UFV Campus Florestal	**Universidade Federal de Viçosa** **Campus Florestal** **Algoritmos e Estruturas de Dados II** **2023/1**

1º Trabalho Prático de CCF212 – 20 pts

APLICAÇÃO COM ÁRVORES DIGITAIS

Formação do grupo: O trabalho deverá ser desenvolvido em **13 grupos de 4** integrantes e **1 grupo com 3** integrantes. Sugere-se que os grupos sejam organizados dentro da divisão de turmas práticas (não obrigatório).

Problema: Construção de índice invertido para uma máquina de busca dos POCs do curso de Ciência da Computação. Atualmente, os POCs estão disponíveis como arquivos, em formato pdf, organizados no site do curso (https://ccp.caf.ufv.br/tccs/) por semestre de apresentação. Ainda não existe nenhum tipo de sistema de busca que permita a aplicação de filtros para recuperação dos trabalhos.

Fundamentação Teórica: Máquinas de busca, tais como Google, trabalham com a busca de palavras-chave em textos armazenados na Web. Para que os documentos contendo os termos sejam recuperados, os mesmos precisam ser devidamente indexados à priori. Nesse contexto, são utilizadas estruturas de dados que facilitem a recuperação das informações, como é o caso do uso de arquivos invertidos. Dada uma coleção de documentos, um índice invertido é uma estrutura contendo uma entrada para cada palavra (termo de busca) que aparece em, pelo menos, um dos documentos. Essa entrada associa a cada palavra do texto um ou mais pares do tipo <*qtde, idDoc*>, onde *qtde* corresponde ao número de vezes em que a palavra em questão apareceu em um determinado documento identificado por *idDoc*. O nome dado à estrutura indica que houve uma inversão da hierarquia da informação, ou seja, ao invés de uma lista de documentos contendo termos, é obtida uma lista de termos, referenciando documentos. Essa estrutura de índices é comumente implementada com base em *árvores* e tabelas *hash*, pois as mesmas não precisam ser reconstruídas a cada atualização.

Para exemplificar, considere os dois documentos mostrados abaixo[1]:

Texto 1 (arquivo1.txt) = "*Quem casa quer casa. Porem ninguem casa. Ninguem quer casa tambem. Quer apartamento.*"

Texto 2 (arquivo2.txt) = "*Ninguem em casa. Todos sairam. Todos. Quer entrar? Quem? Quem?*"

[1] Nos exemplos, os textos não têm acentuação. Além disso, palavras com letras maiúsculas foram transformadas para minúsculas antes da inserção no índice.

Supondo os identificadores 1 e 2 para os textos apresentados, arquivo1.txt e arquivo2.txt, respectivamente, o índice invertido para as palavras contidas nos textos pode ser visualizado na Tabela 1.

Tabela 1 – Índices invertidos para os textos apresentados

Palavra	**<qtde, idDoc>**	**<qtde, idDoc>**	**...**
apartamento	<1,1>		
casa	<4, 1>	<1, 2>	
em	<1, 2>		
entrar	<1, 2>		
ninguem	<2, 1>	<1, 2>	
porem	<1, 1>		
quem	<1, 1>	<2, 2>	
quer	<3, 1>	<1, 2>	
sairam	<1, 2>		
tambem	<1, 1>		
todos	<2, 2>		

Conforme pode ser observado na Tabela 1, para cada palavra, existe uma lista de pares de números *<qtde, idDoc>*, ordenada pelo campo *idDoc*. Em termos práticos, essa lista poderia ser implementada como uma lista encadeada para cada palavra indexada.

A partir dos textos indexados no arquivo invertido, podem ser realizadas pesquisas, com base em termos de busca, para se calcular a relevância de documentos. Neste trabalho, será utilizado um método de ponderação baseada no cálculo da frequência da ocorrência do (s) termo (s) nos documentos, conhecida como **TF-IDF (*Term frequency – Inverse Document Frequency*)**. Esse método é baseado na frequência dos termos da consulta em cada documento (TF) da

coleção bem como na frequência inversa dos documentos (IDF). O componente IDF estima o quanto um termo ajuda a discriminar os documentos entre relevantes e não relevantes. Um termo que aparece em muitos documentos tem valor de IDF baixo, enquanto um termo que aparece em poucos documentos apresenta IDF alto, sendo um bom discriminador.

Dada uma consulta com *q* termos, t1, t2, ..., tq, a relevância de um documento *i*, *r(i)*, é computada como:

$r(i) = \frac{1}{n_i} \sum_{j=1}^{q} w_j^i$	*ni* = número de termos distintos do documento *i* $w_{j,i}$= peso do termo *tj* no documento *i*
$w_j^i = f_j^i \frac{\log(N)}{d_j}$	$f_{j,i}$ = número de ocorrências do termo *tj* no documento *i* dj = número de documentos na coleção que contém o termo *tj* N = número de documentos na coleção. Se o termo *tj* não aparece no documento *i*, $f_{j,i}$ = 0

Para exemplificar, considere uma consulta com os termos "*quer*" e "*todos*"[2]:

- dois termos (q = 2)
- dois documentos: N = 2
  - o documento 1 tem 7 termos (n1 = 7)
  - o documento 2 tem 8 termos (n2 = 8)
- o número de ocorrências do primeiro termo (quer), no documento 1 é 3 e no documento 2 é 1, logo $f_{1,1} = 3$ e $f_{1,2} = 1$. Além disto dj = 2.
  - $w_{1,1} = 3*\log(2)/2 = 1.5$
  - $w_{1,2} = 1*\log(2)/2 = 0.5$
- o número de ocorrências do segundo termo (todos), no documento 1 é 0 e no documento 2 é 2, logo $f_{2,1} = 0$, $f_{2,2} = 2$ e dj = 1
  - $w_{2,1} = 0*\log(2)/1 = 0$
  - $w_{2,2} = 2*\log(2)/1 = 2$

[2] Exemplo retirado do material da Profa. Jussara Marques de Almeida (UFMG)

• Logo, as relevâncias dos documentos pra esta consulta são:

- r(1) = 1/7 *(1.5 + 0) = 0.21
- r(2) = 1/8 *(0.5 + 2) = 0.31

A partir das relevâncias calculadas para cada documento, o método retornará os documentos ordenados da seguinte forma:

Texto 2 (arquivo2.txt)
Texto 1 (arquivo1.txt)

Tarefas:

1. **Receber, como entrada,** o arquivo **entrada.txt**, com *N* arquivos com textos (POCs) cujas palavras serão indexadas. O arquivo deverá conter o seguinte formato:

   N
   arquivo1.txt
   arquivo2.txt
   arquivo3.txt
   .....
   arquivoN.txt

   A primeira linha deste arquivo contém um número *N* que representa o número de documentos da coleção. Cada linha a seguir contém o nome do arquivo que contém um dos documentos da coleção. No exemplo acima, há *N* documentos que estão armazenados nos arquivos arquivo1.txt, arquivo2.txt, ..., arquivoN.txt. Você pode assumir que estes arquivos, caso existam (você precisa testar), estarão no diretório corrente de execução.

   O sistema deverá processar cada um dos arquivos, lendo palavra após palavra e construindo o índice invertido. Ele deverá também associar a cada documento um *idDoc* único e associar, em memória, este identificador com o nome do documento.

   Para simplificar e reduzir o volume de informações manipuladas, podem ser indexados apenas Título e Resumo dos trabalhos (inclua também palavras-chave, se houver). Sugere-se que os textos a serem indexados sejam copiados manualmente e armazenados localmente em um formato de manipulação mais fácil (txt, csv etc). Também para simplificar, sugere-se que sejam utilizados os trabalhos em inglês para evitar a manipulação de acentos.

2. **Criar um Índice Invertido**, usando um TAD árvore **PATRICIA**. Para isso, você deve adaptar os algoritmos fornecidos em sala de aula para permitir o armazenamento de palavras. A solução mais comum é inserir mais um campo de comparação em cada nó, ou seja, além do campo de índice (que avança x posições na palavra) será necessário também ter um campo com o caracter que está sendo comparado naquela posição para se decidir o caminho a seguir (esquerda ou direita). A decisão de se colocar, no nó interno, o menor ou o maior caracter de comparação e se os iguais ficarão à esquerda ou à direita deste nó, deverá levar em conta o melhor uso de memória e a diferença de tamanho entre as palavras. **Essa decisão é um componente importante no trabalho prático e precisa ser tomada com atenção!**
3. **Imprimir o índice invertido**: imprime as palavras da **PATRICIA**, em ordem alfabética, uma por linha, seguidas da lista das ocorrências, ordenada pelo índice do documento (pares <*qtde, idDoc*>).
4. **Implementar uma função de busca por textos com base em termo (s) de busca,** utilizando o índice invertido para localizar os textos em que ele (s) aparece (m). Os textos devem ser retornados de forma ordenada pela sua relevância para a consulta, ou seja, textos que contém um maior número de ocorrências do (s) termo (s) de busca devem aparecer primeiro.
5. **Implementar um menu com as seguintes opções:** a) receber o arquivo de entrada com os textos a serem indexados; b) construir o índice invertido, a partir dos textos de entrada, usando o TAD árvore PATRICIA; c) imprimir o índice invertido, contendo as palavras em ordem alfabética, uma por linha, com suas respectivas listas de ocorrências; d) realizar buscas por um ou mais termo(s) de busca, no índice construído, apresentando os arquivos ordenados por relevância.
6. **Utilizar a biblioteca GTK (GIMP Tool Kit) OU similar para implementação da interface gráfica com o usuário**.
7. **Elaborar um relatório sintético**, contendo:
    - Capa com título e formação do grupo;
    - Introdução, com o objetivo do trabalho e as principais fontes (referências) dos algoritmos utilizados;
    - Metodologia, documentando como o grupo se organizou em termos da divisão e da execução das tarefas e compartilhamento de código.
    - Resultados do desenvolvimento, apresentando: detalhes técnicos de implementação da PATRICIA para armazenar palavras; a interface construída; e resultados dos testes.
    - Considerações finais, em que o grupo pode registrar as principais facilidades, dificuldades e lições aprendidas; e

- Referências bibliográficas utilizadas.

Entrega:

- ✔ O trabalho deverá ser entregue via PVANET Moodle, por **um** dos integrantes do grupo, através de um **único** arquivo compactado (em formato .zip e nomeado com nome do grupo), contendo:
  - código-fonte do programa em C, com implementações dos TADs e manipulação dos textos de entrada em arquivos separados;
  - arquivos utilizados como entrada (escolha POCs de pelo menos 2 semestres);
  - arquivo Makefile (com utilitário make) para compilação e geração de código executável automaticamente.
  - arquivo "leiame.txt" com explicações de uso e execução do programa; e
  - relatório em formato pdf.
- ✔ Data de entrega:**29/05/23**
- ✔ Data de apresentação/entrevista: **30/05 e 01/06/23, em cronograma a ser divulgado.**

Comentários Gerais:

- O grupo deverá tomar como base, os códigos discutidos em aula, retirados do livro texto da disciplina (Ziviani, 2010). Outras fontes poderão ser consultadas;
- O código-fonte DEVERÁ ser devidamente comentado;
- As implementações relativas a cada TAD devem estar em arquivos separados;
- As operações referentes à leitura e à carga dos dados devem estar em um arquivo separado;
- As operações referentes à montagem do índice invertido devem estar em um arquivo separado;
- Atenção quanto ao uso e inicializações de variáveis no programa principal, que podem comprometer o funcionamento do seu código (Encapsular funções sempre que possível);
- Os integrantes do grupo deverão ser identificados no cabeçalho de TODOS os arquivos do código-fonte;
- Apesar de o trabalho ser em grupo, a nota poderá ser individualmente atribuída, a critério da professora (entrevistas individuais poderão ser realizadas);

- Em caso de plágio entre trabalhos, será atribuída nota **zero** para todos os envolvidos (dos grupos em questão) e atribuição de conceito **F**. Se houver discussões entre os grupos acerca de soluções para questões específicas dos algoritmos, não há problema, desde que isso esteja devidamente documentado no relatório e no código-fonte (na função correspondente).
- Trabalhos entregues **em atraso** ou que **não sejam apresentados** pelo grupo receberão nota ZERO.
- Durante o desenvolvimento do trabalho, caberá ao grupo propor e construir uma implementação para o problema apresentado. A professora e os monitores não irão resolver erros em código-fonte nem tampouco fornecer detalhes técnicos da solução a ser construída.