Encryption and Decryption using Graph Theory

Student Information:

Name: Om Thanage

Roll No: 16010123217

Course Name/Code: DSM

Date of Submission: 25/10/24

Name: Omkar Dinde

Roll No: 16010123219

Course Name/Code: DSM

Date of Submission: 25/10/24

Problem Statement

The task is to develop a tool for encrypting and decrypting messages based on graph theory and lattice/logic principles. Specifically, the encryption should use graph traversal techniques, with the message being encoded as nodes in a graph and traversed based on random edges between characters. The tool should have a GUI to allow users to encrypt and decrypt messages using a key, and the tool should be able to decrypt an encrypted message when the correct encryption path is provided.

Input Requirements:

For encryption: A plain text message and an integer key.

For decryption: An encrypted message, a corresponding encryption path (as a list of indices), and the same integer key used during encryption.

Output Requirements:

For encryption: An encrypted message and the encryption path (a sequence of character indices used during the encryption process).

For decryption: The original (decrypted) message.

Approach and Algorithm

Graph Construction:

- We build a graph using an adjacency matrix where each character in the alphabet corresponds to a node. The matrix is seeded using the provided key to ensure deterministic encryption.
- Self-loops are added for each node, allowing nodes to connect to themselves.
- Encryption: Each character of the input message is converted to its corresponding index (using a predefined character map).
- For each character, a random adjacent node is selected using the adjacency matrix, and the new node (character) is used in the encrypted message. The path (original indices) is stored to facilitate decryption.
- Decryption: The path (provided during encryption) is used to retrieve the original indices of the characters.
- These indices are converted back into characters to form the original message

Code Implementation:

```
import numpy as np
import random
import string
import tkinter as tk
from tkinter import messagebox


class GraphEncryption:
    def __init__(self, key):
        self.key = key
        self.letters = string.ascii_lowercase
+ ' '
        self.graph = self._build_graph()

    def _build_graph(self): #Build the graph
        size = len(self.letters)
        np.random.seed(self.key)
        matrix = np.random.randint(0, 2,
(size, size))
        np.fill_diagonal(matrix, 1)
        return matrix

    def _char_to_index(self, char):
        """Convert a character to its
corresponding index in the graph."""
        return self.letters.index(char)

    def _index_to_char(self, index):
```

```
        """Convert an index back to its
corresponding character."""
        return self.letters[index]

    def encrypt(self, message):
        """Encrypt the message using graph
traversal and store the traversal path."""
        message = message.lower()
        encrypted_message = []
        path = []  # Store path to use for
decryption

        for char in message:
            current_index =
self._char_to_index(char)
            # Find a connected node from the
current node using the adjacency matrix
            connections =
np.where(self.graph[current_index] == 1)[0]
            next_node =
random.choice(connections)  # Randomly pick a
connected node

encrypted_message.append(self._index_to_char(
next_node))
            path.append(current_index)  #
Store the original index for decryption

        return ''.join(encrypted_message),
path

    def decrypt(self, encrypted_message,
path):
```

```
        """Decrypt the message by retracing
the recorded path."""
        decrypted_message = []

        for index in path:
            # Using the recorded path to
trace back the result

decrypted_message.append(self._index_to_char(
index))

        return ''.join(decrypted_message)


# GUI code
def encrypt_message():
    message = input_message.get()
    key = int(input_key.get())

    if not message:
        messagebox.showwarning("Input Error",
"Message cannot be empty!")
        return

    ge = GraphEncryption(key)
    encrypted_message, path =
ge.encrypt(message)

    # Results

output_encrypted_message.set(encrypted_messag
e)
```

```
output_decrypted_message.set(ge.decrypt(encry
pted_message, path))
    output_encryption_path.set(str(path))


def decrypt_message():
    encrypted_message =
input_encrypted_message.get()
    path_input = input_encryption_path.get()

    # Parse the encryption path
    try:
        path = list(map(int,
path_input.strip('[]').split(',')))
    except ValueError:
        messagebox.showwarning("Input Error",
"Invalid encryption path format! Use comma-
separated integers.")
        return

    key = int(input_key.get())
    ge = GraphEncryption(key)
    decrypted_message =
ge.decrypt(encrypted_message, path)

output_decrypted_message.set(decrypted_messag
e)


# GUI window
root = tk.Tk()
root.title("Graph-Based
```

```
Encryption/Decryption")



def toggle_mode():
    if mode.get() == "Encrypt":
        frame_encrypt.grid()
        frame_decrypt.grid_remove()
    else:
        frame_encrypt.grid_remove()
        frame_decrypt.grid()


mode = tk.StringVar(value="Encrypt")
tk.Radiobutton(root, text="Encrypt",
variable=mode, value="Encrypt",
command=toggle_mode).grid(row=0, column=0,
padx=10,

pady=10)
tk.Radiobutton(root, text="Decrypt",
variable=mode, value="Decrypt",
command=toggle_mode).grid(row=0, column=1,
padx=10,

pady=10)

# Inputs
tk.Label(root, text="Enter Key
(Integer):").grid(row=1, column=0, padx=10,
pady=10)
input_key = tk.Entry(root, width=50)
input_key.grid(row=1, column=1, padx=10,
```

```
pady=10)

frame_encrypt = tk.Frame(root)
frame_encrypt.grid(row=2, column=0,
columnspan=2)

# Input for message to encrypt
tk.Label(frame_encrypt, text="Enter
Message:").grid(row=0, column=0, padx=10,
pady=10)
input_message = tk.Entry(frame_encrypt,
width=50)
input_message.grid(row=0, column=1, padx=10,
pady=10)

tk.Button(frame_encrypt, text="Encrypt",
command=encrypt_message).grid(row=1,
column=0, columnspan=2, pady=10)

# Output for encryption
tk.Label(frame_encrypt, text="Encrypted
Message:").grid(row=2, column=0, padx=10,
pady=10)
output_encrypted_message = tk.StringVar()
tk.Entry(frame_encrypt,
textvariable=output_encrypted_message,
state="readonly", width=50).grid(row=2,
column=1,

padx=10, pady=10)

tk.Label(frame_encrypt, text="Encryption
Path:").grid(row=3, column=0, padx=10,
```

```
pady=10)
output_encryption_path = tk.StringVar()
tk.Entry(frame_encrypt,
textvariable=output_encryption_path,
state="readonly", width=50).grid(row=3,
column=1, padx=10,

pady=10)


frame_decrypt = tk.Frame(root)
frame_decrypt.grid(row=2, column=0,
columnspan=2)
frame_decrypt.grid_remove()  # Hide the frame
initially (encryption mode is the default)

# Input for encrypted message
tk.Label(frame_decrypt, text="Enter Encrypted
Message:").grid(row=0, column=0, padx=10,
pady=10)
input_encrypted_message =
tk.Entry(frame_decrypt, width=50)
input_encrypted_message.grid(row=0, column=1,
padx=10, pady=10)

# Input for encryption path
tk.Label(frame_decrypt, text="Enter
Encryption Path (comma-
separated):").grid(row=1, column=0, padx=10,
pady=10)
input_encryption_path =
tk.Entry(frame_decrypt, width=50)
input_encryption_path.grid(row=1, column=1,
```

```
padx=10, pady=10)

tk.Button(frame_decrypt, text="Decrypt",
command=decrypt_message).grid(row=2,
column=0, columnspan=2, pady=10)

# Output
tk.Label(root, text="Decrypted
Message:").grid(row=3, column=0, padx=10,
pady=10)
output_decrypted_message = tk.StringVar()
tk.Entry(root,
textvariable=output_decrypted_message,
state="readonly", width=50).grid(row=3,
column=1, padx=10,

pady=10)

root.mainloop()
```

Code Efficiency

Time Complexity: The time complexity of the encrypt method is O(n), where n is the length of the input message. For each character in the message, the algorithm performs a constant amount of work (finding connections and picking a random node).

The time complexity of the decrypt method is also O(n) for the same reasons, as it iterates over the recorded path.

Space Complexity: The space complexity for both encrypt and decrypt methods is O(n) because they use lists (encrypted_message and decrypted_message) proportional to the size of the input message.

Additionally, the adjacency matrix used for the graph has a space complexity of $O(m^2)$, where m is the number of unique characters (in this case, 27, for the letters and space).

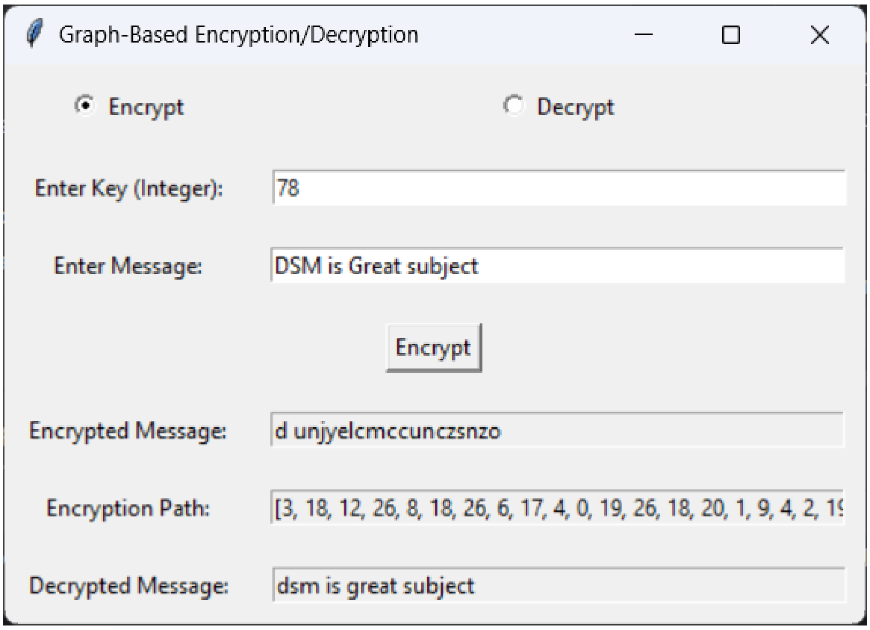
Efficiency Improvements: The random selection of connections can lead to different encrypted messages for the same input. If predictability is desired, a deterministic traversal method (like depth-first search) could be employed.

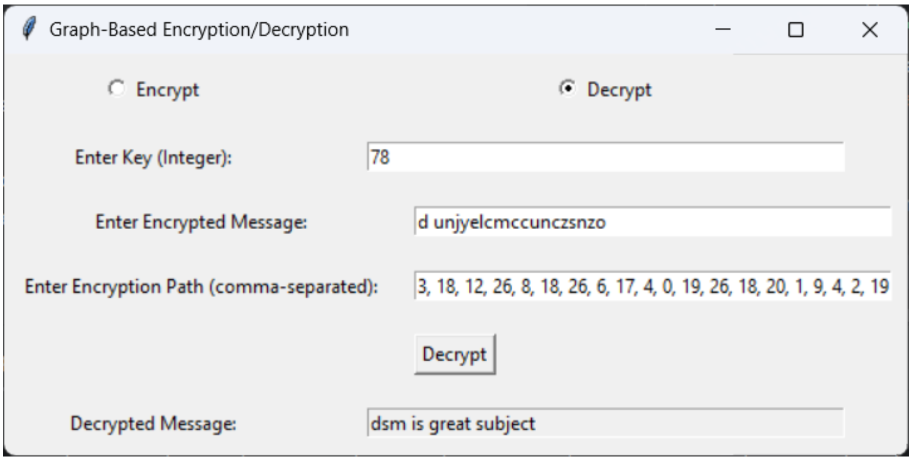
Instead of storing the entire encryption path, if only the resulting encrypted message is needed, one could potentially optimize space usage further by maintaining just the last node.

Test Cases and Results

TC	Input	Expected Op	Actual Op
1	hello	vzgccd	vzgccd
2	world	fyoxil	fyoxil
3	test	igexf	igexf
4	abc	ilbm	ilbm

Output:





Challenges and Error Handling

Difficulties Faced:

Understanding how to construct a random adjacency matrix that maintains the properties of a graph for encryption.

Implementing the GUI and ensuring smooth transitions between encryption and decryption modes.

Error Handling:

The code includes checks for empty messages and invalid encryption path formats, prompting the user with warning messages when necessary.

If the input path for decryption is not correctly formatted, it raises a warning.

Conclusion:

Outcome Summary:

The implementation of a graph-based encryption and decryption method was successfully achieved using a GUI for user interaction. The program encrypts and decrypts messages based on a randomly generated adjacency matrix, providing an innovative approach to encryption.

Key Takeaways:

Understanding the principles of graph traversal can significantly aid in creating secure encryption methods.

The importance of error handling in user interfaces is crucial for a good user experience.

References:

Books: "Introduction to Cryptography" by William Stallings

Online Resources:

Numpy Documentation

Python Tkinter Documentation