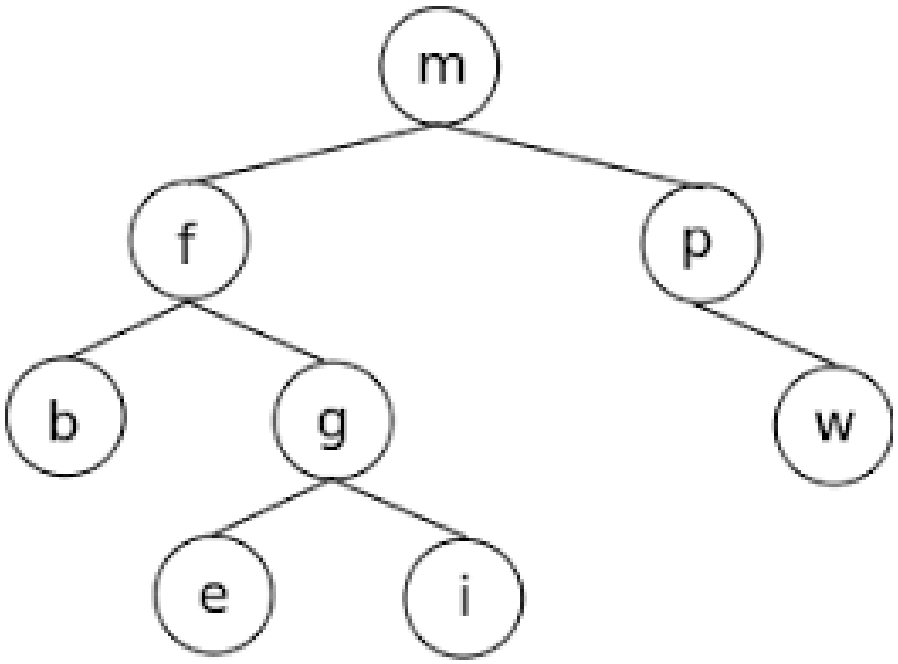
PRÀCTICA 3. ARBRES BINARIS



Pau Bernabé i Marc Cerrillo

Estructura de Dades – Universitat de Barcelona

13/05/2018

ÍNDEX

INTRODUCCIÓ

Els objectius d'aquesta pràctica és familiaritzar-se amb les estructures de dades no lineals. En específic sobre els arbres binaris, analitzant les seves característiques i les seves diferències.

EXERCICI 1

En aquest primer exercici ens demanen implementar un arbre binari de cerca amb una representació encadenada.

Comencem creant la classe *BinarySearchTree* amb templates i implementem els diferents mètodes que ens demanen (size, height, empty, root, insert, search, print…).

Un cop creat aquest mètode implementem el TAD *NodeTree,* també amb templates. Aquesta classe és la que ens permetrà crear l'arbre binari.

Com sabem, el cost computacional d'aquestes operacions és igual a l'altura de l'arbre O(h) que és equivalent a $O(\log_2 n)$.

Per finalitzar implementem l'operació mirall sobre l'arbre dins la classe *BinarySearchTree.* De tal manera que el fill dret passi a ser el fill esquerra i l'esquerra passi a ser el fill dret.

EXERCICI 2

En aquest exercici ens demanen implementar un cercador de pel·lícules utilitzant un arbre de cerca binària.

Per a fer-ho haurem de crear una classe TAD *Movie* que reculli totes les dades de les pel·lícules, és a dir, el seu ID, el títol i la seva puntuació.

Després d'això creem una classe anomenada *BSTMovieFinder* que és la que farà de pont entre el main i el BST i implementem les funcions que ens demanen.

Per acabar de fer el programa reutilitzem part del codi de l'exercici 1 canviant certes coses. Per exemple a la classe *BinarySearchTree* haurem de canviar el mètode setElement() per dos mètodes diferents. El setKey() que ens serveix per ordenar els elements dins de l'arbre (l'id de la movie) i el setValue() on guardem l'objecte movie.

EXERCICI 3

En aquest exercici ens demana que mantinguem la funcionalitat de l'exercici 1, però canviant el *BinarySearchTree* per un TAD *BalancedBST*.

Un cop tenim ja tenim implementada la classe, implementem les operacions que ens demanen. Una funció que retorni el títol de la pel·lícula més llarga, el valor de puntuació més alt i més baix i mostrar aquestes pel·lícules.

Les diferències més significatives entre el BalancedBST i el BinarySearchTree és la implementació dels mètodes de rotació per a poder balancejar l'arbre. El cost computacional d'aquestes classes l'explicarem a l'exercici 5.

EXERCICI 4

En aquest exercici hem d'implementar un cercador de pel·lícules tal i com hem fet a l'exercici 2 però en comptes d'un BST amb un BalancedBST.

Per fer aquest exercici hem reaprofitat part del codi escrit i explicat en els exercicis anteriors.

EXERCICI 5

En aquest exercici ens demana que calculem el rendiment de les dues implementacions (Exercici 2 i 4)

Per accedir al fitxer petit amb un BST tarda 0.00064s i al fitxer gran 0.196037s.

Per accedir al fitxer petit amb un AVL tarda 0.00386s i al fitxer gran 1.20913s.

Els resultats que ens donen són els que esperàvem ja que com l'arbre AVL ha de balancejar els nodes el temps d'execució d'aquest serà més gran que el del arbre BST. Tot i que en un primer moment podem pensar que els arbres AVL tenen un cost computacional més elevat, a la llarga a l'hora de treballar amb aquests arbres serà molt més eficient que no pas amb els BST.

El cost computacional d'un arbre BST desbalancejat, en el pitjor dels casos costa O(n). Si està perfectament balancejat en canvi, costa $O(\log_2 n)$. En un arbre AVL el cost de cerca, inserir o eliminar serà sempre de O(log n).