CSCI 3901 Assignment 2

Name: Aayushi Rashesh Gandhi
ID: B00890697
Date: 13/10/2021

Problem 1:

First of all, all the details of thesis information will be gathered from the student and validation will be performed in the getDetails() method of the class. Further, the information will be stored in a HashMap which will be passed as a parameter to confirmSchedule() method to get confirmation of the schedule with the supervisor and then the readers and defense chair. Moreover, HashMap will be passed as a parameter to the approveDefense() method to get approval for the defense schedule with the supervisor and then the readers and defense chair. Lastly, the postAnnouncement() method will post the announcement 7 days before the defense.

HashMap<int, List<String>> will be created in main() and will be accessed in all the methods with the help of getters and setters. There are following 4 methods:

1. Public boolean getDetails()
   This method gets all the inputs from the user regarding defense information of the student (String studentName, String dateTime, String title, String abstract, String supervisor, String reader1, String reader2, String defenseChair) and validates them. If the information provided is correct then all the information will be stored in a HashMap and the method will return true otherwise the method will return false.

2. Public boolean confirmSchedule(HashMap<int, List<String>>)
   Confirms schedule with the supervisor and then the reader and defense chair. Returns true if the schedule has been approved and false otherwise.

3. Public boolean approveDefense(HashMap<int, List<String>>)
   Confirms if the defense is at least after 2 weeks. If not, returns false. Otherwise, by sending emails, this method gets approval from the supervisor and if the supervisor confirms the schedule then it takes the approval of readers and the defense chair. If the defense is approved by all three, returns true and false otherwise.

4. Public postAnnouncement(HashMap<int, List<String>>)
   This method posts an announcement 7 days before the defense, if the defense schedule has been approved.

Input validation

1. Public boolean getDetails(studentName, String dateTime, String title, String abstract, String supervisor, String reader1, String reader2, String defenseChair)

Checks following test cases for student’s name, thesis title, thesis abstract, supervisor’s name, reader’s name, defense chair

- Null value and empty string passed as student name
- Null value and empty string passed as thesis title
- Null value and empty string passed as thesis abstract
- Null value and empty string passed as supervisor
- Null value and empty string passed as reader 1 or reader2
- Null value and empty string passed as defense chair
- Duplicate string passed as student name
- Only numbers passed as student name, supervisor, readers or defense chair
- Special character, white spaces, tab passed as student name, supervisor, readers or defense chair
- Negative value passed for date and time
- Date value passed in past tense
- More than 2 characters passed as date DD or month MM
- Exactly 4 characters not passed as year YYYY
- Different date format
- Time format is not 24-Hour time format
- Supervisor, readers and defense chair does not exists

Boundary tests

1. Public boolean getDetails(studentName, String dateTime, String title, String abstract, String supervisor, String reader1, String reader2, String defenseChair)

   Checks following test cases for student’s name, thesis title, thesis abstract, supervisor’s name, reader’s name, defense chair
   - 1 character passed
   - Too many characters passed
   - 1 name passed for a supervisor
   - Multiple names passed for supervisor or co-supervisors

2. Public boolean confirmSchedule(HashMap<int, List<String>>)
   - Get confirmation of schedule when no other confirmations are pending with the supervisor
   - Get confirmation of schedule when only 1 confirmation is pending with the supervisor
   - Get confirmation of schedule when many confirmations are pending with the supervisor

3. Public boolean approveDefense(HashMap<int, List<String>>)
   - Defense is in exactly 2 weeks
   - Take approval when no other defense approvals are pending with the supervisor
   - Take approval when 1 defense approval is pending with the supervisor
   - Take approval when many defense approvals are pending with the supervisor
   - Take approval when no other defense approvals are pending with the readers and defense chair

    - Take approval when 1 defense approval is pending with the readers and defense chair
    - Take approval when many defense approvals are pending with the readers and defense chair

4. Public postAnnouncement(HashMap<int, List<String>>)
    - One day is left for the system to post the announcement
    - Defense is in exactly 7 days

Control flow tests

1. Public boolean getDetails(studentName, String dateTime, String title, String abstract, String supervisor, String reader1, String reader2, String defenseChair)
    - Create a defense schedule when no defense has already been scheduled
    - Create a defense schedule when only 1 defense has already been scheduled
    - Create a defense schedule when many defenses have already been scheduled
    - Create a new schedule for a student with an existing defense schedule at some other time

2. Public boolean confirmSchedule(HashMap<int, List<String>>)
    - Confirmation of schedule disapproved by the supervisor
    - Confirmation of schedule approved by the supervisor
    - Confirmation of schedule disapproved by readers
    - Confirmation of schedule approved by readers but not the defense chair
    - Confirmation of schedule approved by readers and the defense chair

3. Public boolean approveDefense(HashMap<int, List<String>>)
    - Defense is scheduled when more than 2 weeks are left
    - Defense disapproved by the supervisor
    - Defense approved by the supervisor
    - Late response from the supervisor
    - Defense disapproved by one of the readers
    - Defense approved by the readers
    - Late response from the supervisor
    - Defense disapproved by the defense chair
    - Defense approved by defense chair
    - Late response from the supervisor

Data flow tests

1. Public boolean confirmSchedule(HashMap<int, List<String>>)
    - Getting confirmation of schedule on a date and time which overlaps with other already confirmed defense schedule

2. Public boolean approveDefense(HashMap<int, List<String>>)
    - Getting approval for an already approved defense
    - Getting approval for an already disapproved defense

Problem 2:

Overview
The program develops Huffman code. It takes an input string, encodes it and then decodes the string to its original form. The file is encoded with the help of adaptive huffman coding. As the file is encoded dynamically, only one pass is required through the file and therefore it is efficient.

Efficiency

- If you chose a simple data structure to ensure that you can complete the work then the mark is on a very short description of why you chose this data structure plus a description of the data structure that you would have chosen (given more time / experience) and why?

  I chose TreeSet to sort the characters according to their frequencies because in TreeSet all the elements remain in sorted order. However, I would have chosen Priority Queue and would have built it from scratch, if I had more experience and hands-on knowledge of using Priority Queues on low levels because the root value in priority queue always yields smallest or largest value depending on the situation. Additionally, Priority Queue provides the smallest/largest element in O(1) time whereas TreeSet takes O(logN) time.

Files and external data
In total, there are 5 files. mainUI.java contains the main public method and is the starting point of the code. FileCompressor.java contains all the key methods and huffmanTree.java creates the tree and rebuilds it. Additionally, there are 3 different text files.

1. test.txt : contains the string that needs to be encoded
2. encode_output.txt : contains encoded string of the given input string
3. decode_output.txt : contains decoded string of the above created encoded string

Data structures and their relations to each other

1. HashMap<Integer, huffmanNode> finalRebuildEncodeMap : Contains final rebuild trees of all the stages.
2. HashMap<String, Integer> periodicFrequencyMap : Contains characters and its frequencies.
3. HashMap<Character, String> codeBookMap : Contains character and its path from the final rebuild tree.

Key algorithms and design elements

In FileCompressor.java file, there are 7 key methods and one interface:

1. **public String[] sortMap(HashMap<String, Integer> periodicFrequencyMap)**
   Takes HashMap(character , frequency) as an input through parameter passing and returns a sorted array of key values of the HashMap according to the increasing order of the frequency value obtained from the value of the HashMap.

2. **public huffmanNode insertAtRightMostNode(huffmanNode hn1, String character, int value, String rootCharName)**

As the name suggests, this method inserts a given node to the rightmost node of the tree. Through parameter passing, it takes a tree, the character and its frequency which needs to be added and the name of the new node with combined frequency of left and right nodes as an input. This method uses recursion. It continuously traverses the right node of the tree by calling the method recursively until a leaf node is encountered. Once a leaf node is obtained, the recursion stops and the existing node is made as the left child and the new character is made as the right child of the tree.

3. **public boolean nodePath(huffmanNode node, String key, String returnString)**
   This method takes a tree, a key value (the character whose path needs to be obtained) and an empty string(returnString: where the final path will be stored) as an input. There are 4 cases:
    1. If the key is null, then the method returns false
    2. If key is present at the root of the tree, then the method returns empty string as an output path
    3. The method runs recursively to the left of the tree and if key is present at the left of the current root node, then the method appends 0 to the path string
    4. The method runs recursively to the right of the tree and if key is present at the right of the current root node, then the method appends 1 to the path string

   Thus, the final path is obtained as a combination of 0's and 1's.

4. **public String searchNode(huffmanNode decodeMapNode, String expression)**
   This method takes tree and path of 0's and 1's as an input and returns the character which is present at the given path in the tree. Method traverses the path character by character and if it encounters 0 then it means, the left route needs to be traversed otherwise, the right route needs to be traversed. By following this technique, the pointer finally reaches the node where the character is present and the method returns its value.

5. **public boolean encode(String input_filename, int level, boolean reset, String output_filename)**
   This is the most important method where encoding of the given string is performed.
    1. This method reads the file character by character.
    2. It will check whether the character has come for the first time or does it already exist.
    3. If the character already exists then only the frequency of the character is updated by 1. Consequently, it computes the path as a combination of 0's and 1's at the node where the character exists. This string is used to encode the input string.
    4. If it has been encountered for the first time then the character is inserted to the frequency map and its frequency is updated to 1 and value of "new character nc" is increased by 1.
        a. The new character is added to the rightmost node of the tree and it computes the updated path of "nc".
        b. It computes the path of the string by appending the path of "nc" to the character itself.

5. During the first adaptation, the right side of the tree grows quickly. Therefore, the above steps run for 2^n times and then after completion of the loop, the tree is periodically rebuilt. After completion of rebuild, the updated rebuild tree is passed for the first adaptation for the next loop of 2^(n+1) and so on.
6. Thus, the final encoded string is obtained.

6. **public boolean decode(String input_filename, String output_filename)**
   This method decodes the encoded input string. FIrst of all, the method splits the given input encoded string by “ ” (space). If a character is already present in the expression (when the character is encountered for the first time), then it directly returns the character as the decoded value and adds the character to the rightmost node of the tree. Otherwise, when the character is not encountered for the first time, the path of 0’s and 1’s is obtained. By traversing the tree with the help of searchNode() method, the decoded value of that particular character is obtained. By this way decoding is performed and the input string is obtained in its normal form again.

7. **public Map<Character, String> codebook()**
   This method is used to display the path of a particular character after the final rebuild is performed for the whole string.

In huffmanTree.java file, there is 1 key method:

1. **public huffmanNode rebuildHuffmanTree(HashMap<String, Integer> frequencyMap)**
   This method rebuilds the tree after completion of first adaptation. It sorts the frequencies and takes two smallest entries from the list and builds the tree from bottom.

Limitations
This program does not work for reset=true.