Atividade de Programação 6

Compressão de Dados

Algoritmos e Estruturas de Dados I

Tempo limite: 1s

Um algoritmo muito simples utilizado na compressão de dados é o *Run-length encoding* (RLE), o qual consiste em fazer uma contagem simples de elementos (caracteres) iguais em uma *string*. Uma pequena variação deste consiste em eliminar possíveis ruídos da sequência. Os ruídos são caracterizados por *substrings* de tamanho igual à 1 (apenas um único caractere). Para eliminar o ruído, deve-se verificar os *substrings* vizinhos à esquerda e à direita deste caracter, e identificar qual destes dois *substrings* vizinhos tem o maior tamanho. Uma vez identificado o *substring* vizinho de maior tamanho, este terá o seu tamanho aumentado em 1 unidade (incorporando o ruído). Na sequência será removido o caracter considerado como ruído na lista. Entretanto, somente pode-se fazer a remoção do ruído em algumas condições:

- Caso o ruído encontrado esteja localizado no meio da lista e ambas as *substrings* vizinhas possuem tamanho maior ou igual a 2 caracteres.
- Caso o ruído seja encontrado na primeira posição da *string* de entrada, deve-se considerar apenas o *substring* vizinho da direita.
- Caso o ruído seja encontrado na última posição da *string* de entrada, deve-se considerar apenas o *substring* vizinho da esquerda.

Em caso de empate, ou seja, os dois *substrings* vizinhos têm a mesma quantidade de caracteres (tamanho), considera-se o *substring* vizinho da esquerda para receber o incremento no tamanho. Deve-se notar que as *strings* são lidas da esquerda para direita, considerando-se apenas sequências de letras maiúsculas ([A-Z]), sem números ou símbolos.

Deve-se escrever um programa que realize a compressão de uma *string* de letras maiúsculas, levando em consideração a eliminação de ruídos.

Entrada

A entrada para cada teste se inicia informado a *string* de entrada com tamanho **N** ($2 \leq N \leq 1023$) que vai ser comprimida. A *string* é composta apenas de letras maiúsculas ([A-Z]).

Saída

A saída do programa é exibida em quatro linhas:

- 1ª linha: Tamanho (quantidade de caracteres) da lista duplamente encadeada de *substrings,* gerada a partir da *string* de entrada, ainda sem as remoções;

- 2ª linha: Sequência da lista duplamente encadeada de *substrings,* gerada a partir da *string* de entrada, ainda sem as remoções;
- 3ª linha: Tamanho da nova *string* final com as remoções (se houver);
- 4ª linha: Sequência da lista duplamente encadeada de *substrings,* gerada com as remoções, como resultado final.

Dicas:

- Recomenda-se encontrar o tamanho da *string* original;
- O tamanho da *string* final, em alguns casos, pode ser maior que o original na técnica simples de compressão utilizada. Ex.:
    - *String* original de entrada: `BW` (tamanho 2)
    - Lista duplamente encadeada de *substrings* gerada sem remoções: `[1, B] ⇄ [1, W]` (tamanho 4).
- Alguns casos/exemplos de remoção de ruídos:
    - `CCBW => [2, C] ⇄ [1, B] ⇄ [1, W]` //Não pode fazer a remoção do *substring* `[1, B]` pois o vizinho da direita tem tamanho 1. *String* final `2C1B1W`.
    - `CCBWWWK => [2, C] ⇄ [1, B] ⇄ [3, W] ⇄ [1, K]` //Pode-se fazer a remoção das *substrings* `[1, B]` e `[1, K]` pois os *substrings* vizinhos tem valor tamanho maior que 1. String final `2C5W` (sentido: esquerda para direita).

Restrições:

- O programa deve ser escrito em C;
- Deve-se utilizar uma Lista Duplamente Encadeada para armazenar a *string* original na forma de uma lista de *substrings*;
- Deve-se ter no código as funções de inicialização, remoção, inserção na lista duplamente encadeada;
- A *string* é lida da esquerda para a direita para as remoções;

Exemplo de Entrada	**Exemplo de Saída**
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWW WWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWW WWW	18 12W1B12W3B24W1B14W 14 13W12W3B25W14W

Exemplo de Entrada	Exemplo de Saída
ABC	6 1A1B1C 6 1A1B1C

Exemplo de Entrada	Exemplo de Saída
AAAAAAAATTTTTTTTTTTUUUUUIIIIENN NNNNNNNWNNNNNNNNNN	18 8A12T5U4I1E9N1W10N 15 8A12T5U4I11N10N