Árboles

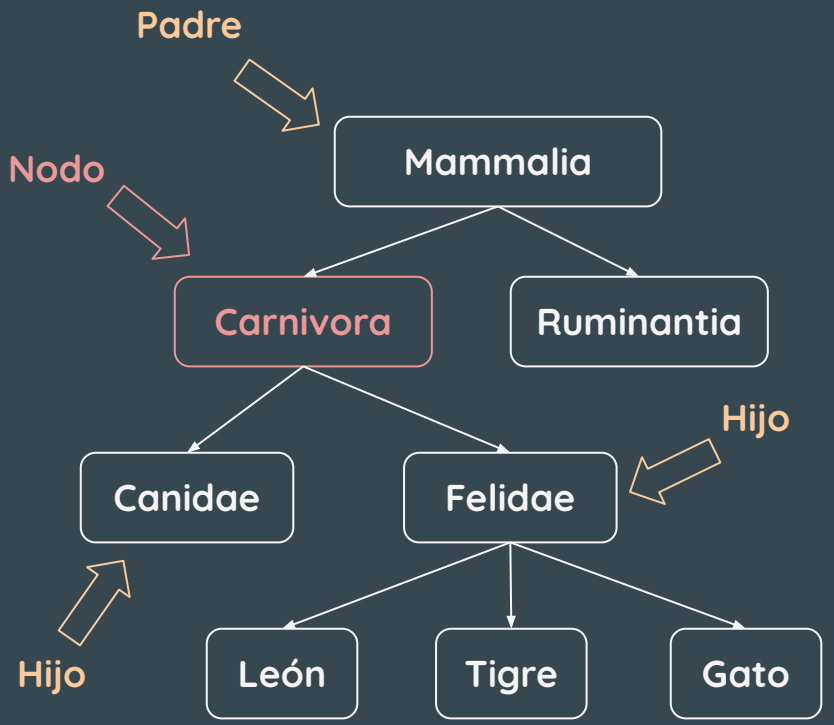
...

Algoritmos y Estructuras de Datos
2023

UNRaf UNIVERSIDAD NACIONAL DE RAFAELA

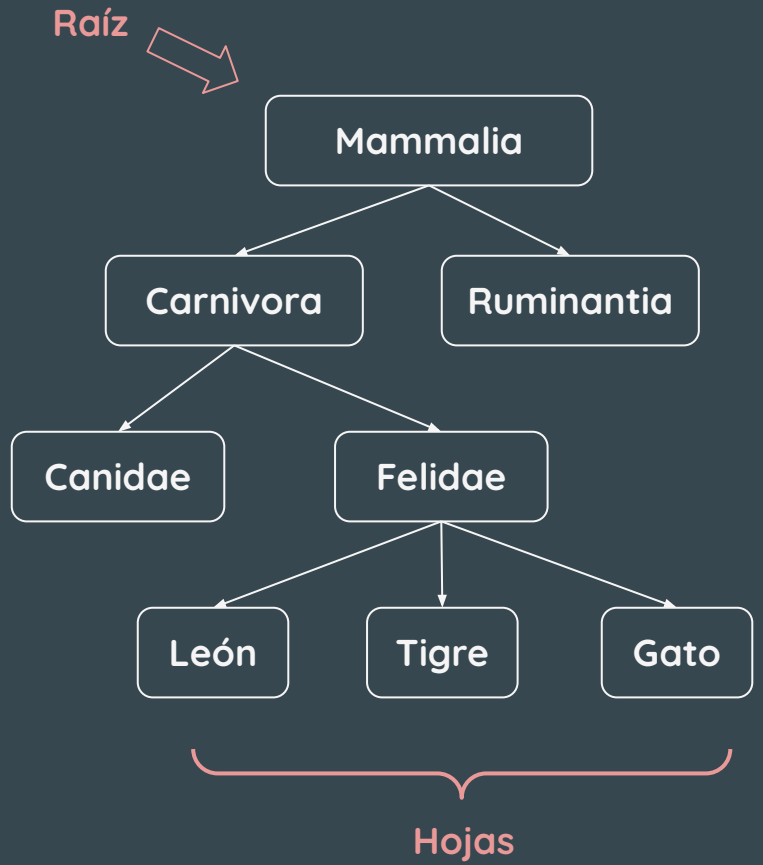
Definición

- Un árbol es una estructura de datos abstracta que ordena sus elementos de manera **jerárquica**.
- Están compuestos por nodos que, en su mayoría, poseen un **nodo padre** y 0 o más **nodos hijos**.



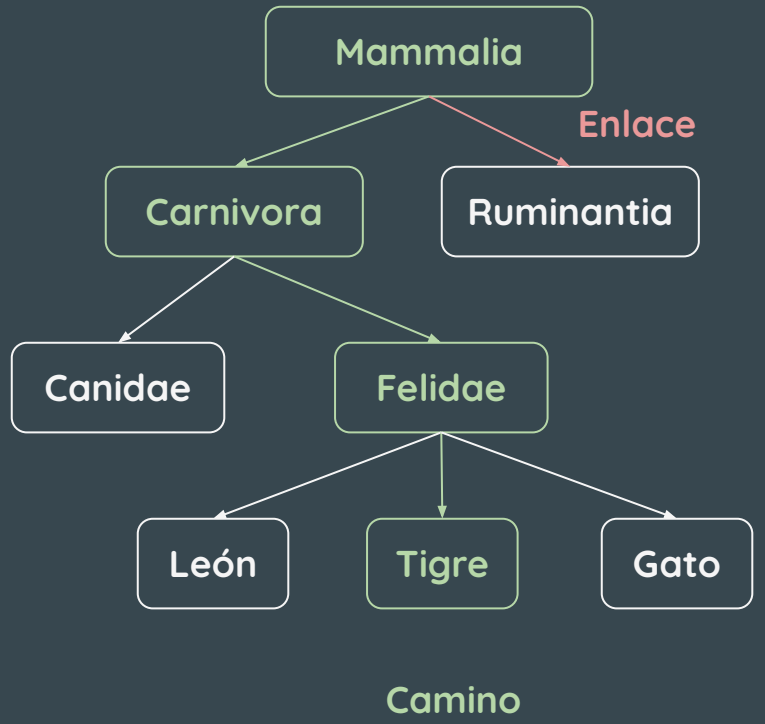
Definición

- El nodo **raíz** es el nodo principal de un árbol y es el único sin nodo padre.
- Los nodos al final del árbol, sin nodos hijos, se denominan **hojas**.



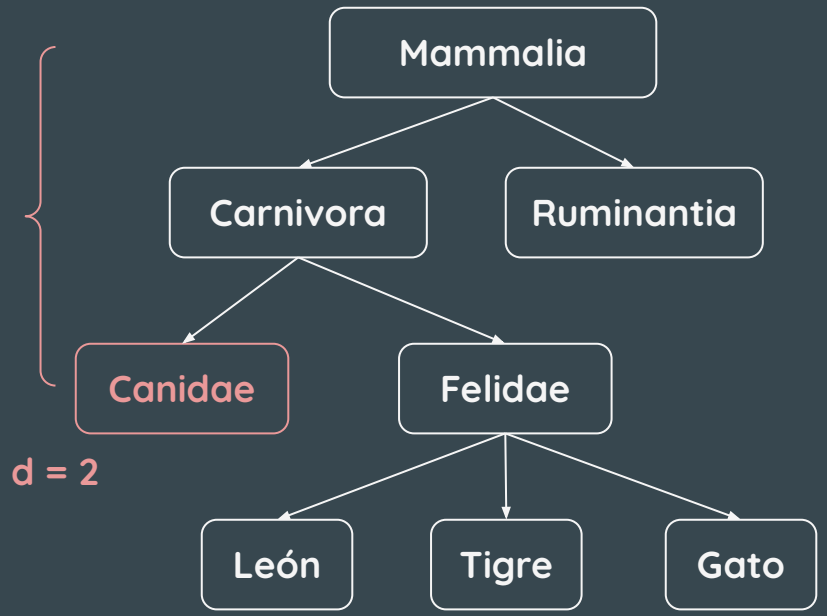
Más terminología

- La conexión entre dos nodos se denomina **enlace** o **unión** (***edge*** en inglés).
- La sucesión de nodos enlazados que conecta dos nodos específicos se denomina **camino** (***path***)



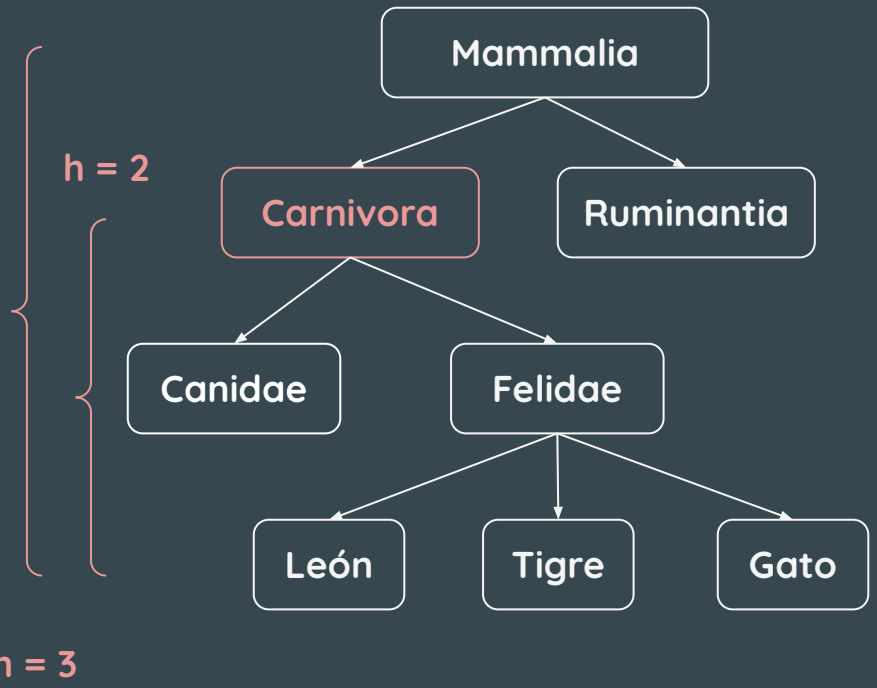
Profundidad y altura

- Se denomina **profundidad** (***depth***) a la cantidad de nodos padres que tiene un nodo.
- La profundidad del nodo raíz es siempre 0.

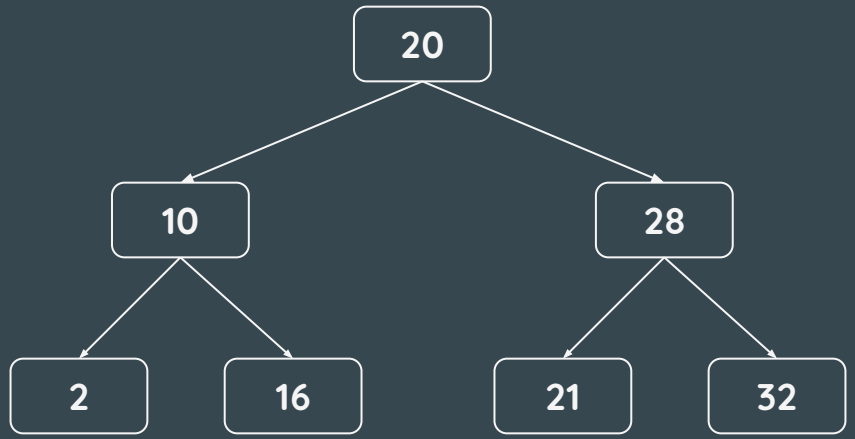


Profundidad y altura

- La **altura** de un nodo es la **máxima profundidad** alcanzada por sus hojas hasta llegar a dicho nodo.
- La altura de las hojas es siempre 0.
- La altura total de un árbol es la altura de su nodo raíz.



Árboles binarios

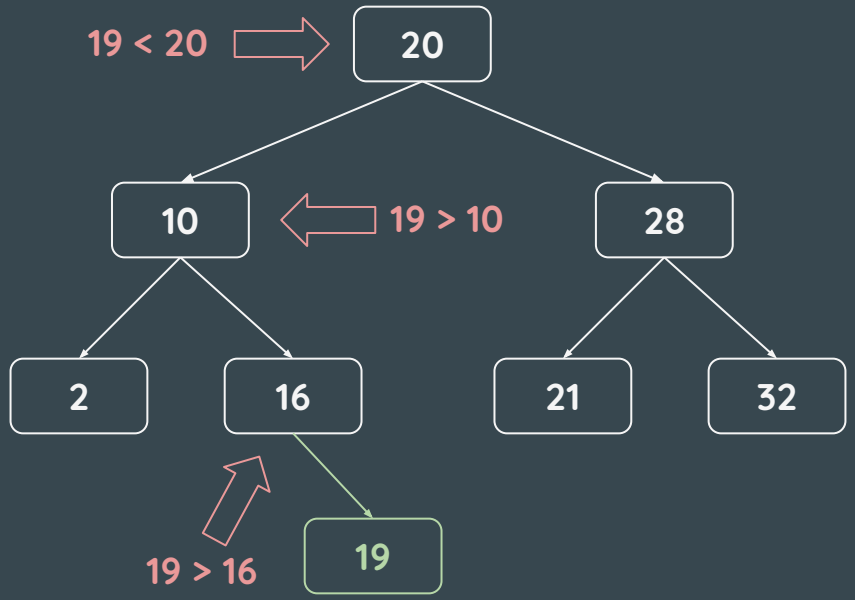


- Son una subclase de árboles donde cada nodo puede tener, a lo sumo, **dos nodos hijos**.
- Estos nodos son denominados normalmente **izquierda** y **derecha**.

Características de los árboles binarios

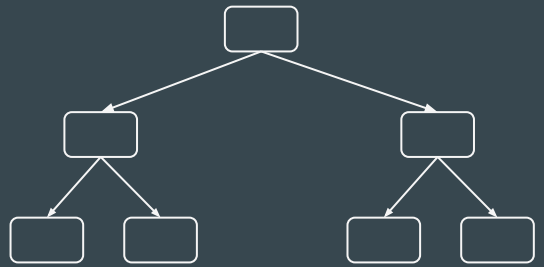
- En general, los datos en un árbol son ubicados en un árbol en relación con su valor.
- Si el valor a agregar es mayor que el de un nodo, se lo agrega en el **subárbol de la derecha**; sino en el **subárbol de la izquierda**.
- Un árbol se denomina **balanceado** si la diferencia la altura del subárbol de la izquierda y del subárbol de la derecha es, a lo sumo, 1.

Árboles binarios

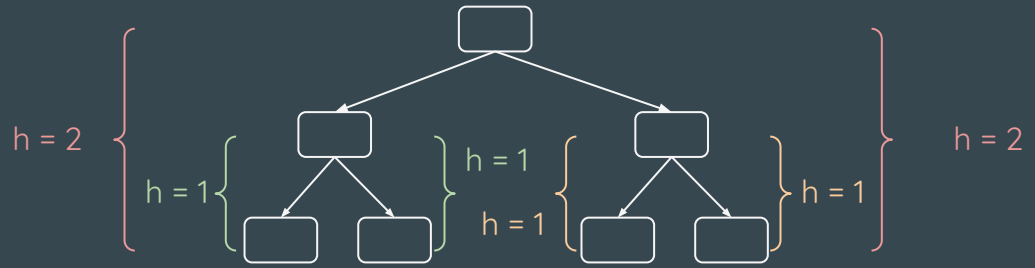


- Si se quisiera agregar el valor **19**, se comienza comparando desde la raíz.
- Se visita, recursivamente, un nodo hijo dependiendo de si el valor a ingresar es mayor o menor que el almacenado en el nodo.

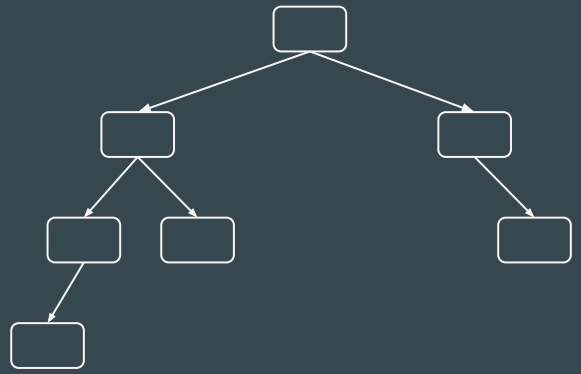
Balanceo de árboles



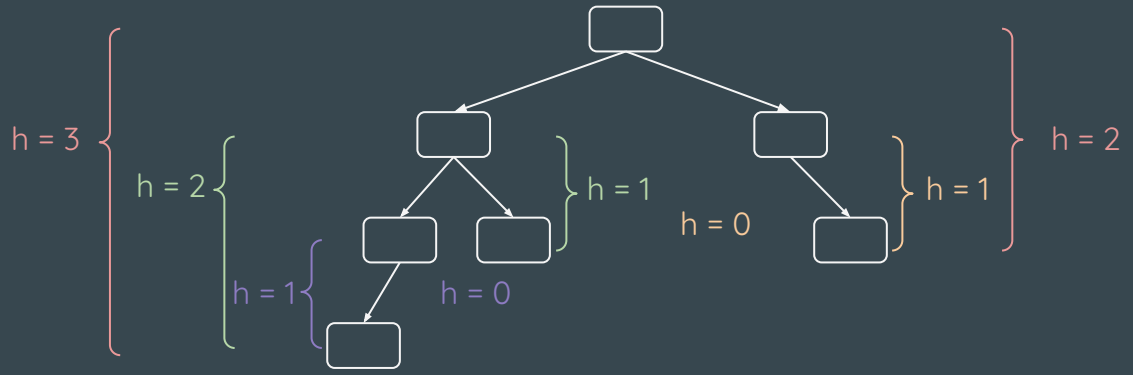
Balanceo de árboles



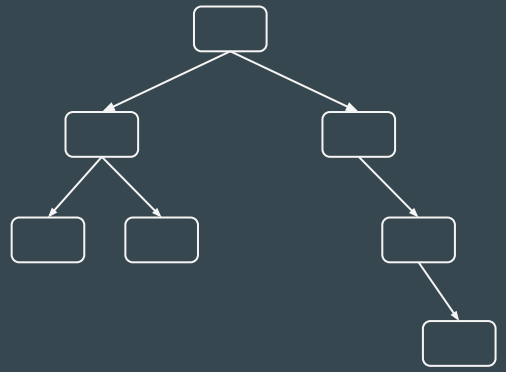
Balanceo de árboles



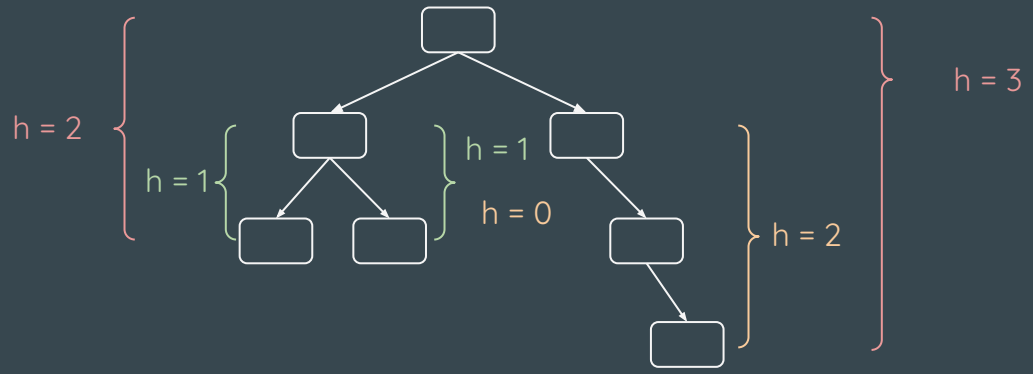
Balanceo de árboles



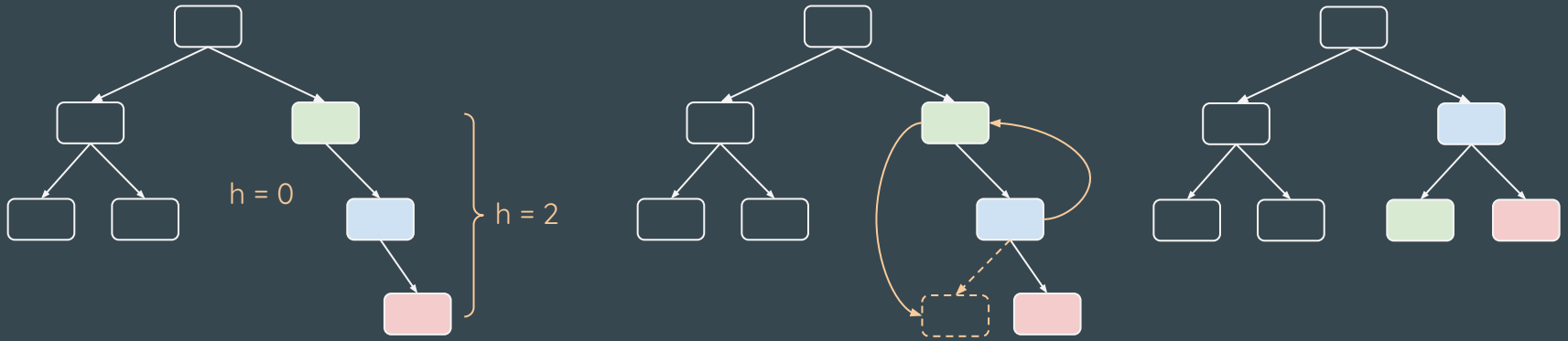
Balanceo de árboles



Balanceo de árboles



Balanceo de árboles



Implementando árboles

- Un árbol está compuesto por **nodos**
- Un nodo contiene un **elemento** o valor y **punteros**.
- En árboles binarios se requieren 3 punteros: padre, izquierda y derecha

```
class Nodo:
  def __init__(self, e):
    self._elemento = e
    self._padre = None
    self._izquierda = None
    self._derecha = None
```

Implementando árboles

- La clase árbol tiene que tener si o si definido cual es el **nodo raíz**.
- Al momento de iniciar, se pueden agregar más variables, como por el ejemplo el tamaño (cantidad de nodos)

```
class ArbolBinario:
  def __init__(self):
    self._raiz = None
    self._tamano = 0
```

Implementado funciones para los árboles

```
class ArbolBinario:
  def __len__(self):
    return self._tamano

  def vacio(self):
    if self._tamano == 0:
      return True

  def raiz(self):
    return self._raiz

  def es_raiz(self, n):
    if self._raiz == n:
       return True
```

Implementado funciones para los árboles

```
class ArbolBinario:
  def es_hoja(self, n):
    if n._izquierda == None and n._derecha == None:
      return True

  def padre(self, n):
    if self.es_raiz(n):
      raise ValueError (“n es el nodo raiz”)
    return n._padre

  def izquierda(self, n):
    if n._izquierda != None:
      return n._izquierda

  def derecha(self, n):
    if n._derecha != None:
      return n._derecha
```

Implementado funciones para los árboles

```
class ArbolBinario:
  def profundidad(self, n):
    if self.es_raiz(n):
      return 0
    else:
      return 1 + profundidad(n._padre)

  def altura(self, n):
    if self.es_hoja(n):
      return 0
    elif n._izquierda == None:
      return 1 + self.altura(n._derecha)
    elif n._derecha == None:
      return 1 + self.altura(n._izquierda)
    elif self.altura(n._izquierda) > self.altura(n._derecha):
      return 1 + self.altura(n._izquierda)
    else:
      return 1 + self.altura(n._derecha)
```

Implementado funciones para los árboles

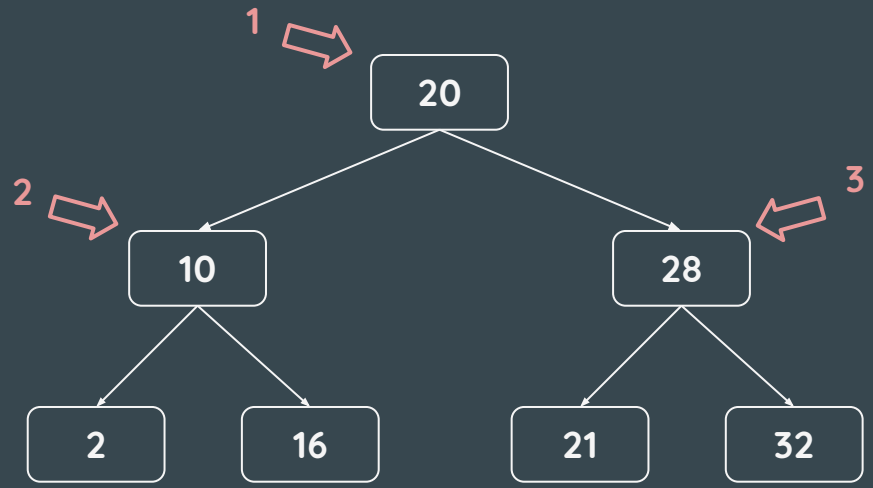
```
def agregar(self, e, n=None):
  if self.vacio():
    n = Nodo(e)
    self._raiz = n
    self._tamano += 1
  elif n == None:
    n = self._raiz

  if e < n._elemento and n._izquierda != None:
    self.agregar(e, n._izquierda)
  elif e < n._elemento and n._izquierda == None:
    new = Nodo(e)
    new._padre = n
    n._izquierda = new
    self._tamano += 1
  elif e > n._elemento and n._derecha != None:
    self.agregar(e, n._derecha)
  elif e > n._elemento and n._derecha == None:
    new = Nodo(e)
    new._padre = n
    n._derecha = new
    self._tamano += 1
```

Recorrido de árboles

- Existen diversas formas de recorrer todos los nodos de un árbol.
- Las 4 mas comunes son:
  - Preorder.
  - Postorder.
  - Inorder.
  - por anchura (breadth-first).

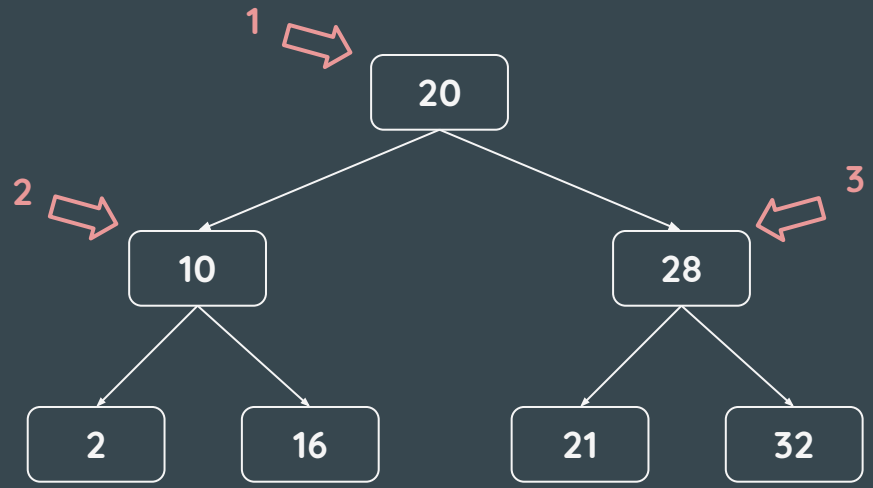
Recorrido preorder

1) Se visita el nodo
2) Se recorre el subárbol de la izquierda
3) Se recorre el subárbol de la derecha



Recorrido preorder

1) Se visita el nodo
2) Se recorre el subárbol de la izquierda
3) Se recorre el subárbol de la derecha



Resultado: 20; 10; 2; 16; 28; 21; 32

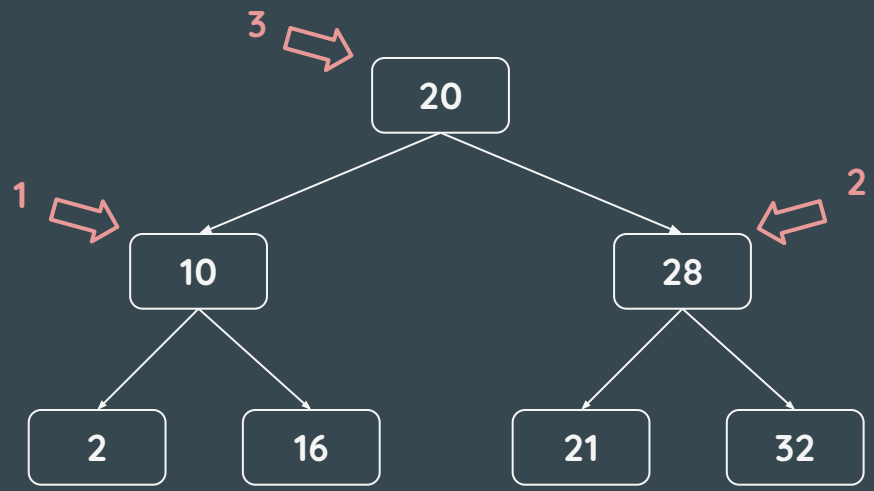
Recorrido preorder

1) Se visita el nodo
2) Se recorre el subárbol de la izquierda
3) Se recorre el subárbol de la derecha

```
def preorder(self, n = None):
  if n == None:
    n = self._raiz

  print(n._elemento)
  if n._izquierda != None:
    self.preorder(n._izquierda)
  if n._derecha != None:
     self.preorder(n._derecha)
```

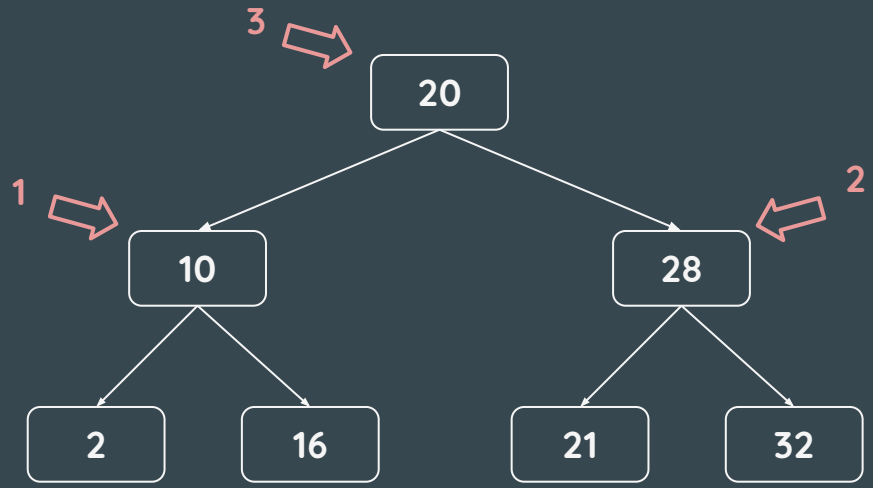
Recorrido postorder

1) Se recorre el subárbol de la izquierda.
2) Se recorre el subárbol de la derecha.
3) Se visita el nodo.



Recorrido postorder

1) Se recorre el subárbol de la izquierda.
2) Se recorre el subárbol de la derecha.
3) Se visita el nodo.



Resultado: 2; 16; 10; 21; 32; 28; 20

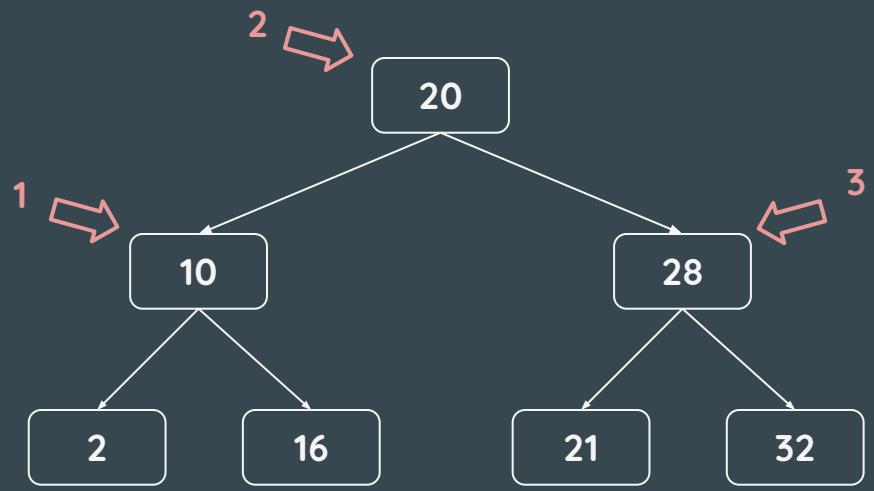
Recorrido postorder

1) Se recorre el subárbol de la izquierda.
2) Se recorre el subárbol de la derecha.
3) Se visita el nodo.

```
def postorder(self, n = None):
  if n == None:
    n = self._raiz

  if n._izquierda != None:
    self.postorder(n._izquierda)
  if n._derecha != None:
     self.postorder(n._derecha)
  print(n._elemento)
```

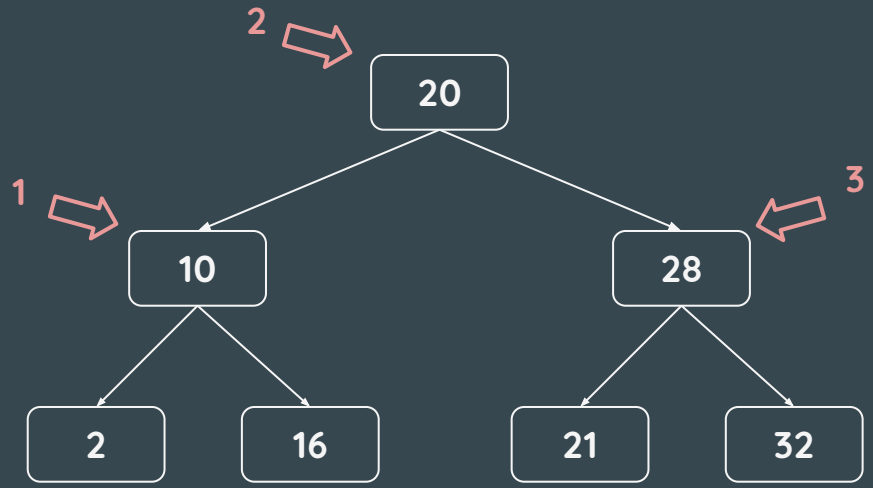
Recorrido inorder

1) Se recorre el subárbol de la izquierda
2) Se visita el nodo
3) Se recorre el subárbol de la derecha



Recorrido inorder

1) Se recorre el subárbol de la izquierda.
2) Se visita el nodo.
3) Se recorre el subárbol de la derecha.



Resultado: 2; 10; 16; 20; 21; 28; 32

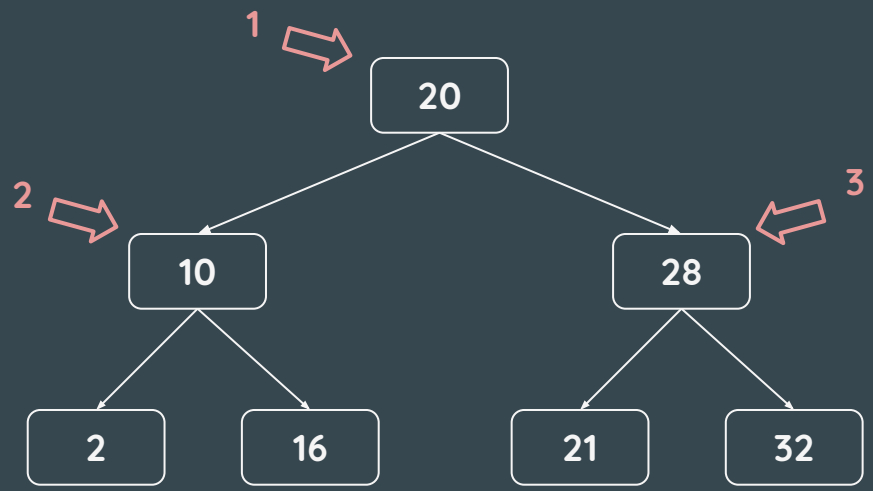
Recorrido inorder

1) Se recorre el subárbol de la izquierda.
2) Se visita el nodo.
3) Se recorre el subárbol de la derecha.

```
def inorder(self, n = None):
  if n == None:
    n = self._raiz

  if n._izquierda != None:
    self.inorder(n._izquierda)
  print(n._element)
  if n._derecha != None:
     self.inorder(n._derecha)
```

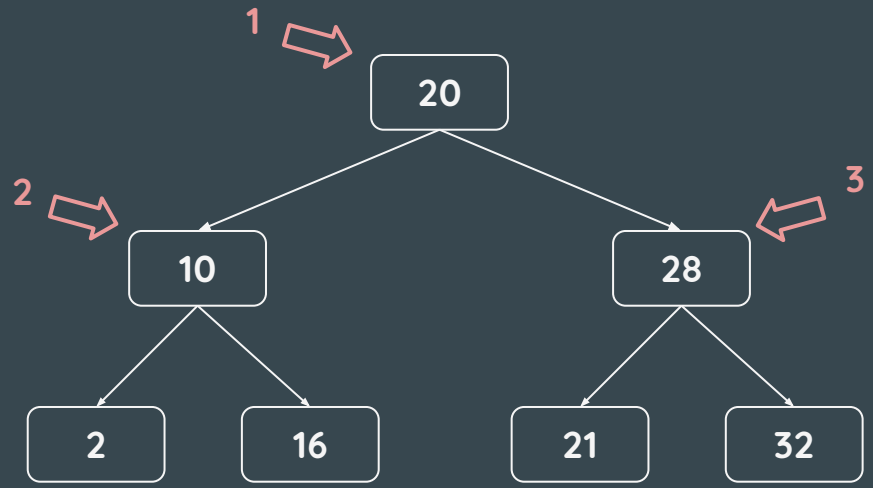
Recorrido por anchura

1) Se inicia una cola de visitas con el nodo raíz.
2) Se agregan todos los nodos hijos.
3) Se quita de la cola.
4) Se visita el siguiente nodo en la cola.



Recorrido por anchura

1) Se inicia una cola de visitas con el nodo raíz.
2) Se agregan todos los nodos hijos.
3) Se quita de la cola.
4) Se visita el siguiente nodo en la cola.



Resultado: 20; 10; 28; 2; 16; 21; 32

Recorrido por anchura

1) Se inicia una cola de visitas con el nodo raíz.
2) Se agregan todos los nodos hijos.
3) Se quita de la cola.
4) Se visita el siguiente nodo en la cola.

```
def por_anchura(self):
  cola = []
  cola.append(self._raiz)
  while len(cola) > 0:
    n = cola.pop(0)
    print(n._elemento)
    if n._izquierda != None:
      cola.append(n._izquierda)
    if n._derecha != None:
      cola.append(n._derecha)
```

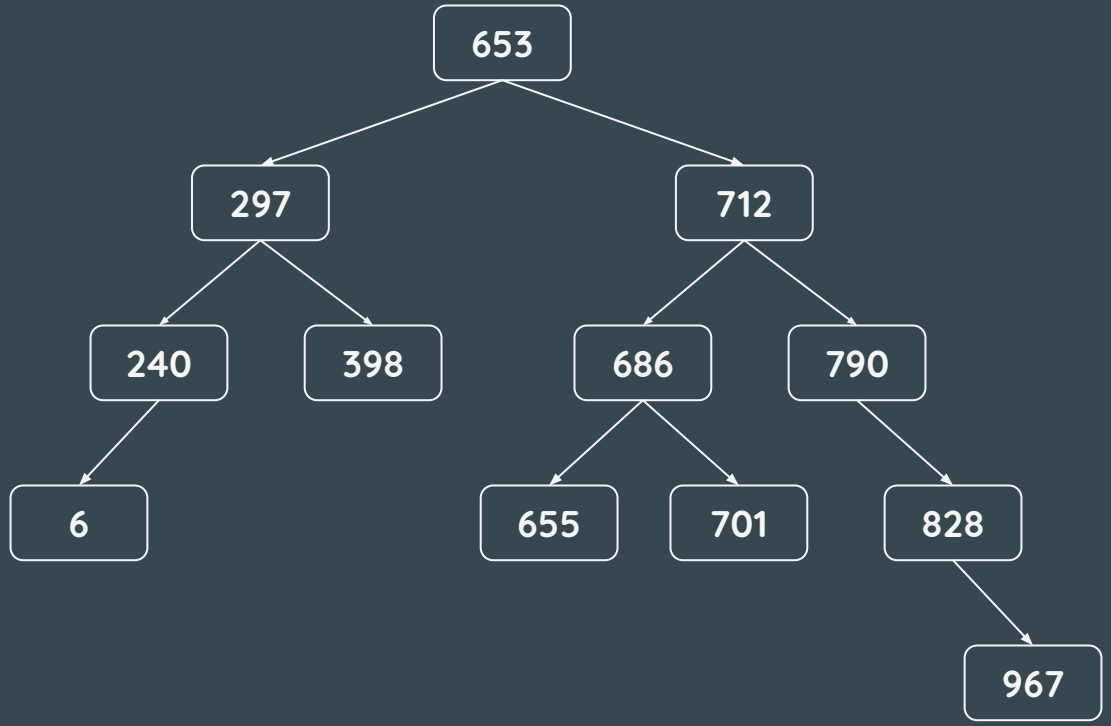
Ejemplo

Dada la siguiente lista de números:

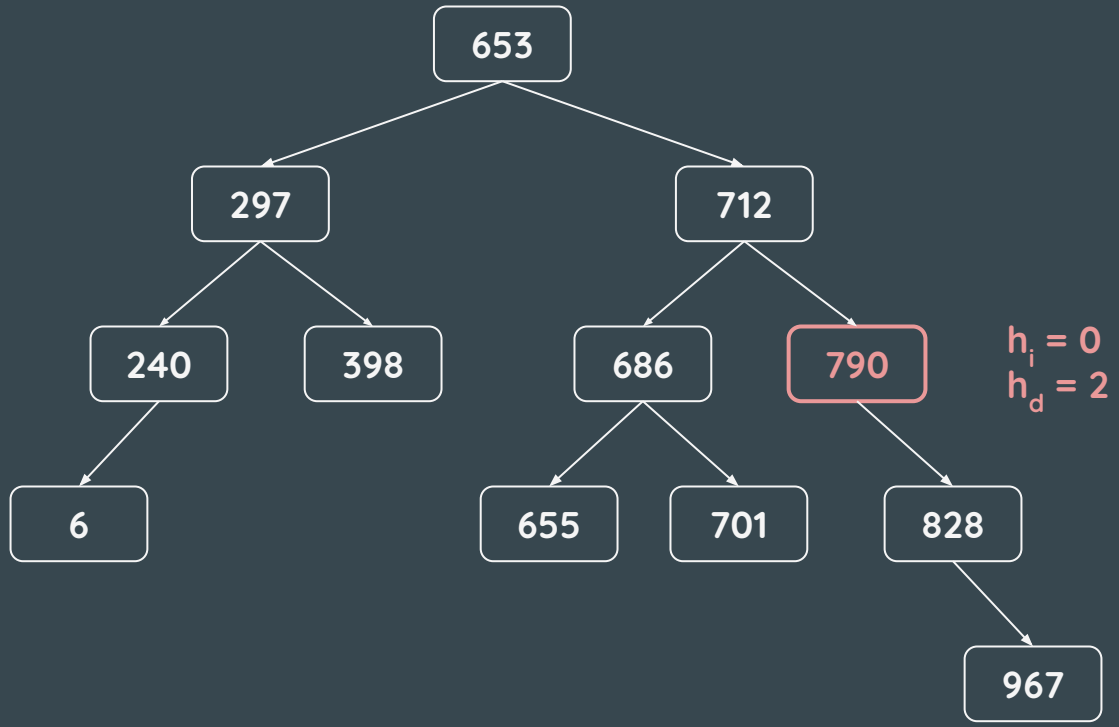
653, 712, 790, 297, 686, 398, 701, 240, 6, 655, 828, 967

1. Construya un árbol, respetando el orden en el que aparecen los números
2. Determine si está balanceado
3. Recorralo usando las estrategias preorder, inorder y por anchura

Ejemplo: construcción



Ejemplo: balanceado?



Ejemplo: recorridos

- Preorder:

  653; 297; 240; 6; 398; 712; 686; 655; 701; 790; 828; 967

- Inorder:

  6; 240; 297; 398; 653; 655; 686; 701; 712; 790; 828; 967

- Por anchura:

  653; 297; 712; 240; 398; 686; 790; 6; 655; 701; 828; 967

Bibliografía

- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data structures and algorithms in Python. Capitulo 8.