



UNIVERSIDADE FEDERAL DE MINAS GERAIS

Programação e Desenvolvimento de Software II

RELATÓRIO SOBRE TRABALHO PRÁTICO FINAL:

Desenvolvimento de Algoritmo de Máquina de Busca em C++

Equipe:

- Bruno de Almeida Sena
- José Raimundo de Castro Filho
- Leonardo Medeiros de Sena

Belo Horizonte, Junho 2019.

1. INTRODUÇÃO

Uma máquina de busca fornece suporte para a procura de informações armazenadas em um banco de dados. Quando realizamos uma consulta é criada previamente uma lista de ocorrências para podermos cruzar os dados e obter um resultado satisfatório que é saber em quais documentos estão as palavras que procuramos ou quais os possíveis documentos.

Foi proposto aos alunos a implementação de uma Máquina de busca que gera um ranking que exhibe a ordem dos arquivos que provavelmente apresentam a palavra. O desafio proposto foi deste projeto foi estabelecer funcionalidades e soluções dadas ao problema de criação dessa máquina de busca com índice invertido. Neste conceito, o índice invertido é uma estrutura contendo uma entrada para cada palavra (termo) que aparece em, pelo menos, um documento.

Para isto este relatório organizou as informações em quatro fases de realização separadas em Planejamento, Implementação, Análise crítica e Validação, Ações e melhorias.

Durante os testes do projeto foram registradas informações de problemas que foram encontradas e soluções que foram aplicadas pela equipe e que solucionaram a necessidade do momento.

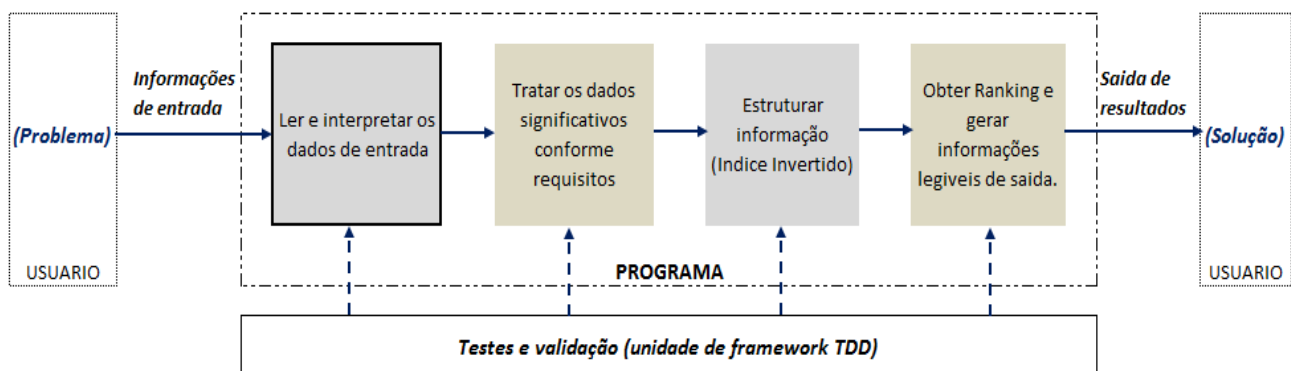
2. DESENVOLVIMENTO

2.1. Planejamento do projeto

Para construir a máquina de busca do presente trabalho criamos três classes, sendo elas: Leitura, Índice e Ranking. Para isso no programa principal foi definido a inclusão de uma função mestra chamada “Máquina_Busca” na qual funciona como um roteiro que vai chamando as rotinas na sequencia necessária de execução. Essa ideia da função mestra inclusive foi incorporada posteriormente durante a revalidação do programa como modo de melhorar o entendimento da execução do programa.

Este projeto para resolução do problema proposto, devido a suas características, foi classificado em quatro partes lógicas, sendo estas:

- Recebimento de dados de entrada leitura de arquivos;
- Tratamento dos dados significativos;
- Organização estruturada de índice invertido;
- Formação do ranking e registros de saídas.



2.2. Implementação do projeto

Neste tópico as informações do projeto foram organizadas em funcionamento geral que explica o funcionamento de forma sistêmica do programa e funcionamento específico que descreve de forma detalhada dos principais componentes do código no qual formam o algoritmo do programa.

2.2.1 Funcionamento geral do programa

Com base no conceito apresentado no tópico 2.1, foi criado uma função na “main” chamada “Maquina_busca” que não retorna nenhum valor, recebe como parâmetros a quantidade de arquivos que serão lidos e formarão a base de dados e a quantidade de palavras que serão pesquisadas posteriormente. Ao final dessa função o ranking é impresso na tela.

O primeiro passo do nosso programa é receber os nomes dos arquivos e extrair as palavras de cada um deles. Isso foi feito através da função “ReadFile” presente na classe leitura ela recebe como parâmetros a quantidade de arquivos que serão lidos e o objeto da classe “Índice” que foi criada para armazenar o índice invertido (um map que armazena todas as palavras do banco de dados as relacionando com os arquivos que ela está presente), além de abrigar funções úteis para a montagem desse Map), como por exemplo as funções “TiraCaracter” e “TudoMinusculo” que retiram caracteres do texto e transformam todas as letras maiúsculas em minúsculas respectivamente.

Nesta função todas as palavras de um arquivo são inseridas em um list e ao final da leitura do arquivo esse list é inserido em um vector de list o qual se encontra também na classe índice.

Um ranking de coordenadas de cada arquivo foi então construído e armazenado em um vector de vector da classe ranking para isso utilizamos o índice invertido que continha todas as palavras de todos os arquivos.

Para a construção do ranking utilizamos as classe índice ,ranking e leitura.

Então o usuário digita as palavras que vão ser consultadas.

É criado um vector de coordenadas da busca.

É criado um vector de similaridade de cada arquivo com as palavras pesquisadas

Finalmente é impresso na tela o ranking .

2.2.2 Funcionamento específicos do programa

Com base nos requisitos estabelecidos para realizar este trabalho, o grupo estruturou as informações no quadro esquemático abaixo de modo a facilitar o que foi realizado na prática, por meio dos programas necessários para desenvolvimento de modo a atender estes requisitos.

QUADRO ESQUEMÁTICO PARA DESCRIÇÃO DAS PRINCIPAIS FUNÇÕES COMPOSTA NO PROGRAMA		
Requisitos do TP.	Nome do arquivo/ tópico/ funções	Modo de operação / observações
Maquina de Busca	main.cpp	A função principal chama a função mestra “Maquina_busca” que serve como roteiro para chamar as funções subsequentes para realizar cada operação conforme descritas nas linhas abaixo.
Leitura de arquivos	“leitura.h”	O sistema recebe como entrada um conjunto de arquivos de texto, que devem ser lidos, palavra após palavra, para construir o índice invertido.
	Classe: Leitura	Classe declarada para atribuir os objetos e variáveis para realização das Leituras de arquivos. Possuem variáveis para registrar a quantidade de arquivos lidos e armazenar o nome dos mesmos. Esta classe é amiga da classe Ranking .
	Método: ReadFile	Método que lê uma quantidade de arquivos e dentro dela será feita uma consulta de palavras e será gerado um ranking de arquivos que contem as palavras da busca
	arquivo_	Variável ponteiro tipo “ifstream” para ler as palavras do arquivo.


Requisitos do TP.	Nome do arquivo/ tópico/ funções	Modo de operação / observações
Leitura de arquivos	nomeArquivo;	Variavel tipo “vector <string>” onde são armazenados os nomes dos arquivos.
	quantidadeArquivos;	Variavel tipo inteiro quantidade de Arquivos lidos
Organização das informações dos arquivos lidos	“indice.h”	Arquivo que declara as funções e variáveis que possibilitaram a organização das palavras no modo de índice invertido.
	Classe: Indice	Classe declarada para atribuir os objetos e variáveis para organizar as palavras em modo “índice invertido”. Esta classe é amiga da classe Ranking .
	Método: Pertence	Função do tipo booleana que retorna TRUE se uma palavra procurada (tipo string “Word”) pertence ao arquivo que está sendo lido.
	Método: Tudominusculo	Função do tipo void que converte todas as letras (tipo string&Word) de uma palavra para minúsculo.
	Método: TiraCaracter	Função do tipo void que remove caracteres (variável tipo string&word) não alfanuméricos, de pontuação e especiais tais como, . ; : !][(-) durante a leitura do arquivo.
	Método: TermFrequency	Função do tipo inteiro que retorna quantas vezes uma palavra, variável tipo string passando por referencia as variáveis “word,vector< list<string> >palavras,int i) na qual será registrada em um arquivo após a leitura.
	Método: palavrasDocs;	Função do tipo vector <list <string> > na qual armazena as palavras de todos os documentos ou seja o list de cada arquivo.
	palavrasDocs	Variavel tipo “vector <list <string> >” armazena as palavras de todos os documentos ou seja o list de cada arquivo
	palavras;	Variavel do tipo “map <string, set<string> >” que armazena o índice invertido.
Consultas através de um ranking	“ranking.h”	Arquivo que declara os métodos e objetos estruturados para ranquear as consultas de palavras por ordem de maior quantidade encontrada nos documentos consultados pelos métodos contidos nas classes de leitura. Os principais métodos estão explicados nos tópicos seguintes.
	Classe: Ranking	Classe declarada para atribuir os objetos e variáveis para organizar as palavras em ranking conforme por quantidade encontrada na busca. Esta classe é amiga da classe Leitura .
	Método: idf	Função do tipo “double” que calcula o idf, sigla do termo “Inverse term frequency” de cada palavra recebendo por referencias as variáveis contidas na classe “Indice”, da string “WordIDF” e “quantidadeArquivos”.
	Método: coordenadas_W	Função do tipo “void” que registram as coordenadas de um vetor correspondente a um documento extraídas da base de dados da base de dados (índice 'W') que recebe por referencia dados da classe “índice” e variáveis posição do documento “posicao_doc”, e “quantidadeArquivos”.

Requisitos do TP.	Nome do arquivo/ tópico/ funções	Modo de operação / observações
Consultas através de um ranking Consultas através de um ranking	Método: coordenadas_Q	Função do tipo "void" que cria as coordenadas referente a busca e armazena no vetor coordenadasBusca que recebe por referencia dados da classe "indice" e variáveis posição do documento "posicao_doc", e "quantidadeArquivos".
	Método: TermFrequency_Q	Função do tipo "int" que calcula a quantidade de vezes que uma palavra aparece na busca que recebe por referencia dados das variáveis "word" e "vector<string>palavrasBusca".
	Método: consultaQ	Função do tipo "void" que executa uma consulta de Palavras que recebe por referencia dados do objeto de classe "Indice".
	Método: Similaridade	Função do tipo "void" que calcula as similaridades de palavras e recebe a variável tipo vector.
	Método: ImprimeRanking	Função do tipo "void" que imprime o Ranking contido em vector<string>nomeArquivo).
	Método: RetornavalorMaior	Função do tipo "double" que retorna o maior valor em um vetor contido em um "vector<double>pesos".
Estruturas de dados para armazenar as coordenadas dos documentos	coordenadas_docs;	Variável tipo "vector" que armazena as coordenadas de cada documento
	palavrasBusca;	Variavel tipo "vector<string>" que armazena as palavras da busca em um vetor
	coordenadasBusca;	Variavel tipo "vector<double>" que armazena as coordenadas da busca
	similaridade_;	Variavel tipo "vector<double>" que armazena as similaridades obtidas.
Testes de unidade	Framework doctest "doctest.h"	"frameworks" de automação de testes "doctest" estrutura de testes unitários em C++
Documentação	Relatório TP Final (formato.pdf)	Conforme os registros apresentados por meio deste relatório.
Referencia de Consulta	Plataforma de hospedagem de código-fonte GITHUB	A lista completa está disponível na web conforme link abaixo: https://github.com/lmsena/TP_FINAL_PDS2

2.2.3 Instruções de operação do programa

Para utilizar o programa o usuário deve fornecer como parâmetro da função "Maquina_busca", presente na main, (vide figura abaixo) um inteiro correspondendo ao número de Arquivos que serão lidos e outro inteiro referente ao numero de palavras que serão consultadas.

```
int main () {
    cout<<"TP FINAL DA DISCIPLINA PDS2 , ALUNOS: JOSE RAIMUNDO, LEONARDO E BRUNO"<<endl<<endl;
    Maquina_busca(4,2);
    return 0;
}
```



Após, será solicitado no terminal os nomes dos arquivos em sequencia.

Posteriormente serão solicitadas as palavras de busca. E assim será gerado o Ranking.

Esta informação se encontra disponível na plataforma github conforme link em destaque neste relatório.

Exemplo da tela de Execução de teste.

```
TP FINAL DA DISCIPLINA PDS2 , ALUNOS: JOSE RAIMUNDO, LEONARDO E BRUNO
Digite o nome do 1 arquivo no formato file.txt
d1.txt
Digite o nome do 2 arquivo no formato file.txt
d2.txt
Digite o nome do 3 arquivo no formato file.txt
d3.txt
Digite o nome do 4 arquivo no formato file.txt
d4.txt
Digite a 1 palavra que voce deseja pesquisar
a
Digite a 2 palavra que voce deseja pesquisar
b
-----Ranking-----
Posicao ---Documentos
1          d4.txt
2          d1.txt
3          d2.txt e d3.txt
Process returned 3 (0x3)   execution time : 25.448 s
Press any key to continue.
```

2.3. Verificação, testes e validação do projeto

Os testes de unidades foram conduzidos ao longo do desenvolvimento do programa, fazendo uso de funções 'cout', de inserção de parâmetros com resultados previsíveis nas funções para testar o funcionamento de todas as etapas de código bem como validar a capacidade de detectar erros nas entradas dos parâmetros em alguns casos.

Foi utilizado o "frameworks" de automação de testes "doctest" é uma nova estrutura de testes em C ++, é considerada mais rápida, tanto em tempos de compilação (por ordem de grandeza) quanto em tempo de execução, em comparação com outras frameworks.

Para realizar os testes nos documentos do formato ".txt", foram criados um construtor para testes que invoca a função "**void Leitura::ReadFileTeste**" que executa as rotinas de manipulação de palavras de cada arquivo.

2.4. Análise crítica ações e melhorias do projeto

As melhorias e adequações foram sendo implementadas gradativamente, na medida em que os testes e as verificações vêm sendo realizadas ao longo do desenvolvimento do trabalho. Alguns dos problemas encontrados e soluções obtidas que foram registradas, foram inclusas neste relatório e organizadas no quadro a seguir. As melhorias foram consideradas as etapas que já funcionavam atendendo os requisitos, porém foram sendo aprimoradas na medida em que o programa foi sendo testado e atualizado. As que foram registradas estão no tópico B a seguir.

A) Tabela relacionada a problemas e soluções

Nome do arquivo/ elementos de código	Lista de problemas x solução.	
	Problema encontrado	Solução sugerida/obtida
main.cpp	A estrutura main criada anteriormente executava o código de modo menos amigável para entender sua execução.	Foi criada a função mestra "Maquina_de_busca" para facilitar o entendimento.
Leitura.cpp	Dificuldade para ler arquivos ".txt" relativamente grandes. Ex.: livro Hamlet.txt 139 kB. Durante os testes, quando para PC's de processamento relativamente rápido o tempo diminuía.	Criação de um vetor específico para registrar as frequências uma etapa anterior de gerar o Ranking.

B) Tabela de situação antes e depois das melhorias

Nome do arquivo/ elementos de código	Situação antes da melhoria	Situação depois das melhorias
palavrasDocs	Variavel tipo "vector <list <string> >" armazena as palavras de todos os documentos ou seja o list de cada arquivo	Melhoria: Estas funções substituiu o a variavel wordFile armazena as palavras de um unico arquivo tipo list <string> na qual foi inserida em vector para facilitar a organização dos dados.
palavras;	Variavel do tipo "map <string, set<string> >" que armazena o índice invertido.	

3. CONCLUSÃO

Este trabalho possibilitou entender de maneira prática como uma máquina de busca funciona por meio dos recursos disponíveis na linguagem de programação C++. Com isso, pôde-se perceber o funcionamento e utilização de bibliotecas pouco conhecidas até o momento como “map” e “list” bem como técnicas que podem ser aplicadas conforme o desafio proposto pelos critérios pelo roteiro do professor. O celular como uma ferramenta de pesquisa.

Para se atingir o resultado desse projeto, definiram-se dois objetivos específicos. O primeiro, de fazer o levantamento de todos os recursos que a linguagem C++ nos oferecia considerando o conteúdo ministrado durante o semestre, que nos ajudaria a realizar este trabalho em tempo hábil. Percebemos que apesar do programa ter funcionado encontramos dificuldades para aumentar sua eficiência para leitura e “ranqueamento” dos resultados dos IDF’s.

Durante a realização dos testes percebemos o quanto os testes unitários por bloco de código foi importante para explicitar erros nos quais são omitidos pelos compiladores convencionais das IDE’s.

Observou-se o quanto o compilador é limitado para detecção de bug’s que ocorrem durante a execução que não são facilmente percebidos nos dispositivos de saída.

Como já esmiuçado no tópico de implementação do projeto, tem-se, de maneira geral, um melhor conhecimento sobre algumas diretivas de comando disponíveis nas bibliotecas padrão de C++. Conversas entre a equipe foi muito útil para apontar os problemas e encontramos as soluções.

Em consonância com os exemplos elencados nos materiais e bibliografia utilizados na disciplina de PDS 2 percebe-se que desafiou o grupo demonstra o potencial criativo que ainda precisa ser muito explorado para compreensão do conteúdo apresentado conforme ementa da disciplina.

Por fim este trabalho nos colocou mais preparados para outras atividades similares que certamente hão de vir nas disciplinas posteriores ao longo do curso. O exercício deste trabalho foi útil e eficaz para agregar uma noção sólida de qual caminho a seguir para desenvolver uma máquina de busca mais robusta e eficiente que esta realizada por meio deste trabalho.