Projeto Prático 2 de Estrutura de Dados II

Verificador de Similaridade de Textos com Hash e AVL

Prof. Dr. Jean M. Laine

2025/2

Objetivo Geral

Desenvolver um sistema capaz de identificar o grau de similaridade entre documentos textuais, uma tarefa fundamental em diversas áreas da computação, como a **detecção de plágio acadêmico, a recomendação de notícias, a organização de grandes volumes de documentos e a otimização de motores de busca**.

Para isso, o projeto deve integrar conceitos de estruturas de dados avançadas, como **tabelas hash** e **árvores AVL**, aplicando-as de forma eficiente para resolver um problema real de análise de textos e detecção de similaridades.

Objetivos Específicos

- Implementar uma tabela hash para armazenar e representar os documentos processados.
- Implementar uma árvore AVL para manter ordenados os pares de documentos por grau de similaridade.
- Aplicar e comparar diferentes estratégias de dispersão e balanceamento.
- Analisar experimentalmente o desempenho das estruturas implementadas.

Descrição do Problema

Cada documento deve ser lido a partir de um arquivo de texto e processado para gerar um vocabulário significativo. **Este processamento, conhecido como normalização de texto, é obrigatório e deve incluir, no mínimo, os seguintes passos:**

- **Converter todo o texto para letras minúsculas.**
- **Remover toda pontuação e caracteres não-alfanuméricos.**
- **Realizar a tokenização (separação do texto em palavras ou "tokens").**

- **Remover "stop words"(palavras comuns como 'a', 'o', 'de', 'em', 'para', etc.) a partir de uma lista pré-definida para o português.**

Após a normalização, o sistema deve ser capaz de:

- Ler múltiplos documentos.
- Armazenar as palavras e suas frequências em uma tabela hash para cada documento.

Requisitos Técnicos

- A tabela hash deve ser implementada manualmente (sem uso de bibliotecas prontas).
- A função de dispersão deve ser definida e justificada pelo aluno. **O aluno deverá implementar duas funções de dispersão distintas** para fins de comparação no relatório técnico.
- A árvore AVL deve armazenar a similaridade como chave. **Atenção: é comum que múltiplos pares de documentos resultem no mesmo valor de similaridade.** Para tratar corretamente este caso, cada nó da árvore deve ser capaz de armazenar uma **lista de pares de documentos** associada àquela chave, em vez de um único par.
- A implementação do método de inserção na Árvore AVL deve ser modificada para **retornar um registro ou estrutura de dados que quantifique as rotações** (Simples Esquerda/Direita, Dupla Esquerda/Direita) realizadas durante a operação.
- O cálculo de similaridade deve ser baseado em uma métrica escolhida e justificada pelo aluno (exemplo: Cosseno, Jaccard, Dice, etc.).

Exemplo Ilustrativo (Não Vinculante)

O exemplo a seguir tem fins apenas didáticos e serve para ilustrar o conceito geral do projeto. O formato exato das estruturas, dos cálculos e do código **deve ser definido pelo aluno**.

Documentos de Exemplo

```
doc1.txt: "estrutura de dados é essencial em ciência da computação"
doc2.txt: "dados estruturados são essenciais para a ciência"
```

Após tokenização e remoção de palavras irrelevantes:

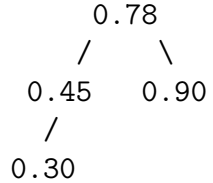
```
doc1 → { "estrutura":1, "dados":1, "essencial":1, "ciência":1, "computação":1 }
doc2 → { "dados":1, "estruturados":1, "essenciais":1, "ciência":1 }
```

Cálculo de Similaridade (Exemplo Conceitual)

A similaridade entre os dois documentos pode ser estimada com base na sobreposição dos termos, utilizando uma métrica como Cosseno, Jaccard ou Dice. Por exemplo, uma similaridade de 0.45 indicaria sobreposição parcial entre os vocabulários — sugerindo que os textos tratam de um mesmo tema, mas não são cópias.

Uso da AVL

Após calcular as similaridades entre todos os pares de documentos, o sistema deve armazenar os resultados em uma árvore AVL ordenada pela similaridade.



Cada nó pode conter uma lista de pares de documentos com a pontuação de similaridade, permitindo consultas eficientes.

Observação: O exemplo é apenas ilustrativo; o aluno deve adaptar e justificar todas as decisões de representação, cálculo e balanceamento.

Relatório Técnico Obrigatório

O relatório deve seguir a estrutura abaixo. As respostas para as questões de análise devem ser integradas de forma coesa dentro das seções indicadas, evitando um formato de questionário.

1. **Introdução:** Contextualização sobre o problema de comparação de textos, a relevância da detecção de similaridade e os objetivos do trabalho.
2. **Metodologia:** Explicação detalhada das estruturas de dados e algoritmos utilizados.
    - Descrição da Tabela Hash implementada, incluindo as **duas funções de dispersão** escolhidas e a estratégia de tratamento de colisões.
    - Descrição da Árvore AVL, com foco na lógica de balanceamento, nas rotações e na **forma como a contagem de rotações foi implementada**.
    - Justificativa para a escolha da métrica de similaridade (Cosseno, Jaccard, etc.).
3. **Descrição do Algoritmo:** Apresentação do funcionamento geral do sistema, o fluxo de dados e, se necessário, pseudocódigos dos algoritmos mais complexos (ex: inserção na AVL).
4. **Análise Experimental:** Apresentação e discussão dos testes de desempenho realizados.

- Gráficos e tabelas mostrando o tempo médio de execução com diferentes volumes de documentos.
- Comparação de desempenho da busca na AVL com uma estrutura mais simples (ex: busca em lista ordenada).
- **Nesta seção, responda obrigatoriamente:**
  - Apresente um gráfico ou tabela comparando a distribuição de chaves (ex: colisões por bucket) para as **duas funções de hash implementadas**. Discuta qual foi mais eficaz para o seu conjunto de dados.
  - Apresente os dados coletados sobre o **número de rotações** (simples e duplas) durante a inserção dos resultados na AVL. Discuta o que esses números indicam sobre a ordem de inserção dos dados.

5. **Resultados e Discussão:** Análise dos resultados de similaridade obtidos e interpretação crítica dos mesmos.
   - Apresentação dos pares de documentos com maior similaridade encontrados em seus testes.
   - **Nesta seção, discuta obrigatoriamente:**
     - Qual seria o impacto (em termos de complexidade e adequação ao problema) de substituir a Árvore AVL por outra estrutura, como uma *heap*?
     - Como as etapas de pré-processamento de texto (normalização, remoção de *stop words*, etc.) ajudaram a evitar falsos positivos de similaridade? Que outros passos poderiam ser tomados?

6. **Conclusões:** Considerações finais sobre o projeto, o aprendizado obtido e possíveis melhorias futuras.
   - **Nesta seção, reflita obrigatoriamente sobre:**
     - Quais foram os principais desafios técnicos e conceituais encontrados ao implementar a função hash e o balanceamento da Árvore AVL?
     - Descreva em detalhes o bug mais desafiador que você encontrou durante a implementação. Explique qual era o sintoma do erro, como você investigou a causa e qual foi a solução aplicada ao código.

Critérios de Avaliação

- Implementação correta e funcional das estruturas.
- Clareza e coerência do relatório técnico.
- Originalidade das respostas e análises.
- Testes de desempenho e profundidade das discussões.
- Evidências de autoria e domínio do conteúdo.

Penalizações:

- Projeto copiado ou gerado automaticamente sem explicação: nota 0.
- Programa que não compila ou não executa: nota 0.
- Relatório sem referências, prints ou respostas analíticas: até -2 pontos.
- Entrega fora do formato especificado: -1 ponto.

Execução e Testes Automatizados

O programa deve ser executável via linha de comando (terminal) e aceitar os seguintes formatos:

```
java Main <diretorio_documentos> <limiar> <modo> [argumentos_opcionais]
```

onde:

- `<diretorio_documentos>` — caminho da pasta contendo os arquivos de texto (ex: `./documentos/`);
- `<limiar>` — valor entre 0 e 1 indicando o nível mínimo de similaridade a ser exibido;
- `<modo>` — define o comportamento do programa:
  - `lista` — exibe todos os pares com similaridade acima do limiar;
  - `topK` — exibe apenas os K pares mais semelhantes (o valor de K deve ser informado após o modo);
  - `busca` — compara dois arquivos específicos, informados nos argumentos seguintes.

Exemplos de Execução

Modo lista:

```
java Main ./documentos 0.75 lista
```

Modo topK:

```
java Main ./documentos 0.8 topK 5
```

Modo busca:

```
java Main ./documentos 0.0 busca doc1.txt doc4.txt
```

Neste último modo, o valor do limiar pode ser 0.0, pois será ignorado. O programa deve exibir apenas a similaridade entre os dois documentos especificados.

Saída esperada:

```
=== VERIFICADOR DE SIMILARIDADE DE TEXTOS ===
Comparando: doc1.txt <-> doc4.txt
Similaridade calculada: 0.67
Métrica utilizada: Cosseno
```

Formato de Saída Esperado

A saída padrão deve ser exibida no terminal e também gravada em um arquivo de log (`resultado.txt`) no mesmo diretório de execução. O formato da saída deve seguir o modelo abaixo:

```
=== VERIFICADOR DE SIMILARIDADE DE TEXTOS ===
Total de documentos processados: 5
Total de pares comparados: 10
Função hash utilizada: hashMultiplicativo
Métrica de similaridade: Cosseno

Pares com similaridade >= 0.75:
---------------------------------
doc1.txt  <->  doc2.txt  = 0.82
doc3.txt  <->  doc4.txt  = 0.79

Pares com menor similaridade:
---------------------------------
doc1.txt  <->  doc5.txt  = 0.12
```

Requisitos obrigatórios:

- O programa deve aceitar argumentos pela linha de comando sem interação manual.
- Todos os resultados devem ser impressos no terminal e gravados em arquivo.
- O nome do arquivo de saída deve ser sempre `resultado.txt`.
- O programa deve funcionar em ambiente padrão (JDK 17 ou superior).

Arquitetura Recomendada

Para promover um design de software modular e de fácil manutenção, recomenda-se que o projeto seja estruturado nas seguintes classes. A descrição abaixo detalha as principais responsabilidades de cada uma.

- `Main`
  - **Responsabilidade Principal:** Orquestrar a execução completa do programa.
  - **Detalhes:** Esta classe deve ser o ponto de entrada do sistema. Suas tarefas incluem: ler e validar os argumentos da linha de comando, coordenar a leitura dos arquivos, instanciar os objetos `Documento`, disparar o processo de comparação de todos os pares, inserir os resultados na árvore AVL e, por fim, invocar os métodos para exibir os resultados no terminal e gravá-los no arquivo `resultado.txt`.
- `Documento`
  - **Responsabilidade Principal:** Representar um único arquivo de texto e seu conteúdo processado.
  - **Detalhes:** Recomenda-se que esta classe encapsule toda a lógica de processamento de um único arquivo. Ela deve conter atributos como o nome do arquivo e sua própria **Tabela Hash interna** para mapear cada palavra única à sua frequência. Seus métodos devem realizar a leitura do arquivo, a normalização do texto e popular a tabela hash.
- `HashTable`
  - **Responsabilidade Principal:** Implementação manual de uma tabela hash genérica.
  - **Detalhes:** Deve ser uma implementação do zero, sem usar o `HashMap` do Java. A classe deve gerenciar um array interno e as **duas funções de dispersão (hash)** criadas pelo aluno. Métodos essenciais incluem `put(chave, valor)` e `get(chave)`. O tratamento de colisões é uma parte crítica desta implementação.
- `AVLTree`
  - **Responsabilidade Principal:** Implementação manual de uma árvore AVL.
  - **Detalhes:** A chave de cada nó será a similaridade (`double`), e o valor armazenado será uma **lista de objetos** `Resultado`. A implementação deve conter a lógica de inserção que calcula o fator de balanceamento e realiza as **rotações simples e duplas**. O método de inserção deve ser modificado para permitir a **contagem de rotações**. A classe deve prover métodos de travessia para listar os pares ordenadamente.
- `ComparadorDeDocumentos`
  - **Responsabilidade Principal:** Isolar a lógica de cálculo de similaridade.

  - **Detalhes:** Esta classe deve conter um ou mais métodos estáticos ou de instância que recebem dois objetos `Documento` como entrada e retornam um `double` representando a similaridade entre eles. É aqui que a métrica escolhida (Cosseno, Jaccard, etc.) será implementada. Separar essa lógica facilita testes e futuras extensões (ex: adicionar novas métricas).

- `Resultado`
  - **Responsabilidade Principal:** Estrutura de dados simples para armazenar o resultado de uma comparação.
  - **Detalhes:** Uma classe simples, similar a uma *struct*, contendo apenas atributos para guardar os nomes dos dois documentos comparados e o valor da similaridade calculada. Objetos desta classe serão armazenados nas listas dentro dos nós da árvore AVL.

Testes Automatizados pelo Professor

Durante a correção, o professor executará o programa com diferentes conjuntos de testes (documentos e limiares). O aluno deve garantir que:

- O programa compile e execute corretamente com o comando:

```
javac *.java && java Main ./documentos 0.7 lista
```

- Todos os arquivos de teste sejam processados sem necessidade de ajustes no código.
- As saídas sejam consistentes e apresentem formatação semelhante ao modelo acima.

O não cumprimento dessas regras poderá resultar em descontos na nota ou desclassificação do projeto. **Dica:** antes da entrega, o aluno pode testar seu programa usando scripts de automação (`.sh` ou `.bat`) com diferentes parâmetros para garantir robustez.

Entrega

- O trabalho deve ser entregue via plataforma oficial da disciplina, em um único arquivo compactado (.zip) contendo:
  - Código-fonte completo;
  - Relatório em PDF;
  - Pasta `documentos/` com os textos usados nos testes.
- Nenhum outro meio de envio será aceito.

Extensões Opcionais

Aviso: As extensões a seguir são totalmente opcionais e servem para alunos que desejam aprofundar seus conhecimentos. Recomenda-se fortemente que os alunos só iniciem qualquer um desses itens após terem uma versão completa, funcional e testada de todos os requisitos obrigatórios do projeto.

- **Suporte a diferentes idiomas**

  ↪ Adapte o sistema para que ele possa processar textos em outros idiomas (como inglês ou espanhol), permitindo, por exemplo, que o usuário especifique qual lista de *stop words* deve ser utilizada.

- **Implementação de TF-IDF**

  ↪ Substitua a contagem de frequência simples pela métrica TF-IDF (Term Frequency-Inverse Document Frequency). Isso resultará em uma análise de similaridade mais sofisticada, que valoriza palavras que são importantes para um documento, mas raras na coleção geral. (Understanding TF-IDF for machine learning)

- **Exportação dos resultados em formato CSV ou PDF**

  ↪ Além do arquivo `resultado.txt`, permita que o programa gere um arquivo `resultados.csv` (valores separados por vírgula), que pode ser facilmente importado em planilhas para análise de dados. A exportação para PDF é uma alternativa mais complexa.

- **Visualização da AVL gerada**

  ↪ Crie uma funcionalidade que gere uma representação visual da árvore AVL final. Isso pode ser feito gerando um arquivo no formato DOT (para ser lido por ferramentas como o Graphviz) ou imprimindo uma estrutura em formato de texto no console, ajudando a entender o balanceamento da árvore.

Normas para Realização e Entrega

Para a organização e avaliação deste trabalho, sigam atentamente as seguintes normas:

- **Grupos:** O trabalho poderá ser realizado em **grupos de 2 até 3 (três) alunos**. Não serão aceitos trabalhos individuais.

- **Apresentação Obrigatória:** A apresentação do trabalho é **obrigatória** para a atribuição da nota. Todos os integrantes do grupo devem estar presentes e aptos a explicar qualquer parte do código e dos conceitos aplicados. A ausência de qualquer integrante no momento da apresentação acarretará nota zero no projeto para o aluno ausente.

- Durante a apresentação, cada aluno deve demonstrar domínio sobre o código-fonte, sendo capaz de:

  - Explicar a lógica de funcionamento de qualquer parte da implementação;
  - Justificar as decisões de estrutura e algoritmo utilizadas;
  - Realizar pequenas alterações ou correções solicitadas pelo professor no momento da apresentação, como forma de verificação de autoria.

- Caso o aluno ou o grupo não demonstre compreensão adequada sobre o funcionamento do código, ou revele desconhecimento sobre a implementação, o projeto será anulado e receberá nota zero.

- **Identificação no Código:** O nome completo e o RA de todos os integrantes do grupo devem constar em um **bloco de comentários no cabeçalho** do arquivo de código-fonte principal (`Main.java`).

- **Documentação:** O código-fonte deve ser claro e **bem documentado com comentários**, explicando as partes mais importantes da sua lógica, especialmente as seções que tratam da **função de dispersão, tratamento de colisões e das rotações da árvore AVL**.

- **Submissão da Tarefa:** A entrega do projeto (arquivos `.java`) deve ser realizada por **apenas um integrante** do grupo na plataforma online da disciplina. Envios por outros meios não serão considerados válidos.

- **Plágio:** O trabalho deve ser integralmente desenvolvido pelo grupo. Casos de plágio ou uso não autorizado de ferramentas automáticas resultarão em nota zero e poderão ensejar encaminhamento à coordenação.

Orientações sobre o Uso de Ferramentas de Inteligência Artificial (LLMs)

Ferramentas de Inteligência Artificial generativa (como *ChatGPT*, *Copilot*, *Gemini*, entre outras) podem ser recursos úteis para estudo, esclarecimento de dúvidas e aprimoramento do raciocínio lógico e técnico, desde que utilizadas de forma ética e consciente. Nas atividades desta disciplina, o uso dessas ferramentas deve apoiar o aprendizado, e não substituir o trabalho autoral do estudante. O objetivo é que o aluno compreenda, projete e implemente as soluções por conta própria.

É permitido:

- Utilizar LLMs para revisar conceitos teóricos (ex.: "o que é uma árvore AVL?");
- Pedir explicações ou exemplos genéricos para compreender um tema;
- Consultar sobre boas práticas de programação ou convenções de código;
- Obter ajuda para depurar (*debugar*) erros após tentativa prévia de resolução;
- Discutir ideias e caminhos possíveis, sem solicitar o código completo da solução.

Não é permitido:

- Solicitar à IA que resolva ou implemente diretamente a atividade ou o projeto;
- Submeter códigos, textos ou relatórios gerados parcial ou totalmente pela IA sem autoria clara;
- Entregar soluções que o aluno não saiba explicar, justificar ou modificar durante a avaliação;
- Utilizar a IA para traduzir automaticamente instruções e gerar código sem compreender seu funcionamento.

Importante: A utilização de códigos prontos, trechos obtidos de terceiros ou gerados por ferramentas de Inteligência Artificial (LLMs, como ChatGPT, Copilot, Gemini, etc.), sem o devido entendimento e autoria, configura fraude acadêmica.

Referências Bibliográficas Sugeridas

Para aprofundar os conceitos teóricos e práticos necessários para a realização deste projeto, recomenda-se fortemente a consulta das seguintes obras de referência. Elas fornecem a base para a implementação das estruturas de dados, o cálculo de similaridade e a análise de algoritmos.

Estruturas de Dados e Algoritmos (Base Teórica)

- **CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C.** ***Algoritmos: Teoria e Prática.*** 3ª Edição, Editora Campus/Elsevier, 2012.
  - ↪ Considerada a principal referência na área. Essencial para o estudo aprofundado de tabelas hash, árvores rubro-negras (conceitualmente próximas das AVLs) e, principalmente, para a análise de complexidade de algoritmos exigida no relatório.
- **SEDGEWICK, R.; WAYNE, K.** ***Algorithms.*** 4th Edition, Addison-Wesley Professional, 2011.
  - ↪ Obra famosa por sua abordagem prática e didática, com exemplos de implementação claros e diretos, muitos dos quais disponíveis em Java. Excelente para guiar a implementação da Tabela Hash e da Árvore de Busca Binária.
- **GOODRICH, M. T.; TAMASSIA, R.; GOLDWASSER, M. H.** ***Data Structures and Algorithms in Java.*** 6th Edition, Wiley, 2014.
  - ↪ Um livro focado especificamente na implementação das estruturas de dados utilizando a linguagem Java. É um guia prático perfeito para este projeto, cobrindo em detalhes a implementação de árvores AVL e tabelas hash.

Recuperação de Informação e Processamento de Texto

- **MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H.** ***Introduction to Information Retrieval.*** Cambridge University Press, 2008.

  ↪ A referência definitiva para entender a parte de similaridade de textos. Cobre em detalhes o modelo de espaço vetorial, normalização de texto (tokenização, stop words) e métricas como a similaridade de cosseno. O capítulo 6 é particularmente relevante.

- **JURAFSKY, D.; MARTIN, J. H.** ***Speech and Language Processing.*** 3rd Edition, Prentice Hall, 2020.

  ↪ Uma obra mais ampla sobre Processamento de Linguagem Natural que fornece um contexto robusto para as técnicas de pré-processamento e representação de texto utilizadas no projeto.

Boas Práticas de Programação em Java

- **BLOCH, J.** ***Java Efetivo: as Melhores Práticas Para a Plataforma Java.*** 3ª Edição, Editora Alta Books, 2019.

  ↪ Leitura opcional, mas altamente recomendada para os alunos que desejam não apenas entregar um código funcional, mas um código de alta qualidade. Oferece conselhos práticos sobre como escrever software robusto, eficiente e de fácil manutenção em Java.