CARLOS LLANES, MICHELLE CÁMARA, MARCELO MEDINA

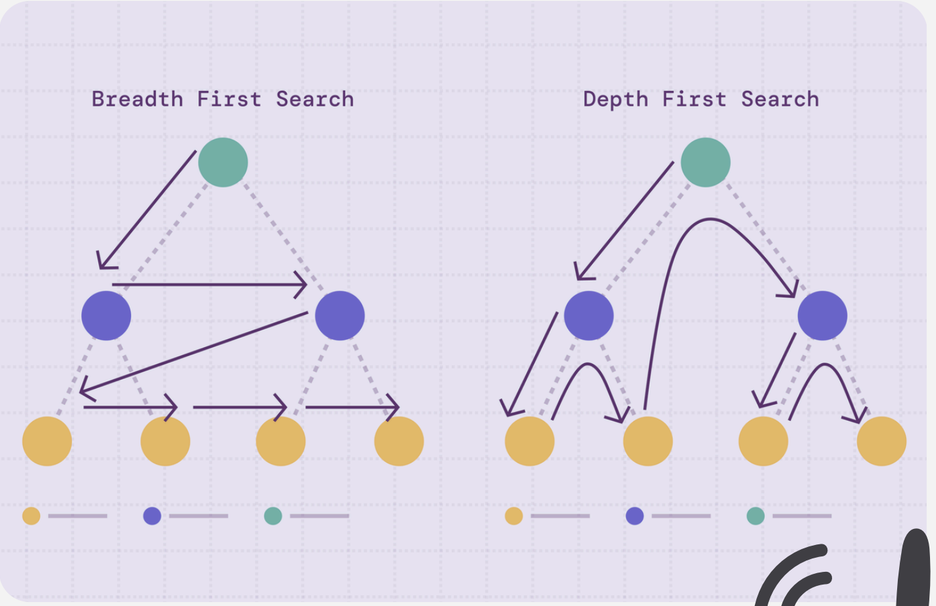
2ºC

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.

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

qué es un cola

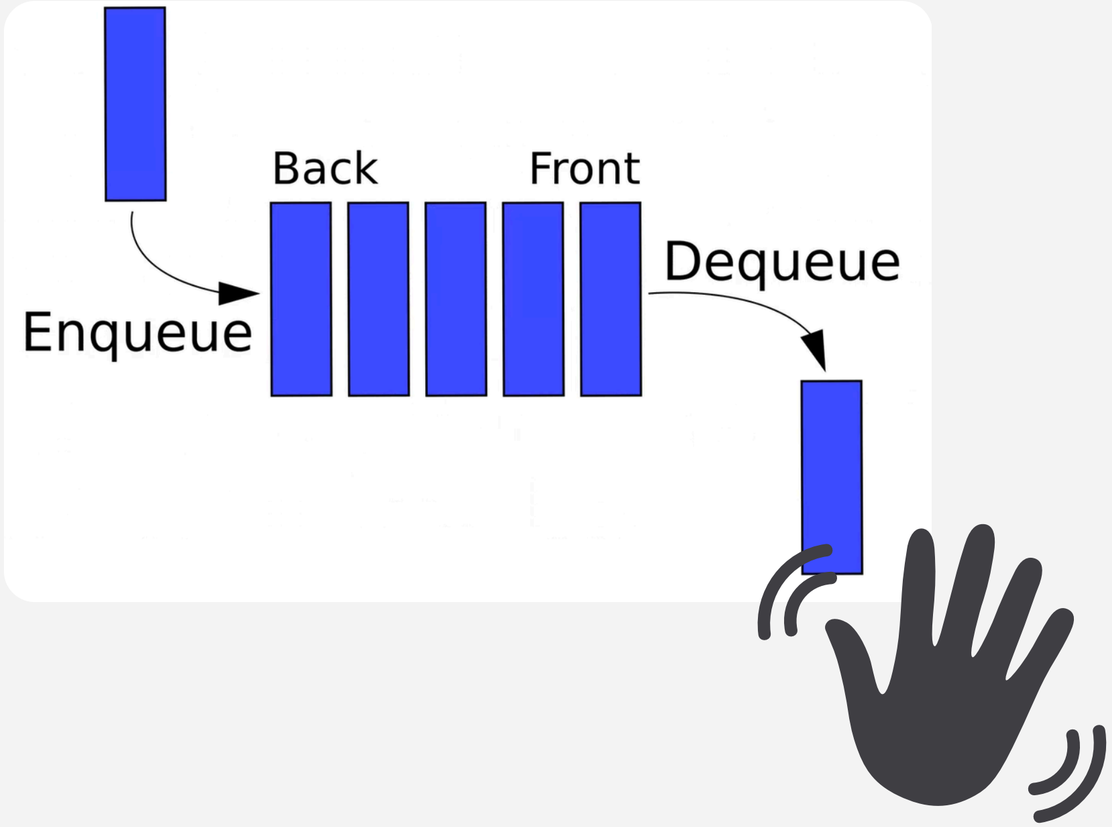
Una cola es una estrucura de datos lineal que sigue el principio FIFO, donde se añaden por un extremo y se eliminan por el otro.

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:

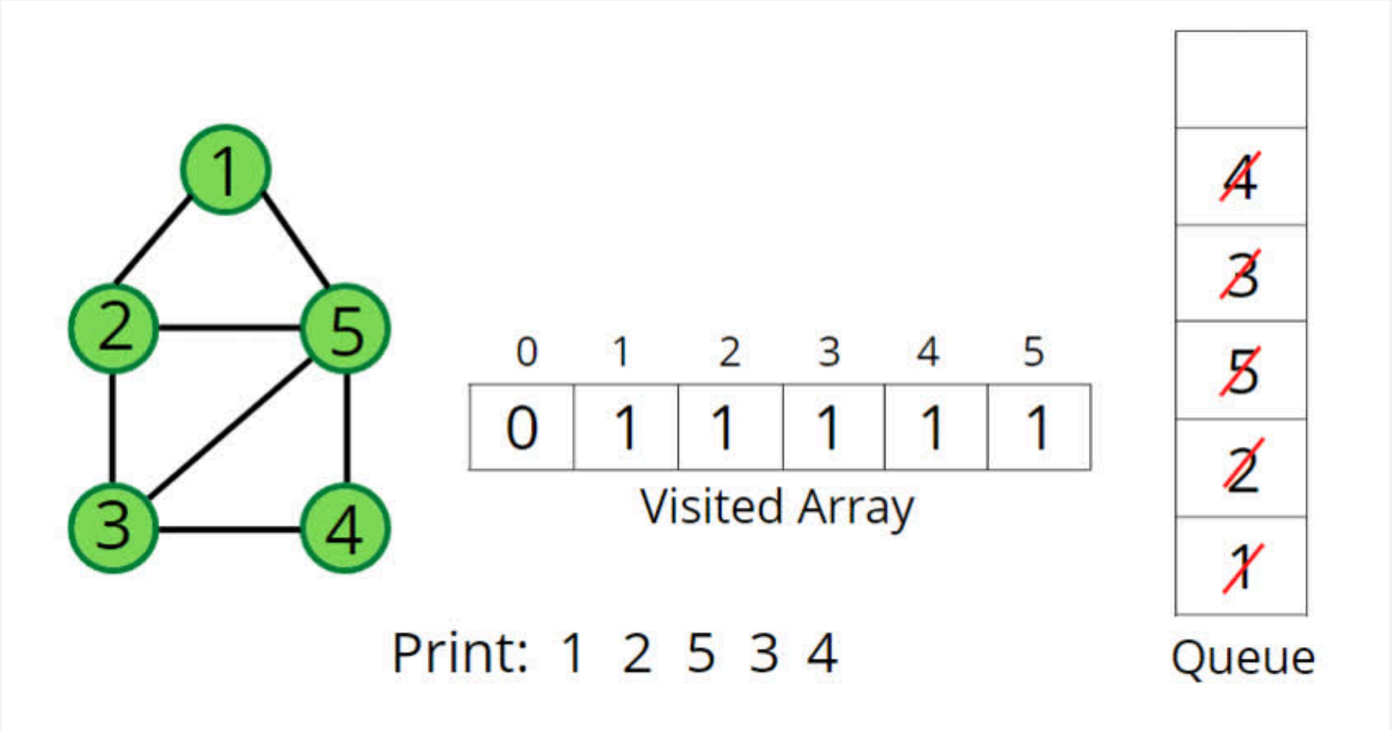
- *Queue()* para guardar nodos pendientes.
- *visited* para no repetir nodos.
- *order* para guardar el orden de visita.

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:
1-2
1-3
2-4
2-5
3-6
4-7

Resultado:
El BFS recorre todos los nodos del grafo comenzando desde 1

```
...  ========================================
           PRUEBA 1: BFS EN GRAFO CONEXO
     ========================================
     Grafo 1:
     1: [2, 3]
     2: [1, 4, 5]
     3: [1, 6]
     4: [2, 7]
     5: [2]
     6: [3]
     7: [4]

     BFS(1):  [1, 2, 3, 4, 5, 6, 7]

     Parents(1):  {1: None, 2: 1, 3: 1, 4: 2, 5: 2, 6: 3, 7: 4}

     Path 1 -> 7:  [1, 2, 4, 7]
```

prueba 2: grafo con dos componentes

Se creó un grafo con dos grupos separados.

Componente 1:
1-2
2-3

Componente 2:
10-20
30-40

Resultados:
BFS(1) recorre solo 1,2,4
BFS(10) recorre solo 10,20,30

```
...  ============================================
           PRUEBA 2: GRAFO CON DOS COMPONENTES
     ============================================
     Grafo 2:
     1: [2]
     2: [1, 3]
     3: [2]
     10: [20]
     20: [10, 30]
     30: [20]

     BFS(1):  [1, 2, 3]

     BFS(10):  [10, 20, 30]
```

Esto demuestra que BFS solo visita nodos conectados.

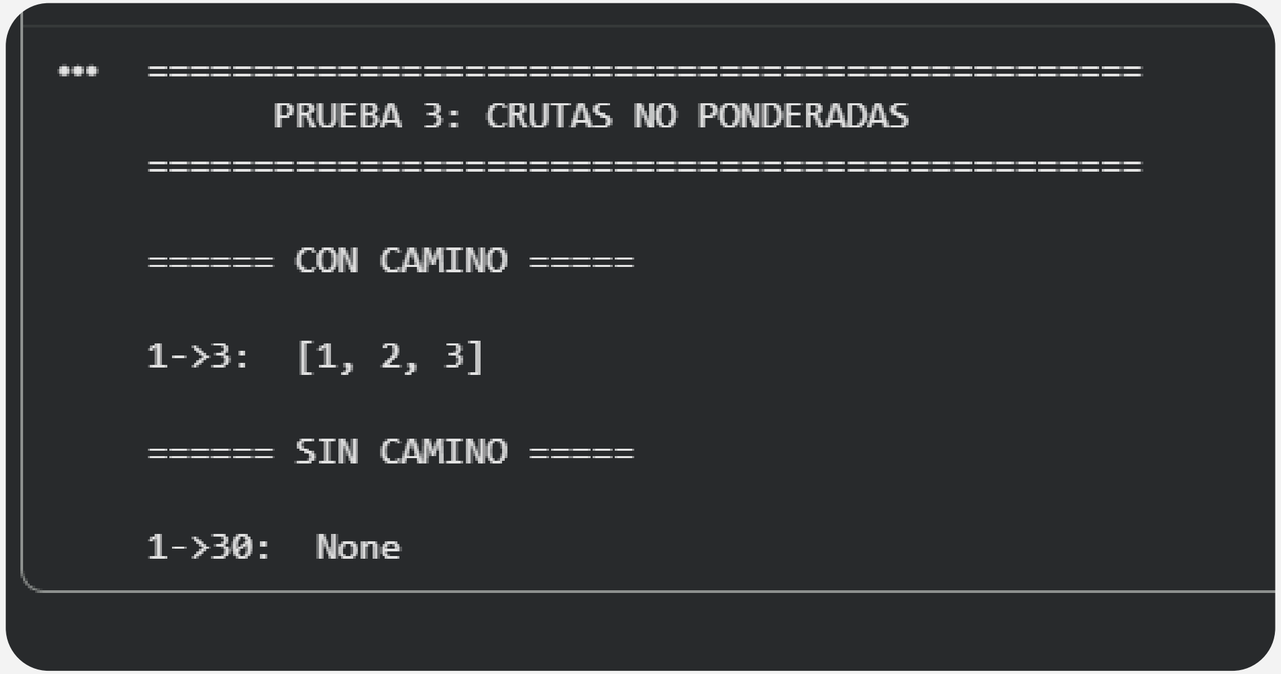
prueba 3: rutas no ponderadas

Se probaron dos casos.
Caso 1 — Sí existe camino
1 → 3

Resultado:
El algoritmo encuentra el camino.

Caso 2 — No existe camino
1 → 30

Resultado:
El algoritmo devuelve None.



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