TRABAJO PRÁCTICO ESPECIAL

Análisis y diseño de algoritmos I

Grupo Nº: 13

Integrantes:

Ferraro, Bautista

DNI: 43909230

Mail: Bautistaferraro82@gmail.com

Goicoechea, José María

DNI: 43658906

Mail: joegoico@gmail.com

Ayudante:

Objetivos (Primera parte):

La primera parte consiste en especificar e implementar los tipos de datos abstractos básicos: Fila, Lista y Árbol Binario en C++. Estos deberán cumplir un conjunto mínimo de funciones entre las que se encuentran:

Fila:

- agregarFila: Agrega un elemento a la fila.
- vaciaFila: Consulta si la fila posee algún elemento, de no poseer ninguno devuelve true.
- recuperarFila: Devuelve el valor del próximo elemento a retirar de la fila (sin retirarlo).
- cantidadElementos: Consulta la cantidad de elementos que contiene la fila.
- eliminarFila: Retira el elemento más próximo a salir de la fila.

Lista:

- agregarPrincipioLista: Agrega un elemento al principio de la lista.
- agregarFinalLista: Agrega un elemento al final de la lista.
- agregarListaOrdenado: Agrega un elemento a la lista según el criterio de orden dado.
- cantElementos: Consulta la cantidad de elementos que posee la lista.
- pertenece: Verifica si un elemento está o no en la lista.
- listaVacia: Consulta si la lista posee algún elemento, de no poseer ninguno devuelve true.
- eliminar: Elimina un elemento de la lista.

Árbol Binario:

- agregarArbol: Agrega un elemento al árbol.
- arbolVacio: Verifica si el árbol está vacío.
- perteneceArbol: Consulta si un elemento pertenece al árbol.
- cantidadElementos: Consulta la cantidad de elementos que contiene el árbol.
- profundidadArbol: Consulta la profundidad del árbol, considerando la raíz como profundidad=1.
- listarFrontera: Lista los elementos que forman la frontera del árbol.
- listarInOrder: Lista los elementos del árbol en InOrder.

Especificación formal de los tipos de datos abstractos en Nereus de la parte sintáctica

TDA FILA

CLASS Fila[Elemento]

IMPORTS Natural

BASIC CONSTRUCTOR inicFila, agregarFila

EFFECTIVE

TYPE Fila

OPERATIONS

inicFila-> Fila; //Constructora ∈ $O(1)$

agregarFila: Fila*Elemento->Fila; //Constructora ∈ $O(n)$

vaciaFila: Fila -> Boolean; //Observadora ∈ $O(1)$

recuperarFila: Fila(f) -> Elemento //Observadora ∈ $O(1)$

pre: not vaciaFila (f);

cantidadElementos: Fila(f) -> Natural //Observadora ∈ $O(1)$

pre: not vaciaFila (f);

eliminarFila: Fila(f) -> Fila //Modificadora ∈ $O(n)$

pre: not vaciaFila(f);

END-CLASS

TDA LISTA

CLASS Lista[Elemento]

IMPORTS Natural

BASIC CONSTRUCTOR inicLista, agregarLista

EFFECTIVE

TYPE Lista

OPERATIONS

inicLista->Lista; //Constructora ∈ $O(1)$

listaVacia: Lista->Boolean; //Observadora ∈ $O(1)$

agregarPrincipioLista: Lista(l)*Elemento->Lista; //Constructora ∈ $O(1)$

pre: not listaVacia (l);

agregarFinalLista: Lista(l)*Elemento->Lista; //Constructora ∈ $O(n)$

pre: not listaVacia (l);

agregarOrdenadoLista: Lista(l)*Elemento*Natural(n)->Lista;

pre: (n>=1) and (n<=cantElementos(l)+1);//Constructora ∈ $O(n)$

cantElementos: Lista(l)-> Natural //Observadora ∈ $O(n)$

pre: not listaVacia (l);

pertenece: Lista(l)*Elemento->Boolean //Observadora ∈ $O(n)$

pre: not listaVacia (l);

eliminar: Lista(l)*Natural(n)-> Lista //Modificadora ∈ $O(n)$

pre: (n>=1) and (n<=cantElementos(l));

END-CLASS

TDA ARBOL

CLASS Árbol[Elemento]

IMPORTS Natural, Lista

BASIC CONSTRUCTOR inicArbol, agregarArbol

EFFECTIVE

TYPE Árbol

OPERATIONS

inicArbol->Árbol; //Constructora ∈ $O(1)$

agregarArbol: Árbol*Árbol*Elemento->Árbol //Constructora ∈ $O(n)$

arbolVacio: Árbol->Boolean; //Observadora ∈ $O(1)$

perteneceArbol: Árbol(a)*Elemento->Boolean //Observadora ∈ $O(n)$

pre: not arbolVacio(a);

cantidadElementos: Árbol(a) ->Natural //Observadora ∈ $O(1)$

pre: not arbolVacio(a);

profundidadArbol: Árbol(a)-> Natural //Observadora ∈ $O(n)$

pre: not arbolVacio(a);

listarFrontera: Árbol(a)*Lista->Lista //Observadora ∈ $O(n)$

pre: not arbolVacio(a);

listarInOrder: Árbol(a)*Lista->Lista //Observadora ∈ $O(n)$

pre: not arbolVacio(a);

END-CLASS

Objetivos (segunda parte):

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

Desarrollo:

Para la implementación del trabajo decidimos crear un menú, desde el cual se maneja todo el programa. Teniendo en este 5 opciones para que el usuario decida cual usar en cada momento. Una vez efectuada una opción el programa le preguntara al usuario si desea seguir por alguna otra opción o si desea finalizar.

Cada cliente que ingresa al banco, al pasar por la mesa de entrada le serán consultados todos los datos que pide la consigna (nombre completo, edad, operación a realizar, destinatario, monto, y si es cliente del banco). A partir de ahí el usuario decidirá como desea seguir en base a una de las 5 opciones del menú, entre las que se encuentran: ingreso de nuevo cliente, atender cliente, abrir cola especial, cerrar cola especial y por ultimo listar las operaciones históricas según un rango de montos decidido por el usuario, explicitando al final el promedio de edad de los clientes que conforman el listado.

Los algoritmos utilizados son:

Tomardatos: esta función es utilizada cuando el usuario elige la opción 1(ingresar nuevo cliente), y su finalidad es preguntar al cliente todos los datos necesarios para posteriormente ser ingresado en la fila correspondiente y en la lista de atendidos históricos. $\in O(n)$

Agregrarcliente: este método agrega al cliente con todos los datos previamente solicitados a la lista de atendidos históricos, ordenada de menor a mayor por monto de la operación a realizar. $\in O(n)$

Ingresar: este procedimiento agrega al cliente a la fila general, desde la cual puede, o ser atendido, o ser derivado a alguna de las filas especiales (las cuales deben haber sido previamente creadas por el usuario al elegir la opción 3 del menú). $\in O(n)$

Atendercliente: este método elimina al cliente que se encuentra en la primera posición de la fila dada. ∈ O(n)

Atendercliente: este procedimiento pregunta al usuario de que fila desea atender un cliente, mostrándole las filas abiertas entre las que estará la general y puede llegar a estar una o dos especiales previamente creadas por el usuario. Luego llama al método correspondiente para hacerlo. ∈ O(n)

Abrircola: este procedimiento verifica que no haya más de dos filas especiales abiertas, y en caso de no haberlas crea una nueva fila llamando a los métodos correspondientes según el criterio dado por el usuario. ∈ O(1)

Getfilaabierta: este método booleano devuelve verdadero o falso si la fila solicitada está abierta o no respectivamente. ∈ O(1)

Abrirfilaespecial: este método abre una nueva fila según el criterio dado por el usuario. ∈ O(1)

Cerrarcola: este procedimiento muestra al usuario si existe alguna cola especial abierta y le da la posibilidad de cerrar la que el quiera. ∈ O(n)

Filavacia: este método booleano devuelve true si la fila solicitada esta vacía, en caso contrario devolverá false. ∈ O(1)

Listaroperaciones: este procedimiento solicita al usuario un monto mínimo y un monto máximo, entre los cuales se recorrerá la lista de atendidos históricos y se imprimirán todos los datos de los clientes que cumplan con el rango de montos. Además, al finalizar la impresión de los clientes en pantalla se explicitara el promedio de edad de los mismos.

Aclaraciones

Al comenzar con el trabajo nos inclinamos por utilizar un árbol binario para tener todos los atendidos históricos ordenados por monto de manera más eficiente que en una lista simple. Pero al seguir con el trabajo, nos dimos cuenta de que sería mejor utilizar una lista simple debido a que todos los procedimientos en los que usemos el árbol iban

a ser recursivos. Por lo tanto al utilizar una lista simple se reduciría la complejidad temporal del programa.

Por otro lado queríamos explicar que no pudimos realizar la opción 5(listar operaciones según rango de montos) ya que nos dijeron que no podíamos imprimir cosas en un método y eso nos complico mucho la resolución. El método de todas formas esta implementado y en la parte en la que nos trabamos pusimos un comentario “imprimir cliente”, dejando en claro que en esa línea del código es donde pretendíamos que se imprimieran en pantalla todos los datos del cliente.

Conclusión

Al comenzar el trabajo no teníamos muy claro el manejo de las clases y los objetos en c++. Hoy podemos decir que mejoramos mucho nuestros conocimientos en el lenguaje y además nuestra capacidad para resolver problemas mediante el código. Ambos estamos muy conformes con nuestro trabajo y esperamos que este a la altura de lo solicitado.