EIE
Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

UNIVERSIDAD DE COSTA RICA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA ELÉCTRICA

ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA

LABORATORIO 4: LISTAS

ESTUDIANTES:
JESÚS ZÚÑIGA MÉNDEZ (B59084)
DENNIS CHAVARRÍA SOTO (B82097)

PROFESOR:
RICARDO ROMÁN BRENES; M. SC.

II CICLO 2019

Índice

Índice de figuras

1. Reseña del programa

Las estructuras de datos son un elemento importante en sistemas de comunicación en todo el mundo, dado que, en la perspectiva más simple, permiten organizar datos de diferentes maneras, sin embargo, es necesario buscar implementaciones que sean eficientes y prácticas para el usuario.

Las estructuras de datos vistas en clase abarcan dos ejemplos principales, los arreglos y las listas enlazadas (Las cuales puede ser doble o simplemente enlazadas). Las diferencias entre ellos pueden ser los beneficios o deficiencias respecto al otro, debe considerarse que, una ventaja de las listas enlazadas es que funcionan como bloques, que, dado una dirección de memoria (para el caso del lenguaje c++, que es el utilizado para este laboratorio) se pueden referir a bloques próximos que almacenan más datos. En este caso, la memoria puede no ser contigua. Los arreglos son más sencillos de recorrer, pues, la memoria es contigua, es decir los espacios que almacenan los datos en él, generalmente están al lado del otro, o, mejor dicho, en un bloque común.

Es fundamental que los estudiantes comprendan como funcionan las listas enlazadas, así como métodos para recorrerlas, ordenarlas y comparar elementos de ellas. Así, pues, este laboratorio tiene como finalidad la implementación y análisis de la complejidad de algunos algoritmos de búsqueda, así como crear dos clases que hereden de una clase abstracta Lista; sus hijas serán el array y la lista enlazada, que heredan sus metodos virtuales.

2. Funcionamiento del programa

El programa se desarrolló con el lenguaje de programación C++, se implementó la clase List, que tiene los métodos virtuales base de cada estructura de datos. También se emplantilló las clases y lo métodos de búsqueda y ordenamiento, para hacerlos utilizables con variedad de tipos de datos. Entre otras clases y archivos se encuentra:

2.1. Clase List

Esta clase, ubicada en el archivo List.h; incluye los siguientes metodos virtuales publicos:

- List (): Constructor por defecto de lista
- List (const List orig): Constructor por copia de lista
- virtual List (): Destructor de las estructuras de datos
- Virtual void emptyList (): Se utiliza para borrar los elementos de la estructura de datos
- virtual void insert (Data , Position): Metodo que se implementa para guardar un dato en una posicion especifica.
- virtual void insert (Data): Metodo que inserta un elemento en la posicion final de la estructura de datos. Su parámetro es el dato.

- virtual void delete (Data): Este metodo busca una serie de elementos en la estructura de datos y los elimina. Su parámetro es el dato.
- virtual void delete (Position): Este metodo busca una posición de la serie de elementos en la estructura de datos y la elimina. Su parámetro es la posición.
- virtual Data getElement (Position): Solicita el dato que se encuentra en determinada posición de la estructura de datos. Su parámetro es la posición.
- virtual Position find (Data): Encuentra la posición de un elemento específico que el usuario ingresa. Su parámetro es el dato.
- virtual Position next (Position): devuelve el elemento que se encuentra en la posición siguiente a la que el usuario ingresa. Su parámetro es la posición.
- virtual Position prev (Position): devuelve el elemento que se encuentra en la posición anterior a la que el usuario ingresa. Su parámetro es la posición.
- virtual void print (): Método genérico que imprime los elementos de la estructura de datos.

Tambien cuenta con un atributo privado que indica la cantidad de items que posee.

2.2. Clase Arraylist

El ArrayList, encontrado en el archivo Arraylist.h corresponde a la clase que trabaja con un arreglo dinámico. Hace uso de memoria dinámica, para poder aumentar el tamaño del mismo. Como se mencionó anteriormente, está emplantillado.

2.2.1. Constructor por defecto

Para el constructor por defecto, se define un nuevo espacio de memoria de 3 casillas, apuntado por el puntero *arreglo* para el arreglo, luego *numero-items* del arreglo sera igual que 3, osea la cantidad maxima de items con los que puede contar hasta el nuevo redimensionamiento, luego cuenta con una variable de tipo int (Igual que numero-items) que se llama *items-activos*, esta indica la cantidad de items accesibles y es util para los ciclos que implican busqueda y borrado de elementos.

- resizer(): No tiene parámetros específicos. Toma un puntero arreglo-temporal de un tamaño mayor que el llamado arreglo, luego le copia los datos de este último y le hace apuntar al nuevo espacio. La copia de datos se logra con un ciclo for.
- void emptyList(): Borra los datos, asignando nuevo espacio de memoria y cambiando la variable items-activos a 0
- reorder(Data data, Position position): Se define una variable temporal que se utiliza para almacenar el dato que se desea salvar hasta la sustitución de las casillas. La finalidad de este metodo es reordenar los elementos cuando se agrega uno en medio de todos.
- reorder$_{inv}(Datadata, Positionposition) : Sedefineunavariabletemporalqueseutilizaparaalmacenareldatoq$
  $Utilizalavariableitems-activosparasaberencualposiciondebeirelultimoelemento.Suparámetroeselvalorqu$

- void insert(Data data, Position position): Utiliza data y position, el primero corresponde a lo que se desea guardar, el segundo la posicion en la quede debe ir el elemento. Su parámetro es el valor que desea agregar y su posicion
- void insert(Data data, Position position): Utiliza data y position, el primero corresponde a lo que se desea guardar, el segundo la posicion en la quede debe ir el elemento. Su parámetro es el valor que desea agregar y su posicion.
- void remove(Data data): Quita los elementos coincidentes con el parámetro data, luego lo reacomoda, y reduce la cantidad items-activos.
- void remove(Position position): Quita el elemento coincidente con el parámetro de posición, luego lo reacomoda, y reduce la cantidad items-activos.
- Data getElement(Position position): Retorna el elemento que corresponde a la posición dada por el parámetro.
- Position find(Data data): Se pasa un valor data y se devuelve la posicion en la que se encuentra. Para ello debe iterar por cada elemento, tal y como la busqueda linal. Si no se encuentra, devuelve un mensaje.
- Position next(Data data): Se pasa un valor data y se devuelve la del siguiente elemento. Usa un puntero para navegar entre los elementos y buscar el indicado
- Position next(Data data): Se pasa un valor data y se devuelve la del anterior elemento.Usa un puntero para navegar entre los elementos y buscar el indicado.
- void print(): Consiste en un ciclo for que itera y navega por cada elemento del arreglo, cuando lo hace, lo imprime. No usa parámetros adicionales.

La estructura de este método consiste en un conjunto de ciclos anidados de tipo ”for”. El primero de ellos itera por cada fila, luego uno interno itera por columna, seguido hay uno que repite tres veces, para cambiar el color; este tiene la particularidad de que su valor de repetición se utiliza en los vectores para asignar el los promedios iprom a una posición determinada.

2.3. Clase SingleLinked

Clase correspondiente a la lista simplemente enlazada. Cuenta con una variable de tipo int llamada encontado.

2.4. Constructor por defecto

No tiene parámetros específicos. Toma un puntero hacia una posición de memoria en la que se encuentra una primera posición un una vieja. Instancia un puntero de tipo SimplePosition que guarda los valores de position siguiente, osea la proxima casilla de la lista, así como el dato que se desea guardar, esto por medio de un puntero. Considérese que se tomarán los tipos igual que las plantillas proporcionadas por el profesor, con la finalidad de mantener la legibilidad.

- void emptyList(): Borra los datos, asignando nuevo espacio de memoria y cambiando la variable items-activos a 0

- actualizar items(): Se utiliza para reordenar los datos de la lista.
- SinglePosition * ultimo () se encarga de asignar a una dirección de memoria que se apunta actualmente a una próxima. Para así mantener continuidad.
- insert(Data data): Utiliza la variable items-activos para saber en cual posicion debe ir el ultimo elemento. Su parámetro es el valor que desea agregar.
- void insert(Data data, Position position): Utiliza data y position, el primero corresponde a lo que se desea guardar, el segundo la posicion en la quede debe ir el elemento. Su parámetro es el valor que desea agregar y su posicion
- void insert(Data data, Position position): Utiliza data y position, el primero corresponde a lo que se desea guardar, el segundo la posicion en la quede debe ir el elemento. Su parámetro es el valor que desea agregar y su posicion. Utiliza un puntero para reasignar la dirección del próximo nodo que sucede al que se agrega
- void remove(Position): Con ayuda de 3 condicionales determina si se encuentra el primer elemento, el intermedio o el ultimo. Y asigna las direcciones del siguiente nodo.
- void remove (Dato dato): Quita un determinado elemento; depende de punteros temporales llamados actual y respaldo. Con ayuda de 3 condicionales determina si se encuentra el primer elemento, el intermedio o el ultimo. Y asigna las direcciones del siguiente nodo.
- void remove(Position position): Quita un determinado elemento que se encuentra de una posición especificada; depende de punteros temporales llamados actual y respaldo. Con ayuda de 3 condicionales determina si se encuentra el primer elemento, el intermedio o el ultimo.
- Data getElement(Position position): Retorna el elemento que corresponde a la posición dada por el parámetro. Depende de una variable encontrado para saber si coinciden los resultados y así devolver una respuesta de resultado encontrado o no encontrado.
- Position find(Data data): Se pasa un valor data y se devuelve la posicion en la que se encuentra. Para ello debe iterar por cada elemento *como si saltara* pues debe pasar de nodo a nodo por medio de los punteros, tal y como la busqueda linal.
- Position next(Data data): Se pasa un valor data y se devuelve la del siguiente elemento. Usa un puntero para navegar entre los elementos y buscar el indicado. Depende de un valor de encontrado para saber si debe continuar o no.
- Position next(Data data): Se pasa un valor data y se devuelve la del anterior elemento.Usa un puntero para navegar entre los elementos y buscar el indicado. Retorna el valor encontrado en tal posicion.
- void print(): Consiste en un ciclo for que itera y navega por cada elemento del arreglo, cuando lo hace, lo imprime. No usa parámetros adicionales, pero considérese que al momento de imprimir detalla la posición, el puntero y el contenido del nodo.

2.5. Algoritmo de busqueda linea

Este itera por cada elemento y una vez hecho eso, hasta encontrar el indicado por el usuario.

2.6. Algoritmo de ordenamiento BubbleSort

Se encarga de comparar entre dos elementos de la estructura de datos, luego, si el primero es mayor que el segundo, los intercambia, hace eso para cada elemento y, cuando no hayan ocurrido cambios, se rompe un ciclo while y la lista esta ordenada.

2.7. Algoritmo de ordenamiento Mergesort

Este algoritmo divide la lista en muchas partes pequeñas para agruparlas y ordenarlas poco a poco. Este algoritmo fue implementado para listas enlazadas, pero presenta fallos.

3. Conclusiones

- Debido a la complejidad de los algoritmos de busqueda, no fue posible implementarlos todos ni realizar el respectivo analisis de complejidad, pues, la implementacion de unos requirio mucho tiempo y fue imposible implementar los demás.
- Se implementa el Array dinamico con todos sus metodos de forma satisfactoria.
- Se implementa la lista simplemente enlazada con todos sus metodos de forma satisfactoria
- Fue posible desarrollar e implementar el bubblesort, parcialmente el mergesort, y la búsqueda linea sí fue implementada completamente.