Custom Vigenère Encryption Using Huffman Coding Algorithm

Antonio Natthan Krishna - 13521162[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521162@std.stei.itb.ac.id

Abstract—This paper discusses simple implementation of Vigenère cipher on compressed text using Huffman Code. This method of encryption is easy to understand and suitable for beginners as an introductory material to Cryptography. It uses some of the most foundational topics in Computer Science e.g., Binary Tree, Combinatorics, and Graph. This paper also includes implementation of data structures, algorithms, and experiments to some test cases.

Keywords— Binary Tree, Huffman Coding, Cryptography, Vigenère Cipher.

I. Introduction

Cryptography is one of the most crucial parts of computer science. Computer science students, like me, are required to understand the process of cryptography which includes the process of encryption and decryption. There are many encryption and decryption algorithms out there, from the simplest i.e., Caesar cipher, to the more advanced and high-tech algorithms e.g., Advanced Encryption Standards (AES) and RSA Security. Since Caesar cipher is no longer a secure algorithm, the understanding of advanced encryption algorithms is needed.

However, the advanced algorithms that I stated above is not easy to comprehend at all. Caesar cipher is still mandatory material in cryptography. The more "high-end" simple algorithm, Vigenère cipher, is one way to dig deeper in cryptography. The comprehension to those 2 topics is important before learning advanced algorithms that are applicable in real life.

This paper discusses Vigenère cipher algorithm. We do not encrypt original text but compressed text using Huffman coding algorithm. Huffman coding algorithm one way to compress text, so we do not have store or send text in its original size. Huffman coding algorithm is similar to general encryption algorithms, where we try to convert original text to another form, but since we have to store all Huffman code in a list, Huffman coding cannot be categorized as encryption algorithms.

I try to combine Huffman coding compressed text and Vigenère cipher to implement a new way to encrypt a text. By using this method, we can have a secure and minimal-size text. This method can be taught to computer science students as an introduction to cryptography because it uses fundamental concepts that computer science students must have, i.e., graph, binary-tree, and combinatorics.

This method might not be the most effective algorithm, but this is suitable for beginners who are interested to learn cryptography

II. Theory and Concepts

A. Introduction to Cryptography

Cryptography is a subject that discusses secret transmission of messages between two parties. The first party (let us say as A) will translate the message to some unknown language. The message will be delivered to destination party (let us say as B). After the message is delivered, B would translate it back to normal form, so that B would understand what A is trying to say. The goal of this system is that there is no one, except A and B, would understand the message. In today's world, cryptography is a foundation of cyber security. Internet account, text messages, calls, and video calls use cryptography to keep user's personal information safe.

Cryptography consists of encryption and decryption. Encryption is a process to convert original message to another form, while conversely decryption bringing back encrypted messages to its original form. There are many algorithms to encrypt and decrypt messages. One of the most secure examples like Triple DES, Blowfish, Twofish and many more. In the next section, we will discuss one of the most ancient but powerful cipher algorithms (Vigenère cipher) that has been a foundation of many advanced algorithms I stated previously.

B. Vigenère Cipher

The idea used in Vigenère cipher is instead using single cipher key to encrypt all character in data test, we use a string of key which we called as password to encrypt our data test. The enciphering process starts with repeating user's password as many times as necessary to span the length of data test.

Every character has its own ID, we use Caesar cipher for a particular character with corresponding password character as its cipher key. Yes, Vigenère cipher is technically just like Caesar cipher, but it uses different cipher key for different character.

For example, suppose we are going to encrypt only uppercase letter of alphabet. Our password is "KEY" and we are going to encrypt the word "HELLO" using Vigenère cipher. So, the process of enciphering is listed below,

1. “HELLO” is consisted by 5 characters, then we span the word “KEY” until there are 5 characters.

K	E	Y	K	E

2. Now, each letter in “HELLO” has a corresponding password letter.

K	E	Y	K	E
H	E	L	L	O

3. Using Caesar cipher, we encrypt each character in row 2, with cipher key as listed in row 1. For example, we have encryption table for uppercase letter,

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

The encryption process uses this formula,

Cipher Text = Original Text + Key

The letter H (7) would be encrypted using key K (10), then the cipher text letter corresponds to alphabet 7 + 10 = 17 (R). If enciphering process results a number that is larger than 25, use modulo concepts e.g., L (11) with key Y (24) enciphers 24 + 11 = 35. Using modulo concepts 35 mod 26 = 9 results the letter “J”. Do the same algorithm for all characters in data test until we have table as below,

Password	K	E	Y	K	E
Data test	H	E	L	L	O
Cipher text	R	I	J	V	S

4. Note that for the letter “L”, we can get different cipher text letter. So, it will take a longer time for one who does not know cipher password to decrypt the data test. In contrast, it is easy to do a brute force attack to Caesar cipher. Because of that, Vigenère cipher has influenced many cipher algorithms throughout history and is still in use today.

Decrypting Vigenère cipher use the similar algorithm to the encryption one with steps listed below,

1. “RIJVS” is consisted by 5 characters, then we span the word “KEY” until there are 5 characters.

K	E	Y	K	E

2. Again, each of letter of “RIJVS” has a corresponding letter password.

K	E	Y	K	E
R	I	J	V	S

3. The decryption process uses this formula,

Original Text = Cipher Text - Key

The letter R (17) would be decrypted using key K (10), then the cipher text letter corresponds to alphabet 17 – 10 = 7 (H). If deciphering process results a number that is lower than 0, use modulo concepts e.g., J (9) with key Y (24) deciphers 9 - 24 = -15. Using modulo concepts -15 mod 26 = 11 results the letter “L”. Do the same algorithm for all characters in data test until we have table as below,

Password	K	E	Y	K	E
Cipher text	R	I	J	V	S
Data test	H	E	L	L	O

C. Graph and Tree

1. Graph

Graph is a way to represent discrete objects and relationships among them. It consists of two main properties: vertex and edge. Formal definition of graph,

G = (V, E)

where V is nonempty set of vertices and E is set of edges (can be empty).

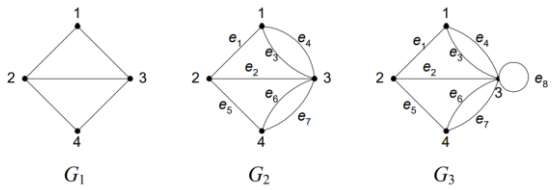

Fig. 1 Examples of graph
(Source: References [1])

Based on type of edges, graph is classified into 3 main categories,

a. Simple graph
Graph which does not contain multiple edges (edges that have the same initial and final vertex with initial vertex ≠ final vertex) and loop (edge that start and finish on the same vertex).

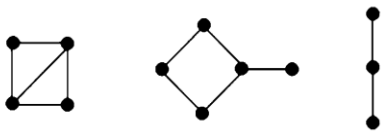

Fig. 2 Examples of Simple Graph
(Source: References [1])

b. Multi-graph
Graph which has multiple edges but not loop.

Fig. 3 Examples of Multi-Graph
(Source: References [1])

c. Pseudo-graph
Graph which has multiple edges and/or loop(s).

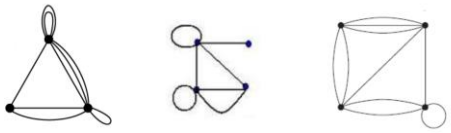

Fig. 4 Examples of Pseudo-Graph
(Source: References [1])

Based on edge directions, graph is classified into 2 categories,

1. Undirected Graph
Graph that has undirected edge.

Fig. 5 Examples of Undirected Graph
(Source: References [1])

2. Directed Graph
Graph that has directed edge.

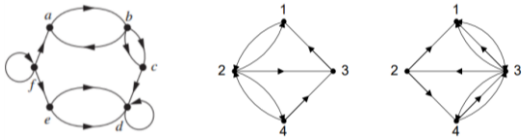

Fig. 6 Examples of Directed Graph
(Source: References [1])

In Graph, there are properties called, path and circuit. Path (or trail) is series of vertices and edges from initial vertex (v_i) to final vertex (v_f), such that $e_1 = (v_i, v_1)$, $e_2 = (v_1, v_2)$, …, $e_n = (v_{n-1}, v_f)$ are edges of the graph. For instance, In Fig 1-G1, 1-2-4-3 is an example of path in the graph, but 1-4-3-2 is not a path since there is no edge (1,4). Circuit is a path, the initial and final vertex are the same, e.g. 1-2-4-3 is not a circuit (since vertex 1 ≠ vertex 3), 1-2-4-3-1 is a circuit (because 1-2-4-3-1 is a path and initial vertex = final vertex), and 1-4-3-2-1 is not a circuit (because 1-4-3-2-1 is not a path).

2. Tree

Tree is a graph that does not have a circuit. We are using a specific type of tree in this paper: rooted tree. Rooted tree is a tree with one vertex will be treated as a root and all edges of the graph will be given direction such that there is no edge directed to the root.

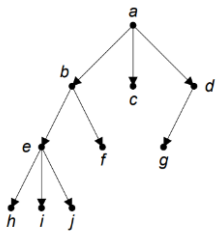

Fig. 7 Tree with a is a root
(Source: References [1])

There are some terminologies used to differentiate properties of a tree. We are not going to cover all terminologies since we do not use all of them in this paper. This paper only includes,

a. Child/Children and Parent
In Fig. 7, vertex b, c, and d are children of vertex a. Conversely, vertex a is parent of vertex b, c, and d.

b. Level
In Fig. 7, vertex b, c, and d are on level 1, Vertex a is on level 0, and vertex h, i, j are on level 3.

c. Ordered Tree
Ordered tree is a tree with differentiation of each child (first child, second child, etc). In Fig. 7, we cannot determine whether the tree is an ordered tree since there are no additional information (or because there is no information added, we can assume that the tree is not ordered).

d. N-ary tree
If a tree is a N-ary tree, it means that the tree has minimum one vertex that has N child such that N is a maximum number of children of all vertex. In Fig.7, the tree is a 3-ary tree.

In the next section, we only use ordered 2-ary tree (read: binary tree). Binary tree is a tree that has two child properties: left child and right child (remember binary tree only have maximum 2 children).

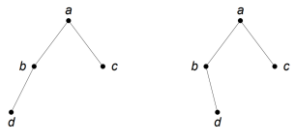

Fig. 8 Two different binary trees
(Source: References [1])

Look at Fig.8, the tree on the left is different with the tree on the right because on the left tree, d is the left child of b, while on the right tree, d is the right child of b. In ordered tree, position matters. However, if it is given that the trees is not ordered, the left and the right tree are the same.

D. Huffman Coding Algorithm

Huffman coding algorithm is a way to compress text. It fits best when we are not using all characters in the same string of data test. For example, string "aaaaa" originally has 5 × 8-bit data. With Huffman code, "aaaaa" can only take 5 bits!

The Huffman algorithm is explained below,

1. Suppose we are going to compress the word "DISCRETEMATHAREHARD". We first then list all of character that are in string.

Letter	D	I	S	C	R	E	T	M	A	H
Frequency	2	1	1	1	3	3	2	1	3	2

2. We start by making trees for letters with least frequency: I, S, C, and M.

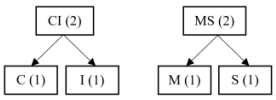

Fig. 9.a Huffman Coding v.1
(Source: Personal Library)

3. Any combination is acceptable. We now have 2 trees that have frequency = 2. Then we pair all letters dan have frequency = 2.

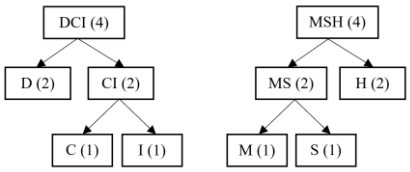

Fig. 9.b Huffman Coding v.1
(Source: Personal Library)

4. T is left alone with frequency = 2. Since there is no other letter or tree that has frequency = 2, we pair T with one letter that have frequency = 3. Any combination is acceptable.

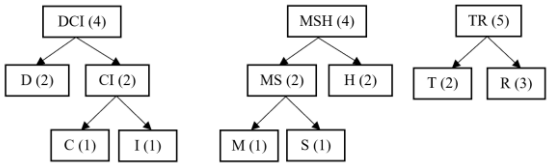

Fig. 9.c Huffman Coding v.1
(Source: Personal Library)

5. Again, any combination is acceptable. Next step is not combining DCI and MSH because we have letters which have lower frequency than them: E, A.

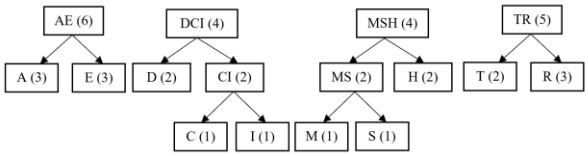

Fig. 9.d Huffman Coding v.1
(Source: Personal Library)

6. Since all letters have been on one of the trees, we now combine tree that has least frequency, these are MSH and DCI.

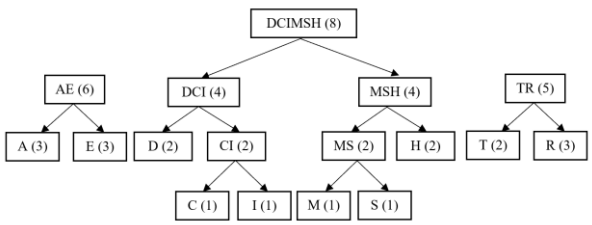

Fig. 9.e Huffman Coding v.1
(Source: Personal Library)

7. We combine the next trees with least frequency, these are AE and TR.

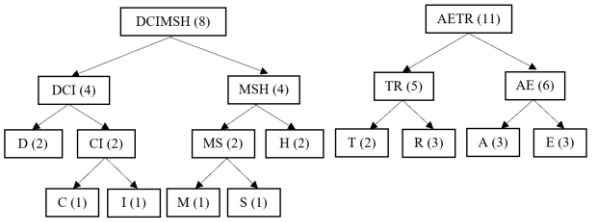

Fig. 9.f Huffman Coding v.1
(Source: Personal Library)

8. Finally, the last two trees.

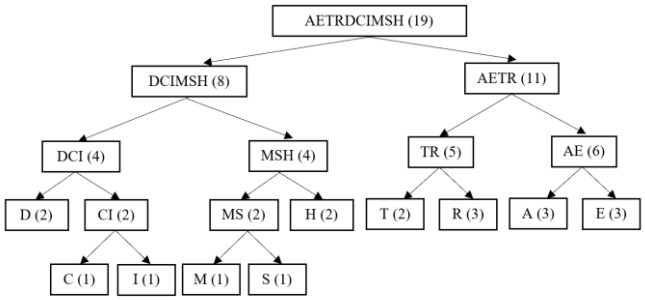

Fig. 9.g Huffman Coding v.1
(Source: Personal Library)

9. We then order the children of the tree using Huffman code.

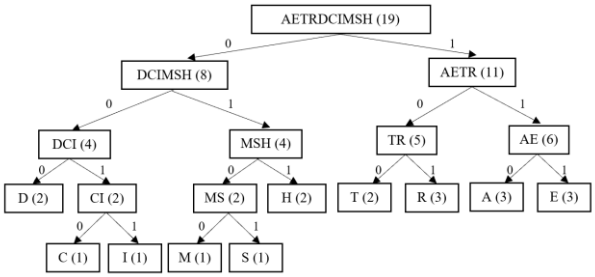

Fig. 9.h Huffman Coding v.1
(Source: Personal Library)

10. Note that different tree might results different Huffman code. It is fine. Convert all letters to Huffman code in a table.

Letter	Huffman Code	Length Code
D	000	3
I	0011	4
S	0101	4
C	0010	4
R	101	3
E	111	3
T	100	3
M	0100	4
A	110	3
H	011	3

11. To compress text, we convert string “DISCRETEMATHAREHARD” into Huffman code, which results “00000110101001010111110011101001101000111101 01111011110101000”. It is a 61 long string which is great! The string “DISCRETEMATHAREHARD” consists of 19 characters, hence without compression the string would take 19 bytes = 152 bits of memory while Huffman code form of the string would only take 61 bits of memory. We save around 59.9% memory space.
12. To note, although your Huffman code might be different than I stated above, the length of all Huffman code for the string “DISCRETEMATHAREHARD” should be still 61.

III. Algorithm And Data Structure

A. Big Picture

In this paper, we are going to encrypt a message using Vigenère cipher, but the key we are going to use is not only from user’s password letter, but also Huffman coding results. Huffman coding may vary, but we can make a unification to it. In the next section, we are going to discuss how we could make a structured compressed text using Huffman Coding Algorithm.

B. Huffman Coding Unification

A string of message can have multiple Huffman Code. Differences between those coding could lead different interpretations. So, how can we make a Huffman Coding constant?

We could use a priority list for all characters possible. This list is not for public. Only a master (could be a human or stored in a secret place) would know this priority list. We can use Machine Learning/Artificial Intelligence to randomize elements of the list in a certain period regularly.

Suppose we have a priority list as listed below,

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m
26	27	28	29	30	31	32	33	34	35	36	37	38
n	o	p	q	r	s	t	u	v	w	x	y	z
39	40	41	42	43	44	45	46	47	48	49	50	51
SPACE		!	?	,	.	-	_	&	(	)	[	]
52		53	54	55	56	57	58	59	60	61	62	63

This list is an example of priority list. We use the table above in this paper’s experiment to give a big picture how custom Vigenère cipher works. In the real use, this list is not open to public. You can customize the table: use another element order or add symbol, e.g., “\n” or “#”. Lower corresponding number means that the character is more prioritized.

Back to the previous example of Huffman Coding in earlier section, we are going to redo our work to compress “DISCRETEMATHAREHARD”, but now we are going to use our priority list to determine which character should we work on first.

The steps are,

1. We have table frequency of “DISCRETEMATHAREHARD”

Letter	D	I	S	C	R	E	T	M	A	H
Frequency	2	1	1	1	3	3	2	1	3	2

2. We start by making trees for letters with least frequency: I, S, C, and M. Since C and I have more priority than M and S, we should pair CI and MS. We can’t pair C and S or any other combination because we are wanting this Huffman Coding constant.

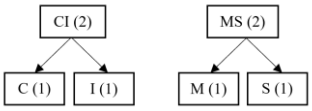

Fig. 10.a Huffman Coding v.2
(Source: Personal Library)

Note that node that has more priority list should be a left child of its parent node. Notice that C is the left child of CI and S is the right child of MS.

3. There is no longer character with frequency = 1. Hence, we now combine character with frequency = 2. There are five characters possible: CI, MS, D, H, T. The Huffman coding tree would be,

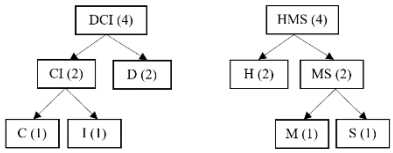

Fig. 10.b Huffman Coding v.2
(Source: Personal Library)

Notice how CI is the left child of DCI and MS is the right child of HMS. For composite characters, the value if its priority is the highest priority (means lower corresponding number on priority list) of characters that formed it, i.e., CI has the same priority value as C.

4. T is left behind. Now, we combine T with character with frequency = 3 that has the highest priority, i.e., A.

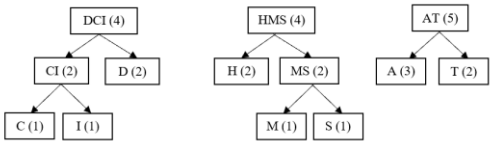

Fig. 10.c Huffman Coding v.2
(Source: Personal Library)

Notice how A is the left child of AT since A is more prioritized than T even though T has a lower frequency.

5. Combine the rest of characters which have frequency = 3, i.e., E and R.

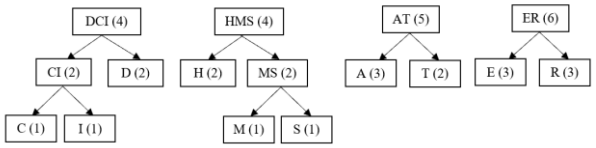

Fig. 10.d Huffman Coding v.2
(Source: Personal Library)

6. Combine DCI and HMS (characters with frequency = 4).

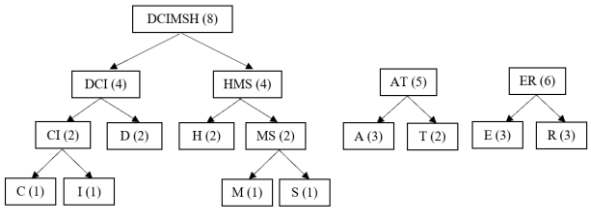

Fig. 10.e Huffman Coding v.2
(Source: Personal Library)

7. We combine the next trees with least frequency, these are AE and TR.

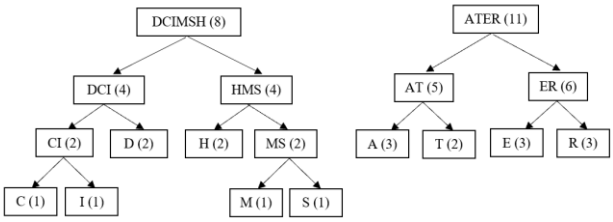

Fig. 10.f Huffman Coding v.2
(Source: Personal Library)

8. We finally combine the last two trees.

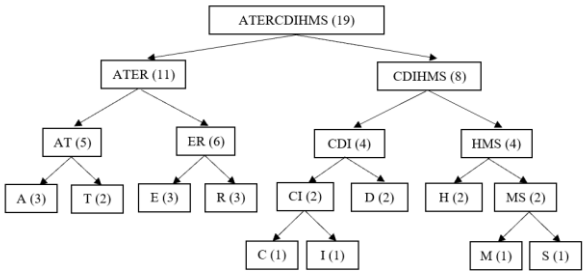

Fig. 10.g Huffman Coding v.2
(Source: Personal Library)

9. We then order the children of the tree using Huffman code.

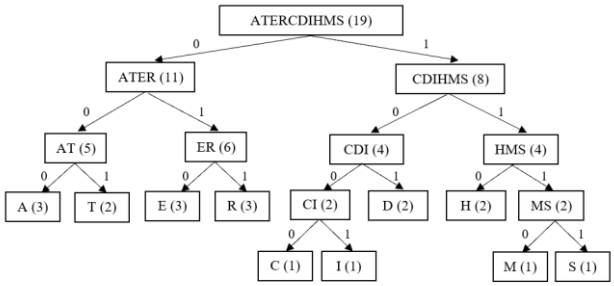

Fig. 10.h Huffman Coding v.2
(Source: Personal Library)

10. Based on the tree above, we get Huffman code listed in table below.

Letter	Huffman Code	Length Code
D	101	3
I	1001	4
S	1111	4
C	1000	4
R	011	3
E	010	3
T	001	3
M	1110	4
A	000	3
H	110	3

11. Notice that the table gives us different coding compared to the previous one. If you notice that all characters have the same code length, it is just a coincidence, different coding might different. However, the thing we can conclude based on two Huffman coding that we have done previously is Huffman coding could give us the same TOTAL length of compressed text.

C. Data Structure

1. Tree (Class)

```
class Node:
    def __init__(self, listchar,
frequency, prio, left, right):
        self.frequency = frequency
        self.listchar = listchar
        self.prio = prio
        self.left = left
        self.right = right
        self.code = ''
```

2. Priority List (Ordered List)

```
priorityList = [
    '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', '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', ' ', '!', '?',
    ',', '.', '-', '_', '&',
    '(', ')', '[', ']'
]
```

3. Dictionary (Map)

```
listHuffmanCodes = dict()
listEncodedHuffmanCodes = dict()
#Example
{'A': '000', 'T': '001', 'E': '010',
```

```
'R': '011', 'C': '1000', 'I':
'1001', 'D': '101', 'H': '110', 'M':
'1110', 'S': '1111'}
```

D. Huffman Coding Algorithm

The subsequent program is written in Python.

1. countCharacter
   Type: Function returns dictionary
   Process: count frequency of characters in string input

```
def countCharacter(stringinput):
  chartable = dict()

  for charinput in stringinput:
    if chartable.get(charinput) == None:
      chartable[charinput] = 1
    else:
      chartable[charinput] += 1

  return chartable
```

2. huffmanTreeCode
   Type: Function returns dictionary
   Process: list all Huffman Code based on Huffman coding algorithm

```
def huffmanTreeCode(
           nodeTree, currentCode=''):
  newCode =
       currentCode + str(nodeTree.code)
  if (nodeTree.left):
    huffmanTreeCode(
              nodeTree.left, newCode)
  if (nodeTree.right):
    huffmanTreeCode(
              nodeTree.right, newCode)

  if not nodeTree.left and
                  not nodeTree.right:
    listHuffmanCodes[nodeTree.listchar] =
                                 newCode

  return listHuffmanCodes
```

3. huffmanEncoding
   Type: Procedure
   Process: processing Huffman Coding Algorithm

```
def huffmanEncoding(stringinput):
    listcharfrequency =
             countCharacter(stringinput)
    listchar = listcharfrequency.keys()
    listfrequency =
              listcharfrequency.values()

    huffmanTreeNode = []

    for char in listchar:
       huffmanTreeNode.append(Node(char,
       listcharfrequency.get(char),
       priorityList[char], None, None))

    while (len(huffmanTreeNode) > 1):
        huffmanTreeNode =
             sorted(huffmanTreeNode,
             key=lambda x: (x.frequency,
             x.prio))

        left = huffmanTreeNode[0]
        right = huffmanTreeNode[1]

        if left.prio > right.prio:
            left, right = right, left

        left.code = 0
        right.code = 1

        if left.prio < right.prio:
            newNodeTree =
                  Node(left.listchar +
                       right.listchar,
                       left.frequency +
                       right.frequency,
                       left.prio, left,
                       right)
        else:
            newNodeTree =
                  Node(left.listchar +
                       right.listchar,
                       left.frequency +
                       right.frequency,
                       right.prio, left,
                       right)

        huffmanTreeNode.remove(left)
        huffmanTreeNode.remove(right)
        huffmanTreeNode.append(
                              newNodeTree)
```

E. Encryption Algorithm

This custom Vigenère cipher is a little bit different than the original one. In this version, we cannot have the same cipher results from different letters and it only works if number of characters of string input is less or equal to number of characters in priority list used. This happens because every character in Huffman code is unique, then we should have a unique cipher to every character in string input

The subsequent program is written in Python.

1. Encryption

```
def vigenereEncode(stringinput,
password):
    encodedText = ''
    for i in range (0, len(stringinput)):
        lengthHuffman =
len(listHuffmanCodes[stringinput[i]])
        passwordchar =
password[i%len(password)]
        key =
priorityList.index(stringinput[i]) +
priorityList.index(passwordchar) +
lengthHuffman
        encodedChar =
priorityList[key%len(priorityList)]
        keyoffset =
len(listHuffmanCodes[stringinput[i]])
        while encodedChar in
listEncodedHuffmanCodes.keys():
```

```
            encodedChar =
priorityList[priorityList.index(encodedCh
ar)+1]
            keyoffset = keyoffset + 1

listEncodedHuffmanCodes[encodedChar] =
keyoffset
        encodedText += encodedChar
    return encodedText
```

2. Decryption

```
def vigenereDecode(decodedText,
password):
    originalText = ''
    for i in range (0, len(decodedText)):
        passwordchar =
password[i%len(password)]
        key =
priorityList.index(decodedText[i]) -
priorityList.index(passwordchar) -
listEncodedHuffmanCodes[decodedText[i]]
        originalText +=
priorityList[key%len(priorityList)]
    return originalText
```

F. Main Program

The subsequent program is written in Python.

```
listHuffmanCodes = dict()
listEncodedHuffmanCodes = dict()
priorityList = [
    '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', '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', ' ', '!', '?',
    ',', '.', '-', '_', '&',
    '(', ')', '[', ']'
]
passkey = input("Enter your password: ")
stringinput = input("Enter your message: ")
huffmanEncoding(stringinput)
print("Psst.. It's a Secret: ", end='')
secrettext     =     vigenereEncode(stringinput,
passkey)
print(secrettext)
print("Check Decode ", end='')
print(vigenereDecode(secrettext, passkey))
```

IV. EXPERIMENT

1. Test Case 1
   DISCRETEMATHAREHARD with password: HELLO

```
Enter your password: HELLO
Enter your message: DISCRETEMATHAREHARD
Psst.. It's a Secret: NQhRiOaSbTdPUfVWHgX
Check Decode DISCRETEMATHAREHARD
```

Fig. 11 TC1 Results
(Source: Personal Library)

2. Test Case 2
   DISCRETEMATHAREHARD with password: PASSWORD

```
Enter your password: PASSWORD
Enter your message: MATHDISCRETEAREHARD
Psst.. It's a Secret: fDocdanJjHpZbiYNSUe
Check Decode MATHDISCRETEAREHARD
```

Fig. 12 TC2 Results
(Source: Personal Library)

3. Test Case 3
   DISCRETEMATHAREHARD with password: DISCRETEMATHAREHARD

```
Enter your password: DISCRETEMATHAREHARD
Enter your message: DISCRETEMATHAREHARD
Psst.. It's a Secret: JUoIlLpMcDqREmNSFnK
Check Decode DISCRETEMATHAREHARD
```

Fig. 13 TC3 Results
(Source: Personal Library)

4. Test Case 4
   "DID YOU KNOW THERE IS A TUNNEL UNDER OCEAN BOULEVARD " with password: "LANA DEL REY"

```
Enter your password: LANA DEL REY
Enter your message: DID YOU KNOW THERE IS A TUNNEL UNDER OCEAN BOULEVARD
Psst.. It's a Secret: SNU?TVcBEhW CYaHJK_ZMILOjbdQ&X(kFePtDfgR-i)lGpmnoqrs
Check Decode DID YOU KNOW THERE IS A TUNNEL UNDER OCEAN BOULEVARD
```

Fig. 14 TC4 Results
(Source: Personal Library)

5. Test Case 5
   "GoOd PeRsOn" with password: "Ingrid"

```
Enter your password: Ingrid
Enter your message: GoOd PeRsOn
Psst.. It's a Secret: STxMZwp(P)N
Check Decode GoOd PeRsOn
```

Fig. 15 TC5 Results
(Source: Personal Library)

V. CONCLUSION

. Vigenère cipher, an algorithm that is simple and easy to understand, is a fundamental and a way to dig deeper in cryptography. This suitable and intended for beginners like me, who are willing to learn more about cyber security world and its high-tech properties. We can custom Vigenère cipher with Huffman coding algorithm which makes that algorithm more exciting. We can also learn other fundamental computer science concepts, i.e., binary tree and its properties, data

structure and its efficiency on algorithms by using this method. This method can help us understand a big picture of what cryptography is and an open gate to more advanced algorithms that waits for us ahead.

VI. Acknowledgment

Firstly, I want to thank me and God for making this paper finished. I cannot express what I am feeling now because I cannot believe that I wrote this paper fully in English. I want to thank Mr. Rinaldi Munir and Ms. Fariska Ruskanda for the fundamental knowledge that you have taught me earlier this semester. That knowledge made me do an exploration to something I could not ever imagined before. Hope this paper helps beginner student like me, who are interested in computer science and want to dig deeper into that, in order to comprehend basic yet fundamental knowledge in informatics and cryptography.

Appendix

1. Code documentation on GitHub
   https://github.com/natthankrish/Custom-Vigenere-Huffman

References

[1] Munir, Rinaldi, Pohon (Bagian 1 &2), (2022), accessed on https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf and https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf on December 1st - 11th 2022.

[2] Rodriguez-Clark, Dan (2017), Vigenère Cipher, Crypto Corner

[3] Yağmur Çiğdem Aktaş (2021), Huffman Encoding & Python Implementation, accessed on Medium https://towardsdatascience.com/huffman-encoding-python-implementation-8448c3654328 on December 1st - 11th 2022.

[4] Martin, Keith M. (2012). Everyday Cryptography. Oxford University Press. p. 142. ISBN 978-0-19-162588-6.

[5] Gupta, Sanjali, Nikhil Shanker Mathur, Priyank Chauhan, (2016), A New Approach to Encryption using Huffman Coding, International Journal of Progressive Sciences and Technologies (IJPSAT)

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Desember 2022

Antonio Natthan Krishna - 13521162