Engenharia e Ciência da Computação – Estruturas de Dados

Trabalho 3 – Search Engine

Informações Gerais

- **Data Limite de Entrega**: 15/12/2023 (23:59).
- **Pontuação**: 10 pontos + 1 ponto extra.
- **Formato de Entrega**: Os arquivos produzidos no trabalho devem ser compactados em formato **.zip** e submetidos na tarefa do ambiente virtual de aprendizagem (AVA).
- Os trabalhos devem ser desenvolvidos individualmente.
- **Importante**: Trabalhos entregues após a data limite sem justificativa com comprovação documental (atestado médico, etc.), ou que não estiverem de acordo com a especificado receberão nota zero.

Especificação

Este projeto tem como objetivo desenvolver uma ferramenta para buscar documentos que contêm trechos de textos digitados pelo usuário. Deverão ser desenvolvidas duas versões do sistema, uma utilizando tabelas hash e outra utilizando árvores binárias e a performance das duas versões deverá ser comparada. A tabela hash deverá utilizar a função de hash universal para *strings* (ver slides).

Deverão ser construídos dois programas.

- **Indexador**: recebe como entrada o nome de um diretório que irá conter arquivos texto e produz como saída um arquivo contendo o **índice**, uma estrutura de dados para tornar o processo de busca por textos dos arquivos mais eficiente.
- **Buscador**: recebe como entrada o arquivo contendo o índice e o texto que o usuário deseja buscar e retorna 10 recomendações de documentos, ordenados do mais relevante para o menos relevante.

O tempo para construção do índice e para obter o resultado da busca deverá exibido na tela. Junto à especificação são disponibilizados *scripts* python que solucionam o trabalho. Os scripts podem ser usados como pseudocódigo de referência. Dicionários fazem o papel da tabela hash e da árvore binária de busca do trabalho.

Indexador

A entrada do programa indexador é um diretório dentro do qual existirá um arquivo chamado “files.txt” que terá em cada linha o nome de um arquivo texto. A Figura 1 (a) ilustra as estruturas dos diretórios e a Figura 1(b) ilustra o conteúdo do arquivo “files.txt”. Cada um dos arquivos texto existente em “files.txt” possuirá uma linha de texto em minúsculo, sem pontuação ou acentos, e com palavras separadas por espaço. Estes textos são notícias de jornal que foram preprocessadas para tornar mais fácil a sua manipulação por computador.

O programa indexador deverá um índice no formato ilustrado na Figura 2. Para cada palavra, o índice deve indicar em que documentos ela aparece e com que frequência. A tabela hash ou

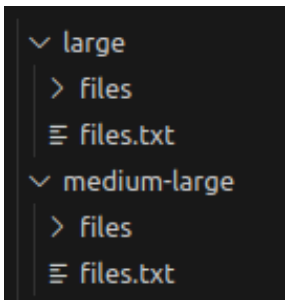
árvore binária de busca deve ser usada para armazenar o índice de forma que a sua construção e posterior busca sejam feitas eficientemente. Estas estruturas de dados armazenam pares chave-valor. A chave será a palavra e o valor será a coleção de documentos em que ela aparece e com que frequência. Esta coleção também deverá ser uma tabela hash ou árvore binária. Nesta segunda estrutura, chaves são os nomes dos documentos e os valores são a frequência da palavra nos documentos.

O procedimento para construir o índice é dado a seguir.

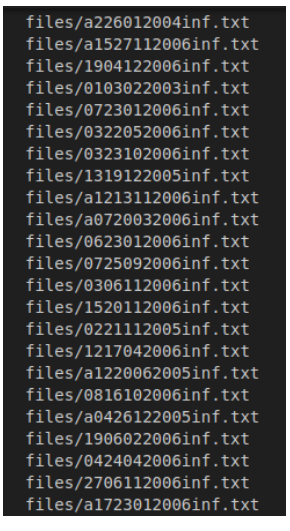
1. Crie o índice inicialmente vazio.
2. Para cada arquivo em “files.txt”, leia o conteúdo do arquivo e separe as palavras.
3. Para cada palavra:
    a. se ela ainda não existe no índice, adicione um novo par (palavra, tabela hash/árvore vazia). Esta segunda estrutura associada à palavra será utilizada para armazenar os documentos e a frequência da palavra nos documentos.
    b. Se o documento atual não existe na coleção de documentos, adicione um par (documento, 1) para indicar que é a primeira vez que a palavra é visualizada no documento.
    c. Se o documento já existe, incremente a frequência da palavra no documento.

Buscador

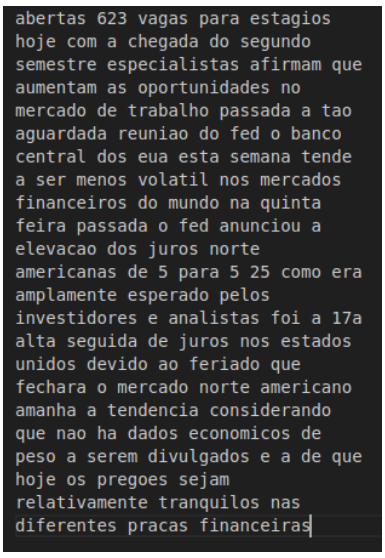
O programa buscador deverá solicitar que o usuário digite um texto e usar o índice para identificar os documentos em que as palavras do texto aparecem com maior frequência. Os dez documentos mais relevantes deverão ser exibidos na tela. A relevância de um documento é dada pela soma das frequências das palavras diferentes existentes na consulta. Se dois ou mais documentos possuem a mesma relevância, a ordem deles não é relevante.



(a) Exemplos de diretórios que poderiam ser usados



(b) Exemplo de conteúdo do arquivo files.txt.



(c) Exemplo de conteúdo de um dos arquivos texto.

como entrada para o indexador.

Figura 1: Ilustração das entradas do programa indexador.

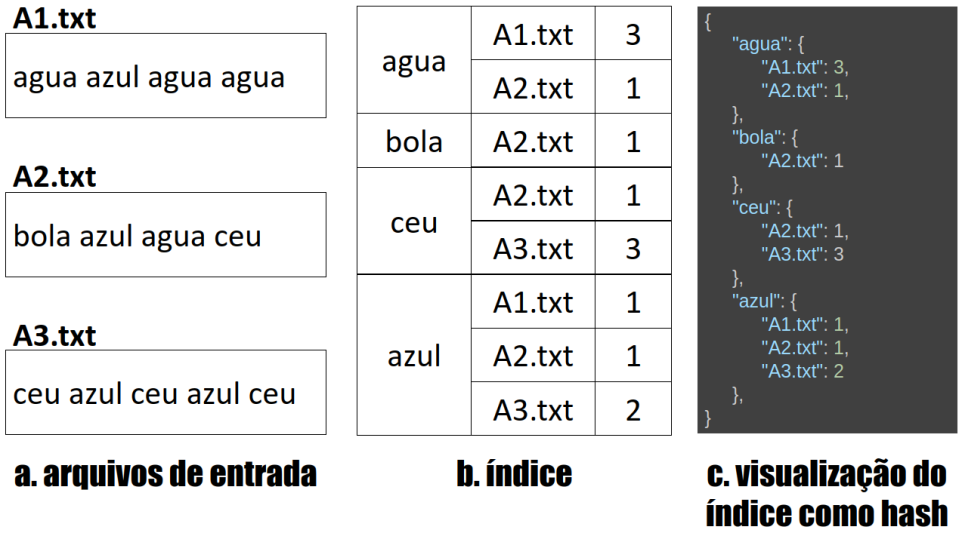


Figura 2: Ilustradação do índice.

A Figura 3 ilustra o funcionamento do buscador. Suponha que o usuário pesquisou o texto “ceu azul”. As palavras diferentes existentes na consulta são “ceu” e “azul”. Para cada palavra, vamos retornar os documentos em que ela aparece e as frequências. Para cada documento, verificamos se ele já existe na estrutura de dados de saída. Se sim, incrementamos sua relevância somando a frequência da palavra. Caso contrário, adicionamos o documento com a frequência da palavra como relevância inicial. No exemplo, o documento “A3.txt” foi o mais relevante para a busca por ter as palavras “ceu” e “azul” com frequência maior que os demais documentos.

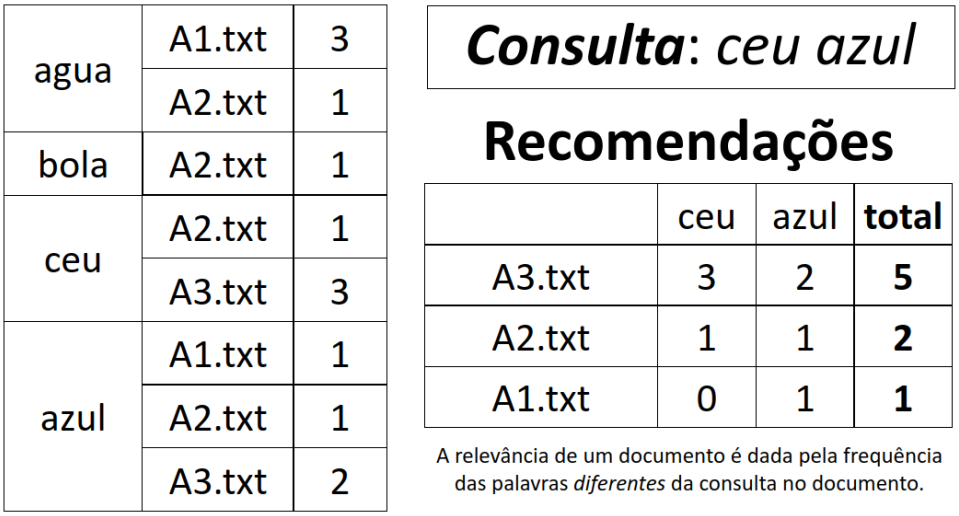


Figura 3: Ilustração do processo de funcionamento do buscador.

Ponto Extra

Será premiado com um ponto extra o aluno que implementar uma terceira versão do programa usando Vectors ao invés da tabela hash e da árvore. O objetivo desta terceira implementação é verificar uma substancial queda de performance utilizando esta estrutura de dados em comparação com as alternativas.

Regras

- As estruturas de dados utilizadas para armazenar coleções de itens (vector, listas encadeadas, heaps, etc.) devem ser opacas e genéricas. Demais estruturas de dados específicas do problema **não** precisam ser nem opacas nem genéricas.
- Envio de códigos que não compilam ou erros no valgrind do tipo *invalid read* ou *invalid write* resultarão em nota zero.
- Número de alocações diferente do número de *frees* resultará em punições graves na nota.
- Busque ser consistente no estilo de escrita do código. Um exemplo conciso de regras de estilo é dado a seguir. Você não precisa seguir estas regras específicas, mas é importante que você siga algum estilo específico.
    - Funções, variáveis e atributos de estruturas devem ser escritos em minúsculas e usando *underline* ("_") para separação de palavras.
    - Nomes de estruturas devem utilizar UpperCamelCase (ou PascalCase). Isto é, as palavras que compõe o nome devem iniciar com maiúsculas e devem ser unidas sem espaços.
    - Blocos de código em uma função devem ser separados por uma linha vazia. Um exemplo seria separar a declaração de variáveis do primeiro *loop* ou condicional da função. Funções devem ter menos de uma tela de tamanho.
    - Duas linhas vazias devem ser utilizadas para separar funções e para separar os includes das instruções iniciais em um arquivo.
    - Comentários devem ser utilizados para explicar trechos de código obscuros ou para documentar a interface (entradas e saídas) de funções. Demais trechos de código devem ser escritos de forma a serem compreensíveis sem a necessidade de comentários.