Multiplicação de Matrizes Iterativo - Recursivo

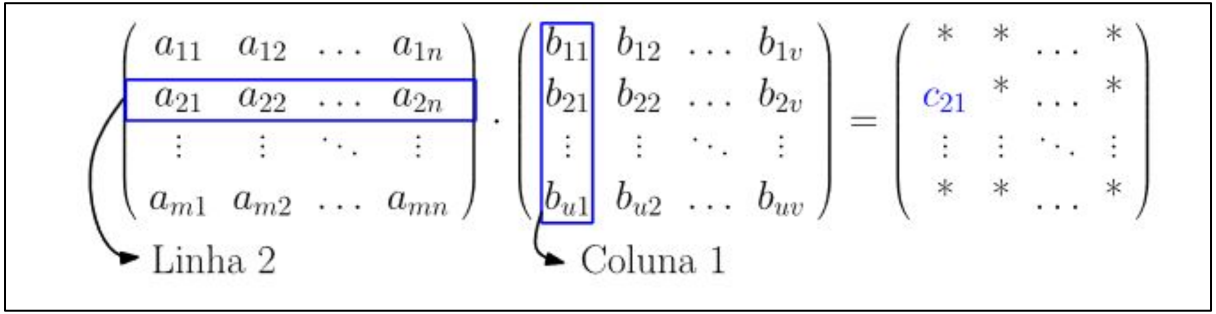
Disciplina: Análise de Algoritmos

Alunos:

- Adriana Peres de Oliveira
- Jocival Dantas Dias Junior

Multiplicação de Matrizes - Ideia Geral

Dado duas matrizes, A e B, a ideia básica deste algoritmo consiste em percorrer, para cada linha de A, todas as colunas de B, multiplicando e acumulando os resultados na matriz resultante.



Método Iterativo - Algoritmo

```
algoritmo multiplicacaoMatrizes(A, B)
var
        i, j, k: inteiro;
        C: Matriz[linha(A), coluna(B)];
Início
        for i <- 1 to linhas(A)
                for j <- 1 to colunas(B)
                        for k <- 1 to coluna(A)
                                C[i][j] <- C[i][j] + A[i][k] * B[k][j]
```

Método Iterativo - Análise Intuitiva

```
algoritmo multiplicacaoMatrizes(A, B)
1 var
2       i, j, k: inteiro;
3       C: Matriz[linha(A), coluna(B)];
4 Início
5       for i <- 1 to linhas(A)
6             for j <- 1 to colunas(B)
7                   for k <- 1 to coluna(A)
8                         C[i][j] <- C[i][j] + A[i][k] * B[k][j]
```

O algoritmo de multiplicação de Matrizes iterativo é intuitivamente O(n³) devido ao fato de ter três laços de repetição em função do tamanho das matrizes de entrada A e B.

Para provar este fato podemos analisar as linhas: 5, 6, 7 e 8.

Seja X = quantidade de linhas da matriz A,
Y = quantidade de colunas da matriz B e
Z = quantidade de colunas da matriz A e de linhas da matriz B.

Método Iterativo - Análise Intuitiva

```
algoritmo multiplicacaoMatrizes(A, B)
1 var
2         i, j, k: inteiro;
3         C: Matriz[linha(A), coluna(B)];
4 Início
5         for i <- 1 to linhas(A)
6                 for j <- 1 to colunas(B)
7                         for k <- 1 to coluna(A)
8                                 C[i][j] <- C[i][j] + A[i][k] * B[k][j]
```

Linha	Quantidade de Execuções	Análise Assintótica (O)
5	X vezes	$O(n)$
6	X * Y vezes	$O(n)*O(n) \Rightarrow O(n^2)$
7	X * Y * Z vezes	$O(n)*O(n)*O(n) \Rightarrow O(n^3)$
8	X * Y * Z vezes	$O(n)*O(n)*O(n) \Rightarrow O(n^3)$

Método Iterativo - Prova Formal

Formalmente:

$$T(X,Y,Z) = \sum_{a=1}^{X}\sum_{b=1}^{Y}\sum_{c=1}^{Z} d = d * \sum_{a=1}^{X}\sum_{b=1}^{Y}\sum_{c=1}^{Z} 1 = d * \sum_{a=1}^{X}\sum_{b=1}^{Y} Z = d * Z * \sum_{a=1}^{X}\sum_{b=1}^{Y} 1$$

$$T(X,Y,Z) = = d * Z * \sum_{a=1}^{X} Y = d * Z * Y * \sum_{a=1}^{X} 1 = d * Z * Y * X$$

Tendo que Z, Y e X estão representados em função da entrada e d é uma constante podemos dizer que o resultado deste somatório (representado de forma assintótica) é:

$$T(X,Y,Z) = O(1) * O(X) * O(Y) * O(Z) = O(X * Y * Z)$$

Método Iterativo - Prova Formal

$$T(X, Y, Z) = O(1) * O(X) * O(Y) * O(Z) = O(X * Y * Z)$$

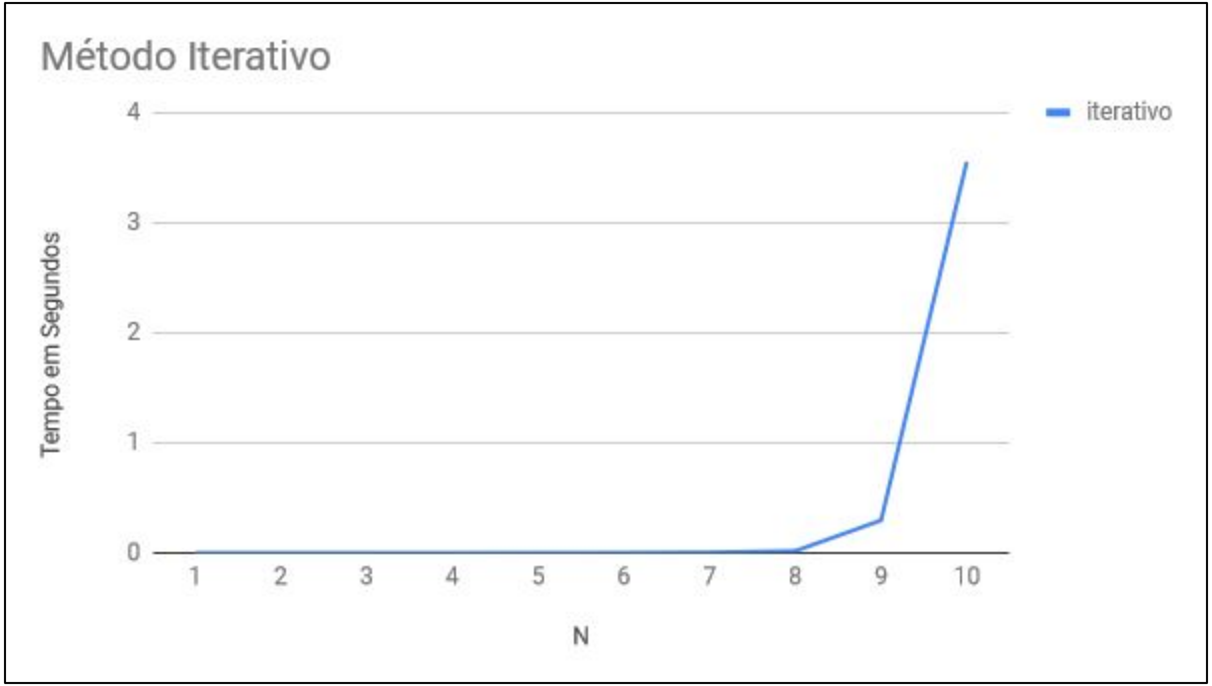
Para fins de análise, dado que o algoritmo recursivo tem a restrição de que as matrizes sejam quadradas e de tamanho 2^n, será considerado que X, Y e Z são iguais. Portanto, o resultado da análise pode ser reescrito como:

$$T(X, Y, Z) = O(1) * O(N) * O(N) * O(N) = O(N^3)$$

Como o algoritmo não realiza nenhum tipo de análise da entrada, temos que o melhor caso é igual ao pior caso e consequentemente ao caso médio. Ou seja:

$$\Omega(N^3) = \Theta(N^3) = O(N^3)$$

Método Iterativo - Tempo de Execução



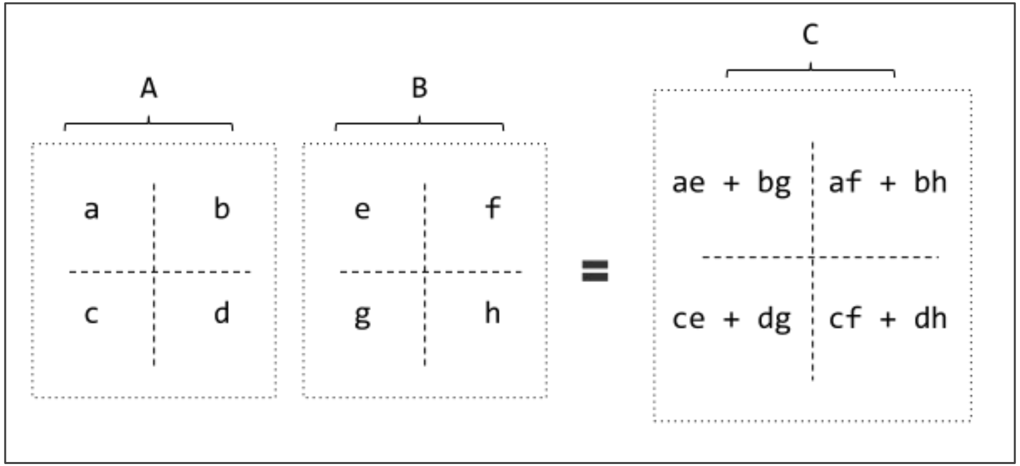
Método *Divide and Conquer* - Algoritmo

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
        + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Método *Divide and Conquer* - Algoritmo

O algoritmo recursivo de multiplicação de matrizes apresenta uma restrição de que as matrizes devem ser quadradas e de tamanho 2^N.

Essa restrição existe devido ao processo de divisão ao meio que o algoritmo recursivo executa (Linhas e Colunas ao meio).



Método *Divide and Conquer* - Caso Base

Caso base: O algoritmo recursivo será encerrado quando as matrizes de entrada apresentar tamanho igual a 1. Com custo constante.

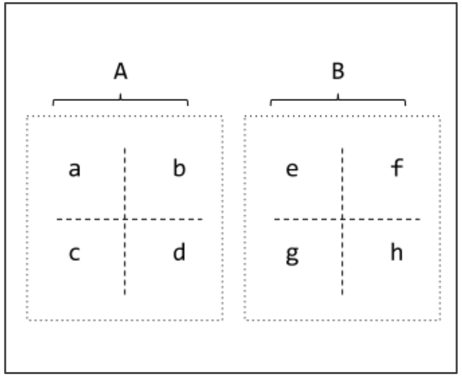
```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Método *Divide and Conquer* - Divisão

Divisão: O processo de divisão do algoritmo ocorre na linha 05 com custo constante. O processo de partição das matrizes pode ser executado apenas determinando os índices de início e fim.

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

1 $n = A.rows$
2 let C be a new $n \times n$ matrix
3 **if** $n == 1$
4 $\quad c_{11} = a_{11} \cdot b_{11}$
5 **else** partition A, B, and C
6 $\quad C_{11} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{11})
$\qquad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{21})
7 $\quad C_{12} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{12})
$\qquad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{22})
8 $\quad C_{21} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{11})
$\qquad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{21})
9 $\quad C_{22} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{12})
$\qquad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{22})
10 **return** C



Método *Divide and Conquer* - Conquista

Nas linhas 6, 7 , 8 e 9 são realizadas um total de 8 chamadas recursivas ao método, onde cada uma das chamadas vai tratar um problema de tamanho **N/2** por **N/2**. Este processo de recursão pode ser descrito como sendo 8 * T(**N/2**). Ainda nas linhas 6, 7, 8 e 9 ocorre a realização de quatro somas de matrizes resultando em um custo (N²).

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  if n == 1
4      c11 = a11 · b11
5  else partition A, B, and C
6      C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
7      C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
8      C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
9      C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
           + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10 return C
```

Método *Divide and Conquer* - Recorrência

Logo, a recorrência do algoritmo é definida como:

$$T(n) = \Theta(1),\ se\, N \leq 1$$
$$T(n) = 8 * T(N/2) + \Theta(1) + \Theta(N^2)$$

Resolvendo esta recorrência temos:

Método *Divide and Conquer* - Recorrência

$$
\begin{aligned}
T(n) &= 8 * T(N/2) + \Theta(1) + \Theta(N^2) = 8 * T(N/2) + N^2 \\
&= 8 * (8 * T(N/4) + (N/2)^2) + N^2) \\
&= 8^2 * T(N/4) + 8 * (N/2)^2 + N^2) \\
&= 8^3 * T(N/8) + 8^2 * (N/4)^2 + 8 * (N/2)^2 + N^2) \\
&\dots \\
&= 8^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} 8^i * (N/2^i)^2 \\
&= 8^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} (8/4)^i * N^2 \\
&= 8^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} 2^i * N^2 \\
&= 8^{log2\, n} * T(1) + N^2 * \sum_{i=0}^{log2\,(n-1)} 2^i \\
&= \Theta(1) + \Theta(N^2) * \sum_{i=0}^{log2\,(n-1)} 2^i \\
&= \Theta(1) + \Theta(N^2) * \Theta(N) \\
&= \Theta(N^3)
\end{aligned}
$$

Comparação Iterativo x Recursivo

Análise Algoritmo Divide and Conquer- Algoritmo de Strassen

Algoritmo de Strassen

STRASSEN (X, Y, n)

se $n = 1$ **devolva** $X \cdot Y$
$(A, B, C, D) \leftarrow$ PARTICIONE(X, n)
$(E, F, G, H) \leftarrow$ PARTICIONE(Y, n)
$P_1 \leftarrow$ STRASSEN$(A, F - H, n/2)$
$P_2 \leftarrow$ STRASSEN$(A + B, H, n/2)$
$P_3 \leftarrow$ STRASSEN$(C + D, E, n/2)$
$P_4 \leftarrow$ STRASSEN$(D, G - E, n/2)$
$P_5 \leftarrow$ STRASSEN$(A + D, E + H, n/2)$
$P_6 \leftarrow$ STRASSEN$(B - D, G + H, n/2)$
$P_7 \leftarrow$ STRASSEN$(A - C, E + F, n/2)$
$R \leftarrow P_5 + P_4 - P_2 + P_6$
$S \leftarrow P_1 + P_2$
$T \leftarrow P_3 + P_4$
$U \leftarrow P_5 + P_1 - P_3 - P_7$
devolva $P \leftarrow$ CONSTRÓI-MAT(R, S, T, U)

Strassen descobriu uma abordagem recursiva diferente que exige apenas 7 multiplicações recursivas de matrizes de tamanho n/2 * n/2 e $\Theta(N^2)$ adições e subtrações escalares, produzindo a recorrência:

$$T(n) = \Theta(1),\ se\ N \leq 1$$
$$T(n) = 7 * T(N/2) + \Theta(1) + \Theta(N^2)$$

Resolvendo a recorrência

Resolvendo a recorrência do algoritmo, com a = 7, b = 2 e f(n) = (n²) temos que:

$$
\begin{aligned}
T(n) &= 7 * T(N/2) + \Theta(1) + \Theta(N^2) = 7 * T(N/2) + N^2 \\
&= 7 * (8 * T(N/4) + (N/2)^2) + N^2) \\
&= 7^2 * T(N/4) + 7 * (N/2)^2 + N^2) \\
&= 7^3 * T(N/8) + 7^2 * (N/4)^2 + 7 * (N/2)^2 + N^2) \\
&\dots \\
&= 7^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} 7^i * (N/2^i)^2 \\
&= 7^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} (7/4)^i * N^2 \\
&= 7^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} (7/4)^i * N^2 \\
&= 7^{log2\, n} * T(1) + \sum_{i=0}^{log2\,(n-1)} (7/4)^i * N^2 \\
&= 7^{log2\, n} * \Theta(1) + \Theta((7/4)^{log2\,(n-1)} * N^2) \\
&= 7^{log2\, n} * \Theta(1) + \Theta((7/4)^{log2\,(n-1)} * N^2) \\
&= 7^{log2\, n} * \Theta(1) + \Theta((7/4)^{log2\,(n-1)} * N^2) \\
&= 7^{log2\, n} * \Theta(1) + \Theta(7^{\,log2\, n}) \\
&= \Theta(7^{log2\, n})
\end{aligned}
$$

$$
\begin{aligned}
7^{log2\, n} &= 7^{(log7\, n\, /\, log7\, 2)} \\
&= (7^{log7\, n})^{(1/log7\, 2)} \\
&= n^{(log2\, 7\, /\, log2\, 2)} \\
&= n^{log2\, 7} \\
&= n^{log2\, 7}
\end{aligned}
$$

Nota:

O custo total do algoritmo é $\Theta(N^{\mathbf{2,81}})$ sendo assim, ligeiramente melhor que os demais algoritmos apresentados neste trabalho.

Estado da Arte

- $O(n^3)$, naive approach
- $O(n^{2.808})$, Strassen (1969)
- $O(n^{2.796})$, Pan (1978)
- $O(n^{2.522})$, Schönhage (1981)
- $O(n^{2.517})$, Romani (1982)
- $O(n^{2.496})$, Coppersmith and Winograd (1982)
- $O(n^{2.479})$, Strassen (1986)
- $O(n^{2.376})$, Coppersmith and Winograd (1989)
- $O(n^{2.374})$, Stothers (2010)
- $O(n^{2.3728642})$, V. Williams (2011)
- $O(n^{2.3728639})$, Le Gall (2014)

Obrigado!