

UNIVERSIDADE 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	4
Entrada dos Arquivos	4
Compilação com MakeFile	4
Compilação e execução dos testes	5
Saída do Programa	6
Explicação das classes	7
Classe Leitura	7
iniciarLeitura	7
contarArquivos.....	7
listarArquivos	7
Classe Processar.....	7
mapearPalavras.....	7
imprimirRelacoes	7
getIndice	8
processarArquivos.....	8
Classe Busca	8
realizarBusca	8
imprimirRank	8
calcularRank.....	8
calcularCoordenadas.....	9
calcularNx	9
calcularIdf	9
calcularTf	9
calcularW	9
imprimirCoordenadas	9
CosineRank.....	9
imprimirTfs.....	9
Estruturas de dados utilizadas	10
Índice	10
Coordenadas	10
Conclusão.....	10

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:

> pds2-maquina-de-busca > data				
Nome	Data de modificação	Tipo	Tamanho	
doc1.txt	21/11/2019 00:26	Documento de Te...	1 KB	
doc2.txt	21/11/2019 00:26	Documento de Te...	1 KB	
doc3.txt	21/11/2019 00:26	Documento de Te...	1 KB	
doc4.txt	21/11/2019 00:26	Documento de Te...	1 KB	

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:

```
mateus@Mateus-Ultrabook: /mnt/c/Users/Mateus/Desktop/pds2-maquina-de-busca$ make
cd src && g++ -std=c++14 -Wall main.cpp busca.cpp leitura.cpp processar.cpp -o output.exe && ./output.exe && cd ../
Iniciando leitura dos arquivos da pasta './data/'

Foram encontrados 4 arquivos na pasta './data/'.
doc1.txt
doc2.txt
doc3.txt
doc4.txt
Término do processamento de leitura.
```

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

```
mateus@Mateus-Ultrabook:/mnt/c/Users/Mateus/Desktop/teste/pds2-maquina-de-busca$ make
cd src && g++ -std=c++14 -Wall main.cpp busca.cpp leitura.cpp processar.cpp -o output

Iniciando leitura dos arquivos da pasta './data/'
Foram encontrados 4 arquivos na pasta './data/'.

doc1.txt
doc2.txt
doc3.txt
doc4.txt

Término do processamento de leitura.

Entre com as palavras a serem pesquisadas, separadas por 'espaço':
EXEMPLO DE PALAVRAS
```

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

```
mateus@Mateus-Ultrabook:/mnt/c/Users/Mateus/Desktop/teste/pds2-maquina-de-busca$ make
cd src && g++ -std=c++14 -Wall main.cpp busca.cpp leitura.cpp processar.cpp -o output.exe && ./output.exe && cd ../

Iniciando leitura dos arquivos da pasta './data/'
Nenhum arquivos encontrado na pasta './data/'. Adicione Um arquivo '.txt' para poder executar este programa.
```

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.

```
mateus@Mateus-Ultrabook:/mnt/c/Users/Mateus/Desktop/teste/pds2-maquina-de-busca$ make teste
=====
[doctest] test cases:      9 |      9 passed |      0 failed |      0 skipped
[doctest] assertions:    32 |     32 passed |      0 failed |
[doctest] Status: SUCCESS!
mateus@Mateus-Ultrabook:/mnt/c/Users/Mateus/Desktop/teste/pds2-maquina-de-busca$
```

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

Saída do Programa

Após a pesquisa é retornando o ranking com os arquivos.

```
mateus@Mateus-Ultrabook:/mnt/c/Users/Mateus/Desktop/teste/pds2-maquina-de-busca$ make
cd src && g++ -std=c++14 -Wall main.cpp busca.cpp leitura.cpp processar.cpp -o output.

Inicializando leitura dos arquivos da pasta './data/'
Foram encontrados 4 arquivos na pasta './data/'.

doc1.txt
doc2.txt
doc3.txt
doc4.txt

Término do processamento de leitura.

Entre com as palavras a serem pesquisadas, separadas por 'espaço':
A casa

Ranking do termo de busca: a
1º Arquivo doc3.txt -> Nível de similaridade: 1
2º Arquivo doc1.txt -> Nível de similaridade: 0.779673
3º Arquivo doc2.txt -> Nível de similaridade: 0.383333
4º Arquivo doc4.txt -> Nível de similaridade: 0
```

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.

```
Inicio da impressão das relações

KEY['a']
    ('../data/doc1.txt, 3')
    ('../data/doc2.txt, 2')
    ('../data/doc3.txt, 2')
KEY['b']
    ('../data/doc1.txt, 1')
    ('../data/doc4.txt, 2')
KEY['c']
    ('../data/doc2.txt, 1')

Fim da impressão das relações
```

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

`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 calcular 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

```
mateus@Mateus-Ultrabook:/mnt/c/Users/Mateus/Desktop/teste/pds2-maquina-de-busca$ make
cd src && g++ -std=c++14 -Wall main.cpp busca.cpp leitura.cpp processar.cpp -o output.exe && ./output.exe && cd ../

Iniciando leitura dos arquivos da pasta './data/'
Foram encontrados 4 arquivos na pasta './data/'.

doc1.txt
doc2.txt
doc3.txt
doc4.txt

Término do processamento de leitura.

começou impressão das coordenadas
../data/doc1.txt
|1.24511 |1      |0
../data/doc2.txt
|0.830075 |0      |2
../data/doc3.txt
|0.830075 |0      |0
../data/doc4.txt
|0      |2      |0

Cada coluna é uma palavra em um arquivo.
Cada linha é uma palavra em um arquivo.
Cada função a ser chamada. Após reunir todos os dados que são necessários para o
calcula o CosineRank a função calcula o CosineRank.

terminou impressão das coordenadas
```

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  
3.  
4. std::map<std::string, std::vector<double>> coordenadas;  
5. // Map com as coordenadas do processamento das palavras  
6.
```

Índice

Para o índice fizemos um Map de Map:

O **Map** externo é composto por uma **String** e Outro **Map**, a string referência as palavras existentes nos arquivos. O **Map** interno é uma **String** por **Int**. A **String** irá referenciar 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.

```
começou impressão das coordenadas  
../data/doc1.txt  
|1.24511 |1      |0  
  
../data/doc2.txt  
|0.830075 |0      |2  
  
../data/doc3.txt  
|0.830075 |0      |0  
  
../data/doc4.txt  
|0        |2      |0  
terminou impressão das coordenadas
```

```
KEY['a']  
  ('../data/doc1.txt', 3')  
  ('../data/doc2.txt', 2')  
  ('../data/doc3.txt', 2')  
KEY['b']  
  ('../data/doc1.txt', 1')  
  ('../data/doc4.txt', 2')  
KEY['c']  
  ('../data/doc2.txt', 1')
```

Conclusão

Creemos 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.

Obrigado :D.