```java
package com.eomcs.quiz.ex01.sol;

// 출처: codefights.com
//
// 한번에 한 자리의 숫자만 바꿀 때 이전 값과 다르게 만들 수 있는 경우는 몇가지인가?
// 단 맨 앞의 숫자는 0이 될 수 없다.
// 예)
// 10 => 17 번
//
// [시간 복잡도]
// - O(k) : k는 10진수의 자릿수이다.
//
public class Test05 {

public static void main(String[] args) {
System.out.println(countWaysToChangeDigit(10) == 17);
}

static int countWaysToChangeDigit(int value) {
int answer = 0;
while (value > 0) {
answer += 9 - value % 10;
value /= 10;
}
return answer;
}
}
```

1	0	3	4	7	6		
2	1	9+	9+	9+	9+	36-1 = 35번 바꿀 수 있음.	
3	2						
4	3	9-2=7	9-3=6	9-4=5	9-5=4		
5	4	2	3	4	5		
6	5	3	4	5	6		
7	6	4	5	6	7		
8	7	5	6	7	8		
9	8	6	7	8	9		
	9	7	8	9			
		8	9				
8번	9번	9					

비호비트를 봐야하기 때문에 31만큼 반복한다.

1의 개수는 같지만, 값만 달라짐.

0 0 0 0 1 > 0 1 > 0

10 ---> 0 0 0 0 1 0 1 0

9 0 0 0 0 1 0 0 1

12 0 0 0 0 1 1 0 0

6 0 0 0 0 0 1 1 0

18 0 0 0 1 0 0 1 0

6	**7**	**8**	**9**	**10**	**11**	**12**	**- - -**	**18**

```java
// 양의 정수 x를 2진수로 표현했을 때 1 값을 갖는 비트 개수를 정수의 무게라고 정의할 때,
// 같은 무게를 가지는 양의 정수 중에서 x와 가장 가까운 양의 정수를 구하시오!
public class Test06 {

public static void main(String[] args) {
System.out.println(closestIntSameBit(0b00001010) == 0b00001001); // 10 ==> 9
System.out.println(closestIntSameBit(0b11001000) == 0b11000100); // 200 ==> 196
}

static int closestIntSameBit(int x) {
// 이 메서드를 완성하시오!

// x의 모든 비트가 0이거나 1이면 오류를 반환한다.
throw new IllegalArgumentException("모든 비트가 0 또는 1이다.");
}
}
```

x : 0 0 0 0 1 0 1 0
i = 0;

x >> 0 : 0 0 0 0 1 0 1 0
&1 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

x >>(i+1) ; 0 0 0 0 1 0 1 0
&1 0 0 0 0 0 0 0 1
1

특정 비트 값 추출하기
1. 맨 끝 비트로 이동
2. & 연산을 통해 맨 끝 비트 추출

i = 0
int bit1 = (x >> i) &1; ==> 맨끝 비트를 추출
int bit2 = (x>>(i+1))&1;

두 비트교환을 위한 마스크 준비
0 0 0 0 0 0 0 1 <--- 1 << i
0 0 0 0 0 0 1 0 <--- 1 << i +1 |
0 0 0 0 0 0 1 1 <== mask

x : 0 0 0 0 1 0 1 0
xor : 0 0 0 0 0 0 1 1
0 0 0 0 1 0 0 1

ex02.Test01

```java
public class Test01 {

public static void main(String[] args) {
System.out.println(maxDiff(new int[]{2, 4, 1, 0}) == 3);
System.out.println(maxDiff(new int[]{3, 1, 4, 3, 8, 7}) == 5);
}

static int maxDiff(int[] values) {
int answer = 1;
```

```
// 이 메서드를 완성하시오!
return answer;
}
}
```

ex01.Test07

https://en.wikipedia.org/wiki/Trailing_zero

Factorial [edit]

The number of trailing zeros in the decimal representation of *n*!, the factorial of a non-negative integer *n*, is simply the multiplicity of the prime factor 5 in *n*!. This can be determined with this special case of de Polignac's formula:[1]

$$f(n) = \sum_{i=1}^{k} \left\lfloor \frac{n}{5^i} \right\rfloor = \left\lfloor \frac{n}{5} \right\rfloor + \left\lfloor \frac{n}{5^2} \right\rfloor + \left\lfloor \frac{n}{5^3} \right\rfloor + \cdots + \left\lfloor \frac{n}{5^k} \right\rfloor,$$

where k must be chosen such that

$$5^{k+1} > n,$$

more precisely

$$5^k \le n < 5^{k+1},$$
$$k = \lfloor \log_5 n \rfloor,$$

and $\lfloor a \rfloor$ denotes the floor function applied to *a*. For *n* = 0, 1, 2, ... this is

0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 6, ... (sequence A027868 in the OEIS).

For example, 5^3 > 32, and therefore 32! = 263130836933693530167218012160000000 ends in

$$\left\lfloor \frac{32}{5} \right\rfloor + \left\lfloor \frac{32}{5^2} \right\rfloor = 6 + 1 = 7$$

zeros. If *n* < 5, the inequality is satisfied by *k* = 0; in that case the sum is empty, giving the answer 0.

The formula actually counts the number of factors 5 in *n*!, but since there are at least as many factors 2, this is equivalent to the number of factors 10, each of which gives one more trailing zero.

Defining

$$q_i = \left\lfloor \frac{n}{5^i} \right\rfloor,$$

the following recurrence relation holds:

$$q_0 = n,$$
$$q_{i+1} = \left\lfloor \frac{q_i}{5} \right\rfloor.$$

This can be used to simplify the computation of the terms of the summation, which can be stopped as soon as q_i reaches zero. The condition $5^{k+1} > n$ is equivalent to $q_{k+1} = 0$.

ex02.Test04 등비수열

	108		108	
0	1	2	3	4
2	6	18	54	162
	x3	x3	x3	x3

1번항목과 2번 항목의 비율
1번항목과 0번 항목의 비율
2번항목과 3번 항목의 비율
2번항목과 1번 항목의 비율
- - -
0번과 1번의 비율을 가지고 나머지 값을 구한다.

ex01.Test08

10^2 10^1 10^0 — 20 x 7 = 140
128 x 7 — 2 x 70 = 140
128 7x10^0 = 56
12 7x10 = 140
1x7x10^2 = 700
700+140+56=896

128
x 7
896

128 x 7
10_2 10^1 10^0

8 *7 — 8 * 7
20 * 7 — 2 * 70
100* 7 — 1 * 700

5 : 0101 — 0101 x 0011
3 : 0011

2^0	0001x0011 =	1 * 3 = 3	
~~2^1~~	0000x0110 =	0 * 3 = 0	0과 곱하면 0이기 때문에 버린다.
2^2	0001x1100 =	4 * 3 = 12	
~~2^3~~	0000x11000 =	0 * 3 = 0	0과 곱하면 0이기 때문에 버린다.
		15	

5 : 0101 — 0101 x 0011
3 : 0011

2^0	1 * 0011 =	0 0 1 1	
~~2^1~~	0 * 0011 =	0 0 0 0	0과 곱하면 0이기 때문에 버린다.
2^2	4 * 0011 =	1 1 0 0	
~~2^3~~	0 * 0011 =	0 0 0 0	0과 곱하면 0이기 때문에 버린다.
		1 1 1 1 (15)	

비트 연산자가 이동하면 *2

5 : 0101 — 2^3 2^2 2^1 2^0 — **이진수의 특징**
3 : 0011 — 0101 x 0011

2^0 자리 계산	1 * 0011 =	0 0 1 1	**1 * 0011**

~~2^1 자리 계산~~	0 * 0011 =	0 0 0 0
2^2 자리 계산	4 * 0011 =	1 1 0 0
~~2^3 자리 계산~~	0 * 0011 =	0 0 0 0
		1 1 1 1 (15)

1 * 1100 (4배를 만드는 것이 왼쪽으로 2비트 이동한것과 같다.)

비트 연산자가 이동하면 *2

ex01.Test09

원을 이루는 값에서 서로 마주치는 숫자.

1	2	3	4	5	6
2	5	8	11	14	17

3	**4**	**5**	**6**	**7**	
8	**9**	**10**	**11**	**12**	

3+(n/2)

ex02.Test05

분수 비교 상대편 값을 곱해도 등호는 동일하다.

6 x 3 x $\frac{5}{6}$ > $\frac{2}{3}$ x 3 x 6

$\frac{3 * 5}{15}$ > $\frac{2 * 6}{12}$

가장 크다라고 가정

0	1	2
↓	↓	↓
5	2	5
6	3	4

6*2 > 3*5 크다면?

ex01, quiz11

*문자열에서 알파벳 개수 구하기, 중복 제거

"cabca"

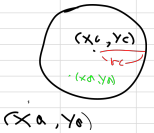
a b c d e f g h i - - - - -

일치하면 count 1 1 1 x x x x x - - - -

result

for(int i = 'a'; i < 'z'; i++) {
//파라미터로 주어진 문자열에서 문자를 반복해서 꺼낸다.
for(int x = 0; x <s.length(); x++) {
char c = s.charAt(x); // 문자열에서 인덱스 X의 문자를 꺼낸다.
if (c == i) { //알파벳의 코드값과 문자의 코드 값이 같다면
result++ ; //계수를 한개 증가 시킨다.
break;// 다음 항목은 검사할 필요가 없다. 왜? [중복 카운트]는 하지 않는다.

ex01, quiz12



반지름(rc)

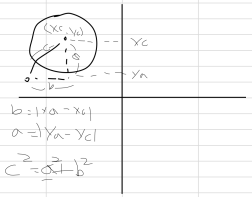
a = y

b = x

피타고라스의 정리

$c = \sqrt{a^2 + b^2}$

$c^2 = a^2 + b^2$




```
public cass Test11

public static void main(String[] args) {
System.out.println(differentSymbolsNaive("cabca") == 3);
}
문자 코드값 'a'
비교
a' 문자 i값 0부터
문자코드 증가    > 문자로 출력
```

```
package com.eomcs.quiz.ex01;

// 출처: codefights.com
//
// 소문자 알파벳의 문자열이 주어질 때, 서로 다른 알파벳의 개수는?
//
// 예) "cabca" ==> 3
//
/*
Given a string, find the number of different characters in it.
```

문자의 값을 꺼낼때는 char
x번째 문자를 꺼냄.

for "cabca" output should be 3

[input] string s

a string of lowercase latin letters
[output] integer
*/
//
// [시간 복잡도]
// - ?
//
public class Test11 {

static int differentSymbolsNaive(String s) {
int result = 0;

//알파벳 반복문을 돌린다.

//for(int i = 0; i < 26; i++) {
for(int i = 'a'; i <= 'z'; i++) {

//파라미터로 주어진 문자열에서 문자를 반복해서 꺼낸다.
for(int x = 0; x <s.length(); x++) {
char c = s.charAt(x); // 문자열에서 인덱스 X의 문자를 꺼낸다.
if (c == i) { //알파벳의 코드값과 문자의 코드 값이 같다면
result++ ; //개수를 한개 증가 시킨다.
break;// 다음 항목은 검사할 필요가 없다. 왜? [중복 카운트]는 하지 않는다.
}
}
}
//문자 코드값과 문자를 출력해 본다.
//System.out.printf("%1$%x == %1$c\n", 'a' + i);
//첫번째 값을 10진수로 ==> 두번째를 문자로

// 이 메서드를 완성하시오!
return result;
}
}

ex01, quiz14

123 -> 12 ->1 ->1
3 2 1
//나누기 10 //나누기 10
30+2
320+1
321
1의 자리
10의 자리
100의 자리

제곱으로 증가하는지 // 일정인지 /

ex01, quiz17

// 주어진 수 보다 큰 수 중에서 한 개 이상의 0으로 끝나는 가장 작은 수를 구하라

public static void main(String[] args) {
System.out.println(nearestRoundNumber(122) == 130);
System.out.println(nearestRoundNumber(99722) == 99730);
}
<==========>
static int nearestRoundNumber(int value) {
while (value % 10 != 0) { //value의 끝 자리가 0으로 끝날 때까지
value++; **//value 보다 큰 수를 찾는다.**
}
return value;
}
}

ex01, quiz18

주어진 수의 범위 안에서 특정 수의 배수 중에서 가장 큰 값을 구하라!

// 조건:

만약 찾지 못했다면 -1을 리턴하라!

```
//    maxDivisor(int left, int right, int divisor)
//    - left: 왼쪽 수
//    - right: 오른쪽 수. 오른쪽 수는 항상 왼쪽 수 보다 같거나 커야 한다.
//    - divisor: 나누는 수
// 예)
//    maxDivisor(2, 17, 4) => 16
```

```
public class Test18x {

public static void main(String[] args) {
System.out.println( maxDivisor(2, 17, 4));
}

static int maxDivisor(int left, int right, int divisor) {
for (int i = right; i >= left; i--) {
if(i % divisor == 0) {
return i;
}
}
return -1;
}
}
```

//큰 수를 찾는 것이기 때문에 반복문을 돌릴 때 큰 수에서 작은 수로 돌린다.

//주어진 수가 나누는 수의 배수인지 검사

---

ex01. quiz24

1 2 3 4 5

v v v v v

v 3^3

---

ex02. test11

c2 = a2 + b2

*주어진 배열에서 가장 큰 수를 찾는 것.

가장 큰 세변의 길이를 알아내야함.