

Curso de

# **Presentación de Entrevistas con Estructuras de Datos y Algoritmos**

@camilalonart

**¿Qué son las  
estructuras de  
datos y algoritmos?**

# **Próximos Cursos Avanzados de Algoritmos y Estructuras de Datos:**

- Patrones de Arrays y Strings
- Estructuras de Datos Lineales
- Grafos y Árboles
- Más ejercicios

# Calculando la complejidad de algoritmos

# Complejidad Lineal

```
def complejidad_lineal(lista):
    suma = 0
    multiplicacion = 1

    for numero in range(len(lista)):
        suma += numero

    for numero in range(len(lista)):
        multiplicacion *= numero

    return suma, multiplicacion
```

# Complejidad Lineal

```
def complejidad_lineal(lista):
    suma = 0
    multiplicacion = 1
    for numero in range(len(lista)):
        suma += numero
    for numero in range(len(lista)):
        multiplicacion *= numero
    return suma, multiplicacion
```

The diagram illustrates the time complexity of the given Python function. It uses brackets to group code segments and arrows to map them to their respective complexities:

- The first two assignments (`suma = 0` and `multiplicacion = 1`) are grouped by a bracket and mapped to  $O(1)$ .
- The loop `for numero in range(len(lista)):` is grouped by a bracket and mapped to  $O(n)$ .
- The assignment `suma += numero` is part of the loop and is not explicitly bracketed but is implied to be included in the  $O(n)$  complexity.
- The second loop `for numero in range(len(lista)):` is grouped by a bracket and mapped to  $O(n)$ .
- The assignment `multiplicacion *= numero` is part of the second loop and is not explicitly bracketed but is implied to be included in the  $O(n)$  complexity.

# Complejidad Lineal

```
def complejidad_lineal(lista):
    suma = 0
    multiplicacion = 1
    for numero in range(len(lista)):
        suma += numero
    for numero in range(len(lista)):
        multiplicacion *= numero
    return suma, multiplicacion
```

~~O(1)~~

+

O(n)

+

O(n)

---

~~O(2n)~~

↓

O(n)

# Complejidad Lineal

```
def complejidad_lineal_2(lista):
    calculo = 0

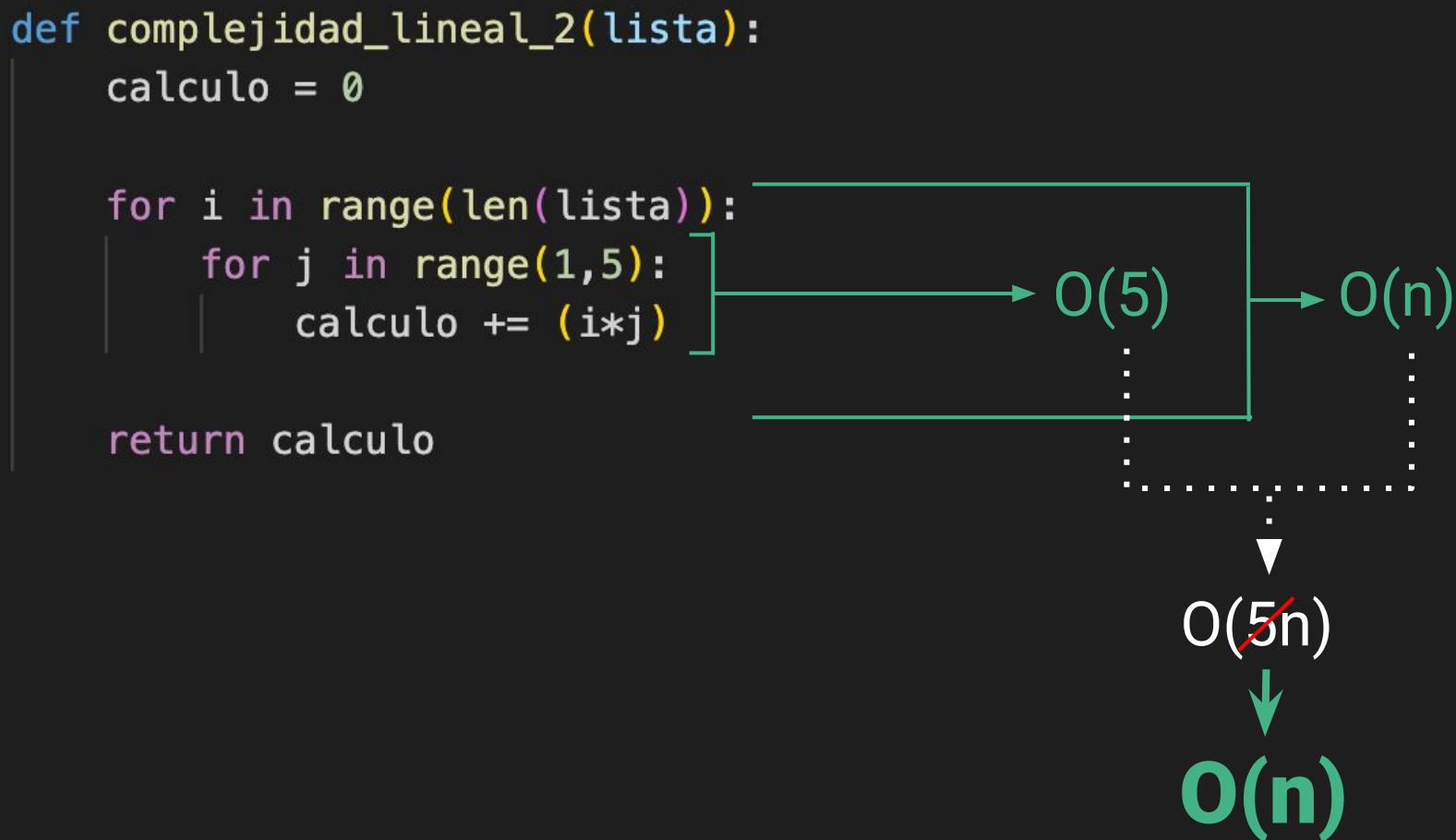
    for i in range(len(lista)):
        for j in range(1,5):
            calculo += (i*j) → O(5)

    return calculo → O(n)
```

# Complejidad Lineal

```
def complejidad_lineal_2(lista):
    calculo = 0

    for i in range(len(lista)):
        for j in range(1,5):
            calculo += (i*j)
```



The diagram illustrates the time complexity analysis of the given Python code. It shows the nested loops and their respective complexities. The inner loop, which iterates from 1 to 5, is grouped by a green bracket and labeled  $O(5)$ . This is then multiplied by the outer loop, which iterates from 0 to the length of the list, and is also labeled  $O(n)$ . The final result is  $O(5n)$ , which is crossed out in red, indicating that the dominant term is  $O(n)$ .

# Complejidad Logarítmica

```
def busqueda_binaria(lista):
    apuntador_izquierdo = 0
    apuntador_derecho = len(nums) - 1

    while apuntador_izquierdo <= apuntador_derecho:
        mitad = (apuntador_izquierdo+apuntador_derecho) // 2
        if nums[mitad] == target:
            return mitad
        elif nums[mitad] < target:
            apuntador_izquierdo = mitad + 1
        else:
            apuntador_derecho = mitad - 1
    return -1
```

Si nuestra lista es [1, 3, 8, 14, 15, 22, 22, 23, 28, 30, 32, 33, 44, 47] y el elemento no se encuentra en la lista

Iteración 1: recorremos [1, 3, 8, 14, 15, 22, 22, 23, 28, 30, 32, 33, 44, 47]

Iteración 2: recorremos [1, 3, 8, 14, 15, 22, 22]

Iteración 3: recorremos [1, 3, 8, 14]

Iteración 4: recorremos [1, 3]

Iteración 5: recorremos [1]

# Complejidad Logarítmica $O(\log n)$

```
def busqueda_binaria(lista):
    apuntador_izquierdo = 0
    apuntador_derecho = len(nums) - 1

    while apuntador_izquierdo <= apuntador_derecho:
        mitad = (apuntador_izquierdo+apuntador_derecho) // 2
        if nums[mitad] == target:
            return mitad
        elif nums[mitad] < target:
            apuntador_izquierdo = mitad + 1
        else:
            apuntador_derecho = mitad - 1
    return -1
```

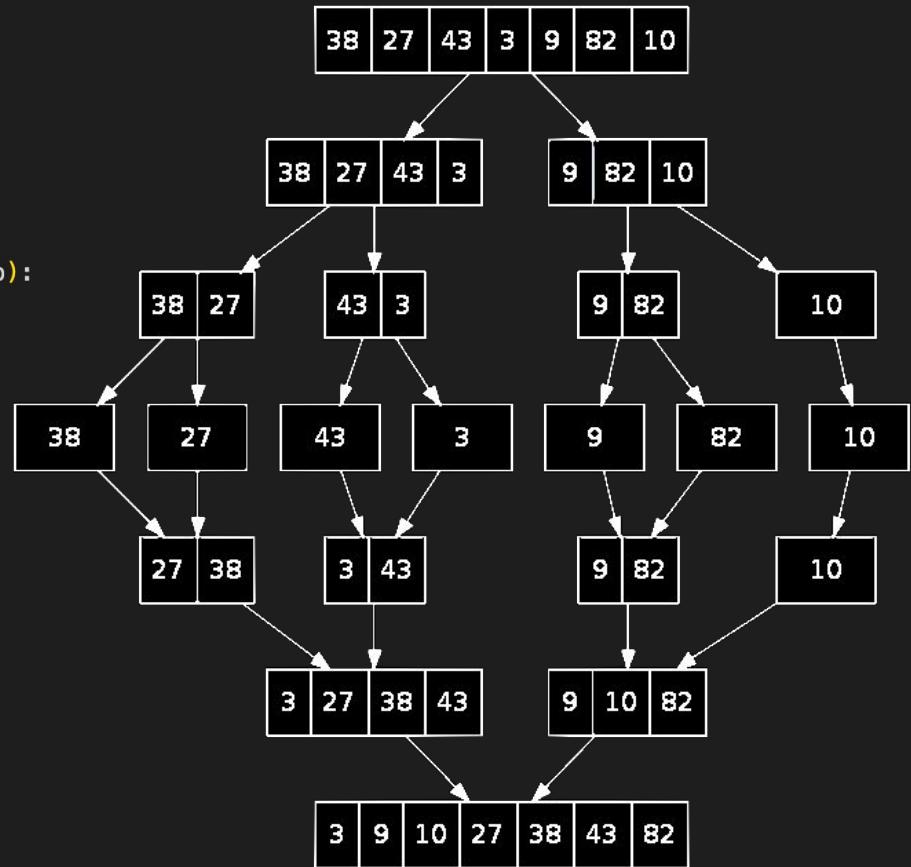
# Complejidad Logarítmica $O(\log n)$

*Algoritmo de ordenamiento Merge Sort*

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        izquierdo = arr[:mid]
        derecho = arr[mid:]
        merge_sort(izquierdo)
        merge_sort(derecho)
        i = j = k = 0
        while i < len(izquierdo) and j < len(derecho):
            if izquierdo[i] < derecho[j]:
                arr[k] = izquierdo[i]
                i += 1
            else:
                arr[k] = derecho[j]
                j += 1
            k += 1

        while i < len(izquierdo):
            arr[k] = izquierdo[i]
            i += 1
            k += 1

        while j < len(derecho):
            arr[k] = derecho[j]
            j += 1
            k += 1
```



# Complejidad Cuadrática

```
def complejidad_cuadratica(matriz):
    for i in range(len(matriz)):
        for j in range(len(matriz[0])):
            if num[i][j] != 0:
                print(num[i][j])
```

# Complejidad Cuadrática

```
def complejidad_cuadratica(matriz):
    for i in range(len(matriz)):
        for j in range(len(matriz[0])):
            if num[i][j] != 0:
                print(num[i][j])
```

The diagram illustrates the time complexity analysis of the given Python code. A green box encloses the inner loop, which iterates over the columns of the matrix. An arrow points from this box to the text "O(n)", indicating the time complexity of this inner loop. Another arrow points from this "O(n)" text to a dashed arrow labeled "O(n \* n)", indicating the time complexity of the entire nested loop structure. A large green arrow points down from "O(n \* n)" to the final result "O(n<sup>2</sup>)", indicating the overall time complexity of the function.

$O(n^2)$

# Complejidad Cuadrática

```
def three_sum(numeros: List[int]) -> List[List[int]]:
    if len(numeros) < 3:
        return []
    numeros.sort()
    resultado = set()
    for i in range(len(numeros)-2):
        if numeros[i] <= 0:
            if i == 0 or numeros[i-1]<numeros[i]:
                mapaParejas = {}
                for num in numeros[i+1:]:
                    if num not in mapaParejas:
                        mapaParejas[-numeros[i]-num] = 1
                    else:
                        resultado.add((numeros[i], num, - numeros[i]))
    return [list(group) for group in resultado]
```

# Complejidad Cuadrática

```
def three_sum(numeros: List[int]) -> List[List[int]]:  
    if len(numeros) < 3:  
        return []  
    numeros.sort() → O(n log n)  
    resultado = set()  
    for i in range(len(numeros)-2):  
        if numeros[i] <= 0:  
            if i == 0 or numeros[i-1]<numeros[i]:  
                mapaParejas = {}  
                for num in numeros[i+1:]:  
                    if num not in mapaParejas:  
                        mapaParejas[-numeros[i]-num] = 1  
                    else:  
                        resultado.add((numeros[i], num, - numeros[i])) → O(n)  
    return [list(group) for group in resultado] → O(n)
```

The diagram illustrates the time complexity of the `three_sum` function. It shows the following complexity steps:

- The initial sort operation is  $O(n \log n)$ .
- The nested loops (from the `for` loop to the `resultado.add` call) are  $O(n)$ .
- The final list comprehension is  $O(n)$ .
- The total complexity of the function is  $O(n * n)$ .

# Complejidad Cuadrática

```
def three_sum(numeros: List[int]) -> List[List[int]]:  
    if len(numeros) < 3:  
        return []  
    numeros.sort() → O(n log n)  
    resultado = set()  
    for i in range(len(numeros)-2):  
        if numeros[i] <= 0:  
            if i == 0 or numeros[i-1]<numeros[i]:  
                mapaParejas = {}  
                for num in numeros[i+1:]:  
                    if num not in mapaParejas:  
                        mapaParejas[-numeros[i]-num] = 1  
                    else:  
                        resultado.add((numeros[i], num, - numeros[i])) → O(n)  
    return [list(group) for group in resultado] → O(n)
```

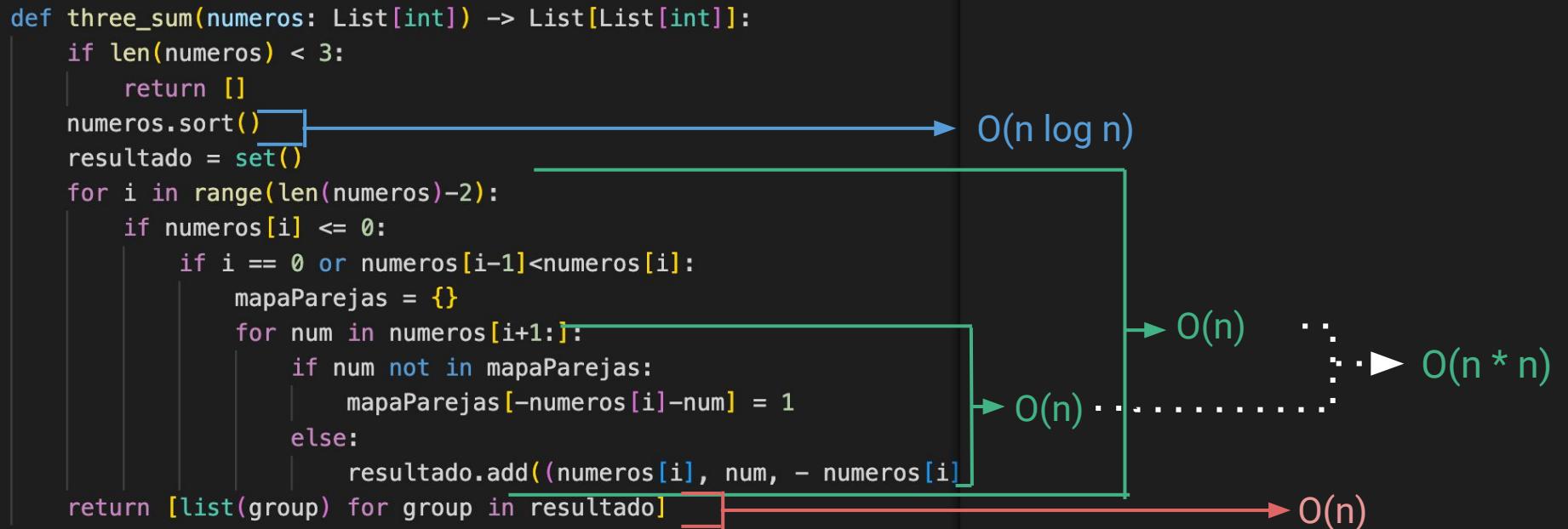
$$O(n \log n) + O(n^2) + O(n)$$

# Complejidad Cuadrática

```
def three_sum(numeros: List[int]) -> List[List[int]]:  
    if len(numeros) < 3:  
        return []  
    numeros.sort() → O(n log n)  
    resultado = set()  
    for i in range(len(numeros)-2):  
        if numeros[i] <= 0:  
            if i == 0 or numeros[i-1]<numeros[i]:  
                mapaParejas = {}  
                for num in numeros[i+1:]:  
                    if num not in mapaParejas:  
                        mapaParejas[-numeros[i]-num] = 1  
                    else:  
                        resultado.add((numeros[i], num, - numeros[i])) → O(n)  
    return [list(group) for group in resultado] → O(n)
```

¿Cuál es el más grande? →  $O(n \log n)$  +  $O(n^2)$  +  $O(n)$

# Complejidad Cuadrática



$$O(n \log n) + O(n^2) + O(n)$$

$$\downarrow \\ O(n^2) = O(n^2)$$

# **Estructuras de datos y algoritmos a estudiar**

Patrones

Estructuras  
de datos

*Resolver  
problemas*

Algoritmos

# Estructuras de datos

- Arreglos
- Strings
- Listas encadenadas
- Tablas de Hash y Conjuntos de Hash
- Pilas
- Colas
- Árboles
- Grafos
- Tries

# Conceptos y Algoritmos

- Notación Big O
- Algoritmos de ordenamiento
- Algoritmos de búsqueda
- DFS y BFS
- Manipulación de bits
- Recursión
- Programación dinámica

# Patrones

- Dos Apuntadores
- Ventana Deslizante
- Apuntador rápido y lento

Entre otros...

**¿Cómo es (comúnmente) una  
entrevista con problemas  
de programación?**

# **Recursos útiles**

# **Libros**

Cracking the Coding Interview - Gayle Laakmann McDowell

Grokking Algorithms - Aditya Bhargava

Clean Code - Robert Martin

Introduction to Algorithms - Thomas H. Cormen

## **Para resolver problemas de programación**

Leetcode

HackerRank

CodeChef

## **Para practicar entrevistas con otras personas**

Pramp

interviewing.io/

## **Para visualizar estructuras de datos y algoritmos**

<https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>

Visualgo

# **End of course**

**- No eliminar -**

Curso Avanzado de

# **Estructuras de Datos y Algoritmos: Patrones de Arrays y Strings**

@camilalonart

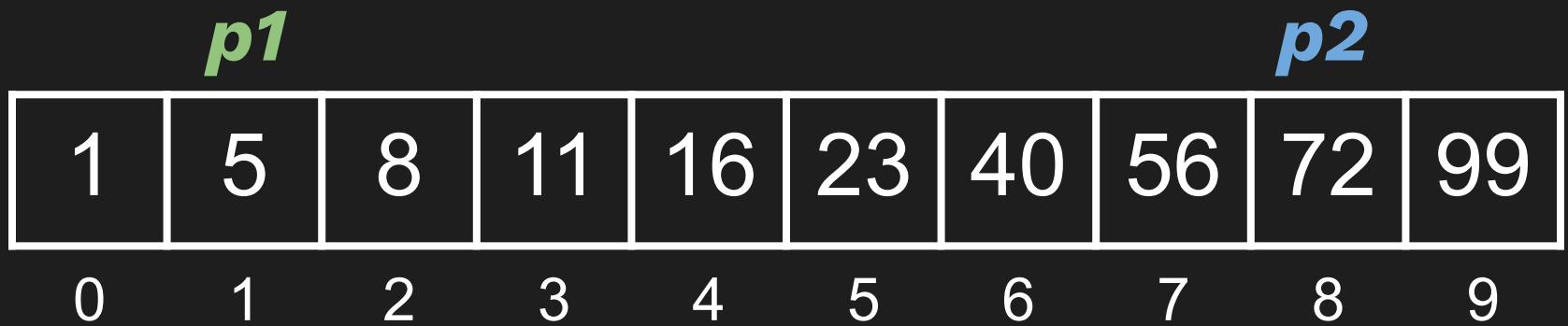
# ¿Qué aprenderemos?

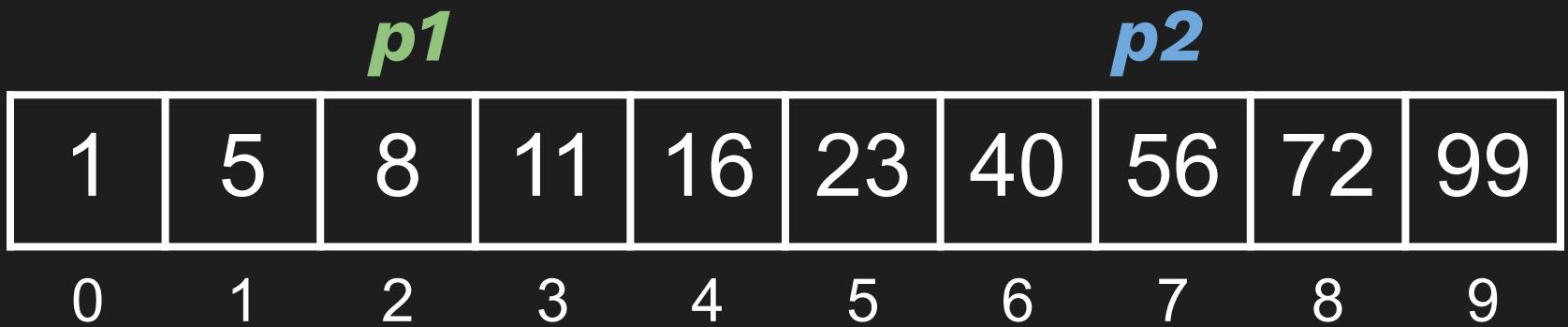
- Dos Apuntadores
- Ventana Deslizante
- Búsqueda Binaria

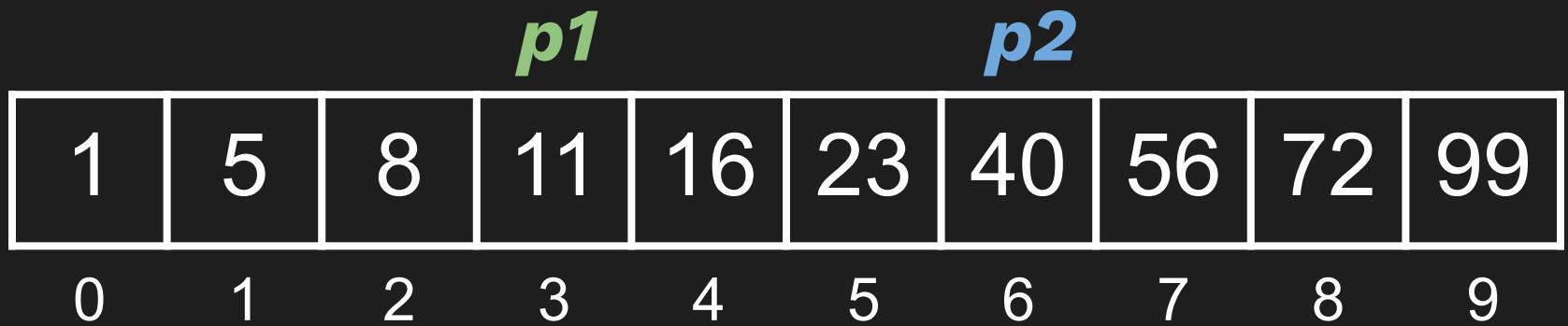
Patrón de  
**Dos Apuntadores**

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

<i>p1</i>	1	5	8	11	16	23	40	56	72	99	<i>p2</i>
0	1	2	3	4	5	6	7	8	9		



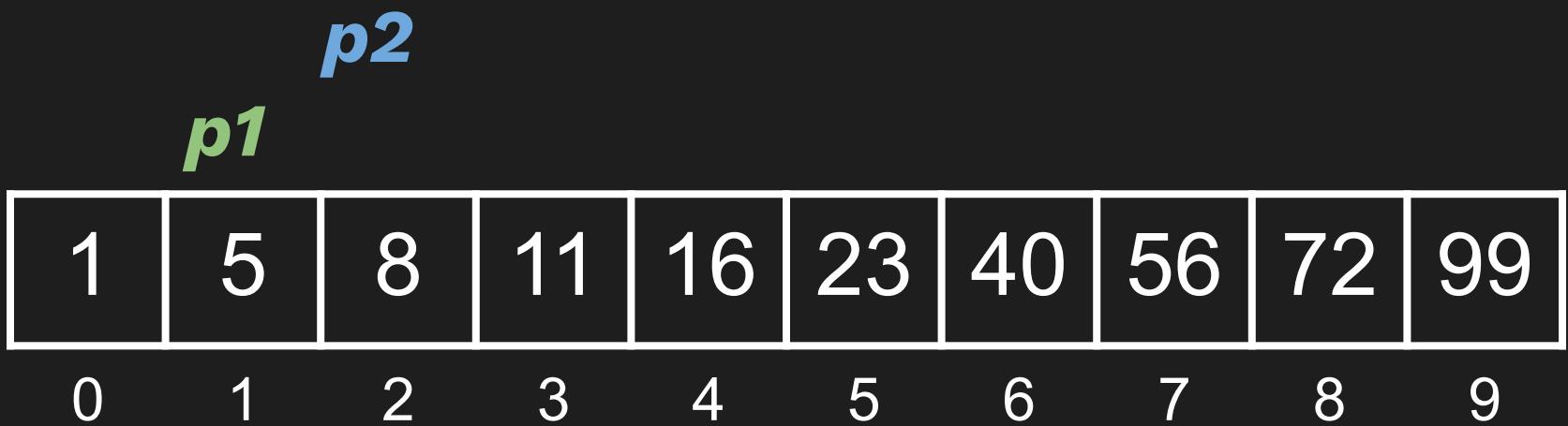


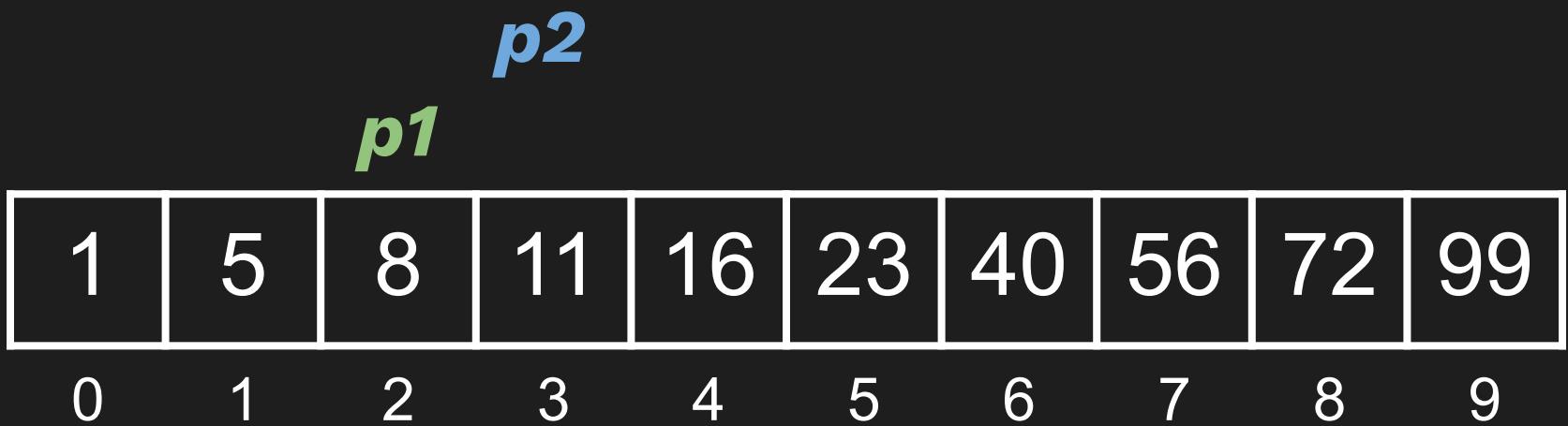


		<i>p1</i>	<i>p2</i>						
1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

<i>p2</i>									
<i>p1</i>	1	5	8	11	16	23	40	56	72
	0	1	2	3	4	5	6	7	8







Análisis del problema:  
**Verifying Alien  
Dictionary**

En una lengua alienígena, sorprendentemente, también utilizan las letras del español, pero posiblemente en un orden diferente, una permutación de nuestro alfabeto.

Dada una secuencia de palabras escritas en el idioma extranjero, y el orden del alfabeto, devuelve verdadero si y solo si las palabras dadas están ordenadas lexicográficamente en este idioma extranjero.

```
palabras = ["conocer", "cono"]
```

```
orden_alfabeto =
```

```
"abcdefghijklmnoprstuvwxyz"
```

 **Falso**

```
palabras = ["habito", "hacer", "lectura", "sonreir"]
```

```
orden_alfabeto = "habcdefghijklmnoprstuvwxyz"
```

 **Verdadero**

Diagrama de solución:

# **Verifying Alien Dictionary**

Solución en código:  
**Verifying Alien  
Dictionary**

Análisis del problema:

# Merge Two Sorted Lists

Dadas dos listas de números enteros  $\text{nums1}$  y  $\text{nums2}$ , cada una ordenada en orden ascendente, y dos enteros  $m$  y  $n$ , que representan la cantidad de elementos en  $\text{nums1}$  y  $\text{nums2}$  respectivamente.

Combinar  $\text{nums1}$  y  $\text{nums2}$  en un único array ordenado de forma ascendente.

Para ello,  $\text{nums1}$  tiene una longitud de  $m + n$ , donde los primeros  $m$  elementos denotan los elementos que deben ser combinados, y los últimos  $n$  elementos son 0 y deben ser ignorados.

**nums1** = [1,2,3,0,0,0]

**m** = 3



[1,2,2,3,5,6]

**nums2** = [2,5,6]

**n** = 3

Diagrama de solución:

# Merge Two Sorted Lists

Solución en código:

# Merge Two Sorted Lists

Análisis del problema:  
**Container With  
Most Water**

Dada una lista de números que representan un grupo de líneas de diferentes alturas.

Encuentra dos líneas que formen un contenedor, tal que este contenga la mayor cantidad de agua.

Devuelve la cantidad máxima de agua que puede almacenar un contenedor.

alturas = [1,8,6,2,5,4,8,3,7] → 49

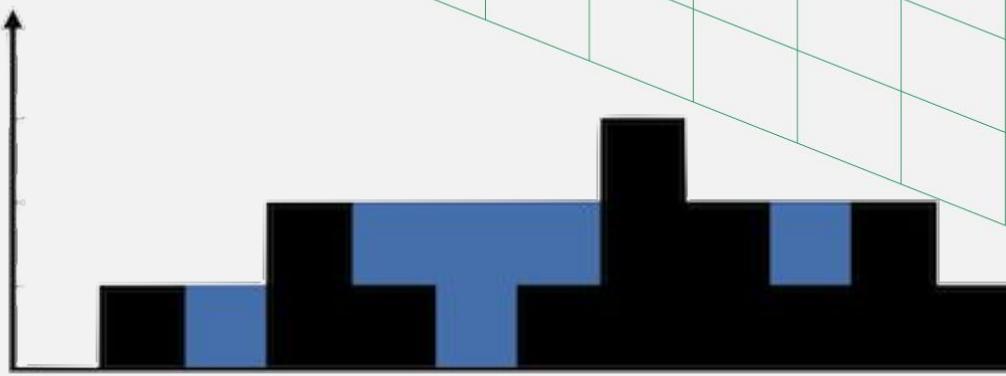
Diagrama de solución:  
**Container With  
Most Water**

Solución en código:  
**Container With  
Most Water**

Reto  
**Trapping  
rain water**

Dada una lista de números que representan un grupo de líneas de diferentes alturas.

Calcula cuánta agua puede atrapar después de llover.



$$\text{alturas} = [0,1,0,2,1,0,1,3,2,1,2,1] \rightarrow 6$$

Patrón de  
**Ventana**  
**Deslizante**

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

*p2*

*p1*

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

*p2*

*p1*

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

*p1*

*p2*

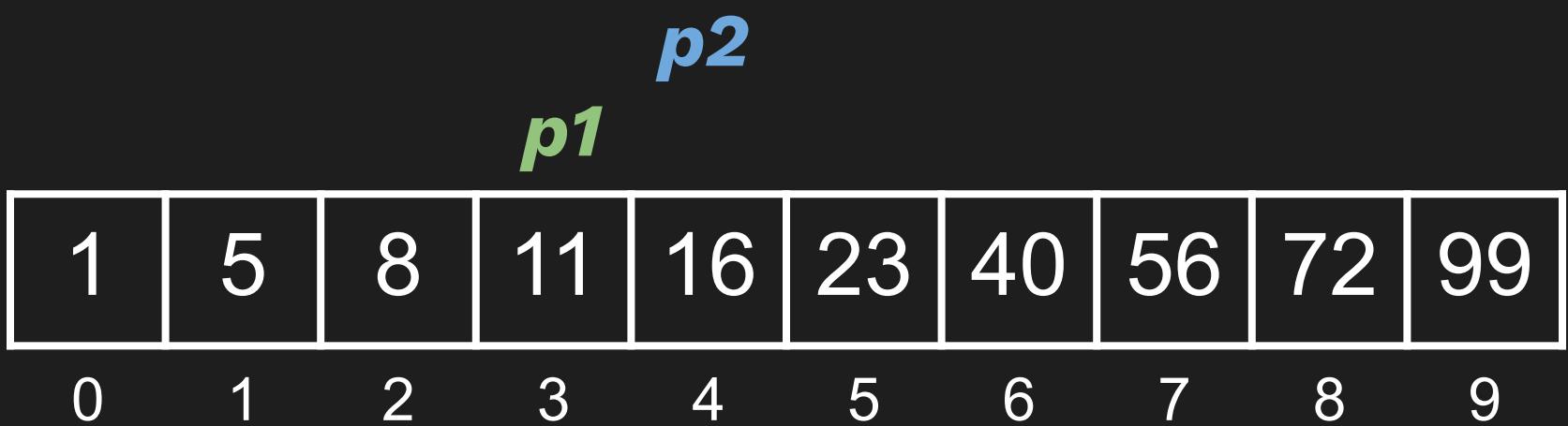
1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

*p1*

*p2*

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9







Análisis del problema:

**Longest substring  
without repeating  
characters**

Dada una cadena s, encontrar la longitud de la subcadena más larga sin repetir caracteres.

**s = "abcabcbb"** → 3

**s = "jdkafnlcdsalkxcmpoiuytfccv"** → 15

**Este algoritmo lo encontrarás  
también con estos nombres:**

- Longest Repeating  $k$  Chars Replacement
- Permutation in String
- Find all Anagrams in String

Diagrama de solución:

**Longest substring  
without repeating  
characters**

Solución en código:

**Longest substring  
without repeating  
characters**

# Búsqueda Binaria

# Algoritmo de Búsqueda Binaria

**Buscar** un elemento en una  
lista **ordenada** de elementos

$O(\log n)$

1	5	8	11	16	23	40	56	72	99
0	1	2	3	4	5	6	7	8	9

Buscar **23**

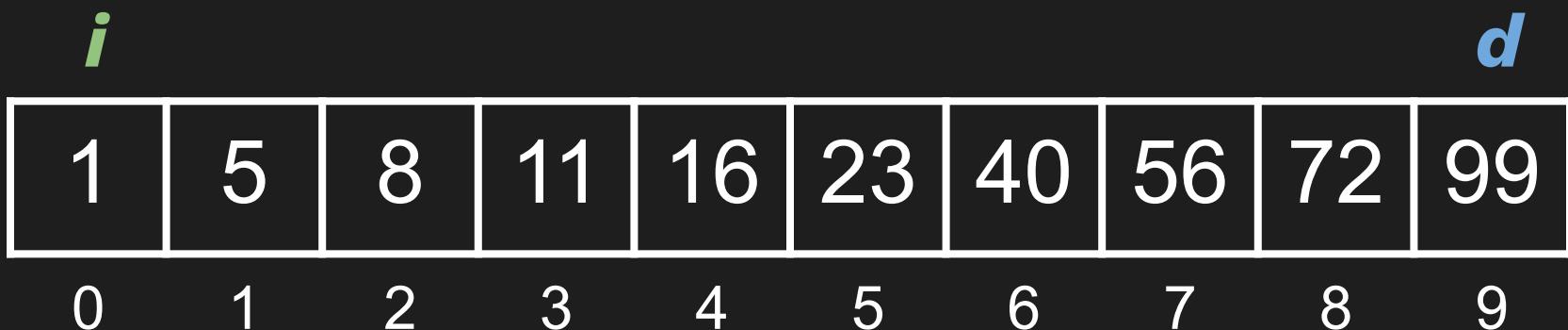


## Buscar **23**



## Buscar **23**

`len(numeros) - 1`



## Buscar **23**

<i>i</i>	<i>m</i>	<i>d</i>
1	5	99
0	1	2
2	3	4
3	4	5
4	6	7
5	7	8
6	8	9
7	9	
8		
9		

# Buscar **23**



## Buscar **23**

<i>i</i>	<i>m</i>	<i>d</i>
1	5	99
0	1	2
2	3	4
3	4	5
4	6	7
5	7	8
6	8	9
7	9	
8		
9		

## Buscar **23**

<i>i</i>	<i>m</i>	<i>d</i>
1	5	99
0	1	8
2	2	9
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	

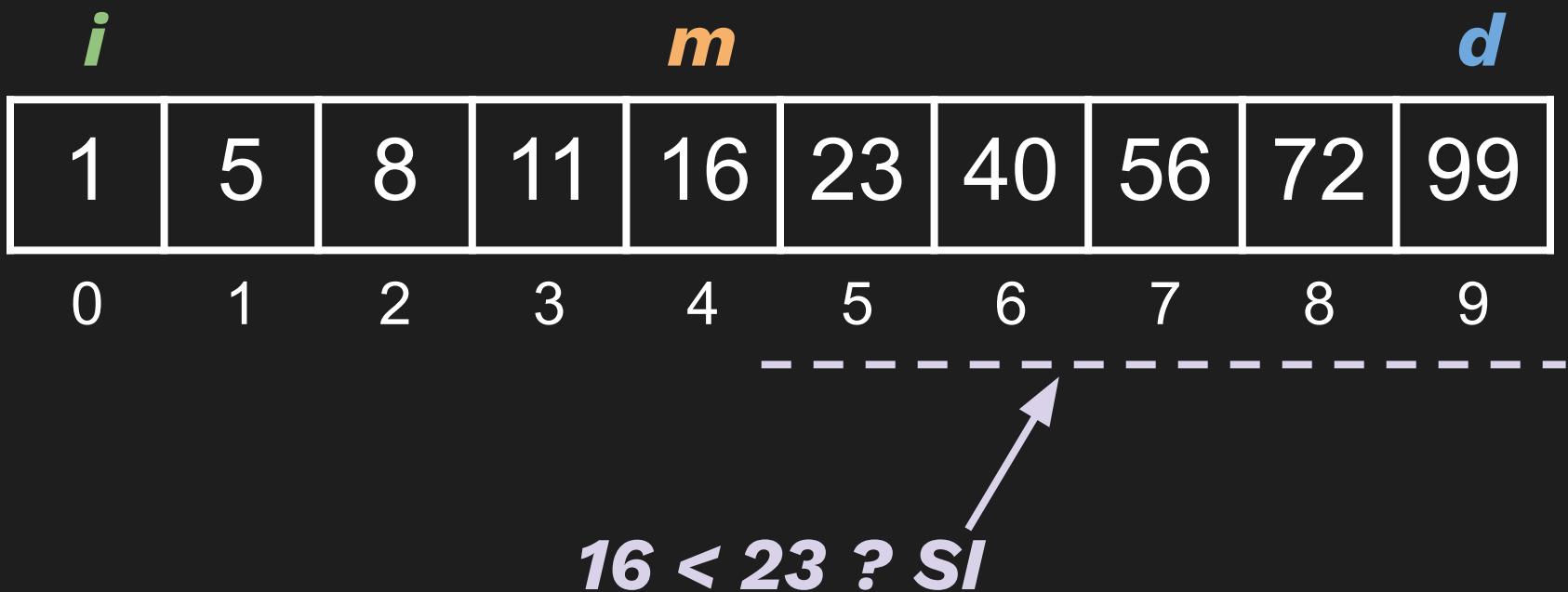
```
izquierda = 0
derecha = len(numeros) - 1
while izquierda <= derecha:
    mitad = izquierda + (derecha - izquierda) // 2
```

# Buscar **23**

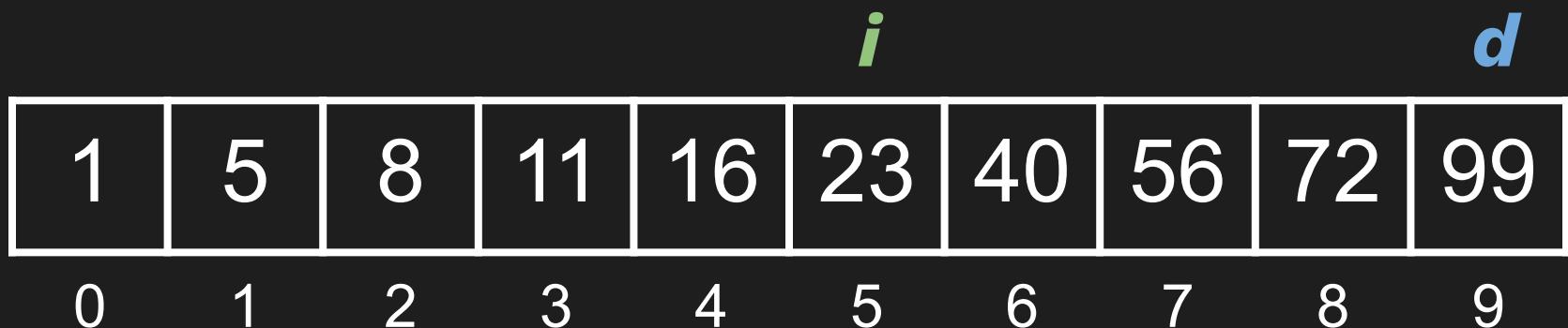
<i>i</i>	<i>m</i>	<i>d</i>
1	5	8
8	11	16
16	23	40
40	56	72
72	99	
0	1	2
3	4	5
6	7	8
9		

**16 < 23 ? SI**

# Buscar **23**



## Buscar **23**

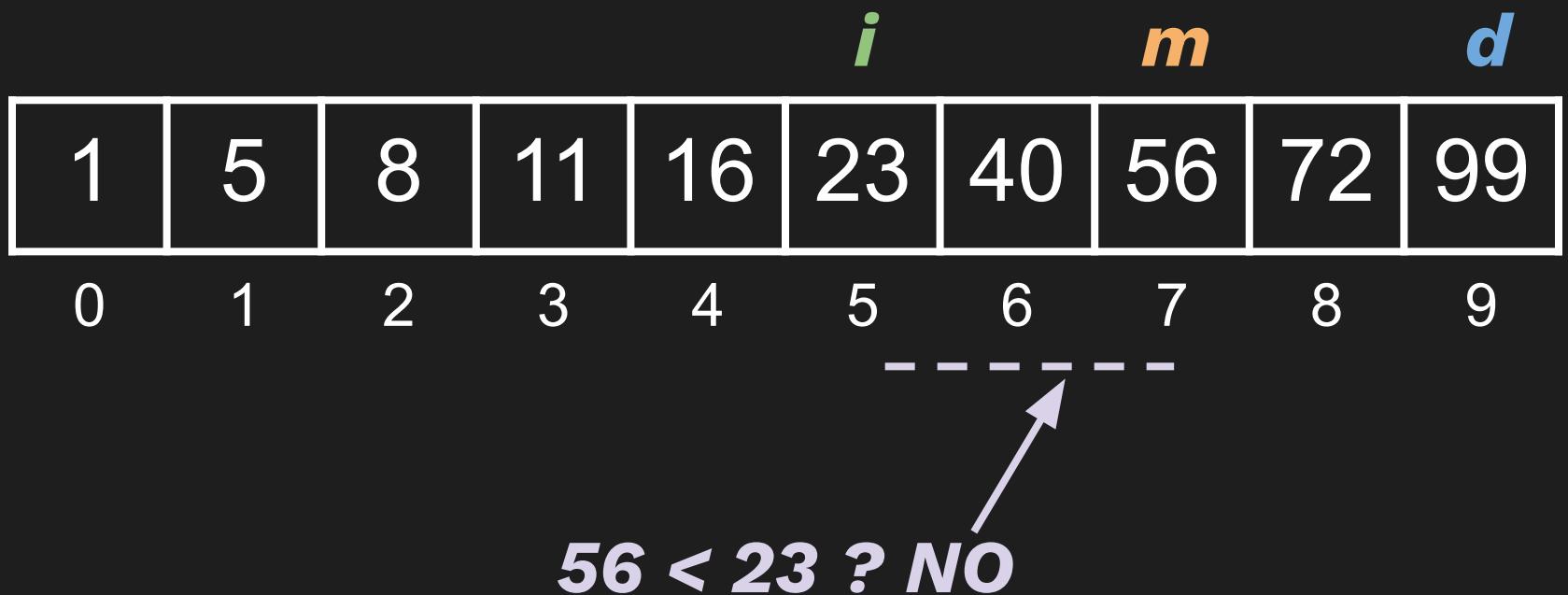


```
if numeros[mitad] < objetivo:  
    izquierda = mitad+1
```

# Buscar **23**

<i>i</i>	<i>m</i>	<i>d</i>
1	5	99
5	23	72
8	40	16
11	56	11
16	72	5
23	1	8
40	99	2
56	23	3
72	56	4
99	72	6
0	0	7
1	1	8
2	2	9

# Buscar **23**

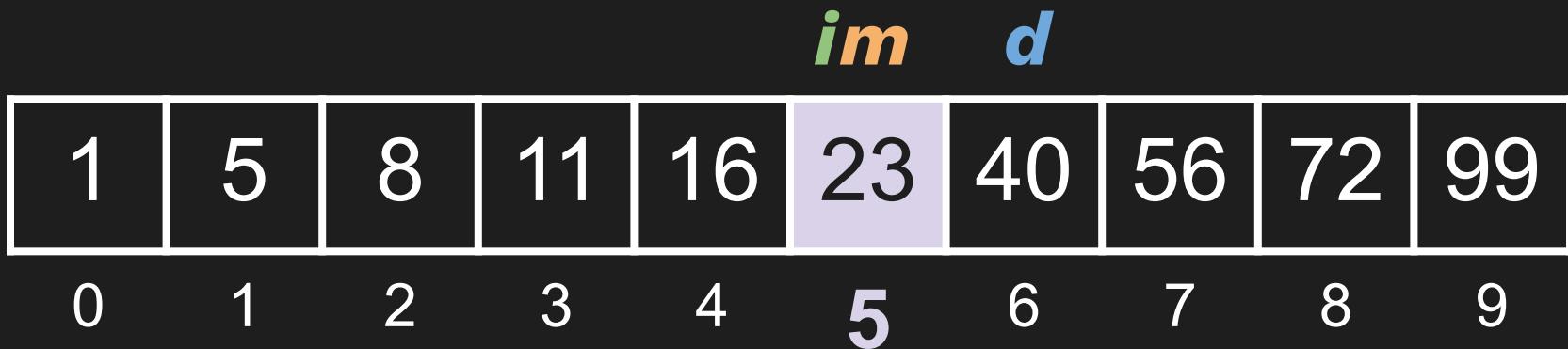


## Buscar **23**

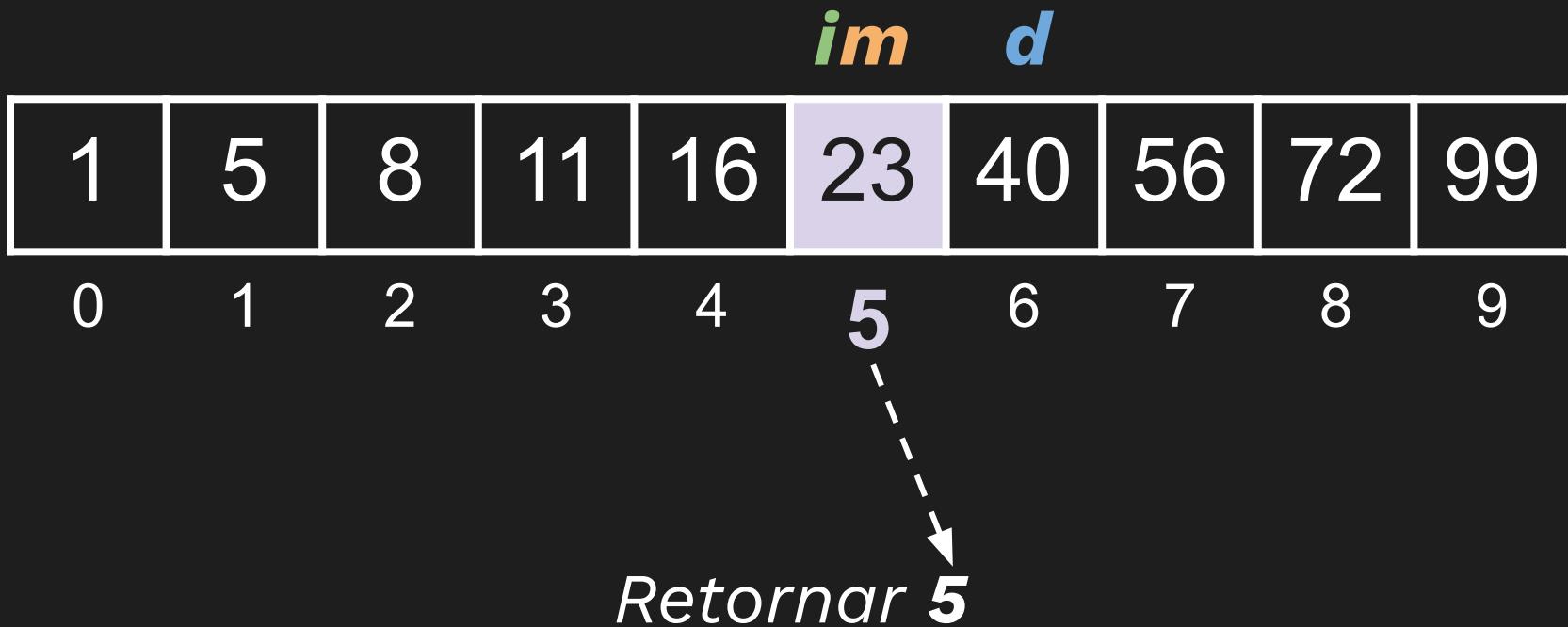


```
    else:  
        |     derecha = mitad-1
```

## Buscar **23**



# Buscar **23**



```
def binarySearch(numeros, objetivo):
    izquierda = 0
    derecha = len(numeros) - 1
    while izquierda <= derecha:
        mitad = izquierda + (derecha - izquierda) // 2
        if numeros[mitad] == objetivo:
            return mitad
        elif numeros[mitad] < objetivo:
            izquierda = mitad+1
        else:
            derecha = mitad-1
    return -1
```

```
int busquedaBinaria(vector<int> &nums, int target) {
    int izquierda = 0;
    int derecha = nums.size() - 1;
    int mitad;
    while (izquierda <= derecha) {
        mitad = izquierda + (derecha - izquierda) / 2;
        if (nums[mitad] == target) {
            return mitad;
        }
        else if (nums[mitad] < target) {
            izquierda = mitad+1;
        } else {
            derecha = mitad-1;
        }
    }
    return -1;
}
```

Análisis del problema:  
**Search in  
Rotated Arrays**



Tienes una lista de enteros ordenada  
de forma ascendente con valores distintos.

La lista (nums) se encuentra posiblemente rotada en un índice pivote desconocido  $k$  ( $1 \leq k < \text{nums.length}$ ) de tal manera que el array resultante es  $[\text{nums}[k], \text{nums}[k+1], \dots, \text{nums}[\text{n}-1], \text{nums}[0], \text{nums}[1], \dots, \text{nums}[k-1]]$ .

Por ejemplo,  $[0,1,2,4,5,6,7]$  podría girar en el índice pivote 3 y convertirse en  $[4,5,6,7,0,1,2]$ .

Dado el array nums después de la posible rotación y un objetivo entero, devuelve el índice del objetivo si está en nums o -1 si no está en nums.

**nums** = [4,5,6,7,0,1,2]  4  
**target** = 0

**nums** = [4,5,6,7,0,1,2]  -1  
**target** = 3

Diagrama de solución:

# **Search in Rotated Arrays**

Solución en código:  
**Search in  
Rotated Arrays**

Análisis del problema:

# **Search 2D Array Matrix**



Escriba un algoritmo eficiente que busque un valor objetivo en una matriz de  $m \times n$  enteros.

Esta matriz tiene las siguientes propiedades:

- Los enteros de cada fila están ordenados de izquierda a derecha.
- El primer entero de cada fila es mayor que el último entero de la fila anterior.

target = 3

1	3	5	7
10	11	16	20
23	30	34	60



Verdadero

Diagrama de solución:

# **Search 2D Array Matrix**

Solución en código:  
**Search 2D  
Array Matrix**

# **End of course**

**- No eliminar -**

Curso Avanzado de  
Algoritmos:  
**Estructuras de  
datos lineales**

@camilalonart

# **Lista Enlazada**

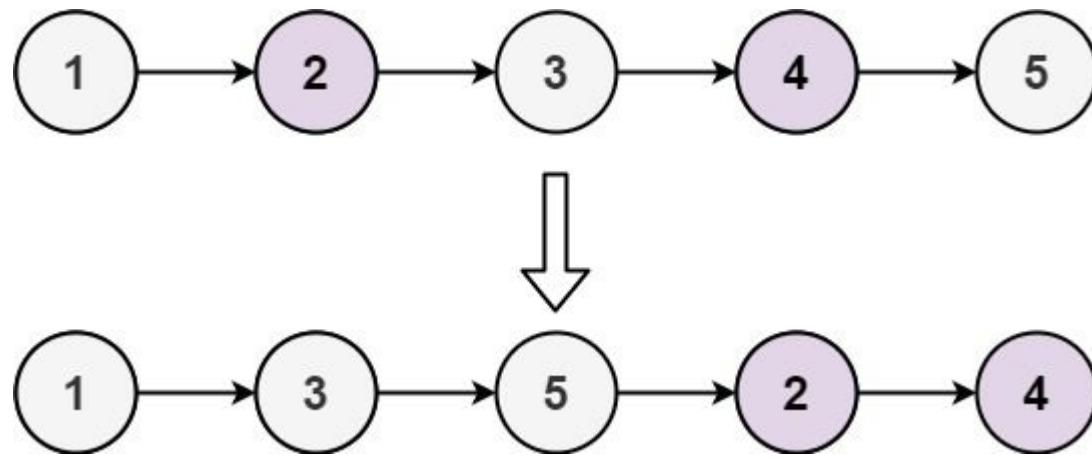
# Invertir una lista enlazada

Análisis del problema:  
**Odd Even  
Linked List**



Dada la cabeza de una lista enlazada simple, agrupa todos los nodos con índices impares seguidos de los nodos con índices pares, y devuelve la lista reordenada.

El primer nodo se considera impar, y el segundo nodo es par, y así sucesivamente.



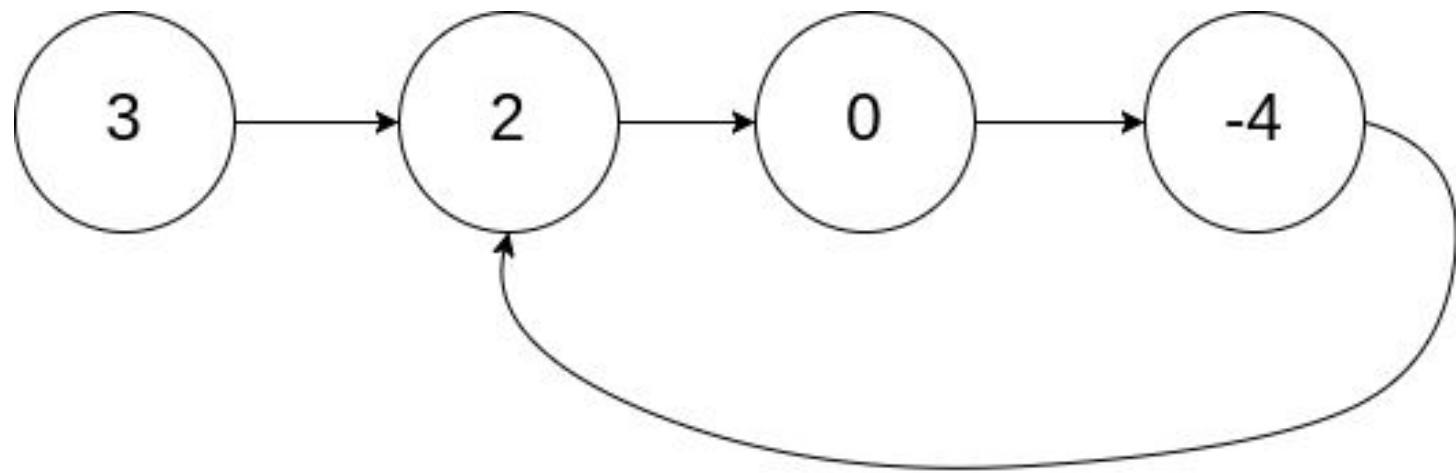
cabeza = [1,2,3,4,5] → [1,3,5,2,4]

Diagrama de solución:  
**Odd Even  
Linked List**

Solución en código:  
**Odd Even  
Linked List**

# Análisis del problema: **Linked List Cycle**

Dada la cabeza de una lista enlazada,  
determinar si la lista enlazada tiene un  
ciclo en ella.



cabeza = [3,2,0,-4] → verdadero

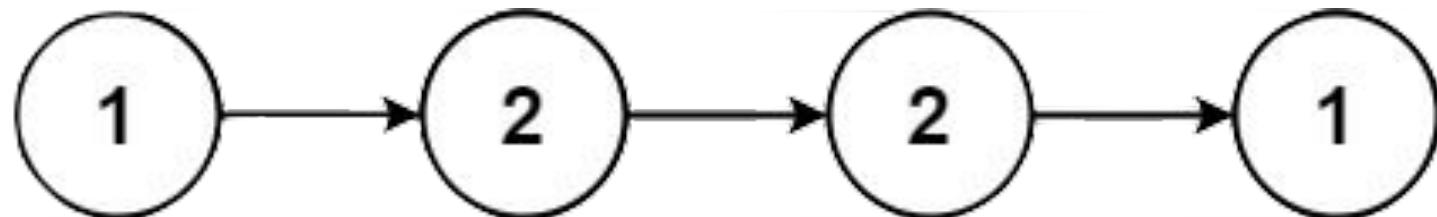
# Diagrama de solución: **Linked List Cycle**

Solución en código:  
**Linked List Cycle**

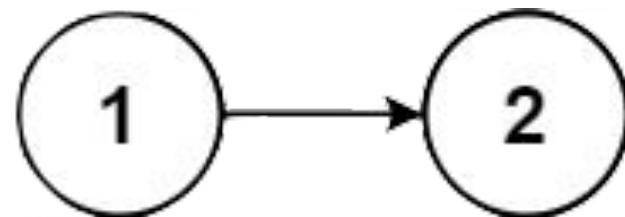
Análisis del problema:  
**Palindrome  
Linked List**



Dada la cabeza de una lista unidireccional,  
devuelve true si es un palíndromo.



cabeza = [1,2,2,1]  Verdadero



cabeza = [1,2]  False

Diagrama de solución:  
**Palindrome  
Linked List**

Solución en código:

# **Palindrome Linked List**

# Análisis del problema: **Reorder List**

Tienes la cabeza de una lista unidireccional.

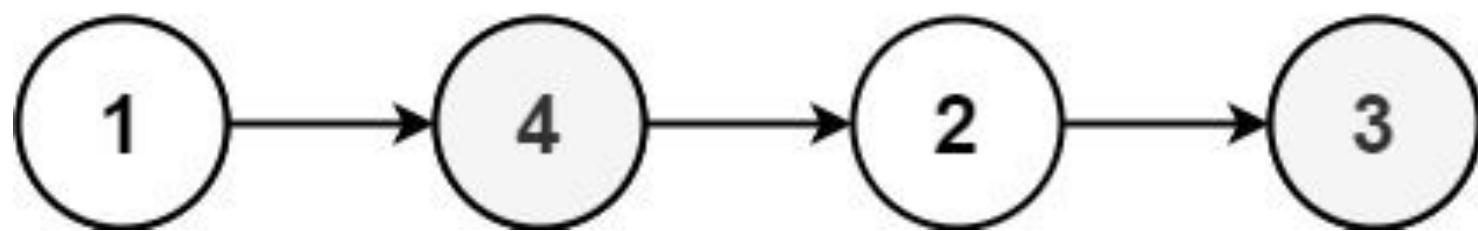
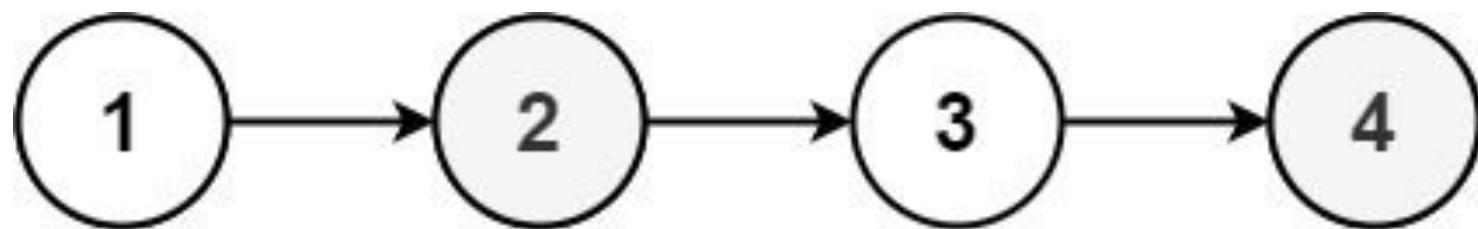
La lista enlazada se puede representar como:

$$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$$

Reordena la lista enlazada para que tenga la siguiente forma:

$$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$$

No puede modificar los valores de los nodos de la lista. Sólo se pueden modificar los propios nodos.



# Diagrama de solución: **Reorder List**

Solución en código:  
**Reorder List**

# Reto **LRU Cache**



Diseñar una estructura de datos  
que siga las restricciones de una caché  
de uso menos reciente (LRU).



Inicializa la clase caché LRU  
con una capacidad de tamaño positivo.

`int get(int key)` Devuelve el valor de la llave si esta existe, en caso contrario devuelve -1.

`void put(int key, int value)` Actualiza el valor de la llave si esta existe. En caso contrario, añade el par llave-valor al caché. Si el número de llaves supera la capacidad del caché, elimina la pareja llave-valor menos utilizada recientemente.

# Pilas y Colas

# Análisis del problema: **Paréntesis Válido**

Dada una cadena s que sólo contiene los caracteres '(', ')' y letras, determinar si la cadena de entrada es válida.

(( a )( b ))  Verdadero

( hola )  Verdadero

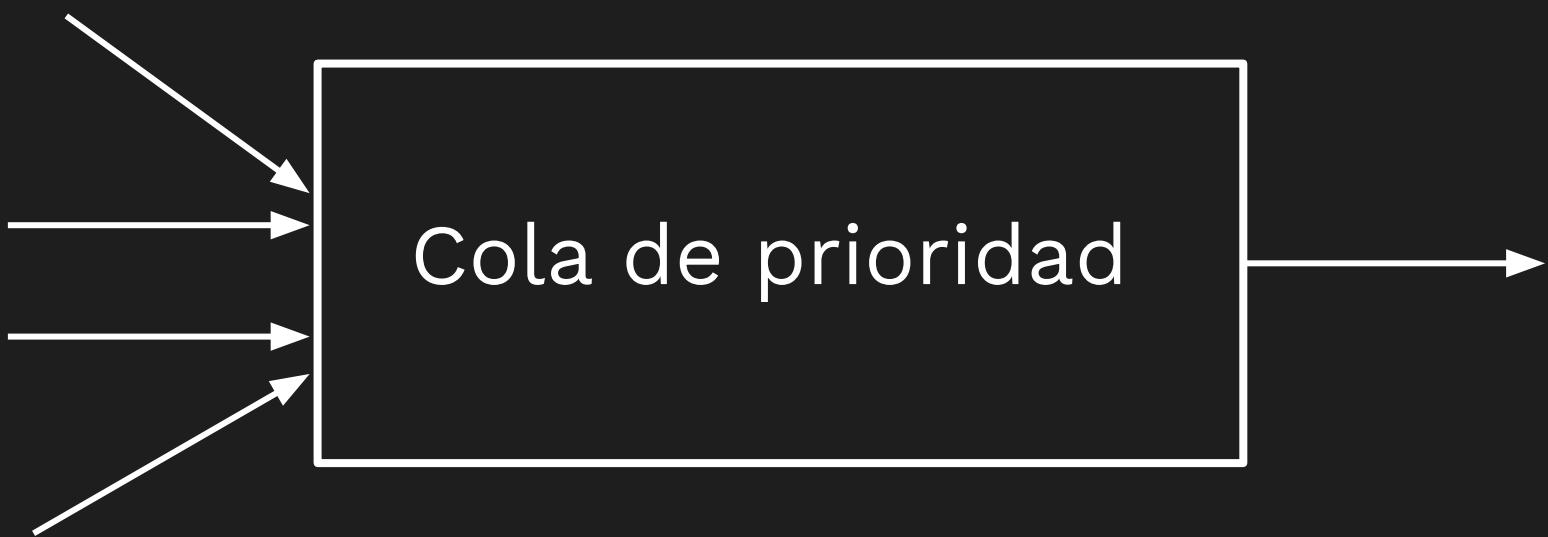
( x ( )  Falso

) (  Falso

# Diagrama de solución: **Paréntesis Válido**

Solución en código:  
**Paréntesis Válido**

# Colas de prioridad



insertar -  $O(\log n)$

remover -  $O(\log n)$

peek -  $O(\log n)$

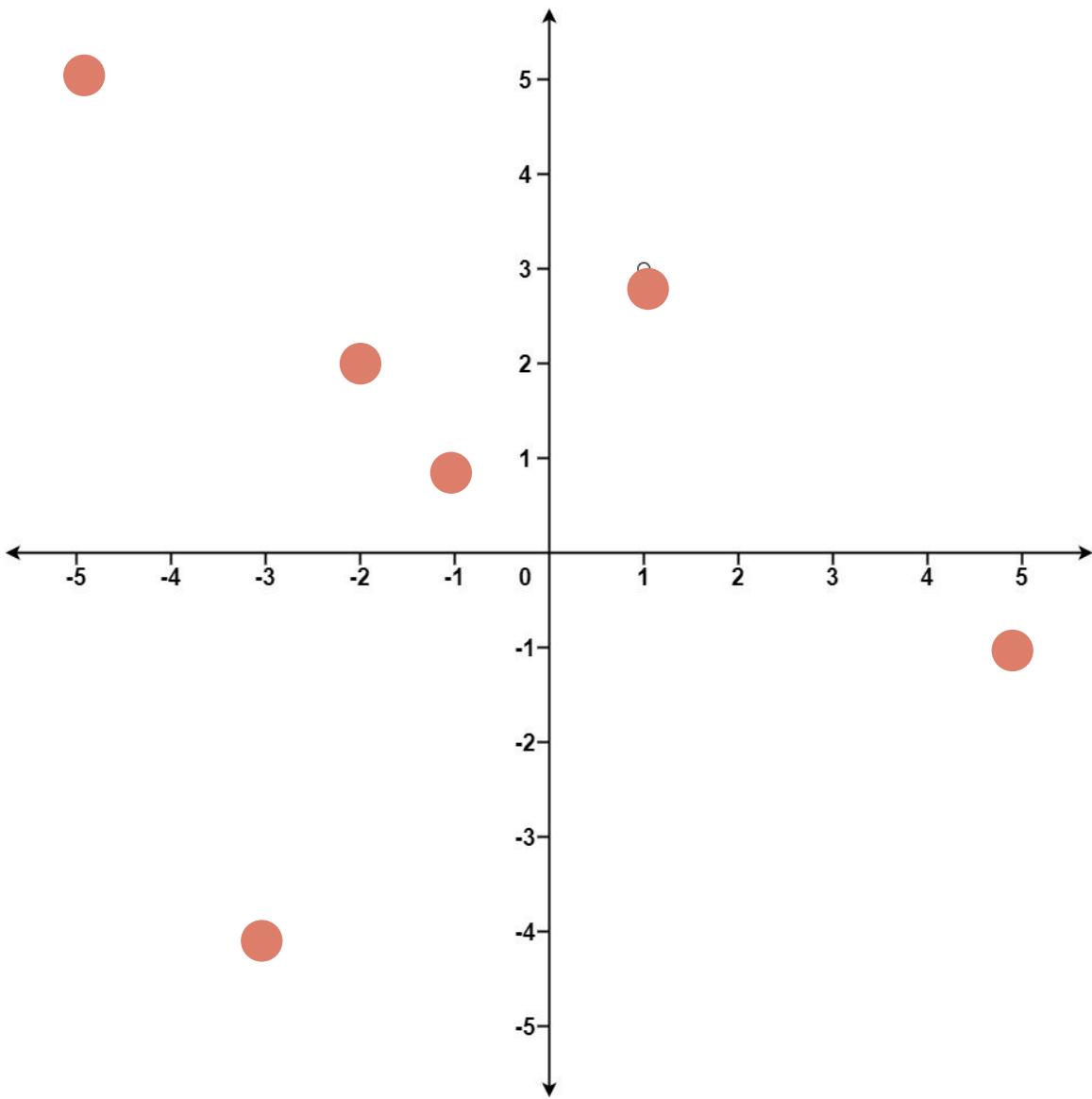
construir cola -  $O(n)$

Análisis del problema:  
**K Closest Points  
to Origin**

Dada una matriz de puntos donde  $\text{puntos}[i] = [x_i, y_i]$  representa un punto del plano X-Y y un número entero  $k$ , devuelve los  $k$  puntos más cercanos al origen  $(0, 0)$ .

La distancia entre dos puntos del plano X-Y es la distancia euclidiana.

Puede devolver la respuesta en cualquier orden. Se garantiza que la respuesta es única (excepto por el orden en que se encuentre).



$[[1,3],[-2,7],[3,4],[-10,1]]$    $[[1,3],[3,4]]$

$k = 2$

Diagrama de solución:  
**K Closest Points  
to Origin**

Solución en código:  
**K Closest Points  
to Origin**

Análisis del problema:  
**Reorganize String**

Dada la string s, reordenar los caracteres de s para que dos caracteres adyacentes cualesquiera no sean iguales.

Retorna el nuevo orden de s y si no es posible reordenar, retorna " "

`s = "aacab"` → `"acaba"`

`s = "aaab"` → `" "`

Diagrama de solución:  
**Reorganize String**

Solución en código:  
**Reorganize String**

# **End of course**

**- No eliminar -**

Curso Avanzado de

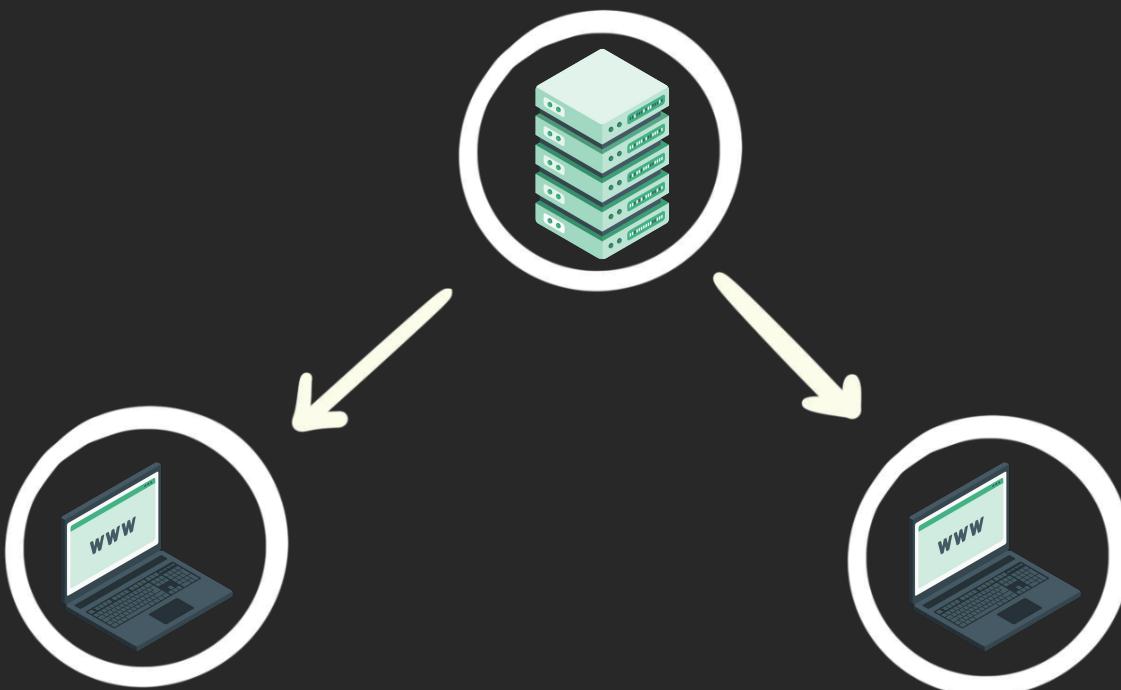
# **Algoritmos Recursivos**

@camilalonart

¿Qué es  
recursión?

# Aplicaciones reales de **grafos y árboles**

# Redes de computación



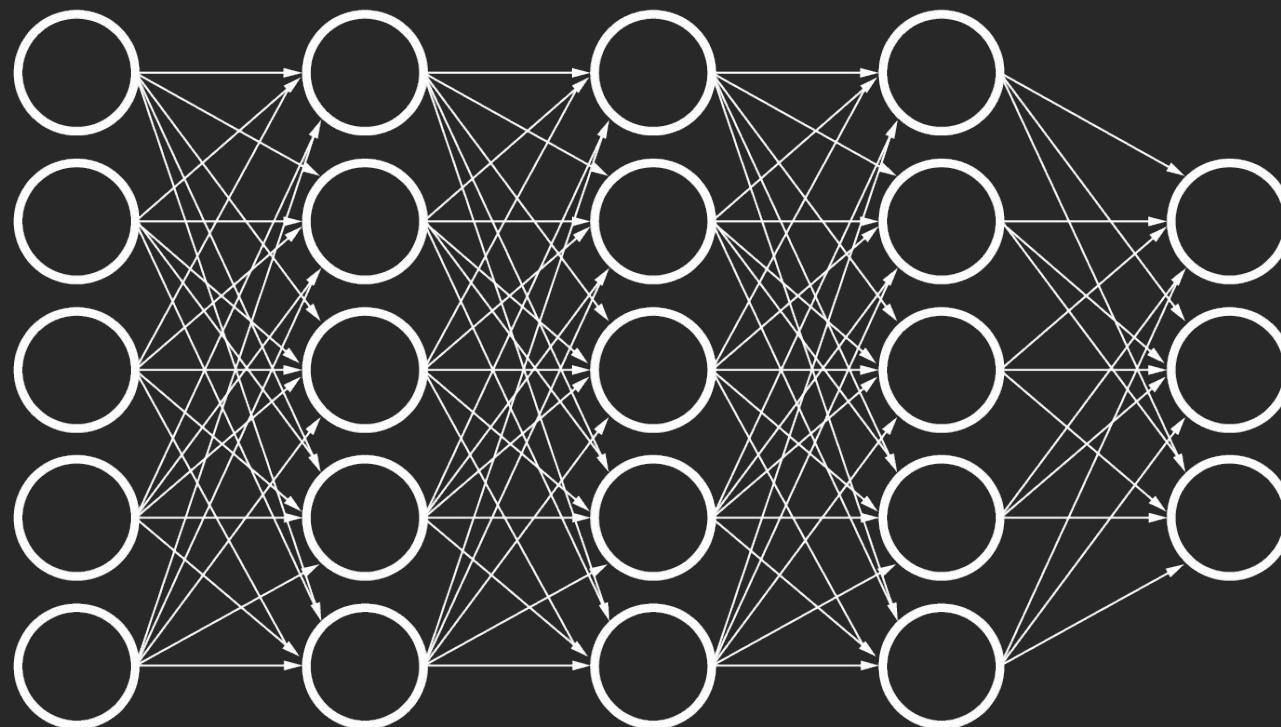
# Infrastructura



# Redes sociales



# Inteligencia Artificial



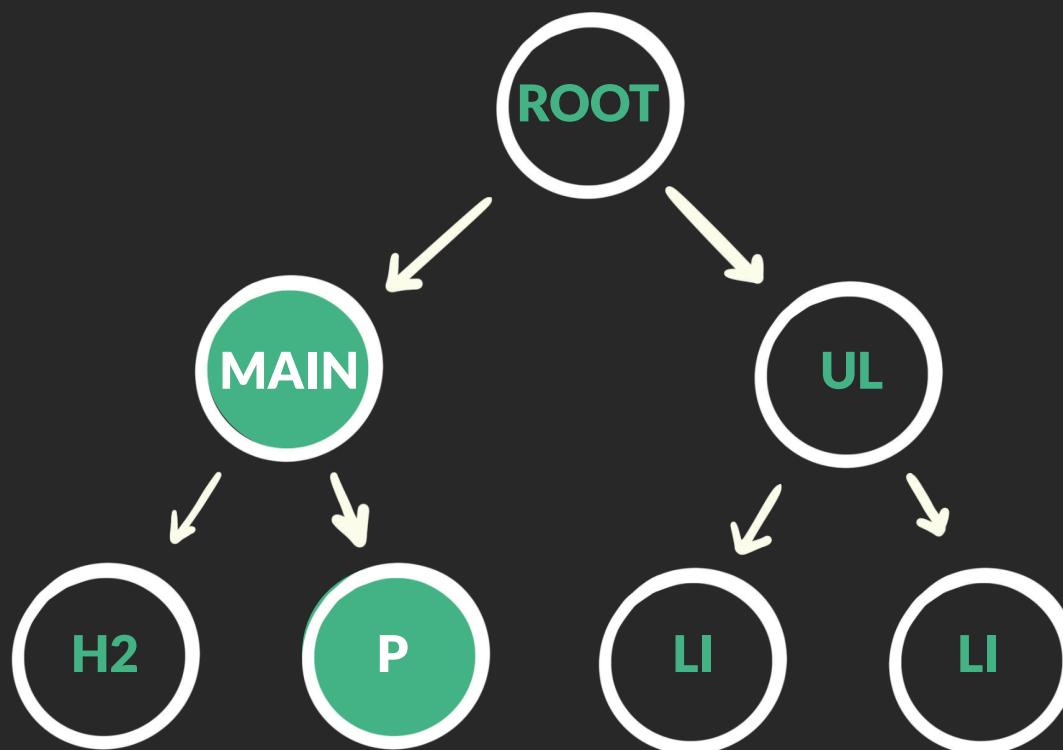
# Sistema de recomendaciones



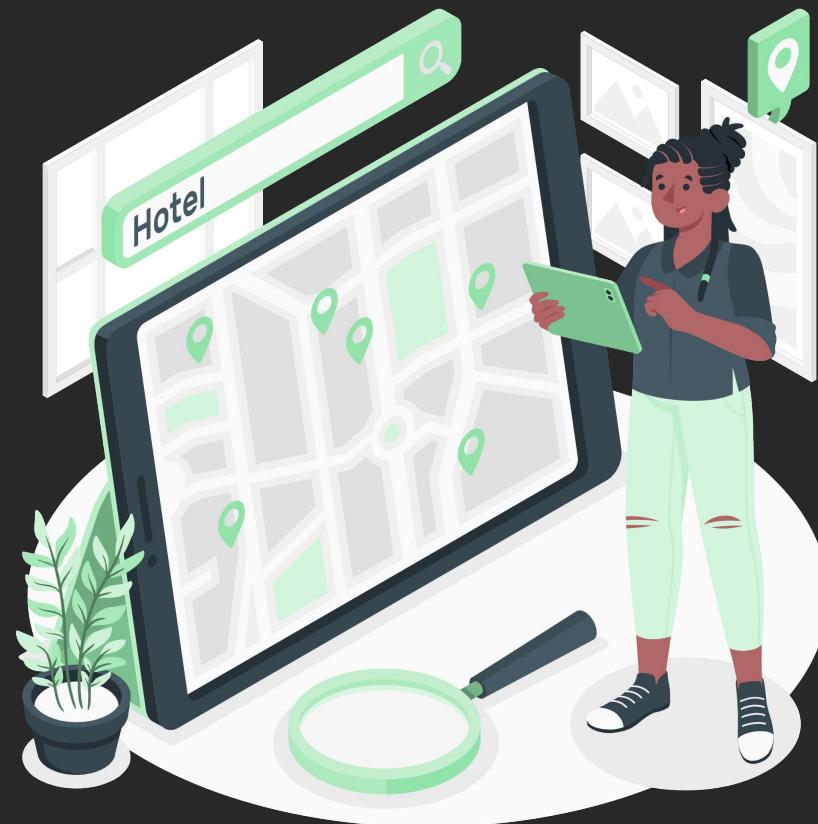
# DOM / Virtual DOM



# DOM / Virtual DOM



# Navegación



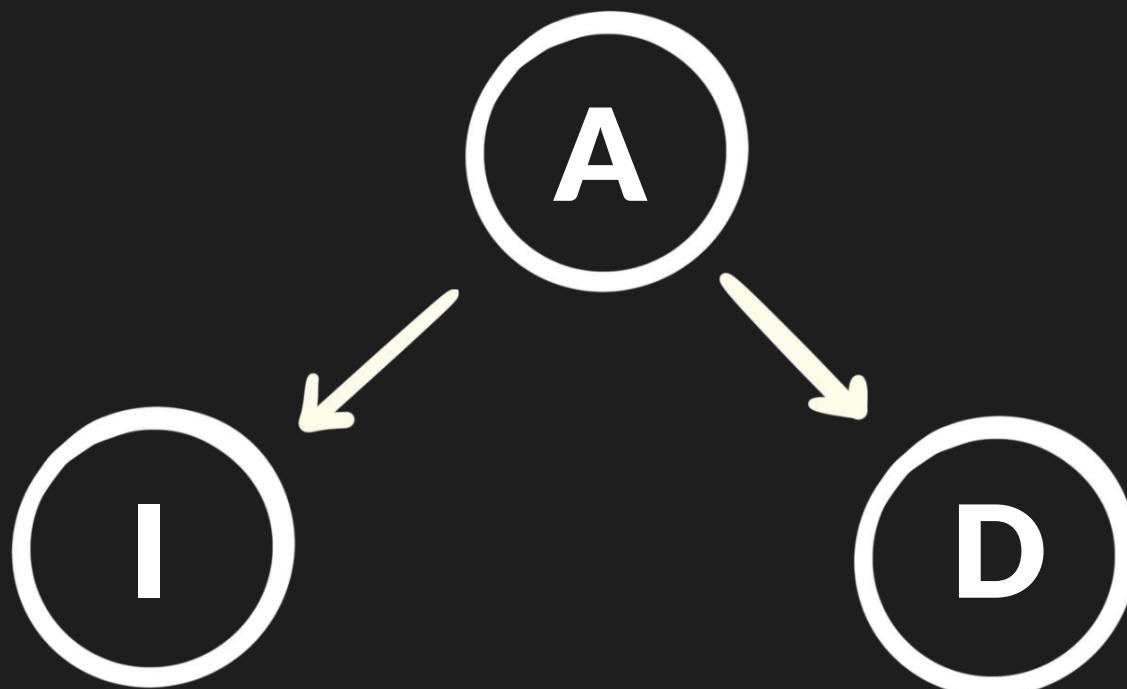
# Análisis del algoritmo: **Búsqueda en Profundidad (DFS)**

Diagrama de solución:  
**Búsqueda en  
Profundidad (DFS)**

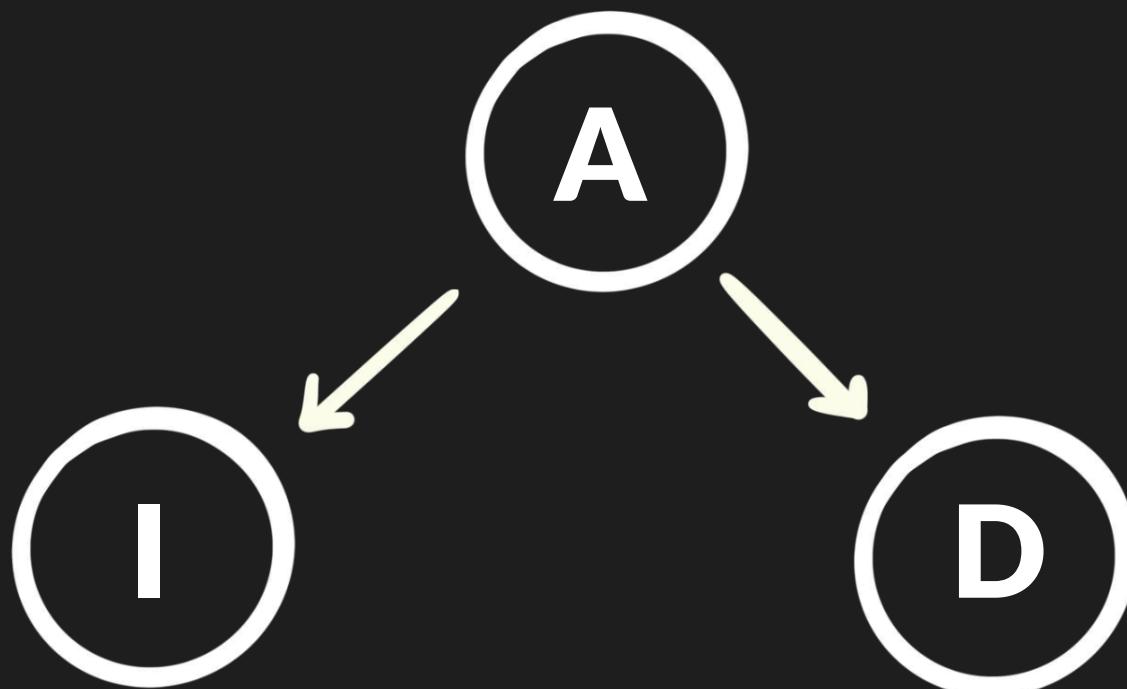
Solución en código:  
**Búsqueda en  
Profundidad (DFS)**

# Recorridos y Profundidad de un Árbol

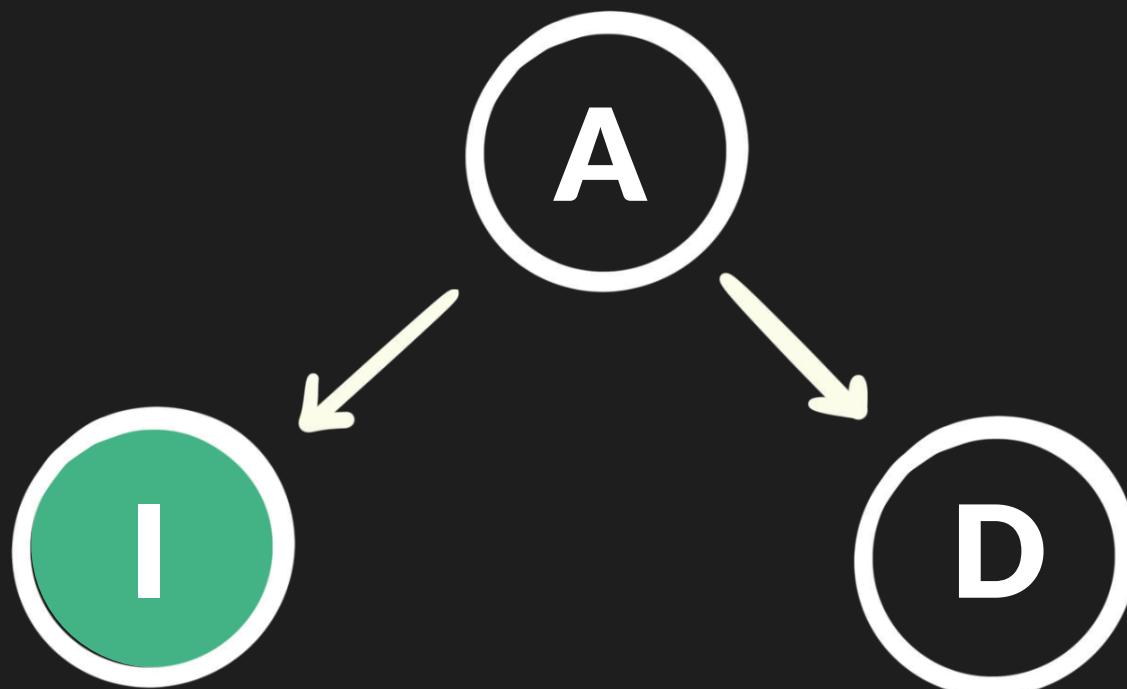
# Recorridos



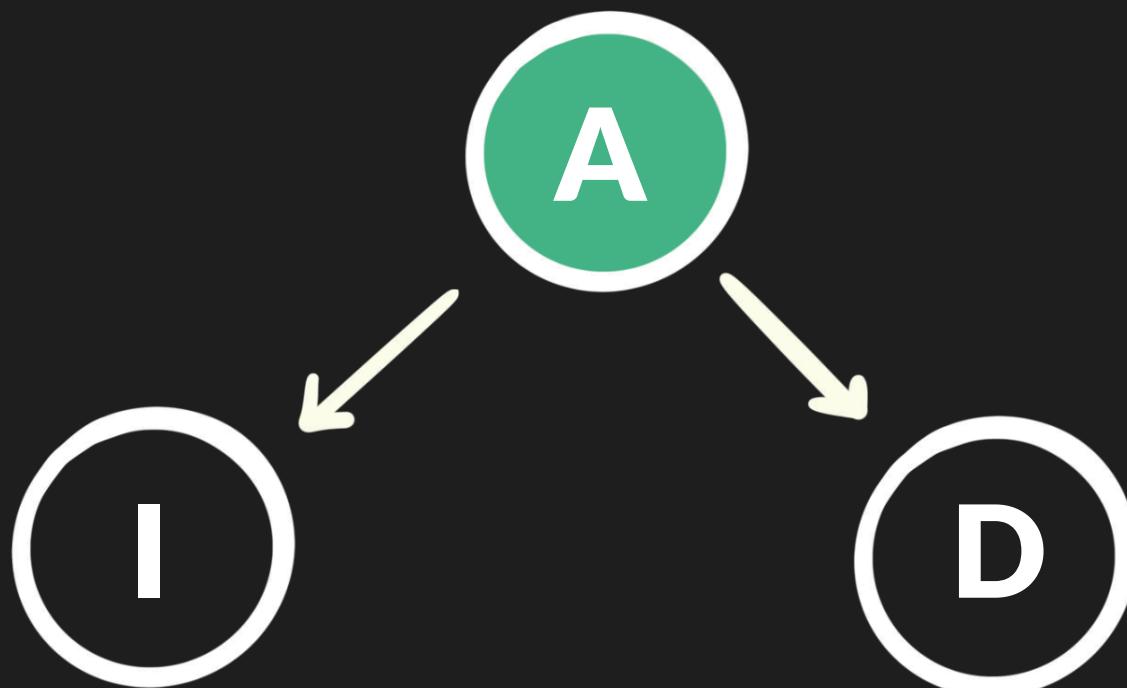
# Recorridos: inorder



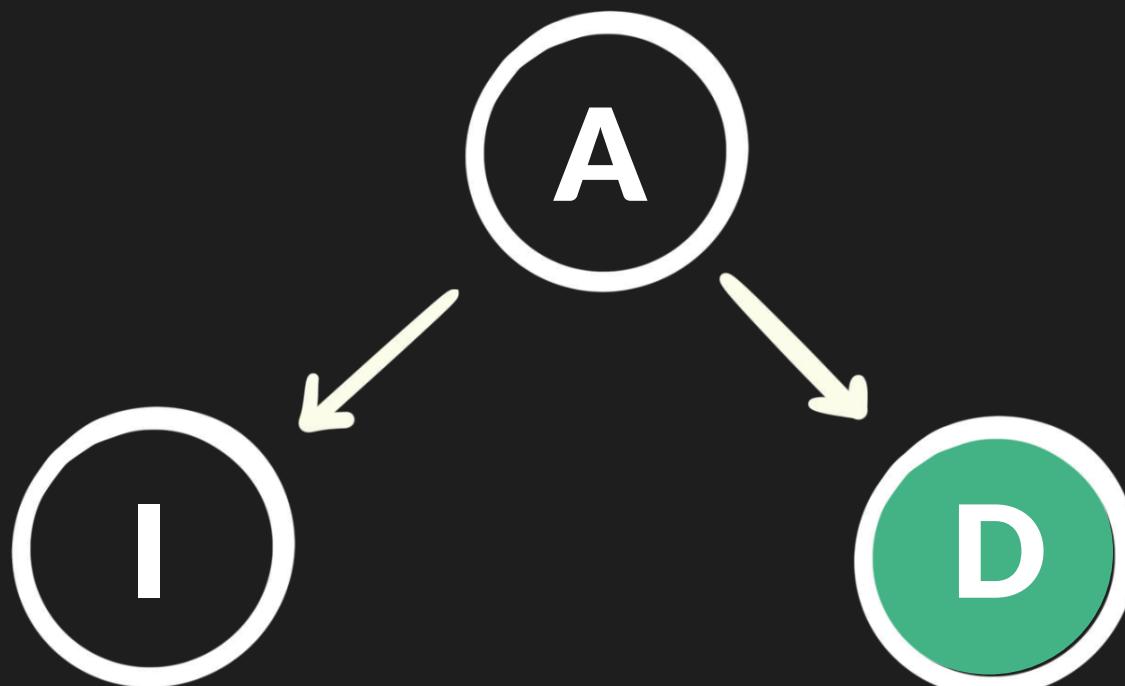
# Recorridos: inorder



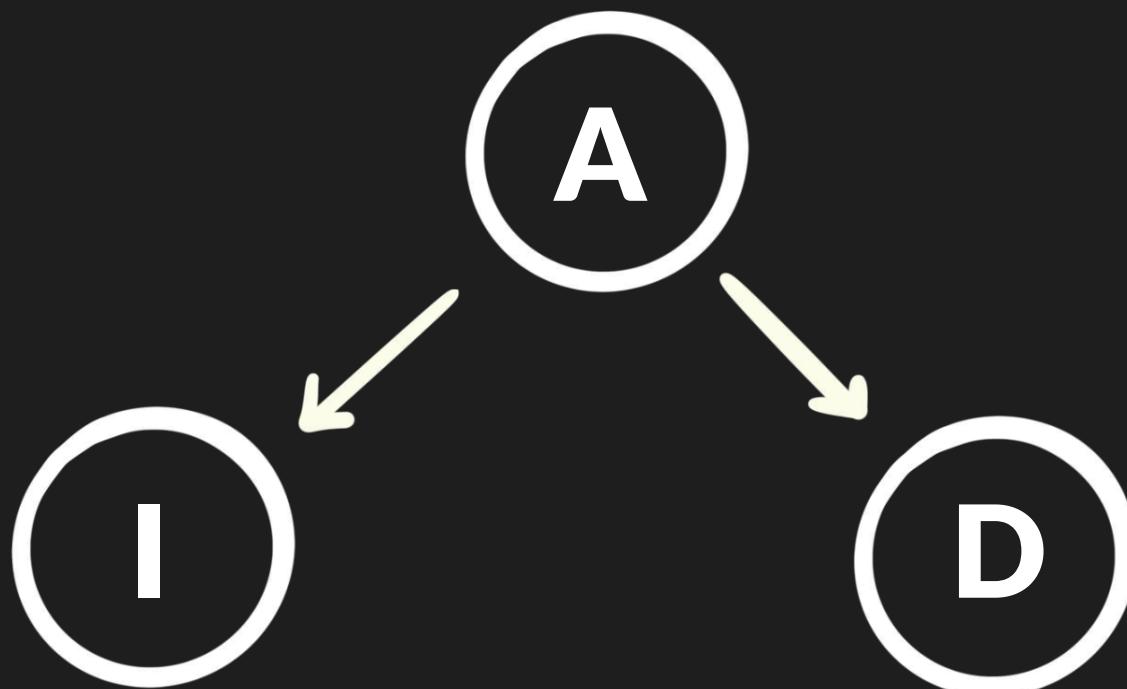
# Recorridos: inorder



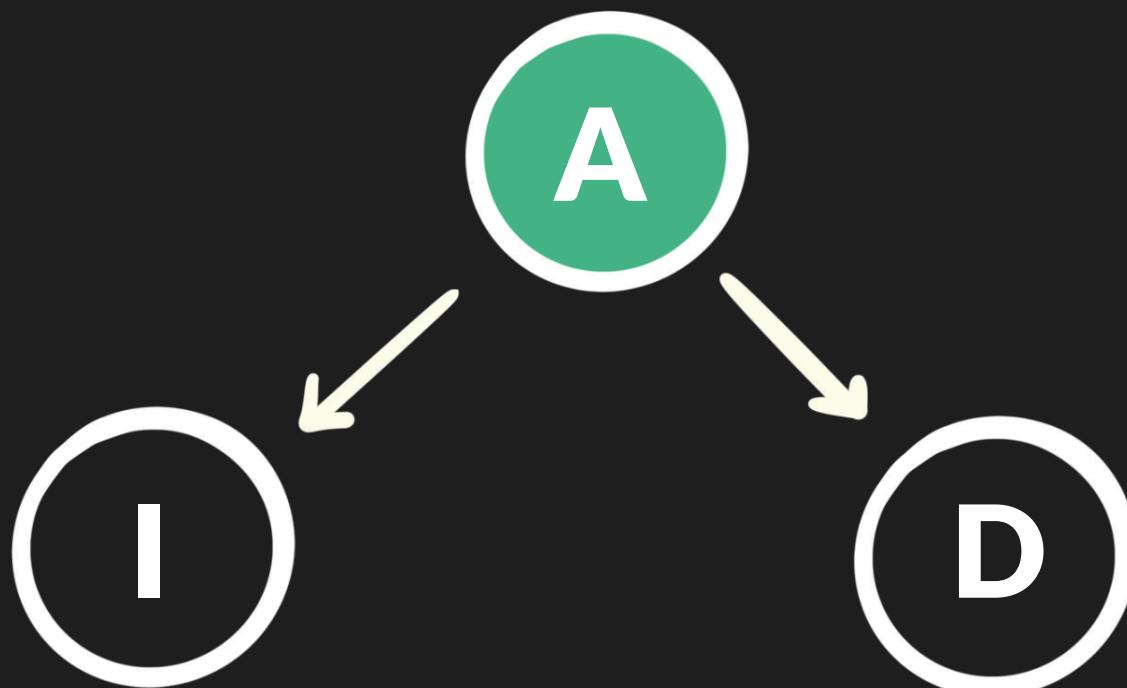
# Recorridos: inorder



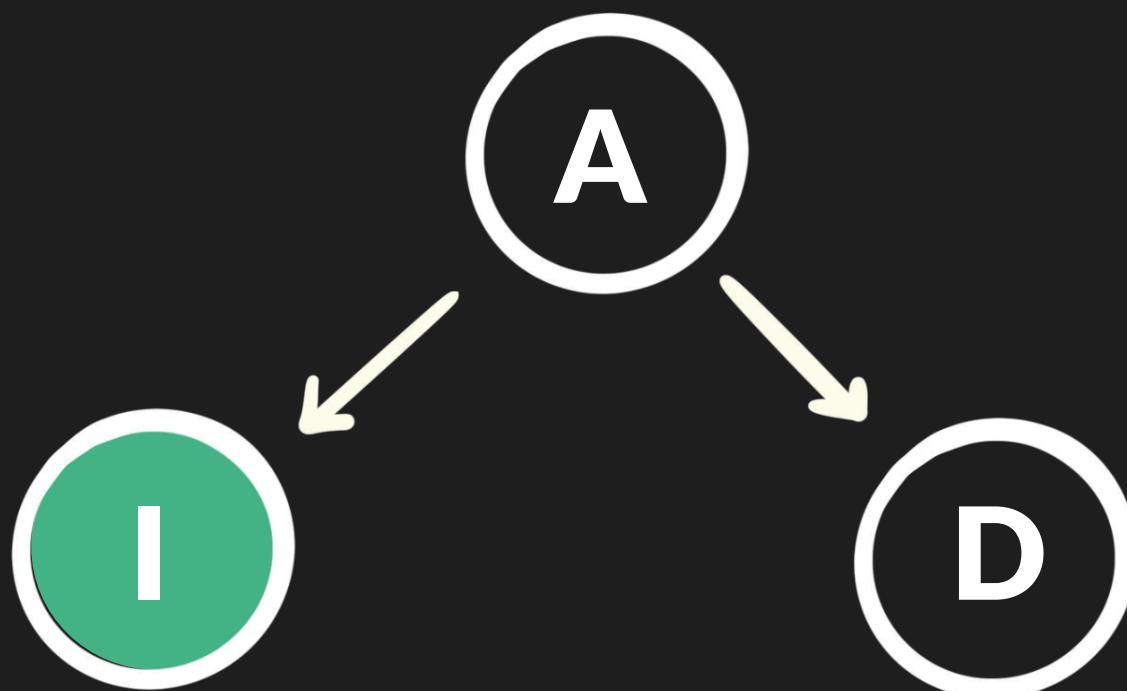
# Recorridos: preorder



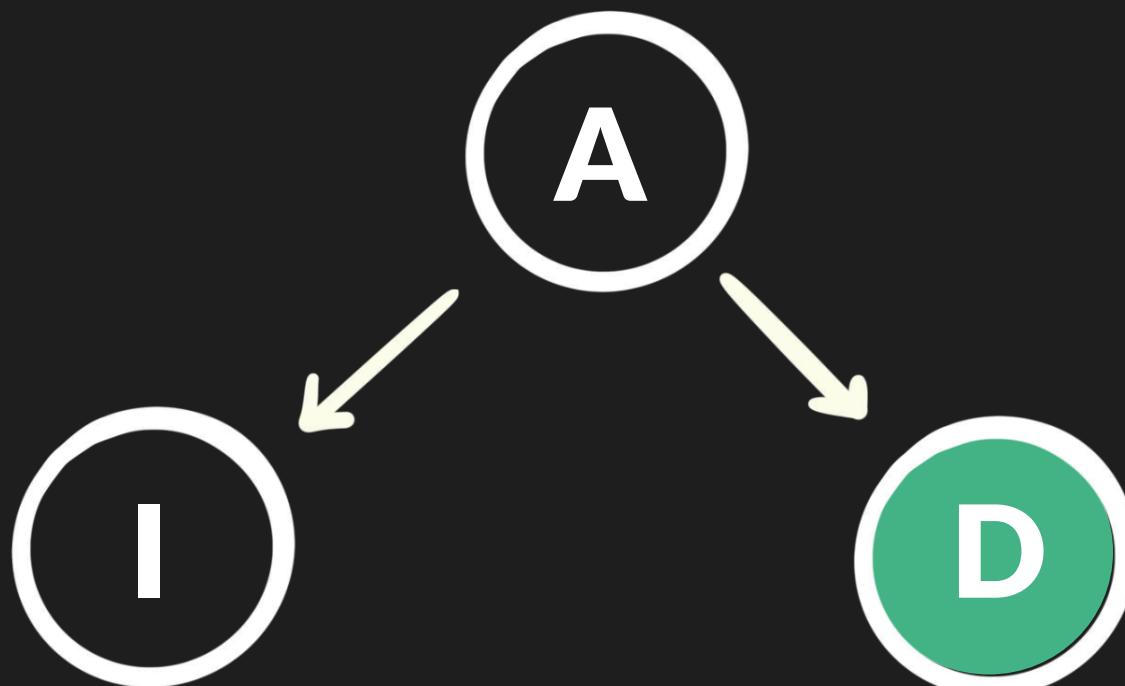
# Recorridos: preorder



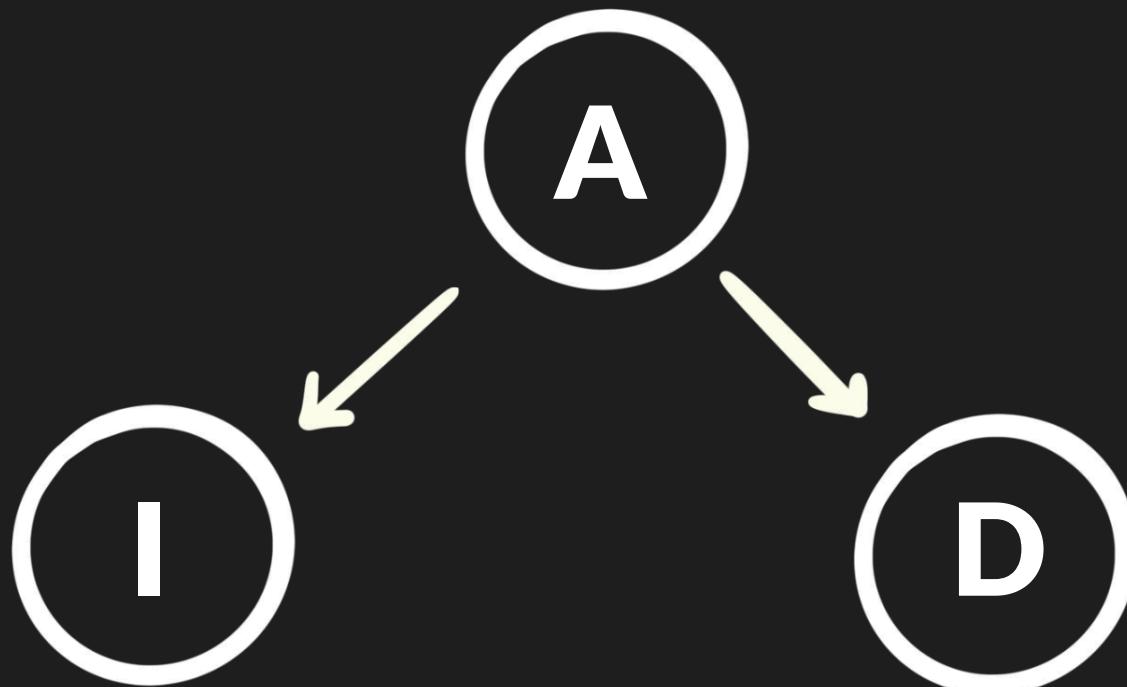
# Recorridos: preorder



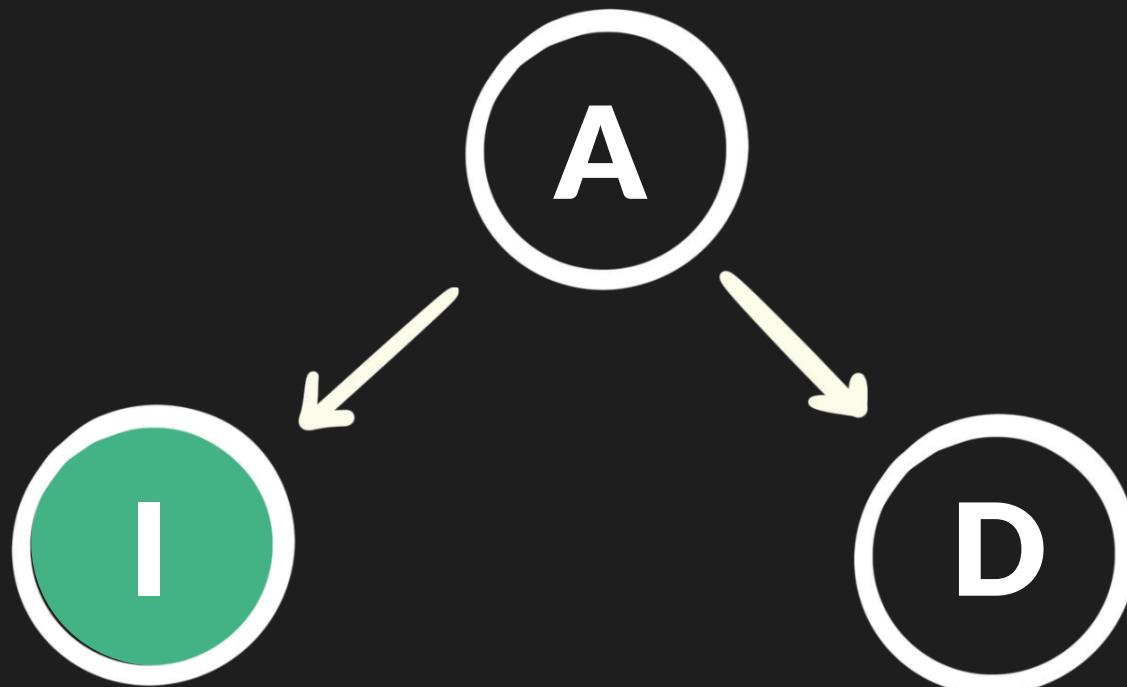
# Recorridos: preorder



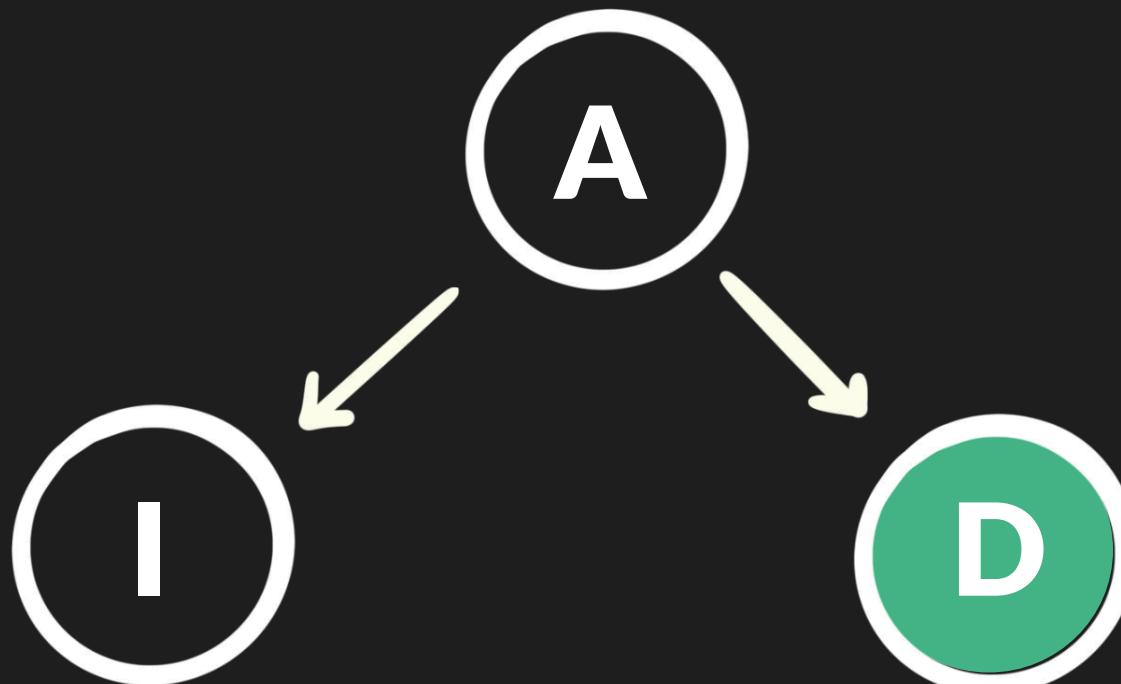
# Recorridos: postorder



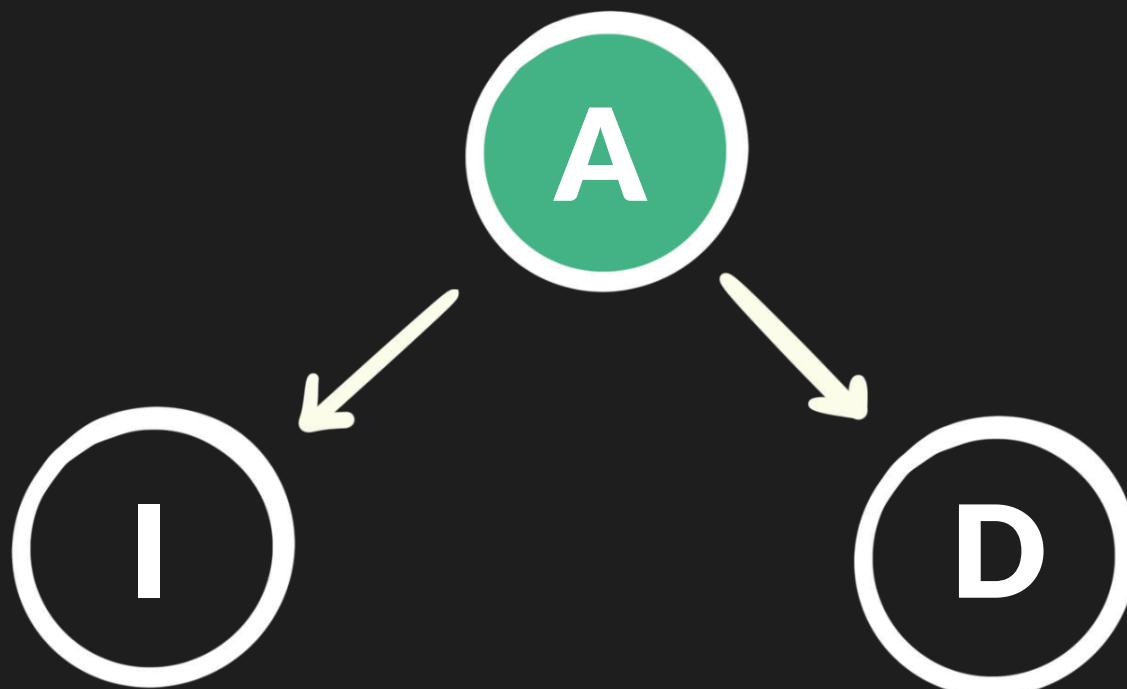
# Recorridos: postorder



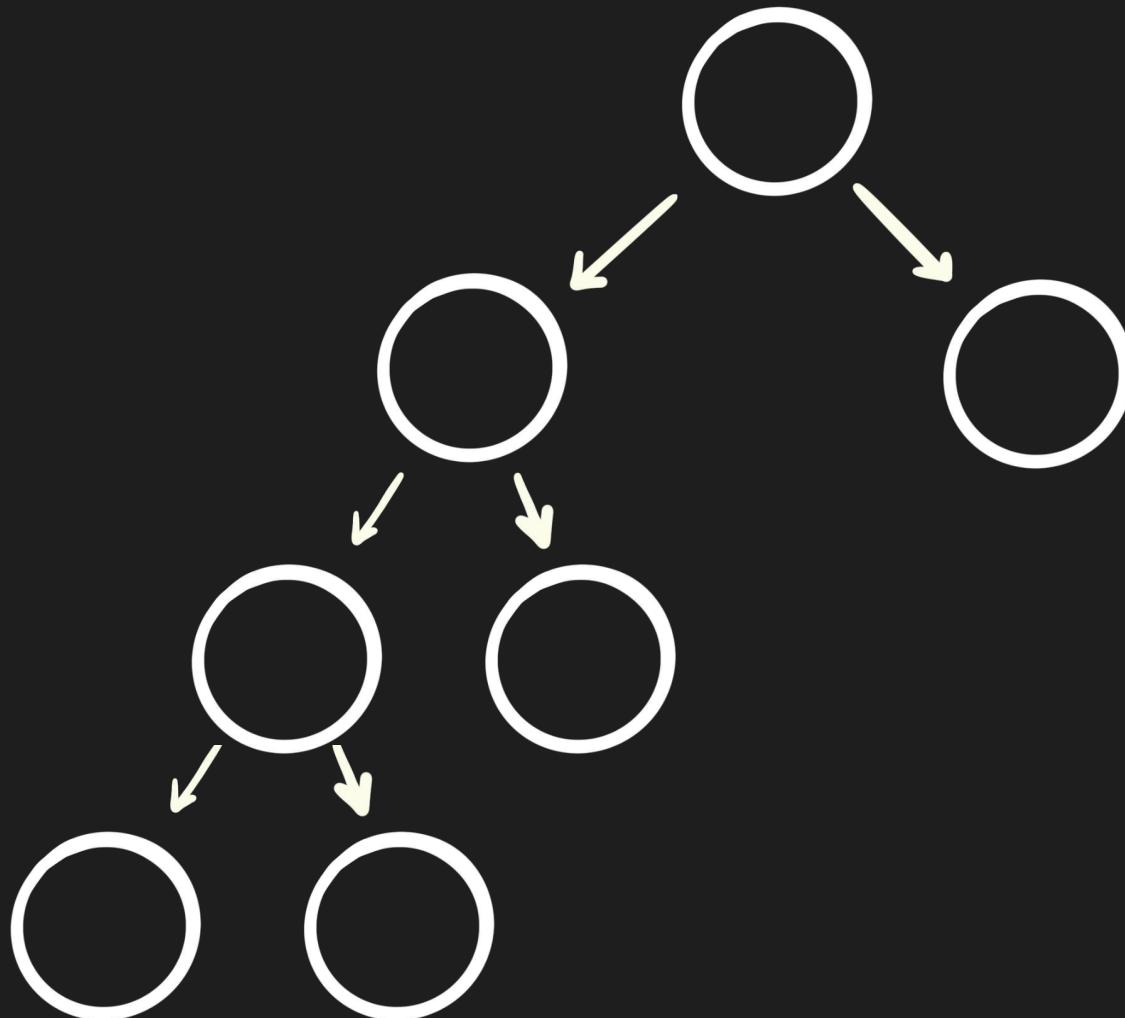
# Recorridos: postorder



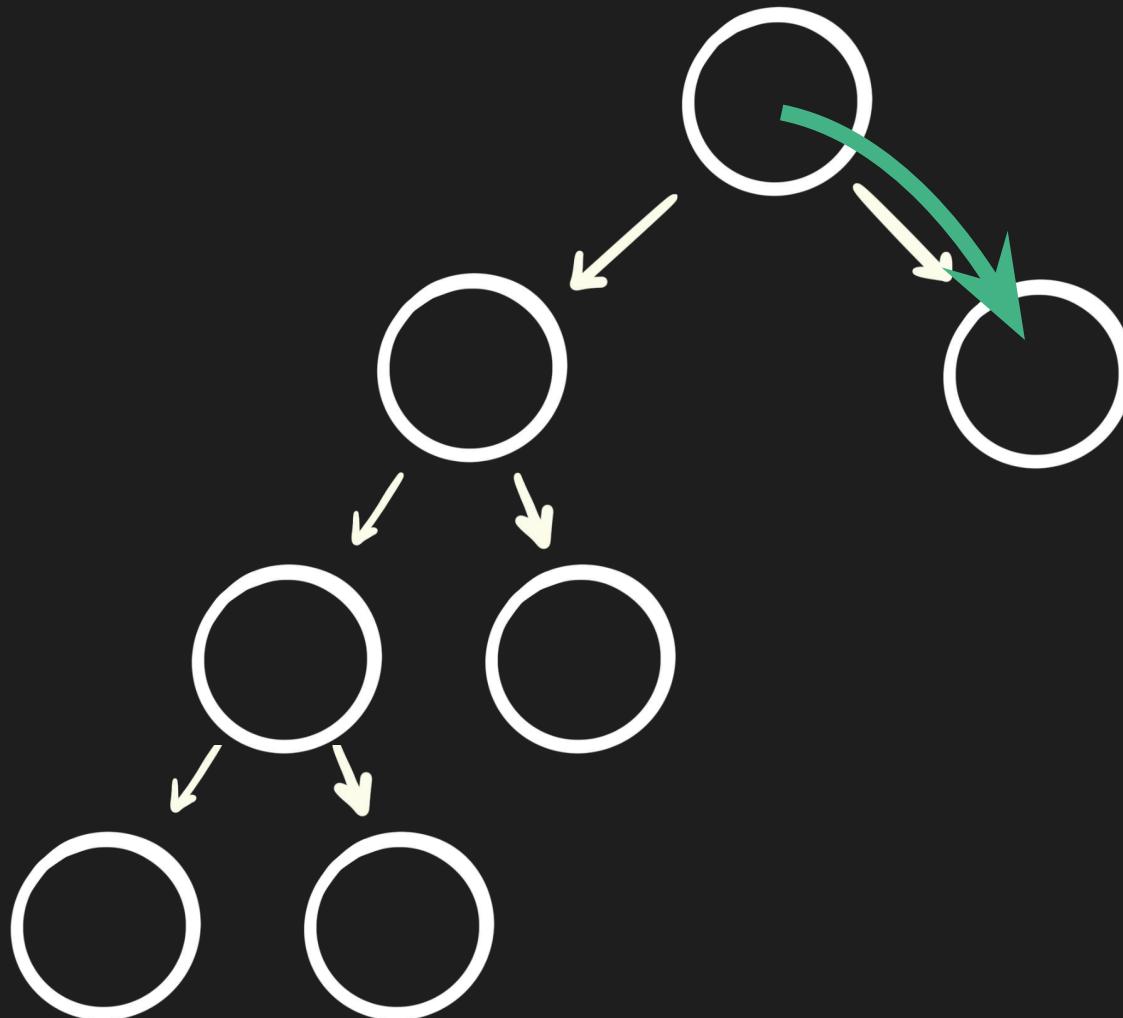
# Recorridos: postorder



# Profundidad

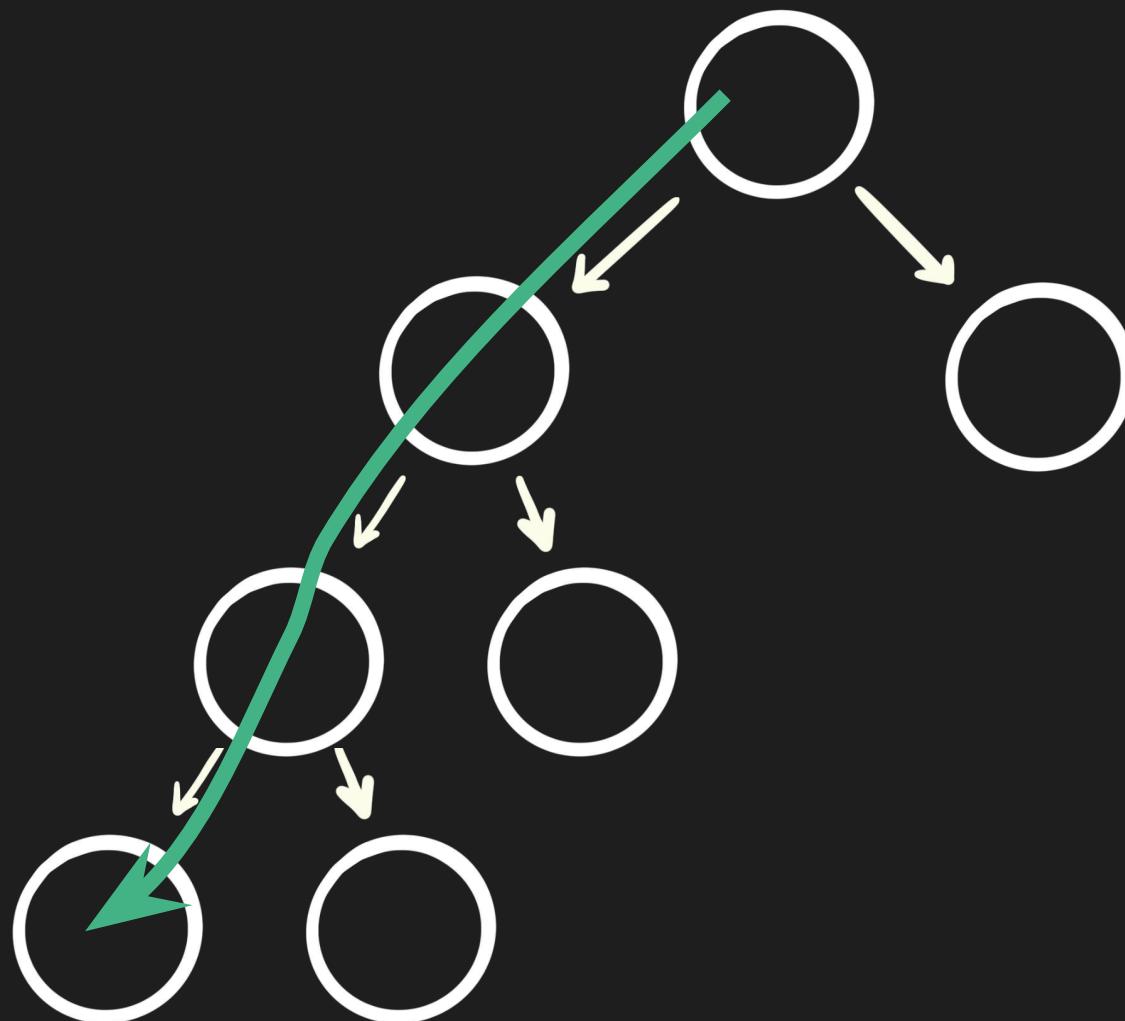


# Profundidad mínima



2

# Profundidad máxima



4

# Profundidad mínima

```
def profundidad(raiz):
    if not raiz: return 0
    izquierda = profundidad(raiz.izquierda)
    derecha = profundidad(raiz.derecha)
    if not izquierda:
        return derecha + 1
    elif not derecha:
        return izquierda +1
    else:
        return min(izquierda,derecha)+1
```

# Profundidad máxima

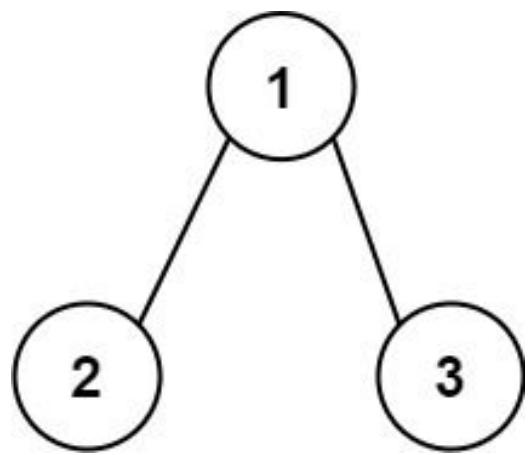
```
def profundidad(raiz):
    if not raiz: return 0
    izquierda = profundidad(raiz.izquierda)
    derecha = profundidad(raiz.derecha)
    if not izquierda:
        return derecha + 1
    elif not derecha:
        return izquierda +1
    else:
        return max(izquierda,derecha)+1
```

Análisis del problema:

# **Sum Root to Leaf Numbers**



Tienes la raíz de un árbol binario que contiene sólo dígitos del 0 al 9. Devuelve la suma total de todos los números formados por los caminos entre la raíz y los nodos que son hojas.



**25**

Diagrama de solución:  
**Sum Root to  
Leaf Numbers**

Solución en código:

**Sum Root to  
Leaf Numbers**

# Análisis del problema: **Número de Islas**

Tienes una matriz binaria  $m \times n$  que representa un mapa de "1" (tierra) y "0" (agua), retorna el número de islas.

Una isla está rodeada de agua y se forma conectando tierras adyacentes.

Puede suponer que los cuatro bordes de la cuadrícula están rodeados de agua.

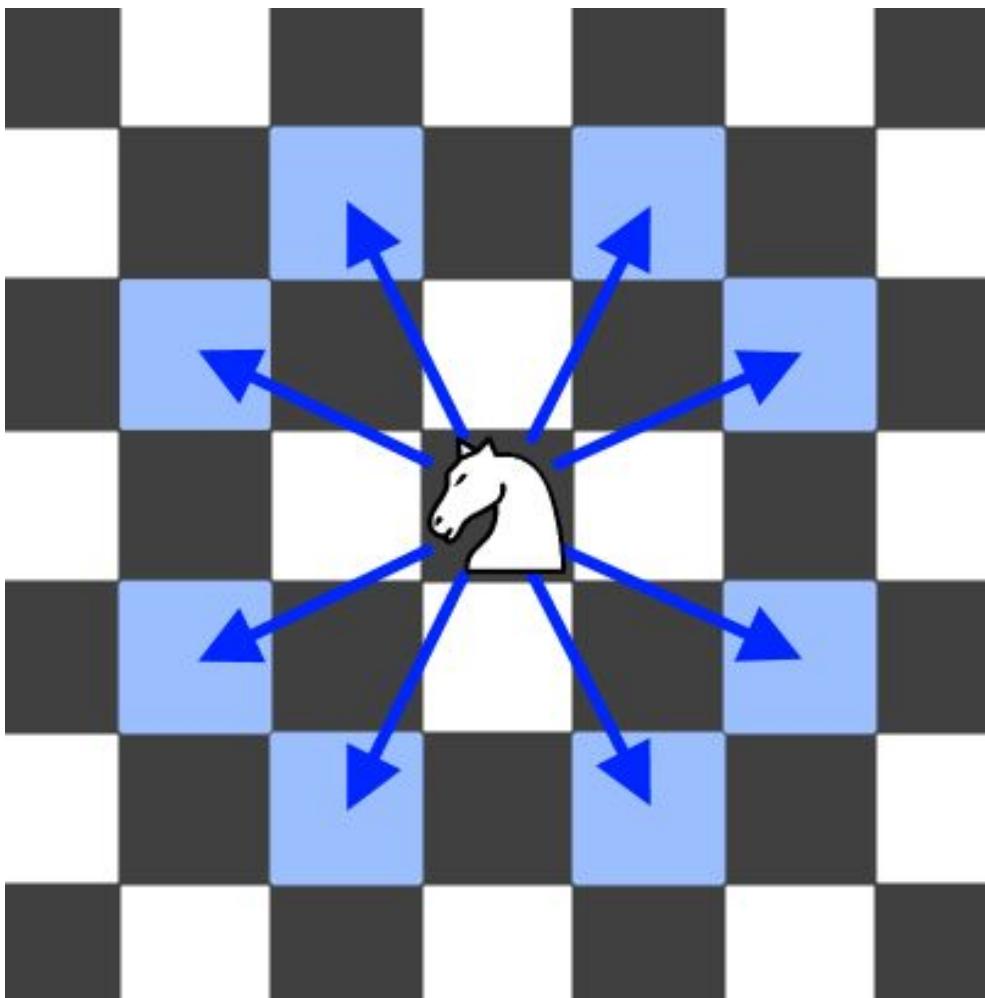
```
["1","1","0","0","0"],  
["1","1","0","0","0"],  
["0","0","1","0","0"],  
["0","0","0","1","1"]
```



3

# Diagrama de solución: **Número de Islas**

Solución en código:  
**Número de Islas**



origenX = 0  
origenY = 0  
objetivoX = 5  
objetivoY = 5



**4**

[0, 0] → [2, 1] → [4, 2] → [3, 4] → [5, 5]

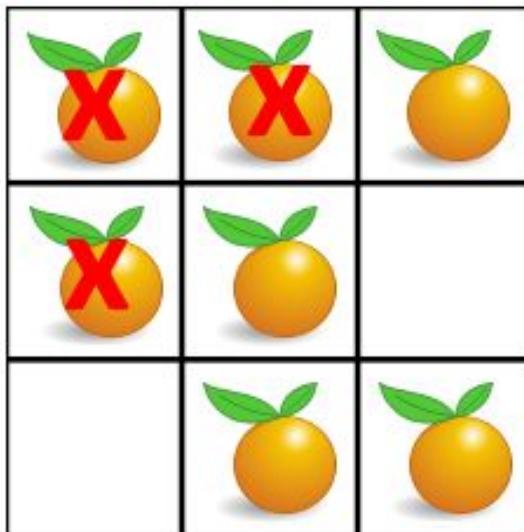
Diagrama de solución:  
**Minimum Knights  
Move**

Solución en código:  
**Minimum Knights  
Move**

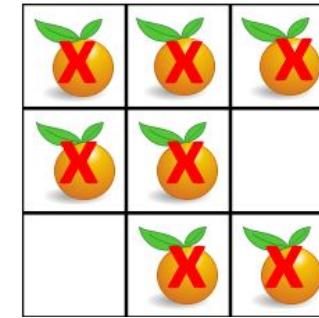
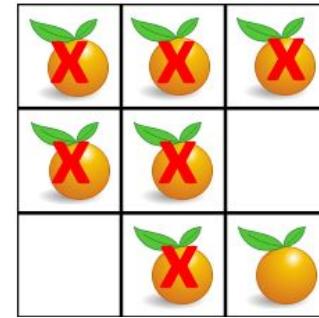
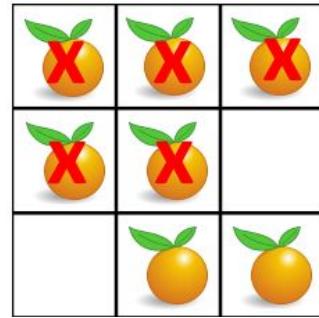
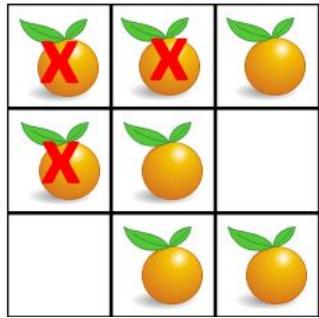
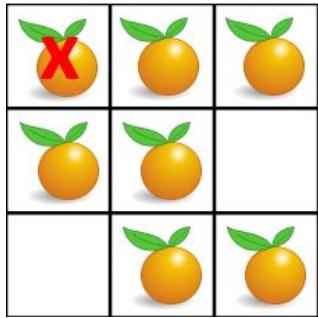
# Análisis del problema: **Rotting Oranges**



Se le da una cuadrícula  $m \times n$  en la que cada celda puede tener uno de los tres valores siguientes



- 0 que representa una celda vacía,
- 1 que representa una naranja fresca
- 2 que representa una naranja podrida



Cada día, cualquier naranja fresca que sea 4-direccionalmente adyacente a una naranja podrida se convierte en podrida.

Retorna el número mínimo de días que deben transcurrir hasta que ninguna celda tenga una naranja fresca. Si esto es imposible, devuelve -1.

$[2, 1, 1],$   
 $[1, 1, 0], \longrightarrow$  **4**  
 $[0, 1, 1]$

# Diagrama de solución: **Rotting Oranges**

Solución en código:  
**Rotting Oranges**

Análisis del problema:  
**Shortest Bridge  
Between Islands**

Dada una matriz binaria  $n \times n$  en la que 1 representa la tierra y 0 el agua, asumiendo que hay dos islas exactas en el mapa, retorna el tamaño del puente más corto posible para conectar las dos islas.



Una isla está rodeada de agua y se forma conectando celdas con tierra adyacentes.

Puede suponer que los cuatro bordes de la cuadrícula están rodeados de agua.

[1,1,1,1,1],  
[1,0,0,0,1],  
[1,0,1,0,1],  
[1,0,0,0,1],  
[1,1,1,1,1] → 1

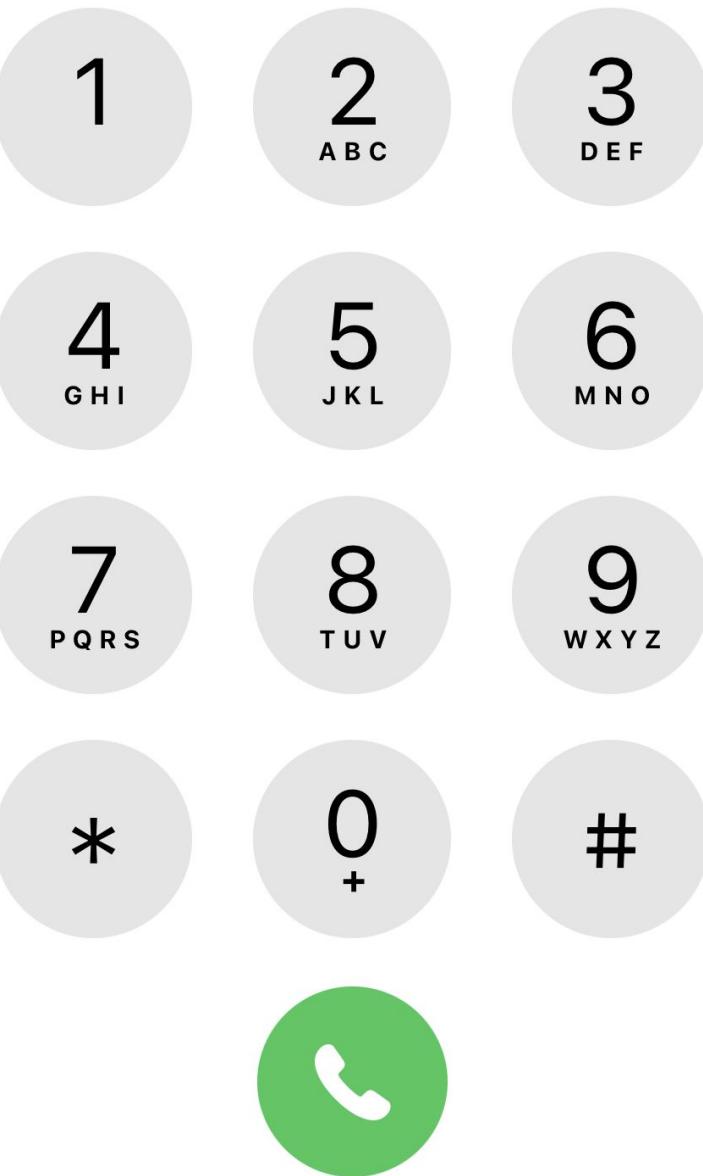
Diagrama de solución:  
**Shortest Bridge  
Between Islands**

Solución en código:  
**Shortest Bridge  
Between Islands**

Análisis del problema:  
**Letter Combinations  
of a Phone Number**

Dada una cadena que contiene dígitos del 2 al 9, devuelve en cualquier orden todas las posibles combinaciones de letras que el número podría representar.

A continuación se muestra una asignación de dígitos a letras (como en los botones del teléfono).



"23"



[ "ad", "ae",  
"af", "bd",  
"be", "bf",  
"cd", "ce",  
"cf" ]

Diagrama de solución:  
**Letter Combinations  
of a Phone Number**

Solución en código:  
**Letter Combinations  
of a Phone Number**

Análisis del problema:  
**Restore IP  
Addresses**



Dada una cadena **s** que contiene sólo dígitos,  
devuelve todas las posibles  
direcciones IP válidas que se pueden formar  
insertando puntos en **s**.

No se permite reordenar o eliminar ningún  
dígito en **s**. Se pueden devolver las direcciones IP  
válidas en cualquier orden.



Una dirección IP válida está formada por exactamente cuatro enteros separados por puntos simples.

Cada número entero está comprendido entre 0 y 255 y no puede tener ceros a la izquierda.

Por ejemplo, "0.1.2.201" y "192.168.1.1" son direcciones IP válidas, pero "0.011.255.245", "192.168.1.312" y "192.168@1.1" son direcciones IP no válidas.

"25525511135" → ["255.255.11.135",  
"255.255.111.35"]

Solución en código:  
**Restore IP  
Addresses**

Dada una cuadrícula  $m \times n$  de caracteres board y una cadena word, devuelve true si word existe en la cuadrícula.

La palabra puede construirse a partir de letras de celdas secuencialmente adyacentes, donde las celdas adyacentes son vecinas horizontal o verticalmente. La misma celda de letra no puede ser utilizada más de una vez.

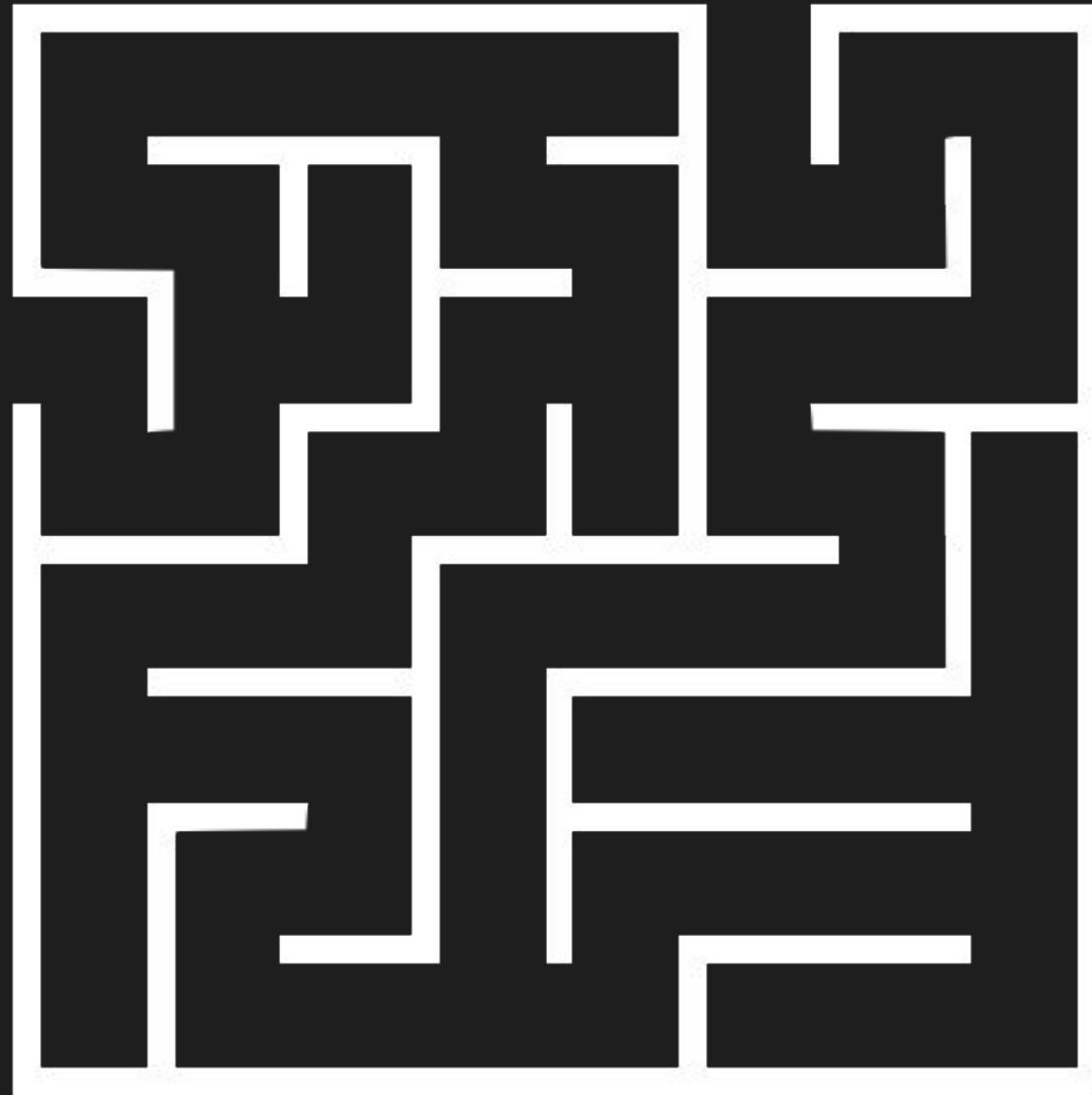
A	B	C	E
S	F	C	S
A	D	E	E

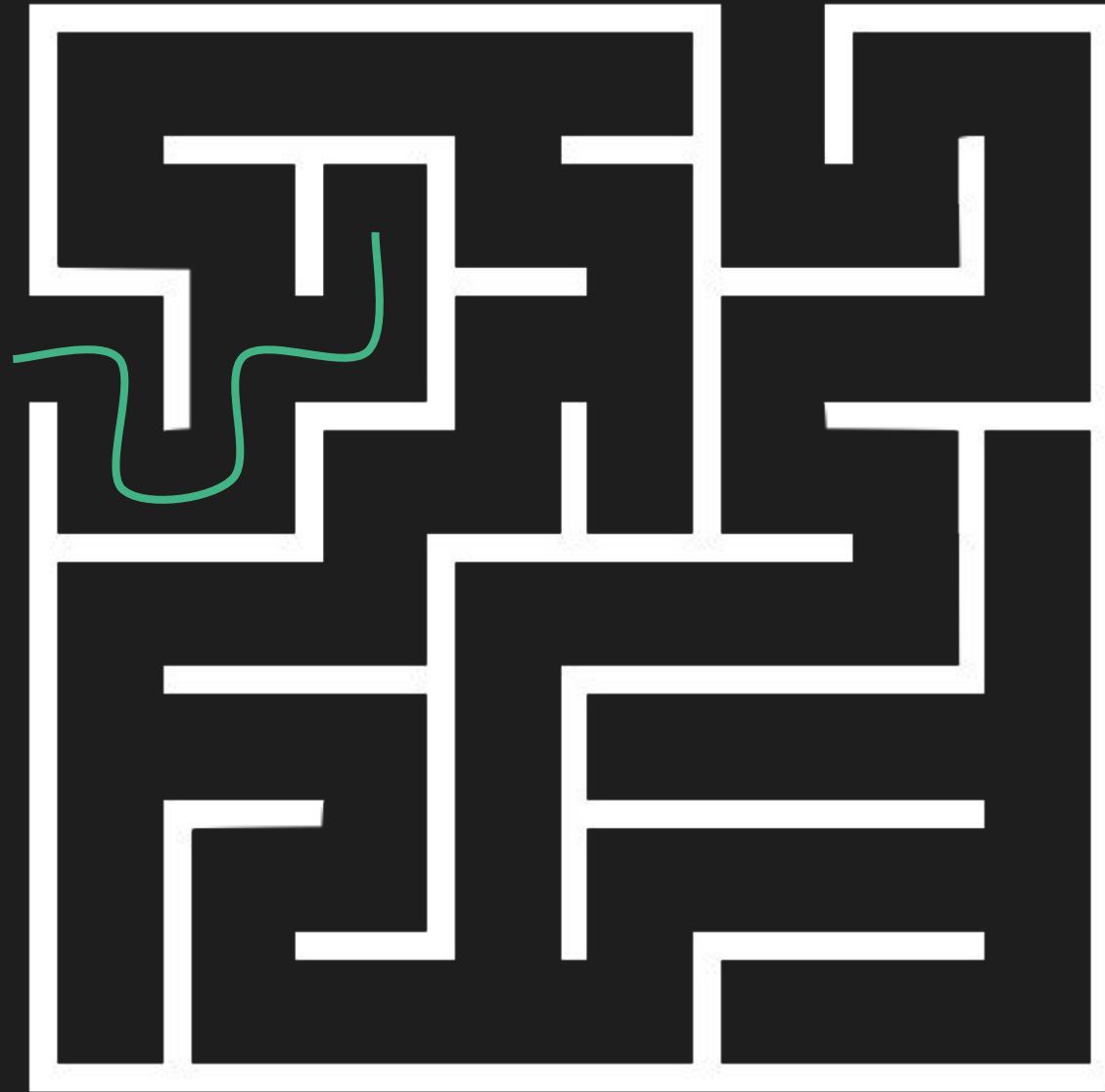
```
Input: board =
[[ "A", "B", "C", "E" ], [ "S", "F", "C", "S" ], [
"A", "D", "E", "E" ]], word = "ABCED"
Output: true
```

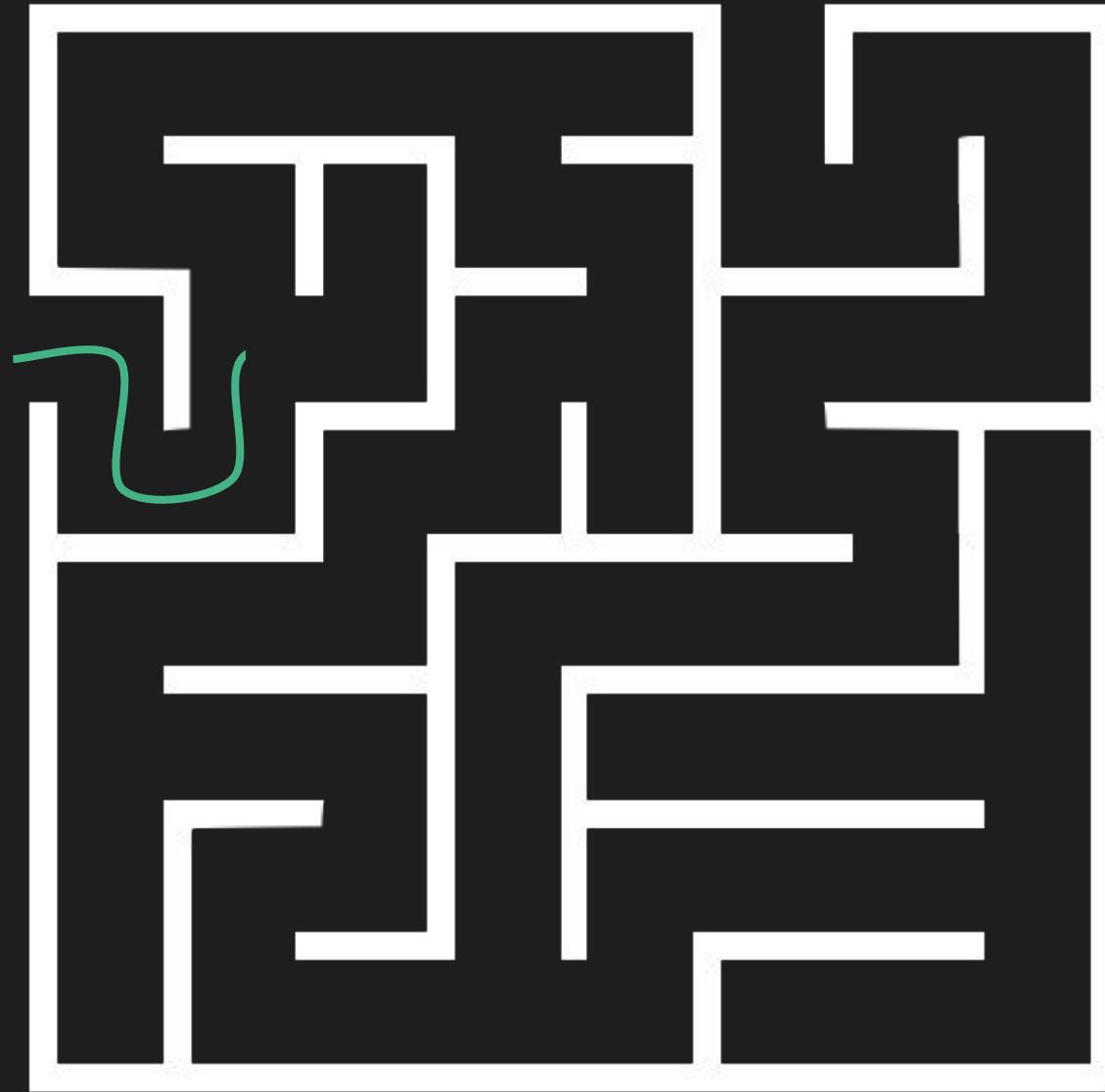
Diagrama de solución:  
**Letter Combinations  
of a Phone Number**

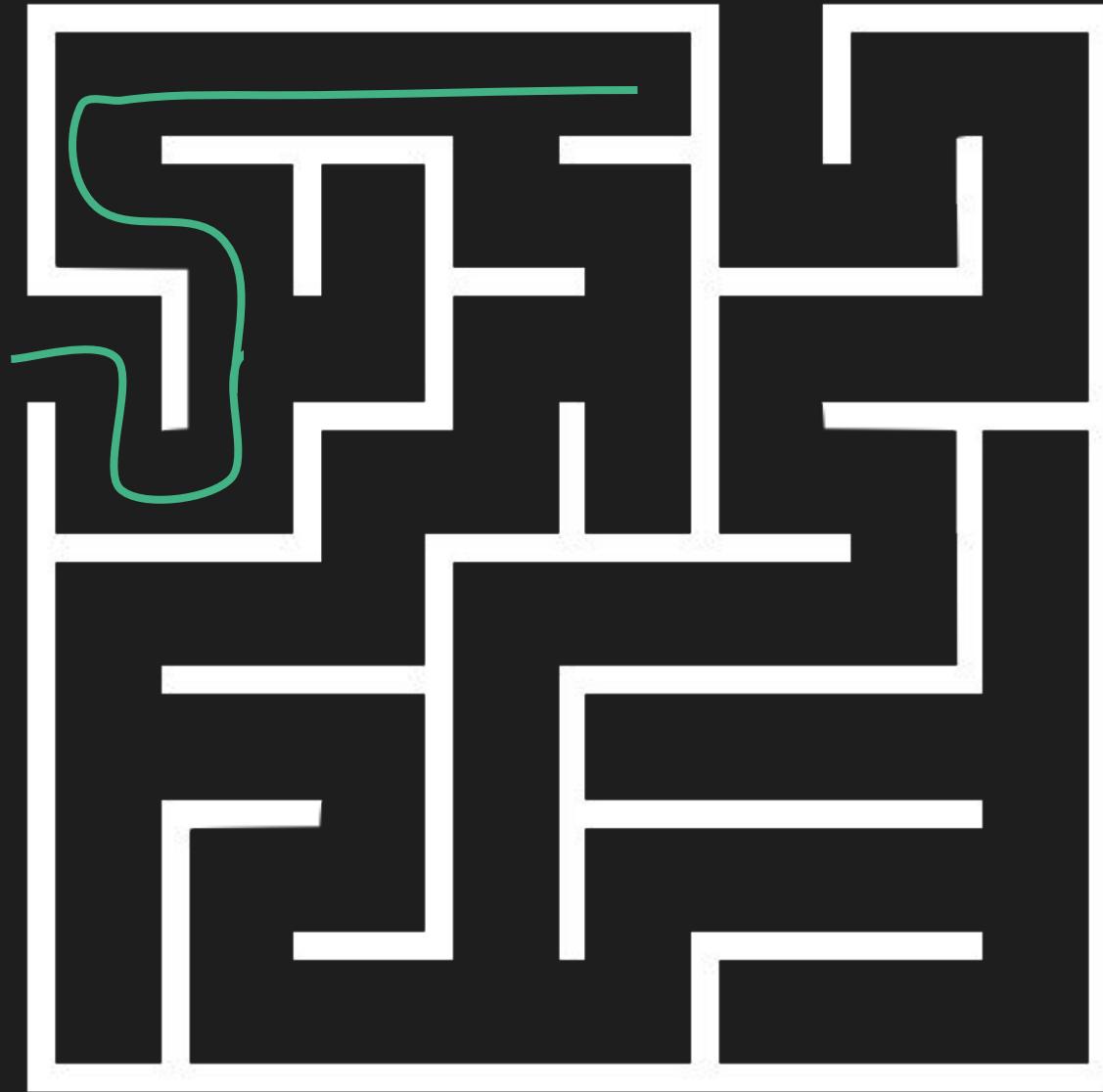
Solución en código:  
**Letter Combinations  
of a Phone Number**

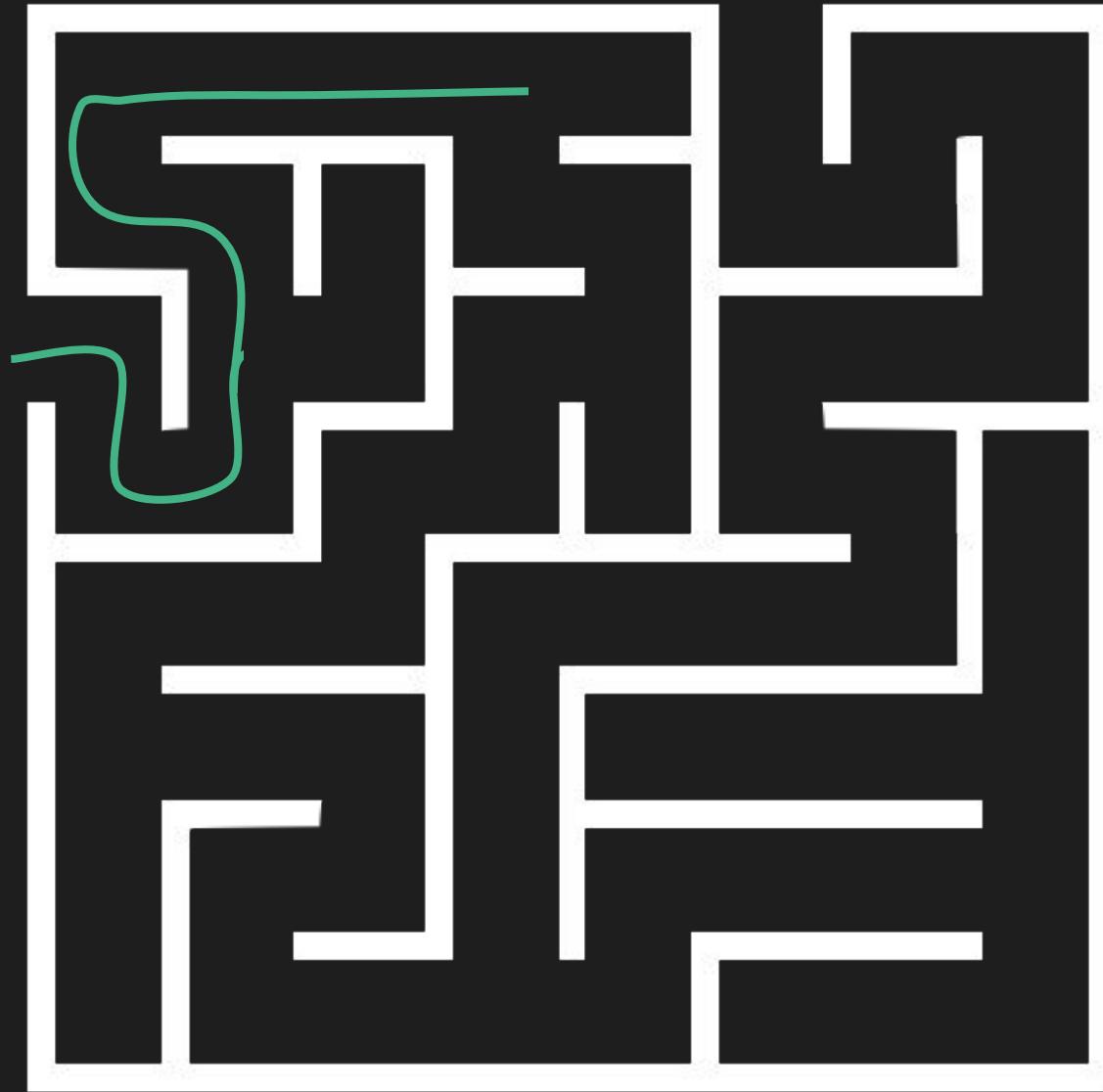
# Backtracking

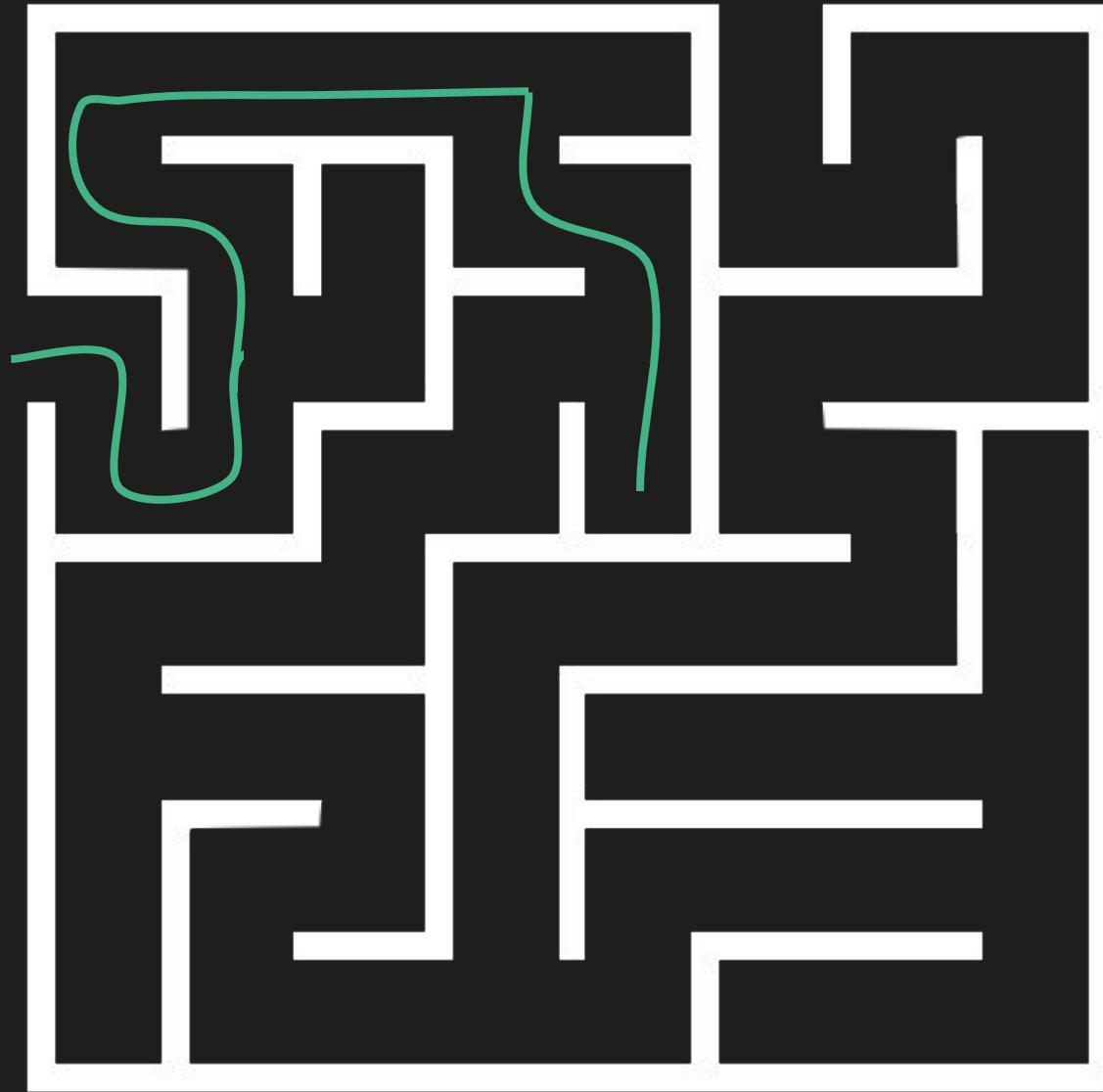


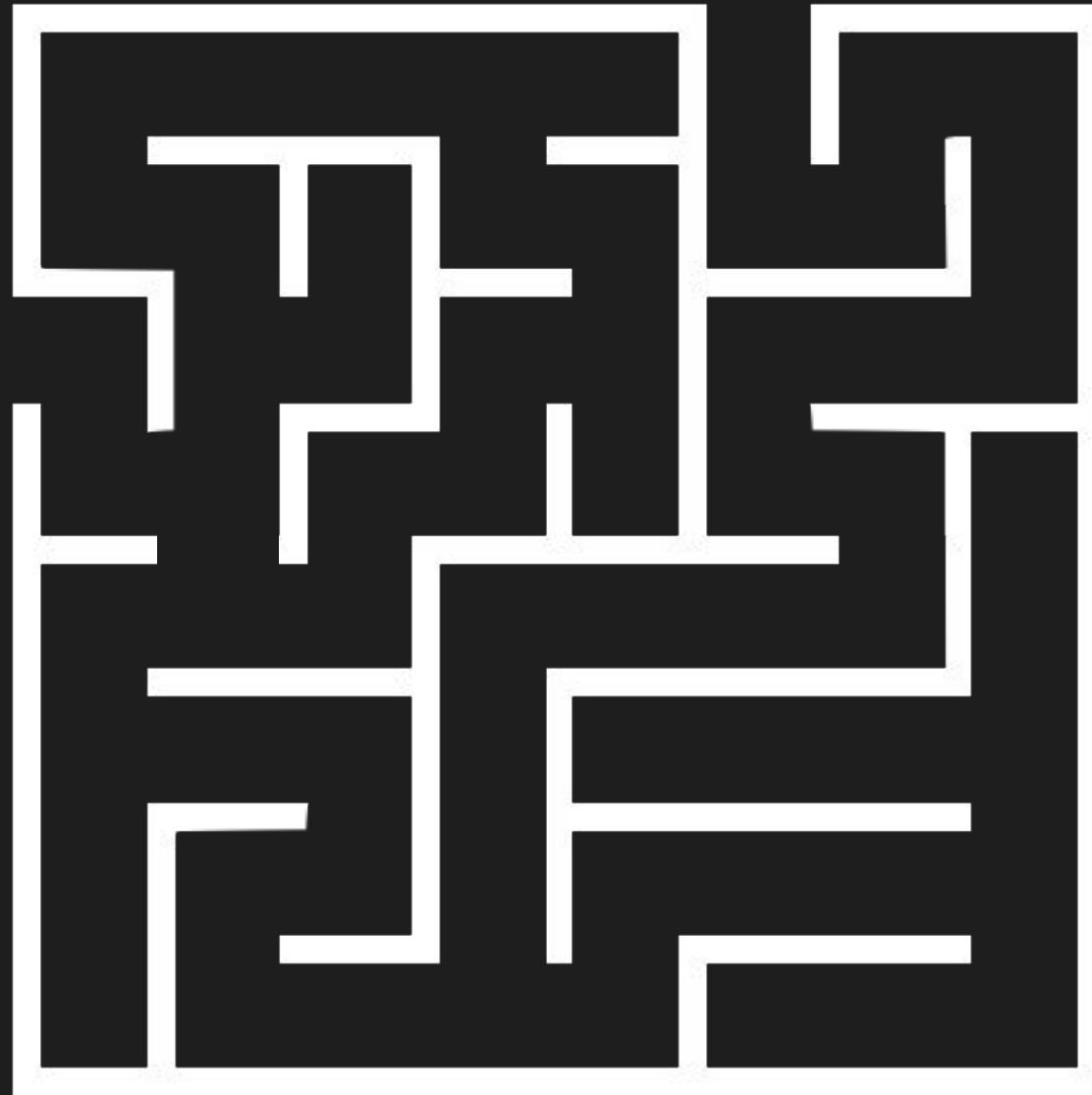


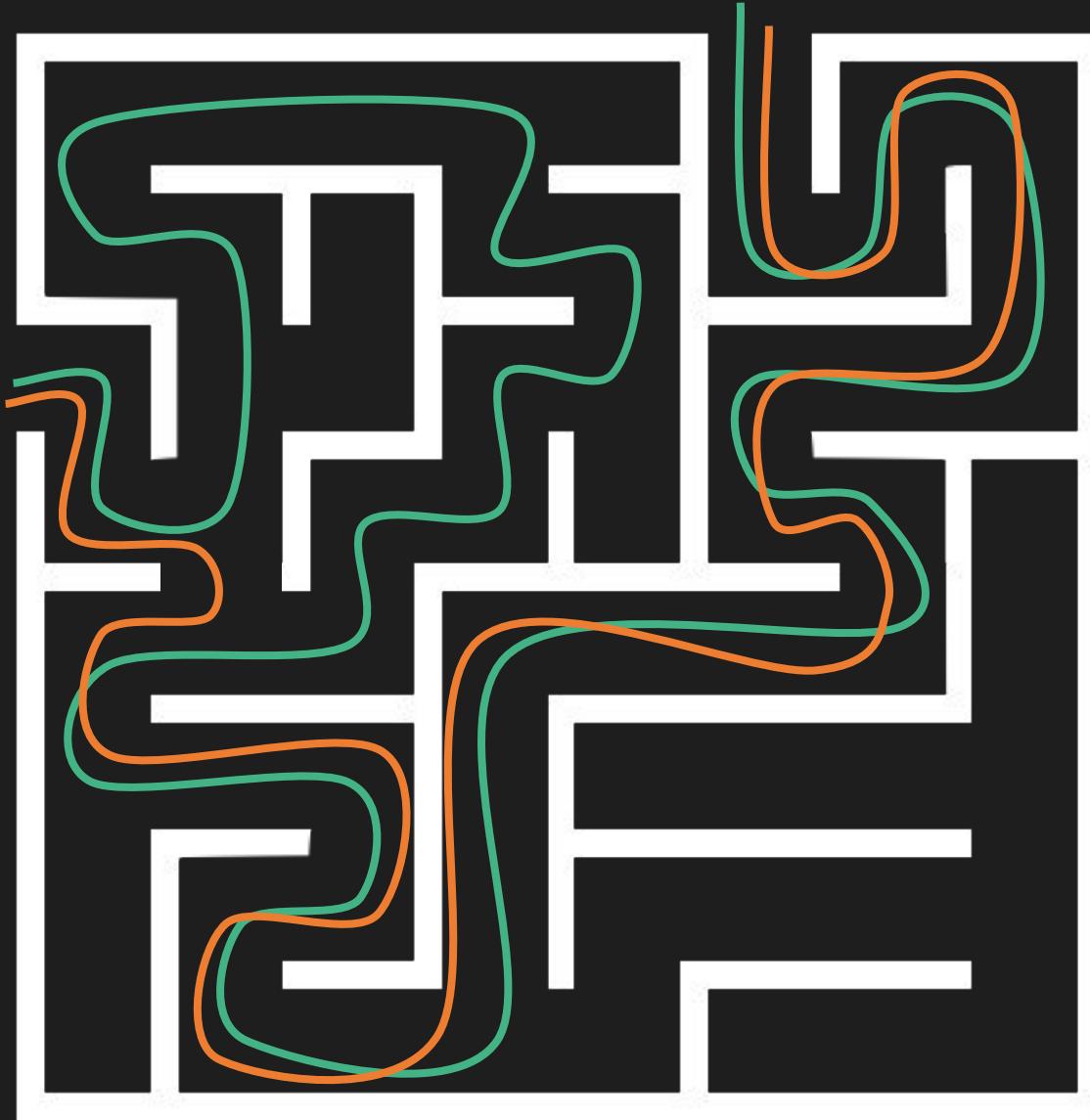










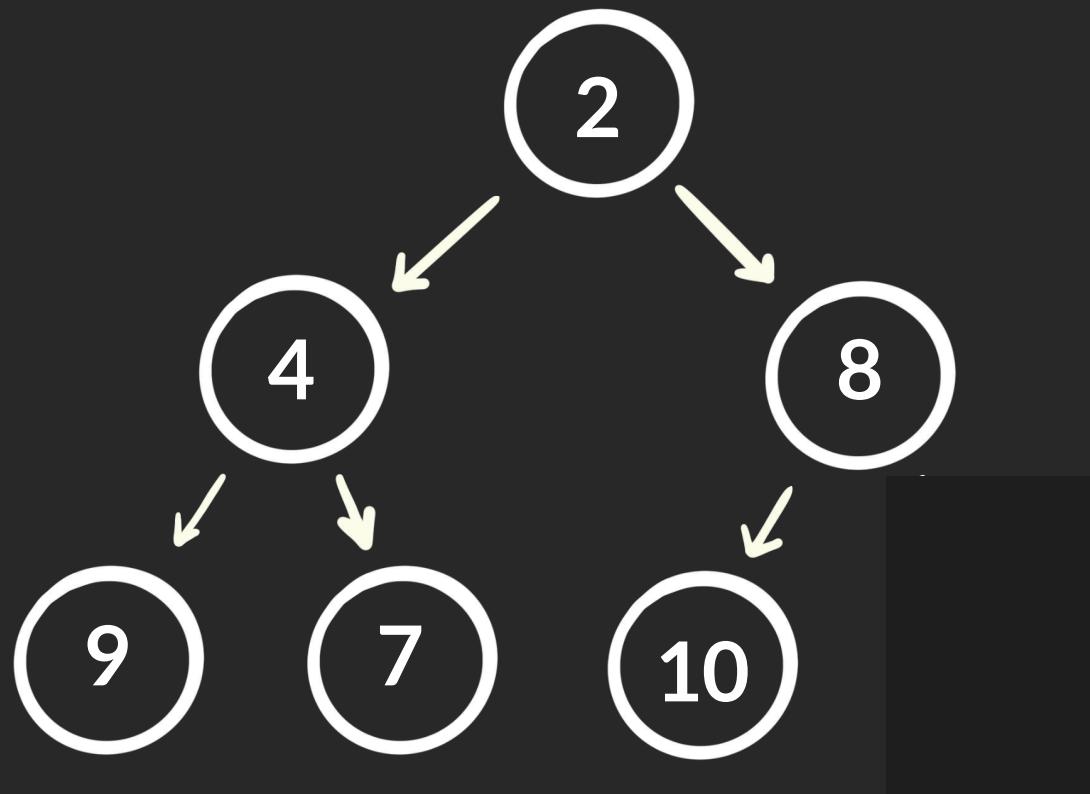


Abandono las opciones cuando veo que  
estas ya no pueden llevarme a la  
solución que necesito.

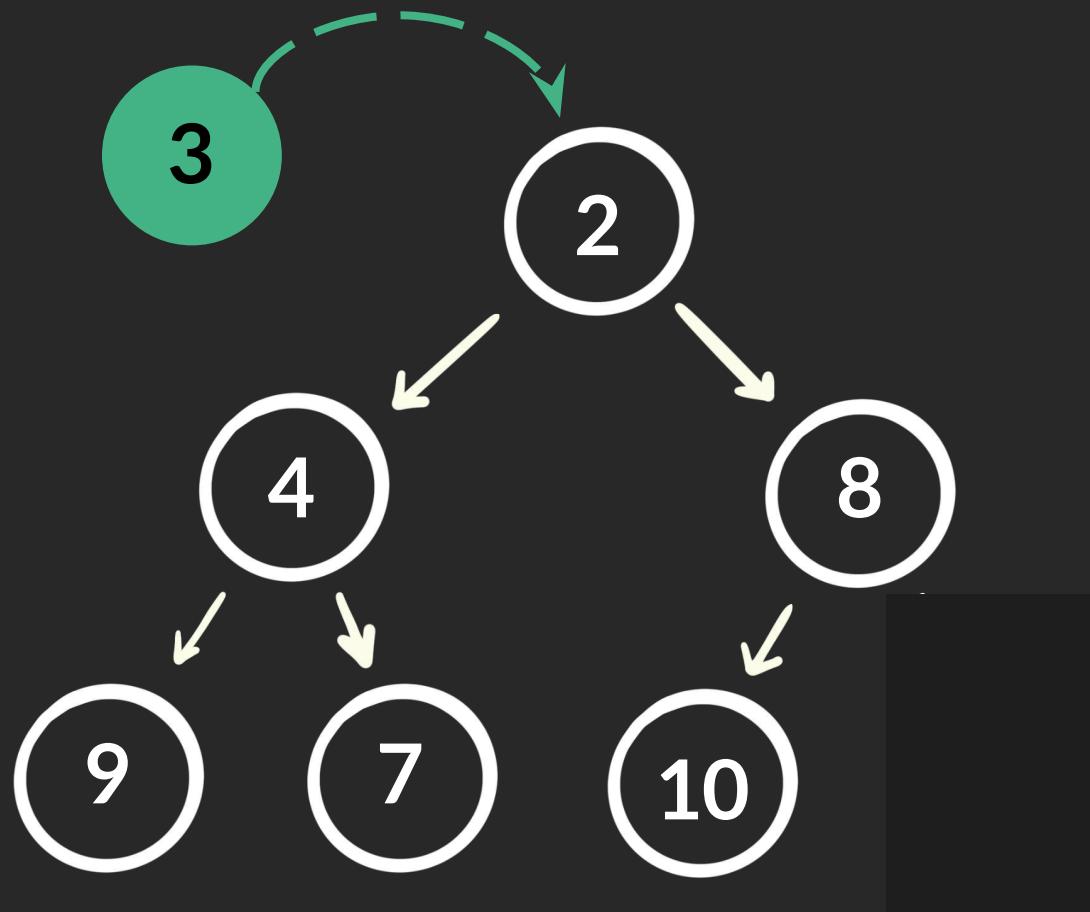
Se buscan **todas** las combinaciones posibles para resolver un problema computacional.

**¿Qué otros  
algoritmos y  
tipos de grafos  
puedes aprender?**

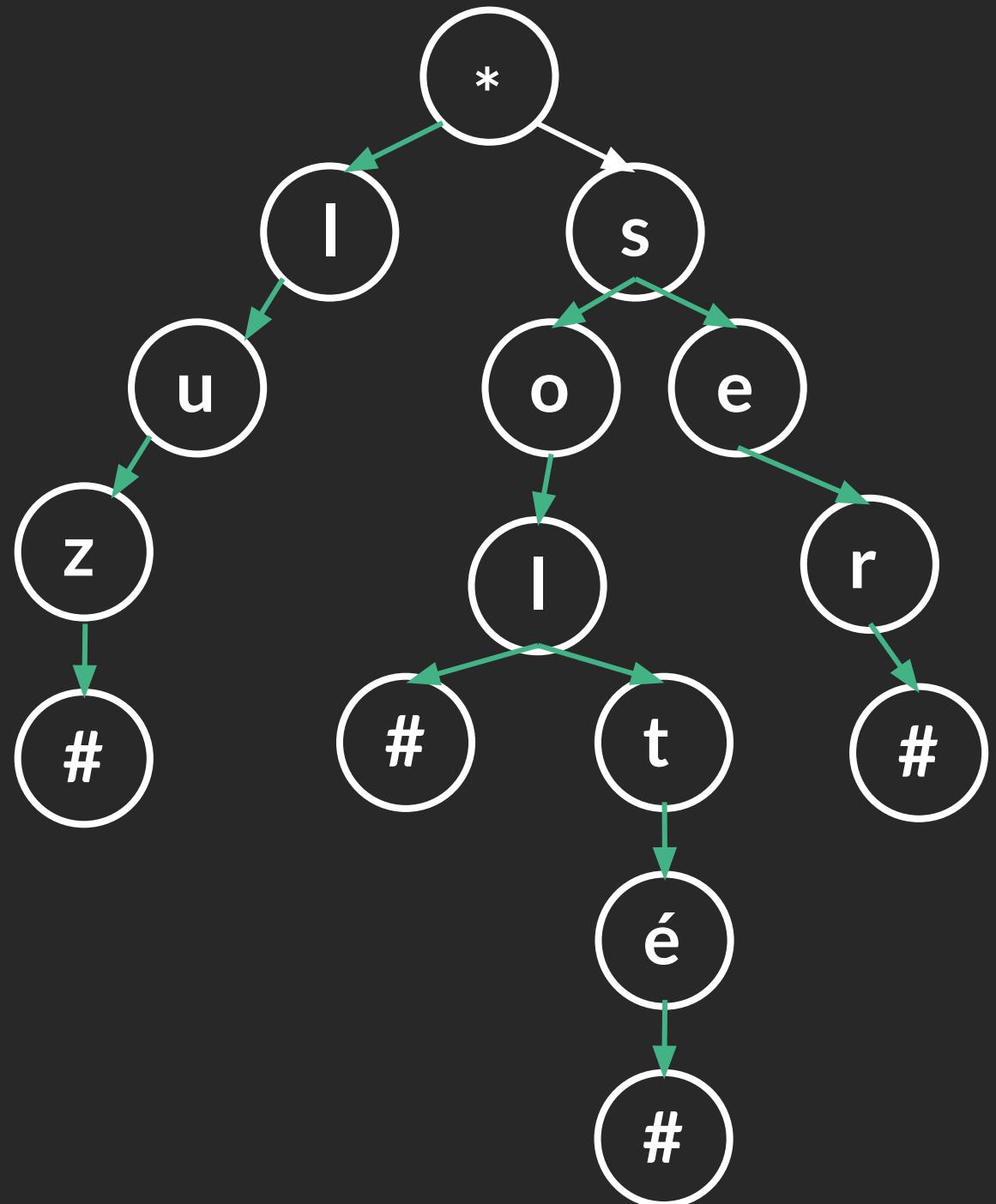
# Max/Min Heap



# Max/Min Heap

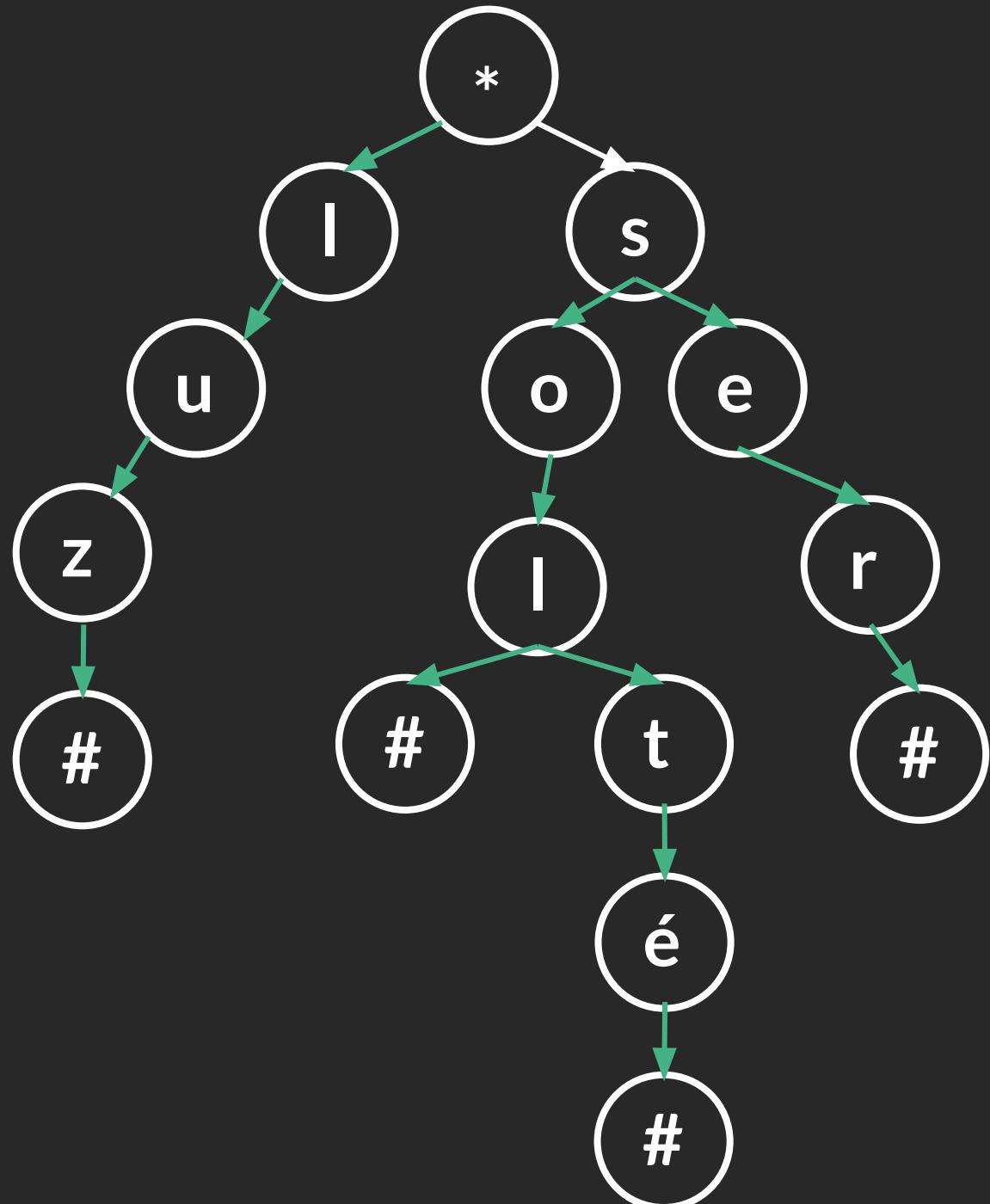


# Tries



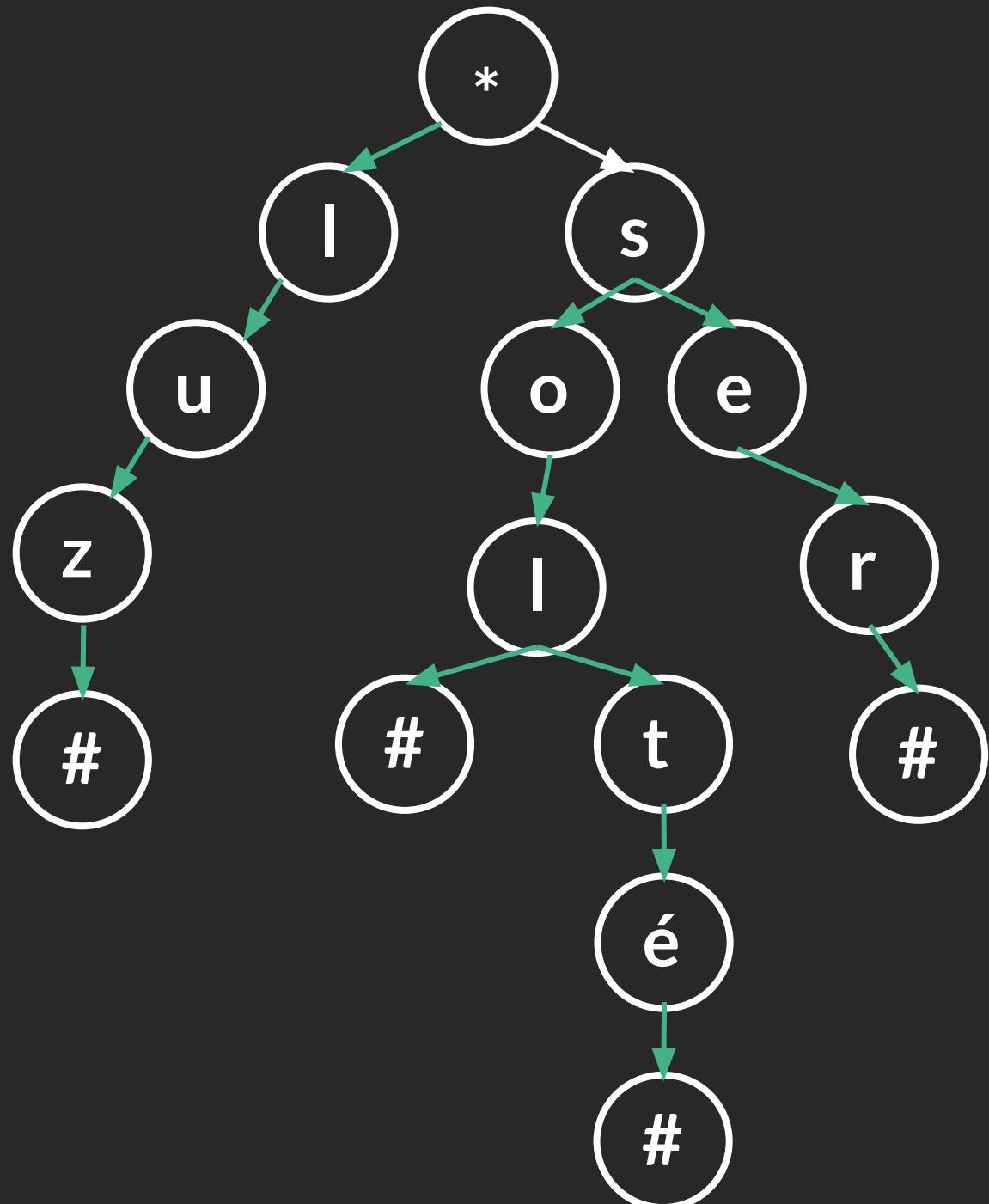
# Tries

- Luz



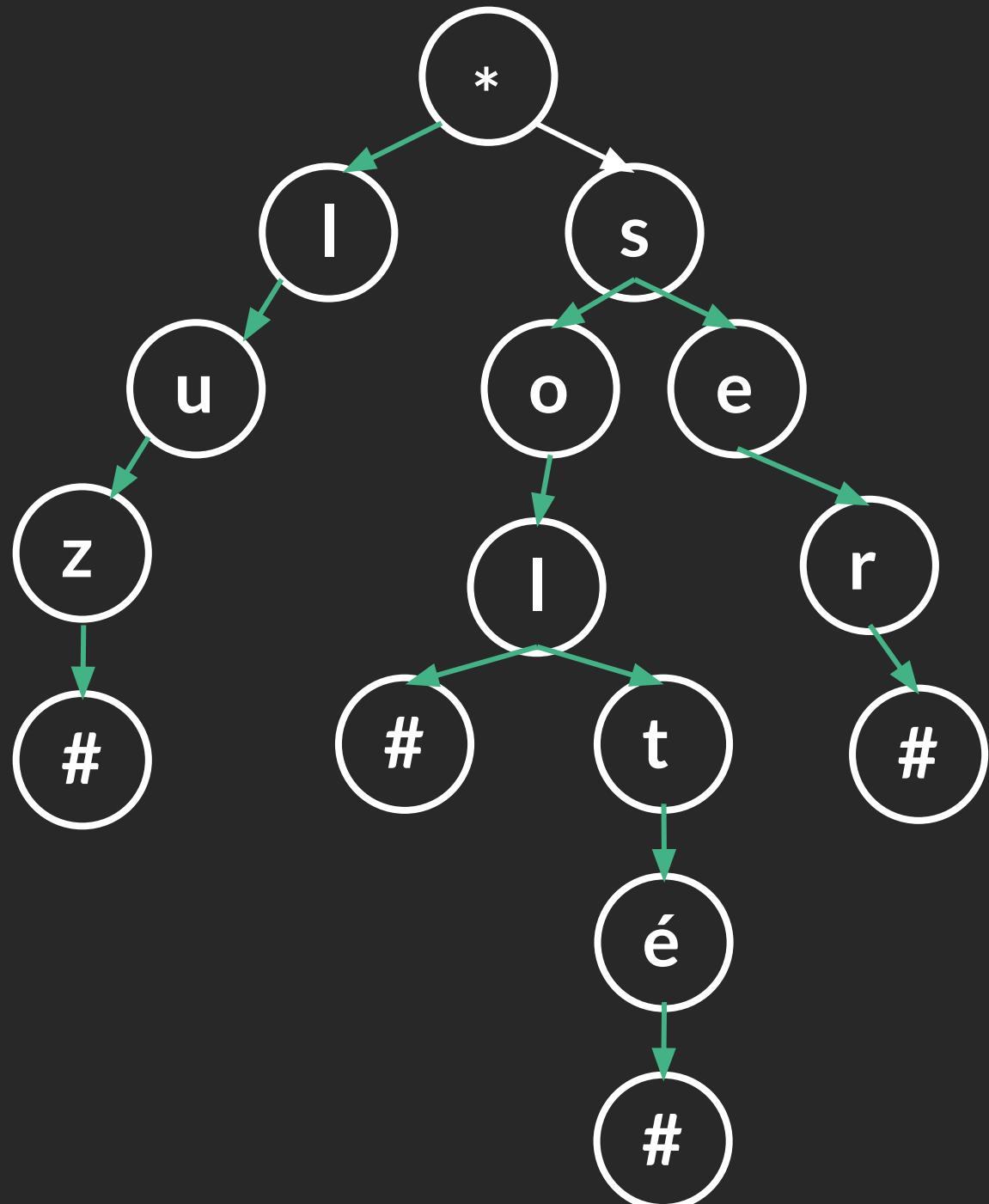
# Tries

- Luz
- Sol
- Solté

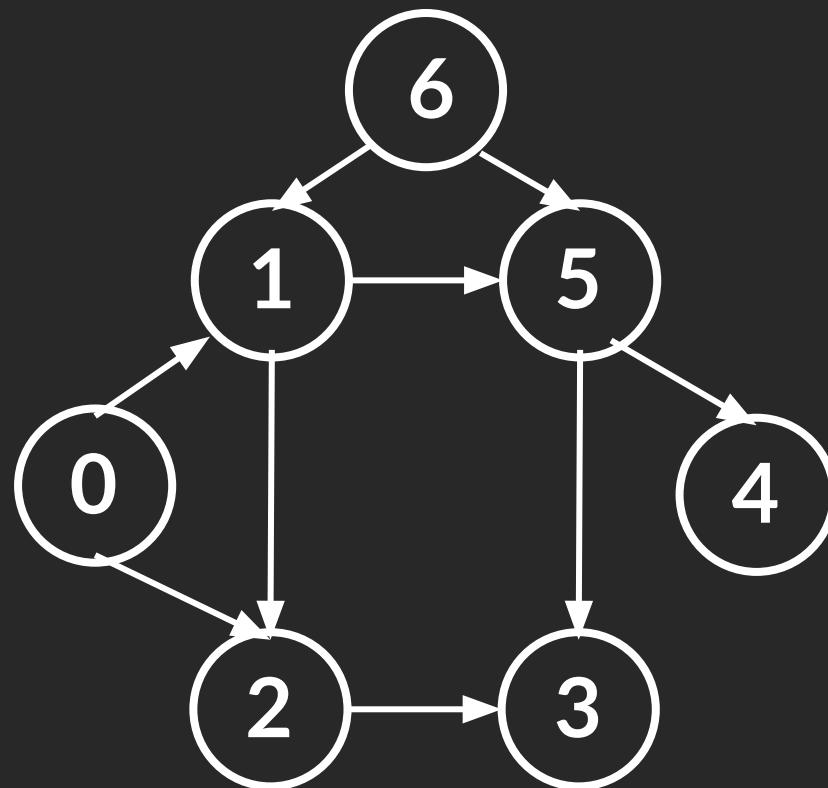


# Tries

- Luz
- Sol
- Solté
- Ser



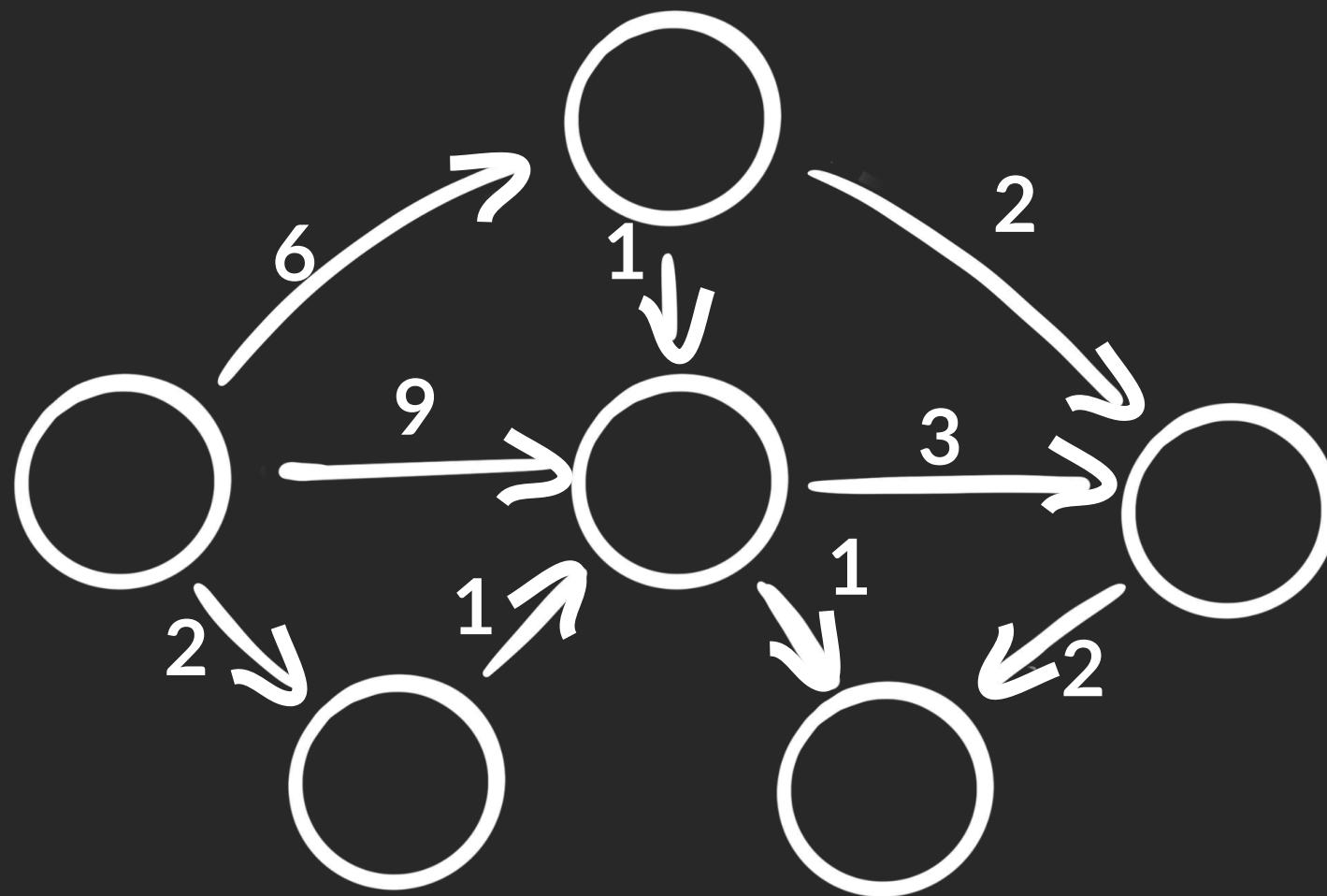
# Topological Sort



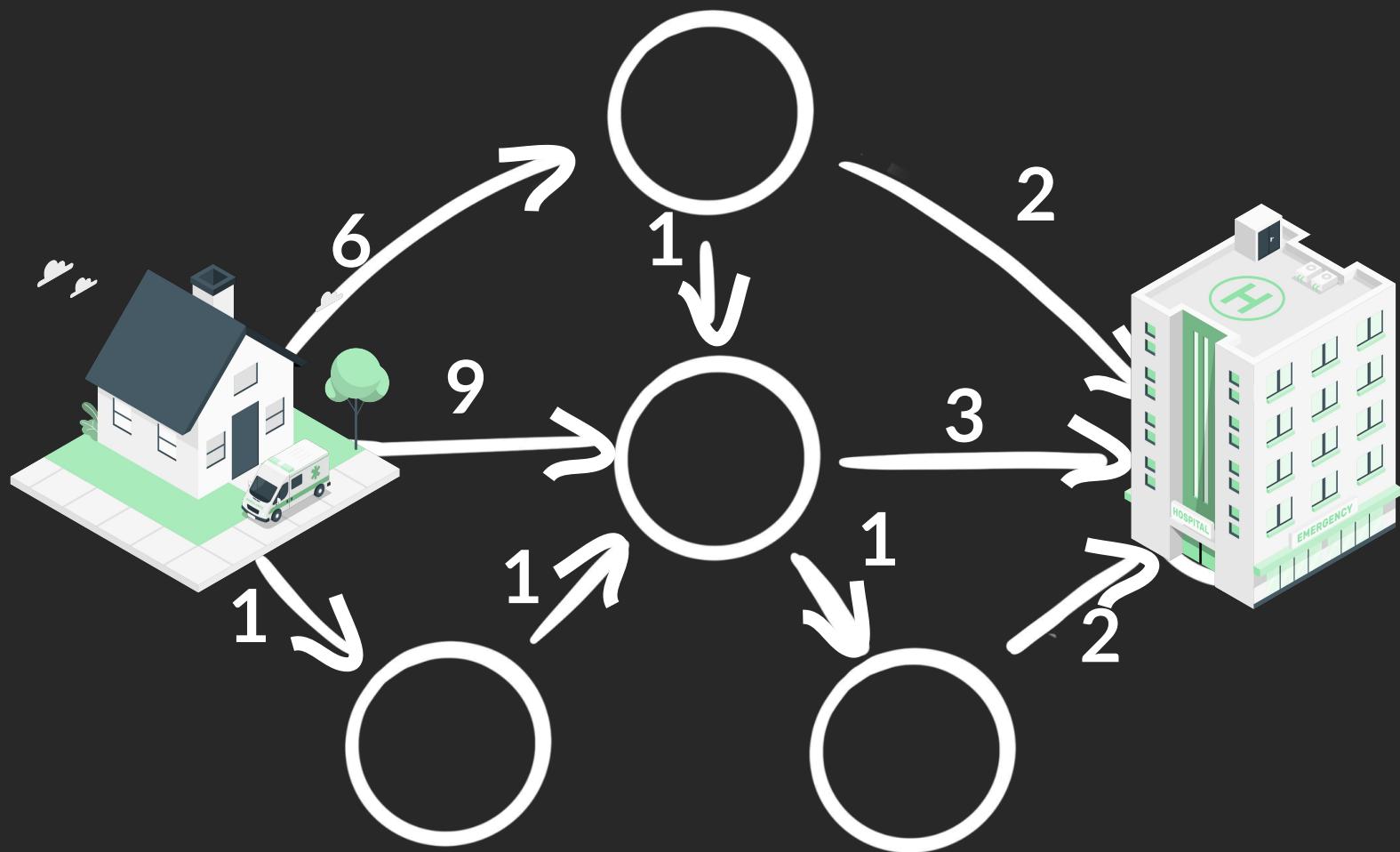
# Topological Sort



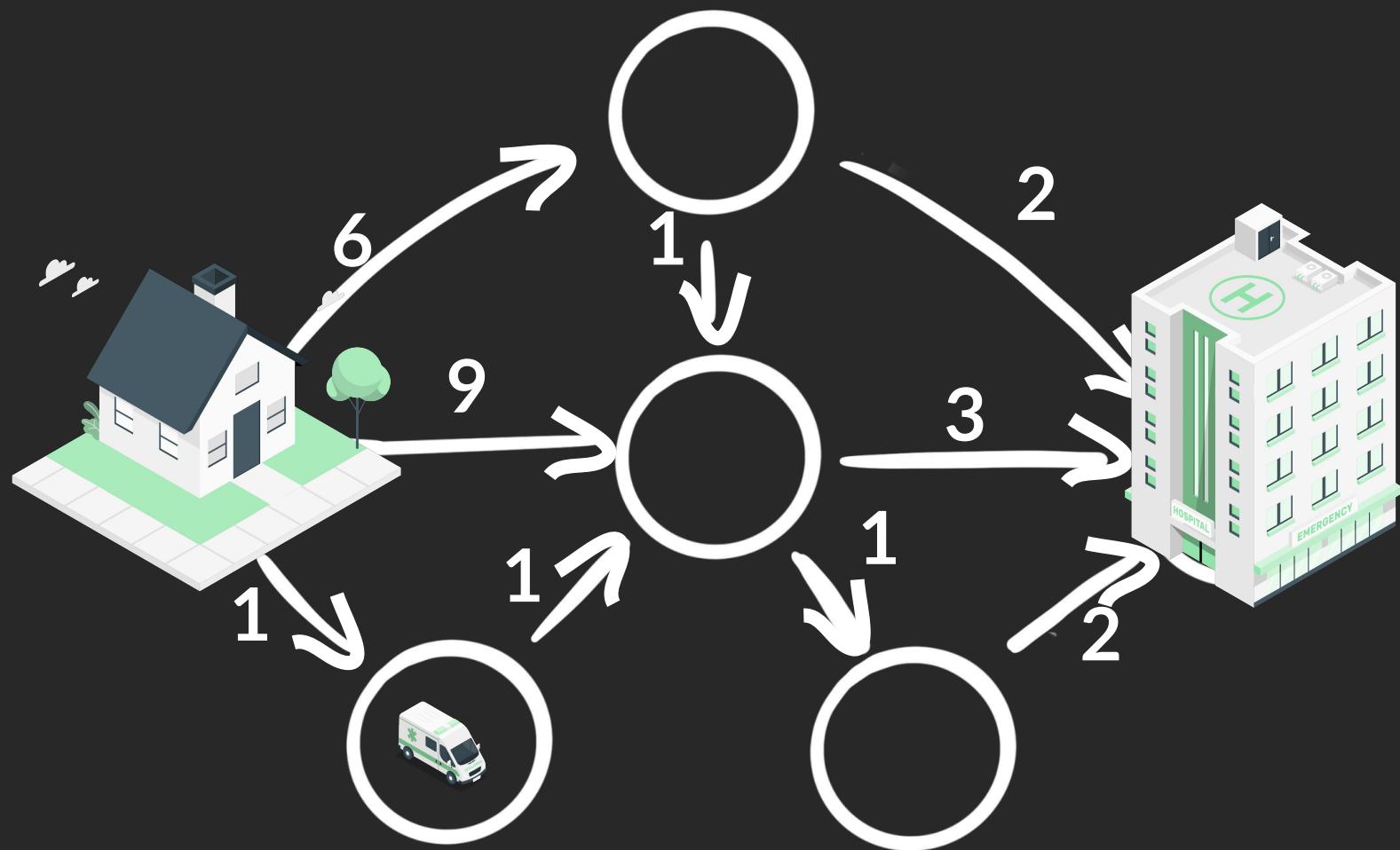
# Dijkstra



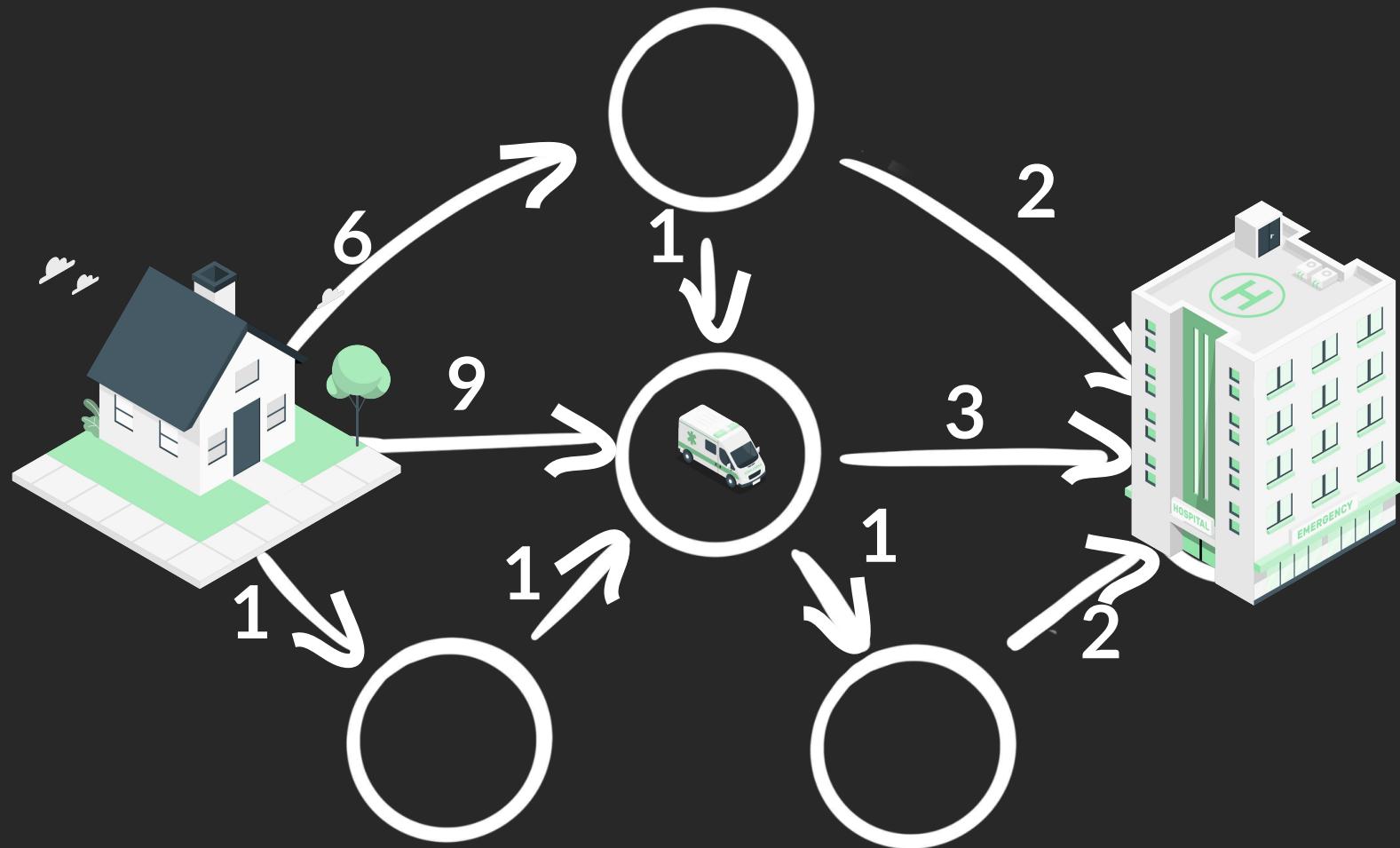
# Dijkstra



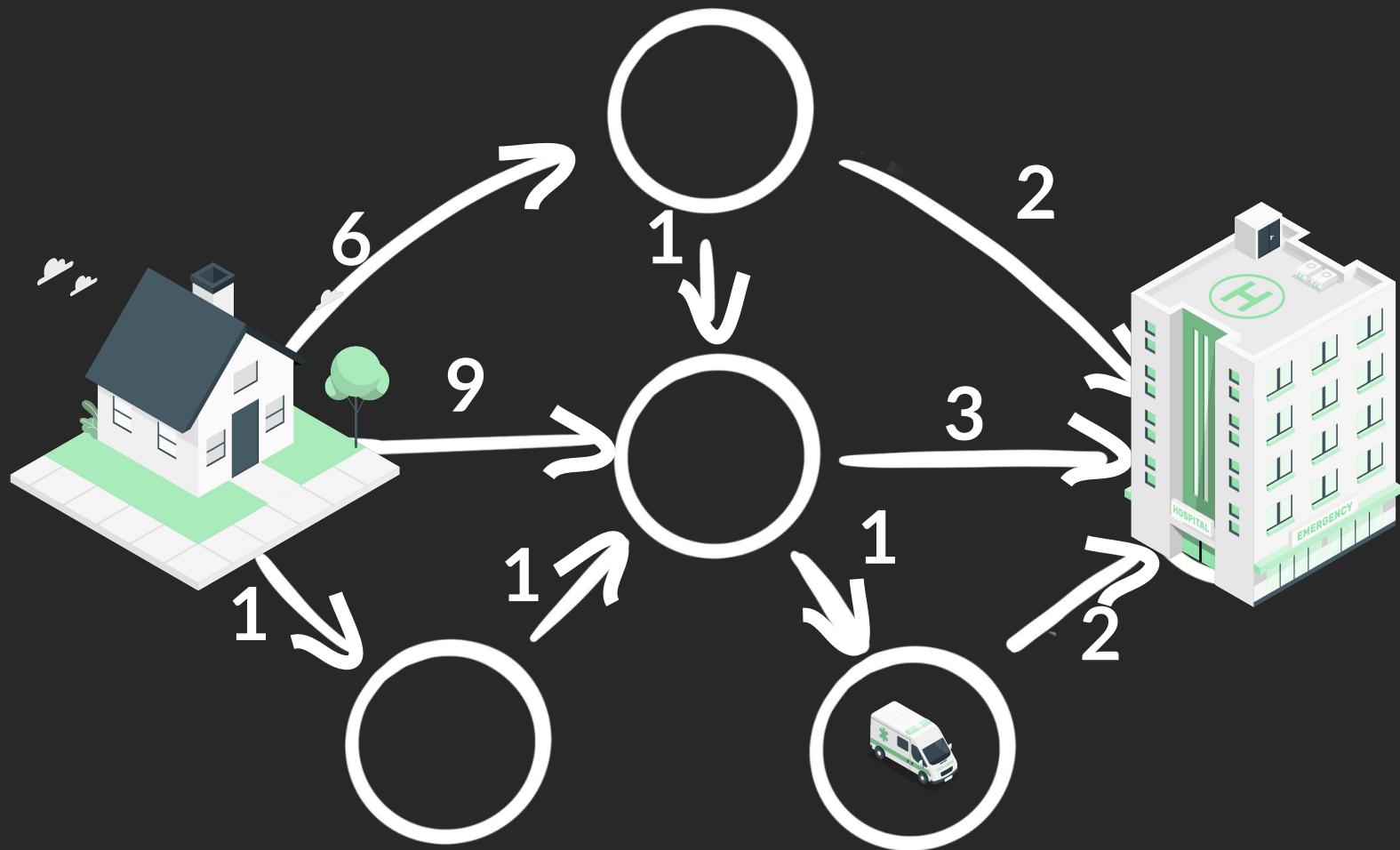
# Dijkstra



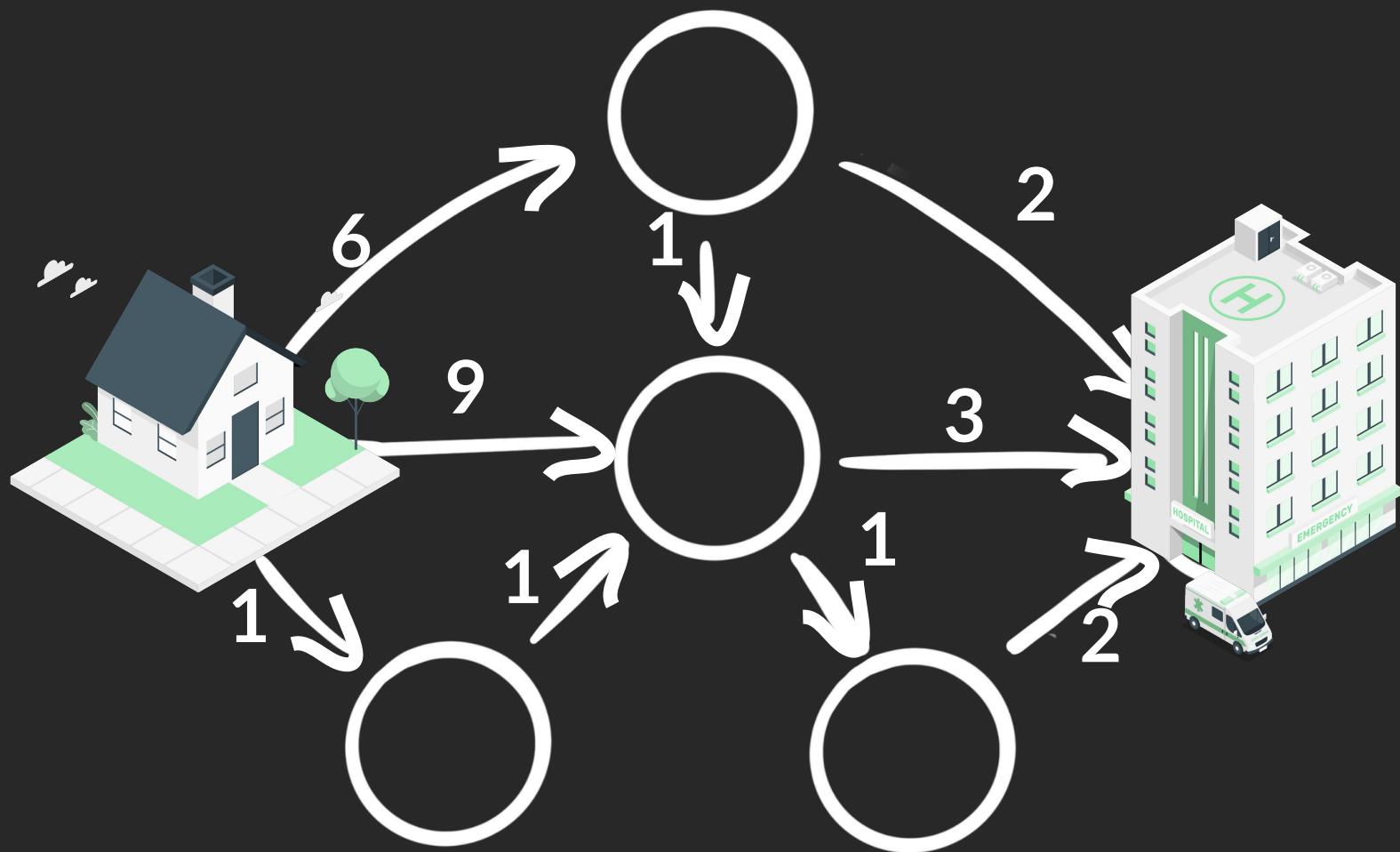
# Dijkstra



# Dijkstra



# Dijkstra



Floyd-Warshall Algorithm

Bellman Ford

Minimum Spanning Tree

Cycle Detection

Maximum Flow

Graph Coloring

# **End of course**

**- No eliminar -**

# Curso Avanzado de Algoritmos: **Ordenamiento Topológico, Backtrack, Islas y Paréntesis**

@camilalonart

Análisis del problema:  
**Letter Combinations  
of a Phone Number**

Diagrama de solución:  
**Letter Combinations  
of a Phone Number**

Solución en código:  
**Letter Combinations  
of a Phone Number**

Análisis del problema:  
**Valid String**

# Diagrama de solución: **Valid String**

Solución en código:  
**Valid String**

Análisis del problema:

# **Generate Parentheses**

Diagrama de solución:

# Generate Parentheses

Solución en código:

# **Generate Parentheses**

Análisis del problema:  
**Number of Islands**

# Diagrama de solución: **Number of Islands**

Solución en código:  
**Number of Islands**

¿Qué es  
Ordenamiento  
Topológico?

# Análisis del problema: **Word Search**



Dada una cuadrícula **m x n** de caracteres y una cadena **palabra**, devuelve true si esta existe en la cuadrícula.

La palabra puede construirse a partir de letras de celdas secuencialmente adyacentes, donde las celdas adyacentes son vecinas horizontal o verticalmente.

La misma celda de letra no puede ser utilizada más de una vez.

```
[[ "A", "B", "C", "E" ],  
 [ "S", "F", "C", "S" ],  
 [ "A", "D", "E", "E" ]],
```



Verdadero

```
palabra = "ABCDED"
```

A	B	C	E
S	F	C	S
A	D	E	E

# Diagrama de solución: **Word Search**

Solución en código:  
**Word Search**

**¿Qué otros  
algoritmos y  
tipos de grafos  
puedes aprender?**

# Algoritmos y grafos más conocidos

- Ordenamiento topológico
- Dijkstra

# **End of course**

**- No eliminar -**