Laboratory work 3

Author: Dávid Kromka
Personal key: 11

Task 1

Functions:

1. **polybius_encode(message, square)**:
    - **Input: message** (string) - The message to be encoded, **square** (list) - The Polybius square.
    - **Output:** Encoded message (string).
    - **Description:** Encodes the input **message** using the Polybius square cipher. Each character in the message is replaced by its 2-digit coordinates in the square. Characters are converted to uppercase before encoding.

Encoding Process:

1. For each character **char** in the input **message**:

    a. Convert **char** to uppercase.

    b. Find the coordinates (row index and column index) of the character in the Polybius square.

    c. Add 1 to both the row and column indices to make it 1-based index.

    d. Append the 2-digit coordinates to the **encoded_message** list.

2. Join the elements of the **encoded_message** list with spaces to form the encoded message and return it.

Usage:

1. Two sample messages (**message_a** and **message_b**) are provided in the script.
2. The script encodes these messages using the given Polybius square.
3. The encoded messages are printed.

Encoded Messages:

1. **Encoded Message a):**
    - Original Message: "ENCRYPT ME 2 DAY"
    - Encoded Message: "15 32 13 36 51 34 42 31 15 55 14 11 51 "
2. **Encoded Message b):**
    - Original Message: "Kromka"
    - Encoded Message: "25 36 33 31 25 11 "

```
Encoded message a): 15 32 13 36 51 34 42 31 15 55 14 11 51
Encoded message b): 25 36 33 31 25 11
```

Task 2

1. Initialize values a1, b1, c1, a2, b2, and c2 with binary values:
    - a1 = 1011 (in binary), which is 11 in decimal.
    - b1 = 0110 (in binary), which is 6 in decimal.
    - c1 = 0100 (in binary), which is 4 in decimal.
    - a2 = 0101 (in binary), which is 5 in decimal.
    - b2 = 1110 (in binary), which is 14 in decimal.
    - c2 = 1101 (in binary), which is 13 in decimal.
2. Perform XOR operations on a1, b1, and c1:
    - result = a1 ^ b1 ^ c1 ^ a1 ^ b1
        - This XOR operation combines a1, b1, and c1 multiple times.
        - In binary, it computes 1011 ^ 0110 ^ 0100 ^ 1011 ^ 0110, resulting in 1111.
        - Converting 1111 to binary representation: binary_result = '1111'.
3. Perform XOR operations on a2, b2, and c2:
    - result2 = a2 ^ b2 ^ c2 ^ a2 ^ b2
        - This XOR operation combines a2, b2, and c2 multiple times.
        - In binary, it computes 0101 ^ 1110 ^ 1101 ^ 0101 ^ 1110, resulting in 0001.
        - Converting 0001 to binary representation: binary_result2 = '0001'.
4. Ensure that the binary representation has 4 bits (pad with leading zeros if necessary):
    - binary_result and binary_result2 already have 4 bits, so no additional padding is required.
5. Print the results:
    - The algorithm prints the binary representations of the results, along with labels indicating which operation was performed on them.
        - For the first result: "1. Result of a ^ b ^ c ^ a ^ b: 1111"
        - For the second result: "2. Result of a ^ b ^ c ^ a ^ b: 0001"

Output:

```
1. Result of a ^ b ^ c ^ a ^ b: 0100
2. Result of a ^ b ^ c ^ a ^ b: 1101
```

Task 3

Functions:

1. **calculate_entropy(num_states)**:
    - **Input: num_states** (integer) - The number of states in the system.
    - **Output:** Entropy (float) in bits.
    - **Description:** Calculates the entropy of a system with the given number of states assuming an equiprobable distribution.

Entropy Calculation:

1. Calculate the probability of each state:

   probability=1/num_states.

2. Apply the entropy formula for each state and sum the results to get the total entropy.

Usage:

1. The script calculates entropy for two systems:
    - **System X with 8 states:**
    - **System X with 128 states:**

Result:

```
Entropy for System X with 8 states: 3.00 bits
Entropy for System X with 128 states: 7.00 bits
```