Trabalho Prático 3
Máquina de Busca Avançada

Vinicius Trindade Dias Abel
Matrícula: 2020007112

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

viniciustda@ufmg.br

1. Introdução

O problema proposto foi desenvolver um indexador e um processador de consultas de uma máquina de busca avançada. O indexador cria um índice invertido, onde cada palavra recebe um hash que define sua posição no índice, e cada uma delas também tem uma lista com os documentos em que ela aparece e a frequência em que ela aparece em cada um desses documentos. E o processador de consulta ordena os documentos de acordo com sua similaridade com a consulta, que é calculada utilizando o Modelo Espaço Vetorial (MEV).

Para isso, o algoritmo recebe como entrada uma pasta contendo os documentos em que a consulta será realizada, um arquivo que contém a consulta que deve ser feita, um documento onde será impresso os dez documentos com maior similaridade à consulta e um arquivo contendo as stopwords, que são palavras que não devem ser consideradas durante todo o processo, desde a criação do índice invertido até a consulta.

2. Método

2.1. Estrutura de Dados

A implementação do programa teve como base da estrutura de dados um vetor que utiliza uma função hash para alocar itens da classe Termo, onde cada termo possui uma lista encadeada.

Para esta implementação foram criadas classes (detalhadas no próximo tópico), sendo elas: Termo, Lista e ListaEncadeada, Frequencia, Similaridade, Fila e Filaencadeada, e Arquivo. Cada classe foi criada em um arquivo .hpp (e implementadas em arquivo .cpp de mesmo nome) separado das outras.

Além destes, foram criados um arquivo .hpp e um .cpp de nome arq_funcao que possui funções que trabalham com arquivos, um arquivo .hpp e um .cpp de nome indice que possui a função que cria o índice invertido, um arquivo .hpp e um .cpp de nome hash que implementa a função que calcula o

hash de cada termo, um arquivo .hpp e um .cpp de nome consulta que realiza a consulta com base no índice invertido criado previamente.

2.2. Classes

Termo

A classe Termo possui uma string palavra que armazena o termo, e uma lista encadeada frequencia que armazena o id do documento com a frequência em que a palavra aparece no documento.

Ela também possui as funções GetTermo que define qual é o termo armazenado na string palavra, SetTermo que retorna a string palavra, GetFrequencia que retorna a frequência que a palavra aparece em determinado documento, GetIDPrimeiro que retorna o primeiro documento da lista encadeada, GetProxID que retorna o próximo documento da lista encadeada e IncrementaFrequencia que incrementa a frequência da palavra em determinado documento.

Lista e ListaEncadeada

Estas classes, onde ListaEncadeada é uma classe filha de Lista, foram vistas em aula. Entretanto sofreram algumas alterações para se adequarem a solução do problema.

A primeira alteração é que todos os elementos da lista são do tipo Frequencia.

Outra alteração foi a substituição de algumas funções para que fosse possível realizar operações com a classe Frequencia. Para isso, foram criadas as funções GetFrequencia que retorna a frequência que a palavra aparece em determinado documento, GetIDPrimeiro que retorna o primeiro documento da lista encadeada, GetProxID que retorna o próximo documento da lista encadeada, IncrementaFrequencia que incrementa a frequência da palavra em determinado documento e PesquisaID que procura determinado documento na lista.

Além disso, funções que não foram utilizadas durante a implementação do TP foram retiradas, como as variações da função Insere e as funções de remoção.

Frequencia

A classe Frequencia possui uma variável ID do tipo int, que é onde fica armazenado o id de um arquivo em que a palavra aparece.

Outra variável da classe é a variável freq do tipo int, que é onde fica armazenado a frequencia em que a palavra aparece no arquivo armazenado em ID.

Também possui variável prox do tipo Frequencia que é um ponteiro que aponta para o próximo da lista.

Além disso, a classe possui apenas as funções construtor e destrutor.

Similaridade

A classe Similaridade possui uma variável conteudo do tipo string e uma variável id do tipo int.

Além disso, a classe possui apenas as funções construtor e destrutor.

Esta classe é usada em diferentes momentos, então o conteúdo não é sempre o mesmo.

Fila e FilaEncadeada

Estas classes, onde FilaEncadeada é uma classe filha de Fila, foram vistas em aula. Entretanto sofreram algumas alterações para se adequarem a solução do problema.

A primeira alteração é que todos os elementos da lista são do tipo Arquivo.

Outra alteração foi que apenas as funções Vazia, GetTamanho, Enfileira, Desenfileira e limpa foram implementadas (além do construtor e destrutor).

Arquivo

A classe Arquivo possui uma variável conteudo do tipo string, que é onde fica armazenado uma linha do arquivo que está sendo trabalhado.

Também possui variável prox do tipo Arquivo que é um ponteiro que aponta para o próximo da fila.

Além disso, a classe possui apenas as funções construtor e destrutor.

2.3. Main

A função main lê os parâmetros passados pela linha de comando, conta o número de arquivos da pasta corpus, e o numero de palavras do vocabulário ao mesmo tempo em que prepara os documentos (retira as stopwords, deixa todas as palavras em caixa baixa, substitui a pontuação por espaço e retira os termos com números) para a criação do índice e a realização da consulta. Em seguida, inicializa um índice e chama as funções CriaIndice e Consulta.

2.4. Configuração para teste

– Sistema Operacional do seu computador: Windows 11;
– Linguagem de programação implementada: C++;
– Compilador utilizado: G++ da GNU Compiler Collection;
– Dados do seu processador: i7;
– Quantidade de memória primaria: 16GB;
– Quantidade de memória secundaria: 512GB.

3. Análise de Complexidade

função ContaArquivos - complexidade de tempo: esta função possui um while que é executado n vezes, sendo n o número de arquivos contidos na pasta. Este while apenas incrementa a variável num_arq que ao final do while

retorna o número de arquivos da pasta. Dessa forma, a complexidade assintótica de tempo desta função é Θ(n).

função ContaArquivos - complexidade de espaço: esta função armazena apenas a variável num_arq. Assim, a complexidade assintótica de espaço desta função é Θ(1).

função PreparaCorpus - complexidade de tempo: esta função possui três laços separados de tamanho n. Os dois primeiros têm complexidade O(n). E o terceiro laço possui outro laço dentro dele, que por sua vez possui outros cinco laços separados dentro dele, além disso, três destes cinco também possuem mais um laço dentro, fazendo com que o while que contem todos estes laços tenha complexidade $O(n^4)$ Dessa forma, a complexidade assintótica de tempo da função é $\Theta(n^4)$, fazendo desta a função mais demorada do TP.

OBSERVAÇÃO IMPORTANTE: Tentei implementar de outras formas para diminuir o tempo de execução do programa, mas não consegui. Com esta função, o TP gasta cerca de 15s para uma pasta com aproximadamente 200 arquivos e cerca de 30 MINUTOS para uma pasta com aproximadamente 18600 arquivos. Com isso, foi criada uma variável que conta quantos arquivos já foram tratados pela função e a cada 200 arquivos é impresso o número de arquivos tratados, para que seja possível saber que o programa ainda está rodando.

função PreparaCorpus - complexidade de espaço: esta função realiza operações com seis filas, que possui tamanho n. Assim, a complexidade assintótica de espaço desta função é Θ(n).

função Hash - complexidade de tempo: esta função possui um for que é executado n vezes, sendo n o tamanho da string passada como parâmetro. Dessa forma, a complexidade assintótica de tempo desta função é Θ(n).

função Hash - complexidade de espaço: esta função armazena apenas três variáveis do tipo int. Assim, a complexidade assintótica de espaço desta função é Θ(1).

função CriaIndice - complexidade de tempo: a complexidade assintótica de tempo da função também é $\Theta(n^4)$, uma vez que possui quatro laços um dentro do outro. Apesar disto, ela não contribui tanto para a demora na execução do programa quanto a função PreparaCorpus.

função CriaIndice - complexidade de espaço: a complexidade de espaço desta função é $\Theta(n^2)$, já que ela trabalha com um vetor em que cada elemento possui uma lista encadeada, este é o indice.

função Consulta - complexidade de tempo: a complexidade assintótica de tempo da função também é $\Theta(n^3)$. Esta função possui diversos laços e o maior deles mais dois laços um dentro do outro.

função Consulta - complexidade de espaço: esta função trabalha com alguns vetores e uma matriz, sendo a matriz mais significativa para a análise de espaço. Então a complexidade de espaço é $\Theta(n^2)$.

Geral ou função main - complexidade de tempo: A função main por chamar as funções descritas anteriormente (menos a função Hash), assume a complexidade assintótica de tempo $\Theta(n^4)$, por ser a maior entre as funções.

Geral ou função main - complexidade de espaço: A função main por chamar as funções descritas anteriormente (menos a função Hash), assume a complexidade assintótica de espaço $\Theta(n^2)$, por ser a maior entre as funções.

4. Estratégias de Robustez

As classes foram implementadas de forma que apenas as funções são públicas, para evitar o acesso indevido aos atributos.

Se ocorrer algum erro na abertura de arquivo, o erro é indicado.

As etapas (criação do indice invertido e realização da consulta) do tp não foram implementadas diretamente no main para evitar que o código de cada etapa seja alterado e consequentemente não cumpra sua função. Além de facilitar o entendimento do código e deixar bem definida cada etapa.

Foram criadas diversas classes e bibliotecas, que facilitam o entendimento do código e permite que sejam usadas em futuros trabalhos. Como foi o caso da FilaEncadeada, usada no tp1.

O programa imprime as opções de entrada caso ela não seja atendida corretamente.

O programa utiliza da alocação dinâmica em partes que poderiam causar erro na hora de alocar a variável.

5. Análise Experimental

Desempenho computacional: Está ligado ao número de arquivos que existem na pasta corpus, quanto mais arquivos, maior o tempo de execução. Como foi dito na analise de complexidade de tempo da função PreparaCorpus.

Acesso à memória: O programa foi implementado de forma que o tamanho do indice é definido pelo número de arquivos e tamanho do vocabulário (vocabulário são todas as palavras contidas nos arquivos, desconsiderando as repetições). Dependendo do número e do tamanho dos arquivos da pasta corpus, a memória utilizada altera. Quanto mais arquivos e maior o vocabulário, mais memória será usada.

6. Conclusões

Este trabalho lidou com um indexador e um processador de consultas de uma máquina de busca avançada. Na qual a abordagem utilizada para sua resolução foi a utilização de um índice invertido, onde cada palavra recebe um hash que define sua posição no índice, e cada uma delas também tem uma lista com os documentos em que ela aparece e a frequência em que ela aparece em cada um desses documentos. E o processador de consulta que ordena os documentos de acordo com sua similaridade com a consulta, para isso, é calculada a similaridade utilizando o Modelo Espaço Vetorial (MEV).

Por meio da resolução desse trabalho, foi possível praticar os conceitos relacionados a hash, filas e listas, manipulação de arquivos e pastas passados como parâmetro, a criação de estruturas de dados, classes e funções, a trabalhar com algoritmo de ordenação e a implementar formulas.

Durante a implementação da solução para o problema, houveram importantes desafios a serem superados, por exemplo a manipulação do arquivo .txt, o modo de acessar estas arquivos dentro da pasta, trabalhar com os parâmetros passados por linha de comando, entender o funcionamento do algoritmo de ordenação à ponto de conseguir modificá-los para atender a necessidade do problema, entender o MEV para conseguir implementa-lo corretamente, criar classes e funções para resolver problemas que apareciam, utilizar alocação dinâmica para que não ocorresse erro na hora da alocação, e a implementação do makefile. Por fim, a redução do tempo de execução, sendo que este último, infelizmente não consegui resolver da melhor forma.

7. Bibliografia

- String Hashing. Disponível em: https://cp-algorithms.com/string/string-hashing.html#improve-no-collision-probability. Acesso em: 15 de fevereiro de 2022.

- Tabela ASCII. Disponível em: https://web.fe.up.pt/~ee96100/projecto/Tabela%20ascii.htm. Acesso em: 15 de fevereiro de 2022.

- cplusplus reference. Disponível em: https://www.cplusplus.com/reference. Acesso em: 15 de fevereiro de 2022.

- Usando As Funções getopt() e getopt_long() Em C. Disponível em: https://daemoniolabs.wordpress.com/2011/10/07/usando-com-as-funcoes-getopt-e-getopt_long-em-c/. Acesso em: 15 de fevereiro de 2022.

- função getopt() em C para analisar argumentos de linha de comando. Disponível em: https://www.geeksforgeeks.org/getopt-function-in-c-to-parse-command-line-arguments/. Acesso em: 15 de fevereiro de 2022.

- Alocação dinâmica de matriz em C++. Disponível em: https://www.clubedohardware.com.br/forums/topic/1229551-aloca%C3%A7%C3%A3o-din%C3%A2mica-de-matriz-em-c/. Acesso em: 15 de fevereiro de 2022.

- Slides do professor.

8. Instruções para compilação e execução

• Acesse o diretório TP
• Utilizando o terminal, compile o TP3 utilizando o comando: make
• Com esse comando, será gerado o arquivo buscar.exe no diretório bin e os arquivos *.o no diretório obj
• Acesse o diretório bin
• Utilizando o terminal, execute o arquivo buscar.exe junto com o arquivo de consulta.txt, o arquivo de ranking.txt, a pasta_corpus e o arquivo stopwords.txt, utilizando -i antes do arquivo consulta.txt, -o antes do arquivo ranking.txt, -c antes da pasta_corpus e -s antes do arquivo stopwords.txt. Exemplo: buscar -i consulta1.txt -o ranking1.txt -c corpus -s stopwords.txt