Indexação e busca de documentos

1. Problema 1: Construção do Índice Invertido para Máquinas de Busca

A base para o funcionamento de uma máquina de busca, tais como Google e Yahoo!, é a construção de um índice invertido para uma coleção de documentos. Dada a coleção de documentos, um índice invertido é uma estrutura contendo uma entrada para cada palavra (termo) que aparece em pelo menos um documento. Esta entrada associa à palavra pares do tipo <count, doc_id>, onde doc_id identifica um documento e count é o número de vezes em que a palavra em questão apareceu no documento doc_id. Documentos que não contém a palavra não precisam ser indicados.

Como exemplo, suponha uma coleção com apenas os dois documentos seguintes:

<doc1> Quem casa quer casa. Porem ninguem casa. Ninguem quer casa tambem. Quer apartamento.

<doc2> Ninguem em casa. Todos sairam. Todos. Quer entrar? Quem? Quem?

Supondo os identificadores 1 e 2 para <doc1> e <doc2>, respectivamente, o índice invertido seria:

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)

Note que, adjacente a cada palavra do índice está uma lista de pares de números representando ocorrências da palavra nos documentos. O primeiro número de cada par representa o número de vezes que a palavra ocorre em um documento e o segundo representa o identificador do documento em questão. Note que a lista está ordenada por identificadores de documentos.

O sistema deverá processar cada um dos documentos, lendo palavra após palavra e construindo o índice invertido. Ele deverá também associar a cada documento um doc_id único e associar, em memória, este identificador com o nome do documento. Cabe ressaltar que:

a) Uma palavra é considerada como uma sequência de letras e dígitos, começando com uma letra. Portanto, ignore sinais de pontuação.

b) Palavras com letras em maiúsculas devem ser primeiramente transformadas para minúsculas antes da inserção no índice. Desta forma, a mesma palavra apresentada com letras minúsculas ou maiúsculas não serão diferenciadas. Além disso, palavras com acentos serão normalizadas para palavras sem acento. Ou seja: “ação” virará “acao”.

2. Problema 2: Consultas sobre o índice invertido

Nesta parte do projeto, você utilizará o índice invertido construído para realizar consultas de um ou mais termos. O objetivo é, dada uma consulta, retornar uma lista de documentos, ordenados pela **relevância** para a consulta.

Existem vários métodos de estimar a relevância de documentos de uma coleção para uma consulta. Vocês utilizarão um simples, baseado na frequência dos termos da consulta em cada documento (TF – *term frequency*) da coleção bem como na frequência inversa dos documentos (IDF – *inverse document frequency*). O componente IDF estima o quão bem um termo ajuda a discriminar os documentos entre relevantes e não relevantes: um termo que aparece em muitos documentos (valor de IDF baixo) não é um bom discriminador, enquanto que um termo que aparece em poucos documentos (IDF alto) é um bom discriminador dos mesmos.

Em outras palavras, dada uma consulta com q termos, t_1, t_2, ..., t_q, a relevância de um

documento i, *r(i)*, é computada como:

$$r(i) = \frac{1}{n_i} \sum_{j=1}^{q} w_j^i$$

onde n_i é o número de termos distintos do documento i e w^i_j é o peso do termo t_j no documento i, que é calculado como:

$$w_j^i = f_j^i \frac{\log(N)}{d_j}$$

onde f^i_j é o número de ocorrências do termo t_j no documento i, d_j é o número de documentos na coleção que contém o termo t_j e *N* é o número de documentos na coleção. Se o termo t_j não aparece no documento i, $f^i_j = 0$.

No exemplo dado acima, uma consulta “quer todos”:

- dois termos (q = 2): quer e todos
- há 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 f11 = 3 e f21 = 1. Além disto dj = 2.
  - w11 = 3*log(2)/2 = 1.5
  - w21 = 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 f12 = 0, f22 = 2 e dj = 1
  - w12 = 0*log(2)/1 = 0
  - w22 = 2*log(2)/1 = 2
- Logo, as relevâncias dos documentos pra esta consulta são:
  - r(1) = 1/7 *(1.5 + 0)
  - r(2) = 1/8 *(0.5 + 2)
  - você deve retornar a lista de documentos
    <doc2>
    <doc1>

3. O que será desenvolvido (enunciado)

Neste trabalho iremos indexar um conjunto de notícias. Para isso, você poderá baixar esse dataset (https://www.kaggle.com/rmisra/news-category-dataset/data#) que contém >200mil *headlines* e descrições de notícias, bem como sua URL de destino. Você colocará como um único documento o *headline* e a descrição. O sistema deverá:

1. Quando o usuário buscar por uma palavra-chave, você deverá encontrar as 20 notícias mais relevantes (em ordem) para a consulta do usuário e retornar o headline+abstract e a URL para acesso à notícia.
2. O usuário poderá informar no máximo 2 palavras para a busca. Neste caso, o sistema deverá contabilizar a relevância de cada documento considerando a soma dos pesos das palavras para cada documento. Os documentos que não possuem um dos termos também poderão aparecer no resultado da busca (caso fique entre os 20 mais relevantes).
3. Para indexação, o programador deverá implementar duas das estruturas abaixo e permitir que o programa seja instanciado com uma delas:
   a. Tabela Hash
   b. Árvore AVL
   c. SkipList
   d. TRIE
4. Será feito um experimento para analisar o desempenho e gasto de memória da aplicação ao utilizar cada uma das duas estruturas. Para tal, analise o tempo gasto para construção do índice invertido (usando o arquivo completo), gasto de memória após o índice criado, tempo gasto para processar 10.000 consultas aleatórias com 1 termo em cada abordagem e 10.000 consultas aleatórias com 2 termos em cada abordagem.

Será entregue um relatório descrevendo sua abordagem para desenvolvimento do sistema, a

arquitetura do sistema, detalhes de implementação mais relevantes (escolha da função de hash, implementação da estrutura de busca, forma de rankeamento dos resultados, etc). Não precisa discutir códigos-utilitários como leituras de arquivos, impressão, etc. A ideia aqui é discutir tua solução propriamente dita e os pontos do código que você gostaria de destacar por considerar que teve boas ideias pra melhoria de tempo e consumo. Sobre o experimento, interprete os dados gerados e fundamente a sua argumentação de acordo com o conteúdo discutido no curso e estudado por você. Faça gráficos mostrando o crescimento do custo da solução (ex: criando 10 *chunks* com 1000 consultas e coletando o tempo de inserção no índice a cada 20.000 registros).

4. Entrega

Este é um trabalho individual. O trabalho (relatório + código-fonte) serão enviados pelo Classroom. Pode implementar em C, C++, Java, Scala, PHP ou Python. Caso use outra linguagem, consulte o professor antes de implementar. Eu não tenho máquina com Windows (então C# tá fora de cogitação). O seu código tem que rodar em outros computadores, então cuidado com valores de diretórios *hardcoded* ou outras soluções pouco portáveis. Você pode ter um alto gasto de memória na tua solução, então cuidado com a linguagem que vai escolher (fazer em javascript, por exemplo, pode ser inviável).

Se preocupe em criar códigos legíveis. A nota será atribuída de acordo com o que eu entender do código. Então, quanto mais claro o código estiver, melhor.

5. Avaliação

O trabalho será avaliado em 4 dimensões: completude/corretude, qualidade da solução e relatório. Os pesos de cada dimensão serão definidos posteriormente. Dentre os critérios, estão:

- Completude/Corretude: o sistema deve ter todas as funcionalidades pedidas e retornar os resultados esperados. Caso o programa não funcione, a nota será zero nesta dimensão. O professor irá rodar o programa com seus arquivos de teste.
- Relatório bem redigido. Experimentos e discussão bem fundamentada.
- Qualidade da solução: Código documentado e boa prática de programação (o mínimo necessário de variáveis globais, variáveis e funções com nomes de fácil compreensão, soluções elegantes de programação, código bem modularizado, etc). Boas soluções para melhorar o desempenho do sistema (tempo e/ou espaço). Boa fundamentação para escolha das soluções de melhoria.

Vale ressaltar que pontualidade não será um critério de avaliação pois não poderei aceitar trabalhos fora do prazo. Lembro que a disciplina terá que ser fechada logo após a entrega do trabalho, então sequer dá tempo para entregas com atraso.