Introdução

Este documento é referente ao Trabalho Final de Programação e Desenvolvimento de Software II, que foi proposto pelos professores Thiago Ferreira de Noronha e Lucas Victor Silva Pereira e realizado, em grupo, pelos alunos: André L. A. Rezende - 2018104050, Arthur de Lapertosa Lisboa - 2018115469 e Thales Pereira Tenebra 2018072080.

O projeto consiste em uma Máquina de Busca, que nada mais é que um programa capaz de recuperar informações em uma base de documentos por meio de um parâmetro de busca apresentado pelo usuário como representante de sua necessidade e retornando, a este, um resultado que cubra as necessidades deste usuário.

Implementação:

Código fonte GitHub:
https://github.com/arthurlapertosa/tpPDS2

Leitura de arquivos:

Introdução:

A classe “LeituraArquivos”, contida no arquivo “LeituraArquivos.cpp”, é a classe responsável por fazer a leitura de todos os arquivos “.txt”. A partir dessa leitura, ela também é encarregada de chamar todas as outras classes, que serão apresentadas ao longo deste relatório, com exceção da classe Busca, que modelam toda a estrutura de dados que usamos para armazenar as palavras e as informações relacionadas à elas.

Atributos da classe:

- **int num_doc_:**
  É a variável responsável por conter a quantidade de documentos que foi lido pelo programa e que por fim representará a quantidade total de documentos da coleção.

- **indice_invertido indice_:**
  É a variável do tipo indice_invertido, classe que será melhor explicado posteriormente neste documento, responsável pelo armazenamento do índice invertido do trabalho.

- **frequencia_palavra frequencia_:**
  É a variável da classe frequencia_palavra, também a ser apresentada mais à frente no relatório, responsável por conter a frequência [TF(term frequency)] de cada palavra em cada documento lido pelo programa.

- **frequencia_invertida frequencia_invertida_:**

É a variável da classe frequencia_invertida, que logo será apresentada mais aprofundadamente neste relatório, responsável por conter a frequência inversa (IDF - *inverse document frequency*) de cada palavra que está contida na coleção.

- **wvector wvector_:**

É a variável do tipo wvector - que será apresentada mais para frente no relatório - responsável pelo armazenamento de maps que contenham a palavra e seu valor, além de tratamento de vetores. Melhorar isso

- **vector<string> palavras_:**

Um vector (classe da biblioteca padrão do C++), que nada mais é que um vetor alocado dinamicamente, que contém todas as palavras dos documentos.

Métodos da Classe:

- **LeituraArquivos():**

O construtor da classe. Optamos por deixá-lo apenas inicializando a variável do num_doc_: que recebe o valor de 0.

- **void ler():**

É nesse método que acontece toda a leitura dos documentos ".txt" contidos mesma pasta do projeto. A lógica de leitura dos arquivos é a seguinte: somente os arquivos do tipo "d1.txt", "d2.txt", "dn.txt" n = 1, 2, 3, 4 … são lidos pelo programa.
Inicialmente há a tentativa de ler o arquivo do tipo d1.txt. Se não conseguir ser lido, o programa acusará erro de leitura. Em seguida, seguindo o laço do for, são lidos todos os documentos do mesmo formato apresentado acima.

Assim que o programa detecta que acabou os documentos da pasta, a leitura de arquivos é interrompida. Funciona da seguinte maneira: se arquivo "dn.txt" não pôde ser aberto e n > 1, então um comando break é realizado, que interrompe o laço do "for" e a leitura dos arquivos é finalizada.

Para cada arquivo aberto, é feito um laço while, que percorre todas as palavras do documento e vai chamando os métodos das classes índice invertido, frequência das palavras (TF), e o vector palavras_. Nos dois primeiros, é passado como parâmetro para a função, a palavra que está sendo lida no momento, e o documento no qual essa palavra pertence. Toda a manipulação com essas palavras e será explicada mais à frente no relatório.

Após a iteração do for em todos os documentos, é chamado o método inserir através da variável frequencia_invertida_ para que seja montada a frequência invertida. Este método recebe 2 parâmetros, o primeiro um map<string, map<string, int>> referente a frequência da palavra e o segundo um int referente ao número total de documentos da coleção e, como ainda será melhor abordado, este método calcula e adiciona o elemento ao mapa frequencia_total_palavra_.

Por fim, é chamado o método de transformar o vetor palavras_, que pode conter palavras repetidas em um vetor em que não existe repetição de palavras. Após ele chama o método ler clone para continuar o processo.

- **string verifica(string a):**

Verifica cada caractere da string para encontrar os que não forem entre a e z ou ç, cada vez que encontrar, ele troca o resto da string pelo caractere seguinte(com exceção do último) além de apagar o último.

- **string minusculo(string a):**
  Transforma todos os caracteres da string em minúsculo.

- **void imprimirIndice():**
  Chama um método dentro da classe “indice_invertido” que imprime na tela o índice invertido dos documentos.

- **void imprimirFrequenciaPalavras():**
  Chama um método dentro da classe “frequencia_palavra” (TF) que imprime na tela a frequência das palavras nos documentos dos documentos.

- **int tf(string documento, string palavra):**
  Chama um método dentro da classe “frequencia_palavra” (TF) que retorna o TF(Dj,Pt), que é a frequência da palavra Pj do documento Pt.

- **void imprimirFrequenciaInvertidaPalavras():**
  Chama o método imprimi_Invertida dentro da classe frequencia_invertida para imprimir na tela todas as palavras e suas respectivas frequências invertidas.

- **double idf(string palavra):**
  Chama o método frequencia_invertida_palavra(palavra) dentro da classe frequencia_invertida para retornar o valor double da frequência invertida da palavra que foi recebida como parâmetro e retorna este mesmo valor.

- **indice_invertido indiceInvertido():**
  Função que retorna o índice invertido dos documentos.

- **int numeroDocs():**
  Função que retorna o número total de documentos da coleção.

- **int numero_Doc_Palavra(string palavra):**
  Retorna o número de documentos que a palavra “palavra” aparece.

- **void lerclone(vector<string> palavras):**
  Refaz a função de percorrer os documentos, porém, já com o vetor de palavras completo e sem repetições, necessário para que quando passar pelos documentos ele crie um wmap(objeto que será detalhado mais à frente) para cada um documento, associando cada palavra do vetor palavras com o IDF* TF referente à ela correspondente ao documento em específico, e após, inserir cada qual wmap auxiliar no wvector_.

- **vector<string> palavras_return():**
  Retorna o atributo palavras_, que contém todas as palavras de todos os documentos.

- **void imprimir_w():**
  Imprime o objeto wvector_, que é um vetor de maps.

- **wvector retornar_w_vector()**:
  Retorna o atributo wvector_.

Conclusão:

Essa classe funciona basicamente da seguinte maneira: o método ler() é o método encarregado de chamar todas as funções necessárias para a leitura de todos os arquivos e a estruturação de todas as estruturas de dados necessárias para o ranqueamento dos documentos. Ela chama todos os métodos para que seja construída o TF (dj ,Pt), que é a frequência da palavra Pt no documento dj, o IDF(Pt), que é a importância de Pt na coleção, e posteriormente constrói o vetor de W's, onde W(dj ,Pt) é a coordenada do documento dj no eixo Pt.

Índice invertido:

Introdução:

Classe encarregada de construir todo o índice invertido dos documentos. Está contida no arquivo "indice_invertido.h", com o nome de "Class indice_invertido" O índice invertido é uma função Hash perfeita, que relaciona cada palavra da coleção com os documentos em que ela aparece.

Atributos da classe:

- **map<string, set<string>> my_map_:**

É uma estrutura de dados do tipo dicionário, encarregada de conter o índice invertido dos documentos. As chaves dos elementos desse dicionário são as palavras da coleção. Os valores desse dicionário são sets - outra estrutura de dados implementada como árvores binárias em C++ - com todos os documentos em que a palavra (chave) aparece.

Métodos da Classe:

- **void inserir(string a, string documento):**

Método responsável por adicionar elementos no atributo my_map_ dessa classe. Ela recebe duas variáveis como parâmetros. A variável "a" é a palavra a ser adicionada, e a variável "documento" é o nome do documento em que a palavra "a" aparece.

Para realizar essa inclusão no dicionário, precisamos inicialmente saber que um dicionário só aceita chaves únicas: ou seja, se tentarmos incluir uma palavra já existente como chave de algum elemento do dicionário, ele não poderá ser incluído, pois a chave é única. Para resolver esse problema, toda vez que tentamos incluir uma nova palavra no dicionário, testamos se ela foi efetivamente incluída, por meio do iterator bool "ret", que tem exatamente esse papel. Se o iterator dizer que a palavra foi incluída com sucesso, não fazemos mais nada. Caso contrário, se o iterator indicar que o elemento não foi criado, isso significa que ele já existia anteriormente no dicionário. Se isso ocorrer, simplesmente adicionamos a palavra nova ao set de strings relacionada à palavra "a".

- **void imprimirIndice():**

Imprime o índice invertido na tela.

Conclusão:

A classe tem função de construir o índice invertido dos documentos. Para fazer isso, basicamente é só ir chamando a função "void inserir(string a, string documento)". Todo o índice invertido é então criado no atributo my_map_ da classe.

Frequência das palavras nos documentos:

Introdução:

É a classe que relaciona cada palavra com seus respectivos documentos e a quantidade de vezes que repetiu naquele documento. Está contida no arquivo “frequencia_palavra.h” com o nome de “Class frequencia_palavra”. Essa classe é a responsável por criar o índice TF(dj,Pt), que é a frequência da palavra Pt no documento dj. Esse índice é de fundamental importância para montarmos o vetor W para realizarmos o Ranking Cosseno nos documentos juntamente com a pesquisa do usuário.

Atributos da classe:

- **map<string, map<string, int>> frequencia_palavra_;**

Relaciona as palavras com seus respectivos documentos e a quantidade de vezes que repetiu naquele documento - a estrutura de dados que contém o TF(dj,Pt).

Métodos da Classe:

- **void inserir(string palavra, string documento)**
  Insere novas palavras, associando-as com seus respectivos documentos.

- **void imprimir()**
  Imprime na tela o map frequencia_palavra.

- **map<string, map<string, int>> frequenciaPalavra():**
  Retorna o atributo frequencia_palavra_.

- **map<string, int> frequenciaPalavra(string palavra)**
  Retorna o map com os documentos que a palavra "palavra" aparece.

- **int frequenciaPalavraNoDocumento(string documento, string palavra)**
  Retorna o TF(dj,Pt) propriamente dito. Recebe como parâmetro o documento desejado e a palavra, e retorna um int, que é a frequência daquela palavra no documento.

Conclusão:

Essa classe é fundamental para a montagem da frequência invertida das palavras (o IDF), por conta da sua propriedade de armazenar quantas vezes uma palavra repetiu em um documento específico, além disso, é essencial para o cálculo do vetor w, já que ela representa o TF em si, que quando multiplicado pelo IDF, se torna o w, que será explicitado com detalhes mais à frente na documentação.

Frequência Invertida das palavras:

Introdução:

Esta classe, implementada no arquivo frequencia_invertida.cpp, tem o intuito de calcular, criar e relacionar as palavras a suas respectivas frequências invertidas, valor que será utilizado para cálculos posteriores.

Temos que IDF(*inverse document frequency*, ou como chamamos, frequência invertida da palavra) expressa a importância de uma palavra dentro da coleção de documentos (o quão significante é o fato da palavra ocorrer em um documento qualquer) e é calculada por: IDF(t) = log((Número Total De Documentos) /(número documentos onde a palavra ocorreu)).

Atributos da classe:

- **map<string, double> frequencia_invertida_palavra_;**
  Variável que relaciona cada palavra ao seu IDF.

Métodos da Classe:

- **void inserir(map<string, map<string, int>> frequencia_palavra_, int num_docs_total);**
  Método responsável por calcular o IDFde cada palavra e relaciona-los na variável frequencia_invertida_palavra_;

- **void imprimi_Invertida();**
  Método que imprime na tela todas as palavras e seus respectivos IDFs.
- **double frequencia_invertida_palavra(string palavra);**
  Retorna a frequência invertida da palavra.

Conclusão:

Esta classe será essencial para o cálculo das coordenadas do vetor relativo a cada palavra em cada documento na classe WMAP e por consequência para o ranqueamento dos documentos, já que este é um dos parâmetros essenciais para tal.

CLASSE WMAP:

Introdução:

A classe wmap tem a função de associar em um atributo do tipo Map, cada palavra existente ao seu w(IDF*TF) referente a um documento específico. Será, na prática, um vetor de coordenadas W de cada um dos documentos. Será posteriormente utilizada para realização do cálculo do Cosine Ranking.

Atributos da classe:

- **map<string, double> wmap_:**

É uma variável do tipo map<string, double> que associa cada palavra(string) a seu w(double).

- **double norma_vetor_;**

É uma variável que armazena a norma do vetor construído por essa classe. A norma estará ao quadrado, pois a raiz quadrada não será calculada aqui, e sim na hora da montagem do cosine ranking.

Métodos da Classe:

- **void inserir_no_wmap(string palavra, double valor):**

Serve para inserir associações de palavras com seus W no atributo wmap_. Ele será calculado a partir da multiplicação do tf e do Idf, assim como na fórmula apresentada na documentação do trabalho.

- **void exibir():**
  Apresenta todo o nmap_ como saída para o usuário na tela.

- **double operator[](string palavra):**

Faz com que se o usuário inserir uma palavra como parâmetro(de um operator [ ]), o wmap retorna o W(IDF*TF) da palavra referida.

- **double norma_vetor():**
  Retorna o atributo norma_vetor_: útil para o cálculo do Cosine Ranking.

Conclusão:
A classe WMAP funciona basicamente para armazenar os maps que irão para a classe wvector, e possuir métodos para modificar, exibir e retornar seu atributo de tipo map<string, double>.

classe WVECTOR:

Introdução:
A classe wvector nada mais é do que a implementação do w(requisitado no documento) em si. Cada uma das posições de seu vetor representa um documento, que possui nessa posição uma associação de todas as palavras existentes com seus respectivos valores w(IDF * TF).

Atributos da classe:
- **vector<wmap> w_;**
  É um vector que armazena dados do tipo wmap(que já foram explicitados no trabalho).

Métodos da Classe:
- **void inserir_vetor(wmap documento)**
  Insere no final de um vetor w_ um objeto do tipo wmap, que é referente a um documento(cada wmap inserido é referente a um documento em específico).

- **void exibir()**
  Exibe todo o w_. Cada posição do w, é um objeto wmap, essa função aciona o método exibir de todos esses objetos do vetor.

- **vector<string> vetorNaoRep(vector<string> palavra)**
  Método para auxiliar na construção do w_, usado para receber um vetor de palavras existentes em todos os documentos e eliminar todas as repetições de palavra.

- **bool existe(vector<string> x, string palavra)**
  Método em que se passa como parâmetro um vetor que contenha palavras repetidas ou não e uma string palavra, esse método diz se existe essa string palavra dentro do vetor de palavras que foi passado.

- **wmap operator[](int documento)**
  Método que retorna o conteúdo referente a posição do vetor inserida no operator [ ];

Conclusão:
A classe wvector é essencial para se fazer o ranking de cossenos, de forma em que é possível acessar facilmente o valor w de cada palavra no documento indicado através do operator[ ].

classe Busca:

Introdução:

A classe Busca, localizada no arquivo Busca.cpp, tem como função interligar todas as classes supracitadas com a pesquisa do usuário, criando através delas as informações necessárias, e calcular “cosine ranking”, o ranking de similaridade entre os documentos e a pesquisa do usuário e retorná-lo para o usuário imprimindo na tela.

Atributos da classe:

- **LeituraArquivos arquivos_;**
  Variável da classe LeituraArquivos, que será utilizada para invocar os métodos desta classe.

- **string pesquisa_;**
  Variável que contém a pesquisa do usuário propriamente dita.

- **wmap w_pesquisa_;**
  Variável da classe wmap, que será utilizada para invocar os métodos desta classe.

- **frequencia_palavra tf_pesquisa_;**
  Variável da classe frequencia_palavra, que será utilizada para invocar os métodos desta classe.

- **std::map<double, list<string>> cosine_ranking_;**
  Variável que armazena o valor da similaridade entre os vetores pesquisa usuario e documentos com os documentos que possuem tal valor, sendo possivel haver 1 ou mais com mesmo valor.

- **wvector wvector_;**
  Variável da classe wvector, que será utilizada para invocar os métodos desta classe.

- **double parte_de_cima_;**
  Armazena o dividendo da fórmula para cálculo da similaridade (Cosseno) entre os vetores no cosine_ranking_build.

Métodos da Classe:

- **Busca();**
  Inicializa um objeto do tipo busca vazia, chamando todos os métodos para ler os arquivos, da classe “LeituraArquivos”.

- **Busca(string pesquisa);**
  Inicializa um objeto do tipo busca, chamando todos os métodos para ler os arquivos, da classe “LeituraArquivos”. Posteriormente, realiza uma busca nos documentos com a string recebida como parâmetro. Usada principalmente nos testes unitários.

- **void LeituraDosArquivos();**
  Método responsável por ler os documentos da coleção.

- **void pesquisa_usuario_digita();**
  Método responsável por receber, do teclado a busca do usuário e encaminhá-la para o método pesquisa_usuario(string pesquisa).

- **void pesquisa_usuario(string pesquisa);**
  Método que inicia a busca pelos arquivos mais relevantes ao usuário. Este método é responsável por iniciar, sequencialmente, todos os outros métodos responsáveis pela ordenação dos documentos de acordo com sua similaridade com a busca digitada pelo usuário.

- **void tf_pesquisa();**
  Método responsável por criar a frequência das palavras na pesquisa do usuário.

- **int tf_retorna(string palavra);**
  Método responsável por retornar a frequência da palavra passado como parâmetro na pesquisa do usuário.

- **void w_pesquisa_construcao();**
  Constrói o vetor “w” para a pesquisa do usuário.

- **void cosine_ranking_build();**
  Constrói a variável de similaridade entre cada documento e a pesquisa do usuário e armazena na variável cosine_ranking_,que já representa o ranking de relevância dos documentos.

- **void parte_de_cima_sim(int num_doc);**
  Método responsável pelo cálculo das variáveis que armazenam os valores isolados a serem utilizados na fórmula para o cálculo da similaridade (Cosseno) entre os vetores no cosine_ranking_build.

- **void imprimir_resultado_pesquisa();**
  Escreve na tela o resultado da pesquisa do usuário, mais especificamente os documentos em ordem decrescente de similaridade com a pesquisa digitada pelo usuário.

- **std::map<double, std::list<string>> cosine_ranking();**
  Retorna o atributo cosine_ranking_. Pode ser útil para diferentes tipos de implementação.

Conclusão:

Nesta classe é realmente calculada a similaridade entre a busca do usuário e retornado a ele o ranking de relevância de cada documento. A partir desta, todas as outras classes serão iniciadas, de forma direta e/ou indireta, e suas funções convergem para assim retornar o resultado esperado.

Conclusão

Em conclusão à este trabalho, houve sucessivos testes unitários como requisitado no documento de requisições, com base nesses testes podemos relatar a execução bem sucedida do software, e a boa organização da estrutura do código, que permitiu a otimização dos trabalhos do grupo, que foi o grande beneficiado de todo esse processo, pois cada membro além de melhorar seus conhecimentos em C++, desenvolveu-se em outras habilidades como ferramentas como o GitHub e suas funcionalidades, desenvolvimento de software em equipe, tato para escolher qual seria o melhor TAD para cada situação, entre outras.

Com base também nos testes, o código não é muito eficiente enquanto velocidade de execução: testes realizados no programa, constataram que ele consegue ler e fazer busca em 1000 documentos num tempo médio de 3 minutos. Foram tomados todos os cuidados para com o tratamento de erros e exceções, fazendo um trade-off entre velocidade e qualidade dos algoritmos.