Resolução Exercícios Teoria dos Números

Exercício B, E e F

B - T-Primes

- **T-Primes** são números positivos que tem exatamente **3 divisores distintos**.
- Dado **n** inteiros, determine quando os mesmo são ou não **T-primes.**

Entrada:

n $(1 \leq n \leq 10^5)$

x x_i $(1 \leq x_i \leq 10^{12})$

B - T-Primes

- **Como encontrar os T-Primes ?**

  Pela definição, conseguimos assumir que um número ***x*** é um T-prime, se seus divisores forem apenas:

  1, (algum_numero_primo), x

  Por exemplo, um número primo, não é um T-prime, pois ele é divisível apenas por ele mesmo e 1.

B - T-Primes

- Mas qual número primo se encaixa nessa configuração?

1 < (algum_numero_primo) < x

O único número que se encaixa nessa configuração seria um número primo multiplicado por ele mesmo que gerasse x, exemplo:

Para x = 9,

1 < **3** < 9

B - T-Primes

- **Solução:** Dado um número **n:**

```
if (n == 1) cout << "NO";
else if (quadrado_perfeito(n) && ehPrimo[sqrt(n)])
    cout << "YES";
else
    cout << "NO";
```

B - T-Primes

Para x = 16,

16 é um **quadrado perfeito**, mas como sqrt(16) = 4 e 4 não é um número primo, ficamos no final com mais de 3 divisores:

1,2,4,8,16

B - T-Primes

```
vector<bool> ehPrimo;
ll MAXN = 1000000;
void crivo()
{
    ehPrimo = vector<bool>(MAXN + 1, true);
    ehPrimo[0] = ehPrimo[1] = false;
    for (ll i = 2; i * i <= MAXN; i++)
    {
        if (!ehPrimo[i])continue;
        for (ll m = i * i; m <= MAXN; m += i)
            ehPrimo[m] = false;
    }
}
```

E - RSA Attack

- **O enunciado nos apresenta o seguinte problema**
  - Achar um inteiro m que satisfaça a equação:

$$m^e \pmod n = c \pmod n$$

  - Onde:
    - **n** é o **produto de dois números primos ímpares**, p e q
    - **gcd(e, (p - 1)(q - 1)) = 1**
    - **e < (p - 1)(q - 1)**
    - **e, n, c <= 32000**

E - RSA Attack

- **1ª ETAPA:** manipular a equação
  - Precisamos encontrar o valor de m, mas, no estado atual de nossa equação, temos apenas o valor de m^e, logo, queremos eliminar essa potência.

$$m^e \pmod{n} = c \pmod{n}$$

elevar ambos os lados da equação por x

$$(m^e)^x \pmod{n} = (c)^x \pmod{n}$$

$$m^{ex} \pmod{n} = c^x \pmod{n}$$

E - RSA Attack

- Podemos, então, supor que **$ex = 1$**, pois, assim, **$m^{ex} = m^1$**.
- Assim, temos:

$$m \pmod{n} = c^x \pmod{n}$$

- Agora, sabemos que o **valor de m** é o resultado da **potência c^x**.
- Contudo, **não sabemos o valor de x**, portanto precisamos encontrá-lo.

E - RSA Attack

- **2ª ETAPA:** encontrar o valor de x
  - Para tal, vamos utilizar as seguintes afirmações do exercício:
    i. $ex = 1$
    ii. $gcd(e, (p - 1)(q - 1)) = 1$
    iii. $e < (p - 1)(q - 1)$
  - Podemos perceber, pelas afirmações **i** e **ii**, que:

$$\mathbf{ex = gcd(e, (p - 1)(q - 1)) = 1}$$

E - RSA Attack

- Uma equação diofantina tem o formato:

$$ax + by = c$$

- E sabemos que podemos resolver através do algoritmo estendido de Euclides equações diofantinas com a seguinte configuração;

$$ax + by = gcd(a, b) = c$$

E - RSA Attack

- Comparando nossa equação atual com a equação diofantina geral:

$$ex = gcd(e, (p - 1)(q - 1)) = 1 \quad (I)$$

$$ax + by = gcd(a, b) = c \quad (II)$$

- Percebemos que:
  - **a = e**
  - **b = (p - 1)(q - 1)**
  - **c = 1**

E - RSA Attack

- Portanto, podemos escrever nossa equação como:

$$ex + (p - 1)(q - 1)y = \gcd(e, (p - 1)(q - 1)) = 1$$

- E **descobrimos os valores de x e de y** aplicando o **algoritmo estendido de Euclides**.
- O exercício nos **garante** que encontraremos **uma solução** para o problema, então não precisamos nos preocupar com isso.

E - RSA Attack

- **3ª Etapa:** definir os valores para p e q
  - Os valores de p e q não estão definidos para aplicarmos na equação que obtemos, porém sabemos que **n = p * q**.
  - Assim, testamos valores de p e q da seguinte maneira:
    1. Dado um número **p**, **verificamos se p é primo**.
    2. Se p é primo, verificamos se a **divisão n / p é inteira** por meio da **expressão n % p == 0**.
    3. Se a divisão for inteira, v**erificamos se o valor q = n / p também é primo**.
    4. Se **todos os itens anteriores forem verdadeiros**, **encontramos p e q**.

E - RSA Attack

- **Cuidados:**
  - **x pode ser um valor negativo** na resolução da equação diofantina, pois se **uma solução é possível**, elas **admitem infinitas soluções**.
  - Porém, temos que:

$$e < (p - 1)(q - 1) \quad (I)$$

$$ex = 1 \rightarrow e = 1 / x \quad (II)$$

  - Para garantir I, podemos reescrever II como:

$$e \ (\text{mod } (p - 1)(q - 1)) * x = 1$$

E - RSA Attack

- Aplicando propriedades da álgebra modular, obtemos:

ex (mod (p - 1)(q - 1)) = 1 (mod (p - 1)(q - 1))

- Assim, e levando em consideração que **x é a multiplicativa inversa de e,** podemos concluir que:

e, x < (p - 1)(q - 1) = mod

$$\begin{cases} \textbf{Se x < 0, x = (x \% mod + mod) \% mod} \\ \textbf{Senão, x = x \% mod} \end{cases}$$

E - RSA Attack

```
vector<bool> is_prime;
vector<int> primes;

void crivo(const int &n) {
    is_prime = vector<bool>(n + 1, true);
    is_prime[0] = is_prime[1] = false;

    for (int i = 2; i * i <= n; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= n; j += i) {
                is_prime[j] = false;
            }
        }
    }

    for (int i = 2; i <= n; i++) {
        if (is_prime[i])
            primes.push_back(i);
    }
}
```

E - RSA Attack

```
crivo(32000);
while (k--) {
    int e, n, c, p, q;
    cin >> e >> n >> c;

    for (int i = 0; i < m; i++) {
        p = primes[i];

        if (n % p == 0 && is_prime[n / p]) {
            q = n / p;
            break;
        }
    }

    int x, y, mod = (p - 1) * (q - 1);
    extended_gcd(e, mod, x, y);
    x = (x % mod + mod) % mod;

    cout << pow_mod(c, x, n) << "\n";
}
```

E - RSA Attack

```
ll pow_mod(ll b, ll x, ll mod) {
    ll m = 1LL;

    while (x) {
        if (x & 1) {
            m = (m * b) % mod;
        }

        b = (b * b) % mod;
        x >>= 1;
    }

    return m;
}
```

F - DDF

- É dado o início e o fim de um intervalo
- O objetivo é saber qual o número com a maior sequência DDF nesse intervalo.
- DDF -> Decimal Digit Factor Sequence

F - DDF

DDF

712 72 69 24 33 12 19 11 3 4 7 8 15

F - DDF

DDF

712 72 69 24 33 12 19 11 3 4 7 8 15

Divisores de 712 -> 1 2 4 8 89 178 356 712

Soma dos dígitos dos divisores

1 + 2 + 4 + 8 + 8 + 9 + 1 + 7 + 8 + 3 + 5 + 6 = 72

F - DDF

DDF

712 72 69 24 33 12 19 11 3 4 7 8 15

Divisores de 712 -> 1 2 4 8 89 178 356 712

Soma dos dígitos dos divisores

1 + 2 + 4 + 8 + 8 + 9 + 1 + 7 + 8 + 3 + 5 + 6 = 72

F - DDF

DDF

712 72 69 24 33 12 19 11 3 4 7 8 15

Divisores de 72 -> 1 2 3 4 6 8 9 12 18 24 36 72

Soma dos dígitos dos divisores

1 + 2 + 3 + 4 + 6 + 8 + 9 + 1 + 2 + 1 + 8 + 2 + 4 + 3 + 6 + 7 + 2 = 69

F - DDF

Objetivo: Encontrar a maior DDF no intervalo dado.

- Como o intervalo vai de 1 a 3000 no máximo e o tamanho de uma DDF é no máximo 1000, pode-se calcular todas as DDFs, de maneira offline, e pegar a maior.

F - DDF

```
ll soma_digitos(ll num){
    ll sum = 0;
    while(num){
        sum += num%10;
        num/=10;
    }
    return sum;
}
ll fatorar(ll n){
    vector<ll> fator;
    ll soma = 0;
    for (ll i = 1; i * i <= n; i++){
        if(n%i == 0){
            if(n != i*i)soma += soma_digitos(n/i);
            soma += soma_digitos(i);
        }
    }
    return soma;
}
```

F - DDF

```
vector<ll> ddf [3001];
for(int i = 1; i <= 3000; i++){
    ll ant = i, res = i;
    ddf[i].push_back(i);
    while(1){
        res = fatorar(res);
        if(ant == res)break;
        ddf[i].push_back(res);
        ant = res;
    }
}
```