

Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

Linguagem de Programação

Cifra de Caesar e Vigenère

Guilherme Marques de Oliveira – 201835008

Gabriel Bronte Cardoso – 201835002

Professor: Leonardo Vieira dos Santos Reis

Juiz de Fora

Maio - 2022

Conteúdo

1	Organização de pastas	1
2	Estrutura dos fatos	1
3	Gerenciar dados	2
4	Funcionalidades	2
5	Predicados Auxiliares	5
6	Funcionalidades não implementadas	5
7	Dificuldades enfrentadas	6
8	Como executar	7

1 Organização de pastas

Fizemos a separação do código em cinco arquivos:

- Predicados.pl - Responsável por armazenar os predicados auxiliares que são utilizados nos demais arquivos.
- Palavras.pl - Responsável por armazenar os predicados que contém a base de palavras utilizadas para garantia do funcionamento dos algoritmos.
- Code.pl - Responsável por armazenar o predicado que contém os caracteres e seus respectivos códigos.
- Caesar.pl - Responsável por codificar, decodificar e decifrar textos da língua portuguesa usando as cifras de César.
- Vigenere.pl - Responsável por codificar, decodificar e decifrar textos da língua portuguesa usando as cifras de Vigenere.

2 Estrutura dos fatos

- code(Char, Code) : relaciona um caractere com o seu respectivo código.
- palavra(Palavra) : representa uma palavra da base de dados.

3 Gerenciar dados

O gerenciamento dos dados na base de palavra foi realizado através do predicado **writefacts(Palavra)** no qual o usuário pode digitar a palavra que deseja inserir na base de dados.

Este predicado é responsável por realizar a persistência dos dados, tanto em memória, através do assertz, quanto no arquivo Palavras.pl.

```
?- palavra('programar').  
false.  
  
?- writefacts('programar').  
true.  
  
?- palavra('programar').  
true.  
  
?-
```

Figura 1: Adição de palavras na base de dados

4 Funcionalidades

Quando a execução iniciar, o usuário pode utilizar as seguintes funcionalidades:

- **caesar(String, Char, Encodeds)**: passando uma String e um caractere presente nos fatos, o Encodeds retornará a frase criptografada pelo Char. Para fazer a desconversão o usuário passará o caractere, e no Encodeds passará uma frase criptografada, recebendo na variável String a frase original.

```
2 ?- caesar('a ligeira raposa marrom saltou sobre o cachorro cansado','c',X).  
X = "dcoljhludcudsrvcpcduurpcvdowrxvrehcrcfdfkruurcfdqvdgr".
```

Figura 2: Codificação de uma mensagem dado um caractere de deslocamento

```
1 ?- caesar(X,'c','dcoljhludcudsrvcpcduurpcvdowrxvrehcrcfdfkruurcfdqvdgr').  
X = "a ligeira raposa marrom saltou sobre o cachorro cansado"
```

Figura 3: Decodificação de uma mensagem dado um caractere de deslocamento

- `quebra_caesar(String,X)`: Passando uma frase codificada para a variável `String`, será buscado um caractere dentre todos os presentes na base de dados que descriptografe tal frase, de forma a buscar palavras que estejam presentes na base de fatos. Ao descriptografar a frase, certos caracteres especiais serão removidos ('!', ', ' , '. ' , etc), após isso, a nova frase será separada em uma lista, no qual cada posição é uma possível palavra e verificará uma a uma se estão presentes na base de dados. Caso todas as palavras sejam encontradas, será mostrado qual caractere que codificou a frase original.

```
1 ?- quebra_caesar('dcoljhludcudsrvcpcduurpcvdowrxvcvrehcrcfdkruurcfdqvgr',X).
X = c .
```

Figura 4: Decodificação de uma frase, e mostrando qual caractere que a codificou (mesma frase utilizada no exemplo acima)

- `vigenere(String, String_Key, Encodeds)`: passando uma `String` e uma `String_Key` com caracteres presentes nos fatos, o `Encodeds` retornará uma nova `String` após ter sido criptografada utilizando o códigos dos caracteres presentes na `String_Key`. Para fazer a desconversão o usuário passará a `String_Key`, e no `Encodeds` passará uma frase criptografada, recebendo na variável `String` a frase original

```
?- vigenere('flamengo campeão','lua',X).
X = "rGbyzosJaovnBzbA".

?- █
```

Figura 5: Codificação de uma mensagem dada uma chave de deslocamento

```
?- vigenere(X,'lua','rGbyzosJaovnBzbA').
X = "flamengo campeão" .

?- █
```

Figura 6: Decodificação de uma mensagem dada uma chave de deslocamento.

- `quebra_vigenere(String,N,X)`: Passando uma frase codificada para a variável `String`, o tamanho da chave de deslocamento para a variável `N`, será buscado uma string contendo cada combinação possível de caracteres presentes na base de dados para que a frase seja descriptografada, de forma a buscar palavras que estejam presentes na base de fatos. Ao descriptografar a frase, certos caracteres especiais serão removidos ('!', ', ', '. ', etc), após isso, a nova frase será separada em uma lista, no qual cada posição é uma possível palavra e verificará uma a uma se estão presentes na base de dados. Caso todas as palavras sejam encontradas, será mostrado qual chave de deslocamento que codificou a frase original.

```
?- vigenere('flamengo campeao','lua',X).  
X = "rGbyzosJaovnBzbA".  
  
?- quebra_vigenere('rGbyzosJaovnBzbA',3,X).  
X = "lua".  
  
?-
```

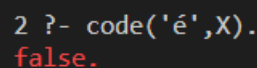
Figura 7: Decodificação de uma mensagem dado o tamanho da chave de deslocamento, e mostrando qual chave que a codificou.

5 Predicados Auxiliares

- `search_words(Lista)`: Recebe uma lista de palavras e verifica se todas as palavras estão presentes na base de dados, caso uma não esteja, falha na unificação.
- `delMember(Char, Lista, Listresult)`: Remove de uma lista todas as ocorrências de um dado caractere.
- `organize_string(String, Resultstring)`: Recebe uma String e retorna uma nova String com todos os caracteres especiais que não compõem uma palavra.
- `string2code(List1, List2)`: Recebe uma lista de caracteres e a converte em uma nova lista de códigos dos respectivos caracteres.
- `string_to_list_of_characters(String, Characters)`: Recebe uma String e a converte em uma lista de caracteres.
- `sum_char_code(Code, List, ListResult)`: Recebe um código, uma lista e realiza a soma de cada carácter da lista com o código.
- `sub_char_code(Code, List, ListResult)`: Recebe um código, uma lista e realiza a subtração de cada carácter da lista com o código.

6 Funcionalidades não implementadas

- Mapeamento de caracteres com acentuação: Letras acentuadas estão com problemas para serem codificadas no Windows, o predicado `code(Char, Code)` resulta em False, quando passado uma letra acentuada. Sendo assim, removemos os caracteres que não funcionavam, para que tanto no ambiente Windows quanto Linux não ocorressem problemas.



```
2 ?- code('é', X).  
false.
```

Figura 8: Exemplo de letra acentuada retornando falso no ambiente Windows, mesmo estando nos fatos.

- Persistência de remoção de palavras: Para persistir modificações na base de dados após o uso do sistema, utilizamos o arquivo Palavras.pl para salvar os fatos, porém não conseguimos implementar a funcionalidade de remoção de uma linha em arquivo.

```
?- palavra(presunto).  
true.  
  
?- deletefacts(presunto).  
true.  
  
?- palavra(presunto).  
false.  
  
?-
```

Figura 9: Remoção de palavras em memória.

```
?- palavra(presunto).  
true.  
  
?-
```

Figura 10: Remoção de palavras, após o reuso do sistema.

7 Dificuldades enfrentadas

- Desenvolver um predicado para codificar e decodificar uma cifra, para conseguirmos solucionar esse problema utilizamos o predicado `non-var(X)` que verifica se uma dada variável `X` é livre, a partir desse predicado conseguimos escolher se iremos prosseguir o processo de unificação com a codificação ou decodificação.
- Entender o conceito de corte e aplicá-lo corretamente foi uma tarefa complexa, no predicado `organize_string(Z,X)` foi necessário adicionar um corte logo após ele, pois o próximo predicado `search_words(Listadepalavras)`, falhava e retrocedia para o `organize_string` sem necessidade, repetindo novamente o processo em um loop eterno.

8 Como executar

Para executar o programa basta abrir a pasta contendo os arquivos e executar no SWI-prolog a seguinte linha de comando no terminal:

swipl vigenere.pl ou **swipl caesar.pl**

Obs: é preciso ter o swipl instalado no computador.

ppMMnndaOHmCvaHxwCvazN