Enigma

STATUS: Not Cracked

To test enigma machine implementation run `main.py`

```
//Python 3.7 or above
pip install networkx
python3 main.py
```

The Approach To The Problem

To solve the problem we decided to implement the advanced Turing Bombe. The Advanced Turing Bombe introduces a new method of representing the permutation σ, which describes the plugboard settings of the Enigma machine. It uses a 26x26 grid, where both the rows and columns represent the letters of the alphabet.

Enigma Machine Implementation

To make this possible we needed an enigma machine simulator. The simulator, implemented in Python, consists of several modules that replicate the functionality of the historical Enigma cipher machine:

1. `enigma.py`: This is the core module, integrating all other components to simulate the Enigma machine's operations.
2. `rotor.py`: It defines the `Rotor` class, responsible for the rotor mechanism in the Enigma machine. Each rotor includes a specific wiring permutation, a position, and a unique name. It can rotate and encode letters in both forward and reverse directions.
3. `plugboard.py`: Simulates the plugboard of the Enigma, enabling the swapping of letters based on predefined pairings.
4. `reflector.py`: Represents the reflector part of the Enigma, which ensures that a letter is never encoded as itself.

Test Encoding/Decoding

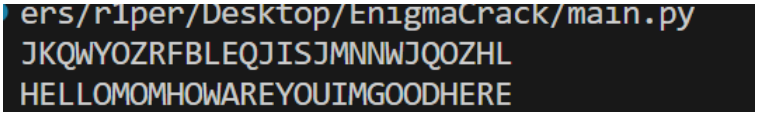
To test the machine we need to setup the rotor configuration, the reflector and the plugboard.

```
from EnigmaMachine.enigma import Enigma, enigma_machine
from EnigmaMachine.plugboard import Plugboard
from EnigmaMachine.reflector import Reflector
from EnigmaMachine.rotor import Rotor


rotor1 = Rotor("EKMFLGDQVZNTOWYHXUSPAIBRCJ",23,"I")
rotor2 = Rotor("AJDKSIRUXBLHWTMCQGZNPYFVOE",3,"II")
rotor3 = Rotor("BDFHJLCPRTXVZNYEIWGAKMUSQO",4,"III")
reflector = Reflector("YRUHQSLDPXNGOKMIEBFZCWVJAT")
#fast,medium,slow
rotors = [rotor1,rotor2,rotor3]
plugboard = Plugboard([])
enigma_machine = Enigma(rotors,reflector,plugboard) #global istance of engima machine

message = "HELLOMOMHOWAREYOUIMGOODHERE" #plain text
encoded_message = enigma_machine.encode(message)
print(encoded_message)

rotor1_copy = Rotor("EKMFLGDQVZNTOWYHXUSPAIBRCJ", 23, "I")
rotor2_copy = Rotor("AJDKSIRUXBLHWTMCQGZNPYFVOE", 3, "II")
rotor3_copy = Rotor("BDFHJLCPRTXVZNYEIWGAKMUSQO", 4, "III")
enigma_copy = Enigma([rotor1_copy,rotor2_copy,rotor3_copy],reflector,plugboard)
decoded_message=enigma_copy.encode(encoded_message)
print(decoded_message)
```



These modules collectively recreate the encoding and decoding processes of the original Enigma machine. With this in our hands, we can now focus on the graph analysis of the crib to find the correct rotor position and decipher the text.

Crib Analysis And 26x26 Matrix

Crib Graph

For the crib analysis and the construction of the matrix graph we used networkx python library. For the crib we create a graph where the vertices are the letters of the alphabet and the edges are connections between the plaintext letters and the encrypted letters. We also add the position in the crib for each letter

```
def create_crib_graph(crib, encrypted_text):
    G = nx.DiGraph()
    n = len(crib)
    encrypted_text = encrypted_text[:n]  # Considering only the first n characters
    for i, (p, c) in enumerate(zip(crib, encrypted_text)):
        if G.has_edge(p, c):
            G[p][c]['weight'].append(i)
        else:
            G.add_edge(p, c, weight=[i])
    return G, crib, encrypted_text
```

- The function iterates over pairs of characters from the crib and the corresponding encrypted text.
- For each pair of characters `(p, c)` (where `p` is a character from the crib and `c` is the corresponding character from the encrypted text):
  - If an edge from `p` to `c` already exists in the graph `G`, the index `i` (position of the character in the crib and encrypted text) is appended to the `weight` attribute of the edge. This `weight` attribute is actually used as a list to keep track of all positions where the crib character `p` maps to the encrypted character `c`.
  - If such an edge does not exist, a new edge from `p` to `c` is added to the graph with the `weight` attribute initialized to a list containing the index `i`

26x26 Matrix

The matrix is used to describe the permutation of the plugboard. To create this matrix we again use the networkx library

```
def create_gamma_k_graph(crib_analysis_results):
    G = nx.DiGraph()
    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

    # Add edges for symmetry
    for L1 in alphabet:
        for L2 in alphabet:
            if L1 != L2:
                G.add_edge((L1, L2), (L2, L1))

    # Add edges based on crib analysis
    for L1, L2 in crib_analysis_results:
        print(L2)
        L3_transformed = enigma_machine.encode_letter(L2)
        G.add_edge((L1, L2), (L2, L3_transformed))

    return G
```

1. **Adding Symmetry Edges**:
    - The function iterates over all pairs of different letters `(L1, L2)` from the alphabet.
    - For each pair, it adds an edge in the graph from `(L1, L2)` to `(L2, L1)`. This step establishes a symmetric relationship between every pair of distinct letters, reflecting the reciprocal nature of the Enigma's encoding process.
2. **Incorporating Crib Analysis Results**:
    - The function then processes the `crib_analysis_results`, which is a list of letter pairs derived from the crib analysis.
    - For each pair `(L1, L2)` in the `crib_analysis_results`:
        - The letter `L2` is transformed using the Enigma machine's encoding mechanism, which is presumably implemented in the `enigma_machine.encode_letter` method. This yields `L3_transformed`, the encoded version of `L2`.
        - The function then adds an edge in the graph from the pair `(L1, L2)` to the pair `(L2, L3_transformed)`. This edge represents the transformation that occurs in the Enigma machine for these specific letter pairings

Final Solution

We tried to implement the algorithm described in the code theory book but we didn't have enough time to finish and test it. We successfully created the crib graph and the 26x26 matrix representing the plugboard. To complete this assignment we need to implement the routine to find the correct position K based on the analysis of the 26x26 matrix.

Each vertex on the matrix graph has the value 0 or 1. To get the correct K position we must make these assertions:

- Each row has exactly one 1 and 25 times a 0.
- Each column has exactly one 1 and 25 times a 0.
- Vertices that are connected have the same number.

Once we find that, we find the correct K and we are able to decipher the entire text. We didn't complete it due to time constraints