

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMIA

Laboratorio 4

2015-1

Indicaciones generales:

- Duración: 2h 50 min.
 - Al inicio de cada respuesta, el alumno deberá incluir, a modo de comentario, la estrategia que utilizará para resolver el problema. De no incluirse dicho comentario, el alumno perderá el derecho a reclamo en la pregunta correspondiente.
 - Si la implementación es significativamente diferente a la estrategia indicada o no la incluye, la pregunta será corregida sobre el 50 % del puntaje asignado y sin derecho a reclamo.
 - Un programa que no muestre resultados coherentes y/o útiles será corregido sobre el 60 % del puntaje asignado a dicha pregunta.
 - Debe utilizar comentarios para explicar la lógica seguida en el programa elaborado.
 - El orden será parte de la evaluación.
 - Su trabajo deberá ser subido a PAIDEIA en el espacio indicado por los Jefes de Práctica.
-

Pregunta 1 (13 puntos) Lectura y escritura de árboles

Trabajaremos con árboles en los cuales cada nodo contiene un solo caracter. Consideraremos un formato en particular para representar árboles en una cadena de caracteres y mediante el uso de paréntesis indicaremos la estructura del árbol. La cadena no tiene espacios y, para los problemas planteados, puede asumir que la cadena siempre tendrá un formato correcto. La Figura 1 muestra algunos ejemplos. Note que los árboles **no necesariamente son binarios**.

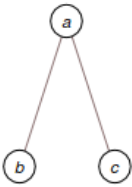
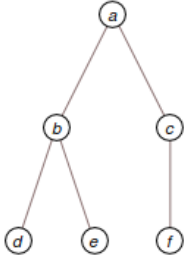
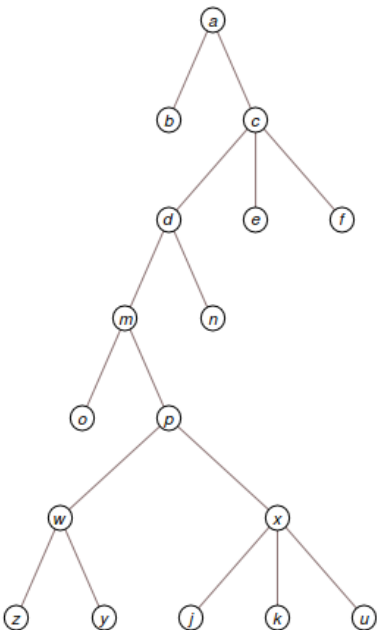
Árbol	Representación
 <pre> graph TD a((a)) --- b((b)) a --- c((c)) </pre>	<code>a(bc)</code>
 <pre> graph TD a((a)) --- b((b)) a --- c((c)) b --- d((d)) b --- e((e)) c --- f((f)) </pre>	<code>a(b(de)c(f))</code>
 <pre> graph TD a((a)) --- b((b)) a --- c((c)) c --- d((d)) c --- e((e)) c --- f((f)) d --- m((m)) d --- n((n)) m --- o((o)) m --- p((p)) p --- w((w)) p --- x((x)) w --- z((z)) w --- y((y)) x --- j((j)) x --- k((k)) x --- u((u)) </pre>	<code>a(bc(d(m(op(w(zy)x(jku)))n)ef))</code>

Figura 1: Ejemplos de representaciones de árboles

Defina las estructuras de datos necesarias (**3 puntos**) e implemente las funciones `Tree Tree_read(char* strTree)` y `void Tree_write(Tree tree, char* str)` (**10 puntos**). La función `Tree_read` recibe una cade-

na de caracteres en el formato indicado anteriormente y devuelve el árbol correspondiente. La función *Tree_write* recibe un árbol como primer parámetro y hace que la cadena *str* (segundo parámetro) contenga la representación en texto del árbol luego de ejecutar la función.

Se le proporciona el archivo *p1.c* que contiene casos de prueba. No debe modificar este archivo. Este archivo básicamente hace llamadas similares a la siguiente, pero sobre distintos casos de prueba:

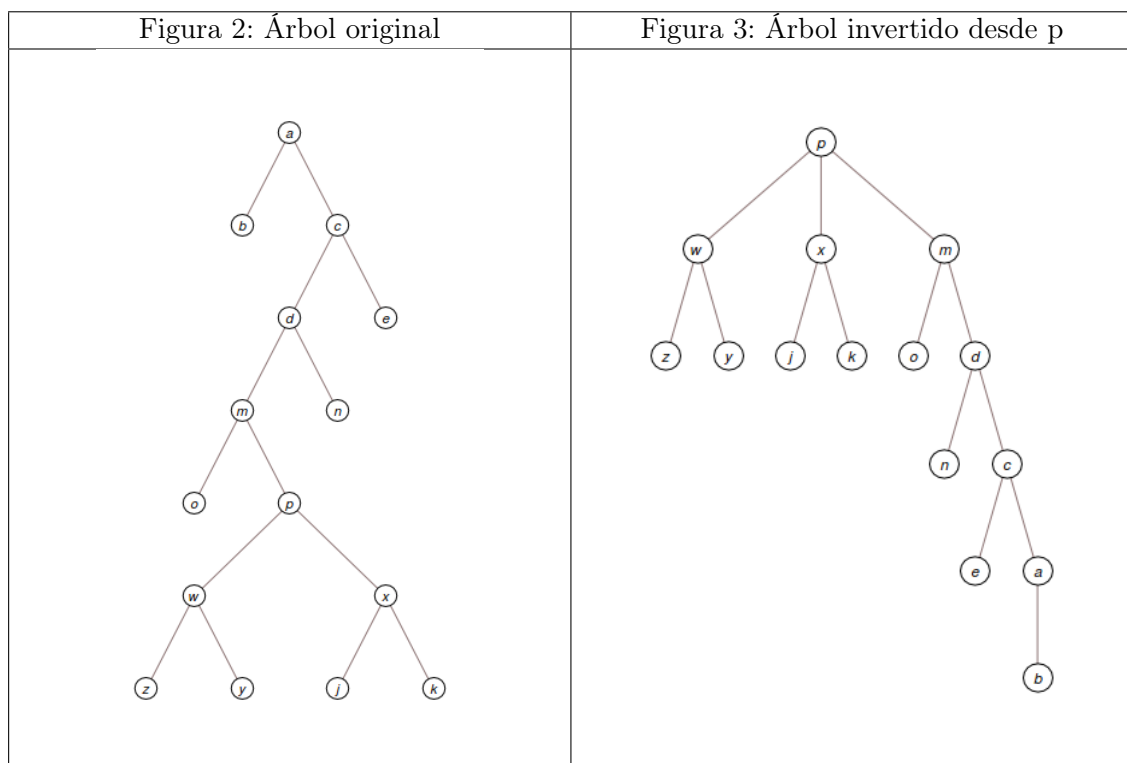
```
char result[256];
Tree_write(Tree_read("a(bc)"), result);
```

Se espera que la cadena *result* también sea "a(bc)" luego de ejecutar este ejemplo. Modifique el archivo *p1.h* con la definición de las funciones *Tree_read* y *Tree_write*. Puede crear estructuras, funciones y archivos auxiliares si lo necesita, pero **no debe modificar el archivo p1.c**. Si compila y ejecuta el archivo *p1.c* obtendrá un reporte de los casos de prueba correctos e incorrectos.

Pregunta 2 (7 puntos) Invertir árbol desde un nodo

Dado un árbol binario de caracteres (no necesariamente de búsqueda), se desea modificar el árbol de tal manera que la nueva raíz sea un nodo seleccionado. El proceso para formar el nuevo árbol consistirá en invertir la relación del nodo seleccionado con su padre e invertir la relación de todos los antecesores del nodo seleccionado con sus respectivos padres. Invertir la relación entre dos nodos **a** y **b** quiere decir que si **a** era padre de **b**, entonces **b** será padre de **a** en el nuevo árbol.

Por ejemplo, para el árbol de la Figura 2, si queremos hacer la inversión desde el nodo **p**, comenzamos con dicho nodo, cuyo padre es **m**. Ahora **m** será el último hijo del nodo **p**. Continuamos con **m**, cuyo padre es **d**, pero ahora **d** será el último hijo de **m**. De similar manera, **c** será el último hijo de **d** y **a** será el último hijo de **c**. Como **a** era la raíz original, el proceso termina en este punto y obtenemos el árbol de la Figura 3. Es importante resaltar que cada nodo afectado en este proceso sigue manteniendo sus demás hijos originales. Note también que el árbol obtenido podría no ser binario, como sucedió en este ejemplo.



Implemente la función `void Tree_invertFromNode(char* strTree, char nodeChar, char* newStrTree)`. El primer parámetro es un árbol en el formato indicado en la Pregunta 1, el segundo parámetro es el nodo seleccionado para ser la nueva raíz y la función debe modificar el tercer parámetro de manera que en él quede la representación en texto del nuevo árbol. Esta función será llamada de manera similar al siguiente ejemplo:

```
char result[256];  
Tree_invertFromNode("a(bc)", 'b', result);
```

En este caso, el contenido de la variable `result` debe ser `"b(a(c))"` luego de la llamada a la función. Puede asumir que el árbol nunca tendrá más de un nodo con el caracter dado como segundo parámetro. Se le proporciona el archivo `p2.c` que contiene casos de prueba. No debe modificar este archivo. Modifique el archivo `p2.h` con la definición de la función `Tree_invertFromNode`. Puede crear estructuras, funciones y archivos auxiliares si lo necesita, pero **no debe modificar el archivo `p2.c`**. Si compila y ejecuta el archivo `p2.c` obtendrá un reporte de los casos de prueba correctos e incorrectos.

Profesores del curso: Robert Ormeño
Fernando Alva

Pando, 9 de junio del 2015