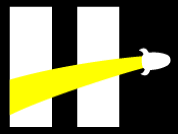
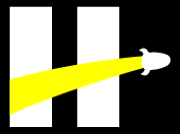
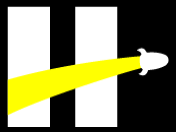

HENRY



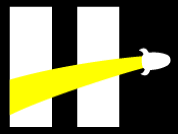
Recursion



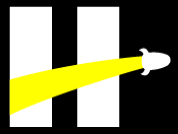
Para aprender la recursión, primero hay que aprender la recursión.



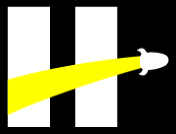
```
function factorial(x)
 {
    if (x > -1 && x < 2) return 1;  // Cuando -1 < x < 2
        // devolvemos 1 puesto que 0! = 1 y 1! = 1
    else if (x < 0) return 0;       // Error no existe factorial de números negativos
    return x * factorial(x - 1);    // Si x >= 2 devolvemos el producto de x por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



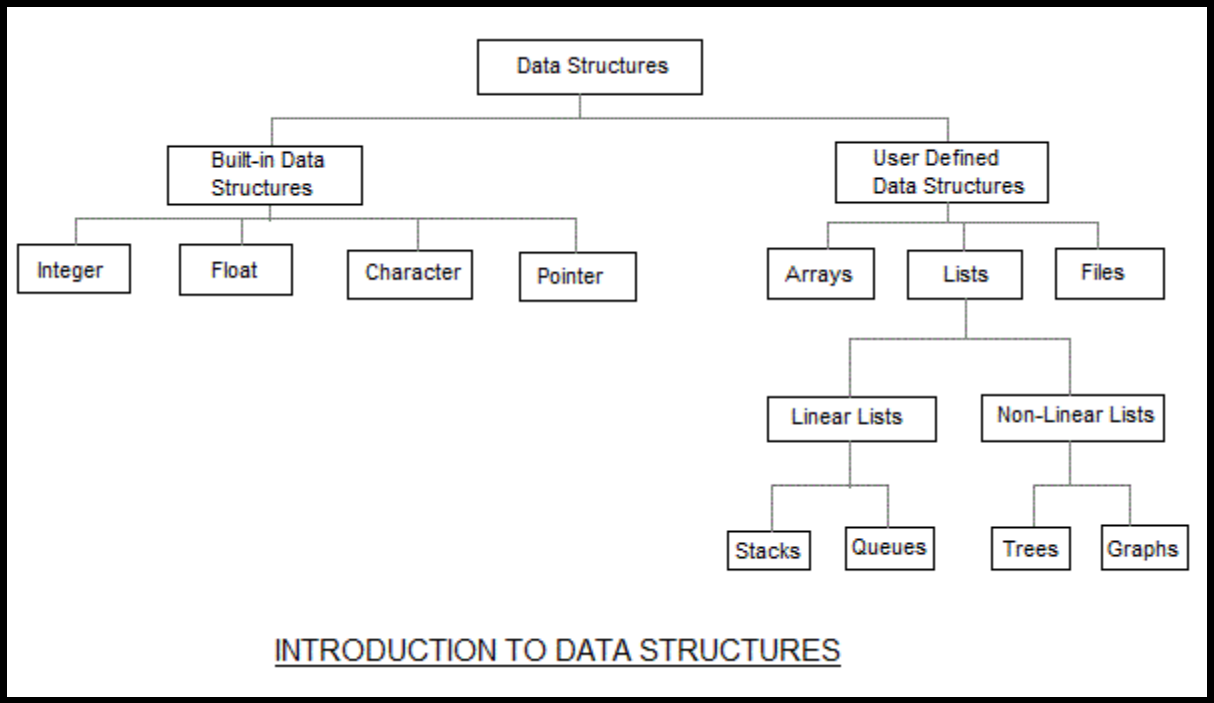
< Demo />

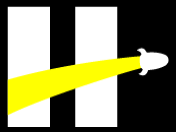


Estructuras de Datos - Parte I

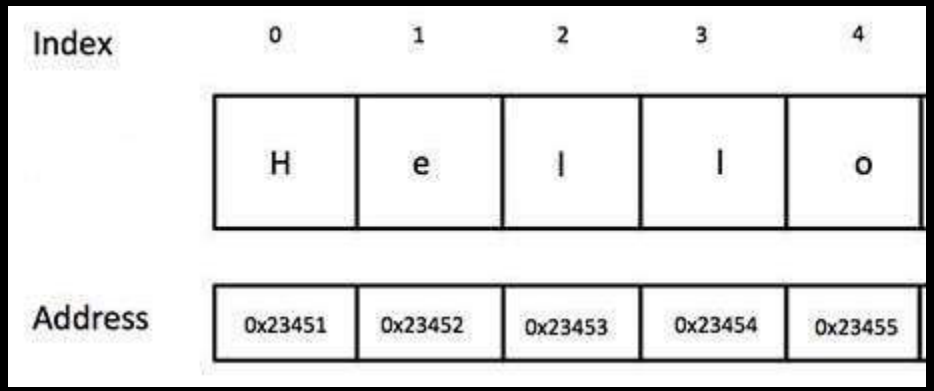


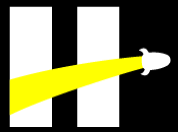
Cuando hablamos a estructura de Datos nos referimos a cómo organizamos los datos cuando programamos. Básicamente, este tema trata de encontrar formar particulares de organizar datos de tal manera que puedan ser utilizados de manera eficiente.





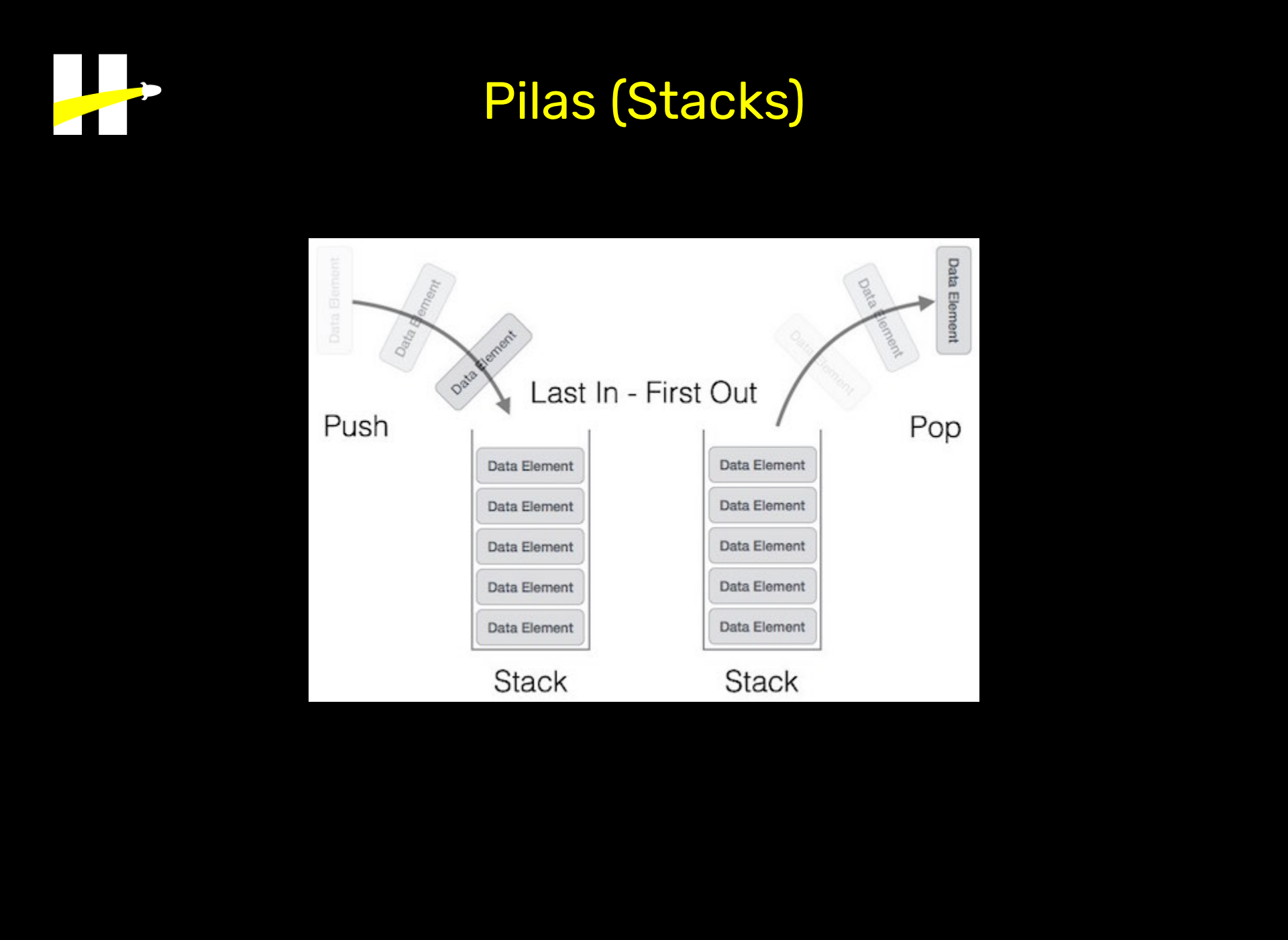
Arreglos


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

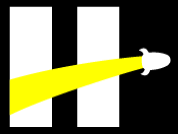


Sets

```
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 }
```

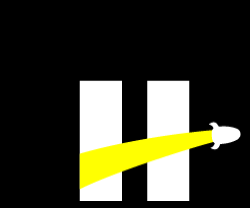


Pilas (Stacks)

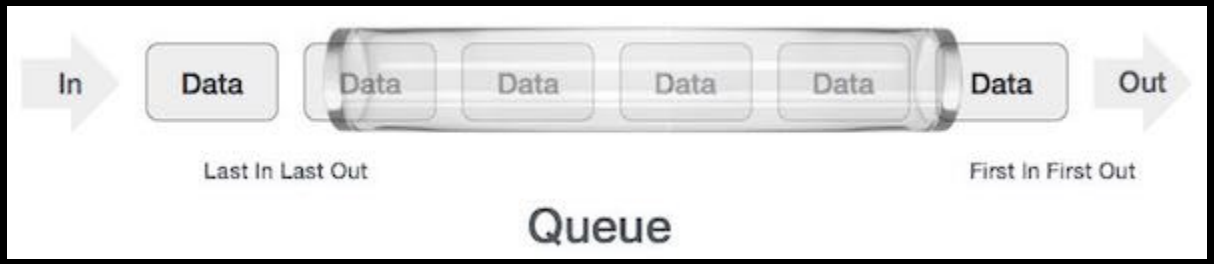


Pilas (Stacks)

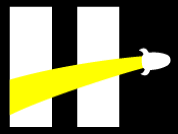
```
1 var stack = [];
2 stack.push(1);        // la pila es [1]
3 stack.push(10);       // la pila es ahora [1, 10]
4 var i = stack.pop(); // la pila [1]
5 console.log(i);       // muestra 10
```



Colas (Queue)



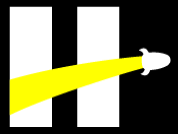

```
1 var queue = [];
2 queue.push(1);          // la cola es [1]
3 queue.push(2);          // la cola es [1, 2]
4 var i = queue.shift(); // la cola es [2]
5 console.log(i);         // muestra 1
```



Estructuras de Datos - Parte II

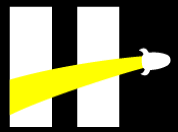
Listas Enlazadas

```
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 econtremos 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.

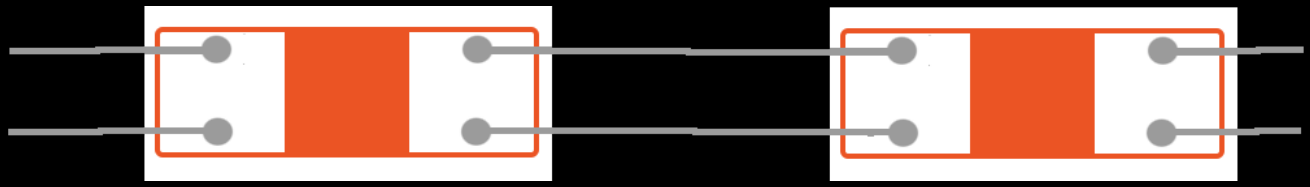


Listas Enlazadas

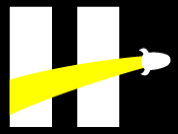
```
List.prototype.add = function(data) {
    var node = new Node(data),
    current = this.head;
    // Si está vacia
    if (!current) {
        this.head = node;
        this._length++;
        return node;
    }
    // Si no esta vacia, recorro hasta encontrar el último
    while (current.next) {
        current = current.next;
    }
    current.next = node;
    this._length++;
    return node;
};

List.prototype.getAll = function(){
        current = this.head //empezamos en la cabeza
        if(!current){
                console.log('La lista esta vacia!')
                return
        }
        while(current){
                console.log(current.data);
                current = current.next;
        }
        return
};
```

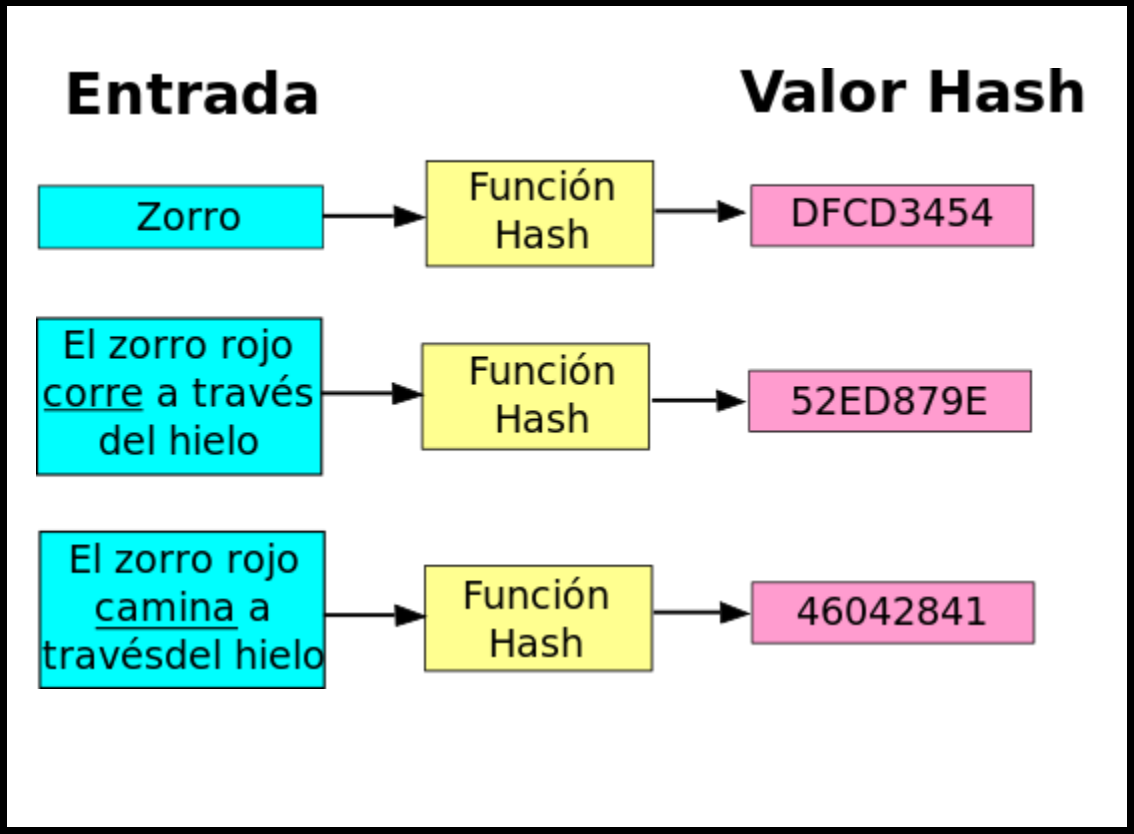
Listas Doblemente Enlazadas

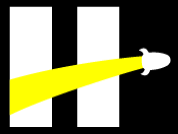


En la lista que vimos antes, sólo podemos recorrer la lista en un solo sentido. En algunos casos nos puede servir recorrer la lista en los dos sentidos, para tales casos lo que vamos a usar es una lista doblemente enlazada

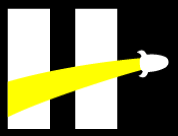


Hash Table

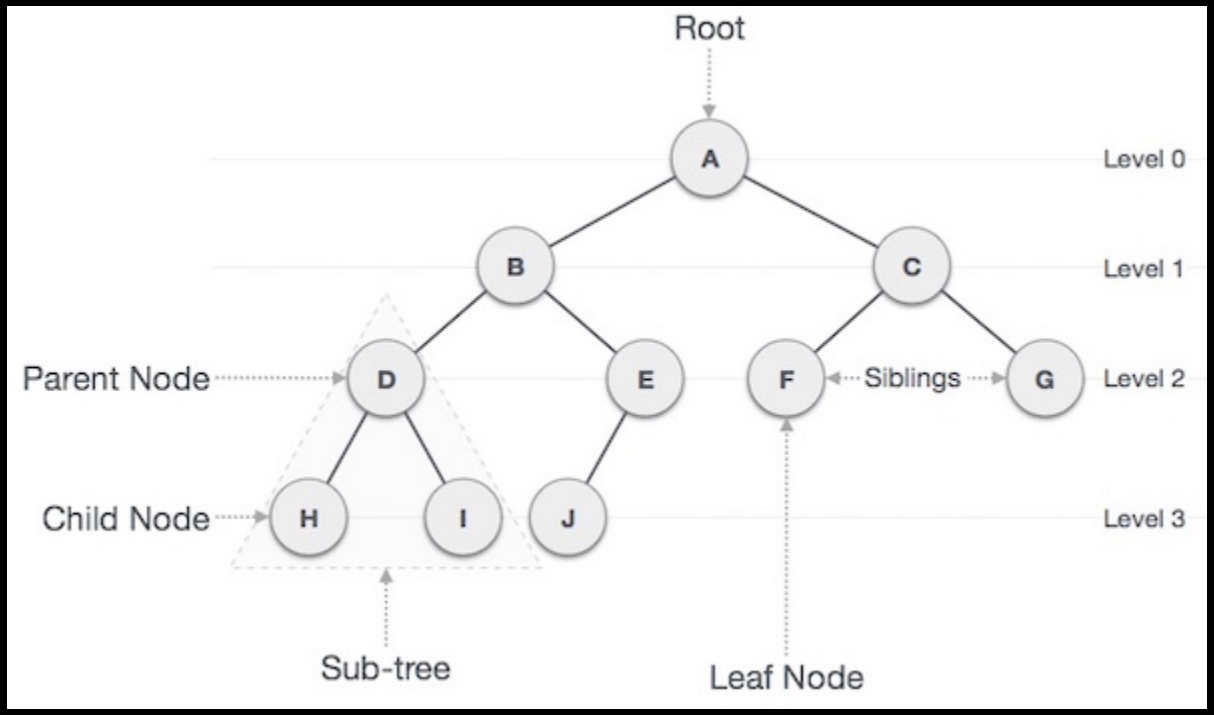

Entrada
Valor Hash
Zorro
Función Hash
DFCD3454
El zorro rojo corre a través del hielo
Función Hash
52ED879E
El zorro rojo camina a travésdel hielo
Función Hash
46042841

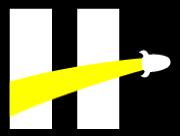


Estructuras de Datos - Parte III

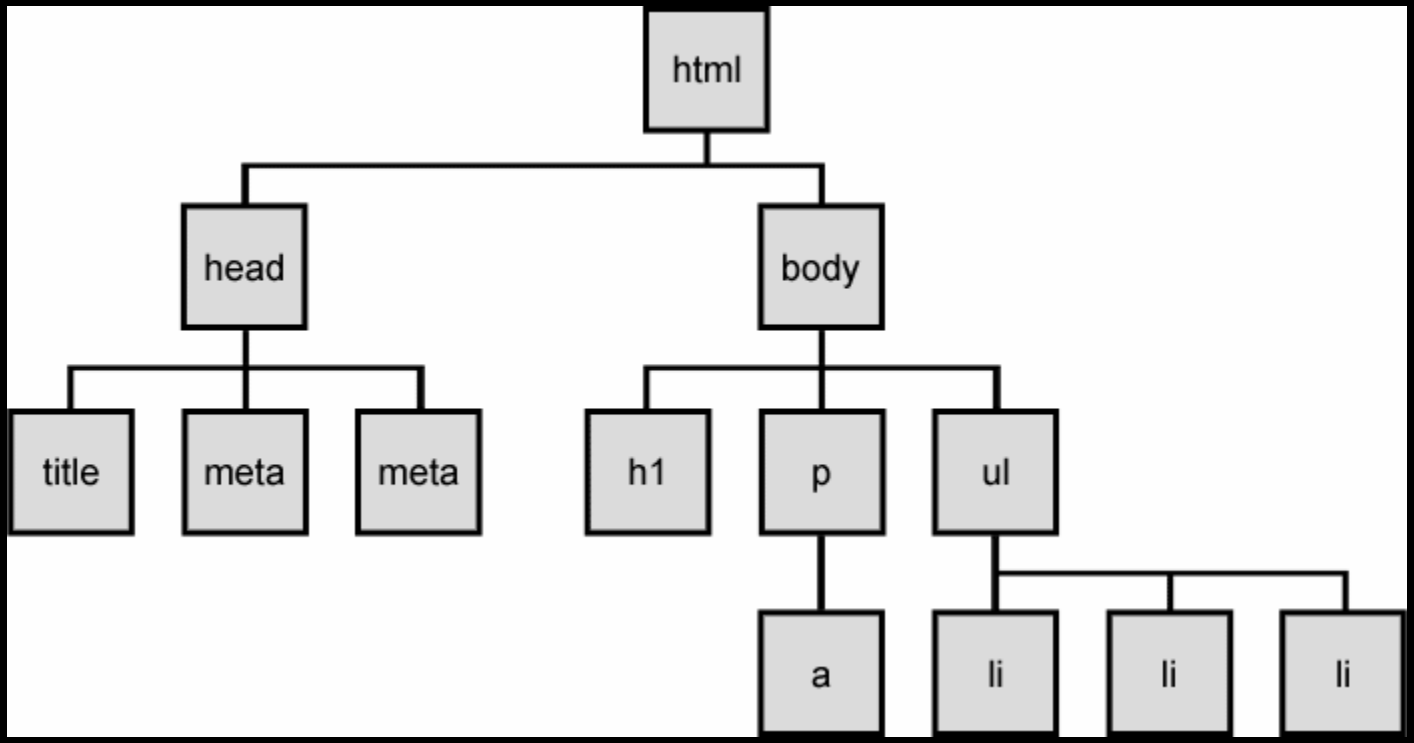


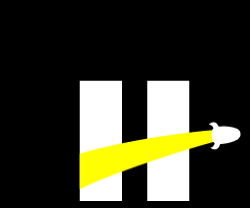
Árboles (trees)



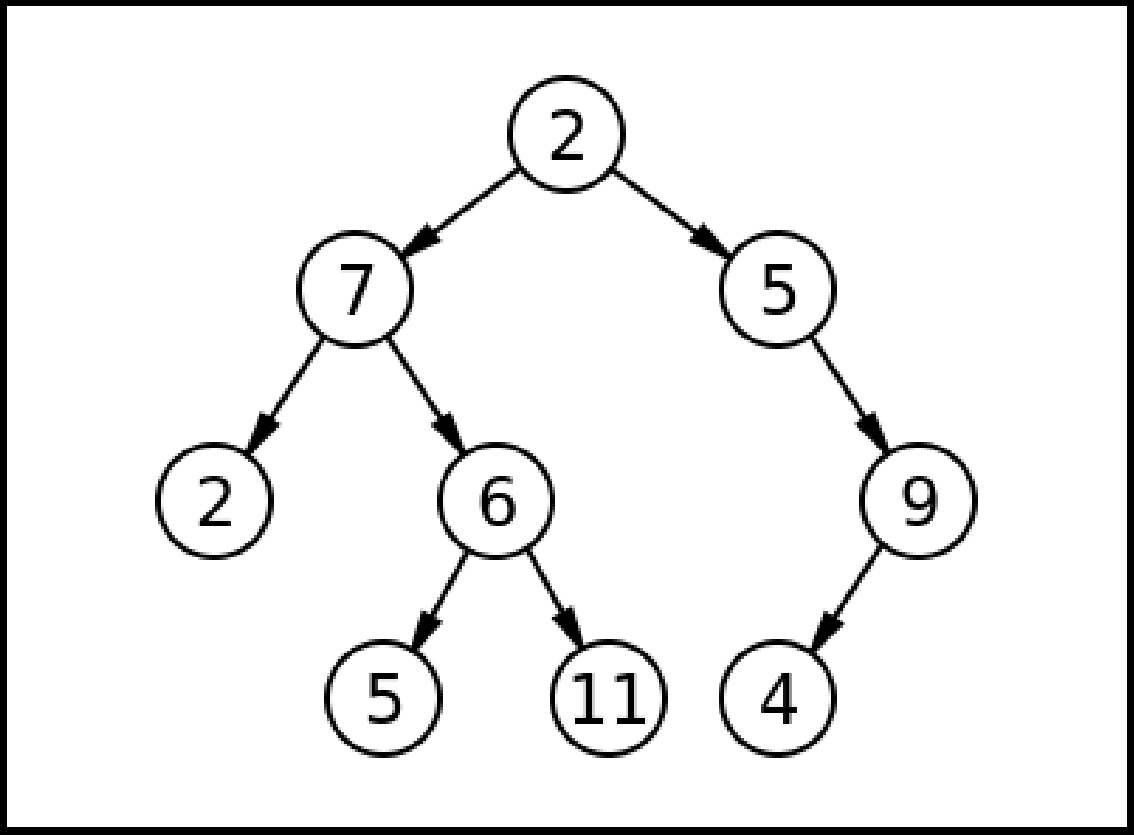


Árboles (trees)

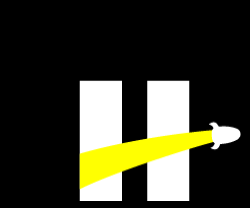




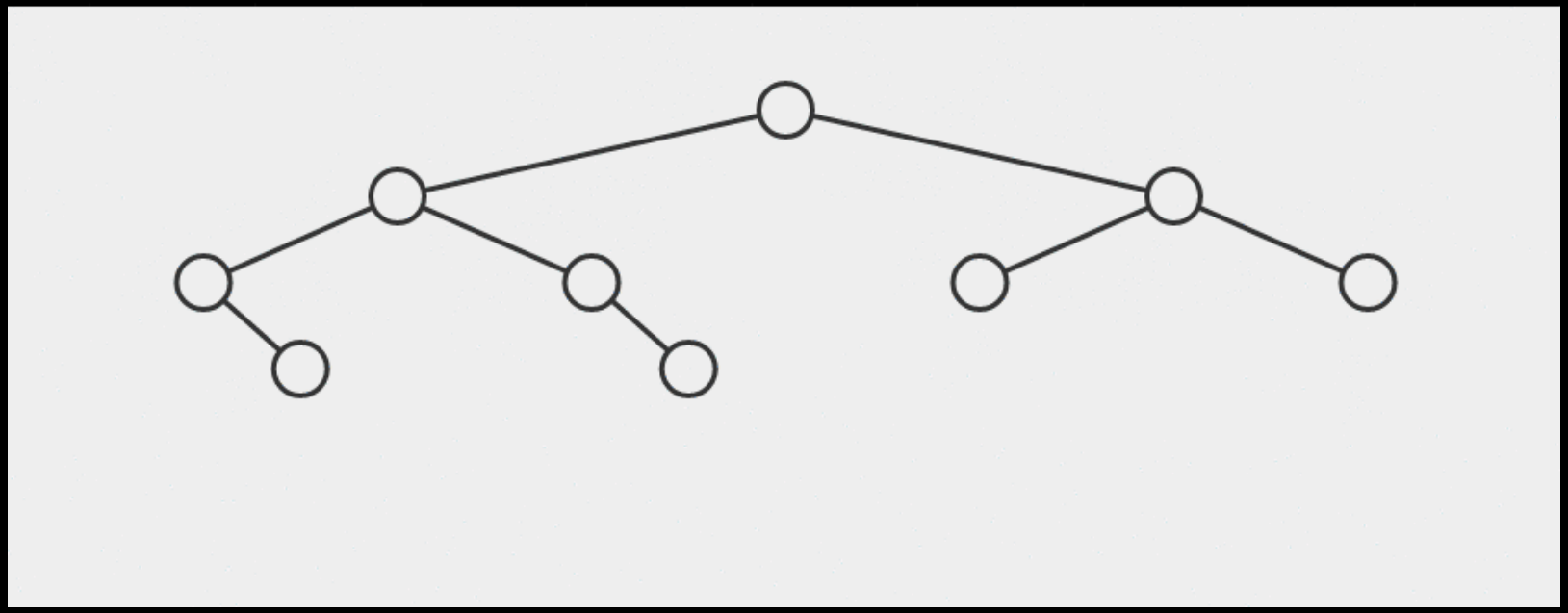
Tipos de árboles



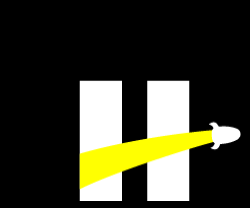
Binary Tree



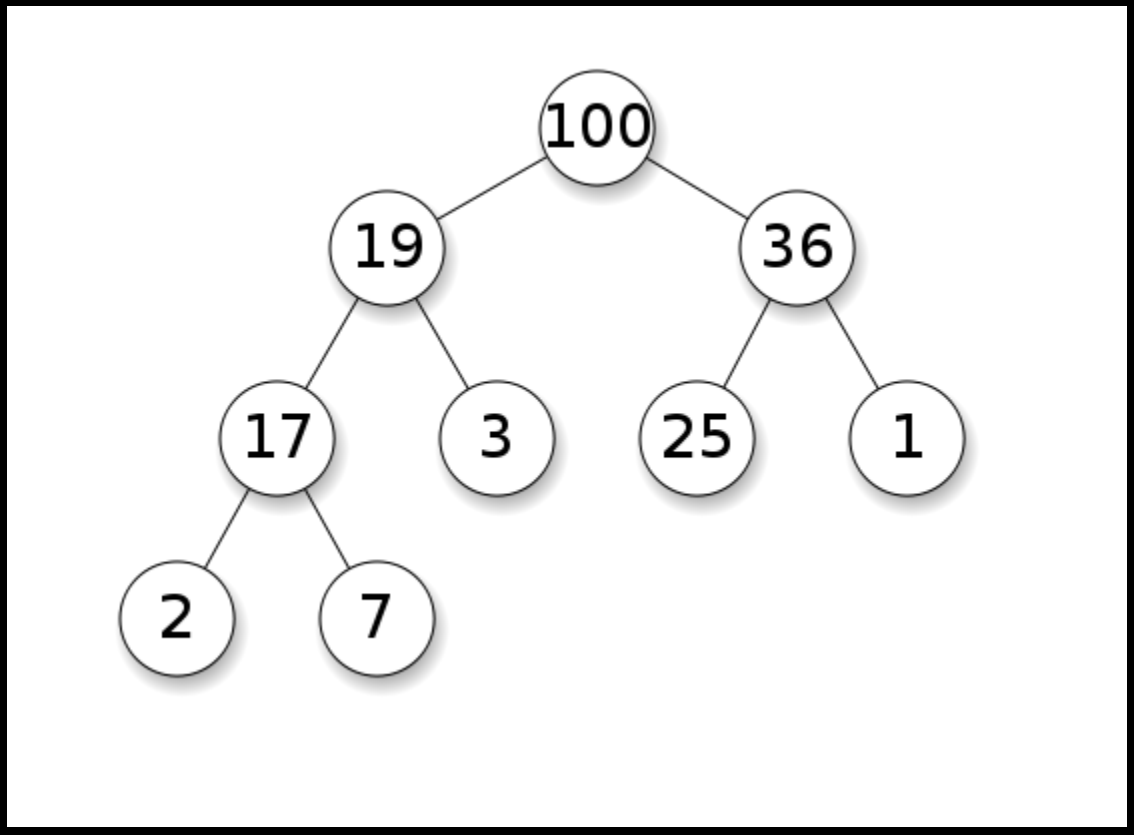
Tipos de árboles



AVL Tree



Tipos de árboles



Heap