/\*

Merge list

Implement a queue using linked list\*/

He tin chodkar baki sab he

//Duplicate 1

import java.util.HashMap;

import java.util.Map;

class Solution

{

public static boolean hasDuplicate(int[] nums, int k)

{

Map<Integer, Integer> map = new HashMap<>();

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

{

if (map.containsKey(nums[i]))

{

// return true if the current element repeats within range of `k`

if (i - map.get(nums[i]) <= k) {

return true;

}

}

map.put(nums[i], i);

}

return false;

}

public static void main(String[] args)

{

int[] nums = { 5, 6, 8, 2, 4, 6, 9 };

int k = 4;

if (hasDuplicate(nums, k)) {

System.out.println("Duplicates found");

}

else {

System.out.println("No duplicates were found");

}

}

}

—--------------------------------

//2. :Smallest missing positive number

import java.io.\*;

import java.util.\*;

public class Solution {

public static int findSmallestMissing(int[] nums, int left, int right)

{

// base condition

if (left > right) {

return left;

}

int mid = left + (right - left) / 2;

// if the mid-index matches with its value, then the mismatch

// lies on the right half

if (nums[mid] == mid) {

return findSmallestMissing(nums, mid + 1, right);

}

else {

// mismatch lies on the left half

return findSmallestMissing(nums, left, mid - 1);

}

}

public static void main(String[] args)

{

int[] nums = {0, 1, 2, 6, 9, 11, 15};

int left = 0, right = nums.length - 1;

System.out.println("The smallest missing element is "

+ findSmallestMissing(nums, left, right));

}

}

—--------------------------------

//3 Bubble Sort

import java.io.\*;

import java.util.\*;

public class Solution {

public static void swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

public static void bubbleSort(int[] arr)

{

for (int k = 0; k < arr.length - 1; k++)

{

// last `k` items are already sorted, so the inner loop can

// avoid looking at the last `k` items

for (int i = 0; i < arr.length - 1 - k; i++)

{

if (arr[i] > arr[i + 1]) {

swap(arr, i, i + 1);

}

}

// the algorithm can be terminated if the inner loop

// didn't do any swap

}

}

public static void main(String[] args)

{

int[] arr = { 3, 5, 8, 4, 1, 9, -2 };

bubbleSort(arr);

// print the sorted array

System.out.println(Arrays.toString(arr));

}

}

—--------------------------------

//4 Print the Elements of a Linked List

static void printLinkedList(SinglyLinkedListNode head) {

SinglyLinkedListNode n=head;

while (n!=null)

{

System.out.println(n.data);

n=n.next;

}

}

—--------------------------------

//5 Insert a node at the head of a linked list

static SinglyLinkedListNode insertNodeAtHead(SinglyLinkedListNode llist, int data) {

SinglyLinkedListNode temp;

if(llist == null){

llist = new SinglyLinkedListNode(data);

return llist;

}

temp = new SinglyLinkedListNode(data);

temp.next = llist;

llist = temp;

return llist;

}

—--------------------------------

// 6 insert a Node at the Tail of a Linked List

static SinglyLinkedListNode insertNodeAtTail(SinglyLinkedListNode head, int data) {

SinglyLinkedListNode newNode = new SinglyLinkedListNode(data);

if(head == null){

return newNode;

}

SinglyLinkedListNode cur = head;

while(cur.next != null){

cur = cur.next;

}

cur.next = newNode;

return head;

}

—--------------------------------

// 7 insert a node at a specific position in a linked list

class Result {

}

public static SinglyLinkedListNode insertNodeAtPosition(SinglyLinkedListNode llist, int data, int position) {

// Write your code here

SinglyLinkedListNode node = new SinglyLinkedListNode(data);

if(position==0){

node.next = llist.next;

llist=node;

}else{

SinglyLinkedListNode aux = new SinglyLinkedListNode(0);

aux.next = llist;

for(int i=0;i<position;i++)aux=aux.next;

node.next = aux.next;

aux.next=node;

}

return llist;

}

—--------------------------------

//8 delete node

class Result {

public static SinglyLinkedListNode deleteNode(SinglyLinkedListNode llist, int position) {

// Write your code here

int currentNodePosition = 0;

SinglyLinkedListNode head = llist;

SinglyLinkedListNode currentNode = llist;

if (position == 0) {

head = head.next;

return head;

}

while (currentNodePosition < position - 1) {

currentNode = currentNode.next;

currentNodePosition++;

}

if (currentNode.next != null && currentNode.next.next != null) {

currentNode.next = currentNode.next.next;

}

return head;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// 9 Delete duplicate-value nodes from a sorted linked list

public static SinglyLinkedListNode removeDuplicates(SinglyLinkedListNode llist) {

// Write your code here

SinglyLinkedListNode temp = llist;

while(temp.next!=null)

{

if(temp.data == temp.next.data)

{

temp.next = temp.next.next;

}

else

{

temp = temp.next;

}

}

return llist;

}

class Result {

}

—--------------------------------

// 10 Merge two sorted linked lists

static SinglyLinkedListNode mergeLists(SinglyLinkedListNode head1, SinglyLinkedListNode head2) {

if(head1==null) {

return head2;

}

if(head2 == null) {

return head1;

}

SinglyLinkedListNode t1 = head1, t2 = head2;

SinglyLinkedListNode head = null, tail = null;

if(t1.data<=t2.data) {

head = t1;

tail = t1;

t1= t1.next;

} else {

head = t2;

tail = t2;

t2 = t2.next;

}

while(t1!=null && t2!=null) {

if(t1.data<=t2.data) {

tail.next = t1;

tail = t1;

t1 = t1.next;

} else {

tail.next = t2;

tail = t2;

t2 = t2.next;

}

}

if(t1!=null) {

tail.next = t1;

} else {

tail.next = t2;

}

return head;

}

—--------------------------------

// 11 Find Merge Point of Two Lists

static int findMergeNode(SinglyLinkedListNode head1, SinglyLinkedListNode head2) {

SinglyLinkedListNode temp1=head1;

SinglyLinkedListNode temp2=head2;

List<SinglyLinkedListNode>list=new ArrayList<SinglyLinkedListNode>();

while(temp1!=null){

list.add(temp1);

temp1=temp1.next;

}

while(temp2!=null){

if(list.contains(temp2)){

break;

}

temp2=temp2.next;

}

return temp2.data;

}

—--------------------------------

//12 merge list

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

private static MyScanner sc;

private static PrintWriter out;

public static void main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

sc = new MyScanner();

out = new PrintWriter(System.out);

int t = sc.nextInt();

for(int i = 0; i < t; i++)

{

int a = sc.nextInt();

int b = sc.nextInt();

out.println(bin(a + b, a, 1000000007));

}

out.flush();

out.close();

}

public static long bin(int N, int K, int p) {

long[][] binomial = new long[N + 1][K + 1];

for (int k = 1; k <= K; k++) binomial[0][k] = 0;

for (int n = 0; n <= N; n++) binomial[n][0] = 1;

for (int n = 1; n <= N; n++)

for (int k = 1; k <= K; k++)

binomial[n][k] = (binomial[n-1][k-1] + binomial[n-1][k]) % p;

return binomial[N][K];

}

public static class MyScanner

{

BufferedReader br;

StringTokenizer st;

public MyScanner()

{

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

}

String next()

{

while (st == null || !st.hasMoreElements())

{

try

{

st = new StringTokenizer(br.readLine());

} catch (IOException e)

{

e.printStackTrace();

}

}

return st.nextToken();

}

int nextInt()

{

return Integer.parseInt(next());

}

long nextLong()

{

return Long.parseLong(next());

}

double nextDouble()

{

return Double.parseDouble(next());

}

String nextLine()

{

String str = "";

try

{

str = br.readLine();

} catch (IOException e)

{

e.printStackTrace();

}

return str;

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// 13 print in Reverse

static void reversePrint(SinglyLinkedListNode head) {

if(head.next != null) {

reversePrint(head.next);

}

System.out.println(head.data);

}