Teoria dos Números - Parte 1

SCC5900 - Projeto de Algoritmos

João Batista

Introdução

- Teoria dos Números é um das mais bonitas e interessantes áreas da matemática.
- É o ramo da matemática que se preocupa com as propriedades dos números inteiros:
  - Envolve muitos problemas práticos que são facilmente compreendidos mesmo por não-matemáticos;
  - Muitos desses problemas podem ser encontrados em situações práticas, científicas e problemas de competição.
- Existe uma coleção de algoritmos interessantes relacionados que solucionam problemas de forma inteligente e eficiente.

Números Primos

- Um **número primo** é um inteiro $p > 1$ que somente é divisível por 1 e por ele mesmo:
  - Se p é um número primo, então $p = a \times b$ para inteiros $a \leq b$ implica que $a = 1$ e $b = p$;
  - Os 10 primeiros primos são: 2, 3, 5, 7, 11, 13, 17, 19, 23 e 27.
  - Alguns primos grandes são: 104729, 1299709, 15485863, 179424673, 2147483647, etc.
- Qualquer número não-primo é chamado de **composto**.

Encontrando Primos

- A forma mais simples de testar se um número *n* é primo e por divisões sucessivas:
  - Comece pelo menor divisor candidato e tente todos os possíveis divisores maiores;
  - Como 2 é o único primo par (porque?), uma vez que for verificado que *n* não é par, pode-se verificar somente números ímpares como possíveis fatores;
  - Pode-se considerar *n* como primo tão logo não se consiga encontrar fatores primos menores que $\sqrt{n}$ (por que?).
  - Por fim, pode-se testar se *n* é divisível somente por divisores primos (O($\sqrt{n}$ / ln($\sqrt{n}$)).

Crivo de Eratóstenes

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Primzahlen:

Encontrando Primos com Crivo

```cpp
typedef long long ll;
typedef vector<int> vi;
typedef map<int, int> mii;

ll _sieve_size;
bitset<10000010> bs;   // 10^7 should be enough for most cases
vi primes;   // compact list of primes in form of vector<int>

void sieve(ll upperbound) {          // create list of primes in [0..upperbound]
  _sieve_size = upperbound + 1;                   // add 1 to include upperbound
  bs.set();                                                 // set all bits to 1
  bs[0] = bs[1] = 0;                                     // except index 0 and 1
  for (ll i = 2; i <= _sieve_size; i++) if (bs[i]) {
    // cross out multiples of i starting from i * i!
    for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
    primes.push_back((int)i);  // also add this vector containing list of primes
} }                                              // call this method in main method

bool isPrimeSieve(ll N) {            // a good enough deterministic prime tester
  if (N <= _sieve_size) return bs[N];                     // O(1) for small primes
  for (int i = 0; i < (int)primes.size() && (ll) primes[i]*primes[i] <= N ; i++)
    if (N % primes[i] == 0) return false;
  return true;                    // it takes longer time if N is a large prime!
}                      // note: only work for N <= (last prime in vi "primes")^2
```

Encontrando Primos sem Crivo

- A complexidade do algoritmo que encontra o crivo é:
  - $O((n \log n)(\log \log n))$ para tempo;
  - $O(n)$ para memória
- Caso essas complexidades sejam um problema, pode-se encontrar um primo sem a ajuda do crivo

```
bool isPrime(ll N) {   // a good enough deterministic prime tester without sieve
  if (N > 2 && N % 2 == 0) return false;                    // the only even prime is 2
  for (ll i = 3; i * i <= N; i += 2)    //it tests all odd numbers up to sqrt(N)
    if (N % i == 0) return false;
  return true;
}
```

Teorema Fundamental da Aritmética

- Números primos são importantes por causa do **teorema fundamental da aritmética**.
  - Qualquer número inteiro *n* pode ser expresso somente de uma forma como um produto de primos.
    - Por exemplo, 105 é unicamente expresso como 3 * 5 * 7, e 32 como 2 * 2 * 2 * 2 * 2.
- Esse coleção de números que quando multiplicada resulta em *n* é chamada de **fatoração em primos** de *n*.

Fatoração em Primos

- A ordem não importa em uma fatoração em primos:
  - Pode-se canonicamente listar os números em ordem crescente;
  - Mas a multiplicidade de cada primo importa, e o que diferencia a fatoração de 4 e 8.
- Um número primo p é um **fator** de x se ele ocorre na fatoração em primos de x.

Fatoração em Primos

```
vi primeFactors(ll N) {   // remember: vi is vector of integers, ll is long long
  vi factors;                      // vi `primes' (generated by sieve) is optional
  ll PF_idx = 0, PF = primes[PF_idx];     // using PF = 2, 3, 4, ..., is also ok
  while (N != 1 && (PF * PF <= N)) {   // stop at sqrt(N), but N can get smaller
    while (N % PF == 0) { N /= PF; factors.push_back(PF); }    // remove this PF
    PF = primes[++PF_idx];                               // only consider primes!
  }
  if (N != 1) factors.push_back(N);      // special case if N is actually a prime
  return factors;         // if pf exceeds 32-bit integer, you have to change vi
}

mii primeFactorsMap(ll N) {        // remember: mii is map of integer to integer
  mii factors;                     // vi `primes' (generated by sieve) is optional
  ll PF_idx = 0, PF = primes[PF_idx];     // using PF = 2, 3, 4, ..., is also ok
  while (N != 1 && (PF * PF <= N)) {   // stop at sqrt(N), but N can get smaller
    while (N % PF == 0) { N /= PF; factors[PF]++; }                // remove this PF
    PF = primes[++PF_idx];                               // only consider primes!
  }
  if (N != 1) factors[N]++;              // special case if N is actually a prime
  return factors;         // if pf exceeds 32-bit integer, you have to change vi
}
```

Divisibilidade

- Um inteiro b divide a, denotado por $b|a$, se $a = bk$ para algum inteiro k:
  - Equivalentemente, b é um divisor de a ou a é um múltiplo de b, se $b|a$;
  - Como conseqüência, o menor divisor natural de qualquer número é 1.
- Como se pode encontrar todos os divisores de um número inteiro?
  - Pode-se utilizar os fatores primos.

Divisibilidade

- A partir do teorema fundamental da aritmética sabe-se que um inteiro *n* é unicamente representado pelo produto de seus fatores primos:
  - Cada divisor é o produto de algum subconjunto dos fatores primos;
  - Tais subconjuntos podem ser construídos utilizando *backtracking*;
  - Mas cuidado com fatores primos repetidos. Por exemplo, 12 (2, 2 e 3) e divisores (1,2,3,4,6,12).

Factors and Factorials (Uva 160)

The factorial of a number *N* (written *N*!) is defined as the product of all the integers from 1 to *N*. It is often defined recursively as follows:

$$1! = 1$$

$$N! = N * (N-1)!$$

Factorials grow very rapidly--5! = 120, 10! = 3,628,800. One way of specifying such large numbers is by specifying the number of times each prime number occurs in it, thus 825 could be specified as (0 1 2 0 1) meaning no twos, 1 three, 2 fives, no sevens and 1 eleven.

Factors and Factorials (Uva 160)

Write a program that will read in a number *N* ($2 \leq N \leq 100$) and write out its factorial in terms of the numbers of the primes it contains.

Input

Input will consist of a series of lines, each line containing a single integer *N*. The file will be terminated by a line consisting of a single 0.

Factors and Factorials (Uva 160)

Output

Output will consist of a series of blocks of lines, one block for each line of the input. Each block will start with the number N, right justified in a field of width 3, and the characters `!', space, and `='. This will be followed by a list of the number of times each prime number occurs in *N*!.

These should be right justified in fields of width 3 and each line (except the last of a block, which may be shorter) should contain fifteen numbers. Any lines after the first should be indented. Follow the layout of the example shown below exactly.

Factors and Factorials (Uva 160)

Sample input

```
5
53
0
```

Sample output

```
  5! =  3  1  1
 53! = 49 23 12  8  4  4  3  2  2  1  1
1  1  1  1  1
```

Factovisors (UVa 10139)

The factorial function, *n*! is defined as follows for all non-negative integers *n*:

0! = 1
n! = n * (n-1)!   *(n > 0)*

We say that *a* divides *b* if there exists an integer *k* such that
k*a = b

The input to your program consists of several lines, each containing two non-negative integers, *n* and *m*, both less than 2^31. For each input line, output a line stating whether or not *m* divides *n*!, in the format shown below.

Factovisors

Sample Input

```
6 9
6 27
20 10000
20 100000
1000 1009
```

Output for Sample Input

```
9 divides 6!
27 does not divide 6!
10000 divides 20!
100000 does not divide 20!
1009 does not divide 1000!
```