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

Implementação fixa (não atualizável)

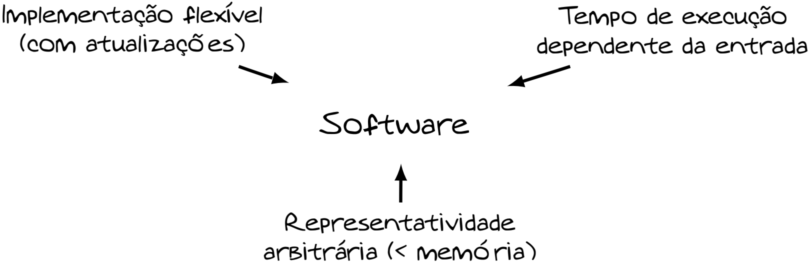
Tempo de execução constante (ciclos)

Hardware

Representatividade limitada (< 128 bits)

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

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

$$(d_{n-1}d_{n-2}\ldots d_1d_0)_b$$

Introdução

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

$$(d_{n-1}d_{n-2}\ldots d_1d_0)_b$$
$$=$$
$$d_{n-1}\times b^{n-1}+d_{n-2}\times b^{n-2}+\ldots+d_1\times b^1+d_0\times b^0$$

Introdução

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

$$(d_{n-1}d_{n-2}\ldots d_1d_0)_b$$
$$=$$
$$d_{n-1}\times b^{n-1}+d_{n-2}\times b^{n-2}+\ldots+d_1\times b^1+d_0\times b^0$$
$$=$$
$$\sum_{i=0}^{n-1} d_i\times b^i$$

Introdução

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

$$(d_{n-1}d_{n-2}\ldots d_1d_0)_b$$
$$=$$
$$d_{n-1}\times b^{n-1}+d_{n-2}\times b^{n-2}+\ldots+d_1\times b^1+d_0\times b^0$$
$$=$$
$$\sum_{i=0}^{n-1} d_i\times b^i$$

  - O valor da base B determina a quantidade de dígitos

$$\begin{aligned}
b=2 &\quad\longrightarrow\quad \{0,1\}\\
b=10 &\quad\longrightarrow\quad \{0,1,\ldots,9\}\\
b=k &\quad\longrightarrow\quad \{0,1,\ldots,k-1\}
\end{aligned}$$

Introdução

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

$$
\begin{aligned}
23 &= 2 \times 10^1 + 3 \times 10^0 \\
&= 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= \underline{0}0010111_2
\end{aligned}
$$

Introdução

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

$$
\begin{aligned}
23 &= 2 \times 10^1 + 3 \times 10^0 \\
&= 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= \underline{0}0010111_2
\end{aligned}
$$

$$
\begin{aligned}
-23 &= -00010111_2 \\
&= 11101000_2 + 1_2 \\
&= \underline{1}1101001_2
\end{aligned}
$$

Introdução

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

$$
\begin{array}{r@{\quad}cccc}
 & 0 & 1 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 1 & 1
\end{array}
$$

$$
\begin{array}{r@{\quad}ccccccc}
 & & & & 0 & 1 & 0 & 1 \\
\times & & & & 0 & 1 & 1 & 0 \\
\hline
 & & & & 0 & 0 & 0 & 0 \\
 & & & 0 & 1 & 0 & 1 & \\
 & & 0 & 1 & 0 & 1 & & \\
+ & 0 & 0 & 0 & 0 & & & \\
\hline
 & 0 & 0 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

Introdução

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

$$
\begin{array}{rcccc}
 & 0 & 1 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 1 & 1
\end{array}
$$

$O(n)$

$$
\begin{array}{rccccccc}
 & & & & 0 & 1 & 0 & 1 \\
\times & & & & 0 & 1 & 1 & 0 \\
\cline{5-8}
 & & & & 0 & 0 & 0 & 0 \\
 & & & 0 & 1 & 0 & 1 & \\
 & & 0 & 1 & 0 & 1 & & \\
+ & 0 & 0 & 0 & 0 & & & \\
\hline
 & 0 & 0 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

$O(n^2)$

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: $7 = 1 \times 7$)

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: $7 = 1 \times 7$)

$$\begin{aligned} Primo &\equiv \neg Composto \\ Composto &\equiv \neg Primo \end{aligned}$$

Algoritmos numéricos

- Estrutura de número com precisão dupla e simples

$$
\begin{aligned}
Dupla\,(x) &= {x_{2n-1} \ldots x_{n-1} \ldots x_0}_b \\
Simples\,(x) &= {x_{n-1} \ldots x_0}_b
\end{aligned}
$$

Algoritmos numéricos

- Estrutura de número com precisão dupla e simples

$$\begin{aligned} Dupla\,(x) &= x_{2n-1} \ldots x_{n-1} \ldots x_{0\,b} \\ Simples\,(x) &= x_{n-1} \ldots x_{0\,b} \end{aligned}$$

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

$$
\begin{aligned}
w_b &= u_b + v_b \\
w_n \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} + v_{n-1} \dots v_{0\,b}
\end{aligned}
$$

Algoritmos numéricos

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

$$\begin{aligned} w_b &= u_b + v_b \\ w_n \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} + v_{n-1} \dots v_{0\,b} \end{aligned}$$

  - 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

$$
\begin{aligned}
w_b &= u_b + v_b \\
w_n \ldots w_0{}_b &= u_{n-1} \ldots u_0{}_b + v_{n-1} \ldots v_0{}_b
\end{aligned}
$$

  - 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

$$
\begin{aligned}
w_b &= u_b + v_b \\
w_n \ldots w_{0\,b} &= u_{n-1} \ldots u_{0\,b} + v_{n-1} \ldots v_{0\,b}
\end{aligned}
$$

  - 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

$$\begin{aligned} w_b &= u_b + v_b \\ w_n \ldots w_0{}_b &= u_{n-1} \ldots u_0{}_b + v_{n-1} \ldots v_0{}_b \end{aligned}$$

  - 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

$$\begin{aligned} w_b &= u_b + v_b \\ w_n \ldots w_{0\,b} &= u_{n-1} \ldots u_{0\,b} + v_{n-1} \ldots v_{0\,b} \end{aligned}$$

  - 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

```
// Procedimento de adição
void adicionar(num_t* w, num_t* u, num_t* v) {
    // Inicialização de variáveis
    s_t c = 0; d_t wi = 0;
    w->n = max(u->n, v->n);
    // wi = ui + vi + c
    for(uint32_t i = 0; i < w->n; i++) {
        wi = d_t(u->d[i]) + d_t(v->d[i]) + d_t(c);
        w->d[i] = wi;
        c = wi / b;
    }
    // Ajuste do carry e dígitos
    if(w->n < w->t) w->d[w->n++] = c;
    ajustar_n(w);
}
```

Algoritmos numéricos

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

```
// Procedimento de adição
void adicionar(num_t* w, num_t* u, num_t* v) {
    // Inicialização de variáveis
    s_t c = 0; d_t wi = 0;
    w->n = max(u->n, v->n);
    // wi = ui + vi + c
    for(uint32_t i = 0; i < w->n; i++) {
        wi = d_t(u->d[i]) + d_t(v->d[i]) + d_t(c);
        w->d[i] = wi;
        c = wi / b;
    }
    // Ajuste do carry e dígitos
    if(w->n < w->t) w->d[w->n++] = c;
    ajustar_n(w);
}
```

Espaço e tempo $O(n)$

Algoritmos numéricos

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

$$
\begin{aligned}
w_b &= u_b - v_b \\
w_n \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} - v_{n-1} \dots v_{0\,b}
\end{aligned}
$$

Algoritmos numéricos

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

$$\begin{aligned} w_b &= u_b - v_b \\ w_n \ldots w_{0\,b} &= u_{n-1} \ldots u_{0\,b} - v_{n-1} \ldots v_{0\,b} \end{aligned}$$

  - 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 com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$\begin{aligned} w_b &= u_b - v_b \\ w_n \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} - v_{n-1} \dots v_{0\,b} \end{aligned}$$

  - 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 com n dígitos para um número inteiro w_b com $n+1$ dígitos

$$\begin{aligned} w_b &= u_b - v_b \\ w_n \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} - v_{n-1} \dots v_{0\,b} \end{aligned}$$

  - 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 com n dígitos para um número inteiro w_b com $n + 1$ dígitos

$$\begin{aligned} w_b &= u_b - v_b \\ w_n \ldots w_{0\,b} &= u_{n-1} \ldots u_{0\,b} - v_{n-1} \ldots v_{0\,b} \end{aligned}$$

  - 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 com n dígitos para um número inteiro w_b com $n+1$ dígitos

$$
\begin{aligned}
w_b &= u_b - v_b \\
w_n \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} - v_{n-1} \dots v_{0\,b}
\end{aligned}
$$

  - 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 com n dígitos para um número inteiro w_b com $n+1$ dígitos

```
// Procedimento de subtração
void subtrair(num_t* w, num_t* u, num_t* v) {
    // Inicialização de variáveis
    s_t c = 0; d_t wi = 0;
    w->n = max(u->n, v->n);
    // wi = ui - vi + c
    for(uint32_t i = 0; i < w->n; i++) {
        wi = d_t(u->d[i]) - d_t(v->d[i]) + es(c);
        w->d[i] = wi;
        c = wi / b;
    }
    // Ajuste do carry (extensão de sinal) e dígitos
    while(c != 0 && w->n < w->t) w->d[w->n++] = c;
    ajustar_n(w);
}
```

Algoritmos numéricos

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

```
// Procedimento de subtração
void subtrair(num_t* w, num_t* u, num_t* v) {
    // Inicialização de variáveis
    s_t c = 0; d_t wi = 0;
    w->n = max(u->n, v->n);
    // wi = ui - vi + c
    for(uint32_t i = 0; i < w->n; i++) {
        wi = d_t(u->d[i]) - d_t(v->d[i]) + es(c);
        w->d[i] = wi;
        c = wi / b;
    }
    // Ajuste do carry (extensão de sinal) e dígitos
    while(c != 0 && w->n < w->t) w->d[w->n++] = c;
    ajustar_n(w);
}
```

Espaço e tempo $O(n)$

Algoritmos numéricos

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

$$
\begin{aligned}
w_b &= u_b \times v_b \\
w_{n+m-1} \ldots w_0{}_b &= u_{n-1} \ldots u_0{}_b \times \; v_{m-1} \ldots v_0{}_b
\end{aligned}
$$

Algoritmos numéricos

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

$$\begin{aligned} w_b &= u_b \times v_b \\ {w_{n+m-1} \dots w_0}_b &= {u_{n-1} \dots u_0}_b \times {v_{m-1} \dots v_0}_b \end{aligned}$$

  - 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 e m dígitos para um número inteiro w_b com $n+m$ dígitos

$$
\begin{aligned}
w_b &= u_b \times v_b \\
w_{n+m-1} \dots w_0{}_b &= u_{n-1} \dots u_0{}_b \times v_{m-1} \dots v_0{}_b
\end{aligned}
$$

  - 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 e m dígitos para um número inteiro w_b com $n+m$ dígitos

$$\begin{aligned} w_b &= u_b \times v_b \\ w_{n+m-1} \dots w_{0\,b} &= u_{n-1} \dots u_{0\,b} \times v_{m-1} \dots v_{0\,b} \end{aligned}$$

  - 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 e m dígitos para um número inteiro w_b com $n+m$ dígitos

$$
\begin{aligned}
w_b &= u_b \times v_b \\
w_{n+m-1} \ldots w_{0\,b} &= u_{n-1} \ldots u_{0\,b} \times v_{m-1} \ldots v_{0\,b}
\end{aligned}
$$

  - 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 e m dígitos para um número inteiro w_b com $n+m$ dígitos

$$\begin{aligned} w_b &= u_b \times v_b \\ w_{n+m-1} \ldots w_{0\,b} &= u_{n-1} \ldots u_{0\,b} \times v_{m-1} \ldots v_{0\,b} \end{aligned}$$

  - 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 e m dígitos para um número inteiro w_b com $n + m$ dígitos

```
// Procedimento de multiplicação
void multiplicar(num_t* w, num_t* u, num_t* v) {
    // Inicialização de variáveis
    s_t c = 0; d_t wij = 0; num_t* x = criar();
    x->n = min(u->n + v->n, x->t);
    // wi+j = wi+j + ui * vj + c
    for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
        for(j = 0; j < v->n; j++) {
            wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
                d_t(v->d[j]) + d_t(c);
            x->d[i + j] = wij; c = wij / b;
        }
        if(i + j < x->t) x->d[i + j] = c;
    }
    ajustar_n(x); atribuir(w, x); destruir(&x);
}
```

Algoritmos numéricos

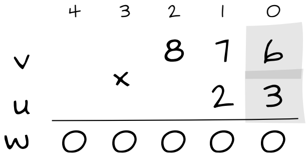
- Multiplicação de números positivos u_b e v_b com n e m dígitos para um número inteiro w_b com $n + m$ dígitos

```
// Procedimento de multiplicação
void multiplicar(num_t* w, num_t* u, num_t* v) {
    // Inicialização de variáveis
    s_t c = 0; d_t wij = 0; num_t* x = criar();
    x->n = min(u->n + v->n, x->t);
    // wi+j = wi+j + ui * vj + c
    for(uint32_t i = 0, j; i < u->n; c = 0, i++) {
        for(j = 0; j < v->n; j++) {
            wij = d_t(x->d[i + j]) + d_t(u->d[i]) *
                d_t(v->d[j]) + d_t(c);
            x->d[i + j] = wij; c = wij / b;
        }
        if(i + j < x->t) x->d[i + j] = c;
    }
    ajustar_n(x); atribuir(w, x); destruir(&x);
}
```

Espaço e tempo $O(n \times m)$

Algoritmos numéricos

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



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

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

Algoritmos numéricos

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

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

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

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

Algoritmos numéricos

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

	4	3	2	1	0
v			8	7	6
		×			
u				2	3
w	0	0	0	2	8

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

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

Algoritmos numéricos

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

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

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

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

Algoritmos numéricos

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

	4	3	2	1	0
v			8	7	6
		×			
u				2	3
w	0	2	6	4	8

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

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

Algoritmos numéricos

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

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

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

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

Algoritmos numéricos

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

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

Algoritmos numéricos

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

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

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

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 n e m dígitos, para q_b e r_b, com $n - m + 1$ e m dígitos

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

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

Aplicando a normalização de u_b e v_b, o erro da estimativa $\hat{q}_b$ será de até 2 unidades ($\hat{q}_b - 2 \leq q_b \leq \hat{q}_b$)

Algoritmos numéricos

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

```c
// Procedimento de divisão
void dividir(num_t* q, num_t* r, num_t* u, num_t* v) {
    // Divisor com apenas 1 dígito
    if(v->n == 1) {
        // q = u / v[0], r = u % v[0]
        zerar(r); r->d[0] = dividir_digito(q, u,
            v->d[0]); r->n = (r->d[0] != 0);
    }
    // Divisor é maior do que dividendo
    else if(v->n > u->n) {
        // q = 0, r = u
        zerar(q); atribuir(r, u);
    }
    // Divisão longa dos números
    else dividir_numero(q, r, u, v);
}
```

Algoritmos numéricos

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

```c
// Procedimento de divisão
void dividir(num_t* q, num_t* r, num_t* u, num_t* v) {
    // Divisor com apenas 1 dígito
    if(v->n == 1) {
        // q = u / v[0], r = u % v[0]
        zerar(r); r->d[0] = dividir_digito(q, u,
            v->d[0]); r->n = (r->d[0] != 0);
    }
    // Divisor é maior do que dividendo
    else if(v->n > u->n) {
        // q = 0, r = u
        zerar(q); atribuir(r, u);
    }
    // Divisão longa dos números
    else dividir_numero(q, r, u, v);
}
```

Espaço $O(n \times m)$ e tempo $O((n - m + 1) \times m)$

Algoritmos numéricos

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

```c
// Procedimento de divisão longa
void dividir_numero(num_t* q, num_t* r, num_t* u,
    num_t* v) {
    // Criando números auxiliares
    num_t* x = criar();
    num_t* y = criar();
    // Normalização dos números
    s_t f = b / (d_t(v->d[v->n - 1]) + 1);
    multiplicar_digito(x, u, f);
    multiplicar_digito(y, v, f);
    // Inicializando q
    zerar(q);
    // Quantidade de dígitos de q
    q->n = u->n - v->n + 1
    ...
}
```

Algoritmos numéricos

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

```
// Procedimento de divisão longa
void dividir_numero(num_t* q, num_t* r, num_t* u,
    num_t* v) {
    ...
    for(int32_t i = q->n - 1; i >= 0; i--) {
        // Calculando e ajustando as estimativas
        d_t xx = d_t(x->d[v->n + i]) * b +
            d_t(x->d[v->n - 1 + i]);
        d_t qc = xx / y->d[v->n - 1], rc = xx %
            y->d[v->n - 1];
        ajustar_qc_rc(&qc, &rc, x, y, i);
        // Realizando a divisão
        multiplicar_digito(r, y, qc);
        deslocar_esquerda(r, i); subtrair(x, x, r);
        q->d[i] = qc; checar_overflow(q, x, y, i);
    }
    ...
}
```

Algoritmos numéricos

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

```
// Procedimento de divisão longa
void dividir_numero(num_t* q, num_t* r, num_t* u,
    num_t* v) {
    ...
    // Calculando o resto
    dividir_digito(r, x, f);
    // Ajuste na quantidade de dígitos de q
    ajustar_n(q);
    // Desalocando números auxiliares
    destruir(&x);
    destruir(&y);
}
```

Algoritmos numéricos

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

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

Algoritmos numéricos

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

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

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

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

Algoritmos numéricos

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

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

$$\hat{q}_{10} = \left\lfloor \frac{(x_{m+i} \times b + x_{m-1+i})}{y_{m-1}} \right\rfloor = \left\lfloor \frac{(1 \times 10 + 6)}{8} \right\rfloor = 2$$

Algoritmos numéricos

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

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

$$
\begin{aligned}
\hat{q_{10}} &= \left\lfloor \frac{(x_{m+i} \times b + x_{m-1+i})}{y_{m-1}} \right\rfloor = \left\lfloor \frac{(1 \times 10 + 6)}{8} \right\rfloor = 2 \\
\hat{r}_{10} &= (x_{m+i} \times b + x_{m-1+i}) \bmod y_{n-1} = (1 \times 10 + 6) \bmod 8 = 0
\end{aligned}
$$

Algoritmos numéricos

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

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

$$\hat{q}_{10} \stackrel{?}{=} 10 \quad \vee \quad \hat{q}_{10} \times y_{m-2} > \hat{r}_{10} \times 10 + x_{m-2+i}$$

Algoritmos numéricos

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

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

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

Algoritmos numéricos

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

```c
// Procedimento para checagem de estimativa
void checar_estimativa(d_t* qc, d_t* rc, num_t* x,
    num_t* y, int32_t i) {
    // qc = b or qc * y[m - 2] > rc * b + x[m - 2 + i]
    if((*qc) == b || (*qc) * y->d[y->n - 2] > (*rc) *
        b + x->d[y->n - 2 + i]) {
        // Ajustando quociente e resto
        (*qc) = (*qc) - 1;
        (*rc) = (*rc) + y->d[y->n - 1];
    }
}
// Procedimento de ajuste de estimativas
void ajustar_qc_rc(d_t* qc, d_t* rc, num_t* x, num_t*
    y, int32_t i) {
    // Checando estimativa
    checar_estimativa(qc, rc, x, y, i);
    if((*rc) < b) checar_estimativa(qc, rc, x, y, i);
}
```

Algoritmos numéricos

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

```c
// Procedimento de checagem de overflow
void checar_overflow(num_t* q, num_t* x, num_t* y,
    int32_t i) {
    // Checagem de overflow da subtração
    if(x->d[y->n + 1 + i] != 0) {
        // Reduzir quociente
        q->d[i] = q->d[i] - 1;
        // Adicionar divisor no dividendo
        y->d[y->n] = 0;
        deslocar_esquerda(y, i);
        adicionar(x, x, y);
    }
}
```

Algoritmos numéricos

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

$$
\begin{array}{cccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \cline{7-9}
- & & 8 & 8 & 6 & \downarrow & & 1 & \\ \cline{2-5}
 & & 7 & 4 & 0 & 4 & & &
\end{array}
$$

Algoritmos numéricos

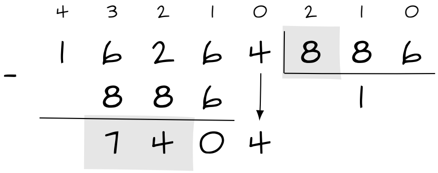
- Divisão de números positivos u_b e v_b, com n e m dígitos, para q_b e r_b, com $n-m+1$ e m dígitos
  - Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$
\begin{array}{c|ccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & \boxed{8} & 8 & 6 \\
\cline{7-9}
- & & 8 & 8 & 6 & & & 1 & \\
\cline{2-5}
 & & \boxed{7} & \boxed{4} & 0 & 4 & & & \\
\end{array}
$$

$$\hat{q_{10}} = \left\lfloor \frac{(x_{m+i} \times b + x_{m-1+i})}{y_{m-1}} \right\rfloor = \left\lfloor \frac{(7 \times 10 + 4)}{8} \right\rfloor = 9$$

Algoritmos numéricos

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



$$
\begin{aligned}
\hat{q}_{10} &= \left\lfloor \frac{(x_{m+i} \times b + x_{m-1+i})}{y_{m-1}} \right\rfloor = \left\lfloor \frac{(7 \times 10 + 4)}{8} \right\rfloor = 9 \\
\hat{r}_{10} &= (x_{m+i} \times b + x_{m-1+i}) \bmod y_{n-1} = (7 \times 10 + 4) \bmod 9 = 2
\end{aligned}
$$

Algoritmos numéricos

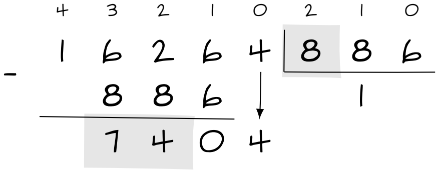
- Divisão de números positivos u_b e v_b, com n e m dígitos, para q_b e r_b, com $n - m + 1$ e m dígitos
  - Calculando $\frac{x_{10}}{y_{10}} = \frac{16264}{886} = q_{10} \times y_{10} + r_{10}$, com $i = 0$

$$
\begin{array}{cccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\
\cline{7-9}
- & & 8 & 8 & 6 & \downarrow & & 1 & \\
\cline{2-5}
 & & 7 & 4 & 0 & 4 & & &
\end{array}
$$

$$\hat{q}_{10} \stackrel{?}{=} 10 \quad \vee \quad \hat{q}_{10} \times y_{m-2} > \hat{r}_{10} \times 10 + x_{m-2+i}$$

Algoritmos numéricos

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



$$\hat{q}_{10} \stackrel{?}{=} 10 \quad \vee \quad \hat{q}_{10} \times y_{m-2} > \hat{r}_{10} \times 10 + x_{m-2+i}$$

$$9 \neq 10 \quad \vee \quad 9 \times 8 > 2 \times 10 + 6$$

Algoritmos numéricos

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

$$
\begin{array}{rccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\
\cline{7-9}
- & & 8 & 8 & 6 & & & 1 & 8 \\
\cline{2-5}
 & & 7 & 4 & 0 & 4 & & & \\
- & & 7 & 0 & 8 & 8 & & & \\
\cline{3-6}
 & & & 3 & 1 & 6 & & &
\end{array}
$$

$$q_{10} = 18$$

Algoritmos numéricos

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

$$
\begin{array}{cccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \cline{7-9}
- & & 8 & 8 & 6 & & & 1 & 8 \\ \cline{2-5}
 & & 7 & 4 & 0 & 4 & & & \\
- & & 7 & 0 & 8 & 8 & & & \\ \cline{3-6}
 & & & 3 & 1 & 6 & & &
\end{array}
$$

$$
\begin{aligned}
q_{10} &= 18 \\
r_{10} &= \frac{r_{10}}{f} = \frac{316}{2} = 158
\end{aligned}
$$

Algoritmos numéricos

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

$$
\begin{array}{cccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \cline{7-9}
- & & 8 & 8 & 6 & & & 1 & 8 \\ \cline{2-5}
 & & 7 & 4 & 0 & 4 & & & \\
- & & 7 & 0 & 8 & 8 & & & \\ \cline{3-6}
 & & & 3 & 1 & 6 & & &
\end{array}
$$

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

Algoritmos numéricos

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

$$\begin{array}{cccccc|ccc}
 & 4 & 3 & 2 & 1 & 0 & 2 & 1 & 0 \\
 & 1 & 6 & 2 & 6 & 4 & 8 & 8 & 6 \\ \cline{7-9}
- & & 8 & 8 & 6 & & & 1 & 8 \\ \cline{2-5}
 & - & 7 & 4 & 0 & 4 & & & \\
 & & 7 & 0 & 8 & 8 & & & \\ \cline{3-6}
 & & & 3 & 1 & 6 & & &
\end{array}$$

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

$$\frac{8132}{443} = 18 \times 443 + 158$$

Algoritmos numéricos

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

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

```c
// Procedimento de exponenciação
void exponenciar(num_t* v, num_t* u, num_t* k) {
    // x = u, y = k, v = 1
    num_t* x = criar(); num_t* y = criar();
        atribuir(x, u); atribuir(y, k); atribuir(v,
        1);
    // Repete enquanto y > 0
    while(y->n > 0) {
        if(dividir_digito(y, y, 2) == 1)
            multiplicar(v, v, x);
        multiplicar(x, x, x);
    }
    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

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

```c
// Procedimento de exponenciação
void exponenciar(num_t* v, num_t* u, num_t* k) {
    // x = u, y = k, v = 1
    num_t* x = criar(); num_t* y = criar();
        atribuir(x, u); atribuir(y, k); atribuir(v,
        1);
    // Repete enquanto y > 0
    while(y->n > 0) {
        if(dividir_digito(y, y, 2) == 1)
            multiplicar(v, v, x);
        multiplicar(x, x, x);
    }
    destruir(&x); destruir(&y);
}
```

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

Algoritmos numéricos

- Aritmética modular: é definida pela redução modular $u = v \mod n$, que é o resto da divisão inteira de v por n, tal que $u \in \mathbb{Z}_n = \{0, 1, \ldots, n - 1\}$

Algoritmos numéricos

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

Algoritmos numéricos

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

Algoritmos numéricos

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

Algoritmos numéricos

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

Algoritmos numéricos

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

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

Algoritmos numéricos

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

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

  - Subtração

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

Algoritmos numéricos

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

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

  - Subtração

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

  - Multiplicação

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

Algoritmos numéricos

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

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

  - Subtração

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

  - Multiplicação

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

  - Inverso multiplicativo

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

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: $w_b = mdc(u_b, v_b)$

```
// Procedimento de maior divisor comum
void mdc(num_t* w, num_t* u, num_t* v) {
    // a = u, b = v
    num_t* a = criar(); num_t* b = criar();
        atribuir(a, u); atribuir(b, v);
    // Repete enquanto b > 0
    while(b->n > 0) {
        // w = a % b, a = b, b = w
        modulo(w, a, b); atribuir(a, b); atribuir(b,
            w);
    }
    // w = a
    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: $w_b = mdc(u_b, v_b)$

```
// Procedimento de maior divisor comum
void mdc(num_t* w, num_t* u, num_t* v) {
    // a = u, b = v
    num_t* a = criar(); num_t* b = criar();
        atribuir(a, u); atribuir(b, v);
    // Repete enquanto b > 0
    while(b->n > 0) {
        // w = a % b, a = b, b = w
        modulo(w, a, b); atribuir(a, b); atribuir(b,
            w);
    }
    // w = a
    atribuir(w, a); destruir(&a); destruir(&b);
}
```

Espaço e tempo $O(n)$

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: $w_b = u_b \times x_b + v_b \times y_b$

```
// Procedimento de maior divisor comum estendido
void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
    num_t* v) {
    // Criando números auxiliares
    num_t* a = criar(); num_t* b = criar(); num_t* x1
        = criar(); num_t* x2 = criar(); num_t* y1 =
        criar(); num_t* y2 = criar(); num_t* q =
        criar(); num_t* r = criar(); num_t* qx1 =
        criar(); num_t* qy1 = criar();
    // a = u, b = v, x2 = 1, x1 = 0, y2 = 0, y1 = 1
    atribuir(a, u); atribuir(b, v); atribuir(x2, 1);
        zerar(x1); zerar(y2); atribuir(y1, 1);
    ...
}
```

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: $w_b = u_b \times x_b + v_b \times y_b$

```
// Procedimento de maior divisor comum estendido
void mdce(num_t* w, num_t* x, num_t* y, num_t* u,
    num_t* v) {
     ...
     // Repete enquanto v > 0
     while(v->n > 0) {
          // u / v = v * q + r,
          dividir(r, q, u, v);
          // qx1 = x1 * q, qy1 = y1 * q
          multiplicar(qx1, q, x1); multiplicar(qy1, q,
              y1);
          // x = x2 - qx1, y = y2 - qy1
          subtrair(x, x2, qx1); subtrair(y, y2, qy1);
          ...
     }
     ...
}
```

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: $w_b = u_b \times x_b + v_b \times y_b$

```
// Procedimento de maior divisor comum estendido
void mdce(num_t* w, num_t* x, num_t* y, num_t* u, num_t* v) {
    ...
    // Repete enquanto v > 0
    while(v->n > 0) {
        ...
        // u = v, v = r, x2 = x1, x1 = x, y2 = y1, y1 = y
        atribuir(u, v); atribuir(v, r); atribuir(x2, x1); atribuir(x1, x); atribuir(y2, y1); atribuir(y1, y);
    }
    ...
}
```

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: $w_b = u_b \times x_b + v_b \times y_b$

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

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: $w_b = u_b \times x_b + v_b \times y_b$

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

Espaço $O(n)$ e tempo $O(n^2)$

Algoritmos numéricos

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

```
// Procedimento de inverso multiplicativo
void inverso_m(num_t* v, num_t* u, num_t* m) {
    // Criando números auxiliares
    num_t* w = criar();
    num_t* x = criar();
    num_t* y = criar();
    // w = vx + my
    mdce(w, x, y, u, m);
    // w == 1 -> v = x
    if(igual(w, 1)) atribuir(v, x);
    // v = 0
    else zerar(v);
    // Desalocando números auxiliares
    destruir(&w); destruir(&x); destruir(&y);
}
```

Algoritmos numéricos

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

```c
// Procedimento de inverso multiplicativo
void inverso_m(num_t* v, num_t* u, num_t* m) {
    // Criando números auxiliares
    num_t* w = criar();
    num_t* x = criar();
    num_t* y = criar();
    // w = vx + my
    mdce(w, x, y, u, m);
    // w == 1 -> v = x
    if(igual(w, 1)) atribuir(v, x);
    // v = 0
    else zerar(v);
    // Desalocando números auxiliares
    destruir(&w); destruir(&x); destruir(&y);
}
```

Espaço $O(n)$ e tempo $O(n^2)$

Algoritmos numéricos

- Inverso multiplicativo do número inteiro positivo u_b para o número v_b em $\mathbb{Z}_m$ com n dígitos
  - Ex: $u = 8 \bmod 13 \rightarrow u \times v \equiv 1 \bmod 13$

Q	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

Algoritmos numéricos

- Inverso multiplicativo do número inteiro positivo u_b para o número v_b em $\mathbb{Z}_m$ com n dígitos
  - Ex: $u = 8 \bmod 13 \rightarrow u \times v \equiv 1 \bmod 13$

Q	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

$$u \times v \equiv 1 \bmod m \rightarrow 8 \times 5 \bmod 13 \equiv 1 \bmod 13$$