CSCI 3901: Assignment 2

Submitted by **Nadipineni Hemanth Kumar [B00899473]**

PART – 1

Methods Expected:

→ gatherThesisInfo()
→ approveOrReject()
→ scheduleDefense()

1) Input Validation:

→ gatherThesisInfo()

- Null passed as Student name
- Null passed as Date and Time
- Null passed as Thesis Title
- Null passed as Thesis Abstract
- Null passed as Supervisor name
- Null passed as co-supervisors' names
- For each of two thesis readers
    - Null passed
- Null passed for the defense chair
- Empty String passed as Student name
- Empty value passed as Date and Time
- Empty String passed as Thesis Title
- Empty String passed as Thesis Abstract
- Empty String passed as Supervisor name
- Empty String passed as co-supervisors' names
- Student name has more than 2^31 – 1 characters
- Thesis Title has more than 2^31 – 1 characters
- Thesis Abstract has more than 2^31 – 1 characters
- Supervisor Name has more than 2^31 – 1 characters
- Each of co-supervisors' name has more than 2^31 – 1 characters

- The first thesis reader name has more than 2^31 – 1 characters
- The second thesis reader name has more than 2^31 – 1 characters
- The defense chair has more than 2^31 – 1 characters
- For each of two thesis readers
    - Empty string passed
- Empty String passed for the defense chair
- Date and time format is incorrect
- Date and time are out of the expected range
- Special characters in date and time

2)Boundary Cases:

→ gatherThesisInfo()

- 1 character Student name
- Date given is exactly the 7th day from the current date
- Date given is exactly the 14th day from the current date
- 1 character Thesis Title
- 1 character Thesis Abstract
- 1 character Supervisor Name
- 1 character co-supervisors' names
- For each of two thesis readers
    - 1 character name
- 1 character defense chair

→ approveOrReject()

- The supervisor has 0 thesis to approve
- The first reader has 0 thesis to approve
- The second reader has 0 thesis to approve
- The defense has 0 thesis to approve

→ scheduleDefense()

- 0 Thesis scheduled at given date and time
- 1 Thesis scheduled at given date and time

3)Control Flow Cases:

→ gatherThesisInfo()

- Gather thesis information when there no theses registered
- Gather thesis information when 1 thesis is already registered
- Gather thesis information when multiple theses are already registered
- //Duplicate student name, thesis title, thesis abstract
- Duplicate date and time
- Date given is within 7 days from the current date
- Date given is within 14 days from the current date
- Date given is longer than 14 days from the current date
- Duplicate supervisor/co-supervisor name
- For each of two thesis readers
    - Duplicate names
- Duplicate name for supervisor/co-supervisor

→ approveOrReject()

- No approvals on theses
- No rejections on theses
- The first thesis reader tries to approve before supervisor/co-supervisors' approval
- The second thesis reader tries to approve before supervisor/co-supervisors' approval
- The defense tries to approve before supervisor/co-supervisors' approval
- Only supervisor/co-supervisors approval on a thesis
- Ony supervisor/co-supervisors' + first reader approval on a thesis
- Only supervisor/co-supervisors' + first reader + second reader's approval on a thesis
- Supervisor/co-supervisor + first reader + second reader + Defense approval on a thesis

→ scheduleDefense()

- Duplicate theses were scheduled
- 0 theses were scheduled at any date and time
- Multiple theses were scheduled at any date and time

4)Data Flow Cases:

→ approveOrReject()

- Supervisor/co-supervisor approves a thesis
- Supervisor/co-supervisor rejects a thesis
- The first reader approves a thesis after supervisor's/co-supervisors' approval
- The first reader rejects a thesis after supervisor's /co-supervisors' approval

- The second reader approves a thesis after supervisor's /co-supervisors' approval
- The second reader rejects a thesis after supervisor's /co-supervisors' approval
- The defense approves a thesis after supervisor's /co-supervisors' + first reader + second reader's approval
- The defense rejects a thesis after supervisor's /co-supervisors' + first reader + second reader's approval

→ scheduleDefense()

- Schedule a thesis after getting all approvals.

PART – 2

Overview:

The problem given is to implement Huffman code with constraints such as using only Arrays/HashMaps/TreeSets. Also, there are three more adaptions considered. Here, in this problem, I am implementing **only the Basic Huffman Code.** We have encode(), decode(), codebook() methods inside the Huffman class and these can be called through the interface.

The entire program has to be implemented in java without importing any data structures such as PriorityQueue, Heap or so. The program was tested on Timberlea.

File and External data:

1)MainUI.java

MainUI contains the switch to take call the method according to the input.

```java
import java.util.Scanner;

public class MainUI {
    //Main method to implement different actions to given by the user
    public static void main( String[] args ) throws Exception {
        Scanner in = new Scanner( System.in );
        System.out.println("Input file name please:");
        String input = in.next();
        System.out.println("Output filename please:");
        String output = in.next();
        System.out.println("Enter 'E' for encode, 'D' for Decode");
        char option = in.next().charAt(0);

        if(!new File(input).exists()) {
            System.out.println("File not found");
            in.close();
            return;
        }
        //Switch case to implament particular encode or decode
        switch(option) {
            case 'E' :
                encode(input, output);
                break;
            case 'D' :
                decode(input, output);
                break;

            default:
                System.out.println("Invalid Option");
        }
        in.close();
    }
    //Method to call encode
    static void encode(String input, String output) throws Exception {
        Huffman filecompressor= new Huffman();
        //System.out.println("First");
        if(filecompressor.encode(input, level: 0 , reset: false, output)) {
            System.out.println("Given file has been encoded Successfully. Please open the output file");
        }
```

2)FileCompresser.java

FileCompressor is an interface given so that the main can adopt to any kind of Huffman with any changes inside it.

```java
package com.A2;

import java.util.Map;

public interface FileCompressor {
    boolean encode(String input_filename, int level, boolean reset,
                   String output_filename);

    boolean decode(String input_filename, String output_filename);

    Map<Character, String> codebook();
}
```

3)Huffman.java

Huffman.java implements filecompressor interface and has all the methods such as encode, decode, codebook.

```java
import java.io.*;
import java.util.HashMap;
import java.util.Map;

public class Huffman implements FileCompressor {
    //Variable Initialization
    private int level;
    private boolean reset;
    private String thisLine;
    private String info = "";
    private String codeToOutput="";
    private String newCh = "&";
    //private  String decodeString = "";
    private String dInfo;
    private String enVal;
    private String dVal= "";
    HashMap<Character, Integer> mapFreqs = new HashMap<>();
    HashMap<Character, String> C1 = new HashMap<>();
    implementHeap H1 = new implementHeap();
    implementHeap h2 = new implementHeap();
    Node hNodes = new Node();
    Node hDNodes = new Node();
    BufferedReader B1;
    BufferedReader B2;
    char[] cArray;
    String[] bits;
    Node r;
    //Default main method
    public static void main(String[] args) {

    }
    //Implemented encode method
    @Override
    public boolean encode(String input_filename, int level, boolean reset, String output_filename) {
        setLevel(level);
        setReset(reset);
        //File reading and storing those characters or string and their frequencies in a map
        try {
          setB1(new BufferedReader(new FileReader(input_filename)));
```

4)implementHeap.java

In this implemetation, I used heap to sort every time an item popped. A heap is implemented for that purpose and all the methods such as construct, pop, insert was included.

```java
package com.A2;

import java.util.Map;

//Implementation of Heap
public class implementHeap {
    public Node[] buildHTree = new Node[1000];
    private int num = -1;
    private int point;
    private int flag;
    Node n;
    Node p1;
    Node thisNode;
    Node k;

    //Methon to coinstruct a minHeap
    public void construct(Map<Character, Integer> M1) {
        //Looping through each entry of map and insert()
        for (Map.Entry<Character, Integer> p : M1.entrySet()) {
            n = new Node(Character.toString(p.getKey()), p.getValue());
            insert(n);
        }
    }
    //Method to insert nodes
    public void insert(Node n) {
        buildHTree[++num] = n;
        setPoint(getNum());
        //Loop through as long as point is not zero and not false of toCompare()
        while(point != 0 && toCompare(getH( s: "parent", getPoint()), buildHTree[getPoint()])) {
            p1 = getH( s: "parent", getPoint());
            //Building a heap tree
            thisNode = buildHTree[getPoint()];
            buildHTree[getPoint() /2 - (getPoint() +1)%2] = thisNode;
            buildHTree[getPoint()] = p1;
            setPoint(getPoint() /2 - (getPoint() +1)%2);
        }
    }
    //Method to pop elements
    public Node pop() {
        Node root = buildHTree[0];
        //If heap has elements - pop
```

5)Node.java

To keep track of left and right of the tree, node class is being created.

```java
package com.A2;

public class Node {
    //Node variables
    private String ch;
    private int freq;
    private Node right;
    private Node left;
    //Node contructor
    public Node(String ch, int freq, Node right, Node left) {
        setCh(ch);
        setFreq(freq);
        setRight(right);
        setLeft(left);
    }
    //Node constructor with two arguments
    public Node(String ch, int freq){
        setCh(ch);
        setFreq(freq);
        setLeft(null);
        setRight(null);
    }
    //Node constructer with two node arguments
    public Node(Node left, Node right){
        setCh("root");
        setFreq(left.getFreq() + right.getFreq());
        setLeft(left);
        setRight(right);
    }
    //Default
    public Node() {

    }

    //Getters and Setters
    public String getCh() { return ch; }

    public void setCh(String ch) { this.ch = ch; }

    public int getFreq() { return freq; }
```

Assumptions:

After the code is encoded, and while decoding we need a separator to distinguish frequency and character. For that, “ ‘ ” single quote is being used.

Key Data Structure:

Key data structure implemented in this program is Heap and binary tree. Also, HashMap and Arrays were used. As there is a constraint not to use data structures other than Arrays/HashMap/TreeSet, Heap has to be implemented from scratch.

Way to run:

Have all the files in a directory.

Step1) javac MainUI.java and java MainUI

Step2) Give input filepath

Step3) Give output filepath

Step4) Enter ‘E’ to encode

Step5) Open the output textfile

Step6) Rerun the program

Step7) Give input filepath

Step8) Give output filepath

Step9) Enter ‘D’ to decode

Step10) Open the output file