Cryptography: Graph Theory Implementation in Text Encryption and Decryption

Albert Ghazaly - 13522150
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522150@itb.ac.id

***Abstract*—Graph Theory is implemented in many companies and industries as it is applicative for real world cases. As one of the implementations mentioned above, graph theory has such great role which even it can be applied to cryptography. In other terms, graph theory can keep information safe by improving the security of the information. However, how graph theory can be implemented in cryptography is what will be shown deeply.**

***Keywords*—Complexity Algorithm, Data Security, Graph Theory, Number Theory**

I. INTRODUCTION

Technology has been rapidly growing, leading to an era where everyone can share their information to others separated in less than a minute. The information that is mentioned is actually data. The data itself is defined, referred to as Merriam-Webster dictionary, as factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation, while data security, as defined by IBM, is the practice of protecting digital information from unauthorized access, corruption, or theft throughout its entire lifecycle. The main focus of the paper is applying graph theory to encrypt and decrypt data. The data that will be encrypted is a message containing a alphabet, numbers, and symbols. The purpose of the paper is to explain how graph theory also can protect privacy data by implementing it to cryptography. Its impact will be giving protections to the data as if anyone cannot receive, modify, or even corrupt the data without permission.

In this paper, the algorithm will be not only explained, but also be shown by the source code and the result. The purpose of the paper is also implementing the concept besides inventing it. This will help anyone to understand how the algorithm actually works and how to use it. The concept of the algorithm is to change the data or message as fast as possible because the message can be as long as a paragraph. The speed of the process also impacts on how well the algorithm is. The speed also determines how the algorithm can actually be implemented in real world technology as the information's size cannot be restricted. The time complexity of the algorithm will be discussed in the paper as it is important for understanding and analyzing the algorithm.

The graph theory role in the encryption and decryption algorithms is for the representation of each character of the data. All of the numbers, alphabets, and symbols will be represented as a whole graph. The encryption algorithm will take the data , manipulate it as same as the rule given in the algorithm, and return graphs as the results. In other side, the decryption algorithm will take input as graph, manipulate the graph as the rule given in the algorithm, and return the original message. The main concept of the graph that is used mostly is Hamilton graph. It will be the base of the data structure in both encryption algorithm and decryption algorithm. The concept holds a different role in encryption and decryption algorithms.

The encryption algorithm can be described as (1) and the decryption algorithm process can be described as (2).

$$data(alphabets, symbols, numbers) \rightarrow package(graph, key) \quad (1)$$

$$graph + key \rightarrow data(alphabets, symbols, numbers) \quad (2)$$

II. THEORETICAL BASIS

1. Graph

1.1 Graph Definition and Types

In the terminology, referred to Collins Dictionary, a graph is a mathematical diagram which shows the relationship between two or more sets of numbers or measurements.

The objects inside a graph, called vertices or nodes, can be connected to each other by an edge. Using that functionality, graph can describe the relationship between sets of numbers or measurements. Let G as a graph, V as a set of vertices, and E a set of edge, the graph can be defined :

$$G = (V, E)$$

For a set of n objects, the set V contains n vertices:

$$V = \{v1, \mathrm{v2}, \dots, \mathrm{vn}\}$$

Where a set of m relations, the set E contains m edges:

$$E = \{e1, \mathrm{e2}, \dots, \mathrm{em}\}$$

Graphs can be distinguished by orientation as directed graph and undirected graph. A Directed graph is a graph that the edge directs to a certain vertice. Which means if a vertice has an edge directed to another vertice, the relation of the two object is one-directional. For example, the predator and the prey has a directional relationship which can be described as a directed graph. The undirected graph is a graph that the edges don't have any direction. In the other words, the relationship between the objects is both-directional.

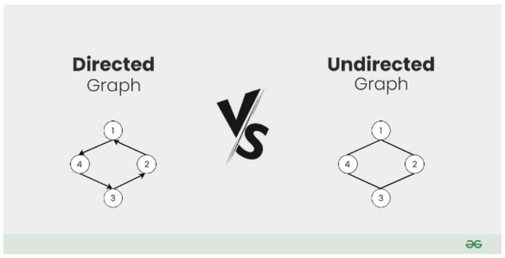


Fig 2.1 Directed and undirected graph
Source: https://www.geeksforgeeks.org/what-is-the-difference-between-an-undirected-and-a-directed-graph/

In the other hand, a graph can also be distinguished into a simple graph or unsimple-graph. A simple graph is a graph that does not contain multi-edge and loop whilst unsimple-graphs does. A multi-edge is defined as having two edges to a same node and loop can be defined as having edge to the node itself. The unsimple-graph can also be multi-graph that contains multi-edge and pseudo-graph which contains loop.

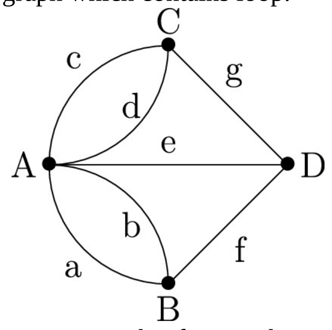


Fig 2.2 Example of unsimple-graph
Source: https://link.springer.com/chapter/10.1007/978-981-19-0957-3_1

1.2 Graph Terminology

1.2.1 Adjacence

A vertex is called adjacent to another vertex if it is directly connected to it.

1.2.2 Incidence

A vertex is called incident with an edge if the edge connects to the vertex.

1.2.3 Isolated Vertex

A vertex is called isolated if it has no edge connected to it.

1.2.4 Null Graph

A Null graph is a graph that doesn’t contains any edges.

1.2.5 Degree

The degree of a vertex is the sum of how many edges are connected to the vertex or how many edges are incident with the vertex. The sum of all degrees in a graph will always be even, with a quantity of two times the sum of all edges. Which then can be derived that in a Graph G, the count of every vertex with odd degrees will be even.

1.2.6 Path A finite or infinite sequence of edges which joins a sequence of vertices which are all distinct. A path with length n is a traversal sequence of vertex and edge

$\{v0, e1, v1, e2, \ldots, vn{-}1, en, vn\}$

1.2.7 Circuit

A circuit is a path that ends on the same vertex that it started.

1.2.8 Connection

A graph is termed "connected" if there's a path between every pair of its vertices. If it lacks such connectivity, it's considered a "disconnected" graph. In directed graphs, vertices u and v are "strongly connected" if there's a directed path from u to v and vice versa. If not strongly connected but connected in the undirected version of the graph, they are termed "weakly connected."

1.2.9 Subgraph

A graph $G1 = \{V1, E1\}$ is a subgraph of another graph $G = \{V, E\}$ if $V1$ is a subset of V and $E1$ is a subset of E. The complement of a subgraph $G1$, denoted as $G2 = \{V2, E2\}$, satisfies $V2 \subseteq V$ and $E2 = E - E1$. The components of a graph refer to the maximum connected subgraphs within it.

1.2.10 Spanning Subgraph

A spanning subgraph contains all the vertices of the original graph. Specifically, graph $G1 = \{V1, E1\}$ is a spanning subgraph of $G = \{V, E\}$ if $V1$ equals V and $E1$ is a subset of E.

1.2.11 Cut-Set

A cut-set in a connected graph G is a set of edges. If these edges are removed, the graph becomes disconnected, resulting in two components.

1.2.12 Weighted Graph

A weighted graph is a graph that has a weight in every edges.

1.3 Special Graphs

1.3.1 Complete Graph (Kn)

This type of graph ensures that every vertex connects to every other vertex within the graph. In a complete graph with n vertices, the total edges sum up to $\Sigma|E| = n(n - 1)/2$.

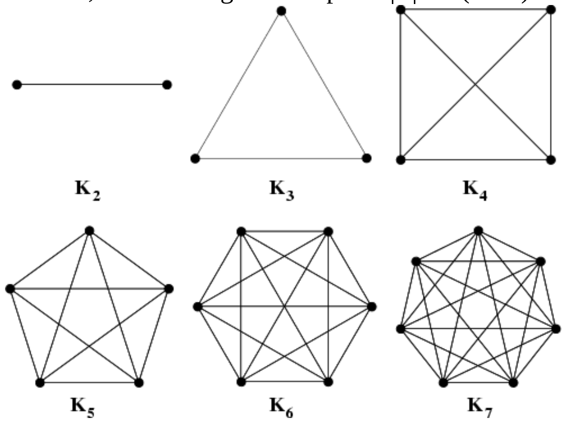


Fig 2.3 Complete graph
Source: https://archive.lib.msu.edu/crcmath/math/math/c/c477.htm

1.3.2 Circle Graph (Cn)

This simple graph assigns a degree of 2 to each vertex which means that all vertex’s degree is two.

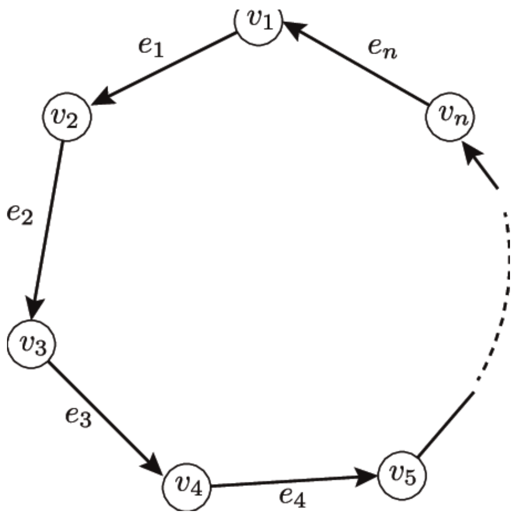


Fig 2.4 Circle graph
Source: https://www.researchgate.net/figure/A-cycle-graph-C-n-with-n-vertices-and-edges_fig1_363128491

1.3.3 Regular Graph (Rr)
In a regular graph, all vertices possess the same degree. A regular graph with degree r ensures that every vertex has a degree of r. The total number of edges in a regular graph is $\Sigma|E| = nr/2$, where n is the number of vertices.

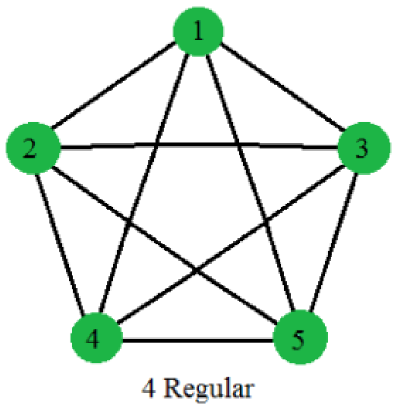


Fig 2.4 Regular graph
Source: https://www.geeksforgeeks.org/regular-graph-in-graph-theory/

1.3.4 Bipartite Graph ($G(V1, V2)$)
This graph comprises two vertex subsets, $V1$ and $V2$, where each edge connects a vertex from $V1$ to a vertex in $V2$.

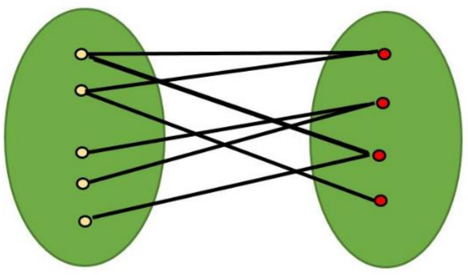


Fig 2.5 Bipartite graph
Source: https://www.geeksforgeeks.org/bipartite-graph/

1.3.5 Hamiltonian Graph
Hamiltonian graph is a graph that contains a closed path (cycle) visiting each vertex exactly once, starting and ending at the same vertex. his closed path is also called a Hamiltonian cycle.

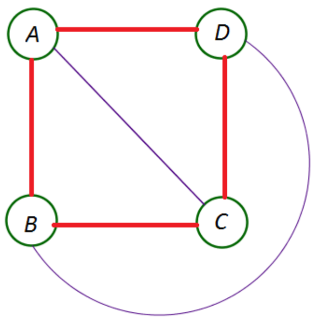


Fig 2.5 Hamiltonian graph
Source: https://mathspace.co/textbooks/syllabuses/Syllabus-1030/topics/Topic-20297/subtopics/Subtopic-266708/

1.3.6 Semi-Hamiltonian Graphs
A semi-Hamiltonian path visits each vertex once, starting and ending at different vertices. Semi-Hamiltonian graphs are also known as traceable graphs.

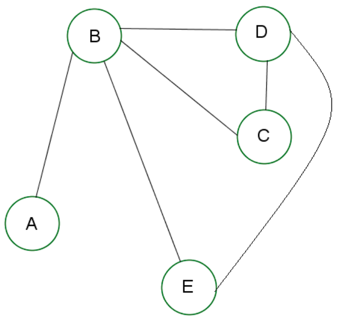


Fig 2.6 Semi-Hamiltonian graph
Source: https://mathspace.co/textbooks/syllabuses/Syllabus-1030/topics/Topic-20297/subtopics/Subtopic-266708/

2. Number Theory
Number theory delves into the realm of mathematics dedicated to the exploration of integers and integer-based functions. Integers, by definition, encompass whole numbers without fractions, zero, comprising the set of positive natural numbers or negative integers. For integers a and b, a divides b if there exists an integer c such that $b = ac$, denoted as $a \mid b$, where c belongs to the set of integers, and $a \neq 0$.

2.1 Euclidean Theorem/Division
When an integer m is divided by a positive integer n, the division results in a quotient q and a remainder r, satisfying the equation $m = nq + r$, where $0 \leq r < n$.

2.2 Greatest Common Divisor (GCD)
The GCD of two integers, a and b, is the largest integer d such that $d \mid a$ and $d \mid b$, represented as $GCD(a, b) = d$.

2.3 Euclidean Algorithm
This algorithm, starting with non-negative integers m and n (where $m \geq n$), continuously divides r values until reaching rn

= 0. It determines the GCD using the sequence of remainders.

$r0 = m$, $r1 = n$. With continuous division:

$r0 = r1q1 + r2$, $0 \leq r2 < r1$

$r1 = r2q2 + r3$, $0 \leq r3 < r2$

…

$rn{-}2 = rn{-}1qn{-}1 + rn$, $0 \leq rn < rn{-}1$

$rn{-}1 = rnqn + 0$

as for every $m = nq + r$, $0 \leq r < n$, $GCD(m, n) = GCD(n, r)$, thus:

$GCD(m, n) = GCD(r0 , r1) = GCD(r1 , r2) = \cdots = GCD(rn{-}1 , rn) = GCD(rn, 0) = rn$

2.4 Modulo Arithmetic

For every $m = nq + r$ equation, modulo operations $m \ mod \ n = r$, with $0 \leq r < m$. The modulo operation returns the remainder when a number is divided by the modulus (n).

2.5 Congruence

Given an integer $n > 1$ as the modulus, two integers a and b are congruent modulo n if their remainders are equal when divided by n. The equation $a \equiv b$ (mod n) defines this relationship, illustrating that a and b differ by a multiple of n. The equation can be written as a = kn + b.

2.6 Hash Function

A function used to transform data of varying sizes into fixed-size outputs called hashes. Hash functions aim to efficiently convert variable-length keys into uniform, fixed-length values, ensuring minimal collisions (instances where different data produce the same hash). The hash equation:

$$h(K) = K \ mod \ m$$

maps input keys K into a memory space m, returning their respective hash values.

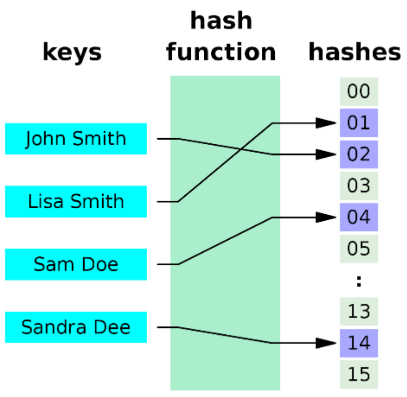


Fig 2.7 Hash function

Source: https://en.wikipedia.org/wiki/Perfect_hash_function

2.7 Cryptography

Cryptography, derived from Greek roots, translates to "secret writing." It involves the methods and exploration of safeguarding communication between two or more parties from external entities. Essentially, cryptography involves designing and scrutinizing protocols aimed at preventing unauthorized access by third parties to the communication between two involved parties. The original text, known as plaintext or data, undergoes encryption—a process where an algorithm transforms it into ciphertext. This ciphertext holds no meaning for an unauthorized entity lacking access to the decryption algorithm, which reverses the process back to plaintext. One of the earliest forms of substitution ciphers, the Caesar Cipher, shifted plaintext letters by a fixed number to generate the corresponding ciphertext.

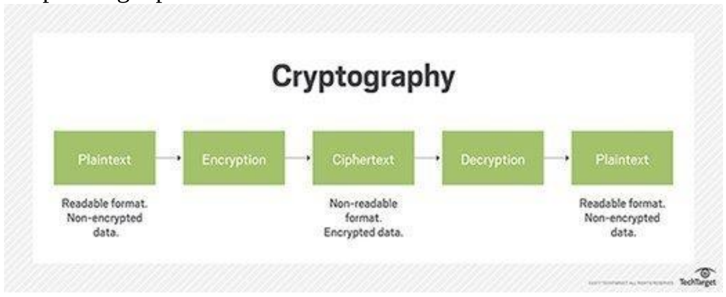

Cryptography
Plaintext
Encryption
Ciphertext
Decryption
Plaintext
Readable format. Non-encrypted data.
Non-readable format. Encrypted data.
Readable format. Non-encrypted data.
TechTarget

Fig 2.8 Cryptography visualization

Source: https://www.techtarget.com/searchsecurity/definition/cryptography

2.8 RSA Algorithm

The RSA Algorithm, coined after its developers Ronald Rivest, Adi Shamir, and Leonard Adleman in 1976, represents an asymmetric cryptography technique. In this method, the encryption and decryption keys are distinct from one another. The encryption key, referred to as the public key, is openly accessible and not kept confidential, similar to the decrypting key or private key, which remains exclusively known to the key owner. The primary process of key generation in the RSA Algorithm involves several steps: Firstly, selecting two undisclosed prime numbers, p and q, and then computing $n = pq$ and $m = (p - 1)(q - 1)$. While n need not be kept secret, m must remain confidential. Subsequently, choosing a number e that is coprime to m, specifically $GCD(e, m) = 1$, and deriving the private key d from the equation $ed \equiv 1$ (mod m). Then, the encryption and decryption can be done using equation in *Fig 2.9*.

Encryption:

$Plaintext$	$M < n$
$Ciphertext$	$C = M^e (mod \ n)$

Decryption:

$Ciphertext$	C
$Plaintext$	$M = C^d (mod \ n)$

Fig 2.9 RSA Encryption and decryption

Source: https://www.geeksforgeeks.org/rsa-algorithm-using-multiple-precision-arithmetic-library/

3. XOR Chiper

XOR Chiper uses XOR operator to the input to a certain target and return the result. In cryptography, the simple XOR cipher is a type of additive cipher, an encryption algorithm that operates according to the principles:

A ⊕ 0 = A,

A ⊕ A = 0,

A ⊕ B = B ⊕ A,

(A ⊕ B) ⊕ C = A ⊕ (B ⊕ C),

(B ⊕ A) ⊕ A = B ⊕ 0 = B,

Where ⊕ denotes XOR operator.

4. Time Complexity

In theoretical computer science, the time complexity is the computational complexity that describes the amount of computer time it takes to run an algorithm. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform. The asymptotic notation, can be used to measure time complexity of algorithm which are Big-O notation, Omega notation, Theta notation.

4.1 Big-O notation (O-notation)

Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm. If f(n) describes the running time of an algorithm, f(n) is O(g(n)) if there exist a positive constant C and n0 such that, $0 \le f(n) \le cg(n)$ for all $n \ge n0$

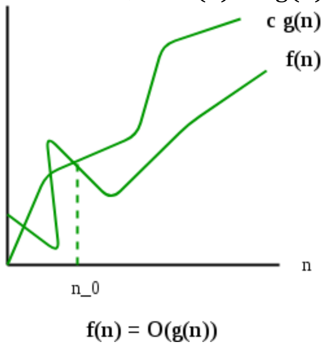


Fig 2.10 Big-O notation visualization
Source: https://www.geeksforgeeks.org/types-of-asymptotic-notations-in-complexity-analysis-of-algorithms/

4.2 Omega notation (Ω-notation)

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm. Let g and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that $c*g(n) \le f(n)$ for all $n \ge n0$

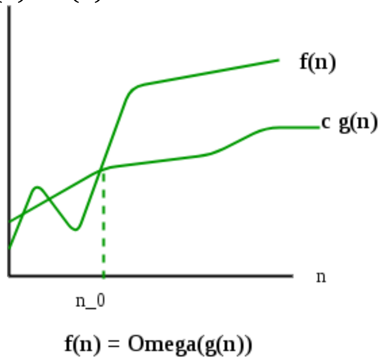


Fig 2.11 Omega notation visualization
Source: https://www.geeksforgeeks.org/types-of-asymptotic-notations-in-complexity-analysis-of-algorithms/

4.3 Theta notation (Θ-notation)

Theta notation represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm. Let g and f be the function from the set of natural numbers to itself. The function f is said to be Θ(g), if there are constants c1, c2 > 0 and a natural number n0 such that $c1* g(n) \le f(n) \le c2 * g(n)$ for all $n \ge n0$

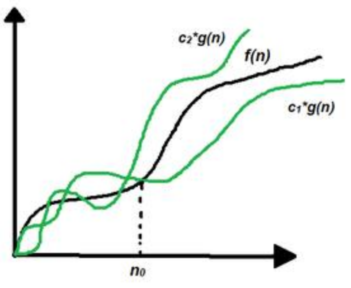


Fig 2.12 Theta notation visualization
Source: https://www.geeksforgeeks.org/types-of-asymptotic-notations-in-complexity-analysis-of-algorithms/

III. Proposed Algorithm

A. Data Structure

Before beginning algorithms, we need to know how we save and load the data either it is plain data or encrypted data. First, we need to have string as an input that we want to encrypt. Second, we need to have a list of graphs that will store the encrypted message which has been turned into graph. The graph, is an Hamiltonian graph and represented as adjacency matrix with the value of the m[i][j] =0 if vertex I and j not connected, m[i][j] = 1, if vertex I and j connected, m[i][j] = X, X>1 if vertex I and j connected and have a weight of the edge that is X. Third, we need to have a key that stores the first or start node of Hamiltonian graph and the last or end node of the Hamiltonian graph and create a list of key that stores the key. To run the algorithm, we also need to have an array of integer with size of seven (7x1) that will be used to turn ascii code (integer) into binary which each of its digits will be stored in the array.

B. Encryption Algorithm

There are steps for the encryption algorithm, let list of graph is G, list of key is K, the input string is S, and the array of integer with size of seven is arr[7]:
1. Take a first character from the S, assume it to be char
2. turn char's ascii code and store it to arr[7]
3. using the XOR algorithm, do XOR chipper of char's ascii in arr[7] to 127 (Binary code; 1111111). The result will reverse each element of the binary in arr[7] from '0' to '1' and from '0' to '1'.
4. turn the reversed binary code in arr[7] into a graph. First, count how many '1' in arr[7] and make it as the number of the graph's vertex. The graph now is a null graph because even though there is a vertex, there is still no edge.
5. start from a random vertex (v0), count how many zero from the first, start from the left (arr[1]), '1' to the right(second)'1' in the arr[7] and add it with 1 .E.g: 1010011, zero count of first '1' will be 2 because (1+1=2).

6. assume the count of the first zero is Z, then store the Z in the graph as the edge and its weight.
7. Redo steps 5 and 6, until the vertex reaches the final Hamiltonian graph's node. Remember when, choosing random vertex, it must follow the Hamiltonian graph rule which means the generated vertex cannot be same as the previous, cannot create loop, and other Hamiltonian graph's rule.
8. The result of the process is a Hamiltonian graph. Then, store the key as a tuple of 2 integer, which are the first node of the Hamiltonian graph and the final node of the Hamiltonian graph.
9. After getting the graph and the key, store both of them each in list of graph (G) and list of key (K).
10. Do the exact from step 1, but for the next character in the S. After, the final character is encrypted then the encryption process is finished. The result is the list of graphs (G) and the list of key (K).

C. Decryption Algorithm

Assume we have the list of graphs (G) as encrypted message and the list of key (K), then the decryption algorithm will be:

1. take one graph, the first, from the list of graphs (G)
2. Reverse the Encryption algorithm from step 4 to step 7 using the key in the list of key (K), by turning graph into the binary code that is stored in a list of integer with size of seven (7x1), assume it is named arr[7].
3. After getting the binary code in arr[7], turn the binary code into a character (binary to ascii code).
4. Do XOR chipper again with 127 (binary code: 1111111) to decrypt the character to its original character.
5. Store the decrypted character to a string or list of characters. Repeat the exact to other graphs in the list of graphs (G) and using the key in the list of keys (K) until all the graphs decrypted.
6. If all graphs in G is decrypted, then the decryption algorithm is complete with the return of list of characters or string containing the original message.

IV. PROGRAM IMPLEMENTATION

A. Main Components

To implement the both algorithms, we need to provide proper data structures to be used in the algorithm. Even though, there are other components such as string and list of integer, the main components used by the algorithms are four.

The main components of the program are:

1. Graph – The part that saves encrypted messages as a graph. Represented as adjacency matrix, in the program it is a dynamic matrix which is needed to give flexibility of the size of the graph.
2. Key – The part that store the first and the last node of the Hamiltonian graph. In the program, it is a tuple of two integer.
3. List of graphs – This part stores the graphs. Because we do not know how many graph are there, then we use linked list as the data structure of list of graph. So, we can append the linked list if we want to add another graph.
4. List of keys – This part stores the keys. Because we do not know how many keys are there, then we use linked list as the data structure of list of keys. So, we can append the linked list if we want to add another key.
5. Message – This part is actually what we call "encrypted message", it contains a pointer to the list of graphs and the length of the original message.

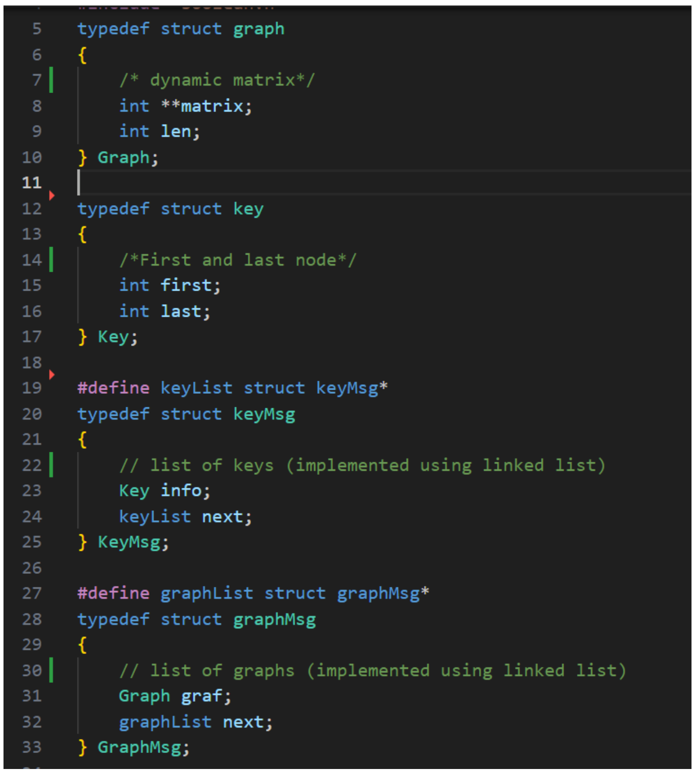
B. Implementation Product

The implementation of the products require the realization of the algorithm and the data structure. Here are the program's structure:

- main.c (this is not necessary)
- encrypt.c
- encrypt.h
- decrypt.c
- decrypt.h
- adt.h
- adt.c

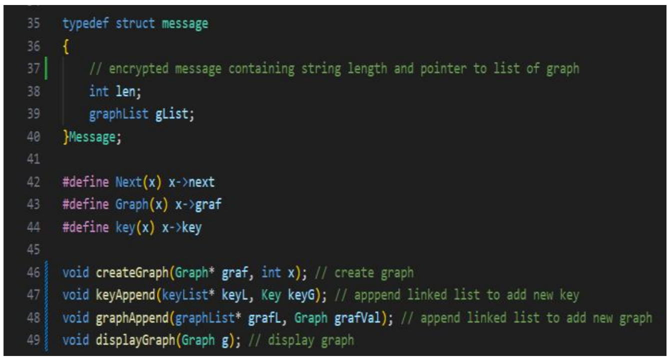
Note that main.c is not necessary, but it is to implement and test the encryption and decryption program.

The implementation of the data structure is in adt.h file:



```c
typedef struct graph
{
    /* dynamic matrix*/
    int **matrix;
    int len;
} Graph;

typedef struct key
{
    /*First and last node*/
    int first;
    int last;
} Key;

#define keyList struct keyMsg*
typedef struct keyMsg
{
    // list of keys (implemented using linked list)
    Key info;
    keyList next;
} KeyMsg;

#define graphList struct graphMsg*
typedef struct graphMsg
{
    // list of graphs (implemented using linked list)
    Graph graf;
    graphList next;
} GraphMsg;
```

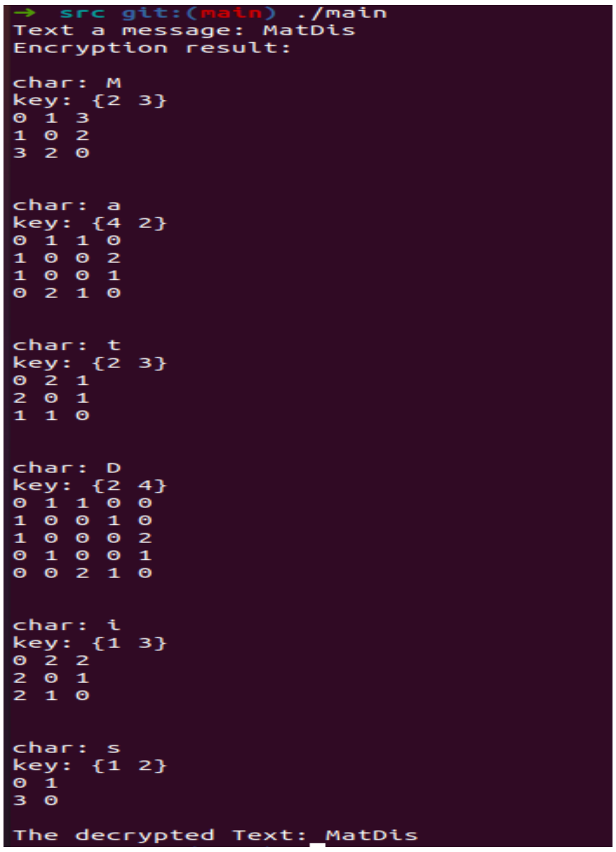
Fig 4.1 Datatype implementation 1
Source: Personal



```c
typedef struct message
{
    // encrypted message containing string length and pointer to list of graph
    int len;
    graphList gList;
}Message;

#define Next(x) x->next
#define Graph(x) x->graf
#define key(x) x->key

void createGraph(Graph* graf, int x); // create graph
void keyAppend(keyList* keyL, Key keyG); // apppend linked list to add new key
void graphAppend(graphList* grafL, Graph grafVal); // append linked list to add new graph
void displayGraph(Graph g); // display graph
```

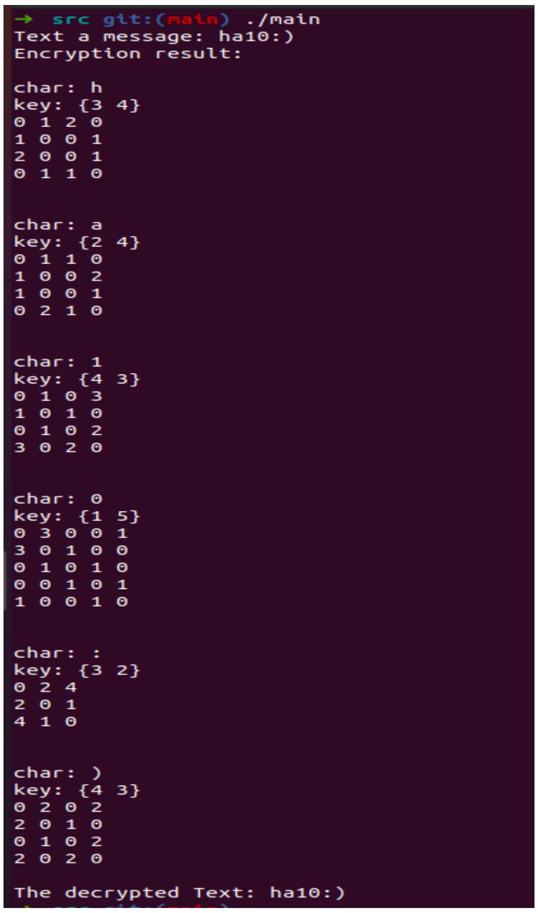
Fig 4.2 Datatype implementation 2
source: Personal

The result of the program can be viewed by running main.c file. For testing, let the input be “MatDis” and “ha10:)”, then the result will be:



```
→ src git:(main) ./main
Text a message: MatDis
Encryption result:

char: M
key: {2 3}
0 1 3
1 0 2
3 2 0


char: a
key: {4 2}
0 1 1 0
1 0 0 2
1 0 0 1
0 2 1 0


char: t
key: {2 3}
0 2 1
2 0 1
1 1 0


char: D
key: {2 4}
0 1 1 0 0
1 0 0 1 0
1 0 0 0 2
0 1 0 0 1
0 0 2 1 0


char: i
key: {1 3}
0 2 2
2 0 1
2 1 0


char: s
key: {1 2}
0 1
3 0

The decrypted Text: MatDis
```

Fig 4.3 Test case 1
Source: Personal



```
→ src git:(main) ./main
Text a message: ha10:)
Encryption result:

char: h
key: {3 4}
0 1 2 0
1 0 0 1
2 0 0 1
0 1 1 0


char: a
key: {2 4}
0 1 1 0
1 0 0 2
1 0 0 1
0 2 1 0


char: 1
key: {4 3}
0 1 0 3
1 0 1 0
0 1 0 2
3 0 2 0


char: 0
key: {1 5}
0 3 0 0 1
3 0 1 0 0
0 1 0 1 0
0 0 1 0 1
1 0 0 1 0


char: :
key: {3 2}
0 2 4
2 0 1
4 1 0


char: )
key: {4 3}
0 2 0 2
2 0 1 0
0 1 0 2
2 0 2 0

The decrypted Text: ha10:)
```

Fig 4.4 Test case 2
Source: Personal

The full implementation of the repository can be checked here https://github.com/albert260302/Graph-Theory-implementation-in-Cryptography. The source code is in src directory and there is a guide about the implementation in readme file, especially about how to use the program.

C. Time Complexity

Let the number of characters of the input as n. Then the worst case of the algorithm is when the total vertex gained from a character is 7 which means that we need to create and manipulate 7x7 adjacency matrix of the graph. To measure the time complexity, let the measurements focuses on how many operation of matrix assignment. Note that because graph is undirected, so the assignment is two times each iteration.

$$M[i][j] = 0 \qquad M[j][i] = 0$$

Assume the worst case that every character inputted turns into 7-vertex graph, we can calculate T(n) of the encryption algorithm:

$$N = 1 \text{ -> } 2*(7) = 28$$
$$N = 2 \text{ -> } 2*(7) + 2*(7) = 28$$
$$\ldots$$
$$N = n \text{ -> } 2*(7) + \ldots + 2*(7) = 2*7*n = 14*n \ (1)$$

Then, using (1) we can conclude that

$T(n) = 14*n$

$T(n) = 14*n <= 15*n, n>=1$

For C = 15 and N0 = 1: O(n)

$T(n) = 14*n$

$T(n) = 14*n <= 13*n, n>=1$

For C=13 and N0 = 1: Ω(n)

Because $\Omega(f(n))$, $O(f(n))$, and $f(n) = n$, then the theta notation is $\Theta(n)$.
Thus, The time complexity result of encryption algorithm is $O(n)$, $\Omega(n)$, and $\Theta(n)$. The time complexity of decryption algorithms will be same as it only reverse the step of the encryption algorithm. It is also because the measurement of this time complexity is how many matrix adjacency assignment is.

V. Review

The Encryption and decryption algorithm shows how the graph theory is useful if we can utilize it. The encryption is not actually only use graph theory but it also combines XOR chipper into it. However, the main focus of the algorithm is the utilization and implementation of graph theory. Both algorithms, encryption and decryption, use Hamiltonian graph as a basis for the algorithm and data structure. The data structure and algorithm also implemented in C language because C is flexible in manipulating and allocating memory. The algorithm can still be improved by maximizing the efficiency of the code, especially in memory usage as the algorithm does require lot of memories. The result of time complexity O(n) cannot be interpreted as a light or efficient algorithm because the algorithm may run fast, but it still requires lot of memory allocation for linked list, dynamic matrix, and array. That is why, we can improve the algorithm not only in time complexity but also in memory cost.

VI. Conclusion

The Algorithm proposed shows how the input as string can be encrypted not only to another string but also to another data structure such as graph. The paper also mentions the implementation of the algorithm by testing it and measuring the time complexity. However, there is still components in the algorithm and data structures which can be improved for the time and memory efficiency.

VII. Acknowledgment

First of all, the Author would like to thank our lecturers in Class 3, Mr. Rinaldi Munir and Mr. Monterico Adrian for giving us numerous lesson. Not to forget, the Author would also like to all the teachers in ITB for striving to share knowledge to us while knowing it is not easy to handle many students at the same time. Lastly, the Author would like to tell that The lesson that you taught was the reason the Author can accomplish this paper.

References

[1] https://mathspace.co/textbooks/syllabuses/Syllabus-1030/topics/Topic-20297/subtopics/Subtopic-266708/, accessed 9 December 2023, 14.00 P.M.
[2] https://medium.com/@logsign/how-does-xor-cipher-work-xor-chipher-encryption-b7ad316209ca, accessed 9 December 2023, 14.10 P.M
[3] https://www.acadpubl.eu/hub/2018-119-13/articles/40.pdf, accessed 9 December 2023, 20.30 P.M.
[4] https://www.techtarget.com/searchsecurity/definition/cryptography, accessed 10 December 2023, 15.00 P.M.
[5] https://www.geeksforgeeks.org/time-complexity-and-space-complexity/, accessed 10 December 2023, 15.20 P.M.

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, 3 Desember 2023

[illegible]

Albert Ghazaly - 13522150