O objetivo desse exercício é implementar um compactador e um descompactador de arquivos usando codificação de *Huffman*.

A árvore de *Huffman* pode ser representada em *Haskell* usando o seguinte tipo algébrico de dados:

```
data Huffman = Folha Int Char | No Int Huffman Huffman
```

Declare uma função que dada uma *String* retorne uma lista de folhas da árvore contendo o caractere e a frequência com que esse caractere ocorreu na *String*:

```
freqSimb:: String → [Huffman]
```

Declare uma função que dada a lista com a frequência de cada caractere e retorne a árvore de *Huffman*:

```
construirArvore:: [Huffman] → Huffman
```

Declare uma função que dada a árvore de *Huffman* retorne os códigos de *Huffman* para cada caractere:

```
codHuffman:: Huffman → [(Char, String)]
```

Declare uma função que codifique uma *String* numa sequência binária de códigos de *Huffman*:

```
codificar:: String → Huffman → String
```

Declare uma função que faça o processo inverso que a função anterior:

```
decodificar:: String → Huffman → String
```

Declare uma função que receba como parâmetro o nome de um arquivo e compacte seu conteúdo usando a codificação de *Huffman*.

Declare uma função que receba como parâmetro o nome de um arquivo compactado e descompacte seu conteúdo.

Sugestão para o formato do arquivo compactado:

Word8	Word32		Word8	Word32	Word8	Word32		Word8	Word32	
n	t		c_1	f_1	c_2	f_2	…	c_n	f_n	
b_1	b_2	b_3	…						$b_{t/8}$	$b_{t/8+1}$

Sendo n o número símbolos (caracteres) distintos que ocorrem no arquivo original, t o total de caracteres do arquivo, c_i o *i-ésimo* símbolo do arquivo, f_i a frequência com que o símbolo c_i ocorre[1] no arquivo e b_s os bytes contendo os códigos de *Huffman*.

[1] A frequência de cada caracter é necessária para reconstruir a árvore de *Huffman*. Isso pode ser armazenado de forma mais eficiente usando código canônico de *Huffman*, mas para simplificar a implementação será aceito armazenar o símbolo e frequência