Relatório - Cifra de Vigenère

Camila Frealdo Fraga - 17/0007561
José Roberto Interaminense Soares - 19/0130008

Dep. de Ciência da Computação - Universidade de Brasília (UnB)

1. Introdução

A cifra de Vigenère é um dos exemplos mais clássicos de criptografia por substituição polialfabética, que foi inventada no século XVI e desafiou a criptoanálise por vários séculos. Neste relatório, exploramos o projeto dedicado à quebra da cifra de Vigenère, uma tarefa que envolve a aplicação de princípios fundamentais da criptoanálise, técnicas de análise de frequência, e o uso de algoritmos computacionais para decifrar mensagens codificadas. Este projeto revela a eficácia das ferramentas e técnicas modernas na quebra de um dos sistemas de criptografia históricos mais intrigantes e desafiadores.

2. Metodologia

2.1. Codificação

A codificação foi implementada na função chamada **encode**, essa função recebe dois argumentos: a mensagem a ser cifrada e a chave utilizada para a cifra. São codificadas apenas as letras, assim como na cifra de Vigenère, sendo ignorados quaisquer caracteres especiais.

O laço **for** percorre cada caractere da mensagem. Se o caractere for uma letra ela será codificada. Primeiramente é calculado o quanto ela será deslocada (shift) e em seguida é considerado se ela é maiúscula (então o shift será somado ao caractere da mensagem e depois subtraído de 'A') ou minúscula (somado ao caractere da mensagem e subtraído de 'a'). A operação de resto é necessária caso a letra cifrada passe do caractere 'z' ou 'Z'. A função retorna o texto completamente cifrado.

```
string encode (string msg, string key) {
  string encoded = "";
  key = cleanText(key);  // Limpa a chave
  int j = 0;
  for (int i = 0; i < msg.length(); i++) {
    if (isalpha(msg[i]) != 0) {
      int shift = key[j] - 'a';
      if (islower(msg[i]))
        encoded += (char)((msg[i] + shift - 'a') % 26 + 'a');
      else
        encoded += (char)((msg[i] + shift - 'A') % 26 + 'A');
```

```
      j++;
      if (j == key.length()) j = 0;
    }
    else encoded += msg[i];
  }
  return encoded;
}
```

2.2. Decodificação

A decodificação foi implementada na função **decode** e é similar à codificação, sendo que o *shift* realizado nas letras da cifra com base no caractere da chave é feito na direção oposta a anterior, ou seja, subtraindo do caractere um número de posições. A operação de resto também é necessária, uma vez que o shift pode passar do 'a' ou 'A'. A função retorna o texto completamente decifrado.

```
string decode (string cript, string key) {
  string decoded = "";
  key = cleanText(key);   // Limpa a chave
  int j = 0;
  for (int i = 0; i < cript.length(); i++) {
    if (isalpha(cript[i]) != 0) {
      int shift = key[j] - 'a';
      if (islower(cript[i]))
        decoded += (char)((cript[i] - shift - 'a' + 26) % 26 + 'a');
      else
        decoded += (char)((cript[i] - shift - 'A' + 26) % 26 + 'A');
      j++;
      if (j == key.length()) j = 0;
    }
    else decoded += cript[i];
  }
    return decoded;
}
```

2.3. Ataque

Para descobrir a chave foram realizados dois passos, sendo o primeiro uma estimativa do tamanho da chave, baseado no cálculo de índices de coincidência [Índice de coincidência] e o segundo passo, a análise de frequência. É possível dividir o texto em várias partes e recuperar a chave original. Dessa forma, foi criada uma função principal chamada **attack**, que contém os dois passos descritos acima. É importante que ela receba a linguagem que o texto se encontra, pois tanto o

índice de coincidência (IC) como a análise de frequência dependem do idioma escolhido, que é passado pelo usuário.

```
string attack (string cipher, int language) {
    cipher = cleanText(cipher);
    int key_length = keyLength(cipher, language);
    string key = keyFinder(cipher, key_length, language);
    return key;
}
```

2.3.1 Tamanho da Chave

A função **keyLength** recebe a cifra e a linguagem escolhida pelo usuário (PT ou EN) e faz uma previsão do tamanho da chave. A ideia é percorrer os possíveis tamanhos da chave (considerada de 2 até 20 caracteres) e, para cada um dos possíveis tamanhos, calcular o IC médio do bloco. Após isso, é calculada a diferença entre os ICs médios de cada bloco e o IC da língua do texto. O bloco de texto que obteve a menor diferença é considerado como o tamanho da chave, e é retornado pela função. Para o português foi considerado IC de 0.0745 e para o inglês IC de 0.0667 (FRWiki).

Uma definição de IC (Criptoanálise) pode ser vista a seguir:

> Definição (Índice de Coincidência)
> *Suponhamos que $x = x_1 x_2 \dots x_n$ é um texto de n caracteres do alfabeto. O índice de coincidência de x, designado por $I_c(x)$, é definido como a probabilidade de dois caracteres aleatórios de x serem iguais.*
>
> $$I_c(x) = \frac{\sum_{i=1}^{|\mathcal{A}|} \binom{f_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=1}^{|\mathcal{A}|} f_i(f_i - 1)}{n(n-1)}$$
>
> *com f_i a frequência do carácter de codificação i na mensagem.*

2.3.2 Análise de Frequência

A função **keyFinder** recebe três argumentos: o texto cifrado, o tamanho da chave e a linguagem escolhida pelo usuário. E retorna a chave descoberta.
Na primeira parte chamamos a função **divideCipher**, que recebe como argumentos o texto cifrado e o tamanho da chave, essa função divide o texto em blocos de tamanho até **n** (tamanho da chave), essa divisão é feita da seguinte forma: Por exemplo, dado o texto cifrado: "**mqggywwkhjvqltqkvkbmmptkymbxk**" e tamanho de chave igual a 5 essa função irá retornar o seguinte vetor: {{**mwvkmm**}, {**qwqvpb**}, {**gklktx**}, {**ghtbkk**}, {**yjqmy**}}. Cada string do vetor representa um bloco de caracteres.

Na segunda parte, cada bloco será responsável por descobrir um caractere da chave, pois cada bloco contém letras que foram cifradas pelo mesmo caractere da chave. Aqui analisamos as frequências de cada letra em cada bloco com a função **calculateLetterFrequencies**.

Neste ponto temos a frequência de aparição de cada letra que foi cifrada pelo mesmo caractere, similar a uma cifra de César. Precisamos agora analisar dois conjuntos de dados e descobrir qual é o melhor deslocamento que "casa" os dois gráficos (Breaking Vigenere Cypher - 2:30). Para isso, precisamos levar em conta todas as frequências e calcular a diferença entre a aparição de cada caractere no gráfico desejado e o gráfico atual. No código, isso foi feito com uma função auxiliar chamada **diff.** Cada conjunto de dados pode ser deslocado em 26 posições, pois temos 26 letras no alfabeto e isso foi feito com a função auxiliar **shiftValues**. Após testar todos os 26 deslocamentos podemos analisar as diferenças no vetor e escolher a menor dessas diferenças. Achando o índice da menor diferença conseguimos saber qual caractere foi responsável por cifrar aquela parte do texto e então temos um caractere da chave. Esse código é repetido até o tamanho da chave, retornando a chave completa.

A função de achar cada caractere da chave pode ser vista a seguir:

```
char keyGuessLetter (map<char,float> cipher_letter_frequencies, int language)
{
  vector<float> differences;
// diferencas entre as frequencias das letras do texto cifrado e as
frequencias das letras em portugues
    map <char,float> default_letter_frequencies;

    if (language == 1) default_letter_frequencies =
portuguese_letter_frequencies;          // define as frequencias das letras
do idioma
    else if (language == 2) default_letter_frequencies =
english_letter_frequencies;

    for (int i = 0; i < 26; i++) {
        map<char,float> cipher_shifted =
shiftValues(cipher_letter_frequencies, i); // faz o shift das frequencias das
letras do texto cifrado
        float difference = diff(cipher_shifted, default_letter_frequencies);
// calcula a diferenca entre as frequencias
        differences.push_back(difference);
// armazena a diferenca
    }
    int min_index = min_element(differences.begin(), differences.end()) -
differences.begin();  // acha o indice do menor valor no vetor de diferencas
    return char('a' + min_index);
}
```

3. Conclusão

Com este trabalho vimos que é possível quebrar a cifra de Vigenère através de técnicas de análise de frequência e padrões de repetição. É possível concluir que uma senha grande é muito mais segura, mas ainda possível de ser quebrada. Apesar de ter sido eficiente durante vários séculos, podemos concluir que hoje devem ser utilizadas técnicas mais modernas e fortes, que exigem mais poder computacional para quebrar por força bruta.

Durante a implementação do trabalho algumas etapas foram bastante desafiadoras, uma etapa que foi bastante desafiadora foi a implementação da função **keyLength,** primeiramente tentamos implementá-la através do cálculo de distâncias entre grupos de bloco de 3 caracteres, calculando as distâncias em que os blocos mais frequentes apareciam, com essas distâncias era possível calcular os divisores e achar os que mais apareciam, mas dessa forma nem sempre era possível determinar o valor certo para o tamanho da chave. Após diversos testes, optamos por utilizar o cálculo do índice de frequências que foi explicado na seção metodologia. Dessa forma reduzimos nossos erros e conseguimos descobrir chaves com tamanhos maiores.

Referências

[Funcionamento-Vigenere]

[FRWiki] Índice de coincidência.
https://pt.frwiki.wiki/wiki/Indice_de_co%C3%AFncidence

[Informata] Quebrando a Cifra de Vigenère, 2013
http://informatabrasileiro.blogspot.com/2013/04/quebrando-cifra-de-vigenere.html?m=1

[Ataque-Vigenere] Ataques às Cifras de Substituição (Cifra de Vigenère), 2021
https://www.youtube.com/watch?v=b00ektDHDXA&t=700s&ab_channel=GaneshICMC

[Vinegere-Breaker] How to crack Vigenère ciphers, 2021.
https://www.youtube.com/watch?v=6qXFwH_JXeY

[Criptoanálise] Doutor Pedro Quaresma de Almeida, 2010.
https://www.mat.uc.pt/~pedro/lectivos/CodigosCriptografia1011/apontamentos45a64.pdf

[Frequencia das letras] Frequência de letras - Wikipedia.
https://pt.wikipedia.org/wiki/Frequ%C3%AAncia_de_letras

[Breaking Vigenere Cypher] Cryptography - Breaking the Vigenere Cipher, 2014.
https://www.youtube.com/watch?v=P4z3jAOzT9I&t=142s