ESTRUCTURA DE DATOS

Práctica Laboratorio 5

Grado en Ingeniería Informática

Jordi Lazo Florensa

Alejandro Clavera Poza

Universitat de Lleida
Escola Politècnica Superior

Tarea 1: Implementación de árboles binarios

Esta tarea consiste en implementar un árbol binario mediante árboles enlazados. Para esto contamos con una serie de nodos que son los encargados de almacenar tanto el elemento raíz de los árboles como sus hijos izquierdos y derechos.

Para esto hemos implementado la clase *LinkedBinaryTree* donde hemos realizado diferentes operaciones como la operación *equals* que compara si dos árboles son iguales, para esto inicialmente se comprueba si el árbol a comparar es una instancia de *LinkedBinaryTree* en el caso que no lo sea podemos decir que los árboles no son iguales. Por el contrario, comprobamos si los árboles son iguales mediante la operación *equals* que hemos implementado en la clase anidada *Node* la cual comprueba si el elementos raíz de los nodos son es iguales, después de esto realiza la misma acción de forma recursiva con el resto de nodos hijo hasta recorrer el árbol entero o bien que se encuentre alguna diferencia lo cual significa que los arboles no son iguales. La operación *isEmpty* comprueba si el árbol este vacío, para eso comprueba si su raíz es *null*. Para finalizar implementamos las operaciones *left* y *right* devuelven el árbol hijo izquierdo o derecho, respectivamente.

Tarea 2: Recorridos iterativos en árboles binarios

Esta tarea consiste en realizar los recorridos de un árbol binario *preorder, inorder, postorder*,

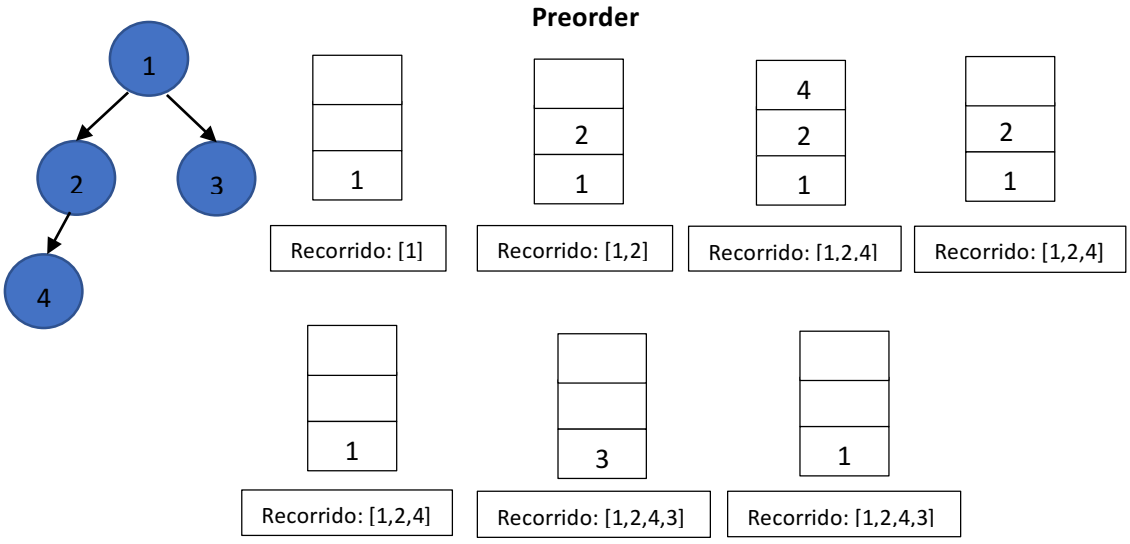

Figura 1

Para realizar la solución de este recorrido como podemos observar en la figura 1 comenzamos en el nodo raíz y vamos añadiendo al recorrido todos los elementos de los hijos izquierdos. En el momento en el se llega a un hijo izquierdo que no tienes mas hijos izquierdos, ese nodo se elimina de la pila y se recorre el árbol derecho hijo de la misma manera que se ha recorrido anteriormente. Este recorrido acaba cuando la pila está vacía y se haya terminado de analizar todos los árboles hijos. Todos los elementos son añadidos a la pila en el momento que son leídos.

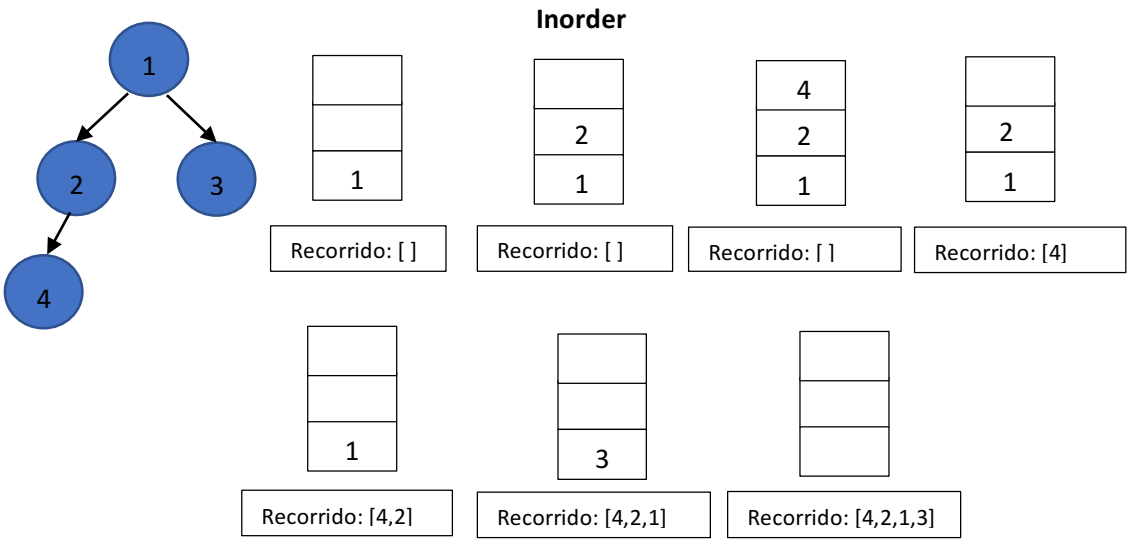

Figura 2

Por otro lado hemos realizador el recorrido *inorder* de la misma forma que el *preorder* con la pequeña diferencia que el elemento se añade al recorrido cuando este se elimina de la pila, es decir, en el momento que se ha acabado de recorrer el árbol hijo izquierdo y se va a comenzar a recorrer su árbol hijo derecho.

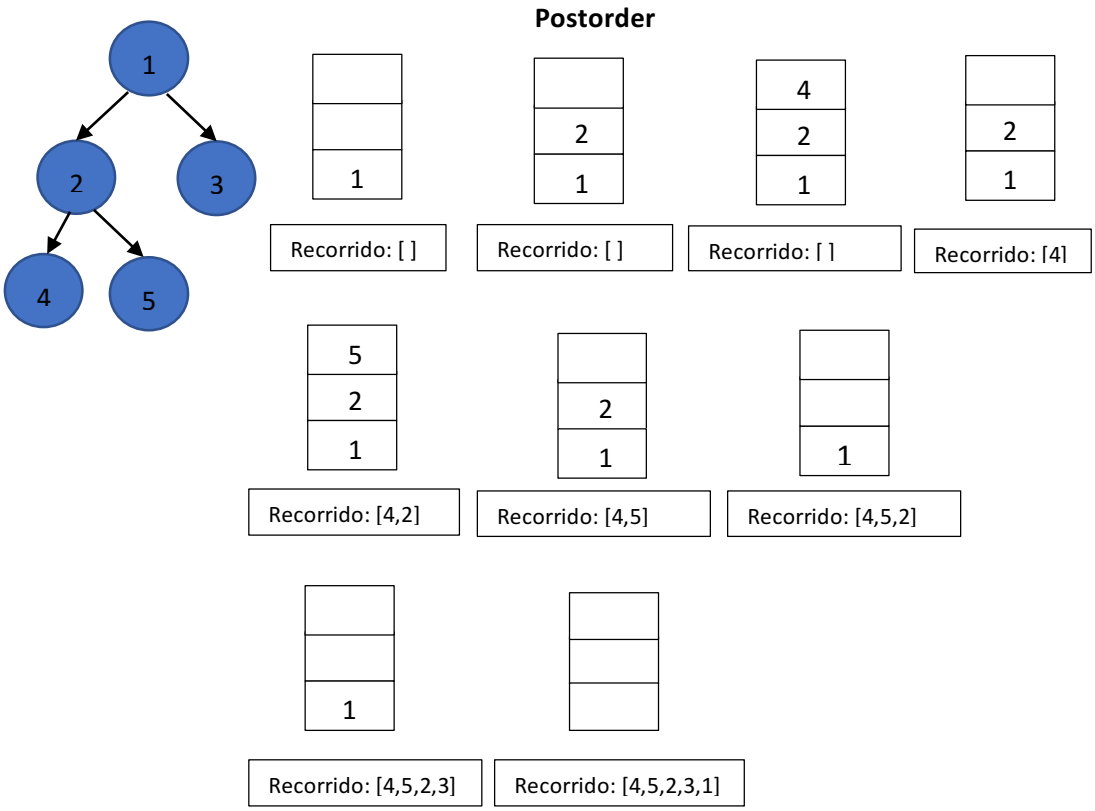

Figura 3

En cuanto al *postorder*, nos movemos por el árbol de la misma forma que los métodos anteriores con la diferencia que este añadirá un elemento al recorrido en el momento que se ha recorrido

tanto el hijo izquierdo como el derecho. Por lo tanto no se eliminara de la pila el árbol hasta que no se hayan recorrido ambos hijos.

El principal error que nos hemos encontrado a la hora de realizar la solución de estos recorrido es cuando un árbol tiene ambos hijos iguales, cuando esto ocurría el programa entra en un bucle infinito. Esto ocurría porque nuestra solución miraba si venia del hijo izquierdo comprobando si el el ultimo árbol que se había eliminado de la pila era igual al hijo izquierdo de la raíz que estamos analizando si esto ocurre se recorría el hijo derecho si lo tiene. El problema aparecía cuando se terminaba de recorrer este y se vuelve a su raíz, como ambos hijos son iguales, el ultimo pop va ser igual al hijo izquierdo por lo que volvía a recorrer el árbol derecho y así de forma infinita. Para solventar esto en el caso de los recorridos *preorde*r y *inorder*, eliminamos de la pila la raíz cuando se ha recorrido su hijo izquierdo y a continuación recorremos su hijo derecho. Para el caso del recorrido *postorder* nos aseguramos que se viene del hijo derecho mediante una booleana.

Tarea 3: Reconstrucción del árbol binario

Esta tarea consiste en la creación de un árbol binario a partir de sus recorridos *postorder* y *inorder.* Para esto nos valemos de una función recursiva la cual inicialmente busca la raíz del árbol en el *postorder* la cual se encuentra en la última posición de la lista. A continuación, calculamos los limites de los rangos donde se encuentran el hijo izquierdo y derecho de la raíz en ambos recorridos. Estos limites nos permitirán generar las sublist donde estarán contenidos los el hijo izquierdo y derecho en postorder y inorder. Una vez generadas estas listas se volverá a llamar a la función dos veces, la pirmera pasándole las sublists postorden y inorder del hijo izquierdo y la segunda pasándole la sublist del hijo derecho, la función repite esta acción hasta que recibe una lista que solo contiene un elemento. De esta forma podremos reconstruir el árbol.