Relatório Técnico - Trabalho Prático 1: Máquina de Busca na BBC News com Trie Compacta

Layla Raissa Silva Pereira , Samuel Lucca Viana Miranda

[1] Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG, Brasil
Disciplina: DCC207 - Algoritmos 2 – Professor: Renato Vimieiro

Resumo. *Este relatório descreve a implementação de um protótipo de máquina de busca para o corpus BBC News. O núcleo do sistema é um índice invertido implementado sobre uma Trie Compacta, desenvolvida do zero. O módulo de recuperação processa consultas booleanas complexas, com suporte para operadores AND, OR e parênteses, utilizando o algoritmo Shunting Yard para parsing. Os resultados são ordenados por relevância, calculada pela média dos z-scores dos termos da consulta. Por fim, uma interface web desenvolvida em Flask apresenta os resultados de forma paginada (10 por página), incluindo a geração de snippets com destaque dos termos.*

1. Introdução

Este relatório descreve a implementação de um protótipo de máquina de busca para o corpus de notícias da BBC News, conforme especificado no Trabalho Prático 1 da disciplina DCC207 - Algoritmos 2. O objetivo central do trabalho é explorar os aspectos práticos da manipulação de sequências, especificamente a implementação de árvores de prefixo para a construção de índices invertidos.

O sistema desenvolvido atende a todos os requisitos propostos: implementa uma Trie Compacta (Patricia Trie) do zero, constrói um índice invertido eficiente para 2225 documentos de notícias, processa consultas booleanas complexas (com operadores `AND`, `OR` e parênteses) e ordena os resultados por relevância usando o método *z-score*. Além disso, foi desenvolvida uma interface web interativa utilizando o framework Flask.

1.1. Especificação

O projeto implementa integralmente as seguintes tarefas:

- Implementação de uma Trie Compacta.
- Criação de um índice invertido para o corpus BBC.
- Persistência do índice em formato próprio (JSON).
- Carregamento do índice em memória principal na inicialização.
- Suporte completo a consultas booleanas (`AND`, `OR`, `()`).
- Ordenação de resultados por relevância (média dos z-scores).
- Geração de snippets de 80 caracteres antes/depois do termo, com destaque.
- Interface web em Flask, com paginação de 10 resultados.

2. Arquitetura do Sistema

O sistema foi projetado de forma modular para garantir a separação de responsabilidades, facilitando a manutenção e os testes. A arquitetura é composta por quatro módulos principais em Python, descritos a seguir:

- `compact_trie.py`: Implementa a estrutura de dados base, a Trie Compacta.
- `inverted_index.py`: Utiliza a Trie para construir o índice invertido e gerencia todos os metadados e estatísticas do corpus.
- `query_processor.py`: Contém toda a lógica de recuperação: parsing booleano, avaliação da consulta, cálculo de relevância e geração de snippets.
- `app.py`: A aplicação Flask que serve como "Controller", unindo os módulos e expondo a funcionalidade ao usuário via interface web e API.

3. Estruturas de Dados

3.1. Trie Compacta (Patricia Trie)

3.1.1. Motivação

A Trie tradicional armazena um caractere por nó, o que resulta em longas cadeias e maior uso de memória. A Trie Compacta resolve esse problema armazenando substrings inteiras em cada nó, reduzindo o número total de nós e aumentando a eficiência de cache.

Vantagens:

- Redução no número de nós e no uso de memória;
- Melhor aproveitamento de cache;
- Busca eficiente por prefixos.

3.1.2. Implementação

A implementação consiste em duas classes principais: `TrieNode` e `CompactTrie`.

Listing 1. Definição da classe TrieNode

```
class TrieNode:
    def __init__(self, prefix=""):
        self.prefix = prefix
        self.children = {}
        self.documents = set()
        self.is_end_of_word = False
```

A classe `CompactTrie` implementa métodos de inserção, busca e serialização (`insert`, `search`, `starts_with`, `to_dict`, `from_dict`).

3.1.3. Complexidade

- Inserção: $O(m \times k)$
- Busca: $O(m \times k)$
- Espaço: $O(n \times l)$

3.2. Índice Invertido

O índice invertido foi implementado sobre a Trie Compacta e armazena as frequências dos termos e metadados dos documentos.

Listing 2. Atributos da classe InvertedIndex

```
class InvertedIndex:
    def __init__(self):
        self.trie = CompactTrie()
        self.documents = {}
        self.term_frequencies = {}
        self.doc_lengths = {}
        self.corpus_term_freq = {}
        self.total_docs = 0
```

A persistência foi feita em formato JSON, garantindo portabilidade e facilidade de depuração.

4. Módulo de Indexação

O processo de indexação é iniciado pela função `initialize_index` em `app.py`.

4.1. Tokenização

A etapa inicial e fundamental do módulo de indexação é o processamento e a tokenização do texto. Esta operação, implementada no método `_tokenize` do módulo `inverted_index.py`, é responsável por normalizar o conteúdo bruto dos documentos em uma lista de termos (tokens) prontos para a indexação. O processo consiste em algumas operações sequenciais:

- **Conversão para Minúsculas:** Primeiramente, todo o conteúdo textual do documento é convertido para letras minúsculas (utilizando `text.lower()`). Esta etapa é crucial para garantir a uniformidade dos termos, tratando "Economy", "economy"e "ECONOMY"como um único termo.
- **Limpeza de Pontuação:** Toda a pontuação (como '!', '.', '?', '(' ou '&') é implicitamente descartada, pois não faz parte do padrão alfanumérico definido na expressão regular.
- **Extração Alfanumérica:** Em seguida, aplicamos uma expressão regular (`r'\b[a-z0-9]+\b'`) sobre o texto em minúsculas. Esta expressão é projetada para localizar e extrair todas as sequências contínuas de caracteres que sejam letras (a-z) ou números (0-9) e que formem "palavras"(delimitadas por `\b`, ou "limite de palavra").
- **Preservação de Números:** Termos que contêm dígitos, como "f1", "mp3"ou o ano "2005", são corretamente identificados e preservados como tokens válidos, o que é essencial para a relevância das buscas no corpus da BBC.

4.2. Construção e Persistência

O método `index_documents` itera por todos os arquivos, extrai os termos, e os insere na Trie associados a um ID de documento único (ex: `"business/001.txt"`).

Conforme exigido, o sistema verifica na inicialização se um arquivo de índice (`index.json`) já existe.

- **Se sim:** O método `load_index` é chamado. Ele usa `json.load` e `CompactTrie.from_dict` para reconstruir o índice e todas as estatísticas em memória, resultando em uma inicialização de aproximados 2 segundos.

- **Se não:** O método `create_new_index` é chamado, que executa a indexação completa (aproximadamente 13 segundos) e salva o resultado com `save_index`.

Nossa implementação, através do método `save_index` no módulo `inverted_index.py`, utiliza `json.dump` para serializar a Trie Compacta e todas as estatísticas do corpus. Optamos pelo formato JSON como já dito anteriomente, isso por algumas, sendo as principais:

- **Facilidade de Depuração:** Sendo um formato de texto legível por humanos, o JSON nos permitiu inspecionar manually o índice salvo e validar a estrutura da nossa Trie serializada durante o desenvolvimento.
- **Portabilidade:** O JSON é um formato universal, garantindo que o índice seja facilmente carregado em diferentes sistemas sem problemas de compatibilidade.

5. Módulo de Recuperação de Informação (RI)

Este módulo, é implementado em `query_processor.py`. Vejamos sobre seu funcionamento.

5.1. Processamento de Consultas Booleanas

A especificação exigia suporte a `AND`, `OR` e parênteses `()`. Para implementar a precedência correta de operadores (onde `AND` é avaliado antes de `OR`), utilizamos o clássico algoritmo Shunting Yard.

O fluxo de avaliação, implementado no método `process_query`, é o seguinte:

1. **Tokenização:** A consulta do usuário (ex: `(economy AND growth) OR recession`) é quebrada em uma lista de tokens: `['(', 'economy', 'AND', 'growth', ')', 'OR', 'recession']`.
2. **Conversão para RPN:** O método `_to_rpn` converte a lista infixa para Notação Polonesa Reversa (RPN) usando uma pilha de operadores, respeitando a precedência. O resultado é: `['economy', 'growth', 'AND', 'recession', 'OR']`.
3. **Avaliação RPN:** O método `_evaluate_rpn` avalia a expressão RPN. Ele usa uma pilha de operandos:
    - Termos (ex: "economy") são buscados na Trie (`self.index.search_term`).
    - Operadores ('AND', 'OR') realizam operações de **interseção (&)** e **união (|)** de conjuntos nos dois últimos `set()` de documentos da pilha.

5.2. Cálculo de Relevância (Z-Score)

A lista de resultados retornada pela busca booleana é então ordenada por relevância. A relevância de um documento para uma consulta é calculada como a média dos z-scores dos termos da consulta. O z-score de um termo em um documento mede o quão atípica é a frequência desse termo no documento em relação ao corpus.

Documentos com z-scores maiores são considerados mais relevantes.

O método `_calculate_z_score` implementa a fórmula:

$$z_{d,t} = \frac{f_{d,t} - \mu_t}{\sigma_t}$$

Onde $f_{d,t}$ é a frequência do termo t no documento d, e μ_t e σ_t são a média e o desvio padrão da frequência do termo t em todo o corpus. Esses valores são rapidamente acessados a partir das estruturas de dados do `InvertedIndex` (ver Seção 3.2).

O score final do documento d para a consulta Q é:

$$score_d = \frac{1}{|Q|} \sum_{t \in Q} z_{d,t}$$

Os resultados são então ordenados de forma decrescente por este score.

5.3. Geração de Snippets

O requisito de exibir um trecho do documento foi implementado no método `generate_snippet`.

- O termo da consulta com o maior z-score naquele documento é selecionado como pivô.
- 80 caracteres de contexto antes e 80 caracteres depois do termo são extraídos, como pedido.
- O termo é destacado na interface usando a tag HTML `<mark>`.

6. Interface Web (Flask)

A interface web foi implementada obrigatoriamente em Flask, conforme solicitado. O `app.py` define todas as rotas e interage com os módulos de back-end.

6.1. Rotas

- `/` : Renderiza `index.html`, a página inicial que exibe as estatísticas do índice.
- `/search` : Recebe a consulta `q` e a página `page`. Ele chama `query_processor.process_query`, realiza a paginação (dividindo os resultados em lotes de 10) e renderiza `results.html`.
- `/document/<path:doc_id>` : Exibe o conteúdo completo de um documento.
- `/api/search` e `/api/stats` : Endpoints JSON para busca e estatísticas, respectivamente.

6.2. Arquitetura

A aplicação Flask segue o padrão MVC (Model-View-Controller):

- **Model:** `InvertedIndex` e `CompactTrie`
- **View:** Templates HTML (Jinja2)
- **Controller:** Rotas Flask em `app.py`

6.3. Design

O design prioriza simplicidade e responsividade, com gradiente roxo-azul e destaque em amarelo para termos encontrados.

7. Exemplos de Uso

7.1. Consulta Simples: football

Busca direta do termo `football` retorna 156 documentos relacionados à categoria esportes.

7.2. Consulta Boolean AND

`football AND player` retorna 23 documentos contendo ambos os termos.

7.3. Consulta Complexa

`(economy AND growth) OR recession` retorna 87 documentos após avaliação via RPN.

8. Análise de Performance e Testes

8.1. Tempo de Indexação

- Indexação inicial: ≈ 13 s
- Carregamento subsequente: ≈ 2 s

8.2. Testes e Validação

Testes foram cruciais para validar a implementação. Um arquivo `test_system.py` foi criado para validar a lógica da Trie (divisão de nós) e a avaliação de consultas booleanas, além de validação manual sobre o corpus BBC.

9. Conclusão

O trabalho atingiu todos os objetivos propostos, entregando uma máquina de busca funcional e eficiente, construída inteiramente em Python, com Trie Compacta, índice invertido, cálculo de relevância e interface web em Flask.

O processo permitiu explorar na prática os conceitos de manipulação de sequências, indexação de texto e algoritmos de parsing. As decisões de design, como a manutenção de números na tokenização e o uso de JSON para persistência, mostraram-se acertadas para os requisitos do corpus e da disciplina.

Referências

1. Morrison, D. R. (1968). *PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric*. Journal of the ACM, 15(4), 514–534.
2. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.
3. Flask Documentation (2024). https://flask.palletsprojects.com/
4. BBC News Dataset. http://mlg.ucd.ie/files/datasets/bbc-fulltext.zip