Introdução à Teoria dos Números

Laboratório de Programação Competitiva I

Pedro Henrique Paiola

Rene Pegoraro

Wilson M Yonezawa

Arissa Yoshida

Nicolas Barbosa Gomes

Luis Henrique Morelli

BigInteger

- Certos problemas da Maratona de Programação recebem como entrada números inteiros que extrapolam o limite de variáveis do tipo **long long int**
- Tamanho de uma variável long long int: **8 bytes**
- Intervalo de números que podem ser armazenados em uma variável desse tipo:
  - -9.223.372.036.854.775.808 à 9.223.372.036.854.775.807
  - 0 à 18.446.744.073.709.551.615 (unsigned long long int)

} 10^{18}

BigInteger

- Certos problemas da Maratona de Programação recebem como entrada números inteiros que extrapolam o limite de variáveis do tipo **long long int**
- Exemplo: 2667 - Jogo de Boca
  - Entrada: $N\ (3 \leq N \leq 10^{100})$

BigInteger

- **1ª Situação**: dependendo das operações necessárias de se fazer com o número, podemos ler o número como sendo uma **string** e trabalhar com essa string.

- Exemplos:
  - Operações simples com dígitos
  - Uso de Aritmética Modular

BigInteger

- **2ª Situação**: se precisarmos fazer operações com esse número **como soma, subtração, multiplicação e divisão**, o problema se torna mais complexo.

- Nesses casos, não recomendamos usar a linguagem C++. É possível trabalhar com BigInteger em C++ (a biblioteca do Thiago traz códigos para isso), porém a quantidade de código necessária é relativamente grande.
- Sugestões: **Java** ou **Python**

BigInteger

- Em Java podemos usar a classe BigInteger da biblioteca java.math

```
String Num;
BigInteger NumGrande;
Scanner S = new Scanner(System.in);
Num = S.nextLine();

NumGrande = new BigInteger(Num);
NumGrande = NumGrande.mod(new BigInteger("3"));

System.out.println(NumGrande);
```

BigInteger

- Em Python, não precisamos nos preocupar muito com o tamanho de um inteiro, a memória é alocada conforme o necessário para comportar o tamanho do número.

Entrada e Saída em Python

Python em Programação Competitiva

Muita coisa sobre Python

BigInteger

- Em Python, não precisamos nos preocupar muito com o tamanho de um inteiro, a memória é alocada conforme o necessário para comportar o tamanho do número.

```
U = int(input())
print(U % 3)
```

Teoria dos Números

- A Teoria dos Números é o ramo da matemática que se preocupa com as propriedades dos **números inteiros**.
- Existe uma coleção de algoritmos interessantes derivados de estudos da Teoria dos Números que solucionam problemas de forma inteligente e eficiente.
- Aqui faremos uma breve introdução à alguns tópicos relativos à Teoria dos Números.

Números primos

- Diversos problemas envolvem o uso de números primos.
- Dessa forma, precisamos, inicialmente, de uma forma de testar se um número é primo ou não.

- Recordando: números primos são **números naturais** que têm **apenas dois divisores**: 1 e ele mesmo.

Números primos

- Algoritmo ingênuo $O(n)$

```
bool ehPrimo(int n)
{
    for(int i = 2; i < n; i++)
        if (n % i == 0)
            return false;
    return true;
}
```

Números primos

- Porém, na verdade só precisamos testar até $\sqrt{n}$
- **Demonstração**:

Suponha que não, nesse caso existe n tal que o menor fator primo p de n é maior que $\sqrt{n}$.

Se p divide n, então $\frac{n}{p}$ também divide n, e $\frac{n}{p}$ deve ser maior que $\sqrt{n}$.

Mas se $p > \sqrt{n}$ e $\frac{n}{p} > \sqrt{n}$, então $p . \frac{n}{p} > n$, o que é um absurdo! ■

Números primos

- Algoritmo $O(\sqrt{n})$

```
bool ehPrimo(int n)
{
    int raiz = sqrt(n);
    for(int i = 2; i <= raiz; i++)
        if (n % i == 0)
            return false;
    return true;
}
```

Números primos

- Algoritmo $O(\sqrt{n})$

```
bool ehPrimo(int n)
{
    for(int i = 2; i*i <= n; i++)
        if (n % i == 0)
            return false;
    return true;
}
```

Crivo de Eratóstenes

- O Crivo de Eratóstenes é um método de encontrar os números primos até um certo valor limite.
- Útil em casos que faremos vários testes de primalidade e na fatoração de números.
- Ideia geral: dado que um número p é primo, marcamos os múltiplos de p como não sendo números primos.

Crivo de Eratóstenes

- Algoritmo:
  - Cria-se uma lista de 2 a MAX, marcando todos como primos
  - Para cada número i de 2 até $\sqrt{MAX}$
    - Se i está marcado como primo
      - Marcar todos os números múltiplos de i a partir de $i.i$ como compostos (não primos)

Crivo de Eratóstenes

- Algoritmo:
  - Cria-se uma lista de 2 a MAX, marcando todos como primos
  - Para cada número i de 2 até $\sqrt{MAX}$
    - Se i está marcado como primo
      - Marcar todos os números múltiplos de i a partir de $i.i$ como compostos (não primos)

Por que podemos marcar só a partir de $i.i$?

Crivo de Eratóstenes

- Antes de $\boldsymbol{i.i}$ temos: $i.2,\ i.3,\ i.4, \ldots\ i.(i-1)$. Ou ainda, $\boldsymbol{i.j} \mid \boldsymbol{2 \leq j < i}$
- Seja $\boldsymbol{x = i.j}$, $\boldsymbol{x}$ é **múltiplo de $\boldsymbol{i}$** e também é **múltiplo de $\boldsymbol{j}$**

Crivo de Eratóstenes

- Antes de $\boldsymbol{i}.\boldsymbol{i}$ temos: $i.2,\ i.3,\ i.4, \ldots\ i.(i-1)$. Ou ainda, $\boldsymbol{i}.\boldsymbol{j} \mid \mathbf{2} \leq \boldsymbol{j} < \boldsymbol{i}$
- Seja $\boldsymbol{x} = \boldsymbol{i}.\boldsymbol{j}$, $\boldsymbol{x}$ é **múltiplo de** $\boldsymbol{i}$ e também é **múltiplo de** $\boldsymbol{j}$
- Todo $\boldsymbol{j}$ ou é primo, ou é múltiplo de um número primo menor que $\boldsymbol{i}$, ou seja, um primo já "descoberto" pelo algoritmo

Crivo de Eratóstenes

- Antes de $i.i$ temos: $i.2,\ i.3,\ i.4, \ldots\ i.(i-1)$. Ou ainda, $i.j \mid 2 \leq j < i$
- Seja $x = i.j$, x é **múltiplo de i** e também é **múltiplo de j**
- Todo j ou é primo, ou é múltiplo de um número primo menor que i, ou seja, um primo já "descoberto" pelo algoritmo
- Se j é primo
  - Todos os seus múltiplos foram marcados como não primo, inclusive $i.j$

Crivo de Eratóstenes

- Antes de $\boldsymbol{i.i}$ temos: $i.2,\ i.3,\ i.4, \ldots\ i.(i-1)$. Ou ainda, $\boldsymbol{i.j} \mid \boldsymbol{2 \leq j < i}$
- Seja $\boldsymbol{x = i.j}$, $\boldsymbol{x}$ é **múltiplo de $\boldsymbol{i}$** e também é **múltiplo de $\boldsymbol{j}$**
- Todo $\boldsymbol{j}$ ou é primo, ou é múltiplo de um número primo menor que $\boldsymbol{i}$, ou seja, um primo já "descoberto" pelo algoritmo
- Se $\boldsymbol{j}$ é primo
  - Todos os seus múltiplos foram marcados como não primo, inclusive $\boldsymbol{i.j}$
- Se $\boldsymbol{j}$ é múltiplo de um primo $\boldsymbol{p < i}$
  - Então ele já foi marcado como composto, por ser múltiplo de $\boldsymbol{p}$, assim como todos os seus múltiplos

Crivo de Eratóstenes

- Antes de $i.i$ temos: $i.2,\ i.3,\ i.4, \ldots\ i.(i-1)$. Ou ainda, $i.j \mid 2 \leq j < i$
- Seja $x = i.j$, x é **múltiplo de i** e também é **múltiplo de j**
- Todo j ou é primo, ou é múltiplo de um número primo menor que i, ou seja, um primo já "descoberto" pelo algoritmo
- Se j é primo
  - Todos os seus múltiplos foram marcados como não primo, inclusive $i.j$
- Se j é múltiplo de um primo $p < i$
  - Então ele já foi marcado como composto, por ser múltiplo de p, assim como todos os seus múltiplos
- Logo, todos os números $i.j \mid 2 \leq j < i$ já foram marcados

Crivo de Eratóstenes

```
bool ehPrimo[MAX];
vector<int> primos;

void crivo(int n){
    memset(ehPrimo, true, sizeof(ehPrimo));
    for(int p = 2; p * p <= n; p++){
        if (ehPrimo[p]){
            primos.push_back(p); //Lista incompleta, primos até sqrt(n)
            for(int i = p*p; i <= n; i += p)
                ehPrimo[i] = false;
        }
    }
}
```

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

Números primos:

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

Números primos:
2

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

Números primos:
2 3

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

Números primos:
2 3 5

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

Números primos:

2 3 5 7

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

Números primos:

2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97
101	103	107	109	113

Crivo de Eratóstenes

- Este algoritmo possui complexidade $O(n \log \log n)$
- Esta demonstração não é muito simples. Caso queira conferir, veja o artigo do CP-Algorithms.
- Com certas otimizações ainda é possível obter um algoritmo de complexidade linear.

Fatoração

- Fatoração em $O(\sqrt{n})$

```
vector<int> fatorar(int n) {
    vector<int> fator;
    for (int i = 2; i*i <= n; i++){
        while (n % i == 0){
            fator.push_back(i);
            n /= i;
        }
    }
    if (n > 1)
        fator.push_back(n);
    return fator;
}
```

Fatoração

- Também é possível obter um algoritmo de fatoração com complexidade $O(\log n)$, baseando-se no Crivo de Eratóstenes.
- Primeiramente, ao invés de utilizarmos o crivo para descobrirmos todos os primos, faremos uma pequena alteração para computar para cada número o seu Menor Fator Primo (*Shortest Prime Factor* - SPF).

Fatoração

- Crivo para Fatoração

```
int spf[MAXN];

void crivo(){
    for(int i=2; i < MAXN; i++){
        if(spf[i] == 0){
            spf[i] = i;
            for(int j=i*i; j<MAXN; j+=i){
                if(spf[j] == 0) spf[j] = i;
            }
        }
    }
}
```

Fatoração

- A partir do vetor SPF pré-calculado, podemos realizar a fatoração de um número qualquer seguindo o seguinte algoritmo:
  - `fatores = []`
  - `enquanto n > 1`
    - `Inserir spf[n] em fatores`
    - `n = n/spf[n]`

Fatoração

- Fatoração em $O(\log n)$

```
vector<int> fatorar(int n){
    vector<int> fator;
    while(n > 1){
        fator.push_back(spf[n]);
        n /= spf[n];
    }
    return fator;
}
```

Look-up tables

- Existem casos onde podemos gerar um **vetor ou matriz de consulta** manualmente (ou previamente por outro programa), e inseri-los prontos no nosso código. Dessa forma, economiza-se o tempo de gerar tal vetor/matriz.
- Por exemplo, se para resolver um problema precisamos de todos os primos até N, podemos embutir um vetor de primos já dentro do código.

```
int primos[] = {2, 3, 5, 7, 11, 13, … }
```

- Isso também pode ser gerado por um programa auxiliar.

Look-up tables

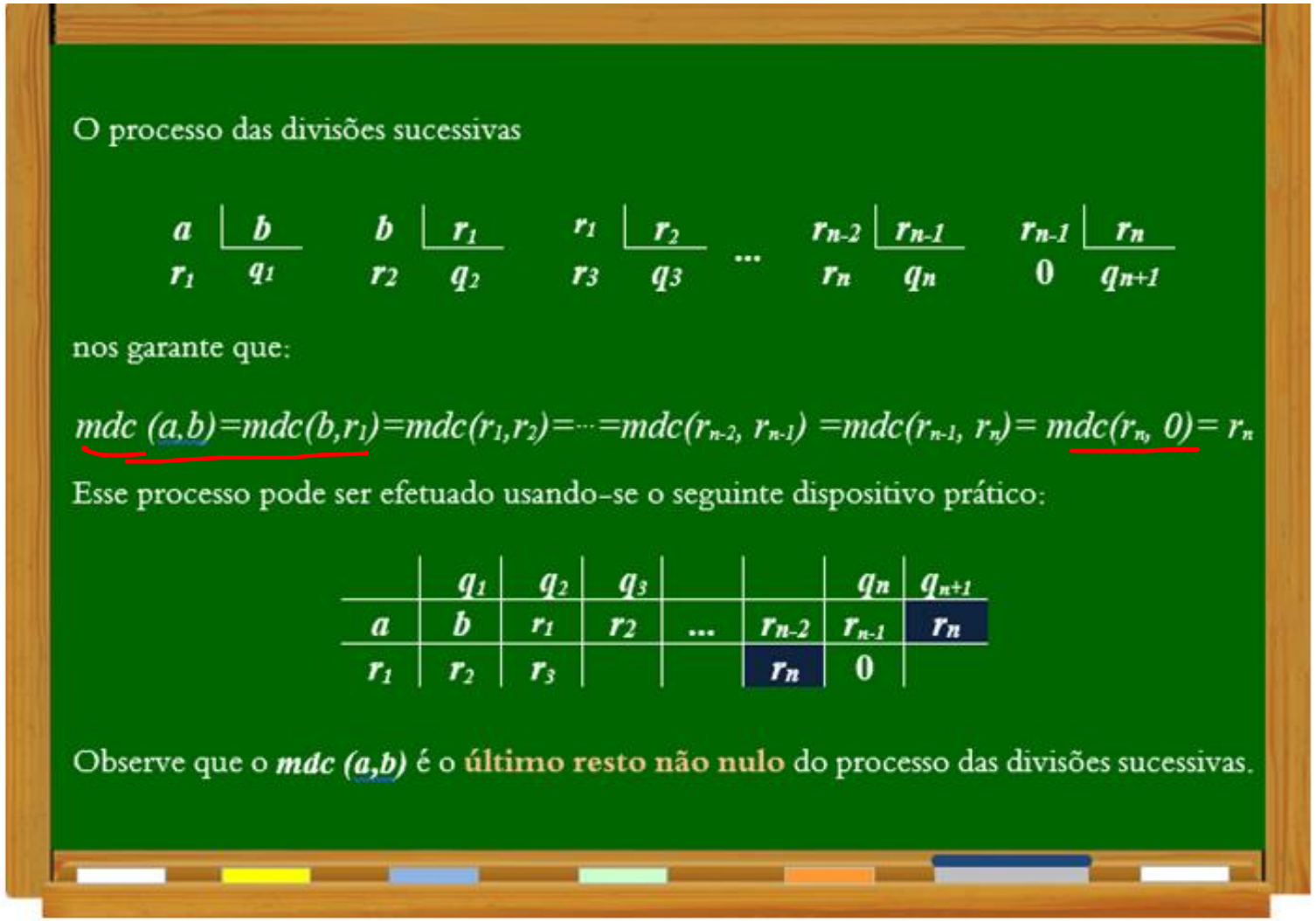
"The judge can't look into your heart or your program to see your intentions - it only checks the results."
(Skiena & Revilla, 2003; p. 129)

Máximo Divisor Comum

- Problema: encontrar o maior divisor comum de um par de números.
- Algoritmo de Euclides

- OBS: se $mdc(x, y) = 1$, então dizemos que x e y são coprimos ou primos entre si.

Máximo Divisor Comum



O processo das divisões sucessivas

$$\begin{array}{c|c} a & b \\ r_1 & q_1 \end{array} \quad \begin{array}{c|c} b & r_1 \\ r_2 & q_2 \end{array} \quad \begin{array}{c|c} r_1 & r_2 \\ r_3 & q_3 \end{array} \quad \cdots \quad \begin{array}{c|c} r_{n-2} & r_{n-1} \\ r_n & q_n \end{array} \quad \begin{array}{c|c} r_{n-1} & r_n \\ 0 & q_{n+1} \end{array}$$

nos garante que:

$$mdc\,(a,b) = mdc(b,r_1) = mdc(r_1,r_2) = \cdots = mdc(r_{n-2}, r_{n-1}) = mdc(r_{n-1}, r_n) = mdc(r_n, 0) = r_n$$

Esse processo pode ser efetuado usando-se o seguinte dispositivo prático:

	q_1	q_2	q_3			q_n	q_{n+1}
a	b	r_1	r_2	...	r_{n-2}	r_{n-1}	r_n
r_1	r_2	r_3			r_n	0	

Observe que o ***mdc (a,b)*** é o **último resto não nulo** do processo das divisões sucessivas.

Máximo Divisor Comum

- MDC/GCD em $O(\log(a+b))$

```
int gcd(int a, int b){
    if (a == 0)
        return b;
    return gcd(b % a, a);
}
```

Mínimo Múltiplo Comum

- Problema: encontrar o menor múltiplo comum entre um par de inteiros.
- Para encontrar o mmc(x, y), podemos calcular o mdc(x, y) e utilizar a seguinte fórmula:

$$mmc(x, y) * mdc(x, y) = x * y$$

Ou seja:

$$mmc(x, y) = x * y / mdc(x, y)$$

Mínimo Múltiplo Comum

- MMC/LCM em $O(\log(a+b))$

```
int lcm(int a, int b){
    return a * (b / gcd(a, b));
}
```

Equações diofantinas

- Podemos definir uma equação diofantina linear como uma equação da forma

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = c$$

sendo $a_1, \ldots, a_n$ coeficientes inteiros não nulos, $x_1 \ldots, x_n$ as **variáveis inteiras** a serem determinadas e c uma constante inteira.

- Diversos problemas podem ser modelados usando equações diofantinas. Em especial, vamos nos preocupar com equações diofantinas de duas variáveis

$$ax + by = c$$

Equações diofantinas – Fantastic Beasts

- Exemplo de Problema: Fantastic Beasts (Final da Maratona SBC de Programação - 2018)

- Resumindo: considere um **grafo direcionado** em que os vértices representam zoológicos, e cada zoológico aponta para apenas para um outro zoológico (grau de saída = 1). Temos animais espalhados por esses zoológicos, e **a cada unidade de tempo todos os animais avançam para o próximo** zoológico.
- Objetivo: determinar **onde e quando TODOS os animais se encontrarão,** no mesmo zoológico ao mesmo tempo (se isso puder ocorrer em diversos momentos e locais, determinar o primeiro deles)

Equações diofantinas – Fantastic Beasts

- Supondo que já estamos em uma fase um pouco mais avançada no problema, onde conseguimos modelar para cada **zoológico z** uma **equação** que determina os momentos em que um animal a passa por lá (os animais vão acabar presos em ciclos).

$$t = t_0 + k.i$$

Equações diofantinas – Fantastic Beasts

- Se quisermos saber quando que os animais a_1 e a_2 se encontram no zoológico z, temos:

Para animal a_1: $t_1 = t_0^1 + k_1 i$

Para animal a_2: $t_2 = t_0^2 + k_2 j$

Como queremos saber o momento de encontro, $t_1 = t_2$

$$t_0^1 + k_1 i = t_0^2 + k_2 j$$

$$\boldsymbol{k_1 i - k_2 j = t_0^2 - t_0^1}$$

$$ai + bj = c$$

Equação diofantina com $a = k_1, b = -k_2, c = t_0^2 - t_0^1$ e com variáveis i, j

Equações diofantinas

- Proposição 1: $\boldsymbol{ax + by = c}$ admite solução sse $\boldsymbol{gcd(a, b) \mid c}$

  $\Rightarrow$

  - Sendo (x_0, y_0) uma solução da equação
  - Seja $d = \gcd(a, b)$, então $d|a$ e $d|b$. Logo podemos reescrever $a = Ad$ e $b = Bd$

Equações diofantinas

- Proposição 1: $\boldsymbol{ax + by = c}$ admite solução sse $\boldsymbol{gcd(a, b) \mid c}$

  $\Rightarrow$

  - Sendo (x_0, y_0) uma solução da equação
  - Seja $d = \gcd(a, b)$, então $d|a$ e $d|b$. Logo podemos reescrever $a = Ad$ e $b = Bd$

$$c = ax_0 + by_0 = (Ad)x_0 + (Bd)y_0$$
$$c = d(Ax_0 + By_0)$$

Equações diofantinas

- Proposição 1: $\boldsymbol{ax + by = c}$ admite solução sse $\boldsymbol{gcd(a, b) \mid c}$

  $\Rightarrow$

  - Sendo (x_0, y_0) uma solução da equação
  - Seja $d = \gcd(a, b)$, então $d|a$ e $d|b$. Logo podemos reescrever $a = Ad$ e $b = Bd$

$$c = ax_0 + by_0 = (Ad)x_0 + (Bd)y_0$$
$$c = d(Ax_0 + By_0)$$

Denotando $q = Ax_0 + By_0$

$$c = dq$$

Portanto, $d|c$

Equações diofantinas

- Proposição 1: $\boldsymbol{ax + by = c}$ admite solução sse $\boldsymbol{gcd(a, b) \mid c}$

  $\Leftarrow$

  - Seja $d = \gcd(a, b)$
  - Pelo Teorema de Bézout, existe solução (x_0, y_0) para $ax + by = d$
  - Por hipótese, $d|c \Rightarrow \exists t \,/\, c = dt$

Equações diofantinas

- Proposição 1: $\boldsymbol{ax + by = c}$ admite solução sse $\boldsymbol{gcd(a, b) \mid c}$

  $\Leftarrow$

  - Seja $d = \gcd(a, b)$
  - Pelo Teorema de Bézout, existe solução (x_0, y_0) para $ax + by = d$
  - Por hipótese, $d|c \Rightarrow \exists t \,/\, c = dt$

$$c = dt$$
$$c = (ax_0 + by_0)t$$
$$c = a(x_0 t) + b(y_0 t)$$

Portanto, se $d \mid c$, então a equação $ax + by = c$ admite solução ■

Equações diofantinas

- Como determinar uma solução?
  1. Obter uma solução (x_0, y_0) para $ax + by = \gcd(a, b)$
  2. Para $ax + by = c$:
     - *a)* $t = c/d$ em que $d = \gcd(a, b)$
     - *b)* $x = x_0 t$
     - *c)* $y = y_0 t$
  3. Se uma equação diofantina tem uma solução, então ela tem infinitas:

$$S = \left\{ \left(x + \frac{b}{d}k, y - \frac{a}{d}k \right), k \in \mathbb{Z} \right\}$$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$

1. Caso base ($a = 0$):
   - Se $a = 0$ então temos $by = \gcd(0, b)$
   - Sabemos que $\gcd(0, b) = b$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$

1. Caso base ($a = 0$):
   - Se $a = 0$ então temos $by = \gcd(0, b)$
   - Sabemos que $\gcd(0, b) = b$
   - Então $by = b$, logo $y = 1$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$

1. Caso base ($a = 0$):
   - Se $a = 0$ então temos $by = \gcd(0, b)$
   - Sabemos que $\gcd(0, b) = b$
   - Então $by = b$, logo $y = 1$
   - Nesse caso x pode assumir qualquer valor. Como queremos uma solução qualquer, por motivos de simplificação, faremos $x = 0$

   - Solução base: $(0, 1)$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$

2. Passo da indução:
   - Temos $ax + by = \gcd(a, b)$
   - Pelo Algoritmo de Euclides, sabemos que
     $\gcd(a, b) = \gcd(b\%a, a) = d$
   - Logo, podemos obter outra equação diofantina:
     $(b\%a)x_1 + ay_1 = d \qquad (*)$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$
  - Considerando o resultado da divisão inteira, podemos dizer que:

$$b = \frac{b}{a}a + b\%a$$

$$b\%a = b - \frac{b}{a}a$$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$
  - Considerando o resultado da divisão inteira, podemos dizer que:

$$b = \frac{b}{a}a + b\%a$$

$$b\%a = b - \frac{b}{a}a$$

  - Substituindo em $(*)$

$$\left(b - \frac{b}{a}a\right)x_1 + ay_1 = d$$

$$bx_1 - \frac{b}{a}ax_1 + ay_1 = d$$

$$a\left(y_1 - \frac{b}{a}x_1\right) + bx_1 = d$$

Equações diofantinas

- Solução para $ax + by = \gcd(a, b)$
  - Considerando o resultado da divisão inteira, podemos dizer que:

$$b = \frac{b}{a}a + b\%a$$

$$b\%a = b - \frac{b}{a}a$$

  - Substituindo em $(*)$

$$\left(b - \frac{b}{a}a\right)x_1 + ay_1 = d$$

$$bx_1 - \frac{b}{a}ax_1 + ay_1 = d$$

$$a\underbrace{\left(y_1 - \frac{b}{a}x_1\right)}_{\boldsymbol{ax}} + \underbrace{bx_1}_{\boldsymbol{by}} = \boldsymbol{d}$$

■

Equações diofantinas

- Implementação

```
int gcd(int a, int b, int &x, int &y){
    if (a == 0){
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(a, b % a, x1, y1);
    x = y1 - x1 * (b/a);
    y = x1;
    return d;
}
```

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

Equações diofantinas

- Implementação

```
bool solve(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

Aritmética Modular

- Em vários problemas precisamos operar com os restos de divisões de inteiros.
- A aritmética modular permite fazer cálculos com restos de divisões de modo eficiente, e é especialmente útil quando estamos trabalhando com números grandes (BigInteger).
- Na verdade, a Aritmética Modular pode nos ajudar a evitar ter que trabalhar com números muito grandes.

Aritmética Modular

- A aritmética modular se baseia nas seguintes propriedades:

$$(x + y) \,\%\, n = ((x \,\%\, n) + (y \,\%\, n)) \,\%\, n$$

$$(x - y) \,\%\, n = ((x \,\%\, n) - (y \,\%\, n)) \,\%\, n$$

$$(x * y) \,\%\, n = ((x \,\%\, n) * (y \,\%\, n)) \,\%\, n$$

$$(x \,\hat{}\, y) \,\%\, n = ((x \,\%\, n) \,\hat{}\, y) \,\%\, n$$

Aritmética Modular

- UVa 374 - Big Mod
  - Calcule $R = B^P \ mod \ M$
  - $0 \leq B, P \leq 2147483647$ e $1 \leq M \leq 46340$

Aritmética Modular

- Parte da solução do problema UVA 374 – Big Mod

```
long long pow(long long x, long long y, long long mod) {
    if (y == 0)
        return 1;

    long long p = pow(x, y/2, mod);
    if (y % 2 == 0)
        return (p * p) % mod;
    else
        return (((p * p) % mod) * (x % mod)) % mod;
}
```

Inverso Modular

- A aritmética modular não se aplica a divisão. Porém, temos o conceito de inverso multiplicativo modular.
- Lembre-se que um número multiplicado pelo seu **inverso** é igual a 1

Inverso Modular

- A aritmética modular não se aplica a divisão. Porém, temos o conceito de inverso multiplicativo modular.
- Lembre-se que um número multiplicado pelo seu **inverso** é igual a 1

- Da aritmética básica, sabemos que:
  - O inverso de um número A é $\frac{1}{A}$
  - Todos os reais diferentes de 0 têm um inverso
  - Multiplicar um número pelo inverso de A é o mesmo que dividir por A
    - $X * A^{-1} = X * \frac{1}{A} = \frac{X}{A}$

Inverso Modular

- O **inverso modular** de $A\ (mod\ C)$ é A^{-1}.
  - $(A\ *A^{-1}) \equiv 1\ (mod\ C)$ ou de modo equivalente $(A\ *A^{-1})\ mod\ C = 1$

- OBS: Apenas os números coprimos de C têm um inverso modular (mod C)

Inverso Modular

- O **inverso modular** de $A\ (mod\ C)$ é A^{-1}.
  - $(A\ *A^{-1}) \equiv 1\ (mod\ C)$ ou de modo equivalente $(A\ *A^{-1})\ mod\ C = 1$

- Exemplo: A=3 e C=7

$$(3 * 5)\ mod\ 7 = 15\ mod\ 7 = 1$$

$$\therefore (3\ *5) \equiv 1\ (mod\ 7)$$

- Logo, 5 é o inverso modular de 3 (mod 7) .

Inverso Modular

- Como encontrar um inverso multiplicativo?
  - Determinar um $x \in \mathbb{Z}$ tal que $Ax \equiv 1 \ (mod\ C) \Rightarrow x = A^{-1}$

Inverso Modular

- Como encontrar um inverso multiplicativo?
  - Determinar um $x \in \mathbb{Z}$ tal que $Ax \equiv 1 \ (mod\ C) \Rightarrow x = A^{-1}$
  - Da congruência, temos que
    - $C \mid (Ax - 1)$
    - Logo, $\exists y \in \mathbb{Z} \mid Ax - 1 = Cy$

$$Ax - Cy = 1$$

    - Equação diofantina!

Inverso Modular

- Maratona SBC de Programação – Regional 2021 – C – Criando Múltiplos

$$(D_1 \cdot B^{L-1} + D_2 \cdot B^{L-2} + \ldots + D_L \cdot B^0) \,\%\, (B+1) = 0$$

Queremos substituir algum D_i por $x < D_i$ tal que →

Inverso Modular

- Maratona SBC de Programação – Regional 2021 – C – Criando Múltiplos

$$(D_1 \cdot B^{L-1} + D_2 \cdot B^{L-2} + \ldots + D_L \cdot B^0) \,\%\, (B+1) = 0$$

Queremos substituir algum D_i por $x < D_i$ tal que

$$S - D_i \cdot B^{L-i} + x \cdot B^{L-i} \equiv 0 \pmod{B+1}$$

$$x \cdot B^y \equiv D_i B^y - S$$

$$x \equiv \frac{D_i B^y}{B^y} - \frac{S}{B^y}$$

$$x \equiv D_i - S \cdot B^{-y}$$

inverso multiplicativo

$$n \cdot n^{-1} \equiv 1 \pmod{m}$$

Referências

Biblioteca de códigos de Thiago Alexandre Domingues de Souza.

Matemática Discreta e Suas Aplicações. Kenneth H. Rosen.

Programming Challenges: The Programming Contest Training Manual. Stevem S. Skiena e Miguel A. Revilla.

https://www.geeksforgeeks.org/sieve-of-eratosthenes/

http://www.lcad.icmc.usp.br/~jbatista/scc210/AulaTeoriadosNumeros1.pdf

http://www.lcad.icmc.usp.br/~jbatista/scc210/AulaTeoriadosNumeros2.pdf

https://www.ufsj.edu.br/portal2-repositorio/File/comat/tcc_Ricardo.pdf

https://cp-algorithms.com/algebra/linear-diophantine-equation.html

https://noic.com.br/materiais-informatica/curso/math-02/

https://noic.com.br/materiais-informatica/curso/math-03/

Referências

https://pt.khanacademy.org/computing/computer-science/cryptography/modarithmetic/pi/fast-modular-exponentiation

https://www.cin.ufpe.br/~gdcc/matdis/aulas/aritmeticaModular_parte2.pdf