PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS - LOURDES

DIEGO HENRIQUE XAVIER DOS SANTOS, MARCOS VINÍCIUS NUNES REIS,
RAFAEL GEORGETTI GROSSI E VITOR DANIEL SILVA MELO

TRABALHO PRÁTICO - FASE IV

ALGORITMOS E ESTRUTURAS DE DADOS III

BELO HORIZONTE - MG

2025

DIEGO HENRIQUE XAVIER DOS SANTOS, MARCOS VINÍCIUS NUNES REIS,
RAFAEL GEORGETTI GROSSI E VITOR DANIEL SILVA MELO

TRABALHO PRÁTICO - FASE IV

ALGORITMOS E ESTRUTURAS DE DADOS III

Trabalho apresentado ao curso superior de Ciência da Computação da PUC-MG Lourdes para o cumprimento das exigências da disciplina Algoritmos e Estruturas de Dados III

Orientador: Walisson Ferreira de Carvalho

BELO HORIZONTE - MG
2025

SUMÁRIO

FORMULÁRIO

1. Qual foi a taxa de compressão obtida com o algoritmo de Huffman?

a) Tamanho do arquivo original

2,29KB

b) Tamanho do arquivo comprimido

5,76KB

c) Cálculo da taxa

-151,15%

d) Interpretação do resultado

Para registros pequenos, Huffman não se torna muito eficiente.

2. Qual foi a taxa de compressão obtida com o algoritmo de LZW?

a) Tamanho do arquivo original

2,29 KB

b) Tamanho do arquivo comprimido

1,97KB

c) Cálculo da taxa

14,34%

d) Interpretação do resultado

LZW comprimiu o tamanho do arquivo de forma eficiente mesmo com registros pequenos.

3. Quais dificuldades surgiram ao implementar Huffman e LZW e como você resolveu?

O principal desafio do Huffman foi o bit packing, exigindo a criação da classe VetorDeBits para armazenar códigos de comprimento variável de forma contínua e eficiente. Já no LZW, a dificuldade residiu em garantir que os índices de 12 bits fossem corretamente empacotados e desempacotados em sequências de bytes e na complexa sincronização do dicionário entre codificador e decodificador, o que foi resolvido pelo design inerente ao algoritmo e pela mesma classe VetorDeBits.

4. Justifique a escolha da estrutura de dados usada para armazenar as tabelas, dicionários e árvores utilizados pelos algoritmos.

A escolha das estruturas de dados para armazenamento e indexação segue diretamente da natureza das relações e dos padrões de acesso que se deseja otimizar. No caso das relações 1:N, como ocorre em Tarefa com Usuário, o uso de uma árvore B+ é apropriado porque esse tipo de índice oferece busca ordenada, acesso eficiente por intervalo e custo previsível mesmo para arquivos grandes. Árvores B+ mantêm nós balanceados, garantem profundidade reduzida e proporcionam operações de inserção, remoção e busca com complexidade logarítmica. Isso é essencial quando se tem uma chave primária simples (por exemplo, id da tarefa) e outras chaves secundárias que também precisam ser consultadas de forma rápida, como o id do usuário. Além disso, esse tipo de estrutura organiza as chaves de forma sequencial nas folhas, facilitando leituras ordenadas do arquivo, algo muito adequado para relações hierárquicas e consultas que percorrem faixas inteiras de valores.

Já nas relação N:N, entre Tarefa e Categoria, a cardinalidade elevada implica muitas ocorrências para a mesma chave, e o acesso geralmente não exige ordenação, mas sim recuperação rápida e direta de entradas específicas. Para isso, o Hash Extensível oferece desempenho superior, pois permite acesso praticamente constante às posições armazenadas, independentemente do tamanho do arquivo. Além de rápido, ele suporta crescimento dinâmico sem necessidade de reestruturações completas, o que é crucial para tabelas de associação que tendem a crescer de forma imprevisível. Como cada par (idTarefa, idCategoria) funciona como uma chave composta, a dispersão oferecida pelo hash reduz colisões e mantém o desempenho mesmo com grande volume de registros.

5. Qual campo foi escolhido para criptografia? Por quê?

O campo escolhido para criptografia foi a senha. Porque se trata de um dado sensível que fornece acesso às contas dos usuários. A fim de proteger seus dados, como as tarefas, esse campo foi criptografado para evitar acesso indevido.

6. Descreva como o RSA foi implementado no projeto.

a. Estrutura das chaves pública e privada

O projeto define duas classes específicas: RSAPublicKey e RSAPrivateKey. Cada chave contém dois valores fundamentais, a chave pública que é composta pelo expoente público **e** e pelo módulo **n**. E a chave privada que é composta pelo expoente privado **d** e também pelo módulo **n**. Esses dois componentes são suficientes para que operações de criptografia (com e) e descriptografia (com d) sejam realizadas corretamente, garantindo segurança conforme o esquema RSA tradicional.

b. Como e onde foram armazenadas

As chaves são geradas dinamicamente em memória por meio do método generateKeyPair(). O sistema produz um par contendo: RSAPublicKey RSAPrivateKey Esses objetos são utilizados diretamente pelas operações de criptografia e descriptografia. Não há persistência em arquivos externos, bancos de dados ou estruturas de índice, a cada execução, novas chaves são criadas.

c. Como foram carregadas pelo sistema

Como as chaves não são persistidas, elas são simplesmente instanciadas e utilizadas no momento da execução, permanecendo em memória enquanto o programa está ativo. O método generateKeyPair() é responsável por: Gerar dois primos aleatórios p e q; Calcular n = p * q; Calcular $\varphi(n) = (p - 1)(q - 1)$; Definir o expoente público fixo e = 65537; Calcular o inverso modular $d = e^{-1} \bmod \varphi(n)$ usando o algoritmo estendido de Euclides. O conjunto (e, n) e (d, n) é então retornado ao sistema.

d. Tamanho das chaves escolhidas e justificativa

O tamanho das chaves utilizado no projeto é definido pelo parâmetro fornecido à função generateKeyPair(int keySize).A implementação divide o tamanho total em dois primos aproximadamente iguais (keySize/2 cada). Em testes e na aplicação prática, o tamanho escolhido foi 1024 bits, pois fornece segurança adequada ao ambiente acadêmico do projeto e mantém tempo de processamento aceitável para geração das chaves e operações de criptografia.

e. Em qual momento a criptografia do(s) campo(s) ocorre (no CRUD).

A criptografia ocorre antes da escrita do registro no arquivo, ou seja, no momento do CREATE ou UPDATE, sempre que um campo protegido precisar ser armazenado de forma segura. No caso do projeto, o RSA foi aplicado especificamente ao(s) campo(s) escolhidos (ex.: senha, e-mail ou campo sensível definido pelo grupo). O valor textual é convertido em número, criptografado com a chave pública e só então armazenado no disco.

f. Em qual momento ocorre a descriptografia.

A descriptografia ocorre no momento em que os dados precisam ser recuperados e exibidos ao usuário ou utilizados pelo sistema. Isso acontece durante as operações READ e eventualmente durante buscas ou autenticações. Nessa etapa, o valor cifrado é passado ao método decrypt(), que utiliza a chave privada (expoente d) para retornar o texto original.

g. Conversões realizadas (ex.: string → bytes → blocos).

Para que o RSA possa operar sobre o campo textual, são realizadas várias conversões. String → bytes, em que o texto é convertido para byte[] usando UTF-8. Bytes → BigIntegerm em que o array é interpretado como número positivo (new BigInteger(1, bytes)). Criptografia O valor criptografado é guardado como BigInteger ou convertido para string/bytes conforme o formato adotado no registro. E então na descriptografia é convertido o BigInteger resultante para bytes e então reconstrói a string original em UTF-8.

REPOSITÓRIO GITHUB

- https://github.com/vmelooo/tp-aeds3