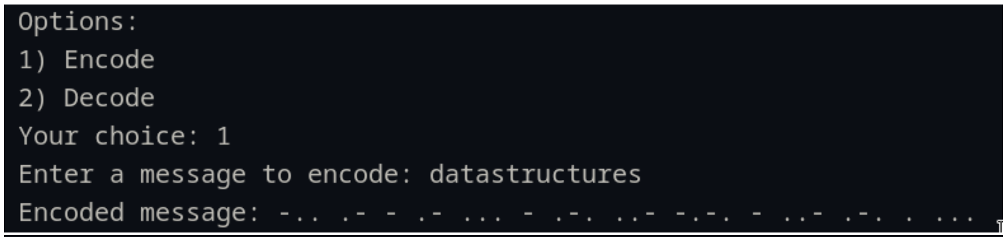
Assignment 6: Heaps and Trees
Part 2: Trees
CS3305/W01 Data Structures

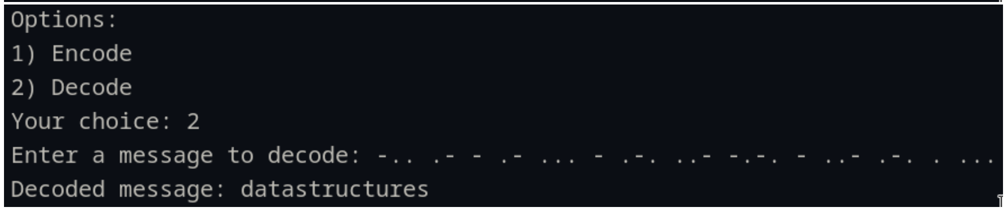
Casey Hampson

September 26, 2024

Program Output



```
Options:
1) Encode
2) Decode
Your choice: 1
Enter a message to encode: datastructures
Encoded message: -.. .- - .- ... - .-. ..- -.-. - ..- .-. . ...
```



```
Options:
1) Encode
2) Decode
Your choice: 2
Enter a message to decode: -.. .- - .- ... - .-. ..- -.-. - ..- .-. . ...
Decoded message: datastructures
```

Source Code

```java
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 6-Part-2-Trees


import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;




class BinaryTreeNode<E> {
    public E elem = null;
    public BinaryTreeNode<E> left = null, right = null;

    public BinaryTreeNode() {}
    public BinaryTreeNode(E e) { this.elem = e; }
}




class MorseCodeTree {
    BinaryTreeNode<Character> bst = null;


    // I spent quite a while trying to figure this out.
    // There is no doubt a better way to do it with nifty Java stuff like
    // iterators and the like, but I simply do not know those things fully yet.
    // Since I enjoy C/C++ pointers, my mind was led there first,
    // and while this IntPointer is kinda dumb and I hate how annoying Java is with
    // references, it works.
    // Essentially, this is just the pre-order search, but with a little extra pizzazz
    // to account for the fact that we aren't just searching, we are placing values.
    private static class IntPointer {
        public int idx = 0;
    }
    private BinaryTreeNode<Character> __fill_bst(
        List<Character> chars,
        IntPointer idx,
        BinaryTreeNode<Character> node
    ) {
        // if we encounter a question mark (which is what I made the NULL values)
        // then we are at the end of this line, so we return a null node
        // and increment the pointer to skip past the question mark
        // and on to the next letter
        // Or, if we run out of letters, then we are done as well and just return null
        // repeatedly
        if (idx.idx >= chars.size() || chars.get(idx.idx) == '?') {
            idx.idx++;
            return null;
        }

        // make the current root be the current character.
        node = new BinaryTreeNode<>(chars.get(idx.idx));
        idx.idx++; // move the index to the next character in the list

        // recursively do this for the left then the right subtrees
        node.left = __fill_bst(chars, idx, node.left);
        node.right = __fill_bst(chars, idx, node.right);

        return node;
    }


    public MorseCodeTree(String morse_file_path) {
        // grab a scanner to the morse code file
        Scanner file_scanner;
        try {
            File morse_file = new File(morse_file_path);
            file_scanner = new Scanner(morse_file);
        } catch (FileNotFoundException e) {
            System.out.printf("File \"%s\" not found!\n", morse_file_path);
            return;
        }

        // stick everything in a list
        List<Character> list = new ArrayList<>();
        while (file_scanner.hasNextLine()) {
            String line = file_scanner.nextLine();
            if (line.compareToIgnoreCase("NULL") == 0) {
                list.add('?');
                continue;
            }
            list.add(line.charAt(0));
        }

        // now place it all in the tree
        this.bst = __fill_bst(list, new IntPointer(), this.bst);


        file_scanner.close();
    }



    // returns the path taken to get to c.
    // this is just the preorder search, but with an extra parameter being the path
    // and whenever we go into the left tree, we add a '.',
    // and whenever we go into the right tree, we add a '-'
    public String Search(BinaryTreeNode<Character> node, char c, String path) {
        // base case; if we reach null, then stop, because it's not this way
        if (node == null) return null;
        // other base case: we find the letter! just return the path:
        if (node.elem == c) return path;

        // otherwise, grab the path from going along the left first
        // if it is not null, then we found it somewhere, so we keep propagating it up
        String left_path = Search(node.left, c, path + '.');
        if (left_path != null) return left_path;
        // similarlyl for right, but with a dash instead
        String right_path = Search(node.right, c, path + '-');
        if (right_path != null) return right_path;

        return null; // unreachable (we assume the user enters normal alphabetic characters)
    }


    // this just splits the string into individual characters
    // and calls the above search function on each to get the path
    public String Encode(String message) {
        char[] char_arr = message.toLowerCase().toCharArray();
        String encoded_message = "";
        for (char c: char_arr) {
            encoded_message += Search(this.bst, c, "") + " ";
        }
        return encoded_message;
    }


    public String Decode(String message) {
        String[] message_tokens = message.split(" ");
        String decoded_message = "";
        for (String message_token : message_tokens) {
            BinaryTreeNode<Character> current = this.bst;
            char[] path_tokens = message_token.toCharArray();
            for (char path_token : path_tokens) {
                // see if it's a space
                if (path_token == ' ') continue;
                // check that the path isn't going to take us somewhere null
                if (
                    (current.left == null && path_token == '.') ||
                    (current.right == null && path_token == '-')
                ) {
                    decoded_message += '?';
                    continue;
                }
                if (path_token == '.') current = current.left;
                if (path_token == '-') current = current.right;
            }
            decoded_message += current.elem;
        }

        return decoded_message;
    }
}
```

```java
public class P2 {
    private static Scanner sc;
    private static enum MenuChoice {
        Encode, Decode, Error
    };

    public static MenuChoice Menu() {
        System.out.printf("Options:\n");
        System.out.printf("1) Encode\n");
        System.out.printf("2) Decode\n");
        System.out.printf("Your choice: ");

        int choice = Integer.parseInt(sc.nextLine());
        while (choice<1 || choice>2) {
            System.out.printf("Please enter a valid option!\nYour choice: ");
            choice = Integer.parseInt(sc.nextLine());
        }

        switch(choice) {
            case 1 -> { return MenuChoice.Encode; }
            case 2 -> { return MenuChoice.Decode; }
        }
        return MenuChoice.Error; //unreachable
    }

    public static void main(String[] args) {
        sc = new Scanner(System.in);

        MorseCodeTree morsecode_tree = new MorseCodeTree("./P2/res/morse.txt");

        MenuChoice choice = Menu();
        switch(choice) {
            case Encode -> {
                System.out.printf("Enter a message to encode: ");
                String encoded_message = morsecode_tree.Encode(sc.nextLine());
                System.out.printf("Encoded message: %s\n", encoded_message);
            }
            case Decode -> {
                System.out.printf("Enter a message to decode: ");
                String decoded_message = morsecode_tree.Decode(sc.nextLine());
                System.out.printf("Decoded message: %s\n", decoded_message);
            }
            default -> {} // unreachable
        }

        sc.close();
    }
}
```