FilmBox

GitHub

Ana Paula Natsumi Yuki
Theo Diniz Viana
Gabriel Henrique Pereira Carvalhaes
Lucas Teixeira Reis

Formulário Técnico

1) Qual foi a taxa de compressão obtida com o algoritmo de Huffman?
   a. Tamanho do arquivo original
      - 12.457 bytes
   b. Tamanho do arquivo comprimido
      - 2.986 bytes
   c. Cálculo da taxa
      - 76,03% de redução
   d. Interpretação do resultado
      - **Excelente compressão**, o algoritmo de Huffman foi muito eficiente, reduzindo o arquivo para menos de 1/4 do tamanho original.
2) Qual foi a taxa de compressão obtida com o algoritmo de LZW?
   a. Tamanho do arquivo original
      - 12.457 bytes
   b. Tamanho do arquivo comprimido
      - 1441 bytes
   c. Cálculo da taxa
      - 88,43% de redução
   d. Interpretação do resultado
      - O arquivo comprimido ficou com apenas cerca de 11% do tamanho original. Esse resultado indica uma compressão altamente eficiente.
3)
   - **Huffman**: A maior dificuldade foi integrar o algoritmo de Huffman que outra pessoa implementou, tentar entender como o código funcionava sem ter participado da sua criação, descobrir como passar os dados corretamente para ele e como lidar com os bytes comprimidos que voltavam, tudo isso sem quebrar o fluxo da sua aplicação Spring Boot.

   - **LZW**: Uma das principais dificuldades na implementação do LZW foi integrar corretamente o algoritmo , que já estava pronto no repositório do professor Kutova, com a estrutura da aplicação, especialmente na comunicação entre *controller*, *service* e o fluxo de arquivos. Além disso, foi necessário adaptar o código para lidar com múltiplos arquivos do banco local, gerar o arquivo compactado corretamente e garantir que os tamanhos original e comprimido fossem medidos de forma precisa para calcular a taxa de compressão. Também houve desafios para organizar o processo de download e retorno das informações no front-end. Essas dificuldades foram resolvidas ajustando a lógica

do service, padronizando os retornos do controller e realizando testes para garantir que a compressão e descompressão funcionassem de maneira consistente.

4) Tanto no LZW quanto no Huffman, a escolha por dicionários (HashMaps) mostrou-se fundamental, embora por razões complementares. No LZW, os HashMaps garantem acesso rápido às sequências já codificadas, essencial para um algoritmo que depende de consultas e inserções constantes de combinações de caracteres, oferecendo a simplicidade e performance necessárias que me permitiram focar na integração do service e controller com o algoritmo já disponível. Já no Huffman, a estratégia foi mais diversificada: combinamos Min-Heap para construção ótima da árvore, HashMaps para acesso rápido às frequências e códigos, e árvore binária para manter a propriedade de prefixos únicos, com BitSet resolvendo a manipulação eficiente de bits. Essa combinação de estruturas garantiu que cada etapa do algoritmo - da análise de frequências à codificação final - operasse com complexidades adequadas, traduzindo a teoria em implementação prática e eficiente.
5) O campo escolhido para criptografia foi a senha, porque é o único dado guardado no banco de dados que exige medidas de proteção externa e que justifica a necessidade de ser criptografado.
6)
    a) Estrutura das chaves pública e privada
        - Chave Pública: É formada pelos números N e E. Ela serve apenas para criptografar (trancar) a senha.
        - Chave Privada: É formada pelos números N e D. Ela serve para descriptografar (destrancar). O D é calculado matematicamente e fica escondido no sistema.
    b) Como e onde foram armazenadas?
        - As chaves (os números primos originais P, Q e o expoente E) foram escritas como constantes fixas no código Java (UserService). Por que: Se eu gerasse números aleatórios toda vez, ao reiniciar o programa eu perderia a capacidade de ler os dados que salvei antes.
    c) Como foram carregadas pelo sistema
        - Elas são carregadas automaticamente quando o programa inicia o UserService. O código lê os números fixos e calcula a chave privada na hora.
    d) Tamanho das chaves escolhidas e justificativa
        - Tamanho: Usei números primos pequenos (61 e 53).
        - Justificativa: Como é um projeto acadêmico, números pequenos facilitam os testes e evitam que o computador fique lento fazendo contas gigantes. Além disso, a senha criptografada não ocupa espaço demais no arquivo.
    e) Em qual momento a criptografia ocorre?
        - Acontece na hora de CRIAR o usuário. O código transforma a senha em números cifrados antes de mandar salvar no arquivo .db.
    f) Em qual momento ocorre a descriptografia?

- Acontece na hora de LER ou BUSCAR o usuário. O sistema lê os números do arquivo, faz a conta inversa para descobrir a senha original e só então mostra na tela.

g) Conversões realizadas
   - O passo a passo foi
     1) Pega a senha (ex: "ABC").
     2) Transforma cada letra em número (código ASCII).
     3) Faz a conta do RSA nesses números.
     4) O resultado vira uma sequência de números separados por espaço (ex: "123 998 442"), que é o que fica salvo.