Relatório: Análise de Frequência de Palavras em Haskell

Higor Gabriel Lino Silva

21 de julho de 2025

1 Introdução

Este trabalho implementa um programa em Haskell para análise de frequência de palavras em textos, seguindo os princípios da programação funcional. O sistema processa arquivos de texto, identifica parágrafos, filtra stopwords, normaliza palavras e gera relatórios de frequência ordenados. O objetivo é desenvolver habilidades em programação funcional, com foco em manipulação de listas, strings e arquivos.

2 Decisões de Implementação

2.1 Normalização de Texto

- Conversão para minúsculas com `toLower`
- Remoção manual de acentos via mapeamento de caracteres (função `removerAcentuacao`)

2.2 Processamento de Parágrafos

- Divisão do texto em linhas com `linhas`
- Agrupamento de linhas não vazias em parágrafos com `agruparLinhasEmParagrafos`

2.3 Eficiência

- Uso de `Data.Map` para contagem eficiente de palavras (`frequenciaMap`)
- Ordenação por frequência com quicksort personalizado (`ordenarPorFrequencia`)

2.4 Stopwords

- Lista embutida no código com palavras funcionais do português (artigos, preposições, etc.)

3 Bibliotecas e Estruturas de Dados

3.1 Bibliotecas

- `Data.Char`: Manipulação de caracteres (minúsculas, verificação de tipo)
- `Data.Map`: Estrutura de dados para contagem eficiente de frequências
- `Data.List`: Funções utilitárias para listas (`nub`, `sortBy`)
- `System.Environment`: Leitura de argumentos da linha de comando (`getArgs`)
- `System.IO`: Operações de I/O

3.2 Estruturas de Dados

- **Listas**: Armazenamento de palavras, linhas e parágrafos
- **Mapas**: Dicionários para contagem de frequência de palavras

4 Fluxo do Programa

O fluxo principal do programa pode ser visualizado como:

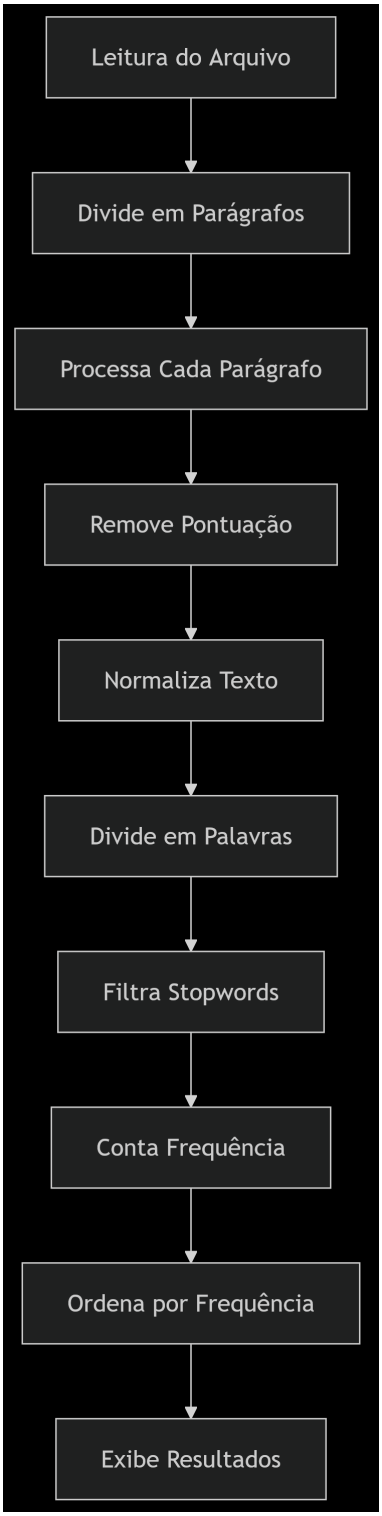

Figura 1: Fluxograma do programa

O processo segue estas etapas:

1. Leitura do arquivo de entrada

2. Divisão do conteúdo em parágrafos
3. Para cada parágrafo:
   (a) Remoção de caracteres não alfabéticos
   (b) Normalização (minúsculas e remoção de acentos)
   (c) Divisão em palavras
   (d) Filtragem de stopwords
   (e) Contagem de frequência com `Data.Map`
   (f) Ordenação por frequência decrescente
4. Exibição dos resultados

5 Funções Principais

- `main`: Ponto de entrada, gerencia o menu interativo
- `processarParagrafo`: Orquestra o processamento de um parágrafo
- `agruparLinhasEmParagrafos`: Agrupa linhas em parágrafos
- `frequenciaMap`: Conta frequências usando `Data.Map`
- `ordenarPorFrequencia`: Ordena palavras por frequência (decrescente)
- `normaliza`: Converte para minúsculas e remove acentos
- `removeNonAlpha`: Remove caracteres não alfabéticos

6 Dificuldades e Soluções

6.1 Normalização de Caracteres

- **Problema**: Tratamento de acentos e conversão de maiúsculas
- **Solução**: Implementação manual de mapeamento de caracteres acentuados para suas versões sem acento

6.2 Eficiência na Contagem

- **Problema**: Versão inicial com listas (`frequenciaPalavras`) tinha complexidade $O(n^2)$
- **Solução**: Migração para `Data.Map` (complexidade $O(n \log n)$)

6.3 Divisão em Parágrafos

- **Problema**: Identificar blocos de texto separados por linhas vazias
- **Solução**: Função recursiva que agrupa linhas não vazias consecutivas

7 Testes e Validação

Foi criado um texto de teste com 3 parágrafos (`textoTeste`) para verificar:

- Correta divisão em parágrafos
- Remoção de stopwords
- Contagem e ordenação de frequências

Exemplo de saída para o texto de exemplo fornecido no enunciado:

```
Parágrafo 1:
("acao",3)
("essencial",1)
("reacao",1)
("interacao",1)

Parágrafo 2:
("outra",3)
("linha",1)
("acao",1)
("reacao",1)
```

8 Instruções de Uso

8.1 Compilação

```
ghc -o analisador Analisador.hs
```

8.2 Execução

- Via argumento de linha de comando:

  ```
  ./analisador arquivo.txt
  ```

- Menu interativo:

```
./analisador
Escolha uma opção:
1 - Demonstração com texto teste
2 - Processar arquivo (digite o nome)
3 - Ler entrada padrão
4 - Usar argumentos da linha de comando
```

9 Conclusão e Aprendizados

Este trabalho permitiu o desenvolvimento de habilidades em programação funcional com Haskell, incluindo:

- Manipulação de strings e listas
- Processamento de arquivos
- Uso de estruturas de dados eficientes (`Data.Map`)
- Organização de código em funções puras e separação de I/O

As principais dificuldades foram o tratamento de caracteres especiais e a otimização do desempenho, superadas com a aplicação de conceitos funcionais e estruturas adequadas. O programa atende aos requisitos e demonstra a elegância e expressividade da programação funcional.