**• Árvores AVL**

* Elaborar a implementação do algoritmo (em Java) pedido no “Desafio 1” do item “Aula 05”, slide 33, disponível na aba “Conteúdo” do curso “Estruturas de Dados II ”, no BlackBoard. São os mesmos slides usados em aula.
* A implementação do algoritmo deve estar em JAVA e deve ser criado em um documento do tipo doc, docx ou txt.

**A implementação do Desafio 1 :**

*Crie a Árvore e Insira os seguintes elementos em uma árvore AVL.*

*• Arvore Inicial: 20, 15, 18, 29, 26, 27*

*• Busque o Nó 18*

*• Insira um a uma: 2, 7, 19, 10*

*2. Em seguida, exclua a raiz principal.*

*• Lembrar que o nó eliminado será substituído pelo maior valor da subárvore esquerda*

**Conteúdo (Classe principal - ARVORE):**

*public class Arvore {*

*public static void main(String[] args) {*

*AVL avl = new AVL();*

*avl.insereAVL(20);*

*avl.insereAVL(15);*

*avl.insereAVL(18);*

*avl.insereAVL(29);*

*avl.insereAVL(26);*

*avl.insereAVL(27);*

*System.out.println("\n Dados Inseridos de Inicio: " );*

*emNivel\_Queue(avl.getRaiz());*

*System.out.println("\n Busque: " + avl.searchAVL(18).*

*toString());*

*avl.insereAVL(2);*

*System.out.println("\n Inserimos o elemento 2 na arvore: " );*

*emNivel\_Queue(avl.getRaiz());*

*avl.insereAVL(7);*

*System.out.println("\n Inserimos o elemento 7 na arvore: " );*

*emNivel\_Queue(avl.getRaiz());*

*avl.insereAVL(19);*

*System.out.println("\n Inserimos o elemento 19 na arvore: " );*

*emNivel\_Queue(avl.getRaiz());*

*avl.insereAVL(10);*

*System.out.println("\n Inserimos o elemento 10 na arvore: " );*

*emNivel\_Queue(avl.getRaiz());*

*System.out.println("\n Inserimos novos elementos na Árvore: " );*

*emNivel\_Queue(avl.getRaiz());*

*avl.removeAVL(18);*

*System.out.println("\n Movemos a Raiz da arvore: " );*

*emNivel\_Queue(avl.getRaiz());*

*}*

*private static void emNivel\_Queue(NoAVL raiz) {*

*NoAVL nodeAux;*

*Queue\_Dinamic queue;*

*queue = new Queue\_Dinamic();*

*queue.enqueue(raiz);*

*while(!queue.isEmpty()){*

*nodeAux = (NoAVL)queue.dequeue();*

*if(nodeAux.getEsquerda()!=null){*

*queue.enqueue(nodeAux.getEsquerda());*

*}*

*if(nodeAux.getDireita()!=null){*

*queue.enqueue(nodeAux.getDireita());*

*}*

*System.out.println(nodeAux.getDado() + " ");*

*}*

*}*

*}*

**2. Confira também as demais classes dentro do projeto:**

**Classe AVL:**

*import java.util.LinkedList;*

*public class AVL {*

*private NoAVL raiz; //Raiz da árvore*

*private boolean flagInsercao; //Verifica se já foi feita a inserção*

*private boolean flagRemove; //Verifica se já foi feita a remoção*

*public AVL(Object dado, NoAVL pai, NoAVL esq, NoAVL dir) {*

*raiz = new NoAVL(dado, pai, esq, dir);*

*}*

*public AVL(Object dado) {*

*this(dado, null, null, null);*

*}*

*public AVL() {*

*raiz = null;*

*}*

*public NoAVL getRaiz() {*

*return raiz;*

*}*

*public void setRaiz(NoAVL \_raiz) {*

*raiz = \_raiz;*

*}*

*public boolean isEmpty() {*

*return (raiz == null);*

*}*

*private int compara(Object ob1, Object ob2) {*

*return ((Comparable)ob1).compareTo(ob2);*

*}*

*private NoAVL searchNoAVL(NoAVL raiz, Object e) {*

*//Se a raiz estiver nula, o elemento não existe*

*if (raiz == null) {*

*return null;*

*} else //Elemento encontrado na raiz*

*if (compara(e, raiz.getDado()) == 0) {*

*return raiz;*

*} else //Continue procurando recursivamente*

*if (compara(e, raiz.getDado()) < 0) {*

*return searchNoAVL(raiz.getEsquerda(), e);*

*} else {*

*return searchNoAVL(raiz.getDireita(), e);*

*}*

*}*

*public NoAVL searchAVL(Object e) {*

*return searchNoAVL(raiz, e);*

*}*

*//Rotação Simples para a Direita*

*private NoAVL rotacaoSD(NoAVL A) {*

*NoAVL B = A.getEsquerda();*

*//Se não for a raiz, A tem um pai:*

*if (A.getPai() != null) {*

*if (A.getPai().getEsquerda() == A) //Se A for o filho esquerdo, o pai assume como filho esquerdo o B*

*{*

*A.getPai().setEsquerda(B);*

*} else //Senão o pai assume como filho direito o B*

*{*

*A.getPai().setDireita(B);*

*}*

*}*

*//O pai de B agora é o pai de A*

*B.setPai(A.getPai());*

*//Como o B subiu, pode ter deixado um órfão (direito) que quem assume é o A*

*A.setEsquerda(B.getDireita());*

*//Se A assumiu o filho do B, então setar o pai dele sendo o A*

*if (A.getEsquerda() != null) {*

*A.getEsquerda().setPai(A);*

*}*

*//B passa a ser o pai de A e A será filho de B*

*B.setDireita(A);*

*A.setPai(B);*

*return B;*

*}*

*//Rotação Simples para a Esquerda*

*private NoAVL rotacaoSE(NoAVL A) {*

*NoAVL B = A.getDireita();*

*//Se não for a raiz, tem um pai*

*if (A.getPai() != null) //Se A for o filho esquerdo, o pai assume como filho esquerdo o B*

*{*

*if (A.getPai().getDireita() == A) {*

*A.getPai().setDireita(B);*

*} //Senão o pai assume como filho direito o B*

*else {*

*A.getPai().setEsquerda(B);*

*}*

*}*

*//O pai de B agora é o pai de A*

*B.setPai(A.getPai());*

*//Como o B sumiu, pode ter deixado um órfão que quem assume é o A*

*A.setDireita(B.getEsquerda());*

*//Se assumiu o filho, setar o pai dele sendo o A*

*if (A.getDireita() != null) {*

*A.getDireita().setPai(A);*

*}*

*//B passa a ser pai de A e A filho de B*

*B.setEsquerda(A);*

*A.setPai(B);*

*return B;*

*}*

*//Rotação dupla para a direita*

*private NoAVL rotacaoDD(NoAVL A) {*

*rotacaoSE(A.getEsquerda());*

*return (rotacaoSD(A));*

*}*

*//Rotação dupla para a esquerda*

*private NoAVL rotacaoDE(NoAVL A) {*

*rotacaoSD(A.getDireita());*

*return (rotacaoSE(A));*

*}*

*//Insere um item na árvore a partir da raiz (método público)*

*public void insereAVL(Object k) {*

*flagInsercao = false;*

*setRaiz(insereNoAVL(raiz, k));*

*}*

*//Método que faz a inserção*

*private NoAVL insereNoAVL(NoAVL raiz, Object x) {*

*if (raiz != null) { //Se o nó não for nulo*

*if (compara(x, raiz.getDado()) < 0) { //Se x for menor que o nó atual, insere recursivamente à esquerda*

*raiz.setEsquerda(insereNoAVL(raiz.getEsquerda(), x));*

*raiz.getEsquerda().setPai(raiz);*

*if (flagInsercao) { //Se já inseriu*

*switch (raiz.getFb()) {*

*case 1: //Caso ele tinha 1 filho direito, o filho esquerdo balanceou*

*raiz.setFb(0);*

*flagInsercao = false;*

*break;*

*case 0: //Caso não tinha filhos, agora tem só o esquerdo*

*raiz.setFb(-1);*

*break;*

*case -1: //Caso já tinha um filho esquerdo, tem que rotacionar*

*//Se o filho esquerdo só tinha um filho esquerdo, então rotação simples para a direita*

*if (raiz.getEsquerda().getFb() == -1) {*

*raiz = rotacaoSD(raiz);*

*raiz.setFb(0);*

*raiz.getDireita().setFb(0);*

*}*

*else { //Caso contrário a rotação é dupla para a direita*

*raiz = rotacaoDD(raiz); //rotacaoDD retorna a nova raiz*

*raiz.getDireita().setFb(0);*

*raiz.getEsquerda().setFb(0);*

*raiz.setFb(0);*

*}*

*flagInsercao = false;*

*break;*

*}*

*}*

*} //fim da inserção recursiva à esquerda*

*else { //Insere Recursivamente à direita*

*raiz.setDireita(insereNoAVL(raiz.getDireita(), x));*

*raiz.getDireita().setPai(raiz);*

*if (flagInsercao) { //Se já inseriu*

*switch (raiz.getFb()) {*

*case 0: //Se não tinha filhos, agora tem só o direito*

*raiz.setFb(1);*

*break;*

*case -1: //Se só tinha um esquerdo, equilibrou*

*raiz.setFb(0);*

*flagInsercao = false;*

*break;*

*case 1: //Se jã tinha filhos direito, tem que rotacionar*

*//Se o filho direito tiver apenas um filho direito, então é rotação simples para a esquerda*

*if (raiz.getDireita().getFb() == 1) {*

*raiz = rotacaoSE(raiz);*

*raiz.setFb(0);*

*raiz.getEsquerda().setFb(0);*

*}*

*else { //Caso contrário, rotação dupla para a esquerda*

*raiz = rotacaoDE(raiz); //rotacaoDE retorna a nova raiz*

*raiz.getDireita().setFb(0);*

*raiz.getEsquerda().setFb(0);*

*raiz.setFb(0);*

*}*

*flagInsercao = false;*

*break;*

*}*

*}*

*} //fim da inserção recursiva à direita*

*} //Se chegar depois da folha (raiz==null) criar nó:*

*else { // este é o else do if (raiz != null)*

*//Quando chegar na folha, inserir novo NoAVL e trocar a flagInsercao*

*//para passar pelo processo de rotação*

*raiz = new NoAVL(x);*

*flagInsercao = true;*

*}*

*return raiz;*

*}*

*//Remove uma Object k da árvore AVl (método público)*

*public boolean removeAVL(Object k) {*

*flagRemove = false;*

*if (isEmpty()) {*

*System.out.println("Erro ao remover, árvore AVL está vazia!");*

*return false;*

*} else if (searchAVL(k) == null) {*

*System.out.println("Erro ao remover, elemento não existe na árvore!");*

*return false;*

*} else {*

*raiz = removeNoAVL(raiz, k);*

*return true;*

*}*

*}*

*//Método privado recursivo*

*private NoAVL removeNoAVL(NoAVL raiz, Object x) {*

*//Se o elemento for menor que a raiz, chamar recursivamente para o lado esquerdo*

*if (compara(x, raiz.getDado()) < 0) {*

*raiz.setEsquerda(removeNoAVL(raiz.getEsquerda(), x));*

*//Se já removeu, relabancear*

*if (flagRemove) {*

*raiz = balanceamentoEsquerdo(raiz);*

*}*

*} //Se o elemento for maior que a raiz, chamar recursivamente para o lado direito*

*else if (compara(x, raiz.getDado()) > 0) {*

*raiz.setDireita(removeNoAVL(raiz.getDireita(), x));*

*//Se já removeu, relabancear*

*if (flagRemove) {*

*raiz = balanceamentoDireito(raiz);*

*}*

*} //Se o elemento a remover está na raiz (encontrou o nó)*

*else {*

*//Se não tiver um filho direito*

*if (raiz.getDireita() == null) {*

*//Se tiver o filho esquerdo (assume o lugar do pai)*

*if (raiz.getEsquerda() != null) {*

*raiz.getEsquerda().setPai(raiz.getPai());*

*}*

*//Filho esquerdo sobe*

*raiz = raiz.getEsquerda();*

*flagRemove = true;*

*} //Se não tiver um filho esquerdo*

*else if (raiz.getEsquerda() == null) {*

*//Se tiver o filho direito (assume o lugar do pai)*

*if (raiz.getDireita() != null) {*

*raiz.getDireita().setPai(raiz.getPai());*

*}*

*//Filho direito sobe*

*raiz = raiz.getDireita();*

*flagRemove = true;*

*} //Tem os dois filhos, calcular o GetMax*

*else {*

*raiz.setEsquerda(buscaRemove(raiz.getEsquerda(), raiz));*

*//Se necessário efetuar balanceamento esquerdo, pois a remoção foi à esquerda*

*if (flagRemove) {*

*raiz = balanceamentoEsquerdo(raiz);*

*}*

*}*

*}*

*return raiz;*

*}*

*//Reorganiza os fatores de balanceamento na remoção*

*private NoAVL balanceamentoEsquerdo(NoAVL no) {*

*switch (no.getFb()) {*

*case -1: //Se tinha um nó esquerdo, removeu e balanceou*

*no.setFb(0);*

*break;*

*case 0: //Se não tinha filhos, ficou com um à direita*

*no.setFb(1);*

*break;*

*case 1: //Se tinha 1 nível a mais à direita, Balanceou*

*NoAVL subDir = no.getDireita();*

*int fb = subDir.getFb();*

*if (fb >= 0) {*

*subDir = rotacaoSE(no);*

*if (fb == 0) {*

*no.setFb(1);*

*subDir.setFb(-1);*

*flagRemove = false;*

*} else {*

*no.setFb(0);*

*subDir.setFb(0);*

*}*

*no = subDir;*

*} else {*

*no = rotacaoDD(no);*

*if (no.getFb() == 0) {*

*no.getDireita().setFb(0);*

*no.getEsquerda().setFb(0);*

*} else if (no.getFb() == 1) {*

*no.setFb(0);*

*no.getDireita().setFb(0);*

*no.getEsquerda().setFb(-1);*

*} else {*

*no.setFb(0);*

*no.getDireita().setFb(1);*

*no.getEsquerda().setFb(0);*

*}*

*}*

*}*

*return no;*

*}*

*//Reorganiza os fatores de balanceamento na remoção*

*private NoAVL balanceamentoDireito(NoAVL no) {*

*switch (no.getFb()) {*

*case 1: //Se tinha um nó direito, removeu e balanceou*

*no.setFb(0);*

*break;*

*case 0: //Se não tinha filhos, ficou com um à esquerda*

*no.setFb(-1);*

*flagRemove = false;*

*break;*

*case -1: //Se tinha 1 nível a mais à direita, balanceou*

*NoAVL subEsq = no.getEsquerda();*

*int fb = subEsq.getFb();*

*if (fb <= 0) {*

*subEsq = rotacaoSD(no);*

*if (fb == 0) {*

*no.setFb(-1);*

*subEsq.setFb(1);*

*flagRemove = false;*

*} else {*

*no.setFb(0);*

*subEsq.setFb(0);*

*}*

*no = subEsq;*

*} else {*

*no = rotacaoDE(no);*

*if (no.getFb() == 0) {*

*no.getDireita().setFb(0);*

*no.getEsquerda().setFb(0);*

*} else if (no.getFb() == -1) {*

*no.setFb(0);*

*no.getDireita().setFb(1);*

*no.getEsquerda().setFb(0);*

*} else {*

*no.setFb(0);*

*no.getDireita().setFb(0);*

*no.getEsquerda().setFb(-1);*

*}*

*}*

*}*

*return no;*

*}*

*//Busca o maior valor da subárvore esquerda para substituir o nó excluído*

*private NoAVL buscaRemove(NoAVL raiz, NoAVL noChave) {*

*NoAVL noRemovido;*

*if (raiz.getDireita() != null) {*

*raiz.setDireita(buscaRemove(raiz.getDireita(), noChave));*

*if (flagRemove) {*

*raiz = balanceamentoDireito(raiz);*

*}*

*} else {*

*//Altera o valor da chave*

*noChave.setDado(raiz.getDado());*

*noRemovido = raiz;*

*//Se nó direito com maior valor tem subárvore esquerda deve ser removido*

*raiz = raiz.getEsquerda();*

*if (raiz != null) {*

*raiz.setPai(noRemovido.getPai());*

*}*

*flagRemove = true;*

*noRemovido = null;*

*}*

*return raiz;*

*}*

*//Método público que retorna a String*

*public String emOrdemString() {*

*return emOrdemString(raiz);*

*}*

*//Atravessamento em ordem*

*private String emOrdemString(NoAVL raiz) {*

*String resp = "";*

*if (raiz != null) {*

*resp += emOrdemString(raiz.getEsquerda());*

*resp += raiz.getDado() + ", ";*

*resp += emOrdemString(raiz.getDireita());*

*}*

*return resp;*

*}*

*public String preOrdemString() {*

*return preOrdemString(raiz);*

*}*

*//Atravessamento em ordem*

*private String preOrdemString(NoAVL raiz) {*

*String resp = "";*

*if (raiz != null) {*

*resp += raiz.getDado() + ", ";*

*resp += preOrdemString(raiz.getEsquerda());*

*resp += preOrdemString(raiz.getDireita());*

*}*

*return resp;*

*}*

*//Método público*

*public void posOrdem() {*

*posOrdem(raiz);*

*}*

*//Atravessamento em pós-ordem*

*private void posOrdem(NoAVL raiz) {*

*if (raiz != null) {*

*posOrdem(raiz.getEsquerda());*

*posOrdem(raiz.getDireita());*

*System.out.print(raiz.getDado() + ", ");*

*}*

*}*

*//Método público*

*public String posOrdemString() {*

*return posOrdemString(raiz);*

*}*

*//Atravessamento em ordem*

*private String posOrdemString(NoAVL raiz) {*

*String resp = "";*

*if (raiz != null) {*

*resp += posOrdemString(raiz.getEsquerda());*

*resp += posOrdemString(raiz.getDireita());*

*resp += raiz.getDado() + ", ";*

*}*

*return resp;*

*}*

*//Atravessamento em nível*

*public void emNivel() {*

*NoAVL noAux;*

*LinkedList f;*

*if (!isEmpty()) {*

*f = new LinkedList();*

*f.addLast(raiz);*

*while (!f.isEmpty()) {*

*noAux = (NoAVL) f.removeFirst();*

*if (noAux.getEsquerda() != null) {*

*f.addLast(noAux.getEsquerda());*

*}*

*if (noAux.getDireita() != null) {*

*f.addLast(noAux.getDireita());*

*}*

*System.out.print(noAux.getDado() + ", ");*

*}*

*}*

*}*

*public String emNivelString() {*

*NoAVL noAux;*

*LinkedList f;*

*String resp = "";*

*if (!isEmpty()) {*

*f = new LinkedList();*

*f.addLast(raiz);*

*while (!f.isEmpty()) {*

*noAux = (NoAVL) f.removeFirst();*

*if (noAux.getEsquerda() != null) {*

*f.addLast(noAux.getEsquerda());*

*}*

*if (noAux.getDireita() != null) {*

*f.addLast(noAux.getDireita());*

*}*

*resp += noAux.getDado() + ", ";*

*}*

*}*

*return resp;*

*}*

*public static void emNivel\_Queue(NoAVL raiz){*

*NoAVL nodeAux;*

*Queue\_Dinamic queue;*

*queue = new Queue\_Dinamic();*

*queue.enqueue(raiz);*

*while(!queue.isEmpty()){*

*nodeAux = (NoAVL)queue.dequeue();*

*if(nodeAux.getEsquerda()!=null){*

*queue.enqueue(nodeAux.getEsquerda());*

*}*

*if(nodeAux.getDireita()!=null){*

*queue.enqueue(nodeAux.getDireita());*

*}*

*System.out.println(nodeAux.getDado() + " ");*

*}*

*}*

*boolean removeAVL() {*

*throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.*

*}*

*}*

**Classe NoAVL:**

*public class NoAVL {*

*private Object dado; //Dado do nó*

*private NoAVL pai;//Pai do nó*

*private NoAVL esquerda; //Filho Esquerdo*

*private NoAVL direita; //Filho Direito*

*private int fb; //Fator de Balanceamento*

*public NoAVL(Object x, NoAVL p, NoAVL e, NoAVL d) {*

*dado = x;*

*pai = p;*

*esquerda = e;*

*direita = d;*

*fb = 0;*

*}*

*public NoAVL() {*

*this("",null,null,null);*

*}*

*public NoAVL(Object \_dado) {*

*this(\_dado,null,null,null);*

*}*

*public Object getDado() {*

*return dado;*

*}*

*public void setDado(Object \_dado) {*

*dado = \_dado;*

*}*

*public NoAVL getPai() {*

*return pai;*

*}*

*public void setPai(NoAVL \_pai) {*

*pai = \_pai;*

*}*

*public NoAVL getEsquerda() {*

*return esquerda;*

*}*

*public void setEsquerda(NoAVL \_esq) {*

*esquerda = \_esq;*

*}*

*public NoAVL getDireita() {*

*return direita;*

*}*

*public void setDireita(NoAVL \_dir) {*

*direita = \_dir;*

*}*

*public void setFb(int \_fb) {*

*fb = \_fb;*

*}*

*public int getFb() {*

*return fb;*

*}*

*}*

**Classe Node:**

*public class Node {*

*private Object valor;*

*private Node previa, proximo;*

*public Object getValor() {*

*return valor;*

*}*

*public void setValor(Object valor) {*

*this.valor = valor;*

*}*

*public Node getProximo() {*

*return proximo;*

*}*

*public void setNext(Object o) {*

*this.proximo = proximo;*

*}*

*public <E> void getValue(E novo) {*

*}*

*}*

**Classe Queue\_Dinamic:**

*public class Queue\_Dinamic implements TAD\_Queue {*

*private Node head = null;*

*private Node tail = null;*

*public Queue\_Dinamic() {*

*head = null;*

*tail = null;*

*}*

**Classe TAD\_Queue:**

*public interface TAD\_Queue {*

*public boolean isEmpty();*

*public boolean isFull();*

*public Object enqueue(Object x);*

*public Object dequeue();*

*public Object peek();*

*@Override*

*public String toString();*

*}*