

## 1. Tipos iniciais e definição da árvore (Word', Line, Doc e Tree)

Esses tipos são aliases para String, criados apenas para melhorar a clareza do código. Eles indicam explicitamente o papel de cada elemento: 'Doc' representa o conteúdo completo do arquivo, 'Line' representa uma única linha do texto e 'Word' representa uma palavra. A estrutura Tree é uma árvore binária de busca usada para organizar as palavras do texto em ordem alfabética. Cada nó contém: a palavra, uma lista com os números das linhas onde ela aparece, a subárvore esquerda (palavras menores) e a subárvore direita (palavras maiores). Leaf representa uma árvore vazia.

## 2. Função numLines :: [Line] -> [(Int, Line)]

A função numLines recebe uma lista de linhas e devolve uma lista de pares, onde cada linha é associada ao seu número (começando em 1). Ela usa a função zip para unir a lista infinita [1..] com a lista de linhas. É importante porque o índice final precisa saber onde cada palavra apareceu.

## 3. Função allNumWords :: [(Int, Line)] -> [(Int, Word')]

Esta função recebe pares (número\_da\_linha, conteúdo\_da\_linha). Para cada linha, ela usa a função words, que automaticamente separa a string em palavras considerando espaços. Em seguida, filtra as palavras com menos de 4 caracteres, removendo palavras curtas como 'artigos e preposições'. Para cada palavra válida, gera um par (número\_da\_linha, palavra). Este é o material bruto que será inserido na árvore.

## 4. Função insOrd :: Ord a => a -> [a] -> [a]

Essa função insere um elemento em uma lista ordenada, mantendo a ordenação e evitando duplicatas. Ela é usada especificamente para inserir números de linha associados a uma palavra dentro do nó da árvore. Assim, mesmo que uma palavra apareça várias vezes na mesma linha, o número não é duplicado, e a lista continua ordenada.

## 5. Função ins :: Word' -> Int -> Tree -> Tree

A função ins é responsável por inserir uma palavra e o número da linha correspondente na árvore de busca. Se a árvore estiver vazia (Leaf), ela cria um novo nó com a palavra e uma lista contendo apenas o número da linha. Se a árvore não estiver vazia, compara a palavra com o conteúdo do nó: – Se for menor, insere na subárvore esquerda. – Se for maior, insere na subárvore direita. – Se for igual, apenas atualiza a lista de números de linha usando insOrd. Essa função garante que a árvore

permaneça ordenada alfabeticamente e evita duplicações de palavras.

## 6. Função mIndexTree :: [(Int, Word')] -> Tree

Esta função monta a árvore completa. Ela percorre a lista de pares (linha, palavra) e, para cada um, chama a função ins para inserir na árvore. É uma construção recursiva: primeiro constrói a árvore do restante da lista, depois insere o elemento atual. Ao final, toda a árvore estará montada.

## 7. Função makeIndexTree :: Doc -> Tree

Essa função representa o pipeline completo de processamento. Ela recebe o documento inteiro em forma de String e aplica as etapas: 1. lines doc — divide o documento em linhas. 2. numLines — numera cada linha. 3. allNumWords — extrai as palavras e as associa às suas linhas. 4. mIndexTree — insere todas as palavras na árvore binária. O resultado final é a árvore totalmente construída com todas as palavras relevantes ordenadas alfabeticamente e associadas às linhas onde aparecem.

## 8. Função printTree :: Tree -> IO ()

A função printTree imprime a árvore em ordem alfabética. Ela faz um percurso in-order: — primeiro imprime a subárvore esquerda (palavras menores), — depois a palavra atual com suas linhas, — por fim a subárvore direita (palavras maiores). Isso garante que a saída seja apresentada em ordem lexicográfica.

## 9. Função main :: IO ()

A função main é o ponto de entrada do programa. Ela: 1. Pede ao usuário o nome do arquivo. 2. Lê o conteúdo do arquivo usando readFile. 3. Constrói a árvore de índice com makeIndexTree. 4. Imprime o índice final usando printTree. Assim, todo o processo de leitura, processamento e exibição é coordenado na função principal.