Manual Técnico Proyecto Uno

Carlos Agustin Ché Mijangos, 201800624

Facultad de Ingeniería, Universidad de San Carlos

Laboratorio Estructuras de Datos

Aux. Susel Retana

18 de junio 2020

Contenido

DESCRIPCION **3**

LIBRERIAS UTILIZADAS **3**

CLASES CREADAS **4**

ESTRUCTURA DEL PROGRAMA **6**

MAIN **7**

MÉTODOS 7

```
void logIn();
//ACCIONES ADMINISTRADOR
void menuAdmin();
void agregarUsuario();//1
void reporteMatriz();//2
void reporteActivosDepa();//3
void reporteActivosEmp();//4
void reporteTransacciones();//5
void activosUsuario();//6
void activosRentadosUsuario();//7
void ordenarTransacciones();//8
//ACCIOENS USUARIO
void menuUsuario();
void agregarActivo();//1
void generarID();
void generarID2();
void modificarActivo();//3
void eliminarActivo();//2
void rentarActivo();//4
void misActivosRentados();//6
void activosRentados();//5
void devolverActivo();//7
```

..... 7

LISTA CIRCULAR DOBLE **7**

MÉTODOS 7

MATRIZ DISPERSA **8**

MÉTODOS 8

ARBOL AVL **8**

MÉTODOS 8

ROTACIONES 8

Rotación Simple a la derecha 9

Rotación simple a la izquierda.....9

Rotación Doble a la Derecha.....10

Rotación Doble a la derecha 11

DESCRIPCION

El proyecto trata sobre la renta de activos de diferentes usuarios y un administrador que tiene acceso a la información de estos, puede ver reportes de los usuarios activos y las transacciones que estos realizada además de crear nuevos. Los usuarios pueden crear activos, rentar activos y ver cómo van sus activos.

Para el manejo de la aplicación se utilizaron diferentes estructuras de datos las cuales son una matriz dispersa para los usuarios, un árbol balanceado para los activos de cada usuario y una lista circular doble para las transacciones.

LIBRERIAS UTILIZADAS

Aquí se muestra las librerías utilizadas para la solución del proyecto.

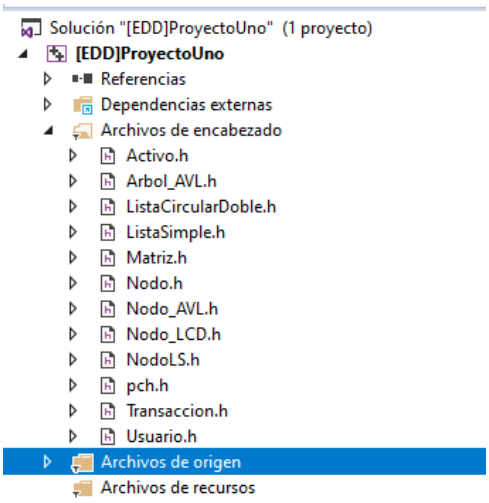
```
#include "pch.h"
#include <iostream>
#include "Matriz.h"
#include "Usuario.h"
#include <stdlib.h>
#include <string>
#include "Nodo.h"
#include <time.h>
#include "Arbol_AVL.h"
#include "Activo.h"
#include "Transaccion.h"
#include "ListaCircularDoble.h"
#include "ListaSimple.h"
#include <fstream>
```

Estas fueron utilizadas para realizar diferentes acciones a continuación se lista una descripción de estas:

- Pch.h esta es la librería que crea predeterminadamente el IDE
- Iostream es la librería general de c++ para acceder a funciones como imprimir en pantalla
- Stdlib.h nos sirvió para comparar cadenas de caracteres y algunas operaciones con strings
- String nos sirve para poder utilizar el tipo de dato string en nuestras variables
- Time.h fue utilizada para la creación de números aleatorios
- Fstream fue creado para escribir un archivo de extensión .dot para generar los reportes pedidos

CLASES CREADAS

Aquí se muestran las clases creadas para resolver el presente proyecto:



Estas clases se crearon con sus respectivos atributos y métodos para poder crear las estructuras de datos pedidas en el proyecto, además de crear una solución adecuada para este, una explicación de lo que realiza cada clase se presenta a continuación:

- Activo esta clase nos permite crear objetos de tipo activo que son los activos que crear los usuarios y estos pueden rentar después, esta clase tiene un ID único un nombre y una descripción todos de tipo string
- Arbol_AVL esta clase nos permite manejar nuestra estructura de datos de tipo árbol balanceado, es en esta donde se presentan todos sus métodos, las rotaciones, insertar, eliminar, modificar, graficar.
- ListaCircularDoble esta clase nos permite manejar nuestra estructura de datos de tipo lista circular doblemente enlazada, esta presenta los métodos de insertar, eliminar, imprimir en consola y graficar.
- Matriz esta clase nos permite manejar nuestra estructura de datos de tipo matriz dispersa con sus nodos arriba, abajo, siguiente, anterior, atrás, adelante, y su atributo tipo Usuario.
- Las clases Nodo, Nodo_AVL, Nodo_LCD, son las clases nodo que contienen punteros y nos permiten manejar las estructuras de datos, matriz dispersa, árbol balanceado y lista circular doblemente enlazada respectivamente.
- La clase Transaccion tiene un ID de transacción un ID de activo, fecha, tiempo de duración, el nombre de la empresa, usuario y departamento de quien realizar una renta o devuelve una.
- La clase Usuario nos permite crear objetos de tipo usuario para almacenarlos en la matriz dispersa y llevar un control de estos.

ESTRUCTURA DEL PROGRAMA

Al principio el programa inicia en el main, la cual llama a nuestra función Log In que cuando se ingresa admin como nombre de usuario y contraseña nos permite entrar al modulo del administrador, y si no va a pedir el departamento y empresa del usuario que dese ingresar.

Al entrar al modulo administrador, se hace llamado a la función menú administrador la cual posee 9 opciones, las cuales se deben seleccionar el numero de la opción que se desea realizar seguido de enter luego esta hace llamado a la función que se desea realizar y al concluir su acción regresa al menú del administrador.

El modulo de usuarios funciona de la misma manera que el del administrador con la diferencia que solo tiene 8 acciones.

Para el manejo de las estructuras de datos, la estructura de datos principal es la matriz dispersa la cual esta declarada en la clase del main, y todas las funciones pueden acceder a esta, de la misma manera que la lista circular doble, es decir solo tenemos una matriz y una lista circular doble.

El árbol balanceado si tenemos muchos de estos los cuales es uno por usuario, el cual este contenido en la clase usuario para que cuando se acceda con un usuario se guarda un puntero de este usuario de forma global y así este puede acceder a los métodos del árbol balanceado.

MAIN

Métodos

```
void logIn();
//ACCIONES ADMINISTRADOR
void menuAdmin();
void agregarUsuario();//1
void reporteMatriz();//2
void reporteActivosDepa();//3
void reporteActivosEmp();//4
void reporteTransacciones();//5
void activosUsuario();//6
void activosRentadosUsuario();//7
void ordenarTransacciones();//8
//ACCIOENS USUARIO
void menuUsuario();
void agregarActivo();//1
void generarID();
void generarID2();
void modificarActivo();//3
void eliminarActivo();//2
void rentarActivo();//4
void misActivosRentados();//6
void activosRentados();//5
void devolverActivo();//7
```

LISTA CIRCULAR DOBLE

Métodos

```
1   #pragma once
2   #include <iostream>
3   #include "Nodo_LCD.h"
4   #include "Transaccion.h"
5   class ListaCircularDoble
6   {
7   public:
8       ListaCircularDoble() {
9           tam = 0;
10          primero = 0;
11      }
12      Nodo_LCD* primero;
13      int tam;
14
15      void ingresar(Transaccion* transaccion);
16      bool estaVacia();
17      void imprimir();
18      void graficar();
19      void ordenarAscendente();
20      void ordenarDescendente();
21  };
22
23
```

MATRIZ DISPERSA

Métodos

```
#include "Usuario.h"
#include "Nodo_AVL.h"
#include <string>
using namespace std;
class Matriz
{
    Usuario *admin = new Usuario("admin", "admin", "", "", "ADMIN");
public:
    Nodo *cabecera;
    Matriz()
    {
        cabecera = new Nodo("admin", -1, admin);
    }
    void insertarElemento(string usuario, int numero, string empresa, string departamento, Usuario *user);
    Nodo *crearEmpresa(string departamento);
    Nodo *crearDepartamento(string empresa);
    Nodo *buscarEmpresa(string empresa, Nodo *inicio);
    Nodo *buscarDepa(string departamento, Nodo *inicio);
    bool verificarEmpresa(string empresa, Nodo *inicio, Nodo *USER);
    bool verificarDepa(string departamento, Nodo *inicio, Nodo *USER);
    void graficar();
    Nodo *buscarUsuario(string usuario, string password, string departamento, string empresa);
    Nodo *buscarUsuario(string usuario, string departamento, string empresa);
    void graficarDepa(Nodo* depa);
    void graficarEmp(Nodo* emp);
    void imprimirActivos(Usuario* mio);
    Nodo_AVL* rentarActivo(string ID);
    string emp_, dep_, user_;
};
```

ARBOL AVL

Métodos

```
class Arbol_AVL
{
public:
    std::string dato, datoDos, datoRentado, datoDosRentado;
    Arbol_AVL() { raiz_ = 0; contador = 0; auxi = 0; }

    Nodo_AVL* rotacionSimpleDerecha(Nodo_AVL* padre, Nodo_AVL* aux);
    Nodo_AVL* rotacionSimpleIzquierda(Nodo_AVL* padre, Nodo_AVL* aux);
    Nodo_AVL* rotacionDobleIzquierda(Nodo_AVL* padre, Nodo_AVL* aux);
    Nodo_AVL* rotacionDobleDerecha(Nodo_AVL* padre, Nodo_AVL* aux);

    void insertar(int dato, Nodo_AVL* raiz, int* b, Activo* activo);

    void inOrden(Nodo_AVL* raiz);

    void eliminar(Nodo_AVL* raiz,Nodo_AVL* padre,  int dato, int *tipo);
    Nodo_AVL* balanceoIzq(Nodo_AVL* padre, int *tipo);
    Nodo_AVL* balanceoDer(Nodo_AVL* padre, int *tipo);
    void reordenar(Nodo_AVL* padre, Nodo_AVL* aux, int *tipo);
    Nodo_AVL* raiz_;

    void graficar(int numero);
    void cuadritoG(Nodo_AVL*, std::string, int);
    void conexiones(Nodo_AVL*, std::string, int);

    void conexionesRentado(Nodo_AVL* raiz, std::string dato_, int numero);
    void cuadritoGRentado(Nodo_AVL*, std::string, int);
    int contador;

    void modificarActivo(int dato, Nodo_AVL* aux, std::string des, std::string nombre, int cont_);
    void preOrden(Nodo_AVL* raiz);

    void imprimir(Nodo_AVL* raiz);
    void buscarID(std::string ID, Nodo_AVL* raiz);
    Nodo_AVL* auxi;
    void imprimirActivosRentados(Nodo_AVL* raiz);
};
```

Rotaciones

Aquí se presenta las rotaciones que se implementaron en el proyecto.

Rotación Simple a la derecha

Foto del Código

```
#include <iostream>

//ROTACIONES SIMPLES
Nodo_AVL* Arbol_AVL::rotacionSimpleDerecha(Nodo_AVL* padre, Nodo_AVL* aux)
{   //El padre estaba apuntando a su hijo izq, que es aux
    padre->hijoIzq = aux->hijoDer;//apunto el hijo izq del padre a lo que tenga a la derecha aux
    aux->hijoDer = padre;//apunto el hijo der el aux al padre
    padre->factorEqu = 0;//cambio el factor de equilibrio del padre a 0
    return aux;
}
```

Explicación

Para realizar una rotación simple a la derecha, llamo a esta función cuando se cumplen las condiciones para realizar, le pido de parámetros el nodo padre que tiene a los dos hijos izquierdos y el nodo aux es su hijo izquierdo, al principio el hijo izquierdo del padre es aux, lo primero que hago es apuntarle como hijo izquierdo al padre lo que el aux tenga como hijo derecho ya que este adoptara su posición, luego a aux le apunto el padre con el hijo derecho, con esto ya adopto su posición y por ultimo le pongo el factor de equilibro del padre a 0.

Rotación simple a la izquierda

Foto del Código

```
Nodo_AVL* Arbol_AVL::rotacionSimpleIzquierda(Nodo_AVL* padre, Nodo_AVL* aux)
{
    //El padre estaba apuntado a su hijo der, que es aux
    padre->hijoDer = aux->hijoIzq;//apunto el hijo der del padre a lo que tenga a la izquierda aux
    aux->hijoIzq = padre;//apunto el hijo izq de aux al padre
    padre->factorEqu = 0;//cambio el facto de equilibro del padre a 0
    return aux;
}
```

Explicación

Para realizar una rotación simple a ala izquierda, llamo a esta función cuando se cumplen las condiciones para realizar, le pido de parámetros el nodo padre que tiene a los dos hijos derechos y el nodo aux que es su hijo derecho, al principio el hijo derecho del padre es aux, lo primero que hago es apuntarle como hijo derecho al padre lo que aux tenga como hijo izquierdo

ya que esta adoptara su posición, luego a aux le apunto el padre como hijo izquierdo, con esto ya adopto su posición y pongo el factor de equilibrio del padre a 0.

Rotación Doble a la Derecha

Foto del Código

```
//ROTACIONES DOBLES
Nodo_AVL* Arbol_AVL::rotacionDobleIzquierda(Nodo_AVL* padre, Nodo_AVL* aux)
{   //Al principio el padre esta apuntando a aux como su hijo izq
    Nodo_AVL* temp;
    temp = aux->hijoDer;//creo un nodo temp que me ayudara a cambiar los valores y le asigno el hijo der de aux
    padre->hijoIzq = temp->hijoDer;//al padre le asigno como hijo izquierdo lo que el hijo derecho de aux tengo como der
    temp->hijoDer = padre;//le asigno a temp como hijo derecho al padre
    aux->hijoDer = temp->hijoIzq;//luego a aux como hijo derecho le asigno lo que temp tuviera ala izq
    temp->hijoIzq = aux;//y asigno como  hijo izq de temp a aux
    if (temp->factorEqu == -1)
    {
        padre->factorEqu = 1;
    }
    else {
        padre->factorEqu = 0;
    }
    if (temp->factorEqu == 1)
    {
        aux->factorEqu = -1;
    }
    else {
        aux->factorEqu = 0;
    }
    return temp;
}
```

Explicación

Para realizar la rotación doble a la izquierda, llamo a esta función cuando se cumplen sus condiciones, le mando como parámetro al padre con un hijo izquierdo y otro debajo de el a la derecha y el aux que es el hijo izquierdo del padre, al principio aux es el hijo izquierdo del padre, creo una variable temp que me ayudara a hacer los cambios de posición y le asigno el hijo derecho de aux, luego le asigno como hijo izquierdo del padre lo que temp tenga a la derecha, luego apunto el hijo derecho de temp como el padre, luego le doy como hijo derecho de aux lo que temp tenga a la izquierda y apunto como hijo izquierdo de temp a aux, con esto ya se realizo el cambio de posición.

Rotación Doble a la derecha

Foto del Código

```
Nodo_AVL* Arbol_AVL::rotacionDobleDerecha(Nodo_AVL* padre, Nodo_AVL* aux)
{
    Nodo_AVL* temp;
    temp = aux->hijoIzq;
    padre->hijoDer = temp->hijoIzq;
    temp->hijoIzq = padre;
    aux->hijoIzq = temp->hijoDer;
    temp->hijoIzq = aux;

    if (temp->factorEqu == 1)
    {
        padre->factorEqu = -1;
    }
    else {
        padre->factorEqu = 0;
    }
    if (temp->factorEqu == -1)
    {
        aux->factorEqu = 1;
    }
    else {
        aux->factorEqu = 0;
    }
    return temp;
}
```

Explicación

Para realizar la rotación doble a la derecha, llamo a esta función cuando se cumplen sus condiciones, le mando como parámetro al padre con un hijo derecho y otro debajo de el a la izquierda y el aux que es el hijo derecho del padre, al principio aux es el hijo derecho del padre, creo una variable temp que me ayudara a hacer los cambios de posición y le asigno el hijo izquierdo de aux, luego le asigno como hijo derecho del padre lo que temp tenga a la izquierda, luego apunto el hijo izquierdo de temp como el padre, luego le doy como hijo izquierdo de aux lo que temp tenga a la derecha y apunto como hijo derecho de temp a aux, con esto ya se realizó el cambio de posición.