Dequeue efficient queue using stack

Implement a Queue using two stacks Make it Dequeue efficient.

Input Format

Enter the size of the queue N add 0 - N-1 numbers in the queue

Constraints

Output Format

Display the numbers in the order they are dequeued and in a space separated manner

Sample Input

```
5
```

Sample Output

```
0 1 2 3 4
```

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        Stack<Integer> st = new Stack<Integer>();
        for(int i=0; i<n; i++){
            st.push(i);
        }
        Stack<Integer> st1 = new Stack<Integer>();
        while(st.size() > 0){
            st1.push(st.pop());
        }

        while(st1.size() > 0){
            System.out.print(st1.pop()+" ");
        }
    }
}
```

Kartik Bhaiya And The Celebrity Problem

Kartik Bhaiya, mentor at Coding Blocks, organized a party for their interns at Coding Blocks. In a party of N people, only one person is known to everyone. Such a person may be present in the party, if yes, she/he doesn't know anyone in the party. We can only ask questions like "does A know B? ".
Find the stranger (celebrity) in minimum number of questions.

Input Format

First line contains **N**, number of persons in party. Next line contains the matrix of N x N which represents A knows B when it's value is one.

Constraints

```
None
```

Output Format

Print the celebrity ID which is between 0 and N-1. If celebrity is not present then print "No Celebrity".

Sample Input

```
4
0 0 1 0
0 0 1 0
0 0 0 0
0 0 1 0
```

Sample Output

```
2
```

Explanation

In the given case there are 4 persons in the party let them as A, B, C, D. The matrix represents A knows B when it's value is one. From the matrix, A knows C, B knows C and D knows C. Thus C is the celebrity who doesnot know anyone and it's ID is 2. Hence output is 2.

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int [n];
        int [][] celebrity = new int[n][n];
        for(int i=0; i<n;i++){
            for(int j=0; j<n; j++){
                celebrity[i][j] = sc.nextInt();
                if(celebrity[i][j] == 1 && i != j){
                    arr[j]++;
                }
            }
        }

        int id = -1;
        for(int i=0; i<arr.length;i++){
            if(arr[i] == n-1){
                id = i;
            }
        }

        if(id > 0){
            System.out.println(id);
        }
        else{
            System.out.println("No Celebrity");
        }
    }
}
```

Stock Span

The stock span problem is a financial problem where we have a series of N daily price quotes for a stock and we need to calculate span of stock's price for all N days. You are given an array of length N, where ith element of array denotes the price of a stock on ith. Find the span of stock's price on ith day, for every 1<=i<=N.
A span of a stock's price on a given day, i, is the maximum number of consecutive days before the (i+1)th day, for which stock's price on these days is less than or equal to that on the ith day.

Input Format

First line contains integer N denoting size of the array.
Next line contains N space separated integers denoting the elements of the array.

Constraints

```
1 <= N <= 10^6
```

Output Format

Display the array containing stock span values.

Sample Input

```
5
30
35
40
38
35
```

Sample Output

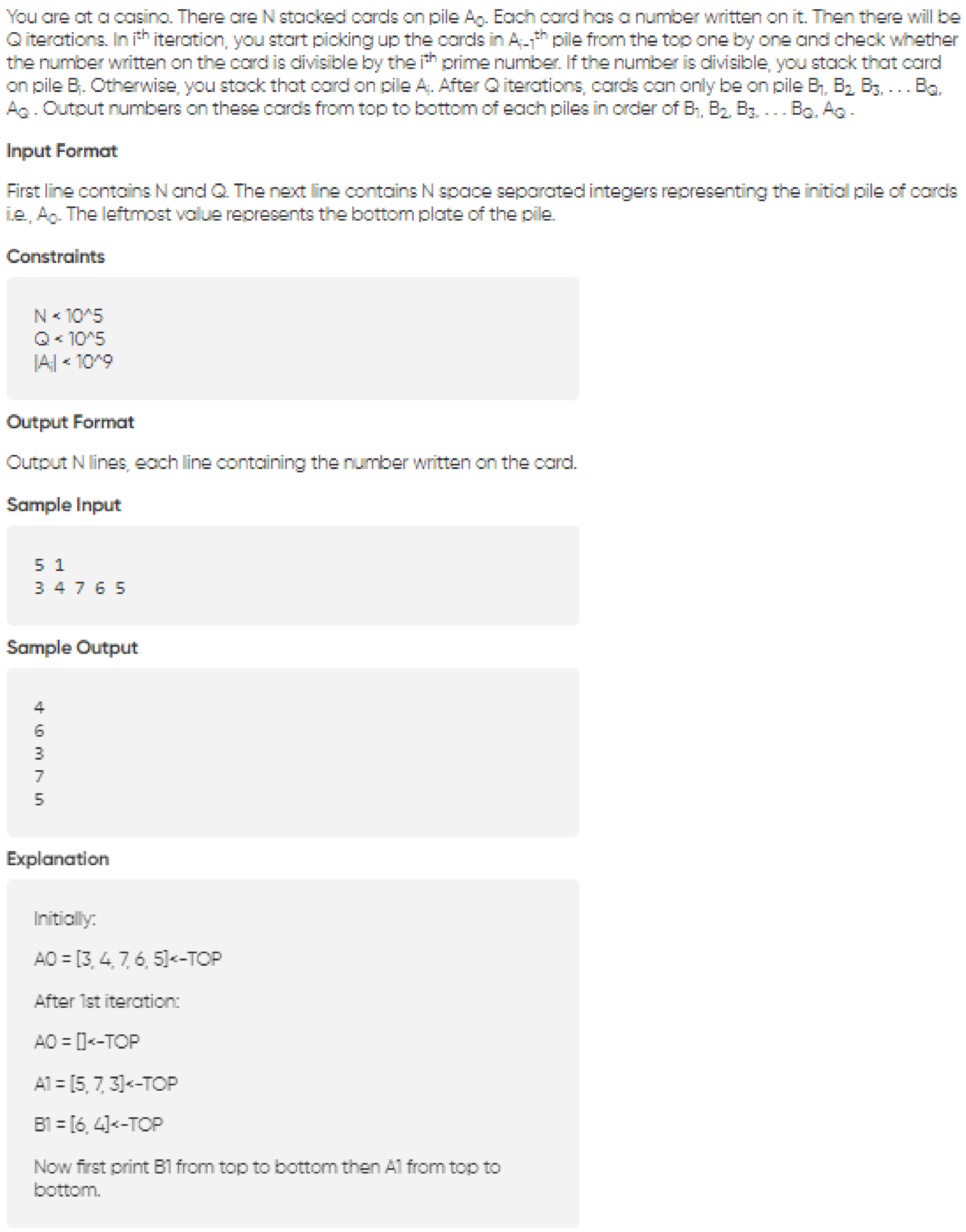
```
1 2 3 1 1 END
```

Explanation

```
For the given case
for day1 stock span =1
for day2 stock span =2 (as 35>30 so both days are included in
it)
for day3 stock span =3 (as 40>35 so 2+1=3)
for day4 stock span =1 (as 38<40 so only that day is included)
for day5 stock span =1 (as 35<38 so only that day is included)
hence output is 1 2 3 1 1 END
```

```
import java.util.*;
public class Main {

    public static void main(String args[])  {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int[n];
        Stack<Integer>s = new Stack<Integer>();
        for(int i=0; i<n;i++){
            arr[i] = sc.nextInt();
        }

        int [] answer = new int[n];
        for(int i=0; i<n; i++){
            while(!s.isEmpty() && arr[i] >=
arr[s.peek()]){
                s.pop();
            }

            if(s.isEmpty()){
                answer[i] = i+1;
            }
            else{
                answer[i] = i - s.peek();
            }
            s.push(i);
        }

        for(int i=0; i<n; i++){
            System.out.print(answer[i]+" ");
        }
        System.out.print("END");

    }
}
```

Playing with cards (In stack)



You are at a casino. There are N stacked cards on pile A_Q. Each card has a number written on it. Then there will be Q iterations. In i^{th} iteration, you start picking up the cards in $A_{i-1}{}^{th}$ pile from the top one by one and check whether the number written on the card is divisible by the i^{th} prime number. If the number is divisible, you stack that card on pile B_i. Otherwise, you stack that card on pile A_i. After Q iterations, cards can only be on pile B_1, B_2, B_3, . . . B_Q, A_Q . Output numbers on these cards from top to bottom of each piles in order of B_1, B_2, B_3, . . . B_Q, A_Q .

Input Format

First line contains N and Q. The next line contains N space separated integers representing the initial pile of cards i.e., A_Q. The leftmost value represents the bottom plate of the pile.

Constraints

```
N < 10^5
Q < 10^5
|Ai| < 10^9
```

Output Format

Output N lines, each line containing the number written on the card.

Sample Input

```
5 1
3 4 7 6 5
```

Sample Output

```
4
6
3
7
5
```

Explanation

Initially:

A0 = [3, 4, 7, 6, 5]<-TOP

After 1st iteration:

A0 = []<-TOP

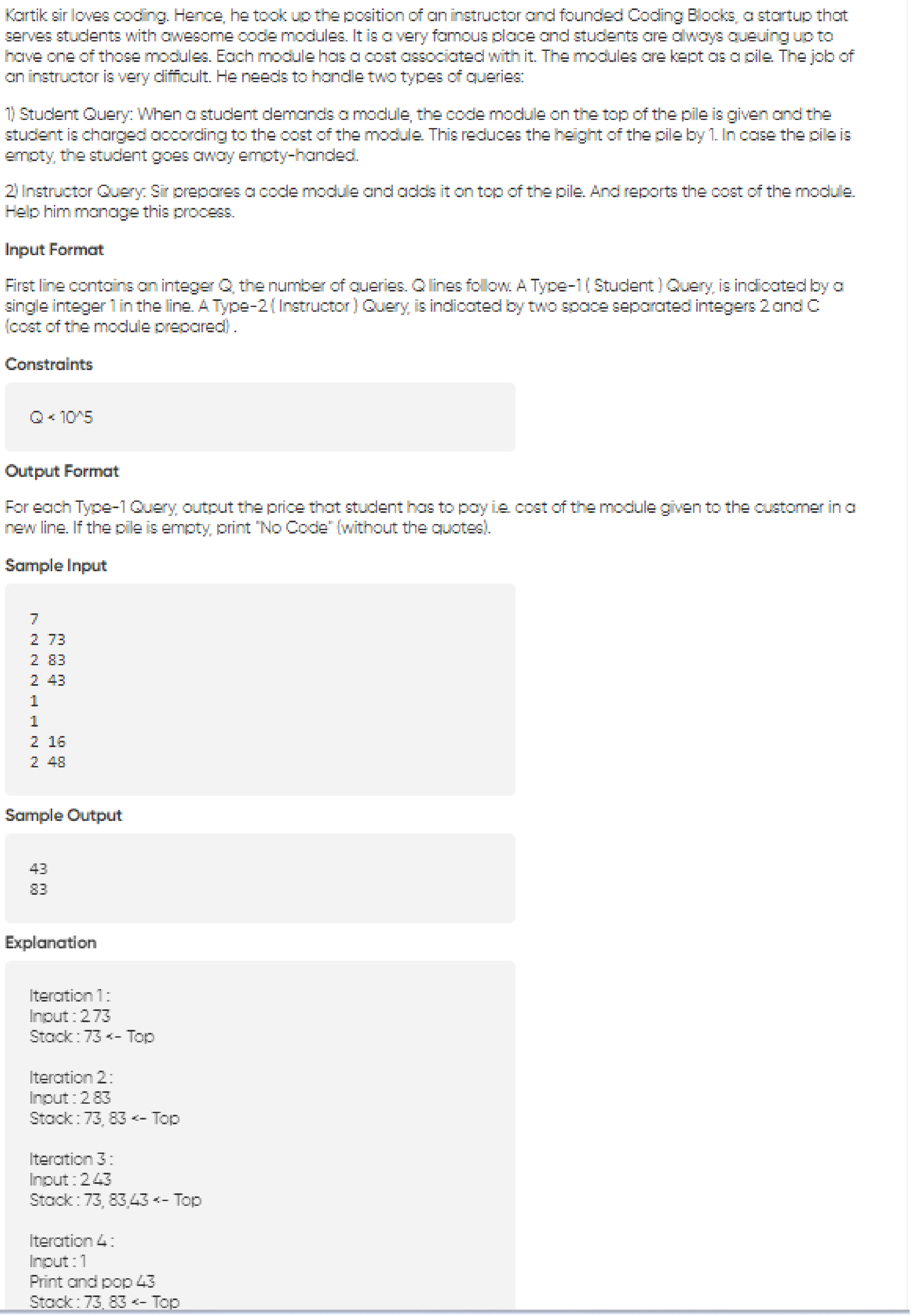
A1 = [5, 7, 3]<-TOP

B1 = [6, 4]<-TOP

Now first print B1 from top to bottom then A1 from top to bottom.

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int q = sc.nextInt();
        Stack<Integer>st = new Stack<Integer>();
        for(int i=0; i<n; i++){
            st.push(sc.nextInt());
        }
        int [] arr = new int[q];
        int i = 0;
        int num = 2;
        while(i < q){
            boolean pr = true;
            for(int j=2; j<num; j++){
                if(num%j == 0){
                    pr = false;
                    break;
                }
            }
            if(pr){
                arr[i] = num;
                i++;
            }
            num++;
        }
        i=0;
        Stack<Integer>b = new Stack<Integer>();
        Stack<Integer>a1 = new Stack<Integer>();
        Stack<Integer>b1 = new Stack<Integer>();
        while(i < q){
            if(i%2==0){
                while(!b1.isEmpty()){
                    b.push(b1.pop());
                }
                while(!st.isEmpty()){
                    if(st.peek() % arr[i] == 0){
                        b1.push(st.pop());
                    }
                    else{
                        a1.push(st.pop());
                    }
                }

                while(!b.isEmpty()){
                    b1.push(b.pop());
                }
            }
            else{
                while(!b1.isEmpty()){
                    b.push(b1.pop());
                }
                while(!a1.isEmpty()){
                    if(a1.peek() % arr[i] == 0){
                        b1.push(a1.pop());
                    }
                    else{
                        st.push(a1.pop());
                    }
                }

                while(!b.isEmpty()){
                    b1.push(b.pop());
                }
            }
            i++;
        }

        while(!b.isEmpty()){
            System.out.println(b.pop());
        }
        while(!b1.isEmpty()){
            System.out.println(b1.pop());
        }
        while(!a1.isEmpty()){
            System.out.println(a1.pop());
        }
        while(!st.isEmpty()){
            System.out.println(st.pop());
        }
    }
}
```

Kartik Sir and Coding



Kartik sir loves coding. Hence, he took up the position of an instructor and founded Coding Blocks, a startup that serves students with awesome code modules. It is a very famous place and students are always queuing up to have one of those modules. Each module has a cost associated with it. The modules are kept as a pile. The job of an instructor is very difficult. He needs to handle two types of queries:

1) Student Query: When a student demands a module, the code module on the top of the pile is given and the student is charged according to the cost of the module. This reduces the height of the pile by 1. In case the pile is empty, the student goes away empty-handed.

2) Instructor Query: Sir prepares a code module and adds it on top of the pile. And reports the cost of the module. Help him manage this process.

Input Format

First line contains an integer Q, the number of queries. Q lines follow. A Type-1 (Student) Query, is indicated by a single integer 1 in the line. A Type-2 (Instructor) Query, is indicated by two space separated integers 2 and C (cost of the module prepared) .

Constraints

```
Q < 10^5
```

Output Format

For each Type-1 Query, output the price that student has to pay i.e. cost of the module given to the customer in a new line. If the pile is empty, print "No Code" (without the quotes).

Sample Input

```
7
2 73
2 83
2 43
1
1
2 16
2 48
```

Sample Output

```
43
83
```

Explanation

```
Iteration 1 :
Input : 2 73
Stack : 73 <- Top

Iteration 2 :
Input : 2 83
Stack : 73, 83 <- Top

Iteration 3 :
Input : 2 43
Stack : 73, 83,43 <- Top

Iteration 4 :
Input : 1
Print and pop 43
Stack : 73, 83 <- Top
```

```
import java.util.*;
public class Main {
    static Scanner s = new Scanner(System.in);
     public static void main(String args[]) throws Exception
{
          // Your Code Here


          int q = s.nextInt();
         Main obj = new Main();
         StacksUsingArrays stack = obj.new
StacksUsingArrays();
     Calculate(stack, q);
     }

     public static void Calculate(StacksUsingArrays stack,
int q) throws Exception {

        //Write Your Code Here
        /* Donot initialize another Scanner use the static
scanner already declared*/
         for(int i=0; i<q;i++){
              int q1 = s.nextInt();
              if(q1 == 1){
                   if(stack.isEmpty()){
                        System.out.println("No Code");
                   }
                   else{
                          System.out.println(stack.pop());
                   }
              }
              else{
                   stack.push(s.nextInt());
              }
         }
     }

     private class StacksUsingArrays {
          private int[] data;
          private int tos;

          public static final int DEFAULT_CAPACITY = 10;
          public StacksUsingArrays() throws Exception {
               // TODO Auto-generated constructor stub
               this(DEFAULT_CAPACITY);
          }

          public StacksUsingArrays(int capacity) throws
Exception {
               if (capacity <= 0) {
                    System.out.println("Invalid Capacity");
               }
               this.data = new int[capacity];
               this.tos = -1;
          }

          public int size() {
               return this.tos + 1;
          }

          public boolean isEmpty() {
               if (this.size() == 0) {
                    return true;
               } else {
                    return false;
               }
          }

          public void push(int item) throws Exception {
               if (this.size() == this.data.length) {

                   int[] temp = new int[2 * data.length];
                   for(int i = 0;i < data.length;i++){
                       temp[i] = data[i];
                   }

                   data = temp;
               }
               this.tos++;
               this.data[this.tos] = item;
          }

          public int pop() throws Exception {
                if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
                }
                int retVal = this.data[this.tos];
                this.data[this.tos] = 0;
                this.tos--;
                return retVal;
            }

            public int top() throws Exception {
                if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
                }
                int retVal = this.data[this.tos];
                return retVal;
            }

            public void display() throws Exception {
                if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
                }
                for (int i = this.tos; i >= 0; i--) {
                    System.out.println(this.data[i]);
                }

            }

        }

}
```

Balanced Parenthesis

You are given a string of brackets i.e. '{', '}' , '(' , ')', '[' , ']' . You have to check whether the sequence of parenthesis is balanced or not.
For example, "(())", "(())()" are balanced and "())(", "(()))" are not.

Input Format

A string of '(' , ')' , '{' , '}' and '[' , ']' .

Constraints

```
1<=|S|<=10^5
```

Output Format

Print "Yes" if the brackets are balanced and "No" if not balanced.

Sample Input

```
(())
```

Sample Output

```
Yes
```

Explanation

```
(()) is a balanced string.
```

```
import java.util.*;
public class Main {

      public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub
            Scanner s = new Scanner(System.in);
            String str = s.next();
            Main mainobj = new Main();
            StacksUsingArrays stack = mainobj.new
StacksUsingArrays(1000);
            if (isBalanced(str, stack)) {
                  System.out.println("Yes");
            } else {
                  System.out.println("No");
            }

      }

      public static boolean isBalanced(String str, StacksUsingArrays
stack) throws Exception {
            for(int i=0; i<str.length(); i++){
                  if(str.charAt(i) == '(' || str.charAt(i) == '[' ||
str.charAt(i)=='{'){
                        stack.push(str.charAt(i));
                  }
                  else{
                        int s = stack.pop();
                        if(s != (int)(str.charAt(i))-1 && s !=
(int)(str.charAt(i))-2){
                              return false;
                        }
                  }
            }
            if(stack.size() > 0)
                  return false;
            return true;

      }

      private class StacksUsingArrays {
            private int[] data;
            private int tos;

            public static final int DEFAULT_CAPACITY = 10;

            public StacksUsingArrays() throws Exception {
                  // TODO Auto-generated constructor stub
                  this(DEFAULT_CAPACITY);
```

```
	}

	public StacksUsingArrays(int capacity) throws Exception {
		if (capacity <= 0) {
			System.out.println("Invalid Capacity");
		}
		this.data = new int[capacity];
		this.tos = -1;
	}

	public int size() {
		return this.tos + 1;
	}

	public boolean isEmpty() {
		if (this.size() == 0) {
			return true;
		} else {
			return false;
		}
	}

	public void push(int item) throws Exception {
		if (this.size() == this.data.length) {
			throw new Exception("Stack is Full");
		}
		this.tos++;
		this.data[this.tos] = item;
	}

	public int pop() throws Exception {
		if (this.size() == 0) {
			throw new Exception("Stack is Empty");
		}
		int retVal = this.data[this.tos];
		this.data[this.tos] = 0;
		this.tos--;
		return retVal;
	}

	public int top() throws Exception {
		if (this.size() == 0) {
			throw new Exception("Stack is Empty");
		}
		int retVal = this.data[this.tos];
		return retVal;
	}
```

```
        public void display() throws Exception {
            if (this.size() == 0) {
                throw new Exception("Stack is Empty");
            }
            for (int i = this.tos; i >= 0; i--) {
                System.out.println(this.data[i]);
            }

        }

    }

}
```

Importance of Time

There are N processes to be completed. All the processes have a unique number assigned to them from 1 to N.

Now, we are given two things:

1)The calling order in which all the processes are called.
2)The ideal order in which all the processes should have been executed.

Executing a process takes 1 unit of time. Changing the position takes 1 unit of time.

We have to find out the unit of time required to complete all the process such that a process is executed from the ideal order only when it exists at the same index in the calling order. We can push the first term from the calling order to the last thus rotating the order.

Input Format

First line contains a single integer N.
Next line contains N space separated integers denoting the calling order.
Last line contains N space separated integers denoting the ideal order.

Constraints

```
1 <= N <= 10^6
```

Output Format

The total time required

Sample Input

```
5
5 4 2 3 1
5 2 1 4 3
```

Sample Output

```
7
```

Explanation

Iteration #1: Since the ideal order and calling order both has process #5 to be executed first. Process #5 is executed taking 1 unit of time. The new calling order is: 4 - 2 - 3 - 1. Time taken in step #1: 1.

Iteration #2: Since the ideal order has process #2 to be executed firstly, the calling ordered has to be changed again, i.e., the first element has to be pushed to the last place. The new calling order is: 2 - 3 - 1 - 4 and process #2 is executed. Time taken in step #2: 2.

Iteration #3: Since the ideal order has process #1 to be executed firstly, the calling ordered has to be changed again, i.e., the first element has to be pushed to the last place. The new calling order is: 1 - 4 - 3 and process #1 is executed. Time taken in step #2: 2.

Iteration #4: Since the new first element of the calling order is same as the ideal order, that process will be executed. Time taken in step #4: 1.

Iteration #5: Since the last element of the calling order is same as the ideal order, that process will be executed. Time taken in step #5: 1.

Total time taken = 7

```
import java.util.*;

public class Main {

    protected int size;

    protected int front;
    protected int[] data;

    public Main() {
          this.size = 0;
          this.front = 0;
          this.data = new int[5];
     }

     public Main(int cap) {
          this.size = 0;
          this.front = 0;
          this.data = new int[cap];
     }

     public int size() {
          return size;
     }

     public boolean isEmpty() {
          return (size == 0);
     }

     public void enQueue(int item) throws Exception {
          if (this.size() == this.data.length) {
               int[] oa = this.data;
               int[] na = new int[oa.length * 2];
               for (int i = 0; i < this.size(); i++) {
                    int idx = (i + front) % oa.length;
                    na[i] = oa[idx];
               }

               this.data = na;
               this.front = 0;
          }

          // if (this.size == this.data.length) {
          // throw new Exception("Main is full");
          // }

          this.data[(front + size) % this.data.length] =
item;
          size++;

     }
     public int deQueue() throws Exception {
          if (this.size == 0) {
               throw new Exception("Main is empty");

          }

          int rv = this.data[front];
          front = (front + 1) % this.data.length;
          size--;

          return rv;

     }

     public int getFront() throws Exception {
          if (this.size == 0) {
               throw new Exception("Main is empty");
          }

          int rv = this.data[front];

          return rv;
     }

     public void display() {
          System.out.println();
          for (int i = 0; i < size; i++) {
               int idx = (i + front) % this.data.length;
               System.out.print(this.data[idx] + " ");
          }
        System.out.print("END");
     }


    public static int ImpofTime(Main q,int[] orig_order)
throws Exception{

     // Write your Code here
     int i = 0;
     int time = 0;
     while(q.size() > 0){
          if(q.getFront() == orig_order[i]){
                q.deQueue();
            }
            else{
                while(q.getFront() != orig_order[i]){
                    q.enQueue(q.deQueue());
                    time+=1;
                }
                q.deQueue();
            }
                time+=1;
                i++;
        }

        return time;


       }


        static Scanner scn = new Scanner(System.in);

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

            Main q = new Main();

            int n = scn.nextInt();
            int[] process = new int[n];
            for (int i = 0; i < n; i++) {
                q.enQueue(scn.nextInt());
            }

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

               process[i] = scn.nextInt();
            }



            System.out.print(ImpofTime(q,process));
        }
}
```

Next Greater Element

Given an array, print the Next Greater Element (NGE) for every element. The Next Greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.

Input Format

First line of the input contains a single integer T denoting the number of testcases. First line of each testcase contains an integer N denoting the size of array. Second line of each testcase contains N space seperated integers denoting the array.

Constraints

```
1 <= T <= 50 1 <= N <= 10^5
```

Output Format

For each index, print its array element and its next greater element seperated by a comma in a new line.

Sample Input

```
2
4
11 13 21 3
5
11 9 13 21 3
```

Sample Output

```
11,13
13,21
21,-1
3,-1
11,13
9,13
13,21
21,-1
3,-1
```

Explanation

```
For the first testcase , the next greater element for 11 is 13 , for
13 its 21 and 21 being the largest element of the array does
not have a next greater element. Hence we print -1 for 21. 3 is
the last element of the array and does not have any greater
element on its right. Hence we print -1 for it as well.
```

```java
import java.util.*;
```

```
public class Main {
    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        int t = scn.nextInt();

        while (t > 0) {
            int n = scn.nextInt();
            int[] arr = new int[n];

            for (int i = 0; i < arr.length; i++)
                arr[i] = scn.nextInt();

            nextLarger(arr);

            t--;
        }

    }

    // Function to print Next Greater Element for
each element of the array
    public static void nextLarger(int[] arr) {

// Write Code here
        int []ans = new int[arr.length];
        Stack<Integer>stacks = new Stack<>();
        for(int i=0; i<arr.length;i++) {
            while(!stacks.isEmpty() && arr[i] >
arr[stacks.peek()]) {
                ans[stacks.pop()] = arr[i];
            }
            stacks.push(i);
        }
```

```
          while(!stacks.isEmpty()) {
               ans[stacks.pop()] = -1;
          }

          for(int i=0; i<ans.length;i++) {
               System.out.println(arr[i]+","+ans[i]);
          }
     }
}
```

Find the greater element

We are given a circular array, print the next greater number for every element. If it is not found print -1 for that number. To find the next greater number for element A_i , start from index i + 1 and go uptil the last index after which we start looking for the greater number from the starting index of the array since array is circular.

Input Format

First line contains the length of the array n. Second line contains the n space separated integers.

Constraints

```
1 <= n <= 10^6
-10^8 <= A_i <= 10^8 , 0<= i< n
```

Output Format

Print n space separated integers each representing the next greater element.

Sample Input

```
3
1 2 3
```

Sample Output

```
2 3 -1
```

Explanation

```
Next greater element for 1 is 2,
for 2 is 3 but not present for 3 therefore -1
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int[n];
        for(int i=0; i<n;i++){
            arr[i] = sc.nextInt();
        }
        nextLarger(arr);
    }

    public static void nextLarger(int[] arr) {
        int []ans = new int[arr.length];
        Stack<Integer>stacks = new Stack<>();
        for(int i=0; i<arr.length;i++) {
            while(!stacks.isEmpty() && arr[i] >
arr[stacks.peek()]) {
                ans[stacks.pop()] = arr[i];
            }
            stacks.push(i);
        }

        for(int i=0; i<arr.length;i++){
            while(!stacks.isEmpty() && arr[i] >
arr[stacks.peek()]){
                ans[stacks.pop()] = arr[i];
            }
        }

        while(!stacks.isEmpty()){
            ans[stacks.pop()] = -1;
        }
        for(int i=0; i<ans.length;i++) {
            System.out.print(ans[i]+" ");
        }
    }
}
```

Recycling

You are a marketing head of a big MNC. You made a bar graph for your latest presentation. Now for your next project you need a rectangular sheet. But you don't want to waste anymore paper.

Since you don't need that bar graph anymore, you can use sheets from that. So you have to find the biggest possible rectangle you can cut out of that bar graph.

Input Format

First line takes integer N(no. of histogram bar).
Second line contains N integers representing the height of each bar respectively.

Constraints

```
Constraints: 1<=N<=10^5
0<=height[i]<=10^4
```

Output Format

Print the largest area of the rectangle that can be formed in histogram.

Sample Input

```
6
2 1 5 6 2 3
```

Sample Output

```
10
```

Explanation

```
The above is a histogram where the width of each bar is 1.
The largest rectangle is shown in the red area, which has an
area = 10 units.
```

```
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] br_height = new int[n];
        for(int i=0; i<n;i++){
            br_height[i] = sc.nextInt();
        }
        int max_height = 0;
        Stack<Integer>stack = new Stack<Integer>();
        for(int i=0; i<n;i++){
            while(!stack.isEmpty() && br_height[i] <=
br_height[stack.peek()]){
                int h = br_height[stack.pop()];
                int w = 0;
                if(stack.isEmpty()){
                    w = i;
                }
                else{
                    w = i - stack.peek()-1;
                }
                max_height =
Math.max(w*h,max_height);
            }
            stack.push(i);
        }

        while(!stack.isEmpty()){
            int h = br_height[stack.pop()];
            int w = 0;
            if(stack.isEmpty())
                w = n;
            else
                w = n - stack.peek()-1;
            max_height = Math.max(h*w, max_height);
        }
        System.out.print(max_height);
    }
}
```

Form minimum number from given Sequence

Given an array of patterns containing only I's and D's. I for increasing and D for decreasing. Devise an algorithm to print the minimum number following that pattern. Digits from 1-9 and digits can't repeat.

Input Format

The First Line contains an Integer N, size of the array. Next Line contains N Strings separated by space.

Constraints

```
1 ≤ T ≤ 100 1 ≤ Length of String ≤ 8
```

Output Format

Print the minimum number for each String separated by a new Line.

Sample Input

```
4
D I DD II
```

Sample Output

```
21
12
321
123
```

Explanation

For the Given sample case, For a Pattern of 'D' print a decreasing sequence which is 2 1.

```
import java.util.*;
```

```
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for(int k=0; k<n;k++){
            String s = sc.next();
            int [] answer = new int [s.length()+1];
             int count = 1;
             for(int i=0; i<=s.length(); i++) {
                 if(i == s.length() ||
s.charAt(i)=='I') {
                     answer[i] = count;
                     count++;
                     for(int j= i-1; j>=0 &&
s.charAt(j) != 'I'; j--) {
                         answer[j]= count;
                         count++;
                     }
                 }
             }
             String ans_new = "";
             for(int val : answer) {
                 ans_new+=val;
                 }
            System.out.println(ans_new);
            }
        }
}
```

Only Ladders

Take as input N, a number. N is the size of a snakes and ladders board. There are no snakes but we've ladders from 1st prime number to last prime number in range, 2nd prime number to 2nd from last prime number and so-on.

a. Write a recursive function which returns the count of different distinct ways this board can be crossed with a normal dice. Print the value returned.

b. Write a recursive function which prints all valid paths (void is the return type for function).

Input Format

Enter the size of the snake and ladders board N

Constraints

```
None
```

Output Format

Display the number of ways in which the board can be crossed and also display all the possible ways to cross the board in a comma separated manner

Sample Input

```
3
```

Sample Output

```
0 1 2 END , 0 1 END , 0 2 END , 0 END ,
4
```

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
           Scanner pavan = new Scanner(System.in);
           int p = pavan.nextInt();

           List<Integer>list = count_prime_sieve(p-1);
           ladder(list,p,"0 ",0,0);
           System.out.println();
           System.out.print(total);
    }
      static int total = 0;
      public static void ladder(List<Integer>list, int n, String
ways, int curr_count, int pr_count){
           if(curr_count > n){
                return;
```

```
            }
            if(curr_count == n-1){
                  total++;
                  System.out.print(ways+"END , ");
                  return;
            }
            if(curr_count == n){
                  total++;
                  System.out.print(ways.substring(0,ways.length()-
2)+"END , ");
                  return;
            }
            if(pr_count < list.size()/2 && list.get(pr_count) ==
curr_count ){
                  pr_count++;
                  ladder(list,n,ways+list.get(list.size()-pr_count)+"
",list.get(list.size()-pr_count),pr_count);
                  return;
            }
            for(int i=1; i<=6;i++){
                  ladder(list,n,ways+(curr_count+i)+"
",curr_count+i,pr_count);
            }
      }

      public static List<Integer> count_prime_sieve(int n) {
            boolean []pr = new boolean[n+1];
            pr[0] = pr[1] = true;
            for(int i=2; i*i<pr.length;i++) {
                  if(pr[i] == false) {
                        for(int j=2; j*i<=n; j++) {
                              pr[j*i] = true;
                        }
                  }
            }
            List<Integer>list = new ArrayList<>();
            for(int i=2; i<pr.length;i++){
                  if(pr[i] == false)
                        list.add(i);
            }
            return list;
      }
}
```

REVERSE A STACK USING RECURSION

Reverse a Stack using Recursion. Do not use any extra stack.

Input Format

First line contains an integer N (size of the stack).
Next N lines follow each containing an integer to be stored in the stack where the last integer is at the top of the stack.

Constraints

```
1 <= N <= 10^4
```

Output Format

Output the values of the reversed stack with each value in one line each.

Sample Input

```
3
3
2
1
```

Sample Output

```
3
2
1
```

Explanation

```
Original Stack = [ 3 , 2 , 1 ] <-TOP
Reverse Stack = [ 1 , 2 , 3 ] <-TOP
```

```
import java.util.*;
public class Main {

	public static void main(String args[]) throws Exception {
		// Your Code Here
		Scanner s = new Scanner(System.in);
		int N=s.nextInt();
		Main mainobj = new Main();
		StacksUsingArrays obj = mainobj.new StacksUsingArrays(N);
		StacksUsingArrays helper=mainobj.new 
StacksUsingArrays(N);
		for(int i=1;i<=N;i++){
			obj.push(s.nextInt());
		}
		obj.reverseStack(obj, helper, 0);
		helper.display();

	}

	private class StacksUsingArrays {
		private int[] data;
		private int tos;

		public static final int DEFAULT_CAPACITY = 10;

		public StacksUsingArrays() throws Exception {
			// TODO Auto-generated constructor stub
			this(DEFAULT_CAPACITY);
		}

		public StacksUsingArrays(int capacity) throws Exception {
			if (capacity <= 0) {
				System.out.println("Invalid Capacity");
			}
			this.data = new int[capacity];
			this.tos = -1;
		}

		public int size() {
			return this.tos + 1;
		}

		public boolean isEmpty() {
			if (this.size() == 0) {
				return true;
			} else {
				return false;
			}
		}

		public void reverseStack(StacksUsingArrays stack, 
StacksUsingArrays helper, int idx) throws Exception {
			if(stack.isEmpty()){
				return;
			}
```

```
                int n = stack.pop();
                helper.push(n);
                reverseStack(stack, helper, idx+1);
            }

            public void push(int item) throws Exception {
                if (this.size() == this.data.length) {
                    throw new Exception("Stack is Full");
                }
                this.tos++;
                this.data[this.tos] = item;
            }

            public int pop() throws Exception {
                if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
                }
                int retVal = this.data[this.tos];
                this.data[this.tos] = 0;
                this.tos--;
                return retVal;
            }

            public int top() throws Exception {
                if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
                }
                int retVal = this.data[this.tos];
                return retVal;
            }

            public void display() throws Exception {
                if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
                }
                for (int i = this.tos; i >= 0; i--) {
                    System.out.println(this.data[i]);
                }

            }

        }

}
```

The Queue Game

The Game is as follows You have given a binary array, where 1 denotes push operation and 0 denotes a pop operation in a queue. The task is to check if the possible set of operations are valid or not.
Print Valid if the set of Operations are Valid Otherwise Print Invalid.

Input Format

The First Line contains an Integer T, as the number of Test cases.
The Next Line contains an Integer N, as the Size of the Array.
The Next Line contains N Binary numbers separated by space.

Constraints

Output Format

Print *Valid* If the set of operations are valid Otherwise Print *Invalid* for each Test Case separated by a new Line.

Sample Input

```
2
5
1 1 0 0 1
5
1 1 0 0 0
```

Sample Output

```
Valid
Invalid
```

```
import java.util.*;
```

```java
public class Main {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        while(t>0){
            int n = sc.nextInt();
            int [] arr = new int[n];
            for(int i=0; i<n;i++){
                arr[i] = sc.nextInt();
            }

            int zeros = 0;
            int ones = 0;
            boolean flag = true;
            for(int i=0; i<arr.length;i++){
                if(arr[i] == 0)
                    zeros++;
                else
                    ones++;
                if(zeros > ones){
                    System.out.println("Invalid");
                    flag = false;
                    break;
                }
            }
            if(flag)
                System.out.println("Valid");
            t--;
        }
    }
}
```

Hoodies At Coding Blocks

It's winter season. There is a long queue of students from the **four** prime courses at Coding Blocks and everyone is here to grab his hoodie. Each student of a course has a different roll number. Whenever a new student will come, he will search for his friend from the end of the queue. Note that a student can only has friend from his course and not from any other course. As soon as he will find any of the friend in the queue, he will stand behind him, otherwise he will stand at the end of the queue. At any moment Kartik Bhaiya will ask the student, who is standing in front of the queue, to come and put his name and grab his hoodie and then remove that student from the queue. There are **Q** operations of one of the following types:

1. **E x y** : A new student of course x whose roll number is y will stand in queue according to the method mentioned above.
2. **D** : Kartik Bhaiya will ask the student, who is standing in front of the queue, to come and put his name for the hoodie and remove him from the queue.

Find out the order in which student put their name.

Note: Number of dequeue operations will never be greater than enqueue operations at any point of time.

Input Format

First line contains an integer **Q**, denoting the number of operations. Next **Q** lines will contains one of the **2** types of operations.

Constraints

```
1 ≤ x ≤ 4 1 ≤ y ≤ 50000 1 ≤ Q ≤ 100000
```

Output Format

For each **2nd** type of operation, print **two** space separated integers, the front student's course and roll number.

Sample Input

```
5
E 1 1
E 2 1
E 1 2
D
D
```

Sample Output

```
1 1
1 2
```

```
import java.util.*;
```

```
class Main{

     protected int size;

     protected int front;
     protected int[] data;

     public Main() {
          this.size = 0;
          this.front = 0;
          this.data = new int[5];
     }

     public Main(int cap) {
          this.size = 0;
          this.front = 0;
          this.data = new int[cap];
     }

     public int size() {
          return size;
     }

     public boolean isEmpty() {
          return (size == 0);
     }

     public void enqueue(int item) throws Exception {
          if (this.size() == this.data.length) {
               int[] oa = this.data;
               int[] na = new int[oa.length * 2];
               for (int i = 0; i < this.size(); i++) {
                    int idx = (i + front) % oa.length;
                    na[i] = oa[idx];
               }

               this.data = na;
               this.front = 0;
          }

          // if (this.size == this.data.length) {
          // throw new Exception("queue is full");
          // }

          this.data[(front + size) % this.data.length] = item;
          size++;
```

```
    }

    public int dequeue() throws Exception {
          if (this.size == 0) {
                throw new Exception("queue is empty");

          }

          int rv = this.data[front];
          front = (front + 1) % this.data.length;
          size--;

          return rv;

    }

    public int getFront() throws Exception {
          if (this.size == 0) {
                throw new Exception("queue is empty");
          }

          int rv = this.data[front];

          return rv;
    }

    public int get_index(int index){
          return this.data[index];
    }

    public void display() {
          System.out.println();
          for (int i = 0; i < size; i++) {
                int idx = (i + front) % this.data.length;
                System.out.print(this.data[idx] + " ");
          }
       System.out.print("END");
    }


  public static void hoodies(int Q) throws Exception{

    // Write your Code here
    ArrayList<LinkedList<Integer>> cose = new ArrayList<>();
          for (int i = 0; i <Q; i++) {
                cose.add(new LinkedList<Integer>());
          }
          ArrayList<Integer> check = new ArrayList<Integer>();
```

```
            while(Q-- >0) {
                  char ch = pavan.next().charAt(0);
                  if(ch=='E') {
                   int x = pavan.nextInt();
                   int y = pavan.nextInt();
                   if(check.contains(x)) {
                         cose.get(x).add(y);
                   }
                   else {
                         check.add(x);
                         cose.get(x).add(y);
                   }

                  }
                  else {

                        int cose_id = check.get(0);
                        int roll_no= cose.get(cose_id).removeFirst();
                        System.out.println(cose_id+" "+roll_no);
                        if(cose.get(cose_id).isEmpty()) {
                             check.remove(0);
                        }


                  }
            }
      }


      static Scanner pavan = new Scanner(System.in);

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

         int n = pavan.nextInt();

         hoodies(n);
      }

}
```

First negative integer in every window of size k

You are given given an array and a positive integer k, find the first negative integer for each and every window(contiguous subarray) of size k. If a window does not contain a negative integer, then print 0 for that window.

Input Format

First line contains integer t as number of test cases. Each test case contains two lines. First line contains two integers n and k where n is length of the array and k is the size of window and second line contains n space separated integer.

Constraints

```
1 < t < 10 1< n, k < 10000000
```

Output Format

For each test case you have to print the required output as given below.

Sample Input

```
2
8 3
12 -1 -7 8 -15 30 16 28
 8 4
12 -1 -7 8 -15 30 16 28
```

Sample Output

```
-1 -1 -7 -15 -15 0
-1 -1 -7 -15 -15
```

Explanation

For first test case : Subarray of window size 3 is (12 -1 -7), (-1, -7, 8), and so on.. Take first negative number from each window and print them.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();

        while(t>0){
            int n = sc.nextInt();
            int k = sc.nextInt();
            int [] a = new int[n];
            int [] ans = new int[n-k+1];
            for(int i=0; i<n; i++){
                a[i] = sc.nextInt();
            }

            int i=0;
            int count = 0;
            while(i<n && count < ans.length){
                if(a[i] < 0){
                    ans[count] = a[i];
                    count++;
                }
                else if(count+k-1 <= i){
                    count++;
                }
                if(i <= count-1 || a[i] >= 0){
                    i++;
                }
            }

            for(i=0; i<ans.length;i++){
                System.out.print(ans[i]+" ");
            }
            System.out.println();

            t--;
        }
    }
}
```

Mapped String

We are given a hashmap which maps all the letters with number. Given 1 is mapped with A, 2 is mapped with B.....26 is mapped with Z. Given a number, you have to print all the possible strings.

Input Format

A single line contains a number.

Constraints

```
Number is less than 10^6
```

Output Format

Print all the possible strings in sorted order in different lines.

Sample Input

```
123
```

Sample Output

```
ABC
AW
LC
```

Explanation

```
'1' '2' '3' = ABC
'1' '23' = AW
'12' '3' = LC
```

```
import java.util.*;
public class Main {
  static String arr[] = { "", "A", "B", "C", "D", "E", "F",
"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
            "R", "S", "T", "U", "V", "W", "X", "Y", "Z" };

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        String s = ""+n;
        Mappedstring(s, "");
    }

    public static void Mappedstring(String s, String ans) {
        if (s.length() == 0) {
            System.out.println(ans);
            return;
        }
        char s1 = s.charAt(0);
        Mappedstring(s.substring(1), ans + arr[s1-'0']);
        if (s.length() >= 2) {
            String s2 = s.substring(0, 2);
            int num = Integer.parseInt(s2);
            if (num <= 27) {
                Mappedstring(s.substring(2), ans +
arr[num]);
            }

        }
    }
}
```

Min Stack

Design a stack that supports push,pop,top,retrieving the minimum element in constant time.

Input Format

First line of input contains integer n denoting the size of the string array. Next line of input contains n space separated string where i-th string represent i-th operation.(i.e, push,pop,top,getMin)

Constraints

```
Functions pop, top and getMin operations will always be
called on non-empty stacks.
```

Output Format

Print the answer according to input operations.

Sample Input

```
7
push push push getMin pop top getMin
-2 0 -3
```

Sample Output

```
-3 0 -2
```

Explanation

```
push -2
push 0
push -3
print -3 (getMin)
pop -3
print 0 (top)
print -2 (getMin)
```

```
import java.util.*;
public class Main {
    public static void main (String args[]){
        Scanner sc = new Scanner(System.in);
        Stack<Integer> s = new Stack<>();

        int n = sc.nextInt();
        String [] oper = new String[n];
        for(int i=0; i<n;i++){
            oper[i] = sc.next();
            // System.out.println(oper[i]);
        }

        Stack<Integer>min = new Stack<>();
        for(int i=0; i<n;i++){
            if(oper[i].equals("push")){
                s.push(sc.nextInt());
                if(!min.isEmpty() && min.peek() <
s.peek()){
                    min.push(min.peek());
                }
                else{
                    min.push(s.peek());
                }
            }
            else if(oper[i].equals("pop")){
                s.pop();
                // System.out.print(s.pop()+" ");
                min.pop();
            }
            else if(oper[i].equals("top")){
                System.out.print(s.peek()+" ");
            }
            else{
                System.out.print(min.peek()+" ");
            }
        }

    }
}
```

New Permutation

Kartik and Parth and very obsessed with strings. then they decided to play a game. where Parth given kartik a string and ask him to print all the possible strings in lexographical order(smaller first and then largest). But while calculating the permuation the following condition must be satisfied. That he can change every letter individually to be lowercase or uppercase to create another string. without changing the order

Input Format

string input

Constraints

```
1 <= s.length <= 12
```

Output Format

all possibel permuation of strings under given condition

Sample Input

```
3z4
```

Sample Output

```
3z4 3Z4
```

```
import java.util.*;
```

```
public class Main {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        String s  = sc.next();
        List<String>list = new ArrayList<String>();
        case_perm(list,s,"",0);
        Collections.sort(list);
        for(int i=list.size()-1; i>=0;i--){
            System.out.print(list.get(i)+" ");
        }
    }

     public static List<String> case_perm(List<String>list,
String s, String ans, int curr_index){
        if(ans.length() == s.length()) {
               list.add(ans);
               return list;
          }

               if(s.charAt(curr_index) < 91 &&
s.charAt(curr_index)>64) {
                    case_perm(list,
s,ans+s.charAt(curr_index),curr_index+1);
                 case_perm(list,
s,ans+(char)(s.charAt(curr_index)+32),curr_index+1);
            }
               else if(s.charAt(curr_index)>96 &&
s.charAt(curr_index)<123) {
                    case_perm(list, s,
ans+s.charAt(curr_index),curr_index+1);
                    case_perm(list,
s,ans+(char)(s.charAt(curr_index)-32),curr_index+1);
               }
               else {
                    case_perm(list, s,
ans+s.charAt(curr_index),curr_index+1);
               }
        return list;
     }
}
```

Girlfriends Dearrangementt

Monu bhaiya wants to count all his girlfriend's Derangement. A Derangement is a permutation of n elements, so no element appears in its original position. For example, a derangement of {0, 1, 2, 3} is {2, 3, 1, 0}. Given a number n, find the total number of Derangements of a set of n elements.

Input Format

An integer N

Constraints

```
N=[1,10^6]
```

Output Format

Number of possible derangements

Sample Input

```
2
```

Sample Output

```
1
```

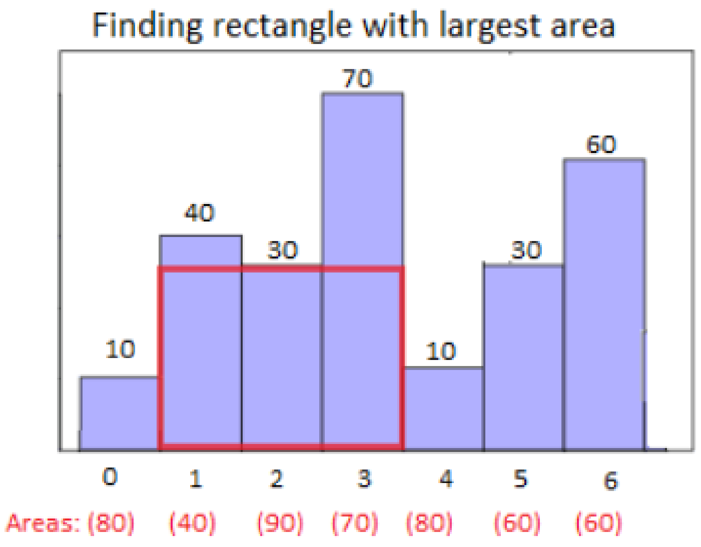
Explanation

For two elements say {0, 1}, there is only one possible derangement {1, 0}

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int[n];
        List<Integer>list = new ArrayList<Integer>();
        count_arrangement(0, arr, list);
        System.out.println(count);
    }
    static int count = 0;
    public static void count_arrangement(int index, int
[]arr, List<Integer>list){
        if(list.size()==arr.length){
//              System.out.println(list);
            count++;
            return;
        }
        for(int i=0; i<arr.length;i++){
            if(list.size() != i && arr[i] == 0){
                list.add(i);
                arr[i]--;
                count_arrangement(index+1, arr, list);
                list.remove(index);
                arr[i]++;
            }
        }
    }
}
```

Histogram

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars.



Input Format

First line contains a single integer N, denoting the number of bars in th histogram.
Next line contains N integers, i^{th} of which, denotes the height of i^{th} bar in the histogram.

Constraints

```
1<=N<=10^6
Height of each bar in histogram <= 10^10
```

Output Format

Output a single integer denoting the area of the required rectangle.

Sample Input

```
5
1 2 3 4 5
```

Sample Output

```
9
```

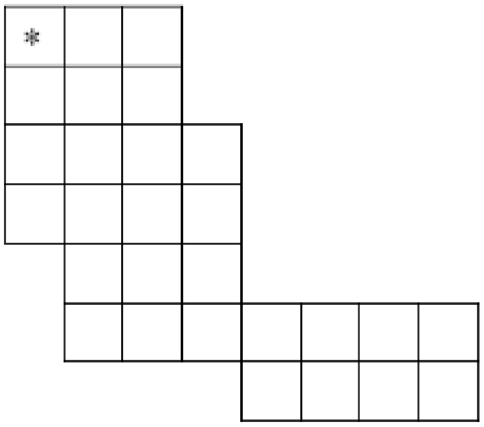
```
import java.util.*;
public class Main {

     public static void main(String[] args) throws Exception
{
          // TODO Auto-generated method stub
          Scanner s = new Scanner(System.in);

          int n = s.nextInt();
          int[] arr = new int[n];

          for(int i = 0;i < n;i++)
              arr[i] = s.nextInt();

          Main mainobj = new Main();
          StacksUsingArrays stack = mainobj.new
StacksUsingArrays(1000);
          System.out.println(hist(arr, stack));
          }


     public static long hist(int[] he, StacksUsingArrays st)
throws Exception {

         //Write Your Code here

          long max_area = 0;
         for(int i=0; i<he.length;i++){
             while(!st.isEmpty() && he[i] < he[st.top()]){
                 int ci = st.pop();
                 if(st.isEmpty()){
                     max_area =
Math.max((long)i*(long)he[ci],max_area);
                 }
                 else{
                     int l = st.top();
                     max_area = Math.max((long)(i-l-
1)*(long)he[ci],max_area);
                 }
             }
             st.push(i);
         }
```

```
        while(!st.isEmpty()){
            int ci = st.pop();
                if(st.isEmpty()){
                    max_area =
Math.max((long)he.length*(long)he[ci],max_area);
                }
                else{
                    int l = st.top();
                    max_area = Math.max((long)(he.length-l-
1)*(long)he[ci],max_area);
                }
        }

        return max_area;

     }

     private class StacksUsingArrays {
          private int[] data;
          private int tos;

          public static final int DEFAULT_CAPACITY = 10;

          public StacksUsingArrays() throws Exception {
               // TODO Auto-generated constructor stub
               this(DEFAULT_CAPACITY);
          }

          public StacksUsingArrays(int capacity) throws
Exception {
               if (capacity <= 0) {
                    System.out.println("Invalid Capacity");
               }
               this.data = new int[capacity];
               this.tos = -1;
          }

          public int size() {
               return this.tos + 1;
          }
          public boolean isEmpty() {
               if (this.size() == 0) {
                    return true;
               } else {
                    return false;
               }
          }

          public void push(int item) throws Exception {
               if (this.size() == this.data.length) {
                    throw new Exception("Stack is Full");
               }
               this.tos++;
               this.data[this.tos] = item;
          }

          public int pop() throws Exception {
               if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
               }
               int retVal = this.data[this.tos];
               this.data[this.tos] = 0;
               this.tos--;
               return retVal;
          }
          public int top() throws Exception {
               if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
               }
               int retVal = this.data[this.tos];
               return retVal;
          }
          public void display() throws Exception {
               if (this.size() == 0) {
                    throw new Exception("Stack is Empty");
               }
               for (int i = this.tos; i >= 0; i--) {
                    System.out.println(this.data[i]);
               }
          }
     }
}
```

Funky Chessboard

A knight is a piece used in the game of chess. The chessboard itself is square array of cells. Each time a knight moves, its resulting position is two rows and one column, or two columns and one row away from its starting position. Thus a knight starting on row r, column c – which we'll denote as (r,c) – can move to any of the squares (r-2,c-1), (r-2,c+1), (r-1,c-2), (r-1,c+2), (r+1,c-2), (r+1,c+2), (r+2,c-1), or (r+2,c+1). Of course, the knight may not move to any square that is not on the board.

Suppose the chessboard is not square, but instead has rows with variable numbers of columns, and with each row offset zero or more columns to the right of the row above it. The figure to the left illustrates one possible configuration. How many of the squares in such a modified chessboard can a knight, starting in the upper left square (marked with an asterisk), not reach in any number of moves without resting in any square more than once? Minimize this number.



If necessary, the knight is permitted to pass over regions that are outside the borders of the modified chessboard, but as usual, it can only move to squares that are within the borders of the board.

Input Format

First line contains an integer *n*, representing the side of square of chess board. The next *n* line contains *n* integers separated by single spaces in which j_{th} integer is 1 if that cell(i,j) is part of chessboard and 0 otherwise.

Constraints

The maximum dimensions of the board will be 10 rows and 10 columns. That is, any modified chessboard specified by the input will fit completely on a 10 row, 10 column board.

Output Format

Print the minimum number of squares that the knight can not reach.

Sample Input

```
3
1 1 1
1 1 1
1 1 1
```

Sample Output

```
1
```

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
          Scanner sc = new Scanner(System.in);
          int n = sc.nextInt();
          int [][]mat = new int [n][n];
          int c = 0;
          for(int i=0; i<n;i++){
               for(int j=0; j<n;j++){
                    mat[i][j] = sc.nextInt();
                    if(mat[i][j] == 1){
                         c++;
                    }
               }
          }

          boolean[][] visited = new boolean[n][n];
          funky(mat, 0,0,0,visited);
          System.out.println(c - total);
    }
     static int total = 0;
     public static void funky(int[][]mat, int cr, int cc,
int count, boolean [][] visited){
          if(cr >= mat.length || cc >= mat[0].length || cr <
0 || cc < 0 || visited[cr][cc] || mat[cr][cc]==0){
               total = Math.max(total,count);
               return;
          }

          visited[cr][cc] = true;
          funky(mat, cr+1, cc+2, count+1, visited);
          funky(mat, cr+1, cc-2, count+1, visited);
          funky(mat, cr-1, cc+2, count+1, visited);
          funky(mat, cr-1, cc-2, count+1, visited);
          funky(mat, cr+2, cc+1, count+1, visited);
          funky(mat, cr+2, cc-1, count+1, visited);
          funky(mat, cr-2, cc+1, count+1, visited);
          funky(mat, cr-2, cc-1, count+1, visited);
          visited[cr][cc] = false;
     }

}
```

Chessboard Problem – 2

Take as input N, a number. N represents the size of a chess board. The cells in board are numbered. The top-left cell is numbered 1 and numbering increases from left to right and top to bottom. E.g. The following is the chessboard for a value of n=4.

```
1    2    3    4

5    6    7    8

9    10   11   12

13   14   15   16
```

Prime numbers act as mines and ports alternately i.e. first prime number is a mine while second is a port and so on. Piece can go over a mine but cannot stop on it. Piece can directly move from a port to the destination (but may not chose to).

We've a piece standing in top-left corner and it must reach the bottom-right corner. The piece moves as follows –

a. In any cell, the piece moves like a knight. But out of the possible 8 moves for a knight, only the positive ones are valid i.e. both row and column must increase in a move.

b. On the walls (4 possible walls), the piece can move like a rook as well (in addition of knight moves). But, only the positive moves are allowed i.e. as a rook, piece can move any number of steps horizontally or vertically but in a manner, such that row or column must increase.

c. On the diagonals (2 possible diagonals), the piece can move like a bishop as well (in addition to the knight and possibly rook moves). But, only the positive moves are allowed i.e. as a bishop, piece can move in a way such that row and column must increase.

You are supposed to write the following functions

a. Write a recursive function which returns the count of different distinct ways this board can be crossed. Print the value returned.

b. Write a recursive function which prints all valid paths (void is the return type for function).

Input Format

Enter the size of the chessboard N

Constraints

```
None
```

Output Format

Display the total number of ways to cross the board and display all the possible paths in a space separated manner

Sample Input

```
3
```

```
{0-0}K{2-1}R{2-2} {0-0}K{1-2}R{2-2} {0-0}R{0-2}P{2-2} {0-0}R{0-2}R{1-2}R{2-2} {0-0}R{0-
2}R{2-2} {0-0}R{1-0}K{2-2} {0-0}R{1-0}R{1-2}R{2-2} {0-0}R{1-0}R{2-0}P{2-2} {0-0}R{1-0}R{2-
0}R{2-1}R{2-2} {0-0}R{1-0}R{2-0}R{2-2} {0-0}R{2-0}P{2-2} {0-0}R{2-0}R{2-1}R{2-2} {0-0}R{2-
0}R{2-2} {0-0}B{2-2}
14
```

```
import java.util.*;
public class Main {
    static int pc = 0;
    public static void main(String args[]) {

        Scanner pavan = new Scanner(System.in);
        int n = pavan.nextInt();
        boolean[] pr = new boolean[n*n + 1];
        pr_sv(pr);

        int[] to_check = new int[n*n+1];
        int p =1;
        for(int i=0;i<pr.length;i++){
            if(!pr[i]){
                to_check[i] = p;
                p++;
            }
        }

        System.out.println("\n" + chess(0,0,n,"{0-
0}",pr,to_check));

    }

    private static int chess(int cr,int c,int n,String
ans,boolean[] pr,int[] to_check){

        if(cr>=n || c>=n)
              return 0;

        if(cr==n-1 && c==n-1){
            System.out.print(ans + " ");
            return 1;
        }

        int count = 0;
        if(!pr[cr*n+c+1]){
            if(to_check[cr*n+c+1]%2==1){
                return 0;
            }
              else{
                count += chess(n-1,n-1,n,ans +
String.format("P{%d-%d}",n-1,n-1),pr,to_check);
            }
        }

        count += chess(cr+2,c+1,n,ans +
String.format("K{%d-%d}",cr+2,c+1),pr,to_check);
        count += chess(cr+1,c+2,n,ans +
String.format("K{%d-%d}",cr+1,c+2),pr,to_check);

        if(cr==0 || cr==n-1 || c==0 || c==n-1){

            for(int k=1;k+c<n;k++){
                count += chess(cr,c+k,n,ans +
String.format("R{%d-%d}",cr,c+k),pr,to_check);
            }

            for(int k=1;k+cr<n;k++){
                count += chess(cr+k,c,n,ans +
String.format("R{%d-%d}",cr+k,c),pr,to_check);
            }
        }

        if(cr==c || cr+c==n-1){

            for(int k=1;k+cr<n;k++){
                count += chess(cr+k,c+k,n,ans +
String.format("B{%d-%d}",cr+k,c+k),pr,to_check);
            }
        }

        return count;

    }

    private static void pr_sv(boolean[] pr){
        pr[0] = true;
        pr[1] = true;

        for(int i=2;i*i<pr.length;i++){

            if(!pr[i]){

                for(int j=2;i*j<pr.length;j++){
                    pr[i*j] = true;
                }
            }
        }
    }
}
```

Smart Keypad -1

You will be given a numeric string **S**. Print all the possible codes for **S**.

Following vector contains the codes corresponding to the digits mapped.

```
string table[] = { " ", ".+@$", "abc", "def", "ghi", "jkl" , "mno", "pqrs" , "tuv", "wxyz" };
```

For example, string corresponding to **0** is " " and **1** is ".+@$"

Input Format

A single string containing numbers only.

Constraints

```
length of string <= 10
```

Output Format

All possible codes one per line in the following order.

The letter that appears first in the code should come first

Sample Input

```
12
```

Sample Output

```
.a
.b
.c
+a
+b
+c
@a
@b
@c
$a
$b
$c
```

Explanation

For **code 1** the corresponding string is **.+@$** and **abc** corresponds to **code 2.**

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
         String s = sc.next();
        String keys[] = { " ", ".+@$", "abc", "def",
"ghi", "jkl" , "mno", "pqrs" , "tuv", "wxyz" };
         smart(s,keys,"");
    }

    public static void smart(String s, String []
keys, String ans){
         if(s.length()==0){
             System.out.println(ans);
              return;
         }
         char c = s.charAt(0);
         String se = keys[c - '0'];
         for(int i=0; i<se.length();i++){
              smart(s.substring(1), keys,
ans+se.charAt(i));
         }
    }


}
```

Search Word in Monu Bhaiya’s Board

Monu Bhaiya has a board of size M x N consisting of characters and a list of strings words, print all words which are on the board in a **sorted order** .

Each word must be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Input Format

First Line contains 3 space-separated Integers M ,N and K(length of words list) .
Next M Lines contains strings of length N .
Next Line contains a list of words .

Constraints

```
1 <= N,M <= 12
1 <= word.length <= 10^4
1 <= word[i].length <= 10
```

Output Format

print space-separated words which are on the board.

Sample Input

```
4 4 5
oaan
etae
ihkr
iflv
oath pea eat rain fifa
```

Sample Output

```
eat oath
```

Explanation

```
There are only two words "oath" and "eat" from the list which
are also in the character board .
```

```
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        int n = sc.nextInt();
        int words = sc.nextInt();
        char [][]board = new char [m][n];
        for(int i=0; i<m;i++){
            String s = sc.next();
            for(int j=0; j<n;j++){
                board[i][j] = s.charAt(j);
            }
        }

        List<String>list = new ArrayList<String>();
        while(words > 0){
            String word = sc.next();
            for(int i=0; i<m; i++){
                boolean ans = false;
                for(int j=0; j<n;j++){
                    ans = is_in(board,word,0,i,j);
                    if(ans){
                        list.add(word);
                        break;
                    }
                }
                if(ans)
                    break;
            }
            words--;
        }

        Collections.sort(list);
        for(int i=0; i<list.size();i++){
            System.out.print(list.get(i)+" ");
        }
    }

    public static boolean is_in(char[][] board, String
word,int index, int cr, int cc){
        //System.out.println(word+" "+index);
        if(index == word.length()){
            return true;
        }
        if(cr >= board.length || cc >= board[0].length ||
cr < 0 || cc < 0 || word.charAt(index) != board[cr][cc]){
            return false;
        }
        board[cr][cc] = '#';
        int [] r = {0, 0, 1, -1};
        int [] c = {-1, 1, 0, 0};
        for(int i=0; i<r.length; i++){
            boolean ans = is_in(board,
word,index+1,cr+r[i], cc+c[i]);
            if(ans){
                board[cr][cc] = word.charAt(index);
                return true;
            }
        }
        board[cr][cc] = word.charAt(index);
        return false;
    }
}
```