using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

namespace \_16189\_16191\_Projeto4ED

{

class ArvoreDeBusca<Tipo> : IComparable<NoArvore<Tipo>>

where Tipo : IComparable<Tipo>

{

public NoArvore<Tipo> raiz,

atual,

antecessor;

public Panel painelArvore;

public Panel OndeExibir

{

get { return painelArvore; }

set { painelArvore = value; }

}

public ArvoreDeBusca()

{

raiz = null;

atual = null;

antecessor = null;

}

public NoArvore<Tipo> Raiz

{

get { return raiz; }

}

public String InOrdem // propriedade que gera a string do percurso in-ordem da árvore

{

get { return FazInOrdem(raiz); }

}

public String PreOrdem // propriedade que gera a string do percurso pre-ordem da árvore

{

get { return FazPreOrdem(raiz); }

}

public String PosOrdem // propriedade que gera a string do percurso pos-ordem da árvore

{

get { return FazPosOrdem(raiz); }

}

private String FazInOrdem(NoArvore<Tipo> r)

{

if (r == null)

return ""; // retorna cadeia vazia

else

{

return FazInOrdem(r.esquerdo) +

" " + r.Info.ToString() + " " +

FazInOrdem(r.direito);

}

}

private String FazPreOrdem(NoArvore<Tipo> r)

{

if (r == null)

return ""; // retorna cadeia vazia

else

{

return

r.Info.ToString() +

" " + FazPreOrdem(r.esquerdo) + " " +

FazPreOrdem(r.direito);

}

}

private String FazPosOrdem(NoArvore<Tipo> r)

{

if (r == null)

return ""; // retorna cadeia vazia

else

{

return FazPosOrdem(r.esquerdo) + " " +

FazPosOrdem(r.direito) +

" " + r.Info.ToString();

}

}

public int CompareTo(NoArvore<Tipo> o)

{

return atual.Info.CompareTo(o.Info);

}

public void inserir(Tipo novosInfo)

{

bool achou = false, fim = false;

NoArvore<Tipo> novoNo = new NoArvore<Tipo>(novosInfo);

if (raiz == null) // árvore vazia

raiz = novoNo;

else // árvore não-vazia

{

antecessor = null;

atual = raiz;

while (!achou && !fim)

{

antecessor = atual;

if (novosInfo.CompareTo(atual.Info) < 0)

{

atual = atual.esquerdo;

if (atual == null)

{

antecessor.esquerdo = novoNo;

fim = true;

}

}

else

if (novosInfo.CompareTo(atual.Info) == 0)

achou = true; // pode-se disparar uma exceção neste caso

else

{

atual = atual.direito;

if (atual == null)

{

antecessor.direito = novoNo;

fim = true;

}

}

}

}

}

public bool ApagarNo(Tipo chaveARemover)

{

atual = raiz;

antecessor = null;

bool ehFilhoesquerdouerdo = true;

while (atual.Info.CompareTo(chaveARemover) != 0) // enqto não acha a chave a remover

{

antecessor = atual;

if (atual.Info.CompareTo(chaveARemover) > 0)

{

ehFilhoesquerdouerdo = true;

atual = atual.esquerdo;

}

else

{

ehFilhoesquerdouerdo = false;

atual = atual.direito;

}

if (atual == null) // neste caso, a chave a remover não existe e não pode

return false; // ser excluída, dai retornamos falso indicando isso

} // fim do while

// se fluxo de execução vem para este ponto, a chave a remover foi encontrada

// e o ponteiro atual indica o nó que contém essa chave

if ((atual.esquerdo == null) && (atual.direito == null)) // é folha, nó com 0 filhos

{

if (atual == raiz)

raiz = null; // exclui a raiz e a árvore fica vazia

else

if (ehFilhoesquerdouerdo) // se for filho esquerdouerdo, o antecessor deixará

antecessor.esquerdo = null; // de ter um descendente esquerdouerdo

else // se for filho direitoeito, o antecessor deixará de

antecessor.direito = null; // apontar para esse filho

atual = antecessor; // feito para atual apontar um nó válido ao sairmos do método

}

else // verificará as duas outras possibilidades, exclusão de nó com 1 ou 2 filhos

if (atual.direito == null) // neste caso, só tem o filho esquerdouerdo

{

if (atual == raiz)

raiz = atual.esquerdo;

else

if (ehFilhoesquerdouerdo)

antecessor.esquerdo = atual.esquerdo;

else

antecessor.direito = atual.esquerdo;

atual = antecessor;

}

else

if (atual.esquerdo == null) // neste caso, só tem o filho direitoeito

{

if (atual == raiz)

raiz = atual.direito;

else

if (ehFilhoesquerdouerdo)

antecessor.esquerdo = atual.direito;

else

antecessor.direito = atual.direito;

atual = antecessor;

}

else // tem os dois descendentes

{

NoArvore<Tipo> menorDosMaiores = ProcuraMenorDosMaioresDescendentes(atual);

atual.Info = menorDosMaiores.Info;

menorDosMaiores = null; // para liberar o nó trocado da memória

}

return true;

}

public NoArvore<Tipo> ProcuraMenorDosMaioresDescendentes(NoArvore<Tipo> noAExcluir)

{

NoArvore<Tipo> paiDoSucessor = noAExcluir;

NoArvore<Tipo> sucessor = noAExcluir;

NoArvore<Tipo> atual = noAExcluir.direito; // vai ao ramo direitoeito do nó a ser excluído, pois este ramo contém

// os descendentes que são maiores que o nó a ser excluído

while (atual != null)

{

if (atual.esquerdo != null)

paiDoSucessor = atual;

sucessor = atual;

atual = atual.esquerdo;

}

if (sucessor != noAExcluir.direito)

{

paiDoSucessor.esquerdo = sucessor.direito;

sucessor.direito = noAExcluir.direito;

}

return sucessor;

}

public int alturaArvore(NoArvore<Tipo> atual, ref bool balanceada)

{

int alturadireitoeita, alturaesquerdouerda, result;

if (atual != null && balanceada)

{

alturaesquerdouerda = 1 + alturaArvore(atual.esquerdo, ref balanceada);

alturadireitoeita = 1 + alturaArvore(atual.direito, ref balanceada);

result = Math.Max(alturaesquerdouerda, alturadireitoeita);

//if (alturadireitoeita > alturaesquerdouerda)

// result = alturadireitoeita;

//else

// result = alturaesquerdouerda;

if (Math.Abs(alturadireitoeita - alturaesquerdouerda) > 1)

balanceada = false;

}

else

result = 0;

return result;

}

/\* public int getAltura(NoArvore<Tipo> no)

{

if (no != null)

{

NoArvore<Tipo> atual = raiz;

int alturaAt = 0;

while (atual != null)

{

switch (atual.Info.CompareTo(no.Info))

{

case -1:

atual = atual.direito;

break;

case 0:

break;

case 1:

atual = atual.esquerdo;

break;

}

}

}

else

return -1;

}\*/

/\* public NoArvore<Tipo> Insert(Tipo item, NoArvore<Tipo> n)

{

if (n == null)

{

n = new NoArvore<Tipo>(item);

OndeExibir.Invalidate();

Application.DoEvents();

MessageBox.Show("Inclusao sem rotacao " + n.Info.ToString());

OndeExibir.Invalidate();

Application.DoEvents();

}

else

{

if (item.CompareTo(n.Info) < 0)

{

n.esquerdo = Insert(item, n.esquerdo);

if (getAltura(n.esquerdo) - getAltura(n.direito) == 2) // a propriedade Altura testa nulo!

if (item.CompareTo(n.esquerdo.Info) < 0)

n = RotateWithLeftChild(n);

else

n = DoubleWithLeftChild(n);

}

else

if (item.CompareTo(n.Info) > 0)

{

n.direito = Insert(item, n.direito);

if (getAltura(n.direito) - getAltura(n.esquerdo) == 2) // a propriedade Altura testa nulo!

if (item.CompareTo(n.direito.Info) > 0)

n = RotateWithRightChild(n);

else

n = DoubleWithRightChild(n);

}

//else ; - do nothing, duplicate value

n.altura = Math.Max(getAltura(n.esquerdo), getAltura(n.direito)) + 1;

OndeExibir.Invalidate();

Application.DoEvents();

}

return n;

}\*/

/\* private NoArvore<Tipo> RotateWithLeftChild(NoArvore<Tipo> no)

{

NoArvore<Tipo> temp = no; // apenas para declarar

temp = no.esquerdo;

no.esquerdo = temp.direito;

temp.direito = no;

no.altura = Math.Max(getAltura(no.esquerdo), getAltura(no.direito)) + 1;

temp.altura = Math.Max(getAltura(temp.esquerdo), getAltura(no)) + 1;

// System.Threading.Thread.Sleep(2000);

OndeExibir.Invalidate();

Application.DoEvents();

MessageBox.Show("Rotação à direitoeita do nó " + temp.Info.ToString());

OndeExibir.Invalidate();

Application.DoEvents();

return temp;

}\*/

/\* private NoArvore<Tipo> RotateWithRightChild(NoArvore<Tipo> no)

{

NoArvore<Tipo> temp = no; // apenas para declarar

temp = no.direito;

no.direito = temp.esquerdo;

temp.esquerdo = no;

no.altura = Math.Max(getAltura(no.esquerdo), getAltura(no.direito)) + 1;

temp.altura = Math.Max(getAltura(temp.direito), getAltura(no)) + 1;

//System.Threading.Thread.Sleep(2000);

OndeExibir.Invalidate();

Application.DoEvents();

MessageBox.Show("Rotação à esquerdouerda do nó "+temp.Info.ToString());

OndeExibir.Invalidate();

Application.DoEvents();

return temp;

}\*/

/\* private NoArvore<Tipo> DoubleWithLeftChild(NoArvore<Tipo> no)

{

OndeExibir.Invalidate();

Application.DoEvents();

MessageBox.Show("Rotação dupla à direitoeita do nó "+no.Info.ToString());

no.esquerdo = RotateWithRightChild( no.esquerdo );

OndeExibir.Invalidate();

Application.DoEvents();

return RotateWithLeftChild(no);

}

private NoArvore<Tipo> DoubleWithRightChild(NoArvore<Tipo> no)

{

OndeExibir.Invalidate();

Application.DoEvents();

MessageBox.Show("Rotação dupla à esquerdouerda do nó " + no.Info.ToString());

no.direito = RotateWithLeftChild( no.direito );

OndeExibir.Invalidate();

Application.DoEvents();

return RotateWithRightChild( no );

}\*/

}

}