Algoritmos e Estruturas de Dados ||

Marcella Netto de Oliveira

Marcelo Ari de Oliveira

Universidade Federal de São João Del Rei – MG

27 de junho de 2019

Documentação – Trabalho prático

Algoritmos e Estruturas de Dados (2)

Introdução

Árvores Binárias, segundo o seu significado em Ciência da computação, são estruturas de dados usadas para organizar um conjunto de informações ou dados de forma eficiente na memória do computador.

Dessa forma, se torna algo de extrema utilidade em projetos de programação, visto que assim como as listas encadeadas, as árvores binárias precisam de “pouco espaço” para que sejam implementadas, ou seja, possuem um excelente custo/benefício. Além disso, quando balanceadas, essas árvores possibilitam um ótimo tempo de pesquisa em relação as listas encadeadas, o que as torna mais eficientes pois, seu tempo de inserção, remoção e procura, são, no pior caso “lg(n+1)”.

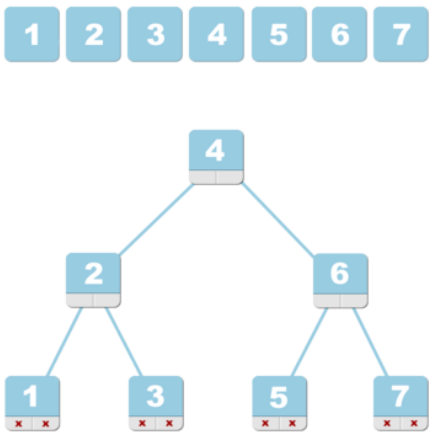
O trabalho proposto relaciona árvores binárias com código Morse, um sistema de representação de letras, algarismos e sinais de pontuação através de um sinal codificado enviado de modo intermitente. Seu objetivo é fazer um programa que converta mensagens de código Morse para mensagens de texto e vice-versa, na atividade, cada símbolo do alfabeto é codificado através de pontos (.) e traços (-) que são representados através de uma tabela auxiliar.

Com isso, tendo em vista as regras apresentadas pelo trabalho, a realização da implementação e a análise do código se torna uma forma intuitiva de aprendizado.

Implementação:

Iniciamos o trabalho realizando um estudo sobre árvores binárias e suas classificações, pois é necessário conhecer sua forma de construção para facilitar o entendimento da implementação através de códigos base disponibilizados pelo professor e também presentes em livros que relatam sobre linguagem C.

A imagem a seguir representa uma árvore binária de busca:



Através da imagem é possível perceber que a árvore é uma estrutura de dados baseada em nós, onde todos os nós da sub-árvore esquerda possuem um valor numérico inferior ao nó raiz, e todos os nós da sub-árvore direita possuem um valor superior ao nó raiz. O objetivo desta árvore é estruturar os dados de forma flexível, permitindo a pesquisa binária.

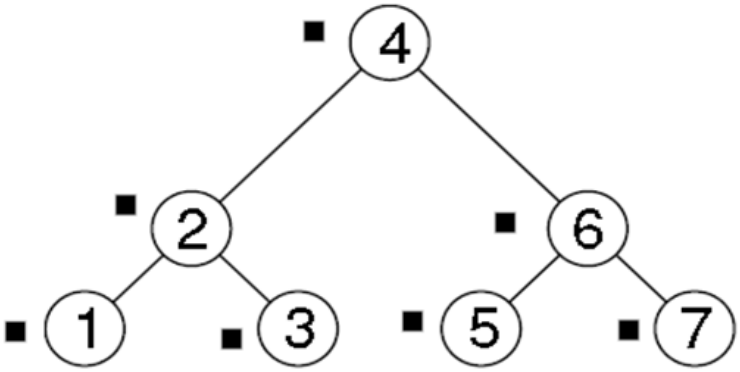
Nós - São todos os itens guardados na árvore;

Raiz – É o nó do topo da árvore (no caso da figura acima, a raiz é o número 4);

Folhas – São os nós que não têm filhos; são os últimos nós da árvore (no caso da figura acima, as folhas são 1, 3, 5 e 7).

Além disso, as árvores podem ser percorridas de 3 maneiras distintas:

Pré-ordem: Visita primeiro a raiz e depois vai para a sub-árvore esquerda e por último para a direita.



A imagem acima, apresenta uma forma mais didática sobre o funcionamento da pré-ordem, ao adicionar pontos à esquerda de cada nó, ficaria fácil perceber que a ordem a ser percorrida seria: 4 – 2 – 1 – 3 – 6 – 5 – 7.

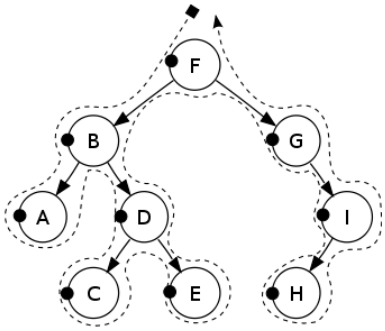
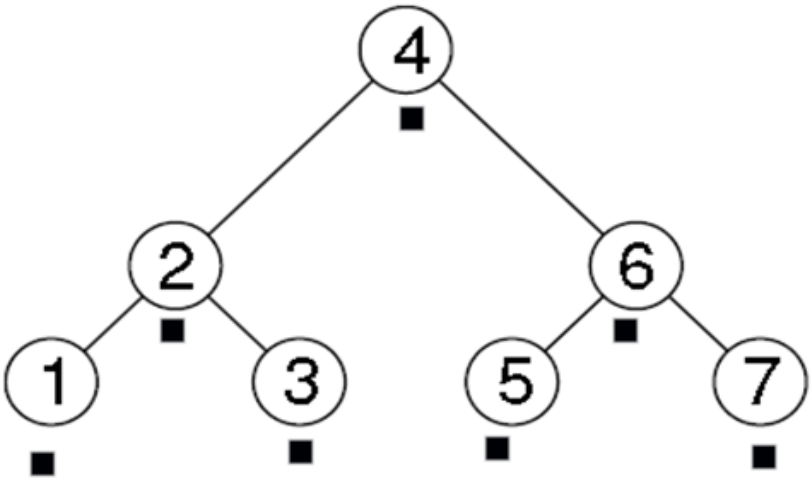


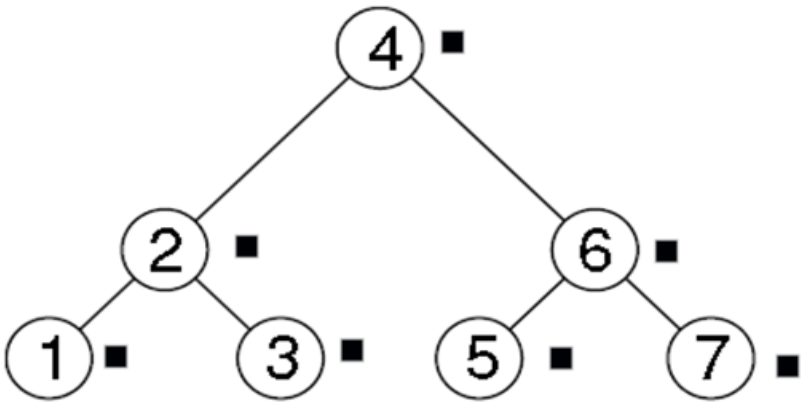
Imagem que apresenta um pequeno exemplo de como os pontos são percorridos**.**

Ordem: Visita a sub-árvore esquerda, depois a sub-árvore direita e por último visita a raiz.



A imagem acima, apresenta uma forma mais didática sobre o funcionamento da ordem, ou forma simétrica, ao adicionar pontos em baixo de cada nó, ficaria fácil perceber que a ordem a ser percorrida seria: 1 – 2 – 3 – 4 – 5 – 6 – 7.

Pós-ordem: Visita a sub-árvore a esquerda, depois a raiz e por último a sub-árvore a direita.



A imagem acima, apresenta uma forma mais didática sobre o funcionamento da pós-ordem, ao adicionar pontos à direita de cada nó, ficaria fácil perceber que a ordem a ser percorrida seria: 1 – 3 – 2 – 5 – 7 – 6 – 4.

É importante ressaltar também, que as árvores em que todos os níveis estão preenchidos são denominadas completas.

Para sabermos qual a altura de uma árvore binária completa de altura h, basta utilizarmos a fórmula **nós = log2 (n+1)/2** (para n igual à quantidade total de nós da árvore) .

Para sabermos se uma árvore binária é balanceada a altura da sub-árvore esquerda menos a altura da direita deve ser menor ou igual a 1 para todo nó **(He – Hd < = 1).**

TADs implementados:

Para a implementação do trabalho foi criado um tipo abstrato de dados que pode ser visto a seguir:

```
4   struct ArvoreMorse
5   {
6       char letra;
7       char simbolo;
8       struct ArvoreMorse* esq;
9       struct ArvoreMorse* dir;
10  };
11  typedef struct ArvoreMorse Arv;
```

Dessa forma, para implementar o que foi solicitado, criamos estruturas (structs) visto que essas nos permitem trabalhar com diversos tipos de informações de maneira rápida e organizada, e são muito utilizadas em programas que necessitam e fazem o uso de vários tipos de variáveis e características, além de possuírem auto referência (uso de ponteiros).

O tipo Arv contém 4 campos que armazenam caracteres e contém dois ponteiros do mesmo tipo.

struct ArvoreMorse* esq;

struct ArvoreMorse* dir;

Estes ponteiros farão a conexão entre os nós da árvore binária, e seus nomes feitos de forma intuitiva indicam o sentido das suas ramificações, para esquerda ou para a direita.

Funções e Procedimentos

Para o funcionamento do código, o programa foi dividido em dois arquivos .h e um arquivo .c, a fim de facilitar a sua implementação e entendimento, desta forma é mais fácil encontrar possíveis erros e fazer eventuais modificações de forma prática e eficiente.

****ArvoreMorse.h***

TraduzLetra.h

****TrabalhoAES3.c***

Os arquivos possuem as seguintes funções:

Arv* iniciaArvore();

Tem o objetivo de iniciar a árvore.

Arv* criaArvore (Arv* item, char simbolo, char letra);

A função criaArvore contém três parâmetros de entrada, sendo um ponteiro Arv *item, uma variável char símbolo, e uma variável char letra.

Esta função recebe um ponteiro contendo o endereço onde deve ser inserido um novo nó que será criado na função. Logo, o ponteiro recebido é apontando para um espaço alocado para um novo nó, e em seguida este nó é preenchido com os caracteres recebidos, e seus ponteiros esq e dir são apontados para NULL. Ao final das operações a função retorna o ponteiro.

Arv* insereElemento (Arv* item, char simbolo, char letra);

A função insereElemento recebe três parâmetros de entrada, Arv* item, char simbolo e char letra. Essa função serve para inserir um novo elemento na árvore binária. Então através de testes com os valores recebidos, é determinado se o nó deverá ser inserido na direita ou na esquerda, então desta forma, o ponteiro recebido, um de seus ponteiros, recebe um novo nó, por exemplo item->esq ou item->dir, então ele recebe o retorno da função que cria o nó, ex.: item->esq = criaArvore(item->esq, simbolo, letra); e então este ponteiro é retornado.

return item->esq; ou return item->dir, caso não seja nenhum dos dois casos a função retorna 0.

void percorre(Arv* item, char palavra);

A função percorre recebe dois parâmetros de entrada, um ponteiro Arv* item, e uma variável char palavra. Então é testado, se o ponteiro recebido aponta para NULL, caso não aponte, a função é chamada novamente, em uma recursão. Caso a chamada da função termine e NULL seja encontrado, a chamada da função termina e volta para a chamada anterior, sendo que ao atingir o nó desejado é testado, se o caractere recebido como parâmetro é igual ao contido no nó, caso seja então a função TraduzLetra é chamada, recebendo o caractere em questão, sendo que esta função será responsável por printar o Morse correspondente a esse caractere recebido.

void imprime(Arv* item);

A função imprime, tem como parâmetro de entrada um ponteiro Arv* item, que recebe o endereço do nó raiz da árvore binária. Então é testado se esse ponteiro aponta para NULL, que serve para parar a chamada recursiva, caso não seja NULL, então é executado um bloco de código. Neste bloco de código é testado se o elemento presente no nó em questão é a letra 'm', este caractere é utilizado para outros fins e não deve ser printado o seu conteúdo pois ele apenas faz parte de um dos nós do caminho, e caso não seja o 'm' o caractere é printado na tela, então é passado o caractere presente neste nó para a função TraduzLetra que também irá printar a conversão deste caractere em Morse.

void TraduzLetra(char letra);

A função TraduzLetra recebe como parâmetro uma passagem por valor, ou seja, um caractere externo que é copiado para char letra. Então através dessa variável que contém um caractere alfanumérico é utilizado um switch case para a conversão para código Morse. De acordo com o valor do caractere é verificado em qual dos casos ele

se encaixa, sendo ele uma letra do alfabeto (apenas letras maiúsculas), ou um número de 0 a 9 então é printada a respectiva conversão do caractere. Caso não seja nenhum dos caracteres possíveis de realizar a conversão a função não faz nada.

Programa Principal:

A main, principal função do programa é por onde o programa começa e termina, ela concentra todas as principais ações e chamadas de funções.

No início são criados alguns ponteiros, FILE *fp, *fp2, *fp3, *fp4 que serão cruciais para as ações seguintes, e também são declaradas as variáveis necessárias.

Então nas próximas linhas de código é lido o arquivo DicionarioMorse.txt que contém todos os caracteres alfanuméricos e sua respectiva conversão em Morse, que serão lidos no programa e utilizados para a construção da árvore. Para cada letra alfanumérica existe um caractere em Morse que representa o seu caminho na árvore binária, a depender dos pontos e traços do caractere em Morse o um ponteiro é utilizado para percorrer a árvore, sendo que percorre para a esquerda caso o caractere lido seja um ponto, e percorre para a direita caso o caractere lido seja um traço.

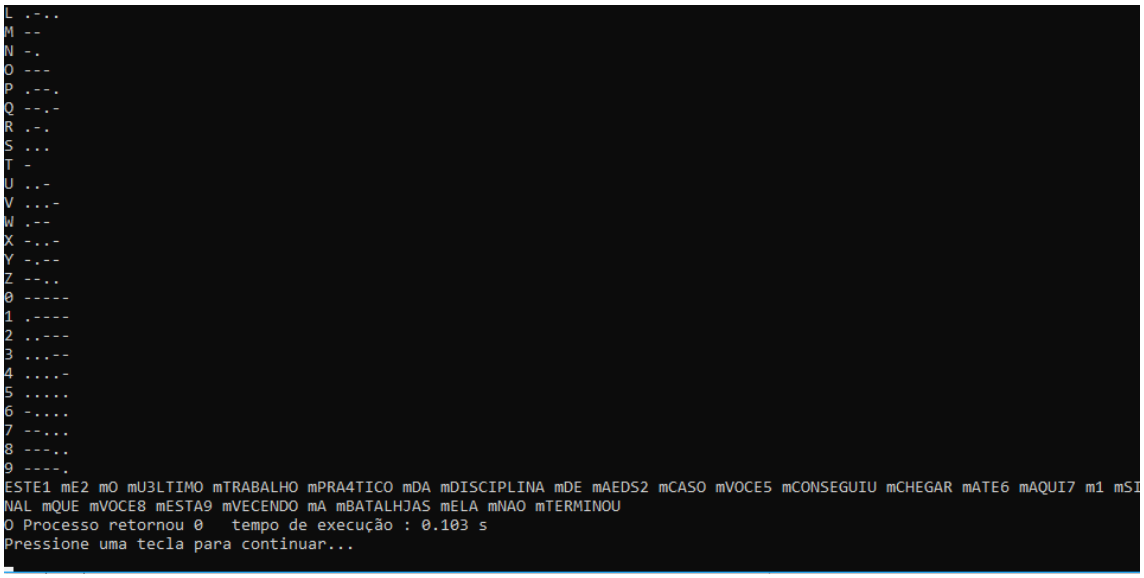
Ao final do caminho lido será encontrado um ponteiro que aponta para NULL, e esse ponteiro recebe um endereço retornado da função CriaArvore, que aloca espaço e insere um novo nó.

Logo abaixo, após a construção da árvore é exibido um menu do usuário a fim de coletar um valor que representa uma ação a ser tomada. Caso a opção escolhida seja 1, é lido o arquivo que contém os dados a serem convertidos, esta mensagem é exibida na tela e em seguida é chamada a função de conversão de código Morse para alfanumérico.

Caso a opção seja 2, é lido o arquivo que também contém uma mensagem a ser convertida então é chamada a função de conversão de alfanumérico para Morse, e caso a opção requisitada seja 3, é chamada a função que imprime a arvore binária, sendo exibido o caractere alfanumérico armazenado nos nós da arvore e sua respectiva conversão em Morse. Caso a opção não seja nenhuma das acima, é exibida uma mensagem de opção inválida. Por fim é retornado zero para indicar o fim do programa.

Testes

Para a criação do código foram utilizados dois aplicativos:
Falcon (versão para Windows)
Atom (versão para Linux)
Os testes foram realizados em dois computadores
– AMD Quad-Core A12-9720P RADEON R7 12 COMPUTE CORES 4C+8G 2.70GHz
– Intel(R)Core(TM)i3-4005 CPU @ 1.70GHz k1.70GHz
A figura abaixo mostra uma saída com erros na execução que foram resolvidos durante a criação do programa.



Conclusão

Dessa forma, tendo em vista tudo o que foi proposto, os estudos em sala e externos, as discussões sobre o tema e as implementações realizadas, foi possível finalizar o trabalho.

A respeito das dificuldades encontradas pelo caminho, essas que contribuíram para o atraso da entrega do trabalho, houve estudos paralelos a partir de leituras (obra declarada em referências), acesso a videoaulas disponibilizadas online e também o comparecimento da dupla na monitoria disponibilizada pelo orientador e pela universidade, para a resolução e melhor compreensão dos erros e acertos durante a implementação.
Dessa forma, concluímos que a realização do trabalho foi de suma importância para o aprendizado sobre árvores binárias e pesqusa, visto que trabalhamos com diversas implementações até chegarmos no código correto e entregamos o trabalho com maior facilidade em enxergar soluções para os problemas propostos pelo assunto.

Referências

Obra *C completo e total - terceira edição*
Autor Herbert Schildt – Osborne
www.cprogressivo.net

Canal no Youtube: Linguagem C descomplicada.

Embasamento em sala de aula – Professor Rafael Sachetto UFSJ