Relatório referente ao Trabalho II de Programação de Software Básico

Bernardo de Cesaro[*], Gustavo Machado Possebon[†]
Faculdade de Informática — PUCRS

21 de Junho de 2019

Resumo

Este relatório descreve uma alternativa de solução para o segundo problema proposto na disciplina de Programação de Software Básico no semestre 2019/1, onde deve-se construir uma aplicação que implemente o algoritmo de Huffman, sendo capaz de codificar/descodificar arquivos textuais.

Introdução

Dentro da disciplina de Programação de Software Básico foi-nos proposto a implementação de um algoritmo bastante conhecido, chamado de *Algoritmo de Huffman*.Este algoritmo de compressão utiliza as probabilidades da ocorrência de símbolos e palavras em um conjunto de dados para determinar quantos bits serão utilizados para cada simbolo. A partir disso, para realizar a compressão, é contado a ocorrência de cada caractere na sequência e montada uma respectiva árvore, utilizando sempre os 2 caracteres com menor ocorrência para a criação da árvore.

Pegando estes 2 caracteres, soma-se suas frequências em um novo nodo da árvore. Depois de construida a árvore final, basta percorrer a árvore a partir dos bits correspondentes para cada aresta e identificar quanto vale cada um dos caracteres presentes na árvore.

Desenvolvimento

Dividiu-se o desenvolvimento do algoritmo de Huffman em duas partes fundamentais, sendo estas uma classe para manipulação de uma lista encadeada e outra para a geração da árvore. Posterior a isto, tem-se a criação de uma classe para pegar os bits de cada um dos nós gerados na árvore.

Classe *Main*

Após todo o processo de compilação executado pelo GCC(*GNU Compiler Collection*), a *main.c* é a responsável pelo restante do trabalho pois é onde se encontra todo o código fonte do algoritmo. Nas primeiras linhas são feitas as inclusões das bibliotecas tanto padrões da linguagem quanto as desenvolvidas por terceiros e alguns defines implementados. Em seguida, os tipos de estruturas que serão utilizadas durante todo funcionamento: uma *struct* em relação à árvore, e outras duas para tanto com a própria lista quanto para os nodos presentes nela. Funções mais utilizadas sem seus respectivos argumentos:

[*]b.cesaro@edu.pucrs.br
[†]gustavo.possebon@edu.pucrs.br

- insertNodeLinkedList() - Função que faz alocação de memória e mexe com o tratamento de ponteiros dos nós da lista.
- insereNodoArvore() - Função que faz alocação de memória e mexe com o tratamento de ponteiros dos nós da árvore.
- insertOrderly() - Função que manipula um determinado nó em uma lista e sua frequencia em relação aos outros elementos presentes na lista.
- returnMin() - Responsável pela retirada de um determinado nodo da lista.
- dibbeHuff() - Função principal onde toda a árvore é gerada conforme a frequência dos caracteres. Dentro deste método teremos a retirada dos dois caracteres de menor frequência e a soma deles sendo inserida como um novo nodo contendo um caractere específico para representar esta soma.
- generateCodeOfChars() - Responsável pela atribuição dos códigos 0 ou 1 para cada nodo presente.

No final da classe é onde acontece as entradas e saídas, impressões de tela com os valores de frequência de cada caractere e seus respetivos bits de caminhamento.

Função de *Compressão*

Para o desenvolvimento da parte de codificação dos caracteres presentes na árvore, foi feito o uso de um dicionário para guardar as referências de caminhamento para cada nodo visitado e a criação de uma função para a manipulação dessas informações, chamada de *generateCodeOfChars*. Nesta função é realizada a adição de 0 ou 1 dependendo do caminhamento *(0 para esquerda e 1 para a direita)*. No final o algoritmo gera um arquivo .piz onde contém toda a tabela de huffman com seus caracteres, frequencia e representação binaria na arvore de huffman.

Conclusões

Encontramos muitas dificuldades na implementação deste trabalho por diversos motivos. Uma das dificuldades enfrentadas foi na forma como iriamos manipular os nodos de uma árvore com a lista encadeada, bem como em uma implementação fácil e de bom entendimento do algoritmo para geração da árvore. Outra dificuldade que tivemos, esta talvez sendo a que mais nos trouxe problemas, foi na forma como íriamos realizar o caminhamento da árvore para pegar o código atribuído para cada nodo.

Quanto a estrutura do programa, acreditamos que a eficiência possa melhorar a partir da criação de classes separadas, uma para codificar e outra para decodificar, assim apenas instânciando-as na classe principal.

Concluimos que a implementação atende apenas alguns dos requisitos propostos, infelizmente não tivemos tempo/nem conhecimento suficiente para realizar uma implementação completa da parte de descriptografar o arquivo.