

CARLOS LLANES, MICHELLE CÁMARA, MARCELO MEDINA

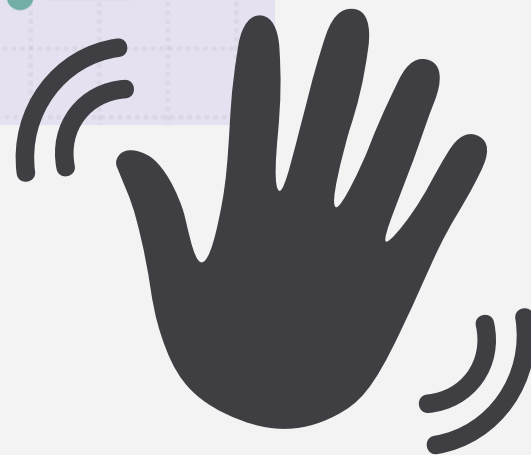
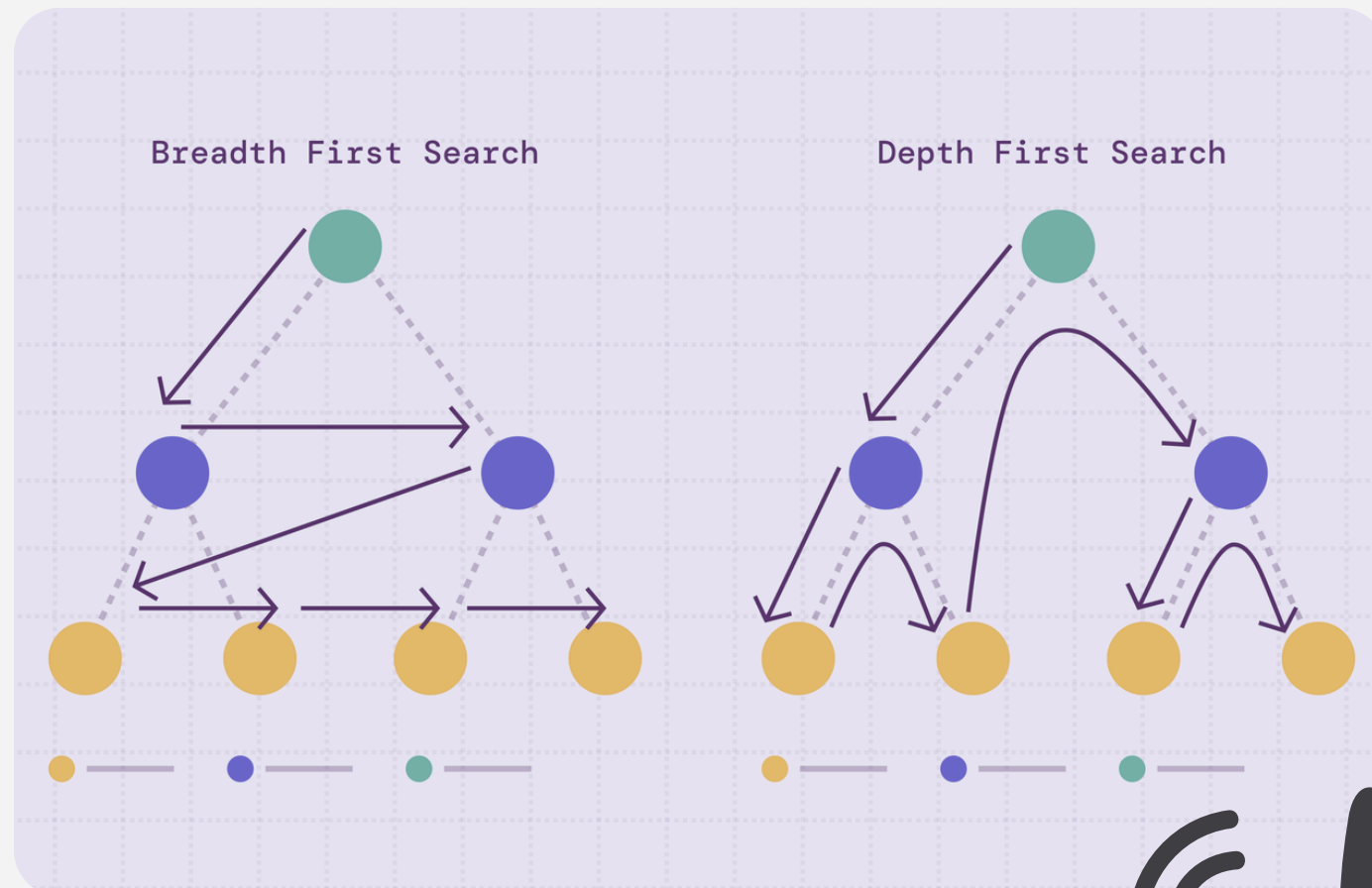
2°C

# bfs project

FEBRUARY 26TH, 2026

DATA STRUCTURE  
DIDIER GAMBOA ANGULO

# bfs project



## introducción

En esta presentación se hablará acerca de la implementación del BFS con el objetivo de implementar una **cola** desde cero y un **grafo**. Aplicar **BFS** para obtener:

1. Orden de visita.
2. Guardar parents para caminos.
3. Reconstruir un camino.

bfs project

# qué es un grafo

Es una estructura matemática que consiste de vértices conectados por aristas que permiten modelar datos.

En mi código el grafo se guarda como un diccionario `adj` y cada nodo tiene sus vecinos.

bfs project

bfs project

# qué es un cola

Una cola es una estructura de datos lineal que sigue el principio FIFO, donde se añaden por un extremo y se eliminan por el otro.

Métodos:

`enqueue(x)` → meter al final

`dequeue()` → sacar del frente

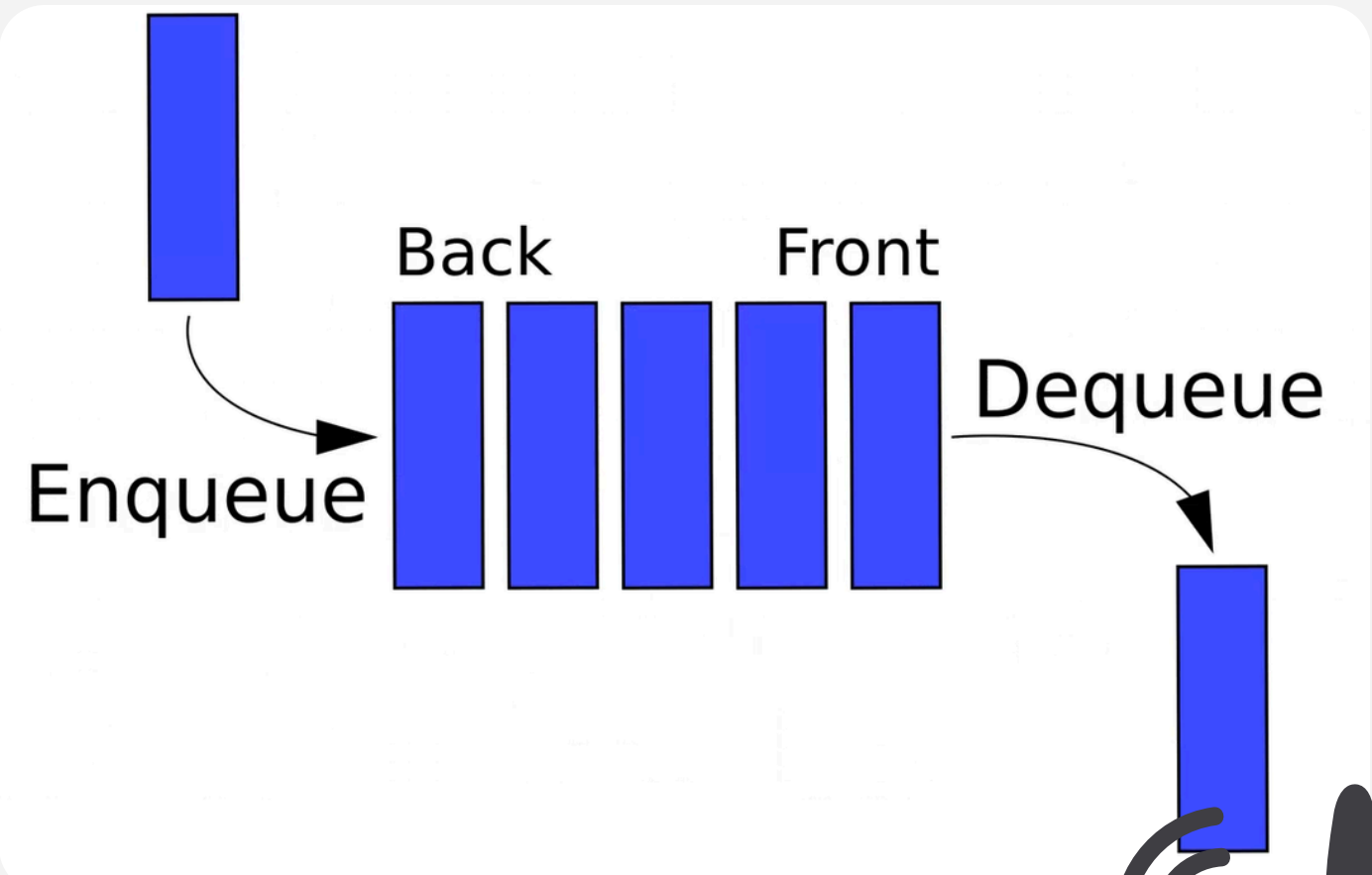
`first()` → ver el frente sin sacarlo de la cola

`is_empty()` → checa si está vacío

`size()` → devuelve el tamaño de la cola

bfs project

# bfs project



## qué es bfs

BFS recorre el grafo por niveles.

Cómo funciona:

1. Pone el **nodo inicial** en la cola.
2. Mientras no esté vacío, realiza un **dequeue**, y agrega sus **vecinos** no visitados.

Esto hace que nuestro BFS vaya por niveles.

## BFS

En BFS se usó:

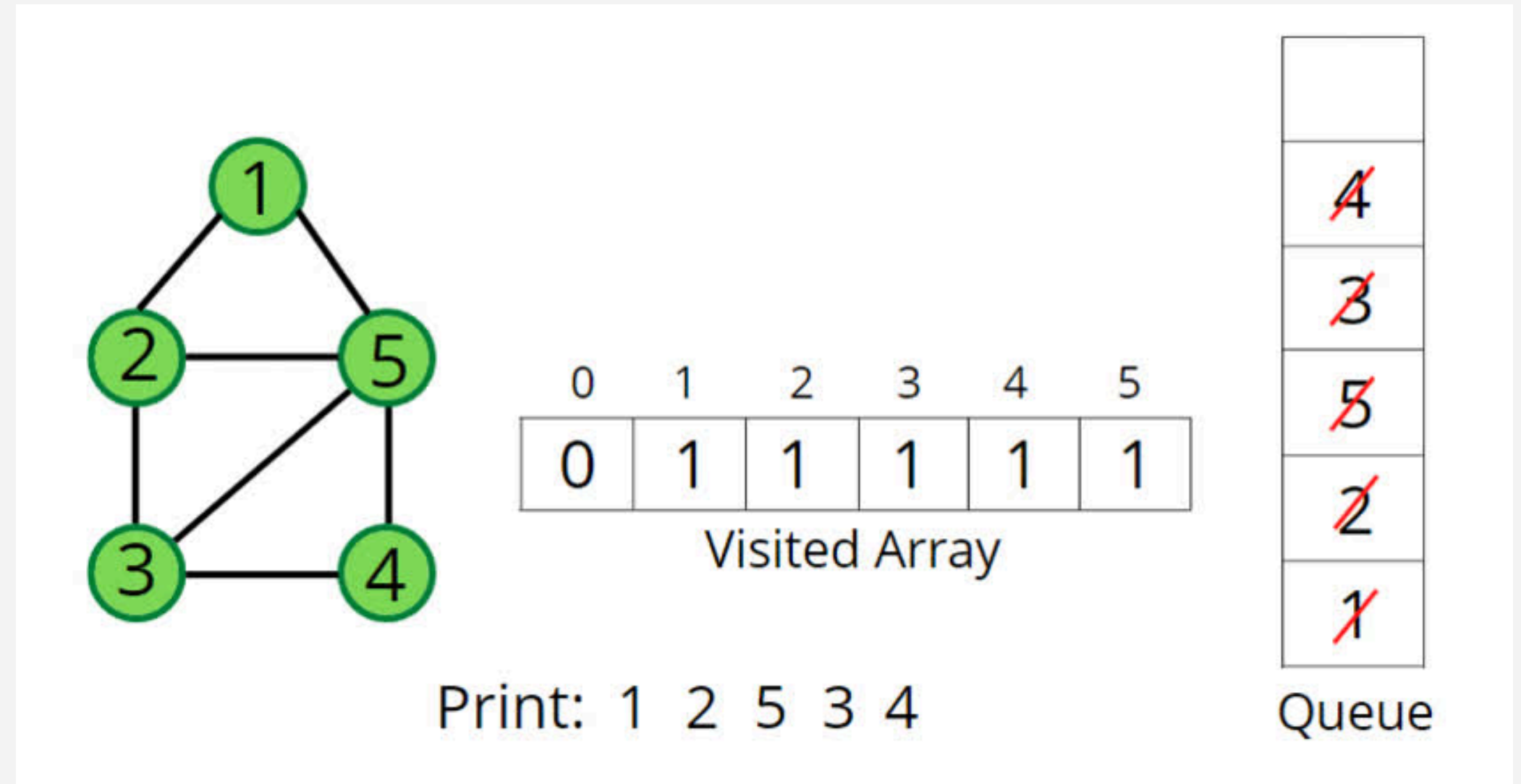
- *Queue()* para guardar nodos pendientes.
- *visited* para no repetir nodos.
- *order* para guardar el orden de visita.

## BFS\_PARENTS

Además guardo:

- `parents[vecino] = nodo_actual`

Sirve para reconstruir caminos.



# implementación

# pruebas realizadas

Se realizaron tres pruebas principales:

Se agregaron elementos:  
A, B, C, D, E

## SE VERIFICÓ:

- tamaño
- primer elemento
- eliminación correcta

```
... ¿Vacía?: True | Size: 0 | First: None | Dequeue: None
```

```
===== Después de encolar A, B, C, D, E =====
```

```
Size: 5 | First: A | Size: 5
```

```
=====
```

```
===== Desencolar A =====
```

```
Size: 4 | First: B | Size: 4
```

# prueba 1: grafo conexo

Se creó un grafo donde todos los nodos están conectados.

Ejemplo de conexiones:

A - B

A - C

B - C

B - D

C - D

Resultado:

El BFS recorre todos los nodos del grafo comenzando desde A.

...

## PRUEBA 1: BFS EN GRAFO CONEXO

Grafo 1:

A: [B, C]

B: [A, C, D]

C: [A, B, D]

D: [B, C]

BFS(A): ['A', 'B', 'C', 'D']

Parents(A): {'A': None, 'B': 'A', 'C': 'A', 'D': 'B'}

Path A -> D: ['A', 'B', 'D']



# prueba 2: grafo con dos componentes

Se creó un grafo con dos grupos separados.

Componente 1:

A - B

A - C

Componente 2:

D - E

E - F

Resultados:

BFS(A) recorre solo A, B, C

BFS(D) recorre solo D, E, F

Esto demuestra que BFS solo visita nodos conectados.

```
=====
PRUEBA 2: GRAFO CON DOS COMPONENTES
=====

Grafo 2:
A: [B, C]
B: [A]
C: [A]
D: [E]
E: [D, F]
F: [E]

BFS(A):  ['A', 'B', 'C']

BFS(D):  ['D', 'E', 'F']
```

# prueba 3: rutas no ponderadas

Se probaron dos casos.  
Caso 1 — Sí existe camino  
 $A \rightarrow C$

Resultado:  
El algoritmo encuentra el camino.

Caso 2 — No existe camino  
 $A \rightarrow E$

Resultado:  
El algoritmo devuelve None.

```
=====
PRUEBA 3: CRUTAS NO PONDERADAS
=====

===== CON CAMINO =====

A->C:  ['A', 'C']

===== SIN CAMINO =====

A->E:  None
```

bfs project

# Conclusión

**Este proyecto permitió comprender:**

- Cómo implementar una cola desde cero
- Cómo representar grafos con estructuras de datos
- Cómo funciona el algoritmo BFS
- Cómo encontrar rutas en grafos no ponderados

bfs project