

HENRY

A bright yellow beam of light originates from the left edge of the frame and points towards the letter 'R' in the word 'HENRY'. The beam is wider on the left and tapers as it moves to the right. The word 'HENRY' is written in a bold, black, sans-serif font.



Recursion




Para aprender la recursión, primero hay que aprender la
recursión.



```
1 function factorial(x)
2 {
3     if (x > -1 && x < 2) return 1; // Cuando -1 < x < 2
4     // devolvemos 1 puesto que 0! = 1 y 1! = 1
5     else if (x < 0) return 0;      // Error no existe factorial de números negativos
6     return x * factorial(x - 1);  // Si x >= 2 devolvemos el producto de x por el factorial de x - 1
7 }
```

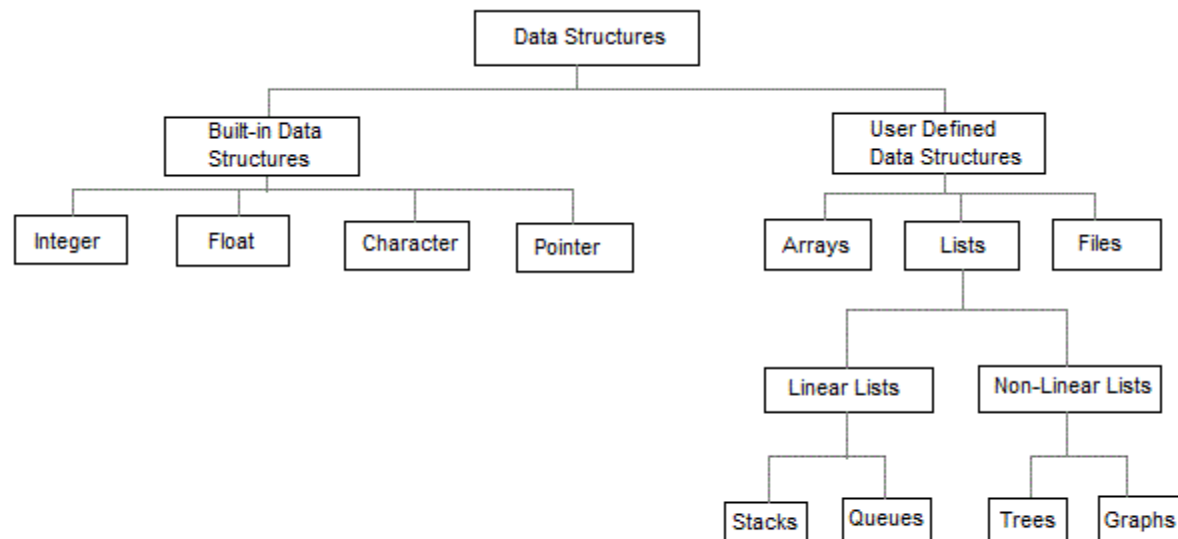


< Demo />



Estructuras de Datos - Parte I

Cuando hablamos a estructura de Datos nos referimos a cómo organizamos los datos cuando programamos. Básicamente, este tema trata de encontrar formas particulares de organizar datos de tal manera que puedan ser utilizados de manera eficiente.



INTRODUCTION TO DATA STRUCTURES

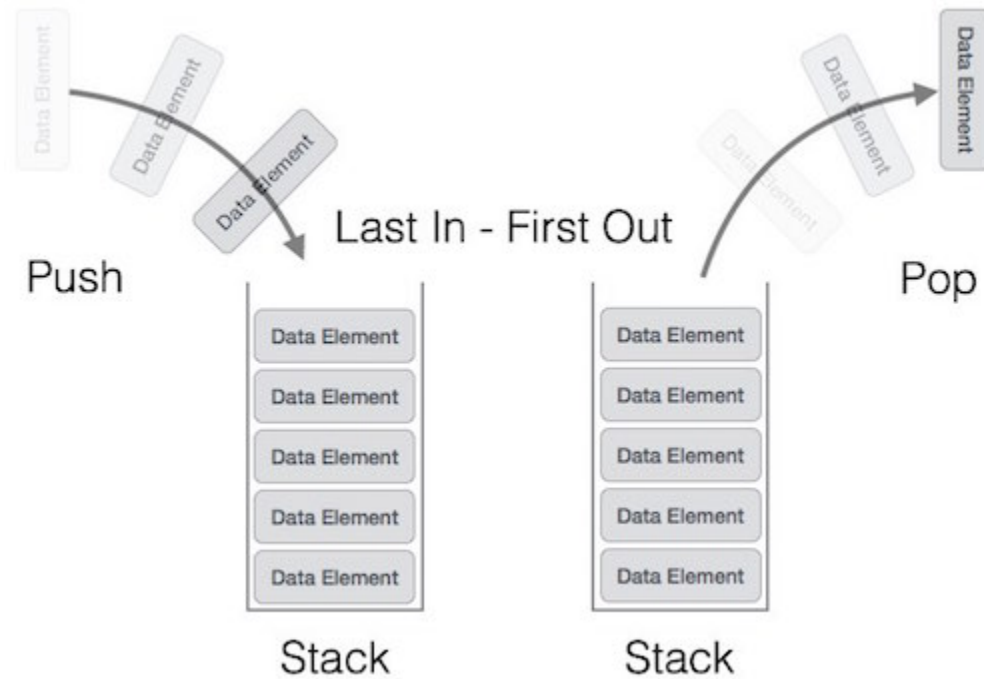
Arreglos

Index	0	1	2	3	4
	H	e	l	l	o
Address	0x23451	0x23452	0x23453	0x23454	0x23455

Sets

```
1 var arreglo = [1,2,3,4,4,5,5,1,2]
2 var set1     = new Set(arreglo)
3 console.log(arreglo)    // [ 1, 2, 3, 4, 4, 5, 5, 1, 2 ]
4 console.log(set1)       // Set { 1, 2, 3, 4, 5 }
```

Pilas (Stacks)






Pilas (Stacks)

```
1 var stack = [];  
2 stack.push(1);      // la pila es [1]  
3 stack.push(10);     // la pila es ahora [1, 10]  
4 var i = stack.pop(); // la pila [1]  
5 console.log(i);     // muestra 10
```

Colas (Queue)



```
1 var queue = [];  
2 queue.push(1);           // la cola es [1]  
3 queue.push(2);           // la cola es [1, 2]  
4 var i = queue.shift();    // la cola es [2]  
5 console.log(i);           // muestra 1
```



Estructuras de Datos - Parte II

Listas Enlazadas



```
1 function Node(data) {  
2     this.data = data;  
3     this.next = null;  
4 }  
5  
6 function List() {  
7     this._length = 0;  
8     this.head = null;  
9 }
```



Listas Enlazadas

- *Iterar sobre la lista:* Recorrer la lista viendo sus elementos o hasta que encontremos el elemento deseado.
- *Insertar un nodo:* La operación va a cambiar según el lugar donde querramos insertar el nodo nuevo:
 - Al principio de la lista.
 - En el medio de la lista.
 - Al final de la lista.
- *Sacar un nodo:*
 - Del principio de la lista.
 - Del medio de la lista.

Listas Enlazadas

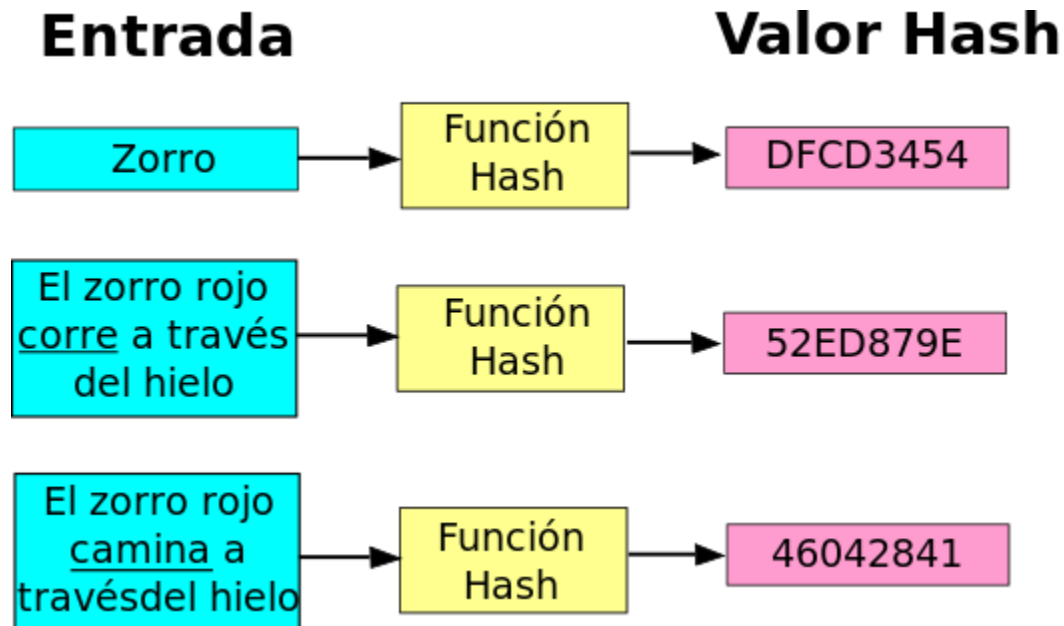
```
1 List.prototype.add = function(data) {
2     var node = new Node(data),
3     current = this.head;
4     // Si está vacia
5     if (!current) {
6         this.head = node;
7         this._length++;
8         return node;
9     }
10    // Si no esta vacia, recorro hasta encontrar el último
11    while (current.next) {
12        current = current.next;
13    }
14    current.next = node;
15    this._length++;
16    return node;
17 };
18
19 List.prototype.getAll = function(){
20     current = this.head //empezamos en la cabeza
21     if(!current){
22         console.log('La lista esta vacia!')
23         return
24     }
25     while(current){
26         console.log(current.data);
27         current = current.next;
28     }
29     return
30 };
```



Listas Doblemente Enlazadas



En la lista que vimos antes, sólo podemos recorrer la lista en un solo sentido. En algunos casos nos puede servir recorrer la lista en los dos sentidos, para tales casos lo que vamos a usar es una lista doblemente enlazada

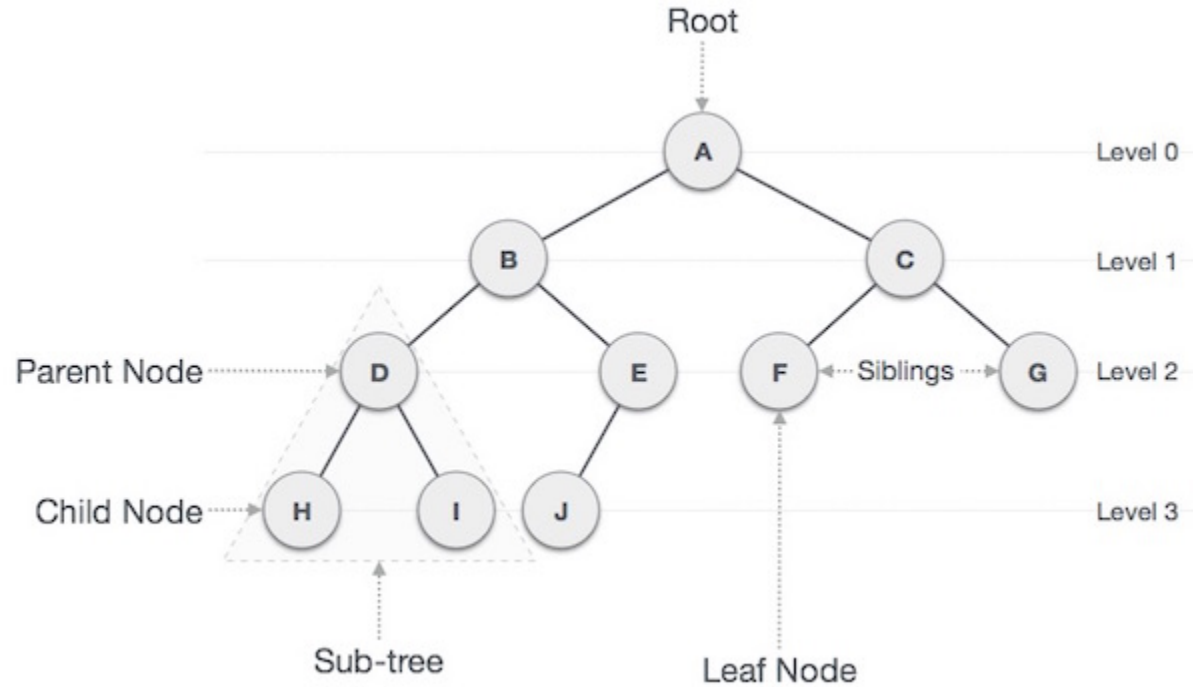
Hash Table



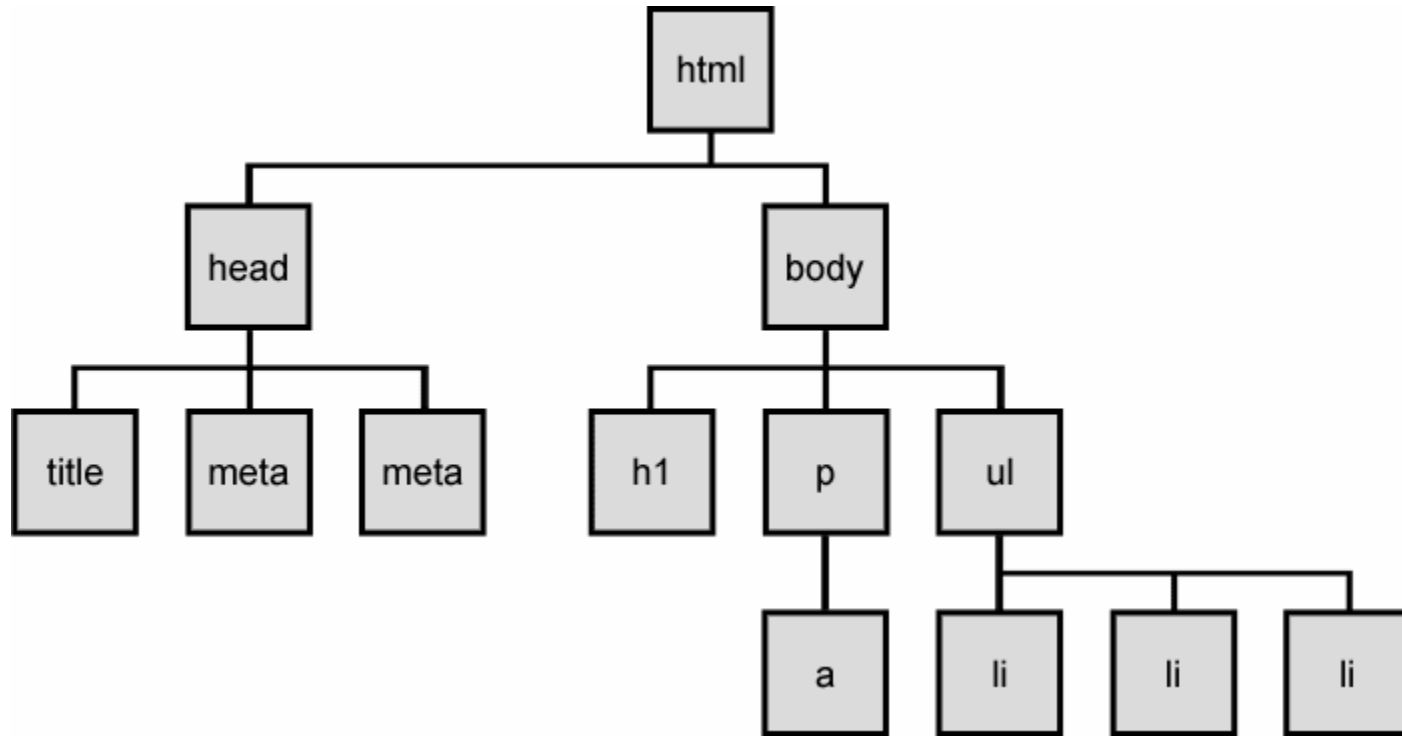


Estructuras de Datos - Parte III

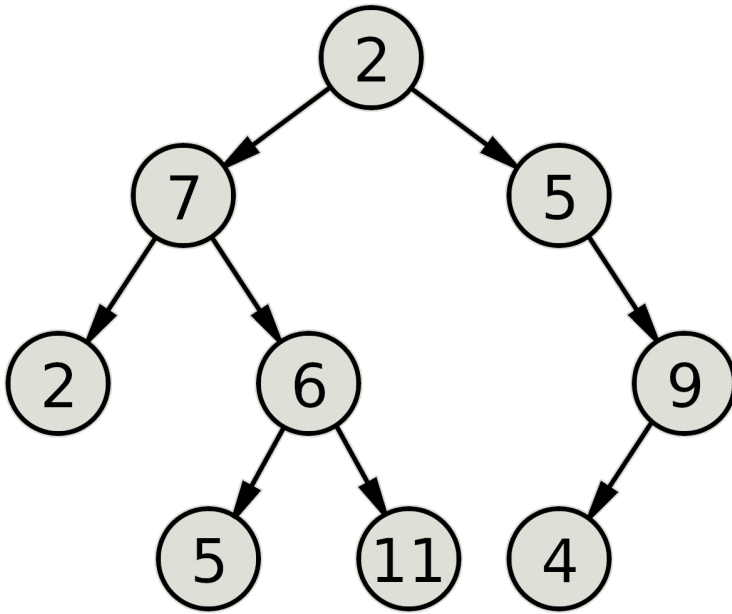
Árboles (trees)



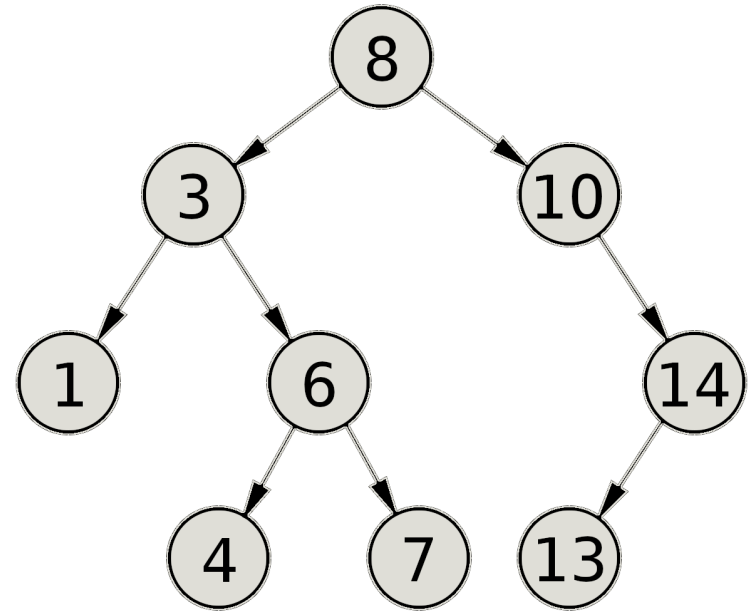
DOM



Árboles Binarios



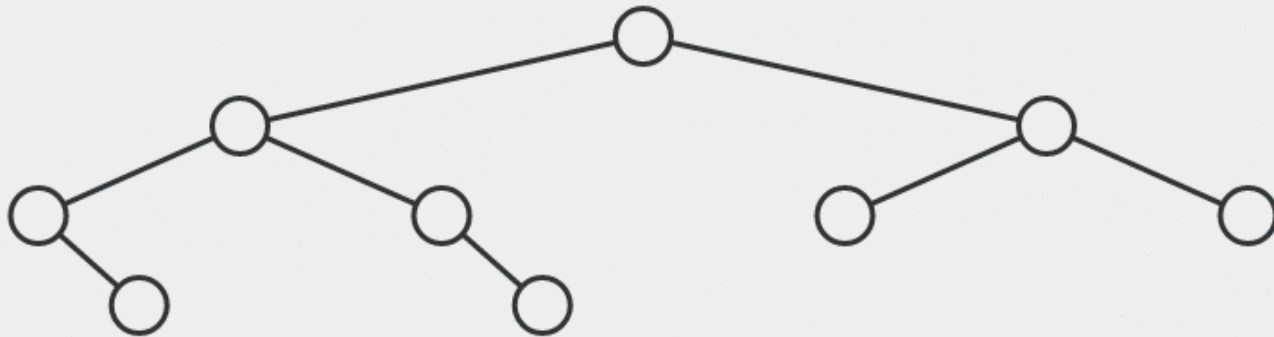
Binary Tree



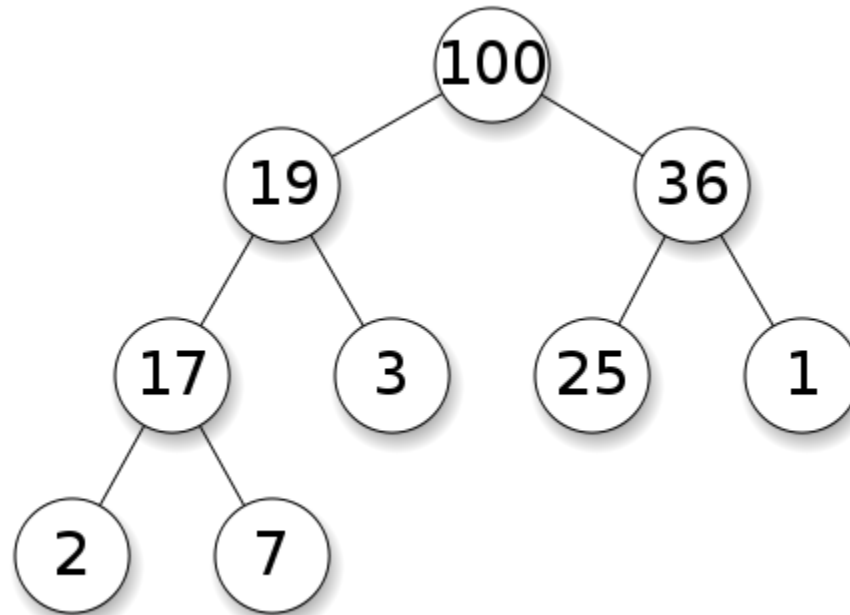
Binary Search Tree

Árbol Binario Autobalanceado

AVL Tree



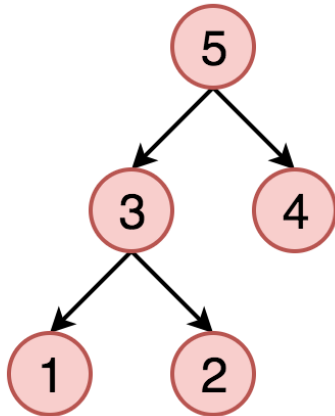
Max Heap



Recorridos

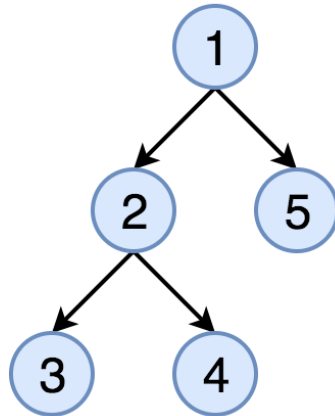
DFS Postorder

Bottom -> Top
Left -> Right



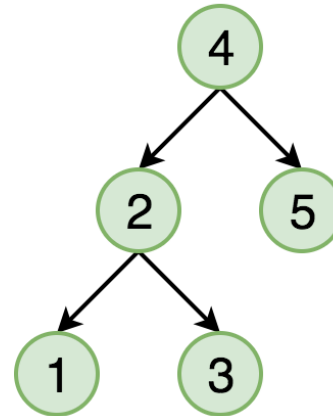
DFS Preorder

Top -> Bottom
Left -> Right



DFS Inorder

Left -> Node -> Right



BFS

Left -> Right
Top -> Bottom

