Job Sheet 14 Binary Tree

From:

AL AZHAR RIZQI RIFA'I FIRDAUS

Class:

1 I

Absence:

01

Student Number Identity:

2241720263

Department:

Information Technology

Study Program:

Informatics Engineering

Practicum 1 :

Code :

- Node

```
package practicum1;

public class Node {
    int data;
    Node left;
    Node right;

    public Node() {

    }

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

```

- Binary Tree

```java
package practicum1;

public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    boolean isEmpty() {
        return root == null;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void add(int data) {
        if (isEmpty()) {
            root = new Node(data);
        } else {
            Node current = root;
            while(true) {
                if(data<current.data) {
                    if(current.left!=null) {
                        current = current.left;
                    } else {
                        current.left = new Node(data);
                        break;
                    }
                } else if (data > current.data) {
                    if (current.right != null) {
                        current = current.right;
                    } else {
                        current.right = new Node(data);
                        break;
                    }
                } else { // data already exist
                    break;
                }
            }
        }
    }
```

```java
    Codeium: Refactor | Explain | Generate Javadoc
    boolean find(int data) {
        boolean result = false;
        Node current = root;
        while(current != null) {
            if (current.data == data) {
                result = true;
                break;
            } else if (data < current.data) {
                current = current.left;
            } else {
                current = current.right;
            }
        }
        return result;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void traversePreOrder(Node node) {
        if (node != null) {
            System.out.print(" " + node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void traversePostOrder(Node node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" " + node.data);
        }
    }

```

```
Codeium: Refactor | Explain | Generate Javadoc
void traverseInOrder(Node node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
        traverseInOrder(node.right);
    }
}

Codeium: Refactor | Explain | Generate Javadoc
Node getSuccessor(Node del) {
    Node successor = del.right;
    Node successorParrent = del;
    while(successor.left != null) {
        successorParrent = successor;
        successor = successor.left;
    }
    if (successor != del.right) {
        successorParrent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}

Codeium: Refactor | Explain | Generate Javadoc
void delete(int data) {
    if (isEmpty()) {
        System.out.println("Tree is empty!");
        return;
    }
    // find node (current) that will be deleted
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while (current != null) {
        if (current.data == data) {
            break;
        } else if (data < current.data) {
            parent = current;
            current = current.left;
            isLeftChild = true;
```

```
    } else if ( data > current.data) {
        parent = current;
        current = current.right;
        isLeftChild = false;
    }
}
if (current == null) {
    System.out.println("Couldn't find data!");
    return;
} else {
    // if there is no child, simply delete it
    if (current.left == null && current.right == null) {
        if (current == root) {
            root = null;
        } else {
            if (isLeftC Node left
                parent.left = null;
            } else {
                parent.right = null;
            }
        }
    } else if (current.left == null) { // uf there is 1 child (right)
        if (current == root) {
            root = current.right;
        } else {
            if (isLeftChild) {
                parent.left = current.right;
            } else {
                parent.right = current.right;
            }
        }
    } else if (current.right == null) { // if there is 1 child (left)
        if (current == root) {
            root = current.left;
        } else {
            if (isLeftChild) {
                parent.left = current.left;
            } else {
                parent.right = current.left;
            }
        }
```

```
                } else { // if there are 2 child
                    Node successor = getSuccessor(current);
                    if (current == root) {
                        root = successor;
                    } else {
                        if (isLeftChild) {
                            parent.left = successor;
                        } else {
                            parent.right = successor;
                        }
                    }
                    successor.left = current.left;
                }
            }
        }

}
```

- Main

```java
package practicum1;

public class BinaryTreeMain {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        BinaryTree bt = new BinaryTree();

        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);

        bt.traversePreOrder(bt.root);
        System.out.println();
        bt.traverseInOrder(bt.root);
        System.out.println();
        bt.traversePostOrder(bt.root);
        System.out.println();
        System.out.println("Find " + bt.find(data:5));
        bt.delete(data:8);
        bt.traversePreOrder(bt.root);
        System.out.println();
    }
}
```

Result :

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/
eet_14/coding]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsIn
xceptionMessages -cp /home/zharsuke/Documents/College/Semester_2/Data_Structure_and_
lgorithm_Practicum/Meet_14/coding/bin practicum1.BinaryTreeMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15

┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/
eet_14/coding]
└─$
```

Practicum 2

Code :

- Binary Tree Array

```java
package practicum2;

public class BinaryTreeArray {
    int [] data;
    int idxLast;

    public BinaryTreeArray() {
        data = new int[10];
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void populateData(int data [], int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void traverseInOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traverseInOrder(2*idxStart+1);
            System.out.print(data[idxStart] + " ");
            traverseInOrder(2*idxStart+2);
        }
    }
}

```

- Main

```java
package practicum2;

public class BinaryTreeMain {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        BinaryTreeArray bta = new BinaryTreeArray();

        int [] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.traverseInOrder(idxStart:0);
    }
}

```

Result :

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum
eet_14/coding]
└─$ /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsI
xceptionMessages -cp /home/zharsuke/Documents/College/Semester_2/Data_Structure_and
lgorithm_Practicum/Meet_14/coding/bin practicum2.BinaryTreeMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
3 4 5 6 7 8 9

┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum
eet_14/coding]
└─$
```

QUESTIONS

1. Why the data searching process is more efficient in the Binary search tree

than in an ordinary binary tree?

- **The data searching process is more efficient in a Binary Search Tree (BST) compared to an ordinary binary tree because the BST's structure allows for a focused search by eliminating half of the remaining elements at each step. This results in a faster search process, as it has a logarithmic time complexity (O(log n)) in the average case, while an ordinary binary tree has a linear time complexity (O(n)) for searching, where n is the number of elements in the tree.**

2. Why do we need the Node class? What are the left and right attributes?

- **The Node class is needed in the given code to represent individual nodes in a binary tree. Each Node object contains three attributes: data, left, and right. The data attribute stores the value associated with the node. It represents the actual data being stored or processed within the tree. The left attribute is a reference to the left child of the current node. It points to another Node object representing the left subtree. The right attribute is a reference to the right child of the current node. It points to another Node object representing the right subtree.**

3. a. What are the uses of the root attribute in the BinaryTree class?

b. When the tree object was first created, what is the value of root?

- **a. The root attribute in the BinaryTree class is used to store the reference to the root node of the binary tree. It serves as the starting point for accessing and traversing the tree's nodes and represents the overall structure of the tree.**
- **b. When the tree object was first created, the value of the root attribute is null. This indicates that the tree is initially empty and does not have any nodes. The root attribute is assigned the value of null in the constructor of the BinaryTree class.**

4. When the tree is still empty, and a new node is added, what process will

happen?

- **The add() method in the BinaryTree class will be called, passing the data value of the new node as a parameter. Since the tree is empty (root is null), the add() method will create a**

new Node object with the given data value. The newly created node will become the root node of the tree, and its left and right attributes will be set to null. The root attribute of the BinaryTree class will now reference the newly added node. Subsequent nodes can be added by traversing the tree and finding the appropriate position based on the values of the nodes.

5. Pay attention to the add() method, in which there are program lines as

below. Explain in detail what the program line is for?

```
if(data<current.data){
   if(current.left!=null){
      current = current.left;
   }else{
      current.left = new Node(data);
      break;
   }
}
```

- **The mentioned code snippet is part of the add() method in the BinaryTree class and is responsible for inserting a new node with a given data value into the binary tree.**

  If the data value of the new node is less than the data value of the current node being examined:

  - **If the current node has a left child, update the current node reference to its left child.**
  - **If the current node does not have a left child, create a new node with the given data value and assign it as the left child of the current node.**

6. What is the difference between pre-order, in-order and post-order

traverse modes?

- **Pre-order traversal: Visits the current node, then traverses the left subtree, and finally traverses the right subtree.**
- **In-order traversal: Traverses the left subtree, visits the current node, and then traverses the right subtree.**
- **Post-order traversal: Traverses the left subtree, traverses the right subtree, and then visits the current node.**

7. Look at the delete() method. Before the node removal process, it is

preceded by the process of finding the node to be deleted. Besides

intended to find the node to be deleted (current), the search process will

also look for the parent of the node to be deleted (parent). In your opinion,

Why is it also necessary to know the parent of the node to be deleted?

- **In the delete() method, finding the parent of the node to be deleted is crucial for correctly reassigning the child pointers when removing the node. By knowing the parent, we can update the appropriate child pointer of the parent node to properly link the remaining subtree after the deletion. If we only had a reference to the node to be deleted (current) and not its parent, we wouldn't be able to modify the parent's child pointer to reflect the removal. This could result in an inconsistent or incorrect binary tree structure. By keeping track of the parent node, we can determine whether the node to be deleted is a left or right child of its parent. Based on this information, we can appropriately update the parent's child pointer to skip the node being deleted and connect it with the remaining subtree.**

8. For what is a variable named isLeftChild created in the delete() method?

- **The variable isLeftChild in the delete() method is used to determine whether the node to be deleted is a left child or a right child of its parent. This information is important for correctly updating the parent's child pointer during the deletion process.**

9. What is the getSuccessor() method for?

- **The getSuccessor() method in the provided code is used to find the successor node of a given node during the deletion process in a binary search tree. The successor node is the node with the smallest value that is greater than the value of the given node. This method helps in maintaining the binary search tree property when deleting a node with two children.**

10. In a theoretical review, it is stated that when a node that has 2 children is

deleted, the node is replaced by the successor node, where the successor

node can be obtained in 2 ways, namely 1) looking for the largest value of

the subtree to the left, or 2) looking for the smallest value of subtree on

the right. Which 1 of 2 methods is implemented in the getSuccessor()

method in the above program?

- **The getSuccessor() method in the provided code implements the second method of finding the successor node when deleting a node with two children. It looks for the smallest value in the right subtree of the node to be deleted and returns it as the successor node.**

11. What are the uses of the data and idxLast attributes in the

BinaryTreeArray class?

- **The idxLast attribute keeps track of the index of the last element in the array, indicating the number of elements currently present in the binary tree.**

12. What are the uses of the populateData() and traverseInOrder()

methods?

- **The populateData() method in the BinaryTreeArray class is used to assign a given array of data and its corresponding idxLast value to the data and idxLast attributes of the class. It allows for initializing the binary tree with the provided data. The traverseInOrder() method in the BinaryTreeArray class is used to perform an in-order traversal of the binary tree**

represented by the data array. It recursively traverses the elements in the binary tree and prints them in the order specified by an in-order traversal.

13. If a binary tree node is stored in index array 2, then in what index are the

left-child and right child positions respectively?

- **If a binary tree node is stored at index 2 in the array representation, then the left-child position would be at index 2*2+1 = 5, and the right-child position would be at index 2*2+2 = 6.**

ASSIGNMENTS

1. Create a method inside the BinaryTree class that will add nodes with

recursive approach.

```
    Codeium: Refactor | Explain | Generate Javadoc
    void addRecursive(int data) {
        root = addRecursiveNode(root, data);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    private Node addRecursiveNode(Node current, int data) {
        if (current == null) {
            return new Node(data);
        }

        if (data < current.data) {
            current.left = addRecursiveNode(current.left, data);
        } else if (data > current.data) {
            current.right = addRecursiveNode(current.right, data);
        }
        return current;
    }
```

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

        bt.traversePreOrder(bt.root);
        System.out.println();
        bt.traverseInOrder(bt.root);
        System.out.println();
        bt.traversePostOrder(bt.root);
        System.out.println();
        System.out.println("Find " + bt.find(data:5));
        bt.delete(data:8);
        bt.traversePreOrder(bt.root);
        System.out.println();
```

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/
eet_14/coding]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjd
xceptionMessages -cp /home/zharsuke/Documents
lgorithm_Practicum/Meet_14/coding/bin practic
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFon
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
```

2. Create a method in the BinaryTree class to display the smallest and largest values in the tree.

```java
    Codeium: Refactor | Explain | Generate Javadoc
    public int findSmallestValue() {
        if (isEmpty()) {
            System.out.println("Tree is empty!");
        }

        Node current = root;
        while(current.left != null) {
            current = current.left;
        }
        return current.data;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public int findLargestValue() {
        if (isEmpty()) {
            System.out.println("Tree is empty!");
        }
        Node current = root;
        while(current.right != null) {
            current = current.right;
        }
        return current.data;
    }

```

```java
    System.out.println("Smallest value = " + bt.findSmallestValue());
    System.out.println("Largest value = " + bt.findLargestValue());
    System.out.println();
```

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semes
eet_14/coding]
└─$  /usr/bin/env /usr/lib/jvm/java-17-
xceptionMessages -cp /home/zharsuke/Doc
lgorithm_Practicum/Meet_14/coding/bin p
Picked up _JAVA_OPTIONS: -Dawt.useSyste
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Smallest value = 3
Largest value = 15
```

3. Create a method in the BinaryTree class to display the data in the leaf.

```java
    Codeium: Refactor | Explain | Generate Javadoc
    public void displayLeafNode() {
        displayLeafNode(root);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void displayLeafNode(Node node) {
        if (node == null) {
            return;
        }
        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
        }
        displayLeafNode(node.left);
        displayLeafNode(node.right);
    }

```

```java
        System.out.print("Data in leaf = ");
        bt.displayLeafNode();
        System.out.println();
```

```
┌──(zharsuke㉿asus-vivobook)-[~/
eet_14/coding]
└─$  /usr/bin/env /usr/lib/jvm/j
xceptionMessages -cp /home/zhars
lgorithm_Practicum/Meet_14/codin
Picked up _JAVA_OPTIONS: -Dawt.u
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Smallest value = 3
Largest value = 15

Data in leaf = 3 5 7 15
```

4. Create a method in the BinaryTree class to display the number of leaves

in the tree.

```java
Codeium: Refactor | Explain | Generate Javadoc
public int countLeave() {
    return countLeave(root);
}

Codeium: Refactor | Explain | Generate Javadoc
public int countLeave(Node node) {
    if (node == null) {
        return 0;
    }
    if (node.left == null && node.right == null) {
        return 1;
    }
    int leftCount = countLeave(node.left);
    int rightCount = countLeave(node.right);
    return leftCount + rightCount;
}

```

```java
System.out.println("Number of leaf node : " + bt.countLeave());
```

```
┌──(zharsuke㉿asus-vivobook)-
eet_14/coding]
└─$ /usr/bin/env /usr/lib/jv
xceptionMessages -cp /home/zh
lgorithm_Practicum/Meet_14/co
Picked up _JAVA_OPTIONS: -Daw
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Smallest value = 3
Largest value = 15

Data in leaf = 3 5 7 15
Number of leaf node : 4
```

5. Modify the BinaryTreeMain class, so that it has a menu option:

a. add

b. delete

c. find

d. traverse inOrder

e. traverse preOrder

f. traverse postOrder

g. keluar

```java
    Codeium: Refactor | Explain | Generate Javadoc
    public static void display() {
        System.out.println("Binary Tree Operations");
        System.out.println("1. Add");
        System.out.println("2. Delete");
        System.out.println("3. Find");
        System.out.println("4. Traverse Order");
        System.out.println("5. Traverse PreOrder");
        System.out.println("6. Traverse PostOrder");
        System.out.println("7. Exit");
    }
```

```java
        int choice;

        do {
            display();
            System.out.print("Insert choice : ");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Insert number to add : ");
                    int number1 = sc.nextInt();
                    bt.add(number1);
                    break;
                case 2:
                    System.out.print("Insert number to delete : ");
                    int number2 = sc.nextInt();
                    bt.delete(number2);
                    break;
                case 3:
                    System.out.print("Insert number to find : ");
                    int number3 = sc.nextInt();
                    System.out.println("Number " + number3 + " found : " + bt.find(number3));
                    break;
                case 4:
                    System.out.print("Traverse in order : ");
                    bt.traverseInOrder(bt.root);
                    System.out.println();
                    break;
                case 5:
                    System.out.print("Traverse pre order : ");
                    bt.traversePreOrder(bt.root);
                    System.out.println();
                    break;
                case 6:
                    System.out.print("Traverse post order : ");
                    bt.traversePostOrder(bt.root);
                    System.out.println();
                    break;
                case 7:
                    System.exit(0);
                    break;

                default:
                    System.out.println("Invalid input!");
                    break;
            }
        } while (choice != 7);
        sc.close();
```

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Da
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-
acticum/Meet_14/coding/bin practicum1.BinaryTre
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontS
Binary Tree Operations
1. Add
2. Delete
3. Find
4. Traverse Order
5. Traverse PreOrder
6. Traverse PostOrder
7. Exit
Insert choice : 4
Traverse in order :  3 4 5 6 7 8 9 10 15
Binary Tree Operations
1. Add
2. Delete
3. Find
4. Traverse Order
5. Traverse PreOrder
6. Traverse PostOrder
7. Exit
Insert choice : 1
Insert number to add : 99
```

```
Binary Tree Operations
1. Add
2. Delete
3. Find
4. Traverse Order
5. Traverse PreOrder
6. Traverse PostOrder
7. Exit
Insert choice : 5
Traverse pre order :  6 4 3 5 8 7 9 10 15 99
Binary Tree Operations
1. Add
2. Delete
3. Find
4. Traverse Order
5. Traverse PreOrder
6. Traverse PostOrder
7. Exit
Insert choice : 2
Insert number to delete : 6
```

```
Binary Tree Operations
1. Add
2. Delete
3. Find
4. Traverse Order
5. Traverse PreOrder
6. Traverse PostOrder
7. Exit
Insert choice : 6
Traverse post order :  3 5 4 99 15 10 9 8 7
Binary Tree Operations
1. Add
2. Delete
3. Find
4. Traverse Order
5. Traverse PreOrder
6. Traverse PostOrder
7. Exit
Insert choice : 3
Insert number to find : 99
Number 99 found : true
```

6. Modify the BinaryTreeArray class, and add:

a. Add add method (int data) to enter data into the tree

b. traversePreOrder() and traversePostOrder() methods

code :

```java
package practicum2;

public class BinaryTreeMain {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        BinaryTreeArray bta = new BinaryTreeArray();

        int [] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.traverseInOrder(idxStart:0);
        System.out.println();
        bta.traversePreOrder();
        System.out.println();
        bta.traversePostOrder();
        System.out.println();
        bta.add(data:10);
        bta.add(data:15);
        bta.traverseInOrder(idxStart:0);
    }
}
```

```java
    Codeium: Refactor | Explain | Generate Javadoc
    void traversePreOrder() {
        traversePreOrder(idxStart:0);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void traversePostOrder() {
        traversePostOrder(idxStart:0);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(data[idxStart] + " ");
            traversePreOrder(2*idxStart+1);
            traversePreOrder(2*idxStart+2);
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void traversePostOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traversePreOrder(2*idxStart+1);
            traversePreOrder(2*idxStart+2);
            System.out.print(data[idxStart] + " ");
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void add(int data) {
        if (idxLast == this.data.length -1) {
            System.out.println("Tree is full! Cannot add data!");
            return;
        }
        idxLast++;
        this.data[idxLast] = data;
    }
```

Result :

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_
eet_14/coding]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -X
xceptionMessages -cp /home/zharsuke/Documents/College/Semester_2
lgorithm_Practicum/Meet_14/coding/bin practicum2.BinaryTreeMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswin
3 4 5 6 7 8 9
6 4 3 5 8 7 9
4 3 5 8 7 9 6
10 3 15 4 5 6 7 8 9

┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_
eet_14/coding]
└─$
```