Universidade de São Paulo – ICMC
Bacharelado em Ciência da Computação
SCC0221 – Introdução à Ciência da Computação I
Prof. Rodrigo Fernandes de Mello – mello@icmc.usp.br
Monitores: Filipe Mariano– filipe.mariano.silva@usp.br,
Igor Martinelli – igor.martinelli@usp.br,
Victor Forbes – victor.forbes@usp.br,
Yule Vaz – yule.vaz@usp.br

Trabalho 3

1 Descrição

Para esse trabalho, o seu programa deverá ser capaz de ler de um arquivo binário do disco e resolver 3 problemas com os dados contidos no arquivo. Em cada arquivo binário foi armazenada uma mensagem m criptografada. A criptografia da mensagem foi feita a partir dos seguintes passos:

- Um vetor v de n `unsigned int`s é alocado e zerado.
- Todos os bytes da mensagem m (incluindo o byte do caractere '\0') são copiados para esse vetor de `unsigned int`s.
- Elementos v_i do vetor v são selecionados aleatoriamente e todos os bits desses elementos são shiftados 1 vez para a esquerda.
- O vetor v é então armazenado de trás para frente em um arquivo binário. Seu primeiro elemento v_0 se torna o último, seu segundo elemento v_1 se torna o penúltimo e assim por diante.

Seu programa deve, em primeiro lugar, armazenar na memória Heap (em um vetor de `unsigned int`s) os n inteiros não negativos obtidos do arquivo **com uma única leitura** e desinverter esse vetor **usando uma função**.

Parte 1: Mapeamento de Dígitos

Além da mensagem criptografada, há nos dados um código secreto. O código é uma sequência de n inteiros formada pela concatenação dos dígitos d_{ik}, tal que d_{ik} é o dígito da posição k do i-ésimo elemento v_i do vetor v.

Os dígitos d_{xk} de um inteiro v_x não negativo são indexados da direita para a esquerda, como mostra a tabela abaixo (para $v_x = 347$).

```
                        Position k
                            v
 n   n-1  ...   2    1     0     <- Position indexes
---  ---  ---  ---  ---   ---
 0    0    0    3    4     7     <- Number vx
                             ^
                        Digit dxk
```

O número 347 possui o dígito 7 na posição 0, o dígito 4 na posição 1, o dígito 3 na posição 2 e 0 nas posições $k > 2$.

Crie uma função que obtenha esse código do vetor v. Crie também uma função que dado um `unsigned int` x e uma posição k, extraia o $(k+1)$ésimo dígito (considere que os inteiros são preenchidos com zeros à esquerda) de x. Para obter o dígito d_{ik} correto de cada elemento, seu programa deve seguir os seguintes passos:

- O primeiro dígito do código ($d_{0\text{k}}$) é o dígito que está na posição k do primeiro elemento do vetor, tal que k é o número dado no nome do arquivo (antes da extensão .bin) como entrada do programa. Se o nome do arquivo for 3.bin, por exemplo, o primeiro dígito do código será d_{03}, o quarto dígito do primeiro elemento do vetor v.
- A posição dos dígitos subsequentes será dada pelo dígito anterior. Utilizando o arquivo 3.bin, por exemplo, temos que $k = d_{03}$ para o dígito $d_{1\text{k}}$, ou $d_{1\text{k}} = d_{1(d_{03})}$.

Exemplo da execução do algoritmo

```
Filename: 3.bin

v        i = 0 / k = 3
2741
  v      i = 1 / k = 2
13043
 v       i = 2 / k = 0
31
    v    i = 3 / k = 1
131765

Resultado: 2016
```

Parte 2: Descriptografia

Agora que o vetor v não está mais de trás para frente, basta desfazer a bagunça feita com os bits. Para descriptografar a mensagem seu programa deve realizar o processo inverso do realizado na criptografia. **Crie uma função que "shifte" para a direita os bits dos elementos v_{i} do vetor v que foram shiftados para a esquerda durante a criptografia**.

Para descobrir quais inteiros tiveram seus bits shiftados para a esquerda seu programa precisa verificar se os bytes desses inteiros representam caracteres válidos, ou seja, caracteres que podem pertencer à mensagem (`A mensagem pode conter caracteres alfanuméricos, espaços, vírgulas, pontos, dois pontos, parênteses, pontos de exclamação, pontos de interrogação e o \0`).

Para facilitar, você pode criar uma função que recebe um `unsigned int` x e retorna 1 (verdadeiro) caso todos os bytes de x sejam caracteres válidos ou 0, caso contrário. Para fazer essa verificação, utilize um ponteiro `char *` para percorrer os bytes do inteiro x.

Agora que os bytes do vetor v são os próprios caracteres da mensagem m original, basta imprimí-la. É proibido copiar os dados do vetor v para outro local para realizar a impressão.

Parte 3: Contagem de ocorrências

Para realizar essa operação você precisa ter descriptografado a mensagem m na parte 2 do trabalho.

Leia da entrada padrão uma `string` s e imprima na tela a quantidade de ocorrências de s na mensagem m. Lembre-se de que você deve utilizar você ainda deve utilizar a mesma região de memória para onde os dados do disco foram copiados.

2 Entrada

A primeira linha da entrada contém o nome do arquivo binário que possuí os dados. A segunda linha da entrada contém o número da operação *op* a ser executada em cima dos dados do arquivo (1 para Mapeamento de Dígitos, 2 para Descriptografia e 3 para Contagem de Ocorrências).

Caso *op* seja 3 haverá uma terceira linha na entrada com a `string` s.

3 Saída

Caso *op* seja 1, imprima na tela o código obtido pelo Mapeamento de Dígitos. Caso *op* seja 2, imprima na tela a mensagem descriptografada. Caso *op* seja 3, imprima na tela a quantidade de ocorrências da string s na mensagem descriptografada m.

4 Instruções Complementares

- É obrigatório o uso de **apenas uma** região da memória Heap para armazenar os dados do arquivo binário (todas as operações necessárias devem ser executadas em cima dessa mesma região da memória).
- É obrigatória a criação de uma função para desinverter o vetor de `unsigned int`s e pelo menos uma função para cada uma das 3 partes do trabalho.
- Pesquise sobre as funções **ftell()** e **fseek()** para realizar a leitura do arquivo conforme o pedido.
- Comente seu código.

5 Exemplos

Entrada

```
0.bin
1
```

Saída

```
64410
```

Entrada

```
0.bin
2
```

Saída

```
This is an example!
```

Entrada

```
0.bin
3
is
```

Saída

```
2
```