Programação e Desenvolvimento de Software II

Trabalho Prático

Junho de 2019

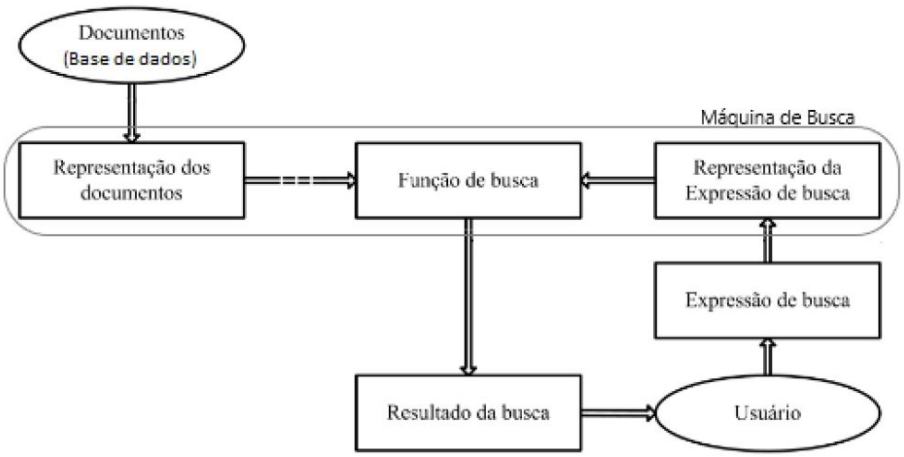
Bruno Oliveira — Matrícula: 2018070708

Fernando Couto — Matrícula: 2018014352

Guilherme Felberg — Matrícula: 2018115000

Introdução

O trabalho teve por objetivo o desenvolvimento de uma máquina de busca, capaz de ler arquivos de texto e a partir destes encontrar os melhores resultados possíveis de acordo com o desejo do usuário. O seguinte diagrama representa sua estrutura:



O programa funciona com a inserção de arquivos .txt pelo usuário, através de uma interface simples que pede o nome do arquivo que se deseja inserir, então as palavras são separadas uma por uma e adicionadas no índice invertido. Feito isso o usuário pode realmente utilizar a máquina de busca e pesquisar quaisquer palavras desejadas dentro dos arquivos.

Como compilar e executar

O trabalho foi desenvolvido nos sistemas operacionais Mac e Windows, utilizando tanto o Sublime quanto o Visual Studio, com o compilador g++.

Implementação:

- **Header:**

No header “buscador.h” foram declaradas quatro funções, uma função “armazena_arquivo( )” do tipo vector que recebe como parâmetro um ponteiro para uma variável do tipo inteiro e tem como finalidade pegar o nome dos arquivos de texto que o usuário deseja inserir na máquina. Uma função “separa( )” do tipo vector que recebe como parâmetros uma string e um char. A função “soma_palavras_arquivo( )”,do tipo map, que recebe como parâmetros uma variável map e uma variável vector, e serve para adicionar as palavras dos arquivos ao índice invertido. E por último uma função “ranking_co( )” do tipo map que recebe como parâmetros três variáveis do tipo map, três ponteiros do tipo float e duas do tipo vector. Essa função tem como objetivo fazer o ranqueamento das opções mais próximas à busca feita pelo usuário.

- **.CPP:**

A implementação das funções declaradas no header foi feita em um arquivo chamado “buscador.cpp”.

A “função armazena_arquivo( )” foi implementada utilizando uma entrada de usuário e armazenando os dados em um vector por meio do método push_back. O usuário pode inserir dados até marcar a opção que não deseja mais inserir arquivos.

A função “separa( )” foi implementada também utilizando um vector e armazenando informações nele por meio de um push_back.

A função “soma_palavras_arquivo( )” foi implementada utilizando um map para armazenar as informações lidas do arquivo.

- **Main:**

Na main, após a entrada de dados do usuário, é feita a leitura dos arquivos desejados.

Feita a leitura dos arquivos é criado um map de um map para guardar cada palavra do documento e em cada arquivo que essa palavra está, ou seja, se ela aparecer em mais de um documento isso é computado, além disso, é contado em cada documento quantas vezes essa palavra aparece. O map de map foi utilizado pois desse modo já é possível armazenar de uma vez o Tf.

```
62        //Criando um map de coordenadas para cada documento
63        map <string, map<string, float>  > doc_coordenate;
```

Após isso, foi criado um map de um map para armazenar cada coordenada de documento, diferente do mapa do índice invertido, no mapa das coordenadas do documento, a primeira chave é o próprio documento e a segunda chave é cada palavra, então são feitas as

contas dadas na descrição do trabalho e com isso é armazenado o valor das coordenadas de cada documento.

$$idf(t) = \log(\frac{N}{n_t})$$

$$W(d_j, P_t) = tf(d_j, P_t) \times idf(P_t)$$

Depois disso são criadas as coordenadas do documento para consulta, ou seja, as contas são feitas de acordo com a busca feita pelo usuário. Foi criado um índice invertido também para a consulta, o que poupou o trabalho de criar uma tabela hash, por exemplo. Nesse índice invertido primeiro são zerados todos os espaços do map e depois preenchidos corretamente, pois ficou mais fácil para armazenar todos de uma vez sem precisar zerar antes.

```
72        //Criando um map de coordenadas para a consulta.
73        map <string, float > doc_coordenate_query;
74        //Criando um índice invertido para a consulta.
75        map <string, int> inverted_index_query;
```

Também foi criado um map para fazer o Idf, já que se fosse feito junto com o Tf iria ficar mais difícil para implementar, além de ficar maior.

Dificuldades

De modo a otimizar o tempo e facilitar os testes, a forma como os arquivos são lidos no presente trabalho se deu de uma forma mais simplificada.

Por exemplo, se o usuário quiser analisar 1000 arquivos, uma nova forma mais robusta de leitura teria de ser implementada. Da maneira como está no trabalho, os arquivos têm que estar na mesma pasta que os códigos implementados.

O conhecimento também foi um fator decisivo, embora o trabalho tenha sido apresentado aos alunos há muito tempo, o conhecimento necessário para a execução do trabalho de uma forma mais simples e direta foi adquirido aos poucos durante as aulas da matéria.

Encontramos barreiras também para executar os testes, tentamos de várias maneiras e nos deparamos com diversas dificuldades em relação ao “doctest”.

Por último, o tempo limitou nossas decisões. Estamos cientes de que o código poderia ser otimizado e ser mais “inteligente”, no entanto o final de período dificultou o nosso desempenho.

Exemplo de compilação

```
C:\Users\bholi\OneDrive\Área de Trabalho\TP PDS>main
Digite o nome do arquivo com a extens├úo txt: teste1.txt

Arquivo adicionado com sucesso. Deseja continuar? (1)-Sim (2)-N├úo: 1
Digite o nome do arquivo com a extens├úo txt: teste2.txt

Arquivo adicionado com sucesso. Deseja continuar? (1)-Sim (2)-N├úo: 1
Digite o nome do arquivo com a extens├úo txt: teste3.txt

Arquivo adicionado com sucesso. Deseja continuar? (1)-Sim (2)-N├úo: 1
Digite o nome do arquivo com a extens├úo txt: teste4.txt

Arquivo adicionado com sucesso. Deseja continuar? (1)-Sim (2)-N├úo: 2
CARREGANDO ARQUIVOS E SEU CONTE├ÚDO.



------------------------------------------------------------



Digite a sua consulta (quais palavras voc├¬ deseja achar nos documentos): a b



De acordo com sua consulta e com os documentos fornecidos, o ranking coseno ficou assim:
teste1.txt-> 0.578353
teste2.txt-> 0
teste3.txt-> 0
teste4.txt-> 0.92361
```

Nesse caso, existiam 4 documentos:

D1= A A A B

D2= A A C

D3= A A

D4= B B

Conclusão

O projeto mostrou-se desafiador e trabalhoso, mas conseguimos resolver bem os problemas com algumas ideias inteligentes, como por exemplo a utilização de um map de um map, que diminuiu drasticamente o tamanho do código final, pois otimiza a forma de trabalho. Agradecemos ao professor e seus monitores pela ajuda, pelas aulas e pela boa didática.