Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2012

Práctico 2 – Parte 2

Docentes: Silvia Pelozo, Felipe Espósito, Emmanuel Gunther

1. Sea *bintree* una implementación del TAD *arbol binario*. Escribí las siguientes funciones:

   a) **fun** *nodes*($t : bintree$) **ret** $n : Nat$. que devuelve la cantidad de nodos del árbol t.

   b) **fun** *height*($t : bintree$) **ret** $h : Nat$, que devuelve la *altura* del árbol t.

   c) **fun** *ancestor*($t : bintree, e_1 : elem, e_2 : elem$) **ret** $res : Bool$, que devuelve *true* si y solo si e_1 es *ancestro* de e_2 en el árbol t.

   d) **fun** *descendant*($t : bintree, e_1 : elem, e_2 : elem$) **ret** $res : Bool$, que devuelve *true* si y solo si e_1 es *descendiente* de e_2 en el árbol t.

   Ayuda: Las soluciones recursivas son más sencillas que las iterativas.

2. Implementá el TAD *arbol binario* utilizando la siguiente estructura de datos:

   type *tnode* = **tuple**
   left : **pointer to** *tnode*
   value : *elem*
   right : **pointer to** *ntode*

   type *binTree* = **pointer to** *tnode*

   con la que se representa un *árbol binario* como un puntero a un nodo, que a su vez posee punteros a los subárboles izquierdo y derecho (*left* y *right* respectivamente).

3. Investigá las diferentes maneras de iterar sobre un árbol binario: *in-order*, *pre-order*, *post-order*. Escribí un procedimiento para cada una de estas formas, suponiendo que se quiere modificar el valor de cada nodo del árbol con la función **fun** *process*($e : elem$) **ret** $f : elem$.

4. Con las valores $\{1, 4, 5, 10, 16, 17, 21\}$ construí *árboles binarios de búsqueda* de alturas 2, 3, 4, 5 y 6.

5. Dado un ABB con cuyos nodos poseen valores entre 1 y 1000, se quiere encontrar el número 363. ¿Cuáles de las siguientes secuencias no puede ser una secuencia de nodos examinados por el algoritmo de búsqueda?

   a) $2, 252, 401, 398, 330, 344, 397, 363$.

   b) $924, 220, 911, 244, 898, 258, 362, 363$.

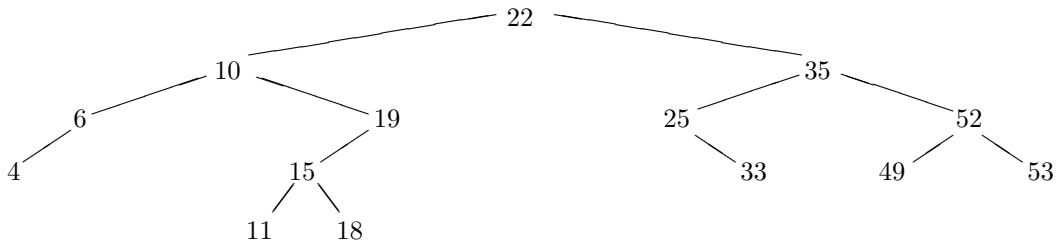
   c) $925, 202, 911, 240, 912, 245, 363$.

   d) $2, 399, 387, 219, 266, 382, 381, 278, 363$.

   e) $935, 278, 347, 621, 299, 392, 358, 363$.

6. Dada la secuencia de números $23, 35, 49, 51, 41, 25, 50, 43, 55, 15, 47$ y 37, determiná el ABB resultante de realizar las inserciones de dichos números exactamente en ese orden a partir del ABB vacío.

7. Determiná una secuencia de inserciones que dé lugar al siguiente ABB:



8. Reimplementá la función *search* de un ABB (vista en teórico) sin utilizar recursión.

9. Un *multiconjunto* es una colección desordenada de valores en la que puede haber duplicados, que se puede especificar como sigue:

TAD $multiconjunto_A$
constructores
$\emptyset : multiconjunto_A$
$ins : A \times multiconjunto_A \to multiconjunto_A$
operaciones
$mult : A \times multiconjunto_A \to Nat$
$elim : A \times multiconjunto_A \to multiconjunto_A$
ecuaciones

$$\begin{aligned} mult(e, \emptyset) &= 0 \\ mult(e, ins(e', M)) &= (e = e' \to 1 + mult(e, M) \\ &\quad \square \neg e = e' \to mult(e, M) \\ &\quad) \end{aligned}$$

$$\begin{aligned} elim(e, \emptyset) &= \emptyset \\ elim(e, ins(e', M)) &= (e = e' \to elim(e, M) \\ &\quad \square \neg e = e' \to add(e', elim(e, M)) \\ &\quad) \end{aligned}$$

   a) Extendé la especificación agregando tres operaciones: una para observar si un multiconjunto es vacío; otra para unir dos multiconjuntos; y otra que dados dos multiconjuntos elimina del primero la cantidad de ocurrencias en el segundo de cierto valor v (esto es, la resta de multiconjuntos).

   b) Implementá este TAD utilizando como estructura de datos un árbol binario de búsqueda (abstracto).

   Ayuda: Tené en cuenta que con el invariante de ordenación de un árbol binario de búsqueda no es posible tener dos nodos con un mismo valor.

10. Reimplementá el TAD *multiconjunto* del ejercicio 9 utilizando como estructura de datos un *árbol binario de búsqueda* implementado con punteros.

11. Un *heap binario*, además de la propiedad de ordenación, debe cumplir la propiedad de *forma*: el árbol debe ser (casi) completo, esto es, los nodos de todos los niveles, excepto posiblemente el último, tienen tanto hijo izquierdo como derecho. En el último nivel, si un nodo no tiene hijos, entonces todos sus *hermanos* a la derecha no pueden tener hijos.

    Teniendo en cuenta estas dos propiedades características de un *heap*:

    a) ¿Cuál es el número máximo y mínimo de nodos que puede tener un *heap binario* de altura h?

    b) ¿Dónde está ubicado el elemento mínimo de un *heap*? ¿Y el máximo?

    c) ¿Un arreglo ordenado de forma descendente implementa un *heap*? ¿Un *heap* implementado con un arreglo, siempre da lugar a un arreglo ordenado de manera descendente?

    d) El arreglo $[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$ ¿es un *heap*?

12. Cuando implementamos un *heap binario* con un arreglo a, el procedimiento $sink(a, i)$ "hunde" el elemento $a[i]$ a través de sus hijos $a[2 * i]$ y $a[2 * i + 1]$ de manera de que la estructura resultante mantenga la propiedad de ordenación del *heap*. Representá gráficamente la evolución, paso a paso, del *heap* al ejecutar $sink(a, 3)$, cuando $a = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0]$.

13. ¿Qué sucede si llamamos a $sink(a, i)$ cuando $a[i]$ es mas grande que sus hijos? ¿Y cuando $i > N/2$, donde N es el tamaño de a?

14. Considerá una *cola de prioridades* Q representada con el *heap binario* $[15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]$. Graficá el *heap* paso a paso cuando:

    a) se ejecuta *dequeue(Q)*,

    b) se ejecuta *enqueue(Q,10)*.

15. Un *heap n-ario* es un árbol cuyos nodos tienen n hijos que mantiene las propiedad de ordenación y forma del *heap binario*.

    a) ¿Cómo se representa un *heap n-ario* con un arreglo?

    b) ¿Cuál es la altura de un *heap n-ario* con m nodos?

    c) Implementá los procedimientos *sink* y *float* (y posiblemente sus procedimientos auxiliares) para esta clase de *heaps*.

16. Un *árbol AVL* es un ABB auto balanceado. En un AVL las alturas de los subárboles de cierto nodo difieren a lo sumo en uno. Manteniendo este invariante (extra) se obtiene buena eficiencia en las operaciones asociadas, a costa de que la inserción y el borrado de nodos puede requerir rebalancear el árbol, aplicando una o mas operaciones de *rotación*.

    a) Investigá como funcionan los AVL.

    b) Implementa las operaciones de *search*, *insert* y *delete* sobre un AVL implementado con punteros.

    Ayuda: Puede resultar útil pensar primero cómo resolver el problema sobre arboles abstractos. Además una buena modularización de estas operaciones *abstractas* y otras auxiliares como las de rotación puede resultar de mucha ayuda.