

Universidade Federal de Minas Gerais

Trabalho Prático: Máquina de Busca

João Guilherme da Costa
Matheus Brito Faria
Victor Emmanuel Dias de Barros Campos

Versão 1.5
Terça, 11 de Junho de 2019

Sumário

Introdução	3
Proposta	3
Máquina utilizada	3
Implementação	3
Etapas	3
Testes	4
Erros	4
Conclusão	4
Classe FileSearch	5
Referência do Arquivo filesearch.h	5
Membros Públicos	5
Membros Privados	5
Atributos Privados	5
Amigas	5
Construtores e Destrutores	6
FileSearch::FileSearch (string directory)	6
Funções membros	6
void FileSearch::find (string & query)	6
double FileSearch::cosine_distance (vector< double > & a, vector< double > & b)[private]	6
void FileSearch::list_file (string directory)[private]	6
void FileSearch::read_files (string directory)[private]	6
vector< string > FileSearch::return_query (string & query)[private]	6
void FileSearch::set_idf ()[private]	6
void FileSearch::set_w_file_word ()[private]	6
void FileSearch::set_w_query (vector< string > & words)[private]	6
Amigas e Funções Relacionadas	6
friend class Teste[friend]	6
Atributos	7
map<string,set<string> > FileSearch::coordinate[private]	7
vector<string> FileSearch::files_names[private]	7
map<string, double> FileSearch::inverse_document_frequency[private]	7
map<string, map<string, double> > FileSearch::term_frequency[private]	7
map<string, vector<double> > FileSearch::w_file_word[private]	7
vector<double> FileSearch::w_query[private]	7
Funções	7
int main ()	7
Como compilar e usar	8

Introdução

Proposta

O Trabalho Prático: Máquina de Busca instrui construir um algoritmo que possa recuperar informações, neste caso, especificamente, de um dataset de arquivos de texto, com base em uma consulta realizada por um possível usuário.

A estrutura de organização e a localização dos arquivos é previamente conhecida e espera-se que a consulta realizada por um usuário não possua caracteres especiais.

A apresentação dos resultados, assim, mostrará os arquivos ordenadamente mais relevantes.

Máquina utilizada

A classe foi implementada e testada no Ubuntu 18.04.2 LTS com 6 GB de memória principal.

Implementação

Etapas

Após compreendida o prospecto de o que se espera de uma máquina de busca baseada no algoritmo de Information Retrieval, procuramos descrever as funções e etapas que o código iria realizar. As fases foram definidas na ordem em que se espera, sendo: Leitura dos Arquivos do Dataset, criação do índice Invertido, marcação do inverso da frequência nos arquivos, Formatação da Query Consultada, marcação da frequência de um termo da query dentro de um certo arquivo, estabelecimento das coordenadas vetoriais em relação no plano com uma determinada palavra e o ranqueamento com base na similaridade entre o que foi consultado com o que há nos arquivos.

Para a Leitura dos Arquivos do Dataset, procuramos primeiramente estabelecer quais são os nomes dos documentos dentro de um directório que contém a coleção do arquivos e assim armazená-los em um vetor, `vector<string>`. Foi usado uma biblioteca que possui as funções para manipulação e leitura de diretórios, com isso criamos uma função própria para fazer essa leitura. Tendo armazenado os nomes dos arquivos em um atributo da nossa classe, definimos um método que através de um loop percorrerá linha a linha de arquivo a arquivo determinando e alimentando dois containers atributos de `FileSearch()`, `map<string, set<string>>` e `map<string, map<string, double>>`, que são, respectivamente, o índice invertido e a frequência de cada palavra dentro do arquivo.

Para o cálculo do Inverso da Frequência nos Documentos, tendo os atributos, número total de documentos e número total de documentos que termos da consulta aparecem, definimos outro método com o objetivo de criar um processo *callable*. Nesse método dois processos são realizados: Uma variável local do tipo `double` é inicializada com a quantidade de arquivos totais e o segundo processo é a contagem através de um loop, usando o container do índice invertido, de quantos documentos

um determinado termo aparece. Com isso é feito a criação dos vetores necessários, e os armazenando em `w_file_word`.

Para a formatação da query fornecida por um possível usuário estabelecemos um método dentro da Classe Principal (Ou chamamos de Estrutura de Dados, verificar na revisão), `FileSearch()`, com o objetivo de verificar se entre os caracteres de entrada existe alguma pontuação que pode ocasionar algo que não foi preparado. Após verificado, o próximo passo será dividir a consulta com mais de um termo, se for o caso, como valores de entradas dos índices de um vetor de strings, `vector<string>`.

Todos os próximos passos do algoritmo acontecem e são executados dentro de um loop infinito que para apenas quando o usuário dá o comando `ctrl + "c"` ou fecha a execução do programa.

Durante a execução do loop ocorre, sempre que uma nova pesquisa for realizada, o cálculo das coordenadas dos vetores em relação a um termo da query (que pode ou não possuir mais de uma palavra. O número de termos dimensiona o vetor), sua normalização e o estabelecimento da similaridade com o plano.

Testes

Visando observar se o algoritmo realmente retornava o que era esperado, foi criado um diretório com o nome "teste" com quatro arquivos como segue o exemplo demonstrado no PDF de apresentação do TP.

Com isso, no arquivo `teste.cpp` foi criada uma classe amiga `Teste` que retorna além de resultado de variáveis, alguns retornos de funções para que a maior parte do código seja coberto.

A maioria das funções da classe são privadas, pois não há utilidade para o usuário utilizá-las, o usuário apenas está permitindo criar uma classe e usar uma função única, que serve apenas para a busca de alguma query, criando assim uma abstração maior para o usuário.

Como alguns resultados retornam valores numérico muito grande e com diversas casas decimais, optamos para no teste usar a função `floor` do `cmath` para limitar esse valor a duas casa decimais e assim poder fazer o teste.

Erros

Diversos erros ocorreram durante a implementação da classe, a maioria em função de não limparmos algumas variáveis entre buscas, o que causava uma falha de segmentação e uma sobrescrita do vetor causando resultados diversos.

Com o uso do dataset disponibilizado, melhoramos a retirada de caracteres, rodando o mesmo algoritmo mais de uma vez para evitar que pontuação consecutivas fossem adicionadas.

O arquivo de teste apresenta um problema que pode variar de compilador onde, a leitura dos arquivos podem ocorrer em ordem diferente, isso não é um problema para o algoritmo, mas pode gerar um erro no arquivo de teste.

Conclusão

Durante a execução e revisão do Código conseguimos todo o resultado esperado, que é apresentar entre os arquivos de um dataset aqueles os quais seriam mais relevantes conforme a solicitação do usuário.

Classe FileSearch

```
#include <filesearch.h>
```

Referência do Arquivo filesearch.h

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <list>
#include <sstream>
#include <algorithm>
#include <map>
#include <set>
#include <cmath>
#include <dirent.h>
#include <string.h>
```

Membros Públicos

- **FileSearch** (string directory)
- void **find** (string &query)

Membros Privados

- void **list_file** (string directory)
- void **set_idf** ()
- void **read_files** (string directory)
- void **set_w_file_word** ()
- void **set_w_query** (vector< string > &words)
- vector< string > **return_query** (string &query)
- double **cosine_distance** (vector< double > &a, vector< double > &b)

Atributos Privados

- vector< string > **files_names**
- map< string, set< string > > **coordinate**
- map< string, map< string, double > > **term_frequency**
- map< string, double > **inverse_document_frequency**
- map< string, vector< double > > **w_file_word**
- vector< double > **w_query**

Amigas

- class **Teste**()

Construtores e Destrutores

FileSearch::FileSearch (string *directory*)

Construtor da classe, o qual realizará a leitura dos arquivos de um dataset que encontra-se em um diretório que deve estar em dentro da pasta FileSearch com arquivos de texto dentro. OBS: a leitura dos arquivos só será possível se ela se encontrar dentro desse diretório.

Além disso os métodos que criam as variáveis relacionadas aos arquivos também são chamados.

Funções membros

void FileSearch::find (string & *query*)

Método usado para encontrar a string desejada.

double FileSearch::cosine_distance (vector< double > & *a*, vector< double > & *b*)[private]

Método que retorna a distância de cossenos de dois vetores.

void FileSearch::list_file (string *directory*)[private]

Método que lê os arquivos de um diretório os armazena em um vetor.

void FileSearch::read_files (string *directory*)[private]

Método que faz a leitura dos arquivos e chama os outros métodos.

vector< string > FileSearch::return_query (string & *query*)[private]

Método que retorna a consulta em em vetor retirando pontos e letras maiúsculas.

void FileSearch::set_idf ()[private]

Método que armazenará os valores no container map<string, double> FileSearch::inverse_document_frequency[private] .

void FileSearch::set_w_file_word ()[private]

Método que atribui os valores ao vetor vector<double> FileSearch::w_file_word[private] de acordo com os arquivos.

void FileSearch::set_w_query (vector< string > & *words*)[private]

Método que atribui os valores ao vetor vector<double> FileSearch::w_query[private] de acordo com os arquivos.

Amigas e Funções Relacionadas

friend class Teste[friend]

Atributos

map<string, set<string> > FileSearch::coordinate [private]

Variável que recebe o nome da palavra e retorna um set com os arquivos onde essa palavra se encontra.

vector<string> FileSearch::files_names [private]

Variável que armazena o nome dos arquivos.

map<string, double> FileSearch::inverse_document_frequency [private]

Variável que retorna a frequência inversa do documento, recebe a palavra como key.

map<string, map<string, double> > FileSearch::term_frequency [private]

Variável que retorna a frequência do termo, com as keys nome do arquivo e palavra.

map<string, vector<double> > FileSearch::w_file_word [private]

Variável map que recebe o nome do arquivo como key e retorna o vetor para o cálculo de similaridade.

vector<double> FileSearch::w_query [private]

Variável que armazena o vetor da busca para o cálculo da similaridade.

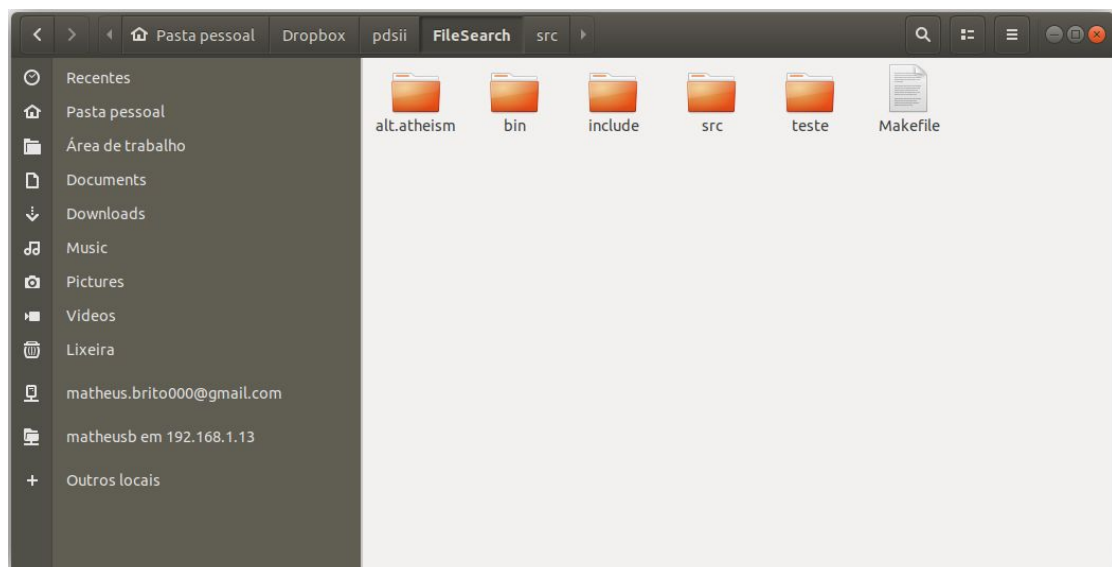
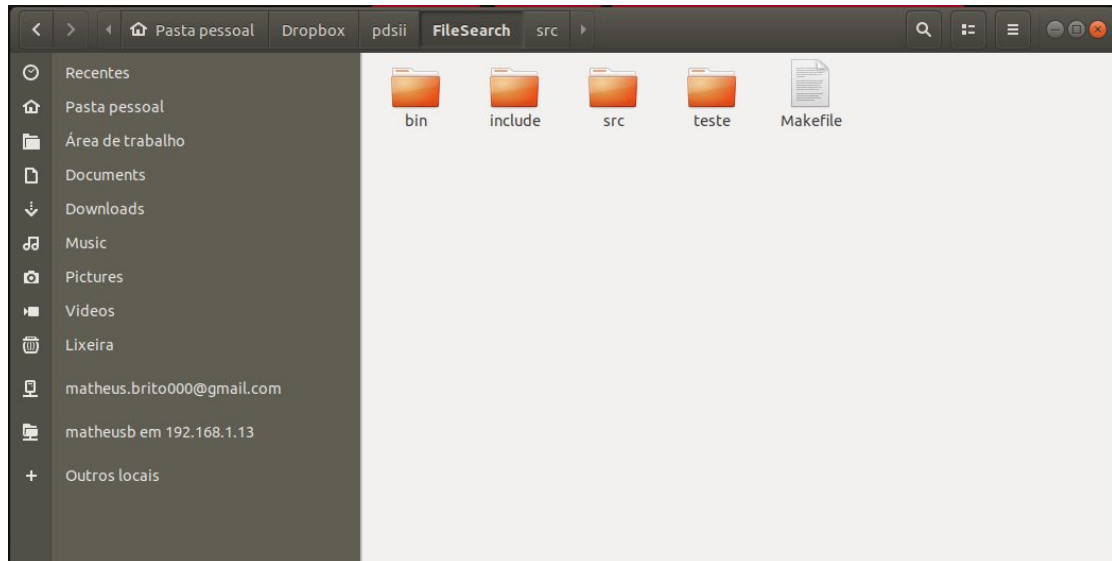
Funções

int main ()

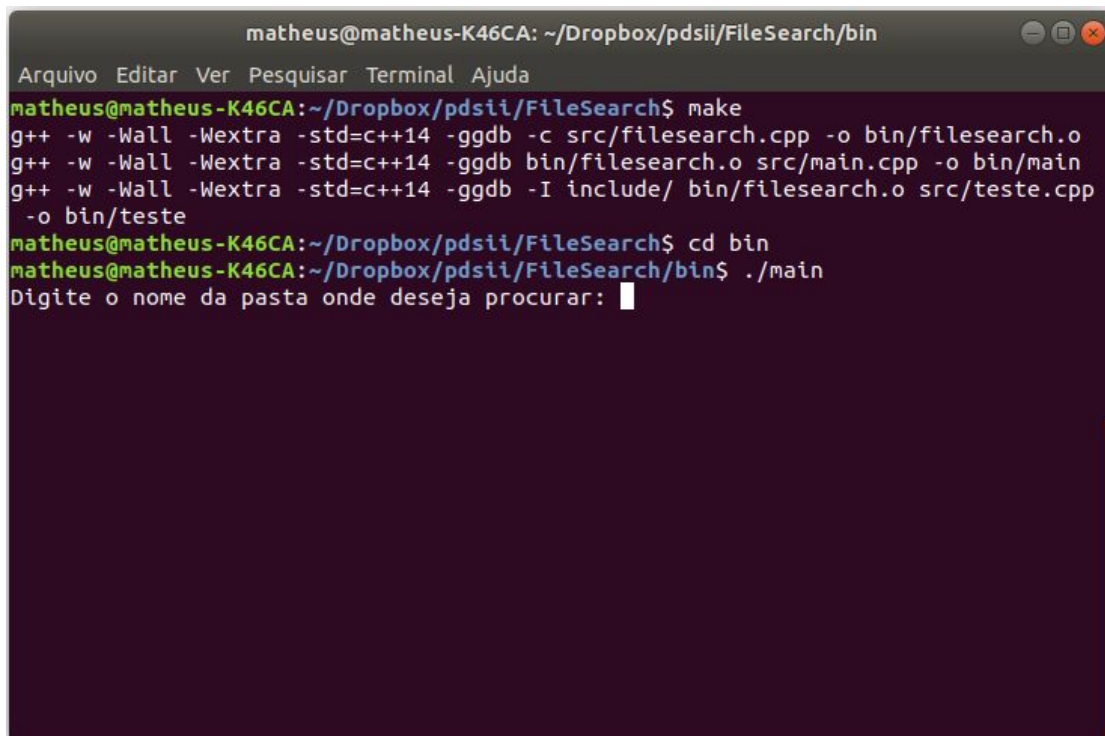
Cria uma plataforma para usar a classe, onde com o diretório com os arquivos de texto já estiver dentro de FileSearch, é necessário colocar o nome do diretório, e depois pesquisar a palavra desejada.

Como compilar e usar

Todo código funciona de uma maneira bem intuitiva, antes de tudo é necessário adicionar os arquivos de texto dentro do diretório FileSearch:

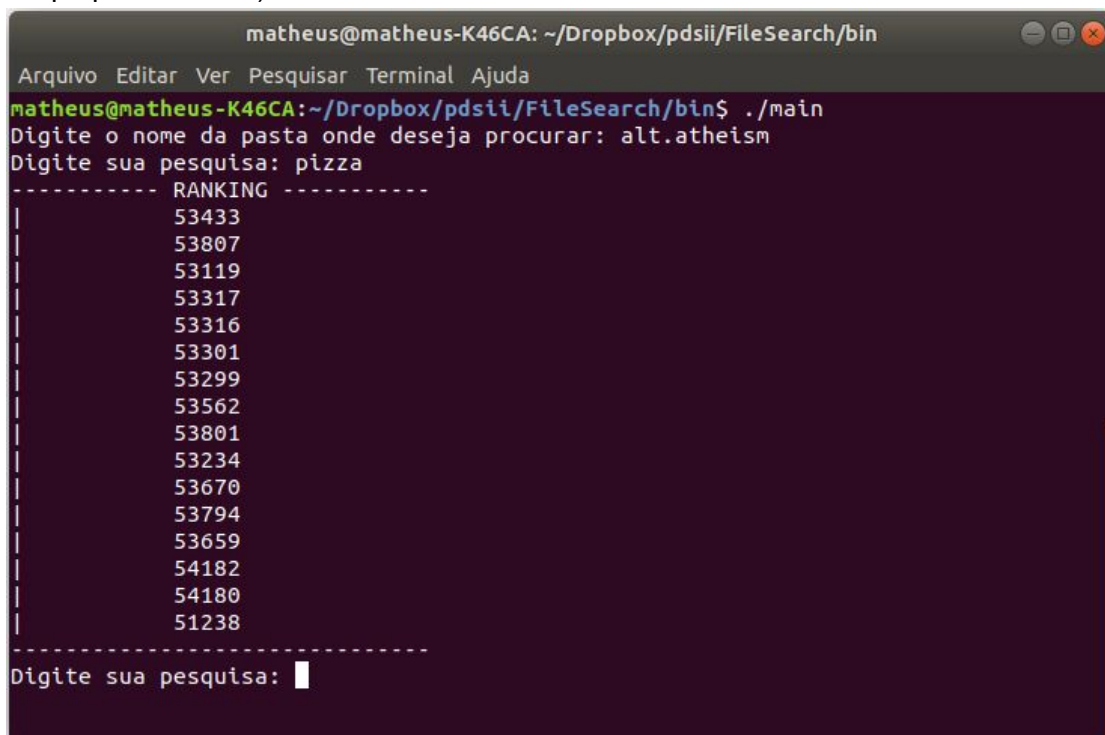


Depois podemos compilar o código usando o arquivo Makefile criado, e entrar na pasta bin onde será encontrado o arquivo main e teste.



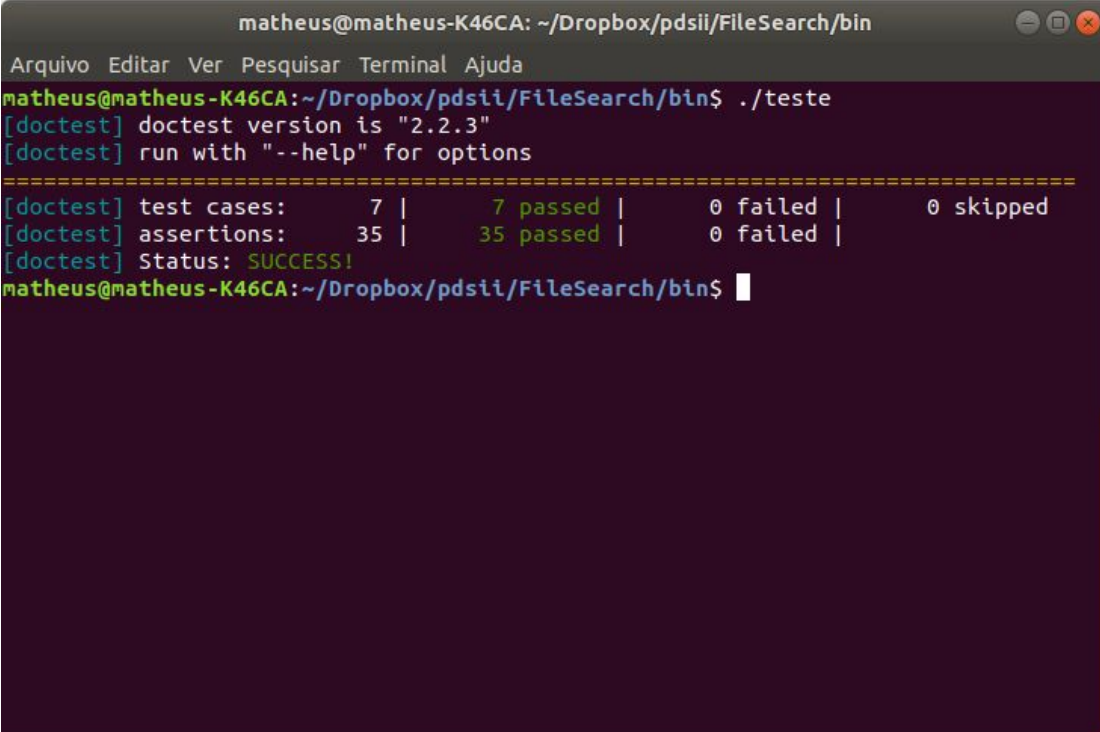
```
matheus@matheus-K46CA: ~/Dropbox/pdsii/FileSearch/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
matheus@matheus-K46CA:~/Dropbox/pdsii/FileSearch$ make
g++ -w -Wall -Wextra -std=c++14 -ggdb -c src/filesearch.cpp -o bin/filesearch.o
g++ -w -Wall -Wextra -std=c++14 -ggdb bin/filesearch.o src/main.cpp -o bin/main
g++ -w -Wall -Wextra -std=c++14 -ggdb -I include/ bin/filesearch.o src/teste.cpp
-o bin/teste
matheus@matheus-K46CA:~/Dropbox/pdsii/FileSearch$ cd bin
matheus@matheus-K46CA:~/Dropbox/pdsii/FileSearch/bin$ ./main
Digite o nome da pasta onde deseja procurar: 
```

Escrevendo exatamente o mesmo nome do diretório que foi colocado na pasta, todos os arquivos serão lidos. (O tempo de processamento pode variar muito, no caso dos arquivos do dataset, que são bem grandes, o algoritmo gasta bastante tempo para ler tudo).



```
matheus@matheus-K46CA: ~/Dropbox/pdsii/FileSearch/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
matheus@matheus-K46CA:~/Dropbox/pdsii/FileSearch/bin$ ./main
Digite o nome da pasta onde deseja procurar: alt.atheism
Digite sua pesquisa: pizza
----- RANKING -----
| 53433
| 53807
| 53119
| 53317
| 53316
| 53301
| 53299
| 53562
| 53801
| 53234
| 53670
| 53794
| 53659
| 54182
| 54180
| 51238
-----
Digite sua pesquisa: 
```

E para rodar o arquivo de teste segue o mesmo padrão.



```
matheus@matheus-K46CA: ~/Dropbox/pdsii/FileSearch/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
matheus@matheus-K46CA:~/Dropbox/pdsii/FileSearch/bin$ ./teste
[doctest] doctest version is "2.2.3"
[doctest] run with "--help" for options
=====
[doctest] test cases:      7 |      7 passed |      0 failed |      0 skipped
[doctest] assertions:    35 |     35 passed |      0 failed |
[doctest] Status: SUCCESS!
matheus@matheus-K46CA:~/Dropbox/pdsii/FileSearch/bin$
```