'''

Bindu is passionated about clocks.

She found a digital clock, which has 2 rows of led lights,

Top row has 4-leds OOOO indicates hours (0-11).

Bottom row has 6-leds OOOOOO indicates minutes(0-59).

Where O -> led is OFF and \* -> led is ON.

For Example:

If the clock showing the led lights are as follows:

OO\*O indiactes a binary number, 0010 -> 2 hrs

OO\*O\*\* indiactes a binary number, 001011 -> 11 minutes.

So, Time is 2:11

Now, you are given an integer N, number of led lights that are ON.

Your task is to find, all the possible times in ascending order.

Format of time should be as follows:

- Hours should not have leading 0's.

- Minutes should be represented with 2 digits.

Suppose Current time is 1 hous 1 minute

- Valid time is 1:01

- Invalid time is as follows:- 01:1, 01:01, 1:1

Input Format:

-------------

An integer N, number of led lights ON.

Output Format:

--------------

Print the list of times in ascending order.

Sample Input:

---------------

1

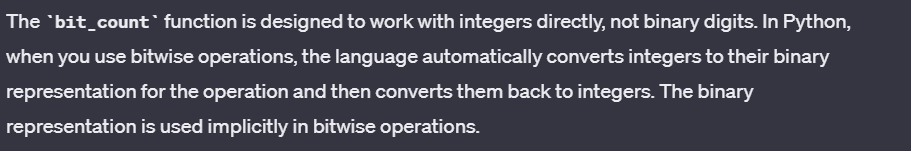
Sample Output:

----------------

[0:01, 0:02, 0:04, 0:08, 0:16, 0:32, 1:00, 2:00, 4:00, 8:00]

Write your python code below

'''



def bit\_count(num):

count=0

while (num>0):

count+=num&1

num=num>>1

return count

n=int(input())

l=[]

for i in range(12):

for j in range(60):

if(bit\_count(i)+bit\_count(j)==n):

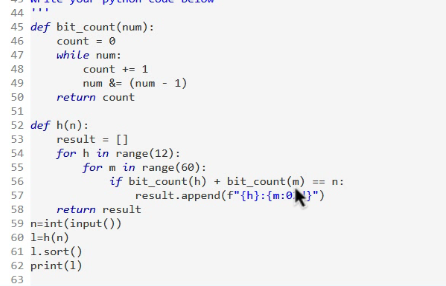
if(len(str(j))==1):

l.append(str(i)+":"+"0"+str(j))

else:

l.append(str(i)+":"+str(j))

print(l)



You are entering into a maze N\*N grid consist of(0's and 1's)

Initially you will start from (0,0) position in the maze,

Your target is to reach the end position (N-1, N-1).

Among the four directions available(top, down, left, right),

you can move in two directions only, right and down.

In the maze, '0' indicates dead end and '1' indicates open way.

You can travel through only open way.

For Example:-

Given Maze of size N=4 , starting position is (0, 0),

1 0 0 0

1 1 0 1

0 1 0 0

1 1 1 1

You can reach the (3, 3) position in the following way.

1 0 0 0

1 1 0 0

0 1 0 0

0 1 1 1

Return true, if you can reach the end position(N-1, N-1).

otherwise return false.

Input Format:

-------------

Line-1 -> An integer N, size of square board.

Next N lines -> N space separated integers

Output Format:

--------------

Print a boolean value.

Sample Input-1:

---------------

4

1 0 0 0

1 1 0 1

0 1 0 0

1 1 1 1

Sample Output-1:

----------------

true

Sample Input-2:

---------------

4

1 1 0 0

1 0 0 1

0 1 1 0

1 0 0 1

Sample Output-2:

----------------

False

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[][] arr=new int[n][n];

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

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

arr[i][j]=sc.nextInt();

}

}

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

}

public static boolean fun(int[][] arr,int n){

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

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

if(arr[i][j]==1){

return dfs(arr,i,j,arr.length);

}

}

}

return false;

}

public static boolean dfs(int[][] arr,int r,int c,int n){

if(r==n-1 && c==n-1){

return true;

}

if(c+1<n && arr[r][c+1]!=0){

arr[r][c]=0;

// dfs(a)

if(dfs(arr,r,c+1,n)){

return true;

}

arr[r][c]=1;

}

if(r+1<n && arr[r+1][c]!=0){

arr[r][c]=0;

if(dfs(arr,r+1,c,n)){

return true;

}

arr[r][c]=1;

}

return false;

    }

}



Given a sorted set of positive integers numset[], and a target T.

Your task is to print all subsets of numset with sum equal to given target.

If no subset found, print -1.

Note: Subsets should be distinct and lexicographiclly ascending order..

Input Format:

-------------

Line-1 -> An integer N, number of elements.

Line-2 -> N space separated integers, numset[].

Line 3 -> An integer T, target.

Output Format:

--------------

Print all the subsets.

Sample Input-1:

---------------

7

1 2 3 5 6 8 10

10

Sample Output-1:

----------------

1 3 6

2 3 5

2 8

10



import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n1=sc.nextInt();

int[] arr=new int[n1];

ArrayList<Integer> a1=new ArrayList<>();

ArrayList<Integer> a2=new ArrayList<>();

for(int i=0;i<n1;i++){

arr[i]=sc.nextInt();

a1.add(arr[i]);

}

int n=sc.nextInt();

// System.out.println(subseq(a2,a1,n));

subseq(a2,a1,n);

}

public static int add(ArrayList<Integer> arr){

int sum=0;

for(int i=0;i<arr.size();i++){

sum+=arr.get(i);

}

return sum;

}

public static void subseq(ArrayList<Integer> arr,ArrayList<Integer> uarr,int n){

if(uarr.size()==0){

if(add(arr)==n){

Collections.sort(arr);

for(int i=0;i<arr.size();i++){

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

}

System.out.println();

}

return;

}

// int num=uarr.get(0);

ArrayList<Integer> kkk=new ArrayList<>(arr);

kkk.add(uarr.get(0));

subseq(kkk,new ArrayList<>(uarr.subList(1,uarr.size())),n);

subseq(arr,new ArrayList<>(uarr.subList(1,uarr.size())),n);

}

}

