UDESC

Complexidade de Algoritmos

Prof. Rafael Alceste Berri
rafaelberri@gmail.com

Prof. Diego Buchinger
diego.buchinger@outlook.com

Prof. Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

UDESC

Algoritmos com Inteiros Grandes

UDESC

Algoritmos com Inteiros Grandes

Até esse momento temos considerado **constante** a complexidade de operações como: **adição, subtração, multiplicação, divisão, módulo e comparação**.

Mas o que acontece quando essas operações envolvem números cujo o tamanho, em **número de bits**, é muito maior que a **palavra do processador** (atualmente 32 ou 64 bits)?

Soma

Um primeiro modelo de soma entre números inteiros não limitados a palavra do processador poderia ser similar ao método que utilizamos para somar números decimais:

Processador de 1 bit

n1 (238)		0	0	1	1	1	0	1	1	1	0
n2 (379)	+	0	1	0	1	1	1	1	0	1	1
soma		1	0	0	1	1	0	1	0	0	1

Quantos passos são necessários para calcular uma soma entre números de **k** dígitos?

Soma

Contudo, um processador consegue trabalhar com palavras de tamanho *p* – ou seja – em apenas uma única operação ele soma dois números que tenham no máximo *p* bits

Processador de 8 bits

									1	=> (*carry* entre suboperações)						
n1 (238)		0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0
n2 (379)	+	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	1
soma		0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	1

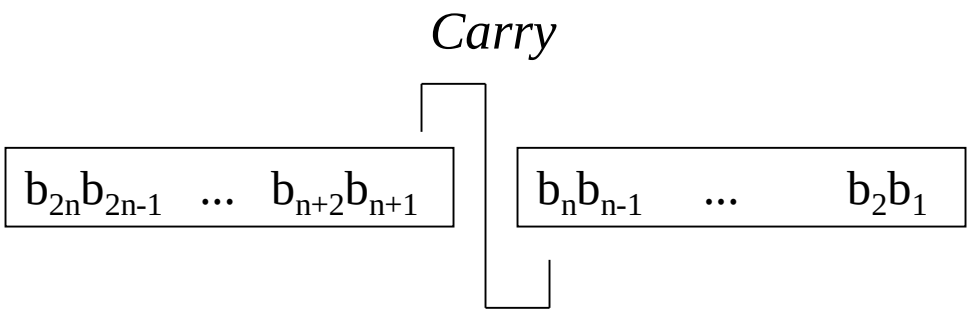
Qual é o maior valor de *carry*?

Espaço adicional – pior caso

Soma

De forma genérica

Adição de $2p$ bits, onde p é o tamanho da palavra do processador:



Assim, quando o número de bits (k) do número for próximo ao tamanho da palavra do processador teremos uma complexidade de tempo constante O(1)

UDESC

Soma

E quando o tamanho do número (*k*) for significativamente superior à palavra do processador, ou quando **k for variável**?

p = 32 / *k* = 64 ➔ serão necessárias 2 operações

p = 32 / *k* = 480 ➔ serão necessárias 15 operações

p = 32 / *k* = 4992 ➔ serão necessárias 156 operações

O(*k* / *p*) onde p é uma constante

Assim, podemos considerar que a adição de grandes números tem complexidade ***O*(*k*)**, onde *k* é o número de bits do maior número.

UDESC

Soma

Soma de inteiros: **O(*k*)**

Mas quanto vale k em relação aos números utilizados na soma?

R: o número de bits de *n*

379 (n) => 101111011 (k)

Como fazemos para sair do 379 e chegar no valor em binário?

$k = \log n$

O(k) = O(log n)

Multiplicação

Um primeiro modelo de multiplicação seria realizar somas sucessivas de um valor:

x = 8

y = 5

$$5 \times 8$$

$$\underbrace{8 + 8 + 8 + 8 + 8}_{\text{y vezes}} = 40$$

Multiplicação

```
bigInt mul (bigInt x, bigInt y) {
    bigInt r = 0;
    while (y > 0){    // Comparação entre inteiros grandes
        r = r + x;  // Adição entre inteiros grandes O(k).
        y--;
    }
    return r;
}
```

$$\underbrace{O(\mathrm{k}) + O(\mathrm{k}) + \ldots + O(k)}_{y \text{ vezes}}$$

Note que o tamanho de ***r*** também aumenta, assim como a complexidade da soma. Contudo esse aumento não ocorre em todas as operações

Sendo *k* o número de bits de x,
qual a complexidade de tempo para este algoritmo?

Multiplicação

$$\underbrace{O(k) + O(k) + \ldots + O(k)}_{y \text{ vezes}}$$

Considerando que x e y tem o mesmo número k de bits, qual a complexidade de tempo para este algoritmo?

$k = \log y \ (= \log x)$

$y = 2^k$

$O(y * k) = O(y * \log y)$

Logo: $O(2^k * k) = \mathbf{O(2^k)}$

Multiplicação

Um segundo modelo seria similar ao método que utilizamos para multiplicar números na base decimal:

```
   12
x 215
----
   60   (multiplica por 5, desloca 0)
  12    (multiplica por 1, desloca 1)
 24     (multiplica por 2, desloca 2)
----
2580
```

```
    1010 (10)
x   1101 (13)
-------------
    1010   (multiplica por 1, desloca 0)
   0000    (multiplica por 0, desloca 1)
  1010     (multiplica por 1, desloca 2)
 1010      (multiplica por 1, desloca 3)
-------------
10000010 (130)
```

Multiplicação

```
bigInt mul( bigInt x, bigInt y ){
   bigInt r;
   if (y == 0) return 0;
   r = mul(x, y >> 1) // r = x * (y/2)
   if ( par(y) )
        return r << 1; // return 2*r
   else
        return x + r << 1; // return x+2*r
}
```

Verificar se um número binário é par: O(?)

Deslocamento de bits: O(?)

Sendo k o número de bits de x e y,
qual a complexidade de tempo para este algoritmo?

Multiplicação

```
bigInt mul( bigInt x, bigInt y ){
   bigInt r;
   if (y == 0) return 0;
   r = mul(x, y >> 1) // r = x * (y/2)
   if ( par(y) )
        return r << 1; // return 2*r
   else
        return x + r << 1; // return x+2*r
}
```

Verificar se um número binário é par: O(1)

Deslocamento de bits: O(k)

Sendo k o número de bits de x e y,
qual a complexidade de tempo para este algoritmo?

UDESC

Multiplicação

Sendo k o número de bits de x e y,
qual a complexidade de tempo para este algoritmo?

São feitas k somas de complexidade O(k), logo:

k * O(k) = **$O(k^2)$**

ou **$O(\log^2 n)$**

<u>Será que tem como fazer melhor?</u>

Multiplicação

Existe um método que utiliza a abordagem de divisão e conquista para realizar a multiplicação.

$$xy = 2^{k} x_L y_L + 2^{\frac{k}{2}} (x_L y_R + x_R y_L) + x_R y_R$$

onde: x_L e y_L = parte esquerda do número (binário)

x_R e y_R = parte direita do número (binário)

Multiplicação

$$xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + (x_R y_R)$$

```
bigInt mul( bigInt x, bigInt y){
    bigInt xl, xr, yl, yr, p1, p2, p3, p4;
     int n = max( x.size(), y.size() );      // número de bits do maior número
    if (n == 1) return x*y;          // se número de bits for 1 (retorna 1 se x = 1 e y =1).
    xl = leftMost(x, n/2); xr= rightMost(x, n/2); // bits mais a esquerda e mais a direita.
    yl = leftMost(y, n/2); yr= rightMost(y, n/2);
    p1 = mul (xl, yl);
    p2 = mul (xl, yr);
    p3 = mul (xr, yl);
    p4 = mul (xr, yr);
    return    (p1 << n) + (p2 << (n/2)) + (p3 << (n/2)) + p4;
}
```

Multiplicação

Sendo *k* o número de bits de x e y,
qual a complexidade de tempo para este algoritmo?

$$T(k) = 4*T(k/2) + O(k)$$

Será que tem como fazer melhor?

UDESC

Multiplicação

$$xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + (x_R y_R)$$

```
bigInt mul( bigInt x, bigInt y){
    bigInt xl, xr, yl, yr, p1, p2, p3;
     int n = max( x.size(), y.size() );      // número de bits do maior número
    if (n == 1) return x*y;          // se numéro de bits for 1 (retorna 1 se x = 1 e y =1).
    xl = leftMost(x, n/2); xr= rightMost(x, n/2); // bits mais a esquerda e mais a direita.
    yl = leftMost(y, n/2); yr= rightMost(y, n/2);
    p1 = mul (xl, yl);
    p2 = mul (xl+xr, yl+yr);
    p3 = mul (xr, yr);
    return          (p1 << n) + ((p2 - p1 - p3) << (n/2)) + p3;
}
```

Multiplicação

Sendo *k* o número de bits de x e y,
qual a complexidade de tempo para este algoritmo?

$$T(k) = 3*T(k/2) + O(k)$$

Será que tem como fazer melhor?

Multiplicação

Sendo *k* o número de bits de x e y,
qual a complexidade de tempo para este algoritmo?

$T(k) = 3*T(k/2) + O(k)$

Será que tem como fazer melhor?

Sim, de acordo com Cormen et al (2002) existe um algoritmo que tem complexidade **O(k log(k) log(log(k)))**

UDESC

Exercícios

Calcule qual a complexidade da função de recorrência abaixo usando teorema mestre e o método da substituição:

$$T(k) = 3*T(k/2) + O(k)$$

$$T(1) = 1$$

Escreva um algoritmo de divisão para inteiros grandes.
Determine o melhor e pior caso para esta operação e analise a sua complexidade de tempo.