Puzle para niños

Probablemente conozcan este tipo de puzle:

Dada una instancia, ¿cuáles son los pasos que llevan a la solución?

Formalmente: El “puzle de 15”

Este puzle consta de 15 piezas con números en una grilla de 4×4

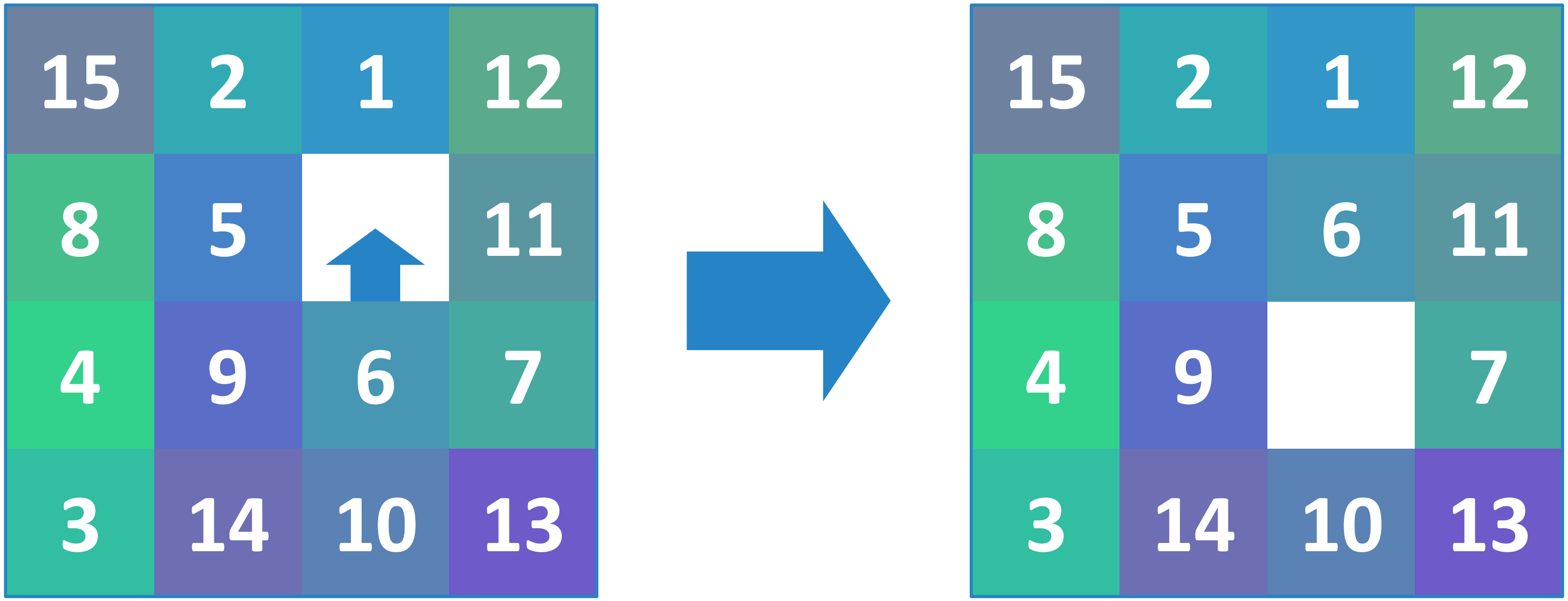
15	2	1	12
8	5		11
4	9	6	7
3	14	10	13

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

La idea es deslizar las piezas hasta dejar los números en orden

Las cuatro operaciones posibles

1. Deslizar hacia arriba



Las cuatro operaciones ...

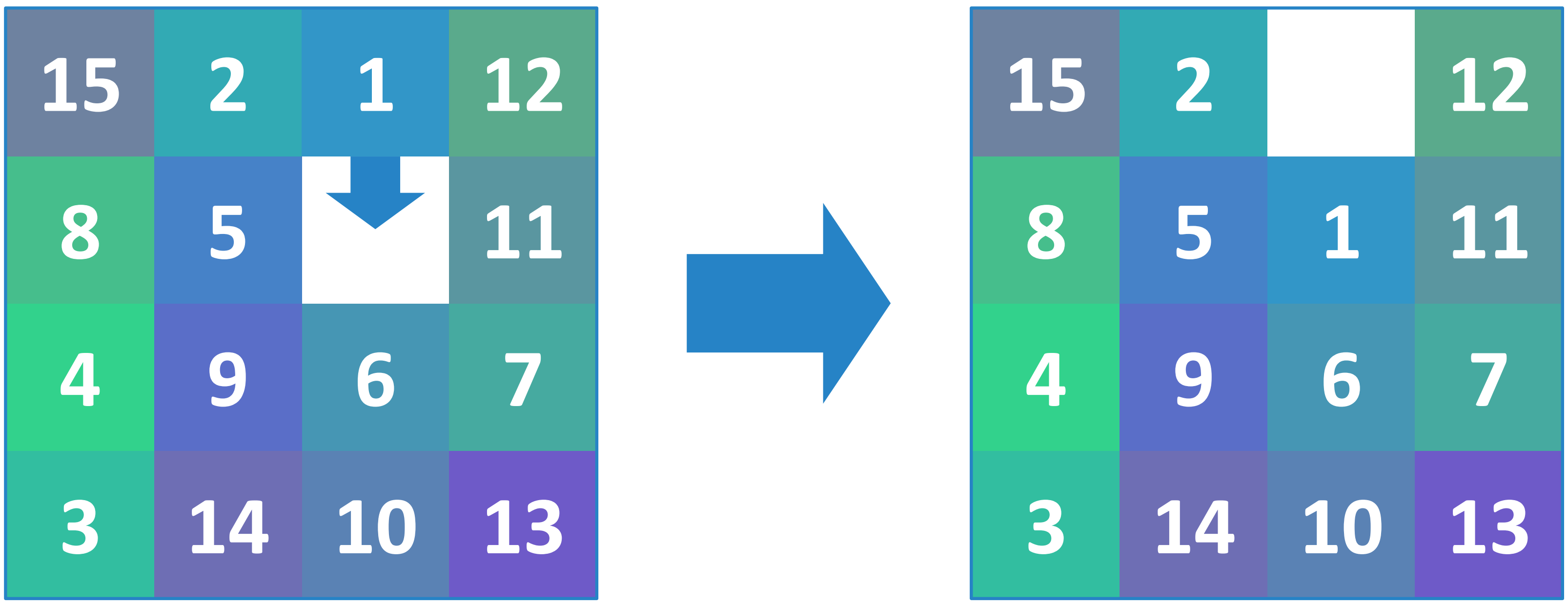
2. Deslizar hacia la derecha

15	2	1	12
8	5		11
4	9	6	7
3	14	10	13

15	2	1	12
8		5	11
4	9	6	7
3	14	10	13

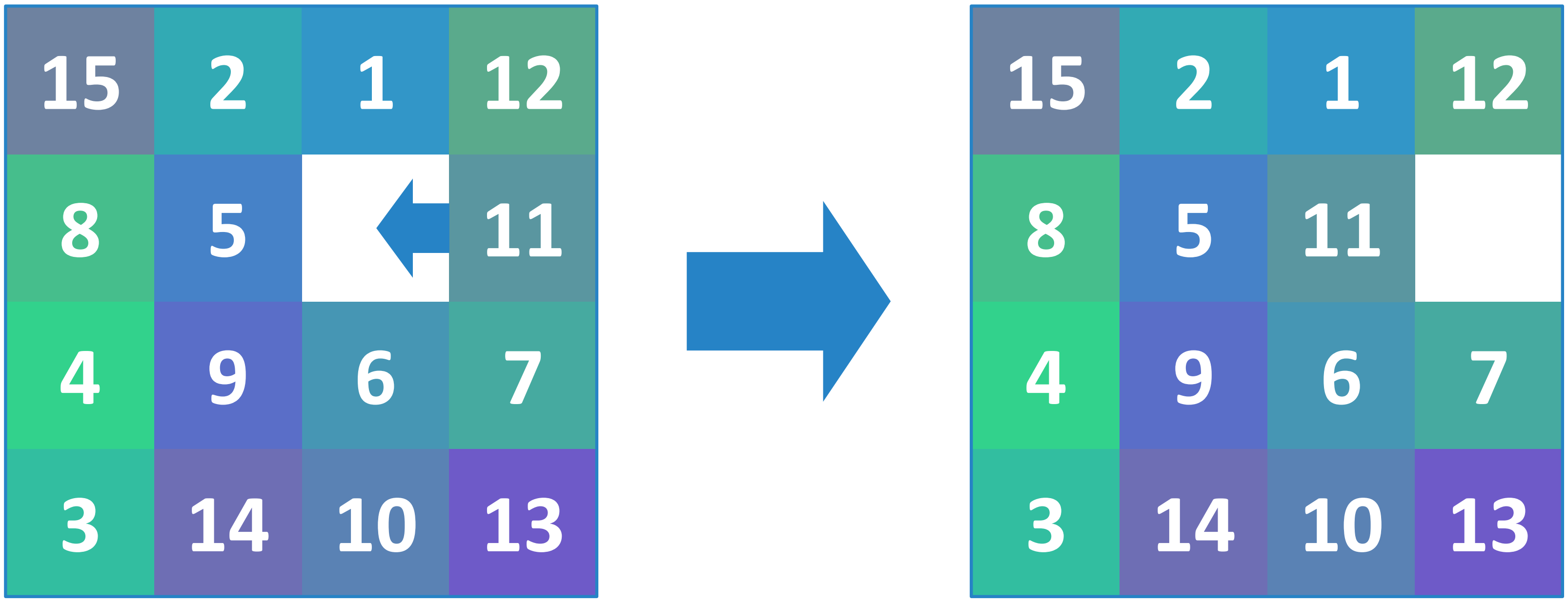
Las cuatro …

3. Deslizar hacia abajo



Las ...

4. Deslizar hacia la izquierda



Entonces … ¿cómo lo resolvemos?

Planteamiento del problema como un problema de búsqueda en un grafo

Podríamos hacer lo siguiente:

1. Construir un **grafo** que represente el problema
2. Utilizar **DFS** para **buscar** el camino a la solución

¿Cómo hacemos esto?

Primero, el paso 1

Podríamos hacer lo siguiente:

1. Construir un **grafo** que represente el problema
2. Utilizar **DFS** para **buscar** el camino a la solución

¿Cómo hacemos esto?

Grafo de estados y sus transiciones

Un **grafo de estados** $G(V,E)$ se define de la siguiente manera:

Cada nodo en V es una **configuración** (estado) distinta del problema

Hay una arista (hay una **transición**) de u a v si se puede pasar de u a v en un paso.

En el caso del puzle de 15

15	2		12
8	5	1	11
4	9	6	7
3	14	10	13

15	2	1	12
8		5	11
4	9	6	7
3	14	10	13

15	2	1	12
8	5		11
4	9	6	7
3	14	10	13

15	2	1	12
8	5	6	11
4	9		7
3	14	10	13

15	2	1	12
8	5	11	
4	9	6	7
3	14	10	13

Las **operaciones** definen las conexiones entre estados

¡Cuidado con el uso de memoria!

¿Qué tamaño tiene el grafo de estados del puzle de 15?

¿Hay algún problema con eso?

Diccionarios al rescate

Los problemas de este tipo suelen tener *muchos* estados

Hay que generar el grafo a medida que se exploran los estados

Se necesita un **diccionario** para no generar estados repetidos

Veamos ahora el paso 2

Podríamos hacer lo siguiente:

1. Construir un **grafo** que represente el problema
2. Utilizar **DFS** para **buscar** el camino a la solución

¿Cómo hacemos esto?

$$\boldsymbol{buscar\ dfs}(\boldsymbol{D}, \boldsymbol{s}, \boldsymbol{g}):$$

$$\boldsymbol{if\ s \in D,\ return\ false}$$

Insertar $\boldsymbol{s}$ en $\boldsymbol{D}$

$$\boldsymbol{if\ s = g,\ return\ true}$$

$$\boldsymbol{foreach\ operation\ op}:$$

$$\boldsymbol{t \leftarrow op(s)}$$

$$\boldsymbol{t.parent \leftarrow s, \quad t.operation \leftarrow op}$$

$$\boldsymbol{if\ buscar\ dfs(D, t, g)}:$$

$$\boldsymbol{return\ true}$$

$$\boldsymbol{return\ false}$$

El "puzle de 15++"

Dada una configuración del puzle de 15

… ¿cuáles son los pasos necesarios para llegar a la solución

… **de modo que la ruta desde la partida a la solución sea la más corta posible**?

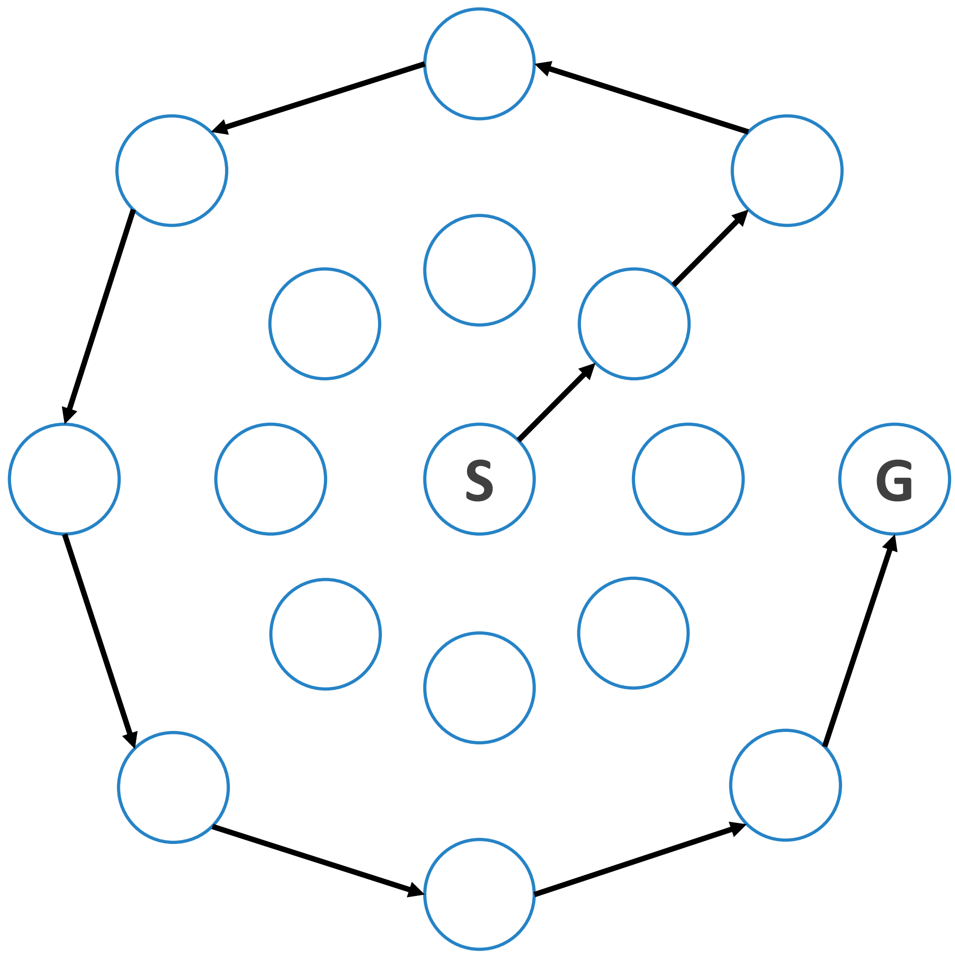
Ruta más corta

Si la **distancia** entre dos nodos es el largo de la **ruta más corta** entre ellos,

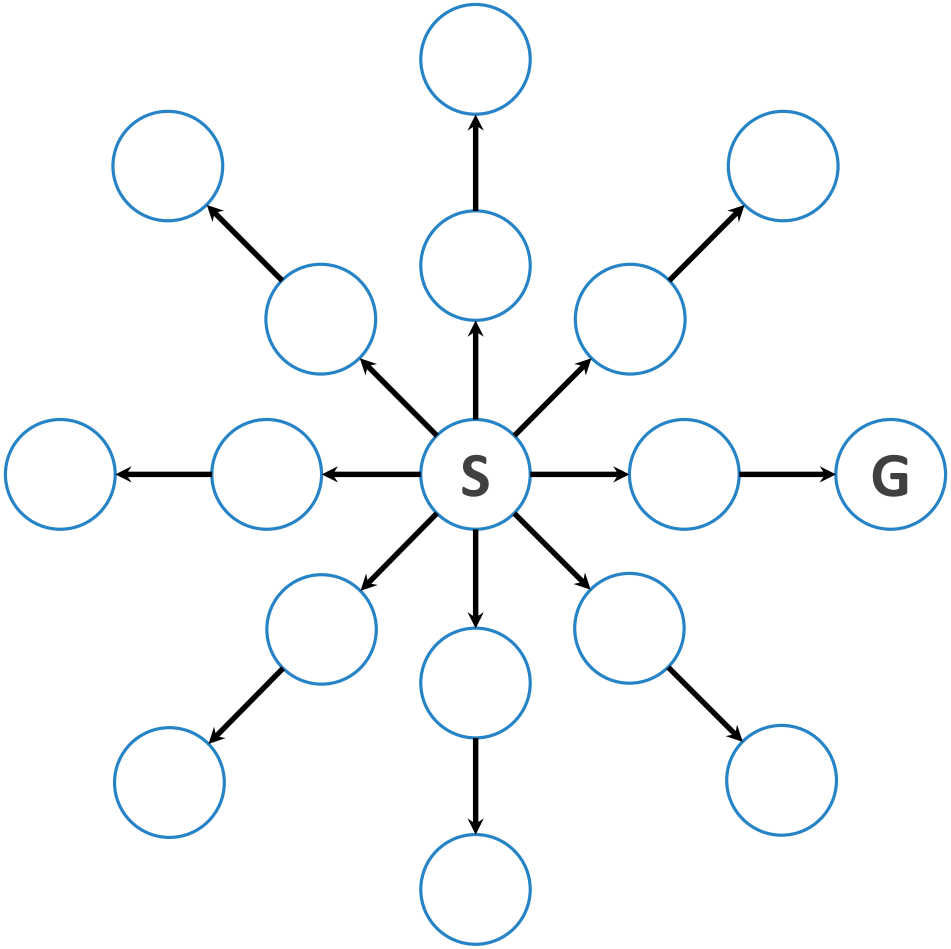
… ¿está la solución a distancia 1 del nodo de partida, u origen? ¿y a distancia 2?

¿Cómo podemos responder esa pregunta para una distancia n?

Tenemos *depth first search*



… y queremos *breadth first search*



La idea del algoritmo de BFS

Partiendo de $i = 1$:

1. Generar los estados a distancia i del origen
2. Si alguno de esos es el destino, estamos listos
3. Si no, incrementar i en 1 y volver a 1.

¿Cómo hacemos esto eficientemente?

$buscar\ bfs(\boldsymbol{D}, \boldsymbol{s}, \boldsymbol{g})$:

$\boldsymbol{Open} \leftarrow$ una cola vacía. $\boldsymbol{D} \leftarrow$ un diccionario vacío.

Insertar $\boldsymbol{s}$ en $\boldsymbol{Open}$ y en $\boldsymbol{D}$

$\boldsymbol{while}\ \boldsymbol{Open} \neq \emptyset$:

$s \leftarrow$ el siguiente elemento de $\boldsymbol{Open}$

$\boldsymbol{foreach}\ \boldsymbol{operation}\ \boldsymbol{op}$:

$\boldsymbol{t} \leftarrow \boldsymbol{op}(\boldsymbol{s})$

$\boldsymbol{t.parent} \leftarrow \boldsymbol{s}, \quad \boldsymbol{t.operation} \leftarrow \boldsymbol{op}$

$\boldsymbol{if}\ \boldsymbol{t} = \boldsymbol{g},\ \boldsymbol{return\ true}$

$\boldsymbol{if}\ \boldsymbol{t} \notin \boldsymbol{D}$, Insertar $\boldsymbol{t}$ en $\mathbf{D}$ y en $\boldsymbol{Open}$

$\boldsymbol{return\ false}$

Relación entre DFS y BFS

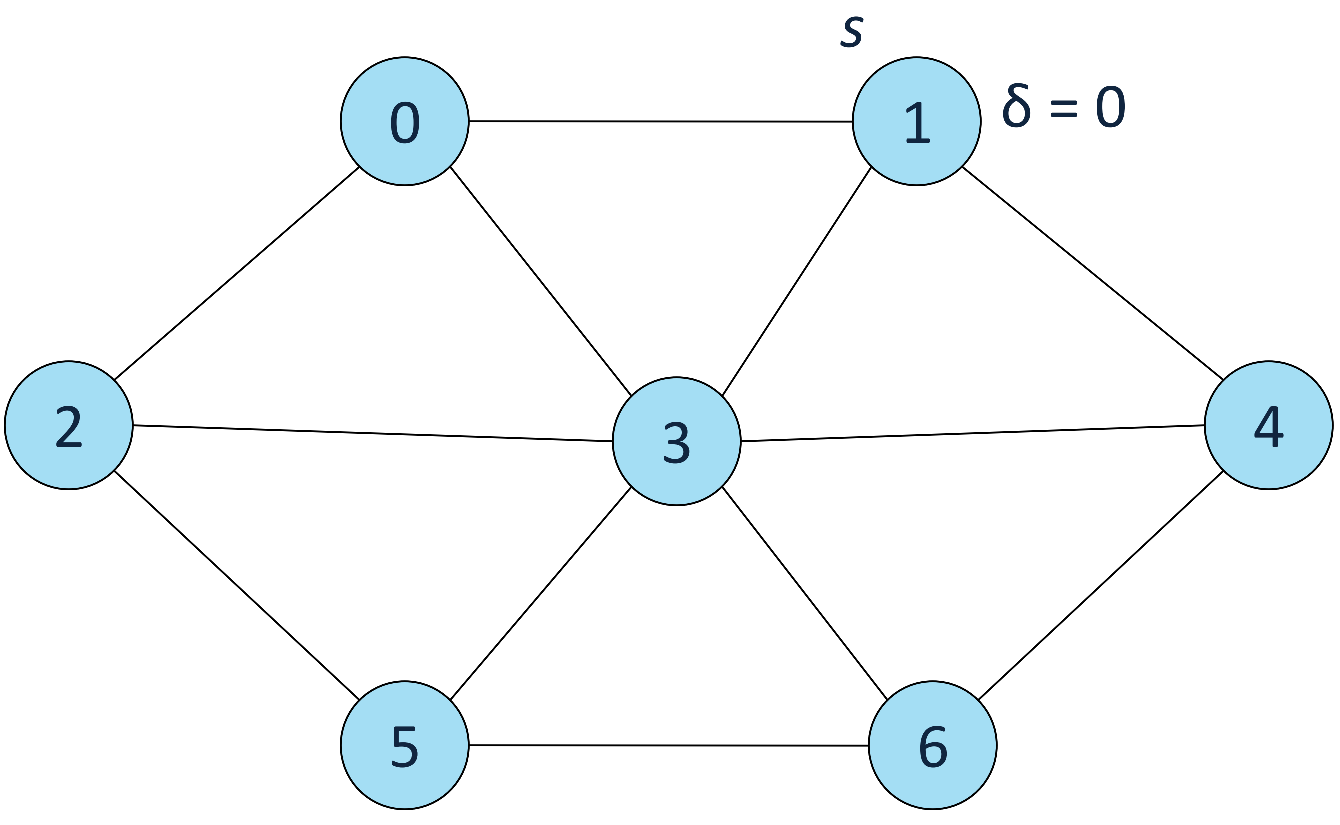
Si reemplazamos la **cola** en **BFS** por un **stack**, tenemos **DFS**

Sería una forma de implementar **DFS** de manera iterativa

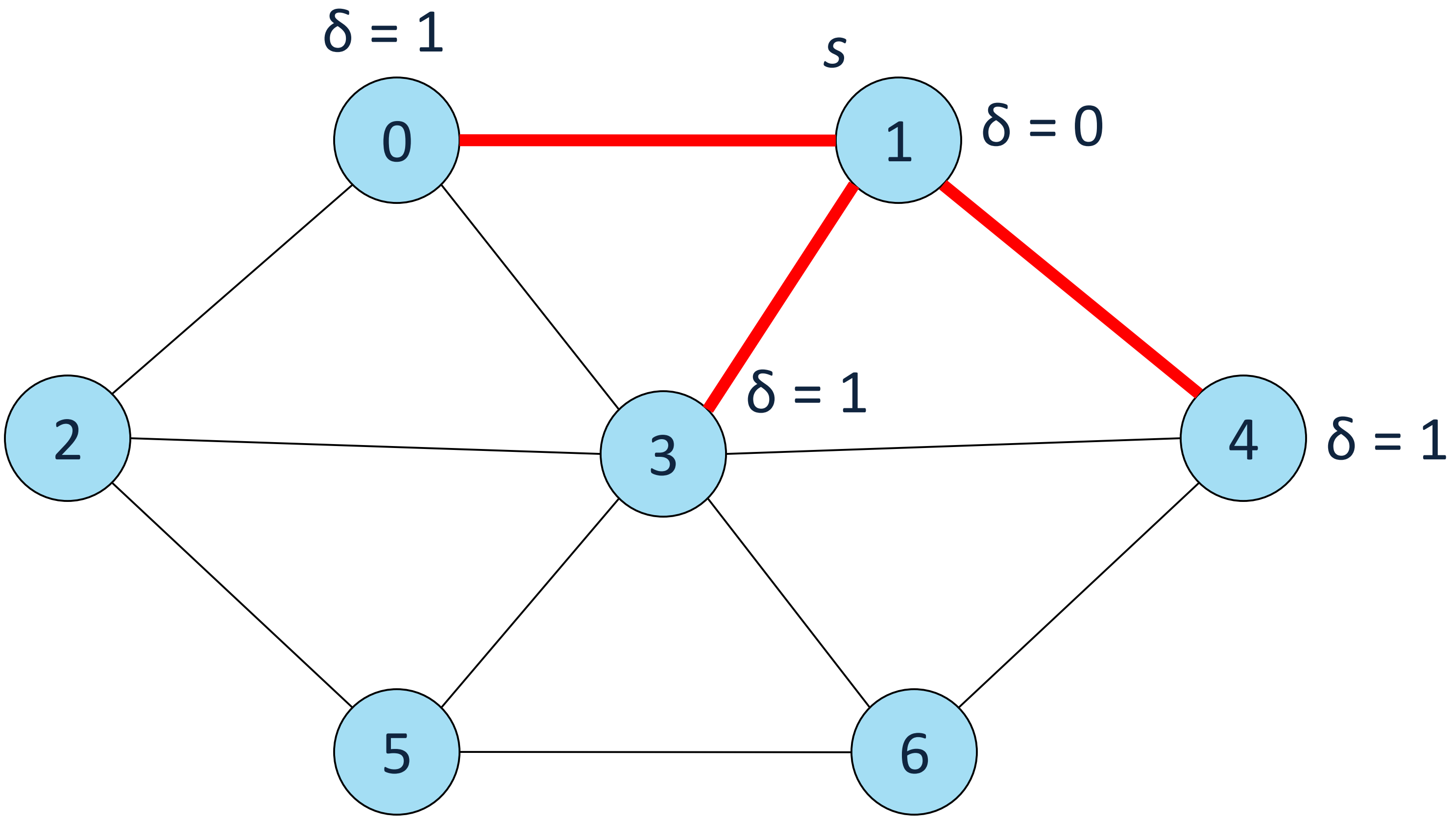
La complejidad de ambos algoritmos es la misma

```
bfs(s): —s es el vértice de partida
  for each u in V−{s}:
     u.color = white; u.δ = ∞; π[u] = null
  s.color = gray; s.δ = 0; π[s] = null
  q = Queue(); q.enqueue(s)
  while !q.empty():
     u = q.dequeue()
     for each v in α[u]:
        if v.color == white:
           v.color = gray; v.δ = u.δ+1
           π[v] = u; q.enqueue(v)
     u.color = black
```

BFS a partir del vértice s = 1:
Vértices a distancia δ = 0 de s



BFS a partir del vértice s = 1:
Vértices a distancias δ = 0 y 1 de s



BFS a partir del vértice s = 1:
Vértices a distancias δ = 0, 1 y 2 de s