public int getAltura(){

return getAltura(raiz, 0);

}

public int getAltura(No i, int altura){

if(i == null){

altura--;

} else {

int alturaEsq = getAltura(i.esq, altura + 1);

int alturaDir = getAltura(i.dir, altura + 1);

altura = (alturaEsq > alturaDir) ? alturaEsq : alturaDir;

}

return altura;

}

public class Principal2 {

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

ArvoreBinaria a = new ArvoreBinaria();

Random gerador = new Random();

gerador.setSeed(0);

for(int i = 1; i <= 100000; i++){

int valor;

do {

valor = Math.abs(gerador.nextInt());

} while (a.pesquisar(valor) == true);

a.inserir(valor);

if(i % 1000 == 0){

double log2 = (Math.log(i) / Math.log(2));

log2 \*= 1.39;

System.out.println("Número de nós = " + i + " --- log(i,2) = " + log2 + " --- h = " + a.getAltura());

}

}

}

}

public int soma(){

return soma(raiz);

}

public int soma(No i){

int resp = 0;

if(i != null){

resp = i.elemento + soma(i.esq) + soma(i.dir);

}

return resp;

}

public int quantidadePares(){

return quantidadePares(raiz);

}

public int quantidadePares(No i){

int resp = 0;

if(i != null){

resp = ((i.elemento % 2 == 0) ? 1 : 0) + quantidadePares(i.esq) + quantidadePares(i.dir);

}

return resp;

}

public static boolean igual (ArvoreBinaria a1, ArvoreBinaria a2){

return igual(a1.raiz, a2.raiz);

}

private static boolean igual (No i1, No i2){

boolean resp;

if(i1 != null && i2 != null){

resp = (i1.elemento == i2.elemento) && igual(i1.esq, i2.esq) && igual(i1.dir, i2.dir);

} else if(i1 == null && i2 == null){

resp = true;

} else {

resp = false;

}

return resp;

}

public boolean hasDiv11(){

return hasDiv11(raiz);

}

public boolean hasDiv11(No i){

boolean resp = false;

if(i != null){

resp = (i.elemento % 11 == 0) || hasDiv11(i.esq) || hasDiv11(i.dir);

}

return resp;

}

No toAB(Celula p1, CelulaDupla p2){

No resp = null;

p1 = p1.prox;

p2 = p2.prox;

while(p1 != null && p2 != null){

resp = inserir(p1.elemento, raiz);

resp = inserir(p2.elemento, raiz);

p1 = p1.prox;

p2 = p2.prox;

}

while (p1 != null) {

resp = inserir(p1.elemento, raiz);

p1 = p1.prox;

}

while (p2 != null) {

resp = inserir(p2.elemento, raiz);

p2 = p2.prox;

}

return resp;

}

// no = rotacionarEsq(no);

No rotacionarEsq (No no){

No noDir = no.dir;

No noDirEsq = noDir.esq;

noDir.esq = no;

no.dir = noDirEsq;

return noDir;

}

No rotacionarDir (No no){

No noEsq = no.esq;

No noEsqDir = noEsq.dir;

noEsq.dir = no;

no.esq = noEsqDir;

return noEsq;

}

No rotacionarDirEsq( No no){

no.dir = rotacionarDir(no.dir);

return rotacionarEsq(no);

}

No rotacionarEsqDir( No no){

no.esq = rotacionarEsq(no.esq);

return rotacionarDir(no);

}

//Balanciamento

void balancear(){

if(raiz.esq != null && raiz.dir != null){

System.out.println("Arvore Balanceada");

} else if (raiz.dir != null){

if(raiz.dir.dir != null){

raiz = rotacionarEsq(raiz);

}else {

raiz = rotacionarDirEsq(raiz);

}

} else{

if(raiz.esq.dir != null){

raiz = rotacionarEsqDir(raiz);

}else{

raiz = rotacionarDir(raiz);

}

}

}

As operações básicas (inserir, remover e pesquisar) em uma árvore binária demoram um tempo proporcional à sua altura. No caso de uma árvore binária balanceada com **n** nós, executamos tais operações com custo **θ(lg n)** no pior caso. Logo, o fato do custo em uma árvore binária balanceada ser **θ(lg n)** significa que executamos tais operações fazendo aproximadamente **lg n**comparações entre elementos da árvore. Lg é o logaritmo na base dois de **n**.

A principal vantagem das árvores binárias é sua eficiência nas operações de pesquisa, inserção e remoção que podem ser efetuadas com o custo de θ(lg n)no pior caso quando a árvore está balanceada. O processo de balanceamento acontece após as operações de inserção ou remoção de um elemento. O algoritmo de balanceamento verifica se existe algum nó desbalanceado no caminho entre a folha inserida/removida e a raiz. Um nó está desbalanceado quando a diferença entre a altura das suas subárvores à direita e esquerda é maior que uma unidade.

Árvore binária:

Estudar no micro-fundamentos a parte árvores binárias

Mas linguagens Java e c, temos apenas passagem de parâmetros por valor. Na verdade, nunca passamos uma variável e sim seu valor. Assim, um método chamado nunca modifica um valor passado por parâmetro em Java nem em c.

Quando a árvore estiver balanciada o custo de execução será teta de Lg(i)

Quando inserimos elementos aleatórios, nossa árvore terá uma altura de teta de Lg(i)

Assisti o. Micro-fundamentos de aeds sobre balanciamento em árvore.

Árvore balanceada

As rotações sempre acontecem no avó.

Nas rotações duplas, a segunda rotação é a mais importante e ela acontece no avó.

As rotações duplas, serão efetuadas quando avó, pai e filho estiverem formando um V(para direita) depois um V(para esquerda)

Faremos a dupla Dir Esq quando tivermos um V(para direta). Faremos a dupla Esq Dir quando tivermos um V(para esquerda).

Na rotação dupla, a primeira rotação faz com que a árvore(avó, pai e filho) fiquem totalmente desbalanceada(uma linha reta).

A primeira rotação acontece no pai.

A lista contém duas referências (ou ponteiros) chamados primeiro e último que apontam para a primeira e a última célula. A primeira célula da lista flexível é o nó cabeça, uma célula "vazia" cuja função é inicializar as referências primeiro e último

public boolean pesquisar(int x){

CelulaDupla i;

boolean resp = false;

for(i = primeiro; i.getProx() != null; i = i.getProx()){

if( x == i.getElemento()){

resp = true;

}

}

if(resp == true){

System.out.print("\nO número " + x + " foi encontrado!! ");

}else{

System.out.print("\nO número " + x + " não foi encontrado!! ");

}

return resp;

}

public void inserirOrdem(int x){

CelulaDupla i, tmp = new CelulaDupla(x);

for(i = primeiro; i.getProx() != null && i.getProx().getElemento() < x; i = i.getProx() );

tmp.setProx(i.getProx());

tmp.setAnt(i);

i.getProx().setAnt(tmp);

i.setProx(tmp);

tmp = null;

if( primeiro == ultimo){

ultimo = ultimo.getProx();

}

}

public void remover(int x) throws Exception {

raiz = remover(x, raiz);

}

private No remover(int x, No i) throws Exception {

if (i == null) {

throw new Exception("Erro ao remover!");

} else if (x < i.elemento) {

i.esq = remover(x, i.esq);

} else if (x > i.elemento) {

i.dir = remover(x, i.dir);

// Sem no a direita.

} else if (i.dir == null) {

i = i.esq;

// Sem no a esquerda.

} else if (i.esq == null) {

i = i.dir;

// No a esquerda e no a direita.

} else {

i.esq = maiorEsq(i, i.esq);

}

return i;

}

public void inserir(int x) throws Exception {

raiz = inserir(x, raiz);

}

private No inserir(int x, No i) throws Exception {

if (i == null) {

i = new No(x);

} else if (x < i.elemento) {

i.esq = inserir(x, i.esq);

} else if (x > i.elemento) {

i.dir = inserir(x, i.dir);

} else {

throw new Exception("Erro ao inserir!");

}

return i;

}