Matt Weaver

Data Structures

Final Project Documentation

**Purpose:** Design and implement a program that will compress and uncompress an ASCII text file using the Huffman encoding algorithm of your own design.

**Classes:** This program contains six classes. Mainprogram (Main program), binarytree (support class) charFreq (support class), Node (supporting class), Fileoperations (supporting class),. All support classes will be defined below. The main program class will be defined in description of the pseudocode.

**Note**-the pseudocode for the compression and decompression algorithms are defined as smaller or individual methods under their corresponding heading.

|  |
| --- |
| **charFreq** |
| -character : char  -count : long |
| + charFreq (character, count: char, long)-constructor  + charFreq (character: char )-constructor  +getCharacter() : char  +setCharacter(in character : char) : void  +getCount() : long  +setCount(in count : long) : void  +Increment() : void  + equals (object o): Boolean  + compareto (object o): int  + frequencytostring (): string  + sort (LinkedList<charFreq> letters):LinkedList<charFreq>  -swap (frequency: [] charFreq, one: int, two: int): void |

|  |
| --- |
| **Node** |
| -Node <T>: left  -Node <T>: right  - T: element |
| + Node (T e)-constructor  + getElement () : T  +setElement (e: T) : void  + getLeft ():Node <T>  + setLeft (n: Node<T >): void  + getRight ():Node <T>  + setRight (n: Node<T): void  + isleaf (): Boolean |

|  |
| --- |
| **Binarytree** |
| -Node <T>: root  -Node <T >: current  -int: size  + Location (leftChild, rightChild, parent, root): static enum |
| + BinaryTree ()-constructor  + BinaryTree(n: Node<T >): constructor  + getRoot ():Node <T>  + getCurrent ():Node <T>  + getSize (): int  + destroy(n: Node<T >): void  + leaf (): Boolean  -FindParent (n: Node<T >): Node <T>  + movedTo (locate: location): Boolean  + add (Value: T, locate: location): Boolean  + update (e: T) : void  + inorder(p: Node<T >): void  + postorder(p: Node<T >): void  + preorder(p: Node<T >): void  + Empty (): Boolean  + buildencodingtree (letters: linkedlist <charFreq>, leafs: linkedlist<Node>): binary tree  + builddecodingtree (encodedletters: linkedlist <charFreq> ): binary tree  + createtable (p: Node<T>, Codes: StringBuffer, encodingtree: binarytree):StringBuffer |

|  |
| --- |
| **Fileoperations** |
|  |
| + Readfile (character: charFreq, available: Boolean, letters:LinkedList<charFreq>,index:int): void  + writefile (letters:LinkedList<charFreq>):void  + writetable (Codes: StringBuffer): void  readtable (encodingcharacter: encodingchar, encodedletters:LinkedList<encodingchar>  compression (encodedcharacter encodingchar, exist: Boolean, encodedletters:LinkedList<encodingchar> encodedindex:int): void  decompression (p: Node<T>, decodingtree binarytree):  + frequencytostring (): string  + binarytostring (): string |

|  |
| --- |
| **encodingChar** |
| -character : char  -encodingnumber: string |
| + encodingChar (character, encodingnumber: char, string)-constructor  + encodingChar's (character: char )-constructor  +getCharacter() : char  +setCharacter(in character : char) : void  +getEncodingnumber () : string  + setEncodingnumber (inencodingnumber string) : void  + equals (object o): Boolean  + compareto (object o): int  + binarytostring (): string  + sort (LinkedList<encodingChar> encodedletters):LinkedList<charFreq>  -swap (frequency: []encodingChar, one: int, two: int): void |

**Method Descriptions for charFreq**

1. **charFreq**

* initialize character and count to null and zero respectively

1. **charFreq**

* initialize character to null

1. **getcharacter**

* return character

1. **setcharacter**

* character=incharacter

1. **getcount**

* return count

1. **setCount**

* count=incount
* guard against count less than zero

1. **increment**

* increase count by one

1. **equals**

* sees if charFreq object equals object o
* returns equals

1. **compareto**

* compares charFreq to object o if charFreq is greater than object o compare equals -1
* compares charFreq to object o if charFreq is less than object o compare equals 1
* compares charFreq to object o if charFreq is equal to object o compare equals 0
* returns compare

1. **tostring**

* format output into two columns, character column and count column
* returns S

1. **Sort**

* converts a linked list into an array
* bubble sorts charFreq objects by frequency
* converts array back into a linked list
* returns letters

1. **swap**

* interchanges two elements in an array for the bubble sort in the method above

**Method Descriptions for Node**

1. **Node**

* initialize right, left, and element to null , null and e respectively

1. **getElement**

* return element

1. **setElement**

* this.element = e

1. **getLeft**

* return left

1. **setleft**

* this.left =n

1. **getRight**

* return right

1. **setRight**

* this.right =n

1. **isleaf**

* return left and right node equal to null

**Method Descriptions for Binary Tree**

1. **BinaryTree**

* initialize root, current , and size to null , null and 0 respectively

1. **BinaryTree**

initialize root, current , and size to n ,root and 0 respectively

1. **getRoot**

* return root

1. **getCurrent**

* return current

1. **getSize**

* return size

1. **Destroy**

* remove left and right node
* n= null
* decrement size

1. **leaf**

* return current left and current right node equal to null

1. **findParent**

* find predecessor of current node
* if node is on the left node becomes the left
* if noted on the right it is added
* else nodes removed
* return n

1. **movedTo**

* moved to the user indicated node
* make the indicated node the current node
* options are right, left, parent, root
* return found

1. **add**

* if the location specified by the user already has a value do not add the node
* else create a new node and added to the tree at location the user specifies
* increment size
* return inserted

1. **update**

* set current element to user-specified value

1. **inorder**

* put the notes of the binary tree in ascending order
* increment size

1. **preorder**

* put the notes of the binary tree in pre-order
* increment size

1. **postorder**

* put the notes of the binary tree in post-order
* increment size

1. **Empty**

* return root equal to null

1. **build encoding tree**

* loop through letters
* if the element at I is not null, then
* create a new node using the value at I.
* Add new node to leafs.
* While leafs sizes greater than one then.
* Pull the two smallest notes from leafs.
* Set new node left to the first one removed.
* Set new node, right to the second one removed.
* Add new node to leafs.
* Create a binary tree object and add new node as root of binary tree.

1. **Build decoding tree**

* Create binary tree object with node top being the root
* loop through encoded letters link list
* getting the encoding number from each element
* have line equal the encoding number of each object
* loop through the line
* if the character at I is zero then
* if current left equals null then
* create a new node
* current left equals node
* current equals current left
* else if current right equals null then
* create a new node
* current right equals node
* current equals current right
* if left and right equal null
* print the character at the node

1. **Createtable**

* if p not null, then
* add a zero to codes.
* Call create table method on p.left, codes and encoding tree
* remove the previous character from codes
* add a one to codes.
* Call create table method on p.right, codes and encoding tree
* remove the previous character from codes
* return codes

**Method Descriptions for file operations**

1. **readfile**

* get user directory.
* Use buffered reader to read the correct file.
* While not at end of file.
* Read the first character from file.
* If letters contains character than.
* Get the index of that character.
* Increment the count at that character.
* Else add character to letters with a count of one
* read next character from file.
* close file

1. **writefile**

* get user directory.
* Use print writer to print out to the desired file.
* Format heading of file to character, count.
* Insert blank line
* Loop through letters, printing out the character and count at I
* flush print writer.
* close file
* give the location of written file

1. **writetable**

* convert codes to a string.
* Convert string into a string array called binary using the "," character as the separator.
* get user directory.
* Use print writer to print out to the desired file.
* Format heading of file to character, code.
* Insert blank line
* Loop through binary printing out the character and code at I
* close file
* give the location of written file.

1. **Readtable**

* get user directory.
* Use buffered reader to read the correct file.
* While not at end of file.
* Read the first character from file
* if the character from file is a character convert to a character
* else converted to a string
* added to a string
* creating new encoded character object from the character and string
* add new encoded character object to encoded letters linked list
* read next character from file.
* close file

1. **tostringfrequency**

* format output into two columns, character column and count column
* returns S

1. **tostringbinary**

* format output into two columns, character column and encodingnumber column
* returns S

1. **compression**

* get user directory.
* Use buffered reader to read the correct file.
* While not at end of file.
* Read the first character from file.
* Convert character value into a character called original
* If encoded letters contains original than.
* Get the index of that original
* set the encoded character equal to the index value
* get the encoding number from the encoding character
* for each encoding string while I is greater than or equal to zero and less than or equal to seven
* if the character at I is a one set the corresponding bit
* I is greater than seven then
* and the bit to a linked list
* subject eight characters from the encoding string
* if the length of the string results in zero then add the byte to link list
* reset the byte to zero
* read next character from file
* else added to the length of the next string
* reset byte to zero
* read next character from file
* Use print writer to print out to the desired file.
* Loop through bites, printing out the byte at I
* flush print writer.
* close file
* give the location of written file

1. **Decompression**

* if p not null, then
* Call decompression method on p.left, and decoding tree
* add the element at p to finish process link list
* Call decompression method on p.right, and decoding tree
* Use print writer to print out to the desired file.
* Loop through finish process printing out the element at I
* close file
* give the location of written file.

**Method Descriptions for encodingChar**

1. **encodingChar**

* initialize character and encodingnumber to null and empty respectively

1. **encodingChar**

* initialize character to null

1. **getcharacter**

* return character

1. **setcharacter**

* character=incharacter

1. **getencodingnumber**

* return encoding number

1. **setencodingnumber**

* encodingnumber=inencodingnumber

1. **equals**

* sees if encodingChar object equals object o
* returns equals

1. **compareto**

* compares encodingChar to object o if encodingChar is greater than object o compare equals -1
* compares encodingChar to object o if c encodingChar is less than object o compare equals 1
* compares encodingChar to object o if encodingChar is equal to object o compare equals 0
* returns compare

1. **binarytostring**

* format output into two columns, character column and encodingnumber column
* returns S

1. **Sort**

* converts a linked list into an array
* bubble sorts encodingChar objects by character
* converts array back into a linked list
* returns letters

1. **swap**

* interchanges two elements in an array for the bubble sort in the method above

**Files Needed:** finalproject.txt, frequencyresults.txt, compressed.txt , uncompressed.txt and binarytable.txt all of which are located in the final project bin folder

**finalproject.txt-**input file for project

**frequencyresults.txt-**stores the letters and their frequencies

**binarytable.txt-** stores letters and binary code along with character count

**compressed.txt-** stores compressed file

**Uncompressed.txt-**output file for project.

**Main program (Mainprogram) pseudocode**

{

**Step #1-declare/initialize variables and get appropriate utilities**

1. get appropriate Java utilities and Java I/O.
2. Declare link list called leathers of charFreq object
3. declare link list called encodedletters of encodingChar object
4. declare a charFrequency object called character and set it equal to\ 0, 0
5. declare a encodingchar called encoded character and set it equal to\0, ""
6. declare Boolean called available and set it equal to false
7. declare a Boolean variable called exist and set it equal to false
8. declared integer variable called index and set it equal to zero
9. declared integer variable called encoded index and set it equal to zero
10. declare a string variable called compress filename
11. declare string variable called uncompress filename
12. declare a new binary tree object called bt
13. declare a new node object called p
14. declare a new file operations object called fo
15. declare a new string builder called codes

**Step #2-create program**

beginning of main method

1. print a welcome message to the screen
2. print a message to let the user know they're about to compress a file
3. begin do while loop
4. prompt the user for the file they wish to compress specifying not to have any spaces in the file name
5. compress file name = input
6. closed do
7. while input not equal finalproject.txt
8. add double backslash to input variable
9. declare a new file object called f and set it equal to compress filename
10. if file exists then print file is being compressed to the screen
11. execute the steps to compress the file
12. when compression is finished that the user know and the location of the compressed file.
13. else print a message telling the user that the file or path does not exist and to restart the program
14. print a message to let the user know they're about to decompress a file
15. begin do while loop
16. prompt the user for the file they wish to uncompress specifying not to have any spaces in the file name
17. uncompress file name = input
18. closed do
19. while input not equal compressed.txt
20. add double backslash to input variable
21. declare a new file object called f2 and set it equal to uncompress filename
22. if file exists then print file is being uncompressed to the screen
23. execute the steps to uncompress the file
24. when decompression is finished that the user know and the location of the uncompressed file.
25. else print a message telling the user that the file or path does not exist and to restart the program

ending of main method

**compress file**

**Step #3-read give frequency**

beginning of read file method

1. set up try catch
2. Declare a string called path for the system.get property to get the user directory.
3. Declared buffer reader called in and have it set to path plus location of the bin folder of finalproject where final project. TXT is located.
4. Declare integer variable called value and set it equal to in.read
5. set up a while loop and set value not equal to -1
6. set character to new charFrequency to char value and 1
7. set available equal to letters contains character.
8. if available = true then.
9. Set index to index of character
10. get the index of the character in letters
11. increment the count
12. if available = false then
13. add the character to the letters
14. close if then statement
15. set value equal to in.read
16. continue loop until value equal -1
17. type in.close to close file
18. close try
19. catch any exceptions which are relevant to file process

ending of read file method

**Step #4 -sort frequencies in ascending order**

beginning of sort method

1. declare an array of charFreq object and the size of frequency array to letters.size
2. convert letters to array frequency
3. declare Boolean variable done set it equal to true
4. set up a nested for loop with integer variables I and J
5. for the for loop concerning I set I equal to the length of the array, continue while I is greater than zero and decrement I
6. inside the I for loop set done equal to true
7. for the for loop concerning J set J equal to one, continue while J is greater than I and increment J
8. inside the J for loop if the index of J get count is less than the index of J -1 get count then

beginning of swap method

1. declare a charFreq object called temp
2. temp =frequency index one
3. frequency index one = frequency index two
4. frequency index two = temp

ending of swap method

1. set done equal to false
2. if done then
3. declare a list of charFreq object called storage and convert array frequency to the list
4. set letters to equal a new link list of charFreq storage.
5. Close if statement

ending of sort method

**Step #5-results sent to file.**

Beginning of write file method

1. set up try
2. declare a string called path to the users directory
3. declare a new print writer variable called out and have it set to path plus location of the bin folder of final project where frequencyresultstxt is located.
4. Use out.printf to format heading of file to character , frequency
5. set up for loop
6. declare a integer variable I and set it equal to zero, have loop continuing until I is greater than letters and increment I.
7. Within for loop use out.print line to get the character and count at I from letters
8. format using frequencytostring method and close for loop
9. type out.flush and out.close
10. have the exact location of frequencyresults .txt appear on the Java Coundole
11. close try
12. catch any exceptions which are relevant to file process

ending of write file method

**Step #6 -create binary tree**

beginning of build tree method

1. create a new link list of nodes containing charFrequency objects called leafs
2. declare nodes first, second, combine and n
3. set up for loop
4. for each element in letters node n = new node of charFreq object
5. add new nodes to leafs
6. close for loop
7. while leafs greater than one
8. note first = first element in leafs
9. remove first element from leafs
10. node second = first element in leafs
11. remove first element from leafs
12. combine = new node of new charFreq object null and count of first + count of second
13. set left of combine equal to first
14. set right of combine equal to second
15. add combine to leafs
16. close a while loop
17. declaring a binary tree object called encoding tree and put first element in leafs in as the root.
18. Return encoding tree

ending of build tree method

**Step #6 -create encoding table**

beginning of create table method

1. if p not equal to null, then
2. add "0" to code string
3. call create table method on get left for p, codes and encodingtree,
4. add "0" to code string
5. add "," to code string
6. use the delete char at method to subtract one from codes string.
7. add "1" to code string.
8. call create table method on get right for p codes, encodingtree
9. use the delete char at method to subtract one from code string.
10. close if statement
11. return codes

ending of create table method

**Step #7 - write encoding table to file**

Beginning of write encoding table method

1. declare a string called data and send it equal to codes to string
2. declare a string array called info and set it equal to data.split at the ","
3. set binary equal to new encodingchar array of info.length
4. set up try
5. declare a new print writer variable called out and have it set to path plus location of the bin folder of final project where binarytable.txt is located.
6. Use out.printf to format heading of file to character , code
7. set up for loop
8. declare a integer variable I and set it equal to zero, have loop continuing until I is greater than binary and increment I.
9. format using binarytostring method and close for loop
10. type out.flush and out.close
11. have the exact location of binarytables.txt appear on the Java Coundole
12. close try
13. catch any exceptions which are relevant to file process

ending of write encoding table method

**Step #8 -compressed file**

beginning of compress method

1. declare a string called encoding
2. declaring link list called bytes
3. declare a byte variable called b and set it equal to zero
4. declare an integer variable called power is that it equal to zero
5. set up try catch
6. Declare a string called path for the system.get property to get the user directory.
7. Declared buffer reader called in and have it set to path plus location of the bin folder of finalproject where final project. TXT is located.
8. Declare integer variable called value and set it equal to in.read
9. set up a while loop and set value not equal to -1
10. declare a char variable called the original and set it equal to the character equivalent of value
11. set exist equal to encoded letters contains original
12. if exist = true then
13. Set encoded index to index of original
14. set encoded character equal to the element at the encoded index encoded letters
15. set encoding equal to the element at the encoded index's encoding number
16. set up for loop
17. initialize integer I to zero, have I go the length of the encoding string, increment I
18. while I is less than or equal to seven and greater than or equal to zero
19. if the element I equals one then
20. power = math.pow 2, I convert answer to an integer
21. using power as the index set the bit
22. if I is greater than seven then
23. add the byte to bytes link list
24. subject eight characters from the string
25. if the length of the current string is zero then
26. reset the bite equal zero
27. read next encoding number from file
28. else if the length of the current string is greater than zero add it to the next string from file.
29. Reset the bite to equal zero
30. read next encoding number from file
31. after all the bytes have input into the link list
32. declare a new print writer variable called out and have it set to path plus location of the bin folder of final project where compressed.txt is located.
33. set up for loop
34. declare a integer variable I and set it equal to zero, have loop continuing until I is greater than bytes and increment I.
35. Within for loop use out.print line to get the byte at I
36. type out.flush and out.close
37. have the exact location of compressed.txt appear on the Java Coundole
38. close try
39. catch any exceptions which are relevant to file process

ending of compress method

Uncompress

**Step #1 -read the encoding table**

beginning of read table method

1. set up try catch
2. Declare a string called path for the system.get property to get the user directory.
3. Declared buffer reader called in and have it set to path plus location of the bin folder of finalproject where binary table.txt is located.
4. Declare a char called character and set it equal to null
5. declare a string called number and set it equal to ""
6. declare string called I intended equal to null
7. Declare integer variable called value and set it equal to in.read
8. set up a while loop and set value not equal to -1
9. if value is a letter
10. character equals value casted as char
11. else
12. I = value as string
13. number = number + I
14. encoded character = new encoding char object
15. add the encoded character to the encodedletters
16. close if then statement
17. set value equal to in.read
18. continue loop until value equal -1
19. type in.close to close file
20. close try
21. catch any exceptions which are relevant to file process

ending of read table method

**step #2-create binary tree**

beginning of build decoding tree method

1. declare a new binary tree object
2. declare a string called line
3. declare a generic node object called n
4. declaring new node called top
5. have top equal root
6. set up for loop start integer I at zero, go in till I is greater than encoding letters link list, increment I
7. for each item in the list get the elements encoded number string
8. set line equal to the element encoding number string
9. set up a for loop start integer J at zero, go until J is greater than the length of line, increment J
10. read each character of the string
11. if character is zero then
12. if current.left is equal to null then
13. create new generic node
14. set current left to n
15. current equals current left
16. else if current.right is equal to null then
17. create new generic node
18. set current right to n
19. current equals current right
20. if right and left are null then
21. print the character at the node

ending of build decoding tree method

**Step #3 -uncompress the file**

beginning of decompression method

1. set up try catch
2. Declare a string called path for the system.get property to get the user directory.
3. Declare a link list called finished process
4. if P not equal to null then
5. decompression method on get left on p, and decodingtree
6. add the element at P to finished process link list
7. call create table method on get right for p and decodingtree
8. close if statement
9. declare a new print writer variable called out and have it set to path plus location of the bin folder of final project where uncompressed.txt is located.
10. set up for loop
11. declare a integer variable I and set it equal to zero, have loop continuing until I is greater than finishedprocess and increment I.
12. Within for loop use out.print line to get the element at I
13. type out.flush and out.close
14. have the exact location of uncompressed.txt is appear on the Java Coundole
15. close try
16. catch any exceptions which are relevant to file process

ending of decompress method