



**UNIVERSIDADE FEDERAL DE MINAS GERAIS  
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**Messias da Silva Sabadini – 2021032188**

**[messiassaba08@ufmg.br](mailto:messiassaba08@ufmg.br)**

**Paulo Vitor da Silva Rodrigues – 2023028200**

**[paulovrodrigues@ufmg.br](mailto:paulovrodrigues@ufmg.br)**

**Trabalho Prático de Algoritmos 2  
Manipulação de Sequências**

**BELO HORIZONTE  
2025**

## 1 - Introdução

O trabalho teve como foco principal a aplicação dos algoritmos estudados em sala de aula em um contexto mais realista, indo além da teoria. O núcleo do projeto consistiu na implementação de uma estrutura de dados de árvore de prefixo (Trie). A partir dela, construímos um índice invertido, uma técnica fundamental em mecanismos de busca. Especificamente, este projeto aplica esses conceitos na implementação de um mecanismo de busca para o conjunto de dados (dataset) da BBC News. O objetivo é fornecer a recuperação eficiente de documentos com base em consultas booleanas, utilizando o índice invertido baseado na Trie. O objetivo central foi solidificar o conteúdo teórico. Acreditamos que, ao implementar a estrutura do zero, obtivemos uma compreensão mais profunda de seu funcionamento, vantagens e complexidades. Além dos tópicos centrais da disciplina, o projeto proporcionou a oportunidade de explorar conceitos complementares, como os princípios de Recuperação de Informação e o desenvolvimento de aplicações web, melhorando a experiência do usuário por meio de uma interface web construída com Flask.

## 2 - Implementação

Nesta seção, descrevemos as decisões técnicas e de arquitetura tomadas durante o desenvolvimento do mecanismo de busca para o corpus BBC News.

### 2.1 - Indexação

Optamos por uma abordagem híbrida, usando duas estruturas de dados complementares para armazenar e gerenciar o índice invertido.

No arquivo `inverted_index.py`, mantemos duas representações do índice em memória:

**Dicionário Principal** (`self.index = {}`): Mapeia cada termo para um dicionário de documentos e suas frequências: `dict[term, dict[doc_id, freq]]`.

**Trie Compacta** (`self.trie = TrieCompact()`): Uma árvore de prefixos compacta (Radix/Patricia), implementada em `trie.py`.

#### 2.1.1 - Por que um Dicionário Padrão?

O dicionário nativo do Python oferece buscas diretas com complexidade  $O(1)$  em média e é trivial de serializar para JSON, um formato legível e depurável. Decidimos usá-lo como fonte principal para persistência (save/load) e para recuperação de postings (`get_postings`).

#### 2.1.2 - Trie Compacta (Radix/Patricia)

Implementamos uma Trie compacta em `trie.py`, onde cada aresta armazena uma substring (não apenas um caractere), reduzindo o número de nós e melhorando a eficiência de memória.

Características:

- Compressão de arestas: Prefixos comuns são compactados em uma única aresta rotulada.
- Split de arestas: Quando inserimos um termo com prefixo parcial, dividimos a aresta existente para acomodar o novo ramo.
- Postings nos nós terminais: Cada nó final armazena `{doc_id: freq}` diretamente, transformando a Trie em um índice invertido funcional.

API pública:

- `insert(term: str, doc_id: str, freq: int = 1)`: Insere ou atualiza a frequência de um termo em um documento.
- `get_postings(term: str) -> dict`: Retorna `{doc_id: freq}` para match exato do termo.

Benefícios:

- Menos nós que uma Trie tradicional.
- Melhor localidade de memória para termos com prefixos longos em comum.
- Estrutura preparada para extensões futuras (busca por prefixo, autocompletar).

### 2.1.3 - Trade-off no load

Durante o load, carregamos o índice principal do JSON (dicionário) e, em seguida, reconstruímos a Trie em memória iterando sobre o índice carregado. Isso torna a inicialização um pouco mais lenta, mas:

- Mantém um arquivo JSON legível para depuração e inspeção manual.
- Garante consistência entre o dicionário e a Trie.
- Evita formatos binários opacos ou uso de pickle (conforme requisito).

Método `load_or_build`:

- Se o arquivo JSON existir e for válido, carrega-o.
- Caso contrário, constrói o índice do zero percorrendo o corpus e salva automaticamente.

### 2.1.4 - Tokenização e Construção do Índice

Ao invés de um simples `split()` por espaços, utilizamos uma expressão regular robusta:

```
python TOKEN_RE = re.compile(r'\w+', re.UNICODE)
```

Vantagens:

- Lida automaticamente com pontuação, espaços múltiplos e caracteres especiais.
- Suporta Unicode (essencial para textos multilíngues e acentuação).
- Normaliza para minúsculas (`[t.lower() for t in ...]`), garantindo case-insensitive search.

Percorrendo o Corpus:

- Usamos `os.walk` para percorrer recursivamente todas as subpastas (economy, sport, tech, etc.).
- Os `doc_ids` são caminhos relativos (ex: sport/001.txt), garantindo unicidade e facilitando organização por categoria.

Fluxo de Construção:

1. Para cada arquivo, lemos o conteúdo e tokenizamos.
2. Calculamos frequências locais de cada termo no documento.
3. Inserimos no `self.index` (dicionário) e no `self.trie` (Trie compacta).
4. Armazenamos o comprimento do documento em `self.doc_lengths`.

Estatísticas para Ranking:

- Após a indexação, chamamos `_compute_term_stats()` que calcula média ( $\mu$ ) e desvio padrão ( $\sigma$ ) da frequência de cada termo no corpus inteiro.
- Essas estatísticas são usadas posteriormente para calcular z-scores.

## 2.2 - Processamento de Consultas Booleanas

Para permitir buscas complexas como (economia AND política) OR esporte, um simples `split()` da consulta não seria suficiente. Implementamos um parser de descida recursiva que constrói uma AST (Árvore de Sintaxe Abstrata).

### 2.2.1 - Tokenização da Consulta

A consulta do usuário é primeiro quebrada em tokens usando regex:

```
python TOKEN_RE = re.compile(r'\'(|\)|AND|OR|[\^\s()]+',  
re.IGNORECASE)
```

### 2.2.2 - AST (Árvore de Sintaxe Abstrata)

Criamos três classes para representar os nós da árvore:

- **Term(term):** Nó folha representando um termo de busca.
- **And(left, right):** Nó binário representando interseção.
- **Or(left, right):** Nó binário representando união.

### 2.2.3 - Parsing com Precedência e Parênteses

A classe `Parser` implementa um parser recursivo descendente com suporte a precedência de operadores:

**Hierarquia de precedência:**

1. `parse_or()`: Menor precedência (topo da árvore).
2. `parse_and()`: Maior precedência que OR.
3. `parse_factor()`: Trata parênteses e termos individuais.

Exemplo:

- Consulta: (casa AND piscina) OR praia
- AST gerada: `Or(And(Term("casa"), Term("piscina")), Term("praia"))`

**Função auxiliar `extract_terms(ast)`:**

- Percorre a AST recursivamente e extrai todos os termos únicos em ordem.
- Útil para calcular a relevância sem duplicatas.

#### 2.2.4 - Vantagens da AST

- Execução precisa: Podemos avaliar a lógica booleana exatamente como especificado.
- Extensível: Fácil adicionar novos operadores (NOT, NEAR, etc.) ou modificar precedência.
- Separação de responsabilidades: Parser não conhece o índice; apenas constrói a estrutura lógica.

### 2.3 - Lógica de Busca

O arquivo `search.py` orquestra a execução da consulta contra o índice invertido.

#### 2.3.1 - Função `perform_search(query, inverted_index)`

Fluxo:

1. **Parseia a consulta:** `ast = parse_query(query)` retorna a AST ou None se vazia.
2. **Executa a AST:** `docs_for_ast(inverted_index, ast)` retorna o conjunto de `doc_ids` candidatos.
3. **Extrai termos:** `extract_terms(ast)` obtém a lista única de termos da consulta.
4. **Calcula scores:** `score_docs(inverted_index, query_terms, candidate_docs)` ranqueia por relevância.
5. **Ordena resultados:** Retorna lista `[(doc_id, score), ...]` em ordem decrescente de score.

#### 2.3.2 - Execução Recursiva da AST (`docs_for_ast`)

Implementada em `ranking.py`, esta função percorre a AST recursivamente:

```
def docs_for_ast(index, ast) -> Set[str]:
    """Executa a expressão booleana (AST) e retorna os doc_ids candidatos."""
    if ast is None:
        return set()
    from ri.query_parser import Term, And, Or
    if isinstance(ast, Term):
        return _evaluate_postings_for_term(index, ast.term)
    if isinstance(ast, And):
        return docs_for_ast(index, ast.left) & docs_for_ast(index, ast.right)
    if isinstance(ast, Or):
        return docs_for_ast(index, ast.left) | docs_for_ast(index, ast.right)
    return set()
```

### Operações de conjunto:

- **AND:** Interseção (`&`) → documentos que contêm **ambos** os termos.
- **OR:** União (`|`) → documentos que contêm **qualquer um** dos termos.

Esta abordagem é eficiente, aproveitando operações nativas de conjuntos do Python.

## 2.4 - Ranking

O ranking é baseado em **z-scores**, uma medida estatística que indica o quão "incomum" é a frequência de um termo em um documento em relação ao corpus inteiro.

### 2.4.1 - Z-Score: Conceito

Fórmula:  $z\text{-score}(\text{termo}, \text{doc}) = (\text{freq}(\text{termo}, \text{doc}) - \mu(\text{termo})) / \sigma(\text{termo})$

Onde:

- $\text{freq}(\text{termo}, \text{doc})$ : Frequência do termo no documento.
- $\mu(\text{termo})$ : Média da frequência do termo em todos os documentos.
- $\sigma(\text{termo})$ : Desvio padrão da frequência do termo no corpus.

Interpretação:

- **Z-score alto:** O termo aparece com frequência muito acima da média no documento → alta relevância.
- **Z-score baixo ou negativo:** O termo aparece com frequência igual ou abaixo da média → menos relevante.

### 2.4.2 - Pré-cálculo de Estatísticas

Durante a indexação, o método `_compute_term_stats()` calcula  $(\mu, \sigma)$  para cada termo em todo o corpus. Essas estatísticas são armazenadas em `self.term_stats[term] = (mean, stddev)`.

### 2.4.3 - Pontuação de Documentos

A função `score_docs(index, query_terms, candidate_docs)` em `ranking.py` calcula o score de cada documento candidato como a **média dos z-scores** de todos os termos da consulta:

```
def score_docs(index, query_terms: List[str], candidate_docs: Set[str]):  
    """Calcula a média dos z-scores por documento."""  
    scores = {}  
    for doc in candidate_docs:  
        zs = [index.zscore_for(term, doc) for term in query_terms]  
        scores[doc] = sum(zs) / len(zs) if zs else 0.0  
    return scores
```

**Exemplo:**

- Consulta: football AND brazil
- Termos: ["football", "brazil"]
- Doc A:  $z(\text{"football"}, A) = 2.5$ ,  $z(\text{"brazil"}, A) = 1.8 \rightarrow \text{Score} = (2.5 + 1.8) / 2 = \mathbf{2.15}$
- Doc B:  $z(\text{"football"}, B) = 1.2$ ,  $z(\text{"brazil"}, B) = 0.5 \rightarrow \text{Score} = (1.2 + 0.5) / 2 = \mathbf{0.85}$

Doc A é ranqueado acima de Doc B.

### 2.4.4 - Vantagens sobre Contagem Simples

Z-scores são mais robustos que contagem bruta porque:

- Normalizam pela variação do termo no corpus.
- Penalizam termos muito comuns (baixo desvio) e valorizam termos discriminativos (alto desvio).
- Fornecem uma medida estatisticamente fundamentada de relevância.

## 2.5 - Script de Construção (build\_index.py)

O script foi simplificado para delegar toda a lógica de indexação para a classe `InvertedIndex`.

**Código principal:**

```
def build_inverted_index(corpus_directory, index_file):  
    """Carrega o índice existente ou constrói e salva em JSON."""  
    inverted_index = InvertedIndex()  
    inverted_index.load_or_build(index_file, corpus_directory)
```

**Fluxo automático:**

1. Verifica se o arquivo de índice JSON já existe.

2. Se existir, carrega-o.
3. Se não existir, constrói do zero percorrendo o corpus e salva automaticamente.

### **Vantagens:**

- Evita duplicação de lógica.
- Garante consistência entre indexação manual e automática.
- Facilita manutenção e testes.

## **2.6 - Interface Web**

A interface web foi implementada em `web/routes.py` usando Flask e template strings inline.

### **2.6.1 - Rotas Principais**

#### **1. Rota / (index):**

- Exibe a página inicial com formulário de busca.
- Inclui dicas de uso (tooltip explicando AND/OR e parênteses).

#### **2. Rota /search (search):**

- Processa a consulta do usuário (parâmetro ``q``).
- Suporta paginação (parâmetro ``page``, 10 resultados por página).
- Parseia a query, executa a busca booleana e ranqueia os resultados.

Para cada resultado:

- Identifica o termo de maior z-score no documento.
- Gera um snippet de 80 caracteres antes e 80 depois do termo.
- Destaca o termo com ``<b>...</b>``.
- Remove a extensão ``.txt`` do título do link.
- Renderiza os resultados com navegação anterior/próxima.

#### **3. Rota /document/<path:doc\_id> (view\_document):**

- Exibe o conteúdo completo de um documento.
- Preserva os parâmetros da busca (``q`` e ``page``) para permitir voltar aos resultados.
- Inclui botão "← Voltar aos resultados" que retorna para a mesma página de busca.

### **2.6.2 - Snippets com Destaque**

Implementado no método `snippet_for(doc_id, term, context=80)` em `inverted_index.py`:



### Lógica:

1. Abre o arquivo do documento e lê o conteúdo completo.
2. Localiza a primeira ocorrência do termo (case-insensitive).
3. Extrai 80 caracteres antes e 80 depois da ocorrência.
4. Envolva o termo com `**...**` para destaque visual.
5. Remove quebras de linha para manter o snippet inline.

### Se o termo não for encontrado:

- Retorna os primeiros 160 caracteres do documento como fallback.

### Na rota de busca:

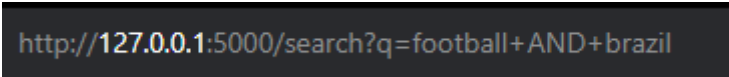
- Escolhemos o termo com **maior z-score** no documento para gerar o snippet, maximizando a relevância visual.

## 2.6.3 - Paginação

### Implementação:

- `per_page = 10`: Limite de 10 resultados por página (conforme requisito).
- Cálculo de `start` e `end` para fatiar a lista de resultados: `page_docs = sorted_docs[start:end]`.
- Links de navegação "anterior" e "próxima" gerados dinamicamente quando aplicável.

### URL de exemplo:



```
http://127.0.0.1:5000/search?q=football+AND+brazil
```

## 2.6.4 - Design Visual

### Estilo CSS inline:

- Gradiente no título "Buscador" para impacto visual.
- Campo de busca arredondado com sombra ao hover.
- Cards de resultados com hover effect (elevação e sombra).
- Botão de ajuda (?) com tooltip explicativo.
- Página de documento com layout limpo e botão de voltar estilizado.

### Usabilidade:

- Interface simples e intuitiva.
- Feedback visual claro (hover, transições suaves).
- Navegação fluida entre busca e visualização de documento.

## 2.7 - Testes

Testes unitários foram implementados no diretório `src/tests` para validar componentes críticos do sistema.

### 2.7.1 - test\_trie.py

Testa a Trie compacta:

- Inserção e recuperação de postings.
- Comportamento com termos que compartilham prefixos (ex: `compress`, `company`, `comparison`).
- Split correto de arestas.
- Match exato de termos (busca por `comp` não retorna resultados, apenas termos completos).
- Tratamento de strings vazias.

### 2.7.2 - test\_inverted\_index.py

Testa o índice invertido:

- Criação de um corpus temporário com arquivos de teste.
- Construção do índice via `build_from_corpus`.
- Verificação de postings e frequências corretas.
- Cálculo de `doc_lengths` (comprimentos de documentos).
- Função `zscore_for` retornando valores válidos.

### 2.7.3 - test\_query\_parser.py

Testa o parser de consultas:

- - Construção correta da AST para consultas simples (`Term`).
- - Parsing de expressões com AND e OR.
- - Precedência de operadores respeitada.
- - Suporte a parênteses para alterar precedência.
- - Função `extract_terms` retornando lista única de termos.
- - Consultas vazias retornando `None`.

### 2.7.4 - Validação Contínua

Durante o desenvolvimento:

- - Executamos os testes após cada mudança significativa.
- - Validamos sintaxe via linters (Pylance).
- - Realizamos testes manuais na interface web para garantir usabilidade.

## 2.8 - Decisões de Arquitetura e Trade-offs

## 2.8.1 - Separação de Responsabilidades

### Módulos bem definidos:

- indexer/: Trie, índice invertido, persistência.
- ri/: Parser, ranking, busca.
- web/: Interface Flask, rotas, templates.
- scripts/: Utilitários CLI para construção do índice.

### Benefícios:

- Facilita manutenção e testes.
- Permite reutilização de componentes.
- Clara separação entre lógica de negócio e apresentação.

## 2.8.2 - JSON vs Formatos Binários

### Escolha: JSON texto

#### Vantagens:

- Legível e depurável manualmente.
- Interoperável com outras ferramentas.
- Atende ao requisito de não usar pickle.

#### Desvantagens:

- Arquivo maior que formato binário.
- Parsing e serialização um pouco mais lentos.

**Trade-off aceito:** Priorizamos legibilidade e conformidade com requisitos.

## 2.8.3 - Trie Compacta vs Trie Tradicional

Escolha: Trie compacta (Radix/Patricia)

#### Vantagens:

- Menos nós → menor uso de memória.
- Melhor localidade de cache.
- Atende explicitamente ao requisito de "Trie compacta".

#### Complexidade adicional:

- - Lógica de split de arestas é mais complexa que Trie simples.
- - Implementação 100% própria (não usamos bibliotecas externas).

**Trade-off aceito:** Complexidade justificada pelo ganho de eficiência e conformidade com requisitos.

### 3 - Como rodar a aplicação

#### 3.1- Clone o repositório usando o link abaixo:

<https://github.com/Messiassaba08/Trabalho-Pratico-Manipulacao-de-sequencias.git>

#### 3.2 - Instale as dependências necessárias

Python e Flask

#### 3.3 - Execute a aplicação

No CMD ou Terminal Linux, entre dentro da pasta do projeto, exemplo:

```
Microsoft Windows [versão 10.0.22631.5624]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\messi> cd C:\Users\messi\OneDrive\Documentos\TP1\bbc-search-engine

C:\Users\messi\OneDrive\Documentos\TP1\bbc-search-engine>|
```

Em seguida, rode o programa utilizando a instrução **python src/run.py**

Dependendo da máquina, pode demorar alguns segundos para carregar. Ao carregar deve-se ver algo parecido com esse tela:

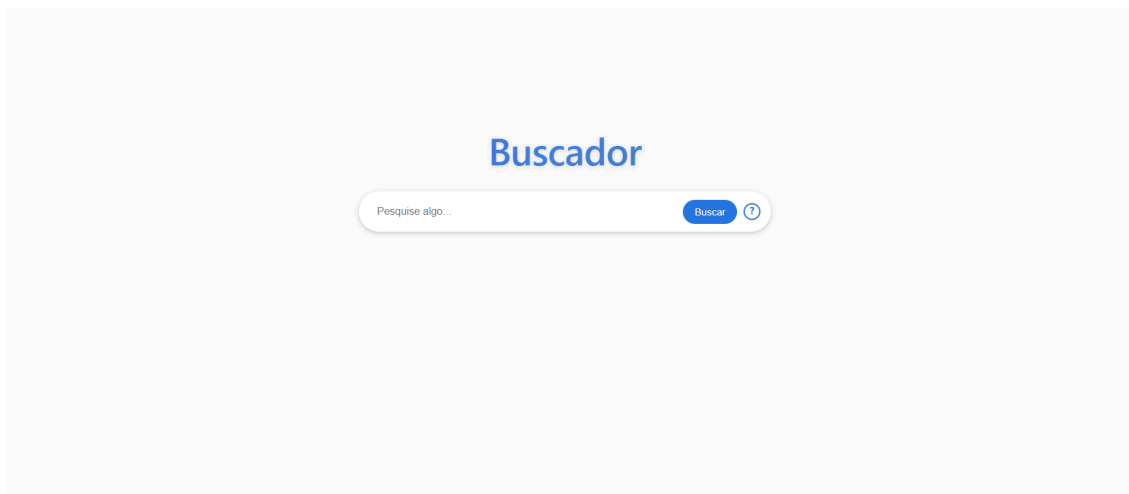
```
C:\Users\messi\OneDrive\Documentos\TP1\bbc-search-engine> python src/run.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 140-551-507
|
```

Acesse a aplicação no seu navegador web em:

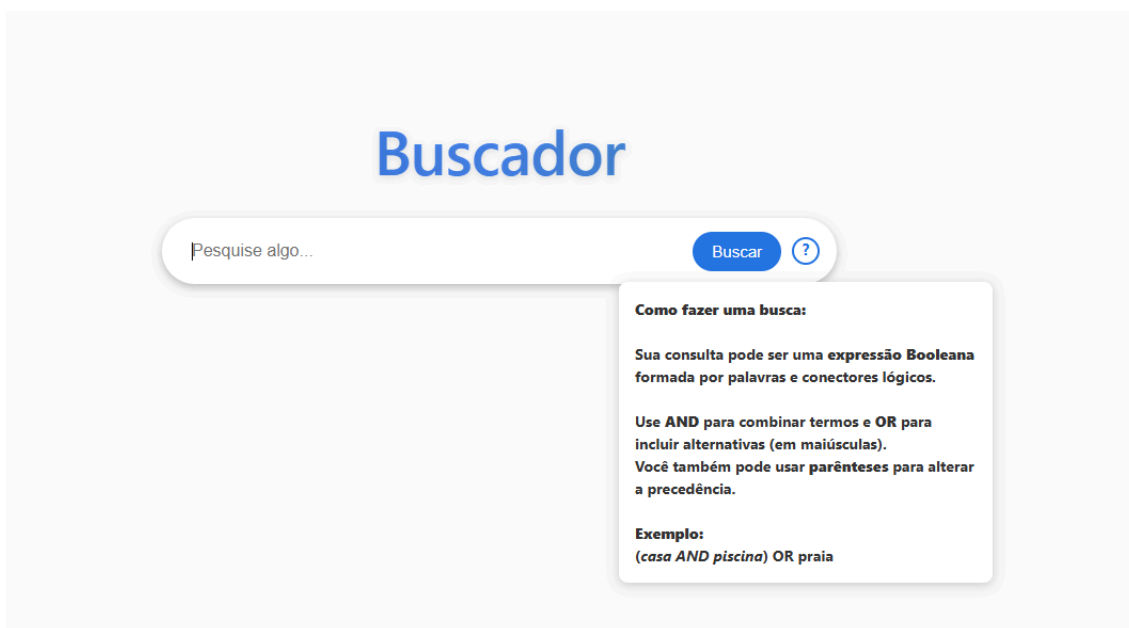
<http://127.0.0.1:5000>

## 4 - Exemplos de uso

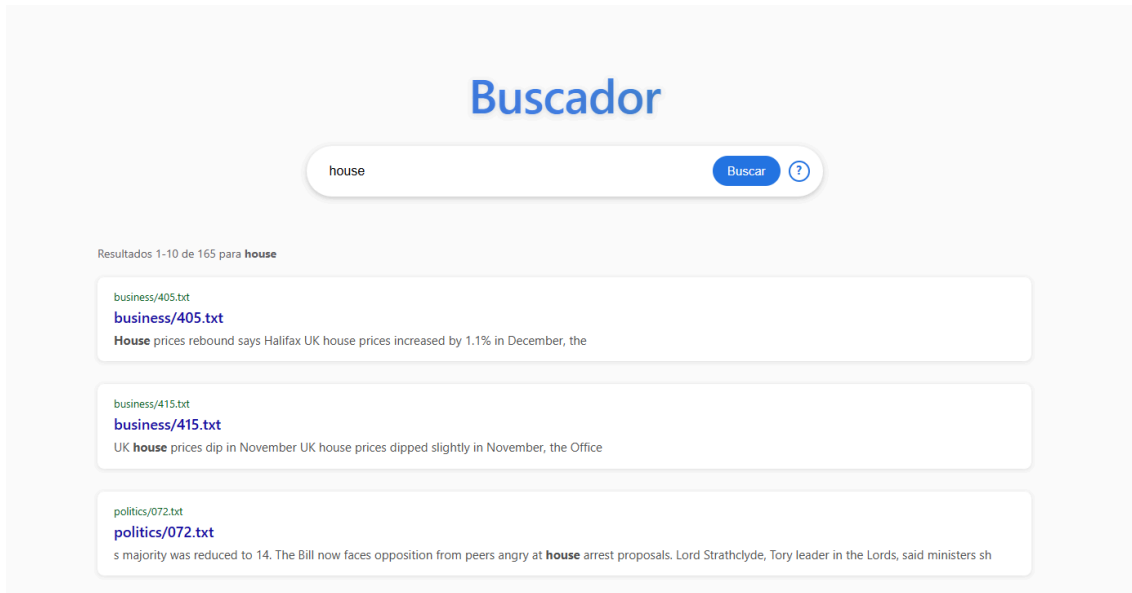
Essa é a tela inicial ao rodar a aplicação e acessar no navegador web



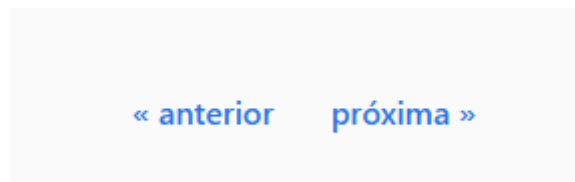
Ao passar o cursor do mouse sobre o sinal de interrogação, há instruções de como utilizar o buscador



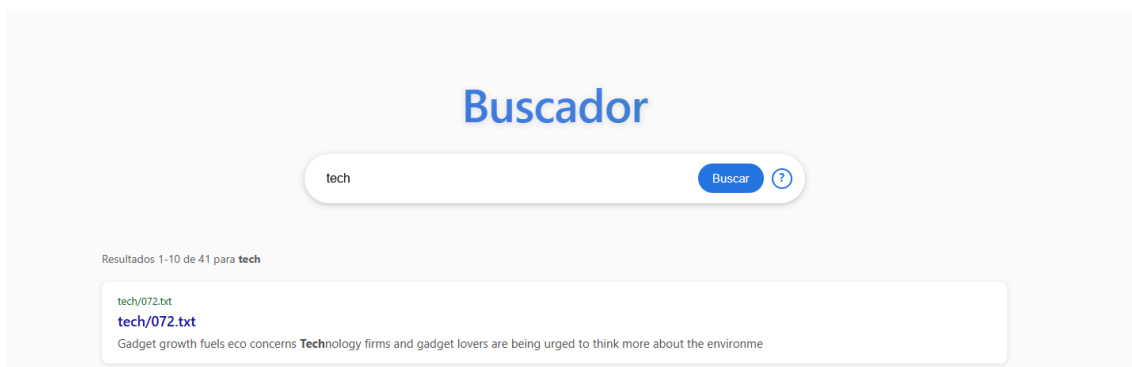
Ao realizar a busca, aparece a quantidade de resultados encontrados e uma breve parte do texto de todas as notícias que contém a(s) palavra(s) pesquisada(s), ordenadas em relação à média do z-score de cada uma.



A cada página, 10 notícias são mostradas. Podemos navegar por esse menu e visualizar todas as notícias por meio das opções de anterior e próxima na parte inferior do menu.



Ao pesquisar alguma string A que é substring de outra string B, a string B aparecerá no resultado da pesquisa. Exemplo: tech e technology



Ao clicar somos redirecionados para as notícias com a opção de voltar para o menu novamente

[← Voltar aos resultados](#)

tech/243.txt

PC photo printers challenge pros

Home printed pictures can be cheaper and higher quality than those from High Street developers, tests shows.

A survey carried out by PC Pro magazine looked at which of 100 home photo printers offered a better deal than handing your snaps to a photo lab. The tests found that images from top PC printers kept their colour longer than professionally produced photographs. But using the wrong printer cartridge could mean snaps fade in months, warned the magazine.

The group test of 100 home photo printers for PCs discovered how much it costs to create images using the devices compared to online developers as well as High Street names such as Jessops, Boots and Snappy Snaps. The comprehensive test also revealed how quickly different printers produced images, the quality of the finished image and how resistant finished pictures were to smudging or water damage. It found that although some ink for printers now costs more than £2 per millilitre it can still be cheaper to produce prints from photographs at home than it is to send them off to a High Street store. "If you really like your photos, then it's definitely worth printing at home instead of going to the high street, but only if you choose the right printer," said Nick Ross, top tester at PC Pro.

Mr Ross said that a new generation of printers produced images with brighter colours and that were less likely to fade than many High Street developers or even some professional wedding photographers. Some High Street photo shops can be the cheapest when it came to developing prints that were 6x4in, said Mr Ross, but the test revealed that images 7x5in and 8x10in in size were cheaper to produce at home. According to PC Pro, producing a print 8x10in on an Epson R800 printer using top quality paper costs £1.87. At Jessops the same image would cost £2.50 and at Snappy Snaps £9.99. A 10x7in snap at Boots would cost £4.99. "Considering how inconvenient it can be to go to the High Street and how silver-halide prints can fade in the sun, we're adamant that it's now better, cheaper and more convenient to print at home," he said. Ann Simpson, marketing manager at Snappy Snaps believes the convenience of high street printing will continue to attract customers. "Some people will want to do their own thing on their computer but the feedback to us is that customers often have to print two or three pictures at home in order to get a good one," she said. "Many people are not skilled at getting the colour, contrast and cropping right and they don't want the hassle," she added. The magazine test found that which ink consumers use determined how long their prints lasted before they started fading. It recommended avoiding so-called third-party inks not produced by printer makers because they tended to produce prints that fade the quickest.

## **5 - Conclusão**

A implementação deste mecanismo de busca foi um exercício prático fundamental, demonstrando como estruturas de dados e algoritmos complexos são aplicados para construir uma aplicação web funcional. O Python provou ser uma ferramenta completa, capaz de lidar desde a manipulação de baixo nível do sistema de arquivos até a lógica de negócios, enquanto o Flask atuou como a ponte para o usuário, escondendo toda a complexidade do índice invertido e do parser booleano por meio de rotas e templates simples. Este projeto destacou a importância da prática. Fomos forçados a tomar decisões reais como a realização de um método de tokenização e implementar a precedência de operadores no parser que transformaram conceitos teóricos, como a Trie e o índice invertido, em aplicações reais, fixando o conhecimento por meio da realização do mesmo.



## Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (s.d.). *Algoritmos: teoria e prática*. Editora Campus.

Greene, D., & Cunningham, P. (2006). *BBC News Dataset*. [Conjunto de dados].

Kleinberg, J., & Tardos, E. (2006). *Algorithm Design*. Pearson Education.

Pallets Projects. (s.d.). *Flask Documentation*. Recuperado de <https://flask.palletsprojects.com/>

Vimieiro, R. (2025). *Slides da disciplina*. [Material de aula].