Estructures de dades
Curs 16-17
Pràctica II

Biel Moyà

2 d'abril de 2017

Data d'entrega: 09 / 04 / 2017; 23:55 h

Aquesta pràctica consta de dos problemes. El primer s'ha d'entregar de manera obligatòria i l'altre serà optatiu. Entregant només el primer problema podreu optar a tenir una nota de 8, això vol dir que el segón problema val 2 punts.

L'entrega constarà d'una carpeta comprimida que contindrà el codi font en carpetes separades i un fitxer anomenat **resultats.txt** que serà modificat per ambdos programes. **No entregueu el projecte de gps ja que dificulta la correcció.**

L'exercici, es pot fer de manera individual o en parelles, a la carpeta comprimida heu d'afegir un document anomenat *participants.txt* on indicau els noms i DNIS de les persones que han realitzat la pràctica. Es suficient amb que un dels dos components del grup realitzi la entrega.

1 És el meu arbre, un arbre ACB?

Els Arbres de Cerca Binaris (ACB) com bé sabeu, són arbres binaris que segueixen critèris d'ordre específic. Tot fill esquerra d'un node ha de tenir la clau amb un valor menor al seu valor i tot fill dret d'un node ha de tenir la clau amb un valor major al seu valor.

Donat un fitxer de texte la primera línia del qual conté un conjunt de lletres majúsucules separades per espais el significat del qual és el recorregut **inordre** d'un arbre i la segona línia conté un conjunt de lletres majúsucules separades per espais el significat del qual és el recorregut **preordre** d'un arbre.

Es demana

1. Realitzar un programa que a partir de la combinació de les dues línies del fitxer construeixi l'arbre corresponent. El programa ha de contenir almanco un programa principal i un *package* anomenat **darbre_binari**. Aquest cop l'estructura arbre serà construïda amb memòria dinàmica.

2. Per tal de validar que la construcció de l'arbre és correcte ho fareu mitjançant un recorregut inordre de l'arbre i comparant la seqüència resultant amb la primera línia del fitxer. El *package* **darbre_binari** ha de implementar aquesta funció anomenada:

   arbre_correcte(a: *in* **arbre**; r: *in* **recorregut**) **return** *boolean*.

3. El resultat d'aquest procés serà l'escriptura d'un 1 o un 0 a un fitxer anomenat **resultats.txt**. 1 si el resultat és correcte i 0 si no ho és.

Suposarem que no hi ha claus repetides.

Com nosaltres som persones amb una gran curiositat el repte no acaba aquí. El que en realitat ens interessa és saber si l'arbre que hem construït és realment un arbre ACB per això:

Es demana

1. Construir una funció al paquet **darbre_binari** anomenada:

   es_ACB(a: *in* **arbre**) **return** *boolean*.

2. El resultat d'aquest procés serà l'escriptura d'un 1 o un 0 a un fitxer anomenat **resultats.txt**. 1 si l'arbre és ACB i 0 si no ho és. **Heu de concatenar aquest valor al resultat ja escrit sense deixar cap espai**.

El nom del fitxer serà introduit mitjançant la línia de comandes.

Ajuda per la construcció de l'arbre

En un recorregut en preorde, l'element més a l'esquerra és l'arrel de l'arbre. Mitjançant la cerca del primer element del preorde en la seqüència inordre, descobrireu que tots els elements a l'esquerra de la arrel estan en el subarbre esquerre i tots els que queden a la part dreta estan en subarbre dret. A partir d'aquest coneixement l'algorisme general és:

1. Seleccionar un element de la seqüència preodre. Hem de tenir una variable index que ens indiqui en quin element ens trobam.

2. Crear un nou node amb les dades de l'element escollit.

3. Cercar aquest element en el recorregut inordre.

4. Fer una crida recursiva per a construir l'arbre esquerra de l'element escollit.

5. Fer una crida recursiva per a construir l'arbre dret de l'element escollit.

6. Realitzar les assignacions corresponents dels nous subarbres al node creat.

7. Retornar el node.

Ajuda per la comprovació ACB

Pensau que no basta que els fills esquerra i dreta d'un node compleixin la regla d'ACB, també ho han de complir els nets, els besnets ...

- **Mètode 1:** *(Correcte, però no eficient)* Per cada node un dels nodes de l'arbre, comprovar si el valor màxim en el subarbre esquerre és més petit que el valor del node i que el valor mínim en el subarbre dret més gran que el valor del node.

- **Mètode 2:** *(Correcte, i eficient)* La resolució d'aquest problema es pot fer visitant cada node 1 sol cop. Cada node que visitau ha de saber quin és el valor mínim i màxim que pot tenir.

Nota: Al vostre projecte podeu afegir un README per especificar particularitats de la pràctica (símbols especials o consideracions que NO contradiguin l'enunciat)

2 Has_path_sum versió dinàmica

En aquest segon exercici es demana que realitzeu l'exercici en el que és demanava que implementessiu la funció *has_path_sum* (veure enunciat de l'exercici a campus extens) amb els següents canvis:

1. El contingut de l'arbre seràn nombres enters, aquests es trobaràn a un fitxer que s'especificarà per línia de comandes.

2. El valor pel qual volem ordenar serà el segón paràmetre de la línia de comandes.

3. L'estructura de l'arbre no serà amb un cursor sino que ho fareu amb memòria dinàmica (2 punters i el valor de la clau).

4. El resultat de la funció s'escriurà en el mateix fitxer de resultats que en l'exercici anterior. 1 si *has_path_sum* és vertader i 0 si el resultat de *has_path_sum* es fals.

Si vàreu fer l'exercici voluntari, el codi de la funció *has_path_sum* no ha de canviar massa, la idea és la mateixa.

Haureu de canviar la funció *put*. Per realitzar aquesta funció, podeu usar estructures de dades de suport tipus llistes o piles.