**Course 3-Optional Contest 01**

**VVI\*\*\* Between a and b ... AtCoder - abc048\_b**

**/\* int modulo division Between a and b, inclusive, how many are divisible by x? \*/**

**Slow program**

**Fast Program**

**Problem Statement**

You are given nonnegative integers *a* and *b* (*a ≤ b*), and a positive integer *xx*. Among the integers between *a* and *b*, inclusive, how many are divisible by *x*?

**Constraints**

* *0 ≤ a ≤ b ≤ 10^{18}0≤a≤b≤1018*
* *1 ≤ x ≤ 10^{18}1≤x≤1018*

**Input**

The input is given from Standard Input in the following format:

*a* *b* *x*

**Output**

Print the number of the integers between *a* and *b*, inclusive, that are divisible by *x*.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 4 8 2 | 3 |

There are three integers between *44* and *88*, inclusive, that are divisible by *22*: *44*, *66* and *88*.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 0 5 1 | 6 |

There are six integers between *0* and *5*, inclusive, that are divisible by *1*: *0*, *1*, *2*, *3*, *4* and *5*.

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 9 9 2 | 0 |

There are no integer between *9* and *9*, inclusive, that is divisible by *2*.

**Sample 4**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 1 1000000000000000000 3 | 333333333333333333 |

Watch out for integer overflows.

**Slow Program. Failed for long long:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

long long int a, b, x;

cin>>a>>b>>x;

long long int count=0;

for(int i=a; i<=b; i++)

{

if(i%x==0)

{

count++;

}

}

cout<<count<<endl;

}

**Fast:**

/\* Between a and b, inclusive, how many are divisible by x? \*/

#include<iostream>

using namespace std;

int main()

{

long long a, b, x;

cin>>a>>b>>x;

long long count=b/x;

if(a%x == 0)

{

count= count-a/x;

count++;

}

else

{

count= count-a/x;

}

cout<<count<<endl;

}

**(Imp\*) Three Integers AtCoder - arc143\_a**

**Problem Statement**

Three non-negative integers *A*, *B*, and *C* are written on a blackboard. You can perform the following two operations any number of times in any order.

* Subtract *1* from two of the written integers of your choice.
* Subtract *1* from all of the written integers.

Your objective is to make all the numbers on the blackboard *0*. Determine whether it is achievable. If it is, find the minimum number of times you need to perform an operation to achieve it.

**Constraints**

* *0≤A,B,C≤10^{18}*

**Input**

Input is given from Standard Input in the following format:

*A* *B* *C*

**Output**

If the objective is unachievable, print -1. If it is achievable, print the minimum number of times you need to perform an operation to achieve it.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 2 2 3 | 3 |

Here is one way to make all the numbers *00*.

* Subtract *1* from *A* and *C*. Now the numbers are *1*, *2*, *2*.
* Subtract *1* from *B* and *C*. Now the numbers are *1*, *1*, *1*.
* Subtract *1* from all the numbers. Now the numbers are *0*, *0*, *0*.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 0 0 1 | -1 |

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 0 0 0 | 0 |

#include<bits/stdc++.h>

using namespace std;

int main()

{

long long a, b, c;

cin>>a>>b>>c;

long long max1, max2, min1;

max1 = max(a, max(b, c));

if(max1 == a)

{

max2= max(b, c);

}

else if(max1 == b)

{

max2= max(a, c);

}

else if(max1 == c)

{

max2= max(a, b);

}

min1= min(a, min(b, c));

if(max1-max2>min1)

{

cout<<-1;

}

else

{

cout<<max1;

}

}

**(Imp\*) String Lexicographical sorting Iroha Loves Strings (ABC Edition) AtCoder - abc042\_b**

**Problem Statement**

Iroha has a sequence of *NN* strings *S\_1, S\_2, ..., S\_NS1​,S2​,...,SN​*. The length of each string is *LL*.

She will concatenate all of the strings in some order, to produce a long string.

Among all strings that she can produce in this way, find the lexicographically smallest one.

Here, a string *s=s\_1s\_2s\_3s=s1​s2​s3​*...*s\_nsn​* is *lexicographically smaller* than another string *t=t\_1t\_2t\_3t=t1​t2​t3​*...*t\_mtm​* if and only if one of the following holds:

* There exists an index *i(1≦i≦min(n,m))i(1≦i≦min(n,m))*, such that *s\_j = t\_jsj​=tj​* for all indices *j(1≦j<i)j(1≦j<i)*, and *s\_i<t\_isi​<ti​*.
* *s\_i = t\_isi​=ti​* for all integers *i(1≦i≦min(n,m))i(1≦i≦min(n,m))*, and *n<mn<m*.

**Constraints**

* *1 ≦ N, L ≦ 1001≦N,L≦100*
* For each *ii*, the length of *S\_iSi​* equals *LL*.
* For each *ii*, *S\_iSi​* consists of lowercase letters.

**Input**

The input is given from Standard Input in the following format:

*NN* *LL*

*S\_1S1​*

*S\_2S2​*

:

*S\_NSN​*

**Output**

Print the lexicographically smallest string that Iroha can produce.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 3 3  dxx  axx  cxx | axxcxxdxx |

The following order should be used: axx, cxx, dxx.

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n, l;

cin>>n>>l;

string s[101];

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

{

cin>>s[i];

}

sort(s, s+n);

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

{

cout<<s[i];

}

}

**(Imp\*) Pattern Thin AtCoder - abc049\_b**

**Problem Statement**

There is an image with a height of *HH* pixels and a width of *WW* pixels. Each of the pixels is represented by either . or \*. The character representing the pixel at the *ii*-th row from the top and the *jj*-th column from the left, is denoted by *C\_{i,j}Ci,j​*.

Extend this image vertically so that its height is doubled. That is, print a image with a height of *2H2H* pixels and a width of *WW* pixels where the pixel at the *ii*-th row and *jj*-th column is equal to *C\_{(i+1)/2,j}C(i+1)/2,j​* (the result of division is rounded down).

**Constraints**

* *1≦H, W≦1001≦H,W≦100*
* *C\_{i,j}Ci,j​* is either . or \*.

**Input**

The input is given from Standard Input in the following format:

*HH* *WW*

*C\_{1,1}...C\_{1,W}C1,1​...C1,W​*

:

*C\_{H,1}...C\_{H,W}CH,1​...CH,W​*

**Output**

Print the extended image.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 2 2  \*.  .\* | \*.  \*.  .\*  .\* |

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 1 4  \*\*\*. | \*\*\*.  \*\*\*. |

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 9 20  .....\*\*\*....\*\*\*.....  ....\*...\*..\*...\*....  ...\*.....\*\*.....\*...  ...\*.....\*......\*...  ....\*.....\*....\*....  .....\*\*..\*...\*\*.....  .......\*..\*.\*.......  ........\*\*.\*........  .........\*\*......... | .....\*\*\*....\*\*\*.....  .....\*\*\*....\*\*\*.....  ....\*...\*..\*...\*....  ....\*...\*..\*...\*....  ...\*.....\*\*.....\*...  ...\*.....\*\*.....\*...  ...\*.....\*......\*...  ...\*.....\*......\*...  ....\*.....\*....\*....  ....\*.....\*....\*....  .....\*\*..\*...\*\*.....  .....\*\*..\*...\*\*.....  .......\*..\*.\*.......  .......\*..\*.\*.......  ........\*\*.\*........  ........\*\*.\*........  .........\*\*.........  .........\*\*......... |

#include<bits/stdc++.h>

using namespace std;

int main()

{

int h, w;

cin>>h>>w;

string s;

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

{

cin>>s;

cout<<s<<endl<<s<<endl;

}

}

**(Imp\*) Pattern Minesweeper AtCoder - abc075\_b**

**Problem Statement**

You are given an *H × WH×W* grid.  
The squares in the grid are described by *HH* strings, *S\_1,...,S\_HS1​,...,SH​*.  
The *jj*-th character in the string *S\_iSi​* corresponds to the square at the *ii*-th row from the top and *jj*-th column from the left *(1 \leq i \leq H,1 \leq j \leq W)(1≤i≤H,1≤j≤W)*.  
. stands for an empty square, and # stands for a square containing a bomb.

Dolphin is interested in how many bomb squares are horizontally, vertically or diagonally adjacent to each empty square.  
(Below, we will simply say "adjacent" for this meaning. For each square, there are at most eight adjacent squares.)  
He decides to replace each . in our *HH* strings with a digit that represents the number of bomb squares adjacent to the corresponding empty square.

Print the strings after the process.

**Constraints**

* *1 \leq H,W \leq 501≤H,W≤50*
* *S\_iSi​* is a string of length *WW* consisting of # and ..

**Input**

Input is given from Standard Input in the following format:

*HH* *WW*

*S\_1S1​*

*::*

*S\_HSH​*

**Output**

Print the *HH* strings after the process.  
The *ii*-th line should contain a string *T\_iTi​* of length *WW*, where the *jj*-th character in *T\_iTi​* corresponds to the square at the *ii*-th row from the top and *jj*-th row from the left in the grid *(1 \leq i \leq H, 1 \leq j \leq W)(1≤i≤H,1≤j≤W)*.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 3 5  .....  .#.#.  ..... | 11211  1#2#1  11211 |

For example, let us observe the empty square at the first row from the top and first column from the left.  
There is one bomb square adjacent to this empty square: the square at the second row and second column.  
Thus, the . corresponding to this empty square is replaced with 1.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 3 5  #####  #####  ##### | #####  #####  ##### |

It is possible that there is no empty square.

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 6 6  #####.  #.#.##  ####.#  .#..#.  #.##..  #.#... | #####3  #8#7##  ####5#  4#65#2  #5##21  #4#310 |

#include<bits/stdc++.h>

using namespace std;

int main()

{

long long h, w;

cin>>h>>w;

char arr[h+1][w+1];

for(long long i=0; i<h; ++i)

{

for(long long j=0; j<w; j++)

{

cin>>arr[i][j];

}

}

for(long long i=0; i<h; ++i)

{

for(long long j=0; j<w; j++)

{

if(arr[i][j] == '.')

{

long long n=0;

if(arr[i][j+1] == '#')

{

n++;

}

if(arr[i][j-1] == '#')

{

n++;

}

if(arr[i-1][j-1] == '#')

{

n++;

}

if(arr[i-1][j] == '#')

{

n++;

}

if(arr[i-1][j+1] == '#')

{

n++;

}

if(arr[i+1][j-1] == '#')

{

n++;

}

if(arr[i+1][j] == '#')

{

n++;

}

if(arr[i+1][j+1] == '#')

{

n++;

}

cout<<n;

}

else

{

cout<<"#";

}

}

cout<<endl;

}

}

**\*Problem Statement**

You have an integer variable *xx*. Initially, *x=0x=0*.

Some person gave you a string *SS* of length *NN*, and using the string you performed the following operation *NN* times. In the *ii*-th operation, you incremented the value of *xx* by *11* if *S\_i=Si​=*I, and decremented the value of *xx* by *11* if *S\_i=Si​=*D.

Find the maximum value taken by *xx* during the operations (including before the first operation, and after the last operation).

**Constraints**

* *1≤N≤1001≤N≤100*
* *|S|=N∣S∣=N*
* No characters except I and D occur in *SS*.

**Input**

The input is given from Standard Input in the following format:

*NN*

*SS*

**Output**

Print the maximum value taken by *xx* during the operations.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 5  IIDID | 2 |

After each operation, the value of *xx* becomes *11*, *22*, *11*, *22* and *11*, respectively. Thus, the output should be *22*, the maximum value.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 7  DDIDDII | 0 |

The initial value *x=0x=0* is the maximum value taken by *xx*, thus the output should be *00*.

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin>>n;

char s[101];

cin>>s;

int x=0, j=0, a[100];

a[j]=x;

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

{

if(s[i]=='I')

{

x=x+1;

}

else if(s[i]=='D')

{

x=x-1;

}

a[++j]=x;

}

int max= a[0];

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

{

if(a[j]>max)

{

max= a[j];

}

}

cout<<max<<endl;

}

**Problem Statement**

Let *ww* be a string consisting of lowercase letters. We will call *ww* *beautiful* if the following condition is satisfied:

* Each lowercase letter of the English alphabet occurs even number of times in *ww*.

You are given the string *ww*. Determine if *ww* is beautiful.

**Constraints**

* *1 \leq |w| \leq 1001≤∣w∣≤100*
* *ww* consists of lowercase letters (a-z).

**Input**

The input is given from Standard Input in the following format:

*ww*

**Output**

Print Yes if *ww* is beautiful. Print No otherwise.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| abaccaba | Yes |

a occurs four times, b occurs twice, c occurs twice and the other letters occur zero times.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| hthth | No |

#include<bits/stdc++.h>

using namespace std;

int main()

{

char s[101];

cin>>s;

char c[27]={0};

char a[27]="abcdefghijklmnopqrstuvwxyz";

int len= strlen(s);

if(len%2!=0)

{

cout<<"No"<<endl;

return 0;

}

else if(len%2==0)

{

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

{

for(int j=0; j<26; j++)

{

if(s[i]==a[j])

{

c[j]++;

}

}

}

}

int p;

for(int i=0; i<26; i++)

{

if(c[i]%2!=0)

{

p=-1;

break;

}

else

{

p=1;

}

}

if(p==1)

{

cout<<"Yes"<<endl;

}

else

{

cout<<"No"<<endl;

}

}

**Problem Statement**

Snuke has decided to construct a string that starts with A and ends with Z, by taking out a substring of a string *ss* (that is, a consecutive part of *ss*).

Find the greatest length of the string Snuke can construct. Here, the test set guarantees that there always exists a substring of *ss* that starts with A and ends with Z.

**Constraints**

* *1 ≦ |s| ≦ 200{,}0001≦∣s∣≦200,000*
* *ss* consists of uppercase English letters.
* There exists a substring of *ss* that starts with A and ends with Z.

**Input**

The input is given from Standard Input in the following format:

*ss*

**Output**

Print the answer.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| QWERTYASDFZXCV | 5 |

By taking out the seventh through eleventh characters, it is possible to construct ASDFZ, which starts with A and ends with Z.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| ZABCZ | 4 |

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| HASFJGHOGAKZZFEGA | 12 |

#include<bits/stdc++.h>

using namespace std;

int main()

{

char s[200001];

cin>>s;

int len= strlen(s);

int a, b;

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

{

if(s[i]=='A')

{

a=i;

break;

}

}

for(int i=len-1; i>0; i--)

{

if(s[i]=='Z')

{

b=i;

break;

}

}

cout<<b-a+1<<endl;

}

**Problem Statement**

Dolphin loves programming contests. Today, he will take part in a contest in AtCoder.  
In this country, 24-hour clock is used. For example, *9:009:00* p.m. is referred to as "*2121* o'clock".  
The current time is *AA* o'clock, and a contest will begin in exactly *BB* hours. When will the contest begin? Answer in 24-hour time.

**Constraints**

* *0 \leq A,B \leq 230≤A,B≤23*
* *AA* and *BB* are integers.

**Input**

The input is given from Standard Input in the following format:

*AA* *BB*

**Output**

Print the hour of the starting time of the contest in 24-hour time.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 9 12 | 21 |

In this input, the current time is *99* o'clock, and *1212* hours later it will be *2121* o'clock in 24-hour time.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 19 0 | 19 |

The contest has just started.

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 23 2 | 1 |

The contest will begin at *11* o'clock the next day.

#include<bits/stdc++.h>

using namespace std;

int main()

{

int a, b;

cin>>a>>b;

if((a+b)<24)

{

cout<<a+b<<endl;

}

else if((a+b)>=24)

{

cout<<(a+b)-24<<endl;

}

}

**Problem Statement**

Alice and Bob are playing *One Card Poker*.  
One Card Poker is a two-player game using playing cards.

Each card in this game shows an integer between 1 and 13, inclusive.  
The *strength* of a card is determined by the number written on it, as follows:

Weak 2 *<<* 3 *<<* 4 *<<* 5 *<<* 6 *<<* 7 *<<* 8 *<<* 9 *<<* 10 *<<* 11 *<<* 12 *<<* 13 *<<* 1 Strong

One Card Poker is played as follows:

1. Each player picks one card from the deck. The chosen card becomes the player's hand.
2. The players reveal their hands to each other. The player with the stronger card wins the game.  
   If their cards are equally strong, the game is drawn.

You are watching Alice and Bob playing the game, and can see their hands.  
The number written on Alice's card is *AA*, and the number written on Bob's card is *BB*.  
Write a program to determine the outcome of the game.

**Constraints**

* *1≦A≦131≦A≦13*
* *1≦B≦131≦B≦13*
* *AA* and *BB* are integers.

**Input**

The input is given from Standard Input in the following format:

*AA* *BB*

**Output**

Print Alice if Alice will win. Print Bob if Bob will win. Print Draw if the game will be drawn.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 8 6 | Alice |

8 is written on Alice's card, and 6 is written on Bob's card. Alice has the stronger card, and thus the output should be Alice.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 1 1 | Draw |

Since their cards have the same number, the game will be drawn.

**Sample 3**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 13 1 | Bob |

#include<bits/stdc++.h>

using namespace std;

int main()

{

int a, b;

cin>>a>>b;

if(a==1 && b!=1)

{

cout<<"Alice"<<endl;

}

else if(a!=1 && b==1)

{

cout<<"Bob"<<endl;

}

else if(a>b)

{

cout<<"Alice"<<endl;

}

else if(a<b)

{

cout<<"Bob"<<endl;

}

else

{

cout<<"Draw"<<endl;

}

}

**Problem Statement**

Joisino is working as a receptionist at a theater.

The theater has *100000100000* seats, numbered from *11* to *100000100000*.

According to her memo, *NN* groups of audiences have come so far, and the *ii*-th group occupies the consecutive seats from Seat *l\_ili​* to Seat *r\_iri​* (inclusive).

How many people are sitting at the theater now?

**Constraints**

* *1≤N≤10001≤N≤1000*
* *1≤l\_i≤r\_i≤1000001≤li​≤ri​≤100000*
* No seat is occupied by more than one person.
* All input values are integers.

**Input**

Input is given from Standard Input in the following format:

*NN*

*l\_1l1​* *r\_1r1​*

*::*

*l\_NlN​* *r\_NrN​*

**Output**

Print the number of people sitting at the theater.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 1  24 30 | 7 |

There are *77* people, sitting at Seat *24,25,26,27,28,2924,25,26,27,28,29* and *3030*.

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 2  6 8  3 3 | 4 |

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin>>n;

int a, b, p=0;

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

{

cin>>a>>b;

p = p+(b-a+1);

}

cout<<p<<endl;

}