USP – ICMC – SSC

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

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
no_chave = lista_chave.inicio

enquanto (no_chave != NULL):
  ||  deslocamento = deslocamento ^ no_chave.caractere // "^" é o operador XOR bit a bit do C.
  ||  no_chave = no_chave.proximo
FIM enquanto

deslocamento = deslocamento % 26

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

i = 0
no_chave = lista_chave.inicio

enquanto (no_chave != NULL):
  ||  intervalos[i] = (no_chave.caractere - 97) ^ deslocamento
  ||  no_chave = no_chave.proximo
  ||  i++
FIM enquanto

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

contador_intervalo = 0
seletor_intervalo = 0
no_chave = lista_chave.inicio
no_mensagem = lista_mensagem.inicio

enquanto (no_mensagem != NULL):
  ||  se (contador_intervalo >= intervalos[seletor_intervalo % intervalos.size]):
  ||    ||  no_mensagem.inserir_apos(no_chave) // Recomenda-se a cópia de no_chave, antes da inserção na mensagem.
  ||    ||
  ||    ||  contador_intervalo = 0
```

```
  ||    ||  seletor_intervalo++
  ||    ||
  ||    ||  no_chave = no_chave.proximo
  ||    ||  no_mensagem = no_mensagem.proximo
  ||    ||
  ||    ||  se (no_chave == NULL): // É possível usar lista circular para evitar esta condicional.
  ||    ||    ||  no_chave = lista_chave.inicio
  ||    ||  FIM se
  ||  FIM se
  ||
  ||  contador_intervalo++
FIM enquanto

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

no_mensagem = lista_mensagem.inicio

enquanto (no_mensagem != NULL):
  ||  no_mensagem.caractere = (((no_mensagem.caractere - 97) + deslocamento) % 26) + 97
  ||  deslocamento++
FIM enquanto

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

Apêndice B

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

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

deslocamento = 0
no_chave = lista_chave.inicio

enquanto (no_chave != NULL):
  ||  deslocamento = deslocamento ^ no_chave.caractere // "^" é o operador XOR bit a bit do C.
  ||  no_chave = no_chave.proximo
FIM enquanto

deslocamento = deslocamento % 26

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

aux_deslocamento = deslocamento

no_mensagem = lista_mensagem.inicio

enquanto (no_mensagem != NULL):
  ||  aux_caractere = (no_mensagem.caractere - 97) - (aux_deslocamento % 26)
  ||
  ||  se aux_caractere < 0:
  ||    ||  no_mensagem.caractere = aux_caractere + 26 + 97
  ||  FIM se
  ||
  ||  senão:
  ||    ||  no_mensagem.caractere = aux_caractere + 97
  ||  FIM senão
  ||
  ||  aux_deslocamento++
FIM enquanto

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

i = 0
```

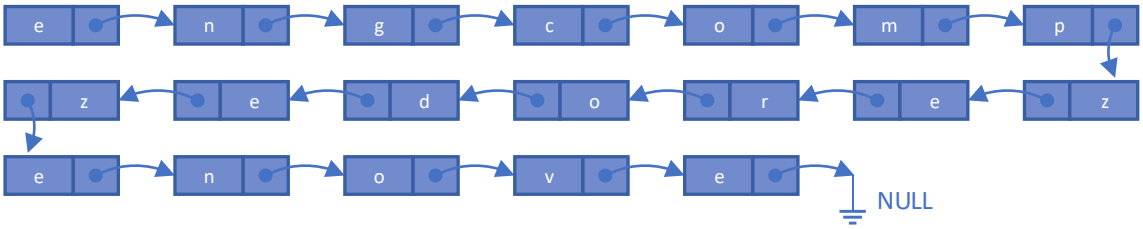
```
no_chave = lista_chave.inicio
enquanto (no_chave != NULL):
  ||  intervalos[i] = (no_chave.caractere - 97) ^ deslocamento
  ||  no_chave = no_chave.proximo
  ||  i++
FIM enquanto

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

contador_intervalo = 0
seletor_intervalo = 0
no_mensagem = lista_mensagem.inicio

enquanto (no_mensagem != NULL):
  ||  se (contador_intervalo >= intervalos[seletor_intervalo % intervalos.size]):
  ||    ||  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
  ||  FIM se
  ||
  ||  contador_intervalo++
FIM enquanto

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

Exemplo:

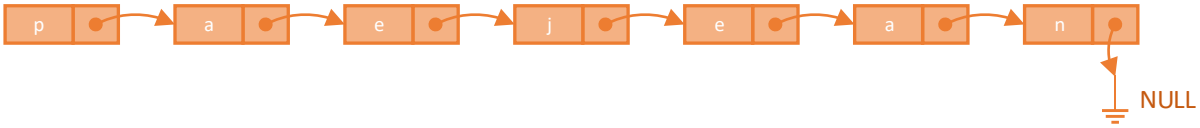
```
Entrada:
0
engcompzerodezenove
paejean

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



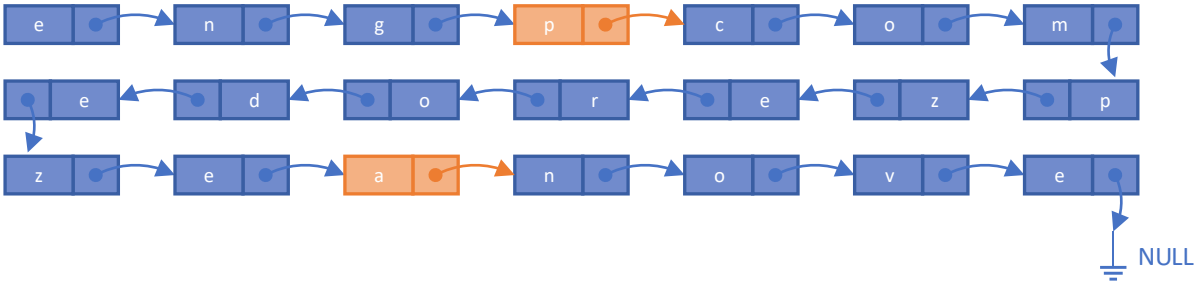
Chave



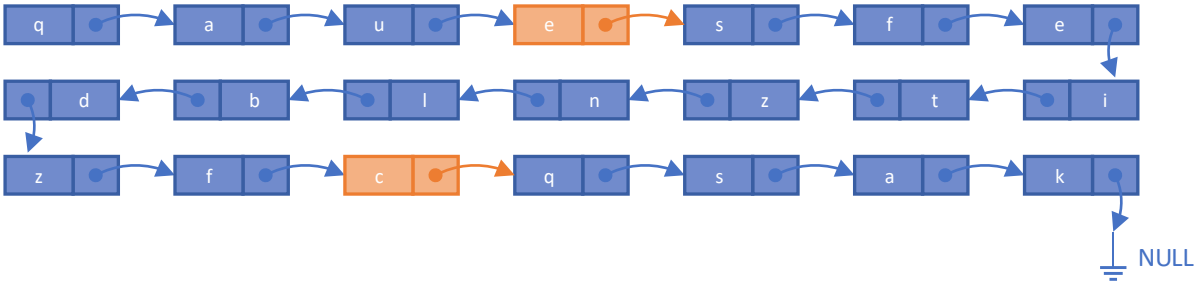
Seguindo o fluxo do algoritmo descrito no Apêndice A, calcula-se o deslocamento que será usado para a geração dos intervalos e posterior deslocamento na Cifra de César. O cálculo é feito pela operação (0 ^ ‘p’ ^ ‘a’ ^ ‘e’ ^ ‘j’ ^ ‘e’ ^ ‘a’ ^ ‘n’) módulo 26 ou, usando o valor decimal da tabela ASCII,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