Universitat Politecnica de Lleida

Practica 5

Binarytree

Author:
Martin, Francisco Manuel
Martinez, Victor

DNI:
48057095k
84848893x

December 19, 2019

Índex

1 Introducción

2 Tarea 1: Implementación de los arboles binarios

2.1 Constructores

Para generar la rama *left* y la rama *right* que generan un nuevo árbol apartir del nodo *root* creamos el constructor como se recomienda en la documentacion de la practica de la sigiente forma.

```
this.root = root;
```

2.2 *equals*

2.2.1 *equals* para Arboles binarios

Para comemzar comprovamos que el *object* sea instancia de *LinkedBinaryTree* si no lo es devuelve **false** si no hacemos un cast de object a un *LinkedBinaryTree*. Siguiendo con el orden de ejecucion llamamos a la funcion recursiva que apartir de la root de los dos arboles comprovara si son el mismo.

2.2.2 *equals* para Nodos

De nuevo como en la otra implementacion de equals hacemos *instanceof* esta vez de *Node* y cast para llamar a la misma funcion recursiva que llamamos para *LinkedBinaryTree*.

2.2.3 Funcion recEquals

Esta funcion auxiliar, que llamamos tanto en el equals para arboles o para nodos, de forma recursiva comprueva si dos nodos son los mismos apartir del root comprovando sus ramas derecha e izquierda y comparandolas en el momento que alguno de sus hijos no sea igual evaluara a falso.

3 Tarea 2: Recorridos iterativos en árboles binarios

3.1 Interfaz Traversals

¿Qué diferencias provocaría que la interfaz fuera genérica y los métodos no?
En el caso que tenemos los metodos son genericos i por ello aunque la *interface* no es gernerica podemos trabajar con todos los elementos que contiene *Binarytree*. En el caso contrario no declaras un parametro de tipo que esto conllevara a que estaras sujeto al tipo donde instanciamos la clase *traversals*.

3.2 Clase Iterative Traversals

En esta implementacion como indica en la documentacion de la practica, usaremmos la forma imperativa del recorrido haciendo uso de una pila tal como lo haria la maquina virtual de Java.

3.2.1 Preorden

Este recorrido recuerda al algorismo *DFS* donde hace un busqueda en profundidad, al alcanzar la hoja mas a la izquierda busca en la pila la hoja derecha si no la encuentra busca el nodo superior a este y busca su nodo derecho y asi hasta eliminar de la pila todos los nodos.

3.2.2 Inorder

Este recorrido consiste en localizar la hoja mas a la izquierda y comenzar la lista desde alli despues de localizar la hoja visitamos el nodo superior y el elemento derecho no lo incluiremos en la lista hasta llegar al la hoja izquierda de ese nodo.

3.2.3 Postorder

En este usa la pila para cargar los nodos hermanos que visitaremos antes de visitar al padre, por eso en la lista almazenamos la ultima hoja y cargamos en la pila las ramas de su *root* y al asi almazenamos en la pila las funciones para recorrer hermanos antes que al padre.

4 Tarea 3: Reconstrucción del árbol binario

Para esta implementacion los dos miembros del grupo hicimos dos variantes una usando punteros y otra con sublistas. En el caso de los punteros nos resulto muy dificil realizarlo, teniamos que comprovar muchos casos donde los punteros se salian del index, al final optamos por la implementacion mas simple que requeria usar sublistas estas eliminaban el problema con los indices.

Esta implementacion se sustenta apartir del razonamiento en el cual tomamos como primer elemeto el root del arbol del ultimo elemto de *postorden* y buscamos este mismo elemento en in orden, los elemetos por la izquierda formaran parte del arboles izquierdo y el derecho de los elementos de la derecha, el index donde se encuentra este elemeto los usaremos como herramienta de corte para las sublistas ya que marca el inicio de un arbol y su final. Hacemos este algorismo de forma recursiva i asi alcanzamos a formar el arbol apartir de las dos listas.