UNIVERDADE FEDERAL DE MINAS GERAIS

GRUPO

**MATEUS FELIPE DA SILVA - 2017014529**

**TÚLIO DIAS ALTÍSSIMO - 2017014375**

**TRABALHO FINAL DE PDS2**

**Documentação do Código**

Belo Horizonte, 20 de novembro de 2019

Verão 1.0.0

Esta é a documentação do trabalho final da Disciplina Programação e Desenvolvimento de Software II que foi desenvolvida por **Túlio Dias Altíssimo** e **Mateus Felipe da Silva**.

O Link para o GITHUB do nosso trabalho é:

**https://github.com/mateusf97/pds2-maquina-de-busca**

Basta no terminal fazer um **git clone** no link

**git clone https://github.com/mateusf97/pds2-maquina-de-busca.git**

Ou baixar diretamente do github.

Sumário

[Como compilar o programa? – MakeFile 5](#_Toc25336289)

[Entrada dos Arquivos 5](#_Toc25336290)

[Compilação com MakeFile 5](#_Toc25336291)

[Compilação e execução dos testes 6](#_Toc25336292)

[Explicação das classes 7](#_Toc25336293)

[Classe Leitura 7](#_Toc25336294)

[iniciarLeitura 7](#_Toc25336295)

[contarArquivos 7](#_Toc25336296)

[listarArquivos 7](#_Toc25336297)

[Classe Processar 7](#_Toc25336298)

[mapearPalavras 7](#_Toc25336299)

[imprimirRelacoes 7](#_Toc25336300)

[getIndice 7](#_Toc25336301)

[processarArquivos 7](#_Toc25336302)

[Classe Busca 8](#_Toc25336303)

[realizarBusca 8](#_Toc25336304)

[imprimirRank 8](#_Toc25336305)

[calcularRank 8](#_Toc25336306)

[calcularCoordenadas 8](#_Toc25336307)

[**calcularNx** 8](#_Toc25336308)

[**calcularIdf** 9](#_Toc25336309)

[**calcularTf** 9](#_Toc25336310)

[**calcularW** 9](#_Toc25336311)

[imprimirCoordenadas 9](#_Toc25336312)

[CosineRank 9](#_Toc25336313)

[imprimirTfs 9](#_Toc25336314)

[Estruturas de dados utilizadas 9](#_Toc25336315)

[Indice 9](#_Toc25336316)

[Coordenadas 10](#_Toc25336317)

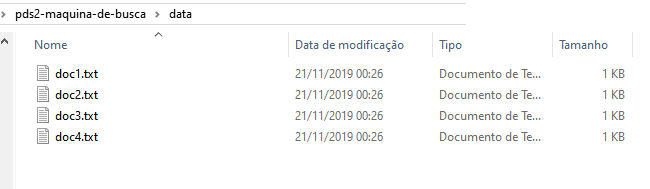
[Conclusão 10](#_Toc25336318)

# Como compilar o programa? – MakeFile

## Entrada dos Arquivos

Inicialmente você deverá colocar na pasta **data** os arquivos **.txt** do dataset que você quer usar no programa.

Exemplo:



Aqui neste exemplo, os 4 arquivos serão usados no programa.

## Compilação com MakeFile

Utilizamos make file para fazer a compilação do programa de forma reduzida.

Caso não tenha o MakeFile instalado no seu sistema, Linux ou Mac, basta digitar no terminal:

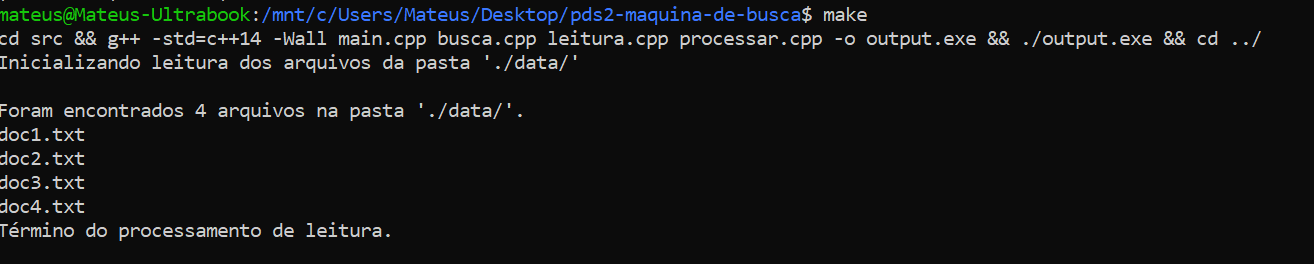
sudo apt Install make

sudo apt Install make-guile

Para compilar o programa, você deve caminhar até a pasta do projeto, e digitar a palavra **make** no seu terminal para poder compilar e executar o programa.

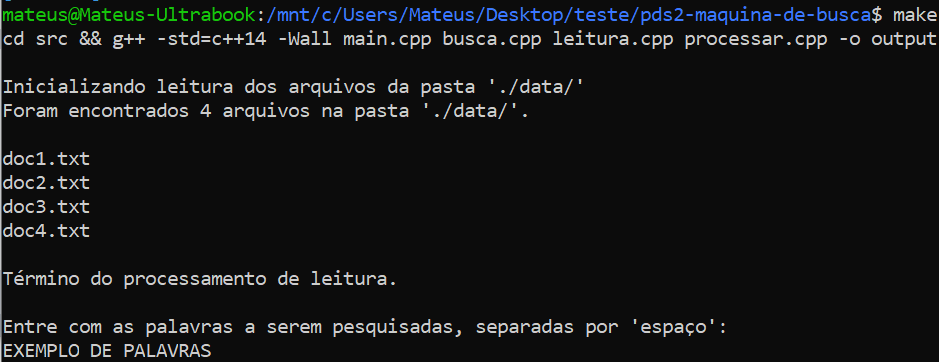
make

Você verá algo parecido com isso:

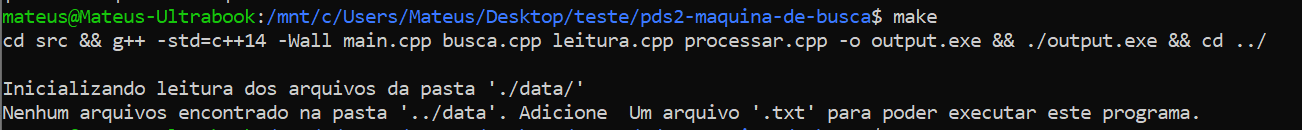


Ele irá lista os arquivos que ele encontrou na pasta **data.** E imprimir uma mensagem quando terminar de fazer o processamento da leitura. Não altere os arquivos enquanto esse comando estiver executando.

Após isso basta digitar **SEPARADO POR ESPAÇOS** a entrada que você quer pesquisar



Caso não tenha arquivos na pasta **data** você verá a seguinte mensagem:



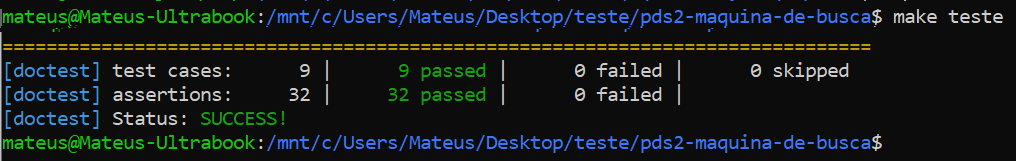
Lembre-se de dar **make clean**, quando quiser limpar os arquivos de saída ou rodar o **make teste**.

## Compilação e execução dos testes

Para executar os testes de unidade, executar o seguinte código no terminal:

make teste

Após isso os testes irão executar.



São 32 Checagens e Asserções em 9 teste cases que cobrem todas as classes utilizadas no projeto.

# Explicação das classes

## Classe Leitura

É onde tudo começa. A classe leitura realiza a leitura dos arquivos que estão na pasta **data**. Temos alguns métodos.

### iniciarLeitura

É a primeira função a ser chamada, faz algumas verificações na pasta **data** e verifica se existe acesso a ela.

### contarArquivos

Realiza a contagem do número de arquivos que está dentro da pasta **data** e retorna um inteiro**.**

### listarArquivos

Retorna um Vector com a listagem dos arquivos encontrados na pasta com a rota para eles

## Classe Processar

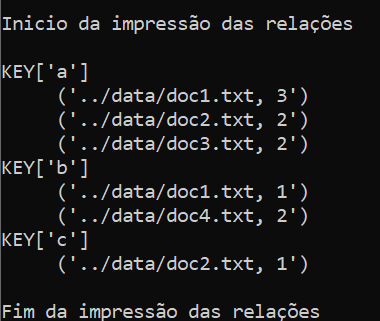
Realiza o processamento dos dados dos arquivos retornados na leitura. Aqui é criado o **índice.** Aqui temos dois métodos, a **processarArquivos**() que processa os arquivos e o **getIndice**() que retorna o índice já processado, e alguns métodos privados.

### mapearPalavras

Cria o **Map** com a relação de palavras por arquivos.

### imprimirRelacoes

Função de Debug para ver as relações do **Map** criada.



Aqui temos o mapeamento de quantas vezes as palavras ‘a’, ‘b’ e ‘c’ aparecem em cada um dos arquivos lindos.

### getIndice

Devolve o **Map** (o índice) processado.

### processarArquivos

Realiza o processamento do **Map** dos arquivos lidos.

## Classe Busca

A classe busca é a que realiza a busca do termo digitado pelo usuário no terminal. O Construtor dela recebe o termo a ser pesquisado e a partir daí uma série de funções privadas são executadas para calculas os valores do Cosine Rank. A Classe busca tem alguns atributos:

**Índice**: Índice gerado pelo processamento dos arquivos.

**Coordenadas**: Mapeamento das coordenadas das palavras.

**Endereços**: Endereço dos arquivos que são usados. (Que foram deixados na pasta **Data**)

E também atributos intermediários que são usados no CosineRank:

**N**: Número de arquivos

**Nx**: Número de arquivos com a palavra da pesquisa

**Idf**: Relação Log na base 2 de N por Nx

**Tf**: Quantas vezes o termo de pesquisa aparece em cada arquivo

**W**: Relação de cada Tf \* Idf por cada arquivo

A Classe conta com dois métodos públicos além do construtor e destrutor:

### realizarBusca

Recebe a entrada do usuário realiza a busca do termo criando o Rank de resultados utilizando o CosineRank com as coordenadas dos arquivos.

### imprimirRank

Imprime o rank da busca realizada.

A classe também conta com uma série de funções privadas:

### calcularRank

É a função que é chamada pelo **realizarBusca** para calcular o Rank e chamar as funções auxiliar para criar as coordenadas.

### calcularCoordenadas

Calcula as coordenadas para cada palavra existente no Índice.

Para calcula as coordenadas esse método chama os seguintes métodos:

### **calcularNx**

Calcula a quantidade de arquivos com a palavra especificada

### **calcularIdf**

Calcula a relação Idf para o CosineRank que relaciona Nx e o Número de arquivos.

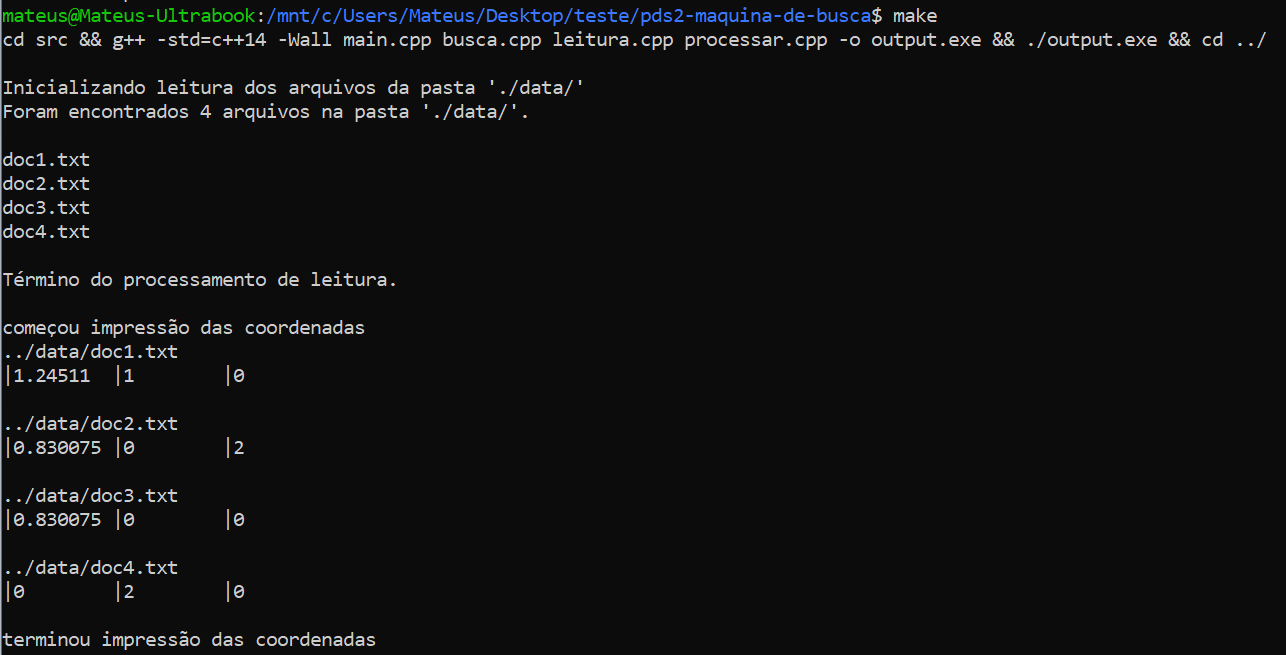
### **calcularTf**

Calcula a quantidade de vezes que o termo pesquisado aparece em cada arquivos.

### **calcularW**

Calcula a relação W que relaciona o Tf com o Idf para cada arquivo.

### imprimirCoordenadas

Função de debug para realizar a impressão das coordenadas

Cada coluna é uma palavra em um arquivo.

### CosineRank

Última função a ser chamada. Após reunir todos os dados que são necessários para o cálculo do CosineRank a função calcula o CosineRank.

### imprimirTfs

Função de debug para ver os Tfs.

# Estruturas de dados utilizadas

Algumas estruturas de dados mais complexas para fazer as matrizes:

1. std::map<std::string, std::map<std::string, **int**>> indice;
2. // Indice Invertido
4. std::map<std::string, std::vector<**double**>> coordenadas;
5. // Map com as coordenadas do processamento das palavras

### Indice

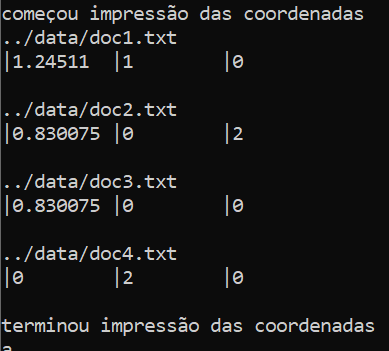
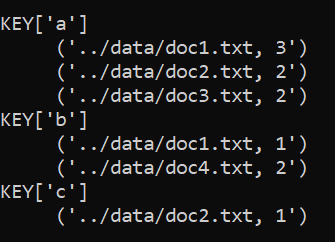
Para o índice fizemos um Map de Map:

O Map externo é composto por uma String e Outro Map, a string referencia as palavras existentes nos arquivos. O Map interno é uma String por Int. A String referencia o arquivo e o int quantas vezes a palavra aparece no arquivo.

As palavras ‘a’, ’b’ e ‘c’ são as Keys do MAP externo. Os documentos são as Keys do Map interno e o valor é a quantidade de vezes que a palavra aparece no arquivo.

Coordenadas

As coordenadas são o Map de String por Vector de doubles. A string guarda o arquivo e o Vector as coordenadas de cada palavra do arquivo.

# Saída do programa

# Conclusão

Cremos que as estruturas de dados e a forma com que organizamos as classes e modularizamos o projeto foram suficientes para o aprendizado. Achamos que os resultados estão coerentes nos testes que fizemos e obtivemos sucesso nos aprendizados de usar de forma prática classes e Estruturas de dados a fundo.