# Data Structures and Algorithms (AI) CA1 Assignment Report

Name: Ethan Tan

Admin: P2012085

Class: DAAA/2B/03

## Description

This report entails the documentation of a Morse Code Analyser program written in the Python programming language. The program has been designed to be user-friendly and configurable. It supports the encoding of plain text to morse code, printed either horizontally or vertically, and the decoding and opinionated analysis of a message written in morse code.

## User Guidelines

When run, the program will display a menu which offers users 4 options to choose from:

Text

Description automatically generated

As indicated above, the 4 choices are:

### Change Printing Mode

This option allows users to switch from horizontal printing mode to vertical and vice versa.



The default print mode is horizontal. Users can select horizontal/vertical mode by entering h/v respectively. Whitespace is ignored and the user’s input is case insensitive.



If the user does not enter any input, the current printing mode will be retained.



### Convert Plain Text to Morse Code

This option encodes a plain text message to morse code and prints it out;

Either horizontally:

Text

Description automatically generated

Or vertically:

A picture containing diagram

Description automatically generated

Multi-line input is also supported.

*\*Note: Do not include periods (.), dashes (-) or commas (,) as these are symbolic to the function*

### Analyse Morse Code Message

This option first decodes a morse code message in a file, then provides a breakdown of the frequencies of the words therein and constructs an essential message therefrom with the stop words filtered out.

The user will be prompted to enter an input file which contains the morse code message and an output file to write the report of the analysis to. If the input file does not exist or the output file name is invalid, the user will be prompted again correspondingly. If the user does not provide an output file, the report will not be saved. If the input file’s contents are in an invalid format, the process will be aborted, and the user will be redirected to the menu screen.

Text

Description automatically generated

Text

Description automatically generated Text

Description automatically generated

The decoded morse text will be displayed, followed by a breakdown of the individual words in the message [encoded word; frequency; positions – (line, word)], followed by the essential message.

## Data Structures Implemented

##### Morse\_Code\_Analyser

#### Description:

* This is the class for the main object to be run in the main program. It integrates all the core operational functionality mentioned above into one centralised object.

#### Purpose:

* Centralises all the major functionality related to operation of the program

#### Properties:

* \_\_author (contains details of the author)
* \_\_print\_mode (default printing mode)
* \_\_min\_significant\_frequency (minimum frequency of words to be shown in Option 3)

#### Methods (Only time complexities are analysed):

* run (the only public method)
* \_\_change\_printing\_mode\_1 [O(i); i = frequency of invalid input]
* \_\_convert\_text\_to\_morse\_2 [O(s); s = size of input]
* \_\_analyse\_morse\_message\_3 [O(i + o + s *log*(s)); i = frequency of invalid input files, o = frequency of invalid output files, s = size of input file’s contents]
* \_\_exit\_4 [O(1)]

*\*Note: size of input is the total length of the flattened input*

#### Reasons:

* Properties are all **encapsulated** as they should not be changed after instantiation
* Methods (besides run) are all **encapsulated** as they are for internal use only

#### Challenges Faced/Improvements Made:

* Code was distributed into modules to improve structure and design of the application, and reduce repetitiveness

##### Morse\_Utils

#### Description:

* This is an abstract class which contains all the functionality related to morse-to-plaintext translation. It cannot be instantiated, and its methods are all static.

#### Purpose:

* Centralises all the functionality related to morse-to-text translation

#### Methods:

* encode\_morse [O(s); s = size of the plaintext message]
* decode\_morse [O(s); s = size of the morse message]

*\*Note: size of input is the total length of the flattened input*

#### Reasons:

* Class is abstract as it is merely a namespace to hold correlated functionality
* Methods are public as they are meant for external use

#### Challenges Faced/Improvements Made:

* This class did not originally exist
* It improves organisation of code by using a “namespace” to group related functions

##### Abstract\_Stack

#### Description:

* This is an abstract interface for the Stack class. It merely declares what methods the Stack class should support

#### Purpose:

* Ensure that the Stack class supports all necessary functionality

#### Methods:

* push [O(1)]
* pop [O(1)]
* peek [O(1)]
* empty [O(1)]
* size [O(1)]

#### Reasons:

* Methods are all abstract as they are meant to be overridden by the Stack class

#### Challenges Faced/Improvements Made:

* None

##### Stack

#### Description:

* This is a generic Stack class, which support the common push, pop, peek, empty and size methods at O(1) time complexity

#### Purpose:

* Data structure with the LIFO constraint

#### Properties:

* \_\_head (top item of stack)
* \_\_size (size of the stack)

#### Methods:

* \_\_len\_\_ [O(1)] (alias for size method)
* \_\_iadd\_\_ [O(1)] (alias for push method)

*\*Note: with the addition of all the methods declared in Abstract\_Stack*

#### Reasons:

* **Polymorphism** is implemented as method overriding of methods in Abstract\_Stack
* **Polymorphism** is also implemented as operator overloading for the alias methods
* Properties are **encapsulated** as they are meant to be for internal use

#### Challenges Faced/Improvements Made:

* I had problems overloading the \_\_iadd\_\_ magic method
* The solution was to return self

##### Morse\_Character

#### Description:

* This is the class which is used to embody encoded text for vertical printing as in Option 2. This class inherits the Stack class

#### Purpose:

* Provide a logical data structure to meet the need for vertical printing

#### Properties/Methods:

* \_\_init\_\_ [O(1)] (loads a morse character in string format)

*\*Note: with all the properties and methods as defined in the Stack class it inherits from*

#### Reasons:

* Vertical printing is implemented using a list of Morse\_Character objects, as the morse characters are printed from the back to front – adheres to LIFO principle
* Morse\_Character class is used instead of the generic Stack class as it makes loading the string morse characters easier and more explicit

#### Challenges Faced/Improvements Made:

* I realised that I had the wrong impression of vertical printing – I thought it was bottom up when it was actually top down
* Solution was to reverse the word string in the constructor

##### Word

#### Description:

* This is an abstract class which is meant to model a generic word in the morse code message for Option 3

#### Purpose:

* Model a word object

#### Properties:

* \_word (word in string form)
* \_frequency (number of occurrences in message)

#### Methods:

* size [O(1)]
* getWord [O(1)]
* getFrequency [O(1)]
* addInstance [abstract]
* \_\_lt\_\_ [abstract]

#### Reasons:

* Properties are prefixed with single underscores as they are meant to be **protected** (private but inheritable)
* Abstract methods are to be overridden in subclasses, Message\_Breakdown\_Word and Essential\_Message\_Word

#### Challenges Faced/Improvements Made:

* Challenge was to overload the less than operator in 2 different ways
* Initial solution was to implement a custom key when sorting
* Final solution was to add 2 subclasses (Message\_Breakdown\_Word and Essential\_Message\_Word)

##### Message\_Breakdown\_Word

#### Description:

* This is the class of the word objects specific to the \_\_get\_message\_breakdown method

#### Purpose:

* Provide an easy way to compare words for sorting

#### Properties:

* \_\_all\_positions (stores all occurrences of the word in the message)

#### Methods:

* addInstance [O(1)] (register a new instance of the word, noting its position in the message)
* getDetails [O(w + f); w = length of word, f = frequency of word] (return details on the word)
* \_\_lt\_\_ [O(1)] (compares frequency descending, followed by size, followed by word)

#### Reasons:

* **Overloaded** < operator enables easy sorting of words by the conditions as stated in the brief
* getDetails collects all the necessary details and returns it to be inserted in the report

#### Challenges Faced/Improvements Made:

* Traded space complexity for time complexity by keeping track of an Essential\_Message\_Word alongside a Message\_Breakdown\_Word in \_\_get\_frequencies

##### Essential\_Message\_Word

#### Description:

* This is the class of the word objects specific to the \_\_get\_essential\_message method

#### Purpose:

* Provide an easy way to compare words for sorting

#### Properties:

* \_\_first\_position (stores the initial position of the word in the message)

#### Methods:

* addInstance [O(1)] (increments frequency by 1)
* getFirstPos [O(1)] (getter for the \_\_first\_position property)
* \_\_lt\_\_ [O(1)] (compares frequency descending, followed by first position)

#### Reasons:

* **Overloaded** < operator enables easy sorting of words by the conditions as stated in the brief

#### Challenges Faced/Improvements Made:

* Traded space complexity for time complexity by keeping track of a Message\_Breakdown\_Word alongside an Essential\_Message\_Word in \_\_get\_frequencies

## Algorithms Implemented

##### QuickSort

#### Description:

* This is an implementation of the commonly known quicksort algorithm, which performs efficiently for large lists through a divide-and-conquer approach

#### Purpose:

* Sort the list of word objects efficiently

#### Time Complexity (Average/Best):

* O(n *log*(n)); n = the number of items in the list

#### Reasons:

* Quicksort is used to sort the lists of Message\_Breakdown\_Word/Essential\_Message\_Word objects instead of SortedLists as it is more efficient
* SortedLists use insertion sort over many insertions to maintain state of being sorted, which has average time complexity of O(n­2)

#### Challenges Faced/Improvements Made:

* Initially, the list was sorted in-place
* Afterward, it was changed to return a sorted copy instead
* Had to tune the boundaries a few times to get partition working right

## Inheritance Tree

Text

Description automatically generated

## Appendix

### \_\_main\_\_.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  ## ENTRY POINT TO MAIN PROGRAM ##  # run "python ." or "python \_\_main\_\_.py" from this directory  # Import the main object  from tools.morse\_code\_analyser import Morse\_Code\_Analyser  # Declare a custom configuration dictionary  CONFIG = {      "author": {          "name": "Ethan Tan",          "admin": "2012085",          "class": "DAAA/2B/03",          "module": "ST1507 DSAA",      },      "min\_significant\_frequency": 1  }  # Instantiate the analyser object and run it  morse\_code\_analyser = Morse\_Code\_Analyser(config=CONFIG)  morse\_code\_analyser.run() |

### morse\_code\_analyser.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from typing import Dict, List, Literal, Set, Tuple, Union  from .data\_structures.morse\_character import Morse\_Character  from .data\_structures.message\_breakdown\_word import Message\_Breakdown\_Word  from .data\_structures.essential\_message\_word import Essential\_Message\_Word  from .sorting.quicksort import quicksort  from .utils.io\_utils import strip\_special\_characters, simple\_input, multi\_line\_input, file\_input, clear\_console  from .utils.morse\_utils import Morse\_Utils  from os.path import exists, isfile  # Main object  class Morse\_Code\_Analyser:      def \_\_init\_\_(self, config=None):          self.\_\_load\_config(config=config)      # Load custom configuration options      def \_\_load\_config(self, config=None):          DEFAULT\_CONFIG = {              'author': {                  'name': 'John Doe',                  'admin': '1234567',                  'class': 'UNKNOWN',                  'module': 'UNKNOWN',              },              'print\_mode': 'h',              'stopwords\_file': 'data/stopwords.txt',              'min\_significant\_frequency': 1          }          # Defaults to default configuration if key is not specified in config          CUSTOM\_CONFIG = DEFAULT\_CONFIG.copy()          if config is not None:              for k, v in config.items():                  CUSTOM\_CONFIG[k] = v          # Details to be displayed          self.\_\_author: Dict[str, Dict[str, str]] = CUSTOM\_CONFIG['author']          # Default printing mode          self.\_\_print\_mode: Union[Literal['h'],                                   Literal['v']] = CUSTOM\_CONFIG['print\_mode']          # Minimum lowest frequency to display in the get\_message\_breakdown method in Option 3          self.\_\_min\_significant\_frequency: int = CUSTOM\_CONFIG['min\_significant\_frequency']          # Relative path to file containing stop words          self.\_\_set\_stop\_words(file=CUSTOM\_CONFIG['stopwords\_file'])      # Operational Methods      # Runs the program      def run(self):          choice = 0          while choice < 4:              clear\_console()              # Displays the author's information              self.\_\_print\_info()              # Get the user's choice              choice = self.\_\_get\_choice()              if choice == 1:                  self.\_\_change\_printing\_mode\_1()              elif choice == 2:                  self.\_\_convert\_text\_to\_morse\_2()              elif choice == 3:                  self.\_\_analyse\_morse\_message\_3()              elif choice == 4:                  self.\_\_exit\_4()                  break              input('\nPress Enter to continue...')      # Allows the user to change the printing mode for Option 2      # Empty input will return to the menu without changing the printing mode      def \_\_change\_printing\_mode\_1(self):          mode = self.\_\_print\_mode          modes = {              'h': 'horizontal',              'v': 'vertical'          }          print(f'Current print mode is {mode}')          while True:              try:                  mode = simple\_input(                      f'Enter \'h\' for horizontal or \'v\' for vertical, then press enter: ')                  assert mode == 'h' or mode == 'v' or mode == ''                  break              except Exception:                  print('Invalid input')          if mode != '':              self.\_\_print\_mode = mode              print('The print mode has been changed to', modes[mode])          else:              print('Operation cancelled by user. The print mode remains as',                    modes[self.\_\_print\_mode])      # Converts a (multi-line) message in plain text to morse code and prints it out      #   based on the current printing mode      def \_\_convert\_text\_to\_morse\_2(self):          print('Enter text to be converted:')          lines = multi\_line\_input()          morse = Morse\_Utils.encode\_morse(lines)          if self.\_\_print\_mode == 'h':              self.\_\_print\_morse\_h(morse)          else:              self.\_\_print\_morse\_v(morse)      # Converts a morse code message in a file to plain text      # Displays a breakdown of the frequencies of each word in the message      # Displays the essential message, with stop words removed      def \_\_analyse\_morse\_message\_3(self):          try:              input\_file = self.\_\_get\_input\_file()              decoded\_text = self.\_\_get\_decoded\_message(input\_file=input\_file)              output\_file = self.\_\_get\_output\_file()              stripped\_decoded\_text = strip\_special\_characters(decoded\_text)              message\_breakdown\_ls, essential\_message\_ls = self.\_\_get\_frequencies(                  text=stripped\_decoded\_text)              message\_breakdown = self.\_\_get\_message\_breakdown(                  word\_ls=message\_breakdown\_ls)              essential\_message = self.\_\_get\_essential\_message(                  stop\_words=self.\_\_stop\_words, word\_ls=essential\_message\_ls)              report = self.\_\_build\_report(                  decoded\_message=decoded\_text, message\_breakdown=message\_breakdown, essential\_message=essential\_message)              print('\n', report, sep='')              self.\_\_save\_report(message=report, file=output\_file)          except AssertionError as err:              print(err, 'Aborting...')      # Displays a friendly farewell message      def \_\_exit\_4(self):          print('Bye, thanks for using {}: Morse Code Analyser!'.format(              self.\_\_author['module']))      # Utility Methods      # Option 2      # Prints the encoded string horizontally      @staticmethod      def \_\_print\_morse\_h(morse: str):          print(morse)      # Prints the encoded string vertically      @staticmethod      def \_\_print\_morse\_v(morse: str):          lines = morse.splitlines()          for line in lines:              ls = []              # A list is used to collect the characters as              #   repeated appending has an amortized worst case time complexity of O(n) whereas              #   repeated string concatenation has a time complexity of O(n^2)              print\_ls = []              for char in line.split(sep=','):                  ls.append(Morse\_Character(                      morse\_char=char, pad\_char=' ', padding=5))              # Each morse character has a maximum length of 5              for \_ in range(5):                  for char in ls:                      print\_ls.append(char.pop())                  print\_ls.append('\n')              print(''.join(print\_ls))      # Option 3      # Loads the stop words from the file specified      def \_\_set\_stop\_words(self, file: str):          with open(file=file, mode='r') as f:              stop\_words = {w.upper() for w in f.read().splitlines()}          self.\_\_stop\_words = stop\_words      # Tries recursively to obtain a valid input file from the user      def \_\_get\_input\_file(self):          try:              input\_file = file\_input('Enter input file:  ')              assert exists(input\_file), 'Invalid input file'              assert isfile(input\_file), 'Not a file'              return input\_file          except AssertionError as err:              print(err)              return self.\_\_get\_input\_file()      # Tries recursively to obtain a valid output file from the user      # Empty input will cause report to not be saved      def \_\_get\_output\_file(self):          try:              output\_file = file\_input('Enter output file: ')              if output\_file == '':                  return ''              with open(file=output\_file, mode='w') as f:                  f.write('')              return output\_file          except FileNotFoundError:              print('Invalid output file name')              return self.\_\_get\_output\_file()      # Decodes morse code message from input file      def \_\_get\_decoded\_message(self, input\_file: str):          decoded\_message = Morse\_Utils.decode\_morse(input\_file)          return decoded\_message      # Returns a list of words to be used by \_\_get\_message\_breakdown and \_\_get\_essential\_message methods      def \_\_get\_frequencies(self, text: str):          word\_dict: Dict[str, Tuple[Message\_Breakdown\_Word,                                     Essential\_Message\_Word]] = {}          for line\_index, line in enumerate(text.splitlines()):              for word\_index, word in enumerate(line.split(sep=' ')):                  if word in word\_dict:                      word\_dict[word][0].addInstance((line\_index, word\_index))                      word\_dict[word][1].addInstance()                  else:                      word\_dict[word] = (                          Message\_Breakdown\_Word(                              word=word, first\_pos=(line\_index, word\_index)),                          Essential\_Message\_Word(                              word=word, first\_pos=(line\_index, word\_index))                      )          message\_breakdown\_word\_list = []          essential\_message\_word\_list = []          for mbw, emw in word\_dict.values():              message\_breakdown\_word\_list.append(mbw)              essential\_message\_word\_list.append(emw)          return message\_breakdown\_word\_list, essential\_message\_word\_list      # Analyses frequencies and positions of the words in the morse code message      def \_\_get\_message\_breakdown(self, word\_ls: List[Message\_Breakdown\_Word]):          sorted\_words: List[Message\_Breakdown\_Word] = quicksort(              word\_ls)          try:              previous\_frequency = sorted\_words[0].getFrequency()          except IndexError:              previous\_frequency = self.\_\_min\_significant\_frequency - 1          message\_breakdown = []          if previous\_frequency >= self.\_\_min\_significant\_frequency:              message\_breakdown.append(                  f'\*\*\* Morse words with frequency = {previous\_frequency}\n')          for word in sorted\_words:              current\_frequency = word.getFrequency()              if current\_frequency < self.\_\_min\_significant\_frequency:                  break              if current\_frequency < previous\_frequency:                  message\_breakdown.append(                      f'\n\*\*\* Morse words with frequency = {current\_frequency}\n')                  previous\_frequency = current\_frequency              message\_breakdown.append(word.getDetails())          return ''.join(message\_breakdown)      # Analyses essential message through sorting by frequency and first position of the words      def \_\_get\_essential\_message(self, stop\_words: Set[str], word\_ls: List[Essential\_Message\_Word]):          sorted\_words: List[Essential\_Message\_Word] = quicksort(              word\_ls)  # type: ignore          essential\_message = []          for word in sorted\_words:              w = word.getWord()              if w not in stop\_words and w.isalpha():                  essential\_message.append(w)          return ' '.join(essential\_message)      # Builds report from the three components      def \_\_build\_report(self, decoded\_message: str, message\_breakdown: str, essential\_message: str):          report = '\*\*\* Decoded morse text\n' + decoded\_message + '\n' \              if decoded\_message != '' \              else '\*\*\* No decoded morse text\n'          report += message\_breakdown + '\n' \              if message\_breakdown != '' \              else f'\*\*\* No morse words with frequency >= {self.\_\_min\_significant\_frequency}\n'          report += '\*\*\* Essential Message\n' + essential\_message \              if essential\_message != '' \              else '\*\*\* No essential message'          return report      # Saves report to the output file, if specified      def \_\_save\_report(self, message: str, file: str):          if file != '':              with open(file=file, mode='w') as f:                  f.write(message)      # Generic Utility Methods      # Retrieves valid option from user, recursively      def \_\_get\_choice(self):          print(              'Please select your choice (1, 2, 3 or 4):\n \              \t1.  Change printing mode\n \              \t2.  Convert plain text to morse code\n \              \t3.  Analyse morse code message\n \              \t4.  Exit'.expandtabs(tabsize=4)          )          try:              choice = int(input('Enter your choice: '))              assert 1 <= choice <= 4          except Exception:              print('Invalid input')              return self.\_\_get\_choice()          return choice      # Prints the author's details      def \_\_print\_info(self):          print('\*' \* 57)          print(              f'''\*\t{self.\_\_author['module']}: Morse Code Message Analyser\t\*''')          print('\*' + '-' \* 55 + '\*')          print('\*\t\t\t\t\t\t\t\*')          print(f'''\*\t- Done By: {self.\_\_author['name']} ({self.\_\_author['admin']})\t\t\t\*''')          print(f'''\*\t- Class: {self.\_\_author['class']}\t\t\t\t\*''')          print('\*\t\t\t\t\t\t\t\*')          print('\*' \* 57) |

### data\_structures/abstract\_stack.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from abc import ABC, abstractmethod  # Template for Stack class  class Abstract\_Stack(ABC):      @abstractmethod      def push():          pass      @abstractmethod      def peek():          pass      @abstractmethod      def pop():          pass      @abstractmethod      def empty():          pass      @abstractmethod      def size():          pass |

### data\_structures/essential\_message\_word.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from typing import Tuple  from .word import Word  # Model for a word in the essential message  class Essential\_Message\_Word(Word):      def \_\_init\_\_(self, word: str, first\_pos: Tuple[int, int]):          super().\_\_init\_\_(word=word)          self.\_\_first\_position: Tuple[int, int] = first\_pos      def addInstance(self) -> None:          self.\_frequency += 1      def getFirstPos(self) -> Tuple[int, int]:          return self.\_\_first\_position      def \_\_lt\_\_(self, otherWord) -> bool:          return (-self.getFrequency(), self.getFirstPos()) < (-otherWord.getFrequency(), otherWord.getFirstPos()) |

### data\_structures/message\_breakdown\_word.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from typing import List, Tuple  from .word import Word  from ..utils.morse\_utils import Morse\_Utils  # Model for a word in the message breakdown  class Message\_Breakdown\_Word(Word):      def \_\_init\_\_(self, word: str, first\_pos: Tuple[int, int]):          super().\_\_init\_\_(word=word)          self.\_\_all\_positions: List[Tuple[int, int]] = [first\_pos]      def addInstance(self, instance\_position: Tuple[int, int]) -> None:          self.\_\_all\_positions.append(instance\_position)          self.\_frequency += 1      def getDetails(self) -> str:          details = ''          details += Morse\_Utils.encode\_morse(self.\_word)          details += f'[{self.\_word}] ({self.\_frequency}) {self.\_\_all\_positions}\n'          return details      def \_\_lt\_\_(self, otherWord) -> bool:          return (-self.getFrequency(), self.size(), self.getWord()) < (-otherWord.getFrequency(), otherWord.size(), otherWord.getWord()) |

### data\_structures/morse\_character.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from .stack import Stack  # Models a morse character for vertical printing  class Morse\_Character(Stack):      def \_\_init\_\_(self, morse\_char: str, pad\_char: str = ' ', padding: int = 5):          super().\_\_init\_\_()          # Pads all characters to length 5, as longest morse character is 5,          #   thus enforcing O(1) time complexity          for symbol in morse\_char[::-1]:              self += symbol          for \_ in range(padding - self.size()):              self.push(pad\_char) |

### data\_structures/node.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Class for an individual node in the Stack class  class Node:      def \_\_init\_\_(self, val, next\_node=None):          # Value property is immutable after initialisation          self.\_\_value = val          # Next node is not encapsulated          self.next = next\_node      # Getter for value      def get\_value(self):          return self.\_\_value |

### data\_structures/stack.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from .abstract\_stack import Abstract\_Stack  from .node import Node  # Generic linear data structure implementing LIFO principle  class Stack(Abstract\_Stack):      def \_\_init\_\_(self):          self.\_\_head = None          self.\_\_size = 0      def push(self, val):          new\_node = Node(val=val)          if self.\_\_head is None:              self.\_\_head = new\_node          else:              new\_node.next = self.\_\_head              self.\_\_head = new\_node          self.\_\_size += 1      def peek(self):          if self.\_\_head is not None:              return self.\_\_head.get\_value()          else:              return None      def pop(self):          tmp\_val = None          if self.\_\_head is not None:              tmp\_val = self.\_\_head.get\_value()              self.\_\_head = self.\_\_head.next              self.\_\_size -= 1          return tmp\_val      def \_\_len\_\_(self) -> int:          return self.\_\_size      def \_\_iadd\_\_(self, val):          self.push(val=val)          return self      def empty(self):          self.\_\_size = 0          self.\_\_head = None          return self      def size(self) -> int:          return self.\_\_size |

### data\_structures/word.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from abc import ABC, abstractmethod  # Models a word in the morse code message  class Word(ABC):      def \_\_init\_\_(self, word: str):          self.\_word: str = word.upper()          self.\_frequency: int = 1      def size(self) -> int:          return len(self.\_word)      def getWord(self) -> str:          return self.\_word      def getFrequency(self) -> int:          return self.\_frequency      @abstractmethod      def addInstance(self):          pass      @abstractmethod      def \_\_lt\_\_(self, otherWord):          pass |

### sorting/quicksort.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # type: ignore  # Import Dependencies  from typing import List  # Swap 2 values in the list by specified indices  def swap(ls, i, j):      ls[j], ls[i] = ls[i], ls[j]  # Recursive body of the algorithm  def partition(ls, l, h):      if l < h:          partition\_index = pointer = l - 1          partition\_num = ls[h]          for el in ls[l:h]:              pointer += 1              if el < partition\_num:                  partition\_index += 1                  swap(ls, pointer, partition\_index)          partition\_index += 1          swap(ls, h, partition\_index)          partition(ls, l, partition\_index - 1)          partition(ls, partition\_index + 1, h)  # Main function  # Sorts an arbitrary list using the quicksort algorithm  # Returns a new list, sorted  def quicksort(ls: List):      ls\_copy = ls.copy()      partition(ls\_copy, 0, len(ls) - 1)      return ls\_copy |

### utils/io\_utils.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  import os  import re  # Clears console, imitates reloading the screen  def clear\_console():      os.system('cls' if os.name == 'nt' else 'clear')  # Removes all internal whitespace  def whitespace\_reducer(text: str):      return re.sub(pattern='[\\s]+', repl=' ', string=text)  # Converts non alphanumeric characters to whitespace  def strip\_special\_characters(text: str):      return re.sub(pattern='[^A-Za-z0-9\\s]', repl=' ', string=text)  # Retrieves formatted multi-line input  def multi\_line\_input():      lines = []      while True:          line = input()          if line:              lines.append(whitespace\_reducer(line))          else:              break      return "\n".join(lines)  # Retrieves formatted input  def simple\_input(prompt: str):      return whitespace\_reducer(input(prompt)).strip().lower()  # Retrieves formatted file path input  def file\_input(message: str):      return whitespace\_reducer(input(message)).strip() |

### utils/morse\_utils.py

|  |
| --- |
| # Name:     Ethan Tan  # Admin:    P2012085  # Class:    DAAA/2B/03  # Import Dependencies  from abc import ABC, abstractmethod  # Merely a namespace for morse-to-text translation functions  class Morse\_Utils(ABC):      @abstractmethod      def \_\_init\_\_(self):          pass      # Encodes plain text to morse code      @staticmethod      def encode\_morse(plain\_text: str):          TEXT\_TO\_MORSE = {              "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": "-----",          }          morse\_ls = []          contents = plain\_text.upper().splitlines()          for line in contents:              for char in line:                  morse\_ls.append(                      TEXT\_TO\_MORSE[char] if char in TEXT\_TO\_MORSE else char)                  morse\_ls.append(",")              morse\_ls.pop()              morse\_ls.append('\n')          return ''.join(morse\_ls)      # Decodes morse code to plain text      @staticmethod      def decode\_morse(file: str):          MORSE\_TO\_TEXT = {              ".-": "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",          }          try:              with open(file=file) as f:                  contents = f.read().splitlines()                  text\_ls = []                  for line in contents:                      for char in line.split(","):                          text\_ls += MORSE\_TO\_TEXT[char] if "." in char or "-" in char else char                      text\_ls.append("\n")              return "".join(text\_ls)          except KeyError:              raise AssertionError(                  f'Morse code in file {file} is in invalid format.') |

*~ END ~*