CHESSBOARD PROBLEM

Take as input N, a number. N represents the size of a chess board. We've a piece standing in top-left corner and it must reach the bottom-right corner. The piece moves as follows –

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

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

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

You are supposed to write the following functions

a. Write a recursive function which prints all valid paths.

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

Input Format

Enter the size of the chessboard N

Constraints

None

Output Format

Display the total number of valid paths and print all the valid paths in a space separated manner

Sample Input

```
3
```

Sample Output

```
{0-0}K{2-1}R{2-2} {0-0}K{1-2}R{2-2} {0-0}R{0-1}K{2-
18
```

Sample output –

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

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println("\n"+chessboard(n - 1, "{0-0}", 0, 0));
    }

    public static int chessboard(int n, String ans, int curr_row,
int curr_col) {
        if (curr_row == curr_col && curr_row == n) {
            System.out.print(ans+" ");
            return 1;
        }

        if(curr_row > n || curr_col > n){
            return 0;
        }

        int a = 0;
        a+=chessboard(n, ans + "K{" + (curr_row + 2) + "-" +
(curr_col + 1) + "}", curr_row + 2, curr_col + 1);
        a+=chessboard(n, ans + "K{" + (curr_row + 1) + "-" +
(curr_col + 2) + "}", curr_row + 1, curr_col + 2);

        if (curr_row == 0 || curr_col == 0 || curr_row == n ||
curr_col == n) {
            for(int i=1; i<=n; i++){
                a+=chessboard(n, ans + "R{" + curr_row + "-" +
(curr_col + i) + "}", curr_row, curr_col +i);
            }
            for(int i=1; i<=n; i++){
                a+=chessboard(n, ans + "R{" + (curr_row + i) + "-" +
curr_col + "}", curr_row + i, curr_col);
            }
        }

        if (curr_row == curr_col || curr_col+curr_row == n) {
            for(int i=1;  i<=n; i++){
                a+=chessboard(n, ans + "B{" + (curr_row + i) + "-" +
(curr_col + i) + "}", curr_row + i, curr_col + i);
            }
        }
        return a;
    }
}
```

SUBSET PROBLEM

Take an input N, a number. Take N more inputs and store that in an array. Take an input target, a number

a. Write a recursive function which prints subsets of the array which sum to target.

b. Write a recursive function which counts the number of subsets of the array which sum to target. Print the value returned.

Input Format

Take an input N, a number. Take N more inputs and store that in an array. Take an input target, a number

Constraints

None

Output Format

Display the number of subsets and print the subsets in a space separated manner.

Sample Input

```
3
1
2
3
3
```

Sample Output

```
1 2  3
2
```

Explanation

Add 2 spaces between 2 subset solutions

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

```
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        int target = sc.nextInt();

        int count = subset(arr,target,0,0,0,"");
        System.out.println();
        System.out.println(count);
    }


    public static int subset(int [] arr, int target, int
curr_sum, int curr_index, int count, String ans){
        if(curr_sum == target){
            System.out.print(ans+" ");
            return 1;
        }
        else if(curr_sum > target){
            return 0;
        }
            int a = 0;
        for(int i=curr_index; i<arr.length; i++){
             a +=
subset(arr,target,curr_sum+arr[i],i+1,count,ans+arr[i]+"
");
        }
        return a;
    }
}
```

MAZEPATH_D

Take as input N1 and N2, both numbers. N1 and N2 is the number of rows and columns on a rectangular board. Our player starts in top-left corner of the board and must reach bottom-right corner. In one move the player can move 1 step horizontally (right) or 1 step vertically (down) or 1 step diagonally (south-east).

Write a recursive function which:
a. Returns the count of different ways the player can travel across the board. Print the value returned.
b. Prints moves for all valid paths across the board.

Input Format

Enter the number of rows N1 and number of columns N2

Constraints

None

Output Format

Display the total number of paths and print all the possible paths in a space separated manner

Sample Input

```
3
3
```

Sample Output

```
VVHH VHVH VHHV VHD VDH HVVH HVHV HVD HHVV HDV DVH D
13
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner  sc= new Scanner(System.in);
        int row = sc.nextInt();
        int col = sc.nextInt();
        int count = Maze_Path(row,col,0,0,"");
        System.out.println();
        System.out.print(count);
    }
```

```
        public static int Maze_Path(int row, int col, int
curr_col, int curr_row, String ans) {
        if(curr_col == col-1 && curr_row == row-1) {
            System.out.print(ans+" ");
            return 1;
        }
        if(curr_col >= col || curr_row >= row) {
            return 0;
        }
        int a = Maze_Path(row, col, curr_col, curr_row+1,
ans+"V");
        int b = Maze_Path(row, col, curr_col+1, curr_row,
ans+"H");
        int c =
Maze_Path(row,col,curr_col+1,curr_row+1,ans+"D");
        return a+b+c;
    }

}
```

RECURSION TWINS

Take an input str, a string. A "twin" is defined as two instances of a char separated by a char. E.g. "AxA" the A's make a "twin". "twins" can overlap, so "AxAxA" contains 3 "twins" - 2 for A and 1 for x. Write a function which recursively counts number of "twins" in a string. Print the value returned.

Input Format

Enter the string

Constraints

None

Output Format

Display the number of twins

Sample Input

AXAXA

Sample Output

3

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        twins(s,0,0);
    }

    public static int twins(String s, int index, int
count){
        if(index == s.length()-2){
            System.out.print(count);
            return count;
        }
```

```
        if(index < s.length()-2){
            if(s.charAt(index) == s.charAt(index+2) &&
s.charAt(index) != s.charAt(index+1)){
                twins(s,index+1,count+1);
            }
            else{
                twins(s,index+1,count);
            }
        }

        return count;
    }
}
```

MAZEPATH(COUNT,PRINT)

Take as input N1 and N2, both numbers. N1 and N2 is the number of rows and columns on a rectangular board. Our player starts in top-left corner of the board and must reach bottom-right corner. In one move the player can move 1 step horizontally (right) or 1 step vertically (down).

a. Write a recursive function which returns the count of different ways the player can travel across the board. Print the value returned.

b. Write a recursive function which returns an ArrayList of moves for all valid paths across the board. Print the value returned.

e.g. for a board of size 3,3; a few valid paths will be "HHVV", "VVHH", "VHHV" etc. c. Write a recursive function which prints moves for all valid paths across the board (void is the return type for function).

Input Format

Enter the number of rows N and columns M

Constraints

None

Output Format

Display the total number of paths and display all the possible paths in a space separated manner

Sample Input

```
3
3
```

Sample Output

```
VVHH VHVH VHHV HVVH HVHV HHVV
6
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner  sc= new Scanner(System.in);
        int row = sc.nextInt();
        int col = sc.nextInt();
        int count = Maze_Path(row,col,0,0,"");
        System.out.println();
```

```
        System.out.print(count);
    }

        public static int Maze_Path(int row, int col, int
curr_col, int curr_row, String ans) {
        if(curr_col == col-1 && curr_row == row-1) {
            System.out.print(ans+" ");
            return 1;
        }
        if(curr_col >= col || curr_row >= row) {
            return 0;
        }
        int a = Maze_Path(row, col, curr_col, curr_row+1,
ans+"V");
        int b = Maze_Path(row, col, curr_col+1, curr_row,
ans+"H");
        return a+b;
    }

}
```

BOARDPATH

Take as input N, a number. N is the size of a snakes and ladder board (without any snakes and ladders). Take as input M, a number. M is the number of faces of the dice.

a. Write a recursive function which returns the count of different ways the board can be traveled using the dice. Print the value returned.

b. Write a recursive function which prints dice-values for all valid paths across the board (void is the return type for function).

Input Format

Enter a number N (size of the board) and number M(number of the faces of a dice)

Constraints

None

Output Format

Display the number of paths and print all the paths in a space separated manner

Sample Input

```
3
3
```

Sample Output

```
111 12 21 3
4
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int target = sc.nextInt();
        int dice_face = sc.nextInt();
        System.out.println("\n" +
Print_path(0,target,"",dice_face));
    }
```

```
        public static int Print_path(int sum, int
des,String ans, int dice_face) {
        if(sum==des) {
            System.out.print(ans+" ");
            return 1;
        }
        else if(sum > des) {
            return 0;
        }
        int count = 0;
        for(int dice=1; dice<=dice_face; dice++) {
            count += Print_path(sum+dice, des,
ans+dice,dice_face);
        }
        return count;
    }

}
```

GENERATE PARETHESIS

Given an integer 'n'. Print all the possible pairs of 'n' balanced parentheses.
The output strings should be printed in the sorted order considering '(' has higher value than ')'.

Input Format

Single line containing an integral value 'n'.

Constraints

```
1<=n<=11
```

Output Format

Print the balanced parentheses strings with every possible solution on new line.

Sample Input

```
2
```

Sample Output

```
()()
(())
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n= sc.nextInt();
        parenthesis(0,0,"",n);
    }

    public static void parenthesis(int open, int close,
String ans,int n) {
        if(open == close && open ==n) {
            System.out.println(ans);
            return;
        }
        else if(close > open || open>n) {
            return;
        }

        parenthesis(open,close+1,ans+")",n);
        parenthesis(open+1,close,ans+"(",n);
    }
}
```

VIVEK LOVES ARRAY GAME

```
Initially, there is an array, A , containing 'N' integers.

In each move, Vivek must divide the array into  two non-empty
contiguous parts such that the sum of the elements in the left part is
equal  to the sum of the elements in the right part. If Vivek can make
such a move, he gets '1' point; otherwise, the game ends.
```

Vivek loves to play with array . One day Vivek just came up with a new array game which was introduced to him by his friend Ujjwal. The rules of the game are as follows:

```
Initially, there is an array, A , containing 'N' integers.

In each move, Vivek must divide the array into  two non-empty contiguous parts such that the s

After each successful move, Vivek have to discards either the left part or the right part and
```

the remaining partition as array 'A'.

Vivek loves this game and wants your help getting the best score possible. Given 'A', can you find and print the maximum number of points he can score?

Input Format

First line of input contains an integer T denoting number of test cases. Each further Test case contains first line an integer 'N' , the size of array 'A'. After that 'N' space separated integers denoting the elements of array.

Constraints

1 <= T <= 10 1 <= N <= 17000 0 <= A[i] <= 10^9

Output Format

For each test case, print Vivek's maximum possible score on a new line.

Sample Input

```
3
3
3 3 3
4
2 2 2 2
7
4 1 0 1 1 0 1
```

Sample Output

```
0
2
3
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        for(int i=0; i<t; i++){
            int n = sc.nextInt();
            int [] arr = new int[n];
            for(int j=0; j<n; j++){
                arr[j] = sc.nextInt();
            }
            if(n <= 1){
                System.out.println(0);
            }
            else{
            System.out.println(equal_arrays(arr,0,n-1));
            }
        }
    }
    public static int equal_arrays(int [] arr, int
start_index, int end_index){
        for (int center = start_index; center < end_index;
center++) {
            int left_sum = 0;
            for (int i = start_index; i <= center; i++) {
                left_sum += arr[i];
            }
            int right_sum = 0;
            for (int i = center+1; i <= end_index; i++) {
                right_sum += arr[i];
            }
            if(left_sum == right_sum) {
                int left_count = equal_arrays(arr,
start_index, center);
                int right_count = equal_arrays(arr,
center+1, end_index);
                return Math.max(left_count,
right_count)+1;
            }
        }
        return 0;
    }}
```

DICTIONARY ORDER SMALLER

Take as input str, a string. Write a recursive function which returns all the words possible by rearranging the characters of this string which are in dictionary order smaller than the given string. The output strings must be lexicographically sorted.

Input Format

Single line input containing a string

Constraints

Length of string <= 10

All characters are ***unique***

Output Format

Display all the words which are in dictionary order smaller than the string entered in a new line each. The output strings must be sorted.

Sample Input

```
cab
```

Sample Output

```
abc
acb
bac
bca
```

Explanation

The possible permutations of string "cab" are "abc" , "acb" ,"bac" , "bca" , "cab" and "cba" . Only four of them are lexicographically less than "cab". We print them in lexicographical order.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        String sorted = "";
        for (int i=0; i<256; i++){
            int j=0;
            while(j<s.length()){
                if(s.charAt(j)==(char)i){
                    sorted+=s.charAt(j);
                    break;
                }
                j++;
            }
        }

        lex(sorted,"",s);

    }

    public static void lex(String sorted, String ans,
String ques){

        if(sorted.length() == 0 ){
            if(ques.compareTo(ans) > 0){
            System.out.println(ans);
            return;
            }

        }

        for(int i=0;i<sorted.length();i++){
                String s1 = sorted.substring(0,i);
                String s2 = sorted.substring(i+1);
                lex(s1+s2,ans+sorted.charAt(i),ques);
        }
    }
}
```

TOWER OF HANOI

Using a helper stick (peg), shift all rings from peg **A** to peg **B** using peg **C**.
All rings are initally placed in ascending order, smallest being on top.
No bigger ring can be placed over a smaller ring.

Input Format

Single line input containing a single integer N denoting the no of rings.

Constraints

```
1 <= N <= 10
```

Output Format

Print the instructions to move all the rings from peg **A** to **B** in a new line each.
Each line should follow format : *Moving ring **i** from **A**/**B**/**C** to **A**/**B**/**C***

Sample Input

```
4
```

Sample Output

```
Moving ring 1 from A to C
Moving ring 2 from A to B
Moving ring 1 from C to B
Moving ring 3 from A to C
Moving ring 1 from B to A
Moving ring 2 from B to C
Moving ring 1 from A to C
Moving ring 4 from A to B
Moving ring 1 from C to B
Moving ring 2 from C to A
Moving ring 1 from B to A
Moving ring 3 from C to B
Moving ring 1 from A to C
Moving ring 2 from A to B
Moving ring 1 from C to B
```

Explanation

Read Tower of Hanoi

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        TOH(n,"A","C","B");
    }

    public static void TOH(int n, String source, String
helper, String destination){
        if(n==0)
            return;
        TOH(n-1,source,destination,helper);
        System.out.println("Moving ring "+n+" from
"+source+" to "+destination);
        TOH(n-1,helper,source,destination);
    }
}
```

DICTIONARY ORDER LARGER

Take as input str, a string. Write a recursive function which prints all the words possible by rearranging the characters of this string which are in dictionary order larger than the given string.
The output strings must be lexicographically sorted.

Input Format

Single line input containing a string

Constraints

Length of string <= 10

All characters are ***unique***

Output Format

Display all the words which are in dictionary order larger than the string entered in a new line each. The output strings must be sorted.

Sample Input

```
cab
```

Sample Output

```
cba
```

Explanation

The possible permutations of string "cab" are "abc" , "acb" ,"bac" , "bca" , "cab" and "cba" . Only one of them is lexicographically greater than "cab". We only print "cba".

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        String sorted = "";
        for (int i=0; i<256; i++){
            int j=0;
            while(j<s.length()){
                if(s.charAt(j)==(char)i){
                    sorted+=s.charAt(j);
                    break;
                }
                j++;
            }
        }

        lex(sorted,"",s);

    }

    public static void lex(String sorted, String ans,
String ques){

        if(sorted.length() == 0 ){
            if(ques.compareTo(ans) < 0){
            System.out.println(ans);
            return;
            }

        }

        for(int i=0;i<sorted.length();i++){
                String s1 = sorted.substring(0,i);
                String s2 = sorted.substring(i+1);
                lex(s1+s2,ans+sorted.charAt(i),ques);
        }
    }
}
```

LAST INDEX

Take as input N, the size of array. Take N more inputs and store that in an array. Take as input M, a number. Write a recursive function which returns the last index at which M is found in the array and -1 if M is not found anywhere. Print the value returned.

Input Format

Enter a number N and add N more numbers to an array, then enter number M to be searched

Constraints

None

Output Format

Display the last index at which the number M is found

Sample Input

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

Sample Output

```
3
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int [] arr = new int [n];
        for(int i=0; i<arr.length; i++){
            arr[i] = sc.nextInt();
        }
        int target = sc.nextInt();
        last_index(target, arr, n-1);
    }

    public static void last_index(int target, int []arr,
int curr_index){
        if(curr_index == -1){
            System.out.println(-1);
            return;
        }
        if(arr[curr_index]==target){
            System.out.println(curr_index);
            return;
        }
        last_index(target,arr,curr_index-1);
    }
}
```

RECURSION LEXICOGRAPHICAL ORDER

Take as input N, a number. Write a recursive function which prints counting from 0 to N in lexicographical order. In lexicographical (dictionary) order 10, 100 and 109 will be printed before 11.

Input Format

Enter a number N.

Constraints

None

Output Format

Display all the numbers upto N in a lexicographical order

Sample Input

```
10
```

Sample Output

```
0 1 10 2 3 4 5 6 7 8 9
```

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

    public static void Print_count(int curr, int n) {
        // TODO Auto-generated method stub
        if(curr > n) {
            return;
        }
        System.out.print(curr+" ");
        int i=0;
        if(curr == 0) {
            i=1;
        }
        for(; i<=9; i++) {
            Print_count(curr*10+i,n);
        }
    }
}
```

FIRST INDEX

Take as input N, the size of array. Take N more inputs and store that in an array. Take as input M, a number. Write a recursive function which returns the first index at which M is found in the array and -1 if M is not found anywhere. Print the value returned.

Input Format

Enter a number N and add N more elements to an array, then enter a number M

Constraints

None

Output Format

Display the first index at which number M is found

Sample Input

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

Sample Output

```
1
```

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

    public static void first_index(int target, int []arr,
int curr_index){
        if(curr_index == arr.length){
            System.out.println(-1);
            return;
        }
        if(arr[curr_index]==target){
            System.out.println(curr_index);
            return;
        }
        first_index(target,arr,curr_index+1);
    }
}
```

GENERATE BINARY STRINGS

Given a string containing of '0', '1' and '?' wildcard characters, generate all binary strings that can be formed by replacing each wildcard character by '0' or '1'.

Input Format

The first line of input contains a single integer T denoting the number of test cases. Then T test cases follow. Each test case consist of two lines. The first line of each test case consists of a string S.

Constraints

1 ≤ T ≤ 60 1 ≤ length of string S ≤ 30

Output Format

Print all binary string that can be formed by replacing each wildcard character separated by space.

Sample Input

```
1
1??0?101
```

Sample Output

```
10000101 10001101 10100101 10101101 11000101 110011
```

Explanation

For the Given test case, 10000101 can be generated by replacing wildcard character from 1??0?101 by 0s.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for(int i=0; i<n; i++){
            String s = sc.next();
            binary_string(s,0,"");
            System.out.println();
        }
    }

    public static void binary_string(String s, int
curr_index, String ans){
        if(ans.length()==s.length()){
            System.out.print(ans+" ");
            return;
        }

            if(s.charAt(curr_index)=='0' ||
s.charAt(curr_index)=='1'){

binary_string(s,curr_index+1,ans+s.charAt(curr_index));
            }
            else{
                binary_string(s,curr_index+1,ans+'0');
                binary_string(s,curr_index+1,ans+'1');
            }
        }
}
```

ALL INDICES PROBLEM

Take as input N, the size of array. Take N more inputs and store that in an array. Take as input M, a number. Write a recursive function which returns an array containing indices of all items in the array which have value equal to M. Print all the values in the returned array.

Input Format

Enter a number N(size of the array) and add N more numbers to the array Enter number M to be searched in the array

Constraints

```
1 <= Size of array <= 10^5
```

Output Format

Display all the indices at which number M is found in a space separated manner

Sample Input

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

Sample Output

```
1 3
```

Explanation

```
2 is found at indices 1 and 3.
```

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

    public static void indexes(int []arr, int curr_index,
int target){
        if(curr_index == arr.length){
            return;
        }
        else if(arr[curr_index] == target){
            System.out.print(curr_index+" ");
        }

        indexes(arr,curr_index+1,target);
    }
}
```

RECURSION KEYPAD CODES

Take as input str, a string. str represents keys pressed on a nokia phone keypad. We are concerned with all possible words that can be written with the pressed keys.

Assume the following alphabets on the keys: 1 -> abc , 2 -> def , 3 -> ghi , 4 -> jkl , 5 -> mno , 6 -> pqrs , 7 -> tuv , 8 -> wx , 9 -> yz

E.g. "12" can produce following words "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"

a. Write a recursive function which returns the count of words for a given keypad string. Print the value returned.

b.Write a recursive function which prints all possible words for a given keypad string (void is the return type for function).

Input Format

Single line input containing a single integer.

Constraints

```
1 <= Length of string <= 10^3
```

Output Format

Print all the words in a space separated manner. Also print the count of these numbers in next line.

Sample Input

```
12
```

Sample Output

```
ad ae af bd be bf cd ce cf
9
```

Explanation

```
1 can correspond to 'a' , 'b' or 'c' .
2 can correspond to 'd', 'e' or 'f'.
We print all combinations of these
```

```java
import java.util.*;
public class Main {
    static String [] key =
{"","abc","def","ghi","jkl","mno","pqrs","tuv","wx","yz"};
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println("\n"+comb(s,""));

    }

        public static int comb(String ques, String ans) {
        if(ques.length() == 0) {
            System.out.print(ans+" ");
            return 1;
        }

        char c = ques.charAt(0);
        String press = key[c-'0'];
        int a = 0;
        for(int i=0; i<press.length(); i++) {
            a+=comb(ques.substring(1),
ans+press.charAt(i));
        }
        return a;
    }

}
```

RECURSION SUBSEQUENCE

Take as input str, a string. We are concerned with all the possible subsequences of str. E.g.

a. Write a recursive function which returns the count of subsequences for a given string. Print the value returned.

b. Write a recursive function which prints all possible subsequences for a "abcd" has following subsequences "", "d", "c", "cd", "b", "bd", "bc", "bcd", "a", "ad", "ac", "acd", "ab", "abd", "abc", "abcd".given string (void is the return type for function).
Note: Use cin for input for C++

Input Format

Enter a string

Constraints

None

Output Format

Print all the subsequences in a space separated manner and isplay the total no. of subsequences.

Sample Input

```
abcd
```

Sample Output

```
 d c cd b bd bc bcd a ad ac acd ab abd abc abcd
16
```

Explanation

Print all possible subsequences of the given string.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println("\n"+subseq(s,"",0));
    }

    public static int subseq(String ques, String ans, int
count) {
        if(ques.length()==0) {
            System.out.print(ans+" ");
            return 1;
        }
        char ch = ques.charAt(0);
        int a = subseq(ques.substring(1),ans,count);
        int b = subseq(ques.substring(1),ans+ch,count);
        return a+b;
    }
}
```

REPLACE ALL PI

Replace all occurrences of pi with 3.14

Input Format

Integer N, no of lines with one string per line

Constraints

$0 < N < 1000$
$0 <= len(str) < 1000$

Output Format

Output result one per line

Sample Input

```
3
xpix
xabpixx3.15x
xpipippixx
```

Sample Output

```
x3.14x
xab3.14xx3.15x
x3.143.14p3.14xx
```

Explanation

All occurrences of pi have been replaced with "3.14".

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

    public static void replace(String s, int index, String
ans){

        if(index == s.length()-1){
            System.out.println(ans+s.charAt(index));
            return;
        }
        else if(index == s.length()-1){
            System.out.println(ans);
            return;
        }

        if((s.substring(index,index+2)).equals("pi")){
                replace(s,index+2,ans+3.14);
            }
        else{
                replace(s,index+1,ans+s.charAt(index));
            }
        }
}
```

REPLACE 0 WITH 5

Given an integer N, now you have to convert all zeros of N to 5.

Input Format

The first Line takes input integer N, denoting the number.

Constraints

1<=N<=10000

Output Format

Print the number after replacing all 0's with 5.

Sample Input

```
103
```

Sample Output

```
153
```

Explanation

Testcase 1: after replacing 0 with 5 ,number will become 153.
Testcase 2: there is no zero in number so it will remain same.

```java
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        replace(n, 1, 0);
    }

    public static void replace(int n, int power, int num){
        if(n==0){
        System.out.println(num);
            return;
        }
        else if(n % 10 == 0){
            num = num + 5*power;
        }
        else{
            num = num + (n%10)*power;
        }
        replace(n/10,power*10,num);
    }
}
```

RAT IN A MAZE

You are given an N*M grid. Each cell (i,j) in the grid is either blocked, or empty. The rat can move from position (i,j), down or right on the grid.
Initially rat is on the position (1,1). It wants to reach position (N,M). Find the rightmost path through which, rat can reach this position. A path is rightmost, if the rat is at position (i,j), it will always move to (i,j+1), if there exists a path from (i,j+1) to (N,M).

Input Format

First line contains 2 integers, N and M, denoting the rows and columns in the grid. Next N line contains. M characters each. An 'X' in position (i,j) denotes that the cell is blocked and ans 'O' denotes that the cell is empty.

Constraints

1<=N,M<=1000 GRID(i,j)='X' or 'O'

Output Format

If a solution exists: Print N lines, containing M integers each. A 1 at a position (i,j) denotes that the $(i,j)^{th}$ cell is covered in the path and a 0 denotes that the cell is not covered in the path.
If solution doesn't exist, just print "-1".

Sample Input

```
5 4
0X00
000X
00X0
X000
XX00
```

Sample Output

```
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1
0 0 0 1
```

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

        int [][] ans = new int[n][m];
        path(matrix, 0, 0, ans);

        if(flag == false){
            System.out.println("-1");
        }
    }

    static boolean flag = false;
    public static void path(char [][] matrix, int
curr_row, int curr_col, int[][]ans){
        if(curr_row == matrix.length-1 && curr_col ==
matrix[0].length-1 && matrix[curr_row][curr_col] != 'X'){
            ans[curr_row][curr_col] = 1;
            dis(ans);
            flag = true;
            return;
        }
        if(curr_row >= matrix.length || curr_col >=
matrix[0].length || matrix[curr_row][curr_col] == 'X'){
            return;
        }
            if(flag == false){
                ans[curr_row][curr_col] = 1;
                matrix[curr_row][curr_col] = 'X';
                path(matrix,curr_row,curr_col+1, ans);
                path(matrix, curr_row+1, curr_col, ans);
```

```
                matrix[curr_row][curr_col] = 'O';
                ans[curr_row][curr_col] = 0;
            }
        }


    public static void dis(int[][]ans){
        for(int i=0; i<ans.length; i++){
            for(int j=0; j<ans[0].length;j++){
                System.out.print(ans[i][j]+" ");
            }
            System.out.println();
        }
    }

}
```

ODD EVEN

Take as input **N**, a number. Print *odd* numbers in decreasing sequence (up until 0) and *even* numbers in increasing sequence (up until N) using Recursion

Input Format

Constraints

```
1 <= N <=1000
```

Output Format

Sample Input

```
5
```

Sample Output

```
5
3
1
2
4
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        int cp = num;
        if(cp%2 == 0){
            cp-=1;
        }
        else{
            num-=1;
        }
        print(cp,num,2);
    }

    public static void print(int cp, int num, int seq){

        if(cp == -1 && seq == num+2){
            return;
        }
        if(cp>=1){
            System.out.println(cp);
            print(cp-2,num,seq);
        }
        else if(seq <= num){
            System.out.println(seq);
            print(cp,num,seq+2);
        }
    }
}
```

COUNT REMOVE REPLACE HI

Take as input str, a string. a.Write a recursive function which counts the number of times 'hi' appears in the string. Print the value returned. b.Write a recursive function which removes all 'hi' in the string. Print the value returned. c.Write a recursive function which replaces all 'hi' in the string with 'bye'. Print the value returned.

Input Format

Enter a string

Constraints

None

Output Format

Display the no. of 'hi', string without 'hi' , string in which 'hi' is replaced with 'bye'

Sample Input

```
abchibi
```

Sample Output

```
1
abcbi
abcbyebi
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        replace(s,0,"","",0);
    }

     public static void replace(String s, int index,
String ans,String ans1, int count){
        if(index == s.length()-1){

System.out.println(count+"\n"+ans+s.charAt(index)+"\n"+ans
1+s.charAt(index));
            return;
        }


        if((s.substring(index,index+2)).equals("hi")){
                replace(s,index+2,ans,ans1+"bye",count+1);
            }
        else{

replace(s,index+1,ans+s.charAt(index),ans1+s.charAt(index)
,count);
            }
        }
}
```

NTH TRIANGLE

Take as input N, a number. Write a recursive function to find Nth triangle where 1st triangle is 1, 2nd triangle is 1 + 2 = 3, 3rd triangle is 1 + 2 + 3 = 6, so on and so forth. Print the value returned.

Input Format

Integer **N** is the single line of input.

Constraints

```
1 <= N <= 100
```

Output Format

Print the output as a single integer which is the **nth** triangle.

Sample Input

```
3
```

Sample Output

```
6
```

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

    public static void triangle(int n, int sum){
        if(n==0){
            System.out.println(sum);
            return;
        }
        triangle(n-1,sum+n);
    }
}
```

SPLIT ARRAY

Take as input N, a number. Take N more inputs and store that in an array.

a. Write a recursive function which counts the number of ways the array could be split in two groups, so that the sum of items in both groups is equal. Each number in the array must belong to one of the two groups. Print the value returned.

b. Write a recursive function which keeps track of ways an array could be split in two groups, so that the sum of items in both groups is equal. Each number in the array must belong to one of the two groups. Return type of function should be void. Print the two groups, each time you find a successful split.

Input Format

Take as input N, a number. Take N more inputs and store that in an array.

Constraints

None

Output Format

Display all the groups in a comma separated manner and display the number of ways the array can be split

Sample Input

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

Sample Output

```
1 2 3 3 and 4 5
1 3 5 and 2 3 4
1 3 5 and 2 3 4
2 3 4 and 1 3 5
2 3 4 and 1 3 5
4 5 and 1 2 3 3
6
```

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

    public static int split_array(int [] arr, int cur_i,
int lt_sum, String lt, int rt_sum, String rt){
        if(cur_i == arr.length){
            if(lt_sum == rt_sum){
                System.out.println(lt+"and"+rt);
                return 1;
            }
            return 0;
        }
        int a = 0;
        a += split_array(arr, cur_i+1, lt_sum+arr[cur_i],
lt+arr[cur_i]+" ",rt_sum, rt );
        a += split_array(arr, cur_i+1, lt_sum, lt,
rt_sum+arr[cur_i], rt+" "+arr[cur_i]);
        return a;
    }
}
```

COUNT REMOVE REPLACE HI PART 2

Take as input str, a string.

a.) Write a recursive function which counts the number of times 'hi' appears in the string – but skip all such 'hi' which are followed by 't' to form 'hit'. Print the value returned.

b.) Write a recursive function which removes all 'hi' in the string – but skip all such 'hi' which are followed by 't' to form 'hit'. Print the value returned.

c.) Write a recursive function which replaces all 'hi' in the string with 'bye' – but skip all such 'hi' which are followed by 't' to form 'hit'. Print the value returned.

Input Format

Enter the String

Constraints

None

Output Format

Display the total no.of "hi" ("hi" should not be followed by a 't'), string in which all "hi" are removes("hi" should not be followed by a 't'), string in which all "hi" are replace by "bye"("hi" should not be followed by a 't')

Sample Input

```
abchihitfhi
```

Sample Output

```
2
abchitf
abcbyehitfbye
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
     Scanner sc = new Scanner(System.in);
        String s = sc.next();
        replace(s,0,"","",0);
```

```
    }

     public static void replace(String s, int index,
String ans,String ans1, int count){
        if(index == s.length()){
            System.out.println(count+"\n"+ans+"\n"+ans1);
            return;
        }
        else if(index == s.length()-2){
            if((s.substring(index,index+2)).equals("hi")){

System.out.println(count+1+"\n"+ans+"\n"+ans1+"bye");
            }
            else{

System.out.println(count+"\n"+ans+s.substring(index,index+
2)+"\n"+ans1+s.substring(index,index+2));
            }
            return;
        }
        else if(index == s.length()-1){

System.out.println(count+"\n"+ans+s.charAt(index)+"\n"+ans
1+s.charAt(index));
            return;
        }

        if((s.substring(index,index+2)).equals("hi")){
            if(s.charAt(index+2)=='t'){

replace(s,index+3,ans+"hit",ans1+"hit",count);
            }
            else{
                replace(s,index+2,ans,ans1+"bye",count+1);
            }
        }
        else{
replace(s,index+1,ans+s.charAt(index),ans1+s.charAt(index)
,count);
            }
        }
}
```

COUNT N QUEEN

You are given an empty chess board of size N*N. Find the number of ways to place N queens on the board, such that no two queens can kill each other in one move. A queen can move vertically, horizontally and diagonally.

Input Format

A single integer N, denoting the size of chess board.

Constraints

1<=N<=11

Output Format

A single integer denoting the count of solutions.

Sample Input

4

Sample Output

2

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        boolean [][] queen_placed = new boolean[n][n];
        System.out.println(count_queens(n,0,0,0,queen_plac
ed,0));
    }


    public static int count_queens(int n, int queens, int
cr, int cc, boolean[][] queen_placed, int count) {

        if(queens == n) {

            return count+1;
        }
        if(cr >= n || cc >= n) {
```

```
            return 0;
        }

        for(int i=0; i<n; i++) {
                if(is_possible(queen_placed, cr, i, n)) {
                    queen_placed[cr][i] = true;
                    count =
count_queens(n,queens+1,cr+1,i,queen_placed,count);
                    queen_placed[cr][i] = false;
                }
        }
        return count;
    }

    public static boolean is_possible(boolean [][]
queen_placed, int cr, int cc, int n) {

        for(int i=0; i<cr; i++) { // for checking column
            if(queen_placed[i][cc]) {
                return false;
            }
        }

        for(int i=cr, j=cc; i>-1 && j>-1; i--,j--) { //
upper left diagonal
            if(queen_placed[i][j]) {
                return false;
            }
        }

        for(int i=cr, j=cc; i>-1 && j<n; i--,j++) { //
upper right diagonal
            if(queen_placed[i][j]) {
                return false;
            }
        }
        return true;
    }
}
```

RAT CHASES ITS CHEESE

You are given an N*M grid. Each cell (i,j) in the grid is either blocked, or empty. The rat can move from a position towards left, right, up or down on the grid.
Initially rat is on the position (1,1). It wants to reach position (N,M) where it's cheese is waiting for. There exits a unique path in the grid . Find that path and help the rat reach its cheese.

Input Format

First line contains 2 integers N and M denoting the rows and columns in the grid.
Next N line contains M characters each. An 'X' in position (i,j) denotes that the cell is blocked and ans 'O' denotes that the cell is empty.

Constraints

```
1 <= N , M <= 10
```

Output Format

Print N lines, containing M integers each. A 1 at a position (i,j) denotes that the $(i,j)^{th}$ cell is covered in the path and a 0 denotes that the cell is not covered in the path.
If a path does not exits then print "NO PATH FOUND"

Sample Input

```
5 4
0X00
000X
X0X0
X00X
XX00
```

Sample Output

```
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 0
0 0 1 1
```

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

        int [][] ans = new int[n][m];
        path(matrix, 0, 0, ans);

        if(flag == false){
            System.out.println("NO PATH FOUND");
        }
    }

    static boolean flag = false;
    public static void path(char [][] matrix, int
curr_row, int curr_col, int[][]ans){
        if(curr_row == matrix.length-1 && curr_col ==
matrix[0].length-1 && matrix[curr_row][curr_col] != 'X'){
            ans[curr_row][curr_col] = 1;
            dis(ans);
            flag = true;
            return;
        }
        if(curr_row >= matrix.length || curr_col >=
matrix[0].length || curr_col < 0 || curr_row < 0 ||
matrix[curr_row][curr_col] == 'X'){
            return;
        }

                ans[curr_row][curr_col] = 1;
                matrix[curr_row][curr_col] = 'X';
                path(matrix, curr_row-1, curr_col, ans);
                path(matrix, curr_row+1, curr_col, ans);
```

```
                path(matrix,curr_row,curr_col+1, ans);
                path(matrix,curr_row, curr_col-1, ans);
                matrix[curr_row][curr_col] = 'O';
                ans[curr_row][curr_col] = 0;
        }


    public static void dis(int[][]ans){
        for(int i=0; i<ans.length; i++){
            for(int j=0; j<ans[0].length;j++){
                System.out.print(ans[i][j]+" ");
            }
            System.out.println();
        }
    }

}
```

CONVERT STRING TO INT

Take as input str, a number in form of a string. Write a recursive function to convert the number in string form to number in integer form. E.g. for "1234" return 1234. Print the value returned.

Input Format

Enter a number in form of a String

Constraints

1 <= Length of String <= 10

Output Format

Print the number after converting it into integer

Sample Input

```
1234
```

Sample Output

```
1234
```

Explanation

Convert the string to int. Do not use any inbuilt functions.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        s_i(s,0,s.length()-1,1);
    }

    public static void s_i(String s, int num, int
index,int power){
        if(index == -1){
            System.out.println(num);
            return;
        }
        int i = s.charAt(index)-48;
        num = num + i*power;
        s_i(s,num,index-1,power*10);
    }
}
```

MAPPED STRINGS

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

Input Format

A single line contains a number.

Constraints

Number is less than 10^6

Output Format

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

Sample Input

```
123
```

Sample Output

```
ABC
AW
LC
```

Explanation

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

```java
import java.util.*;

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

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

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

        }
    }
}
```

N KNIGHT PROBLEM

Take as input N, the size of a chess board. We are asked to place N number of Knights in it, so that no knight can kill other.

a. Write a recursive function which returns the count of different distinct ways the knights can be placed across the board. Print the value returned.

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

Input Format

Enter the size of the chessboard N

Constraints

None

Output Format

Display the number of ways a knight can be placed and print all the possible arrangements in a space separated manner

Sample Input

```
2
```

Sample Output

```
{0-0} {0-1}  {0-0} {1-0}  {0-0} {1-1}  {0-1} {1-0}
6
```

```
{0-0} {0-1}  {0-0} {1-0}  {0-0} {1-1}  {0-1} {1-0}  {0-1} {1-1}  {1-0} {1-1}
6
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        boolean [][]placed = new boolean[n][n];
        count_knight(placed,n,0,0,0,"");
        System.out.print("\n"+count);
```

```
    }
    static int count = 0;
    public static void count_knight(boolean [][]placed,
int n, int cr, int cc, int knight, String ans){
        if(knight == n){
            System.out.print(ans+" ");
            count++;
            return;
        }
        if(cc == n){
            cr++;
            cc = 0;
        }
        if(cr == n){
            return;
        }

        if(could_placed(placed,n,cr,cc)){
            placed[cr][cc] = true;
            count_knight(placed,n,cr,cc+1,knight+1,
ans+"{"+cr+"-"+cc+"} ");
            placed[cr][cc] = false;
        }
        count_knight(placed,n,cr,cc+1,knight, ans);
    }

    public static boolean could_placed(boolean[][]placed,
int n, int cr, int cc){
        if(cr-2 >= 0 && cc-1 >= 0 && placed[cr-2][cc-1])
                return false;

        if(cc+1 < n && cr-2 >=0 && placed[cr-2][cc+1])
                return false;

        if(cr-1 >= 0 && cc-2 >= 0 && placed[cr-1][cc-2])
               return false;

        if(cc+2 < n && cr-1>=0 && placed[cr-1][cc+2])
                return false;
        return true;
    }
}
```

RECURSION OBEDIENT STRING

Take as input str, a string. Write a recursive function that checks if the string was generated using the following rules and returns a Boolean value:
1. the string begins with an 'a'
2. each 'a' is followed by nothing or an 'a' or "bb"
3. each "bb" is followed by nothing or an 'a' Print the value returned.

Input Format

Enter the String

Constraints

None

Output Format

Display the boolean result

Sample Input

```
abba
```

Sample Output

```
true
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.print(obedient_string(s,0));
    }
    static boolean stat = true;
```

```
    public static boolean obedient_string(String s, int
ci){
        if(ci == s.length() && stat == true){
            return stat;
        }

        if(ci == 0 && s.charAt(ci)=='a'){
            ci++;
            obedient_string(s, ci);
        }
        else if(ci == 0 && s.charAt(ci) != 'a'){
            stat=false;
            return stat;
        }

        if(ci <= s.length()-2){
            if((s.substring(ci,ci+2)).equals("bb") &&
s.charAt(ci-1) != 'b'){
                ci+=2;
                obedient_string(s,ci);
            }
            else if(s.charAt(ci)=='a'){
                ci++;
                obedient_string(s,ci);
            }
            else{
                stat=false;
            }
        }

        if(ci == s.length()-1){
            if(s.charAt(ci)=='a'){
                ci++;
                obedient_string(s,ci);
            }
            else{
                stat=false;
            }
        }
        return stat;
    }
}
```

SUDOKU SOLVER

A sudoku solution must satisfy all of the following rules:

Each of the digits 1-9 must occur exactly once in each row.
Each of the digits 1-9 must occur exactly once in each column.
Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.<br.

Input Format

First line contains a single integer N. Next N lines contains N integers each, where j^{th} integer int i^{th} line denotes the value at i^{th} row and j^{th} column in sudoku grid. This value is 0, if the cell is empty.

Constraints

N=9
At least one solution does exists for input matrix.

Output Format

Print N lines containing N integers each, where j^{th} integer int i^{th} line denotes the value at i^{th} row and j^{th} column in sudoku grid, such that all cells are filled.

Sample Input

```
9
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

Sample Output

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

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

    }
    static boolean flag = false;
    public static void sudoku_solver(int [][] sudoku, int
cr, int cc) {

        if(cc == sudoku.length) {
            cr++;
            cc=0;
        }

        if(cr == sudoku.length) {
            print(sudoku);
            flag = true;
            return;
        }

        if(flag)
            return;
        if(sudoku[cr][cc]!=0)
            sudoku_solver(sudoku,cr,cc+1);

        for(int i=1; i<=sudoku.length;i++) {
            if(sudoku[cr][cc]==0 && is_exist_3(sudoku, cr,
cc, i) && is_exist_9(sudoku, cr, cc, i)) {
```

```
                sudoku[cr][cc] = i;
                sudoku_solver(sudoku, cr, cc+1);
                sudoku[cr][cc] = 0;
            }
        }

    }

    public static boolean is_exist_3(int [][]sudoku, int
cr, int cc, int num) {
        int check_row = (cr/3)*3;
        int check_col = (cc/3)*3;
        for(int i=check_row; i<check_row+3;i++) {
            for(int j=check_col; j<check_col+3;j++) {
                if(sudoku[i][j] == num) {
                    return false;
                }
            }
        }
        return true;
    }

    public static boolean is_exist_9(int [][]sudoku, int
cr, int cc, int num) {
        for(int i=0; i<sudoku.length;i++) {
            if(sudoku[i][cc]==num) {
                return false;
            }
            if(sudoku[cr][i]==num) {
                return false;
            }
        }
        return true;
    }

    public static void print(int [][] ans){
        for(int i=0; i<ans.length;i++){
            for(int j=0; j<ans[0].length;j++){
                System.out.print(ans[i][j]+" ");
            }
            System.out.println();
        }
```

```
        }
}
```