

UNIVERSIDADE FEDERAL DE MINAS GERAIS  
INSTITUTO DE CIÊNCIAS EXATAS  
CURSO DE SISTEMAS DE INFORMAÇÃO

BRUNA CAROLINE CARDOSO DA SILVA  
GABRIELE PINHEIRO SÁ

**TRABALHO PRÁTICO**

Programação e desenvolvimento de software II

BELO HORIZONTE  
2023

## ***Introdução***

Neste trabalho, implementamos um sistema de recuperação de informações que indexa um conjunto de documentos e recupera os documentos relevantes para uma consulta fornecida pelo usuário. O sistema usa um índice invertido para indexar os documentos e um algoritmo de recuperação para encontrar os documentos relevantes.

Durante o desenvolvimento do trabalho, enfrentamos alguns desafios, como garantir que o índice invertido fosse construído corretamente e que o algoritmo de recuperação retornasse apenas os documentos que contêm todas as palavras da consulta. Conseguimos superar esses desafios através de testes cuidadosos e depuração do código.

## ***Implementação***

Para implementar o sistema de recuperação de informações, criamos uma classe chamada SearchEngine. Essa classe possui métodos para construir o índice invertido, normalizar as palavras e recuperar os documentos relevantes para uma consulta.

O método InvertedIndex lê o conteúdo dos arquivos especificados, normaliza as palavras usando o método normalizeWord e atualiza o índice invertido com as informações dos arquivos. O índice invertido é armazenado em um map que mapeia cada palavra para um map de nomes de arquivos e suas respectivas pontuações.

O método retrieveDocuments recebe um vetor de palavras da consulta e usa o índice invertido para encontrar os documentos relevantes. Ele verifica se cada palavra da consulta existe no índice invertido e, em caso afirmativo, itera sobre os documentos associados à palavra. Para cada documento, ele incrementa um contador e adiciona a pontuação do documento à pontuação total. Depois de iterar sobre todas as palavras da consulta, ele filtra os documentos que foram encontrados em todas as listas de documentos e os classifica de acordo com a pontuação total e o nome do arquivo.

A seguir, está a explicação em pseudocódigo para cada função do programa de máquina de busca:

1. `string SearchEngine::normalizeWord(string word):`

- Recebe uma palavra como entrada
- Inicializa uma string vazia chamada `normalizeWord`
- Para cada caractere da palavra de entrada:
  - Se o caractere for uma letra, adiciona sua versão em minúsculas à `normalizeWord`
- Retorna `normalizeWord`, que é a palavra já normalizada

2. `void SearchEngine::InvertedIndex(vector<string>& files, string folderPath, set<string>& wordbook):`

- Recebe uma lista de nomes de arquivos, o caminho da pasta e um conjunto chamado `wordbook`
- Para cada nome de arquivo na lista de arquivos:
  - Constrói o caminho completo do arquivo usando a pasta e o nome do arquivo
  - Abre o arquivo para leitura
  - Inicializa a variável `word` para armazenar cada palavra lida do arquivo
  - Enquanto houver palavras no arquivo:
    - Normaliza a palavra lida chamando a função `normalizeWord`
    - Insere a palavra normalizada no `wordbook`
    - Incrementa o contador da palavra no índice invertido `invertedIndex`
- Ao final, o índice invertido terá mapeamentos de palavras para os arquivos em que ocorrem e suas frequências

3. `vector<string> SearchEngine::retrieveDocuments(const vector<string>& queryWords):`

- Recebe uma lista de palavras de consulta chamada `queryWords`
- Inicializa dois maps: `documentCounts`, para contar a frequência de documentos relevantes e `documentScores`, para somar as pontuações dos documentos
- Para cada palavra na lista de consulta:
  - Verifica se a palavra está presente no índice invertido

- Se estiver presente, obtém a lista de documentos em que a palavra ocorre e suas contagens
- Para cada documento na lista de documentos:
  - Armazena o nome do documento em `documentName`
  - Obtém a pontuação do documento a partir do índice invertido
  - Incrementa o contador de documentos `documentCounts` para o documento
  - Adiciona a pontuação ao score do documento em `documentScores`
- Constrói uma lista de pares ordenados `sortedDocuments`, contendo o nome do documento e seu score
- Filtra a lista `sortedDocuments` mantendo apenas os documentos que ocorrem em todas as palavras de consulta
- Classifica a lista `sortedDocuments` com base no score em ordem decrescente e, em caso de empate, pelo nome do documento em ordem alfabética
- Constrói uma lista de documentos relevantes `relevantDocuments`, contendo apenas os nomes dos documentos ordenados
- Retorna a lista `relevantDocuments` como o resultado da pesquisa

## ***Execução***

Para executar o programa, você precisa ter um sistema operacional compatível com o compilador g++ e o comando make. O programa foi testado em sistemas operacionais Linux e Windows, mas deve funcionar em outros sistemas operacionais com suporte ao g++ e ao make.

Para compilar o programa, abra um terminal e navegue até o diretório onde estão os arquivos do programa. Em seguida, execute o comando ``make`` para compilar o programa e criar um arquivo executável chamado "main".

Para executar o programa, execute o comando ``./main`` no terminal. Isso iniciará o programa e exibirá a mensagem "Digite sua consulta:". Digite uma consulta e pressione Enter para ver os documentos relevantes para a consulta.

Lembre-se de que você precisa ter arquivos de texto na pasta "documents" para que o programa funcione corretamente. Além disso, para fazer uma nova consulta, você precisa executar o comando `./main` novamente.

*O projeto está organizado da seguinte forma:*

```
seachengine/
├── Makefile
├── build/
│   └── [arquivos.o]
├── include/
│   └── searchengine.hpp
├── src/
│   ├── entidades/
│   │   └── searchengine.cpp
│   └── main.cpp
└── documents/
    ├── document1.txt
    ├── document2.txt
    └── ...
```

## ***Conclusão***

Ao trabalhar neste projeto, aprendemos sobre a construção e o uso de índices invertidos para recuperar informações. Também ganhamos experiência na implementação de algoritmos de recuperação e no tratamento de casos especiais.

Uma possível melhoria para o nosso trabalho seria implementar um algoritmo mais sofisticado para classificar os documentos relevantes, levando em consideração fatores como a frequência das palavras da consulta nos documentos e a proximidade das palavras nos documentos.

No geral, estamos satisfeitos com o resultado do nosso trabalho e acreditamos que ele atende aos requisitos especificados.