2da Parte

29/06/2022

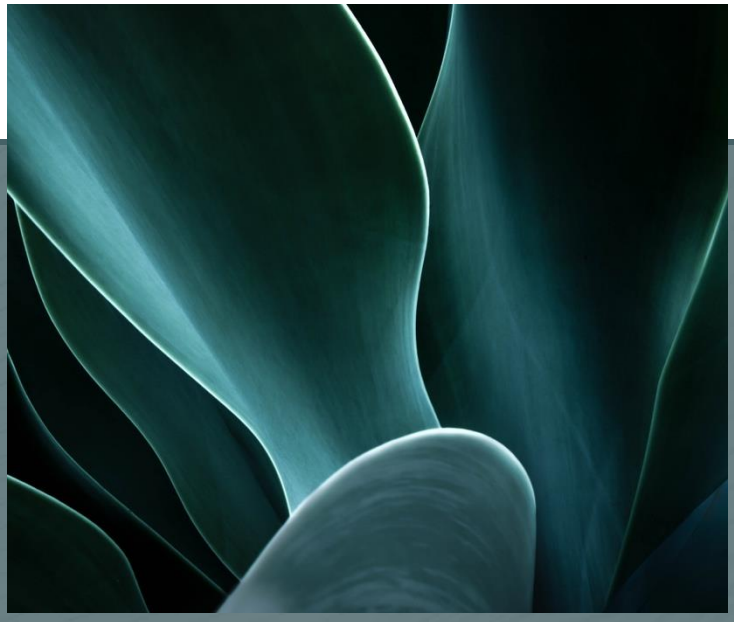
—

Análisis y Diseño de Algoritmos

—

Introducción

El trabajo consiste en la implementación de las clases creadas en la primera parte, en una problemática planteada por la catedra, utilizar nuestras capacidades cognitivas y habilidades para poder realizar una simulación de una entidad bancaria, creando colas de espera, centros de atención, recopilación de datos y listar conceptos excluidos. Nos permite la agilización de nuestra habilidad de codificación.



Objetivos

1. **Objetivo Principal: El objetivo de la segunda parte del trabajo práctico especial es simular, de forma muy simplificada, la llegada, espera y atención de los clientes durante su operatoria en un banco.**

2. **Creación de una clase especifica de clientes para la próxima recopilación de datos acerca del usuario.**

3. **La utilización de un menú para que el usuario pueda decidir sobre la ejecución del programa y los datos recopilados**

4. **Para cada una de las funciones especificadas dar su complejidad temporal.**

Desarrollo

Descripción del Programa

Cuando un cliente ingresa al banco es atendido por la mesa de entrada donde se le solicitan los siguientes datos: nombre completo, edad, operación a realizar (Retiro, Depósito, Transferencia o Pago), tipo de destinatario de la operación (persona, banco o impuesto), monto aproximado de la operación y si es cliente del banco. A partir de esa información se le ubicará en la cola que corresponda, para su posterior atención, considerando el orden de llegada. El banco cuenta con una única caja para atender a todos los clientes. Inicialmente, se encuentra abierta una cola común donde todos los clientes esperarán su turno de acuerdo al orden de llegada. Luego, además de esta cola general, se podrán habilitar (a lo sumo) dos colas especiales para atención. Cada cola especial agrupa a los clientes según un criterio combinado: la operación que van a realizar y si poseen o no cuenta en el banco. Habilitar una nueva cola especial implica reubicar los clientes de la cola común que cumplen con el criterio establecido; respetando, claro está, el orden de llegada de los mismos. Si ya se encuentra habilitada alguna de las colas

especiales, los nuevos clientes que ingresen al banco deberán ubicarse en la cola correspondiente (se trate de colas especiales o la cola por orden de llegada) Al finalizar el día, el banco elabora un listado selectivo de operaciones diarias según un rango de montos determinado. Este listado debería visualizar toda la información solicitada por la mesa de entrada. *(copiado de la consigna dada por catedra)*

Implementación de Clases

Clase Clientes:

TDA simple conformado por los siguientes elementos:

- Nombre: string;
- Edad: integer;
- Operación: integer (utilizamos integer para simplificar la comparación en el código principal).
- Destinatario: integer (por la misma razón que operación).
- Monto: integer;
- Pertenece: boolean (guarda si el cliente pertenece o no al banco, lo cual le dará prioridades en las colas auxiliares).

Con sus correspondientes operaciones de retorno las cuales devuelven los datos solicitados (por ejemplo, MontoCliente retorna el monto de la clase cliente).

Al tener solamente declaraciones la clase completa pertenece a O (1).

Clase ListaSimple:

(n = cantidad de elementos)

Es un modificado de ListaSimple adaptado a nuestras necesidades, este ya no es abstracto, ya que sabemos que va a ser una lista exclusivamente de Clientes.

Utilizamos una lista y no una fila ya que la consigna requiere recorrer la estructura en varias ocasiones y utilizar elementos que no están en el último acceso.

Explicación de procedimientos nuevos:

void ListaSimple::BorrarLista(Cliente dato)

Ahora borrar recorre hasta encontrar un cliente que contenga el mismo nombre que el nodo en el que estoy parado. Se podría mejorar ya que podemos encontrar (muy raro) dos clientes con el mismo nombre. Pertenece a O(n) ya que recorre como peor caso toda la lista.

int ListaSimple::promedio() const

Este procedimiento se utiliza solamente en la última opción (explicada próximamente) la cual pide el promedio de edad de una lista. Pertenece a O(n) ya que recorre toda la lista.

Clase Arbin:

Modificado de Arbin dedicado solamente a datos del tipo Cliente. Utilizamos este tipo para guardar los Clientes atendidos ya que nos pareció más eficiente recorrer según el monto debido a la última opción.

Explicación de procedimientos nuevos:

```
void Arbin::recorrerP(ListaSimple & listamonto,int MMin,int MMax,NodoArbin * Arbol) const{

void Arbin::recorrer(ListaSimple & listamonto,int MMin,int MMax) const
```

Esta operación modifica una lista dada agregando en ella los nodos del árbol que apliquen a un rango de montos dado. Pertenece a O(n/2) ya que no recorre todo el árbol (suponemos que este está balanceado).

Programa Principal:

Main:

```
int main()
```

Carga de el arreglo puertas (guarda valores a utilizar en las filas auxiliares), el árbol de los clientes que ya fueron atendidos, y las colas (Principal, Aux1 y Aux2). Llama a menú.

Menú:

Es el procedimiento principal el cual le permite al usuario acceder a las 6 opciones que el programa nos deja realizar.

```
void Menu(ListaSimple Principal,ListaSimple Aux1,ListaSimple Aux2,Arbin Atendidos, int puertas[])

void LimpiarPantalla()
```

LimpiarPantalla es solamente un procedimiento que borra los caracteres de la pantalla del usuario para una mejor estética. O(1)

Se escoge una opción a realizar y se llama al procedimiento correspondiente. Si el usuario desea realizar otra acción se utiliza recursión. La complejidad depende de las opciones que utilicemos.

Opción 1: Agregar cliente a la cola.

```
void AgregarCliente(ListaSimple & Principal,ListaSimple & Aux1,ListaSimple & Aux2, int puertas[]){

void AgregarClienteFila(Cliente NuevoCliente, ListaSimple & Principal,ListaSimple & Aux1,ListaSimple & Aux2,int puertas[])
```

AgregarCliente es el procedimiento principal donde le pedimos al cliente que ingrese

su nombre para guardarlo en un string, la edad para guardarla en un integer. También le damos 4 opciones de operaciones que puede realizar, estas son 1-retiro, 2-deposito, 3-transferencia y 4-pago, el cliente ingresa el numero de la operación correspondiente para poder guardarla en un integer ya que es un número. Le pedimos que ingrese el destinatario para lo cual le damos 3 opciones la cuales son 1-persona, 2-banco y 3-impuesto, guardamos el dato en un integer así también como lo hacemos con el monto y por último le preguntamos al cliente si pertenece o no al banco, para esto le damos 2 opciones de las cuales 1 sola vamos a guardar en el tipo de dato char. Todos los datos guardados los pasamos a "Nuevocliente", que genera un cliente con todos los datos pedidos. Por último, el procedimiento llama a AgregarClienteFila donde agregamos el cliente a la fila correspondiente, si la fila auxiliar-1 está abierta, entonces al cliente lo ingresamos a la fila auxiliar-1 si cumple con los requisitos de la misma. Si no cumple con los requisitos de las fila auxiliar-1 y la fila auxiliar-2 está abierta, al cliente se lo ingresa en este si cumple con los requisitos de la fila, de lo contrario si no se ingresó en ninguna de las filas anteriormente mencionadas, al cliente se lo ingresa en la fila principal.

AgregarClienteFila tiene complejidad O(n) (n= cantidad de elementos que contenga la fila) y AgregarCliente también por herencia de la anterior.

Opción 2: Agregar fila auxiliar.

```
void AgregarFilaAux(ListaSimple & Principal,ListaSimple & Aux1,ListaSimple & Aux2,int puertas[]){

void ChequeoPrincipal(ListaSimple & Principal,ListaSimple & Aux, int puerta){
```

AgregarFilaAux es el procedimiento principal, al usuario se le pide que ingrese por teclado un numero correspondiente con la opción de operación que la fila auxiliar utilizara como criterio para que un cliente se le pueda ingresar. Estas opciones son 1-retiro, 2-deposito, 3-transferencia y 4-pago. La información solicitada al usuario se guarda en integer. Luego si ninguna fila auxiliar está abierta, se abre la fila auxiliar-1 con el criterio que haya elegido el usuario. Luego se llama a ChequeoPrincipal donde se revisa la fila principal por si algún cliente cumple con los criterios de la fila que recién se abrió, si los cumple se los transfiere a la fila auxiliar-1. Si ya se tiene una fila auxiliar abierta al momento de abrir una fila auxiliar, se abre la segunda fila auxiliar con el criterio solicitado al usuario y se transfieren los clientes que cumplan con tal criterio de la fila principal a la auxiliar recién abierta. En caso de que ya se hayan abierto 2 filas auxiliares y se desee abrir otra, el usuario no va a poder y el programa imprimirá "todas las filas auxiliares están ocupadas".

ChequeoPrincipal corresponde a O(n) (n=cantidad de elementos que contenga Principal) y AgregarFilaAux corresponde a O(n) por herencia de la anterior.

Opción 3: Mostrar filas.

```
void MostrarFilas (ListaSimple Principal,ListaSimple Aux1,ListaSimple Aux2, int puertas[]){
```

void MostrarFila(ListaSimple fila)

void MostrarCliente(Cliente aux1,int i)

en MostrarFilas se verifica para cada fila si la fila estaba vacía o no, si la cola no estaba vacía se llama a MostrarFila donde cada nodo de la cola se llama a MostrarCliente, que consiste en mostrar por pantalla el número de cliente en la fila de espera, el nombre, la edad, la operación a realizar, quien es el destinatario de la operación a realizar, el monto aproximado de la operación y si pertenece o no al banco.

MostrarCliente pertenece a O (1) solamente declaraciones constantes, MostrarFIla pertenece a O(n) (n=cantidad de elementos Principal + Aux1 + Aux2) y MostrarFilas pertenece a O(n) por herencia de la anterior.

Opción 4: Atender cliente.

void MostrarCliente(Cliente aux1,int i)

void AtenderFila(ListaSimple & Lista, Arbin & Atendidos)

void AtenderCliente(ListaSimple & Principal,ListaSimple & Aux1,ListaSimple & Aux2,int puertas[],Arbin & Atendidos)

AtenderCliente le pide al usuario que ingrese por teclado un número que corresponda con la fila que desea atender (las cuales son Cola principal, auxiliar-1, auxiliar-2), si la cola que el usuario quiere atender no está vacía o inexistente, se llama a AtenderFila que muestra al cliente por pantalla con MostrarCliente y agrega al cliente a un árbol ordenado por monto (Atendidos) donde se encuentran todos los clientes atendidos hasta el momento. Por último, borra al cliente de la fila.

Si la fila está vacía o es una fila auxiliar que no se abrió, el programa imprime por pantalla “fila vacía o inexistente”.

AtenderFila pertenece a O(n) (n=cantidad de elementos de la cola correspondiente) por BorrarLista y AtenderCliente O(n) por herencia de la anterior.

Opción 5: Cerrar cola auxiliar.

void CerrarColaAux(ListaSimple Principal,ListaSimple Aux1,ListaSimple Aux2, int puertas[])

En este procedimiento donde cerramos la fila que el usuario desea cerrar, se le pregunta a este si desea cerrar fila auxiliar-1 o la fila auxiliar-2, luego se le verifica si la cola está o no vacía, si lo esta se procede a cerrar la cola e imprimir por pantalla que la cola que deseo cerra se encuentra cerrada, ¡si la fila no está cerrada el programa imprime por pantalla “la lista que desea cerrar no está vacía!”.

Pertenece a O(1), solo declaraciones y constantes.

Opción 6: Listar operaciones atendidas por monto.

void MostrarFila(ListaSimple fila)

void ListarOppMonto (Arbin Atendidos)

ListarOppMonto consiste en recorrer el árbol donde se encuentran todos los clientes anteriormente entendidos y mediante se recorre el árbol se verifica que si el los clientes por los cuales se recorre tienen un monto dentro del rango anteriormente ingresado por el usuario, si el cliente atendido se encuentra dentro del rango de monto mínimo monto máximo, se procede a ingresar al nodo en una lista en la cual los nodos de la misma solo entran dentro del rango ingresado por el usuario, luego se procede a imprimir la lista con MostrarFila cuando se terminó de recorrer el árbol. La lista se imprime recorriendo la misma al mismo tiempo que se imprime cada cliente por pantalla, cuando se termina de recorrer el programa también da por pantalla el promedio de edad de los clientes atendidos.

ListarOppMonto pertenece a O(n) (n=cantidad de elementos Principal + Aux1 + Aux2) por herencia de MostrarFila.

Conclusión:

gracias a esta nueva forma de codificar aprendida durante el desarrollo de la cursada, (a comparación de lo que antes conocíamos), nos permitió el poder realizar estructuras más eficientes y con menor carga de memoria. Aun así, esta reducción en la complejidad y uso de memoria, se ve equilibrada por el elevado costo que conlleva construirlas. Se podría afirmar entonces, que la implementación de este tipo de estructuras y algoritmos son las practicas más acertadas para ser lo más eficiente posible en cualquier código futuro.