# Algoritmos y Estructuras de Datos

## Ejercicios sobre Arboles

**Lars-Åke Fredlund**

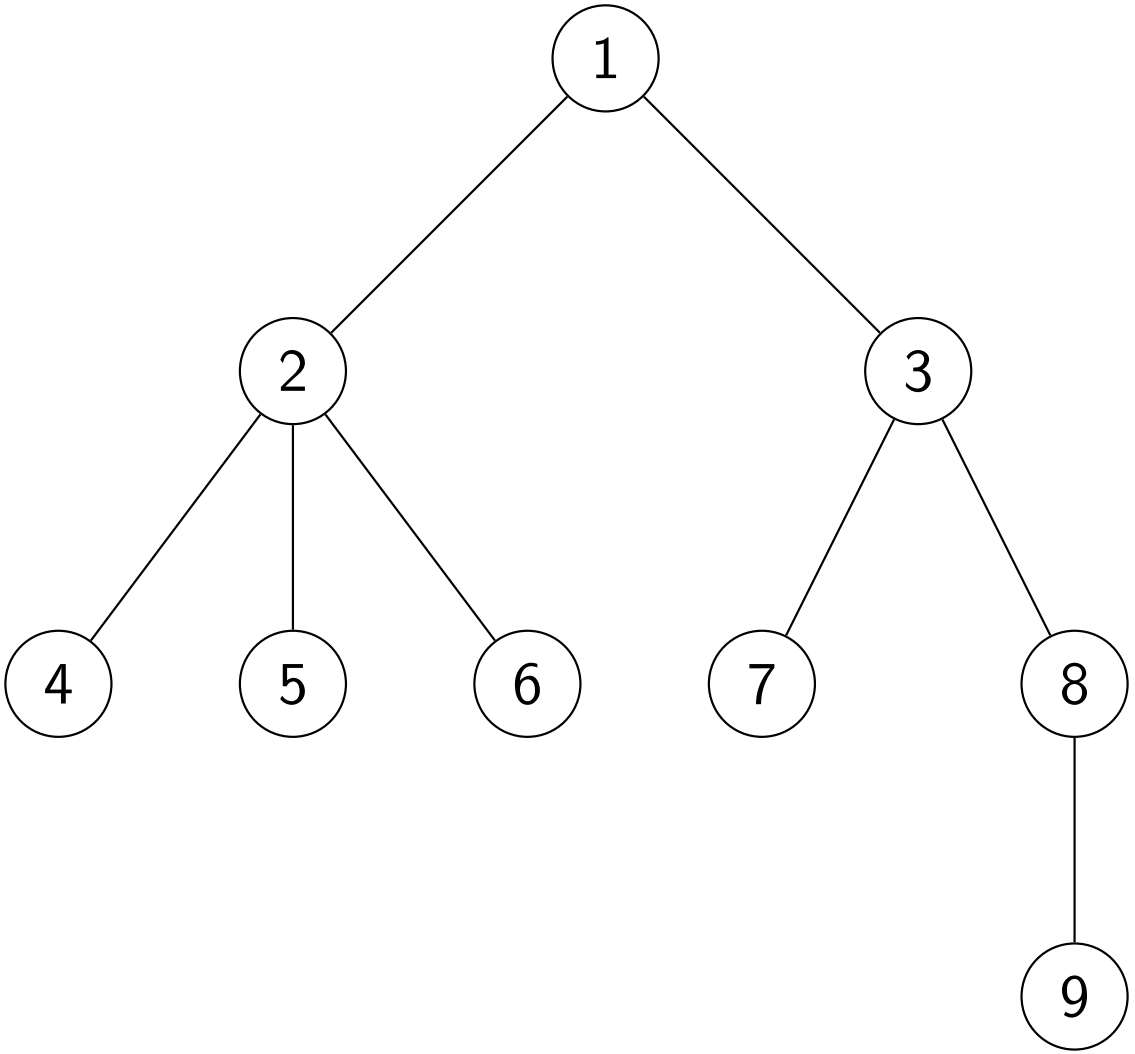
**lfredlund@fi.upm.es**

Universidad Politécnica de Madrid

Curso 2021/2022

# Ejercicios con arbole generales: 1

Construye un arbol $t_{ex}$:




Experimenta con llamar ambos a `addChildXXX` y `insertSiblingXXX`

# Ejercicios 2:

1. Implementa un método `printPreOrden(Tree<E> t)` que imprime los nodos de $t$ en *pre-orden*. Prueba el método con $t_{ex}$.
2. Implementa un método `printPostOrden(Tree<E> t)` que imprime los nodos de $t$ en *post-orden*. Prueba el método con $t_{ex}$.
3. Implementa un método *recursivo* para contar el numero de nodos en un arbol: `int sizeRec(Tree<E> t)`. No se puede llamar `t.size()`.
4. Implementa un método *no recursivo* para contar el numero de nodos en un arbol: `int sizeNoRec(Tree<E> t)`. No se puede llamar `t.size()`.
5. Implementa un método para imprimir el camino que llega a un nodo desde la raiz: `void printPath(Tree<E> t, Position<E> node)`
6. Implementa un método para devolver el camino que llega a un nodo desde la raiz:

```
PositionList<Position<E>> printPath(Tree<E> t,
                                    Position<E> node)
```

# Ejercicios 3:

1. Calcula la maxima profunidad de un arbol:
   `int maxDepth(Tree<E> t)`
2. Cuenta el numero de hojas en un arbol:
   `int numHojas(Tree<E> t)`
3. Calcula la altitura de un nodo:
   `int altitude(Tree<E> t, Position<E> nodo)`
4. Implementa un método
   `boolean equals(Tree<E> t1, Tree<E> t2)` que devuelve true si t1 y t2 son iguales. Se asume que t1 y t2 no son null. Los nodos pueden contener elementos null.

# Ejercicios 4:

1. Dado un arbol t donde los elementos de los nodos son String, y tienen valores "Verde", "Amarillo", "Naranja", "Rojo" o "Vacio", implementa el metodo `boolean esOtono(Tree<String> t)`. El método devuelve true (es otoño!) si el numero de hojas que tienen color rojo, amarillo o naranja son mas que el 75% de las hojas del arbol.
2. *Algo Dificil*: Implementa un método `boolean member(Tree<E> t, E e)` que devuelve true si se encuentra el elemento e dentro el arbol t. *No se puede hacer llamadas recursivas, ni crear estructuras de datos nuevas como pilas, lists, fifos, strings, etc.*
3. Implementa un método `void sumChildren(Tree<Integer> t)` que cambia el elemento de cada nodo padre a ser la suma de los elementos de sus hijos (empezad con las hojas). Prueba con el arbol $t_{ex}$. Imprime el arbol antes y despues.

# Arboles binarios

1. `static <E> void show(BinaryTree<E> t)`: imprime los elementos del arbol en orden "inorden" y "pre-orden". Implementación recursiva o no.
2. `static <E> int numHojas(BinaryTree<E> t)`: calcula el numero de hojas en el arbol.
3. `static <E> int height(BinaryTree<E> t, Position<E> p)`: calcula la altitura del subarbol definido por la posicion `p`.
4. `static <E> int maxDifferenceHeight(BinaryTree<E> t)`: devuelve la maxima differencia en altitura entre dos subarboles en el arbol `t`.
5. Implementa un arbol binario de busqueda. Implementa las operaciones de buscar (primero), insertar (segundo) y borrar (tercero, si hay tiempo).