Módulo 3

Estruturas de Dados

Lição 1

Conceitos Básicos e Notações

Versão 1.0 - Mai/2007

Autor
Joyce Avestro

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Reginald Hutcherson
Rebecca Ong
John Paul Petines
Sang Shin
Raghavan Srinivas
Matthew Thompson

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Anna Carolina Ferreira da Rocha
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bárbara Angélica de Jesus Barbosa
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alexandre de Sene
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Everaldo de Souza Santos
Fabrício Ribeiro Brigagão
Fernando Antonio Mota Trinta
Frederico Dubiel
Givailson de Souza Neves
Jacqueline Susann Barbosa
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Kefreen Ryenz Batista Lacerda
Leonardo Leopoldo do Nascimento
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luís Carlos André
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Marcus Borges de S. Ramos de Pádua
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Renato Alves Félix
Renê César Pereira
Reyderson Magela dos Reis
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Fernandes Suguiura
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Silvio Sznifer
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Na criação de solução para os processos de resolução de problemas, existe a necessidade de representação dos dados em nível mais alto a partir de informações básicas e estruturas disponíveis em nível de máquina. Existe também a necessidade de se sintetizar o algoritmo a partir das operações básicas disponíveis em nível de máquina para manipular as representações em alto nível. Estas duas características são muito importantes para obtenção do resultado desejado. **Estruturas de dados** (*Data Structures*) são necessárias para a representação dos dados, enquanto que os **algoritmos** precisam operar no dado para obter a saída correta.

Nesta lição iremos discutir os conceitos básicos por detrás do **Processo Resolução de Problemas** (*Problem Solving*), tipos de dados, tipos de dados abstratos, algoritmos e suas propriedades, métodos de endereçamento, funções matemáticas e complexidade dos algoritmos.

Ao final desta lição, o estudante será capaz de:

- Explicar os processos de **resolução de problemas**
- Definir **tipos de dados** (*data type*), **tipos de dados abstratos** (*abstract data type*) e **estrutura de dados** (*data structure*)
- Identificar as propriedades de um **algoritmo**
- Diferenciar os dois **métodos de endereçamento** – endereçamento computado e endereçamento por link
- Utilizar as **funções matemáticas** básicas para analisar algoritmos
- Mensurar a **complexidade dos algoritmos** expressando a eficiência em termos de complexidade de tempo e notação Big-O

2. Processo de resolução de problemas

Programação é um processo de resolução de problemas. Por exemplo, o problema é identificado, o dado a ser manipulado e trabalhado é distinguido e o resultado esperado é determinado. Isso é implementado em uma máquina chamada computador e as informações fornecidas para ela são utilizadas para solucionar um dado problema. O processo de resolução de problemas pode ser visto em termos de **Domínio de Problema** (*Domain Problem*), máquina e solução.

Domínio do problema inclui a **entrada** (*input*), ou os dados brutos, em um processo, e a **saída** (*output*), ou os dados processados. Por exemplo, na classificação de um conjunto de números aleatórios, o dado bruto é um conjunto de números na ordem original, aleatórios, e o dado processado são os números em ordem classificada, crescente, por exemplo.

O **domínio de máquina** (*machine domain*) consiste em meios de armazenamento (*storage medium*) e unidades processadas. Os meios de armazenamento – *bits*, *bytes*, etc – consistem na combinação de *bits* em seqüências que são endereçáveis como unidade. As unidades processadas nos permitem melhorar o desempenho de operações básicas que incluem a aritmética, a comparação e assim por diante.

O **domínio de solução** (*solution domain*), em outras palavras, liga os domínios de problema e de máquina. É no domínio de solução que as estruturas de dados de alto nível e os algoritmos são afetados.

3. Tipo de Dado, Tipo de Dado Abstrato e Estrutura de Dados

Tipo de dado (*data type*) refere-se à classificação do dado que um atributo pode assumir, armazenar ou receber em uma linguagem de programação e para a qual as operações são automaticamente fornecidas. Em Java, os dados primitivos são:

Palavra-chave	***Descrição***
byte	Inteiro do tamanho de um byte
short	Inteiro curto
int	Inteiro
long	Inteiro longo
float	Ponto flutuante de precisão simples
double	Ponto flutuante de precisão dupla
char	Um único caractere
boolean	Valor booleano (verdadeiro ou falso)

Tabela 1: Dados primitivos

Tipo de dado abstrato (*Abstract Data Type* – ADT) é um modelo matemático contendo uma coleção de operadores. Ele especifica um tipo de dado armazenado, *o que* a operação faz, mas não como é feito. Um **ADT** pode ser expresso por uma **interface** que contém apenas uma lista de métodos. Por exemplo, esta é uma interface para a *stack* ADT:

```
public interface Stack{
  public int size();        // retorna o tamanho da stack
  public boolean isEmpty(); // verifica se está vazia
  public Object top() throws StackException;
  public Object pop() throws StackException;
  public void push(Object item) throws StackException;
}
```

Estrutura de dados é a implementação de um **TDA** em termos de tipos de dados ou outras estruturas de dados. Uma estrutura de dados é modelada através de **classes**. Classes especificam como as operações são executadas. Para implementar um **TDA** como uma estrutura de dados, uma interface é implementada através de uma classe.

Abstração e representação ajudam-nos a entender os princípios por detrás dos grandes sistemas de *software*. Encapsulamento de informação pode ser utilizada junto com abstração para particionar um sistema grande em subsistemas menores com interfaces simples que são mais fáceis de entender e utilizar.

4. Algoritmo

Algoritmo é um conjunto finito de instruções que, se seguidas corretamente, completam uma determinada tarefa. Possui cinco importantes propriedades: finito, definido, entrada, saída e efetivo. **Finito** quer dizer que um algoritmo sempre terá um fim após um número finito de passos. **Definido** é a garantia de que todos os passos do algoritmo foram precisamente definidos. Por exemplo: "dividir por um número x" não é suficiente. O número x deve ser precisamente definido, ou seja, x deve ser inteiro e positivo. **Entrada** é o domínio do algoritmo que pode ser nenhum (zero) ou vários. **Saída** é o conjunto de um ou mais resultados que também é chamado de alcance do algoritmo. **Efetivo** é a garantia de que todas as operações do algoritmo são suficientemente simples de maneira que também possam, a princípio, ser executadas, em um tempo exato e finito, por uma pessoa utilizando papel e caneta.

Considere o seguinte exemplo:

```
public class Minimum {
  public static void main(String[] args) {
    int a[] = { 23, 45, 71, 12, 87, 66, 20, 33, 15, 69 };
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
      if (a[i] < min) min = a[i];
    }
    System.out.println("The minimun value is: " + min);
  }
}
```

A classe acima obtém o valor mínimo de um *array* de inteiros. Não há entrada de dados por parte do usuário uma vez que o *array* já está pronto dentro da classe. Para cada propriedade de entrada e saída cada passo da classe é precisamente definido; neste ponto esta poderá ser **definida**. A declaração do laço *for* e suas respectivas saídas terão um número finito de execução. Logo, a propriedade **finito** é satisfeita. E quando executado, a classe retornará o valor mínimo entre os valores do *array*, e por isso é dito **efetivo**.

Todas as propriedades devem ser garantidas na construção de um algoritmo.

5. Métodos de Endereçamento

Na criação de uma estrutura de dados é importante definir como acessar estes dados. Isto é determinado pelo método de acesso a dados. Existem dois tipos de métodos de endereçamento – método calculado e método de endereçamento – computado e por *link*.

5.1. Método de Endereçamento Computado

O método de endereçamento computado é utilizado para acessar os elementos de uma estrutura em um espaço pré-alocado. É essencialmente estático. Um *array*, por exemplo:

```
int x[] = new int[10];
```

Um item de dado pode ser acessado diretamente pelo índice de onde o dado está armazenado.

5.2. Método de Endereçamento por Link

Este método de endereçamento fornece um mecanismo de manipulação dinâmica de estruturas, onde o tamanho e a forma não são conhecidos de antemão, ou que são alterados durante a execução. O importante para este método é o conceito de ***node*** contido nestes dois campos: INFO e LINK.

Figura 1: Estrutura de node

Em Java:

```
public class Node {
  public Object info;
  public Node link;

  public Node(Object o) {
    info = o;
  }
  public Node(Object o, Node n) {
    info = o;
    link = n;
  }
}
```

5.2.1. Alocação de ligação: O Pool de Memória

O *pool de memória* é a fonte dos *nodes*, onde são construídas as estruturas de ligação. Também são conhecidas como lista de espaços disponíveis (ou *nodes*) ou simplesmente ***lista disponível:***

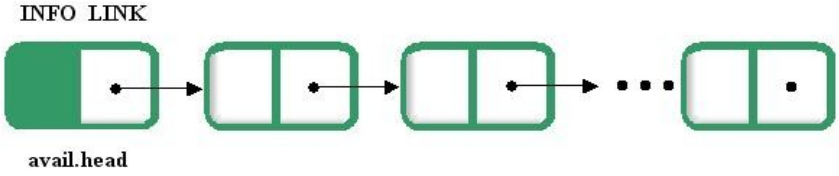

Figura 2: Lista Disponível

A seguir uma classe Java chamada *AvailList*:

```
public class AvailList {
  private Node head;

  public AvailList() {
    head = null;
  }

  public AvailList(Node n){
    head = n;
  }
}
```

Criando uma lista disponível através de uma simples declaração:

```
AvailList avail = new AvailList();
```

5.2.2. Dois Procedimentos Básicos

Os dois procedimentos básicos que manipulam a lista disponível são *getNode* e *setNode*, que obtém e retornam um *node*, respectivamente.

O método seguinte na classe *AvailList* obtém um *node* da lista disponível:

```
public Node getNode() {
  return head;
}
```

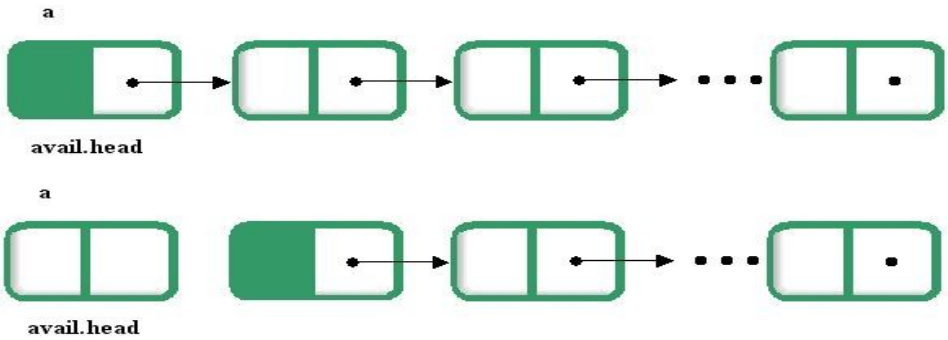

Figura 3: Obtém um node

enquanto o método a seguir na classe *Avail* retorna um node para a lista disponível:

```
public void setNode(Node n) {
  n.link = head.link;  // Adiciona o novo node no início da lista disponível
  head.link = n;
}
```

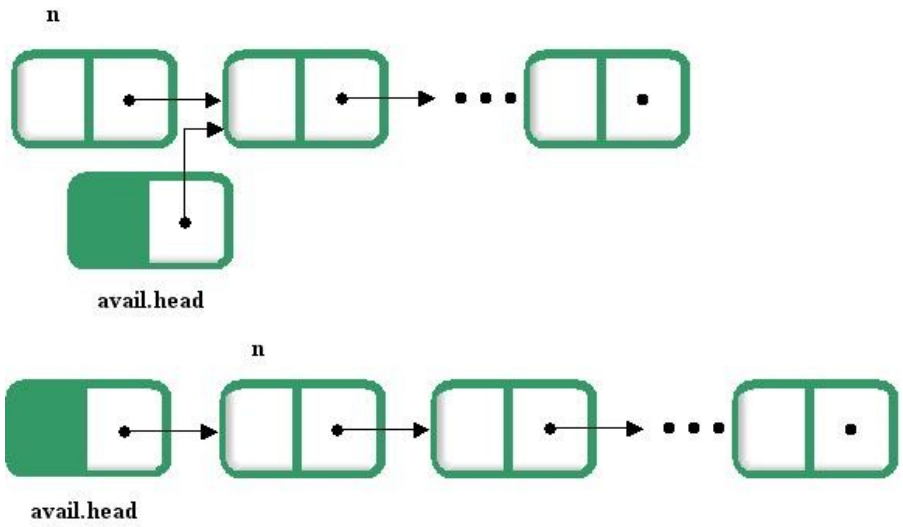

Figura 4: Retorna um node

Os dois métodos poderiam ser usados por estruturas de dados que usam alocação de ligação para pegar os *nodes* e retorná-los para o *pool* de memória. E como teste final teremos a seguinte classe:

```
public class TestNodes {
  public static void main(String [] args) {
    Node n1 = new Node("1");
    Node n2 = new Node("2", n1);
    Node n3 = new Node("3", n2);
    Node n4 = new Node("4", n3);
    AvailList avail = new AvailList(n4);

    System.out.println(avail.getNode().info);
    System.out.println(avail.getNode().link.info);
    System.out.println(avail.getNode().link.link.info);
    System.out.println(avail.getNode().link.link.link.info);
  }
}
```

6. Funções Matemáticas

Funções matemáticas são úteis na criação e na análise de algoritmos. Nesta seção, algumas das funções mais básicas e mais comumente usadas e suas propriedades serão mostradas.

- ***Floor* de x** – o maior inteiro menor que ou igual a x, onde x é um número real qualquer.

  Notação: ⌊ x ⌋

  ex. ⌊ 3.14 ⌋ = 3 ⌊ 1/2 ⌋ = 0 ⌊ -1/2 ⌋ = - 1

- ***Ceil* de x** – é o menor inteiro maior que ou igual a x, onde x é um número real qualquer.

  Notação : ⌈ x ⌉

  ex. ⌈ 3.14 ⌉ = 4 ⌈ 1/2 ⌉ = 1 ⌈ -1/2 ⌉ = 0

- **Módulo -** Dados quaisquer dois números reais x e y, x mod y é definido como

  x mod y = x se y = 0
  = x - y * ⌊ x / y ⌋ se y <> 0

  ex. 10 mod 3 = 1 24 mod 8 = 0 -5 mod 7 = 2

6.1. Identidades

O que segue são identidades relacionadas às funções matemáticas definidas acima:

- ⌈ x ⌉ = ⌊ x ⌋ se e somente se x é um inteiro
- ⌈ x ⌉ > ⌊ x ⌋ se e somente se x não é um inteiro
- ⌊- x ⌋ = - ⌈ x ⌉
- ⌊ x ⌋ + ⌊ y ⌋ <= ⌊ x + y ⌋
- x = ⌊ x ⌋ + x mod 1
- z (x mod y) = zx mod zy

7. Complexidade de Algoritmos

Diversos algoritmos podem ser criados para resolver um único problema. Estes algoritmos podem variar no modo de obter, processar e dar saída nos dados. Com isso, eles podem ter diferença significativa em termos de performance e utilização de memória. É importante saber como analisar os algoritmos, e saber como medir a eficiência dos algoritmos ajuda bastante no processo de análise.

7.1. Eficiência de Algoritmos

A eficiência dos algoritmos é medida através de dois critérios: utilização de espaço e eficiência de tempo. Utilização de espaço é a quantidade de memória requerida para armazenar dados enquanto eficiência de tempo é a quantidade de tempo gasta para processar os dados.

Antes de podermos medir a eficiência de tempo de um algoritmo, temos que obter o tempo de execução. **O tempo de execução** é a quantidade de tempo gasto para se executar as instruções de um dado algoritmo. Ele depende do computador (*hardware*) sendo usado. Para exibir o tempo de execução, usamos a seguinte notação:

T(n), onde T é a função e n o tamanho da entrada.

Existem vários fatores que afetam o tempo de execução. Eles são:

- Tamanho da entrada
- Tipo da instrução
- Velocidade da máquina
- Qualidade do código-fonte da implementação do algoritmo
- Qualidade do código de máquina gerado pelo compilador

7.2. A Notação Big-O

Embora T(n) forneça a quantidade real de tempo na execução de um algoritmo, é mais fácil classificar as complexidades de algoritmos utilizando uma notação mais abrangente, a *notação Big-O (ou simplesmente O).* **T(n)** cresce a uma taxa proporcional a **n** e dessa forma **T(n)** é dita como tendo *"ordem de magnitude n"* denotada pela notação ***O***:

$$T(n) = O(n)$$

Esta notação é usada para descrever a complexidade de tempo ou espaço de um algoritmo. Ela fornece uma medida aproximada do tempo de computação de um algoritmo para um grande número de dados de entrada. Formalmente, a notação ***O*** é definida como:

$$g(n) = O(f(n)) \text{ se existem duas constantes c e } n_0 \text{ tais que}$$
$$|g(n)| <= c * |f(n)| \text{ para todo } n >= n_0$$

A seguir temos exemplos de tempos de computação sobre análise de algoritmos:

Big-O	***Descrição***	***Algoritmo***
O(1)	Constante	
$O(\log_2 n)$	Logarítmica	Busca Binária
O(n)	Linear	Busca Seqüencial
$O(n \log_2 n)$		Heapsort
$O(n^2)$	Quadrática	Inserção Ordenada
$O(n^3)$	Cúbica	Algoritmo de Floyd

Big-O	*Descrição*	*Algoritmo*
O(2^n)	Exponencial	

Tabela 2: Tempos de computação sobre análise de algoritmos

Para tornar clara a diferença, vamos efetuar a comparação baseada no tempo de execução onde n=100000 e a unidade de tempo = 1 mseg:

F(n)	*Tempo de Execução*
$\log_2 n$	19.93 microssegundos
n	1.00 segundos
n $\log_2 n$	19.93 segundos
n^2	11.57 dias
n^3	317.10 séculos
2^n	Eternidade

Tabela 3: Tempo de execução

7.3. Operações sobre a Notação O

1. Regra para Adição

Suponha que $T_1(n) = O(f(n))$ e $T_2(n) = O(g(n))$.
Então, $t(n) = T_1(n) + T_2(n) = O(max(f(n), g(n)))$.

Prova: Por definição da notação O,
$T_1(n) \leq c_1 f(n)$ para $n \geq n_1$ e
$T_2(n) \leq c_2 g(n)$ para $n \geq n_2$.

Seja $n_0 = max(n_2, n_2)$. Então

$$\begin{aligned} T_1(n) + T_2(n) &\leq c_1 f(n) + c_2 g(n) && n \geq n_0. \\ &\leq (c_1 + c_2) \max(f(n),g(n)) && n \geq n_0. \\ &\leq c \max (f(n), g(n)) && n \geq n_0. \end{aligned}$$

Sendo assim, **$T(n) = T_1(n) + T_2(n) = O(max(f(n), g(n)))$.**

Por exemplo, 1. $T(n) = 3n^3 + 5n^2 = O(n^3)$
2. $T(n) = 2^n + n^4 + n\log_2 n = O(2^n)$

2. Regra para Multiplicação

Suponha que T1(n) = O(f(n)) e T2(n) = O(g(n)).
Então, T(n) = T1(n) * T2(n) = O(f(n) * g(n)).

Por exemplo, considere o algoritmo abaixo:

```
for (int i=1; i < n-1; i++)
  for (int i=1; i <= n; i++)
    // as iterações são executadas O(1) vezes
```

Já que as iterações no laço mais interno são executadas:

n + n-1 + n-2 + ... + 2 + 1 vezes,

então

$n(n+1)/2 = n^2/2 + n/2$
$= O(n^2)$

Exemplo: Considere o trecho de código abaixo:

```
for (i=1; i <= n, i++)
  for (j=1; j <= n, j++)
    // iterações que são executados O(1) vezes
```

Já que as iterações no laço mais interno serão executadas n + n-1 + n-2 + ... + 2 + 1 vezes, então o tempo de execução será:

$n(n+1)/2 = n^2/2 + n/2$
$= O(n^2)$

7.4. Análise de Algoritmos

Exemplo 1: Revisitação Mínima

```
public class Minimum {
  public static void main(String [] args) {
    int a[] = {23, 45, 71, 12, 87, 66, 20, 33, 15, 69};
    int min = a[0];
    for (int i = 0; i < a.length; i++) {
      if (a[i] < min)
        min = a[i];
    }
    System.out.println("Minimun value is: " + min);
  }
}
```

No algoritmo, as declarações de *a* e *min* terão tempos constantes. O tempo constante da sentença *if* no loop *for* serão executadas **n** vezes, onde *n* é o número de elementos do array **a.** A última linha também será executada em tempo constante.

Linha	***#Vezes executada***
4	1
5	1
6	n+1
7	n
9	1

Tabela 4: Quantidade de execuções por linha

Usando a regra para adição, temos:

T(n) = 2n +4 = O(n)

Já que g(n) <= c f(n) para n >= n_0, então

```
2n + 4 <= cn
2n + 4 <= c
  ---------
     n

2 + 4/n <= c
```

Assim c = 4 e n0 = 3.

Seguem abaixo as regras gerais para se determinar o tempo de execução de um algoritmo:

- Laços FOR
  - Tempo de execução da declaração dentro do laço FOR vezes o número de iterações.
- Laços FOR ANINHADOS
  - A Análise é feita a partir do laço mais interior para fora. O tempo total de execução de uma declaração dentro de um grupo de laço FOR é o tempo de execução da declaração, multiplicado pelo produto dos tamanhos de todos os laços FOR.
- DECLARAÇÕES CONSECUTIVAS
  - A declaração com o maior tempo de execução.
- Condicional IF/ELSE
  - Quanto maior o tempo de execução do teste, maior será o tempo de execução do bloco condicional.

8. Exercícios

a) *Funções Piso, Teto e Módulo.* Compute para os valores resultantes:

a) $\lfloor -5.3 \rfloor$
b) $\lfloor 6.14 \rfloor$
c) 8 mod 7
d) 3 mod –4
e) –5 mod 2
f) 10 mod 11
g) $\lceil (15 \bmod -9) + \lfloor 4.3 \rfloor \rceil$

b) Qual é a complexidade de tempo do algoritmo com os seguintes tempos de execução?

a) $3n^5 + 2n^3 + 3n + 1$
b) $n^3/2+n^2/5+n+1$
c) n^5+n^2+n
d) $n^3 + \lg n + 34$

c) Imagine que tenhamos duas partes em um algoritmo, sendo que a primeira parte toma $T(n_1)=n^3+n+1$, tempo para executar e a segunda parte toma $T(n_2) = n^5+n^2+n$. Qual é a complexidade do algoritmo, se a parte 1 e a parte 2 forem executadas uma de cada vez?

d) Ordene as seguintes complexidades de tempo em ordem ascendente.

$0(n \log_2 n)$	$0(n^2)$	$0(n)$	$0(\log_2 n)$	$0(n^2 \log_2 n)$
$0(1)$	$0(n^3)$	$0(n^n)$	$0(2^n)$	$0(\log_2 \log_2 n)$

e) Qual é o tempo de execução e a complexidade de tempo do algoritmo abaixo?

```
void warshall(int A[][], int C[][], int n){
  for (int i=1; i<=n; i++)
    for (int j=1; j<=n; j++)
      A[i][j] = C[i][j];
  for (int i=1; i<=n; i++)
    for (int j=1; j<=n; j++)
      for (int k=1; k<=n; k++)
        if (A[i][j] == 0)
          A[i][j] = A[i][k] & A[k][j];
}
```

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.