**TAD – ÁRVORES**

**Conceitos**

As árvores são estruturas de dados baseadas em listas encadeadas que possuem um nó superior também chamado de raiz que aponta para outros nós, chamados de nós filhos, que podem ser pais de outros nós.

Uma árvore de busca binária tem as seguintes propriedades:

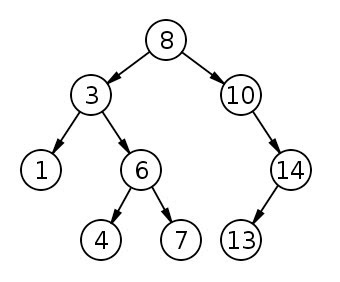
* todos os elementos na subárvore esquerda de um determinado nó n são menores que n;
* todos os elementos na subárvore direita de um determinado nó n são maiores ou iguais a n.

Na **Figura a seguir** temos uma ilustração de um exemplo de árvore:

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

**Implementação**



Na figura acima tem-se uma árvore binária onde a raiz é o elemento 8, o filho da esquerda do elemento 8 é o elemento 3, o filho da direita é o elemento número 10. Nota-se que todos elementos da árvore binária possuem no máximo dois filhos, sendo o da esquerda sempre menor e o da direita sempre maior que o elemento pai.

Vejamos abaixo os algoritmos para inserção e exclusão em árvores binárias implementados na linguagem Java.

|  |  |
| --- | --- |
|  | public void inserir(No node, int valor) {    //verifica se a árvore já foi criada     if (node != null) {      //Verifica se o valor é menor que o nodo, vai para subárvore esquerda      if (valor < node.valor) {          //Se tiver elemento no nodo esquerdo continua a busca          if (node.esquerda != null) {              inserir(node.esquerda, valor);          } else {              //Se nodo esquerdo vazio insere o novo nodo aqui              System.out.println(valor + " a esquerda de " + node.valor);              node.esquerda = new No(valor);          }      //Verifica se o valor é maior que o nodo, vai para sub árvore direita      } else if (valor > node.valor) {          //Se tiver elemento no nodo direito continua a busca          if (node.direita != null) {              inserir(node.direita, valor);          } else {              //Se nodo direito vazio insere o novo nodo aqui              System.out.println(valor + " a direita de " + node.valor);              node.direita = new No(valor);          }      }    }  } |

Pode-se notar que o algoritmo funciona recursivamente, chamando a função repetidas vezes. Os detalhes do código estão exemplificados através dos comentários no seu corpo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public No removeDaArvore(No node) {      if (node == null) {          System.out.println("  ERRO ");      } else if (node.esquerda != null) {          node.esquerda = removeDaArvore(node.esquerda);          return node;      } else {        //Se não tiver elemento esquerdo só nos resta o da direita          return node.direita;      }      return null;  } |

No código acima nota-se que o menor elemento da árvore está sendo excluído, o código também poderia ser implementado recebendo um nó a ser excluído, procurar este nó e excluir da árvore. O caminhamento em árvores será descrito a seguir. Nota-se que os elementos filhos tomam a posição do elemento excluído.

**Caminhamentos**

Uma árvore binária pode ser percorrida utilizando caminhamento prefixado, pós-fixado e em ordem, todos serão detalhados a seguir.

**Caminhamento Prefixado**

Basicamente no caminhamento prefixado será utilizado o algoritmo abaixo:

* Executa-se a ação a ser realizada Recursivamente percorre-se a subárvore esquerda
* Recursivamente percorre-se a subárvore direita

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void prefixado(No no) {      if(no != null){          System.out.print(no.valor + " ");          prefixado(no.esquerda);          prefixado(no.direita);      }  } |

Para o exemplo da árvore acima ter-se-ia a seguinte ordem de visita dos nós: 8, 3, 1, 6, 4, 7, 10, 14, 13.

Passos de manipulação:

* imprime(8) //inicialmente executa-se a impressão do valor
* preordem(8.esquerda) //chama prefixado(no.esquerda);
* Imprime(3) //ao chamar novamente o método imprime o 3 que é o nó corrente
* preordem(3.esquerda) //chama prefixado(no.esquerda);
* Imprime(1) //ao chamar novamente o método imprime o 1 que é o nó corrente
* preordem(1.esquerda) //chama prefixado(no.esquerda);
* preordem(1.dir) //como prefixado(no.esquerda) retornou null, executa-se agora o prefixado(no.direita)
* preordem(3.dir) //como prefixado(no.direita) é null, volta-se para o elemento anterior que é o 3 (antes do 1). Como o 1 também já executou o prefixado(no.esquerda) executa-se agora o prefixado(no.direita)
* Imprime(6) //ao chamar novamente o método imprime o 6 que é o nó corrente
* preordem(6.esquerda) //chama prefixado(no.esquerda) que ainda não foi chamado para este elemento.
* Imprime(4) //ao chamar novamente o método imprime o 4 que é o nó corrente
* preordem(4.esquerda) //chama prefixado(no.esquerda) que ainda não foi chamado para este elemento.
* preordem(4.direita) //como prefixado(no.esquerda) retornou null, executa-se agora o prefixado(no.direita).
* preordem(6.direita) //como prefixado(no.direita) é null, volta-se para o elemento anterior que é o 6 (antes do 4). Como o 6 também já executou o prefixado(no.esquerda) executa-se agora o prefixado(no.direita)
* Imprime(7) //ao chamar novamente o método imprime o 7 que é o nó corrente.
* preordem(7.esquerda) //chama prefixado(no.esquerda) que ainda não foi chamado para este elemento.
* preordem(7.direita) //como prefixado(no.esquerda) retornou null, executa-se agora o prefixado(no.direita).
* preordem(8.direita) //como prefixado(no.direita) é null, volta-se para o elemento anterior que é o 8. Como o 8 também já executou o prefixado(no.esquerda) executa-se agora o prefixado(no.direita)
* Imprime(10) //ao chamar novamente o método, imprime-se o 10 que é o nó corrente
* preordem(10.esquerda) //chama prefixado(no.esquerda) que ainda não foi chamado para este elemento.
* preordem(10.direita) //como prefixado(no.esquerda) retornou null, executa-se agora o prefixado(no.direita).
* Imprime(14) //ao chamar novamente o método, imprime-se o 14 que é o nó corrente
* preordem(14.esquerda) //chama prefixado(no.esquerda) que ainda não foi chamado para este elemento.
* Imprime(13) //ao chamar novamente o método, imprime-se o 13 que é o nó corrente
* preordem(13.esquerda) //chama prefixado(no.esquerda) que ainda não foi chamado para este elemento.
* preordem(13.direita) //como prefixado(no.esquerda) retornou null, executa-se agora o prefixado(no.direita).
* preordem(14.direita) //como prefixado(no.direita) é null, volta-se para o elemento anterior que é o 14. Como o 14 também já executou o prefixado(no.esquerda) executa-se agora o prefixado(no.direita) que também é null. Agora nota-se que não há mais nada na pilha, todos os elementos e seus filhos esquerdos e direitos já foram executados, portanto está finalizado o algoritmo.

**Caminhamento Pós-fixado**

Basicamente no caminhamento Pós-fixado será utilizado o algoritmo abaixo:

* Recursivamente percorre-se a subárvore esquerda
* Recursivamente percorre-se a subárvore direita
* Executa-se a ação a ser realizada

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void posfixado(No no) {      if(no != null){        posfixado(no.esquerda);        posfixado(no.direita);          System.out.print(no.valor + " ");      }  } |

Para o exemplo da árvore acima ter-se-ia a seguinte ordem de visita dos nós: 1, 4, 7, 6, 3, 13, 14, 10, 8.

**Caminhamento Em Ordem**

Basicamente no caminhamento Em Ordem será utilizado o algoritmo abaixo:

* Recursivamente percorre-se a subárvore esquerda
* Executa-se a ação a ser realizada
* Recursivamente percorre-se a subárvore direita

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void emordem(No no) {      if(no != null){        emordem(no.esquerda);  System.out.print(no.valor + " ");        emordem(no.direita);      }  } |

Para o exemplo da árvore acima ter-se-ia a seguinte ordem de visita dos nós: 1, 3, 4, 6, 7, 8, 10, 13, 14.

**Aplicações**

Árvores binárias são largamente utilizadas em diversas aplicações. Entre as aplicações pode-se citar as árvores de decisão usadas na Inteligência Artificial. Outra aplicação é na representação de expressões aritméticas. No caso da representação das expressões aritméticas pode-se utilizar um caminhamento pós-fixado para resolver o problema, onde, por exemplo, para uma árvore binária de expressões aritméticas ter-se-ia para cada nó externo um valor associado e para cada nó interno um operador aritmético associado, esse algoritmo calcularia facilmente o resultado da expressão. Sugerimos ao leitor tentar este exercício e em caso de dúvidas contate o autor para fornecer as respostas para as dúvidas.