Algoritmo do Crivo
Professor Wladimir

Crivo de Eratóstenes

O **Crivo de Eratóstenes** é um algoritmo clássico para encontrar todos os números primos até um limite N.

Algoritmo

1. Criar um vetor booleano $is\_composto[0 \dots N]$ inicializado como `false`, indicando que nenhum número foi marcado como composto.

2. Para cada número i de 2 até N:

   (a) Se i não está marcado como composto ($is\_composto[i] = false$), então i é primo. Adiciona i à lista de primos.

   (b) Marca todos os múltiplos de i como compostos:

   $$\text{para } j = i^2, i^2 + i, i^2 + 2i, \dots \leq N \text{ faça } is\_composto[j] = true$$

   (c) Observação: começamos em i^2 porque todos os múltiplos menores já foram marcados por primos menores.

Complexidade

O custo total do algoritmo é aproximadamente:

$$O\Big(n \sum_{p \leq n} \frac{1}{p}\Big) = O(n \log \log n)$$

onde a soma é feita sobre todos os primos $p \leq n$. Cada número composto é marcado algumas vezes proporcional à quantidade de seus fatores primos.

```
void crivo_eratostenes(int N, vector <int> & primos){
    vector <bool> is_composto(N+1, false);
    for(int i = 2; i <= N; i++){
        if( !is_composto[i] ){
            primos.push_back(i);
            printf("primo %d\n", i);
            for(int j = i*i; j <= N; j += i){
                printf("composto %d\n", j);
                is_composto[j] = true;
            }
        }
    }
}
```

Execução para N = 30

```
primo 2
composto 4
composto 6
composto 8
composto 10
composto 12
composto 14
composto 16
composto 18
composto 20
composto 22
composto 24
composto 26
composto 28
composto 30
primo 3
composto 9
composto 12
composto 15
composto 18
composto 21
composto 24
composto 27
composto 30
primo 5
composto 25
composto 30
primo 7
primo 11
primo 13
primo 17
primo 19
primo 23
primo 29
```

Crivo de Euler (Crivo Linear)

O **Crivo de Euler** é uma variação do Crivo de Eratóstenes que gera todos os números primos até N em **tempo linear**.

Algoritmo

1. Criar um vetor booleano $is\_composto[0 \dots N]$ inicializado como `false`.
2. Manter uma lista $primos$ inicialmente vazia.
3. Para cada número $i = 2$ até N:
   (a) Se $is\_composto[i] = false$, então i é primo. Adiciona i à lista $primos$.
   (b) Para cada primo p em $primos$:
      i. Se $i \cdot p > N$, interrompa o loop.
      ii. Marcar $i \cdot p$ como composto: $is\_composto[i \cdot p] = true$.
      iii. Se $i \bmod p = 0$, interrompa o loop (*break*). Isso garante que cada número composto seja marcado **apenas pelo seu menor primo divisor**.

Complexidade

O algoritmo processa cada número composto exatamente uma vez, resultando em complexidade:

$$O(N)$$

Código

```
void crivo_euler(int N, vector <int> & primos) {
    vector <bool> is_composto(N+1, false);
    for (int i = 2; i <= N; i++) {
        if (!is_composto[i])
            primos.push_back(i);
        printf("analisando i %d\n", i);
        for (int p : primos) {
            if (i * p > N) break;
            is_composto[i * p] = true;
            printf("composto %d\n", i*p);
            if (i % p == 0) break; // p é o menor primo que divide i
        }
    }
}
```

Execução para N = 30

```
analisando i 2
composto 4
analisando i 3
composto 6
composto 9
analisando i 4
composto 8
analisando i 5
composto 10
composto 15
composto 25
analisando i 6
composto 12
analisando i 7
composto 14
composto 21
analisando i 8
composto 16
analisando i 9
composto 18
composto 27
analisando i 10
composto 20
analisando i 11
composto 22
analisando i 12
composto 24
analisando i 13
composto 26
analisando i 14
composto 28
analisando i 15
composto 30
analisando i 16
analisando i 17
analisando i 18
analisando i 19
analisando i 20
analisando i 21
analisando i 22
analisando i 23
analisando i 24
analisando i 25
analisando i 26
analisando i 27
analisando i 28
analisando i 29
```

```
analisando i 30
```

Considere $i = 6$, com a lista de primos encontrada até o momento: $[2, 3, 5]$. Em princípio, poderíamos marcar os múltiplos:

$$6 \cdot 2 = 12, \quad 6 \cdot 3 = 18, \quad 6 \cdot 5 = 30.$$

No entanto, como 2 divide 6, aplicamos o `break` do Crivo de Euler e marcamos apenas 12. Os múltiplos 18 e 30 não são perdidos; eles serão marcados mais tarde:

- 18 será marcado quando $i = 9$ $(9 \cdot 2)$,
- 30 será marcado quando $i = 15$ $(15 \cdot 2)$.

Dessa forma, cada número composto é marcado **uma única vez**, sempre pelo seu **menor primo divisor**, garantindo a eficiência linear do algoritmo.

A Função Totiente de Euler

A **função totiente de Euler**, denotada por $\varphi(n)$, é definida como o número de inteiros positivos menores ou iguais a n que são coprimos com n. Formalmente:

$$\varphi(n) = |\{k \in \mathbb{N} : 1 \leq k \leq n,\ \gcd(k, n) = 1\}|$$

Por exemplo:

$$\varphi(1) = 1, \quad \varphi(2) = 1, \quad \varphi(6) = 2 \text{ (pois apenas 1 e 5 são coprimos com 6)}$$

Propriedades Fundamentais

A função totiente é **multiplicativa**, isto é, se $\gcd(a, b) = 1$, então:

$$\varphi(ab) = \varphi(a) \cdot \varphi(b)$$

Além disso, se p é um número primo, temos:

$$\varphi(p) = p - 1$$

e, mais geralmente:

$$\varphi(p^k) = p^k - p^{k-1} = p^k \left(1 - \frac{1}{p}\right)$$

Assim, para a fatoração prima de $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, vale:

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

Implementação do Crivo

O código abaixo implementa esse raciocínio:

```
vector <int> phi(MAXN);
phi[1] = 1;
for(int i = 2; i < MAXN; i++) phi[i] = i;
for(int i = 2; i <  MAXN; i++){
    if( phi[i] == i ){
        phi[i] = i - 1;
        for(int j = 2*i; j < MAXN; j += i){
            phi[j] = (phi[j]*(i-1))/i;
        }
    }
}
```

Complexidade e Vantagens

- **Complexidade de tempo:** $O(n \log \log n)$
- **Complexidade de espaço:** $O(n)$
- Calcula todos os valores de $\varphi(i)$ de forma eficiente, sem fatorar cada número individualmente.

Exercícios

https://www.spoj.com/problems/DCEPCA03/

```
#include <bits/stdc++.h>

using namespace std;

#define size 10000

long long tot[size+1];

void euler_tot() {
    tot[1] = 1;
    for(int i=2; i<size; i++)
    {
        if(!tot[i])
        {
            tot[i] = i-1;
            for(int j=(i<<1); j<=size; j+=i)
            {
                if(!tot[j]) tot[j] = j;
                tot[j] = (tot[j]/i)*(i-1);
            }
        }
    }

    for(int i=2;i<=size;++i)
       tot[i]+=tot[i-1];
}

int main(){

 int n,t;

 euler_tot();

 scanf("%d", &t);
 while(t--){
        scanf("%d", &n);
        printf("%lld\n", tot[n]*tot[n] );
 }
}
```