Módulo

Entrada:

- Inteiros `a` (podendo ser negativo!) e `n` > 0.

Saída:

- Valor de `a` módulo `n`.

Complexidade: `O( 1 )`

Global:

```
int mod(int a, int n) {
  return (a%n + n)%n;
}
```

Exponenciação Modular Rápida

Entrada:

- Inteiros `a`, `b` e `n`.

Saída:

- Valor de `a^b mod n`.

Complexidade: `O( log(b) )`

Global:

```
int expmod(int a, int b, int n) {
  if(b == 0)
    return 1;
  else {
    long long res = expmod(a, b/2, n);
    res = (res*res) % n;
    if(b%2 == 1)
      res = (res*a) % n;
    return (int) res;
  }
}
```

Máximo Divisor Comum

Utiliza o algoritmo de Euclides.

Entrada:

- Inteiros `a` e `b`.

Saída:

- Maior inteiro que divide `a` e `b`.

Complexidade: `O( log(a) + log(b) )`

Global:

```
int mdc(int a, int b) {
  if(a<0) a = -a;
  if(b<0) b = -b;

  if(b == 0)
    return a;
  else
    return mdc(b, a%b);
}
```

Máximo Divisor Comum Estendido

Utiliza o algoritmo de Euclides estendido.

Entrada:

- Inteiros positivos `a` e `b`.

Saída:

- Maior inteiro que divide `a` e `b` como retorno.
- Variáveis inteiras `x` e `y` tais que `a.x + b.y = mdc(a,b)`.

Complexidade: `O( log(a) + log(b) )`

Global:

```
int mdc(int a, int b, int *x, int *y) {
  int xx, yy, d;
  if(b==0) {
    *x=1; *y=0;
    return a;
  }

  d = mdc(b, a%b, &xx, &yy);
  *x = yy;
  *y = xx - a/b*yy;
  return d;
}
(1)    120/23 = 5 resta 5
(2)      23/5 = 4 resta 3
(3)       5/3 = 1 resta 2
(4)       3/2 = 1 resta 1
(5)       2/1 = 2 resta 0

(1)     5 = 1*120 - 5*23
(2)     3 = 1*23 - 4*5   Substituindo o 5 temos
        3 = 1*23 - 4*(1*120 - 5*23)
        3 = -4*120 + 21*23
(3)     2 = 1*5 - 1*3    Substituindo o valor de 5 e 3 temos
        2 = 1(1*120 - 5*23) - 1(-4*120 + 21*23)
        2 = 5*120 - 26*23
(4)     1 = 1*3 - 1*2   Novamente substituindo 3 e 2
        1 = 1(-4*120 + 21*23) - 1(5*120 - 26*23)
        1 = -9*120 + 47*23
```

portanto, **x = -9** e **y = 47** e temos:

`MDC(120,23)` `= 120 * ( – 9) + 47 * 23`

Teste de Primalidade

Entrada:

- Inteiro `n`.

Saída:

- Valor booleano que indica se `n` é primo ou não.

Complexidade: `O( sqrt(n) )`

Global:

```
int ehPrimo(int n) {
  if(n==2) return 1;
  if(n<=1 || n%2 == 0) return 0;

  for(int i=3; i*i<=n; i+=2)
    if(n%i == 0)
      return 0;

  return 1;
}
```

Crivo de Eratóstenes

Entrada:

- Inteiro `n` <= `MAXN`.

Saída:

- Vetor `ehprimo[]` que indica se os números menores ou iguais a `n` são primos ou não.

Complexidade: `O( n.log(n) )`

Global:

```
int ehprimo[MAXN+1];

void achaPrimos(int n) {
  ehprimo[0] = ehprimo[1] = 0;
  ehprimo[2] = 1;

  for(int i=3; i<=n; i++)
    ehprimo[i] = i%2;

  for(int i=3; i*i<=n; i+=2)
    if(ehprimo[i])
      for(int j=i*i; j<=n; j+=i)
        ehprimo[j] = 0;
}
```

Função 'phi' de Euler

Entrada:

- Inteiro `n` > 0.

Saída:

- Valor ϕ(`n`), onde ϕ é a função *phi* de Euler, equivalente ao número de primos relativos a `n` (mdc=1) menores o iguais a `n`.

Complexidade: `O( n )`

Dependências:

- Crivo de Eratóstenes: `ehprimo[i]`, para `i <= n/2`.

Global:

```
int phi(int n) {
  int res = n;
  for(int p=2; 2*p<=n; p++)
    if(ehprimo[p] && n%p==0)
      res = res/p*(p-1);
  return res;
}
```

Resolvedor de Equação Modular Linear

Entrada:

- Inteiro `a` e `b` quaiquer e `n` positivo.

Saída:

- Menor valor positivo `x` tal que `a.x == b (mod n)` ou `-1` se não houver solução.

Complexidade: `O( n )`

Dependências:

- Módulo: `mod(int a, int n)`.
- MDC Estendido: `mdc(int a,int b,int *x, int*y)`.

Global:

```
int solve(int a, int b, int n) {
  int d, x, y;

  a = mod(a, n);
  b = mod(b, n);
```

```
  d = mdc(a, n, &x, &y);
  if(b%d==0)
    return mod( ((long long)x%(n/d)) * ((b/d)%(n/d)) , n/d );
  else
    return -1;
}
```