universidade de aveiro

Ano Letivo 2020/2021

Métodos Probabilísticos para Engenharia Informática

Relatório da Secção para Avaliação 3

Turma P7

Realizado por:

Ricardo Antunes, 98275

Henrique Sousa, 98324

Conteúdo

Introdução

Considere a geração aleatória de palavras em português com base em cadeias de Markov. Para avaliar a eficiência de um gerador de palavras aleatórias, considere 2 parâmetros:

1) número de palavras diferentes geradas
2) A probabilidade de uma palavra gerada ser uma palavra portuguesa válida.

Quanto mais altos esses dois parâmetros, mais eficiente é o gerador de palavras aleatórias. Nas experiências seguintes, considere a geração aleatória de palavras portuguesas compostas apenas pelas letras 'a', 'm', 'o' e 'r'. Considere também o arquivo "wordlist-preao-20201103.txt" (disponível em https://natura.di.uminho.pt/download/sources/Dictionaries/wordlists/) com uma lista de quase 1 milhão de palavras portuguesas válidas, organizado em 1 palavra por linha.

Todas as questões foram resolvidas recorrendo à função crawl fornecida pelos docentes.

```
function state = crawl(H, first, last)
    state = [first];
    while (1)
        state(end+1) = nextState(H, state(end));
        if (state(end) == last)
            break;

        end
    end
end

function state = nextState(H, currentState)
    probVector = H(:,currentState)';
    n = length(probVector);
    state = discrete_rnd(1:n, probVector);
end

function state = discrete_rnd(states, probVector)
    U=rand();
    i = 1 + sum(U > cumsum(probVector));
    state= states(i);
end
```

1.

(a) Defina, no Matlab, a matriz de transição de estados T e simule a geração de uma palavra aleatória.

Para facilitar a realização do guião definimos uma função denominada “createWord” que irá ser usadas nos restantes exercícios da seguinte forma:

```
function [word] = createWord(T)
random = randi(4, 1); %numero random entre 1 e 4
% random equivale ao estado('letra') inicial
% 1 equivale a 'r'
% 2 equivale a 'o'
% 3 equivale a 'm'
% 4 equivale a 'a'
state = crawl (T, random, 5);
str = ""; %string que servira para ser a palavra
    for n = 1 : length(state)
        switch (state(n))%incrementar a letra correspondente de cada elemento da matrix formada pela funcão crawl
                case 1
                    str = str + "r";
                case 2
                    str = str + "o";
                case 3
                    str = str + "m";
                case 4
                    str = str + "a";
            case 5 %equivale ao ponto ('.')
                    break
        end
    end
    word = str;
end
```

Considerando a matriz de transição de estados T:

```
T = [0    1/3    0       0.25    0       %COLUNAS: r-o-m-a-. ;
     0.5  0      0.5     0.25    0       %linhas : r o m a . ;
     0    1/3    0       0.25    0
     0.5  0      0.5     0       0
     0    1/3    0       0.25    0]
```

Sendo os estados ordenados por “r, o, m, a, .” .

Para gerar a palavra chamamos a função criada;

```
word = createWord(T)
```

O output deste programa (corrido 3 vezes) foi o seguinte:

```
mamomo    aramora    aro
```

(b) Simule a geração de 10^5 palavras aleatórias para estimar a lista de palavras geradas e a probabilidade de cada palavra. Quantas palavras diferentes foram geradas? Apresente as 5 palavras com as probabilidades estimadas mais altas e respetivos valores de probabilidade

Para responder à questão (b) começamos por criar um Cell Array com todas as 10^5 palavras geradas

```
words = {};
wordsCount = {};
count = 1;
nWords= 1e5;
% ciclo for para criar nWords(10^5) palavras
for N = 1 : nWords
    words{end+1} = createWord(T);
end
%
```

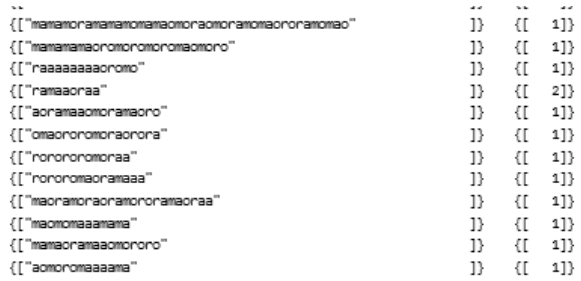
De seguida para calcular o número de palavras diferentes geradas e as vezes que cada uma das palavras foi gerada criamos outro cell array mas desta vez com 2 colunas, onde a primeira coluna representa cada palavra diferente e a segunda coluna o número de vezes que a palavra desta mesma linha foi gerada:

```
% inicializar o novo cell array onde 1ª coluna = palavra
% 2ª coluna = numero de vezes que foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;
%

%ciclo for a iniciar no 2º elemento (pois o primeiro já foi gerado)
for J = 2 : length(words) %este ciclo percorre todas as palavras geradas anteriormente
    found = false; % variavel que informa se a palavra foi ou nao encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra ja existir
            wordsCount{k, 2} = wordsCount{k, 2} + 1; %aumentamos 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end

    end
    if found == false %se a palavra não foi encontrada
        wordsCount{end+1, 1} = words{J}; %adicionar a nova palavra
        wordsCount{end, 2} = 1; %definir o seu contador a 1
    end
end
%
```

Isso criará um cell array com o seguinte aspeto:



```
{["mamamoramamamomamaomoraomoramomaororamomao"   ]}   {[  1]}
{["mamamamaoromoromoromaomoro"                   ]}   {[  1]}
{["raaaaaaaaoromo"                               ]}   {[  1]}
{["ramaaoraa"                                    ]}   {[  2]}
{["aoramaaomoramaoro"                            ]}   {[  1]}
{["omaororomoraorora"                            ]}   {[  1]}
{["rorororomoraa"                                ]}   {[  1]}
{["rororomaoramaaa"                              ]}   {[  1]}
{["maoramoraoramororamaoraa"                     ]}   {[  1]}
{["maomomaaamama"                                ]}   {[  1]}
{["mamaoramaaomororo"                            ]}   {[  1]}
{["aomoromaaaama"                                ]}   {[  1]}
```

Por fim, para extrair os resultados dividimos o número de vezes em que cada palavra foi gerada pelo número total de palavras de forma a retirar a probabilidade estimada desta palavra ser gerada e imprimimos os resultados pedidos

```
fprintf("Foram criadas %d palavras diferentes\n", length(wordsCount))
sortedWords = flip(sortrows(wordsCount,[2])); % ordenar as palavras das mais vezes para menos
for N = 1 : length(sortedWords) %transforma o numero de vezes que a palavra foi gerado pela probabilidade de ser gerada
    sortedWords{N,2} = sortedWords{N,2}/nWords;
end

for N = 1:5
    fprintf('A %dª palavra criada mais vezes foi "%s" com uma probabilidade de %f\n', N, sortedWords{N,1}, sortedWords{N,2})
end
```

Ao correr todo estre programa obtivemos os seguintes resultados:

```
Foram criadas 17243 palavras diferentes
A 1ª palavra criada mais vezes foi "o" com uma probabilidade de 0.083130
A 2ª palavra criada mais vezes foi "a" com uma probabilidade de 0.061770
A 3ª palavra criada mais vezes foi "mo" com uma probabilidade de 0.040930
A 4ª palavra criada mais vezes foi "ro" com uma probabilidade de 0.040130
A 5ª palavra criada mais vezes foi "ra" com uma probabilidade de 0.032290
```

(c) Determine as probabilidades teóricas das 5 palavras apresentadas na questão anterior. Compare os valores teóricos com os valores estimados anteriores. O que conclui?

A probabilidade de cada letra ser a primeira letra da palavra é equivalente para todos as quatro letras possíveis, o que perfaz 0.25 de probabilidade de cada letra ser a primeira da palavra.

- Começando pela palavra com maior probabilidade de ser gerada ‘o’ o seu valor teórico é dado por 0.25 (probabilidade de ‘o’ ser o primeiro carater) *1/3 (probabilidade de transição de ‘o’ para o estado absorvente ‘.’) o que resulta em prob’o’ = 0.25*(1/3) = 0.0833333
- Para ‘a’ temos 0.25 (probabilidade de ‘a’ ser o primeiro carater) * 0.25 (probabilidade de transição de ‘a’ para ‘.’), logo prob’a’=0.25*0.25=0.0625
- Agora no caso de ‘mo’ temos de efetuar os seguintes cálculos: 0.25 (probabilidade de ‘m’ ser o primeiro carater) * 0.5 (probabilidade de transição do ‘m’ para o ‘o’) * 1/3 (probabilidade de transição de ‘o’ para o estado absorvente ‘.’) o que conclui prob’mo’ = 0.25 * 0.5 * (1/3) = 0.041666
- Para a palavra ‘ro’ temos que: 0.25 (probabilidade de ‘r’ ser o primeiro carater) * 0.5 (probabilidade de transição do ‘r’ para o ‘o’) * 1/3 (probabilidade de transição de ‘o’ para o estado absorvente ‘.’). Assim, prob’ro’ = 0.25 * 0.5 * (1/3) = 0.041666
- Por fim, ‘ra’ temos 0.25 (probabilidade de ‘r’ ser o primeiro carater) * 0.5 (probabilidade de transição do ‘r’ para o ‘a’) * 0.25 (probabilidade de transição de ‘a’ para o estado absorvente ‘.’). Assim, prob’ro’ = 0.25 * 0.5 * 0.25 = 0.03125

Os resultados obtidos teoricamente estão praticamente idênticos aos adquiridos por simulação criando cem mil palavras, logo concluímos que os valores estimados obtidos na simulação são muito próximos do valor real

(d) Importe (do arquivo `wordlist-preao-20201103.txt`) a lista de palavras em português para um” cell array”. Com essa lista e os resultados da questão b), estime a probabilidade de o gerador de palavras aleatórias gerar uma palavra válida em português.

Para importar a lista de palavras em português para um cell array recorremos ao documento auxiliar fornecido pelos docentes:

```
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
```

Com o cell array definido resta agora resolver o que é pedido na questão e para isso desenvolvemos o seguinte código:

```
words = {};
count = 0; %contagem de palavras válidas
nWords= 1e5;
for N = 1 : nWords
    if ismember(createWord(T), dicionario) %se a palavra gerada existir no dicionario
        count = count + 1; %acrescentamos 1 ao count
    end
end
%prob = count/nWords
fprintf("Há %.3f de probabilidade de ser gerada uma palavra válida\n", count/nWords)
```

Este programa resulta no seguinte output:

```
Há 0.354370 de probabilidade de ser gerada uma palavra válida
```

Observação: Aquando da realização dos exercícios posteriores verificamos que esta resolução do exercício 1(d) não era a mais eficiente e estava a demorar imenso tempo para nos dar os resultados pretendidos, decidimos então alterar o código para algo semelhante ao demonstrado na questão 1(e), o que melhorou o tempo de execução do programa. Corremos novamente o programa, agora mais eficiente, e obtivemos o seguinte resultado:

```
Há 0.351730 de probabilidade de ser gerada uma palavra válida
```

(e) Altere o gerador de palavras aleatórias para considerar um novo parâmetro de entrada n que representa o tamanho máximo da palavra (em número de letras) das palavras geradas (ou seja, o gerador de palavras para se atingir o estado'.' ou se já tiver gerado n letras)

Acrescentando um novo argumento ('limit') à função usada anteriormente e um 'if' para controlar o tamanho das palavras conseguimos, facilmente, resolver o problema descrito na questão da seguinte forma:

```
function [word] = createWord(T, limit)
    count = 0;
    random = randi(4, 1);
    state = crawl (T, random, 5);
    str="";
    for n = 1 : length(state)
        switch (state(n))
                case 1
                    str = str + "r";
                case 2
                    str = str + "o";
                case 3
                    str = str + "m";
                case 4
                    str = str + "a";
                case 5
                    break
        end
        count = count + 1; %adicionar 1 à contagem de letras da palavra
        if count >= limit %se o numero de letras for >= ao limite terminamos o ciclo
            break;
        end
    end
    word = str;
end
```

(f) Para n= 8,6 e 4, simule a geração de 10^5 palavras aleatórias para estimar o número de palavras diferentes geradas e a probabilidade de uma palavra gerada ser uma palavra válida em português. Compare esses resultados entre eles e com os resultados de 1b) e 1d). O que conclui? Explique assuas conclusões!

Repetindo o raciocínio usado na questão 1d), mas agora com o novo argumento na função createWord, temos de criar as palavras de maneira diferente, mas o restante raciocínio mantém-se:

```
maxLength = 4;
% ciclo for para criar nWords(10^5) palavras
for N = 1 : nWords
    words{end+1} = createWord(T, maxLength);
end
%
```

Para obter os resultados pedidos na questão tivemos, também de alterar a forma como os outputs serão visualizados e recorrer ao raciocínio da alínea 1b para calcular o número de palavras diferentes:

```
% inicializar o novo cell array onde 1ª coluna = palavra
% 2ª coluna = numero devezes que foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;
%

%ciclo for a iniciar no 2º elemento (pois o primeiro já foi gerado)
for J = 2 : length(words) %este ciclo percorre todas as palavras geradas anteriormente
    found = false; % variavel que informa se a palavra foi ou nao encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra ja existir
            wordsCount{k, 2} = wordsCount{k, 2} + 1; %aumentamos 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end

    end
    if found == false %se a palavra não foi encontrada
        wordsCount{end+1, 1} = words{J}; %adicionar a nova palavra
        wordsCount{end, 2} = 1; %definir o seu contador a 1
    end
end
%
fprintf("Para n = %d : \n", maxLength);
fprintf("Foram criadas %d palavras diferentes\n", length(wordsCount))
%
```

```
%Verificar se as palavras geradas são válidas
for N = 1 : length(wordsCount)
    if ismember(wordsCount{N,1}, dicionario)
        count = count + wordsCount{N,2}; % se forem acrescentamos o numero de vezes que a palavra
                                         % foi gerada ao contador de palavra válidas
    end
end
fprintf("Há %f de probabilidade de ser gerada uma palavra válida\n", count/nWords)
```

O output do programa foi o seguinte:

```
Para n = 4 :
Foram criadas 61 palavras diferentes
Há 0.501690 de probabilidade de ser gerada uma palavra válida

Para n = 6 :
Foram criadas 307 palavras diferentes
Há 0.365960 de probabilidade de ser gerada uma palavra válida

Para n = 8 :
Foram criadas 1515 palavras diferentes
Há 0.354980 de probabilidade de ser gerada uma palavra válida
```

2. Altere a matriz de transição de estados T assumindo as probabilidades de transição definidas no diagrama dado. Para n = ∞, 8, 6 e 4, simule a geração de 10^5 palavras aleatórias para estimar o número de palavras geradas e a probabilidade de uma palavra gerada ser uma palavra valida em português.

Com a ajuda do seguinte código:

```
      %r      %o      %m      %a      %.
T =  [0      0.3     0       0.3     0  %r
      0.3    0       0.3     0.1     0  %o
      0      0.2     0       0.2     0  %m
      0.7    0       0.7     0       0  %a
      0      0.5     0       0.4     0] %.

% % colunas- r-o-m-a-. ;
% linhas - r-o-m-a-. ;
```

É pedido para gerar 10^5 palavras aleatórias, sendo mudado o tamanho máximo das palavras (n) entre infinito, 8, 6 e 4 e a probabilidade de uma palavra gerada ser uma palavra valida em português.

- Para n = 4:

```
count = 0;
words = {}; % criação do cell array
% Para n= 4
for i = 1: 1e5 %    Ciclo for para criar 10^5 palavras atraves da função crawl
    string = "";
    aleatorio = randi(4); % O primeiro número é aleatório, logo fazer random de nums entre 1 e 4
    state = crawl(T, aleatorio, 5);

    for n = 1 : length(state) %fazer um ciclo for para percorrer os numeros do state e transformá-los em caractéres
        switch state(n)
           case 1
            string = string + "r";
           case 2
            string = string + "o";
           case 3
            string = string + "m";
           case 4
            string = string + "a";
        end
        count = count + 1; % Count aumenta um sempre que é adicionado um caracter à string
        if(count == 4) % Quando o count é igual a 4,8, .6,infinito, siginifica que alcançamos o tamanho máximo da palavra
            count = 0; % Reset no count para assim ver restringir o tamanho max da próxima palavra
            break; % Passa para o próximo state
        end
    end
    fprintf('%s\n',string) %print na palavra

    words{end+1} = string; %Guardar cada palavra no cell array words
end
```

Começamos por criar conjuntos de números aleatórios, um a um, que começam entre 1 e 4 e acabam no número 5, sendo criados 10^5 conjuntos. De seguida vamos a cada um desses conjuntos de números e transformamos cada um dos números em letras (r, o, m, a).

O count dá-nos o tamanho da palavra. Como queremos que o tamanho máximo de uma palavra seja até 4 então quando o count for igual a 4, imprimo a palavra, e passamos para o próximo conjunto de números. Cada palavra gerada é guardada no cell array “words”.

De seguida, é pedido o número de palavras diferentes geradas e a probabilidade de uma palavra gerada ser uma palavra válida.

1) Número de palavras diferentes geradas:

```
wordsCount = {};
%Incializamos o novo cell array onde a 1ª coluna é igual a palavra
% A 2ª coluna é o numero de vezes que a respetiva palavra foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;

%ciclo for a iniciar no 2º elemento ( pois o primeiro já foi gerado)
for J = 2 : length(words) % Este ciclo percorer todas as palavras geradas anteriormente
    found = false; %variável que informa se a palavra foi ou não encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %Percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra já existe
            wordsCount{k, 2} = wordsCount{k, 2} + 1; % aumenta 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end
    end
    if found == false % se a apalvra nao foi encontrada
        wordsCount{end+1, 1} = words{J}; % adicionar a nova palavra
        wordsCount{end, 2} = 1; % definir o contador a 1
    end
end
  fprintf("São geradas %d palavras diferentes\n",length(wordsCount))
```

2) Probabilidade de uma palavra gerada ser uma palavra válida:

```
%% import list of words to a cell array 'dicionario':
clc
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
dicionario{100};
dicionario{Nwords};

clc
count = 0;
   for i = 1: 1e5 % Ciclo for a percorrer todas as minhas palavras
       ismember(words{i},dicionario); % Retorna um boolean. Caso seja Verdadeiro/1, a palavra é uma palavra válida em português.
       if(ismember(words{i},dicionario) == true) % se a minha palavra for válida entao contador aumenta 1
           count = count + 1; % Dá o número de palavras válidas em português
       end
   end

  prob = count/length(words) %probabilidade de uma palavra ser válida é o
  % número total de palavras válidas a dividir pelo número total de
  % palavras armazenadas no meu cell array words
```

Então, percorremos o cell array que contém todas as palavras. Começamos por ver se esta pertence ao dicionário (cell array que contém todas as palavras válidas em português) ou não, fazendo isto para as outras todas. Caso fosse válida, o contador, que começa a 0, aumenta 1.

Assim, o contador dá-nos o número de palavras geradas que são palavras válidas em português.

- Para n = 6:

```
count = 0;
words = {}; % criação do cell array
% Para n= 6
for i = 1: 1e5 %    Ciclo for para criar 10^5 palavras atraves da função crawl
    string = "";
    aleatorio = randi(4); % O primeiro número é aleatório, logo fazer random de nums entre 1 e 4
    state = crawl(T, aleatorio, 5);

    for n = 1 : length(state) %fazer um ciclo for para percorrer os numeros do state e transformá-los em caractéres
        switch state(n)
           case 1
            string = string + "r";
           case 2
            string = string + "o";
           case 3
            string = string + "m";
           case 4
            string = string + "a";
        end
        count = count + 1; % Count aumenta um sempre que é adicionado um caracter à string
        if(count ==6) % Quando o count é igual a 4,8, 6,infinito, siginifica que alcançamos o tamanho máximo da palavra
            count = 0; % Reset no count para assim ver restringir o tamanho max da próxima palavra
            break; % Passa para o próximo state
        end
    end
    fprintf('%s\n',string) %print na palavra

    words{end+1} = string; %Guardar cada palavra no cell array words
end
%Inicializamos o novo cell array onde a 1ª coluna é igual a palavra
% A 2ª coluna é o numero de vezes que a respetiva palavra foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;

%ciclo for a iniciar no 2º elemento ( pois o primeiro já foi gerado)
for J = 2 : length(words) % Este ciclo percorer todas as palavras geradas anteriormente
    found = false; %variável que informa se a palavra foi ou não encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %Percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra já existe
            wordsCount{k, 2} = wordsCount{k, 2} + 1; % aumenta 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end
    end
    if found == false % se a apalvra nao foi encontrada
        wordsCount{end+1, 1} = words{J}; % adicionar a nova palavra
        wordsCount{end, 2} = 1; % definir o contador a 1
    end
end
  fprintf("São geradas %d palavras diferentes\n",length(wordsCount))
```

2. Probabilidade de uma palavra gerada ser uma palavra válida:

```
%% import list of words to a cell array 'dicionario':
clc
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
dicionario{100};
dicionario{Nwords};

clc
count = 0;
   for i = 1: 1e5 % Ciclo for a percorrer todas as minhas palavras
       ismember(words{i},dicionario); % Retorna um boolean. Caso seja Verdadeiro/1, a palavra é uma palavra válida em português.
       if(ismember(words{i},dicionario) == true) % se a minha palavra for válida entao contador aumenta 1
           count = count + 1; % Dá o número de palavras válidas em português
       end
   end

  prob = count/length(words) %probabilidade de uma palavra ser válida é o
  % número total de palavras válidas a dividir pelo número total de
  % palavras armazenadas no meu cell array words
```

- Para n = 8:

```
count = 0;
words = {}; % criação do cell array
% Para n= 8
for i = 1: 1e5 %    Ciclo for para criar 10^5 palavras atraves da função crawl
    string = "";
    aleatorio = randi(4); % O primeiro número é aleatório, logo fazer random de nums entre 1 e 4
    state = crawl(T, aleatorio, 5);

    for n = 1 : length(state) %fazer um ciclo for para percorrer os numeros do state e transformá-los em caractéres
        switch state(n)
           case 1
            string = string + "r";
           case 2
            string = string + "o";
           case 3
            string = string + "m";
           case 4
            string = string + "a";
        end
        count = count + 1; % Count aumenta um sempre que é adicionado um caracter à string
        if(count == 8) % Quando o count é igual a 4,8, 6,infinito, siginifica que alcançamos o tamanho máximo da palavra
            count = 0; % Reset no count para assim ver restringir o tamanho max da próxima palavra
            break; % Passa para o próximo state
        end
    end
    fprintf('%s\n',string) %print na palavra

    words{end+1} = string; %Guardar cada palavra no cell array words
end
```

1) Número de palavras diferentes geradas:

```
wordsCount = {};
%Incializamos o novo cell array onde a 1ª coluna é igual a palavra
% A 2ª coluna é o numero de vezes que a respetiva palavra foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;

%ciclo for a iniciar no 2º elemento ( pois o primeiro já foi gerado)
for J = 2 : length(words) % Este ciclo percorer todas as palavras geradas anteriormente
    found = false; %variável que informa se a palavra foi ou não encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %Percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra já existe
            wordsCount{k, 2} = wordsCount{k, 2} + 1; % aumenta 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end
    end
    if found == false % se a apalvra nao foi encontrada
        wordsCount{end+1, 1} = words{J}; % adicionar a nova palavra
        wordsCount{end, 2} = 1; % definir o contador a 1
    end
end
  fprintf("São geradas %d palavras diferentes\n",length(wordsCount))
```

2) Probabilidade de uma palavra gerada ser uma palavra válida:

```
%% import list of words to a cell array 'dicionario':
clc
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
dicionario{100};
dicionario{Nwords};

clc
count = 0;
   for i = 1: 1e5 % Ciclo for a percorrer todas as minhas palavras
       ismember(words{i},dicionario); % Retorna um boolean. Caso seja Verdadeiro/1, a palavra é uma palavra válida em português.
       if(ismember(words{i},dicionario) == true) % se a minha palavra for válida entao contador aumenta 1
           count = count + 1; % Dá o número de palavras válidas em português
       end
   end

  prob = count/length(words) %probabilidade de uma palavra ser válida é o
  % número total de palavras válidas a dividir pelo número total de
  % palavras armazenadas no meu cell array words
```

- Para n = infinito:

```
words = {}; % criação do cell array

for i = 1: 1e5 % Ciclo for para criar 10^5 palavras atraves da função crawl
    string = "";
    aleatorio = randi(4); % O primeiro número é aleatório, logo fazer random de nums entre 1 e 4
    state = crawl(T, aleatorio, 5);

    for n = 1 : length(state) %fazer um ciclo for para percorrer os numeros do state e transformá-los em caractéres
        switch state(n)
           case 1
            string = string + "r";
           case 2
            string = string + "o";
           case 3
            string = string + "m";
           case 4
            string = string + "a";
        end
    end
    fprintf('%s\n',string) %print na palavra

    words{end+1} = string; %Guardar cada palavra no cell array words
end
```

1) Número de palavras diferentes geradas:

```
wordsCount = {};
%Incializamos o novo cell array onde a 1ª coluna é igual a palavra
% A 2ª coluna é o numero de vezes que a respetiva palavra foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;

%ciclo for a iniciar no 2º elemento ( pois o primeiro já foi gerado)
for J = 2 : length(words) % Este ciclo percorer todas as palavras geradas anteriormente
    found = false; %variável que informa se a palavra foi ou não encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %Percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra já existe
            wordsCount{k, 2} = wordsCount{k, 2} + 1; % aumenta 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end
    end
    if found == false % se a apalvra nao foi encontrada
        wordsCount{end+1, 1} = words{J}; % adicionar a nova palavra
        wordsCount{end, 2} = 1; % definir o contador a 1
    end
end
  fprintf("São geradas %d palavras diferentes\n",length(wordsCount))
```

2) Probabilidade de uma palavra gerada ser uma palavra válida:

```
%% import list of words to a cell array 'dicionario':
clc
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
dicionario{100};
dicionario{Nwords};

clc
count = 0;
   for i = 1: 1e5 % Ciclo for a percorrer todas as minhas palavras
       ismember(words{i},dicionario); % Retorna um boolean. Caso seja Verdadeiro/1, a palavra é uma palavra válida em português.
       if(ismember(words{i},dicionario) == true) % se a minha palavra for válida entao contador aumenta 1
           count = count + 1; % Dá o número de palavras válidas em português
       end
   end

  prob = count/length(words) %probabilidade de uma palavra ser válida é o
  % número total de palavras válidas a dividir pelo número total de
  % palavras armazenadas no meu cell array words
```

O output deste programa foi o seguinte:

```
Para n = 4 :
Foram criadas 61 palavras diferentes
Há 0.651840 de probabilidade de ser gerada uma palavra válida

Para n = 6 :
Foram criadas 307 palavras diferentes
Há 0.537930 de probabilidade de ser gerada uma palavra válida

Para n = 8 :
Foram criadas 1470 palavras diferentes
Há 0.521530 de probabilidade de ser gerada uma palavra válida

Para n = infinito:
Foram criadas 7353 palavras diferentes
Há 0.518060 de probabilidade de ser gerada uma palavra válida
```

Após análise do código utilizado para a questão verificamos que o mesmo demorava muito tempo a rodar e decidimos mudar o código para algo mais rápido, onde a principal diferença é que invés de verificar se cada palavra gerada se encontra na lista de 1 milhão de palavras válidas portuguesas (“dicionário”) verifica apenas no cell array das 88 palavras portuguesas válidas que são formadas apenas pelas 4 letras que podemos usar. Estávamos a verificar 1 milhão de palavras desnecessariamente, visto que apenas existem 88 palavras possíveis. Usamos então o seguinte código alternativo para obter os resultados

```
words = {};
wordsCount = {};
count = 0;
specWords= {};
nWords= 1e5;
parar = 0;
set_of_letters= 'amor';
for n = 1 : Nwords
    if min(ismember(dicionario{n},set_of_letters))
        specWords{end+1} = dicionario{n}; %palavra formado por 'a', 'm', 'o', 'r'
    end
end
maxLength = 8;
for N = 1 : nWords
    words{end+1} = createWord(T,maxLength); %lista de todas as palavras geradas
end
```

```
% inicializar o novo cell array onde 1ª coluna = palavra
% 2ª coluna = numero devezes que foi gerada
wordsCount{end+1, 1} = words{1};
wordsCount{end, 2} = 1;
%

%ciclo for a iniciar no 2º elemento (pois o primeiro já foi gerado)
for J = 2 : length(words) %este ciclo percorre todas as palavras geradas anteriormente
    found = false; % variavel que informa se a palavra foi ou nao encontrada no novo cell array
    for k = 1 : length(wordsCount(:, 1)) %percorrer as palavras que já estão no novo cell array
        if words{J} == wordsCount{k,1}  %se a nova palavra ja existir
            wordsCount{k, 2} = wordsCount{k, 2} + 1; %aumentamos 1 vez
            found = true; %definir que foi encontrada a palavra
            break
        end

    end
    if found == false %se a palavra não foi encontrada
        wordsCount{end+1, 1} = words{J}; %adicionar a nova palavra
        wordsCount{end, 2} = 1; %definir o seu contador a 1
    end
end
%
fprintf("Para n = %d : \n", maxLength);
fprintf("Foram criadas %d palavras diferentes\n", length(wordsCount))
%
%Verificar se as palavras geradas são válidas
for N = 1 : length(wordsCount)
    if ismember(wordsCount{N,1}, specWords)
        count = count + wordsCount{N,2}; % se forem acrescentamos o numero de vezes que esta palavra foi gerada
                                         % ao contador de palavra válidas
    end
end
fprintf("Há %f de probabilidade de ser gerada uma palavra válida\n", count/nWords)
```

3. Considere a matriz de transição de estados T da questão 2. Usando as palavras do ficheiro wordlist-preao-20201103.txt que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r’, estime a probabilidade de cada letra ser a primeira letra da palavra. Repita a questão 2 assumindo agora estas novas probabilidades para a primeira letra.

Utilizando a matriz de transição de estados T:

```
      %r      %o      %m      %a      %.
T =  [0       0.3     0       0.3     0   %r
      0.3     0       0.3     0.1     0   %o
      0       0.2     0       0.2     0   %m
      0.7     0       0.7     0       0   %a
      0       0.5     0       0.4     0] %.

% % colunas- r-o-m-a-. ;
% linhas - r-o-m-a-. ;
```

Para realizar este exercício recorremos ao Matlab e resolvemo-lo da seguinte forma:

```
clc
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
dicionario{100};
dicionario{Nwords};

%determine if a word has only letters of a given set of letters
clc
set_of_letters= 'amor'; % Array de caracteres, que nos dá quais os caracteres possíveis numa palavra
lista = {}; %criacao de um cell array, onde vão ser guardadas as palavras que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r’


for(i = 1: length(dicionario))

    if(min(ismember(dicionario{i},set_of_letters)) == 1) %returns true
       lista{end+1} = dicionario{i};
    end
end

lista %Lista de palavras que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r’
```

Começamos por importar o ficheiro com as palavras para um cell array dicionário.

De seguida, definimos um array de caracteres (set_of_letters) que nos dá os caracteres possíveis numa palavra e um cell array lista. Percorremos todas as palavras do dicionário e vemos se as palavras apenas contêm as letras do set_of_letters ou não. Caso contenham apenas as letras em cima referidas, então é V (ou 1), caso contrário é F (ou 0). Quando o valor é 1, então adiciono essa palavra à lista.

Assim, a lista é constituída por todas as palavras que apenas contêm as letras ‘a’, ‘m’, ‘o’ e ‘r’.

A seguir, calculamos a probabilidade de cada letra ser a primeira letra da palavra. Para isso, criamos um contador para cada letra, todos iguais a 0.

Vamos a cada palavra da lista e vemos por que letra começa. De acordo com a letra que começar, adicionamos ao seu contador 1. Por exemplo, caso comece com a letra r, então o countR passa a 1. O contador serve para ver quantas palavras começam com uma certa letra.

Para obter a probabilidade de cada letra ser a primeira letra da palavra fizemos 4 probabilidades, cada probabilidade para cada letra (a, m, o, r) e basta dividir o contador dessa letra pelo número total de palavras.

```
countA = 0; %countA serve para countar as palavras que começam com o caracter 'a'
countO = 0; %countO serve para countar as palavras que começam com o caracter 'o'
countM = 0; %countM serve para countar as palavras que começam com o caracter 'm'
countR = 0; %countR serve para countar as palavras que começam com o caracter 'r'

   for(i = 1: length(lista)) %ciclo for a percorrer as palavras que apenas contenham as letras 'a', 'm', 'o' e 'r
       palavra = lista{i}; %criação de uma variável palavra para igualar a cada uma das palavras da minha lista
      if(palavra(1) == 'r')   %palavra(1) dá-nos o caracter contido na 1º posição de cada palavra
        countR = countR + 1;  % Caso a minha palavra começe com r entao vou adicionar 1 ao meu contador de palavras começadas com r
      elseif ( palavra(1) == 'a')
         countA = countA + 1;
      elseif ( palavra(1) == 'm')
        countM = countM + 1;
      else
       countO = countO + 1;
      end
   end

   % A probabilidade de cada letra ser a primeira letra da palavra.
   %Por exemplo a probabilidade da palavra começar com r é a soma de todas
   %as palavras que começam com r a dividir pelo número total de palavras,
   %neste caso, length(lista)

    probA = countA/length(lista)
    probO = countO/length(lista)
    probM = countM/length(lista)
    probR = countR/length(lista)
```

O output do código acima é o seguinte:

```
probA = 0.5114
probO = 0.0795
probM = 0.3295
probR = 0.0795
```

O exercício pede também para fazer a questão 2, assumindo estas novas probabilidades.

Tanto o número de palavras diferentes geradas como a probabilidade são feitas de igual forma como no exercício 2, em cima resolvido. O que vai alterar são os seus valores, sendo estes referidos em baixo.

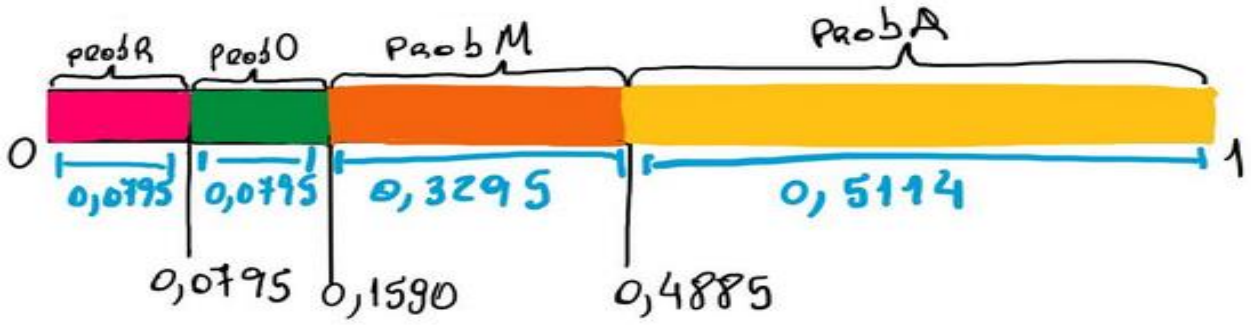
A diferença está na probabilidade da primeira letra de cada palavra, sendo o código em baixo a resolução para o mesmo.

```
count = 0;
words = {}; % criação do meu cell array

for i = 1: 1e5
    string = "";
    random = rand(1);
    initialState = 0;
    if random <= 0.0795
        initialState = 1; %r
    elseif (random > 0.0795 && random <= 0.1590)
        initialState = 2; %o
    elseif (random > 0.1590 && random <= 0.4885)
        initialState = 3; %m
    else
        initialState = 4; %a
    end
    state = crawl (T,initialState, 5);

    for n = 1 : length(state) %fazer um ciclo for para percorrer os caracteres de cada palavra
        switch state(n)
           case 1
            string = string + "r";
           case 2
            string = string + "o";
           case 3
            string = string + "m";
           case 4
            string = string + "a";
        end
        count = count + 1; % Count aumenta um sempre que é lido uma parcela da palavra
        if(count == 4) %Quando o count é igual a 4,6,8,infinito, siginifica que alcançamos o tamanho máximo da palavra
            count = 0; %Reset no count para a proxima palavra
            break;
        end
    end
    fprintf('%s\n',string) %print na palavra
    words{end+1} = string; %Guardar cada palavra no cell array words
end
```

Para a realização do algoritmo de escolha do "initialState" feito acima pensamos nas probabilidades como sendo uma geração aleatória de um número entre 0 e 1. Definimos intervalos específicos para cada letra, onde a sua dimensão corresponde à probabilidade da letra em questão ser a primeira da palavra. Assim, gerando um número entre 0 e 1 e verificando em que intervalo o mesmo se encontra é possível escolher que letra será a primeira tendo em conta as suas probabilidades. Como podemos verificar no esquema seguinte:



O output deste programa foi o seguinte:

```
Para n = 4 :
Foram criadas 61 palavras diferentes
Há 0.743960 de probabilidade de ser gerada uma palavra válida

Para n = 6 :
Foram criadas 306 palavras diferentes
Há 0.646830 de probabilidade de ser gerada uma palavra válida

Para n = 8 :
Foram criadas 1417 palavras diferentes
Há 0.621930 de probabilidade de ser gerada uma palavra válida

Para n = infinito:
Foram criadas 6892 palavras diferentes
Há 0.620060 de probabilidade de ser gerada uma palavra válida
```

Concluímos que tanto na questão 2 como na questão 3, o número de palavras diferentes geradas aumenta proporcionalmente com o valor de n (tamanho máximo de cada palavra). Por sua vez, a probabilidade desta palavra gerada ser válida diminui. Verificou-se que o gerador da questão 3 é o mais eficiente, visto que a probabilidade de uma palavra gerada ser válida é maior. Assim, concluímos que é mais eficiente se ao gerar uma palavra tivermos em conta a probabilidade de cada letra (‘a’, ‘m’, ‘o’, ‘r’) para o primeiro caráter de cada palavra.

4. Repita a questão 3 para este caso e analise os resultados obtidos. Compare-os entre si e com os resultados anteriores. O que conclui sobre a eficiência destes geradores de palavras aleatórias quando comparados com os geradores anteriores? Explique as suas conclusões!

A matriz transição de estados T é:

```
T = [0    0.3  0    0.3  0
     0.3  0    0.3  0.1  0
     0.1  0.2  0    0.2  0
     0.6  0    0.7  0    0
     0    0.5  0    0.4  0]   % colunas- r-o-m-a-.
                               % linhas- r-o-m-a-.

%Matriz transição de estados
```

Ao repetir o exercício 3 e utilizando a matriz T:

```
clc
fid= fopen('wordlist-preao-20201103.txt','r');
dicionario= textscan(fid,'%s');
fclose(fid);
dicionario= dicionario{1,1};
Nwords= length(dicionario);
dicionario{100};
dicionario{Nwords};

%determine if a word has only letters of a given set of letters
clc
set_of_letters= 'amor'; % Array de caracteres, que nos dá quais os caracteres possíveis numa palavra
lista = {}; %criacao de um cell array, onde vão ser guardadas as palavras que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r’

for(i = 1: length(dicionario))

    if(min(ismember(dicionario{i},set_of_letters)) == 1) %returns true
       lista{end+1} = dicionario{i};
    end
end

lista %Lista de palavras que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r’

countA = 0; %countA serve para countar as palavras que começam com o caracter 'a'
countO = 0; %countO serve para countar as palavras que começam com o caracter 'o'
countM = 0; %countM serve para countar as palavras que começam com o caracter 'm'
countR = 0; %countR serve para countar as palavras que começam com o caracter 'r'
```

```
for(i = 1: length(lista)) %ciclo for a percorrer as palavras que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r
    palavra = lista{i}; %criação de uma variável palavra para igualar a cada uma das palavras da minha lista
   if(palavra(1) == 'r')   %palavra(1) dá-nos o caracter contido na 1º posição de cada palavra
     countR = countR + 1;  % Caso a minha palavra começe com r entao vou adicionar 1 ao meu contador de palavras começadas com r
   elseif ( palavra(1) == 'a')
      countA = countA + 1;
   elseif ( palavra(1) == 'm')
     countM = countM + 1;
   else
    countO = countO + 1;
   end
end

% A probabilidade de cada letra ser a primeira letra da palavra.
%Por exemplo a probabilidade da palavra começar com r é a soma de todas
%as palavras que começam com r a dividir pelo número total de palavras,
%neste caso, length(lista)

 probA = countA/length(lista)
 probO = countO/length(lista)
 probM = countM/length(lista)
 probR = countR/length(lista)
```

O output deste código foi o seguinte:

```
probA = 0.5114
probO = 0.0795
probM = 0.3295
probR = 0.0795
```

Na resolução do exercício 3, é também pedido para repetir o exercício 2. Uma vez que a única diferença ao fazer o exercício 2 é a matriz de transição de estados, sendo o resto do código todo igual,

```
for i = 1: 1e5
    string = "";
    random = rand(1);
    initialState = 0;
    if random <= 0.0795
        initialState = 1; %r
    elseif (random > 0.0795 && random <= 0.1590)
        initialState = 2; %o
    elseif (random > 0.1590 && random <= 0.4885)
        initialState = 3; %m
    else
        initialState = 4; %a
    end
    state = crawl (T,initialState, 5);

    for n = 1 : length(state) %fazer um ciclo for para percorrer os caracteres de cada palavra
        switch state(n)
           case 1
            string = string + "r";
           case 2
```

Então, os resultados obtidos são:

```
Para n = 4 :
Foram criadas 77 palavras diferentes
Há 0.736160 de probabilidade de ser gerada uma palavra válida

Para n = 6 :
Foram criadas 456 palavras diferentes
Há 0.632530 de probabilidade de ser gerada uma palavra válida

Para n = 8 :
Foram criadas 2263 palavras diferentes
Há 0.611430 de probabilidade de ser gerada uma palavra válida

Foram criadas 8665 palavras diferentes
Há 0.607560 de probabilidade de ser gerada uma palavra válida
```

Concluímos que na questão 3 e 4 os valores obtidos são semelhantes. Contudo, com o aumento do tamanho máximo da palavra verifica-se que a probabilidade na questão 4 diminui com maior intensidade, enquanto na questão 3 os valores são mais constantes. Assim, verificamos que como o gerador obtido no exercício 3 apresenta uma maior probabilidade de ser gerada uma palavra válida portuguesa, este é ligeiramente mais eficiente.

5. Usando as palavras do ficheiro `wordlist-preao-20201103.txt` que apenas contenham as letras ‘a’, ‘m’, ‘o’ e ‘r’, estime as probabilidades de transição de estados da matriz T baseado nas sequencias de 2 letras que aparecem nessas palavras (e nas´ultimas letras para estimar a transição para o estado ’.’). Com esta matriz T estimada, repita questão 3.

Para este exercício é necessário guardar várias variáveis para conseguir calcular as probabilidades de todas as transições:

```
%inicializar os counts necessários
countTotal = 0;

countRr = 0;
countRo = 0;
countRm = 0;
countRa = 0;

countOr = 0;
countOo = 0;
countOm = 0;
countOa = 0;

countMr = 0;
countMo = 0;
countMm = 0;
countMa = 0;

countAr = 0;
countAo = 0;
countAm = 0;
countAa = 0;

countRdot = 0;
countOdot = 0;
countMdot = 0;
countAdot = 0;
%
```

Sendo “countRr” o número de vezes que um ‘r’ aparece depois de um ‘r’, e o mesmo raciocínio para as restantes. A countTotal refere-se ao número total de transições.

De seguida criamos um ciclo para percorrer todas as palavras geradas com outro ciclo para verificar cada carater desta mesma palavra, da seguinte forma:

specWords refere-se ao cell array de palavras específicas, geradas apenas com os quatro carateres ‘r, o, m, a’.

```
for n = 1 : length(specWords)
    palavra = specWords{n};
    countTotal = countTotal + 1; %uma nova transiçao vai ser feita, logo count + 1
    for j = 1 : length(palavra) %para percorrer todos os carateres da palavra
        if j == length(palavra) %significa que estamos na última letra e a transiçao será para o estado '.'
            if palavra(j) == 'r'
                countRdot = countRdot + 1;
            elseif palavra(j) == 'o'
                countOdot = countOdot + 1;
            elseif palavra(j) == 'm'
                countMdot = countMdot + 1;
            elseif palavra(j) == 'a'
                countAdot = countAdot + 1;
            end

            break;
        end

        if palavra(j) == 'r' % se o carater for o 'r' verificar o seguinte
            if palavra(j+1) == 'r'
                countRr = countRr + 1;
            elseif palavra(j+1) == 'o'
                countRo = countRo + 1;
            elseif palavra(j+1) == 'm'
                countRm = countRm + 1;
            elseif palavra(j+1) == 'a'
                countRa = countRa + 1;
            end

        elseif palavra(j) == 'o' % se o carater for o 'o' verificar o seguinte
            if palavra(j+1) == 'r'
                countOr = countOr + 1;
            elseif palavra(j+1) == 'o'
                countOo = countOo + 1;
            elseif palavra(j+1) == 'm'
                countOm = countOm + 1;
            elseif palavra(j+1) == 'a'
                countOa = countOa + 1;
            end


         elseif palavra(j) == 'm' % se o carater for o 'm' verificar o seguinte
             if palavra(j+1) == 'r'
                 countMr = countMr + 1;
             elseif palavra(j+1) == 'o'
                 countMo = countMo + 1;
             elseif palavra(j+1) == 'm'
                 countMm = countMm + 1;
             elseif palavra(j+1) == 'a'
                 countMa = countMa + 1;
             end

         elseif palavra(j) == 'a' % se o carater for o 'a' verificar o seguinte
             if palavra(j+1) == 'r'
                 countAr = countAr + 1;
             elseif palavra(j+1) == 'o'
                 countAo = countAo + 1;
             elseif palavra(j+1) == 'm'
                 countAm = countAm + 1;
             elseif palavra(j+1) == 'a'
                 countAa = countAa + 1;
             end
         end
    end
end
```

Com os valores retirados após correr estes ciclos é nos possível criar uma matriz com as probabilidades de cada transição:

```
%matriz não estocástica
T = [countRr/countTotal     countOr/countTotal      countMr/countTotal      countAr/countTotal      0 ;
     countRo/countTotal     countOo/countTotal      countMo/countTotal      countAo/countTotal      0 ;
     countRm/countTotal     countOm/countTotal      countMm/countTotal      countAm/countTotal      0 ;
     countRa/countTotal     countOa/countTotal      countMa/countTotal      countAa/countTotal      0 ;
     countRdot/countTotal   countOdot/countTotal    countMdot/countTotal    countAdot/countTotal    0 ;
    ]
```

```
T =

    0.1705    0.2045         0    0.8750         0
    0.1932    0.0227    0.2500    0.0114         0
    0.0682    0.0682    0.0114    0.6364         0
    0.7273    0.0455    0.5682         0         0
    0.1705    0.2159    0.2841    0.3295         0
```

Como esta matriz T é não estocástica é necessário efetuar alterações, logo fizemos o seguinte:

```
%transformar T em matriz estocástica
for n = 1 : length(T)-1
   T(:,n)= T(:,n)/sum(T(:,n));
end
%
```

```
T =

    0.1282    0.3673         0    0.4724         0
    0.1453    0.0408    0.2245    0.0061         0
    0.0513    0.1224    0.0102    0.3436         0
    0.5470    0.0816    0.5102         0         0
    0.1282    0.3878    0.2551    0.1779         0
```

Agora que a matriz T é estocástica podemos repetir a questão 3, agora com esta nova matriz. Onde obtivemos os seguintes resultados:

```
Para n = 4 :
Foram criadas 218 palavras diferentes
Há 0.609410 de probabilidade de ser gerada uma palavra válida
```

```
Para n = 6 :
Foram criadas 1465 palavras diferentes
Há 0.470960 de probabilidade de ser gerada uma palavra válida
```

```
Para n = 8 :
Foram criadas 4866 palavras diferentes
Há 0.421840 de probabilidade de ser gerada uma palavra válida
```

De acordo com os resultados obtidos, concluímos que este último gerador não é tão eficiente como o gerado em questões anteriores, como por exemplo na questão 4 e na questão 3.

Por fim, e analisando os resultados obtidos em todas as questões o gerador que, pelos nossos cálculos, se revelou mais eficiente foi o gerador da questão 3.