Teoría de Números

Sofía Martin

Facultad de Informática

TTPS 2016

Contenido

Contenido

Aplicación

- Muchos ejercicios se resuelven aplicando el algoritmo correcto, correctamente.
- Sin embargo, algunos ejercicios se pueden resolver mediante “fuerza bruta”, que es simplemente probar todas las combinaciones “a lo bruto”.
- Otra forma interesante puede ser que, en vez de calcular algún valor, ya tenerlo precalculado en el código fuente.
  - Por ejemplo, si necesitamos los primeros 10 números primos, no hace falta usar una criba.
- Siempre estar atento al tamaño de la entrada
  - Si es suficientemente pequeña, puede que se resuelva por FB. Encontrar un método iterativo que nos permita resolver sistemas de ecuaciones lineales.
  - Revisar la librería cd C++ <cmath> y la librería Java.Util.Math/Java.Math

Aplicación

- Muchos ejercicios se resuelven aplicando el algoritmo correcto, correctamente.
- Sin embargo, algunos ejercicios se pueden resolver mediante “fuerza bruta”, que es simplemente probar todas las combinaciones “a lo bruto”.
- Otra forma interesante puede ser que, en vez de calcular algún valor, ya tenerlo precalculado en el código fuente.
  - Por ejemplo, si necesitamos los primeros 10 números primos, no hace falta usar una criba.
- Siempre estar atento al tamaño de la entrada
  - Si es suficientemente pequeña, puede que se resuelva por FB. Encontrar un método iterativo que nos permita resolver sistemas de ecuaciones lineales.
  - Revisar la librería cd C++ <cmath> y la librería Java.Util.Math/Java.Math

Aplicación

- Muchos ejercicios se resuelven aplicando el algoritmo correcto, correctamente.
- Sin embargo, algunos ejercicios se pueden resolver mediante “fuerza bruta”, que es simplemente probar todas las combinaciones “a lo bruto”.
- Otra forma interesante puede ser que, en vez de calcular algún valor, ya tenerlo precalculado en el código fuente.
  - Por ejemplo, si necesitamos los primeros 10 números primos, no hace falta usar una criba.
- Siempre estar atento al tamaño de la entrada
  - Si es suficientemente pequeña, puede que se resuelva por FB. Encontrar un método iterativo que nos permita resolver sistemas de ecuaciones lineales.
  - Revisar la librería cd C++ <cmath> y la librería Java.Util.Math/Java.Math

Aplicación

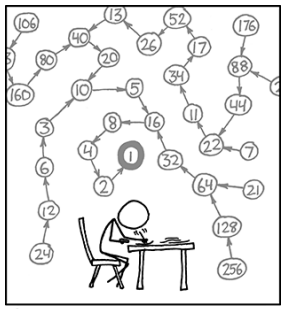
- Muchos ejercicios se resuelven aplicando el algoritmo correcto, correctamente.
- Sin embargo, algunos ejercicios se pueden resolver mediante “fuerza bruta”, que es simplemente probar todas las combinaciones “a lo bruto”.
- Otra forma interesante puede ser que, en vez de calcular algún valor, ya tenerlo precalculado en el código fuente.
  - Por ejemplo, si necesitamos los primeros 10 números primos, no hace falta usar una criba.
- Siempre estar atento al tamaño de la entrada
  - Si es suficientemente pequeña, puede que se resuelva por FB. Encontrar un método iterativo que nos permita resolver sistemas de ecuaciones lineales.
  - Revisar la librería cd C++ <cmath> y la librería Java.Util.Math/Java.Math

Ejemplo Conjetura de Collatz

$3 * n + 1$ - uva.onlinejudge.org/external/1/100.html

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

1. input n
2. print n
3. if$n = 1$ then STOP
4. if n is odd then $n \longleftarrow 3n + 1$
5. else $n \longleftarrow n/2$
6. GOTO 2



Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 . Cycle:16

Soluciones

For any two numbers i and j you are to determine the maximum cycle length over all numbers between i and j.

- Probar con cada entrada.
- Pre calcular...

Soluciones

For any two numbers i and j you are to determine the maximum cycle length over all numbers between i and j.

- Probar con cada entrada.
- Pre calcular...

Código - Probando cada entrada

```
while ( i <= j ) {
  unsigned int n = i;
  longciclo = 1;
  while ( n != 1 ) {
    if ( n % 2 == 1 ) n = 3 * n + 1;
    else n /= 2;
    longciclo++;
  }
  if ( longciclo > max_longciclo )
  max_longciclo = longciclo;
  i++;
}
cout << temp_i << " "<< temp_j <<" "<< max_longciclo << endl;
```

Solución más eficiente

¿Y si probamos con dinámica?

Enteros Grandes

- En general, un **int** es de 32 bits y un **long** de 64 bits. *En C++, **long long int** es de 64, "long" es 32.*
- Si queremos más bits, podemos intentar con un double, aunque que no nos ofrece mucha más precisión.
- En Java existen las clases llamadas "BigDecimal" y "BigInteger" que nos permiten trabajar con números de precisión arbitraria, decimales y enteros respectivamente.
- En C++ no hay nada "nativo", hay que hacerlo.

Enteros Grandes

- En general, un **int** es de 32 bits y un **long** de 64 bits. *En C++, **long long int** es de 64, "long" es 32.*
- Si queremos más bits, podemos intentar con un double, aunque que no nos ofrece mucha más precisión.
- En Java existen las clases llamadas "BigDecimal" y "BigInteger" que nos permiten trabajar con números de precisión arbitraria, decimales y enteros respectivamente.
- En C++ no hay nada "nativo", hay que hacerlo.

Enteros Grandes

- En general, un **int** es de 32 bits y un **long** de 64 bits. *En C++, **long long int** es de 64, "long" es 32.*
- Si queremos más bits, podemos intentar con un double, aunque que no nos ofrece mucha más precisión.
- En Java existen las clases llamadas "BigDecimal" y "BigInteger" que nos permiten trabajar con números de precisión arbitraria, decimales y enteros respectivamente.
- En C++ no hay nada "nativo", hay que hacerlo.

Enteros Grandes

- En general, un **int** es de 32 bits y un **long** de 64 bits. *En C++, **long long int** es de 64, "long" es 32.*
- Si queremos más bits, podemos intentar con un double, aunque que no nos ofrece mucha más precisión.
- En Java existen las clases llamadas "BigDecimal" y "BigInteger" que nos permiten trabajar con números de precisión arbitraria, decimales y enteros respectivamente.
- En C++ no hay nada "nativo", hay que hacerlo.

Enteros Grandes - Ejemplo

10424 - Love Calculator: ¿Cuánto amas a tu novia/o?
"...It's like this, a = 1, b = 2, c = 3, ..., z = 26. Both upper case and lower case holds the same values. Then make the sum of these numbers until it comes in one digit. [For example, consider a name 'bcz'. Here, b = 2, c = 3 and z = 26. So, the sum is (2+3+26) = 31 = (3+1) = 4 . Then the ratio of these two numbers in percentage will be the result. **Remember**: Result can not be more than 100% . Take the ratio carefully to avoid this problem"

```
int calc (string word) {
    int counter = 0;
    for (int i = 0; i < word.size(); i++) {
        word[i] = tolower(word[i]);
        if (word[i] >= 'a' && word[i] <= 'z') {
            counter += word[i] - 96;
            }
    }
    while (counter > 9) {
        int c = 0;
        while (counter != 0) {
            c += counter % 10;
            counter /= 10;
        }counter = c;
    }
 return counter;}
```

Enteros Grandes - Ejemplo

10424 - Love Calculator: ¿Cuánto amas a tu novia/o?
"...It's like this, a = 1, b = 2, c = 3, ..., z = 26. Both upper case and lower case holds the same values. Then make the sum of these numbers until it comes in one digit. [For example, consider a name 'bcz'. Here, b = 2, c = 3 and z = 26. So, the sum is (2+3+26) = 31 = (3+1) = 4 . Then the ratio of these two numbers in percentage will be the result. **Remember**: Result can not be more than 100% . Take the ratio carefully to avoid this problem"

```
int calc (string word) {
    int counter = 0;
    for (int i = 0; i < word.size(); i++) {
        word[i] = tolower(word[i]);
        if (word[i] >= 'a' && word[i] <= 'z') {
            counter += word[i] - 96;
            }
    }
    while (counter > 9) {
        int c = 0;
        while (counter != 0) {
            c += counter % 10;
            counter /= 10;
        }counter = c;
    }
 return counter;}
```

Enteros Grandes - Resolución

```
int main (void) {
  string primero, segundo;
  double porcentajeAmor;
  while (getline(cin, primero)) {
    getline(cin, segundo);
    int a = calc(primero), b = calc(segundo);
    if (a > b) {
        porcentajeAmor = 100.0 * b / a;
    } else {
        porcentajeAmor = 100.0 * a / b;
        }
        printf("%.2f %%\n", porcentajeAmor);
    }
    return 0;
}
```

Contenido

Goldbach's Conjecture

In 1742, Christian Goldbach, a German amateur mathematician, sent a letter to Leonhard Euler in which he made the following conjecture: Every number greater than 2 can be written as the sum of three prime numbers. **Goldbach** was considering 1 as a primer number, a convention that is no longer followed. Later on, Euler re-expressed the conjecture as: Every even number greater than or equal to 4 can be expressed as the sum of two prime numbers.

For example:

8 = 3 + 5. Both 3 and 5 are odd prime numbers.
20 = 3 + 17 = 7 + 13.
42 = 5 + 37 = 11 + 31 = 13 + 29 = 19 + 23.

Today it is still unproven whether the conjecture is right. (Oh wait, I have the proof of course, but it is too long to write it on the margin of this page.) Anyway, your task is now to verify Goldbach's conjecture as expressed by Euler for all even numbers less than a million.

¿Posibles soluciones...?

Veamos primero cómo encontrar los primos.

¿Posibles soluciones...?

Veamos primero cómo encontrar los primos.

Definición

Dado $p > 1$ es primo si 1 y p son sus únicos divisores. Dado $n \varepsilon Z$, podemos factorizarlo de manera única como:
$n = p_1^{e1} \dots p_k^{ek}$ **Factorización prima de n**
donde $p_1 \dots p_k$
son números primos conocidos como *factores* de n

- $105 = 3 * 5 * 7$
- $48 = 2 * 2 * 2 * 2 * 3 = 2^4 * 3$

Periocidad variable

- Hay **25** en [0..100]
- Hay **168** en [0..1000]
- Hay **1000** en [0..7919]

Algoritmos para encontrar números primos

¿Cómo averiguar si un número es primo? Un método consiste en dividir el número dado por todos los números menores o iguales a su raíz cuadrada, si ninguno de los cocientes es entero se puede deducir que el número es primo. ¿Por qué sólo hasta su raíz cuadrada?

Teorema

Si n > 1 no es primo (es un número compuesto) entonces n tiene un divisor primo $p \leq \sqrt{n}$.

Exlicación

Si N es divisible por d, entonces $N = d * \frac{N}{d}$. Si $\frac{N}{d}$es más chico que d , entonces $\frac{N}{d}$ o un un factor primo de $\frac{N}{d}$ debería haber dividido a N antes. Por lo tanto d y $\frac{N}{d}$ no pueden ser ambos mayores que$\sqrt{N}$.

Ahora, queremos encontrar todos los números primos hasta un valor n dado.

Contenido

Criba de Eratóstenes

Armamos una tabla con todos los números de [2,n] y los recorremos en orden hasta $\sqrt{n}$. Tomamos el 2 y tachamos todos los múltiplos de 2.
Luego buscamos el primero sin tachar y tachamos todos sus múltiplos.
Repetimos el proceso hasta llegar a $\sqrt{n}$ y los que quedaron sin tachar son los que no tienen divisores o sea los primos.
Sólo hasta $\sqrt{n}$, todos los múltiplos de los números menores a $\sqrt{n}$ ya habrán sido tachados.

(Un matemático marroquí del siglo XIII llamado Al-Marrakushi Ibn al-Banna demostró que basta con probar con los números inferiores o iguales a la ráiz cuadrada de n).

```
memset (isp, true, sizeof(p)) ;
  for (i=2; i<= int(sqrt(n)); i++)
    if(isp[i] )
      for(j= i*i ; j<=n; j+=i )
        isp[j] = false;
```

El tiempo de ejecución es *O(N log log N)*, y puede ser llevado a *O(N)* con algunas optimizaciones.

Criba de Eratóstenes

Armamos una tabla con todos los números de [2,n] y los recorremos en orden hasta $\sqrt{n}$. Tomamos el 2 y tachamos todos los múltiplos de 2. Luego buscamos el primero sin tachar y tachamos todos sus múltiplos. Repetimos el proceso hasta llegar a $\sqrt{n}$ y los que quedaron sin tachar son los que no tienen divisores o sea los primos.

Sólo hasta $\sqrt{n}$, todos los múltiplos de los números menores a $\sqrt{n}$ ya habrán sido tachados.

(Un matemático marroquí del siglo XIII llamado Al-Marrakushi Ibn al-Banna demostró que basta con probar con los números inferiores o iguales a la ráiz cuadrada de n).

```
memset (isp, true, sizeof(p)) ;
  for (i=2; i<= int(sqrt(n)); i++)
    if(isp[i] )
      for(j= i*i ; j<=n; j+=i )
        isp[j] = false;
```

El tiempo de ejecución es *O(N log log N)*, y puede ser llevado a *O(N)* con algunas optimizaciones.

Criba de Eratóstenes

Armamos una tabla con todos los números de [2,n] y los recorremos en orden hasta $\sqrt{n}$. Tomamos el 2 y tachamos todos los múltiplos de 2.
Luego buscamos el primero sin tachar y tachamos todos sus múltiplos.
Repetimos el proceso hasta llegar a $\sqrt{n}$ y los que quedaron sin tachar son los que no tienen divisores o sea los primos.
Sólo hasta $\sqrt{n}$, todos los múltiplos de los números menores a $\sqrt{n}$ ya habrán sido tachados.

(Un matemático marroquí del siglo XIII llamado Al-Marrakushi Ibn al-Banna demostró que basta con probar con los números inferiores o iguales a la ráiz cuadrada de n).

```
memset (isp, true, sizeof(p)) ;
  for (i=2; i<= int(sqrt(n)); i++)
    if(isp[i] )
      for(j= i*i ; j<=n; j+=i )
        isp[j] = false;
```

El tiempo de ejecución es *O(N log log N)*, y puede ser llevado a *O(N)* con algunas optimizaciones.

Criba de Eratóstenes

Armamos una tabla con todos los números de [2,n] y los recorremos en orden hasta $\sqrt{n}$. Tomamos el 2 y tachamos todos los múltiplos de 2.
Luego buscamos el primero sin tachar y tachamos todos sus múltiplos.
Repetimos el proceso hasta llegar a $\sqrt{n}$ y los que quedaron sin tachar son los que no tienen divisores o sea los primos.
Sólo hasta $\sqrt{n}$, todos los múltiplos de los números menores a $\sqrt{n}$ ya habrán sido tachados.

(Un matemático marroquí del siglo XIII llamado Al-Marrakushi Ibn al-Banna demostró que basta con probar con los números inferiores o iguales a la ráiz cuadrada de n).

```
memset (isp, true, sizeof(p)) ;
  for (i=2; i<= int(sqrt(n)); i++)
    if(isp[i] )
      for(j= i*i ; j<=n; j+=i )
        isp[j] = false;
```

El tiempo de ejecución es *O(N log log N)*, y puede ser llevado a *O(N)* con algunas optimizaciones.

Criba de Eratóstenes

Armamos una tabla con todos los números de [2,n] y los recorremos en orden hasta $\sqrt{n}$. Tomamos el 2 y tachamos todos los múltiplos de 2.
Luego buscamos el primero sin tachar y tachamos todos sus múltiplos.
Repetimos el proceso hasta llegar a $\sqrt{n}$ y los que quedaron sin tachar son los que no tienen divisores o sea los primos.
Sólo hasta $\sqrt{n}$, todos los múltiplos de los números menores a $\sqrt{n}$ ya habrán sido tachados.

(Un matemático marroquí del siglo XIII llamado Al-Marrakushi Ibn al-Banna demostró que basta con probar con los números inferiores o iguales a la ráiz cuadrada de n).

```
memset (isp, true, sizeof(p)) ;
  for (i=2; i<= int(sqrt(n)); i++)
    if(isp[i] )
      for(j= i*i ; j<=n; j+=i )
        isp[j] = false;
```

El tiempo de ejecución es *O(N log log N)*, y puede ser llevado a *O(N)* con algunas optimizaciones.

Criba de Eratóstenes

Armamos una tabla con todos los números de [2,n] y los recorremos en orden hasta $\sqrt{n}$. Tomamos el 2 y tachamos todos los múltiplos de 2.
Luego buscamos el primero sin tachar y tachamos todos sus múltiplos.
Repetimos el proceso hasta llegar a $\sqrt{n}$ y los que quedaron sin tachar son los que no tienen divisores o sea los primos.
Sólo hasta $\sqrt{n}$, todos los múltiplos de los números menores a $\sqrt{n}$ ya habrán sido tachados.

(Un matemático marroquí del siglo XIII llamado Al-Marrakushi Ibn al-Banna demostró que basta con probar con los números inferiores o iguales a la ráiz cuadrada de n).

```
memset (isp, true, sizeof(p)) ;
  for (i=2; i<= int(sqrt(n)); i++)
    if(isp[i] )
      for(j= i*i ; j<=n; j+=i )
        isp[j] = false;
```

El tiempo de ejecución es *O(N log log N)*, y puede ser llevado a *O(N)* con algunas optimizaciones.

Solución Goldbach's Conjecture

Ya sabiendo cómo calcular los primos, ¿cómo podemos resolver el problema?

Recordar el problema:

8 = 3 + 5. Both 3 and 5 are odd prime numbers.
20 = 3 + 17 = 7 + 13.
42 = 5 + 37 = 11 + 31 = 13 + 29 = 19 + 23.

Solución Goldbach's Conjecture

```
main() {
    Criba();
    int n;
    while (cin >> n && n) {
        bool f = true;
        for (int nn=n/2, i=0; f && primo[i]<=nn; i++) {
            if (esPrimo[n-prime[i]]) {
                printf("%d = %d + %d", n, primo[i], n-primo[i]);
                f = false;
            }       }
        if (f) cout << "Goldbach's conjecture is wrong.";
        cout << endl;
    }}
```

Factorización con la Criba

La criba puede guardar más información (Almacena en la tabla para cada número un divisor primo, no solo si es primo o no: por ejemplo p[6] =3, p[45]=5) Y entonces

```
memset(p ,-1, sizeof(p)) ;
for(i=4;i<n; i+=2)p[i]=2 ;
for(i=3;i<=int(sqrt(n));i
    +=2)
  if(p[i]== -1)
    for(j=i*i;j<=n;j+=2*i)
      p[j] = i ;
```

```
int fact(int n,int f[]){
 int F= 0 ;
 while(p[n]!= -1) {
  f[F++] = p[n] ;
  n /= p[n] ;
 }
 f[F++] = n ;
 return F ;
}
```

Ej: $p[45] = 5 \Rightarrow 45/5 = 9 \Rightarrow p[9] = 3 \Rightarrow 9/3 = 3 \Rightarrow p[3] = -1(esprimo) \Rightarrow f = (5, 3, 3)$

Factorización con la Criba

La criba puede guardar más información (Almacena en la tabla para cada número un divisor primo, no solo si es primo o no: por ejemplo p[6] =3, p[45]=5) Y entonces

```
memset(p ,-1, sizeof(p)) ;
for(i=4;i<n; i+=2)p[i]=2 ;
for(i=3;i<=int(sqrt(n));i
    +=2)
  if(p[i]== -1)
    for(j=i*i;j<=n;j+=2*i)
      p[j] = i ;
```

```
int fact(int n,int f[]){
 int F= 0 ;
 while(p[n]!= -1) {
  f[F++] = p[n] ;
  n /= p[n] ;
 }
 f[F++] = n ;
 return F ;
}
```

Ej: $p[45] = 5 \Rightarrow 45/5 = 9 \Rightarrow p[9] = 3 \Rightarrow 9/3 = 3 \Rightarrow p[3] = -1(esprimo) \Rightarrow f = (5, 3, 3)$

Funciones útiles

Si además de usar la criba para saber si un número es primo o no, guardo un arreglo con todos los primos puedo utilizar este arreglo, **primos** para calcular:

- Contar el número de factores primos de *N*.

```
long long numPF(long long N){
  long long pf_ind=0, pf=primos[pf_ind], resp=0;
  while(pf * pf <= N){
    while (N % pf == 0){N /= pf; resp++;}
    pf = primos{++pf_ind};
  }
  if(N != 1) resp++;
  return resp;
}
```

Divisores

- Contar el número de divisores de N. Un divisor de N está definido como un entero que divide N sin resto. Si un número $N = a^i * b^j * ... * c^k$, entonces N tiene $(i+1) * (j+1) * ... * (k+1)$ divisores. Por ejemplo $N = 84 = 2^2 * 3^1 * 7^1$ tiene $(2+1) * (1+1) * (1+1) = 3 * 2 * 2 = 12$ divisores. Los 12 divisores son:{1, 2, 3, 4, 6, 7, 12, 14, 21, 28, 42, 84}

```
long long numPF(long long N){
  long long pf_ind=0, pf=primos[pf_ind], resp=1;
  while(pf * pf <= N){
    long long potencia = 0;
    while (N % pf == 0){N /= pf; potencia++;}
    resp *= (potencia+1);
    pf = primos{++pf_ind};
  }
  if(N != 1) resp *=2;
  return resp;
}
```

Divisores

Siguiendo con los divisores de un número:
$N = 84 = 2^2 * 3^1 * 7^1$
Tomando un número **12**, por ejemplo, cómo podemos saber si lo divide a través de sus divisores?

$N = 12 = 2^2 * 3^1$

Y si tomamos el número **36**, por ejemplo, cómo podemos saber si lo divide?
$N = 36 = 2^2 * 3^2$

Divisores

Siguiendo con los divisores de un número:
$N = 84 = 2^2 * 3^1 * 7^1$
Tomando un número **12**, por ejemplo, cómo podemos saber si lo divide a través de sus divisores?

$N = 12 = 2^2 * 3^1$

Y si tomamos el número **36**, por ejemplo, cómo podemos saber si lo divide?
$N = 36 = 2^2 * 3^2$

Divisores

Siguiendo con los divisores de un número:
$N = 84 = 2^2 * 3^1 * 7^1$
Tomando un número **12**, por ejemplo, cómo podemos saber si lo divide a través de sus divisores?

$N = 12 = 2^2 * 3^1$

Y si tomamos el número **36**, por ejemplo, cómo podemos saber si lo divide?
$N = 36 = 2^2 * 3^2$

Contenido

10407 – Simple division - El problema

Integer division between a dividend n and a divisor d yields a quotient q and a remainder r. q is the integer which maximizes $q * d$ such that $q * d \leq n$ and $r = n - q * d$. For any set of integers there is an integer d such that each of the given integers when divided by d leaves the same remainder.
Dado una lista A de enteros con longitud n, encontrar un número d tal que para todos los enteros de la lista el resto sea el mismo, o sea,

```
A[0]=ad + r
A[1]=bd + r
A[2]=cd + r
etc..
```

Entonces se va a cumplir para cada par i, j, que A[i]-A[j] es múltiplo de d. Por lo tanto podemos decir que el número buscado d, es:

```
d = GCD(A[i]-A[j]) for all i,j < n
```

GCD - Máximo Común Divisor

El máximo común divisor entre dos enteros *a* y *b* es el mayor entero *d* que los divide a ambos sin dejar resto.
Ej: gcd(42,56) = 14 => 42/14=3 y 56/14=4.

Propiedades

- gcd(0, 0) = 0
- gcd(a, b) = gcd(b, a).
- gcd(a, b) = gcd(|a|, |b|).
- $0 < gcd(a, b) <= min|a|, |b|$. si a y b son distintos de 0.
- Si $g = gcd(a, b) \Rightarrow gcd(a/g, b/g) = 1$.
- Si $a > 0 \Rightarrow gcd(a, 0) = gcd(a, a) = a$.
- Dado $a > b > 0$. Si $a = bq + r$, $\Rightarrow gcd(a, b) = gcd(b, r)$

Algoritmo de Euclides

Observamos que: a = q.b + r => gcd (a, b) = gcd (b, r) Ejemplo: Calcular gcd(803, 154).

- gcd(803, 154) = gcd(154, 33) ya que 803 = 154 * 5 + 33
- gcd(154, 33) = gcd(33, 22) ya que 154 = 33 * 4 + 22
- gcd(33, 22) = gcd(22, 11) ya que 33 = 22 * 1 + 11
- gcd(22, 11) = gcd(11, 0) ya que 22 = 11 * 2 + 0
- gcd(11, 0) = 11

Resultado

gcd(803, 154) = 11.

```
int gcd (int a, int b){
  if(b==0) return a;
  return gcd(b, a%b)}
```

Algoritmo de Euclides

Observamos que: a = q.b + r => gcd (a, b) = gcd (b, r) Ejemplo: Calcular gcd(803, 154).

- gcd(803, 154) = gcd(154, 33) ya que 803 = 154 * 5 + 33
- gcd(154, 33) = gcd(33, 22) ya que 154 = 33 * 4 + 22
- gcd(33, 22) = gcd(22, 11) ya que 33 = 22 * 1 + 11
- gcd(22, 11) = gcd(11, 0) ya que 22 = 11 * 2 + 0
- gcd(11, 0) = 11

Resultado

gcd(803, 154) = 11.

```
int gcd (int a, int b){
  if(b==0) return a;
  return gcd(b, a%b)}
```

Algoritmo de Euclides

Observamos que: a = q.b + r => gcd (a, b) = gcd (b, r) Ejemplo: Calcular gcd(803, 154).

- gcd(803, 154) = gcd(154, 33) ya que 803 = 154 * 5 + 33
- gcd(154, 33) = gcd(33, 22) ya que 154 = 33 * 4 + 22
- gcd(33, 22) = gcd(22, 11) ya que 33 = 22 * 1 + 11
- gcd(22, 11) = gcd(11, 0) ya que 22 = 11 * 2 + 0
- gcd(11, 0) = 11

Resultado

gcd(803, 154) = 11.

```
int gcd (int a, int b){
  if(b==0) return a;
  return gcd(b, a%b)}
```

Algoritmo de Euclides

Observamos que: a = q.b + r => gcd (a, b) = gcd (b, r) Ejemplo: Calcular gcd(803, 154).

- gcd(803, 154) = gcd(154, 33) ya que 803 = 154 * 5 + 33
- gcd(154, 33) = gcd(33, 22) ya que 154 = 33 * 4 + 22
- gcd(33, 22) = gcd(22, 11) ya que 33 = 22 * 1 + 11
- gcd(22, 11) = gcd(11, 0) ya que 22 = 11 * 2 + 0
- gcd(11, 0) = 11

Resultado

gcd(803, 154) = 11.

```
int gcd (int a, int b){
  if(b==0) return a;
  return gcd(b, a%b)}
```

Algoritmo de Euclides

Observamos que: a = q.b + r => gcd (a, b) = gcd (b, r) Ejemplo: Calcular gcd(803, 154).

- gcd(803, 154) = gcd(154, 33) ya que 803 = 154 * 5 + 33
- gcd(154, 33) = gcd(33, 22) ya que 154 = 33 * 4 + 22
- gcd(33, 22) = gcd(22, 11) ya que 33 = 22 * 1 + 11
- gcd(22, 11) = gcd(11, 0) ya que 22 = 11 * 2 + 0
- gcd(11, 0) = 11

Resultado

gcd(803, 154) = 11.

```
int gcd (int a, int b){
  if(b==0) return a;
  return gcd(b, a%b)}
```

Algoritmo de Euclides

Observamos que: a = q.b + r => gcd (a, b) = gcd (b, r) Ejemplo: Calcular gcd(803, 154).

- gcd(803, 154) = gcd(154, 33) ya que 803 = 154 * 5 + 33
- gcd(154, 33) = gcd(33, 22) ya que 154 = 33 * 4 + 22
- gcd(33, 22) = gcd(22, 11) ya que 33 = 22 * 1 + 11
- gcd(22, 11) = gcd(11, 0) ya que 22 = 11 * 2 + 0
- gcd(11, 0) = 11

Resultado

gcd(803, 154) = 11.

```
int gcd (int a, int b){
  if(b==0) return a;
  return gcd(b, a%b)}
```

Iterativo

```
int gcd_iter(int u, int v) {
  int t;
  while (v) {
    t = u;
    u = v;
    v = t % v;
  }
  return u < 0 ? -u : u; /* abs(u) */
}
```

Solución - Simple division

Veamos cómo armar los números en combinación del primero. Definimos:

$$A_1 = d * q_1 + r$$
$$A_2 = d * q_2 + r$$
$$A_3 = d * q_3 + r$$
$$.....$$

Definimos las diferencias de cada número con el primero:

$$A_2 - A_1 = (d * q_2 + r) - (d * q_1 + r) = d * (q_2 - q_1)$$
$$A_3 - A_1 = (d * q_3 + r) - (d * q_1 + r) = d * (q_3 - q_1)$$
$$.....$$

Nos queda definido :

$$R_2 = A_2 - A_1 = d * (q_2 - q_1)$$
$$R_3 = A_3 - A_1 = d * (q_3 - q_1)$$
$$....$$

Prestar atención a que los R_2, R_3.. tienen como divisor d, **con resto 0.** El GCD($R_2, R_3, ..$) por definición es d.

Solución - Simple division

Veamos cómo armar los números en combinación del primero. Definimos:

$$A_1 = d * q_1 + r$$
$$A_2 = d * q_2 + r$$
$$A_3 = d * q_3 + r$$
$$\ldots\ldots$$

Definimos las diferencias de cada número con el primero:

$$A_2 - A_1 = (d * q_2 + r) - (d * q_1 + r) = d * (q_2 - q_1)$$
$$A_3 - A_1 = (d * q_3 + r) - (d * q_1 + r) = d * (q_3 - q_1)$$
$$\ldots\ldots$$

Nos queda definido :

$$R_2 = A_2 - A_1 = d * (q_2 - q_1)$$
$$R_3 = A_3 - A_1 = d * (q_3 - q_1)$$
$$\ldots\ldots$$

Prestar atención a que los R_2, R_3.. tienen como divisor d, **con resto 0.** El GCD($R_2, R_3, ..$) por definición es d.

Solución - Simple division

Veamos cómo armar los números en combinación del primero.
Definimos:

$$A_1 = d * q_1 + r$$
$$A_2 = d * q_2 + r$$
$$A_3 = d * q_3 + r$$
$$\ldots\ldots$$

Definimos las diferencias de cada número con el primero:

$$A_2 - A_1 = (d * q_2 + r) - (d * q_1 + r) = d * (q_2 - q_1)$$
$$A_3 - A_1 = (d * q_3 + r) - (d * q_1 + r) = d * (q_3 - q_1)$$
$$\ldots\ldots$$

Nos queda definido :

$$R_2 = A_2 - A_1 = d * (q_2 - q_1)$$
$$R_3 = A_3 - A_1 = d * (q_3 - q_1)$$
$$\ldots\ldots$$

Prestar atención a que los R_2, R_3.. tienen como divisor d, **con resto 0.** El GCD($R_2, R_3, ..$) por definición es d.

Solución - Simple division

Veamos cómo armar los números en combinación del primero. Definimos:

$$A_1 = d * q_1 + r$$
$$A_2 = d * q_2 + r$$
$$A_3 = d * q_3 + r$$
$$\ldots\ldots$$

Definimos las diferencias de cada número con el primero:

$$A_2 - A_1 = (d * q_2 + r) - (d * q_1 + r) = d * (q_2 - q_1)$$
$$A_3 - A_1 = (d * q_3 + r) - (d * q_1 + r) = d * (q_3 - q_1)$$
$$\ldots\ldots$$

Nos queda definido :

$$R_2 = A_2 - A_1 = d * (q_2 - q_1)$$
$$R_3 = A_3 - A_1 = d * (q_3 - q_1)$$
$$\ldots\ldots$$

Prestar atención a que los R_2, R_3.. tienen como divisor d, **con resto 0.** El GCD($R_2, R_3, ..$) por definición es d.

Solución - Simple division

Tomando los valores del problema Se puede calcular el valor de d encontrando el GCD de:

```
GCD (1059−701, 1417−701, 2312−701) =
GCD (358, 716, 1611) = 179
```

El valor d = 179 divide al conjunto de enteros 701 1059 1417 2312 , con el mismo resto r = 164. Dado GCD (179) divide en forma exacta a 358, 716, 1611 ..., entonces los números originales 701 1059 1417 2312 tendrán como resto en forma exacta a $701 mod 179 = 164$. Teniendo en cuenta que el orden de los números dados es incremental, estamos seguros que el primero es el mas chico, y de esta forma obtendremos el *d* más grande.

Contenido

LCM - Mínimo Común Múltiplo

El mínimo común múltiplo entre dos enteros a y b es el menor entero m que es múltiplo de ambos.

Observamos que:

$lcm(a, b) = a.b/gcd(a, b)$
Ej: $lcm(21, 6) = 21.6/gcd(21, 6) = 21.6/3 = 126/3| = 42$ También se cumple: $gcd(a, b) = a.b/lcm(a, b)$

Ejemplos

Problema 1

Encontrar el tiempo de sincronización entre 2 luces de tráfico. Si la luz de tráfico A muestra una luz verde cada 3 minutos y la luz de tráfico B muestra un color verde cada 2 minutos.

Respuesta: cada 6 minutos ambas mostrarán color verde a la vez, dado que $lcm(2, 3) = 6$

Problema 2

Cuál será el próximo año (después de 2000) que una elección presidencial, la cuál sucede cada 4 años coincidirá con un censo estadístico el cuál sucede cada 10 años?

Respuesta: este evento sucederá cada 20 años, dado que $lcm(4, 10) = 20$

Ejemplos

Problema 1

Encontrar el tiempo de sincronización entre 2 luces de tráfico. Si la luz de tráfico A muestra una luz verde cada 3 minutos y la luz de tráfico B muestra un color verde cada 2 minutos.
Respuesta: cada 6 minutos ambas mostrarán color verde a la vez, dado que $lcm(2, 3) = 6$

Problema 2

Cuál será el próximo año (después de 2000) que una elección presidencial, la cuál sucede cada 4 años coincidirá con un censo estadístico el cuál sucede cada 10 años?
Respuesta: este evento sucederá cada 20 años, dado que $lcm(4, 10) = 20$

Ejemplos

Problema 1

Encontrar el tiempo de sincronización entre 2 luces de tráfico. Si la luz de tráfico A muestra una luz verde cada 3 minutos y la luz de tráfico B muestra un color verde cada 2 minutos.
Respuesta: cada 6 minutos ambas mostrarán color verde a la vez, dado que $lcm(2, 3) = 6$

Problema 2

Cuál será el próximo año (después de 2000) que una elección presidencial, la cuál sucede cada 4 años coincidirá con un censo estadístico el cuál sucede cada 10 años?
Respuesta: este evento sucederá cada 20 años, dado que $lcm(4, 10) = 20$

Ejemplos

Problema 1

Encontrar el tiempo de sincronización entre 2 luces de tráfico. Si la luz de tráfico A muestra una luz verde cada 3 minutos y la luz de tráfico B muestra un color verde cada 2 minutos.
Respuesta: cada 6 minutos ambas mostrarán color verde a la vez, dado que $lcm(2, 3) = 6$

Problema 2

Cuál será el próximo año (después de 2000) que una elección presidencial, la cuál sucede cada 4 años coincidirá con un censo estadístico el cuál sucede cada 10 años?
Respuesta: este evento sucederá cada 20 años, dado que $lcm(4, 10) = 20$

Problema 11388 - GCD LCM

Dados dos números : el **gcd - G** y el **lcm - L**, encontrar los valores *a* y *b* que cuyos gcd y lcm sean los valores dados como input .
OJO: In case there is more than one pair satisfying the condition, output the pair for which *a* is minimized.
Sabemos que $gcd(a, b) = G$.
Entonces G divide *a* y *b*. Y *a* es múltiplo de *G*. Pero debemos asegurarnos que *a* sea el múltiplo más chico posible y cuál es el número más chico que divide a G.
$\therefore a = G$

Problema 11388 - GCD LCM

Dados dos números : el **gcd - G** y el **lcm - L**, encontrar los valores a y b que cuyos gcd y lcm sean los valores dados como input .
OJO: In case there is more than one pair satisfying the condition, output the pair for which a is minimized.
Sabemos que $gcd(a, b) = G$.
Entonces G divide a y b. Y a es múltiplo de G. Pero debemos asegurarnos que a sea el múltiplo más chico posible y cuál es el número más chico que divide a G.
$\therefore a = G$

Problema 11388 - GCD LCM

Dados dos números : el **gcd - G** y el **lcm - L**, encontrar los valores a y b que cuyos gcd y lcm sean los valores dados como input .
OJO: In case there is more than one pair satisfying the condition, output the pair for which a is minimized.
Sabemos que $gcd(a, b) = G$.
Entonces G divide a y b. Y a es múltiplo de G. Pero debemos asegurarnos que a sea el múltiplo más chico posible y cuál es el número más chico que divide a G.
$\therefore a = G$

Problema 11388 - GCD LCM

Sabemos: $gcd(a, b) \times lcm(a, b) = a \times b$ **Pensemos un ejemplo:**

a=12
b=18
gcd(12,18)=6
lcm(12,18)=36
12 x 18 = 6 x 36

Pero debe ser el más chico:
Si tomamos entonces:

a=6
b=36
gcd(6,36)= 6
lcm(6,36)= 36