

<https://gist.github.com/likejazz/dc5b530affdc7a90aff34eea9bfa89ab>

ANs

import java.io.\_

import java.math.\_

import java.security.\_

import java.text.\_

import java.util.\_

import java.util.concurrent.\_

import java.util.function.\_

import java.util.regex.\_

import java.util.stream.\_

import scala.collection.immutable.\_

import scala.collection.mutable.\_

import scala.collection.concurrent.\_

import scala.collection.parallel.immutable.\_

import scala.collection.parallel.mutable.\_

import scala.concurrent.\_

import scala.io.\_

import scala.math.\_

import scala.sys.\_

import scala.util.matching.\_

import scala.reflect.\_

object Result {

/\*

\* Complete the 'minimumMoves' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY a

\* 2. INTEGER\_ARRAY m

\*/

def minimumMoves(a: Array[Int], m: Array[Int]): Int = {

var i=0; var j=0;var first=0;var second=0;var ai=0;var mi=0;var diff=0;var counter=0;

for(counter<-0 until a.length){

i=a(counter);

j=m(counter);

while(i>=1){

ai = i % 10;

mi = j % 10;

i = i/10;

j = j/10;

diff += abs(ai - mi);

println(i,j,abs(ai - mi))

}

}

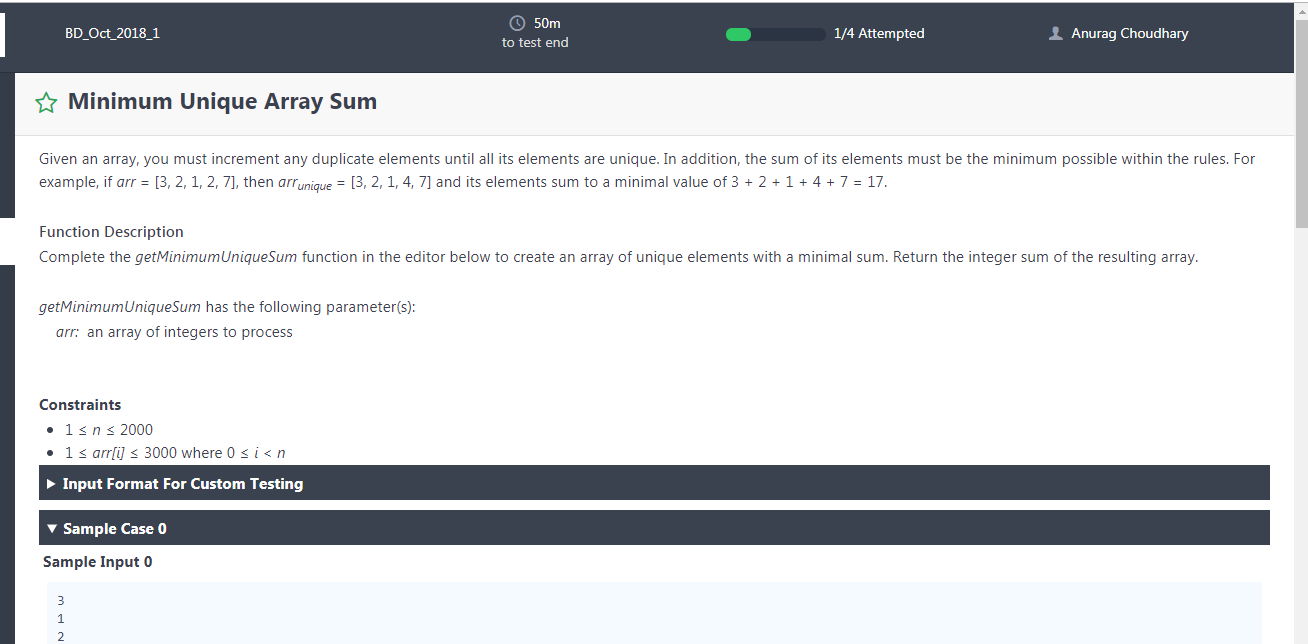
diff

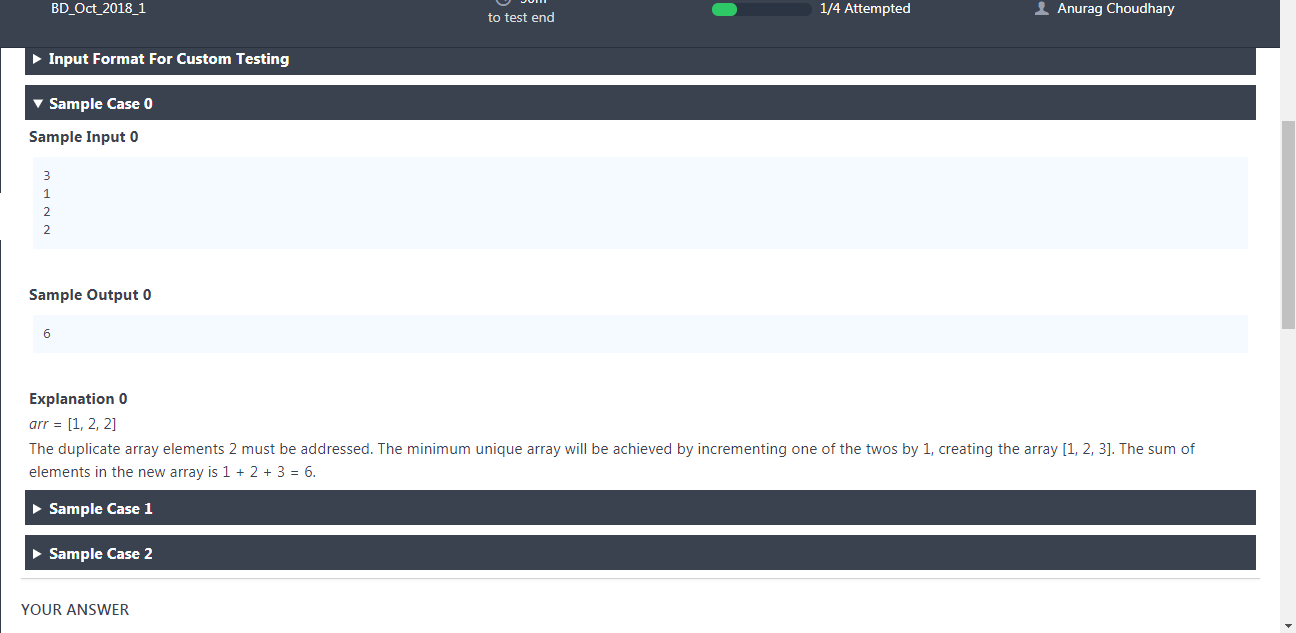
}

}

object Solution {

2.





Ans:--

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'getMinimumUniqueSum' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static int getMinimumUniqueSum(List<Integer> arr1) {

int[] arr = new int[arr1.size()];

for (int q = 0; q < arr.length; q++)

arr[q] = arr1.get(q);

int sum = arr[0];

for (int i = 1; i < arr.length; i++) {

if (arr[i] == arr[i - 1]) {

// While current element is same as

// previous or has become smaller

// than previous.

int j = i;

while (j < arr.length && arr[j] <= arr[j - 1]) {

arr[j] = arr[j] + 1;

j++;

}

}

sum = sum + arr[i];

}

return sum;

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int arrCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = IntStream.range(0, arrCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int result = Result.getMinimumUniqueSum(arr);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

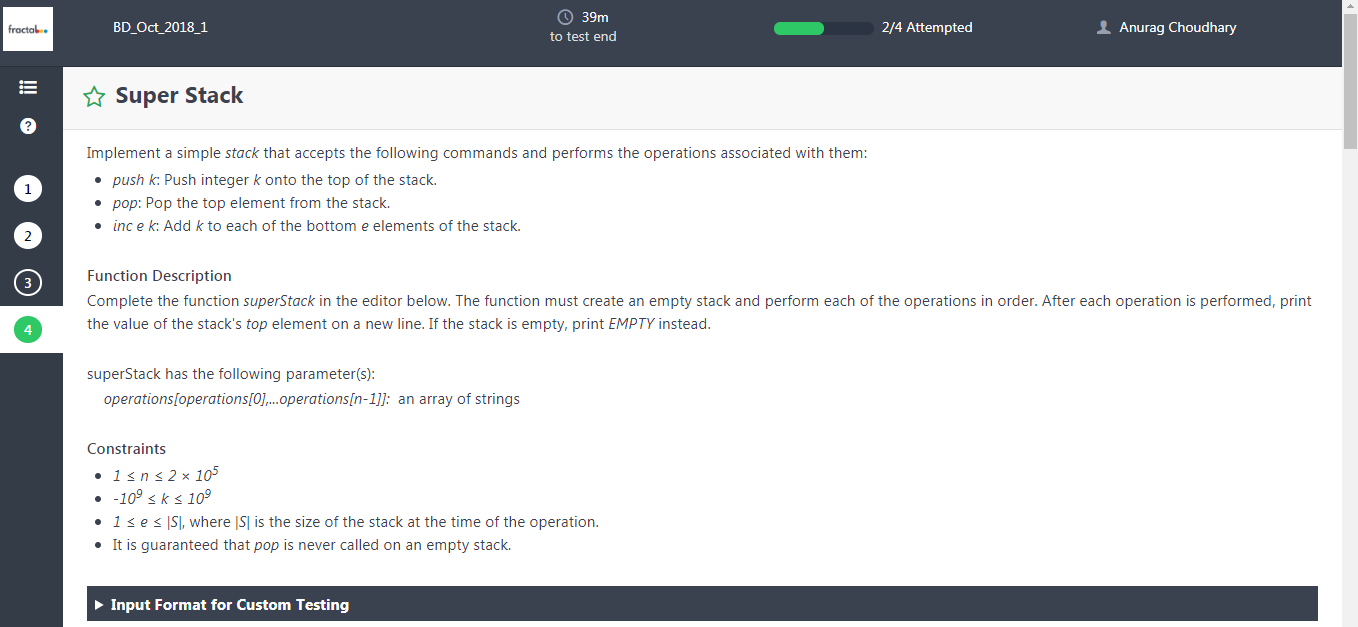
bufferedWriter.close();

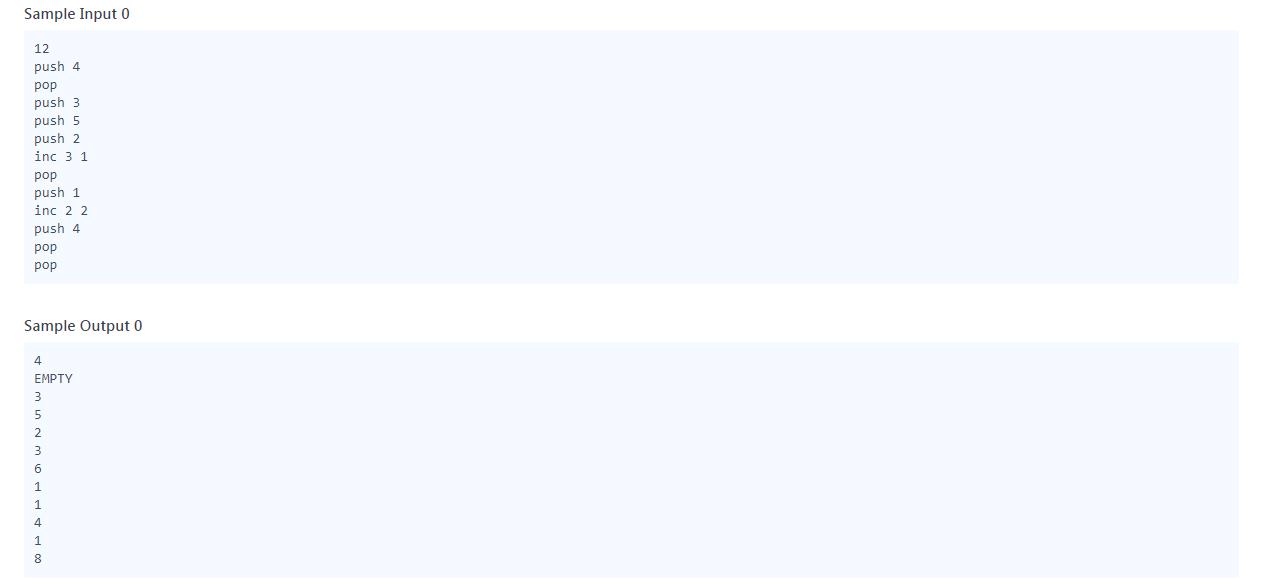
}

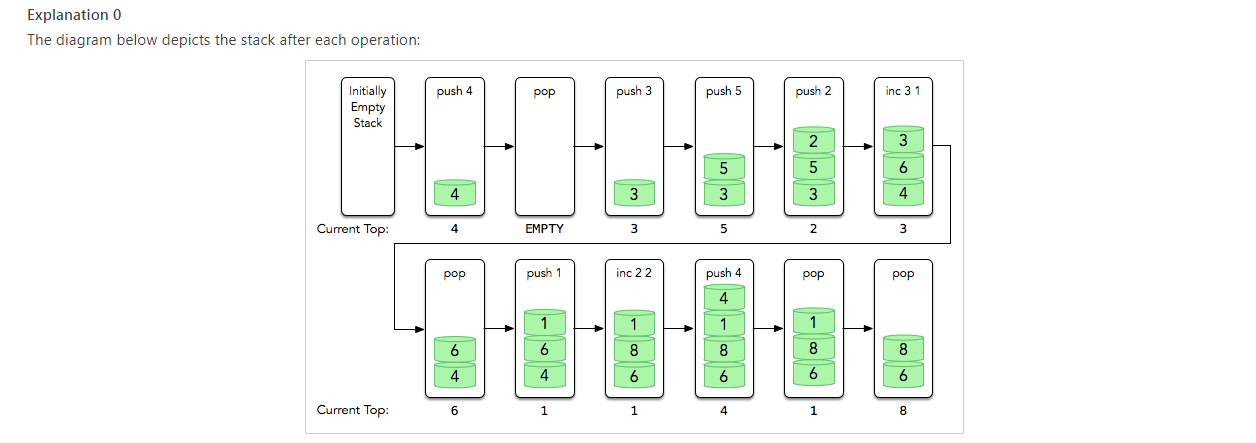
}

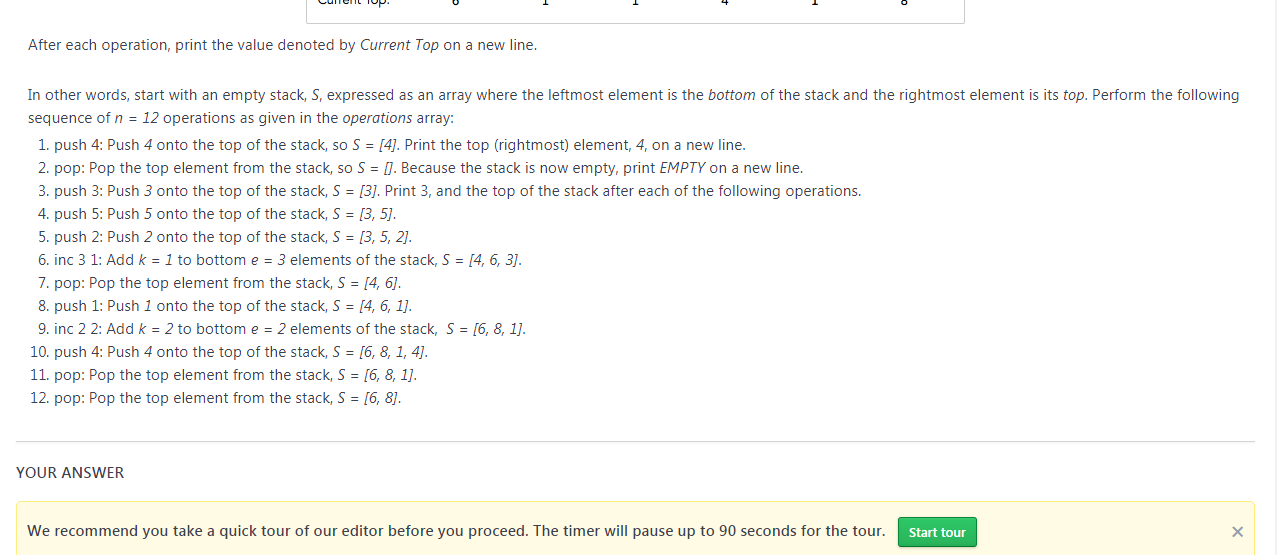
Many test case failed

3.











Ans:--4 connection due to timeout

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

/\*

\* Complete the function below.

\*/

static void superStack(String[] operations) {

if (operations == null || operations.length == 0) {

System.out.println("EMPTY");

return;

}

LinkedList<Integer> list = new LinkedList<>();

for (int i = 0; i < operations.length; i++) {

String current = operations[i];

if (current.equals("pop")) {

list.removeLast();

} else {

if (current.startsWith("push")) {

list.addLast(Integer.parseInt(current.split(" ")[1]));

} else {

int e = Integer.parseInt(current.split(" ")[1]);

int k = Integer.parseInt(current.split(" ")[2]);

ListIterator<Integer> listIterator = list.listIterator();

int j = 1;

while (listIterator.hasNext()) {

if (j > e)

break;

listIterator.set(listIterator.next() + k);

j++;

}

}

}

if (list.isEmpty())

System.out.println("EMPTY");

else

System.out.println(list.getLast());

}

}

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int operations\_size = 0;

operations\_size = Integer.parseInt(in.nextLine().trim());

String[] operations = new String[operations\_size];

for(int i = 0; i < operations\_size; i++) {

String operations\_item;

try {

operations\_item = in.nextLine();

} catch (Exception e) {

operations\_item = null;

}

operations[i] = operations\_item;

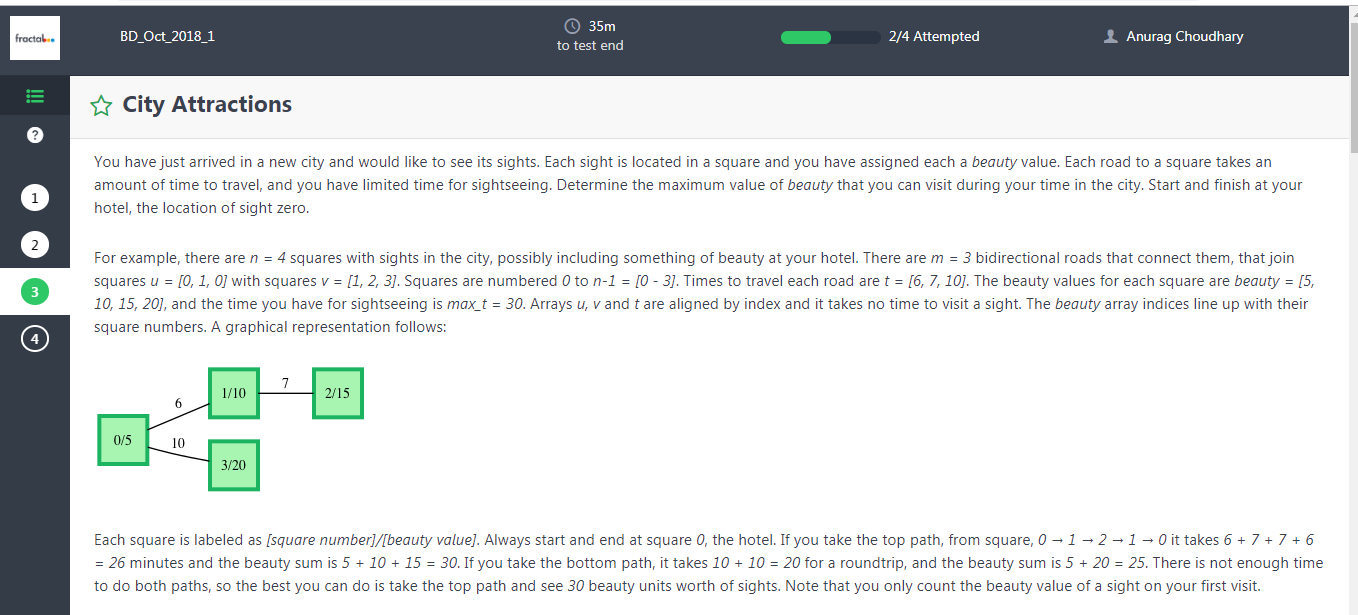
}

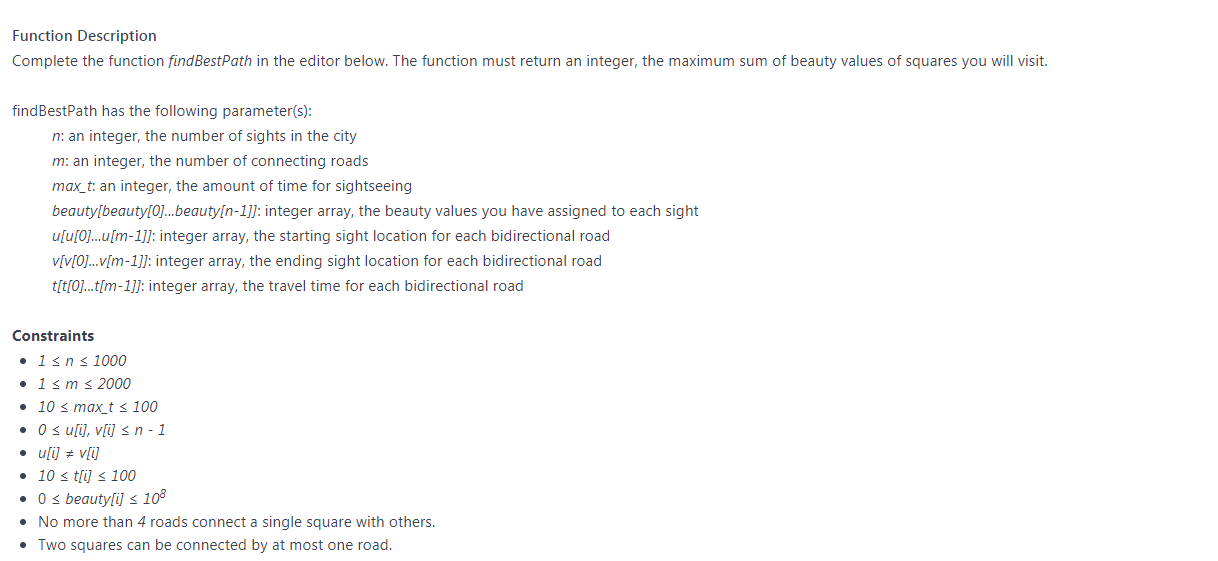
superStack(operations);

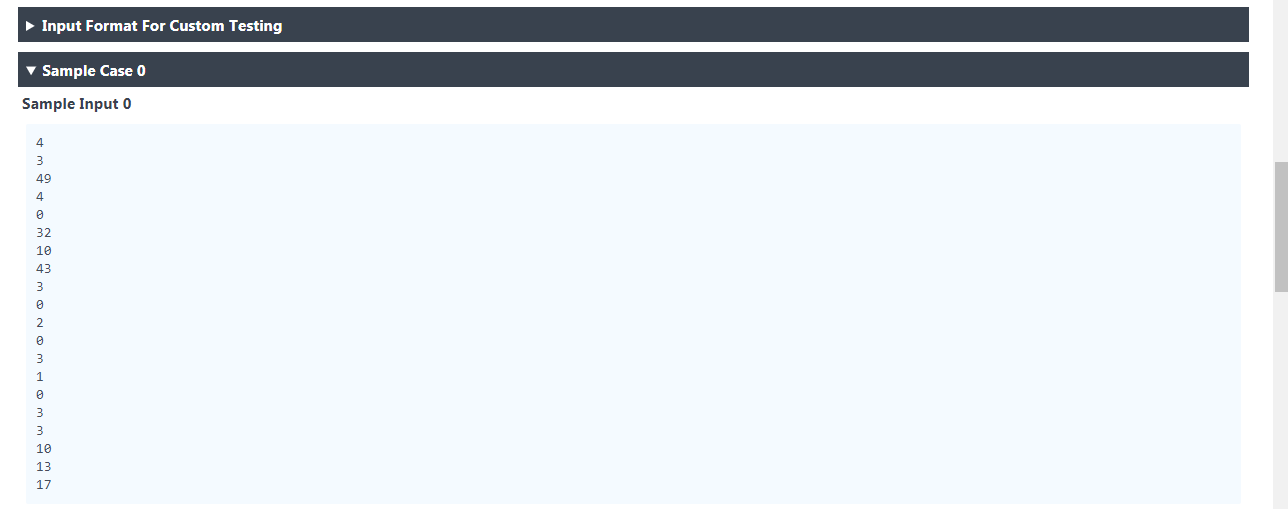
}

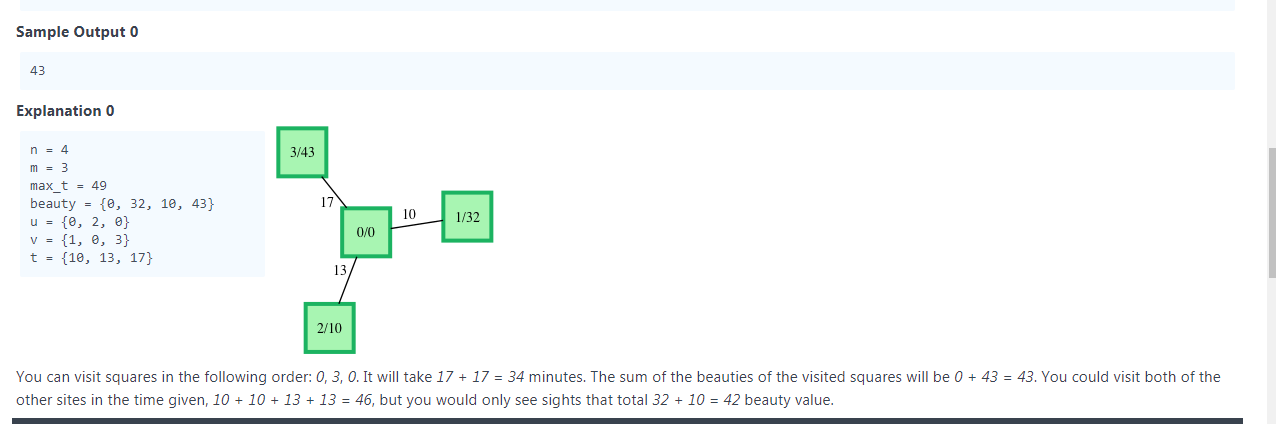
}

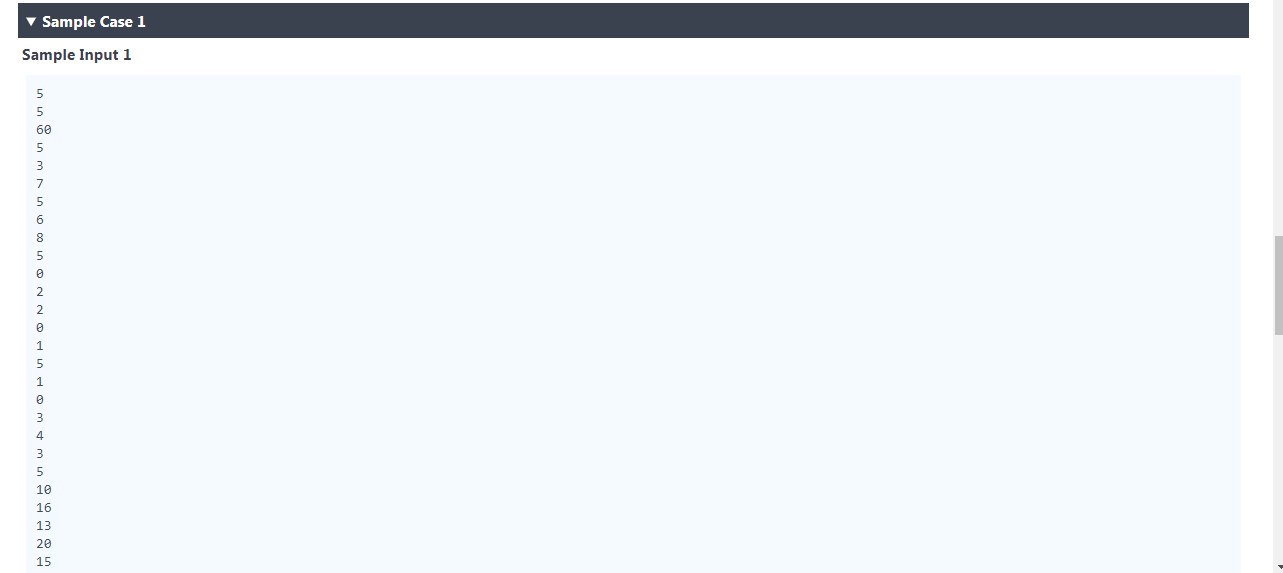
4.

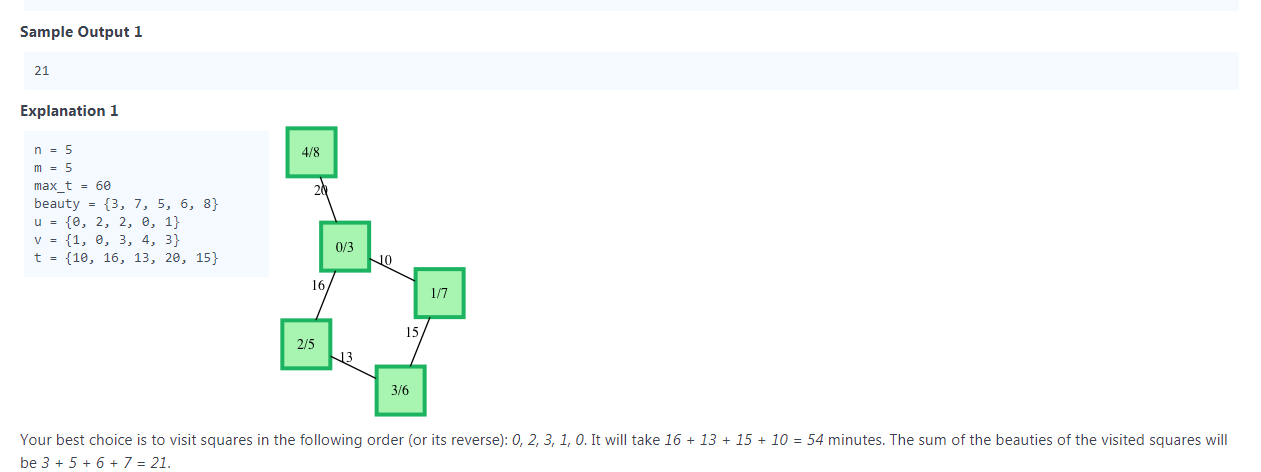


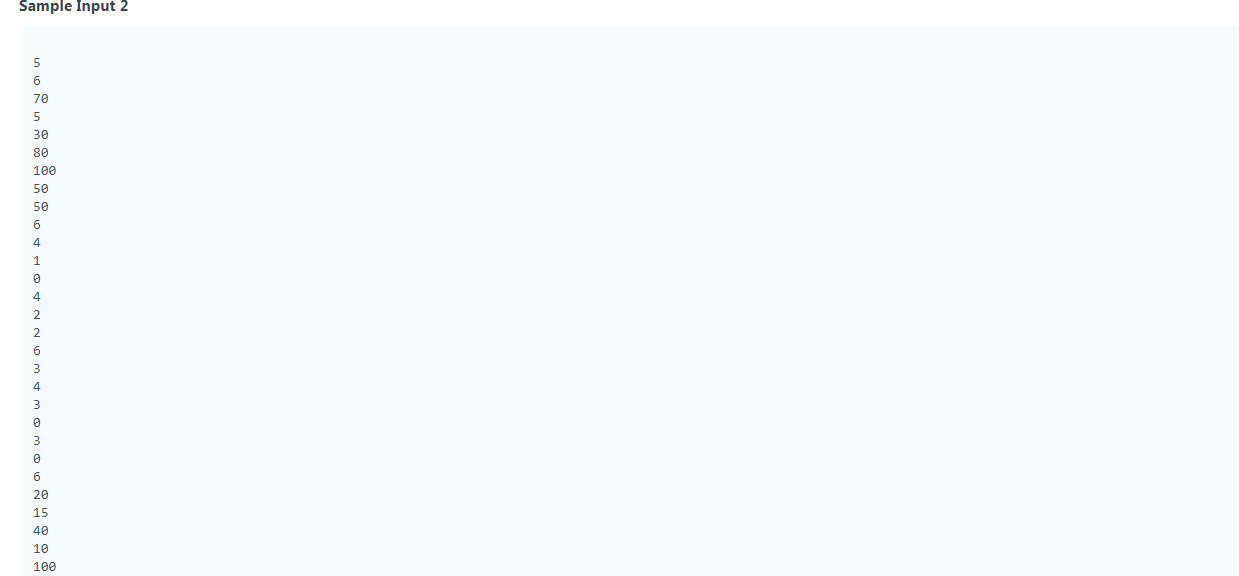


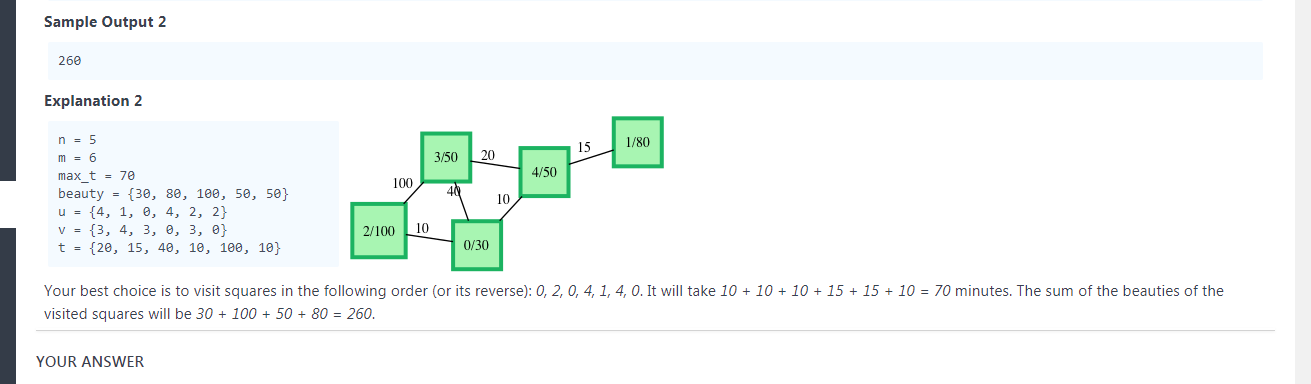














import java.io.\_

import java.math.\_

import java.security.\_

import java.text.\_

import java.util.\_

import java.util.concurrent.\_

import java.util.function.\_

import java.util.regex.\_

import java.util.stream.\_

import scala.collection.immutable.\_

import scala.collection.mutable.\_

import scala.collection.concurrent.\_

import scala.collection.parallel.immutable.\_

import scala.collection.parallel.mutable.\_

import scala.concurrent.\_

import scala.io.\_

import scala.math.\_

import scala.sys.\_

import scala.util.matching.\_

import scala.reflect.\_

object Result {

/\*

\* Complete the 'findBestPath' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER n

\* 2. INTEGER m

\* 3. INTEGER max\_t

\* 4. INTEGER\_ARRAY beauty

\* 5. INTEGER\_ARRAY u

\* 6. INTEGER\_ARRAY v

\* 7. INTEGER\_ARRAY t

\*/

def findBestPath(n: Int, m: Int, max\_t: Int, beauty: Array[Int], u: Array[Int], v: Array[Int], t: Array[Int]): Int = {

// Write your code here

}

}

object Solution {

def main(args: Array[String]) {

val printWriter = new PrintWriter(sys.env("OUTPUT\_PATH"))

val n = StdIn.readLine.trim.toInt

val m = StdIn.readLine.trim.toInt

val max\_t = StdIn.readLine.trim.toInt

val beautyCount = StdIn.readLine.trim.toInt

val beauty = Array.ofDim[Int](beautyCount)

for (i <- 0 until beautyCount) {

val beautyItem = StdIn.readLine.trim.toInt

beauty(i) = beautyItem

}

val uCount = StdIn.readLine.trim.toInt

val u = Array.ofDim[Int](uCount)

for (i <- 0 until uCount) {

val uItem = StdIn.readLine.trim.toInt

u(i) = uItem

}

val vCount = StdIn.readLine.trim.toInt

val v = Array.ofDim[Int](vCount)

for (i <- 0 until vCount) {

val vItem = StdIn.readLine.trim.toInt

v(i) = vItem

}

val tCount = StdIn.readLine.trim.toInt

val t = Array.ofDim[Int](tCount)

for (i <- 0 until tCount) {

val tItem = StdIn.readLine.trim.toInt

t(i) = tItem

}

val result = Result.findBestPath(n, m, max\_t, beauty, u, v, t)

printWriter.println(result)

printWriter.close()

}

}