Busca de Artigos na Wikipedia com TF-IDF

Pedro Henrique Honorio Saito
122149392

Milton Leandro Salgado
122169279

Marcos Henrique Junqueira Muniz Barbi Silva
122133854

Relatório Parcial

Programação Concorrente (ICP-361) - 2025/2

1. Descrição do Problema

Desenvolver um sistema de busca léxica (*bag of words*) para recuperação de artigos da Wikipédia em inglês. Dado um conjunto de documentos e uma consulta textual, o sistema deve ranquear os documentos por similaridade textual usando um modelo vetorial clássico, o **TF-IDF** (*Term Frequency - Inverse Document Frequency*) e retornar os top k documentos mais semelhantes.

O TF-IDF é uma medida estatística que tem o intuito de indicar a importância de uma palavra de um documento em relação a uma coleção de documentos ou em um corpus linguístico. O valor TF-IDF de uma palavra aumenta proporcionalmente à medida que aumenta o número de ocorrências dela em um documento, no entanto, esse valor é equilibrado pela frequência da palavra no corpus.

O TF (*Term Frequency*), como o nome indica, quantifica a frequência de aparição de um termo em cada documento específico. Seu valor é tradicionalmente computado da seguinte forma:

$$\mathrm{TF}_{i,j} = \begin{cases} 1 + \log_2 f_{ij} & \text{se } f_{ij} > 0 \\ 0 & \text{c.c.} \end{cases}$$

em que f_{ij} representa a frequência absoluta do termo i no documento j, i denota o índice da palavra no *corpus* (conjunto de todas as palavras) e j o índice de um documento na coleção.

Por outro lado, o IDF (*Inverse Document Frequency*) expressa o grau de especificidade de um termo em relação ao conjunto de documentos, atribuindo maior peso a palavras raras e menor peso a termos comuns. Sua forma clássica é dada por:

$$\mathrm{IDF}_i = \log_2 \left(\frac{N}{n_i} \right)$$

onde N corresponde ao número total de documentos na coleção e n_i ao número de documentos em que o termo i ocorre.

Dessa forma, o **esquema de ponderação TF-IDF** combina ambas as duas métricas para atribuir a cada termo i em cada documento j um peso w_{ij} proporcional à sua relevância no documento e no *corpus*:

$$w_{ij} = \begin{cases} (1 + \log_2 f_{ij}) \times \log_2 \left(\frac{N}{n_i} \right) & \text{se } f_{ij} > 0 \\ 0 & c.c. \end{cases}$$

Por fim, cada documento é codificado como um vetor de pesos w_{ij} calculados sobre o *corpus*. A similaridade é então determinada por meio da **medida do cosseno**, que avalia o ângulo entre os vetores correspondentes.

Definido o modelo de ponderação adotado, o próximo passo consiste em aplicar o método ao conjunto de dados selecionado.

A base utilizada consiste em uma amostra correspondente a aproximadamente 2% dos artigos da Wikipédia em inglês, datada de 2017. A partir dessa base, o processamento segue o fluxo descrito abaixo:

1. **Coleta e ingestão dos artigos**: Obtenção do texto bruto inicialmente na linguagem de marcação *WikiText* e migração para SQlite.
2. **Pré-processamento dos documentos**:
   - **Normalização**: Conversão para caixa baixa, remoção de caracteres de controle e restrição ao conjunto ASCII.
   - **Tratamento de pontuação**: Manejo de apóstrofes, vírgulas e hífens para preservar a integridade semântica dos termos.
3. **Stemming/Lematização**: Aplicação do algoritmo *SnowballStemmer* para redução das palavras às suas raízes morfológicas.
4. **Construção do índice invertido**: Estrutura baseada em tabela hash, onde as chaves correspondem aos termos processados e os valores consistem em listas de *postings* no formato `[<doc_id>, <tf>]`.
5. **Pré-computar métricas de ponderação do TF-IDF**:
   - Cálculo do IDF para cada termo no *corpus*.
   - Geração dos vetores de representação dos documentos e suas normas.
   - Armazenamento tanto do índice invertido quanto das métricas em um arquivo binário.

Aqui está um exemplo da estrutura de índice de arquivo invertido sobre um conjunto arbitrário de palavras e seus documentos.

peã [3] →	1 f_{11} | 2 f_{12} | 3 f_{13}
caval [2] →	1 f_{21} | 4 f_{24}
pec [1] →	1 f_{31}
xadrez [2] →	1 f_{41} | 5 f_{45}
melhor [1] →	1 f_{51}
jog [3] →	1 f_{61} | 2 f_{62} | 5 f_{65}

Tabela 1: *Representação da estrutura de índice de arquivo invertido.*

Situado à esquerda, ao lado do termo, encontra-se o valor de n_i, que indica o número de documentos no qual a palavra ocorre. À direita, apresenta-se a lista de *postings*, na qual cada elemento contém o identificador do documento em que o termo aparece, bem como a frequência de ocorrência correspondente.

Obs. Em termos práticos, a estrutura de dados para a recuperação da informação pode ser uma matriz ou uma *hash* de *hashes*, isto é, um dicionário em que cada palavra pré-processada é uma chave, e o valor é outro dicionário com par (identificador do documento, frequência da palavra no documento).

Após essa etapa, o mesmo procedimento de pré-processamento deve ser aplicado à **consulta do usuário** a fim de transformá-la em um vetor de n-dimensional, em que n corresponde ao número total de termos do *corpus*. Como foi dito, a medição da similaridade é feita pelo cosseno por meio da fórmula abaixo:

$$sim\left(\vec{d}_j, \vec{q}\right) = \frac{\vec{d}_j \cdot \vec{q}}{\left|\vec{d}_j\right||\vec{q}|}$$

Denotando por $\vec{q}$ o vetor associado à consulta e por $\vec{d}_j$ o vetor previamente calculado do documento j, a obtenção dos top k documentos mais similares requer o cálculo da similaridade para todos os pares $\left(\vec{d}_j, \vec{q}\right)$.

Portanto, a solução concorrente permitiria melhor aproveitamento do potencial do processador da máquina para reduzir o tempo de pré-processamento da base de dados e da consulta do usuário por meio do paralelismo de dados.

2. Projeto da Solução Concorrente

A solução concorrente foi projetada para otimizar o **pré-processamento** e a **execução das consultas** no esquema de ponderação TF-IDF, empregando **paralelismo de dados** no processamento independente dos artigos quanto na consulta do usuário, além de **exclusão mútua** na unificação das *hashes*. O Diagrama 1 ilustra o fluxo geral de execução da solução paralela.

O conjunto de documentos é particionado em intervalos de tamanho fixo e uniforme, distribuídos entre as *threads*. Cada *thread* realiza o processamento completo do seu intervalo, incluindo:

- Leitura dos textos diretamente do banco embarcado `.sqlite`;
- Tokenização das palavras;
- Remoção de *stopwords* (palavras comuns sem relevância textual);
- Aplicação de *stemming*;
- Armazenamento dos resultados em uma *hash* local.

Após essa etapa, as *hashes* locais são unificadas (`merge`) em uma *hash* global por meio de mecanismos de exclusão mútua. As chaves resultantes compõem o vocabulário e permitem o cálculo do IDF. Em seguida, libera-se a memória das estruturas temporárias anteriores, e cada *thread* gera seus vetores TF-IDF para o intervalo de documentos e respectivas normas.

Com os vetores prontos, o sistema calcula a similaridade entre o vetor da consulta e os vetores de cada documento usando a medida do cosseno e retorna os k artigos mais semelhantes. Esse processo é totalmente paralelizado: tanto a normalização dos vetores quanto o cálculo das similaridades são distribuídos entre as threads, assegurando desempenho escalável e eficiência em consultas complexas com múltiplos termos.

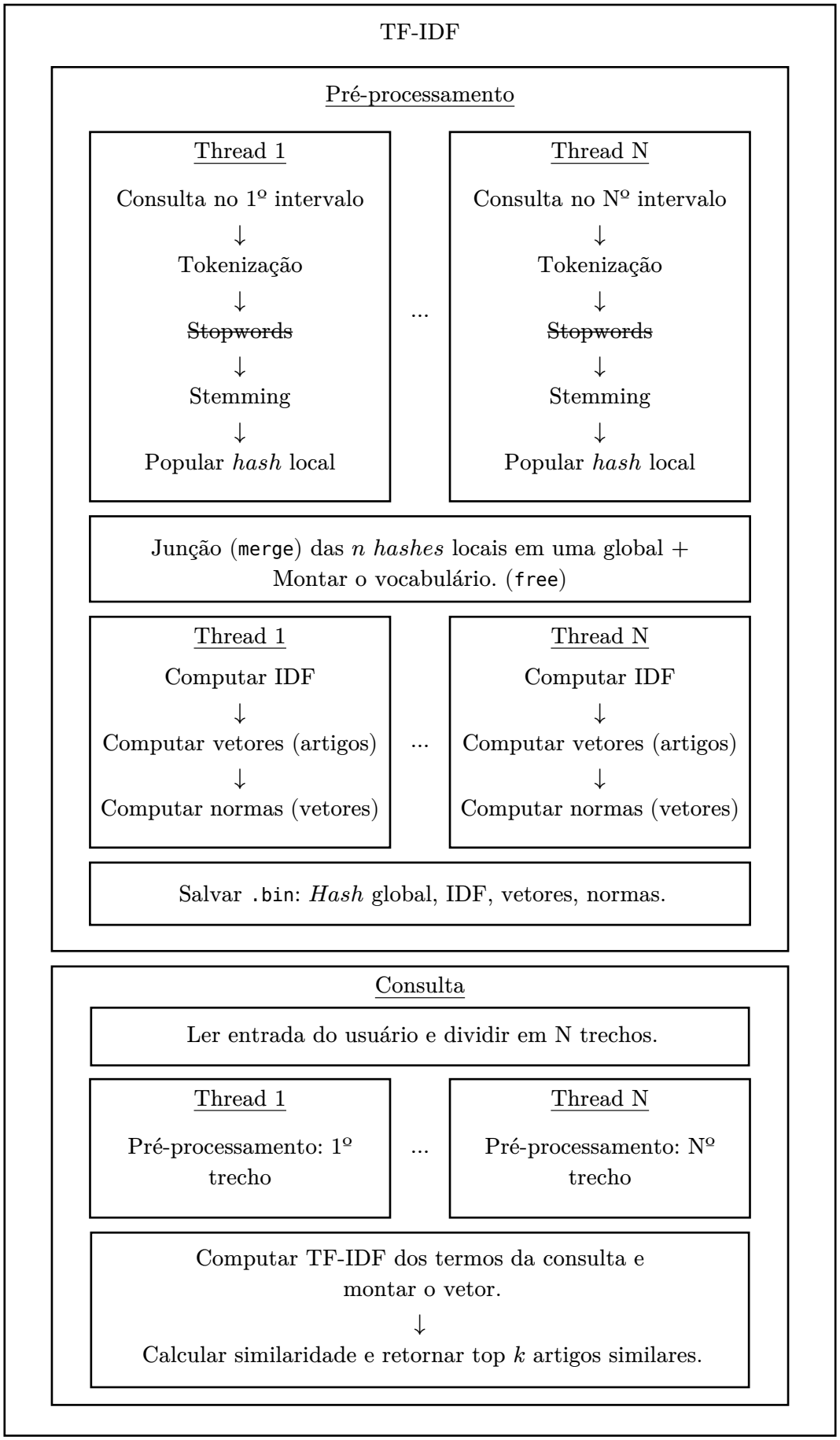

Diagrama 1: *Projeto da solução concorrente.*

3. Casos de teste de corretude e desempenho

Os casos de teste de corretude e de desempenho estão descritos abaixo. Primeiramente, vamos abordar os casos de teste

3.1. Casos de Corretude

A saída do programa será comparada, para várias entradas citadas mais abaixo, com o resultado obtido a partir de duas outras implementações do modelo TF-IDF:

- Implementação em Python do TF-IDF pelo Pedro Saito.
- Implementação do `scikit-learn` (`TfidfVectorizer`).

Os casos de teste serão compostos por um conjunto de consultas pré-definidas sobre diferentes áreas do conhecimento. O objetivo é avaliar a coerência dos artigos retornados em relação à busca e a proximidade com o retorno das outras implementações. As consultas de teste definidas foram as seguintes:

```txt
search open source linux frameworks for digital forensics and compiler
optimization tools supporting amd processors virtualization hypervisors and
graphical editors compatible with unix systems for software development and
analysis
```

Consulta 2: *Busca por ferramentas Open source em Linux, relacionadas à compiladores, hipervisores, e editores.*

```txt
search william shakespeare adaptations in film theatre and literature
including hamlet richard iii and comedy of errors exploring actors playwrights
stage societies and modern interpretations of elizabethan drama
```

Consulta 3: *Busca por adaptações de Shakespeare em filmes, teatro e literatura, incluindo Hamlet e Richard III.*

```txt
search japanese culture politics and sports including olympic athletes
surnames architecture festivals shinto shrines football teams and cinema
focusing on kyoto tokyo sapporo and historical figures like oda nobunaga
```

Consulta 4: *Busca por cultura japonesa, abrangendo esportes, política, arquitetura e cinema.*

```txt
search computer network protocols and technologies including ftp out-of-band
control routing gsm openbsc snmp smi tcp udp registered ports fiber channel
and satellite internet routers like cisco cleo
```

Consulta 5: *Busca por cultura japonesa, abrangendo esportes, política, arquitetura e cinema.*

```txt
search super mario related media and cultural references including video games
rom hacks music albums television series film editing and nintendo franchises
across entertainment and popular culture
```

Consulta 6: *Busca mídia e referências culturais ligadas a Super Mario, abrangendo jogos eletrônicos, séries, filmes e álbuns musiciais.*

As consultas do usuário serão fornecidas via linha de comando: `cat query.txt | ./app`.

3.2. Desempenho

O desempenho do programa será avaliado por partes. Primeiramente, avaliaremos a etapa de pré-processamento e, posteriormente, avaliaremos a consulta tendo como métrica principal o tempo de execução.

O programa possuirá uma flag `verbose` que permitirá a avaliação do desempenho entre as etapas desde passos individuais do pré-processamento até queries específicas.

Com relação aos testes, serão selecionados documentos de tamanho suficiente para avaliar o desempenho das consultas, como:

- The Complete Works of William Shakespeare (ASCII).
- 20 Newsgroups Dataset (ASCII).

A principal métrica analisada será o *tempo de execução*.

Serão conduzidos experimentos variando o número de *threads* e, possivelmente, o ambiente de execução, utilizando máquinas distintas pertencentes aos três integrantes do grupo juntamente de um servidor à parte se necessário.

3.2.1. Threads

Avaliaremos entre os seguintes números de *threads*:

- **Sequencial** (sem threads)
- **1 thread**.
- **2 threads**.
- **4 threads**.
- **8 threads**.
- **16 threads**.

3.2.2. Métricas de Avaliação

Para cada configuração, mediremos:

1. **Tempo de Pré-processamento**: Tempo total de execução da etapa de pré-processamento (sem contar consulta do usuário).
2. **Tempo de Processamento de Consultas**: Tempo total de execução para consultas simples (~20 termos) e mais complexas (+500 termos).
3. **Aceleração**:
   - Aceleração teórica: $S(n) = \frac{T_1}{T_n}$, onde T_1 é o tempo com 1 thread e T_n o tempo com n threads
   - Aceleração ideal vs. aceleração observada.
4. **Eficiência**: Eficiência de paralelização dada por $E(n) = \frac{S(n)}{n}$.

3.2.3. Variáveis Controladas

Para garantir a validade dos experimentos:

- Cada consulta será executada múltiplas vezes (mínimo de 10 repetições)
- Será calculada a média e o desvio padrão dos tempos
- O sistema operacional e processos em segundo plano serão padronizados

3.2.4. Análise Comparativa

Os resultados serão apresentados através de:

- Gráficos de aceleração vs. número de threads.

- Tabelas comparativas de tempo de execução.
- Identificação do número ótimo de threads para cada tipo de consulta.

Referências

1. Salton, G.: The SMART Retrieval System – Experiments in Automatic Document Processing. Prentice-Hall Inc., Englewood Cliffs, New Jersey (1971).
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley, New York (1999).