

Complexidade de Algoritmos

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

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

Algoritmos com Inteiros Grandes

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 \Rightarrow (*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

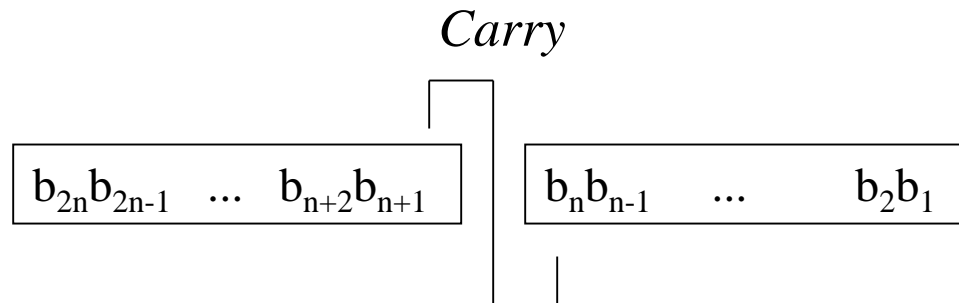
Qual é o maior valor de *carry*?

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 $O(1)$

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

$p = 32 / k = 64 \rightarrow$ serão necessárias 2 operações

$p = 32 / k = 480 \rightarrow$ serão necessárias 15 operações

$p = 32 / k = 4992 \rightarrow$ 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.

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 (k) \Rightarrow 101111011 (n)

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:

$$x = 8$$

$$y = 5$$

$$\begin{array}{c} 5 \times 8 \\ \underbrace{8 + 8 + 8 + 8 + 8}_{y \text{ vezes}} = 40 \end{array}$$

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;  
}
```

$O(k) + O(k) + \dots + O(k)$
y 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

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

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

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

$$y = 2^k$$

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

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

Multiplicação

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

$$\begin{array}{r}
 12 \\
 \times 215 \\
 \hline
 60 \quad (\text{multiplica por } 5, \text{ desloca } 0) \\
 12 \quad (\text{multiplica por } 1, \text{ desloca } 1) \\
 24 \quad (\text{multiplica por } 2, \text{ desloca } 2) \\
 \hline
 2580
 \end{array}$$

$$\begin{array}{r}
 1010 \text{ (10)} \\
 \times 1101 \text{ (13)} \\
 \hline
 1010 \text{ (multiplica por } 1, \text{ desloca } 0) \\
 0000 \text{ (multiplica por } 0, \text{ desloca } 1) \\
 1010 \text{ (multiplica por } 1, \text{ desloca } 2) \\
 1010 \text{ (multiplica por } 1, \text{ desloca } 3) \\
 \hline
 10000010 \text{ (130)}
 \end{array}$$

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

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:

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

$$\text{ou } \mathbf{O(\log^2 n)}$$

Será que tem como fazer melhor?

Multiplicação

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

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

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 xy;                // se número de bits for 1 (retorna 1 se x = 1 ou 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?

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

Será que tem como fazer melhor?

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 xy;                // se número de bits for 1 (retorna 1 se x = 1 ou 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?

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

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?

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

Será que tem como fazer melhor?

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

Exercícios

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

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

$$T(1) = 1$$

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