Técnicas de Programação 1
Primeira Lista de Exercícios

Rodrigo Bonifácio

6 de abril de 2017

Parte 1: Caesar Cipher

O uso de códigos como meio de ocultar o significado das mensagens tem origens remotas— por exemplo, o primeiro uso militar conhecido é o de Julius Caesar (50-60 AC). O *Caesar Cipher* especifica que cada letra do alfabeto é codificada com uma letra três posições a frente no alfabeto. Por exemplo, o caracter 'a' é codificado como 'd', 'b' é codificado como 'e', 'c' é codificado como 'f' e assim sucessivamente. A codificação retorna ao início no fim do alfabeto, ou seja, o caracter 'x', 'y' e 'z' são codificados como 'a', 'b' e 'c', respectivamente. O mapeamento completo de letras é descrito na Figura 1

```
abcdefghijklmnopqrstuvwxyz
|||||||||||||||||||||||||||
defghijklmnopqrstuvwxyzabc
```

Figura 1: Caesar Cipher usando uma chave igual a 3.

A `classe Caesar` (disponível no ambiente da disciplina) contem uma implementação parcial do *Caesar Cipher*. Os métodos `codificar` e `decodificar` podem ser usados para codificar / decodificar letras minúsculas com o deslocamento em *key* posições no alfabeto. Modifique esses métodos de tal forma que eles possam lidar com letras maiúsculas e caracteres especiais. Uma letra maiúscula deve ser codificada / decodificada da mesma forma que as letras minúsculas. Por outro lado, caracteres especiais devem ser preservadas (o caracter '!' deve ser mantido como '!').

Parte 2: Rotation Cipher

O Caesar Cipher segue a abordagem de substituição, suscetível a ataques que o torna pouco eficiente na prática. Em particular, as letras possuem probabilidades diferentes de ocorrência nos textos em um dado idioma, de tal forma que

a frequência relativa dos caracteres em um texto codificado pode servir para quebrar a segurança. Por exemplo, 'e' é o caracter mais frequente no Inglês. Caso a letra 'w' apareça de forma mais frequente em uma mensagem codificada, existe razoável probabilidade da letra 'e' estar sendo mapeada na letra 'w'.

Uma forma de contornar essa limitação presente nas técnicas de substituição é adicionar *rotação* ao algoritmo. Após qualquer letra ser codificada, é feita uma *rotação* da chave, de tal forma que, na primeira codificação, o caracter 'a' seria mapeado para o caracter 'd'; em uma segunda codificação, o mesmo caracter 'a' deveria ser mapeado no caracter 'e'; em uma terceira codificação, o mesmo caracter 'a' seria mapeado no caracter 'f'; e assim sucessivamente (ver Figura 2).

```
abcdefghijklmnopqrstuvwxyz
||||||||||||||||||||||||||
defghijklmnopqrstuvwxyzabc (mapeamento inical)

abcdefghijklmnopqrstuvwxyz
||||||||||||||||||||||||||
efghijklmnopqrstuvwxyzabcd (mapeamento apos o primeiro encode)

abcdefghijklmnopqrstuvwxyz
||||||||||||||||||||||||||
fghijklmnopqrstuvwxyzabcde (mapeamento apos o segundo encode)

abcdefghijklmnopqrstuvwxyz
||||||||||||||||||||||||||
ghijklmnopqrstuvwxyzabcdef (mapeamento apos o terceiro encode)
...
```

Figura 2: Caesar Cipher usando uma chave igual a 3, após sucessivas codificação.

Modifique a classe `Caesar` de tal forma que seja feita uma rotação após cada codificação / decodificação. Para isso, implemente um novo método `rotaciona`, que rotaciona o atributo `key` de uma instância da classe `Caesar`. Decodifique o arquivo `sample.txt` usando como chave inicial a string:

```
jklmnopqrstuvwxyzabcdefghi
```

Caso o conteúdo esteja legível, sua implementação está (possivelmente) correta.

Parte 3: Um modelo simples da máquina Enigma

Conforme mencionado, algoritmos de criptografia baseados no princípio de substituição são vulneráveis a ataques estatísticos. Cientes disso, durante a Segunda Grande Guerra o Exército Alemão utilizou um esquema de criptografia que mitiga tal vulnerabilidade. Este esquema era implementado com máquinas

(de dimensões semelhantes às máquinas de escrever) conhecidas como *Enigma*, que trouxeram vantagens estratégias a Alemanha durante os primeiros anos de guerra. De forma semelhante, a quebra desse esquema de codificação por pesquisadores ingleses (incluindo Alan Turing) é considerada um marco para o encerramento do conflito.

As máquinas *Enigma* usam rotores que podem ser posicionados em diferentes orientações, variando os padrões de substituição. De forma mais específica, os rotores rotacionam após cada caracter ter sido codificado, alterando o padrão de substituição e tornando bastante difícil a quebra do algoritmo de codificação. O comportamento dos rotores podem ser modelados, de uma forma simplificada, por um dispositivo que consiste de anéis concêntricos. Por exemplo, o modelo da Figura 3 possui três anéis contemplando as letras do alfabeto.

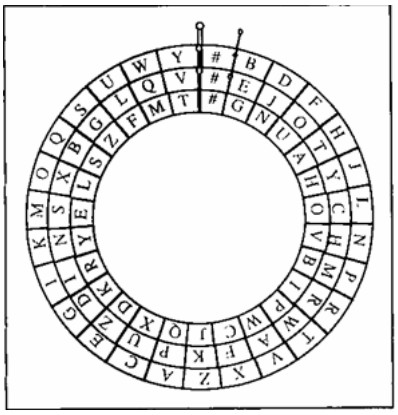

Figura 3: Modelo simplificado da máquina Enigma.

Para cifrar um caracter c_1 usando esse modelo, primeiro é necessário encontrar o caracter correspondente no rotor mais interno e anotar o caracter no alinhamento correspondento com o rotor mais externo (c_2). Em seguida, identifique o caracter c_2 no rotor intermediário e retorna o caracter c_3 no alinhamento correspondente do rotor mais externo. Após o caracter c_3 ser retornado, rotacione o rotor mais interno em sentido horário. Sempre que o rotor mais interno retorna a orientação original, o rotor intermediário realiza uma rotação (também em sentido horário). Por exemplo, na configuração da Figura 3, o caracter 'A' seria codificado como 'N', uma vez que o rotor mais interno está alinhado com o caracter 'H' no rotor mais externo, e o caracter 'H' no rotor intermediário está aninhado com o caracter 'N' no rotor mais externo. Após essa codificação, o rotor mais interno deve ser rotacionado no sentido horário em uma posição, de tal forma que, nesse novo cenário, o caracter 'A' seria codificado como o caracter 'D'. Note que a decodificação segue um processo inverso (partindo do rotor

mais externo), observando a restrição que os anéis deveriam ser retornados á configuração inical.

Implemente uma classe `Enigma` que recebe como construtor a configuração dos três anéis (mais interno, intermediário e mais externo, respectivamente) como strings (contemplando apenas caracters em maiúsculo e considerando o caracter `#` como espaço; conforme Figura 3). Utilizando a sua implementação, devemos ser capazes de codificar / decodificar textos usando o algoritmo descrito neste documento. Perceba que a configuração inical dos anéis mais interno e intermediário podem ser ajustados com base nas sucessivas codificações. Por outro lado, conforme mencionado, quando for realizada a decodificação de um texto, a máquina precisa ser reconfigurada para o estado inicial.

Parte 4: Estruturas de Dados

Utilizando como referência a implementação da classe `LinkedList` (e do *trait* `Lista`) vistos em sala de aula, implemente uma biblioteca de estruturas de dados em Scala contemplando:

- estruturas lineares (fila, pilha, lista duplamente encadeada)
- estruturas hierárquicas (árvores binárias de busca)
- grafos e algoritmos sobre grafos

A apresentação dos resultados pode apoiar o cálculo da nota da primeira prova.