



# Club de Algoritmia US { [Universidad de Sevilla]

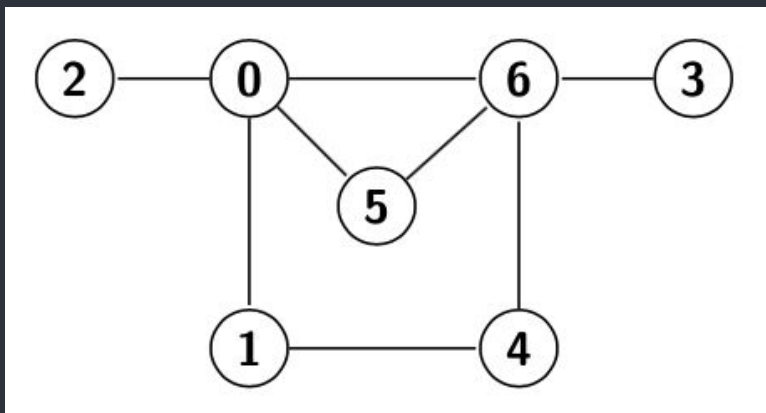
Sesión 11: Árboles

}



# Grafos

Es un conjunto de vértices (o nodos) y un conjunto de aristas (o arcos) que los unen.

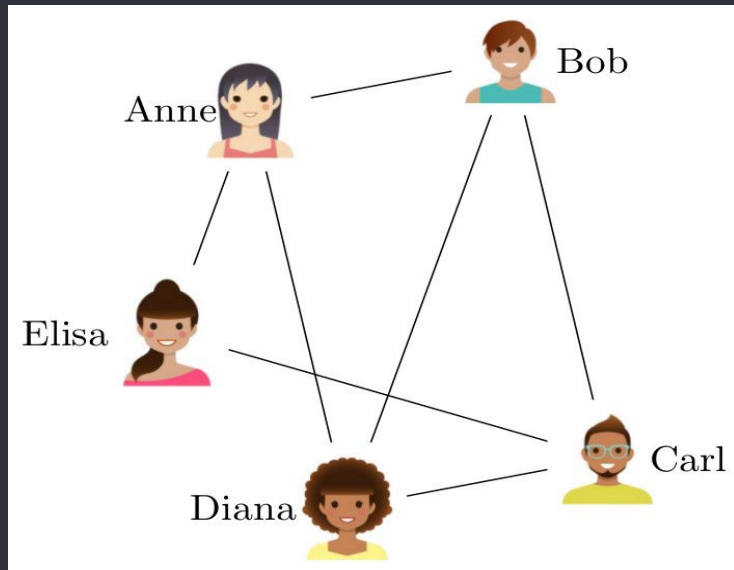
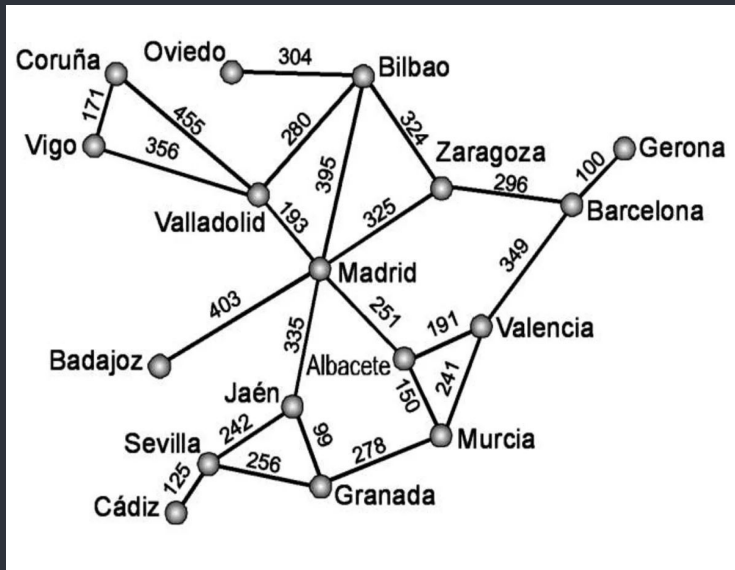




graph.py

workshop.java

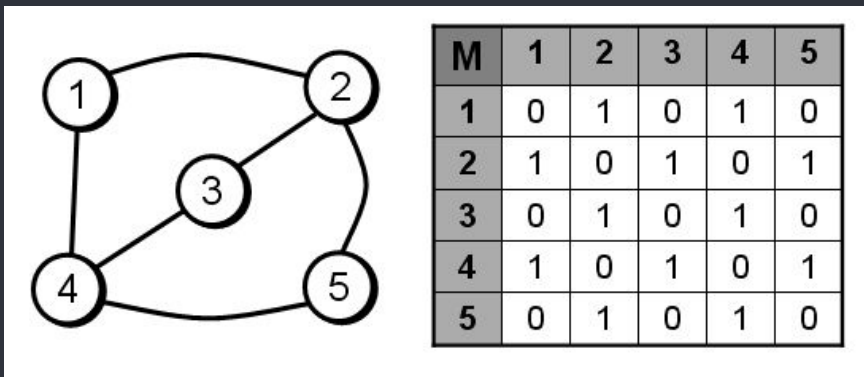
# Grafos: 'Ejemplos'





# Grafos: 'Implementación'

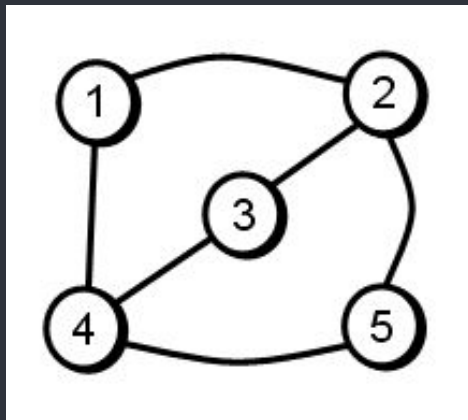
- **Matriz de adyacencias:** Se utiliza una matriz de tamaño  $n \times n$  donde el elemento  $(i,j)$  contiene la distancia entre los vértices  $i$  y  $j$  del grafo.





# Grafos: 'Implementación'

- **Lista de adyacencias:** Por cada nodo se almacena una lista de los vértices adyacentes a i.

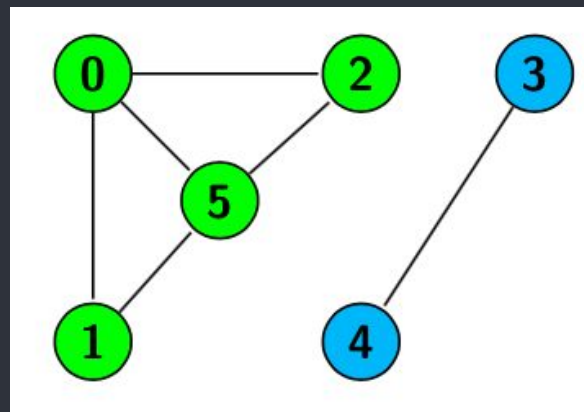
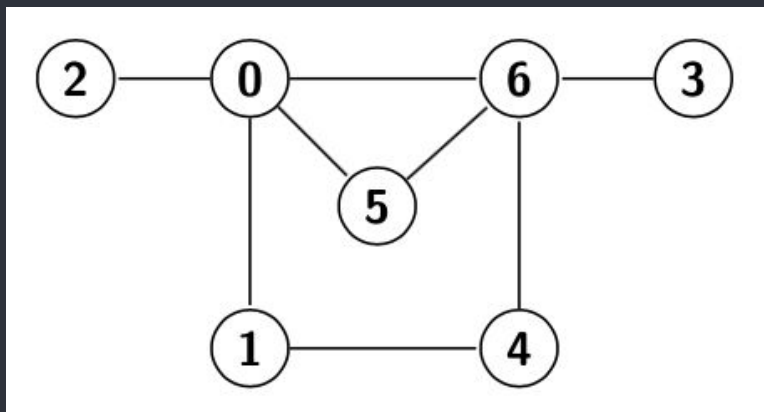


1	2, 4
2	1, 5
3	2, 4
4	1, 3, 5
5	2, 4



## Grafo conexo

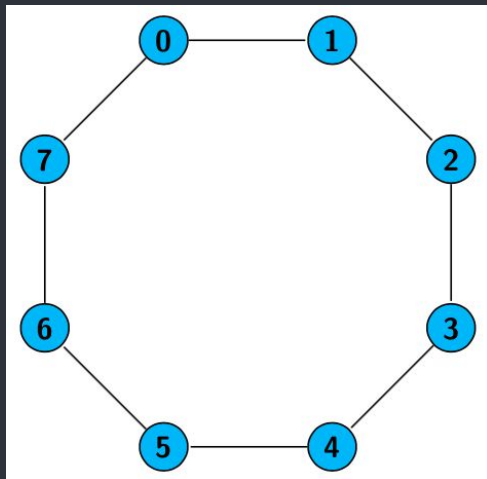
Un grafo es conexo si puedes ir desde cualquier punto a cualquier otro punto siguiendo las conexiones del grafo.





# Ciclos

Son grafos en los que cada nodo está conectado al siguiente y el último al primero.



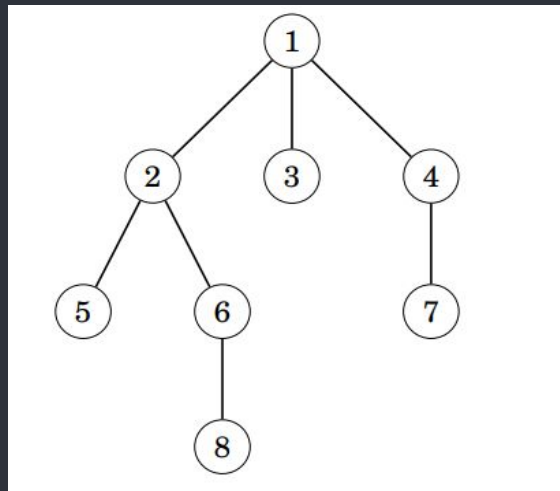
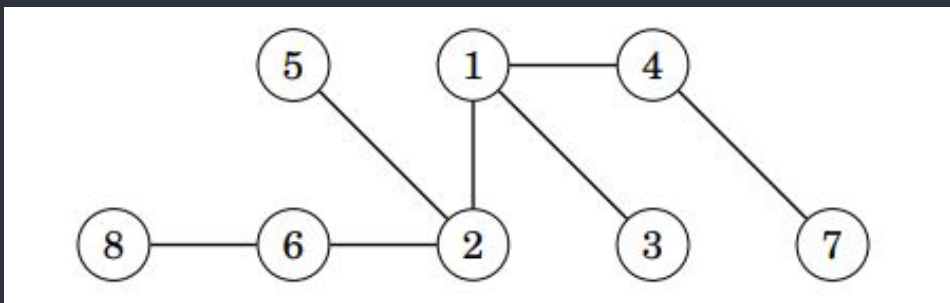


trees.py

workshop.java

# Árboles

Un árbol no es más que  
un grafo acíclico conexo.

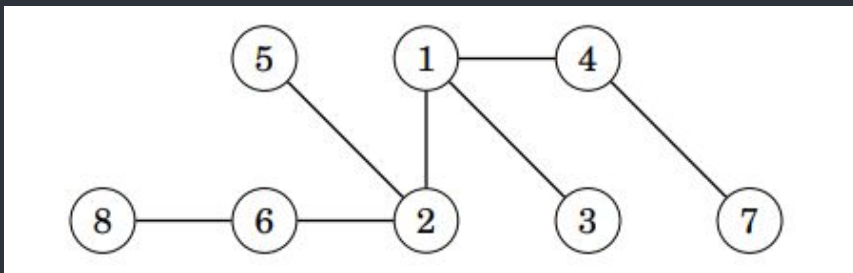






# Árboles: 'Características'

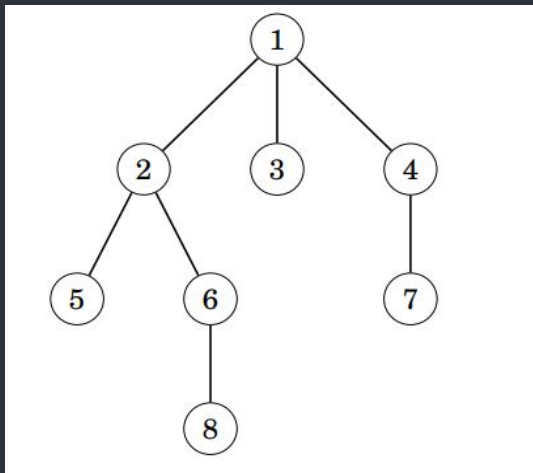
- La eliminación de cualquier arista divide el árbol en 2 componentes.
- La agregación de cualquier arista a un árbol crea un ciclo
- Siempre hay un único camino entre 2 nodos cualesquiera de un árbol





# Árbol arraigado

En determinadas ocasiones, consideramos que un árbol tiene un nodo principal al que llamamos “nodo raíz”



```
/
├── bin
├── boot
├── dev
├── etc
│   ├── init.d
│   │   ├── script1
│   │   ├── script2
│   │   └── ...
│   ├── network
│   │   ├── interfaces
│   │   └── ...
│   └── ...
├── home
│   ├── user1
│   │   ├── documents
│   │   ├── downloads
│   │   └── ...
│   ├── user2
│   │   ├── documents
│   │   ├── downloads
│   │   └── ...
│   └── ...
├── lib
│   ├── modules
│   │   ├── module1
│   │   ├── module2
│   │   └── ...
│   └── ...
```



# Árbol arraigado: 'Nomenclatura'

Los nodos del árbol se pueden organizar en capas dependiendo de su distancia a la raíz:

- El nodo principal se llama “raíz”
- Dado un nodo, sus nodos adyacentes más alejados de la raíz (capa siguiente) se les llama “hijos”.
- Dado un nodo, su nodo adyacente más cercano a la raíz (capa anterior) se le llama “padre”.
- A los nodos sin hijos se les llama “hojas”.
- La “profundidad” del árbol es la máxima distancia de la raíz a la que está alguna de las hojas.



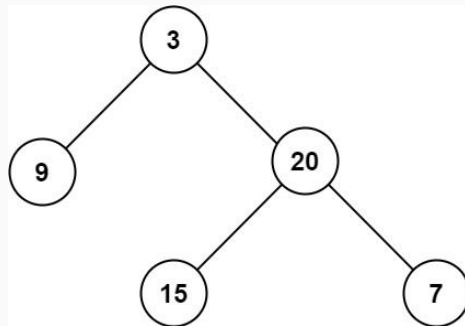
# Árbol arraigado: 'Implementación'

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

```
# Creamos los nodos
node_3 = TreeNode(3)
node_9 = TreeNode(9)
node_20 = TreeNode(20)
node_15 = TreeNode(15)
node_7 = TreeNode(7)
```

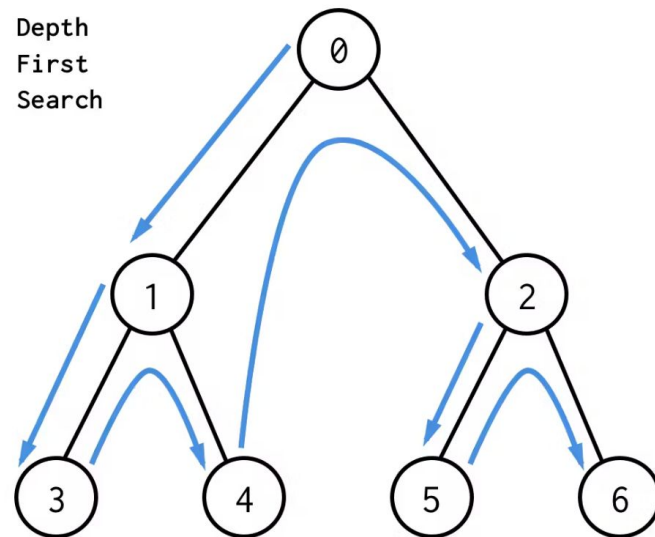
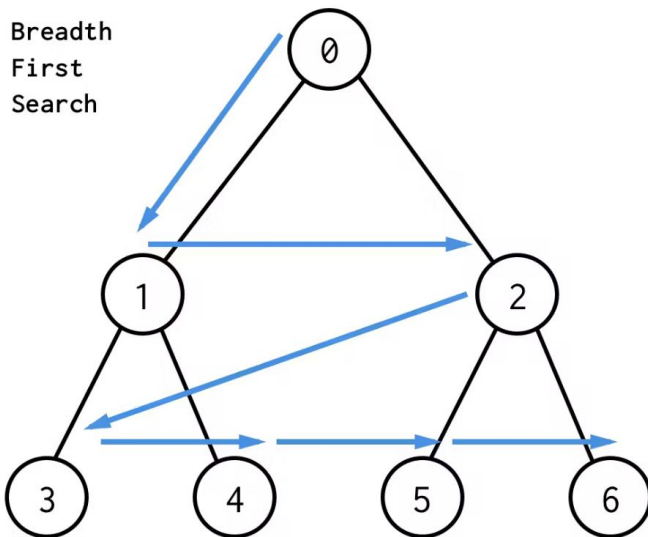
```
# Conectamos los nodos
node_3.left = node_9
node_3.right = node_20
node_20.left = node_15
node_20.right = node_7
```

```
# Asignamos el nodo raíz
root = node_3
```





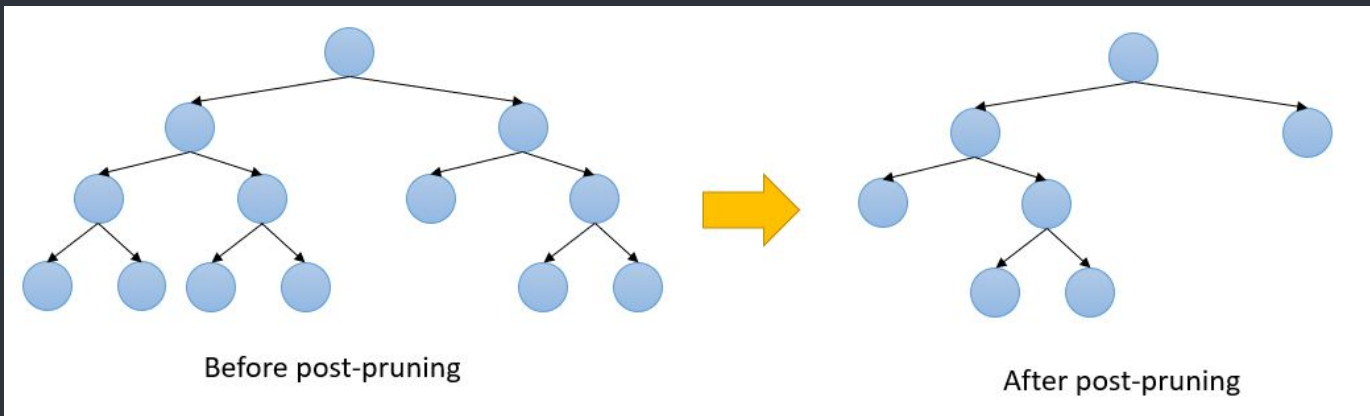
# Métodos de recorrido de árboles





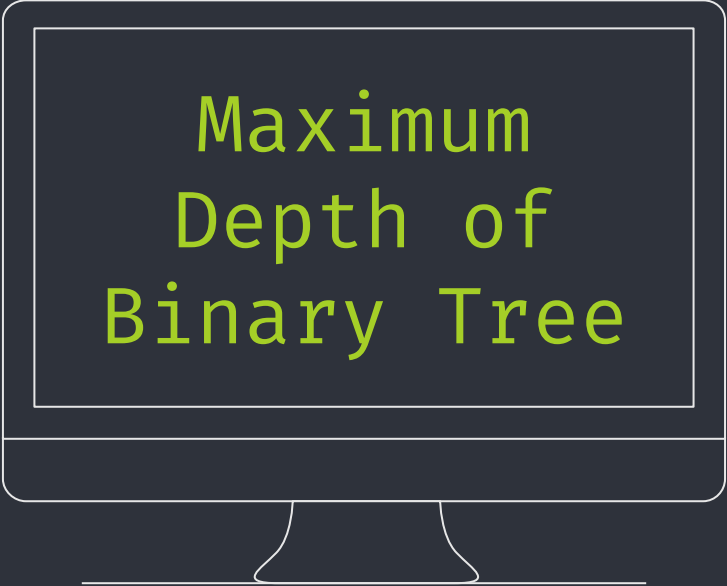
# Poda

La poda en árboles es una técnica utilizada para reducir el tamaño del árbol de búsqueda eliminando subárboles que no contribuyen a la solución del problema.





# LeetCode - Número 104



Maximum  
Depth of  
Binary Tree



## Problema: 'Maximum Depth of Binary Tree'

{

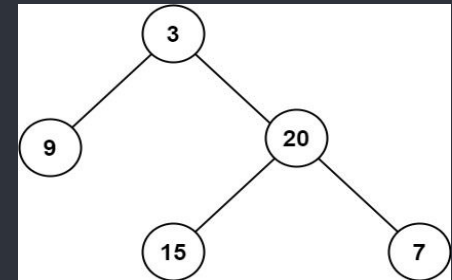
Dada la raíz de un árbol binario, devuelve su profundidad máxima.

La profundidad máxima de un árbol binario es el número de nodos a lo largo del camino más largo desde el nodo raíz hasta el nodo hoja más lejano.

**Input:** root = [3,9,20,null,null,15,7]

**Output:** 3

}





# Solución

```
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0
        return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1
```

# Ejemplo input

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

```
# Creamos los nodos
```

```
node_3 = TreeNode(3)
node_9 = TreeNode(9)
node_20 = TreeNode(20)
node_15 = TreeNode(15)
node_7 = TreeNode(7)
```

```
# Conectamos los nodos
```

```
node_3.left = node_9
node_3.right = node_20
node_20.left = node_15
node_20.right = node_7
```

```
# Asignamos el nodo raíz
```

```
root = node_3
```



# LeetCode - Número 111

Minimum  
Depth of  
Binary Tree

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14



## Problema: 'Minimum Depth of Binary Tree'

{

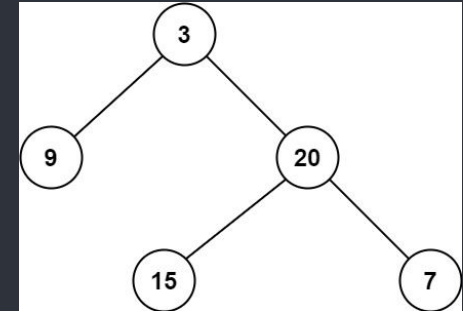
Dado un árbol binario, hallar su profundidad mínima.

La profundidad mínima es el número de nodos a lo largo del camino más corto desde el nodo raíz hasta el nodo hoja más cercano.

**Input:** root = [3,9,20,null,null,15,7]

**Output:** 2

}



# Solución

— □ ×

```
class Solution(object):
    def minDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root is None:
            return 0
        if not root.left and not root.right:
            return 1
        if not root.left:
            return self.minDepth(root.right) + 1
        if not root.right:
            return self.minDepth(root.left) + 1
        return min(self.minDepth(root.left), self.minDepth(root.right)) + 1
```



# LeetCode - Número 1022

Sum of Root  
To Leaf  
Binary  
Numbers

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14



## Prob: 'Sum of Root To Leaf Binary Numbers'

{

Se te da la raíz de un árbol binario donde cada nodo tiene un valor de 0 o 1. Cada camino de la raíz a una hoja representa un número binario comenzando con el bit más significativo.

Por ejemplo, si el camino es:  $0 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow 1$ , esto podría representar 01101 en binario, que es 13.

}



## Prob: 'Sum of Root To Leaf Binary Numbers'

{

Para todas las hojas en el árbol, considera los números representados por el camino desde la raíz hasta esa hoja. Devuelve la suma de estos números.

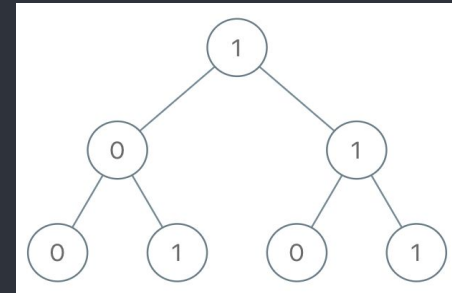
**Input:** root = [1,0,1,0,1,0,1]

**Output:** 22

**Explanation:**

$(100) + (101) + (110) + (111) =$   
 $= 4 + 5 + 6 + 7 = 22$

}





# Solución

tree.py

```
class TreeNode:
```

```
    def __init__(self, val=0, left=None, right=None):  
        self.val = val  
        self.left = left  
        self.right = right
```

```
class Solution:
```

```
    def solve(self, root: Optional[TreeNode], acc: int):  
        new_acc = 2*acc + root.val  
  
        if root.left is None and root.right is None:  
            return new_acc  
  
        if root.left is None:  
            left_sol = 0  
        else:  
            left_sol = self.solve(root.left, new_acc)  
  
        if root.right is None:  
            right_sol = 0  
        else:  
            right_sol = self.solve(root.right, new_acc)  
  
        return left_sol + right_sol  
  
    def sumRootToLeaf(self, root: Optional[TreeNode]) -> int:  
        if root is None:  
            return 0  
  
        return self.solve(root, 0)
```



# LeetCode - Número 110



Balanced  
Binary Tree



# Problema: 'Balanced Binary Tree'

{

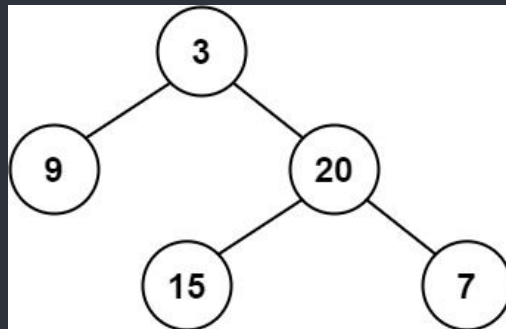
Dado un árbol binario, determinar si está equilibrado en altura

**Input:**

root = [3,9,20,null,null,15,7]

**Output:** true

}



# Solución

tree.py

```
class TreeNode:

    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

```
class Solution:

    def solve(self, root: Optional[TreeNode]) -> int | None:
        if root.left is None:
            depth_left = 0
        else:
            depth_left = self.solve(root.left)

        if root.right is None:
            depth_right = 0
        else:
            depth_right = self.solve(root.right)

        if (
            depth_right is None
            or depth_left is None
            or abs(depth_right - depth_left) > 1
        ):
            return None
        else:
            return max(depth_left, depth_right) + 1

    def isBalanced(self, root: Optional[TreeNode]) -> bool:
        if root is None:
            return True

        return self.solve(root) is not None
```



end.py

workshop.java

1 ¡Gracias!  
2  
3  
4 ¡Nos vemos la semana que  
5 viene!  
6  
7  
8  
9

- 10 • Próxima sesión: Colas de prioridad
- 11    ◦ (Esta vez de verdad)
- 12 • Alguien se viene a escalar?
- 13
- 14

