UNIVERSIDADE FEDERAL DO PARANÁ

EFI FUCHS SANTOS DA SILVA

SOFIA BARBOSA CANDITTO

TRABALHO DE REDES

Codificação e Correção de Erros com Hamming(31,26)

Curitiba

2025

1. Introdução

A confiabilidade na transmissão e armazenamento de dados é um requisito essencial em sistemas computacionais. Uma técnica amplamente utilizada para garantir a integridade dos dados é a **codificação com detecção e correção de erros**. Este projeto apresenta a implementação de um sistema simples de codificação e decodificação utilizando o **código de Hamming (31,26)**, que é capaz de detectar e corrigir erros de 1 bit em cada bloco de dados.

2. Objetivo

O objetivo deste projeto é:

- Implementar a codificação de arquivos binários utilizando o código de Hamming(31,26);
- Corrigir automaticamente erros de 1 bit durante a decodificação;
- Armazenar os dados codificados em formato textual, com bits representados como caracteres `'0'` e `'1'`, separados por espaços;
- Permitir que os dados sejam recuperados corretamente mesmo em presença de erros de 1 bit.

3. Fundamentos Teóricos

3.1 Código de Hamming (31,26)

O **código de Hamming(31,26)** é uma técnica de codificação linear que transforma 26 bits de dados em 31 bits codificados. Os 5 bits adicionais são **bits de paridade**, posicionados

nas potências de dois (posições 1, 2, 4, 8, 16). Esses bits permitem detectar e corrigir um erro de até 1 bit por bloco.

Características:

- Tamanho do bloco de dados: 26 bits
- Tamanho do bloco codificado: 31 bits
- Capacidade de correção: 1 erro por bloco
- Eficiência: 26/31 ≈ 83.87%

4. Estrutura do Projeto

O projeto é composto por três funcionalidades principais:

- `-e <arquivo>`: Codifica o arquivo utilizando Hamming(31,26)
- `-d <arquivo>`: Decodifica e corrige o arquivo codificado
- `-s <arquivo>`: (não implementado) Simular erro de 1 bit para testes

Além disso, o programa exibe uma mensagem de ajuda com a opção `-h`.

5. Implementação

5.1 Codificação (`encodeFileHamming3126`)

Durante a codificação:

1. O arquivo de entrada é lido byte a byte.
2. Os bits são agrupados em blocos de 26 bits.
3. Cada bloco é codificado usando `encodeHamming3126`.
4. O bloco de 31 bits resultante é transformado em uma string de `'0'` e `'1'` e escrito no arquivo de saída, com espaços entre os blocos.
5. Um bloco extra é adicionado ao final contendo metadados de preenchimento (padding) para garantir a reconstrução correta no final da decodificação.

5.2 Decodificação (`decodeFileHamming3126`)

Durante a decodificação:

1. O arquivo é lido como uma sequência de strings de bits separados por espaço.

2. Cada bloco de 31 bits é convertido em `std::bitset<31>` e decodificado com `decodeHamming3126`.
3. Os bits de dados são extraídos e armazenados em um buffer.
4. A cada 8 bits, é reconstruído um byte original.
5. O último bloco de dados é tratado com base nos metadados de padding.

6. Formato de Armazenamento

Diferentemente da maioria das implementações binárias, este projeto salva os blocos codificados como **texto puro**, com:

- Cada bit representado como `'0'` ou `'1'`
- Blocos separados por um espaço
- Facilita leitura e inspeção manual

7. Resultados Esperados

- O arquivo `.hamming` deve conter apenas `'0'` e `'1'` com espaços entre blocos.
- O arquivo `.dec` gerado após decodificação deve ser idêntico ao original, mesmo com um erro de 1 bit introduzido em qualquer bloco.
- O sistema identifica e corrige automaticamente a posição errada em cada bloco.

---

8. Conclusão

Este projeto demonstra, de forma prática e educacional, como é possível implementar mecanismos de **correção de erro confiáveis** utilizando o **código de Hamming(31,26)**. A abordagem baseada em texto facilita a compreensão e a depuração dos dados codificados. A lógica de detecção e correção foi cuidadosamente implementada para garantir que erros simples possam ser corrigidos sem perda de dados.

9. Considerações Finais

Durante o desenvolvimento do projeto e análise da implementação, algumas limitações técnicas e conceituais merecem destaque:

1. Limitação do Código de Hamming com Erros Múltiplos

O código de Hamming(31,26) é eficaz apenas para a **correção de um único erro de bit** por bloco de 31 bits. Isso significa que:

- **Se ocorrerem dois ou mais erros** em um mesmo bloco, o código pode **não detectá-los** corretamente ou, pior, **corrigir de forma errada**, corrompendo o dado original.
- Este comportamento é **esperado e documentado** na teoria de códigos de Hamming, que são classificadas como *SEC* (Single Error Correction).
- Para proteção contra múltiplos erros, seria necessário utilizar códigos mais robustos, como **SECDED (Single Error Correction, Double Error Detection)** ou códigos de Reed-Solomon.

2. Leitura Completa do Arquivo em Memória

A função `decodeFileHamming3126` atualmente:

- Lê **todo o conteúdo do arquivo `.hamming` para a memória**, armazenando todos os blocos de bits como strings em um vetor (`std::vector<std::string> chunks`).
- Essa abordagem é simples e eficiente para arquivos pequenos ou médios, mas:
    - **Não é escalável** para arquivos muito grandes (acima de centenas de MBs ou GBs).
    - Pode causar **estouro de memória (out-of-memory)** em sistemas com recursos limitados.

Possível melhoria: a leitura poderia ser feita em **streaming** (linha por linha ou bloco por bloco), processando e descartando os blocos decodificados à medida que são lidos. Isso evitaria a sobrecarga de memória e aumentaria a escalabilidade do programa.