

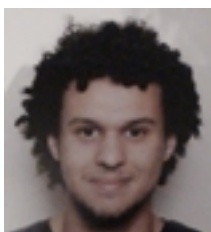


# **Relatório Final**

## **Métodos de programação I**

**Mestrado Integrado em Engenharia de Comunicações**

# **Descodificação de morse**



Rúben Sousa, nº 70013

Email: rubensousa94@gmail.com

# Índice

Página 03	–	Introdução
Página 04	–	Algoritmo não refinado
Página 05	–	Algoritmo refinado
Página 09	–	Descrição do Algoritmo
Página 16	–	Fluxograma
Página 19	–	Código em C
Página 23	–	Conclusão

# Introdução

Este projeto visa propor uma solução para determinar o número de frases que é possível formar dada uma sequência em código morse e um dicionário, ambos introduzidos pelo utilizador do programa.

Para a resolução deste problema ser possível, foi necessário compreender cada passo que o programa teria de fazer recorrendo a um algoritmo bem construído.

Dado que o enunciado do problema não explicitava qual deveria ser o comportamento do nosso programa na situação de introdução de dados inválidos (no caso de haver código morse misturado com outros caracteres, palavras com dígitos no dicionário, palavras com letras minúsculas, palavras repetidas, etc.) não foram programadas quaisquer restrições na introdução de dados, ficando o correto funcionamento do programa dependente do utilizador.

# Algoritmo não refinado

1. Ler sequência de código morse;
2. Ler número (n) de palavras a serem introduzidas no dicionário;
3. Ler as n palavras introduzidas para o dicionário;
4. Verificar quais das palavras podem iniciar a sequência morse.
  - 4.1. Sempre que for possível representar uma palavra, verificar quais das palavras podem ser a seguir à palavra já correspondida (voltar ao ponto 4).
5. Contar uma frase válida sempre que é possível preencher as palavras ocupando todos os "." e "-". Voltar para 4 enquanto não tiver percorrido as palavras todas.

# Algoritmo refinado

1. [programa principal]
  - 1.1. [Ler sequência morse (seqmorse)]
  - 1.2. [Ler número de palavras do dicionário (n)]
    - 1.2.1. Para  $i \leftarrow 0$  até 1,2... até n fazer  
ler dicionário [i]
  - 1.3. [Calcular o tamanho da sequência morse]
    - 1.3.1. [Inicializar variável que contém o tamanho da sequência morse]
      - 1.3.1.1.  $sizemorse \leftarrow 0$
    - 1.3.2. [Percorrer cada índice da string que contém a sequência morse]
      - 1.3.2.1. Enquanto  $seqmorse[sizemorse] < > '\0'$  fazer  
 $sizemorse \leftarrow sizemorse + 1$
  - 1.4. [Inicializar o índice de verificação da sequência morse]
    - 1.4.1.  $startpoint \leftarrow 0$
  - 1.5. [Inicializar o número de frases possíveis]
    - 1.5.1.  $valid \leftarrow 0$
  - 1.6. [Chamar função que verifica a correspondência das palavras do dicionário]
    - 1.6.1.  $checkword(n, sizemorse, dicionario, seqmorse, startpoint)$
  - 1.7. Escrever número de frases válidas (variável: valid)

2. [Verificar as palavras que podem ser correspondidas (checkword)]
  - 2.1. Inicializar variável que para o ciclo caso encontre um caractere não correspondível]
    - 2.1.1.  $s \leftarrow 1$
  - 2.2. [Percorrer cada palavra do dicionário]
    - 2.2.1. Para  $i \leftarrow 0$  até 1,2... até  $n$  fazer
      - $j \leftarrow 0$
      - $\text{validchar} \leftarrow 0$
      - $\text{mrsindex} \leftarrow \text{startpoint}$
      - Enquanto  $(\text{dic}[i][j] \neq '\0') \wedge (s=1)$  fazer
        - $\text{eqmorse}(i,j,\text{dic},\text{checkmorse})$
        - Se  $(\text{checkchar}(\text{mrsindex},\text{seqmorse},\text{checkmorse})=0)$  fazer
          - $s \leftarrow 0$
        - Senão, fazer
          - $\text{mrsindex} \leftarrow \text{checkchar}(\text{mrsindex},\text{seqmorse},\text{checkmorse})$
          - $j \leftarrow j+1;$
          - $\text{validchar} \leftarrow \text{validchar}+1;$
        - FimSenão
        - FimEnquanto
      - Se  $(\text{validchar}=\text{strlen}(\text{dic}[i])) \wedge (\text{mrsindex} \neq \text{sizemrs})$  fazer
        - $\text{checkword}(n,\text{sizemrs},\text{dic},\text{seqmorse},\text{mrsindex})$
      - Se  $\text{mrsindex}=\text{sizemrs}$  fazer
        - $\text{valid} \leftarrow \text{valid}+1$
      - $s \leftarrow 1$
    - FimPara

### 3. [Determinar o equivalente morse de um caractere (eqmorse)]

#### 3.1. Caso (dic[i][j]) Seja:

```
'A' fazer:  strcpy(checkmorse,".-")
'B' fazer:  strcpy(checkmorse,"-...")
'C' fazer:  strcpy(checkmorse,"-.-.")
'D' fazer:  strcpy(checkmorse,"-..")
'E' fazer:  strcpy(checkmorse,".")
'F' fazer:  strcpy(checkmorse,"..-.")
'G' fazer:  strcpy(checkmorse,"--.")
'H' fazer:  strcpy(checkmorse,"....")
'I' fazer:  strcpy(checkmorse,"..")
'J' fazer:  strcpy(checkmorse,".---")
'K' fazer:  strcpy(checkmorse,"-.")
'L' fazer:  strcpy(checkmorse,"-..")
'M' fazer:  strcpy(checkmorse,"--")
'N' fazer:  strcpy(checkmorse,"-.")
'O' fazer:  strcpy(checkmorse,"---")
'P' fazer:  strcpy(checkmorse,".-.")
'Q' fazer:  strcpy(checkmorse,"--.-")
'R' fazer:  strcpy(checkmorse,".-")
'S' fazer:  strcpy(checkmorse,"...")
'T' fazer:  strcpy(checkmorse,"-")
'U' fazer:  strcpy(checkmorse,"..-")
'V' fazer:  strcpy(checkmorse,"...-")
'W' fazer:  strcpy(checkmorse,"-.-")
'X' fazer:  strcpy(checkmorse,"-.-")
'Y' fazer:  strcpy(checkmorse,"-.-")
'Z' fazer:  strcpy(checkmorse,"--..")
Outra coisa: devolver 0
```

4. [Verificar se um equivalente morse de um caractere corresponde à sequência morse atual (checkchar)]

4.1. [Inicializar variável que servirá como índice para a string checkmorse]

4.1.1.  $i \leftarrow 0$

4.2. [Verificar se a string checkmorse corresponde à sequência morse atual]

4.2.1. Enquanto (checkmorse[i] <> '\0') fazer  
    se (checkmorse[i] <> seqmorse[mrsindex])  
        devolver 0

$i \leftarrow i + 1$

$mrsindex \leftarrow mrsindex + 1$

FimEnquanto

4.3. [Devolver nova posição do índice mrsindex]

4.3.1. Devolver mrsindex



## Descrição do Algoritmo

Os exemplos que se seguem, servem para poder descrever o algoritmo detalhadamente.

Vamos assumir que introduzimos uma sequência morse que corresponde a "MARIA". Assim, temos uma string que contém os seguintes pontos e traços:

M		A		R		I		A	
-	-	.	-	.	-	.	.	.	-
0	1	2	3	4	5	6	7	8	9

Introduziremos agora 7 palavras no dicionário. Como o dicionário é uma matriz, ficará cada palavra por linha. Como podemos repetir sempre a mesma palavra, as combinações possíveis de frases usando estas palavras são:  $7^7 = 823543$

Índice	0	1	2	3	4	5	6
0	B	R	I	N	C	A	\0
1	M	A	R	I	A	\0	
2	I	A	\0				
3	M	A	\0				
4	M	A	R	\0			
5	G	A	T	O	\0		
6	R	I	A	\0			

Depois de acabarmos o ciclo da etapa 1.3.2.1 do algoritmo refinado, a variável **sizemorse** ficará com o valor 11.

A primeira chamada à função **checkword**, que tem como parâmetros (n,sizemorse,dicionario,seqmorse,startpoint), será feita da seguinte forma:

**checkword(7,11,dicionário introduzido,sequência morse introduzida,0)**

Nesta função, será declarado:

- Uma nova string que irá conter a conversão em morse de um caractere de uma das 7 palavras. A string tem o nome de **checkmorse**.
- Uma variável do tipo inteiro que irá conter o valor do índice da sequência morse que está a ser comparado, **mrsindex**.
- Um contador de caracteres correspondidos corretamente, **validchar**.

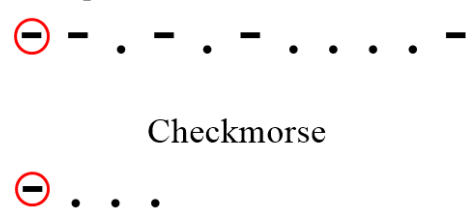

O ciclo “para” da etapa 2.2.1 começará com a primeira palavra do dicionário, “BRINCA”. **mrsindex** está com o valor 0 (**startpoint** ainda não foi alterado).

Como **i** e o **j** estão ambos a 0 de momento, a função eqmorse irá receber e converter a letra “B” para “-...” na string **checkmorse**:

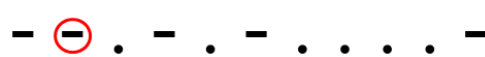

—	.	.	.
0	1	2	3

Agora, a função **checkchar** será chamada para comparar a string **checkmorse** com a string que contém a sequência morse.

As variáveis usadas para o índice das duas strings têm de ser diferentes, por isso, no início da função **checkchar**, é inicializado sempre um **i** a 0 que será para a string **checkmorse**.

Sequência morse introduzida  
  
 Checkmorse  
  
 mrsindex=0

Como **checkmorse[0]=seqmorse[0]**, incrementamos **i** e **mrsindex** por uma unidade para testar o traço a seguir.

Sequência morse introduzida  
  
 Checkmorse  
  
 mrsindex=1

Como desta vez **checkmorse[1]≠seqmorse[1]**, a função checkchar devolve 0.

Como as condições do ciclo da função checkword eram enquanto **dic[i][j] ≠'\0'** **^ s=1**, **s** toma o valor de 0 para parar de verificar a palavra "BRINCA", visto o "B" não corresponder ao início da sequência morse.

**s** volta a tomar o valor de 1, fora do ciclo, para poder verificar a próxima palavra.

**i** é incrementando por um valor para verificar a próxima palavra do dicionário.

A próxima palavra a ser testada é "MARIA".

Como a variável **startpoint** não foi alterada, **mrsindex** volta a ser 0.

Entrando no ciclo **dic[1][0] ≠ '\0' ∧ s=1**, a função eqmorse recebe o caractere "M" e converte para "--" na string **checkmorse**.

Sequência morse introduzida

— — . — . . . . —

Checkmorse

— —

Desta vez, a função **checkchar** verifica que o caractere "M" foi corretamente correspondido, e devolve o novo valor de **mrsindex**, 2.

De volta à função **checkword**, como foi correspondido um caractere e ainda não chegou ao fim da palavra, **j** é incrementado por uma unidade. Será agora testada a letra "A".

Sequência morse introduzida

— — • — . — . . . . —

Checkmorse

• —

Como também foi correspondida, **mrsindex** ← 4

Para as próximas letras, "R", "I", "A":

Letra R	Letra I	Letra A
<p>Sequência morse introduzida</p> <p>— . — ● — ● . . . —</p> <p>Checkmorse</p> <p>● — ●</p>	<p>Sequência morse introduzida</p> <p>— . — . — . ● ● . —</p> <p>Checkmorse</p> <p>● ●</p>	<p>Sequência morse introduzida</p> <p>— . — . — . . . ● —</p> <p>Checkmorse</p> <p>● —</p>
<b>mrsindex</b> ← 7	<b>mrsindex</b> ← 9	<b>mrsindex</b> ← 11

Como acabou a verificação de "MARIA" e **mrsindex** ficou com o mesmo valor de **sizeMrs**, temos uma frase válida (apesar de ser constituída apenas por uma palavra). **valid** ← 1.

Mas... ainda restam 5 palavras que podem iniciar a sequência.

A próxima palavra a ser testada é a "IA".

Passando o "I" para morse, fica "..".

Como a sequência morse começa por "—" a função checkchar devolve 0 e passa para a palavra seguinte.

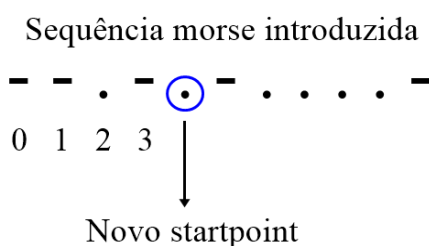
Prosseguindo então para a "MA".

Letra M	Letra A
<p>Sequência morse introduzida</p> <p>— — ● — . — . . . —</p> <p>Checkmorse</p> <p>— —</p>	<p>Sequência morse introduzida</p> <p>— — ● — . — . . . —</p> <p>Checkmorse</p> <p>● —</p>
<b>mrsindex</b> ← 2	<b>mrsindex</b> ← 4

Como "MA" foi correctamente correspondida sem ocupar a sequência morse toda, é necessário verificar quais das palavras podem ser a seguir.

Nestes casos, a função checkword faz uma chamada recursiva e passa o **mrsindex** onde ficou como parâmetro para **startpoint**.

Portanto, desta vez, temos de voltar a verificar todas as palavras novamente mas com **startpoint=4**.



Sabemos que "BRINCAR" não pode ser porque um "B" é "-...-", "MARIA" também não pode porque um "M" é "--".

"IA" começa por um "I" que é "..-", "GATO" por um "G" que é "--.". Nenhuma destas serve.

Resumindo, a única palavra do dicionário que pode ser a seguir é "RIA".

Letra R	Letra I	Letra A
<p style="text-align: center;">Sequência morse introduzida</p> <p style="text-align: center;"> </p> <p style="text-align: center;">Checkmorse</p> <p style="text-align: center;"> </p>	<p style="text-align: center;">Sequência morse introduzida</p> <p style="text-align: center;"> </p> <p style="text-align: center;">Checkmorse</p> <p style="text-align: center;"> </p>	<p style="text-align: center;">Sequência morse introduzida</p> <p style="text-align: center;"> </p> <p style="text-align: center;">Checkmorse</p> <p style="text-align: center;"> </p>

Temos então mais uma frase válida, "MA" + "RIA". **valid** ← 2.

Como o ciclo “para” da primeira chamada da função ainda não chegou ao fim (ficamos na palavra “MA” e chamámos recursivamente a função), vamos procurar agora as restantes palavras.

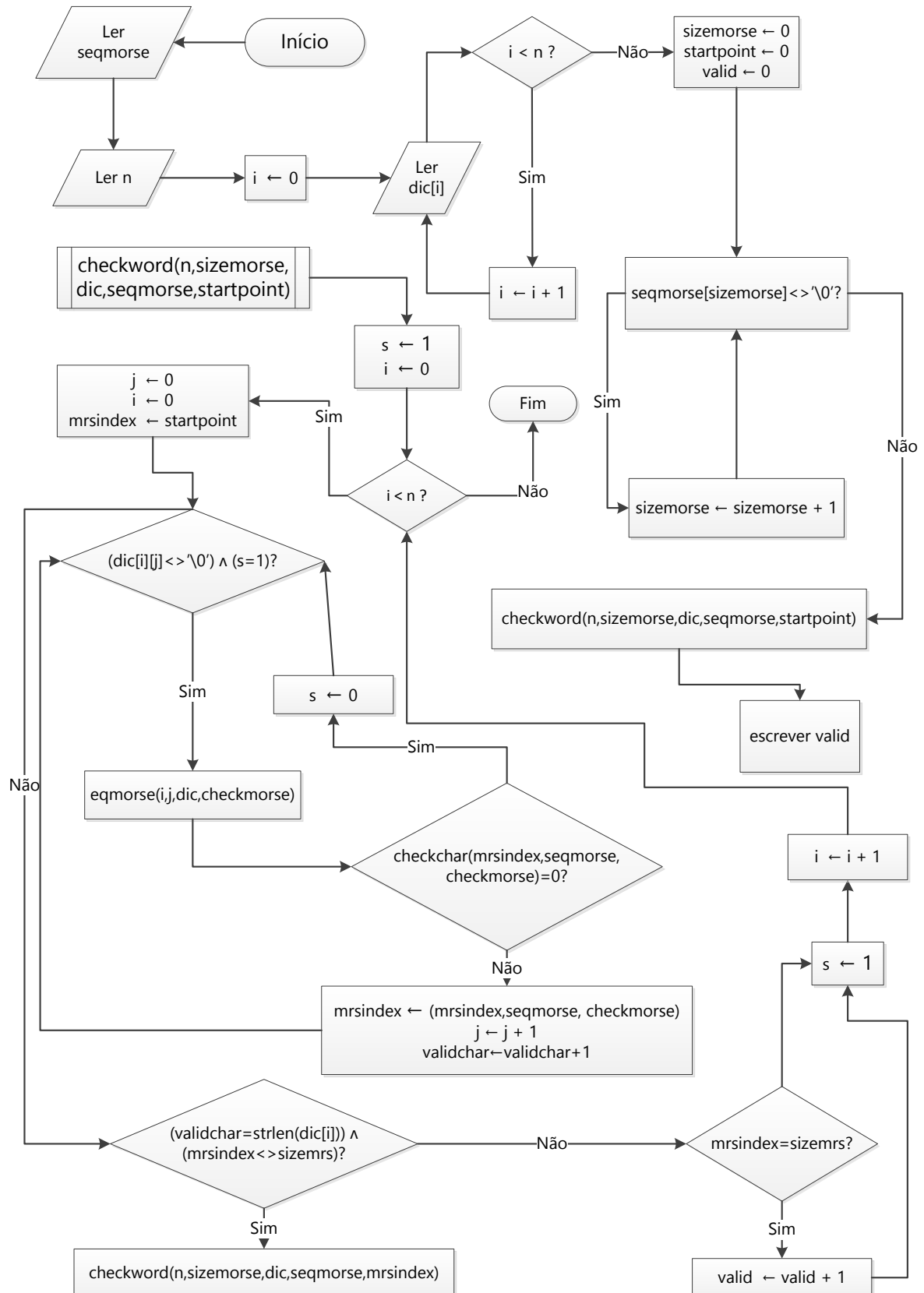
Avançando rápido usando o método anterior, descobre-se rapidamente que a única palavra que ainda pode começar a sequência é a palavra “MAR”.

Assim, como **mrsindex** é igual a 7 no fim da verificação desta palavra, o novo **startpoint** para a chamada recursiva passa a ser 7 também.

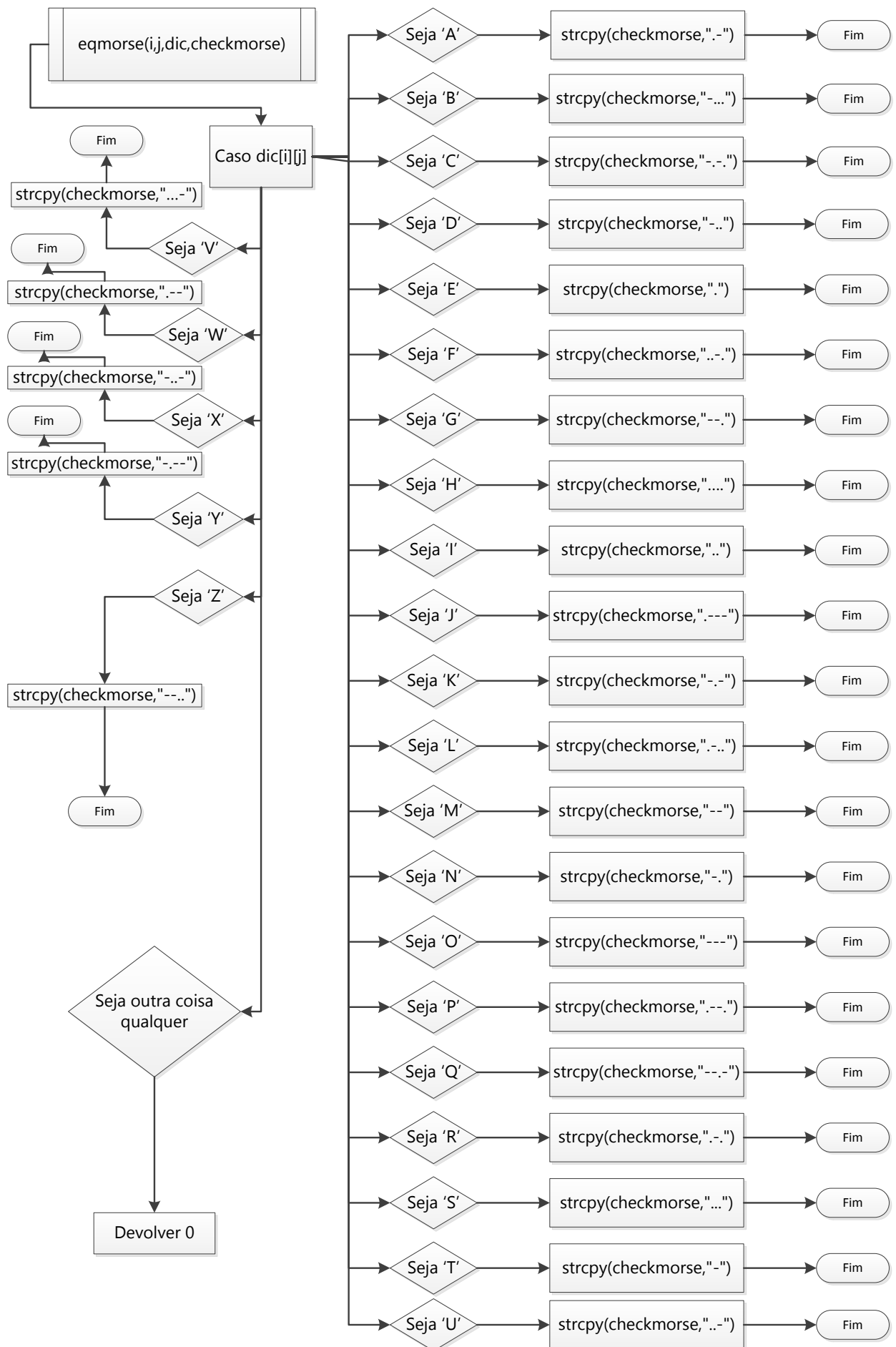
A partir deste novo **startpoint**, a única palavra que pode ser a seguir é a palavra “IA”.

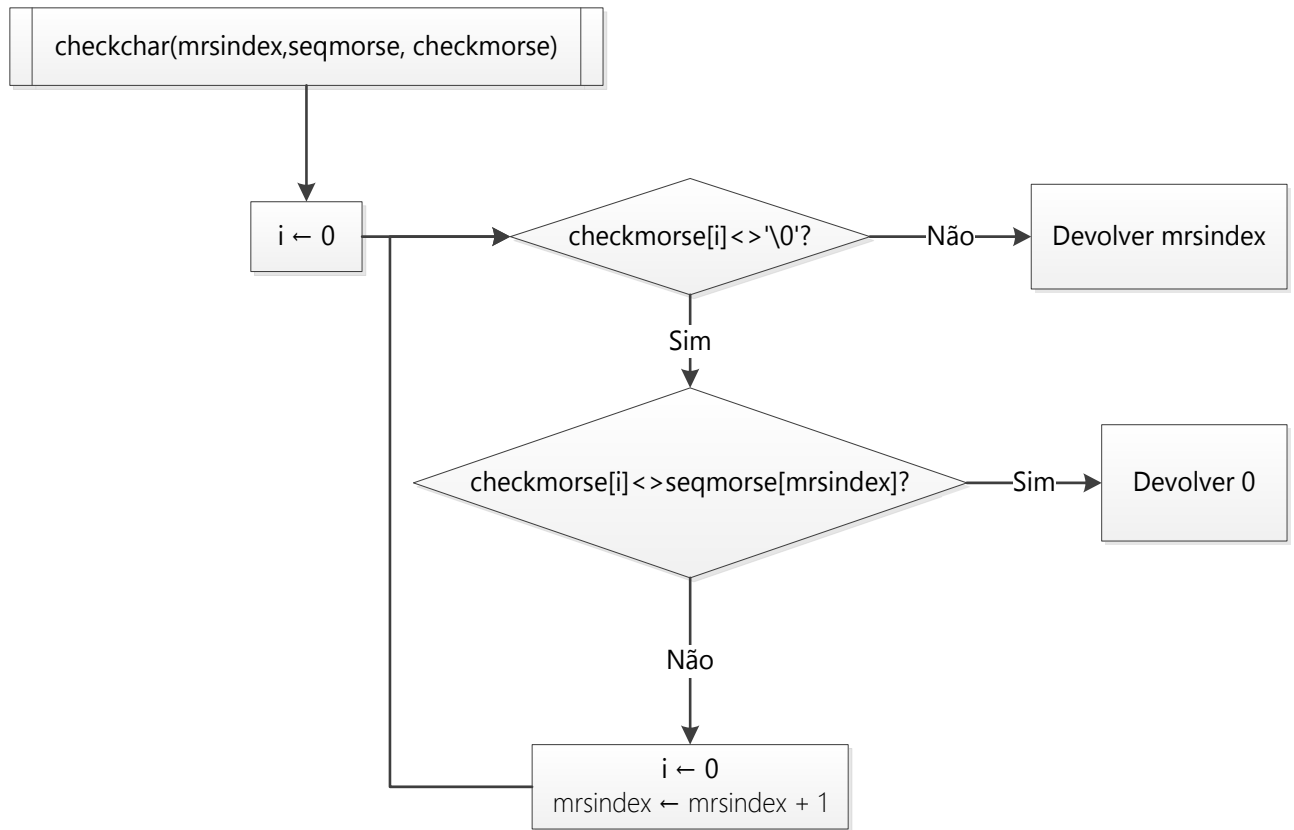
Concluindo, temos **três** frases válidas e é isso que programa escreve no fim.

# Fluxograma









# Código em C

```

/* PROJECT MIECOM 2012/2013 Métodos de Programação I */
/* Made by: Rúben Sousa nº 700013 */
/* Last edit: 19/01/2013 */
/* FINAL VERSION */

#include <stdio.h>
#include <string.h>

int eqmorse(int i, int j, char dic[][51], char checkmorse[]); /* Function that creates morse equivalent of a character */
int checkchar(int mrsindex, char seqmorse[], char checkmorse[]); /* Function that checks if a character is in the morse sequence */
void checkword(int n, int sizemrs, char dic[][51], char seqmorse[], int startpoint); /* Function that checks if a word is in the morse sequence */
int valid=0; /* Number of possible sentences */

void main()
{
    int n,i;
    int startpoint=0; /* Index used to start checking a word. */
    int sizemorse=0; /* Variable that contains the size of the morse sequence. */
    char dic[10000][51]; /* Dictionary string. Each line of the matrix contains one word. */
    char seqmorse[1001]; /* Morse sequence string. */

    scanf("%s",seqmorse); /* Read morse sequence. */

    while(seqmorse[sizemorse]!='\0') /* Count length of morse sequence to variable sizemorse. */
        sizemorse++;

    scanf("%d",&n); /* Reads number of words to put in the dictionary */

    for(i=0;i<n;i++) /* Put those n words in dictionary matrix */
        scanf("%s",dic[i]);

    checkword(n,sizemorse,dic,seqmorse,startpoint); /* Call checkword function to check number of possible sentences */
    printf("%d",valid); /* Print number of possible sentences */
}

```

```

void checkword(int n, int sizemrs, char dic[][51], char seqmorse[], int startpoint)
{
    char checkmorse[5];    /* String that contains morse equivalent of a character */
    int i,j,s;
    int mrsindex;          /* Index used to compare checkmorse and seqmorse */
    int validchar;         /* Number of characters that were correct */
    s=1;
    for(i=0;i<n;i++)
    {
        j=0;              /* Start at the first character */
        validchar=0;       /* Resets counter of characters that were correct */
        mrsindex=startpoint; /* Make sure mrsindex is pointing to where it should be */

        while((dic[i][j]!='\0')&&(s==1))
        {
            eqmorse(i,j,dic,checkmorse); /* Calls function that will create the equivalent of the current character */
            if(checkchar(mrsindex,seqmorse,checkmorse)==0) /* If character wasn't found, skips current word */
                s=0;
            else
            {
                mrsindex=checkchar(mrsindex,seqmorse,checkmorse); /* mrsindex is updated based on checkchar's output */
                j++; /* Increment j to check next char */
                validchar++; /* Increment number of characters found in morse sequence */
            }
        } /* end of while loop */

        if((validchar==strlen(dic[i]))&&(mrsindex!=sizemrs)) /* If number of characters found equals word's length */
            checkword(n,sizemrs,dic,seqmorse,mrsindex); /* and mrsindex isn't equal to morse's length */
                                                    /* calls new checkword with new startpoint */
        if(mrsindex==sizemrs) /* If mrsindex is equal to morse's length */
            valid++; /* increment number of possible sentences */

        s=1; /* Allows next word to be checked by while loop */
    } /* end of for loop */
}

```

```

int eqmorse(int i, int j, char dic[][51], char checkmorse[])
{
    switch(dic[i][j]) /* Reads character at dic[i][j] and copies its equivalent to checkmorse string */
    {
        case 'A': strcpy(checkmorse, ".-"); break;
        case 'B': strcpy(checkmorse, "..."); break;
        case 'C': strcpy(checkmorse, "-.-."); break;
        case 'D': strcpy(checkmorse, "-.."); break;
        case 'E': strcpy(checkmorse, "."); break;
        case 'F': strcpy(checkmorse, "..-"); break;
        case 'G': strcpy(checkmorse, "--."); break;
        case 'H': strcpy(checkmorse, "...."); break;
        case 'I': strcpy(checkmorse, ".."); break;
        case 'J': strcpy(checkmorse, ".---"); break;
        case 'K': strcpy(checkmorse, "-.-"); break;
        case 'L': strcpy(checkmorse, "-.."); break;
        case 'M': strcpy(checkmorse, "--"); break;
        case 'N': strcpy(checkmorse, "-."); break;
        case 'O': strcpy(checkmorse, "---"); break;
        case 'P': strcpy(checkmorse, "--."); break;
        case 'Q': strcpy(checkmorse, "--.-"); break;
        case 'R': strcpy(checkmorse, "-."); break;
        case 'S': strcpy(checkmorse, "..."); break;
        case 'T': strcpy(checkmorse, "-"); break;
        case 'U': strcpy(checkmorse, "..-"); break;
        case 'V': strcpy(checkmorse, "...-"); break;
        case 'W': strcpy(checkmorse, "--."); break;
        case 'X': strcpy(checkmorse, "-.-"); break;
        case 'Y': strcpy(checkmorse, "-.-"); break;
        case 'Z': strcpy(checkmorse, "--."); break;
        default: return 0; /* Return 0 if a character isn't found */
    }
}

```

```
int checkchar(int mrsindex, char seqmorse[], char checkmorse[])
{
    int i=0;    /* Starts index at the first position of checkmorse string */

    while(checkmorse[i]!='\0')
    {
        if(checkmorse[i]!=seqmorse[mrsindex]) /* Returns 0 if character isn't found in the morse sequence */
            return 0;

        i++;
        mrsindex++;
    }

    return mrsindex; /* Returns new mrsindex to checkword function */
}
```

# Conclusão

O problema foi resolvido com sucesso graças ao uso da recursividade, o que permitiu testar as frases válidas ignorando um grande número de combinações que não dariam resultado.

A implementação em C, ao contrário do algoritmo, foi o processo mais trabalhoso, nomeadamente na implementação recursiva da função que verifica palavras. Tive dificuldades em descobrir as condições em que seria feita a nova chamada à função.

As restantes funções foram fáceis de elaborar graças à semelhança das que construímos nas aulas práticas.