# ANALISIS Y DISEÑO DE ALGORITMOS I

PROYECTO DE CURSADA 2022

Simulación Banco

ENTREGA FINAL

ARTAZA SHEILA

CAMPO AGUSTÍN

GMAILS: [artazasheii@gmail.com](mailto:artazasheii@gmail.com)

agustincampo\_@hotmail.com

GRUPO: 58

AYUDANTE: FEDERICO.

Introducción:

En este informe vamos a explicar los algoritmos desarrollados, tipos de datos y estructuras que aplicamos para implementar la simulación del funcionamiento de un banco, la llegada, espera y atención de los clientes durante el tiempo activo del mismo, implementado en el lenguaje C++.

También vamos a realizar el análisis de la complejidad temporal de los métodos implementados y el impacto de las estructuras utilizadas.

Desarrollo:

Algoritmos, datos y estructuras utilizadas:

Para la implementación de dicho banco utilizamos como estructuras 3 filas (que funcionan como las colas), las cuales fueran previamente creadas a partir de la lista. Aplicamos esta estructura ya que al ser (first-in, first-out) coinciden con el orden de llegada y salida de los clientes de dicho banco, con esto se cumple que el primero en ingresar va a ser el primero en atenderse.

Aparte se utilizó una lista para listar todas las operaciones realizadas por el banco en el día.

Implementamos una clase cliente con todos los datos solicitados del mismo y una clase banco en el cual implementamos todos los métodos que este necesita.

Al iniciar el banco, todos los clientes se van a encontrar en la cola general que es la única que se encuentra habilitada, las otras dos colas, las especiales van a ser habilitadas posteriormente cuando se desee abrir una cola especial bajo ciertos criterios.

Entonces arrancamos de un menú principal, que va a tener un par de opciones donde el banco va a elegir qué operación realizar.

El mismo consta de 5 opciones en las cuales va a poder realizar:

1. Ingresar un nuevo usuario: pide los datos del nuevo usuario y lo agrega a la cola general, de no haber colas especiales con los criterios del cliente.

En caso contrario se lo agrega a la cola especial correspondiente.

1. Atender un cliente: pide que cola se quiere atender y muestra por pantalla los datos del usuario.
2. Abrir una nueva cola especial: de no haber colas especiales abiertas, se solicita los criterios con los cuales quiere abrirse la nueva cola y se habilita.

Posteriormente se revisa en la cola general si algún usuario cumple con dichos criterios y de ser así se lo saca de la cola general y se lo agrega a la cola especial habilitada.

1. Cerrar una cola especial: pide que cola especial se desea cerrar, si está vacía se podrá cerrar y ya no estará habilitada para los usuarios nuevos.
2. Listar las operaciones atendidas dentro de un monto mínimo y un monto máximo: se pide un monto mínimo y uno máximo, y a partir de estos los usuarios que hayan realizado operaciones dentro de esos montos son mostradas en una lista.

Complejidad de los algoritmos:

1. Ingresar cliente: su complejidad es O (1), ya que para ingresar un cliente solo invoca a una función donde todos los datos del mismo son solicitados, y lo agrega a la cola general, siempre al principio, sin tener que recorrer toda la lista.
2. Atender Cliente: su complejidad es O(n), ya que invocamos a cantelem que su complejidad pertenece a O(n), con n siendo la cantidad de nodos de la fila. ACLARACIÓN: se invoca 3 veces dentro de atender clientes a cantelem, ya que disponemos de 3 colas distintas.
3. Abrir cola esp: su complejidad es O(n), n=tamaño de la fila(nodos), ya que para abrir una nueva cola especial debo recorrer toda la cola general para ver cuál nodo cumple con los criterios de la nueva cola especial.
4. Criterio: su complejidad es O (1), ya que no recorro nada, sino que solo pregunto los criterios que se desean implementar para abrir la lista especial nueva.
5. Acomodar cliente: su complejidad es O(n), n=tamaño de la fila(nodos), ya que para acomodar el cliente tengo que recorrer toda la cola general para chequear si coincide con la cola especial.
6. Cerrar cola esp: su complejidad es O (1), ya que solo cambia el estado de una variable booleana para indicar que la misma se cerró, teniendo en cuenta que la cantidad de elementos en la misma fila que quiere ser cerrada es igual a cero.
7. Lista de op: su complejidad es O(n), n=tamaño de la fila(nodos), ya que recorro toda la lista de operaciones realizadas.

Análisis del impacto de las estructuras elegidas:

A la hora de plantear el problema podríamos haber implementado la estructura de un arreglo, pero teniendo en cuenta los beneficios que nos daba la estructura utilizada (una secuencia de nodos), tales como posibilidad de tener un tamaño variable de la fila (cola), una menor complejidad para la implementación de la estructura.

Una clara ventaja del arreglo con respecto a la lista es poder acceder a cualquier elemento con complejidad constante. Ventaja que dado nuestro problema no es utilizada ya que siempre operamos en el inicio o el final de la estructura implementada.

Una de las mejoras que podría aplicar seria en cantidad de elementos ya que al implementar en los atributos de la lista uno que sea cantelem; podemos bajar la complejidad de esta función de O(n), siendo n la cantidad de elementos a O (1).

Conclusión:

Para finalizar concluimos que, gracias a este trabajo especial, adquirimos conocimientos sobre el lenguaje de programación C++, y los pudimos implementar.

Además, pudimos aplicar la teoría aprendida en las clases teóricas, como desarrollo de especificación nereus, tipos de datos abstractos, entre otros.

Código fuente:

#ifndef CLIENTE\_H

#define CLIENTE\_H

#include <iostream>

using namespace std;

class Cliente

{

public:

Cliente(); //constructor

virtual ~Cliente(); //destructor

void ingresarcliente();

void ingresaredad();

void elegiroperacion();

void elegirdestinatario();

void montoOperacion();

void esCliente();

//devuelvo;

string muestranom();

int muestraedad();

string muestradest();

string muestraop();

int muestramonto();

bool muestraCl();

bool operator== (const Cliente & otrocliente) const;

bool operator!= (const Cliente & otrocliente) const;

//protected:

private:

string nombre;

int edad;

string operacion;

string destinatario;

int montoOp;

bool esClien;

};

#endif // CLIENTE\_H

#include "Cliente.h"

#include <iostream>

using namespace std;

Cliente::Cliente()

{

}

Cliente::~Cliente()

{

//dtor

}

void Cliente:: ingresarcliente() {

cin >> nombre;

}

void Cliente:: ingresaredad() {

cin >> edad;

}

void Cliente:: elegiroperacion(){

int opcion = 0;

cin >> opcion;

switch (opcion) {

case 1: operacion = "Retiro";

break;

case 2: operacion = "Deposito";

break;

case 3: operacion = "Transferencia";

break;

case 4: operacion = "Pago";

break;

}

}

void Cliente:: elegirdestinatario() {

int opcion = 0;

cin >> opcion;

switch (opcion) {

case 1: destinatario = "Persona";

break;

case 2: destinatario = "Banco";

break;

case 3: destinatario = "Impuesto";

break;

}

}

void Cliente:: montoOperacion() {

cin >> montoOp;

}

void Cliente:: esCliente() {

int verificacion;

cin >> verificacion;

if (verificacion == 1) {

esClien = true;

}

else {

esClien = false;

}

}

string Cliente:: muestranom() {

return (nombre);

}

int Cliente:: muestraedad(){

return (edad);

}

string Cliente:: muestraop(){

return (operacion);

}

int Cliente:: muestramonto(){

return (montoOp);

}

string Cliente:: muestradest(){

return (destinatario);

}

bool Cliente:: muestraCl(){

return (esClien);

}

bool Cliente:: operator== (const Cliente & otrocliente) const {

return false;

}

bool Cliente:: operator!= (const Cliente & otrocliente) const {

return true;

}

#ifndef BANCO\_H

#define BANCO\_H

#include "Cliente.h"

#include "lista.h"

#include "Fila.h"

#include <string.h>

#include <conio.h>

using namespace std;

template<typename B>

class Banco

{

public:

Banco();

~Banco();

void ingresarCliente();

void atenderCliente();

void abrirColaEsp();

void criterios(Fila<Cliente> & cola);

void acomodarclientes();

void cerrarColaEsp();

void listaDeOp();

void Menu();

private:

Fila<Cliente> colaGeneral;

Fila<Cliente> colaespecial\_1;

Fila<Cliente> colaespecial\_2;

lista<Cliente> lista\_de\_atendidos;

};

#endif // BANCO\_H

#include <iostream>

#include "Banco.h"

using namespace std;

template<typename B>

Banco<B>:: Banco() {

}

template<typename B>

Banco<B>:: ~Banco() {

}

Cliente nuevoUsuario(){

Cliente nuevo\_usuario;

cout << "Registre sus datos: " << endl;

cout << "Nombre completo: " ;

cin.ignore();

nuevo\_usuario.ingresarcliente();

cout << "Edad: " ;

nuevo\_usuario.ingresaredad();

cout << "Elija una operaracion: " << endl;

cout << "1- Retiro" << endl;

cout << "2- Deposito" << endl;

cout << "3- Transferencia" << endl;

cout << "4- Pago" << endl;

nuevo\_usuario.elegiroperacion();

cout << "Elija un destinatario: " << endl;

cout << "1-Persona" << endl;

cout << "2-Banco" << endl;

cout << "3-Impuesto" << endl;

nuevo\_usuario.elegirdestinatario();

cout << "Ingrese el monto de la operacion: ";

nuevo\_usuario.montoOperacion();

cout << "Usted es?" << endl;

cout << "1-Cliente" << endl;

cout << "2-No cliente" << endl;

nuevo\_usuario.esCliente();

return nuevo\_usuario;

}

template<typename B>

void Banco<B>::ingresarCliente(){

Cliente nuevo\_usuario;

nuevo\_usuario = nuevoUsuario();

if (colaespecial\_1.getcolaabierta() == true){

if ((colaespecial\_1.getoperacion() == nuevo\_usuario.muestraop()) && (colaespecial\_1.getcliente() == nuevo\_usuario.muestraCl())){

colaespecial\_1.agregar(nuevo\_usuario);

}

else {

if (colaespecial\_2.getcolaabierta() == true) {

if ((colaespecial\_2.getoperacion() == nuevo\_usuario.muestraop()) && (colaespecial\_2.getcliente() == nuevo\_usuario.muestraCl())){

colaespecial\_2.agregar(nuevo\_usuario);

}

else colaGeneral.agregar(nuevo\_usuario);

}

else colaGeneral.agregar(nuevo\_usuario);

}

}

else {

if (colaespecial\_2.getcolaabierta() == true) {

if ((colaespecial\_2.getoperacion() == nuevo\_usuario.muestraop()) && (colaespecial\_2.getcliente() == nuevo\_usuario.muestraCl())){

colaespecial\_2.agregar(nuevo\_usuario);

}

else colaGeneral.agregar(nuevo\_usuario);

}

else {

colaGeneral.agregar(nuevo\_usuario);

}

}

}

void imprimirCliente(Cliente cliente) {

cout << "Nombre: " <<cliente.muestranom() << endl;

cout << "Edad: " << cliente.muestraedad() << endl;

cout << "Operacion: " <<cliente.muestraop() << endl;

cout << "Destinatario: " << cliente.muestradest() << endl;

cout << "Monto: " << cliente.muestramonto() << endl;

cout << "Cliente 1(si)/2(no): " << cliente.muestraCl() << endl;

}

template<typename B>

void Banco<B>:: atenderCliente() {

Cliente persona;

cout << "1-Cola General" << "Personas :" << colaGeneral.cantidadelementos() << endl;

cout << "2-Cola especial 1" << "Personas :" << colaespecial\_1.cantidadelementos() << endl;

cout << "3-Cola especial 2" << "Personas :" << colaespecial\_2.cantidadelementos() << endl;

int opcion;

cin >> opcion;

if ((opcion == 1) && (colaGeneral.cantidadelementos() != 0)) {

persona = colaGeneral.devolverelemento();

colaGeneral.sacarelemento();

lista\_de\_atendidos.agregaralfinal(persona);

imprimirCliente(persona);

}

if ((opcion == 2) && (colaespecial\_1.cantidadelementos() != 0)) {

persona = colaespecial\_1.devolverelemento();

colaespecial\_1.sacarelemento();

lista\_de\_atendidos.agregaralfinal(persona);

imprimirCliente(persona);

}

if ((opcion == 3) && (colaespecial\_2.cantidadelementos() != 0)) {

persona = colaespecial\_2.devolverelemento();

colaGeneral.sacarelemento();

lista\_de\_atendidos.agregaralfinal(persona);

imprimirCliente(persona);

}

}

template<typename B>

void Banco<B>:: cerrarColaEsp() {

int opcion;

if ((colaespecial\_1.filavacia() == true) || (colaespecial\_2.filavacia() == true)) {

cout << "1-Cola especial 1" << endl;

cout << "2-Cola especial 2" << endl;

cin >> opcion;

if (opcion == 1) {

if (colaespecial\_1.cantidadelementos() != 0) {

cout << "no se puede cerrar porque tiene personas sin atender" << endl;

}

else {

colaespecial\_1.cerrar();

}

}

else { if (opcion == 2) {

if (colaespecial\_2.cantidadelementos() != 0) {

cout << "no se puede cerrar porque tiene personas sin atender" << endl;

}

else {

colaespecial\_2.cerrar();

}

}

}

}

else cout << "no hay colas para cerrar" << endl;

}

template<typename B>

void Banco<B>:: criterios(Fila<Cliente> & cola){

string operacion;

bool esCliente, verificacion;

int opcion;

do { cout << "Ingrese el numero de la operacion: " << endl;

cout << "1-Retiro" << endl;

cout << "2-Deposito" << endl;

cout << "3-Transferencia" << endl;

cout << "4-Pago" << endl;

cin >> opcion;

switch (opcion) {

case 1: operacion = "Retiro";

verificacion = true;

break;

case 2: operacion = "Deposito";

verificacion = true;

break;

case 3: operacion = "Transferencia";

verificacion = true;

break;

case 4: operacion = "Pago";

verificacion = true;

break;

}

} while (verificacion == false);

verificacion == false;

cola.criterio(operacion);

do {cout << "Es cliente?" << endl;

cout << "1-si" << endl;

cout << "2-no" << endl;

cin >> opcion;

switch (opcion) {

case 1: esCliente = true;

verificacion = true;

break;

case 2: esCliente = false;

verificacion = true;

break;

}

} while (verificacion == false);

cola.criterioCl(esCliente);

}

template<typename B>

void Banco<B>:: listaDeOp(){

int montoMin, montoMax;

Cliente usuario;

cout << "Ingrese monto minimo: " << endl;

cin >> montoMin;

cout << "Ingrese monto maximo: " << endl;

cin >> montoMax;

int suma = 0;

int cant = lista\_de\_atendidos.cantelem();

int contador=0;

int cantidad=0;

while (contador < cant) {

if ((lista\_de\_atendidos.getDato().muestramonto() >= montoMin) && (lista\_de\_atendidos.getDato().muestramonto() <= montoMax)) {

imprimirCliente(lista\_de\_atendidos.getDato());

suma += lista\_de\_atendidos.getDato().muestraedad();

cantidad ++;

}

contador ++;

lista\_de\_atendidos.agregaralfinal(lista\_de\_atendidos.getDato());

lista\_de\_atendidos.eliminaralinicio();

}

if (cantidad != 0) {

cout << suma/cantidad << endl;

}

}

template<typename B>

void Banco<B>:: acomodarclientes() {

int cant=0;

int contador = 0;

Cliente usuario;

cant = colaGeneral.cantidadelementos();

while (contador < cant) {

usuario = colaGeneral.devolverelemento();

cout << usuario.muestranom() << endl;

if ((usuario.muestraop() == colaespecial\_1.getoperacion()) && (usuario.muestraCl() == colaespecial\_1.getcliente())){

colaGeneral.sacarelemento();

colaespecial\_1.agregar(usuario);

contador ++;

}

else {

if ((usuario.muestraop() == colaespecial\_2.getoperacion()) && (usuario.muestraCl() == colaespecial\_2.getcliente())){

colaGeneral.sacarelemento();

colaespecial\_2.agregar(usuario);

contador ++;

}

else {

colaGeneral.sacarelemento();

colaGeneral.agregar(usuario);

contador ++;

}

}

}

}

template<typename B>

void Banco<B>:: abrirColaEsp(){

if (colaespecial\_1.getcolaabierta() == false) {

colaespecial\_1.abrir();

criterios(colaespecial\_1);

cout << colaespecial\_1.getoperacion() << endl;

cout << colaespecial\_1.getcliente() << endl;

acomodarclientes();

cout << "Se creo la cola especial 1 nueva" << endl;

}

else { if (colaespecial\_2.getcolaabierta() == false) {

colaespecial\_2.abrir();

criterios(colaespecial\_2);

acomodarclientes();

cout << "Se creo la cola especial 2 nueva" << endl;

}

else cout << "Las colas especiales ya estan habilitadas" << endl;

}

}

template class Banco<Cliente>;

#include <iostream>

#include <conio.h>

#include "Banco.h"

#include "Cliente.h"

#include "lista.h"

#include "Fila.h"

using namespace std;

void Menu() {

int opcion;

Cliente nuevo\_usuario;

bool Salir = false;

Banco<Cliente> Banco;

while (Salir == false) {

cout << "Usted ingreso a Banco" << endl;

cout << endl;

cout << "Ingrese una opcion:" << endl;

cout << "1-Mesa de entrada" << endl;

cout << "2-Atender proximo cliente" << endl;

cout << "3-Abrir cola especial" << endl;

cout << "4-Cerra cola especial" << endl;

cout << "5-Lista de operaciones" << endl;

cout << "6-Salir" << endl;

cin >> opcion;

switch (opcion) {

case 1: system("CLS");

Banco.ingresarCliente();

break;

case 2: system("CLS");

Banco.atenderCliente();

break;

case 3: system("CLS");

Banco.abrirColaEsp();

break;

case 4: system("CLS");

Banco.cerrarColaEsp();

break;

case 5: system ("CLS");

Banco.listaDeOp();

break;

case 6: Salir = true;

break;

default: system("CLS");

break;

}

}

}

int main()

{

Menu();

return 0;

}