CURSO DE PROGRAMACIÓN COMPETITIVA

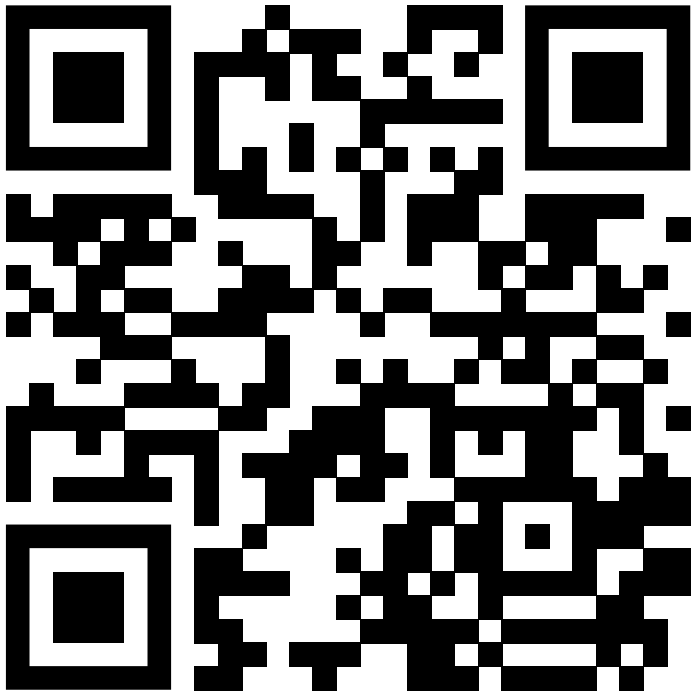
ESTRUCTURAS DE DATOS II

CURSO DE PROGRAMACIÓN COMPETITIVA URJC - 2025

Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Sergio Salazar (sergio.salazar@urjc.es)
- Adaya Ruiz (am.ruiz.2020@alumnos.urjc.es)
- Eva Gómez (e.gomezf.2020@alumnos.urjc.es)
- **Lucas Martín** (lucas.martin@urjc.es)
- Iván Penedo (ivan.penedo@urjc.es)
- Alicia Pina (alicia.pina@urjc.es)
- Sara García (sara.garciar@urjc.es)
- Raúl Fauste (r.fauste.2020@alumnos.urjc.es)
- **Alejandro Mayoral** (a.mayoralg.2020@alumnos.urjc.es)
- David Orna (de.orna.2020@alumnos.urjc.es)

Clasificatorio Ada Byron 2025



Estructuras de Datos 2

Estructuras de Datos Principales

- Listas
- Arrays
- Strings
- Pilas
- Colas
- Set

Estructuras de Datos Principales

- Listas
- Arrays
- Strings
- Pilas
- Colas
- Set

- **Mapas**
- **Colas de Prioridad**
- **Árboles**

Repaso ED1

LISTAS

- **Estructura de datos dinámica** que permite almacenar una **secuencia de elementos**.
- Puede contener distintos tipos de datos. !Python no te avisa!

```
1   # Definición de una lista
2   mi_lista = [1, 2, 3, 4, 5]
3
4   print(mi_lista)  # Salida: [1, 2, 3, 4, 5]
5
```

LISTAS

- Búsqueda
  - Mediante índice: O(1)
  - Mediante Valor: **O(n)**
- Eliminación **O(n)**
- Añadir al final O(1)
- Insertar en medio **O(n)**
- Modificar O(1)

```
 6  mi_lista[3] #Índice
 7
 8  if valor in mi_lista: #Valor
 9      print("Encontrado")
10
11
12  mi_lista.pop(indice) #Índice
13  mi_lista.remove(valor) #Valor
14
15  mi_lista.append(valor)
16  mi_lista.insert(indice, valor)
17
18
19  mi_lista[indice] = valor
20
```

STRINGS

- Listas inmutables de caracteres.
- Mismo acceso que listas
- Concatenación **O(n)**

```
mi_string = "hola"

elemento_string = mi_string[3]

concatenado_str = mi_string + " mundo"


mi_string.upper()    # Mayúsculas
mi_string.lower()    # Minúsculas
mi_string.replace("o", "O")
mi_string.split(" ")  # Dividir
```

STRINGS

- Modificar strings mejor con listas.

```
mi_lista = list(mi_string)
mi_lista.append("!")    # Añadir
mi_lista.pop()          # Eliminar último
mi_lista.insert(1, "E")  # Insertar
mi_lista.remove("o")    # Eliminar valor
mi_lista.reverse()      # Invertir
mi_lista.sort()         # Ordenar
```

PILAS (STACKS)

- Estructura **LIFO** (Last In, First Out).

- **collections.deque :**
  - Doble **lista enlazada**
  - Más eficiente que **list.**

```
2   from collections import deque
3
4   pila = deque()
5
```

- Aplicaciones:
  - Backtracking (DFS).
  - Paréntesis balanceados, evaluación de expresiones.
  - Historial de deshacer/rehacer en editores.

PILAS (STACKS)

- Complejidad Operaciones:
- Apilar: **O(1)**
- Desapilar (Pop): **O(1)**
- Ver cima (Top): **O(1)**
- Comprobar vacía: **O(1)**

```
2   from collections import deque
3
4   pila = deque()
5
6   pila.append(10) # Apilar
7   pila.pop()      # Desapilar
8
9   top = pila[-1]
10  esta_vacia = len(pila) == 0
```

COLAS (QUEUES)

- Estructura **FIFO** (First In, First Out).

- **collections.deque :**
  - Doble **lista enlazada**
  - Más eficiente que **list.**

```
2  from collections import deque
3
4  cola = deque()
5
```

- Aplicaciones:
  - BFS en grafos.
  - Orden de llegada.
  - Colas de impresión, eventos, supermercados.

COLAS (QUEUES)

- Complejidad Operaciones:
- Encolar: **O(1)**
- Desencolar: **O(1)**
- Ver frente (Front): **O(1)**
- Comprobar vacía: **O(1)**

```
2   from collections import deque
3
4   cola = deque()
5
6   cola.append(10)    # Encolar
7   cola.popleft()     # Desencolar
8
9   frente = cola[0]
10  esta_vacia = len(cola) == 0
```

COJUNTOS (SETS)

- Estructura de datos que almacena elementos **únicos**.
- Basado en una tabla **hash**.
- No está ordenado, sin acceso por índices.
- Aplicaciones:
  - **Búsqueda** rápida de elementos.
  - Eliminación de **duplicados**.
  - Operaciones de **conjuntos** (unión, intersección, diferencia).

```
2   set1 = set()
3   set2 = {30, 40, 50}
4
```

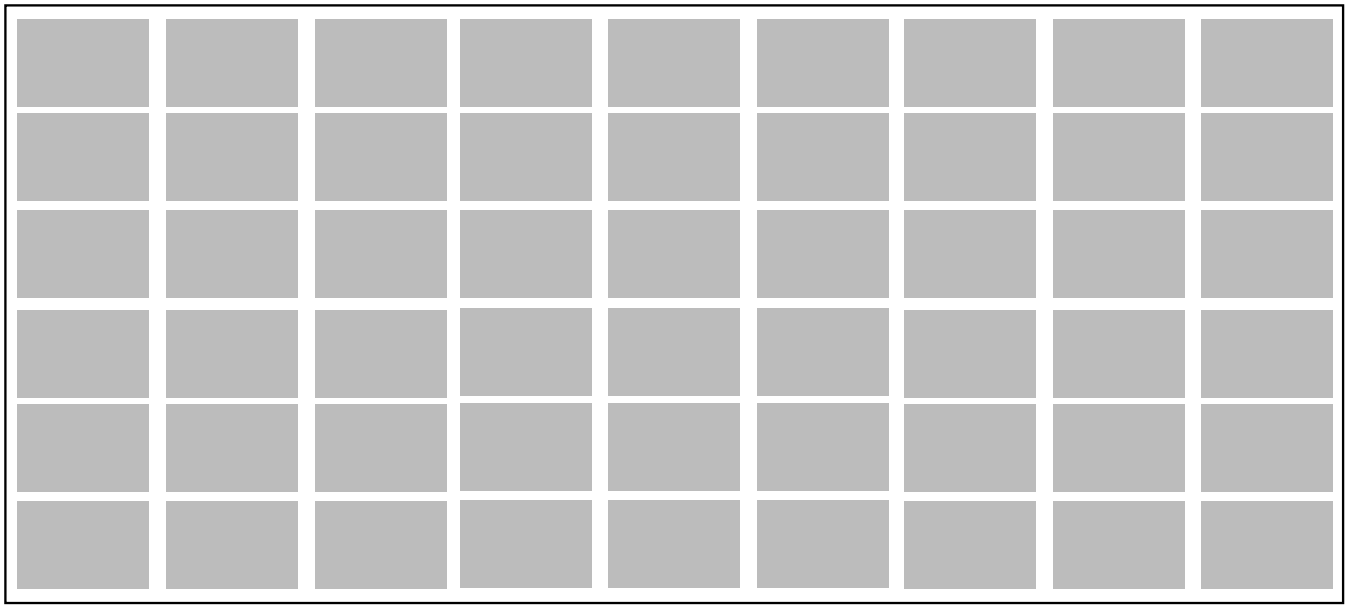
COJUNTOS (SETS)

- Complejidad Operaciones:
- Inserción: **O(1)**
- Eliminación: **O(1)**
- Búsqueda: **O(1)**
- Unión, Intersección, Diferencia: **O(n)**

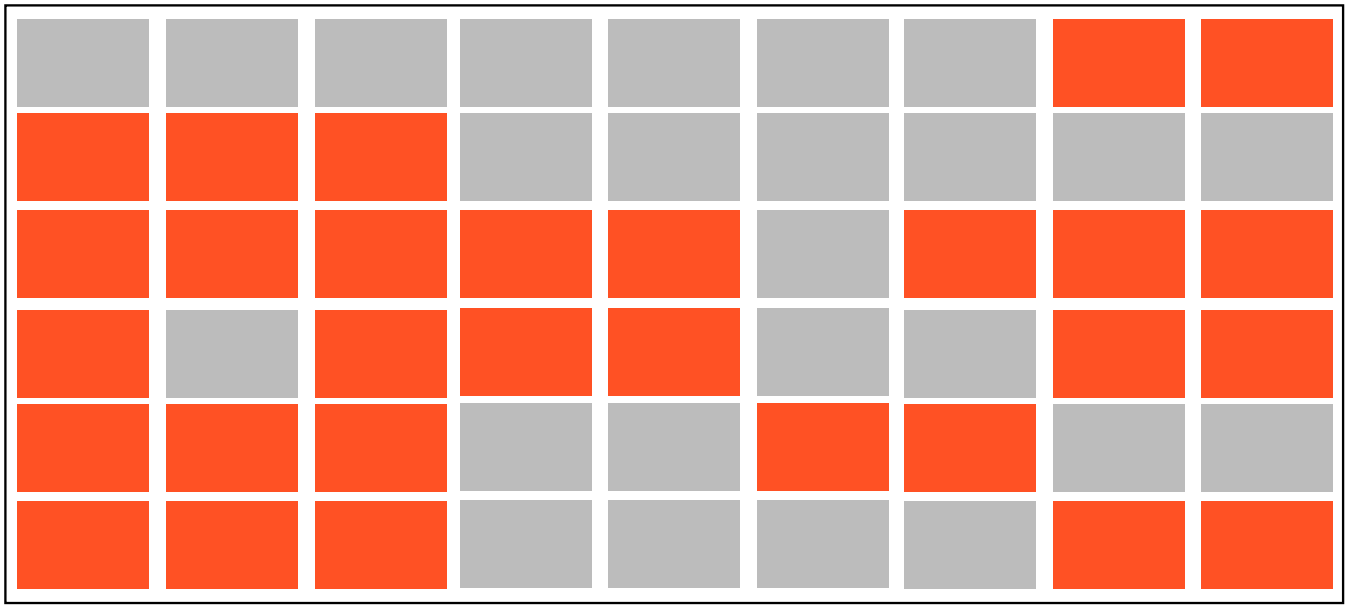
```
2   set1 = set()
3   set2 = {30, 40, 50}
4
5   set1.add(10)    # Insertar
6   set1.remove(20)  # Eliminar
7
8   existe = 10 in set1
9
10  union = set1 | set2
11  interseccion = set1 & set2
12  diferencia = set1 - set2
```

Funcionamiento Interno

Funcionamiento Interno

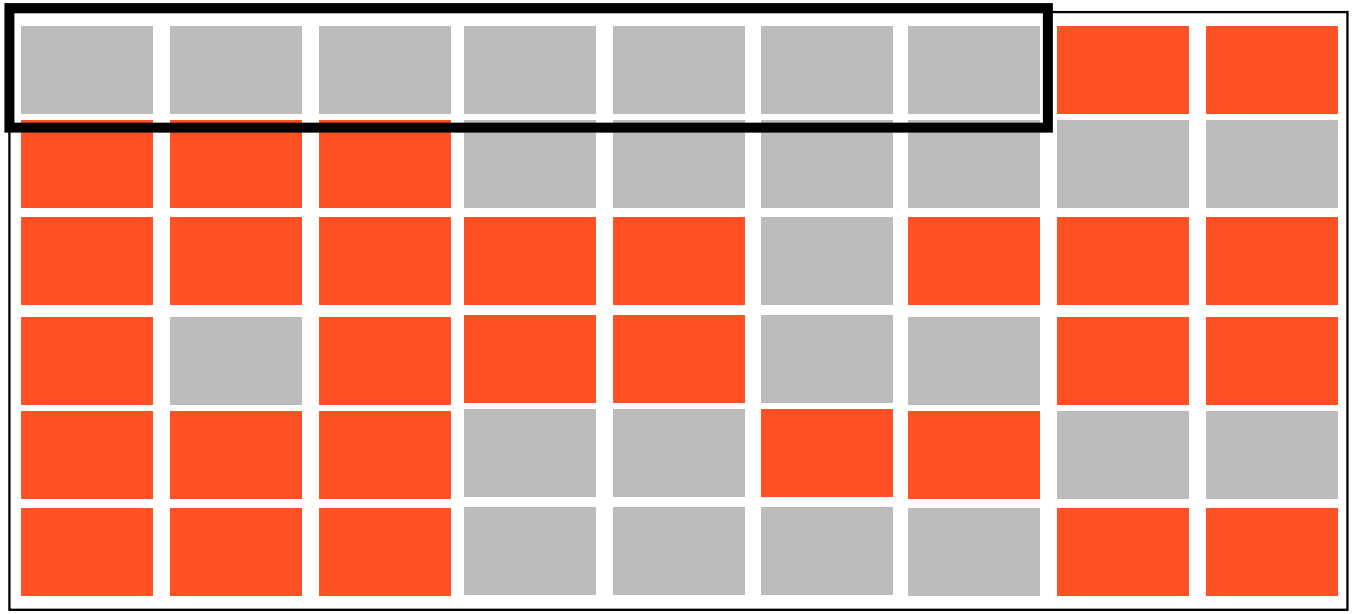


Funcionamiento Interno



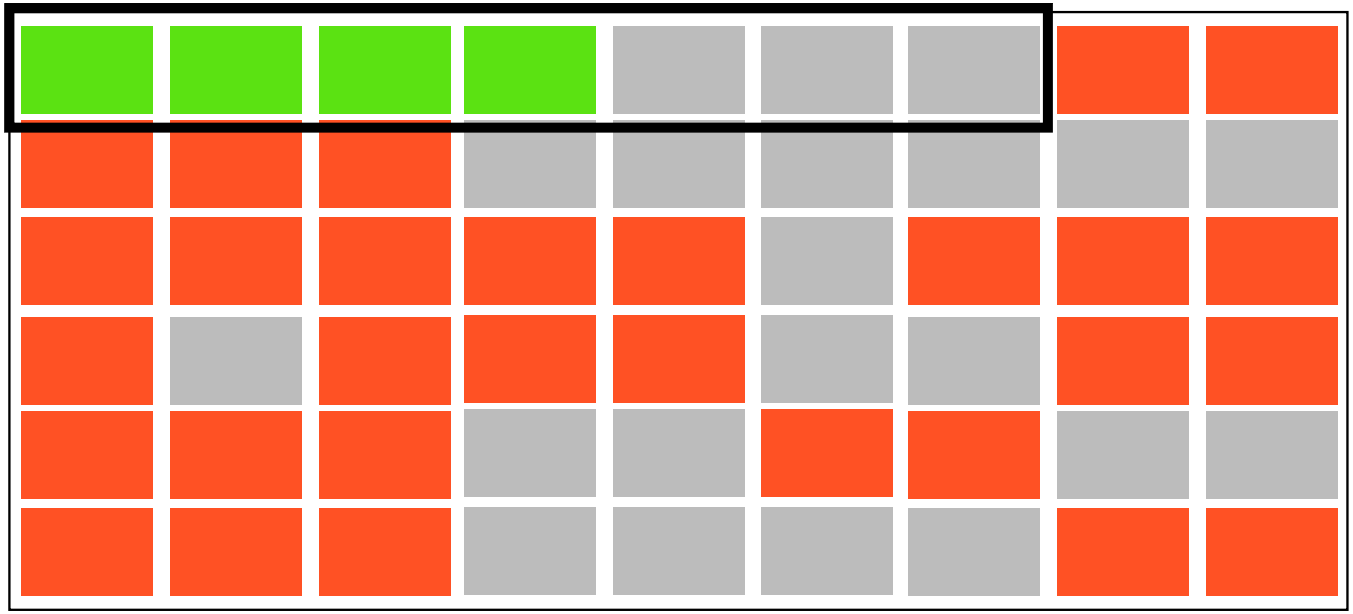
Funcionamiento Interno

Listas



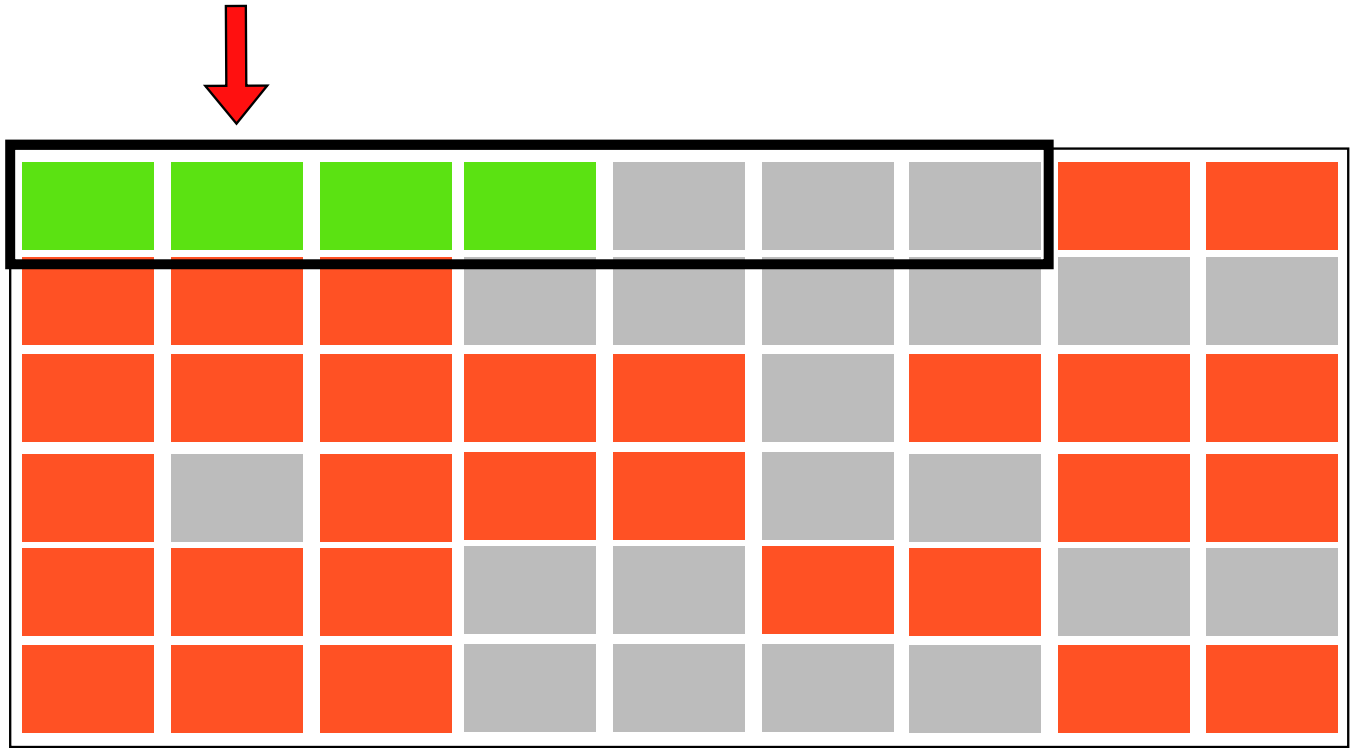
Funcionamiento Interno

Listas



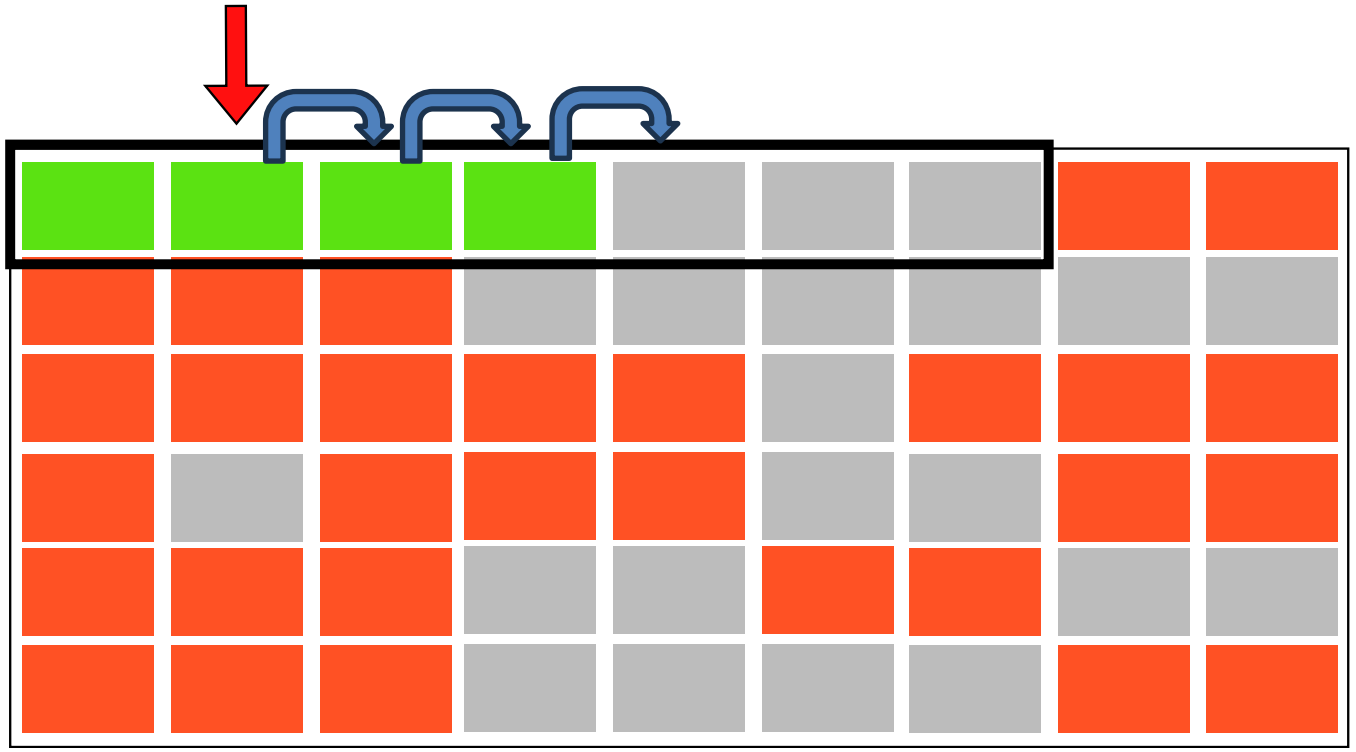
Funcionamiento Interno

Listas: Inserción



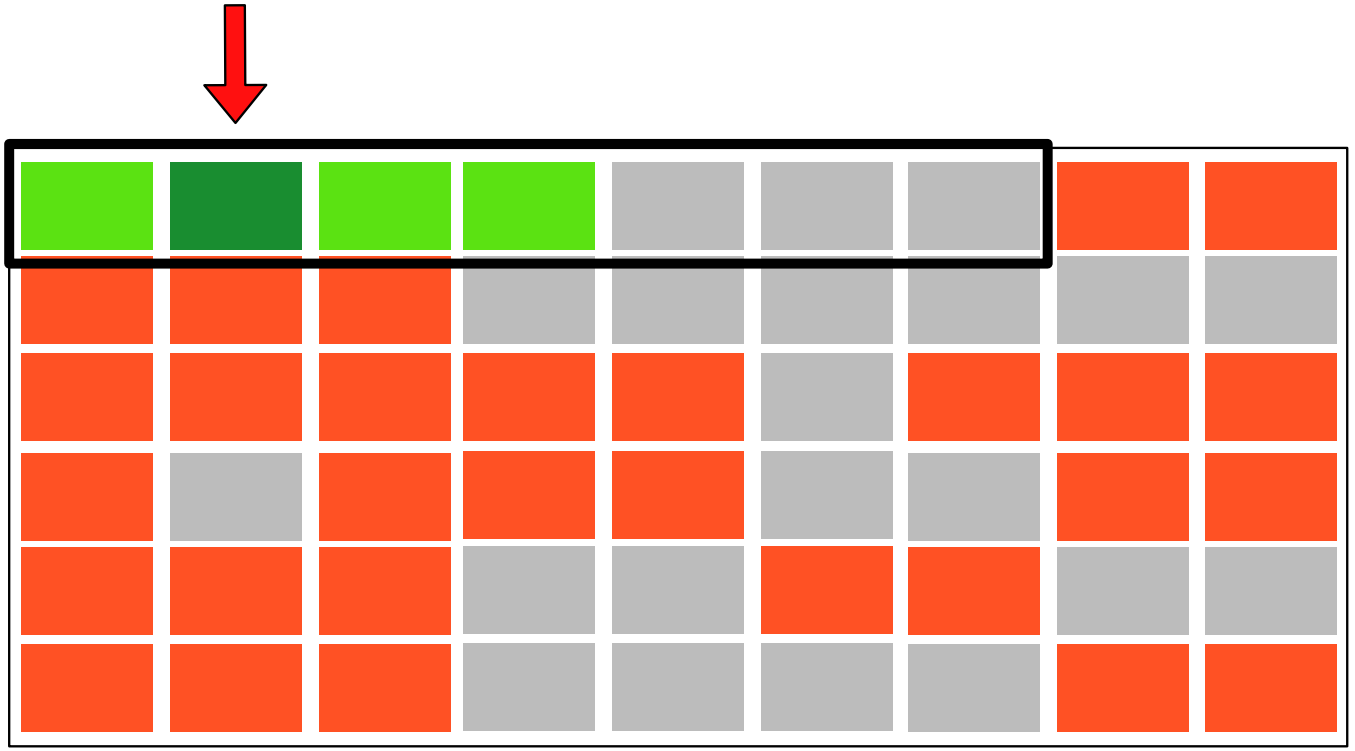
Funcionamiento Interno

Listas: Inserción **O(n)**



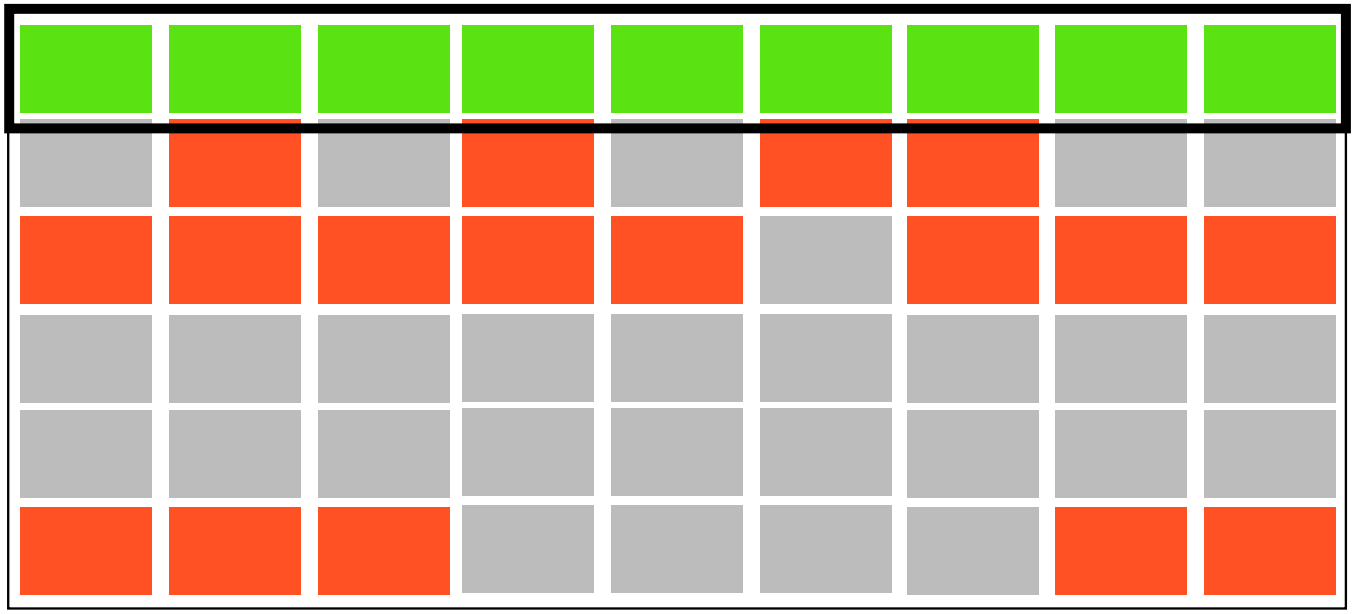
Funcionamiento Interno

Listas: Modificación **O(1)**



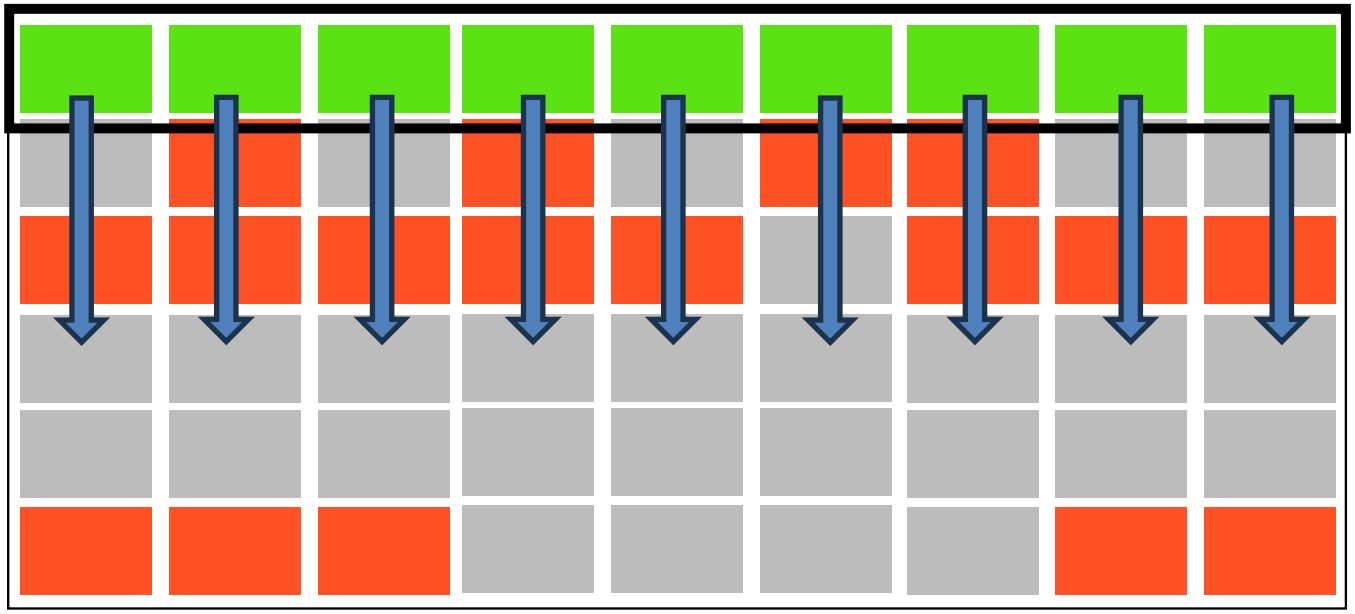
Funcionamiento Interno

Listas: Redimensionado **O(n)**



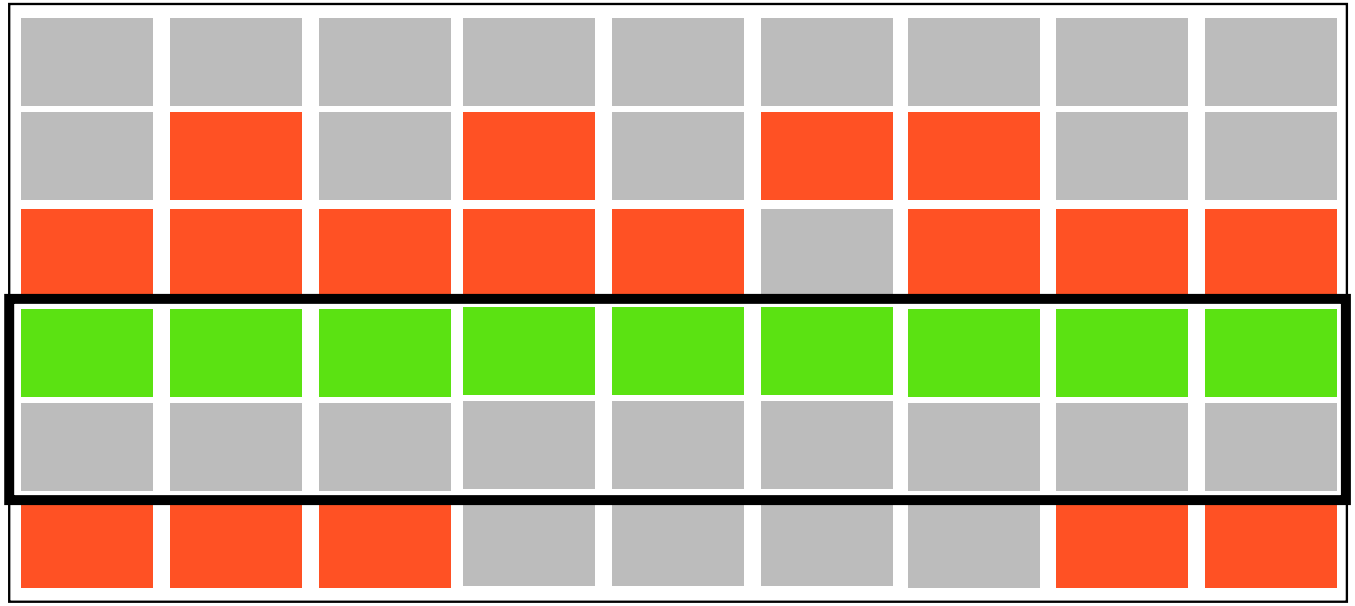
Funcionamiento Interno

Listas: Redimensionado **O(n)**



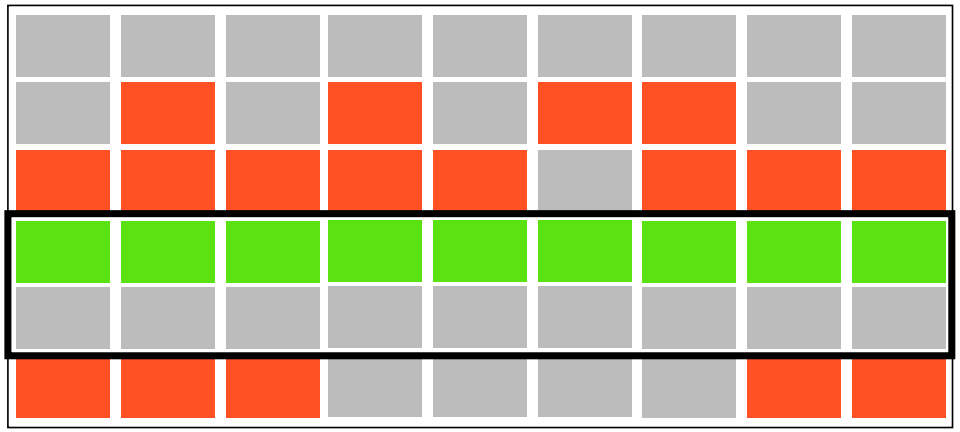
Funcionamiento Interno

Listas: Redimensionado **O(n)**



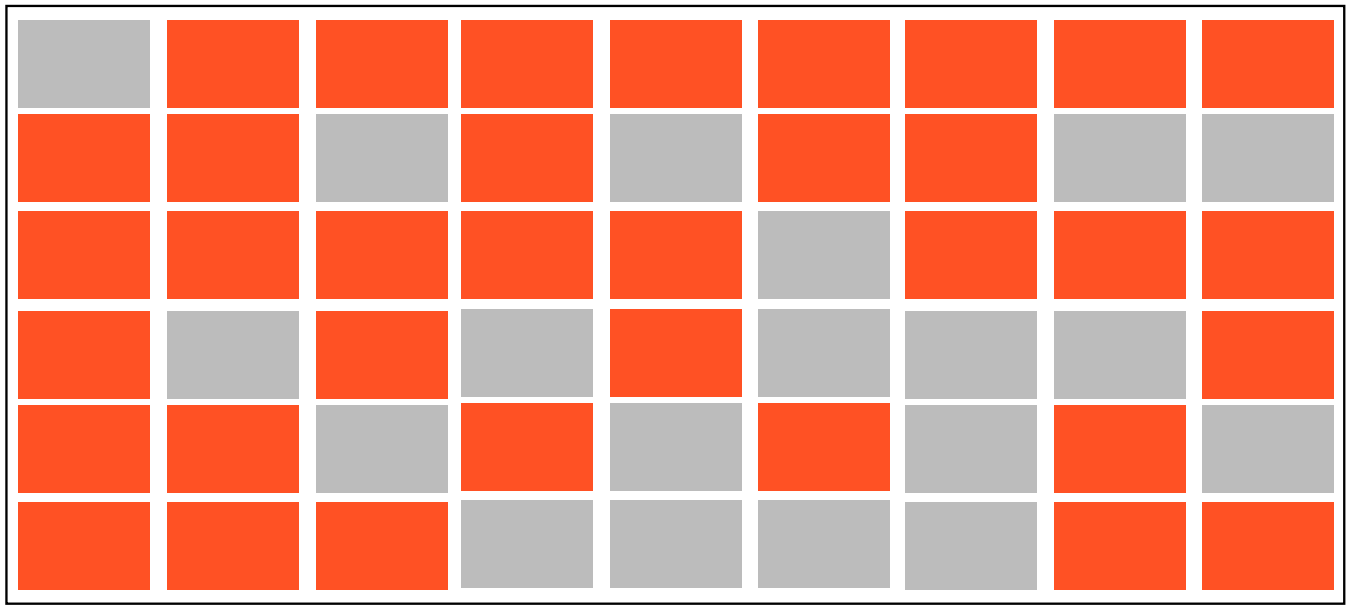
LISTAS

- Búsqueda
  - Mediante índice: O(1)
  - Mediante Valor: **O(n)**
- Eliminación **O(n)**
- Añadir al final O(1)
- Insertar en medio **O(n)**
- Modificar O(1)



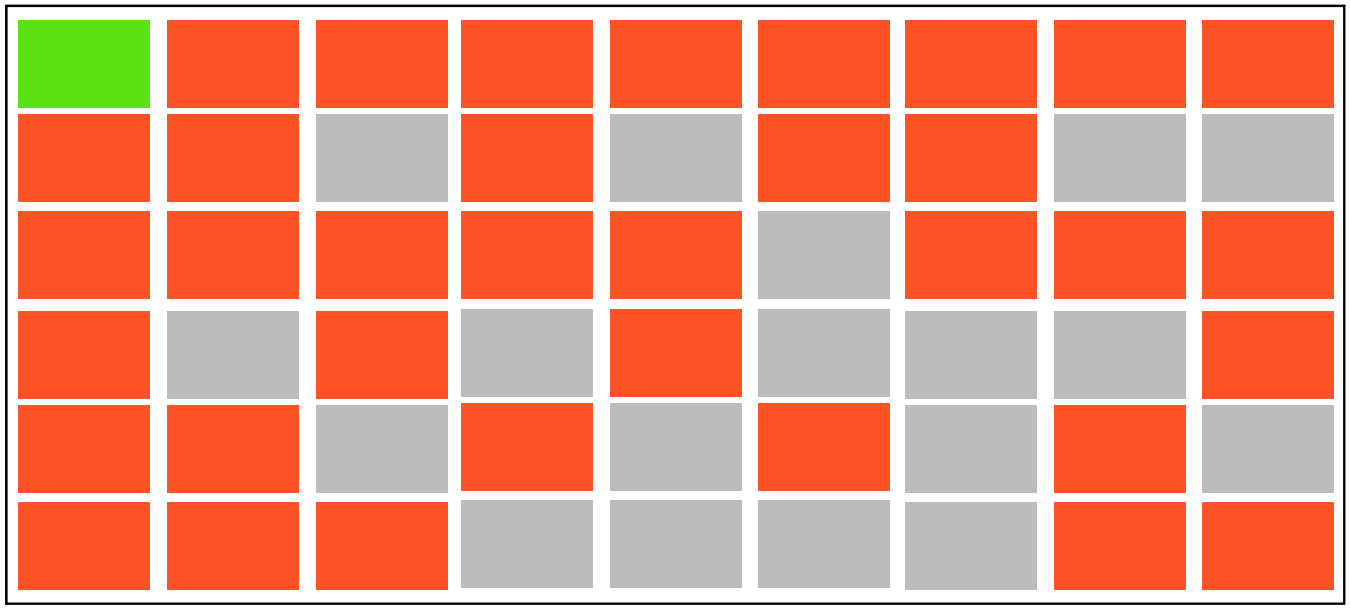
Funcionamiento Interno

Pilas y Colas (Doble Lista Enlazada)



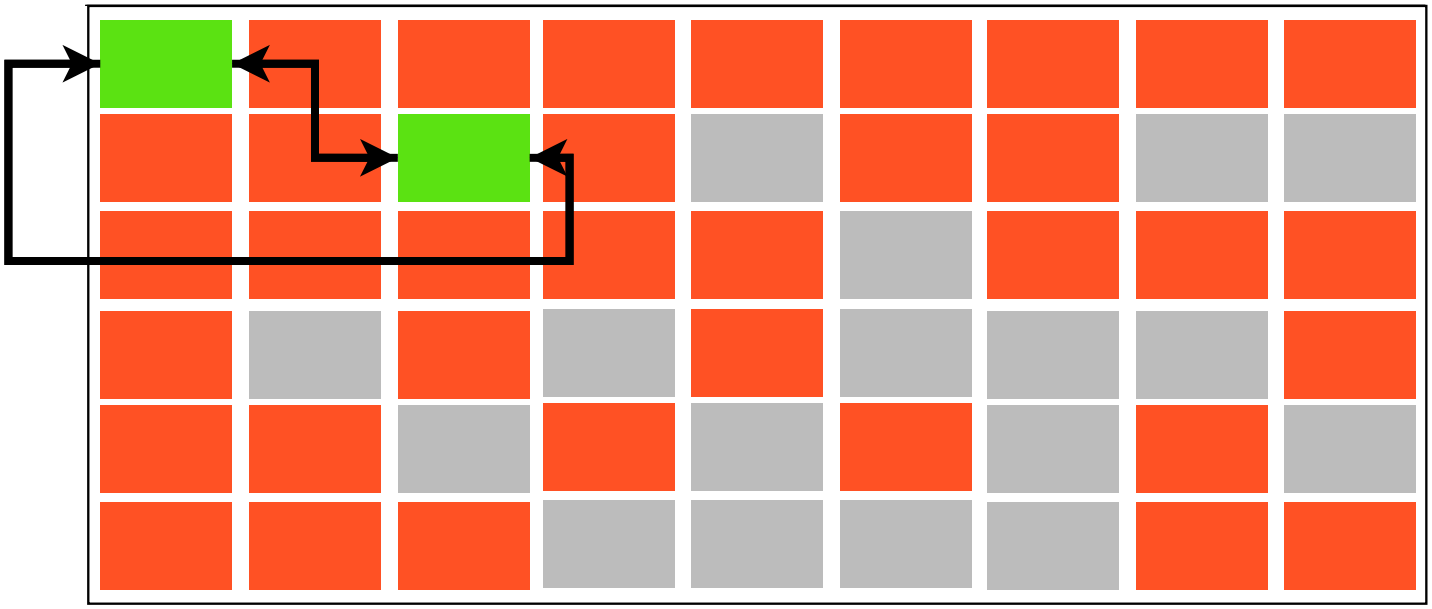
Funcionamiento Interno

Pilas y Colas (Doble Lista Enlazada)



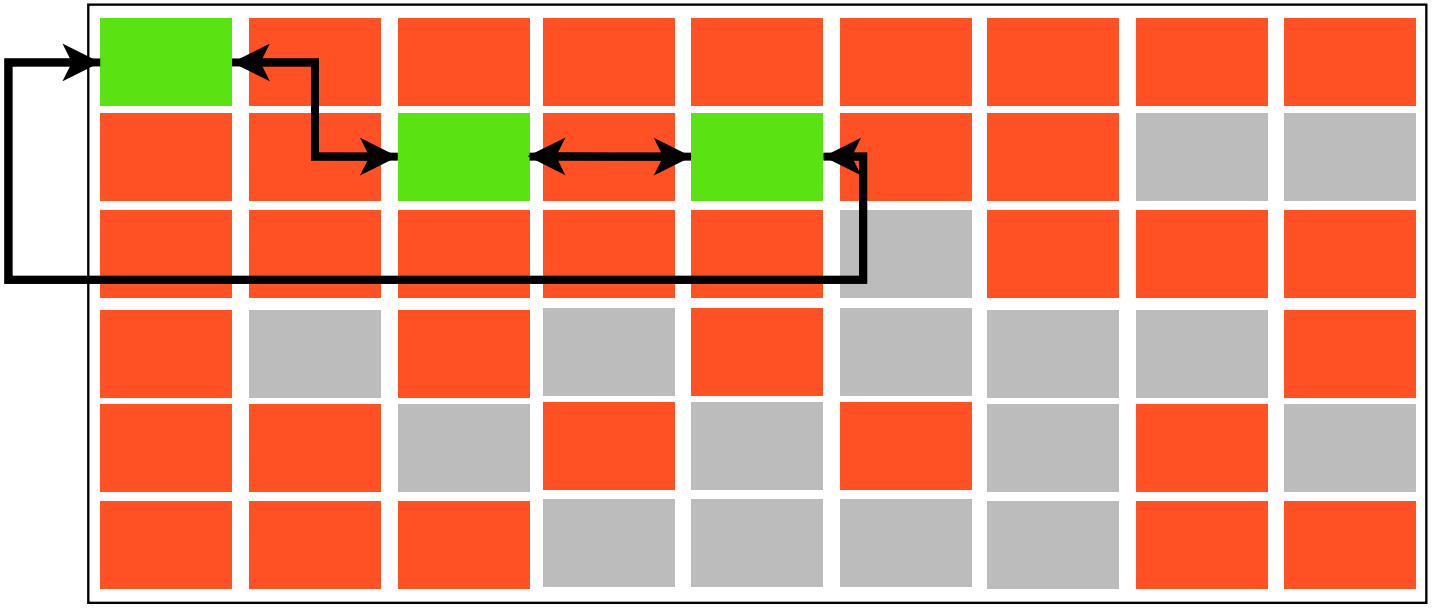
Funcionamiento Interno

Pilas y Colas (Doble Lista Enlazada)



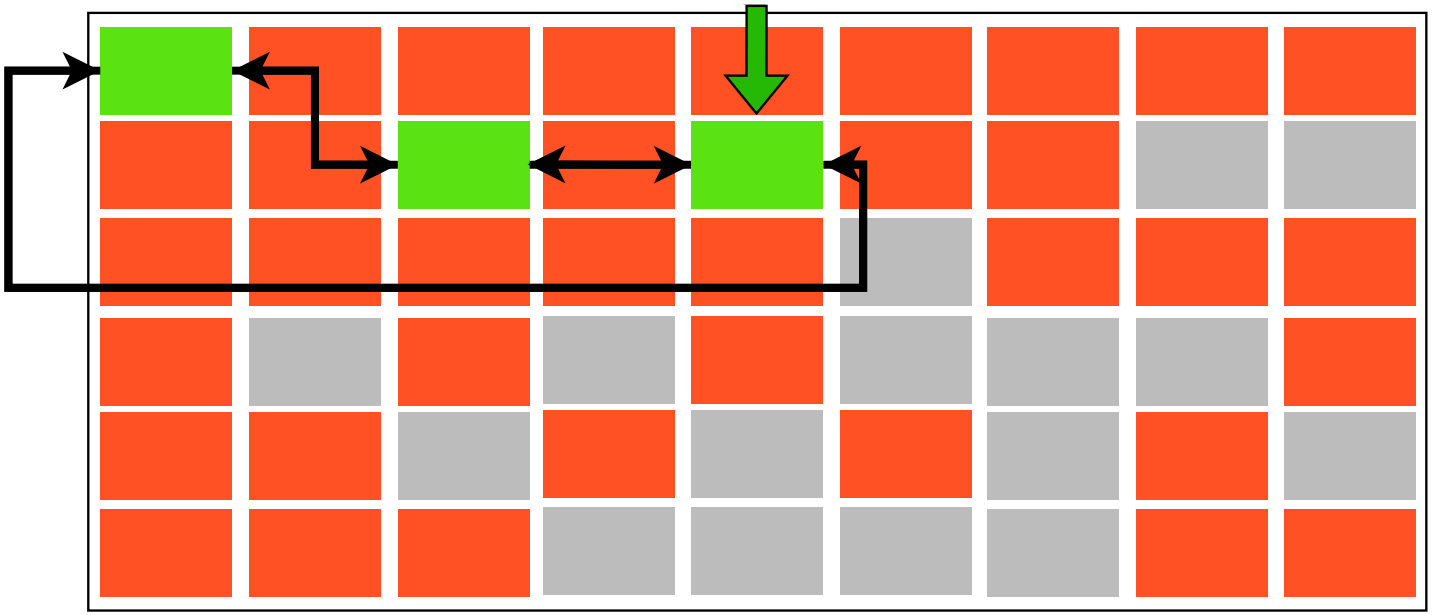
Funcionamiento Interno

Pilas y Colas (Lista Enlazada Doble)
Inserción **O(1)**



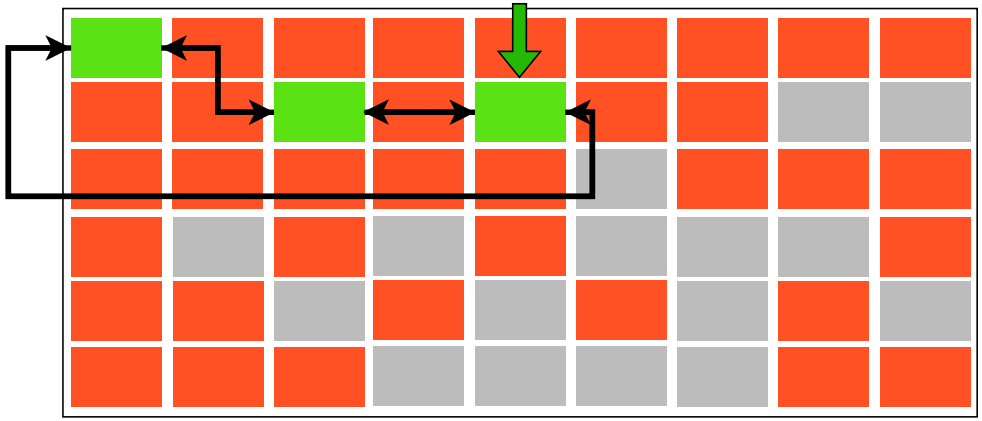
Funcionamiento Interno

Pila (Lista Enlazada Doble)



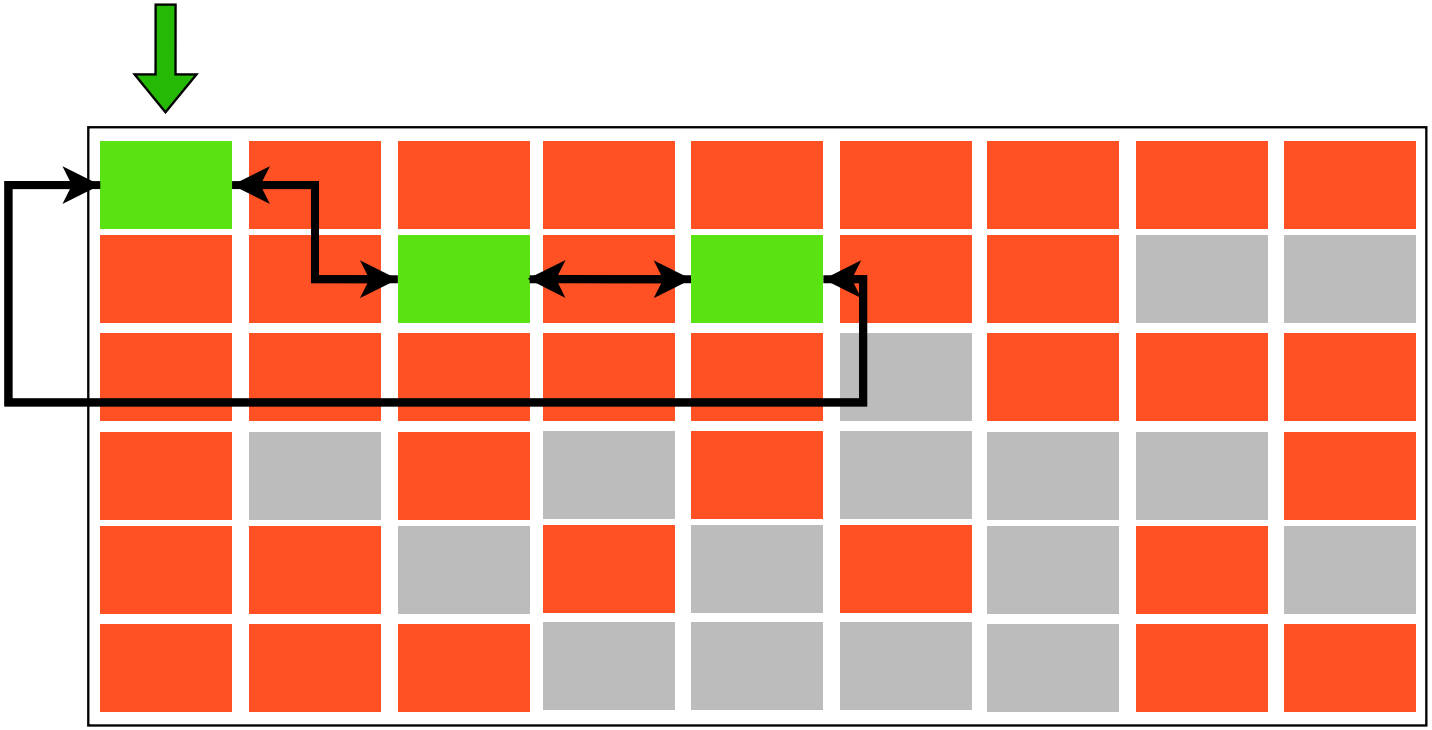
PILAS (STACKS)

- Complejidad Operaciones:
- Apilar: **O(1)**
- Desapilar (Pop): **O(1)**
- Ver cima (Top): **O(1)**
- Comprobar vacía: **O(1)**



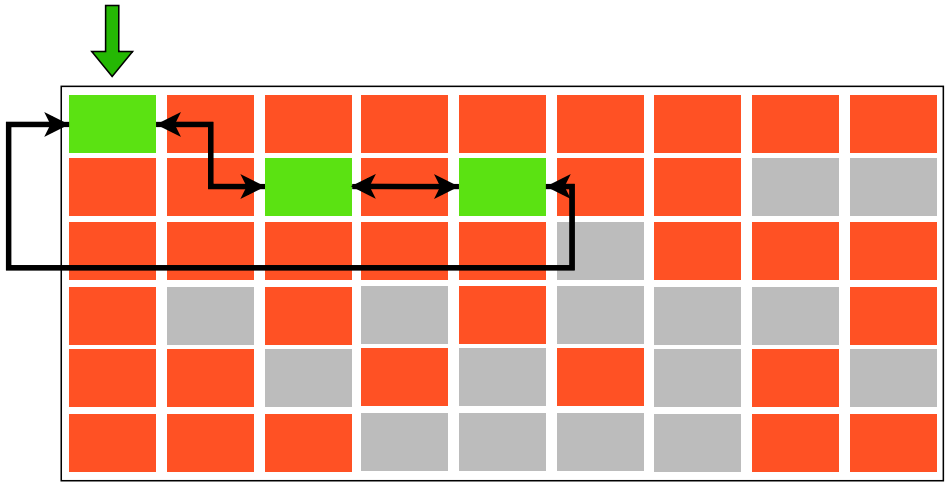
Funcionamiento Interno

Cola (Lista Enlazada Doble)



COLAS (QUEUES)

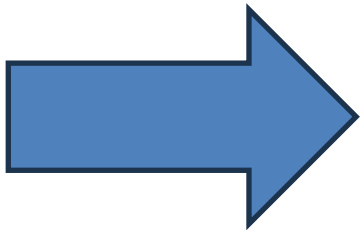
- Complejidad Operaciones:
- Encolar: **O(1)**
- Desencolar: **O(1)**
- Ver frente (Front): **O(1)**
- Comprobar vacía: **O(1)**



Funciones Hash

Conjunto de Strings de longitud hasta 10

- Hola
- Mañana
- Qué tal?
- Alberto
- ….



Indices de una lista de N posiciones

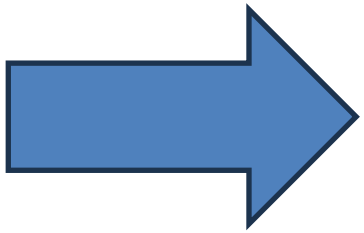
0,1,…, N-1

Funcionamiento Interno

Funciones Hash

Conjunto de Strings de longitud hasta 10

- Hola
- Mañana
- Qué tal?
- Alberto
- ….



$$f(s)$$

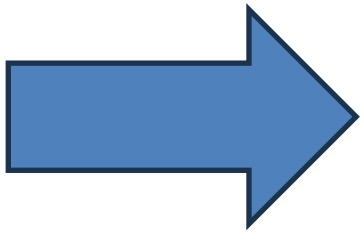
Indices de una lista de N posiciones

0,1,…, N-1

Funciones Hash

Conjunto de Strings de longitud hasta 10

- Hola
- Mañana
- Qué tal?
- Alberto
- ….



$$f(s)$$

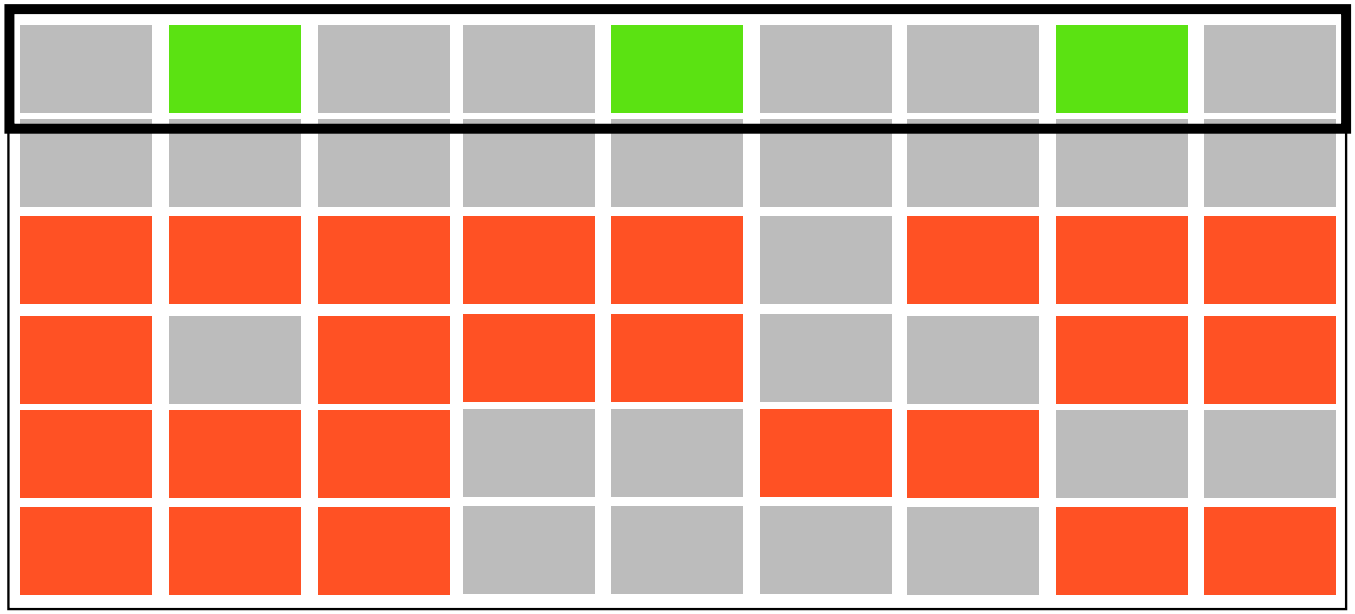
Indices de una lista de N posiciones

0,1,…, N-1

$$f(s) = bits(S) \mod N$$

Funcionamiento Interno

Conjuntos (Tabla Hash)

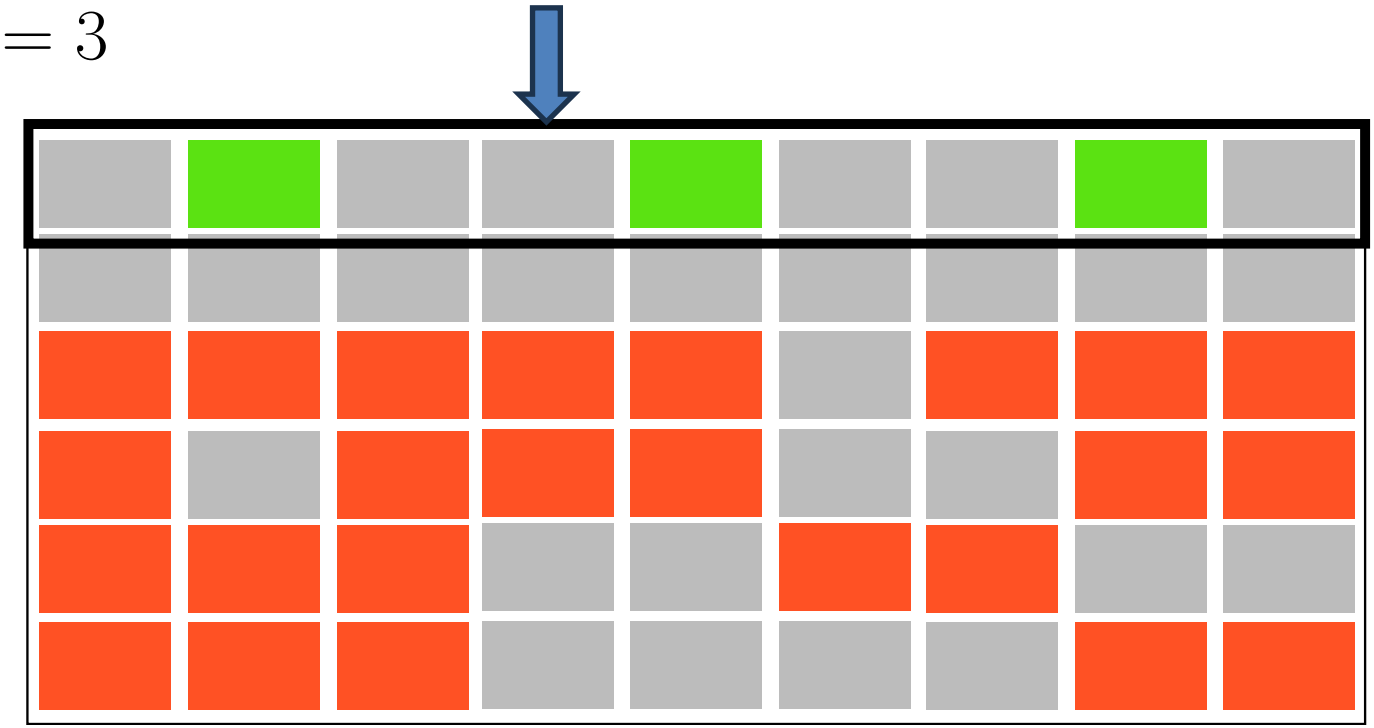


Funcionamiento Interno

Conjuntos (Tabla Hash)

s = hola $f(s) = bits(S) \mod N = 103413 \mod 9 = 3$

$f(s) = 3$

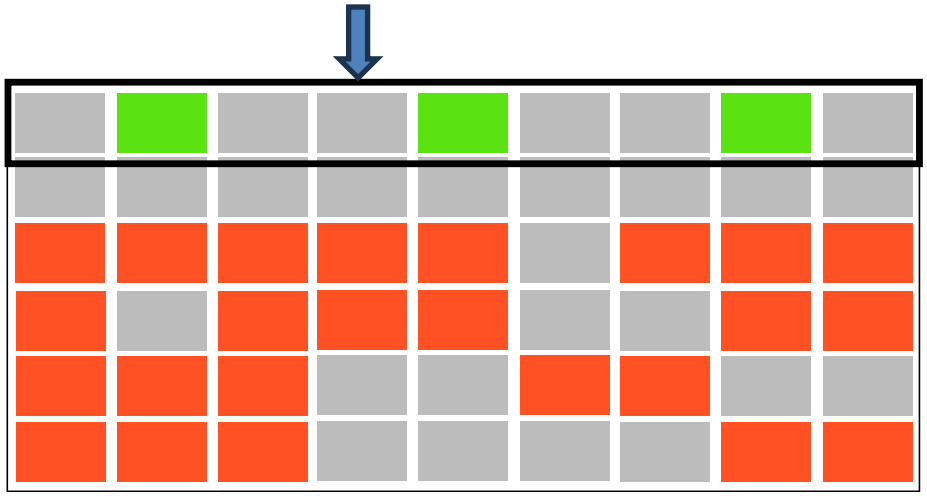


COJUNTOS (SETS)

- Complejidad Operaciones:
- Inserción: **O(1)**
- Eliminación: **O(1)**
- Búsqueda: **O(1)**
- Unión, Intersección, Diferencia: **O(n)**

s = hola

$f(s) = 3$



Estructuras de Datos Principales

- Listas
- Arrays
- Strings
- Pilas
- Colas
- Set

- **Mapas**
- **Colas de Prioridad**
- **Árboles**

Mapas

MAPAS (DICTIONARY)

- Estructura de datos que almacena pares **clave-valor**.
- Basado en una tabla **hash**.

```
2   mapa1 = {}
3   mapa2 = {"a": 10, "b": 20}
```

- No está ordenado (a partir de Python 3.7 mantiene el orden de inserción).
- Aplicaciones:
  - **Búsqueda** rápida de valores **clave-valor**.
  - **Agrupación** de datos por clave

MAPAS (DICTIONARY)

- Complejidad **Promedia** Operaciones:

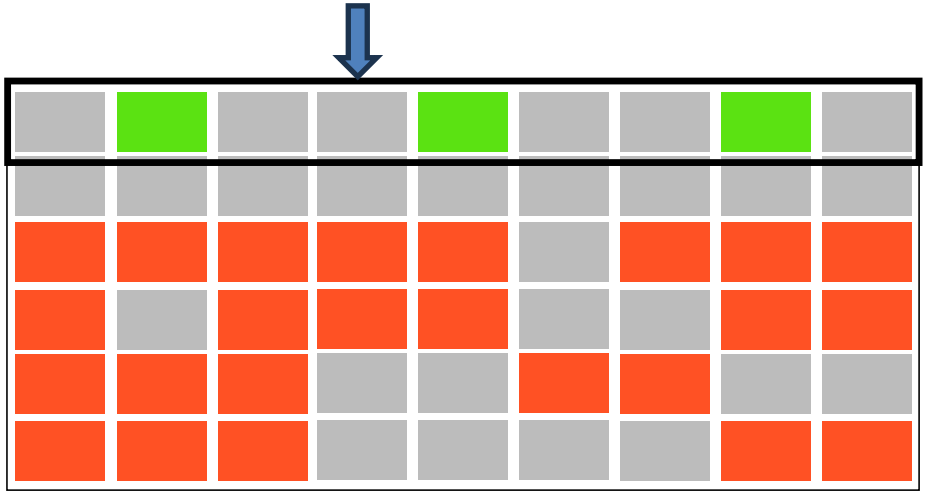
- Inserción: **O(1)**
- Eliminación: **O(1)**
- Búsqueda: **O(1)**

```
 6  print(mapa["a"])
 7
 8  mapa["w"] = 400  # Inserción
 9  del mapa["x"]  # Eliminación
10
11  existe = "y" in mapa  # Búsqueda
12
13  for clave, valor in mapa.items():
```

MAPAS (DICTIONARY)

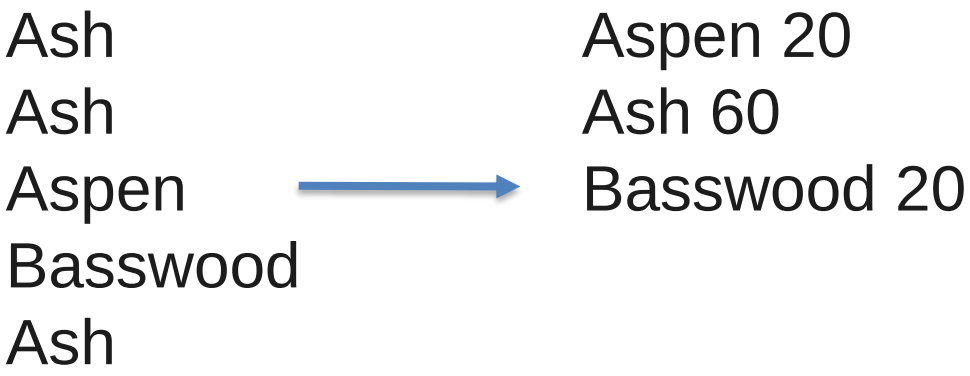
- Complejidad **Promedia** Operaciones:
- Inserción: **O(1)**
- Eliminación: **O(1)**
- Búsqueda: **O(1)**

<s,1> s = hola $f(s) = 3$



Hardwood Species

- https://open.kattis.com/problems/hardwoodspecies
- ID: hardwoodspecies
- Ejemplo:



Colas de prioridad

COLAS DE PRIORIDAD (PRIORITY QUEUES)

- Estructura donde los elementos se atienden según su **prioridad** y no solo su orden de llegada.

- **heapq :**
  - **Montículo**
  - Más eficiente que **list.**

```
1  import heapq
2
3  colap = []
4  heapq.heappush(colap, 1)
5  heapq.heappop(colap)
6
```

- Aplicaciones:
  - Algoritmo Dijkstra.
  - Gestión de tareas o recursos.
  - Simulaciones de eventos

COLAS DE PRIORIDAD (PRIORITY QUEUES)

- Complejidad Operaciones:
- Insertar: **O(log(n))**
- Eliminación: **O(log(n))**
- Ver primero: **O(1)**
- Comprobar vacía: **O(1)**

```python
import heapq

colap = []

heapq.heappush(colap, 1) #Inserción
heapq.heappop(colap)     #Eliminación

primero = colap[0]
esta_vacia = len(cola) == 0

```

COLAS DE PRIORIDAD (PRIORITY QUEUES)

- Por defecto ordenación ascendente.
- Para ordenar descendentemente se introducen los elementos en negativo y se extraen también en negativo.

```
1  import heapq
2
3  colap = []
4
5  heapq.heappush(colap, -1)
6  heapq.heappush(colap, -3)
7  heapq.heappush(colap, -2)
8
9  primero = -heapq.heappop(colap) #3
10 segundo = -heapq.heappop(colap) #2
11 tercero = -heapq.heappop(colap) #1
```

Annoyed coworkers

- https://open.kattis.com/problems/annoyedcoworkers
- ID: annoyedcoworkers
- Ejemplo:

```
4 4
1 2
2 3  ⟶  7
3 4
4 5
```

Assigning Workstations

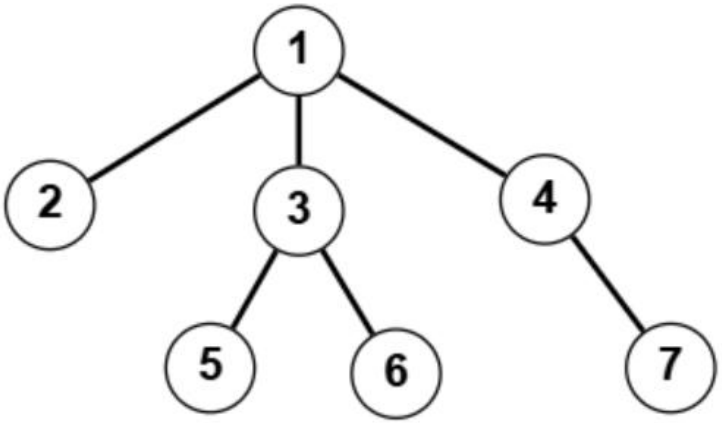
- https://open.kattis.com/problems/workstations
- ID: workstations
- Ejemplo:

```
3 5
1 5
6 3
14 6
```

```
2
```

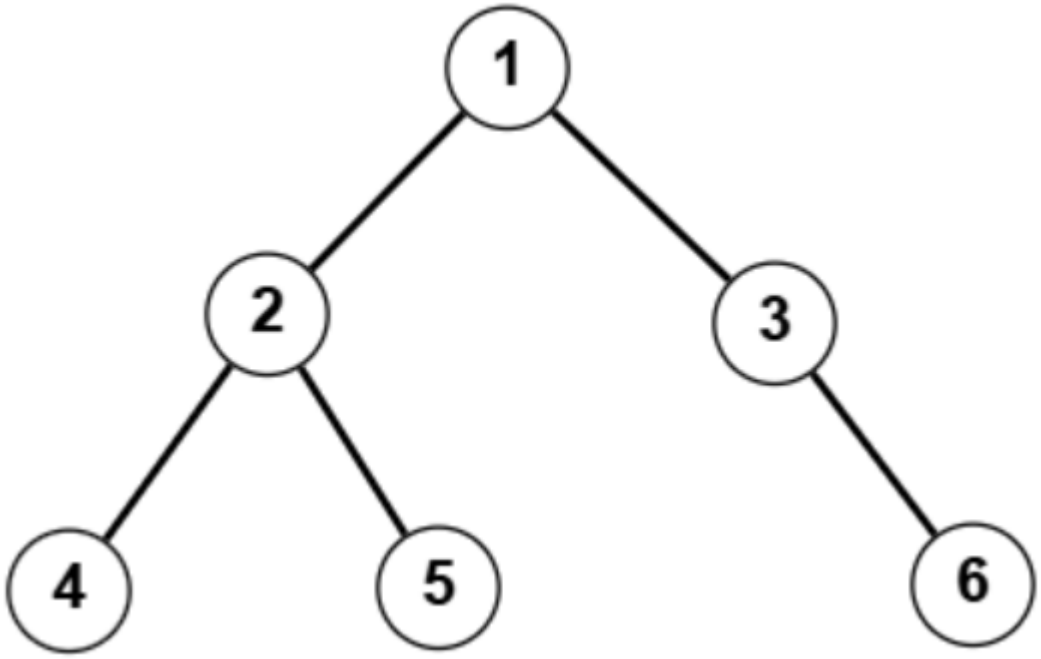
Árboles

ÁRBOLES (TREE)



- Representan relaciones de jerarquía, por niveles
- Tipos de nodos:
  - Raíz
  - Intermedios
  - Hoja
- Atributos:
  - Altura
  - Profundidad
  - Grado

ÁRBOLES BINARIOS (BINARY TREE)

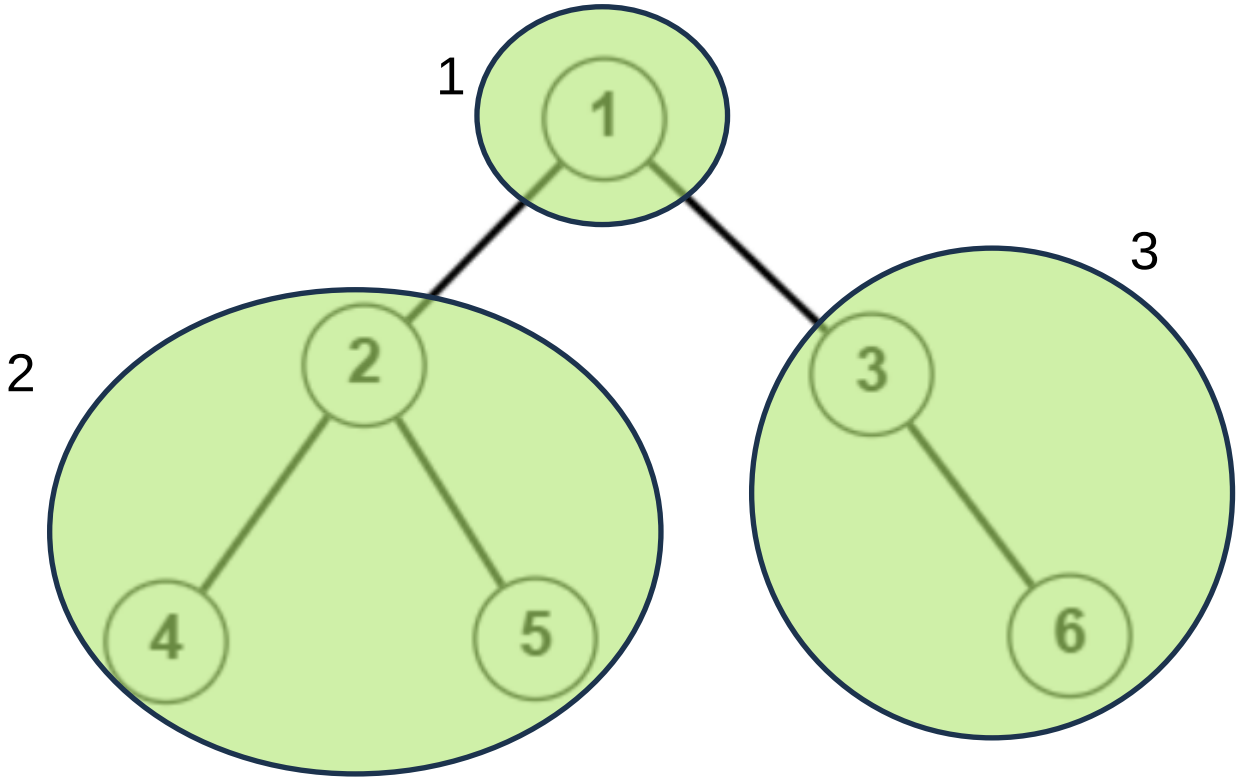


- Máximo dos hijos por nodo
- Recorridos:
  - Preorden
  - Postorden
  - Inorden

ÁRBOLES BINARIOS (BINARY TREE)

Preorden:

Nodo – Izquierda - Derecha



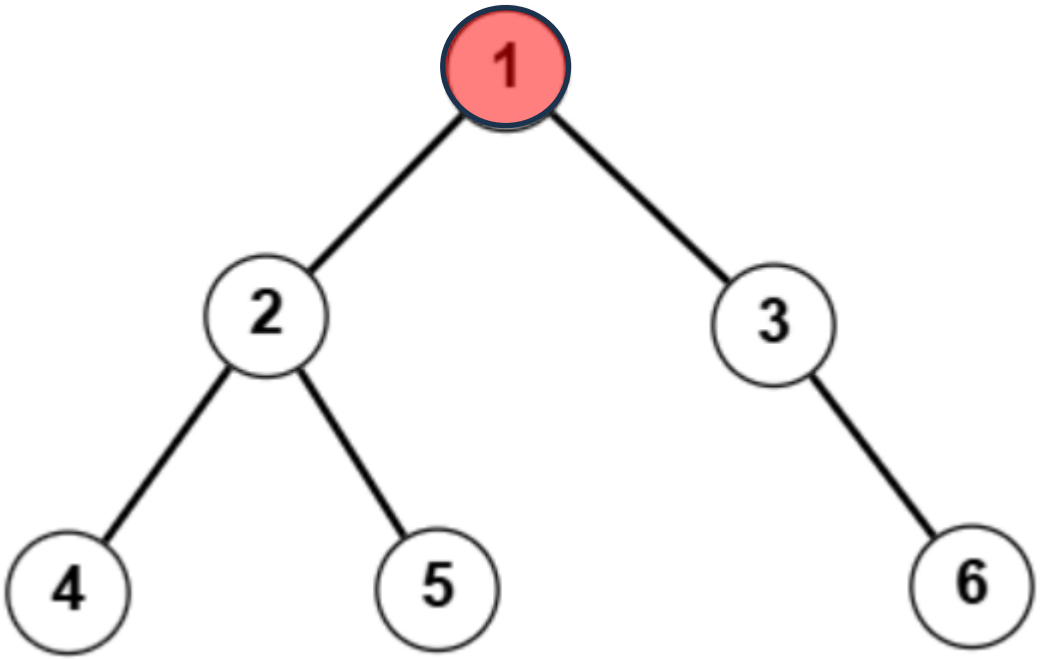
ÁRBOLES BINARIOS (BINARY TREE)

Preorden:

Nodo – Izquierda - Derecha

Recorrido:

1



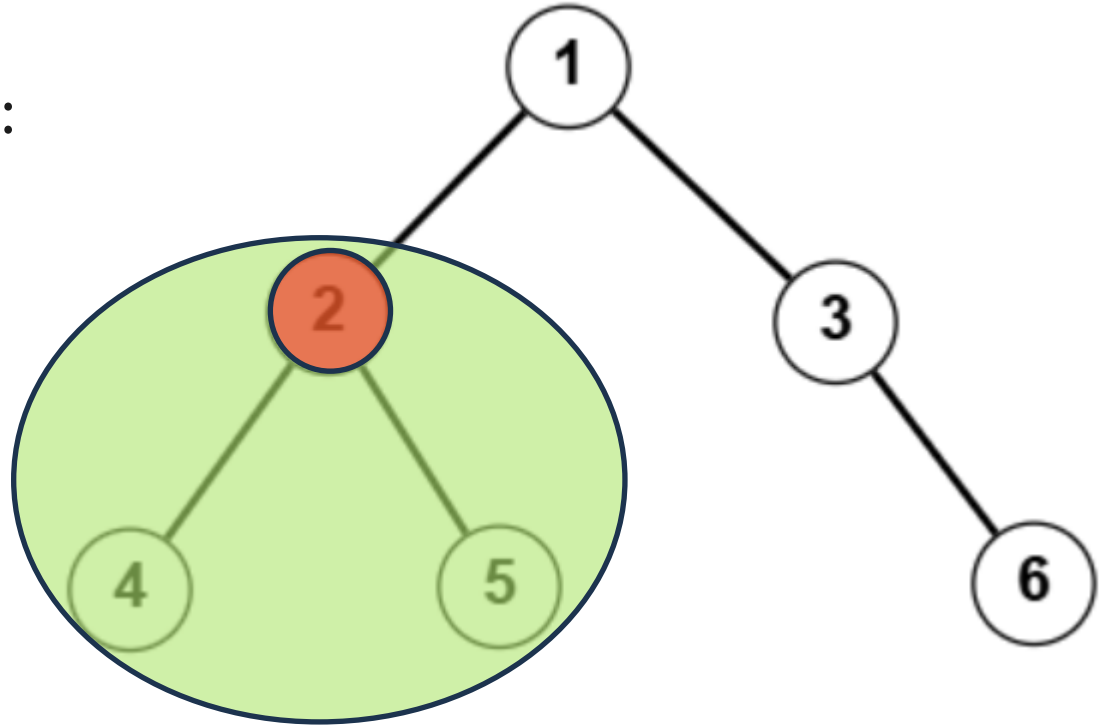
ÁRBOLES BINARIOS (BINARY TREE)

Preorden:

Nodo – Izquierda - Derecha

Recorrido:

1 - 2



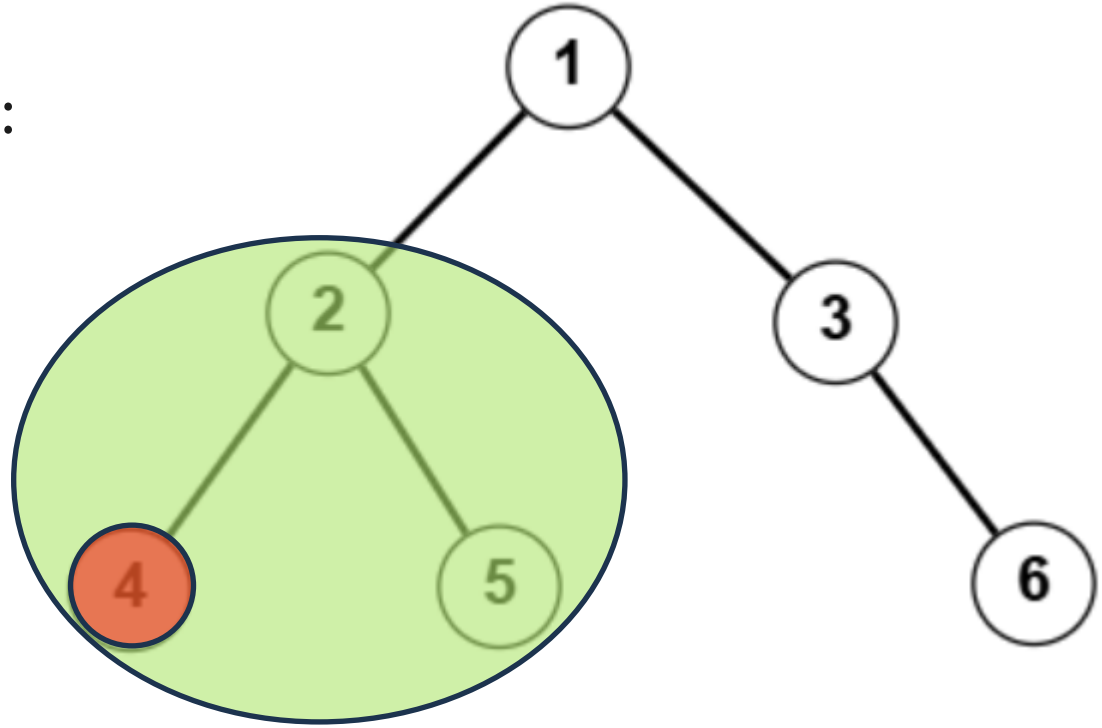
ÁRBOLES BINARIOS (BINARY TREE)

Preorden:

Nodo – Izquierda - Derecha

Recorrido:

1 – 2 – 4



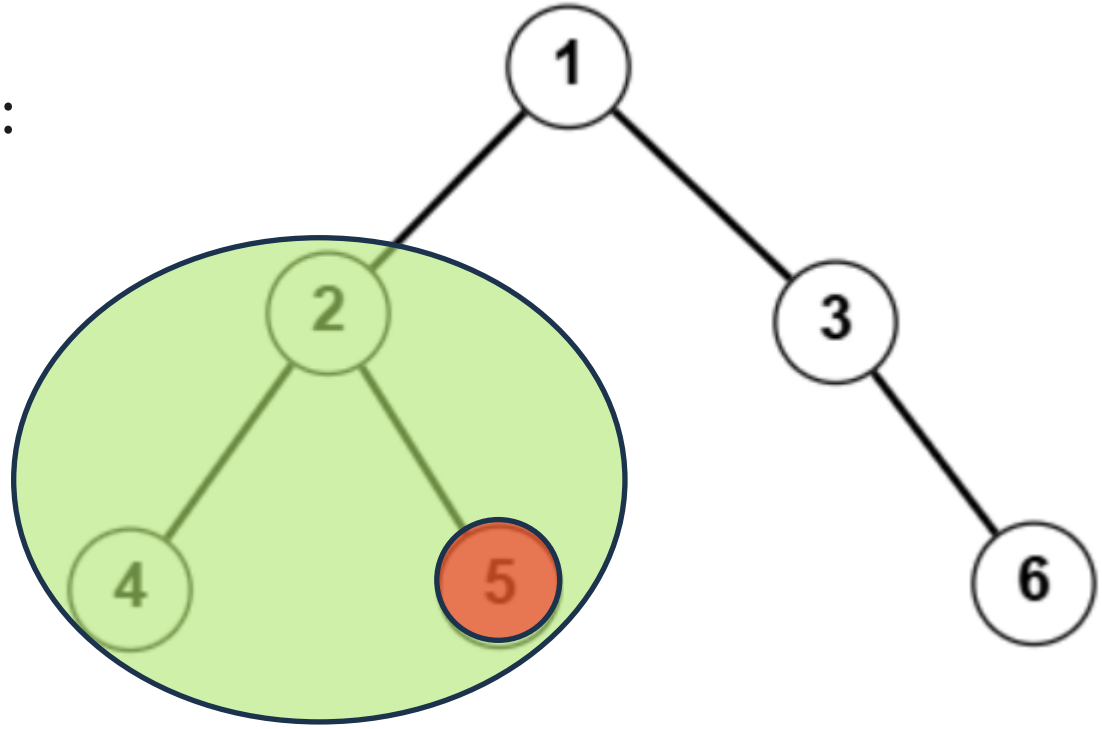
ÁRBOLES BINARIOS (BINARY TREE)

Preorden:

Nodo – Izquierda - Derecha

Recorrido:

1 – 2 – 4 - 5



ÁRBOLES BINARIOS (BINARY TREE)

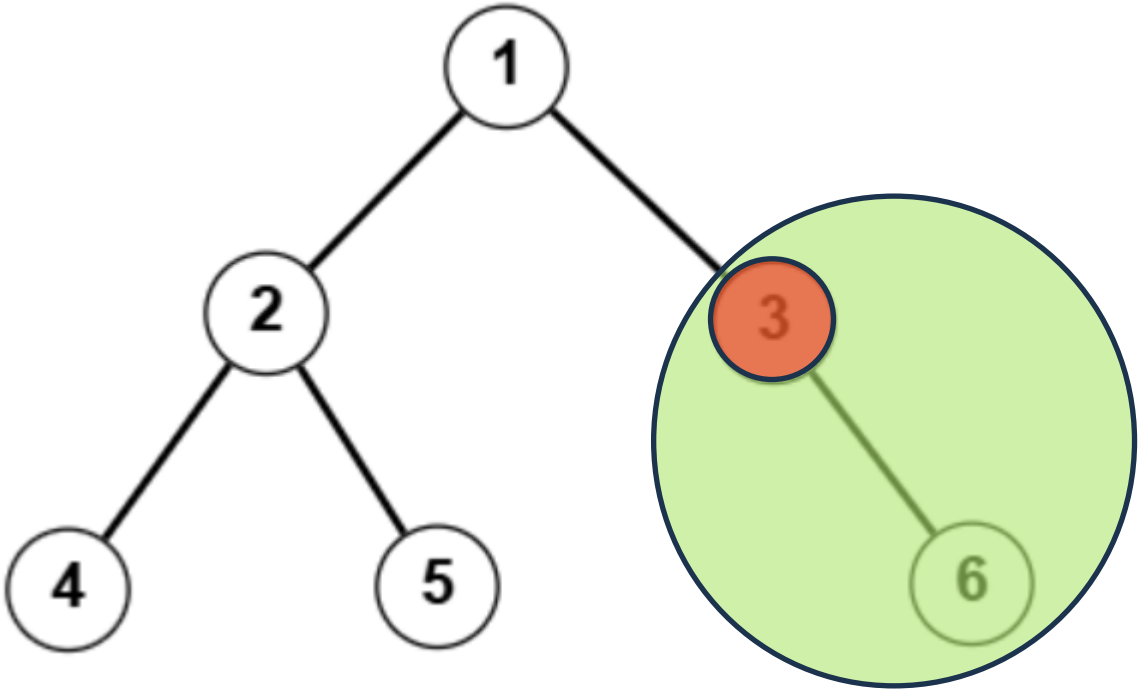
Preorden:

Nodo – Izquierda - Derecha

Recorrido:

1 – 2 – 4 -

5 – 3



ÁRBOLES BINARIOS (BINARY TREE)

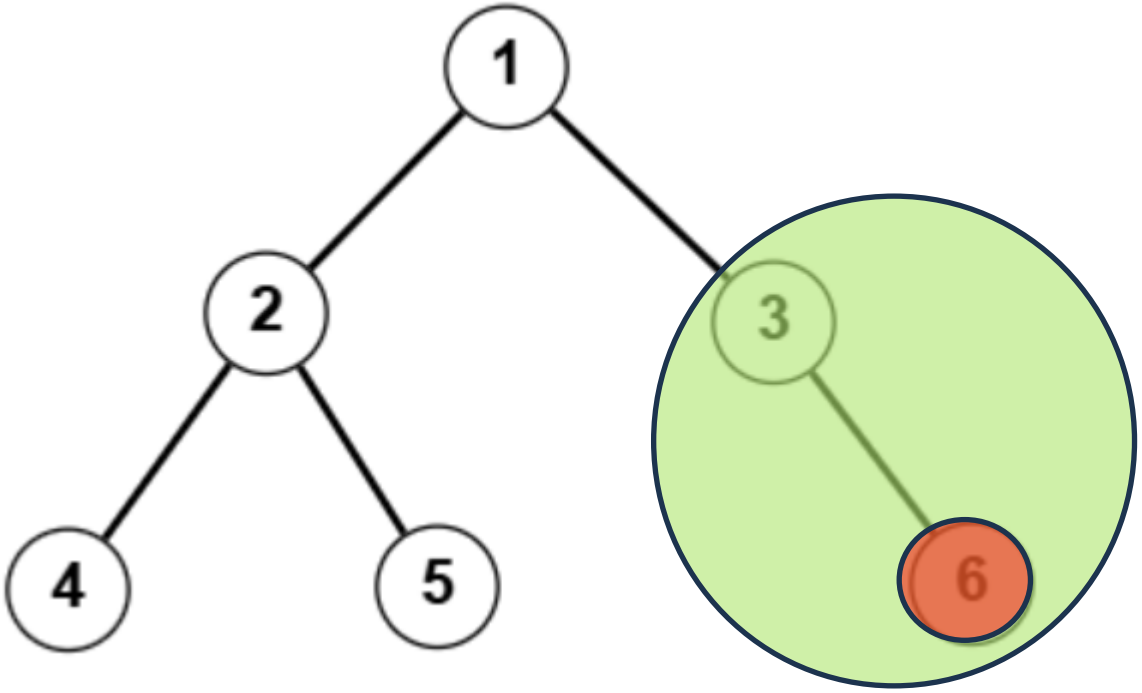
Preorden:

Nodo – Izquierda - Derecha

Recorrido:

1 – 2 – 4 -

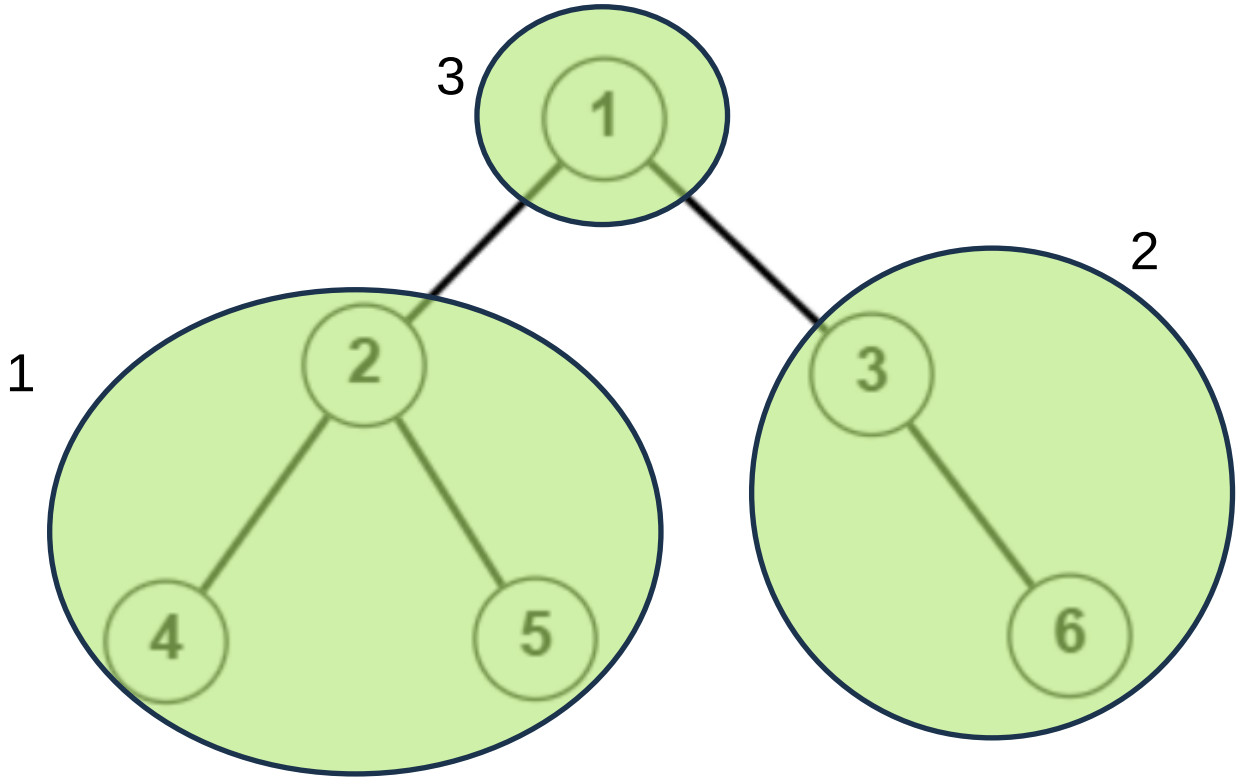
5 – 3 – 6



ÁRBOLES BINARIOS (BINARY TREE)

Postorden:

Izquierda - Derecha - Nodo



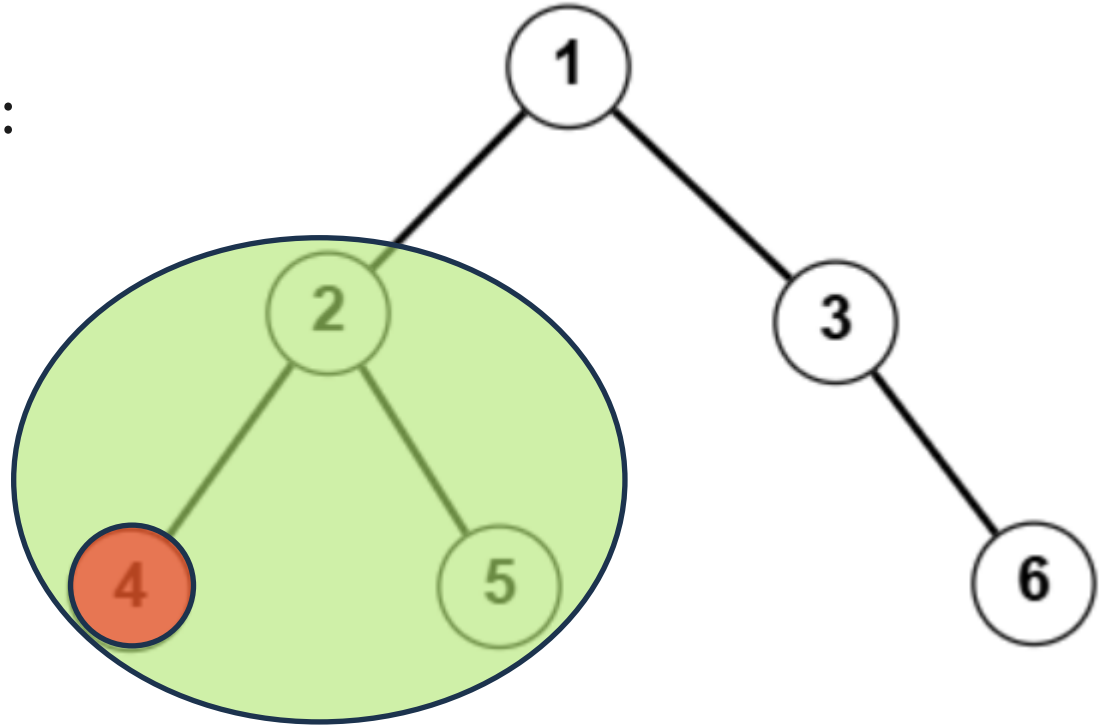
ÁRBOLES BINARIOS (BINARY TREE)

Postorden:

Izquierda - Derecha - Nodo

Recorrido:

4



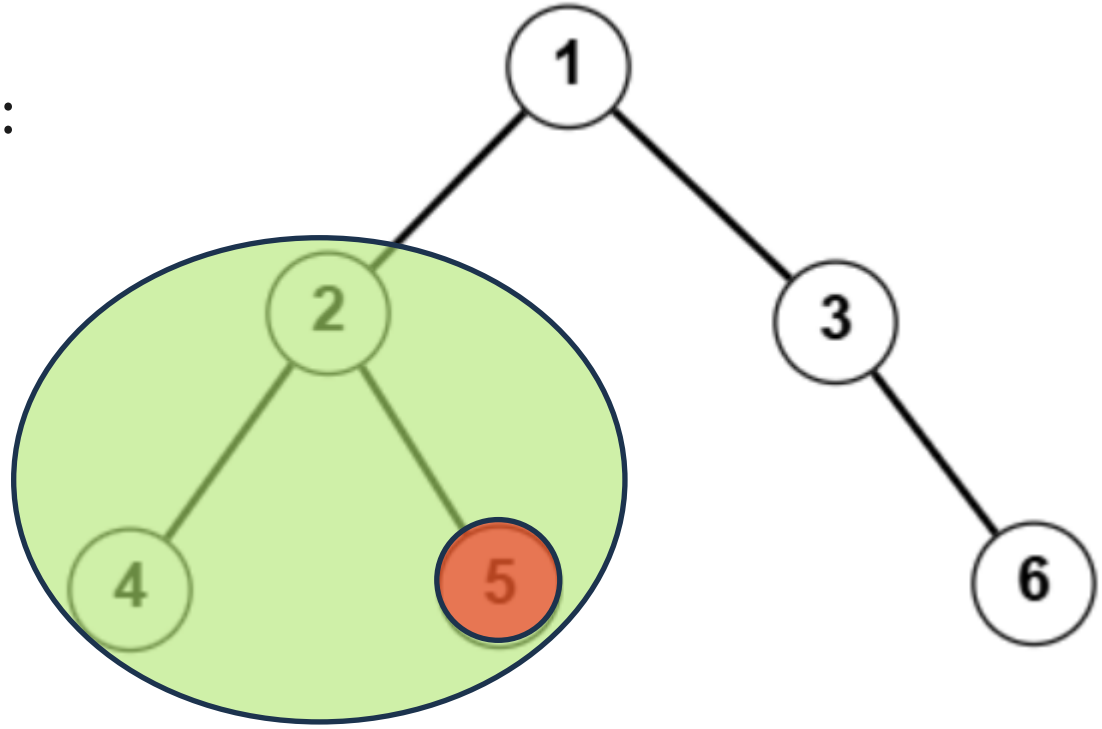
ÁRBOLES BINARIOS (BINARY TREE)

Postorden:

Izquierda - Derecha - Nodo

Recorrido:

4 – 5



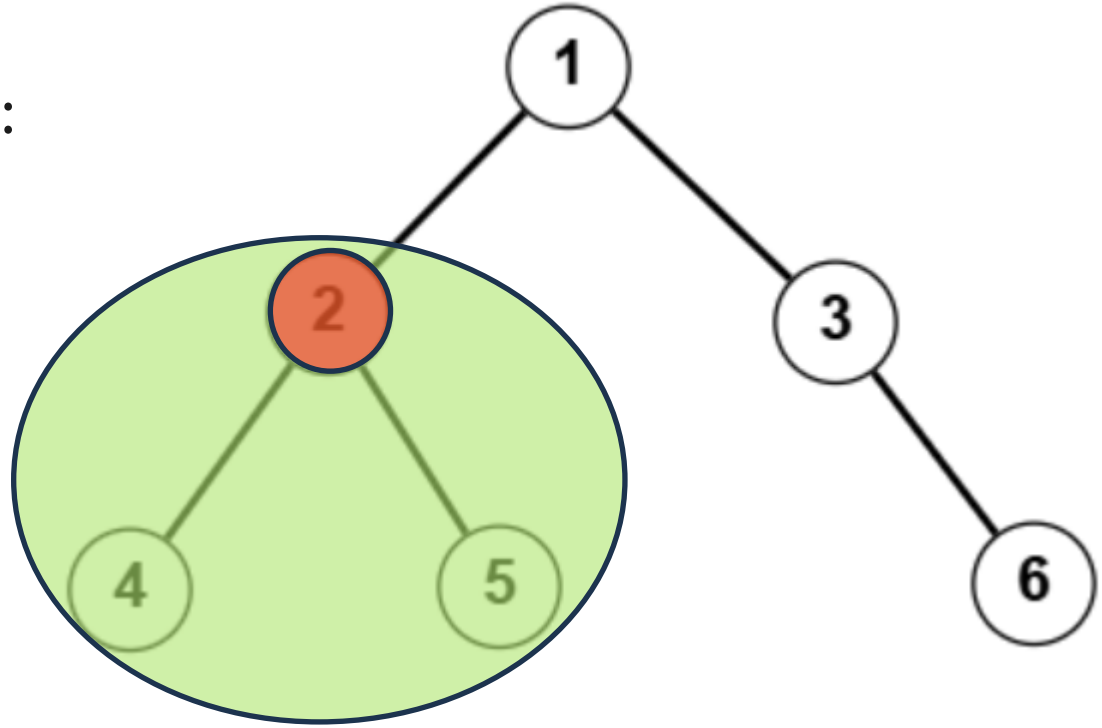
ÁRBOLES BINARIOS (BINARY TREE)

Postorden:

Izquierda - Derecha - Nodo

Recorrido:

4 – 5 –2



ÁRBOLES BINARIOS (BINARY TREE)

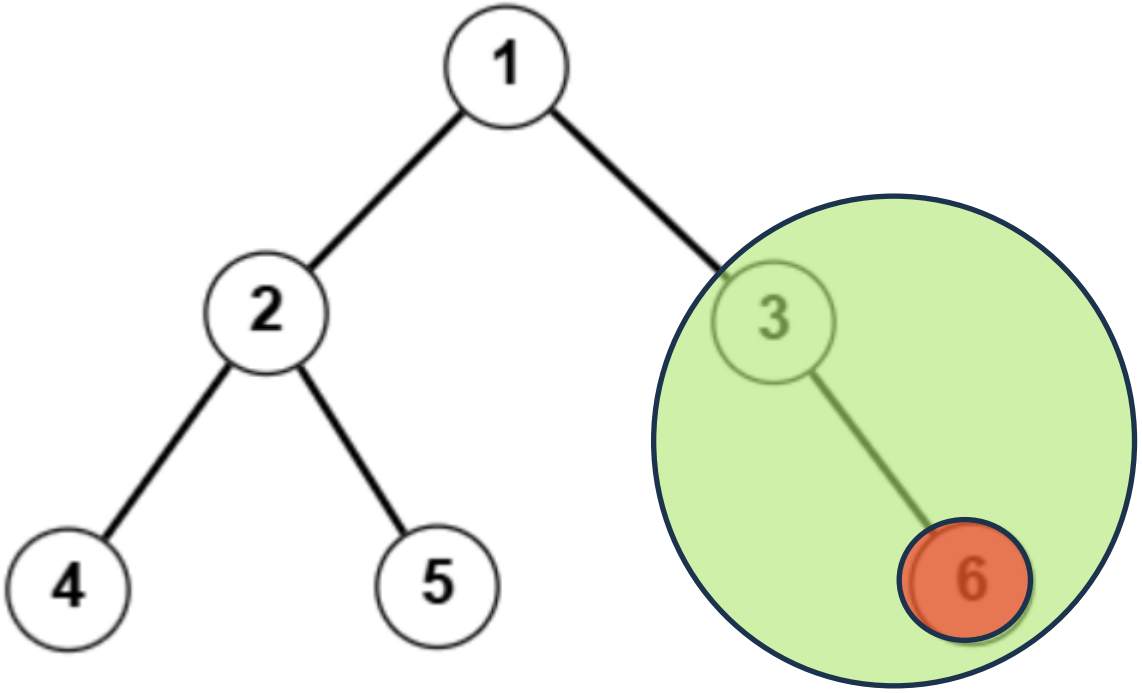
Postorden:

Izquierda - Derecha - Nodo

Recorrido:

4 – 5 –2 -

6



ÁRBOLES BINARIOS (BINARY TREE)

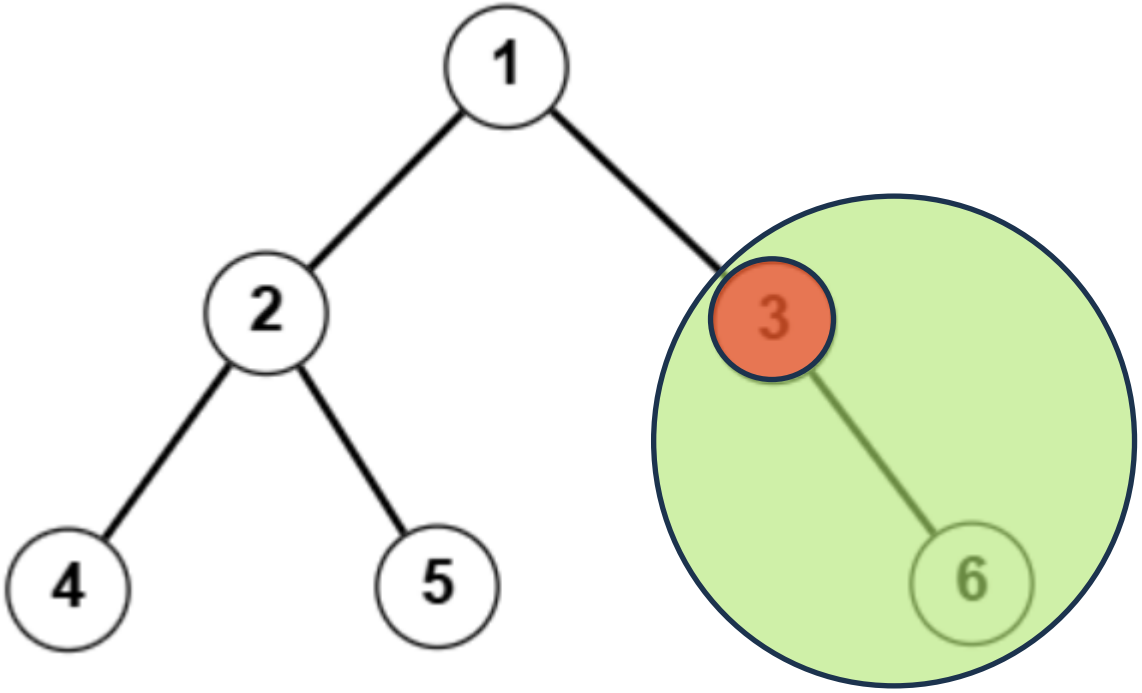
Postorden:

Izquierda - Derecha - Nodo

Recorrido:

4 – 5 –2 -

6 – 3



ÁRBOLES BINARIOS (BINARY TREE)

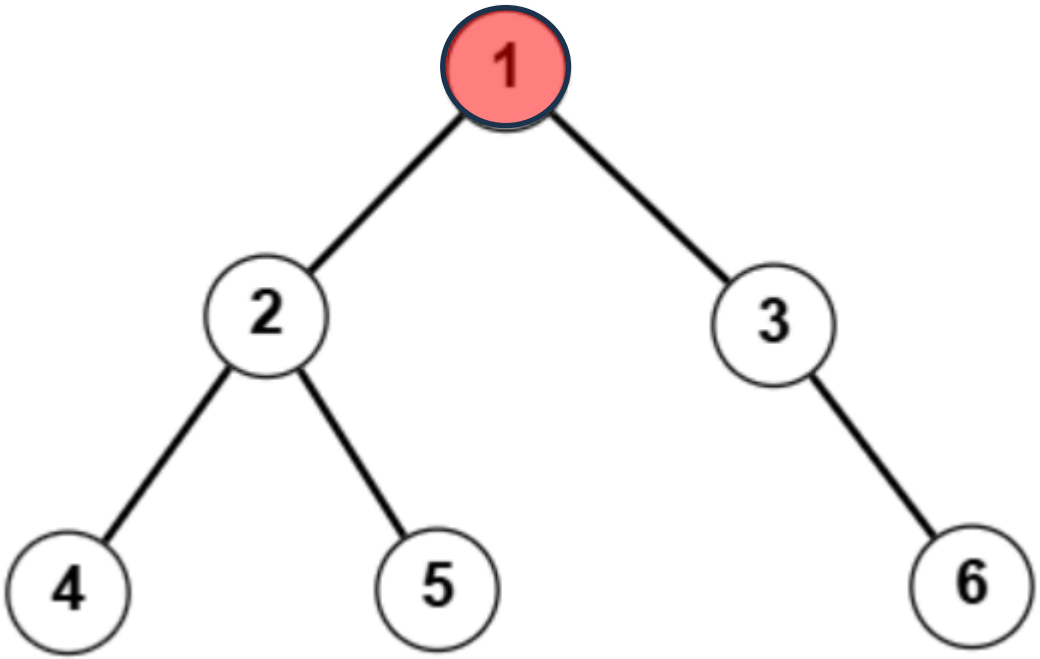
Postorden:

Izquierda - Derecha - Nodo

Recorrido:

4 – 5 –2 -

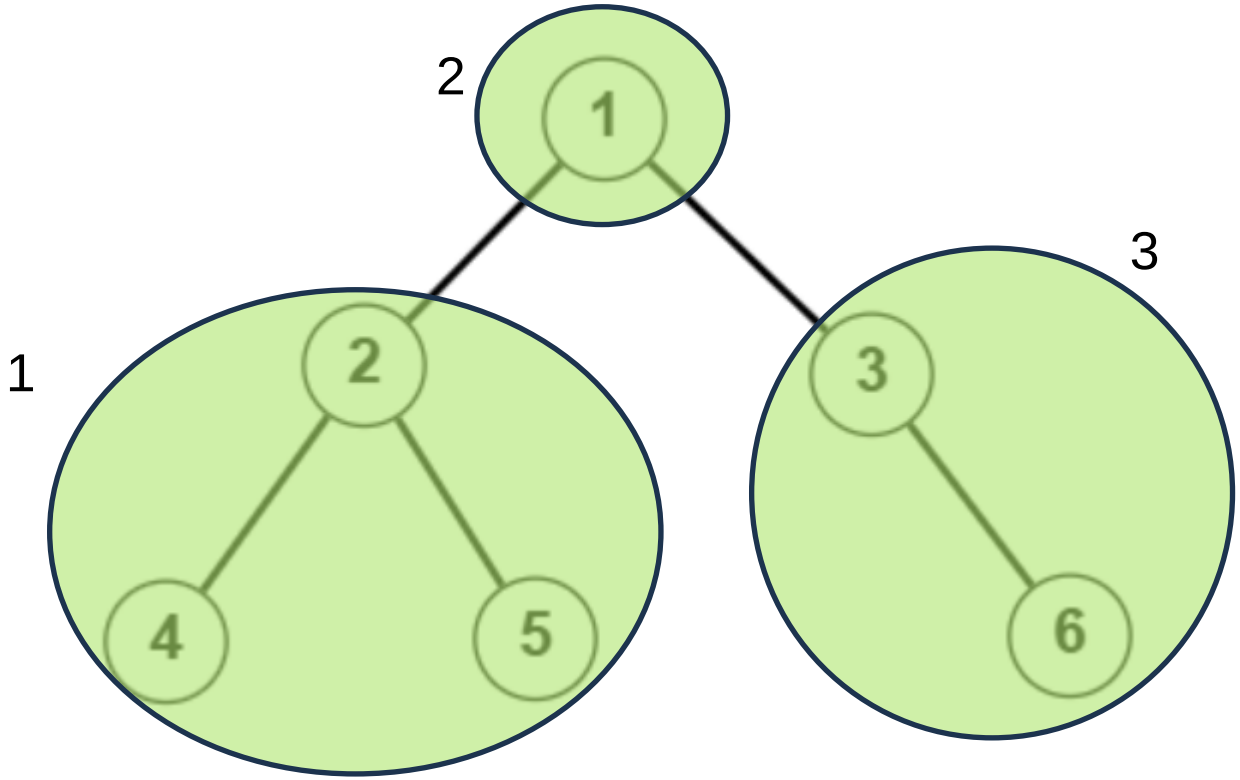
6 – 3 – 1



ÁRBOLES BINARIOS (BINARY TREE)

Inorden:

Izquierda - Nodo - Derecha



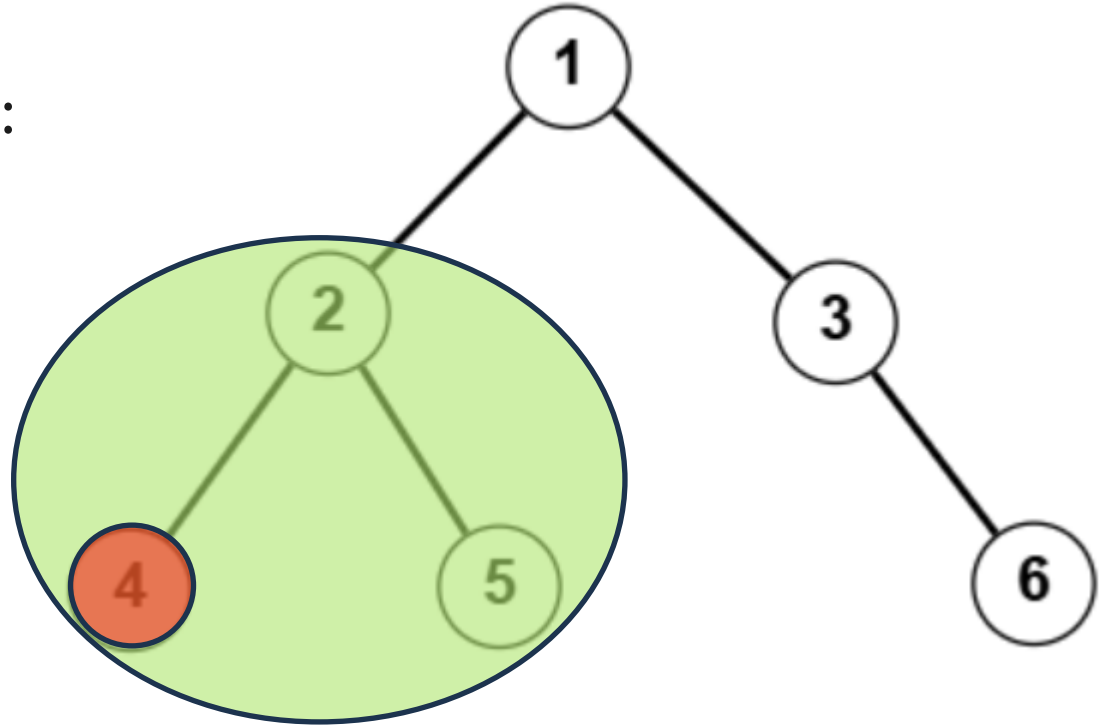
ÁRBOLES BINARIOS (BINARY TREE)

Inorden:

Izquierda - Nodo - Derecha

Recorrido:

4



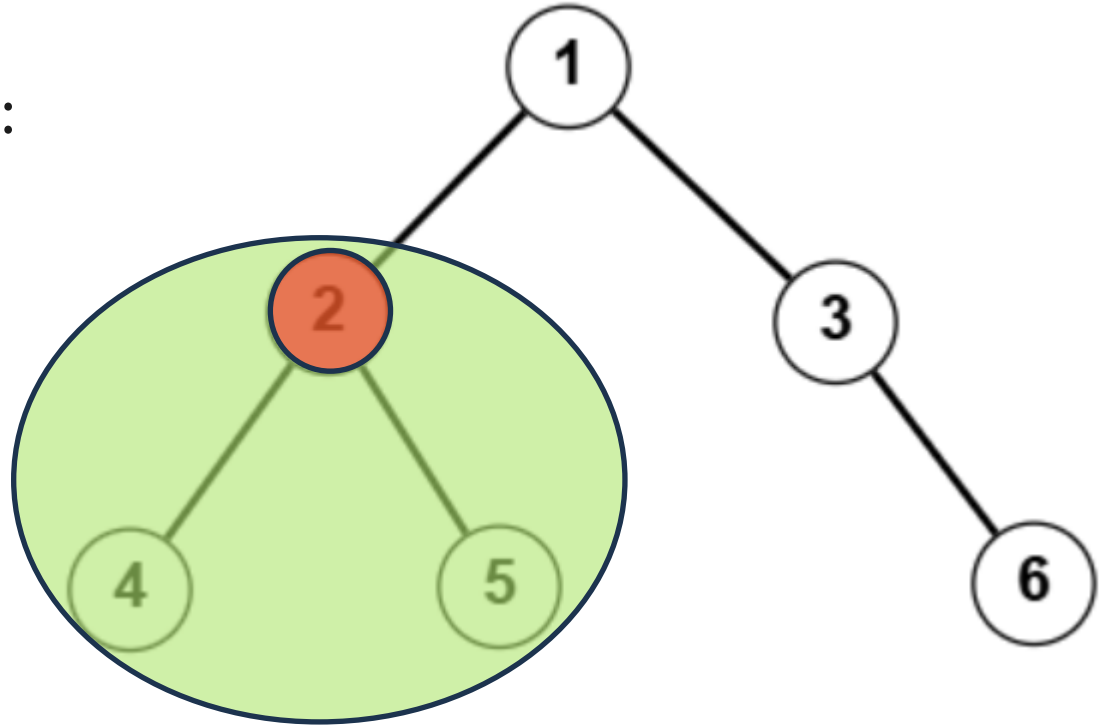
ÁRBOLES BINARIOS (BINARY TREE)

Inorden:

Izquierda - Nodo - Derecha

Recorrido:

4 – 2



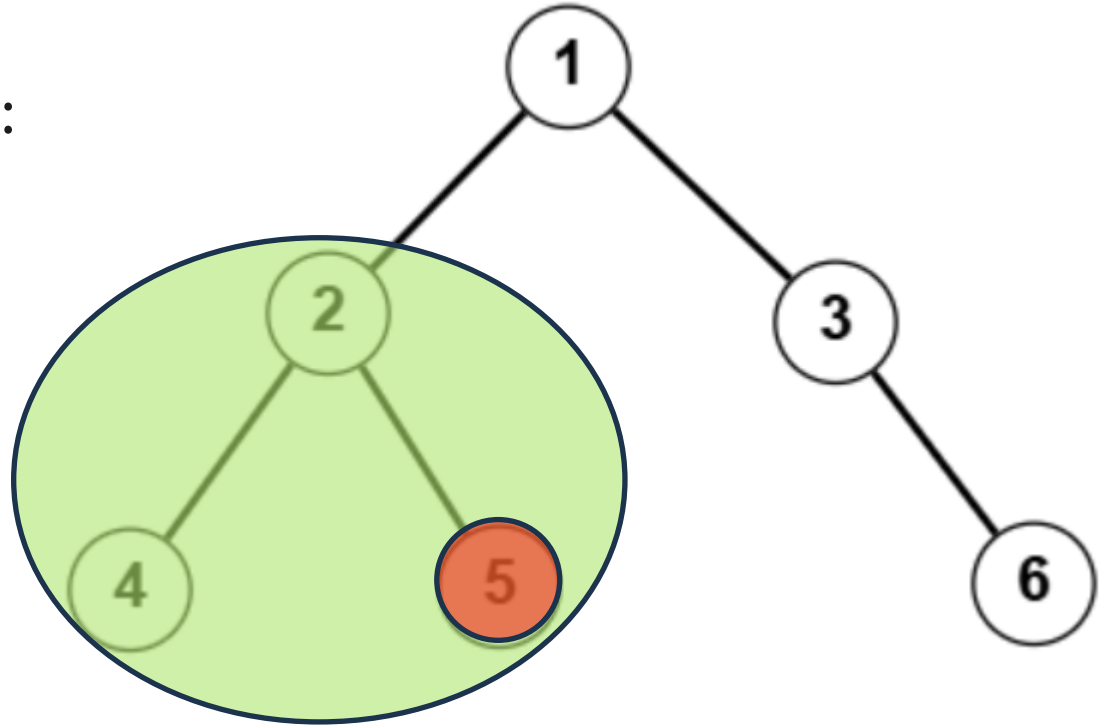
ÁRBOLES BINARIOS (BINARY TREE)

Inorden:

Izquierda - Nodo - Derecha

Recorrido:

4 – 2 – 5



ÁRBOLES BINARIOS (BINARY TREE)

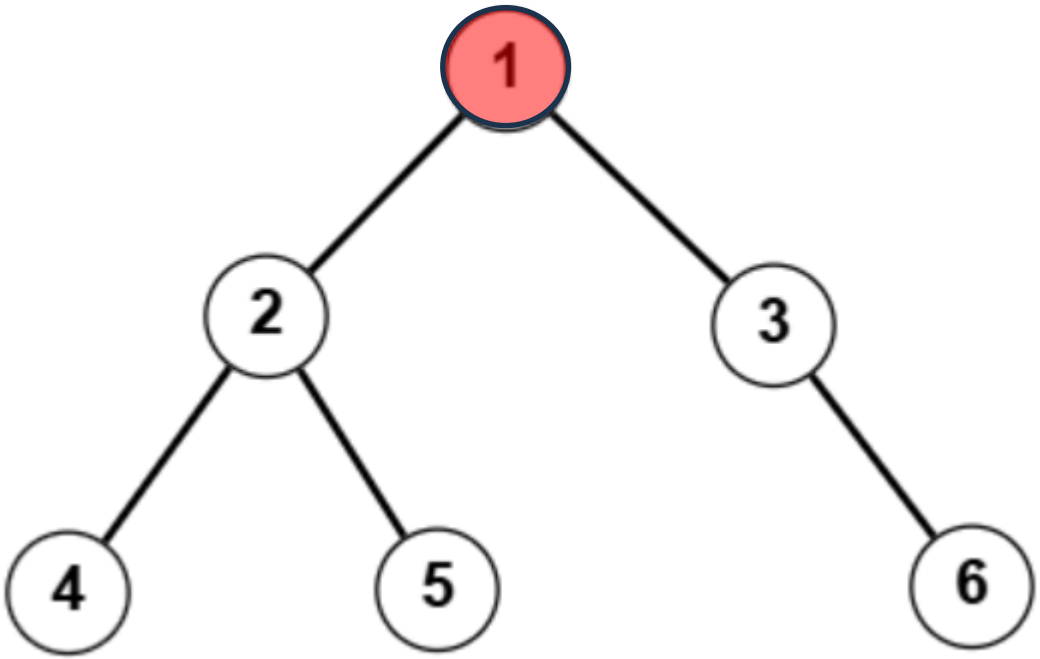
Inorden:

Izquierda - Nodo - Derecha

Recorrido:

4 – 2 – 5 -

1



ÁRBOLES BINARIOS (BINARY TREE)

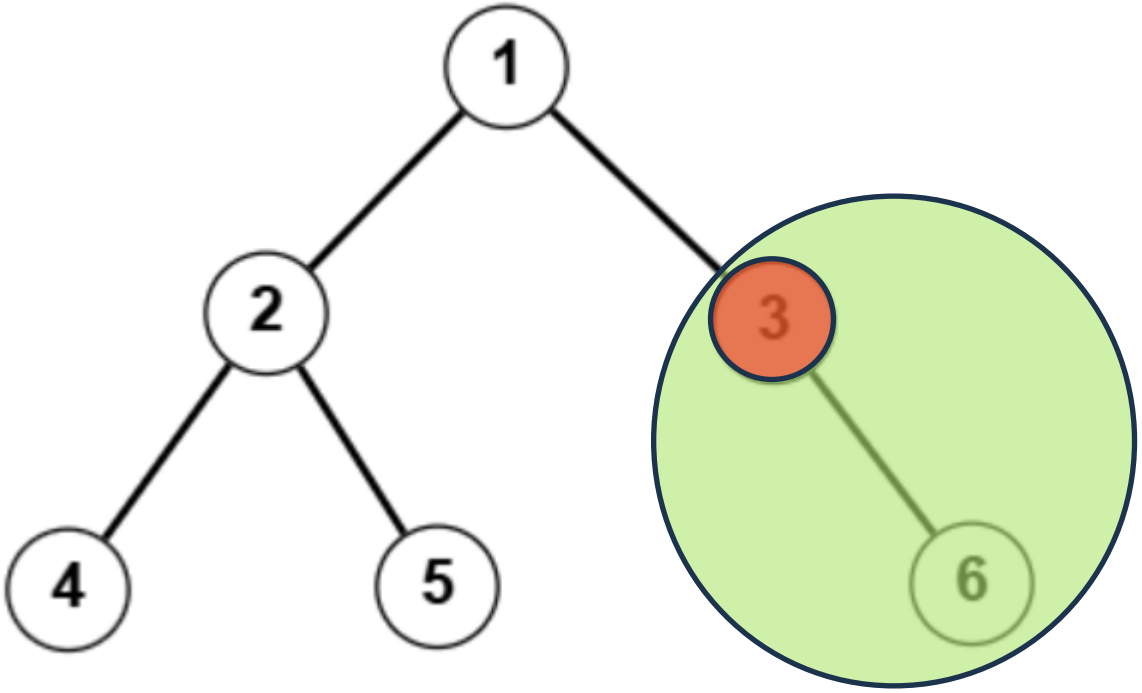
Inorden:

Izquierda - Nodo - Derecha

Recorrido:

4 – 2 – 5 –

1 – 3



ÁRBOLES BINARIOS (BINARY TREE)

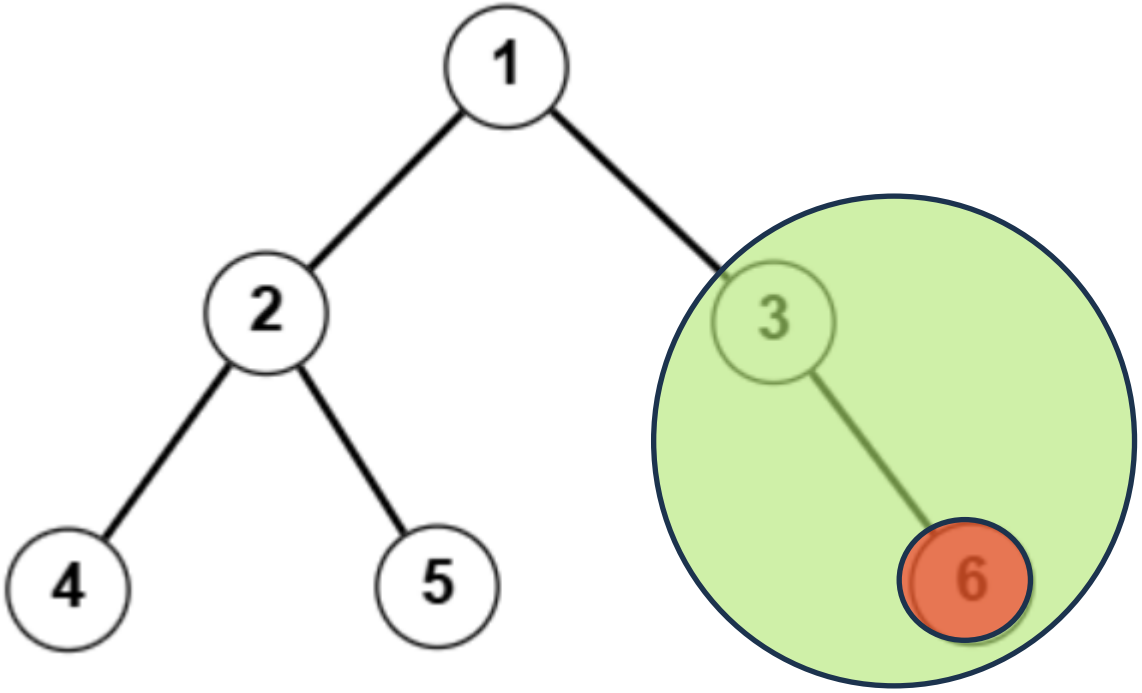
Inorden:

Izquierda - Nodo - Derecha

Recorrido:

4 – 2 – 5 –

1 – 3 – 6



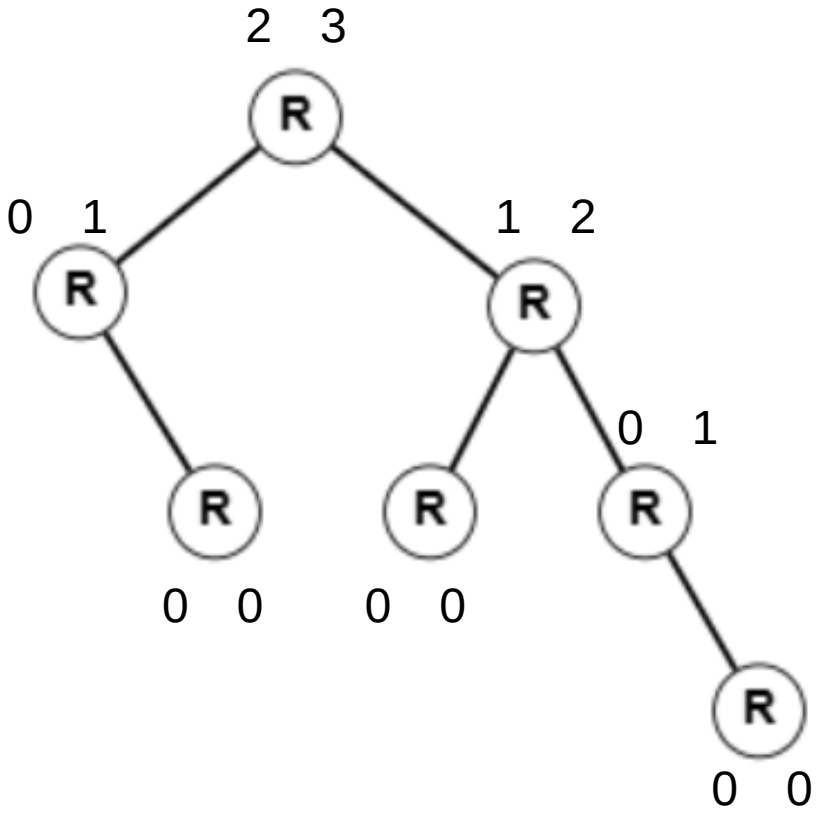
¿Está el árbol equilibrado?

- https://aceptaelreto.com/problem/statement.php?id=275
- ID: 275
- Ejemplos:

RRR..R..R.R..	→	SI
RR.R..RR..R.R..	→	SI
RR..RR..R.R..	→	NO

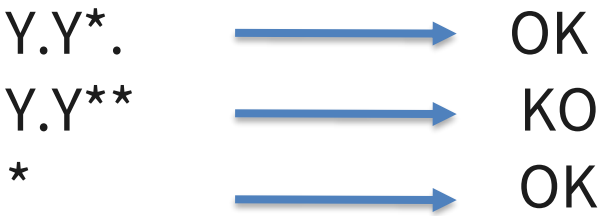
¿Está el árbol equilibrado?

RR.R..RR..R.R.. → SI



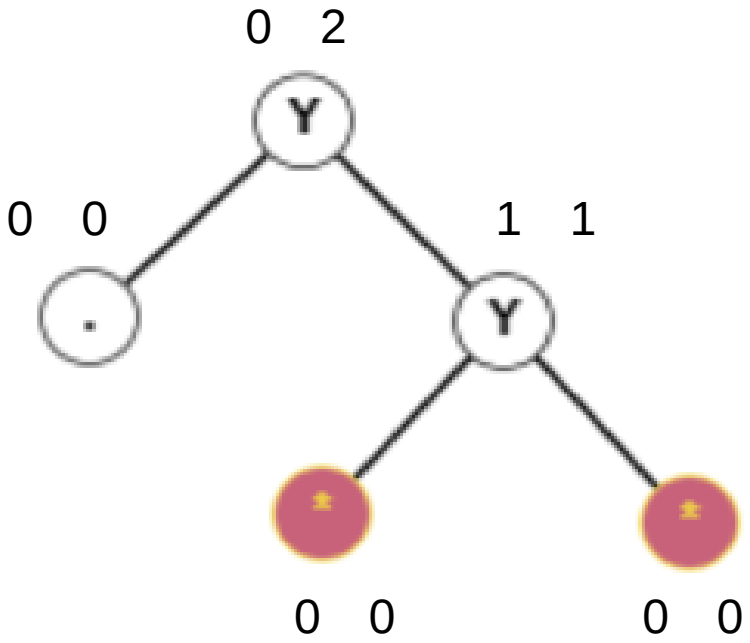
Árbol de navidad

- https://aceptaelreto.com/problem/statement.php?id=204
- ID: 204
- Ejemplos:

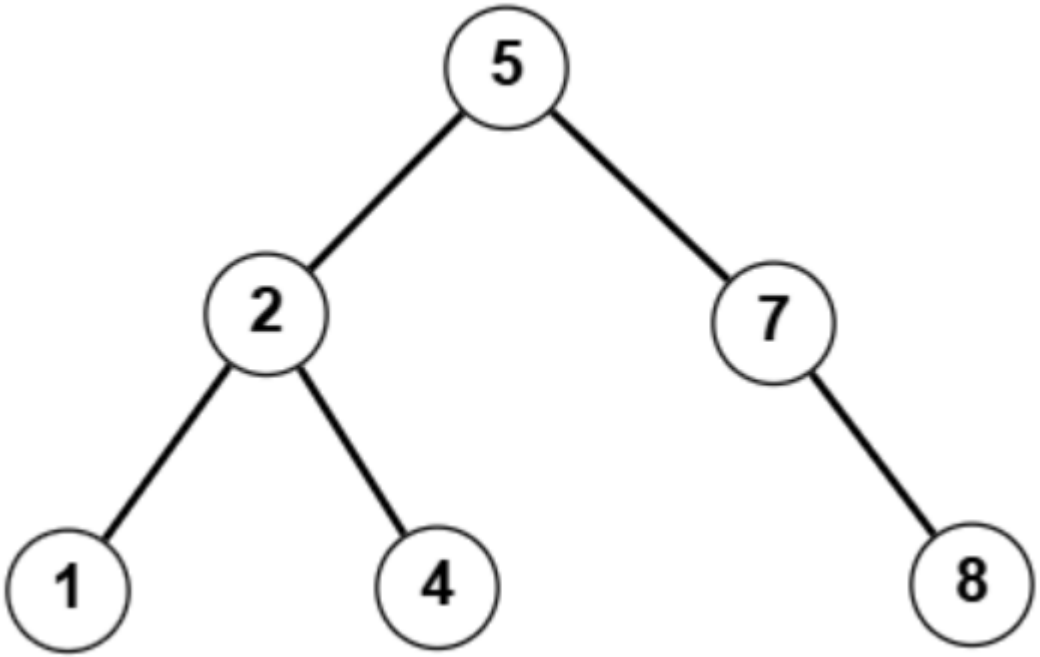


Árbol de navidad

Y.Y** → KO

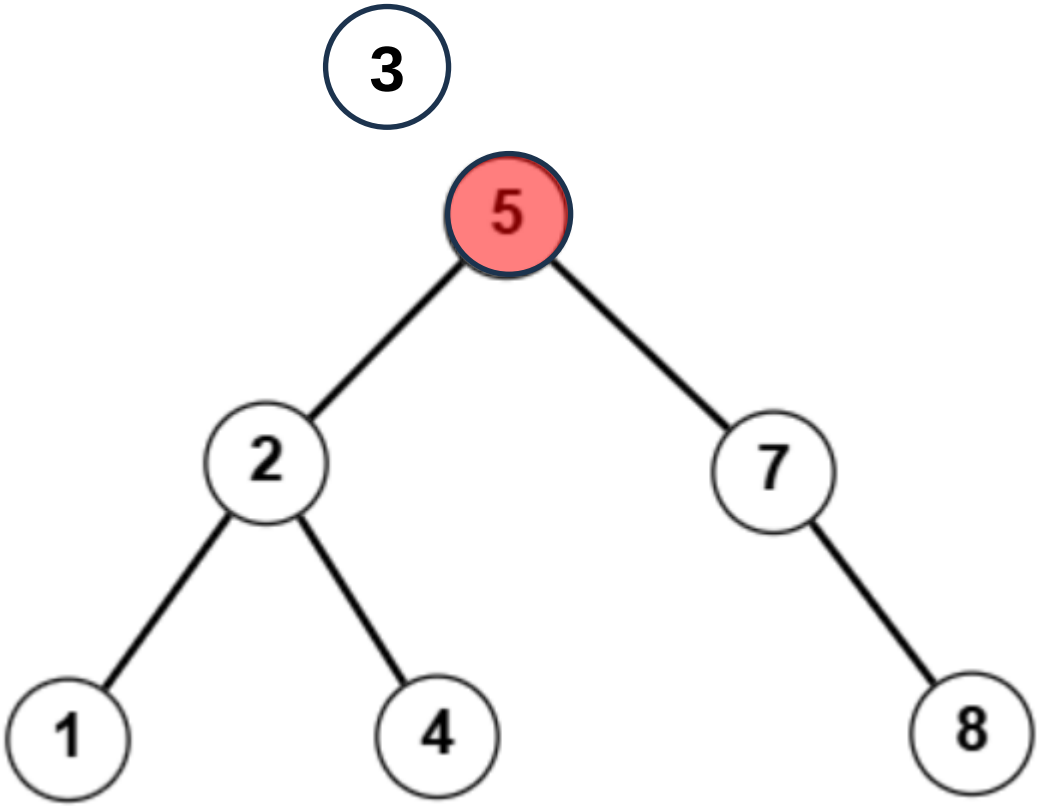


ÁRBOLES BINARIOS DE BÚSQUEDA (BINARY SEARCH TREE)



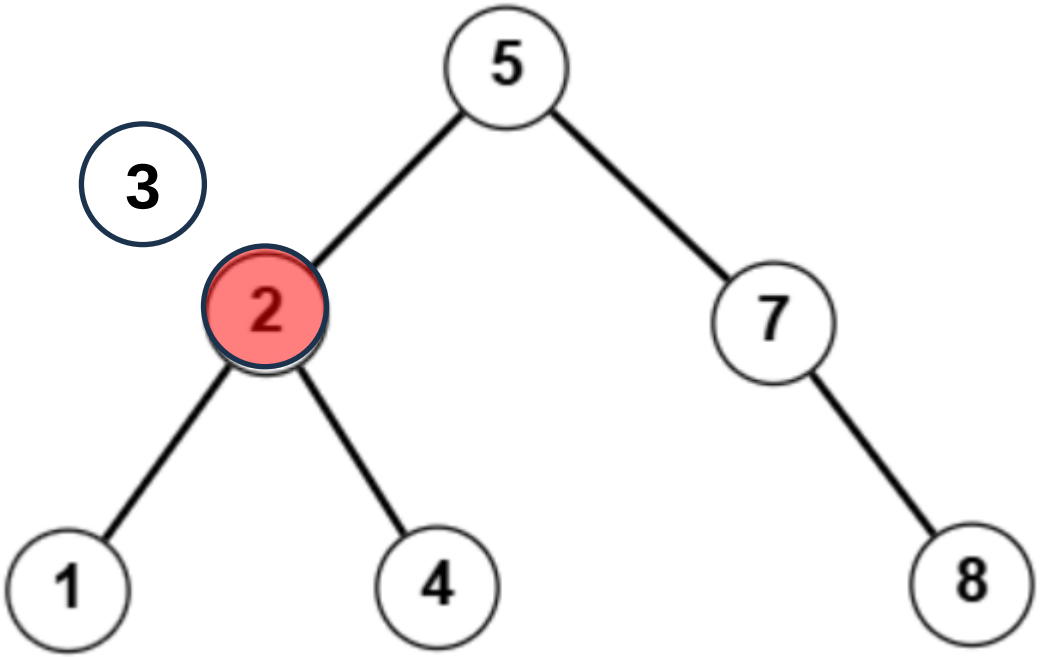
- Máximo dos hijos por nodo
- Todos los hijos a la izquierda de un nodo son menores
- Todos los hijos a la derecha de un nodo son mayores

ÁRBOLES BINARIOS DE BÚSQUEDA (BINARY SEARCH TREE)



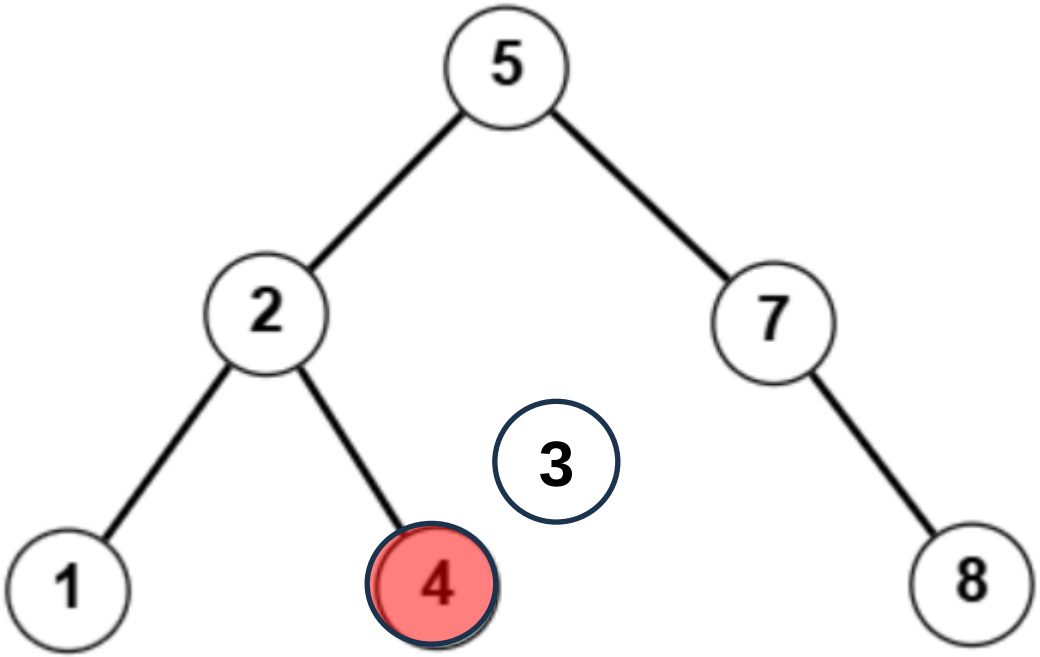
3 < 5 : a la izquierda

ÁRBOLES BINARIOS DE BÚSQUEDA (BINARY SEARCH TREE)



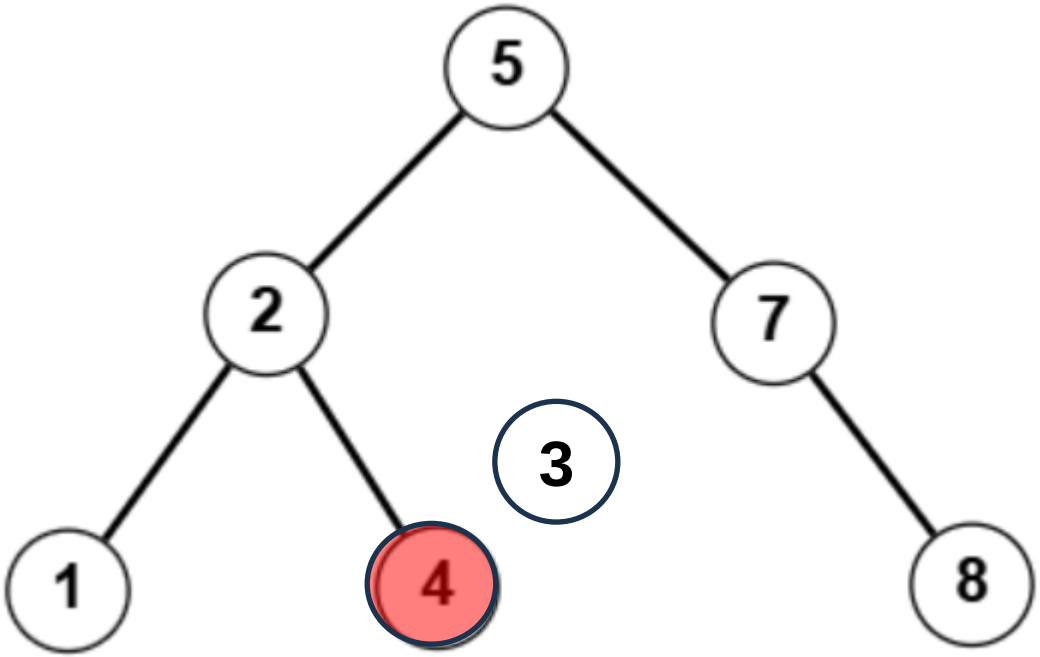
3 > 2 : a la derecha

ÁRBOLES BINARIOS DE BÚSQUEDA (BINARY SEARCH TREE)



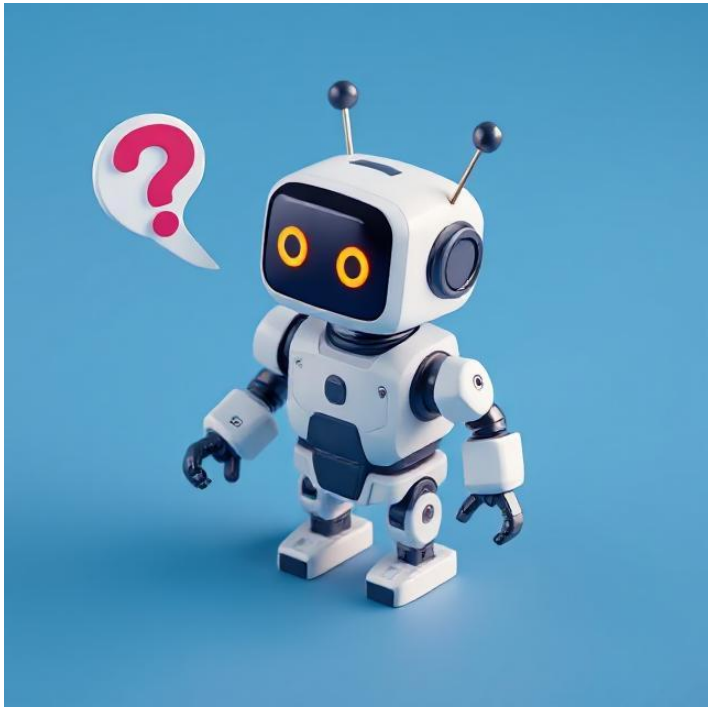
3 < 4 : a la izquierda

ÁRBOLES BINARIOS DE BÚSQUEDA (BINARY SEARCH TREE)



3 < 4 : a la izquierda

¿PREGUNTAS?



HASTA LA SEMANA QUE VIENE!

@URJC_CP

@Dijkstraidos

Bingo Ties

- https://open.kattis.com/problems/bingoties
- ID: bingoties
- Ejemplo:

```
2
3 29 45 56 68
1 19 43 50 72
11 25 40 49 61
9 23 31 58 63
4 27 42 54 71

14 23 39 59 63
8 17 35 55 61
15 26 42 53 71
10 25 31 57 64
6 20 44 52 68
```

```
1 2
```