ESTRUCTURA DE DADES: PRÁCTICA 3

Sergio Barril Pizarro — 16653350
Rodrigo Cabeza Quirós — 20100426
Grupo: D00
Adán Beltran Gómez

BinarySearchTree y BinaryBalancedTree:

- Tiempo generación estructura:

Fichero	Tiempo	Estructura
movie_rating_small.txt	0.0004793	BST
	0.0004383	AVL
movie_rating.txt	0.0147271	BST
	0.0139691	AVL

Tienen un tiempo similar. El AVL es ligeramente más rápido, ya que en el peor de los casos es O(log n), mientras que BST es O(n) en el peor de los casos.

- Tiempo acceso estructura (cercaFitxers):

Fichero	Tiempo	Estructura
movie_rating_small.txt	0.0011812	BST
	0.0011785	AVL
movie_rating.txt	0.0028823	BST
	0.0025183	AVL

Tienen un tiempo similar, siendo el del AVL ligeramente inferior. Al estar balanceado, todas las ramas son de altura de O(log n), en vez de ser O(n) como puede pasar con los BST.

La diferencia principal entre el BST y el AVL a nivel de implementación es en la inserción. El AVL, después de insertar la entrada, evalúa si el árbol

está balanceado. En caso contrario, lo reestructura para que lo esté, mediante las funciones de rotación del TAD.

- Coste computacional teórico:

NodeTree

Función	Coste	Justificación
Constructor	O(1)	Inicializa atributos
Constructor copia	O(1)	Inicializa los atributos
Destructor	O(1)	
getRight()	O(1)	Retorna atributo
getLeft()	O(1)	Retorna atributo
getParent()	O(1)	Retorna atributo
hasRight()	O(1)	Comprueba un puntero
hasLeft()	O(1)	Comprueba un puntero
isRoot()	O(1)	Comprueba un puntero
isExternal()	O(1)	Comprueba dos punteros
getValue()	O(1)	Retorna atributo
getKey()	O(1)	Retorna atributo
getHeight()	O(1)	Retorna atributo
getBalance()	O(1)	Retorna atributo
setBalance(balance)	O(1)	Actualiza el atributo
setValue(value)	O(1)	Actualiza el atributo
setRight(node)	O(1)	Actualiza el atributo

BinarySearchTree

Función	Coste	Justificación
Constructor	O(1)	Inicializa un puntero a null
Constructor copia	O(n)	Recorremos el árbol
Destructor	O(n)	Recorremos el árbol
size()	O(n)	Recorremos el árbol

isEmpty()	O(1)	Comprueba un puntero
root()	O(1)	Retorna un puntero
search(int key)	O(n)	Recorre solo una rama, pero worst case la rama es demasiado larga
printInorder()	O(n)	Recorremos el árbol
printPreOrder()	O(n)	Recorremos el árbol
printPostOrder()	O(n)	Recorremos el árbol
getHeight()	O(1)	Consulta atributo de Nodo
insert(value, key)	O(n)	Recorre una rama, pero worst case la rama es demasiado larga.
insert(Node, value, key)	O(n)	
mirror()	O(n)	Recorremos el árbol
postDelete(Node)	O(n)	Recorremos el árbol
preCopy(Node)	O(n)	Recorremos el árbol
size(Node)	O(n)	Recorremos el árbol
printPreorder(Node)	O(n)	Recorremos el árbol
printPostorder(Node)	O(n)	Recorremos el árbol
printInorder(Node)	O(n)	Recorremos el árbol
getHeight(Node)	O(1)	Comparaciones simples
mirror(Node)	O(n)	Recorremos el árbol

BalancedBST:

Función	Coste	Justificación
Constructor	O(1)	Inicializa un puntero a null
Constructor copia	O(n)	Recorremos el árbol
Destructor	O(n)	Recorremos el árbol
size()	O(n)	Recorremos el árbol
isEmpty()	O(1)	Comprueba un puntero
root()	O(1)	Retorna un puntero
search(int key)	O(log n)	Recorre solo una rama
printInorder()	O(n)	Recorremos el árbol
printPreOrder()	O(n)	Recorremos el árbol

printPostOrder()	O(n)	Recorremos el árbol
getHeight()	O(1)	Consulta atributo de Nodo
insert(value, key)	O(log n)	Recorre una rama
insert(Node, value, key)	O(log n)	
mirror()	O(n)	Recorremos el árbol
postDelete(Node)	O(n)	Recorremos el árbol
preCopy(Node)	O(n)	Recorremos el árbol
size(Node)	O(n)	Recorremos el árbol
printPreorder(Node)	O(n)	Recorremos el árbol
printPostorder(Node)	O(n)	Recorremos el árbol
printInorder(Node)	O(n)	Recorremos el árbol
getHeight(Node)	O(1)	Comparaciones simples
mirror(Node)	O(n)	Recorremos el árbol
rotateExtern()	O(1)	Reasignación de un número constante de punteros
rotateIntern()	O(1)	Reasignación de un número constante de punteros