FIT2004 Assignment 2

FIT2004 Assignment 2

Student: Siew Ming Shern
StudentID: 28098552

Siew

Task 1

```
def messageFind(filename):

    let filecontent be a list of words

    read filename to filecontent

    let n be length of first string

    let m be length of second string

    let memo be a list containing m number of list with n zero initialization.

    for N in range(1,n+1):
        for M in range(1,m+1):
            if filecontent[1][M-1] != filecontent[0][N-1]:
                memo[N][M] = max(memo[N - 1][M], memo[N][M - 1])
            else:
                memo[N][M] = 1 + memo[N-1][M-1]

    string = ""
    while n > 0 and m > 0:
        if filecontent[1][m-1] is not equal to  filecontent[0][n-1] and
        memo[n-1][m] > memo[n][m-1]:
            n -= 1
        elif filecontent[1][m-1] is equal to filecontent[0][n-1]:
            string += filecontent[0][n - 1]
            n -= 1
            m -= 1
        else:
            m -= 1

    self.message = ""
    for char in range(len(string)-1,-1,-1):
        self.message += string[char]
```

Time complexity: O(nm), occurs when reading first text and seconds text from file. No early termination since every character of n word and every character of m word is needed to read and compare, and backtracking would also took O(nm) for backtracking either n words or m words to obtain optimal solution. Thus, resulting O(nm)

Space complexity: O(nm), no matter what length of two string is, ardency matrix (nm size) is created to store optimal solution of two string for each iteration.

where n and m be the size of first text and second text respectively.

Task 2

maxDictLength(dict[])method called to check for length of longest word

```
def wordBreak (filename):
    read dictionary file to dictionary for list of words named dict[]

    #get instance message
    currentMessage = self.getMessage()
    #get longest words of dictionary and length of instance message
    M = self.maxDictLength(dict)
    k = len(currentMessage)

    possible = [[0 for i in range(k)] for i in range(M)]

    for m in range(M):
        k -= m
        for j in range(k):
            if currentMessage[j:j+m+1] in dict:
                    possible[m][j] = 1
        k += m
    #reinitialise k with length of instance message
    k = len(currentMessage)

    #create a memo which store number of character that current index i
    can iterate for k character
    memo = [0 for i in range(k)]
    #looping from longest length of dictionary to 0

    for m in range(M-1,-1,-1):
        #looping for length of instance message to 0
        for j in range(k):
            if possible[m][j] == 1:
                flag = True
                #looping through m times to check if any value after
                the index i..i+m is allocated already
                for i in range(m+1):
                    if m + 1 <= memo[j + i]:
                        flag = False
                        break
                if flag:
                #when all condition is fulfilled, loop through
                j..i+j to allocated the new substrings
                    for i in range(m+1):
                        memo[j + i] = m + 1
```

```
#use flaging to check if last iteration is a words
isLastAWord = False

#get counter to allocate substring to new instance message
counter = 0

#reset instance message
self.message = ""
while counter < len(memo):
    #when character added is > 0, it indicate it is a string from
    dictionary
    if memo[counter] > 0:
        #flag as current word is word from dictionary
        isLastAWord = True
        # as long as counter is after first substring
        if counter > 0:
            # add extra space
            self.message += " "
        #add the substring of instance message that found to new
         instance message
        for j in range(memo[counter]):
            self.message += currentMessage[counter+j]
        counter += memo[counter]
    else:
        #when last iteration is a word and counter is after first
        substring
        # add a space before adding words
        if isLastAWord and counter > 0:
            isLastAWord = False
            self.message += " "
        #add a character that not in dictionary into instance
         message
        self.message += currentMessage[counter]
        counter += 1
```

Time complexity: O(k*M*N*M), occurs when checking if each combination of substrings in instance string is in dictionary. The time complexity gets worse when all n word in dictionary are having M character, this result each substring of input string of m size in instance message of k size is needed to compare all n words in dictionary with M character.

Space complexity: O(k*M+N*M), occurs when list of dictionaries is read from file which would takes O(NM) and adjacency matrix is created to check if substring in input string is in dictionary which would takes O(kM). Therefore, space complexity would takes O(kM+NM) for adjacency matrix and list of dictionary.

where k is input string, N be the number of words in dictionary and M be the maximal size of the words in dictionary respectively.