Relatório T3 – Redes I

Alunas: Giovanna Fioravante Dalledone e Nadia Luana Lobkov

1. Sobre o Trabalho

Foram apresentadas, em sala, duas maneiras de implementar o código Hamming (31, 26). A escolhida foi utilizar bits de paridade. Essa implementação leva em conta 5 bits de paridade – 1, 2, 4, 8, 16 respectivamente – e a posição do bit ligado no vetor de 26 bits.

A ideia é fazer um AND bit a bit com a posição do bit de paridade e a posição corrente do vetor de dados, caso o resultado seja 1 então realiza-se um XOR com o valor dentro da posição atual e o valor dentro da posição do bit de paridade. Feito isso, a decodificação segue de maneira similar. Para a detecção dos erros, o valor dos bits de paridade são calculados novamente e comparados com o anterior. Em caso de erro, eles serão diferentes e fornecerão a posição binária de onde ocorreu o erro.

2. Implementação

A estrutura escolhida foi a mais simples possível. Implementamos, somente, um vetor de 32 espaços e o chamamos de Hamming. São 32 espaços e não 31, porque os índices dessa codificação devem ser iniciados em 1, por conta do cálculo dos bits de paridade.

2.1. Funções:

marcaPosicoes() - preenche com dados todas as posições que não são potência de 2, pois elas são exclusivas dos bits de paridade.

codificaHamming() - realiza o cálculo dos bits de paridade.

traduz() - responsável por montar os bytes dos bits recebidos do arquivo temporário.

decodificaHamming() - é quem verifica o cálculo dos bits de paridade do arquivo intermediário, corrige os erros – caso existam -, e escreve os bits de dados (sem os de paridade) em um arquivo temporário.

Com relação à implementação das funções, o maior desafio foi lidar com a montagem e desmontagem dos bytes de cada linha lida dos arquivos. Nos primeiros testes do trabalho, o arquivo final (monologo.dec) exibia somente caracteres estranhos, porque a montagem dos bytes não era feita corretamente. Em seguida – depois de alguns dias – percebemos que, além do erro na montagem dos bytes, o arquivo temporário estava sendo escrito errado – as linhas estavam escritas umas sobre as outras. Outro erro percebido foi que alguns bits estavam sendo escritos no formato errado dentro do vetor Hammin→codigo.

Um problema persistente é a impressão de caracteres desconhecidos ao final do arquivo decodificado, eles são decorrentes dos zeros inseridos na codificação para completar o vetor de 31 bits de dados.

3. Fluxo

Inicialmente, o usuário deve digitar a entrada que ele quer codificar. O programa, então, irá gerar um arquivo intermediário com a codificação especificada. Esse arquivo é montado de 32 em 32 bits, porque são 31 bits do vetor Hamming→codigo + ‘\n’.

Quando o arquivo intermediário for completamente gerado, há um fclose que faz com que ele fique salvo mesmo com o programa ainda em execução (dessa forma podemos modificar o arquivo para testar a detecção dos erros).

Depois que o arquivo intermediário for lido, a função decodificaHamming() cumpre seu papel em conferir os bits de paridade, corrigir possíveis erros e passar os dados para um arquivo temporário que será traduzido pela função traduz.