Homework 2

Eli Bullock-Papa

February 17, 2025

Problem 1

Write a function which takes as input two integers a and n, with $n \geq 0$ and returns the value of $\gcd(a, n)$ using the Euclidean Algorithm.
It is not required that you validate the input to this function; that is, you may assume that both input parameters will, in fact, be integers, and that n will be positive.

Solution:
Implementation in Python:

```
def euclidean_gcd(a: int, n: int) -> int:
    if n == 0:
        return a
    return euclidean_gcd(n, a % n)
```

This passed all tests I threw at it, such as:

```
gcd(48, 18) = 6
gcd(54, 24) = 6
gcd(7, 13) = 1
gcd(28, 0) = 28
```

Problem 2

Write a function which takes as input two integers a and n, with $n \geq 0$ and uses the Extended Euclidean Algorithm to return the multiplicative inverse of a mod n if such an inverse exists. As in problem 1, your code does not need to validate that your inputs are positive integers, but you *do* need to check that $\gcd(a, n) = 1$ and return an error if a and n are not relatively prime.

Solution:
My implementation in Python:

```python
def extended_euclidean(a: int, n: int) -> int:
    # Lists to store the quotients and remainders
    quotients = []
    remainders = [n, a]
    # k_values: These will eventually yield the modular inverse
    k_values = [0, 1]

    # Run the Euclidean algorithm
    while remainders[-1] != 0:
        q = remainders[-2] // remainders[-1]
        r = remainders[-2] % remainders[-1]
        quotients.append(q)
        remainders.append(r)

    # The gcd is the second-to-last remainder
    gcd = remainders[-2]
    if gcd != 1:
        raise ValueError(
            "Multiplicative inverse does not exist because gcd(a,n) != 1"
        )

    # Compute the sequence of k-values
    # We already have k_0 and k_1; now compute k_2, k_3, ... up to k_{len(remainders)-2}
    for i in range(2, len(remainders) - 1):
        # Using the recurrence: k_i = k_{i-2} - (quotient_{i-2} * k_{i-1})
        next_k = k_values[i - 2] - quotients[i - 2] * k_values[i - 1]
        k_values.append(next_k)

    # The modular inverse is the last computed k_value modulo n
    return k_values[-1] % n
```

Problem 3

The following was encrypted by the Vigenère method. Decrypt it. Carefully document your work—in general, more details are better than fewer details. In particular, document how you determine the key length.

XKJUROWMLLPXWZNPIMBVBQJCNOWXPCCHHVVFVSLLFVXHAZITYXOHULX
QOJAXELXZXMYJAQFSTSRULHHUCDSKBXKNJQIDALLPQSLLUHIAQFPBPC
IDSVCIHWHWEWTHBTXRLJNRSNCIHUVFFUXVOUKJLJSWMAQFVJWJSDYLJ
OGJXDBOXAJULTUCPZMPLIWMLUBZXVOODYBAFDSKXGQFADSHXNXEHSAR
UOJAQFPFKNDHSAAFVULLUWTAQFRUPWJRSZXGPFUTJQIYNRXNYNTWMHC

The key for this encryption is a sequence of letters which form a recognizable pattern, even though they don't form a word.

Solution:
First, I implemented the Friedman test using the IOC to determine the likely key length:

```python
def friedman_test(ciphertext: str, max_key_length: int) -> list[tuple[int, float]]:
    results = []

    # Try different key lengths
    for key_length in range(1, min(max_key_length + 1, len(clean_text))):
        # Split text into key_length cosets
        cosets = [""] * key_length
        for i, char in enumerate(clean_text):
            cosets[i % key_length] += char

        # Calculate IoC for each coset
        coset_iocs = [IoC.calc_ioc(coset) for coset in cosets]
        avg_ioc = sum(coset_iocs) / len(coset_iocs)

        results.append((key_length, avg_ioc))

    # Sort by IoC in descending order
    return sorted(results, key=lambda x: x[1], reverse=True)
```

Running this test on the ciphertext produced the following results:

```
Possible key lengths (sorted by average IoC):
Length 15: Average IoC = 0.075
Length 10: Average IoC = 0.074
Length 5: Average IoC = 0.074
Length 8: Average IoC = 0.043
Length 2: Average IoC = 0.043
```

The three highest scoring key lengths are 15, 10, and 5, all with very similar IoC values (around 0.074-0.075). Since these lengths are multiples of 5, and 10 and 15 would likely score highly if they were simply repetitions of a length-5 key, I will proceed with the assumption that the key length is 5.

Splitting the text into 5 columns and analyzing the frequency distribution for each column:

```
Col 1: F(18.2%), U(10.9%), O(9.1%), I(9.1%), P(7.3%)
Col 2: H(14.5%), W(12.7%), D(10.9%), Q(7.3%), V(7.3%)
Col 3: S(16.4%), J(14.5%), X(12.7%), M(10.9%), T(9.1%)
Col 4: L(18.2%), A(18.2%), V(9.1%), H(9.1%), P(7.3%)
Col 5: X(12.7%), L(10.9%), C(10.9%), Q(10.9%), N(9.1%)
```

After analyzing the column frequencies, I performed a chi-square analysis to determine the most likely key. The chi-square statistic measures how well the observed letter frequencies match expected English frequencies:

$$\chi^2 = \Sigma(Observed - Expected)^2/Expected$$

where:

- Observed is the count of each letter in the text
- Expected is N * (standard English frequency / 100) for text length N

For each column, I:

1. Tried all 26 possible shifts (A-Z)
2. For each shift, calculated the chi-square score comparing the decrypted text's letter frequencies to standard English frequencies
3. Kept the shifts that produced the lowest chi-square scores (closest to English)

Here's the implementation of the chi-square analysis:

```python
def chi_square_for_text(text: str) -> float:
    N = len(text)
    counts = {}
    for c in text:
        counts[c] = counts.get(c, 0) + 1

    chi_sq = 0.0
    for letter, expected_freq in ENGLISH_FREQ.items():
        # Expected count for this letter in the text
        expected_count = N * (expected_freq / 100)
        observed_count = counts.get(letter, 0)
        chi_sq += ((observed_count - expected_count) ** 2) /
           expected_count
```

```
    return chi_sq

def candidate_shifts_for_column(column: str) -> list[tuple[int, float]]:
    candidates = []
    for shift in range(26):
        # Try decrypting with this shift
        decrypted = decrypt_with_shift(column, shift)
        score = chi_square_for_text(decrypted)
        candidates.append((shift, score))
    candidates.sort(key=lambda x: x[1])
    return candidates
```

Combining the best shifts for each column produced these candidate keys:

```
Key: BDFHJ, Combined Chi-square: 143.74
Key: BDFTJ, Combined Chi-square: 234.98
Key: BDQHJ, Combined Chi-square: 243.16
```

The key BDFHJ produced the best overall chi-square score by far, suggesting it is the most likely key. The decrypted text is the beginning of the Declaration of Independence:

WHENINTHECOURSEOFHUMANEVENTSITBECOMESNECESSARYFORO
NEPEOPLETODISSOLVETHEPOLITICALBANDSWHICHHAVECONNEC
TEDTHEMWITHANOTHERANDTOASSUMEAMONGTHEPOWERSOFTHEEA
RTHTHESEPARATEANDEQUALSTATIONTOWHICHTHELAWSOFNATUR
EANDOFNATURESGODENTITLETHEMADECENTRESPECTTOTHEOPIN
IONSOFMANKINDREQUIRESTHAT

The key BDFHJ follows a simple pattern: it consists of every other letter in the alphabet starting with B.

Problem 4

You are an anthropologist studying on location an isolated language that has only the letters K, O, and Z. Previous studies have indicated that the index of coincidence for this language is about 0.37.
A village merchant offers to sell you three rare manuscripts, and you can read their titles:

"OOK OOK ZOOK"
"OK KOZ KOZZ"
"ZOKO ZOKO OOK"

Your trusty guide warns you that one of the manuscripts is an illiterate and unreadable forgery. Knowing only the titles of these manuscripts, identify with explanation the one most likely to be forged.

Solution:
To identify the likely forgery, I calculated the Index of Coincidence (IoC) for each manuscript title. Since we know the expected IoC for this language is approximately 0.37, the manuscript with the IoC that deviates most from this value is likely the forgery.

Here's the Python implementation used to calculate the IoC:

```python
def calc_ioc(text: str) -> float:
    # Count frequencies
    N = len(text)
    freqs = {}
    for c in text:
        freqs[c] = freqs.get(c, 0) + 1

    # Calculate IoC
    sum_fi_2 = sum(f * (f - 1) for f in freqs.values())
    ioc = sum_fi_2 / (N * (N - 1))
    return ioc
```

Results for each manuscript:

- Text 1 ("OOK OOK ZOOK"): IoC = 0.400
- Text 2 ("OK KOZ KOZZ"): IoC = 0.250
- Text 3 ("ZOKO ZOKO OOK"): IoC = 0.345

Text 2 ("OK KOZ KOZZ") has the largest deviation from the expected IoC of 0.37, with an IoC of only 0.250. However, the sample sizes are so small that I wouldn't make any conclusions with confidence.

Problem 5

Using a Playfair cipher with the following array as the key,

```
N Y M P H
W A L T Z
Q U I C K
B O X E S
F D R G V
```

encrypt the message "He deceived me."

Solution:

- Split into digraphs: HE DE CE IV ED ME

Now, let's encrypt each digraph using the Playfair cipher with the given key grid:

- HE $\rightarrow$ PS
- DE $\rightarrow$ GO
- CE $\rightarrow$ EG (C and E are in the same column, so we move down)
- IV $\rightarrow$ KR
- ED $\rightarrow$ OG
- ME $\rightarrow$ PX

Therefore, the encrypted message is:

PSGOEGKROGPX

Problem 6

From your linear algebra classes, you might remember that the inverse of a 2×2 matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is given by $\frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$, provided that the determinant $ad - bc$ is non-zero.
The formula for the inverse of a 2×2 matrix in $\mathbb{Z}_{26}$ is almost the same, except that (once again!) since we cannot divide in $\mathbb{Z}_{26}$, we have to multiply by an inverse instead. The inverse of a 2×2 matrix in $\mathbb{Z}_{26}$ is $(ad - bc)^{-1}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$, provided that $\gcd(ad - bc, 26) = 1$.
Compute the inverse of these matrices mod 26. (We will need to be able to do this during week 3!)

(a) $\begin{bmatrix} 2 & 5 \\ 1 & 16 \end{bmatrix}$ (b) $\begin{bmatrix} 3 & 17 \\ 2 & 5 \end{bmatrix}$

Solution:

- (a) $\begin{bmatrix} 2 & 5 \\ 1 & 16 \end{bmatrix}$

  $det = (2 \cdot 16) - (5 \cdot 1) = 32 - 5 = 27 \equiv 1 \pmod{26}$

  Since $\gcd(1, 26) = 1$, this matrix has an inverse mod 26.

  The inverse will be $1 \cdot \begin{bmatrix} 16 & -5 \\ -1 & 2 \end{bmatrix}$

  Normalizing mod 26: $-5 \equiv 21 \pmod{26}$ $-1 \equiv 25 \pmod{26}$

  Therefore, the inverse is $\begin{bmatrix} 16 & 21 \\ 25 & 2 \end{bmatrix}$

- (b) $\begin{bmatrix} 3 & 17 \\ 2 & 5 \end{bmatrix}$

  $det = (3 \cdot 5) - (17 \cdot 2) = 15 - 34 = -19 \equiv 7 \pmod{26}$

  Since $\gcd(7, 26) = 1$, this matrix has an inverse mod 26.

  Using the extended Euclidean algorithm, we find that $7^{-1} \equiv 15 \pmod{26}$

  The inverse will be $15 \cdot \begin{bmatrix} 5 & -17 \\ -2 & 3 \end{bmatrix}$

  Normalizing mod 26: $15 \cdot 5 \equiv 23 \pmod{26}$ $15 \cdot (-17) \equiv 15 \cdot 9 \equiv 5 \pmod{26}$ $15 \cdot (-2) \equiv 15 \cdot 24 \equiv 22 \pmod{26}$ $15 \cdot 3 \equiv 19 \pmod{26}$

  Therefore, the inverse is $\begin{bmatrix} 23 & 5 \\ 22 & 19 \end{bmatrix}$