

Estructura de datos

Marks Calderón Niquin

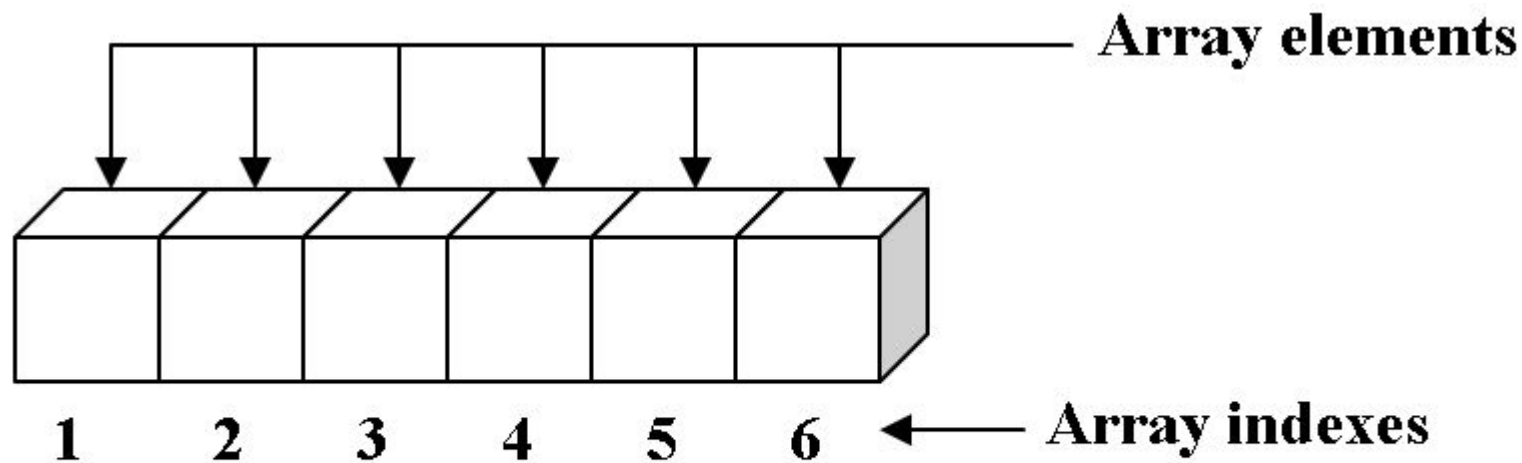
Temario

— — —

- Pilas
- Deque
- Colas
- Listas
- Árboles

Estructuras lineales

Son colecciones de datos



One-dimensional array with six elements

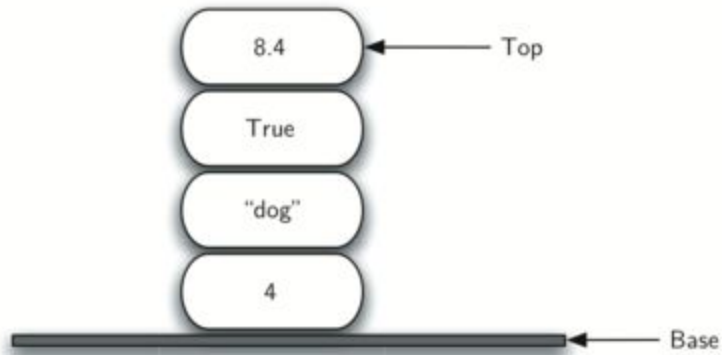
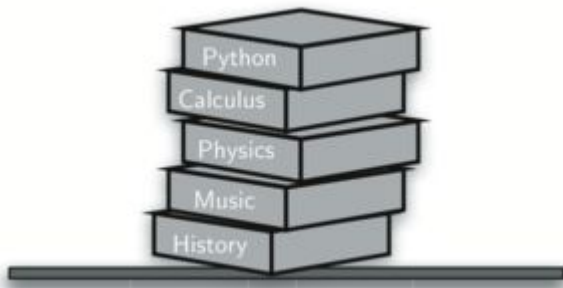


Pilas

Pilas

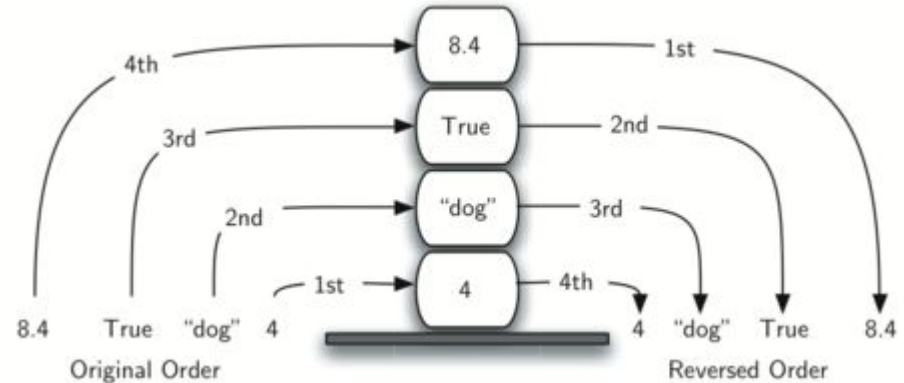
— — —

- Es un tipo de colección diferente
- Mantiene la idea de LIFO (Last-in First-out)
- Nuevos elementos van a la cima y los antiguos a la base



Pilas

- El orden de remover elementos es el inverso de insertarlos
- Son muy empleados para invertir elementos



Tipo abstracto de datos

— — —

- Una pila es un tipo abstracto de datos definido por su estructura y operaciones.
- Una pila es estructurada como una colección ordenada de items donde estos son añadidos y removidos desde la cima

Tipo abstracto de datos

— — —

Operación Pila	Contenido de Pila	Valor retornado
<code>s.isEmpty()</code>	<code>[]</code>	<code>True</code>
<code>s.push(4)</code>	<code>[4]</code>	
<code>s.push('dog')</code>	<code>[4, 'dog']</code>	
<code>s.peek()</code>	<code>[4, 'dog']</code>	<code>'dog'</code>
<code>s.push(True)</code>	<code>[4, 'dog', True]</code>	
<code>s.size()</code>	<code>[4, 'dog', True]</code>	<code>3</code>

Tipo abstracto de datos

— — —

Operación pila	Contenido de pila	Valor retornado
<code>s.isEmpty()</code>	<code>[4, 'dog', True]</code>	<code>False</code>
<code>s.push(8.4)</code>	<code>[4, 'dog', True, 8.4]</code>	
<code>s.pop()</code>	<code>[4, 'dog', True]</code>	<code>8.4</code>
<code>s.pop()</code>	<code>[4, 'dog']</code>	<code>True</code>
<code>s.size()</code>	<code>[4, 'dog']</code>	<code>2</code>

Implementación

— — —

```
s= Pila()
s
print(s.isEmpty())
s.push(4)
s.push('dog')
print(s.peak())
s.push(True)
print(s.size())
print(s.isEmpty())
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

```
class Pila:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

Casos de uso

— — —

Balanceo de paréntesis(Correcto)

```
{ { ( [ ] [ ] ) } ( ) }
```

```
[ [ { { ( ( ) ) } } ] ]
```

```
[ ] [ ] [ ] ( ) { }
```

Balanceo de paréntesis(Incorrecto)

```
( [ ) ]
```

```
( ( ( ) ] ) )
```

```
[ { ( ) ]
```

			{	{	{	{							
		((((()						
	{	{	{	{	{	{	{	}	()			
{	{	{	{	{	{	{	{	{	{	}	}		

Ejemplo incorrecto

— — —

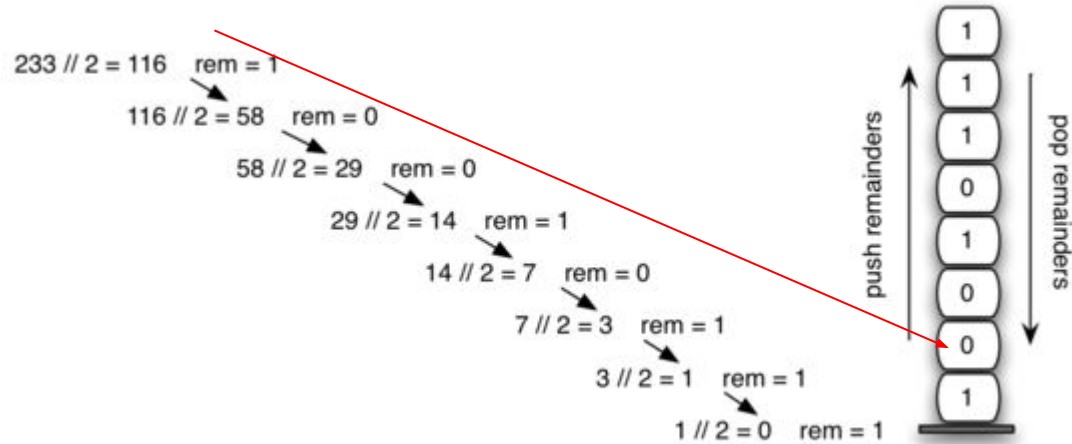
```
( ( ( ) ] ) )
```

		(
	((((
(((((

Casos de uso

— — —

Decimal a binario



11101001

Casos de uso: notacion postfija

— — —

3 + 4 = 5 (notacion infija)
+ 3 4 = 5 (notación prefija)
3 4 + = 5 (notación postfija o
notacion polaca)

2*3 + 4*2

2 3 *

4 2 *

2 3 * 4 2 * + sqrt

2	3	*	4	2	*	+	sqrt
				2			
	3		4	4	8		
2	2	6	6	6	6	14	3....

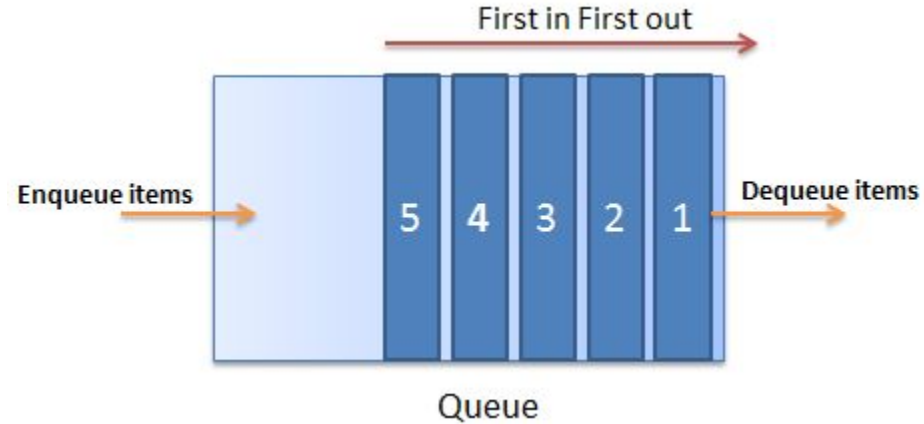


Colas

Colas

— — —

- Es una colección de datos donde el añadir nuevos items es al final, este es llamado **rear**
- Mantiene la idea FIFO(First-in First-out)
-



Tipo abstracto de datos

— — —

- Una cola es un tipo abstracto de datos definido por su estructura y operaciones.
- Una cola es estructurada como una colección ordenada de items donde estos son añadidos al inicio y removidos desde el final

Tipo abstracto de datos

— — —

Operaciones de Cola	Contenido de Cola	Valores retornados
<code>q.isEmpty()</code>	<code>[]</code>	<code>True</code>
<code>q.enqueue(4)</code>	<code>[4]</code>	
<code>q.enqueue('dog')</code>	<code>['dog', 4]</code>	
<code>q.enqueue(True)</code>	<code>[True, 'dog', 4]</code>	
<code>q.size()</code>	<code>[True, 'dog', 4]</code>	<code>3</code>

Tipo abstracto de datos

— — —

Operaciones de Cola	Contenido de cola	Valor retornado
<code>q.isEmpty()</code>	<code>[True, 'dog', 4]</code>	<code>False</code>
<code>q.enqueue(8.4)</code>	<code>[8.4, True, 'dog', 4]</code>	
<code>q.dequeue()</code>	<code>[8.4, True, 'dog']</code>	<code>4</code>
<code>q.dequeue()</code>	<code>[8.4, True]</code>	<code>'dog'</code>
<code>q.size()</code>	<code>[8.4, True]</code>	<code>2</code>

Implementación

```
class Cola:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

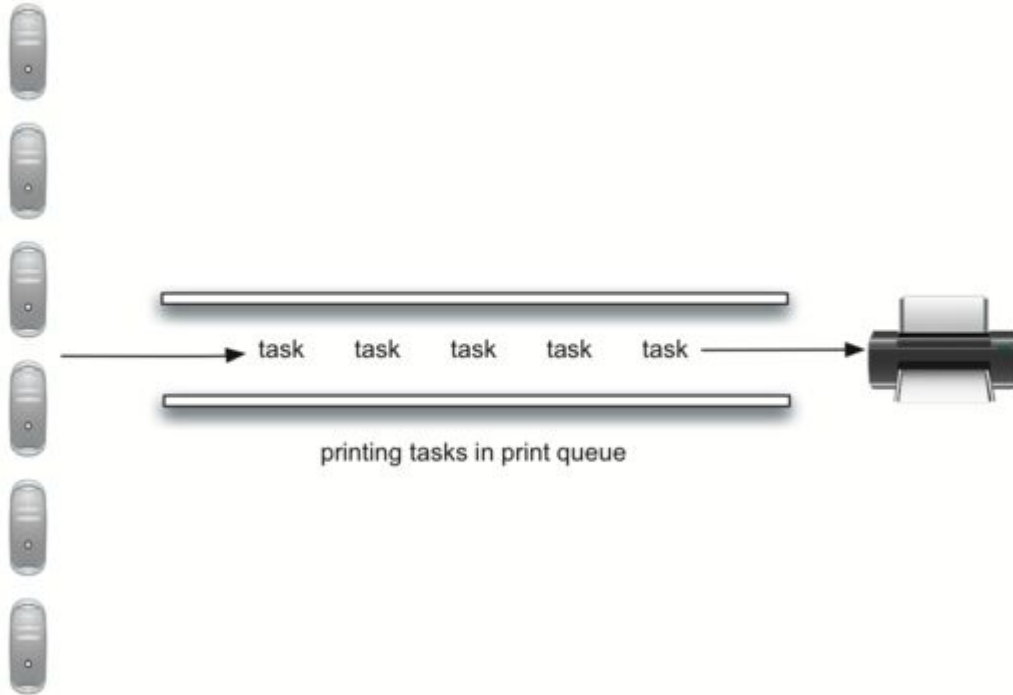
    def size(self):
        return len(self.items)
```

```
q=Cola()
q.enqueue(4)
q.enqueue('dog')
q.enqueue(True)
print(q.size())
```

Caso de uso: Impresión de tareas

— — —

Lab Computers

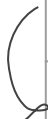


Caso: invertir elementos de una cola

— — —

Abcde

Edcba



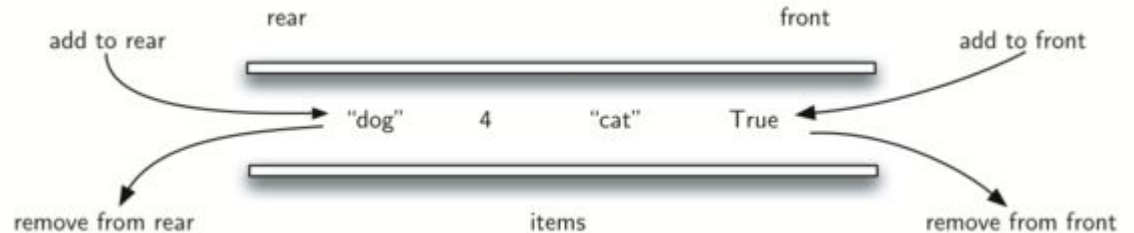
a	b	c	d	e
e	d	c	b	a

Deque

Deque

— — —

- Deque es conocida como una cola doblemente enlazada
- Es una colección ordenada similar a la cola, la diferencia es que se puede añadir/remove elementos al inicio o final



Tipo abstracto de datos

— — —

Operación deque	Contenido deque	Valor retornado
<code>d.isEmpty()</code>	<code>[]</code>	<code>True</code>
<code>d.addFront(4)</code>	<code>[4]</code>	
<code>d.addFront('dog')</code>	<code>['dog', 4]</code>	
<code>d.addRear('cat')</code>	<code>['dog', 4, 'cat']</code>	
<code>d.addRear(True)</code>	<code>['dog', 4, 'cat', True]</code>	

Tipo abstracto de datos

— — —

Operación Deque	Contenido Deque	Valor retornado
<code>d.size()</code>	<code>['dog', 4, 'cat', True]</code>	<code>4</code>
<code>d.isEmpty()</code>	<code>['dog', 4, 'cat', True]</code>	<code>False</code>
<code>d.addFront(8.4)</code>	<code>[8.4, 'dog', 4, 'cat', True]</code>	
<code>d.removeFront()</code>	<code>['dog', 4, 'cat', True]</code>	<code>8.4</code>
<code>d.removeRear()</code>	<code>['dog', 4, 'cat']</code>	<code>True</code>

Implementación

```
d=Deque()  
print(d.isEmpty())  
d.addRear(4)  
d.addRear('dog')  
d.addFront('cat')  
d.addFront(True)  
print(d.size())  
print(d.isEmpty())  
d.addRear(8.4)  
print(d.removeRear())  
print(d.removeFront())
```

class Deque:

def `__init__`(self):

self.items = []

def isEmpty(self):

return self.items == []

def addFront(self, item):

self.items.append(item)

def addRear(self, item):

self.items.insert(0,item)

def removeFront(self):

return self.items.pop()

def removeRear(self):

return self.items.pop(0)

def size(self):


return len(self.items)

Caso: invertir elementos usando un deque

— — —

Abcde

Edcba

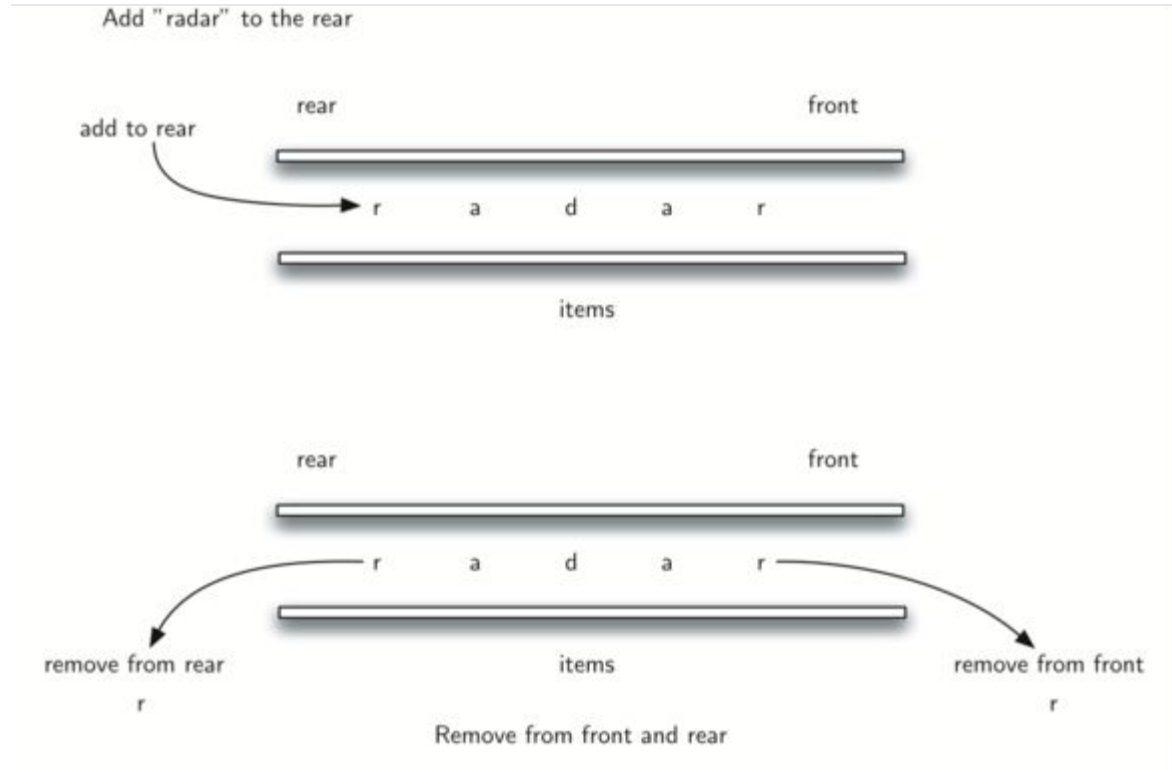


a	b	c	d	e
e	d	c	b	a

a				
a	b			
a	b	c		
a	b	c	d	
a	b	c	d	e

Caso de uso: Verificador de palindromos

— — —



-- --

R	A	D	A	R
---	---	---	---	---

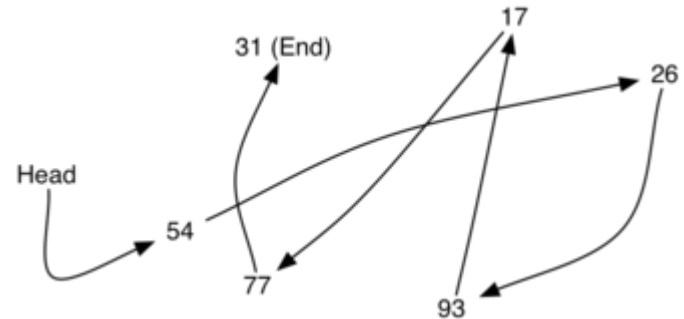
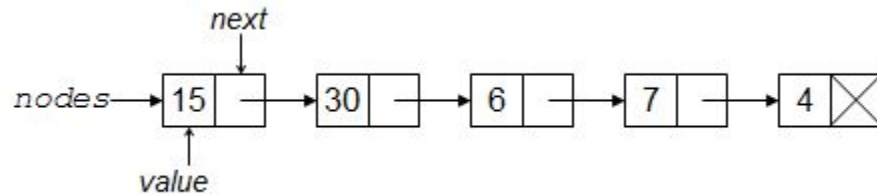
A	N	N	A
---	---	---	---

Listas



Lista Enlazada

- Es una colección de items donde cada uno de ellos se mantiene uno al siguiente
- Podemos tener acceso al primer o último elemento, pero solo tenemos acceso a todos los elementos a través del primer.
- También se le puede denominar una lista desordenada



Tipo abstracto de datos

- `List()` crea una nueva lista que esta vacia.
- `add(item)` añade un nuevo item a la lista. Es necesario el item y no retorna nada.
- `remove(item)` remueve el item de la lista. Es necesario el item y modifica la lista. Asume que el item es presente en la lista.
- `search(item)` busca el item en la lista. Retorna un valor booleano
- `isEmpty()` prueba si la lista esta vacia o no. Retorna un valor booleano.
- `size()` retorna el número de items en la lista. Retorna un valor entero.
- `append(item)` añade un nuevo item al final de la lista.
- `index(item)` retorna la posición del item en la lista. Es necesario el item y retorna el indice.
- `insert(pos,item)` añade un nuevo item a la lista a la posición pos.
- `pop()` remueve y retorna el ultimo item de la lista. Asumimos que la lista tiene por lo menos un item.
- `pop(pos)` remueve y retorna el item a la posición pos. Es necesario la posición y retornará el item

Implementación en Python

class Nodo:

```
def __init__(self, initdata):  
    self.data = initdata  
    self.next = None
```

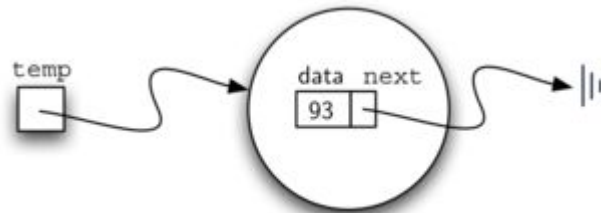
```
def getData(self):  
    return self.data
```

```
def getNext(self):  
    return self.next
```

```
def setData(self, newdata):  
    self.data = newdata
```

```
def setNext(self, newnext):  
    self.next = newnext
```

```
temp = Nodo(93)  
temp.getData()
```



Implementación en Python

— — —

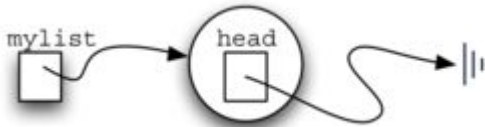
```
class UnorderedList:
```

```
    def __init__(self):
```

```
        self.head = None
```

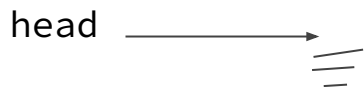
```
mylist = UnorderedList()
```

```
    def isEmpty(self):  
        return self.head == None
```

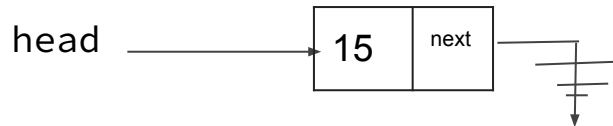
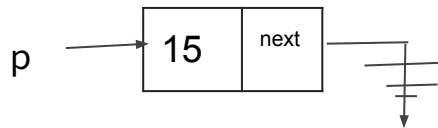


Add agregar un elemento

Lista

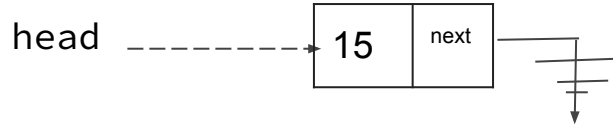
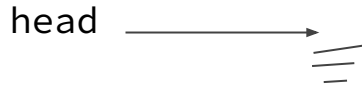


Nodo

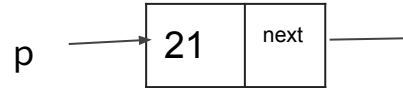
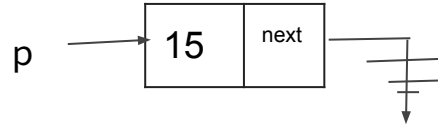


Add agregar un elemento

Lista

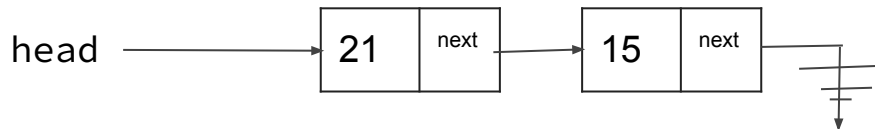


Nodo



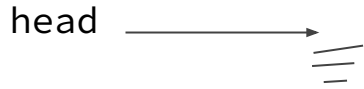
`.setNext(v)`
`.getNext()`

`p.setNext(head)`
`head = p`

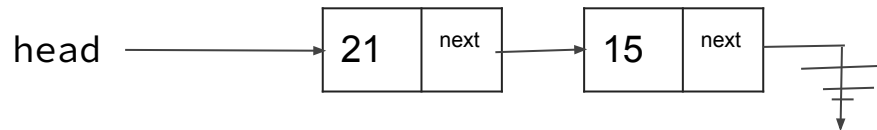
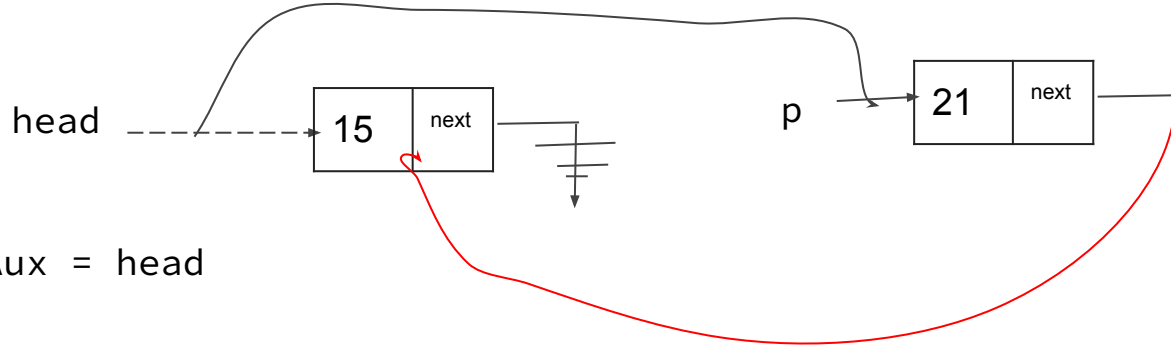
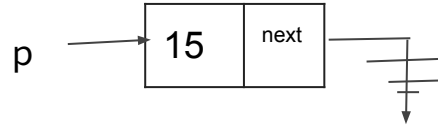


Add agregar un elemento

Lista



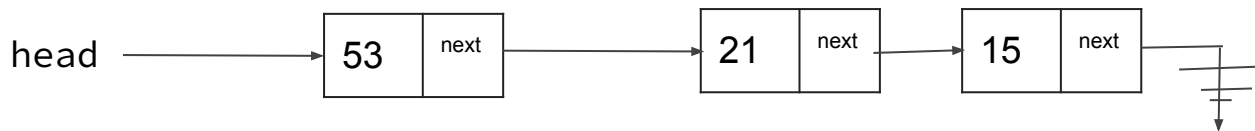
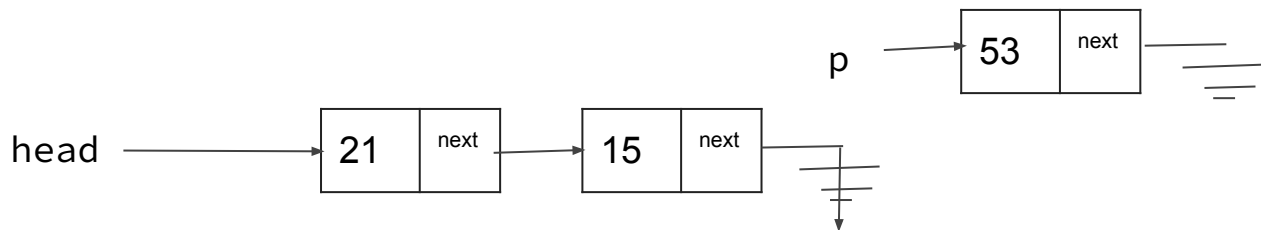
Nodo



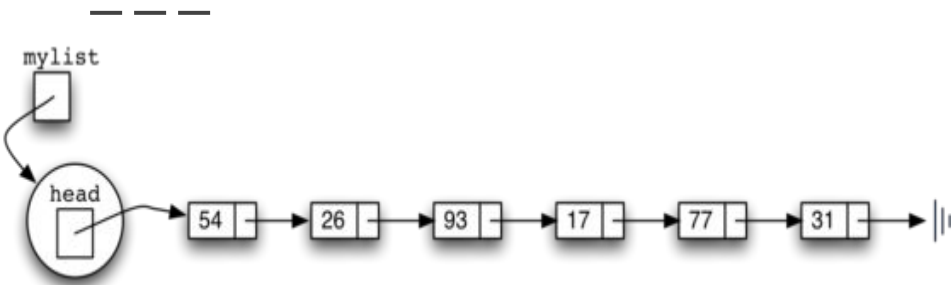
Add agregar un elemento

Lista

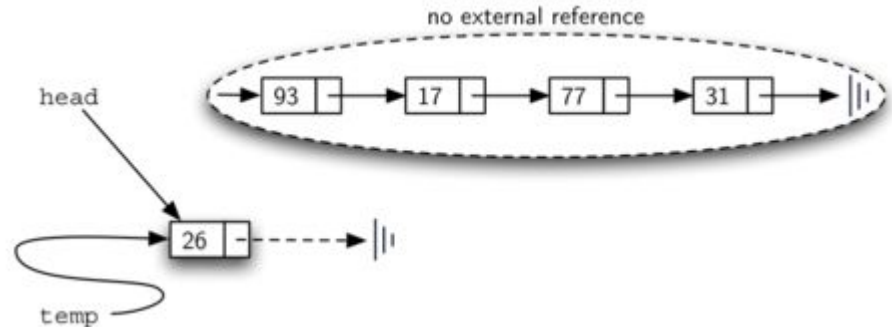
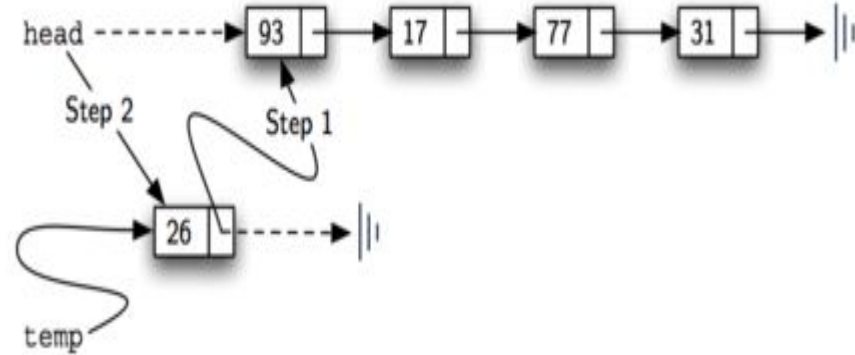
Nodo



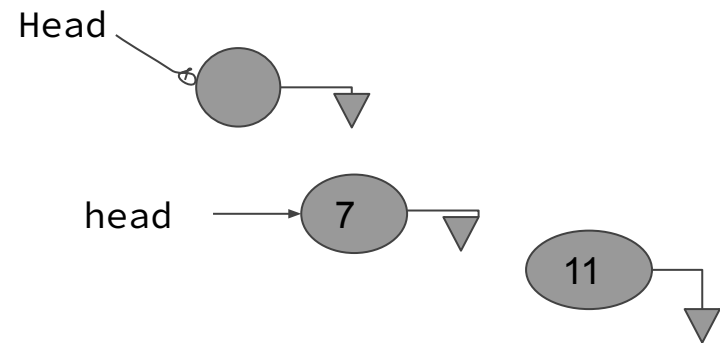
Implementación en Python



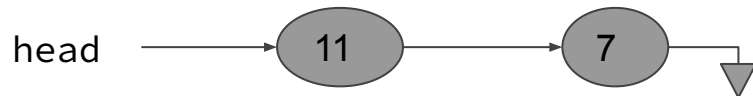
```
def add(self,item):  
    p = Nodo(item)  
    p.setNext(self.head)  
    self.head = p
```



Ejemplo: agregar elemento



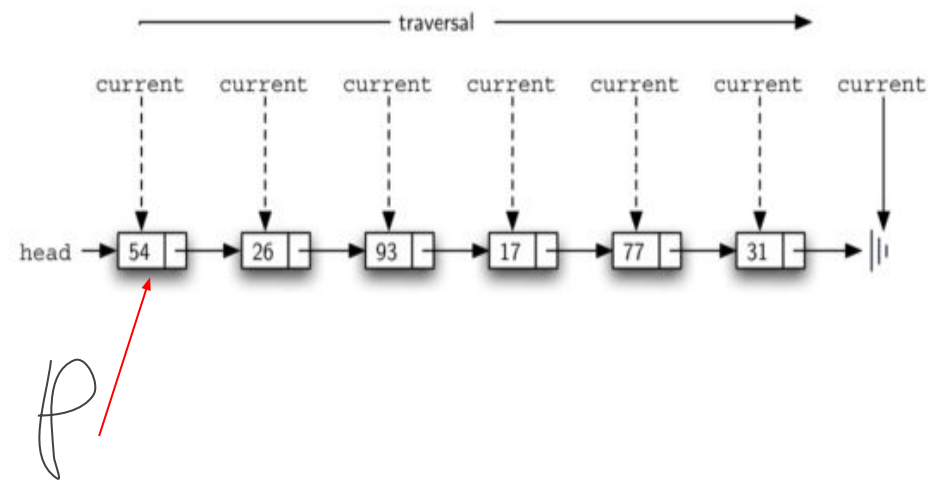
```
def add(self,item):  
    temp = Nodo(item)  
    temp.setNext(self.head)  
    self.head = temp
```



Implementación en Python

— — —

Mostrar una lista?

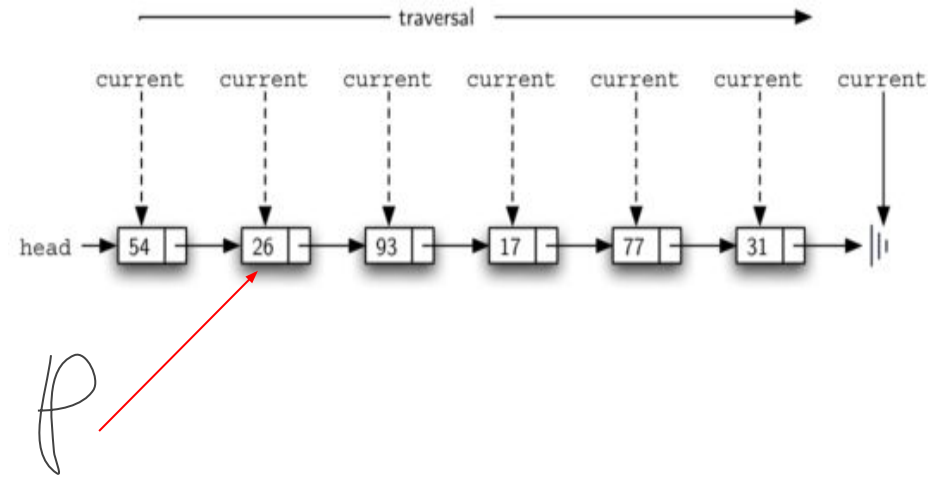


```
def mostrarLista(self):  
    P = self.head  
  
    while (P != None):  
        print(P.getData(), end=' , ')  
        P = P.getNext()
```

Implementación en Python

— — —

Mostrar una lista?

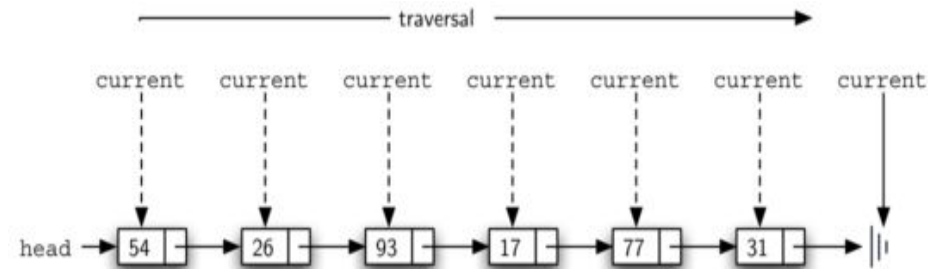


```
def mostrarLista(self):  
    P = self.head  
  
    while (P != None):  
        print(P.getData(), sep=' , ')  
        P = P.getNext()
```

Implementación en Python

— — —

Mostrar una lista?

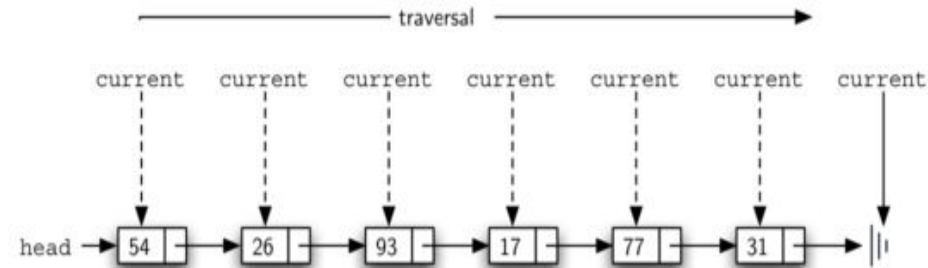


```
def mostrarLista(self):  
    P = self.head  
  
    while (P != None):  
        print(P.getData(), sep=' , ')  
        P = P.getNext()
```

Implementación en Python

— — —

Mostrar una lista?

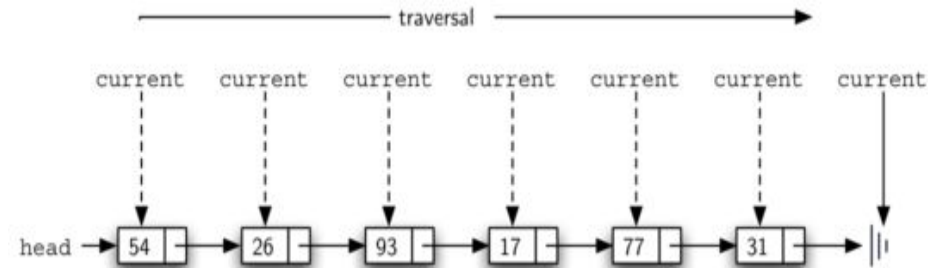


```
def mostrarLista(self):  
    P = self.head  
  
    while (P != None):  
        print(P.getData(), sep=' , ')  
        P = P.getNext()
```

Implementación en Python

— — —

Mostrar una lista?

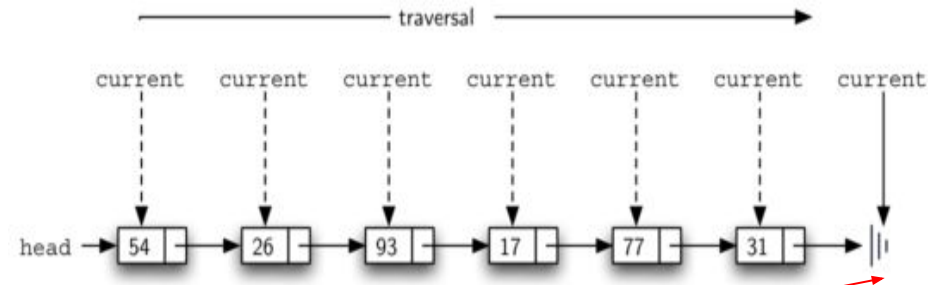


```
def mostrarLista(self):  
    P = self.head  
  
    while (P != None):  
        print(P.getData(), end=' , ')  
        P = P.getNext()
```

Implementación en Python

— — —

Mostrar una lista?



```
def mostrarLista(self):  
    P = self.head  
  
    while (P != None):  
        print(P.getData(), sep=' , ')  
        P = P.getNext()
```


Tarea

— — —

1. Suma los elementos de la lista

2.

3. Cálculo del producto de
elementos de la lista

1. Cuente el número de valores impares de una lista creada
2. Calcule el promedio de la lista de elementos

Tarea

— — —

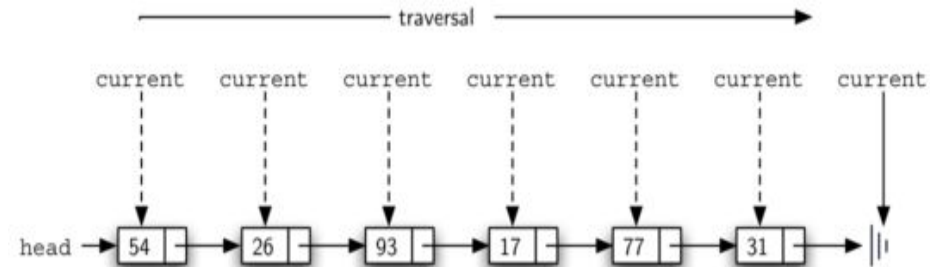
1. Dada una lista de números cree un método donde construya dos listas donde una de ellas almacena los valores impares y la otra los pares de la lista original

2. Dada una lista cree un método que dada una lista de números cree dos sublistas una con los valores primos y otra con el resto.

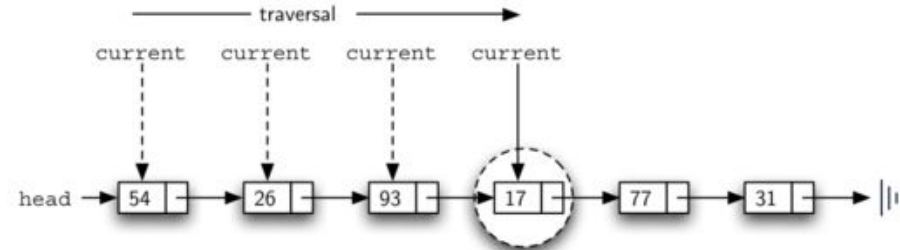
Implementación en Python

— — —

Tamaño de una lista?



¿Búsqueda de un elemento en una lista?

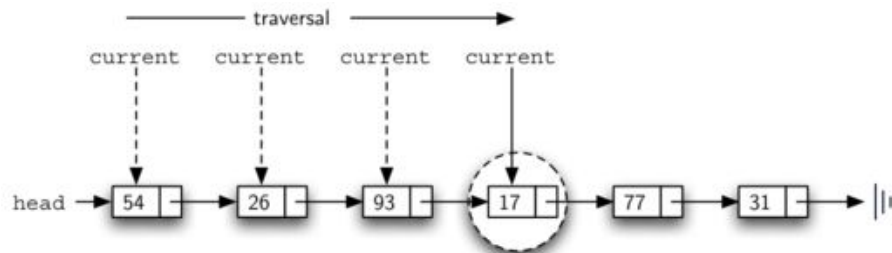


Implementación en Python

— — —

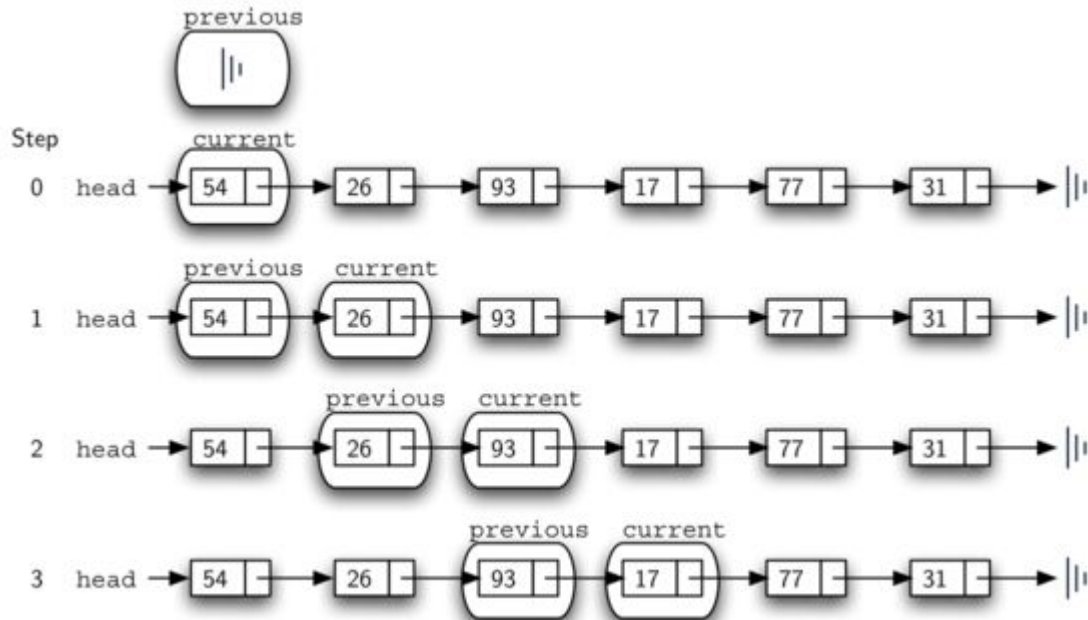
```
def retornar(self, posicion):  
    P = self.head  
    cont = 0  
    while (P != None):  
        if cont == posicion:  
            return P.getData()  
        cont = cont + 1  
        P = P.getNext()  
    return None
```

Retornar elemento acorde a un posición?

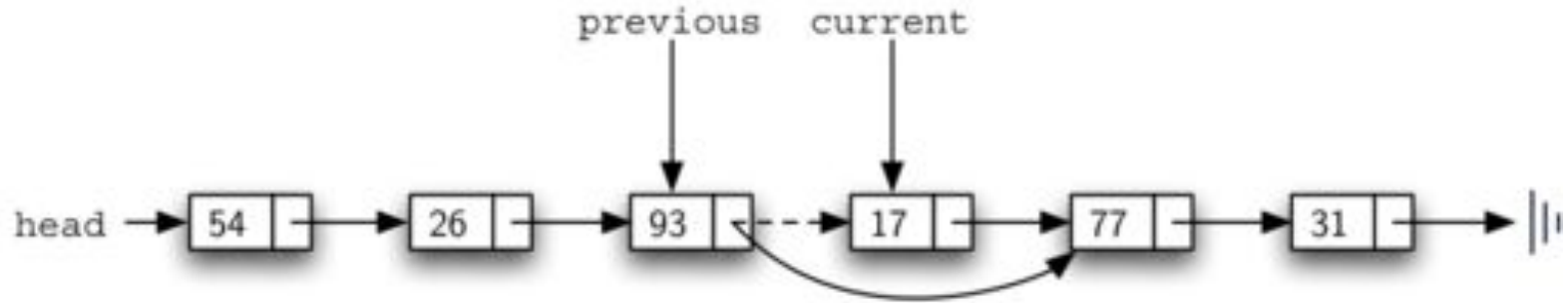


Implementación en Python: eliminar 17

— — —

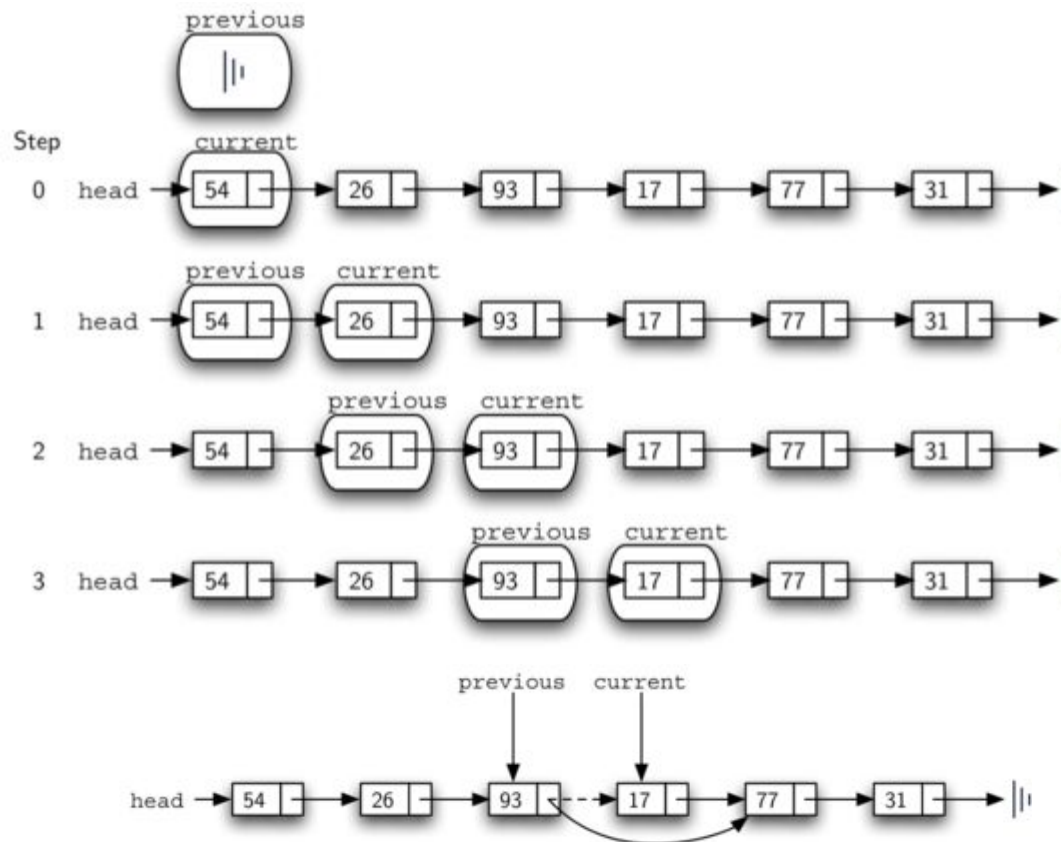


Implementación en Python: eliminar 17

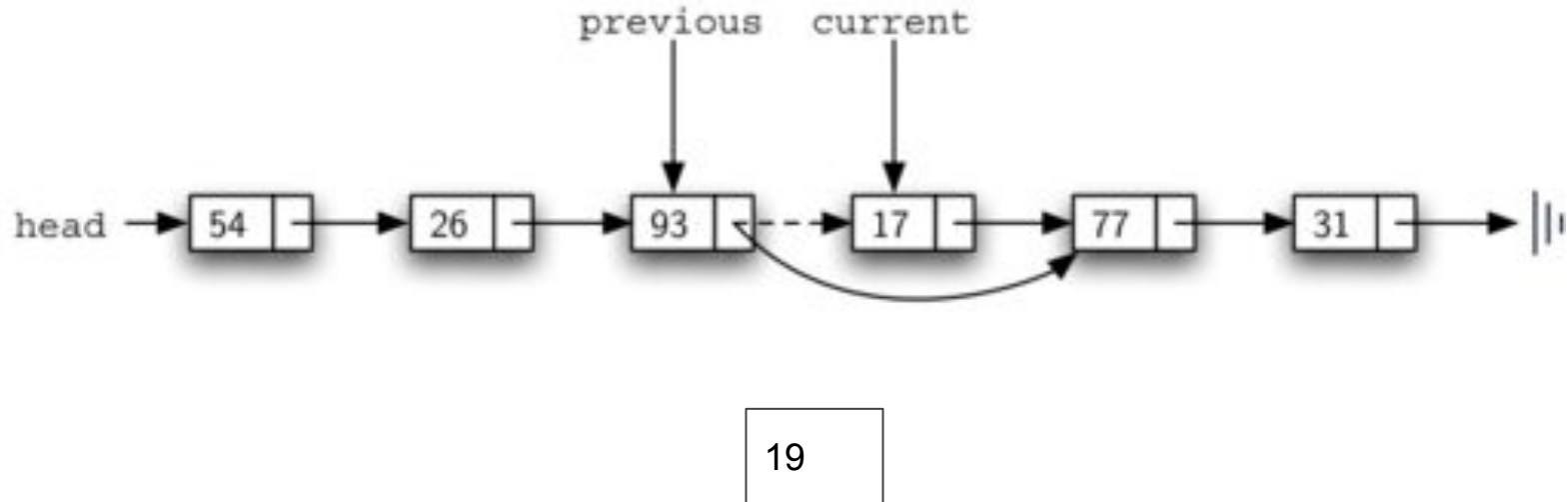


Implementación en Python: eliminar 17

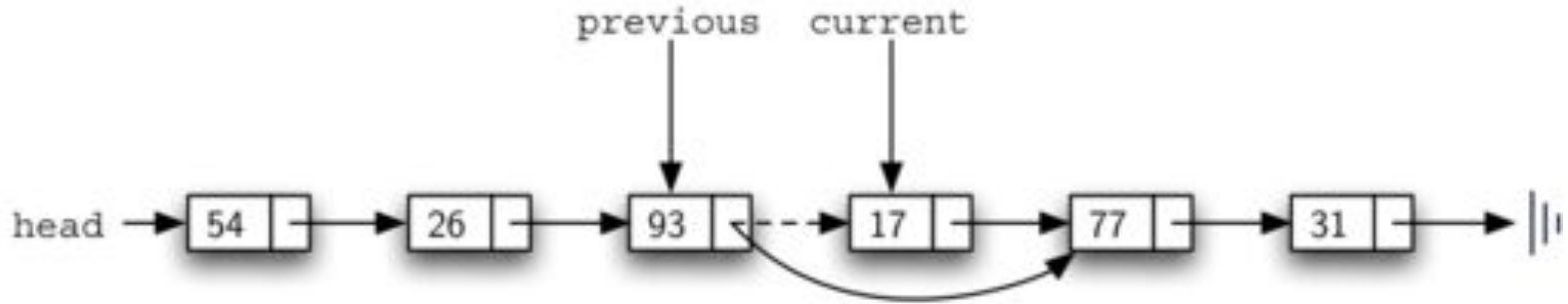
```
def remove(self, item):  
    current = self.head  
    previous = None  
    found = False  
    while current != None:  
        if current.getData() == item:  
            found = True  
            break  
        else:  
            previous = current  
            current = current.getNext()  
    if(found == True):  
        if previous == None:  
            self.head = current.getNext()  
        else:  
            previous.setNext(current.getNext())
```



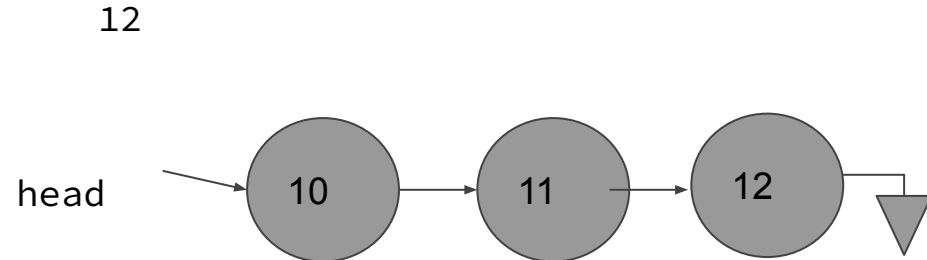
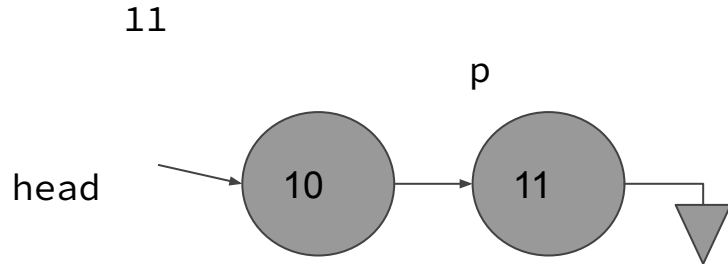
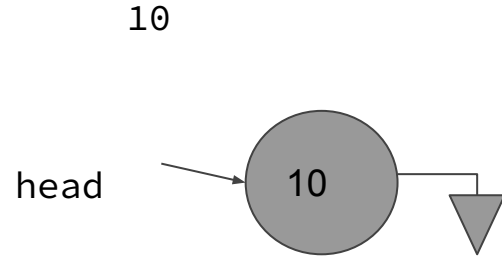
Implementación en Python: insertar 19 antes de 17



Implementación en Python: insertar 19 después de 17

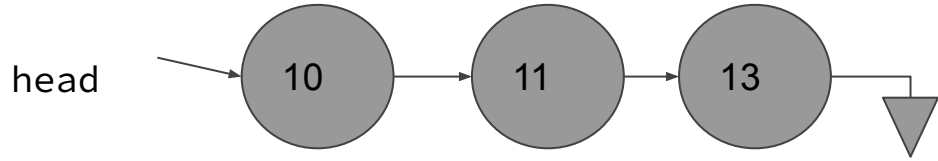


Implementación en Python: insertar al final de la lista

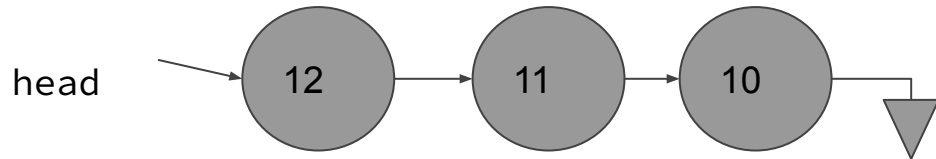


Implementación en Python: insertar al final de la lista

12 11 10



12 11 10



Árboles

- Los árboles representan las estructuras no-lineales y dinámicas más importantes en computación.
- La estructura de un árbol puede cambiar durante la ejecución de un programa.
- No lineal, puesto que a cada elemento del árbol pueden seguirle varios elementos



It's just a tree, son...

But is a binary tree...

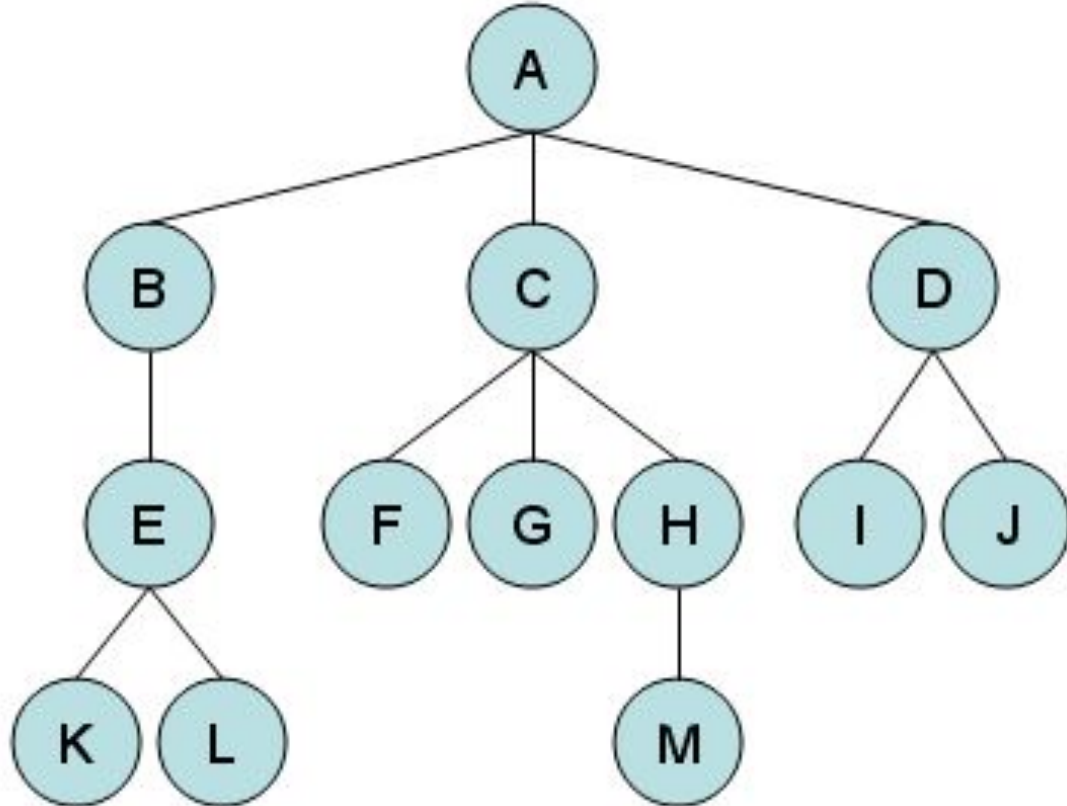


Árboles binarios de búsqueda

Árboles

Propiedades

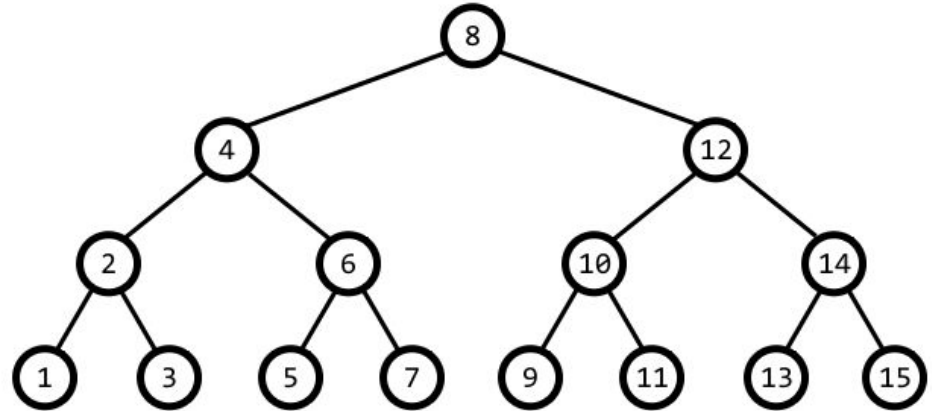
- Todo nodo que no es raíz, ni terminal u hoja se llama **nodo interior**.
- **Grado** es el número de descendientes de un nodo.
- **Grado del árbol** es el máximo grado de todos los nodos del árbol
- **Nivel** es el número de arcos que son recorridos para llegar a un nodo. La raíz tiene nivel 1
- **Altura** del árbol es el máximo número de niveles de todos los nodos del árbol



Árboles binarios de búsqueda (ABB)

— — —

- Un árbol binario es una estructura tipo árbol que tiene una raíz en donde cada vértice tiene no más que dos hijos.
- Cada hijo de un vértice es llamado hijo derecho o izquierdo
- Un árbol binario **completo** por cada nodo interior tienen sus hijos izquierdo y derecho. Además, el número de nodos = $2^h - 1$
-



Implementación de un nodo de ABB

— — —

- Un nodo debe contener el valor a guardar.
- Los hijos izquierdo y derecho de un nodo al inicio deben ser nulos.



class Node:

```
def __init__(self, data):  
    self.left = None  
    self.right = None  
    self.data = data
```

```
def PrintTree(self):  
    print(self.data)
```

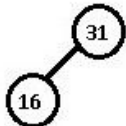
```
root = Node(10)  
root.PrintTree()
```


Insertar un nuevo nodo ABB

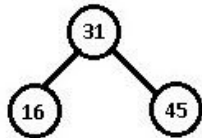
— — —



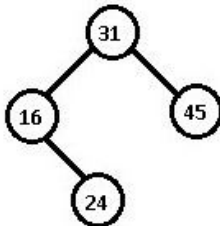
Insert 31



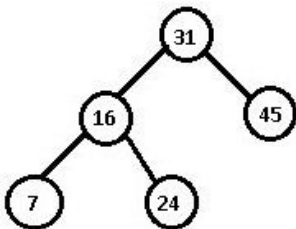
Insert 16



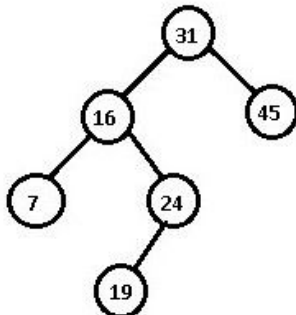
Insert 45



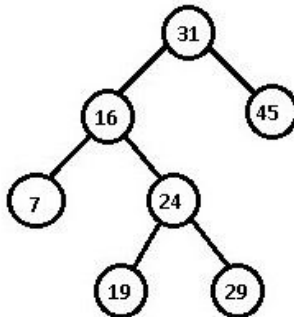
Insert 24



Insert 7



Insert 19



Insert 29

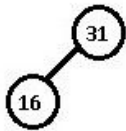


Insertar un nuevo nodo ABB

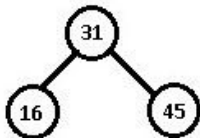
— — —



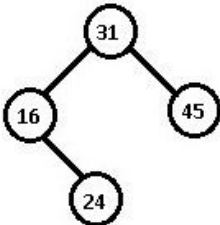
Insert 31



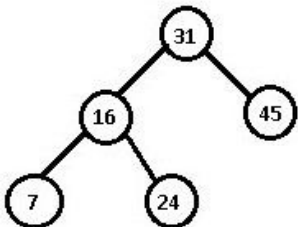
Insert 16



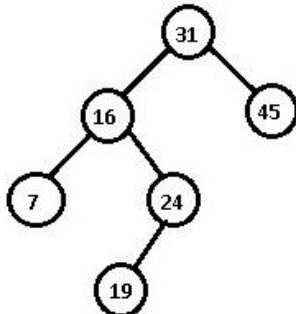
Insert 45



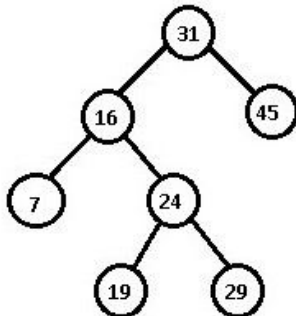
Insert 24



Insert 7



Insert 19

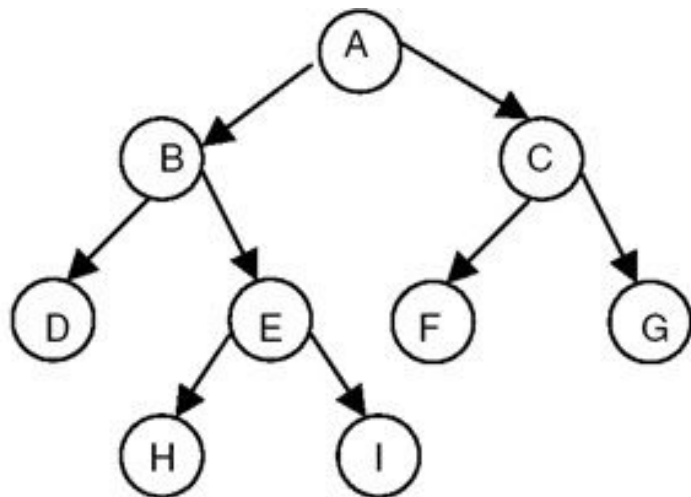


Insert 29

```
def insert(self, data):  
    # Compara el nuevo valor con el nodo padre  
    if self.data:  
        if data < self.data:  
            if self.left is None:  
                self.left = Node(data)  
            else:  
                self.left.insert(data)  
        elif data > self.data:  
            if self.right is None:  
                self.right = Node(data)  
            else:  
                self.right.insert(data)  
    else:  
        self.data = data
```

Recorrido de un ABB

— — —



Inorder : DBHEIAFCG

Preorder : ABDEHICFG

Postorder : DHIEBFGCA

```
def PreOrder(self):  
    print( self.data),  
    if self.left:  
        self.left.PreOrder()  
    if self.right:  
        self.right.PreOrder()
```

[10, -5, 5, 12, 8, 0] Alvarado

— — —



In Order: -5 5 0 8 10 12
Pre Order: 10 -5 5 0 8 12
Post Order: 5 0 8 -5 12 10

= [13, 11, 17, 32, 21, 10] Chavez

— — —

InOrder: 10,11,13,21,32,17

PreOrder: 13,11,10,17,32,21

PostOrder: 10,11,21,32,17,13



= [23, 1, 7, 12, 31, 30] Fernandez

Postorder: 1 12 7 30 31 23

Inorder: 1 7 12 23 30 31

Preorder: 23 1 7 12 31 30



= [9, 1, 17, 2, 13, 30] Mendoza

Inorder: 1,2,9,13,17,30

Preorder: 9,1,2,17,13,30

Postorder: 2,1,13,30,17,9



= [25, 21, 37, 32, 13, 30]:Vara

— — —

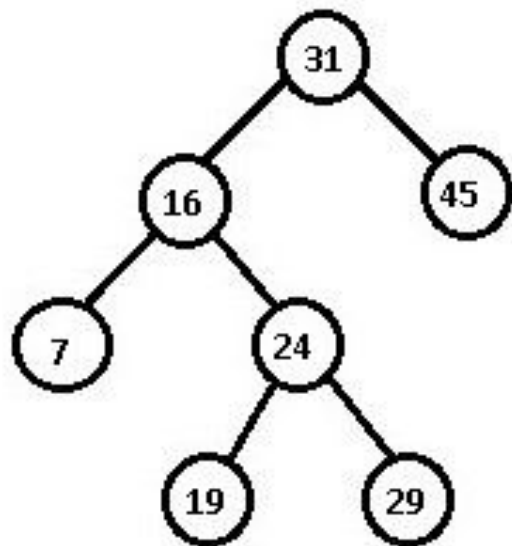
inorder: 13,21,25,30,32,37

preorder: 25,21,13,37,32,30

postorder: 13,21,30,32,37,25



Ejemplo recorrido



Insert 29

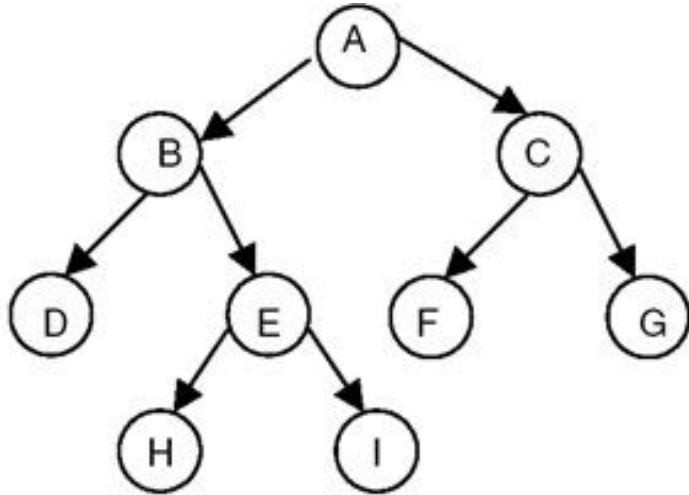
Postorder: 7 19 29 24 16 45 31

Inorder: 7 16 19 24 29 31 45

Preorder: 31 16 7 24 19 29 45

Recorrido de un ABB

— — —



Inorder : DBHEIAFCG

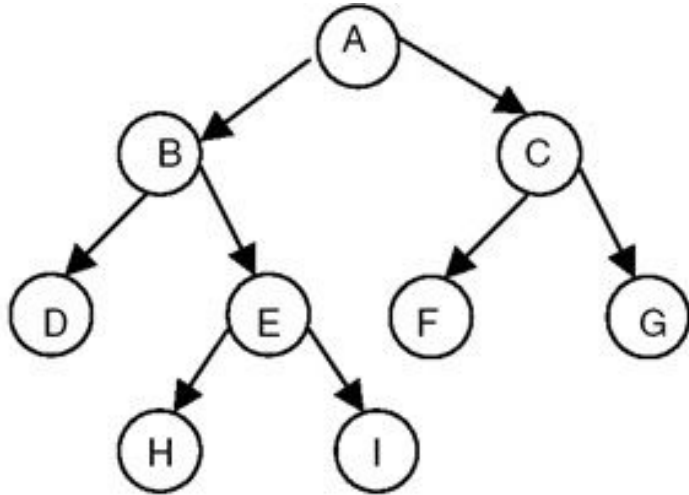
Preorder : ABDEHICFG

Postorder : DHIEBFGCA

```
def PrintTree(self):  
    if self.left:  
        self.left.PrintTree()  
    if self.right:  
        self.right.PrintTree()  
  
    print( self.data),
```

Recorrido de un ABB

— — —



Inorder : DBHEIAFCG

Preorder : ABDEHICFG

Postorder : DHIEBFGCA

```
def PrintTree(self):
```

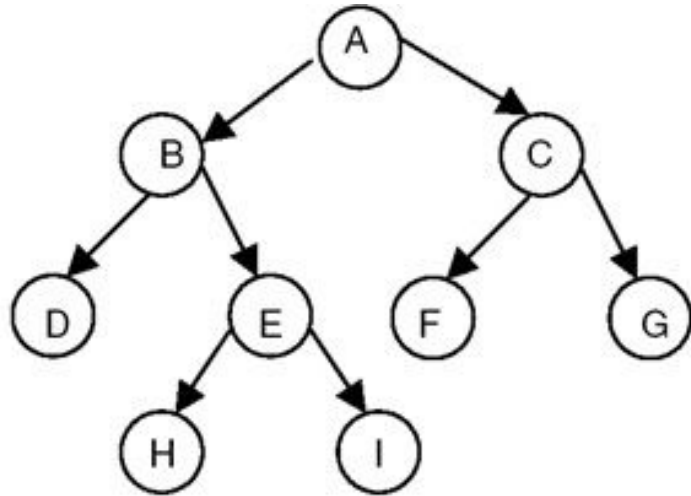
```
    if self.left:
        self.left.PrintTree()
```

```
    print( self.data),
```

```
    if self.right:
        self.right.PrintTree()
```

Búsqueda de un elemento X en un ABB

— — —



Inorder : DBHEIAFCG

Preorder : ABDEHICFG

Postorder : DHIEBFGCA

Ejercicios

— — —

1. La suma de todos los elementos de un árbol binario
2. Contar todo los elementos de un árbol binario
3. Retornar cuantos elementos del árbol binario son impares
4. La suma de la última cifra de cada elemento de un árbol binario

