Amina Emurlai, Elaine Fink

ÜBUNG 2 – TREECHECK-PROTOKOLL

Datenstruktur:

Die Node-Struktur repräsentiert einen Knoten im binären Suchbaum und speichert einen Schlüssel und Zeiger auf die linken und rechten Kinder.

Funktionen:

- ***newNode(int key)****
  - Funktion: Erstellt einen neuen Knoten mit einem gegebenen Schlüssel.
  - Parameter:
    - key: Der Schlüsselwert, der dem neuen Knoten zugewiesen werden soll.
  - Rückgabewert: Gibt einen Zeiger auf den neu erstellten Knoten zurück.
- ***insert(Node node, int key)****
  - Funktion: Fügt einen neuen Knoten mit einem gegebenen Schlüssel in den Baum ein.
  - Parameter:
    - node: Zeiger auf den aktuellen Knoten im Baum.
    - key: Der Schlüsselwert, der in den Baum eingefügt werden soll.
  - Rückgabewert: Gibt einen Zeiger auf den aktualisierten Baum zurück.
- ***height(Node node)****
  - Funktion: Berechnet die Höhe eines Knotens im Baum.
  - Parameter:
    - node: Zeiger auf den aktuellen Knoten im Baum.
  - Rückgabewert: Die Höhe des Knotens.
- ***balanceFactor(Node node)****
  - Funktion: Berechnet den Balancefaktor eines Knotens im Baum.
  - Parameter:
    - node: Zeiger auf den aktuellen Knoten im Baum.
  - Rückgabewert: Der Balancefaktor des Knotens.
- ***inOrder(Node node, std::vector<int>& keys)****
  - Funktion: Führt eine Inorder-Traversierung des Baums durch und speichert die Schlüssel in einem Vektor.

  - Parameter:
    - node: Zeiger auf den aktuellen Knoten im Baum.
    - keys: Ein Vektor, der die Schlüssel der Knoten speichert.
  - Rückgabewert: Keiner (void).
- ***isAVL(Node node)****
  - Funktion: Überprüft, ob der gegebene Baum ein AVL-Baum ist.
  - Parameter:
    - node: Zeiger auf den aktuellen Knoten im Baum.
  - Rückgabewert: Ein boolscher Wert, der angibt, ob der Baum ein AVL-Baum ist oder nicht.
- ***checkAVL(Node root)****
  - Funktion: Überprüft, ob der gegebene Baum ein AVL-Baum ist und gibt das Ergebnis auf der Konsole aus.
  - Parameter:
    - root: Zeiger auf die Wurzel des zu überprüfenden Baums.
  - Rückgabewert: Keiner (void).
- ***searchkey(Node root, int key, std::string path = "")****
  - Funktion: Sucht einen Schlüssel im Baum und gibt den Pfad zum Schlüssel auf der Konsole aus.
  - Parameter:
    - root: Zeiger auf die Wurzel des Baums.
    - key: Der zu suchende Schlüssel.
    - path: Ein String, der den Pfad zum Schlüssel speichert.
  - Rückgabewert: Keiner (void).
- ***isSubtree(Node root, Node subtree)*****
  - Funktion: Überprüft, ob ein gegebener Subtree in einem anderen Baum vorhanden ist.
  - Parameter:
    - root: Zeiger auf die Wurzel des Baums.
    - subtree: Zeiger auf die Wurzel des Subtrees.
  - Rückgabewert: Ein boolscher Wert, der angibt, ob der Subtree im Baum vorhanden ist oder nicht.

Dateiein-/ausgabe:

Der Hauptteil des Programms liest einen Dateinamen ein, öffnet die Datei und fügt die Werte in den Baum ein. Dann werden Suchen ausgeführt, je nach Benutzerwahl.

O-Notation und Laufzeitvergleich: (sortiert: schnell > langsam)

- **newNode(int key):** Erstellt einen neuen Knoten mit dem gegebenen Schlüssel.
  - Laufzeit: O(1)
- **inOrder(Node* node, std::vector<int>& keys):** Durchläuft den Baum in inorder-Reihenfolge und speichert die Schlüssel in einem Vektor.
  - Laufzeit: O(n)
- **height(Node* node):** Berechnet die Höhe eines Knotens im Baum.
  - Laufzeit: O(n)
- **balanceFactor(Node* node):** Berechnet den Balancefaktor eines Knotens im Baum.
  - Laufzeit: O(n)
- **isAVL(Node* node):** Überprüft, ob der Baum die AVL-Eigenschaft erfüllt.
  - Laufzeit: O(n)
- **checkAVL(Node* root):** Überprüft, ob der gegebene Baum ein AVL-Baum ist und gibt das Ergebnis aus.
  - Laufzeit: O(n)
- **insert(Node* node, int key):** Fügt einen neuen Knoten mit dem gegebenen Schlüssel in den Baum ein.
  - Laufzeit: Im Durchschnitt O(log n), im schlimmsten Fall O(n), wobei n die Anzahl der Knoten im Baum ist.
- **searchkey(Node* root, int key, std::string path):** Sucht einen Schlüssel im Baum und gibt den Pfad aus.
  - Laufzeit: Im Durchschnitt O(log n), im schlimmsten Fall O(n), wobei n die Anzahl der Knoten im Baum ist.
- **isSubtree(Node* root, Node* subtree):** Überprüft, ob ein Baum ein Unterbaum eines anderen Baums ist.
  - Laufzeit: Im Durchschnitt O(n*m), im schlimmsten Fall O(n^2), wobei n die Anzahl der Knoten im Hauptbaum und m die Anzahl der Knoten im Unterbaum ist.

Funktionen, die in einem AVL-Baum arbeiten sind im Allgemeinen schneller als Funktionen, die jeden Knoten im Baum besuchen müssen. Funktionen, die einen Unterbaum überprüfen, sind am langsamsten.