Documentação do Trabalho 1 de Linguagem de Programação

Departamento de Ciência da Computação DCC - UFJF

Professor: Leonardo Vieira dos Santos Reis

Nome: Guilherme Fiorini Justen

Matrícula: 201965041AC

Para fazer a codificação de letras para números, no alfabeto de letras foram consideradas apenas as 26 letras do alfabeto (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z). Para usar as cifras para fazer a encriptação e decriptação, cada letra tem um número associado, a letra a é igual a 1, b = 2, …, z = 26.

No código, já constam 100 palavras que estão em texto claro. Além destas palavras, novas palavras podem ser inseridas na base de dados usando o predicado assert_word(Palavra). Além deste predicado, podem ser inseridas palavras na base de dados que vão ser encriptadas antes de serem inseridas. Para isso, o usuário deve optar entre a Cifra de César, que tem o predicado assert_encripted_cesar(Palavra, Chave), ou pela Cifra de Vigenère, que tem o predicado assert_encripted_vigenere(Palavra, Chave).

Para que o código tivesse uma redigibilidade melhor, foram implementados predicados auxiliares, que desempenham tarefas simples como pegar o primeiro elemento de uma lista, concatenar uma lista com outra, etc. Foram também criados os predicados string2code e code2string, que transformam, respectivamente, uma string em sua lista de numerais equivalentes, e uma lista de numerais em sua string equivalente. Todos estes predicados foram construídos usando a manipulação de listas.

A base de dados na qual podem ser inseridas novas palavras, que foi citada anteriormente, vem da biblioteca persistency, que é uma biblioteca que permite com que cláusulas adicionadas em tempo de execução e que não estejam presentes no código fonte continuem na base de dados após o término da execução do programa. A biblioteca faz isso através de um arquivo exterior, que é criado caso o usuário não tenha ele na página do código fonte. O nome do arquivo a ser criado é database.db.

Cifra de César

Para fazer a encriptação com a Cifra de César, foi usado o predicado encript_cesar. Ele recebe a palavra a ser encriptada, e uma chave numérica. A encriptação é feita mapeando a string de input para uma lista de números equivalentes, e somando o valor da chave em cada um dos elementos da lista, e depois essa lista é novamente convertida para uma string. Foi criado também um predicado para inserir palavras encriptadas com Cifra de César na base de dados. O predicado é assert_encripted_cesar.

A decriptação de uma Cifra de César, foi feita com força bruta. Visto que o alfabeto da base de dados contém 26 caracteres, serão testadas 26 chaves diferentes na palavra a ser buscada, para ver se ela consta na base. O código começa testando com a chave 0, texto plano. Depois ele incrementa de um em um o valor de cada caracter da string de entrada, até encontrá-la ou até chegar em 26. O predicado criado para esta decriptação foi o decript_cesar.

Cifra de Vigenère

Para fazer a encriptação com a Cifra de Vigenère, foi usada uma lógica parecida com a usada na Cifra de César, porém, nesta cifra é necessário fazer mais conversões e tratamentos, visto que a chave também é uma string. A chave e a string de input são convertidas para lista de numerais, e depois cada caracter da string de input vai ser somado com o caracter adequado da chave passada. Existem tratamentos para que quando a chave for completamente percorrida, a contagem recomece do primeiro elemento da chave. Isso foi implementado usando um predicado auxiliar de aridade 4, onde um dos parâmetros é uma cópia da chave original, que será usada para reconstruir a chave para quando acabarem os elementos da chave original.

A decriptação com Cifra de Vigenère não foi implementada. Embora tenha tentado fazer o código funcionar, vários erros foram encontrados. A minha primeira tentativa foi implementar os quatro predicados que constam na especificação do trabalho, porém, não consegui implementá-los. Depois, tentei escrever o código com uma lógica que usava força bruta, onde, sabendo que N era o tamanho da chave, eu criava uma chave com N caracteres ‘a’, e fazia a encriptação com a Cifra de Vigenère, e logo depois buscava a palavra na base de dados, caso a palavra não fosse encontrada, a chave era atualizada, por meio de um predicado que pegava a soma de todos os elementos da chave (se a chave fosse ‘aaa’, a soma seria 1 + 1 + 1 = 3), e os distribuía entre os caracteres de uma lista com N elementos. Na próxima chamada do predicado, o valor era incrementado em 1, até chegar em N*26, que cobre todas as possíveis chaves de tamanho N. Porém, foram encontrados muitos erros em diversas partes da construção do predicado, então acabei desistindo.

Interação com o usuário

Todo o código fonte está no arquivo index.pl, para executá-lo via terminal, basta estar na pasta do arquivo fonte e inserir o seguinte comando: ‘swipl index.pl’.

Ao entrar no ambiente do swi-prolog, o usuário pode interagir com o sistema usando os seguintes predicados:

- encript_cesar(Palavra, Chave, Resultado): A palavra é encriptada pela Cifra de Ceśar, e mostrada no terminal. A chave é um inteiro. O resultado da encriptação é mostrado na variável Resultado.

- encript_vigenere(Palavra, Chave, Resultado): A palavra é encriptada pela Cifra de Vigenère, e mostrada no terminal. A chave é uma string. O resultado da encriptação é mostrado na variável Resultado.

- retract_word(Palavra): Remove a palavra da base de dados.

- assert_word(Palavra): Insere uma palavra na base de dados.

- assert_encripted_cesar(Palavra, Chave): Insere uma palavra encriptada por Cifra de César na base de dados.

- assert_encripted_vigenere(Palavra, Chave): Insere uma palavra encriptada por Cifra de Vigenère na base de dados.

- word(Palavra): Verifica se uma palavra está na base de dados.

- decript_cesar(Palavra, Resultado): A palavra é buscada na base de dados através de busca por Cifra de César. Se uma ou mais palavras existirem, elas serão mostradas na variável Resultado.