ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS ARBORESCENTES

Funciones sobre árboles binarios

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

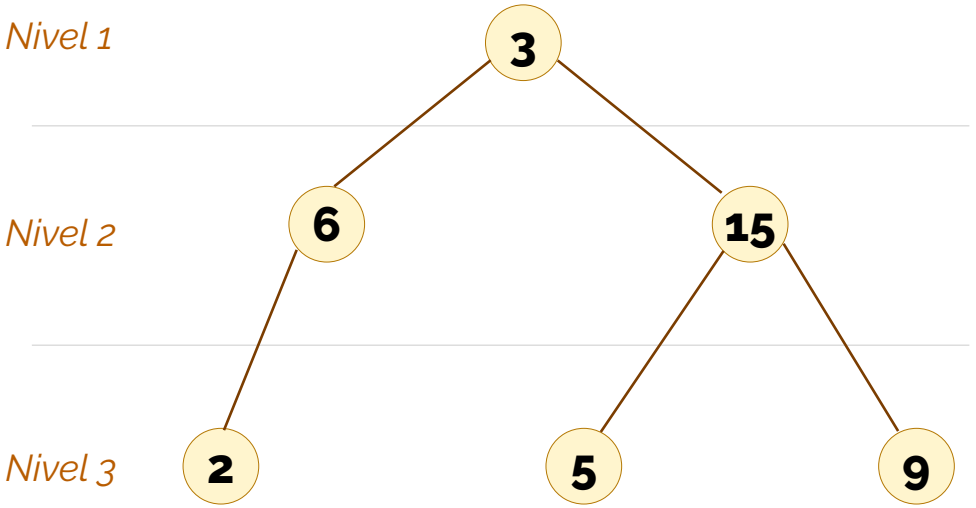
Recordatorio: interfaz de BinTree<T>

```
template<class T>
class BinTree {
public:
  BinTree();
  BinTree(const T &elem);
  BinTree(const BinTree &left, const T &elem, const BinTree &right);

  const T & root() const;
  BinTree left() const;
  BinTree right() const;
  bool empty() const;

private:
  ...
};
```

Recordatorio: altura de un árbol binario

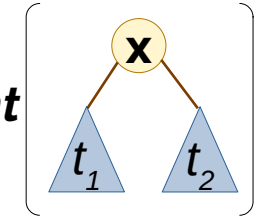


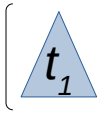
- La **altura** de un árbol es el máximo de los niveles de los nodos.

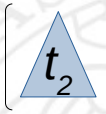
Definición recursiva de altura

- Es posible definir recursivamente la altura de un árbol binario:

height(—) = 0

height 

= 1 + max(***height*** 

, ***height*** 

)

Función height

$$height(—) = 0$$

$$height\left(\begin{array}{c} x \\ t_1 \quad t_2 \end{array}\right) = 1 + \max(\, height(t_1), height(t_2)\,)$$

```
template<typename T>
int height(const BinTree<T> &tree) {
  if (tree.empty()) {
    return 0;
  } else {
    return 1 + std::max(height(tree.left()), height(tree.right()));
  }
}
```

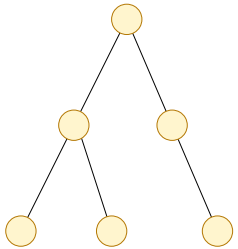
Coste en tiempo

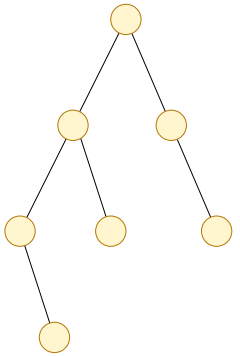
```
template<typename T>
int height(const BinTree<T> &tree) {
  if (tree.empty()) {
    return 0;
  } else {
    return 1 + std::max(height(tree.left()), height(tree.right()));
  }
}
```

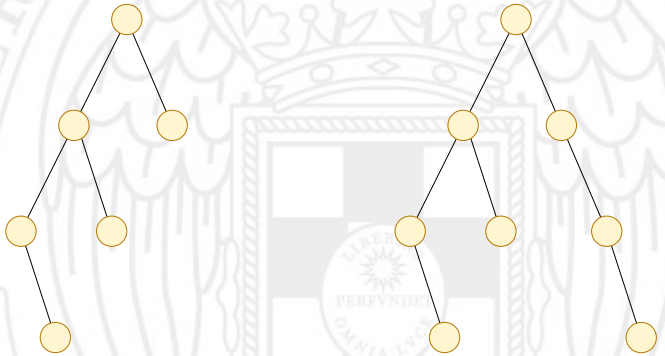
Árboles equilibrados en altura

Un árbol está **equilibrado en altura** si:

- Es el árbol vacío, o bien
- La diferencia entre las alturas de sus hijos es, como mucho, 1, y ambos están equilibrados en altura.







Definición recursiva

$$\textbf{balanced}(\text{—}) = \textit{true}$$

$$\textbf{balanced}\left(\begin{array}{c} x \\ t_1 \quad t_2 \end{array}\right) = \textbf{balanced}(t_1) \wedge \textbf{balanced}(t_2) \wedge \left|\textbf{height}(t_1) - \textbf{height}(t_2)\right| \leq 1$$

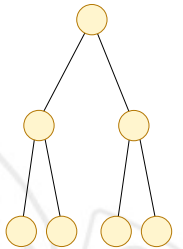
Función balanced

```
template<typename T>
bool balanced(const BinTree<T> &tree) {
  if (tree.empty()) {
    return true;
  } else {
    bool bal_left = balanced(tree.left());
    bool bal_right = balanced(tree.right());
    int height_left = height(tree.left());
    int height_right = height(tree.right());
    return bal_left && bal_right && abs(height_left - height_right) ⩽ 1;
  }
}
```

¿Cuál es el coste en tiempo?

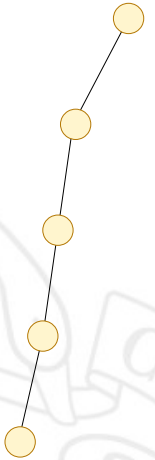
Función balanced: caso mejor

```
template<typename T>
bool balanced(const BinTree<T> &tree) {
  if (tree.empty()) {
    return true;
  } else {
    bool bal_left = balanced(tree.left());
    bool bal_right = balanced(tree.right());
    int height_left = height(tree.left());
    int height_right = height(tree.right());
    return bal_left && bal_right && abs(height_left - height_right) ⩽ 1;
  }
}
```

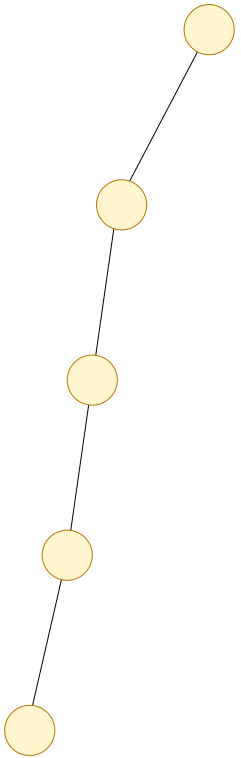


Función balanced: caso peor

```
template<typename T>
bool balanced(const BinTree<T> &tree) {
  if (tree.empty()) {
    return true;
  } else {
    bool bal_left = balanced(tree.left());
    bool bal_right = balanced(tree.right());
    int height_left = height(tree.left());
    int height_right = height(tree.right());
    return bal_left && bal_right && abs(height_left - height_right) ⩽ 1;
  }
}
```



Problema de llamar a `height`



¿Cómo solucionarlo?

- Implementando una función auxiliar recursiva que **simultáneamente** calcule la altura y determine si un árbol está equilibrado.
- Esta función devuelve dos valores:
  - `balanced` (`bool`) – si el árbol está equilibrado o no.
  - `height` (`int`) – altura del árbol.
- La función `balanced_height` devolverá ambos valores como parámetros de salida.

Función balanced_height

```
template<typename T>
void balanced_height(const BinTree<T> &tree, bool &balanced, int &height) {
  if (tree.empty()) {
    balanced = true;
    height = 0;
  } else {
    bool bal_left, bal_right;
    int height_left, height_right;
    balanced_height(tree.left(), bal_left, height_left);
    balanced_height(tree.right(), bal_right, height_right);
    balanced = bal_left && bal_right && abs(height_left - height_right) ⩽ 1;
    height = 1 + std::max(height_left, height_right);
  }
}
```

Función balanced

```
template<typename T>
bool balanced(const BinTree<T> &tree) {
  bool balanced;
  int height;
  balanced_height(tree, balanced, height);
  return balanced;
}
```

Moraleja

- La mayoría de las funciones que operan sobre árboles son recursivas.
- En muchos casos estas funciones deben devolver valores auxiliares adicionales para evitar costes en tiempo elevados.