Estruturas de Dados

Trabalho AV3 - Código de Autenticação Textual (simulado)

Professor: Prof. Msc. Paulo Cirillo Souza Barbosa

Discussão inicial sobre o trabalho.

Em termos gerais, a autenticação digital (ou autenticação eletrônica) é um processo por meio do qual se garante a identificação correta dos autores em um documento expedido de modo eletrônico. Essa ferramenta consiste em um mecanismo capaz de assegurar a veracidade da identidade do signatário de um documento, o que é fundamental para proporcionar a segurança jurídica em procedimentos legais de diferentes naturezas, como petições, transações comerciais e acordos. Além de garantir que o arquivo foi gerado por uma pessoa autorizada e devidamente identificada, a autenticação digital também confirma a origem e a integridade de determinado documento. Dessa forma, é possível visualizar se houve alguma alteração no documento eletrônico, por menor que ela tenha sido, o que o resguarda de eventuais interceptações que comprometam o seu conteúdo. Por fim, a autenticação digital é ainda uma ferramenta útil para autorizar os leitores finais de um documento, já que só terão acesso ao arquivo após passarem por um processo de confirmação de identidade. Essa característica é muito útil já que permite não apenas assegurar a veracidade do emissor do documento, mas também controlar quem serão os destinatários do arquivo enviado, mantendo o sigilo das informações a outras pessoas.

1) Descrição do Projeto

O projeto a ser desenvolvido é uma versão de autenticador digital, ou seja, um aplicativo para verificação da autenticidade de documentos. O aplicativo deverá gerar códigos de autenticação para documentos textuais.

A geração do código de autenticação do documento deverá ser feita com o auxílio de um conjunto de estruturas de dados. Cada estrutura tem um papel específico na geração de tal código e segue a seguinte sequência de passos:

1. Realizar a leitura de um arquivo textual, em arquivo .TXT;
2. Cada linha do texto é consumida individualmente e as palavras presentes nessa linha textual são inseridas em uma lista **dinâmica**. Nesse caso, as palavras são obtidas ao identificar os espaços em branco entre textos.
3. Deve-se em seguida, acessar os elementos da lista no sentido reverso (do final até o início) e inserir tais elementos um a um em uma árvore AVL. Para realizar as comparações lexicográficas Strings, pode-se utilizar o método **compareToIgnoreCase()**. Além disso, deve-se desconsiderar quaisquer palavras duplicadas na String em questão.
4. Após o cadastro de todas as Strings para aquela linha, deve-se inserir a árvore AVL em uma pilha.
5. Os procedimentos devem ser repetidos para todas as linhas no texto.
6. Após a pilha de árvores ser construída, deve-se desempilhar as árvores de uma em uma, e gerar o código hash para aquela árvore desempilhada.
7. Cada hash gerado de cada uma das árvores deve ser concatenado e separados por uma quebra de linha.

O fluxo de construção da pilha de árvores pode ser visualizado na Figura 1.

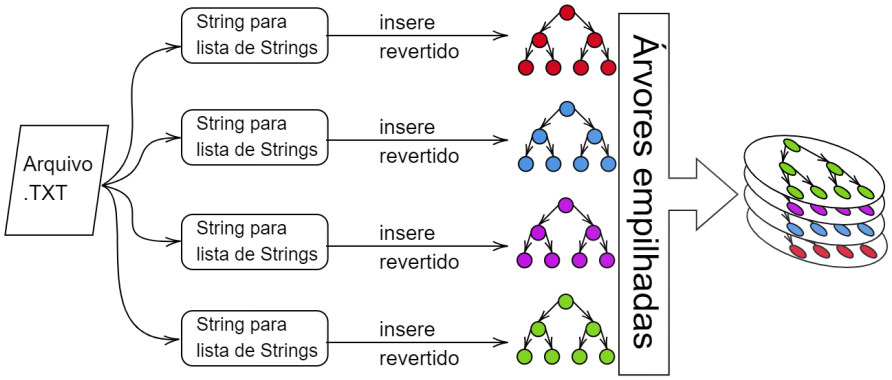


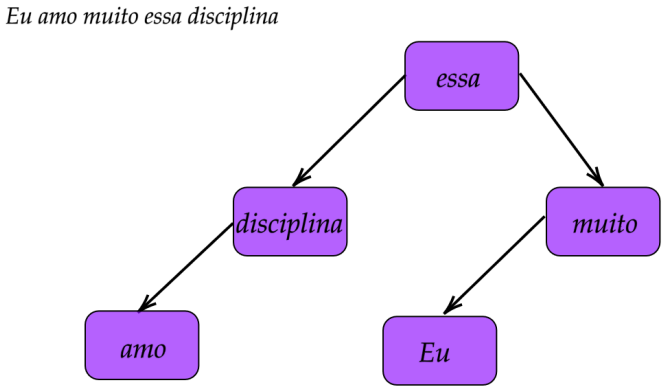
Figura 1: Fluxograma de construção da pilha de árvores

Para a construção do hash para uma única linha textual presente em um arquivo, considere o exemplo em a linha lida tem a seguinte String: "*Eu amo muito essa disciplina!*". A sequência de passos para construção da árvore é a seguinte:

1. Divida a String em uma lista de substrings separadas por espaço, obtendo: ["Eu", "amo","muito", "essa", "disciplina!"]

2. Em seguida, faça a inserção dos itens na árvore balanceada no sentido reverso (do fim até o início, ou seja, de "disciplina"até "Eu") de que foram inseridos na lista. Para o balanceamento, considere o uso da análise lexicográfica das Strings por meio do método **compareToIgnoreCase()**.

Uma observação **importante** é que como se trata de uma árvore binária, strings duplicatas não deverão ser inseridas na estrutura, neste caso quando o método **compareToIgnoreCase()** retornar 0, as Strings são iguais. Segue exemplo da árvore construída após a inserção da String exemplo:



Obs: Note que esta árvore já se encontra balanceada.

Como passo final, após a inserção dos elementos na árvore, deve-se realizar a travessia dos nós e computar o *hash* da árvore através da lógica: os nós pais armazenam o *hash* da combinação dos *hash* de seus filhos. Por exemplo, se A e B são filhos de um nó P, então o *hash* de P será $h(h(A) + h(B) + h(P))$. Assim, o hash computado para aquela String, encontra-se armazenado no nó raiz.

Assim, o hash final encontra-se no nó raíz da árvore AVL e para este exemplo específico, pode ser visualizado na figura 2:

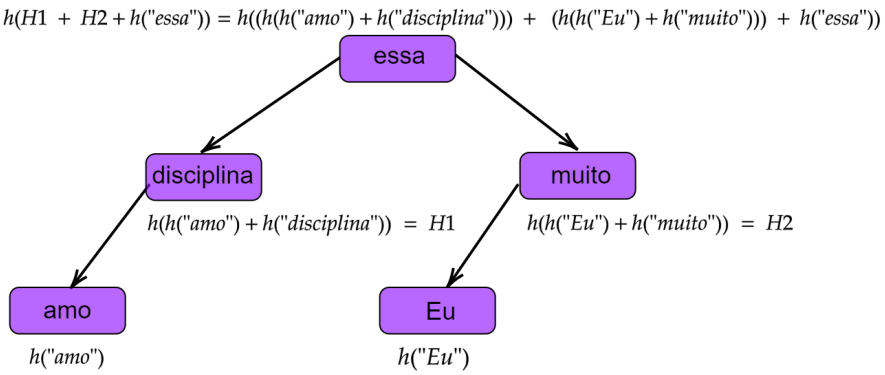


Figura 2: Exemplo geração de hashes

Destaca-se que quando não há filhos a esquerda ou direita, não se deve aplicar um hash para *null*. Para esse caso específico, o hash final gerado pelo nó raiz é: *f44040deec0de6e3100f7ff772bb7a25646e71ed*

Novamente, é importante destacar que o processo em questão, é operado em apenas uma linha textual. Desta maneira, caso seja de desejo computar o autenticador para um texto com múltiplas linhas, necessário realizar o procedimento descrito para cada linha.

Requisitos

Os requisitos do projeto são:

1. Deve-se receber como entrada, um arquivo de texto (formato .TXT), podendo conter **múltiplas linhas**.
2. Gerar como saída o código de autenticação do documento.
3. A geração do código de autenticação do documento deverá ser realizada conforme procedimento descrito na seção 1.
4. Para cada String presente no texto, deve-se criar uma árvore com a estrutura descrita na seção 1.
5. Deve-se armazenar cada uma dessas árvores em uma pilha e gerar o código de cada linha de texto, ao desempilhar cada árvore e realizar a travessia computando cada **hash**. O código gerado para cada árvore deve ser impresso separado por quebra de linha.
6. Os *hashes* deverão ser gerados pela função de dispersão criptográfica SHA-1. Um gerador online e o código-fonte da função pode ser encontrado no site SHA-1 online.

Estratégia de Condução

- Dúvidas acerca dos requisitos do projeto deverão ser esclarecidas com o professor.
- O código-fonte do projeto deverá ser enviado para o AVA até o prazo estipulado para a entrega. Não serão aceitas entregas após o prazo;
- Os projetos deverão ser apresentados pessoalmente ao professor na data definida. Trabalhos feitos em dupla deverão ser apresentados por ambos os alunos;
- A apresentação consistirá em execução e teste do aplicativo, seguido de arguição do professor sobre o código-fonte. Trabalhos não apresentados receberão a nota ZERO.
- O código fonte do projeto será submetido a uma ferramenta de verificação de plágio. Qualquer tentativa de cópia do projeto de outro aluno ou da Internet, ou qualquer outra tentativa de fraudar o projeto, incluindo cópia de trechos do código fonte, resultará em aplicação de nota ZERO.