Módulo 2

Introdução à Programação II

Lição 6

Algoritmos de Ordenação

Versão 1.0 - Mar/2007

Autor
Rebecca Ong

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Rebecca Ong
John Paul Petines
Sun Microsystems
Sun Philippines

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
Andro Márcio Correa Louredo
Antoniele de Assis Lima
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alberto Vitorino de Almeida
Carlos Alexandre de Sene
Carlos André Noronha de Sousa
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Cleyton Artur Soares Urani
Cristiano Borges Ferreira
Cristiano de Siqueira Pires
Derlon Vandri Aliendres
Fabiano Eduardo de Oliveira
Fábio Bombonato
Fernando Antonio Mota Trinta
Flávio Alves Gomes
Francisco das Chagas
Francisco Marcio da Silva
Gilson Moreno Costa
Givailson de Souza Neves
Gustavo Henrique Castellano
Hebert Julio Gonçalves de Paula
Heraldo Conceição Domingues
Hugo Leonardo Malheiros Ferreira
Ivan Nascimento Fonseca
Jacqueline Susann Barbosa
Jader de Carvalho Belarmino
João Aurélio Telles da Rocha
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Leonardo Borges de Melo
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Lafaiete de Sá Guimarães
Leandro Silva de Morais
Leonardo Leopoldo do Nascimento
Leonardo Pereira dos Santos
Leonardo Rangel de Melo Filardi
Lucas Mauricio Castro e Martins
Luciana Rocha de Oliveira
Luís Carlos André
Luís Octávio Jorge V. Lima
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Manoel Cotts de Queiroz
Marcello Sandi Pinheiro
Marcelo Ortolan Pazzetto
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio Azevedo Gamarra
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Néres Chaves Rebouças
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo José Lemos Costa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Pedro Henrique Pereira de Andrade
Renato Alves Félix
Renato Barbosa da Silva
Reyderson Magela dos Reis
Ricardo Ferreira Rodrigues
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Pereira Machado
Rodrigo Rosa Miranda Corrêa
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Sergio Pomerancblum
Silvio Sznifer
Suzana da Costa Oliveira
Tásio Vasconcelos da Silveira
Thiago Magela Rodrigues Dias
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida
Vastí Mendes da Silva Rocha
Wagner Eliezer Roncoletta

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
- **Regina Mariani -** Coordenadora responsável pela parte jurídica
- **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

Ordenação tem a tarefa de organizar elementos em uma ordem particular, e isto é implementado em uma variedade de aplicações. Considerando uma aplicação bancária, a qual exibe a lista de contas de clientes ativos, por exemplo. A maioria dos usuários deste sistema provavelmente preferem ter a lista em uma ordem crescente, por conveniência.

Nesta lição veremos vários algoritmos de ordenação foram inventados porque essa tarefa é fundamental e freqüentemente utilizada. Por estas razões, examinar os algoritmos existentes será muito benéfico.

Ao final desta lição, o estudante será capaz de:

- Explicar os algoritmos utilizados em ordenação por inserção, ordenação por seleção, *Merge Sort* e *Quick Sort*
- Implementar seu próprio algoritmo utilizando essas técnicas

2. Ordenação por inserção

Um dos mais simples algoritmos desenvolvidos é o de ordenação por inserção. A idéia de algoritmo é absolutamente intuitiva. A seguinte sinopse descreve como a ordenação por inserção funciona para ordenar uma série de cartas. Desejamos ordenar uma série de cartas do menor até o maior da categoria. Todas as cartas estão inicialmente colocadas em uma tabela e a chamaremos de 1ª tabela, seguindo estritamente a ordem da esquerda para a direita, do topo ao fundo. Nós temos outra tabela, chamada de 2ª tabela, onde as cartas serão posicionadas. Escolha a primeira carta disponível da esquerda na 1ª tabela, e que esteja no topo, e coloque esta carta no local adequado (ou seja, ordenado) posicionando na 2ª tabela. Escolha a próxima carta disponível da 1ª tabela e compare-a com as cartas da 2ª tabela e coloque-a na posição adequada. O processo continua até que todas as cartas sejam colocadas na 2ª tabela.

O algoritmo de ordenação por inserção basicamente divide os elementos a serem ordenados em dois grupos: os não ordenados (semelhante à 1ª tabela) e os ordenados (semelhante à 2ª tabela). O primeiro elemento disponível é selecionado dentre os não ordenados do array e este é corretamente posicionado na parte ordenada do array. Esse passo é repetido até que não hajam mais elementos à esquerda, na parte não ordenada do array.

2.1. O algoritmo

```
public void insertionSort(Object array[], int startIdx, int endIdx) {
   for (int i = startIdx; i < endIdx; i++) {
      int k = i;
      for (int j = i + 1; j < endIdx; j++) {
         if (((Comparable) array[k]).compareTo(array[j])>0) {
            k = j;
         }
      }
      swap(array[i], array[k]);
   }
}
```

2.2. Um exemplo

Dados	***1º passo***	***2º passo***	***3º passo***	***4º passo***
Manga	Manga	Maçã	Laranja	Banana
Maçã	Maçã	Manga	Maçã	Laranja
Pêssego	Pêssego	Pêssego	Manga	Maçã
Laranja	Laranja	Laranja	Pêssego	Manga
Banana	Banana	Banana	Banana	Pêssego

Figura 1: Exemplo de ordenação por inserção

3. Ordenação por seleção

Ao se criar um primeiro algoritmo de ordenação, provavelmente foi criado algo parecido com o algoritmo de ordenação por inserção. Como o algoritmo de ordenação por inserção, esse algoritmo é muito intuitivo e fácil de implementar.

Novamente vamos observar como este algoritmo funciona em uma escala de maço de cartas. Considere que as cartas serão organizadas em ordem crescente. Inicialmente, as cartas estão organizadas linearmente na tabela da esquerda para a direita e do topo até embaixo. Confira o nível de cada carta e escolha a carta com o menor nível. Troque a posição dessa carta com a da primeira carta, a que está no topo da esquerda. Depois, localize a carta com o menor nível entre as cartas restantes, excluindo a primeira carta escolhida. Troque novamente a carta selecionada com a carta na segunda posição. Repita esse mesmo passo até que, da segunda à última posição na tabela sejam analisadas e possivelmente trocadas por cartas com o menor valor.

A idéia principal por trás do algoritmo de ordenação por seleção é selecionar o elemento com o menor valor e então trocar o elemento selecionado com o elemento na i posição. O valor de i inicia com 1 até *n*, onde *n* é o número total de elementos menos 1.

3.1. O algoritmo

```
public void selectionSort(Object array[], int startIdx, int endIdx) {
   int min;
   for (int i = startIdx; i < endIdx; i++) {
      min = i;
      for (int j = i + 1; j < endIdx; j++) {
         if (((Comparable) array[min]).compareTo(array[j])>0) {
            min = j;
         }
      }
      swap(array[min], array[i]);
   }
}
```

3.2. Um exemplo

Dados	***1º passo***	***2º passo***	***3º passo***	***4º passo***
Maricar	Hannah	Hannah	Hannah	Hannah
Vanessa	*Vanessa*	Margaux	Margaux	Margaux
Margaux	*Margaux*	*Vanessa*	Maricar	Maricar
Hannah	Maricar	*Maricar*	*Vanessa*	Rowena
Rowena	Rowena	Rowena	*Rowena*	Vanessa

Figura 2: Exemplo de ordenação por seleção

4. Merge Sort

Antes de examinar o algoritmo Merge Sort, vamos primeiramente dar uma rápida olhada no paradigma do dividir-e-conquistar para que melhor se compreenda o Merge Sort.

4.1. Paradigma do dividir-e-conquistar

Vários algoritmos utilizam a repetição (recursividade) para solucionar um determinado problema. O problema original é dividido em subproblemas, então as soluções dos subproblemas conduzem a solução do poblema principal. Esse tipo de algoritmo tipicamente seguem o paradigma do dividir-e-conquistar.

Em cada nível de recursividade, o paradigma consiste de três passos.

1.Dividir
   Dividir o problema principal em subproblemas.

2.Conquistar
   Conquistar os subproblemas resolvendo-os através da recursividade. No caso em que os subproblemas são simples e pequenos o suficiente, resolver de maneira direta.

3.Combinar
   Unir as soluções dos subproblemas, direcionando à solução do problema principal.

4.2. Entendendo Merge Sort

Como mencionado anteriormente, Merge Sort utiliza a técnica do dividir-e-conquistar. Dessa foma, a descrição deste algoritmo é seguida como exemplo depois dos três passos do paradigma do dividir-e-conquistar. Aqui está como o Merge Sort funciona.

1.Dividir
   Dividir a sequência dos elementos dados em duas partes.

2.Conquistar
   Conquistar cada parte de modo repetitivo, chamando o método Merge Sort.

3.Combinar
   Combinar ou fundir as duas partes recursivamente para apresentar a seqüência ordenada.

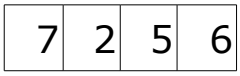
A repetição acaba quando o objetivo básico é alcançado. Este é o caso onde a parte a ser ordenada possui exatamente um elemento. Já que apenas um elemento seja separado para ser ordenado, essa parte já está organizada na sua própria sequência.

4.3. O algoritmo

```
void mergeSort(Object array[], int startIdx, int endIdx) {
   if (array.length != 1) {
      mergeSort(leftArr, startIdx, midIdx);
      mergeSort(rightArr, midIdx+1, endIdx);
      combine(leftArr, rightArr);
   }
}
```

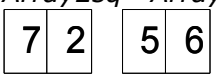
4.4. Um exemplo

Dados:



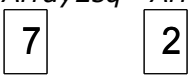
Dividir o array de dados em dois:

ArrayEsq *ArrayDir*

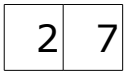


Dividir o ArrayEsq em dois:

ArrayEsq *ArrayDir*

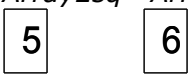


Combinar

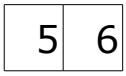


Dividir *ArrDir* em dois:

ArrayEsq *ArrayDir*



Combinar



Combinar

ArrayEsq e *ArrayDir*.

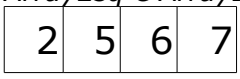


Figura 3: Exemplo de Merge sort

5. Quick Sort

Quick Sort foi criado por C.A.R. Hoare. Como o Merge Sort, este algoritmo é baseado no paradigma do dividir-e-conquistar. Mas ao invés de possuir as três fases, ele envolve apenas as seguintes fases:

1.Dividir
Separação dos arrays em dois subarrays A[p...q-1] e A[q+1...r] onde cada elemento em A[p...q-1] é menor ou igual a A[q] e cada elemento em A[q+1...r] é maior ou igual a A[q]. A[q] é chamado de eixo. Cálculo de q é parte do processo de separação.

2.Conquistar
Ordenar os subarrays pela recursividade, chamado de método *quickSort*.

Não existe a fase de "Combinar" pois os subarrays são ordenados localmente.

5.1. O algoritmo

```
void quickSort(Object array[], int leftIdx, int rightIdx) {
   int pivotIdx;
   if (rightIdx > leftIdx) {
      pivotIdx = partition(array, leftIdx, rightIdx);
      quickSort(array, leftIdx, pivotIdx-1);
      quickSort(array, pivotIdx+1, rightIdx);
   }
}
```

5.2. Um exemplo

Array dado:

3	1	4	1	5	9	2	6	5	3	5	8

Escolher o primeiro elemento para ser o eixo = 3.

3	1	4	1	5	9	2	6	5	3	5	8

Inicializar a esquerda com o ponteiro no segundo elemento, e a direita com o ponteiro no último elemento.

	Esq.										Dir.
3	1	4	1	5	9	2	6	5	3	5	8

Mover o ponteiro da esquerda na direção da direita até ser localizado um valor maior do que o do eixo. Mover o ponteiro direito na direção esquerda até ser localizado um valor menor do que o eixo.

		Esq.							Dir.		
3	1	4	1	5	9	2	6	5	3	5	8

Trocar os elementos referidos com os ponteiros esquerdo e direito.

		Esq.							Dir.		
3	1	3	1	5	9	2	6	5	4	5	8

Mover os ponteiros direito e esquerdo novamente.

				Esq.		Dir.					
3	1	3	1	5	9	2	6	5	4	5	8

Trocar os elementos.

				Esq.		Dir.					
3	1	3	1	2	9	5	6	5	4	5	8

Mover os ponteiros esquerdos e direitos novamente.

				Dir.	Esq.						
3	1	3	1	2	9	5	6	5	4	5	8

Observe que os ponteiros da esquerda e direita se cruzaram e o direito < esquerdo. Neste caso, trocar o eixo pelo valor do ponteiro direito.

				pivô							
2	1	3	1	**3**	9	5	6	5	4	5	8

Figura 4: Exemplo de Quick Sort

A caminhada do eixo agora está completa. Ordenação de arrays por recursividade em cada lado, utilizando o eixo.

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.