USP – ICMC – SSC

SSC0603 – Estrutura de Dados – Trabalho 1: Criptografia com Chave Simétrica v1.2

Implemente um software que possa encriptar e decriptar uma mensagem, seguindo as especificações abaixo:

1) O software deve ser capaz de encriptar e decriptar uma mensagem, utilizando apenas uma chave simétrica (https://bit.ly/2yE09mP), ou seja, os processos de encriptação e decriptação devem utilizar a mesma chave.
2) As regiões de memória que guardarão os dados de entrada (mensagem e chave) devem ser alocadas dinamicamente, por meio de lista encadeada, fazendo o uso de ponteiros.
3) Cada nó da lista deve conter exatamente um caractere.
4) Os caracteres permitidos serão apenas os alfabéticos minúsculos, i.e., caracteres de ‘a’ a ‘z’, ou 97 a 122, pela tabela ASCII.
5) Não é permitida a existência de nós com dados inválidos ou nós avulsos que não participarão do processo de encriptação/decriptação de alguma forma.
6) O algoritmo de encriptação (Apêndice A) é composto por:
    a) Inserção de caracteres da chave na mensagem a ser encriptada em intervalos específicos.
    b) Uso da Cifra de César com deslocamento progressivo (base do algoritmo: https://en.wikipedia.org/wiki/Caesar_cipher; a diferença é que o número do deslocamento muda de acordo com um padrão) para a direita.
7) O algoritmo de decriptação (Apêndice B) é composto pelas funções inversas do item 6, ou seja:
    a) Uso da Cifra de César com deslocamento progressivo para a esquerda.
    b) Remoção de caracteres da chave em intervalos específicos.
8) O padrão de entrada é:
    a) Modo de execução:
        i) Número 0 para encriptar.
        ii) Número 1 para decriptar.
    b) Mensagem a ser processada, de acordo com o modo de execução.
    c) Chave simétrica.

d) A leitura da mensagem encriptada/decriptada e chave deve ser feita usando “scanf” ou função de comportamento semelhante.
e) Nos dois modos de execução, caracteres inválidos devem ser desconsiderados, incluindo ‘\n’ e ‘\r’.

9) O padrão de saída é:
   a) Tamanho da mensagem de entrada, considerando apenas os caracteres válidos.
   b) Tamanho da mensagem encriptada/decriptada.
   c) Mensagem encriptada/decriptada.
   d) A impressão da mensagem encriptada/decriptada deve ser feita usando “printf” ou função de comportamento semelhante.
   e) Uma quebra de linha deve ser inserida entre cada um dos itens anteriores.
10) Todas as regiões de memória alocadas dinamicamente DEVEM ser liberadas antes do encerramento da execução.
11) COMENTAR O CÓDIGO!!!

Apêndice A

```
/*-----------------------------------------------------INÍCIO-----------------------------------------------------*/

/*-------------------------------------------------Cálculo do deslocamento-------------------------------------------------*/

deslocamento = 0 // Variável inteira qualquer.
no_chave = lista_chave.inicio // Inicialização de uma variável que aponta para o primeiro nó da chave.

enquanto (no_chave != NULL): // Enquanto todos os nós da chave não forem visitados.
  ||  deslocamento = deslocamento ^ no_chave.caractere // "^" é o operador XOR bit a bit do C.
  ||  no_chave = no_chave.proximo // Avanço em uma posição na lista que contém a chave.
FIM enquanto

deslocamento = deslocamento % 26   // Esta linha faz com que o deslocamento na Cifra de César não seja
                                   // maior que a quantidade de caracteres no nosso alfabeto.

/*-------------------------------------------------Cálculo dos intervalos-------------------------------------------------*/

// Os intervalos representam a quantidade de nós da mensagem que serão “pulados” até que um nó da chave seja inserido.

// Note que a variável “intervalos” é um vetor de inteiros, que deve ser inicializado com o tamanho da quantidade de
// nós presentes na lista de chave.

i = 0 // Iterador usado na indexação dos intervalos.
no_chave = lista_chave.inicio // Inicialização de uma variável que aponta para o primeiro nó da chave.

enquanto (no_chave != NULL): // Enquanto todos os nós da chave não forem visitados.
  ||  intervalos[i] = (no_chave.caractere - 97) ^ deslocamento   // Inicialização de todos os intervalos a partir
  ||                                                            // da chave. A subtração “- 97” serve para
  ||                                                            // deslocar os caracteres para o intervalo [0, 26[.
  ||  no_chave = no_chave.proximo // Avanço em uma posição na lista que contém a chave.
  ||  i++
FIM enquanto

/*-------------------------------------------------Inserção dos nós da chave-------------------------------------------------*/

contador_intervalo = 0 // Contador de nós que auxiliará na inserção de nós chave na mensagem.
seletor_intervalo = 0 // Indexador do vetor de intervalos.
```

```
no_chave = lista_chave.inicio // Inicialização de uma variável que aponta para o primeiro nó da chave.
no_mensagem = lista_mensagem.inicio // Inicialização de uma variável que aponta para o primeiro nó da mensagem.

enquanto (no_mensagem != NULL): // Enquanto todos os nós da mensagem não forem visitados.
  ||  se (contador_intervalo >= intervalos[seletor_intervalo % intervalos.tamanho_vetor]):
  ||    ||  // A leitura da condicional acima é:
  ||    ||  // se “a quantidade de nós pulados” >= “valor indexado na posição ‘seletor_intervalo’ módulo
  ||    ||  // ‘intervalos.tamanho_vetor’ do vetor ‘intervalos’”
  ||    ||  // “intervalos.tamanho_vetor” é um inteiro que representa o tamanho do vetor “intervalos”.
  ||    ||  // Note que o acesso do vetor de intervalos é feito de maneira circular, por causa do módulo.
  ||    ||
  ||    ||  no_mensagem.inserir_apos(no_chave) // Inserção de um nó da chave na mensagem.
  ||    ||                                     // Recomenda-se a cópia de no_chave, antes da inserção na mensagem.
  ||    ||
  ||    ||  contador_intervalo = 0 // Redefinição do contador.
  ||    ||  seletor_intervalo++ // Avanço em uma posição do vetor de intervalos.
  ||    ||
  ||    ||  no_chave = no_chave.proximo // Avanço de um nó na lista da chave.
  ||    ||  no_mensagem = no_mensagem.proximo // Avanço de um nó na lista da mensagem.
  ||    ||
  ||    ||  se (no_chave == NULL):  // Esta condicional verifica o fim da lista de chave.
  ||    ||    ||                    // É possível usar lista circular para evitar esta condicional.
  ||    ||    ||  no_chave = lista_chave.inicio // Volta-se ao início da lista.
  ||    ||  FIM se
  ||  FIM se
  ||
  ||  no_mensagem = no_mensagem.proximo // Avanço de um nó na lista da mensagem.
  ||  contador_intervalo++
FIM enquanto

/*-----------------------------------------Aplicação da Cifra de César-----------------------------------------*/

no_mensagem = lista_mensagem.inicio // Inicialização de uma variável que aponta para o primeiro nó da mensagem.

enquanto (no_mensagem != NULL):
  ||  no_mensagem.caractere = (((no_mensagem.caractere - 97) + deslocamento) % 26) + 97 // Mágica de encriptação.
  ||
  ||  no_mensagem = no_mensagem.proximo // Avanço de um nó na lista da mensagem.
  ||
  ||  deslocamento++
```

```
FIM enquanto

/*----------------------------------------------------FIM-----------------------------------------------------*/
```

Apêndice B

```
/*-----------------------------------------------------INÍCIO-----------------------------------------------------*/

/*-------------------------------------------Cálculo do deslocamento-------------------------------------------*/

deslocamento = 0 // Variável inteira qualquer.
no_chave = lista_chave.inicio // Inicialização de uma variável que aponta para o primeiro nó da chave.

enquanto (no_chave != NULL): // Enquanto todos os nós da chave não forem visitados.
  ||  deslocamento = deslocamento ^ no_chave.caractere // "^" é o operador XOR bit a bit do C.
  ||  no_chave = no_chave.proximo // Avanço em uma posição na lista que contém a chave.
FIM enquanto

deslocamento = deslocamento % 26   // Esta linha faz com que o deslocamento na Cifra de César não seja
                                   // maior que a quantidade de caracteres no nosso alfabeto.

/*-----------------------------------Aplicação da Cifra de César em sentido reverso-----------------------------------*/

aux_deslocamento = deslocamento // Variável inteira inicializada com o valor “deslocamento”.

no_mensagem = lista_mensagem.inicio // Inicialização de uma variável que aponta para o primeiro nó da mensagem.

enquanto (no_mensagem != NULL): // Enquanto todos os nós da mensagem não forem visitados.
  ||  aux_caractere = (no_mensagem.caractere - 97) - (aux_deslocamento % 26) // Mágica de encriptação.
  ||
  ||  // As condicionais abaixo servem para deslocar os caracteres para os devidos domínios.
  ||  se aux_caractere < 0:
  ||    ||  no_mensagem.caractere = aux_caractere + 26 + 97
  ||  FIM se
  ||
  ||  senão:
  ||    ||  no_mensagem.caractere = aux_caractere + 97
  ||  FIM senão
  ||
  ||  no_mensagem = no_mensagem.proximo // Avanço de um nó na lista da mensagem.
  ||  aux_deslocamento++
FIM enquanto
```

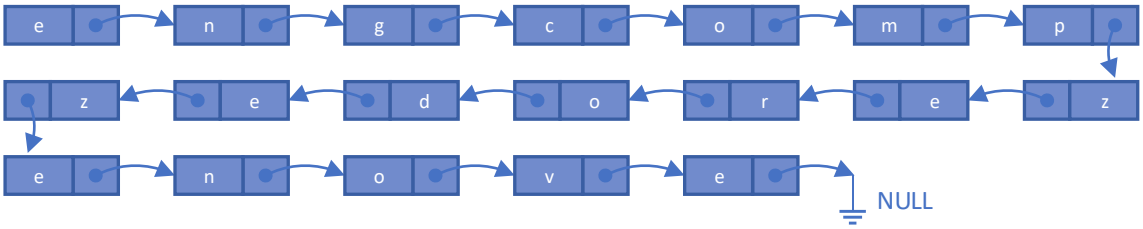
```
/*-----------------------------------------Cálculo dos intervalos-----------------------------------------*/

i = 0
no_chave = lista_chave.inicio // Inicialização de uma variável que aponta para o primeiro nó da chave.

enquanto (no_chave != NULL): // Enquanto todos os nós da chave não forem visitados.
  ||  intervalos[i] = (no_chave.caractere - 97) ^ deslocamento   // Inicialização de todos os intervalos a partir
  ||                                                            // da chave. A subtração "- 97" serve para
  ||                                                            // deslocar os caracteres para o intervalo [0, 26[.
  ||
  ||  no_chave = no_chave.proximo // Avanço em uma posição na lista que contém a chave.
  ||  i++
FIM enquanto

/*-------------------------------------------Remoção dos nós da chave-------------------------------------------*/

contador_intervalo = 0 // Contador de nós que auxiliará na inserção de nós chave na mensagem.
seletor_intervalo = 0 // Indexador do vetor de intervalos.
no_mensagem = lista_mensagem.inicio // Inicialização de uma variável que aponta para o primeiro nó da mensagem.

enquanto (no_mensagem != NULL): // Enquanto todos os nós da mensagem não forem visitados.
  ||  // A condicional abaixo tem comportamento semelhante à "Inserção dos nós da chave" do apêndice A.
  ||  se (contador_intervalo >= intervalos[seletor_intervalo % intervalos.tamanho_vetor]):
  ||    ||  no_mensagem = lista_mensagem.remover(no_mensagem)    // Remoção de no_mensagem e atualização do ponteiro.
  ||    ||                                                       // Agora, no_mensagem aponta para o próximo nó.
  ||    ||
  ||    ||  contador_intervalo = 0
  ||    ||  seletor_intervalo++
  ||    ||
  ||    ||  no_mensagem = no_mensagem.proximo // Avanço de um nó na lista da mensagem.
  ||  FIM se
  ||
  ||  no_mensagem = no_mensagem.proximo // Avanço de um nó na lista da mensagem.
  ||
  ||  contador_intervalo++
FIM enquanto

/*-----------------------------------------------------FIM-----------------------------------------------------*/
```

Exemplo:

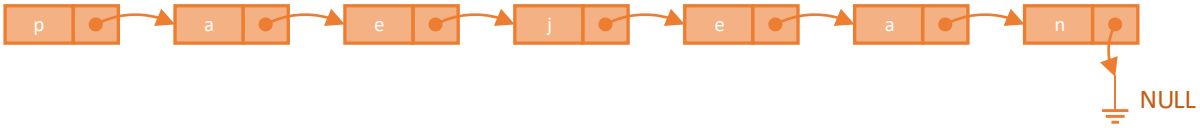
```
Entrada:
0
engcompzerodezenove
paejean

Saída:
19 // Tamanho da mensagem original.
21 // Tamanho da mensagem encriptada.
qauesfeitznlbdzfcqsak // Mensagem encriptada.
```

Após a leitura das entradas, mensagem e chave se organizarão de forma semelhante aos diagramas abaixo:

Mensagem



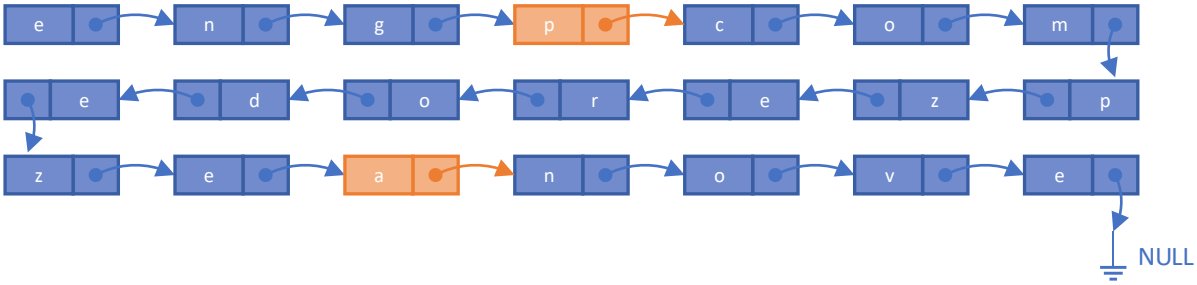
Chave



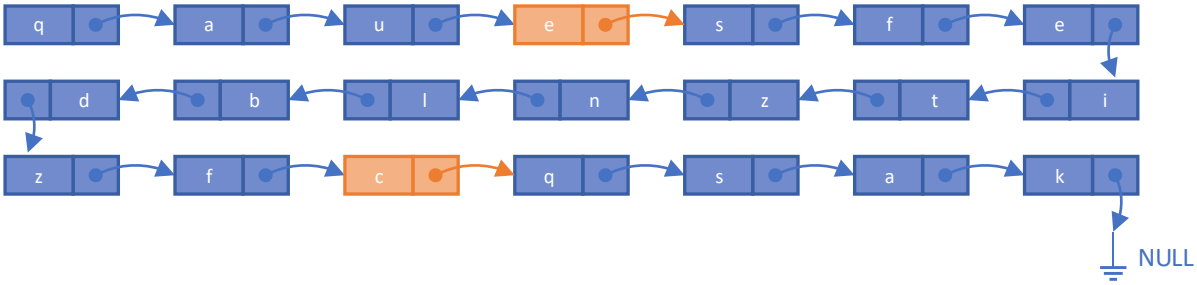
Seguindo o fluxo do algoritmo descrito no Apêndice A, calcula-se o deslocamento que será usado para a geração dos intervalos e posterior deslocamento na Cifra de César. O cálculo é feito pela operação (0 ^ ‘p’ ^ ‘a’ ^ ‘e’ ^ ‘j’ ^ ‘e’ ^ ‘a’ ^ ‘n’) módulo 26 ou, usando o valor decimal da tabela ASCII, (0 ^ 112 ^ 97 ^ 101 ^ 106 ^ 101 ^ 97 ^ 110) módulo 26, resultando no valor 12.

Fazendo as operações descritas em “Cálculo dos intervalos” do Apêndice A, tem-se o vetor de intervalos:

Os valores acima podem ser interpretados como quantidades de nós que deverão ser pulados até inserir um nó da chave. Para o nosso exemplo, o “enxerto” dos nós da chave ficará assim:



Como último passo, faz-se a rotação dos caracteres com um deslocamento inicial (para este exemplo, o deslocamento inicial é o 12 calculado anteriormente), sendo incrementado pelo índice da lista, i.e., 12 para a posição 0, 13 para a posição 1, 14 para a posição 2...



O diagrama acima é a mensagem encriptada pelo algoritmo, devendo ser exibida pelo programa.

Exemplo:

Entrada:

```
1
qauesfeitznlbdzfcqsak
paejean
```

Saída:

```
21 // Tamanho da mensagem original.
19 // Tamanho da mensagem decriptada.
engcompzerodezenove // Mensagem decriptada.
```

O caso acima faria o processo reverso da encriptação, como descrito no Apêndice B:

- Rotação em sentido oposto.
- Remoção dos nós inseridos na encriptação.