**Data Structure and Algorithms**

(HackerEarth solved Quiz) 2022

Rakesh Keshari

AIMDek Trainee

Q 1) Monk and Rotation

https://www.hackerearth.com/problem/algorithm/monk-and-rotation-3-bcf1aefe/

Monk loves to preform different operations on arrays, and so being the principal of Hackerearth School, he assigned a task to his new student Mishki. Mishki will be provided with an integer array *A* of size *N* and an integer *K* , where she needs to rotate the array in the right direction by K steps and then print the resultant array. As she is new to the school, please help her to complete the task.

Java source code:

import java.util.\*;

class TestClass {

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

Scanner sc = new Scanner(System.in);

int T = sc.nextInt();

int size = sc.nextInt();

int rot = sc.nextInt();

int [] arr = new int [size];

for(int i=0; i<size; i++){

arr[i] = sc.nextInt();

}

for(int i = 0; i < rot; i++){

int j, last;

last = arr[arr.length-1];

for(j = size-1; j > 0; j--){

//Shift element of array by one

arr[j] = arr[j-1];

}

arr[0] = last;

}

//Displays resulting array after rotation

for(int i = 0; i< size; i++){

System.out.print(arr[i] + " ");

}

}

}

Q 2) Monk and Inversion

Monk's best friend Micro, who happen to be an awesome programmer, got him an integer matrix *M* of size *N×N* for his birthday. Monk is taking coding classes from Micro. They have just completed array inversions and Monk was successful in writing a program to count the number of inversions in an array. Now, Micro has asked Monk to find out the number of inversion in the matrix *M*. Number of inversions, in a matrix is defined as the number of unordered pairs of cells *{(i,j),(p,q)}* such that *M[i][j]>M[p][q] & i≤p & j≤q*.  
Monk is facing a little trouble with this task and since you did not got him any birthday gift, you need to help him with this task.

Java Source Code:

import java.util.\*;

class TestClass {

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

Scanner sc = new Scanner(System.in);

int t = sc.nextInt();

while (t != 0){

int n = sc.nextInt();

int[][] a = new int[n][n];

for (int i = 0; i < n; i++){

for (int j = 0; j < n; j++){

a[i][j] = sc.nextInt();

}

}

int inversions = 0;

for (int i = 0; i < n; i++){

for (int l = 0; l < n; l++){

for (int j = 0; j <= i; j++){

for (int k = 0; k <= l; k++){

if (a[i][l] < a[j][k]){

inversions++;

}

}

}

}

}

System.out.println(inversions);

t--;

}

}

}

Q 3) Cyclic Shift

A large binary number is represented by a string *A* of size *N* and comprises of *0s* and *1s*. You must perform a cyclic shift on this string. The cyclic shift operation is defined as follows:

* If the string *A* is *[A0,A1,A2,...,AN−1]*, then after performing one cyclic shift, the string becomes *[A1,A2,...,AN−1,A0]*.

You performed the shift infinite number of times and each time you recorded the value of the binary number represented by the string. The maximum binary number formed after performing (possibly *0*) the operation is *B*. Your task is to determine the number of cyclic shifts that can be performed such that the value represented by the string *A* will be equal to *B* for the *Kth* time.

Java source code:

import java.io.\*;

import java.util.\*;

class TestClass {

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

Scanner sc = new Scanner(System.in);

int T = sc.nextInt();

while(T-- > 0){

int N; int K;

N = sc.nextInt();

K = sc.nextInt();

String input = sc.next();

String B = "";

String inter = input;

int d = 0;

int period = -1;

for(int i = 0; i < N;i++){

if (B.compareTo(inter) < 0){

B = inter;

d = i;

}else if (B.compareTo(inter) == 0){

period = i - d;

break;

}

inter = inter.substring(1, inter.length()) + inter.substring(0, 1);

}

if(period == -1){

System.out.println(d + (K - 1L ) \* N);

}else{

System.out.println(d + ((K - 1L) \* period));

}

}

}

}

Q 4) Monk and Nice Strings

Monk's best friend Micro's birthday is coming up. Micro likes Nice Strings very much, so Monk decided to gift him one. Monk is having *N* nice strings, so he'll choose one from those. But before he selects one, he need to know the Niceness value of all of those. Strings are arranged in an array *A*, and the Niceness value of string at position *i* is defined as the number of strings having position less than *i* which are lexicographicaly smaller than *A[i]*. Since nowadays, Monk is very busy with the Code Monk Series, he asked for your help.

Java source code:

import java.util.\*;

class TestClass {

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

Scanner sc = new Scanner(System.in);

int n = scan.nextInt();

String[] s = new String[1010];

int j = 0;

s[0] = "A";

for(int i = 1 ; i <= n ; i++) {

String str = "";

str = sc.next();

for(j = i - 1 ; j >= 0 ; j--) {

if(s[j].compareTo(str) >= 0) {

s[j+1] = s[j];

}

else {

break;

}

}

System.out.println(j);

s[j+1] = str;

}

}

}

Q 5) Monk and Suffix Sort

Monk loves to play games. On his birthday, his friend gifted him a string *S*. Monk and his friend started playing a new game called as Suffix Game. In Suffix Game, Monk's friend will ask him lexicographically *kth* smallest suffix of the string *S*. Monk wants to eat the cake first so he asked you to play the game.

Java source code:

import java.util.\*;

class TestClass {

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

Scanner sc = new Scanner(System.in);

String S = sc.next();

int k=sc.nextInt();

String[] output=new String[S.length()];

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

output[i]=S.substring(i);

}

Arrays.sort(output);

System.out.println(output[k-1]);

}

}