LAPORAN PRAKTIKUM
MATA KULIAH ALGORITMA DAN STRUKTUR DATA

Dosen Pengampu : Triana Fatmawati, S.T, M.T

PERTEMUAN - 13 - Tree

Nama : **M. Zidna Billah Faza**

NIM : **2341760030**

Prodi : **D-IV Sistem Informasi Bisnis**

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

Percobaan 1

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan array (praktikum 2) dan linked list (praktikum 1). Sebelumnya, akan dibuat class Node, dan Class BinaryTree

Node
data: int left: Node right: Node
Node(left: Node, data:int, right:Node)

BinaryTree
root: Node size : int
DoubleLinkedLists() add(data: int): void find(data: int) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) add(item: int, index:int): void delete(data: int): void

1) Buatlah class NodeNoAbsen, BinaryTreeNoAbsen dan BinaryTreeMainNoAbsen

```
public static class Node_18 {
```

2) Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```
public static class Node_18 {
    int data;
    Node_18 left;
    Node_18 right;

    public Node_18() {}

    public Node_18(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

3) Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.

```
public static class BinaryTree_18 {
    Node_18 root;
```

4) Tambahkan konstruktor default dan method isEmpty() di dalam class BinaryTreeNoAbsen

```
public BinaryTree_18() {
    root = null;
}

boolean IsEmpty() {
    return root == null;
}
```

5) Tambahkan method add() di dalam class BinaryTreeNoAbsen. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya, jika dilakukan dengan proses rekursif, penulisan kode akan lebih efisien.

```
void Add(int data) {
    if (IsEmpty()) {
        root = new Node_18(data);
    } else {
        Node_18 current = root;
        while (true) {
            if (data < current.data) {
                if (current.left == null) {
                    current.left = new Node_18(data);
                    break;
                } else {
                    current = current.left;
                }
            } else if (data > current.data) {
                if (current.right == null) {
                    current.right = new Node_18(data);
                    break;
                } else {
                    current = current.right;
                }
            } else {
                break;
            }
        }
    }
}
```

6) Tambahkan method find()

```
boolean Find(int data) {
    Node_18 current = root;
    while (current != null) {
        if (current.data == data) {
            return true;
        } else if (data < current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return false;
}
```

7) Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam tree, baik dalam mode pre-order, in-order maupun post-order.

```
void TraversePreOrder(Node_18 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        TraversePreOrder(node.left);
        TraversePreOrder(node.right);
    }
}

void TraversePostOrder(Node_18 node) {
    if (node != null) {
        TraversePostOrder(node.left);
        TraversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

void TraverseInOrder(Node_18 node) {
    if (node != null) {
        TraverseInOrder(node.left);
        System.out.print(" " + node.data);
        TraverseInOrder(node.right);
    }
}
```

8) Tambahkan method getSuccessor(). Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
Node_18 getSuccessor(Node_18 del) {
    Node_18 successorParent = del;
    Node_18 successor = del;
    Node_18 current = del.right;

    while (current != null) {
        successorParent = successor;
        successor = current;
        current = current.left;
    }

    if (successor != del.right) {
        successorParent.left = successor.right;
        successor.right = del.right;
    }

    return successor;
}
```

9) Tambahkan method delete(). Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```
void delete(int data) {
    if (IsEmpty()) {
        System.out.println("Tree is empty");
        return;
    }

    Node_18 parent = root;
    Node_18 current = root;
    boolean isLeftChild = false;

    while (current != null && current.data != data) {
        parent = current;
        if (data < current.data) {
            current = current.left;
            isLeftChild = true;
        } else {
            current = current.right;
            isLeftChild = false;
        }
    }
```

10) Kemudian tambahkan proses penghapusan didalam method delete() terhadap node current yang telah ditemukan.

```
        if (current == null) {
            System.out.println("Couldn't find data");
            return;
        }

        if (current.left == null && current.right == null) {
            if (current == root) {
                root = null;
            } else {
                if (isLeftChild) {
                    parent.left = null;
                } else {
                    parent.right = null;
                }
            }
        }

        else if (current.left == null) {
            if (current == root) {
                root = current.right;
            } else {
                if (isLeftChild) {
                    parent.left = current.right;
                } else {
                    parent.right = current.right;
                }
            }
        }

        else if (current.right == null) {
            if (current == root) {
                root = current.left;
            } else {
                if (isLeftChild) {
                    parent.left = current.left;
                } else {
                    parent.right = current.left;
                }
            }
        }

        else {
            Node_18 successor = getSuccessor(current);
            if (current == root) {
                root = successor;
            } else {
                if (isLeftChild) {
                    parent.left = successor;
                } else {
                    parent.right = successor;
                }
            }
            successor.left = current.left;
        }
    }
```

11) Buka class BinaryTreeMainNoAbsen dan tambahkan method main() kemudian tambahkan kode berikut ini

```
BinaryTree_18 bt = new BinaryTree_18();

bt.Add(data:6);
bt.Add(data:4);
bt.Add(data:8);
bt.Add(data:3);
bt.Add(data:5);
bt.Add(data:7);
bt.Add(data:9);
bt.Add(data:10);
bt.Add(data:15);

System.out.print(s:"Pre Order Traversal   : ");
bt.TraversePreOrder(bt.root);
System.out.println();

System.out.print(s:"In Order Traversal    : ");
bt.TraverseInOrder(bt.root);
System.out.println();

System.out.print(s:"Post Order Traversal  : ");
bt.TraversePostOrder(bt.root);
System.out.println();

System.out.println("Find Node 5: " + bt.Find(data:5));
System.out.println(x:"Delete Node 8");
bt.delete(data:8);

System.out.println();
System.out.print(s:"Pre Order Traversal:");
bt.TraversePreOrder(bt.root);
System.out.println();
```

12) Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.
13) Amati hasil running tersebut.

```
Pre Order Traversal   :  6 4 3 5 8 7 9 10 15
In Order Traversal    :  3 4 5 6 7 8 9 10 15
Post Order Traversal  :  3 5 4 7 15 10 9 8 6
Find Node 5: true
Delete Node 8

Pre Order Traversal: 6 4 3 5 9 7 10 15
```

Pertanyaan Percobaan 1

1) Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?

Binary search tree (BST) memungkinkan pencarian data yang lebih efisien karena setiap node kiri memiliki nilai yang lebih kecil dan setiap node kanan memiliki nilai yang lebih besar dari node induknya. Ini memfasilitasi eliminasi cepat setengah bagian pohon pada setiap langkah pencarian, mengurangi jumlah perbandingan yang diperlukan dibandingkan dengan binary tree biasa yang tidak terstruktur secara terurut.

2) Untuk apakah di class Node, kegunaan dari atribut left dan right?

Atribut tersebut menunjuk ke Node anak kiri dan anak kanan dari Node tersebut sehingga membantu dalam pembentukan struktu Binary Tree

3) a) Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?

Kegunaan dari atribut root pada class BinaryTree adalah sebagai pointer ke Node utama atau Node paling atas, Node ini merupakan titik awal dari Binary Tree

b) Ketika objek tree pertama kali dibuat, apakah nilai dari root?

Nilai dari root saat pertama kali dibuat adalah null, yang artinya Tree masih kosong tanpa Node

4) Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

Ketika Tree masih kosong dan akan ditambahkan Node baru maka Node tersebut akan diatur sebagai root sehingga menjadi titik awal Binary Tree

5) Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
while (true) {
    if (data > current.data) {
        if (current.left == null) {
            current = current.left;
        } else {
            current.left = new Node_18(data);
            break;
        }
```

Kode ini digunakan untuk menambahkan data ke dalam binary search tree. Jika data yang akan ditambahkan lebih kecil dari data pada node saat ini (current), program akan memeriksa apakah node anak kiri dari node saat ini sudah ada. Jika belum, data baru akan dibuat sebagai node anak kiri dari node saat ini. Jika sudah ada, pencarian akan berlanjut ke node anak kiri tersebut.

Percobaan 2

1) Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.
2) Buatlah class BinaryTreeArrayNoAbsen dan BinaryTreeArrayMainNoAbsen
3) Buat atribut data dan idxLast di dalam class BinaryTreeArrayNoAbsen. Buat juga method populateData() dan traverseInOrder().

```
public static class BinaryTreeArray_18 {

    int[] data;
    int idxlast;

    public BinaryTreeArray_18() {
        data = new int[18];
    }

    void PopulateData(int data[], int idxlast) {
        this.data = data;
        this.idxlast = idxlast;
    }

    void TraverseInOrder(int idxStart) {
        if (idxStart <= idxlast) {
            TraverseInOrder(2 * idxStart + 1);
            System.out.print(data[idxStart] + " ");
            TraverseInOrder(2 * idxStart + 2);
        }
    }

}
```

4) Kemudian dalam class BinaryTreeArrayMainNoAbsen buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method Main

```
BinaryTreeArray_18 bta = new BinaryTreeArray_18();
int data[] = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
int idxlast = 6;

bta.PopulateData(data, idxlast);
System.out.print(s:"\nInOrder Traversal   : ");
bta.TraverseInOrder(idxStart:0);
System.out.println(x:"\n");
```

5) Jalankan class BinaryTreeArrayMain dan amati hasilnya!

```
InOrder Traversal   : 3 4 5 6 7 8 9
```

Pertanyaan Percobaan 2

1) Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?

Atribut data dalam class BinaryTreeArray digunakan untuk menyimpan elemen-elemen dari pohon biner dalam bentuk array. Sedangkan idxLast digunakan untuk menunjukkan indeks terakhir dari elemen yang terisi dalam array tersebut, yang membantu dalam mengetahui jumlah elemen atau node yang telah ditambahkan ke dalam pohon

2) Apakah kegunaan dari method populateData()?

Method populateData() digunakan untuk menginisialisasi array data dan indeks terakhir (idxLast) pada class BinaryTreeArray. Method ini mengambil array data dan nilai idxLast sebagai parameter, kemudian menetapkan array tersebut ke variabel data dan nilai idxLast ke variabel idxLast dalam class.

3) Apakah kegunaan dari method traverseInOrder()?

Method traverseInOrder() digunakan untuk menampilkan / print elemen-elemen dari binary tree dalam bentuk array dalam urutan inOrder (kiri, root, kanan). Method ini menggunakan rekursi untuk mengecek setiap node dan menampilkan elemen-elemen dalam bentuk inOrder

4) Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?

Untuk node yang disimpan di indeks 2, left child akan berada di indeks 22+1 = 5, dan right child akan berada di indeks 22 + 2 = 6

5) Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

int idxLast = 6 menetapkan nilai idxLast menjadi 6, yang menunjukkan bahwa elemen atau node terakhir yang ditambahkan ke dalam array berada pada indeks 6

Tugas Praktikum 1

1) Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.

```
void Rekursif(Node_18 current, int data) {

    if (data < current.data) {
        if (current.left != null) {
            Rekursif(current.left, data);
        } else {
            current.left = new Node_18(data);
        }
    } else if (data > current.data) {
        if (current.right != null) {
            Rekursif(current.right, data);
        } else {
            current.right = new Node_18(data);
        }
    }

}
```

2) Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

```
int CariTerkecil() {

    Node_18 current = root;

    if (current == null) {
        System.out.println(x:"Tree is empty");
    }

    while (current.left != null) {
        current = current.left;
    }

    return current.data;

}

int CariTerbesar() {

    Node_18 current = root;

    if (current == null) {
        System.out.println(x:"Tree is empty");
    }

    while (current.left != null) {
        current = current.right;
    }

    return current.data;

}
```

3) Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf

```
void PrintLeaf(Node_18 node) {
    if (node == null) {
        return;
    } else if (node.left == null && node.right == null) {
        System.out.println(node.data + " ");
    }

    PrintLeaf(node.left);
    PrintLeaf(node.right);

}
```

4) Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```
int CountLeaf(Node_18 node) {

    if (node == null) {
        return 0;
    } else if (node.left == null && node.right == null) {
        return 1;
    }

    return CountLeaf(node.left) + CountLeaf(node.right);

}
```

5) Modifikasi class BinaryTreeArray, dan tambahkan

- method add(int data) untuk memasukan data ke dalam tree

```
void Add(int newData) {
    if (idxlast == data.length - 1) {
        System.out.println(x:"Tree is full");
        return;
    }
    data[idxlast++] = newData;
}
```

- method traversePreOrder() dan traversePostOrder()

```
void TraversePreOrder(int idxStart) {
    if (idxStart <= idxlast) {
        System.out.println(data[idxStart] + " ");
        TraversePreOrder(2 * idxStart + 1);
        TraversePreOrder(2 * idxStart + 2);
    }
}

void TraversePostOrder(int idxStart) {
    if (idxStart <= idxlast) {
        TraversePostOrder(2 * idxStart + 1);
        TraversePostOrder(2 * idxStart + 2);
        System.out.println(data[idxStart] + " ");
    }
}
```

Repository : https://github.com/zidnafaz/Praktikum-Algoritma-Struktur-Data