|  |  |
| --- | --- |
|  | **UNIVERSIDAD ABIERTA INTERAMERICANA** |
|  | **FACULTAD DE TECNOLOGÍA INFORMÁTICA-COLEGIO UNIVERSITARIO** |
| **Carrera:** Ingeniería en Informática |
| **Asignatura: Programación I** |
| **TRABAJO PRÁCTICO NRO 2** |
|  | **Fecha Entrega: 14 de Noviembre 2020** |

|  |  |
| --- | --- |
| **Alumno:** Julián Ryan | **Legajo:** B00044969-T4 |
| **Obs:** | **Nota:** |

**Teoría**

1. ¿Qué es una lista doblemente enlazada?
2. ¿Cuál es la diferencia entre un árbol binario y un grafo?
3. ¿Qué técnicas existen para recorrer un árbol?
4. ¿A qué se conoce como grado de un nodo?
5. ¿Cuándo se dice que un árbol está balanceado?
6. ¿Cuál es la diferencia entre un archivo secuencial y un indexado?
7. ¿Para qué se utilizan las clases StreamReader y StreamWriter?
8. ¿Cómo puedo verificar que un archivo contiene información para la lectura?
9. ¿Para qué se utilizan las sentencias try y catch?
10. ¿Cuál es la diferencia entre el pasaje de parámetros por valor o por referencia a un método?

**Práctica**

Problemáticas:

* 1. ÁRBOLES BINARIOS: Dado el ejercicio visto en la clase del 17/10 agregar las siguientes funcionalidades:
     1. Agregar una clase Arbol que maneje internamente la estructura del árbol binario y ofrezca los métodos para el recorrido. Cada método podrá mostrar por consola o winform los datos visitados.
     2. Implementar los 4 algoritmos vistos en clase para recorrer el árbol.
     3. Implementar un mecanismo de búsqueda de un elemento dado. Utilizar una clase creada por usted como dato dentro de un Nodo del árbol. Por ejemplo -> Profesor
     4. (Punto extra) Permitir generar un árbol desde la interfaz visual utilizando el control treeview, un textbox para los datos a ingresar y un botón para ir creando los nodos.
     5. (Punto extra) Validar que cada nodo del treeview solo permita generar un nodo a la izq y derecha de cada elemento.
  2. ARCHIVOS: Se desea crear un pequeño sistema que consolide los saldos de las tarjetas de créditos registrados en una base de datos contra los datos recibidos desde los comercios en un archivo de texto.

El archivo recibido “tarjetas.txt” tiene el siguiente formato y datos para test (Los datos de nombres de columna pueden vienen en el primer renglón):

Tarjetas.txt (Datos separados por ‘;’)

|  |  |  |
| --- | --- | --- |
| **NroTarjeta** | **Comercio** | **Saldo** |
| **1234** | **A** | **3500** |
| **4321** | **B** | **6788** |
| **9999** | **C** | **5000** |

Se debe generar un archivo llamado “Consolidado.txt” con la siguiente información:

# NroTarjeta;SaldoConsolidado;Diferencia 1234;SI;0

**4321;NO;+1000**

**9999;NO;-500**

Si no hay diferencia entre los saldos, el segundo campo contendrá la palabra “SI”, de lo contrario “NO”. En el tercer campo se deberá volver la diferencia con el signo correspondiente. (“-“ o “+”). Si no hay diferencias se devolverá el valor 0.

Para realizar el test (Emular datos leídos de la DB) utilizar los siguientes casos de prueba:

# Tarjeta1(“1234”, 3500)

**Tarjeta2(“4321”, 5788)**

**Tarjeta3(“9999”, 5500)**

Confeccionar la solución en C#

Desarrollo:

Teoría

**1. ¿Qué es una lista doblemente enlazada?**

Una lista doblemente enlazada es una estructura de datos que consiste en nodos enlazados secuencialmente. Cada nodo contiene, al menos, tres campos, dos para los llamados enlaces, que son referencias al nodo siguiente y al anterior en la secuencia de nodos, y otro más para el almacenamiento de la información.

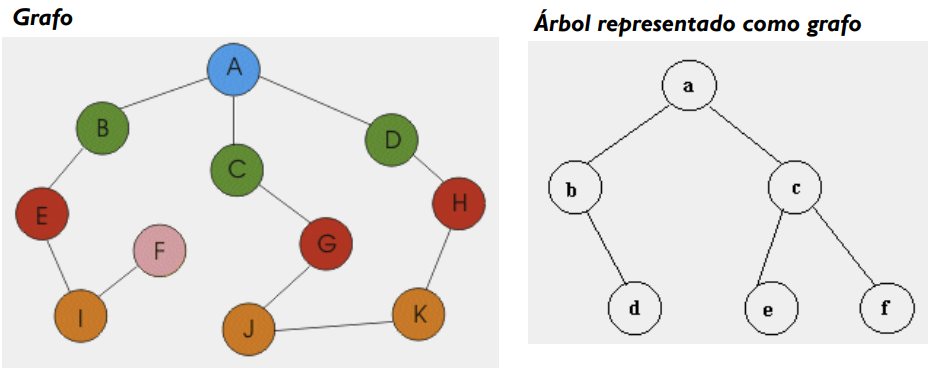
Una lista doblemente enlazada cuyos nodos contienen tres campos: un valor entero, el enlace al nodo siguiente, y el enlace al nodo anterior.

*Esquema de una lista doblemente enlazada*

La ventaja del doble enlace de los nodos es que permite recorrer la lista en cualquier dirección. Sin embargo, que agregar o eliminar un nodo requiere cambiar más enlaces que en un enlace simple

**2. ¿Cuál es la diferencia entre un árbol binario y un grafo?**

La diferencia que hay entre un Árbol Binario y un Grafo es que, el Grafo es cualquier conjunto de objetos llamados nodos unidos por enlaces llamados aristas y que permiten representar relaciones binarias entre elementos de un conjunto, mientras que un Árbol Binario es un como un tipo de grafo, con las características de ser acíclico, conexo y no dirigido (en el que existe exactamente un camino entre todo par de nodos)



*Esquemas de un Grafo y un Árbol Binario que representa sus diferencias*

**3. ¿Qué técnicas existen para recorrer un árbol?**

Las técnicas que existen para recorrer un árbol son: en Profundidad (PreOden, InOrden, PostOrden) y en Amplitud.

Métodos de recorrido en Profundidad:

PreOrden: (raíz, izquierdo, derecho). Para recorrer un árbol binario no vacío en preorden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:

1. Visite la raíz
2. Atraviese el sub-árbol izquierdo
3. Atraviese el sub-árbol derecho

InOrden: (izquierdo,raíz, derecho). Para recorrer un árbol binario no vacío en inorden (simétrico), hay que realizar las siguientes operaciones recursivamente en cada nodo:

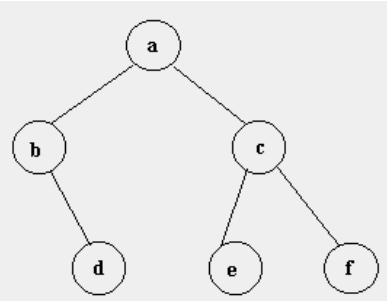
1. Atraviese el sub-árbol izquierdo
2. Visite la raíz
3. Atraviese el sub-árbol derecho

PostOrden: (izquierdo, derecho,raíz). Para recorrer un árbol binario no vacío en postorden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

1. Atraviese el sub-árbol izquierdo
2. Atraviese el sub-árbol derecho
3. Visite la raíz

En Amplitud: Consiste en ir visitando el árbol por niveles. Primero se visitan los nodos de nivel 1 (como mucho hay uno, la raíz), después los nodos de nivel 2, así hasta que ya no queden más.

Si se hace el recorrido en amplitud del siguiente árbol se visitaría los nodos en el orden: a,b,c,d,e,f.

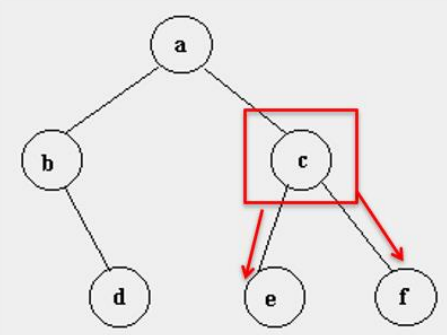


*Esquema que ejemplifica como sería el recorrido en Amplitud*

En este caso el recorrido no se realizará de forma recursiva sino iterativa, utilizando una cola. como estructura de datos auxiliar. El procedimiento consiste en encolar (si no están vacíos) los subárboles izquierdo y derecho del nodo extraido de la cola, y seguir desencolando y encolando hasta que la cola esté vacía.

**4. ¿A qué se conoce como grado de un nodo?**

Grado de un nodo: el número de descendientes directos que tiene.



*Ejemplo: c tiene grado 2, d tiene grado 0, a tiene grado 2*

**5. ¿Cuándo se dice que un árbol está balanceado?**

Se dice que un árbol está balanceado cuando para todos sus nodos la altura de la rama izquierda no difiere más de una unidad de la altura de la rama derecha (o vicecersa). El factor de equilibrio es la diferencia entre las alturas de un subárbol con las del otro subárbol: FE=0, -1 o 1.

* FE=0 -> Nodo perfectamente equilibrado
* FE=1 -> Nodo derecho es un nivel más alto
* FE=-1 -> Nodo Izquierdo es un nivel más alto
* FE>=|2| -> Hay que equilibrar el árbol

Para equilibrar o balancear un árbol hay que aplicar rotaciones

**6. ¿Cuál es la diferencia entre un archivo secuencial y un indexado?**

La diferencia entre un archivo secuencial y uno indexado es la forma de acceso a los registros. En el caso del primer tipo de archivo, su organización es secuencial y su forma de acceso a los distintos registros es únicamente secuencial. Es decir, para acceder al registro 50 debo haber pasado por los 49 anteriores. Mientras que el segundo tipo de archivo, tiene como particularidad el acceso restringido, por lo que es necesario tener clave para acceder a los mismos. Están ordenados por claves. Si bien la organización física de los registros es de acuerdo con el orden de grabación, su organización lógica está dada por la clave. La organización de un archivo indexado es indexada y, según el lenguaje, las formas de acceso pueden ser secuencial, random o al azar y dinámica.

**7. ¿Para qué se utilizan las clases StreamReader y StreamWriter?**

Las clases StreamReader y StreamWriter se utilizan en el contexto del trabajo con archivos para leer el contenido de los mismos y escribir nuevos datos.

La clase StreamReader tiene métodos que permiten: leer el próximo carácter disponible a partir de la posición actual y avanza (Read), leer un bloque de caracteres y lo almacena en un vector de chars (ReadBlock), leer una línea del archivo (Readline), leer el contenido de todo el archivo y lo guarda en un string (ReadToEnd), obtener el valor del próximo carácter disponible pero no avanza (Peek) y cerrar el stream liberando recursos y referencias al archivo (Close)

La clase StreamWriter tiene métodos que permiten: escribir una cadena de texto en el archivo (Write), escribir una cadena y agregar un salto de fin de línea (Writeline), devolver el contenido del buffer del stream al archivo (Flush) y realizar cualquier escritura pendiente sobre el archivo y cerrarlo (Close).

**8. ¿Cómo puedo verificar que un archivo contiene información para la lectura?**

La forma de verificar que un archivo existe y contiene información para la lectura es a través del método File.Exists(NombreArchivo). Esto sumado a un If permite prevenir fallas, archivos inexistentes, dañados o sin permisos de lectura.

**9. ¿Para qué se utilizan las sentencias try y catch?**

Las sentencias Try y Catch se utilizan para llevar a cabo la administración de excepciones. De esta manera se evita que un programa deje de funcionar, por ejemplo, cuando se ingresa un tipo de dato que no corresponde a lo esperado por el programa.

**10. ¿Cuál es la diferencia entre el pasaje de parámetros por valor o por referencia a un método?**

La diferencia entre el pasaje de parámetros por valor o por referencia a un método son que cuando se pasan parámetros por valor se realiza una copia de la variable y esta es enviada al método y no la original, entonces todos los cambios realizados dentro del método solo afectan a la copia actual, mientras que, en el pasaje por referencia, los punteros de las variables originales son pasadas a los métodos lo cual permite a los miembros de funciones, métodos, propiedades, indexadores, operadores y constructores cambiar el valor de los parámetros y hacer que ese cambio persista en el entorno de la llamada.

Práctica

1) ÁRBOLES BINARIOS: Dado el ejercicio visto en la clase del 17/10 agregar las siguientes funcionalidades:

a. Agregar una **clase Árbol** que maneje internamente la **estructura del árbol binario** y ofrezca los **métodos para el recorrido**. Cada ***método podrá mostrar por consola o winform los datos visitados***.

b. Implementar los **4 algoritmos** vistos en clase para recorrer el árbol.

c. Implementar un **mecanismo de búsqueda** de un elemento dado. Utilizar ***una clase*** creada por usted ***como dato dentro de un Nodo(lo que carga o lleva el árbol, la finalidad del TDA?)*** del árbol. Por ejemplo -> Profesor

*d. (Punto extra) Permitir generar un árbol desde la interfaz visual utilizando el control treeview, un textbox para los datos a ingresar y un botón para ir creando los nodos. (crear nodo, ver preOrden, inOrden, postOrden, en Amplitud )*

*e. (Punto extra) Validar que cada nodo del treeview solo permita generar un nodo a la izq y derecha de cada elemento*.

2) ARCHIVOS: Se desea crear un pequeño sistema que consolide los saldos de las tarjetas de créditos registrados en una base de datos contra los datos recibidos desde los comercios en un archivo de texto.

El archivo recibido “tarjetas.txt” tiene el siguiente formato y datos para test (Los datos de nombres de columna pueden vienen en el primer renglón):

Tarjetas.txt (Datos separados por ‘;’)

NroTarjeta Comercio Saldo

1234 A 3500

4321 B 6788

9999 C 5000

Se debe generar un archivo llamado “Consolidado.txt” con la siguiente información:

NroTarjeta; SaldoConsolidado; Diferencia

1234; SI; 0

4321; NO; +1000

9999; NO; -500

Si no hay diferencia entre los saldos, el segundo campo contendrá la palabra “SI”, de lo contrario “NO”. En el tercer campo se deberá volver la diferencia con el signo correspondiente. (“-“ o “+”). Si no hay diferencias se devolverá el valor 0.

Para realizar el test (Emular datos leídos de la DB) utilizar los siguientes casos de prueba:

Tarjeta1(“1234”, 3500)

Tarjeta2(“4321”, 5788)

Tarjeta3(“9999”, 5500)

***Programa de consola que muestre el archivo original, la base de datos, el resultado consolidado luego de contrastar los datos, o sea que tendría que trabajar con lo original, estático, el archivo txt, estático que hay que traer y el resultado, que se puede generar más de una vez. No?***

Tarjetas.txt (Datos separados por ‘;’)

NroTarjeta Comercio Saldo

1234 A 3500

4321 B 6788

9999 C 5000

Para realizar el test (Emular datos leídos de la DB) utilizar los siguientes casos de prueba:

Tarjeta1(“1234”, 3500)

Tarjeta2(“4321”, 5788)

Tarjeta3(“9999”, 5500)

Se debe generar un archivo llamado “Consolidado.txt” con la siguiente información:

NroTarjeta; SaldoConsolidado; Diferencia

1234; SI; 0

4321; NO; +1000

9999; NO; -500

Ejercicio 1 Árboles:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ArbolesEjemploCSharp

{

public class Nodo

{

public string Nombre { get; set; }

public int Edad { get; set; }

public Nodo(string nombre, int edad)

{

Nombre = nombre;

Edad = edad;

}

public Nodo Derecha { get; set; }

public Nodo Izquierda { get; set; }

}

}

using Microsoft.VisualBasic;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ArbolesEjemploCSharp

{

public partial class Form1 : Form

{

Nodo raiz;

Nodo seleccionado;

public Form1()

{

InitializeComponent();

}

//Metodo para crear nodo con los datos de nombre y edad pedidos por un InputBox

Nodo crearNodo()

{

string nombre = Interaction.InputBox("Ingrese nombre del familiar");

int edad = Convert.ToInt32(Interaction.InputBox("Ingrese la edad del familiar"));

//Se devuelve el nodo creado con los datos ingresados

return new Nodo(nombre,edad);

}

//Metodo para obtener la altura y el ancho del arbol para mostrar en Form a través de labels

void EvaluarArbol()

{

this.lblAltura.Text = $"Altura:{Alto(raiz)}";

int inicio = 0;

this.lblAncho.Text = $"Ancho:{Ancho(raiz,ref inicio)}";

}

int Ancho(Nodo n, ref int ancho)

{

//Se obtiene el ancho luego del recorrido un acumulador

if (n.Derecha == null && n.Izquierda == null)

ancho += 1;

if (n.Derecha != null) Ancho(n.Derecha, ref ancho);

if (n.Izquierda != null) Ancho(n.Izquierda, ref ancho);

return ancho;

}

int Alto(Nodo n)

{

//Se obtiene el alto luego del recorrido con acumuladores, luego se devuelve el de mayor valor

if (n == null) return 0;

int izq = Alto(n.Izquierda) + 1;

int der = Alto(n.Derecha) + 1;

return Math.Max(izq, der);

}

//Metodo para trabajar sobre el nodo seleccionado del arbol en el Treeview

private void treeView1\_AfterSelect(object sender, TreeViewEventArgs e)

{

CambiarSeleccion((Nodo)e.Node.Tag);

}

//Metodo para cambiar el nodo seleccionado a trabajar del arbol en el Treeview

void CambiarSeleccion(Nodo n)

{

seleccionado = n;

this.lblNombreNodo.Text = $"{n.Nombre} - {n.Edad}";

}

//Metodo para llenar el Treeview y mostrar, incluyendo el alto y ancho

public void LlenarTreeView()

{

treeView1.Nodes.Clear();

MostrarNodo(raiz, null, string.Empty);

treeView1.ExpandAll(); //para mostrar el Treeview desplegado

EvaluarArbol();

}

//Metodo para mostrar el arbol con sus nodos en el Treeview del Form

public void MostrarNodo(Nodo n, TreeNode tnpadre, string lado)

{

if (n == null) return;

TreeNode nuevo = new TreeNode();

if (tnpadre == null && lado==String.Empty)

{

//Cuando se trata del nodo inicial o nodo raiz

tnpadre = new TreeNode();

nuevo.Text = $"{n.Nombre} - {n.Edad}";

nuevo.Tag = n;

treeView1.Nodes.Add(nuevo);

}

else

{

//Cuando es un nodo sucesor o hijo

nuevo.Text = $"{lado} - {n.Nombre} - {n.Edad}";

nuevo.Tag = n;

tnpadre.Nodes.Add(nuevo);

}

if (n.Derecha != null) MostrarNodo(n.Derecha, nuevo, "D");

if (n.Izquierda != null) MostrarNodo(n.Izquierda, nuevo, "I");

}

//Boton para agregar familiar (nodo raiz)

private void button1\_Click(object sender, EventArgs e)

{

//En caso de que ya haya un nodo inicial o raiz, se pregunta si se quiere reemplazar

if (raiz != null)

{

DialogResult r = MessageBox.Show("Se eliminará el árbol, desea continuar?", "Consulta", MessageBoxButtons.YesNo);

if (r == DialogResult.Yes)

{

raiz = crearNodo();

}

}

else

{

//En caso de no haber nodo inicial o raiz, se crea

raiz = crearNodo();

}

//Metodo para mostrar label y seleccionar raiz en Treeview

CambiarSeleccion(raiz);

//Metodo para llenar el Treeview

LlenarTreeView();

}

//Boton para agregar un familiar a la derecha del nodo selecciondo

private void button2\_Click(object sender, EventArgs e)

{

if (seleccionado != null)

{

seleccionado.Derecha = crearNodo();

LlenarTreeView();

}

else

MessageBox.Show("Debe tener algun nodo seleccionado");

}

//Boton para agregar un familiar a la izquierda del nodo selecciondo

private void button3\_Click(object sender, EventArgs e)

{

if (seleccionado != null)

{

seleccionado.Izquierda = crearNodo();

LlenarTreeView();

}

else

MessageBox.Show("Debe tener algun nodo seleccionado");

}

//Metodo de impresion de los recorridos del arbol, que muestra los nodos visitados por el recorrido

void Visitar(Nodo n)

{

this.txtRecorrido.Text += "---" + n.Nombre + n.Edad;

}

//Boton para recorrido InOrden

private void button4\_Click(object sender, EventArgs e)

{

this.txtRecorrido.Text = string.Empty;

RecorridoInOrden(raiz);

}

//Metodo con algoritmo del recorrido InOrden

void RecorridoInOrden(Nodo n)

{

if (n.Izquierda!=null) RecorridoInOrden(n.Izquierda);

Visitar(n); //Raiz en el medio

if (n.Derecha != null) RecorridoInOrden(n.Derecha);

}

//Boton para recorrido PreOrden

private void button5\_Click(object sender, EventArgs e)

{

this.txtRecorrido.Text = string.Empty;

RecorridoPreOrden(raiz);

}

//Metodo con algoritmo del recorrido PreOrden

void RecorridoPreOrden(Nodo n)

{

Visitar(n); //Raiz al principio

if (n.Izquierda != null) RecorridoPreOrden(n.Izquierda);

if (n.Derecha != null) RecorridoPreOrden(n.Derecha);

}

//Boton para recorrido PostOrden

private void button6\_Click(object sender, EventArgs e)

{

this.txtRecorrido.Text = string.Empty;

RecorridoPostOrden(raiz);

}

//Metodo con algoritmo del recorrido PostOrden

void RecorridoPostOrden(Nodo n)

{

if (n.Izquierda != null) RecorridoPostOrden(n.Izquierda);

if (n.Derecha != null) RecorridoPostOrden(n.Derecha);

Visitar(n); //Raiz al final

}

//Boton para recorrido en Amplitud

private void button7\_Click(object sender, EventArgs e)

{

this.txtRecorrido.Text = string.Empty;

RecorridoAmplitud(raiz);

}

//Metodo con algoritmo del recorrido en Amplitud

void RecorridoAmplitud(Nodo n)

{

//Se pasan los datos a una cola y se imprime la cola

Queue<Nodo> Cola = new Queue<Nodo>();

Cola.Enqueue(raiz);

while (Cola.Count > 0)

{

Nodo Actual = Cola.Dequeue();

if (Actual == null)

continue;

Cola.Enqueue(Actual.Izquierda);

Cola.Enqueue(Actual.Derecha);

Visitar(Actual);

}

}

//Boton para busqueda del nombre del familiar

private void button8\_Click(object sender, EventArgs e)

{

Nodo buscado = new Nodo(txtBusqueda.Text, 0);

Busqueda(raiz, buscado);

}

//Metodo con algoritmo de busqueda del nombre del familiar

void Busqueda(Nodo n, Nodo buscado)

{

if (n != null)

{

if (n.Nombre == buscado.Nombre) {

buscado = n;

MessageBox.Show($"El edad de {buscado.Nombre} es de {buscado.Edad}");

}

else {

if (n.Izquierda != null) Busqueda(n.Izquierda, buscado);

if (n.Derecha != null) Busqueda(n.Derecha, buscado);

}

}

else

MessageBox.Show("Debe escribir algún nombre!");

}

}

}

Ejercicio 2 Archivos: