EjercicioMontículo2

Estructuras de Datos

Tema 4: árboles

1º Grado en Ingeniería de la Computación
© Profesor Dr. Carlos Grima Izquierdo (www.carlosgrima.com)
URJC (www.urjc.es)

Vamos a ampliar la resolución del EjercicioMontículo1 con los métodos de "borrar" y "getRaiz". Para ello, ten en cuenta lo siguiente:

- Primero deberás programar el método privado "reestructurar", que reestructurará el árbol completo, suponiendo como precondición que todo el árbol es un montículo excepto la raíz (para ello usa el método, ya programado en EjercicioMonticulo1, que comprueba si un subárbol es un montículo). ¿Cómo comprobar que todo el árbol es un montículo excepto la raíz? Pues comprobando que todos los subárboles que cuelgan directamente de la raíz son montículos.
- Recuerda que los elementos del montículo se numeran desde el 1 (la raíz) hasta el n (la última hoja). El elemento 1 estará en la posición 0 de la lista contigua, y así sucesivamente hasta el elemento n, que estará en la posición n-1.
- El método "borrar" borra el nodo raíz. Pone el último elemento en su lugar, y entonces reestructura.
- El método "getRaiz" devuelve una copia del elemento que está en la raíz del montículo, sin borrarlo.
- Programa un main en el cual, al estilo del main de EjercicioMonticulo1, se van insertando nodos hasta que tienes el árbol inicial del ejemplo de eliminar que tienes en las diapositivas. Por cada inserción, imprime el árbol. Entonces llama al método borrar y vuelve a imprimir el árbol. Te tiene que quedar el árbol final que aparece en el ejemplo de eliminar de las diapositivas. Finalmente vuelve a borrar la raíz (esta vez hay un 3) y vuelve a imprimir el árbol de nuevo y comprueba si el resultado es correcto.

Ahora programa un método "ordenarPorMonticulo()", que no pertenezca a ninguna clase (estará en una biblioteca llamada "ordenación", por lo tanto su declaración habrá que ponerlo en el correspondiente archivo ordenacion.h y su definición en el correspondiente archivo ordenacion.cpp), con las siguientes características:

- Recibe como parámetro un puntero a un objeto de tipo ListaContigua de ints (reutiliza la actividad EjercicioListaContigua2), con los elementos desordenados.
- Construirá un montículo local a partir de inserciones sucesivas de todos los elementos de la lista pasada como parámetro. Aquí es donde gastamos memoria temporal extra.
- Ahora irá obteniendo sucesivamente la raíz del montículo (el mínimo elemento) y lo irá poniendo en la posición 0, 1, 2… n-1 del objeto de tipo ListaContigua que nos pasaron como parámetro.
- De este modo, la lista pasada como parámetro quedará ordenada de menor a mayor.

Ahora amplía el main para que, a continuación de lo programado antes en dicho main, genere un objeto de tipo ListaContigua que contenga 20 ints aleatorios (números entre 50 y 100, ambos inclusive), y lo muestre por pantalla (reutilizando el método para imprimir desarrollado en

EjercicioListaContigua2). A continuación lo ordenará utilizando el algoritmo de ordenación “ordenarPorMonticulo”, mostrando el resultado por pantalla.