UNIVERSIDADE FEDERAL DE MINAS GERAIS

Trabalho prático Máquina de Busca

Alunos:
Geam Lauriano Alves
Júlia Fernanda Teixeira de Miranda
Kelly Katiussy Pereira

Belo Horizonte, junho de 2019

Sumário

1. Introdução

Máquina de busca (também chamada de motor de busca, ferramenta de busca ou buscador) é um programa que tem como objetivo encontrar palavras-chaves fornecidas pelo usuário em documentos ou base de dados. É uma ferramenta importante e indispensável no meio de acesso à informação.
O presente trabalho tem como proposta o desenvolvimento de uma máquina de busca que fará a leitura de diversos arquivos previamente fornecidos e, então receberá do usuário uma palavra-chave. O programa fará a busca pela palavra fornecida pelo usuário e apresentará os arquivos mais relevantes, conforme um ranking que será apresentado posteriormente.
Conforme orientações fornecidas para a realização deste trabalho prático, algumas preocupações tiveram que ser tomadas, tais como o tratamento dos textos lidos, a forma que o programa deve *rankear* os resultados obtidos para o usuário. Essas preocupações serão mais bem detalhadas posteriormente.

1.1 Vantagem do índice invertido

As vantagens de utilização do índice invertido são:

- Permite pesquisas rápidas em textos completos;
- É fácil de desenvolver;
- É a estrutura de dados mais popular usada em sistemas de recuperação de documentos, usada em grande escala, por exemplo, em mecanismos de pesquisa como Google.

1.2 Desvantagens Índice

- Grande sobrecarga de armazenamento e altos custos de manutenção na atualização, exclusão e inserção.

2 Código

O trabalho realizado pode ser entendido por três partes bem definidas. A primeira consiste na leitura dos arquivos e na preparação dos seus dados, de modo que é convertido numa string ideal na qual é realizada a busca. Essa etapa foi desenvolvida por meio de uma estrutura de dados. A segunda parte consiste na busca propriamente dita e na classificação dos arquivos associados à busca. A terceira parte consiste na realização de testes de unidades, em que os métodos desenvolvidos são testados, cujo intuito é maximizar a assertividade do código.

3 Decisões e Requisitos de Implementação

3.1 Requisitos de Implementação

Para a implementação do programa foi solicitado que:

- Fosse utilizada uma estrutura do tipo <map>;
- Somente fossem considerados caracteres alfanuméricos sendo que qualquer outro símbolo deveria ser desconsiderado;
- Todas as palavras devem conter somente letras minúsculas;
- A inserção dos dados deve ser feita palavra por palavra;
- Os resultados devem ser apresentados de acordo com o raking.

3.2 Principais decisões

A máquina de busca foi desenvolvida utilizando a linguagem c++, na versão c++11 2011 Standart.
A implementação foi dividida nas seguintes etapas:

- Busca dos documentos a serem utilizados, para tal foi utilizado a biblioteca “dirent”.
- Inserção palavra a palavra de cada documento calculando a frequência de cada palavra em uma estrutura do tipo “map”.
- Inserção do “map” gerado no tópico anterior em um “multimap” contendo a palavra, a frequência e o nome do arquivo.
- Calculo do df e ranking.

4 Implementação

A princípio, o sistema recebe a pasta principal que contém subpastas com os arquivos a serem lidos utilizando a biblioteca “dirent” essas pastas são abertas e o nome dos arquivos a serem lidos são gravados em uma string. Esses arquivos são abertos e lidos, palavra por palavra e então é construído o índice invertido.
Para a leitura dos arquivos optou-se pela utilização da biblioteca “fstream”, cogitou-se a atualização da biblioteca “boost” mas percebemos que a diferença de velocidade de leitura dos arquivos é mínima e a biblioteca.

Ao realizar a leitura dos arquivos, os dados lidos são devidamente tratados, o que otimiza o desenvolvimento das funções associadas à busca.
Os três primeiros tratamentos citados ocorrem na função “copiar arquivo”, por meio de funções simples da linguagem. O quarto ocorre na função “remover” que é responsável por retirar qualquer caractere que não seja alfanumérico.

```
void Index::copiar_arquivo(Armazenar_arquivos &nomearquivos){
        for( int x = 0; x <= nomearquivos.nomearquivos.size(); x++ ){
        std::ifstream f;
        std::string name = "./20news-19997//20_newsgroups//" + nomearquivos.nomearquivos[x];
        f.open( name.c_str(), std::ios::in );
        if(f.is_open()){
            while( !f.eof() ) {
                f >> s;
                std::transform(s.begin(), s.end(), s.begin(),::tolower);
                std::string ert=remover(s);
                adicionar( x+1,ert, nomearquivos.nomearquivos[x]);
                s.clear();
            }
            f.close();
        }
```

Figura 1- Função copiar arquivo

```
std::string Index::remover(std::string text)
{

   for(int i=0;i<text.length();i++){
        if (!isalnum(text[i])){
        text.erase(i,1);
    }
}
return text;
}
```

Figura 2- Função remover

Na sequência, todo o arquivo lido e tratado é armazenado em um multimap. Que funciona como um dicionário com umas estruturas associativa que armazenam elementos combinando uma chave com um valor. As chaves são as palavras contidas nos documentos, a frequência da palavra no documento e o nome do documento a qual ela está associada.

```
}
for (std::map<std::string, int>::iterator itx = sub_map.begin(); itx != sub_map.end(); ++itx){
 Ar_Index.insert(std::make_pair(itx->first,Chave_interna(nomearquivos.nomearquivos[x],itx->second)));
}
sub_map.clear();
```

Figura 3- Função adicionar

Após a adição de todas as palavras no “map” é possível realizar a busca nos documentos. Na função encontrar é realizado o cálculo do idf e do w(dj ,Pt) que é utilizado na função cossine raking.

```
void Index::encontrar(Armazenar_arquivos &nomearquivos,std::string busca){
        double idf=log(nomearquivos.nomearquivos.size()/Ar_Index.count(busca));
        auto eqr = Ar_Index.equal_range(busca);
        if (eqr.first!=eqr.second){
        std::cout << "A Palavra: "<< busca<< " foi achada no(s) arquivo(s):" << '\n';
        auto st = eqr.first, en = eqr.second;
            for(auto itr = st; itr != en; ++itr){
            std::cout << itr->second.first << ", " << itr->second.second << std::endl;
            }
        }
        else{
        std::cout << "A Palavra: " << busca<< " nao foi encontrada em nenhum arquivo."<< '\n';
        }
}
```

Figura 4 - Função encontrar

Posteriormente, uma vez que o usuário realiza a busca, é realizada uma classificação dos arquivos que contêm as palavras buscadas, de acordo com a importância do arquivo para a busca.
Essa classificação é realizada por meio do método cosine ranking.

5 Testes de Unidades

Os Testes de unidade têm como objetivo testar cada fase do programa individualmente. Neste TP foram feitos os testes das funções de acordo com sua classe e, para isto, foi incluída a biblioteca chamada “doctest.h”.

5.1 CLASSE INDEX

5.1.1 Teste da função “copiar_arquivo”

Esta função consiste em abrir os arquivos mapeados e chamar a função “adicionar” para cada palavra lida dos arquivos. O teste consiste na criação de um objeto do tipo “Index” e de um objeto do tipo “Armazenar_arquivos”. Depois disso é chamada a função “adicionar” em que são passados seus parâmetros e é verificado se o texto que estava no arquivo foi escrito no vetor “stringvec”.

```
55  TEST_SUITE("Index"){
56      TESTE_CASE("copiar_arquivo"){
57          Index t;
58          Armazenar_arquivos tl;
59          tl.nomearquivos[0] = {"arquivo1.txt"};
60          t.copiar_arquivo(tl.nomearquivos);
61          CHECK(tl.stringvec[0] == "a");
62          CHECK(tl.stringvec[1] == "b");
63          CHECK(tl.stringvec[2] == "c");
64          CHECK(tl.stringvec[3] == "d");
65      }
```

5.1.2 Teste da função “remover”

Esta função tem como objetivo remover todos caracteres inválidos para a leitura do arquivo. Para testá-la, foi criada uma string com alguns caracteres inválidos. Posteriormente é passada esta string como parâmetro para a função e em seguida, é checado se a string ainda contém os caracteres inválidos.

```
66      TEST_CASE("remover"){
67          std::string s = "ab'{}cde";
68          s = remover(s);
69          CHECK(s=="abcde");
70      }
```

5.1.3 Teste da função “adicionar”

Esta função consiste em adicionar determinada palavra, lida no arquivo, em um vector “stringvec”. O teste consiste na criação de um objeto do tipo “Index” e de uma string. É passado como parâmetro esta string para a função adicionar e depois disso, é checado se a string foi adicionada ao vector “stringvec”.

```
72      TEST_CASE("adicionar"){
73          Index t;
74          std::string arquivo_teste = "arquivo_teste1.txt";
75          std::string s; = "abcde";
76          t.adicionar(0,s, nome_arquivo);
77          CHECK(stringvec->second == arquivo_teste);
78          CHECK(stringvec->first == s);
79  }
```

5.2 CLASSE ARMAZENAR_ARQUIVOS

5.2.1 - Teste da função “checar maiúscula”

Esta função tem como objetivo transformar as letras maiúsculas de uma string em letras minúsculas. Para testá-la, foi criado um vector com algumas strings que, por sua vez, é passado como parâmetro e posteriormente checado para averiguar se as strings estão com letras minúsculas.

```
12  TEST_SUITE ("Armazenar_arquivos"){
13      TEST_CASE("checar maiuscula"){
14          std::vector <std::string> vetor {"A","B","C","D"};
15          for(int i = 0 ;i < 4; i++){
16              chechar_maiscula(vetor[i]);
17          }
18          CHECK(t.vetor[0] == "a");
19          CHECK(t.vetor[1] == "b");
20          CHECK(t.vetor[2] == "c");
21          CHECK(t.vetor[3] == "d");
22          }
```

5.2.2 Teste da função “nome diretórios”

Esta função tem como objetivo ler o nome dos diretórios dos arquivos. Para testá-la, foi criado um vector que contém as strings com todos os nomes dos diretórios. O teste consiste em comparar cada string deste vector com o diretório do arquivo lido, e posteriormente, em conferir se alguma dessas strings correspondem ao diretório do arquivo lido.

```
TEST_CASE("nome_diretorios"){
    std::vector <std::string> diretorios;
    std::string aux;
    diretorios.push_back("alt.atheism");
    diretorios.push_back("comp.graphics");
    diretorios.push_back("comp.os.ms-windows.misc");
    diretorios.push_back("comp.sys.ibm.pc.hardware");
    diretorios.push_back("comp.sys.mac.hardware");
    diretorios.push_back("comp.windows.x");
    diretorios.push_back("misc.forsale");
    diretorios.push_back("rec.autos");
    diretorios.push_back("rec.motorcycles");
    diretorios.push_back("rec.sport.baseball");
    diretorios.push_back("rec.sport.hockey");
    diretorios.push_back("sci.crypt");
    diretorios.push_back("sci.electronics");
    diretorios.push_back("sci.med");
    diretorios.push_back("sci.space");
    diretorios.push_back("soc.religion.christian");
    diretorios.push_back("talk.politics.guns");
    diretorios.push_back("talk.politics.mideast");
    diretorios.push_back("talk.politics.misc");
    diretorios.push_back("talk.religion.misc");
    while(!NULL){
        for(int i = 0; i<diretorios.size(); i++){
            if (diretorios[i] == dp->d_name){
                aux = diretorios[i];
            }}}
    CHECK(aux == dp->d_name);
```

Bibliografia

https://pt.wikipedia.org/wiki/Motor_de_busca
http://www.linhadecodigo.com.br/artigo/1494/aprendendo-a-documentar-o-seu-codigo.aspx
https://homepages.dcc.ufmg.br/~glpappa/aeds2/exemplodoc.pdf
https://www.cplusplus.com
https://lemire.me/blog/2012/06/26/which-is-fastest-read-fread-ifstream-or-mmap/