Trabalho Prático 3: Decifrando os Segredos de Arendelle

Gabriel Victor Carvalho Rocha

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

gabrielcarvalho@dcc.ufmg.br - 2018054907

1. Introdução

O problema abordado no trabalho, foi a implementação de uma Árvore Digital (Trie binária), que ao invés de utilizar a comparação de chaves, percorremos pela árvore utilizando os dígitos em Morse (ponto e traço) até chegar na posição exata, podendo então ter como retorno a letra correspondente. Um detalhe da árvore utilizada para solucionar o problema é que o número de dígitos do Morse é variado, podendo ter registros em vários nós ao longo da árvore, diferenciando-se então daquela apresentada em sala, em que os registros se encontravam nas folhas.

Já o programa consiste em ler um arquivo txt com os códigos Morse, inserindo na Trie todos os símbolos do arquivo junto a seu respectivo código morse. Após isso, é recebido por meio do stdin os códigos que devem ser decodificados, que serão enviados, código por código, para a função de pesquisa, retornando a letra correspondente. No final, é retornado todas as mensagens decodificadas e opcionalmente, o caminhamento pré-ordem da árvore.

2. Implementação

Todo o programa foi desenvolvido na linguagem C, utilizando o compilador GCC (GNU Compiler Collection).

2.1. Principais funções

As principais funções consistem em: criar um nó vazio, inserir os símbolos na árvore e pesquisar qual letra corresponde ao morse desejado.

cria_no_vazio(), irá apenas alocar memória para um No, colocando valores vazios e fazendo seus apontadores apontarem para NULL.

A função insere(No*, Registro*) irá receber a raiz da Trie e o Registro com o símbolo e seu código morse coletado do arquivo morse.txt. Esta função simplesmente irá chamar a função insere_rec(No*, Registro*, long int, int), enviando o tamanho do código morse e inicializando as chamadas de recursão com nível 0. Ao entrar no insere_rec, irá recursivamente percorrer pela Trie de acordo com o código em relação ao nível da árvore, verificando se o dígito é ponto ou traço, testando sempre se o nó apontado é NULL, que em caso positivo, irá criar um nó vazio no lugar para dar continuidade ao caminhamento se chamando recursivamente com o nível incrementado de 1, tendo como condição de parada o nível ser igual ao tamanho do código, onde será inserido de fato os registros no nó.

Já a função procura(No*, char*), irá receber a raiz da Trie e o vetor de char do código

morse do qual se deseja decodificar. Esta função simplesmente irá chamar a função pesquisa_rec(No*, char*, long int, int) enviando os mesmos parâmetros do procura mas incluindo também o tamanho do código e inicializando a recursão com nível 0. Ao entrar na pesquisa_rec, irá caminhar ao longo da árvore de acordo com o dígito morse em relação ao nível que se encontra e após realizar esse procedimento, irá se chamar recursivamente com o nivel incrementado de 1, após chegar na condição de parada onde o nível é igual ao tamanho do código, é retornado o char do símbolo.

Algumas decisões a respeito destas funções descritas anteriormente foram tomadas, que serão explicadas no tópico abaixo.

No main, depois que se abre o arquivo morse.txt, há um loop while que irá pegar do arquivo os 36 símbolos, inserindo-os na Trie. E logo após, é feito um outro loop while até que não seja o fim do arquivo, recebendo os códigos (ou "/") por meio do stdin, para pesquisá-los e depois que encontrado, imprimi-los na saída (no caso do "/", apenas imprimir um espaço). Por fim, há um if para verificar se o argumento opcional "-a" foi inserido, caso seja verdadeiro, irá chamar a função de imprimir a árvore em pré-ordem.

2.2. Algumas decisões

Para criar nós vazios, tomei a decisão de deixar tanto o morse como a letra com valores " " (espaço), e nas funções de pesquisa se o apontador apontar para um nó NULL (ou seja, não existe o caminhamento para o morse), é retornado também o valor " " (espaço).

Outro ocorrido foi quando se executa apenas o executável ./tp3 ou ./tp3 -a. Quando se coloca os códigos Morse e se obtém a decodificação, após pressionar Ctrl + D (para finalizar a leitura) é gerado um espaço a mais, que não consegui achar uma maneira de evitá-lo.

3. Instruções de compilação, execução e formato de entrada/saída

A compilação é feita através de um arquivo Makefile, gerando o executável "tp3", já a execução é realizada das seguintes formas:

./tp3 [-a] [<caso.in] [>caso.out] , forma geral.

./tp3 , que irá permitir colocar valores em morse manualmente, tendo os resultados no terminal após ser pressionado Enter, para terminar a leitura é necessário pressionar Ctrl + D.

./tp3 <caso.in >caso.out , que irá receber um arquivo .in com os códigos morse, sendo colocado os resultados obtidos em um arquivo .out. Caso seja omitido o >caso.out, o resultado será impresso no próprio terminal.

./tp3 -a <caso.in >caso.out , que possui as mesmas coisas descritas no caso acima, com a diferença do parâmetro opcional "-a", que quando utilizado, será impresso o caminhamento pré-ordem da árvore criada.

Formato de entrada e saída:

Na entrada, o conjunto de código Morse (. e -) referente à cada letra é separado um do outro por espaços, já as palavras são separadas com "/":

“cod0” “cod1” “cod2” “/” “cod3” “cod4” “cod5” “cod6” “cod7” (genérico)

-... --- ..-. / --. --- ..- -.. . (“BOA NOITE”)

Já a saída consiste nas mensagens codificadas, sendo cada linha uma nova mensagem:

<mensagem 0 decodificada>
<mensagem n-1 decodificada>

Caso “-a” for utilizado, será impresso também em cada linha a letra e o morse correspondente:

<mensagem 0 decodificada>
<mensagem n-1 decodificada>

<simbolo 0> <codigo 0>
<simbolo n-1> <codigo n-1>

4. Análise de complexidade

4.1. Tempo

Na complexidade em relação ao tempo, devemos analisar o código principal main.

Primeiramente, definindo m como o número total de pontos e traços do arquivo morse.txt, temos então o loop para ler este arquivo, que irá inserir os símbolos na árvore com a função insere, que possui complexidade O(m), pois para cada caracter do Morse é necessário percorrer um nó, entretanto aqui temos um custo total constante O(1) pois o tamanho do arquivo é constante e será realizado independente do número de entradas do stdin, pois o tamanho da árvore neste trabalho é fixo.

Então temos o segundo loop, que consiste em um while que rodará enquanto não for o fim da entrada stdin, realizando a pesquisa a cada iteração. Definindo n como a quantidade de caracteres total de todos os códigos Morse (quantidade de pontos e traços), como para cada carácter precisamos movimentar uma vez dentro da árvore, para n caracteres precisamos de n movimentos, logo, aqui temos a complexidade O(n) para pesquisar todos os códigos da entrada.

E a função imprime_preordem, sendo n o número total de nós na árvore, irá percorrer por toda a Trie, tendo custo O(n), mas como explicado anteriormente, a quantidade de nós é fixa, portanto este custo não se altera com o aumento do número de entradas, tendo complexidade O(1).

O custo total do programa em relação ao tempo é O(n).

4.2. Espaço

Já na complexidade de espaço, temos que o custo é O(m), sendo m o número total de nós da Trie, porém, como mencionado anteriormente, a árvore possui um tamanho fixo, possuindo

então um custo constate que não varia de acordo com o número n (quantidade de caracteres total) de entradas, tendo complexidade constante O(1).

5. Conclusão

Por fim, o trabalho se mostrou muito interessante, podendo ser vista a rapidez da pesquisa e também da inserção na árvore Trie, tendo uma boa performance até mesmo para arquivos grandes se comparado aos algoritmos de pesquisa sequencial, e além disso, possuindo um código simples e de fácil implementação tanto de maneira iterativa, que foi a primeira tentativa de implementação, como a recursiva, que foi a de fato implementada no trabalho.

Ao longo do desenvolvimento, algumas dificuldades foram enfrentadas, como a leitura dos arquivos, que foi a parte mais trabalhosa de todo o trabalho e também na construção da árvore, onde alguns nós ficaram com valores "vazios", gerando dúvida se estaria correto, pois em pesquisas de códigos que chegam nestes nós vazios, retornam um valor " " (espaço). Outro problema encontrado foi explicado no tópico 2.2 onde quando se executa o código apenas com o executável ./tp3, gerou-se um espaço a mais, que não foi encontrada uma maneira de evitá-lo.

Assim que foi entendido como funcionava o caminhamento pela árvore, verificando se o próximo nó a seguir era NULL ou não, para então saber se era preciso criar um novo nó ou prosseguir no caminhamento, o trabalho fluiu melhor e acabou se tornando uma ótima maneira de entender ainda mais a Trie binária.

6. Bibliografia

ZIVIANI, Nivio. Projeto de Algoritmos com Implementações em Pascal e C. 3ª Edição, Cengage Learning, 2011.

CHAIMOWICZ, Luiz.; PRATES, Raquel. Aula 7 - Árvores. PDF disponibilizado via Moodle UFMG.

CHAIMOWICZ, Luiz.; PRATES, Raquel. Aula 16 - Pesquisa Digital. PDF disponibilizado via Moodle UFMG.