

PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II

Trabalho Prático - Máquina de Busca

Priscila Pires de Carvalho Rocha

13 de junho de 2019

Sumário

1	Introdução	1
2	Implementação	1
2.1	Main.cpp	2
2.2	MaqBusca.h	2
2.3	MaqBusca.cpp	4
3	Testes	4
4	Conclusão	6

1 Introdução

O desenvolvimento desse trabalho faz parte da disciplina de Programação e Desenvolvimento de Software II e visa colocar em prática os conhecimentos adquiridos em sala de aula. No trabalho, foi desenvolvida uma máquina de buscas utilizando um índice invertido.

Dada uma quantidade ilimitada de documentos, o algoritmo deve ter a capacidade de realizar a leitura de todas as palavras existentes dentro desses documentos e criar um índice onde cada uma dessas palavras apresentará uma lista de documentos nos quais está contida. Assim, quando o usuário buscar uma palavra específica, ele consegue saber onde essa palavra é usada e qual seu grau de importância, ou seja, o quanto ela aparece no texto.

O trabalho foi desenvolvido em C++, no Windows, utilizando o Code::Blocks.

O trabalho está salvo no GitHub e pode ser acessado nesse link: https://github.com/watashipri/TP_2019_1

2 Implementação

A máquina de busca desenvolvida neste trabalho permite a indexação de arquivos e a consulta pelos termos nesse índice. O usuário começa inserindo os textos que deseja

utilizar nessa busca e, em cada arquivo adicionado, suas palavras são tratadas antes de serem incluídas no índice. Esse tratamento consiste em:

- Transformar todas as letras maiúsculas em minúsculas;
- Apagar todos os caracteres que não são letras ou números

Após esse tratamento, a palavra é incluída no índice invertido e, associada a ela, há uma lista de ocorrência da palavra entre os documentos.

```
apartamento: txt1.txt txt3.txt
casa: txt1.txt txt2.txt
em: txt2.txt
entrar: txt2.txt
esta: txt3.txt
ninguem: txt1.txt txt2.txt txt3.txt
no: txt3.txt
porem: txt1.txt
quem: txt1.txt txt2.txt txt3.txt
quer: txt1.txt txt2.txt
sairam: txt2.txt txt3.txt
tambem: txt1.txt
todos: txt2.txt txt3.txt
```

Figura 1: Índice Invertido

Após o índice ser construído, o usuário pode fazer consultas por palavras, essa consulta retorna se a palavra aparece ou não no índice e, caso tenha aparecido, em quais arquivos ela apareceu.

```
palavra buscada: Casa
Documentos: txt1.txt
```

Figura 2: Busca por palavra

Os testes feitos serão tratados em uma seção posterior.

2.1 Main.cpp

Esse é o arquivo que contém a inicialização do algoritmo. É através dele que o usuário realiza a interface com o programa. No main.cpp todas as funções implementadas são chamadas.

O usuário é perguntado sobre quantos textos deseja abrir e qual a palavra deseja buscar. Todas as outras operações realizadas não precisam de interação com o usuário.

2.2 MaqBusca.h

No arquivo do tipo *header* foi criada uma classe **MaqBusca**, com seus membros privados e públicos. Essa classe contempla todas as funcionalidades e abstrações necessárias para que a máquina de busca fosse implementada de acordo com as instruções passadas para realização do trabalho.

Ela possui os métodos públicos:

- *void Abrearquivo()*

- *void FecharArquivo()*

Utilizados para realizar a abertura e fechamento dos arquivos que serão utilizados.

- *string Tratamento(string palavra)*

Recebe uma string, pode ser uma entrada feita pelo usuário ou uma entrada do documento que será utilizado, retira os caracteres especiais, transforma letras maiúsculas para minúsculas e então retorna a palavra.

- *void IndiceInvertido()*

Recebe o documentos que deve se incluído no índice invertido. É feita uma análise palavra a palavra deste documento. Cada palavra recebe o tratamento feito com o método Tratamento() e é procurada no índice já existente. Se a palavra não existe no índice, ela é então adicionada junto com o a informação de a qual documento pertence. Caso a palavra já exista no índice, é verificado se o documento que está sendo analisado já foi incluído no vetor, em caso positivo, não ocorre nada, em caso negativo, o nome do documento é incluído no vetor. Também ocorre o armazenamento da frequência em que a palavra aparece nos documentos.

- *void ImprimirMap(map<string, int> m)*
- *void ImprimirMap(map<string, vector<string>> m)*

Estes dois métodos imprimem os maps que são utilizados no trabalho.

- *void Pesquisa(string palavra)*

Recebe e trata uma string, realiza a busca no índice invertido e retorna o resultado.

- *void Coordenadas(string palavra, int ndocs);*

Esse método calcula as coordenadas W de um vetor correspondente a um documento da base de dados. A equação utilizada foi passada no documento com informações do trabalho, na Seção 3.1.

- *double Ranking()*

Esse método retorna a similaridade entre os documentos utilizados, de acordo com a palavra que está sendo buscada pelo usuário.

As variáveis privadas são:

- *map<string, vector<string>> indiceinvertido*

O map foi criado de forma que as palavras são as chaves do map e os elementos, armazenados no vetor, armazenem todos os documentos em que as palavras aparecem.

- *map<string, int> contagempalavra*

Um outro map foi criado para que fosse armazenada a quantidade de vezes que cada palavra aparece nos documentos. Para facilitar o cálculo das coordenadas e do ranking.

2.3 MaqBusca.cpp

Esse arquivo contém a implementação de todas as funções presentes no arquivo *MaqBusca.h*. É nele onde o arquivo *MaqBusca.h* puxa o código de todas as suas funções. Não é necessário comentar uma ou mais funções contidas nesse arquivo, devido ao fato de que isso já foi feito durante a descrição do arquivo *MaqBusca.h*.

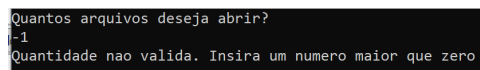
3 Testes

Os testes foram realizados durante o desenvolvimento do algoritmo. Devido a dificuldades em utilizar os testes de unidade, utilizando ferramentas como, por exemplo, o doctest, foram feitos apenas testes manuais no algoritmo.

Todos os testes realizados retornam o erro ou uma mensagem para o usuário através de uma mensagem de texto.

- Quantidade errada de arquivos

O número de documentos não pode ser menor do que 0 para que o programa possa funcionar.

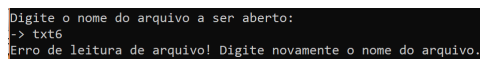


```
Quantos arquivos deseja abrir?  
-1  
Quantidade nao valida. Insira um numero maior que zero
```

Figura 3: Teste 1 1

- Nome do arquivo está errado

Caso o usuário digite errado o nome de um documento, ou tente adicionar um documento que não existe, o programa pede para que outro nome seja adicionado, garantindo que não ocorram erros de leitura ou um arquivo seja adicionado erroneamente.



```
Digite o nome do arquivo a ser aberto:  
-> txt6  
Erro de leitura de arquivo! Digite novamente o nome do arquivo.
```

Figura 4: Teste 2

- Vetor errado

Um erro encontrado durante o desenvolvimento e testes do algoritmo foi o armazenamento errado dentro do vetor que constitui o map do índice invertido. O vetor armazenava a mesma informação diversas vezes, dificultando o trabalho posterior com os dados.

```
apartamento: txt1.txt
txt1.txt
txt1.txt
txt1.txt
txt1.txt
txt1.txt
txt1.txt
txt1.txt
casa: txt1.txt
txt1.txt
em: txt2.txt
txt2.txt
entrar: txt2.txt
txt2.txt
txt2.txt
txt2.txt
txt2.txt
ninguem: txt1.txt
txt1.txt
```

Figura 5: Teste 3

- Palavra não encontrada

Um teste realizado várias vezes foi a busca de palavras que não existem no índice invertido.

```
Palavra Buscada: dragao
Palavra não encontrada
```

Figura 6: Teste 4

- Tratamento das palavras

Um teste realizado várias vezes foi o tratamento das palavras, esse teste foi feito tanto para as palavras inseridas pelo usuário quanto para as palavras que pertencem aos textos.

```
Palavra Buscada: GuArDa-CHuVAA
guardachuvaa
```

Figura 7: Teste 5

- Coordenadas

Foi realizado um teste para o cálculo das coordenadas e o ranking da palavra. Esse cálculo retorna o valor do logaritmo calculado, além da multiplicação vetorial da coordenada e o valor do ranking calculado.

```
QTD 4
Docs 1
log: 1.39
W: 5.55
Similaridade: 1.00
```

Figura 8: Teste 6

Com a utilização de testes podemos verificar o desempenho do código, tanto para comunicar ao usuário algum tipo de ocorrência indevida durante a execução ou para realizar o bloqueio ou interrupção de alguma parte do código durante a execução.

Ao desenvolver os testes durante a implementação, vários erros puderam ser evitados e muitas situações que não haviam sido previstas inicialmente foram incluídas.

4 Conclusão

Através da implementação desse projeto foi possível perceber que os conceitos adquiridos durante o decorrer da disciplina foram todos colocados em prática. O problema inicialmente se mostrou bem mais complexo do que o esperado, mas a medida em que o desenvolvimento foi sendo realizado e houve um maior entendimento de como as estruturas, como o map, funcionam, o processo diminuiu consideravelmente em dificuldade.

Foi importante manter uma organização no código, pois a medida que a complexidade foi aumentando, foi crucial que as informações estivessem organizadas para uma boa execução do projeto.

O projeto foi testado a medida em que foi sendo implementado, garantindo assim que o número de erros carregados até o final da implementação fosse reduzido consideravelmente.

Conclui-se que este trabalho foi uma ótima ferramenta para aplicar os conceitos adquiridos em sala de aula.