CSCI 3901 Assignment 2 - Part 1: Test Case writing

Objective : Get practice in writing test cases.

Report by:

- Shivam Bhojani (B00895637) - shivam.bhojani@dal.ca

Input Validation (generally, tests on bad input data for which you shouldn't crash)

Method for taking user input will return "Request Accepted" for success scenarios and it will return "Request failed" in case of any failure in data input.

- ***public String defenceRequest (String StudentName, Int Date, String thesisTitle, String Abstract, String supervisor, String thesisReader1, String thesisReader2, String defenceChair)***

    - Null value passed as StudentName >> Request Failed
    - Empty String passed as StudentName >> Request Failed
    - Numeric value passed as StudentName. >> Request Accepted
    - String value passed as Studentname >> Request Accepted
    - String value with special characters passed as StudentName >> Request Accepted

  (Taking Date value in three parts as year, month and date all in Integers >> combining them with Date function)

    - Null value passed as Year >> Request Failed
    - Empty value passed as Year >> Request Failed
    - Numeric value passed as Year. >> Request Accepted
    - String value passed as Year >> Request Failed
    - Null value passed as month >> Request Failed
    - Empty value passed as month >> Request Failed
    - Numeric value passed as month . >> Request Accepted
    - String value passed as month >> Request Failed
    - Null value passed as date>> Request Failed
    - Empty value passed as date>> Request Failed
    - Numeric value passed as date. >> Request Accepted
    - String value passed as date>> Request Failed

    - Null value passed as thesisTitle >> Request Failed
    - Empty String passed as thesisTitle >> Request Failed
    - Numeric value passed as thesisTitle . >> Request Accepted
    - String value passed as thesisTitle >> Request Accepted
    - String value with special characters passed as thesisTitle >> Request Accepted

    - Null value passed as Abstract >> Request Failed
    - Empty String passed as Abstract >> Request Failed
    - Numeric value passed as Abstract. >> Request Accepted
    - String value passed as Abstract >> Request Accepted
    - String value with special characters passed as Abstract >> Request Accepted

    - Null value passed as supervisor>> Request Failed
    - Empty String passed as supervisor>> Request Failed
    - Numeric value passed as supervisor. >> Request Accepted

- String value passed as supervisor>> Request Accepted
- String value with special characters passed as supervisor>> Request Accepted

- Null value passed as thesisReader1>> Request Failed
- Empty String passed as thesisReader1>> Request Failed
- Numeric value passed as thesisReader1. >> Request Accepted
- String value passed as thesisReader1>> Request Accepted
- String value with special characters passed as thesisReader1>> Request Accepted

- Null value passed as thesisReader2>> Request Failed
- Empty String passed as thesisReader2>> Request Failed
- Numeric value passed as thesisReader2 >> Request Accepted
- String value passed as thesisReader2>> Request Accepted
- String value with special characters passed as thesisReader2>> Request Accepted

- Null value passed as defenceChair >> Request Failed
- Empty String passed as defenceChair >> Request Failed
- Numeric value passed as defenceChair >> Request Accepted
- String value passed as defenceChair >> Request Accepted
- String value with special characters passed as defenceChair >> Request Accepted

Boundary Cases (tests at the edge of inputs or problem structures)

- 1 Character as StudentName >> Request Accepted
- 0 Character as Year, Month and Date value >> Request Failed
- 1 Character as thesisTitle >> Request Accepted
- 1 Character as Abstract >> Request Accepted
- 1 Character as supervisor >> Request Accepted
- 1 Character as thesisReader1 >> Request Accepted
- 1 Character as thesisReader2 >> Request Accepted
- 1 Character as defenceChair >> Request Accepted

Control Flow Cases (tests of the core operations)

- Duplicate entry for all the mandatory fields. (In other words, making 2 request with the same details for the fields) >> Request Failed
- With duplicate Student Name and other values are distinct >> Request Accepted
- With duplicate Year, Month and Date values and other values are distinct >> Request Accepted
- With duplicate thesisTitle and other values are distinct >> Request Failed (Logically, thesis title should be unique for all Students)
- With duplicate Abstract and other values are distinct >> Request Failed (Logically, thesis abstract should be unique for all Students)
- With a duplicate supervisor and other values are distinct >> Request Accepted

- With duplicate thesisReader1 and other values are distinct >> Request Accepted
- With duplicate thesisReader2 and other values are distinct >> Request Accepted
- With duplicate defenceChair and other values are distinct >> Request Accepted

- Create Defence Request when there are no previous request done >> Request Accepted
- Create Defence Request when there already many request done >> Request Accepted
- Create Defence Request when no request are approved >> Request Accepted
- Create Defence Request when there are many request approved >> Request Accepted
- Check the email content which is sent to all the approvers >> Request Accepted

- Check when supervisor rejects the approval >> Request Declined
- Check when reader rejects the approval >> Request Declined
- Check when defence chair rejects the approval >> Request Declined

- Check when the defence request date is more than 21 days from current date >> Request Accepted
- Check when the defence request date is less to 21 days from current date
- Check when no other request is done for the particular date >> Request Accepted
- Check when all the request are done for the same particular date >> Request Accepted

- Check when supervisor approves schedule, then Readers and then defence chair >> Request Accepted

- Check schedule when all the approvers has successfully accepted the request >> Request Accepted
- Check schedule when supervisor and reader has approved but defence chair has rejected the request. >> Request Declined
- Check schedule when supervisor has approved but reader has rejected the request. >> Request Declined

Data Flow Cases (tests around the order in which things are done)

- Check when Readers tries to approve schedule before supervisor
- Check when defence chair tries to approve schedule before Readers
- Check when defence chair tries to approve schedule before supervisor

- Check when reader tries to reject the approval before supervisor
- Check when defence chair tries to reject the approval before reader
- Check when the defence chair tries to reject the approval before the supervisor.

- Check schedule when there are no request
- Check schedule when there no approved request
- Check schedule when all the requests are rejected

- Check the schedule when only where only supervisor has approved and other approvals are left. >> particular request should not be shown.
- Check the schedule when only supervisor and Readers has approved and other approvals are left. >> particular request should not be shown.

CSCI 3901 Assignment 2 - Part 2: Huffman Encoding/Decoding

Objective : Implement a data structure from basic objects.

Code & Report by:

- Shivam Bhojani (B00895637) - shivam.bhojani@dal.ca

Solution Files Attached:

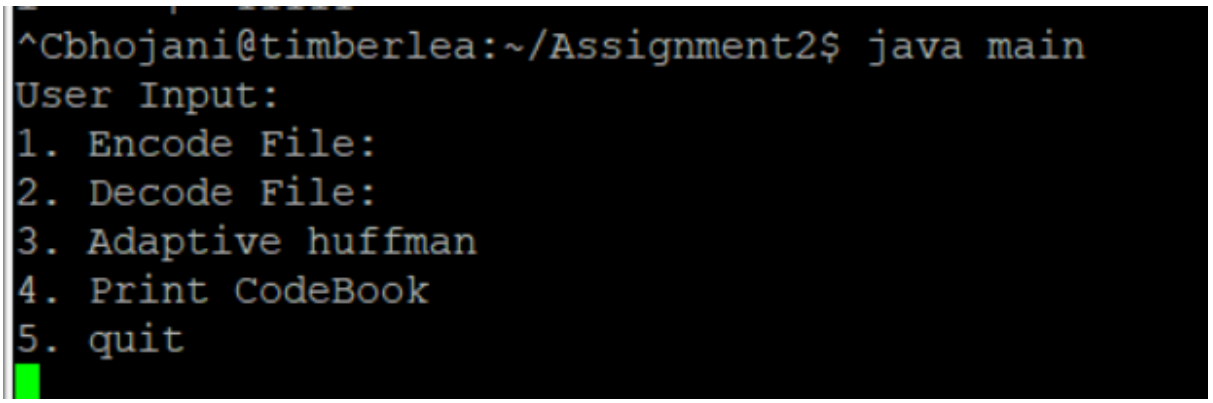
1. main.java
2. huffmanNode.java
3. Huffman.java
4. getSmallest.java
5. getEncodedValues.java
6. fileWriter.java
7. FileCompressor.java

What each file and its method does:

mainUi.java

1. Takes user Command and user inputs for various tasks such as:
    - Take the user command for Encode/Decode/Code Book.
    - Take the filename from which users want to read the data and encode it.
    - Take the filename in which the user wants to write the encoded bits.

    - User input of filename, which needs to be decoded.
    - User Input of filename which stores the decoded data.



Huffman.java

public boolean encode(String input_filename, int level, boolean reset, String output_filename)

- This method will be triggered by main.java
- String input_filename will take the input from the provided filename and start encoding using huffman logic
- Encoded bits are stored in text file with name mentioned in String output_filename
- This method is using Treeset, Hashmap, for loops and conditional statements to encode the data.

boolean decode(String input_filename, String output_filename)

- This method will be triggered by main.java when user has successfully encoded the particular file and now, tries to decode the date
- String input_filename will be used to get the encoded bits one by one and the binary tree will be used to decode the date.
- After reaching every leaf node, the current pointer will again target to root node and the decoding will start again for next bits.
- The decoded date will be stored in separate text file names with String output_filename

public void printCodeBook()

- This method will be used to print the Code Book.
- Code book is basically the table format representation of code words for each and every character in the particular file, including spaces.

getEncodedValues.java

- It is used to store the encoded values in HashMap
- It is used to print the codeBook with each character in the data.

getSmallest.java

- This file is used to get the smallest key value from the HashMap

huffmanNode.java

- This java file is always to create a new node for Binary tree.

fileWriter.java

- This file is used to write files

For Example:

- To write encoded data in given file name
- To write decoded data in given file name

Data Structures and Implementation Logic

HashMap

- HashMap is used in the code to store the characters and their frequency
- If HashMap does not contain that particular character in its Key value, then the code just puts the data into the Hashmap
- If the character already exists then, the frequence is just updated and again puts the data in hashmap

TreeSet

- Each unique character stored in HashMap is then sorted by the character frequency and one by one, the data is sorted in Treeset.
- At the time of storing data in treeset, a comparator class is used to sort the data according to frequency and if the frequency is the same for more than one character then that particular data is again sorted in lexicographic manner.

Logic/Implementation

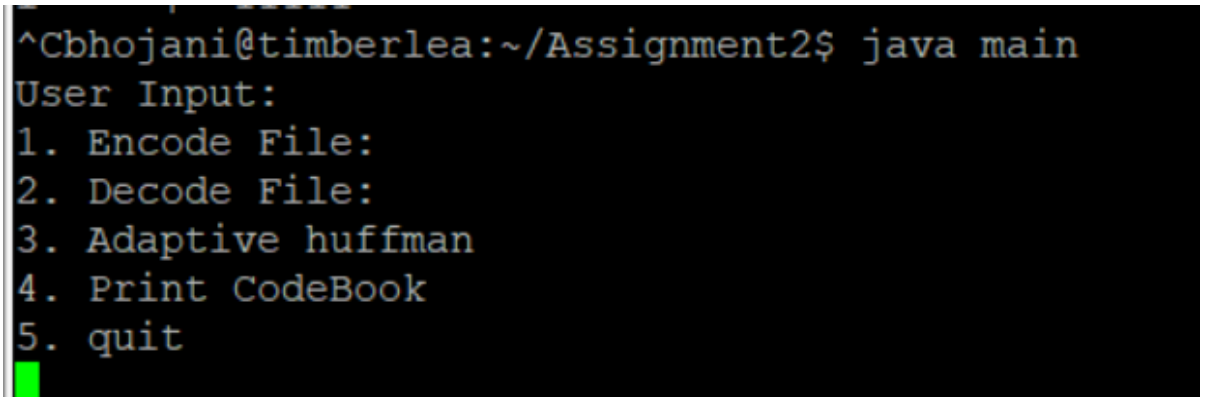
1. Read data from Input file and store it into String
2. Store each character from that String into HashMap
3. Map<String, huffmanNode> map = new HashMap<String, huffmanNode>();
4. Here, if a character is new or not there in hashmap, then code makes a new entry in Hashmap, otherwise it updates the frequency of the existing data.
5. From HashMap, one by one pick the key with lowest frequency and store it into a treeset.
6. Remove that particular data from Hashmap
7. While storing the object values in treeset, we sort them by frequency and then by lexicographical order.
8. Create root node, left node and right node
9. Pull out first element from treeset and assign it to left node
10. Pull out second element from treeset and assign it to right node
11. Assign root's left and root's right with the left and right node respectively.
12. Again add the root node into tree (which will go through the sorting logic again)
13. Repeat Step 8-12, till the treeset has one element at the end.

14. Once the encoding is complete, make note of our root node.

15. In Decoding, start reading each and every 0s and 1s from the encoded file and traverse through the tree depending on 0 and 1
16. If 0, then root = root.left
17. If 1, then root = root.right
18. Repeat 16-17, until we find a leaf node whose left child and right child are null.

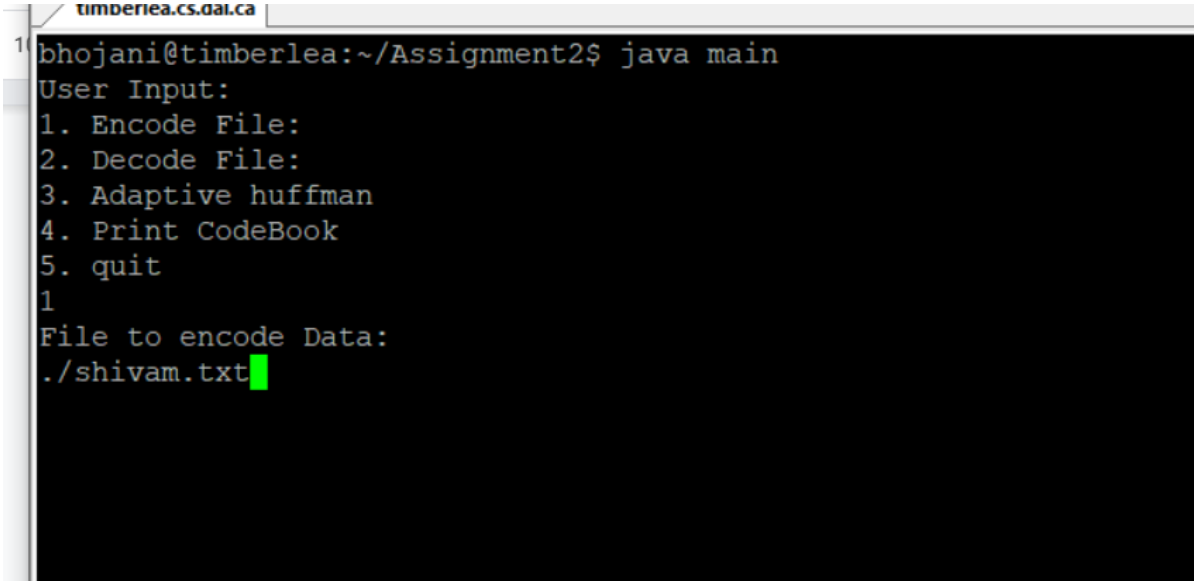
Code Execution

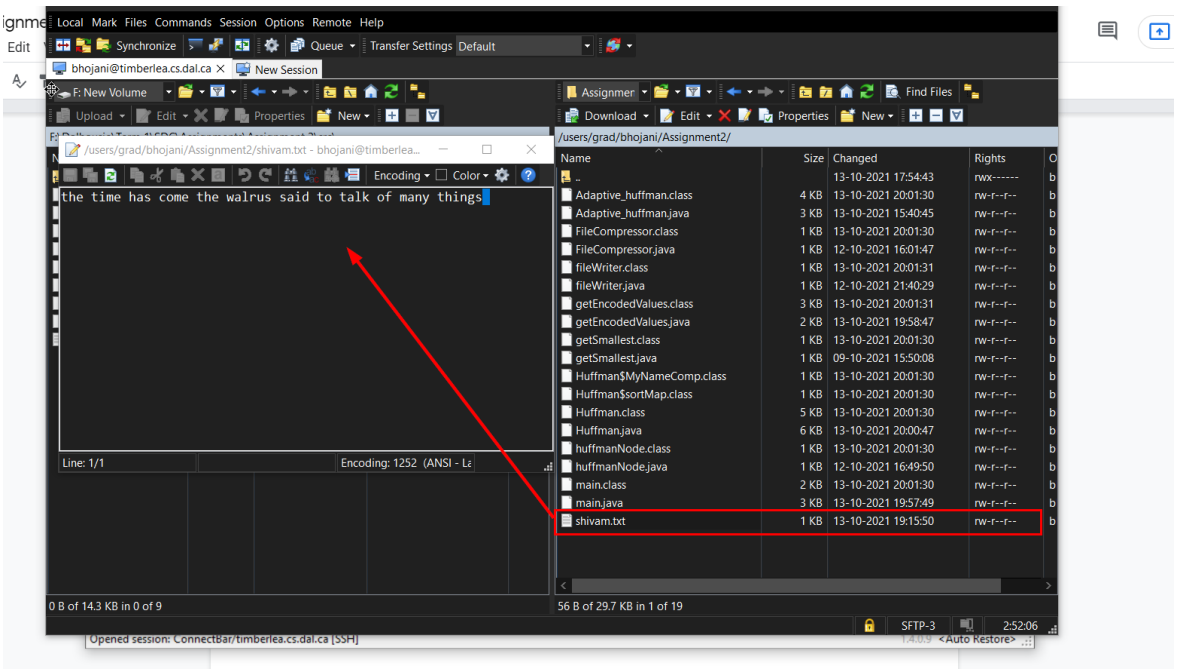
1. After compiling the main.java file, execution the file with command **java main**



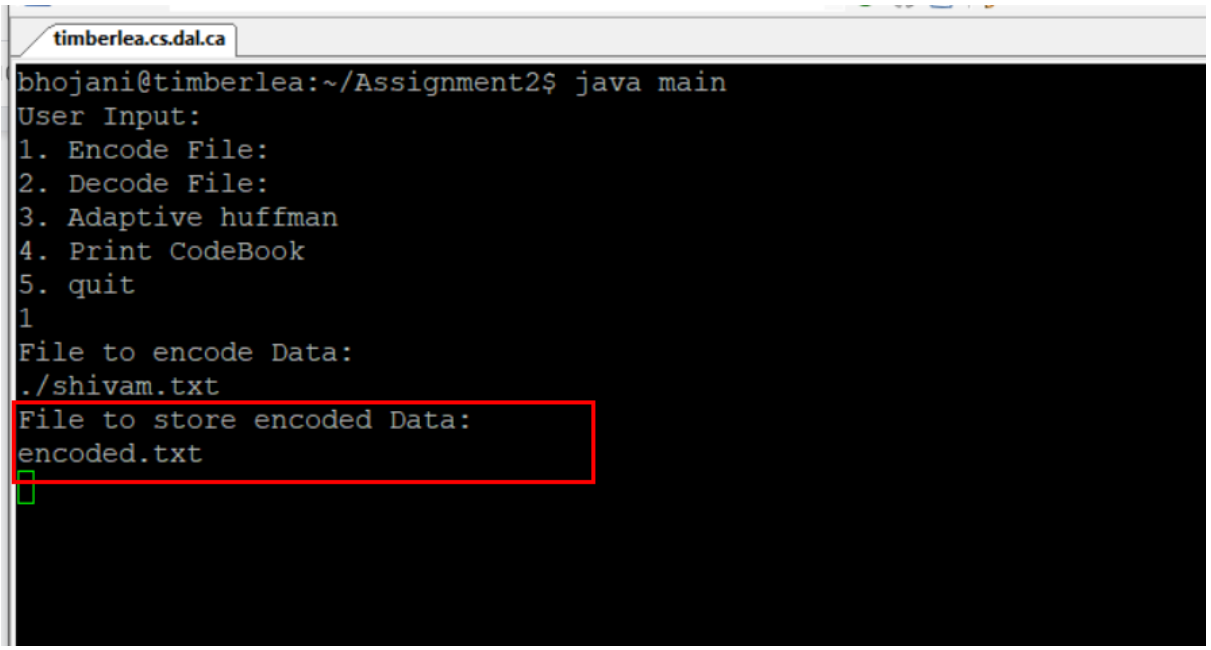
2. Choose first option but entering 1

3. Enter the filename. In this case, we have a file called shivam.txt

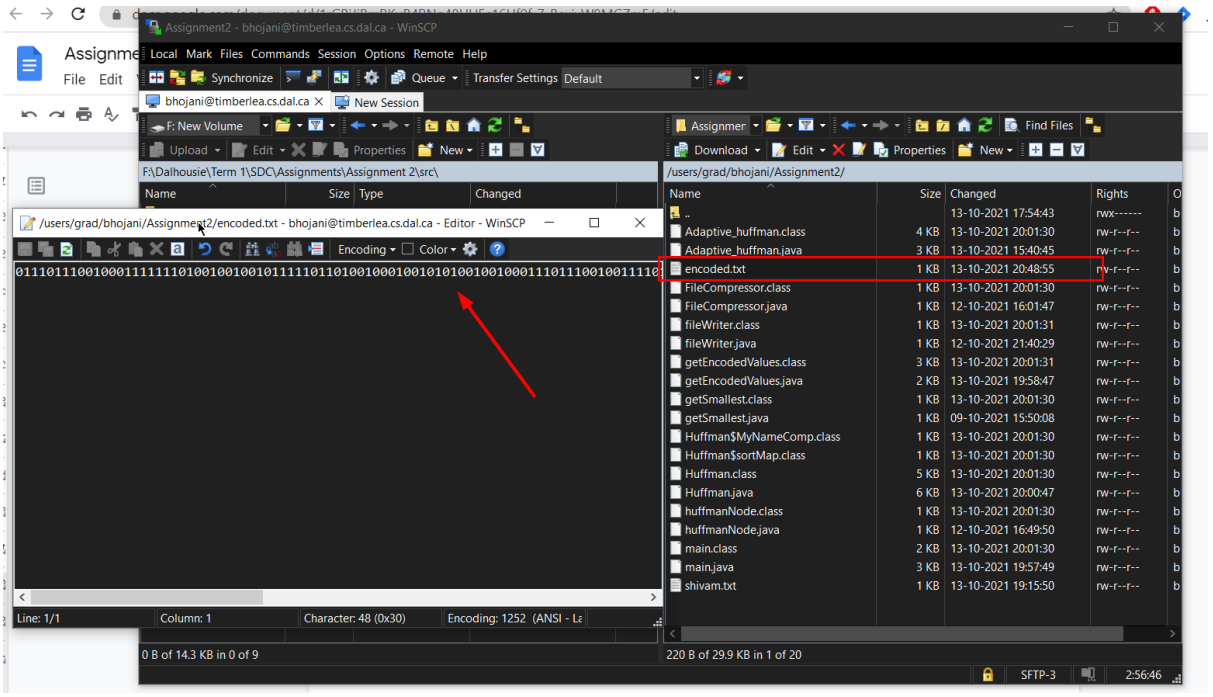




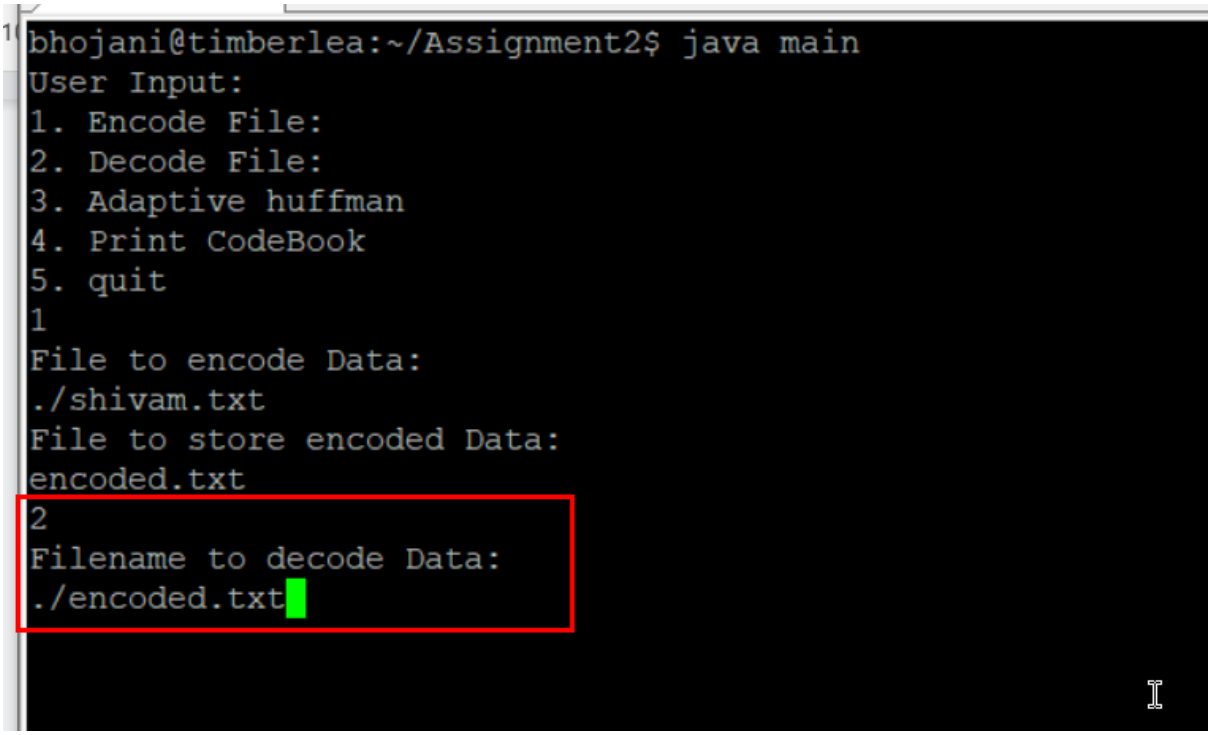
4. Next the code will ask the user to enter the filename, where the user wants to store the encoded data.
   In this case, we have entered the filename as **encoded.txt**



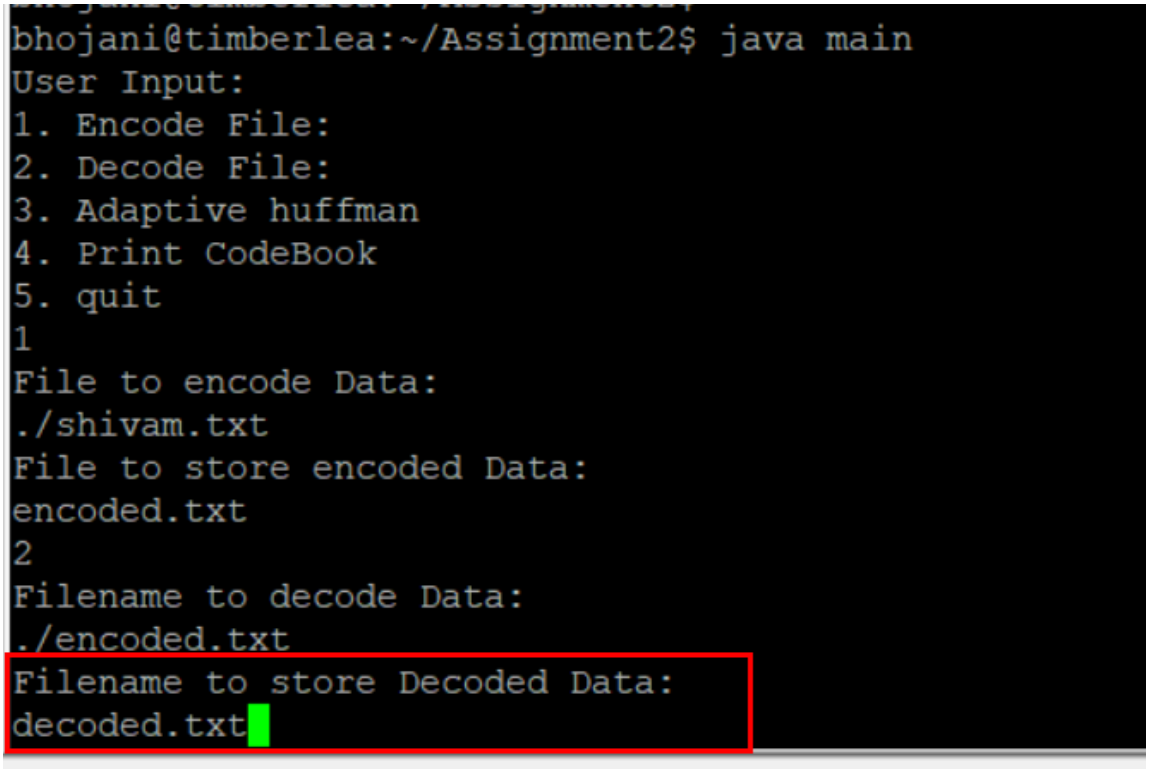
5. Now at the backend, encoding is already completed and a file with name encoded is created.



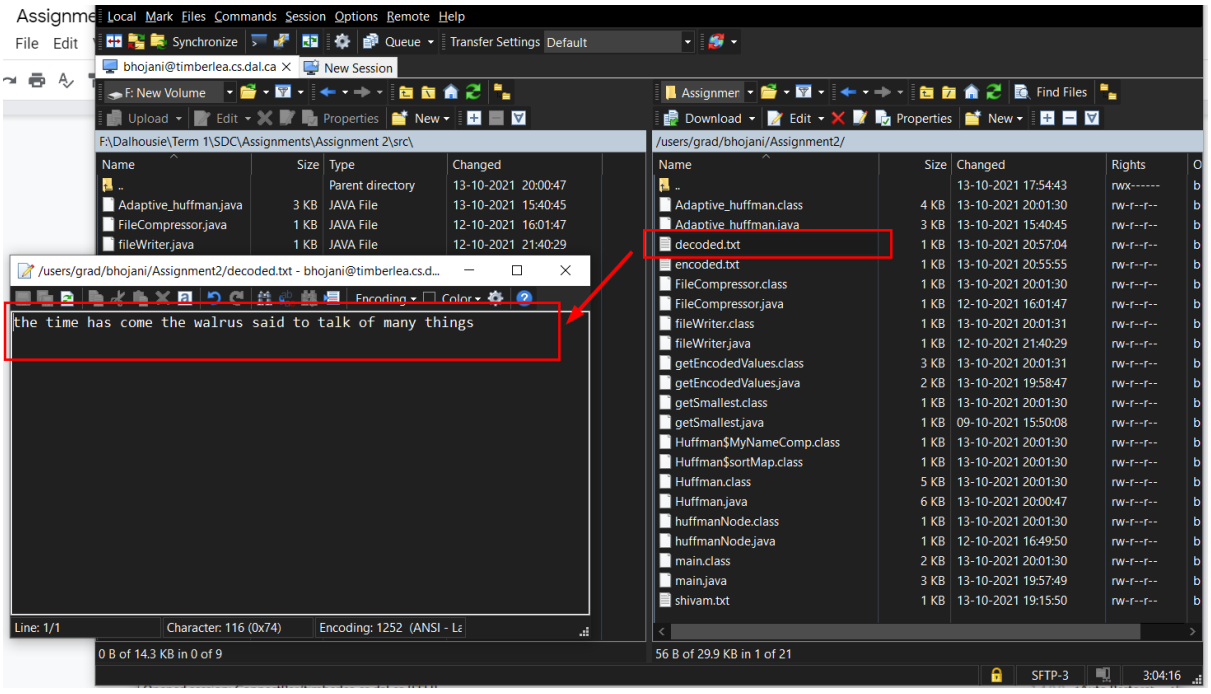
6. Now the user will select the command to decode the new created file with the same Binary tree running in current code.



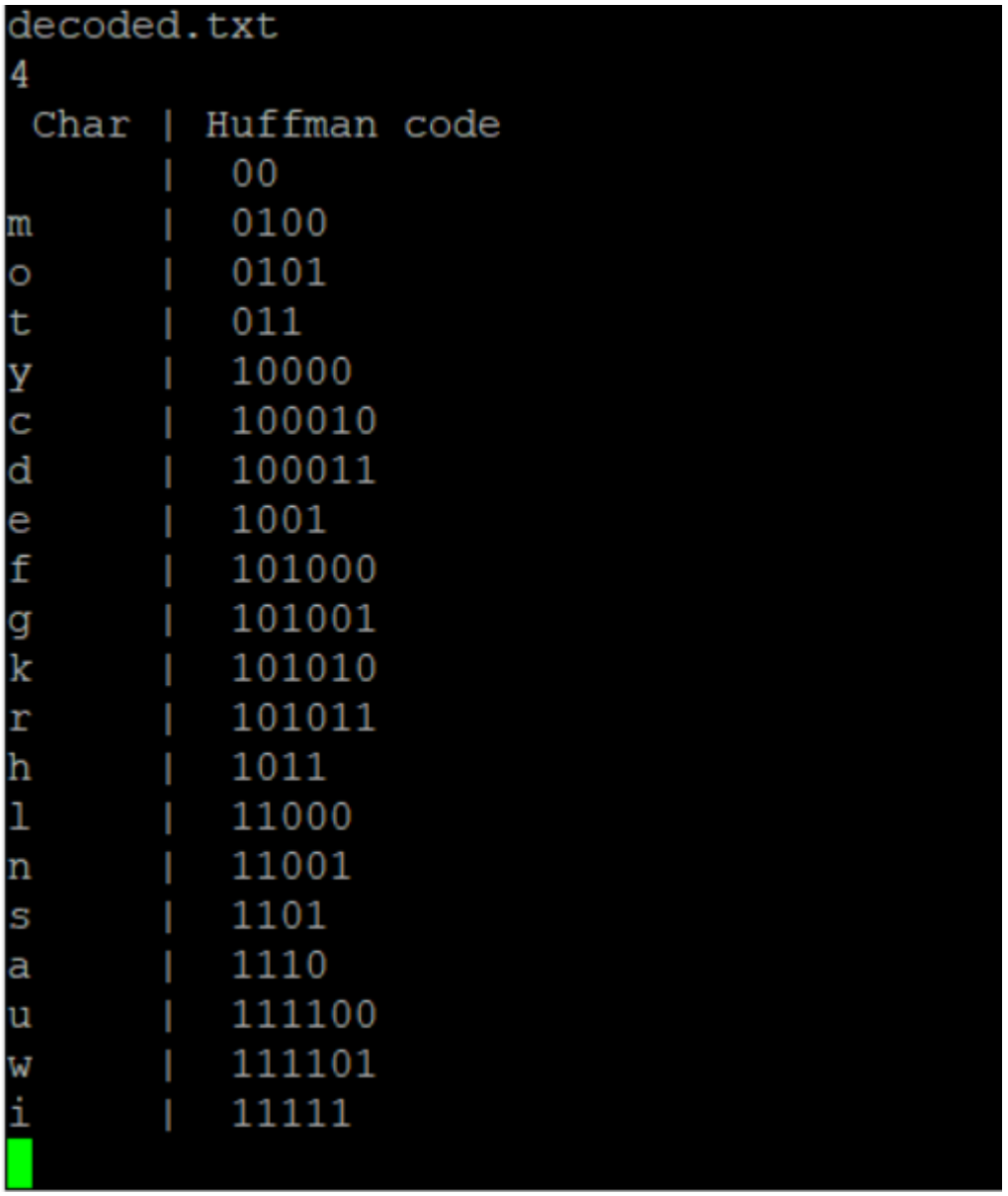
7. Now the user will enter the filename to store the decoded data.



8. At the BackEnd, the file is already decoded and its stored in the file named as decoded.txt



9. User can also print the code book by choosing the appropriate option and it shows the data as bellow



References:

[1] “Huffman Coding Algorithm,” *Programiz.com*, 2021. [Online]. Available: https://www.programiz.com/dsa/huffman-coding. [Accessed: 13-Oct-2021]

[2] “Huffman Coding | Greedy Algo-3 - GeeksforGeeks,” *GeeksforGeeks*, 03-Nov-2012. [Online]. Available: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/. [Accessed: 14-Oct-2021]