MARCUS VINICIUS ROLIM DE OLIVEIRA

ÁRVORE BINÁRIA

Campo Grande - MS
2021

INTRODUÇÃO

O programa sugerido para o desenvolvimento se trata de uma árvore binária, onde se utiliza uma estrutura, onde os nós de alocação de memória são divididos em três partes, uma parte com a informação do dado a ser alocado, e dois ponteiros, onde cada ponteiro direciona para uma direção, um para a esquerda e o outro para a direita.

Assim temos uma árvore binária, pois um nó pode se ter dois caminhos para outros nós, então cada nó, pode ter dois filhos, ou no caso como é uma árvore binária, tem duas folhas.

A função do programa é a facilidade de inserir, buscar e remover dados de uma árvore, fazendo muito mais efetivo comparado com as listas encadeadas, duplamente encadeadas, filas e pilhas.

Existem três formas de busca: pré-ordem, em-ordem e pós-ordem. Podendo ser utilizado de acordo com a melhor forma de adequar a finalidade que se utilizará o programa.

Para um entendimento geral, pode se dizer que criando um primeiro nó, onde é a raiz da árvore, inicia a árvore onde todos os dados menores que a raíz irão para o lado esquerdo e os maiores irão para o lado direito, criando assim um padrão que é utilizado para facilitar as buscas.

IMPLEMENTAÇÃO

O programa desenvolvido teve como estrutura base de alocação de dados na memória a estrutura de árvore binária.

Árvore binária consiste em um nó tendo dois caminhos, esquerda ou direita, fazendo assim uma árvore sempre com duas folhas, onde sempre o caminho esquerdo do nó será alocado um novo dado menor que o nó anterior, e no lado direito será o dado maior que o dado anterior.

Na prática iniciamos com inserção das bibliotecas que serão utilizadas através do comando include, que são as bibliotecas stdio.h, que utilizaremos os comandos de entrada e saída de dados, printf, scanf e a biblioteca stdlib.h que usaremos o comando malloc, para a alocação de espaço na memória.

A alocação do espaço da memória montando a estrutura do nó antes de iniciar o programa principal, que é dividido em três partes, uma parte com o dado e duas partes com ponteiros, criando assim a estrutura base da árvore.

Iniciando o programa deve-se executar o comando de criação de uma árvore, como será o primeiro nó da árvore chamado de raiz, criando assim um ponteiro com valor nulo (NULL) indicando o ponteiro da raiz.

O primeiro nó é criado através de uma função utilizando o comando malloc, onde se verifica se já existe um nó raiz, caso não exista se aloca o espaço necessário, caso contrário retorna o valor da raiz existente. Assim no programa principal (main), se cria o ponteiro direcionando para o novo nó raiz criado.

A árvore pode ser medida por altura, onde é a quantidade de camadas que ela tem, iniciando da raiz contando de 1 em diante, e também pode ser medida por níveis, onde se conta a quantidade de camadas, porém iniciando do zero, pois a raíz não conta como nível.

Existem formas de saber o tamanho da árvore através da contagem de quantos nós se tem, e essa contagem pode ser feita de três formas, para não se perder na contagem, contagem seguindo um caminho tipo pré-ordem, o pós-ordem e o em-ordem.

Que é o recurso do programa que vamos utilizar na opção 1 do menu, onde imprimimos na tela todos os nós da árvore, acionando a opção de imprimir do primeiro menu, se abre o segundo menu mostrando essas três opções para serem mostradas.

Os menus são mostrados utilizando o recurso do switch case, onde cada opção escolhida é disparado um caso específico, tudo isso feito através de funções externas do programa principal.

O que é importante de se explicar é que não se pode ter números repetidos em uma árvore binária de busca.

Fazendo a busca de forma pré-ordem, é feito a validação da existência da raiz da árvore, caso não exista, retorna para o programa sem fazer nenhuma ação, caso exista, é impresso o número da raiz, e em seguida avança para o lado esquerdo, imprime o nó esquerdo, e avança para o lado esquerdo, caso o lado esquerdo seja nulo (NULL), avança para o nó anterior e vai para o lado direito, caso tenha lado direito, imprime o lado direito, e avança para o lado esquerdo, e assim sucessivamente, até ter impresso todos os nós da árvore.

Já na busca em-ordem, inicia a validação como na anterior, caso valide positivo, inicia com a diferença de avançar para o lado esquerdo, sem imprimir nada, até chegar um nó com lado esquerdo nulo (NULL), assim que chegar no nulo, volta para o anterior e imprime, e em seguida vai para o lado direito, e chegando no lado direito, vai para o esquerdo até que seja nulo (NULL), e volta e imprime, sucessivamente até acabar a árvore.

Finalizando os modelos de buscas do programa existe o pós-ordem, que como os outros após a validação positiva da raiz, seguindo para o lado esquerdo até o

final, pulando para o direito após chegar no esquerdo nulo (NULL), caso o direito também seja nulo (NULL), ele imprime o nó anterior, e volta para o anterior, buscando o direito novamente, até completar toda a árvore.

São essas três opções que existem no submenu da opção 1 do menu principal, onde está definida como “1. Imprimir”.

Já na opção dois, inserimos um nó na árvore, ou podemos chamar de folha, através de uma função externa que será executada no caso 2, do switch case do menu principal.

A função de adicionar uma nova folha na árvore, irá receber dois dados, um o ponteiro da raiz, e o outro o valor a ser inserido, assim iniciamos a função com sua validação da existência ou não da raiz, caso ela seja nula (NULL), retorna para o programa sem ação alguma, caso a raiz exista, se cria uma nova estrutura utilizando o comando malloc, para se adicionar o valor no nó, e nulo nos ponteiros para assim iniciar o procedimento de busca onde será inserido esse nó.

Iniciando a busca para a inserção, valida o ponteiro da raiz em uma estrutura condicional, caso seja nulo (NULL), já atribui o novo nó ali mesmo, caso não seja nulo utilizamos o artifício de mais duas estruturas de nó auxiliares, onde um se recebe o valor do ponteiro da raiz e o outro se atribui o valor nulo (NULL).

Entrando em uma estrutura de repetição do tipo while, pode-se fazer o comparativo onde o ponteiro auxiliar que mostra o ponteiro da raiz, seja diferente de nulo (NULL), o segundo nó auxiliar recebe o valor também do ponteiro da raiz, e em seguida entra em outra estrutura condicional, if, onde se faz o comparativo se o dado escolhido tenha o mesmo dado da raiz, ele é excluído, pois não se pode ter o mesmo valor na árvore e retorna sem fazer ação. Caso o dado escolhido seja maior que o dado da raiz, o primeiro nó auxiliar que tem o ponteiro da raiz, é atribuído um novo valor, ele começa a ter o valor do ponteiro da direita dele, caso seja menor ele é alocado no ponteiro da esquerda.

Em outra condicional, em sequência faz o comparativo de que se o dado seja maior que o segundo nó auxiliar, o ponteiro da direita é atribuído o novo dado, caso contrário o ponteiro da esquerda que recebe o novo dado.

Com o fim desse processo, é enviado uma mensagem de sucesso ou de dado já existente para o usuário.

Entrando na terceira opção do menu, temos a remoção do dado que é feito através de duas funções externas, onde uma interliga na outra e a primeira é chamada dentro do case 3 do switch case do menu principal.

A primeira função se inicia com dois dados recebidos, um é o ponteiro da raiz e o outro é qual o dado a ser removido. Recebendo esses dados através do menu, utilizando o printf para perguntar e o scanf para guardar o dado, fazemos a validação se a raiz é nula (NULL), caso seja retorna sem ação nenhuma, através do comando return, caso a raiz exista, utilizaremos os mesmos artifícios da adição.

Criamos dois nós auxiliares, onde o primeiro se adiciona o ponteiro da raiz e o segundo se atribui o valor nulo (NULL), em seguida entramos em uma estrutura de repetição while, onde o primeiro auxiliar com o valor do ponteiro da raiz, seja diferente de nulo (NULL) entrará em uma estrutura condicional if/else, onde o valor deve ser igual ao valor do dado do nó auxiliar, caso seja entra em outra condicional, que se compara com o auxiliar inicial seja igual ao ponteiro da raiz, caso positivo, o ponteiro da raiz recebe o valor de uma função que será criada para remover o nó auxiliar, caso de negativa, se entra em outra condicional, fazendo com que o ponteiro do segundo auxiliar da direita seja igual ao primeiro auxiliar, o ponteiro do segundo auxiliar direito executará a função de remoção, em caso de negativo, será o ponteiro do segundo auxiliar esquerdo que executará a função de remoção, finalizando retornando para o programa.

Caso não entre no primeiro comparativo onde o dado seja igual ao dado da raiz, o segundo nó auxiliar recebe o valor do primeiro nó auxiliar e inicia outra estrutura condicional, onde se o valor seja maior que o da raiz, o primeiro nó auxiliar

recebe o valor do ponteiro da direita, caso seja menor, receberá o valor do ponteiro da esquerda. Finalizando assim e retornando para o programa.

A segunda função que na função acima será requisitada para a exclusão do nó é feita através de uma nova estrutura, onde se cria uma função do tipo nó, onde recebe o valor da estrutura do primeiro nó auxiliar e em seguida se cria mais dois nós, um terceiro e um quarto e inicia uma estrutura condicional onde a primeira condição é caso o valor do ponteiro do primeiro auxiliar a esquerda seja nulo (NULL), o terceiro nó auxiliar recebe o valor do ponteiro da direita do primeiro nó auxiliar e em seguida é excluído o primeiro nó auxiliar, retornando o valor do terceiro nó auxiliar.

Caso não entre nessa condição, o terceiro nó auxiliar recebe o valor do primeiro nó auxiliar e o quarto nó auxiliar recebe o valor do ponteiro esquerdo do primeiro nó auxiliar.

Em seguida se desenvolve uma estrutura de repetição tipo while, onde a regra para ser executada é do quarto nó auxiliar seja diferente de nulo (NULL), caso entre o terceiro nó auxiliar recebe o valor do quarto nó auxiliar e o quarto nó auxiliar recebe o valor do ponteiro do quarto nó a direita.

Continuando a função se cria uma condição onde se o terceiro nó auxiliar seja diferente do primeiro nó auxiliar, o ponteiro do terceiro nó auxiliar a direita recebe o valor do ponteiro do quarto nó a esquerda, e o ponteiro do quarto nó a esquerda, recebe o valor do primeiro nó auxiliar a esquerda.

Sendo negativa essa condição, o ponteiro do quarto nó a direita recebe o valor do ponteiro do primeiro nó auxiliar a direita e em seguida se exclui o primeiro nó auxiliar e retorna o valor do quarto nó auxiliar, para se utilizar na função de exclusão.

E por fim é mostrado para o usuário mensagem de sucesso na remoção ou a falha na busca por não existir o número na árvore.

Entrando na última opção do menu é o de sair, onde simplesmente fecha o programa.

CONCLUSÃO

A conclusão que se obtém do programa é a facilidade que se tem com a busca de informações em uma memória, pois a forma que se é organizado os dados inseridos, fazem com que a busca seja muito mais rápida, otimizando o tempo de espera e a quantidade de processamento.

Comparando com as listas duplamente encadeadas, que tem o mesmo tamanho estrutural, com um nó dividido em 3 partes, com a diferença de que a lista duplamente encadeada vai para frente ou para trás, obrigando sempre a busca sequencial normal, a árvore binária, pode se buscar indo para dois lados, um maior e um menor, fazendo assim a economia de busca.

Porém também vemos muitas barreiras como não poder adicionar o mesmo valor na árvore.

Utilizando o gancho descobrindo como a árvore funciona, dá para entender por que a árvore é a forma mais utilizada de implementação na arquitetura paralelas e distribuídas.

A otimização de processamento traz muitos benefícios para a aplicação principalmente por se tratar de poder de processamento que é o maior gargalo da programação e principalmente o custo.

Outra grande vantagem das árvores também é a forma de se expressar através da lógica de programação através dos algoritmos, que no caso utilizando figuras, como fluxograma, fica muito mais fácil de se entender e ocupando muito menos espaço, pois no desenho ele ocupa proporcionalmente altura x largura, diferente de uma lista ou fila, que demanda muito mais espaço horizontal, onde muitas vezes fica muito mais complicado de se analisar grandes números de nós.

BIBLIOGRAFIA

- Material disponibilizado pelo professor, slides, aulas virtuais da matéria de Algoritmo e Estrutura de Dados Avançados.
- Trabalho para a N2 do terceiro semestre da matéria de Algoritmo e Estrutura de Dados, sobre lista encadeada, duplamente encadeada, fila e pilha.