**Usos del recorrido en orden:**

En el caso de los árboles de búsqueda binaria (BST), el recorrido en orden proporciona nodos en orden no decreciente. Para obtener los nodos de BST en orden no creciente, se puede utilizar una variación del recorrido en orden en el que se invierte el recorrido en orden.

Implementación de código de recorrido Inorder.

|  |
| --- |
| // Java program for different tree traversals  // Class containing left and right child of current  // node and key value  **class** Node {  **int** key;      Node left, right;    **public** Node(**int** item)      {          key = item;          left = right = **null**;      }  }    **class** BinaryTree {      // Root of Binary Tree      Node root;      BinaryTree() { root = **null**; }      // Given a binary tree, print its nodes in inorder  **void** printInorder(Node node){  **if** (node == **null**)  **return**;            // First recur on left child          printInorder(node.left);            // Then print the data of node          System.out.print(node.key + " ");            // Now recur on right child          printInorder(node.right);      }      // Driver code  **public** **static** **void** main(String[] args){          BinaryTree tree = **new** BinaryTree();          tree.root = **new** Node(1);          tree.root.left = **new** Node(2);          tree.root.right = **new** Node(3);          tree.root.left.left = **new** Node(4);          tree.root.left.right = **new** Node(5);            // Function call          System.out.println(              "Inorder traversal of binary tree is ");          tree.printInorder(tree.root);      }  } |

**Usos del pedido anticipado:**

El recorrido de pedido anticipado se utiliza para crear una copia del árbol. El recorrido de preorden también se utiliza para obtener expresiones de prefijo en un árbol de expresiones.

Implementación del código del recorrido de pedido anticipado:

|  |
| --- |
| // Java program for different tree traversals  // Class containing left and right child of current  // node and key value  **class** Node {  **int** key;      Node left, right;    **public** Node(**int** item)      {          key = item;          left = right = **null**;      }  }    **class** BinaryTree {        // Root of Binary Tree      Node root;        BinaryTree() { root = **null**; }        // Given a binary tree, print its nodes in preorder  **void** printPreorder(Node node){  **if** (node == **null**)  **return**;            // First print data of node          System.out.print(node.key + " ");            // Then recur on left subtree          printPreorder(node.left);            // Now recur on right subtree          printPreorder(node.right);      }        // Driver code  **public** **static** **void** main(String[] args){          BinaryTree tree = **new** BinaryTree();          tree.root = **new** Node(1);          tree.root.left = **new** Node(2);          tree.root.right = **new** Node(3);          tree.root.left.left = **new** Node(4);          tree.root.left.right = **new** Node(5);            // Function call          System.out.println(              "Preorder traversal of binary tree is ");          tree.printPreorder(tree.root);      }  } |

**Usos del postorder:**

El recorrido posterior al pedido se utiliza para eliminar el árbol. Consulte la pregunta sobre la eliminación de un árbol para obtener más detalles. El recorrido posterior al orden también es útil para obtener la expresión postfija de un árbol de expresión.

A continuación se muestra la implementación de los métodos transversales anteriores:

|  |
| --- |
| // Java program for different tree traversals  // Class containing left and right child of current  // node and key value  **class** Node {  **int** key;      Node left, right;    **public** Node(**int** item)      {          key = item;          left = right = **null**;      }  }  **class** BinaryTree {        // Root of Binary Tree      Node root;        BinaryTree() { root = **null**; }        // Given a binary tree, print its nodes according to the      // "bottom-up" postorder traversal.  **void** printPostorder(Node node){  **if** (node == **null**)  **return**;            // First recur on left subtree          printPostorder(node.left);            // Then recur on right subtree          printPostorder(node.right);            // Now deal with the node          System.out.print(node.key + " ");      }        // Driver code  **public** **static** **void** main(String[] args){          BinaryTree tree = **new** BinaryTree();          tree.root = **new** Node(1);          tree.root.left = **new** Node(2);          tree.root.right = **new** Node(3);          tree.root.left.left = **new** Node(4);          tree.root.left.right = **new** Node(5);            // Function call          System.out.println(              "Postorder traversal of binary tree is ");          tree.printPostorder(tree.root);      }  } |