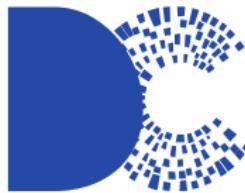




UNIVERSIDADE
FEDERAL DE
SERGIPE



DEPARTAMENTO
DE COMPUTAÇÃO

Algoritmos numéricos

Projeto e Análise de Algoritmos

Bruno Prado

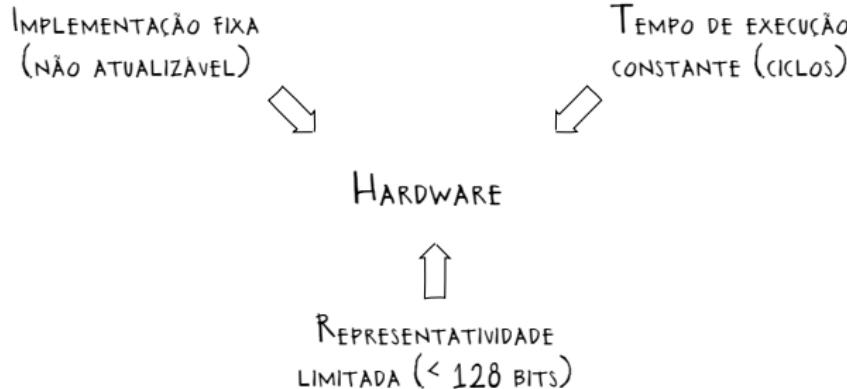
Departamento de Computação / UFS

Introdução

- ▶ Por que são utilizados os algoritmos numéricos?
 - ▶ Para realização de operações aritméticas básicas de adição, subtração, multiplicação e divisão, além de funções mais complexas como exponenciação, máximo divisor comum ou teste de primalidade

Introdução

- ▶ Por que são utilizados os algoritmos numéricos?
 - ▶ Para realização de operações aritméticas básicas de adição, subtração, multiplicação e divisão, além de funções mais complexas como exponenciação, máximo divisor comum ou teste de primalidade



Introdução

- ▶ Por que são utilizados os algoritmos numéricos?
 - ▶ Para realização de operações aritméticas básicas de adição, subtração, multiplicação e divisão, além de funções mais complexas como exponenciação, máximo divisor comum ou teste de primalidade

IMPLEMENTAÇÃO FLEXÍVEL
(COM ATUALIZAÇÕES)



TEMPO DE EXECUÇÃO
DEPENDENTE DA ENTRADA



SOFTWARE



REPRESENTATIVIDADE
ARBITRÁRIA (< MEMÓRIA)

Introdução

- ▶ Notação numérica posicional
 - ▶ É definida pelos dígitos d_i e pela base b adotada para representação dos números

$$d_{n-1}d_{n-2}\dots d_1d_0_b$$

Introdução

- ▶ Notação numérica posicional
 - ▶ É definida pelos dígitos d_i e pela base b adotada para representação dos números

$$\begin{aligned} & d_{n-1}d_{n-2}\dots d_1d_0{}_b \\ & = \\ & d_{n-1} \times b^{n-1} + d_{n-2} \times b^{n-2} + \dots + d_1 \times b^1 + d_0 \times b^0 \end{aligned}$$

Introdução

- ▶ Notação numérica posicional
 - ▶ É definida pelos dígitos d_i e pela base b adotada para representação dos números

$$\begin{aligned} & d_{n-1}d_{n-2}\dots d_1d_0{}_b \\ & = \\ & d_{n-1} \times b^{n-1} + d_{n-2} \times b^{n-2} + \dots + d_1 \times b^1 + d_0 \times b^0 \\ & = \\ & \sum_{i=0}^{n-1} d_i \times b^i \end{aligned}$$

Introdução

- ▶ Notação numérica posicional
 - ▶ É definida pelos dígitos d_i e pela base b adotada para representação dos números

$$\begin{aligned} & d_{n-1}d_{n-2}\dots d_1d_0{}_b \\ & = \\ & d_{n-1} \times b^{n-1} + d_{n-2} \times b^{n-2} + \dots + d_1 \times b^1 + d_0 \times b^0 \\ & = \\ & \sum_{i=0}^{n-1} d_i \times b^i \end{aligned}$$

- ▶ O valor da base B determina a quantidade de dígitos

$$b = 2 \longrightarrow \{0, 1\}$$

$$b = 10 \longrightarrow \{0, 1, \dots, 9\}$$

$$b = k \longrightarrow \{0, 1, \dots, k-1\}$$

Introdução

- ▶ Representação do sinal
 - ▶ No método de complementação a 2 é utilizado o bit mais significativo para indicar o sinal do número

$$\begin{aligned} 23 &= 2 \times 10^1 + 3 \times 10^0 \\ &= 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= \underline{00010111}_2 \end{aligned}$$

Introdução

- ▶ Representação do sinal
 - ▶ No método de complementação a 2 é utilizado o bit mais significativo para indicar o sinal do número

$$\begin{aligned} 23 &= 2 \times 10^1 + 3 \times 10^0 \\ &= 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= \underline{00010111}_2 \end{aligned}$$

$$\begin{aligned} -23 &= -00010111_2 \\ &= 11101000_2 + 1_2 \\ &= \underline{11101001}_2 \end{aligned}$$

Introdução

► Análise de complexidade

- Para um número x e uma base b arbitrários, a quantidade de dígitos deste número é $n = \lceil \log_b x \rceil$

$$\begin{array}{r} + \\ \begin{array}{rrrr} \varnothing & 1 & 1 & \varnothing \\ \varnothing & 1 & \varnothing & 1 \end{array} \\ \hline 1 & \varnothing & 1 & 1 \end{array}$$

$$\begin{array}{r} \times \\ \begin{array}{rrrr} \varnothing & 1 & \varnothing & 1 \\ \varnothing & 1 & 1 & \varnothing \end{array} \\ \hline \varnothing & \varnothing & \varnothing & \varnothing \\ \\ + \\ \begin{array}{rrrr} \varnothing & 1 & \varnothing & 1 \\ \varnothing & \varnothing & \varnothing & \varnothing \end{array} \\ \hline \varnothing & \varnothing & 1 & 1 & 1 & 1 & \varnothing \end{array}$$

Introdução

► Análise de complexidade

- Para um número x e uma base b arbitrários, a quantidade de dígitos deste número é $n = \lceil \log_b x \rceil$

$$\begin{array}{r} + \\ \begin{array}{rrrr} \emptyset & 1 & 1 & \emptyset \\ \emptyset & 1 & \emptyset & 1 \end{array} \\ \hline 1 & \emptyset & 1 & 1 \end{array}$$

$O(N)$

$$\begin{array}{r} \times \\ \begin{array}{rrrr} \emptyset & 1 & \emptyset & 1 \\ \emptyset & 1 & 1 & \emptyset \end{array} \\ \hline \emptyset & \emptyset & \emptyset & \emptyset \end{array}$$

$$\begin{array}{r} + \\ \begin{array}{rrrr} & \emptyset & 1 & \emptyset & 1 \\ \begin{array}{rrrr} \emptyset & 1 & \emptyset & 1 \\ \emptyset & \emptyset & \emptyset & \emptyset \end{array} \\ \hline \emptyset & \emptyset & 1 & 1 & 1 & 1 & \emptyset \end{array} \end{array}$$

$O(N^2)$

Introdução

- ▶ Números compostos e primos
 - ▶ Considere dois números inteiros positivos c e p , tal que $c > 1$ (composto) e $p > 1$ (primo)

Introdução

- ▶ Números compostos e primos
 - ▶ Considere dois números inteiros positivos c e p , tal que $c > 1$ (composto) e $p > 1$ (primo)
 - ▶ Um número composto c só precisa ter um divisor diferente de 1 e c (ex: $8 = 1 \times 2 \times 4 = 1 \times 2 \times 2 \times 2$)

Introdução

- ▶ Números compostos e primos
 - ▶ Considere dois números inteiros positivos c e p , tal que $c > 1$ (composto) e $p > 1$ (primo)
 - ▶ Um número composto c só precisa ter um divisor diferente de 1 e c (ex: $8 = 1 \times 2 \times 4 = 1 \times 2 \times 2 \times 2$)
 - ▶ Para que p seja definido como primo, ele deve ser divisível somente por 1 e p (ex: $7 = 1 \times 7$)

$$\text{Primo} \equiv \neg \text{Composto}$$

$$\text{Composto} \equiv \neg \text{Primo}$$

Algoritmos numéricos

- ▶ Estrutura de número com precisão dupla e simples

$$\text{Dupla}(x) = x_{2n-1} \dots x_{n-1} \dots x_0 b$$

$$\text{Simples}(x) = x_{n-1} \dots x_0 b$$

```
1 // Definição dos dígitos
2 #define d_t (uint64_t)
3 #define s_t (uint32_t)
4 // Estrutura de número
5 typedef struct num_t {
6     s_t* d;
7     uint32_t n;
8     uint32_t t;
9 } num_t;
10 // Definição da base numérica
11 const d_t b = ((d_t)(1) << (sizeof(s_t) << 3));
```

Para converter um dígito simples com sinal para duplo é necessário fazer a extensão do sinal

Algoritmos numéricos

- ▶ Adição dos números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b + v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b + v_{n-1} \dots v_0 b$$

Algoritmos numéricos

- ▶ Adição dos números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b + v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b + v_{n-1} \dots v_0 b$$

- ▶ Adição dos dígitos w_i

$$w_i = (u_i + v_i + c) \bmod b$$

Algoritmos numéricos

- ▶ Adição dos números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b + v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b + v_{n-1} \dots v_0 b$$

- ▶ Adição dos dígitos w_i

$$w_i = (u_i + v_i + c) \bmod b$$

- ▶ Cálculo do *carry* c

$$c = \left\lfloor \frac{(u_i + v_i + c)}{b} \right\rfloor$$

Algoritmos numéricos

- ▶ Adição dos números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b + v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b + v_{n-1} \dots v_0 b$$

- ▶ Adição dos dígitos w_i

$$w_i = (u_i + v_i + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(u_i + v_i + c)}{b} \right\rfloor$$

$$0 \leq u_i + v_i + c \leq (b - 1) + (b - 1) + 1$$

Algoritmos numéricos

- ▶ Adição dos números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b + v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b + v_{n-1} \dots v_0 b$$

- ▶ Adição dos dígitos w_i

$$w_i = (u_i + v_i + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(u_i + v_i + c)}{b} \right\rfloor$$

$$0 \leq u_i + v_i + c < 2b$$

Algoritmos numéricos

- ▶ Adição dos números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b + v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b + v_{n-1} \dots v_0 b$$

- ▶ Adição dos dígitos w_i :

$$w_i = (u_i + v_i + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(u_i + v_i + c)}{b} \right\rfloor$$

$$0 \leq \frac{(u_i + v_i + c)}{b} < 2$$

Algoritmos numéricos

- Adição de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

```
1 // Procedimento de adição
2 void adicionar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wi = 0;
5     w->n = max(u->n, v->n);
6     // wi = ui + vi + c
7     for(uint32_t i = 0; i < w->n; i++) {
8         wi = d_t(u->d[i]) + d_t(v->d[i]) + d_t(c);
9         w->d[i] = wi;
10        c = wi / b;
11    }
12    // Ajuste do carry e dígitos
13    if(w->n < w->t) w->d[w->n++] = c;
14    ajustar_n(w);
15 }
```

Algoritmos numéricos

- ▶ Adição de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

```
1 // Procedimento de adição
2 void adicionar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wi = 0;
5     w->n = max(u->n, v->n);
6     // wi = ui + vi + c
7     for(uint32_t i = 0; i < w->n; i++) {
8         wi = d_t(u->d[i]) + d_t(v->d[i]) + d_t(c);
9         w->d[i] = wi;
10        c = wi / b;
11    }
12    // Ajuste do carry e dígitos
13    if(w->n < w->t) w->d[w->n++] = c;
14    ajustar_n(w);
15 }
```

Espaço e tempo $O(n)$

Algoritmos numéricos

- ▶ Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b - v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b - v_{n-1} \dots v_0 b$$

Algoritmos numéricos

- ▶ Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b - v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b - v_{n-1} \dots v_0 b$$

- ▶ Subtração dos dígitos w_i

$$w_i = (u_i - v_i + c) \bmod b$$

Algoritmos numéricos

- ▶ Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b - v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b - v_{n-1} \dots v_0 b$$

- ▶ Subtração dos dígitos w_i

$$w_i = (u_i - v_i + c) \bmod b$$

- ▶ Cálculo do *carry* c

$$c = \left\lfloor \frac{(u_i - v_i + c)}{b} \right\rfloor$$

Algoritmos numéricos

- ▶ Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b - v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b - v_{n-1} \dots v_0 b$$

- ▶ Subtração dos dígitos w_i

$$w_i = (u_i - v_i + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(u_i - v_i + c)}{b} \right\rfloor$$

$$0 - (b - 1) - 1 \leq u_i - v_i + c \leq (b - 1) - 0 - 0$$

Algoritmos numéricos

- ▶ Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b - v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b - v_{n-1} \dots v_0 b$$

- ▶ Subtração dos dígitos w_i

$$w_i = (u_i - v_i + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(u_i - v_i + c)}{b} \right\rfloor$$

$$-b \leq u_i - v_i + c < b$$

Algoritmos numéricos

- ▶ Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$w_b = u_b - v_b$$

$$w_n \dots w_0 b = u_{n-1} \dots u_0 b - v_{n-1} \dots v_0 b$$

- ▶ Subtração dos dígitos w_i

$$w_i = (u_i - v_i + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(u_i - v_i + c)}{b} \right\rfloor$$

$$-1 \leq \frac{(u_i - v_i + c)}{b} < 1$$

Algoritmos numéricos

- Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

```
1 // Procedimento de subtração
2 void subtrair(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wi = 0;
5     w->n = max(u->n, v->n);
6     // wi = ui - vi + c
7     for(uint32_t i = 0; i < w->n; i++) {
8         wi = d_t(u->d[i]) - d_t(v->d[i]) + es(c);
9         w->d[i] = wi;
10        c = wi / b;
11    }
12    // Ajuste do carry (extensão de sinal) e dígitos
13    while(c != 0 && w->n < w->t) w->d[w->n++] = c;
14    ajustar_n(w);
15 }
```

Algoritmos numéricos

- Subtração de números u_b e v_b com n dígitos para um número inteiro w_b com $n + 1$ dígitos

```
1 // Procedimento de subtração
2 void subtrair(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wi = 0;
5     w->n = max(u->n, v->n);
6     // wi = ui - vi + c
7     for(uint32_t i = 0; i < w->n; i++) {
8         wi = d_t(u->d[i]) - d_t(v->d[i]) + es(c);
9         w->d[i] = wi;
10        c = wi / b;
11    }
12    // Ajuste do carry (extensão de sinal) e dígitos
13    while(c != 0 && w->n < w->t) w->d[w->n++] = c;
14    ajustar_n(w);
15 }
```

Espaço e tempo $O(n)$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$w_b = u_b \times v_b$$

$$w_{2n-1} \dots w_0 b = u_{n-1} \dots u_0 b \times v_{n-1} \dots v_0 b$$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$w_b = u_b \times v_b$$

$$w_{2n-1} \dots w_0 b = u_{n-1} \dots u_0 b \times v_{n-1} \dots v_0 b$$

- ▶ Multiplicação dos dígitos w_{i+j}

$$w_{i+j} = (w_{i+j} + u_i \times v_j + c) \bmod b$$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$w_b = u_b \times v_b$$

$$w_{2n-1} \dots w_0 b = u_{n-1} \dots u_0 b \times v_{n-1} \dots v_0 b$$

- ▶ Multiplicação dos dígitos w_{i+j}

$$w_{i+j} = (w_{i+j} + u_i \times v_j + c) \bmod b$$

- ▶ Cálculo do *carry* c

$$c = \left\lfloor \frac{(w_{i+j} + u_i \times v_j + c)}{b} \right\rfloor$$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$w_b = u_b \times v_b$$

$$w_{2n-1} \dots w_0 b = u_{n-1} \dots u_0 b \times v_{n-1} \dots v_0 b$$

- ▶ Multiplicação dos dígitos w_{i+j}

$$w_{i+j} = (w_{i+j} + u_i \times v_j + c) \bmod b$$

- ▶ Cálculo do *carry* c

$$c = \left\lfloor \frac{(w_{i+j} + u_i \times v_j + c)}{b} \right\rfloor$$

$$0 \leq w_{i+j} + u_i \times v_j + c \leq (b-1) + (b-1)^2 + (b-1)$$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$w_b = u_b \times v_b$$

$$w_{2n-1} \dots w_0 b = u_{n-1} \dots u_0 b \times v_{n-1} \dots v_0 b$$

- ▶ Multiplicação dos dígitos w_{i+j}

$$w_{i+j} = (w_{i+j} + u_i \times v_j + c) \bmod b$$

- ▶ Cálculo do *carry* c

$$c = \left\lfloor \frac{(w_{i+j} + u_i \times v_j + c)}{b} \right\rfloor$$

$$0 \leq w_{i+j} + u_i \times v_j + c < b^2$$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$w_b = u_b \times v_b$$

$$w_{2n-1} \dots w_0 b = u_{n-1} \dots u_0 b \times v_{n-1} \dots v_0 b$$

- ▶ Multiplicação dos dígitos w_{i+j}

$$w_{i+j} = (w_{i+j} + u_i \times v_j + c) \bmod b$$

- ▶ Cálculo do carry c

$$c = \left\lfloor \frac{(w_{i+j} + u_i \times v_j + c)}{b} \right\rfloor$$

$$0 \leq \frac{(w_{i+j} + u_i \times v_j + c)}{b} < b$$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

```
1 // Procedimento de multiplicação
2 void multiplicar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wij = 0; num_t* x = criar();
5     x->n = min(u->n + v->n, x->t);
6     // wi+j = wi+j + ui * vj + c
7     for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
8         for(j = 0; j < v->n; j++) {
9             wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
10                d_t(v->d[j]) + d_t(c);
11             x->d[i + j] = wij; c = wij / b;
12         }
13         if(i + j < x->t) x->d[i + j] = c;
14     }
15     ajustar_n(x); atribuir(w, x); destruir(&x);
```

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

```
1 // Procedimento de multiplicação
2 void multiplicar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wij = 0; num_t* x = criar();
5     x->n = min(u->n + v->n, x->t);
6     // wi+j = wi+j + ui * vj + c
7     for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
8         for(j = 0; j < v->n; j++) {
9             wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
10                d_t(v->d[j]) + d_t(c);
11            x->d[i + j] = wij; c = wij / b;
12        }
13        if(i + j < x->t) x->d[i + j] = c;
14    }
15    ajustar_n(x); atribuir(w, x); destruir(&x);
```

Espaço $O(n)$ e tempo $O(n^2)$

Departamento de Computação / UFS

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

```
1 // Procedimento de multiplicação
2 void multiplicar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wij = 0; num_t* x = criar();
5     x->n = min(u->n + v->n, x->t);
6     // wi+j = wi+j + ui * vj + c
7     for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
8         for(j = 0; j < v->n; j++) {
9             wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
10                d_t(v->d[j]) + d_t(c);
11             x->d[i + j] = wij; c = wij / b;
12         }
13         if(i + j < x->t) x->d[i + j] = c;
14     }
15     ajustar_n(x); atribuir(w, x); destruir(&x);
```

$$\text{Karatsuba } O\left(n^{\log_2 3}\right)^*$$

Departamento de Computação / UFS

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

```
1 // Procedimento de multiplicação
2 void multiplicar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wij = 0; num_t* x = criar();
5     x->n = min(u->n + v->n, x->t);
6     // wi+j = wi+j + ui * vj + c
7     for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
8         for(j = 0; j < v->n; j++) {
9             wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
10                d_t(v->d[j]) + d_t(c);
11             x->d[i + j] = wij; c = wij / b;
12         }
13         if(i + j < x->t) x->d[i + j] = c;
14     }
15     ajustar_n(x); atribuir(w, x); destruir(&x);
}
```

Schönhage–Strassen $O(n \times \log n \times \log \log n)^*$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

```
1 // Procedimento de multiplicação
2 void multiplicar(num_t* w, num_t* u, num_t* v) {
3     // Inicialização de variáveis
4     s_t c = 0; d_t wij = 0; num_t* x = criar();
5     x->n = min(u->n + v->n, x->t);
6     // wi+j = wi+j + ui * vj + c
7     for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
8         for(j = 0; j < v->n; j++) {
9             wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
10                d_t(v->d[j]) + d_t(c);
11             x->d[i + j] = wij; c = wij / b;
12         }
13         if(i + j < x->t) x->d[i + j] = c;
14     }
15 } ajustar_n(x); atribuir(w, x); destruir(&x);
```

Harvey-Hoeven $O(n \times \log n)^*$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & 8 & 7 & 6 \\ \times & & & & & \\ u & & & & 2 & 3 \\ \hline w & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{aligned} w[0 + 0] &= w[0 + 0] + u[0] \times v[0] + c \bmod 10 \\ &= 0 + 6 \times 3 + 0 \bmod 10 \\ &= 8 \end{aligned}$$

$$c = \left\lfloor \frac{18}{10} \right\rfloor = 1$$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & 8 & 7 & 6 \\ \times & & & & & \\ u & & 2 & 3 \\ \hline w & 0 & 0 & 0 & 0 & 8 \end{array}$$

$$\begin{aligned} w[0 + 1] &= w[0 + 1] + u[0] \times v[1] + c \bmod 10 \\ &= 0 + 3 \times 7 + 1 \bmod 10 \\ &= 2 \end{aligned}$$

$$c = \left\lfloor \frac{22}{10} \right\rfloor = 2$$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & & 8 & 7 & 6 \\ \times & & & & & & \\ u & & & & & 2 & 3 \\ \hline w & 0 & 0 & 0 & 2 & 8 \end{array}$$

$$\begin{aligned} w[0 + 2] &= w[0 + 2] + u[0] \times v[2] + c \bmod 10 \\ &= 0 + 3 \times 8 + 2 \bmod 10 \\ &= 6 \end{aligned}$$

$$c = \left\lfloor \frac{26}{10} \right\rfloor = 2$$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & 8 & 7 & 6 \\ & & & \times & & \\ u & & & & 2 & 3 \\ \hline w & \emptyset & 2 & 6 & 2 & 8 \end{array}$$

$$\begin{aligned} w[1 + 0] &= w[1 + 0] + u[1] \times v[0] + c \bmod 10 \\ &= 2 + 2 \times 6 + 0 \bmod 10 \\ &= 4 \end{aligned}$$

$$c = \left\lfloor \frac{14}{10} \right\rfloor = 1$$

Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & 8 & 7 & 6 \\ & & & \times & & \\ u & & & & 2 & 3 \\ \hline w & \emptyset & 2 & 6 & 4 & 8 \end{array}$$

$$\begin{aligned}w[1+1] &= w[1+1] + u[1] \times v[1] + c \bmod 10 \\&= 6 + 2 \times 7 + 1 \bmod 10 \\&= 1\end{aligned}$$

$$c = \left\lfloor \frac{21}{10} \right\rfloor = 2$$

Algoritmos numéricos

- Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & 8 & 7 & 6 \\ & & & \times & & \\ u & & 2 & 3 & & \\ \hline w & 0 & 2 & 1 & 4 & 8 \end{array}$$

$$\begin{aligned} w[1+2] &= w[1+2] + u[1] \times v[2] + c \bmod 10 \\ &= 2 + 2 \times 8 + 2 \bmod 10 \\ &= 0 \end{aligned}$$

$$c = \left\lfloor \frac{20}{10} \right\rfloor = 2$$

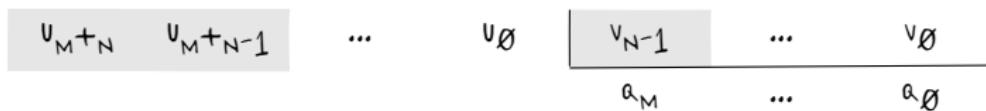
Algoritmos numéricos

- ▶ Multiplicação de números positivos u_b e v_b com n dígitos para um número inteiro w_b com $2n$ dígitos

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 \\ v & & & 8 & 7 & 6 \\ & & & \times & & \\ u & & & & 2 & 3 \\ \hline w & 2 & 0 & 1 & 4 & 8 \end{array}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos



$$\hat{q}_b = \left\lfloor \frac{u_{m+n_b} \times b + u_{m+n-1_b}}{v_{n-1_b}} \right\rfloor$$
$$\hat{r}_b = u_b - v_b \times \hat{q}_b$$

$$0 \leq \hat{r}_b < v_b$$

Ideia do algoritmo: estimar um \hat{q}_b próximo do valor de q_b , dividindo-se os dois dígitos mais significativos de u_b pelo dígito mais significativo de v_b

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

$$\begin{array}{ccccccccc} u_{m+n} & u_{m+n-1} & \cdots & u_0 & & v_{n-1} & \cdots & v_0 \\ & & & & & & & \\ & & & & & a_m & \cdots & a_0 \end{array}$$

$$\begin{aligned}\hat{q}_b &= \left\lfloor \frac{u_{m+n_b} \times b + u_{m+n-1_b}}{v_{n-1_b}} \right\rfloor \\ \hat{r}_b &= u_b - v_b \times \hat{q}_b\end{aligned}$$

$$0 \leq \hat{r}_b < v_b$$

A normalização de u_b e v_b garante que $v_{n-1_b} \geq \left\lfloor \frac{b}{2} \right\rfloor$ e que o erro de \hat{q}_b será de até 2 ($\hat{q}_b - 2 \leq q_b \leq \hat{q}_b$)

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento de divisão
2 void dividir(num_t* q, num_t* r, num_t* u, num_t* v) {
3     // Divisor com apenas 1 dígito
4     if(v->n == 1) {
5         // q = u / v[0], r = u % v[0]
6         zerar(r); r->d[0] = dividir_digito(q, u,
7             v->d[0]); r->n = (r->d[0] != 0);
8     }
9     // Divisor é maior do que dividendo
10    else if(v->n > u->n) {
11        // q = 0, r = u
12        zerar(q); atribuir(r, u);
13    }
14    // Divisão longa dos números
15    else dividir_numero(q, r, u, v);
}
```

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento de divisão
2 void dividir(num_t* q, num_t* r, num_t* u, num_t* v) {
3     // Divisor com apenas 1 dígito
4     if(v->n == 1) {
5         // q = u / v[0], r = u % v[0]
6         zerar(r); r->d[0] = dividir_digito(q, u,
7             v->d[0]); r->n = (r->d[0] != 0);
8     }
9     // Divisor é maior do que dividendo
10    else if(v->n > u->n) {
11        // q = 0, r = u
12        zerar(q); atribuir(r, u);
13    }
14    // Divisão longa dos números
15    else dividir_numero(q, r, u, v);
}
```

Espaço $O(m + n)$ e tempo $O(m \times n)$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento de divisão longa
2 void dividir_numero(num_t* q, num_t* r, num_t* u,
3     num_t* v) {
4     // Criando números auxiliares
5     num_t* x = criar();
6     num_t* y = criar();
7     // Normalização dos números
8     s_t f = b / (d_t(v->d[v->n - 1]) + 1);
9     multiplicar_digito(x, u, f);
10    multiplicar_digito(y, v, f);
11    // Inicializando q
12    zerar(q);
13    // Quantidade de dígitos de q
14    q->n = u->n - v->n + 1
15    ...
16 }
```

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento de divisão longa
2 void dividir_numero(num_t* q, num_t* r, num_t* u,
3 num_t* v) {
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 for(int32_t i = q->n - 1; i >= 0; i--) {
15     // Calculando e ajustando as estimativas
16     d_t xx = d_t(x->d[v->n + i]) * b +
17         d_t(x->d[v->n - 1 + i]);
18     d_t qc = xx / y->d[v->n - 1], rc = xx %
19         y->d[v->n - 1];
20     ajustar_qc_rc(&qc, &rc, x, y, i);
21     // Realizando a divisão
22     multiplicar_dígito(r, y, qc);
23     deslocar_esquerda(r, i); subtrair(x, x, r);
24     q->d[i] = qc; checar_overflow(q, x, y, i);
25 }
26 ...
27 ...
28 }
```

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento de divisão longa
2 void dividir_numero(num_t* q, num_t* r, num_t* u,
3     num_t* v) {
4     ...
5     ...
6     // Calculando o resto
7     dividir_dígito(r, x, f);
8     // Ajuste na quantidade de dígitos de q
9     ajustar_n(q);
10    // Desalocando números auxiliares
11    destruir(&x);
12    destruir(&y);
13 }
```

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Considerando $u_{10} = 8132$ e $v_{10} = 443$, é feita a normalização para termos $\hat{q}_b - 2 \leq q_b \leq \hat{q}_b$

$$f = \left\lfloor \frac{b}{(v_{n-1_b} + 1)} \right\rfloor = \left\lfloor \frac{10}{(4 + 1)} \right\rfloor = 2$$
$$x_{10} = u_{10} \times f = 16264$$
$$y_{10} = v_{10} \times f = 886$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Considerando $u_{10} = 8132$ e $v_{10} = 443$, é feita a normalização para termos $\hat{q}_b - 2 \leq q_b \leq \hat{q}_b$

$$f = \left\lfloor \frac{b}{(v_{n-1_b} + 1)} \right\rfloor = \left\lfloor \frac{10}{(4 + 1)} \right\rfloor = 2$$
$$x_{10} = u_{10} \times f = 16264$$
$$y_{10} = v_{10} \times f = 886$$

A normalização não afeta
o quociente da divisão, uma vez que $\frac{u}{v} = \frac{x}{y}$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 1$

$$\begin{array}{cccccccccc} & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ & \boxed{1} & 6 & 2 & 6 & 4 & \boxed{8} & 8 & 6 \\ & & & & & & & & \end{array}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 1$

$$\begin{array}{cccccccc|cc} & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \hline & & & & & & & & \end{array}$$

$$\hat{q}_{10} = \left\lfloor \frac{(x_{4_{10}} \times 10 + x_{3_{10}})}{y_2} \right\rfloor = \left\lfloor \frac{(1 \times 10 + 6)}{8} \right\rfloor = 2$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 1$

$$\begin{array}{cccccccc|cc} & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \hline & & & & & & & & \end{array}$$

$$\begin{aligned}\hat{q}_{10} &= \left\lfloor \frac{(x_{4_{10}} \times 10 + x_{3_{10}})}{y_2} \right\rfloor = \left\lfloor \frac{(1 \times 10 + 6)}{8} \right\rfloor = 2 \\ \hat{r}_{10} &= (x_{4_{10}} \times 10 + x_{3_{10}}) \bmod y_{2_{10}} = (1 \times 10 + 6) \bmod 8 = 0\end{aligned}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 1$

$$\begin{array}{ccccccccc} & 4 & 3 & 2 & 1 & \emptyset & 2 & 1 & \emptyset \\ & \boxed{1} & 6 & 2 & 6 & 4 & \boxed{8} & 8 & 6 \\ & & & & & & & & \hline \end{array}$$

$$\hat{q_{10}} = ? \quad \vee \quad \hat{q_{10}} \times y_{10} ? > \hat{r_{10}} \times 10 + x_{2_{10}}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 1$

$$\begin{array}{ccccccccc} & 4 & 3 & 2 & 1 & \emptyset & 2 & 1 & \emptyset \\ & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \hline & & & & & & & & \end{array}$$

$$\begin{aligned} \hat{q}_{10} &\stackrel{?}{=} 10 \quad \vee \quad \hat{q}_{10} \times y_{10} \stackrel{?}{>} \hat{r}_{10} \times 10 + x_{2_{10}} \\ 2 &\neq 10 \quad \vee \quad 2 \times 8 > 0 \times 10 + 2 \end{aligned}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 1$

$$\begin{array}{cccccccc|ccccc} & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ 1 & 6 & 2 & 6 & 4 & & 8 & 8 & 6 \\ & & & & & & & & \\ & & & & & & & & \end{array}$$

$$\begin{aligned} \hat{q}_{10} &\stackrel{?}{=} 10 \quad \vee \quad \hat{q}_{10} \times y_{10} \stackrel{?}{>} \hat{r}_{10} \times 10 + x_{210} \\ 2 &\neq 10 \quad \vee \quad 2 \times 8 > 0 \times 10 + 2 \end{aligned}$$

Checando se a estimativa do quociente \hat{q}_{10} é igual a base (não representável) ou se o resto estimado $\hat{r}_{10} \times 10 + x_{210}$ é menor do que $\hat{q}_{10} \times y_{10}$ (próxima subtração será negativa)

Algoritmos numéricos

- Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento para checagem de estimativa
2 void checar_estimativa(d_t* qc, d_t* rc, num_t* x,
3                         num_t* y, int32_t i) {
4     // qc = b or qc * y[n - 2] > rc * b + x[n - 2 + i]
5     if((*qc) == b || (*qc) * y->d[y->n - 2] > (*rc) * b
6         + x->d[y->n - 2 + i]) {
7         // Ajustando quociente e resto
8         (*qc) = (*qc) - 1;
9         (*rc) = (*rc) + y->d[y->n - 1];
10    }
11
12 // Procedimento de ajuste de estimativas
13 void ajustar_qc_rc(d_t* qc, d_t* rc, num_t* x, num_t*
14                     y, int32_t i) {
15     // Checando estimativa
16     checar_estimativa(qc, rc, x, y, i);
17     if((*rc) < b) checar_estimativa(qc, rc, x, y, i);
18 }
```

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos

```
1 // Procedimento de checagem de overflow
2 void checar_overflow(num_t* q, num_t* x, num_t* y,
3     int32_t i) {
4     // Checagem de overflow da subtração
5     if(x->d[y->n + 1 + i] != 0) {
6         // Reduzir quociente
7         q->d[i] = q->d[i] - 1;
8         // Adicionar divisor no dividendo
9         y->d[y->n] = 0;
10        deslocar_esquerda(y, i);
11        adicionar(x, x, y);
12    }
}
```

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & \downarrow & & & \\ \hline & 7 & 4 & 0 & 4 & & & 1 & & \end{array}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & \downarrow & & & \\ \hline & 7 & 4 & 0 & 4 & & & 1 & & \end{array}$$

$$\hat{q}_{10} = \left\lfloor \frac{(x_{310} \times b + x_{210})}{y_{210}} \right\rfloor = \left\lfloor \frac{(7 \times 10 + 4)}{8} \right\rfloor = 9$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & \downarrow & & \\ \hline & 7 & 4 & 0 & 4 & & & 1 & \\ \end{array}$$

$$\hat{q}_{10} = \left\lfloor \frac{(x_{3_{10}} \times b + x_{2_{10}})}{y_{2_{10}}} \right\rfloor = \left\lfloor \frac{(7 \times 10 + 4)}{8} \right\rfloor = 9$$

$$\hat{r}_{10} = (x_{3_{10}} \times b + x_{2_{10}}) \bmod y_{2_{10}} = (7 \times 10 + 4) \bmod 8 = 2$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & \downarrow & & \\ & 7 & 4 & 0 & 4 & & & 1 & \\ \hline \end{array}$$

$$\hat{q_{10}} \stackrel{?}{=} 10 \quad \vee \quad \hat{q_{10}} \times y_{10} \stackrel{?}{>} \hat{r_{10}} \times 10 + x_{10}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & \downarrow & & \\ & 7 & 4 & 0 & 4 & & & 1 & \\ \hline \end{array}$$

$$\begin{aligned} \hat{q_{10}} &\stackrel{?}{=} 10 \quad \vee \quad \hat{q_{10}} \times y_{10} \stackrel{?}{>} \hat{r_{10}} \times 10 + x_{10} \\ 9 &\neq 10 \quad \vee \quad 9 \times 8 > 2 \times 10 + 6 \end{aligned}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ - & & & & & & 1 & 8 \\ 8 & 8 & 6 & & & & & \\ - & & & 7 & 4 & 0 & 4 \\ & & & 7 & 0 & 8 & 8 \\ - & & & 3 & 1 & 6 \end{array}$$

$$q_{10} = 18$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & & 1 & 8 \\ \hline & 7 & 4 & 0 & 4 & & & \\ - & 7 & 0 & 8 & 8 & & & \\ \hline & 3 & 1 & 6 & & & & \end{array}$$

$$q_{10} = 18$$

$$r_{10} = \frac{r_{10}}{f} = \frac{316}{2} = 158$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & & 1 & 8 \\ \hline & 7 & 4 & 0 & 4 & & & \\ - & 7 & 0 & 8 & 8 & & & \\ \hline & 3 & 1 & 6 & & & & \end{array}$$

$$\frac{16624}{886} = \frac{8132}{443}$$

Algoritmos numéricos

- ▶ Divisão de números positivos u_b e v_b , com $m + n + 1$ e n dígitos, para q_b e r_b , com m e n dígitos
 - ▶ Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$\begin{array}{r} 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\ - & 1 & 6 & 2 & 6 & 4 & | & 8 & 8 & 6 \\ & 8 & 8 & 6 & & & & 1 & 8 \\ \hline & 7 & 4 & 0 & 4 & & & \\ - & 7 & 0 & 8 & 8 & & & \\ \hline & 3 & 1 & 6 & & & & \end{array}$$

$$\begin{array}{rcl} \frac{16624}{886} & = & \frac{8132}{443} \\ \frac{8132}{443} & = & 18 \times 443 + 158 \end{array}$$

Algoritmos numéricos

- Exponenciação de um número positivo u_b com n dígitos por um exponente k_b para v_b com kn dígitos

$$v_b = u_b^k \rightarrow v_{kn-1} \dots v_0 b = u_{n-1} \dots u_0 b^k$$

```
1 // Procedimento de exponenciação
2 void exponenciar(num_t* v, num_t* u, num_t* k) {
3     // x = u, y = k, v = 1
4     num_t* x = criar(); num_t* y = criar(); atribuir(x,
5             u); atribuir(y, k); atribuir(v, 1);
6     // Repete enquanto y > 0
7     while(y->n > 0) {
8         if(dividir_dígito(y, y, 2) == 1)
9             multiplicar(v, v, x);
10            multiplicar(x, x, x);
11        }
12    destruir(&x); destruir(&y);
}
```

Algoritmos numéricos

- Exponenciação de um número positivo u_b com n dígitos por um exponente k_b para v_b com kn dígitos

$$v_b = u_b^k \rightarrow v_{kn-1} \dots v_0 b = u_{n-1} \dots u_0 b^k$$

```
1 // Procedimento de exponenciação
2 void exponenciar(num_t* v, num_t* u, num_t* k) {
3     // x = u, y = k, v = 1
4     num_t* x = criar(); num_t* y = criar(); atribuir(x,
5             u); atribuir(y, k); atribuir(v, 1);
6     // Repete enquanto y > 0
7     while(y->n > 0) {
8         if(dividir_dígito(y, y, 2) == 1)
9             multiplicar(v, v, x);
10            multiplicar(x, x, x);
11        }
12    destruir(&x); destruir(&y);
}
```

Espaço $O(kn)$ e tempo $O(n^2 \log_2 n)$

Algoritmos numéricos

- ▶ Aritmética modular: é definida pela redução modular $u \equiv v \pmod{m}$, que é o resto da divisão inteira de v por m , tal que $u \in \mathbb{Z}_m = \{0, 1, \dots, m - 1\}$

Algoritmos numéricos

- ▶ Aritmética modular: é definida pela redução modular $u \equiv v \pmod{m}$, que é o resto da divisão inteira de v por m , tal que $u \in \mathbb{Z}_m = \{0, 1, \dots, m-1\}$
 - ▶ Propriedades das operações \oplus no grupo finito \mathbb{Z}_m
 - ▶ Fechamento: se $u, v \in \mathbb{Z}_m$, então $u \oplus v \pmod{m} \in \mathbb{Z}_m$

Algoritmos numéricos

- ▶ Aritmética modular: é definida pela redução modular $u \equiv v \pmod{m}$, que é o resto da divisão inteira de v por m , tal que $u \in \mathbb{Z}_m = \{0, 1, \dots, m - 1\}$
 - ▶ Propriedades das operações \oplus no grupo finito \mathbb{Z}_m
 - ▶ Fechamento: se $u, v \in \mathbb{Z}_m$, então $u \oplus v \pmod{m} \in \mathbb{Z}_m$
 - ▶ Reflexividade: $v \equiv v \pmod{m}$

Algoritmos numéricos

- ▶ Aritmética modular: é definida pela redução modular $u \equiv v \pmod{m}$, que é o resto da divisão inteira de v por m , tal que $u \in \mathbb{Z}_m = \{0, 1, \dots, m-1\}$
 - ▶ Propriedades das operações \oplus no grupo finito \mathbb{Z}_m
 - ▶ Fechamento: se $u, v \in \mathbb{Z}_m$, então $u \oplus v \pmod{m} \in \mathbb{Z}_m$
 - ▶ Reflexividade: $v \equiv v \pmod{m}$
 - ▶ Simetria: se $u \equiv v \pmod{m}$, então $v \equiv u \pmod{m}$

Algoritmos numéricos

- ▶ Aritmética modular: é definida pela redução modular $u \equiv v \pmod{m}$, que é o resto da divisão inteira de v por m , tal que $u \in \mathbb{Z}_m = \{0, 1, \dots, m-1\}$
 - ▶ Propriedades das operações \oplus no grupo finito \mathbb{Z}_m
 - ▶ Fechamento: se $u, v \in \mathbb{Z}_m$, então $u \oplus v \pmod{m} \in \mathbb{Z}_m$
 - ▶ Reflexividade: $v \equiv v \pmod{m}$
 - ▶ Simetria: se $u \equiv v \pmod{m}$, então $v \equiv u \pmod{m}$
 - ▶ Transitividade: se $u \equiv v \pmod{m}$ e $v \equiv w \pmod{m}$, então $u \equiv w \pmod{m}$

Algoritmos numéricos

- ▶ Operações básicas com $u, v \in \mathbb{Z}_m$
 - ▶ Adição

$$(u + v) \bmod m \equiv \begin{cases} u + v, & \text{se } u + v < m \\ u + v - m, & \text{se } u + v \geq m \end{cases}$$

Algoritmos numéricos

- ▶ Operações básicas com $u, v \in \mathbb{Z}_m$

- ▶ Adição

$$(u + v) \bmod m \equiv \begin{cases} u + v, & \text{se } u + v < m \\ u + v - m, & \text{se } u + v \geq m \end{cases}$$

- ▶ Subtração

$$(u - v) \bmod m \equiv \begin{cases} u - v, & \text{se } u - v \geq 0 \\ u - v + m, & \text{se } u - v < 0 \end{cases}$$

Algoritmos numéricos

- ▶ Operações básicas com $u, v \in \mathbb{Z}_m$

- ▶ Adição

$$(u + v) \bmod m \equiv \begin{cases} u + v, & \text{se } u + v < m \\ u + v - m, & \text{se } u + v \geq m \end{cases}$$

- ▶ Subtração

$$(u - v) \bmod m \equiv \begin{cases} u - v, & \text{se } u - v \geq 0 \\ u - v + m, & \text{se } u - v < 0 \end{cases}$$

- ▶ Multiplicação

$$(u \times v) \bmod m \equiv (u \bmod m \times v \bmod m) \bmod m$$

Algoritmos numéricos

- ▶ Operações básicas com $u, v \in \mathbb{Z}_m$

- ▶ Adição

$$(u + v) \bmod m \equiv \begin{cases} u + v, & \text{se } u + v < m \\ u + v - m, & \text{se } u + v \geq m \end{cases}$$

- ▶ Subtração

$$(u - v) \bmod m \equiv \begin{cases} u - v, & \text{se } u - v \geq 0 \\ u - v + m, & \text{se } u - v < 0 \end{cases}$$

- ▶ Multiplicação

$$(u \times v) \bmod m \equiv (u \bmod m \times v \bmod m) \bmod m$$

- ▶ Inverso multiplicativo

$$(u \times v) \equiv 1 \bmod m \rightarrow \text{mdc}(u, m) = 1 \wedge v \equiv u^{-1} \bmod m$$

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para o número w_b com n dígitos
 - Algoritmo de Euclides: $w_b = mdc(u_b, v_b)$

```
1 // Procedimento de maior divisor comum
2 void mdc(num_t* w, num_t* u, num_t* v) {
3     // a = u, b = v
4     num_t* a = criar(); num_t* b = criar(); atribuir(a,
5             u); atribuir(b, v);
6     // Repete enquanto b > 0
7     while(b->n > 0) {
8         // w = a % b, a = b, b = w
9         modulo(w, a, b); atribuir(a, b); atribuir(b, w);
10    }
11    // w = a
12    atribuir(w, a); destruir(&a); destruir(&b);
}
```

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para o número w_b com n dígitos
 - Algoritmo de Euclides: $w_b = mdc(u_b, v_b)$

```
1 // Procedimento de maior divisor comum
2 void mdc(num_t* w, num_t* u, num_t* v) {
3     // a = u, b = v
4     num_t* a = criar(); num_t* b = criar(); atribuir(a,
5             u); atribuir(b, v);
6     // Repete enquanto b > 0
7     while(b->n > 0) {
8         // w = a % b, a = b, b = w
9         modulo(w, a, b); atribuir(a, b); atribuir(b, w);
10    }
11    // w = a
12    atribuir(w, a); destruir(&a); destruir(&b);
}
```

$$\text{Espaço } O(n) \text{ e tempo } O(n^2 \log n)$$

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para w_b, x_b e y_b com n dígitos
 - Algoritmo de Euclides Estendido: $w_b = u_b \times x_b + v_b \times y_b$

```
1 // Procedimento de maior divisor comum estendido
2 void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
3           num_t* v) {
4     // Criando números auxiliares
5     num_t* a = criar(); num_t* b = criar(); num_t* x1 =
6         criar(); num_t* x2 = criar(); num_t* y1 =
7         criar(); num_t* y2 = criar(); num_t* q =
8         criar(); num_t* r = criar(); num_t* qx1 =
9         criar(); num_t* qy1 = criar();
10    // a = u, b = v, x2 = 1, x1 = 0, y2 = 0, y1 = 1
11    atribuir(a, u); atribuir(b, v); atribuir(x2, 1);
12    zerar(x1); zerar(y2); atribuir(y1, 1);
13    ...
14 }
15 }
```

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para w_b, x_b e y_b com n dígitos
 - Algoritmo de Euclides Estendido: $w_b = u_b \times x_b + v_b \times y_b$

```
1 // Procedimento de maior divisor comum estendido
2 void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
3           num_t* v) {
4   ...
5   ...
6   ...
7   // Repete enquanto v > 0
8   while(v->n > 0) {
9     // u / v = v * q + r,
10    dividir(r, q, u, v);
11    // qx1 = x1 * q, qy1 = y1 * q
12    multiplicar(qx1, q, x1); multiplicar(qy1, q,
13                  y1);
14    // x = x2 - qx1, y = y2 - qy1
15    subtrair(x, x2, qx1); subtrair(y, y2, qy1);
16    ...
17  }
18  ...
19 }
```

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para w_b, x_b e y_b com n dígitos
 - Algoritmo de Euclides Estendido: $w_b = u_b \times x_b + v_b \times y_b$

```
1 // Procedimento de maior divisor comum estendido
2 void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
3           num_t* v) {
4   ...
5   ...
6   ...
7   // Repete enquanto v > 0
8   while(v->n > 0) {
9     ...
10    ...
11    // u = v, v = r, x2 = x1, x1 = x, y2 = y1, y1 =
12      y
13    atribuir(u, v); atribuir(v, r); atribuir(x2,
14      x1); atribuir(x1, x); atribuir(y2, y1);
15      atribuir(y1, y);
16    }
17    ...
18  }
```

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para w_b, x_b e y_b com n dígitos
 - Algoritmo de Euclides Estendido: $w_b = u_b \times x_b + v_b \times y_b$

```
1 // Procedimento de maior divisor comum estendido
2 void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
3     num_t* v) {
...
18     // w = u, x = x2, y = y2
19     atribuir(w, u); atribuir(x, x2); atribuir(y, y2);
20     // Desalocando números auxiliares
21     destruir(&a); destruir(&b); destruir(&x1);
        destruir(&x2); destruir(&y1); destruir(&y2);
        destruir(&q); destruir(&r); destruir(&qx1);
        destruir(&qy1);
22 }
```

Algoritmos numéricos

- Maior divisor comum dos números inteiros positivos u_b e v_b , com $u_b \geq v_b$, para w_b, x_b e y_b com n dígitos
 - Algoritmo de Euclides Estendido: $w_b = u_b \times x_b + v_b \times y_b$

```
1 // Procedimento de maior divisor comum estendido
2 void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
3     num_t* v) {
...
18     // w = u, x = x2, y = y2
19     atribuir(w, u); atribuir(x, x2); atribuir(y, y2);
20     // Desalocando números auxiliares
21     destruir(&a); destruir(&b); destruir(&x1);
        destruir(&x2); destruir(&y1); destruir(&y2);
        destruir(&q); destruir(&r); destruir(&qx1);
        destruir(&qy1);
22 }
```

$$\text{Espaço } O(n) \text{ e tempo } O(n^2 \log n)$$

Algoritmos numéricos

- Inverso multiplicativo do número inteiro positivo u_b para o número v_b em \mathbb{Z}_m com n dígitos

```
1 // Procedimento de inverso multiplicativo
2 void inverso_m(num_t* v, num_t* u, num_t* m) {
3     // Criando números auxiliares
4     num_t* w = criar();
5     num_t* x = criar();
6     num_t* y = criar();
7     // w = vx + my
8     mdce(w, x, y, u, m);
9     // w == 1 -> v = x
10    if(igual(w, 1)) atribuir(v, x);
11    // v = 0
12    else zerar(v);
13    // Desalocando números auxiliares
14    destruir(&w); destruir(&x); destruir(&y);
15 }
```

Algoritmos numéricos

- Inverso multiplicativo do número inteiro positivo u_b para o número v_b em \mathbb{Z}_m com n dígitos

```
1 // Procedimento de inverso multiplicativo
2 void inverso_m(num_t* v, num_t* u, num_t* m) {
3     // Criando números auxiliares
4     num_t* w = criar();
5     num_t* x = criar();
6     num_t* y = criar();
7     // w = vx + my
8     mdce(w, x, y, u, m);
9     // w == 1 -> v = x
10    if(igual(w, 1)) atribuir(v, x);
11    // v = 0
12    else zerar(v);
13    // Desalocando números auxiliares
14    destruir(&w); destruir(&x); destruir(&y);
15 }
```

$$\text{Espaço } O(n) \text{ e tempo } O(n^2 \log n)$$

Algoritmos numéricos

- ▶ Inverso multiplicativo do número inteiro positivo u_b para o número v_b em \mathbb{Z}_m com n dígitos
 - ▶ Considerando que $u_{10} = 8$, vamos calcular o valor de seu inverso multiplicativo $u_{10} \times v_{10} \equiv 1 \pmod{13}$

a	R	x	y	A	B	x_2	x_1	y_2	y_1
0	0	0	0	8	13	1	0	0	1
0	8	1	0	13	8	0	1	1	0
1	5	-1	1	8	5	1	-1	0	1
1	3	2	-1	5	3	-1	2	1	-1
1	2	-3	2	3	2	2	-3	-1	2
1	1	5	-3	2	1	-3	5	2	-3
2	0	-13	8	1	0	5	-13	-3	8
2	0	5	-3	1	0	5	-13	-3	8

$$mdce(u, m) = u \times x + m \times y$$

Algoritmos numéricos

- ▶ Inverso multiplicativo do número inteiro positivo u_b para o número v_b em \mathbb{Z}_m com n dígitos
 - ▶ Considerando que $u_{10} = 8$, vamos calcular o valor de seu inverso multiplicativo $u_{10} \times v_{10} \equiv 1 \pmod{13}$

a	R	x	y	A	B	x_2	x_1	y_2	y_1
0	0	0	0	8	13	1	0	0	1
0	8	1	0	13	8	0	1	1	0
1	5	-1	1	8	5	1	-1	0	1
1	3	2	-1	5	3	-1	2	1	-1
1	2	-3	2	3	2	2	-3	-1	2
1	1	5	-3	2	1	-3	5	2	-3
2	0	-13	8	1	0	5	-13	-3	8
2	0	5	-3	1	0	5	-13	-3	8

$$mdce(8, 13) = 8 \times x + 13 \times y$$

Algoritmos numéricos

- ▶ Inverso multiplicativo do número inteiro positivo u_b para o número v_b em \mathbb{Z}_m com n dígitos
 - ▶ Considerando que $u_{10} = 8$, vamos calcular o valor de seu inverso multiplicativo $u_{10} \times v_{10} \equiv 1 \pmod{13}$

a	R	x	y	A	B	x_2	x_1	y_2	y_1
0	0	0	0	8	13	1	0	0	1
0	8	1	0	13	8	0	1	1	0
1	5	-1	1	8	5	1	-1	0	1
1	3	2	-1	5	3	-1	2	1	-1
1	2	-3	2	3	2	2	-3	-1	2
1	1	5	-3	2	1	-3	5	2	-3
2	0	-13	8	1	0	5	-13	-3	8
2	0	5	-3	1	0	5	-13	-3	8

$$mdce(8, 13) = 8 \times 5 + 13 \times -3 = 1$$

Algoritmos numéricos

- ▶ Inverso multiplicativo do número inteiro positivo u_b para o número v_b em \mathbb{Z}_m com n dígitos
 - ▶ Considerando que $u_{10} = 8$, vamos calcular o valor de seu inverso multiplicativo $u_{10} \times v_{10} \equiv 1 \pmod{13}$

a	R	x	y	A	B	x_2	x_1	y_2	y_1
0	0	0	0	8	13	1	0	0	1
0	8	1	0	13	8	0	1	1	0
1	5	-1	1	8	5	1	-1	0	1
1	3	2	-1	5	3	-1	2	1	-1
1	2	-3	2	3	2	2	-3	-1	2
1	1	5	-3	2	1	-3	5	2	-3
2	0	-13	8	1	0	5	-13	-3	8
2	0	5	-3	1	0	5	-13	-3	8

$$8 \times 5 \equiv 1 \pmod{13}$$