Relatório de decisões sobre o trabalho 2

Rede de Computadores I

Ciência da Computação - UFPR

Prof. Dr. Luiz Carlos Pessoa Albini

Guilherme M. Lopes do Santos - GRR20163043
Leonardo Stefan - GRR20163052

Curitiba, 2019

Objetivo do trabalho

Desenvolver um programa que receba um arquivo de texto de tamanho arbitrário em codificação ASCII, efetue a codificação do texto todo em Hamming (11, 15), e em seguida, a decodificação, tendo como produto final um arquivo equivalente ao aquivo de entrada.

Do relatório

O objetivo deste relatório é comentar as principais decisões de projeto tomadas, explicar o funcionamento do programa, e documentar os problemas conhecidos e propor possíveis soluções para eles, a serem implementados em uma segunda versão do programa.

Linguagem de programação

Apesar da dificuldade em lidar com ponteiros e acesso à memória, a linguagem C (padrão C99) foi escolhida por conta da facilidade em lidar com baixo nível. Por exemplo, utilizamos estruturas de união (unions) para processar os bits de cada caractere individualmente; e funções das bibliotecas stdio.h e stdlib.h para criar fluxos de leitura e escrita de caracteres individuais dinamicamente em arquivos.

Projeto

Decidimos separar o projeto em dois: a codificação/decodificação e a estrutura em volta necessária para manipular os dados. A divisão nos surpreendeu pois a estrutura criada para manipular o texto antes de enviar para a codificação, bem como para recuperar o texto, mostrou-se mais complexa e demorada para desenvolver do que as funções de codificação e decodificação. A primeira versão da estrutura seguia os seguintes algoritmos:

Codificação:

1) Lê o arquivo entrada.txt;
2) Separa cada caractere, e, para cada um, armazena seu código ASCII equivalente em bits em um vetor, na sequência correta;
3) Armazena a sequência binária em um arquivo chamado string.txt;
4) Lê o arquivo string.txt de 11 em 11 bits, e, para cada 11 bits lidos, chama a função encodeHamming, que codifica os dados em Hamming(11, 15) armazenando o resultado em outro array chamado encoded_array;
5) Guarda o resultado de encoded_array em um arquivo chamado out_file.txt;
6) Na maioria das vezes, sobrará X bits no final da sequência, onde X < 11, e neste caso, tratamos esta sobra completando com zeros ao final (até que chegue a 11 bits);

Decodificação:

1) Lê o arquivo out_file.txt de 15 em 15 bits, e, para cada sequência de 15 bits, chama a função decodeHamming guardando o resultado em um array chamado out2;
2) o array out2 é encaminhado para um buffer através da função out_buffer, que também recebe um ponteiro para o arquivo final de saída denominado final_output;
3) o buffer então escreve 8 bytes (1 caractere) no arquivo final_output sempre que este possui ao menos 8 bytes esperando para serem escritos;
4) é gerado o arquivo de saída com o texto equivalente ao que existia no arquivo de entrada.

Entretanto, percebemos que a abordagem do buffer era problemática, pois o texto final continha alguns erros. Acreditamos que fosse por conta de acessos errados à memória, que eventualmente ocorrem quando tratamos de uma grande quantidade de acessos dinâmicos (mais de mil caracteres manipulados individualmente em laços de repetição, por exemplo). Então, decidimos mudar para uma abordagem de array: guardamos os resultados da decodificação em um array do tamanho no texto todo, e em seguida apenas imprimimos na tela e no arquivo final_output o conteúdo deste array. Assim, a entrada e a saída ficaram equivalentes.

Problemas conhecidos

Por algum motivo ainda desconhecido o programa não funciona como esperado para entradas com mais de 12 mil caracteres. A solução em um cenário hipotético de uso da ferramenta seria codificar/decodificar textos de 12 mil em 12 mil caracteres. Acreditamos que tenha a ver com o tamanho máximo do array que utilizamos como substituto do buffer.