1. **Cifra de César.** Implemente em Haskell funções para codificação e decodificação para a cifra de César. Utilize o teste do chi-quadrado para auxiliar da decodificação.

   Codificação

   - Defina uma função que converte letras minúsculas em valores de 0 a 25 (let2int :: Char -> Int) e uma função (int2let :: Int -> Char) que faz o inverso
   - Usando estas duas funções, pode-se definir uma função shift :: Int -> Char -> Char que aplicar um fator de deslocamento para letras minúsculas
   - Usando shift , defina encode :: Int -> String -> String que codifica uma string dado um fator de deslocamento.

   Decodificação

   Para a decodificação, leva-se em consideração a observação de que algumas letras são usadas com maior frequência que outras (Tabela de frequência)

   - Definir uma função que calcula a porcentagem de um inteiro em relação a um outro, retornando um ponto flutuante (percent :: Int -> Int-> Float). Dica: usem a função fromIntegral para converter Int em Num
     - Exemplo: percent 5 15 = 33.33336
   - Usando percent defina uma função que retorna uma lista com a tabela de frequência para uma dada string (freqs :: String -> [Float]). Dica: use uma função auxiliar para devolver os caracteres minúsculos de uma string e uma outra função para contar o número de ocorrências de um determinado caracter minúsculo
     - Exemplo:freqs "abbcccddddeeeee" = [6.666667, 13.3333334, 20.0, 26.666668, ..., 0.0]. Ou seja, a letra 'a' ocorre com uma frequência de aproximadamente 6.6%
   - Um método padrão para comparar uma lista de frequências de valores observados com uma lista de frequência de valores esperados é o chi-quadrado

     $$\sum_{i=0}^{n-1} \frac{(os_i - es_i)^2}{es_i}$$

     Defina a função chisqr :: [Float] -> [Float] -> Float
   - Defina a função rotate :: Int -> [a] -> {a] que rotaciona os elementos de uma em n posições para esquerda
     - Exemplo: rotate 3 [1,2,3,4,5] = [4,5,1,2,3]
   - Suponha que seja dada uma string codificada, mas que conhecemos o fator de deslocamento utilizado na codificação, queremos descobrir este fator. Vamos produzir uma tabela de frequência da string codificada, calculando o chi-quadrado para cada possível rotação desta tabela com respeito à tabela de frequência esperada e usando a posição do menor valor chi-quadrado como fator de deslocamento.
     - Exemplo
       * encode 3 "haskell is fun" = "kdvnhoo lv ixq"
       * table'= freqs "kdvnhoo lv ixq"
       * [chisqr (rotate n table') table n <- [0..25]] = [1408.8524, 640.0218, 612.3969, 202.42024, ... , 626.4024]|, em que o valor mínimo, 202.42024, aparece na posição 3 nesta lista. Daí, concluímos que 3 é provavelmente o fator de deslocamento usado na decoficação

- Defina a função crack :: String -> String para decofficação
  - Exemplo: crack "kdvnhoo lv ixq" = "haskell is fun"

2. **Verificador de tautologia.** Desenvolva uma função que decide se proposições são sempre verdadeiras (tautologias). Considere que a linguagem de proposições é construída a partir de valores básicos (True e False), que vamos considerar como constantes, e variáveis (A, B, ..., Z) por meio de uso de negação, conjunção, implicação e parênteses.

   - O primeiro passo é definir um tipo algébrico para proposições, com um construtor para cada uma das possíveis formas que uma proposição pode ter (não é necessário um construtor para parênteses).
   - Para avaliar uma proposição para um valor lógico, é necessário conhecer o valor de cada uma das duas variáveis. Declare um tipo Subst como uma tabela de busca que associa nomes de variáveis a valores lógicos. Exemplo: [('A', False), ('B', True)]
   - Escreva uma função que avalia uma proposição dada uma substituição para as variáveis (eval :: Subst -> Prop -> Prop) . Utilize casamento de padrão. Por exemplo: o valor de uma proposição constante é a própria constante; o valor de uma variável é obtido por uma busca na tabela
   - Para decidir se uma proposição é uma tautologia, temos de considerar todas as possíveis substituições para variáveis contidas nela. Defina uma função que retorna todas as variáveis de uma proposição (vars :: Prop -> [Char])
   - Para gerar as substituições é necessário produzir listas de valores lógicos de um dado tamanho. Defina a função bools :: Int -> [[Bool]]. Exemplo: bools 3 retorna 8 listas com todas as combinações de valores lógicos.
   - Usando a função bools, defina a função substs :: Prop -> [Subst] que gera todas as possíveis substituições para uma proposição por meio da extração das variáveis, removendo duplicatas de variáveis desta lista, gerando todas as possíveis listas de valores lógicos para as variáveis e combinando com a lista de variáveis
     - p2 = Imply (And (Var 'A') (Var 'B')) (Var 'A')
     - substs p2 = [[('A', False),('B', False)],[('A', False),('B', True)], ...]
   - Definação a função isTaut :: Prop -> Bool que decide se uma proposição é uma tautologia
     - isTaut p2 = True

3. **Método de Newton.** O método de Newton para computar a raiz quadrada de um número de ponto-flutuante (não-negativo) n pode ser expresso assim

   - comece com uma aproximação inicial do resultado;
   - dada a aproximação atual, a próxima aproximação é definida pela função next a = (a + n/a) / 2;
   - repita do segundo passo até que as duas aproximações mais recentes estejam dentro de uma distância desejada uma da outra, quando o valor mais recente é retornado como resultado.

   Defina a função sqroot :: Double -> Double que implementa este procedimento. Dica: primeiro produza uma lista de aproximações usando a função iterate da biblioteca. Para simplificar, tome 1.0 como a aproximação inicial e 0.00001 como valor para distância.

4. **Estruturas de dados.** Implemente o tipo abstrato Set utilizando árvores de busca.