75.41 Algortimos y Programación II Curso 4
Arboles Iteradores

Dr. Mariano Méndez[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

26 de mayo de 2020

1. Introducción

Así como es posible abstraer el recorrido de una lista en una entidad llamada iterador, es decir, el Tipo de Dato Abstracto Iterador de Lista. En los arboles es posible también construir esta abstracción el tda iterador de arboles. En este caso existe la posibilidad de recorrer un árbol de dos formas:

- Recorrido en profundidad.
- Recorrido a lo ancho.

Un recorrido en profundidad se presenta cuando se comienza a recorrer el árbol por las hojas intentando ir en profundidad dentro de la estructura del árbol. Los recorrido en:

- pre-orden
- in-orden
- post-orden

todos ellos son recorridos en profundidad.

A continuación se vera la implementación de varios recorridos y de algunos iteradores, a partir de la siguiente declaración.

```
typedef struct nodo_t arbol_t;

struct nodo{
        void * dato;

        nodo * izquierda;
        nodo * derecha:
} nodo_t;
```

2. Recorridos en Profundidad (DFS)

2.1. In-Order

El recorrido primero se visita el sub árbol izquierdo luego el nodo por el que se ingreso y finalmente se visita el sub árbol derecho (IND). Normalmente se utiliza par obtener los nodos en orden ascendente. A continuación se implementara una versión en una función recursiva:

```
void inorden (nodo_t * nodo){

        if (!nodo)
                return;

        if(nodo->izquierda)
                inorden(nodo->izquierda);

        process(nodo);

        if(nodo->derecha)
                inorden(nodo->derecha);
}
```

2.2. Post-Order

El recorrido primero se visita el sub árbol izquierdo luego, visita el sub árbol derecho y por ultimo el nodo en cuestión (IDN). Es utilizado principalmente para vaciar el árbol. A continuación se implementara una versión en una función recursiva:

```
void postorden (nodo_t * nodo){

        if (!nodo)
                return;

        if(nodo->izquierda)
                postorden(nodo->izquierda);

        if(nodo->derecha)
                postorden(nodo->derecha);

        process(nodo);
}
```

2.3. Pre-Order

El recorrido primero se visita el sub árbol izquierdo luego, visita el sub árbol derecho y por ultimo el nodo en cuestión (NID). Es utilizado principalmente para vaciar el árbol. A continuación se implementara una versión en una función recursiva:

```
void preorden (nodo_t * nodo){

        if (!nodo)
                return;

        process(nodo);

        if(nodo->izquierda)
                preorden(nodo->izquierda);

        if(nodo->derecha)
                preorden(nodo->derecha);
}
```

2.4. Iterador externo

Un iterador es una abstracción del recorrido de un arbol, con lo cual debe tener ciertas operaciones:

- crear(arbol_t arbol): crea el iterador
- destruir(): destruye el iterador;
- corriente(): devuelve el elemento en el cual se esta.
- siguiente(): devuelve el elemento siguiente del recorrido.

Para realizar un iterador externo de un árbol este debe hacerse de forma iterativa. Para ello es necesario utilizar una estructura auxiliar que es una Pila en este caso.

```
#include"pila.h"

typedef struct iterador iterador_t;

struct iterador{
        pila_t * pila;
}iterador_t;


iterador_t * crear(arbol_t* root){
        iterador_t * it;

        it = malloc( sizeof(iterador_t) );

        if(!it)
                return NULL;

        it->pila = crear_pila();
```

```
19
20          if(it->pila)
21                  push(it->pila,root);
22
23          return it;
24  }
25
26  void * corriente(iterador_t*  iterador){
27          nodo_t curr = tope(iterador->pila);
28          return curr->elemento;
29  }
30
31  void * siguiente(iterador_t *  iterador){
32
33          nodo_t * current=pop(iterador->pila);
34
35          if (current->derecho)
36                  push(iterador->pila,current->derecho):
37
38          if (current->izquierdo)
39                  push(iterador->pila,current->izquierdo);
40
41          return current->elemento;
42  }
43
44  void destruir(iterador_t iterador){
45
46          if(!iterador)
47                  return;
48
49          destruir(iterador->pila);
50          free(iterador);
51  }
```

3. Recorrido a lo Largo (BSF)

Al contrario que la DFS, este tipo de recorrido lo que intenta es recorrer por niveles los nodos del árbol.

```
1   #include"cola.h"
2
3   void recorrido_en_anchura(arbol_t arbol){
4           cola_t cola;
5           nodo_t corriente;
6
7           if(!arbol)
8                   return;
9
10          cola=crear();
11
12          encolar(cola,arbol);
13
14          while (!vacia(cola)) {
15                  corriente = desencolar(cola);
16                  // hacer algo
17
18                  if (corriente->izquierda != NULL)
19                          encolar(cola, corriente->izquierda);
20
21                  if (corriente->derecha != NULL)
22                          encolar(cola, corriente->derecha);
23          }
24
25          destruir(cola);
26  }
```

Referencias