Módulo 3

Estruturas de Dados

Lição 8

Tabelas

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

Uma das operações mais comuns no processo de solução de problemas é a busca. Esta refere-se ao problema de encontrar dados que estão em algum lugar da memória do computador. Algumas informações identificadas como dados desejados são alimentados para mostrar resultados desejados. Tabelas são mais comuns em estruturas de armazenamento de dados para buscas.

Ao final desta lição, o estudante será capaz de:

- Discutir os conceitos básicos e as definições sobre **tabelas**: **chaves**, **operações** e **implementação**
- Explicar as organizações de tabelas – ordenadas e não ordenadas
- Executar buscas usando uma tabela sequencial, indexação sequencial, binária e busca por Fibonacci

2. Definições e Conceitos Correlatos

Uma tabela é definida como um grupo de elementos, cada um chamado de registro**.** Cada registro tem uma única chave associada com o seu registro distinto a ser utilizado.

Chave	Dado
K_0	X_0
K_1	X_1
…	…
K_i	X_i
…	…
K_{n-1}	X_{n-1}

Na tabela acima, n registros estão armazenados. K_i é a chave da posição i, enquanto X_i é associado ao dado. A notação usada para um registro é (K_i, X_i).

A classe definição utilizada para a tabela em Java é

```
class Table {
  int key[];
  int data[];
  int size;

  // Cria uma tabela vazia
  public Table() {
  }

  // Cria uma tabela de tamanho s
  public Table(int s) {
    size = s;
    key = new int[size];
    data = new int[size];
  }
}
```

2.1. Tipos de Chaves

Se uma chave é contida dentro de um registro e este é relativo ao início do registro específico, esta é conhecida como **interna** ou **chave embutida**. Se a chave esta contida em uma tabela separada como ponteiros associando-a aos dados, a chave é classificada como uma **chave externa.**

2.2. Operações

Do lado da busca, muitas outras operações podem ser feitas numa tabela. A seguir uma lista das operações possíveis:

- Busca por registro em que $K_i = K$, onde K é dado pelo usuário
- Inserção
- Deleção
- Busca do registro com chave menor (mais larga)
- Dada uma chave K_i, encontrar o registro com a próxima chave mais larga (menor)
- E outras…

2.3. Implementação

Uma tabela pode ser implementada usando alocação sequencial, alocação por *link* ou uma combinação de ambas. Na Implementação da árvore ADT, existem diversos fatores a considerar:

- Tamanho de espaço de chavo U_k, isto é, o número de chaves possíveis
- Natureza da tabela: dinâmica ou estática
- Tipo e misto de operações realizadas na tabela

Se o espaço de chave é fixo, por exemplo **m**, não tão grande, então a tabela pode simplesmente ser implementada como um array de **m** células. Com isto toda chave no conjunto é associada a um campo na tabela. Se a chave é a mesma que o índice do *array*, ela é conhecida como **tabela de endereço direto.**

Fatores de Implementação

Ao implementar uma tabela de endereçamento direto, as seguintes coisas devem ser consideradas:

- Desde que os índices identifiquem registros unicamente, não é necessário armazenar a chave k_i explicitamente.
- Os dados podem ser armazenados em qualquer lugar. Se não há espaço bastante para os dados X_i com a chave K_i, utiliza-se uma estrutura externa à tabela, um ponteiro para o dado atual é então armazenado como X_i. Neste caso, a tabela serve como um índice para o dado atual.
- É necessário para indicar células em desuso correspondentes a chaves em desuso.

Vantagens

Com as tabelas de endereços diretos, a busca é eliminada pois a célula X_i que contém o dado ou um ponteiro para o dado é apontado para a chave K_i. Da mesma forma, operações de inserção e deleção são relativamente diretas.

3. Tabelas e Busca

Um **algoritmo de busca** aceita um argumento e tenta encontrar um registro ao qual a chave é igual à especificada. Se a busca for executada com sucesso, um ponteiro é retornado. **Recuperação** ocorre quando a busca é realizada com sucesso. Esta seção discute as maneiras de organizar uma tabela, bem como as operações de busca nas diferentes organizações de tabelas.

3.1. Organização de Tabela

Há dois modos genéricos para organizar uma tabela: ordenado e desordenado. Em uma tabela ordenada, os elementos são sorteados baseados em suas chaves. A referência ao primeiro elemento, ao segundo elemento, e assim sucessivamente torna-se possível. Em uma tabela desordenada, não existem relações presumidas entre os registros e suas chaves associadas.

3.2. Busca Sequencial em uma Tabela Desordenada

Buscas sequenciais lêem cada registro seguidamente do início até que o registro ou registros procurados sejam encontrados. Isto é aplicável a uma tabela que é organizada também como um array ou como uma lista *linkada*. Esta busca é também conhecida como **busca linear**.

	CHAVE	DADO
1	K_0	X_0
2	K_1	X_1
…	…	…
i	K_i	X_i
…	…	…
n	K_n	X_n

O algoritmo

Dado: Uma tabela de registros R_0, R_1, ..., R_{n-1} com chaves K_0, K_1, ... K_{n-1} respectivamente, onde $n \geq 0$. Procurar por um valor K:

1. Inicialize: faça i = 0
2. Compare: se K = K_i, pare – busca com sucesso
3. Avance: Incremente i por 1
4. Fim do arquivo?: se i < n, vá para o passo 2. então pare: Busca sem sucesso

Eis uma implementação de busca sequencial:

```
class Search {
  final static int notFound = -1;
  public int sequentialSearch(int k, int key[]) {
    for (int i=0; i<key.length; i++)
      if (k == key[i])
        return i; // busca com sucesso
    return -1; // busca sem sucesso
  }
}
```

A busca sequencial realiza **n** comparações no pior caso, com uma complexidade de tempo O(n). Este algoritmo trabalha bem quando a tabela é relativamente pequena ou é mal percorrida. A vantagem sobre este algoritmo é que ele trabalha uniformemente se a tabela está desordenada.

3.3. Buscando em uma Tabela Ordenada

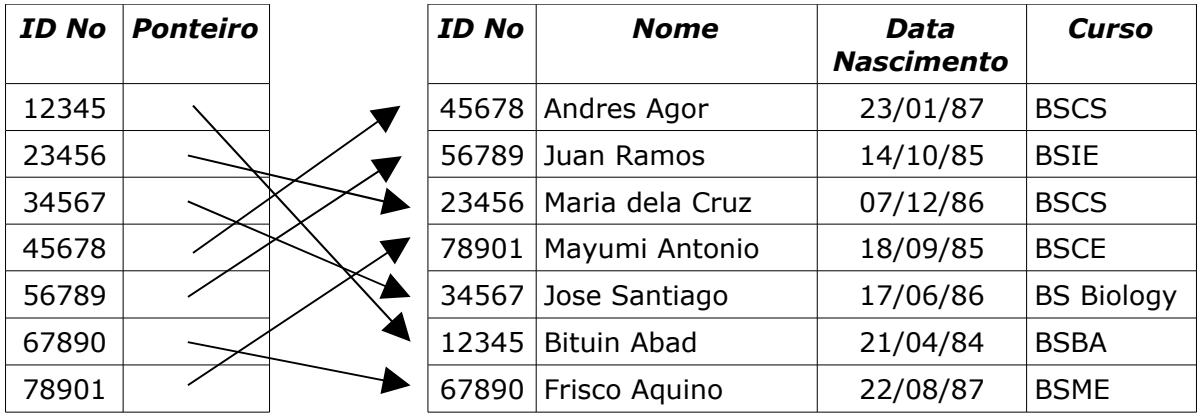
Existem três métodos de busca em uma tabela ordenada: busca sequencial indexada, busca binária e busca de Fibonacci.

Busca Sequencial Indexada

Na busca sequencial indexada, uma tabela auxiliar, chamada **índice**, aponta para a tabela ordenada. As seguintes são características do algoritmo de busca sequencial indexada:

- Cada elemento no índice consiste de uma chave e um ponteiro para o registro no arquivo que corresponde a K_{index}
- Elementos no índice devem ser ordenados baseados na chave
- O arquivo de dados atual deve ou não ser ordenado

A figura seguinte mostra um exemplo:



	ID No	*Ponteiro*
1	12345	
2	23456	
3	34567	
4	45678	
5	56789	
6	67890	
7	78901	

ID No	*Nome*	*Data Nascimento*	*Curso*
45678	Andres Agor	23/01/87	BSCS
56789	Juan Ramos	14/10/85	BSIE
23456	Maria dela Cruz	07/12/86	BSCS
78901	Mayumi Antonio	18/09/85	BSCE
34567	Jose Santiago	17/06/86	BS Biology
12345	Bituin Abad	21/04/84	BSBA
67890	Frisco Aquino	22/08/87	BSME

Com este algoritmo, o tempo de busca para um item particular é reduzido. Da mesma forma, um índice poderá ser usado para apontar para uma tabela ordenada implementada como um array ou com uma lista *linkada*. A última implementação implica em grande sobrecarga de espaço para ponteiros mas inserções e deleções podem ser realizadas imediatamente.

Busca Binária

Busca binária começa com um intervalo ocupando a tabela inteira, este é o valor médio. Se o valor procurado é menor que o item no meio do intervalo, o intervalo é encurtado para menos que a metade. Senão, é encurtado para mais que a metade. Este processo de redução do tamanho da busca pela metade é repetidamente realizado até que o valor seja encontrado ou o intervalo fique vazio. O algoritmo para a busca binária faz uso das seguintes relações na busca pela chave K:

- $K = K_i$: pare, o registro desejado foi encontrado
- $K < K_i$: procura a menos metade – registros com as chaves K_1 to K_{i-1}
- $K > K_i$: procura a maior metade – registros com as chaves K_{i+1} to K_n

Onde I é inicialmente o valor médio do índice.

O Algoritmo

```
// Retorna o índice da chave k se encontrado, senão -1
public int binarySearch(int k, Table t) {
  int lower = 0;
```

```
  int upper = t.size - 1;
  int middle;
  while (lower < upper) {
    // assume o médio
    middle = (int) Math.floor((lower + upper) / 2);
    if (k == t.key[middle])
      return middle;            // busca com sucesso
    else if (k > t.key[middle])
      lower = middle + 1;       // menor metade descartada
    else
      upper = upper - 1;        // maior metade descartada
  }
  return notFound;              // busca sem sucesso
}
```

Por exemplo, busca pela chave k = 34567

0	1	2	3	4	5	6
12345	23456	34567	45678	56789	67890	78901

menor = 0, maior = 6, médio = 3: $k < k_{middle}$ (45678)

menor = 0, maior = 2, médio = 1: $k > k_{middle}$ (23456)

menor = 2, maior = 2, médio = 2: $k = k_{middle}$ (34567) ==> busca com sucesso

Então a área de procura é reduzida *logaritmicamente*, isto é, cada vez que o tamanho é reduzido, a complexidade de tempo do algoritmo é $O(\log_2 n)$. O algoritmo pode se usado se a tabela usa organização sequencial indexada. Entretanto, pode apenas ser usado com tabelas ordenadas armazenadas como um *array*.

Busca Binária Multiplicativa

É similar ao algoritmo anterior, mas evita a divisão necessária para se encontrar a chave média. Para fazer isto, é necessário reorganizar os registros na tabela:

1. Atribua chaves $K_1 < K_2 < K_3 < \ldots < K_n$ aos nós de uma árvore binária completa na sequência in *order*

2. Organize os registros na tabela de acordo com sequência *level-order* correspondente na árvore binária

Algoritmo de Busca

A comparação começa na raiz da árvore binária , j = 0, que é a chave média na tabela original.

```
/*
 * Recebe um conjunto de chaves representado como uma árvore binária completa
 */
public int multiplicativeBinarySearch(int k, int key[]) {
  int i = 0;
  while (i < key.length) {
    if (k == key[i])
      return i;          // busca bem sucedida
    else if (k < key[i])
```

```
        i = 2 * i + 1;      // vá para esquerda
      else
        i = 2 * i + 2;      // vá para direita
    }
    return -1;              // busca mal sucedida
  }
```

Como a computação do elemento médio é eliminada, a busca binária multiplicativa é mais rápida que a busca binária tradicional. Entretanto, há necessidade de uma reorganização de um dado conjunto de chaves antes que o algoritmo possa ser aplicado.

Busca de *Fibonacci*

Busca de *Fibonacci* utiliza as propriedades simples da sequência numérica de *Fibonacci* definida pela seguinte relação de recorrência:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_j = F_{i-2} + F_{i-1} \quad , i \geq 2$$

Ou seja, a sequência 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Em Java,

```
public int fibonacci(int i) {
  if (i == 0)
    return 0;
  else if (i == 1)
    return 1;
  else
    return fibonacci(i-1) + fibonacci(i-2);
}
```

No algoritmo de busca, duas variáveis auxiliares, **p** e **q**, são usadas:

$p = F_{i-1}$
$q = F_{i-2}$

K_j é escolhida inicialmente de tal forma que $j = F_i$, onde F_i é o maior número da série de *Fibonacci* que é menor ou igual ao tamanho da tabela (n).

É uma suposição que a tabela é de seja de tamanho $n=F_{i+1}-1$.

Três estados de comparação são possíveis:

- Se $K = K_j$, pare: busca bem sucedida
- Se $K < K_j$
  - Descarte todas as chaves com índices maiores que j
  - Faça $j = j - q$
  - Desloque p e q uma posição para a esquerda na sequência numérica
- Se $K > K_j$,
  - Descarte todas as chaves com índices menores que j
  - Faça $j = j + q$

- Desloque p e q duas posições para a esquerda na sequência numérica

- $K < K_j$ e q=0 ou $K > K_j$ e p=1: busca mal sucedida

Esse algorítimo encontra o elemento do índice 1 ao **n** e, como a indexação no Java começa em 0, há uma necessidade de lidar com o caso onde k = key[0].

Por exemplo, procure pela chave k = 34567:

0	1	2	3	4	5	6
12345	23456	34567	45678	56789	67890	78901

0 1 1 2 3 5 8 13 F
0 1 2 3 4 5 6 7 i
i = 5; F_i = 5; (Suposição) table size = F_{i+1} - 1 = 7

j = 5, p = 3, q = 2: k < key[j]
j = 3, p = 2, q = 1: k < key[j]
j = 2, p = 1, q = 1: k = key[j] Bem sucedido

Outro exemplo, procure pela chave = 15:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

i = 7; F_i = 13; (Suposição) table size = F_{i+1} - 1 = 20

j = 13; p = 8, q=5: k > key[j]
j = 18; p = 3; q=2: k < key[j]
j = 15; p = 2; q=1: k = key[j] Bem sucedido

Este é o algorítimo:

```
public int fibonacciSearch(int k, int key[]) {
  int p = 0, q = 0, j = 0;
  int f = 0, i = 0;
  int temp;
  while (true) {
    f = fibonacci(i);
    if (key.length < f) {
      j = f;
      p = fibonacci(i-1);
      q = fibonacci(i-2);
      break;
    }
    i++;
  }
  if (k == key[0])
    return 0;
  while (true) {
    if (j >= key.length)
      j = key.length-1;
    if (k == key[j])
      return j;
```

```
      else if (k < key[j]) {
        if (q == 0)
          break;
        else {
          j = j - q;
          temp = p;
          p = q;
          q = temp - q;
        }
      } else {
        if (p == 1)
          break;
        else {
          j = j + q;
          p = p - q;
          q = q - p;
        }
      }
    }
    return notFound;
  }
```

Podemos testar estas pesquisas através do seguinte método principal inserido na classe:

```
  public static void main(String args[]) {
    int size = 2000;
    int key[] = new int[size];
    for (int i = 0; i < size; i++)
      key[i] = i;
    Search s = new Search();
    SimpleDateFormat sdT = new SimpleDateFormat("hh:mm:ss:SSSS");
    SimpleDateFormat sdR = new SimpleDateFormat("ssSSS");

    int inicial = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    for (int i = 0; i < size; i++)
      s.sequentialSearch(i, key);
    int fim = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    System.out.println("Result Sequential: " + (fim - inicial));

    inicial = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    for (int i = 0; i < size; i++)
      s.binarySearch(i, key);
    fim = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    System.out.println("Result Binary: " + (fim - inicial));

    inicial = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    for (int i = 0; i < size; i++)
      s.multiplicativeBinarySearch(i, key);
    fim = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    System.out.println("Result Multiplicative Binary: " + (fim - inicial));

    inicial = Integer.parseInt(sdR.format(new Date()));
```

```
    System.out.println("Time: " + sdT.format(new Date()));
    for (int i = 0; i < size; i++)
      s.fibonacciSearch(i, key);
    fim = Integer.parseInt(sdR.format(new Date()));
    System.out.println("Time: " + sdT.format(new Date()));
    System.out.println("Result Fibonacci: " + (fim – inicial));
  }
```

Este método criará um array de 2000 posicões, contendo o valor em cada posição. O que faremos com ele será analizar os tempos iniciais e finais com cada método. O resultado final pode variar pois depende do computador utilizado, eis um exemplo de saída:

```
Time: 05:19:43:0593
Time: 05:19:43:0609
Result Sequential: 16
Time: 05:19:43:0609
Time: 05:19:43:0734
Result Binary: 125
Time: 05:19:43:0734
Time: 05:19:43:0750
Result Multiplicative Binary: 16
Time: 05:19:43:0750
Time: 05:19:44:0187
Result Fibonacci: 437
```

Realize alguns testes modificando os valores e descubra qual o método de pesquisa ideal para as suas necessidades.

4. Exercícios

Utilizando os métodos de pequisa abaixo,

a) Pesquisa Sequencial
b) Pesquisa Binária Multiplicativa
c) Pesquisa Binária
d) Pesquisa Fibonacci

Pesquisar por

1. A em

S	E	A	R	C	H	I	N	G

2. D em

W	R	T	A	D	E	Y	S	B

3. T em

C	O	M	P	U	T	E	R	S

4.1. Exercício para Programar

1. Estenda a classe Java definida neste capítulo para que a mesma contenha métodos para a inserção em um local específico e exclusão de um item com chave K. Utilize a representação sequencial.

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.