Integrantes

Lucca Augusto, Matheus Marchesotti, Tiago Denicoli

Introdução

Os sistemas de buscas são ferramentas extremamente úteis quando consideramos a quantidade enorme de informação presente na internet, por isso são acessados diariamente por grande parte da população mundial. Além disso, a Google, uma das maiores empresas do mundo, se estabeleceu com a criação de um buscador, sendo hoje provedora de um uma quantidade enorme de serviços variados.

A ideia básica do sistema de buscas desse trabalho é retornar um subconjunto de documentos que correspondem a uma busca feita, sendo que o conjunto é ordenado para melhor responder os interesses do usuário. O código foi feito utilizando principalmente a linguagem C++, com a utilização de Python em apenas um script.

Para executar o programa, basta executar o comando make no terminal, estando na pasta principal, onde está o arquivo “Makefile”. Os documentos a serem utilizados como base para a busca devem ficar no diretório “documentos/”.

Implementação

Primeiramente, o programa cria uma lista do nome dos documentos presente na pasta “documentos”, a partir do script de python “cria_arquivo.py” e a armazena no arquivo “lista_de_documentos.txt”, para que o código em C++ consiga acessar.

Depois, é lida cada uma das linhas desse documento e tenta-se ler o documento relativo ao nome escrito na lista. Se a leitura for bem sucedida, as palavras presentes no documento são adicionadas a um vector de strings , apenas uma vez por palavra, após submetidas ao tratamento de retirar caracteres especiais e transformar letras maiúsculas. Outro vector, de pares, é responsável por armazenar o id de um documento e quantas vezes uma palavra aparece nele. Foi utilizado o vector nesse caso pelo fato de seus iteradores serem de acesso randômico, o que possibilita acessar índices equivalentes nos dois vectors e compará-los facilmente.

Índice	Vocabulário	Documentos que contém
id	palavra	(id do documento, quantas vezes aparece)
0	amor	(1, 3), (4, 3), (5, 1)
1	alegria	(2, 1), (4, 3)

Para cada documento, são calculadas as coordenadas de cada palavra e armazenadas em um vetor. Para armazenar isso na memória principal seria equivalente a uma matriz de tamanho “numero de documentos”x”numero de palavras”, o que ocuparia muito espaço. Por isso, cada um dos vetores de coordenada é registrado no arquivo “indice_invertido.txt”, sendo acessados apenas quando necessário.

Cada uma das buscas inseridas pelo usuário é considerada uma query que deve ser executada rapidamente, tendo em vista que o número total de buscas pode ser bastante elevado. Sendo assim, as coordenadas W(documento, palavra) que relacionam as palavras e os documentos são pré-computadas, de maneira que inserir documentos no índice invertido é um pouco demorado, porém as querys são rápidas.

Agora, com relação à busca, é lida a entrada como uma string e cada uma de suas palavras é armazenada em um vector de strings, após o mesmo tratamento submetido às palavras do documento. A partir desse vector, é criado um map que associa um inteiro a cada palavra, representando o número de vezes que essa palavra aparece na busca do usuário, informação necessária para o ranqueamento. O map se justifica porque é feita uma associação entre dois valores e cada palavra aparece uma única vez, podendo servir de chave.

Assim como para cada documento, as coordenadas de cada palavra são calculadas armazenadas em um vector , sendo que cada palavra tem o mesmo índice do vector utilizado para os documentos. Nesse caso é apenas um vetor e esse valor será usado várias vezes, para cada documento, portanto é interessante deixá-lo armazenado na memória principal. As palavras na consulta que não fazem parte de nenhum documento são descartadas, pois não interferem no algoritmo de ordenação dos resultados.

Para ranquear quais documentos possuem maior importância para a busca realizada, é feito o *cosine ranking*, o qual classifica os documentos a partir da proximidade do vector de coordenadas de cada documento em relação ao vector de coordenadas da consulta. Quanto mais próximo, mais bem ranqueado estará o documento. Por fim, esse ranking é exibido no terminal, e vale ressaltar que se dois documentos tiverem o mesmo ranking são exibidos em uma mesma posição.

Testes

A maneira encontrada para testar o programa foi calcular o índice invertido manualmente para um dataset pequeno, composto de apenas dois documentos apresentados nesta tabela:

Título do Documento	Texto do Documento
Lorem.txt	Documento teste!
Ipsum.txt	Teste docu-mento teste

Assim teremos um dataset cujo o vector relacao_ da classe dataset seria assim:

Índice da Palavra	Índice das suas ocorrências	Id do Documento	Quantidade de vezes que aparece neste documento
Vector (externo)	Vector (Interno)	Pair→first	Pair→second
0	0	1	1
0	1	2	1
1	0	1	1
1	1	2	2

E o vector palavras_ também da classe dataset estariam assim:

Índice da Palavra	String da palavra em si
0	documento
1	teste

Com isso em mente, sabemos os seguintes dados:

- O método Le_lista() itera sobre um documento auxiliar que contém os nomes de todos os arquivos utilizados e chama o método Le_documento() para cada um
- O método Le_documento() faz o dataset propriamente dito.
- O método Formata_palavra(string palavra) retorna uma string sem qualquer caractere especial, além de converter as letras maiúsculas em minúsculas.
- O método Contar_palavras() deve retornar 2.
- O método Numero_de_Documentos deve retornar 2.
- O método Quantos_documentos_possuem_essa_palavra(int idPx) deve retornar 2 para o valor 0, 2 para o valor 1, e 0 para qualquer outro valor.

- O método Quantas_vezes_dj_possui_Px(int dj, int idPx) deve retornar os seguintes valores:

int dj	int idPx	retorno
1	0	1
1	1	1
2	0	1
2	1	2
Valor que não está na tabela	Valor que não está na tabela	0

E assim podemos efetuar o teste da classe dataset e utilizar tais valores para as classes dependentes dela.

Conclusão

Esse projeto se apresentou como um desafio para todos os integrantes do grupo e, justamente por isso, serviu de grande aprendizado. Após as aulas, qualquer impressão de que já estávamos sabendo o conteúdo apresentado e como utilizar as ferramentas, foi derrubada rapidamente, logo quando surgiram os primeiros problemas de sintaxe, seguidos pelos problemas relacionados ao software Github e de depuração, mostrando que tínhamos muito mais dúvidas do que respostas.

Após passar por todos os desafios apresentados, é gratificante observar que aprendemos como resolver diversos problemas que provavelmente aparecerão novamente durante nossa jornada na ciência da computação, além de ter fixado bem o conteúdo de testes, POO, STL, modularização e etc. Além disso, foram desenvolvidas outras habilidades imprescindíveis para um profissional, principalmente o trabalho em equipe e a capacidade de pensar em como resolver bugs e onde procurar ajuda, seja em algum fórum ou outro colega. Particularmente, foi interessante aprender que todas pessoas podem acrescentar alguma coisa, sendo ela um conhecimento específico, um atalho que facilita programar ou um ponto de vista diferente sobre o código e a lógica.

Link no Github:

https://github.com/LuccaAug/GoogleHumilde/