Module: DATA STRUCTURES & ALGORITHMS(AI) [ST1507]

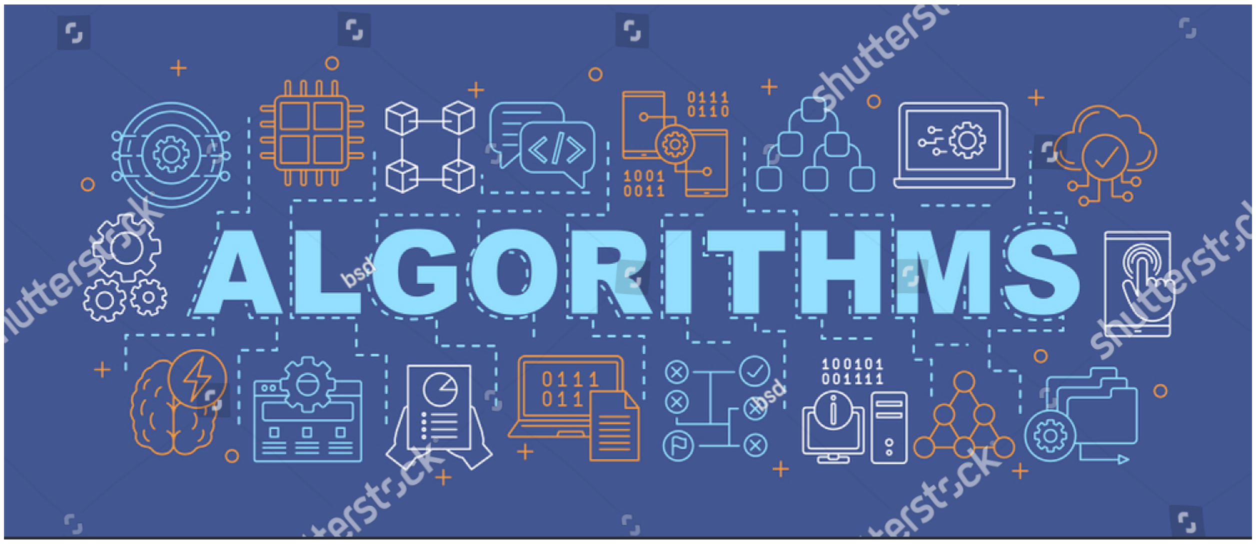
CA1 Morse Code Analyzer Report

Lecturer: Ms Hwee Shan Tay
Name: Ang Kah Shin (P2004176)

Class: DAAA/FT/2B/04

Table of Contents

Introduction



This project is a practice on the various Data Structures, Algorithms and the concept of Object-Oriented Programming by developing a Morse Code Analyzer. The program will allow the user to choose between different functions such as encoding a Message into Morse code, and analyzing a specified morse file for its contents. The application is able to analyze multiline morse code messages from an external file source and analyze and extract essential messages from the decoded string. This is done so by removing the stop words from the message and making use of the frequency of the word occurring and the location of the words to sort them by importance to form the essential message.

User Guideline

Switch to the project directory and start the program

```
cd <project_directory>
python main.py
```

Main Program

```
*******************************************
* ST1507 DSAA: MorseCode Message Analyzer *
*-----------------------------------------*
*                                         *
* - Done by: Ang Kah Shin (P2004176)      *
* - Class: DAAA/FT/2B/04                  *
*******************************************



Please select your choice (1, 2, 3, 4)
1. Change Printing Mode
2. Convert Text to Morse Code
3. Analyze Morse Code Message
4. Exit
Enter choice:
```

You will be given 4 different options to either change the printing output format, encoding regular text into morse code as well as conducting an analysis on an input text file that contains a paragraph of morse code to be decoded.

Option 01: Change Printing Mode

```
Current printing mode is: H


Enter 'h' for horizontal or 'v' for vertical, then press enter
Enter choice:
```

The program allows the user to switch between horizontal or vertical printing of the encoded message for option 2 (Converting text to morse code). The program will parse the user input to ensure that the user only specifies either a ‘h’ or ‘v’ and will reject any other forms of inputs. The printing mode will then be set for the output operation until the user exits the program or when the user performs a change in the printing mode again by selecting a different choice. The program defaults to printing the morse code output horizontally if the user does not change the printing mode.

Option 02: Converting Text to Morse Code

```
Enter Message you wish to encode: hello there how are you
....,.,.-..,.-..,--- -,....,.,.-.,. ....,---,.-- .-,.-.,. -.--,---,..-
Press Enter to continue...
```

In the above code block, the user inputs in a sentence that he wishes to encode. e.g. “hello there how are you”. Since the user did not change the printing format of the program, the program defaults to horizontal printing and you can observe from the above shell output that the morse code prints out horizontally with each encoded character separated by a comma(,) and each encoded word separated with a space( ) in between.

```
Enter Message you wish to encode: hello there how are you
hello there how are you
. ..   .    .       -
. ---  . .  .-.  .  .-.
. ..-  . -  .-- .-  --.
....- -.... .-- -.. ---

Press Enter to continue...
```

By changing the printing mode to be ‘Vertical’, the program will then print out the same encoded message this time with each encoded character printed vertically without being separated by comma(,) while each word is still separated by a space.

Option 03: Analyze Morse Code Message

```
Enter the input file name: input.txt
Enter the output file name: output.txt
```

The Morse Code analysis option allows the user to specify an input text file containing the encoded message in morse as well as specify the output file name to write the analysis report to. Afterwards, the program will perform 3 operations for the analysis: Decode the message, sort and print out the decoded words along with the positions of the word in the message by the frequency of occurrence of the word, length of the decoded word followed by alphabetical order. After printing out the analysis, the program will also generate a report and save the output into the output file specified at the beginning. If no output file is specified, the program will automatically generate and store the output into a text file called ‘report.txt’

Data Structures & Algorithms Used

Linear Data Structure (SortedList & Nodes)
I implemented a SortedList class that sorts and stores nodes based on a custom condition that I specified to facilitate a more efficient form of sorting for the morse codes and words. The implementation of the SortedList class allowed me to implement an efficient sorting algorithm that only has a complexity of O(n). As such, the sorting of the input nodes inside the array will be much faster and efficient.

Morse Class
I implemented the Morse class to use class-specific functions such as attribute protection to restrict the usage and visibility of the morse lookup table. I also find it sensible to implement the Morse class that stores all the relevant functions relating to Morse code encoding/decoding and analysis. The complexity of each of the different functions in the class varies with the most complex operation having a complexity of $O(n^2)$.

EncodedWord Class
This class was implemented as a custom Data Structure that allows me to store attributes such as the text, encoded morse code of the text, an array of morse encoded letters of the word, as well as the length of each morse encoding. The class also has its own functions to add padding to each of the character encoding based on the largest morse encoding of the word. This function allows the EncodedWord object to easily be used for both horizontal and vertical printing which is one of the requirements of the project.

Text Class
The Text class is a custom Data Structure implemented that utilizes the concepts of PolyMorphism as well as inheritance of attributes from its Parent Class (Node). The Text class contains critical attributes such as the x and y positions of the word in the message as well as the length of the morse encoding of the text. The class also contains the logic to aid the SortedList class to properly sort the text nodes by the (x, y) positions of the text in the message.

Node Class
The Node class is an abstract class that only contains the information for the next node after the current. This class is used as a display of implementing a simple abstract class for attribute inheritance between the Parent and Child classes.

Sorted function
Throughout the functions I built for decoding, sorting and analyzing the morse code and texts, I often tap on the built-in `sorted()` function to help me sort a given array by a list of key conditions. The maximum complexity of the sorted function used was $O(n^2 logn)$. Given the limitations of the SortedList class, the Python built-in sorted function allows me to easily specify the different conditions that I would like to sort my data through.

Object-Oriented Programming

The project is implemented using Object-Oriented Programming(OOP) so as to make use of concepts:

1) Encapsulation
2) PolyMorphism
3) Inheritance

These concepts allow me to protect and restrict certain Class attributes and functions.

Encapsulation
Using the concepts of Encapsulation, I was able to restrict the Morse Lookup table to only be accessible within the Morse Class. This allows for the program to prevent the user or any of the other functions in the program to alter the values in the lookup table.

Apart from protecting sensitive values in the Morse Class, I also applied the concepts of Encapsulation to restrict certain class functions to only be accessed by itself. I encapsulated the *checkUserInput* function found in the Menu Class to prevent access to the function as the function's primary role is to validate the user's input against the menu options without having to validate the input within the main program.

Inheritance
Attribute inheritance was displayed by the Text class as it inherits the nextNode property from its parent class (Node). This allows for a more structured way of coding since multiple custom Node classes can be made and it will all be able to inherit the same nextNode attribute from the Node parent class, the implementation in this project is very limited as there are very few classes that requires the use of the nextNode attribute found in the Node class.

Appendix

Main.py

```
'''
Name: Ang Kah Shin
Class: DAAA/FT/2B/04
Admin: P2004176
'''
# Writing program

from Morse import Morse
from Menu import Menu


def mainBanner():
   banner = '''
*******************************************
* ST1507 DSAA: MorseCode Message Analyzer *
*-----------------------------------------*
*                                         *
* - Done by: Ang Kah Shin (P2004176)      *
* - Class: DAAA/FT/2B/04                  *
*******************************************
   '''
   return banner



def main():
   '''The main Program loop'''
   # program variables:
   pMode = 'H'
   while 1:
       morse = Morse()

       print(mainBanner())
       # Main menu selection
       mainMenu = Menu("Please select your choice (1, 2, 3, 4)", True )
       # add options into mainMenu
       mainMenu.insert([
           "Change Printing Mode",
           "Convert Text to Morse Code",
           "Analyze Morse Code Message",
           "Exit"
       ])
       # print mainMenu
```

```
        userInput = mainMenu.show()

        if userInput == '1':
            print(f"Current printing mode is: {pMode}")
            changeMode = Menu("\nEnter 'h' for horizontal or 'v' for vertical, then 
press enter", True, ['H', 'V'] )
            userInput = changeMode.show()
            print(f"The print mode has been changed to {'Vertical' if userInput == 
'V' else 'Horizontal'}")
            pMode = userInput
        elif userInput == '2':
            inputMorse = input("Enter Message you wish to encode: ")
            print(morse.encode(inputMorse, mode=pMode))
            # print("Encoded message:", msg)
        elif userInput == '3':
            inputFile = input("Enter the input file name: ")
            outputFile = input("Enter the output file name: ")
            print(morse.analyze(inputFile, outputFile))

        elif userInput == '4':
            print("Exiting program...")
            exit()
        else:
            print("Invalid input. Please try again.")
            main()

        userInput = input("\nPress Enter to continue...")



if __name__ == "__main__":

    main()
```

Morse.py

```
from Text import Text
from EncodedWord import EncodedWord
from itertools import zip_longest
import numpy as np
from SortedList import SortedList
'''
Name: Ang Kah Shin
Class: DAAA/FT/2B/04
Admin: P2004176
'''
```

```
# Morse Code Class
class Morse:
    """
        Name: Morse
        Description:

    """
    table = {
        "A": ".-", "B": "-...", "C": "-.-.", "D": "-..",
        "E": ".", "F": "..-.", "G": "--.", "H": "....",
        "I": "..", "J": ".---", "K": "-.-", "L": ".-..",
        "M": "--", "N": "-.", "O": "---", "P": ".--.",
        "Q": "--.-", "R": ".-.", "S": "...", "T": "-",
        "U": "..-", "V": "...-", "W": ".--", "X": "-..-",
        "Y": "-.--", "Z": "--..", "1": ".----", "2": "..---",
        "3": "...--", "4": "....-", "5": ".....", "6": "-....",
        "7": "--...", "8": "---..", "9": "----.", "0": "-----"
    }

    def __init__(self):
        self.__lookup = self.__class__.table

    def displayOutput(self, output, method='H'):
        output = output.split(" ")
        encoded = [word.split(',') for word in output]
        countEncoded = [(x.count('.') + x.count('-')) for i, x in
enumerate(encoded)]
        print(encoded)
        def maxNum(l):
            if len(l) == 1:
                return l[0]
            else:
                if l[1] > l[0]:
                    return maxNum(l[1:])
                else:
                    return maxNum( [l[0]] + l[2:] )

        print("Third:", maxNum(countEncoded))


    def encode(self, sentence, mode='H'):
        '''Takes in 1 argument(sentence) and return the encoded sentence in Morse'''

        morseCode = []
        # validate to ensure that sentence only consist of only alphabets and space
        words = sentence.split(" ")
        for word in words:
            encoded = ""
            for i, letter in enumerate(word):
                if not letter.isalpha() and not letter.isspace() and not
letter.isdigit():
                    return "Please only include alphabets and spaces in your
sentence."
                if letter.upper() in self.__lookup:
                    if i != len(word) - 1:
                        encoded += self.__lookup[letter.upper()] + ","
                    else:
                        encoded += self.__lookup[letter.upper()]
            morse = EncodedWord(word, encoded)
            morseCode.append(morse)


        if mode == 'V':
            print(sentence)
            arr = []
            for obj in morseCode:
                arr.extend(obj.char)
                arr.append(" ")

            maxLength = len(max(arr, key=len))
            arr = ','.join([ (" " * (maxLength - len(x))) + x for x in arr])

            for char in zip_longest(*arr.split(','), fillvalue=''):
                print(''.join(char))
            return ''

        return ' '.join(obj.morse for obj in morseCode)

    def decode(self, morse):
        decoded = ""
        # validate to ensure that sentence only consist of only alphabets and space

        letters = morse.split(",")
        for letter in letters:
            for alpha, morse in self.__lookup.items():
                if morse == letter:
                    decoded += alpha

        return decoded

    def analyze(self, input, output=’report.txt’):
        '''Takes in 2 arguements: Input file and Output. Conducts a morse analysis
```

```
on the input file and return the results into the output file'''

       report = ""
       try:
           with open(input, "r") as f:
               data = f.readlines()
               f.close()
       except FileNotFoundError:
           return "<Input file not found.>\n"

       try:
           with open("stopwords.txt", "r") as f:
               stopwords = [word.upper() for word in f.read().split("\n")]
               f.close()
       except FileNotFoundError:
           return "Stopwords file not found."

       print("\n>>>Analysis has started:")
       stats = {}
       decodedString = ""

       for row, sentence in enumerate(data):
           for column, word in enumerate(sentence.split(" ")):
               decoded = Text(self.decode(word), word, row, column)
               if decoded.text in stats:
                   stats[decoded.text].insert(decoded)
               else:
                   newList = SortedList()
                   newList.insert(decoded)
                   stats[decoded.text] = newList
               decodedString += decoded.text + " "
           decodedString += "\n"
       uniqueWords = [key for key in stats.keys()]

       # Add decoded string into report
       report += ("*** Decoded Message\n" + decodedString)

       sortedUniqueWords = sorted(uniqueWords, key=lambda x: (stats[x].length,
len(x), x))
       lengths = sorted({stats[key].length for key in sortedUniqueWords},
reverse=True)

       for length in lengths:
           report += f"\n*** Morse Code with frequency => {length}\n"
           for key in sortedUniqueWords:
               if stats[key].length == length:
                   report += f"{self.encode(key, 'H')}\n[{key}]:
({stats[key].length}) {stats[key]}\n"

       # Essential Message Printing
       # Load Stopwords file
       try:
           with open("stopwords.txt", "r") as f:
               stopwords = [word.upper() for word in f.read().split("\n")]
               f.close()
       except FileNotFoundError:
           return "Stopwords file not found."

       # remove stopwords from uniqueWords
       noStopwords = [(word, stats[word].outputArr()) for word in sortedUniqueWords
if word not in stopwords]

       # sort nostopwords
       noStopwords = sorted(noStopwords, key=lambda x: (x[1][0], len(x[1])))

       ESSENTIAL = ""
       for length in lengths:
           for word, arr in noStopwords:
               if len(arr) == length:
                   ESSENTIAL += f"{word} "
       report += f"\n*** Essential Message:\n{ESSENTIAL}\n"
       print(report)

       # write report to output file
       try:
           with open(output, "w") as f:
               f.write(report)
               f.close()
       except FileNotFoundError:
           return "Output file not found."
```

Menu.py

```
'''
Name: Ang Kah Shin
Class: DAAA/FT/2B/04
Admin: P2004176
'''


class Menu:
   '''Menu class takes in 3 parameters:
   1. question: The question to be asked to the user
```

```
    2. order: Boolean value to determine whether the menu should be ordered or not
    3. callback: The function to be called when the user enters a valid input

    Functions Available:
    1. insert(items): Inserts items into the menu
    2. __str__(): Prints the menu
    '''

    def __init__(self, question, order, options=None):
        self.qn = question
        self.ord = order
        self.__items = []
        self.length = 0
        self.options = options

    def insert(self, items):
        for item in items:
            self.length += 1
            self.__items.append(item)

    def __checkInput(self, userInput):
        # Check user input to see if it exists in the list
        # If it does, return the index of the item
        if self.options == None:
            if userInput.isnumeric():
                userInput = int(userInput)
                if userInput >= 1 and userInput <= self.length:
                    return str(userInput)
            print("** <Invalid input> **")
            return self.show()
        else:
            if userInput.upper() in self.options:
                return userInput.upper()
            print("** <Invalid input> **")
        return self.show()

    def __str__(self):
        print("\n" + self.qn)
        if self.ord:
            for i in range(self.length):
                print(str(i+1) + ". " + self.__items[i])
        else:
            for item in self.__items:
                print("-", item)

        return ""
```

```python
    def show(self):
        print("\n" + self.qn)
        if self.ord:
            for i in range(self.length):
                print(str(i+1) + ". " + self.__items[i])
        else:
            for item in self.__items:
                print("-", item)

        userInput = self.__checkInput(input("Enter choice: "))
        return userInput


def testFunction(userInput):
    print("You entered: ", userInput + 1)
    return ""
```

Node.py

```python
class Node:
    def __init__(self):
        self.nextNode = None
```

SortedList.py

```python
'''
Name: Ang Kah Shin
Class: DAAA/FT/2B/04
Admin: P2004176
'''
# SortedList class
class SortedList:
    def __init__(self):
        # Pointer towards the first Node (currently nothing)
        self.headNode = None
        self.currentNode = None
        self.length = 0  # Variable we manually keep track of number of Nodes in the
list

    def __appendToHead(self, newNode):
        oldHeadNode = self.headNode
        self.headNode = newNode
        self.headNode.nextNode = oldHeadNode
        self.length += 1

    def insert(self, newNode):
        self.length += 1
        # If list is currently empty
        if self.headNode == None:
            self.headNode = newNode
            return

        # Check if it is going to be new head
        if newNode < self.headNode:
            self.__appendToHead(newNode)
            return

        # Check it is going to be inserted
        # between any pair of Nodes (left, right)
        leftNode = self.headNode
        rightNode = self.headNode.nextNode
        while rightNode != None:
            if newNode < rightNode:
                leftNode.nextNode = newNode
                newNode.nextNode = rightNode
                return
            leftNode = rightNode
            rightNode = rightNode.nextNode
        # Once we reach here it must be added at the tail
        # Beacuse newNode is largest than all the other nodes.
        leftNode.nextNode = newNode

    def outputArr(self):
        output = []
        node = self.headNode
        firstNode = True
        while node != None:
            if firstNode:
                output.append(eval(node.__str__()))
                firstNode = False
            else:
                output.append(eval(node.__str__()))
            node = node.nextNode
        return output


    def __str__(self):
        # We start at the head
        output = ""
        node = self.headNode
        firstNode = True
        while node != None:
            if firstNode:
                output = f"{node.__str__()}"
```

```python
                firstNode = False
            else:
                output += (', ' + f"{node.__str__()}")
            node = node.nextNode
        return f"[{output}]"
```

EncodedWord.py

```python
'''
Name: Ang Kah Shin
Class: DAAA/FT/2B/04
Admin: P2004176
'''

class EncodedWord:
   '''
   sentence: array
   morse: array
   '''
   def __init__(self, word, morse):
       self.word = word
       self.morse = morse
       self.morseLength = len(morse)
       self.char = morse.split(",")

   def maxChar(self):
       maxNum = 0
       for char in self.char:
           if len(char) > maxNum:
               maxNum = len(char)
       return maxNum

   def padding(self):
       padded = []
       maxLen = self.maxChar()
       for char in self.char:
           if len(char) < self.maxChar():
               padded.append(" " * (maxLen - len(char)) + char)
           else:
               padded.append(char)
       return padded

   def __str__(self):
       return f"{self.morse}"
```

Text.py

```python
from Node import Node

'''
Name: Ang Kah Shin
Class: DAAA/FT/2B/04
Admin: P2004176
'''


class Text(Node):
   def __init__(self, text, morse, x, y):
       self.text = text
       self.morse = morse
       self.morseLength = len(morse)
       self.x = x
       self.y = y
       super().__init__()

   def __eq__(self, otherNode):
       if otherNode == None:
           return False
       else:
           return self.x == otherNode.x and self.y == otherNode.y

   def __lt__(self, otherNode):
       if otherNode == None:
           raise TypeError(
               "'<' not supported between instances of 'Point' and 'NoneType'")
       if self.x < otherNode.x:
           return True
       elif self.x == otherNode.x:
           return self.y < otherNode.y
       else:
           return False

   def __str__(self):
       return f"{(self.x, self.y)}"
```