Quebrando a Cifra de Vigenere

Eduardo Freire dos Santos, 211010299
Ruan Petrus Alves Leite, 211010459

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
Segurança Computacional

dudufreiresantos@gmail.com, 211010459@aluno.unb.br

1. Introdução

Nesse projeto é introduzida a cifra de Vigenere e é implementado um método para quebrá-la que funciona consistentemente para textos em inglês ou português contendo ao menos 100 palavras.

2. Metodologia

2.1. Cifragem e Decifragem

A cifra de Vigenere é uma maneira simples de criptografar mensagens de texto. Para cifrar uma mensagem m, escolhemos uma string k chamada de chave e incrementamos cada caractere de m de acordo com o caractere correspondente em k, de maneira cíclica caso o final do alfabeto seja atingido.

Note que o tamanho da chave escolhida pode ser menor que o da mensagem. Para lidar com isso, simplesmente estendemos a chave ciclicamente até que esta possua a mesma quantidade de caracteres latinos que a frase original.

Dado uma mensagem criptografada e uma chave basta reverter o processo descrito acima, ou seja, ao invés de somar a cada caractere da mensagem o caractere correspondente da chave, utiliza-se a subtração.

A implementação desse processo em Python é direta e pode ser vista na Figura 1.

2.2. Quebrando a Cifra de Vigenere

Para quebrar a cifra de Vigenere foi utilizada uma variação da Análise de Kasiski, descrita por exemplo em [Rodriguez-Clark 2020].

A primeira etapa do ataque consiste em estimar o tamanho da chave usada para cifrar a mensagem. Para isso, buscamos na mensagem cifrada pela ocorrência de substrings de três letras (chamadas de trigramas) que se repetem ao longo da mensagem. Em seguida, listamos todos os fatores das diferenças das posições em que ocorre cada repetição, considerando os fatores que ocorrem mais frequentemente como os tamanhos mais plausíveis da chave. Podemos ver a implementação na Figura 2

Dado um tamanho k considerado plausível geramos uma chave que é considerada a melhor para aquele tamanho. Para isso, escolhe-se um valor qualquer para o primeiro caractere da chave e geramos uma tabela de frequência das letras que ocorrem de k em k caracteres após a decifragem da mensagem original utilizando o caractere escolhido. Esse caractere recebe uma pontuação baseada em quanto sua tabela de frequência difere

```python
def encrypt(text : str, key : str) -> str:
    cipher = ""
    key_idx = 0
    for i in range(len(text)):
        if not is_latin(text[i]):
            cipher += text[i]
            continue

        new_c = add_char(text[i], key[key_idx % len(key)])
        cipher += new_c
        key_idx += 1

    return cipher

def decrypt(text : str, key : str) -> str:
    cipher = ""
    key_idx = 0
    for i in range(len(text)):
        if not is_latin(text[i]):
            cipher += text[i]
            continue

        new_c = sub_char(text[i], key[key_idx % len(key)])
        cipher += new_c
        key_idx += 1

    return cipher
```

Figure 1. Implementação da cifra de Vigenere

```python
def trigram_period_table(
        words: list[str], text : str,
    ) -> dict[int, int]:
    result: dict[int, int] = {}
    text = text.lower()
    for word in words:
        for i in range(len(word)-2):
            periods = trigram_periods(word[i:i+3], text)
            add_dict(result, periods)

    return result

def trigram_periods(tri : str, text : str) -> dict[int, int]:
    occurrences = find_all(tri, text)
    differences = list()
    for i in range(len(occurrences)):
        for j in range(i+1, len(occurrences)):
            differences.append(occurrences[j] - occurrences[i])

    periods: dict[int, int] = {}
    for diff in differences:
        for d in range(1, diff + 1):
            if diff % d == 0:
                if d in periods:
                    periods[d] += 1
                else:
                    periods[d] = 1

    return periods
```

Figure 2. Achar períodos

```python
def find_key(
        cipher: str, key_size: int,
        language_letter_frequency: dict[str, float]
    ) -> str:
    key = ""
    for i in range(key_size):
        best_shift = "a"
        best_error = float("inf")
        for shift in ALPHABET:
            freq: dict[str, float] = defaultdict(float)
            total = 0
            for idx in range(i, len(cipher), key_size):
                if is_latin(cipher[idx]):
                    shifted_char = sub_char(cipher[idx], shift)
                    freq[shifted_char] += 1
                    total += 1

            for k in freq.keys():
                freq[k] *= 100/total

            error = frequency_table_error(freq, language_letter_frequency)
            if (error < best_error):
                best_error = error
                best_shift = shift

        key += best_shift

    return key
```

Figure 3. Achar melhor chave de tamanho k

da frequência esperada das letras em português ou em inglês, de acordo com a tabela encontrada em [Wikipedia 2014]. Para gerar essa pontuação é feita uma soma das diferenças dos quadrados entre as entradas de cada tabela e é escolhido para cada posição da chave o caractere com menor pontuação. A Figura 3 demonstra a implementação desse processo.

Assim, obtemos para cada tamanho avaliado (por padrão avaliamos 5 tamanhos) uma chave considerada a mais plausível para aquele tamanho. Utilizamos cada uma dessas chaves para decifrar o texto e escolhemos a chave que gera o texto que é julgado como o mais próximo de um texto ”válido” em português ou em inglês.

Para julgar a probabilidade de um texto ser válido em uma certa língua, olhamos para cada palavra contida neste e verificamos se ela é ”válida”, ou seja, se ela aparece em um dicionário da linguagem em questão [Wehar 2014] [omegahat 2011]. Atribuímos a proporção das palavras válidas em um texto como a probabilidade dele ser válido.

Esse processo é feito tanto em português quanto em inglês e a chave que gera o texto com maior probabilidade de validez entre ambas as linguagens é escolhida. A Figura 4 demonstra a implementação desse processo.

3. Conclusão

Foi implementado um programa capaz de encriptar, decriptar e quebrar a cifra de Vigenere. O programa está disponível no github.

Os resultados do programa estão satisfatórios para chaves de até 10 de tamanho e textos grandes. Para o programa ser capaz de encontrar melhores resultados com chaves maiores e textos menores seria necessário um reimplementação.

```python
def likelihood_of_valid_text(text: str, language_words: list[str]) -> float:
    text_words = get_latin_words(text)
    number_of_valid_words = reduce(
        lambda v, word: v + (
            1
            if word in language_words
            else 0
        ),
        text_words, 0
    )
    return number_of_valid_words / len(text_words)
```

Figure 4. Probabilidade do texto ser valido

References

[omegahat 2011] omegahat (2011). Portuguese vocabulary. https://github.com/omegahat/Rstem/blob/master/inst/words/portuguese/voc.txt. [Online; accessed 1-October-2023].

[Rodriguez-Clark 2020] Rodriguez-Clark, D. (2020). Kasiski analysis: Breaking the code. https://crypto.interactive-maths.com/kasiski-analysis-breaking-the-code.html#intro. [Online; accessed 1-October-2023].

[Wehar 2014] Wehar, M. (2014). Public domain word lists. https://raw.githubusercontent.com/MichaelWehar/Public-Domain-Word-Lists/master/5000-more-common.txt. [Online; accessed 1-October-2023].

[Wikipedia 2014] Wikipedia (2014). Relative frequencies of letters in other languages. https://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_other_languages. [Online; accessed 1-October-2023].