PUC Minas

Buscador de músicas

Recuperação de Informação na Web.

Alunos:

Henrique Fonseca Araujo

Isabela Stefany Pereira de Faria

João Álvaro Cardoso de Oliveira

João José Cardoso Ribeiro

Julia Vitória da Silva Vieira

Rafael Penido Rocha

Professor:

Pedro Felipe Alves de Oliveira

1. Introdução

O presente relatório descreve o desenvolvimento de um sistema de recuperação de informação capaz de processar, indexar e buscar documentos em formato HTML contendo letras de músicas. Este projeto foi realizado com o objetivo de explorar técnicas de processamento de linguagem natural (PLN), como extração de dados, pré-processamento textual, vetorização TF-IDF e cálculo de similaridade de cosseno para realizar buscas eficientes.

2. Descrição Geral do Sistema

O sistema é dividido em quatro grandes etapas:

- **Coleta de dados na internet:** Varredura e coleta de páginas na internet .
- **Extração e limpeza dos dados**: Leitura de arquivos HTML e extração de informações relevantes como título, nome do artista e letras das músicas.
- **Pré-processamento dos dados**: Tokenização, remoção de “stopwords”, “stemming”, e vetorização dos textos utilizando TF-IDF.
- **Recuperação de informação**: Implementação de uma busca baseada em similaridade de cosseno entre a query do usuário e os documentos indexados.

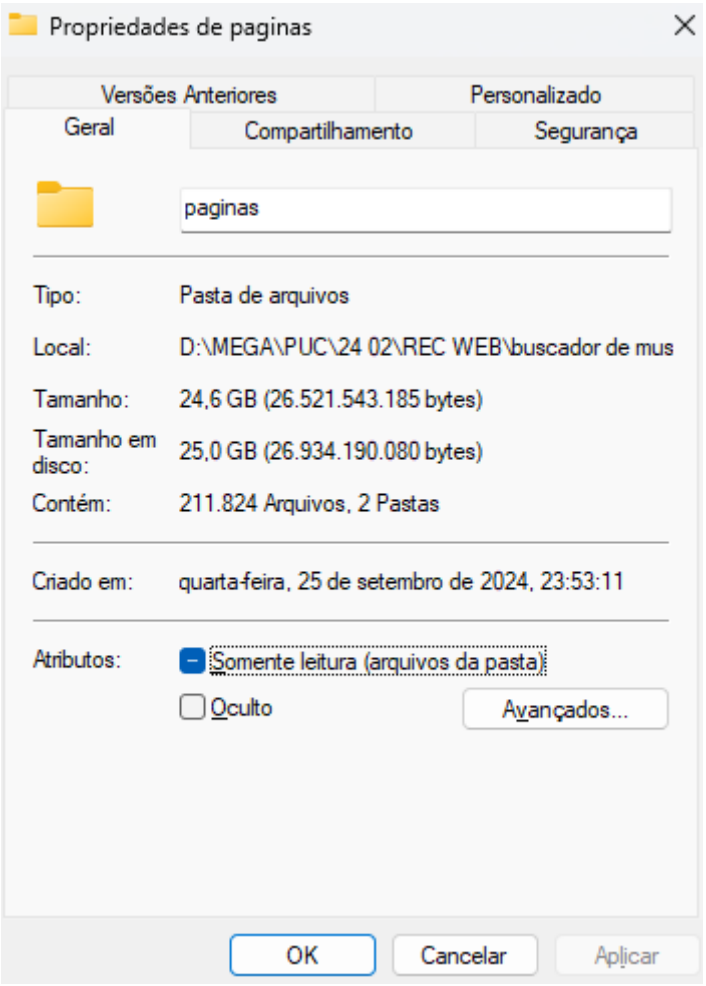
3. Etapas do Desenvolvimento

A coleta de dados para o sistema de recuperação de informações foi realizada através da web scraping de um site que disponibiliza letras de músicas em formato HTML, o site “letras.com”.

3.1. Coleta de dados

Para obter os dados necessários, utilizamos um script em Python, utilizando a biblioteca “BeautifulSoup” que coleta as letras das músicas diretamente do site e consegue fazer alguns tratamentos para através da página inicial, conseguir fazer uma varredura nos filtros de pesquisa. Primeiramente o algoritmo acessa uma lista contendo todos os artistas que iniciam com uma letra do alfabeto, salva a URL de cada artista, e depois acessa todas as músicas desse artista, fazendo o download e armazenando no diretório do sistema.

Foram coletados cerca de 200 mil páginas HTML, resultando em mais de 20gbs de memória interna:



3.2. Extração e Limpeza dos Dados

A função “limpar_dados()” realiza a extração das informações essenciais do arquivo HTML. As principais decisões tomadas foram:

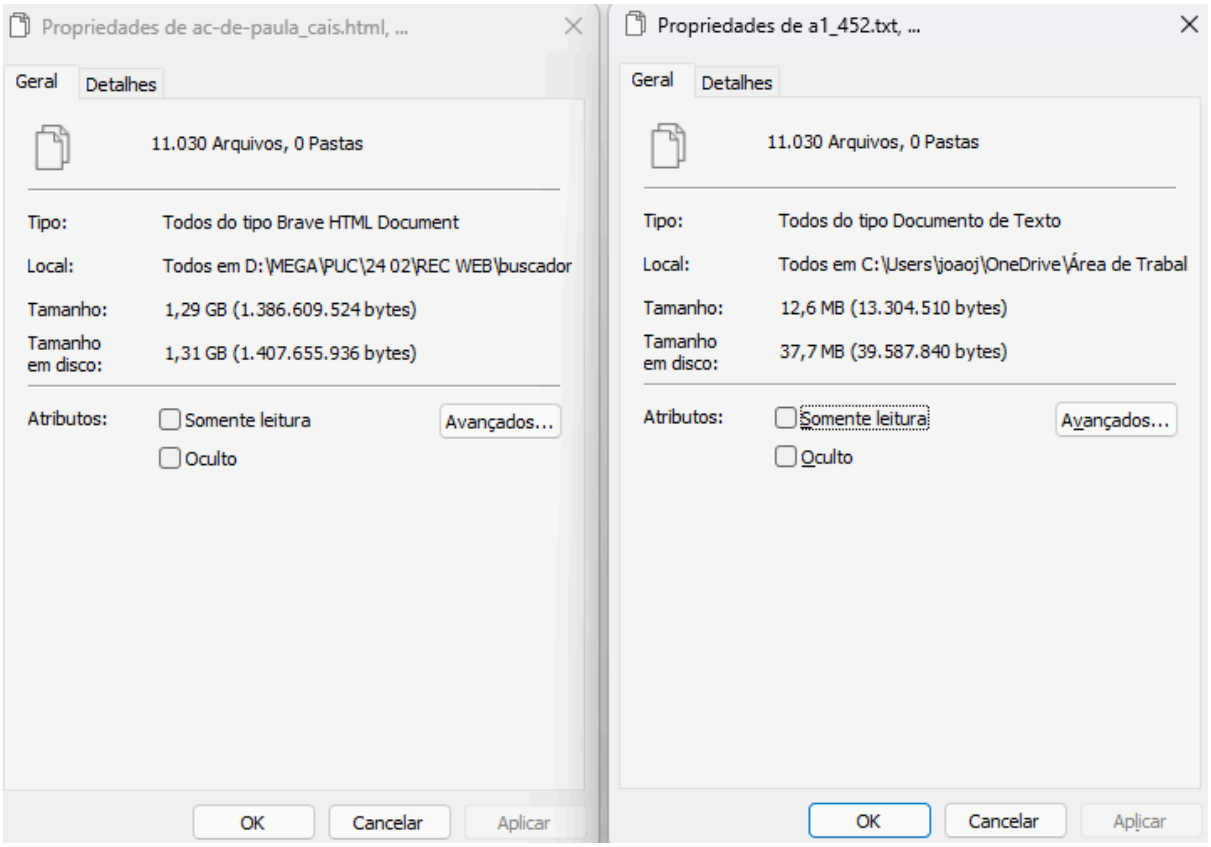
- **Uso do BeautifulSoup**: Para facilitar a manipulação e extração dos elementos HTML.
- **Extração estruturada**:
  - h1 para o título da música.
  - h2 para o nome do artista.
  - div com classe lyric-original para a letra da música, tratando também as quebras de linha (
).

Melhoria implementada: A substituição de tags
 foi ajustada para preservar o espaçamento correto dentro das letras das músicas.

```
def limpar_dados(caminho_arquivo, caminho_destino):
    # ...
```

```
    for br in p.find_all('br'):
        br.replace_with(' ')
    # ...
```

Para fins de testes utilizamos uma amostra de 11.030 páginas, com uma diminuição de aproximadamente 95% no tamanho dos arquivos após a limpeza :



3.3. Pré-Processamento dos Dados

O pré-processamento dos dados é realizado na função “pre_processar()”, que compreende as seguintes etapas:

- **Detecção do idioma**: Utilizando a biblioteca “langdetect”.
- **Tokenização**: Quebra do texto em palavras utilizando o “word_tokenize” da biblioteca NLTK.
- **Remoção de stopwords**: Com base no idioma detectado.
- **Stemming**: Redução das palavras às suas raízes utilizando o SnowballStemmer.

Melhoria implementada: Para garantir maior eficiência, o stemming e a remoção de stopwords foram ajustados para serem aplicados apenas quando o idioma do texto fosse suportado.

```
def pre_processar(texto):
    # ...
    tokens_final = stemming(tokens, idioma)
    # ...
```

3.4. Vetorização e Indexação dos Dados

Após o pré-processamento, os textos são vetorizados utilizando o TF-IDF (TfidfVectorizer). As principais decisões foram:

- **Vocabulário de até 3 milhões de termos** para capturar melhor os diferentes contextos das letras.
- **Uso de n-grams (1 a 3)** para capturar padrões de palavras consecutivas que podem ser relevantes nas buscas.

A função “avaliar_tempo_espaco_indexacao()” mede o tempo e o espaço consumidos durante a indexação dos documentos.

```
vectorizer = TfidfVectorizer(max_features=3000000, ngram_range=(1, 3))
```

3.5. Recuperação de Informação

A busca é realizada com base na similaridade de cosseno entre a “query” (trecho pesquisado pelo usuário) e os documentos indexados, utilizando as funções:

- **buscar_texto**: Retorna o documento mais similar à consulta.
- **buscar_texto_multiple**: Retorna os documentos mais similares, ordenados por relevância.

Melhoria implementada: Um limiar de similaridade foi introduzido para filtrar apenas documentos relevantes, e o sistema agora suporta a exibição dos “*top-n”* documentos mais relevantes.

```
def buscar_texto_multiple(query, tfidf_matrix, vectorizer, limiar, top_n):
    # ...
    resultados = [
            (indice, similaridade) for indice, similaridade in zip(indices_ordenados, similaridades_ordenadas)
        if similaridade >= limiar
    ]
    return resultados[:top_n]
```

3.6. Estrutura do Sistema

- **Diretório de origem (paginas)**: Contém os arquivos HTML originais.
- **Diretório de destino (paginas_processadas_n/)**: Contém os arquivos processados.

- **Diretório de transformação (paginas_processadas_n/A_transformado)**: Contém os textos pré-processados e vetorizados.

4. Desempenho do Sistema

4.1. Tempo de Indexação

O tempo total para indexar um diretório de amostra com aproximadamente 11.000 arquivos foi de aproximadamente **66.84 segundos**, com um consumo de memória de **453;26 MB**.

```
Tempo total de indexação: 66.84 segundos
Memória utilizada: 453.16 MB
Índice TF-IDF e Vetorizador salvos em indice_tfidf.pkl e vectorizer.pkl.
Número de termos no vocabulário: 1958495
Vocabulário salvo em vocabulario.txt
```

4.2. Precisão da Busca

As buscas apresentaram precisão elevada para consultas com frases completas, mas resultados menos relevantes para palavras isoladas. Melhorias potenciais incluem:

- **Ajuste fino dos parâmetros TF-IDF**.
- **Incorporação de modelos de linguagem neural** para melhor compreensão semântica.

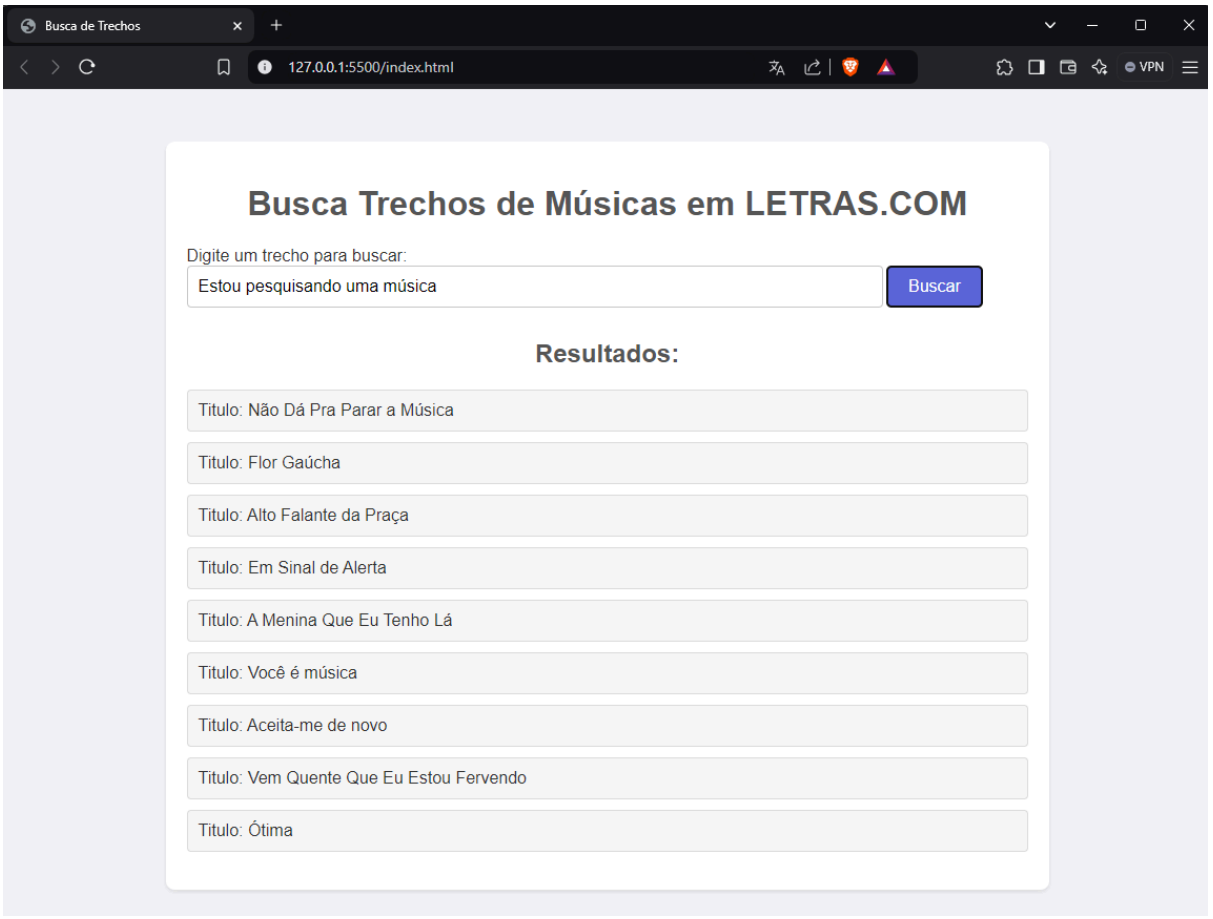
5. Potenciais Melhorias

Após a entrega do relatório parcial, as seguintes melhorias foram identificadas:

- **Expansão do suporte a idiomas**: Implementar suporte a idiomas adicionais no stemming e remoção de stopwords.
- **Uso de embeddings de palavras**: Substituir TF-IDF por representações de palavras como *Word2Vec* ou *BERT* para melhorar a semântica da busca.
- **Interface de usuário**: Criar uma interface web para facilitar o uso do sistema por usuários finais.
- **Indexação incremental**: Permitir a indexação de novos documentos sem a necessidade de reindexar todo o conjunto.

6. Sistema

Foi utilizado uma página HTML como interface, onde tem um campo para digitar o trecho a ser pesquisado, um botão para iniciar busca e uma área para mostrar os resultados:



7. Conclusão

Este sistema demonstrou a eficácia da combinação de técnicas clássicas de PLN, como TF-IDF e similaridade de cosseno, para recuperar informações relevantes em um conjunto de documentos. As melhorias propostas visam aumentar ainda mais a precisão e a usabilidade do sistema.