TP Máquina de Busca – Documentação:

Victor Yuji Yano
José Eduardo Duarte Massucato
Rodrigo Reis de Sá Guimarães Cabral

Como usar o programa

O usuário insere como entrada palavras a serem pesquisadas nos documentos e digita “q“, ou ctrl + d, para encerrar essa consulta. O programa então retornará os nomes dos documentos que contenham todas as palavras que foram pesquisadas.

Observação: Caso as palavras a serem pesquisadas contenham acento como caracteres especiais, elas deverão ser escritas com acento pelo usuário. Isto se deve à implementação da normalização, que é por exclusão de caracteres com acento, em vez da substituição pela “forma normalizada”, o que foi autorizado pelo professor da disciplina.

Exemplo de funcionamento nos casos com acento:

Texto do arquivo: “olá” -> normalização -> Saída: “ol”

Entrada do usuário: “olá” -> normalização -> Saída: “ol” (correspondência)

Entrada do usuário: “ola” -> normalização -> Saída: “ola” (não correspondência)

Introdução

A implementação do código passou por várias etapas até sua forma definitiva. Inicialmente, o programa foi feito de forma não modularizada, tendo em vista agilizar a programação e facilitar a progressão lógica. Durante a criação deste protótipo, descobrimos várias bibliotecas que simplificaram o processo de implementação, como a *dirent.h* para coletar nomes de arquivos, o *algorithm* para a normalização e algumas funções do STL. O nosso maior desafio nessa etapa foi a criação de um Subsistema de Recuperação eficiente.

Após criado o protótipo, modularizamos o programa e o adaptamos para o paradigma da orientação ao objeto. Nessa etapa, a passagem foi facilitada pela lógica do código já estar completa, o que demandou somente uma conversão lógica. Por conseguinte, criamos o arquivo *makefile*, que auxiliou na execução do *software*.

Implementação

Documento “main.cpp”:

1) Recebendo a entrada:

- Começamos recebendo a entrada do usuário, sendo que o *input* é encerrado quando o usuário digita o caractere “q” ou aperta Ctrl+D.

- Essa entrada é passada, palavra por palavra, para um *vector<string>* chamado de “consulta”, que é normalizado.

- As palavras entradas pelo usuário são aplicadas como índice do “índice invertido” (chamada no programa de “IndexInvertido”).

2) Abrindo os arquivos de texto:

- Utilizamos a biblioteca *“dirent.h”* para receber os nomes dos arquivos na pasta “documentos”, a qual terá como caminho (*“./documentos/”*) .

- Cada nome de arquivo é recebido pelo struct “entry->name”, que, por sua vez, é inserido no vetor de strings “filename”.

- Em seguida fechamos a pasta.

3) Iterando com os documentos:

- Após receber os nomes dos arquivos, podemos abri-los uma a um, o que é feito em:

```
inFile.open("./documentos/"+*it, ios::in);
```

  Aqui, indicamos o caminho para a abertura do documento, então concatenamos a string: ./documentos/ + “nome do arquivo”, utilizando a opção de leitura (ios::in).

- Recebemos o conteúdo de cada arquivo e passamos ele para o vetor de string “palavras_temp”.

- Então, o vetor "palavras_temp" é normalizado, ordenado e “singularizado” (retiramos palavras duplicadas).

- Fazemos um laço que vai de 0 até “tam” (tamanho do vetor "palavras_temp") e inserimos em um *map* chamado “palavras”, o nome do arquivo como índice e um texto formado pelas palavras concatenadas de "palavras_temp".

- Em seguida, enviamos o nome do arquivo (“*it”), um texto (“palavras[*it]) e o vector das palavras consultadas(“consulta”) para a função comparador.

4) Imprimindo o resultado:

- Ao final do main, imprimimos o nome dos arquivos que foram adicionados a “IndexInvertido”.

Documento “busca.h”:

- Listamos os métodos que serão utilizados dentro de *public*.

- Dentro do *private*, utilizamos a variável “*map<string, set<string>>* IndexInvertido”. Esse map tem como índice as palavras consultadas e como valores o conjunto dos nomes dos arquivos que contém todas as palavras consultadas.

Documento “busca.cpp”:

1) Funções de retorno (“Valor()” e “Index()”):

- A função “Valor()” retorna o conjunto de documentos presente no índice invertido (“IndexInvertido”).

- A função “Index()” retorna uma cópia do próprio índice invertido (“IndexInvertido”).

2) Função de normalização (“normalizador”):

- Essa função recebe como parâmetro um vetor de string (tanto das palavras contidas nos arquivos quanto das palavras entradas).

- Entramos em um laço para percorrer cada uma das palavras contidas no vetor. Então, mandamos cada uma delas para a função “RemoveCaractereEspecial”, para retirarmos todos os caracteres que não pertencem ao alfabeto (A-Z ou a-z) e que não sejam números (0-9). Após isso, inserimos essas palavras para um vetor chamado “formatado”.

- Em seguida, entramos em um laço que percorre as palavras do vetor “formatado”, que, por meio das funções *for_each* e *tolower* da biblioteca *“algorithm”*, são convertidas à forma minúscula em toda sua extensão de caracteres.

- Esse vetor normalizado é retornado para onde a função normalizadora é chamada.

3) Função RemoveCaractereEspecial:

- Essa função recebe uma string, a qual, dentro do laço, é percorrida caractere por caractere.

- Caso um caractere seja especial, isto é, não pertença ao alfabeto (A-Z ou a-z), nem seja um número (0-9), então o retiramos.

- Por fim, o novo string formatado é reenviado à função que o chamou.

4) Função adicionar

- Essa função recebe uma string e a insere como índice do *map* “IndexInvertido”

5) Função comparador:

- Essa função recebe uma string com nome do arquivo (“nome”), o conteúdo do mesmo (“texto”) e um vetor contendo as palavras de entrada (“termo_buscado”).

- Dentro dessa função, ordenamos as palavras inseridas pelo usuário (“sort()”), as singularizamos (“unique()”) e contamos o números de palavras que o vetor contém (“size()”).

- Em seguida, entramos em um laço que vai de 0 até o tamanho da entrada formatada. Dentro desse laço, aplicamos a função “find()”, que retorna a posição da palavra pesquisada ou -1, caso a entrada não seja encontrada.

- Caso encontremos a palavra consultada no conteúdo do arquivo, essa palavra é inserida em um vetor chamado “achados” e incrementamos 1 em um contador “count” (o qual contará o número de palavras consultadas que cada arquivo tem).

- Caso o número de palavras consultadas singularizadas (todas as palavras de entrada diferentes entre si) seja igual ao “count”, isso significa que o documento contém todas as palavras consultadas. Portanto, inserimos o nome desse arquivo dentro do *set<string>* do “IndexInvetido”, cujo índice associado é a palavra correspondente.

6) Função Impressão:

- Imprimimos o nome de documentos que contenham todas as palavras consultadas pelo usuário, ou seja, tudo que foi adicionado no *set<string>* do “IndexInvertido” ao longo pro programa.

Conclusão

Encerrada a implementação do programa, a equipe passou para a montagem dos testes de unidade, o chamado “Busca_teste.cpp”, que avaliam o funcionamento do programa final. Assim, pudemos comprovar o desempenho de cada uma das funções, além de termos adicionado algumas novas funcionalidades, como a que retorna o valor do índice invertido e a que retorna uma cópia do índice invertido.

A submissão do código no Github facilitou o trabalho em equipe, pois a sincronização do programa pela plataforma manteve todos os membros sempre atualizados sobre as mudanças. Além disso, a organização da lógica prévia permitiu com que a coesão do trabalho mantivesse um fluxo constante.

Portanto, destacamos como pontos principais que enriqueceram este trabalho prático: a experiência do trabalho em equipe, o aprendizado sobre diversas bibliotecas e funções de manipulação de vetores, string e map, e o aprendizado a respeito do versionamento no Github.