Furkan BAYTAK
210316033

Furkan ÖZKAYA
200316060

CSE 2105 – Data Structures
2022 – 2023 Fall Semester Project

Contents:

- Question1.java
  - Class “CustomNode”
  - Class “BTree”

- Question2.java
  - Class “HashTable”
  - Class “BinarySearchTree”

- Question3.java
  - Class “MultipleStacks”
  - Class “FullStackException”
  - Class “EmptyStackException”

- Question4.java
  - Class “Node”
  - Class “Graph”

QUESTION1

```java
class CustomNode {
    2 usages
    private CustomNode customLeftNode = null;
    2 usages
    private CustomNode customRightNode = null;
    2 usages
    int value = 0;

    1 usage
    public CustomNode(int value) { setValue(value); }

    4 usages
    public CustomNode getCustomLeftNode() { return customLeftNode; }

    4 usages
    public CustomNode getCustomRightNode() { return customRightNode; }

    4 usages
    public int getValue() { return value; }

    1 usage
    public void setCustomLeftNode(CustomNode n) { customLeftNode = n; }

    1 usage
    public void setCustomRightNode(CustomNode n) { customRightNode = n; }

    1 usage
    public void setValue(int d) { value = d; }
}
```

"CustomNode" Class:

CustomNode class has 3 attributes. There is a value of every node, and every node has left and right node for themselves. With methods which we wrote, we can set values for nodes (and it's left and right nodes) and get values of this nodes.

"BTree" Class:

With using "CustomNode" class we create a new custom node and with insert method, we insert incoming data to right places.

```java
class BTree {
    5 usages
    private CustomNode root = null;

    1 usage
    public void insert(int data) {
        root = insert(root, data);
    }

    3 usages
    private CustomNode insert(CustomNode node, int data) {
        if (node == null) {
            node = new CustomNode(data);
        } else {
            if (data <= node.getValue()) {
                node.setCustomLeftNode(insert(node.getCustomLeftNode(), data));
            } else {
                node.setCustomRightNode(insert(node.getCustomRightNode(), data));
            }
        }
        return node;
    }
```

```java
1 usage
public void inorder() { inorder(root); }

3 usages
private void inorder(CustomNode r) {
    if (r != null) {
        inorder(r.getCustomLeftNode());
        System.out.print(r.getValue() + " ");
        inorder(r.getCustomRightNode());
    }
}
```

```java
1 usage
public void preorder() { preorder(root); }

3 usages
private void preorder(CustomNode r) {
    if (r != null) {
        System.out.print(r.getValue() + " ");
        preorder(r.getCustomLeftNode());
        preorder(r.getCustomRightNode());
    }
}
```

Since B-tree is called a sorted tree as its nodes are sorted in inorder traversal, we can use preorder method to print BTree before it's sorted and use inorder method to print after it's with respect to BTree rules.

<u>Inorder Traversal:</u>

1. Traverse the left subtree (left->subtree)
2. Visit the root.
3. Traverse the right subtree (right->subtree)

<u>Preorder Traversal:</u>

1. Visit the root.
2. Traverse the left subtree (left->subtree)
3. Traverse the right subtree (right->subtree)

```java
public class Question1 {
    1 usage
    public static int[] CreateRandomValues(int numberCount) {
        Random random = new Random();
        int[] numbers = new int[numberCount];
        for (int i = 0; i < numberCount; i++) {
            numbers[i] = Math.abs(random.nextInt( bound: 100));
        }
        return numbers;
    }

    public static void main(String[] args) {
        BTree bt = new BTree();
        int[] randomValues = CreateRandomValues( numberCount: 20);
        for (int randomValue : randomValues) bt.insert(randomValue);
        System.out.println("Tree Elements: ");
        bt.preorder();
        System.out.println();
        System.out.println("Sorted Version: ");
        bt.inorder();
    }
```

Within the "Question1" class where our main method works, we created a method for creating an array with random values in it. Then we use random created integer arrays for controlling our sorting algorithms. We used "insert" method in "BTree class" to insert random created values in our BTree and first we print "preorder" sorted values of this tree and with respect to BTree rules we print "inorder" sorted values of this tree.

QUESTION2

```java
public HashTable(int size) {
    this.size = size;
    this.table = new BinarySearchTree[size];
    Arrays.fill(this.table, val: null);
}

3 usages
public static int hashKey(String word) {
    int key = 0;
    char[] c = word.toCharArray();
    for (Character ss : c) {
        key += ss - 'a' + 1;
    }
    return key;
}

2 usages
public int hashFunction(String word) {
    int key = hashKey(word);
    return key % this.size;
}
```

"HashTable" Class:

In this question we are trying to read words and their meanings from a file. With using "hashKey" method, with the word in file which we took with "fileIn" method, we turned the word to an integer value. So, we can match the words' meanings with a key value.

```java
public static void fileIn(HashTable words, String path) {
    try {
        File myObj = new File(path);
        Scanner myReader = new Scanner(myObj);

        while (myReader.hasNextLine()) {
            String word = myReader.nextLine();
            String means = myReader.nextLine();
            words.insert(word, means);
        }

        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

1 usage
public void insert(String word, String means) {
    int index = hashFunction(word);
    int key = hashKey(word);
    if (this.table[index] == null) {
        this.table[index] = new BinarySearchTree();
    }
    this.table[index].insert(key, means);
}

3 usages
public void search(String word) {
    int index = hashFunction(word);
    if (this.table[index] == null) {
        System.out.println("Word doesn't exist");
    } else {
        String means = this.table[index].Search(this.table[index].root, word);
        System.out.printf("%s: %s%n", word, means);
    }
}
```

With using "insert" method, we insert the words and their meanings in a binary search tree which we aslo created "BinaryTree" class ourselves and this class has regular search and insert mechanism for binary tree operations. "search" method in "HashTable" class, takes index as key we created with using "hashFunction" method, and with the "Search" method which we created in "BinaryTree" class, it searches words' meanings and print them.

```java
public class Question2 {
    public static void main(String[] args) {
        HashTable words = new HashTable( size: 18);
        HashTable.fileIn(words, path: "C:\\Users\\onyxe\\Desktop\\DataStructure\\src\\mywords.txt");
        //words{accoy,sigillography,caespitose,tephra,bunting,hetaerocracy,tepefaction,xebec
        //quippery,equatorium,diffrangible,divellent,frondiferous,torsiograph,tilleul
        //microsomatous,laminary,zugtrompete,toponym,waftage,ideophone,hypertrichologist
        //haemal,magnality,ballaster,refugium,rhumb,arcate,diacope,ubique}
        words.search( word: "accoy");
        words.search( word: "waftage");
        words.search( word: "tilleul");
    }
```

Within the "Question1" class where our main method works, we created a new HashTable from our class and with using "fileIn" method and with the ".txt" file which we located its path from our computer's path, we filled this HashTable with words and with their meanings. For trying our code, we print some word's meaning to screen with search method in "HashTable" class.

QUESTION3

```java
class MultipleStacks {
    3 usages
    private final int stackCapacity;
    5 usages
    private final int[] values;
    6 usages
    private final int[] sizes;

    1 usage
    public MultipleStacks(int numStacks, int stackCapacity) {
        this.stackCapacity = stackCapacity;
        this.values = new int[numStacks * stackCapacity];
        this.sizes = new int[numStacks];
    }

    5 usages
    public void push(int stackNum, int value) throws FullStackException {
        if (isFull(stackNum)) {
            throw new FullStackException();
        }
        sizes[stackNum]++;
        values[indexOfTop(stackNum)] = value;
    }
```

```java
1 usage
public int pop(int stackNum) throws EmptyStackException {
    if (isEmpty(stackNum)) {
        throw new EmptyStackException();
    }
    int topIndex = indexOfTop(stackNum);
    int value = values[topIndex];
    values[topIndex] = 0;
    sizes[stackNum]--;
    return value;
}

1 usage
public int peek(int stackNum) throws EmptyStackException {
    if (isEmpty(stackNum)) {
        throw new EmptyStackException();
    }
    return values[indexOfTop(stackNum)];
}
```

"MultipleStacks" Class:

We created a class and this classes attributes are;

stackCapacity: Number of stacks which this stack can store.

values: Values in these stacks.

sizes: Size of these stacks.

We created pop, push, peek, isFull, isEmpty and indexOfTop methods for this class. They're working like usual stack methods but these methods have attributes because we have to select which stack we'll use this methods for.

```java
public class Question3 {
    public static void main(String[] args) throws FullStackException, EmptyStackException {
        MultipleStacks myStack = new MultipleStacks( numStacks: 5,  stackCapacity: 3);
        myStack.push( stackNum: 2,  value: 1);
        myStack.push( stackNum: 2,  value: 3);
        myStack.push( stackNum: 1,  value: 2);
        myStack.push( stackNum: 1,  value: 3);
        myStack.push( stackNum: 4,  value: 4);
        myStack.pop( stackNum: 2);
        System.out.println(myStack.peek( stackNum: 2));
    }
}
```

In our main method, we push values to selected stacks and control if our code work.

QUESTION4

```
1 usage
public static void calculateShortestPathFromSource(Graph graph, Node source) {
    source.setDistance(0);

    Set<Node> settledNodes = new HashSet<>();
    Set<Node> unsettledNodes = new HashSet<>();

    unsettledNodes.add(source);

    while (unsettledNodes.size() != 0) {
        Node currentNode = getLowestDistanceNode(unsettledNodes);
        unsettledNodes.remove(currentNode);
        for (Map.Entry<Node, Integer> adjacencyPair :
                currentNode.getAdjacentNodes().entrySet()) {
            Node adjacentNode = adjacencyPair.getKey();
            Integer edgeWeight = adjacencyPair.getValue();
            if (!settledNodes.contains(adjacentNode)) {
                calculateMinimumDistance(adjacentNode, edgeWeight, currentNode);
                unsettledNodes.add(adjacentNode);
            }
        }
        settledNodes.add(currentNode);
    }
    System.out.println(source.getName());
    for (Node node : settledNodes) {
        System.out.println(node.getName());
    }
}
```

For this question, we created “Node” class to create nodes, finding distance between these nodes, getting adjacent nodes, and getting shortest path between these nodes. And we created “Graph” class to collect these nodes in one graph.


```
public static void calculateShortestPathFromSource(Graph graph, Node source) {
    source.setDistance(0);

    Set<Node> settledNodes = new HashSet<>();
    Set<Node> unsettledNodes = new HashSet<>();

    unsettledNodes.add(source);

    while (unsettledNodes.size() != 0) {
        Node currentNode = getLowestDistanceNode(unsettledNodes);
        unsettledNodes.remove(currentNode);
        for (Map.Entry<Node, Integer> adjacencyPair :
                currentNode.getAdjacentNodes().entrySet()) {
            Node adjacentNode = adjacencyPair.getKey();
            Integer edgeWeight = adjacencyPair.getValue();
            if (!settledNodes.contains(adjacentNode)) {
                calculateMinimumDistance(adjacentNode, edgeWeight, currentNode);
                unsettledNodes.add(adjacentNode);
            }
        }
        settledNodes.add(currentNode);
    }
    System.out.println(source.getName());
    for (Node node : settledNodes) {
        System.out.println(node.getName());
    }
}
```


```
1 usage
private static void calculateMinimumDistance(Node evaluationNode,
                                             Integer edgeWeigh, Node sourceNode) {
    Integer sourceDistance = sourceNode.getDistance();
    if (sourceDistance + edgeWeigh < evaluationNode.getDistance()) {
        evaluationNode.setDistance(sourceDistance + edgeWeigh);
        LinkedList<Node> shortestPath = new LinkedList<>(sourceNode.getShortestPath());
        shortestPath.add(sourceNode);
        evaluationNode.setShortestPath(shortestPath);
    }
}

1 usage
private static Node getLowestDistanceNode(Set<Node> unsettledNodes) {
    Node lowestDistanceNode = null;
    int lowestDistance = Integer.MAX_VALUE;
    for (Node node : unsettledNodes) {
        int nodeDistance = node.getDistance();
        if (nodeDistance < lowestDistance) {
            lowestDistance = nodeDistance;
            lowestDistanceNode = node;
        }
    }
    return lowestDistanceNode;
}
```

We used “calculateShortestPathFromSource” method to calculate for given graph that finds path which connects all the vertices together, without any cycles and with the minimum possible total edge weight. To calculate this path, we created two other methods “calculateMinimumDistance” and “getLowestDistanceNode”. But there is a mistake in this code which we couldn’t figure it out.

```java
public static void main(String[] args) {
    Node nodeA = new Node( name: "A");
    Node nodeB = new Node( name: "B");
    Node nodeC = new Node( name: "C");
    Node nodeD = new Node( name: "D");
    Node nodeE = new Node( name: "E");
    Node nodeF = new Node( name: "F");
    Node nodeG = new Node( name: "G");
    Node nodeH = new Node( name: "H");

    nodeA.addDestination(nodeB,  distance: 12);
    nodeA.addDestination(nodeC,  distance: 17);
    nodeA.addDestination(nodeD,  distance: 20);

    nodeB.addDestination(nodeC,  distance: 21);
    nodeB.addDestination(nodeH,  distance: 19);

    nodeC.addDestination(nodeD,  distance: 4);
    nodeC.addDestination(nodeG,  distance: 6);
    nodeC.addDestination(nodeE,  distance: 88);

    nodeD.addDestination(nodeF,  distance: 15);
    nodeD.addDestination(nodeG,  distance: 13);

    nodeG.addDestination(nodeF,  distance: 44);
    nodeG.addDestination(nodeE,  distance: 37);

    nodeF.addDestination(nodeE,  distance: 30);

    nodeH.addDestination(nodeE,  distance: 19);
```

```java
Graph graph = new Graph();

graph.addNode(nodeA);
graph.addNode(nodeB);
graph.addNode(nodeC);
graph.addNode(nodeD);
graph.addNode(nodeE);
graph.addNode(nodeF);
graph.addNode(nodeG);
graph.addNode(nodeH);

calculateShortestPathFromSource(graph, nodeB);
```

In our main method, we created nodes first. Then we connect all the destinations between these nodes and wrote their distance value. Then we created a graph with using our “Graph” class and add these nodes in our graph. But when we try to use “calculateShortestPathFromSource” method there is a mistake facing with us.

```
B
B
E
G
F
D
C
H

Process finished with exit code 0
```

It starts with “B” as it should be, and then it jumps into “E” which they aren’t connect but rest of this code works fine until the end. Like at the beginning it suddenly jumps to “H” from “C” and they also aren’t connected. When we tried it with other source nodes, there is always a problem in “E” and “H” nodes. We’ll try to fix that.