



Estrutura de Dados

José Marcio Benite Ramos

Liluyoud Cury de Lacerda

Sara Luize Oliveira Duarte



INSTITUTO FEDERAL
RONDÔNIA

Cuiabá-MT
2014

Presidência da República Federativa do Brasil
Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Diretoria de Integração das Redes de Educação Profissional e Tecnológica

© Este caderno foi elaborado pelo Instituto Federal de Educação, Ciência e Tecnologia de Rondônia-RO, para a Rede e-Tec Brasil, do Ministério da Educação em parceria com a Universidade Federal do Mato Grosso.

Equipe de Revisão

Universidade Federal de Mato Grosso – UFMT

Coordenação Institucional
Carlos Rinaldi

Coordenação de Produção de Material Didático Impresso
Pedro Roberto Piloni

Designer Master
Neure Rejane Alves da Silva

Ilustração
Tatiane Hirata

Diagramação
Tatiane Hirata

Revisão de Língua Portuguesa
Livia de Sousa Lima Pulchério Monteiro

Revisão Final
Claudinet Antonio Coltri Junior

Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

Campus Porto Velho Zona Norte

Direção-Geral
Miguel Fabrício Zamberlan

Direção de Administração e Planejamento
Gilberto Laske

Departamento de Produção de EaD
Ariádne Joseane Felix Quintela

Coordenação de Design Visual e Ambientes de Aprendizagem
Rafael Nink de Carvalho

Coordenação da Rede e-Tec
Ruth Aparecida Viana da Silva

Projeto Gráfico

Rede e-Tec Brasil / UFMT

Estrutura de Dados - Informática para Internet

R1756e Ramos, José Marcio Benite.

Estrutura de dados/ José Marcio Benite Ramos; Liluyoud Cury de Lacerda; Sara Luize Oliveira Duarte; org. Instituto Federal de Educação, Ciência e Tecnologia; Universidade Federal do Mato Grosso - Cuiabá : UFMT; Porto Velho: IFRO, 2013.

123 p. ; -- cm.
Curso Informática para internet.

ISBN 978-85-68172-02-5

1. Estrutura de dados (Computação). 2. Estruturas lineares. 3. Algoritmos. 4. Árvores. I. Lacerda, Liluyoud Cury de. II. Duarte, Sara Luize Oliveira. III. Instituto Federal de Educação, Ciência e Tecnologia. IV. Universidade Federal do Mato Grosso. V. Título.

CDD 005.1
CDU 004.6

Apresentação Rede e-Tec Brasil

Prezado(a) estudante,

Bem-vindo(a) à Rede e-Tec Brasil!

Você faz parte de uma rede nacional de ensino que, por sua vez, constitui uma das ações do Pronatec - Programa Nacional de Acesso ao Ensino Técnico e Emprego. O Pronatec, instituído pela Lei nº 12.513/2011, tem como objetivo principal expandir, interiorizar e democratizar a oferta de cursos de Educação Profissional e Tecnológica (EPT) para a população brasileira propiciando caminho de acesso mais rápido ao emprego.

É neste âmbito que as ações da Rede e-Tec Brasil promovem a parceria entre a Secretaria de Educação Profissional e Tecnológica (Setec) e as instâncias promotoras de ensino técnico, como os institutos federais, as secretarias de educação dos estados, as universidades, as escolas e colégios tecnológicos e o Sistema S.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade e ao promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

A Rede e-Tec Brasil leva diversos cursos técnicos a todas as regiões do país, incentivando os estudantes a concluir o ensino médio e a realizar uma formação e atualização contínuas. Os cursos são ofertados pelas instituições de educação profissional e o atendimento ao estudante é realizado tanto nas sedes das instituições quanto em suas unidades remotas, os polos.

Os parceiros da Rede e-Tec Brasil acreditam em uma educação profissional qualificada – integradora do ensino médio e da educação técnica – capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Abril de 2014

Nosso contato
etecbrasil@mec.gov.br



Indicação de Ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.



Atenção: indica pontos de maior relevância no texto.



Saiba mais: oferece novas informações que enriquecem o assunto ou “curiosidades” e notícias recentes relacionadas ao tema estudado.



Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.



Mídias integradas: remete o tema para outras fontes: livros, filmes, músicas, *sites*, programas de TV.



Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.



Reflita: momento de uma pausa na leitura para refletir/escrever sobre pontos importantes e/ou questionamentos.



Palavra dos Professores-autores

Caro(a) estudante,

Seja bem-vindo(a) à disciplina de Estrutura de Dados.

Iniciaremos nossa jornada rumo ao conhecimento das estruturas de dados, um modo particular de manipulação das informações, utilizando a memória do computador.

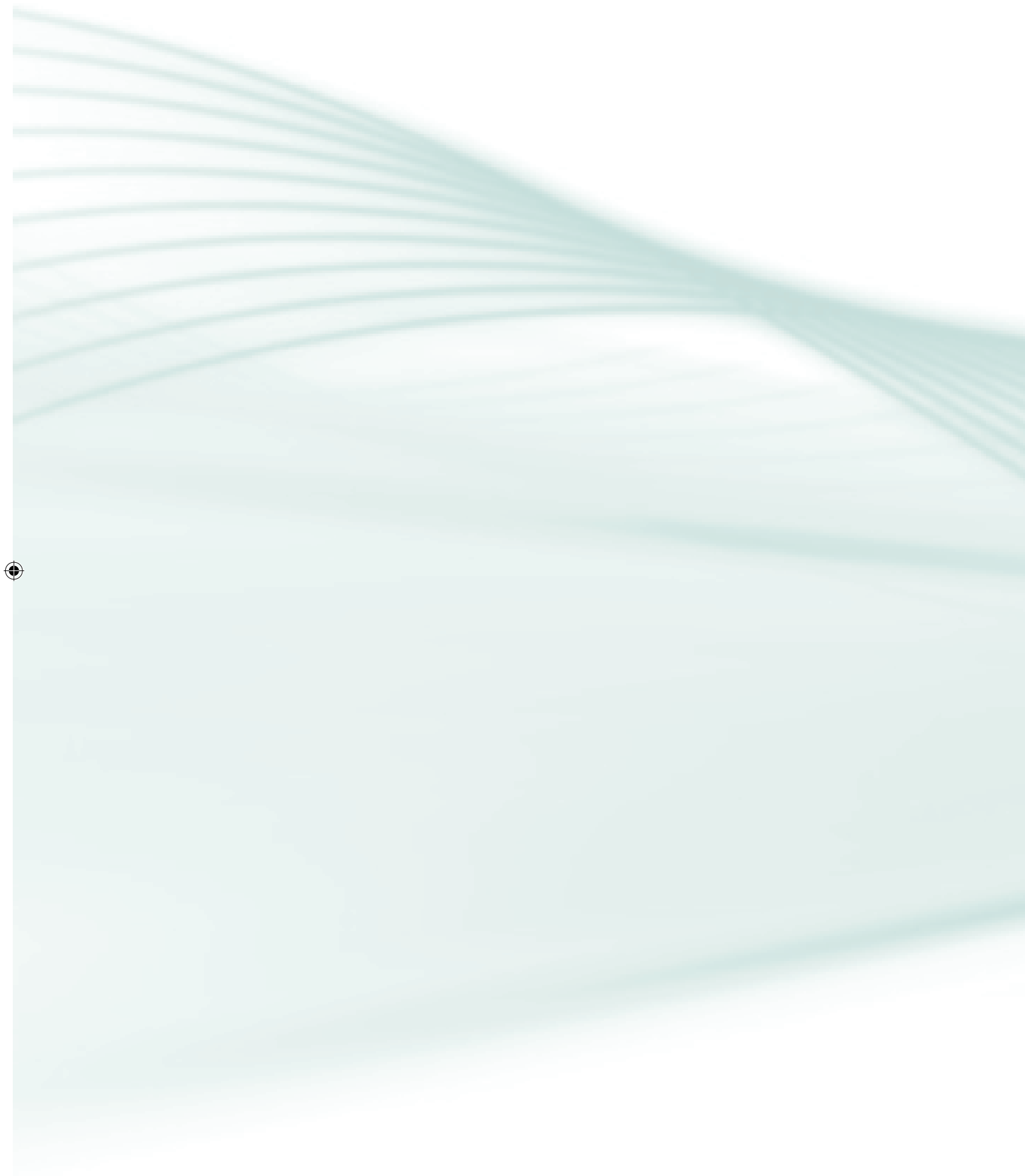
Parabéns pela sua iniciativa de realizar este curso. Somos seres humanos e, como tais, temos um temor inicial ao adentrar em um mundo desconhecido, mas somos movidos pela curiosidade, pela vontade de aprender mais e essa força nos permite enfrentar esse temor e conquistar novos mundos.

O esforço empreendido durante esta jornada será recompensado com o conhecimento, o maior bem que podemos ter. O conhecimento adquirido nunca será tirado de você, ele pode ser compartilhado, mas nunca subtraído e é através dele que podemos abrir novas portas, aproveitar as oportunidades que o mercado irá nos oferecer.

Vale ressaltar que este material é apenas uma parte de um conjunto de acontecimentos que compõem a trajetória dos estudos que você deverá realizar. Deverá ser complementado através da utilização do AMBIENTE VIRTUAL, do apoio do PROFESSOR TUTOR, do aprofundamento dos conteúdos proporcionados pelos LIVROS indicados etc. Este material é o início para o aprendizado do que a disciplina deve oferecer. Não deixe de acompanhar o recurso “Mídias Integradas” que fará indicações de ampliações possíveis, importantes para o aprofundamento do aprendizado.

Muito importante também é a realização das ATIVIDADES propostas. Nunca deixe de executá-las, pois é o momento indispensável de trabalho no qual você irá imergir na disciplina lidando com suas propostas.

Bom estudo!
Professores



Apresentação da Disciplina

A disciplina de Estrutura de Dados compreende as principais estruturas, técnicas de representação e manipulação de dados e como aplicá-las na resolução de problemas.

Veremos que todas as informações manipuladas em um programa devem ser previamente armazenadas na memória dos computadores e a forma em que elas são estruturadas poderá facilitar ou dificultar seu tratamento.

O entendimento e implementação das estruturas de dados, como vetores, matrizes, registros, ponteiros, listas e árvores e as operações de manipulação de dados são essenciais à formação do profissional da área de tecnologias da informação.

Ao final desta disciplina, você deverá possuir habilidade para conhecer e compreender os tipos de dados primitivos da linguagem e para implementar as mais variadas estruturas de dados, de modo a manipular as informações e utilizar a estrutura que melhor se adapta aos problemas analisados. Com isso, será capaz de implementar e analisar estruturas como vetores, listas, filas, pilhas e árvores, assim como utilizar métodos de classificação e pesquisas, os quais melhorarão o desempenho da manipulação dos dados das aplicações desenvolvidas.

É essencial que você relembre os conceitos utilizados em disciplinas anteriores para um melhor aproveitamento desta disciplina.

As aulas estão estruturadas de forma a conduzi-lo(a) às técnicas de manipulação de dados e dar a visão para utilização dos recursos mais adequados para implementação dos projetos.



Sumário

Aula 1. Conceitos básicos de dados	15
1.1 Introdução à Estrutura de Dados	16
1.2 Tipo de dados	18
1.3 Tipo de dado primitivo	19
1.4 Tipos abstratos de dado	20
Aula 2. Vetores e Matrizes	25
2.1 Vetores	26
2.2 Matrizes	30
Aula 3. Estruturas Lineares Estáticas	33
3.1 Introdução	34
3.2 Lista simples	36
3.3 Lista ligada estática	39
Aula 4. Estruturas Lineares Dinâmicas	43
4.1 Lista ligada dinâmica	44
4.2 Lista duplamente ligada	53
4.3 Lista circular	53
Aula 5. Filas e Pilhas	57
5.1 Filas	58
5.2 Pilhas	62
Aula 6. Classificação: Algoritmos de Pesquisa	69
6.1 Pesquisa	70
6.2 Pesquisa sequencial	70
6.3 Pesquisa binária	72
Aula 7 . Classificação: Algoritmos de Ordenação	77
7.1 Classificação	78
7.2 Bolha	79



7.3 Seleção direta.....	81
7.4 Inserção direta.....	82
Aula 8.Árvores.....	87
8.1 Tipos de representações de árvores.....	90
8.2 Árvores binárias.....	92
8.3 Árvores binárias de busca (pesquisa).....	95
8.4 Árvores balanceadas.....	99
8.5 Árvores B.....	102
Palavras Finais.....	105
Guia de Soluções.....	106
Referências.....	121
Currículo dos Professores-autores.....	122



Aula 1. Conceitos básicos de dados

Objetivos:

- reconhecer o que são e qual a importância das estruturas de dados;
- identificar os tipos de dados;
- distinguir o que são “variáveis”; e
- interpretar os tipos abstratos de dados.

A natureza parece encontrar forma práticas de disposição de suas estruturas as quais, muitas vezes, são “copiadas” pelo homem. Um exemplo prático é uma árvore. Nela temos o tronco, o qual sustenta os galhos que, por sua vez, sustentam as folhas. O caminho percorrido pela seiva para atingir cada uma delas é relativamente pequeno, se pensarmos na distância que teria se os galhos fossem dispostos em uma fileira. Essa estrutura de ramos permite que cada folha seja atingida com o menor esforço, uma característica muito importante para a sobrevivência.

Podemos também observar no exemplo anterior certa padronização. As folhas e os frutos de uma árvore são praticamente idênticos e, assim, cada estrutura tem sua uniformização. Essas características podem ser utilizadas em estruturas para armazenamento de informações, como, por exemplo, uma árvore genealógica.



Figura 1: Árvore com folhas e frutos.

Fonte: sxc.hu



A-Z

Programa de Computador pode ser considerado como um conjunto de instruções (comandos) que descrevem uma tarefa a ser realizada pelo computador.



Acesse o site para mais informações sobre sistema operacional: <http://www2.ic.uff.br/~aconci/SistemasOperacionais.html>

Acesse o site para mais informações sobre a linguagem JAVA: http://www.java.com/pt_BR/download/faq/whatis_java.

Estamos dizendo isso, pois, nesta aula você verá a importância de organizar adequadamente as informações que serão manipuladas e que os computadores devem ser preparados para receberem as informações que serão tratadas pelos programas. Os **programas de computadores** que serão desenvolvidos devem estabelecer um contrato com o sistema operacional do computador. As informações que serão tratadas devem respeitar os tipos definidos nesse contrato, isto é, quando você definir que irá trabalhar com números em um determinado momento, apenas valores numéricos (existem vários tipos numéricos) deverão ser informados. Para tanto, você será apresentado aos tipos mais comuns de dados suportados pela linguagem de programação (será utilizado o **JAVA**) como caracteres, números, datas, valores booleanos e estruturas que podem conter mais de um tipo e como as informações são representadas em um programa e armazenadas na memória do computador através de variáveis.

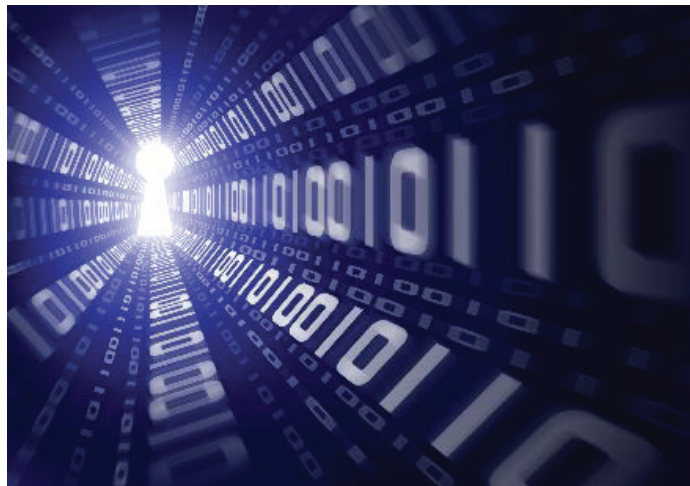


Figura 2: Bits formando uma luz no fim do túnel.

Fonte: < <http://scottkantner.com/bits-bytes-and-bandwidth/> > Acesso em: 07 març.2013.

E, então, está preparado? Podemos começar? Vamos em frente.

1.1 Introdução à Estrutura de Dados

Vamos iniciar a nossa aula com uma suposição.

Considere que você foi designado para organizar os registros dos empregados de uma empresa. Sua primeira missão é criar uma estrutura que contenha todos os nomes dos empregados da empresa. Para começar, você faz uma lista dos empregados com suas funções, conforme quadro abaixo:

Nome	Função
João	Gerente
Pedro	Supervisor
Ana	Supervisor
Maria	Programador
Carla	Programador
Carlos	Programador
Marcos	Programador

Figura 03: Lista de funcionários

Fonte: Autor

A estrutura acima nos dá apenas uma visão parcial da empresa. Não sabemos, por exemplo, os relacionamentos entre empregados, isto é, quem são seus superiores. Não temos uma visão de quais gerentes são responsáveis por quais empregados e assim por diante.

Após pensar sobre o problema por um tempo, você decide que um diagrama é a estrutura melhor para representar os relacionamentos dos empregados na empresa.

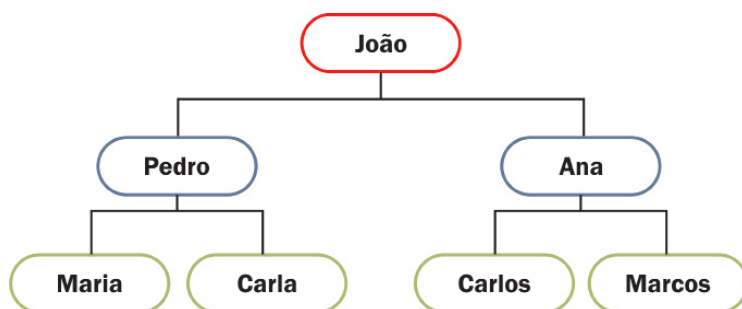


Figura 04: Estrutura hierárquica de funcionários.

Fonte: Autor

Os diagramas acima são exemplos de estruturas de dados diferentes, já que, na primeira representação temos os dados organizados em uma lista.

No caso da lista, os nomes dos empregados podem ser armazenado na ordem alfabética de modo que possamos encontrar o registro de um deles muito rapidamente. Entretanto, para exibir a relação entre os empregados, esta estrutura não é muito útil. A segunda estrutura (representando uma árvore) é mais adequada para esta finalidade.

Como você pode perceber, as estruturas com as quais representaremos os dados poderão promover uma organização mais adequada e, conseqüentemente, facilitar suas consultas. Precisamos estar atentos, pois existem várias maneiras de organizar os dados em estruturas e cada uma será mais indi-





cada ao tipo de informação trabalhada. Por isso, para podermos conhecer quais as estruturas disponíveis, precisaremos conhecer primeiro como uma informação simples deve ser manipulada.

Vamos conhecer os tipos de dados?

1.2 Tipo de dados

Os tipos de dados definem uma combinação de valores que uma **variável** pode receber e o conjunto de operações que esta pode executar.

A-Z

Variável é uma posição na memória capaz de armazenar um valor. Vimos seus conceitos em técnicas de programação

Os valores expressos podem ser de forma genérica, numéricos ("1", "25", "5,3"), caracteres ("a", "abc", "João"), datas ("10/10/2012", "20 de maio de 2010") etc.

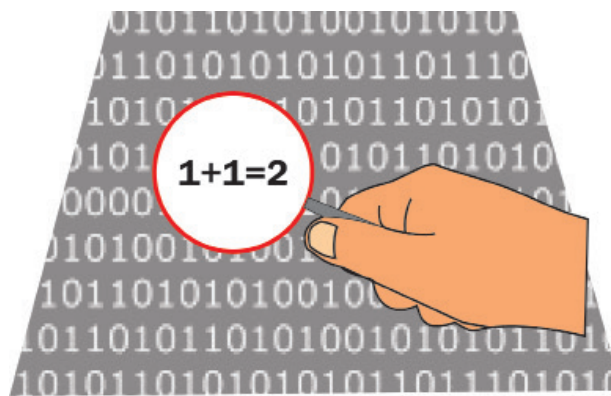


Figura 5: Tipo de dado.

Fonte: <<http://www.efetividade.blog.br/2011/05/26/voce-sabe-como-funciona-as-memorias-do-seu-pc>>.
Acesso em: 07 març.2013

Precisamos lembrar que as operações realizadas entre números do tipo inteiro são diferentes dos números reais (com casas decimais), os quais podem ser divididos. O mesmo vale para tipos como caracteres que não possuem operações matemáticas, mas podem, por exemplo, serem concatenados ("ca" + "sa" = "casa"). Como as variáveis dependem do sistema operacional e da linguagem utilizada, elas podem possuir muitas variantes.

Até aqui está tudo bem? Conseguiu compreender o conceito? Vamos prosseguir?

Então, vamos lá.

É importante salientar que definir o tipo de dado da informação que será



manipulada é muito importante para o SO (sistema operacional) determinar o espaço que será reservado, pois é através dele que o SO solicitará uma porção da memória para armazenar a informação tratada.

Outra questão é que obter a informação da memória também depende do tipo de dado, pois são eles que possibilitam ao **compilador** realizar as conversões dos dados em memória para obter os valores manipulados nos programas. Lembre-se, como foi apresentado em Fundamento da Informática, que os dados em memória são representados apenas por “0” e “1” e é através do tipo de dado que esse conjunto de números binários são definidos como uma letra ou um número.

Imagine que eu lhe peça para buscar um recipiente para armazenar uma “coisa”. Sem saber o que é, você poderia pegar uma garrafa de 2 litros ou um pequeno saco de papel. Se eu pretendia lhe entregar um litro de água, a garrafa atenderia, mas o saco de papel não iria servir. Agora imagine que a quantidade poderá ser imensa e, desta forma, ficaria impossível idealizar o recipiente correto. Isso vale para o SO, quando um sistema vai manipular informações. Elas devem ser tipificadas para que não ocorra uma reserva de espaço em memória insuficiente ou extremamente grande.

Agora que já compreendemos a importância de definirmos o tipo de dados, vamos falar sobre as variáveis.

A seguir, vamos ver os tipos de dados primitivos.

1.3 Tipo de dado primitivo

Vimos que os **tipos de dados** são uma combinação de valores que uma variável pode armazenar, o que pode variar conforme o sistema operacional e a linguagem de programação. Assim, uma variável nomeada “nome” utilizada para armazenar nomes de pessoas deverá estar associada a um tipo de dado compatível com os valores armazenados.

De acordo com a linguagem de programação utilizada, o tipo de um dado é verificado de maneira diferente, conforme a análise do compilador. O Java é uma linguagem compilada e possui seu conjunto de tipos de dado básico. Esses tipos são conhecidos como tipos primitivos, pois são suportados diretamente pelo compilador e são utilizados durante a codificação na definição de variáveis.

A-Z

Um compilador é um programa que converte um código descrito em uma linguagem de alto nível para um programa em linguagem simbólica “compreensível” pelo processador.





Veja o quadro abaixo, onde temos o tipo de dados, sua descrição e seu tamanho.

A-Z

O UNICODE é uma notação que utiliza uma tabela contendo representação numérica dos caracteres.

Ponto flutuante é um formato de representação dos números reais, onde temos uma parte inteira, a vírgula e a parte fracionária.

Tipo	Descrição	Tamanho
boolean	Pode assumir o valor true (verdadeiro) ou o valor false (falso)	1 bit
Char	Caractere em notação Unicode . Armazena dados alfanuméricos.	16 bits
Byte	Inteiro. Pode assumir valores entre -128 e 127.	8 bits
Short	Inteiro. Valores possíveis: -32.768 a 32.767	16 bits
Int	Inteiro. Valores entre -2.147.483.648 e 2.147.483.647.	32 bits
Long	Inteiro. Valores entre -9.223.372.036.854.770.000 e 9.223.372.036.854.770.000.	64 bits
Float	Real (representa números em notação de ponto flutuante). Valores entre -1,4024E-37 e 3.40282347E + 38	32 bits
Double	Real (representa números em notação de ponto flutuante). Valores entre -4,94E-307 e 1.79769313486231570E + 308	64 bits

A tabela acima representa alguns dos principais tipos de dados utilizados para manipulação de informações em programas desenvolvidos na linguagem JAVA. A coluna Tamanho especifica o espaço de memória utilizado por cada tipo. Sendo assim, é muito importante especificar adequadamente o tipo empregado para não reservarmos espaços de memória que não serão utilizados e, também, para não subdimensionarmos o espaço a ser utilizado.

1.4 Tipos abstratos de dado

Como vimos anteriormente, um tipo de dado é associado à forma de interpretação do conteúdo da memória do computador. Quando vamos trabalhar com valores inteiros temos um tipo de dados "int".

Quando analisamos as informações sobre a perspectiva da forma com que as manipulamos como: somar, consultar etc. tem-se um conceito de Tipo de Dado chamado Tipo Abstrato de Dado (TAD). As estruturas de dados são uma maneira particular de se representar um TAD.

Exemplos de TAD:



- listas;
- pilhas e filas; e
- árvores.

Um TAD é uma especificação de uma coleção de dados e um grupo de operações que podem ser executadas sobre esses dados. Assim, um TAD descreve quais dados podem ser armazenados em uma única estrutura e o que é possível fazer com esses dados através das suas operações.



Os dados em um TAD devem ser protegidos de intervenções externas e apenas as operações especificadas no TAD podem manipular seus dados. Assim, o usuário de um TAD só tem acesso às operações disponibilizadas, as quais podem, por exemplo, consultar ou alterar os dados.

Vamos ver um exemplo:

Considere que você deseja representar um elevador com as seguintes características: capacidade de pessoas, total de pessoas no elevador e andar em que se encontra. Sem os TADs nós teríamos três variáveis distintas, mas com ele podemos ter a seguinte estrutura:

Elevador
int capacidade int totalPessoas int andar
Subir() Descer() EntrarPessoa() SairPessoa()

Figura 06: TAD de um elevador.

Fonte: Autor.

Somente as funções definidas poderiam alterar os valores dos atributos e, assim, se desejássemos alterar os valores de totalPessoas, usaríamos as funções EntrarPessoa() para adicionar e SairPessoa() para retirar alguém.

Resumo

As estruturas de dados são essenciais para a otimização da manipulação das informações tanto para o armazenamento quanto para a pesquisa/recupera-



ção. Utilizar uma correta estrutura facilitará o desenvolvimento dos sistemas e os tornará mais eficientes. Existem várias estruturas e cada uma é mais adequada para um determinado conjunto de informações. Manipular informações depende também das especificações correta dos tipos de dados, as quais serão representadas através das variáveis. O tipo de dado informa ao compilador que tipo de informação está sendo tratada, isto é, se os valores binários armazenados representam números, caracteres ou outros valores e permite que uma porção correta da memória seja reservada para o armazenamento da informação.



Atividades de Aprendizagem

1. Com base na linguagem de programação utilizada no curso, especifique o tipo de dado mais adequado para as seguintes variáveis:

nome: _____

idade: _____

salário: _____

2. De acordo com o trecho de pseudo-código abaixo, e os valores definidos, informe qual o tipo correto para as variáveis informadas:

$x = 6;$

$y = 3;$

$\text{resultado} = x / y;$

x: _____

y: _____

resultado: _____

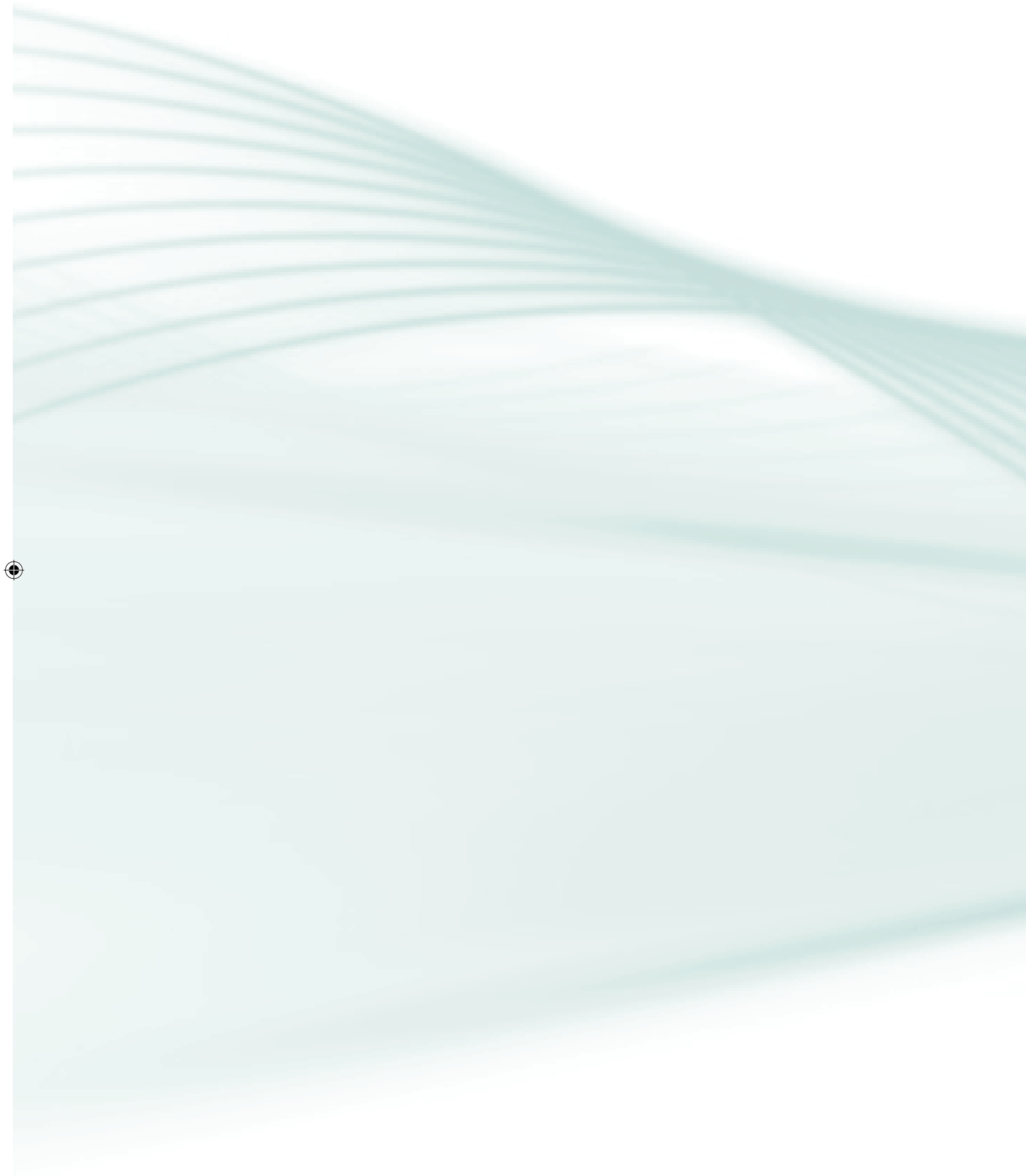
3. “Diz a lenda” que o xadrez foi criado para entretenimento de um rei o qual agradecido, solicitou que seu criador pedisse um presente. Sem hesitar, ele pediu grãos de trigo calculados da seguinte maneira: um grão na primeira casa do tabuleiro, o dobro na segunda casa e dobrando sucessivamente nas casas seguintes. Lembre-se, um tabuleiro possui 64 casas, se você fosse



criar uma variável para armazenar o total de grãos, qual seria o seu tipo de dados?

Terminamos a nossa primeira etapa. Muito bom estarmos trabalhando juntos. Espero que esteja com motivação para seguir em frente. Podemos prosseguir? Encontramo-nos na próxima aula. Até!





Aula 2. Vetores e Matrizes

Objetivos:

- organizar várias informações em uma única estrutura, para facilitar a manipulação e diminuir a complexidade do código;
- reconhecer vetores e matrizes;
- declarar e manipular vetores; e
- distinguir a declaração e percurso de uma matriz bidimensional.

Nesta nossa segunda aula, vamos estudar os vetores e as matrizes. Podemos começar com uma reflexão. Vamos lá?

Tente se imaginar em uma sala repleta de pastas, algumas contendo faturas, outras documentos de clientes e outras fichas de funcionários, espalhadas por todos os lados. Seu chefe entra e solicita que você informe o endereço de um cliente específico.

Parece uma tarefa muito difícil, principalmente se essa quantidade de pastas for maior que 1000 (mil). Verificar cada uma das pastas levaria muito tempo. Uma maneira de aperfeiçoar seu trabalho seria manter estas pastas em um arquivo, onde cada gaveta contivesse um grupo específico de pastas, uma gaveta para faturas, outra para clientes e outra para funcionários.

Sua tarefa agora ficaria um pouco mais fácil, já que teria que procurar o “cliente” na gaveta específica, o que diminuiria a quantidade de pastas para se consultar (ficaria muito mais fácil se estivessem em uma ordem específica, mas isso será tratado depois).

Existem estruturas que são semelhantes a uma pasta (uma variável), outras são semelhantes a uma gaveta do armário, onde se podem armazenar pastas contendo informações semelhantes (serão chamadas de vetores). Outras estruturas se assemelham ao próprio armário, contendo várias gavetas (serão



chamadas de matrizes). Uma matriz poderia representar até mesmo uma sala repleta de armários.



Figura 07: Arquivo com pastas

Fonte: < <http://professor-abrahao.blogspot.com.br/2011/01/para-ajudar-manter-o-site-do-bpcast.html> > Acesso em: 14 mar.2013.

A-Z

Um array também pode ser considerado como um conjunto de variáveis do mesmo tipo.

É importante que você faça essa associação, pois há situações em que nos vamos deparar com uma grande quantidade de variáveis do mesmo tipo de dado. Esperamos que, ao final desta aula, você seja capaz de manipular uma grande quantidade de variáveis do mesmo tipo de dado. Quando se tem que manusear muitos valores (considere, por exemplo, 20 (vinte) números inteiros informados pelo usuário), realizar o somatório ou identificar o maior seria um trabalho complicado, pois você teria que manipular 20 (vinte) variáveis, cada uma com um nome diferente. Teria que fazer vinte verificações para poder encontrar o maior número e o código ficaria muito extenso e complexo. Através das estruturas como vetores (**arrays**) e matrizes, essa tarefa se tornará bem mais simples, pois cada valor é associado a uma posição (parte) da estrutura e, com isso, tem-se apenas uma “variável”. Dessa forma, para acessar um valor, basta identificar a posição onde ele se encontra. Percorrer estas estruturas é mais fácil, o que torna as tarefas de busca dos valores bem mais simples.



2.1 Vetores

Um vetor ou array é uma estrutura de dado linear (uma única dimensão), composto por um determinado número (finito) de elementos (uma coleção de variáveis, as quais deverão ser do mesmo tipo de dado).



Em outras palavras, um vetor é um conjunto de elementos de um mesmo tipo de dado em que cada elemento desse conjunto é acessado através de um índice. A figura abaixo representa um vetor de nomes. Nele têm-se os valores Maria, Carlos e Ana, sendo que o índice de Maria é 0, de Carlos é 1 e o de Ana é 2.

0	1	2
Maria	Carlos	Ana

Para que possamos acessar os valores armazenados nesse vetor, temos que chamá-los pelo índice: 0, 1 ou 2.

Acessar os elementos de um vetor é muito rápido, sendo considerado o tempo constante, pois o acesso aos elementos é feito pelo seu índice. Entretanto, a operação de remoção de um elemento poderá ser complexa se for necessário que não existam espaços "vagos" no meio do vetor, pois nesse caso é necessário mover uma posição todos os elementos depois do elemento removido.

Por exemplo, para excluir o elemento Carlos:

0	1	2
Maria		Ana

Ficará a posição 1 vazia e, assim, teremos que deslocar os elementos a sua direita uma posição:

0	1	2
Maria	Ana	

Vamos aprender, agora, como declarar e manipular vetores

2.1.1 Declarando vetores

Para podermos manipular vetores, precisamos que inicialmente seja informado o tipo de dado dos elementos que serão armazenados. A declaração de um vetor pode-se diferenciar em cada linguagem de programação. Aqui, utilizaremos a linguagem JAVA para representar a declaração.

```
string[] nomes; // Declaração simples de um vetor chamado "nomes" que armazenará elementos do tipo "string".
```



A-Z

Instanciar um vetor constitui em reservar espaço de memória para armazenar os elementos. Isto é feito através do operador `new`. Além disso, é necessário definir o tamanho do vetor, neste caso, três, que é a quantidade de elementos que ele irá armazenar.

Depois de criado o vetor, devemos criar uma **instância** dele:

```
nomes = new string[3]; // Criada uma instância do vetor nomes informando que ele possuirá 3 posições.
```

Esta declaração poderá ser realizada em uma única linha:

```
string[] nomes = new string[3];
```

2.1.2 Manipulando vetores



A manipulação dos vetores se dá através das operações de inserção, consulta e remoção, muitas vezes sendo necessário percorrer os elementos do vetor para encontrar o elemento ou a posição desejada.

Vamos ver como isso funciona?

2.1.2.1 Inserindo valores

Para armazenar um valor em um vetor, é necessário fornecer um índice que indique a posição que esse elemento irá ocupar, por exemplo:

```
nomes[0] = "Maria";  
nomes[1] = "Carlos";  
nomes[2] = "Ana";
```

2.1.2.2 Consultando valores

Para consultarmos um valor do vetor na linguagem JAVA, basta informarmos o nome do vetor e o índice (sua posição no vetor), por exemplo:

```
string nome = nomes[2];
```

Este trecho de código irá atribuir a variável "nome" o valor "Ana" (se considerarmos as inserções realizadas anteriormente).

A-Z

Um valor nulo (NULL) indica que o valor é desconhecido. Um valor NULL é diferente de um valor vazio ou zero.

2.1.2.3 Excluindo valores

Para excluirmos valores de um vetor, basta atribuírmos um **valor nulo** (null) para a posição do vetor que contenha o elemento que desejamos remover ou uma *string* vazia (abrir e fechar aspas duplas).

```
nomes[2] = "";
```



Este comando irá atribuir um espaço em branco à posição 2 do vetor. Assim, teremos o vetor com os seguintes elementos:

0	1	2
Maria	Ana	

2.1.2.4 Cuidados ao manipular vetores

- A posição do primeiro elemento de um vetor é indicada pelo índice 0 (zero).
- O último elemento de um vetor de tamanho 10 (dez) é o de índice 9 (nove).
- Acessar uma posição inválida de um vetor causará um erro na execução de seu programa.



2.1.2.5 Percorrendo um vetor e listando seus valores

Para percorrer (acessar) todos os elementos de vetor, utilizaremos o comando **"for"**, conforme exemplo abaixo:

```
for (int i = 0; i < nomes.length; i++)  
    System.out.print(nomes[i]);
```

A-Z

O FOR é um comando de repetição onde uma variável é usada para contar o número de iterações (laços).

O comando `nomes.length` retorna o valor do tamanho do vetor, isto é, quantas posições ele possui. Em nosso exemplo esse valor é igual a três.

Neste trecho de código, todos os elementos do vetor serão escritos na console do sistema.

Para realizarmos uma busca de um valor dentro do vetor, sem saber se o mesmo existe ou seu índice, por exemplo, se desejamos saber em qual índice do vetor está o nome Carlos, e se ele existe:

```
for (int i = 0; i < nomes.length; i++)  
    if(nomes[i].toString() == "Carlos")  
        System.out.print("O nome Carlos está na  
posição: " + i);
```

Este código percorre o vetor e a cada volta ele compara se o valor é Carlos e, se for, ele escreve no console uma mensagem informando o índice em que se encontra a cor verde. Caso não exista, nada será escrito no console.



Pois bem, agora que aprendemos sobre vetores, chegou a hora de tratarmos das matrizes.

2.2 Matrizes



Arranjos são agrupamentos formados por n elementos, ($n > 1$) de forma que os n elementos sejam distintos entre si pela ordem ou pela natureza.



Por padronização, os programadores costumam representar estes índices por letras, o “i” para linha, o “j” para coluna, o “k” para a terceira dimensão (vamos ficar apenas com as duas dimensões) e assim por diante. A representação genérica de um índice de um elemento em uma matriz bidimensional seria $[i, j]$.

Matrizes são **arranjos** que podem possuir várias dimensões. Um quadrado desenhado em uma folha possui duas dimensões largura ou base e altura. Quando desenhamos um cubo, acrescentamos outra dimensão, a profundidade. Um vetor, por ser representado em uma “linha”, é uma matriz de uma dimensão. Matrizes de duas dimensões, ou mais, possuem propriedades semelhantes a um vetor. A diferença é que devemos utilizar o número de índices para acessar um elemento, isto é, em uma matriz de duas dimensões (bidimensional – podemos associar ao quadrado) temos um índice para a base e outro para a altura.

Por convenção, o primeiro índice de uma matriz bidimensional (primeira dimensão), é nomeado como “linha”, enquanto o segundo índice (segunda dimensão) corresponde à “coluna” onde serão armazenados os elementos.

Como vimos anteriormente, podemos comparar uma matriz bidimensional a um arquivo com gavetas, uma das dimensões representaria o número de cada gaveta, a outra dimensão representaria o número das pastas contidas em cada gaveta.

Uma matriz bidimensional de cinco linhas e cinco colunas:

		C O L U N A S				
L I N H A S		0	1	2	3	4
	0	32	78	97	43	54
	1	23	22	76	64	26
	2	44	37	43	85	67
	3	56	33	32	27	54
	4	78	54	12	17	33

Figura 08: Matriz bidimensional

Fonte: autor



Para podermos acessar o valor 22, localizamos o índice que será formado pela junção da linha “i” com a coluna “j”, $i = 1$ e $j = 1$, deste modo seu índice é [1,1]. O valor 54, por exemplo, será [0, 4].

Assim como os vetores, as matrizes têm o índice de cada linha, coluna ou qualquer outra dimensão iniciados com o valor 0 (zero). Por exemplo, o elemento posicionado na primeira linha ($i=0$) e na primeira coluna ($j=0$) da matriz terá índice [0,0].

Se acrescentarmos uma dimensão à matriz acima, teríamos uma tridimensional (cubo), a qual poderia representar uma sala com vários gaveteiros. Uma dimensão representaria os gaveteiros, outra representaria suas gavetas e, por último, a representação de pastas nas gavetas.

2.2.1 Declaração de uma matriz bidimensional

Veja como declarar uma matriz de duas dimensões:

```
int[] [] matriz1 = new int[2] [2];
```

Na sintaxe acima, declaramos uma matriz, de nome matriz1, de duas dimensões com duas linhas e duas colunas, ou seja, temos um arranjo com 4 posições:

Inicialização das posições da matriz:

```
matriz1[0][0] = 1;
matriz1[0][1] = 2;
matriz1[1][0] = 3;
matriz1[1][1] = 4;
```

Representação gráfica da matriz criada:

	Coluna 0	Coluna 1
Linha 0	1	2
Linha 1	3	4

2.2.2 Percorrendo uma matriz bidimensional

Vamos apresentar agora como percorrer cada posição da matriz:

```
for(int i = 0; i < 2; i++) {
    for(int j = 0; j < 2; j++) {
        System.out.print(matriz1[i][j]);
    }
}
```



O código acima mostra que precisamos de dois “for”, um dentro de outro. O primeiro percorre as “linhas” enquanto o interno percorre as “colunas”.

Resumo

Vimos nesta aula que podemos organizar várias informações em uma única estrutura o que facilita a manipulação e diminui a complexidade do código. Um vetor é estruturalmente uma matriz de uma dimensão, isto é, possui uma linha e várias colunas e, para criarmos um vetor, precisamos informar o tipo de dados dos elementos que ele irá armazenar. O comando “for” é o mais indicado para se manipularem os vetores e as matrizes, pois sua estrutura facilita percorrer os elementos nestas estruturas.



Atividades de Aprendizagem

1. Escreva um programa em linguagem JAVA que lê as matrículas e as notas de no máximo 50 alunos. O programa deve ler e armazenar uma nova matrícula e uma nova nota até que o usuário digite uma matrícula negativa.

2. Faça um programa que, dado o vetor {2; 4; 35; 50; 23; 17; 9; 12; 27; 5}, retorne:

a) maior valor

b) média dos valores

3. Faça um programa que leia três valores inteiros por linha de uma matriz e outros três valores por coluna e depois faça uma rotina que some todos os valores informados.

4. Escreva um algoritmo que receba as notas referentes a duas avaliações realizadas por 5 alunos e as armazene numa matriz, juntamente com a média total obtida pelo aluno.

Bem, terminamos a nossa segunda aula. Espero que o seu interesse venha aumentando a cada aula. E, então, pronto para continuar?

Aula 3. Estruturas Lineares Estáticas

Objetivos:

- empregar as listas;
- criar e utilizar uma lista simples; e
- reconhecer uma lista ligada estática.

Estamos iniciando a nossa terceira aula. Esperamos que tenha aprendido e apreendido o que trabalhamos até agora. Para começarmos, vamos pensar naquilo com que lidamos todos os dias, de várias formas, sejam mais simples ou complexas: as listas.

As listas estão presentes em muitas atividades do dia a dia como, por exemplo, uma lista de compras de supermercado. Acrescentam-se nessa estrutura vários elementos, normalmente dispostos de cima para baixo, conforme figura abaixo:

ITEM
Arroz
Feijão
Sabão em pó
Papel higiênico
Óleo

Figura 09: Lista de compras

Fonte: autor

Usualmente, você não se preocupa com a ordem dos elementos, já que vai inserindo conforme vai lembrando-se das necessidades.

Essa representação de uma lista, com elementos em uma coluna, é normalmente utilizada em anotações. Porém, se imaginar essa lista “deitada” para a esquerda, ela ficará semelhante à estrutura de um vetor.

Arroz	Feijão	Sabão em pó	Papel higiênico	Óleo
-------	--------	-------------	-----------------	------



Figura 10: Lista de compras

Fonte: ilustradora

Assim, ao final desta aula, esperamos que você seja capaz de organizar as informações em estruturas denominadas listas, onde elas serão representadas através de vetores.

A estrutura de vetor, para representar uma lista, é conhecida como lista simples, já que são mais fáceis de criar. O problema é realizar a exclusão de um elemento, pois demandará muito esforço.

Outra solução que você verá são as listas encadeadas, onde a ordem de pesquisa dos elementos é baseada em índices. Neste caso, a remoção de um elemento necessitará apenas de alterar o índice de ligação entre o elemento anterior e o posterior, diferente de uma lista simples que necessita realizar a movimentação de todos os elementos uma posição à esquerda.

3.1 Introdução

Como pudemos observar anteriormente, é simples criar as estruturas de dados que são similares ao modo em que a memória dos computadores é arranjada.

Podemos representar a memória de um computador como uma lista:



Endereço	Valor
1	1
2	23
3	43
4	22
5	5

Figura 11: Representação memória RAM

Fonte: autor.

Os valores são armazenados em uma “lista” associados a um endereço que representa a posição onde se encontram.

A relação de empregados da companhia ACME é uma **estrutura** de dados **linear**. Como a memória do computador é também linear, é fácil ver como podemos representar esta estrutura como uma lista na memória.

João	Pedro	Ana	Maria	Carla	Carlos	Marcos
------	-------	-----	-------	-------	--------	--------

Podemos observar que a visão dos empregados da empresa ACME não é exatamente a mesma da lista original (representada na aula 01). Quando nós fazemos uma lista dos nomes, nós tendemos a organizar esta lista em uma coluna melhor que em uma fileira. Neste caso, a representação conceitual de uma lista é a coluna dos nomes. Entretanto, a representação física da lista na memória de computador é uma fileira.

Formalmente, uma lista é uma coleção de elementos, cada elemento da lista é comumente chamado de **nó**. Um nó da lista pode conter mais de um tipo de informação. Em uma lista simples o índice do elemento é utilizado para sua localização. Uma lista pode estar ordenada ou não.

Uma Lista L de tamanho n pode ser vista como sendo uma sequência de elementos da seguinte forma $L = a_1, a_2, \dots, a_n$, onde:

- a_{i+1} é sucessor de a_i ($i < n$);
- a_{i-1} precede a_i ($i > 1$);
- uma lista de tamanho 0 é chamada nula ou vazia.

Exemplos:

1. dias da semana: segunda, terça, ..., domingo;

A-Z

Endereço de memória é um identificador empregado para representar um espaço de memória utilizado para armazenar informações.

Estrutura Linear é uma forma de organização de elementos semelhantes dispostos de maneira sequencial.



Podemos considerar que os nós podem representar uma estrutura que armazene o valor de uma nota e o nome de um aluno, assim, podemos ter mais de um tipo de informação sequencial.



2. cartas do baralho (um naipe): dois, três, ..., reis, ás.

E, então, como operar a lista?

3.1.1 Operações

Existe um conjunto de operações básicas que são aplicadas a uma lista para manipular seus elementos. São elas:

1. inicializar a lista;
2. encontrar a posição de um determinado elemento;
3. verificar se a lista é vazia (nula);
4. inserir um elemento na lista;
5. remover um elemento da lista; e
6. percorrer a lista.

3.1.2 Formas de representação

As listas podem ser divididas de acordo com a forma com que são dispostas na memória do computador:

- sequencial (simples): os elementos são armazenados sequencialmente na memória do computador; e
- ligada (encadeada): os elementos não obedecem a uma sequência na memória. Existe um componente responsável pela ligação entre os elementos da lista.

Existem várias maneiras de se representarem as listas. Vamos conhecê-las?

3.2 Lista simples



Uma Lista Linear Simples é uma coleção de elementos os quais são dispostos linearmente na memória.

Uma lista $L:[a_1, a_2, \dots, a_n]$, com $n > 0$, obedece às seguintes propriedades:



- a_1 é o primeiro elemento;
- a_n é o último elemento; e
- a_k , $1 < k < n$, é precedido pelo elemento a_{k-1} e seguido pelo elemento a_{k+1} .



Considerando n o número de elementos, se $n=0$, dizemos que a lista é vazia.

Uma representação sequencial permite um rápido acesso ao i -ésimo termo da lista uma vez que podemos utilizar diretamente a capacidade de indexação dos vetores.



3.2.1 Implementação de lista simples

De acordo com a definição acima, pode-se dizer que uma lista linear é um vetor de uma dimensão. Assim, a implementação de uma lista é semelhante à implementação de um vetor simples.

```
1. public class Lista {  
2.     private int[] valores;  
3.     private int tamanho;  
4. }
```

- linha 1: temos a declaração da classe Lista a qual terá as seguintes propriedades:
- linha 2: um vetor de inteiros chamado valores;
- linha 3: o tamanho máximo da lista

3.2.1.1 Inicialização da lista

Método para inicializar a lista e definir o seu tamanho.

```
1. public Lista(int tam) {  
2.     if (tam > 0) {  
3.         tamanho = tam;  
4.         valores = new int[tamanho];  
5.     }  
6. }
```

- linha 4: cria o vetor de valores definindo seu tamanho.



3.2.1.2 Inserção de elementos

Na inserção deve-se:

- verificar se a posição é válida (está dentro do tamanho da lista);
- armazenar a informação X na posição desejada.

```
1. public int Insere(int valor, int pos) {  
2.     if (pos >= 0 && pos < valores.length) {  
3.         valores[pos] = valor;  
4.         return valor;  
5.     }  
6.     else return -1;  
7. }
```

- linha 2: verifica se a posição “pos” para inserir é válida (maior/igual a zero e menor que o tamanho da lista).
- linha 3: insere o valor “valor” na posição “pos”.
- linha 6: retorna o valor -1 se a posição for inválida (menor que zero ou maior/igual ao tamanho).

3.2.1.3 Remoção de elementos

Na remoção deve-se:

- verificar se a posição é válida (está dentro do tamanho da lista);
- atribuir o valor nulo para a posição desejada.

```
1. public int Remove(int pos) {  
2.     int valor;  
3.     if (pos > 0 && pos < valores.length) {  
4.         valor = valores[pos];  
5.         valores[pos] = 0;  
6.         return valor;  
7.     }  
8.     else return -1;  
9. }
```

- linha 3: verifica se a posição “pos” para inserir é válida (maior/igual a zero e menor que o tamanho da lista).



- linha 5: atribui o valor nulo a posição “pos”.
- linha 8: posição inválida.

3.2.1.4 Iniciando a lista e resgatando valores

Para iniciar a lista, basta instanciar um objeto, l1 e, para resgatar um valor da lista, basta informar o índice (sua posição na lista), como por exemplo:

```
public static void main(String[] args) {  
    Lista l1 = new Lista(5);  
    l1.Insere(1, 0);  
    l1.Insere(2, 1);  
    System.out.print(l1.valores[1]);  
}
```

O trecho de código acima iniciará uma lista “l1”, atribuirá os valores 1 e 2 às posições 0 e 1 respectivamente e escreverá na console o valor 2.

3.2.1.5 Consultando um determinado valor

Como realizarmos uma busca de um valor dentro da lista, sem saber se o mesmo existe? Por exemplo, desejamos saber em qual índice da lista está a cor verde e se ela existe:

```
for (int i = 0; i < valores.length; i++ )  
    if(valores[i] == "verde")  
        System.out.print("O valor está no ín-  
dice: " + i);
```

Este código percorre o vetor com o comando “for” e a cada laço ele compara se o valor é verde e, se for, ele escreve na console uma mensagem informando o índice em que se encontra. Se o valor não for encontrado, nada será escrito.

Agora que aprendemos sobre as listas simples, é importante que se diga que há desvantagens no seu uso. Vamos ver quais são.

Outro tipo de lista é a lista ligada estática. Vamos estudá-la, agora.

3.3 Lista ligada estática

As listas ligadas (ou encadeadas) são utilizadas para evitar que operações de inserção e remoção dos elementos tenham **custo linear**, isto é, para evitar a necessidade de deslocamentos de partes inteiras da lista.

A-Z

O custo linear representa o esforço necessário (tempo de processamento) para deslocar parte dos elementos, a partir de uma posição, uma casa a esquerda ou direita.



As listas ligadas consistem em uma série de elementos que não estão necessariamente armazenados em posições contíguas da memória, os quais são denominados nós.

Representação gráfica de um nó



Cada elemento da lista é composto por campo adicional (ponteiro) que contém o endereço do seu sucessor. O ponteiro da última estrutura aponta para NULO indicando o fim da lista ligada.

Representação Gráfica de Lista Ligada



Representação Gráfica de Lista Ligada Nula



Para efetuarmos operações em listas ligadas, é necessário saber onde o primeiro elemento da lista se encontra, pois é a partir dele que podemos encontrar os outros elementos. Como os elementos não estão dispostos de maneira sequencial, utilizamos o ponteiro de cada elemento para saber onde está o próximo.

Considere a seguinte situação em que temos uma lista de pessoas onde cada pessoa sabe o endereço da próxima pessoa e sabemos, inicialmente, apenas onde mora a primeira pessoa da lista. Como fazer para encontrar uma determinada pessoa?

Se conhecermos o endereço da primeira pessoa, podemos ir até ela e perguntar qual o endereço da próxima pessoa já que ela tem essa informação. De tal modo, podemos ir até a próxima pessoa e perguntar o endereço da seguinte, assim sucessivamente até encontrar a pessoa desejada ou chegar ao final da lista se ela não existir.

Uma lista ligada estática é definida através de um vetor de duas dimensões com o tamanho máximo da lista representada. Uma das dimensões armazena a informação desejada. A outra dimensão armazena o campo ponteiro, um inteiro que contém o índice (endereço) do próximo elemento no vetor.



Veja a representação Gráfica

Endereço	Informação	Ponteiro
0	João	2
1	Maria	4
2	Pedro	1
3		
4	Carlos	-1

Se considerarmos que o primeiro elemento desta lista é o “João”, podemos percorrê-la da seguinte maneira:

João aponta para Pedro que aponta para Maria que aponta para Carlos que aponta para “nulo” (fim da lista).

Essa definição permite que espaços livres sejam criados na estrutura. Assim sendo, como gerenciar estes espaços livres?

- Solução: criar uma lista de espaços livres.

Realizaremos uma implementação utilizando uma matriz de duas dimensões. Na coluna 0 (zero) são armazenados os valores e na coluna 1 (um) são armazenados os ponteiros.

Observação: nesta implementação a inserção de um elemento é realizada logo após o último elemento da lista, não se considerando os espaços livres que podem surgir após a realização de exclusões.

Resumo

Utilizamos em nosso cotidiano uma infinidade de listas: compras, telefônica, clientes etc. Vimos nesta aula que as estruturas de vetores e matrizes podem ser utilizadas para representar uma estrutura linear denominada lista. Estas listas podem ser sequenciais ou ligadas. As listas sequenciais armazenam seus elementos respeitando a sequencialidade da memória do computador, enquanto as ligadas podem ter seus elementos armazenados em posições não contíguas da memória.



Atividades de Aprendizagem

1. Considerando a primeira implementação, lista simples, reescreva o método `Inserir()` de forma que possamos inserir elementos na primeira posição vazia da lista encontrada.
2. Escreva um método `Remover` que permita ao usuário informar o valor do elemento que se deseja excluir.

Assim, chegamos ao fim da nossa terceira aula. Saiba que é bom estar em sua companhia. Levante-se um pouco da cadeira, faça alguns exercícios de alongamento e vamos em frente. Até a próxima aula!

Aula 4. Estruturas Lineares Dinâmicas

Objetivos:

- reconhecer a elaboração implantação e manipulação de uma lista ligada dinâmica;
- identificar uma lista duplamente ligada; e
- distinguir uma lista circular.

Agora que você já domina a lista simples e a lista ligada estática, podemos avançar no nosso curso, trabalhando com as listas ligadas dinâmicas. Vamos para a nossa quarta aula?

Para começarmos, lembre-se das listas sequenciais estáticas, vistas anteriormente. Elas requerem a definição inicial do tamanho (número de elementos que irá armazenar), o que engessa a estrutura. Quando a lista está cheia, não é possível inserir mais elementos e uma quantidade pequena de elementos fará com que o espaço de memória reservado para o restante da lista fique ocioso. Ainda, se desejar excluir elementos, a operação poderá se tornar muito complexa e custosa, pois exige uma grande quantidade de movimentações.



Figura 12: Bloco de notas

Fonte: < <http://www.mbsdigital.com.br/blog/como-construir-sua-lista-de-email-e-manter-os-assinantes-ativos> > Acesso em: 14 mar.2013.



Uma forma prática de se manter uma lista de contatos, onde você poderá acrescentar e remover elementos, é utilizar um bloco de notas com grampo. Cada vez que desejar remover um contato, basta procurá-lo no bloco e abrir na posição do mesmo retirando a folha. Se desejar acrescentar um contato, você poderá inserir na posição mais adequada, bastando abrir o grampo e inserindo a folha nesta posição. A limitação da quantidade de contatos é determinada pelo tamanho do grampo, que, quanto maior, mais folhas poderão ser acrescentadas.



Nesta aula, vamos utilizar uma forma mais flexível de representar uma estrutura de lista, as listas ligadas dinâmicas. Essa estrutura não exige a definição de tamanho inicial, permitindo que inserções possam ser realizadas limitadas apenas pela disponibilidade de memória. Outra facilidade é a remoção de elementos e até mesmo a inserção de elementos no meio da lista, que não exige a movimentação dos outros elementos.



Figura 13: Lista de pastas

Fonte: sxc.hu

E, então, o que é uma lista ligada dinâmica?



4.1 Lista ligada dinâmica

Uma lista ligada (ou encadeada) dinamicamente é uma estrutura de dados linear e dinâmica. Ela é composta por um conjunto de nós, cada nó possuindo dois elementos: o primeiro armazena informações e o segundo armazena o ponteiro, o qual guarda informação do endereço (referências a endereços de memória) para o próximo nó na lista.

Em uma implementação de lista ligada, cada nó (item) da lista é ligado com o seguinte através do ponteiro (uma variável com o valor do endereçamento para o próximo nó). Isso significa dizer que cada nó da lista contém o registro de dados (informações) e uma variável que aponta para o próximo nó. Este tipo de implementação permite utilizar posições não contíguas de memória, sendo possível inserir e retirar elementos sem haver a necessidade de deslocar os nós seguintes da lista.

Representação Gráfica de Lista Ligada





A cabeça em uma lista ligada dinamicamente representa o primeiro elemento da lista e deve ser guardada em uma variável (primeiro) e a cauda representa o último elemento a qual também deve ser guardada em uma variável (ultimo). Assim, conhecemos, inicialmente, dois elementos da lista: o primeiro e o último. Se desejarmos manipular a lista, devemos inicialmente nos referenciar a estas variáveis e, a partir delas, percorrer a lista para identificar o restante.

O limite para a alocação dinâmica é diretamente proporcional à quantidade de memória física que está disponível na máquina, valendo, também, o tamanho do espaço físico disponível dentro do sistema de memória virtual

. Se a memória disponível for insuficiente, a expressão lança um `OutOfMemoryError`.

Vamos ver as vantagens e desvantagens da utilização desse tipo de lista, segundo o portal Tol :

Vantagens

- A inserção ou remoção de um elemento na lista não implica a mudança de lugar de outros elementos.
- Não é necessário definir, no momento da criação da lista, o número máximo de elementos que esta poderá ter. em outras palavras, é possível alocar memória "dinamicamente", apenas para o número de nós necessários.

Desvantagens

- A manipulação torna-se mais "perigosa" uma vez que, se o encadeamento (ligação) entre elementos da lista for mal feito, toda a lista pode ser perdida.
- Para aceder ao elemento na posição n da lista, deve-se percorrer os $n - 1$ anteriores.

Conhecedores dos prós e contras, vamos aprender como implementar uma lista ligada dinâmica.



Disponível em <<http://artigos.tol.pro.br/portal/linguagem-pt/Lista%20ligada>>. Acesso em: 06 dez. 2013.

4.1.1 Implementação da lista ligada dinâmica

A implementação a seguir utilizará os conceitos de Orientação a Objeto para criar a lista. Teremos duas classes: No e Lista.

Inicialmente, criamos uma Classe No e ela será a nossa base para a alocação dinâmica e representará cada elemento da lista. Contém 2 (dois) atributos fundamentais: o campo dado para receber a informação e prox para referenciar o próximo objeto da nossa lista.

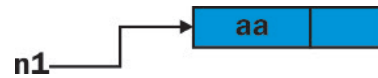
```
public class No {  
    private String dado;  
    private No prox;  
  
    //Propriedades da classe  
    public No getProx() {  
        return this.prox;  
    }  
  
    public void setProx(No prox) {  
        this.prox = prox;  
    }  
  
    public String getDado() {  
        return this.dado;  
    }  
  
    //Construtor da classe No  
    public No(String dadonovo) {  
        dado = dadonovo;  
        prox = null;  
    }  
  
    public No(String dadonovo, No ligacao) {  
        dado = dadonovo;  
        prox = ligacao;  
    }  
}
```



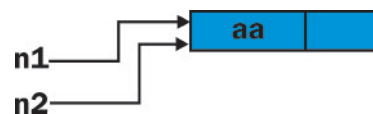
Alguns conceitos importantes relacionados à atribuição de valores a objetos:

`No n1 = new No();` //declara n1 como No, aponta para o nó criado

`n1.Dado = "aa";`

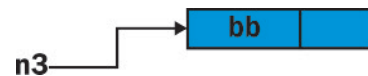


`No n2 = n1;` //declara n2 como No e atribui o endereço de n1 a n2, tanto n1 quanto n2 apontam para o nó criado.

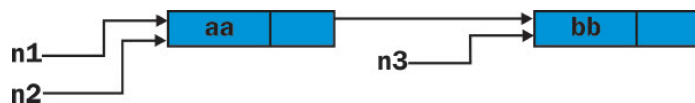


`No n3 = new No();` //declara n3 como No, aponta para o nó criado

`n3.Dado = "bb";`



`n1.Prox = n3;` //atribui o endereço de n3 ao atributo Prox do nó n1. Assim o "prox" de n1 apontará para n3. O mesmo vale para n2, pois ele "aponta" para o mesmo endereço de n1.



A classe Lista possui três atributos principais: primeiro (indica o início da lista, declarado com o tipo No), último (indica o fim da lista, declarado com o tipo No) e nomeDaLista.

Perceba que a lista é "composta" apenas de dois nós. Na verdade, é o que basta para manipularmos a lista e não precisamos conhecer o restante, pois a partir do primeiro podemos percorrer toda a lista, seguindo o conceito de que cada nó conhece o seu sucessor. Assim, o primeiro sabe qual é o próximo, sucessivamente até chegarmos ao último.



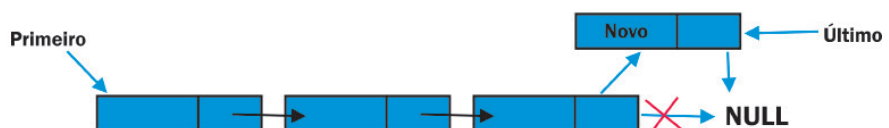
```
public class ListaLigadaDinamica {  
    private No primeiro;  
    private No ultimo;  
    private String nomeDaLista;  
  
    //Construtor da classe Lista  
    public ListaLigadaDinamica(String nome) {  
        nomeDaLista = nome;  
        //Como a lista inicialmente é vazia, tanto o  
        primeiro como o ultimo  
        // receberão o valor nulo.  
        primeiro = ultimo = null;  
    }  
  
    public ListaLigadaDinamica() {  
        nomeDaLista = "Lista Teste";  
        primeiro = ultimo = null;  
    }  
}
```

Os métodos `InsererNaFrente` e `InsererNoFundo` acrescentam, respectivamente, um elemento no início da lista e no final da lista.

Inserer na frente



Inserer no fundo

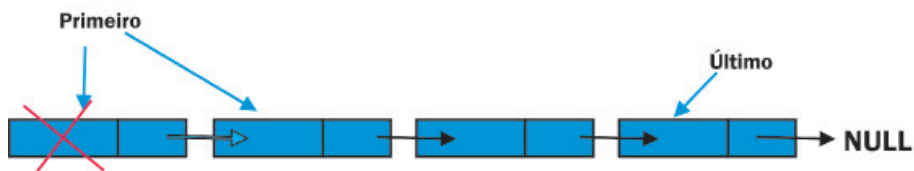




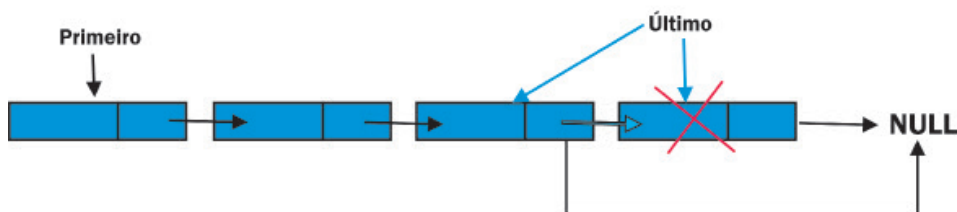
```
public void InsereNaFrente(String item) {  
    //Verifica se a lista está vazia. Se estiver,  
    o primeiro e o ultimo receberão o  
    // nó que está sendo criado.  
    if (Vazia())  
        primeiro = ultimo = new No(item);  
    //Quando a lista não está vazia o primeiro  
    recebe o nó criado e seu "prox"  
    // recebe o "antigo" primeiro.  
    else  
        primeiro = new No(item, primeiro);  
}  
  
public void InsereNoFundo(String item) {  
    if (Vazia())  
        primeiro = ultimo = new No(item);  
    else  
    {  
        ultimo.setProx(new No(item));  
        ultimo = ultimo.getProx();  
    }  
}
```

Os métodos RemoveDaFrente e RemoveDoFundo, retiram, respectivamente, um elemento do início da lista e do final da lista.

Remove da frente



Remove do fundo





```
public String RemoveDaFrente() {
    //Verifica se está vazia, se sim, não faz
    nada e retorna nulo.
    if (Vazia())
    {
        return null;
    }
    String item = primeiro.getDado();
    //Verifica se o primeiro é igual ao ultimo,
    se sim, existe apenas um
    // elemento na lista. Atribui o valor nulo
    para ambos.
    if (primeiro == ultimo)
        primeiro = ultimo = null;
    //Caso contrário, atribui o endereço do se-
    gundo nó à variável primeiro.
    else
        primeiro = primeiro.getProx();
    return item;
}

public String RemoveDoFundo() {
    if (Vazia())
    {
        return null;
    }
    String item = ultimo.getDado();
    if (primeiro == ultimo)
        primeiro = ultimo = null;
    else
    {
        No atual = primeiro;
        while (atual.getProx() != ultimo)
            atual = atual.getProx();
        ultimo = atual;
        atual.setProx(null);
    }
    return item;
}
```

O Método Vazia retorna verdadeiro se a lista está vazia e falso caso contrário.

```
public boolean Vazia() {
    //Se a variável primeiro é nula, então a
    lista não possui elementos (nós), ela está vazia
    return primeiro == null;
}
```




O Método `EscreveLista` percorre toda a lista retornando todas as informações dos nós da lista. Para percorrer uma lista ligada, usamos o atributo `prox` de cada nó.

Considere o seguinte exemplo:



A lista L1 possui os nós: `n1 => primeiro` e `n4 = ultimo`. Os nós são ligados da seguinte maneira:

```
primeiro == n1
n1.prox => n2
n2.prox => n3
n3.prox => n4
n4.prox => null
ultimo == n4
```

Como conhecemos o início da lista, a variável `primeiro` representa-o, basta segui-la, através do "`prox`", até o `ultimo`. Para isso, normalmente utilizamos uma variável (`atual`) que vai sendo associada a cada nó da lista.

No `atual = primeiro;` //A variável `atual` aponta para o nó `n1`, o início da lista.

`atual = atual.prox;` //Como `atual` aponta para `n1`, `atual.prox` aponta para `n2`. Assim, `atual` passará a apontar para `n2`.

Realizando o comando acima, sucessivamente, vamos caminhando pela lista até o último.

```
public String EscreveLista() {
    String temp = "";
    if (Vazia()) {
        temp += "Vazia " + nomeDaLista;
        return temp;
    } //if vazia
    temp += "A " + nomeDaLista + " contem \n\n";
    No atual = primeiro;
    while (atual != null) {
        temp += atual.getDado() + ", ";
        atual = atual.getProx();
    }
    temp += "\n";
    return temp;
}
```



O método `exibePrim` e `exibeUlt`, exibem, sucessivamente o primeiro e o último elemento da lista.

```
public String exibePrim() {  
    if (primeiro == null) return "Primeiro:  
null";  
    else return "Primeiro: " + primeiro.getDa-  
do();  
}  
  
public String exibeUlt() {  
    if (ultimo == null) return "Ultimo: null";  
    else return "Ultimo: " + ultimo.getDado();  
}
```

Teste da classe

```
public static void main(String[] args) {  
    ListaLigadaDinamica l1 = new ListaLigadaDin-  
amica();  
    l1.InsereNaFrente("no1");  
    l1.InsereNaFrente("no2");  
    l1.InsereNoFundo("no3");  
    System.out.print(l1.EscreveLista());  
    l1.RemoveDaFrente();  
    l1.RemoveDoFundo();  
    System.out.print(l1.EscreveLista());  
}
```

Saída do comando `System.out.print(l1.EscreveLista())`: no2, no1, no3,

Saída do comando `System.out.print(l1.EscreveLista())`: no1



As listas ligadas facilitam as operações de inserção ou remoção dos elementos (nós), pois não são necessários os deslocamentos dos demais elementos.

Ao criarmos as listas ligadas dinâmicas, não é necessário definir o seu tamanho, isto é, o número máximo de elementos que ela poderá conter. Seu tamanho é limitado pela disposição de espaço na memória, uma vez que cada elemento é adicionado "dinamicamente".

Existe, ainda, a lista duplamente ligada. Vamos conhecê-la?





4.2 Lista duplamente ligada

As listas duplamente ligadas (ou listas duplamente encadeadas) são uma extensão da lista simples ligada apresentada no item anterior.

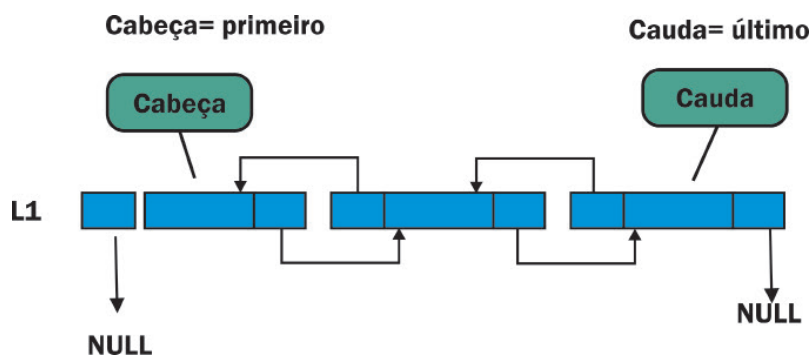
Uma lista duplamente ligada é formada por um conjunto de nós composto normalmente por três elementos, uma variável que armazena a informação, podendo ser objetos, números, caracteres etc. e dois ponteiros que possibilitam a ligação entre os nós anterior e posterior desta lista.



Assim, enquanto em uma lista simples ligada, cada nó (elemento) conhece (aponta) apenas o próximo nó, nas listas duplamente ligadas, os nós conhecem dois outros nós, seu antecessor e seu sucessor, com exceção da cabeça e da cauda em que o primeiro aponta apenas para o seu sucessor e o segundo aponta apenas para o seu antecessor.

Este tipo de solução permite percorrer uma lista nas duas direções. Por exemplo, do início (CABEÇA) até o final da lista (CAUDA), utilizaremos o ponteiro PRÓXIMO. Para percorrer a lista do final até o início (em ordem inversa), devemos começar com o último nó (CAUDA) e, através do ponteiro ANTERIOR, percorrer a lista até encontrar o primeiro nó (CABEÇA).

Representação Gráfica de Lista Duplamente Ligada



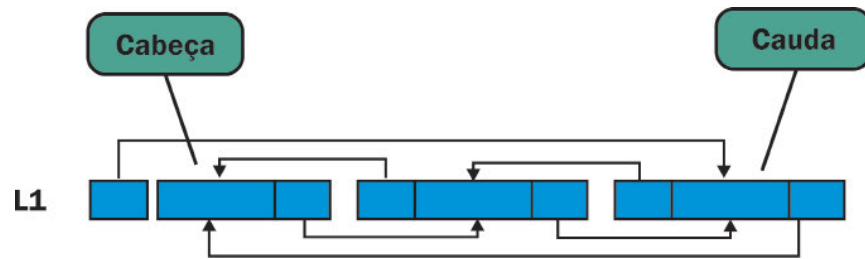
E, por falar em lista, você já ouviu falar da lista circular? Vamos ver?

4.3 Lista circular

Esta lista é uma variação da lista duplamente ligada (encadeada), onde o ponteiro "prox" do último elemento aponta para o primeiro elemento da lista e o ponteiro "ant" do primeiro elemento aponta para o último elemento da lista formando um círculo.



Representação gráfica da Lista Circular:



Resumo

As listas estão presentes em praticamente todas as organizações e sempre nos utilizamos de suas funcionalidades. Vimos nesta aula que existem estruturas de dados que são capazes de representar todas essas listas, as quais não necessitam ter seu tamanho especificado inicialmente e que são chamadas de listas ligadas ou encadeadas. Estas listas armazenam seus elementos em posições que não necessitam ser contíguas na memória do computador. As listas ligadas facilitam operações de remoção e inserção de elementos, uma vez que não necessitam de deslocamentos dos seus nós para realizar tais operações.

As listas ligadas possuem duas variações principais: as listas duplamente ligadas, as quais mantêm o conhecimento dos nós adjacentes (anterior e posterior) e as listas circulares que têm como característica principal a ligação do último nó com o primeiro, formando um círculo.



Atividades de Aprendizagem

1. Crie uma estrutura de dados que represente uma lista de passageiros em um voo.

2. Implemente uma lista de funcionários da empresa ACME e realize as seguintes operações:

a) “João” foi contratado (insira seus dados no cadastro).

b) Verifique se “Maria” é funcionária.

3. Situação: Você recebeu duas listas dinâmicas e encadeadas, de tamanhos iguais.

Implemente uma função que faça a união das duas listas, intercalando os

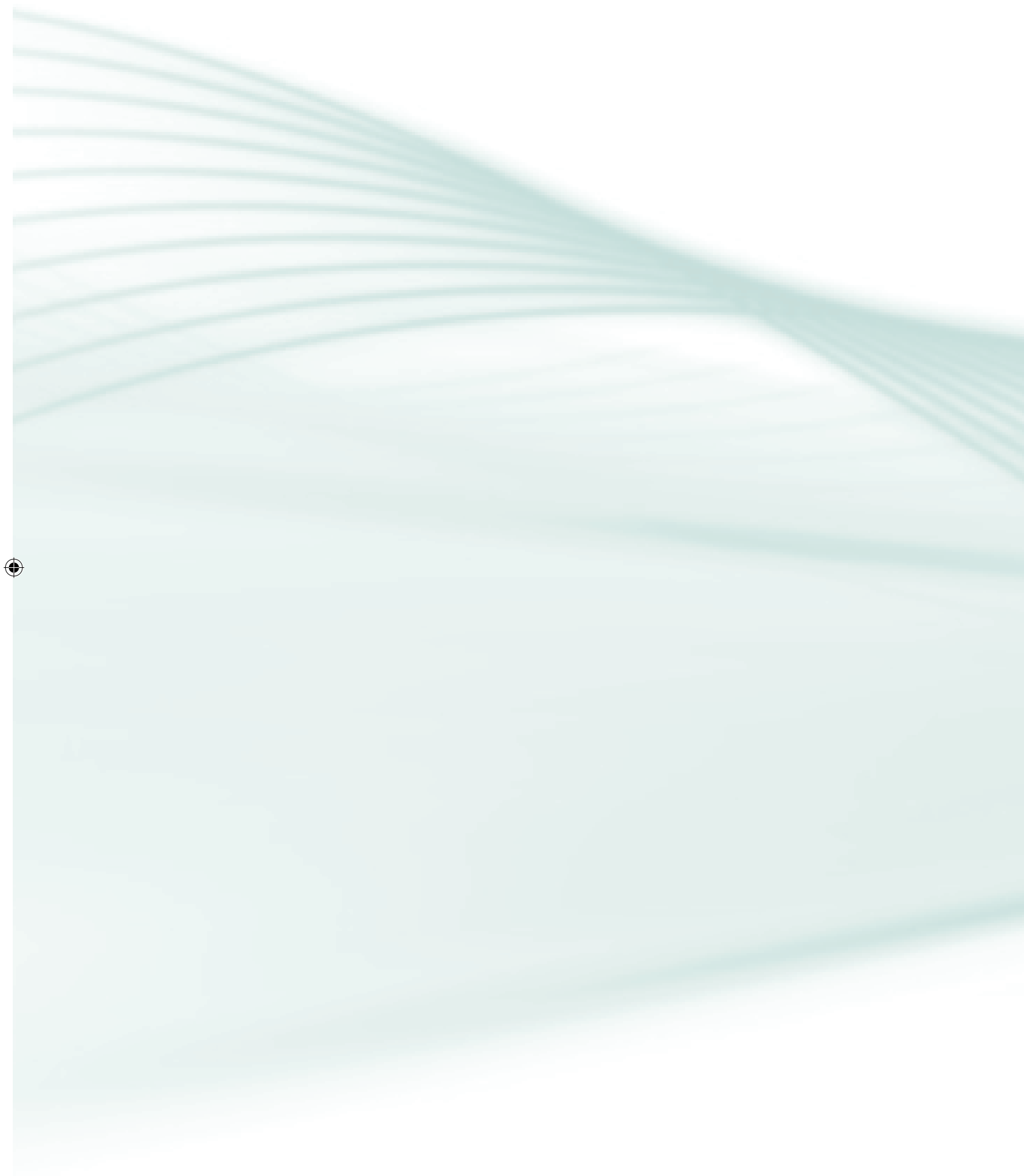


nós de cada uma das listas.

4. Escreva uma função para contar o número de elementos em uma lista encadeada.

Com esta aula, você agora está apto a trabalhar com mais três tipos de listas: a ligada dinâmica, a duplamente ligada e a circular. Com isso, chegamos à metade da nossa disciplina. Está pronto(a) pra prosseguir? Eu o(a) encontro na próxima aula.





Aula 5. Filas e Pilhas

Objetivos:

- criar estruturas de dados que representam as pilhas e filas; e
- identificar os conceitos das listas ligadas para implementação das pilhas e filas.

Estamos entrando, agora, na segunda metade do nosso percurso. Para darmos início a essa nova fase, vamos falar sobre filas e pilhas no que diz respeito aos dados. Mas, para compreendermos melhor, vamos pensar sobre isso no chamado “mundo real”:

Um dos maiores dramas dos clientes que necessitam de serviços bancários ou dos contribuintes à procura de seus direitos em muitas repartições são as filas enormes e demoradas. Sem discutir a eficiência dos serviços prestados, as filas são essenciais para garantir o direito do atendimento das pessoas que chegaram primeiro. Sem elas, seria um caos identificar quais pessoas devem ser atendidas primeiro. A saída de uma fila deve obedecer à ordem de chegada, isto é, será atendido primeiro aquele que chegou primeiro e as novas pessoas que chegam devem ficar no final da fila.



Figura 14: Fila de pessoas

Fonte: < <http://www.gargalhando.com/2010/05/27/coisas-para-se-fazer-na-fila-do-banco/> > Acesso em: 14 marc./2013.



Outra estrutura que muitas vezes é utilizada em nosso cotidiano é a pilha. Pilhas de cartas, pratos e caixas possibilitam que sejam retirados os itens de cima, mas quando colocamos um novo item, esse é posto, também, na parte de cima. Uma pilha de pratos de um restaurante self-service tem seus pratos retirados, pelos clientes, da parte de cima e, sempre que são repostos, eles são colocados na parte de cima.



Figura 14: Pilha de pratos

Fonte: sxc.hu

Você criará, ao final desta aula, estruturas de dados que representam as pilhas e filas. Muito usadas para controle da ordem de utilização do processamento dos aplicativos, os sistemas operacionais trabalham com os conceitos de filas e pilhas. Utilizará os conceitos apresentados de listas ligadas para implementá-las. Assim, elas serão representadas através da ligação de nós, obedecendo à posições de retirada e inserção dos elementos, as quais sempre serão nas extremidades.



5.1 Filas

As filas são estruturas que preservam a ordem de chegada, isto é, o primeiro a chegar será o primeiro a sair, estratégia chamada FIFO (first in, first out). Um jogo de cartas com baralhos será modelado como uma fila, se retirarmos as cartas em cima e colocarmos os descartes de volta por baixo. Exemplos do cotidiano são as filas para atendimento às pessoas.

As intervenções de inserção são realizadas no final da fila e as intervenções de remoção são realizadas no início. A inserção de um elemento torna-o último da fila. As operações básicas em uma fila são:



- Enfileira(x): insere o item x no fim da fila.
- Desenfileira(): retorna e remove o elemento do início da fila.
- Inicializa(): cria uma fila vazia.
- Cheia(), Vazia(): testa a fila para saber se ela está cheia ou vazia.

As filas são mais complexas de implementar do que as pilhas, porque as interações ocorrem nas duas extremidades (início e fim). A implementação mais simples usa um vetor, inserindo novos elementos no final e retirando do início, depois de uma remoção, movendo todos os elementos uma posição para a esquerda. Utilizaremos os conceitos de listas ligadas, vistos anteriormente, para implementação.

Funcionamento:



5.1.1 Implementação de uma fila

Normalmente, as filas são implementadas através da utilização de vetores, mas essa implementação é complexa e tem algumas desvantagens de acordo com o método adotado. Um vetor simples necessitaria de deslocamento de todos os elementos, uma posição à esquerda, para cada retirada da fila. Outra solução emprega uma lista circular no vetor e necessita de controlar os índices à medida que cada elemento é retirado ou inserido na fila.

Vamos utilizar uma solução mais prática e elegante, através de uma lista ligada. Como nestas listas utilizamos as variáveis nomeadas como primeiro e último, fica bem simples adaptá-la para uma fila. Apenas temos que considerar as operações insere na frente e remove do fundo, as outras devendo ser descartadas.

Inicialmente criamos uma Classe No, que representará cada elemento da fila.

```
public class No {  
    ver implementação no item 4.1.1  
}
```



A classe Fila possuirá os três atributos principais das listas: primeiro (indica o início da fila, declarado com o tipo No), último (indica o fim da fila, declarado com o tipo No) e nomeDaFila.

```
public class Fila {  
    private No primeiro;  
    private No último;  
    private String nomeDaFila;  
  
    //Construtor da classe Fila  
    public Fila(String nome) {  
        nomeDaFila = nome;  
        //Como a fila inicialmente é vazia, tanto  
        o primeiro como o último  
        // receberão o valor nulo.  
        primeiro = ultimo = null;  
    }  
  
    public Fila() {  
        nomeDaFila = "Fila Teste";  
        primeiro = ultimo = null;  
    }  
}
```

Não temos um método para inserir elementos na frente, apenas o método Enfileira que acrescentará os elementos no final da fila.

```
public void Enfileira(String item) {  
    if (Vazia())  
        primeiro = ultimo = new No(item);  
    else {  
        ultimo.setProx(new No(item));  
        ultimo = ultimo.getProx();  
    }  
}
```

Aqui também não temos um método para retirar elementos do fim, apenas o método Desenfileira que removerá o primeiro elemento.



```
public String Desenfileira() {  
    //Verifica se está vazia, se sim, não faz nada e  
    retorna nulo.  
    if (Vazia()){  
        return null;  
    }  
    String item = primeiro.getDado();  
    //Verifica se o primeiro é igual ao último, se  
    sim, existe apenas um  
    //elemento na fila. Atribui o valor nulo para  
    ambos.  
    if (primeiro == ultimo)  
        primeiro = ultimo = null;  
    //Caso contrário, atribui o endereço do segundo  
    nó à variável primeiro.  
    else  
        primeiro = primeiro.getProx();  
    return item;  
}
```

O Método Vazia retorna verdadeiro se a fila está vazia e falso caso contrário.

```
public boolean Vazia() {  
    //Se a variável primeiro é nula, então a fila  
    não possui elementos (nós), ela está vazia.  
    return primeiro == null;  
}
```

Teste da Classe:

```
public static void main(String[] args) {  
    Fila f1 = new Fila();  
    f1.Enfileira("el01");  
    f1.Enfileira("el02");  
    System.out.print(f1.Desenfileira());  
}
```

Saída: el02.

Exercício: Dada uma fila representada pela estrutura abaixo, realize as operações abaixo, desenhando o resultado:



Enfileira (11); Enfileira (22); Desenfileira (); Enfileira (2);





Agora que já dominou o conceito de filas, vamos aprender sobre as pilhas?

5.2 Pilhas



Ao contrário das filas, as pilhas são estruturas que não preservam a ordem de chegada, isto é, o elemento que sai será sempre o último a chegar, consistindo numa estratégia chamada LIFO (*last in, first out*). Apenas o elemento do topo da pilha poderá ser manipulado. O jogo denominado Torre de Hanói pode ser modelado como uma pilha e as peças só podem ser retiradas de cima e colocadas em cima das outras, formando as pilhas. O empilhamento de caixas e *containers* também pode ser considerado exemplo.

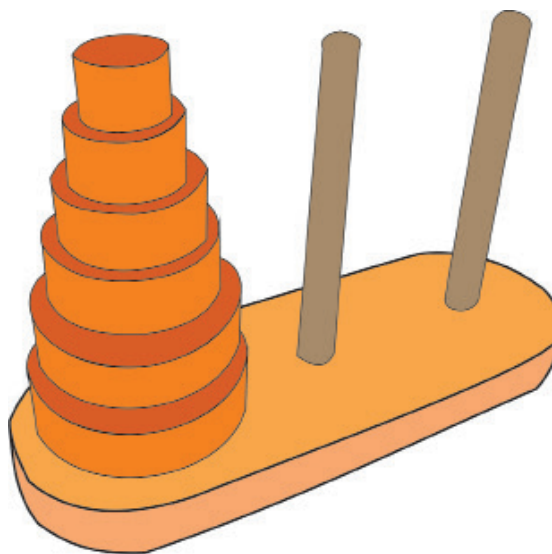


Figura 15: Torre de Hanói

Fonte: ilustradora

As intervenções de inserção e remoção são realizadas apenas no topo (final) da fila. A inserção de um elemento torna-o primeiro da pilha. As operações básicas em uma pilha são:

- Empilhar(x): insere o item x no topo da pilha.
- Desempilhar(): retorna e remove o item do topo da pilha.
- Inicializa(): cria uma pilha s vazia.
- Cheia(), Vazia(): testa a pilha para saber se ela está cheia ou vazia.



Assim como as filas, nas pilhas não temos métodos para percorrer os elementos. Trabalhar com as operações básicas acima possibilita a elaboração de uma pilha genérica, permitindo sua reutilização sem a preocupação com detalhes de implementação.

Funcionamento

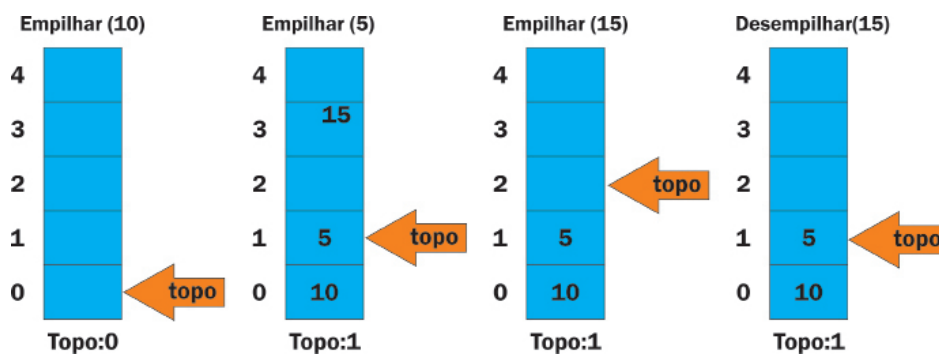


Figura 16: Funcionamento de uma pilha

Fonte: < <http://www.cos.ufrj.br/~rfarias/cos121/pilhas.html> > Acesso em: 14 març.2013.

5.2.1 Implementação de uma pilha

A implementação mais simples utiliza um vetor e uma variável para controlar o topo da pilha. A implementação utilizando uma lista ligada através de ponteiros é mais adequada, não necessitando controlar o tamanho da pilha.

Inicialmente criamos uma Classe No, que representará cada elemento da pilha.

```
public class No {  
    ver implementação no item 4.1.1  
}
```

A classe pilha possuirá os três atributos principais das listas e aqui vamos alterar sua nomenclatura: base (indica o início da pilha, declarado com o tipo No), topo (indica o fim da pilha, declarado com o tipo No) e nomeDaPilha.



```
public class Pilha {  
    private No base;  
    private No topo;  
    private String nomeDaPilha;  
  
    //Construtor da classe Pilha  
    public Pilha(String nome) {  
        nomeDaPilha = nome;  
        //Como a pilha inicialmente é vazia,  
        tanto a base como o topo  
        // receberão o valor nulo.  
        base = topo = null;  
    }  
  
    public Pilha() {  
        nomeDaPilha = "Pilha Teste";  
        base = topo = null;  
    }  
}
```

Não temos um método para inserir elementos na frente, apenas o método Empilha que acrescentará os elementos no topo (final) da pilha.

```
public void Empilha(String item) {  
    if (Vazia())  
        base = topo = new No(item);  
    else {  
        topo.setProx(new No(item));  
        topo = topo.getProx();  
    }  
}
```

Aqui também não temos um método para retirar elementos da frente, apenas o método Desempilha que removerá o topo (último elemento).

```
public String Desempilha() {  
    if (Vazia()) {  
        return null;  
    }  
    String item = topo.getDado();  
    if (base == topo)  
        base = topo = null;  
    else {  
        No atual = base;  
        while (atual.getProx() != topo)  
            atual = atual.getProx();  
        topo = atual;  
        atual.setProx(null);  
    }  
    return item;  
}
```



O Método Vazia retorna verdadeiro se a fila está vazia e falso caso contrário.

```
public boolean Vazia() {  
    //Se a variável primeiro é nula, então a fila não  
    possui elementos (nós), ela está vazia.  
    return base == null;  
}
```

Teste da classe:

```
public static void main(String[] args) {  
    Pilha p1 = new Pilha();  
    p1.Empilha("el01");  
    p1.Empilha("el02");  
    p1.Empilha("el03");  
    p1.Empilha("el04");  
    System.out.print(p1.Desempilha());  
    System.out.print(p1.Desempilha());  
}
```

Saída: el04 el03

Exercício: Dada uma pilha representada pela estrutura abaixo, realize as seguintes operações:



Empilha(11); Empilha (22); Desempilha (); Desempilha (); Empilha (2);



Resumo

Constantemente nos deparamos com estruturas semelhantes às filas e pilhas, em bancos, repartições, restaurantes etc. Estamos sempre nos envolvendo com suas regras e definições. Vimos nesta aula que existem estruturas de dados que são apropriadas para representar as pilhas e filas, utilizando os conceitos de listas ligadas. As filas devem obedecer à estrutura FIFO, isto é, o primeiro a entrar deve ser o primeiro a sair. Já as pilhas obedecem à estrutura LIFO, isto é, o último a entrar deverá ser o primeiro a sair.



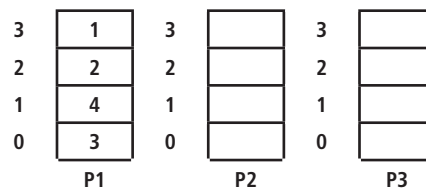
Atividades de Aprendizagem

1. Implemente os códigos apresentados na aula, para pilha e fila, e faça testes inserindo e removendo elementos das estruturas.

2. Considere um algoritmo para determinar se uma *string* de caracteres de entrada é da forma xCy , em que x é uma *string* consistindo das letras A e B e y é o inverso de x , isto é se $x = ABABBA$ então $y = ABBABA$. Qual a estrutura de dados mais adequada para implementar esse algoritmo?

3. Uma palavra é um palíndromo se tem a mesma sequência de letras, quer seja lida da esquerda para a direita ou da direita para a esquerda (exemplo: raia). Qual a Estrutura de Dados vista que melhor representaria esta situação?

4. Dado o estado inicial das pilhas $p1$, $p2$ e $p3$ na figura abaixo, mostre (desenhe as pilhas) o estado final dessas mesmas pilhas após as operações descritas no código abaixo. Considere que $p1$, $p2$ e $p3$ sejam instâncias da classe **Pilha**:



```
int temp = p1.Retira();
```




```
p2.Insere(temp);
```

```
p3.Insere(p1.Retira());
```

```
p2.Insere(p1.Retira());
```

```
temp = p1.Retira();
```

```
p3.Insere(temp);
```

```
p1.Insere(p2.Retira());
```

```
p3.Insere(p2.Retira());
```

```
p3.Insere(p1.Retira());
```

E assim terminamos mais uma etapa da nossa disciplina. Após trabalharmos as filas e pilhas, vamos, na próxima aula, falar sobre algoritmos de pesquisa. Venha comigo!



Aula 6. Classificação: Algoritmos de Pesquisa

Objetivos:

- identificar a estruturação de pesquisas para se obter a informação desejada no menor tempo possível;
- reconhecer a importância do ordenamento de dados; e
- elaborar a implementação de alguns algoritmos de pesquisa como a pesquisa sequencial e a pesquisa binária em conjunto de dados ordenados.

Você certamente já realizou algum tipo de pesquisa, em que tentou encontrar se algum elemento satisfazia a condição desejada. Vamos, agora, considerar o seguinte problema: você tem que encontrar um nome em uma lista telefônica. Sair folheando as páginas indiscriminadamente seria uma atitude pouco eficaz. De forma instintiva, procuramos o nome abrindo a lista aproximadamente na metade, um pouco mais, ou pouco menos, de acordo com a inicial do nome. Se os nomes que aparecem têm suas iniciais maiores que a do nome que estamos procurando, imediatamente dividimos as folhas da esquerda e, caso contrário, dividimos as folhas da direita. Essa operação de dividir a lista é realizada até que nos tenhamos aproximado do nome desejado e, assim podemos folhear as páginas para encontrá-lo. Esse é um método muito eficaz para realizar uma pesquisa, mas, para isso, as informações devem estar ordenadas.



Figura 17: Pesquisa com lupa

Fonte: sxc.hu (adaptado pela ilustradora)



Procurar informações em um conjunto desordenado demanda uma operação pouco produtiva, pois deve-se “olhar” cada um dos elementos até encontrar o desejado e, caso ele não exista, você terá consultado todos.

Pesquisar é uma atividade com a finalidade de encontrar se existe ou não elementos em um conjunto. Nesta aula, serão apresentados alguns algoritmos de pesquisa em um conjunto de dados ordenados, quais sejam: pesquisa sequencial, pesquisa binária e transformação de chave. A eficiência de cada uma está na posição em que o dado pesquisado se encontrar, o que poderá ser avaliado após conhecer os algoritmos.

6.1 Pesquisa



Pesquisar dados envolve determinar se um valor (denominado chave de pesquisa) está presente no conjunto de dados estudado, e, se estiver, encontrar a localização deste valor.



Na Pesquisa Exaustiva utilizam-se comandos de repetição (for, *while* etc.) da linguagem para percorrer cada um dos elementos do *array*. Cada elemento é comparado com o valor pesquisado.

A forma mais simples e também a mais onerosa é a **pesquisa exaustiva**. Nesta técnica, o algoritmo testa cada elemento até encontrar o desejado e, quando alcança o fim do *array*, informa ao usuário que a chave não está presente. Esta pesquisa é geralmente utilizada em *arrays* desordenados e a parada do algoritmo ocorre apenas quando o elemento é encontrado ou até que todos sejam pesquisados.



Para utilizarmos um dos algoritmos de pesquisas devemos ter o conjunto dos dados armazenados em um vetor. Não é possível realizar as pesquisas descritas nesta aula utilizando listas ligadas.

Quando dispomos de um conjunto de dados ordenados, podemos utilizar algoritmos de pesquisas mais eficientes, os quais utilizam estratégias que não necessitam percorrer todos os elementos para encontrar o valor pesquisado. Além disso, não é necessário percorrer todos os elementos quando o valor não é encontrado, pois normalmente os algoritmos identificam sua ausência e param o processo.

6.2 Pesquisa sequencial

Como informado anteriormente, neste algoritmo devemos considerar que o vetor está ordenado.

A busca se inicia no primeiro elemento e deve testar os elementos seguintes, devendo continuar até encontrar a chave procurada ou até encontrar o primeiro elemento com valor maior que a chave pesquisada. Como o vetor está ordenado, ao encontrar um elemento maior que a chave procurada,



não é necessário continuar verificando, pois, a partir deste elemento, todos os elementos seguintes serão maiores.

Considere o vetor de dados abaixo e a chave de pesquisa 5. A pesquisa se inicia no primeiro elemento:

1	2	5	7	9	11	12	18	19	21
---	---	---	---	---	----	----	----	----	----

Caso não encontre, é verificado o seguinte:

1	2	5	7	9	11	12	18	19	21
---	---	---	---	---	----	----	----	----	----

Os elementos seguintes são pesquisados até se encontrar o valor desejado:

1	2	5	7	9	11	12	18	19	21
---	---	---	---	---	----	----	----	----	----

Considere agora a chave de pesquisa 8:

1	2	5	7	9	11	12	18	19	21
---	---	---	---	---	----	----	----	----	----

Os elementos serão visitados até se chegar ao valor 9. Como ele é maior que a chave de pesquisa (8), o algoritmo é interrompido. Temos a certeza de que ele não estará no restante do vetor, pois os demais serão, também, maiores que ele.

Tudo bem, até aqui? Vamos aprender a implementar uma pesquisa sequencial.

6.2.1 Implementação da pesquisa sequencial

O método abaixo retornará o valor da posição do elemento com o valor pesquisado ou a posição de um elemento com o valor maior. O usuário deverá verificar se a posição retornada possui o valor pesquisado (encontrou) ou se possui um valor maior que o pesquisado (não encontrou).

O algoritmo “buscaLinear” recebe (parâmetro) a chave de pesquisa e inicia um comando de repetição (*while*) com condição de parada: até o final do vetor ou até a chave ser maior que o elemento consultado.



```
public int buscaLinear(int num) {  
    int i = 0;  
    //O comando While é interrompido quando:  
    //O contador "i" é maior que o tamanho do vetor  
    ou  
    //A chave de pesquisa (num) é maior que o elemen-  
    to do vetor. Neste  
    //caso, deve-se verificar se a chave encontrada  
    corresponde à chave  
    //de pesquisa.  
    while ((i <= vetor.lenght) && (num > vetor[i]))  
        i++;  
    return i;  
}
```

Temos, também, a pesquisa binária. Vamos aprender sobre ela?

6.3 Pesquisa binária

Este algoritmo segue a estratégia de dividir para conquistar. Deve-se encontrar o elemento central do *array* e, a partir daí, compará-lo com a chave de pesquisa. Três situações podem ocorrer:

- a) o elemento central possui o valor pesquisado, encerra-se o algoritmo;
- b) ele é maior que o valor pesquisado e, então, deve-se descartar os elementos a sua esquerda;
- c) ele é menor que o valor pesquisado então, devem-se descartar os elementos à sua direita.

Se a chave não foi encontrada, deve-se repetir o processo, com o subconjunto de dados selecionados (esquerda ou direita), até encontrá-la ou o subconjunto ser reduzido até se confirmar que o elemento não existe e, neste caso, deve-se encerrar o algoritmo.

Considere o vetor de dados abaixo e a chave de pesquisa 5. A pesquisa se inicia no elemento central, o qual é encontrado dividindo-se o tamanho do vetor (considerar a soma dos índices: 0 início e 9 fim) por 2. O resultado deverá considerar apenas a parte inteira. Neste caso temos: $(0+9) / 2 = 4,5$, usar 4:

0	1	2	3	4	5	6	7	8	9
1	2	5	7	9	11	12	18	19	21



Como o valor encontrado é maior que a chave de pesquisa, descartamos a parte direita do vetor e, em seguida, realizamos uma nova divisão. Cálculo do meio: $(0+4) / 2 = 2$:

0	1	2	3	4
1	2	5	7	9

E, finalmente, encontramos o valor pesquisado.

Na sequência, vamos compreender como se implementa a pesquisa binária.

6.3.1 Implementação da pesquisa binária

A primeira iteração testa o elemento do meio do *array*.

- a) Se isso corresponder a chave de pesquisa, o algoritmo termina.
- b) Se a chave de pesquisa for menor que o elemento do meio, a chave de pesquisa não poderá localizar nenhum elemento na segunda metade do *array* e o algoritmo continua apenas na primeira metade.
- c) Se a chave de pesquisa for maior que o elemento do meio, a chave de pesquisa não poderá localizar nenhum elemento na primeira metade do *array* e o algoritmo continua apenas com a segunda metade do *array*.

Repare que a cada iteração, metade do *array* é descartada, ficando apenas a metade que interessa. E assim sucessivamente até localizar o elemento ou reduzindo o subarray ao tamanho zero.

O algoritmo retornará um índice (posição de um elemento no vetor) que corresponderá ao valor pesquisado ou um índice com um valor diferente ao pesquisado. O usuário deverá realizar a comparação para verificar se foi encontrado.



```
public int buscaBinaria(int num) {
    int meio, inicio, fim;
    inicio = 0;
    fim = vetor.length - 1;
    //Calcula o elemento central
    meio = (int)(inicio + fim) / 2;
    //O comando While termina quando:
    //início maior ou igual ao fim ou
    //a chave de pesquisa (num) igual ao valor cen-
    tral
    while ((inicio < fim) && (num != vetor[meio])) {
        //Se chave de pesquisa for menor que o valor
        central selecionam-se os
        // elementos da esquerda.
        if (num < vetor[meio])
            fim = meio;
        //Caso contrário, selecionam-se os elementos
        da direita
        else
            inicio = meio + 1;
        meio = (int)(inicio + fim) / 2;
    }
    if (num == vetor[meio])
        return meio;
    else
        return meio + 1;
}
```

Resumo

Obter a informação desejada no menor tempo possível é diferencial para a tomada de decisões, além de ser um dos elementos principais na eficiência dos aplicativos, o qual vai ser mais “rápido” se obtiver os dados necessários em menor tempo. Vimos nesta aula que, se os dados não estiverem ordenados de alguma maneira, não temos alternativas de pesquisa e temos que realizar uma pesquisa em cada um dos elementos para verificar sua existência ou ausência. Foram, também, apresentados alguns algoritmos de pesquisa em conjunto de dados ordenados. A pesquisa sequencial visita os elementos, a partir do primeiro, um após o outro até encontrar a chave desejada, ou, se encontrar um valor maior que o pesquisado, interrompe a procura. A pesquisa binária divide o conjunto de dados em duas partes e continua procurando pela chave na parte que potencialmente ela existiria. É eleita a metade da esquerda se a chave for menor que o elemento central, caso contrário, a metade da direita é escolhida. Outro algoritmo visto é a transformação de chave que possui dois métodos, tabela de endereçamento direto, que usa a chave para definir a posição na tabela de armazenamento e tabela hash que usa uma função atribuída à chave para calcular a posição de armazenamento do elemento.



Atividades de Aprendizagem



1. De acordo com o trecho de código (abaixo) da implementação de uma pesquisa sequencial, informe qual será o valor da variável "pos" para o vetor dado:

1	5	7	9	15	17	19	21	25
---	---	---	---	----	----	----	----	----

```
int num = 18;
int pos = 0;
while ((pos<=vetor.Length) && (num>vetor[pos]))
    pos++;
Console.WriteLine(pos);
```

pos = _____

2. Considerando o conjunto de dados do exercício anterior, demonstre as etapas percorridas pela pesquisa binária para encontrar a chave de valor 21.

3. Considerando ainda o conjunto de elementos do exercício 1, faça uma análise comparativa entre a eficiência da pesquisa sequencial e da pesquisa binária para os seguintes casos:

a) Pesquisa da chave de valor 5

b) Pesquisa da chave de valor 19

E assim terminamos a nossa sexta aula. Estamos entrando na reta final da nossa disciplina. Espero-o(a) na próxima aula.



Aula 7 – Classificação: Algoritmos de Ordenação

Objetivos:

- reconhecer o processo de ordenação dos dados;
- utilizar o processo de ordenação por submersão utilizado pelo algoritmo da bolha;
- elaborar a implementação e ordenamento por seleção direta; e
- aplicar o ordenamento por inserção direta.

E, então, vamos para a nossa penúltima aula? Estamos na reta final da nossa disciplina. Aproveite bastante.

Para começarmos, vamos pensar: estamos sempre realizando classificação das coisas. Quem nunca se deparou com uma tabela de classificação de um campeonato esportivo? A classificação no vestibular, no concurso do governo, no concurso de beleza, sempre é realizada através de parâmetros com os quais desejamos especificar uma ordem. O maior número de pontos, maior número de vitórias, quem é mais bonita (se bem que é subjetiva), o mais velho, são utilizados para definir quem será o primeiro, segundo e assim por diante.

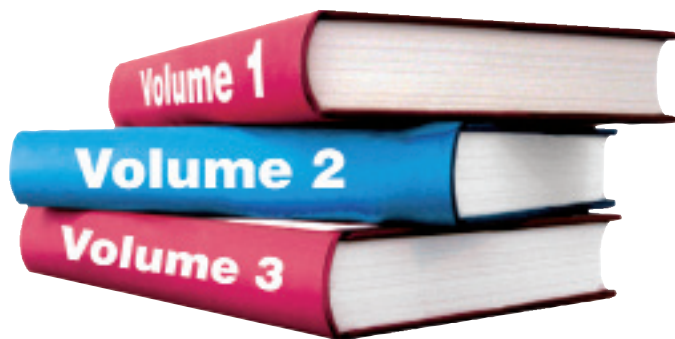


Figura 18: Ordem numérica

Fonte: < <http://www.profcardy.com/desafios/aplicativos.php?id=199> > Acesso em: 18 març.2013.



Serão apresentados nesta aula alguns dos principais tipos existentes de algoritmos de classificação (ordenação) de dados. Entre eles serão vistos os métodos bolha, que ordena os dados fazendo com que os valores menores subam para o início do *array*. O método seleção direta encontra o menor valor e o posiciona na frente, fazendo o mesmo com os demais. A inserção direta compara os dois primeiros elementos ordenando-os e, em seguida, verifica o terceiro e insere na posição correta entre os dois primeiros, fazendo esse processo sucessivamente com os demais. Já o *quicksort* é o método mais rápido de ordenação e nele se elegem um elemento do conjunto e todos os menores que são dispostos à sua direita e todos os maiores à sua esquerda. Esse processo é repetido para cada subconjunto sucessivamente até que o vetor de dados esteja ordenado.

7.1 Classificação

A classificação (ordenação) é um dos processos mais utilizados na computação.



A classificação compreende um processo de rearranjar um conjunto de dados de acordo com certa relação de ordens dentre as quais podemos citar:

- ordem alfabética;
- ordem numérica; e
- ordem cronológica.

Os tipos de ordenação acima podem ser dispostos de duas maneiras: ordem crescente, que rearranja os elementos do menor para o maior, e ordem decrescente, que rearranja os elementos do maior para o menor. Aqui, iremos considerar a ordem crescente para as implementações realizadas.

A classificação pode ser analisada sobre dois aspectos, com base na localização dos dados:

- interna: todos os dados estão contidos na memória principal; e
- externa: utilização de memória secundária.



Para motivos de estudo, consideraremos apenas o tipo interno de classificação.

Classificar dados, isto é, colocar os dados em ordem crescente ou decrescente, é essencial para a eficiência de algumas aplicações. Todos os algoritmos aqui estudados terão o mesmo resultado final: o vetor de dados classificado. A escolha do algoritmo afetará, principalmente, o tempo de execução e o uso de memória do programa. Os algoritmos da bolha, da seleção e da inserção (direta e binária) são simples de programar, porém não são tão eficientes. O método de classificação *quicksort* é o mais eficiente, mas também, é mais complicado de se implementar.

Para suporte e posterior testes da implementação dos algoritmos, vamos ver uma forma de criar um vetor com valores aleatórios:

7.1.1 Implementação de um vetor com números aleatórios

```
public void geraAleatorios() {  
    //O comando "for" irá percorrer todos as  
    posições do vetor  
    for (int i = 0; i < valores.length; i++) {  
        //Cada posição "i" do vetor receberá um  
        número aleatório entre 0 e 99  
        //O método random() da classe Math gera um  
        valor aleatório entre 0 e 1  
        //0 (inclusivo) e 1 (exclusivo).  
        //Multiplicamos por 100 para termos va-  
        lores entre 0 e 99  
        //Fazemos um cast (int) para obtermos um  
        valor inteiro  
        int num = (int) (Math.random()*100);  
        valores[i] = num;  
    }  
}
```

7.2 Bolha

O algoritmo da bolha utiliza a técnica chamada de ordenação por submersão (*sinking sort*) e funciona da seguinte forma: os valores maiores submergem para o final do *array* ou podemos também considerar que os valores menores sobem gradualmente, até o topo do *array* (isto é, em direção ao início do *array*).



Existem vários algoritmos de ordenação na literatura. Veja outros algoritmos em: http://www.dca.fee.unicamp.br/~andrerio/arquivos/Algoritmos_de_ordenacao.pdf





A ordenação pela submersão faz várias passagens pelo array. Cada passagem compara sucessivos pares de elementos. Se um par está em ordem crescente, os valores permanecem na mesma ordem. Se um par está em ordem decrescente, é realizada a troca entre os elementos do par.

Em cada passagem, o elemento maior será “levado” até o final do array e, assim, iremos percorrer na primeira passagem todos os elementos. Na segunda passagem, não precisamos percorrer todos, pois o maior estará no final e, então, percorremos até o penúltimo e assim sucessivamente até que a quantidade de elementos a percorrer seja igual a 1. Temos, então, o vetor ordenado.

Vetor desordenado

3	2	1	5	6	4
---	---	---	---	---	---

1ª Passagem: compara os dois primeiros elementos e ordena-os:

2	3	1	5	6	4
---	---	---	---	---	---

Na sequência compara o segundo com o terceiro e ordena-os:

2	1	3	5	6	4
---	---	---	---	---	---

Continuando, compara o terceiro com o quarto e ordena-os:

2	1	3	5	6	4
---	---	---	---	---	---

O quarto com o quinto:

2	1	3	5	6	4
---	---	---	---	---	---

O quinto com o sexto (último):

2	1	3	5	4	6
---	---	---	---	---	---

Repare que o vetor já está “um pouco mais ordenado” e o maior elemento ocupa a última posição.

Repetimos o processo do primeiro ao quinto elemento e podemos verificar



que o valor 5 ocupará a quinta posição. Assim, sucessivamente, vamos realizando estes passos até que a quantidade de elementos a percorrer no vetor seja igual a 1 e teremos o vetor ordenado.

1	2	3	4	5	6
---	---	---	---	---	---

7.2.1 Implementação da bolha

```
public void Bolha() {  
    int i, aux;  
    int tam = valores.length;  
    while (tam > 1) {  
        for (i = 0; i < tam - 1; i++)  
            if (valores[i] > valores[i + 1]) {  
                aux = valores[i];  
                valores[i] = valores[i + 1];  
                valores[i + 1] = aux;  
            }  
        tam = tam - 1;  
    }  
}
```

7.3 Seleção direta

Tal como o algoritmo anterior, a seleção direta percorre várias vezes o vetor. A primeira iteração do algoritmo seleciona o menor elemento e o permuta pelo primeiro elemento. A segunda iteração seleciona o segundo menor elemento (que é o menor item dos elementos restantes) e o permuta pelo segundo elemento. E, assim, sucessivamente até o penúltimo elemento.

Cada iteração percorre um elemento a menos no vetor, isto é, na primeira todos os elementos são visitados, na segunda, visitamos a partir do segundo elemento, pois o primeiro irá conter o menor elemento. Assim, sucessivamente, vamos percorrendo os trechos menores do vetor, até chegarmos aos dois últimos elementos. Quando encontramos o menor entre os dois últimos, o vetor estará ordenado.

Vetor desordenado

3	1	2	5	6	4
---	---	---	---	---	---



1ª Passagem: encontra o menor valor e troca com o primeiro elemento:

1	3	2	5	6	4
---	---	---	---	---	---

Na segunda passagem, verifica o menor a partir do segundo elemento:

1	2	3	5	6	4
---	---	---	---	---	---

Na terceira passagem, verificamos a partir do terceiro elemento e, neste caso, o menor ocupa a posição correta:

1	2	3	5	6	4
---	---	---	---	---	---

Na quarta passagem:

1	2	3	4	6	5
---	---	---	---	---	---

Na quinta e última teremos o vetor ordenado:

1	2	3	4	5	6
---	---	---	---	---	---

7.3.1 Implementação da seleção direta

```
public void Selecao() {  
    int i, j, menor, posmenor;  
    for (i = 0; i < valores.length - 1; i++) {  
        posmenor = i;  
        menor = valores[i];  
        for (j = i + 1; j < valores.length; j++)  
            if (valores[j] < menor) {  
                menor = valores[j];  
                posmenor = j;  
            }  
        valores[posmenor] = valores[i];  
        valores[i] = menor;  
    }  
}
```

7.4 Inserção direta

O algoritmo de ordenação inserção direta é considerado o mais rápido entre os algoritmos considerados simples: bolha e seleção direta.



A principal característica da inserção direta está em ordenar conjunto de elementos a partir de um subconjunto ordenado localizado em seu início (os dois primeiros elementos) e, a cada novo passo, é acrescentado a este subconjunto mais um elemento na posição adequada para se manter a ordem, até que o último elemento do conjunto seja selecionado e inserido, concluindo a ordenação do vetor.

A primeira iteração seleciona os dois primeiros elementos do vetor e, se o segundo elemento for menor que o primeiro elemento, o permuta pelo primeiro elemento (ou seja, ordena apenas os dois primeiros elementos do array). A segunda iteração seleciona o terceiro elemento e o insere na posição correta com relação aos dois primeiros elementos de modo que todos os três elementos estejam em ordem. E, assim, sucessivamente para os demais elementos.

Neste algoritmo, ao se procurar o local ideal para inserir o novo elemento entre os elementos já ordenados, pode-se optar por uma busca sequencial ou pela busca binária.

Vetor desordenado

3	1	2	5	6	4
---	---	---	---	---	---

Inicialmente os dois primeiros elementos são ordenados:

1	3	2	5	6	4
---	---	---	---	---	---

Em seguida, tomamos o terceiro elemento e inserimos na posição adequada entre os dois primeiros, de modo que eles fiquem ordenados

1	2	3	5	6	4
---	---	---	---	---	---

No próximo passo, tomamos o quarto elemento e repetimos o procedimento anterior:

1	2	3	5	6	4
---	---	---	---	---	---

Continuando, tomamos o quinto elemento:

1	2	3	5	6	4
---	---	---	---	---	---



Finalmente, tomamos o último elemento:

1	2	3	4	5	6
---	---	---	---	---	---

Podemos perceber que ao inserirmos um elemento entre o subconjunto de elementos já ordenados, precisamos realizar o deslocamento uma casa à direita de todos os elementos que forem maiores que ele.

7.4.1 Implementação inserção direta

```
public void insercaoDireta() {  
    int aux, i, j, posi;  
    //Comparamos os dois primeiros elementos, se o  
    segundo for menor,  
    // invertemos suas posições.  
    if (valores[0] > valores[1]) {  
        aux = valores[0];  
        valores[0] = valores[1];  
        valores[1] = aux;  
    }  
    for (i = 2; i < valores.length; i++) {  
        aux = valores[i];  
        //Chamamos o método buscaLinear a passamos  
        por parâmetro o  
        // elemento a ser inserido e o tamanho  
        subconjunto de elementos do  
        // vetor. A variável posi receberá a  
        posição adequada para se inserir o  
        //elemento no subconjunto.  
        posi = buscaLinear(aux, i - 1);  
        //O comando "for" desloca todos os elemen-  
        tos, a partir da posição  
        // encontrada, uma casa a direita.  
        for (j = i; j > posi; j--)  
            valores[j] = valores[j - 1];  
        //Insere o elemento na posição encontrada.  
        Ordenado em relação ao  
        // subconjunto.  
        valores[posi] = aux;  
    }  
}
```



```
public int buscaLinear(int num, int tam) {  
    int i = 0;  
    //O comando While é interrompido quando:  
    //O contador "i" maior que o tamanho do vetor ou  
    //A chave de pesquisa (num) maior que o elemento  
do vetor  
    while ((i <= tam) && (num > valores [i]))  
        i++;  
    return i;  
}
```

Resumo

Os algoritmos de ordenação são de extrema importância para a eficiência de muitos aplicativos, os quais necessitam da rapidez no acesso às informações. A ordenação é caracterizada pelo rearranjo dos elementos de um array, onde uma relação de ordem é especificada: alfabética, numérica, cronológica etc. (crescente ou decrescente). Vimos que os algoritmos apresentados são relativamente simples de implementar, não necessitando de muitos comandos ou lógica elaborada. Foram apresentados quatro tipos diferentes de algoritmos, sendo os mais comuns e simples a bolha, a seleção direta e a inserção direta, os quais são mais fáceis de implementar mas não tão rápidos quando comparados ao quicksort. O quicksort é considerado como o mais veloz, em média, dos algoritmos conhecidos, mas mais complexo de se implementar, se comparado com os outros apresentados.

Atividades de Aprendizagem

1. Escreva um programa em JAVA para ler um vetor V de tamanho definido pelo usuário e ordenar seus elementos usando o algoritmo da bolha.
2. Escreva um programa em JAVA para ler um vetor V de tamanho definido pelo usuário e ordenar seus elementos usando o algoritmo da seleção direta.
3. Escreva um programa em JAVA para ler um vetor V de tamanho definido pelo usuário e ordenar seus elementos usando o algoritmo da inserção direta.
4. Escreva um programa em JAVA que deve criar um vetor aleatório de 1000 elementos e aplique os métodos de ordenação vistos. O programa deverá exibir a quantidade de comparações realizadas por cada método.





E assim finalizamos a nossa penúltima aula. Espero que esteja tirando o melhor proveito possível. Encontramo-nos na próxima aula..



Aula 8.Árvores

Objetivos:

- reconhecer o conceito de árvore dentro da estrutura de dados;
- distinguir os tipos de representações de árvores;
- implementar árvores binárias; e
- identificar o que são árvores binárias e árvores B.

Chegamos à última aula da nossa disciplina. Falta pouco para que você complete mais um ciclo. E, então, vamos percorrer essa última etapa?

Como sabemos, em muitos casos, os pesquisadores utilizam estruturas da natureza como embasamento para suas descobertas. O movimento das formigas já foi objeto de estudos para algoritmos de definição de rotas (caminhos), o DNA e o processo de reprodução foram base para um dos principais métodos para cálculo do melhor (menor) caminho. As árvores são muito utilizadas na representação genealógica, pois sua estrutura de ramificações é ideal para a representação das conexões familiares e ou hierárquicas.

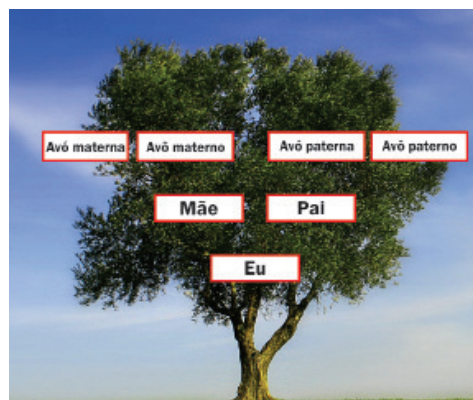


Figura 19: Árvore Genealógica
Fonte: sxc.hu (adaptado pela ilustradora)



Na computação, existem diversas aplicações que necessitam de estruturas mais complexas que as listas estudadas até agora para disposição dos seus elementos. A representação hierárquica dos funcionários da empresa ACME pode ser associada a uma árvore invertida:

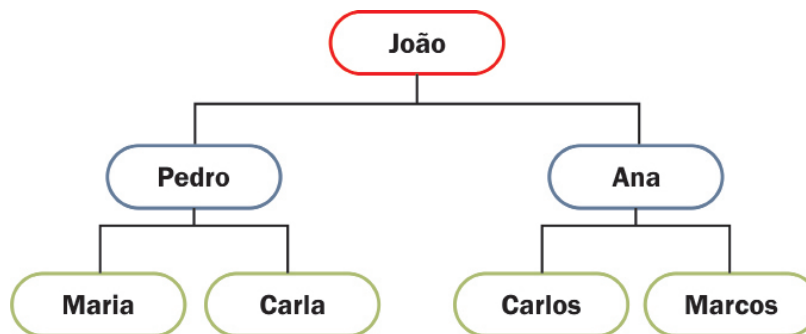


Figura 20: Representação hierárquica de quadro funcional

Fonte: autor.

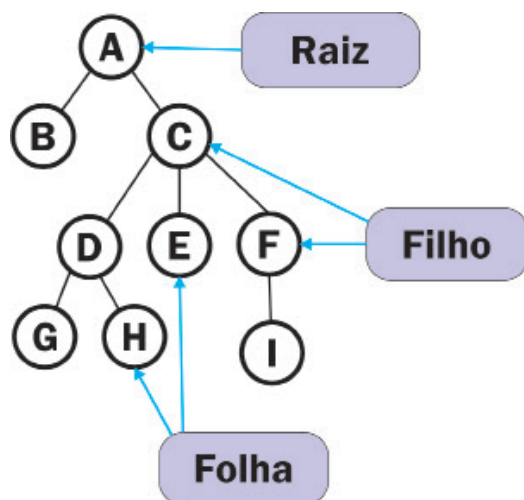
A presidência seria a raiz, os diretores o caule e a primeira ramificação e os demais cargos, seriam representados nas outras ramificações até o nível das folhas. Inúmeros problemas do cotidiano podem ser modelados através de modelos de árvores.

As árvores utilizam uma estrutura de nós, mas são conceitualmente diferentes das listas encadeadas. Nas listas, os elementos (nós) são ligados sequencialmente, já nas árvores os elementos estão dispostos de forma hierárquica. A estrutura das árvores é composta por:

- raiz: elemento principal;
- ramos ou filhos; e
- folha ou nó terminal.

A raiz é ligada aos ramos ou filhos. Estes são ligados a outros elementos que também possuem outros ramos. O elemento da extremidade, que não possui ramos, é chamado folha ou nó terminal.

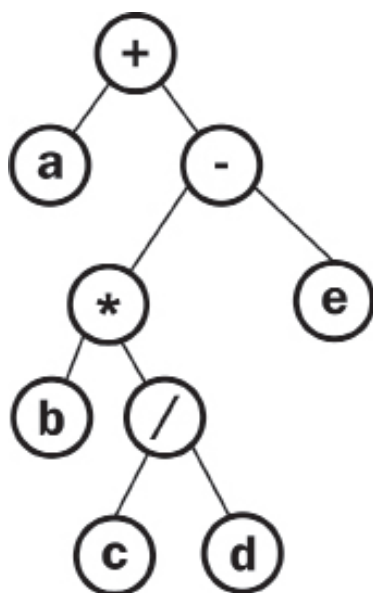
Representação gráfica:



Definições:

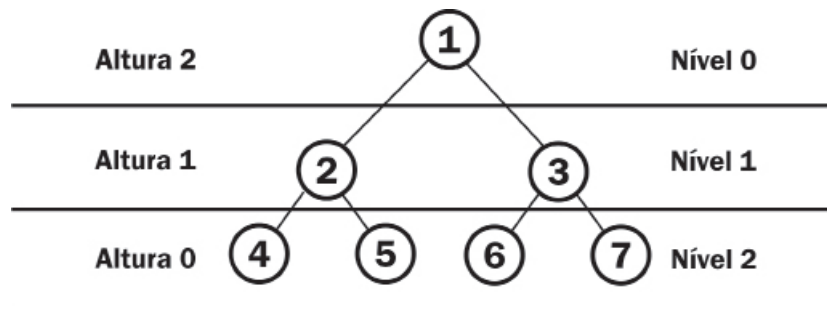
- Nós folhas: são aqueles que não têm filhos. Exemplo: {B, G, H, E, I}
- Ramos ou filhos: não são nós folhas, ou seja, são aqueles que possuem 1 ou mais descendentes. Ex: {A, C, D, F}

Exemplo de representação em árvore: expressão aritmética $a + (b * (c / d) - e)$





- Nível (profundidade) e altura de um nó



O nível ou profundidade de um nó “n” é o número de nós existente entre a raiz até o nó “n”. Portanto, o nível da raiz é 0.

A altura de um nó “n” é o número de nós no maior caminho de “n” até um de seus descendentes. As folhas têm altura 0.

- **Grau:** de um nó é o número de filhos do nó. Da árvore, é o máximo de filhos de um nó. No exemplo acima, a árvore tem grau 2 (dois).

8.1 Tipos de representações de árvores

Para começarmos a falar sobre os tipos de representações de árvores, preste atenção no texto abaixo:



Uma árvore pode ser representada, implementada, com base na disposição de seus nós (elementos) na memória:

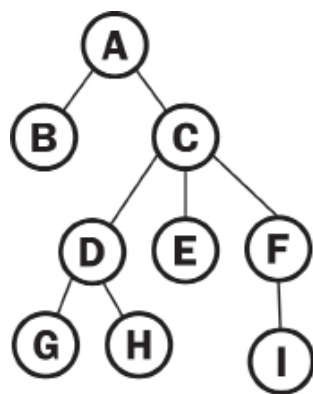
- representação por adjacência e
- representação encadeada.

Vamos conversar um pouco sobre esses dois tipos?

8.1.1 Representação por adjacência

Neste tipo de representação, os nós são armazenados sequencialmente, de acordo com uma convenção pré-determinada.

Essa representação é muito útil para armazenamento permanente (discos e fitas), mas ineficiente para manipular árvores dinâmicas (necessitam de operações de inserções e remoções de nós).



A	2	B	0	C	3	D	2	G	0	H	0	E	0	F	1	I	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

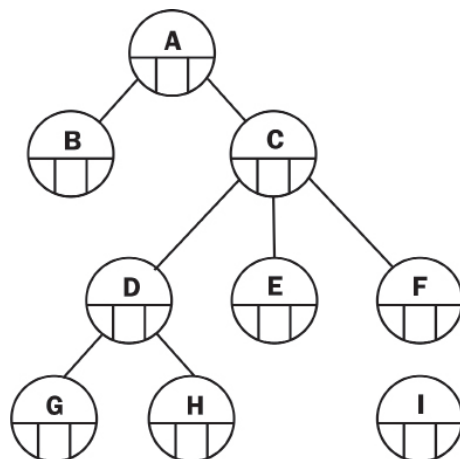
Os valores representam a quantidade de filhos que cada nó possui. Assim, "A" possui dois filhos, os nós "B" e "C". "B" não possui nenhum filho e "C" possui 3 filhos: "D" (que possui 2 filhos: "G" e "H"), "E" que não possui filhos e "F" que possui 1 filho ("I").

8.1.2 Representação encadeada

Este tipo de representação emprega os conceitos de listas encadeadas onde cada nó (elemento) contém:

- As informações do nó; e
- Referências aos ramos do nó (ponteiro para cada um dos seus nós filhos).

A desvantagem desta representação é a grande quantidade de referências a **subárvores** nulas.



Subárvore é a árvore formada abaixo de um determinado nó, o qual é definido com raiz da subárvore.

Cada nó folha possui os ponteiros para os seus possíveis nós filhos, isso faz com que eles possuam ponteiros apontando para nulo (subárvores nulas).



Na árvore acima, temos o nó-raiz “ligado” aos nós “B” e “C” e o nó “B” não possui filhos. Já o nó “C” possui ligação aos nós “D”, “E” e “F”. Nesta representação, deve-se definir inicialmente a quantidade máxima de filhos que cada nó pode possuir, que, neste exemplo, é 3. Assim, ao se implementar cada nó, a quantidade de ponteiros deve ser determinada.

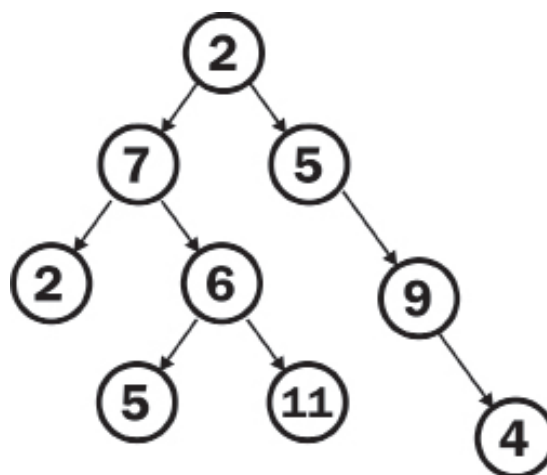
Para as implementações das árvores, nos itens a seguir, utilizaremos a representação encadeada.

8.2 Árvores binárias

Uma árvore binária é uma estrutura de dados bidimensional, com propriedades especiais.

- O nó-raiz é o primeiro nó da árvore. Cada ligação no nó-raiz referencia um filho.
- Os nós de uma árvore binária contêm, no máximo, duas ligações: o filho esquerdo e o filho direito.
- O filho esquerdo é o primeiro nó na sub-árvore esquerda (também conhecido como o nó-raiz da subárvore esquerda). E o filho direito é o primeiro nó na subárvore direita (também conhecido como o nó-raiz da subárvore direita).
- O nó sem filhos é chamado de nó-folha.

Representação gráfica de uma árvore binária:





8.2.1 Percurso em árvores binárias

O percurso em uma árvore binária é a ordem com que todos os seus nós são visitados. Vamos considerar os seguintes percursos:

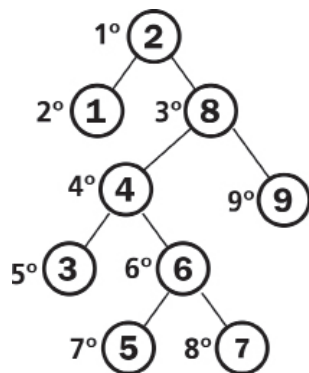
- Pré-ordem;
- In-ordem;
- Pós-ordem.

Os percursos em árvores utilizam a técnica de recursividade nos algoritmos. A recursividade é originada quando uma função chama a si própria.

8.2.1.1 Travessia em pré-ordem:

Os seguintes passos devem ser executados para se percorrer os nós da árvore:

- a) se árvore vazia, fim;
- b) exibir a informação do nó;
- c) percorrer em pré-ordem a subárvore esquerda (recursivamente);
- d) percorrer em pré-ordem a subárvore direita (recursivamente).



Resultado Pré-Ordem:
2 1 8 4 3 6 5 7 9

Percorrendo a árvore acima em Pré-Ordem



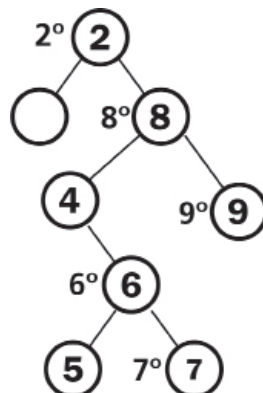
A recursividade é uma técnica muito utilizada em funções matemáticas e árvores. Leia um bom material em: <http://www.decom.ufop.br/menotti/aedl082/slides/aula06-Recursividade.ppt>



```
Escreve(2);  
PreOrdem(2, esq);  
Escreve(1);  
PreOrdem(1, esq);  
PreOrdem(1, dir);  
PreOrdem(2, dir);  
Escreve(8);  
PreOrdem(8, esq);  
Escreve(4);  
PreOrdem(4, esq);  
Escreve(3);  
PreOrdem(3, esq);  
PreOrdem(3, dir);  
PreOrdem(4, dir);  
Escreve(6);  
PreOrdem(6, esq);  
Escreve(5);  
PreOrdem(5, esq);  
PreOrdem(5, dir);  
PreOrdem(6, dir);  
Escreve(7);  
PreOrdem(7, esq);  
PreOrdem(7, dir);  
PreOrdem(8, dir);  
Escreve(9);  
PreOrdem(9, esq);  
PreOrdem(9, dir);
```

8.2.1.2 Travessia em in-ordem

- a) se árvore vazia, fim;
- b) percorrer em pré-ordem a subárvore esquerda (recursivamente);
- c) exibir a informação do nó;
- d) percorrer em pré-ordem a subárvore direita (recursivamente).

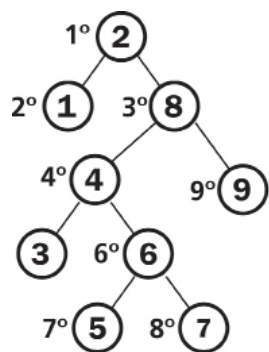


Resultado In-Ordem:
1 2 3 4 5 6 7 8 9



8.2.1.3 Travessia em pós-ordem

- a) se árvore vazia, fim;
- b) percorrer em pré-ordem a subárvore esquerda (recursivamente);
- c) percorrer em pré-ordem a subárvore direita (recursivamente); e
- d) exibir a informação do nó.



Resultado Pré-Ordem:
2 1 8 4 3 6 5 7 9

8.3 Árvores binárias de busca (pesquisa)

Uma importante aplicação de árvores binárias é a pesquisa. A finalidade desta árvore é estruturar os dados de tal maneira que possibilite uma pesquisa binária.

São consideradas árvores binárias de busca as estruturas que obedecem às seguintes características:



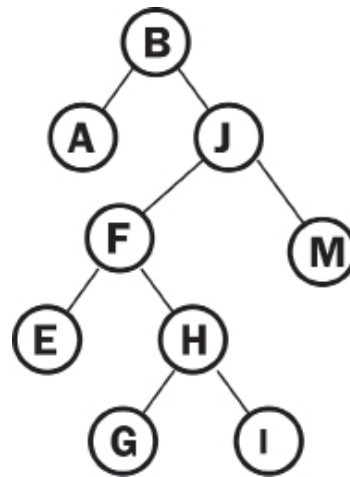
- a) todas as chaves da subárvore esquerda são menores que a chave da raiz;
- b) todas as chaves da subárvore direita são maiores que a chave raiz; e
- c) as subárvores direita e esquerda são também árvores binárias de busca.

Essa estrutura possibilita que a busca por uma determinada chave seja realizada com um número menor de comparações. A busca de uma chave "n" segue os seguintes passos:

- a) inicialmente compare com a raiz;
- b) se menor, vá para a subárvore esquerda; e



c) se maior, para a subárvore direita.



Sabendo que a árvore acima é uma árvore binária de busca, para encontrarmos a chave "H" vamos percorrer os seguintes nós: B -> J -> F -> H

Assim, começando pela raiz temos: H é maior que B, segue a subárvore da direita; "H" é menor que "J", segue a subárvore da esquerda; "H" é maior que "F", segue a subárvore da direita e finalmente encontramos a chave.



Exercício: 1. Crie a seguinte árvore binária: 47, 25, 77, 11, 43, 65, 93, 7, 17, 31, 44, 68.

2. Refaça a árvore com os mesmos números, mas em ordem inversa (comece inserindo o 68 na raiz).

3. Insira a sequência de nomes numa árvore binária de busca: Luís, Carlos, Maria, Mara, Nair, Antonio, Paulo.

Vamos ver abaixo as operações associadas ao TAD árvore binária de busca:

- inicializar uma árvore binária de busca;
- verificar se a árvore está vazia ou não;
- inserir na árvore binária;
- remover na árvore binária;
- busca na árvore (com ou sem recursividade).



8.3.1 Inserção em árvores binárias de busca

Vamos compreender como se dão as operações com árvores binárias?

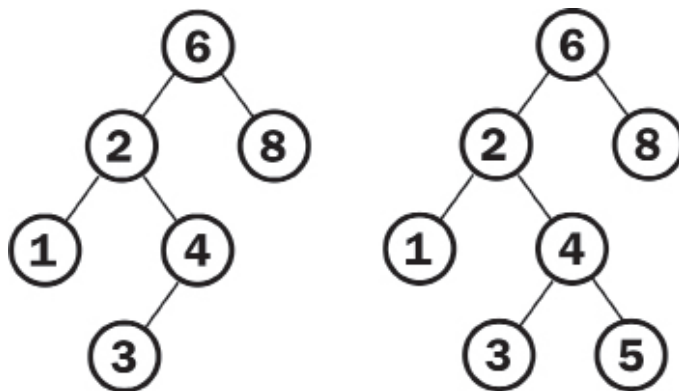
As operações em árvores binárias devem satisfazer as regras que as definem, isto é, todas as chaves da subárvore esquerda devem ser menores que a chave da raiz e todas as chaves da subárvore direita devem ser maiores que a chave da raiz.



Depois de realizada uma inserção, a árvore deverá manter suas características. Para inserir um nó na árvore, são necessários os seguintes passos:

- pesquisar a chave a ser inserida; e
- se não achar, inserir no último nó da pesquisa.

No exemplo abaixo, a chave 5 deve ser inserida na árvore:



Realizamos a pesquisa da chave 5 e, inicialmente, comparamos com a raiz 6. Como o 5 é menor, vamos pela subárvore à esquerda. Comparamos com o nó 2 e, como 5 é maior, vamos pela direita, onde comparamos com o 4 e, como 5 é maior, vamos pela direita. Como o nó 4 é uma folha, não temos mais como caminhar e, então, inserimos o novo nó.

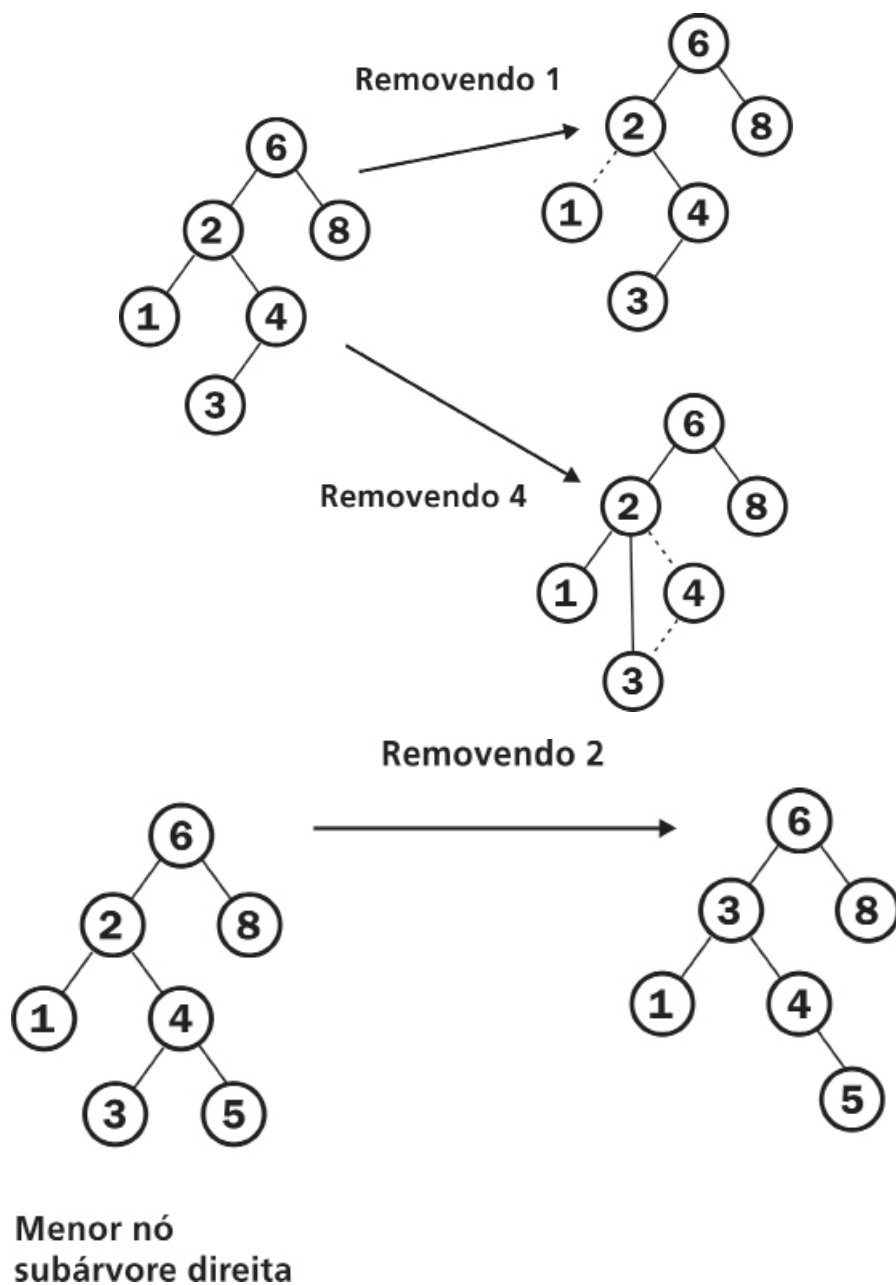
8.3.2 Remoção em árvores binárias

Para completarmos as operações, precisamos entender o processo de remoção em árvores binárias.

A operação de remoção de um nó é mais complexa e envolve alguns arranjos na árvore binária. De acordo com a posição do nó a ser removido, temos três condições que devem ser observadas:



- a) se o nó é folha, apenas remover imediatamente;
- b) se possuir um filho, pode ser removido após os ajustes do ponteiro de seu pai, isto é, seu pai deverá apontar para seu filho; e
- c) se o nó possuir dois filhos, deve-se substituir este nó com o nó da subárvore à direita que possuir a chave com o menor valor e remover aquele nó. Como o menor nó da subárvore direita não pode ter um filho à esquerda, a sua remoção segue a condição do item “a”.





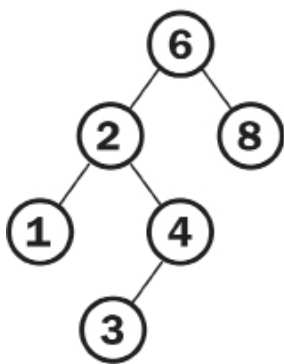
8.4 Árvores balanceadas

Árvores de pesquisa devem manter uma simetria entre as subárvores da direita e esquerda de sua raiz e assim sucessivamente para cada nó. Isso possibilita uma maior eficiência na procura de uma chave. Uma árvore que possua um dos lados muito maior que o outro (desbalanceada) perde sua eficiência na pesquisa, pois, temos que percorrer muitos nós, em sequência, para encontrar uma chave posicionada nas folhas.

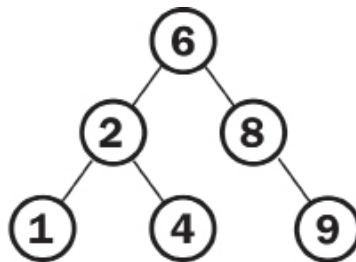
Uma árvore é considerada balanceada se, para cada nó, as alturas de suas subárvores diferem de, no máximo, 1.



Considere os seguintes exemplos de árvore binária:



Árvore 1



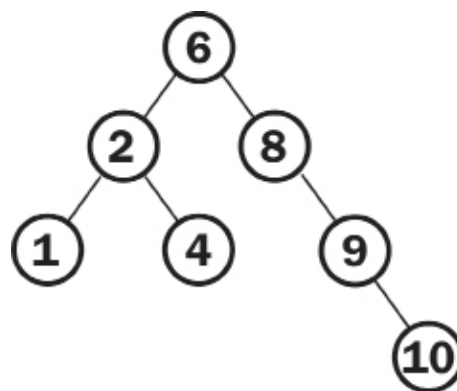
Árvore 2

Árvore 01: está desbalanceada, pois, considerando o nó 6, a altura da subárvore esquerda é igual a 3 e a altura da subárvore direita é igual a 1, a diferença $3 - 1 = 2$.

Árvore 02: está balanceada, pois todas as subárvores de cada nó têm diferença de altura em no máximo 1.

A inserção de chaves em uma árvore binária poderá provocar o seu desbalanceamento, o que pode tornar a busca tão ineficiente quanto a busca sequencial (no pior caso). A solução neste caso seria o balanceamento da árvore quando necessário.

Considerando a Árvore 02 representada acima, insira a chave 10. Teríamos o seguinte resultado:



Teríamos como resultado uma árvore binária desbalanceada. Uma vez que a altura da subárvore à direita do nó 8 é 2 e a altura da sua subárvore à esquerda é 0, portanto, teríamos uma diferença maior que 1.



Acesse o [link](http://www.cs.jhu.edu/~goodrich/dsa/trees/avltree.html) para ver uma animação do balanceamento de uma árvore após as operações de inserção e remoção: <http://www.cs.jhu.edu/~goodrich/dsa/trees/avltree.html>

8.4.1 Balanceamento de árvores binárias

Árvores AVL foram propostas pelos matemáticos russos (G.M. **Adelson-Velski** e E.M. **Landis**). Após as operações de inserção e remoção, podemos gerar o desbalanceamento da árvore. Nestes casos, precisamos nos preocupar em reparar o seu balanço. A restauração deste balanço é efetuada através de operações chamadas de rotações. [...]

Como já vimos, uma árvore é dita balanceada se a diferença entre a altura da subárvore à direita e a altura da subárvore à esquerda for no máximo 1. A esta condição damos o nome de fator de balanceamento (FB).

Balanceamento

1º - Calcular o fator de balanceamento de cada nó

$FB(nó) = (altura\ da\ subárvore\ à\ direita - altura\ da\ subárvore\ à\ esquerda)$

2º - Para uma árvore ser AVL, os fatores de balanço devem ser necessariamente -1, 0, ou 1;

3º - O nó com $FB > 1$ ou $FB < -1$ deve ser balanceado. O balanceamento de um nó é feito através de operações denominadas rotações. Substituição de um nó raiz por um de seus filhos, descendo este um nível:

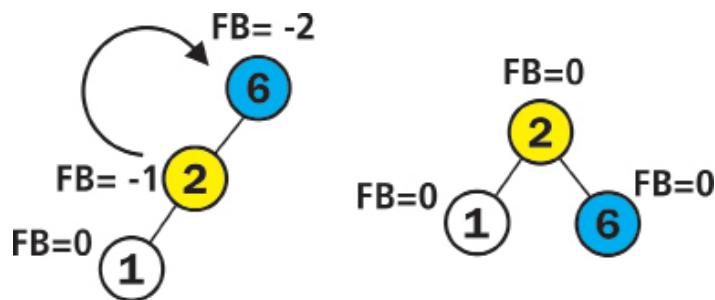
a) Rotação simples: os sinais do FB do nó desbalanceado e de seu filho são iguais, isto é, ambos negativos ou positivos. Exemplo, (+2 e +1) ou (-2 e -1). Neste caso, o nó filho deve ser posicionado no lugar da raiz da subárvore. A



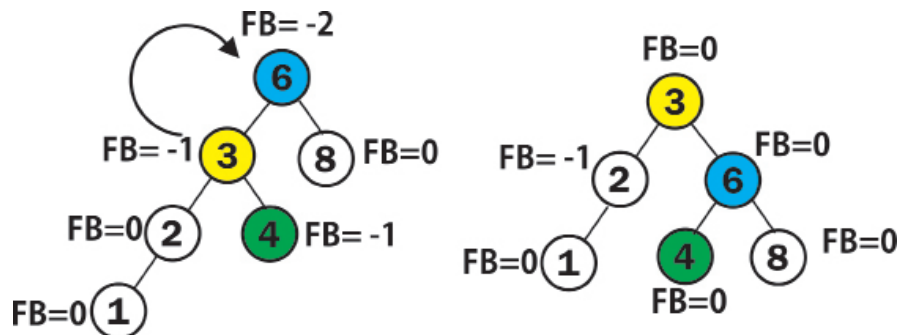
rotação pode ser à direita, quando o nó filho à esquerda toma o lugar da raiz ou à esquerda, quando o nó filho à direita toma o lugar da raiz.

b) Rotação dupla: os sinais do FB do nó desbalanceado e de seu filho são diferentes, isto é, um positivo e outro negativo. Exemplo, (+2 e -1) ou (-2 e +1). Neste caso, devemos realizar, inicialmente, uma rotação na subárvore do nó filho e em seguida fazer uma rotação simples.

8.4.1.1 Rotação simples



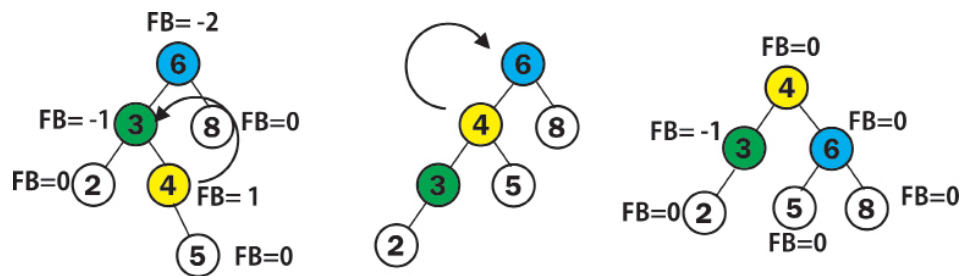
Em alguns casos as operações de rotação necessitam de remanejamento de nós para manter a característica da árvore binária de busca. Veja o exemplo abaixo:



Após a inserção do nó 1, o nó 6 tornou-se desbalanceado. Ao realizar a rotação do nó 3, seu filho, o nó 4, deve ser reposicionado na subárvore à direita.



8.4.1.2 Rotação dupla



No caso acima, o nó 6 está desbalanceado, $FB(6) = -2$ e o $FB(3) = 1$ e, como os sinais estão trocados, devemos realizar a rotação dupla. Primeiramente, realizamos a rotação simples do nó 4 com o nó 3 e fazemos os ajustes necessários e, em seguida, realizamos a rotação simples do nó 4 com o nó 6 e fazemos os ajustes necessários.

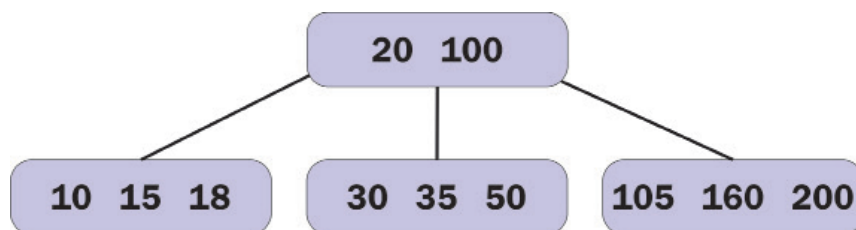
8.5 Árvores B

O que são árvores B



Árvores B (criada por Rudolf Bayer e Edward Meyers McCreight) são árvores de pesquisa balanceadas especialmente projetadas para trabalharem em memória secundária, pesquisa de informações em discos magnéticos, pois tornam mínima a quantidade de operações de movimentação de dados (escrita/leitura) numa pesquisa ou alteração.

Elas não necessitam ser balanceadas frequentemente como as árvores de busca binária. Árvores B possuem vantagens em relação a outros tipos de implementações quanto ao tempo de acesso e pesquisa aos nós.



Para saber mais sobre Árvores B, leia a apresentação no link: <http://www.ic.unicamp.br/~zanoni/mo637/aulas/arvoresB.pdf>

Estrutura do nó

Os nós em **árvores B**, também conhecidos como páginas, geralmente são representados por um conjunto de elementos (chaves) apontando para seus filhos, que por sua vez também podem ser conhecidos como folhas. Cada



elemento dos nós contém um ponteiro para uma subárvore com chaves menores e um ponteiro para uma subárvore com chaves maiores.

10 15 18

8.5.1 Estrutura árvore B

Árvore B de ordem " m " (máximo de filhos para cada nó) é uma árvore que possui as seguintes propriedades:

- a) cada nó tem no máximo " m " filhos;
- b) cada nó (exceto a raiz e as folhas) tem pelo menos " $m/2$ " filhos;
- c) a raiz tem pelo menos dois filhos, se ela mesma não for uma folha;
- d) todas as folhas aparecem no mesmo nível e carregam informação; e
- e) um nó não-folha com " k " filhos deve ter $k-1$ chaves.

Resumo

Muitas vezes necessitamos de estruturas mais complexas que as listas para uma adequada representação dos elementos manipulados. São estruturas muito utilizadas na computação. Árvores são compostas por elementos chamados nós, os quais são dispostos em uma estrutura hierárquica: um nó raiz e suas ramificações, os filhos e um nó sem filhos, que é denominado folha. Uma de suas variações, as árvores binárias, são utilizadas para pesquisas. Com um número máximo de dois filhos por nó e uma subárvore do nó podem ser compostas por chaves menores (subárvore esquerda) e por chaves maiores (subárvore) direita, que é otimizada para uma pesquisa binária. Outra variação, árvores B, são especialmente criadas para a pesquisa em memória secundária, sendo uma adaptação das árvores binárias para se adequarem aos dispositivos magnéticos.

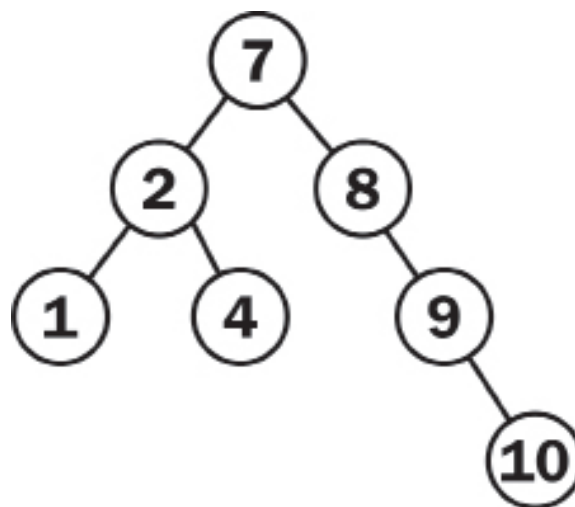
Atividades de Aprendizagem

1. Quantos ancestrais têm um nó no nível n numa árvore binária?
2. Escreva uma função que calcule o número de nós de uma árvore binária.





3. Dada a árvore binária abaixo, insira os seguintes elementos: 5 e 6



4. Realize o balanceamento da árvore resultante do exercício 3.

E assim encerramos a nossa última etapa. Espero que o conhecimento adquirido seja útil em sua vida profissional. Um grande abraço!



Palavras Finais

Caro(a) estudante,

Parabéns por completar mais esta etapa. Chegamos ao final da disciplina e durante nossa caminhada adquirimos conhecimento, pudemos realizar trocas de experiências e de ideias. Estamos mais preparados para seguir, para enfrentar novos desafios, pois este é um ponto de partida para um aprendizado maior que virá com o tempo, com a experiência.

Sabemos que não foi fácil chegar até aqui. Acreditamos que muitas vezes você se desanimou perante as dificuldades e que a fadiga foi um obstáculo cruel, mas sua determinação e persistência fizeram com que você vencesse.

Não podemos esquecer que a busca pelo conhecimento não termina aqui. Ela é cíclica e, portanto, devemos continuar procurando novos desafios. Como criar um site de relacionamento? Qual estrutura é mais adequada? Como manipular tantas informações? Esses são apenas alguns dos problemas enfrentados pelos programadores e podemos sim encará-los e resolvê-los e, para isso, temos que manter nossa disposição para continuar.





Guia de Soluções

Atividades Aula 01

1. Com base na linguagem de programação utilizada no curso, especifique o tipo de dado mais adequado para as seguintes variáveis:

nome:String //nome armazenará caracteres

idade: int //idade armazenará valores inteiros

salario: double //salario armazenará valores com casas decimais

2. De acordo com o trecho de pseudo-código abaixo e os valores definidos, informe qual o tipo correto para as variáveis informadas:

x = 6;

y = 3;

resultado = x / y;

x: int //6 é um número inteiro

y: int //3 é um número inteiro

resultado: double //receberá o resultado de uma divisão a qual poderá gerar valores com casas decimais.

3. “Diz a lenda” que o xadrez foi criado para entretenimento de um rei o qual, agradecido, solicitou que seu criador pedisse um presente. Sem hesitar, ele pediu grãos de trigo calculados da seguinte maneira: um grão na primeira casa do tabuleiro, o dobro na segunda casa e dobrando sucessivamente nas casas seguintes. Lembre-se que um tabuleiro possui 64 casas e, se você fosse criar uma variável para armazenar o total de grãos, qual seria o seu tipo de dados?

As multiplicações sucessivas dos valores irá gerar um valor extremamente grande e, portanto, o tipo de dados poderá ser double.

Atividades Aula 02

1. Escreva um programa em linguagem JAVA que lê as matrículas e as notas de no máximo 50 alunos. O programa deve ler e armazenar uma nova matrícula e uma nova nota até que o usuário digite uma matrícula negativa.

```
import java.util.Scanner;
public class Exercicios {
    public static void main(String[] args) {
        int[][]aluno = new int[50][2];
        Scanner s = new Scanner(System.in);
        for(int i=0; i<50; i++){
            System.out.println("Digite a ma-
trricula: ");
            int mat = s.nextInt();
            if (mat!=-1){
                aluno[i][0] = mat;
                System.out.println("Digite a
nota: ");
                int nota = s.nextInt();
                aluno[i][1] = nota;
            }
            else
                //Se informar -1 o comando
                "break" sairá do "for"
                break;
        }
        //Escreve os valores informados
        for(int i=0; i<50; i++){
            System.out.print(aluno[i][0] +
            ":" );
            System.out.println(aluno[i][1]);
        }
    }
}
```

2. Faça um programa que dado o vetor {2; 4; 35; 50; 23; 17; 9; 12; 27; 5} retorne:

a) maior valor

b) média dos valores



```
public class Exercicio2 {
    public static void main(String[] args) {
        int[] valores = {2, 4, 35, 50, 23, 17, 9,
12, 27, 5};
        int maior = valores[0];
        double media = 0;
        int soma = 0;
        for(int i=0; i<valores.length; i++){
            if(maior<valores[i])
                maior=valores[i];
            soma = soma + valores[i];
        }
        media = soma/valores.length;
        System.out.println("Maior: " + maior);
        System.out.print("Média: " + media);
    }
}
```

3. Faça um programa que leia três valores inteiros por linha de uma matriz e outros três valores por coluna e depois faça uma rotina que some todos os valores informados.

```
import java.util.Scanner;
public class Exercicio3 {
    public static void main(String[] args) {
        int[][]valores = new int[3][3];
        int soma = 0;
        Scanner s = new Scanner(System.in);
        for(int i=0; i<3; i++){
            for(int j=0; j<3; j++){

                System.out.println("Digite o
valor linha:" + i + " coluna: " + j);
                int val = s.nextInt();
                valores[i][j]=val;
            }
        }
        //Somatoria dos valores
        for(int i=0; i<3; i++){
            for(int j=0; j<3; j++){

                soma = soma + valores[i][j];
            }
        }
        System.out.print(soma);
    }
}
```



4. Escreva um algoritmo que receba as notas referentes a duas avaliações realizadas por 5 alunos e as armazene numa matriz, juntamente com a média total obtida pelo aluno.

```
import java.util.Scanner;
public class Exercicio4 {
    public static void main(String[] args) {
        double [][]notas = new double[5][3];
        double media;
        Scanner s = new Scanner(System.in);

        for(int i=0; i<5; i++){
            media = 0;
            System.out.println("Digite a primeira nota do aluno:" + i);
            double n1 = s.nextDouble();
            notas[i][0]=n1;
            media = media + n1;
            System.out.println("Digite a segunda nota do aluno:" + i);
            double n2 = s.nextDouble();
            notas[i][1]=n2;
            media = media + n2;
            notas[i][2] = media;
        }
        //Escreve os valores
        for(int i=0; i<5; i++){
            for(int j=0; j<3; j++){

                System.out.print(notas[i][j]
+ ":");

            }
            System.out.println();
        }
    }
}
```

Atividades Aula 03

1. Considerando a primeira implementação, lista simples, reescreva o método Inserir() de forma que possamos inserir elementos na primeira posição vazia da lista encontrada.

```
public int InserePosicaoVazia(int valor){
    for (int i=0; i< valores.length; i++) {
        if(valores[i] == 0) {
            valores[i]=valor;
            return 1;
        }
    }
    return -1;
}
```



2. Escreva um método Remove que permita ao usuário informar o valor do elemento que se deseja excluir.

```
public int RemoveValor(int valor) {  
    for (int i = 0; i < valores.length; i++)  
        if (valor == valores[i])  
        {  
            valores[i] = 0;  
            return i;  
        }  
    return -1;  
}
```

Atividades Aula 04

1. Crie uma estrutura de dados que represente uma lista de passageiros em um voo.

A solução desta questão é baseada na implementação, apresentada na aula, de uma lista ligada, onde o nó deverá possuir além do ponteiro os atributos nome e voo.

2. Implemente uma lista de funcionários da empresa ACME e realize as seguintes operações:

a) “João” foi contratado (insira seus dados no cadastro)

b) Verifique se “Maria” é funcionária?

Utilizar a implementação apresentada na aula.

Respostas:

a) utilize a inserção;

```
b) public String Consulta(String nome) {  
    No atual = primeiro;  
    while (atual != null) {  
        if (atual.getDado() == nome)  
            return nome;  
        atual = atual.getProx();  
    }  
    return "Não encontrado."  
}
```



3. Situação: Você recebeu duas listas dinâmicas e encadeadas, de tamanhos iguais;

Implemente uma função que faça a união das duas listas, intercalando os nós de cada uma das listas.

```
public ListaLigadaDinamica ConcatenaListas(ListaLigadaDinamica l1, ListaLigadaDinamica l2) {
    ListaLigadaDinamica l3 = new ListaLigadaDinamica();
    No atual1 = l1.primeiro;
    No atual2 = l2.primeiro;
    while (atual1 != null) {
        l3.InsereNoFundo(atual1.getDado());
        l3.InsereNoFundo(atual2.getDado());
        atual1 = atual1.getProx();
        atual2 = atual2.getProx();
    }
    return l3;
}
```

4. Escreva uma função para contar o número de elementos em uma lista encadeada.

```
public int ContaElementos() {
    int num = 0;
    No atual = primeiro;
    while (atual != null) {
        num++;
        atual = atual.getProx();
    }
    return num;
}
```

Atividades Aula 05

1. Implemente os códigos apresentados na aula, para pilha e fila, e faça testes inserindo e removendo elementos das estruturas.

Esta atividade propõe apenas a digitação do código apresentado na aula.

2. Considere um algoritmo para determinar se uma *string* de caracteres de entrada é da forma xCy , onde x é uma string consistindo das letras A e B e y é o inverso de x , isto é se $x = ABABBA$ então $y = ABBABA$. Qual a estrutura de dados mais adequada para implementar esse algoritmo?

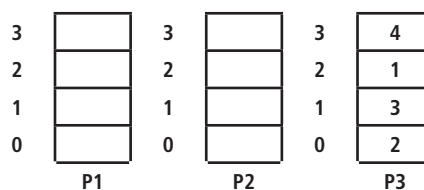


A estrutura mais indicada é a pilha, uma vez que y é o inverso de x . Quando empilhamos em x os elementos (1, 2, 3), podemos montar y apenas desempilhando x .

3. Uma palavra é um *palíndromo* se tem a mesma sequência de letras, quer seja lida da esquerda para a direita ou da direita para a esquerda (exemplo: raíar). Qual a estrutura de dados vista que melhor representaria esta situação?

Aqui podemos utilizar tanto a pilha quanto a fila, já que a ordem de retirada não vai influir no resultado final. Se enfileirarmos em x os elementos (1221), ao desenfileirar x em y temos (1221) e o mesmo vale para a pilha.

4. Dado o estado inicial das pilhas $p1$, $p2$ e $p3$ na figura abaixo, mostre (desenhe as pilhas) o estado final dessas mesmas pilhas após as operações descritas no código abaixo. Considere que $p1$, $p2$ e $p3$ sejam instâncias da classe **Pilha**:



`int temp = p1.Retira();` Retira o elemento 1 de P1 e armazena em temp.

`p2.Insere(temp);` Insere em P2 o valor 1 armazenado em temp.

`p3.Insere(p1.Retira());` Retira o elemento 2 de P1 e insere em P3.

`p2.Insere(p1.Retira());` Retira o elemento 4 de P1 e insere em P2.

`temp = p1.Retira();` Retira o elemento 3 de P1 e armazena em temp.

`p3.Insere(temp);` Insere em P3 o valor 3 armazenado em temp.

`p1.Insere(p2.Retira());` Retira o elemento 4 de P2 e insere em P1.

`p3.Insere(p2.Retira());` Retira o elemento 1 de P2 e insere em P3.



p3.Insere(p1.Retira()); Retira o elemento 1 de P1 e insere em P3.

Atividades Aula 06

1. De acordo com o trecho de código (abaixo) da implementação de uma pesquisa sequencial, informe qual será o valor da variável "pos" para o vetor dado:

```
int num = 18;  
int pos = 0;  
while ((pos<=vetor.Lenght) && (num>vetor[pos]))  
    pos++;  
Console.WriteLine(pos);
```

1	5	7	9	15	17	19	21	25
---	---	---	---	----	----	----	----	----

O trecho de código fará as seguintes comparações para num = 18:

na pos = 0; a chave 1 é menor que o num, continua no laço do while;

na pos = 1; a chave 5 é menor que o num, continua no laço do while;

na pos = 2; a chave 7 é menor que o num, continua no laço do while;

na pos = 3; a chave 9 é menor que o num, continua no laço do while;

na pos = 4; a chave 15 é menor que o num, continua no laço do while;

na pos = 5; a chave 17 é menor que o num, continua no laço do while;

na pos = 6; a chave 19 é maior que o num, sai do laço do while;

Resposta: pos = 6

2. Considerando o conjunto de dados do exercício anterior, demonstre as etapas percorridas pela pesquisa binária para encontrar a chave de valor 21.

ini = 0; fim = 8; meio = 4

0	1	2	3	4	5	6	7	8
1	5	7	9	15	17	19	21	25



Como a chave de pesquisa é maior que o elemento central, continuamos com o subconjunto da direita:

$$\text{meio} = (4 + 8) / 2 = 6$$

4	5	6	7	8
15	17	19	21	25

Como a chave de pesquisa é maior que o elemento central, continuamos com o subconjunto da direita:

$$\text{meio} = (6 + 8) / 2 = 7$$

6	7	8
19	21	25

Encontramos.

3. Considerando ainda o conjunto de elementos do exercício 1, faça uma análise comparativa entre a eficiência da pesquisa sequencial e da pesquisa binária para os seguintes casos:

a) Pesquisa da chave de valor 5

Pesquisa sequencial encontrará a chave na segunda comparação; e

Pesquisa binária encontrará a chave após 3 ciclos de divisões.

b) Pesquisa da chave de valor 19

Pesquisa sequencial encontrará a chave apenas na oitava comparação; e

Pesquisa binária encontrará a chave após 3 ciclos de divisões.

Atividades Aula 07

1. Escreva um programa em JAVA para ler um vetor V de tamanho definido pelo usuário e ordenar seus elementos usando o algoritmo da bolha.


```

import java.util.Scanner;
public class Exerciciol {
    public static int[] Bolha(int[] v) {
        int i, aux;
        int tam = v.length;
        while (tam > 1) {
            for (i = 0; i < tam - 1; i++)
                if (v[i] > v[i + 1]) {
                    aux = v[i];
                    v[i] = v[i + 1];
                    v[i + 1] = aux;
                }
            tam = tam - 1;
        }
        return v;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int[] valores;
        System.out.println("informe o tamanho
do vetor:");
        int tam = s.nextInt();
        valores = new int[tam];
        for(int i=0; i<tam; i++){
            System.out.println("Digite o
valor da posição " + i + ":");
            valores[i] = s.nextInt();
        }
        int[] ordenado = Bolha(valores);
        //Escreve os valores
        for(int i=0; i<ordenado.length; i++){
            System.out.
print(ordenado[i] + ":");
        }

        System.out.println();
    }
}

```



2. Escreva um programa em JAVA para ler um vetor V de tamanho definido pelo usuário e ordenar seus elementos usando o algoritmo da seleção direta.

Com base na resolução anterior, utilize o método `SelecaoDireta` apresentado na aula:

```
public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int[] valores;
    System.out.println("informe o tamanho do
vetor:");
    int tam = s.nextInt();
    valores = new int[tam];
    for(int i=0; i<tam; i++){
        System.out.println("Digite o valor
da posição " + i + ":");
        valores[i] = s.nextInt();
    }
    int[] ordenado = SelecaoDireta(valores);
    //Escreve os valores
    for(int i=0; i<ordenado.length; i++){
        System.out.print(ordenado[i] +
":");
    }
    System.out.println();
}
```

3. Escreva um programa em JAVA para ler um vetor V de tamanho definido pelo usuário e ordenar seus elementos usando o algoritmo da inserção direta.

Com base na resolução anterior, utilize o método `InsercaoDireta` apresentado na aula:

```

public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int[] valores;
    System.out.println("informe o tamanho do
vetor:");
    int tam = s.nextInt();
    valores = new int[tam];
    for(int i=0; i<tam; i++){
        System.out.println("Digite o valor
da posição " + i + ":");
        valores[i] = s.nextInt();
    }
    int[] ordenado = InsercaoDireta(valores);
    //Escreve os valores
    for(int i=0; i<ordenado.length; i++){
        System.out.print(ordenado[i] +
":");
    }
    System.out.println();
}

```

4. Escreva um programa em JAVA que deve criar um vetor aleatório de 1000 elementos e aplique os métodos de ordenação vistos. O programa deverá exibir a quantidade de comparações realizadas por cada método.

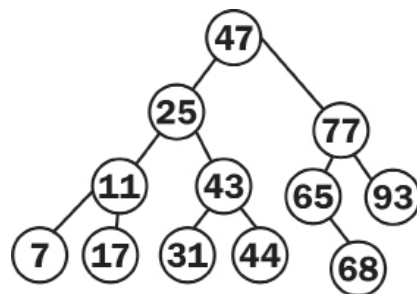
Para a solução desta questão, basta acrescentar um contador nos laços de cada método e, assim, cada vez que ele é acionado, o contador é incrementado de 1. Ao final, teremos a quantidade de vezes que as comparações foram realizadas.

Como o vetor é aleatório, veremos que, a cada vez que o programa é executado, teremos valores distintos na resposta.

Atividades Aula 08

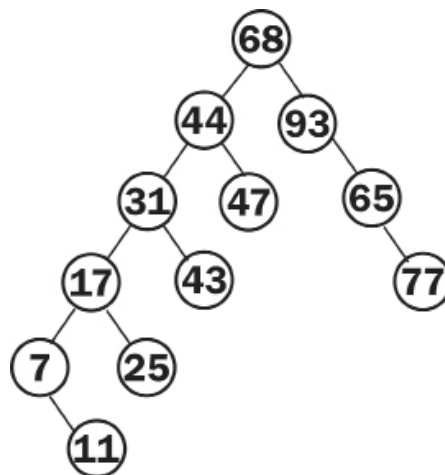
Exercícios item 8.3

1. Crie a seguinte árvore binária: 47, 25, 77, 11, 43, 65, 93, 7, 17, 31, 44, 68.

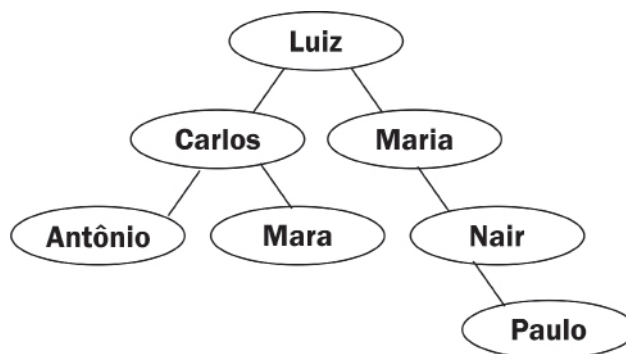




2. Refaça a árvore com os mesmos números, mas em ordem inversa (comece inserindo o 68 na raiz)



3. Insira a sequência de nomes numa árvore binária de busca: Luís, Carlos, Maria, Mara, Nair, Antônio, Paulo.



Atividades aula 8

1. Quantos ancestrais tem um nó no nível n numa árvore binária?

O nível ou profundidade de um nó " n " é o número de nós existente entre a raiz até o nó " n ". Como cada nó em uma árvore binária possui apenas um ancestral direto, temos que a quantidade de ancestrais é o valor do nível do nó, isto é n .



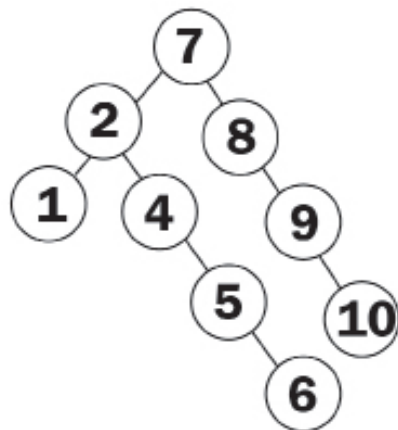
2. Escreva uma função que calcule o número de nós de uma árvore binária.

Devemos inicialmente declarar um atributo “elementos” do tipo “int” para a classe Arvore.

Criamos o método ChamaContaElementos que iniciará o valor de elementos = 0 e chamará o método ContaElementos. Este fará chamadas recursivas percorrendo em ordem a árvore e, cada vez que passa por um nó, o atributo elementos é incrementado em 1.

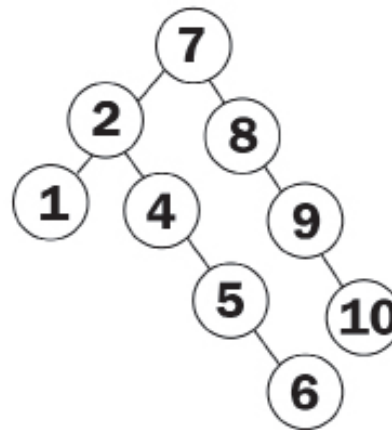
```
public int ChamaContaElementos() {  
    elementos = 0;  
    ContaElementos(raiz);  
    return elementos;  
}  
  
private void ContaElementos(NoArv atual) {  
    if (atual == null) return;  
    ContaElementos(atual.getEsq());  
    elementos++;  
    ContaElementos(atual.getDir());  
}
```

3. Dada a árvore binária abaixo, insira os seguintes elementos: 5 e 6

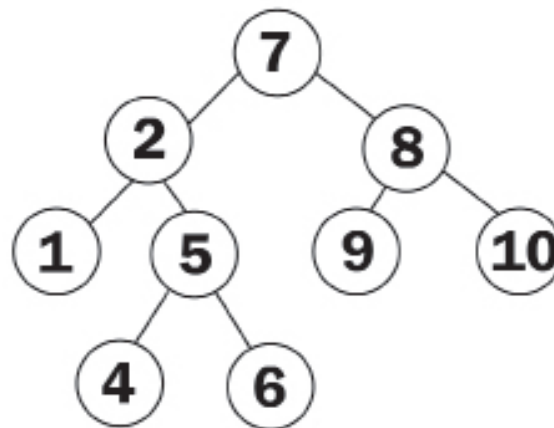




4. Realize o balanceamento da árvore resultante do exercício 3.



Temos duas subárvores desbalanceadas: nó 4 e nó 8, ambas com $FB=+2$ e, como seus filhos possuem $FB=+1$, faremos rotação simples para a esquerda, isto é, os filhos tomarão lugar dos pais.





Referências

GOODRICH, Michael T. e TAMASSIA, Roberto. **Estruturas de dados e Algoritmos em Java**. 4 ed. Porto Alegre: Bookman, 2007.

LAFORE, R. **Estruturas de Dado e Algoritmos em Java**. Ciência Moderna, 2005.

PORTAL TOL. **Artigo: Informações sobre Lista ligada**. Disponível em <<http://artigos.tol.pro.br/portal/linguagem-pt/Lista%20ligada>> Acesso em: 06 dez. 2013

PUGA, Sandra; RISSETTI, Gerson. **Lógica de programação e estruturas de dados: com aplicações em Java**. 2 ed. São Paulo: Pearson Prentice Hall, 2008.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de Dados e Seus Algoritmos**. 3 ed. LTC, 2010.

ZIVIANE, Nívio. **Projeto de Algoritmos Com Implementações em Pascal e C**. 3 ed. São Paulo: Cengage Learning. 2010.

Bibliografia básica

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. trad. 3 ed. Rio de Janeiro: Campus, 2012.

WALDEMAR, Celes; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a Estruturas de dados: com técnicas de programação em C**. Rio de Janeiro: Campus/Elsevier, 2004.

ZIVIANI, Nívio. **Projeto de algoritmos com implementações em Java e C++**. Ed.Cengage Learning. 2006.



Currículo dos Professores-autores



José Marcio Benite Ramos

Bacharel em Ciências da Computação pela Universidade Federal de São Carlos - UFSCar.

Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina - UFSC.

Professor Titular responsável pelas disciplinas de Estrutura de Dados, Programação Web, Sistemas Comerciais da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor Titular responsável pelas disciplinas de Programação Web da Faculdade de São Mateus – FSM-RO.

Analista de Sistemas e diretor da divisão Esc. de Porto Velho do Serviço Federal de Processamento de Dados SERPRO.



Liluyoud Cury de Lacerda

Bacharel em Ciências da Computação pela Universidade Federal de São Carlos - UFSCar.

Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina - UFSC.

Coordenador do Curso de Sistemas de Informação da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Coordenador do Curso de Sistemas para Internet da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Coordenador do Curso de Pós-Graduação em Desenvolvimento Web da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor Titular responsável pelas disciplinas de Algoritmos, Padrões de Pro-



jeto e Inteligência Artificial da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor Titular responsável pelas disciplinas de Programação Orientada a Objetos da Faculdade de São Mateus – FATESM-RO.

Analista Programador do Ministério Público do Estado de Rondônia - MP/RO.



Sara Luize Oliveira Duarte

Mestre em Gestão e Desenvolvimento Regional pela Universidade de Taubaté (UNITAU).

Pós-Graduação em Desenvolvimento Web e Metodologia do Ensino Superior. Graduada em Processamento de Dados, ambos pela FATEC-RO.

Professora das Faculdades FATESM e FATEC. Professora Titular responsável pela disciplina Informática Básica para Trabalhos Acadêmicos, ofertada na modalidade semipresencial na Faculdade de Tecnologia São Mateus – FATESM.

Supervisora de Tecnologia do Laboratório de Educação a Distância - LED e Supervisora das Salas Virtuais da modalidade Semipresencial da FATESM.

Professora Conteudista de alguns Guias de Estudos, como Informática Básica para Trabalhos Acadêmicos, Metodologia da Pesquisa Científica, Como estudar na EAD, Informática Básica e outros.

