Projeto 1 - Estruturas de Dados (ADS)

Título do Projeto: HashSimPy - Verificador de Similaridade Textual em Python com MinHashing

Equipe: até 5 Alunos

1. Ideia Central

Desenvolver uma ferramenta em **Python**, via linha de comando (CLI), que compara documentos e calcula um índice de similaridade usando **MinHashing**. O foco será na implementação clara do algoritmo e no uso eficiente de estruturas de dados Python (dicionários e conjuntos) para lidar com shingles e assinaturas MinHash.

Estruturas de Dados Chave:

- **Dicionários (dict):** Para mapear shingles para IDs, armazenar contagens, e potencialmente para construir as assinaturas MinHash de forma eficiente.
- **Conjuntos (set):** Para armazenar os shingles únicos de cada documento e facilitar operações como a união e interseção (base do Índice de Jaccard).

2. Aplicação e Relevância (Mantida)

A ferramenta HashSimPy mantém suas aplicações em detecção de similaridade (plágio simplificado, agrupamento, duplicatas) e serve como um excelente exemplo prático do poder do hashing e das estruturas de dados associadas para resolver problemas de comparação em larga escala de forma eficiente.

3. Tecnologia

- **Linguagem de Programação: Python 3.x**
- **Bibliotecas Padrão:**
    - hashlib: Para acesso a funções de hash (ex: SHA-1, MD5).
    - re: Para expressões regulares úteis no pré-processamento de texto.
    - collections (Opcional): Pode oferecer estruturas úteis como defaultdict.
- **Ambiente:** Linha de comando (Terminal/Shell).

4. Cronograma de Execução Otimizado (5 Semanas - Sem Testes Formais)

- **Semana 1: Pesquisa, Planejamento e Estrutura Inicial**
    - **Atividades:**

        - Estudo: Entendimento dos conceitos (Hashing, Shingling, Jaccard, MinHashing). Foco na lógica do algoritmo.
        - Definição da Arquitetura: Módulos Python (.py files) para cada etapa (leitura, pré-processamento, shingling, MinHashing, comparação, interface CLI).
        - Decisões de Implementação: Escolha do tamanho do shingle (k), número de funções hash para MinHashing.
        - Divisão de Tarefas: Quem implementará qual(is) módulo(s).
        - Configuração do Ambiente: Repositório Git/GitHub. Estrutura básica de pastas e arquivos no repositório.
    - **Entrega Semanal 1:** Documento com resumo dos conceitos, diagrama de arquitetura (módulos e fluxo), decisões de implementação (k, nº hashes), plano de divisão de tarefas, link do repositório com estrutura inicial.
- **Semana 2: Implementação do Pré-processamento e Shingling**
    - **Atividades:**
        - Codificar funções para ler arquivos de texto (.txt).
        - Implementar o módulo de pré-processamento (lowercase, remoção de pontuação/números via re).
        - Implementar o módulo de shingling (gerar conjuntos de shingles de tamanho k).
        - **Verificação Manual:** Executar as funções com textos de exemplo simples e verificar manualmente se a saída (texto processado, conjunto de shingles) está correta.
        - Adicionar comentários claros ao código explicando a lógica.
    - **Entrega Semanal 2:** Módulos Python (.py) para leitura, pré-processamento e shingling. Relatório de progresso descrevendo a implementação e os resultados da verificação manual.
- **Semana 3: Implementação do MinHashing e Comparação**
    - **Atividades:**
        - Implementar a geração das múltiplas "funções" de hash (ex: usando hashlib com diferentes *seeds* ou combinações).

        - Implementar o módulo MinHashing: Calcular as assinaturas (listas/arrays de Mínimos Hashes) para cada documento processado. Usar dicionários/conjuntos eficientemente.
        - Implementar a função de comparação que estima o Índice de Jaccard a partir das assinaturas.
        - **Verificação Manual:** Criar documentos de teste (idênticos, muito similares, diferentes) e executar o processo de MinHashing/Comparação. Verificar se os índices de similaridade calculados fazem sentido (ex: próximos de 1.0 para idênticos, próximos de 0.0 para muito diferentes).
        - Integrar os módulos (chamar pré-processamento -> shingling -> MinHashing -> Comparação).
    - **Entrega Semanal 3:** Módulos Python para MinHashing e Comparação, integrados com os anteriores. Relatório de progresso com resultados da verificação manual das similaridades.
- **Semana 4: Interface CLI e Integração Final**
    - **Atividades:**
        - Desenvolver a interface de linha de comando (CLI): Usar argparse (recomendado) ou input() para receber os nomes dos arquivos a comparar.
        - Formatar a saída: Apresentar os resultados da similaridade de forma clara para o usuário.
        - Integrar todos os módulos no script principal da CLI.
        - Adicionar tratamento básico de erros (arquivo não existe, etc.).
        - **Experimentação:** Testar a ferramenta completa com diferentes pares de arquivos e observar os resultados. Ajustar parâmetros (k, nº hashes) se necessário e observar o impacto.
        - Iniciar a documentação final (README.md no Git).
    - **Entrega Semanal 4:** Script Python principal da ferramenta CLI funcional. Rascunho do README.md explicando como usar a ferramenta.
- **Semana 5: Refinamento, Documentação e Preparação da Apresentação**
    - **Atividades:**

        - Refinamento do Código: Melhorar clareza, adicionar mais comentários explicativos onde necessário, garantir que a estrutura está organizada.
        - **Verificação Final:** Realizar uma rodada final de verificações manuais com um conjunto mais variado de textos para garantir a robustez da lógica principal.
        - Finalizar a Documentação: Detalhar o algoritmo no README, explicar as escolhas de implementação, incluir exemplos de uso.
        - Preparar a Apresentação: Criar slides (problema, solução teórica, implementação Python, estruturas de dados usadas, resultados da experimentação, conclusões/aprendizados).
        - Garantir que todo o código esteja no repositório Git.
    - **Entrega Final:** Código-fonte final completo e comentado em Python, documentação final (README.md), slides da apresentação, ferramenta CLI funcional.

5. Critérios de Avaliação (Ajustados)

- **Funcionalidade:** A ferramenta executa e produz resultados de similaridade coerentes para diferentes entradas? O fluxo principal funciona?
- **Aplicação de Conceitos:** O algoritmo MinHashing está implementado corretamente? As estruturas de dados Python (dict, set) foram usadas de forma apropriada e eficiente?
- **Qualidade do Código:** Código Python claro, bem organizado (módulos), comentado adequadamente (docstrings, comentários de linha). Segue boas práticas básicas de Python (PEP 8)?
- **Documentação:** O README está claro, explicando o projeto, como usá-lo e as decisões tomadas?
- **Trabalho em Equipe:** Colaboração via Git, cumprimento do cronograma.
- **Apresentação:** Clareza na explicação, demonstração funcional.