Métodos Probabilísticos para Engenharia Informática
Ano Letivo: 2020/2021
Turma P4
Tomé Lopes Carvalho - 97939
Gonçalo Fernandes Machado - 98359

Relatório PL3: Cadeias de Markov

1. a)

```
T = [0   1/3 0   1/4 0
     1/2 0   1/2 1/4 0
     0   1/3 0   1/4 0
     1/2 0   1/2 0   0
     0   1/3 0   1/4 1];

first = randi(4);
str = crawl(T, first, 5);
```

Na matriz de transição de estados, o elemento *T*(*i*, *j*) representa a probabilidade de passar do estado *j* para o estado *i*.
Como a probabilidade de transição de cada estado é a mesma para todos os estados possíveis seguintes, *T*(*i*, *j*) vai ser igual ao inverso do número de opções que podem ser escolhidas após *j*.
Considerámos "." um estado absorvente, porém utilizámos este estado apenas como indicador de fim da palavra.
A função utilizada para gerar as palavras (*crawl*) tem como argumentos a matriz de transição, o estado inicial (gerado através da função *randi*, que, neste caso, gera um número inteiro em [1; 4] com probabilidades uniformemente distribuídas) e o estado final ("."), pelo que a probabilidade de transição do "." para qualquer outro estado é irrelevante.

```
function str = crawl(H, first, last)
    % the sequence of states will be saved in the vector "state"
    % initially, the vector contains only the initial state:
    state = [first];
    % keep moving from state to state until state "last" is reached:
    while (1)
        state(end+1) = nextState(H, state(end));
        if (state(end) == last)
            break;
        end
    end
    base = 'roma.';               %Cada letra corresponde a um estado
    str_com_ponto = base(state);  %Converte os estados em letras
    str = str_com_ponto(1:end-1); %Remove o ponto (último char)
End
```

```
function state = nextState(H, currentState)
% find the probabilities of reaching all states starting at the current one:
    probVector = H(:,currentState)'; % probVector is a row vector
    n = length(probVector); %n is the number of states
    % generate the next state randomly according to probabilities probVector:
    state = discrete_rnd(1:n, probVector);
end

function state = discrete_rnd(states, probVector)
% Generate randomly the next state.
% Inputs:
% states = vector with state values
% probVector = probability vector
    U=rand();
    i = 1 + sum(U > cumsum(probVector));
    state= states(i);
end
```

De modo a substituir o valor de retorno por uma palavra (ao invés de um vetor de estados, como *state*), utilizámos o método exemplificado em *explanation_PL03.m* .
Por fim, removemos o último caractere (“.”), para ser devolvida a palavra sem o seu indicador de término.
As funções *nextState* e *discrete_rnd* são as funções disponíveis no anexo do enunciado.

1. b)

```
N = 1e5;                                %Número de experiências
palavras = containers.Map();              %Mapa que associa a cada palavra o número
                                        %de vezes que foi gerada
for n=1:N
    first = randi(4);                   %Gerar o estado inicial
    str = crawl(T, first, 5);           %Gerar uma palavra
    if isKey(palavras, str)             %Se for chave do mapa
        palavras(str) = palavras(str)+1;  %Incrementar o valor por 1
    else                                %Se não,
        palavras(str) = 1;              %o valor é igual a 1
    end
end

n_palavras_dif = size(palavras, 1); %Nº de palavras diferentes
keys = keys(palavras);              %Lista de palavras geradas
top5 = [0 0 0 0 0]; %Matriz onde serão guardadas as probabilidades de serem geradas cada
                    %palavra nas 5 mais geradas (top5_words)
top5_words = ["" "" "" "" ""]; %5 palavras mais geradas
```

```
%Ciclo que encontra as palavras mais geradas e o seu respectivo nº de vezes geradas
for chave = keys
    if palavras(chave{1}) > top5(1)
        tmp = [top5(1) top5(2) top5(3) top5(4)];
        tmp_words = [top5_words(1) top5_words(2) top5_words(3) top5_words(4)];
        top5(1) = palavras(chave{1});
        top5(2) = tmp(1);
        top5(3) = tmp(2);
        top5(4) = tmp(3);
        top5(5) = tmp(4);
        top5_words(1) = chave{1};
        top5_words(2) = tmp_words(1);
        top5_words(3) = tmp_words(2);
        top5_words(4) = tmp_words(3);
        top5_words(5) = tmp_words(4);
    elseif palavras(chave{1}) > top5(2)
        tmp = [top5(2) top5(3) top5(4)];
        tmp_words = [top5_words(2) top5_words(3) top5_words(4)];
        top5(2) = palavras(chave{1});
        top5(3) = tmp(1);
        top5(4) = tmp(2);
        top5(5) = tmp(3);
        top5_words(2) = chave{1};
        top5_words(3) = tmp_words(1);
        top5_words(4) = tmp_words(2);
        top5_words(5) = tmp_words(3);
    elseif palavras(chave{1}) > top5(3)
        tmp = [top5(3) top5(4)];
        tmp_words = [top5_words(3) top5_words(4)];
        top5(3) = palavras(chave{1});
        top5(4) = tmp(1);
        top5(5) = tmp(2);
        top5_words(3) = chave{1};
        top5_words(4) = tmp_words(1);
        top5_words(5) = tmp_words(2);
    elseif palavras(chave{1}) > top5(4)
        tmp = top5(4);
        tmp_words = top5_words(4);
        top5(4) = palavras(chave{1});
        top5(5) = tmp;
        top5_words(4) = chave{1};
        top5_words(5) = tmp_words;
    elseif palavras(chave{1}) > top5(5)
        top5(5) = palavras(chave{1});
    end
```

```
end

%Transformar, para cada palavra em top5_words, o número de vezes geradas na probabilidade
%de ser gerada (quociente entre número de vezes gerada e número de palavras geradas)
for i = 1:5
   top5(i) = top5(i)/N;
end
```

Funcionamento do ciclo for:
Para cada palavra (*chave*) na lista de palavras (chaves de *palavras*), averigua-se se a probabilidade dessa palavra (*palavras*(*chave*{1})) é maior que pelo menos uma das 5 mais prováveis até agora (*top5_words*). Caso seja, substitui-se a palavra com probabilidade mais elevada de *top5_words* pela *chave*.
As palavras de *top5_words* com menor probabilidade que a substituída descem 1 posição, sendo eliminada a de menor probabilidade. Para isto, são utilizadas variáveis temporárias (*tmp* e *tmp_words*).
Foram feitas 3 execuções para análise na alínea seguinte.

Número de palavras diferentes geradas: 18470

Posição	Palavra	Probabilidade
1	*o*	0.0826
2	*a*	0.0623
3	*mo*	0.0420
4	*ro*	0.0410
5	*ra*	0.0315

Número de palavras diferentes geradas: 18174

Posição	Palavra	Probabilidade
1	*o*	0.0829
2	*a*	0.0619
3	*mo*	0.0424
4	*ro*	0.0422
5	*ma*	0.0324

Número de palavras diferentes geradas: 18462

Posição	Palavra	Probabilidade
1	*o*	0.0826
2	*a*	0.0631
3	*mo*	0.0411
4	*ro*	0.0409
5	*ma*	0.0316

1. c)

Cálculo das probabilidades teóricas em Matlab:

```
%Existem 4 letras, e sendo a probabilidade de começar com cada uma a
mesma, cada letra tem uma probabilidade de 1/4 de ser gerada
prob_o = 1/4*T(5,2);          %1/12 =~ 0.0833
prob_a = 1/4*T(5,4);          %1/16 =  0.0625
prob_ro = 1/4*T(2,1)*T(5,2);  %1/24 =~ 0.0417
prob_mo = 1/4*T(2,3)*T(5,2);  %1/24 =~ 0.0417
prob_ra = 1/4*T(4,1)*T(5,4);  %1/32 =~ 0.0313
prob_ma = 1/4*T(4,3)*T(5,4);  %1/32 =~ 0.0313
```

Cálculo por análise manual da matriz:

Combinação	Cálculo	Valor
o	1/4 * 1/3	1/12 =~ 0.0833
a	1/4 * 1/16	1/16 = 0.0625
ro	1/4 * 1/2 * 1/3	1/24 =~ 0.0417
mo	1/4 * 1/2 * 1/3	1/24 =~ 0.0417
ra	1/4 * 1/2 * 1/4	1/32 =~ 0.0313
ma	1/4 * 1/2 * 1/4	1/32 =~ 0.0313

Comparação das probabilidades (teórica e por simulação)

Combinação	Prob. Teórica	Prob. Exec. 1	Prob. Exec. 2	Prob. Exec. 3
o	0.0833	0.0826	0.0829	0.0826
a	0.0625	0.0623	0.0619	0.0631
ro	0.0417	0.0410	0.0422	0.0409
mo	0.0417	0.0420	0.0424	0.0411
ra	0.0313	0.0315	N/A	N/A
ma	0.0313	N/A	0.0324	0.0316

Comparando a probabilidade teórica e as probabilidades simuladas, conclui-se que:

- Os resultados obtidos experimentalmente são coerentes com os teóricos;
- "o" é a palavra com maior probabilidade de ser gerada;
- "a" é a segunda palavra com maior probabilidade de ser gerada;
- As 3ª e 4ª palavras mais prováveis são "ro" e "mo" (com igual probabilidade, ao contrário das primeiras duas);
- A 5ª palavra mais gerada é "ra" ou "ma", com igual probabilidade.

1. d)

```
fid = fopen('wordlist-preao-20201103.txt', 'r'); %Abre o ficheiro em modo leitura
dicionario = textscan(fid, '%s');                %Lê o ficheiro
fclose(fid);                                     %Fecha o ficheiro
dicionario = dicionario{1, 1};                   %Formata o array
n_palavras_reais = 0;                    %Variável onde se guardará o número de palavras reais
for chave = keys                         %Para cada palavra
    if binsearch(dicionario, chave{1})   %Se a palavra está no dicionário
       n_palavras_reais = n_palavras_reais + palavras(chave{1}); %Adicionar a
                                                                 %n_palavras_reais
    end                                                          %o número de vezes que a
end                                                              %palavra foi gerada
prob_real = n_palavras_reais/N; %Probabilidade de ser gerada uma palavra portuguesa existente
```

Para calcularmos a probabilidade de o gerador de palavras aleatórias criar uma palavra real portuguesa, foi lido o ficheiro *wordlist-preao-20201103.txt* e guardadas as suas palavras num cell array, *dicionario*.
Baseámonos no exemplo presente no ficheiro *explanation_PL03*.
De seguida, utilizamos a função *binsearch* para verificarmos se cada palavra pertencia ao *dicionario*, sendo que caso pertença, é somado ao número de palavras reais (*n_palavras_reais*) o número de vezes que a palavra foi gerada (*palavras*(*chave*{1})).
Por fim, dividimos o número de palavras reais pelo número de palavras geradas (*N*).

Verificou-se que a probabilidade de gerar uma palavra válida em português, *prob* é ~35%.

```
function outbool = binsearch(vec,key)
    low = 1;
    high = length(vec);
    outind = -1;
    while low <= high && outind == -1
        mid = floor((low + high)/2);
        if convertCharsToStrings(vec{mid}) == convertCharsToStrings(key)
            outind = mid;
        elseif convertCharsToStrings(key) < convertCharsToStrings(vec{mid})
            high = mid -1;
        else
            low = mid + 1;
        end
    end
    if outind == -1
        outbool = false;
    else
        outbool = true;
    end
end
```

Nota: Pesquisámos pelo funcionamento da função *isMember*, e vimos que devia verificar se o argumento estava ordenado, o que não aconteceu, pelo que criámos uma função *binsearch* para fazer uma pesquisa binária de modo a otimizar o código, sendo isto possível porque o dicionário está ordenado alfabeticamente com as palavras começadas por letra maiúscula primeiro. Esta alteração diminuiu drasticamente o tempo de execução do código.

1. e)

```
function str = crawl(H, first, last, limit)
    % the sequence of states will be saved in the vector "state"
    % initially, the vector contains only the initial state:
    state = [first];
    % keep moving from state to state until state "last" is reached:
    if limit <= 0 %Caso o n seja <= 0, considera-se que não existe limite
        while (1)
            state(end+1) = nextState(H, state(end));
            if (state(end) == last)
                break;
            end
        end
    else
        for i = 1:limit  %A palavra gerada terá tamanho máximo limit
```

```
            state(end+1) = nextState(H, state(end));
            if (state(end) == last)
                break;
            end
        end
    end
    base = 'roma.';
    str_com_ponto = base(state);
    str = str_com_ponto(1:end-1);
end
```

De modo a podermos colocar um limite no tamanho das palavras que o gerador de palavras irá gerar, adicionámos um terceiro argumento *n*. Depois, foram criados dois casos:

1. Definimos que quando *n* for menor ou igual a 0 as palavras geradas não terão limite, a função funciona como anteriormente;
2. Quando *n* for maior que 0, as palavras terão um tamanho máximo de *n*, sendo para isso utilizado um ciclo *for*, que termina se for atingido o estado final, ou se alcançar o fim do ciclo (limite de letras);

1. f)

```
clc;
clear;
N = 1e5;

limite_tamanho = [4, 6, 8]           %Array com os valores de tamanho máximo de cada palavra
prob_real = zeros(1, 3);             %Array para guardar os valores das probabilidades
n_palavras_diferentes = zeros(1, 3); %Array para guardar os valores dos números de palavras
                                     %diferentes
T = [0   1/3 0   1/4 0
     1/2 0   1/2 1/4 0
     0   1/3 0   1/4 0
     1/2 0   1/2 0   0
     0   1/3 0   1/4 1];

for j = 1:size(limite_tamanho, 2)   %Para cada limite_tamanho
    palavras = containers.Map();
    for n = 1:N
        first = randi(4);
        str = crawl(T, first, 5, limite_tamanho(j));
        if isKey(palavras, str)
            palavras(str) = palavras(str) + 1;
        else
            palavras(str) = 1;
        end
```

```
    end
    clear keys;
    keys = keys(palavras);
    n_palavras_diferentes(j) = size(palavras, 1);

    fid= fopen('wordlist-preao-20201103.txt', 'r');
    dicionario= textscan(fid, '%s');
    fclose(fid);
    dicionario= dicionario{1, 1};
    palavras_reais = 0;
    for chave = keys
        if binsearch(dicionario,chave{1})
           palavras_reais = palavras_reais + palavras(chave{1});
        end
    end
    prob_real(j) = palavras_reais/N;
end
```

Probabilidades de uma palavra gerada ser uma palavra válida portuguesa e número de palavras diferentes geradas

Limite de caracteres	Prob. real	Palavras diferentes
4	0.50	61
6	0.36	307
8	0.35	1515

Analisando os resultados obtidos, podemos concluir que:

- A probabilidade de uma palavra gerada ser uma palavra válida portuguesa é maior quanto menor for o limite do tamanho possível de cada palavra;
- Quanto maior for o limite, maior será o número de palavras diferentes geradas;
- Em comparação com os resultados obtidos na alínea b), podemos afirmar que o número de palavras diferentes que foram geradas nesta alínea é extraordinariamente inferior ao obtido. Isto deve-se ao facto de que, existindo um limite de tamanho, existe um máximo de combinações possíveis, enquanto que não existindo limite de tamanho, também não existe limite de combinações possíveis;
- Em relação aos resultados obtidos na alínea d), podemos afirmar que a grande diferença entre o número de palavras diferentes geradas na alínea d) e f) se deve ao facto explicado no ponto anterior. Quanto à probabilidade de uma palavra gerada ser uma palavra válida portuguesa, que tem um valor de ~35% na alínea d), podemos afirmar que a probabilidade é semelhante quando as palavras geradas têm um limite de tamanho igual ou superior a 6, sendo consideravelmente maior quando o limite de tamanho é 4. Isto deve-se ao facto de que com no máximo 4 letras, muitas das combinações possíveis são palavras válidas portuguesas e estas combinações terem alta probabilidade de ocorrerem.

2)

```
T = [0   0.3 0   0.3 0
     0.3 0   0.3 0.1 0
     0   0.2 0   0.2 0
     0.7 0   0.7 0   0
     0   0.5 0   0.4 1]; %Matriz obtida a partir da figura da pergunta 2

limite_tamanho = [4, 6 , 8 , -1];    %Foi adicionado o elemento "-1"
                                     %para o caso em que não existe limite de tamanho
prob_real = zeros(1, 4);             %Adicionou-se um elemento para o caso de limite infinito
n_palavras_diferentes = zeros(1, 4); %Adicionou-se um elemento para o caso de limite infinito
```

O código utilizado nesta questão é muito semelhante ao utilizado na questão 1.f), sendo que apenas variam os seguintes aspectos:

- A matriz *T*, obtida a partir da figura presente no enunciado da pergunta 2;
- A adição de "-1" ao array *limite_tamanho*, para serem obtidos também os resultados de quando não existe limite de tamanho da palavra gerada;
- A adição de um elemento aos arrays *prob_real* e *n_palavras_diferentes* e ao ciclo for, para guardar informação relativa ao caso ilimitado.

Probabilidades de uma palavra gerada ser uma palavra válida portuguesa e número de diferentes palavras geradas

Limite de caracteres	Prob. real	Palavras diferentes
4	0.65	61
6	0.54	307
8	0.52	1468
∞	0.52	7185

Analisando os resultados obtidos, podemos concluir que:

- Tal como na alínea 1f), quanto menor for o limite de tamanho, maior é a probabilidade de a palavra gerada ser uma palavra portuguesa e quanto maior for o limite de tamanho, maior é o número de palavras diferentes.
- Em relação aos resultados obtidos em 1d), verifica-se que a probabilidade quando o tamanho é ilimitado é menor que na questão corrente;
- Em relação aos resultados obtidos em 1f), pode-se afirmar que as probabilidades obtidas na questão atual são superiores qualquer que seja o limite de tamanho das palavras geradas;

Em suma, é possível afirmar que o gerador desta questão é mais eficiente que os das questões anteriores. Isto deve-se ao facto das probabilidades presentes na matriz de transição representarem as probabilidades das sequências ocorrerem nas palavras de língua portuguesa melhor que a matriz de transição das questões anteriores.

3)

```
N = 1e5;
limite_tamanho = [4, 6, 8, -1];
prob_real = zeros(1, 4);
n_palavras_diferentes = zeros(1, 4);
T = [0   0.3 0   0.3 0
     0.3 0   0.3 0.1 0
     0   0.2 0   0.2 0
     0.7 0   0.7 0   0
     0   0.5 0   0.4 1];
for j=1:4
    fid = fopen('wordlist-preao-20201103.txt', 'r');
    dicionario = textscan(fid, '%s');
    fclose(fid);
    dicionario = dicionario{1, 1};
    letras = 'roma';              %Array de chars com as letras presentes no enunciado
    dicionario_filtrado = {};     %Cell array que irá guardar as palavras que apenas possuem
                                  % as letras em letras
    for i = 1:size(dicionario, 1) %Para cada elemento do dicionario
        if min(ismember(dicionario{i},letras)) %Se a palavra apenas possuir as letras em letras
            dicionario_filtrado{end+1} = dicionario(i, 1); %Adicionar ao dicionario_filtrado
        end
    end
    letra_r = 0;       %Counter para as vezes que a letra r é a primeira da palavra
    letra_o = 0;       %Counter para as vezes que a letra o é a primeira da palavra
    letra_m = 0;       %Counter para as vezes que a letra m é a primeira da palavra
    letra_a = 0;       %Counter para as vezes que a letra a é a primeira da palavra
    for i = 1:size(dicionario_filtrado, 2) %Para cada palavra no dicionario_filtrado
       cell_array_primeira_letra = dicionario_filtrado{i}(1); %Cell array da palavra
       primeira_letra = cell_array_primeira_letra{1}(1);      %Primeira letra da palavra
       if primeira_letra == 'r'     %Se a primeira letra for r
           letra_r = letra_r + 1;   %Adicionar ao counter_r
       elseif primeira_letra == 'o' %Se a primeira letra for o
           letra_o = letra_o + 1;   %Adicionar ao counter_o
       elseif primeira_letra == 'm' %Se a primeira letra for m
           letra_m = letra_m + 1;   %Adicionar ao counter_m
       elseif primeira_letra == 'a' %Se a primeira letra for a
```

```
            letra_a = letra_a + 1;   %Adicionar ao counter_a
        end
    end

    prob_r = letra_r/size(dicionario_filtrado,2); %Probabilidade da letra r ser a primeira
    prob_o = letra_o/size(dicionario_filtrado,2); %Probabilidade da letra o ser a primeira
    prob_m = letra_m/size(dicionario_filtrado,2); %Probabilidade da letra m ser a primeira
    prob_a = letra_a/size(dicionario_filtrado,2); %Probabilidade da letra a ser a primeira
    palavras=containers.Map();
    for n=1:N
        r = rand();                 %Gera um número entre 0 e 1
        if r < prob_r               %Se r entre [0,prob_r]
            first = 1;              %O primeiro estado vai ser 1, ou seja, a primeira letra é r
        elseif r < prob_r + prob_o  %Se r entre [prob_r,prob_o]
            first = 2;              %A primeira letra é o
        elseif r < prob_r + prob_o + prob_m  %Se r entre [prob_o,prob_m]
            first = 3;                       %A primeira letra é m
        else                                 %Se r entre [prob_m,prob_a]
            first = 4;                       %A primeira letra é a
        end
        str = crawl(T, first, 5, limite_tamanho(j));
        if isKey(palavras,str)
            palavras(str) = palavras(str)+1;
        else
            palavras(str) = 1;
        end
    end
    clear keys;
    keys = keys(palavras);
    n_palavras_diferentes(j) = size(palavras,1);
    palavras_reais = 0;
    for chave = keys
        if binsearch(dicionario,chave{1})
           palavras_reais = palavras_reais + palavras(chave{1});
        end
    end
    prob_real(j) = palavras_reais/N;
end
```

Para se estimar as probabilidades de cada letra ser a primeira, criámos, em primeiro lugar, um cell array, *dicinario_filtrado*, onde são colocadas apenas as palavras que possuem apenas as letras ‘r’, ‘o’, ‘m’ e ‘a’.
De seguida, contamos quantas vezes cada letra é a primeira da palavra, e por fim dividimos cada contador pelo número de palavras no cell array.

Probabilidades de uma palavra gerada ser uma palavra válida portuguesa e número de diferentes palavras geradas

Limite de caracteres	Prob. real	Palavras diferentes
4	0.74	61
6	0.65	307
8	0.62	1407
∞	0.62	7030

Analisando os resultados obtidos, podemos concluir que:

- Tal como nas questões anteriores, quanto menor o limite do tamanho das palavras, maior é a probabilidade de ser gerada uma palavra válida portuguesa e menor é o número de palavras diferentes;
- Comparando com as questões anteriores, é possível verificar que a probabilidade aumenta independentemente do limite de tamanho usado.

Em conclusão, podemos afirmar que este gerador de palavras é o mais eficiente, visto que as probabilidades de gerar uma palavra portuguesa válida são as maiores, independentemente do limite de tamanho das palavras geradas, pois as probabilidades da primeira letra já não são idênticas, mas sim calculadas a partir da lista de palavras portuguesas, logo o gerador vai ter uma melhor perceção das probabilidades certas, criando assim mais frequentemente palavras portuguesas válidas.

4)

```
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 1];
```

O código utilizado nesta questão é igual ao utilizado na questão anterior, com exceção da matriz de transição, que foi alterada de maneira a refletir a nova matriz presente no enunciado na questão.

Probabilidades de uma palavra gerada ser uma palavra válida portuguesa e número de diferentes palavras geradas

Limite de caracteres	Prob. real	Palavras diferentes
4	0.73	77
6	0.63	456
8	0.61	2259
∞	0.61	8671

Analisando os resultados obtidos, podemos concluir que:

- Tal como em todas as questões anteriores, quanto menor o limite do tamanho das palavras geradas, maior é a probabilidade de ser gerada uma palavra real e menor é o número de palavras diferentes;
- Contrariamente às questões anteriores, o número de palavras diferentes em todos os diferentes limites de tamanho aumentou, isto porque foi introduzida uma nova possibilidade de sequência de letras “rm”, que permite novas combinações de letras;
- Comparando com as questões anteriores, é possível verificar que as probabilidades de uma palavra gerada ser válida são ligeiramente menores em comparação à pergunta 3. Isto deve-se ao facto de a probabilidade da nova sequência “rm” não refletir o que acontece nas palavras de língua portuguesa. No entanto, as probabilidades continuam a ser maiores que nas restantes questões;

Tendo em conta todos os resultados obtidos nesta questão e nas anteriores, podemos afirmar que este gerador de palavras é ligeiramente menos eficiente que o da questão 3, em termos de probabilidade de gerar palavras reais, mas tem a vantagem de poder gerar mais palavras.

5)

```
seqs = containers.Map();            %HashMap onde se vai colocar as sequências e quantas vezes aparecem
for i=1:size(dicionario_filtrado,2) %Para cada palavra do dicionário filtrado
    word = dicionario_filtrado{i};  %Cell array da palavra do dicionário
    word = word{1};                 %Palavra do dicionário
    word = strcat(word,'.');        %Adicionar o ‘.’, ou seja, o indicador de fim de palavra
    for j=1:size(word,2)-1          %Para cada sequência da palavra
        seq = strcat(word(j),word(j+1));  %Sequência de dois caracteres
        if isKey(seqs,seq)                %Se pertence ao HashMap
            seqs(seq) = seqs(seq)+1;      %Adicionar 1 ao número de vezes que a sequência apareceu
```

```
        Else                            %Se não pertence ao HashMap
            seqs(seq) = 1;              %Criar a sequência no HashMap
        end
    end
end



T = [getvalue(seqs,'rr') getvalue(seqs,'or') getvalue(seqs,'mr') getvalue(seqs,'ar') getvalue(seqs,'.r')
     getvalue(seqs,'ro') getvalue(seqs,'oo') getvalue(seqs,'mo') getvalue(seqs,'ao') getvalue(seqs,'.o')
     getvalue(seqs,'rm') getvalue(seqs,'om') getvalue(seqs,'mm') getvalue(seqs,'am') getvalue(seqs,'.m')
     getvalue(seqs,'ra') getvalue(seqs,'oa') getvalue(seqs,'ma') getvalue(seqs,'aa') getvalue(seqs,'.a')
     getvalue(seqs,'r.') getvalue(seqs,'o.') getvalue(seqs,'m.') getvalue(seqs,'a.') 1];
sumcol = sum(T);  %Array com o total de sequências de cada coluna, para calcular as probabilidades
for col=1:4
   T(:,col) = T(:,col)/sumcol(col); %Calcula as probabilidades de cada sequência
end
```

Para conseguirmos criar uma matriz de transição com as probabilidades baseadas nas palavras do dicionário, precisamos de registar quantas vezes cada sequência ocorre. Para isso, criamos um HashMap, de maneira a guardar cada sequência e quantas vezes ocorre. Depois percorremos cada palavra, adicionamos o carácter '.', para simbolizar o fim de palavra, e percorremos as sequências existentes na palavra, adicionando 1 quando a sequência já existe no HashMap, ou adicionando a sequência ao HashMap, caso ainda não exista. De seguida, utilizando a função *getvalue* , que iremos explicar posteriormente, vamos buscar quantas vezes cada uma das sequências ocorreu, colocando-as numa matriz. Finalmente, somamos todas as sequências numa coluna, e dividimos a coluna por essa soma, calculando assim as probabilidades de cada sequência ocorrer.

```
function value = getvalue(map,seq)
    if isKey(map,seq)
        value = map(seq);
    else
        value = 0;
    end
end
```

Esta função tem como objetivo o de devolver o valor de uma key de um HashMap. Possui como argumentos o HashMap e a key da qual se pretende obter o valor. O Matlab já possui um método de executar esta função, no entanto, quando se tenta ir buscar um valor de uma key que não exista no HashMap, ocorre um erro. De maneira a resolvermos este problema, foi criada esta função trivial, que devolve o valor da key, caso ela exista, e caso não exista, devolve 0.

Probabilidades de uma palavra gerada ser uma palavra válida portuguesa e número de diferentes palavras geradas

Limite de caracteres	Prob. real	Palavras diferentes
4	0.61	221
6	0.47	1462
8	0.42	4829
∞	0.41	15855

Analisando os resultados obtidos, podemos concluir que:

- Tal como em todas as questões anteriores, quanto menor o limite do tamanho das palavras geradas, maior é a probabilidade de ser gerada uma palavra real e menor é o número de palavras diferentes;
- Comparando com as restantes questões, os resultados obtidos são menores que todas as questões com exceção da questão 1f). Isto deve-se ao facto de, contrariamente aos restantes geradores, existirem aqui as sequências de estados para si próprios, como 'ss' e 'rr', o que embora aconteça muitas vezes nas palavras selecionadas, têm que estar em certas posições para terem a hipótese se formarem uma palavra válida portuguesa. Como a geração de palavras não utiliza probabilidade condicionada, estas sequências irão prejudicar as probabilidades de ser gerada uma palavra válida portuguesa. Por exemplo, não existem palavras em português com as sequências 'rrr' ou 'sss', mas o gerador, ao gerar um caractere para adicionar a 'rr' ou 'ss' apenas tem em conta que o caractere anterior é 's' ou 'r', podendo assim gerar outro 'r' ou 's' apesar da sequência ser impossível em português.

Em conclusão, o gerador de palavras desta questão é muito menos eficiente em comparação aos geradores das questões 3 e 4, pois existem na matriz de transição desta questão a hipótese de estados transitarem para si próprios, o que apenas iria funcionar e aumentar a probabilidade de ser gerada uma palavra real caso fosse utilizada probabilidade condicional que tivesse em conta mais caracteres antecedentes.