Programación 2 - Guía
Corrección de Código - Laboratorio 3

Procedimiento "kesimo_elemento"

30 de mayo de 2017

Se pide implementar un algoritmo que devuelva el $k-$ésimo elemento de un árbol binario.

```
/*
  Devuelve el elemento que, según la propiedad de orden de los árboles
  'binario', está en el k-ésimo lugar de 'b'.
  No se deben crear estructuras auxiliares.
  No se deben visitar nuevos nodos después que se encontró el kesimo.
  Precondición: 1 <= k <= cantidad_binario(b).
 */
info_t kesimo_elemento(nat k, binario b);
```

La estructura arborescente sugiere usar una estrategia recursiva. Si en el subarbol izquierdo hay $k-1$ elementos entonces la raíz de `b` es el elemento buscado. En otro caso se busca el elemento con una llamada recursiva en el subarbol izquierdo o en el subárbol derecho dependiendo de la cantidad de elementos que haya en el subárbol izquierdo.

```
kesimo_elemento(nat k, binario b)
  cant_izq = cantidad_binario(izquierdo(b))
  IF cant_izq == k - 1
    res = raiz_binario(b)
  ELSE
    IF cant_izq >= k
      res = kesimo_elemento(k, izquierdo(b))
    ELSE //  cant_izq + 1 < k
      res = kesimo_elemento(k - cant_izq - 1, derecho(b))

  RETURN res;
```

Las llamadas recursivas se hacen en árboles estrictamente más chicos que `b` por lo que la ejecución eventualmente debe terminar.

Debido a que se hacen llamadas a las funciones `izquierdo`, `derecho` y `raiz_binario` que tienen como precondición que el árbol no sea vacio se debe verificar que esto último se cumple. Las desigualdades de la precondición garantizan que en la llamada inicial el árbol tiene al menos un elemento, por lo que no es vacío. Entonces, para demostrar que nunca se llama la función pasándole como parámetro un árbol vacío basta demostrar que la precondición se sigue cumpliendo en las llamadas recursivas. Esto es evidente en la llamada al subárbol izquierdo, ya que el parámetro `k` no se modifica, y por lo tanto cumple $1 \leq k$ y por la condición de la línea anterior también se cumple $k \leq$ cantidad_binario(izquierdo(b)). La precondición también se cumple en la llamada al subárbol derecho ya que: (I) de las condiciones de las líneas anteriores se sabe que $cant\_izq + 1 < k$, de donde $1 \leq k - cant\_izq - 1$, (II) de $k \leq$ cantidad_binario(b) y cantidad_binario(b) = cantidad_binario(izquierdo(b)) + 1 + cantidad_binario(derecho(b)) se deduce que $k -$ cant_izq $- 1 \leq$ cantidad_binario(derecho(b)).

Pero aunque esta implementación es funcionalmente correcta ya que devuelve el elemento buscado, no es aceptable por razones de eficiencia, pedida de manera explicita con el requerimiento no funcional que exige no visitar nodos después de haber encontrado el $k-$ésimo. Esto no se cumple ya que la llamada a `cantidad_binario(izquierdo(b))` recorre todos los nodos del subárbol izquierdo aunque el $k-$ésimo elemento podría no ser el último nodo de ese árbol. Además con la siguiente llamada recursiva se vuelven a visitar todos o algunos de los nodos del subárbol.

Hace falta definir una funcion auxiliar que además de contar la cantidad de elementos en el subárbol izquierdo devuelva el elemento buscado en el caso en que el elemento esté en ese subárbol.

Para una solución alternativa notemos que el $k-$ésimo elemento es el que quedaría en la posición k en una lista en la que se insertan los elementos del árbol en una recorrida en orden.

```
enlistar(cadena & ordenada, binario b)
  IF NOT es_vacio_binario(b)
    enlistar(ordenada, izquierdo(b))
    insertar_siguiente(copia_info(raiz(b)), final_cadena(ordenada), ordenada)
    enlistar(ordenada, derecho(b))

kesimo_elemento(nat k, binario b)
  ordenada = crear_cadena()
  enlistar(ordenada, b)
  cursor = inicio_cadena(ordenada)
  FOR i = 1 TO k - 1
    cursor = siguiente(cursor, cadena)
  res =  info_cadena(cursor, cadena)
  liberar_cadena(cadena)
  RETURN res
```

Un problema con esta solución es que no cumple el requerimiento no funcional de utilización de espacio que exige no crear estructuras auxiliares. Otro problema es que recorre todo el árbol, por lo que visita nodos después de haber encontrado el $k-$ésimo.

Este último problema se puede resolver si se lleva un contador con la cantidad de inserciones en la lista. Al insertar el $k-$ésimo se trunca la recorrida. Y usar un contador también resuelve el primero de los problemas, ya que al llegar al $k-$ésimo se devuelve ese elemento. La forma de mantener el contador no debe ser mediante una varibale global, sino con un parámetro pasado por referencia.

En el listado 1 se puede ver la implementación. Se llama a una funcion auxiliar que realiza una recorrida en orden de `b` truncada al llegar al $k-$ésimo elemento. Como toda recorrida en orden procesa el subárbol izquierdo, luego la raíz y luego el subárbol derecho. Por ser una recorrida truncada en este caso tal vez no se procese el subárbol derecho. El contador es la variable `pos`, inicializada en 0, que se pasa por referencia. Esta variable se incrementa a la vuelta de la recursión de la llamada al subárbol izquierdo. Hasta que se llega al $k-$ésimo nodo el valor de `pos` es la posición que le corresponde al nodo en la recorrida en orden. Luego de eso `pos` se sigue incrementando en los nodos que son ancestros de los primeros k. En una solución alternativa en la línea 12 podría agregarse $pos < k$ como condición para el incremento. Con esta variante el valor final de `pos` seria k.

Las llamadas recursivas se hacen en árboles más pequeños por lo que el programa termina.

Como `pos` es una variable es de entrada y salida debe ser inicializada en `kesimo_elemento`. En cambio `res` no se inicializa porque es una variable de salida.

Listado 1: kesimo_elemento

```
1 /*
2   Si 'cantidad_binario(b) >= k - pos' devuelve en 'res' el elemento que está en
3   la posición 'k - pos' de 'b' y deja 'pos' con un valor mayor o igual a 'k';
4   en otro caso incrementa 'cantidad_binario(b)' el valor de 'pos'.
5  */
```

```
6  static void en_orden_truncada(nat k, binario b, nat &pos, info_t &res) {
7    // procesar subárbol izquierdo
8    binario izq = izquierdo(b);
9    if (!es_vacio_binario(izq))
10     en_orden_truncada(k, izq, pos, res);
11   // procesar raíz
12   pos++;
13   if (pos == k) {
14     res = raiz_binario(b);
15   } else if (pos < k) {
16     // procesar subárbol derecho
17     binario der = derecho(b);
18     if (!es_vacio_binario(der))
19       en_orden_truncada(k, der, pos, res);
20   } else {
21     /*
22       No se hace nada porque
23       pos > k => se encontró el k-esimo en izquierdo(b)
24      */
25   }
26 } // en_orden_truncada
27
28 info_t kesimo_elemento(nat k, binario b) {
29   info_t res;  // parámetro de salida
30   nat pos = 0; // parámetro de entrada/salida
31   en_orden_truncada(k, b, pos, res);
32   return res;
33 }
```

Una variante es tener en lugar de los parámetros `k` y `pos` un sólo parámetro, `faltan`, inicializado en k que indica la cantidad de nodos que falta encontrar. Este parámetro se decrementaría y cuando valga 0 se habría encontrado el $k-$ésimo.

Otra variante es remover el control de árbol no vacío en las líneas 9 y 18 e incluir al inicio la condición de que `b` no sea vacío.

La declaración de la funcion auxiliar debe ser precedida por la palabra `static` para asegurar que su ámbito queda restringido al archivo `binario.cpp`.