Ayudantía:
BINARY SEARCH TREE

Árbol binario de búsqueda (ABB)

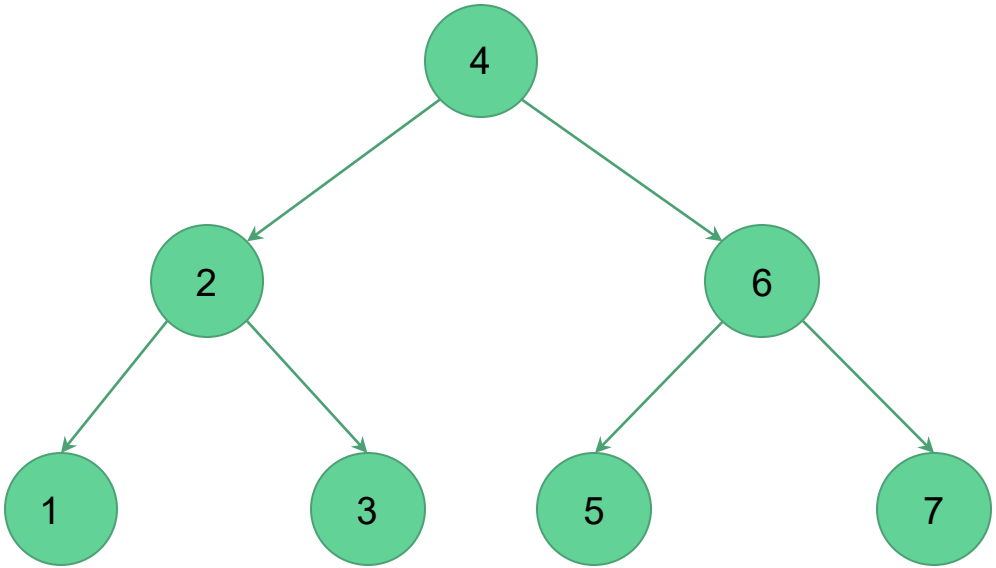
- En inglés **BST (*Binary Search Tree*)**
- Cada nodo tiene asociados **dos** ABB , **mediante punteros**
- Sea x el nodo, entonces:
  - x.key = llave
  - x.value = valor
  - x.p es el padre del nodo. En caso de no tener padre, entonces es la raíz
  - **Tiene máximo dos hijos**
  - x.left es el hijo izquierdo el cual cumple que **x.left < x**
  - x.right es el hijo derecho el cual cumple que **x.right > x**
  - en caso de **x.right == null y x.left == null, entonces x es hoja**

¿Cómo se ve un nodo en C?

```
struct node {
    int key;
    struct node *left;
    struct node *right;
};
```

Binary Search Tree

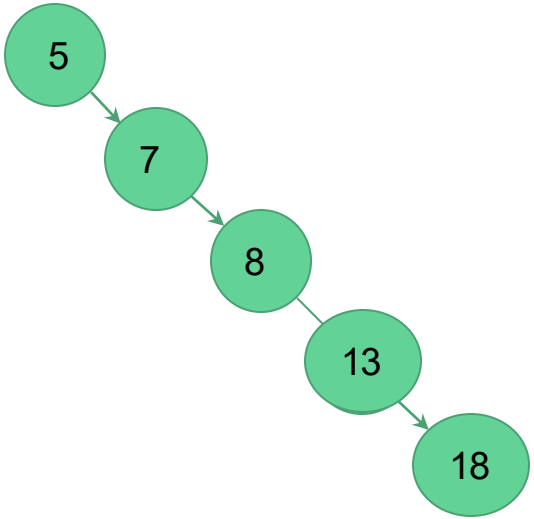
- Altura mínima de la rama más larga es O(log(n)) -> **MEJOR CASO**



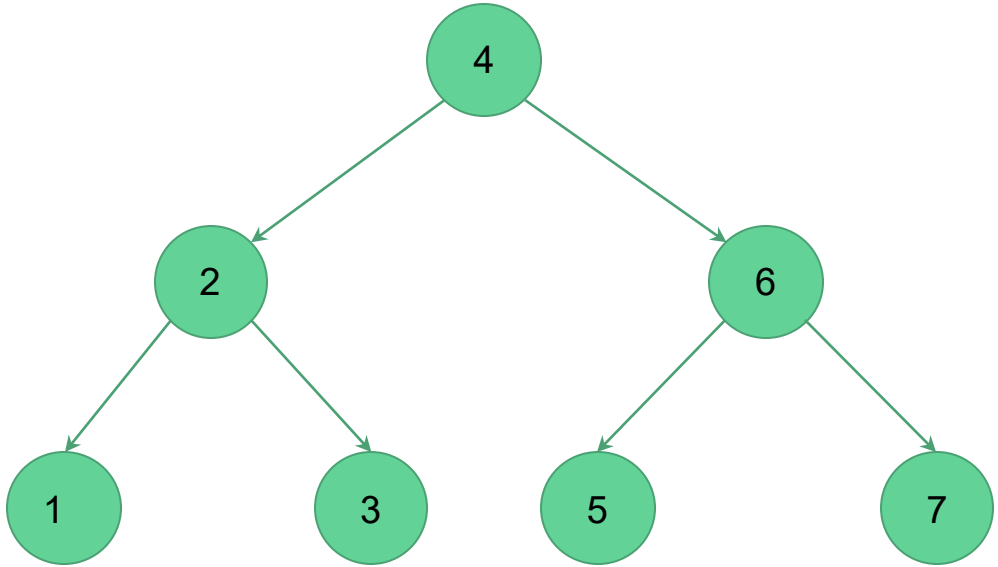
- **Cuando está perfectamente balanceado**
- Dado que la diferencia de altura entre el subárbol izquierdo y el subárbol derecho de cada nodo es como máximo 1. con n el número de nodos.

Árbol binario de búsqueda (ABB)

- Altura máxima de O(n), cuando no está balanceado



¿Cómo recorrer árboles?

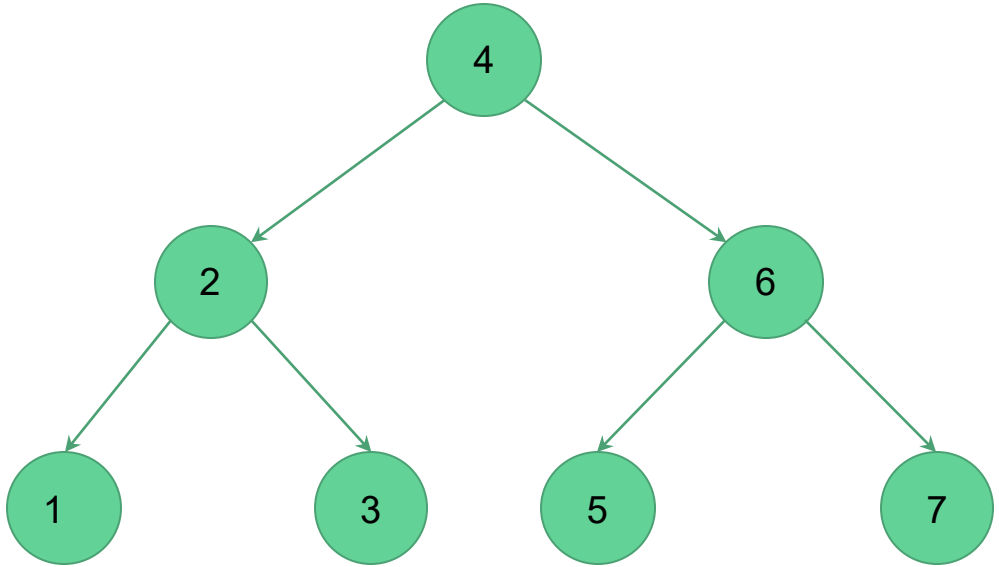



```
void InOrder(Node* node)
{
    if (node == NULL)
       return;
     InOrder(node->left);
     printf('%d', node->value);
     InOrder(node->right);
}
```

```
void PreOrder(Node* node)
{
   if (node == NULL)
        return;
   PreOrder(node->left);
   PreOrder(node->right);
   printf('%d', node->value);
}
```

```
void PostOrder(Node* node)
{
   if (node == NULL)
      return;
   printf('%d', node->value);
   PostOrder(node->left);
   PostOrder(node->right);
}
```

¿Cómo recorrer árboles?

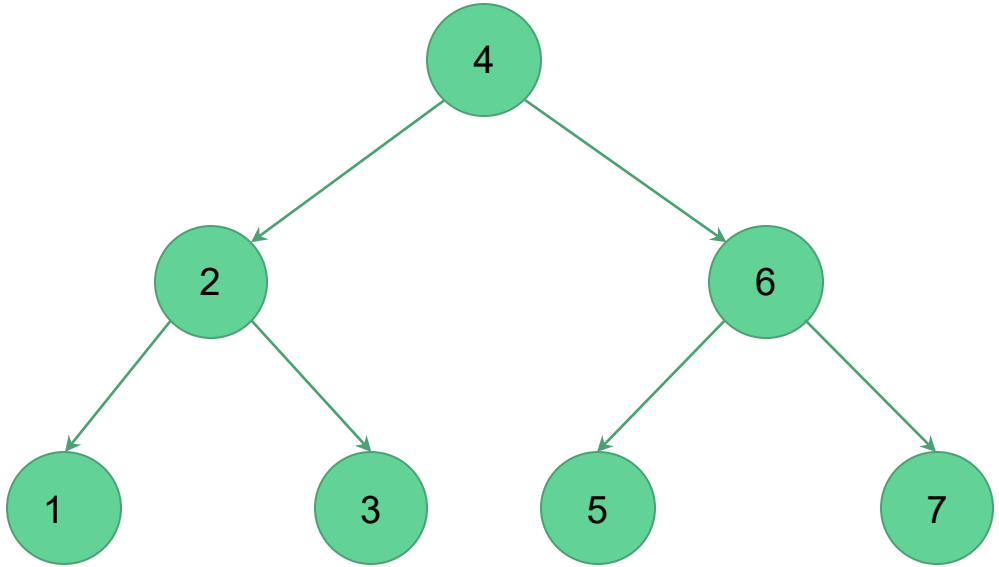


InOrder

```
void InOrder(Node* node)
{
    if (node == NULL)
        return;
     InOrder(node->left);
     printf('%d', node->value);
     InOrder(node->right);
}
```

1234567

¿Cómo recorrer árboles?

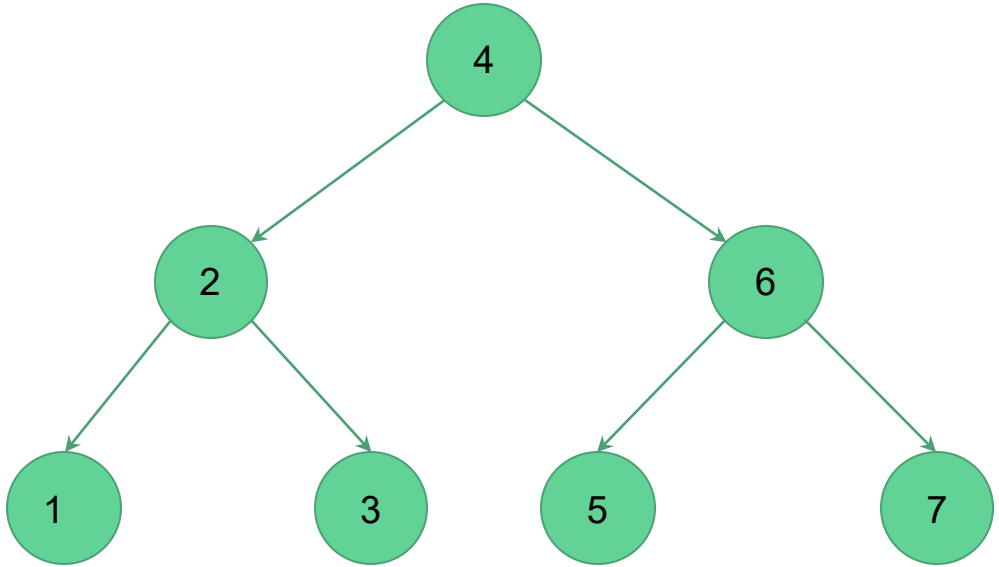


PreOrder

```
void PreOrder(Node* node)
{
  if (node == NULL)
     return;
  PreOrder(node->left);
  PreOrder(node->right);
  printf('%d', node->value);
}
```

4213657

¿Cómo recorrer árboles?



PostOrder

```
void PostOrder(Node* node)
{
    if (node == NULL)
        return;
    printf('%d', node->value);
    PostOrder(node->left);
    PostOrder(node->right);
}
```

1325764

Qué es balance

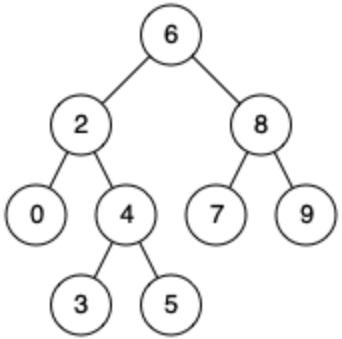
- En un árbol con **n nodos**, la altura se mantiene en **$O(\log_2(n))$**.
- Debe ser **fácil de mantener** en cada operación.
- La definición de balance se basa en criterios **recursivos**, de modo que tras cada inserción o eliminación se puedan aplicar ajustes locales.
- Así, el proceso de balanceo se asegura en **tiempo menor o igual a $O(\log_2(n))$**, evitando que el árbol degenere en una lista.
- Esto se verá más claro en un *árbol AVL* (que veremos la próoxima semana)

Ejercicio 1: Ancestro Común Más Bajo en un Árbol de Búsqueda Binaria (BST)

Dado un árbol de búsqueda binaria (BST), encuentra el nodo que es el **ancestro común más bajo (Lowest Common Ancestor = LCA)** de dos nodos dados en el BST.

"El ancestro común más bajo (LCA) se define entre dos nodos *p* y *q* como el nodo más bajo en *T* que tiene tanto a *p* como a *q* como descendientes (donde permitimos que un nodo sea descendiente de sí mismo)."

Ejemplos



Ejemplo 1

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explicación: El ancestro común más bajo de los nodos 2 y 8 es 6.

Ejemplo 2

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explicación: El ancestro común más bajo de los nodos 2 y 4 es 2, ya que un nodo puede ser descendiente de sí mismo según la definición de LCA.

Solución

```
lowestCommonAncestor(root, p, q):
    small ← mín(p.value, q.value)
    large ← máx(p.value, q.value)

    while root ≠ null:
        if root.valor > large:
            root ← root.left
        elif root.valor < small:
            root ← root.right
        else:
            return raíz

    return null
```

p y q están en el subárbol izquierdo

p y q están en el subárbol derecho

small ≤ root.value ≤ large
Este es el LCA

Ejercicio 2

Sean T1 y T2 dos ABB de n y m nodos respectivamente. Explica, de manera clara y precisa, cómo realizar un merge entre ambos árboles en O(n+m) pasos para dejarlos como un solo ABB balanceado T con los nodos de ambos árboles.

Ejercicio 2

Merge entre T1 y T2 en O(n+m) ?

1. Se itera por sobre los nodos de ambos árboles de manera ordenada, copiando los nodos a un array. Este paso consiste en un proceso recursivo que visita siempre el nodo izquierdo antes que el derecho (recorrido in-order). Haciendo esto tenemos dos arreglos ordenados, con los elementos de T1 y T2 respectivamente.

2. Combinamos estos dos arreglos usando la subrutina merge de MergeSort en O(n + m). Con esto obtenemos un arreglo ordenado A con los n + m elementos de ambos árboles.

3. Finalmente, se convierte el array ordenado A en un ABB balanceado poniendo la mediana como raíz e insertando los valores menores y mayores a esta en las ramas izquierdas y derechas respectivamente. Notar que encontrar la mediana en un array ordenado es O(1) por lo que proceso se realiza en O(m+n) pasos.

Ejercicio 2

¿Complejidad?

La solución consta de tres pasos O(m + n)

La rutina completa es O(m + n)

Búsqueda Binaria

```
BinarySearch (A, x, i, f):
1     if f < i : return −1
2     m ← ⌊(i + f) / 2⌋
3     if A[m] = x : return m
4     if A[m] > x :
5         return BinarySearch (A, x, i, m − 1)
6     return BinarySearch (A, x, m + 1, f)
```

- Elige posición media m
- Si el elemento en la pos m es igual al buscado retorno m
- Si elemento en la pos m es mayor busco por el lado izquierdo del array. eoc busco por lado derecho

Ejercicio 3 Binary Search

Se te da un arreglo de enteros piles donde piles[i] es el número de plátanos en la pila i. También se te da un entero h, que representa el número de horas que tienes para comerte todos los plátanos.

Puedes decidir tu tasa de consumo de plátanos por hora con un valor k. Cada hora puedes elegir una pila de plátanos y comer k plátanos de esa pila. Si la pila tiene menos de k plátanos, puedes terminar de comer esa pila, pero no puedes comer de otra pila en la misma hora.

Devuelve el entero mínimo k tal que puedas comerte todos los plátanos dentro de h horas.

Debes apuntar a una solución con tiempo O(n log m) y espacio O(1), donde n es el tamaño del arreglo de entrada y m es el valor máximo en el arreglo.

Ejemplos

Ejemplo 1:

Input: piles = [1,4,3,2], h = 9
Output: 2
Explicación: Con una velocidad de 2, puedes comer todos los plátanos en 6 horas. Con una velocidad de 1, necesitarías 10 horas para comerlos todos (lo cual excede h = 9), por lo tanto, la velocidad mínima es 2.

Ejemplo 2:

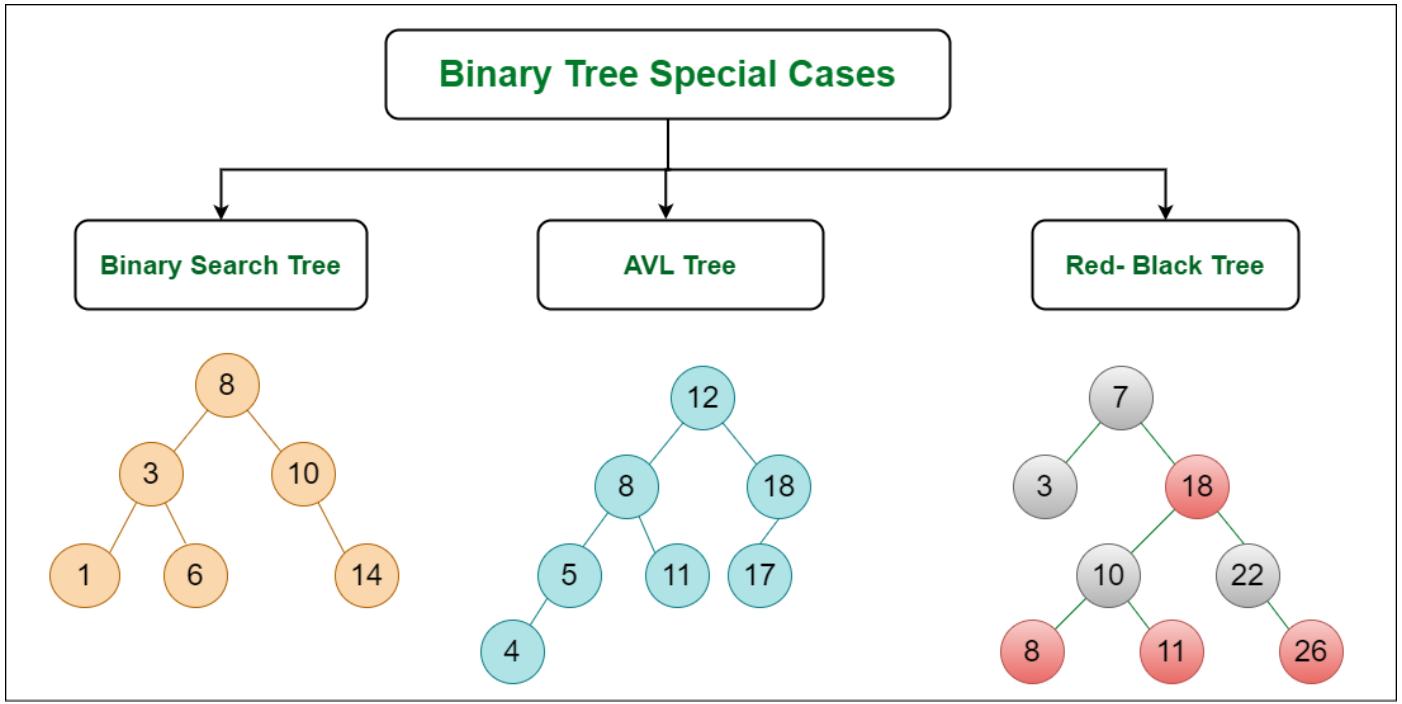
Input: piles = [25,10,23,4], h = 4
Output: 25
Explicación: Con menos de 25 plátanos por hora no alcanzarías a terminar en 4 horas, así que la velocidad mínima es 25.

Solución

```
MinEatingSpeed (piles, h) :
1.      l ← 1
2.      r ← max(piles)
3.      res ← r
4.      while l ≤ r :
5.          k ← ⌊(l + r)/2⌋
6.          totalTime ← 0
7.          for each p ∈ piles :
8.              totalTime ← totalTime + ⌈p/k⌉
9.          if totalTime ≤ h :
10.             res ← k
11.             r ← k − 1
12.         else:
13.             l ← k + 1
14.     return res
```

O(n log(m))

Ejemplos de árboles binarios



Ejemplos de árboles binarios

