Trabalho Prático 3
Decifrando os Segredos de Arendelle

Misael Saraiva de Rezende

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

1. Introdução

Em constante transformação, o reino de Arendelle solicitou mais uma vez uma ajuda para decodificar antigas mensagens criptografadas em código Morse que serão criptografadas novamente em técnicas modernas e mais confiáveis. Sabe-se que os principais especialistas nesta técnica de criptografia já estão aposentados, o que torna esse trabalho mais necessário e importante para o reino.
Esse programa executa a criação de uma estrutura de apoio para ajudar a decodificar palavras encriptadas na técnica código Morse.

2. Implementação

O programa foi desenvolvido na linguagem de programação C, e compilado pelo compilador gcc.

I. Estrutura de Dados

A implementação do programa teve como base a estrutura de dados de uma estrutura (struct) do tipo *Registro* que basicamente armazena três variáveis: *simbolo*, *caractere* e *codigoMorse*. Também foi implementado outra estrutura do tipo *No* que armazena dois ponteiros desse tipo (esquerda e direita) e também contém um *Registro*.
A árvore Trie é uma árvore que representa uma chave na qual fica organizada de acordo com alguma sequência pré-definida. No caso deste trabalho a árvore criada foi uma variação da Trie. Também nota-se que a árvore ser criada é binária e que essa possui muitas semelhanças com a árvore de pesquisa binária. Dado isso, foi definido que a árvore a ser criada seria bem parecida com uma de pesquisa binária com a diferença que ao invés de fazer comparações de valores para caminhar na árvore, faz-se comparação de chaves. Dessa maneira a árvore de codificação é uma variação da Trie, sendo que a principal diferença entre as duas é o fato de que a Trie só possui chaves em nós folhas, o que não acontece para a árvore utilizada nessa implementação.

II. Modularização

Para modularizar a implementação, foram usados dois módulos principais, *Arvore* e o *main.c* (programa principal).

O módulo *Arvore* contém a implementação dos algoritmos relacionados a árvore e suas funções básicas. A função *inicializaArvore* cria o nó raiz da árvore. A função *criaNo* cria um nó para ser adicionado na árvore. A função *insere* faz o que o próprio nome sugere, guardar o valor alfabético da tabela do código morse no caminhamento correto da chave. Quanto a função *decodificaMensagem,* a mesma realiza a decodificação de uma mensagem criptografada inserida pelo usuário. Já a função *imprimePreOrdem* "imprime" a árvore no caminhamento pré-ordem. Por último a função *esvaziaArvore* desaloca os nós alocados no fim da execução do programa.

Por sua vez, o módulo principal *main.c* contém as variáveis que armazenam os dados a serem usados e a implementação que realiza a comunicação direto com o usuário do programa.

III. Entrada e saída de dados

Espera-se que o usuário entre com os parâmetros a serem executados pelo programa pelo terminal de comando do sistema - veja as instruções de execução no próximo tópico. Opcionalmente pode ser passado por parâmetro a flag **-a** que imprime a árvore criada no caminhamento em pré-ordem.
Quanto a entrada, outra forma de entrada de dados é a entrada padrão (stdin) ou passando o nome de um arquivo por parâmetro (de acordo com o padrão do terminal de comando do linux).
Para a saída, tem se a mensagem (ns) decodificada (s), podendo ser mostrada na saída padrão (stdout) ou arquivo, de acordo com a opção do utilizador - veja o próximo tópico para entender como os dados de saída são formatados.

3. Instruções de compilação e execução

O computador que rodou o código implementado foi um computador de uso próprio, com sistema operacional Linux Ubuntu 16.02, processador Intel® Core i3 CPU @ 2.40GHz, 64-bit e Memória RAM de 4.0 GiB.
Esse programa foi editado e compilado em máquinas usando o sistema operacional Linux (variante Ubuntu versão 16.02 LTS). A compilação sempre foi feita utilizando o terminal disponível por padrão no sistema.
Para compilar esse programa, foi usado o arquivo de configuração Makefile que facilita a compilação de todos os códigos nos arquivos modularizados.

Para compilar basta digitar ***make*** na linha de comando do terminal. Para limpar os arquivos gerados pela compilação pode-se digitar o comando ***make clean***.

De acordo com o que foi sugerido pelo enunciado, o modelo para executar é o:

./nomedoprograma [-a].

Exemplo 1:

./tp3 -a

que vai executar o programa gerando a árvore de codificação e decodificando a mensagem inserida pela entrada padrão e a imprimindo na saída padrão. No fim também é impresso as chaves da árvore de codificação usando o caminhamento pré-ordem.

Exemplo 2:

./tp3 < 3.in > 3.out

que vai executar o programa realizando a criação da árvore de codificação, lendo o arquivo *3.in*, com os dados criptografados, e imprimindo as palavras decodificadas no arquivo *3.out*.

4. Conclusão

Foi realizada a execução de um projeto para ajudar o reino arendellense a traduzir mensagens antigas criptografadas em código Morse dos bancos de dados antigos. O trabalho foi concluído com êxito e isso vai possibilitar que os funcionários do reino tenham bastante facilidade em transferir as mensagens criptografadas para outras técnicas melhores em codificação de palavras.

Quanto aos contratempos, tive dificuldades para implementar uma variação do código da árvore Trie disponibilizada nos slides da disciplina. Em um âmbito geral vi que o meu maior entrave foi lidar com acesso a memória, fator importante que a linguagem C permite usar, porém que também exige bom raciocínio e bastante atenção do programador, pois seu uso pode levar o desenvolvedor a confusão.

Observação: Há um bug no código para os casos de teste 2 e 3, que não funcionam corretamente se a entrada dos dados para descriptografar vierem de um arquivo e não da entrada padrão stdin (independente de onde se escolha a saída decodificada).

5. Bibliografia

- slides da professora Raquel Prates:
    - ED_Aula7_Arvore
    - ED_Aula16_PesquisaDigital
    - ED_Aula17_ArvoreBinariaPesquisa

- Slides do livro Projeto de Algoritmos com implementações em Pascal e C
    - Capítulo Pesquisa em memória primária

- Livro Linguagem C - Completa e Descomplicada - André Backes
    - Capítulo 7 - Vetor de caracteres
    - Capítulo 10 - Ponteiro
    - Capítulo 11 - Alocação Dinâmica
    - Capítulo 12 - Arquivos

- **Trie** <https://en.wikipedia.org/wiki/Trie> - Acessado em 24 Jun 2019
- **Estruturas de Dados: Tries - IME-USP** <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html> - Acessado em 24 Jun 2019
- **The getline() Function** <https://c-for-dummies.com/blog/?p=1112> - Acessado em 06 Jul 2019