

Tarea 4

Recorridos en Árboles Binarios

Autor: Marco Gómez F.
E-mail: marcogomezf@gmail.com

Profesor: Patricio Poblete
Auxiliar: Matías Ramírez
Sven Reisenegger
Ayudantes: Nicolás Canales
Pedro Belmonte
Tomas Vallejos
Víctor Garrido

30 de mayo de 2019
Santiago, Chile

Introducción

Los Árboles Binarios son muy versátiles, por lo mismo tienen varias formas de recorrerse, cada una con su particularidad. En el problema que se nos presenta tendremos que recorrer los árboles en **Preorden** y **Postorden**, es decir, Izquierda-Derecha-Raíz y Raíz-Izquierda-Derecha, respectivamente.

El problema consiste en diseñar un algoritmo que reciba Strings de Árboles Binarios en Postorden e imprima un String del mismo Árbol Binario recorrido en Preorden.

Para esto se programarán tres funciones:

- La primera creará un Árbol Binario a partir de un String en Postorden.
- La segunda creará un String de un Árbol Binario en recorrido Preorden.
- Finalmente un programa que reciba inputs y utilice las dos funciones anteriores para imprimir los outputs deseados.

Diseño de la solución

Estructura de Datos y TDA

Para el algoritmo se utilizaron las siguientes estructuras:

1. Árboles Binarios

Implementados por nodos con tres parámetros: Información almacenada en un char y dos punteros a dos nodos hijos izquierdo y derecho.

2. Pilas

Implementadas en un arreglo de nodos, un puntero al final y las operaciones *push* y *pop* que apilan y desapilan elementos de la pila respectivamente.

3. Colas

Implementadas a través de un arreglo de chars, el numero de elementos, dos punteros a los bordes, y las operaciones *enqueue* y *dequeue* que agregan y eliminan elementos de la cola respectivamente.

Algoritmo

Se dividió el problema en cinco funciones, a continuación se explicará brevemente su función e invariante utilizado

1. deStringACharArray

Recibe un String, lo retorna separado en caracteres en un arreglo de chars eliminando los espacios.

Para esto se utilizó un *.split()* que elimina los espacios, luego un for que recorre este nuevo arreglo creado por el split generando un String idéntico al del Input pero sin espacios que finalmente se transforma en un arreglo de chars con *.toCharArray()*.

2. generaArbol

Recibe un String correspondiente al recorrido en Post-orden de un Árbol Binario y retorna un puntero a la raíz del Árbol correspondiente.

El recorrido se transformará en un arreglo de chars con la función anterior y se leerá de izquierda a derecha. Si se encuentra un carácter punto lo apilará a la pila, de lo contrario se harán dos pops asignándolos como hijos derechos e izquierdos (en ese orden) de un nodo con la información del carácter y este será apilado. al recorrer todo el arreglo se retorna el único elemento que queda en la pila.

3. agregarACola

Recibe un puntero a la Raíz de un Árbol Binario y agrega sus elementos a una Cola recorriéndolo en Pre-orden recursivamente.

Recursivamente el caso borde será cuando la Raíz sea null, en ese caso se encola un carácter punto '.' a la Cola. De no ser este el caso se encola la información de la Raíz y se llama recursivamente a la función para los hijos izquierdos y derechos, en ese orden con la misma cola. Finalmente retorna la Raíz.

4. generaPreorden

Recibe un puntero a un Árbol y retorna su recorrido en Pre-orden.

Crea un String vacío, una Cola y llama a la función anterior con esta. Luego iterativamente decola los elementos de esta Cola agregándolos al String. finalmente retorna este String.

5. dePostordenAPreorden

Es un programa que recibe un Input del usuario utilizando *Scanner(System.in)*, para que funcione se asume que el input es un recorrido de un Árbol Binario válido en Post-orden. y utilizando las funciones anteriores imprimirá el recorrido en Pre-orden.

Implementación

TDA

■ Pila

Es un arreglo de nodos con un puntero *i* en la primera casilla vacía. El método *push (nodo)* agrega a este nodo a la casilla vacía y le suma 1 al puntero, y el método *pop* le resta 1 al puntero y retorna lo que hay en esa casilla.

■ Cola

Es un arreglo circular de caracteres de tamaño 999.999, así que amenos que se agregue un millón de caracteres funciona. Dotado de tres variables, dos son índices de los bordes de la cola y otro de la cantidad de elementos. El método *enqueue(char)* suma 1 a las variables ultimo y cantidad, luego agrega este char al final de la cola. El método *dequeue()* resta 1 a cantidad, suma 1 a primero y retorna el primer elemento en la cola.

Código 1: Cola vacía

```
1 public static class Cola{
2     char[] arreglo = new char[999999];
3     int primero = 0;
4     int ultimo = -1;
5     int cantidad = 0;
6     ...
7 }
```

Además se implementan los Nodos de forma clásica.

Funciones relevantes

A continuación se muestra como fueron implementadas las funciones en código, algunas ya han sido explicadas de forma bastante explicita en la sección anterior así que no será necesaria volver a nombrarlas.

Código 2: Función generaArbol

```
1 static Nodo generaArbol(String recorrido){
2     char[] recorridoChar = deStringACharArray(recorrido); //Crea arreglo de chars del recorrido
3     Pila pila = new Pila(recorridoChar.length); //Pila del tamaño del arreglo
4     for (char n : recorridoChar){ //Se recorre el arreglo
5         if ( n == '.' ) pila.push(null); //Caso nodo null
6         else{
7             Nodo derecho = pila.pop(); //Caso nodo no null
8             Nodo izquierdo = pila.pop();
9             Nodo nodo = new Nodo(n, izquierdo, derecho);
10            pila.push(nodo); //Se vuelve a apilar
11        }
12    }return pila.pop(); //se retorna el ultimo elemento de la pila
13 }
```

Código 3: Función agregarACola

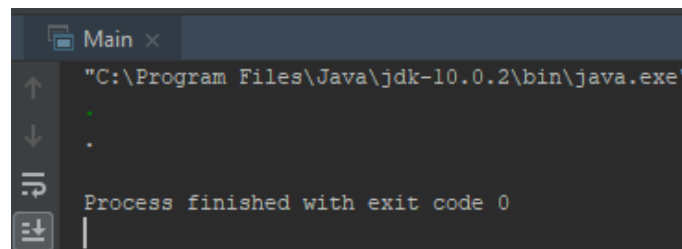
```
1 static Nodo agregarACola(Nodo raiz, Cola cola){
2     if (raiz == null)cola.enqueue('.'); //Caso base de la recursividad
3     else{                                //Recursividad en Pre-orden
4         cola.enqueue(raiz.info);        //Raíz
5         agregarACola(raiz.izq,cola);    //Izquierdo
6         agregarACola(raiz.der,cola);    //Derecho
7     }
8     return raiz;
9 }
```

Código 4: Función dePostordenAPreorden

```
1 static String generaPreorden(Nodo raiz){
2     Cola cola = new Cola();
3     //Llena los elementos del arbol a la cola en recorrido Pre-orden
4     agregarACola(raiz,cola);
5     String Preorden = ""; //String vacío
6     while (cola.cantidad != 0){ //Mientras la cola tenga elementos
7         Preorden += cola.dequeue(); //Agrega los elementos de la cola a un String
8         Preorden += " ";           //Y se separan por un espacio
9     }
10    return Preorden;
11 }
```

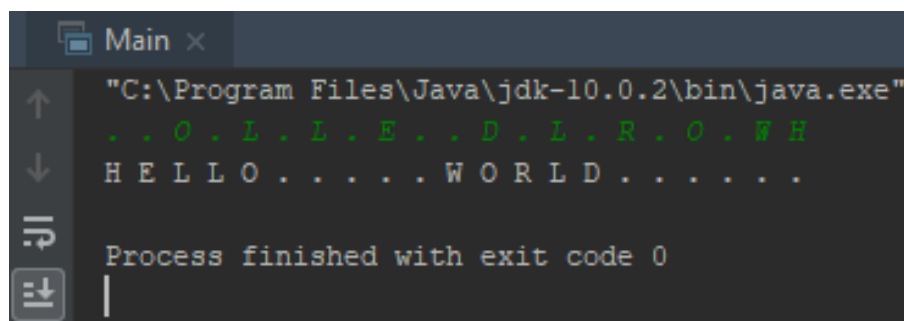
Resultados y conclusiones

Algunos ejemplos



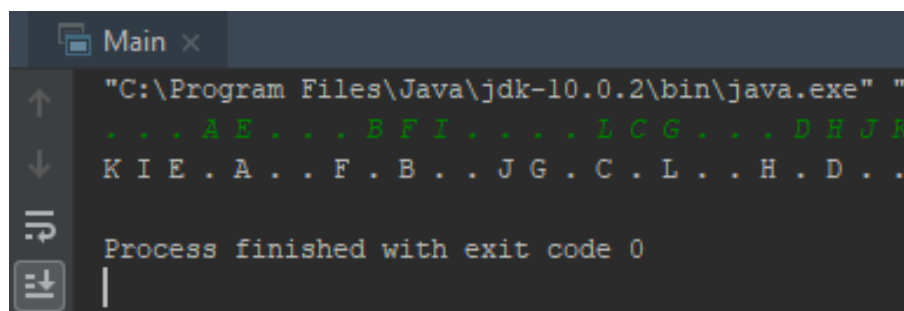
```
Main x
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe"
.
.
Process finished with exit code 0
```

(a) Input 9.



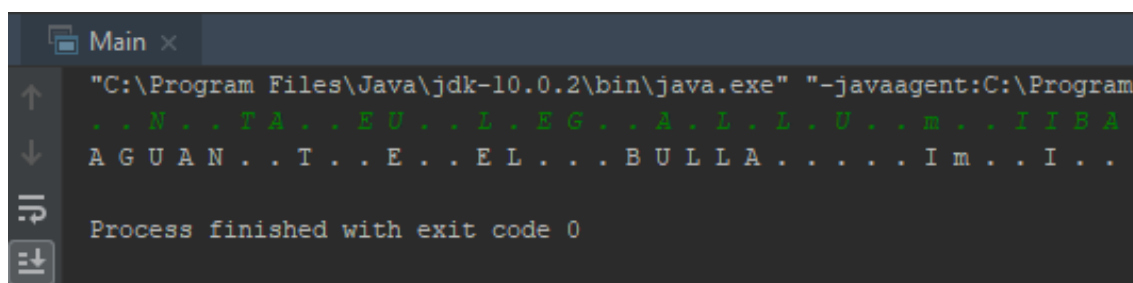
```
Main x
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe"
. . O . L . L . E . . D . L . R . O . W H
H E L L O . . . . . W O R L D . . . . .
Process finished with exit code 0
```

(b) Hello World.



```
Main x
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-
. . . A E . . . B F I . . . . L C G . . . D H J K
K I E . A . . F . B . . J G . C . L . . H . D . .
Process finished with exit code 0
```

(c) Input 2.



```
Main x
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program
. . N . . T A . . E U . . L . E G . . A . L . L . U . . m . . I I B A
A G U A N . . T . . E . . E L . . . B U L L A . . . . . I m . . I . .
Process finished with exit code 0
```

(d) Aguante el Bulla ImI.

Figura 1: Ejemplos de “casosTarea4”

Conclusiones

Como podemos notar en los ejemplos el algoritmo funciona a la perfección, incluso en casos bordes como el ejemplo (a). Por lo que podemos concluir que el uso de las Estructuras de Datos Abstractas (TDA) como Colas y Pilas son muy eficientes para resolver problemas de este tipo, al recorrer arboles binarios en Pre-orden y Post-orden, respectivamente.

El problema se separó en mas problemas pequeños lo que resultó muy útil a la hora de implementar funciones que resolvieran el algoritmo. Esto nos hace concluir que siempre es bueno fragmentar los problemas en muchos problemas pequeños ya que simplifica el trabajo tanto al momento de escribirlo como al leerlo.

Anexo A. Codigo

Código A.1: Main.java

```
1 import java.util.Scanner;
2
3 public class Main {
4
5
6 //Estructuras
7 public static class Nodo{
8     char info;
9     Nodo izq;
10    Nodo der;
11    public Nodo(char info, Nodo izq,Nodo der){
12        this.info = info;
13        this.izq = izq;
14        this.der = der;
15    }
16 }
17 public static class Pila{
18     Nodo[] arreglo;
19     int i=0;
20     public void push(Nodo u){
21         arreglo[i] = u;
22         i++;
23     }
24     Nodo pop(){
25         i--;
26         return arreglo[i];
27     }
28     public Pila(int largo){
29         this.arreglo = new Nodo[largo];
30     }
31 }
32 }
33 public static class Cola{
34     char[] arreglo = new char[999999];
35     int primero = 0;
36     int ultimo = -1;
37     int cantidad =0;
38     void enqueue(char u){
39         ultimo = (ultimo+1)%999999;
40         arreglo[ultimo] = u;
41         cantidad++;
42     }
43     char dequeue(){
44         if(cantidad != 0) {
45             cantidad--;
46             char aux = arreglo[primero];
47             primero = (primero + 1)%999999;
48             return aux;
49         }
50         return 'F';
51     }
52 }
53
54
55 //Funciones
56
57 //recibe un str y lo retorna retorna como un arreglo de char
58 static char[] deStringACharArray(String recorrido){
59     String[] arregloEnString = recorrido.split(" ");
60     String recorridoSinEspacios = "";
61     for(String n:arregloEnString){
62         recorridoSinEspacios +=n;
63     }
```

```
64     char[] arreglo=recorridoSinEspacios.toCharArray();
65     return arreglo;
66 }
67
68 //recibe un str correspondiente al recorrido en postorden y retorna un puntero a la raiz del arbol correspondiente
69 static Nodo generaArbol(String recorrido){
70     char[] recorridoChar = deStringACharArray(recorrido);
71     Pila pila = new Pila(recorridoChar.length);
72     for (char n : recorridoChar){
73         if ( n =='.') pila.push(null);
74         else{
75             Nodo derecho = pila.pop();
76             Nodo izquierdo = pila.pop();
77             Nodo nodo = new Nodo(n, izquierdo, derecho);
78             pila.push(nodo);
79         }
80     }
81     return pila.pop();
82 }
83
84 //recibe una raiz y una cola y agrega todos los info del arbol a la cola en preorden
85 static Nodo agregarACola(Nodo raiz, Cola cola){
86     if (raiz == null)cola.enqueue('.');
87     else{
88         cola.enqueue(raiz.info);
89         agregarACola(raiz.izq,cola);
90         agregarACola(raiz.der,cola);
91     }
92     return raiz;
93 }
94
95
96 static String generaPreorden(Nodo raiz){
97     Cola cola = new Cola();
98     agregarACola(raiz,cola);
99     String preorden = "";
100     while (cola.cantidad != 0){
101         preorden += cola.dequeue();
102         preorden += " ";
103     }
104     return preorden;
105 }
106
107
108 //Programa
109
110 //recibe el input e imprime el output
111 static void dePostordenAPreorden(){
112     Scanner s = new Scanner(System.in);
113     String postorden = s.nextLine();
114     Nodo Arbol = generaArbol(postorden);
115     String preorden = generaPreorden(Arbol);
116     System.out.println(preorden);
117 }
118
119 public static void main(String[] args) {
120
121     dePostordenAPreorden();
122 }
123 }
```