Ejercitación Nº 2: Recursión

Árbol binario

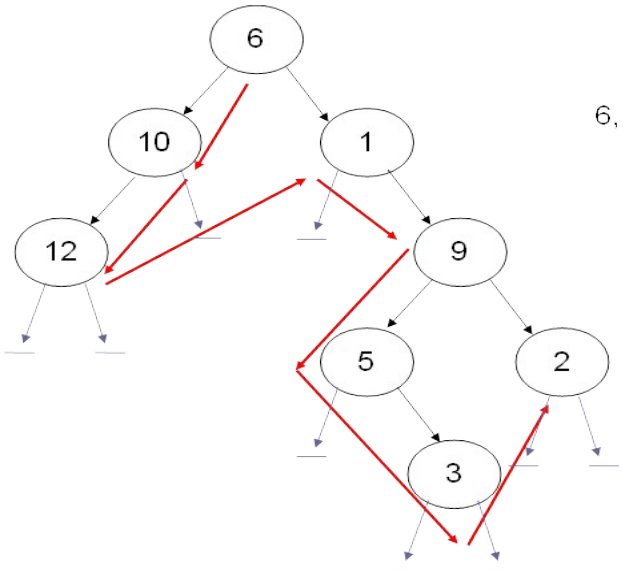
Un árbol binario es una colección de elementos, llamados nodos, uno de los cuales se distingue como raíz; junto con una relación "padre de" que impone una estructura jerárquica sobre los nodos. Cada nodo es padre de, a lo sumo, dos hijos.

A continuación, se declara una estructura posible para implementar un árbol binario:

```
struct nodo_arbol {
    int val;
    nodo_arbol * izq;
    nodo_arbol * der;
};
```

Como se puede ver, esta declaración define un nodo del árbol mediante una estructura compuesta de un valor entero y de dos nodos, para representar los posibles hijos. Notar que el tipo de datos de los nodos `izq` y `der`, es el mismo que se está definiendo (`nodo_arbol`). Esto es posible debido a que se definen punteros a dicho tipo, es decir, se almacenarán únicamente direcciones de memoria.

Construir un árbol por demanda, de acuerdo a un conjunto de entrada, es una tarea compleja que depende del tipo de árbol que se esté construyendo (árbol de búsqueda, árbol balanceado, etc); para esta ejercitación particular se propone el siguiente método para crear un árbol, a partir de datos provistos por el usuario.
La entrada será una cadena de texto que indicará los nodos que contendrá el árbol, y el orden de los mismos. Por ejemplo, el árbol siguiente se construye a partir de la cadena mostrada a continuación:



La cadena para generar el árbol es:
6, 10, 12, -, -, -, 1, -, 9, 5, -, 3, -, -, 2, -, -

"6,10,12,-,-,-,1,-,9,5,-,3,-,-,2"

Como se puede deducir del ejemplo anterior, la cadena de texto se construye siguiendo un recorrido preorden del árbol. Esta cadena de entrada debe cumplir con las siguientes reestricciones:

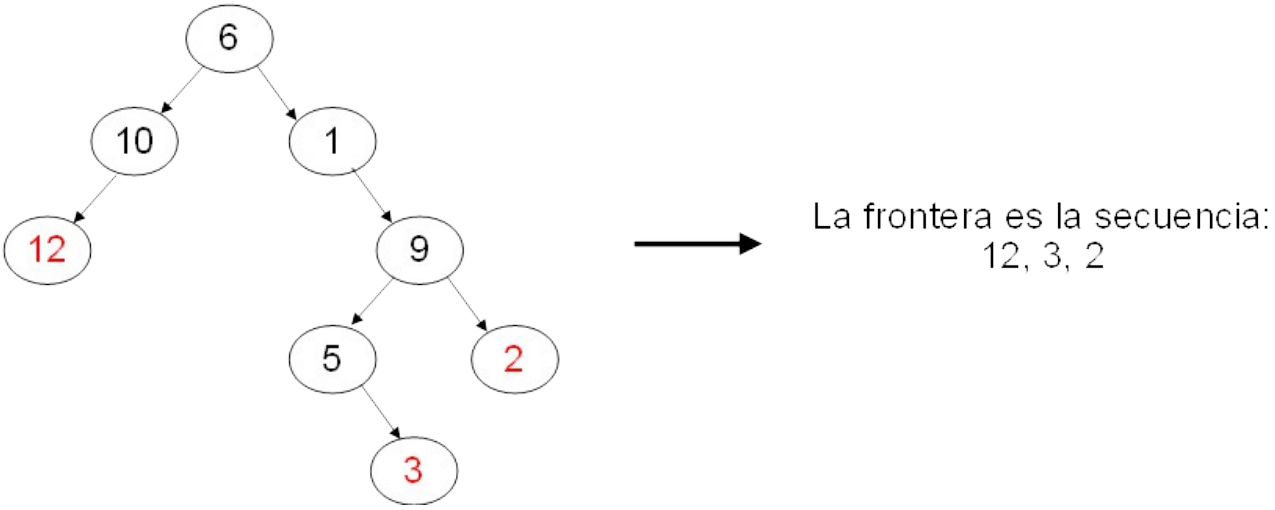
- No debe contener espacios en blanco.
- Se utiliza la coma (",") como separador entre los nodos.
- Para indicar un nodo "vacío" se utiliza el guión medio ("-").
- Se pueden omitir los últimos nodos "vacíos".

El código del árbol binario implementado se puede encontrar en los archivos `arbol.h` y `arbol.cpp`. Asimismo, en los archivos `lista.h` y `lista.cpp` se encontrará la implementación de una estructura de lista, necesaria para el ejercicio que se presentará a continuación.

Ejercicio resuelto

La frontera de un árbol binario es la secuencia formada por los elementos almacenados en las hojas tomados de izquierda a derecha. Escribir una función que dado un árbol binario y una lista vacía, ambos pasados como parámetros, devuelva en dicha lista la frontera del árbol.

Por ejemplo:



De esta forma se propone la siguiente función para resolver el ejercicio:

```
void construir_frontera(nodo_arbol * arb, nodo_lista * & l) {
    if (arb != NULL) {
        if (es_hoja(arb))
            agregar_final(l, arb->elemento);
        else {
            construir_frontera(arb->izq, l);
            construir_frontera(arb->der, l);
        }
    }
}
```

Ejercitación propuesta

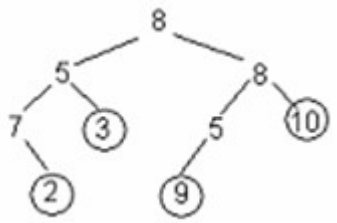
Se supone que el árbol se utilizará en un contexto en el cual se podrá conocer, antes de su construcción, la cantidad de elementos totales que almacenará. Debido a esta restricción se propone utilizar una estructura de arreglo para almacenar los elementos del árbol, sabiendo que:

- La posición inicial del arreglo no se utilizará para almacenar elementos.
- Si un elemento está en el índice `i` (con `i>=1` y `i <= elementos del árbol`), sus hijos (de existir) estarán en las posiciones `2 * i` y `2 * i + 1`, respectivamente.

Requisitos de la entrega

- Reestructura el código encontrado en arbol.h y arbol.cpp para utilizar esta nueva estructura propuesta, modificando las implementaciones de los métodos del árbol.
- Reimplementar el algoritmo que permitía obtener la frontera del árbol, para que resuelva dicho ejercicio utilizando esta nueva estructura.
- Resolver el siguiente ejercicio, considerando únicamente la estructura de arreglo para la implementación del árbol binario.

  Dado un árbol binario de enteros, como el que se presenta a continuación, resuelva:



  - Encontrar la máxima diferencia (en valor absoluto) entre dos hojas adyacentes. Para el árbol del ejemplo, la máxima diferencia es 6 (| 3 – 9|).

<u>Consideraciones adicionales:</u>

- Recordar que el arreglo, una vez creado, debe ser inicializado en algún valor por defecto que luego permita reconocer si existe o no el nodo. Se recomienda utilizar la constante INT_MAX provista por el lenguaje, como parte del encabezado <climits>.
- Las funciones creadas para mostrar el árbol por pantalla pueden ser redefinidas de la forma que consideren necesaria, si con esto se simplifica la visualización de los elementos del árbol.
- Probar cada función con distintas configuraciones del árbol, creadas a partir de las cadenas que se presentaron al comienzo del enunciado.

Entrega

El día **miércoles 27 de abril**, a las **14hs** para la comisión I y a las **16hs** para la comisión II, se deberá traer la ejercitación propuesta para su corrección y evaluación individual dentro del espacio del laboratorio.