**INTRODUÇÃO**

Nesta atividade, trabalhamos em equipe para desenvolver um projeto utilizando o GitHub como plataforma de colaboração e versionamento. O objetivo era criar um sistema de busca chamado "máquina de busca" usando um índice invertido. Durante o projeto, melhorei minhas habilidades de programação e trabalhei com outros membros do grupo.

Para isso, implementamos classes e métodos para carregar e processar os documentos, criar e manter o índice invertido, e realizar consultas de busca. Tivemos que lidar com desafios, como normalizar termos, remover caracteres especiais e calcular frequências nos documentos.

Além disso, o uso do GitHub facilitou nossa colaboração, permitindo compartilhar e integrar alterações de forma eficiente. Também nos ajudou a revisar o código e acompanhar o progresso do projeto.

**IMPLEMENTAÇÃO**

```
/*
 * Bibliotecas inclusas.
 */
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <unordered_map>
#include <unordered_set>
#include <algorithm>
#include <cassert>
#include <dirent.h>
#include <cctype>
#include <locale>

using namespace std;

/**
 * Classe que representa um documento.
 */
class Document {
public:
```

```
    int id;
    string title;
    string content;

    Document() = default;
    Document(int id, const string& title, const string& content) : id(id), title(title),
content(content) {}
};

/**
 * Classe que implementa um índice invertido para pesquisa de documentos.
 */
class InvertedIndex {
public:
    unordered_map<string, unordered_map<int, int>> index;  // índice invertido
    unordered_map<int, Document> documents;  // documentos carregados
    int next_id;  // próximo ID disponível para um documento

    /**
     * Construtor da classe InvertedIndex.
     * Cria um índice invertido vazio e carrega os documentos de um diretório específico.
     *
     * @param folderPath Caminho do diretório contendo os documentos.
     */
    InvertedIndex(const string& folderPath) : next_id(1) {
        load_documents(folderPath);
    }

    /**
     * Função de normalização de termos.
     * Realiza a normalização de um termo, removendo acentos, caracteres especiais e
convertendo para minúsculas.
     *
```

```
     * @param term Termo a ser normalizado.
     * @return O termo normalizado.
     */
    string normalize(const string& term) const {
        string normalized_term = term;

    // Converter para letras minúsculas e acentos e remover números
    string normalize(const string& term) const {
        string normalized_term = term;

    normalized_term.erase(remove_if(normalized_term.begin(), normalized_term.end(),
                                    [](unsigned char c) { return isdigit(c); }),
                          normalized_term.end());

    normalized_term.erase(remove_if(normalized_term.begin(), normalized_term.end(),
                                    [](unsigned char c) { return !isalnum(c); }),
                          normalized_term.end());

    transform(normalized_term.begin(), normalized_term.end(), normalized_term.begin(),
              [](unsigned char c) { return tolower(c); });

        return normalized_term;
    }

    /**
     * Carrega os documentos de um diretório.
     * Percorre o diretório especificado, lê cada arquivo de texto e adiciona os documentos ao índice invertido.
     *
     * @param folderPath Caminho do diretório contendo os documentos.
     */
    void load_documents(const string& folderPath) {
        vector<string> fileNames = get_file_names(folderPath);
```

```
    for (const auto& fileName : fileNames) {
        string filePath = folderPath + "/" + fileName;
        ifstream file(filePath);
        if (file.is_open()) {
            string title = fileName;
            string content;
            string line;
            while (getline(file, line)) {
                content += line + "\n";
            }
            file.close();
            add_document(Document(next_id, title, content));
            next_id++;
        } else {
            cout << "Failed to open the file: " << fileName << endl;
        }
    }
}

/**
 * Obtém os nomes dos arquivos de um diretório.
 *
 * @param folderPath Caminho do diretório.
 * @return Vetor contendo os nomes dos arquivos encontrados.
 */
vector<string> get_file_names(const string& folderPath) {
    vector<string> fileNames;
    DIR* directory;
    dirent* entry;
    if ((directory = opendir(folderPath.c_str())) != nullptr) {
        while ((entry = readdir(directory)) != nullptr) {
            if (entry->d_type == DT_REG) {  // Arquivo regular
                fileNames.push_back(entry->d_name);
```

```
            }
        }
        closedir(directory);
    } else {
        cout << "Failed to open the directory: " << folderPath << endl;
    }
    return fileNames;
}

/**
 * Adiciona um documento ao índice invertido.
 * O documento é adicionado ao mapeamento de documentos e suas palavras são adicionadas ao índice invertido,
 * juntamente com as frequências de ocorrência.
 *
 * @param doc Documento a ser adicionado.
 */
void add_document(const Document& doc) {
    documents[doc.id] = doc;
    vector<string> terms = tokenize(doc.title + " " + doc.content);
    unordered_set<string> unique_terms;
    for (const auto& term : terms) {
        string normalized_term = normalize(term);
        if (!unique_terms.count(normalized_term)) {
            index[normalized_term][doc.id] = 0;
            unique_terms.insert(normalized_term);
        }
        index[normalized_term][doc.id]++;
    }
}

/**
 * Divide uma string em tokens.
```

```
    * Separa uma string em tokens (palavras) usando espaços como delimitadores.
    *
    * @param str String a ser tokenizada.
    * @return Vetor contendo os tokens resultantes.
    */
   vector<string> tokenize(const string& str) const {
       vector<string> tokens;
       string token;
       istringstream iss(str);
       while (iss >> token) {
           tokens.push_back(token);
       }
       return tokens;
   }


   /**
    * Realiza uma busca no índice invertido com base em uma consulta.
    * Para cada termo da consulta, encontra os documentos relevantes e calcula uma pontuação para cada um.
    * Os resultados são ordenados pela pontuação e retornados.
    *
    * @param query Consulta de busca.
    * @return Vetor contendo os pares (ID do documento, pontuação) dos resultados da busca.
    */
   vector<pair<int, int>> search(const string& query) const {
       vector<string> terms = tokenize(query);
       unordered_map<int, int> document_scores;
       for (const auto& term : terms) {
           string normalized_term = normalize(term);
           if (index.count(normalized_term)) {
               const unordered_map<int, int>& term_postings = index.at(normalized_term);
               for (const auto& posting : term_postings) {
```

```
                document_scores[posting.first] += posting.second;
            }
        }
    }
    vector<pair<int, int>> sorted_results(document_scores.begin(), document_scores.end());
    sort(sorted_results.begin(), sorted_results.end(), [this](const auto& a, const auto& b) {
        if (a.second == b.second) {
            const Document& docA = documents.at(a.first);
            const Document& docB = documents.at(b.first);
            return docA.title < docB.title;
        }
        return a.second > b.second;
    });
    return sorted_results;
}
```

**/**

*** Imprime os títulos dos documentos encontrados na busca.**

*** Se não houver resultados, imprime uma mensagem informando que nenhum documento foi encontrado.**

*****

*** @param results Vetor contendo os pares (ID do documento, pontuação) dos resultados da busca.**

*** @param query Consulta de busca.**

***/**

```
void print_titles(const vector<pair<int, int>>& results, const string& query) const {
    if (results.empty()) {
        cout << "No documents found for the search query '" << query << "'." << endl;
        return;
    }
    cout << "Search Results for query '" << query << "':" << endl;
    for (const auto& result : results) {
        int id = result.first;
```

```
            if (documents.count(id)) {
                const Document& doc = documents.at(id);
                cout << "Title: " << doc.title << " | Frequency: " << result.second << endl;
            }
        }
        cout << endl;
    }
};
/*
 * Main para pesquisas interativas.
 */
int main() {
    string folderPath = "./documents.txt";
    InvertedIndex index(folderPath);
    string query;
    while (true) {
        cout << "Enter a search query (or 'exit' to quit): ";
        getline(cin, query);
        if (query == "exit") {
            break;
        }
        vector<pair<int, int>> result = index.search(query);
        index.print_titles(result, query);
    }
     return 0;
}
```

**EXPLICANDO O CÓDIGO E SUAS CLASSES**

**Document (classe):** Essa classe representa um documento e possui três membros de dados: id (identificador do documento), title (título do documento) e content (conteúdo do documento).

**InvertedIndex (classe):** Essa classe implementa um índice invertido para pesquisa de documentos. Ela possui os seguintes membros de dados:

**Index**: Um unordered_map que mapeia termos normalizados para um segundo unordered_map que mapeia IDs de documentos para a frequência de ocorrência desse termo no documento.

**documents**: Um unordered_map que mapeia IDs de documentos para os próprios documentos.

**next_id**: Um inteiro que representa o próximo ID disponível para atribuir a um documento.

**normalize(const string& term):** Essa função recebe um termo e retorna sua versão normalizada. A normalização envolve a remoção de acentos, caracteres especiais, conversão para letras minúsculas, etc.

**remove_accents(const string& term):** Essa função recebe um termo e remove os acentos e caracteres especiais, mantendo apenas letras e dígitos.

**load_documents(const string& folderPath**): Essa função carrega os documentos de um diretório especificado pelo folderPath. Para cada arquivo no diretório, ele lê o título e o conteúdo do arquivo e adiciona o documento ao índice invertido por meio da função add_document.

**get_file_names(const string& folderPath):** Essa função retorna os nomes dos arquivos presentes em um diretório especificado pelo folderPath.

**add_document(const Document& doc):** Essa função adiciona um documento ao índice invertido. Ela adiciona o documento ao mapeamento documents e atualiza o índice invertido index com as palavras e suas frequências de ocorrência no documento.

**tokenize(const string& str):** Essa função recebe uma string e a divide em tokens (palavras) com base nos espaços como delimitadores. Ela retorna um vetor contendo os tokens resultantes.

**search(const string& query):** Essa função realiza uma busca no índice invertido com base em uma consulta (query). Ela percorre os termos da consulta, encontra os documentos relevantes e calcula uma pontuação para cada um com base nas frequências dos termos nos documentos. Os resultados são retornados em um vetor de pares (ID do documento, pontuação), ordenados pela pontuação.

**print_titles(const vector<pair<int, int>>& results, const string& query):** Essa função imprime os títulos dos documentos encontrados na busca. Se não houver resultados, uma mensagem informando que nenhum documento foi encontrado é exibida.

**main():** A função principal do programa. Ela cria uma instância do InvertedIndex, carrega os documentos do diretório especificado, e entra em um loop onde solicita ao usuário uma consulta de busca. A função realiza a busca no índice invertido e imprime os títulos dos documentos encontrados. O loop continua até que o usuário digite "exit" para sair do programa.

**CONCLUSÃO**

Participar deste projeto no GitHub foi uma experiência positiva de trabalho em equipe e colaboração. Aprendi a importância da organização e comunicação para o sucesso do projeto. Melhorei minhas habilidades de programação, especialmente na implementação de algoritmos e manipulação de dados.

Durante a implementação do sistema de busca, percebi como é crucial usar algoritmos eficientes e estruturas de dados adequadas para obter um desempenho satisfatório. A escolha correta dos algoritmos de normalização de termos, remoção de caracteres especiais e cálculo de frequência de termos foi fundamental para obter resultados precisos nas consultas.

O uso do GitHub como plataforma de versionamento e colaboração facilitou a integração do trabalho de cada membro do grupo. Foi mais fácil revisar o código, compartilhar ideias e resolver conflitos. A ferramenta também permitiu acompanhar o progresso do projeto e manter um histórico completo das alterações realizadas.

Resumindo, essa experiência não apenas resultou no desenvolvimento de um sistema funcional, mas também me proporcionou aprendizado valioso em programação. Através desse projeto, pude aplicar meus conhecimentos, adquirir novas habilidades e trabalhar de forma eficaz em equipe utilizando ferramentas de colaboração como o GitHub.

João Henrique Voss Teixeira - 2022056080