UNIVERSIDADE DE SÃO PAULO

TRABALHO DE FORMATURA

Teoria dos Números e Computação: Uma abordagem utilizando problemas de competições de programação

Autor:
Antonio R. de Campos Junior

Supervisor:
Dr. Carlos Eduardo Ferreira

Tese apresentada em cumprimento dos requisitos para o curso Bacharel em Ciência da Computação

Instituto de Matemática e Estatística

21 de outubro de 2015

"To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science."

Albert Einstein

Resumo

Teoria do Números é um vasto ramo da matemática que estuda números inteiros. Números primos, fatorização de números inteiros, funções aritméticas, são alguns dos tópicos mais estudados e também importantes para resolução de problemas computacionais.

Hoje em dia a importância da Teoria do Números na Computação é inquestionável, e desse modo, esse trabalho vem ilustrar como a teoria pode ser aplicada na criação de algoritmos para resolução de problemas computacionais, em especial problemas de competições de programação.

Equações diofantinas, Congruência Modular, Números de Fibonacci, são alguns dos assuntos que serão abordados nesse trabalho. Após a devida demostração da teoria serão exibidos alguns problemas de competições de programação que aplicam essa teoria, seguido da implementação e análise do algoritmo que resolve o problema abordado.

Agradecimentos

I like to acknowledge ...

Sumário

Lista de Figuras

Lista de Tabelas

For/Dedicated to/To my...

Capítulo 1

Divisibilidade

1.1 Introdução

A noção de divisibilidade dos números inteiros é fundamental na **Teoria dos Números**. Nesse seção vamos descrever algumas definições e propriedades que serão utilizados ao longo desse trabalho.

Definição 1 *A notação $d|n$ ("d **divive** n"), significa que existe um inteiro q, tal que, $n = dq$. Se $d|n$ dizemos que n é múltiplo de d. Caso n não seja múltiplo de d (ou seja, d não divide n), escrevemos $d \nmid n$.*

Corolário 1 $d|n$, $d|m \Rightarrow d|(n+m)$

Demonstração: Se $d|n$ e $d|m$, então existe inteiros q e k, tal que, $n = qd$ e $m = kd$. Desse modo temos:
$(n+m) = qd + kd = (q+k)d \Rightarrow d|(n+m)$ □

Corolário 2 $d|(\frac{n}{m}) \Rightarrow dm|n$

Demonstração:
$d|(\frac{n}{m}) \Rightarrow \exists q \in \mathbb{Z} \mid \frac{n}{m} = qd$
$d|(\frac{n}{m}) \Rightarrow n = q(dm) \Rightarrow dm|n$ □

Corolário 3 *Dado um subconjunto dos inteiros $S = \{S_1, S_2, S_3, ..., S_n\}$ ordenado crescentemente, e um número inteiro d, tal que, $d|(S_i - S_{i-1})$, $2 \leq i \leq n$, temos que:*
$d|(S_i - S_j), \forall S_i, S_j \in S$.

Demonstração: Tome $S_i, S_j \in S$ quaisquer, e sem perda de generalidade assuma que $S_i \geq S_j$ (ie, $i \geq j$, pois S está ordenado crescentemente).
Como $i \geq j$, tome $r \in \mathbb{N}$ como sendo a diferença entre i e j : $i = j + r$.
Vamos agora provar por indução que $d|(S_{j+r} - S_j)$.
Para $r = 0$ ou $r = 1$ a demostração segue trivialmente.
Assuma que o corolário funciona para $(r-1)$, ie, $d|(S_{j+r-1} - S_j)$.
Temos então que:
$d|(S_{j+r} - S_{j+r-1}) \Rightarrow d|(S_{j+r} - S_{j+r-1}) + (S_{j+r-1} - S_j)$ (▷ **Corolário** 1)
$d|(S_{j+r} - S_{j+r-1}) \Rightarrow d|(S_{j+r} - S_j)$ □

Corolário 4 *O **Corolário** 3 funciona mesmo se o conjunto S não estiver ordenado.*

Demonstração: Deixaremos a demostração a cargo do leitor.

Teorema 1 (Teorema da divisão) *Para todo número inteiro a e qualquer número inteiro positivo n, existe inteiros únicos q e r, tal que:*
$a = qn + r, 0 \leq r < n$
*O valor q ($q = \lfloor \frac{a}{n} \rfloor$) é chamado de **quociente** da divisão, e o valor r ($r = a \bmod n$) é chamado de **resto** (ou **resíduo**) da divisão.*

Demonstração: Suponha que q e r não sejam únicos, ie, que exista q^* e r^* tal que: $a = q^*n + r^*, 0 \leq r^* < n$.
$a = qn + r = q^*n + r^* \Rightarrow (r - r^*) = (q^* - q)n \Rightarrow (r - r^*) \equiv (q^* - q)n \equiv 0(mod\ n)$
Porém, como $r \neq r^*$, e tanto r quanto r^* são menores que n, temos que:
$r \not\equiv r^*(mod\ n) \Rightarrow (r - r^*) \not\equiv 0(mod\ n)$
Chegando numa contradição, e assim q e r são únicos □

Corolário 5 $d|n$, $d|m \Rightarrow d|(n \bmod m)$

Demonstração:
$d|n \Rightarrow n = k_1 d, k_1 \in \mathbb{Z}$
$d|m \Rightarrow m = k_2 d, k_2 \in \mathbb{Z}$
$n = qm + (n \bmod m) \Rightarrow (n \bmod m) = n - qm$ (▷ Teorema 1)
$(n \bmod m) = k_1 d - qk_2 d = (k_1 - qk_2)d \Rightarrow d|(n \bmod m)$ □

Corolário 6 $d|m$, $d|(n \bmod m) \Rightarrow d|n$

Demonstração:
$d|m \Rightarrow m = k_1 d, k_1 \in \mathbb{Z}$
$d|(n \bmod m) \Rightarrow (n \bmod m) = k_2 d, k_2 \in \mathbb{Z}$
$n = qm + (n \bmod m) \Rightarrow n = qk_1 d + k_2 d$ (▷ Teorema 1)
$n = (qk_1 + k_2)d \Rightarrow d|n$ □

1.2 Números Primos

Definição 2 *Todo número inteiro n (n > 1) que têm apenas dois divisores distintos (1 e n) é chamado de número primo. Se n (n > 1) não for primo, dizemos que n é número composto.*

Teorema 2 (Fatoração Única) *Um número natural qualquer n, pode ser escrito unicamente como um produto da forma: $n = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$, onde os p_i são números primos, $p_1 < p_2 < ... < p_k$, e os números a_i são inteiros positivos.*

Demonstração: Deixaremos a demostração a cargo do leitor. **Dica:** Use o fato de que o conjunto dos primos que divide um número inteiro é único, e fato de que se qualquer potência a_i for alterado o valor de n será alterado.

1.2.1 Crivo de Erastótenes

O *Crivo de Erastótenes* é um algoritmo criado pelo matemático **Erastótenes** (a.C. 285-194 a.C.) para o cálculo de números primos até um certo valor limite N. O algoritmo mantém uma tabela com N elementos, e para cada primo, começando pelo número 2, marca na tabelo os números compostos múltiplos desses primos. Desse modo, ao final do algoritmo, os elementos não marcados são números primos.

Pseudocódigo:
Análise: TODO

Algorithm 1 Crivo de Erastótenes paro o cálculo de números primos

```
1:  procedure CrivoErastótenes (N)
2:      isPrime[] ← new Array[N]                ▷ isPrime[] é um vetor booleano
3:      for (p = 2; p ≤ N; p++) do
4:          isPrime[p] ← true
5:      for (p = 2; p² ≤ N; p++) do
6:          if isPrime[p] = false then
7:              continue
8:          for (n = p²; n ≤ N; n = n + p) do
9:              isPrime[n] ← false
10:     return isPrime[]
```

1.3 Máximo Divisor Comum

Definição 3 *O Máximo Divisor Comum de dois inteiros quaisquer a e b (com a ou b diferente de zero), denotado por $MDC(a, b)$, é o maior inteiro que divide ambos a e b. Se $MDC(a, b) = 1$ dizemos que a e b são primos entre si.*

Corolário 7 *Para números inteiros quaisquer a e b, $MDC(a, b) = MDC(b, a \bmod b)$*

Demonstração: Pelo **Corolário** 5 e 6, temos:
$d|a, d|b \Leftrightarrow d|b, d|(a \bmod b)$
Assim, qualquer divisor de a e b é também divisor de b e $(a \bmod b)$ (e vise versa). Implicando que o **Máximo Divisor Comum** de a e b é igual ao **Máximo Divisor Comum** de b e $(a \bmod b)$. □

Corolário 8 $MDC(a, b) = d \Rightarrow MDC(\frac{a}{d}, \frac{b}{d}) = 1$

Demonstração: Suponha que $MDC(\frac{a}{d}, \frac{b}{d}) = r > 1$. Assim temos:
$r|\frac{a}{d} \Rightarrow dr|a$ (▷ **Corolário** 2)
$r|\frac{b}{d} \Rightarrow dr|b$ (▷ **Corolário** 2)
$r > 1 \Rightarrow dr > d \Rightarrow dr > MDC(a, b)$
Chegamos então numa contradição, pos dr é divisor comum de a e b, e dr é maior que o **Máximo Divisor Comum** de a e b. □

1.3.1 Algoritmo de Euclides

A ideia principal do **Algoritmo de Euclides** é calcular recursivamente o **Máximo Divisor Comum** de dois números baseando-se no **Corolário** 7.

Pseudocódigo:

Algorithm 2 Algoritmo de Euclides

```
1:  procedure MDC(a, b)
2:      if b = 0 then
3:          return a
4:      else
5:          return MDC(b, a mod b)
```

1.3.2 Teorema de Bézout

Teorema 3 (Teorema de Bézout) $\forall\ a, b \in \mathbb{Z}, \exists\ x, y \in \mathbb{Z} \mid ax + by = mdc(a, b)$.

Demonstração: De acordo com Teorema 3

Corolário 9 *Para números inteiros quaisquer a e b, $MDC(a, b) = MDC(a, a \pm b)$*

Demonstração: A prova dessa expressão vem do fato de que qualder divisor de a e b, é também divisor de $(a \pm b)$.

Corolário 10 *Para números inteiros quaisquer a e b, temos:*
$MDC(a, b) = 1 \Rightarrow MDC(a, bk) = MDC(a, k)$, *com* $k \in \mathbb{Z}$

Demonstração: A prova dessa expressão vem do fato de que qualder divisor d de a e bk, é também divisor de k, pois d não divide b ($MDC(a, b) = 1$).

1.4 Crivo de Erastóteles

1.5 Problemas Propostos

1.5.1 UVA-10407

10407 - Simple Division

Resumo: Tome $P(S) := \{x \in \mathbb{Z} \mid \forall a, b \in S, a \equiv b(mod\ x)\}$ em que $S \subset \mathbb{Z}$.
O problema consiste em encontrar o valor máximo de $P(S)$ dado um conjunto S.

Solução: Seja $S = \{S_1, S_2, S_3, ..., S_n\}$, com $n = |S|$, o conjunto dado pelo problema (assumiremos que os valores de S estão ordenados crescentemente).
Tome um número qualquer $d \in P(S)$. Por definição temos que $\forall S_i, S_j \in S$, $S_i \equiv S_j(mod\ d) \Rightarrow (S_i - S_j) \equiv 0(mod\ d) \Rightarrow d \mid (S_i - S_j)$.
Pelo **Corolário** 3 sabemos que:
$d|(S_i - S_{i-1}), \forall i \in \mathbb{N}, 2 \leq i \leq n \Rightarrow d|(S_i - S_j), \forall S_i, S_j \in S \Rightarrow d \in P(S)$.
E desse modo, para calcular o valor máximo de $P(S)$ só precisamos calcular o Máximo Divisor Comum das diferenças $(S_i - S_{i-1})$ com i variando de 2 à n □.

Pseudocódigo:

Algorithm 3 Simple Division

```
1: procedure GetMaximumValue (S)
2:     S ← sort(S)                          ▷ sort(X) retorna o conjunto X ordenado.
3:     maxValue ← 0
4:     for i := 2 to |S| do
5:         maxValue ← MDC(maxValue, S_i − S_{i−1})
6:     return maxValue
```

Capítulo 2

Aritmética Modular

2.1 Congruência

Definição 4 *Para a e b inteiros, dizemos que a é congruente à b módulo m ($a \equiv b(mod\ m), m > 0$) se a e b produzem o mesmo resto na divisão por m (ie, $m|(a-b)$). Caso contrário ($m \nmid (a-b)$), dizemos que a não é congruente à b módulo m ($a \not\equiv b(mod\ m)$).*

Definição 5 *Dizemos que o conjunto de inteiros $S = \{s_1, s_2, ..., s_k\}$ é um sistema completo de resíduos modulo n se:*

1. $\forall a \in \mathbb{Z}, \exists! s_i \in S \mid a \equiv s_i (mod\ n)$
2. $s_i \not\equiv s_j (mod\ n)$ *para* $i \neq j$

2.2 Congruência Linear

Definição 6 *Congruências da forma $ax \equiv b(mod\ m)$, onde a, b e m são inteiros e x é uma incógnita, são chamadas de Congruências Lineares.*

Teorema 4 *$ax \equiv b(mod\ m)$ tem solução, se e somente se, $MDC(a, m)|b$*

Demonstração: TODO

2.3 Teoremas de Fermat e do Resto Chinês

2.3.1 Teorema de Fermat

Teorema 5 (Pequeno Teorema de Fermat) *Dado um número primo qualquer p, temos que:* $a^{p-1} \equiv 1 (\mathrm{mod}\ p), \forall a \in \mathbb{Z} \mid MDC(a, p) = 1$

Demonstração: Deixaremos a demostração a cargo do leitor.

Teorema 6 *Dados os inteiros quaisquer a, b, c e um número primo p, com $MDC(a, p) = 1$, temos que:*
$a^{b^c} \equiv a^{b^c \bmod (p-1)} (\mathrm{mod}\ p)$

Demonstração: Deixaremos a demostração a cargo do leitor.

2.3.2 Teorema do Resto Chinês

Teorema 7 (Teorema do Resto Chinês) *Tome o sistema de congruências lineares:*

$a_1x \equiv c_1 (mod\ m_1)$
$a_2x \equiv c_2 (mod\ m_2)$
$a_3x \equiv c_3 (mod\ m_3)$
...
$a_nx \equiv c_n (mod\ m_n)$

Em que $c_i \in \mathbb{Z}$, $MDC(a_i, m_i) = 1$, *e* $MDC(m_i, m_j) = 1$ *para* $i \neq j$ *Nessas condições o sistema acima tem solução única módulo* M, *em que* $M = m_1m_2m_3...m_n$.

Demonstração: Deixaremos a demostração a cargo do leitor.

2.4 Problemas Propostos

2.4.1 UVA-10090

10090 - Marbles

Resumo: É dado um número inteiro n ($0 < n \leq 10^8$). O problema consite em verificar se n pode, ou não pode, ser escrito como a soma de dois números primos. E em caso afirmativo encontrar o valor desses dois primos.

Solução:

Pseudocódigo:

Algorithm 4 Marbles

1: **procedure** FINDTWOPRIMESSUM (N)

Análise:

2.4.2 CodeChef-IITK2P10

IITK2P10 - Chef and Pattern

Resumo: Tome a seguinte função $f_K : \mathbb{N}^* \longmapsto \mathbb{N}$:

$$f_K(x) = \begin{cases} 1 & \text{se } x = 1 \\ K & \text{se } x = 2 \\ \prod_{i=1}^{x-1} f_K(i) & \text{se } x \geq 3 \end{cases}$$

São dados dois número inteiro N, K ($1 \leq N \leq 10^9$, $1 \leq K \leq 10^5$). O problema consiste em calcular a expressão: $f_K(N) \bmod p$, em que $p = (10^9 + 7)$.

Solução: Escrevendo os valores dos primeiros termos que a função assume, temos: $f(1) = 1, f(2) = K, f(3) = K, f = (4) = K^2, f(5) = K^4, f(6) = K^8, f(7) = K^{16}$.
Provaremos, por indução, que $f_K(N) = K^{2^{N-3}}, N \geq 3$.
Para os primeiros termos essa expressão é trivialmente verificada.

Assuma que a expressão funciona para algum número natural qualquer $(R-1) \geq 3$ ($f_K(R-1) = K^{2^{R-4}}$).

Nessas condições temos que:

$f_K(R) = \prod_{i=1}^{R-1} f_K(i) = 1.K. \prod_{i=3}^{R-1} f_K(i) = K \prod_{i=3}^{R-1} K^{2^{i-3}} = K \prod_{j=0}^{R-4} K^{2^j}$

$f_K(R) = KK^{\sum_{j=0}^{R-4} 2^j} = KK^{2^{R-3}-1} = K^{2^{R-3}}$ □

Para calcular o valor de $f_K(N) \bmod p$, podemos aplicar o Teorema 6, já que p é um número primo e $MDC(p, K) = 1$:

$f_K(N) \bmod p = K^{2^{N-3}} \bmod p = K^{2^{N-3} \bmod (p-1)} \bmod p$

Reduzindo o problema, dessa maneira, em calcular: $K^{2^{N-3}} \bmod (10^9 + 7)$.

Pseudocódigo:

Algorithm 5 Chef and Pattern

```
1: procedure F (N, K)
2:     p ← (10^9 + 7)
3:     exp ← EXPMOD(2, N − 3, p − 1)          ▷ exp = 2^{N−3} mod (p − 1)
4:     solution ← EXPMOD(K, exp, p)           ▷ solution = K^{2^{N−3} mod (p−1)} mod p
5:     return solution
```

Análise: Como vimos anteriormente, as linhas 3 e 4 do algoritmo consomem tempo proporcional à $O(n \log n)$, e assim a complexidade total é $O(n \log n)$.

Capítulo 3

Funções Aritméticas

3.1 Φ de Euler

Definição 7 *A Função Totiente de Euler, denotada por $\Phi(n)$, é a função aritmética que conta o número de inteiros positivos menores ou iguais a n que são primos entre si com n.*

$\Phi(n) := |\{x \in \mathbb{N}^* \mid MDC(x, n) = 1\}|$

Teorema 8 *$\Phi(n^k) = n^{k-1}\Phi(n)$, para inteiros positiovos quaisquer n e k. Em particular $\Phi(p^k) = (p^k - p^{k-1})$, para p primo.*

Demonstração: TODO

Teorema 9 *$\Phi(n)$ é função multiplicativa, ie, $\Phi(mn) = \Phi(m)\Phi(n)$ para $MDC(m, n) = 1$.*

Demonstração: TODO

Teorema 10 (Fórmula Produto de Euler) $\Phi(n) = n\prod_{p|n}(1 - \frac{1}{p}) = n\prod_{p|n}(\frac{p-1}{p})$

Demonstração:

$\Phi(n) = \Phi(p_1^{a_1}p_2^{a_2}...p_k^{a_k})$ (▷ Teorema 2)
$\Phi(n) = \Phi(p_1^{a_1})\Phi(p_2^{a_2})...\Phi(p_k^{a_k})$ (▷ Teorema 9)
$\Phi(n) = (p_1^{a_1} - p_1^{a_1-1})(p_2^{a_2} - p_2^{a_2-1})...(p_k^{a_k} - p_k^{a_k-1})$ (▷ Teorema 8)
$\Phi(n) = p_1^{a_1}p_2^{a_2}...p_k^{a_k}(1 - 1/p_1)(1 - 1/p_2)...(1 - 1/p_k)$
$\Phi(n) = n\prod_{p|n}(1 - \frac{1}{p})$ □

Pseudocódigo:

Algorithm 6 Calcula os primeiros N termos da função Φ

```
1:  procedure PHI(N)
2:      Φ[] ← newArray[N]
3:      for (p = 1; p ≤ N; p++) do
4:          Φ[p] ← p
5:      for (p = 2; p ≤ N; p++) do
6:          if Φ[p] ≠ p then                    ▷ Φ[p] ≠ p ⇔ p não é primo
7:              continue
8:          for (n = p; n ≤ N; n = n + p) do
9:              Φ[n] ← Φ[n](p−1/p)
10:     return Φ[]
```

3.1.1 Teorema de Euler

Teorema 11 (Teorema de Euler) *Dados números inteiros a e n primos entre si, temos que:* $a^{\Phi(n)} \equiv 1 (\mathrm{mod}\ n)$

Demonstração: TODO usa residuos completo mod m

3.2 Sequência de Fibonacci

Definição 8 *A sequência de Fibonacci Fib_n é uma sequência de números inteiros positivos em que cada termo subsequente corresponde a some dos dois termos anteriores.*

$$Fib_n := \begin{cases} 0 & se\ n = 0 \\ 1 & se\ n = 1 \\ Fib_{n-1} + Fib_{n-2} & se\ n \geq 2 \end{cases}$$

Corolário 11 $MDC(Fib_n, Fib_{n-1}) = 1$, *para* $n \geq 2$

Demonstração: Tome os primeiros termos da sequência de fibonacci: $1, 1, 2, 3, 5, 8, ...$ Claramente a expressão acima funciona para os primeiros termos. Assuma que a expressão funciona para um inteiro qualquer $(k-1) > 2$ ($MDC(Fib_{k-1}, Fib_{k-2}) = 1$).

Provaremos por indução que a expressão sempre funciona.
$MDC(Fib_k, Fib_{k-1}) = MDC(Fib_{k-1} + Fib_{k-2}, Fib_{k-1})$
$MDC(Fib_{k-1} + Fib_{k-2}, Fib_{k-1}) = MDC(Fib_{k-2}, Fib_{k-1})$ (▷ Pelo **Corolário** 9)
Logo, temos que:
$MDC(Fib_k, Fib_{k-1}) = MDC(Fib_{k-2}, Fib_{k-1}) = 1$ □

Corolário 12 $Fib_{m+n} = Fib_m Fib_{n+1} + Fib_{m-1} Fib_n$

Demonstração: Provaremos esse corolário por indução no índice n.

A base da indução será, $n = 2$:
$Fib_{m+2} = Fib_m + Fim_{m+1} = Fib_m + Fib_m + Fib_{m-1}$
$Fib_{m+2} = 2Fib_m + 1Fib_{m-1} = Fib_m Fib_3 + Fib_{m-1} Fib_2$
Assumindo que a expressão funciona para todos os valores menores que n, temos:
$Fib_{m+n} = Fib_{m+n-2} + Fib_{m+n-1}$
$Fib_{m+n} = (Fib_m Fib_{n-1} + Fib_{m-1} Fib_{n-2}) + (Fib_m Fib_n + Fib_{m-1} Fib_{n-1})$
$Fib_{m+n} = Fib_m(Fib_{n-1} + Fib_n) + Fib_{m-1}(Fib_{n-2} + Fib_{n-1})$
$Fib_{m+n} = Fib_m Fib_{n+1} + Fib_{m-1} Fib_n$ □

Teorema 12 $MDC(Fib_m, Fib_n) = Fib_{MDC(m,n)}, \forall m, n \in \mathbb{Z}$

Demonstração:

$MDC(Fib_m, Fib_n) = MDC(Fib_m, Fib_{qm+r})$ (▷ Teorema 1, $n = qm + r, 0 \leq r < n$)
$MDC(Fib_m, Fib_n) = MDC(Fib_m, Fib_{qm} Fib_{r+1} + Fib_{qm-1} Fib_r)$ (▷ **Corolário** 12).
$MDC(Fib_m, Fib_n) = MDC(Fib_m, Fib_{qm-1} Fib_r)$
Pelo **Corolário** 10 e sabendo que $MDC(Fib_m, Fib_{qm-1}) = 1$, temos:
$MDC(Fib_m, Fib_n) = MDC(Fib_m, Fib_r)$
$MDC(Fib_m, Fib_n) = MDC(Fib_m, Fib_{n \bmod m})$

Se tirarmos o símbolo funcional Fib, a última equação forma um passo do **Algoritmo de Euclides** ($MDC(m, n) = MDC(m, n \bmod m)$).

Podemos continuar esse processo até que o resto r se torne 0. O último resto não-nulo será exatamente o Máximo Divisor Comum do dois números originais.

Desse modo, se aplicar-mos o **Algoritmo de Euclides** em Fib_m e Fib_n funciona da mesma maneira que se aplicar-mos aos índice m e n. E assim, ao chegarmos na base da recursão, $MDC(m,n) = MDC(s,0) = s$, teremos também: $MDC(Fib_m, Fib_n) = MDC(Fib_s, 0) = Fib_s = Fib_{MDC(m,n)}$ $\square$.

Teorema 13 $Fib_n = \frac{\sqrt{5}}{5}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n)$

Demonstração:

A demostrção asseguir foi baseada no no livro: *OLIVEIRA SANTOS, José Plínio de. Introdução à Teoria dos Números. IMPA, 1998. 85 p.*

$Fib_{n+1} = Fib_n + Fib_{n-1}$

$Fib_{n+1} - kFib_n = Fib_n + Fib_{n-1} - kFib_n$

$Fib_{n+1} - kFib_n = Fib_n + Fib_{n-1} - kFib_n + (kFib_{n-1} - kFib_{n-1}) + (k^2Fib_{n-1} - k^2Fib_{n-1})$

$Fib_{n+1} - kFib_n = (1-k)(Fib_n - kFib_{n-1}) + (1 + k - k^2)Fib_{n-1}$

Se denotarmos as raízes de $k^2 - k - 1 = 0$ por k_1 e k_2, teremos que $k_1 = \frac{1-\sqrt{5}}{2}$ e $k_2 = \frac{1+\sqrt{5}}{2}$.

$Fib_{n+1} - k_1Fib_b = k_2(Fib_n - k_1Fib_{n-1})$

$Fib_{n+1} - k_2Fib_b = k_1(Fib_n - k_2Fib_{n-1})$

Por iterações sucessivas dessas duas equações teremos que:

$Fib_{n+1} - k_1Fib_b = k_2^n(Fib_1 - k_1Fib_0) = k_2^n$

$Fib_{n+1} - k_2Fib_b = k_1^n(Fib_1 - k_2Fib_0) = k_1^n$

Subtraindo membro à membro nos dá:

$Fib_n(k_2 - k_1) = k_2^n - k_1^n$

$Fib_n = \frac{k_2^n - k_1^n}{k_2 - k_1}$

$Fib_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{(\frac{1+\sqrt{5}}{2}) - (\frac{1-\sqrt{5}}{2})}$

$Fib_n = \frac{\sqrt{5}}{5}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n)$ $\square$

3.3 Problemas Propostos

3.3.1 UVA-11424

11424 - GCD - Extreme (I)

Resumo: É dado um inteiro positivo N ($1 < N < 200001$). O problema consiste em calcular o mais rápido possível a expressão:

$G(N) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} MDC(i,j)$.

Solução: Trivialmente a expressão acima pode ser calculada em tempo proporcional à $O(n^2log(N))$, porém essa solução consome muito tempo e não será aceita no Judge Online. Vamos então mostrar uma solução mais eficiente.

Primeiramente reescrevemos a expressão acima da seguinte maneira:

$G(N) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} MDC(i,j)$ ($\triangleright$ Observe que as expressão são equivalentes).

Tome agora a função $F(M) = \sum_{i=1}^{M-1} MDC(i,M) \Rightarrow G(N) = \sum_{j=2}^{N} F(j)$.

Sabemos que todos os valores resultantes do método $MDC(i, M)$ calculados em $F(M)$ são divisores de M. Desse modo, podemos reescrever $F(M)$ da seguinte maneira:

$F(M) = \sum_{i=1}^{M-1} MDC(i, M) = \sum_{l=1}^{n} \lambda_l d_l$, em que, $d_1, d_2, ..., d_n$ são os divisores de M, λ_l é o número de vezes que o divisor d_l aparece na somatória $\sum_{i=1}^{M-1} MDC(i, N)$, e n é o número de divisores de M.

Pelo Corolario 8 temos que: $MDC(i, M) = d_l \Rightarrow MDC(i/d_l, M/d_l) = 1$. Logo o número de vezes que o divisor d_l aparece na somatória, será igual ao número de primos entre si com (M/d_l), ie, $\lambda_l = \Phi(M/d_l)$.

Reescrevendo novamente $F(M)$, temos:

$F(M) = \sum_{i=1}^{M-1} MDC(i, M) = \sum_{l=1}^{n} \lambda_l d_l = \sum_{l=1}^{n} \Phi(M/d_l) d_l$.

$G(N) = \sum_{j=2}^{N} \sum_{l=1}^{n} \Phi(j/d_l) d_l \ \square$.

Pseudocódigo:

Algorithm 7 GCD - Etreme(I)

```
1: procedure G (N)
2:     Φ[] ← PHI(N)
3:     solution ← 0
4:     for j := 2 to N do
5:         for each divisor d de j do
6:             solution ← solution + Φ[j/d]d
7:     return solution
```

Análise: O método $PHI(N)$ na linha 2 consome tempo proporcional à $O(N\sqrt{N})$.

O número de divisores de j é proporcional à $O(\sqrt{N})$, já que $j \leq N$.

Assim a complexidade das linhas 4, 5, 6 do algoritmo é $O(N\sqrt{N})$.

Complexidade final do algoritmo: $O(N\sqrt{N})$.

OBS.: Para resolver o problema no Judge Online será preciso armazenar as soluções usando Programação Dinâmica.

3.3.2 TJU-3506

3506 - Euler Function

Resumo: São dados três números positivos n, m ($1 < n < 10^7$, $1 < m < 10^9$) e $d = 201004$. O problema consiste em calcular a expressão: $\Phi(n^m) \bmod d$.

Solução: Pelo Teorema 8, temos:

$\Phi(n^m) \bmod d = (n^{m-1}\Phi(n)) \bmod d$

$\Phi(n^m) \bmod d = ((n^{m-1} \bmod d)(\Phi(n)) \bmod d) \bmod d)$

Desse modo, podemos calcular a primeiro fator do produto ($n^{m-1} \bmod d$) usando $EXPMOD()$ e a segundo fator com o método $PHI()$.

Pseudocódigo:

Análise: As linhas 3 e 4 do algoritmo consomem tempo proporcional à $O(\log m)$ e $O(1)$ respectivamente. Se precalcular-mos o vetor $\Phi[]$, temos que a complexidade total para calcular cada instância do problema será: $O(\log m)$

Algorithm 8 Euler Functions

1: **procedure** $PhiEulerPotential(n, m, d)$
2: $\Phi[] \leftarrow PHI(n)$
3: $exp \leftarrow EXPMOD(n, m-1, d)$
4: $solution \leftarrow (exp\ \Phi[n]) \bmod d$
5: **return** $solution$

3.3.3 CodeChef-IITK2P05

IITK2P05 - Factorization

Resumo: É dado um inteiro N ($2 \leq N \leq 10^{18}$) e o valora de $\Phi(N)$.
O problema consiste em fatorizar N.

Solução: A solução trivial para fatorar N consome tempo proporcional à $O(\sqrt{N})$ (veja X), porém para uma entrada na ordem de 10^{18} precisames de um algoritmo mais eficiente.

Se N for primo, ie, $\Phi(N) = N - 1$, já temos a solução.

Assuma então que N é um número composto. Primeiro vamos iterar nos primeiros $\sqrt[3]{N}$ inteiros e remover todos os fatores primos de N nesse intervalo. Tome M como sendo o valor resultante.

Imagine que M tenha três ou mais fatores primos. Sabemos que M não tem nenhum fator primo menor que $\sqrt[3]{N}$, temos que: $M > (\sqrt[3]{N})^3 \Rightarrow M > N$ (contradição). Desse modo, temos que M tem no máximo dois fatores primos, e esses valores são maiores que $\sqrt[3]{N}$.

- **Caso 1:** M tem só um fator primo, ie, M é primo: Basta checar se $\Phi(M) = M - 1$
- **Caso 2:** M tem dois fatores primos iguais, $M = p^2$: Basta verificar se M é um quadrado perfeito. Pode ser feito facilmente com busca binária.
- **Caso 3:** M tem dois fatores primos distintos, $M = pq$: Se $M = pq$ então $\Phi(M) = (p-1)(q-1)$. Temos então um sistema com duas equações e duas incógnitas. Se resolvermos o sistema encontraremos a fatoração de M e assim a fatoração de N.

O único problema agora é calcular $\Phi(M)$ a partir de $\Phi(N)$. Assuma que $N = p_1^{a_1} p_2^{a_2} ... p_k^{a_k} M$, com $k \geq 0$ e p_i os fatores primos distintos de N removidos na primeira etapa do algoritmo. Temos então:

$N = p_1^{a_1} p_2^{a_2} ... p_k^{a_k} M \Rightarrow \Phi(N) = \Phi(p_1^{a_1} p_2^{a_2} ... p_k^{a_k} M)$
$N = p_1^{a_1} p_2^{a_2} ... p_k^{a_k} M \Rightarrow \Phi(N) = \Phi(p_1^{a_1}) \Phi(p_2^{a_2}) ... \Phi(p_k^{a_k}) \Phi(M)$ (▷ Teorema 9)
$\Rightarrow \Phi(N) = (p_1^{a_1} - p_1^{a_1 - 1})(p_2^{a_2} - p_2^{a_2 - 1}) ... (p_k^{a_k} - p_k^{a_k - 1}) \Phi(M)$ (▷ Teorema 8)
$\Rightarrow \Phi(M) = \frac{\Phi(N)}{(p_1^{a_1} - p_1^{a_1 - 1})(p_2^{a_2} - p_2^{a_2 - 1}) ... (p_k^{a_k} - p_k^{a_k - 1})}$

Pseudocódigo:

Análise: O laço da linha 10 consome tempo proporcional à $O(\sqrt[3]{N})$. Já o laço da linha 11 consome tempo proporcional à $O(\log_p N)$, pois tem no máximo a iterações (a é o número de vezes que o fator primo p aparece em N) e assim, $p^a < N \Rightarrow a < \log_p N$.

As linhas 15-16 rodam em $O(log_p N)$, já que o número máximo de elementos distintos em S é $log_p N$ (p é o menor primo que divide N). E as linhas 18-23 rodam em $O(1)$.

Algorithm 9 Fatoração de N

procedure $Factorization(N, \Phi_N)$
  $S \leftarrow \emptyset$ ▷ S contém os fatores primos de N
  $M \leftarrow N$
  $\Phi_M \leftarrow \Phi_N$

  if $\Phi_N = N - 1$ **then** ▷ Se N for primo
    $S \leftarrow S \cup \{N\}$
    return S

  for each p primo menor igual à $\sqrt[3]{N}$ **do**
    while $M \equiv 0 (mod\, p)$ **do**
      $M \leftarrow \frac{M}{p}$
      $S \leftarrow S \cup \{p\}$

  for each $p \in S$ **do**
    $\Phi_M \leftarrow \frac{\Phi_m}{p^a - p^{a-1}}$ ▷ $a :=$ número de vezes que o primo p é inserido em S

  if $\Phi_M = M - 1$ **then** ▷ Se M for primo
    $S \leftarrow S \cup \{M\}$
    return S

  $(p, q) \leftarrow System(M, \Phi_M)$ ▷ Resolve o sistema de 2 equações e 2 incógnitas
  $S \leftarrow S \cup \{p, q\}$
  return S

Assim, o algoritmo total consome tempo proporcional à $O(\sqrt[3]{N} \log N)$. Obeserve que esse algoritmo é bem mais eficiente que o algoritmo trivial para fatoração $O(\sqrt{N})$.

3.3.4 CodeChef-MODEFB

71544 - Another Fibonacci

Resumo: São dados dois números inteiros N, K ($1 \leq N \leq 50000$, $1 \leq K \leq N$) e um conjunto $S \subset \mathbb{N}$ com N elementos, tal que, $\forall s \in S, 1 \leq s \leq 10^9$.
Tome a seguinte função:
$F(S) = \sum_{A \subset S \ e \ |A|=K} Fib(sum(A))$, onde $sum(A) = \sum_{a \in A} a$.
O problema consiste em calcular a expressão: $F(S) \bmod 99991$

Solução:

Pseudocódigo:

Algorithm 10 Another Fibonacci

1: **procedure** F (S)

Análise:

3.3.5 UVA-10311

10311 - Goldbach and Euler

Resumo: É dado um número inteiro n ($0 < n \leq 10^8$). O problema consite em verificar se n pode, ou não pode, ser escrito como a soma de dois números primos. E em caso afirmativo encontrar o valor desses dois primos.

Solução:

Pseudocódigo:

Algorithm 11 Goldbach and Euler

1: **procedure** FINDTWOPRIMESSUM (N)

Análise:

3.3.6 Codeforces-227E

227E - Anniversary

Resumo:

Solução:

Pseudocódigo:
Análise:

Algorithm 12 Anniversary

1: **procedure** FINDTWOPRIMESSUM (N)

Capítulo 4

Conclusão

...

Apêndice A

Curiosidades da ACM-ICPC

ACM-ICPC (International Collegiate Programming Contest) é uma competição de programação de várias etapas e baseada em equipe. O principal objetivo é encontrar algoritmos eficientes, que resolvem os problemas abordados pela competição, o mais rápido possível.

Nos últimos anos a ACM-ICPC teve um crescimento significativo. Se compararmos o número de competidores, temos que de 1997 (ano em que começou o patrocinio da IBM) até 2014 houve um aumento maior que 1500%, totalizando 38160 competidores de 2534 universidades em 101 países ao redor do mundo.

Para mais informações sobre as competições passadas acesse icpc.baylor.edu.

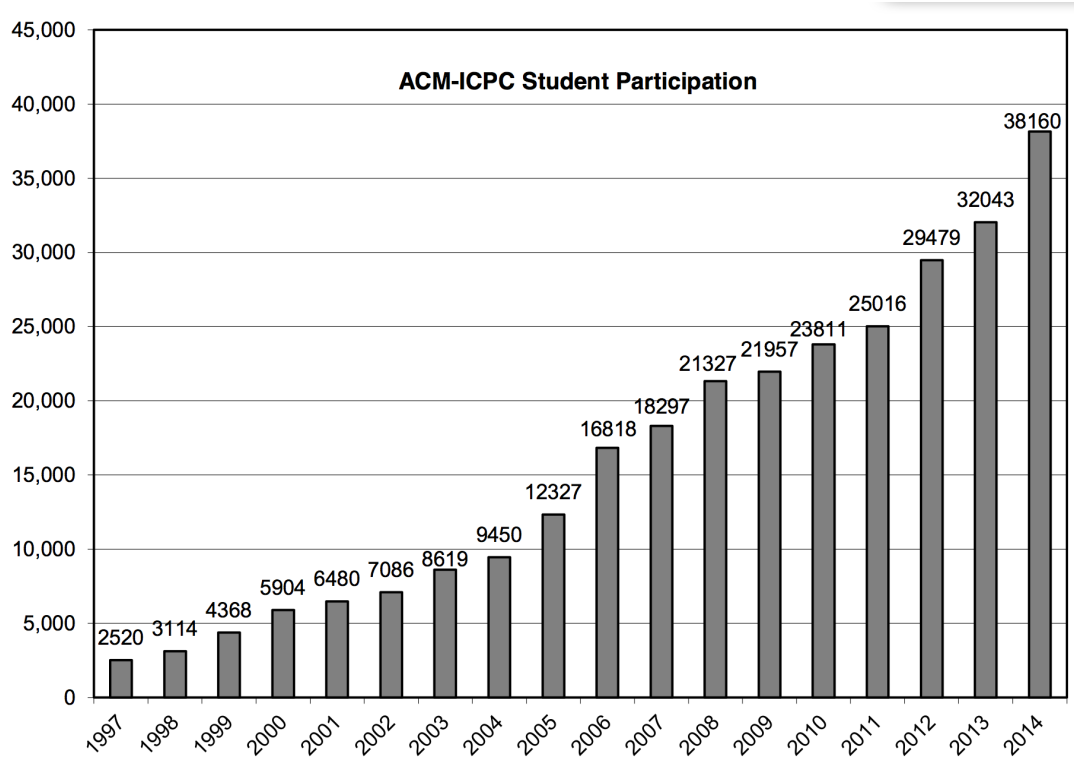

FIGURA A.1: Crescimento do número de participantes por ano.

Apêndice B

Juízes Online (Online Judges)

Online Judges são plataformas online que contam com um banco de dados com diversos tipos de problemas de competições de programação, e com um sistema de correção online.

Para afirmar que sua solução está correta, basta enviar o código fonte da sua solução (em geral escrito em C++ ou JAVA) para uma dessas plataformas.

Alguns desses Online Judges são citados em seguida.

B.1 UVa

Criado em 1995 pelo matemático Miguel Ángel Revilla, é atualmente um dos Online Judges mais famoso entre os participantes da ACM-ICPC.

É hospedado pela Universidade de Valhadolide e conta com mais de 100000 usuários registrados.

Site: https://uva.onlinejudge.org/

B.2 Topcoder

Empresa que administra competições de programação nas linguagens Java, C++ e C#.

É responsável também por aplicar competições de design e desenvolvimento de software.

Site: https://www.topcoder.com/

B.3 Codeforces

Site Russo dedicado competições de programação.

Em 2013, Codeforces superou Topcoder com relação ao número de usuários ativos, apesar de ter sido criado quase 10 anos depois.

O estilo de problemas que esse site aplica é similar aos problemas encontrados na ACM-ICPC.

Site: http://codeforces.com/

B.4 CodeChef

Iniciativa educacional sem fins lucrativos lançada em 2009 pela Direct.

É uma plataforma de progamação competitiva que suporta mais de 35 linguagens de programação.

Site: https://www.codechef.com/

Bibliografia

Arnold, A. S. et al. (1998). "A Simple Extended-Cavity Diode Laser". Em: *Review of Scientific Instruments* 69.3, pp. 1236–1239. URL: http://link.aip.org/link/?RSI/69/1236/1.

Hawthorn, C. J., K. P. Weber e R. E. Scholten (2001). "Littrow Configuration Tunable External Cavity Diode Laser with Fixed Direction Output Beam". Em: *Review of Scientific Instruments* 72.12, pp. 4477–4479. URL: http://link.aip.org/link/?RSI/72/4477/1.

Wieman, Carl E. e Leo Hollberg (1991). "Using Diode Lasers for Atomic Physics". Em: *Review of Scientific Instruments* 62.1, pp. 1–20. URL: http://link.aip.org/link/?RSI/62/1/1.