UNIVERSIDADE ESTADUAL DE PONTA GROSSA

SETOR DE CIÊNCIAS AGRÁRIAS E SUAS TÉCNOLOGIAS

DEPARTAMENTO DE INFORMÁTICA

MAYSA LOVATTO LOPES  
RENANN RODRIGUES DA SILVA

ALGORITIMOS: ESTRUTURA DE DADOS

PONTA GROSSA

2014

MAYSA LOVATTO LOPES

RENANN RODRIGUES DA SILVA

ALGORITIMOS: ESTRUTURA DE DADOS

Trabalho apresentada na disciplina de

Estrutura de dados como requisito

de avaliação parcial referente ao

3º Bimestre. 2º Ano de Informática.

Professor Márcio Augusto de Souza

PONTA GROSSA

2014

//Trabalho de Estrutura de dados Terceiro Bimestre Parte 2

//Academicos Maysa Lovatto Lopes RA: 13015123

// Renann R. da Silva RA: 13106523

#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define TAM 1000

int vetor[TAM];

float aux\_busca\_seq = 0,aux\_busca\_bin = 0,aux\_busca\_arvb = 0,aux\_busca\_arv\_avl = 0;

struct no \*raiz2=NULL, \*raiz;

struct no {

int dado;

struct no \*esq;

struct no \*dir;

int bal;

};

void cria\_vetor() {

int i;

for (i=0;i<=TAM-1;i++)

vetor[i]=rand()%TAM;//Cria vetor randômico

}

void imprimir(){

int i;

for(i=0; i<=TAM-1; i++){

printf("%d ", vetor[i]);

}

printf("\n\n");

}

// quicksort

int partition(int p, int r) {

int piv = vetor[p], i = (p - 1), j = (r + 1), aux = 0;

while(1) {

while(vetor[--j] > piv);

while(vetor[++i] < piv);

if(i < j) {

aux = vetor[i];

vetor[i] = vetor[j];

vetor[j] = aux;

}

else

return j;

}

}

void quicksort(int p, int r) {

int q=0;

if(p < r) {

q = partition(p, r);

quicksort(p, q);

quicksort((q+1), r);

}

}

//------------------------------------------------------------------------------

// inserção arvore AVL

void esquerda(struct no \*p) {

no \*q, \*hold;

q = p->dir;

hold = q->esq;

q->esq = p;

p->dir = hold;

}

void direita(struct no \*p) {

no \*q, \*hold;

q = p->esq;

hold = q->dir;

q->dir = p;

p->esq = hold;

}

struct no \*cria\_no(int valor){

no \*aux = new (struct no);

aux->dado=valor;

aux->dir=NULL;

aux->esq=NULL;

aux->bal=0;

return aux;

};

void insere\_bal(int chave){

no \*pp=NULL, \*p=raiz2, \*pajovem=NULL, \*ajovem=raiz2, \*q, \*filho;

int imbal;

if (p==NULL) { // Arvore vazia

raiz2 = cria\_no(chave); // Funcao para criacao de um novo no

return;

}

// Insere chave e descobre ancestral mais jovem a ser desbalanceado

while (p!=NULL) {

if (chave < p->dado)

q = p->esq;

else q = p->dir;

if (q!=NULL)

if (q->bal != 0) {

pajovem=p;

ajovem=q;

}

pp = p;

p = q;

}

q = cria\_no(chave);

if (chave<pp->dado)

pp->esq=q;

else pp->dir=q;

// Balanceamento de todos os nós entre ajovem e q devem ser ajustados

if (chave<ajovem->dado)

filho = ajovem->esq;

else filho = ajovem->dir;

p = filho;

while (p!=q) {

if (chave < p->dado) {

p->bal=1;

p=p->esq;

} else {

p->bal = -1;

p=p->dir;

}

}

if (chave<ajovem->dado)

imbal = 1;

else imbal = -1;

if (ajovem->bal==0) { //Não houve desbalanceamento

ajovem->bal=imbal;

return;

}

if (ajovem->bal!=imbal) { //Não houve desbalanceamento

ajovem->bal=0;

return;

}

if (filho->bal == imbal) {

p=filho;

if (imbal==1) //Faz rotação simples

direita(ajovem);

else esquerda(ajovem);

ajovem->bal=0;

filho->bal=0;

}

else {

if (imbal==1) { //Faz rotação dupla

p=filho->dir;

esquerda(filho);

ajovem->esq=p;

direita(ajovem);

} else {

p=filho->esq;

direita(filho);

ajovem->dir=p;

esquerda(ajovem);

}

if (p->bal==0) {

ajovem->bal=0;

filho->bal=0;

} else {

if (p->bal == imbal) {

ajovem->bal = - imbal;

filho->bal = 0;

} else {

ajovem->bal = 0;

filho->bal = imbal;

}

}

p->bal=0;

}

if (pajovem == NULL) // Ajusta ponteiro do pai

raiz2 = p;

else if (ajovem==pajovem->dir)

pajovem->dir = p;

else pajovem->esq = p;

return;

}

//------------------------------------------------------------------------------

// inserção arvore Binaria

void insere(int valor){

no \*aux;

no \*atual;

aux = new(struct no);

aux->esq = NULL;

aux->dir = NULL;

aux->dado = valor;

if(raiz == NULL){

raiz=aux;

return;

}

atual = raiz;

while(1){

if(valor<atual->dado){

if(atual->esq == NULL){

atual->esq = aux;

return;

}

else

atual=atual->esq;

}

else{

if(atual->dir == NULL){

atual->dir = aux;

return;

}

else

atual=atual->dir;

}

}

}

//------------------------------------------------------------------------------

// inserir na arvore binaria

void insere\_arvb(){

int i;

for(i=0; i<TAM; i++){

insere(vetor[i]);

}

}

//------------------------------------------------------------------------------

// insere arvore avl

void insere\_arvavl(){

int i;

for(i=0; i<TAM; i++){

insere\_bal(vetor[i]);

}

}

//------------------------------------------------------------------------------

// gerador de chave de busca

int gerador\_chave(){

int valor=0;

valor = rand()%TAM;

return valor;

}

//------------------------------------------------------------------------------

// busca sequencial = corrigir

int busca\_sequencial(int chave){

int i=0;

for(i=0;i<TAM && vetor[i]<=chave ;i++){

aux\_busca\_seq++;

if(vetor[i] == chave)

return 1;

}

return 0;

}

//------------------------------------------------------------------------------

// busca binaria

int busca\_binaria(int chave){

int esq=0, dir=(TAM-1), meio=0;

while (esq<=dir){

meio = (esq+dir)/2;

aux\_busca\_bin++;

if (chave == vetor[meio]){

return 1;

}

if (chave<vetor[meio]){

dir = meio-1;

}

else{

esq = meio+1;

}

}

return 0;

}

//------------------------------------------------------------------------------

// busca arvore binaria

int busca\_arv\_binaria(int chave){

struct no \*atual;

atual = raiz;

while(atual!=NULL){

aux\_busca\_arvb++;

if (atual->dado == chave){

return 1;

}

if (chave < atual->dado){

atual = atual->esq;

}

else{

atual = atual->dir;

}

}

return 0;

}

//------------------------------------------------------------------------------

//busca arvore avl

int busca\_arv\_avl(int chave){

struct no \*atual;

atual = raiz2;

while(atual!=NULL){

aux\_busca\_arv\_avl++;

if (atual->dado == chave){

return 1;

}

if (chave < atual->dado){

atual = atual->esq;

}

else{

atual = atual->dir;

}

}

return 0;

}

//------------------------------------------------------------------------------

// função que verifica se o vetor esta em ordem

int verifica\_ordem(){

int i;

for(i=0; i<TAM-1; i++){

if(vetor[i] > vetor[i+1]){

return 0;

}

}

return 1;

}

//------------------------------------------------------------------------------

main(){

int chave=0,i;

printf(" - Criando vetor...\n");

cria\_vetor();

printf(" - Inserindo em Arvore Binaria...\n");

insere\_arvb();

printf(" - Inserindo em Arvore AVL...\n");

insere\_arvavl();

printf(" - Ordenando vetor... ");

quicksort(0,TAM-1);

printf("%s\n", verifica\_ordem() ? "Vetor ordenado!\n" : "Vetor desordenado!\n");

for(i=0;i<10;i++){

chave = gerador\_chave()%TAM-i;

busca\_sequencial(chave);

busca\_binaria(chave);

busca\_arv\_binaria(chave);

busca\_arv\_avl(chave);

}

printf("\n- Tamanho do Vetor = %d/numero de buscas feitas = %d \n\n", TAM, i);

printf("- Busca Seqüencial - media de: %.1f comparações\n",aux\_busca\_seq/10);

printf("- Busca Binária - media de: %.1f comparações\n",aux\_busca\_bin/10);

printf("- Busca Arvore Binária - media de: %.1f comparações\n",aux\_busca\_arvb/10);

printf("- Busca Arvore AVL - media de: %.1f comparações\n",aux\_busca\_arv\_avl/10);

getch();

}