DATA STRUCTURES AND ANALYSIS OF ALGORITHMS

EXPERIENTIAL LEARNING ASSIGNMENT

BML MUNJAL UNIVERSITY™

FROM HERE TO THE WORLD

SUBMITTED BY: -

MATURY SATYA SAI SAKETH

190C2030076

CSE-2

SUBMITTED TO: -

DR. SOHARAB HOSSAIN SIR

PROBLEM STATEMENT

Bob is a computer security expert working in XYZ Ltd. Recently a group of computers of XYZ Ltd. Has been hacked by a malicious virus. All the files residing in those computers are encrypted and therefore, are not useful. However, those files are very important and the organization has no backup copies. Bob is to get the key (password) to decrypt those files.

After a thorough run through all the computers in the network, Bob has found the malicious program that actually did all the mischief. He debugged the code and came to the conclusion that the key(password) can be formed by the following steps:

i) There are two log files (text files) residing in a local server. Find the most frequently

occurring three words that are common in both the log files.

ii) From the above three words, find the string representing the longest common

subsequence (pair-wise strings to be taken).

iii) The binary string representing the optimal encoding of the longest common subsequence will form the key.

Bob wants to write a program that will generate the key. Put yourself in Bob's shoes and try to

solve the problem.

Take two text files as input for the log files. Make sure there are some common words in the files.

PROBLEM EXPLANATION

Recently a group of computers of XYZ Ltd. Has been hacked by a malicious virus. All the files residing in those computers are encrypted and therefore, are not useful. However, those files are very important and the organization has no backup copies

But there is Bob the computer expert , He has to find the password or a key to decrypt the files using some techniques So Bob is figuring out a solution for the problem??

STEP – 1 Explanation

Finding the most common frequently occurring three words

- Firstly we will import the counter class from the module of collections available in python
- Next, We will split the string into list using split() function it will return the word
- The function most common () inside the Counter class will help to return the list of most frequent words and the count
- Apply the same method for the 2nd log file also
- Then we will be adding the frequencies of same words in the two lists and the top three words with higher frequency where sum is returned.

Algorithm :-

```
import numpy as np

import heapq

from collections import defaultdict

from collections import Counter

log1 = open("encrypt1.txt")

log2 = open("encrypt2.txt")

data1 = log1.read()

data2 = log2.read()

split_it1 = data1.split()

split_it2 = data2.split()


Counter1 = Counter(split_it1)

Counter2 = Counter(split_it2)
```

```
Counter1 += Counter2

freoc = Counter1.most_common(3)
```

STEP – 2 Explanation

Longest Common Subsequence Algorithm(LCS) :-

Meaning:- The **longest common subsequence** (**LCS**) **problem** is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from the longest common substring problem.

Here we will be using LCS implemented using dynamic programing approach than using recursive approach because the method of dynamic programming will reduce the number of function calls everytime. It stores the result of each function call so that it can be used in future calls without the need for redundant calls.

So, the time taken by a dynamic approach will be O(mn)). Whereas the recursion algorithm will have the complexity of 2 Max(m,n)

Algorithm:-

```
def lcs(str1, str2):
    a = len(str1)
    b = len(str2)
    string_matrix = [[0 for i in range(b+1)] for i in range(a+1)]
    for i in range(1, a+1):
        for j in range(1, b+1):
            if i == 0 or j == 0:
                string_matrix[i][j] = 0
            elif str1[i-1] == str2[j-1]:
                string_matrix[i][j] = 1 + string_matrix[i-1][j-1]
            else:
                string_matrix[i][j] = max(string_matrix[i-1][j], string_matrix[i][j-1])
    index = string_matrix[a][b]
```

```
res = [""] * index
i = a
j = b
while i > 0 and j > 0:
    if str1[i-1] == str2[j-1]:
        res[index-1] = str1[i-1]
        i -= 1
        j -= 1
        index -= 1
    elif string_matrix[i-1][j] > string_matrix[i][j-1]:
        i -= 1
    else:
        j -= 1
return res
```

STEP – 3 Explanation

In, the third step we will be using the Huffman Encoding to obtain our Key or the passworrd

HUFFMAN ALGORITHM :- Huffman coding is a lossless data compression algorithm. In this algorithm, a variable-length code is assigned to input different characters. The code length is related to how frequently characters are used. Most frequent characters have the smallest codes and longer codes for least frequent characters.

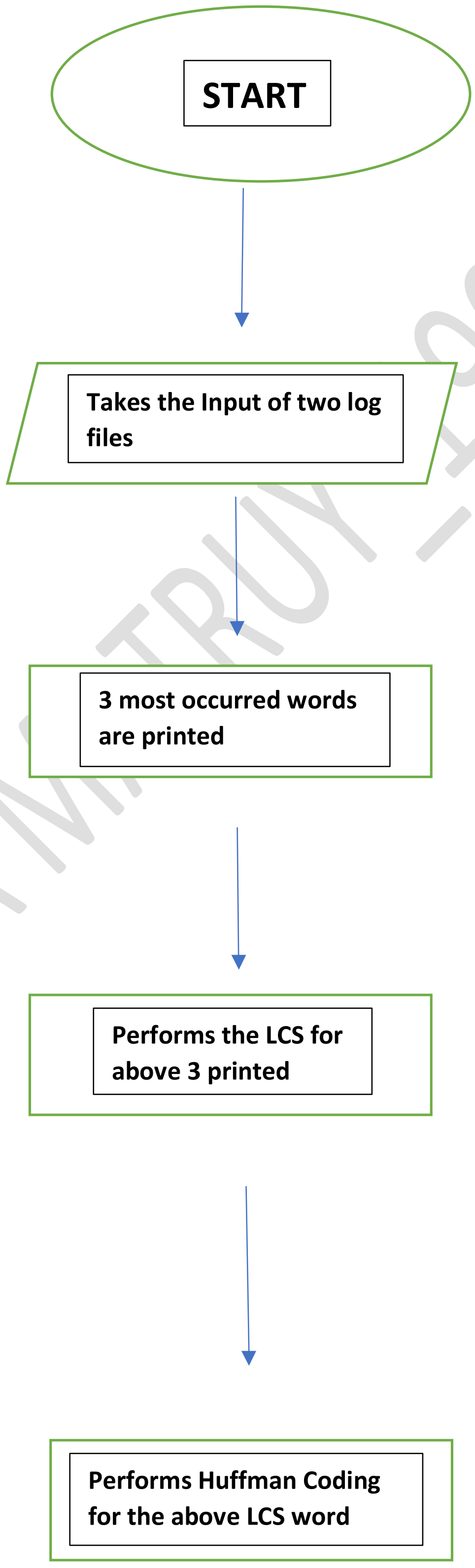
Algorithm:-

```
def encode(frequency):
    heap = [[weight, [symbol, '']] for symbol, weight in frequency.items()]
```

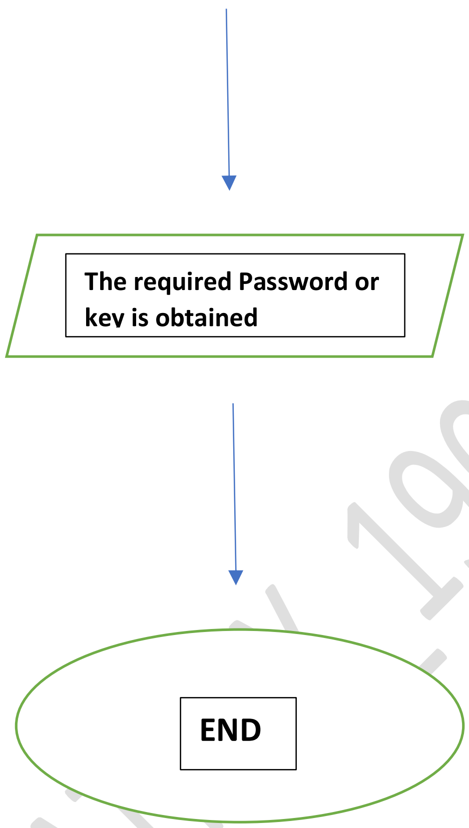
```
heapq.heapify(heap)

while len(heap) > 1:

    lo = heapq.heappop(heap)

    hi = heapq.heappop(heap)

    for pair in lo[1:]:

        pair[1] = '0' + pair[1]

    for pair in hi[1:]:

        pair[1] = '1' + pair[1]

    heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])

return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))
```

Complexity for assigning the code for each character according to their frequency is O(n log n)

FLOWCHART



START
Takes the Input of two log files
3 most occurred words are printed
Performs the LCS for above 3 printed
Performs Huffman Coding for the above LCS word


The required Password or key is obtained
END

CODE

```
import numpy as np
import heapq
from collections import defaultdict
from collections import Counter
log1 = open("encrypt1.txt")
log2 = open("encrypt2.txt")
data1 = log1.read()
data2 = log2.read()
split_it1 = data1.split()
split_it2 = data2.split()

Counter1 = Counter(split_it1)
Counter2 = Counter(split_it2)
Counter1 += Counter2

freoc = Counter1.most_common(3)

print("The most frequently occuring 3 words after going through the two log fi
les  are: ")
print(freoc)
print("                        |                      ")
print("                        |                      ")

#***********Finding the string representing the longest common subsequence ***
********

def lcs(str1, str2):
    a = len(str1)
    b = len(str2)
    string_matrix = [[0 for i in range(b+1)] for i in range(a+1)]
    for i in range(1, a+1):
        for j in range(1, b+1):
            if i == 0 or j == 0:
                string_matrix[i][j] = 0
            elif str1[i-1] == str2[j-1]:
                string_matrix[i][j] = 1 + string_matrix[i-1][j-1]
            else:
                string_matrix[i][j] = max(string_matrix[i-
1][j], string_matrix[i][j-1])
```

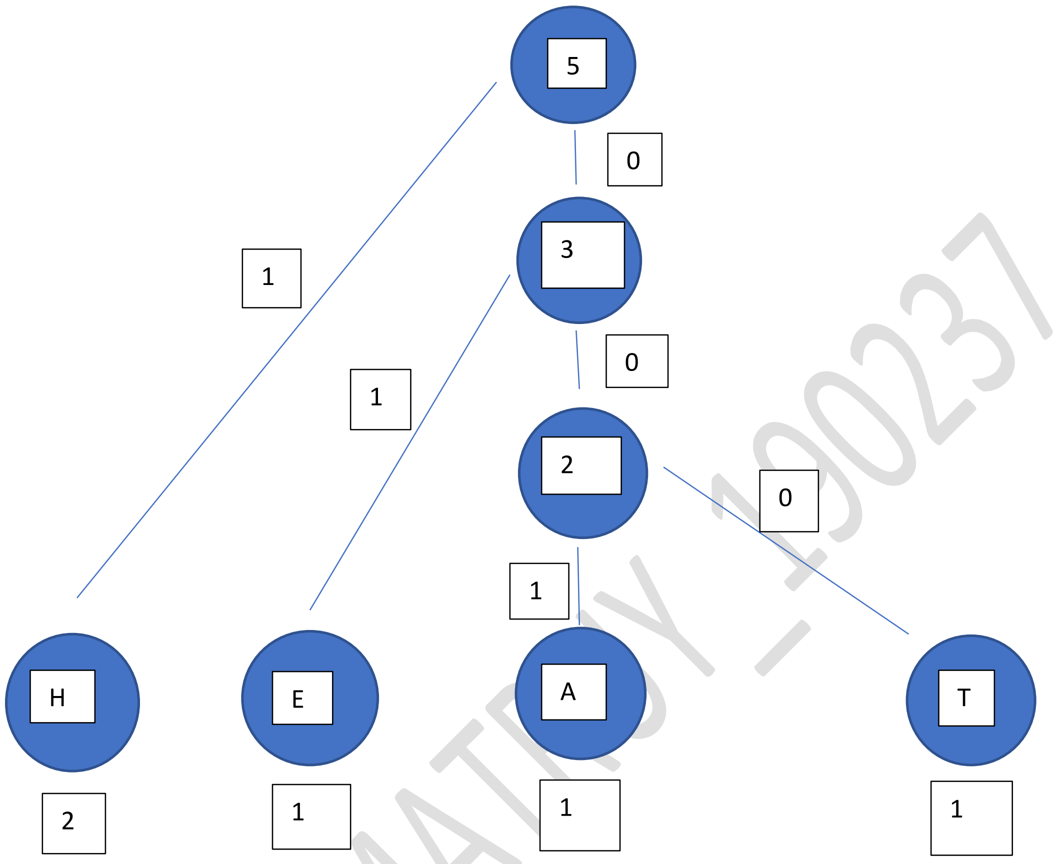
```python
    index = string_matrix[a][b]
    res = [""] * index
    i = a
    j = b
    while i > 0 and j > 0:
        if str1[i-1] == str2[j-1]:
            res[index-1] = str1[i-1]
            i -= 1
            j -= 1
            index -= 1
        elif string_matrix[i-1][j] > string_matrix[i][j-1]:
            i -= 1
        else:
            j -= 1
    return res

str1 = freoc[0][0] + freoc[1][0]
str2 = freoc[1][0] + freoc[2][0]
str3 = freoc[2][0] + freoc[0][0]
lcstring1 = ''.join(lcs(str1, str2))
lcstring2 = ''.join(lcs(lcstring1, str3))
print("Length of least common subsequence is:", len(lcstring2),"\n The common
subsequence is:", lcstring2)

    #Optimal encoding of the longest common subsequence


def encode(frequency):
    heap = [[weight, [symbol, '']] for symbol, weight in frequency.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))


data = lcstring2
frequency = defaultdict(int)
for symbol in data:
    frequency[symbol] += 1

huff = encode(frequency)
print ("Character".ljust(10) + "Weight".ljust(10) + "Huffman Code")
for p in huff:
    print (p[0].ljust(10) + str(frequency[p[0]]).ljust(10) + p[1])

keystring=''
for char in data :
    for p in huff:
        if char==p[0] :
            keystring+=p[1]

print("The required password(key) is: "+ keystring)
```

TIME COMPLEXITY

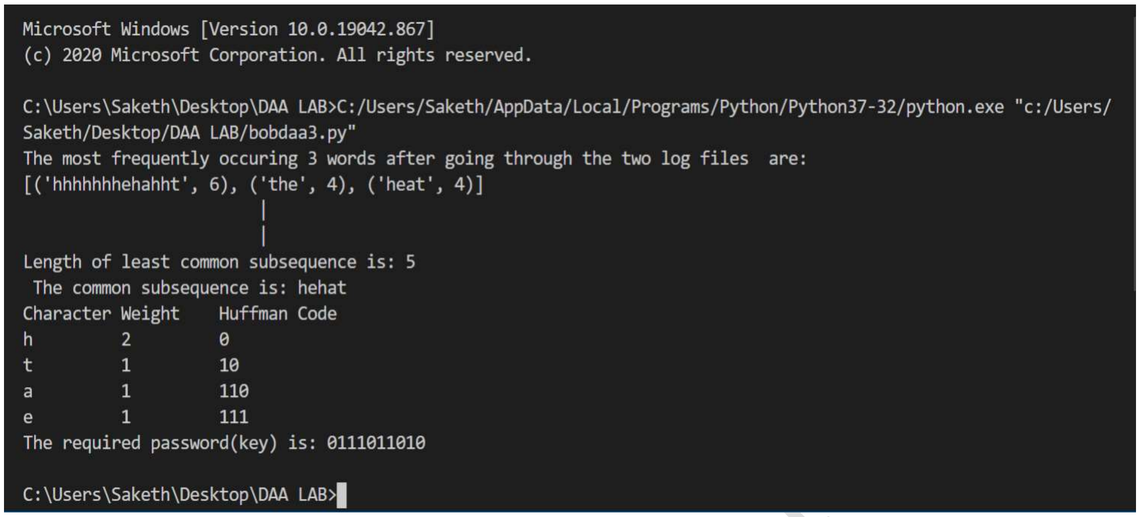
THE RESPECTIVE TIME COMPLEXITIES FOR THE ABOVE CODE IS :-

- **Longest common subsequence –** O(mn)
- **Counter –** O(n)
- **Huffman Encoding –** O(nlogn)
- **Overall Time Complexity –** O(mn)

TREE DIAGRAM FOR HUFFMAN CODING



OUTPUT



```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Saketh\Desktop\DAA LAB>C:/Users/Saketh/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/
Saketh/Desktop/DAA LAB/bobdaa3.py"
The most frequently occuring 3 words after going through the two log files  are:
[('hhhhhhhehahht', 6), ('the', 4), ('heat', 4)]
                        |
                        |
Length of least common subsequence is: 5
 The common subsequence is: hehat
Character Weight    Huffman Code
h         2         0
t         1         10
a         1         110
e         1         111
The required password(key) is: 0111011010

C:\Users\Saketh\Desktop\DAA LAB>
```