UDESC

Projeto e Análise 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 (110)				1	1	0	1	1	1	0
n2 (379)		1	0	1	1	1	1	0	1	1
soma		1	1	1	1	0	1	0	0	1

Quantos passos são necessários para calcular a soma?

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

$$\begin{array}{rrr} & {\scriptstyle 1\,1\quad\;\, 1\;\;\;\;\;\;\;\;\;} & \\ & 1111010 & (122) \\ + & 1100011 & (99) \\ \hline & 11011101 & (221) \end{array}$$

Quantos passos são necessários para calcular a soma?

Soma

Se o número for maior do que o tamanho do processador precisamos somente de uma memória auxiliar para o carry

Processador de 4 bits

	1 1 1 1	1	
	111	1010	(122)
+	110	1011	(107)
	1110	0101	(229)

Quantos passos são necessários para calcular a soma?

Soma

De forma genérica

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

Carry

$b_{2n}b_{2n-1} \; \ldots \; b_{n+2}b_{n+1}$ | $b_nb_{n-1} \; \ldots \; b_2b_1$

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)

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.

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 => 101111011

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

$$5\ x\ 8$$

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

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;
}
```

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

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

Multiplicação

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

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

O(y * k)

Se considerarmos que y tem k bits também, podemos dizer que
$y = 2^k$

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

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
}
```

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:

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

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

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

Multiplicação

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

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

$$x = x_L \qquad x_R = 2^{n/2} x_L + x_R$$

$$y = y_L \qquad y_R = 2^{n/2} y_L + y_R$$

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

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

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 numéro 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?

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

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

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 numéro 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)

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

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)

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

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

Referências

Algoritmos. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliford Stein. Campus.

Algorithms. Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani. McGraw Hill.

Concrete Mathematics: A Foundation for Computer Science (2nd Edition). Ronald L. Graham, Donald E. Knuth, Oren Patashnik. Addison Wesley.

M. R. Garey and D. S. Johnson. 1978. *"Strong" NP-Completeness Results: Motivation, Examples, and Implications*. J. ACM 25, 3 (July 1978)