There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy. Children with a higher rating get more candies than their neighbors. Return the minimum number of candies you need to have to distribute the candies to the children.

**Input Format**

size of an array elements of an array

**Constraints**

n == ratings.length 1 <= n <= 2 \* 104 0 <= ratings[i] <= 2 \* 104

**Output Format**

integer

**Sample Input 0**

3

1 0 2

**Sample Output 0**

5

**Sample Input 1**

3

1 2 2

**Sample Output 1**

4

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 n=sc.nextInt();

int child[]=new int[n];

int i;

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

{

child[i]=sc.nextInt();

}

int count=n;

for(i=0;i<n-1;i++)

{

if(child[i]>child[i+1] || child[i+1]>child[i])

{

count++;

}

}

System.out.println(count);

}

}  
  
  
package arrays;  
import java.util.\*;  
  
public class Demo{  
     
   public static void main(String []args) {  
       Scanner sc=new Scanner(System.in);  
       int n=sc.nextInt();  
       int ratings[]=new int[n];  
       for(int i=0;i<n;i++) {  
           ratings[i]=sc.nextInt();  
       }  
       int candy[]=new int[n];  
       Arrays.fill(candy,1);  
       for(int i=1;i<n;i++) {  
           if(ratings[i]>ratings[i-1]) {  
               candy[i]=candy[i-1]+1;  
           }  
       }  
       for(int i=n-2;i>=0;i--) {  
           if(ratings[i]>ratings[i+1]) {  
               candy[i]=Math.max(candy[i], candy[i+1]+1);  
           }  
       }  
       int sum=0;  
       for(int i:candy) {  
           sum+=i;  
       }  
       System.out.println(sum);  
         
         
   }  
}

Write a program that takes an integer M and M integer array elements as input. The program needs to find the minimum difference between two elements in the integer array. The program then needs to print all those pairs of elements that have the minimum difference. If more than one pair has the minimum difference, then the program should print the output in the ascending order, if an element exists in two or more pairs, then it should be printed two times or more.

**Input Format**

The first line contains an integer M denoting the size of the array. The second line contains M space-separated integers representing the elements of the array.

**Constraints**

1 ≤ M ≤ 10^5 Elements of the array are integers.

**Output Format**

The output should consist of space-separated integers representing the pairs of elements with the minimum difference, printed in ascending order. If multiple pairs have the minimum difference, print all such pairs. If an element exists in two or more pairs, it should be printed multiple times.

**Sample Input 0**

4

55 44 33 22

**Sample Output 0**

22 33 33 44 44 55

**Explanation 0**

Explanation: The minimum difference between two elements is 11. Hence the pairs are printed in the ascending order. Here 33 and 44 appear in two different pairs; hence both are printed twice.

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 n=sc.nextInt();

int arr[]=new int[n];

int i;

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

{

arr[i]=sc.nextInt();

}

Arrays.sort(arr);

int mindiff=Integer.MAX\_VALUE;

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

{

mindiff=Math.min(mindiff,arr[i]-arr[i-1]);

}

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

{

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

{

System.out.print(arr[i-1]+" "+arr[i]+" ");

}

}

}

}

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

**Input Format**

The input consists of an array prices, where prices[i] represents the price of the stock on the ith day. The length of the array prices (number of days) will be between 1 and 10^5. Each element of the array prices will be an integer between 0 and 10^4.

**Constraints**

1 <= prices.length <= 10^5 0 <= prices[i] <= 10^4

**Output Format**

Return an integer representing the maximum profit that can be achieved from this transaction. If no profit can be achieved, return 0.

**Sample Input 0**

6

7 1 5 3 6 4

**Sample Output 0**

5

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 n=sc.nextInt();

int stock[]=new int[n];

int i;

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

{

stock[i]=sc.nextInt();

}

int buy=0,sell=1,profit=0,max=0;

while(sell<n)

{

if(stock[sell]>stock[buy])

{

profit=stock[sell]-stock[buy];

if(profit>max)

{

max=profit;

}

}

else

{

buy=sell;

}

sell++;

}

System.out.print(max);

}

}

You're tasked with developing a program that efficiently identifies pairs of prime numbers whose sum is also a prime number within a given range. This tool will be invaluable for various mathematical and cryptographic applications, where prime numbers play a crucial role.

**Input Format**

The input consists of a single integer representing the range within which prime pairs are to be found. The integer N, where 2 <= N <= 10^6, denotes the upper limit of the range.

**Constraints**

The input integer N represents the upper limit of the range within which prime pairs are to be found. The algorithm should efficiently handle large inputs, ensuring reasonable execution time. The output should be printed in ascending order of the first number in the pair.

**Output Format**

For each pair of prime numbers (x, y) found within the range, where x < y, print them separated by a space on separate lines. If no prime pairs are found within the given range, print "No prime pairs found."

**Sample Input 0**

5

**Sample Output 0**

2 3

2 5

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) {

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

Scanner sc=new Scanner(System.in);

int num=sc.nextInt();

int i,j,k=0;

for (i=2;i<=num;i++)

{

for(j=i+1;j<=num;j++)

{

if(prime(i)&&prime(j)&&prime(i+j))

{

System.out.println(i+" "+j);

}

}

}

}

public static boolean prime(int num)

{

int i;

boolean flag=true;

for(i=2;i<num;i++)

{

if(num%i==0)

{

flag=false;

break;

}

}

return flag;

}

}

Find Second Smallest Element in an Array without any algorithm.

**Input Format**

Size of an array Elements of an array

**Constraints**

n >= 2

**Output Format**

Smallest Integer

**Sample Input 0**

5

13 18 9 1 21

**Sample Output 0**

9

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) {

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

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int arr[]=new int[n];

int i;

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

{

arr[i]=sc.nextInt();

}

int min=arr[0];

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

{

if(arr[i]<min)

{

min=arr[i];

}

}

int min2=Integer.MAX\_VALUE;

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

{

if(arr[i]!=min && arr[i]<min2)

{

min2=arr[i];

}

}

System.out.print(min2);

}

}

You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return true if you can reach the last index, or false otherwise.

**Input Format**

The first line contains an integer n, the length of the integer array nums. The second line contains n space-separated integers, representing the elements of the array nums.

**Constraints**

1 <= nums.length <= 104 0 <= nums[i] <= 105

**Output Format**

A single line containing either "true" if it's possible to reach or "false" otherwise.

**Sample Input 0**

5

2 3 1 1 4

**Sample Output 0**

true

**Sample Input 1**

5

3 2 1 0 4

**Sample Output 1**

false

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) {

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

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int arr[]=new int[n];

int i,j=0;

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

{

arr[i]=sc.nextInt();

}

System.out.println(jump(arr,n));

}

public static boolean jump(int arr[],int n)

{

int i,f=n-1;

for(i=n-2;i>=0;i--)

{

if(arr[i]+i>=f)

{

f=i;

}

}

return f==0;

}

}  
A party has been organised on cruise. The party is organised for a limited time(T). The number of guests entering (E[i]) and leaving (L[i]) the party at every hour is represented as elements of the array. The task is to find the maximum number of guests present on the cruise at any given instance within T hours. Example 1: Input : • 5 -> Value of T • [7,0,5,1,3] -> E[], Element of E[0] to E[N-1], where input each element is separated by new line  
• [1,2,1,3,4] -> L[], Element of L[0] to L[N-1], while input each element is separate by new line. Output : 8 -> Maximum number of guests on cruise at an instance.

**Input Format**

The first line of input will be a single integer, T, representing the number of hours the party lasts. The second set of inputs consists of T integers, each on a new line, representing the array E[i], the number of guests entering the party at each hour. The third set of inputs consists of T integers, each on a new line, representing the array L[i], the number of guests leaving the party at each hour.

**Constraints**

No Constraints

**Output Format**

The output should be a single integer representing the maximum number of guests present on the cruise at any instance during the T hours.

**Sample Input 0**

5

7 0 5 1 3

1 2 1 3 4

**Sample Output 0**

8

package arrays;

import java.util.Scanner;

public class guest {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        int n=sc.nextInt();

        int entry[]=new int[n];

        int leave[]=new int[n];

        int i,max=0,guest = 0;

        for(i=0;i<n;i++)

        {

            entry[i]=sc.nextInt();

        }

        for(i=0;i<n;i++)

        {

            leave[i]=sc.nextInt();

        }

        for(i=0;i<n;i++)

        {

            guest += entry[i]-leave[i];

             max=Math.max(max,guest);

        }

        System.out.println(max);

    }

}

Imagine you are participating in a national coding competition where you've been challenged to display your mastery of loops and arrays. Your task is to develop a program that generates a unique spiral pattern with numbers, starting from 1 and moving in a clockwise direction until the entire matrix is filled. This task not only tests your algorithmic skills but also your ability to manipulate multidimensional arrays efficiently. Your solution will be used to generate patterns for a new kind of puzzle game that is both entertaining and educational, aiming to enhance logical thinking and problem-solving skills among users of all ages.

**Input Format**

The input consists of a single integer, n, representing the size of the square matrix (n x n) to be generated.

**Constraints**

1≤n≤100, where n is the dimension of the square matrix.

**Output Format**

The output should be an n x n matrix filled with integers from 1 to n\*n, arranged in a clockwise spiral pattern. Each number should be formatted to occupy exactly two spaces, ensuring alignment for single and double-digit numbers, followed by a space character.

**Sample Input 0**

5

**Sample Output 0**

01 02 03 04 05

16 17 18 19 06

15 24 25 20 07

14 23 22 21 08

13 12 11 10 09

**Sample Input 1**

2

**Sample Output 1**

01 02

04 03

package arrays;  
import java.util.\*;  
public class Demo{  
   public static void main(String []args) {  
       Scanner sc=new Scanner(System.in);  
       int n=sc.nextInt();  
       int db[][]=new int[n][n];  
       int data=1;  
       int ltors=0,ltore=n-1,ltorp=0;  
       int ttobs=1,ttobe=n-1,ttobp=n-1;  
       int rtols=n-2,rtole=0,rtolp=n-1;  
       int btots=n-2,btote=1,btotp=0;  
       for(int j=0;j<=2;j++) {  
  
             
           for(int i=ltors;i<=ltore;i++) {  
               db[ltorp][i]=data;  
               data++;  
           }  
             
           for(int i=ttobs;i<=ttobe;i++) {  
               db[i][ttobp]=data;  
               data++;  
           }  
             
           for(int i=rtols;i>=rtole;i--) {  
               db[rtolp][i]=data;  
               data++;  
           }  
             
           for(int i=btots;i>=btote;i--) {  
               db[i][btotp]=data;  
               data++;  
           }  
  
           ltors++;ltore--;ltorp++;  
           ttobs++;ttobe--;ttobp--;  
           rtols--;rtole++;rtolp--;  
           btots--;btote++;btotp++;  
       }  
         
       for(int i=0;i<n;i++) {  
           for(int j=0;j<n;j++) {  
               System.out.printf("%02d ",db[i][j]);  
           }  
           System.out.println();  
       }  
         
   }  
}

You are playing the following Nim Game with your friend:

Initially, there is a heap of stones on the table. You and your friend will alternate taking turns, and you go first. On each turn, the person whose turn it is will remove 1 to 3 stones from the heap. The one who removes the last stone is the winner. Given n, the number of stones in the heap, return true if you can win the game assuming both you and your friend play optimally, otherwise return false.

**Input Format**

An integer n representing the number of stones in the heap.

**Constraints**

1 <= n <= 231 - 1

**Output Format**

A boolean value indicating whether you can win the game assuming both you and your friend play optimally.

**Sample Input 0**

4

**Sample Output 0**

false

**Sample Input 1**

1

**Sample Output 1**

true

**Sample Input 2**

2

**Sample Output 2**

true

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) {

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

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

if(n%4==0)

{

System.out.println("false");

}

else

{

System.out.println("true");

}

}

}

My logic:

package arrays;

import java.util.Scanner;

public class nim\_game {

    public static void main(String[] args) {

         Scanner sc=new Scanner(System.in);

        int n=sc.nextInt();

        int k=1;

        if(n<=3)

        {

            System.out.println("true");

        }

        else{

        for(int i=1;i<=n;i++)

        {

          if(k<3)

          {

              k++;

          }

        }

        System.out.println(k);

        if(k==n)

              {

                  System.out.println("true");

              }

        else

            {

                System.out.println("false");

            }

        }

    }

}

Given an array of integers, rearrange the elements such that all even numbers appear on the left side of the array in decreasing order, and all odd numbers appear on the right side of the array in decreasing order. The algorithm should not use any extra space.

**Input Format**

An integer array arr of length n (1 ≤ n ≤ 10^5) Each element of the array is an integer ranging from -10^9 to 10^9.

**Constraints**

The algorithm should not use any extra space. The elements of the array can be negative, zero, or positive integers. There can be duplicate elements in the array.

**Output Format**

The modified array arr after rearranging its elements according to the given conditions.

**Sample Input 0**

5

10 8 6 4 2

**Sample Output 0**

10 8 6 4 2

**Sample Input 1**

8

7 6 9 8 3 2 5 4

**Sample Output 1**

8 6 4 2 9 7 5 3

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

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

public static void reverse(int arr[],int left,int right)

{

while(left<right)

{

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int i, temp,j=n-1;

int arr[]=new int[n];

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

{

arr[i]=sc.nextInt();

}

int left = 0, right = n - 1;

while (left < right) {

if (arr[left] % 2 == 0) {

left++;

} else if (arr[right] % 2 != 0) {

right--;

} else {

temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

Arrays.sort(arr,0,left);

reverse(arr, 0, left-1 );

Arrays.sort(arr,left,n);

reverse(arr, left, n-1 );

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

{

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

}

}

}

Check Whether Or Not the Number is an Automorphic Number in Java Given an integer input, the Objective is to check whether the square of the number ends with the same number or not. Example Input : 5 Output : 25 Explanation : Number = 5 when squared you get 25 as 25 ends with 5 In this square should ends with the total given number that is 5.

25 = (25)2 = 625

Note:76=(76)2 = 5776.. In this 5776 is the square of the 76 number and in this 76 is totally present in the square of 5776.

**Input Format**

The program takes a single integer as input.

**Constraints**

The input number should be a non-negative integer. The input number should be within the range of Java's int data type (i.e., between Integer.MIN\_VALUE and Integer.MAX\_VALUE).

**Output Format**

The program outputs whether the entered number is an Automorphic Number or not i.e)YES OR NO

**Sample Input 0**

5

**Sample Output 0**

YES

**Sample Input 1**

9

**Sample Output 1**

NO

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) {

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

Scanner sc = new Scanner(System.in);

int n=sc.nextInt();

int n1=n\*n;

int num=n%10;

int sqrt=n1%10;

// System.out.print("num "+num+"sqrt "+sqrt);

if(num==sqrt)

{

System.out.print("YES");

}

else

{

System.out.print("NO");

}

}

}

Given two strings s and t, return true if s is a subsequence of t, or false otherwise.

A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

**Input Format**

Two strings s and t.

**Constraints**

0 <= s.length <= 100 0 <= t.length <= 104 s and t consist only of lowercase English letters.

**Output Format**

A boolean value indicating whether s is a subsequence of t or not.

**Sample Input 0**

"abc"

"ahbgdc"

**Sample Output 0**

true

**Sample Input 1**

"axc"

"ahbgdc"

**Sample Output 1**

false

package Strings;

import java.util.Scanner;

public class subsequence {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String S1=sc.next();

        String Substring=sc.next();

        int i=0,j=0;

        boolean flag=false;

        while(i<Substring.length()&&j<S1.length())

        {

            if(Substring.charAt(i)==(S1.charAt(j)))

            {

                flag=true;

                i++;

                j++;

            }

            else{

                flag=false;

                j++;

            }

        }

        if(flag)

        {

            System.out.print("true");

        }

        else{

            System.out.print("false");

        }

    }

}

Write a program that will take one string as input. The program will then remove vowels a, e, i, o, and u (in lower or upper case ) from the string. If there are two or more vowels that occur together then the program shall ignore all of those vowels.

**Input Format**

A single line of text input that can contain any characters from the ASCII character set. The string length can be from 1 to 10^5 characters.

**Constraints**

The input string will only contain ASCII characters (character codes 0-127). The length of the input string will be in the range [1, 10^5].

**Output Format**

A single line of text output, which is the modified version of the input string after applying the vowel removal rule.

**Sample Input 0**

Cat

**Sample Output 0**

Ct

**Sample Input 1**

Compuuter

**Sample Output 1**

Cmpuutr

package Strings;

import java.util.Scanner;

public class vowels {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str =sc.next();

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

        {

                if(str.charAt(i)=='a'||str.charAt(i)=='e'||str.charAt(i)=='i'||str.charAt(i)=='o'||str.charAt(i)=='u'||str.charAt(i)=='A'||str.charAt(i)=='E'||str.charAt(i)=='I'||str.charAt(i)=='O'||str.charAt(i)=='U')

                {

                    if(i < str.length() - 1 &&str.charAt(i)==str.charAt(i+1))

                    {

                        System.out.print(str.charAt(i)+ ""+str.charAt(i+1));

                        i++;

                    }

                    else{

                       continue;

                    }

                }

                else{

                    System.out.print(str.charAt(i)+ "");

                }

        }

    }

}

package Strings;

import java.util.Scanner;

public class vowels {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        String str =sc.next();

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

        {

            if("aeiouAEIOU".indexOf(str.charAt(i))!=-1)

            {

                if((i<str.length()-1 && (str.charAt(i)==str.charAt(i+1)))||(i>0&&(str.charAt(i)==str.charAt(i-1))))

                {

                    System.out.print(str.charAt(i)+ "");

                }

            }

            else{

                System.out.print(str.charAt(i)+ "");

            }

        }

    }

}

Given an integer array nums, find the subarray with the largest sum, and return its sum.

**Input Format**

The first line of input contains an integer n, denoting the size of the integer array nums. The second line contains n space-separated integers, representing the elements of the array nums.

**Constraints**

1 <= nums.length <= 105 -104 <= nums[i] <= 104

**Output Format**

The program outputs a single integer representing the sum of the maximum subarray.

**Sample Input 0**

4

1 2 -5 10

**Sample Output 0**

10

**Sample Input 1**

9

-2 1 -3 4 -1 2 1 -5 4

**Sample Output 1**

6

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) {

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

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();

}

int current\_sum=0,max\_sum=Integer.MIN\_VALUE;

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

{

current\_sum+=arr[i];

if(current\_sum>max\_sum)

{

max\_sum=current\_sum;

}

if(current\_sum<0)

{

current\_sum=0;

}

}

System.out.print(max\_sum);

}

}

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);

String ransom=sc.next();

String magazine=sc.next();

System.out.print(ransom\_note(ransom,magazine));

}

public static boolean ransom\_note(String s1,String s2)

{

int i=0,j=0;

boolean flag=false;

while(i<s1.length()&&j<s2.length())

{

if(s1.charAt(i)==s2.charAt(j))

{

flag=true;

i++;

j++;

}

else{

flag=false;

j++;

}

}

return flag;

}

}

Imagine you are developing a system for a lottery game where participants can only use tickets that contain specific digits. For example, the lottery allows tickets that only contain the digits '3' and '7'. Your task is to identify all possible ticket numbers within a given range that meet this criterion. Participants will input their chosen digits and the range of ticket numbers they want to evaluate. The program will then output all valid ticket numbers that can be formed using only those two digits.

**Input Format**

First Line: Two integers a and b (0 ≤ a, b ≤ 9), representing the two digits that can be used in the numbers. Second Line: An integer range (1 ≤ range ≤ 10^6), representing the upper limit of the range (exclusive) within which to search for valid numbers.

**Constraints**

The digits a and b must be single-digit integers (0 through 9). The range must be a positive integer greater than or equal to 1 and can be as large as 1,000,000.

**Output Format**

The output will be a space-separated list of integers that are composed solely of the digits a and b. If no such numbers exist in the specified range, the output will be an empty line.

**Sample Input 0**

5

6

100

**Sample Output 0**

5 6 55 56 65 66

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) {

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

Scanner sc=new Scanner(System.in);

int n1=sc.nextInt();

int n2=sc.nextInt();

int range=sc.nextInt();

for(int i=1;i<=range;i++)

{

if(check(i,n1,n2))

{

System.out.print(i+" ");

}

}

}

public static boolean check(int i,int n1,int n2)

{

while(i!=0)

{

int rem=i%10;

if(rem!=n1 && rem!=n2){

return false;

}

i=i/10;

}

return true;

}

}

Given an odd length word your task is to complete the function printPattern that takes a string s as its argument and prints it from the middle of the word such that it follows the following pattern.

**Input Format**

String

**Constraints**

1 ≤ T ≤ 20 1<=size of string(s)<=20

**Output Format**

print the pattern in a single line where each row of the pattern is separated by a "$" instead of a new line.

**Sample Input 0**

PROGRAM

**Sample Output 0**

G$ GR$ GRA$ GRAM$ GRAMP$ GRAMPR$ GRAMPRO$

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);

String str=sc.next();

int mid=str.length()/2;

String pattern="";

for(int i=mid;i<str.length();i++)

{

pattern+=str.charAt(i);

System.out.print(pattern+"$"+" ");

}

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

{

pattern+=str.charAt(i);

System.out.print(pattern+"$"+" ");

}

}

}

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);

String str=sc.next();

int mid=str.length()/2;

String pattern="";

int i=mid;

for(int j=0;j<str.length();j++)

{

pattern+=str.charAt(i);

System.out.print(pattern+"$"+" ");

if(i==str.length()-1)

{

i=0;

}

else{

i++;

}

}

}

}

Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

**Input Format**

A string s consisting of only lowercase English letters.

**Constraints**

1 <= s.length <= 105 s consists of only lowercase English letters.

**Output Format**

An integer representing the index of the first non-repeating character in the string. If it does not exist, return -1.

**Sample Input 0**

leetcode

**Sample Output 0**

0

**Sample Input 1**

loveleetcode

**Sample Output 1**

2

**Sample Input 2**

aabb

**Sample Output 2**

-1

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) {

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

Scanner sc=new Scanner(System.in);

String str=sc.next();

System.out.print(non\_repeat(str));

}

public static int non\_repeat(String str)

{

int arr[]=new int[26];

int n=0;

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

{

int index=str.charAt(i)-'a';

arr[index]++;

}

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

{

if(arr[str.charAt(i)-'a']==1)

{

return i;

}

}

return -1;

}

}

Given two strings s and t, return true if they are equal when both are typed into empty text editors. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

**Input Format**

The input consists of two strings s and t, each on a separate line.

**Constraints**

1 <= s.length, t.length <= 200 s and t only contain lowercase letters and '#' characters.

**Output Format**

The output is a boolean value (true or false) indicating whether the two strings are equal after processing the backspaces.

**Sample Input 0**

ab#c

ad#c

**Sample Output 0**

true

**Sample Input 1**

ab##

c#d#

**Sample Output 1**

true

**Sample Input 2**

a#c

b

**Sample Output 2**

false

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) {

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

Scanner sc=new Scanner(System.in);

String str1=sc.next();

String str2=sc.next();

System.out.print(backspace\_str(str1,str2));

}

public static boolean backspace\_str(String str1,String str2)

{

int count1=0,count2=0;

for(int i=0;i<str1.length();i++)

{

if(str1.charAt(i)=='#')

{

count1++;

}

}

for(int i=0;i<str2.length();i++)

{

if(str2.charAt(i)=='#')

{

count2++;

}

}

if(count1==count2)

{

return true;

}

else

{

return false;

}

}

}

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) {

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

Scanner sc=new Scanner(System.in);

String str1=sc.next();

String str2=sc.next();

StringBuffer sb1 = new StringBuffer(str1);

StringBuffer sb2 = new StringBuffer(str2);

String s1=backspace\_str(sb1).toString();

String s2=backspace\_str(sb2).toString();

if((s1).equals(s2))

{

System.out.print("true");

}

else

{

System.out.print("false");

}

}

public static StringBuffer backspace\_str(StringBuffer str)

{

int i=0;

while(i<str.length())

{

if(str.charAt(i)=='#'&&i>0)

{

str.deleteCharAt(i);

str.deleteCharAt(i-1);

i--;

}

else

{

i++;

}

}

return str;

}

}

Imagine you are developing a utility for a software application that processes large datasets of integers. One of the features required is to sort these integers such that the even numbers are in descending order and the odd numbers are in ascending order. This feature is essential for a specific analytical task where the ordering of numbers based on their parity and magnitude provides meaningful insights.

**Input Format**

An integer n representing the number of elements in the array. An array of n integers.

**Constraints**

No Constraints

**Output Format**

A single line of space-separated integers representing the sorted array, where even numbers are in descending order, and odd numbers are in ascending order.

**Sample Input 0**

4

1 2 3 4

**Sample Output 0**

4 2 1 3

**Sample Input 1**

6

2 4 6 8 12 56

**Sample Output 1**

56 12 8 6 4 2

package arrays;

import java.util.Scanner;

public class EvenDescOddAsec {

    public static void main(String args[])

    {

       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();

        }

        for(int i=0;i<n;i++)

        {

            if(arr[i]%2==0)

            {

                arr[i]=arr[i]\*(-1);

            }

        }

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<n-1-i;j++)

            {

                if(arr[j]>arr[j+1])

                {

                  int temp=arr[j];

                  arr[j]=arr[j+1];

                  arr[j+1]=temp;

                }

            }

        }

            for(int i=0;i<n;i++)

        {

            if(arr[i]%2==0)

            {

                arr[i]=arr[i]\*(-1);

            }

        }

            for(int item:arr)

            {

                System.out.print(item+" ");

            }

    }

}

Given a binary array nums and an integer k, return the maximum number of consecutive 1's in the array if you can flip at most k 0's.

**Input Format**

First Line: An integer size representing the size of the binary array nums. Second Line: An integer k representing the maximum number of 0s that can be flipped to 1s. Third Line: size integers (space-separated) representing the elements of the binary array nums.

**Constraints**

1 <= nums.length <= 105 nums[i] is either 0 or 1. 0 <= k <= nums.length

**Output Format**

The output consists of a single integer representing the maximum number of consecutive 1s that can be achieved by flipping at most k zeros.

**Sample Input 0**

11

2

1 1 1 0 0 0 1 1 1 1 0

**Sample Output 0**

6

Scanner sc=new Scanner(System.in);

        int n=sc.nextInt();

        int k=sc.nextInt();

        int arr[]=new int[n];

        for(int i=0;i<n;i++)

        {

            arr[i]=sc.nextInt();

        }

        int l=0,r=0,max=0,zero=0;

        while(r<arr.length)

        {

            if(arr[r]==0)

            {

              zero++;

            }

            while(zero>k)

            {

                if(arr[l]==0)

                {

                    zero--;

                }

                l++;

            }

            max=Math.max(max,r-l+1);

            r++;

                }

                System.out.println(max);

You are tasked with developing a function as part of a larger system managing a virtual environment, simulating physical objects and their interactions. In this virtual environment, objects are represented by integers in an array. Each object's movement or action is symbolized by its numerical value, except for the number 0, which represents an object at rest or a space void of any object.

Your objective is to write an algorithm that rearranges the objects in the environment, pushing all the "inactive" or "rest" objects (denoted by 0) to the end of the environment, while ensuring that the active objects maintain their initial sequence of action or movement. This rearrangement must be done in place to optimize memory usage, as the virtual environment is designed to run on devices with limited memory capacity.

**Input Format**

The first line contains an integer N, the size of the array. The second line contains N space-separated integers representing the array elements.

**Constraints**

1 <= nums.length <= 104 -231 <= nums[i] <= 231 - 1

**Output Format**

A single line containing N space-separated integers representing the array after moving all `0's to the end, maintaining the relative order of the non-zero elements.

**Sample Input 0**

5

12 0 9 0 8

**Sample Output 0**

12 9 8 0 0

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) {

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

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();

}

int index=0;

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

{

if(arr[i]!=0)

{

int temp=arr[i];

arr[i]=arr[index];

arr[index]=temp;

index++;

}

}

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

{

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

}

}

}

You are given two strings s and t.

String t is generated by random shuffling string s and then add one more letter at a random position.

Return the letter that was added to t.

**Input Format**

A string s, representing the original string. A string t, representing the shuffled string with one additional letter.

**Constraints**

0 <= s.length <= 1000 t.length == s.length + 1 s and t consist of lowercase English letters.

**Output Format**

A character, representing the letter that was added to t.

**Sample Input 0**

"abcd"

"abcde"

**Sample Output 0**

e

**Sample Input 1**

""

"y"

**Sample Output 1**

y

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static char findTheDifference(String s, String t) {

int sumS = 0, sumT = 0;

for (char c : s.toCharArray()) {

sumS += c;

}

for (char c : t.toCharArray()) {

sumT += c;

}

return (char) (sumT - sumS);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s = sc.next();

String t = sc.next();

System.out.println(findTheDifference(s, t));

}

}