**UNIVERSIDADE FEDERAL DE MINAS GERAIS**
**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**Integrantes**: Estevão Felipe Fonseca, João Marcos Ribeiro Tolentino e Jonas Francisco Lopes

**Documentação do trabalho prático da disciplina:** Programação e Desenvolvimento de Software II

**Turma: TW**

**Ambiente computacional**: GCC

**Repositório (github)**: https://github.com/jonas-lopes/PDS2-TP.git

## Introdução

O algoritmo em questão implementa um construtor para a classe Diretório que recebe um caminho como parâmetro. O objetivo deste é analisar os arquivos presentes no diretório especificado e criar um registro das palavras encontradas nesses arquivos. Foi dividido as partes do código entre o trio e foi feito uma certa quantidade de testes de acordo . Foi feito via repositório do github o projeto descrito em que fizemos o trabalho usando o Makefile para executar e compilar os arquivos necessários para o seu pleno funcionamento.

Ou seja, para rodar o projeto o monitor que irá testar pode apenas usar o comando “make”. A máquina de busca vai receber como entrada um termo de busca que processa ela, e dá como saída os documentos que, de acordo com a estratégia usada, são os mais relevantes para a consulta fornecida.

Deseja-se buscar por cada termo digitado do teclado em todos os arquivos e retornar apenas aqueles em que achar o texto completo. Tem se no compartimento do Diretório  a função normatizar (essa vai retirar acentuação e os caracteres especiais, além de travessão e ponto) e a de buscar irá fazer esse processo necessário para apresentar os documentos com essa string completa digitada. Temos a definição de que o índice invertido é uma palavra e o valor associado a ela, além de que a base de dados é fixa. Em caso de empate, mostra o arquivo menor lexicograficamente como descrito na descrição do próprio TP. O código todo está comentado para que todos tenham o entendimento e compreensão de cada parte.

## Implementação

*Observação: essa parte mais técnica irá destrinchar cada trecho do código de forma mais técnica.

Em suma, a função “directory_iterator(caminho)” é utilizada para iterar sobre os arquivos do diretório especificado pelo caminho. Para cada arquivo encontrado, verifica-se se um arquivo é regular através de uma função chamada: “entry.is_regular_file()”. Caso seja um arquivo regular, cria-se um objeto para abrir o arquivo. Verifica se o arquivo foi aberto corretamente com arquivo o “is_open()”. Em seguida, lê-se o arquivo linha por linha com o auxílio da função padrão getline(arquivo, linha).

Para cada linha lida, o algoritmo percorre cada palavra presente na linha e realiza as seguintes ações:

São definidos dois iteradores, palavra_begin e palavra_end, que apontam para o início e o fim de uma palavra na linha, respectivamente.

Enquanto não chegar ao fim da linha, verifica-se se o caractere atual não é um espaço em branco.

Se o caractere não for um espaço em branco, incrementa-se o palavra_end para avançar para o próximo caractere.

Caso contrário, cria-se uma palavra a partir do intervalo delimitado por palavra_begin e palavra_end, que representa a palavra encontrada na linha.

A palavra é normalizada através da função normatizar(palavra) antes de ser utilizada como chave em um mapa palavras.

No mapa palavras, a palavra é mapeada para um mapa aninhado, onde a chave é o caminho do arquivo em que a palavra foi encontrada e o valor é o número de ocorrências dessa palavra no arquivo. O operador ++ incrementa o contador de ocorrências para cada palavra encontrada. Em seguida, a palavra "begin" é incrementada para apontar para o próximo caractere após o espaço em branco.

Se, ao final do processamento da linha, o palavra_begin não for igual ao fim da linha, significa que há uma palavra restante que não foi processada no loop anterior. Nesse caso, é criada uma palavra com o intervalo restante e realiza-se as mesmas operações de normalização e registro no mapa das palavras. Após o processamento de todas as linhas do arquivo, fecha-se o arquivo com arquivo.close(). O processo é repetido para todos os arquivos encontrados no

diretório. O resultado desse algoritmo é um registro das palavras encontradas nos arquivos do diretório especificado, onde cada palavra é mapeada para um mapa aninhado que registra o caminho do arquivo e o número de ocorrências da palavra nesse arquivo.

É importante ressaltar que a função normatizar() não está presente no código fornecido e deve ser implementada externamente. Essa função provavelmente é responsável por aplicar alguma forma de normalização nas palavras antes de serem registradas no “mapa” de palavras. O comportamento exato dessa função não pode ser determinado apenas pelo código fornecido.

O método buscar vai receber um termo de busca como parâmetro passado no main e vai retornar uma string que contenha os resultados da mesma. Basicamente, será criada uma variável resultante que vai armazenar os resultados da busca e um ranking para armazenar o número de ocorrências de cada documento.

O termo de busca é dividido em palavras e armazenado no vetor que foi criado. Depois disso, vai iterar os documentos no membro de dados palavras. Para cada documento e verificar se ele contém todas as palavras de busca.

Se o documento contiver todas as palavras, o número total de ocorrências é calculado e armazenado no ranking como. Após iterar sobre os documentos, foi criado um mapa chamado para armazenar os documentos ordenados pelo ranking de ocorrências. Em seguida, iterando sobre o ranking e inserindo sobre os documentos no documento Ordenados. Por fim, vai iterar sobre o documentos Ordenados e constrói-se a string resultante com o nome do documento, o número de ocorrências.

Tem-se também a função normalizar de uma classe “Diretorio”. O objetivo desta função é converter quaisquer letras maiúsculas em minúsculas enquanto preserva as letras minúsculas como elas são.

É declarada uma variável de string “palavraFormatada” para armazenar a string normalizada. Obtém-se o tamanho da string de entrada usando  a função padrão ‘size()’ e é armazenado em uma variável para iterar até a última posição. Foi feito um loop que itera enquanto c for menor que essa variável que representa o comprimento da string de entrada.

Dentro do loop, foi verificado o valor ASCII do caractere no índice c na string de entrada `palavra`. Se o valor ASCII estiver entre 65 e 90 inclusive, significa que o caractere é uma letra maiúscula. Para isso, caso esteja dentro desse intervalo é

somado 32 ao seu valor ASCII e acrescentado esse caractere minúsculo na variável de palavra formatada.

- Se o valor ASCII estiver entre 97 e 122 (inclusive), significa que o caractere é uma letra minúscula.

- É acrescentado o caractere minúsculo na variável “palavraFormatada” sem nenhuma alteração e incrementado o contador em 1 para mover para o próximo caractere na string de entrada. No geral, a função pega uma string de entrada, itera sobre seus caracteres, converte qualquer letra maiúscula em minúscula e retorna a string normalizada resultante.

## Conclusão

Portanto, com esse trabalho, conseguimos usar a ferramenta git, trabalhar conjuntamente, resolver bastante conflito que ocorreu durante o processo e chegar do outro lado de acordo com o que foi proposto. Nesse sentido, o código foi construído da forma mais reduzida e compreensível possível de acordo com os princípios e boas práticas do livro: “Clean Code: A Handbook of Agile Software Craftsmanship”. Durante o processo, ocorreram conflitos na branch principal, então tivemos que recriar a mesma e ir fazendo por cima.

Dessa forma, o integrante que ainda não tinha mexido com git teve sua primeira experiência e os outros dois tiveram essa mesma reforçada. Revisamos também a questão do make e outros conteúdos vistos durante o semestre nos vpls e materiais disponibilizados.

Foi uma experiência que deu para exemplificar a rotina de uma pequena equipe trabalhando em um projeto prático usando ferramentas cooperativas.