Data Structure

|  |  |
| --- | --- |
| **Name** | **ID** |
| Malak Mahmoud Medhat Aref | 20221445867 |
| Nureen Ehab Mahmoud Barakat | 20221465124 |
| Zainab Mohamed Abdallah | 20220131251 |
| Alia Medhat Mohamed | 20221440735 |

Intelligent Systems Department

**Phone Book Management Application**

* We will implement the phone book system using the three types of Binary Search Tree: Avl tree, 2-4 tree and Heaps.
* For implementation we will use Java.
* Our program contains the following:

1. Three packages for each kind of the binary search tree.
2. A class named Contact contains the data of each contact in our csv files. Also, contains the setters and getter for each private variable, and the toString method that print the data of each contact, and a compare to method to compare between the data that the user will enter with the data inside the csv file.
3. The main class is named Test takes object from each Tree class.

* First, we will discuss the Heap class and then we will talk about each class separately.

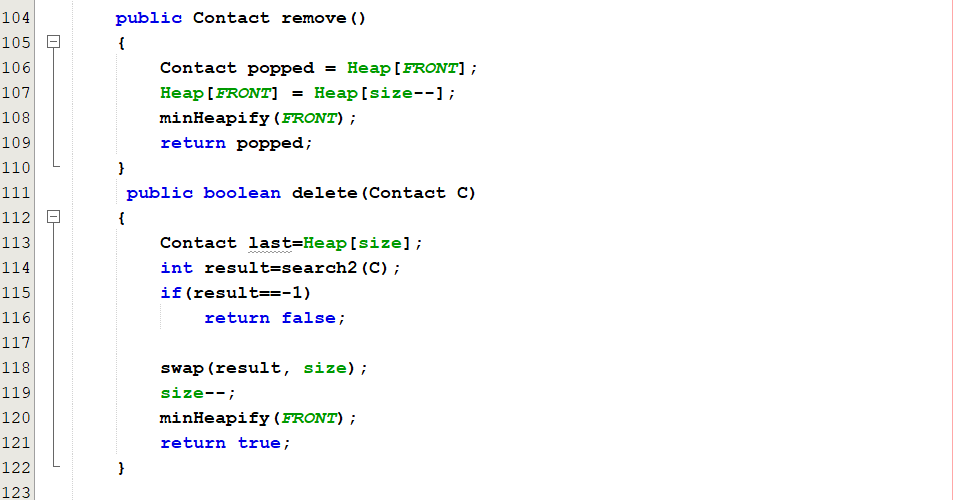
**The Heap** **Class contains the following:**

Algorithm:

* Steps to follow for deletion in Heap.

1. Remove root node.
2. Move the last element of last level to root.
3. Compare the value of this child node with its parent.
4. If value of parent is bigger than child, then swap them(heapify).
5. Repeat step 3 & 4 until Heap property holds.

Code:



Output:

Graphical user interface, text, application

Description automatically generated

* Steps to follow for insertion in Heap.

1. Create a new node at the end of heap.
2. Assign new value to the node.
3. Compare the value of this child node with its parent.
4. If value of parent is bigger than child, then swap them(heapify).
5. Repeat step 3 & 4 until Heap property holds.

Code:

Text

Description automatically generated

Output:

Graphical user interface, text, application

Description automatically generated

* Steps to follow for searching in Heap.

1. Compare the element taken from the user with our data.
2. If they are equal will return the contact information.

Code:

Graphical user interface, text, application, email

Description automatically generated

Output:

Text, application

Description automatically generated with medium confidence

**The 2-4 Tree** **Class contains the following:**

A 2-3-4 tree is a balanced search tree having following three types of nodes.

1. **2-node** has one key and two child nodes (just like binary search tree node).
2. **3-node** has two keys and three child nodes.
3. **4-node** has three keys and four child nodes.
4. Algorithm:

* Steps to follow for deletion in 2-4 Tree.

1. Delete operation depends on the location of the node containing the target (x) to be deleted.
2. We have several cases:

**Case 1**

Element is a Leaf Node with At least 2 Key/Values: If the element to delete in a node is a leaf node and has at least 2 Values in it then we can delete it straight forward. This is an edge case.

**Case 2**

Element is an Internal Node: If the element to delete is in a Node that is non-leaf or Internal there are again three possible conditions:

* 1. If Left Child has 2 Keys then replace the element with its predecessor or value in the left child and then delete it.
  2. Or, If the Right Child has 2 keys then replace it with its successor and then delete it.
  3. Now, If Both Left Child and Right Child have 1 key then merge them to form a balanced node and then delete the element.

**Case 3**

Element is not an Internal Node: If the element to delete is in a node which is a Leaf Node or Non Internal Node then there are again two possibilities.

1. If the Node has 1 key then we check its siblings or neighbor nodes if the Sibling has at least 2 keys. Then, we Rotate elements into the parent node and move the element in the parent node to the next node child node then delete it.
2. If Node has 1 key and both child nodes are siblings with 1 key each then merge the two children and the parent element into the node then delete it.

Code:

Graphical user interface, text, application

Description automatically generated

Output:

Graphical user interface, text, application

Description automatically generated

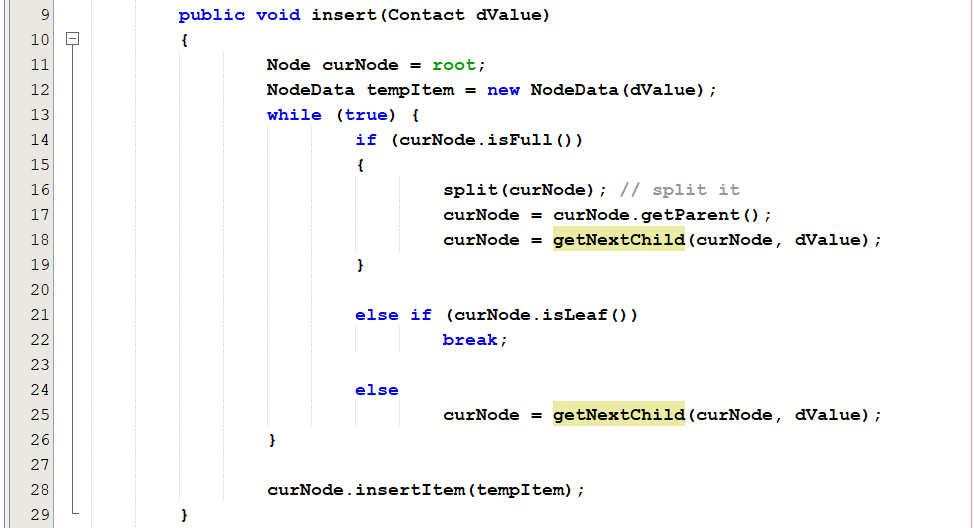
* Steps to follow for insertion 2-4 Tree.

1. Insert each data item in the node in sorted order. A node can hold maximum 3 values.
2. While inserting an element in a 4-Node we will split the node and move the middle item to its parent node if the parent becomes a

4-Node we split it again.

1. Once a node is split we cannot insert value in the parent node, we need to traverse to its child nodes and then insert or split the node itself.
2. The Parent Node cannot be a 4-Node so we can accommodate the extra data inside it while inserting.

Code:



Output:

Graphical user interface, text, application

Description automatically generated

* Steps to follow for searching 2-4 Tree.

To search a key **K** in given 2-3 tree **T**, we follow the following procedure:   
**Base cases:**

1. If **T** is empty, return False (key cannot be found in the tree).
2. If current node contains data value which is equal to **K**, return True.
3. If we reach the leaf-node and it doesn’t contain the required key value **K**, return False.

**Recursive Calls:**

1. If **K** < currentNode.leftVal, we explore the left subtree of the current node.
2. Else if currentNode.leftVal < **K** < currentNode.rightVal, we explore the middle subtree of the current node.
3. Else if **K** > currentNode.rightVal, we explore the right subtree of the current node.

Code:

Graphical user interface, text, application

Description automatically generated

Output:

A picture containing text

Description automatically generated

**The AVL-Tree** **Class contains the following:**

Algorithm:

* Steps to follow for insertion in AVL Tree.

1. Make sure that the tree remains AVL after every insertion by checking that the absolute balance factor equal 0 or 1.
2. Insert operation must perform some re-balancing by two basic operations without violating the BST property

(keys(left) < key(root) < keys(right)).

1) Left Rotation   
2) Right Rotation

Code:

Text

Description automatically generated

Output:

Graphical user interface, application, Word

Description automatically generated

* Steps to follow for searching in AVL Tree.

1. Start from the root of the tree and compare the key with the value of the node.
2. If the key equals the value, return the node. If the key is greater, search from the right child, otherwise continue the search from the left child.

Note: The time complexity of the search is a function of the height. We can assume that time complexity in the worst case is O(log(N)).

Code:

Graphical user interface, text, application

Description automatically generated

* Steps to follow for deletion in AVL Tree.

1. Make sure that the tree remains AVL after every deletion by checking that the absolute balance factor equal 0 or 1.
2. Delete operation must perform some re-balancing.
3. Two basic operations that can be performed to re-balance a BST without violating the BST property

(keys(left) < key(root) < keys(right)).

* 1. Left Rotation
  2. 2) Right Rotation

Code: