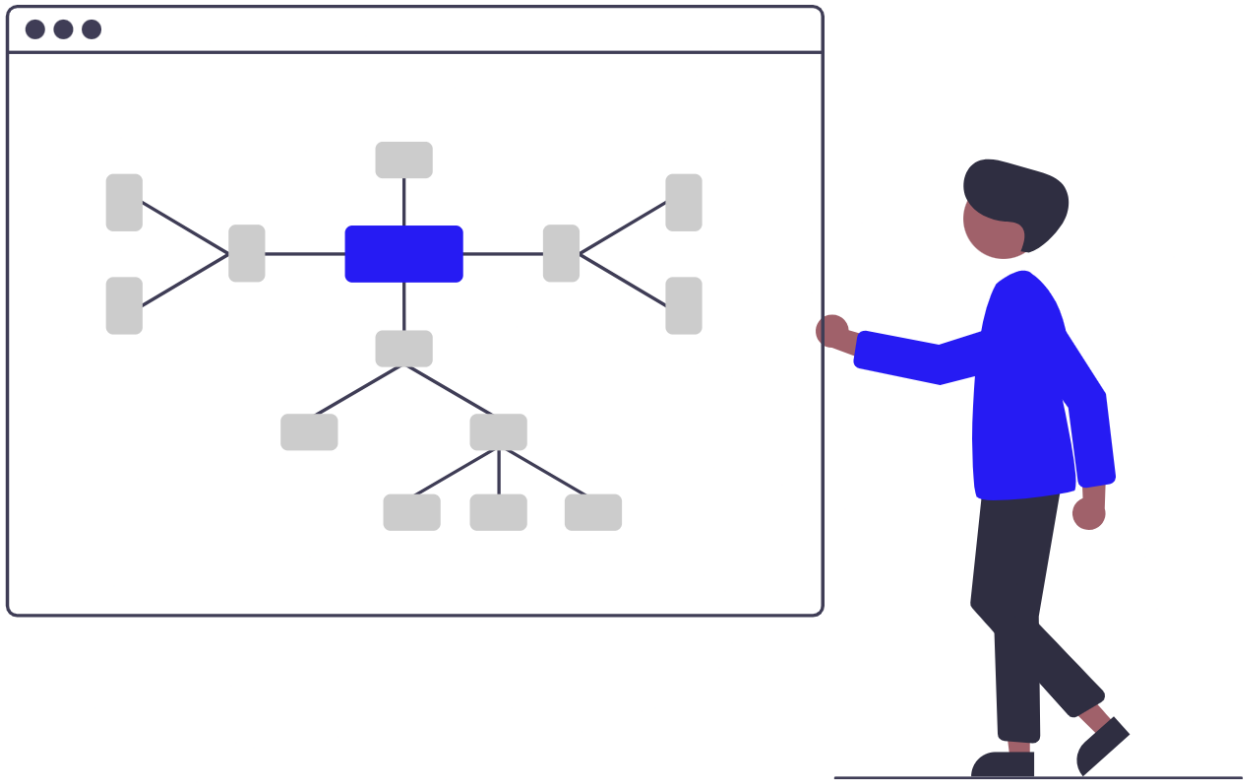
Arboles Binarios Parte II



Equipo 2

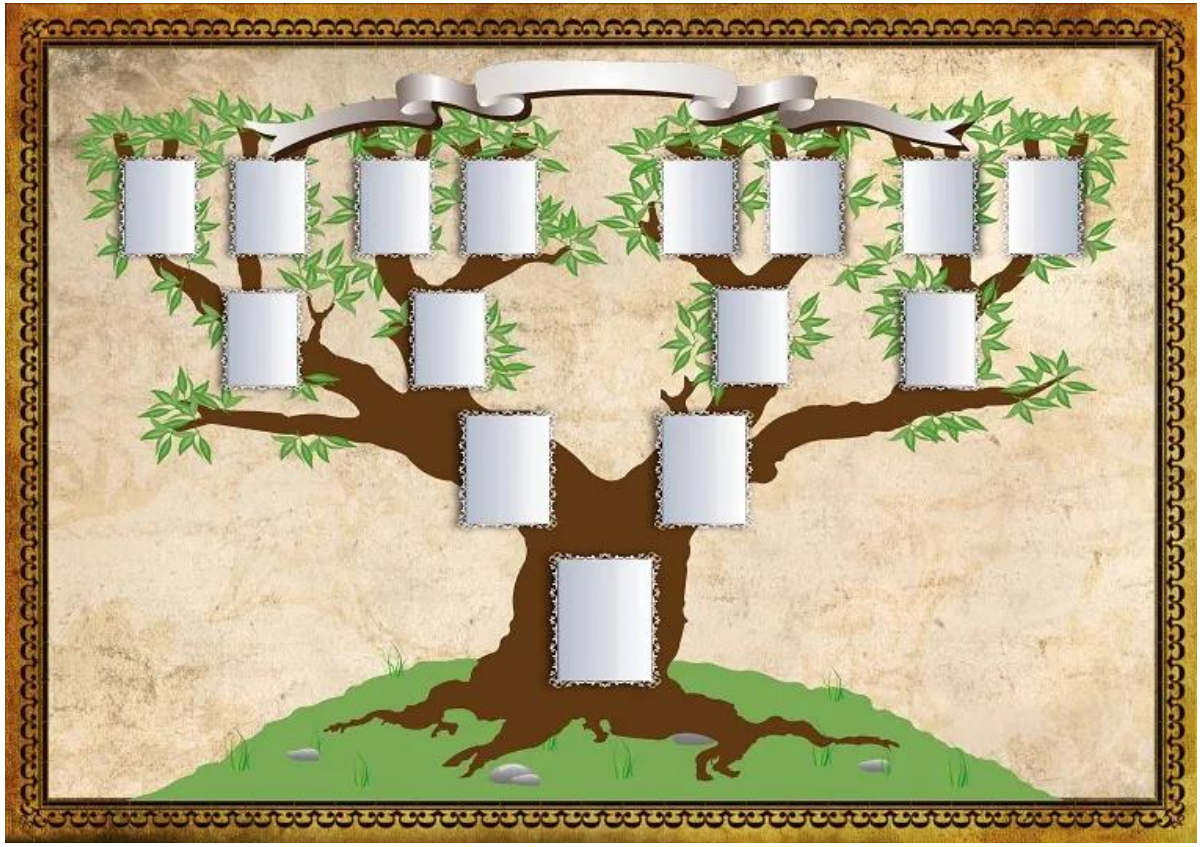
- Alfonso Ruiz
- Manuel García
- Danahe Guillen

¿Que es un árbol?

Estructura en la que **los datos se organizan** de modo que los elementos de información están **relacionados a través de ramas**.

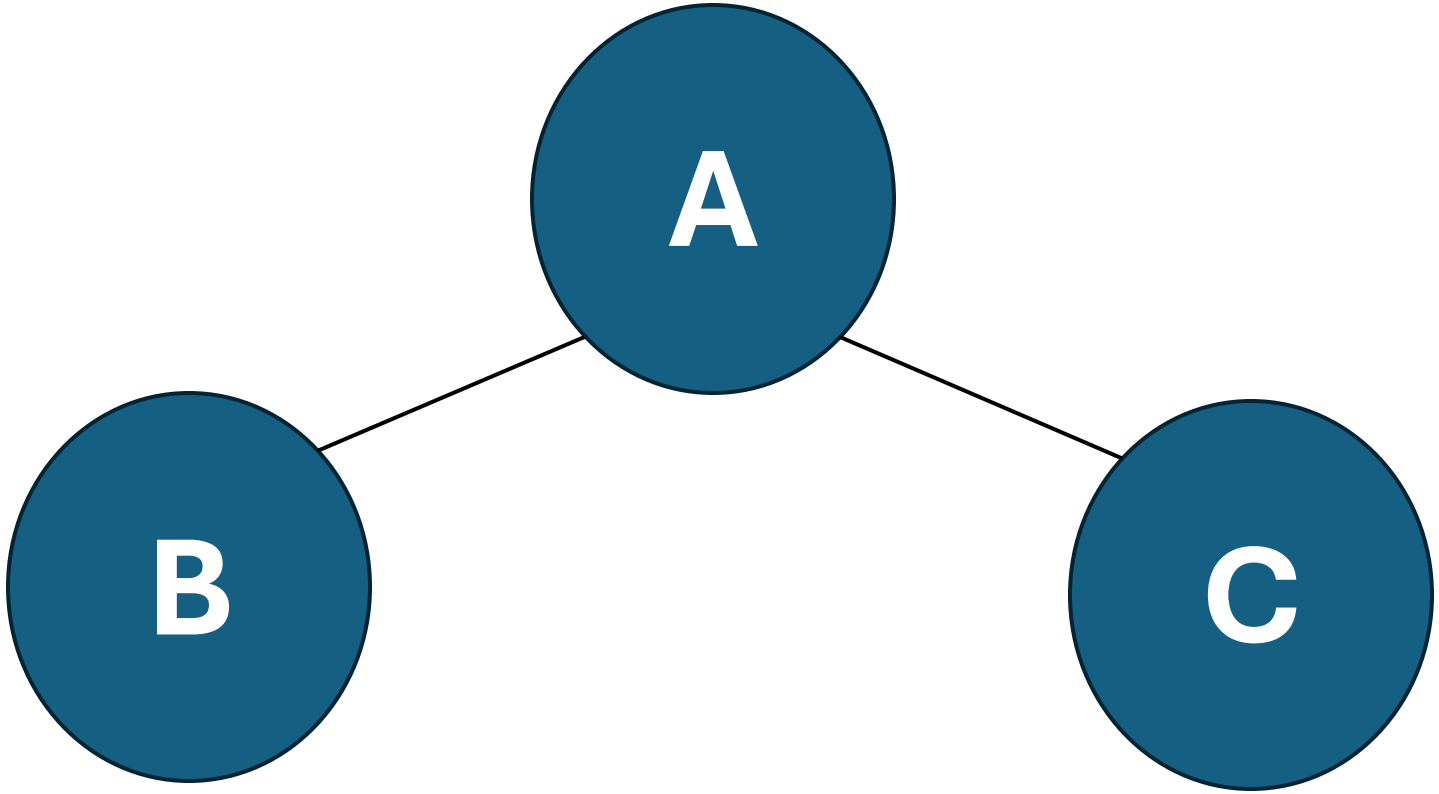
Componentes

1. Nodos
2. Ramas

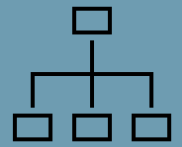


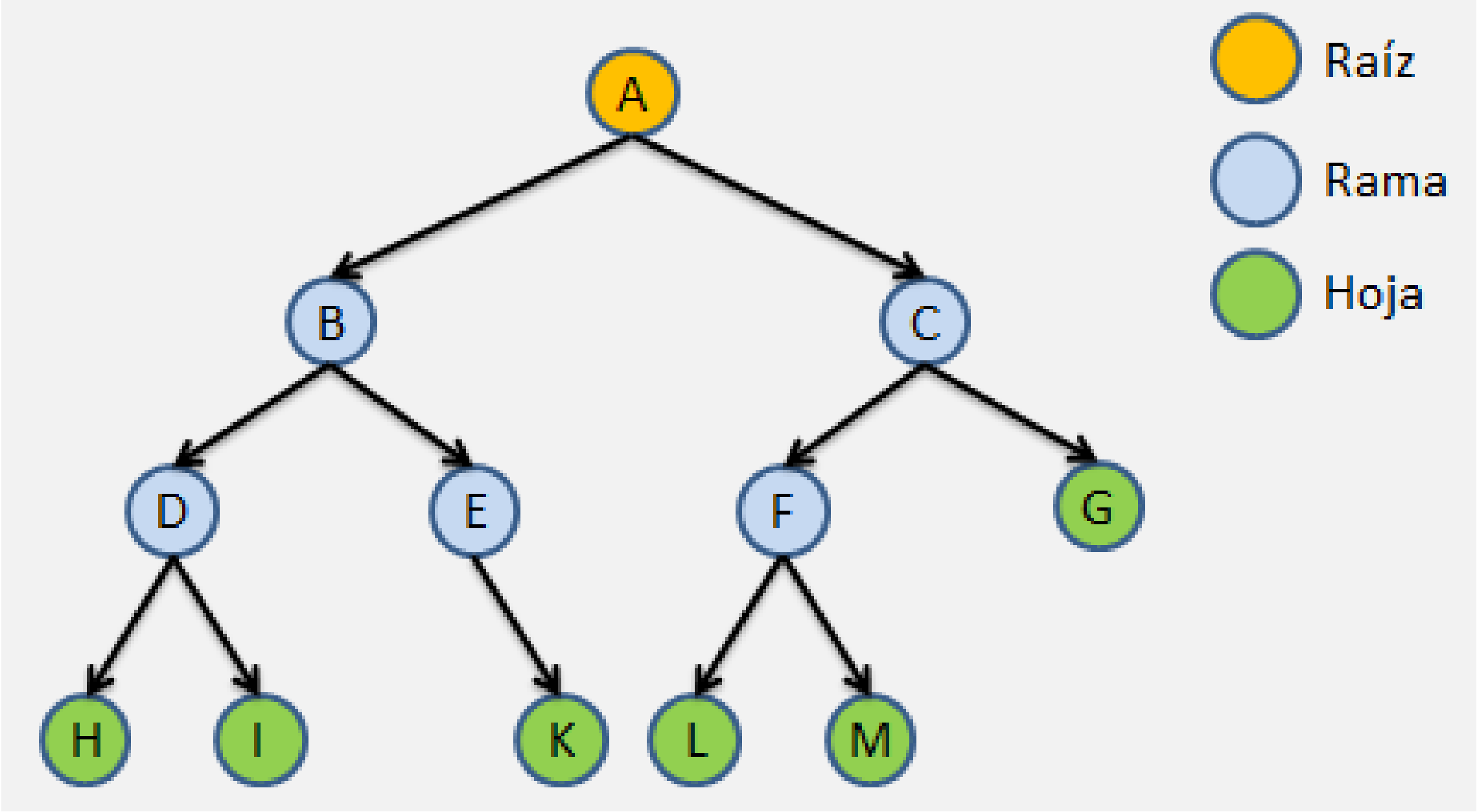
Lo definimos como:

Un conjunto finito de elementos, llamados nodos y un conjunto finito de líneas dirigidas, denominadas ramas, que conectan nodos.



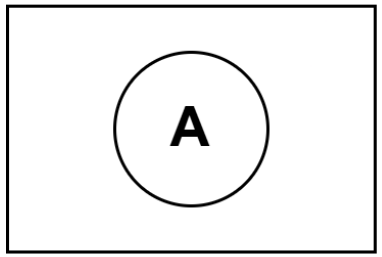
TERMINOLOGIA

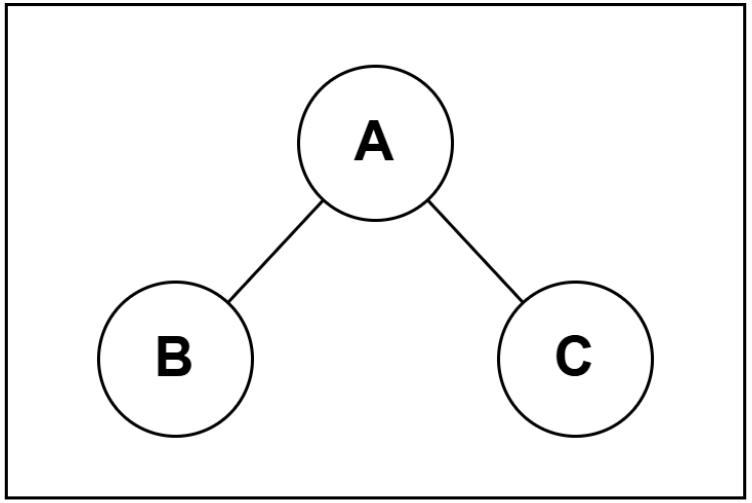
RAIZ	HIJOS	HOJA	CLAVE
Primer nodo 	Nodos sucesores de la raíz 	Nodo sin hijos 	Dato que se desea manipular. 

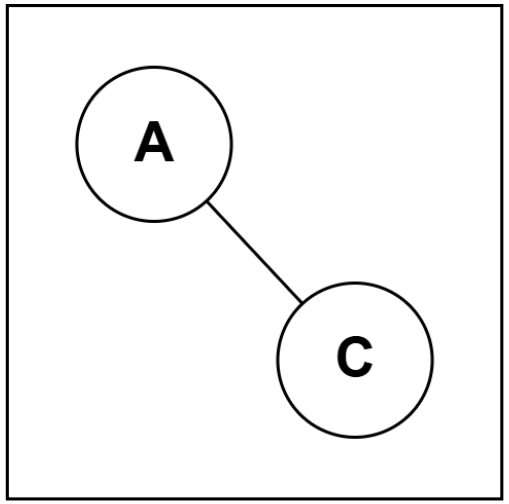

A
B
C
D
E
F
G
H
I
K
L
M
Raíz
Rama
Hoja

ARBOL BINARIO

DEF. Es un árbol en el que *ningún nodo puede tener más de dos subárboles*. Cada nodo puede tener, **cero** , **uno** o **dos hijos** (*subárboles*). Se conoce el nodo de la izquierda como ***hijo izquierdo*** y el nodo de la derecha como ***hijo derecho***.







Componentes

1. Nodo raíz
2. Subárbol izquierdo de R
3. Subárbol derecho de R

Estructura

Su estructura se construye con nodos. Cada nodo debe contener el campo dato *(datos a almacenar)* y dos campos punteros, uno para el subárbol izquierdo y otro al subárbol derecho, conocidos como **puntero izquierdo** y **puntero derecho** respectivamente. Un valor NULL indica un árbol vacío.

Donde:

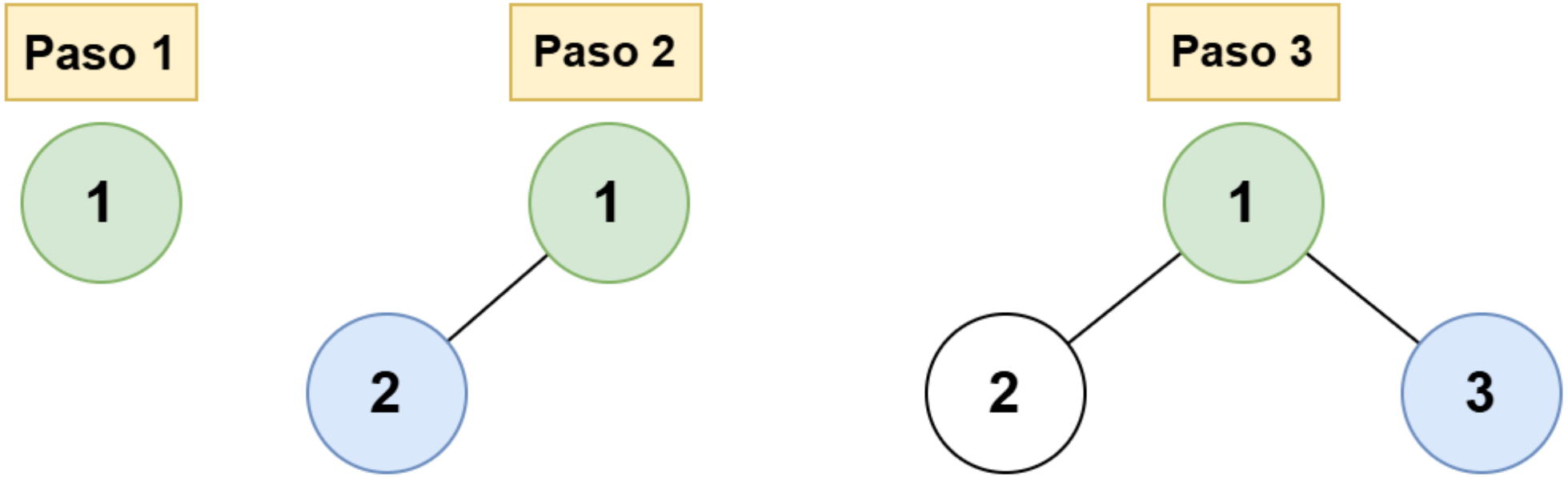
- IZQ: Campo donde se almacena la dirección del subárbol izquierdo del nodo T
- INFO: Representa donde se almacena la información del nodo.
- DER: Almacena la dirección del subárbol derecho del nodo T.

CONSTRUCCION

Su construcción se realiza en consideración de un vector, recorriéndolo de izquierda a derecha. **(Construcción por niveles)**

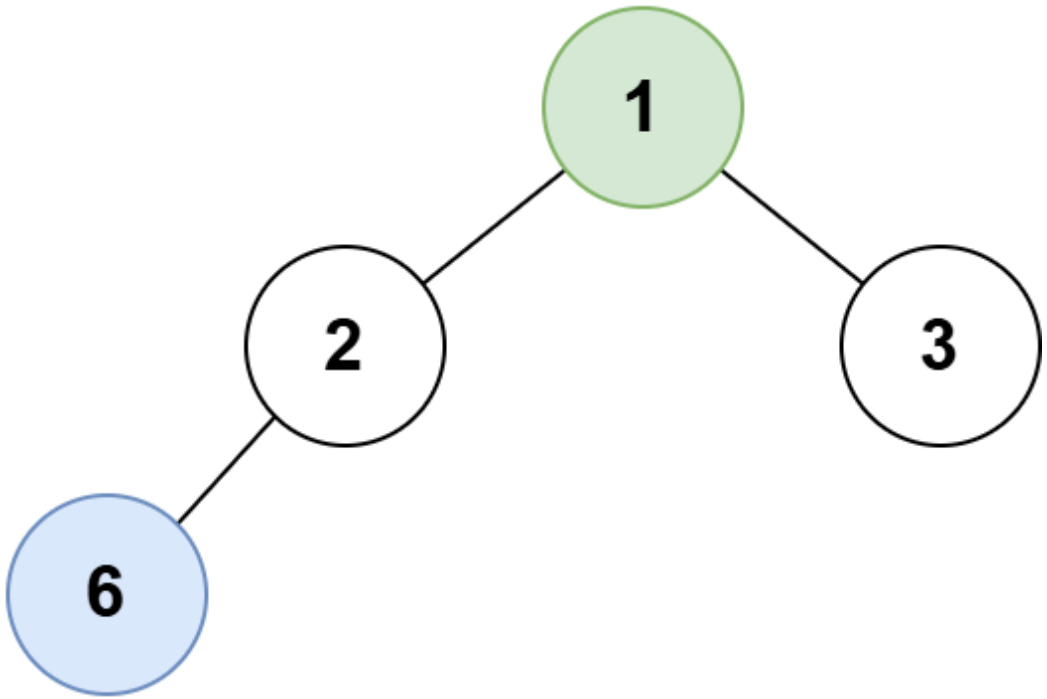
Construya un árbol binario con los siguientes valores:

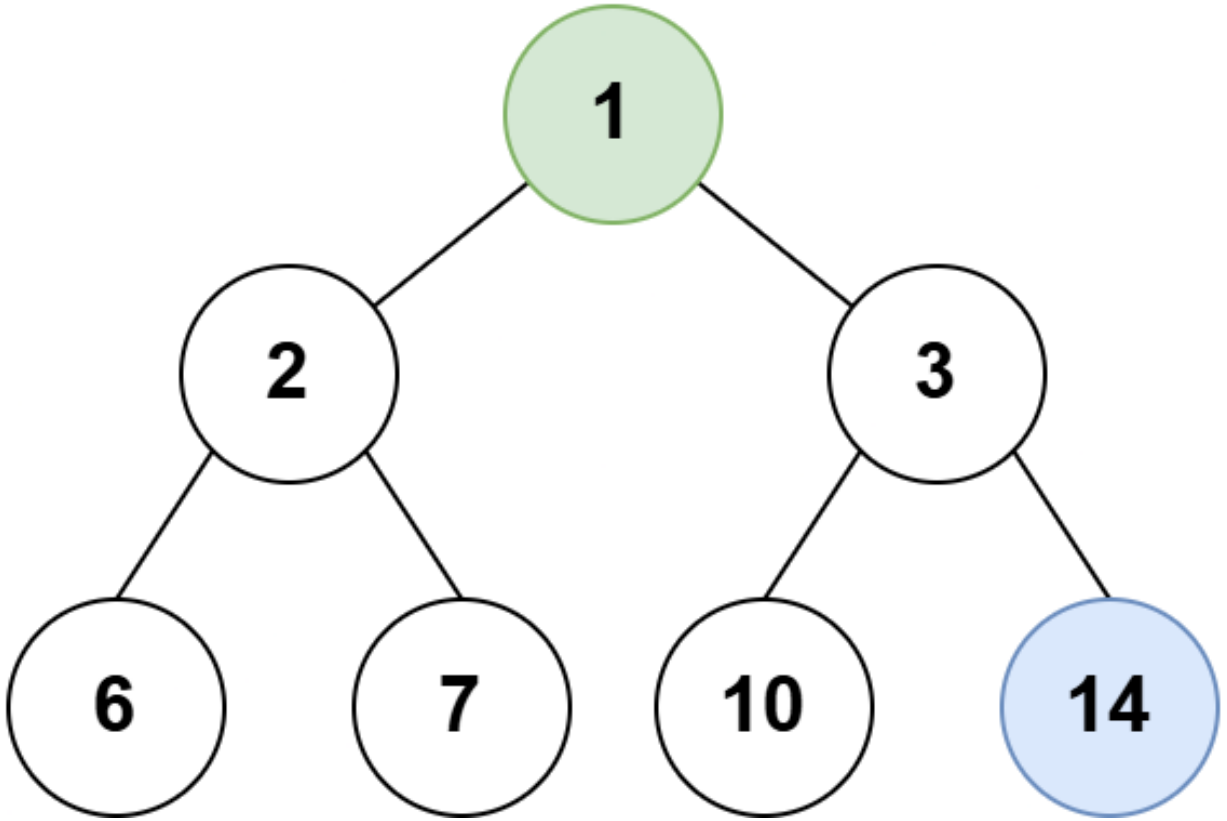
1	2	3	6	7	10	14



1	2	3	6	7	10	14

Paso 4



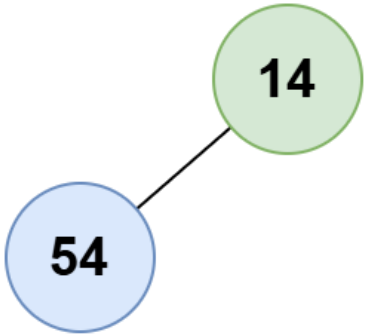


Construya un árbol binario con los siguientes valores:

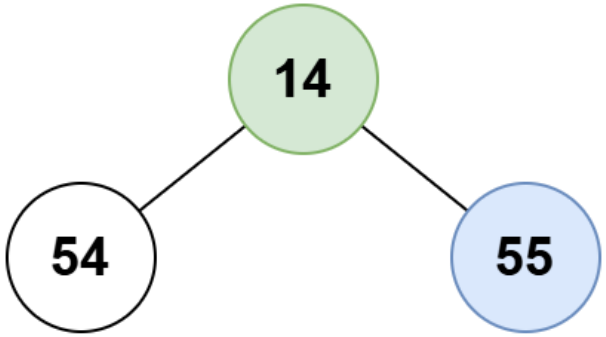
14	54	55			5

Paso 1

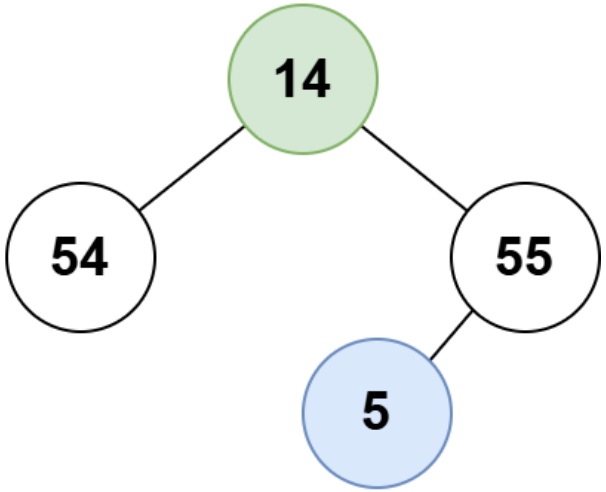
Paso 2

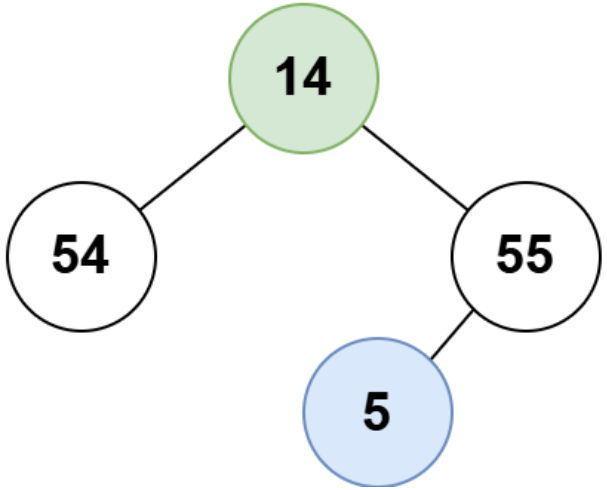


Paso 3



Paso 4





PARTICIPACIONES 

Construya un árbol binario con los siguientes valores:

45	23	12	36	10

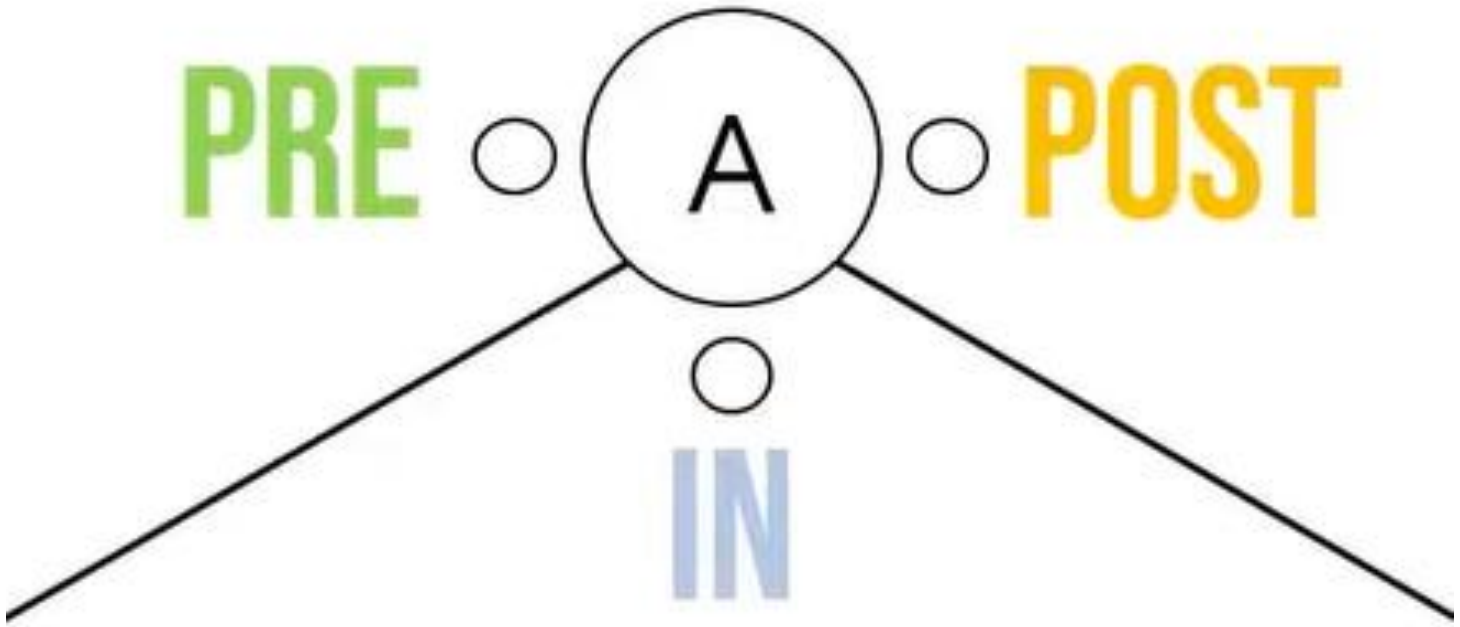
6	4	74	13	43	12	92

RECORRIDOS EN ARBOLES

Se le llama recorrido al orden establecido en el que se debe visitar los nodos de un árbol.

RECORRIDO	ORDEN	DESCRIPCIÓN
Pre-orden	Raíz → Izquierda → Derecha	Empieza en nodo actual, después izquierdo y finalmente derecho.
In-orden	Izquierda → Raíz → Derecha	Empieza lado izquierdo, después actual y finalmente derecho.
Post-orden	Izquierda → Derecha → Raíz	Empieza de lado izquierdo, después derecho y finalmente nodo actual **(TERMINA EN NODO RAIZ)**
Por nivel	Se recorren los nodos nivel por nivel de izquierda a derecha, de arriba hacia abajo.	

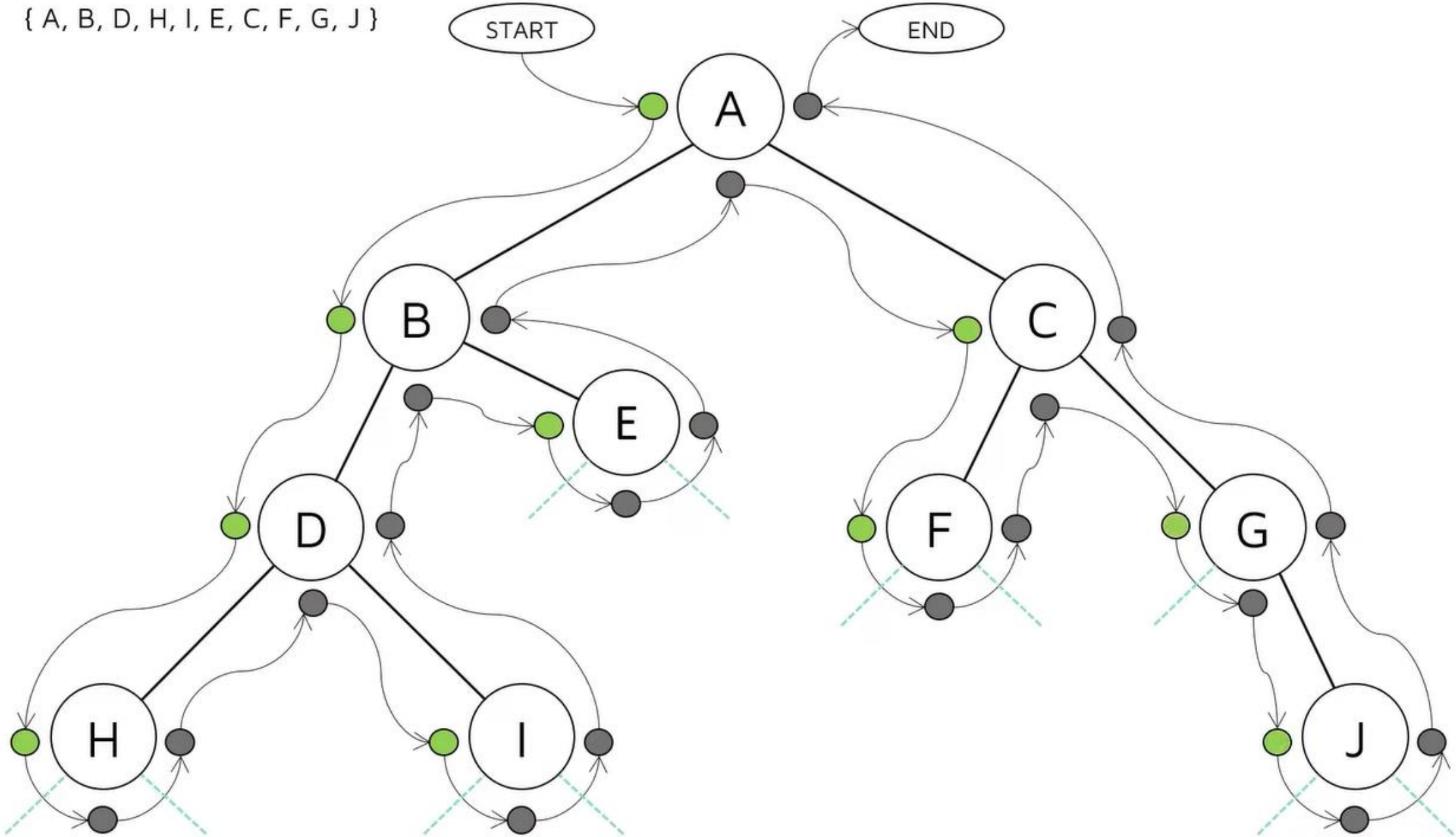
Alternativamente podemos realizar los recorridos estableciendo para cada nodo tres posiciones:



De tal forma que dependiendo del recorrido a realizar activaremos la posición correspondiente. Veamos ejemplos.

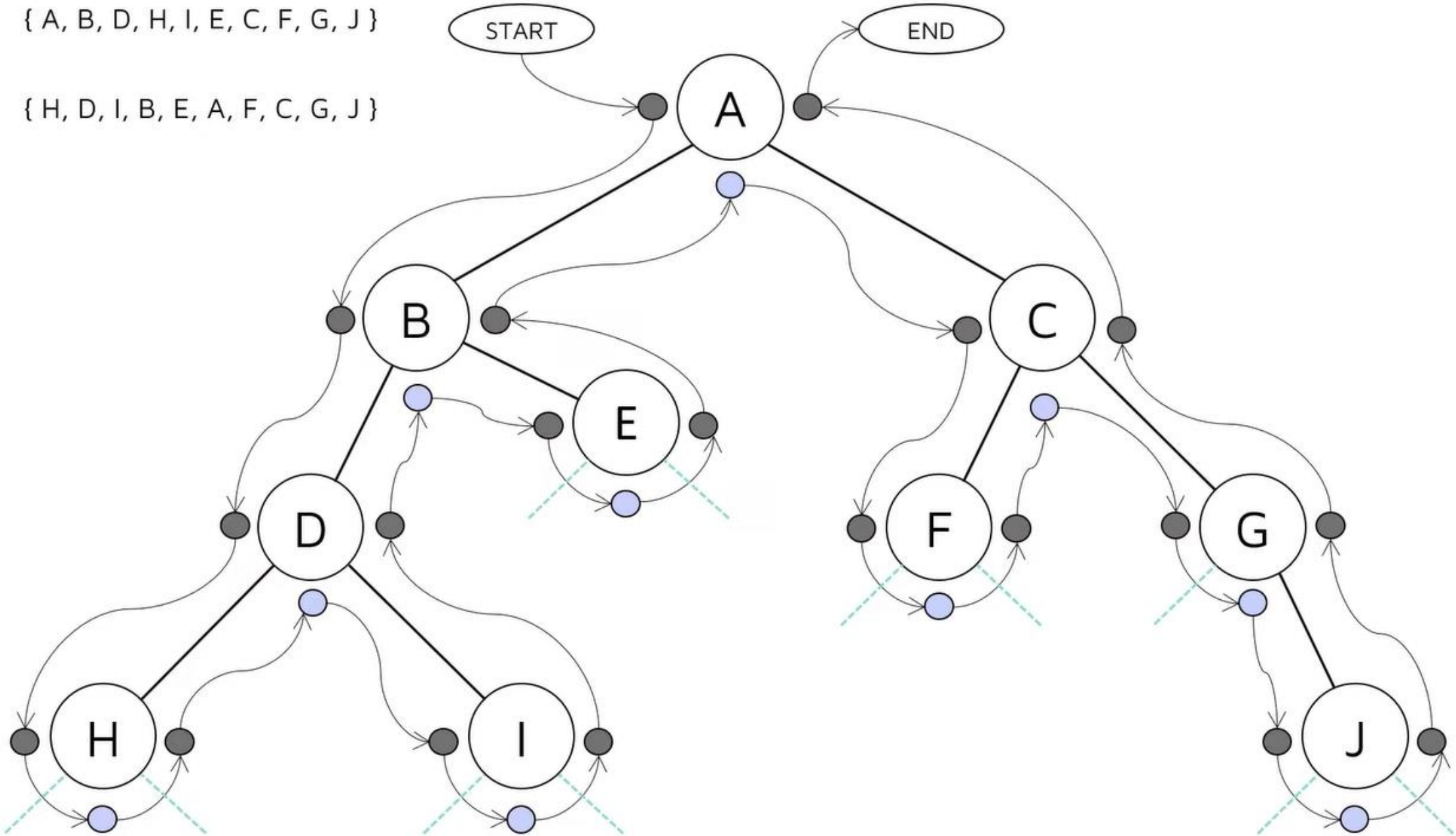
PRE

{ A, B, D, H, I, E, C, F, G, J }



PRE { A, B, D, H, I, E, C, F, G, J }

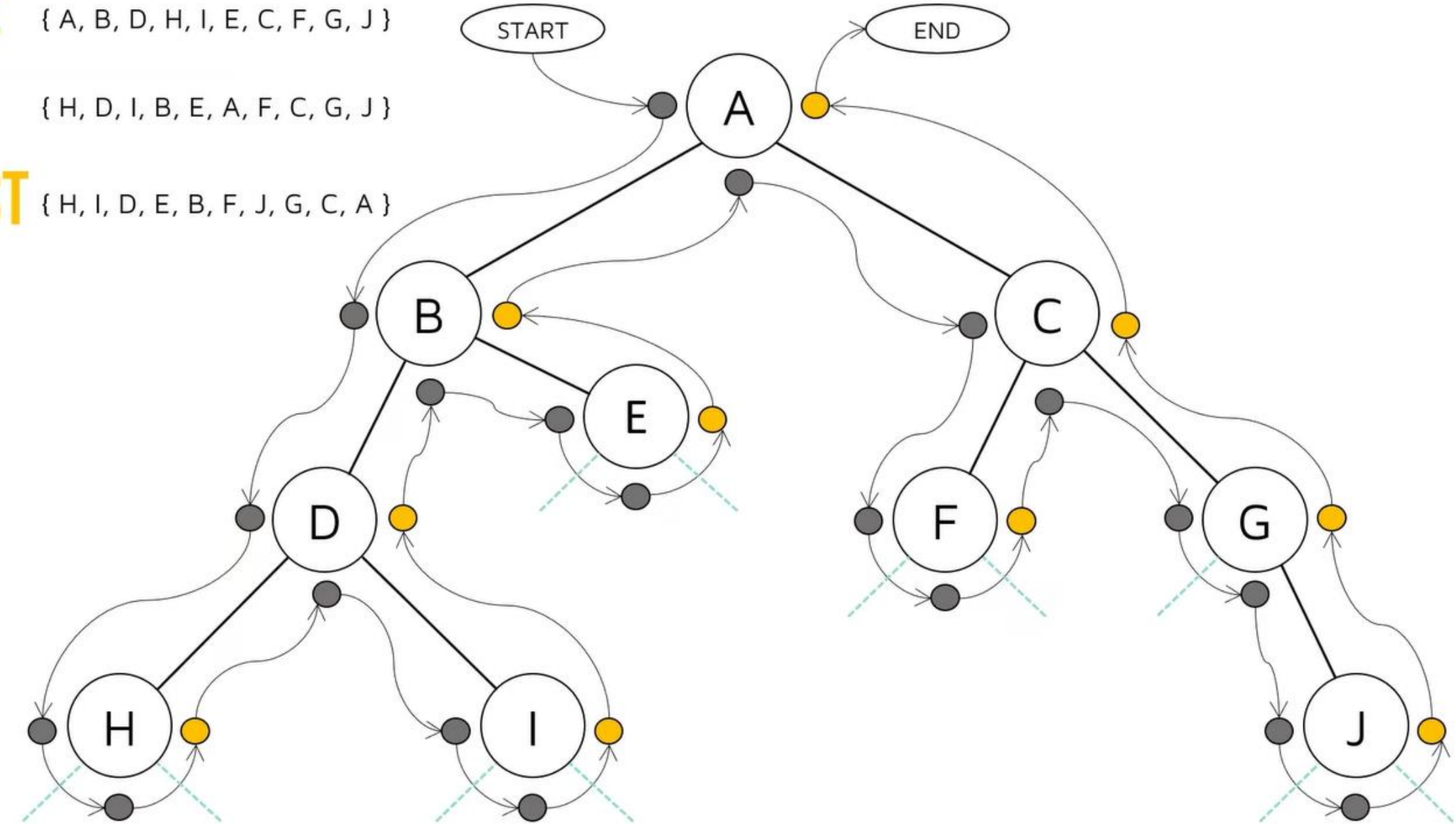
IN { H, D, I, B, E, A, F, C, G, J }

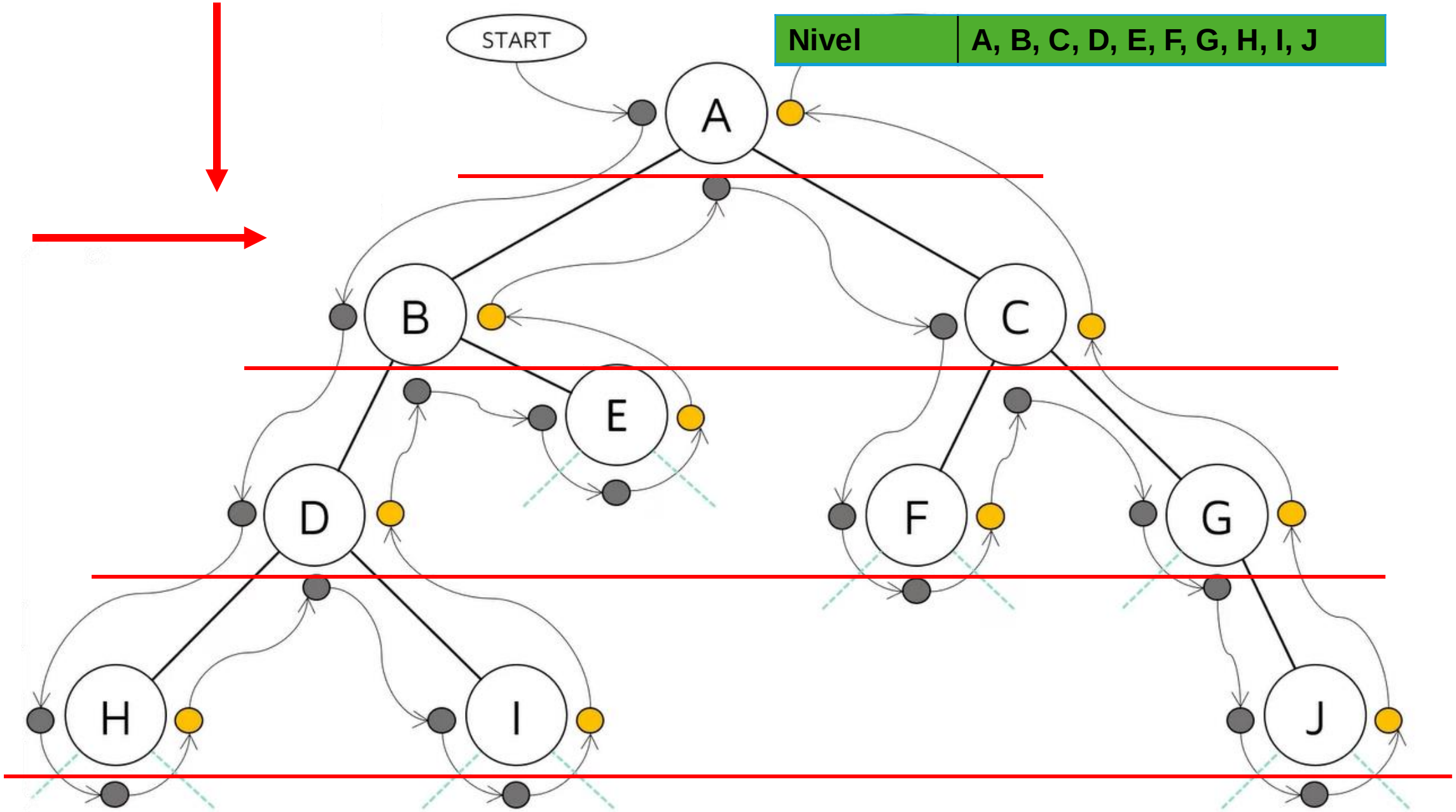


PRE { A, B, D, H, I, E, C, F, G, J }

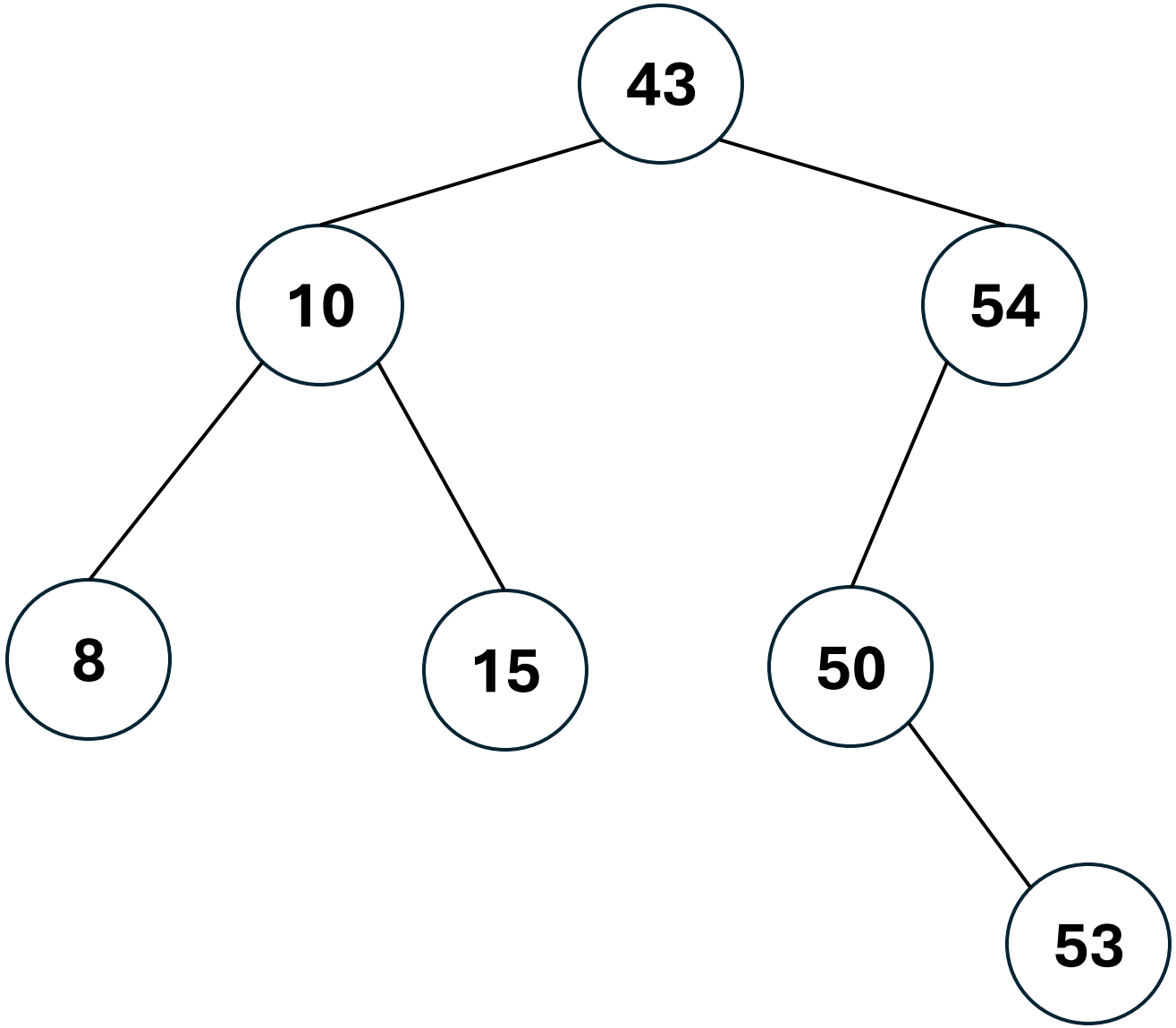
IN { H, D, I, B, E, A, F, C, G, J }

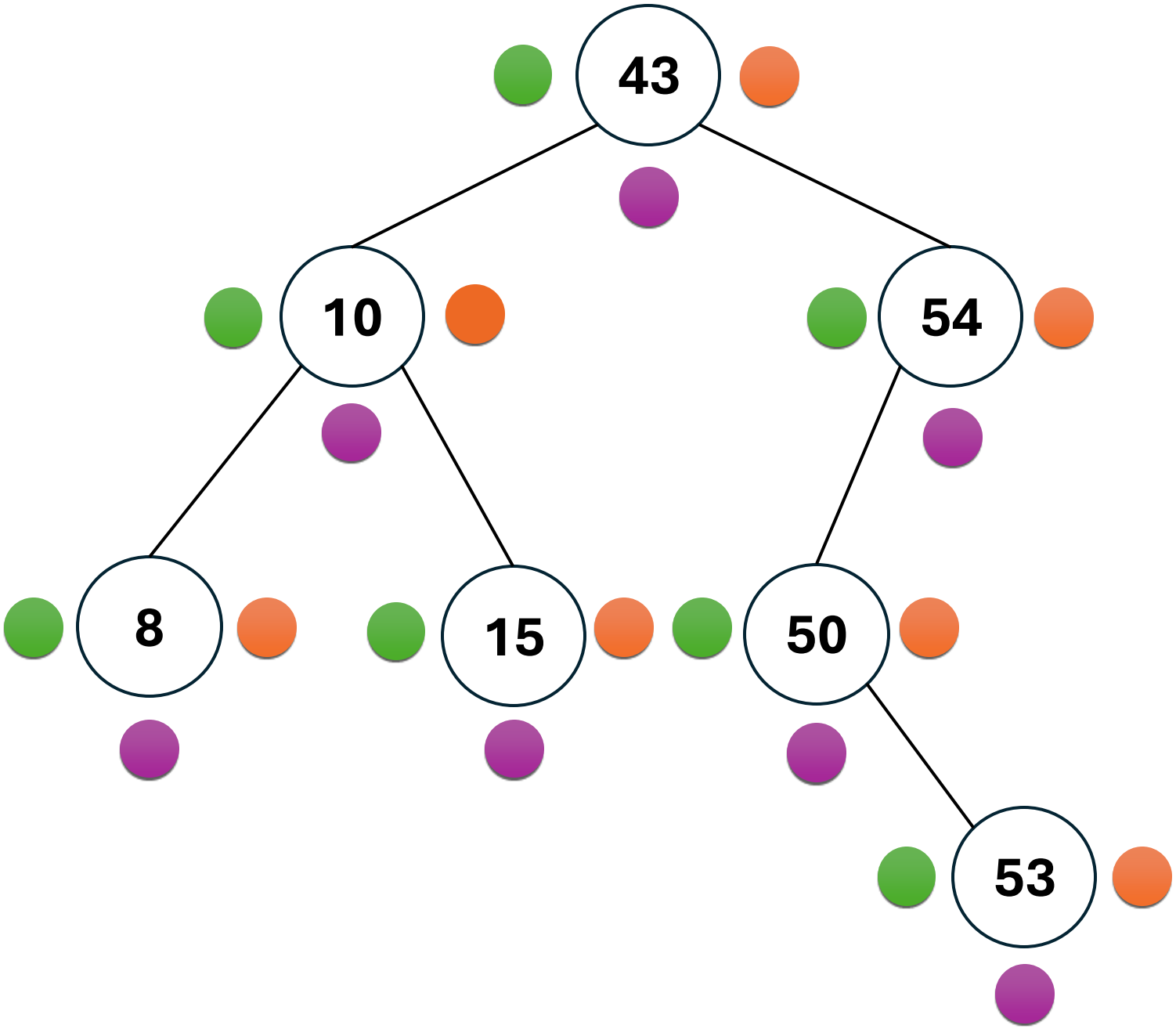
POST { H, I, D, E, B, F, J, G, C, A }



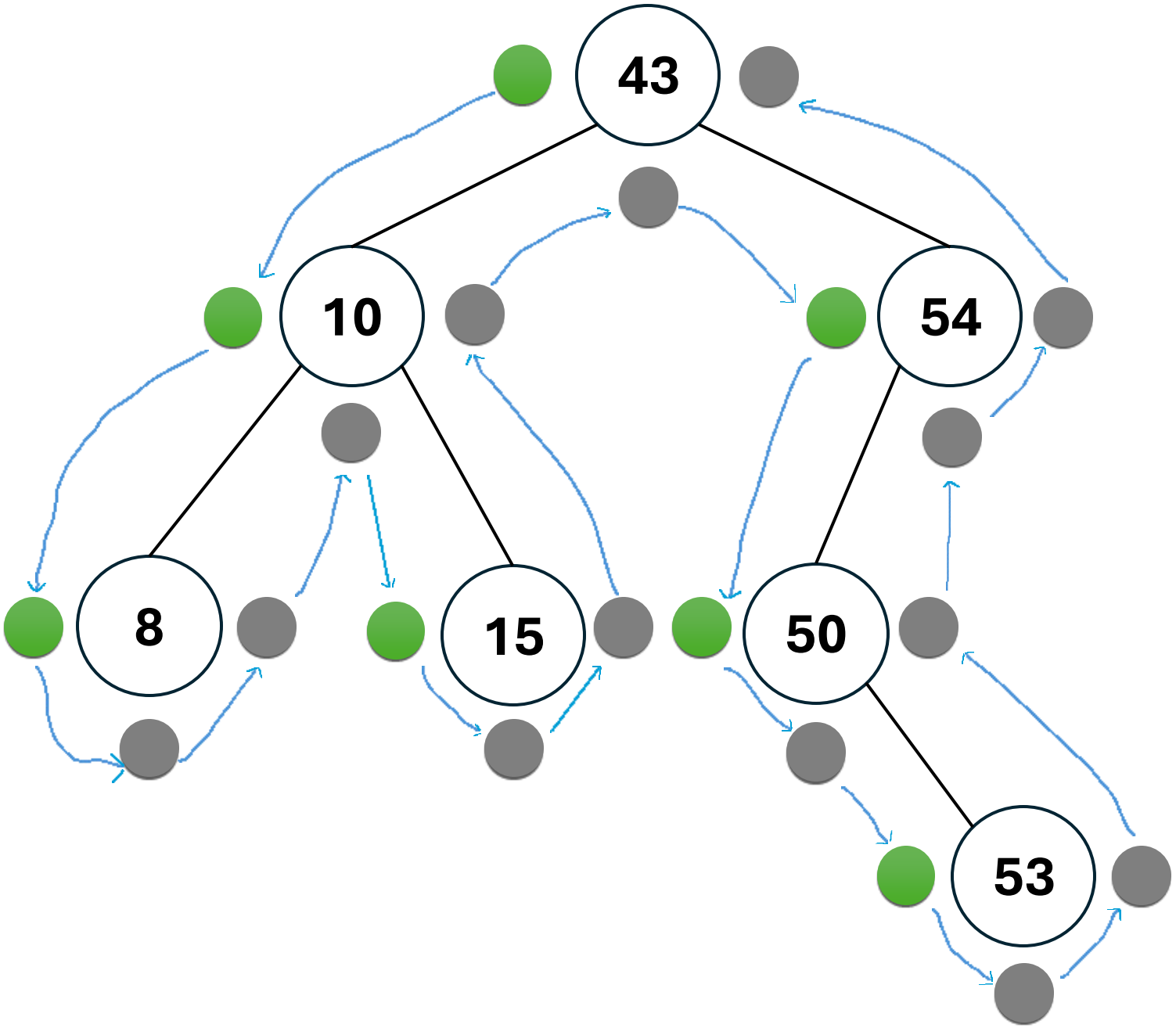

START
A
B
C
E
D
F
G
H
I
J
Nivel
A, B, C, D, E, F, G, H, I, J

Determine los recorridos solicitados.

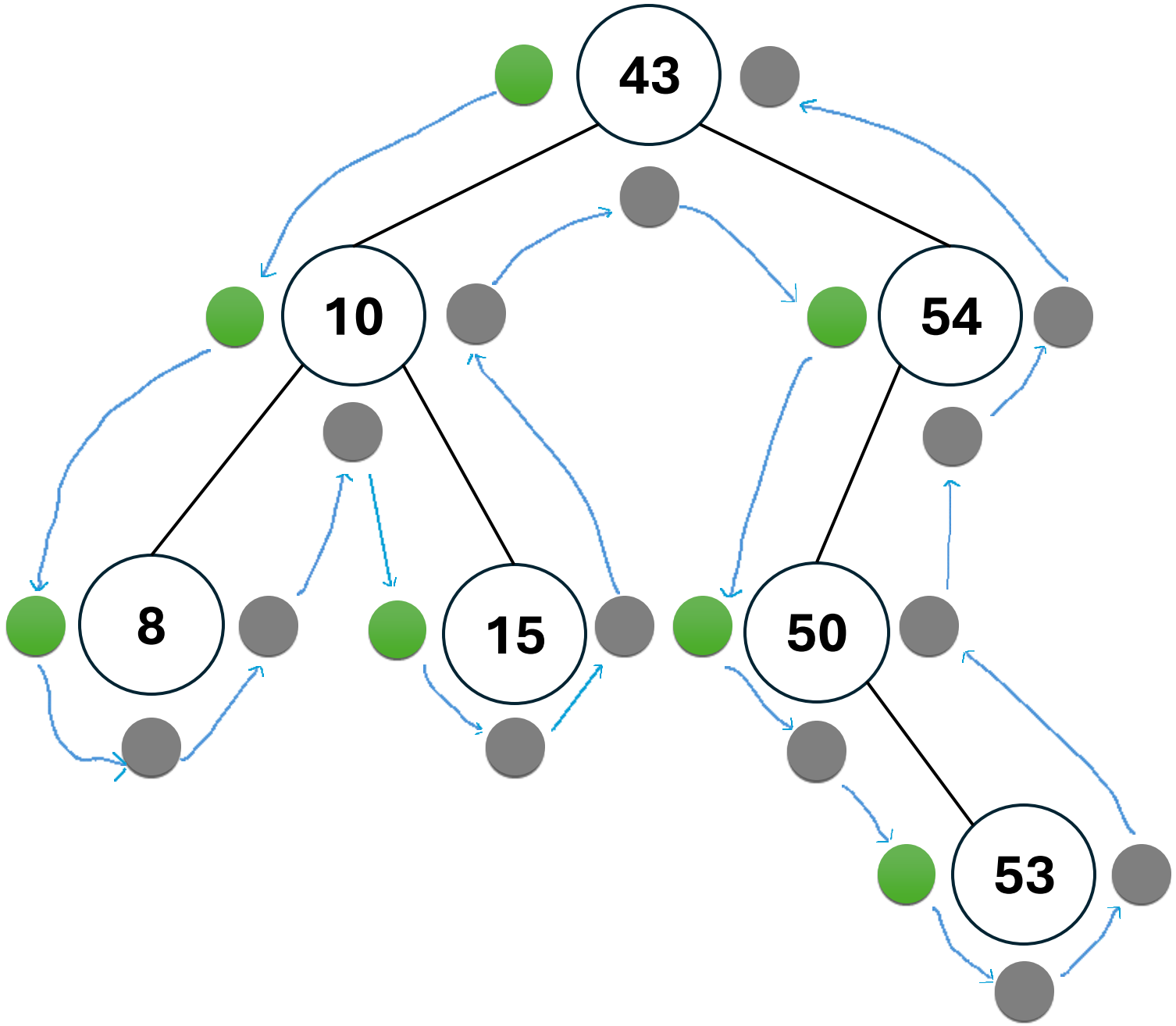




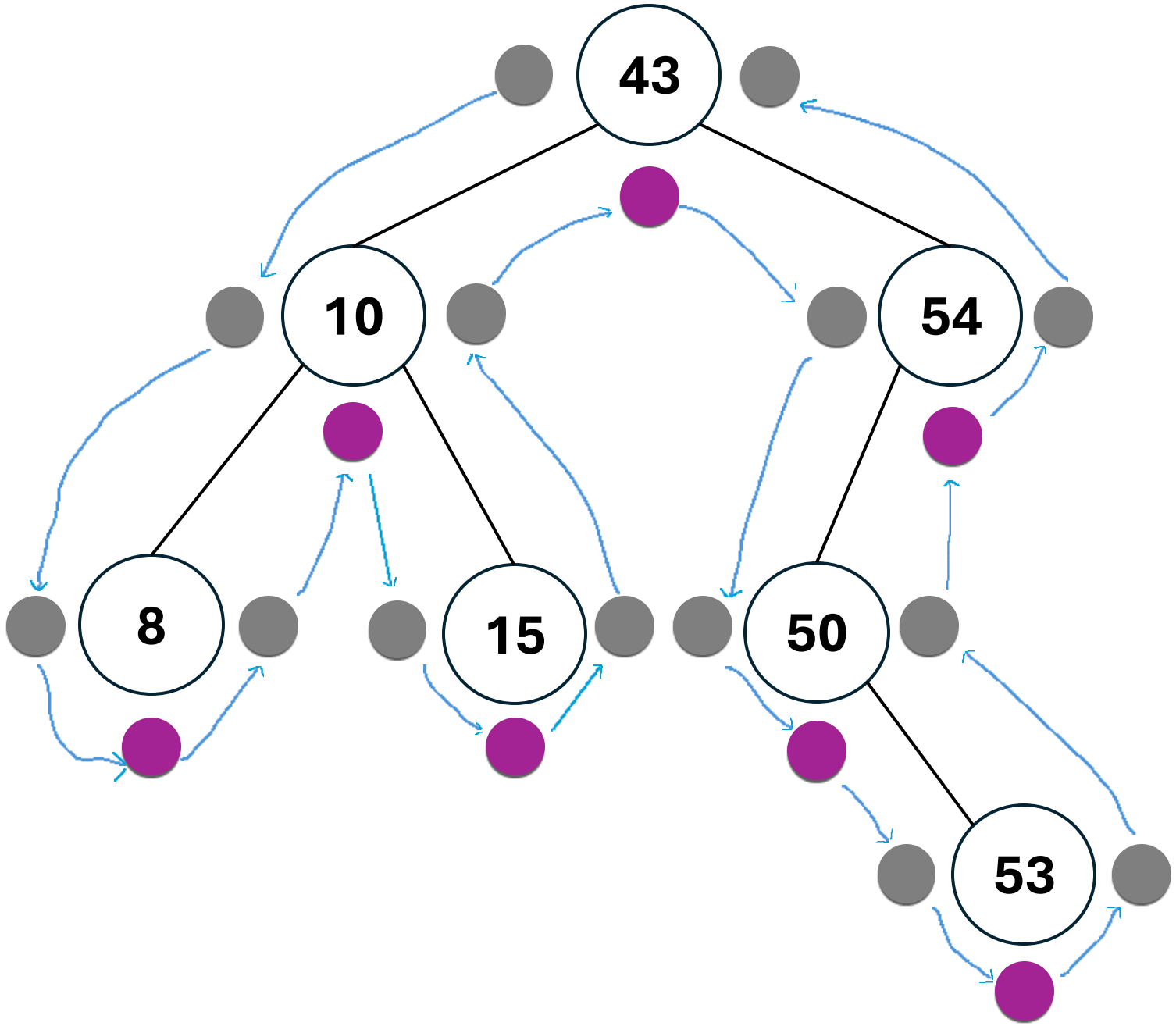
Pre-orden	
In-orden	
Post-orden	



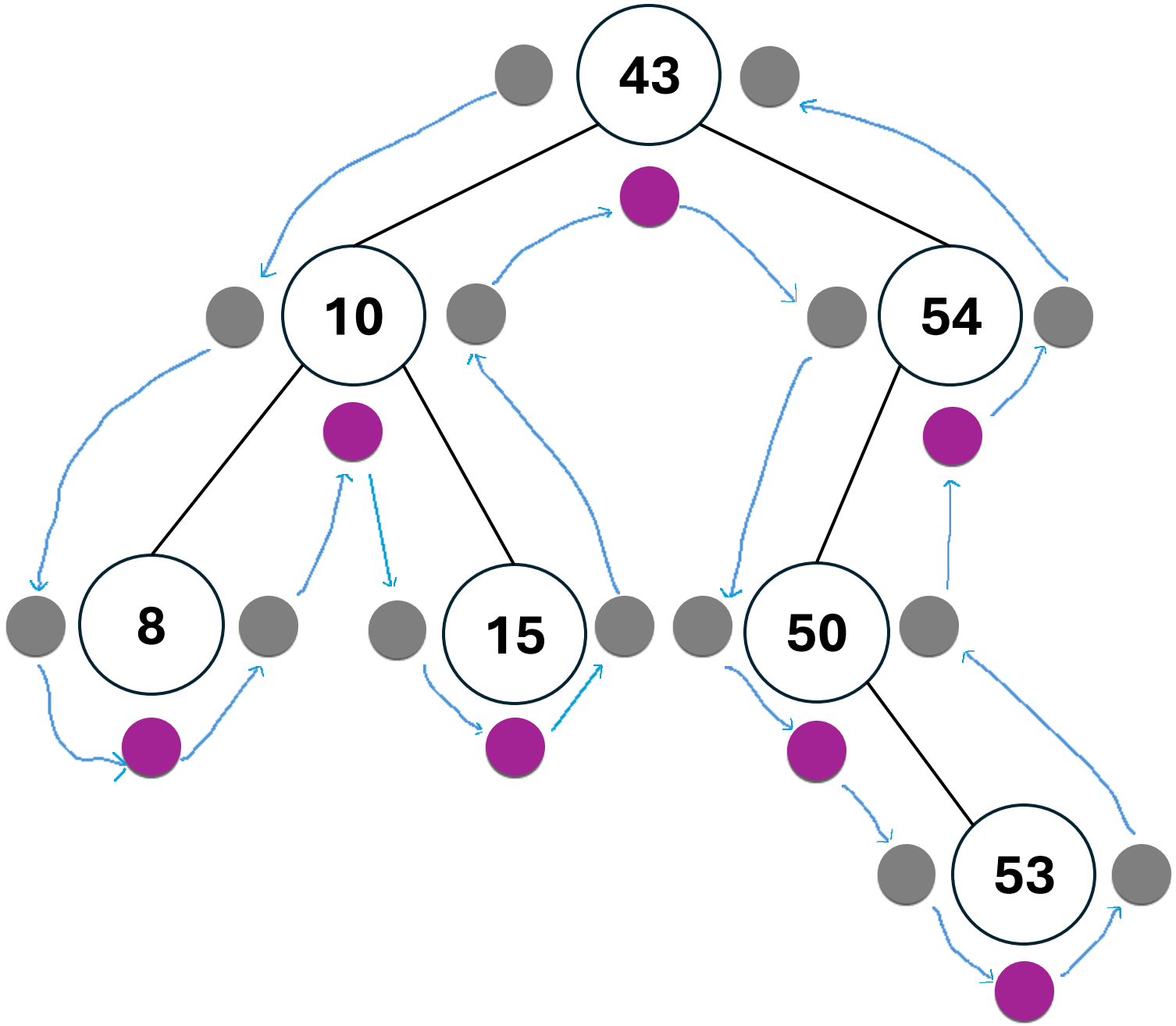
Pre-orden	
In-orden	
Post-orden	



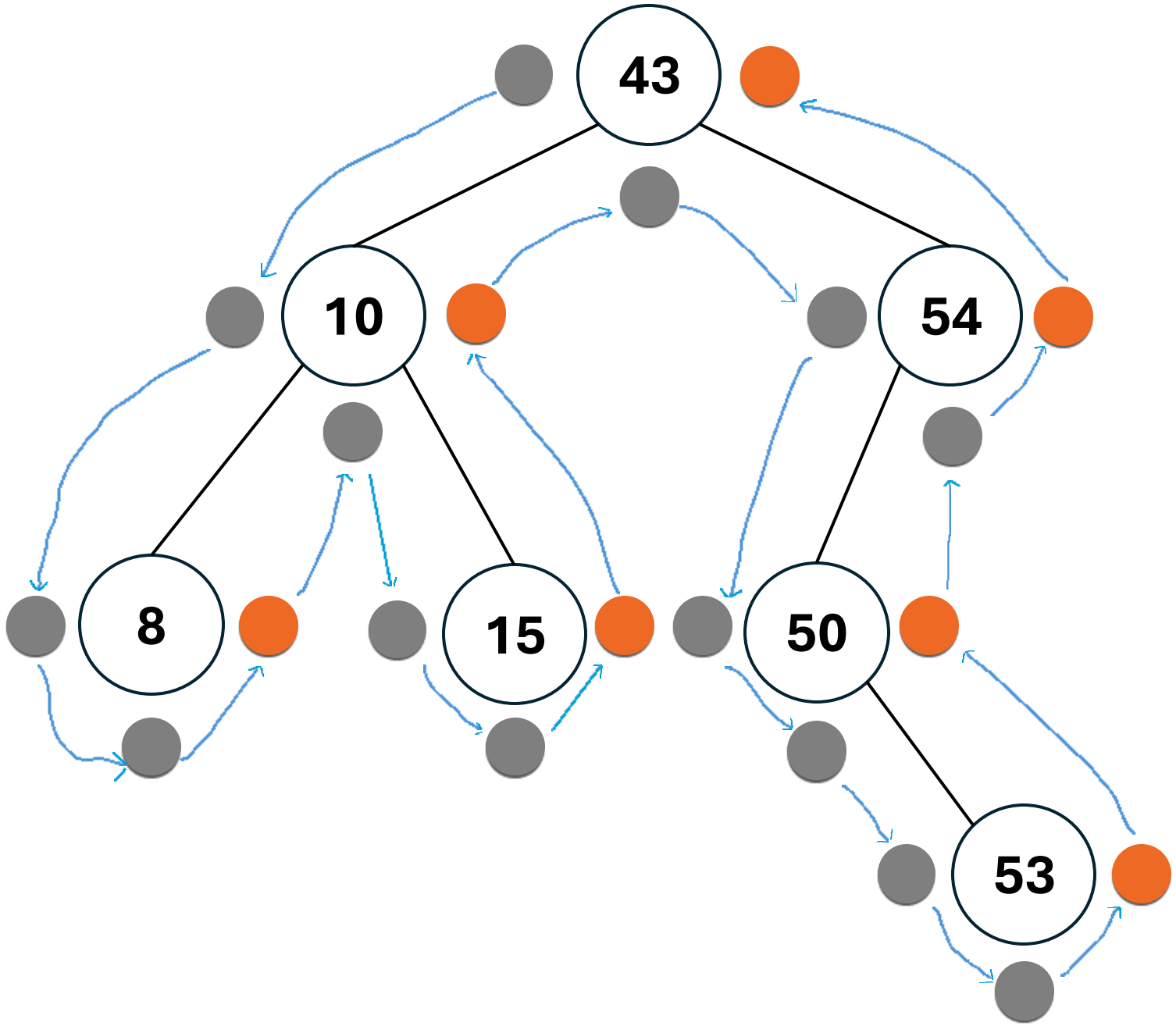
Pre-orden	43, 10, 8, 15, 54, 50, 53
In-orden	
Post-orden	



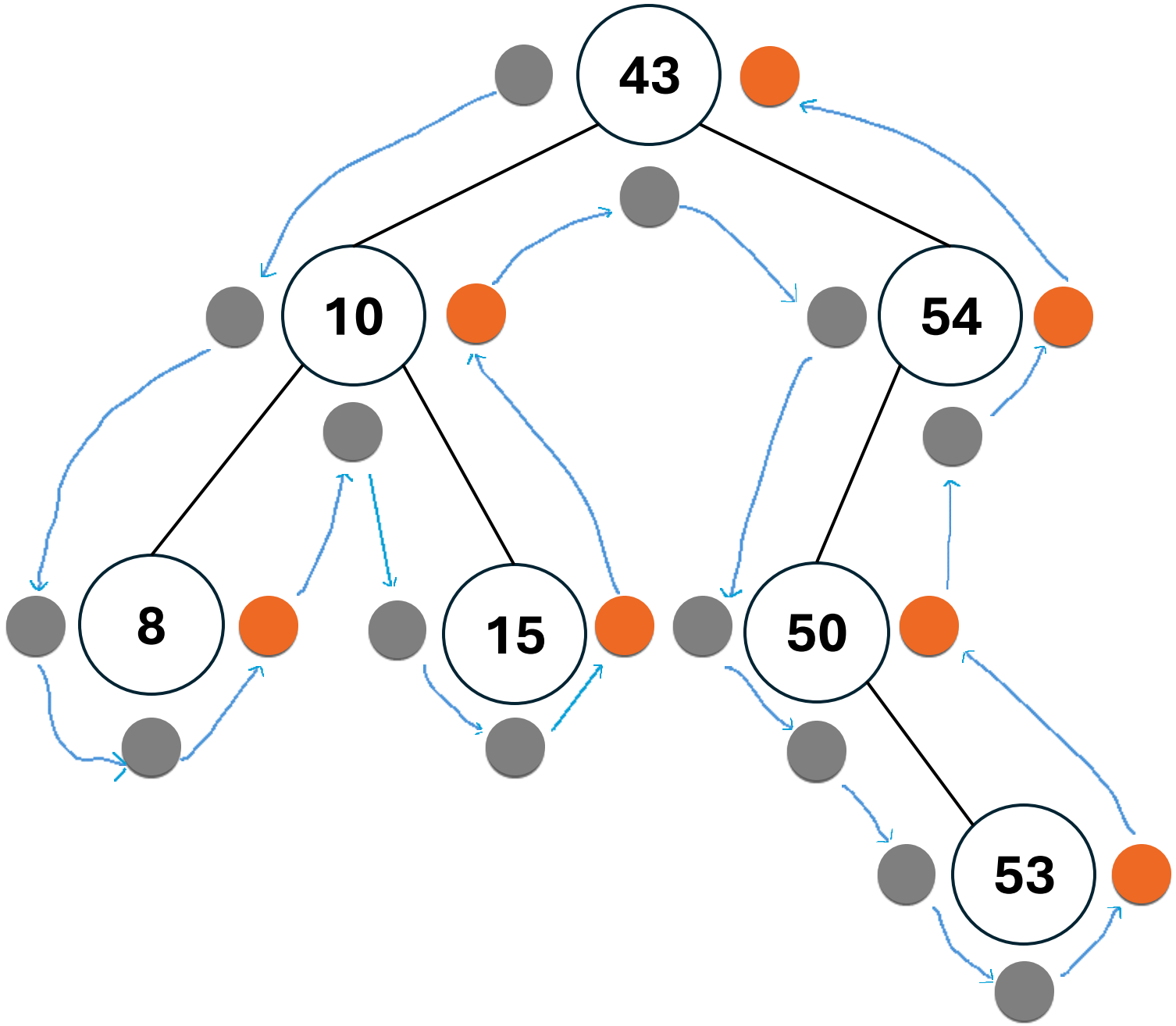
Pre-orden	43, 10, 8, 15, 54, 50, 53
In-orden	
Post-orden	



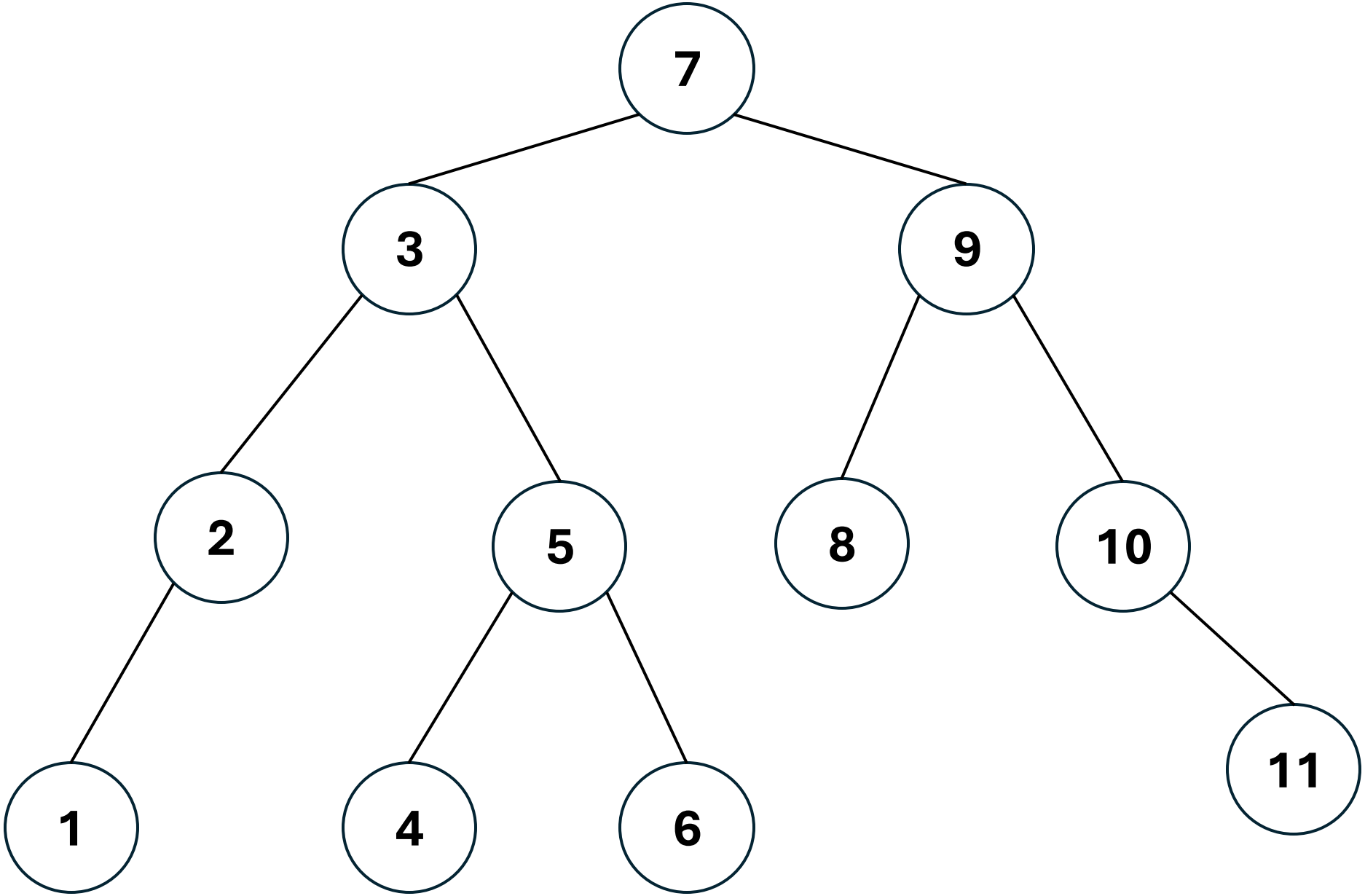
Pre-orden	43, 10, 8, 15, 54, 50, 53
In-orden	8, 10, 15, 43, 50, 53, 54
Post-orden	

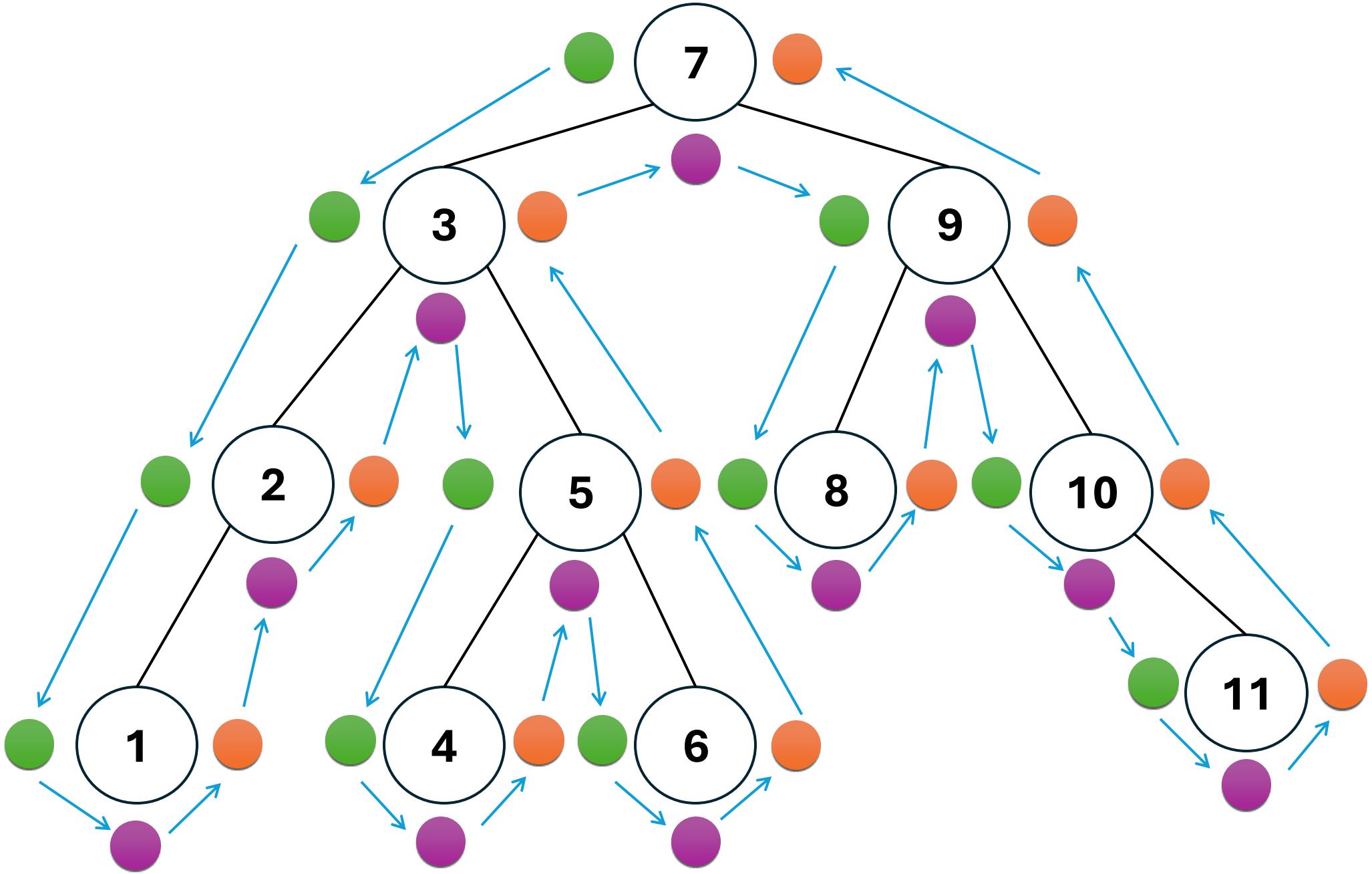


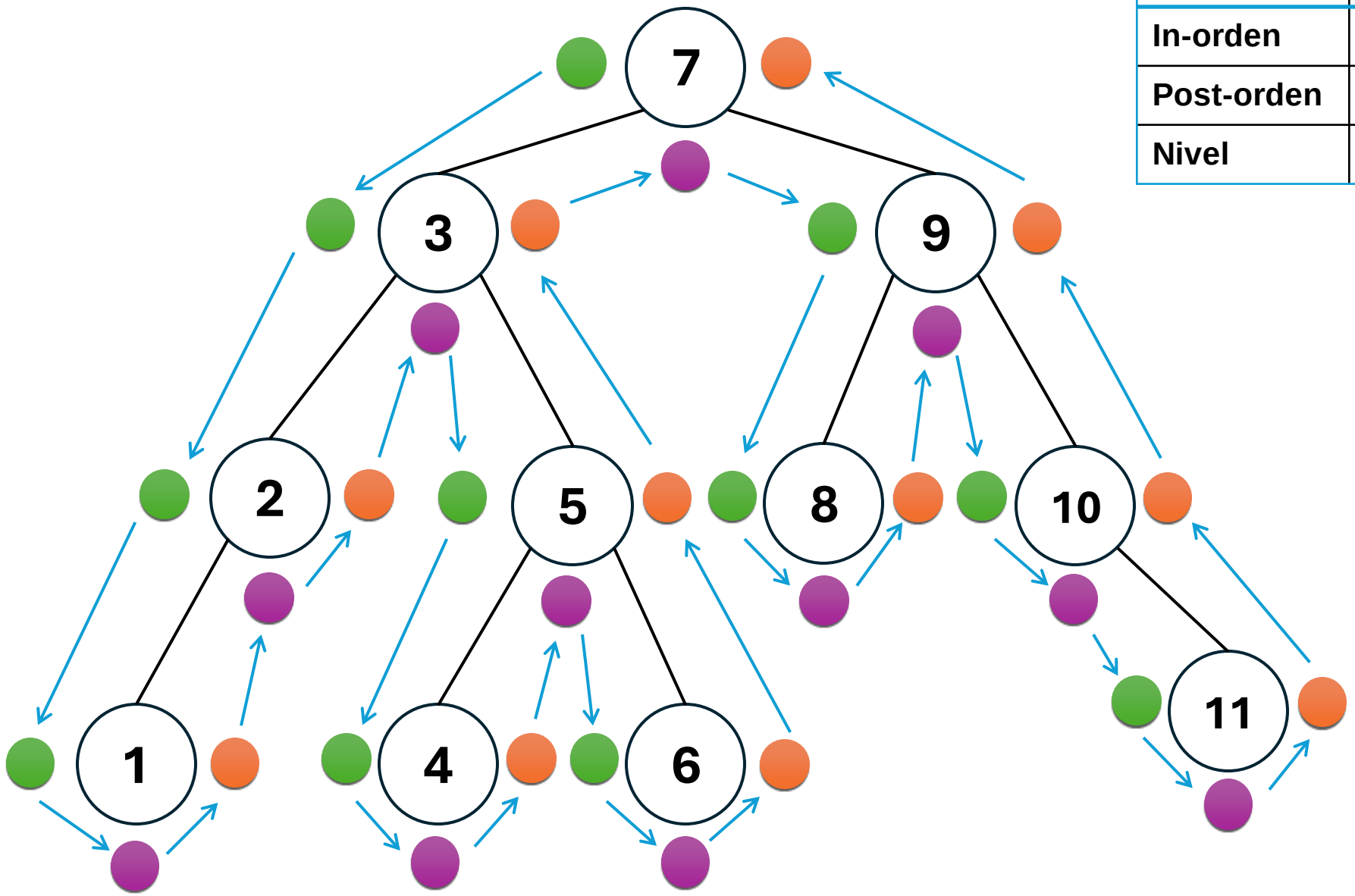
Pre-orden	43, 10, 8, 15, 54, 50, 53
In-orden	8, 10, 15, 43, 50, 53, 54
Post-orden	



Pre-orden	43, 10, 8, 15, 54, 50, 53
In-orden	8, 10, 15, 43, 50, 53, 54
Post-orden	8, 15, 10, 53, 50, 54, 43
Nivel	43, 10, 54, 8, 15, 50, 53

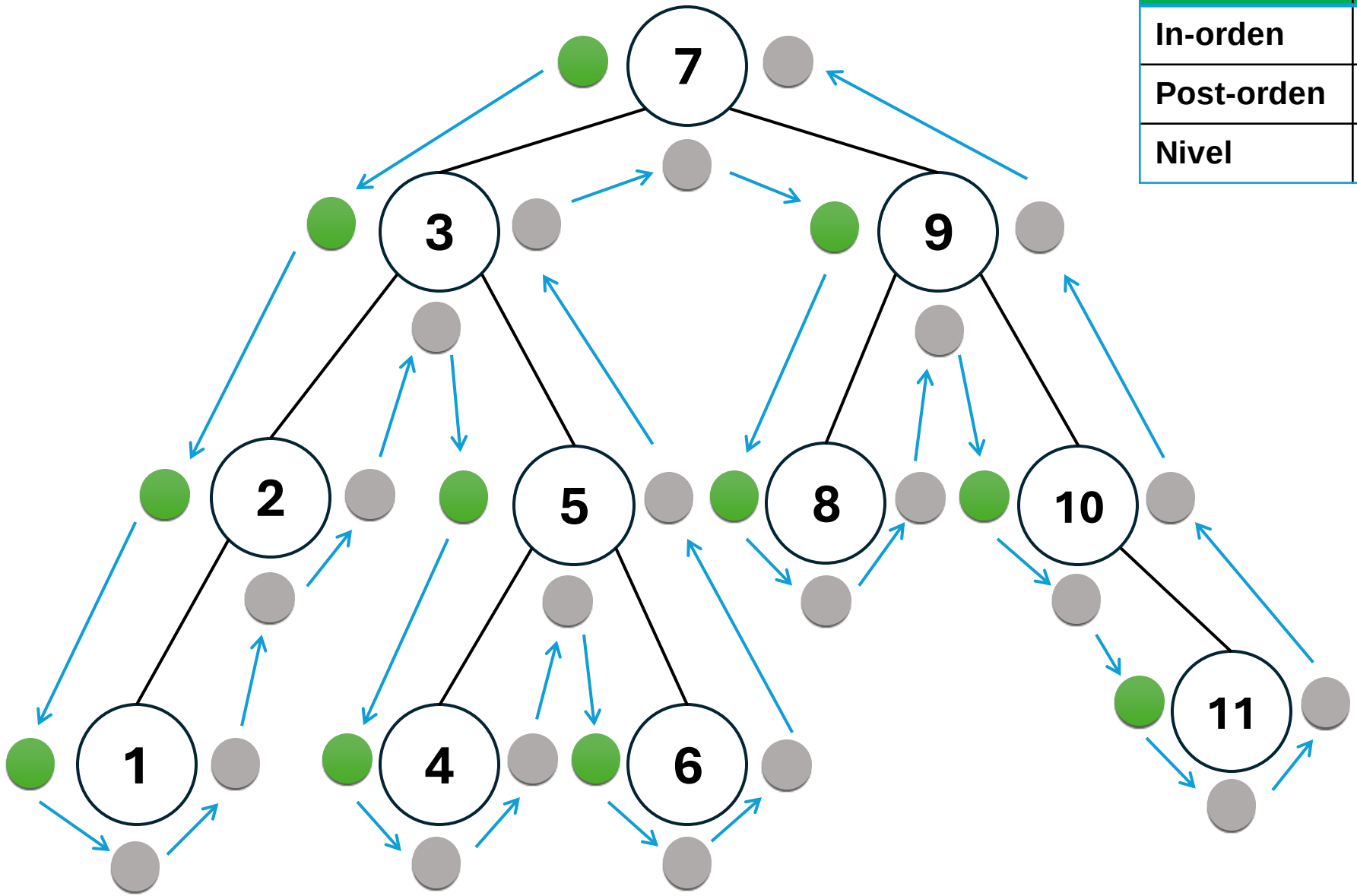

7
3
9
2
5
8
10
1
4
6
11

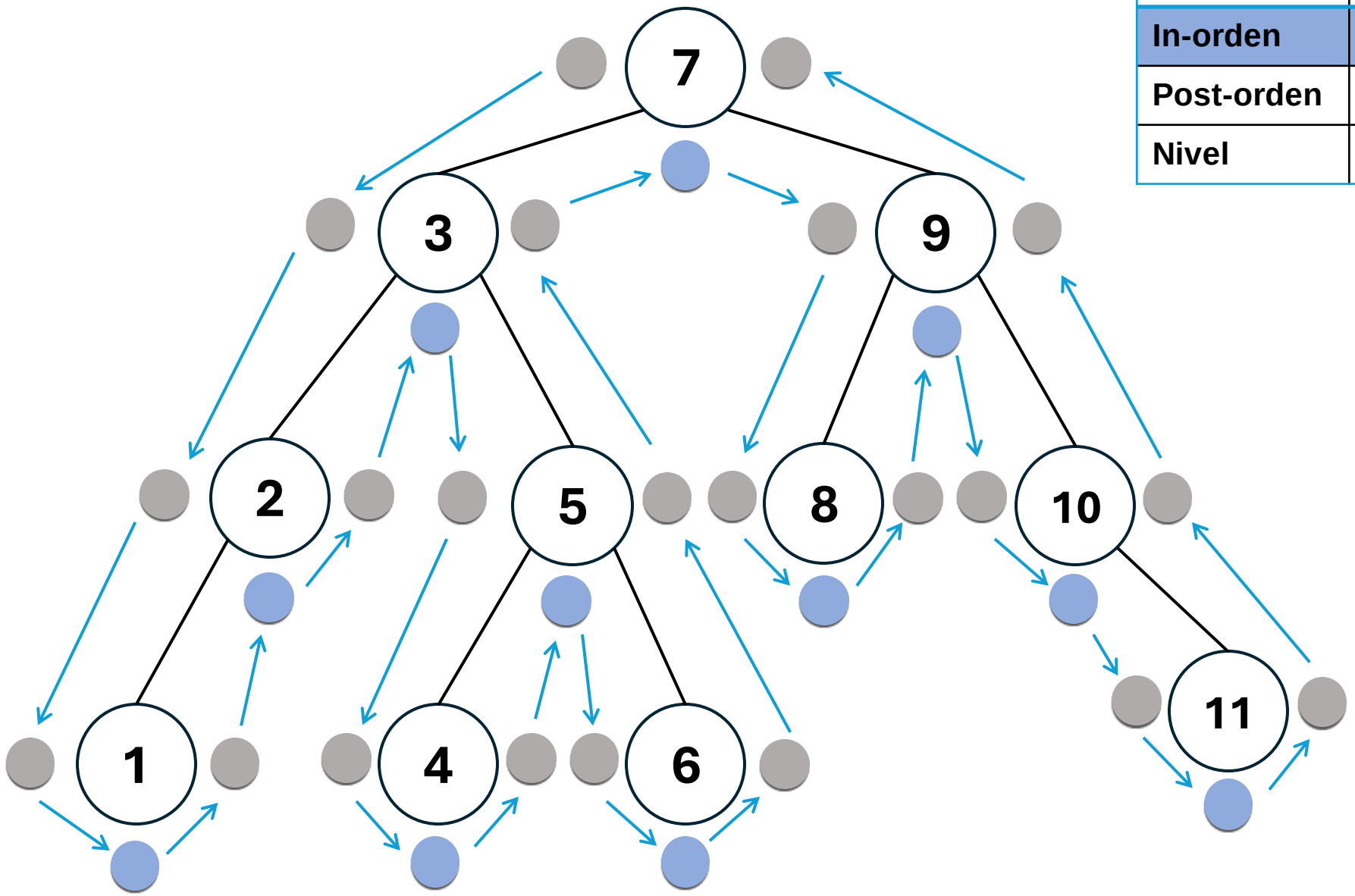

7
3
9
2
5
8
10
1
4
6
11



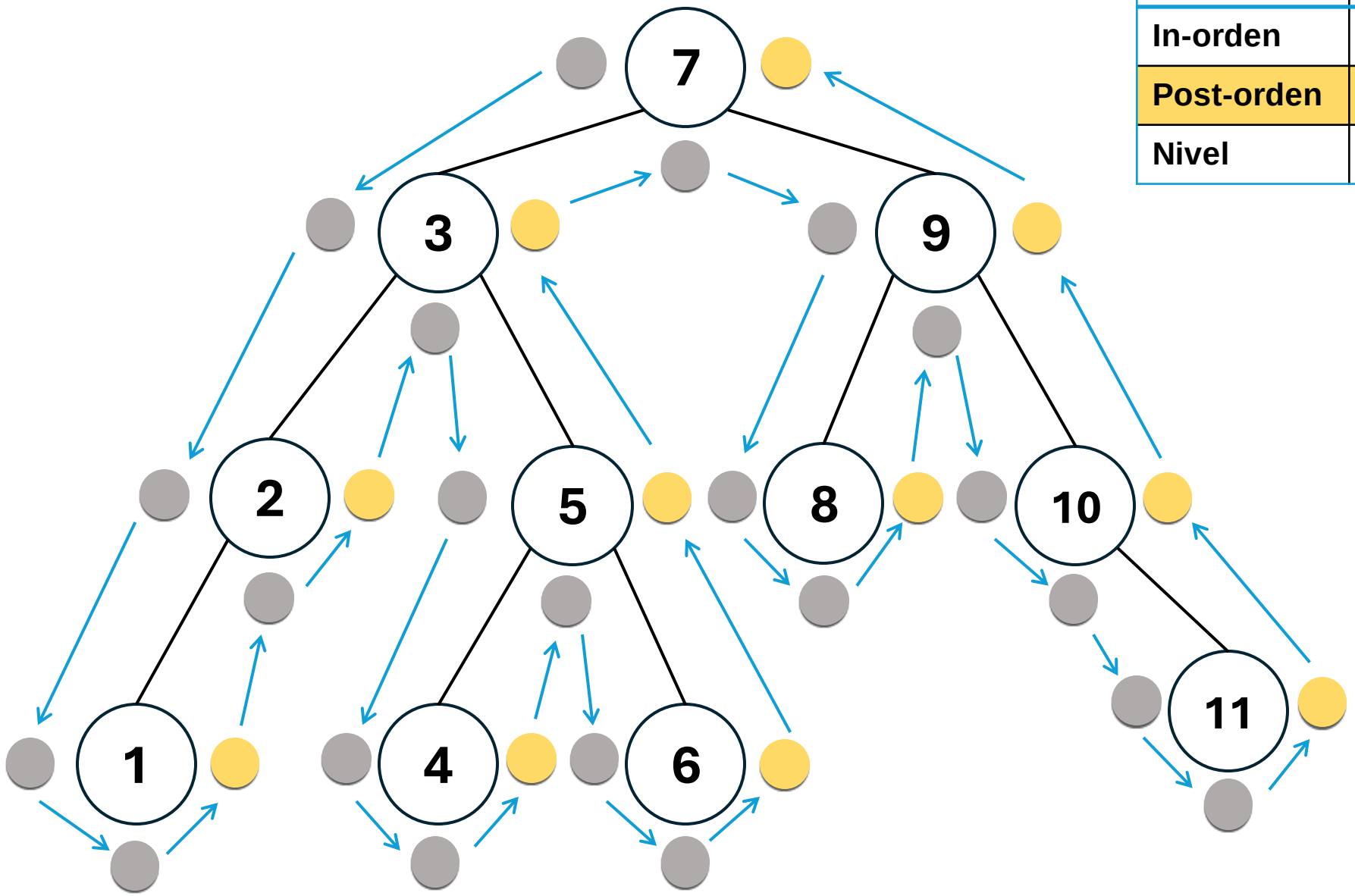
Pre-orden	7, 3, 2, 1, 5, 4, 6, 9, 8, 10,11
In-orden	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Post-orden	1, 2, 4, 6, 5, 3, 8, 11, 10, 9, 7
Nivel	7, 3, 9, 2, 5, 8, 10, 1, 4, 6, 11

Pre-orden	7, 3, 2, 1, 5, 4, 6, 9, 8, 10,11
In-orden	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Post-orden	1, 2, 4, 6, 5, 3, 8, 11, 10, 9, 7
Nivel	7, 3, 9, 2, 5, 8, 10, 1, 4, 6, 11

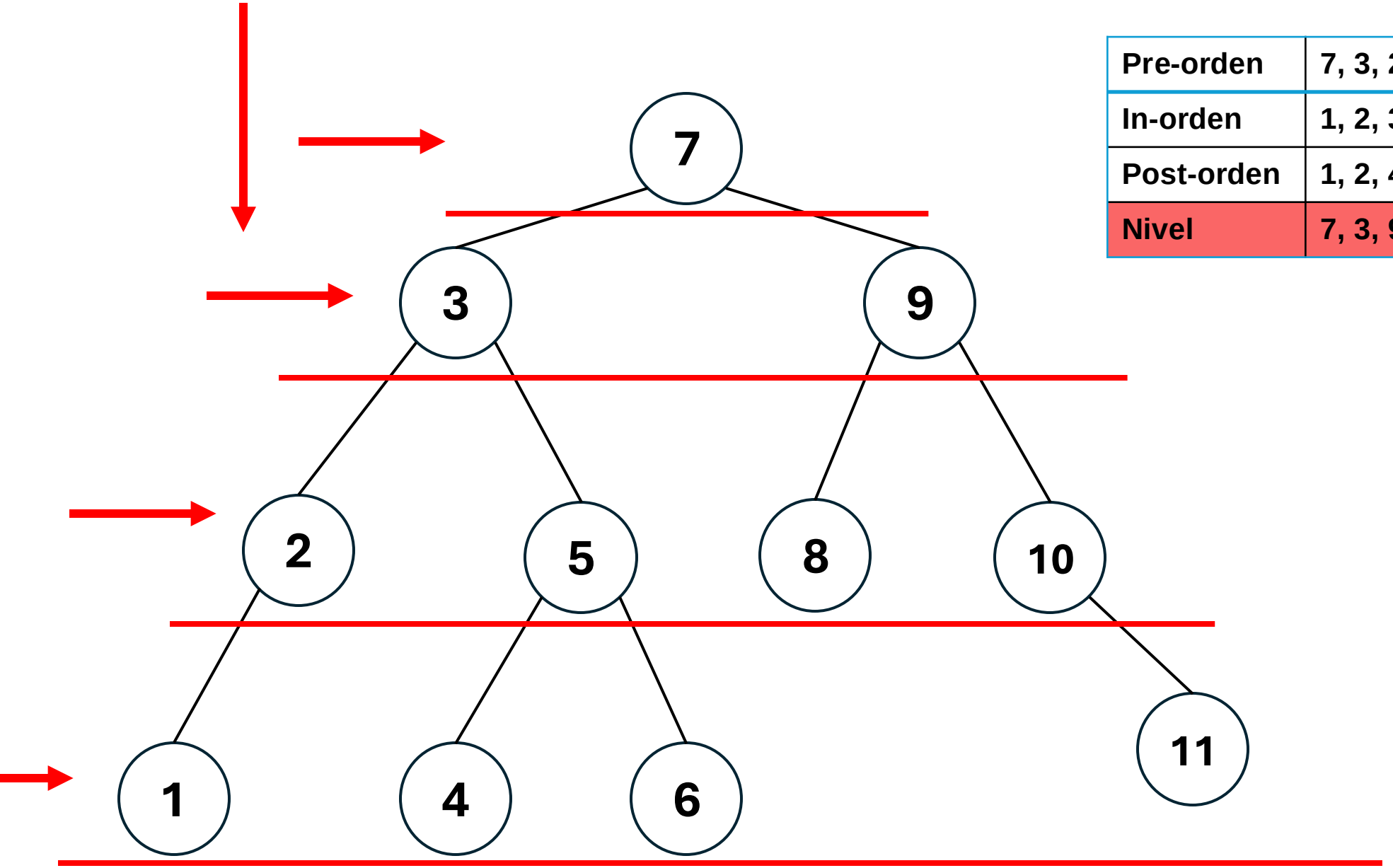




Pre-orden	7, 3, 2, 1, 5, 4, 6, 9, 8, 10,11
In-orden	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Post-orden	1, 2, 4, 6, 5, 3, 8, 11, 10, 9, 7
Nivel	7, 3, 9, 2, 5, 8, 10, 1, 4, 6, 11

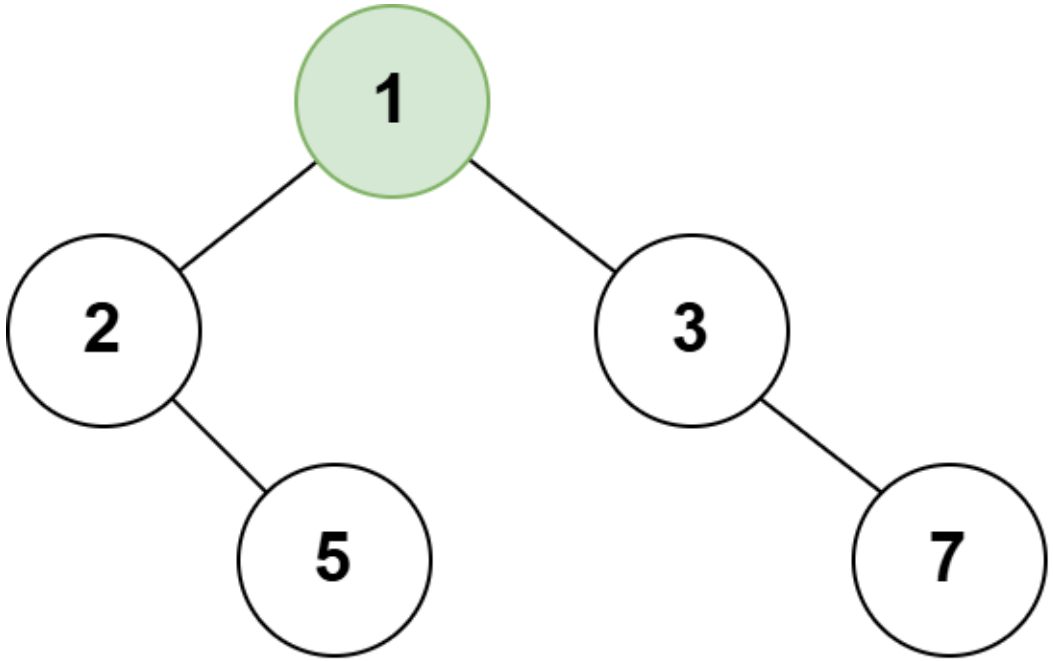


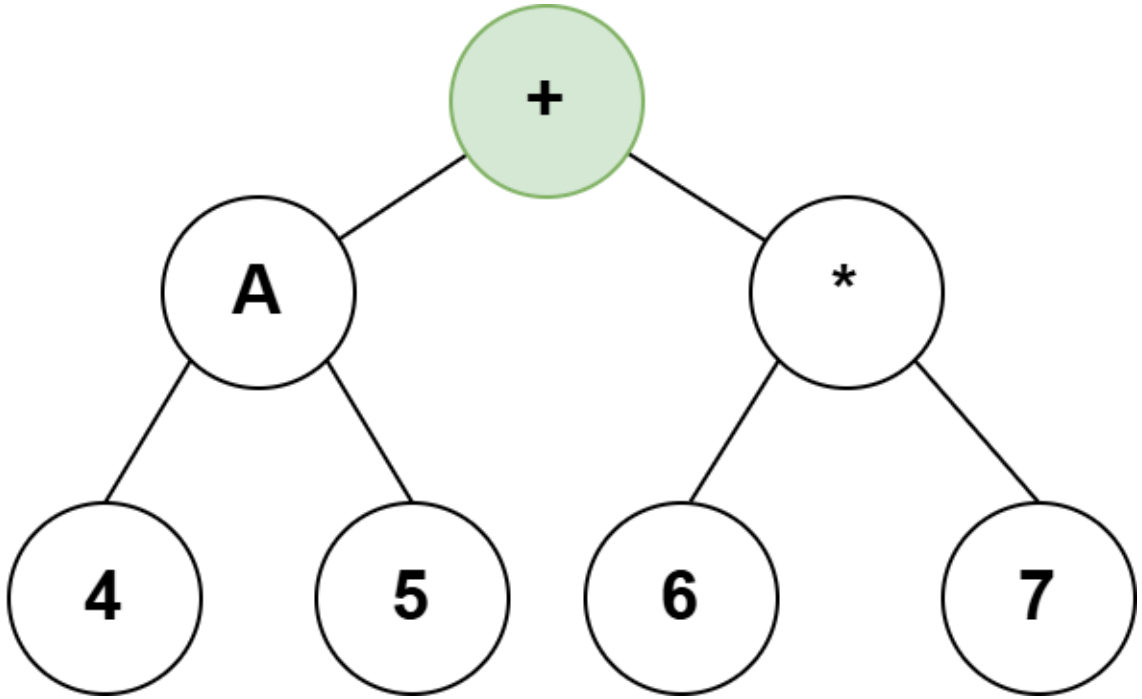
Pre-orden	7, 3, 2, 1, 5, 4, 6, 9, 8, 10,11
In-orden	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Post-orden	1, 2, 4, 6, 5, 3, 8, 11, 10, 9, 7
Nivel	7, 3, 9, 2, 5, 8, 10, 1, 4, 6, 11



Pre-orden	7, 3, 2, 1, 5, 4, 6, 9, 8, 10,11
In-orden	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Post-orden	1, 2, 4, 6, 5, 3, 8, 11, 10, 9, 7
Nivel	7, 3, 9, 2, 5, 8, 10, 1, 4, 6, 11

Determine los recorridos in-orden, post-orden y pre-orden de los siguientes árboles:



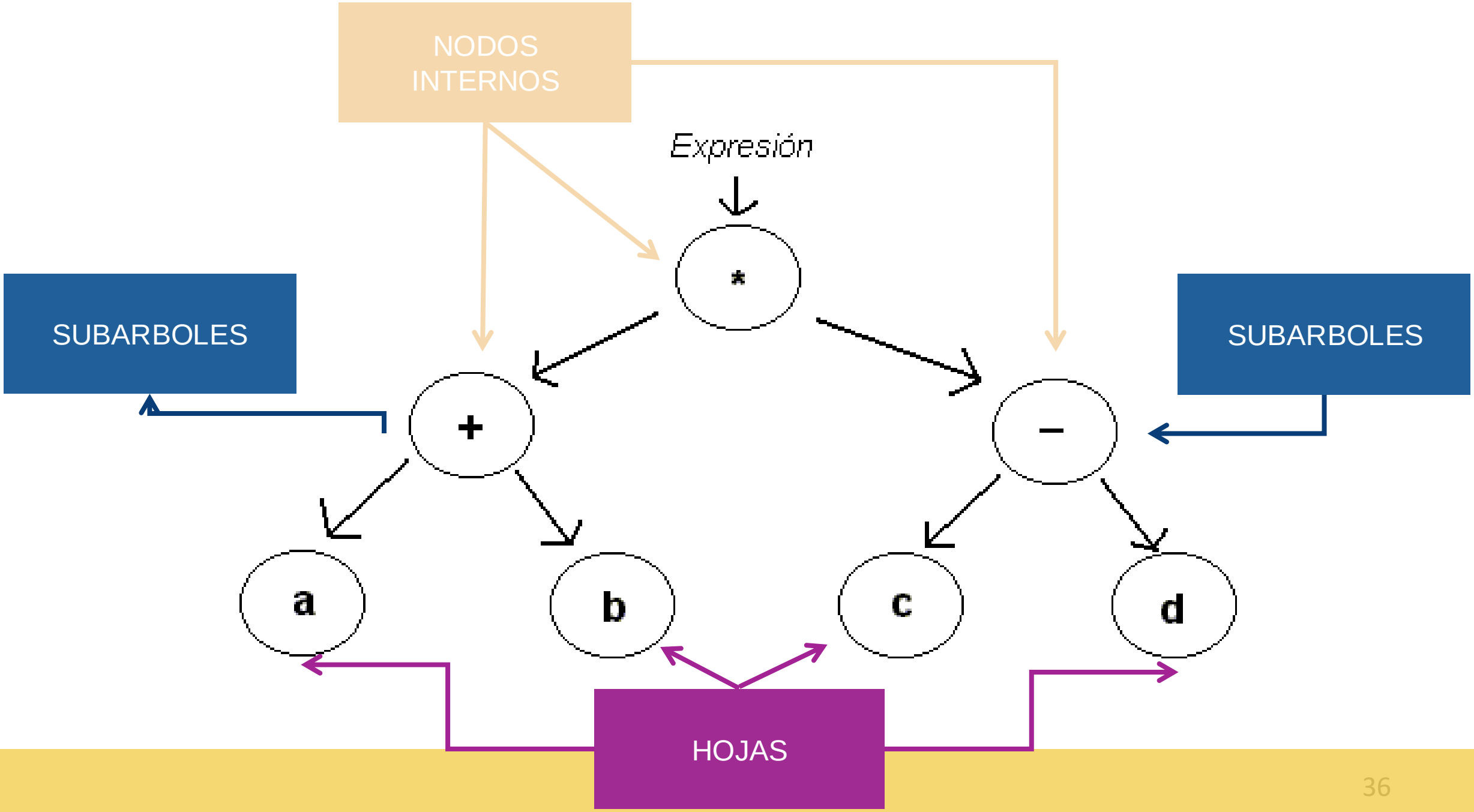


Árboles de expresión

DEF. Los árboles de expresión son estructuras de datos utilizadas para representar expresiones, comúnmente en el contexto de lenguajes de programación y compiladores. Estas estructuras permiten almacenar y manipular expresiones aritméticas o lógicas de manera jerárquica, facilitando su evaluación y transformación.

Están conformados por:

HOJAS	Representan operandos, constantes o variables.
NODOS INTERNOS	Representan operadores
SUBARBOLES	Representan subexpresiones, donde la raíz del subárbol es el operador y sus hijos los operandos u otras subexpresiones.


NODOS INTERNOS
Expresión
*
+
−
a
b
c
d
SUBARBOLES
SUBARBOLES
HOJAS

CONSTRUCCION

1. Análisis de la Expresión:
   - Identificar los operandos y operadores en la expresión.
   - Considerar la jerarquía de los operadores y el uso de paréntesis para determinar el orden de evaluación.

Jerarquía de operadores	
Mayor	( )
.	* , /
.	
.	+, -
Menor	

2. Creación de Nodos:
   - Para cada operando, crear un nodo hoja.
   - Para cada operador, crear un nodo interno que tendrá como hijos los nodos correspondientes a sus operandos.

3.Ensamblado del Árbol:

- Comenzar con los operadores de menor precedencia (o los que están dentro de los paréntesis más internos) y avanzar hacia los de mayor precedencia.
- Conectar los nodos de acuerdo con la estructura de la expresión.

Ejemplo. Construya un árbol de expresión con la siguiente expresión:

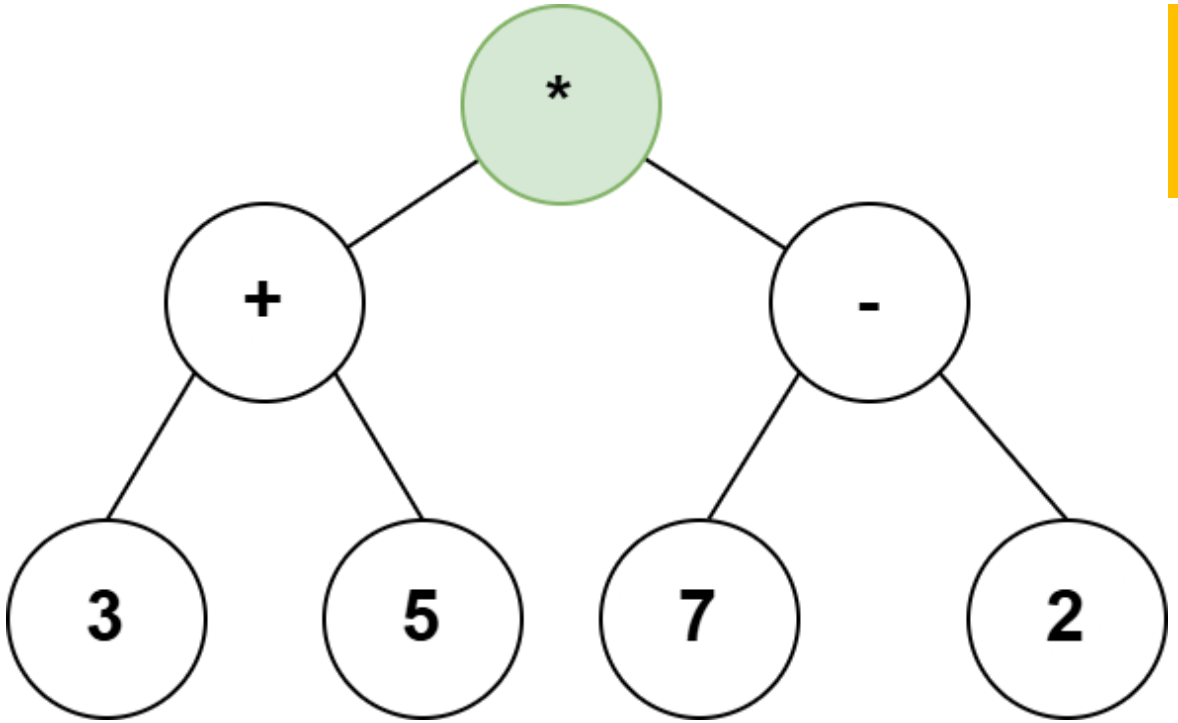
(3 + 5) * (7 - 2)

Paso 1. Identificar operandos y operadores:

Operadores	Operandos
+	3
*	5
-	7
()	2

Paso 2. Considerar jerarquía:
•Primero evaluar (3 + 5) y (7 - 2).
•Luego multiplicar los resultados.

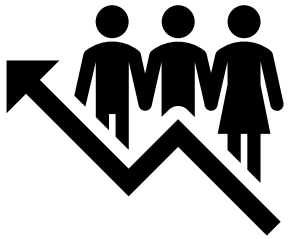
Paso 3. Construir el Árbol:
•Crear nodos para los operandos: 3, 5, 7, 2.
•Crear nodos para los operadores + y -, conectándolos a sus operandos.
•Crear un nodo para el operador *, conectándolo a los nodos de + y -.



Para su evaluación se sigue un recorrido post-orden.

1.Evaluar el subárbol izquierdo (3 + 5), resultado 8.
2.Evaluar el subárbol derecho (7 - 2), resultado 5.
3.Aplicar el operador raíz (*), multiplicando 8 y 5 para obtener 40.

PARTICIPACIONES



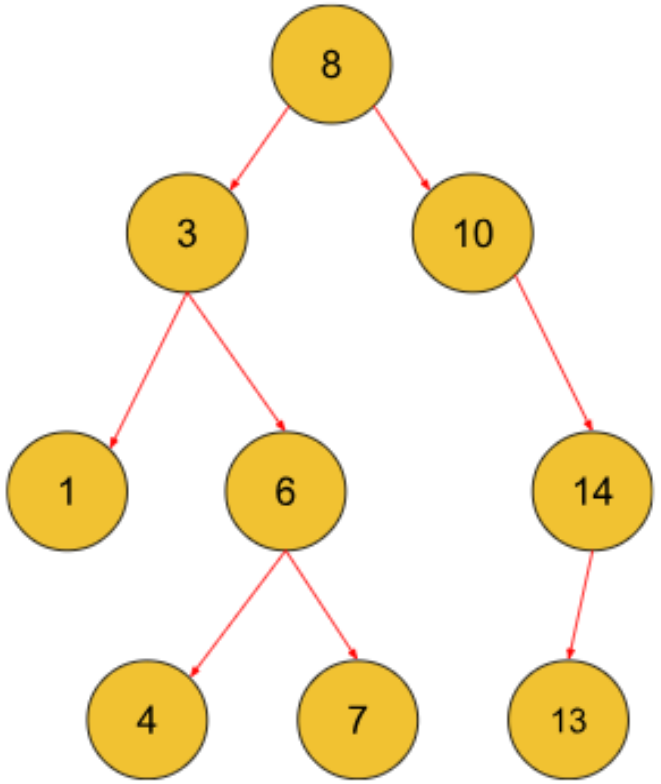
Construya un árbol de expresión para las siguientes expresiones:

1.Expresión: 4 * (5 + 3) – 6

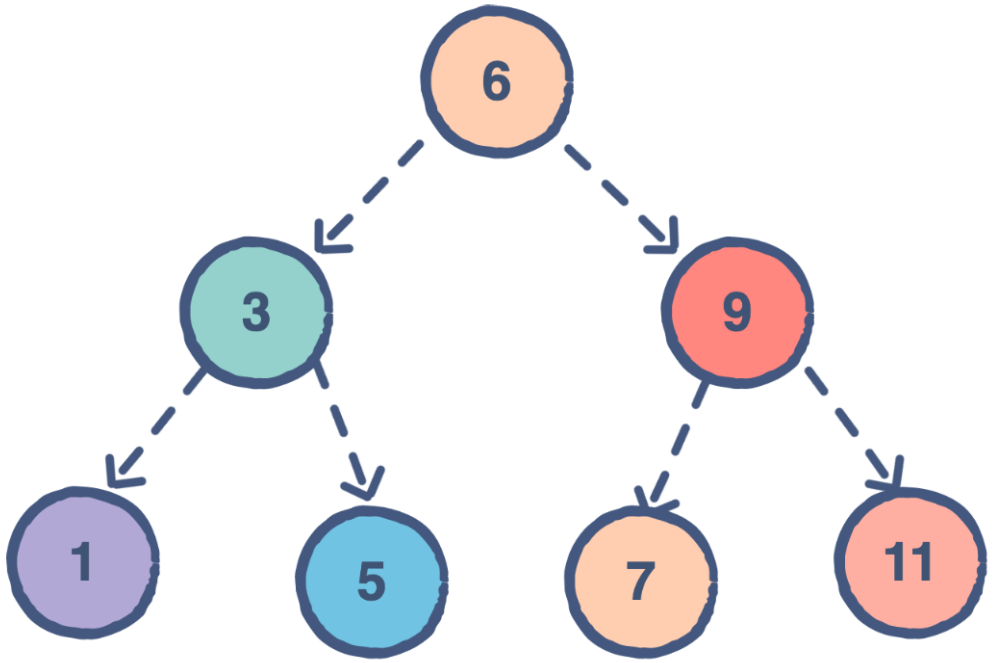
2. Expresión: (7 + 2) * (3 - 1) / 4

Árboles binarios de búsqueda y características

Un tipo de árbol binario que cumple con una propiedad específica nombrada la **propiedad del árbol binario de búsqueda**.
Contiene una clave y punteros para el hijo izquierdo, derecho y el padre de cada nodo.

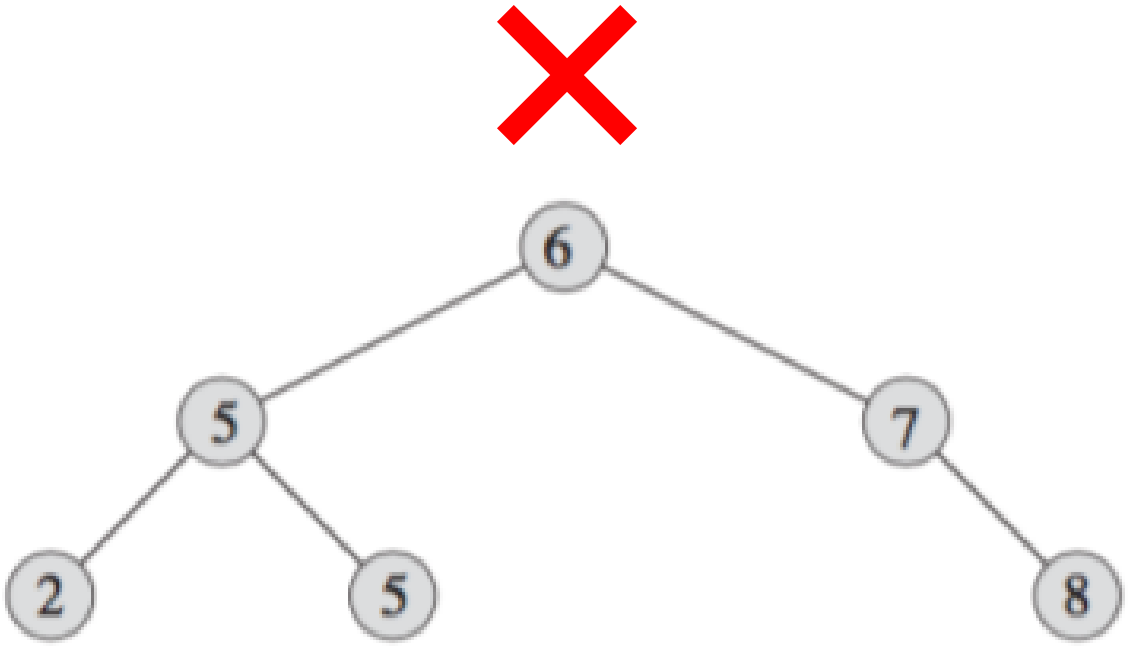


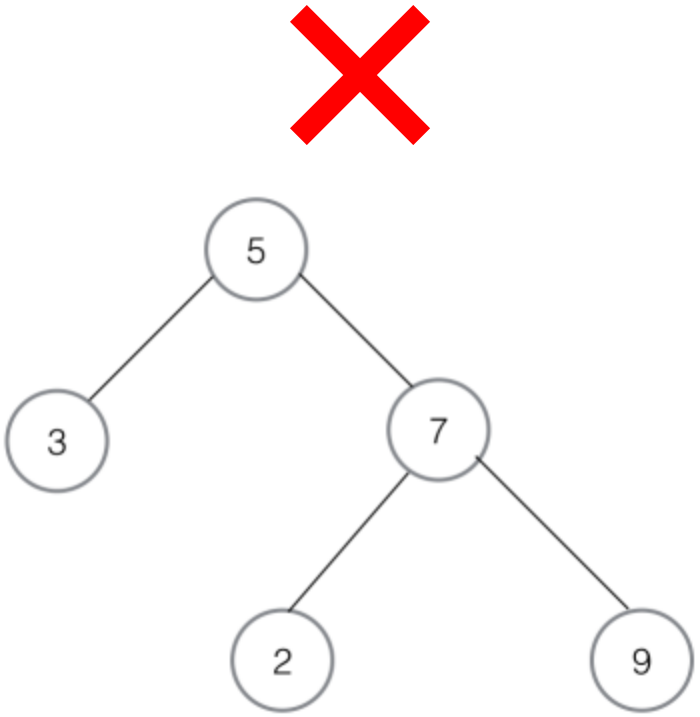
Propiedad del árbol binario de búsqueda



- **DEF**. Sea x un nodo cualquiera en un árbol binario de búsqueda. Si y es un nodo en el subárbol izquierdo de x, entonces y.dato < x.dato. Si y es un nodo en el subárbol derecho de x, entonces y.dato > x.dato.

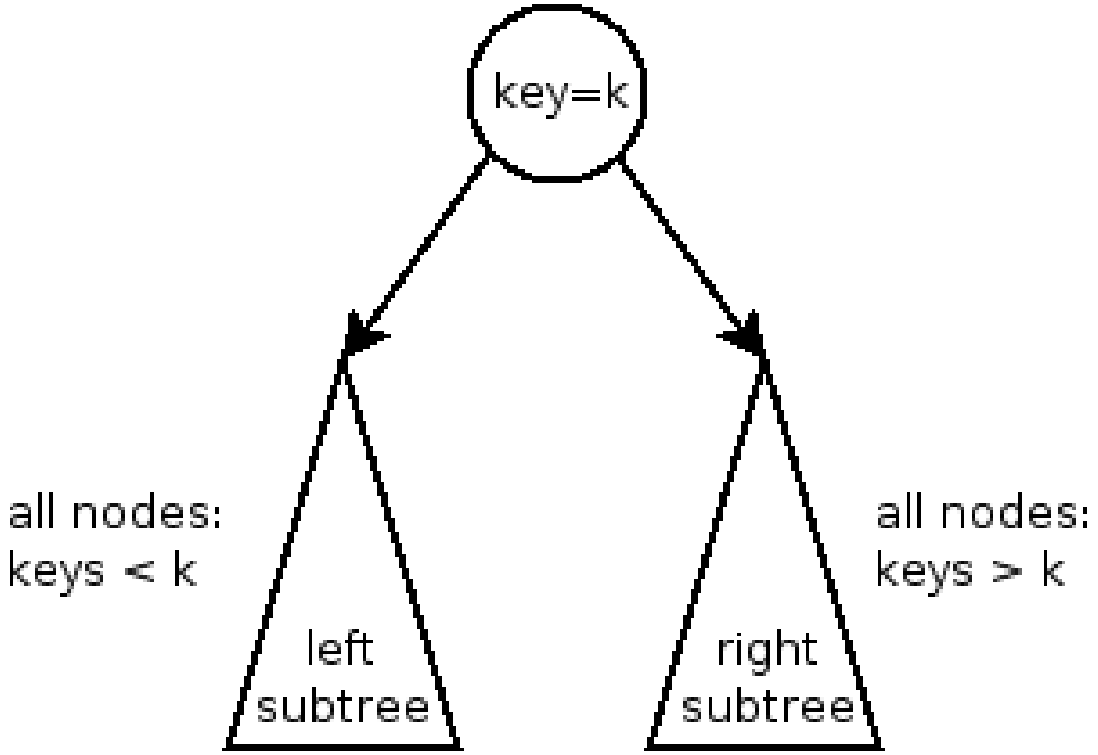
Ejemplos





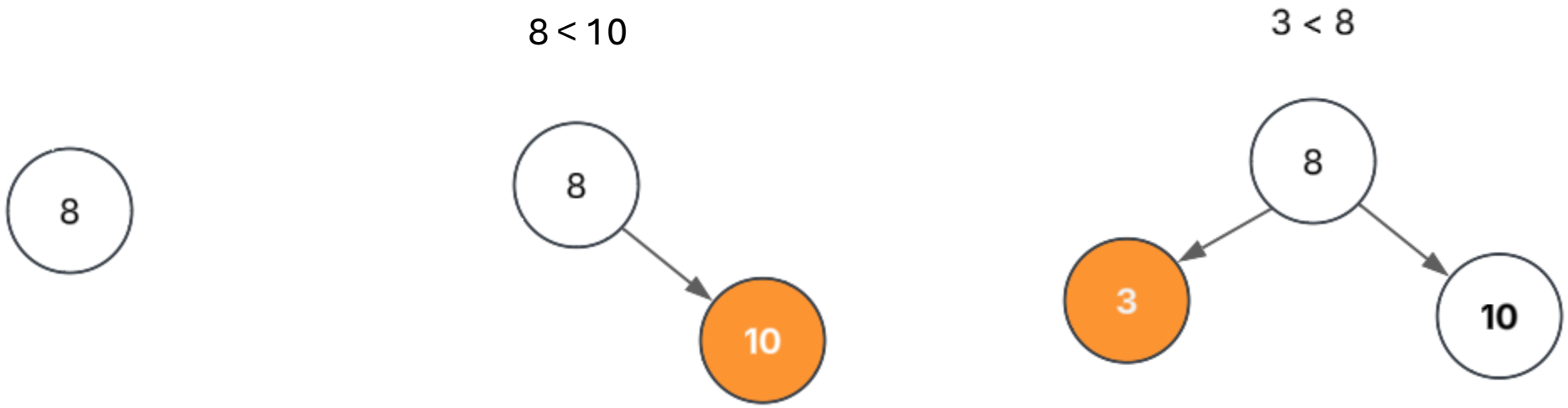
Creación

- El lado izquierdo contiene elementos estrictamente menores que la raíz.
- El lado derecho contiene elementos estrictamente mayores que la raíz.
- Se compara cada elemento con la raíz para saber en qué lado o subárbol irá.
- El primer nodo en el arreglo es la raíz.



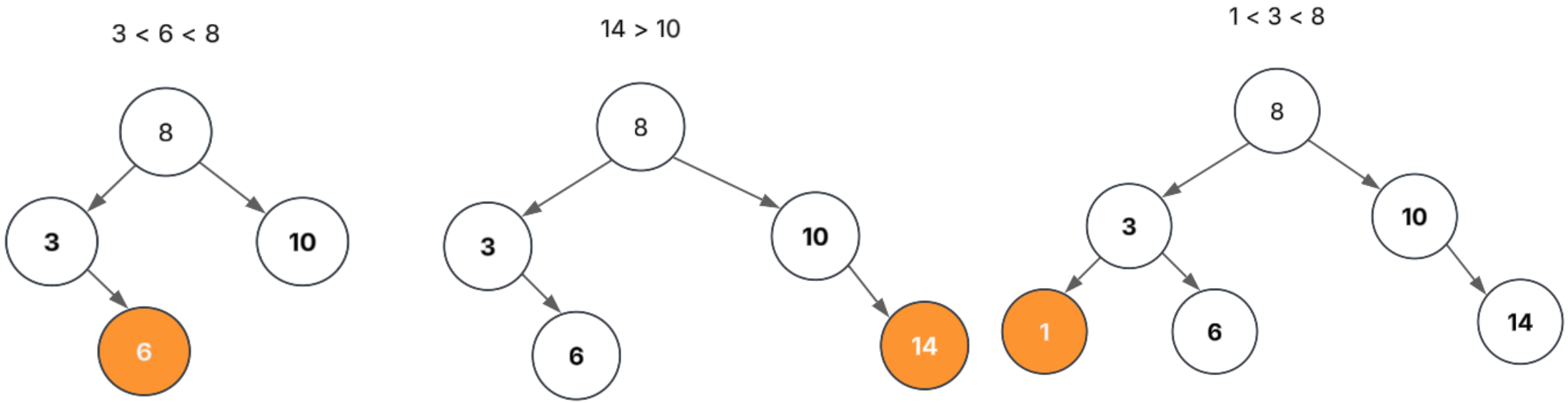
Creación

8	10	3	6	14	1	4	13	7



Creación

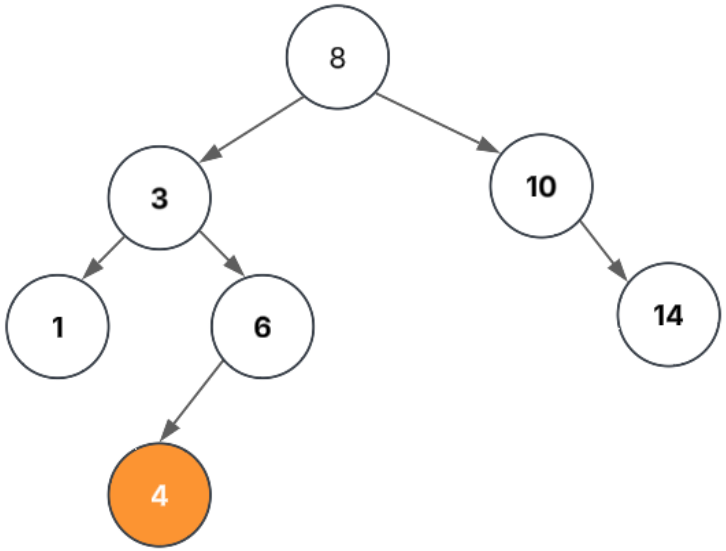
8	10	3	6	14	1	4	13	7



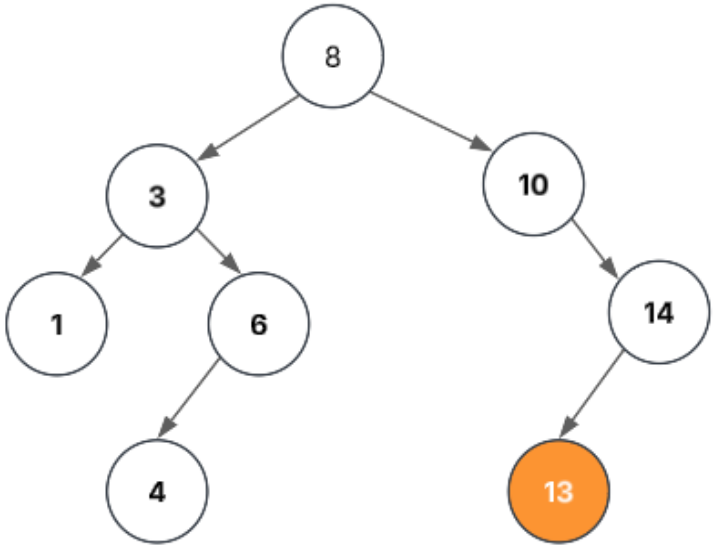
Creación

8	10	3	6	14	1	4	13	7

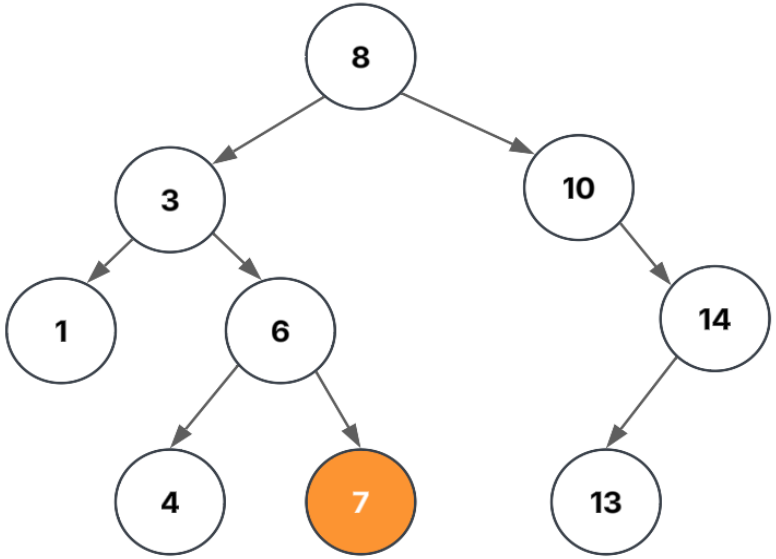
4 < 6 < 8



10 < 13 < 14

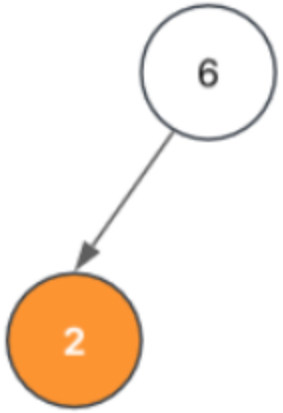


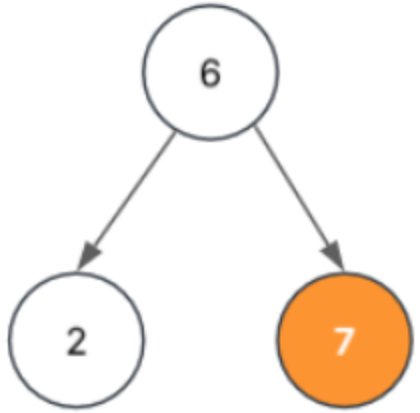
6 < 7 < 8



Creación

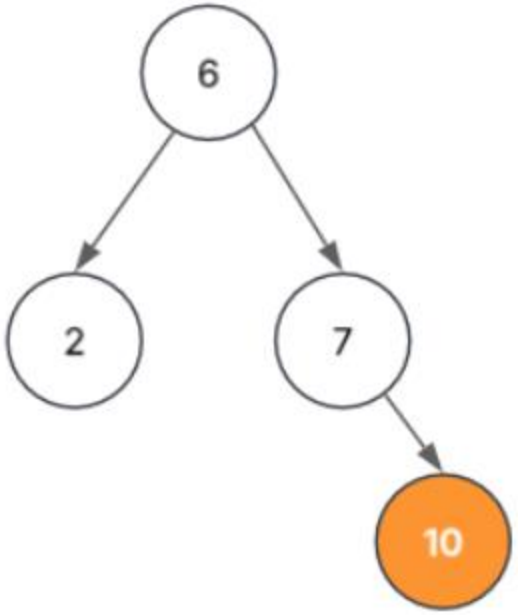
6	2	7	10	5	1

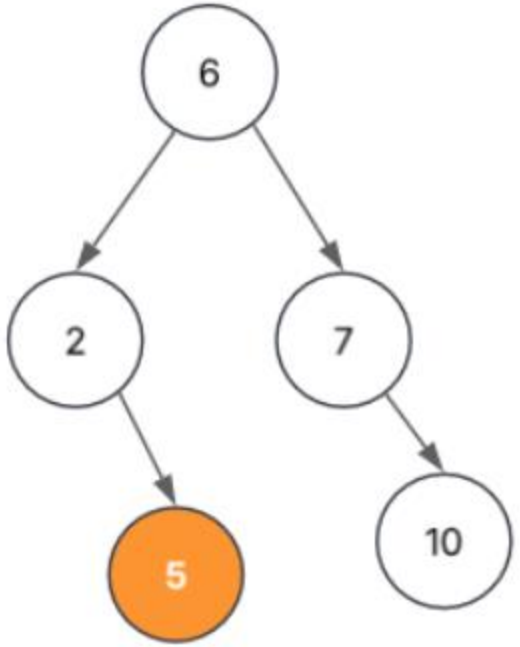


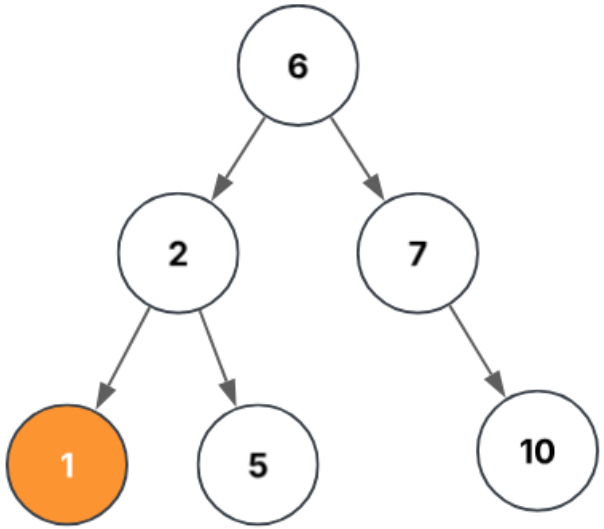


Creación

6	2	7	10	5	1







Operaciones en árboles binarios de búsqueda

Las tres operaciones principales:

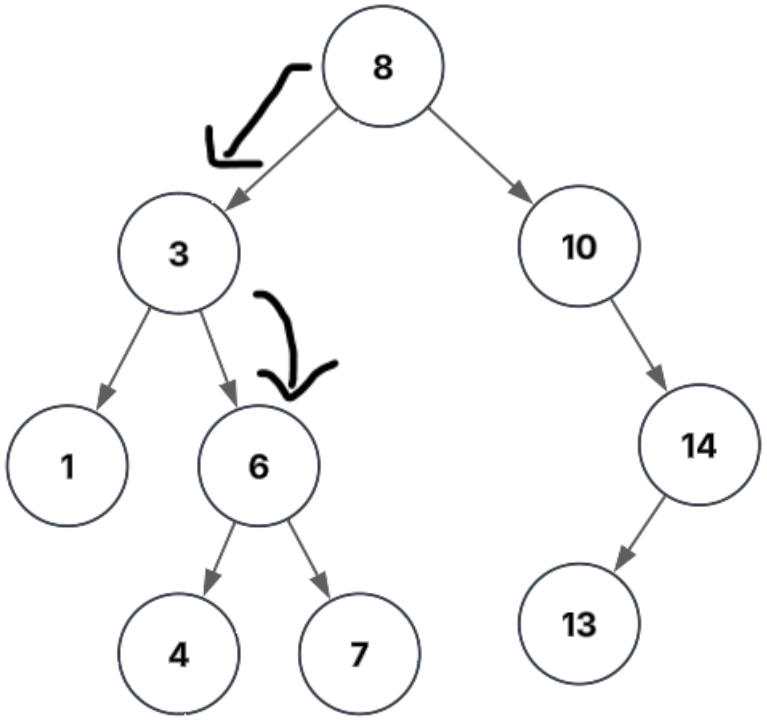
- **Búsqueda**
- **Inserción**
- **Eliminación**

Búsqueda

- Si la clave no se encuentra en la raíz:
- Va hacia la izquierda si la clave deseada es menor que la raíz.
- Va hacia la derecha si la clave deseada es mayor que la raíz.
- Si no existe, el valor será nulo.

Búsqueda

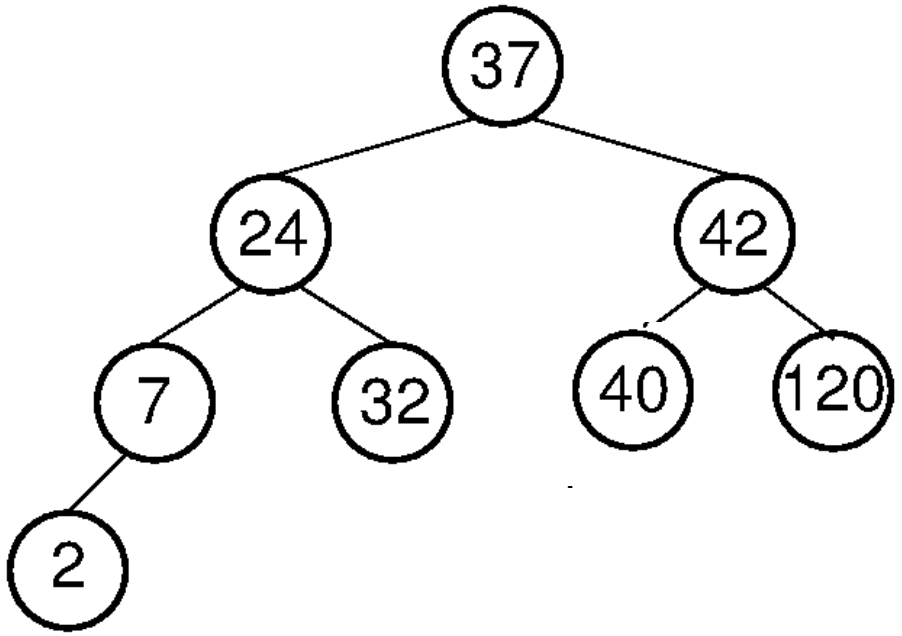
- Objetivo: Encontrar el número 6



1. Empezar en el nodo 8 (raíz)
2. 6 < 8, entonces se busca a la izquierda
3. 6 > 3, entonces se busca a la derecha
4. Se encuentra 6, que es el hijo derecho de 3.

Búsqueda

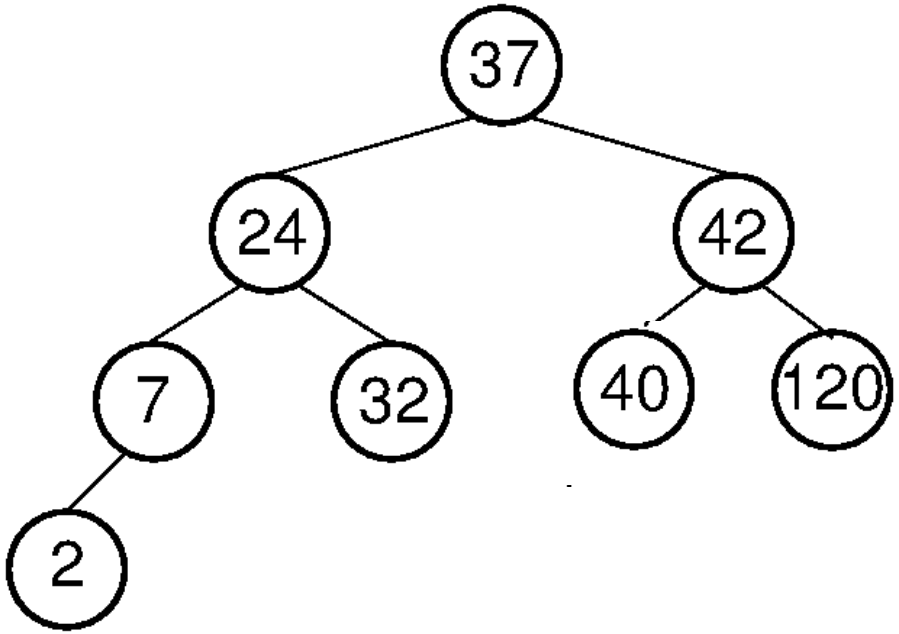
- Objetivo: Encontrar el número 120



1. Empezar en el nodo 37 (raíz)
2. 120 > 37, entonces se busca a la derecha
3. 120 > 42, entonces se busca a la derecha
4. Se encuentra 120, que es el hijo derecho del nodo 42.

Búsqueda

- Objetivo: Encontrar el número 40 y encontrar el número 32.

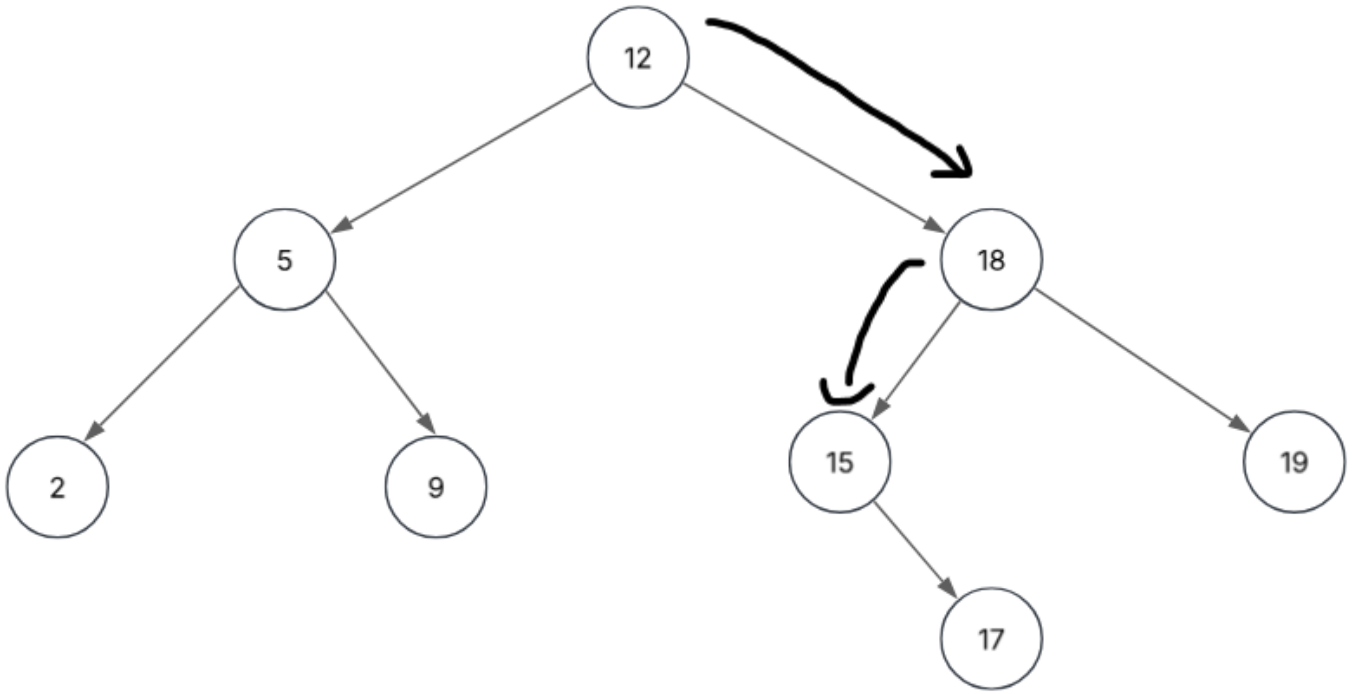


Inserción

- Se crea un nodo nuevo, con los hijos apuntando a NULL.
- Hace un recorrido desde el nodo raíz.
- Si la clave del nodo nuevo es menor, va a la izquierda. De lo contrario, va a la derecha.
- Al no encontrar más nodos que recorrer, se inserta.

Inserción

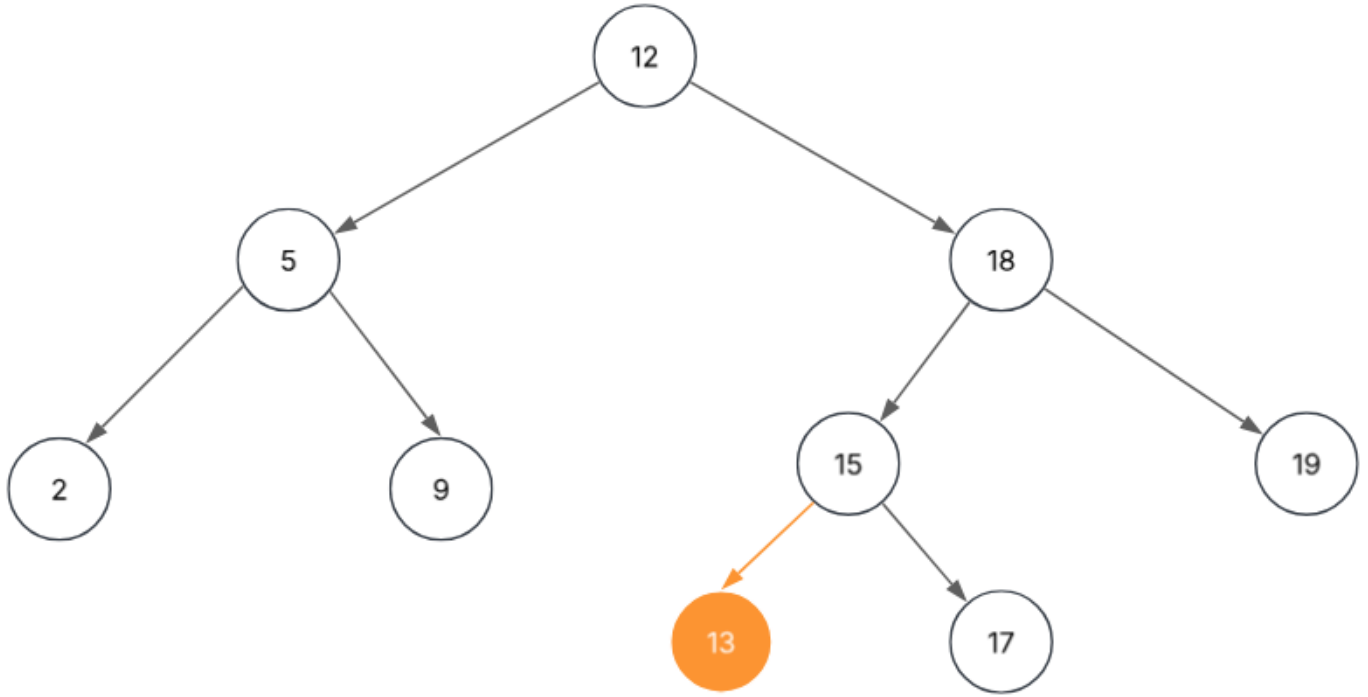
- Objetivo: insertar un nodo con clave 13.



1. 13 > 12, se mueve a la derecha.
2. 13 < 18, se mueve a la izquierda
3. Se inserta 13 como hijo izquierdo de 15, ya que 13 < 15.

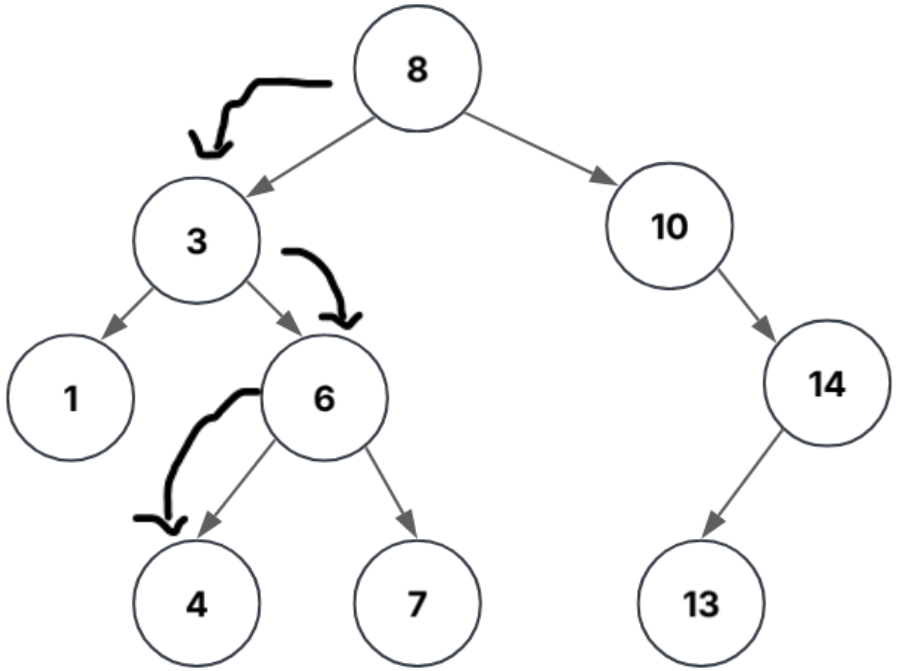
Inserción

- Objetivo: insertar un nodo con clave 13.



Inserción

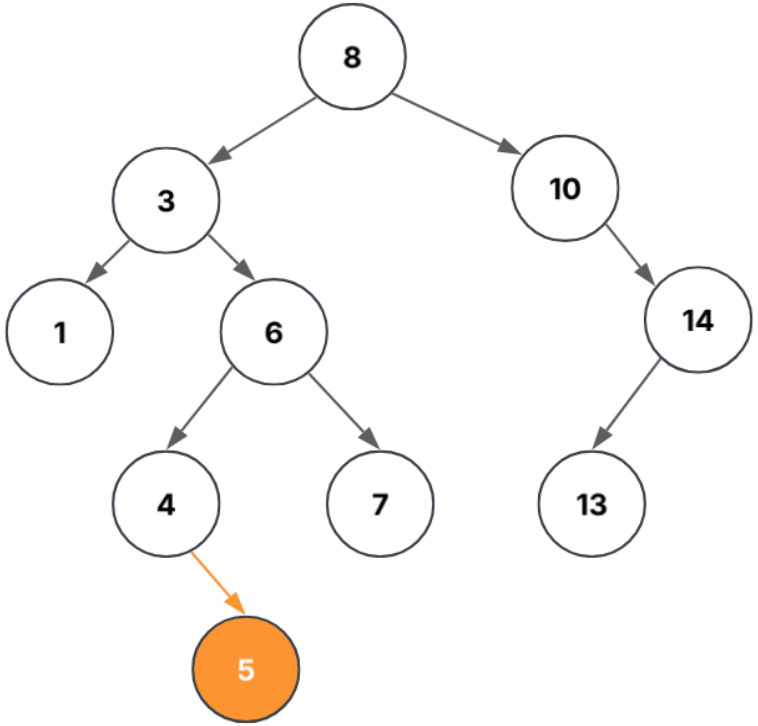
- Objetivo: insertar un nodo con clave 5.



1. 5 < 8, se mueve a la izquierda.
2. 5 > 3, se mueve a la derecha.
3. 5 < 6, se mueve a la izquierda.
4. Se inserta 5 como hijo derecho de 4, ya que 5 > 4.

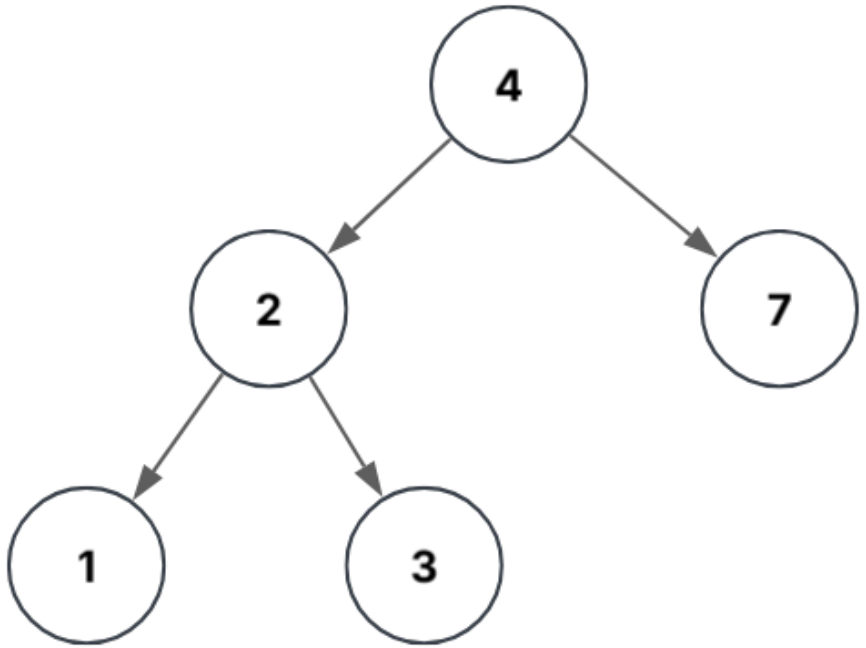
Inserción

- Objetivo: insertar un nodo con clave 5.



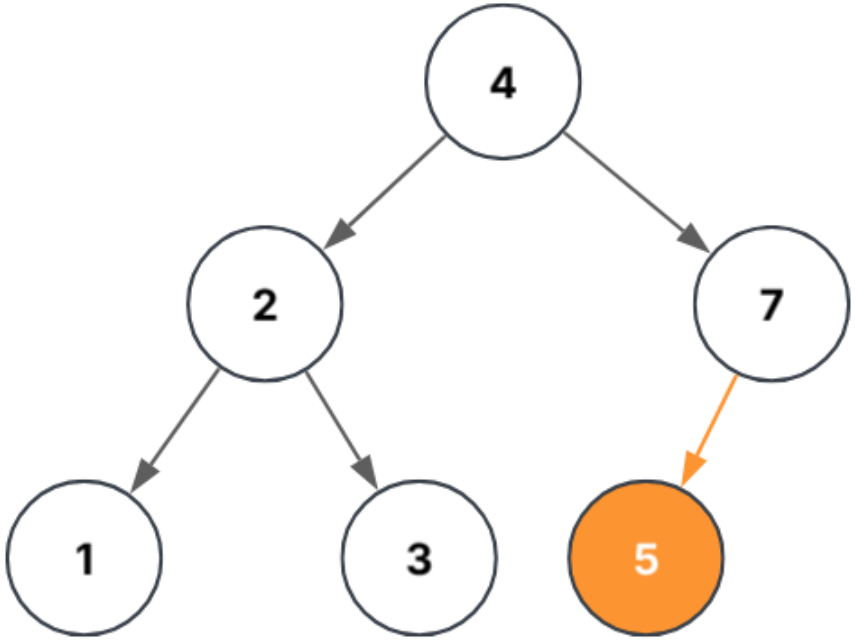
Participación

- Objetivo: insertar un nodo con clave 5.

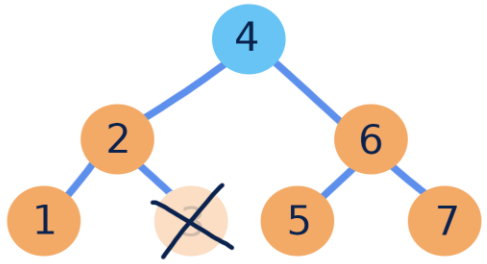


Inserción

- Objetivo: insertar un nodo con clave 5.



Eliminación



Para eliminar un nodo **x** se tienen tres casos base:

- Si x no tiene hijos, se reemplaza por NULL.
- Si x tiene un solo hijo, se reemplaza la posición de x por su hijo (izquierdo o derecho).
- Si tiene dos hijos, se encuentra el sucesor de x (llamado y) el cual es el número más pequeño del subárbol derecho para preservar el orden de conexiones.

En este último caso, solamente puede haber un hijo izquierdo, ya que el derecho sería mas grande que x.

Eliminación

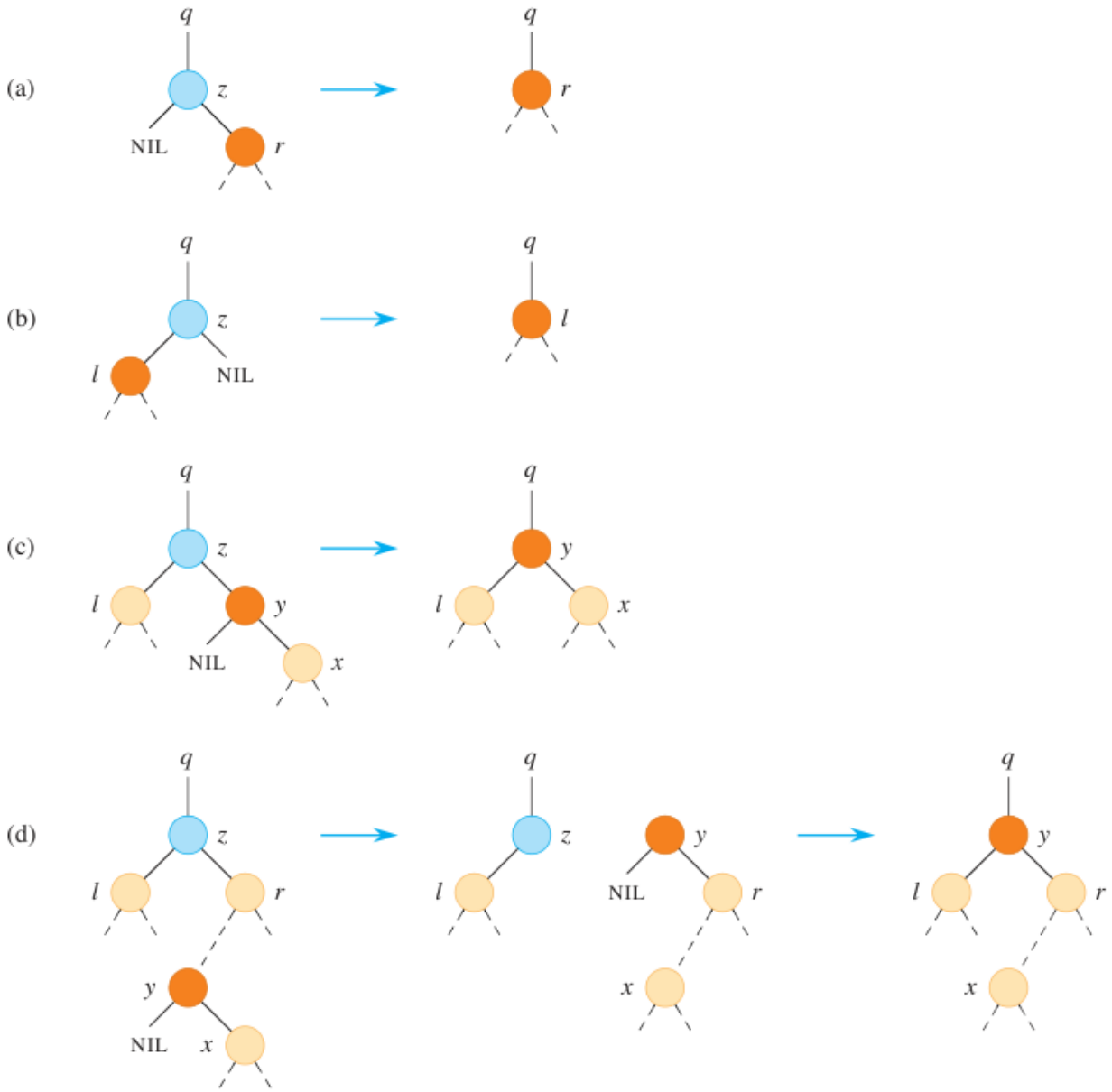
La implementación del algoritmo toma los casos base anteriores:

- Si x no tiene un hijo izquierdo, se reemplaza por su hijo derecho. Si es nulo, se borra x.
- Si x no tiene un hijo derecho, entonces tiene un hijo izquierdo. Se reemplaza x por su hijo izquierdo.
- Si x tiene dos hijos, se encuentra su sucesor y. Se separa y de su ubicación actual y se reemplaza x por y.

Eliminación

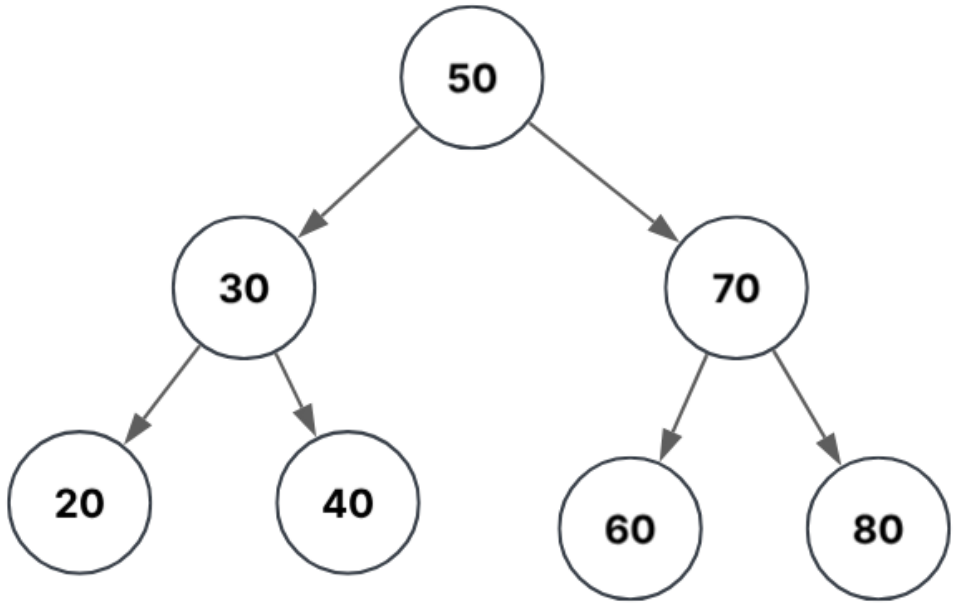
El tercer caso tiene dos formas de hacerse:

1. Si y es el hijo derecho de x, se reemplaza x por y sin modificar el hijo derecho.
2. Si y no es el sucesor directo de x, se encuentra en el subárbol derecho. Se reemplaza y por su hijo derecho, y el nodo y toma la posición de x.



Eliminación

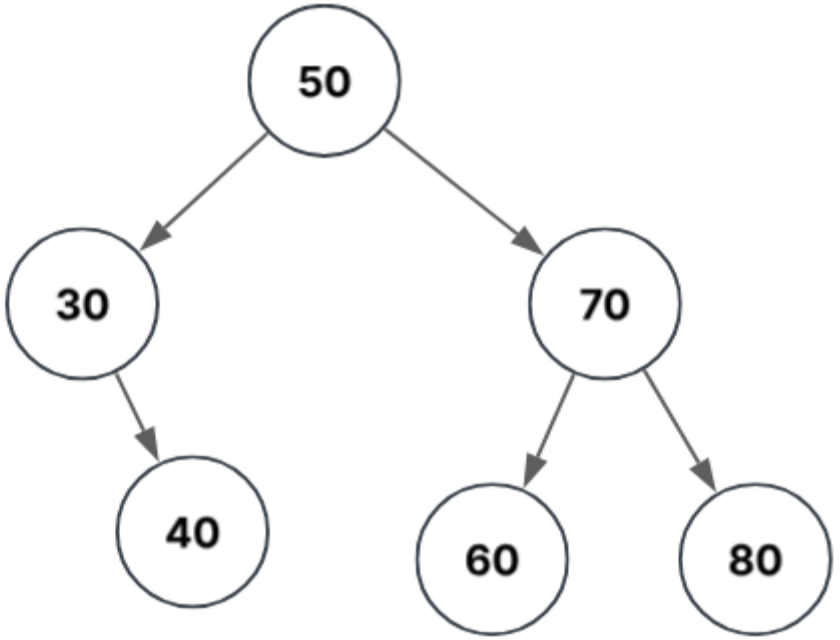
- Objetivo: eliminar el nodo 20



Como el nodo 20 no tiene hijos, sólamente se elimina sin modificar el árbol.

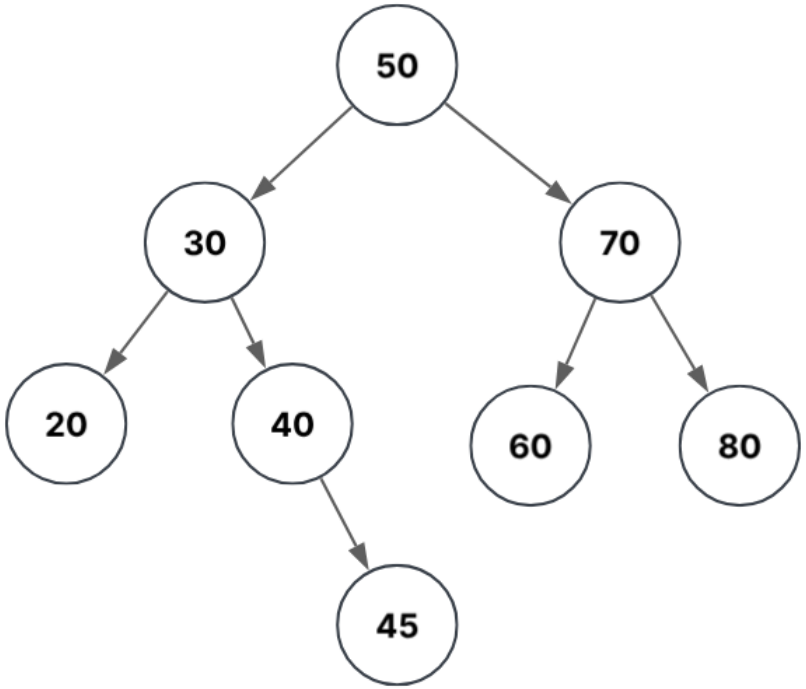
Eliminación

- Objetivo: eliminar el nodo 20



Eliminación

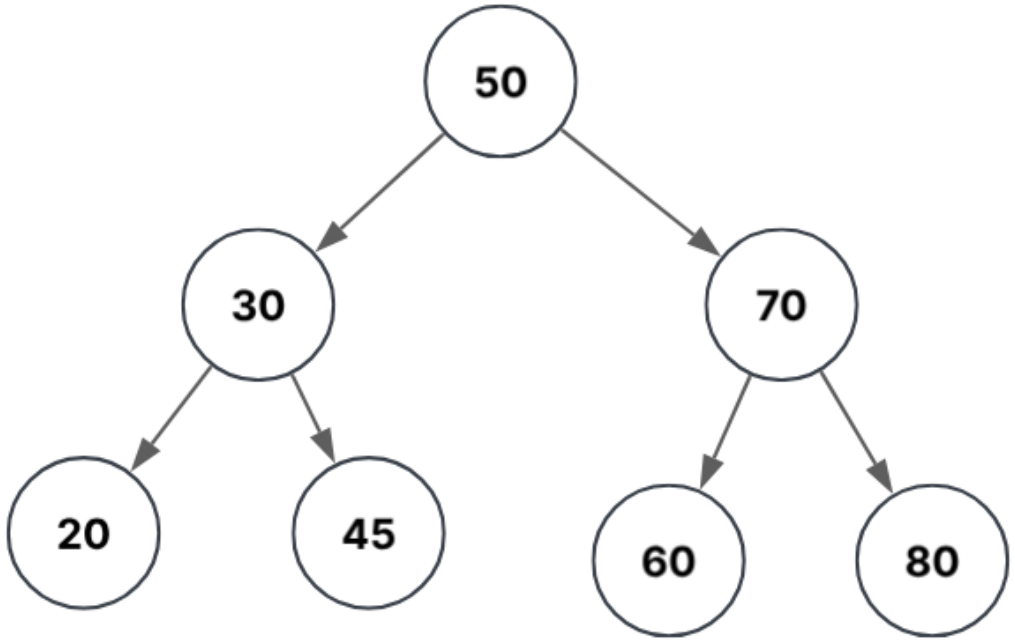
- Objetivo: eliminar el nodo 40



1. Se recorre el nodo 45 un espacio hacia arriba, reemplazando al de clave 40
2. Se elimina el nodo 40

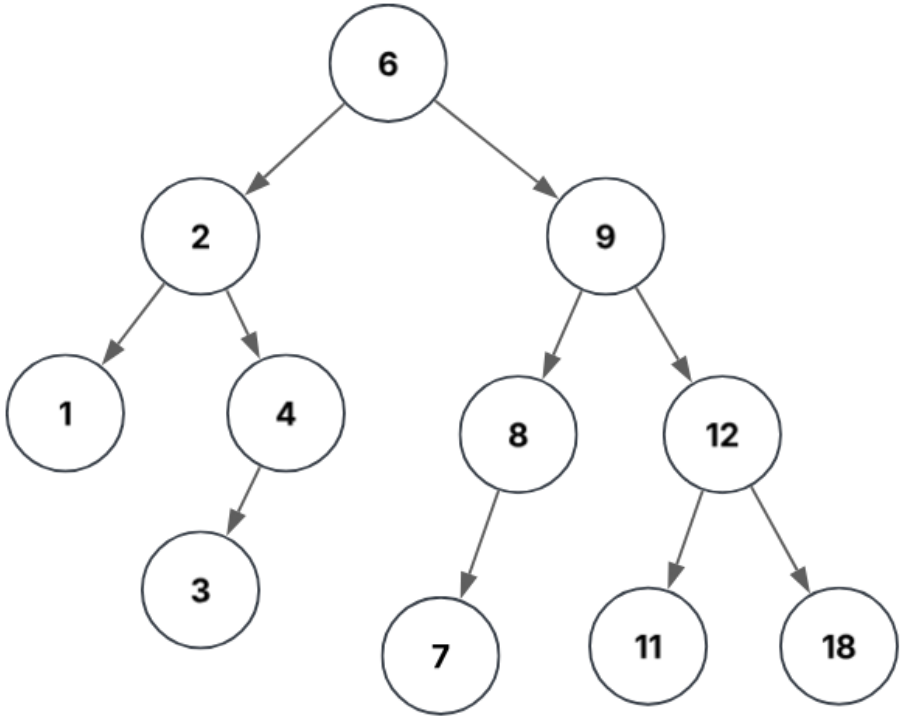
Eliminación

- Objetivo: eliminar el nodo 40



Eliminación

- Objetivo: eliminar el nodo 9

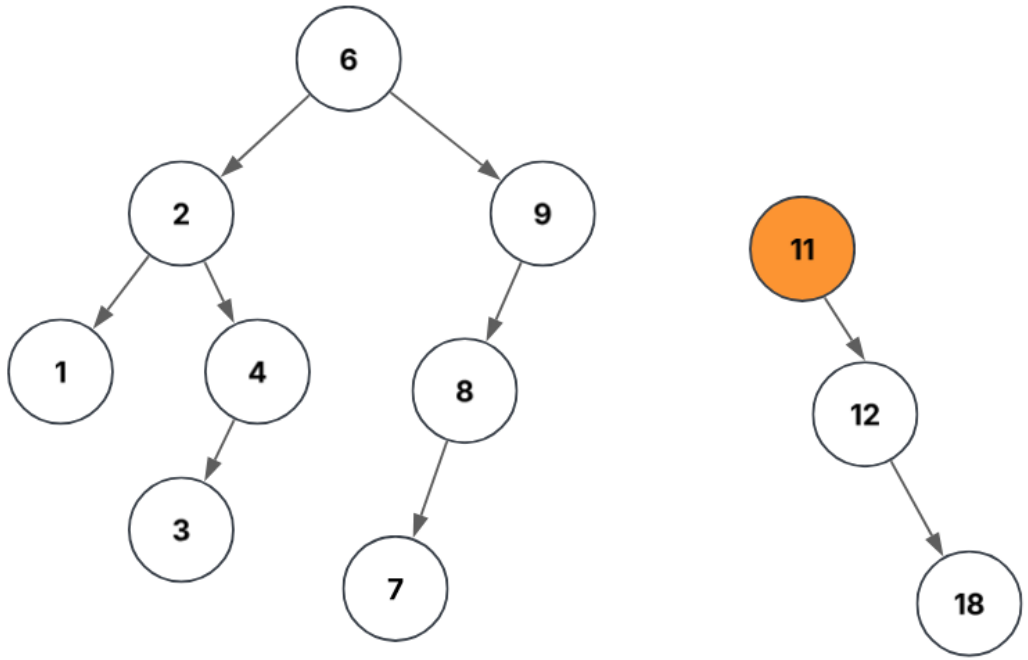


Se toma el tercer caso como sigue:

11 es el sucesor de 9, y se vuelve el padre de 11.

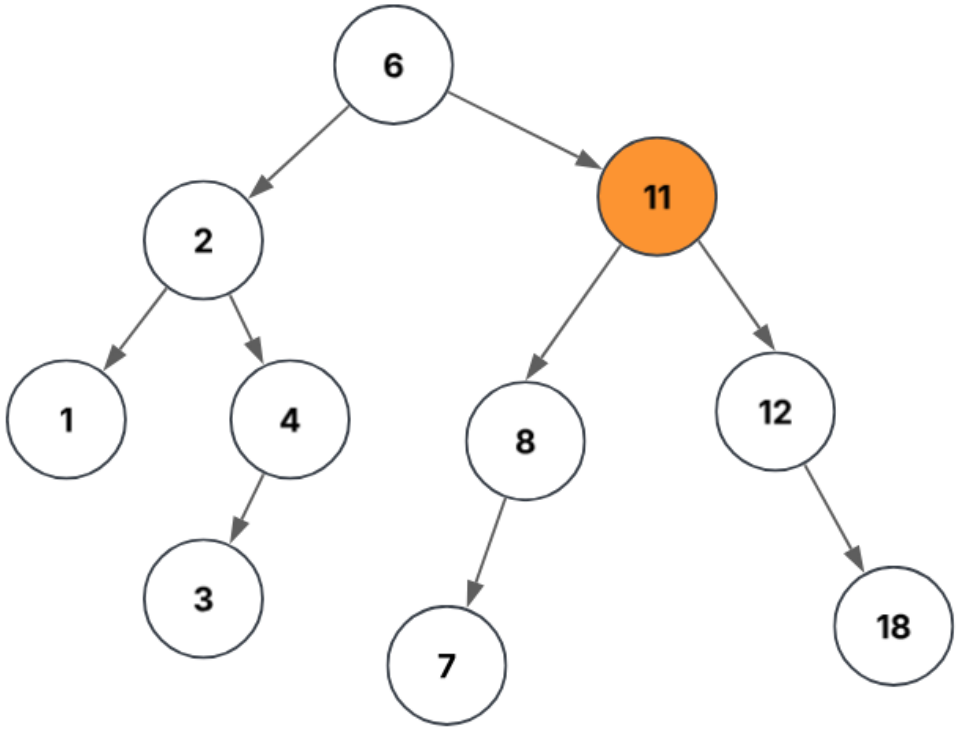
Eliminación

- Objetivo: eliminar el nodo 9



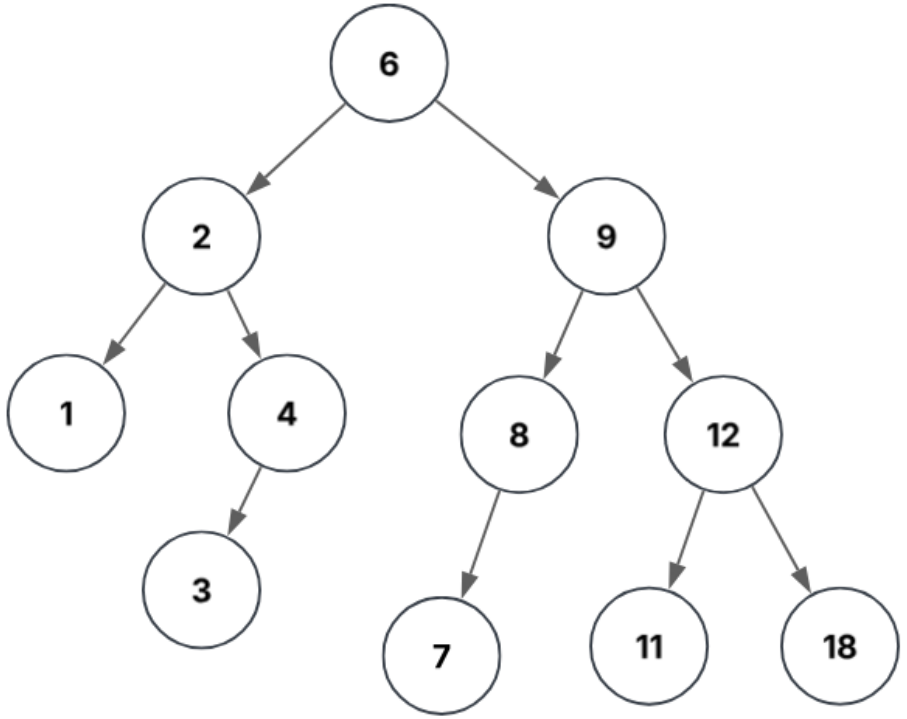
Eliminación

- Objetivo: eliminar el nodo 9



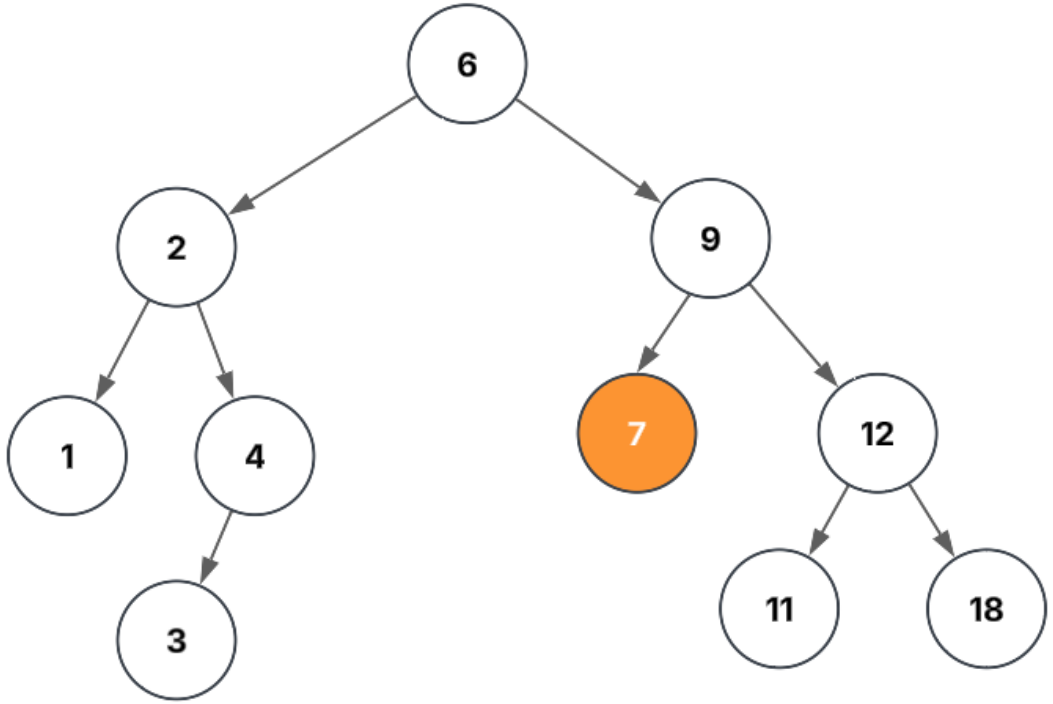
Eliminación

- Participaciones: eliminar el nodo 8, 2 y 12



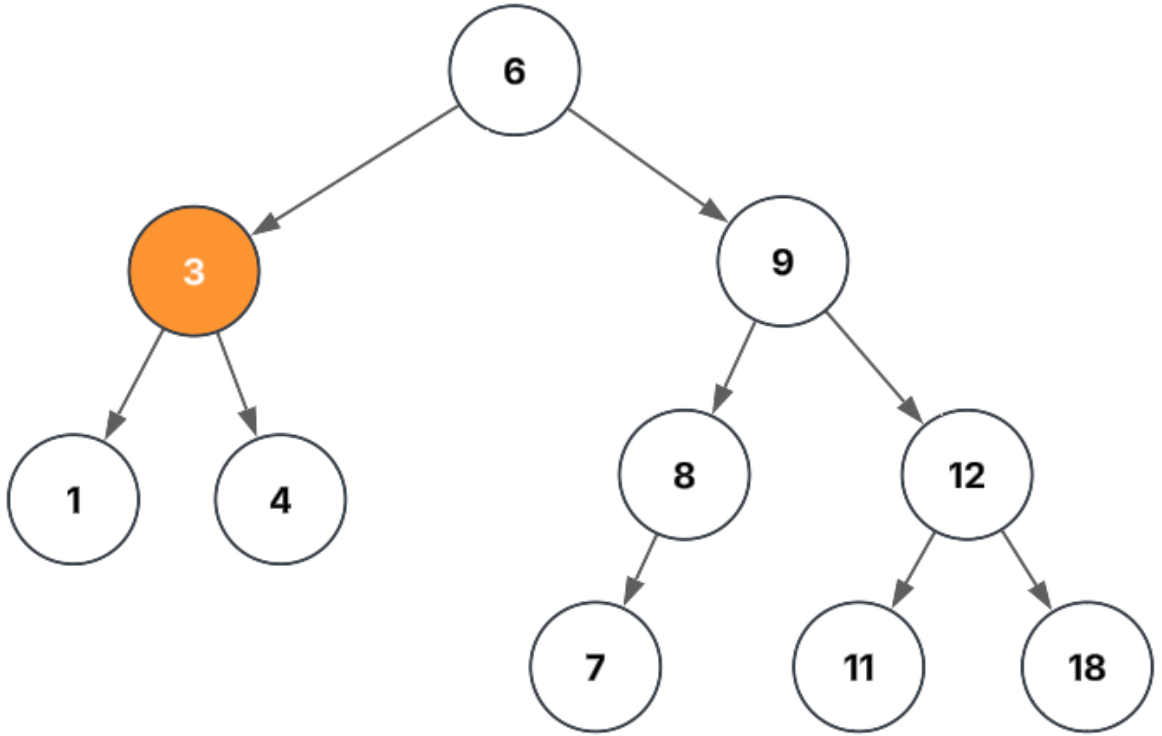
Eliminación

- Eliminando el nodo 8:



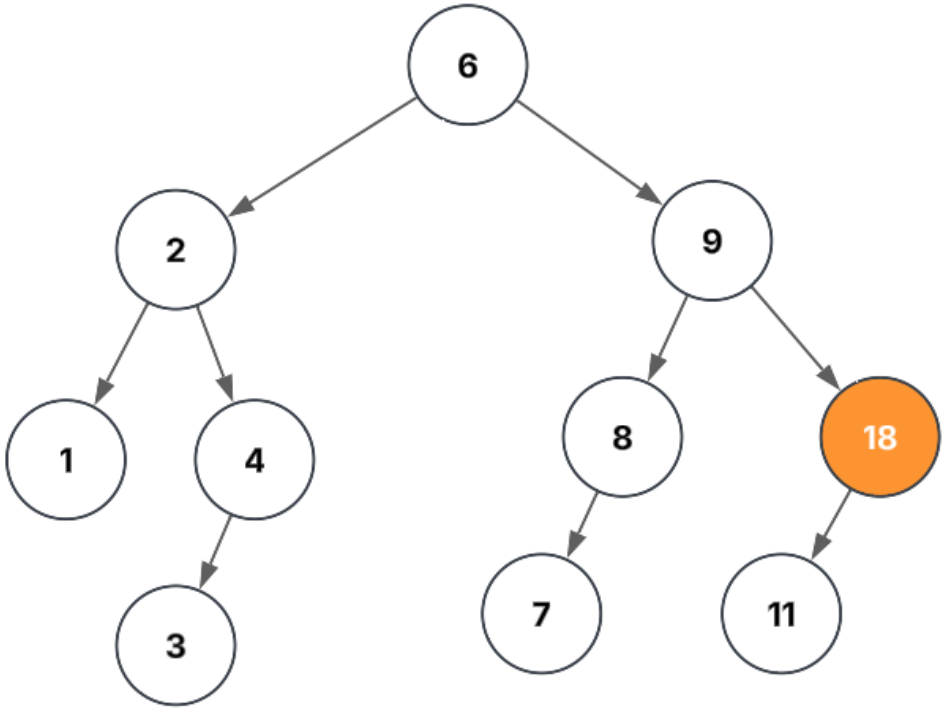
Eliminación

- Eliminando el nodo 2:



Eliminación

- Eliminando el nodo 12:

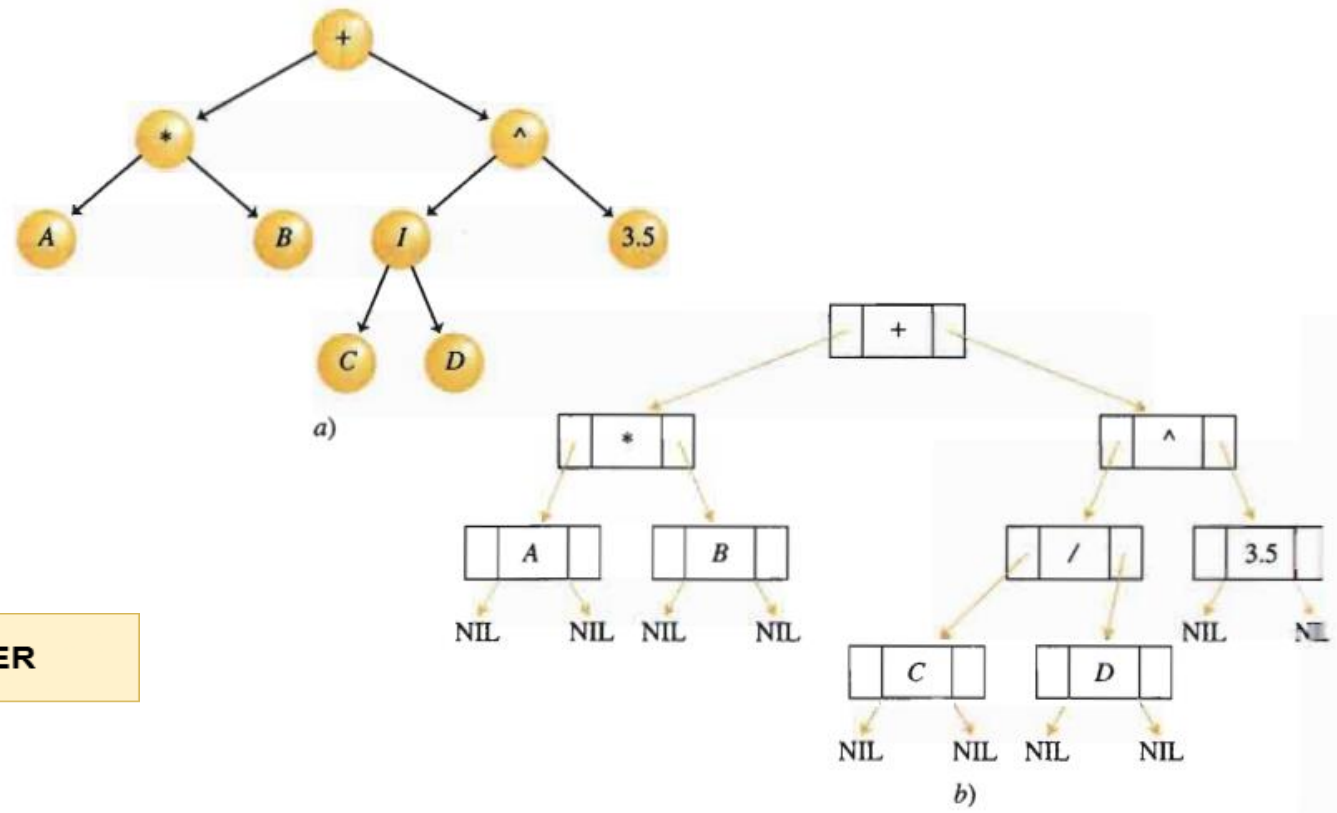


Representación de árboles binarios

Representación de un árbol binario en memoria

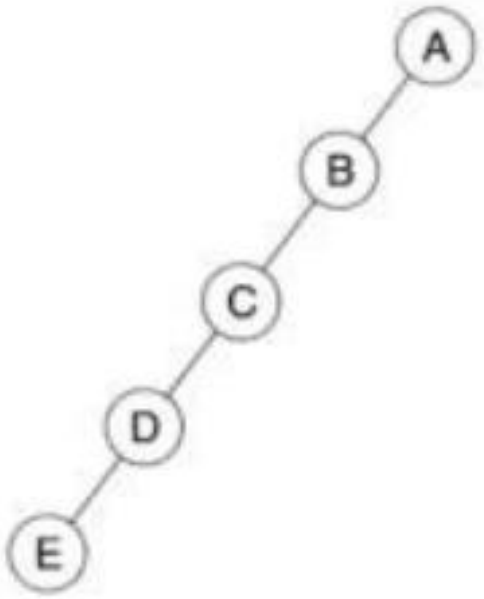
a) Árbol binario
b) Su representación en memoria

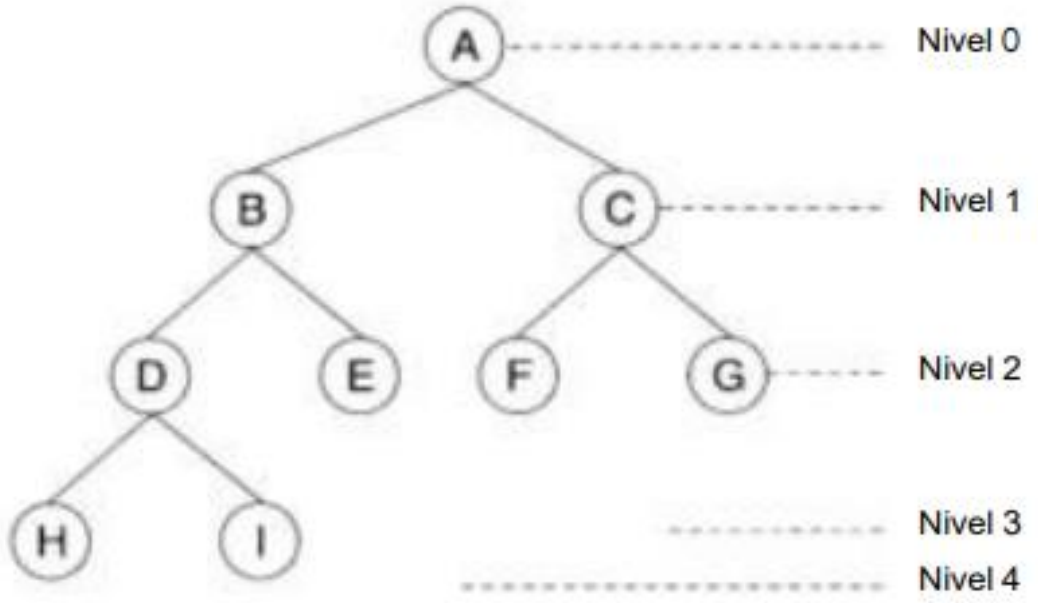
IZQ	INFO	DER



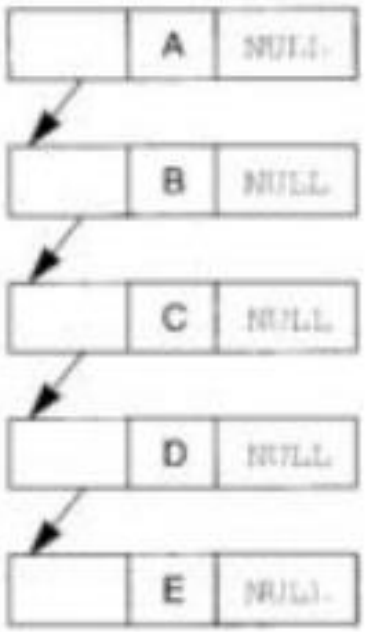
Ejercicios

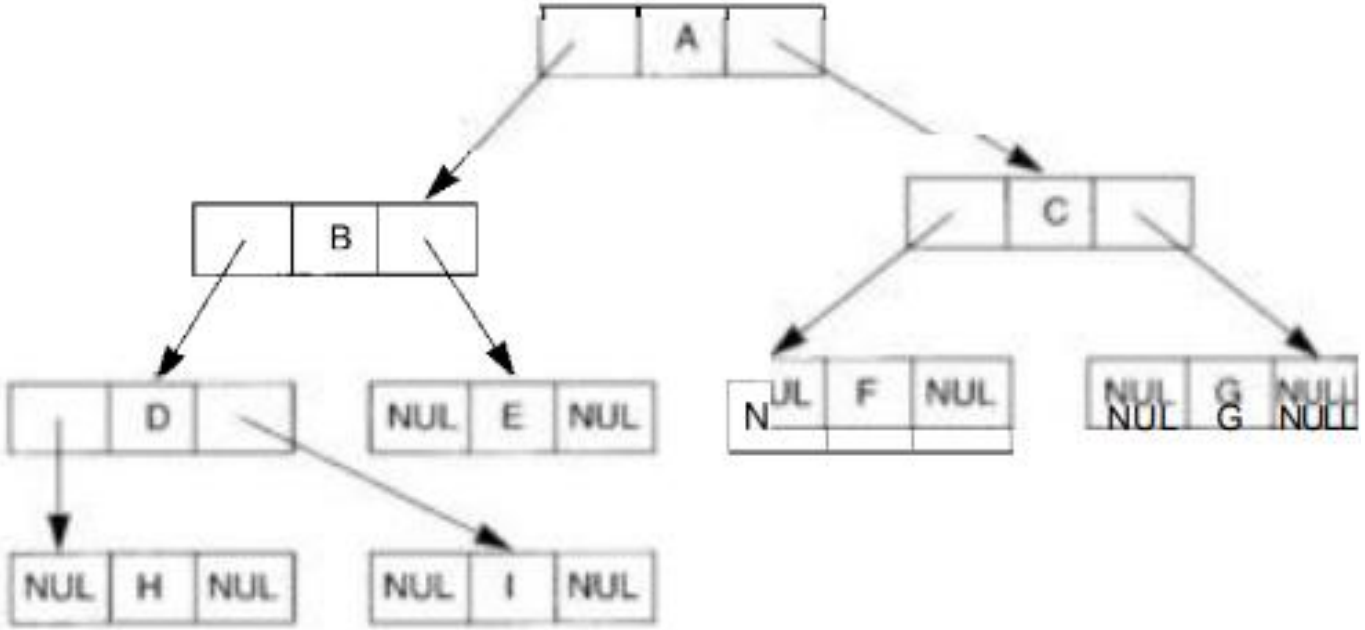
Representar la estructura en nodos de los dos árboles binarios de raíz **A:**





Ejercicios





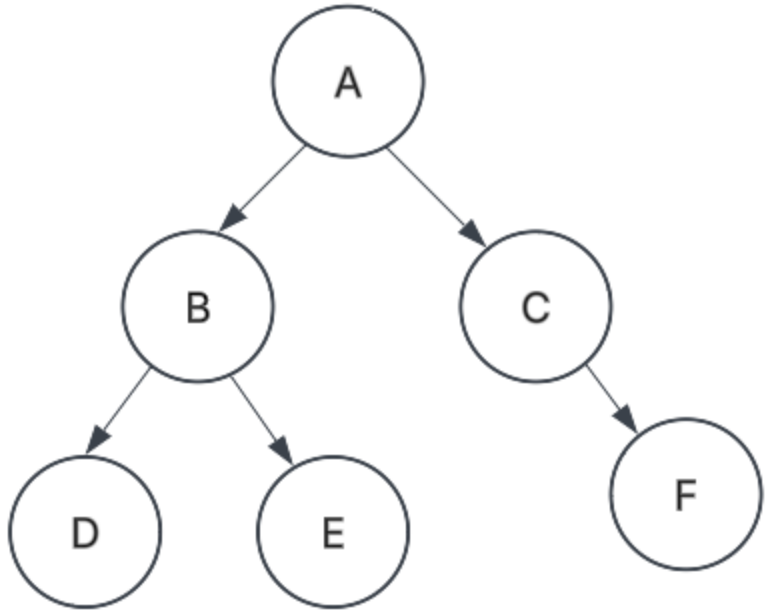
Existen varias formas de **representar los árboles binarios en la memoria,** y las más comunes son:

- Representación **estática** (usando arreglos).
- Representación con **nodos** (estructura de datos con punteros).
- Representación con **nodos encadenados**.

Representación estática (Arreglo)

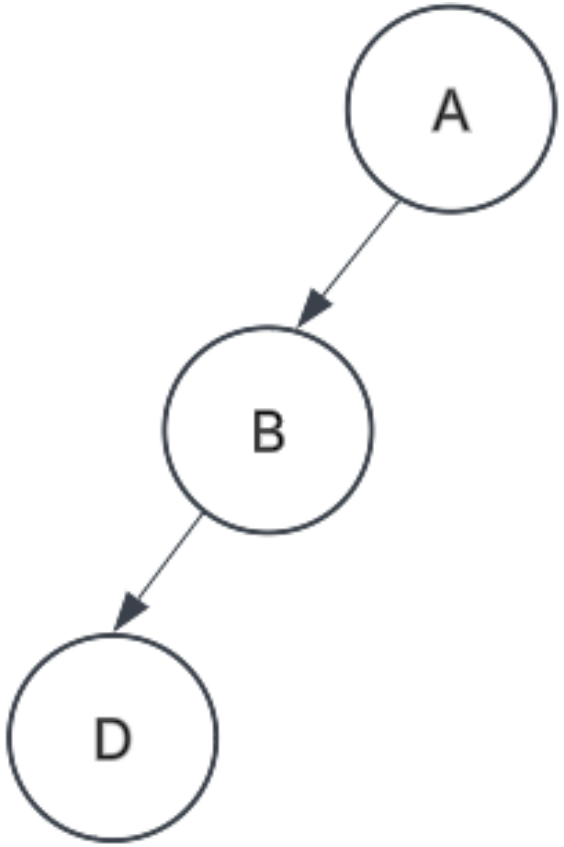
Se utiliza un arreglo unidimensional en el que cada posición representa un nodo. Si el nodo está en la posición i, entonces:

- Su hijo izquierdo está en la posición 2i+1.
- Su hijo derecho está en la posición 2i+2.
- La raíz siempre está en la posición 0.



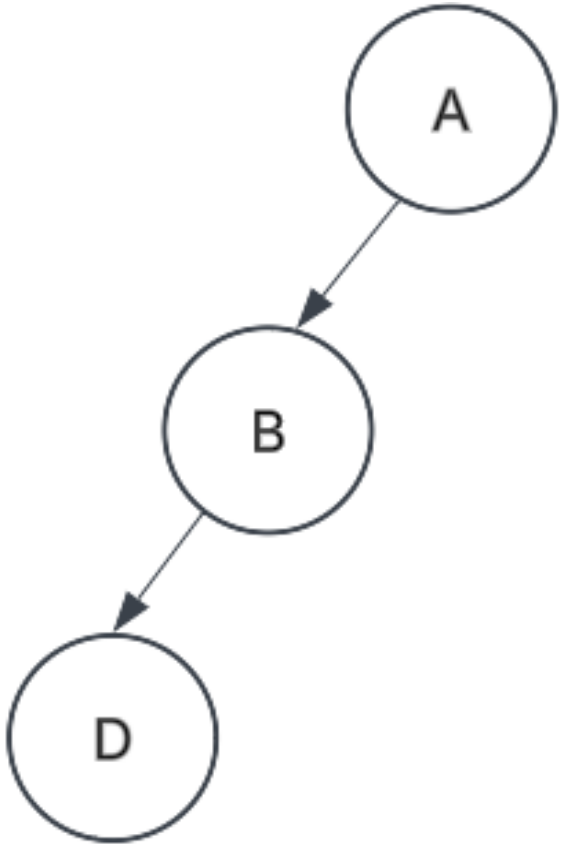
Índice	0	1	2	3	4	5	6
Nodo	A	B	C	D	E	NULL	F

Ejercicios



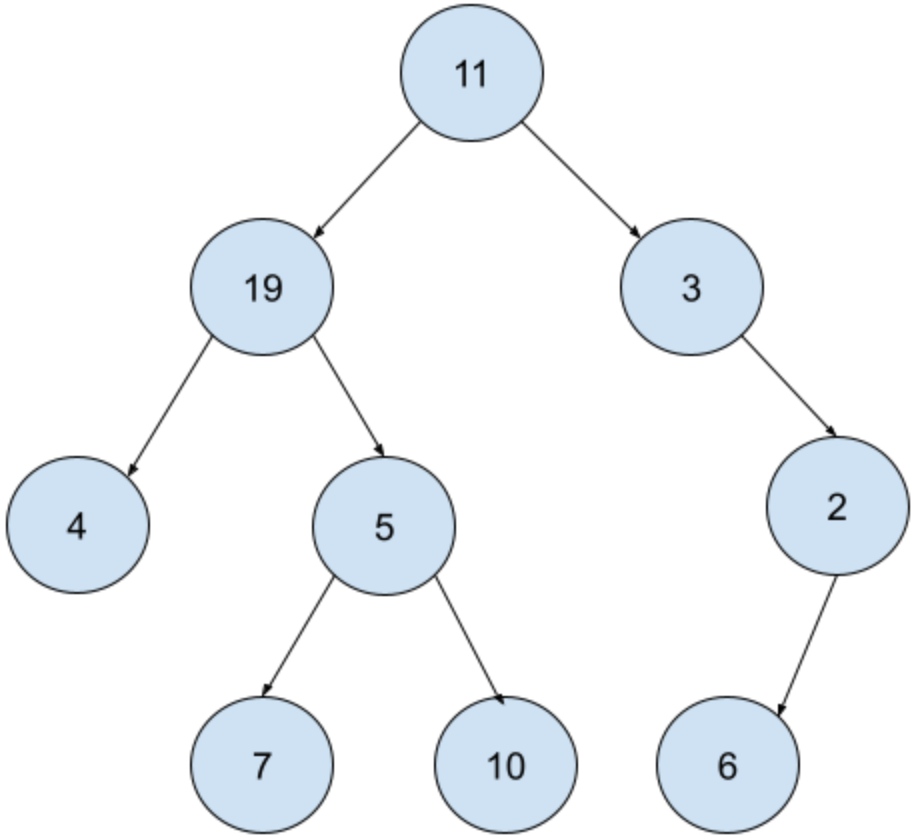
Índice	0	1	2	3	4	5	6
Nodo							

Ejercicios



Índice	0	1	2	3	4	5	6
Nodo	A	B	NULL	C	NULL	NULL	NULL

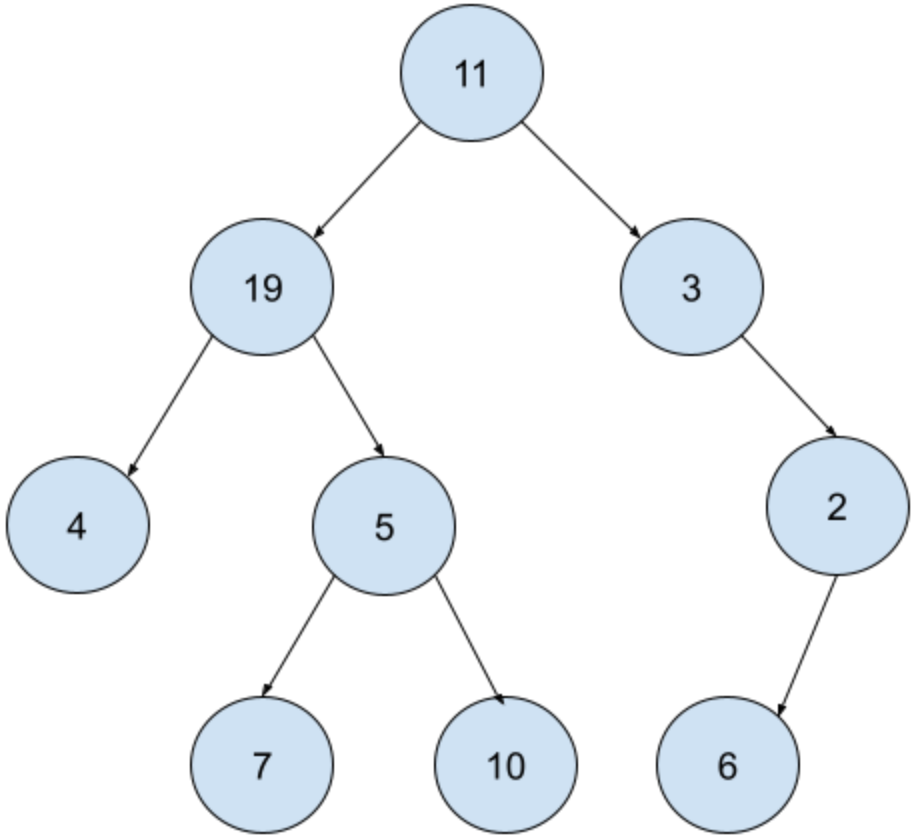
Ejercicios



Índice	0	1	2	3	4	5	6
Nodo							

7	8	9	10	11	12	13	14

Ejercicios



Índice	0	1	2	3	4	5	6
Nodo	11	19	3	4	5	NULL	2

7	8	9	10	11	12	13	14
NULL	NULL	7	10	NULL	NULL	6	NULL

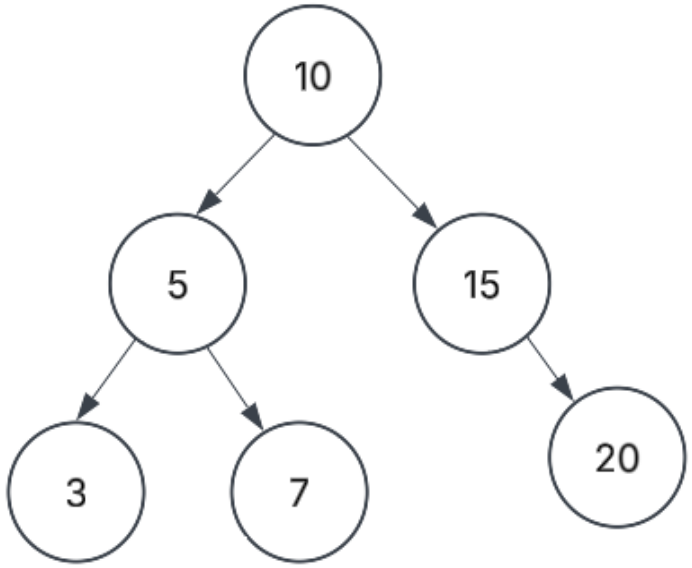
Representación con Nodos (Estructuras de Datos con Punteros)

Aquí, cada nodo es un objeto o estructura con tres componentes:

- Dato: Guarda el valor del nodo.
- Puntero izquierdo: Apunta al hijo izquierdo.
- Puntero derecho: Apunta al hijo derecho.

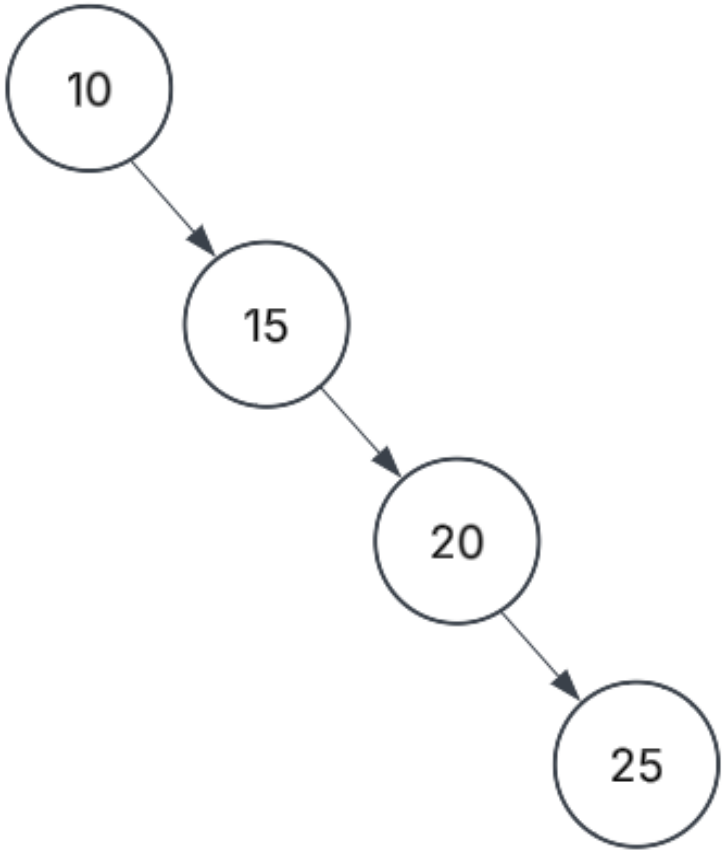
```c
struct Nodo {
    int dato;
    struct Nodo* izq;
    struct Nodo* der;
};

struct Nodo* nuevoNodo(int dato) {
    struct Nodo* nodo = (struct Nodo*)malloc(sizeof(struct Nodo));
    nodo->dato = dato;
    nodo->izq = nodo->der = NULL;
    return nodo;
}
```

Ejemplo

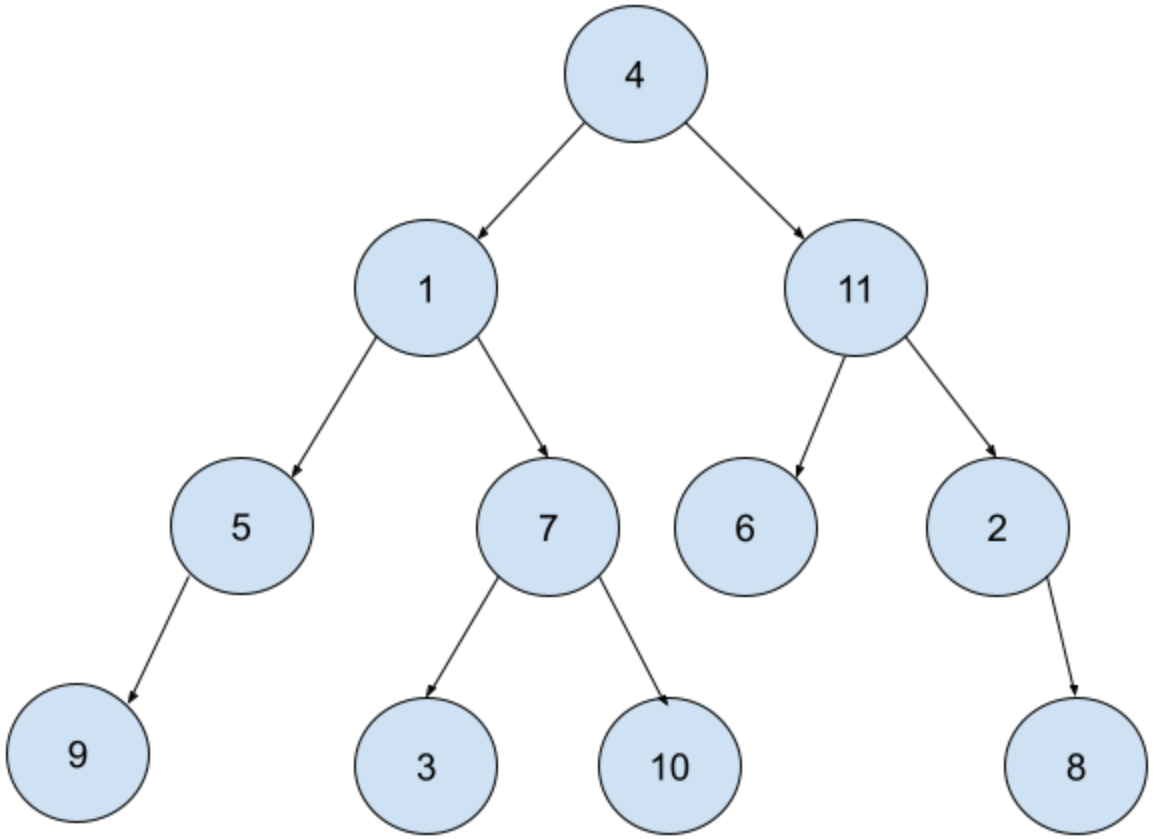



```
struct Nodo* raiz = nuevoNodo(10);
raiz->izq = nuevoNodo(5);
raiz->der = nuevoNodo(15);
raiz->izq->izq = nuevoNodo(3);
raiz->izq->der = nuevoNodo(7);
raiz->der->der = nuevoNodo(20);
```

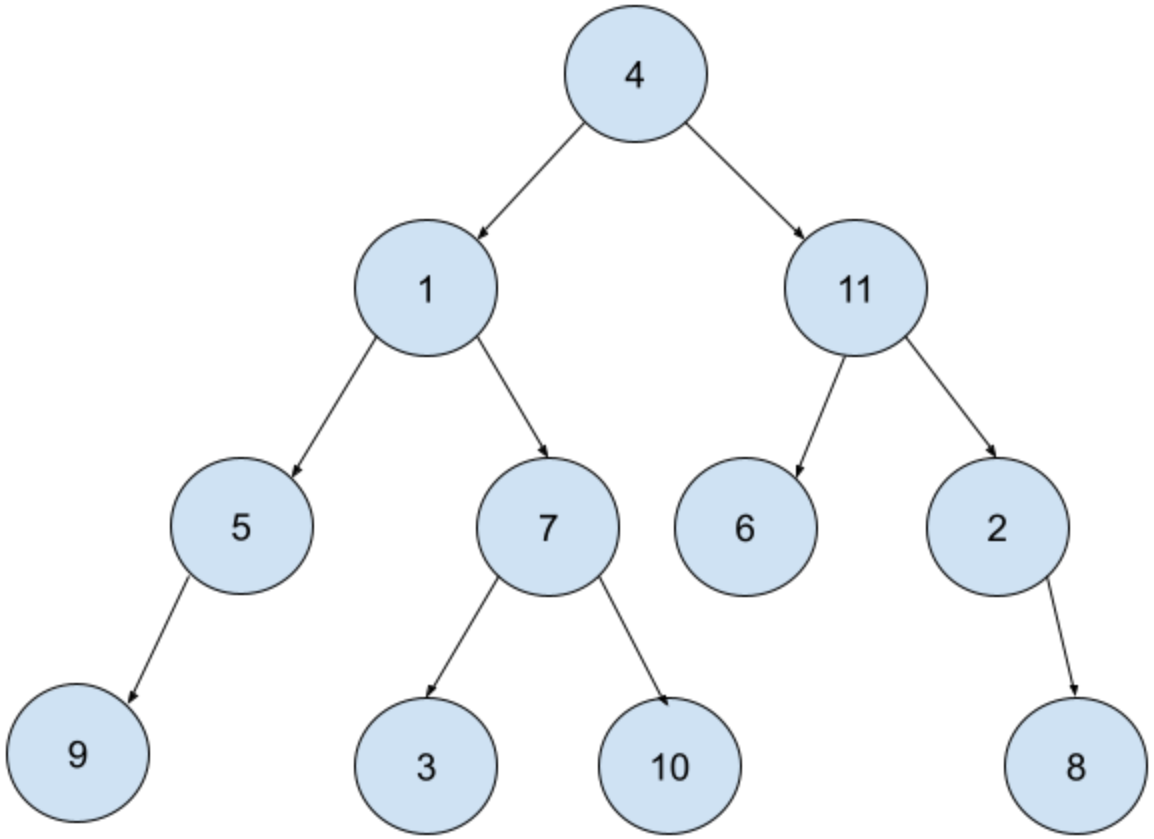
Ejercicios



Ejercicios



Ejercicios



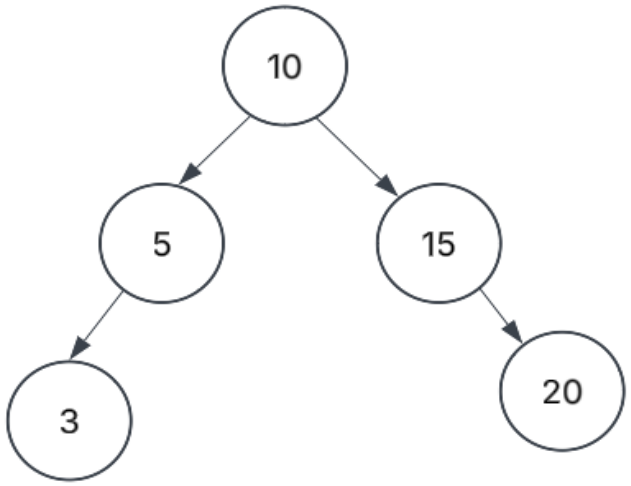

```
struct Nodo* raiz = nuevoNodo(4);
    raiz->izq = nuevoNodo(1);
    raiz->der = nuevoNodo(11);

    raiz->izq->izq = nuevoNodo(5);
    raiz->izq->der = nuevoNodo(7);

    raiz->der->izq = nuevoNodo(6);
    raiz->der->der = nuevoNodo(2);

    raiz->izq->izq->izq = nuevoNodo(9);
    raiz->izq->der->izq = nuevoNodo(3);
    raiz->izq->der->der = nuevoNodo(10);
    raiz->der->der->der = nuevoNodo(8);
```

Representación con Nodos Encadenados

Cada nodo tiene un número y un arreglo de punteros a hijos (hijos[0] e hijos[1]).

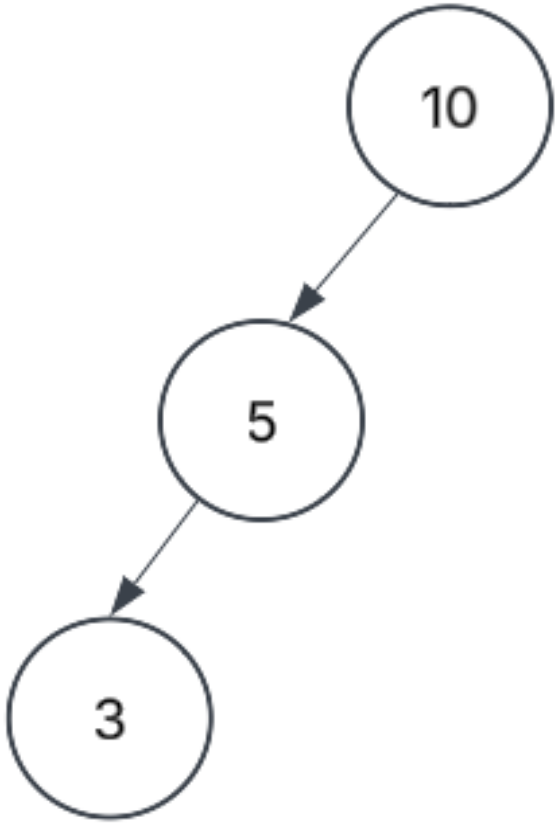
```c
#include <stdio.h>
#include <stdlib.h>

struct Nodo {
    int dato;
    struct Nodo* hijos[2];
};

struct Nodo* nuevoNodo(int dato) {
    struct Nodo* nodo = (struct Nodo*)malloc(sizeof(struct Nodo));
    nodo->dato = dato;
    nodo->hijos[0] = nodo->hijos[1] = NULL;
    return nodo;
}
```

Ejemplo

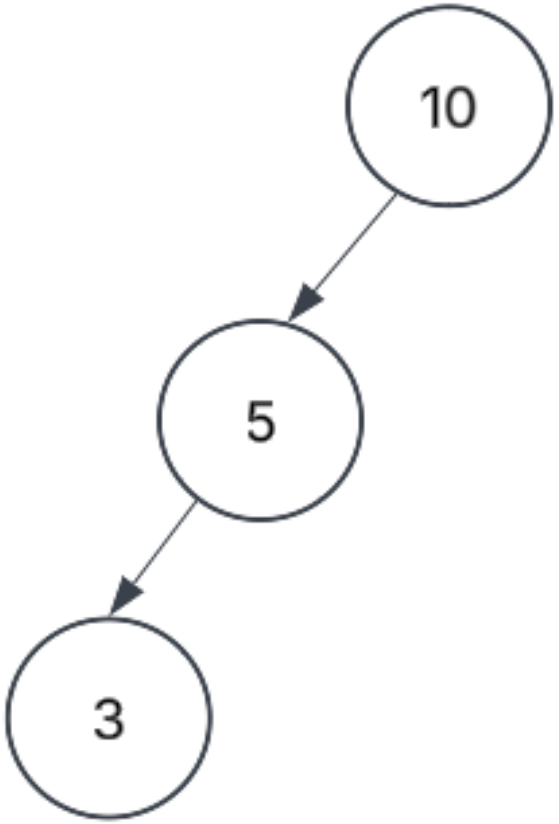



```
struct Nodo* raiz = nuevoNodo(10);
    raiz->hijos[0] = nuevoNodo(5);
    raiz->hijos[1] = nuevoNodo(15);
    raiz->hijos[0]->hijos[0] = nuevoNodo(3);
    raiz->hijos[1]->hijos[1] = nuevoNodo(20);
```

Ejercicios

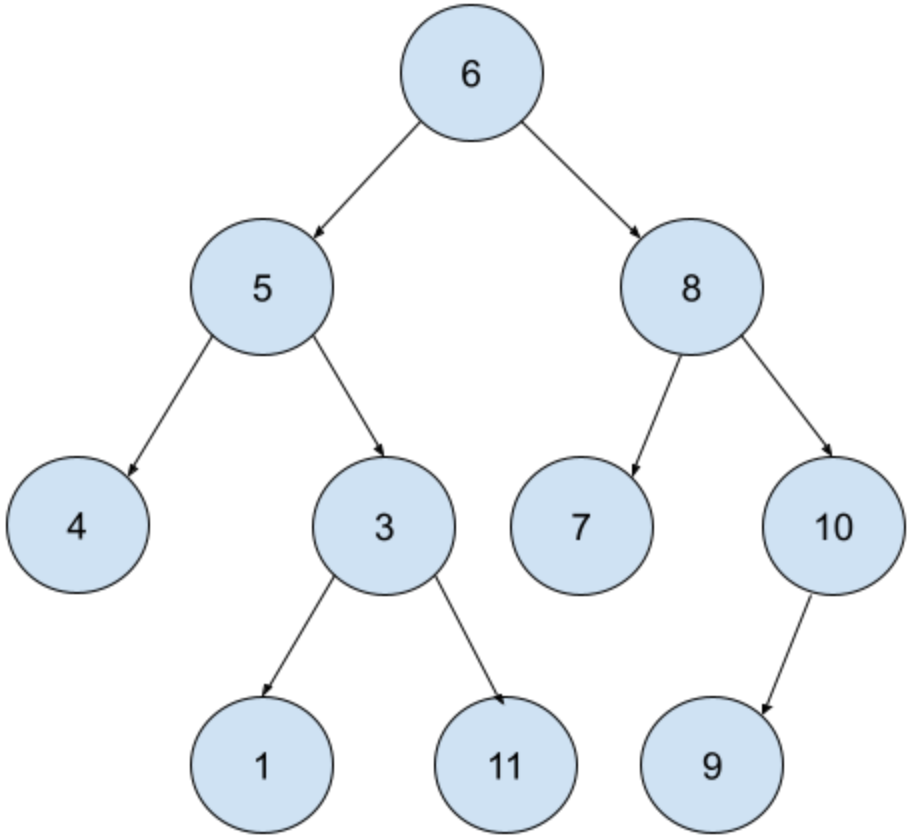


Ejercicios

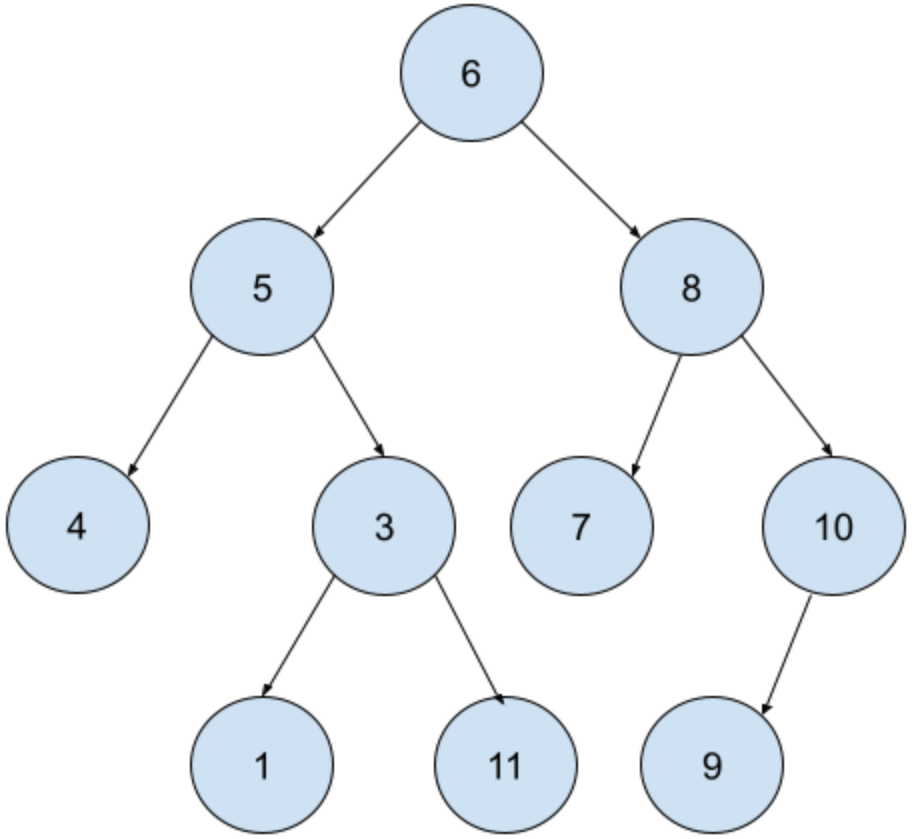



```
struct Nodo* raiz = nuevoNodo(10);
   raiz->hijos[0] = nuevoNodo(5);
   raiz->hijos[0]->hijos[0] = nuevoNodo(3);
```

Ejercicios

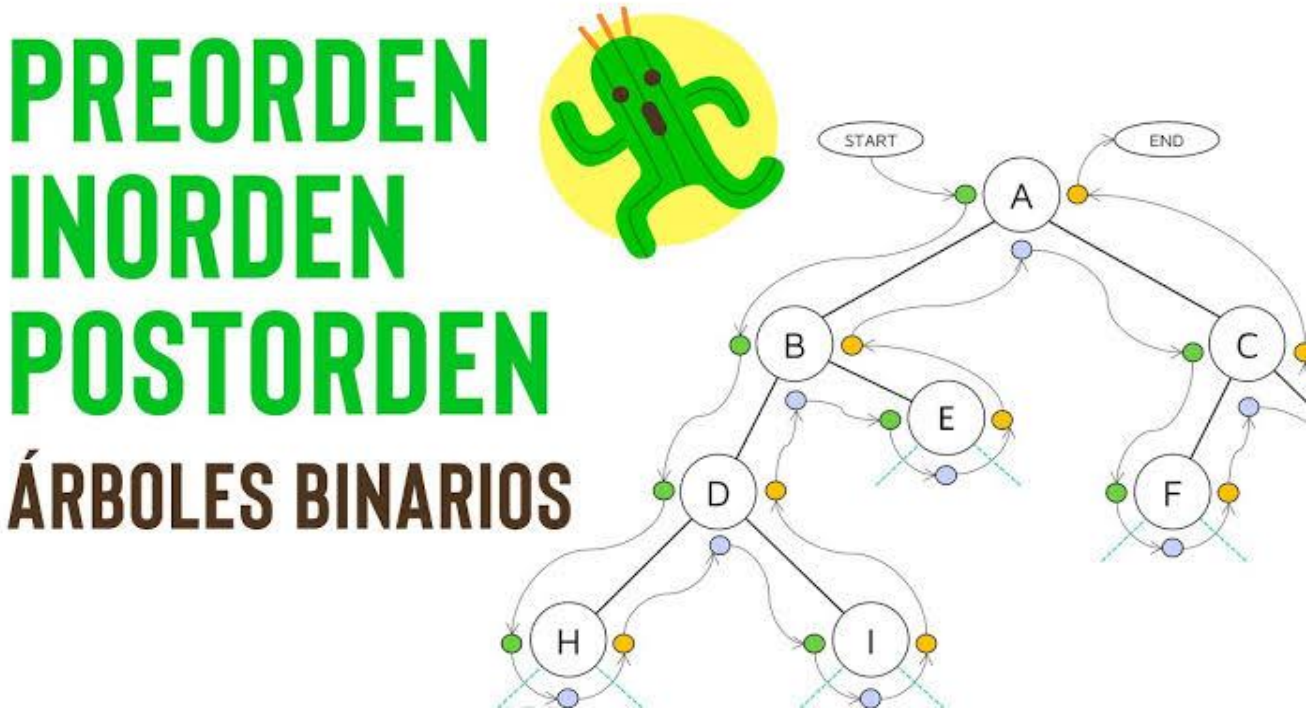


Ejercicios



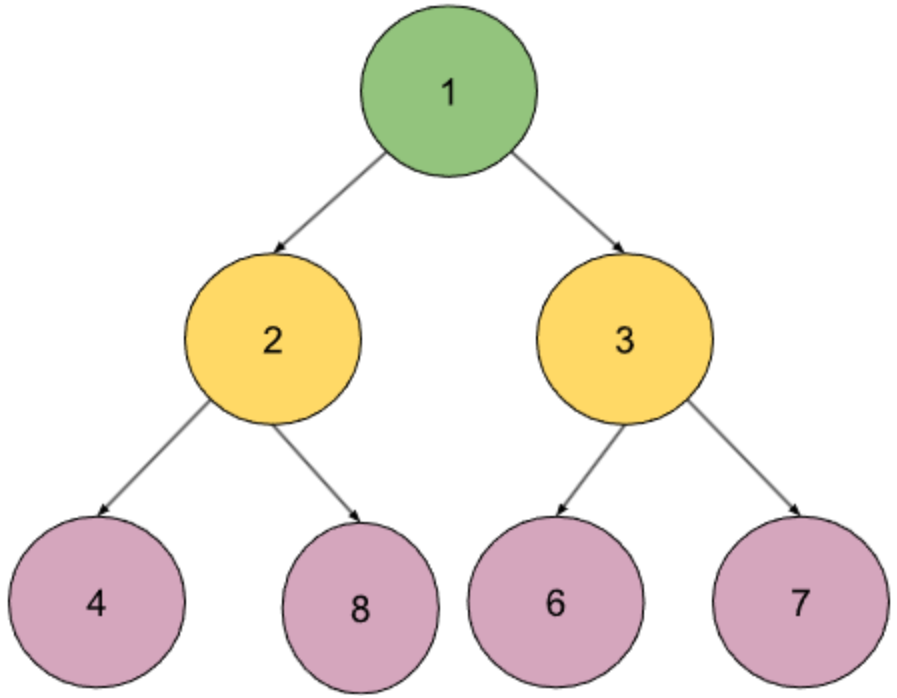

```
struct Nodo* raiz = nuevoNodo(6);
   raiz->hijos[0] = nuevoNodo(5);
   raiz->hijos[1] = nuevoNodo(8);

   raiz->hijos[0]->hijos[0] = nuevoNodo(4);
   raiz->hijos[0]->hijos[1] = nuevoNodo(3);

   raiz->hijos[1]->hijos[0] = nuevoNodo(7);
   raiz->hijos[1]->hijos[1] = nuevoNodo(10);

   raiz->hijos[0]->hijos[0]->hijos[0] = nuevoNodo(1);
   raiz->hijos[0]->hijos[1]->hijos[0] = nuevoNodo(11);
   raiz->hijos[1]->hijos[1]->hijos[0] = nuevoNodo(9);
```

Creación de árboles binarios a partir de los recorridos

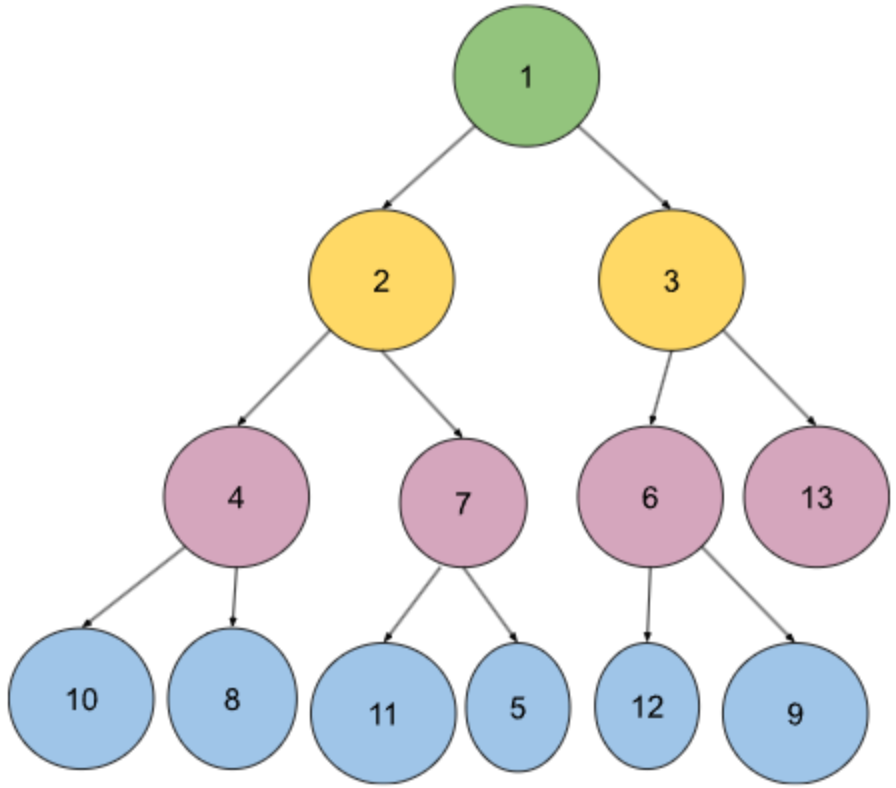


Creación de árbol binario a partir del recorrido Pre-Orden e In-Orden

Preorden: [1, 2, 4, 8, 3, 6, 7]
Inorden: [4, 2, 8, 1, 6, 3, 7]

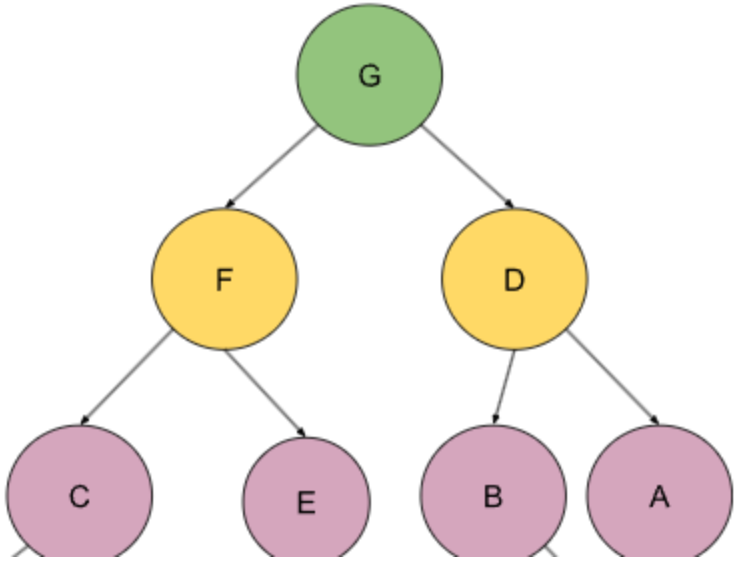


Preorden: [1, 2, 4, 10, 8, 7 , 11, 5, 3, 6, 12, 9, 13]
Inorden: [10, 4, 8, 2, 11, 7 , 5, 1, 12 , 6, 9, 3, 13]

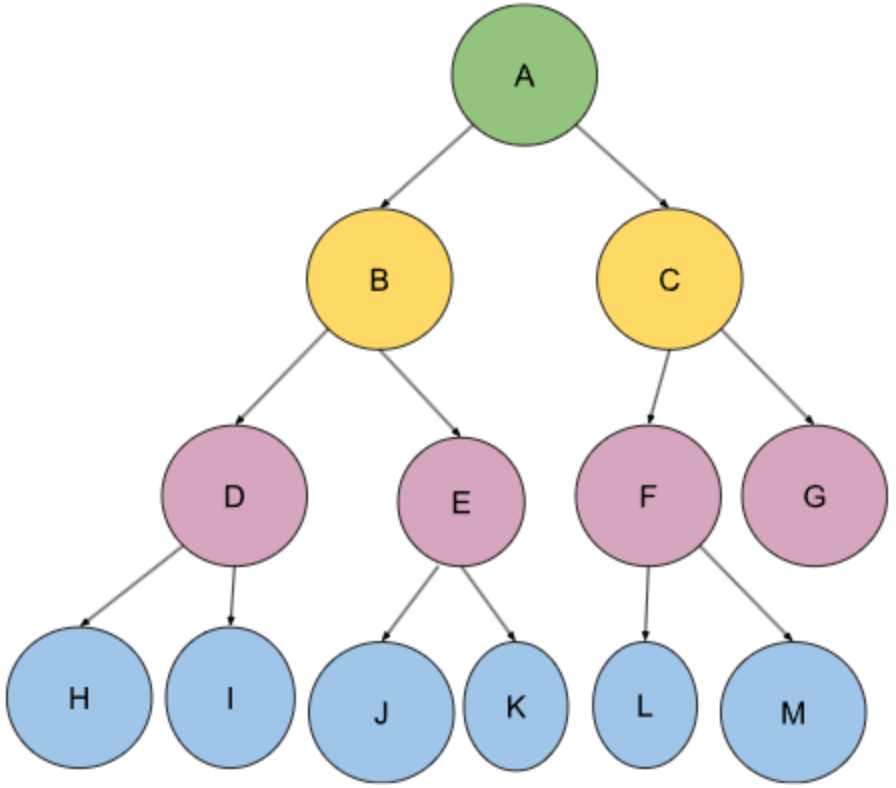


Creación de árbol binario a partir del recorrido In-Orden

C, F, E, G, B, D, A

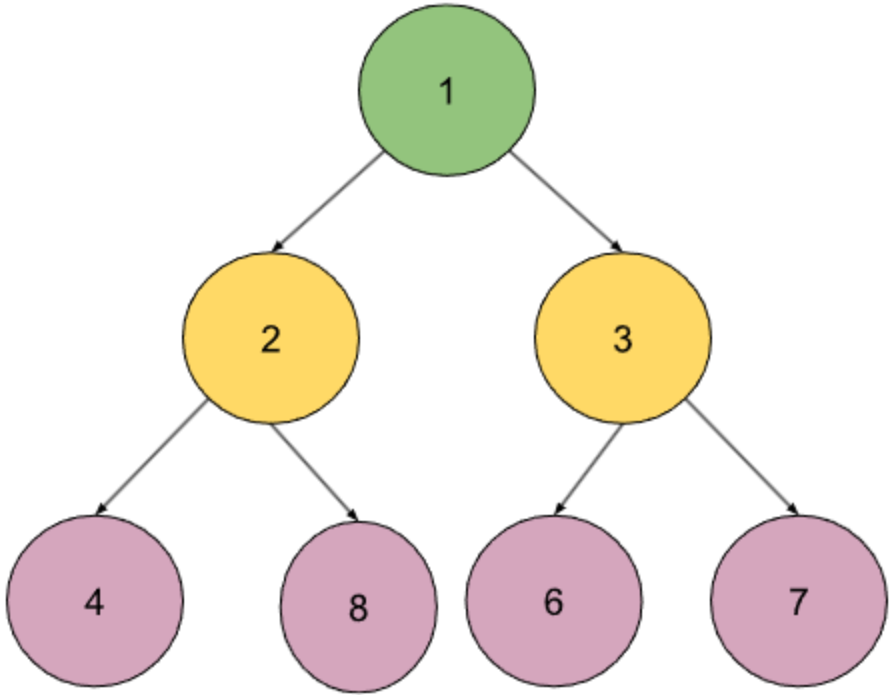


H, D, I, B, J, E, K, A, L, F, M, C, G

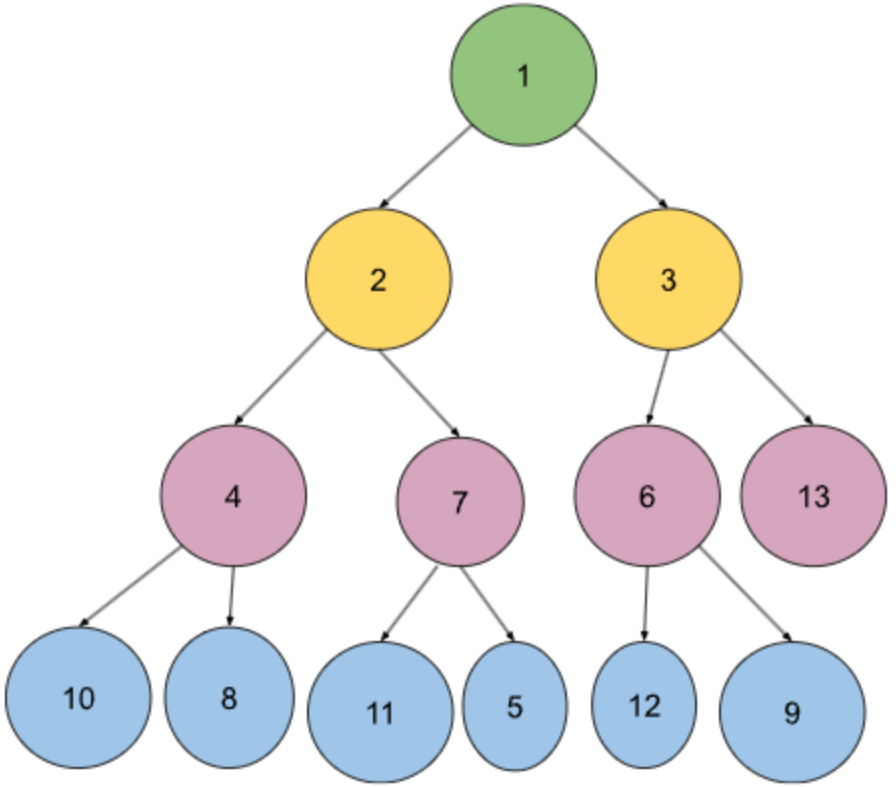


Creación de árbol binario a partir del recorrido Pos-Orden

Postorden: [4, 5, 2, 6, 7, 3, 1]
Inorden: [4, 2, 5, 1, 6, 3, 7]

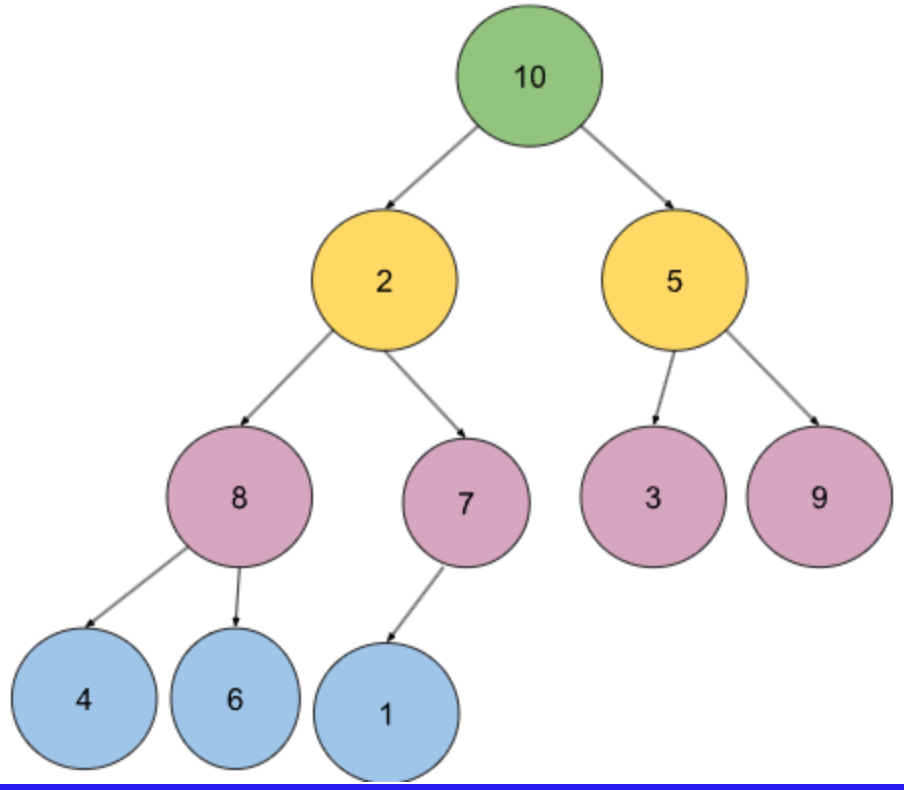


Postorden: [10, 8, 4, 11, 5, 7, 2, 12, 9, 6, 13, 3, 1]

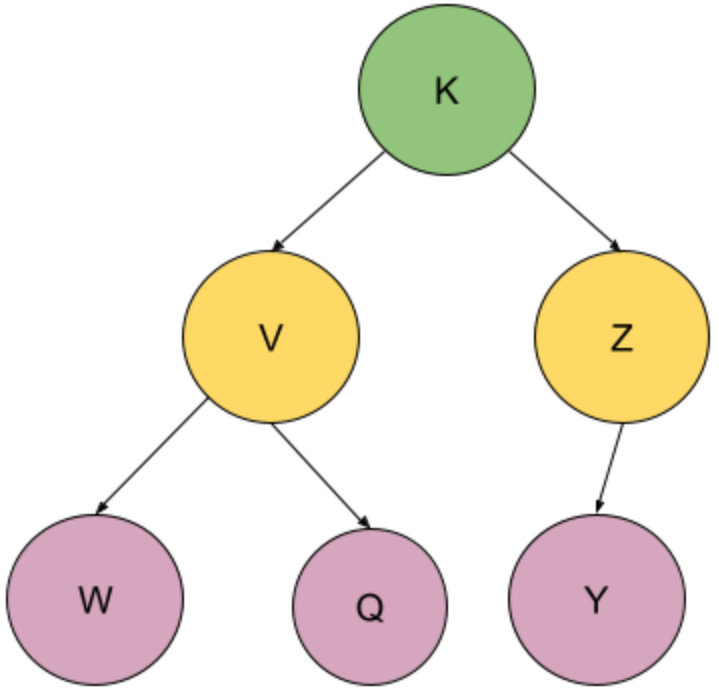


Creación de árbol binario a partir del recorrido por Nivel

10, 2, 5, 8, 7, 3, 9, 4, 6, 1]



[K, V, Z, W, Q, Y]

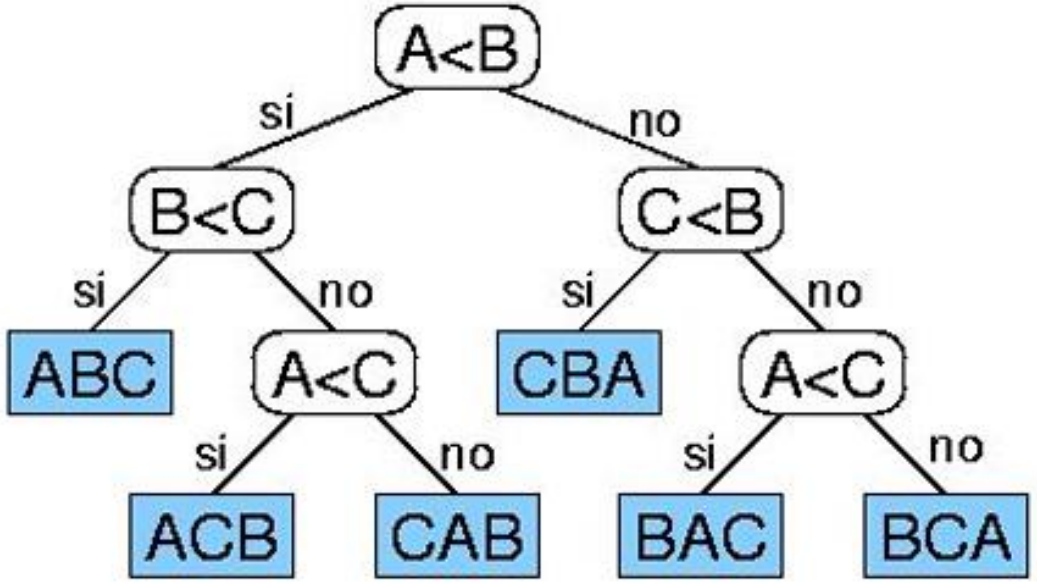


Aplicaciones

Árboles de decisión

- Nodos internos: preguntas con respuestas sí/no
- Nodos hojas: decisiones

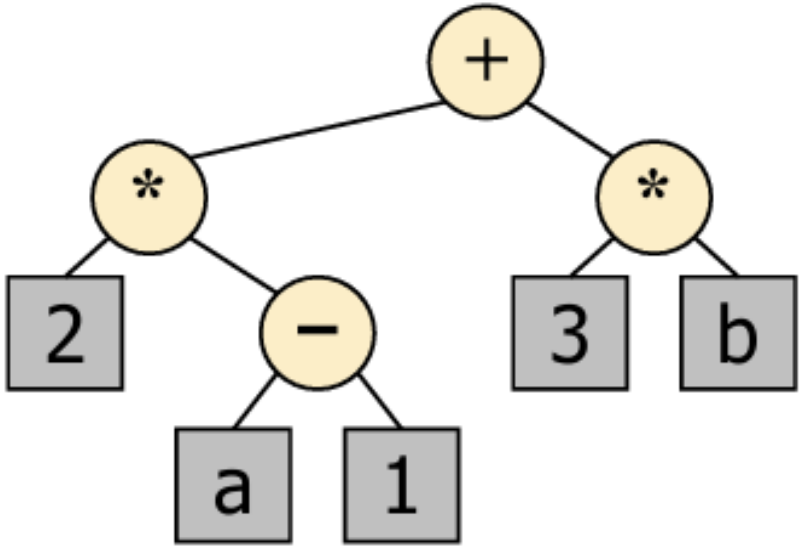
Ejemplo de árbol de decisión para ordenar tres números:



Aplicaciones

Representación de expresiones aritméticas

2*(a-1)+3*b



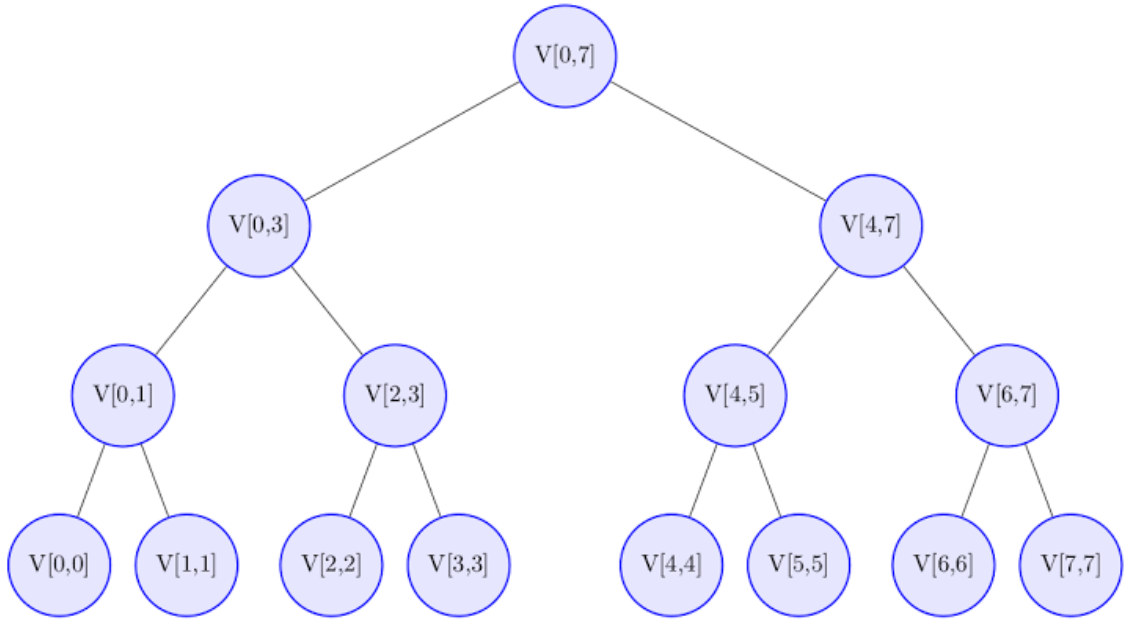
•Nodos internos: operadores
•Nodos hojas: operandos

Ejemplo:

Aplicaciones

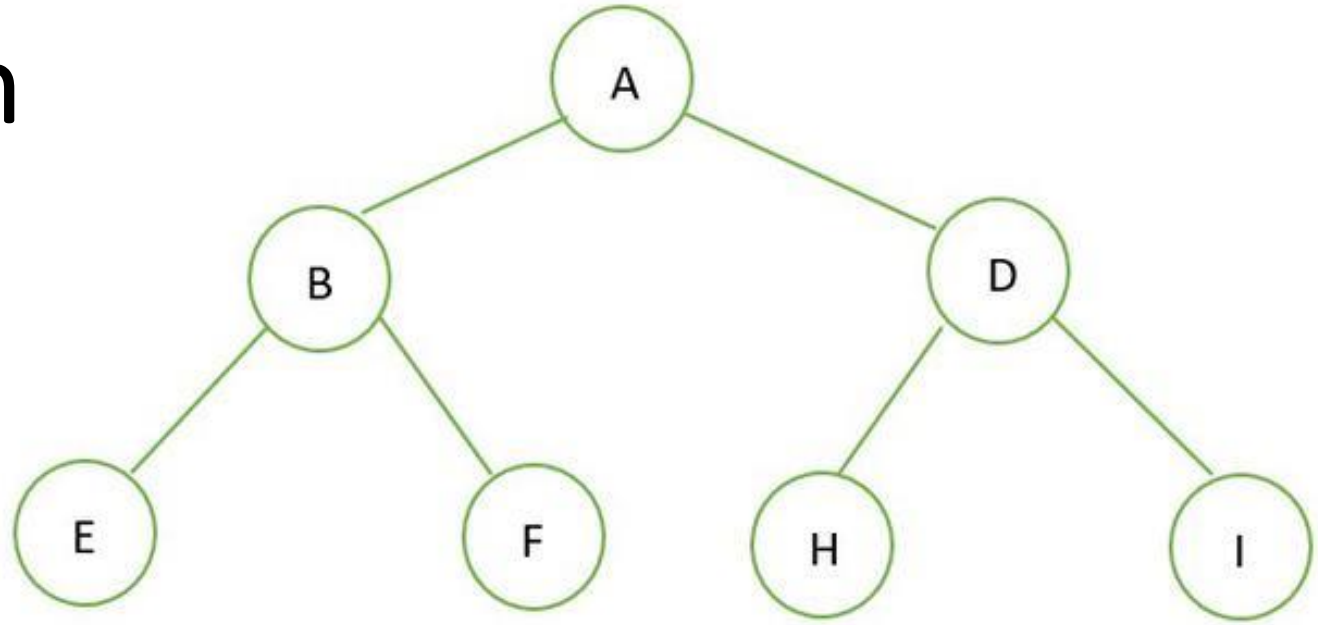
Árboles de segmentos

Es una estructura de datos en forma de árbol binario que se utiliza para almacenar intervalos o segmentos. Cada nodo del árbol representa un segmento del arreglo, y las hojas representan elementos individuales del arreglo.



Ejemplo:

Conclusión



Desde árboles generales hasta árboles binarios de búsqueda y árboles de expresión, cada tipo tiene características y aplicaciones específicas que facilitan tareas como la búsqueda, inserción, eliminación, y evaluación de expresiones. La capacidad de representar árboles en memoria de diferentes maneras también destaca su adaptabilidad a diversas necesidades computacionales. En conclusión, los árboles son una herramienta esencial en el diseño de algoritmos y estructuras de datos, proporcionando soluciones elegantes y eficientes a problemas complejos en la programación y el manejo de datos.

Bibliografía

- Lecture 8 - Binary Search Trees (2025). Consultado 22 Febrero 2025, De https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture8.pdf
- Joyanes Aguilar, L., & Zahonero Martínez, I. (s.f.). Programación en C: Metodología, algoritmos y estructura de datos. McGraw-Hill.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms, fourth edition. MIT Press.
- Aprendiendo sin espinas. 12 de septiembre de 2022. Algoritmos PREORDEN, INORDEN y POSTORDEN en 3 minutos. Youtube. https://www.youtube.com/watch?v=Jo2euX89Oz8
- Laura, V. (s. f.). ARBOLES DE EXPRESIONES. https://vilmalauramoraleszunun.blogspot.com/2019/08/arboles-de-expresiones.html
- Danny, P. (2010). “¿Cuáles son las aplicaciones de los árboles binarios?”. Recuperado de iteramos. URL: https://www.iteramos.com/pregunta/5134/cuales-son-las-aplicaciones-de-los-arboles-binarios