CodeQuotient

Department of Computer Science & Engineering

LAB MANUAL

22CS025 - Algorithm Design & Implementation-5Sem-2022

Student Name	**BHUVESH MITTAL**
Email	**bhuvesh1450.be22@chitkara.edu.in**
Course Name	**22CS025 - Algorithm Design & Implementation-5Sem-2022**

Faculty Incharge

_____

Table of Contents

19	Spell the number	Incorrect Answer
20	Check if strings are rotations or not	30
21	First Unique Character	31
22	Find all pairs with K difference	32
23	Find out the winner	33 - 34
24	Unique characters or not	35
25	Count frequency of each character	36
26	Kth distinct element	37 - 38
27	Maximum Frequency in a sequence	39 - 40
28	Check if two arrays are equal or not	41
29	Pair with the given sum	42 - 43
30	Array Pair Sum Divisibility Problem	44
31	Largest subarray with 0 (ZERO) sum	45
32	Find K largest elements in array	46 - 47
33	Convert min Heap to max Heap	48 - 49
34	Check if a given tree is max-heap or not	50
35	Connect the sticks of different lengths with minimum cost	51 - 52
36	Implement Priority Queue using Linked List	53 - 54
37	Sort an array using heap sort	55
38	Create a binary tree from array	56 - 57

57	Find a lowest common ancestor of a given two nodes in a binary search tree	94 - 95
58	Find the floor and ceil of a key in binary search tree	96 - 97
59	Fractional Knapsack problem	98 - 99
60	Interval scheduling Problem	100 - 101
61	Job Scheduling with deadlines	102 - 103
62	Activity Selection Problem	104 - 105
63	Count number of ways to cover a distance	106
64	Minimum Cost Path to last element of matrix	107
65	Longest Common Subsequence (LCS)	108 - 109
66	The Subset Sum problem	110
67	Matrix Chain Multiplication problem	111 - 112
68	0-1 Knapsack problem	113 - 114
69	Tom And Permutations	115
70	Print all strings of n-bits	116
71	Solve N Queen problem	117 - 118
72	Solve Sudoku	Incorrect Answer
73	Rat in a Maze Problem	119
74	Count word in the board	Incorrect Answer
75	Find the minimum number of edges in a path of a graph	120 - 121
76	Find path in a directed graph	122 - 124

Aim: Factorial using recursion

Write a recursive function **factorial** that accepts an integer *n* as a parameter and returns the factorial of *n*, or *n!*.
A factorial of an integer is defined as the product of all integers from 1 through that integer inclusive. For example, the call of **factorial(4)** should return 1 * 2 * 3 * 4, or 24. The factorial of 0 and 1 are defined to be 1.
You may assume that the value passed is non-negative and that its factorial can fit in the range of type int.

Input Format:
The first line of input contains number of testcases , T.
Then T lines follow, which contains an integer,n.

Output Format:
For each testcase print the factorial in new line

Sample Input
2
4
3

Sample Output
24
6

Solution:

```
class Result{
  /*
   * Complete the function 'factorial' given below
   * @params
   *  n -> an integer whose factorial is to be calculated
   * @return
   *  The factorial of integer n
   */
  static int factorial(int n) {
    // Write your code here
    if(n<=1) return 1;
    return n*factorial(n-1);
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Sum of all the digits using recursion

Write a recursive function **sumOfDigits** that accepts an integer as a parameter and returns the sum of its digits. For example, calling sumOfDigits(1729) should return 1 + 7 + 2 + 9, which is 19. If the number is negative, return the negation of the value. For example, calling **sumOfDigits**(-1729) should return -19.
Constraints: Do not declare any global variables. Do not use any loops; you must use recursion. Also do not solve this problem using a string. You can declare as many primitive variables like ints as you like. You are allowed to define other "helper" functions if you like; they are subject to these same constraints.

Solution:

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
   public static int sum(int n)
   {
     if(n<=0) return 0;
     int digit=n%10;
     n=n/10;
     return digit+sum(n);
   }
   public static void main(String[] args)
   {
       // Write your code here
       Scanner sc=new Scanner(System.in);
       int t=sc.nextInt();
       for(int i=0;i<t;i++){
       int n=sc.nextInt();
       int ans=sum(Math.abs(n));
       if(n<0){
         ans=-ans;
       }
     System.out.println(ans);}
   }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Prime factors using recursion

Write a program that accepts an integer *n* (where n >=2) and print all the prime factors of n using recursion.

Sample Input

24

Sample Output

2
2
2
3

Constraints : Do not declare any global variables. Do not use any loops; you must use recursion. You can declare as many primitive variables like ints as you like. You are allowed to define other "helper" functions if you like; they are subject to these same constraints.

Solution:

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
   public static void prime(int n,int d)
   {
     if(n==1) return;
     if(n%d==0)
     {
       System.out.println(d);
       prime(n/d,d);
     }
     else
     {
       prime(n,d+1);
     }
   }
   public static void main(String[] args)
   {
      // Write your code here
      Scanner sc=new Scanner(System.in);
     int n=sc.nextInt();
     prime(n,2);
   }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: power(base, exp)

Write a recursive function **power** that accepts two integers representing a *base* and an *exponent*, and returns the base raised to that exponent. For example, the call to power(3, 4) should return 3^4 i.e. 81. If the exponent passed is negative, then return -1.
Do not use loops or auxiliary data structures; Solve the problem recursively. Also do not use the provided library pow function in your solution.
Expected Time Complexity: O(log(n)); *here n denotes the exponent*
Input Format:
The first line of input conatins an integer T, denoting the number of test cases.
The second line of input contains 2 integers base and exponent seperated by space.
Output Format:
Print the answer when base is raised to the exponent.
Constraints:
-10 <= base <= 10
þÿ-15 <= exponent <= 15
Sample Input
2 // Test Cases
2 3
5 2
Sample Output
8
25

Solution:

```
class Result {
    static long power(int base, int exp) {
      // Write your code here
      if(exp==0) return 1;
      if(exp<0) return -1;
      return base*power(base,exp-1);
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Form a new number

Write a recursive function evenDigits that accepts an integer parameter n and returns a new integer containing only the even digits from n, in the same order. If n does not contain any even digits, return 0.
For example, the call of evenDigits(8342116) should return 8426 and the call of evenDigits(35179) should return 0.

Solution:

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
  public static int newD(int n)
  {
    if(n==0) return 0;
    int digit=n%10;
    int result=newD(n/10);
    if(digit%2==0)
    {
      return result*10+digit;
    }
    return result;
  }
    public static void main(String[] args)
    {
        // Write your code here
        Scanner sc=new Scanner(System.in);
      int t=sc.nextInt();
      for(int i=0;i<t;i++)
      {
        int n=sc.nextInt();
        System.out.println(newD(n));
      }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Binary equivalent using recursion

Write a recursive function **decimalToBinary** that accepts an integer as a parameter and returns an integer whose digits look like that number's representation in binary (base-2). For example, the call of decimalToBinary(43) should return 101011.
Constraints: Do not use a string in your solution. Also do not use any built-in base conversion functions from the system libraries. solve the problem recursively.
Sample Input :
1 // no. of testcases
43
Sample Output :
101011

Solution:

```
class Result{
   static int decimalToBinary(int n){
    if(n==0 || n==1) return n;
       int digit=n%2;
       int ans=decimalToBinary(n/2);
       return ans*10+digit;
   }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Greatest common divisor using recursion

Write a recursive function **gcd** that accepts two positive non-zero integer parameters *i* and *j* and returns the greatest common divisor of these numbers.

Sample Input

2 // Test Cases
30 18 // i j (testcase 1)
11 17 // i j (testcase 2)þÿ

Sample Output

6
1

Constraints:. Solve the problem recursively.

Solution:

```
class Result
{
  static int gcd(int i, int j)
  {
  if(j==0) return i;
    return gcd(j,i%j);
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find all pairs with sum K

Given a sorted list of **N** integers, find all distinct pairs of integers in the list with sum equal to a given number **K**, with O(n log n) or O(n) time complexity.
Complete the function below which takes the array and K as parameters and return the number of pairs if any and 0 otherwise.

Input Format:

First line of input will contain a positive integer T = number of test cases. Each test case will contain 2 lines.
First line of each test case contains two integers - N and K.
Next line will contain N numbers separated by space in non-decreasing order.

Constraints:

1 "d T "d 10
1 "d N "d 10^5
-(10^9) "d arr[i], K "d 10^9

Output Format:

For each test case, print number of distinct pairs whose sum will be equal to k. A pair must have two numbers at different indices.
Two pairs are different if at least one of the indices in them is uncommon. Indices - (2,3) and (3,2) are not distinct, but (2,3) and (2,4) are.

Sample Input

```
3  // Test Cases
10 11  // N K (testcase 1)
1 2 3 4 5 6 7 8 9 10
5 10   // N K (testcase 2)
2 4 6 8 10
6 27   // N K (testcase 3)
12 15 20 22 34 36
```

Sample Output

```
5
2
1
```

Solution:

```
class Result {
  static int getPairsCount(int arr[], int n, int k) {
  // Write your code here
    HashMap<Integer,Integer> mpp=new HashMap<>();
    int c=0;
    for(int i=0;i<n;i++)
    {
      int diff=k-arr[i];
      if(mpp.containsKey(diff))
      {
        c+=mpp.get(diff);
      }
      mpp.put(arr[i],mpp.getOrDefault(arr[i],0)+1);
    }
    return c;
  }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
};
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find first occurrence of an integer in a sorted list with duplicates

Given a sorted list of integers, find the position of first occurrence of a given number **K** in the list in O(log n) time.

Input Format:

First line of input will contain a positive integer T = number of test cases.
Each test case will contain the following two lines:
First line will contain two positive integer N = number of elements in list and K.
Second line will contain N space separated integers in increasing order.

Constraints:

1 <= N <= 10^5
-(10^9) <= arr[i], K <= (10^9)

Output Format:

For each test case, print on a single line the index of first occurrence of K in the list on 0-based index. Print -1 if you cannot find K in the list.

Sample Input

```
3  // Test Cases
10 4  // N K (testcase 1)
1 2 4 4 4 4 5 8 9 10
15 7  // N K (testcase 2)
1 2 3 3 5 6 7 7 7 7 7 8 8 8 8
9 1   // N K (testcase 3)
-5 -4 -3 -2 -1 0 0 0 1
```

Sample Output

```
2
6
8
```

Solution:

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
    public static int s(int[] arr, int k)
    {
      int low=0;
      int high=arr.length-1;
      int result=-1;
      while(low<=high)
      {
        int mid=low+(high-low)/2;
        if(arr[mid]==k)
        {
           result=mid;
            high=mid-1;
        }
        else if(arr[mid]>k) high=mid-1;
        else low=mid+1;
      }
      return result;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    }
    public static void main(String[] args)
    {
       // Write your code here
      Scanner sc=new Scanner(System.in);
      int t=sc.nextInt();
      for(int i=0;i<t;i++)
      {
        int n=sc.nextInt();
        int k=sc.nextInt();
        int arr[]=new int[n];
        for(int j=0;j<n;j++)
        {
          arr[j]=sc.nextInt();
        }
        System.out.println(s(arr,k));
      }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find count of a number in a sorted list with duplicates

Given a sorted list of integers with duplicates, find the count of a given number **K** in that list in O(log n) time.

Input

First line of input will contain a positive integer T = number of test cases. Each test case will contain 2 lines.
First line of each test case will contain two number N = number of elements in list and K separated by space.
Next line will contain N space separated integers.

Constraints

1 <= N <= 10^5
-(10^9) <= arr[i], K <= (10^9)

Output

For each test case, print on a single line, the count of number K in this list.

Sample Input

```
3  // Test Cases
10 5  // N K (testcase 1)
1 2 2 5 5 5 7 7 7 8
10 1  // N K (testcase 2)
1 1 1 1 1 1 1 2 2 3
20 2  // N K (testcase 3)
1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4
```

Sample Output

```
3
7
5
```

Solution:

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
    public static int f(int[] arr, int k)
    {
      int low=0;
      int high=arr.length-1;
      int first=-1;
      while(low<=high)
      {
        int mid=low+(high-low)/2;
        if(arr[mid]==k)
        {
          first=mid;
          high=mid-1;
        }
        else if(arr[mid]>k) high=mid-1;
        else low=mid+1;
      }
      return first;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    }
  public static int l(int[] arr, int k)
  {
    int low=0;
    int high=arr.length-1;
    int last=-1;
    while(low<=high)
    {
      int mid=low+(high-low)/2;
      if(arr[mid]==k)
      {
        last=mid;
        low=mid+1;
      }
      else if(arr[mid]>k) high=mid-1;
      else low=mid+1;
    }
    return last;
  }
    public static void main(String[] args)
    {
        // Write your code here
      Scanner sc=new Scanner(System.in);
      int t=sc.nextInt();
      for(int i=0;i<t;i++)
      {
        int n=sc.nextInt();
        int k=sc.nextInt();
        int[] arr=new int[n];
        for(int j=0;j<n;j++)
        {
          arr[j]=sc.nextInt();
        }
        int first=f(arr,k);
        if(first==-1) System.out.println(0);
        else System.out.println(l(arr,k)-first+1);
      }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Rotation count of a sorted Array

Given an array of integers, find the minimum number of rotations performed on some sorted array to create this given array.

Input

First line of input will contain a number T = number of test cases. Each test case will contain 2 lines. First line will contain a number N = number of elements in the array (1 <= N <= 50). Next line will contain N space separated integers.

Complete the function **int rotationCount(int array[], int size)** which will receive array and size of the array as input and return how many times the sorted array is rotated. Function should return -1 if the array is not rotated.

Output

Print one line of output for each test case with the minimum number of rotations for given array.

Sample Input:

2
6
15 18 3 4 6 12
6
1 2 3 4 5 6

Sample Output

2
-1

Solution:

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
    public static int check(int[] arr)
    {
      int min=Integer.MAX_VALUE;
      int index=-1;
      for(int i=0;i<arr.length;i++)
      {
        if(arr[i]<min)
        {
          min=arr[i];
          index=i;
        }
      }
      return index==0?-1:index;
    }
    public static void main(String[] args)
    {
        // Write your code here
      Scanner sc=new Scanner(System.in);
      int t=sc.nextInt();
      for(int i=0;i<t;i++)
      {
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        int n=sc.nextInt();
        int[] arr=new int[n];
        for(int j=0;j<n;j++)
        {
          arr[j]=sc.nextInt();
        }
        System.out.println(check(arr));
      }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Search element in a rotated sorted array

Given an array of **n** integers which is sorted but rotated by some number of times after sorting, and a integer **k**. Search the element k in this sorted rotated array efficiently. For example, suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).
Assume there are no duplicate elements in the array.
Expected Time Complexity: O(log(N))
Expected Space Complexity: O(1)
Input Format
First line of input will contain a number T = number of test cases. Each test case will contain 3 lines. The first line will contain an integer k to be searched. Second line will contain a number n = number of elements in the array. Next line will contain N space separated integers.
Complete the function int searchRotatedSortedArray() to search a value target in array a of size given, and if target found in the array return its index, otherwise return -1.
You may assume no duplicate exists in the array.
Output Format
Print the index of k in given array for each test case in new line if found and print -1 if k is not present in given array.
Constraints
1 <= T <= 10
-1000 <= k <= 1000
1 <= n <= 10^5
-1000 <= arr[i] <= 1000
Sample Input
```
2   // Test Cases
3             // k (testcase 1)
6             // n
15 18 2 3 6 12  // arr[]
7             // k (testcase 2)
7             // n
4 5 8 9 1 2 3   // arr[]
```
Sample Output
```
3
-1
```

Solution:

```
class Result {
  static int searchRotatedSortedArray(int arr[], int k) {
    // Write your code here
    int low=0;
    int high=arr.length-1;
    while(low<=high)
    {
      int mid=low+(high-low)/2;
      if(arr[mid]==k) return mid;
      if(arr[low]<=arr[mid])
      {
        if(arr[low]<=k && k<=arr[mid])
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        {
          high=mid-1;
        }
        else
        {
          low=mid+1;
        }
      }
      else
      {
        if(arr[mid]<=k && k<=arr[high]) low=mid+1;
        else high=mid-1;
      }
    }
    return -1;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Reverse the order of words of a string

Given a string of words, reverse the order of words in the string individually, not the whole string.
Complete the function **revWordsOrder(str)** that accepts a string as parameter and reverses the order of words of string.
Input Format:
The first line of input contains an integer T denoting the no of test cases . Then T test cases follow. Each test case contains the string str.
Output Format:
For each test case, print the resultant string in new lines.
Sample Input
2
Code Quotient Loves Code
Hello Coders
Sample Output
Code Loves Quotient Code
Coders Hello

Solution:

```
class Result {
  static String revWordsOrder(String str) {
    // Write your code here
    String word="";
    String ans="";
    for(int i=0;i<str.length();i++)
    {
      char a=str.charAt(i);
      if(a==' ')
      {
        ans=word+" "+ans;
        word="";
      }
      else
      {
        word+=a;
      }
    }
    ans=word+" "+ans;
    ans=ans.substring(0,ans.length()-1);
    return ans;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: String is subsequence or not

Given two strings, find whether second string is subsequence of first string. A subsequence of a string is defined as a string that can be obtained by deleting any number of characters from the original string.
Complete the function **strSubsequence(str1, str2)** that accepts two strings as parameters and print **YES** if str2 is a subsequence of str1 and **NO** otherwise.

Input Format

The first line of input contains an integer T denoting the no of test cases . Then T test cases follow.
Each test case contains a single line of input which contains two strings str1, str2 seperated by a space.

Output Format

For each test case, print YES or NO in new lines.

Constraints

1 <= T <= 10
Given strings consist of uppercase and lowercase English letters.

Sample Input

3
CodeQuotient CQ
CodeQuotient QC
Hello Hi

Sample Output

YES
NO
NO

Solution:

```
class Result{
  // Return true if the str2 is a subsequence of str1, else return false
  static boolean strSubsequence(String str1, String str2) {
    // Write your code here
    int j=0;
    for(int i=0;i<str1.length()&&j<str2.length();i++)
    {
      char a=str1.charAt(i);
      char b=str2.charAt(j);
      if(a==b)
      {
        j++;
      }
    }
    if(j==str2.length()) return true;
    return false;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Technical Issue With The Keyboard

Aman likes to play with strings, so his friend Riya decided to send him a mail, containing a sorted string with unique characters. But, due to some technical issue with her keyboard, whenever she presses a key, the corresponding character gets printed multiple times. Now Riya needs your help for removing all the unwanted characters from her string. For example, If **aabbbccdef** is a sorted string generated by Riya, then you should output the following string as answer **abcdef**.
Complete the function **getDesiredString(str)** that will take the string as parameter and modify it as specified.
Input Format:
The first line of input contains an integer T denoting the no of test cases . Then T test cases follow. Each test case contains a string str.
Output Format:
For each test case, print the new string in new lines.
Constraints:
1 <= T <= 10
1 <= length of string <= 10^5
Each string will contain only lower-case English letters.þÿ
Sample Input
2
aabbccdef
abddddddd
Sample Output
abcdef
abd

Solution:

```
class Result {
  // Return the updated string
  static String getDesiredString(String str) {
    // Write your code here
    TreeMap<Character,Integer> mpp=new TreeMap<>();
    for(int i=0;i<str.length();i++)
    {
      char a=str.charAt(i);
      mpp.put(a,mpp.getOrDefault(a,0)+1);
    }
    String ans="";
    for(Map.Entry<Character,Integer> hm:mpp.entrySet())
    {
      ans+=hm.getKey();
    }
    return ans;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Implement atoi and itoa functions

Implement the below functions with recursion from string library as your own functions.
1. itoa() function converts int data type to string data type.
2. atoi() function converts string data type to int data type.

Input Format:

The first line of input contains an integer T denoting the no of test cases . Then T test cases follow. Each test case contains a string str and a number num.
Function void itoa(int num, char* result) will take the number as parameter and place the string from this number in array result.
Function int atoi(char *str) will take the string as parameter and return the string as an integer value.

Output Format:

For each test case, print the output of itoa() and atoi() in new lines.

þÿSample Input

1
"100" 135

Sample Output

100 "135"

Solution:

```
class Result {
  static String itoa(int num) {
    // Write your code here
    String ans=""+num;
    return ans;
  }
  static int atoi(String str) {
    // Write your code here
    int num=0;
    for(int i=0;i<str.length();i++)
    {
      int d=str.charAt(i)-'0';
      num=num*10+d;
    }
    return num;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Implement strcat function

Implement the strcat() function from string library as your own function. The function will take two strings as arguments and concatenate the second string at the end of first string.

Input Format:

The first line of input contains an integer T denoting the no of test cases .
Then T test cases follow. Each test case contains two strings str2 and str1.
Function strcatCode(str1,str2) will take two parameters and concatenate the str2 string at end of str1.

Output Format:

For each test case, print the concatenated string in new lines.

Sample Input

1
Code Quotient

Sample Output

CodeQuotient

Solution:

```
static String strcatCode(String a, String b) {
  // Write your code here
  return a+b;
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Count words

Write a function **countWords()** to count the numbers of words in a string.
A word is defined as text separated by space(' ') or multiple spaces.
The function will receive a string as input and return the numbers of words in this string.
Input Format
A single line of input which consists of the string whose words is to be counted
Output Format
Print the count the numbers of words in a string
Sample Input
Codequotient get better at coding
Sample Output
5

Solution:

```
class Result {
  static int countWords(String str) {
  if(str.equals("")){
      return 0;
    }
    String words[]=  str.split("\\s+");
    return words.length;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Check if strings are rotations or not

Given two strings, find whether both are rotations of each other or not. For example,
Given str1 = abacd and str2 = acdab, we can get str1 by rotating str2 and,
Given str1 = coder and str2 = cored, we can not get str1 by rotating str2.

þÿInput Format

The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains two strings str1 and str2.

Output Format

For each test case, print YES or NO in new lines.

Sample Input

```
2  // Test Cases
abacd  // testcase 1
acdab
coder  // testcase 2
cored
```

Sample Output

```
YES
NO
```

Solution:

```
class Result {
  // return 1 for YES and 0 for NO.
  static int isRotation(String str1, String str2) {
    // Write your code here
    if(str1.length()!=str2.length()) return 0;
    String a=str2+str2;
    if(a.contains(str1)) return 1;
    return 0;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: First Unique Character

Given a string that contains only lowercase English letters, find the first non-repeating character in it and return its index. If it doesn't exist, then return -1.

Input Format:

First line contains a string.

Output Format:

Print the index of the first non-repeating character in the given string. If it doesn't exist, then print -1.

Constraints:

'a' <= str[i] <= 'z'

1 <= length of str <= 100000

Sample Input 1

codequotientchamp

Sample Output 1

2

Explanation 1

'd' is the first non-repeating character in the given string, therefore the output is its index i.e. 2.

Sample Input 2

silentlisten

Sample Output 2

-1

Explanation 2

All the characters in the given string are repeating, therefore the output is -1.

Solution:

```
import java.util.*;
class Result {
  static int firstUniqueChar(String str) {
    // Write your code here
    HashMap<Character,Integer> map=new HashMap<>();
      for(char a:str.toCharArray())
      {
         map.put(a,map.getOrDefault(a,0)+1);
      }
      for(int i=0;i<str.length();i++)
      {
         if(map.get(str.charAt(i))==1) return i;
      }
      return -1;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find all pairs with K difference

Given an array of **N** distinct integers, find all the pairs of integers in it, with the difference equals to a given number **K**.
Complete the function in the editor, which takes the array and K as parameters and return the number of pairs with the difference K.

Input Format:

First line of input will contain a positive integer T = number of test cases. Each test case will contain 2 lines.
First line of each test case contains two positive integers, N and K.
Next line will contain N distinct numbers separated by space.

Output Format:

For each test case, print number of pairs whose difference will be equal to k.

Constraints:

1 "d T "d 10
1 "d N "d 10^5
1 "d K "d 10^8
-(10^7) "d arr[i] "d 10^7

Sample Input

3 // Test cases
10 7 // N K (testcase 1)
1 2 3 4 5 6 7 8 9 10
5 4 // N K (testcase 2)
4 2 3 1 10
6 27 // N K (testcase 3)
10 15 38 22 11 36

Sample Output

3
0
1

Solution:

```
class Result {
  static int getPairsCount(int arr[], int n, int k) {
    // Write your code here
    HashSet<Integer>set=new HashSet<>();
      int count=0;
      for(int i:arr)
      {
          int diff1=k+i;
          int diff2=i-k;
          if(set.contains(diff1)) count++;
          if(set.contains(diff2)) count++;
          set.add(i);
      }
      return count;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find out the winner

Given an array of strings, find out the string which occurs maximum number of times. If two strings occurs maximum times, return the alphabetically later string. For example,
if array is ["Amit", "Girdhar", "Amit", "Girdhar", "Girdhar", "Amit", "Amit"] then return "Amit", and
if array is ["Amit", "Girdhar", "Amit", "Girdhar", "Girdhar", "Amit"] then return "Girdhar" as both occurs 3 times but Girdhar comes after Amit in alphabetical order.
So, write a function which accepts a string array as input and return the output string.

Input Format

The first line contains an integer n, the number of names in string array.
Each the n subsequent lines contains a string describing array[i] where 0 "d i < n.

Output Format

Print the output string

Constraints

1 "d n "d 10^5
1 "d length of string "d 64
string will contain only lowercase english alphabets i.e. from 'a' to 'z'

Sample Input

6
Amit
Girdhar
Amit
Girdhar
Girdhar
Amit

Sample Output

Girdhar

Solution:

```
class Result{
  /*
   * Complete the 'inspectStrings' function below.
   * @params
   *   words -> A string array, which contains the strings
   *
   * @return
   *   A String, which occurs the maximum numbers of times
   */
  static String inspectStrings(String[] words) {
    // Write your code here
      TreeMap<String,Integer> map=new TreeMap<>();
      for(String str:words)
      {
          map.put(str,map.getOrDefault(str,0)+1);
      }
      String ans="";
      int maxi=Integer.MIN_VALUE;
      for(Map.Entry<String,Integer> entry:map.entrySet())
      {
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            if(entry.getValue()>=maxi)
            {
                maxi=entry.getValue();
                ans=entry.getKey();
            }
        }
        return ans;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Unique characters or not

Given a string, you need to test the characters for their uniqueness. If all the characters occur at most 1 time in the string, then print “YES”, otherwise if some character occurs at least twice in the string print “NO”.

Input Format:

The first line of input contains an integer T denoting the no of test cases . Then T test cases follow. Each test case contains the string str.

Function void isUniqueChars(char *str) will take the string as parameter and print YES or NO according to the occurence of characters in it.

Output Format:

For each test case, print YES or NO in new lines.

Constraints:

1 <= T <= 10

Given string can contain any valid ASCII character.

Sample Input

2

CodeQuotient

Coding

Sample Output

NO

YES

Solution:

```
import java.util.*;
class Result{
  // Return true if string contains all unique characters, else return false
  static boolean isUniqueChars(String str){
    // Write your code here
      HashMap<Character,Integer> map=new HashMap<>();
      for(char ch:str.toCharArray())
      {
          map.put(ch,map.getOrDefault(ch,0)+1);
          if(map.get(ch)>1) return false;
      }
      return true;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Count frequency of each character

Given a string that contains only lowercase characters. Write a program to print all the distinct characters along with their frequency in the order of their occurrence. For example if the string is “helloworld”, then you should print **h1 e1 l3 o2 w1 r1 d1**
Complete the given function **countFrequency()** and print the frequency of each character as per the given statement.

Input Format

First line contains a string with lowercase characters.

Constraints

'a' <= str[i] <= 'z'
1 <= length of str <= 100000

Output Format

Print all the distinct characters along with their frequency followed by a space. And the characters must be printed in the order of their occurrence.

Sample Input

codequotient

Sample Output

c1 o2 d1 e2 q1 u1 t2 i1 n1

Solution:

```
class Result {
  static void countFrequency(String str) {
    // Write your code here
      LinkedHashMap<Character,Integer> map=new LinkedHashMap<>();
      for(char ch:str.toCharArray())
      {
         map.put(ch,map.getOrDefault(ch,0)+1);
      }
      for(Map.Entry<Character,Integer> entry:map.entrySet())
      {
         System.out.print(entry.getKey()+""+entry.getValue()+" ");
      }
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Kth distinct element

Given an array of positive integers, and an integer **K**, your task is to find the **K**th distinct element present in the array. If there are fewer than K distinct elements, then return 0 as the answer.
A distinct element is an element which is present only once in an array.
Note: The elements are considered in the same order in which they appear in the array from left to right.
Example: arr[] = {6, 11, 4, 11, 9, 4}, K = 2
The only distinct elements in the array are 6 and 9.
6 appears first so it the 1st distinct element, and 9 appears second so it the 2st distinct element in the array. Hence, for K=2 the answer is **9**.
Input Format:
First line of input contains T = number of test cases.
Each test case contains three lines:
First Line will contain an integer N, denoting the size of the array.
Second line contains N integers separated by space, denoting the array elements.
Third line contains an integer representing K.
Constraints:
1 <= T <= 10
1 <= N <= 10^5
1 <= arr[i] <= 10^5
1 <= K <= N
Output Format:
Print the Kth distinct element present in the array.
Sample Input 1

```
3  // Test Cases
6          // N (testcase 1)
6 11 4 11 9 4 // arr[]
2          // Kþÿþÿ
5          // N (testcase 2)
7 6 7 3 6     // arr[]
1          // K
6          // N (testcase 3)
8 5 3 5 5 5   // arr[]
4          // K
```

Sample Output

```
9
3
0
```

Explanation
Testcase 1:
9 is the 2nd distinct element present in the array from left to right

Testcase 2:
3 is the first distinct element present in the array from left to right

Testcase 3:
Only 2 distinct elements are present in the array, i.e., 8 and 3
Since there are less than 4 distinct elements, therefore the answer is 0

BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Solution:

```
class Result {
  static int kthDistinctElement(int arr[], int N, int K) {
    // Write your code here
      LinkedHashMap<Integer,Integer> map=new LinkedHashMap<>();
      for(int i:arr)
      {
          map.put(i,map.getOrDefault(i,0)+1);
      }
      int count=0;
      for(Map.Entry<Integer,Integer> entry:map.entrySet())
      {
          if(entry.getValue()==1) count++;
          if(count==K) return entry.getKey();
      }
      return 0;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Maximum Frequency in a sequence

Given a list of integers, find out the number that has the highest frequency. If there are more than one such numbers, output the smaller one.

Input:

First line of input will contain an integer T = number of test cases.
Each test case will contain two lines:
First line will contain an integer N = number of elements in the sequence.
Next line will contain N space separated integers of sequence A.

Output:

For each test case, print on a single line, the smallest number with highest frequency in the sequence.

Constraints

1 <= T <= 100
1 <= N <= 10^5
0 <= A[i] <= 1000

Sample Input

3 // No. of test cases
7
2 4 5 2 3 2 4
6
1 2 1 1 2 1
10
1 1 1 1 1 1 1 1 1 1

Sample Output

2
1
1

Solution:

```
class Result {
  static int maxFrequency(int A[], int n) {
    // Write your code here
    TreeMap<Integer,Integer> map=new TreeMap<>();
      for(int i:A)
      {
         map.put(i,map.getOrDefault(i,0)+1);
      }
      int maxi=Integer.MIN_VALUE;
      int ans=0;
      for(Map.Entry<Integer,Integer>entry:map.entrySet())
      {
         if(entry.getValue()>maxi)
         {
            maxi=entry.getValue();
            ans=entry.getKey();
         }
      }
      return ans;
  }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Check if two arrays are equal or not

Given two array of same size, find out if given arrays are equal or not.
Note: Two arrays are said to be equal if both of them contain same set of elements, although arrangements (or permutation) of elements may be different.
Note : If there are repetitions, then counts of repeated elements must also be same for two array to be equal..
For example, if A = [11, 12, 13] and B = [12, 11, 13] then answer is 1 and,
if A = [11, 12, 13, 12, 13] and B = [12, 11, 13, 12, 13] then answer is 1 and,
if A = [11, 12, 13] and B = [12, 13, 12] then answer is 0.
Complete the function **arraysEqualorNot()** given in the editor, which takes two arrays as input and returns the answer(0/1) as output.
Input Format
The 1st line contains an integer N, the number of elements in A and B.
The 2nd line contains N integers separated by space.
The 3rd line contains N integers separated by space.
Output Format
Print 1 or 0 as per the condition.
Constraints
1 <= N <= 10^5
-50000 <= A[i], B[i] <= 50000
Sample Input
3 // N
11 12 13 // A[]
12 11 13 // B[]
Sample Output
1

Solution:

```
class Result {
  static int arraysEqualorNot(int[] A, int[] B) {
    // Write your code here
    TreeMap<Integer,Integer> map1=new TreeMap<>();
    TreeMap<Integer,Integer> map2=new TreeMap<>();
      for(int i:A)
      {
         map1.put(i,map1.getOrDefault(i,0)+1);
      }
      for(int i:B)
      {
         map2.put(i,map2.getOrDefault(i,0)+1);
      }
      if(map1.equals(map2)) return 1;
      return 0;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Pair with the given sum

Given a sorted array of **n** elements, and an integer **k**. Find whether there exists any two elements in the array that sums up to the given value k. If such a pair is found, then print "Found", else print "Not Found".
Note: You can not use the *same* index element twice.
Input Format
First line will contain an integer T, denoting the number of test cases.
For each test case:
First line will contain an integer n, denoting the number of elements in the array.
Second line will contain n space separated integers that denote the array elements.
Third line contains an integer k, that denotes the target sum.
Output Format
For each test case, print "Found" if there exists at least one pair whose sum equals to k, else print 'Not Found'.
Constraints
1 <= T <= 10
1 <= n <= 10^5
-(10^9) <= arr[i] <= 10^9
-(10^9) <= k <= 10^9
Sample Input
2 // Test Cases
5 // n (testcase 1)
1 3 4 6 7 // arr[]
9 // k
6 // n (testcase 2)
-3 -1 4 5 7 12 // arr[]
7 // k
Sample Output
Found
Not Found
Explanation:
Testcase 1:
Sum of 3 and 6 in the given array is 9.

Testcase 2:
No pair in the given array has sum equals to 7.

Solution:

```
class Result {
  // Return true if a pair with the sum k is found, else return false
  static boolean pairSum(int arr[], int n, int k) {
    // Write your code here
      HashSet<Integer> set=new HashSet<>();
      for(int i:arr)
      {
         int diff=k-i;
         if(set.contains(diff)) return true;
         set.add(i);
      }
      return false;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Array Pair Sum Divisibility Problem

Given an array of size n (which is always a even number in this case) and an integer k.
Complete the function that returns true if given array can be divided into pairs such that sum of every pair is divisible by k.
If such pairing of array is possible return 1 else return 0.

Input Format

First Line will contain an integer N denoting the size of array.
Second line contains N integers separated by space as elements of array.
Third line contains an integer k.

Output Format

Print the result as specified above.

Constraints

1 <= N <= 10^5
1 <= a[i] <= 10^5
1 <= k <= 10^5

Sample Input

4
8 4 5 7
4

Sample Output

1

Explanation:

The array can be divided as
(8,4) and (5,7) whose sum is 12, which is divisible by 4.

Solution:

```
class Result
{
  static int isArrayDivisibleInPairs(int a[], int n, int k){
    // Write your code here
      int[] div=new int[k];
      for(int i:a)
      {
         div[i%k]++;
      }
      for(int i=0;i<=k/2;i++)
      {
         if(i==0) {if(div[i]%2!=0) return 0;}
         else if(i==k-i){ if(div[i]%2!=0) return 0;}
         else{
            if(div[i]!=div[k-i]) return 0;
         }
      }
      return 1;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Largest subarray with 0 (ZERO) sum

Given an unsorted array of integers. Find the largest sub-array which adds to 0 (ZERO). For example,
Given **A = {11, 2, -2, 10, 1, -4, -7, 19}** , Print "**6**" as the sum 0 is found between indexes 1 and 6. Note that from index 1 to index 2 is also resulting in 0, but we need to print the largest.
If no sub-array found print **-1**.

Input Format

First Line will contain an integer N denoting the number of elements.
Second line contains N integers separated by space.

Output Format

Print the length of the largest sub-array as shown if found otherwise print -1.

Sample Input

8
11 2 -2 10 1 -4 -7 19

Sample Output

6

Solution:

```
class Result
{
  static int largSubArray(int ar[], int n){
    // Write your code here
      HashMap<Integer,Integer> map=new HashMap<>();
      int maxi=-1;
      int sum=0;
      for(int i=0;i<n;i++)
      {
          sum+=ar[i];
          if(sum==0)
          {
              maxi=i+1;
          }
          if(map.containsKey(sum))
          {
              int len=i-map.get(sum);
              maxi=Math.max(maxi,len);
          }
          else
          {
              map.put(sum,i);
          }
      }
      return maxi;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find K largest elements in array

Given an array of integer elements and an integer k, print the k largest elements from array separated by space in descending sorted order. For example, if array a = [2, 56, 3, 87, 42, 1, 45, 8, 2, 98] and k=3 then output must be 98 87 56.

Input Format:

First line of input contain T = number of test cases. Each test case contain two lines, in first line, total number of elements in array and value of k are provided and next line will contain the elements.

The array of numbers and integer k is given to you.

Note: Do not write the whole program, just complete the functions void printKLargest(int array[], int k) which takes the array and integer k as parameters and print the k largest elements in descending order separated by space.

Note: Do not read any input from stdin/console. Just complete the functions provided. You can write more functions if required, but just above the given function.

Output Format:

Print the k largest element in descending order separated by space.

Do not print the space after last element.

Sample Input

2
10 3
2 56 3 87 42 1 45 8 2 98
6 2
1 87 2 67 3 45

Sample Output

98 87 56
87 67

Solution:

```
static void printKLargest(int array[], int n, int k){
   // Write your code here
   PriorityQueue<Integer> pq=new PriorityQueue<>();
   for(int i=0;i<k;i++)
   {
      pq.add(array[i]);
   }
   for(int i=k;i<n;i++)
   {
      int a=pq.poll();
      a=Math.max(a,array[i]);
      pq.add(a);
   }
   int[] a=new int[k];
   for(int i=0;i<k;i++)
   {
      a[i]=pq.poll();
   }
   for(int i=k-1;i>0;i--)
   {
      System.out.print(a[i]+" ");
   }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    System.out.print(a[0]);
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Convert min Heap to max Heap

Given array representation of a min-heap convert it in a max-heap representation. For example, if array a = [13 15 19 16 18 120 110 112 118] the array must be modified as a = [120 118 110 112 18 19 13 15 16].
The array of numbers is given to you. Do not write the whole program, just complete the functions **void modifyMintoMax(int array[], int n)** which takes the array and total number of elements n as parameters and modify the array elements with **heapify()** to convert it in max-heap.
Note: Do not read any input from stdin/console. Just complete the functions provided. You can write more functions if required, but just above the given function.
Input Format:
First line of input contain T = number of test cases.
Each test case contain two lines, in first line, total number of elements in array and next line will contain the elements.
Output Format:
For each test case, print the max-heap elements separated by space in new lines.
Sample Input
2
9
13 15 19 16 18 120 110 112 118
6
1 2 3 4 5 6
Sample Output
120 118 110 112 18 19 13 15 16
6 5 3 4 2 1

Solution:

```
static void heapify(int[] array, int i, int n)
{
   int largest=i;
   int left=2*i+1;
   int right=2*i+2;
   if(left<n && array[left]>array[largest])
   {
      largest=left;
   }
   if(right<n && array[right]>array[largest])
   {
      largest=right;
   }
   if(largest!=i)
   {
      int temp=array[largest];
      array[largest]=array[i];
      array[i]=temp;
      heapify(array, largest,n);
   }
}
static void modifyMintoMax(int array[], int n)
{
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    for(int i=(n/2)-1;i>=0;i--)
    {
       heapify(array,i,n);
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Check if a given tree is max-heap or not

A max-heap is a kind of tree which must satisfy the property as “Every node’s value should be greater than its children’s value”.
Your task is given level wise array representation of a binary tree, check if it is a max-heap or not.
Complete the functions **isMaxHeap()** which takes the array and total number of elements n as parameters and return 1 if array represents a max-heap and 0 otherwise.

Input Format

First line of input contain T = number of test cases. Each test case contain two lines, in first line, total number of elements in tree and next line will contain the elements in level order fashion.

Output Format

For each test case, print 1 if array represents max-heap or 0 otherwise in new lines.

Sample Input

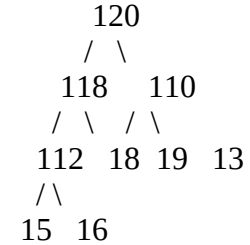
```
2
9
120 118 110 112 18 19 13 15 16
6
1 2 3 4 5 6
```

Sample Output

```
1
0
```

Explanation:

First tree is like below:



It satisfies the properties of max-heap.

Solution:

```
static int isMaxHeap(int array[], int n){
  // Write your code here
    for(int i=0;i<n/2;i++)
    {
        int p=i;
        int left=2*i+1;
        int right=2*i+2;
        if(left<n && array[left]>array[p] || right<n && array[right]>array[p]) return 0;
    }
    return 1;
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Connect the sticks of different lengths with minimum cost

Given n sticks of different length you have to connect them. The cost of connecting sticks is the sum of their length. You have to find the minimum cost for connecting all sticks. For example, if 3 sticks are there having length respectively 3, 6, 1 then we can connect them in many ways like:
Connect 3 and 6 (Cost 9), then we have 2 sticks of length 9 and 1 connect them (cost 10), So total cost is 19.
Connect 3 and 1 (Cost 4), then we have 2 sticks of length 4 and 6 connect them (cost 10), So total cost is 14.
So your function must return 14 in this case.
Complete the function **connectCost()** which takes the array and total number of sticks as input and returns the minimum cost of connecting all these sticks.

Input Format

First line of input contain T = number of test cases.
Each test case contain two lines, in first line, total number of sticks and next line will contain the lengths of each stick.

Output Format

For each test case, print the total cost of connecting all sticks in new lines.

Constraints

1 <= T <= 10
1 <= n <= 10^5
1 <= lengths[i] <= 1000

Sample Input

2
3
3 6 1
4
4 2 3 6

Sample Output

14
29

Solution:

```
class Result
{
  static int connectCost(int lengths[], int n){
    // Write your code here
      PriorityQueue<Integer>pq=new PriorityQueue<>();
      int cost=0;
      for(int i:lengths)
      {
          pq.add(i);
      }
      while(pq.size()>1)
      {
          int a=pq.poll();
          int b=pq.poll();
          cost+=a+b;
          pq.add(a+b);
      }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        return cost;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Implement Priority Queue using Linked List

Implement the priority queue using linked list representation.

Input Format:

First line of input contains an integer Q denoting the number of queries. A query can be of 1 of the below 2 types: -

(a) 1 item p (a query of this type means insert 'item' into the queue with priority p)

(b) 2 (a query of this type means to delete highest priority element from queue and print it, Assume lowest number has highest priority i.e. 0 is highest priority and 9 is lowest).

Next lines of each test case contains Q queries as shown below.

The functions void enqueue() & int dequeue() will take the head of list and modify list and head appropriately. The head pointer is passed by reference to both of them. You are required to complete the methods given.

Output Format:

The output for each test case will be space separated integers having -1 if the queue is empty else the element deleted from the queue.

Sample Input

```
1   // No. of test cases
4   // No. of queries
1   // Query type insert
10  // value
1   // priority
1   // Query type insert
20  // value
0   // priority
2   // Query type delete
2   // Query type delete
```

Sample Output

```
20 10
```

Explanation:

1st query is 1 10 1, which inserts element 10 with priority 1 in queue,
2nd query is 1 20 0, which inserts element 20 with priority 0 in queue,
3rd query is 2, which will delete the highest priority element and print it i.e. 20
4th query is 2, which will delete the highest priority element and print it i.e. 10

Solution:

```
/* class QueueNode
{
  int data;
  int priority;
  QueueNode next;
  public QueueNode(int data, int p)
  {
    this.data = data;
    this.priority = p;
  }
} is provided to you. */
class PQueueLL
{
  public QueueNode front, rear;
    PQueueLL()
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  {
     this.front=null;
     this.rear=null;
  }
 public void EnQueue(int data, int priority)
 {
    QueueNode node=new QueueNode(data,priority);
    if(front==null ||front.priority>priority)
    {
       node.next=front;
       front=node;
    }
    else
    {
       QueueNode temp=front;
       while(temp.next!=null && temp.next.priority<=priority)
       {
          temp=temp.next;
       }
       node.next=temp.next;
       temp.next=node;
    }
 }
 public int DeQueue()
 {
    if(front==null) return -1;
    int val=front.data;
    front=front.next;
    return val;
 }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Sort an array using heap sort

Given an array of N integers, sort them in ascending order using heap sort. Write the two functions **heapify()** and **heapSort(),** to implement the heap.

Input Format

First line contains the number of elements
Second line contains the elements of array separated by space.

Output Format

Print the elements of sorted array in ascending order.

Constraints

1 <= N <= 10^5
-(10^9) <= arr[i] <= 10^9

Sample Input

7
1 3 5 7 2 4 9

Sample Output

1 2 3 4 5 7 9

Solution:

```
void heapify (int array[], int n, int i) {
  // Write your code here
    int largest=i;
    int left=2*i+1;
    int right=2*i+2;
    if(left<n && array[largest]<array[left]) largest=left;
    if(right<n && array[largest]<array[right]) largest=right;
    if(largest!=i)
    {
        int temp=array[largest];
        array[largest]=array[i];
        array[i]=temp;
        heapify(array,n,largest);
    }
}
void heapSort(int array[], int n) {
  // Write your code here
    for(int i=(n/2)-1;i>=0;i--)
    {
        heapify(array,n,i);
    }
    for(int i=n-1;i>=0;i--)
    {
        int temp=array[i];
        array[i]=array[0];
        array[0]=temp;
        heapify(array,i,0);
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Create a binary tree from array

Given an array of integer elements, create a binary tree from this array in level order fashion.
Input Format
The array of elements is given and you have to build the tree from it in level order i.e. elements from left in the array will be filled in the tree level wise starting from level 0.
Do not write the whole program, just complete the function buildTree(int arr[], int n) which takes the array and total number of nodes as parameter and return the root of the binary tree.

Note: Do not read any input from stdin/console. Just complete the functions provided. You can write more functions if required, but just above the given function.
þÿOutput Format
Print the tree in inorder.
Constraints
0 <= n <= 10^5
Sample Input
6 // n
1
2
3
4
5
6
Sample Tree

```
      1
     / \
    2   3
   / \ /
  4 5 6
```

Sample Output
4 2 5 1 6 3

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
// Return the root node of the tree
static Node buildTree(int arr[], int n) {
  Node root = null;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  // Complete the function body
    if(n==0) return root;
    root=new Node(arr[0]);
    Queue<Node>q=new LinkedList<>();
    q.add(root);
    int i=1;
    while(i<n)
    {
        Node temp=q.poll();
        temp.left=new Node(arr[i++]);
        q.add(temp.left);
        if(i<n)
        {
            temp.right=new Node(arr[i++]);
            q.add(temp.right);
        }
    }
   return root;
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Print binary tree with level order traversal

Given a binary tree, print all nodes of the tree in level order traversal. print nodes at same level separated by space and give new line between every level. **(There should be no space after last node of each level)**

Input

The root node of binary tree is given to you, and you have to complete the function void printLevelWise(Node root) to print the nodes at all levels of the binary tree.

Note: Do not read any input from stdin/console. Just complete the function provided. You can write more functions if required, but just above the given function.

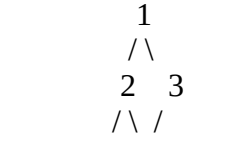
Output

Print the nodes level wise separated by space and new line. There should be no space after last node of each level.

Constraints

0 <= no. of nodes <= 10^5

Sample Input



Sample Output

1
2 3
4 5 6

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
static void printLevelWise(Node root) {
  // Write your code here
    if(root==null) return;
    Queue<Node> q=new LinkedList<>();
    q.add(root);
    q.add(null);
    while(!q.isEmpty())
    {
        Node temp=q.poll();
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        if(temp==null)
        {
            System.out.println();
            if(!q.isEmpty()) q.add(null);
        }
        else
        {
            System.out.print(temp.data);
            if(q.peek()!=null) System.out.print(" ");
            if(temp.left!=null) q.add(temp.left);
            if(temp.right!=null) q.add(temp.right);
        }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Print nodes at odd levels of the binary tree

Given a binary tree, print all nodes at odd levels of the tree. Assume the root node is at level 1 i.e. odd level.
Complete the function **printOdd()** which will take the root node of the tree as parameter and print the nodes at odd levels of the binary tree.

Input Format

First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.

Output Format

Print the nodes at odd levels separated by space.

Constraints

0 <= N <= 10^5

Sample Input

6
1 2 3 4 5 6

Sample Output

1 4 5 6

Explanation:

```
      1          // level-1
     / \
    2   3
   / \ /
  4  5 6         // level-3
```

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
  static void printOdd(Node root) {
    // Write your code here
      if(root==null) return;
      Queue<Node>q=new LinkedList<>();
      q.add(root);
      q.add(null);
      int level=1;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        while(!q.isEmpty())
        {
            Node temp=q.poll();
            if(temp==null)
            {
                level++;
                if(!q.isEmpty()) q.add(null);
            }
            else
            {
                if(level%2!=0) System.out.print(temp.data+" ");
                if(temp.left!=null) q.add(temp.left);
                if(temp.right!=null) q.add(temp.right);
            }
        }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Write iterative version of inorder traversal

Given a binary tree, print it in inorder fashion.

Input Format

The root node of binary tree is given to you, and you have to complete the function void printInorder(Node root) to print the nodes of tree. (The function should use iteration not recursion).

Note: Do not read any input from stdin/console. Just complete the function provided. You can write more functions if required, but just above the given function.

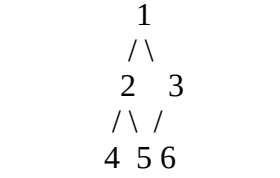
Output Format

Print the nodes of tree separated by space.

Constraints

0 <= N <= 10^5

Sample Input



Sample Output

4 2 5 1 6 3

Solution:

```
/* class Node {
  int data; // data used as key value
  Node leftChild;
  Node rightChild;
  public Node()  {
    data=0;  }
  public Node(int d)  {
    data=d;  }
 } Above class is declared for use. */
static void printInorder(Node root)
{
    if(root==null) return;
    printInorder(root.leftChild);
    System.out.print(root.data+" ");
    printInorder(root.rightChild);
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Complete the inorder(), preorder() and postorder() functions for traversal with recursion

Given a binary tree, print it in inorder, preorder and postorder fashion with recursion.

Input

The root node of binary tree is given to you, and you have to complete the function void inorder(Node root), void preorder(Node root) & void postorder(Node root) to print the nodes of tree. (The function should use recursion).

Note: Do not read any input from stdin/console. Just complete the function provided. You can write more functions if required, but just above the given function.

Output

Print the nodes of tree separated by space by all three traversals in new lines.

Sample Input

6
1 2 3 4 5 6

Above input corresponds to below tree.

```
      1
     / \
    2   3
   / \ /
  4  5 6
```

Sample Output

4 2 5 1 6 3
1 2 4 5 3 6
4 5 2 6 3 1

Solution:

```
/* class Node {
  int data; // data used as key value
  Node leftChild;
  Node rightChild;
  public Node()  {
    data=0;  }
  public Node(int d)  {
    data=d;  }
 } Above class is declared for use. */
static void inorder(Node root)
{
    if(root==null) return;
    inorder(root.leftChild);
    System.out.print(root.data+" ");
    inorder(root.rightChild);
}
static void preorder(Node root)
{
    if(root==null) return;
    System.out.print(root.data+" ");
    preorder(root.leftChild);
    preorder(root.rightChild);
}
static void postorder(Node root)
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
{
   if(root==null) return;
   postorder(root.leftChild);
   postorder(root.rightChild);
   System.out.print(root.data+" ");
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Construct tree from given inorder and post order traversal

Given two array representing the inorder and postorder traversal of a binary tree, construct the tree and return its root node.

Input

Two arrays for traversals and total number of nodes are given and you needs to write a function Node buildTree(int in[], int post[], int N), which accepts the inorder and postorder traversals of a binary tree and the number of nodes in the tree. The function returns the root node of the constructed tree.

Note: Do not read any input from stdin/console. Just complete the function provided. You can write more functions if required, but just above the given function.

Output

The nodes of tree will be printed separated by space using preorder traversal in new lines.

Sample Input

6 // Number of nodes
4 // inorder traversal
2
5
1
6
þÿ3
4 // Postorder traversal
5
2
6
3
1

Sample Output

1 2 4 5 3 6

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node leftChild;
 *    Node rightChild;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
  static Node buildTree(int in[], int post[], int N) {
    // Write your code here
    HashMap<Integer,Integer> hm=new HashMap<>();
      for(int i=0;i<N;i++)
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        {
            hm.put(in[i],i);
        }
        return help(in,0,in.length-1,post,0,post.length-1,hm);
    }
    static Node help(int[]in,int is,int ie,int[]post,int ps,int pe,HashMap<Integer,Integer>hm)
    {
        if(ps>pe || is>ie) return null;
        Node root=new Node(post[pe]);
        int inRoot=hm.get(post[pe]);
        int rem=inRoot-is;
        root.leftChild=help(in,is,inRoot-1,post,ps,ps+rem-1,hm);
        root.rightChild=help(in,inRoot+1,ie,post,ps+rem,pe-1,hm);
        return root;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Count the number of leaf and non-leaf nodes in a binary tree

A leaf in a tree is a node which has no children, i.e. for a binary tree, both its left and right point to NULL. Given a binary tree count the number of leaf and non-leaf nodes in it. Complete the functions **countLeafs()** & **countNonLeafs()** which takes the address of the root node as parameter and return the count of the leaves and non-leafs respectively.

þÿInput Format

First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.

Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.

Output Format

For each test case, print the number of leaf and non-leaf nodes separated by space in new lines.

Constraints

0 <= N <= 10^5

Sample Input

7

1 2 3 4 5 -1 6

Sample Output

3 3

Explanation:

The above tree is:

```
      1
    /  \
   2    3
  / \    \
 4   5    6
```

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
  static int countLeafs(Node root) {
    if(root==null) return 0;
    if(root.left==null && root.right==null) return 1;
    return countLeafs(root.left)+countLeafs(root.right);
  }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  static int countNonLeafs(Node root) {
    if(root==null) return 0;
    if(root.left==null && root.right==null) return 0;
      return 1+countNonLeafs(root.left)+countNonLeafs(root.right);
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Print all paths to leaves and their details of a binary tree

Given a binary tree print all paths from root node to leaf nodes with their respective lengths and total number of paths in it.
The root node of binary tree is given to you. A path from root to leaf in a tree is a sequence of adjacent nodes from root to any leaf node.
Do not write the whole program, just complete the function **void printAllPaths(Node root)** which takes the address of the root as a parameter and print all details as shown below.
Note: If the tree is empty, do not print anything.

Input Format

First line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.

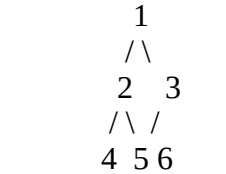
Output

For each test case, print the paths with their length and total paths in new lines.

Constraints

0 <= N <= 10^5
0 <= node data <= 1000

Sample Input



Sample Output

1 2 4 2
1 2 5 2
1 3 6 2
3

Explanation:

First path is from 1 to 4 having 2 edges, so 1 2 4 is path and 2 is length of it.
Similarly for other two leaf nodes. And at last line total number of paths are printed.

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
   static int count=0;
```

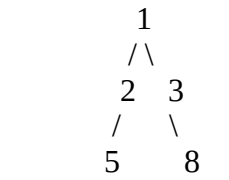

BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  static void help(Node root,ArrayList<Integer>arr)
  {
     arr.add(root.data);
     if(root.left==null && root.right==null)
     {
        print(arr);
     }
     else
     {
        if(root.left!=null) help(root.left,arr);
        if(root.right!=null) help(root.right,arr);
     }
     arr.remove(arr.size()-1);
  }
  static void print(ArrayList<Integer> arr)
  {
     for(int i=0;i<arr.size();i++)
     {
        System.out.print(arr.get(i)+" ");
     }
     System.out.println(arr.size()-1);
     count++;
  }
static void printAllPaths(Node root) {
  // Write your code here
    if(root==null) return;
    ArrayList<Integer> arr=new ArrayList<>();
    help(root,arr);
    System.out.print(count);
}
```

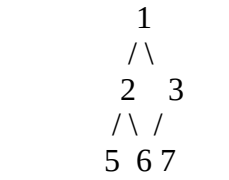

BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find the right node of a given node

Given a binary tree and a key in it, find the node which is on same level and on right of this given key. If no such node present return -1.
Complete the function **findRightSibling()** which takes the address of the root node and an integer key as a parameter and return the right sibling (if exists, otherwise return -1).
þÿInput Format
First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.
Third line contains an integer key whose right sibling is desired.
Output Format
Print the right sibling if any, otherwise print -1.
Sample Input 1
7
1 2 3 5 -1 -1 8
5
Sample Output 1
8
Explanation 1



þÿFor 5, the right sibling is 8.
Sample Input 2
6
1 2 3 5 6 7
7
Sample Output 2
-1
Explanation 2



For 7, there is no node present in its right on the same level.
Therefore, the answer is -1.

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
  static int findRightSibling(Node root, int key) {
    // Write your code here
      if(root==null || root.data==key) return -1;
      Queue<Node> q=new LinkedList<>();
      q.add(root);
      q.add(null);
      while(!q.isEmpty())
      {
          Node temp=q.poll();
          if(temp==null)
          {
              if(!q.isEmpty()) q.add(null);
          }
          else
          {
              if(temp.data==key && q.peek()!=null) return q.peek().data;
              if(temp.left!=null) q.add(temp.left);
              if(temp.right!=null) q.add(temp.right);
          }
      }
      return -1;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Convert a binary tree into its mirror tree

Given a binary tree, convert it in its mirror image. Mirror of a Binary Tree is another Binary Tree in which left and right children of all non-leaf nodes are interchanged.

Input

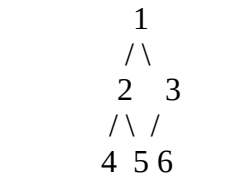
The root node of binary tree is given to you. Do not write the whole program, just complete the function findMirror(Node root) which takes the address of the root as a parameter and change the tree in its mirror image.

Note: Do not read any input from stdin/console. Just complete the function provided. You can write more functions if required, but just above the given function.

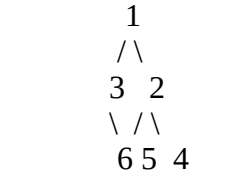
Output

For each test case, print the tree in inorder in new lines.

Sample Input



Sample Output



Inorder: 3 6 1 5 2 4

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
static Node findMirror(Node root) {
  // Write your code here
    if(root==null) return null;
    Node left=findMirror(root.left);
    Node right=findMirror(root.right);
    root.left=right;
    root.right=left;
    return root;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Print cousins of a given node in Binary Tree

Given a binary tree and a key value from this tree, print all the cousins of this node separated by space. If no cousin exists print -1.
Complete the function **printCousins()** which takes the address of the root node and a key k as a parameter and print the cousins of k separated by space or -1 if no cousin exists.

Input Format

First line contains the total number of nodes, second line contains the node labels separated by space. Third line contains an integer key k whose cousins are desired.

Output Format

For each test case, print the cousins of given key separated by space in new lines.

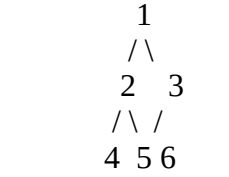
Sample Input

6
1 2 3 4 5 6
2

Sample Output

-1

Explanation:



The parent of node 2 is 1, who have no siblings, no cousins for node 2.
If key is 6, then 2 is the sibling of parent i.e. 3. So the cousins are 4 and 5þÿ.

Solution:

```
import java.util.*;
/* class Node {
  int data; // data used as key value
  Node leftChild;
  Node rightChild;
  public Node()  {
    data=0;  }
  public Node(int d)  {
    data=d;  }
 } Above class is declared for use. */
class Result {
  static void printCousins(Node root, int k) {
    if(root==null ||root.data==k){
          System.out.print("-1");
        return;
    }
      ArrayList<ArrayList<Pair>>arr=new ArrayList<>();
      Queue<Node>q=new LinkedList<>();
      q.add(root);
      q.add(null);
      ArrayList<Pair>row=new ArrayList<>();
      row.add(new Pair(root.data,-1));
      arr.add(new ArrayList<>(row));
      row.clear();
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
int level=0;
int foundLevel=0;
int parent=-1;
boolean found=false;
while(!q.isEmpty())
{
   Node temp=q.poll();
   if(temp==null)
   {
      level++;
      arr.add(new ArrayList<>(row));
      row.clear();
      if(!q.isEmpty()) q.add(null);
   }
   else
   {
       if(temp.data==k)
       {
          foundLevel=level;
          found=true;
       }
      if(temp.leftChild!=null)
      {
         q.add(temp.leftChild);
         row.add(new Pair(temp.leftChild.data,temp.data));
         if(temp.leftChild.data==k) parent=temp.data;
      }
      if(temp.rightChild!=null)
      {
         q.add(temp.rightChild);
         row.add(new Pair(temp.rightChild.data,temp.data));
         if(temp.rightChild.data==k) parent=temp.data;
      }
   }
}
if(!found)
{
   System.out.print("-1");
}
else
{
   boolean print=false;
   for(Pair i:arr.get(foundLevel))
   {
      if(i.parent!=parent)
      {
         System.out.print(i.val+" ");
         print=true;
      }
   }
   if(!print)
   {
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            System.out.print("-1");
            }
        }
    }
}
class Pair
{
    int val;
    int parent;
    Pair(int val,int parent)
    {
        this.val=val;
        this.parent=parent;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Print nodes in a top view of Binary Tree

Given a binary tree, print top view of the binary tree.
For Example:
[image:https://lh4.googleusercontent.com/6ciAz3U8GM6G1FoGxS5R-6xadLRYegCubYTrdNglg5hV55TT4fzTEes9Mkc6MZMV5M0_93uhVw41jc6QqQo8M6ast5iF7Ms_fJIggUTS447inFfYobB20pjcAE_Inw]

Top view of the given binary tree will be: **4 2 1 3 9**
Complete the function **printTopView()** which takes the address of the root as a parameter and print the tree from top view.
Input Format
First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.
Output Format
For each test case, print the tree nodes from top view separated by space in new lines.
Sample Input
7
1 2 3 4 5 -1 6
Sample Output
4 2 1 3 6
Explanation:

```
      1
     / \
    2   3
   / \   \
  4   5   6
```

So, from top, first node is the left most node i.e. 4, and then 2, 1, 3 and lastly 6.
þÿNote that, 5 is not seen from top.

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Pair{
   Node val;
   int d;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    Pair(Node val,int d)
    {
       this.val=val;
       this.d=d;
    }
}
class Result {
  static void printTopView(Node root) {
    // Write your code here
      if(root==null) return;
      TreeMap<Integer,Integer> map=new TreeMap<>();
      Queue<Pair>q=new LinkedList<>();
      q.add(new Pair(root,0));
      while(!q.isEmpty())
      {
         Pair temp=q.poll();
         if(map.get(temp.d)==null) map.put(temp.d,temp.val.data);
         if(temp.val.left!=null) q.add(new Pair(temp.val.left,temp.d-1));
         if(temp.val.right!=null) q.add(new Pair(temp.val.right,temp.d+1));
      }
      for(Map.Entry<Integer,Integer>entry:map.entrySet())
      {
         System.out.print(entry.getValue()+" ");
      }
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find out if the tree can be folded or not

Given a binary tree, find out if it can be folded or not. A tree can be folded if left and right sub-trees of root are structure wise mirror images.
Note: A tree with zero or one node is foldable inherently.
Complete the function **isFoldable()** which takes the address of the root as parameter and return 1 if tree can be folded and 0 otherwise.
Input Format
First line contains an integer T, denoting the number of test cases.
For each test case:
First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.
Output Format
For each test case, print 1 if tree is foldable and 0 otherwise, in new lines.
Sample Input 1

```
    1
   / \
  2   3
```

Sample Output 1
1
Sample Input 2

```
     1
    / \
   2   3
  / \
 4  5
```

Sample Output 2
0

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
   static boolean help(Node left,Node right)
   {
      if(left==null && right==null) return true;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        if(left==null ||right==null) return false;
        return (help(left.left,right.right) &&help(left.right,right.left));
    }
  static int isFoldable(Node root) {
    // Write your code here
      if(root==null) return 1;
      return help(root.left,root.right)?1:0;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Given two trees are identical or not

Given two binary trees, find out both are same or not. Two trees are considered same if they have same nodes and same structure.
So complete the function **areSameTree()** which takes the address of the root nodes of two trees as parameters and return **1** if both are same and **0** otherwise.

Input Format

The first line contains the number of testcases, T.
For each Testcase:
First line contains the number of nodes in first tree and second line contains the node labels in level-wise order.
Third line contains the number of nodes in second tree and fourth line contains the node labels in level-wise order.

Output Format

For each test case, print 1 if both tree are same and 0 otherwise.

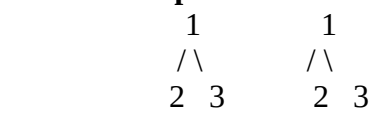
Sample Input 1

1 // Number of testcases
3
1 2 3
3
1 2 3

Sample Output 1

1

Explanation 1



As both the trees have same number of nodes and all same labels so trees are same.

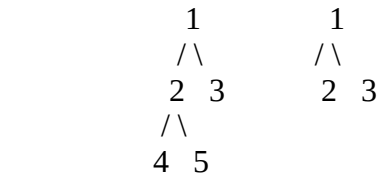
Sample Input 2

1 // Number of testcases
5
1 2 3 4 5
3
1 2 3

Sample Output 2

0

Explanation 2



In this case, trees are not same

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
  /*
   * @Input
   *   t1, t2 -> Given the root node of two binary trees
   *
   * @Output
   *   Return 1 if the two binary trees are identical, else return 0
   */
  static int areSameTree(Node t1, Node t2) {
    // Write your code here
      if(t1==null && t2==null) return 1;
      if(t1==null || t2==null) return 0;
      int left=areSameTree(t1.left,t2.left);
      int right=areSameTree(t1.right,t2.right);
      int ans=0;
      if(t1.data==t2.data) ans=1;
      return (ans==1 && left==1 && right==1)?1:0;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find maximum depth or height of a binary tree

Height of a tree is defined as the length of the longest downward path from root node to any leaf. If tree is empty, height is considered as -1 and for tree with only one node height is considered as 0.
Your task is that given a binary tree, find out the maximum depth of tree (also called height of tree).
Complete the function **treeHeight()** which takes the address of the root node of tree as parameter and return the height of the tree.

Input Format

First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.

Output Format

For each test case, print the height of tree.

Constraints

0 <= N <= 10^5

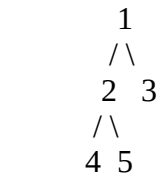
Sample Input

5
1 2 3 4 5

Sample Output

2

Explanation:



The longest path are 1-2-4 and 1-2-5, both have a length of 2, so tree height is 2.

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
  static int treeHeight(Node root) {
    // Write your code here
      if(root==null) return -1;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        int left=treeHeight(root.left);
        int right=treeHeight(root.right);
        return Math.max(left,right)+1;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Evaluation of expression tree

Given a expression binary tree with 4 binary operators (+, -, *, /) and integer operands, evaluate it and print the answer.
Complete the function **evaluateTree()** which takes the address of the root node of tree as parameter and return the result of expression.
Note: The nodes with operators are given as ASCII codes of these operators (e.g. 42 for *(multiply), 43 for +(addition), 45 for -(subtraction) & 47 for /(division)).
Input Format
First line contains the total number of nodes and second line contains the node labels separated by space.
Output Format
For each test case, print the result of expression, in new lines.
Sample Input
7
43 42 43 4 5 2 4
Sample Output
26
Explanation:

```
     +
    / \
   *   +
  / \  / \
 4  5 2  4
```

Nodes which are having child nodes are the operator nodes and leaf nodes are the operand nodes to distinguish between.

Solution:

```
/* class Node {
  int data; // data used as key value
  Node leftChild;
  Node rightChild;
  public Node()  {
    data=0;  }
  public Node(int d)  {
    data=d;  }
 } Above class is declared for use. */
class Result {
  static int evaluateTree(Node t1) {
    if(t1==null) return 0;
      if(t1.leftChild==null && t1.rightChild==null) return t1.data;
      int left=evaluateTree(t1.leftChild);
      int right=evaluateTree(t1.rightChild);
      switch(t1.data)
      {
          case 42:
              return left*right;
          case 43:
              return left+right;
          case 45:
              return left-right;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            case 47:
                return left/right;
            default:
                return 0;
        }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Given binary tree is binary search tree or not

A binary tree is called binary search tree if it holds following three properties: -
a. The left subtree of a node contains nodes whose keys are less than that node's key.
b. The right subtree of a node contains nodes whose keys are greater than that node's key.
c. Both the left and right subtrees must also be binary search trees.
Your task is that, given a binary tree check if it is binary search tree or not. Complete the function **isBinarySearchTree()** which takes the address of the root node of tree as parameter and return **1** if the tree is binary search tree and **0** otherwise.

Input Format

First line contains an integer T, denoting the number of test cases.
For each test case:
First Line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.

Output Format

For each test case, print 0 or 1, in new lines.

Constraints

1 <= T <= 10
0 <= N <= 10^5þÿ

Sample Input

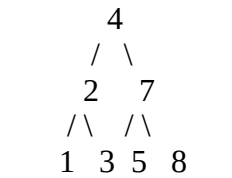
1 // testcases
7 // N
4 2 7 1 3 5 8

Sample Output

1

Explanation:

Given tree is:



Which satisfies the property of binary search tree, so return value is 1.

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
 */
class Result {
    static ArrayList<Integer> inorder(Node root)
    {
        ArrayList<Integer> ans=new ArrayList<>();
        if(root==null) return ans;
        ans.addAll(inorder(root.left));
        ans.add(root.data);
        ans.addAll(inorder(root.right));
        return ans;
    }
  static int isBinarySearchTree(Node root) {
    // Write your code here
      if(root==null) return 1;
      ArrayList<Integer> in=inorder(root);
      for(int i=0;i<in.size()-1;i++)
      {
          if(in.get(i)>in.get(i+1)) return 0;
      }
      return 1;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find the kth smallest element in the binary search tree

Given a binary search tree and a number **k**, print the kth smallest number in tree.

Input

The root node of binary search tree is given to you. Do not write the whole program, just complete the function int kSmallest(Node root, int k) which takes the address of the root node of tree and an integer k as parameters and return the kth smallest number from tree.

Note: Do not read any input from stdin/console. Just complete the function provided. You can write more functions if required, but just above the given function.

Output

Print the kth smallest number from the binary search tree.

Sample Input

```
      4
     / \
    2   7
   / \  / \
  1  3 5  8
```

k = 4
k = 6

Sample Output

4
7

Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
   static ArrayList<Integer> inorder(Node root)
   {
      ArrayList<Integer> in=new ArrayList<>();
      if(root==null) return in;
      in.addAll(inorder(root.left));
      in.add(root.data);
      in.addAll(inorder(root.right));
      return in;
   }
 static int kSmallest(Node root, int k) {
   // Write your code here
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  if(root==null) return 0;
    ArrayList<Integer> in=inorder(root);
    return in.get(k-1);
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Convert Level Order Traversal to BST

Given an array of integer elements representing the level order traversal of a binary search tree, create the binary search tree from this array.
Write the function **buildSearchTree()** which takes the array and total number of nodes as parameters and return the root of the binary search tree.

Input Format

First line contains the number of elements in the array(containing the level order traversal) and second line contains the elements separated by space.

þÿOutput Format

Print the tree with inorder traversal.

Sample Input

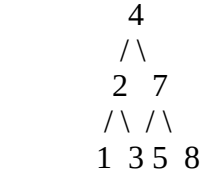
7
4 2 7 1 3 5 8

Sample Output

1 2 3 4 5 7 8

Explanation:

The tree from above level order traversal will be:



Solution:

```
/* class Node {
  int data; // data used as key value
  Node leftChild;
  Node rightChild;
  public Node()  {
    data=0;  }
  public Node(int d)  {
    data=d;  }
 } Above class is declared for use. */
class Result {
    static Node insert(Node root, int k)
    {
        if(root==null)
        {
            root=new Node(k);
        }
        else if(root.data>k)
        {
            root.leftChild=insert(root.leftChild,k);
        }
        else if(root.data<k)
        {
            root.rightChild=insert(root.rightChild,k);
        }
        return root;
    }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  static Node buildSearchTree(int t[], int n) {
    Node root = null;
    // Complete the function body.
      for(int i=0;i<n;i++)
      {
          root=insert(root,t[i]);
      }
    return(root);
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find a lowest common ancestor of a given two nodes in a binary search tree

The Lowest Common Ancestor (LCA) of two nodes in a Binary Search Tree (BST) is defined as the deepest node from the root which lies in the path of both the nodes from the root.
So, given a binary search tree, find the Lowest Common Ancestor of two given nodes in it.
Complete the function **lowestCommonAncestor()** which takes the address of the root node of tree and two integer keys as parameters and return the lowest common ancestor of these two nodes (For empty tree return -1).
You can assume that both the given keys are present in the tree and are different from each other.

Input Format

First line contains an integer N, denoting the number of integers to follow in the serialized representation of the tree.
Second line contains N space separated integers, denoting the level order description of left and right child of nodes, where -1 signifies a NULL child.
Third line contains two integers k1 and k2 separated by space.

Output Format

Print the lowest common ancestor of given two nodes from the binary search tree.

Constraints

0 <= N <= 10^5

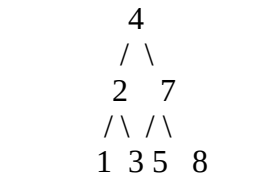
Sample Input

```
7
4 2 7 1 3 5 8
1 5
```

Sample Output

```
4
```

Explanation:



Solution:

```
/*
 *  class Node {
 *    int data;
 *    Node left;
 *    Node right;
 *    public Node() {
 *      data = 0;
 *    }
 *    public Node(int d)  {
 *      data = d;
 *    }
 *  }
 *
 *  The above class defines a tree node.
 */
class Result {
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
  static int lowestCommonAncestor(Node root, int k1, int k2) {
    // Write your code here
    if(root==null) return -1;
      if(root.data>k1 && root.data>k2) return lowestCommonAncestor(root.left,k1,k2);
      if(root.data<k1 && root.data<k2) return lowestCommonAncestor(root.right,k1,k2);
      return root.data;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find the floor and ceil of a key in binary search tree

Given a binary search tree, find the floor and ceil of an key in the tree. They are defined as below:
floor value of k: largest node value which is smaller than or equal to k.
ceil value of k: smallest node value which is greater than or equal to k.
Complete the functions **floorOf()** & **ceilOf()** which takes the address of the root node of tree and an integer key as parameters and return the floor and ceil of that key respectively, and -1 if not found.

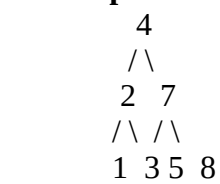
Input Format:
First line contains the number of nodes, second line contains the node labels in level-wise order. Third line contains an integer k whose floor and ceil are desired.

Output Format:
Print the floor and ceil node values of given key from the binary search tree. If not exists, print -1.

Sample Input
7
4 2 7 1 3 5 8
2

Sample Output
2 2

Explanation:



For 2, As 2 is itself present in the tree so 2 will be the floor and ceil of itself.
Furthermore, for 6, the largest node value which is smaller than or equal to 6 is 5, and smallest node value which is greater than or equal to 6 is 7.
And for 9, the largest node value which is smaller than or equal to 9 is 8, and smallest node value which is greater than or equal to 9 is not in the tree so -1 will be printed.

Solution:

```
import java.util.*;
/* class Node {
  int data; // data used as key value
  Node leftChild;
  Node rightChild;
  public Node()  {
    data=0;  }
  public Node(int d)  {
    data=d;  }
 } Above class is declared for use. */
class Result {
    static ArrayList<Integer> inorder(Node root)
    {
        ArrayList<Integer> in=new ArrayList<>();
        if(root==null) return in;
        in.addAll(inorder(root.leftChild));
        in.add(root.data);
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        in.addAll(inorder(root.rightChild));
        return in;
    }
  static int floorOf(Node root, int key) {
    if(root==null) return -1;
      ArrayList<Integer> in=inorder(root);
      int ans=-1;
      for(int i=0;i<in.size();i++)
      {
          if(in.get(i)<=key) ans=in.get(i);
      }
      return ans;
  }
  static int ceilOf(Node root, int key) {
    if(root==null) return -1;
      ArrayList<Integer> in=inorder(root);
      int ans=-1;
      for(int i:in)
      {
          if(i>=key)
          {
              ans=i;
              break;
          }
      }
      return ans;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Fractional Knapsack problem

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
In this problem we will solve a variant of it, in which the items may have a fractional decision i.e. you can pick the full item or partial depending on the capacity you have. Partial items are allowed to fill the bag at fullest. This is also known as fractional knapsack problem.
Given two integer arrays values[0..n-1] and weight[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of values[] such that sum of the weights of this subset is equal to W.

Input Format:

First Line will contain an integer N denoting the number of items.
Second line contains N integers separated by space as values of N items.
Third line contains N integers separated by space as weights of N items.
Fourth line contains an integer denoting the capcacity of knapsack.

Output Format:

Print the maximum value that can be earned with above knapsack.

Constraints:

1 <= N <= 10^5
1 <= val[i] <= 10^3
1 <= weight[i] <= 10^3
1 <= capacity <= 10^7

Sample Input

3
25 24 15
18 15 10
20

Sample Output

31.50

Explanation:

Pick 2nd item in full and 3rd item half which makes total weight as (15 + 10/2) = 20 and profit as (24 + 15/2) = 31.5

Solution:

```
class Pair{
   int val;
   int w;
   Pair(int val,int w)
   {
      this.val=val;
      this.w=w;
   }
}
class Result
{
  static double fractionalKnapsack(int val[], int weight[], int n, int capacity)
  {
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        Pair[] arr=new Pair[n];
        for(int i=0;i<n;i++)
        {
            arr[i]=new Pair(val[i],weight[i]);
        }
        Arrays.sort(arr,(a,b)->Double.compare((double)b.val/b.w,(double)a.val/a.w));
        double ans=0.0;
        for(Pair a:arr)
        {
            int vali=a.val;
            int w=a.w;
            if(capacity-w>=0)
            {
                ans+=vali;
                capacity=capacity-w;
            }
            else
            {
                ans+=((double)vali/w)*capacity;
                break;
            }
        }
        return ans;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Interval scheduling Problem

Interval is defined as a span of time with start and end time. You are given **N** intervals with their start and finish times.
Two intervals can overlap each other if they have any time in common, e.g. Interval(5, 15) and Interval(10,20) are overlapping as time 10 to 15 is shared between them,
whereas Interval(5, 15) and Interval(20, 30) are non-overlapping as they do not have any common time.
Your task is to find the maximum number of intervals that can be scheduled, obviously they need to be non-overlapping.

Input Format:
The 1st line contains an integer N, the number of intervals.
The 2nd line contains N integers separated by space denoting starting times of N intervals.
The 3rd line contains N integers separated by space denoting finishing times of N intervals.

Output Format:
Print the number of non-overlapping intervals that can be scheduled.

Constraints:
1 <= N <= 10^5

Sample Input
4
10 25 12 40
26 32 22 50

Sample Output
3

Explanation:
We can start with the interval no 3 i.e. having start time 12 and finish time 22, then
with the interval no 2 i.e. having start time 25 and finish time 32 and last with the interval no 4 i.e. having start time 40 and finish time 50.

Solution:

```
class Pair{
    int start;
    int end;
    Pair(int start,int end)
    {
        this.start=start;
        this.end=end;
    }
}
class Result {
  static int intervalScheduling(int[] start, int[] end) {
    // Write your code here
      int n=start.length;
      Pair[] pairs=new Pair[n];
      for(int i=0;i<n;i++)
      {
          pairs[i]=new Pair(start[i],end[i]);
      }
      Arrays.sort(pairs,(a,b)->Integer.compare(a.end,b.end));
      int count=0;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        int last=-1;
        for(Pair a:pairs)
        {
            int s=a.start;
            int e=a.end;
            if(s>=last)
            {
                count++;
                last=e;
            }
        }
        return count;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Job Scheduling with deadlines

Mr. Amit have some jobs to be scheduled. Each job must meet the deadline to be counted as completed i.e. scheduling a job after its deadline is of no use.
Each job has some profit associated with it. Mr. Amit wants to schedule the as much jobs as he can and also want to earn the maximum profit.
Note: Each job needs 1 time unit for execution.

Input Format:

The 1st line contains an integer N, the number of jobs.
The 2nd line contains N integers separated by space denoting deadline times of N jobs.
The 3rd line contains N integers separated by space denoting profits of N jobs.

Output Format:

Print the maximum profit that can be earned.

Constraints:

1 <= N <= 10^5
1 <= Deadline <= 100
1 <= Profit <= 500

Sample Input

4
2 1 1 2
6 8 5 10

Sample Output

18

Explanation:

Job2 can be scheduled to execute at time 1 with profit 8.
Job4 can be scheduled to execþÿute at time 2 with profit 10.
So maximum profit that can be earned is 18.

Solution:

```
class Pair{
    int p;
    int d;
    Pair(int p,int d)
    {
        this.p=p;
        this.d=d;
    }
}
class Result {
  static int jobScheduling(int[] deadlines, int[] profits) {
    // Write your code here
      int n=deadlines.length;
      Pair[] pairs=new Pair[n];
      for(int i=0;i<n;i++)
      {
          pairs[i]=new Pair(profits[i],deadlines[i]);
      }
      Arrays.sort(pairs,(a,b)->Integer.compare(b.p,a.p));
      int maxD=-1;
      for(int i:deadlines)
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        {
            maxD=Math.max(i,maxD);
        }
        int[] time=new int[maxD+1];
        Arrays.fill(time,-1);
        int total=0;
        for(Pair a:pairs)
        {
            int p=a.p;
            int d=a.d;
            for(int i=d;i>0;i--)
            {
                if(time[i]==-1)
                {
                    time[i]=p;
                    total+=p;
                    break;
                }
            }
        }
        return total;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Activity Selection Problem

Activity is defined as a interval of time for its execution. Every activity has a start time and a finish time. You are given **N** activities with their start and finish times.
Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.
Note: Remember that you should take care of overlapping of activity times, and always choose the very next activity instead of jump if that does not make any difference to the maximum number of activities.

Input Format:
The 1st line contains an integer N, the number of activities.
The 2nd line contains N integers separated by space denoting starting times of N activities.
The 3rd line contains N integers separated by space denoting finishing times of N activities.

Output Format:
Print the starting time of picked activities in sorted order separated by space.

Constraints:
1 <= N <= 10^5

Sample Input
3 // N
20 22 30 // start[]
30 35 40 // finish[]

Sample Output
20 30

Explanation:
First activity will finish at time 30, thereafter we can not select the 2nd activity.
However we can pick the 3rd activity as it will start after 1st will over.
So in this way we can select a maximum of 2 activities and their starting time are printed.

Solution:

```
class Pair{
    int start;
    int end;
    Pair(int start,int end)
    {
        this.start=start;
        this.end=end;
    }
}
class Result {
  // Print the starting time of selected activities in sorted order separated by space
  static void activitySelection(int[] start, int[] end) {
    // Write your code here
      int n=start.length;
      Pair[] pairs=new Pair[n];
      for(int i=0;i<n;i++)
      {
          pairs[i]=new Pair(start[i],end[i]);
      }
      Arrays.sort(pairs,(a,b)->Integer.compare(a.end,b.end));
      int last=-1;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        for(Pair a:pairs)
        {
            int s=a.start;
            int e=a.end;
            if(s>=last)
            {
                System.out.print(s+" ");
                last=e;
            }
        }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Count number of ways to cover a distance

Mr. Amit has a puzzle to solve. He needs your help. The puzzle is to find total number of ways to cover a distance of D steps where he can take on only 1,2,....,K steps in one go. For example, if D=3 & K=3, he can do it in 4 ways: {1+1+1, 1+2, 2+1, 3}, total 4 ways to complete 3 steps.

Input Format

First Line will contain an integer D denoting the distance to be covered.
Second Line will contain an integer K denoting the steps can be taken at most in one go.

Output Format

Print the total number of ways.

Sample Input

3
3

Sample Output

4

Solution:

```
class Result
{
  static int totalWaysToDistance(int d, int k){
    // Write your code here
      if(d==1 || d==0) return 1;
      int[] dp=new int[d+1];
      dp[0]=1;
      dp[1]=1;
      for(int i=2;i<=d;i++)
      {
          int count=0;
          for(int j=1;j<=k&&j<=i;j++)
          {
              count+=dp[i-j];
          }
          dp[i]=count;
      }
      return dp[d];
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Minimum Cost Path to last element of matrix

Given a cost matrix cost[M][N], write a function that returns cost of minimum cost path to reach (M, N) from (0, 0). Each cell of the matrix represents a cost to traverse through that cell. Total cost of a path to reach (M, N) is sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j), cells (i+1, j), (i, j+1) and (i+1, j+1) can be traversed. You may assume that all costs are positive integers.

Input Format

First Line will contain two integers M and N denoting the size of matrix.
Second line contains M*N integers as matrix elements row-wise separated by space.

Output Format

Print the minimum cost to reach from source to destination in matrix.

Sample Input

3 3
1 2 3 4 8 2 1 5 3

Sample Output

8

Explanation

We can traverse from elements (0,0) -> (0,1) -> (1,2) -> (2,2)

Solution:

```
class Result
{
  static int minCostPath(int cost[][], int m, int n){
    // Write your code here
      int[][]dp=new int[m][n];
      dp[0][0]=cost[0][0];
      for(int i=1;i<m;i++)
      {
         dp[i][0]=cost[i][0]+dp[i-1][0];
      }
      for(int j=1;j<n;j++)
      {
         dp[0][j]=cost[0][j]+dp[0][j-1];
      }
      for(int i=1;i<m;i++)
      {
         for(int j=1;j<n;j++)
         {
            dp[i][j]=cost[i][j]+Math.min(dp[i][j-1],Math.min(dp[i-1][j],dp[i-1][j-1]));
         }
      }
      return dp[m-1][n-1];
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Longest Common Subsequence (LCS)

Overlapping Subproblems and Optimal Substructure properties are the prerequisites for solving a problem with dynamic programming. In this problem you need to figure them out. LCS is a classic computer science problem, the basis of diff (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.
A subsequence is defined as an ordered subset of the array's elements having the same sequential ordering as the original array.
Given two sequences, find the length of longest subsequence present in both of them. For example,
For sequences "**ABAZDC**" and "**BACBAD**" the LCS is "**ABAD**" of length 4.
For sequences "**AGGTAB**" and "**GXTXAYB**" the LCS is "**GTAB**" of length 4.
Note: A string of length n has 2^n-1 different possible subsequences, which implies that the time complexity of the brute force approach will be O(n * 2^n). It takes O(n) time to check if a subsequence is common to both the strings. This time complexity can be improved using dynamic programming.
Input Format
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow.
Each test case contains two strings in two lines.
Output Format
For each test case, print the length of the Longest Common Subsequence in new lines.
Sample Input
2
ABAZDC
BACBAD
AGGTAB
GXTXAYB
Sample Output
4
4

Solution:

```
class Result
{
  static int longestCommonSubsequence(String str1, String str2){
    // Write your code here
      int m=str1.length();
      int n=str2.length();
      int[][] dp=new int[m+1][n+1];
      for(int i=1;i<=m;i++)
      {
          for(int j=1;j<=n;j++)
          {
              if(str1.charAt(i-1)==str2.charAt(j-1)) dp[i][j]=1+dp[i-1][j-1];
              else dp[i][j]=Math.max(dp[i-1][j],dp[i][j-1]);
          }
      }
      return dp[m][n];
  }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: The Subset Sum problem

The subset sum problem is an important problem in computer science.
The challenge is to determine if there is some subset of numbers in an given array that can sum up to some given number S.
For example, if the array is {1, 4, 5, 10, 16} and sum = 9, you should return 1 and if sum = 7, you should return 0.

Input Format

First Line will contain an integer M denoting the sum.
Second Line will contain an integer N denoting the total number of elements.
Third line contains N integers separated by space.

Output Format

Print 1 if some subset found and 0 otherwise.

Sample Input

9
5
1 4 5 10 16

Sample Output

1

Solution:

```
class Result
{
   static boolean sub(int a[],int n,int sum)
   {
      if(sum==0) return true;
      if(n==0) return false;
      if(a[n-1]>sum)
      {
         return sub(a,n-1,sum);
      }
      return sub(a,n-1,sum)||sub(a,n-1,sum-a[n-1]);
   }
  static int subsetSum(int a[], int n, int sum){
    // Write your code here
     return sub(a,n,sum)?1:0;
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Matrix Chain Multiplication problem

A matrix can be represented as a 2-dimensional array. Two matrices are eligible for multiplication if the number of columns in 1st matrix is equal to the number of rows in 2nd matrix.
The total number of operations required for multiplying two matrices A [m x n] and B [n x o] will be (m * n * o).
We have many ways to multiply a chain of matrices because matrix multiplication is associative, each way require different number of operations to complete. Mr. Amit has a sequence of matrices and interested to find the way in which minimum operations are required to multiply all matrices. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.
Example :
If we had four matrices A, B, C, and D, we would have:
(ABC)D or (AB)(CD) or A(BCD) and we can use the same kind of parenthesizes for inner problems to get ((A(BC))D) or (((AB)C)D) or (AB)(CD) or (A((BC)D)) or (A(B(CD))).
Suppose A is a 10 × 20 matrix, B is a 20 × 30 matrix, and C is a 30 × 40 matrix. Then,
For 3 matrices we have 2 options
(A(BC)) = To Multiply BC we need 20*30*40=24000 and we get 20x40 matrix, then it is multiplied with A and take 10*20*40=8000 operations, so total 32000 operations needed.
((AB)C) = To Multiply AB we need 10*20*30=6000 and we get 10x30 matrix, then it is multiplied with C and take 10*30*40=12000 operations, so total 18000 operations needed.
Input Format:
First Line will contain an integer N denoting the number of matrices.
Second line contains N+1 integers denoting the matrix sizes separated by space.
Output Format:
Print the minimum operations needed for matrix multiplication as specified above.
Constraints
2 <= N <= 100
1 <= arr[i] <= 500
Sample Input
3
10 20 30 40
Sample Output
18000

Solution:

```
class Result
{
  // Matrix A[i] has dimension p[i-1] x p[i]  for i = 1..n
  static int matrixChainMultiplication(int p[], int n){
    // Write your code here
      if(p[0]==10)
      {
         if(n==4) return 30000;
         if(n==3) return 18000;
      }
      if(p[0]==1)
      {
         if(n==2) return 6;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            if(n==4) return 30;
        }
        if(p[0]==30) return 9375;
        return 26000;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: 0-1 Knapsack problem

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
In this problem we will solve a variant of it, in which the items can have a binary decision only i.e. either you can pick the item or leave it. No partial items allowed to fill the bag at fullest. This is also known as 0-1 knapsack problem.
Given two integer arrays values[0..n-1] and weight[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of values[] such that sum of the weights of this subset is smaller than or equal to W.
Note: You cannot break an item, either pick the complete item, or don't pick it (0-1 property).
Input Format
First Line will contain an integer N denoting the number of items.
Second line contains N integers separated by space as values of N items.
Third line contains N integers separated by space as weights of N items.
Fourth line contains an integer denoting the capcacity of knapsack.
Output Format
Print the maximum value that can be earned with above knapsack.
Constraints
1 <= N <= 10^3
1 <= val[i] <= 10^3
1 <= weight[i] <= 10^3
1 <= capacity <= 10^3
Sample Input
3
60 100 120
10 20 30
50
Sample Output
220
Explanation
Pick 2nd and 3rd item with weight 20 and 30 and profit 100 and 120.

Solution:

```
class Result
{
  static int zeroOneKnapsack(int val[], int weight[], int n, int capacity){
    // Write your code here
      int[][] dp=new int[n+1][capacity+1];
      for(int i=1;i<=n;i++)
      {
         for(int j=1;j<=capacity;j++)
         {
            if(weight[i-1]>j)
            {
               dp[i][j]=dp[i-1][j];
            }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            else
            {
               dp[i][j]=Math.max(dp[i-1][j],val[i-1]+dp[i-1][j-weight[i-1]]);
            }
         }
      }
      return dp[n][capacity];
   }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Tom And Permutations

Tom loves to generate new strings by using various different methods. So, this time he decided to generate new strings by just rearranging the characters of some given string. In other words, he decided to find all the permutations for a given string and print them.
You task is to help Tom write a function to find all the permutations of a given string, that contains distinct characters.

Input Format:
First lines contains a string

Output Format:
Print all permutations of given string in lexicographical order

Sample Input
ABC

Sample Output
ABC
ACB
BAC
BCA
CAB
CBA

Solution:

```
class Solve {
  /*
   * Return all the permutations of the given string in any order,
   * as they will be sorted at the back-end before printing.
   */
  ArrayList<String> permute(String str) {
    // Write your code here
      ArrayList<String>ans=new ArrayList<>();
      help(str,"",ans);
      return ans;
  }
    void help(String str,String add,ArrayList<String>ans)
    {
        if(str.length()==0)
        {
            ans.add(add);
        }
        for(int i=0;i<str.length();i++)
        {
            char ch=str.charAt(i);
            String sub=str.substring(0,i)+str.substring(i+1);
            help(sub,add+ch,ans);
        }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Print all strings of n-bits

Given n-bits we can generate 2^n different strings from it. For example, with 3 bits we can generate 2^3=8 strings i.e. 000, 001, 010, 011, 100, 101, 110, 111.
You need to write a function to find all strings for a given number of bits.

Input Format:
First lines contains an integer denoting the number of bits

Output Format:
Print all strings for given number of bits in ascending order.

Sample Input
3

Sample Output
000
001
010
011
100
101
110
111

Solution:

```
class Solve
{
    // The first argument is the number of bits. You need to save all binary strings in the ArrayList passed as 4th argument named strs.
   // Dont print the strings as they will be printed after needed processing (sorting in ascending order) at back end.
    // i is initially passed as 0, currStr is a char array to store the current String.
   void generateAllStrings(int n, int i, char currStr[], ArrayList<String> strs){
    // Write your code here
      if(i==n)
      {
         strs.add(new String(currStr));
         return;
      }
      currStr[i]='0';
      generateAllStrings(n,i+1,currStr,strs);
      currStr[i]='1';
      generateAllStrings(n,i+1,currStr,strs);
   }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Solve N Queen problem

The N Queen problem is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.
You must know that a queen in chess can attack in all 8 directions.
Given the size of board as N*N, you need to count how many ways are there to place N queens on the board and save all such solutions in the given array

Input Format:
First lines contains an integer denoting the number of queens

Output Format:
Print all solutions in there column number for all queens.
If no solution exists, print -1.

Sample Input
4

Sample Output
1 3 0 2
2 0 3 1

Explanation:
Solutions are for per row starting from row 0 to row N-1.
1 3 0 2 means For 0th row pick column1, for 1st row pick column 3, for 2nd row pick column 0 and for 3rd row pick column 2.

There are total two solutions:
First is 1 3 0 2, means 1st queen at column 1, 2nd queen at column 3, 3rd queen at column 0 and 4th queen at column 2
Second is 2 0 3 1, means 1st queen at column 2, 2nd queen at column 0, 3rd queen at column 3 and 4th queen at column 1

Solution:

```
class Result
{
// Complete this function to check placing queen at board[row][col] is safe or not by checking current row, left diagonal & right diagonal.
  int isSafe(int board[][], int row, int col, int N)
  {
    for(int i=0;i<col;i++)
    {
       if(board[row][i]==1) return 0;
    }
    for(int i=row,j=col;i>=0&&j>=0;i--,j--)
    {
       if(board[i][j]==1) return 0;
    }
    for(int i=row,j=col;i<N&&j>=0;i++,j--)
    {
       if(board[i][j]==1) return 0;
    }
      return 1;
  }
// Complete this function to solve the problem and save the answers in sol ArrayList as required.
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
 boolean solveNQUtil(int board[][], int col, int N, ArrayList<ArrayList<Integer> > sol)
 {
  if(col>=N)
  {
     ArrayList<Integer> ans=new ArrayList<>();
     for(int i=0;i<N;i++)
     {
        for(int j=0;j<N;j++)
        {
           if(board[i][j]==1) ans.add(j);
        }
     }
     sol.add(ans);
     return true;
  }
  boolean res=false;
    for(int i=0;i<N;i++)
    {
       if(isSafe(board,i,col,N)==1)
       {
          board[i][col]=1;
          res=solveNQUtil(board,col+1,N,sol)||res;
          board[i][col]=0;
       }
    }
    return res;
 }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Rat in a Maze Problem

A Maze is given as N*N binary matrix of blocks. A rat starts from **source** and has to reach the **destination**.
The rat can move only in two directions, i.e. **rightwards** and **downwards**.
In the maze matrix, -1 means the block is a dead end and 0 means the block can be used in the path from **source** to **destination**.
Note: Source block is the upper left most block i.e., maze[0][0] and **destination** block is lower rightmost block i.e., maze[N-1][N-1]
Your task is to find the number of possible solutions for the rat to reach the destination.
Input Format
First line contains number of rows i.e. N
Next N lines contains N binary numbers each denoting the maze
Constraints
2 <= N <= 15
Output Format
Print the number of solutions
Sample Input
4
0 -1 -1 -1
0 0 -1 -1
-1 0 0 -1
-1 0 0 0
Sample Output
2
Explanation:
Solution 1: From cell (0,0) -> down -> (1,0) -> forward -> (1,1) -> down -> (2,1) -> down -> (3,1) -> forward -> (3,2) -> forward -> (3,3)
Solution 2: From cell (0,0) -> down -> (1,0) -> forward -> (1,1) -> down -> (2,1) -> forward -> (2,2) -> down -> (3,2) -> forward -> (3,3)
Total 2 solutions possible for this maze.

Solution:

```
class Result {
  public static int solveMaze(int maze[][], int size) {
    // Write your code here
    return help(maze,0,0,size);
  }
    public static int help(int[][] maze,int i, int j, int size)
    {
        if(i==size-1 && j==size-1) return 1;
        if(i>size || j>size || maze[i][j]==-1) return 0;
        int right=help(maze,i,j+1,size);
        int down=help(maze,i+1,j,size);
        return right+down;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find the minimum number of edges in a path of a graph

Consider a directed graph whose vertices are numbered from 1 to n. There is an edge from a vertex i to a vertex j iff either j = i + 1 or j = 3i. The task is to find the minimum number of edges in a path in G from vertex 1 to vertex n.

Input Format:

The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows.

Each test case contains a single line of input which contains an integer, n.

Output Format:

Print the number of edges in the shortest path from 1 to n.

Sample Input

1

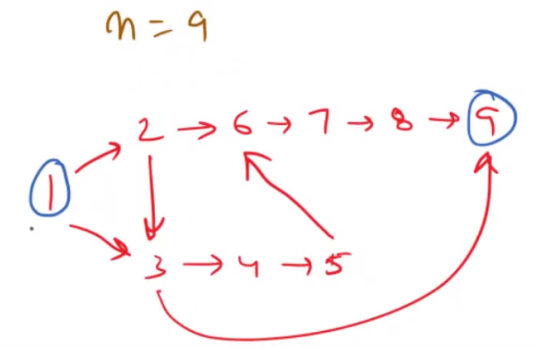
9

Sample Output

2

Explanation:

The below given graph is formed



The minimum number of edges from vertex 1 to vertex 9 is 2.

Solution:

```
class Result{
   static int path(int source,int dest,ArrayList<ArrayList<Integer>> adj)
   {
      boolean[] visited=new boolean[dest+1];
      int[] dis=new int[dest+1];
      Queue<Integer>q=new LinkedList<>();
      q.add(source);
      visited[source]=true;
      dis[source]=0;
      while(!q.isEmpty())
      {
         int a=q.poll();
         for(int i:adj.get(a))
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            {
                if(!visited[i])
                {
                    q.add(i);
                    visited[i]=true;
                    dis[i]=dis[a]+1;
                    if(i==dest) return dis[dest];
                }
            }
        }
        return 0;
    }
  static int number_of_edges(int n){
    // Write your code here
      ArrayList<ArrayList<Integer>> adj=new ArrayList<>();
      for(int i=0;i<=n;i++)
      {
          adj.add(new ArrayList<>());
          if(i+1<=n) adj.get(i).add(i+1);
          if(3*i<=n) adj.get(i).add(3*i);
      }
      return path(1,n,adj);
  }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find path in a directed graph

Given a **directed graph** and two vertices(say source and destination vertex), check whether there exists some path from the given source vertex to the destination vertex or not. If it exists then print “YES”, else print “NO”.

Input Format:

First line contains two space separated integers V, E denoting the number of vertices and edges in the graph.

Following E lines contain two space separated integers u, v denoting a directed edge from u to v.

Last line contains two space separated integers src, dest denoting the source and destination vertex respectively.

Constraints:

1 <= V <= 100

0 <= E < 10000

0 <= u, v, src, dest < 100

Output Format:

Print “YES” if a path exists from the given source vertex to the destination vertex, else print “NO”.

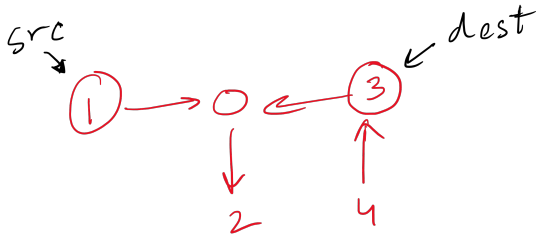
Sample Input 1

```
5 4  // V E
1 0  // directed edge from u to v
0 2
3 0
4 3
1 3  // source destination
```

Sample Output 1

NO

Explanation 1



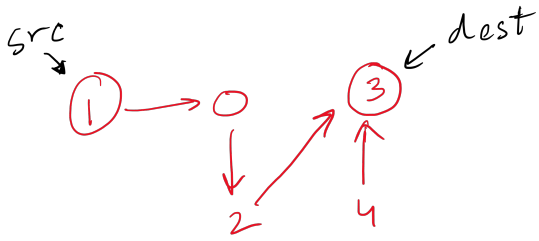
No path exists, from the given source vertex to the destination vertex

Sample Input 2

```
5 4  // V E
1 0
0 2
2 3
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

4 3
1 3
Sample Output 2
YES
Explanation 2



There is a path, from the given source vertex to the destination vertex

Solution:

```
import java.util.*;
// Other imports go here
// Do NOT change the class name
class Main{
   public static boolean path(int source, int dest,ArrayList<ArrayList<Integer>>adj )
   {
      boolean[] visited=new boolean[adj.size()];
      Queue<Integer> q=new LinkedList<>();
      q.add(source);
      visited[source]=true;
      while(!q.isEmpty())
      {
         int a=q.poll();
         for(int i:adj.get(a))
         {
            if(!visited[i])
            {
               visited[i]=true;
               q.add(i);
               if(i==dest) return true;
            }
         }
      }
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        return false;
    }
    public static void main(String[] args)
    {
        // Write your code here
        Scanner sc=new Scanner(System.in);
        int v=sc.nextInt();
        int e=sc.nextInt();
        ArrayList<ArrayList<Integer>>adj=new ArrayList<>();
        for(int i=0;i<v;i++)
        {
            adj.add(new ArrayList<>());
        }
        for(int i=0;i<e;i++)
        {
            int e1=sc.nextInt();
            int e2=sc.nextInt();
            adj.get(e1).add(e2);
        }
        int source=sc.nextInt();
        int dest=sc.nextInt();
        if(path(source,dest,adj)||source==dest) System.out.print("YES");
        else System.out.print("NO");
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Number of Islands

Given a **m** x **n** binary matrix in which 1 represents 'land' and 0 represents 'water', find the number of islands.
An island is surrounded by water(**0s**) and is formed by connecting adjacent lands(**1s**) horizontally or vertically. You may assume all four edges of the given matrix are all surrounded by water.

Input Format:
First line contains two space separated integers m, n that denotes the number of rows and columns in the matrix respectively.
Each of the next m lines contain n space-separated integers representing the matrix.

Constraints:
1 <= m, n <= 150

Output Format:
Print the number of islands.

Sample Input
4 4
1 0 1 0
1 1 0 0
0 0 0 1
1 1 1 1

Sample Output
3

Explanation:
First island consists of following cells : (0, 0), (1, 0), (1,1)
Second island consists of following cells : (0, 2)
Third island consists of following cells : (2, 3), (3, 0), (3, 1), (3, 2), (3, 3)

Solution:

```
class Result {
  static void dfs(int i,int j,int[][] mat,boolean[][] visited)
  {
     visited[i][j]=true;
     int m=mat.length;
     int n=mat[0].length;
     int[] dr={-1,0,1,0};
     int[] dc={0,-1,0,1};
     for(int k=0;k<4;k++)
     {
        int nrow=i+dr[k];
        int ncol=j+dc[k];
        if(nrow>=0 && nrow<m && ncol>=0 && ncol<n && mat[nrow][ncol]==1 &&!
visited[nrow][ncol])
        {
           dfs(nrow,ncol,mat,visited);
        }
     }
  }
 static int countIslands(int mat[][], int m, int n){
   // Write your code here
     boolean[][] visited=new boolean[m][n];
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        int count=0;
        for(int i=0;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(mat[i][j]==1 &&!visited[i][j])
                {
                    dfs(i,j,mat,visited);
                    count++;
                }
            }
        }
        return count;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Shortest path in a binary maze

Given a **m** x **n** binary matrix, find the length of the shortest path in the matrix from a given source cell to a destination cell. The path can only be created out of a cell if its value is 1. If there is no path from a given source cell to a destination cell, return -1.
Note: At any point of time, you can either move horizontally or vertically in the four directions.
Input Format:
First line contains two space separated integers m, n that denotes the number of rows and columns in the matrix respectively.
Each of the next m lines contain n space-separated integers representing the matrix.
Second Last line contains two space separated integers, denoting the position of the source cell.
Last line contains two space separated integers, denoting the position of the destination cell.
Constraints:
1 <= m, n <= 150
Output Format:
Print the length of the shortest path from source to destination.
Sample Input
5 5
0 1 1 1 1
1 1 0 0 1
1 0 0 1 1
1 1 1 1 0
0 1 0 1 0
1 1
3 2
Sample Output
5
Explanation:
The shortest path from (1, 1) to (3, 2) is as follows:
(1, 1) -> (1, 0) -> (2, 0) -> (3, 0) -> (3, 1) -> (3, 2)

Solution:

```
class Pair{
   int row;
   int col;
   Pair(int row,int col)
   {
      this.row=row;
      this.col=col;
   }
}
class Result {
   static int shortestPath(int mat[][], int srcR, int srcC, int destR, int destC, int m, int n){
     // Write your code here
      if(mat[srcR][srcC]==0 || mat[destR][destC]==0) return -1;
      boolean[][] visited=new boolean[m][n];
      int[][] dist=new int[m][n];
      visited[srcR][srcC]=true;
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        dist[srcR][srcC]=0;
        Queue<Pair>q=new LinkedList<>();
        q.add(new Pair(srcR,srcC));
        int[] dr={-1,0,1,0};
        int[] dc={0,-1,0,1};
        while(!q.isEmpty())
        {
            Pair a=q.poll();
            int row=a.row;
            int col=a.col;
            for(int i=0;i<4;i++)
            {
                int nrow=row+dr[i];
                int ncol=col+dc[i];
                if(nrow>=0 && nrow<m && ncol>=0 && ncol<n && mat[nrow][ncol]==1 &&!
visited[nrow][ncol])
                {
                    visited[nrow][ncol]=true;
                    dist[nrow][ncol]=dist[row][col]+1;
                    q.add(new Pair(nrow,ncol));
                    if(nrow==destR && ncol==destC) return dist[destR][destC];
                }
            }
        }
        return -1;
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Depth First Traversal of Graph

Graphs can be traversed popularly in following two ways: Depth First Traversal (DFT) and Breadth First Traversal (BFT). Traversing Graphs can be little bit more tricky than traversing tree because of possibilities of a cycle in graph. So, you need to to write the function for doing a depth first traversal over the given graph.

Input Format:

First line contains an integer V i.e. number of nodes.
Second line contains an integer E i.e. number of edges.
Next E lines contains two integers each v1, v2 denoting the edge from v1 to v2.
Last line of input contains an integer denoting the starting vertex for DFT.

Output Format:

Print the vertex labels starting from given vertex separated by space.

Sample Input

```
4
5
0 1
0 2
1 2
2 0
2 3
2
```

Sample Output

```
2 0 1 3
```

Solution:

```
/* The class is defined with below variables
  class Graph
{
  private Map<Integer, List<Integer>> adjVertices;
  public Graph() {
    this.adjVertices = new HashMap<Integer, List<Integer>>();
  }
  public void addVertex(int vertex) {
    adjVertices.putIfAbsent(vertex, new ArrayList<>());
  }
  public void addEdge(int src, int dest) {
    adjVertices.get(src).add(dest);
  }       */
void DFSUtil(int v, boolean visited[])
{
    visited[v]=true;
    System.out.print(v+" ");
    for(int i:adjVertices.get(v))
    {
       if(!visited[i]) DFSUtil(i,visited);
    }
}
void DFS(int v)
{
    boolean[] visited=new boolean[adjVertices.size()];
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
    DFSUtil(v,visited);
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Breadth First Traversal of Graph

Graphs can be traversed popularly in following two ways: Depth First Traversal (DFT) and Breadth First Traversal (BFT). Traversing Graphs can be little bit more tricky than traversing tree because of possibilities of a cycle in graph. So, you need to to write the function for doing a breadth first traversal over the given graph.

Input Format:

First line contains an integer V i.e. number of nodes.
Second line contains an integer E i.e. number of edges.
Next E lines contains two integers each v1, v2 denoting the edge from v1 to v2.
Last line of input contains an integer denoting the starting vertex for BFT.

Output Format:

Print the vertex labels starting from given vertex separated by space.

Sample Input

```
4
5
0 1
0 2
1 2
2 0
2 3
2
```

Sample Output

```
2 0 3 1
```

Solution:

```
/* The class is defined with below variables
class Graph {
  private int V;
  private Map<Integer, List<Integer>> adjVertices;
  public Graph(int V) {
      this.V = V;
    this.adjVertices = new HashMap<Integer, List<Integer>>();
  }
  public void addVertex(int vertex) {
    adjVertices.putIfAbsent(vertex, new ArrayList<>());
  }
  public void addEdge(int src, int dest) {
    adjVertices.get(src).add(dest);
  }       */
void BFS(int v)
{
    boolean[] visited=new boolean[adjVertices.size()];
    Queue<Integer>q=new LinkedList<>();
    q.add(v);
    visited[v]=true;
    while(!q.isEmpty())
    {
        int node=q.poll();
        System.out.print(node+" ");
        for(int i:adjVertices.get(node))
```

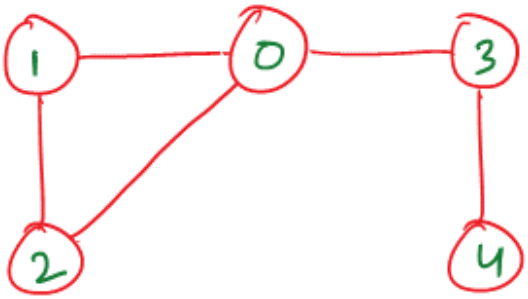

BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
        {
            if(!visited[i]){
                visited[i]=true;
                q.add(i);
            }
        }
    }
}
```

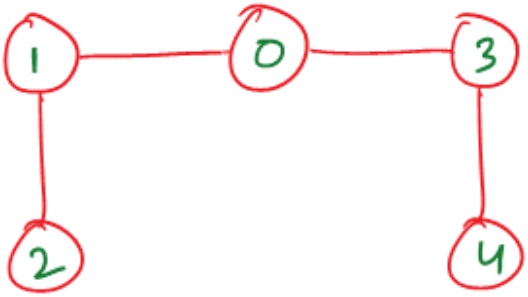

BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

Aim: Find the cycle in undirected graph

Given a un-directed graph find out if there is a cycle in it or not.
Note: The given graph **may** or **may not** be connected.
Example 1: The following graph contains a cycle



Example 2: The following graph does **not** contain a cycle



Input Format
The first line contains the Number of vertices, V.
The second line contains the Number of edges, E.
Then E lines follow , containing 2 numbers,x & y, seperated by space denoting an edge from vertex x to y.
Output Format
Print Yes or No depending on graph has cycle or not.
Sample Input

```
5  // Vertices
5  // Edges
0 1  // Edge from v1 to v2
0 2
1 2
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

0 3
3 4

Sample Output

Yes

Explanation:

The graph is having a cycle from vertices 0, 1 and 2

Solution:

```
import java.util.*;
// Add other imports if you want to
// Do NOT change the class name
class Main{
   public static boolean dfs(int node,int parent,boolean[]
visited,ArrayList<ArrayList<Integer>>adj)
   {
      visited[node]=true;
      for(int i:adj.get(node))
      {
         if(!visited[i])
         {
           if( dfs(i,node,visited,adj)) return true;
         }
         else{
            if(i!=parent) return true;
         }
      }
      return false;
   }
   public static void main(String[] args)
   {
      // Write your code here
      Scanner sc=new Scanner(System.in);
      int v=sc.nextInt();
      int e=sc.nextInt();
      ArrayList<ArrayList<Integer>>adj=new ArrayList<>();
      for(int i=0;i<v;i++)
      {
         adj.add(new ArrayList<>());
      }
      for(int i=0;i<e;i++)
      {
         int e1=sc.nextInt();
         int e2=sc.nextInt();
         adj.get(e1).add(e2);
         adj.get(e2).add(e1);
      }
      boolean[] visited=new boolean[v];
      boolean cycle=false;
      for(int i=0;i<v;i++)
      {
         if(!visited[i])
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in

```
            {
                if(dfs(i,-1,visited,adj))
                {
                    cycle=true;
                    break;
                }
            }
        }
        if(cycle)
        {
            System.out.print("Yes");
        }
        else
        {
            System.out.print("No");
        }
    }
}
```


BHUVESH MITTAL
bhuvesh1450.be22@chitkara.edu.in