Relatório Técnico Final – Trabalho Prático 1 (TP1)
DCC207 – Algoritmos 2

Isabela Ramos dos Santos (Matrícula: 2023034757)
Layane Garcia (Matrícula: 2023034765)

Outubro de 2025

Resumo

Este relatório detalha a implementação de um protótipo de máquina de busca híbrida para o corpus BBC News, conforme os requisitos do Trabalho Prático 1. O projeto demonstrou a aplicação de estruturas de dados avançadas (Trie Compacta), algoritmos de parsing (Shunting-yard) e ranqueamento estatístico (Z-score), resultando em um sistema eficiente e com alta qualidade de código e interface. O foco da implementação foi a justificativa das escolhas de engenharia para otimizar a recuperação e a persistência de dados.

1 Introdução

Este projeto consistiu na implementação de um protótipo de máquina de busca, aplicando conceitos de algoritmos vistos em aula para manipulação de sequências, além de extras como indexação e recuperação de informação. A solução final é um modelo híbrido que une a performance da busca booleana com um ranqueamento estatístico preciso.

O projeto atendeu aos seguintes passos do trabalho prático:

1. Implementação de uma estrutura de árvore de prefixo para a construção do índice (Trie Compacta).
2. Implementação de módulos para indexação de documentos, garantindo a persistência do índice.
3. Implementação de um módulo de recuperação que suporta consultas booleanas (AND, OR e parênteses).
4. Implementação de ranqueamento de resultados usando o método Z-score.
5. Construção de uma interface web (Flask) funcional com qualidade visual.

2 Decisões Estratégicas de Back-end

O desempenho da máquina de busca é diretamente determinado pelas escolhas de engenharia subjacentes. As decisões abaixo visaram a máxima eficiência e modularidade do sistema.

2.1 Escolha da Estrutura de Dados: Trie Compacta

Passo Resolvido: Implementação da estrutura de árvore de prefixo.

A escolha da **Trie Compacta** (ou Radix Tree) em `compact_trie.py` foi crucial para a eficiência de memória:

- **Compressão de Caminho:** Ao eliminar nós intermediários desnecessários, a estrutura reduz significativamente a pegada de memória do vocabulário, tornando viável a manutenção do índice completo em Memória Principal (RAM), crucial para o requisito de tempo real.
- **Indexação Integrada:** Cada nó terminal armazena a *Posting List* (Mapa de Doc ID $\rightarrow$ Frequência X) e os campos estatísticos μ e σ, centralizando toda a informação de recuperação em um único ponto de acesso.

2.2 Otimização de Persistência e Ranqueamento

Passo Resolvido: Persistência do índice e cálculo das estatísticas.

Para garantir que o ranqueamento por Z-score fosse rápido ($O(1)$) após a indexação inicial, a persistência foi otimizada:

- **Integração de Estatísticas:** Os valores de Frequência Média (μ) e Desvio Padrão (σ) para cada termo foram calculados na fase de indexação e armazenados **diretamente no nó terminal da Trie**.
- **Vantagem:** Na fase de consulta, o módulo de Ranqueamento busca o μ e σ junto com a *Posting List* em $O(1)$, eliminando qualquer acesso a tabelas auxiliares ou reprocessamento de dados.

2.3 Implementação Manual da Persistência

Passo Resolvido: Restrição de não usar `pickle` e formatação própria.

O projeto atendeu rigorosamente à restrição de não usar formatos de serialização automática de objetos.

- **Formato Escolhido:** JSON, utilizado como um formato de texto estruturado.
- **Lógica Algorítmica:** A serialização foi implementada **manualmente** em `indexing.py` pelas funções recursivas `trie_to_dict` e `dict_to_trie`. Estas funções são responsáveis por percorrer a estrutura da árvore nó por nó, traduzindo os dados para o formato JSON e, inversamente, **reconstruindo a estrutura Trie** na memória. Esta implementação manual garante a aderência ao requisito do trabalho.

2.4 Instruções de Execução

Para rodar a Máquina de Busca, basta seguir os passos abaixo no ambiente Python 3.13 / Terminal (preferencialmente WSL/Linux):

1. **Download e Descompactação:** Baixe o arquivo ZIP do repositório final do GitHub e descompacte todo o conteúdo em uma pasta local.
2. **Execução:** No terminal, navegue até a pasta raiz do projeto (onde está o arquivo `app.py`) e execute o comando:

   ```
   python3 app.py
   ```

3. **Acesso ao Servidor:** O servidor Flask é iniciado automaticamente. Graças ao bloco de execução robusto implementado, ele buscará a primeira porta livre (`8080`, `8081`, etc.). O terminal exibirá o link (`http://127.0.0.1:[PORTA]/`). Caso o navegador não abra automaticamente, basta clicar neste link para ser redirecionado à interface da Máquina de Busca.

Nota sobre o Corpus: Os documentos do corpus foram renomeados antes da indexação para seguir a convenção `[categoria][id].txt` (ex: `s001.txt` para *sport*, `p001.txt` para *politics*), facilitando a identificação visual da área temática dos resultados.

3 Módulo de Recuperação de Informação (RI)

3.1 Processamento Booleano: Algoritmo Shunting-yard

Passo Resolvido: Suporte a consultas booleanas complexas (AND, OR e parênteses).

Para garantir a avaliação correta de precedência, que exige que AND > OR, e a correta resolução de parênteses:

- **Algoritmo Escolhido:** Utilizamos o algoritmo **Shunting-yard** (em `query.py`) para converter a expressão booleana (Infixa) em Notação Polonesa Reversa (RPN).
- **Vantagem:** O RPN simplifica a execução da consulta, utilizando uma pilha de resultados onde as operações AND (Interseção) e OR (União) são resolvidas sequencialmente, garantindo a correção lógica.

3.2 Ranqueamento Estatístico por Z-score

Passo Resolvido: Ranqueamento dos resultados por relevância.

A relevância é baseada na média dos Z-scores dos termos da consulta. A escolha do Z-score como métrica de ranqueamento foi devido à sua capacidade de medir o quão incomum é a frequência de um termo no documento em relação à média do corpus.

$$Z = \frac{X - \mu}{\sigma} \tag{1}$$

- **Lógica do Ranqueamento:** O Z-score é calculado com base nos dados (μ, σ) obtidos diretamente da Trie, e a relevância final é a média desses scores. Os documentos são ordenados em ordem decrescente, colocando no topo aqueles com maior **concentração temática** (altos Z-scores).
- **Otimização:** O ranqueamento atua de forma **localizada**, processando apenas o subconjunto de Doc IDs retornados pela busca booleana, garantindo performance de ranqueamento próxima a $O(1)$.

4 Interface e Usabilidade (Front-end)

4.1 Qualidade Visual e Design

Passo Resolvido: Implementação da interface web com qualidade visual.

- **Design Escolhido:** A interface foi inicialmente modelada no **Figma** (design "Trie.Go") para garantir a coesão visual, utilizando um esquema de cores moderno (gradiente rosa/roxo) e a tipografia *Abril Fatface* no título principal.
- **Usabilidade Aprimorada:** A lista de 10 resultados por página implementa a navegação tradicional (Anterior/Próxima) e um recurso que permite ao usuário **digitar diretamente o número da página**, otimizando a navegação em grandes resultados.

4.2 Geração de Snippet

O `snippets.py` garante a melhor representação do documento ao usuário. O termo mais relevante (maior Z-score no documento) é identificado e o trecho de texto é extraído com precisão de **80 caracteres antes** e **80 caracteres depois** do termo, com a correta inserção de reticências (...) nas bordas.

5 Conclusão

O projeto cumpriu integralmente todos os requisitos e restrições técnicas definidos para o Trabalho Prático 1. O sistema final é uma máquina de busca híbrida eficiente, cuja principal conquista reside na **coerência e eficiência do fluxo de dados**, desde a persistência otimizada das estatísticas até o ranqueamento imediato na memória. A solução é robusta, modular e demonstra um domínio prático dos algoritmos de manipulação de sequências e recuperação de informação.