UNIVASSOURAS

CAMPUS UNIVERSITÁRIO
DE SAQUAREMA

UNIVASSOURAS

Curso de Graduação em Engenharia de Software

ENGENHARIA DE SOFTWARE
UNIVASSOURAS SAQUAREMA

Estrutura de Dados

Aula 3 - Complexidade

Prof. Dr. André Saraiva
Doutor em Ciência da Computação
Mestre em sistemas Compurtacionais
Especialista em Arquitetura e Projeto de Cloud Computing

UNIVASSOURAS
CAMPUS UNIVERSITÁRIO
DE SAQUAREMA

- ***Temas para o trabalho da P1 – Em Grupo***
  - Árvores Binárias
  - Árvores AVL
  - Árvores Rubro-Negras
  - Árvore Digital (Tries ou árvore de prefixos)
  - Árvores de Huffman

✓ ***Complexidade de Algoritmos***

✓ Uma característica importante de qualquer algoritmo é seu tempo de execução

- é possível determiná-lo através de métodos empíricos, considerando-se entradas diversas
- é também possível obter este tempo a partir de métodos analíticos

✓ ***Complexidade de Algoritmos***

✓ Métodos Empíricos

- Baseados na medição do tempo de execução do algoritmo.
- Dependem do computador, linguagem de programação, compilador, dados, etc., utilizados na experiência.
- Admitem um tratamento estatístico.

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- objetivo: determinar uma expressão matemática que traduza o comportamento de tempo de um algoritmo.
- o tempo de execução independente:
  - do método utilizado
  - da linguagem e compiladores empregados
  - e das condições locais de processamento

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- Somente o comportamento assintótico é avaliado
- Não serão consideradas constantes aditivas ou multiplicativas na expressão considerada
- *Qual a variável em relação à qual a expressão matemática avaliará o tempo de execução?*
  - Um algoritmo funciona a partir de uma entrada para produzir uma saída dentro de um tempo
  - Fornecer o tempo de execução em função da entrada

✓ ***Complexidade de Algoritmos***

✓ **Métodos analíticos**

- O processo de execução de um algoritmo pode ser dividido em etapas elementares – passos
  - um número fixo de operações básicas cujo tempo de execução é considerado constante
  - a operação básica de maior frequência de execução do algoritmo é denominada **operação dominante**

✓ ***Complexidade de Algoritmos***

✓ **Métodos analíticos**

- O processo de execução de um algoritmo pode ser dividido em etapas elementares – passos

  - Desconsiderar constantes aditivas e multiplicativas.
  - Exemplo: $3.n^2+5.n-7$ e $12.n^2+11.n+32$ seriam expressões equivalentes.

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- **Desconsiderar constantes aditivas e multiplicativas**.
- Exemplo: 3.n^2+5.n-7 e 12.n^2+11.n+32 seriam expressões equivalentes.

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- Inversão de uma sequência

```
fim = n/2;
for (i=0; i<fim; i++) {
    temp = S[i];
    S[i] = S[n-1-i];
    S[n-1-i] = temp;
}
```

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- Inversão de uma sequência
- n = 5 ➔ troca S[i] por S[n-1-i]
  - fim = 2
  - i = 0 ➔ troca S[0] por S[5-1-0] (S[4])
  - i = 1 ➔ troca S[1] por S[5-1-1] (S[3])

```
fim = n/2;
for (i=0; i<fim; i++)
{
      temp = S[i];
      S[i] = S[n-1-i];
      S[n-1-i] =
      temp;
}
```

Inicial	➔	final
M A R I A		A I R A M

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- Inversão de uma sequência
- n = 50 ➔ troca S[i] por S[n-1-i]
  - fim = 25
  - i = 0 ➔ troca S[0] por S[50-1-0] (S[49])
  - i = 1 ➔ troca S[1] por S[50-1-1] (S[48])
  - i = 2 ➔ troca S[2] por S[50-1-2] (S[47])
  - .....
  - i = 23 ➔ troca S[23] por S[50-1-23] (S[26])
  - i = 24 ➔ troca S[24] por S[50-1-24] (S[25])

```
fim = n/2;
for (i=0; i<fim; i++)
{
      temp = S[i];
      S[i] = S[n-1-i];
      S[n-1-i] =
      temp;
}
```

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- o algoritmo executa exatamente as mesmas operações para sequências de tamanho **n**
  - cada passo corresponde à troca de posição entre dois elementos da sequência
    - a execução das três atribuições
  - o número de passos é igual ao número de vezes que executa o bloco *for* ⇒ n/2, n>1

```
fim = n/2;
for (i=0; i<fim; i++)
{
        temp = S[i];
        S[i] = S[n-1-i];
        S[n-1-i] =
        temp;
}
```

✓ ***Complexidade de Algoritmos***

✓ Métodos analíticos

- Problema: determinar a soma C e o produto D de duas matrizes dadas A e B.
  - A = (aij) e B = (bij) ambas n x n
  - C e D também serão matrizes n x n
  - seus elementos podem ser calculados como:

$$\Rightarrow cij = aij + bij$$

$$\Rightarrow dij = \Sigma aik * bkj$$

$$1 \leq k \leq n$$

✓ ***Complexidade de Algoritmos***

✓ Soma de Matrizes

⇒ cij = aij + bij

$$\begin{vmatrix} 2 & 3 & 5 \\ 0 & 1 & 2 \\ 3 & 0 & 1 \end{vmatrix} + \begin{vmatrix} 0 & 0 & 2 \\ 3 & -1 & 4 \\ -4 & 1 & 0 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 7 \\ 3 & 0 & 6 \\ -1 & 1 & 1 \end{vmatrix}$$

```
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                cij = aij + bij ;
```

✓ ***Complexidade de Algoritmos***

✓ Soma de Matrizes

⇒ cij = aij + bij

$$\begin{vmatrix} 2 & 3 & 5 \\ 0 & 1 & 2 \\ 3 & 0 & 1 \end{vmatrix} + \begin{vmatrix} 0 & 0 & 2 \\ 3 & -1 & 4 \\ -4 & 1 & 0 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 7 \\ 3 & 0 & 6 \\ -1 & 1 & 1 \end{vmatrix}$$

```
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                cij = aij + bij ;
```

Quantos passos temos na soma de matrizes?

✓ ***Complexidade de Algoritmos***

✓ Soma de Matrizes

⇒ cij = aij + bij

$$\begin{vmatrix} 2 & 3 & 5 \\ 0 & 1 & 2 \\ 3 & 0 & 1 \end{vmatrix} + \begin{vmatrix} 0 & 0 & 2 \\ 3 & -1 & 4 \\ -4 & 1 & 0 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 7 \\ 3 & 0 & 6 \\ -1 & 1 & 1 \end{vmatrix}$$

```
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                cij = aij + bij ;
```

n^2 passos!!!

✓ ***Complexidade de Algoritmos***

✓ Multiplicação de Matrizes

⇒ dij = Σaik * bkj

1≤k ≤ n

$$\begin{vmatrix} 2 & 3 & 5 \\ 0 & 1 & 2 \\ 3 & 0 & 1 \end{vmatrix} . \begin{vmatrix} 0 & 0 & 2 \\ 3 & -1 & 4 \\ -4 & 1 & 0 \end{vmatrix} = \begin{vmatrix} -11 & 2 & 16 \\ -5 & 1 & 4 \\ -4 & 1 & 6 \end{vmatrix}$$

```
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                dij = 0;
                for(k=0; k<n; k++)
                        dij = dij + aik* bkj;
```

Quantos passos temos na soma de matrizes?

✓ ***Complexidade de Algoritmos***

✓ Multiplicação de Matrizes

⇒ dij = Σaik * bkj

1≤k ≤ n

$$\begin{vmatrix} 2 & 3 & 5 \\ 0 & 1 & 2 \\ 3 & 0 & 1 \end{vmatrix} . \begin{vmatrix} 0 & 0 & 2 \\ 3 & -1 & 4 \\ -4 & 1 & 0 \end{vmatrix} = \begin{vmatrix} -11 & 2 & 16 \\ -5 & 1 & 4 \\ -4 & 1 & 6 \end{vmatrix}$$

```
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                dij = 0;
                for(k=0; k<n; k++)
                        dij = dij + aik* bkj;
```

n^3 passos!!!

✓ ***Complexidade de Algoritmos***

- Seja A um algoritmo
  - $\{E_1, ...E_m\}$ o conjunto de todas as entradas possíveis de A
  - seja t_i o número de passos efetuados por A quando a entrada for E_i

✓ ***Complexidade de Algoritmos***

- Complexidade do pior caso:

$$max_{Ei \in E} \{ti\}$$

- Complexidade do melhor caso:

$$min_{Ei \in E} \{ti\}$$

- Complexidade do caso médio:

$$\Sigma_{1 \leq i \leq m} p_i t_i$$

- pi é a probabilidade de ocorrência da entrada Ei

✓ ***Complexidade de Algoritmos***

algoritmo	complexidade de pior caso	complexidade de melhor caso	complexidade de caso médio
inversão de sequências	n	n	n
soma de matrizes	n^2	n^2	n^2
produto de matrizes	n^3	n^3	n^3

➔ constantes aditivas ou multiplicativas são irrelevantes

✓ ***Complexidade de Algoritmos***

- Sejam $A = (aik), B = (bkj)$, duas matrizes, tais que $1 \leq i \leq n, 1 \leq k \leq m$ e $1 \leq j \leq p$. Escrever os algoritmos para calcular $C = A + B$ e $D = A . B$.
- Determinar as suas complexidades.
- Os valores de n, m e p podem ser quaisquer?

✓ ***Complexidade de Algoritmos***

- **algoritmo: soma de matrizes**

```
for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
                c[i][j] = a[i][j] + b[i][j]
```

- Complexidade: n.m

✓ ***Complexidade de Algoritmos***

- **algoritmo: produto de matrizes**

```
for (int i = 0; i < n; ++i)
        for (int j = 0; j < p; ++j)
                for (int k = 0; k < m; ++k)
                        d[i][j] += a[i][k] * b[k][j]
```

- Complexidade: n.m.p

✓ ***Complexidade de Algoritmos***

Vamos começar a nos divertir???

Implemente em C/C++ um algoritmo de soma e uma algoritmo de multiplicação de matrizes

Contato

Professor:
André Saraiva, MSc

E-mail:
andre.saraiva@univassouras.edu.br