2015 Septiembre

J. Viviana Molano
Libardo Pantoja Y.
Departamento de Sistemas
Universidad del Cauca

LIBRO

ESTRUCTURAS DE DATOS DINAMICAS

Una manera fácil de aprender

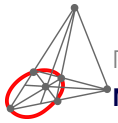
Editorial Unicauca
Departamento de Sistemas, Universidad del Cauca

J. Viviana Mora., W Libardo Pantoja Y.

Estructuras de Datos

Estructuras de Datos Dinámicas

Una forma fácil de aprender



Revista digital
Matemática, Educación e Internet. (http://www.tec-digital.itcr.ac.cr/revistamatematica/).

Correo Electrónico: editorial@unicauca.edu.co
Departamento de Sistemas
Facultad de Ingeniería Electrónica y Telecomunicaciones
Apdo. 159-7050, Popayán
Teléfono (057)8209800
Fax (058)8209900

Mora, Viviana.
Estructuras de Datos Dinámicas
W. Libardo Pantoja Yépez
– Departamento de Sistemas, FIET, Universidad del Cauca. 2015.
xxx p.
ISBN 978-9977-66-227-5
1. Estructuras. 2. de Datos 3. Dinámicas.

Licencia.

Revista digital
Matemática, Educación e Internet.
http://www.tec-digital.itcr.ac.cr/revistamatematica/.

Índice general

Prólogo VII

1 Capítulo 1: Introducción a las Estructuras de Datos 1

1.1 Prueba de entornos 1

Tablas 2

2 Capítulo 2. XXXXXXXX 3

2.1 Introduccion 3

Ejemplo codigo fuente Java 3

3 Capítulo 3. XXXXXXXX 5

3.1 Introduccion 5

4 Pilas 7

4.1 Definición de Pila 7

4.2 El TAD Pila 7

4.3 Implementación del TAD en Java 8

5 Arboles Binarios 13

5.1 Conceptos generales 13

5.2 Tipos de arboles 13

Arboles binarios 13

Arboles AVL 13

Prólogo

Este texto cubre aspectos básicos e intermedios sobre ipsúm dolor sit amet, consectetur adipiscing elit. Nam dignissim varius tempus. Cras eu malesuada ipsum. Pellentesque ut lorem velit. Mauris vehicula est orci, bibendum tincidunt enim mattis a. Interdum et malesuada fames ac ante ipsum primis in faucibus..

...

Popayan, 2015. V. MORA, W. PANTOJA.

1 Capítulo 1: Introducción a las Estructuras de Datos

Advertencia.

Las siguientes plantillas usan la versión 2014 del paquete `tcolorbox` (entre otros paquetes recientes), por lo tanto *debe actualizar los paquetes de sus distribución* TEX o instalar manualmente este paquete (ver el capítulo 9 del libro, `http://www.tec-digital.itcr.ac.cr/revistamatematica/Libros/LATEX/LaTeX_2014.pdf`). El paquete "psboxit" viene incluido en la carpeta.

1.1 Prueba de entornos

Definición 1.1 (Igualdad)

$$a = b$$

Según la definición 1.1, la igualdad...

Teorema 1.1

$$a = b$$

Ejemplo 1.1

$$a = b$$

Lema 1.1

$$a = b$$

Corolario 1.1

$$a = b$$

Una caja de comentario

$$a = b$$

1.1.1 Tablas

Iteración		
	x_i	$y_i = f(x_i)$
A	$x_0 = 0$	0
B	$x_1 = 0{,}75$	$-0{,}0409838$
C	$x_2 = 1{,}5$	1,31799

2 Capítulo 2. XXXXXXXX

2.1 Introduccion

Lorem ipsúm dolor sit amet, consectetur adipiscing elit. Nam dignissim varius tempus. Cras eu malesuada ipsum. Pellentesque ut lorem velit. Mauris vehicula est orci, bibendum tincidunt enim mattis a. Interdum et malesuada fames ac ante ipsum primis in faucibus. Sed mi justo, facilisis eget eros at, tristique tempus metus. Suspendisse potenti. Vivamus sed tellus mollis, accumsan ipsum a, auctor ex. Duis ullamcorper quam ipsum. Donec ullamcorper porttitor pretium. Curabitur urna nunc, placerat sit amet et, fermentum vehicula purus. Pellentesque eget mi ex [**?**].

2.1.1 Ejemplo codigo fuente Java

```
1 package com.unicauca.ejemplo;
2 public class Hello {
3   //Comentario
4   /*Comentario*/
5   /**Comentario*/
6   public static void main(String[] args) {
7     System.out.println("Hola mundo");
8  }
9 }
```

Ahora compila usando `javac`:

```
$ javac HolaMundo.java
```

3 Capítulo 3. XXXXXXXX

3.1 Introduccion

Lorem ipsúm dolor sit amet, consectetur adipiscing elit. Nam dignissim varius tempus. Cras eu malesuada ipsum. Pellentesque ut lorem velit. Mauris vehicula est orci, bibendum tincidunt enim mattis a. Interdum et malesuada fames ac ante ipsum primis in faucibus. Sed mi justo, facilisis eget eros at, tristique tempus metus. Suspendisse potenti. Vivamus sed tellus mollis, accumsan ipsum a, auctor ex. Duis ullamcorper quam ipsum. Donec ullamcorper porttitor pretium. Curabitur urna nunc, placerat sit amet et, fermentum vehicula purus. Pellentesque eget mi ex [**?**].

4 Pilas

4.1 Definición de Pila

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. Se aplica en multitud de ocasiones en informática debido a su simplicidad y ordenación implícita en la propia estructura.
La Figura 4.2 muestra la representación gráfica de una pila con sus operaciones fundamentales de apilar (push en inglés) y desapilar o retirar (pos en inglés.).
La pila es muy útil en situaciones cuando los datos deben almacenarse y luego recuperarse en orden inverso.

Definición 4.1 Pila

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. FALTA REFERENCIA

4.2 El TAD Pila

A continuación se especifica el TAD de la Pila con sus operaciones fundamentales. Las operaciones apilar y desapilar son las más importantes. En seguida la especificación de cada operación del TAD al estilo C.

```
TAD Pila [ T ]
```

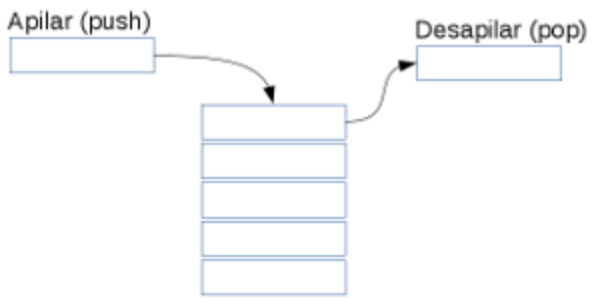


Figura 4.1. Representación de una Pila

```
{ invariante: TRUE }
Constructoras:
  crearPila:
Modificadoras:
   apilar: Pila T
   desapilar: Pila
Analizadoras:
   cima: Pila
   esVacia: Pila
Destructora:
   destruirPila: Pila

Pila crearPila( void )
/* Crea una pila vacia */
{ post: crearPila = }

void apilar(Pila pil, T elem)
/* Coloca sobre el tope de la pila el elemento elem */\
{ post: pil = e1, e2, .. elem}

void desapilar(Pila pil)\\
/* Elimina el elemento que se encuentra en el tope de la pila */\\
{ pre: pil =e1, e2, ..en, n > 0 }
{ post: pil =e1, e2, .., en-1 }

T cima(Pila pil )
/* Retorna el elemento que se encuentra en el tope de la pila */
{ pre: n > 0 }
{ post: cima = en }

int esVacia( Pila pil )
/* Informa si la pila esta vacia */
{ post: esVacia = ( pil = ) }

void destruirPila( Pila pil )
/* Destruye la pila retornando toda la memoria ocupada */
{post: pil ha sido destruida }
```

4.3 Implementación del TAD en Java

A continuación se muestra una implementación en Java del TAD Pila. Debido a que es una Pila dinámica se utilizan nodos enlazados. La Figura **??** muestra el diagrama de clases.
La implementación involucra básicamente dos clases: Nodo y Pila.

```
1 package co.unicauca.pilas;
2 public class Nodo<T> {
3    //Atributo valor de tipo T. Almacena la referencia al objeto que se guarda
```

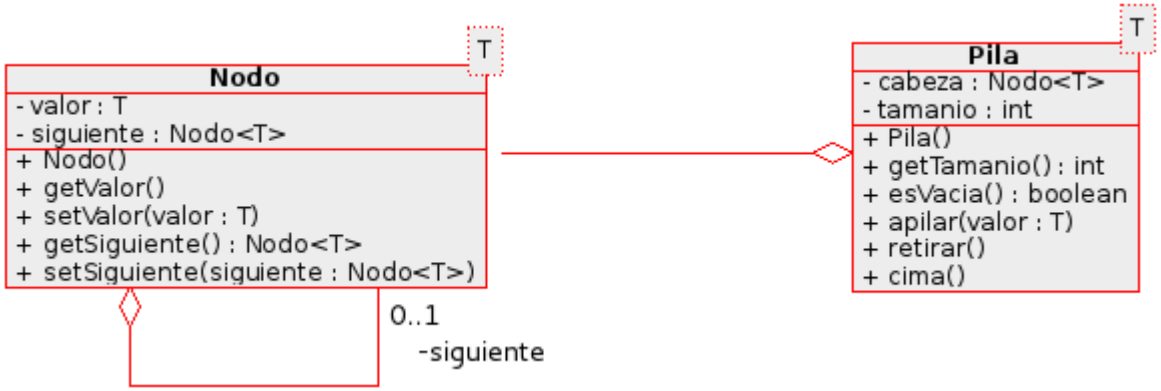


Figura 4.2. Diagrama de clase de la implementación del TAD Pila

```
      en el nodo
4   private T valor;
5   //Referencia al siguiente nodo enlazado
6   Nodo<T> siguiente;
7   //Constructor por defecto
8   public Nodo() {
9      valor = null;
10     siguiente = null;
11  }
12  //Devuelve el valor
13  public T getValor() {
14     return valor;
15  }
16  //Modifica el valor
17  public void setValor(T valor) {
18     this.valor = valor;
19  }
20  //Devuelve el atributo siguiente
21  public Nodo<T> getSiguiente() {
22     return siguiente;
23  }
24   //Modifica el atributo siguiente
25  public void setSiguiente(Nodo<T> siguiente) {
26     this.siguiente = siguiente;
27  }
28 }
```

La clase Nodo representa cada uno de los nodos enlazados que almacenan los objetos que se apilan. Tiene dos atributos, el *valor* representa el valor que guarda el nodo (línea 4), en este caso es una referencia a un objetivo de tipo T (siendo T un tipo genérico). El atributo *siguiente* (línea 6), representa la referencia al siguiente nodo. Los demás son únicamente, constructor y getters y setters de cada atributo.

```
1 package co.unicauca.pilas;
2 public class Pila<T> {
```

```
3     //Atributo cabeza, que apunta al tope la pila
4     private Nodo<T> cabeza;
5     //Almacena el total de elemento de la pila
6     private int tamanio;
7     //Constructor por defecto
8     public Pila() {
9        cabeza = null;
10       tamanio = 0;
11    }
12    //Devuelve el total de elementos de la pila
13    public int getTamanio() {
14       return tamanio;
15    }
16    //Verifica si la pila esta vacia
17    public boolean esVacia() {
18       return (cabeza == null)
19    }
20    //Apila un elemento nuevo
21    public void apilar(T valor) {
22       //Crear un nuevo Nodo
23       Nodo<T> nuevo = new Nodo<T>();
24       //Fijaer el valor dentro del nodo
25       nuevo.setValor(valor);
26       if (esVacia()) {
27          //Cabeza apunta al nodo nuevo
28          cabeza = nuevo;
29       } else {
30          //Se enlaza el campo siguiente de nuevo con la cabeza
31          nuevo.setSiguiente(cabeza);
32          //La nueva cabeza de la pila pasa a ser nuevo
33          cabeza = nuevo;
34       }
35       //Incrementa el tamanio porque hay un nuevo elemento en la pila
36       tamanio++;
37    }
38    //Elimina un elemento de la pila
39    public void retirar() {
40       if (!esVacia()) {
41          cabeza = cabeza.getSiguiente();
42          tamanio--;
43.       }
44    }
45    //Devuelve el elemento almacenado en el tope de la pila
46    public T cima() {
47       if (!esVacia())
48          return cabeza.getValor();
49       else
```

```
50          return null;
51      }
52  }
```

La clase Pila representa la pila como tal con sus operaciones principales de *apilar* y *retirar*.
A continución el código de un Cliente que instancia la Pila que hemos creado. En este caso se almacenan objetos de tipo entero, se apilan algunos números, se imprimen los valores del tope de pila y se desapilan elementos.

```
1  package co.unicauca.pilas;
2  public class ClienteMain {
3     public static void main(String[] args) {
4        //Crear una nueva pila de enteros
5        Pila<Integer> pila2 = new Pila<Integer>();
6        //Se apilan algunos datos enteros
7        pila2.apilar(2);
8        pila2.apilar(5);
9        pila2.apilar(7);
10       System.out.println("El tope de la pila es: " + pila2.cima());
11       //Se desapila
12       pila2.retirar();
13       System.out.println("El tope de la pila es: " + pila2.cima());
14       //Se desapila
15       pila2.retirar();
16       System.out.println("El tope de la pila es: " + pila2.cima());
17       //Se desapila, como la pila esta vacia devuelve null
18       pila2.retirar();
19       System.out.println("El tope de la pila es: " + pila2.cima());
20       //Probar con otra pila, donde se almacenen objetos
21    }
22 }
```

La salida de este programa sería la siguiente.

```
El tope de la pila es: 7
El tope de la pila es: 5
El tope de la pila es: 2
El tope de la pila es: null
```

5 Arboles Binarios

5.1 Conceptos generales

Un arbol es una estructura daksfjaksdjf kajf kdafasd f dajf sdaf as f asdf dsafk asdkf sdaf adf dasf ak sdfasdf asdfas f asd fadf df.

5.2 Tipos de arboles

5.2.1 Arboles binarios

5.2.2 Arboles AVL

```java
package com.unicauca.ejemplo;
public class ArbolBianrio {
  //Comentario
  /*Comentario*/
  /**Comentario*/
  public static void main(String[] args) {
    System.out.println("Hola mundo");
 }
}
```