UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
CURSO DE ENGENHARIA DE PRODUÇÃO

Arthur dos Santos Oliveira
Joubert Lino Fernandes dos Reis
Victoria Dias de Almeida

Trabalho Prático: Máquina de Busca

Trabalho Prática da disciplina da matéria de Programação e Desenvolvimento de Software, do professor Thiago Ferreira de Noronha ministrado na Universidade Federal de Minas Gerais (UFMG), campus Pampulha, Belo Horizonte/MG.

BELO HORIZONTE
2023

Arthur dos Santos Oliveira
Joubert Lino Fernandes dos Reis
Victoria Dias de Almeida

Máquina de Busca

BELO HORIZONTE
2023

Introdução

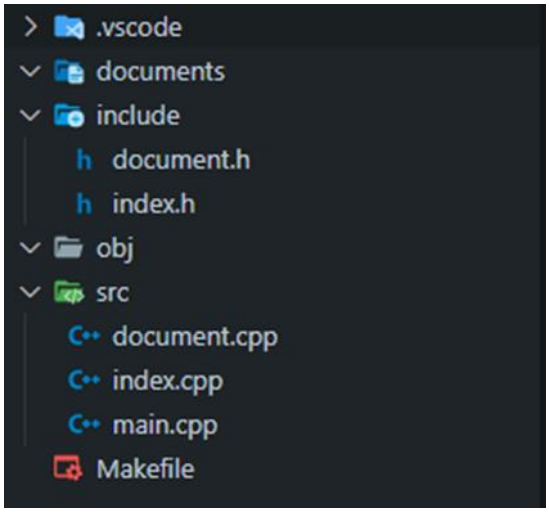
A atividade proposta foi a construção de um software de Máquina de Busca, no qual alguns critérios deveriam ser respeitados. Um dos mecanismos utilizados foi o GitHub, para o versionamento do código, o qual a principio foi um desafio, mas com a pratica se tornou mais tranquilo de utilizar.

Para a construção do software montamos um índice invertido que armazenava informações como o nome do documento, as palavras deles e suas frequência, o que será explicado mais detalhadamente nas próximas sessões.

Implementação

Árvore

O projeto está organizado da seguinte forma:



Onde os códigos fontes ficam dentro da pasta src/, os arquivos que necessitam de ser incluídos na pasta include/, o arquivos .o são gerados dentro da pasta obj/. O arquivo Makefile fica no diretório raiz do projeto, mas adiante será explicado a sua função

Bibliotecas

As bibliotecas utilizadas no código estão listadas abaixo:

- algorithm
- cctype
- cstdlib
- filesystem
- fstream
- iostream

- map
- sstream
- string
- vector

Sendo que as bibliotecas: iostream, string, vector, map fazem parte da biblioteca padrão do C++ que nos permite usar classe que lidam com as entradas e saídas de dados, manipulação de strings, implementação de arrays dinâmicos ou uma tabela hash por exemplo. Já as depois bibliotecas foram utilizadas para lidar com situações mais especificas, que serão listadas abaixo:

- algorithm: define uma coleção de funções que foram definidas para serem usadas em um range de elementos.
- cctype: declara um conjunto de funções para classificar e transformar caracteres individuais.
- cstdlib: esta biblioteca define uma série de funções que podem gerenciar dinamicamente a memória, comunicar com o ambiente, dentre outros.
- filesystem: biblioteca que fornece funcionalidades para lidar com operações relacionadas a diretórios e arquivos, por exemplo. Sendo que ela começou a fazer parte da biblioteca padrão do C++ a partir da versão 17.
- fstream: a biblioteca fornece operação para lidar com entrada e saída de dados de arquivos.
- sstream: biblioteca que fornece classe de fluxo de string.

Códigos

O projeto completo é composto por duas classes:

- Document;
- Index;

Abaixo teremos uma melhor explicação do que cada classe realiza e como estão organizadas.

Document:

Document possui dois arquivos referentes a sua categoria, um .h, onde se realiza a declaração da classe e seus métodos e um struct para o tratamento de erros, e o .cpp, onde está a implementação de cada método presente.

- document.h (include/document.h)

```
#define DOCUMENT_H_

#include <string>
#include <vector>

using namespace std;

struct falhaDocumento {
     string documentoComErro;
};

class Document {
     public:
         Document (string caminho);
         vector<pair<string, vector<string>>> DocumentReader();

     private:
string caminhoDiretorio_;
};
 #endif
```

Em document.h usamos algumas das bibliotecas que foram declaradas anteriormente. O Struct presente no código é utilizado para guardar o nome de um arquivo caso não seja possível acessá-lo.

A class Document por sua vez possui o seu construtor e um método que realiza o acesso ao diretório e a leitura de cada arquivo presente nele.

- document.cpp (src/document.cpp)

Em document.cpp como dito temos a implementação dos métodos que foram declarados em “document.h”, ou seja, a implementação do construtor e do DocumentReader. Neste documento utilizamos uma parte das bibliotecas, como por exemplo “filesystem”:

```
      if (filesystem::is_empty(caminhoDiretorio_)) {
            throw runtime_error("O diretório está vazio.");
       }
       for (const auto& entry :
filesystem::directory_iterator(caminhoDiretorio_)){
           if (entry.is_regular_file()) {
              ifstream file(entry.path());

              if (!file) {
                  throw falhaDocumento{(entry.path().filename().string())};
              }
      // continua …
```

Neste trecho de codigo temos o tratamento de erros, caso o diretório esteja vazio o programa retornará uma mensagem e será encerrado, e caso não seja possível ler um arquivo o seu nome será armazenado no Struct. Nesse trecho também podemos ver o uso da biblioteca “filesystem” que ajuda a percorrer os diretórios.

Index

Assim como document, o Index também possui dois arquivos referentes a eles, o .h que possui declarações do construtor e de métodos, e o .cpp que contém a implementação destes métodos.

- index.h (include/index.h)

```
#ifndef INDEX_H_
#define INDEX_H

#include <string>
#include <vector>
#include <map>

using namespace std;

class Index {
     public:
         Index (vector<pair<string, vector<string>>> documento);

        string normalizarPalavra (const string palavra);

        vector<string> divisaoPalavra (string pesquisa);

        static bool compare (pair<string, int> a, pair<string, int> b);

        vector<pair<string, int>> Pesquisa (string pesquisa);

     private:
        map<string, map<string, int>> dicionario_;
};

#endif
```

- index.cpp (src/index.cpp)

```
#include <algorithm>
#include <cctype>
#include <sstream>

//... Parte do codigo

string Index::normalizarPalavra (string palavra) {
     string palavraNormalizada = palavra;
     transform(
         palavraNormalizada.begin(), palavraNormalizada.end(),
palavraNormalizada.begin(),[](unsigned char c){
             return tolower(c);
     });

     palavraNormalizada.erase(
         remove_if(palavraNormalizada.begin(), palavraNormalizada.end(),
[](unsigned char c) {
             return ispunct(c);
```

```
        }),
    palavraNormalizada.end());

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

    return palavraNormalizada;
}

vector<string> Index::divisaoPalavra(string pesquisa) {
    vector<string> todasPalavras;
    string palavra;
    istringstream iss(pesquisa);
    while (iss >> palavra) {
        todasPalavras.push_back(normalizarPalavra(palavra));
    }
    return todasPalavras;
}

//... Continua
```

Acima temos parte do código presente em index.cpp, nele temos algumas das bibliotecas que já foram declaradas. Duas das bibliotecas são usadas no método “normalizarPalavra”, eles permitem a interação com as strings e sua manipulação.

A terceira biblioteca é utilizada no método “divisaoPalavra”, este método é referenciado em pesquisa (também presente no index) e serve para manipular a entrada informada pelo usuário, assim a pesquisa pode ser realizada palavra a palavra.

Main

- main.cpp (src/main.cpp)

No arquivo main.cpp temos quase que a conexão entre os demais artigos, através dele se passa o diretório aonde os documentos estarão presentes e também aonde realizamos a pesquisa de acordo com o desejo do usuário. Neste arquivo ainda há o tratamento de erros (caso não seja possível ler o arquivo) e de exceções (caso a pesquisa não retorne nada como resultado). Neste arquivo também temos o uso das bibliotecas padrões do C++ e a importação dos demais arquivos (document.h e index.h).

```
// ... codigo
    } catch (falhaDocumento& e) {
        cout << "Não foi possível ler o documento: " << e.documentoComErro <<
endl;
    }
// ... continua
```

Acima temos o exemplo do que é realizado caso ocorra a falha ao tentar ler um documento.

Makefile

O arquivo Makefile também está presente na árvore do nosso projeto. Ele é um arquivo que serve para automatização do processo de compilação, onde há uma série de regras.

```
CFLAGS = -std=c++17
```

Como a regra acima que define que o programa deve ser executado com o C++ na versão 17. Além disso, ao executar o Makefile se cria dentro do diretório "obj/" executáveis "*.o" de todos os arquivos envolvidos na compilação.

Conclusão

Após a conclusão do trabalho, pudemos perceber todos os desafios que passamos para conseguirmos que determinada funcionalidade fosse atendida. Esses desafios foram de grande importância, pois desenvolvemos independência em como buscar soluções e interpretar o tipo de erro que havia ocorrido.

O GitHub a princípio foi um desafio, mas com a prática se demonstrou de extrema importância para que cada integrante pudesse executar mudanças e manter os demais atualizados.

Este trabalho ajudou no melhor entendimento da linguagem de programação e da lógica e também possibilitou aos integrantes que uma evolução dentro da própria. Ainda há o fato que experiência que este desafio proporcionou poderá ser útil em outros desafios em áreas distintas da programação.

Bibliografia

C++ isalpha(). Disponível em: <https://www.programiz.com/cpp-programming/library-function/cctype/isalpha>. Acesso em: 12 jun. 2023.

C++ isdigit(). Disponível em: <https://www.programiz.com/cpp-programming/library-function/cctype/isdigit>. Acesso em: 12 jun. 2023.

C++ ispunct(). Disponível em: <https://www.programiz.com/cpp-programming/library-function/cctype/ispunct>. Acesso em: 12 jun. 2023.

No title. Disponível em: <https://cplusplus.com/reference/cctype/ispunct/>. Acesso em: 12 jun. 2023a.

No title. Disponível em: <https://cplusplus.com/reference/cctype/>. Acesso em: 12 jun. 2023b.

No title. Disponível em: <https://cplusplus.com/reference/cstdlib/>. Acesso em: 12 jun. 2023c.

No title. Disponível em: <https://cplusplus.com/reference/algorithm/remove_if/?kw=remove_if>. Acesso em: 12 jun. 2023d.

using c++17+ - C++ Forum. Disponível em: <https://cplusplus.com/forum/beginner/267035/>. Acesso em: 12 jun. 2023.

Disponível em: <https://chat.openai.com/?model=text-davinci-002-render-sha>. Acesso em: 12 jun. 2023.