A1 - Lista simplesmente Encadeada

Problema: *Smart Binary Strings* (SBS)

As sequências de strings binárias são strings gigantes formadas por 0's e 1's. Por exemplo, a sequência 00000000001111111111100101000000000 é uma string binária (contém apenas os caracteres '0' e '1'). O problema destas sequências é que ocupam muito espaço. Portanto, a ideia inicial é que poderíamos usar uma lista simplesmente encadeada para codificar a string binária em uma lista onde cada nó armazenaria o caractere e a quantidade de vezes que ele aparece.

Exemplo: `111111000000011100` geraria a seguinte lista:

```
[6, 1] -> [7, 0] -> [3, 1] ->  [2, 0] ->
```

Onde o primeiro campo indica a quantidade e o segundo qual caractere armazenado.

O cientista da computação Pedrinho teve uma ideia ainda melhor! Pedrinho chamou esta ideia de de *Smart Binary Strings (SBS)*. Nesta abordagem, decide-se qual caractere binário vai guardar (0 ou 1). Esta informação é fornecida na criação da lista (fica armazenada no nó descritor).

Neste problema a entrada é a mesma do problema anterior, porém agora apenas será armazenado um caractere (o primeiro da sequência) no nó descritor da Lista e os nós agora armazenam o índice onde aparece o caractere de referência. Portanto, as diferenças dos índices indicam a quantidade do caractere oposto ao caractere de referência. Assim, cada nó é composto pela quantidade de caracteres de referência a serem impressos a partir do índice (segunda informação no nó).

Exemplo 1: `111111000000011100` geraria a seguinte lista no codificador:

```
L(1) = [6, 0] -> [3, 13] -> [2, -1] →
```

índice	**0**	1	2	3	4	5	**6**	7	8	9	10	11	12	**13**	14	15	**16**	17
string	**1**	1	1	1	1	1	**0**	0	0	0	0	0	0	**1**	1	1	**0**	0

`L(1)` → indica que o caractere de referência (primeiro) é o '1'.

`[6, 0]` → o primeiro nó da lista indica que a partir do índice 0, há 6 caracteres de referência, neste caso '1'.

`[3, 13]` → o segundo nó da lista indica que a partir do índice 13, há 3 caracteres de referência, neste caso '1'. Ou seja, há 13-6 = 7 caracteres diferentes da referência (neste caso '0'), entre estes dois índices.

`[2, -1]` → o último nó da lista aparecerá com valor negativo indicando que ainda há 2 caracteres diferentes do caractere de referência (quando houver), para finalizar a string. Neste caso, no fim da string ainda há 2 caracteres '0' para finalizar a string.

Exemplo 2: `000110000` geraria a seguinte lista no codificador:

```
L(0) = [3, 0] -> [4, 5] ->
```

`L(0)` → indica que o caractere de referência (primeiro) é o '0'.

`[3, 0]` → o primeiro nó da lista indica que a partir do índice 0, há 3 caracteres de referência, neste caso '0'.

`[4, 5]` → o segundo (e último) nó da lista indica que a partir do índice 5, há 4 caracteres de referência, neste caso '0'. Ou seja, no intervalo 5-3= 2, há dois caracteres diferentes do caractere de referência (neste caso '1').

Além disso, é importante calcular e exibir a diferença entre a quantidade de bytes gastos pela sua estrutura lista e a string de entrada, no caso do codificador. Tomando como base que cada caractere na string possui 1 byte de custo, e que, cada nó da sua lista gastará 12bytes + o tamanho do nó descritor (estima-se 5 bytes, caso use apenas o endereço início e o caractere de referência). Portanto, quando usar o codificador, além de imprimir a lista é preciso imprimir a diferença entre bytes (tamanho da string – tamanho da lista), perceba que este valor pode ser negativo.

Portanto, seu programa se comportará como codificador e decodificador. Quando for chamado em modo codificador, vai ler a string de entrada, construir a lista SBS e exibi-la na tela conforme estipulado no formato da saída (a seguir), informando a quantidade economizada (ou desperdiçada) de bytes ao usar a codificação para a string. Quando seu programa trabalhar como decodificador, seu programa receberá a lista no formato e imprimirá a string binária.

Entrada

A primeira linha da entrada diz qual modo operacional seu programa trabalhará. Usaremos o caractere 'c' para codificador e 'd' para decodificador.

Entrada em modo de codificador:

Caso o primeiro caractere seja 'c' de codificador, a próxima linha consiste na string binária que pode ser até de 10000 caracteres (haverá um \n ao final, que deve ser removido da entrada).

Entrada em modo decodificador

Em modo decodificador, a entrada é formada por um caractere que é '0'ou '1' um espaço e o numero de pares a seguir. Cada par a seguir é formado pelas informações do nó, discutidas anteriormente.

Saída

Em modo de codificador:

A saída é formada por um caractere que é ‘0’ou ‘1’ um espaço e o numero de pares a seguir. Cada par a seguir é formado pelas informações do nó, discutidas anteriormente, separadas por espaço e por fim, a diferença em bytes da string de entrada e a quantidade de bytes usadas pela lista (não se esqueça do \n no final da última linha).

Em modo de decodificador:

A saída é formada pela string binária recuperada da lista (não se esqueça do \n no final).

Exemplos

Exemplo de entrada 1: c 1111110000000011100	**Exemplo de saída:** 1 3 6 0 3 13 2 -1 -23

Exemplo de entrada 2: c 000110000	**Exemplo de saída:** 0 2 3 0 4 5 -20

Perceba que nos dois exemplos acima, não foi compensador usar a lista, pois a quantidade de bytes usadas na lista foi maior.

Exemplo de entrada 3: d 1 3 6 0 3 13 2 -1	**Exemplo de saída:** 1111110000000011100

Exemplo de entrada 4:	**Exemplo de saída:**
d 0 2 3 0 4 5	000110000

IMPORTANTE: O uso de listas simplesmente encadeadas para resolução desta tarefa é **obrigatório**.