EASY LEVEL

1

Greedy Man

1. **Problem statement:**

Once upon a time, in the 22nd century, Steve went to a smartphone market.

There, a smartphone with an i index had a price of Ai bitcoins.

Some smartphones were so old that they had a negative price — the vendor would actually pay Steve just for taking them.

Steve can have any phone he wants, but his fashionable jeans have a limited number of pockets — M.

He won't be able to return to this market again because it will travel to another galaxy soon.

Help Steve earn as much money as possible while he still can.

2. **Input Format:**

The first line contains two space-delimited integers N and M — the number of devices that will fit into Steve's pockets, and the number of smartphones available for sale.

The second line contains n space-delimited integers Ai — the price of a smartphone.

3. **Output Format:**

Output a single integer number — the maximum sum of money that Steve can earn, if he can carry t M smartphones at most.

4. **Constraints:**

$1 \leq M \leq N \leq 100$

$-1000 \leq A_i \leq 1000$

5. Samples Input/Output:

	Input	Output
1	5 3 -6 0 35 -2 4	8
2	4 2 7 0 0 -7	7

6. Test Cases

	Input	Output
1	6 6 756 -611 251 -66 572 -818	1495
2	5 5 976 437 937 788 518	0
3	5 3 -2 -2 -2 -2 -2	6
4	5 1 998 997 985 937 998	0
5	2 2 -742 -187	929
6	3 3 522 597 384	0
7	4 2 -215 -620 192 647	835

7. Solution (Java):

```
import java.util.Arrays;
import java.util.Scanner;
public class SampleClass {
public static void main(String[] args) {
Scanner s = new Scanner(System.in);
int n = s.nextInt();
int m =s.nextInt();
int total = 0;
int tv[]= new int[n];
for(int i=0; i<n;i++){
tv[i]=s.nextInt();
}
Arrays.sort(tv);
for(int i=0;i<m;i++){
if(tv[i]<=0){
total+=tv[i];
}
}
System.out.println(total*(-1));
}
}
```

2

47

1. **Problem statement:**

Jerome loves numbers 4 and 7.

He can perform only two operations with them:

1) replace 4 with 7 or 7 with 4

2) swap any pair of numbers in a string

Jerome wants to know the minimum number of operations he needs to perform to convert string *a* to string *b* *with two given operations.*

2. Input Format:

The first and the second line - strings *a* and *b.*

3. Output Format:

The minimum number of operations needed to convert a into b.

4. Constraints:

A and *b are of equal length and contain only 4 and 7. (length < 105)*

5. Samples Input/Output:

	Input	Output
1	47 74	1
2	774 744	1

6. Test Cases:

	Input data	Output data
1	777 444	3
2	74747474 77777777	4

3	4744744447774474447474774 447777477744444444777447	8
4	477474777477444477747744444447774447474747777774 447774447744774477774447744777747777444447744777	14
5	7777777777 7777777774	1
6	47777777777 77777777774	1
7	444477774477444447777777474774447774444477444444 474447477747747444747477444774447747777777447	13

7. Sample solution (Java):

```
import java.util.Scanner;

public class C104 {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        String p1 = s.next(); String p2 = s.next();

        int wrong47 = 0; int wrong74 = 0;
```

```
        for(int i = 0; i < p1.length(); i++) {

            if(p1.charAt(i) == '4' && p2.charAt(i) == '7') wrong47++;

            else if(p1.charAt(i) == '7' && p2.charAt(i) == '4') wrong74++;

        }

        System.out.println(Math.max(wrong47, wrong74));

    }


}
```

3

A Bag

1. **Problem statement:**

You need to pack some toolkits into bags. Each kit has a volume from one to four liters. Each bag's capacity is four liters. One toolkit needs to be put in one bag. But one bag can contain several toolkits.

2. Input Format:

The first line contains an integer n.

The second line contains a sequence of integers, each separated by a space, that represent the volume of each toolkit.

3. Output Format:

A single number – the minimum number of bags needed to pack all the toolkits.

4. Constraints:

$1 <= n <= 10^5$

5. Samples Input/Output:

	Input	Output
1	5 1 2 4 3 3	4
2	8 2 3 4 4 2 1 3 1	5

6. Test cases:

	Input	Output
1	9 3 1 2 1 1 1 1 1 1	3
2	5 4 4 4 4 4	5
3	12 1 1 1 1 1 1 1 1 1 1 1 1	3
4	2 2 1	1
5	4 3 2 1 3	3
6	4	3

	2 4 1 3	
7	1 1	1

7. Solution (Java):

```
public static void main(String[] args) {
 int numBags = 0;
 //int kitSize = 4;
 Scanner in = new Scanner(System.in);
 int numKits = in.nextInt();
  ArrayList<Integer> kits = new ArrayList<Integer>(numKits);
  for (int i=0; i<numKits; i++) {
kits.add(in.nextInt());
  } // Sort the arraylist
  Collections.sort(kits);
  while (kits.size() > 0) {
  //get last(largest) element in arraylist
  int len = kits.size();
  int thisKitsize = kits.get(len-1);
kits.remove(len-1);
  if (thisKitSize == 4) {
numBags++;
  } else if (len >= 2 && thisKitSize == 3 && kits.get(0) == 1) {
kits.remove(0); numBags++;
  } else if (thisKitSize == 3) {
numBags++;
```

```
  } else if (len >= 2 && thisKitSize == 2 && kits.get(len-2) == 2) {
kits.remove(len-2);
numBags++;
  } else if (len >= 3 && thisKitSize == 2 && kits.get(0) == 1 && kits.get(1) == 1) {
kits.remove(1);
kits.remove(0);
numBags++;
  } else if (len >= 2 && thisKitSize == 2 && kits.get(0) == 1) {
kits.remove(0);
numBags++;
  } else if (thisKitSize == 2) {
numBags++;
  } else if (len >= 4 && thisKitSize == 1 && kits.get(0) == 1 && kits.get(1) == 1 &&
kits.get(2) == 1) {
kits.remove(2);
kits.remove(1);
kits.remove(0);
numBags++;
  } else if (len >= 3 && thisKitSize == 1 && kits.get(0) == 1 && kits.get(1) == 1) {
kits.remove(1);
kits.remove(0);
numBags++;
  } else if (len >= 2 && thisKitSize == 1 && kits.get(0) == 1) {
kits.remove(0);
numBags++;
  } else if (thisKitSize == 1) {
numBags++;
```

```
 }
}
System.out.println(numBags);
}
```

4

Add and Remove

1. **Problem statement:**

Mike has an array of n integers a1, a2, ..., an. Mike loves it when the numbers in the array are the same. That's why he wants to make sure that the array has as many identical numbers as possible. To do this, Mike runs the following operation several times:

he selects two array elements ai, aj (i ≠ j);

then, increments the number ai by 1 and decrements the number aj by one at the same time, i.e. making ai = ai + aj and aj = - 1.

This operation changes exactly two different array elements. Mike can use this operation an unlimited number of times.

Now he wants to know the maximum number of equal elements in the array that he can get by using this operation any number of times. Help Mike.

2. **Input Format:**

The first line contains an integer n - the size of the array. The second line contains integers a1, a2, ..., an - the source array.

3. **Output Format:**

Print out a single integer - the maximum number of equal elements in the array that Mike can get by performing this operation any number of times.

4. Constraints:

For first line: n (1 ≤ n ≤ 10 5)

For second line: a1, a2, ..., an (| ai | ≤ 10 ^ 4)

5. Samples Input/Output:

	Input	Output
1	2 2 1	1
2	3 1 4 1	3

6. Test Cases:

	Input data	Output data
1	4 2 -7 -2 -6	3
2	6 -1 1 0 0 -1 -1	5
3	5 0 0 0 0 0	5
4	4 2 0 -2 -1	3
5	3 2 2 4	2
6	4 1 2 3 4	3

7	**1** **1**	**1**

7. Sample solution (Java):

```
import java.util.*;
public class Solution {
public static void main(String[] args)
{
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
int t = 0;
for(int i =0;i<n;i++)t+= scan.nextInt();
if(t%n==0)
System.out.println(n);
else
System.out.println(n-1);
}
}
```

5

Adventure Time

1. **Problem statement:**

Finn wants to visit Jake at his lake house. Finn loves math and decided to only walk 1,2,3,4 or 5 kilometers each hour.

Help Finn find out how many hours he will spend on his journey.

2. Input Format:

The first line contains an integer N — the distance to Jake's lake house.

3. Output Format:

A single number – the minimum number of hours Finn has to spend walking.

4. Constraints:

$1 \leq N \leq 10^6$

5. Samples Input/Output:

	Input	Output
1	5	1
2	12	3

6. Test cases:

	Input	Output
1	999999	200000
2	41	9
3	1	1
4	21	5

5	34	7
6	199	40
7	135	27

7. Source (Java):

```
import java.util.*;

public class CF617A {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int n = in .nextInt();

System.out.println((n / 5) + (n % 5 == 0 ? 0 : 1));

}

}
```

6

Alcohol Calculator

1. **Problem statement:**

You need to write a program for a robotic bartender that calculates the alcohol content(in percent) of two drinks mixed together.

For input, you have a set of two strings, each containing an alcohol percentage and a volume in ml.

For output, you should have the alcohol content(as %) of the resulting drink.

2. Input Format:

Two lines with integer N ml and float M Vol.

3. Output Format:

SIngle number V

4. Constraints:

N (0<N<=99999)

Float M (0<=M<=100)

SIngle number V (0<=V<=100)

5. Samples Input/Output:

	Input	Output
1	**200 40** **200 20**	**30%**
2	**200 40** **200 0**	**20%**

6. Test Cases:

	Input data	Output data
1	**150 40** **200 8**	**21.714285714285715%**
2	**5000 5.7** **200 40**	**7.019230769230769%**
3	**20 70** **30 25**	**43%**
4	**0 0**	**1%**

	1 1	
5	**10 60** **0 40**	**60%**
6	**5 99** **1000 2**	**2.482587064676617%**
7	**77 2** **100 0**	**0.8700564971751412%**

7. Sample solution (Java):

```
 import java.util.*;


class Main
{
    public static void main (String[] args) throws java.lang.Exception
    {
        Scanner cin = new Scanner(System.in);

        String[] input1 = cin.nextLine().split(" ");
        String[] input2 = cin.nextLine().split(" ");

        Integer ml1 = Integer.parseInt(input1[0]);
        Integer ml2 = Integer.parseInt(input2[0]);

        Integer vol1 = Integer.parseInt(input1[1]);
        Integer vol2 = Integer.parseInt(input2[1]);

        Integer wholeMl = ml1 + ml2;

        Double perc1 = ml1 * 1.0/wholeMl;
        Double perc2 = ml2 * 1.0/wholeMl;

        Double result = perc1*vol1 + perc2*vol2;
```

```
            System.out.print(result + "%");
        }
    }
```

7

All as One

1. **Problem statement:**

You are given an array of n integers, each of them equals to 1,2 or 3. Calculate how many elements need to be changed to make all the elements in the array identical.

2. **Input Format:**

The first line contains an integer *n*. The second line contains an array of integers *a*1, *a*2, ..., *an*.

3. Output Format:

Output the minimum amount of changes necessary for equalizing all the numbers in the array.

4. Constraints:

n (1 ≤ *n* ≤ 10^6)

*a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 3)

5. Samples Input/Output:

	Input	Output
1	9 1 3 2 2 2 1 1 2 3	5
2	6 3 3 2 2 1 3	3

6. Test cases:

	Input	Output
1	12 3 1 3 1 2 1 3 2 2 1 2 1	7
2	15 3 2 1 1 1 1 3 2 2 3 3 1 2 3 2	10
3	2 2 1	1
4	2 2 1	1
5	4 2 1 2 2	1
6	5 3 3 3 3 3	0
7	30 2 1 3 2 3 2 2 2 2 3 2 2 3 2 1 1 3 1 3 2 1 2 3 1 1 3 3 1 3 1	19

7. Solution (Java):

```
import java.util.*;

import java.io.*;

public class SampleClass {
```

```
public static void main(String args[]) throws IOException {
BufferedReader lector = new BufferedReader(new InputStreamReader(System.in));
String tmp = lector.readLine();
tmp = lector.readLine();
int a[] = {
0,
0,
0
};
int sum = 0;
for (int n = 0; n < tmp.length(); n += 2) {
a[(int)(tmp.charAt(n) - '1')]++;
sum++;
}
Arrays.sort(a);
System.out.println(a[0] + a[1]);
}
}
```

8

Bacteria

1. **Problem statement:**

You're a big fan of bacteria. You want to grow it in a little box.

You start with an empty box. Every morning, you can put any number of bacteria in the box. Every night, each bacterium divides into two bacteria. One day you hope to see exactly x bacteria in the box.

What is the minimum total number of bacteria you need to put in the box in order to achieve this goal?

2. Input Format:

A single line containing one integer *x*.

3. Output Format:

A single line containing one integer: the answer.

4. Constraints:

x $(1 \leq x \leq 10^9)$

5. Samples Input/Output:

	Input	Output
1	5	2
2	8	1

6. Test Cases:

	Input	Output
1	536870911	29
2	1	1
3	343000816	14
4	697681824	14
5	41313494	14
6	11742220	19

	4	
7	999999998	20

In the first example, we can put one bacterium into the box on the first day morning, and on the third day morning there will be 4 bacteria in the box. Now we need to put one more bacterium in the box, making the total number of bacteria equal to 5. We have added a total of 2 bacteria tothe box, so the answer is 2.

In the second example, we can put one bacterium on the first day morning, then on the fourth day morning there will be 8 bacteria in the box. Thus, the answer is one.

7. Sample solution (Java):

```
import java.util.Scanner;

public class SampleClass

{

public static void main(String[] args)

{

Scanner in = new Scanner(System.in);

System.out.print(Integer.bitCount(in.nextInt()));

}

}
```

9

Bags with Apples

1. **Problem statement:**

James had two bags of apples, the first one had x (x ≥ 1), apples in it, and the second one had y (y ≥ 1) apples. James lost his first bag of apples. But he can remember that the total number of apples (x + y) in the two bags was no more thanr n, and that it was also divisible by k.

Help James determine the possible number of apples in the first bag. Output all possible values in ascending order.

2. Input Format:

The first line contains three integers, each separated by a space: *y*, *k*, *n*

3. Output Format:

Print a list of integers, each separated by aspace - all possible values of x in ascending order. Each value should only be displayed once.

If there is no suitablex value, print a single integer -1.

4. Constraints:

$(1 \leq y, k, n \leq 10^9; \frac{n}{k} \leq 10^5)$.

5. Samples Input/Output:

	Input	Output
1	10 1 10	-1
2	10 6 40	2 8 14 20 26

6. Test cases:

	Input	Output
1	10 1 20	1 2 3 4 5 6 7 8 9 10
2	84817 1 33457	-1
3	21 37 99	16 53
4	78 7 15	-1
5	74 17 27	-1
6	32 33 3	-1
7	64 59 404	54 113 172 231 290

7. Sample solution (Java):

```
import java.util.Arrays;
import java.util.Scanner;

public class apples {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int y = sc.nextInt();
        int k = sc.nextInt();
        int n = sc.nextInt();
        boolean ans=false;
        if(y<n) {
            long x = ((y/k + 1)*k) -y;
            while(x+y <=n) {
                ans=true;
                System.out.print(x + " ");
                x += k;
            }


        }
        else {
            System.out.println("-1");
            return;
        }
        if(!ans) System.out.println("-1");
    }
}
```

10

Beautiful Photo

1. **Problem statement:**

James likes to take pictures of his friends. He has noticed that his photos don't look too well when there are too many boys or girls standing in a row in a photo. So he decides that there should be no more than 6 boys or 6 girls in a row in a photo to make it a good one.

Help James determine whether a photo he took is a good one or not.

2. Input Format:

You're given a non-empty string of "0" and "1" symbols, where 0 stands for a girl, and 1 - for a boy.

3. Output Format:

Output «YES» if a photo is good. Otherwise, output «NO».

4. Constraints:

The string's length does not exceed 100 characters. There must be at least one girl and one boy in every photo.

5. Samples Input/Output:

	Input	Output
1	001001	YES
2	1000000001	NO

6. Test cases:

	Input	Output
1	00100110111111101	NO
2	11110111111111111	NO
3	01	YES
4	101010101	YES

5	011100001101001101011101001110001011010111010111101101001001111000011101111	YES
6	111101100110001001111001111011010111111101000101010110111111101110110110111	NO
7	100100010101110010001011001110100011100010011110100101100011010001001010001001101111001100	YES

7. Sample solution (Java):

```
import java.util.Scanner;

public class A96 {

    public static void main (String args[]){

        Scanner in = new Scanner(System.in);

        String t = in.next();

        if(t.contains("0000000")||t.contains("1111111"))

            System.out.println("NO");

        else

            System.out.println("YES");

    }

}
```

11

Best Friends

1. **Problem statement:**

Dima and his friends spent the whole night playing hide and seek in Dima's apartment. As a result, there's now a lot of mess in the apartment, so in the morning they decide that someone needs to clean it up.

To determine who will do the cleaning, they want to use a Counting Game: first, all the children stand in a circle; then each of them shows a number of fingers on one hand (from one to five fingers); then they start counting people circle-wise, starting from Dima, for as long as the total number of fingers everyone showed. Eventually, the countdown will stop at one person, and that person will be the one to clean the apartment.

For example, if only Dima and one friend of his played hide and seek, and during the counting game there was a total of 7 fingers shown, then Dima would have to clean the apartment. If there were two fingers shown, or, for example, eight fingers, then it would be Dima's friend who would have to clean up the apartment.

Dima knows how many fingers each of his friends will show during the counting game. Now he wants to know the number of options he has to show a certain amount of fingers on one hand(from one to five fingers) so that he would not be the one to clean the apartment. Help Dima.

2. Input Format:

The first line contains an integer n - the number of Dima's friends. Dima himself is not consider as one of his own friends. The second line contains n positive integers not exceeding 5 - the number of fingers Dima's friends will show during the counting game.

The numbers in the lines are separated by single spaces.

3. Output Format:

Output the answer to the problem on a single line.

4. Constraints:

n $(1 \leq n \leq 100)$

The second line contains n positive integers not exceeding 5

5. Samples Input/Output:

	Input	Output
1	1 1	3
2	1 2	2

6. Test Cases:

	Input	Output
1	2 3 5	3
2	1 5	3
3	7 4 1 3 2 2 4 5	4
4	15 5 5 5 3 5 4 1 3 3 4 3 4 1 4 4	5
5	8 2 2 5 3 4 3 3 2	4
6	3 3 5 1	4
7	6 4 2 3 1 3 5	4

In the first example Dima can show 1, 3 or 5 fingers. If Dima shows 3 fingers, the countdown will go as follows: Dima, friend, Dima, friend.

In thesecond example Dima can show 2 or 4 fingers.

7. Sample solution (Java):

```
import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int sum = 0;

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

sum += sc.nextInt();

}

int son = 5;

for (int i = 1; i <= 5; i++) {

if ((sum + i)%(n+1) == 1) {

son--;

}

}

System.out.println(son);

}

}
```

---

1. **Problem statement:**

Let's look at a group of friends from the Big Bang Theory series: Sheldon, Leonard, Penny, Rajesh and Howard.They are standing in line for Pepsi:

Here is the pepsi drinking algorithm they use: whoever is the first in line, drinks a Pepsi and goes back to the end of the line, adding their name to it two times. For instance, if Sheldon is the first line line, he first drinks a Pepsi then leaves the line and two Sheldons are added to the end of the line!

For example:

`SLPRH → (Sheldon drinks) → LPRHSS → (Leo drinks) → PRHSSLL → ...`

Write a program that will print the name of the character who will drink the *n*-th Pepsi.

2. Input Format:

On a single line - an integer *n*.

3. Output Format:

The name of the person (without quotes) who will drink the *n*-th Pepsi**.**

4. Constraints:

n $(1 \leq n \leq 109)$

5. Samples Input/Output:

	Input	Output
1	1	Sheldon
2	6	Sheldon

6. Test cases:

	Input	Output
1		Penny

	1802	
2	2	Leonard
3	10010	Howard
4	121580142	Penny
5	534	Rajesh
6	5033	Howard
7	4425	Rajesh

7. Sample solution (Java):

```
import java.util.Scanner;

public class A_Double_Cola {

public static void main(String[] args) {

    int n = new Scanner(System.in).nextInt()-1;

    for(;n>=5;){

        n -= 5; n >>= 1;

    }

    System.out.println(new String[]{"Sheldon", "Leonard", "Penny", "Rajesh",
"Howard"}[n]);

}
```

```
}
```

13

BinSequence

1. **Problem statement:**

Create a program that returns a sequence that has the length of entered N, consisting of digits 0 and 1. Make sure that any element of this subsequence is not repeated three or more times in a row.

2. **Input Format:**

N (a number)

3. **Output Format:**

Several numbers (a sequence)

4. **Constraints:**

N = 0 or 1

5. Samples Input/Output:

	Input	Output
1	**1**	**0**
2	**2**	**0 1**

6. Test cases:

	Input	Output
1	**3**	**0 1 1**
2	**4**	**0 1 1 0**
3	**5**	**0 1 1 0 1**
4	**10**	**0 1 1 0 1 0 0 1 1 0**

5	**20**	**0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1**
6	**8**	**0 1 1 0 1 0 0 1**
7	**19**	**0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0**

7. Sample solution (Java):

```
int[] ar = new int[N];

ar[0] = 0;

int count = 1;



 while (count < N) {

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

             if ((count + i) < N) {

                  ar[count + i] = 1 - ar[i];

             } else {

                  break;

             }

 }



count += count;

}
```

14

Bookshelf

1. Problem statement:

A student named Oleg has a shelf with N books in it. The space between each book is equal. He wants to move N-M books to another shelf, so that there are M books left remaining in the first shelf.

How many methods of removing N-M books can Oleg use, so that the space between M books in the first shelf is equal as well?

2. Input Format:

N, M

3. Output Format:

The number of methods

4. Constraints:

Input data must be positive integers numbers.

5. Samples Input/Output:

	Input	Output
1	5 3	4
2	6 2	15

6. Test cases:

	Input	Output
1	10 5	8
2	5 2	10
3	12 8	5

4	**12 2**	**66**
5	**44 42**	**3**
6	**50 5**	**288**
7	**500 100**	**1015**

7. Sample solution (Java):

```
int count = 0;

 if (M == 0) {

   count = 1;

 } else {

    if (M == 1) {

       count = N;

     } else {

       for (int d = 1; d <= (N-1) / (M-1); d++) {

             count += N - (M-1)*d;

       }

  }

}
```

15

Problem statement:

Box of Candies

1. Problem statement:

Mark came over to visit James and Jessica. He saw that they had a lot of boxes with candies. Mark thinks that if he steals a box, then James and Jessica will not notice anything. He decided that it is necessary that the total number of candies in the remaining boxes should be even, That way James and Jessica can divide the candies equally between each other (even if there are 0 candies remaining - the key point is leaving them with an even number of candies). How many ways does Mark have to steal exactly one box of candies, so that the total number of candies left in the remaining boxes is even?

2. Input Format:

The first line contains a single integer n - the number of boxes with candies James and Jessica have. The second line contains n integers ai - the number of candies in the i-th bag.

3. Output Format:

On a single line, ouput a single number - the sought number of ways. If there are no ways at all, output 0.

4. Constraints:

n $(1 \leq n \leq 100)$

a_i $(1 \leq a_i \leq 100)$

5. Samples Input/Output:

	Input data	Output data
1	1 1	1
2	10 1 2 2 3 4 4 4 2 2 2	8

6. Test cases:

	Input data	Output data

1	11 2 2 2 2 2 2 2 2 2 2 99	1
2	2 1 1	0
3	2 2 2	2
4	2 1 2	1
5	7 7 7 7 7 7 7 7	7
6	8 1 2 3 4 5 6 7 8	4
7	17 50 14 17 77 74 74 38 76 41 27 45 29 66 98 38 73 38	7

7. Sample solution (Java):

```
import java.util.Scanner;

public class Code {
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);

        int n=sc.nextInt();
        int ch=0,nch=0;

        int cur;

        for(int i=0;i<n;i++){
            cur=sc.nextInt();
```

```
            if(cur%2==0)
                ch++;
            else
                nch++;
        }

        System.out.println(nch%2==0?ch:nch);

    }
}
```

16

Problem statement:

Cat and Pillows

1. **Problem statement:**

A cat sits on one of n pillows arranged in a circle. Every k minute the cat jumps clockwise over k-1 pillows.

You should find out whether the cat will visit every pillow if it jumps infinitely.

2. Input Format:

The first line contains an integer n - the number of pillows.

3. Output Format:

Output "YES" if the cat visits every pillow, and "NO" otherwise.

4. Constraints:

`n (1 ≤ n ≤ 1000)`

5. Samples Input/Output:

	Input	Output
1	**1**	**YES**
2	**4**	**YES**

Test cases

	Input	Output
1	3	NO
2	5	NO
3	7	NO
4	9	NO
5	512	YES
6	17	NO
7	25	NO
8	64	YES

7. Sample solution (Java):

```
import java.util.Scanner;
public class SampleClass {
public static void main(String[] args) {
Scanner in = new Scanner(System.in);
int b = in .nextInt();
boolean s = true;
for (int i = 0; i < 10; i++)
if ((Math.pow(2, i)) == b) {
System.out.println("YES");
s = false;
break;
}
if (s)
System.out.println("NO");
}
```

}

17

Cat Food

1. Problem statement:

There are *n* cats numbered in clockwise order sitting in a circle: the cat number 2 sits to the left of the cat number 1, the cat number 3 sits to the left of the cat number 2, ..., the cat number 1 sits to the left of the cat number *n*.

There's also a dog that has *m* fish. The dog begins to give fish to the cats starting from the cat number 1 and moving on clockwise. The cat number *i* gets *i* fishes. If the dog can't give the current cat the required amount of fish, then the dog takes the remaining fish for itself. Given *n* and *m,* find out how many fish the dog will get at the end.

2. Input Format:

The first line contains two integers *n* and *m* — the number of cats sitting in a circle and the number of fish.

3. Output Format:

Print the number of fish the dog will get at the end.

4. Constraints:

$1 \le n \le 50, 1 \le m \le 10^4$

5. Samples Input/Output:

	Input	Output
1	4 11	0
2	17 107	2

6. Test cases

	Input	Output
1	3 8	1

2	46 7262	35
3	19 300	5
4	100 100	9
5	9 7601	5
6	1 9058	0
7	45 9465	14
8	32 6864	0

7. Sample solution (Java):

```
import java.util.Scanner;
public class Task92A {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
int m = sc.nextInt();
sc.close();
int i = 0;
while (m > 0) {
i = (i % n) + 1;
m -= i;
}
if (m == 0) {
System.out.println(0);
} else {
```

```
System.out.println(m + i);
}
}
}
```

EXTRA EASY LEVEL 2

1

Banner cycle

1. **Problem statement:**

Print out all elements of the given list in random order, without repeating any of the elements.

2. Input Format:

Generate an ordered list of integers from 0 to 100 within the program

3. Output Format:

A random sample of numbers from an ordered array, without repeating any elements.

4. Constraints:

Integers from 0 to 100

5. Samples Input/Output:

	input	output
1	1 2 3 4 5 6 7 8 9 10	10, 2, 4, 1, 3 9, 7, 6, 8, 5
2	1 2 3 4 5 6 7 8 9 10	5, 9, 3, 10, 2 4, 1, 6, 8, 7

6. Test Cases:

	input	output
1	1 2 3 4 5 6 7 8 9 10	6, 8, 4, 1, 7 3, 10, 5, 9, 2
2	1 2 3 4 5 6 7 8 9 10	8, 4, 1, 6, 7 5, 9, 3, 10, 2
3	1 2 3 4 5 6 7 8 9 10	8, 7, 4, 1, 6, 10, 2, 5, 9, 3
4	1 2 3 4 5 6 7 8 9 10	2, 1, 3 ,5, 4 7, 6, 8, 10, 9
5	1 2 3 4 5 6 7 8 9 10	7, 6, 8, 10, 9 2, 1, 3 ,5, 4
6	1 2 3 4 5	5, 9, 3, 10, 2

	6 7 8 9 10	8, 4, 1, 6, 7
7	1 2 3 4 5 6 7 8 9 10	3, 10, 5, 9, 2 6, 8, 4, 1, 7
8	1 2 3 4 5 6 7 8 9 10	4, 1, 6, 8, 7 5, 9, 3, 10, 2

7. Sample solution (Java):

```
import java.util.Random;

class Banner

{

    public static void main(String[] args)

    {

        Random r = new Random();

        int[] list = new int[100];

        for (int i = 0; i < 100; ++i)

        {

            list[i] = i;

        }
```

```
        int index = 99;

        int number;

        while (index > 0)

        {

            number = r.nextInt(index);

            System.out.println(list[number]);

            list[number] = list[index];

            --index;

        }

    }

}
```

2

Max Even Sum

1. **Problem statement:**

Today giraffe Guffy received N integers. He wants to choose some of them to get the maximum even sum (that is divisible by 2) possible. Please help Guffy do that.

If Guffy does not choose any number, the sum will be 0.

2. Input Format:

The first line contains the number of integers *N*

3. Output Format:

The maximum even sum possible. Each number can only be used once.

4. Constraints:

$N = 1 \leq N \leq 100\ 000$

5. Samples Input/Output:

	Input	Output
1	3 1 2 3	6
2	5 999999999 999999999 999999999 999999999 999999999	3999999996

6. Test Cases:

	Input	Output
1	1 1	0
2	15 39 52 88 78 46 95 84 98 55 3 68 42 6 18 98	870
3	15 59 96 34 48 8 72 67 90 15 85 7 90 97 47 25	840

4	15 87 37 91 29 58 45 51 74 70 71 47 38 91 89 44	922
5	15 11 81 49 7 11 14 30 67 29 50 90 81 77 18 59	674
6	15 39 21 95 89 73 90 9 55 85 32 30 21 68 59 82	848
7	15 87 22 98 32 88 36 72 31 100 97 17 16 60 22 20	798

7. Sample solution (Java):

```
import java.util.Scanner;



public class wetsharkA {



    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int runs = scan.nextInt();

        long sum = 0;

        int minOdd = Integer.MAX_VALUE;
```

```
        for(int i = 0; i < runs; i++) {

            int a = scan.nextInt();

            if(a%2 != 0) minOdd = Math.min(minOdd, a);

            sum += a;

        }

        if(sum%2 != 0) {

            sum -= minOdd;

        }

        System.out.println(sum);

    }

}
```

3

Nearly Prime

1. **Problem statement:**

A nearly prime number is a number that has two different prime divisors. For example, numbers 6, 18, 24 are nearly prime, and 4, 8, 9, 42 are not. Find the amount of nearly prime numbers from 1 to N inclusive.

2. Input Format:

N $(1 \leq N \leq 3000)$

3. Output Format:

The amount of almost prime numbers from 1 to N

4. Constraints

N $(1 \leq N \leq 3000)$

5. Samples Input/Output:

	Input	Output
1	10	**2**
2	21	**8**

6. Test Cases:

	Input	Output
1	1	**0**
2	2	**0**
3	4	**0**
4	19	**6**
5	40	**19**
6	77	41

7	222	125

7. Sample solution (Java):

```
import java.util.Scanner;

public class A_26_Almost_Prime {
    public static void main(String[] args){
        Scanner input=new Scanner(System.in);
        int n=input.nextInt();
        int[] number=new int[n+1];
        int i,j,count=0;

        for(i=2;i<=n;i++){
            if(number[i]==0){
                for(j=1;j*i<=n;j++)
                    number[j*i]++;
            }
            else if(number[i]==2)
                count++;
        }
        System.out.println(count);
    }
}
```

4

Ronaldo’s Bricks

1. **Problem statement:**

Ronaldo got bricks for his birthday present. He decided to build a pyramid with them: at the top of the pyramid there should be 1 brick, on the second level 1 + 2 = 3 bricks, the third should have 1 + 2 + 3 = 6 bricks, etc.

Find out the height of this pyramid made of N bricks.

2. Input Format:

N - the number of bricks.

3. Output Format:

The pyramid's height.

4. Constraints:

$N\ (1 \le N \le 104)$

5. Samples Input/Output:

	Input	Output
1	1	1
2	25	4

6. Test Cases:

	Input	Output
1	2	1
2	4115	28
3	9894	38
4	7969	35
5	6560	33
6	4	2

7	3	1

7. Sample solution (Java):

```
import java.util.*;
public class vanyacubes
{
public static void main(String args[])
{
Scanner s= new Scanner (System.in);
int a=s.nextInt();
int ctr=0;
for(int i=1;;i++)
{
ctr+= i*(i+1) / 2 ;
if(ctr>a)
{System.out.print(i-1); return;}
}
}
}
```

5

Sailor and Beer

1. **Problem statement:**

A drunk sailor wants to buy N bottles of beer. He has to pay K dollars for the first bottle, 2*K dollars for the second bottle, 3*K dollars for the third bottle, etc. (i*K dollars for every i bottle).

He only has P dollars. How much money should he borrow from his friend to buy N bottles of beer?

2. Input Format:

There are three positive integers on the first line of input: K - the cost of the first bottle, P - the amount of money the sailor has, and N - number of bottles he wants to buy.

3. **Output Format:**

A single positive integer, how much money the sailor should borrow from his friend. If there is no need to borrow anything, print out 0.

4. Constraints:

K (K>=1)

P (0<=P<=10^9)

N (N<=1000)

5. Samples Input/Output:

	Input data	Output data
1	3 17 4	13
2	1 2 1	0

6. Test Cases:

	Input data	Output data
1	1 1 1	0
2	1 5 6	16
3	1 1000000000 1	0
4	1000 0 1000	500500000
5	859 453892 543	126416972
6	1000 500500000 1000	0
7	1000 500499999 1000	1

7. Sample solution (Java):

```
import java.util.*;
public class Main {
public static void main(String [] args){
    Scanner in=new Scanner(System.in);
    int k=in.nextInt();
    int n=in.nextInt();
    int w=in.nextInt();
    System.out.print(Math.max(0,w*(w+1)*k/2 - n));
}
}
```

6

Vowels and Consonants

1. **Problem statement:**

Your program is supposed to delete all vowels from a given string, insert a dot (.) before each consonant and convert them to lower case.

(A, O, Y, E, U, I are vowels, the other letters are consonants.)

2. Input Format:

A single string.

3. Output Format:

The resulting string.

4. Constraints:

Letters of the English alphabet.

5. Samples Input/Output:

	Input	Output
1	asdfghjkl	.s.d.f.g.h.j.k.l
2	consonant	.c.n.s.n.n.t

6. Test cases

	Input	Output
1	Query	.q.r
2	qwerty	.q.w.r.t
3	SAMPLE	.s.m.p.l
4	source	.s.r.c
5	typeWriter	.t.p.w.r.t.r
6	water	.w.t.r
7	zxcvbnm	.z.x.c.v.b.n.m

7. Sample solution (Java):

```
public static void main(String[] args) {
  Scanner I = new Scanner(System.in);
 String x[] = (I.nextLine().toLowerCase()).split("[aouiye ]");
 for (String b : x) {
 for(char s:b.toCharArray()) {
  System.out.print("." + s);
}
 }
}
```

Walter Junior

1. **Problem statement:**

Walter Junior really likes to have fun with chemistry. He has A blue, B purple and C orange substances. By mixing 2 substances of the same color, he can get 1 substance of any other color. Teacher has asked him to bring X blue, Y purple and Z orange substances to the class. Will he be able to get them (possibly in a few steps)?

2. Input Formatt:

The first line contains A, B and C - the number of blue, purple and orange substances that Walter has.

The second line contains X, Y and Z - the number of blue, purple and orange substances that Walter needs to get.

3. Output Format:

If Walter can get required number of substances, print out YES, otherwise print out NO.

4. Constraints:

The first line contains A, B and C ($0 \le A, B, C \le 1\ 000\ 000$)

The second line contains X, Y and Z ($0 \le X, Y, Z \le 1\ 000\ 000$)

5. Samples Input/Output:

	Input data	Output data
1	4 4 0 2 1 2	YES
2	5 6 1 2 7 2	NO

6. Test Cases:

	Input data	Output data

1	3 3 3 2 2 2	YES
2	0 0 0 0 0 0	YES
3	0 0 0 0 0 1	NO
4	0 1 0 0 0 0	YES
5	1 0 0 1 0 0	YES
6	2 2 1 1 1 2	NO
7	1 3 1 2 1 1	YES

7. Sample solution (Java):

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

    /**
     * @param args
     */
    public static void main(String[] args) {

        Scanner input=new Scanner(System.in);
```

```
        int a1=input.nextInt();
        int b1=input.nextInt();
        int c1=input.nextInt();
        int a2=input.nextInt();
        int b2=input.nextInt();
        int c2=input.nextInt();

        if((m(a1,a2)+m(b1,b2)+m(c1,c2))>=0)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
    public static int m(int a,int b){
        int k=0;
        if(a>=b)
            k=(a-b)/2;
        else
            k=a-b;
        return k;

    }
}
```

8

Weight of Number

1. **Problem statement:**

The reflection of a positive number N is Z(n), it is calculated by replacing each digit A in N with 9 - A. For example, the reflection of 146 is 853. Note that the leading zeros (if any) must be discarded. Thus, the reflection of 9 is 0, the reflection of 921 is 78.

The weight of a number is the product of its reflection and the number itself. Thus, the weight of 56 is 56 × 43 = 2408.

Your task is to find the maximum weight among the numbers in the given interval [L, R] (boundaries included).

2. Input Format:

L, R are intervals

3. Output Format:

The maximum weight

4. Constraints:

$(1 \leq L \leq R \leq 1000000000)$

5. Samples Input/Output:

	Input	Output
1	3 7	20
2	1 1	8

6. Test Cases:

	Input	Output
1	8 10	890
2	4 6	20
3	10 100	89900
4	1 999	249500
5	40 60	2450

6	66 74	2178
7	27 71	2450

7. Sample solution (Java):

```
import java.util.*;

public class R {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        long l = in.nextLong(), m = in.nextLong(), mx = 1, p;
        while (mx <= m)
            mx *= 10;
        p = mx / 2;
        p = Math.max(p, l);
        p = Math.min(p, m);
        System.out.print(p * (mx - 1 - p));
    }
}
```

9

XOR sum

1. **Problem statement:**

Finn has given an array with non-negative integers that Jake has created for for him. Help Finn find a segment with the maximum xor.

2. Input Format:

The first line contains an integer *n* — the number of elements in the array. The second line contains the integers from the array, each less than 2^{30}.

3. **Output Format:**

Find a segment with maximum xor of its elements in the given array and print out its xor sum.

4. Constraints:

The first line contains an integer n $(1 \le n \le 100)$

The second line contains the integers from the array, each less than 2^{30}

5. Samples Input/Output:

	Input	Output
1	5 1 2 1 1 2	3
2	3 1 2 7	7

6. Test Cases:

	Input	Output
1	4 4 2 4 8	14
2	5 1 1 1 1 1	1
3	2 7 1	7
4	20 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10	15

5	2 1 1	1
6	2 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	15
7	4 1 2 7 1	7
8	16 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	15

7. Sample solution (Java):

```
import java.util.Scanner;


public class SampleClass {

public static void main(String[] Args) {

Scanner scan = new Scanner(System.in);

int x = scan.nextInt();

int[][] arr = new int[x + 1][2];

arr[1][0] = 0;


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

arr[i + 1][0] = scan.nextInt();

arr[i + 1][1] = arr[i][1] ^ arr[i + 1][0];

}


int max = 0;

for (int i = 0; i < x + 1; i++)
```

```
for (int j = 0; j < i; j++)

max = Math.max(max, (arr[i][1]) ^ (arr[j][1]));


System.out.println(max);

}

}
```

10

A and B

1. **Problem statement:**

A string with the length of N consists of symbols "a" and "b". If we choose any two neighbouring symbols in the string, and one of them is "a" and the other one is "b", you can delete them both and the string's length will become N-2. Find the minimum length of the string after such operations are applied to it.

2. **Input Format:**

The first line contains N - the length of the string

Second line contains a string with the length of N

3. **Output Format:**

The length of the string after applying the operations

4. **Constraints:**

$\cap\ (1 \leq N \leq 200000)$

Second line contains a string with the length of N

5. **Samples Input/Output:**

	Input	Output
1	4 bbaa	0
2	5 ababa	1

6. Test Cases:

	Input	Output
1	8 bbbabbbb	6
2	1 a	1
3	1 b	1
4	2	2

	aa	
5	2 ab	0
6	3 aab	1
7	6 bbabba	2

7. Sample solution (Java):

```
import java.util.Scanner;

public class zero1 {
public static void main(String str[]){
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt();
    String s=sc.next();
    int zero=s.replace("a","").length();
    int one=n-zero;
    System.out.println(Math.abs(zero-one));
}
}
```

11

Adventure Breakfast

1. **Problem statement**

Jake is having breakfast. He has a piece of bread the size of *n x m* centimeters.

He wants to cover that piece of bread with slices of cheese that are *c x c* centimeters big.

What is the minimum number of slices of cheese he needs to cover whole piece of bread? The cheese can cover more area and hang down from the edges of the bread, but its sides should be parallel to the sides of the bread.

2. Input Format:

The input contains three positive integer numbers in the first line: *n*, *m* and *c*.

3. Output Format:

Output the number of slices of cheese needed to cover the whole piece of bread.

4. Constraints:

N, *m* and *c* (1 ≤ *n*, *m*, *c* ≤ 10^9).

5. Samples Input/Output:

	Input	Output
1	6 6 4	4
2	1 1 1	1

6. Test Cases:

	Input	Output
1	2 1 1	2
2	1 2 1	2

3	1 1 3	1
4	12 13 4	12
5	222 332 5	3015
6	1001 1000 10	10100
7	1000000000 1000000000 1000000000	1
8	39916800 134217728 40320	3295710

7. Sample solution (Java):

```
import java.util.Scanner;

public class SampleClass {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

long n = in .nextLong(), m = in .nextLong(), w = in .nextLong();

System.out.println((n % w == 0 ? n / w : n / w + 1) * (m % w == 0 ? m / w : m / w + 1));

}

}
```

12

Anniversary

1. **Problem statement:**

Wallmart had an anniversary and was giving all sorts of goodies and discounts to their customers.

Each k customer received a free bottle of milk, l - a kilogram of bacon, m - one 10$ coupon, n - a 99% discount for everything they bought.

There were was a d number of customers that day, find out how many of them received special treatment.

2. Input Format:

Five lines of input will contain one integer each, in the following order: *k*, *l*, *m*, *n* and *d.*

3. Output Format:

Output the number of customers who received special treatment that day.

4. Constraints:

K, *l*, *m*, *n* and *d (1 ≤ k, l, m, n ≤ 10, 1 ≤ d ≤ 10^5)*

5. Samples Input/Output:

	Input	Output
1	1 2 3 4 12	12
2	2 3 4 5 24	17

6. Test Cases:

	Input	Output

1	1 1 1 1 100000	100000
2	10 9 8 7 6	0
3	2 9 8 1 75083	75083
4	6 1 7 2 62982	62982
5	2 7 4 9 56937	35246
6	2 9 8 1 12563	12563

7	4 4 2 3 54481	36320
8	6 4 9 8 72628	28244

7. Sample Solution (Java):

```
import java.util.Scanner;

public class SampleClass {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

int k = s.nextInt();

int l = s.nextInt();

int m = s.nextInt();

int n = s.nextInt();

int d = s.nextInt();

int total = 0;

for (int x = 1; x <= d; x++) {

if (x % k == 0 || x % l == 0 || x % m == 0 || x % n == 0)

total++;

}
```

```
System.out.println(total);
}
}
```

13

Arny's Workout

1. **Problem statement:**

Arny Schwartzweiger is a bodybuilder. His coach gave him a training program for today, which contains *n* numbers $a_1...a_n$. That means that Arny needs to do *n* exercises, a_i times for every *i*-th exercise.

Arny does only three types of exercises - for his chest muscles, his biceps and his back. His training is cyclic: he always begins with his chest, then the biceps, then his back, and then the chest again, and so on to the *n*-th exercise.

Help Arny find out which group of muscles will be exercised the most during his training. We know that the most exercised group of muscles is the one with the maximum number of repeated exercises.

2. Input Format:

The first line contains a single positive integer number *n* , the number of exercises.

The second line contains *n* positive integer numbers a_i the number of repeats for the exercises.

3. Output Format:

Print out "chest" if the chest will get the most exercise, or "biceps" if the biceps will get the most exercise, or "back" if the back will get the most exercise (every word should be printed without quotes).

It is guaranteed that the input provided will imply an unambiguous output.

4. Constraints:

n (1<= *n* <= 20)

a_i (1<= a_i <= 25

5. Samples Input/Output:

	Input	Output
1	2 2 8	biceps
2	3 5 1 10	back

6. Test Cases:

	Input	Output
1	7 3 3 2 7 9 6 8	chest
2	4 5 6 6 2	chest
3	5 8 2 2 6 3	chest
4	10 4 9 8 5 3 8 8 10 4 2	biceps
5	14 13 14 19 8 5 17 9 16 15 9 5 6 3 7	back
6	13 24 10 5 7 16 17 2 7 9 20 15 2 24	chest
7	16 12 6 18 6 25 7 3 1 1 17 25 17 6 8 17 8	biceps

7. Sample solution (Java):

```
import  java.util.*;
public  class   Main{
    public  static  void    main(String[]args){
        Scanner cin=new Scanner(System.in);
        String[]Q={"chest","biceps","back"};
        for(int n,i,S[]=new
int[3];cin.hasNextInt();System.out.println(Q[n])){

for(n=cin.nextInt(),Arrays.fill(S,i=0);n>i;S[i++%3]+=cin.nextInt());
            for(n=i=2;0<i;n=S[n]<S[--i]?i:n);
        }
    }
}
```

14

Apples

1. **Problem statement:**

Help Vasya determine if it is possible to divide N apples among his two friends, so that each friend will have the same total weight of apples.

2. Input Format:

The first line contains *N*, the amount of apples

The second lines contains w_1, w_2, ..., w_n - the weight of each apple

3. Output Format:

YES - if it is possible, NO - if not.

4. Constraints:

$N\ (1 \le n \le 100)$

w_1, w_2, ..., w_n ($w_i = 100$ or $w_i = 200$)

5. Samples Input/Output:

	Input data	Output data
1	3 100 200 100	YES
2	4 100 100 100 200	NO

6. Test Cases:

	Input data	Output data
1	1 100	NO
2	2 100 100	YES
3	2	NO

	100 200	
4	3 100 100 100	NO
5	4 100 100 100 100	YES
6	3 200 100 200	NO
7	9 100 100 100 200 100 100 200 100 200	YES

7. Sample solution (Java):

```
import java.util.*;

public class _433A_Kitahara_Haruki_s_Gift {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] a = new int[3];
        String ans = "NO";
        for (int i = 0; i < n; i++)
            a[in.nextInt() / 100]++;

        if (a[1] % 2 == 0 && !(a[1] == 0 && a[2] % 2 == 1))
```

```
            ans = "YES";
        System.out.println(ans);
    }
}
```

15

Artbook

1. **Problem statement:**

Otaco Mocomoto sells his artbooks. He has 26 paintings, each marked with a character from the English alphabet (from "a" to "z"). He made an artbook with these paintings in a certain order, so the artbook can be described as a string (any painting can occur in the artbook multiple times).

Now he want to make a special edition of his artbook with one extra photo put in a random place in the book. He wants to make as many different artbooks as possible to make more money. How many different artbooks can he make?

2. Input Format:

The input data contains a single string *s* of English characters, with the length from 1 to 20 symbols - the book description with Otaco's marks.

3. Output Format:

Print out a single integer, the maximum number of different artbooks with an extra-photo in a random position that Otaco can make.

4. Constraints:

S - English characters, with the length from 1 to 20 symbols.

5. Samples Input/Output:

	Input	Output
1	a	51

2	hi	76

	Input	Output
1	y	51
2	nfiu	126
3	zoabpyuvus	276
4	spyemhyznjaeyhhbk	451
5	xulsyfkuizjauadjjopu	526
6	sdf	101
7	jselp	151

7. Sample solution (Java):

```
import java.util.*;
public class Main {
public static void main(String[] args) {
System.out.println((new Scanner(System.in).next().length()+1)*25+1);
}}
```

---

1. **Problem statement:**

Alejandro Broogman wants to be very responsible in 2016, so he began to save money. He considers two ways to save money: save 1$ every chosen day of the week, or save 1$ every chosen day of the month.

Count the total amount money Alejadro will save in 2016 by following the plan he chooses.

Notice: Alejandro uses the Gregorian calendar.

2. Input Format:

There is a single string with two possible formats (without quotes):

"***x*** *of week*".

"***x*** *of month*".

3. Output Format:

Print out the total amount of dollars that Broogman will save in 2016.

4. Constraints:

x (1<=*x*<=7) is the day of the week (1 is Monday, 7 is Sunday)

x (1<=*x*<=31) is the day of the month

5. Samples Input/Output:

	Input	Output
1	4 of week	52
2	30 of month	11

6. Test Cases:

	Input	Output

1	17 of month	12
2	31 of month	7
3	6 of week	53
4	1 of week	52
5	2 of week	52
6	2 of month	12
7	28 of month	12

7. Sample solution (Java):

```
import java.util.*;
public class hello {
    public static void main(String [] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        String s = scan.next();
        s = scan.next();
        int ans = 0;
        if (s.equals("month")) {
            if (n <= 29)
                ans = 12;
            else if (n == 30)
                ans = 11;
            else
                ans = 7;
        } else {
            if (n == 5 || n == 6)
                ans = 53;
            else
                ans = 52;
        }
```

```
        System.out.println(ans);
    }
}
```

17

Capitalizer

1. **Problem statement:**

Given a string str, convert the first character from lowercase to uppercase. Other characters should remain the same.

2. Input Format:

The first line contains a single string str with lower and uppercase characters.

3. Output Format:

Output str with the first character in the uppercase format. If str's first character is already in the uppercase format, just output str.

4. Constraints:

Str - single string str with lower and uppercase characters.

5. Samples Input/Output:

	Input	Output
1	samsung	Samsung
2	zEaLOT	ZEaLOT

6. Test Cases:

	Input	Output
1	perfecT	PerfecT
2	Keyboard	Keyboard
3	CheesE	CheesE
4	aWesome	AWesome
5	reDDIT	ReDDIT
6	GooGLE	GooGLE
7	NewYork	NewYork
8	Ribbon	Ribbon

7. Sample solution (Java):

```
import java.util.*;

public class SampleClass {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

String a = scan.next();

System.out.println(a.toUpperCase().charAt(0) + a.substring(1, a.length()));

}

}
```

HARD LEVEL 2

1

Quiz

1. **Problem statement:**

James is taking a quiz. The quiz consists of n consecutive questions. A correct answer to a question is rewarded with 1 point. The game also has a count of consecutive correct answers. When a player gives the right answer to a question, the count is increased by 1. If a player gives the wrong answer to a question, then the counter is reset back to 0. If after an answer the count reaches a number k, then it is reset back to zero and the player's points are doubled. Note that in such case 1 point is first added to the player's score, and then the total score is doubled. At the beginning of the game a player's score and the number in the count of consecutive answers both equal zero.

James remembers that he gave the right answer to exactly m questions, but does not remember the order in which the questions followed. He's trying to figure out what his minimum score might be. Help him out and output the remainder of the that number after it is divided by 1000000009 (10^9 + 9).

2. Input Format:

The line contains three space-delimited integers n, m and k .

3. Output Format:

Print out a single number - the remainder of dividing the minimum possible score that James might have by 1000000009 (10^9 + 9).

4. Constraints:

$2 \le k \le n \le 10^9, 0 \le m \le n$

5. Samples Input/Output:

	Input data	Output data
1	5 3 2	3
2	5 4 2	6

Example 1. James answered 3 questions, and for every two consecutive correct answers he doubled his score. If James answered the first, third and fifth questions, he would have the total score of 3.

Example 2. Now James has already answered 4 questions. He gets the minimum score only if he gave the wrong answer to the fourth question.

6. Test Cases:

	Input data	Output data
1	300 300 3	17717644
2	300 282 7	234881124
3	1000000000 1000000000 1000000000	999999991
4	1000000000 800000000 2	785468433
5	2 0 2	0

6	2 1 2	1
7	2 2 2	4

7. Sample solution (Java):

```
import java.util.*;
public class Main {
static long mod = 1000000009L;
public static long modPow(int a){
if(a==0)
return 1;
long temp=modPow(a/2);
temp=(temp*temp)%mod;
if(a%2==1)
temp=2*temp%mod;
return temp;
}
public static void main(String [] args){
Scanner in=new Scanner(System.in);
int n=in.nextInt();
int m=in.nextInt();
int k=in.nextInt();
if(m + n/k <= n){
System.out.print(m);
```

```
return;
}
int full = n/k - (n - m);
long ans = ((modPow(full+1) - 2 + mod)%mod)*k%mod;
ans += m - k*full;
System.out.print(ans%mod);
}
}
```

2

Concealing Handwriting

1. **Problem statement:**

A spy needs to send a letter, but he must not reveal his handwriting. He has a message (string s) and a news article (string t), so he can cut letters out from the latter.

Help him figure out how many letters he can cut out in the right case and how many letters will be in the wrong case (e. g. caps instead of lowercase).

2. Input Format:

A string (*s*) that the spy needs to send in a letter,

A string (*t*) that is printed in the news article.

3. Output Format:

Space-delimited numbers: the number of letters that have the right case; the number of letters that have the wrong case.

4. Constraints:

$1 \le |s| \le 2 \cdot 10^5$

$|s| \le |t| \le 2 \cdot 10^5$

5. Samples Input/Output:

	Input	Output
1	zzzZZZ ZZZzzZ	5 1
2	l FPbAVjsMpPDTLkfwNYFmBDHPTDSWSOUlrBHYJHPM	1 0

6. Test cases

	Input	Output
1	ncMeXssLHS uwyeMcaFatpInZVdEYpwJQSnVxLK	6 1
2	abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ	0 26
3	abcdefghijklmnopqrstuvwxyz qwertyuiopasdfghjklzxcvbnm	26 0
4	ncMeXssLHS uwyeMcaFatpInZVdEYpwJQSnVxLK	6 1
5	message mess age	7 0
6	quality	7 0

	Qualifiers quantity	
7	Hello world Oh, world is hell.	9 1

7. Solution (Java)

```
public static void main(String[] args) {
 Scanner in = new Scanner(System.in);
 int y = 0, w = 0;
 int[] snt = new int[256];
 int[] tnt = new int[256];
 char[] s = in.nextLine().toCharArray();
 for (int i = 0; i < s.length; i++)
snt[s[i]]++;
 char[] t = in.nextLine().toCharArray();
 for (int i = 0; i < t.length; i++)
tnt[t[i]]++;
 for (char i = 'a'; i <= 'z'; i++) {
 char u = Character.toUpperCase(i);
 int difl = tnt[i] - snt[i];
 int difu = tnt[u] - snt[u];
 if (difl >= 0) {
y += snt[i];
 } else {
y += tnt[i];
 } if (difu >= 0) {
y += snt[u];
```

```
 } else {
y += tnt[u];
 } if (difl < 0 && difu > 0) {
w += Math.min(-difl, difu);
 } if (difu < 0 && difl > 0) {
w += Math.min(-difu, difl);
 }
 }
 System.out.println(y + " " + w);
 in.close();
}
```

3

Trains

1. **Problem statement:**

There is a country with n cities numbered from 1 to n. City 1 is the capital of this country. Also, there are m roads connecting the cities. The i-th road connects the cities of ui and vi, the length of the road is equal to xi. Moreover, there are k railways. The i-th railway connects the capital and the city of si, the length of this railway is yi. All roads and railways are bidirectional.

Some railways need to be closed. Count the maximum number of railways that can be closed with the following condition: the the length of the shortest path from each city to the capital must not change.

2. Input Format:

The first line contains three integers: n, m, k .

Each of the following m lines contains three integers: ui, vi, xi.

Each of these k rows contains two integers - si and yi.

It is guaranteed that there exists at least one path from every city to the capital. Note that there can be several roads connecting two cities. There can also be several railways leading from one city to the capital.

3. Output Format:

Print out a single integer representing the maximum number of railways that can be closed.

4. Constraints:

$2 \le n \le 10^5$, $1 \le m \le 3 \times 10^5$, $1 \le k \le 10^5$

$1 \le u_i, v_i \le n$; $u_i \ne v_i$; $1 \le x_i \le 10^9$

$2 \le s_i \le n$; $1 \le y_i \le 10^9$

5. Samples Input/Output:

	Input data	Output data
1	5 5 3 1 2 1 2 3 2 1 3 3 3 4 4 1 5 5 3 5 4 5 5 5	2
2	2 2 3 1 2 2 2 1 3	2

	2 1 2 2 2 3	

6. Test Cases:

	Input data	Output data
1	5 4 3 1 2 999999999 2 3 1000000000 3 4 529529529 5 1 524524524 5 524444444 5 529999999 2 1000000000	2
2	3 2 5 1 2 2 2 3 4 3 5 3 5 3 5 3 6 3 7	4
3		2

	5 5 3 1 2 999999999 2 3 1000000000 3 4 529529529 5 1 524524524 5 3 1000000000 5 524444444 5 529999999 2 1000000000	
4	2 1 5 1 2 4 2 3 2 5 2 4 2 4 2 5	4
5	3 3 6 1 2 499999999 2 3 500000000 1 3 999999999 2 499999999 2 500000000 2 499999999 3 999999999 3 1000000000 3 1000000000	6
6	2 1 1 1 2 1 2 1000000000	1

7	3 2 2 1 2 4 2 3 4 2 2 3 6	1

7. Sample solution (Java):

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;



public class Main {
static int n,m,k;
static ArrayList<node>[] adj ;
static boolean[] v;
static long[] dist;
static boolean[] parents ;

static class node{
int x;
boolean train ;
long w;
public node(int a,long w,boolean t){
```

```
x=a;this.w=w;train=t;
}
}
static class comp implements Comparator<node>{
public int compare(node n1, node n2) {
if(n1.w<n2.w)
return -1;
else if(n1.w>n2.w)
return 1;
return 0;
}
}
static Comparator<node> compa = new comp();
static PriorityQueue<node> q ;

static void dijkstra(){
dist[1]=0;
parents[1]=false;
q.add(new node(1,0,false));

while(!q.isEmpty()){
node n = q.poll();
if(!v[n.x]){
v[n.x]=true;
for(int i=0;i<adj[n.x].size();i++){
node curr = adj[n.x].get(i);
if(dist[n.x]+curr.w<dist[curr.x]){
dist[curr.x] = curr.w+dist[n.x];
```

```
parents[curr.x]=curr.train;
q.add(new node(curr.x,dist[curr.x],curr.train));
}
else if(dist[n.x]+curr.w==dist[curr.x] && parents[curr.x] && !curr.train){
parents[curr.x]=false;
q.add(new node(curr.x,dist[curr.x],false));
}
}
}
}
}
static public void main(String[] args)throws Exception{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String[]in = br.readLine().split(" ");
n = Integer.parseInt(in[0]);
m = Integer.parseInt(in[1]);
k = Integer.parseInt(in[2]);
v = new boolean[n+10];
dist = new long[n+10];
adj = new ArrayList[n+10];
parents = new boolean[n+10];
q = new PriorityQueue<node>(2*n,compa);
for(int i=0;i<n+10;i++){
adj[i] = new ArrayList<node>();
dist[i]= Long.MAX_VALUE;
}
for(int i=0;i<m;i++){
in = br.readLine().split(" ");
```

```
int from = Integer.parseInt(in[0]), to = Integer.parseInt(in[1]) , w = Integer.parseInt(in[2]);
adj[from].add(new node(to,w,false));
adj[to].add(new node(from,w,false));
}
for(int i=0;i<k;i++){
in = br.readLine().split(" ");
int to = Integer.parseInt(in[0]), w = Integer.parseInt(in[1]);
adj[1].add(new node(to,w,true));
adj[to].add(new node(1,w,true));
}

dijkstra();
int res = 0;
for(int i=0;i<parents.length;i++){
if(parents[i])
res++;
}
System.out.println(k-res);
}
}
```

4

Arrays Minmax

1. **Problem statement:**

There are two arrays - A and B. You can perform operations on these arrays. With one operation you can either decrease or increase any element of any array by 1. Note that the operation can be used several times for each element of any array.

Your task is to to find the lesser number of operations required to meet the following condition: the minimum element of array A should be no less than the maximum element of array B.

2. Input Format:

The first line contains two space-delimited integers *n*, *m*. The second line will contain *n* space-delimited integers representing the contents of array *A*. The third line contains *m* space-delimited integers representing the contents of array *B*.

3. Output Format:

Print out an integer representing the minimum number of operations.

4. Constraints:

$1 \leq n, m \leq 10^5$

$1 \leq ai \leq 10^9$

$1 \leq bi \leq 10^9$

5. Samples Input/Output:

	Input data	Output data
1	2 2 2 3 3 5	3
2	3 2 1 2 3 3 4	4

In the first test case A1 can be increased by 1 and B2 reduced by 1, and then reduced again to 1. Now array A looks like a [3; 3], the array B looks like b [3; 3]. The minimal element of A is not less than the maximal element b.

6. Test Cases:

	Input data	Output data
1	3 2 4 5 6 1 2	0
2	1 1 100 4	0
3	1 1 100 183299	183199
4	1 2 1 2 2	1
5	5 8 14 2 54 11 12 43 15 55 43 119 12 43 54	**232**
6	10	150

	5 4 32 67 12 99 10 23 11 657 12 43 78 11 54 12	
7	12 15 1 43 12 65 98 324 87 23 12 67 348 23 12 54 87 23 67 87 189 54 12 79 43 87 32 87 21	**504**

In Example 1, it is not need to perform any operations.

7. Sample solution (Java):

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

public class Main
{
public static void main(String args[] )
{
BufferedReader br = new BufferedReader( new InputStreamReader( System.in), 8*1024 ) ;
Scanner sc = new Scanner ( br ) ;

int n = sc.nextInt();
int m = sc.nextInt();
```

```
ArrayList<Integer> a = new ArrayList<Integer>(n);

ArrayList<Integer> b = new ArrayList<Integer>(m);


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

a.add( i , sc.nextInt() );


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

b.add( i , sc.nextInt() );


Collections.sort( a );

Collections.sort( b ,Collections.reverseOrder() );


long ans = 0 ;



for(int i = 0 ; i < Math.min( n , m ) && a.get(i) < b.get(i) ; ++i )

ans += b.get(i) - a.get(i) ;


System.out.println( ans );

}


}
```

5

Pocket money

1. **Problem statement:**

James wants to pass *n* exams and get a double sum for his pocket money. For that the average score for all the exams must be higher than or equal to the average score (*avg*). The maximum score for each exam can't exceed r. James passed the exams, scoring ai points for the i-th exam. James can improve his score for the ai-th exam by 1 point by doing extra bi hours of homework.

What is the minimum number of homework hours that James needs to double his pocket money?

2. Input Format:

The first line contains 3 integers *n*, *r*, *avg* — the number of exams, the maximum score and the average score required to double his pocket money.

Each of the following *n* lines contains integers *ai* and *bi* — James' score for the i-th exam and the number of hours required to improve that score by one point.

3. Output Format:

Output the minimum number of homework hours required to double James' pocket money.

4. Constraints:

$1 \le n \le 10^5$, $1 \le r \le 10^9$, $1 \le avg \le min(r, 10^6)$

$1 \le ai \le r$, $1 \le bi \le 10^6$

5. Samples Input/Output:

	Input	Output
1	5 5 4 5 2 4 7 3 1 3 2 2 5	4
2	2 5 4 5 2 5 2	0

6. Test cases

	Input	Output
1	6 5 5 1 7 2 4 3 5 4 6 5 6 4 7	63
2	1 1000000000 1000000 1 1000000	999999000000
3	10 10 7 1 10 2 9 3 8 4 7 5 6 6 5 7 4 8 3 9 2 10 1	70
4	2 1000000000 1000000 1000000 5 999998 7	10
5	10 10 6 1 10 2 9 3 8 4 7 5 6	16

	6 5 7 4 8 3 9 2 10 1	
6	10 10 7 1 10 2 9 3 8 4 7 5 6 6 5 7 4 8 3 9 2 10 1	70
7	1 2 1 2 2	0
	2 1000000 1000000 1 1000000 1 1000000	1999998000000

7. Solution (Java)

```
import java.util.*;

import java.io.*;

public class SampleClass {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);
```

```
int n = in .nextInt();
int max = in .nextInt();
int avg = in .nextInt();
long[] assay = new long[1000001];
long cur = 0;
for (int i = 0; i < n; i++) {
int a = in .nextInt();
int b = in .nextInt();
assay[b] += (max - a);
cur += a;
}
long count = 0;
int i = 0;
long needed = (long) avg * n;
while (cur < needed) {
i++;
if (assay[i] > 0) {
count += (long)((long) Math.min(assay[i], needed - cur)) * (long) i;
cur += assay[i];
}
}
System.out.println(count);
}
}
```

Happy Colleagues

1. **Problem statement:**

You have some M&Ms candies in your pocket. Everyone in your office loves M&Ms. You have candies of three colors: r - red, g - green, b - blue.

To make your colleagues happy you need to give each of them three candies, but at least two of the candies have to be of different color.

What is the maximum number of colleagues that you can make happy if you have a certain amount of candies?

2. Input Format:

The first line contains integers r, g, b - the number of candies of red, green and blue colors respectively.

3. Output Format:

Output the number of colleagues you can make happy with your candies, assuming you don't eat any candies yourself.

4. Constraints:

$1 \leq n \leq 10^9$

5. Samples Input/Output:

	Input	Output
1	5 4 3	4
2	1 1 1	1

6. Test cases

	Input	Output
1	2 3 3	2
2	0 1 0	0
3	0 3 3	2

4	2000000000 2000000000 2000000000	2000000000
5	1 2000000000 1000000000	1000000000
6	0 0 0	0
7	150000 75000 75000	100000
8	100500 100500 3	67001

7. Solution (Java)

```
import java.util.Scanner;

public class SampleClass{

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

long r = in .nextLong(), g = in .nextLong(), b = in .nextLong();


System.out.println(Math.min(Math.min(Math.min(r + g, r + b), b + g), (r + b + g) / 3));

}

}
```

7

Big Chessboard

1. **Problem statement:**

There is a chessboard with 10^9 rows and 10^9 columns. We assume that the rows of the chessboard are numbered with integers from 1 to 10^9 - from the top row to the bottom one.

Similarly, the columns are numbered with integers from 1 to 10^9 from left to right. The cell located in the i-th row and the j-th column is denoted as (i, j).

It is known that some of the cells of the given chessboard are the permitted cells. All the permitted cells on the chessboard are described as n segments. Each segment is described by three integers ri, ai, bi (ai ≤ bi), indicating that cells from ai to bi inclusive are permitted in the ri-th row.

Your task is to find the minimum number of moves in which you will be able to reach the cell (x1, y1) from the cell (x0, y0), by moving only on via the permitted cells.

You can move to another cell if that cell is a neighboring cell. Two cells are considered neighboring if they have at least one common point.

2. Input Format:

The first line contains four integers x0, y0, x1, y1 indicating the starting position and the destination point.

The second line contains an integer n indicating the number of segments of the permitted cells. The following n lines contain descriptions of these segments. The i-th of these lines contains three integers ri, ai, bi, indicating that the ri-th row with columns from ai to bi inclusive, are permitted. Note that the lengths of the permitted cells can be arbitrarily crossed and nested.

It is guaranteed that the starting position and the destination point are permitted cells. It is guaranteed that the starting position does not equal the destination point.

3. Output Format:

If the path between the starting position and the destination point, consisting of permitted cells, does not exist, print out -1.

Otherwise, output a single integer - the minimum number of moves in which you can get from the starting position to the destination point.

4. Constraints:

1 ≤ x0, y0, x1, y1 ≤ 10^9

1 ≤ n ≤ 10^5

1 ≤ ri, ai, bi ≤ 10^9, ai ≤ bi

It is guaranteed that the total length of all segments does not exceed 10^5.

5. Samples Input/Output:

	Input data	Output data
1	5 7 6 11 3 5 3 8 6 7 11 5 2 5	4
2	3 4 3 10 3 3 1 4 4 5 9 3 10 10	6

6. Test Cases:

	Input data	Output data
1	1 1 2 10 2 1 1 3 2 6 10	-1
2	9 8 7 8 9 10 6 6 10 6 6 7 7 8 9 5 6 8 9 9	2

	9 5 5 9 8 8 8 5 6 9 10 10	
3	6 15 7 15 9 6 15 15 7 14 14 6 15 15 9 14 14 7 14 16 6 15 15 6 15 15 7 14 14 8 15 15	1
4	13 16 20 10 18 13 16 16 20 10 10 19 10 10 12 15 15 20 10 10 18 11 11 19 10 10 19 10 10 20 10 10 19 10 10 20 10 10 20 10 10 19 10 10 18 11 11 13 16 16 12 15 15	-1

	19 10 10 19 10 10	
5	89 29 88 30 16 87 31 31 14 95 95 98 88 89 96 88 88 14 97 97 13 97 98 100 88 88 88 32 32 99 88 89 90 29 29 87 31 31 15 94 96 89 29 29 88 32 32 97 89 89 88 29 30	1
6	30 14 39 19 31 35 7 11 37 11 12 32 13 13 37 5 6 46 13 13 37 14 14 31 13 13 43 13 19 45 15 19 46 13 13 32 17 17	9

	41 14 19 30 14 14 43 13 17 34 16 18 44 11 19 38 13 13 40 12 20 37 16 18 46 16 18 34 10 14 36 9 10 36 15 19 38 15 19 42 13 19 33 14 15 35 15 19 33 17 18 39 12 20 36 5 7 45 12 12	
7	1 1 1 2 5 1000000000 1 10000 19920401 1188 5566 1000000000 1 10000 1 1 10000 5 100 200	1

7. Sample solution (Java):

```
import java.io.BufferedReader;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Set;
import java.util.HashSet;
public class KingsPath{
static Set<String> vis;
static Set<String> a;
static int l = 1000000001;
static int dx[] = {1,-1,0,0,1,1,-1,-1},
dy[] = {0,0,1,-1,1,-1,1,-1};
public static void main(String[] args) throws IOException{
BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st = new StringTokenizer(input.readLine());
a = new HashSet<String>();
int x,y,x1,y1;
x = Integer.parseInt(st.nextToken());
y = Integer.parseInt(st.nextToken());
x1 = Integer.parseInt(st.nextToken());
y1 = Integer.parseInt(st.nextToken());
int n = Integer.parseInt(input.readLine());
while(n-->0){
int i,j1,j2;
st = new StringTokenizer(input.readLine());
```

```
i = Integer.parseInt(st.nextToken());
j1 = Integer.parseInt(st.nextToken());
j2 = Integer.parseInt(st.nextToken());
for(int k=j1; k<=j2; k++)
a.add(i+" "+k);
}
System.out.println(bfs(x,y,x1,y1));
}
public static int bfs(int x,int y,int x1,int y1){
vis = new HashSet<String>();
Queue<Node> q=new LinkedList<Node>();
q.add(new Node(x,y,0));
vis.add(x+" "+y);
while(!q.isEmpty()){
Node n = q.poll();
if(n.x==x1 && n.y==y1)return n.t;
for(int i=0; i<8; i++){
int xx = n.x + dx[i];
int yy = n.y + dy[i];
if(xx>=1 && yy>=1 && xx<l && yy<l &&!vis.contains(xx+" "+yy)&&a.contains(xx+" "+yy)){
vis.add(xx+" "+yy);
q.add(new Node(xx,yy,n.t+1));
}
}
}
return -1;
}
}
```

```
class Node{

int x,y;

int t ;

Node(int xx,int yy,int tt){

x = xx;y=yy;

t = tt;

}

}
```

8

Boy and Boxes

1. **Problem statement:**

A little boy has n boxes with balls. The boxes are standing in a row and are numbered from 1 to n from left to right.

One day, the boy chose one of the boxes, let's assume that it was box number i, pulled out all the balls (it is guaranteed that this box initially had at least one ball in it) and started putting one ball into every box with the numbers i + 1, i + 2, i + 3, and so on. If the boy puts a ball into the box number n, he puts the next ball in the box number 1, then in the box number 2, and so on. And so he continued until he had no balls left. It is possible that one or more balls will go into the box number i.

For example, let’s suppose that initially, the boy had four boxes, and in the first box there were 3 balls, in the second - 2, in the third - 5, in the fourth - 4 balls. Then if i = 3, they boy will take out all the five balls from the third box and put them in the boxes numbers 4, 1, 2, 3, 4. After that, the balls will be distributed among the boxes as follows: 4 balls in the first box, 3 balls in the second one, one ball in the third and 6 balls in the fourth box.

At this moment, the boy forgot how the balls were initially distributed among the boxes , but he knows how they are distributed now, and he also knows the number x - the number of the box, where he put the last ball he had pulled out.

He asks you to help him find the initial location of all the balls in the boxes.

2. Input Format:

The first line of input contains two integers - n and x - the number of boxes and the number of the box, where he put the last ball he had pulled out. The second line contains n integers a1, a2, ..., an, where the number of ai stands for the number of balls in the box number i after the little boy had performed all the actions described.

3. Output Format:

Output n integers, where the i-th is the number of balls in the i-th box before all the actions. The numbers are separated by spaces. If there are multiple correct solutions, output any of them.

4. Constraints:

$2 \le n \le 10^5, 1 \le x \le n$

$0 \le a_i \le 10^9, a_x \ne 0.$

5. Samples Input/Output:

	Input data	Output data
1	4 4 4 3 1 6	3 2 5 4
2	5 2 3 2 0 2 7	2 1 4 1 6

6. Test Cases:

	Input data	Output data
1	3 3 2 3 1	1 2 3
2	10 3 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000	0 0 10000000000 0 0 0 0 0 0 0

	1000000000	
3	5 4 0 554459682 978416312 784688178 954779973	3 554459681 978416311 784688177 954779973
4	5 2 1 554459683 978416312 784688178 954779974	6 554459681 978416311 784688177 954779973
5	10 8 994538714 617271264 168716105 915909382 338220996 533154890 507276501 323171960 121635370 33140162	961398551 584131101 135575942 882769219 305080833 500014727 474136338 290031797 88495208 331401628
6	10 5 994538715 617271265 168716106 915909383 338220997 533154890 507276501 323171960 121635371 33140163	961398551 584131101 135575942 882769219 305080833 500014727 474136338 290031797 88495208 331401635
7	10 5 3 3 3 3 4 3 3 3 3 3	0 0 0 31 0 0 0 0 0 0

7. Sample solution (Java):

```
import java.io.*;

import java.util.*;
```

```
public class Main {

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

public static void main(String[] args) {
int n = sc.nextInt();
int x = sc.nextInt();
long[] a = new long[n];
for (int i = 0; i < n; i++) {
a[i] = sc.nextInt();
}

long m = Long.MAX_VALUE;
int pos = 0;

for (int i = 0; i < n; i++) {
if (a[i] < m) {
m = a[i];
pos = i;
}
}
```

```
for (int i = 0; i < n; i++) {
a[i] -= m;
}

x--;
long k = 0;
int p = x;
if (p < 0) p = n-1;
while (a[p] != 0) {
a[p]--;
k++;
p--;
if (p<0) p = n - 1;
}

a[p] = m*n+k;
for (int i = 0; i < n-1; i++)
System.out.print(a[i] + " ");
System.out.print(a[n-1]);
}
}
```

9

Treepath

1. **Problem statement:**

There is a **connected,** weighted and undirected graph G. The shortest path tree from vertex *u* is graph G1 that is a tree with a set of edges that is the subset of the set of edges of the initial graph, and the lengths of the shortest paths from *u* to any vertex to *G graph* and to *G*1 are the same.

You are given a **connected** weighted undirected graph *G* and a vertex *u*. Your task is to find the shortest path tree from vertex u, whose total weight of edges is minimum.

2. Input Format:

The first line contains two numbers, *n* and *m* — the number of vertices and edges of the graph, respectively.

Next *m* lines contain three integers each, representing an edge — *ui*, *vi*, *wi* — the numbers of the vertices connected by an edge and the weight of the edge . It is guaranteed that the graph is connected, and that there is no more than one edge between any pair of vertices. The edges are numbered starting from 1 in accordance with the order they follow in the input data.

The last line of input contains an integer *u* — the number of the starting vertex.

3. Output Format:

In the first line output the minimal total weight of the edges of the tree.

In the next line output the indices of the edges of the tree, separated by spaces. Numbers of the edges can be displayed in any order.

If there are multiple answers, print out any of them.

4. Constraints:

$1 \le n \le 3 \cdot 10^5, 0 \le m \le 3 \cdot 10^5$

$ui \ne vi, 1 \le wi \le 10^9$

$1 \le u \le n$

5. Samples Input/Output:

	Input data	Output data
1	3 3 1 2 1	2 1 2

	2 3 1 1 3 2 3	
2	4 4 1 2 1 2 3 1 3 4 1 4 1 2 4	4 2 3 4

In the first case there are two possible shortest path trees:

- with edges 1 – 3 and 2 – 3 (the total weight is 3);
- with edges 1 – 2 and 2 – 3 (the total weight is 2);

And, for example, a tree with edges 1 – 2 and 1 – 3 won't be a shortest path tree for vertex 3, because the distance from vertex 3 to vertex 2 in this tree equals 3, and in the original graph it is 1.

6. Test Cases:

	Input data	Output data
1	4 5 1 2 1 1 3 1 2 4 1 3 4 1 2 3 10 1	3 1 2 3
2	6 8 1 2 30	230 1 4 5 6 7

	1 3 20 2 3 50 4 2 100 2 5 40 3 5 10 3 6 50 5 6 60 4	
3	2 1 1 2 1000000000 2	1000000000 1
4	5 5 1 2 30 1 3 40 2 3 20 3 4 60 1 5 10 1	**140** **5 1 2 4**
5	5 8 1 2 40 2 3 10 1 3 60 1 4 20	**36** **2 8 7 4**

	3 4 30 1 5 100 4 5 5 3 5 1 2	
6	3 4 2 3 11 2 4 12 3 4 1 1 4 2 2	**14** **1 3 4**
7	6 12 3 6 54 4 6 81 5 3 10 5 6 1 1 2 48 2 6 32 1 3 3 4 5 19	**30** **7 12 3 4 10**

	1 5 25 4 2 11 1 4 51 2 3 5 3	

7. Sample solution (Java):

```
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.TreeSet;

public class PathsAndTrees {
private static class Edge {
int v;
long w;
int index;

Edge(int v, long w, int index) {
this.v = v;
this.w = w;
```

```
this.index = index;
}
}

public static void main(String[] args) {
Scanner in = new Scanner(System.in);
PrintStream out = System.out;

int n = in.nextInt(), m = in.nextInt();
List<Edge>[] graph = new List[n];

for (int i = 0; i < n; i++) {
graph[i] = new ArrayList<PathsAndTrees.Edge>();
}

for (int i = 1; i <= m; i++) {
int v1 = in.nextInt() - 1;
int v2 = in.nextInt() - 1;
long w = in.nextLong();

graph[v1].add(new Edge(v2, w, i));
graph[v2].add(new Edge(v1, w, i));
}
int u = in.nextInt() - 1;

Edge[] lastEdge = new Edge[n];
final long[] min = new long[n];
for (int i = 0; i < n; i++) {
```

```
min[i] = -1;
}

min[u] = 0;

PriorityQueue<Integer> q = new PriorityQueue<Integer>(n, new Comparator<Integer>() {
public int compare(Integer o1, Integer o2) {
if (min[o1] < min[o2]) {
return -1;
} else if (min[o1] > min[o2]) {
return 1;
}
return o1 - o2;
}

});

boolean[] wasInQueue = new boolean[n];

q.add(u);

while (!q.isEmpty()) {
int v = q.poll();

wasInQueue[v] = true;

for (Edge edge : graph[v]) {
```

```
int v1 = edge.v;
long min1 = min[v] + edge.w;

if ((min[v1] == -1) || (min1 < min[v1])
|| (min1 == min[v1] && edge.w < lastEdge[v1].w)) {

min[v1] = min1;
lastEdge[v1] = edge;
q.add(v1);
}
}
}

long res = 0;
boolean[] f = new boolean[m];

for (int i = 0; i < n; i++) {
if (lastEdge[i] != null) {
res += lastEdge[i].w;
f[lastEdge[i].index - 1] = true;
}
}

out.println(res);

StringBuilder s = new StringBuilder();
boolean first = true;
for (int i = 0; i < m; i++) {
```

```
if (f[i]) {
if (!first) {
s.append(" ");
}
s.append(i + 1);
first = false;
}
}
out.println(s.toString());
in.close();
out.close();

}

}
```

10

Another Game of Life

1. **Problem statement:**

The given territory is a table of n * n cells. There are several (m) populations of creatures that want to occupy some land. They also want to have some buffer areas, so the cells occupied by one population must not have any common vertical or horizontal edges with the cells occupied by some other population.

2. Input Format:

Two space-delimited numbers: n, the size of the map, and m, the number of populations that want to occupy some land.

3. Output Format:

If a correct answer does not exist, output 'NO'. Otherwise, output 'YES'. Describe the territory in the next n strings. Each string must consist of n characters, where 'O' stands for a free cell and 'X' stands for a cell occupied by some creatures.

4. Constraints:

$1 \leq n \leq 100$

$0 \leq m \leq n^2$

5. Samples Input/Output:

	Input	Output
1	8 64	NO
2	6 5	YES XOXOXO OXOXOO OOOOOO OOOOOO OOOOOO OOOOOO

6. Test cases

	Input	Output
1	3 5	XOX OXO XOX
2	82 6047	NO

3	1 58	NO
4	9 12	YES XOXOXOXOX OXOXOXOXO XOXOXOOOO OOOOOOOOO OOOOOOOOO OOOOOOOOO OOOOOOOOO OOOOOOOOO OOOOOOOOO
5	10 500	NO
6	20 400	NO
7	100 10000	NO

7. Solution (Java)

```
public static void main(String argv[]) {
 Scanner input = new Scanner(System.in);
 int n,m;
n=input.nextInt();
m=input.nextInt();
 char a[][] = new char[110][110];
 for (int i = 0; i < n; i++)
 for (int j = 0; j < n; j++)
 if ((i+j)%2 == 0 && m > 0) {
m--;
```

```
a[i][j] = 'X';
 } else {
a[i][j] = 'O';
}
if (m==0) {
 System.out.println("YES");
 for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++)
 System.out.print(a[i][j]);
 System.out.println();
 }
 } else {
 System.out.println("NO");
}
}
```

11

Coders

1. **Problem statement:**

In some company, the programmers received a task: write exactly m lines of code. There are n programmers employed in the company, the i-th of them makes exactly ai mistakes in every line of code they write.

Let’s call a sequence of non-negative integers v1, v2, ..., vn a plan if v1 + v2 + ... + vn = m. The programmers are executing the plan as follows: the first programmer writes the first v1 lines of the task, then the second programmer writes the next v2 lines, and so on. At the end of that sequence, the last programmer finishes the remaining lines of code. Let’s call this plan a good one if the sum of all lines of code contains no more than b errors.

You need to determine how many different good plans there are and print out the remainder of that number divided by mod.

2. Input Format:

The first line contains four integers n, m, b, mod - the number of developers, the number of lines of code in the task, the maximum possible number of errors and the module you need to use to get the answer.

The next line contains n space-delimited integers a1, a2, ..., an - the number of errors that each programmer makes per one line of code.

3. Output Format:

Print out a single integer - the remainder of the answer divided by mod.

4. Constraints:

$1 \le n, m \le 500, 0 \le b \le 500, 1 \le mod \le 10^9 + 7$

$0 \le a_i \le 500$

5. Samples Input/Output:

	Input data	Output data
1	3 3 3 100 1 1 1	10
2	3 6 5 1000000007 1 2 3	0

6. Test Cases:

	Input data	Output data
1	3 5 6 11	0

	1 2 1	
2	2 3 3 1000 1 2	1
3	1 4 25 1000 6	1
4	1 1 0 1000 0	1
5	2 500 50 10000 0 50	2
6	21 63 40 1009 4 4 2 2 4 4 3 2 4 2 0 3 3 4 3 4 3 0 4 2 4	1002
7	29 157 50 1 3 0 0 3 1 1 2 0 4 4 1 2 2 1 0 0 2 0 3 2 2 3 3 1 4 1 1 4 1	0

7. Sample solution (Java):

```
import java.util.*;

public class LevelTwo {
```

```
public static void main(String[] args) {
// TODO Auto-generated method stub
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
int m = scan.nextInt();
int b = scan.nextInt();
int mod = scan.nextInt();

int [][] mat = new int[501][501];
for(int i = 0; i < 501; i++)
mat[0][i] = 1;

for(int k = 0; k < n; k++){
int v = scan.nextInt();
for(int i = 1; i <= m; i++){
for(int j = v; j <= b; j++){
mat[i][j] = (mat[i][j] + mat[i - 1][j - v]) % mod;
}
}
}

System.out.println(mat[m][b]);
}
}
```

Snowflakes

1. **Problem statement:**

You are given an array with the length of n, that has k letters of the English alphabet (e.g. A,B,C,D if k == 4).

How many elements do you need to change (assign another letter) so that no neighbouring elements have the same letter?

2. Input Format:

The first input line contains two integers *n* and *k*. The second line contains *n* uppercase English letters. The first *k* English letters can be used.

3. Output Format:

Print out a single integer — the minimum number of changes. On the second line, print any possible variant of the changed array.

4. Constraints:

$1 \le n \le 5 \cdot 105; 2 \le k \le 26$

5. Samples Input/Output:

	Input	Output
1	6 3 ABBACC	2 ABCACA
2	3 2 BBB	1 BAB

6. Test cases

	Input	Output
1	1 2 A	0 A
2	2 2 AA	1 BA

3	6 2 AAABBB	2 ABABAB
4	8 3 AABBABBB	3 ACBCABAB
5	10 26 AAAAAAAAAA	5 ABABABABAB
6	12 3 AAABBBAAABBB	4 ABABABABABAB
7	8 3 AABBCCBB	4 ACBACABA
8	20 2 BBBBAAAAAABBBAAAAAAB	10 BABABABABABABABABABA

7. Solution (Java)

```
import java.io.*;

import java.util.*;

public class Main {


public static void main(String[] args) {

Scanner in = new Scanner(System.in);

PrintWriter out = new PrintWriter(System.out);

int N = in .nextInt();

int K = in .nextInt();

char[] wm = in .next().toCharArray();

if (K == 2) {

int c1 = 0;

int c2 = 0;

for (int i = 0; i < N; i++) {
```

```
if (wm[i] != i % 2 + 'A') c1++;
}
for (int i = 0; i < N; i++) {
if (wm[i] != (i + 1) % 2 + 'A') c2++;
}
System.out.println(Math.min(c1, c2));
int df = (c1 < c2) ? 0 : 1;
for (int i = 0; i < N; i++) {
char c = (char)((i + df) % 2 + 'A');
System.out.print(c);
}
System.out.println();
} else {
int mods = 0;

for (int st = 0; st < N; st++) {
char fr = (st == 0) ? '?' : wm[st - 1];
char bk = (st == N - 1) ? '?' : wm[st + 1];
char cc = wm[st];
if (cc == fr) {
cc = 'A';
if (cc == bk || cc == fr) {
cc++;
}
if (cc == bk || cc == fr) {
cc++;
}
wm[st] = cc;
```

```
mods++;
}
}
System.out.println(mods);
System.out.println(new String(wm));
}
}

}
```

13

Pony Sequence

1. **Problem statement:**

A Pony Sequence is a sequence in which the greater common divisor of all pairs of elements is 1. You have an array A. Your task is to create a Pony Sequence B while meeting the following condition : the following sum,

$$\sum_{i=1}^{n} |a_i - b_i|.$$

will be minimum.

2. Input Format:

The first line contains an integer *n* — the number of elements in the sequences *A* and *B*. The next line contains *n* integers *a*1, *a*2, ..., *an* .

3. Output Format:

Output the sequence B that minimizes the sum described above. If there are multiple optimal sequences, you can output any of them.

4. Constraints:

$1 \le n \le 100$

$1 \le ai \le 30$

5. Samples Input/Output:

	Input data	Output data
1	5 1 1 1 1 1	1 1 1 1 1
2	5 1 6 4 2 8	1 5 3 1 8

6. Test Cases:

	Input data	Output data
1	10 16 3 16 10 12 5 14 14 15 27	16 1 17 9 7 5 11 13 19 29
2	10 8 7 11 5 17 24 28 18 7 8	8 1 11 5 17 23 29 19 7 9
3	10 22 17 28 14 14 26 20 28 21 27	23 16 29 11 13 25 17 31 19 27
4	30 24 11 1 17 15 11 12 8 6 25 22 23 9 3 30 3 10 28 15 8 4 28 3 28 26 14 24 1 6 19	27 1 1 1 1 1 1 1 1 25 17 23 1 1 32 1 1 29 11 7 1 31 1 37 41 13 43 1 1 19
5	30 11 20 1 17 6 20 22 16 20 22 21 8 3 28 30 2 27 14 10 14 29 21 13 3 13 27 11 18 2 15	1 1 1 1 1 1 25 1 1 23 16 1 1 29 31 1 27 1 1 1 37 19 1 1 7 41 11 17 1 13

6	30 17 16 18 4 3 22 28 4 4 17 9 30 2 11 29 12 8 17 9 12 13 11 13 18 28 17 10 12 14 20	1 1 1 1 1 25 27 1 1 1 1 31 1 1 29 1 1 16 1 1 1 1 1 19 37 17 7 11 13 23
7	30 9 26 5 7 29 17 19 22 1 28 5 6 9 8 13 9 3 4 16 16 11 24 22 20 12 9 16 22 2 11	1 27 1 1 29 1 17 23 1 31 1 1 1 1 1 1 1 1 1 16 1 25 37 19 1 7 13 41 1 11

7. Sample solution (Java):

```
import java.util.Arrays;

import java.util.Scanner;


public class B {


static int[] primes = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,

47, 53, 59 };

static int[] mask = new int[60];

static int[][] dp = new int[105][1 << 17];


static int[] arr;

static {

for (int i = 1; i < mask.length; i++)

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

if (i % primes[j] == 0)
```

```
mask[i] |= 1 << j;
for (int i = 0; i < dp.length; i++)
Arrays.fill(dp[i], -1);
}
static int[][] recover = new int[105][1 << 17];

public static int f(int i, int msk) {
if (i == arr.length)
return 0;
if (dp[i][msk] != -1)
return dp[i][msk];
int ans = arr[i] - 1 + f(i + 1, msk);
int choice = 1;
for (int j = 2; j < arr[i] * 2; j++)
if ((msk & mask[j]) == 0) {
int cans = Math.abs(arr[i] - j) + f(i + 1, msk | mask[j]);
if (cans < ans) {
ans = cans;
choice = j;
}
}
recover[i][msk] = choice;
return dp[i][msk] = ans;
}

public static void main(String[] args) {
Scanner in = new Scanner(System.in);
int n = in.nextInt();
```

```
arr = new int[n];

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

arr[i] = in.nextInt();

int result = f(0, 0);

int[] arr2 = new int[n];

int cmsk = 0;

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

arr2[i] = recover[i][cmsk];

cmsk |= mask[recover[i][cmsk]];

}

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

System.out.print(arr2[i] + " ");

}

}
```

14

Elegance of a Sequence

1. **Problem statement:**

There is a sequence of positive integers *a*1, *a*2, ..., *an*. Joshua wants to write down several numbers on a piece of paper, so that the elegance of the numbers he wrote down is at its maximum.

The *elegance* of the written down numbers *b*1, *b*2, ..., *bk* is a maximum non-negative integer *v*, that allows numbers *b*1 *and b*2 *and* ... *and bk to* be divided by number 2^v without a remainder. If such a number *v* doesn't exist, the elegance of the written down numbers equals -1.

What numbers should he write down so that the elegance of the written down numbers is maximum? If there are multiple ways to write down the numbers, you need to choose the one in which Joshua writes down as many numbers as possible.

2. Input Format:

The first line contains an integer *n*. The second line contains *n* space-delimited integers *a*1, *a*2, ..., *an*.

3. Output Format:

On the first line, output the only number k (k> 0) - the amount of numbers that need to be written down. On the second line, print out *k* integers *b*1, *b*2, ..., *bk* — the numbers to write down. The numbers b1, b2, ..., bk can be printed out in any order, but they must all be different. If there are multiple ways to write down the numbers, choose the one with the maximum amount of numbers to write down. If there are still multiple ways left, you are free to print out any of them.

4. Constraints:

$1 \le n \le 10^5$

$1 \le a1 < a2 < ... < an \le 10^9$

5. Samples Input/Output:

	Input data	Output data
1	5 1 2 3 4 5	2 4 5
2	3 1 2 4	1 4

6. Test Cases:

	Input data	Output data
1	3 1 20 22	2 20 22
2	10 109070199 215498062 361633800 406156967 452258663 530571268	6 361633800 406156967 452258663 530571268

	670482660 704334662 841023955 967424642	841023955 967424642
3	2 536870911 536870912	1 536870912
4	5 2 4 16 32 128	**1** **128**
5	6 32 48 54 24 36 42	**5** **32 48 54 36 42**
6	10 840 456 138 537 378 436 672 241 480 356	**3** **840 537 672**
7	4 435625 23458 5329 532	**2** **23458 532**

7. Sample solution (Java):

```
import java.util.*;
public class SampleClass {
public static void main(String [] args){
final Scanner s = new Scanner(System.in);
final int n = s.nextInt();
final int [] a = new int[n];
for (int i = 0; i < n; ++i){
a[i] = s.nextInt();
}
```

```
for (int i = 30; i >= 0; --i){
final List<Integer> res = new ArrayList<Integer>();
int and = ~0;
for (int j = 0; j < n; ++j){
if ((a[j] & (1 << i)) != 0){
res.add(a[j]);
and = and & a[j];
}
}
if ((and & (-and)) == (1 << i)){
System.out.println(res.size());
for (int b : res){
System.out.print(b + " ");
}
System.out.println();
break;
}
}
s.close();
}
}
```

15

Cotton Sequence

1. **Problem statement:**

A sequence of non-negative integers *a*1, *a*2, ..., *an* with the length of *n* is called a cotton sequence if and only if there exist such two integers *l* and *r* (1 ≤ *l* ≤ *r* ≤ *n*), that

$$a_l \oplus a_{l+1} \oplus \cdots \oplus a_r = 0$$

.

The expression $x \oplus y$ means applying the operation of a bitwise xor to numbers *x* and *y*.

Your task is to compute the number of sequences, made of *n* integers from 0 to 2^*m* - 1, that are not a cotton sequence. You should print this number out modulo 1000000009 (10^9 + 9).

2. Input Format:

The only line of input contains two space-delimited integers *n* and *m* .

3. Output Format:

Print out the required number of sequences modulo 1000000009 (10^9 + 9) on the only line of output.

4. Constraints:

1 ≤ *n*, *m* ≤ 10^5

5. Samples Input/Output:

	Input data	Output data
1	3 2	6
2	4 2	0

In first example sequences of length 3 made of integers 0, 1, 2 and 3 that are not a cotton sequence are (1, 3, 1), (1, 2, 1), (2, 1, 2), (2, 3, 2), (3, 1, 3) and (3, 2, 3).

6. Test Cases:

	Input data	Output data
1	1 2	3
2	4 11	433239206
3	5 100	345449482
4	5444 31525	637906839

5	60282 92611	814908070
6	62600 39199	366778414
7	14095 86011	656508583

7. **Sample solution (Java):**

```
import java.util.Scanner;

public class Main {

static void solve() {

Scanner sc = new Scanner(System.in);

final int n = sc.nextInt();

final long m = sc.nextInt();

final long mod = (long)(1e9 + 9);

long[] dp = new long[n+1];

long tpm = 1;

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

tpm = (tpm * 2) % mod;

}

dp[0] = 1;

for (int l = 1; l <= n; ++l) {

dp[l] = (dp[l-1] * (tpm - l + mod)) % mod;

}

System.out.println(dp[n]);
```

```
}
public static void main(String[] args) {
solve();
}
}
```

16

Palindromization

1. **Problem statement:**

Make a palindrome from the given string in as few steps as possible. In one step you can: move left or right (cyclically, moving left from the first position leads to the last position and vice versa, the index is 1-based); increment or decrement the current character (cyclically, incrementing z will result in a, decrementing a will result in z).

2. Input Format:

The first line contains two space-delimited numbers – *n,* – string length; *p*, – the current position of the cursor.

The second line contains the given string of n lowercase Latin letters.

3. Output Format:

The minimum number of steps required to palindromize the given string from your current position.

4. Constraints:

$1 \leq n \leq 10^5$

$1 \leq p \leq n$

5. Samples Input/Output:

	Input	Output
1	5 5	12

	pjfjb	
2	10 6 abcdefdcba	1

6. Test cases

	Input	Output
1	85 19 blkimwzicvbdkwfodvigvmnujnotwuobkjvugbtaseebxvdiorffq nhllwtwdnfodkuvdofwkdbvcizwmiklb	187
2	8 3 aeabcaez	6
3	8 3 abcddcbb	3
4	4 4 rkoa	14
5	39 30 yehuqwaffoiyxhkmdipxroolhahbhzprioobxfy	138
6	40 23 vwjzsgpdsopsrpsyccavfkyyahdgkmdxrquhcplw	169

7	10 5 abcdeedcba	0

7. Solution (Java)

```
public static void main(String[] args) throws IOException {
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 StringTokenizer st = new StringTokenizer(br.readLine());
 int n = Integer.parseInt(st.nextToken());
 int p = Integer.parseInt(st.nextToken()) - 1;
 char[] cc = br.readLine().toCharArray();
 int cnt = 0;
 int l = -1, r = -1;
 for (int i = 0, j = n - 1; i < j; i++, j--)
 if (cc[i] != cc[j]) {
 int diff = Math.abs(cc[i] - cc[j]);
cnt += Math.min(diff, 26 - diff);
 if (l == -1) l = i; r = i;
 }
p = Math.min(p, n - 1 - p);
if (cnt > 0) {
 if (p <= l) cnt += r - p;
 else if (p >= r) cnt += p - l;
 else cnt += Math.min(p - l, r - p) + r - l;
}
System.out.println(cnt);
}
```

17

Two Piles of Balls

1. **Problem statement:**

Max has 2 · n balls, each of the balls has an integer from 10 to 99 written on it. He randomly chooses n balls and puts them to the first pile. The remaining balls form a second pile.

Max decided to play with the balls. He takes a ball from the first pile and writes down the number written on it. Then he takes a ball from the second pile and writes down the two digits written on it next to the two digits that he previously wrote down (to their right). At the end, he got a single four-digit integer — the first two digits of it are written on the ball from the first pile, and the second two digits are written on the second ball from the second pile.

Max is interested in arithmetic, so it wasn't hard for him to find the amount of different four-digit numbers that can he can get as the result of this funny game. The more tricky question is - how to divide the balls into piles so that one could get as many different four-digit numbers as possible?

2. Input Format:

The first line contains an integer n . The second line contains $2 \cdot n$ space-delimited integers a_i, denoting the numbers on the balls.

3. Output Format:

On the first line, print out a single number — the maximum possible amount of distinct four-digit numbers that Max can get. On the second line, print out $2 \cdot n$ numbers b_i $(1 \le b_i \le 2)$. The numbers mean: the i-th ball belongs to the b_i-th pile in your division.

If there are multiple optimal ways to divide the balls among the piles, print out any of them.

4. Constraints:

$1 \le n \le 100$

$10 \le a_i \le 99$

5. Samples Input/Output:

	Input data	Output data
1	1 10 99	1 2 1
2	2 13 24 13 45	4 1 2 2 1

6. Test Cases:

	Input data	Output data
1	5 21 60 18 21 17 39 58 74 62 34	25 1 2 2 2 1 2 1 2 1 1
2	10 26 43 29 92 22 27 95 56 72 55 93 51 91 30 70 77 32 69 87 98	100 2 1 2 1 1 1 1 2 1 1 2 2 2 1 2 2 2 1 1 2
3	2 10 10 10 11	2 1 2 2 1
4	10 109070199 215498062 361633800 406156967 452258663 530571268 670482660 704334662 841023955 967424642	6 361633800 406156967 452258663 530571268 841023955 967424642

5	1 536870912	1 536870912
6	2 536870911 536870912	1 536870912
7	30 61 65 67 71 73 75 77 79 129 131 135 137 139 141 267 520 521 522 524 526 1044 1053 6924600 32125372 105667932 109158064 192212084 202506108 214625360 260071380	8 520 521 522 524 526 109158064 202506108 260071380

7. Sample solution (Java):

```
import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int adist = 0, anum = 0, bdist = 0, bnum = 0;

int[] all = new int[100];

int[] num = new int[2*n];

int[] a = new int[100];

int[] b = new int[100];

boolean[] isA = new boolean[2*n];
```

```
for (int i = 0; i < 100; i ++) {
all[i] = 0;
}
for (int i = 0; i < 2*n; i ++) {
int next = scanner.nextInt();
all[next] += 1;
num[i] = next;
}
boolean nowA = true, nowB = true;
for (int i = 10; i < 100; i ++) {
if(all[i] > 1) {
adist++;
bdist++;
if (all[i] % 2 == 0) {
a[i] = all[i] / 2;
} else {
if (nowA) {
a[i] = all[i] / 2 + 1;
nowA = false;
} else {
a[i] = all[i] / 2;
nowA = true;
}
}
} else if (all[i] == 1) {
if (nowB) {
bdist ++;
a[i] = 0;
```

```
nowB = false;
} else {
adist ++;
a[i] = 1;
nowB = true;
}
}
}
for (int i = 0; i < 2*n; i++) {
if (a[num[i]] > 0) {
isA[i] = true;
a[num[i]] --;
} else {
isA[i] = false;
}
}
System.out.println(adist * bdist);
for (int i = 0; i < 2 * n - 1; i++) {
System.out.print(isA[i] ? "1 " : "2 ");
}
System.out.println(isA[2*n - 1] ? "1" : "2");
scanner.close();
}
}
```

HARD LEVEL

1

MSSUM1

1. **Problem statement:**

You've got an array of N elements (indexed from 1 to N). There are Q queries, and each query is represented by two integers L, R (1 ≤ L ≤ R ≤ N).

A reply to a query is the sum of all array elements with indexes from L to R, inclusive.

You need to maximize the sum of query replies by ordering the array elements.

2. Input Format:

The first line contains two integers N and Q — the number of elements in the array and the number of queries, correspondingly.

The next line contains N integers of ai — the array elements.

Each of the following Q lines contains two integers L and R — the i-th query.

Integers in every line are separated by a space.

3. Output Format:

On a single line print out a single integer — the sum of replies in the reordered array.

4. Constraints:

$1 \le N \le 2 * 100\,000$

$1 \le Q \le 2* 100\,000$

$1 \le a_i \le 2 * 100\,000$

$1 \le L \le R \le N$

5. Samples Input/Output:

	Input data	Output data
1	3 3 5 3 2 1 2 2 3 1 3	25
2	5 3 5 2 4 1 3 1 5 2 3 2 3	33

6. Test Cases:

	Input data	Output data
1	16 13 40 32 15 16 35 36 45 23 30 42 25 8 29 21 39 23 2 9 3 11 8 9 4 14 1 6 5 10 5 14 5 11 13 13	2838

	2 8 9 16 6 10 7 8	
2	34 21 23 38 16 49 44 50 48 34 33 19 18 31 11 15 20 47 44 30 39 33 45 46 1 13 27 16 31 36 17 23 38 5 30 16 8 16 14 27 8 26 1 8 5 6 23 28 4 33 13 30 12 30 11 30 9 21 1 14 15 22 4 11 5 24 8 20 17 33 6 9 3 14 25 34 10 17	9382
3	1 1 1 1 1	1

4	22 7 44 41 40 41 37 42 46 16 50 47 30 7 32 6 20 14 47 25 11 1 35 12 1 10 10 20 4 5 1 8 2 12 8 8 2 16	2202
5	3 2 1 2 3 2 3 1 2	9
6	5 3 45 19 37 12 11 1 2 3 4 1 3	214
7	6 4 3 5 9 1 2 4 1 5 1 4 1 3	**76**

	1 2	

7. Sample solution (Java):

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
int q = sc.nextInt();
long[] arr = new long[n];
for (int i = 0; i < n; i++) arr[i] = sc.nextLong();
long[] qq = new long[n];
for (int i = 0; i < q; i++) {
qq[sc.nextInt() - 1] += 1;
qq[sc.nextInt()] -= 1;
}
long[] bb = new long[n];
long c = 0;
for (int i = 0; i < n; i++) {
c += qq[i];
bb[i] = c;
}
Arrays.sort(bb);
Arrays.sort(arr);
long sum = 0;
for (int i = 0; i < n; i++) {
sum += bb[i] * arr[i];
}
System.out.println(sum);
}
```

Count Of Divisors

1. **Problem statement:**

Find the smallest positive integer that has *n* divisors. It is guaranteed that the answer will be less than 10^18.

2. Input Format:

The first line contains an integer *n*.

3. Output Format

Print out the smallest integer that has exactly *n* divisors.

4. Constraints:

$1 \leq n \leq 1000$

5. Samples Input/Output:

	Input data	Output data
1	4	6
2	6	12

6. Test Cases:

	Input data	Output data
1	15	144
2	20	240
3	940	199495389743677440
4	1000	**810810000**
5	11	1024

6	21	576
7	500	**62370000**

7. Sample solution (Java):

```
import java.io.*;
import java.util.*;
import java.lang.*;

public class SampleClass {

final int p[] = {2,3,5,7,11,13,17,19,23,29};
final long inf = Long.MAX_VALUE;
long[][] dp;

long solve(int x,int rem) {
if (rem == 1)
return 1l;

long mn = inf, mul = p[x];

for (int i = 1; i < rem; i++, mul *= p[x]) {
if (rem % (i+1) == 0) {
double up = Math.log(1000000000000000000.0 / solve(x+1, rem/(i+1))) / Math.log(p[x]);
if (i <= up) {
mn = Math.min(mn, solve(x+1, rem/(i+1)) * mul);
}
}
}

return mn;
```

```
}

public static void main(String args[]) {

Scanner in = new Scanner(System.in);

int n = in.nextInt();

dp = new long[12][n+5];

System.out.println( solve(0, n) );
}
}
```

3

4th Dimension Aliens

1. **Problem statement:**

Aliens from the 4th Dimension have come to Earth with unknown intentions. They speak a strange unknown language, but scientists have found out that their ABC has all the English characters, except for vowels (including Y), and also, that it has two additional characters: one of them sounds similar to the German "ö" and one sounds similar to the Russian "ë". It is also known that the aliens don't use commas or uppercase characters. You need to write a program that translates text from English to the language of the 4th Dimension.

2. Input Format

Any string with Latin characters, periods, commas, "!" and "?".

3. Output Format:

Print out a string (UTF-8) with Latin consonant characters (including "y"), "ö", "ë", periods, "!" and "?"

4. Constraints:

Input data should contains only Latin characters, periods, commas, "!" and "?".

5. Samples Input/Output:

	Input data	Output data
1	Hello, dear aliens! We are so glad to see you there!	l ël hëö ëö dr ööë ln ös! wë r öë sëö lö gd tëö ë së ëö yöë ë hr të!
2	Please, don't kill us	ëö ls pë n ëö' dt ööl kl öës

6. Test Cases:

	Input data	Output data
1	We will give you our women, just don't attack, please!	wë ööl wl ööv gë ëö yöë öë ëör m ëöë wn öës jt n ëö' dt tö tc ök ëö ls pë!
2	We can be a beautiful, gorgeous friends.	wë ö cn bë ö t öëöö öf ëöë bl gë rëö ëööë gs ë öön rd fs.
3	What about a beer? Do you have a beer in your 4th dimension?	hö wt ëö böë öt ö ëë br? dëö ëö yöë öv hë ö ëë br öön ëööë yr t 4h n ës möö ööëö dn?
4	Yes, we will obey	ë ys wë ööl wl bë ëöy
5	Maybe, we can sell you something? Money, all that stuff, do you understand?	y öb më wë ö cn ël sl ëö yöë t ëh möö ëön sg? n ëöë my l öl hö tt öë tf sf dëö ëö yöë rs ët dö nn öëd?
6	We don't understand your language.	wë n ëö' dt rs ët dö nn öëd ëööë yr göë nö ög lë.
7	Please, save our planet!	ëö ls pë öv së öë ëör ön lë pt!

7. Sample solution (Java):

```
 import java.util.*;

class Main
{
    public static void main (String[] args) throws java.lang.Exception
    {
        Scanner cin = new Scanner(System.in);
        String[] words = cin.nextLine().split(" ");
        String resultString = "";

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

            String word = words[i].toLowerCase();
            String endSymbol = "";

            if (word.indexOf(",")>-1) {
                word = word.replace(",", "");
            }

            if (word.indexOf(".")>-1) {
                endSymbol = ".";
                word = word.replace(".", "");
            }

            if (word.indexOf("!")>-1) {
                endSymbol = "!";
                word = word.replace("!", "");
            }

            if (word.indexOf("?")>-1) {
                endSymbol = "?";
                word = word.replace("?", "");
            }

            String[] characters = word.split("");
            String resultWord = "";

            if(characters.length % 2 ==1) {
                resultWord += characters[characters.length/2] + " ";
            }
```

```
            for (int j = characters.length/2-1; j>=0; j--) {
                resultWord += characters[j] + characters[characters.length -
j - 1] + " ";
            }

            if (!endSymbol.equals("")) {
                resultWord = resultWord.substring(0, resultWord.length()-1)
+ endSymbol + " ";
            }

            resultWord = resultWord.replace("a", "ö");
            resultWord = resultWord.replace("e", "ë");
            resultWord = resultWord.replace("i", "öö");
            resultWord = resultWord.replace("o", "ëö");
            resultWord = resultWord.replace("u", "öë");

            resultString += resultWord;
        }

        System.out.print(resultString + "\n");
    }
}
```

4

Bernie and Formula 1

1. **Problem statement:**

As a big fan of Formula 1, Bernie is very happy to help with selling tickets for the next Grand Prix that takes place in his city. Unfortunately, due to the financial crisis in his country, there are only bills of 10 and 20 Euro. Each ticket for the race costs 10 Euro, so if someone wants to buy a ticket with a 20 Euro bill, Bernie will need to give them the change - 10 Euro. Bernie realizes that with only a few 10 Euro bills, selling tickets can be a problem. Bernie has some information about customers, and he needs your help. Precisely N + M people come to him for tickets. N of them will come with 10 EUR bills, and the remaining M of them will come with 20 EUR bills. Now Bernie has K 10 Euro bills that, if necessary, can be used for giving the change. All the N + M people will come to the ticket shop at random, so any sequence is equally probable. Find the probability that the ticket sale will go with no problems with money change, meaning that Bernie will have the change for each person with a 20 EUR bill.

2. Input Format:

N, *M, K*.

3. Output Format:

Probability

4. Constraints:

$0 \le N, M \le 100000, 0 \le K \le 10$

5. Samples Input/Output:

	Input data	Output data
1	5 3 1	0.857143
2	0 5 5	1.000000

6. Test Cases:

	Input data	Output data
1	0 1 0	0.000000
2	95105 76851 10	0.904215
3	60503 53620 1	0.214637
4	25902 30390 2	0.000000
5	53475 7159 4	0.999957
6	18874 21756 8	0.000000

7	84273 98526 10	0.000000

7. Sample solution (Java):

```
import java.util.*;
import java.util.Map.Entry;
import java.io.*;
import java.lang.*;
import java.math.*;

import static java.lang.Math.*;

public class Solution implements Runnable {

    void solve() throws Exception {
        int n = sc.nextInt();
        int m = sc.nextInt();
        int k = sc.nextInt();
        if (k >= m) {
            out.println("1.0");
            return;
        }
        if (n + k < m) {
            out.println("0.0");
            return;
        }
        double res = 1.0;
        for (int i = 0; i <= k; i++) {
            res = res * (m - i);
            res = res / (n + 1 + i);
        }
        out.format("%.10f%n", 1.0 - res);
    }

    BufferedReader in;
    PrintWriter out;
    FastScanner sc;

    final String INPUT_FILE = "stdin";
    final String OUTPUT_FILE = "stdout";
```

```
    static Throwable throwable;

    public static void main(String[] args) throws Throwable {
        Thread thread = new Thread(null, new Solution(), "", (1 << 26));
        thread.start();
        thread.join();
        thread.run();
        if (throwable != null)
            throw throwable;
    }


    public void run() {
        try {
            if (INPUT_FILE.equals("stdin"))
                in = new BufferedReader(new InputStreamReader(System.in));
            else
                in = new BufferedReader(new FileReader(INPUT_FILE));
            if (OUTPUT_FILE.equals("stdout"))
                out = new PrintWriter(System.out);
            else
                out = new PrintWriter(new FileWriter(OUTPUT_FILE));
            sc = new FastScanner(in);
            solve();
        } catch (Exception e) {
            throwable = e;
        } finally {
            out.close();
        }
    }
}

class FastScanner {
    BufferedReader reader;
    StringTokenizer strTok;

    FastScanner(BufferedReader reader) {
        this.reader = reader;
    }
```

```
    public String nextToken() throws Exception {
        while (strTok == null || !strTok.hasMoreTokens())
            strTok = new StringTokenizer(reader.readLine());
        return strTok.nextToken();
    }

    public boolean EOF() throws Exception {
        if (strTok != null && strTok.hasMoreTokens()) {
            return false;
        } else {
            String line = reader.readLine();
            if (line == null)
                return true;
            strTok = new StringTokenizer(line);
            return false;
        }
    }

    public int nextInt() throws Exception {
        return Integer.parseInt(nextToken());
    }

    public long nextLong() throws Exception {
        return Long.parseLong(nextToken());
    }

    public double nextDouble() throws Exception {
        return Double.parseDouble(nextToken());
    }
}
```

5

Changing Root of tree

1. **Problem statement:**

There are *n* nodes on a tree. Each node has its index — an integer number from 1 to *n*. The main node(the tree's root) has the index of *r*1. All nodes besides the main one have a parent. When we change the main node, some parent nodes can change as well. You need to output a new list of parent nodes using the information you have on both the old root and the new one.

2. Input Format:

The first line contains three space-delimited integers *n*, *r*1, *r*2 — the amount of nodes in the tree, the index of the old root and the index of the new one, correspondingly.

The following line contains *n* - 1 space-delimited integers — the old representation of the tree. For each node, apart from *r*1, there is an integer *pi given* — the index of its parent node. All the nodes are described in order of increasing indexes.

3. Output Format:

Output *n* - 1 numbers — a new representation of the tree in the same format.

4. Constraints:

$2 \le n \le 5 \cdot 104, 1 \le r1 \ne r2 \le n$

5. Samples Input/Output:

	Input data	Output data
1	3 2 3 2 2	2 3
2	6 2 4 6 1 2 4 2	6 4 1 4 2

6. Test Cases:

	Input data	Output data
1	7 7 6 7 7 5 5 7 7	7 7 5 5 7 6

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

7. Sample solution (Java):

```
import java.util.Scanner;
import java.util.ArrayList;

public class CFD {
static final int LE = 100666;
static ArrayList < Integer > [] H = new ArrayList[LE];
static boolean[] viz = new boolean[LE];
static int[] father = new int[LE];

public static void main(String[] Args) {
Scanner f = new Scanner(System.in);
int N = f.nextInt();
int r1 = f.nextInt(), r2 = f.nextInt();
```

```
for (int i = 1; i <= N; ++i) H[i] = new ArrayList < Integer > ();

for (int i = 1; i <= N; ++i) {
if (i == r1) continue;
int nd = f.nextInt();
H[nd].add(i);
H[i].add(nd);
}

dfs(r2);

for (int i = 1; i <= N; ++i)
if (i != r2)
System.out.print(father[i] + " ");

}

static void dfs(int nod) {
int N = H[nod].size();
int i;
viz[nod] = true;
for (i = 0; i < N; ++i)
if (viz[H[nod].get(i)] == false) {
dfs(H[nod].get(i));
father[H[nod].get(i)] = nod;
}
}
}
```

6

Colored Balls

1. **Problem statement:**

Yamashi has a bag with n colored balls, painted in k different colors. Colors are numbered from 1 to k. Balls of the same color are indistinguishable from each other. The young man pulls out the balls from the bag one by one until the bag is empty. He notices that for all i from 1 to k - 1 he pulled out the last ball of color i before pulled out the last ball of color i + 1. Now he's wondering how many different color sequences for the balls he pulled out of the bag there are, that would satisfy this condition.

2. Input Format:

The first line of input contains an integer k - the number of colors.

It is followed by k lines. The i-th line contains the number ci, the number of balls of color i.

3. Output Format:

Print out a single integer - the remainder of dividing the number of different ball color sequences that satisfy the given condition by 1,000,000,007.

4. Constraints:

$1 \le k \le 1000$

$1 \le c_i \le 1000$

The total number of balls does not exceed 1000.

5. Samples Input/Output:

	Input data	Output data
1	3 2 2 1	3
2	4 1 2 3 4	1680

6. Test Cases:

	Input data	Output data
1	10 100 100 100 100 100 100 100 100 100 100	12520708
2	5 10 10 10 10 10	425711769
3	11 291 381 126 39 19 20 3 1 20 45 2	902382672

4	1 1	1
5	3 343 317 337	691446102
6	1 5	1
7	5 312 76 12 98 54	**726731916**

In the first example we have two balls of color 1, two balls of color 2 and one ball of color 3. The three possible methods are:

1 2 1 2 3

1 1 2 2 3

2 1 1 2 3

7. Sample solution (Java):

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

```
public class ColoredBalls {
public static int mod = 1000000007;
public static int MAXN = 1010;

public static void main (String[] args) {
Scanner in = new Scanner(System.in);
PrintWriter out = new PrintWriter(System.out, true);

long[][] comb = new long[MAXN][MAXN];
comb[0][0] = 1;
for (int i = 1; i < MAXN; i++) {
comb[i][0] = 1;
for (int j = 1; j <= i; j++) {
comb[i][j] = (comb[i-1][j] + comb[i-1][j-1]) % mod;
}
}

int K = in.nextInt();
int[] color = new int[K];
for (int i = 0; i < K; i++) color[i] = in.nextInt();

long res = 1;
int total = 0;
for (int i = 0; i < K; i++) {
res = (res * comb[total + color[i] - 1][color[i] - 1]) % mod;
total += color[i];
}

out.println(res);
out.close();
System.exit(0);
}
}
```

7

Equality

1. **Problem statement:**

You are given an array of integer values with a length of N.
Calculate how many pairs of i and j exist, so that the sum of elements with indexes from i to j inclusive would be equal to the sum of all thel remaining array elements. Sectors from i to j can not be empty or take up the whole array.

2. Input Format:

The first input line contains an integer N — the amount of array elements.

The second line contains n space-delimited numbers $m_1,m_2,...m_n$ — the array integers elements.

3. Output Format:

Print out a single number – such pairs of i and j, where j >= i, that the sum of elements with indexes from i to j inclusive would be equal to the sum of all the remaining array elements

4. Constraints:

$1 \leq n \leq 10^5$

$-10^4 \leq m_i \leq 10^4$

5. Samples Input/Output:

	Input	Output
1	9 1 5 -6 7 9 -16 0 -2 2	3
2	3 1 1 1	0

6. Test cases

	Input	Output
1	2	1

	0 0	
2	4 100 1 10 111	1
3	10 0 4 -3 0 -2 2 -3 -3 2 5	3
4	10 0 0 0 0 0 0 0 0 0 0	9
5	10 3 -1 -3 -1 3 -2 0 3 1 -2	0
6	1 -10000	0
7	10 0 4 -3 0 -2 2 -3 -3 2 5	3
8	5 1 2 3 4 5	0

7. Solution (Java)

```
import java.util.Scanner;
public class SampleClass {
public static void main(String args[]) {
Scanner in = new Scanner(System.in);
int n = in .nextInt();
int a[] = new int[n];
int s1 = 0;
int s2 = 0;
for (int i = 0; i < n; i++) {
```

```
a[i] = in .nextInt();
s2 += a[i];
}
int ans = 0;
for (int i = 0; i < n - 1; i++) {
s1 += a[i];
s2 -= a[i];
if (s1 == s2) ans++;
}
System.out.println(ans);
}
}
```

8

Factorial game

1. **Problem statement:**

Two soldiers are playing a game. First, one soldier chooses a positive integer n, and tells the second soldier what it is. The second soldier tries to make the largest number of moves. Each turn, the soldier chooses a positive integer x> 1, such that n is divisible by x, and replaces the number n with the number of n / x. When n becomes equal to 1 and there are no more possible moves, then the game ends and the second soldier's score equals the number of moves he made.To make the game more interesting, for the n number, the solider chooses a number of the form a! / B! for some positive integers a and b (a ≥ b). Here, k! stands for the factorial of k, which is defined as the product of all positive integers that do not exceed k.

Determine the maximum possible score the second soldier can have.

2. Input Format:

First line of input consists of a single integer *t* denoting the number of games the soldiers play.

it is followed by *t* lines, each containing a pair of integers *a* and *b* defining the value of *n* for each game.

3. Output Format:

For each game, print out the maximum score the second soldier can have.

4. Constraints:

$1 \le t \le 10^6$

$1 \le b \le a \le 5 * 10 ^ 6$

5. Samples Input/Output:

	Input data	Output data
1	2 3 1 6 3	2 5
2	8 7 1 263 262 1000003 1000002 5000000 4999995 2 1 7 4 12345 12345 10 1	8 1 1 23 1 4 0 15

6. Test Cases:

	Input data	Output data
1	4 88 16 18 2 40 33 88 51	178 31 17 96
2	5 5497 1362 5136 4824 3232 1938 8431 2494 9119 2005	13191 1017 4097 19267 23078
3	5 4648623 1096796 1990207 701571 1126763 359601 1966159 77052 2619920 1180182	13381341 4792699 2821063 6965037 5389831
4	10 88 16 18 2	178 31 17

	40 33 88 51 76 44 37 14 43 27 54 38 23 12 87 14	96 82 54 37 40 22 180
5	1000000 3273469 3116112 3213887 1972478 1736566 1197799 4424234 1904213 3375740 2963175 4365035 1343603 1370672 833609 2530495 688551 4263799 3967077 4433220 1284869	594692 4674125 2008341 9517712 1558991 11386392 1991312 6870259 1126202 11864002
6	1000000 4841037 402161 4046970 3177996 1551526 440080 4860424 2128654 3277950 1619532 4383685 1212656 3833477 2548771 4397108 3449937 2606295 2463186 3298628 2506447 4152484 1342888	16668686 3290832 4109223 10335081 6236861 11944190 4854656 3591985 538720 2988859 10581860

7	1000000 1984351 564437 579442 571527 3530317 3465468 3569061 1802934 3725628 228498 2903576 1361532 4786697 3159697 1316825 168340 4601660 4351150 3268388 787575	5273939 28985 245480 6652755 13060730 5784886 6170889 4216321 952131 9289820

7. Sample solution (Java):

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;


public class SampleClass {


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

 BufferedReader isr = new BufferedReader(new InputStreamReader(System.in));

 int[] P = new int [5000001];
 for(int k =2;k*k<=P.length;k++){
 if(P[k]==0) {
```

```
for(int j = k*k;j<P.length;j+=k)P[j]=1;
}
}
int pn=0;
int[]PP = new int[5000000];
for(int i=2;i<P.length;i++)if(P[i]==0)PP[pn++]=i;
int[]D = new int[5000001];
for(int x=2;x<D.length;x++) {
int y = x;
for(int i=0;i<pn && PP[i]*PP[i]<=y;i++){
int p = PP[i];
while(y%p==0){D[x]++;y/=p;}
}
if(y>1)D[x]++;
}
long[]SD = new long[D.length];
for(int i=2;i<SD.length;i++){
  SD[i]=D[i]+SD[i-1];
}
StringBuffer sb = new StringBuffer();
int t = Integer.parseInt(isr.readLine());
for(int i=0;i<t;i++){
String[]s = isr.readLine().split(" ");
int a = Integer.parseInt(s[0]);
int b =Integer.parseInt(s[1]);
  sb.append(SD[a]-SD[b]);
  sb.append('\n');
}
System.out.println(sb);
}


}
```

9

Flowers, Girls and Grandma

1. **Problem statement:**

There are *n* girls. They are all sitting at a table, so that girls *i* and *i* + 1 are neighbours for all *i* from 1 to *n* - 1. Girls *n* and 1 are neighbours too.

Each girl will grow some number of flowers *si*. For the *i*-th girl value *si* is a random integer equiprobably chosen in range from *li* to *ri*. Grandma has her favourite prime number *p*, and she really likes it! If for any pair of neighbouring girls *i* and *j* , the product of *si*·*sj* is divisible by *p*, that makes Grandma so happy she gives 1000 candies to each of these girls.

At the end of the day, girls sum all candies Grandma gave them. Find the expectation of this value.

2. Input Format:

The first line contains two space-delimited integers *n* and *p* — the number of girls and Grandma's favourite prime number. It is guaranteed that *p* is prime.

The *i*-th of the following *n* lines contains information about the *i*-th girl — two space-delimited integers *li* and *ri* , the range of flowers girl *i* can grow. Remember that *si* is chosen equiprobably among all integers from *li* to *ri*, inclusive.

3. Output Format:

Print out a single real number — the expected number of candies that the girls will receive in total. Your answer will be considered correct if its absolute or relative error does not exceed 10^ - 6.

4. Constraints:

3 ≤ *n* ≤ 100 000, 2 ≤ *p* ≤ 10^9

1 ≤ *li* ≤ *ri* ≤ 10^9

5. Samples Input/Output:

	Input data	Output data
1	3 2	

	1 2 420 421 420420 420421	4500.0
2	3 5 1 4 2 3 11 14	0.0

6. Test Cases:

	Input data	Output data
1	3 3 3 3 2 4 1 1	4666.666667
2	5 5 5 204 420 469 417 480 442 443 44 46	3451.25
3	3 2 2 2 3 3 4 4	6000.0

4	6 7 8 13 14 14 8 13 14 14 8 13 14 14	12000.0
5	3 7 7 14 700000000 700000007 420 4200	2304.251521
6	5 999999937 999999935 999999936 999999937 999999938 999999939 999999940 999999941 999999942 999999943 999999944	2000.000000
7	3 2 1 1 1 2 1 1	2000.000000

7. Sample solution (Java):

```
import java.util.Scanner;
public class C {
public static void main(String [] args){
```

```
Scanner in = new Scanner(System.in);
int N = in.nextInt();
int p = in.nextInt();
double [] a = new double[N+1];
for (int i = 0; i < N; i++){
int l = in.nextInt();
int r = in.nextInt();
a[i] = (double)(r/p - (l-1)/p) / (r-l+1);
}

a[N] = a[0];
double res = 0;

for (int i = 0; i < N; i++){
res += a[i] + (1-a[i])*a[i+1];
}

System.out.printf("%.6f", res*2000);

}
}
```

10

Sort

1. **Problem statement:**

Roman wants to win the world championship in programming and decided to study *N* subjects (for convenience we will number these subjects from 1 to *N*). For this purpose, Roman developed a studying plan. But it turned out that certain subjects can be studied only after he has completed other subjects. So Roman's coach researched all the subjects and prepared a list of *M* limitations in the form "*si ui*" (1 ≤ *si*, *ui* ≤ *N*; *i* = 1, 2, …, *M*), which means that subject *si* must be studied before subject *ui*.

Your task is to verify if the order of subjects being studied by Roman is correct.

Note: It may appear impossible to find the correct order of subjects with the given limitations. In this case, any subject order developed by Roman is incorrect.

2. Input Format:

The first line contains two integers *N* and *M* (*N* is the number of subjects, *M* is the number of limitations). The next *M* lines contain pairs *si*, *ui*, which describe the order of subjects: subject *si* must be studied before *ui*. Furthermore, there is a sequence of *N* unique numbers ranging from 1 to *N* — the proposed study plan.

3. Output Format:

Output a single word “YES” or “NO”. “YES” means that the proposed order is correct and does not contradict the given limitations. “NO” means that the order is incorrect.

4. Constraints:

$1 \le N \le 1000$; $0 \le M \le 100000$.

5. Samples Input/Output:

	Input data	Output data
1	5 6 1 3 1 4 3 5 5 2 4 2 1 2 1 3 4 5 2	YES
2	5 6 1 3 1 4 3 5 5 2 4 2 1 2 1 2 4 5 3	NO

6. Test Cases:

	Input data	Output data
1	3 2 1 2 1 2 2 2 2	YES
2	4 2 1 1 1 1 1 1 1 1	NO
3	1 1 1 1 1	YES
4	3 3 1 2 2 3 3 2 3 3 3	YES
5	5 5 1 2	NO

	2 3 3 2 5 5 2 3 1 2 3 4 5	
6	3 4 2 2 2 1 1 2 2 1 2 2 2	NO
7	7 8 5 6 4 3 2 3 6 4 3 4 1 2 4 4 3 3	NO

	2 2 1 2	

7. Sample solution (Java):

```
import java.io.*;
public class Problem1280 implements Runnable {
private BufferedReader bufferedReader;
private StreamTokenizer in;
private PrintWriter out;
public Problem1280() {
this(System.in, System.out);
}
public Problem1280(InputStream inputStream, OutputStream outputStream) {
bufferedReader = new BufferedReader(new InputStreamReader(inputStream));
in = new StreamTokenizer(bufferedReader);
out = new PrintWriter(new OutputStreamWriter(outputStream));
}
public static void main(String[] args) throws IOException {
new Problem1280().run();
}
public void run() {
try {
solve();
out.flush();
} catch (Exception e) {
throw new RuntimeException(e);
}
}
private double readNumber() throws IOException {
int nextToken = in.nextToken();
if (nextToken == StreamTokenizer.TT_NUMBER) {
return in.nval;
}
```

```
throw new IllegalStateException("Number expected. Found: " + nextToken);
}
private String readWord() throws IOException {
int nextToken = in.nextToken();
if (nextToken == StreamTokenizer.TT_WORD) {
return in.sval;
}
throw new IllegalStateException("Word expected. Found: " + nextToken);
}
//TODO global variables
private void solve() throws Exception {
int n = (int) readNumber();
int m = (int) readNumber();
boolean[][] a = new boolean[n][n];
int[] b = new int[n];
for (int i = 0; i < m; i++) {
int before = (int) readNumber() - 1;
int after = (int) readNumber() - 1;
a[before][after] = true;
}
for (int i = 0; i < n; i++) {
b[i] = (int) readNumber() - 1;
for (int j = 0; j < i; j++) {
if (a[b[i]][b[j]]) {
out.println("NO");
return;
}
}
}
out.println("YES");
}
}
```

11

SQUARE

1. **Problem statement**

The number x is called a square root of number a modulo n (root (a, n)) if and only if x = a * x (mod n) . Write a program that finds all the values for square roots of a modulo n.

2. Input Format:

First line contains single integer k - count of cases. Next k lines contains the integers a and n.

3. Output Format:

The program must compute the root of all possible values of (a, n) in the range from 1 to n - 1 and output them in ascending order on the same line, each separated by a single space . If the roots of the current test do not exist , the program should print out a single line message 'No root'.

4. Constraints:

1 ≤ a, n ≤ 32767 , n prime , a and n are relatively prime.

5. Samples Input/Output:

	Input data	**Output data**
1	1 4 17	2 15
2	1 3 7	No root

6. Test Cases:

	Input data	**Output data**
1	1	3 4

	2 7	
2	1 14 31	13 18
3	1 10007 20011	5382 14629
4	2 5 29 3 3001	11 18 1086 1915
5	3 14 133 14 97 13 137	No root No root No root
6	3 14 1097 4 221 851 31451	No root 15 206 7310 24141
7	3 33 1811 28 1663 67 2699	417 1394 No root No root

7. Sample solution (Java):

```
import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
public class Main
{
public static void main(String[] arg)
{
new Main();
}
Main()
{
Scanner cin=new Scanner(System.in);
PrintWriter cout=new PrintWriter(System.out);
int k=cin.nextInt();
int[] a=new int[k],n=new int[k];
ArrayList<ArrayList<Integer> > map=new ArrayList<ArrayList<Integer> >(32768);
for (int i=0;i<32768;i++) map.add(null);
for (int i=0;i<k;i++)
{
a[i]=cin.nextInt();
n[i]=cin.nextInt();
a[i]%=n[i];
if (map.get(n[i])==null) map.set(n[i],new ArrayList<Integer>());
map.get(n[i]).add(i);
}
int[] ans=new int[k];
```

```
for (int prime=0;prime<map.size();prime++) if (map.get(prime)!=null)
{
ArrayList<Integer> indexs=map.get(prime);
int[] get=new int[prime];
Arrays.fill(get,-1);
for (int i=0;i<=prime/2;i++)
get[(i*i)%prime]=i;
for (int index:indexs) ans[index]=get[a[index]];
}
for (int i=0;i<k;i++)
{
if (ans[i]==-1) cout.println("No root");
else
{
if (ans[i]==0||ans[i]+ans[i]==n[i]) cout.println(ans[i]);
else cout.println(ans[i]+" "+(n[i]-ans[i]));
}
}
cout.close();
cin.close();
}
}
```

Strings Problemo

1. Problem statement:

A boy named Valera loves strings. But he loves it even more when strings are identical. That's why, in his free time, Valera plays the following game all by himself. He takes two random strings consisting of lowercase Latin letters and tries to turn them into two strings equal to each other. According to the rules of his game, during each turn Valera is allowed to replace a single random character Ai to a random character Bi in one of the strings, but for every turn he has to pay some amount of money equal to Wi. He is allowed to take an unlimited amount of turns. Being a very thrifty boy that never wastes his money, Valera has asked you, an experienced programmer, to help him answer the following question: what is the lowest amount of money that Valera is required to have in order to get identical strings?.

2. Input Format:

In the first two lines of the input file are two initial non-empty strings s and t consisting of lowercase Latin letters.. The next line contains an integer n - the number of possible replacements. The following n lines contain two characters Ai and Bi, which are lowercase letters, as well as an integer Wi, meaning that it is allowed to replace the symbol Ai with the symbol Bi in any of the strings, spending the amount of money equal to Wi .

3. Output Format:

If a solution exists, output the answer to the problem and the resulting string into the output file. Otherwise, output -1 on a single line. If there are multiple optimal solutions, output any of them.

4. Constraints:

The lengths of the two strings (s and t) do not exceed 10^5

$0 \leq n \leq 500$

$0 \leq Wi \leq 100$

5. Samples Input/Output:

	Input data	Output data
1	uayd uxxd 3 a x 8 x y 13 d c 3	21 uxyd
2	a b 3	2 b

	a b 2 a b 3 b a 5	

6. Test Cases:

	Input data	Output data
1	abc ab 6 a b 4 a b 7 b a 8 c b 11 c a 3 a c 0	-1
2	xhtuopq rtutbz 10 h x 10 x d 3 r u 4 u d 1 t o 100 o t 7 p e 1 e f 1 b f 2 z q 19	-1

3	abad abad 6 a c 3 b x 100 d e 7 r r 10 o t 17 a a 4	0 abad
4	bbad abxd 4 b a 7 a b 10 x a 0 d t 19	7 abad
5	abcd acer 6 b c 100 c b 10 c x 1 e x 3 c e 7 r d 11	25 abxd
6	abac cbad 7 a c 100	-1

	x y 21 b i 90 d e 89 c z 12 t r 66 a g 78	
7	wye upt 13 z z 5 e t 8 t f 2 f e 3 p l 16 l s 6 s q 13 y o 4 o q 0 u w 5 k m 14 m i 10 w u 12	49 wqe

7. Sample solution (Java):

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.*;

public class Main {
public static void main(String[] args) throws IOException {
FastReadWriteConsole console = new FastReadWriteConsole();
```

```
String a = console.nextString();
String b = console.nextString();
int n = console.nextInt();

int arr[][] = new int[26][26];
for(int i = 0; i < 26; i++){
for(int j = 0; j < 26; j++){
if(i == j)
arr[i][j] = 0;
else
arr[i][j] = Integer.MAX_VALUE;
}
}

for(int i = 0; i < n; i++) {
int from = console.nextChar() - 'a';
int to = console.nextChar() - 'a';
int value = console.nextInt();
arr[from][to] = Math.min(arr[from][to], value);
}
int sum = 0;
StringBuilder sb = new StringBuilder();
if(a.length() != b.length()){
console.print(-1);
console.close();
return;
}
Floid alg = new Floid();
double dist [][] = alg.floid(arr);
for(int i = 0; i < a.length(); i++){
if(a.charAt(i)!= b.charAt(i)){
int minValue = Integer.MAX_VALUE;
int index = -1;
for(int j = 0; j < 26; j++){
if(dist[a.charAt(i) - 'a'][j] != Integer.MAX_VALUE && dist[b.charAt(i) - 'a'][j] != Integer.MAX_VALUE){
```

```
if(dist[a.charAt(i) - 'a'][j] + dist[b.charAt(i) - 'a'][j] < minValue){
minValue = (int) (dist[a.charAt(i) - 'a'][j] + dist[b.charAt(i) - 'a'][j]);
index = j;
}
}
}
if(index == -1){
console.print(-1);
console.close();
return;
}else {
sb.append((char)(index + 'a'));
sum += minValue;
}
}else
sb.append(a.charAt(i));
}
console.print(sum + "\n");
console.print(sb);
console.close();
}
}
class Floid {

private double a[][] = null;

public double[][] floid(int adj[][]){
a = new double[adj.length][adj.length];
for(int i = 0; i < adj.length; i++){
for(int j = 0; j < adj.length; j++){
a[i][j] = adj[i][j];
}
}
for (int k = 0; k < a.length; k++) {
for (int i = 0; i < a.length; i++) {
```

```
for (int j = 0; j < a.length; j++) {
a[i][j] = Math.min(a[i][j], a[i][k] + a[k][j]);
}
}
}
return a;
}

}
class FastReadWriteConsole {

BufferedReader in;
StringTokenizer tok;
PrintWriter out;

public FastReadWriteConsole() {
in = new BufferedReader(new InputStreamReader(System.in));
out = new PrintWriter(System.out);
}

private String readToken() throws IOException {
while (tok == null || !tok.hasMoreTokens()) {
tok = new StringTokenizer(in.readLine());
}
return tok.nextToken();
}

public int nextInt() throws IOException {
return Integer.parseInt(readToken());
}

public int nextChar() throws IOException {
return readToken().charAt(0);
}

public String nextString() throws IOException {
return readToken();
```

```
}

public void print(Object obj){
out.print(obj.toString());
}

public void close() throws IOException {
in.close();
out.close();
}

}
```

13

Two Circles

1. **Problem statement:**

A flowerbed has many flowers and two fountains.

You have two circles in some places. And there are some points. You have to set such *r*1 and *r*2 values that all the points are covered by circles. That is, for each point, the distance between the point and the center of the first circle doesn't exceed *r*1, or the distance to the center of the second circle doesn't exceed *r*2. It's OK if some points are covered by both circles.

You need to set such *r*1 and *r*2 values that all the points are covered and the *r*12 + *r*22 is the minimum value possible. Find this minimum value.

2. Input Format:

The first line contains integers *n*, *x*1, *y*1, *x*2, *y*2 — the number of points, the coordinates of the first and the second circle.

Next, there are *n* lines. The *i*-th of these lines contains integers *xi* and *yi* — the coordinates of the *i*-th point.

It is guaranteed that all *n* + 2 points in the input data are distinct.

3. Output Format:

Print out the minimum possible value of $r1^2 + r2^2$. Note that in this problem the optimal answer is always an integer.

4. Constraints:

For first line: $1 \le n \le 2000, \ -10^7 \le x1, y1, x2, y2 \le 10^7$

For next n lines: $-10^7 \le xi, yi \le 10^7$

5. Samples Input/Output:

	Input data	Output data
1	2 -1 0 5 3 0 2 5 2	6
2	4 0 0 5 0 9 4 8 3 -1 0 1 4	33

6. Test Cases:

	Input data	Output data
1	5 -6 -4 0 10 -7 6 -9 7 -5 -1 -2 1	100

	-8 10	
2	10 -68 10 87 22 30 89 82 -97 -52 25 76 -22 -20 95 21 25 2 -3 45 -7 -98 -56 -15 16	22034
3	2 1 1 2 2 0 0 3 3	4
4	5 1 100 2 100 4 4 5 5 6 6 1 101 2 102	**9220**
5	1 1 1 2 1 1 2	1
6	4 -1000 -1000 1000 1000	**2720000**

	500 500 -500 500 -600 600 500 -500	
7	6 1 1 10 1 2 1 3 1 4 1 5 1 4 2 4 0	**16**

7. Sample solution (Java):

```
import java.util.Scanner;

public class Round340C {
public static void main(String[] args) {
Scanner in = new Scanner(System.in);
int numPoints = in.nextInt();
int x1 = in.nextInt();
int y1 = in.nextInt();
int x2 = in.nextInt();
int y2 = in.nextInt();
long[] dist1 = new long[numPoints];
long[] dist2 = new long[numPoints];
```

```
int x, y, i, j;
long r1Sq = 0, r2Sq = 0, result = 0;
for (i = 0; i < numPoints; i++) {
x = in.nextInt();
y = in.nextInt();
dist1[i] = ((long) (x - x1)) * ((long) (x - x1)) + ((long) (y - y1)) * ((long) (y - y1));
dist2[i] = ((long) (x - x2)) * ((long) (x - x2)) + ((long) (y - y2)) * ((long) (y - y2));
result = Math.max(result, dist2[i]);
}
for (i = 0; i < numPoints; i++) {
r1Sq = dist1[i];
r2Sq = 0;
for (j = 0; j < numPoints; j++) {
if (dist1[j] > r1Sq) r2Sq = Math.max(r2Sq, dist2[j]);
}
result = Math.min(result, r1Sq + r2Sq);
}
System.out.println(result);
}
}
```

14

Grasshopper

1. **Problem statement:**

There is a kind of grasshopper that can jump only vertically and horizontally, and they always jump exactly s centimeters far. In the center of some cell on a rectangular checkered board of n × m centimeters (the size of a single cell is 1 × 1 centimeters) there is a grasshopper of that very kind. It can jump around the board as much as necessary, it can even visit the same cell several times. The only limitation is that the Grasshopper can not jump off the board.

The Grasshopper can count the number of cells it can reach from its initial position (x, y). Let's call this number $d_{x, y}$. Your task is to find the number of these initial grasshopper positions (x, y), for which $d_{x, y}$ is maximum.

2. Input Format:

The first line contains three integers n, m, s - the length of the board, its width and the length of the grasshopper's jump respectively.

3. Output Format:

Print out a single integer number - the number of the grasshopper's initial positions for which $d_{x, y}$ is maximum.

4. Constraints:

1 ≤ n, m, s ≤ 106

5. Samples Input/Output:

	Input data	Output data
1	2 3 1000000	**6**
2	3 3 2	**4**

6. Test Cases:

	Input data	Output data
1	1 2 3	**2**
2	4 5 6	20
3	9 8 7	**8**
4	1 1 1	**1**
5	1000 1000 1000	1000000
6	1 1 2	1
7	1 1 1000000	1

7. Sample solution (Java):

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


public class Main{
    public static void main(String[] args){
        Scanner input=new Scanner(System.in);
        int n=input.nextInt();
        int m=input.nextInt();
        int s=input.nextInt();
        long u=(n-1)%s+1;
        long v=(m-1)%s+1;
        long stepx=(n-1)/s+1;
        long stepy=(m-1)/s+1;
        long ans=u*v*stepx*stepy;
        System.out.println(ans);
    }
}
```

15

Apartment

1. **Problem statement:**

At last, the day when Shalkan has enough money to buy his own apartment has come. He was offered a great option - an apartment in the city center for a great price.

The plan for the apartment Shalkan is offered to buy is a rectangle the size of n × m, divided into squares of size 1 × 1. Each of these squares contains either a wall (such squares are marked with a "*" in the plan), or a wall-free space (such squares are marked with al "." in the plan).

The apartment rooms are connected wall-free cell areas of maximum size,. Two cells are considered adjacent if they have a common side.

Shalkan’s had a dream to have an apartment in which all rooms would be rectangles. He asks you to calculate the minimum number of walls that need to be demolished to make his dream come true. In the process of demolishing the walls, several rooms can be joined into one.

2. Input Format:

The first line contains two integers n, m - the size of Shalkan’s apartment plan.

The following n lines contain m characters - the description of the apartment plan.

If a cell is marked with a "*" character, then there is a wall in the cell.

If a cel isl marked with a "." character, then it’s a wall-free part of the apartment, and the cell it refers to some of the rooms.

3. Output Format:

Print out n lines of n characters - Shalkan’s apartment plan after demolishing the minimum number of walls, when each room (the maximum size of a connected wall-free area), is a rectangle.

If there are several answers, output any of them.

4. Constraints:

$1 \le n, m \le 2000$

5. Samples Input/Output:

	Input data	Output data
1	5 5 .*.*. ***** .*.*. ***** .*.*.	.*.*. ***** .*.*. ***** .*.*.

2	6 7 ***.*.* ..*.*.* *.*.*.* *.*.*.* ..*...* *******	***...* ..*...* ..*...* ..*...* ..*...* *******

6. Test Cases:

	Input data	Output data
1	4 5 ....*** ..*..	....
2	6 6 ****** *.*..* *.**** ****.* *..*.* ******	****** *.*..* *.**** ****.* *..*.* ******
3	6 3 .*. **. *.* .** .**	.*. **. *.* .** **.

	**. .*.	.*.
4	4 4 .*.. .*** ***. *.*.	.*.. .*** ***. *.*.
5	1 1 .	.
6	1 2 .*	.*
7	4 4 .*.. *.*. **** ***.	.... **** ***.

7. Sample solution (Java):

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

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);
```

```
int n = scan.nextInt();
int m = scan.nextInt();
scan.nextLine();
int[][] a = new int[n+2][m+2];
int[][] sqs = new int[n+1][m+1];

ArrayList<Integer> p = new ArrayList<Integer>();
for (int i = 0; i<n; i++) {
String s = scan.nextLine();
for (int j = 0; j<m; j++) {
if (s.charAt(j) == '.') {
a[i+1][j+1] = 1;
sqs[i][j] += 1;
sqs[i+1][j] += 1;
sqs[i][j+1] += 1;
sqs[i+1][j+1] += 1;
}
}
}

for (int i = 0; i<n+1; i++) {
for (int j = 0; j<m+1; j++) {
if (sqs[i][j] == 3) {
p.add(i);
p.add(j);
}
}
}

ArrayList<Integer> p2 = new ArrayList<Integer>();
while (!p.isEmpty()) {
for (int q = 0; q<p.size(); q+=2) {
int x = p.get(q);
int y = p.get(q+1);
for (int i = x; i<x+2; i++) {
```

```
for (int j = y; j<y+2; j++) {
if (a[i][j] == 0) {
a[i][j] = 1;
for (int o = i-1; o<i+1; o++) {
for (int r = j-1; r<j+1; r++) {
sqs[o][r] += 1;
if (sqs[o][r] == 3) {
p2.add(o);
p2.add(r);
}
}
}
}
}
}
}
p = p2;
p2 = new ArrayList<Integer>();
}

for (int i = 1; i<n+1; i++) {
StringBuilder sb = new StringBuilder();
for (int j = 1; j<m+1; j++) {
if (a[i][j] == 1)
sb.append('.');
else
sb.append('*');
}
System.out.println(sb.toString());
}

scan.close();
}
}
```

16

Forgotten Password

1. **Problem statement:**

James forgot the password to his email. The only thing he remembers is that the password consists of n digits, with each digit being either 0 or 1. James made m attempts to enter the password. After each attempt, the email service informed him on how many password positions contained the right numbers. They never told him which positions contained wrong numbers. James was so unfortunate that he never entered a password with more than 5 right digets. Eventually, James got so confused he started thinking there is an error in the system and that it contradicts itself. Help James - count how many possible password options remain that do not contradict the system's prior responses.

2. Input Format:

The first line contains two integers n and m - the number of digits in the password and the number of attempts made by James. It is followed by m lines, each of which contains variables s_i and c_i - James' attempt (a string of n numbers 0 or 1) and the response of the email service (an integer from 0 to 5, inclusive) respectively.

3. Output Format:

Print out a single number - how many possible password options remain that do not contradict the system's prior responses.

4. Constraints:

$6 \le n \le 35,\ 1 \le m \le 10$

5. Samples Input/Output:

	Input data	Output data
1	6 2 000000 2 010100 4	6
2	6 3 000000 2 010100 4	0

	111100 0	

6. Test Cases:

	Input data	Output data
1	6 3 000000 2 010100 4 111100 2	1
2	6 1 101011 2	15
3	7 2 1011111 2 1001111 1	6
4	6 4 000110 2 010001 2 001111 2 001100 2	1
5	8 3 00111100 5 10100111 2 10110101 2	6
6	35 2	20

	00101101100111101110111010001101101 3 00111111100101010110111010001101101 3	
7	35 1 11000110100110101001100101001010110 2	595

7. Sample solution (Java):

```
import java.util.*;
public class Safe {
	static String[] pw;
	static int[] cnt;
	static int counter = 0;
	static int n;
	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);
		n = input.nextInt();
		int m = input.nextInt();
		pw = new String[m];
		cnt = new int[m];
		for (int i=0;i<m;i++){
			pw[i] = input.next();
			cnt[i] = input.nextInt();
		}
		go(0);
		System.out.println(counter);
	}
	public static void go (int b) {
		for (int i=0;i<cnt.length;i++)
			if (cnt[i] < 0 || n-b<cnt[i])
				return;
		if (b == n){
			counter++;
			return;
		}
		for (int i=0;i<cnt.length;i++)
			if (pw[i].charAt(b) == '0')
				cnt[i]--;
		go(b+1);
		for (int i=0;i<cnt.length;i++)
			if (pw[i].charAt(b) == '0')
				cnt[i]++;
		for (int i=0;i<cnt.length;i++)
			if (pw[i].charAt(b) == '1')
				cnt[i]--;
		go(b+1);
		for (int i=0;i<cnt.length;i++)
```

```
                    if (pw[i].charAt(b) == '1')
                        cnt[i]++;
        }
}
```

17

Apple Tree

1. **Problem statement:**

An apple tree has n vertices and its root is the vertex 1.

You want to make this tree multi-colored!

Your task is to process queries of two types:

1. Change the colors of all apples (verticles) in the subtree of the vertex *v* to the colour *c*.
2. Find the number of different colours in the subtree of the vertex *v*.

2. Input Format:

The first line contains two integers N - the number of vertices (apples), M - the number of the queries.

The second line contains *n* integers *c_{i}* — the colour of the *i*-th apple

.Each of the next *n* - 1 lines contains two integers *x_{j}*, *y_{j}* — the vertices of the *j*-th edge. It is guaranteed that you are given a correct undirected tree.

The last *m* lines contain the description of the queries.

Each description starts with the integer *t_{k}* — the type of the *k*-th query.

Queries of the first type contain two integers *v_{k}*, *c_{k}* — the number of the apple (vertex) whose subtree will be re-coloured with the colour *c_{k}*.

Queries of the second type contain the integer *v_{k}* — the number of the vertex for which subtree you need to find the number of different colours.

3. Output Format:

For each query of type 2 print out the integer *a* — the number of different colours in the subtree of the vertex given in the query.

Each number should be printed on a separate line according to the order of queriers appearing in the input data.

4. Constraints:

$1 \le xj, yj \le n$

$1 \le t\{k\} \le 2$

$1 \le v\{k\} \le n$

$1 \le v\_\{k\} \le n$

5. Samples Input/Output:

	Input data	Output data
1	7 10 1 1 1 1 1 1 1 1 2 1 3 1 4 3 5 3 6 3 7 1 3 2 2 1 1 4 3 2 1 1 2 5 2 1 1 6 4 2 1 2 2 2 3	2 3 4 5 1 2
2	23 30 1 2 2 6 5 3 2 1 1 1 2 4 5 3 4 4 3 3 3 3 3 4 6 1 2 1 3 1 4 2 5 2 6 3 7 3 8	6 1 3 3 2 1 2 3

	4 9 4 10 4 11 6 12 6 13 7 14 7 15 7 16 8 17 8 18 10 19 10 20 10 21 11 22 11 23 2 1 2 5 2 6 2 7 2 8 2 9 2 10 2 11 2 4 1 12 1 1 13 1 1 14 1 1 15 1 1 16 1 1 17 1 1 18 1 1 19 1 1 20 1 1 21 1 1 22 1 1 23 1 2 1	5 5 1 2 2 1 1 1 2 3

	2 5 2 6 2 7 2 8 2 9 2 10 2 11 2 4	

6. Test Cases:

	Input data	Output data
1	1 1 14 2 1	1
2	1 1 36 2 1	1
3	10 10 59 59 59 59 59 59 59 59 59 59 6 8 6 10 2 6 2 5 7 2 10 1	1 1 1

	4 2 7 3 9 1 1 8 59 2 8 1 3 59 1 4 59 1 8 59 1 2 59 1 5 59 1 10 59 2 2 2 5	
4	1 1 25 1 1 25	
5	1 1 3 2 1	1
6	1 1 12 1 1 43	
7	10 10 8 8 14 32 14 8 32 8 14 32 4 5 4 1 4 8 4 9 7 4 2 5	1 1 1 1 1

	3 5 4 6 10 4 2 2 1 9 8 1 1 40 1 7 32 1 4 8 2 8 1 1 8 2 2 2 8 2 4	

7. Source(Java):

```
import java.util.; import java.lang.; import java.io.*; import static
java.lang.Math.max; import static java.lang.Math.min; import static
java.lang.Math.abs; import java.util.Arrays;


/ Name of the class has to be "Main" only if the class is public. / public
class ValentineProject {


ArrayList<Integer>[] edge = new ArrayList[400005];


    long seg[] = new long[4 * 400001];


    int l[] = new int[400005];


    int r[] = new int[400005];


    int t = 0;


    int c[] = new int[400005];


    int a[] = new int[400005];


    long layer[] = new long[4*400001];
```

```
private void solve() throws IOException {
    int u,v,color, ty, ans;
    long mask;

    for(int i=1;i<=400000;++i){
      edge[i] = new ArrayList<>();
    }

    Arrays.fill(seg, 0);
    Arrays.fill(layer, -1);

    int n = nextInt();
    int m = nextInt();
    for(int i=1;i<=n;++i){
        c[i] = nextInt();
        c[i]--;
    }
    for(int i=1;i<n;++i){
        u = nextInt(); v = nextInt();
        edge[u].add(v);
        edge[v].add(u);
```

```
}
dfs(1,0);
build(1,1,n);

while( m > 0){
    m --;
    ty = nextInt();
    if( ty == 1){
        v = nextInt(); color = nextInt();
        update(1,1,n,l[v],r[v],color-1);
    }
    else{
        v = nextInt();
        mask = query(1,1,n,l[v], r[v]);
        ans = 0;
        while( mask > 0){
            ans++;
            mask -= mask&(-mask);
        }
        writer.println(ans);
    }
}
```

```
}

public void dfs(int vertex, int parent)
{
    l[vertex] = ++t;
    a[t] = vertex;
    Iterator<Integer> itr = edge[vertex].iterator();
    int vr;
    while( itr.hasNext() ){
        vr = itr.next();
        if( vr != parent){
            dfs( vr, vertex);
        }
    }
    r[vertex] = t;
}

public void build(int n,int it,int f)
{
    if(it == f){
        seg[n] = 1L<<c[a[f]];
```

```
        layer[n] = seg[n];
        return;
    }
    int m = (it + f)/2;
    build(n*2,it,m);
    build(n*2+1,m+1,f);
    seg[n] = seg[n*2] | seg[n*2+1];
}

public void update(int n,int it,int f,int q1,int q2,int color)
{
    if( it==q1 && f==q2){
        seg[n] = 1L << color;
        layer[n] = seg[n];
        return ;
    }

    int m = (it + f)/2, ln = n * 2, rn;
    rn = ln + 1;
    if(layer[n] != -1){
        seg[ln] = seg[rn] = layer[ln] = layer[rn] = layer[n];
        layer[n] = -1;
```

```
    }
    if( q2 <=m ){
        update(ln,it,m,q1,q2,color);
    }
    else if(q1>m)
        update(rn,m+1,f,q1,q2,color);
    else{
        update(ln,it,m,q1,m,color);
        update(rn,m+1,f,m+1,q2,color);
    }
    seg[n] = seg[ln] | seg[rn];
}

public long query(int n,int it,int f,int q1,int q2)
{
    if(layer[n] != -1)
        return layer[n];
    if(q1==it && f==q2)
        return seg[n];
    int m = (it+f)/2;
    if( q2 <= m)
        return query(n*2,it,m,q1,q2);
```

```
    else if ( q1 > m)
        return query(n*2+1,m+1,f,q1,q2);
    return ( query(n*2,it,m,q1,m) | query(n*2+1,m+1,f,m+1,q2));
}

public static void main(String[] args) {
    // TODO code application logic here
    new ValentineProject().run();
}

BufferedReader reader;
StringTokenizer tokenizer;
PrintWriter writer;

public void run() {
    try {
        reader = new BufferedReader(new InputStreamReader(System.in));
        tokenizer = null;
        writer = new PrintWriter(System.out);
        solve();
        reader.close();
        writer.close();
```

```
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
}

int nextInt() throws IOException {
    return Integer.parseInt(nextToken());
}

long nextLong() throws IOException {
    return Long.parseLong(nextToken());
}

double nextDouble() throws IOException {
    return Double.parseDouble(nextToken());
}

String nextToken() throws IOException {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
        tokenizer = new StringTokenizer(reader.readLine());
    }
```

```
        return tokenizer.nextToken();

    }

}
```

MEDIUM LEVEL 2

1

Splitty Palindromes

1. **Problem statement:**

Not all words are palindromes. But some of them are Splitty Palindromes, if you split them into parts.

For example, "TEAMMATE" is not a palindrome, but if you split it into blocks: "TE AM MA TE", it will be a Splitty Palindrome made of 4 blocks.

If there is no way to split a word into blocks, then it is a Splitty Palindromes made of one block.

The word "TEAMMATE" can be split into even more blocks: "TE A M M A TE".

The real palindromes are Splitty Palindromes made of one block for every letter, like: "R A D A R".

Find the Splitty Palindrome in a given word with the maximum possible number of blocks.

2. Input Format:

The first line contains a word

3. Output Format:

A Splitty Palindrome

4. Constraints:

-

5. Samples Input/Output:

	Input data	Output data
1	teammate	te a m m a te
2	radar	r a d a r

6. Test Cases:

	Input data	Output data
1	swift	swift
2	aaaaav	aaaaav
3	aavvaaa	a a vva a a
4	aaaaaaaa	a a a a a a a a
5	aaaaaaa	a a a a a a a
6	a	a
7	swiftswift	swift swift

7. Sample solution (Java):

```
import java.util.*;

public class Solution {
    public static void main(String [] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println(splittyPalindrome(scan.next()));
    }
```

```
    public static String splittyPalindrome(String word) {
        int length = word.length();
        if (length < 2) return word;
        for (int i = 0; i <= length/2 - 1; i++) {
            String block = word.substring(0,i+1);
            if (word.endsWith(block)) {
                int from = i+1; int to = length-1-i;
                String innerPolindrome = from<= to ?
splittyPalindrome(word.substring(from, to)) : "";
                String innerPolindromeF = innerPolindrome.length() > 0 ? " " +
innerPolindrome + " " : " ";
                return block + innerPolindromeF + block;
            }
        }
        return word;
    }
}
```

2

Buttons

1. **Problem statement:**

Peter is trying to pick a very interesting lock. The lock has n buttons that he need to push in a certain sequence to make it open. When a button is pushed, it either remains pressed, which means that it really is the next one in the sequence, or resets to its initial state along with all previously pressed buttons. When all buttons are pressed, the lock opens.

Let’s take a look at an example with three buttons. For instance, the correct sequence of buttons to open the lock, is: {2, 3, 1}. If you first push button 1 or 3, they will be reset immediately. If you press button 2, it will remain pressed. If after pushing button 2 you push button 1, all the buttons will be reset. If after pushing button 2 you push button 3, it will remain pressed along with with button 2. With two buttons pressed, all that remains is pushing button 1 for the lock to open.

Peter does not know the right sequence to open the lock. But he is very smart and will act optimally. Calculate the number of times he will need to push the buttons in order to open the lock in the worst case scenario.

2. Input Format:

The only line contains an integer n - the number of buttons the lock has.

3. Output Format:

On a single line print out the number of times he will need to push the buttons in order to open the lock in the worst case scenario.

4. Constraints:

$1 \leq n \leq 2000$

5. Samples Input/Output:

	Input data	Output data
1	2	3
2	3	7

Let's take a look at the first test case. Peter can have bad luck with his first try and push the wrong button. In this case, at his second try, he can guess the first button right. And at his third try he can get the second button right. Thus, in the worst case, he needs to push the buttons only 3 times.

6. Test Cases:

	Input data	Output data
1	4	14
2	1	1
3	10	175
4	1999	1331335999
5	92	129858
6	1241	318541121
7	1500	562501250

7. Sample solution (Javascript/Java/Python):

```
import java.util.*;

public class p268b {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int N = in.nextInt();


int res = 0;

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

res += (N-i) + (N-i-1)*i;


System.out.println(res);

}

}
```

3

Master of Eight

1. **Problem statement:**

You are given a non-negative number n, which is a record of no more than 100 digits and does not contain any leading nonsignificant zeros.

Your task is to determine whether it's possible to delete some (possibly a zero) number of digits in such a way, that the resulting number of digits left after the deletion would: contain at least one digit, be non-negative, have no leading nonsignificant zeros and be divisible by 8. Swapping digits after the deletion is not allowed.

2. Input Format:

The only input line contains a non-negative number n.

3. Output Format:

Output "NO" (without quotes) if a proper method to delete some of the digits in number n does not exist.

Otherwise, print "YES" on the first line and the numbers that result in deleting certain digits of n, on the second line. The displayed number must be divisible by 8.

If there are several answers, you can to display any of them.

4. Constraints:

The record of numbers n in Input data does not contain any leading nonsignificant zeros and its length does not exceed 100 digits.

5. Samples Input/Output:

	Input data	Output data
1	3454	YES 344
2	10	YES 0

6. Test Cases:

	Input data	Output data
1	111111	NO
2	312	YES

		32
3	7674	YES 64
4	8996988892	YES 8
5	5555555555	NO
6	81475227769199162773068613469229242 21557534659480258977017038624458370 45929984759093775762579123918 8	YES 0
7	126	YES 16

7. Sample solution (Java):

```
import java.util.*;

public class c {

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

char[] s = input.next().toCharArray();

int n = s.length;

boolean good = false;

for(int i = 0; i<n; i++) if(s[i] == '8' || s[i] == '0')

{
```

```
System.out.println("YES\n"+s[i]);
return;
}
for(int i = 0; i<n; i++)
for(int j = i+1; j<n; j++)
{
int cur = 10* (s[i] - '0') + (s[j] - '0');
if(cur%8 == 0)
{
System.out.println("YES\n"+cur);
return;
}
}
for(int i = 0; i<n; i++)
for(int j = i+1; j<n; j++)
{
for(int k = j+1; k<n; k++)
{
int cur = 10*(10* (s[i] - '0') + (s[j] - '0'))+(s[k] - '0');
if(cur%8 == 0)
{
System.out.println("YES\n"+cur);
return;
}
}
}
System.out.println("NO");
}
```

}

4

Calendar

1. **Problem statement:**

The Gregorian calendar is used universally nowadays, it has actually become the international standard and is used for civilian purposes in almost every corner of the Earth. The Gregorian reform corrected the Julian system of leap years in the following manner:

Every year that is divisible by four without a remainder, is a leap year, except for years that are divisible by 100 without a remainder. If a "centurial" year can be divided by 400 with no remainder left, it is still a leap year. For example, 1900 is not a leap year, but 2000 is.

In this task, you are given two dates and required to count the number of days between them. Remember that in February of every leap year there's an unusual number of days.

See the test cases in order to understand which boundaries are included in the response, and which are not.

2. Input Format:

The first two lines contain two dates, each date is written in the format yyyy: mm: dd .

3. Output Format:

Print out a single integer - the answer to the task.

4. Constraints:

`1900 ≤ 2038 and ≤ yyyy yyyy: mm: dd - the correct date`

5. Samples Input/Output:

	Input data	Output data
1	1900:01:01 2038:12:31	50768

2	1996:03:09 1991:11:12	1579

6. Test Cases:

	Input data	Output data
1	1999:12:31 2000:02:29	60
2	1913:11:14 1901:05:11	4570
3	1902:09:06 1951:03:31	17738
4	1996:03:09 1996:03:09	0
5	2015:10:17 1966:07:12	17994
6	2025:08:10 2018:09:03	2533
7	2027:07:31 1945:04:06	30066

7. Sample solution (Java):

```
import java.math.*;
```

```
import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.*;


public class Solution {

public static void main(String[] args) throws ParseException

{

Scanner in = new Scanner(System.in);

SimpleDateFormat sdf = new SimpleDateFormat("yyyy:MM:dd");


sdf.setTimeZone(TimeZone.getDefault());


Date from = sdf.parse(in.next());

Date to = sdf.parse(in.next());


System.out.print(Math.round(Math.abs(to.getTime() - from.getTime()) / 86400000.));

}

}
```

5

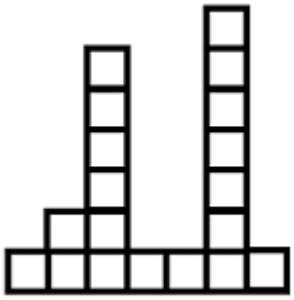
Skyscraper

1. **Problem statement:**

Let's look at the city's skyscrapers. There are N skyscrapers of the same width that go one after another from left to right. The height of each building is i = h_{i}, which stands for how many stories the building has..

But the government thinks that these skyscrapers are too high. They want to remove K stories from the skyscrapers, so that the sum of their heights becomes the minimum possible.

You need to find the indexes of k consecutive stories with the minimum total height.



2. Input Format:

The first line contains N and K.

The second line contains a sequence of integers *h_{1}*, *h_{2}*, ..., *h_{n},* where *h_{i}* is the height of the *i*-th building (the number of stories).

3. Output Format:

Print out an integer *j* that is the sum of heights of the buildings *j*, *j* + 1, ..., and *j* + *k* - 1 is the minimum possible height. If there are multiple such *j*'s, print out any of them.

4. Constraints:

N and K must be positive integers.

5. Samples Input/Output:

	Input data	Output data
1	7 3 1 2 6 1 1 7 1	3
2	2 1 10 20	1

6. Test Cases:

	Input data	Output data

1	10 5 1 2 3 1 2 2 3 1 4 5	1
2	10 2 3 1 4 1 4 6 2 1 4 6	7
3	2 1 20 1	2
4	3 1 1 2 3	1
5	3 1 3 2 1	3
6	4 1 1 5 2 2	1
7	5 2 100 100 100 1	4

7. Source (Java)

```
int[] h = new int[n]; int[] sum = new int[n];
```

```
int min = Integer.MAX_VALUE;
        int minIndex = 0;

        sum[0] = 0;

        for(int i = 0; i < n; i++) {
            h[i] = scanner.nextInt();

            if(i > 0)
                sum[i] = sum[i - 1] + h[i];
            else
                sum[i] = h[i];
        }

        for(int i = 0; i < n - (k - 1); i++) {
            int s;

            if(i > 0)
                s = sum[i + (k - 1)] - sum[i - 1];
```

```
        else

            s = sum[i + (k - 1)];



        if(s < min) {

            min = s;

            minIndex = i + 1;

        }

    }



    System.out.println(minIndex);
```

6

Do You Even Swap

1. **Problem statement:**

You are given a positive integer *a* that has no leading zeroes. You can swap two adjacent decimal digits of this number.

What is the maximum number you can get in *t* swaps?

2. Input Format:

A single line thay contains two integers *a* and *t* .

3. Output Format:

Print out the maximum number that you can get out of a if you make no more than *t* swaps.

4. Constraints:

$1 \le a \le 10^{18}; \ 0 \le t \le 100$

5. Samples Input/Output:

	Input	Output
1	1234 6	4321
2	9022 2	9220

6. Test Cases:

	Input	Output
1	39940894417248510 10	99984304417248510
2	5314 4	5431
3	1026 9	6210
4	219810011901120912 100	999822211111110000
5	10120921290110921 20	99221010120110921
6	337775999910796051 37	999997733751076051
7	39940894417248510 10	99984304417248510
8	521325125110071928 4	552132125110071928

7. Solution (Java)

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

```
public class Main {
public static void main(String[] args) {
Scanner in = new Scanner(System.in);
StringBuilder a = new StringBuilder( in .next());
int k = in .nextInt();

for (int i = 0; i < a.length(); i++) {
int j = -1;
char c = 0;
for (int _j = i;
(_j <= i + k) && (_j < a.length()); _j++) {
if (a.charAt(_j) > c) {
c = a.charAt(_j);
j = _j;
}
}
a.deleteCharAt(j);
a.insert(i, c);
k -= (j - i);
}

System.out.println(a);
}
}
```

7

Dirty Stairs

1. **Problem statement:**

There's a man standing on the first step of stairs consisting of n steps. He knows the number of dirty steps of these stairs. Help him find out whether he can jump over the entire staircase and get to the step number n without ever touching a dirty step. The man can jump to the next step, skip one or two steps.

It should be noted that the man always starts at the first step and ends at the last one, so if one of them is dirty, the man can't make his way visiting clean steps only.

2. Input Format:

The first line contains two integers n and m – the number of steps in the staircase and the number of dirty steps, respectively. The second line contains m distinct integers d1, d2, ..., dm – the numbers of dirty steps (in any order).

3. Output Format:

Output «YES» if the man can get to the step number n by only visiting the steps that are clean. Otherwise, output «NO».

4. Constraints:

$1 \le n \le 10^9, 0 \le m \le 3000$

$1 \le d_i \le n$

5. Samples Input/Output:

	Input	Output
1	10 5 2 4 8 3 6	NO
2	123 13 36 73 111 2 92 5 47 55 48 113 7 78 37	YES

6. Test Cases:

	Input	Output
1	10 10 7 6 4 2 5 10 8 3 9 1	NO
2	10 5 2 4 5 7 9	YES
3	10 9 2 3 4 5 6 7 8 9 10	NO
4	5 2 4 5	NO
5	123 13 36 73 111 2 92 5 47 55 48 113 7 78 37	YES
6	12312 0	YES
7	9817239 1 6323187	YES

7. Solution (Java)

```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 long n = sc.nextInt();
```

```
 int m = sc.nextInt();
 int [] str = new int[m];
 for(int i = 0; i<m; i++)
str[i] = sc.nextInt();
 Arrays.sort(str);
 if(m>0 && (str[0] == 1 || str[str.length-1] == n)) {
  System.out.println("NO");
  return;
 }
 for (int i = 0; i<m-2; i++) {
 if(str[i] +1 == str[i+1] && str[i+1]+1 == str[i+2]) {
 System.out.println("NO");
 return;
 }
 }
 System.out.println("YES");
}
```

8

Small Reform

1. **Problem statement:**

There's a small money reform going in the state of Nippon. It was decided that the most expensive coin should have the value of *n* Nippon dollars. The value of each coin should also be a divisor of any greater coin's value.

Find a way to maximize the number of coins introduced with the new system. Print the coins' values in descending order.

2. Input Format:

The first and only line contains an integer *n* equal to the value of the most expensive coin.

3. Output Format:

Print out the coins' values in descending order. Maximize the number of coins (with the given value *n* of the most expensive coin). The value of each coin should be a divisor of any greater coin's value. Different coins can not have the same value. If there are several solutions to this problem, print out any of them.

4. Constraints:

$1 \leq n \leq 10^6$

5. Samples Input/Output:

	Input	Output
1	10	10 5 1
2	4	4 2 1

6. Test Cases:

	Input	Output
1	3	3 1
2	1000	1000 500 250 125 25 5 1
3	572877	572877 190959 63653 1201 1
4	100000	100000 50000 25000 12500 6250 3125 625 125 25 5 1
5	509149	509149 1
6	572877	572877 190959 63653 1201 1
7	537000	537000 268500 134250 67125 22375 4475 895 179 1
8	999999	999999 333333 111111 37037 5291 481 37 1

7. Solution (Java)

```
import java.io.*;

import java.util.*;

public class j {

public static void main(String a[]) throws IOException {

BufferedReader b = new BufferedReader(new InputStreamReader(System.in));

int m = 0, j = 0;

m = Integer.parseInt(b.readLine());

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

while (m != 1) {

for (j = m - 1; j > 0; j--) {

if (m % j == 0) {

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

break;

}

}

m = j;

}

}

}
```

9

The toy shop

1. **Problem statement:**

James has a large number of similar toys in his shop. He decided to paint all of them in different colors. He put n similarl toys in a circle.

Each toy must be painted in one of the seven colors: red, orange, yellow, green, blue, indigo or violet. The following conditions must be met:

- At least one toy must be colored in each of the seven colors.
- Any four toys in a row must be painted in different colors.

Help James paint the toys the way he planned to. We know that this is always possible.

2. Input Format:

The only line contains an integer n.

3. Output Format:

Print out a string with the length of n, where the n-th character of the string should stand for the color of the i-th toys following the order of the circle. All colors are indicated by the following symbols: «R» - red, «O» - orange, «Y» - yellow, «G» - green, «B» - blue, «I» - dark blue, «V» - violet.

If there are several answers, output any of them.

4. Constraints:

$7 \leq n \leq 100$

5. Samples Input/Output:

	Input data	Output data
1	8	ROYGBIVG
2	13	ROYGBIVGBIVGB

6. Test Cases:

	Input data	Output data
1	7	ROYGBIV
2	10	ROYGBIVGBI
3	14	ROYGBIVGBIVGBI
4	9	ROYGBIVGB
5	11	ROYGBIVGBIV
6	12	ROYGBIVGBIVG

7	15	ROYGBIVGBIVGBIV

7. Sample solution (Java):

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

public class ToyShop {
    public static void main(String[] args) throws IOException {
        BufferedReader f = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.parseInt(f.readLine());
        char[] a = {'R','O','Y','G','B','I','V'};
        for (int i = 0; i < 7; i++)
            System.out.print(a[i]);
        for (int i = 7; i < n; i++)
            System.out.print(a[(i-7)%4+3]);
    }
}
```

10

Magic Square

1. **Problem statement:**

4	9	2
3	5	7
8	1	6

A Magic Square is a square matrix, in which the sums of numbers in each column, row and diagonal are equal.

Find the numbers from the main diagonal of a 3x3 Magic Square, depending on the other numbers.

2. Input Format:

Three strings, each with three space-separated numbers – the values of cells in the matrix.

3. Output Format:

The repaired Magic Square

4. Constraints:

The values of numbers in the main diagonal are replaced with zeros.

5. Samples Input/Output:

	Input	Output
1	0 4 4 4 0 4 4 4 0	4 4 4 4 4 4 4 4 4
2	0 54 48 36 0 78 66 60 0	69 54 48 36 57 78 66 60 45

6. Test Cases:

	Input	Output
1	0 97 56 69 0 71 84 43 0	57 97 56 69 70 71 84 43 83
2	0 1 1 1 0 1 1 1 0	1 1 1 1 1 1 1 1 1

3	0 3 6 5 0 5 4 7 0	6 3 6 5 5 5 4 7 4
4	0 17 14 15 0 15 16 13 0	14 17 14 15 15 15 16 13 16
5	0 1099 1002 1027 0 1049 1074 977 0	1013 1099 1002 1027 1038 1049 1074 977 1063
6	0 98721 99776 99575 0 99123 98922 99977 0	99550 98721 99776 99575 99349 99123 98922 99977 99148
7	0 6361 2304 1433 0 8103 7232 3175 0	5639 6361 2304 1433 4768 8103 7232 3175 3897

7. Solution (Java)

```
public static void main(String[] args) {
 Scanner in = new Scanner (System.in);
 int [] z = new int [9];
 int m = 0;
 for(int i = 0; i < z.length; i++) {
z[i]=in.nextInt();
```

```
m+=z[i];
 }
m/=2;
z[0]=m-(z[1]+z[2]);
z[4]=m-(z[3]+z[5]);
z[8]=m-(z[7]+z[6]);
 for(int i = 0; i < z.length; i+=3)
System.out.println(z[i]+" "+z[i+1]+" "+z[i+2]);
}
```

11

Magic Pool

1. **Problem statement:**

There is a magic pool in Nippon that can change its size. It has the length of *a* and the width f *b.* When tourists visit the pool it is resized so each tourist has at least 6 square meters of personal space.

Each side must have a length value in integer meters, and the pool must have the smallest area possible. Find out how the pool's sides need to change to fit all tourists and keep the total area of the pool to its possible minimum.

2. Input Format:

The first line contains three space-delimited integers *n*, *a* and *b* — the number of tourists and the sizes of the pool.

3. Output Format:

Print out three integers *s*, *a*1 and *b*1 ($a \le a1$; $b \le b1$) — the final area of the pool and its sizes. If there are multiple optimal solutions, print any of them.

4. Constraints:

$1 \le n, a, b \le 10^9$

5. Samples Input/Output:

	Input data	Output data
1	3 3 5	18 3 6
2	2 4 4	16 4 4

6. Test Cases:

	Input data	Output data
1	1 1 1	6 1 6
2	8 7 5	48 8 6
3	11 7 7	70 7 10
4	800000003 7 7	4800000018 11 436363638
5	1000000000 1 1	6000000000 1 6000000000
6	100000 100 1000	600000 100 6000
7	16 19 5	100 20 5
8	258180623 16000 16000	1549083738

		16067 96414

7. Sample solution (Java):

```
import java.util.*;

public class SampleClass {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int n = input.nextInt(), a = input.nextInt(), b = input.nextInt();

long besta = (int) 1e9, bestb = (int) 1e9;

for (int i = 0; i < 1000000; i++) {

long na = a + i;

long needb = ((long) 6 * n + na - 1) / na;

long nb = b + Math.max(0, needb - b);

if ((long) na * nb < besta * bestb) {

besta = na;

bestb = nb;

}

nb = b + i;

long needa = ((long) 6 * n + nb - 1) / nb;

na = a + Math.max(0, needa - a);

if ((long) na * nb < besta * bestb) {

besta = na;

bestb = nb;

}

}

System.out.println(besta * bestb + "\n" + besta + " " + bestb);
```

}

}

12

Changing Username

1. **Problem statement:**

There are some users with certain usernames in the database. Some of them want to change their usernames.

A new username must be unique compared to all of the actual and old usernames.

You have a list of username change requests, but after executing it, you want to know the correspondence between the old usernames and the new ones.

2. Input Format:

The first line contains the integer number *q* of username change requests.

The next lines contain space-delimited pairs of old and new usernames.

3. Output Format:

The first line – the number of people that change their usernames.

The next lines – space-delimited pairs of the first and current usernames.

Each user changing their username must be in this list only once.

4. Constraints:

$1 \le q \le 1000$

5. Samples Input/Output:

	Input	Output
1	3 Miha_x64 Miha_x32 Dog Wolf	2 Dog Wolf Miha_x64

	Miha_x32 Miha_x86-64	Miha_x86-64
2	10 a b b c c d d e e f f g g h h i i j j k	1 a k

6. Test cases

	Input	Output
1	8 M F S D 1 2 F G 2 R D Q Q W W e	3 S e M G 1 R
2	5 a b b c c d e F	2 a d e g

	F g	
3	3 qw er as fg fg jk	2 qw er as jk
4	6 z x c v b n x s v d n m	3 z s c d b m
5	0	0
6	1 name0 name1	1 name0 name1
7	3 one two three four two eight	2 one eight three four

7. Solution (Java)

```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 int n = sc.nextInt();
 HashMap<String, String> m = new HashMap<String, String>();
```

```
 while (n-- != 0) {
 String a, b; a = sc.next();
b = sc.next();
 if(m.containsKey(a)) {
m.put(b, m.get(a));
m.remove(a);
 } else {
m.put(b, a);
 }
 }
 System.out.println(m.size());
 for(String a : m.keySet())
 System.out.println(m.get(a) + " " + a);
}
```

13

Containing Number

1. **Problem statement:**

We call a number ‘k-containing’ if it contains all digits from 0 to k. Count how many k-containing numbers are entered.

2. Input Format:

The first line contains the count of the following numbers to check and the value of k.

The next line contain the numbers to check.

3. Output Format:

The amount of k-containing numbers.

4. Constraints:

For first line: the count of the following numbers to check must be in range from 1 to 100. $0 \leq k \leq 9$

For next lines: each number to check must be in range from 1 to 10^9

5. Samples Input/Output:

	Input	Output
1	5 2 01234 12345 12340 23456 34567	2
2	2 1 1 10	1

6. Test cases

	Input	Output
1	6 0 10 102 120 1032 1212103 1999999	5
2	1 0 1000000000	1

3	6 0 10 102 120 1032 1212103 1999999	5
4	1 3 1000000000	0
5	2 8 123456780 123	1
6	6 1 10 102 120 1032 1212103 1999999	5
7	6 2 10 102 120 1032 1212103 1999999	4

7. Solution (Java)

```
public static void main(String ar[]) {
 Scanner obj = new Scanner(System.in);
 int n = obj.nextInt();
 int m = obj.nextInt();
 int c,s = 0;
 for(int i=0;i<=n;i++) {
 String str = obj.nextLine();
c = 0;
 for (int j=0; j <= m; j++) {
 if (str.contains(j+""))
 ++c;
 } if(c==m+1)
 ++s;
 }
 System.out.print(s);
}
```

14

Squirrel and Nuts

1. **Problem statement:**

A squirrel has n nuts. It puts them in *a* similar rows of *b* pieces each (a> 1).

After placing the nuts, the squirrel takes back any of the resulting rows (i.e. *b* nuts) and throws all the other nuts away. Then it puts the nuts in rows again, takes one row and so on. It continues as until the squirrel has only one nut remaining.

The process can be represented as a finite sequence of integers c_1, ..., c_k, where:

- $c_1 = n$
- c_{i+1} is the number of nuts that will remain in the squirrel's posession after the i-th turn, that is, the number of nuts in a row of some c_i nuts $(1 \leq i < k)$. Note that $c_i > c_{i+1}$.
- $c_k = 1$

The result of the game is the sum of numbers c_i. You are given a number n. Find the maximum possible outcome of the game.

2. Input Format:

The only line of input contains a single integer n - the initial amount of nuts

3. Output Format:

Print out a single integer - the maximum possible outcome of the game.

4. Constraints:

$2 \leq n \leq 10^9$

5. Samples Input/Output:

	Input data	Output data
1	10	16
2	8	15

6. Test Cases:

	Input data	Output data
1	4	7
2	36	67
3	32	63
4	46	70
5	6	10
6	13	14

7	9	13

7. Sample solution (Java):

```
  import java.util.Scanner;


public class B {
    public static void main(String[] args) {
        int n = new Scanner(System.in).nextInt();
        long ans = n;
        while (true){
            int t = n;
            for (int i = 2; i*i <=n; i++) {
                if (n%i == 0){
                    ans += n/i;
                    n /= i;
                    break;
                }
            }
            if (n == t) break;
        }
        ans++;
        System.out.println(ans);
    }
}
```

15

Collection Game Cards

1. **Problem statement:**

Billy works at a factory that produces collection game cards. He's assembling mini-boosters (collections of cards) containing a few cards (with one card at least). Every booster must contain only one ultra-rare card.

Billy gets the cards from the assembly line one-by-one, and a booster should contain only the cards that are near to each other on the line.

Count the number of options that Billy has to assemble different boosters from the given set of cards.

2. Input Format:

The first line contains a single positive number N - the number of cards on the assembly line.

The second line contains N integer numbers ai , where 0 is an ordinary card and 1 is an ultra-rare card.

3. Output Format:

A single positive number of different boosters with only one ultra-rare card in each.

4. Constraints:

1<=N<=100

0<=ai<=1

5. Samples Input/Output:

	Input data	Output data
1	3 0 1 0	1
2	5 1 0 1 0 1	4

6. Test Cases:

	Input data	Output data

1	10 0 0 1 0 0 0 1 1 0 1	8
2	20 0 0 0 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 1 0	24
3	50 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0 1 0 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0	11520
4	1 0	0
5	41 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0
6	1 1	1
7	18 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1

7. Sample solution (Java):

```
import java.util.Scanner;



public class Chocolate {

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

```
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        long lastNut = -1;
        long splits = 0;
        for (long i = 0; i < n; i++){
            if (input.nextInt() == 1){
                if (lastNut != -1){
                    splits*=(i-lastNut);
                } else {
                    splits = 1;
                }
                lastNut = i;
            }
        }
        input.close();
        System.out.println(splits);
    }
}
```

16

Going to a Bar

1. **Problem statement:**

Varzlav and Urgen are going to a bar in Berlin. Varzlav has just lost his job the other day, so he wants to go to the cheapest bar they can find. Urgen is rich, so he wants to go to the most expensive bar out there. He understands the problem of his friend, but doesn't want to drink the cheapest alcohol.

Berlin has N streets and M avenues. There is a bar at every intersection of a street and an avenue. The average cost of spending a night in each bar is A_{nm}

Varzlav and Urgen decided to choose the bar in a simple way: first, Urgen chooses a street and then Varzlac chooses an avenue, and they go to the bar situated at the corresponding intersection. Urgen remembers that Varzlav wants to go to the cheapest bar, but insists on his own choice.

What will be the final cost of their night in a bar?

2. Input Format:

The first line contains two single positive numbers N,M - the number of streets and avenues in Berlin.

Every next N line contains M integer numbers A_{nm} - the cost of spending a night in a bar at the intersection of street N and avenue M.

3. Output Format:

A single positive number A - the cost of Varzlav and Urgen's night in a bar.

4. Constraints:

1<=N,M<=100

$1<=A_{nm}<10^9$

5. Samples Input/Output:

	Input data	Output data

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

5-10 test cases to validate:

	Input data	Output data
1	1 1 1	1
2	1 10 74 35 82 39 1 84 29 41 70 12	1
3	10 1 44 23 65 17 48 29 49 88 91 85	91
4	10 10 256 72 455 45 912 506 235 68 951 92	184

	246 305 45 212 788 621 449 876 459 899 732 107 230 357 370 610 997 669 61 192 131 93 481 527 983 920 825 540 435 54 777 682 984 20 337 480 264 137 249 502 51 467 479 228 923 752 714 436 199 973 3 91 612 571 631 212 751 84 886 948 252 130 583 23 194 985 234 978 709 16 636 991 203 469 719 540 184 902 503 652 826 680 150 284 37 987 360 183 447 51	
5	2 1 999999999 1000000000	1000000000
6	1 1 1000000000	1000000000
7	2 2 1 1 1 1	1

7. Sample solution (Java):

```
import java.util.*;
public class Bar{
    public static void main(String[] args){
        Scanner in=new Scanner(System.in);
        int n=in.nextInt(),m=in.nextInt(),ans=0,min;
        for(int i=0; i<n; i++){
            min=1000000000;
            for (int j=0; j<m; j++){
                min=Math.min(in.nextInt(),min);
            }
            ans=Math.max(min,ans);
        }
        System.out.print(ans);
    }
```

```
}
```

17

Clockwork

1. **Problem statement:**

You are given a moment in time in the 24-hour format hh:mm. Find and print out the time after *n* minutes have passed.

2. Input Format:

The first line contains a moment in time in the format of hh:mm . If the hours or minutes are less than 10 they are given with leading zeros.

The second line contains one integer *n* — the number of minutes to add.

3. Output Format:

Output the moment in time in the format of hh:mm after adding *n* minutes.

4. Constraints:

$0 \le hh < 24$, $0 \le mm < 60$

$0 \le n \le 10^4$

5. Samples Input/Output:

	Input	Output
1	23:59 10	00:09
2	20:20 121	22:21

6. Test cases

	Input	Output
1	10:10 0	10:10
2	12:34 10000	11:14
3	00:00 1440	00:00
4	23:59 1	00:00
5	11:59 1	12:00
6	19:34 566	05:00
7	00:01 59	01:00
8	09:20 40	10:00

7. Solution (Java)

```
import java.util.Scanner;

public class SampleClass {
```

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
String str = sc.next();
String[] strings = str.split(":");
int hour = Integer.parseInt(strings[0]);
int minute = Integer.parseInt(strings[1]);
int time = (hour * 60 + minute + sc.nextInt()) % (24 * 60);
hour = time / 60;
minute = time % 60;
if (hour < 10) {
}
System.out.println(hour / 10 + "" + hour % 10 + ":" + minute / 10 + "" + minute % 10);
}
}
```

MEDIUM LEVEL

1

Junior Broker

1. **Problem statement:**

One day Finn acquired information on the Dollar to Euro exchange rate for the next N days.
The currency rates for one Dollar on day i are equal to Ai.
Finn has E Euro. Finn can only buy or sell currency once a day, and can only use integer values.

What is the maximum amount of Euro that Finn can have by the end of day n?

2. **Input Format:**

The first line contains two integers N and E — the number of days and the amount of Euro Finn has on the first day.

The next line contains N integers Ai — the currency rate of Dollar to Euro on day i.

3. **Output Format:**

Print a single integer number — what is the maximum amount of Euro that Finn can have by the end of day n.

4. Constraints:

$1 \le N, E \le 2000$

$1 \le A_i \le 2000$

5. Samples Input/Output:

	Input data	Output data

1	2 4 3 7	8
2	4 10 4 3 2 1	10

6. Test Cases:

	Input data	Output data
1	4 10 4 2 3 1	15
2	10 595 881 832 1159 171 230 750 361 1800 516 567	5482
3	1 1483 1126	1483
4	4 1663 1904 1049 1622 472	2236
5	5 1293 1183 142 1356 889 134	12219
6	2 755 51 160	2281

7	3 385 978 1604 1888	385

7. Sample solution (Java):

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

public static void main (String args[]){
Scanner sc = new Scanner (System.in);
int n = sc.nextInt();
int b = sc.nextInt();
ArrayList<Integer> price = new ArrayList<>();
for (int i = 0; i < n; i++){
price.add(sc.nextInt());
}
sc.close();
int max = b;
for (int i = 0; i < n; i++){
for (int j = i; j < n; j++){
int buy = b / price.get(i);
int rest = b % price.get(i);
int sell = buy * price.get(j) + rest;
max = Math.max(max, sell);
}
}

System.out.println(max);
}
}
```

2

Treasure Lock

1. **Problem statement:**

There is a lock on a treasure chest. It has n buttons on it. To open the lock, you can press all the buttons in a certain order. When you push a correct button, it stays pressed. And when you push a wrong one, all buttons are reset to their initial state.

2. Input Format:

The number of buttons - n.

3. Output Format:

The number of times you have to push the buttons in the worst case scenario.

4. Constraints:

$1 \leq n \leq 2000$

5. Samples Input/Output:

	Input	Output
1	1	1
2	2	3

6. Test cases

	Input	Output
1	3	7
2	4	14
3	10	175
4	2000	1333335000

5	1747	888644743
6	889	117099969
7	1999	1331335999

7. Solution (Java)

```
public static void main(String args[]) {
  Scanner in = new Scanner(System.in);
 int x = in.nextInt();
 int c = x;
 for (int i = 1; i < x; i++) {
c = c + i*(x-i);
 }
System.out.print(c);
}
```

3

A and B and Compilation Errors

1. **Problem statement:**

A and B are preparing for the Olympic programming championship.

B loves to debug his code. But before he runs a program and starts debugging it, he needs to be compile his code first.

Initially, the compiler gave n comipling errors, with each being represented by a positive integer. After some effort, B managed to first correct one error and then another one

However, despite the fact that B is certain that he has corrected two errors, he can not understand which of the compiling errors disappeared - the language compiler that B is using displays errors in a new order every time! B is sure that, unlike many other programming languages, compilation errors

in his programming language do not depend on each other, i.e. correcting one error does not affect any other errors. Help B - what are the two errors that he has corrected?

2. Input Format:

The first line of input contains an integer n - the initial number of compilation errors.

The second line contains n space-separated integers a1, a2, ..., an - the numbers that the compiler assigned to errors in B's code when he rat it for the first time

The third line contains n - 1 space-separated integers b1, b2, ..., bn - 1 - the error numbers displayed during the second compilation. It is guaranteed that the sequence in the third line contains all numbers from the second line, except for exactly one number.

The fourth line contains n - 2 space-separated integers c1, c2, ..., cn - 2 - the error numbers displayed during the third compilation. It is guaranteed that the sequence in the fourth line contains all numbers from the third line, except for exactly one number.

3. Output Format:

Print out two numbers on a single line: the numbers of compile errors that disappeared after B had introduced the first and second correction to his code, respectively.

4. Constraints:

$3 \leq n \leq 10^5$

$1 \leq a_i \leq 10^9$

5. Samples Input/Output:

	Input data	Output data
1	5 1 5 8 123 7 123 7 5 1 5 1 7	8 123
2	6 1 4 3 3 5 7 3 7 5 4 3 4 3 7 5	1 3

6. Test cases:

	Input data	Output data
1	3 1 2 3 3 2 2	1 3
2	10 460626451 802090732 277246428 661369649 388684428 784303821 376287098 656422756 9301599 25720377 277246428 388684428 661369649 460626451 656422756 802090732 9301599 784303821 376287098 376287098 802090732 388684428 9301599 656422756 784303821 460626451 277246428	25720377 661369649
3	3 374054998 726316780 902899520 902899520 726316780 726316780	374054998 902899520
4	3 168638990 939116221 323703261 168638990 323703261 168638990	939116221 323703261

5	3 77 77 77 77 77 77	77 77
6	3 84 30 9 9 84 9	30 84
7	6 5 4 3 3 5 5 3 5 5 4 3 3 5 4 3	5 5

7. Sample solution (Java):

```
import java.io.*;
import java.util.*;
public class Main
{
public static void main(String[] args)
{
Scanner scanner = new Scanner(System.in);
 int n=scanner.nextInt();
int m=scanner.nextInt();
int k = scanner.nextInt();
int[] number = new int[m];
int fed;
```

```
for(int i=0; i<m; i++)
{
number[i] = scanner.nextInt();
 }
fed = scanner.nextInt();
int result =0;
for(int i=0; i<m; i++)
{
if(Integer.bitCount(fed^number[i])<=k)
{
result++;
}
}
 System.out.println(result);
}
}
```

4

Bookworm

1. **Problem statement:**

John has an important task - he needs to number the books in the class bookshelf, putting a tag with a number on each book. Each of the n number of books should have a number from 1 to n, and, of course, different books should get different numbers.

John wants to know how many digits he has to write down while numbering all those tags.

2. Input Format:

The first line contains an integer n - the number of books in the class bookshelf.

3. Output Format:

Output the number of digits needed to number all the books.

4. Constraints:

$1 \leq n \leq 10^9$

5. Samples Input/Output:

	Input	Output
1	4	4
2	13	17

6. Test cases

	Input	Output
1	100	192
2	99	189
3	999	2889
4	10000	5282829
5	20	31

6	78	147
7	1337	4241

7. Solution (Java)

```
import java.util.Scanner;

public class SampleClass {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

sc.close();

long digits = 0;

for (int i = 1; i <= n; i *= 10) {

digits += n - i + 1;

}

System.out.print(digits);

}

}
```

5

Duff in Love

1. **Problem statement:**

Duff loves beautiful numbers! A positive integer x is considered to be a beautiful number if and only if there is no such positive integer as a>1 that makes x a divisor of a2.

Malek owns a store that sells numbers! His store only sells divisors of a positive integer n (and them only). Malek wants to make a birthday gift for his girlfriend - a beautifull number from his store. He also wants that number to be as great as possible.

Malek was never good at math, so he’s asking you to help him out.. Please find him greatest beautiful number in his store.

2. Input Format:

The first and only line of input contains one integer, *n*.

3. Output Format:

Print out the answer on a single line.

4. Constraints:

$1 \le n \le 10^{12}$

5. Samples Input/Output:

	Input data	Output data
1	10	10
2	12	6

6. Test cases:

	Input data	Output data
1	2	2
2	4	2
3	1000000000000	10
4	2457	273

5	3096	258
6	808453785117	808453785117
7	5000	10

7. Sample solution (Java):

```
import java.util.Scanner;
 public class MainB
 {
Scanner sc = new Scanner(System.in);
 void run()
{
long n = sc.nextLong();
long ans = 1;
 for (long i = 2; i * i <= n; i++)
{
 if (n % i == 0)
 {
ans *= i;
 while (n % i == 0) n /= i;
 }
}
System.out.println(ans * n);
}
public static voidmain(String[] args)
{
```

```
new MainB().run();
}
}
```

6

Elite Architect

1. **Problem statement:**

In the capital of Nippon there are N apartment buildings. The architect who built the capital was very creative, so all the buildings are built in a row.

All the buildings are numbered from left to right, starting from number one. A building is considered to be elite if the number of stories it has is strictly greater than in all the buildings with a greater number. In other words, a building is elite if its number of stories is greater than in all the buildings located to its right. In this task, it is assumed that the story height in all buildings is the same.

The new architect is interested in n questions, the i-th of them is: "How many stories should be added to the i-th building to make it elite?" (where the variable i can take any value from 1 to n). You need to help him with this task.

It should be noted that all questions are independent from each other - the answer to the question for the house i does not affect the other answers (ie, stories will actually not be added to the buildings).

2. Input Format:

The first line of input contains a single integer n - the number of Buildings in the capital of Nippon.

The second line contains n positive integers hi , where hi is equal to the number of stories in the i-th building.

3. Output Format:

Output n integers a1, a2, ..., an, where ai is a number equal to the number of stories that need to be added to building number i to make it elite. If the building is elite already, and hence does not need any additional stories, then ai must equal zero.

All the buildings are numbered from left to right, starting from number 1.

4. Constraints:

$1 \leq n \leq 10^5$

$1 \le h_i \le 10^9$

5. Samples Input/Output:

	Input	Output
1	5 1 2 3 1 2	3 2 0 2 0
2	4 3 2 1 4	2 3 4 0

6. Test cases

	Input	Output
1	1 2	0
2	2 5 4	0 0
3	5 10 18 36 33 20	27 19 0 0 0
4	3 1 2 3	3 2 0

5	4 45 32 99 6	55 68 0 0
6	5 1 4 2 5 3	5 2 4 0 0
7	4 0 0 0 0	1 1 1 1

7. Solution (Java)

```
import java.io.*;

import java.util.*;

public class b1 {

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

Scanner in = new Scanner(System.in);

PrintWriter out = new PrintWriter(System.out);

int n = in .nextInt();

int h[] = new int[n];

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

h[i] = in .nextInt();

}

int ans[] = new int[n];

int max = 0;

for (int i = n - 1; i > -1; i--) {

ans[i] = Math.max(max - h[i] + 1, 0);
```

```
max = Math.max(max, h[i]);
}
for (int i = 0; i < n; i++) {
out.print(ans[i] + " ");
}
out.close();
}
}
```

7

Supergame

1. **Problem statement:**

One day n boys decided to play the ‘Supergame’. In each round of the "Supergame" a game master is chosen from one of the boys, the other n - 1 boys participate in the game as players. For every person participating, you know how many rounds each of them wants to play as a player, not as a game master: i-th person wants to play ai rounds. What is the minimum number of the ‘Supergame’ rounds do the boys need to play so that everyone has played at least as many rounds as they wanted?

2. Input Format:

The first line contains an integer n . The second line contains n integers a1, a2, ..., an - the i-th number in the list indicates the number of rounds that the i-th person wants to play.

3. Output Format:

On a single line, output an integer - the minimum number of game rounds the boys need to play, so that the i-th person has played at least ai rounds.

4. Constraints:

$3 \le n \le 10^5$

$1 \le a_i \le 10^9$

5. Samples Input/Output:

	Input data	Output data
1	3 3 2 2	4
2	4 2 2 2 2	3

6. Test cases:

	Input data	Output data
1	7 9 7 7 8 8 7 8	9
2	10 13 12 10 13 13 14 10 10 12 12	14
3	10 94 96 91 95 99 94 96 92 95 99	106

4	3 2 1 1	2
5	4 1 2 3 4	4
6	3 1000000000 1000000000 10000000	1005000000
7	3 677876423 834056477 553175531	1032554216

7. Sample solution (Java):

```
import java.util.Scanner;
public class SampleClass {
public static void main(String[] args) {
Scanner sc=new Scanner (System.in);
int m=-1;
int n=sc.nextInt(); long s=0;for (int i=0;i<n;i++){
int t=sc.nextInt();
if (t>m)m=t;
s+=t;
}
long r=s%(n-1)==0?s/(n-1):s/(n-1)+1;
if (r>=m)
System.out.println(r);
```

```
else System.out.println(m);

}}
```

8

Encoding

1. **Problem statement:**

Let phi(W) be the result of the following encoding algorithm:

1. If the length of W is 1, then phi(W) is W;

2. Let the coded word be $W = w_1w_2...w_N$ and K = N / 2 (rounded down);

3. $phi(W) = phi(w_Nw_{N-1}...w_{K+1}) + phi(w_Kw_{K-1}...w_1)$.

For example, phi('Ok') = 'kO', phi('abcd') = 'cdab'.

Your task is to find the position of letter w_q in the encoded word phi(W).

2. Input Format:

You are given integers N, q, where N is the length of word W.

3. Output Format:

Print out a number corresponding to the position of letter w_q in encoded word phi(W).

4. Constraints:

1 <= N <= 10^9; 1<= q <= N

5. Samples Input/Output:

	Input data	Output data
1	9 4	8
2	3 18	17

6. Test Cases:

	Input data	Output data
1	3235 35	2126
2	43 6	25
3	22 1	16
4	23423432 333	15615488
5	1 1	1
6	2 0	3
7	100000000 10000000	70000003

7. Sample solution (Java):

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

public class Main
{
public static void main(String arg[])
{
new Main();
}
public Main()
{
```

```
new TaskI();
}
}

class TaskI
{
int get(int N,int q)
{
if (N==1) return q;
int K=N/2;
if (q<=K) return N-K+get(K,K-q+1);
else return get(N-K,N-q+1);
}
public TaskI()
{
Scanner cin=new Scanner(new BufferedInputStream(System.in));
int N=cin.nextInt(),q=cin.nextInt();
System.out.println(get(N,q));
cin.close();
}
}
```

9

E-shopping

1. **Problem statement:**

Alex is looking for a new phone in an online shop. He’s browsed through some models, and some of them appear to be brand new ones.

In one click Alex can:

- open a phone model's description (so, if this description was previoisly unread, it becomes read)
- go back to the list of all models
- go from a model's page to the next one (if that model isn't the last one)
- go from a model's page to the previous one (if that model isn't the first one)

Alex wants to read the description for all models in the quickest way possible. What's the minimum number of clicks that he needs to make in order to do so?

2. Input Format:

The first line contains the number of phone models' descriptions in the shop (n).

The second line contains a sequence of n space-delimited numbers: 0 stands for an unread description, 1 – for a read description.

3. Output Format:

A single number – the minimum number of clicks required to read all the provided descriptions.

4. Constraints:

$1 \le n \le 1000$

5. Samples Input/Output:

	Input	Output
1	9 1 0 1 0 1 0 1 0 1	9
2	14 0 0 1 1 1 0 1 1 1 0 1 1 1 0	11

6. Test cases

	Input	Output
1	27 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	25
2	5 0 1 0 1 0	3
3	5 1 1 0 0 1	4
4	2 0 0	0
5	5 1 1 1 1 1	5
6	14 0 0 1 1 1 0 1 1 1 0 1 1 1 0	11
7	23 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1	23

7. Solution (Java)

```
public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 int a = sc.nextInt();
```

```
 int e[] = new int[a];
 for (int i = 0; i < a; i++) {
e[i] = sc.nextInt();
}
 int c=1;
   for(int i = 0; i < a-1; i++) {
  if (e[i]==1) {
 if (e[i+1] == 1)
c++;
 else
c+=2;
}
 }
 if (e[a-1] == 0)
c-=2;
 System.out.println(c<0 ? "0" : c);
}
```

10

Fabulous Sum

1. **Problem statement:**

In this task, you need to find the sum of all integers between 1 and n, but if a number has a power of twoб it must be added to the the sum with a minus sign.

This kind of sum is called a fabulous sum.

For example, when n = 4, the sum is equal to: - 1 - 3 + 2 - 4 = - 4 as 1, 2 and 4 are 2^0, 2^1 and 2^2, respectively.

Calculate the answer for t values of n.

2. Input Format:

The first line of input contains an integer t - the amount of numbers for which you need to calculate the requested sum.

In each of the following rows t there is one integer n.

3. Output Format:

For each of t numbers of n, output the requested fabulous sum.

4. Constraints:

$1 \le t \le 100$

$1 \le n \le 10^9$

5. Samples Input/Output:

	Input	Output
1	2 4 1000000000	-4 499999998352516354
2	1 414231	85792819222

6. Test cases

	Input	Output
1	1 121	7127
2	1 414234	85794061921
3	1	13351

	166	
4	2 25 34	263 469
5	1 222	24243
6	1 1000	498454
7	13 1 19 31 19 19 92 74 69 32 32 91 42 73	-1 128 434 128 128 4024 2521 2161 402 402 3932 777 2447

7. Solution (Java)

```
import java.util.Scanner;
```

```
public class SampleClass {
public static void main(String[] args) {
try (Scanner s = new Scanner(System.in)) {
final int t = s.nextInt();
for (int i = 0; i < t; ++i) {
final long n = s.nextLong();
long l = 1;
while (2 * l <= n) {
l *= 2;
}
final long res = (n * (n + 1)) / 2 - 2 * (2 * l - 1);
System.out.println(res);
}
}
}
}
```

11

Interesting Game

1. **Problem statement:**

In a very ancient country, there was a popular game. The game is played by two people. It begins with the the first player writing an s1 string consisting of exactly nine digits and representing a number that is not greater than a. Then, after seeing s1, the second player writes a string s2 consisting of exactly nine digits and representing a number that is not greater than b. Here, a and b are given constants, whereas strings s1 and s2 are chosen by players. Leading zeros in the strings are allowed.

If the number formed by concatenating (gluing) the strings s1 and s2 is divisible by mod, the second player wins. Otherwise, the first player wins. You are given numbers a, b and mod. You need to determine the winner of this game, considering that both players make the optimal choices. If the first player wins, you also need to find the lexicographically minimal winning move.

2. Input Format:

The first line contains three integers a, b, mod.

3. Output Format:

If the first player wins, print out "1" and the lexicographically minimal string they need to write out in order to win. If the second player wins, print out a single number "2".

4. Constraints:

`0 ≤ a, b ≤ 10^9, 1 ≤ mod ≤ 10^7`

5. Samples Input/Output:

	Input data	Output data
1	1 10 7	2
2	4 0 9	1 000000001

6. Test Cases:

	Input data	Output data
1	10 7 8	2
2	13 4 51	1 000000001
3	1960930 562910 606828	1 000000011
4	1000000000 0 1	2

5	9902593 9902584 9902593	1 002490619
6	6 4 10	2
7	972037745 4602117 5090186	1 000000011

7. Sample solution (Java):

```
import java.io.*;
import java.util.*;
public class Solution {
public static void main(String[] args) throws IOException {
BufferedReader f = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st = new StringTokenizer(f.readLine());
int a = Integer.parseInt(st.nextToken());
int b = Integer.parseInt(st.nextToken());
int mod = Integer.parseInt(st.nextToken());
if (a == 0 || b >= mod-1 || 1000000000 % mod == 0)
System.out.println(2);
else
{
long c = 1000000000 % mod;
int x = 1;
while (x <= (Math.min(a,mod-1)) && (mod-(c*x % mod))%mod <= b)
x++;
```

```
if (x > (Math.min(a,mod-1)))
System.out.println(2);
else
{
String str = ""+x;
while (str.length() < 9)
str = "0"+str;
System.out.println("1 "+str);
}
}
}
}
```

12

Little Bear With Cards

1. **Problem statement:**

A Little Bear loves to play with colored cards.

He has n cards, each one has exactly two colors (the color on the front side and a color on the back side). Initially, all the cards are placed on the table showing their front side. In one step the Little Bear can turn any card over to its other side. The Little Bear finds a set of cards on the table to be Jolly if at least one half of the cards on the table are of the same color (for each card, we consider the color of the side facing the Little Bear on the table)

Help the Little Bear find the minimum number of steps required to convert a set of n cards into a Jolly set of cards.

2. **Input Format:**

The first line contains a single integer n - the number of cards. The next n lines contain the descriptions of all the cards, one card per line. The cards are described by a pair of positive integers not exceeding 10^9, - the colors on both sides. The first number in the string is the color on the front

side of the card, the second - the color on its back side. The color on the front side may match the color on the back side.

The numbers in the lines are separated by single spaces.

3. Output Format:

On a single line print out a single integer - the required minimum number of steps. If it's impossible to turn the set of cards into a Joyful set, print out -1.

4. Constraints:

$1 \le n \le 10^5$

5. Samples Input/Output:

	Input data	Output data
1	3 4 7 4 7 7 4	0
2	5 4 7 7 4 2 11 9 7 1 1	2

6. Test Cases:

	Input data	Output data
1	1 1 1	1
2	3 7 7 1 2 2 1	1

3	4 1000000000 1000000000 999999999 1000000000 999999997 999999998 47 74	1
4	10 1000000000 999999999 47 74 47474 75785445 8798878 458445 1 2 888888888 777777777 99999999 1000000000 9999999 1000000000 999999 1000000000 99999 1000000000	4
5	2 1 1 1 1	0
6	7 1 2 2 3 3 4 4 5 5 6 6 7 7 8	-1
7	3 7 7 1 2 2 1	1

7. Sample solution (Java):

```
import java.util.*;

public class Solution {

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int n1 = n;

if(n%2 != 0){

n1++;

}

int aux;

Map<Integer,int[]> x = new TreeMap<Integer,int[]>();

int res = -1;

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

int r[] = {0,0};

int r1[] = {0,0};

int a = sc.nextInt();

if(x.containsKey(a)){

r = x.get(a);

r[0] += 1;

x.put(a,r);

}else{r[0] = 1;

r[0] = 1;

x.put(a,r);

}

int b = sc.nextInt();

if(b!=a){

if(x.containsKey(b)){
```

```
r1 = x.get(b);
r1[1] += 1;
x.put(b, r1);
}else{
r1[1] = 1;
x.put(b,r1);
}
}
if(x.get(a)[0] + x.get(a)[1] >= n1/2){
if(x.containsKey(res)){
if(x.get(a)[0] > x.get(res)[0]){
res = a;
}
}else{
res = a;
}
}
if(x.get(b)[0] + x.get(b)[1] >= n1/2){
if(x.containsKey(res)){
if(x.get(b)[0] > x.get(res)[0]){
res = b;
}
}else{
res = b;
}
}


}
```

```
if(res == -1){
System.out.println(res);
}else{
if(x.get(res)[0] >= n1/2 ){
System.out.println("0");
}else{
System.out.println(n1/2 - x.get(res)[0]);
}
}


}
}
```

13

Mom Loves Art

1. **Problem statement:**

At the "Sotheby's" auction, Mom bought two rare paintings and now wants to hang them on the wall. To do this, she purchased a special exhibition stand to mount it on the wall, and place the pictures in it. The stand’s working surface, where the pictures will de displayed, has the shape of a rectangle a1 × b1, and the paintings themselves are shaped as rectangles a2 × b2 and a3 × b3. It is believed that the stand’s supporting structure does not occupy any extra space.

Since the pictures are painted in the style of abstract art, it does not really matter how they are rotated, but nevertheless, one side of the stand and each picture both, should be parallel to the floor. Pictures can touch each other and the edges of the stand’s working surface, but they can not intersect or overreach the edges of the stand’s working surface. Mom asks if she can put the pictures in the stand or if it’s simply not large enough.

2. Input Format:

The first line contains two space-delimited integers a1 and b1 - the part of the stand’s working surface. The next two lines are of a2, b2, a3 and b3 - the sides of the pictures.

3. Output Format:

If the pictures can fit in the stand, print «YES» (without the quotation marks), and if not, print «NO» (without the quotation marks).

4. Constraints:

All input data in ai and bi are integers ranging from 1 to 1000.

5. Samples Input/Output:

	Input data	Output data
1	3 2 1 3 2 1	YES
2	5 5 3 3 3 3	NO

6. Test Cases:

	Input data	Output data
1	4 2 2 3 1 2	YES
2	3 3 1 1 1 1	YES
3	1000 1000 999 999 1 1000	YES
4	7 3 7 8	NO

	1 5	
5	7 7 5 5 2 4	YES
6	3 3 2 2 2 2	NO
7	2 9 5 1 3 2	YES

7. Sample solution (Java):

```
import java.util.Scanner;
public class Sollution
 {
public static Scanner scanner = new Scanner(System.in);
 static public void main (String []args)
{
int board[] = new int[2];
int paint1[] = new int[2];
 int paint2[] = new int[2];
board[0] = scanner.nextInt();
board[1] = scanner.nextInt();
paint1[0] = scanner.nextInt();
```

```
paint1[1] = scanner.nextInt();

paint2[0] = scanner.nextInt();

paint2[1] = scanner.nextInt();

for(int i = 0 ; i < 2 ; i++)

for(int j = 0 ; j < 2 ; j++)

for(int k = 0; k < 2; k++)

 if(paint1[i] + paint2[j]<= board[k])

if(paint1[1-i] <= board[1-k] && paint2[1-j] <= board[1-k])

{

System.out.println("YES");

 return;

}

System.out.println("NO");

}

}
```

14

Notebook with Numbers

1. **Problem statement:**

James has a notebook with n sheets numbered from 1 to n. In the i-th sheet, there is a an integer a written down.

Jessica is going to tear the notebook into k pieces, for that she chooses a k − 1 number $1 \le r_1 < r_2 < \ldots < r_{k-1} < n$ and tears the notebook so that the sheets from 1 to r_1-st are in the first part, the sheets from ($r_1 + 1$) -th to r_2-d are in the second part, etc., the last k-th part contains sheets from($r_{k-1} + 1$) -th to n-th.

After Jessica has torn the notebook, James will find the minimum number in each of the resulting parts and try to piece them back together. That's why Jessica wants to tear the notebook so that the

resulting amount is as great as possible. Help her choose a way to tear the notebook to maximize the sum of the minimum values.

2. Input Format:

The first line contains two numbers: n and k . The second line contains n integers: a_1, a_2,. . . , A_n .

3. Output Format:

On the first line, print out the maximum sum amount that Jessica can reach. On the second line, print out the values r_1, r_2,. . . , r_k-1, that she has to choose. If there are multiple options to tear the notebook to maximize the desired amount, output any of them.

4. Constraints:

$2 \leq k \leq n \leq 300$

$1 \leq a_i \leq 10^9$

5. Samples Input/Output:

	Input data	Output data
1	10 5 1 10 2 8 9 3 5 4 7 6	27 3 4 5 8
2	3 2 7 6 4	11 1

6. Test Cases:

	Input data	Output data
1	4 4 1 2 3 4	10 1 2 3

2	4 2 5 2 7 6	8 2
3	5 2 1 2 3 4 5	6 4
4	3 3 9 32 76	117 1 2
5	5 2 92 63 51 96 22	114 1
6	10 2 9 9 9 8 8 8 7 7 7 6	15 1
7	6 5 10 20 30 40 50 60	190 2 3 4 5

7. Sample solution (Java):

```
package com.company;

import java.util.Scanner;

public class Notebook {
    static long[] a;
    static long[] ans;
    static long[][] rmq;
    static long[][] d;
    static int[][] pr;
```

```
static int n;
static int k;
static long cur;
static int asize;

public static void main (String[] args) throws java.lang.Exception
{
    Scanner input = new Scanner(System.in);
    n = input.nextInt();
    k = input.nextInt();

    a = new long[300];
    rmq = new long[300][300];
    d = new long[300][300];
    pr = new int[300][300];
    ans = new long[300];

    for (int i=1; i<=n; i++){
        a[i] = input.nextLong();
    }

    for (int l=1; l<=n; l++) {
        for (int r=l;r<=n;r++) {
            cur = a[l];
            for (int i = l; i <= r; i++) {
                cur = Math.min(cur, a[i]);
            }
            rmq[l][r] = cur;
        }
    }

    for (int i=1; i <= n; i++) {
        d[0][i] = rmq[1][i];
    }

    for (int i=1; i<k; i++) {
        for (int j=1; j<=n; j++) {
            for (int l = 1; l <= j; l++) {
                if (d[i][j] < d[i-1][l] + rmq[l+1][j]) {
                    d[i][j] = d[i-1][l] + rmq[l+1][j];
                    pr[i][j] = l;
```

```
                }
            }
        }
    }

    System.out.println(d[k-1][n]);

    int i = k-1;
    int j = n;
    asize = 0;
    while (pr[i][j] != 0) {
        asize++;
        ans[asize] = pr[i][j];
        j = pr[i][j];
        i--;
    }
    for (i = asize; i > 0; i--) {
        System.out.println(ans[i]);
    }
  }
}
```

15

Palindrome game

1. **Problem statement:**

Two people are playing a game following this scenario:

1) Given a string STR, they can remove any single letter from it, in each turn.

2) If, prior to removing one letter, a player can change the order of STR's letters to form a palindrome, they win.

3) STR contains only lowercase English letters.

A palindrome is a string that reads the same from right to left and left to right ("abba", "abcba" and 'aaaa" are palindroms, but "abcd" isn't).

If both players are smart and play the game perfectly, calculate which player wins.

2. Input Format

The input contains a single line, containing a string STR with a length between 1 and 1000 characters.

3. Output Format:

Print out "First" if the first player wins or "Second" is the second player wins.

4. Constraints:

The string STR contains only lowercase English letters.

5. Samples Input/Output:

	Input	Output
1	aba	First
2	abca	Second

6. Test cases

	Input	Output
1	aabb	First
2	ctjxzuimsxnarlciuynqeoqmmbqtagszuo	Second
3	gevqgtaorjixsxnbcoybr	First
4	abcdfega	Second
5	asdfasdfssdffssdfasdfghj	First
6	fsssssssssf	First
7	gevqgtsdsdkkfhhfffcoybrr	Second

8	gevqgsssssffddqqwer	First

7. Solution (Java)

```
import java.util.Scanner;

public class Main {

public static void main(String ar[]) {

Scanner in = new Scanner(System.in);

String str = in .next();

int arr[] = new int[26];

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

arr[str.charAt(i) - 'a']++;

}

int odd = 0;

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

if (arr[i] % 2 != 0)

odd++;

}

if (odd == 0 || odd % 2 != 0) {

System.out.println("First");

} else {

System.out.println("Second");

}

}

}
```

16

Inmates

1. **Problem statement:**

There are currently n inmates in your town's prison. Since there are few available cells left in the prison, the mayor of your town decides to move the inmates to a prison in another town.

To do this, the mayor has formed all n prisoners in a line. Every prisoner has an integer written on their chest, that stands for the severity of their crime. The higher the number, the more severe their crime is.

The mayor asked you to choose c inmates who will be transferred to another prison. He has two conditions:

The selected c inmates should form a segment of the line. In other words, the inmates must go in a sequence, without leaving any gaps in the line.

The severity of the crime commited by each inmate must not be greater than t. Otherwise, an inmate is considered to be particularly dangerous, and the mayor is afraid that they will escape during the transfer.

How many ways are there to select c inmates while satisfying both of the mayor's conditions?

2. Input Format:

The first line contains three space-delimited integers n , t and c. The next line contains n space-delimited integers, the i-th number stands for the severity of the crime commited by the i-th inmate in the line (a positive number not greater than 10^9).

3. Output Format:

Print out a single integer - the number of ways to select inmatess.

4. Constraints:

$1 \leq n \leq 2 \cdot 10^5$

$0 \leq t \leq 10^9$

$1 \leq c \leq n$

5. Samples Input/Output

	Input data	Output data
1	4 3 3 2 3 1 1	2
2	1 1 1 2	0

6. Test Cases:

	Input data	Output data
1	11 4 2 2 2 0 7 3 2 2 4 9 1 4	6
2	57 2 10 7 5 2 7 4 1 0 5 2 9 2 9 8 6 6 5 9 6 8 1 0 1 0 3 2 6 5 2 8 8 8 8 0 9 4 3 6 6 2 4 5 1 2 0 1 7 1 1 5 4 5 0 7 5 1 9 6	0
3	2 228885628 1 90897004 258427916	1
4	1 228 1 1	1
5	1 3 1 1	1
6	2 2 2 1 2	1

7	5 1 3 1 2 3 2 1	0

7. Sample solution (Java):

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Sollution {
public static void main(String[] args) throws IOException {
BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
String[] input = bf.readLine().split(" ");
int n = Integer.parseInt(input[0]);
int t = Integer.parseInt(input[1]);
int c = Integer.parseInt(input[2]);
input = bf.readLine().split(" ");
int count = 0;
int sum = 0;
for (int i = 0; i < n; i++) {
if (Integer.parseInt(input[i]) <= t) {
count++;
}
else {
if (count-c+1 > 0) {
```

```
sum += count-c+1;
}
count = 0;
}
}

if (count-c+1 > 0) {
sum += count-c+1;
}
System.out.println(sum);

}
}
```

17

Robot's Task

1. **Problem statement:**

Robot Doc is in a hall with an n number of computers numbered from left to right from 1 to n. Each computer contains exactly one piece of information, and Doc wants to collect each of those pieces eventually. The computers are equipped with a security system, so to crack the i-th of them, the robot needs to collect at least an ai amount of any information pieces from other computers. To carry out the hacking, Doc needs to stand directly next to the computer it's hacking.

The robot is designed using modern technologies and can move along a row of computers in any of the two possible directions, but a change of direction consumes a large amount of Doc's resources. Provide a minimum number of direction changes that is necessary to make the robot collect all n pieces of information, assuming that Doc's initial position was next to the computer number 1.

It is guaranteed that there exists at least one sequence of robot actions that allows Doc to successfully collect all the information. Initially, Doc does not own any information pieces.

2. Input Format:

The first line contains a number *n*. The second line contains *n* non-negative integers *a*1, *a*2, ..., *an*, separated by a space. It is guaranteed that there is a way for the robot to collect all the information pieces.

3. Output Format:

Print out a single number — the minimum number of direction changes the robot will have to make in order to collect all *n* pieces of information.

4. Constraints:

$1 \le n \le 1000$

$0 \le ai < n$

5. Samples Input/Output:

	Input data	Output data
1	3 0 2 0	1
2	5 4 2 3 0 1	3

6. Test Cases:

	Input data	Output data
1	7 0 3 1 0 5 2 6	2
2	9 3 5 6 8 7 0 4 2 1	5
3	10 7 1 9 3 5 8 6 0 2 4	9

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

7. Sample solution (Java):

```
import java.util.Scanner;
import java.util.ArrayList;
public class Sollution {
public static void main(String[] args) {
Scanner sc=new Scanner(System.in);
int num = sc.nextInt();
sc.nextLine();
String[] strings = sc.nextLine().split(" ");
int collected = 0;
int switches = 0;
int direction = 1;
int needed;
for (int i = 1;; i+=direction){
```

```
needed = Integer.parseInt(strings[i-1]);
if (collected >= needed){
collected++;
strings[i-1] = "999999";
}
if (collected == num)
break;
if ((i+direction)>num){
direction = -1;
switches++;
continue;
}
else if ((i+direction) < 1){
direction = 1;
switches++;
continue;
}
}
System.out.println(switches);
}
}
```