections CO3**

* **Handle list of elements using Collection classes (List, Queue, Set, Map)**
  1. Sometimes it's better to use dynamic size arrays. Java's Arraylist can provide you this feature. Try to solve this problem using Arraylist. You are given lines. In each line there are zero or more integers. You need to answer a few queries where you need to tell the number located in positionof line.

**Input Format:**

First Line to read the integer value

Second line get the value of the input with spaces

Third line get the value of the input with spaces and so on

**Output Format:**

In each line, output the number located in positionof line. If there is no such position, just print "ERROR!"

**Constraint:**

0<=n<=100

**Sample Input**

5

5 41 77 74 22 44

1 12

4 37 34 36 52

0

3 20 22 33

5

1 3

3 4

3 1

4 3

5 5

**Sample Output**

74

52

37

ERROR!

ERROR!

**Program**

|  |
| --- |
| import java.io.\*;  import java.util.\*;  import java.text.\*;  import java.math.\*;  import java.util.regex.\*;  public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int numLines = Integer.parseInt(sc.nextLine());  ArrayList<ArrayList> listArray = new ArrayList<ArrayList>();  for(int i = 0;i<numLines;i++){  int numOfIntegers = sc.nextInt();  ArrayList<Integer> intArrayList = new ArrayList<Integer>();  for(int j=0;j<numOfIntegers;j++){  intArrayList.add(new Integer(sc.nextInt()));  }  listArray.add(intArrayList);  sc.nextLine();  }  int numQueries = Integer.parseInt(sc.nextLine());  for(int i=0;i<numQueries;i++){  int x = sc.nextInt()-1;  int y = sc.nextInt()-1;  sc.nextLine();  if(x<listArray.size() && y<listArray.get(x).size()){  System.out.println(listArray.get(x).get(y));  }else{  System.out.println("ERROR!");  }  }  }  } |

* 1. You are given a phone book that consists of people's names and their phone number. After that you will be given some person's name as query. For each query, print the phone number of that person.

**Input Format**

The first line will have an integer n denoting the number of entries in the phone book. Each entry consists of two lines: a name and the corresponding phone number.

After these, there will be some queries. Each query will contain a person's name. Read the queries until end-of-file.

**Constraints:**

A person's name consists of only lower-case English letters and it may be in the format 'first-name last-name' or in the format 'first-name'. Each phone number has exactly 8 digits without any leading zeros.

1≤n≤100000

1≤Query≤100000

**Output Format**

For each case, print "Not found" if the person has no entry in the phone book. Otherwise, print the person's name and phone number. See sample output for the exact format.

To make the problem easier, we provided a portion of the code in the editor. You can either complete that code or write completely on your own.

**Sample Input**

3

uncle sam

99912222

tom

11122222

harry

12299933

uncle sam

uncle tom

harry

Sample Output

uncle sam=99912222

Not found

harry=12299933

**Program:**

|  |
| --- |
| import java.util.\*;  import java.io.\*;  class JavaMap{  public static void main(String []argh)  {  Map database = new HashMap();  Scanner in = new Scanner(System.in);  int n=in.nextInt();  in.nextLine();  for(int i=0;i<n;i++)  {  String name=in.nextLine();  int phone=in.nextInt();  database.put(name,""+phone);  in.nextLine();  }  while(in.hasNext())  {  String s=in.nextLine();  String phone = (String)database.get(s);  System.out.println(phone==null?"Not found":s+"="+phone);  }  }  } |

* 1. In computer science, a double-ended queue (dequeue, often abbreviated to deque, pronounced deck) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail).

Deque interfaces can be implemented using various types of collections such as LinkedList or ArrayDeque classes. For example, deque can be declared as:

Deque deque = new LinkedList<>();

or

Deque deque = new ArrayDeque<>();

You can find more details about Deque here.

In this problem, you are given N integers. You need to find the maximum number of unique integers among all the possible contiguous subarrays of size M.

Note: Time limit is 3 second for this problem.

**Input Format**

The first line of input contains two integers N and M: representing the total number of integers and the size of the subarray, respectively. The next line contains N space separated integers.

**Constraints**

1≤N≤100000

1≤M≤100000

M≤N

The numbers in the array will range between [0,10000000].

**Output Format**

Print the maximum number of unique integers among all possible contiguous subarrays of size M separated by a space.

**Sample Input**

6 3

5 3 5 2 3 2

**Sample Output**

3

**Program**

|  |
| --- |
| import java.util.\*;  public class test {  public static void main(String[] args) {    Scanner in = new Scanner(System.in);  Deque deque = new ArrayDeque<Integer>();  int n = in.nextInt();  int m = in.nextInt();  int maxUnique = 0;  for (int i = 0; i < n; i++) {  int num = in.nextInt();  if(i == 0){  deque.add(num);  maxUnique++;  }else{  if(deque.size() == m){  deque.removeFirst();  }  if(!deque.contains(num) && maxUnique<m){  maxUnique++;  }  deque.addLast(num);  }  }  System.out.println(""+maxUnique);  }  } |