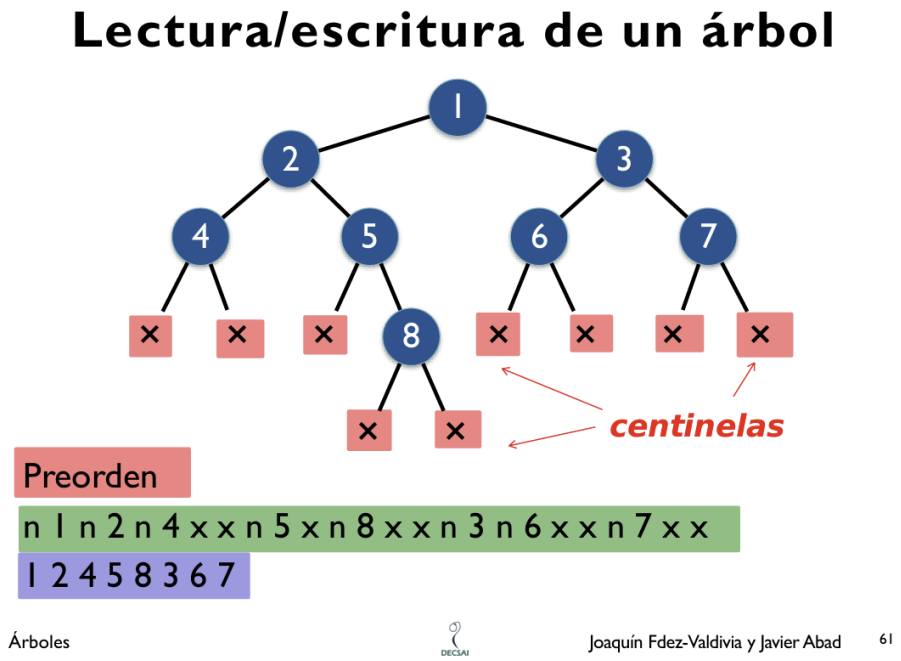
Reto 3

Implementado por Darío Gómez Cisneros y Rubén Callejas Fernández

Hemos preparado un diseño que se encarga de leer y escribir un árbol binario en disco mediante el uso de centinelas (símbolo '#' en nuestro caso). Para llevarlo a cabo, nos hemos basado en la solución que se da en las transparencias del tema de teoría para leer y escribir con centinelas en preorden. Se explica aquí:



En resumen, se va generando el árbol en preorden con las etiquetas dadas, si la etiqueta tiene otra detrás AB → B es hijo izquierdo de A, si tiene un centinela A# → NO tiene hijo izquierdo, y puede tener uno derecho de esta manera A#B → B hijo derecho de A, en caso de no tener hijos se indica con A## → A no tiene hijos. Basándose en esta premisa, se pide minimizar el número de datos de entrada necesarios para formar dicho árbol, con lo que hemos hecho una pequeña modificación que disminuye MÍNIMAMENTE los datos de entrada: si un nodo no tiene hijos, se indicará con otro centinela distinto, '.' en nuestro caso. Con esto los nodos sin hijos necesitarán un carácter menos para ser representados, dependerá del árbol en concreto si se optimiza o no pero tendría potencial de mejorar la solución.

Un ejemplo de árbol con el método propuesto en clase sería: AB##C#D## mientras que con el cambio sugerido por nosotros quedaría: AB.C#D.

Para la construcción del árbol en base a esa cadena de caracteres, el procedimiento seguido es: se va recorriendo uno por uno los caracteres de la cadena y se interpreta qué hacer en base a el mismo, si es cualquiera distinto de # y . se procesa como un nodo normal, asignándole los valores correspondientes y procediendo a evaluar sus dos siguientes hijos; si es uno de ambos centinela,

deja el nodo como nulo y no lo añade al árbol. Antes de comprobar esto, se mira si el siguiente carácter es un ‘.’ para no intentar crear ningún hijo a partir del nodo actual, y se come la posición del ‘.’ para no evaluarla.

Ya para escribir el árbol correctamente cargado, se va nodo por nodo mirando cada hijo y sacando el carácter del mismo, luego revisa sus dos hijos. Antes de eso, se comprueba si es un hijo vacío (nulo) o una hoja, de forma que si es lo primero se saca el # y si es una hoja sin hijos saca el carácter y a continuación un ‘.’, de esta forma se consigue leer y escribir correctamente el árbol entero en base a la cadena dada.

Todo esto lo hemos implementado mi compañero y yo en código C++ de manera funcional con un main() que lee el árbol, muestra su contenido en inorden, y luego lo escribe en un archivo “arbol.txt”, del que también puede leer el árbol. El contenido de esta implementación y todos los datos necesarios para su funcionamiento se encuentra en el .zip subido como entrega.