TRABAJO ESCRITO DEL PROYECTO

PROYECTO #2 - ÁRBOLES BINARIOS

Profesor: Edgar Tista García

Asignatura: Estructura de Datos y Algoritmos II

Grupo: 5

Integrantes: Martínez Trinidad Alexis, Vázquez Flores José Angel, Zarate Menes Quetzalli

No. de lista: 22, 30, 35

Semestre: 2024-2

Fecha de entrega: 25 de mayo del 2024

Observaciones:

CALIFICACIÓN:

OBJETIVO

Que el alumno implemente aplicaciones relacionadas con los árboles binarios y que desarrolle sus habilidades de trabajo en equipo y programación orientada a objetos.

INTRODUCCIÓN

En el mundo de la programación, las estructuras de datos son como los cimientos de una casa: organizan y mantienen la información de manera eficiente. Entre las más útiles y comunes están los árboles binarios, que representan relaciones jerárquicas como las ramas de un árbol.

Este proyecto busca crear un programa que explore las aplicaciones de los árboles binarios. El programa permitirá trabajar con tres tipos:

Árboles AVL: Son árboles binarios de búsqueda que se mantienen "equilibrados", con una altura similar en ambos lados. Esto los hace rápidos para buscar y agregar información.

Árboles Red-Black: Son una variante de los árboles AVL que usan "colores" para mantener el equilibrio. Son ideales para aplicaciones que necesitan un comportamiento predecible y eficiente.

Árboles de expresión aritmética: Representan operaciones matemáticas como sumas y restas. Permiten evaluar expresiones complejas de forma rápida.

El programa tendrá un menú fácil de usar. El usuario podrá elegir el tipo de árbol y la operación que desea realizar, como agregar, buscar, eliminar elementos o resolver expresiones aritméticas. Así, se explorarán las capacidades y ventajas de cada tipo de árbol binario en diferentes situaciones.

En resumen, este proyecto creará una herramienta para entender y usar mejor los árboles binarios, estructuras de datos esenciales en la programación.

Algoritmos de inserción y eliminación (para un árbol binario balanceado AVL)

Algoritmo para la construcción de un árbol de expresión aritmética.

Algoritmos de árbol red black

El algoritmo de árbol rojo-negro es un método para organizar datos en un árbol binario de búsqueda. Garantiza que el árbol esté equilibrado, lo que significa que las operaciones de búsqueda, inserción y eliminación se pueden realizar en tiempo logarítmico promedio, incluso en el peor de los casos. Esto lo convierte en una estructura de datos eficiente para almacenar y recuperar información.

Propiedades de los Árboles Rojo-Negro:

1. Cada nodo es rojo o negro.
2. La raíz es negra.
3. Todas las hojas (NIL o nodos externos) son negras.
4. Si un nodo es rojo, entonces ambos hijos son negros (no puede

haber dos nodos rojos consecutivos).

5. Para cada nodo, todos los caminos simples desde ese nodo hasta sus descendientes NIL contienen el mismo número de nodos negros.

Operaciones en Árboles Rojo-Negro:

Algoritmo de Inserción

- La inserción en un árbol rojo-negro sigue estos pasos generales:
  - Inserta el nuevo nodo como en un árbol binario de búsqueda normal, poniéndolo en la posición adecuada según el orden.
  - Marca el nodo insertado como rojo.
- Rebalanceo del árbol:
  - Si el nodo insertado es la raíz, se pinta de negro (propiedad 2).
  - Si el padre del nodo insertado es negro, el árbol sigue siendo un árbol rojo-negro válido.
  - Si el padre del nodo insertado es rojo, entonces se viola la propiedad 4 y se deben realizar ajustes. Estos ajustes se hacen a través de rotaciones y recoloreos. Hay varios casos que manejar, dependiendo de la estructura del árbol y la posición del nodo insertado.

Casos de Rebalanceo: Sea z el nodo insertado, p el padre de z, g el abuelo de z y t el tío de z.

1. Caso 1: El padre es negro
   - El árbol está balanceado y no se requiere ninguna acción.

2. Caso 2: El padre es rojo y el tío es rojo:
    - Pinta el padre y el tío de negro.
    - Pinta el abuelo de rojo.
    - Repite el proceso desde el abuelo (ahora considerado como el nodo insertado z).
3. Caso 3: El padre es rojo y el tío es negro o NIL:
    - Caso 3a: z es el hijo izquierdo de p y p es el hijo izquierdo de g (o simétricamente):
        - Realiza una rotación a la derecha en g.
        - Intercambia los colores de p y g.
    - Caso 3b: z es el hijo derecho de p y p es el hijo izquierdo de g (o simétricamente):
        - Realiza una rotación a la izquierda en p.
        - Ahora z es el hijo izquierdo de p.
        - Aplica el caso 3a.

Algoritmo de Eliminación: La eliminación de un nodo en un árbol rojo-negro también se realiza en dos fases

1. Eliminación
    - Encuentra el nodo z que se quiere eliminar.
    - Si z tiene dos hijos, encuentra el sucesor, intercambia los valores y elimina el sucesor.
    - Marca el nodo que reemplazará a z como x.
2. Rebalanceo del Árbol

- Si x o el nodo eliminado son rojos, simplemente se pinta de negro y termina.
- Si x y el nodo eliminado son negros, se debe realizar un rebalanceo. Existen varios casos:
  - Caso 1: x es la nueva raíz:
    - No se requiere ninguna acción adicional.
  - Caso 2: x tiene un hermano w rojo:
    - Intercambia los colores del padre de x y w.
    - Realiza una rotación en el padre de x y actualiza w.
  - Caso 3: x tiene un hermano w negro y ambos hijos de w son negros:
    - Pinta w de rojo.
    - Si el padre de x es rojo, se pinta de negro.
    - Si el padre de x es negro, repite el proceso desde el padre de x.
  - Caso 4: x tiene un hermano w negro, el hijo izquierdo de w es rojo y el hijo derecho de w es negro:
    - Pinta el hijo izquierdo de w de negro y w de rojo.
    - Realiza una rotación a la derecha en w.
  - Caso 5: x tiene un hermano w negro y el hijo derecho de w es rojo:
    - Intercambia los colores de w y el padre de x.
    - Pinta el hijo derecho de w de negro.
    - Realiza una rotación a la izquierda en el padre de x.

- Termina.

ANÁLISIS DEL DESARROLLO DEL PROGRAMA

Menus.java

Los dos programas, Menus.java y Main.java, trabajan en conjunto para proporcionar una interfaz de usuario interactiva para gestionar diferentes tipos de árboles binarios. El programa principal (Main.java) llama un método de la clase Menus y la utiliza para mostrar los menús principales y navegar entre las opciones. A su vez, la clase Menus contiene las funciones para mostrar los menús específicos de cada tipo de árbol y manejar las interacciones del usuario.

Elementos teóricos necesarios:

- Estructuras de datos: El código utiliza la estructura de datos "árbol binario"para almacenar y manipular la información. Se implementan tres tipos específicos de árboles binarios:
  - Árbol AVL: Un árbol binario autobalanceado que mantiene un equilibrio entre la altura de sus subárboles.
  - Árbol Red-Black: Otro tipo de árbol binario autobalanceado con propiedades similares al árbol AVL.
  - Árbol de expresión aritmética: Un árbol binario que representa una expresión aritmética, donde cada nodo contiene un operador o un operando.
- Menús interactivos: El código utiliza técnicas de interacción con el usuario para mostrar los menús y capturar las opciones seleccionadas.

Estrategia para resolver el problema:

1. Diseño de la interfaz de usuario: Se definen los menús principales y los menús específicos para cada tipo de árbol, considerando las opciones disponibles para cada tipo de dato.
2. Implementación de las funciones de menú: Se desarrollan las funciones para mostrar cada menú y manejar las acciones del usuario, como agregar claves, buscar valores, eliminar claves, mostrar el árbol y resolver expresiones (en el caso del árbol de expresión).
3. Integración con el programa principal: Se crea una instancia de la clase Menus en el programa principal y se la utiliza para mostrar los menús y navegar entre las opciones.

Avance logrado:

El código proporcionado representa un avance significativo en la implementación de una interfaz de usuario para gestionar diferentes tipos de árboles binarios. Se han diseñado los menús, se ha implementado la lógica para mostrarlos y navegar entre ellos, y se ha esbozado la implementación de las opciones de menú para cada tipo de árbol. Por lo cual en cuanto al menú, consideramos un avance del 100 %.

0.1. RedBlackTree.java

El programa implementa un Árbol Rojo-Negro (Red-Black Tree) con las siguientes funcionalidades:

- **Insertar clave (insert):** Permite agregar una clave en el árbol, manteniendo las propiedades del Árbol Rojo-Negro.
- **Eliminar clave (deleteNode):** Permite eliminar una clave específica del árbol, también manteniendo las propiedades del Árbol Rojo-Negro.
- **Mostrar árbol (printTree):** Imprime el árbol en la consola,

mostrando la estructura y los colores de los nodos.

0.2. NodeRedBlack.java

Define la estructura del nodo del árbol Rojo-Negro. Cada nodo contiene:

- **data:** El valor almacenado en el nodo.
- **parent:** El nodo padre.
- **left:** El hijo izquierdo.
- **right:** El hijo derecho.
- **color:** El color del nodo (0 para negro y 1 para rojo).

0.2.1. Descripción de los Elementos Teóricos

Un Árbol Rojo-Negro es un tipo de árbol binario de búsqueda autobalanceado con las siguientes propiedades:

1. Cada nodo es rojo o negro.
2. La raíz es negra.
3. Todas las hojas (TNULL) son negras.
4. Si un nodo es rojo, entonces ambos hijos son negros.
5. Para cada nodo, todos los caminos desde el nodo hasta sus hojas descendientes contienen el mismo número de nodos negros.

Estas propiedades aseguran que el árbol se mantenga aproximadamente balanceado, garantizando una complejidad de tiempo $O(\log n)$ para las operaciones de búsqueda, inserción y eliminación.

0.2.2. Estrategia para Resolver el Problema

La estrategia para implementar el Árbol Rojo-Negro se dividió en varios pasos clave:

1. **Definición de la estructura del nodo (Node):** Crear una clase para representar los nodos del árbol, incluyendo la información necesaria y el color.
2. **Inicialización del árbol (RedBlackTree):** Implementar el constructor para inicializar el árbol con un nodo nulo (TNULL) que sirve como hoja para todos los nodos.
3. **Implementación de las rotaciones (leftRotate y rightRotate):** Estas funciones son cruciales para mantener el balance del árbol durante las inserciones y eliminaciones.
4. **Inserción de nodos (insert y fixInsert):** Implementar la inserción de nuevos nodos, seguido de la corrección del árbol para mantener las propiedades del Árbol Rojo-Negro.
5. **Eliminación de nodos (deleteNode y fixDelete):** Implementar la eliminación de nodos y la corrección subsecuente del árbol.
6. **Impresión del árbol (printTree):** Desarrollar una función para visualizar la estructura y colores del árbol en la consola.

0.2.3. Principales Dificultades y Soluciones

- **Mantener las propiedades del Árbol Rojo-Negro:** La mayor dificultad fue asegurarse de que las propiedades del Árbol Rojo-Negro se mantuvieran después de cada inserción y eliminación. Esto se resolvió implementando correctamente las rotaciones y las funciones de corrección (*fixInsert* y *fixDelete*).

- **Manejo de casos especiales durante la eliminación:** El proceso de eliminación es más complejo debido a los diversos casos que pueden surgir (por ejemplo, cuando el nodo a eliminar tiene dos hijos). Se implementaron cuidadosamente los casos especiales y se realizaron pruebas para asegurar la corrección del algoritmo.
- **Visualización del árbol:** Imprimir el árbol de manera que la estructura y los colores fueran claros requirió un manejo cuidadoso de la recursión y los prefijos para representar los niveles y ramas del árbol.

0.2.4. Relación con los Elementos Vistos en Teoría

Este proyecto está directamente relacionado con los conceptos de árboles binarios de búsqueda, árboles autobalanceados y algoritmos de inserción y eliminación en estructuras de datos. Específicamente, los Árboles Rojo-Negro son un tipo avanzado de árbol binario de búsqueda que garantiza el balance del árbol mediante reglas de color y rotaciones, lo cual es un tema fundamental en cursos de estructuras de datos y algoritmos.

0.2.5. Avance Logrado de Acuerdo con los Requerimientos

- **Árbol Red Black Implementado:**
  - **Agregar clave:** Implementado con éxito mediante el método *insert* y la función de corrección *fixInsert*.
  - **Eliminar clave:** Implementado con éxito mediante el método *deleteNode* y la función de corrección *fixDelete*.
  - **Mostrar árbol:** Implementado con éxito mediante el método *printTree*, que muestra la estructura y colores de los nodos

en la consola.

En conclusión, se lograron todos los requerimientos solicitados: se implementó un Árbol Rojo-Negro con las funcionalidades de agregar, eliminar y mostrar el árbol, manteniendo las propiedades teóricas y prácticas necesarias para asegurar el balance del árbol.

CONCLUSIONES INDIVIDUALES

Martínez Trinidad Alexis

Vázquez Flores José Angel

Zarate Menes Quetzalli

Análisis del Cumplimiento de los Objetivos

El objetivo principal de la práctica era que el alumno implementara aplicaciones relacionadas con los árboles binarios y desarrollara habilidades de trabajo en equipo y programación orientada a objetos. Este objetivo se cumplió satisfactoriamente por las siguientes razones:

- **Implementación de Árboles Binarios:** Se logró implementar un Árbol Rojo-Negro, un tipo avanzado de árbol binario de búsqueda. Las funcionalidades de inserción, eliminación y visualización fueron desarrolladas con éxito.
- **Trabajo en Equipo:** La práctica requería una división de tareas clara y una colaboración constante para asegurar que todas las partes del programa funcionaran correctamente.
- **Programación Orientada a Objetos:** Se aplicaron principios de programación orientada a objetos, como encapsulación y

modularidad, a través de las clases *RedBlackTree* y *Node*.

Contribución al Aprendizaje del Concepto

Los ejercicios contribuyeron significativamente al aprendizaje de los conceptos relacionados con los árboles binarios de búsqueda autobalanceados, por las siguientes razones:

- **Profundización en Árboles Rojo-Negro:** Implementar un Árbol Rojo-Negro ayudó a entender profundamente sus propiedades y el funcionamiento de las rotaciones y correcciones necesarias para mantener el balance del árbol.
- **Algoritmos de Inserción y Eliminación:** Desarrollar los algoritmos de inserción y eliminación en el árbol Rojo-Negro facilitó la comprensión de las complejidades asociadas con el mantenimiento de las propiedades del árbol.

Posibles Ventajas y Desventajas del Concepto Visto en la Práctica

- **Ventajas:**
  - **Eficiencia:** Los árboles Rojo-Negro ofrecen una complejidad de tiempo $O(\log n)$ para operaciones de búsqueda, inserción y eliminación.
  - **Autobalanceo:** Mantienen el árbol balanceado automáticamente, lo cual es crucial para garantizar un rendimiento constante.
- **Desventajas:**
  - **Complejidad de Implementación:** La implementación es más compleja comparada con otros tipos de árboles binarios de búsqueda debido a la necesidad de manejar colores y

realizar rotaciones.

- **Sobrecarga Adicional:** La necesidad de mantener información adicional (colores y punteros) introduce una pequeña sobrecarga en términos de memoria.

Crítica Constructiva y Propuestas de Mejora

- **Mejoras en la Práctica:**
  - **Incorporación de Visualización Gráfica:** Agregar una herramienta gráfica para visualizar el árbol Rojo-Negro podría facilitar la comprensión de los conceptos.
  - **Más Casos de Prueba:** Incluir una mayor variedad de casos de prueba, especialmente aquellos que cubren casos extremos y bordes, ayudaría a reforzar el entendimiento de las operaciones en el árbol.
- **Propuestas de Modificación:**
  - **División en Módulos Pequeños:** Dividir la práctica en módulos más pequeños podría hacer que los alumnos se concentren en partes específicas del árbol Rojo-Negro antes de abordar la implementación completa.

En resumen, la práctica no solo permitió cumplir con los objetivos establecidos, sino que también proporcionó una comprensión profunda de los árboles Rojo-Negro y sus operaciones, destacando tanto sus ventajas como sus desafíos en la implementación.