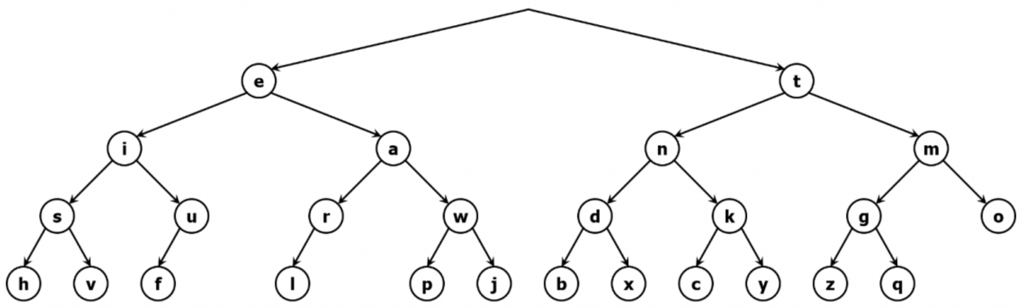
CPT-287

Team 2

Morse Code Binary Tree



Steven Shackleford

Chris Anderson

Luke Hunt

Table of Contents

Table of Contents 3

System Design 4

Binary Tree Methods 5

UML Diagram 6

Test Case 1 7

Test Case 2 8

Team member contribution 9

Future Improvements 10

System Design

The Binary tree is built with the given input file, and then the user is prompted to enter in a message they would like to encode/decode. The system will automatically parse the user input and print out the corresponding value for what the user entered.

Building the Binary Tree

To build the binary tree with the given input file, we first initialize the root node to be {null} and build off the root null. If the current character in the string is a {.}, it will advance the root to the left, if the character is {-}, it will instead advance the root to the right. During this process it's checking if the next node in the list exists. If not, it will create a node and initialize it with the value ‘ ‘. If the last character of the string is reached, it will take the current node it's on and set its' value to the first character of the string, which is the alphabetic character the morse code represents.

Menu

To determine which method should be called, the user is prompted for an input. If the input begins with a {.} or {-}, the decode method will be called using the input as the argument. Otherwise, the encode method will be called to convert the text into Morse code using {.} and {-} characters.

Encoding a message

To encode a message, the user input will be passed into the {encode} method. The method will then loop through all characters in the input string, ignoring spaces. For each character, the {encodeSearch} method will be recursively called to determine what the corresponding morse code string should be. This is done by determining the path to the matching node containing this character from the root and representing each left branch of the path as a {.} and right branch of the path as a {-}.

Decoding a message

In order to decode a message, the user will use the console to enter each morse code value separated by a space. Each value will be passed to the {decode} method. The {decode} method will iterate through each character of the morse code value and traverse the tree accordingly. If the character is a ‘.’ it will traverse left. If the character is a ‘-’ it will traverse right. When spaces are detected, the method will append the current value, in this program's case, a letter, to our “result”. After iterating through the entire input, the result will output to the console with no spaces.

Binary Tree Methods

Builder:

The {builder} method takes in a String token as the character and morse code to parse, and a TreeNode root, which is the {null} root of the Morse Code binary tree. The method then stores the alphabetic character, gets the token length, and then starts to parse the rest of the token input, which is the morse code that corresponds to the alphabetic character. The method will check if the current character is either a {.} or a {-}, and will go left or right accordingly. If the current left/right node is null, it will create a new node and will temporarily initialize it with {“ “}. It will then move to that node and continue to parse the input token’s next character. Once it hits the last character in the string, it will set the current node to the value of the alphabetic character it stored from the beginning of the string. It will repeat this process for every value in the input table.

Encode:

The {encode} method takes in the user input as a String {input} to parse and TreeNode {root} that is the root of the morse code tree created in the {builder} method. An empty String {output} is created to store the combined set of {.} and {-} characters representing the Morse code translation of the {input}.
The {encode} method then loops through all characters in the input string, ignoring any spaces. For each pass, the current character is stored as a char {letter} and a Stack {path} is created to store the characters representing the path. Then the {encodeSearch} method is called to store the Morse code translation of the {letter} within the Stack {path}. Then, the characters in the Stack {path} are popped off and appended to the String {output}. The end of the loop occurs after a space is added to the end of the String {output}.
The method ends after the String {output} is returned.

EncodeSearch:

The {encodeSearch} method is a private method called by the {encode} method that stores the path to a target character in the Morse code tree into a Stack. {True} is returned if the target is located, {false} otherwise. This method takes in parameters char {letter} representing the target of the search, TreeNode {root} representing the root (or current subtree root when called recursively) of the Morse code tree, and Stack of characters {path}.

The base cases are as follows:

1. The node {root} is null, then return {false}
2. The node {root} has a value matching that of {letter}, return true

The recursive relations are as follows:

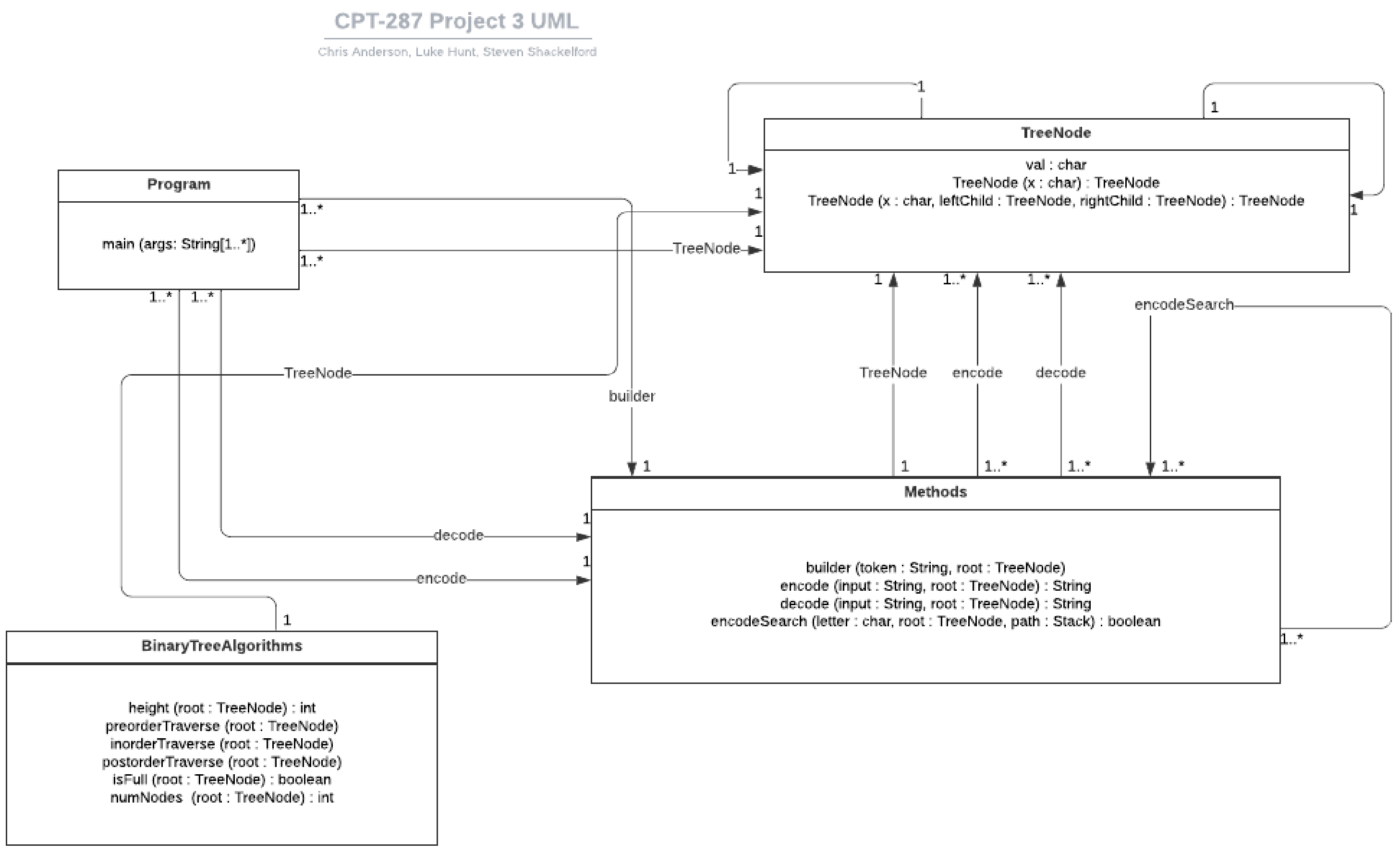
1. If the left child of the current node == {true}, add {.} to the Stack {path}
2. Else, if the right child of the current node == {true}, add {-} to the Stack {path}
3. Else, return {false}

The result of this method is that as soon as a match is found and returns true, the Stack {path} will be loaded with all {.} and {-} characters as the recursive calls are returned true back up to the original {root} that was passed in. When popped off in the {encode} method, the result is a string representing the current {letter} in Morse code, or an empty Stack if no match is found.

Decode:

The {decode} method takes in a String as the {input} and TreeNode {root} as the root of the morse code tree. String {result} is created to hold our decoded message and TreeNode {current} is a temporary variable that holds the TreeNode {root} location initially but traverses the tree based on the morse code characters. If the character is a '.' then we traverse the tree left by assigning {current} to {current.left}. Likewise, if the character is a '-' then we traverse the tree right by assigning {current} to {current.right}. However, if we come across a space, then the method will change {current.val} to {result}, increment the counter by one, and reset {current} to {root}. After the length of the {input} is reached, we append the {current.val} to the {result} and return the {result}.

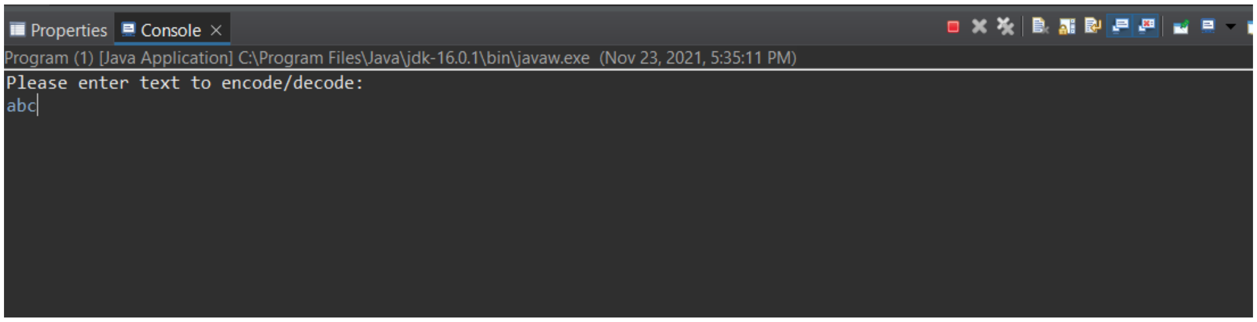
UML Diagram


CPT-287 Project 3 UML
Chris Anderson, Luke Hunt, Steven Shackelford
TreeNode
val : char
TreeNode (x : char) : TreeNode
TreeNode (x : char, leftChild : TreeNode, rightChild : TreeNode) : TreeNode
Program
main (args: String[1..*])
Methods
builder (token : String, root : TreeNode)
encode (input : String, root : TreeNode) : String
decode (input : String, root : TreeNode) : String
encodeSearch (letter : char, root : TreeNode, path : Stack) : boolean
BinaryTreeAlgorithms
height (root : TreeNode) : int
preorderTraverse (root : TreeNode)
inorderTraverse (root : TreeNode)
postorderTraverse (root : TreeNode)
isFull (root : TreeNode) : boolean
numNodes (root : TreeNode) : int
TreeNode
TreeNode
builder
decode
encode
TreeNode
encode
decode
encodeSearch
1
1..*

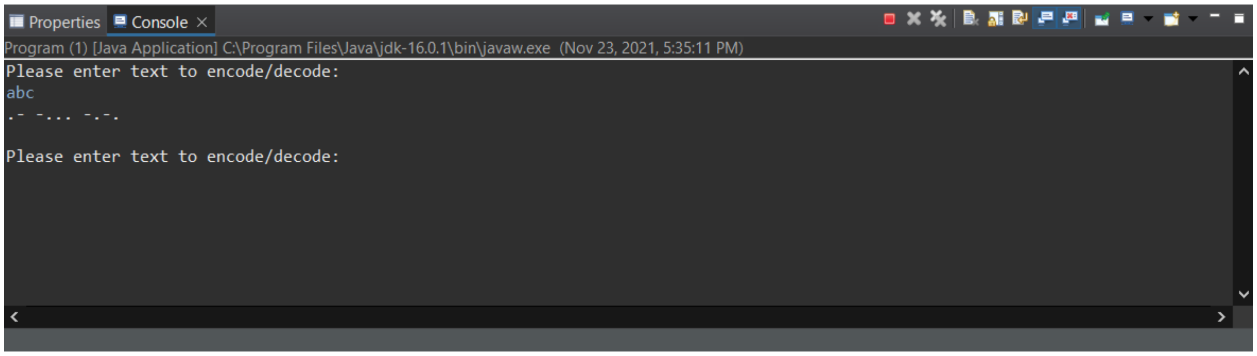
Test Case 1:

Encode Test

For the first test case we will have a user input the letters “abc” or “a b c” in the console. The spaces between letters do not matter. When they hit enter the program should output the morse code value for each letter with a space in between. For this example the console should output: **.- -... -.-.**



```
Program (1) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Nov 23, 2021, 5:35:11 PM)
Please enter text to encode/decode:
abc
```



```
Program (1) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Nov 23, 2021, 5:35:11 PM)
Please enter text to encode/decode:
abc
.- -... -.-.

Please enter text to encode/decode:
```

As we can see after the program executes it returns: **.- -... -.-.**

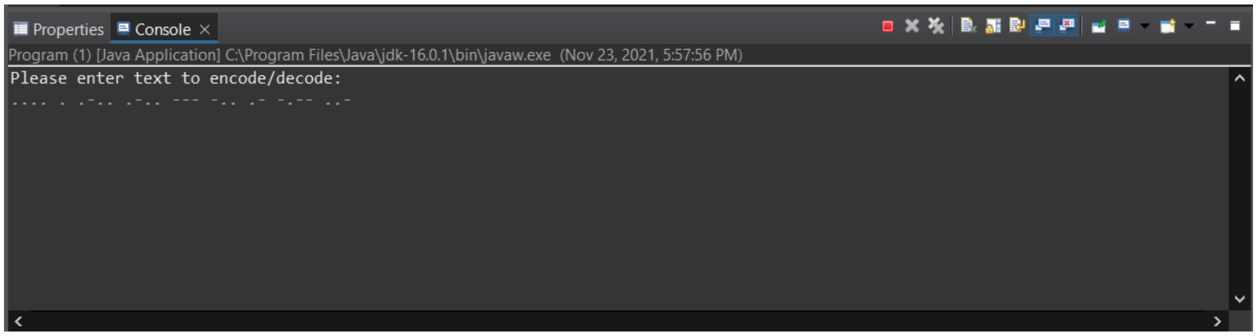
Test Case 2:

Decode Test

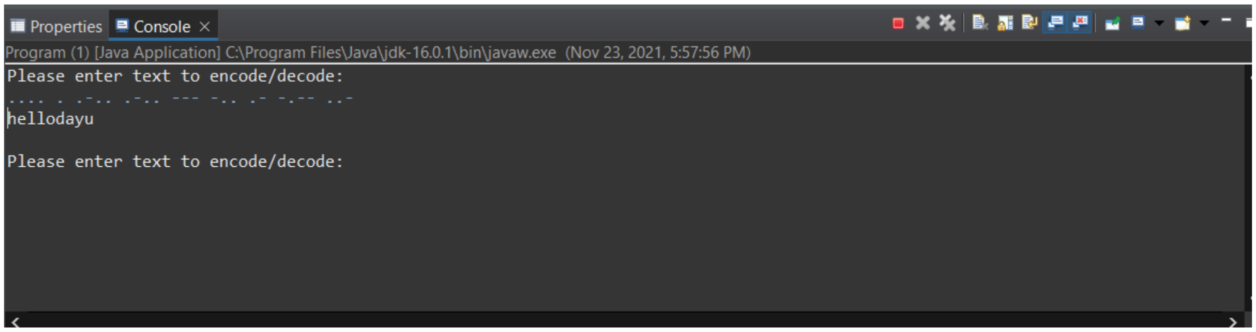
The second test case will have a user decode a message by entering in morse code as the input in the console. The user must separate each morse code value by a space. However, the output will return a string that has no spaces.

Our morse code values will be as follows:
…. . .-.. .-.. --- -.. .- -.-- ..-

This example should output “hellodayu” to the console.



```
Program (1) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Nov 23, 2021, 5:57:56 PM)
Please enter text to encode/decode:
.... . .-.. .-.. --- -.. .- -.-- ..-
```



```
Program (1) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (Nov 23, 2021, 5:57:56 PM)
Please enter text to encode/decode:
.... . .-.. .-.. --- -.. .- -.-- ..-
hellodayu

Please enter text to encode/decode:
```

After the program executes we can see that the expected output of “hellodayu” was correctly returned to the console.

Team member contribution

Steven: morse code file import, {builder} method to build binary tree, created system design doc, worked on system design doc

Luke: user input loop, {encode} and {encodeSearch} method, worked on system design doc

Chris: {decode} method, test cases, UML diagram, worked on system design doc

Future Improvements

An improvement that would add to the user experience, would be to add in the ability to represent spaces in Morse code using a different delimiter (such as {/} or {|}). This would allow the user to more easily read and translate multiple words into a format rather than an extremely long string of letters.

Another improvement would be to inform the user if an unexpected input character for either decoding or encoding is passed in so that they may correct their inputs. Currently, the {encode} method ignores any character that does not exist in the tree and the {decode} method assumes any characters that are not {.}, must be a {-}.