**ANDRÉ DELFINO, FERNANDO DIAS, GIULIA CARVALHO SILVA, LUCAS BOTTEGA**

**PARTE 1**

**Main.java**

|  |
| --- |
| package parte1; |
|  |  |
|  | public class Main { |
|  | public static void main(String[] args) { |
|  | ArvoreBin arvore = new ArvoreBin(); |
|  |  |
|  | arvore.insereElemento(5); |
|  | arvore.insereElemento(20); |
|  | arvore.insereElemento(10); |
|  | arvore.insereElemento(40); |
|  | arvore.insereElemento(30); |
|  | arvore.insereElemento(50); |
|  |  |
|  | //System.out.println(arvore.existeElemento(20)); |
|  | arvore.imprimeInOrdem(arvore.raiz); |
|  | arvore.removeElemento(arvore.raiz, 30); |
|  | arvore.imprimeInOrdem(arvore.raiz); |
|  |  |
|  | //System.out.println(arvore.altura(arvore.raiz)); |
|  | } |
|  | } |

**ArvoreBin.java**

package parte1;

public class ArvoreBin {

public NoAVL raiz;

public ArvoreBin(){

this.raiz = null;

}

public void insereElemento(int valor) {

NoAVL novo = new NoAVL(valor);

// Se a raiz for nula, o novo elemento vira a raiz

if (this.raiz == null) {

this.raiz = novo;

} else {

NoAVL p = this.raiz;

NoAVL q = null;

// Enquanto p for diferente de nulo, inserir na direita ou na esquerda

while (p != null) {

q = p;

if (valor < p.info) {

p = p.esquerda;

} else {

p = p.direita;

}

}

if (valor < q.info) {

q.esquerda = novo;

} else {

q.direita = novo;

}

}

}

public NoAVL removeElemento(NoAVL raiz, int elemento){

// Se for nula e fim da recursiva

if(raiz == null){

return raiz;

}

// Seguir pelo lado esquerdo da subarvore até o elemento ser menor que o valor do nó

else if(raiz.info > elemento){

raiz.esquerda = removeElemento(raiz.esquerda, elemento);

}

// Seguir pelo lado esquerdo da subarvore até o elemento ser maior que o valor do nó

else if(raiz.info < elemento){

raiz.direita = removeElemento(raiz.direita, elemento);

}

else{

/\*Três casos a serem lidados: Elemento com nenhuma sub-arvore,

apenas uma sub-arvore ou duas sub-arvores\*/

// Lidando com a situação caso o elemento tenha uma ou nenhuma sub-arvore

if(raiz.esquerda == null){

return raiz.direita;

}

else if(raiz.direita == null){

return raiz.esquerda;

}

/\* Lidando com a situação caso o elemento tenha duas sub-arvores

Encontrar o mínimo elemento seguindo a sub-arvore da direita e aplicar

o método removeElemento() novamente \*/

raiz.info = encontraMinimo(raiz.direita);

raiz.direita= removeElemento(raiz.direita, raiz.info);

}

return raiz;

}

private int encontraMinimo(NoAVL no){

// Assumindo o minimo como o valor informado

int minimo = no.info;

while(no.esquerda != null){

minimo = no.esquerda.info;

no = no.esquerda;

}

return minimo;

}

public NoAVL existeElemento(int elemento) {

NoAVL p = this.raiz;

while (p != null && p.info != elemento) {

if (elemento < p.info) {

p = p.esquerda;

} else {

p = p.direita;

}

}

return p;

}

public int altura(NoAVL raiz) {

if (raiz == null) {

return 0;

} else {

int alturaEsquerda = altura(raiz.esquerda);

int alturaDireita = altura(raiz.direita);

return Math.max(alturaDireita, alturaEsquerda) + 1;

}

}

public void imprimeInOrdem(NoAVL raiz) {

if (raiz != null) {

imprimeInOrdem(raiz.esquerda);

System.out.println(raiz.info);

imprimeInOrdem(raiz.direita);

}

}

public void imprimePreOrdem(NoAVL raiz){

if (raiz != null){

System.out.println(raiz.info);

imprimePreOrdem(raiz.esquerda);

imprimePreOrdem(raiz.direita);

}

}

public void imprimePosOrdem(NoAVL raiz){

if (raiz != null){

imprimePosOrdem(raiz.esquerda);

imprimePosOrdem(raiz.direita);

System.out.println(raiz.info);

}

}

public int balanceamento(NoAVL raiz){

return altura(raiz.direita) - altura(raiz.esquerda);

}

public NoAVL pai(NoAVL no){

NoAVL noAtual = this.raiz;

if (no == this.raiz){

return raiz;

}

while(noAtual != null){

if (no == noAtual.esquerda || no == noAtual.direita){

break;

}

else if (no.info < noAtual.info){

noAtual = noAtual.esquerda;

}

else{

noAtual = noAtual.direita;

}

}

return noAtual;

}

private void rotacaoEsquerda(NoAVL no){

NoAVL raiz = pai(no);

NoAVL p = no.direita;

if (no == this.raiz){

this.raiz = p;

}

else if (raiz.info <= p.info){

raiz.direita = p;

}

else{

raiz.esquerda = p;

}

no.direita = p.esquerda;

p.esquerda = no;

}

private void rotacaoDireita(NoAVL no){

NoAVL raiz = pai(no);

NoAVL p = no.esquerda;

if (no == this.raiz){

this.raiz = p;

}

else if (raiz.info <= p.info){

raiz.direita = p;

}

else{

raiz.esquerda = p;

}

no.esquerda = p.direita;

p.direita = no;

}

}

**NoAVL.java**

package parte1;

public class NoAVL {

public NoAVL esquerda;

public NoAVL direita;

public int info;

public NoAVL(int valor) {

this.info = valor;

this.esquerda = null;

this.direita = null;

}

}

**PARTE 2**

**ArvoreBin.java**

package parte2;

public class ArvoreBinPalavra {

public NoAVL raiz;

public ArvoreBinPalavra() { this.raiz = null; }

public void inserePalavra(String novaPalavra) {

novaPalavra = novaPalavra.toLowerCase();

if ( raiz == null ) {

// A árvore está vazia.

// A nova palavra se torna a raíz.

raiz = new NoAVL(novaPalavra);

return;

}

NoAVL atual; // Vai checar toda a árvore para encontrar um lugar para a nova palavra

atual = raiz; // Começando pela raíz

while (true) {

if ( novaPalavra.equals(atual.palavra)) {

// A palavra já está na árvore, apenas retorna

atual.contador += 1;

return;

}

if ( novaPalavra.compareTo(atual.palavra) < 0) {

// Se a nova palavra for "menor" que o atual,

// ela deve ser adicionada à sub-árvore da esquerda.

// Se o ponteiro da esquerda estiver nulo, adiciona lá.

// Caso contrário desce um nível para a esquerda

if (atual.esquerda == null) {

atual.esquerda = new NoAVL(novaPalavra);

return; // Adicionou na esquerda

} else {

atual = atual.esquerda;

}

}

else {

// Se a nova palavra for "maior" que o atual,

// ela deve ser adicionada à sub-árvore da direita.

// Se o ponteiro da direta estiver nulo, adiciona lá.

// Caso contrário desce um nível para a direita.

if ( atual.direita == null ) {

atual.direita = new NoAVL(novaPalavra);

return; // Adicionou na direita

} else {

atual = atual.direita;

}

}

} // Fim do while

} // Fim do método

public void imprimeInOrdem(NoAVL raiz) {

if (raiz != null) {

imprimeInOrdem(raiz.esquerda);

System.out.println(raiz.palavra);

imprimeInOrdem(raiz.direita);

}

}

} // Fim da classe

**ListaEncadeada.java**

package parte2;

public class ListaEncadeada {

public NoENC primeiro;

public NoENC ultimo;

public void addNo(int data) {

NoENC novoNo = new NoENC(data);

if (primeiro == null) {

primeiro = novoNo;

} else {

ultimo.proximo = novoNo;

}

ultimo = novoNo;

}

}

**Main.java**

package parte2;

import java.io.\*;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws FileNotFoundException {

final File pasta = new File("C:/Users/lucas/Desktop/teste"); // Caminho do arquivo

Reader reader = new Reader(pasta);

reader.insereArvore(); // Insere os elementos do arquivo na arvore

reader.imprimeArvore(); // Imprime a arvore em ordem alfabetica

reader.encontraPalavra(); // Solicita uma palavra e mostra a quantidade de vezes repetida

}

}

**NoAVL.java**

package parte2;

public class NoAVL {

public NoAVL esquerda; // Ponteiro pra esquerda

public NoAVL direita; // Ponteiro pra direita

public String palavra; // Palavra

public int contador;

public ListaEncadeada lista; // Lista encadeada com as repeticoes

// Construtor recebendo a palavra

public NoAVL(String palavra) {

this.palavra = palavra;

this.direita = null;

this.esquerda = null;

this.contador = 1;

this.lista = null;

}

}

**NoENC.java**

package parte2;

public class NoENC {

public int valor;

public NoENC proximo;

public NoENC(int valor) {

this.valor = valor;

this.proximo = null;

}

}

**Reader.java**

package parte2;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class Reader {

private ArvoreBinPalavra arvore = new ArvoreBinPalavra();

private File pasta;

public Reader(File pasta) {

this.pasta = pasta;

}

public void insereArvore() throws FileNotFoundException {

// Método recebe um diretório, lê as palavras

// e adiciona em uma árvore binária

for (final File arquivo : pasta.listFiles()) {

// Para cada arquivo dentro da pasta informada

Scanner input = new Scanner(arquivo);

String palavra;

while (input.hasNext()) {

palavra = input.next();

arvore.inserePalavra(palavra);

contaPalavra(palavra);

}

}

} // Fim do método lerDiretorio

public void contaPalavra(String palavra) throws FileNotFoundException {

NoAVL p = arvore.raiz;

while (p != null && !p.palavra.equals(palavra)) {

if (palavra.compareTo(p.palavra) < 0) {

p = p.esquerda;

} else {

p = p.direita;

}

}

int counter = 0;

for (final File arquivo : pasta.listFiles()) {

// Para cada arquivo dentro da pasta informada

Scanner input = new Scanner(arquivo);

while (input.hasNext()) {

if (input.next().equals(palavra)) {

counter += 1;

}

}

}

assert p != null;

p.lista = new ListaEncadeada();

p.lista.addNo(counter);

} // Fim do método

public void imprimeArvore() {

System.out.println("Árvore em ordem alfabética: ");

arvore.imprimeInOrdem(arvore.raiz);

}

public void encontraPalavra() {

Scanner sc = new Scanner(System.in);

System.out.print("Digite a palavra a ser encontrada: ");

String palavra = sc.nextLine();

NoAVL p = arvore.raiz;

while (p != null && !p.palavra.equals(palavra)) {

if (palavra.compareTo(p.palavra) < 0) {

p = p.esquerda;

} else {

p = p.direita;

}

}

System.out.println("Palavra: " + palavra);

System.out.println("Quantidade: " + p.lista.primeiro.valor);

}

}