Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

Documentação do Trabalho Prático

Vinícius Antunes de Souza, 2019068880

DCC 204 - Programação e Desenvolvimento de Software II

Ciência da Computação - Turma TN

Semestre 02/2022

Prof. Thiago Ferreira de Noronha

Belo Horizonte

27/11/2022

1. OBJETIVO

O presente documento possui como objetivo apresentar e discutir a execução do trabalho prático da matéria de Programação e Desenvolvimento de Software II, acerca da construção de uma **máquina de busca** na linguagem C++.

2. INTRODUÇÃO

Para o projeto de uma máquina de busca, muitos aspectos devem ser levados em conta, uma vez que ferramentas como Google, ou outras, realizam um trabalho minucioso que determina não apenas se aquele link retornado possui todas as palavras-chave de interesse, como também os organiza de maneira que o usuário possua os principais resultados, definidos por métricas como SEO, por exemplo.

Como forma de projeto, a criação de uma máquina de busca simplificada (que ordena os retornos apenas pela ordem lexicográfica) retira esse tipo de dificuldade, visto que a ordenação é a parte menos "mecânica" do processo, e mais complicada de ser realizada.

De forma geral, é possível, por meio de um projeto desse tipo, aprofundar-se nos conceitos de classe e métodos, uma vez que a orientação a objetos encaixa perfeitamente com o objetivo de separar resultados e manipulá-los, de forma a devolver ao usuário um formato específico. Além disso, a construção desse tipo de software é uma forma extremamente interessante de se aplicar o conceito de modularização, visto que cada tipo de manipulação da entrada até a saída ao usuário deve ser feita de forma independente e eficaz, trazendo ao código uma fácil leitura e baixa dificuldade de debugging, necessários em qualquer tipo de tarefa de construção de software, nos dias atuais.

3. IMPLEMENTAÇÃO

3.1 Idealização

Inicialmente, para a construção do software, imaginou-se o seguinte esquema estrutural de código.

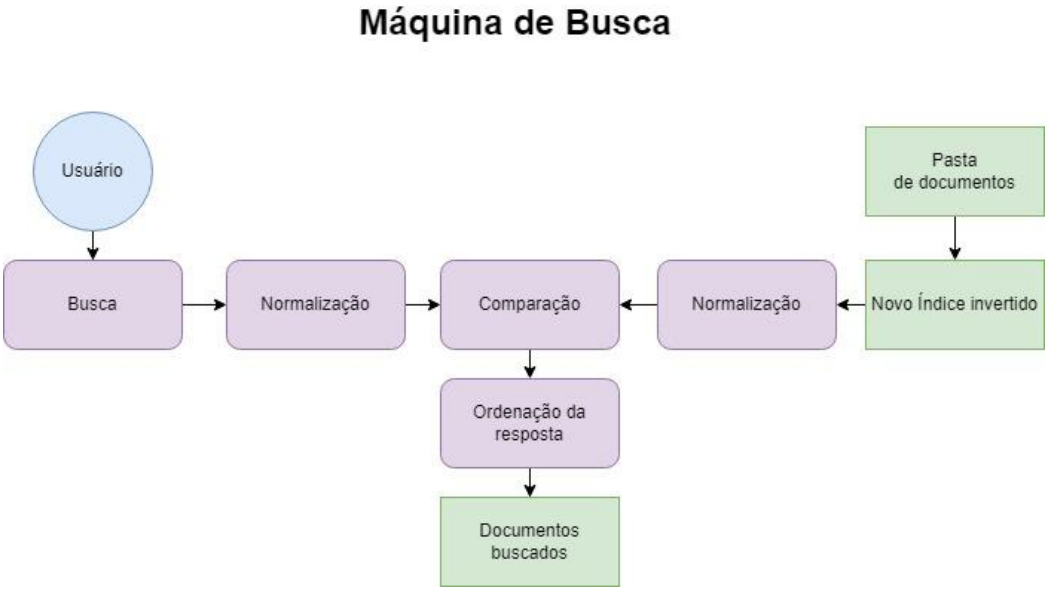

Figura 1 - Diagrama da idealização do programa

Assim que o programa é iniciado, simultaneamente o usuário digita sua pesquisa no terminal, enquanto o código acessa a pasta fixa de documentos, itera entre seus documentos, e cria o índice invertido com todas as palavras que eles possuem. Em seguida, tanto as palavras do índice, quanto as da busca, são normalizadas (retiradas as acentuações, pontuações e outros símbolos; passadas para formato minúsculo e mantidas apenas as letras de a-z), comparadas entre si para identificação dos documentos que as possuem, e então têm seus documentos ordenados em uma lista, sendo depois retornados ao terminal.

Nos próximos tópicos serão detalhados cada um desses processos, no formato em que foram construídos.

3.2 Interface

A interface do programa foi alocada no arquivo principal "main.cpp", com intuito de ser simples e curta, já que seu único objetivo é retirar a busca do usuário e "chamar" o método que recupera os documentos relacionados àquela busca.

3.2.1 Construção

Primeiro, o código da interface cria um objeto da classe **Maquina** (que será mais detalhada no tópico 3.3), com o argumento da pasta em que se encontram os documentos de busca, que neste caso é a pasta "./Documentos".

Para que o programa consiga armazenar todas as palavras da busca, é utilizada uma função **getline** da biblioteca string, e em seguida chamado um método da classe Maquina, que leva como parâmetro a string que foi alocada na variável da função getline, e retorna um set de strings com o nome dos documentos da busca.

Por fim, é realizado um método **for** para iterar pelos documentos, "printando" no terminal cada um deles por linha, como é o objetivo do programa.

3.2.2 Observações

- Como não foi especificada a maneira com a qual se desejava retornar os documentos ao usuário, esse formato de "print" foi escolhido por ser mais intuitivo e organizado (caso fosse retornada uma lista dos documentos na mesma linha, a visualização se tornaria confusa, além de se tornar difícil a checagem de que os documentos estão, realmente, em ordem lexicográfica, como requerido).

- Durante a construção da interface, o objetivo era manter apenas o essencial para o input do usuário, e a utilização dos métodos da classe Maquina. Entretanto, para que fosse possível realizar testes de unidade na maior quantidade possível de funções e métodos, um método (ou função) void, que apenas realiza "cout" dos documentos, não poderia ser testado, uma vez que não retorna nada. Dessa forma, foi tomada a decisão de incluir o código que realiza o "cout" dos documentos direto na interface.

3.3 Classe

Desenvolvida no arquivo "maquina.h", a classe Maquina possui apenas 1 método privado: o índice invertido, que possui tipo de um **map<string, set<string>>** e nome de **Maquina_.** Seu tipo foi definido desta maneira para que ele pudesse ter, relacionado a cada palavra que

existir nos documentos, uma lista desses documentos em que ela aparece. Além disso, o uso de set ainda auxilia na organização dos documentos, uma vez que ele já organiza seu conteúdo de forma lexicográfica.

Para a parte pública da classe foram construídos os seguintes métodos (que possuem um detalhamento maior do código utilizado no tópico seguinte):

1. *Maquina (string Path)* - Construtor do objeto **maquina_,** e responsável por montar o índice invertido do diretório "Path", com todos as palavras e os documentos em que aparecem.
2. *set<string> Buscar(string input)* - Chamado pela interface, com o parâmetro da string que contém as palavras de busca do usuário, e o objetivo de separar as palavras, normalizá-las e inseri-las em um set. Por fim, chamar o método *Selecionar* com o parâmetro do set montado, e a variável que soma o número de palavras no set.
3. *set<string> Selecionar(set<string> buscadas, int numero_de_palavras)* - Método dependente do anterior, que recebe o parâmetro do set das palavras buscadas e o número de palavras desse set. Sua função é comparar as palavras do set recebido com as do objeto da classe, e adicionar em um map<string, int> o nome dos documentos em que elas estão presentes, além do número de vezes que cada documento é encontrado ao longo das palavras. Depois, é iterado nesse map cada documento, e checado se o número de vezes em que cada um aparece é igual ao número de palavras buscadas, retornando um set com os documentos que possuem esse número igual.
4. *string Normalizar(string texto)* - Recebe por parâmetro ou uma palavra da busca, ou uma palavra do índice invertido, e normaliza ela, comparando-a com as letras possíveis do alfabeto, e substituindo as que possuírem algum tipo de diacrítico, ou por uma letra minúscula, caso maiúscula, retornando a palavra após a checagem.
5. *bool Letra(string str)* - Método auxiliar do anterior, que realiza a comparação da letra que é mandada como string e as letras maiúsculas ou minúsculas do alfabeto, retornando true ou false.

É interessante analisar algumas questões relacionadas ao modo como foram construídos os componentes da classe acima.

I) O único elemento criado como privado foi o map<string, set<string>>, visto sua alta utilização no processo de separar as palavras e os respectivos documentos para futura filtragem. Entretanto, outros objetos poderiam ter sido criados, uma vez que mais estruturas foram utilizadas ao longo dos métodos.

II) Um dos objetivos do programa, é o de identificar os documentos em que todas as palavras buscadas aparecem. O modo escolhido para realizar essa análise foi o de comparar o número de palavras diferentes buscadas com o número de vezes que um documento aparece ao buscar cada uma no índice. À primeira vista, esse modo é efetivo, uma vez que as estruturas de set e map retiram a chance de possuir algum tipo de duplicata nas palavras buscadas ou dos documentos. Entretanto, não há uma certeza de que essa escolha seja a prova de falhas, uma vez que apenas utiliza as condições dos tipos escolhidos e não realmente realiza a checagem de que todas as palavras estão em todos os documentos.

III) Por fim, um grande desafio está relacionado ao método de normalização de palavras, uma vez que, no caso da linguagem C++, não existe um método ou função clássica que realiza o processo de retirar os diacríticos das letras. E embora pareça uma tarefa simples, diversos

meios e bibliotecas utilizadas não funcionam em todos os casos, apenas retornando os caracteres especiais como formatos não reconhecidos.

3.4 Métodos

Para este tópico, vamos aprofundar um pouco mais sobre o código utilizado na construção de cada método da classe Máquina, e suas especificidades.

1. *Maquina (string Path)*

O método do construtor utiliza a biblioteca **filesystem**, útil na iteração de documentos que se encontram fora do diretório do programa, ou em uma pasta dentro dele.

É criado um iterador para o caminho Path, onde se encontram os documentos, e cada documento é aberto utilizando a função open, da biblioteca **fstream**. Ao abrir um documento, um laço de repetição é executado, em que cada palavra de cada documento é lida, e é adicionada junto com o nome do documento em um map<string, set<string>>. O nome dos documentos é manipulado antes de ser adicionado ao map, para que a substring que representa o nome da pasta seja retirado, restando apenas o nome do arquivo em si. Além disso, é utilizado o método de normalização em cada palavra, antes de ser incluído no map.

Por fim, os documentos são fechados.

2. *set<string> Buscar(string input)*

O método de busca começa inicializando um int que guarda o número de palavras buscadas, um set que guardará as palavras separadamente, e uma string auxiliar que vai ser usada para passar as palavras do input do usuário para o set.

O input do usuário (que é o parâmetro do método) é transformado em uma stream, pela função **stringstream**, e então começa um laço de repetição no qual, enquanto houver palavras nessa stream, elas serão passadas à string auxiliar inicializada, normalizadas, e então inseridas, uma a uma, no set. Além disso, dentro do laço está o int de número de palavras que soma 1 unidade a cada vez que o laço é repetido (ou seja, a cada palavra diferente que é inserida no set).

Por fim, o método retorna não um elemento em si, mas chama o método Selecionar, que retorna um set de documentos ao método Buscar, que por sua vez retorna esse set ao arquivo “main.cpp”, que chamou o método por primeiro.

3. *set<string> Selecionar(set<string> buscadas, int numero_de_palavras)*

Este método é composto de duas seções de iteradores.

Primeiro é criado um map<string, int> que guardará os documentos em que as palavras aparecem (mesmo aqueles que não possuem todas as buscadas), além do número de vezes que aquele documento é inserido, e um set que guardará apenas os documentos em que **todas** as palavras aparecem.

Em seguida, começa o primeiro iterador, que percorre o set de parâmetro do método, e checa quais são os documentos do índice invertido em que cada palavra do set aparece, inserindo-os no map, e somando uma unidade em sua chave cada vez que são inseridos.

Depois, para retirar os documentos que não possuem todas as palavras buscadas, um outro iterador percorre o map criado no passo anterior, e checa se o número de vezes que um documento foi inserido é igual ao número de palavras buscadas. Se for igual, significa que todas as palavras aparecem naquele documento, e assim ele é inserido no set de documentos final.

4. *string Normalizar(string texto)*

O método de normalização inicia criando uma string vazia de output e então, para cada letra da string do parâmetro, que é adicionada nessa string vazia, é chamado o método Letra, que checa se ela faz parte das 26 letras do alfabeto (maiúsculas ou minúsculas). Caso o método Letra retorne verdadeiro, a letra passa pela função **tolower** que deixa a letra como minúscula, e então é adicionada à string de output. Caso retorne falso, a letra é comparada com todas as possíveis letras com diacrítico do teclado (vogais e cedilha), e substituída por sua versão sem o diacrítico.

Por fim, a string de output é retornada.

5. *bool Letra(string str)*

O método Letra apenas utiliza um laço de repetição que checa se a letra do parâmetro (que é colocado como tipo string) é igual a alguma das 26 letras do alfabeto (maiúsculas ou minúsculas), dispostas em um vector de strings. Ao fim, ela retorna true se a letra for encontrada no vector, e false, caso contrário.

3.5 Testes de Unidade

Por fim, quanto aos testes de unidade, foram definidos códigos que testassem todos os métodos e funções possíveis. Em cada caso, foram criadas artificialmente as respostas esperadas nos métodos discutidos anteriormente, e um arquivo ".txt" de teste, e então utilizados os métodos para checar se eles retornariam a mesma resposta esperada.

Ao todo foram realizados 5 testes, referentes a 4 dos 5 métodos construídos na classe Máquina, sendo apenas não realizado um teste para o construtor da classe, uma vez que ele não retorna algum tipo, apenas define as características de Máquina_, tornando-se difícil de ser testado.

4. CONCLUSÃO

Foi realizado com êxito o projeto de construção de um software de máquina de busca. Todos os métodos construídos estiveram de acordo com a idealização inicial do modelo, e retornaram uma resposta condizente com o que era esperado a cada teste. De forma geral, o projeto não trouxe grande complexidade aos temas trabalhados em sala de aula, porém aplicou muito bem vários dos conceitos relacionados a uma linguagem de orientação a objetos, como é o caso da abstração e do encapsulamento.

Durante a construção, as maiores dificuldades estiveram relacionadas à normalização de strings para o formato requerido, e a organização dos métodos, de forma que realizassem o trabalho sem haver nenhum tipo de redundância desnecessária, ou que comprometesse a independência dos métodos em si.