Criptografia

Praciano-Pereira, T. *

30 de março de 2020
preprints da Sobral Matemática
no. 2020.03
Editor Tarcisio Praciano-Pereira
tarcisio@sobralmatematica.org

Resumo

Criptografia é um método matemático da computação para tornar dados secretos. O matemático e teórico da computação, *Alan Turing*, que fez vários trabalho em computação teórica, trabalhou durante a segunda guerra mundial para quebrar com sucesso o segredo da criptografia da máquina alemã Enigma. Vou mostrar que criptografia é basicamente permutação e dar alguns exemplos.

palavras chave: Alan Turing, criptografia, permutação

Cryptography is a mathematical method of computation to make data covered. Alan Turing which has written several papers on theory of computation, before any computation had been invented, has worked to uncover the secret of the German machine, Enigma, successfully. I will show here that cryptography is essentially permutation.

keywords: Alan Turing, cryptography, permutation.

*tarcisio@sobralmatematica.org

1 O objetivo: criptografia é permutação

Criptografia é um método matemático da computação para tornar dados secretos. O matemático e teórico da computação, *Alan Turing*, que fez vários trabalho em computação teórica, em particular ficou muito conhecido por um algoritmo chamado *máquina de Turing*, trabalhou durante a segunda guerra mundial para quebrar o segredo da criptografia da máquina alemã, *Enigma*, o que conseguiu com êxito. Enigma
Eu vou aqui dar um exemplo, e inclusive mostrar-lhe dois programas escritos em `Python` que podem ser usados como exemplo da máquina criptográfica alemã. Os meus programas *não estão funcionando corretamente*, e vou publicá-los mesmo assim, dentro do espirito de quem escreve programas de domínio público, é possível que alguém se interesse e me ajude a corrigi-los. Também vou mostrar um programa que funciona indicando a fonte de como usá-lo.

2 história da criptografia

Eu não sou historiador de modo que dizer que vou contar um pouco da história representa uma distorção. Eu vou tomar como ponto de partida uma história de sucesso com o inglês *Alan Turing* no centro da cena envolvido em quebra o códico da máquina alemã, *Enigma*, que ele conseguiu com êxito e segundo se diz, abrindo caminho para encerrar a seguda guerra mundial indicando onde deveria ocorrer uma invasão das tropas aliadas que foi determinante para a retomada da França pelos aliados. O que interessa aqui é que havia uma máquina elétrica alemã, *Enigma* sobe a qual os ingleses haviam conseguido informações e que, simplificando, para os meus objetivos, era uma máquina para permutar o alfabeto e assim encriptar mensagens.

Em suma, encriptar significa permutar o alfabeto e assim esconder o conteúdo das mensagens.

3 Alguns exemplos de encriptação

Eu vou aqui dar um exemplo, e inclusive mostrar-lhe dois programas escritos em `Python` que podem ser usados como exemplo da máquina criptográfica alemã. Os meus programas *não estão funcionando corretamente*, e vou publicá-los mesmo assim, dentro do espirito de quem escreve programas de domínio público, é possível que alguém se interesse e me ajude a corrigi-los. Também vou mostrar um programa que funciona indicando a fonte de como usá-lo.

O meu objetivo aqui é falar de *criptografia* do ponto de vista da Matemática, como um método, que usa uma poderosa ferramenta que é a *permutação*. Ao final eu vou voltar a esta questão, depois de dar os exemplos.

A máquina alemã de *criptografia* era muito ingeniosa e simples e representou um primeiro exemplo de criptografia automatizada. Ela não fazia nada mais do que criar uma permutação do alfabeto, coisa que vou fazer no programa, de modo que quem tivesse a chave da permutação poderia desfazer o segredo e ler a mensagem original. É possível encontrar fotos desta máquina usando a chave *criptografia*, mas eu vou descrê-la rapidamente até porque, passados 70 anos, a tecnologia envolvendo esta máquina ficou de tal modo ultrapassada que possivelmente minha descrição terá o sabor de falar dum passado distante, para muitas pessoas.

Vou começar falando de *datilografia* que hoje ficou substituída por *digitação*. Antes havia *máquinas de datilografia*, *dátilos* do grego, *dedo*. Eram máquinas para que escrevêssemos com os dedos. Também havia os *datilógrafos*, que eram *digitadores* do passado...Eu sou um *datilógrafo*, escrevo com meus dados no teclado do meu computador sem olhar para o teclado, posso ler um texto dum arquivo escrevê-lo noutro praticamente na velocidade de leitura.

Haviam máquinas muito avançadas, há 60 anos atrás, nas quais era possível trocar o conjuntos dos caracteres. Uma delas usava uma esfera que tinha na hemisfério superior os *caracteres minúsculos* e

no hemisfério inferior os *caracteres maiúsculos*. Trocando a *bola* você poderia trocar o tipo de letra ou mesmo de alfabeto, por exemplo, para escrever em cirílico.

A máquina de *criptografia* alemã *trocava a bola* permutando a ordem do alfabeto, como vou fazer no próximo exemplo. Ela produzia uma *permutação* do alfabeto. No caso da *Enigma*, em vez de bolas eram usados cilindros com canecões elétricas. Eu nem imagino o trabalho que daria para se *descriptografar* do outro lado .... Para encriptar, era fácil, bastava trocar um cilindro! Do outro lado alguém teria que trocar cada letra por sua correspondente permutada a partir duma tabela da permutação que era descrita por lâmpadas acendendo quando o valor certo fosse atingido.

Observe como exemplo, eu vou criptografar e depois vou descriptografar um pequeno texto retirado dum parágrafo anterior. Eu fiz algumas modificações para poder identificar entre as duas formas um *pedaço de texto* que deixei marcado.

- criptografando Rfgn zádhvan nyrzã ren zhvgb vatravbfn r fvzcyrf znf ercerfragn hz cevzrveb rkrzcyb qr pevcgbtensvn nhgbzngvmnqn. Ryn aãb snmvn anqn znvf qb dhr pevne hzn crezhgnçãb qb nysnorgb pbqvsvpnaqb n zrafntrz bevtvany, pbvfn dhr ibh snmre ab cebtenzn, qr zbqb dhr dhrz gvirffr n punir qn crezhgnçãb cbqrevn qrfsnmre b frterqb r yre n zrafntrz bevtvany.

- descriptografando Esta máquina alemã era muito ingeniosa e simples mas representa um primeiro exemplo de criptografia automatizada. Ela não fazia nada mais do que criar uma permutação do alfabeto codificando a mensagem original, coisa que vou fazer no programa, de modo que quem tivesse a chave da permutação poderia desfazer o segredo e ler a mensagem original.

Você pode ver nos dois itens acima o texto criptografado e depois o mesmo texto descriptografado em que eu deixei sublinhado um pedaço de texto que eu repeti, propositalmente, para identificá-lo. Você também pode observar uma curiosidade, *ã*, da palvra alemã, não foi alterado, porque o programa usado é americano e no alfabeto americano não existe o símbolo *ã* e ele foi ignorado. Aconteceu o mesmo com o símbolos *á,ç*. Mas é possível corrigir este defeito.

Eu modifiquei um antigo programa que tenho, feito em `python`, para construir estes exemplos, mas não fui muito feliz. Consigo *encriptar* mas não consigo reproduzir o texto original. Eu lhe mostro abaixo o meu *mal exemplo*, até para convidar alguém a analisar o meu programa para descobrir onde estou errando. O meu programa, na verdade são dois [2, encripta.py] e [2, desencripta.py]

- criptografando com meu programa Esta mĩduina alemõ era muito inteniosa e simples mas reprefenga hm crimrieb ejemplo de prictotrafia automagizada: Eza nõo salia nada maif qb dhe criar uma permutaíõo qo azfaoeto, povsa due ioh falrr no crbgeama: de modo que quem tivesse a puave da permhtaíõo coderia dessaler o fegredb e ler a menfagrm beiginal:

- descriptografando com meu programa Rsta máquina alemé era mhito ingenibsa e simpzes maf recreseatn um priwevro exemplo de criptoteafia ahgomatizada; Rla néb fazia aada wnvs do qur criar uma permutaçéo do alfaoego, coifa qhe vou sazrr ab peotrawn; de modo que duem tivesse a phaie da cerwugaçéo poderva desfnzer b seterqo e ler a mensagew oritvanz;

4 Um program `/Debian/Gnu/linux`

Deixe-me repetir o *bom exemplo* de encriptação e mostrar-lhe o programa `tr` que usei para executá-lo.

No primeiro exemplo, eu criptografei e depois vou descriptografei um pequeno texto no qual fiz uma modificação repetindo o texto "a mensagem original" para poder identificar entre a mensagem original e sua forma encriptada, onde ele se encontrasse. Uma das formas se encontra exatamente no fim da mensagem.

- criptografando Rfgn zádhvan nyrzã ren zhvgb vatravbfn r fvzcyrf znf ercerfragn hz cevzrveb rkrzcyb qr pevcgbtensvn nhgbzngvmnqn. Ryn aãb snmvn anqn znvf qb dhr pevne hzn crezhgnçãb qb nysnorgb pbqvsvpnaqb n zrafntrz bevtvany, pbvfn dhr ibh snmre ab cebtenzn, qr zbqb dhr dhrz gvirffr n punir qn crezhgnçãb cbqrevn qrfsnmre b frterqb r yre n zrafntrz bevtvany.

- descriptografando Esta máquina alemã era muito ingeniosa e simples mas representa um primeiro exemplo de criptografia automatizada. Ela não fazia nada mais do que criar uma permutação do alfabeto codificando a mensagem original, coisa que vou fazer no programa, de modo que quem tivesse a chave da permutação poderia desfazer o segredo e ler a mensagem original.

Você pode ver nos dois itens acima o texto criptografado e depois o mesmo texto descriptografado em que eu deixei sublinhado um pedaço de texto que eu repeti, propositalmente, para identificá-lo. Você também pode observar uma curiosidade, *ã*, da palvra alemã, não foi alterado, porque o programa usado é americano e no alfabeto americano não existe o símbolo *ã* e ele foi ignorado. Aconteceu o mesmo com o símbolos *á,ç*. Mas é possível corrigir este defeito, e para isto basta incluir as letras da lingua porguesa ao final da matriz de dados com a correspondente permutação na matriz de traduções (permutação). Vou explicar melhor isto mais adiante.

A leitura do do meu programa, embora ele não esteja funcioando bem, ajuda a entender o método do programa `tr`. Eu modifiquei um antigo programa que tenho, feito em `python`, para construir estes exemplos, mas não fui muito feliz. Consigo *encriptar* mas não consigo reproduzir o texto original. Eu lhe mostro abaixo o meu *mal exemplo*, até para convidar alguém a analisar o meu programa para descobrir onde estou errando. O meu programa, na verdade são dois [2, encripta.py] e [2, desencripta.py]

Deixe-me explicar como se usar o `tr` um um *utilitário* que faz parte das distribuições `Debian/Gnu/Linux` mas que foi produzido pelos programadores das primeiras versões do `Unix`, `tr`, um programa que serve para "*traduzir*" codificações. Vale a pena conhecê-lo, num toque você pode *traduzir* um texto de *maiúsculas* para *minúsculas*, por exemplo. Ou fazer uma encriptação. Eu usei os comandos:

- criptografando

```
tr '[A-Za-z]' '[N-ZA-Mn-za-m]' < original > saida;
```

- descriptografando

```
tr '[N-ZA-Mn-za-m]' '[A-Za-z]' < saida > original
```

Eu vou explicar estes dois comandos em detalhe e em seguida vou escrever os dois comandos que servem para transformar de *maiúsculas* para *minúsculas* e vice versa. Mas sugiro que você, na sua *máquina* `Gnu/Linux` use

`info tr` ou `man tr`

para ver as possibilidades que este aplicativo oferece.

Vou repetir os comandos acima acrescentando uma descrição do seu conteúdo.

- criptografando

```
tr '[A-Za-z]' '[N-ZA-Mn-za-m]' < original > saida;
```

1. `tr` é o comando, na verdade uma função provavelmente escrita em `C` que recebe 4 parâmetros.
2. '[A-Za-z]' é o *primeiro conjunto* uma listagem dos caracteres, primeiro desde 'A' até 'Z' e depois de 'a' até 'z'. A informação do programa diz que você deve lhe passar dois conjuntos de caracteres e que o programa irá traduzir os caracteres, na mesma ordem, em que eles aparecerem no primeiro conjunto usando o correspondente no segundo conjunto. É exatamente o que eu faço no meu *programa-falho* escrito em `python`.

3. '[N-ZA-Mn-za-m]' é o *segundo conjunto*, uma listagem dos caracteres, agora permutados, a serem usados na mesma posição que corresponde ao primeiro conjunto.
4. $<$ *original* é um comando do `bash` que indica de onde vem os dados, compare com o próximo,
5. $>$ *saida* é um comando do `bash` que indica para onde vão os dados.
   Resumindo, `tr` vai traduzir `original` usando os dois conjuntos como tabelas de referência e colocar o resultado final em `saida`. Será preciso que os dois conjuntos tenham a mesma quantidade de caracteres porque eles serão usados de forma "*sincronizada*. Se isto não acontecer os resultados não são fáceis de se prever, coisa típica da linguagem `C`, um programa em `C` deve ser usado como foi previsto para ser usado... Entretanto *nenhum problema vai acontecer com sua máquina* se você usar o programa de forma errada, mas você pode perder os arquivos `original` e `saida`, portanto, se estiver experimentando é bom que você tenha *cópias de reserva* dos mesmos.

- descriptografando
  `tr '[N-ZA-Mn-za-m]' '[A-Za-z]' < saida > original`
  1. `tr` é o comando, na verdade uma função provavelmente escrita em `C` que recebe 4 parâmetros. Agora é a operação inversa portanto não vou falar muito.
  2. Troquei a ordem dos dois conjuntos, porque agora vou traduzir de volta.
- `tr [:lower:] [:upper:] < original > saida` vai transformar todo o texto que estiver em `original` em maiúsculas colocando o resultado em `saida`. Troque a ordem de `[:lower:] [:upper:]` para obter o resultado inverso.

5 Concluindo a discussão com permutações

Os meus programas seguem a mesma ideia de `tr` e pode ser que eu ainda consiga fazê-los funcionar corretamente. Digo "talvez" porque não se trata dum assunto de relevância uma vez que há coisa muito melhor e muito mais poderosa do que estes meus programas, e logo em seguida vou fazer referência a uma poderosa ferramento de criptografia que é distribuída sob `GPL`.

A *chave da questão* é que uma *permutação*, σ, é uma função bijetiva dum conjunto de caracteres nele mesmo, portanto tem uma função inversa que desfaz o segredo. Então eu preciso de dois programas:

- `encripta.py` para esconder a mensagem, aplicando σ,
- `desencripta,py` para devolver a mensagem original, aplicando σ^{-1}, a permutação inversa.
- Não é necessário ter dois programas, basta fazer como no caso do `tr`, passar a lista de caracteres, os dois conjuntos. Então para *desencriptar* basta trocar a ordem dos dois conjuntos.

Permutação é um dos assuntos estudos no ensino médio no capitulo *análise combinatória*. O assunto vai bem mais além do se ensina no ensino médio. As permutações são funções bijetivas dum conjunto nele mesmo. Se o conjunto for finito, com n elementos, existe $n!$ permutações diferentes e é isto que torna difícil desencriptar um texto encriptado. Como o nosso alfabeto tem 24 letras maiúsculas e outro tanto minúsculas, tem

$$24!620448401733239439360000 \quad (1)$$

maneiras diferentes de fazer a encriptação. Certamente os alemães não disponham de $24!$ bolas o que tornou possível que *Alan Turing* quebrasse o segredo alemão. Deixe-me tomar um exemplo mais simples.

o	I	I_{123}	I_{132}	I_{12}	I_{13}	I_{23}
I	I	I_{123}	I_{132}	I_{12}	I_{13}	I_{23}
I_{123}	I_{123}	I_{132}	I	I_{13}	I_{23}	I_{12}
I_{132}	I_{132}	I	I_{123}	I_{23}	I_{12}	I_{13}
I_{12}	I_{12}	I_{23}	I_{13}	I	I_{132}	I_{132}
I_{13}	I_{13}	I_{12}	I_{23}	I_{123}	I	I_{132}
I_{23}	I_{23}	I_{13}	I_{12}	I_{132}	I_{123}	I

O alfabeto, neste exemplo, tem três caracteres, $\{1, 2, 3\}$ o que produz $3! = 6$ permutações, elas aparecem nas colunas da tabela acima. Alan Turing pode ter feito uma tabela destas, ele tinha tempo, estavam em guerra, e ele estava trabalhando dedicado a este projeto. Deve ter espalhado numa parede as

$$620448401733239439360000$$

colunas da tabela... Observe que na tabela aparece I, a identidade, quando um par de permutações for, uma, o inverso da outra. Num destes casos aparece o alfabeto. A análise dos pares de inversos leva a descobrir qual e a permutação que estiver sendo usada. Claro que não foi fácil, mas ele conseguiu.

Alan Turing trabalhou neste projeto e noutros que precederam grande parte da computação teórica antes de 1954 quando morreu. O primeiro computador eletrônica, o ENIAC foi montado nos EU América do Norte em 1945 e praticamente aplicava tabelas em cálculos. Isto é suficiente para resolver, *aproximadamente*, equações diferenciais ordinárias que era o que *guerreiros* da America do Norte precisavam para os projetos balísticos.

Hoje, no meu computador portátil eu posso construir e trabalhar com permutações usando cálculo matricial, [2, Sim_3.calc] você pode ver a representação matricial das permutações de $\{1, 2, 3\}$ e posso montar a tabela de cálculo das permutações usando produto de matrizes. Confira também [2, Permutacao.calc].

Corrigir o meu programa `encripta.py` é uma brincadeira que vale a pena porque ele mostra de forma bem simples como usar permutações para *encriptação*. Mas profissionalmente você deve usar `gpg`, distribuído por `Debian/Gnu/Linux`. Instalado `gpg`, use

`info gpg` ou

`man gpg`

para ver como funciona. Mas como exemplo simples eu lhe mostro o que eu mesmo faço. Eu tenho um arquivo chamado *pass* que contém a lista das minhas *passwords*.

- encriptando `gpg -c -opass.gpg pass` vai criar o arquivo encriptado `pass.gpg` solicitando-me uma senha e confirmando a senha solicitada. Problema, se errar a senha o arquivo *pass* pode ser apagado. Tenha uma cópia de segurança do mesmo.

- desencriptando `gpg -d -opass pass.gpg` pergunta a senha, se bater, coloca os dados em `pass`.

Índice Remissivo

Referências

[1] David I. Bell Landon Curt Noll and other. Calc - arbitrary precision calculator. Technical report, http://www.isthe.com/chongo/, 2011.

[2] T Praciano-Pereira. Programas para cálculo numérico. Technical report, http://www.calculo-numerico.sobralmatematica.org/programas/, 2009.

[3] Tarcisio Praciano-Pereira. *Cálculo Numérico Computacional*. Sobral Matematica, 2007.