Mini Proyecto 1
Estructura de datos 2023-01

Profesor:
-José Fuentes

Integrantes:
-Luciano Hernán Argomedo Solís
-Matias Nicolás Javier Cruces Gómez
-Cristóbal Hernán Solar Arias

1-Introducción

Las estructuras de datos son una herramienta fundamental en la programación, ya que nos permiten almacenar y manipular datos de manera eficiente y estructurada. Existen diferentes tipos de estructuras de datos, cada una con sus propias características y aplicaciones. Algunos ejemplos comunes son las listas, los árboles, stacks, etc.
Para poder trabajar de forma sencilla las estructuras de datos, utilizamos los ADT, ya que estos abstraen la complejidad de las estructuras de datos logrando proporcionar una interfaz clara y consistente de las estructuras que queremos utilizar. El adt define un conjunto de operaciones que se pueden realizar sobre una estructura de datos, sin especificar cómo se implementan estas operaciones. Esto permite que los usuarios que usen el ADT se centren en la lógica de su aplicación y no tengan que preocuparse de los detalles internos de la estructura de datos.

El objetivo de este proyecto, es poder desarrollar nuestra habilidad programando y diseñando un ADT, que es el caso del ListArr, y así aplicar todo el conocimiento que hemos obtenido hasta esta fecha del semestre.

2- Desarrollo

2.1- Explicación ListArr

La implementación que se nos pide crear es el “ListArr”, que se basa en una lista enlazada simple con las siguientes características:

-Un array de tamaño “n”.
-Un int igual a n.
-Un int igual a la cantidad de espacios ocupados del array.
-Un puntero al siguiente nodo.

Además de esto, tenemos un árbol binario, el cual tiene las siguientes características:

-Dos punteros, uno para su hijo izquierdo y otro para el derecho.
-Un int igual a la capacidad que tienen los arrays de sus hijos.
-Un int igual al espacio que ocupan sus hijos en sus arrays.
-Dos punteros a dos arrays.

Al crear las hojas, estas apuntan a dos arrays de la lista de arrays, y resumirá la capacidad total que hay entre esos dos arrays y los espacios ocupados que tienen, luego está información irá subiendo por el árbol hasta la raíz.
La función de esta estructura, es poder consultar rápidamente los valores de cierta posición del total de datos, ya que teniendo la información en los nodos del árbol, se puede ir dividiendo en algunos casos porque lado del árbol está la posición que se busca.

2.2-Implementación

La implementación que se decidió para el trabajo es la siguiente:

-ArrayListArr
-NodoListArr
-ListArr

Cada uno de estos tiene su archivo de cabecera y su archivo con implementación de los métodos.
Ahora bien, ¿Cómo funcionan estas 3 clases?, con ArrayListArr se crea la lista enlazada con los arrays. La clase tiene los siguientes atributos:

-Un array de largo n.
-Un int igual al largo del array.
-Un int igual a la cantidad de datos ingresados en el array.
-Un puntero al siguiente ArrayListArr.

Con estos cuatro parámetros y los métodos pertinentes, se crea la lista con los arrays.

NodoListArr es la clase que se ocupa para crear el árbol (aunque desde ListArr se hace el llamado). La clase tiene las siguientes características:

-Un struct Nodo, el cual tiene dos punteros a sus hijos y dos punteros a dos posibles arrays, si el nodo es una hoja se utilizan esos punteros, en caso contrario no se utilizan.

-Un int igualado a la suma de la capacidad de los dos arrays a los que apunta o a la suma de la capacidad de sus hijos.

-Un int igualado a la suma de la cantidad de datos ingresados de los dos arrays que apunta o a la suma de la cantidad de datos ingresados de sus dos hijos.

Con estos parámetros, se tiene ya todo lo necesario para armar el árbol binario.
Finalmente, en la clase ListArr, se llaman los constructores de las dos clases anteriores en un solo método, y así comienza la construcción de la estructura en su totalidad, siendo esta clase una combinación de las dos anteriores para crear la estructura.
Para facilitar este proceso, la clase tiene los siguientes parámetros:

-Un int llamado size para saber la cantidad de arrays en total que hay.
-Un int que tendrá el tamaño que se utilizará en los arrays.
-Un puntero a NodoListArr que será la raíz del árbol.

-Un vector de punteros a NodoListArr que apuntarán a las hojas del árbol.

-Un puntero ArrayListArr que apunta al primer array.
-Un puntero ArrayListArr que apunta al último array.

2.3-Análisis de complejidad

Para esta parte del informe, se analizarán los siguientes métodos de las siguientes clases:

1-getSize()
2-insert_left()
3-insert_right()
4-insert()
5-print()
6-find()
7-delete_left()
8-delete_right()

1) getSize()

Esta implementación será O(1), ya que es simplemente consultar el valor de un int el cual ya se tiene como parámetro en ListArr.

2) insert_left()

Insertar a la izquierda tiene una complejidad en el peor caso de O(b), siendo b el tamaño que tiene cada arreglo. Esto se concluye teniendo que insertar al inicio del array cuando el espacio está ocupado, cuando se tiene ese caso se deben correr todos los elementos del array a la derecha para dejar ese lugar disponible, el mover los elementos de un arreglo con b elementos tiene complejidad O(b). Cuando el array first está lleno se debe crear otro array además de correr los elementos del array first, mover elementos es O(b) y crear un array también es O(b) pues se inicializan las b casillas del nuevo array como desocupado. En resumen, insert_left() tiene complejidad O(b).

3) insert_right()

Guiándonos solo en el peor caso, el cual es tener que crear un nuevo array, como ya se explicó esto se hace en tiempo O(b), si no se requiere crear uno nuevo esto se hará en tiempo constante O(1). Por el peor caso que tiene este método este se queda con una complejidad temporal O(b), con b en tamaño del arreglo.

4) insert()

Este método se divide en dos partes, una que busca el arreglo al cual se le debe agregar el elemento y cuando ya se encuentra tener que agregarlo. Para la búsqueda del arreglo, como se recorre por los hijos de cada nodo al igual que un árbol binario esto toma tiempo O(logn). Para insertar algo a un array ya se dijo que toma tiempo O(b). Al combinar ambos tiempos se obtiene la complejidad de insert(), esta viene siento O(b logn), siendo b el tamaño de cada array y n la cantidad de arrays que hay.

5) print()

Esta implementación será simplemente O(n), ya que únicamente debe recorrer todos los arrays, los cuales la suma de sus tamaños suman n.

6) find()

Esta implementación puede variar , ya que dependerá si el valor que se busca está o no en algún array, en el mejor caso, está en la primera posición del primer array y eso sería O(1). En el peor caso, sería O(n), ya que se deberá recorrer hasta la última posición del último array.

7) delete_right()

Es un método de la clase `ListArr` que borra el último elemento del último array no vacío de la lista de arrays. Si el último array está vacío, entonces se borra el último array no vacío de la lista.

La función primero verifica si el último array de la lista está vacío. Si es así, se busca el último array no vacío de la lista y se borra su último elemento. Si el último array no está vacío, entonces se borra su último elemento. Si después de borrar el elemento, el último array está vacío y no es el primer array de la lista, se elimina el último array de la lista y se actualiza el puntero "end" para que apunte al nuevo último array de la lista.

8) delete_left()

La función toma un nodo como argumento y comprueba si ese nodo tiene un nodo anterior (es decir, si no es el primer elemento de la lista). Si existe un nodo anterior, el puntero next del nodo anterior se establece en el puntero next del nodo actual. Esto elimina el nodo actual de la lista enlazada. Después, la función comprueba si el nodo actual tiene un nodo siguiente (es decir, si no es el último elemento de la lista). Si hay un nodo siguiente, el puntero prev del nodo siguiente se establece en el puntero prev del nodo actual. Esto asegura que los punteros prev y next de la lista enlazada estén actualizados correctamente después de la eliminación del nodo actual.

Por último, la función devuelve el nodo anterior si existe (es decir, el nodo que ahora está a la izquierda del nodo eliminado), o el nodo siguiente si no hay ningún nodo anterior (es decir, el nuevo primer elemento de la lista).

Especificaciones de la pc utilizada:

- **Procesador:** AMD Ryzen 7 3700 U with Radeon Vega Mobile Gfx 2.30 GHz
- **Ram:** 8,00 GB
- **Tipo de sistema operativo:** Sistema operativo de 64 bits, procesador x64
- **Edición:** Windows 10 Home Single Language
- **Versión:** 22H2

Conclusión

Finalmente, por falta de tiempo y poca participación de cierto/s participante/s del grupo, no se pudo tener los métodos delete y la experimentación de los métodos.