Ayudantía 3

Introducción a la búsqueda

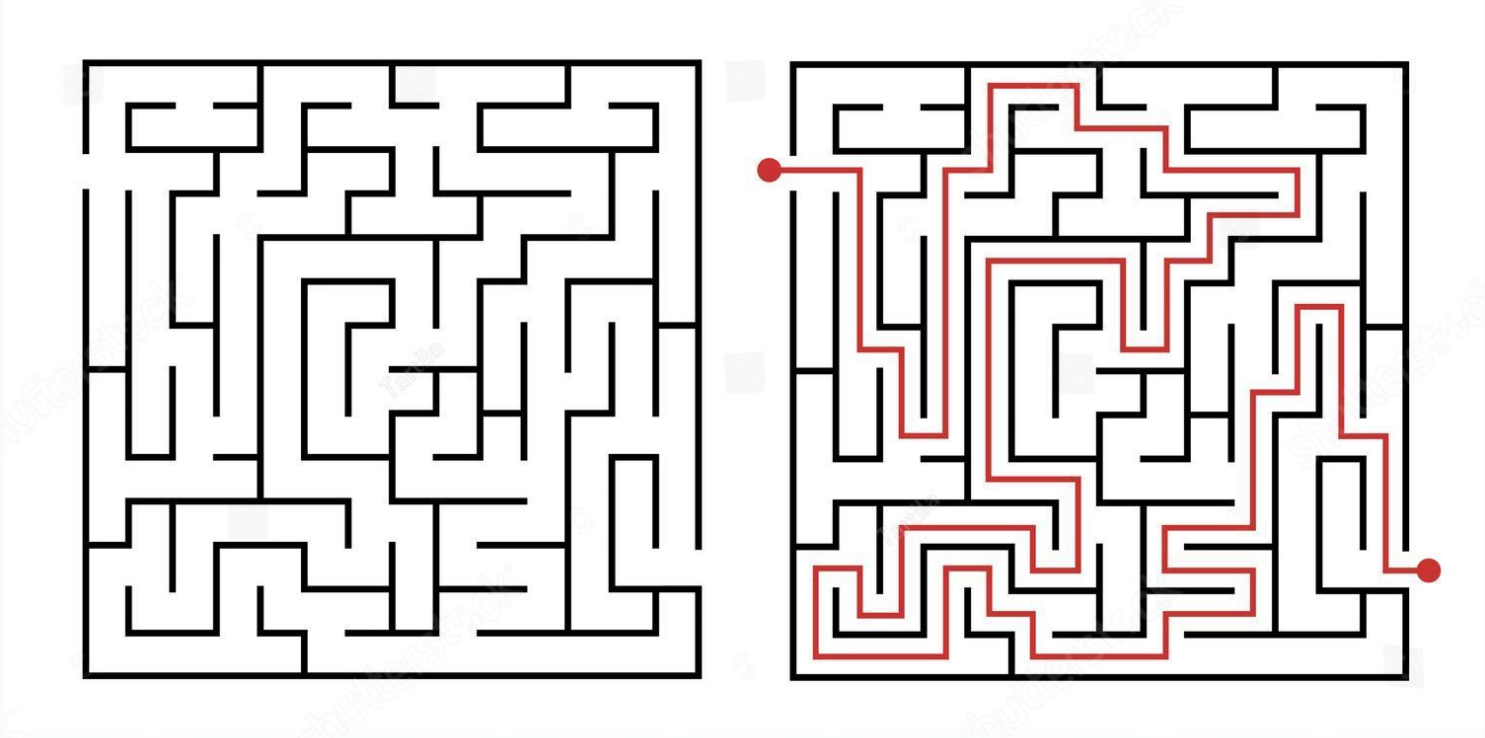
Por Martín Lagies y Felipe Espinoza

08 de abril 2024

Contenidos

1. ¿Qué es búsqueda?
2. Definición formal de un problema de búsqueda
3. Estados y acciones
4. Ejemplo de Puzzle 8
5. Algoritmos de búsqueda
6. Más ejemplos

¿Qué es un problema de búsqueda?



Definición formal de un problema de búsqueda

Mundos Determinísticos con un Agente

Necesitamos...

- Un espacio de **Estados** *S*
- Un conjunto de **Operadores** *A*
- Una **función de costo** *c*
- Un **estado inicial** *s_init*
- Un conjunto de **estados finales** *G*

Un **estado** o **nodo** (*s*) es la **configuración** específica de un sistema

Un **operador**, **acción** o **arco** (*a*) es una función parcial que hace pasar el sistema de un estado (*s*) a otro (*s'*)

Definición formal de un problema de búsqueda

Mundos Determinísticos con un Agente

Necesitamos...

- Un espacio de **Estados** *S*
- Un conjunto de **Operadores** *A*
- Una **función de costo** *c*
- Un **estado inicial** *s_init*
- Un conjunto de **estados finales** *G*

Función parcial que hace pasar el sistema de un estado (*s*) a otro (*s'*)

$$\alpha: S \longrightarrow S$$

Cada estado posee un conjunto $A(s) \subseteq A$ de operadores aplicables en *s*. Si $\alpha \in A(s)$, entonces $\alpha(s)$ está definida

Definición formal de un problema de búsqueda

Mundos Determinísticos con un Agente

Necesitamos...

- Un espacio de **Estados** *S*
- Un conjunto de **Operadores** *A*
- Una **función de costo** *c*
- Un **estado inicial** *s_init*
- Un conjunto de **estados finales** *G*

Cada estado posee un conjunto $A(s) \subseteq A$ de operadores aplicables en *s*. Si $a \in A(s)$, entonces $a(s)$ está definida. Definimos:

$$Succ(s) = \{a(s) \mid a \in A(s)\}$$

Definición formal de un problema de búsqueda

Mundos Determinísticos con un Agente

Necesitamos...

- Un espacio de **Estados** *S*
- Un conjunto de **Operadores** *A*
- Una **función de costo** *c*
- Un **estado inicial** *s_init*
- Un conjunto de **estados finales** *G*

Función que determina el costo de pasar de un estado a otro

Es especialmente útil en algoritmos de búsqueda informada

Definición formal de un problema de búsqueda

Mundos Determinísticos con un Agente

Necesitamos...

- Un espacio de **Estados** *S*
- Un conjunto de **Operadores** *A*
- Una **función de costo** *c*
- Un **estado inicial** *s_init*
- Un conjunto de **estados finales** *G*

Configuración inicial en el problema de búsqueda

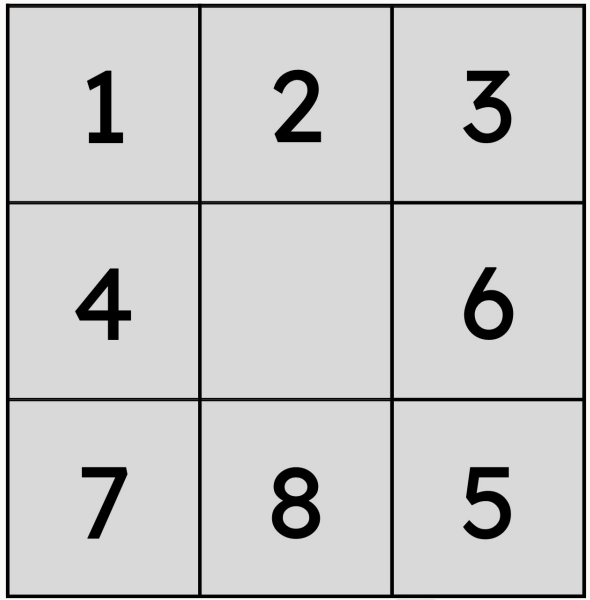
Contiene todas las configuraciones objetivo

Definición formal de un problema de búsqueda

La notación de un problema de búsqueda es:

$$(S, A, s_{init}, G)$$

Ejemplos Típicos



Puzzle 8

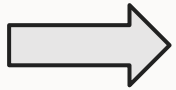
Puzzle 8

Tenemos un problema de búsqueda (S, A, S_{init}, G)

- S = Conjunto de estados
- A = Conjunto de acciones
- S_init = Estado inicial
- G = Conjunto de estados finales

Estado inicial

1	2	3
4		6
7	8	5



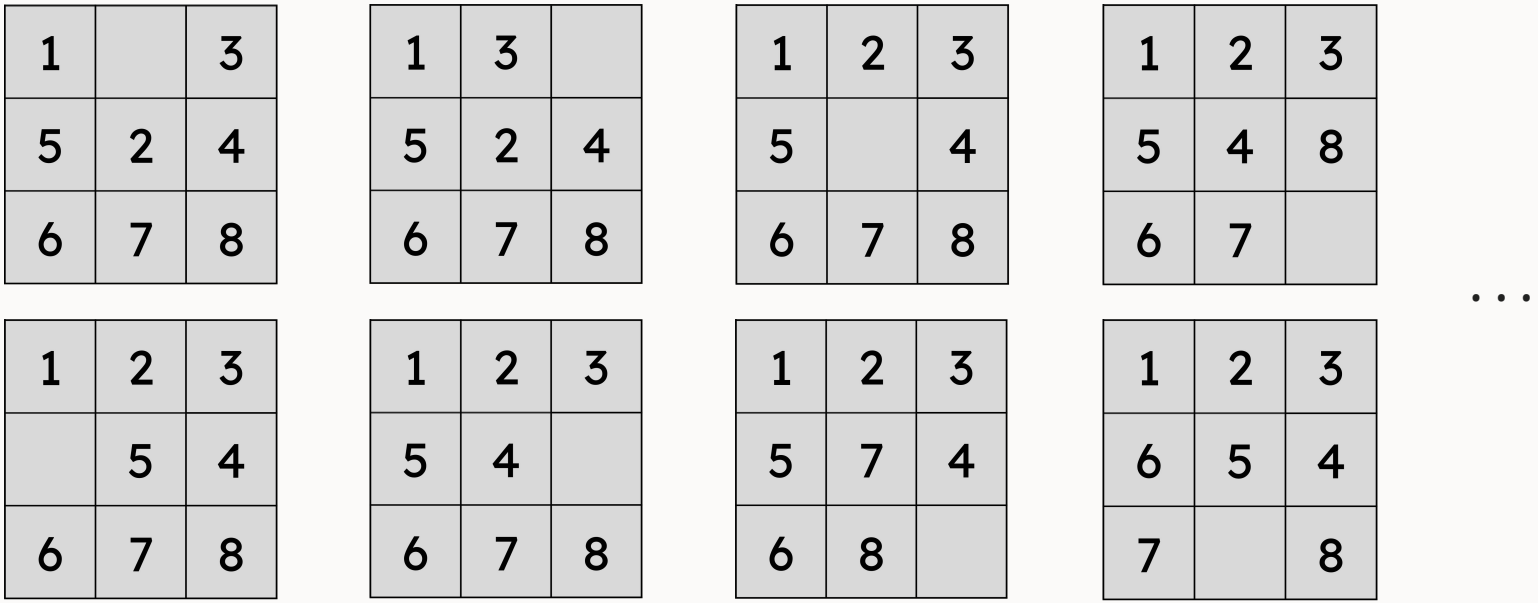
Estado final ∈G

	1	2
3	4	5
6	7	8

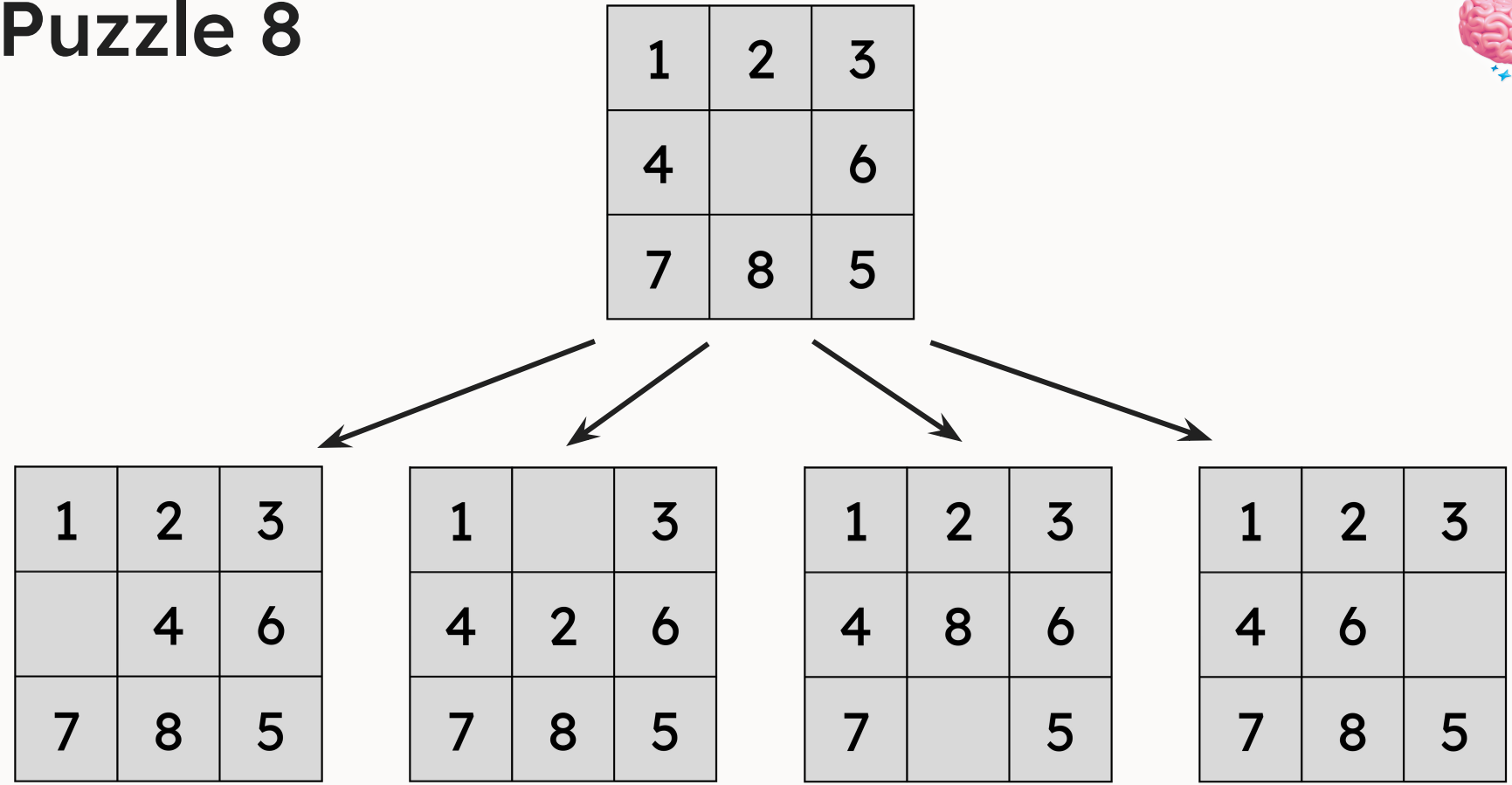
Puzzle 8

A = Conjunto de acciones para cambiar de estados, mover una pieza

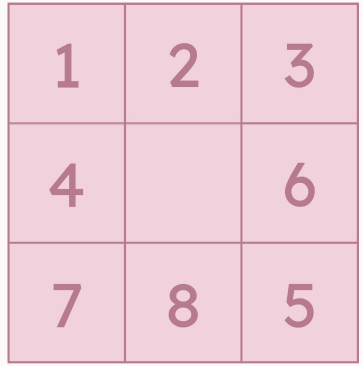
S = Conjunto de estados donde realizamos la búsqueda



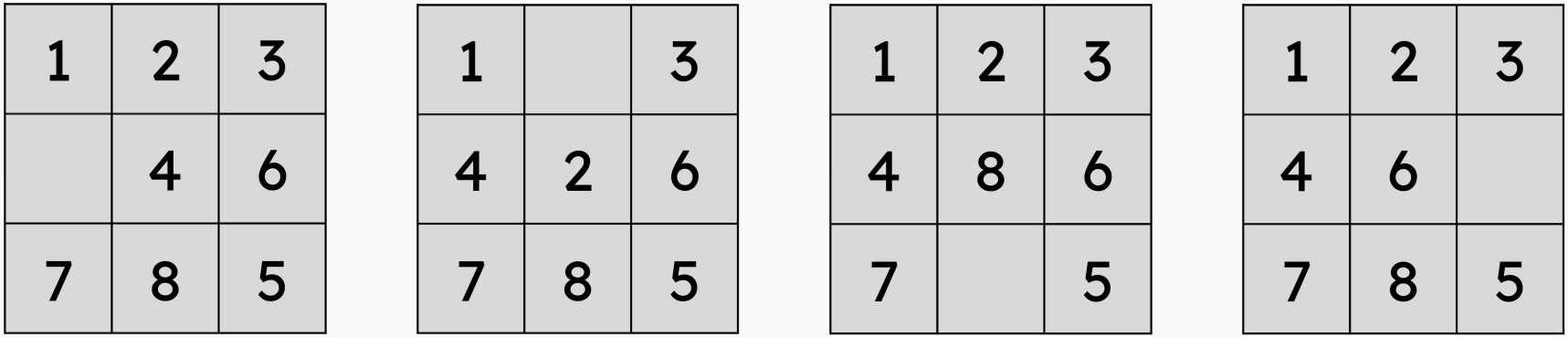
Puzzle 8



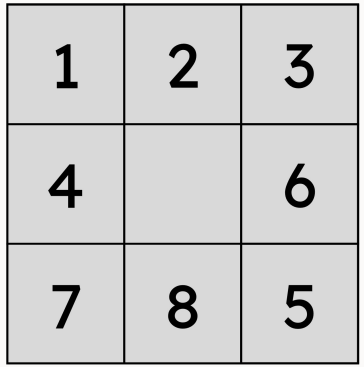
Puzzle 8



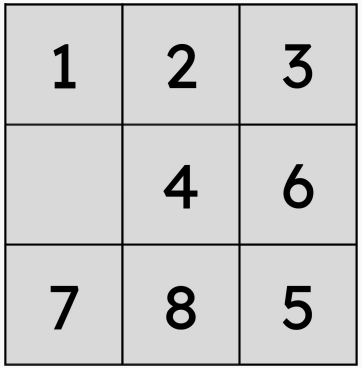
Estado $s \in S$

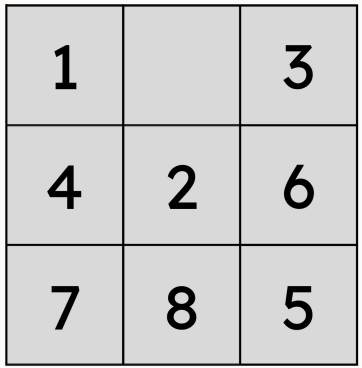


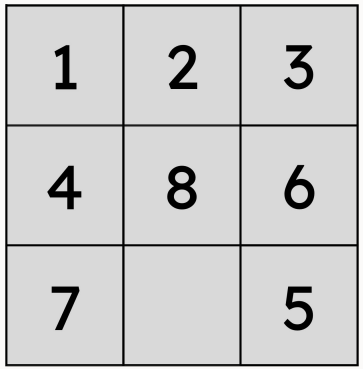
Puzzle 8

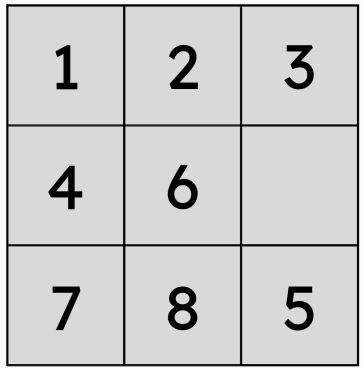


Acciones $a_i \in A$

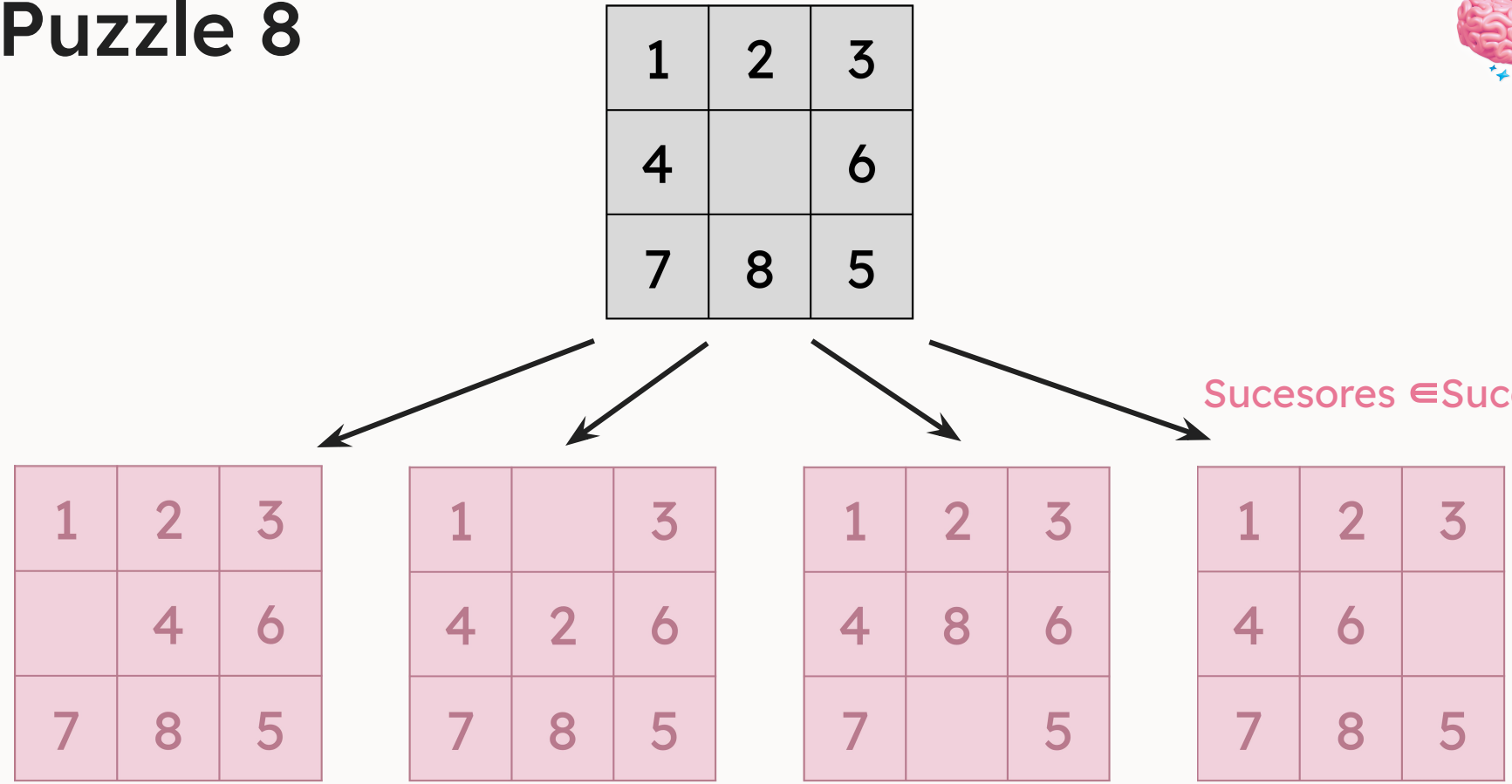




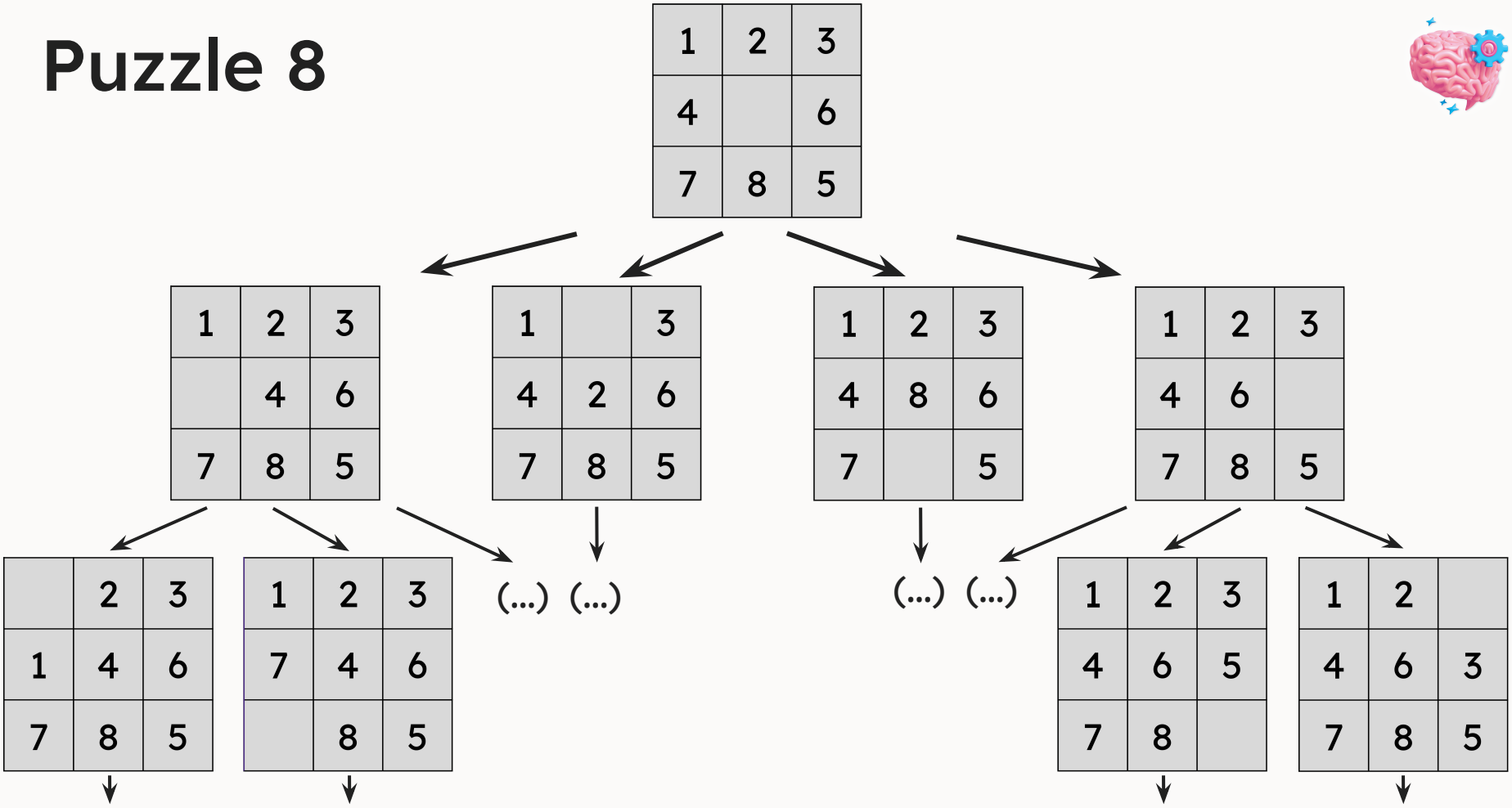




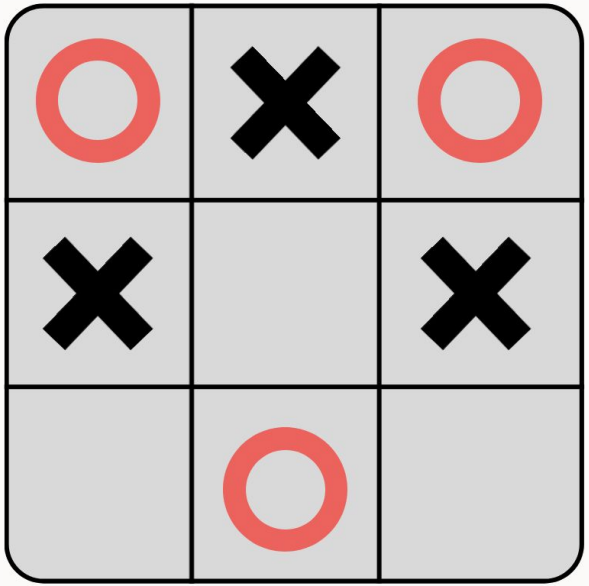
Puzzle 8


1 2 3
4 6
7 8 5
Sucesores ∈Succ(s)
1 2 3
4 6
7 8 5
1 3
4 2 6
7 8 5
1 2 3
4 8 6
7 5
1 2 3
4 6
7 8 5

Puzzle 8


1 2 3
4 6
7 8 5
1 2 3
4 6
7 8 5
1 3
4 2 6
7 8 5
1 2 3
4 8 6
7 5
1 2 3
4 6
7 8 5
2 3
1 4 6
7 8 5
1 2 3
7 4 6
8 5
(...) (...)
(...) (...)
1 2 3
4 6 5
7 8
1 2
4 6 3
7 8 5

Ejemplos Típicos

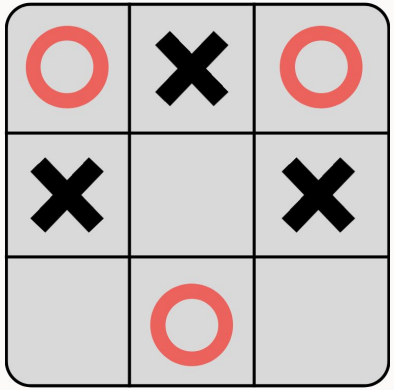


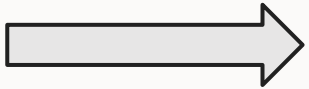
Tic-Tac-Toe / Gato

Tic-Tac-Toe

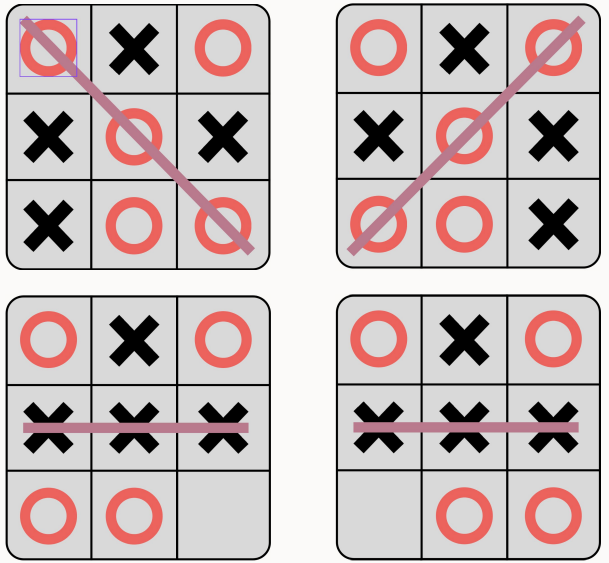
- A = Completar una casilla, siguiendo los turnos

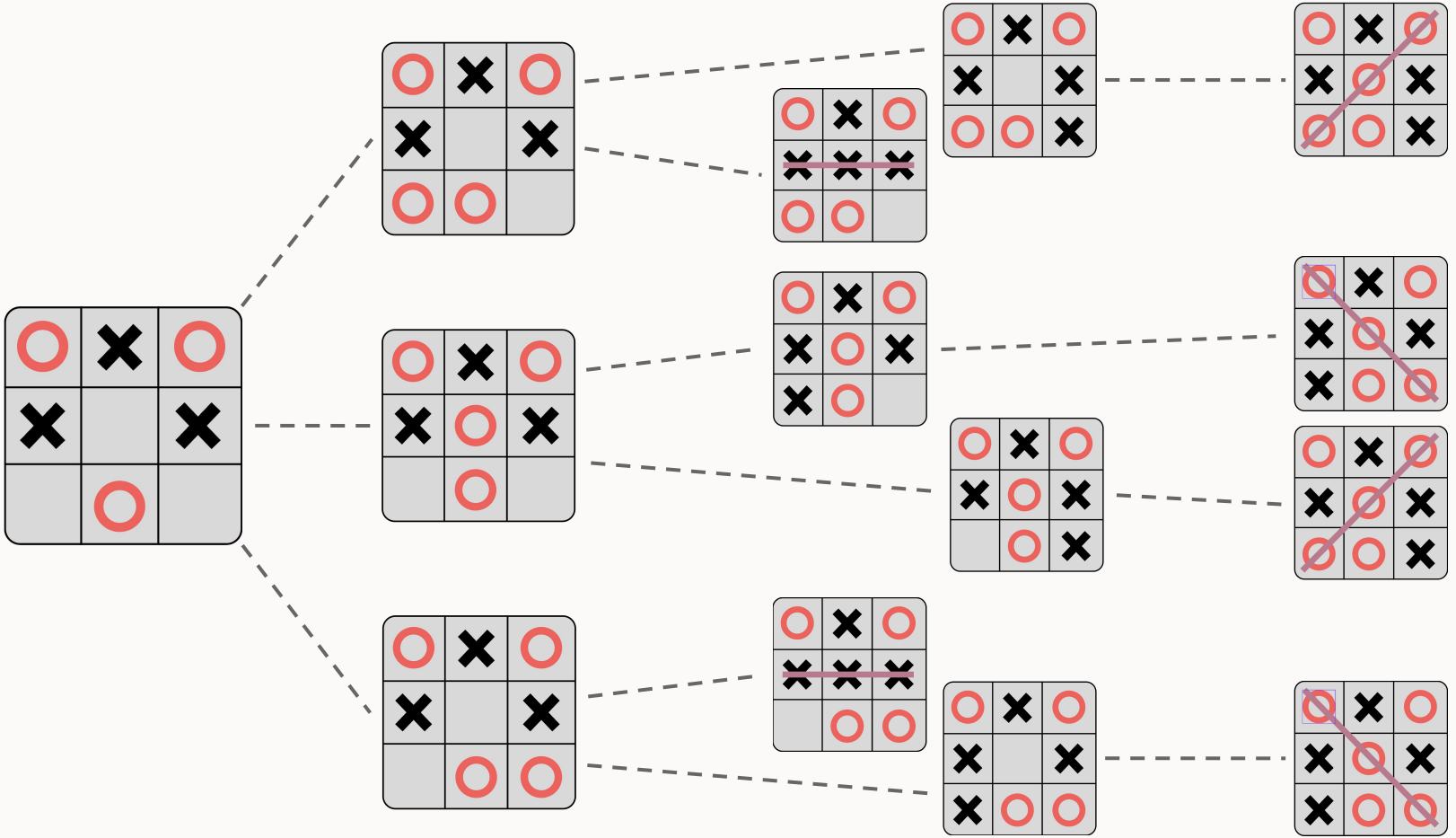
Estado inicial





Estados finales $\in G$





Algoritmos de Búsqueda

Búsqueda Genérica:

Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open $\neq \varnothing$
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

Open (o frontera) contiene todos los nodos generados pero no expandidos

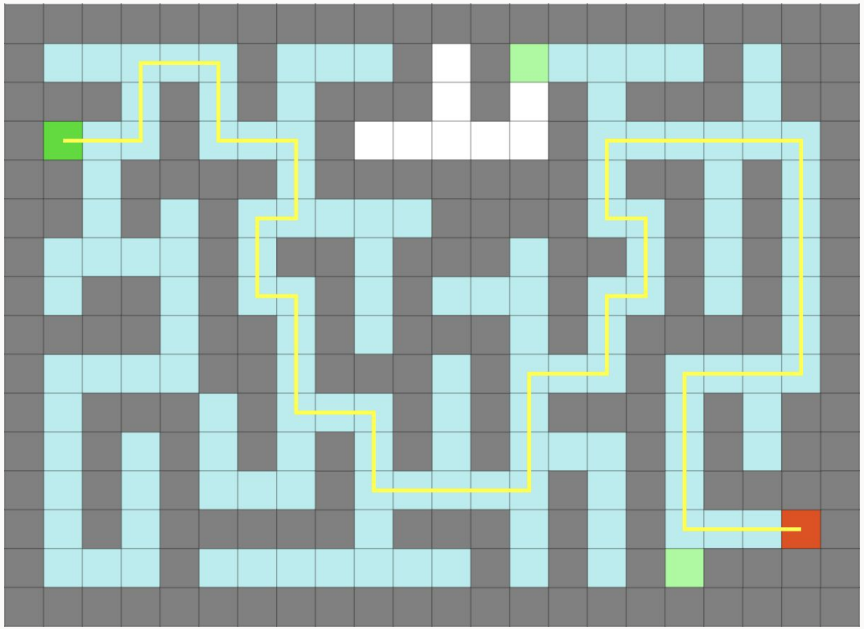
Closed contiene los nodos expandidos

Nodo expandido: sucesores calculados

Algoritmos de Búsqueda

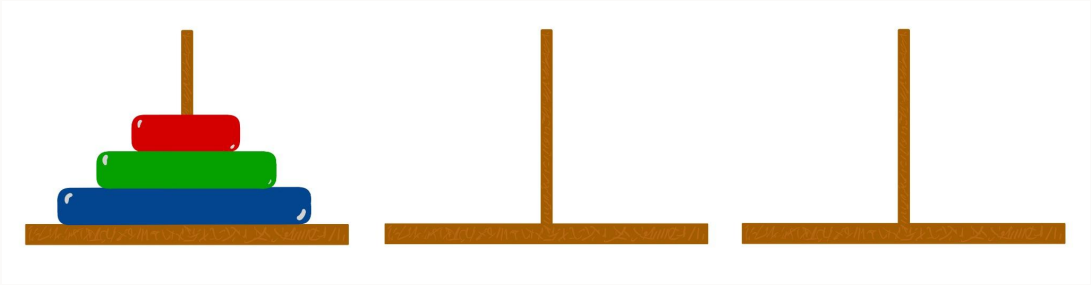
Visualización Dinámica:

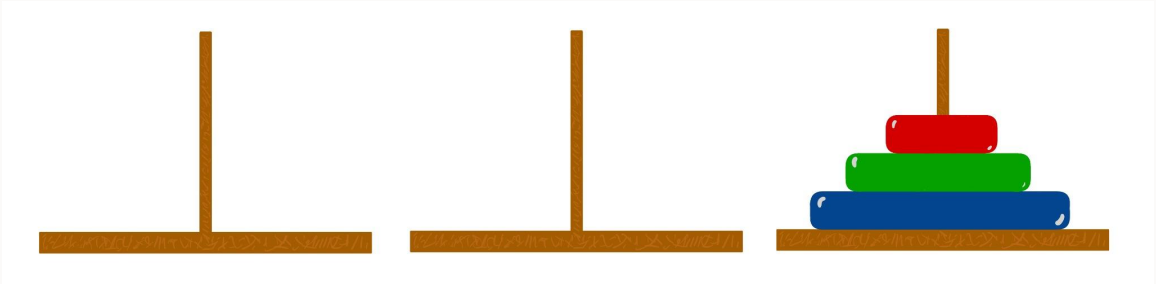
PathFinding.js



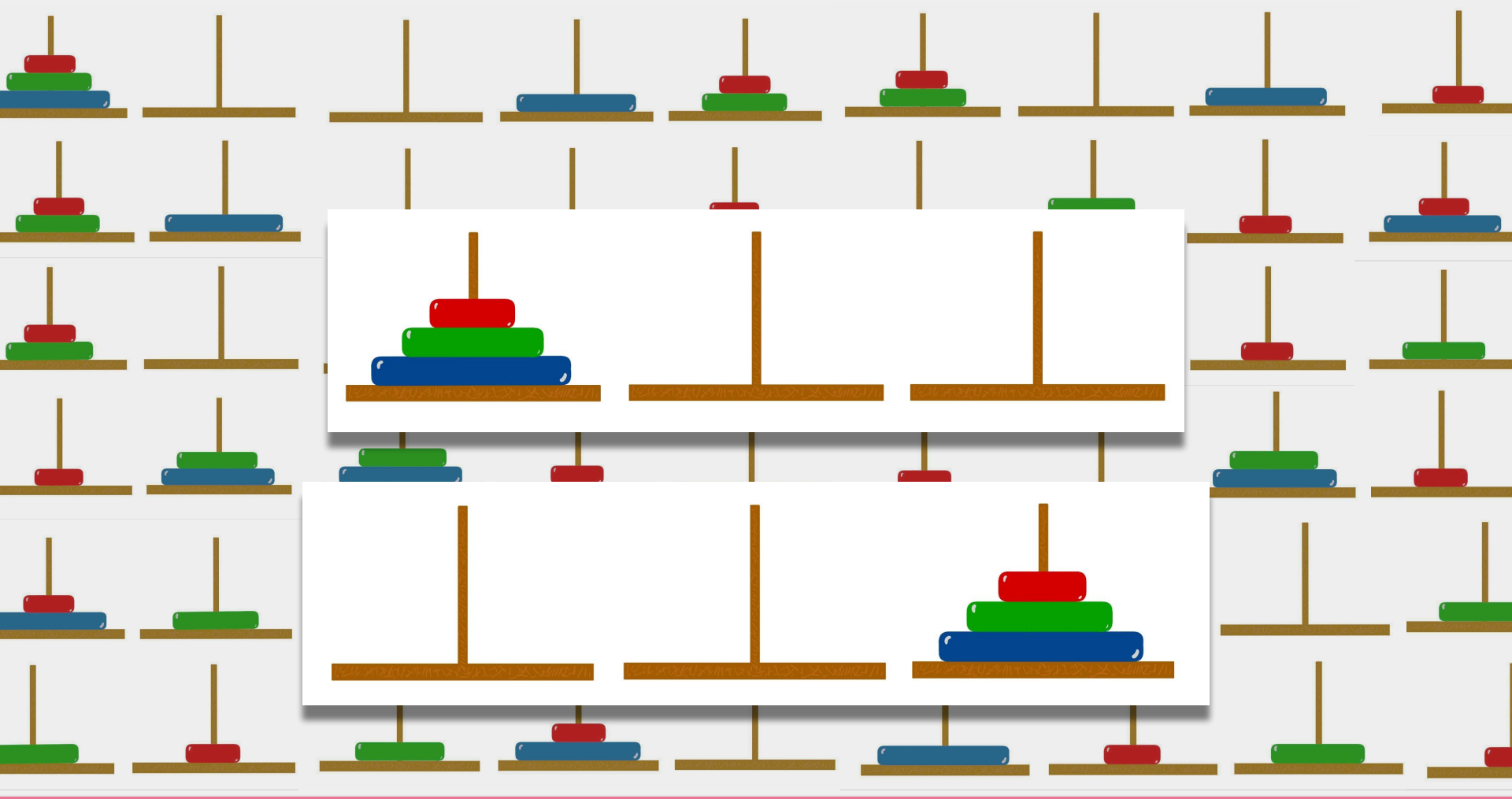
Ejemplo 1: Hoop Stack

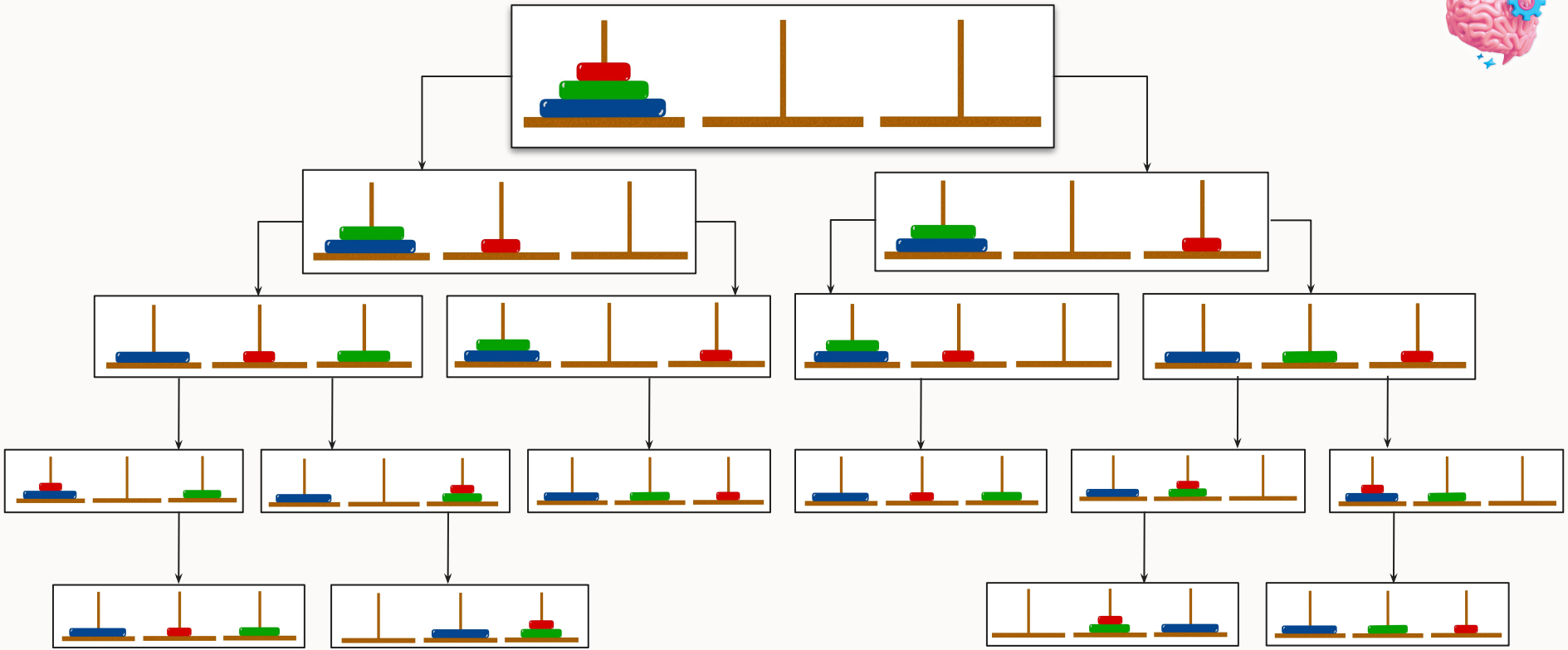
s_init





s_obj





Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1 *Open* es un contenedor vacío
2 *Closed* es un conjunto vacío
3 Inserta s_{init} a Open
4 parent(s_{init}) = null
5 **while** Open ≠ ∅
6 $u \leftarrow$ Extraer(Open)
7 Inserta u en Closed
8 **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9 parent(v) = u
10 **if** $v \in G$ **return** v
11 Inserta v a Open

s_init

s_obj

Acciones:

Son 9 acciones totales...

Mover pieza { roja, verde, azul } a la { derecha, izquierda, centro }

Aunque no todas son permitidas en todos los estados!

Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1 *Open* es un contenedor vacío
2 *Closed* es un conjunto vacío
3 Inserta s_{init} a Open
4 parent(s_{init}) = null
5 **while** Open ≠ ∅
6 $u \leftarrow$ Extraer(Open)
7 Inserta u en Closed
8 **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9 parent(v) = u
10 **if** $v \in G$ **return** v
11 Inserta v a Open

Nodo actual

Open:

Closed:

s_obj

Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1 *Open* es un contenedor vacío
2 *Closed* es un conjunto vacío
3 Inserta s_{init} a Open
4 parent(s_{init}) = null
5 **while** Open ≠ ∅
6 $u \leftarrow$ Extraer(Open)
7 Inserta u en Closed
8 **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9 parent(v) = u
10 **if** $v \in G$ **return** v
11 Inserta v a Open

Nodo actual (u)

Open:

Closed:

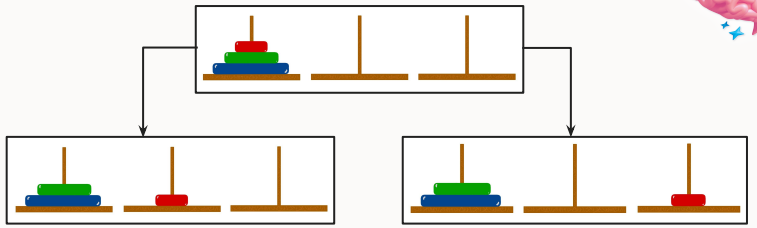
s_obj

Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

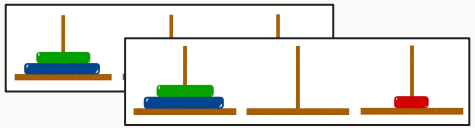
1 *Open* es un contenedor vacío
2 *Closed* es un conjunto vacío
3 Inserta s_{init} a Open
4 parent(s_{init}) = null
5 **while** Open ≠ ∅
6 $u \leftarrow$ Extraer(Open)
7 Inserta u en Closed
8 **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9
10 parent(v) = u
11 **if** $v \in G$ **return** v
Inserta v a Open

Nodo actual (u)



Succ(u)

Open:



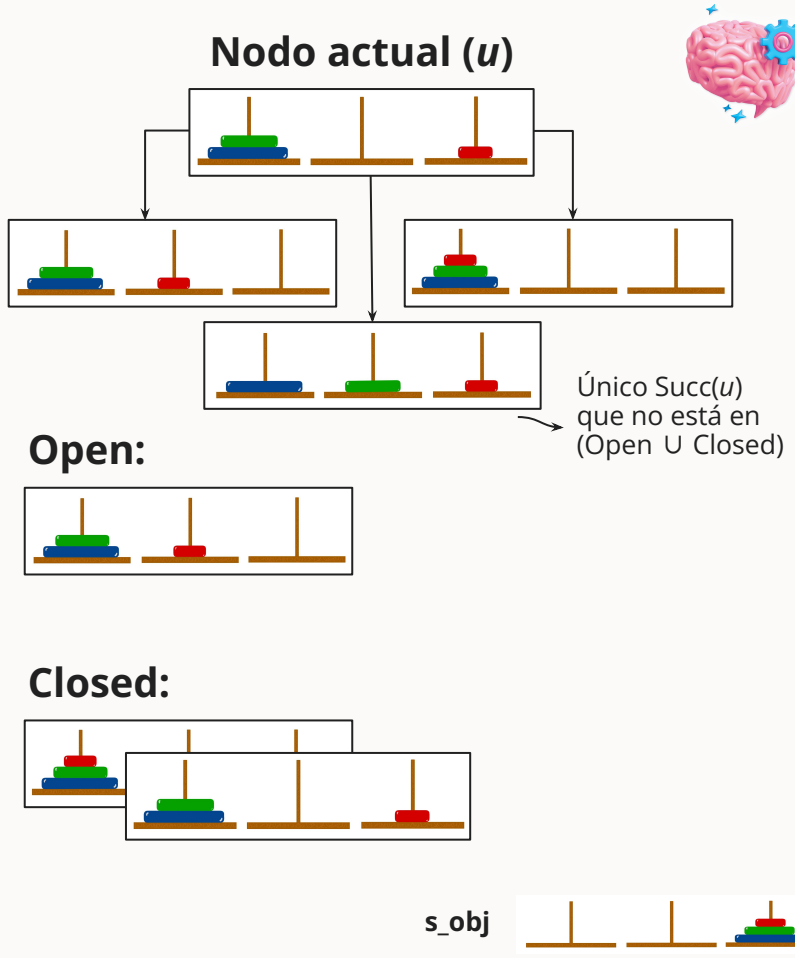
Closed:

s_obj

Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

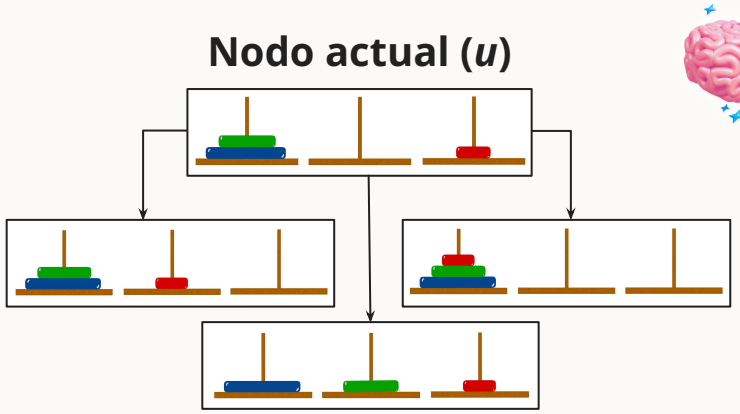
1 *Open* es un contenedor vacío
2 *Closed* es un conjunto vacío
3 Inserta s_{init} a Open
4 parent(s_{init}) = null
5 **while** Open ≠ ∅
6 $u \leftarrow$ Extraer(Open)
7 Inserta u en Closed
8 **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9 parent(v) = u
10 **if** $v \in G$ **return** v
11 Inserta v a Open

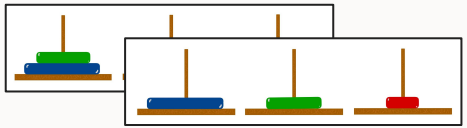


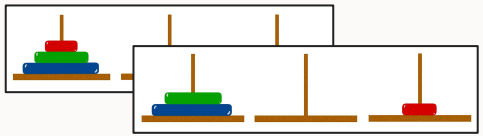
Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open



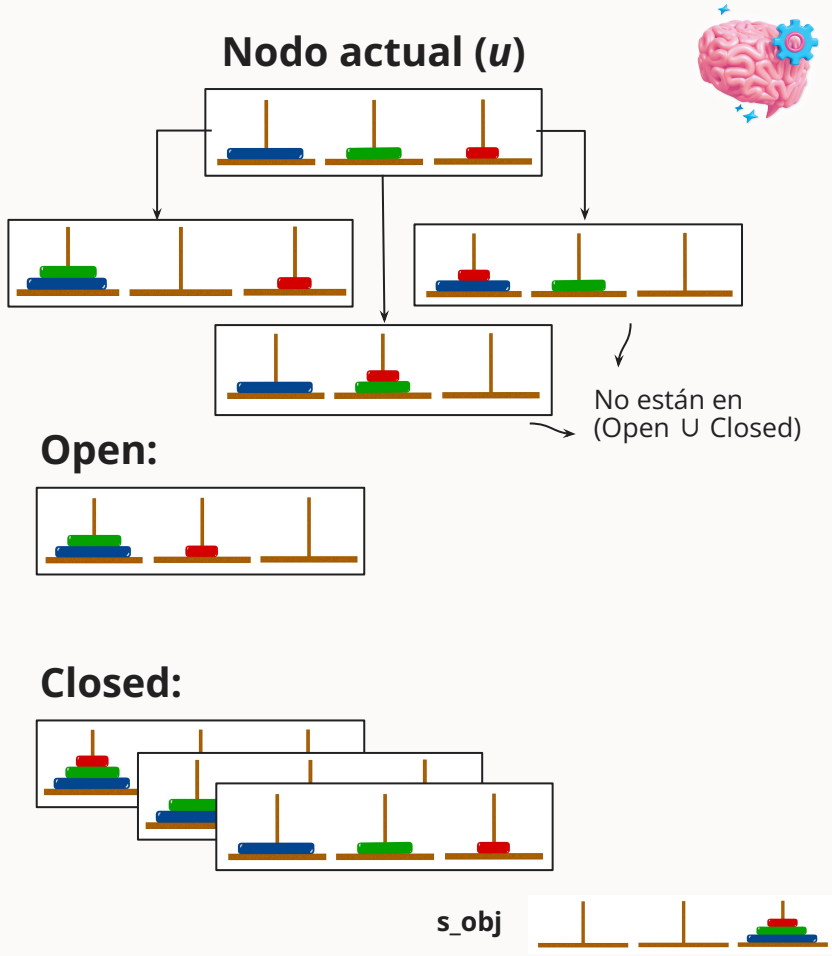




Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

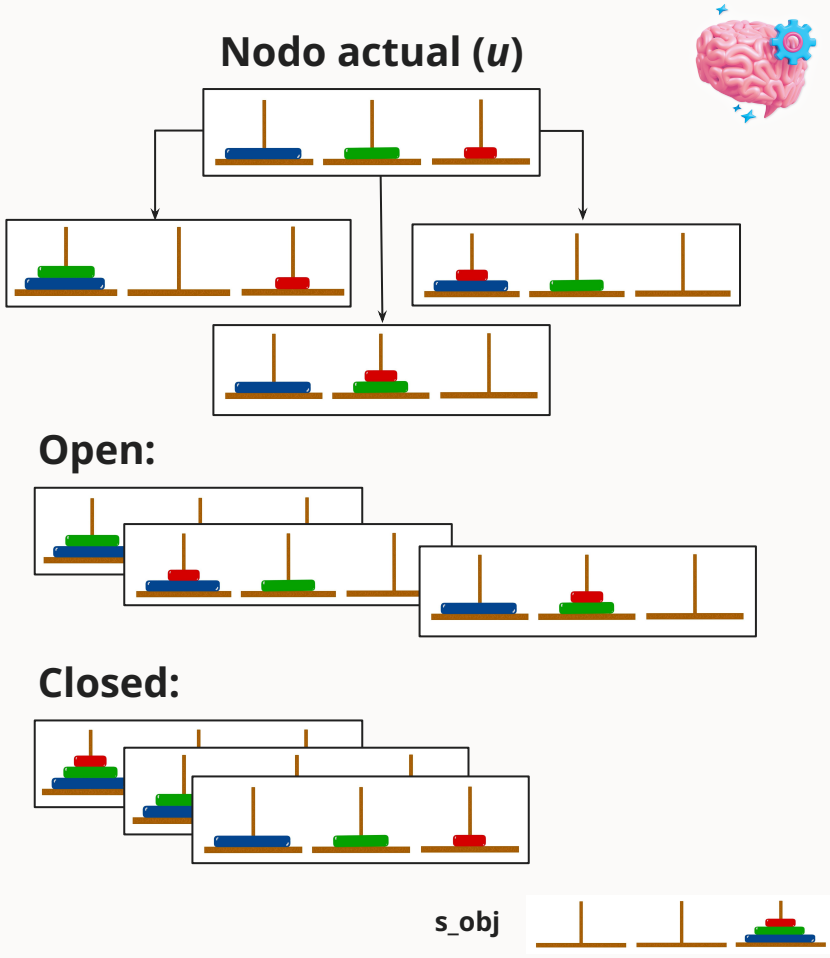
1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open



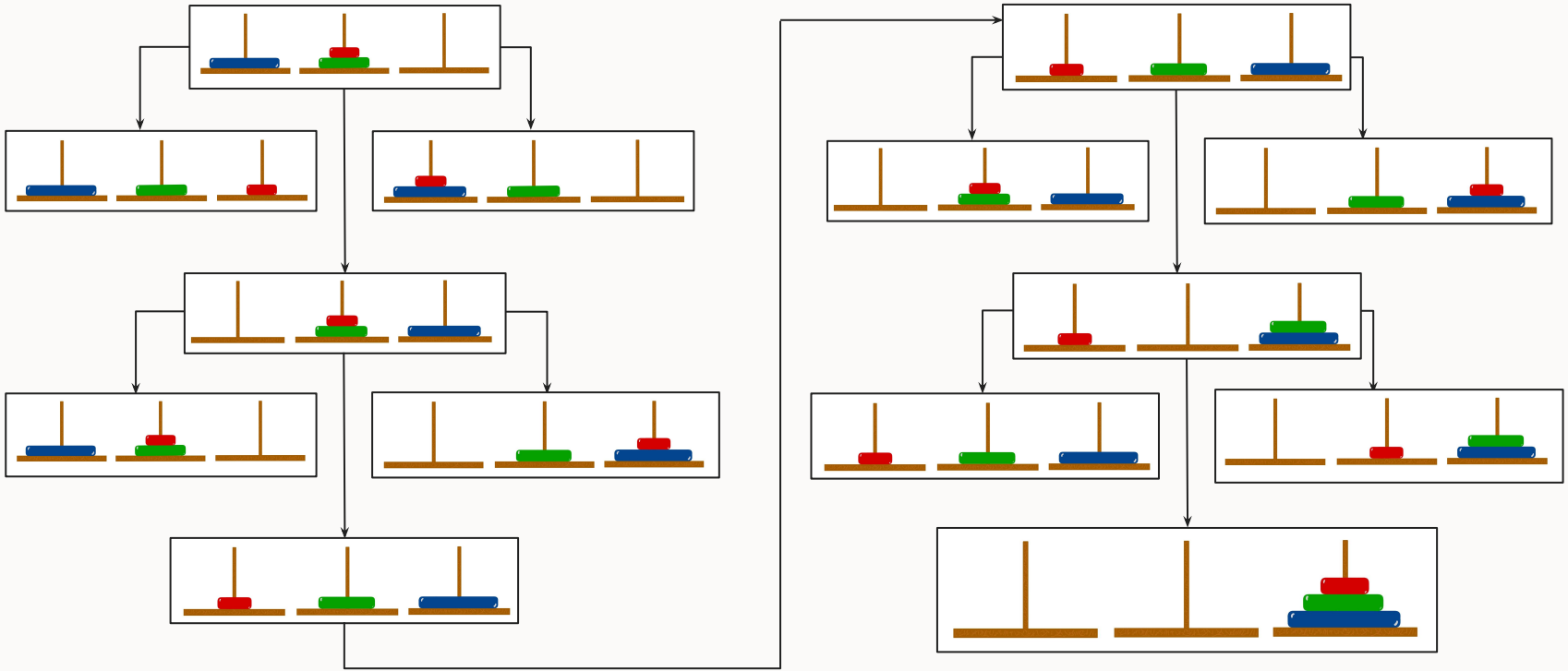
Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

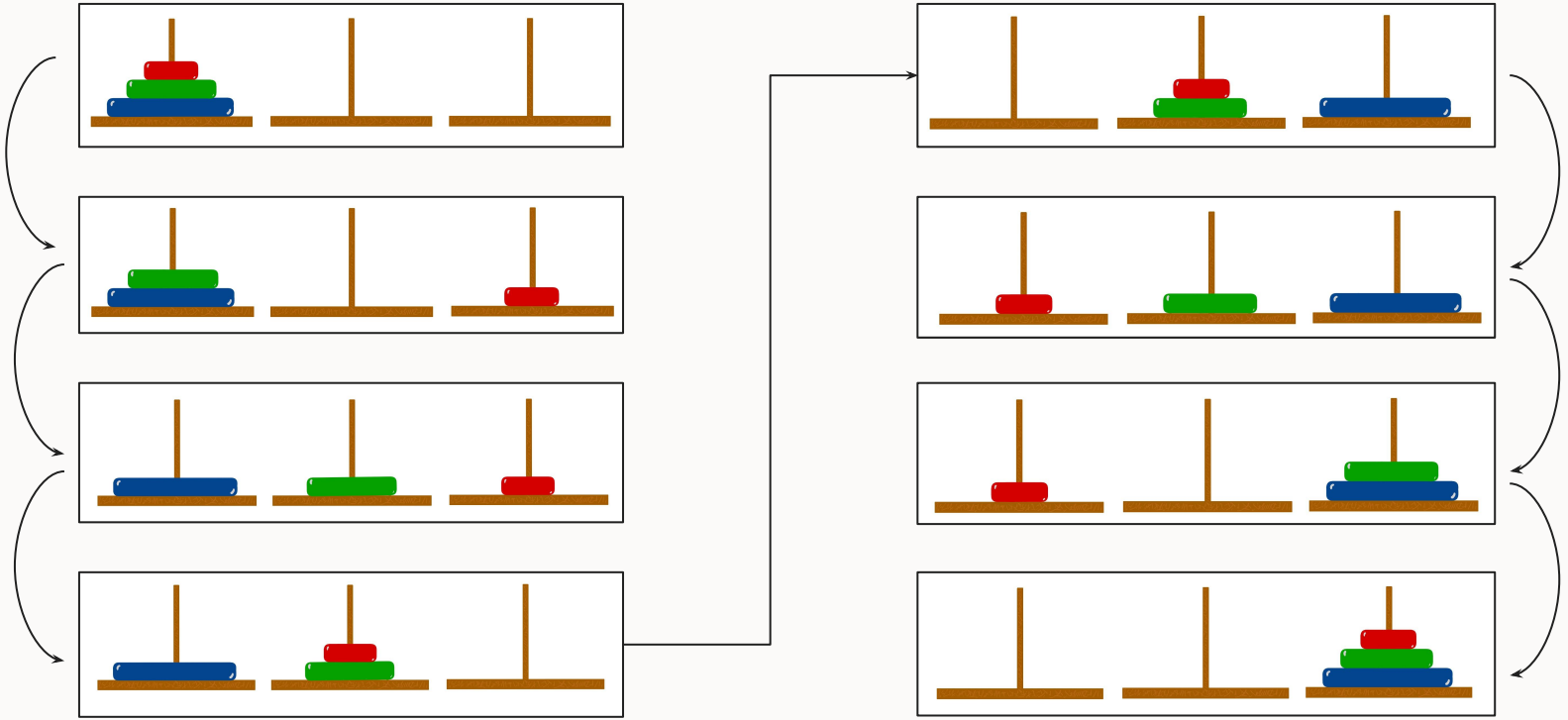


Si seguimos el algoritmo...



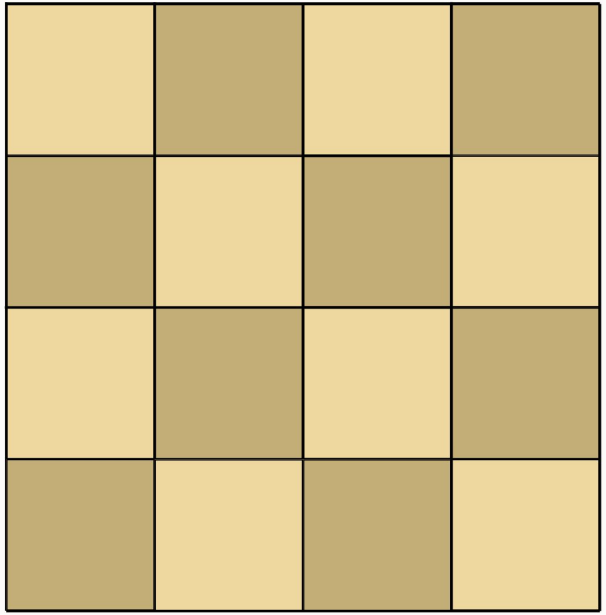
s_obj

Llegamos al nodo objetivo!
Con el registro de *parents* podemos obtener el camino seguido:

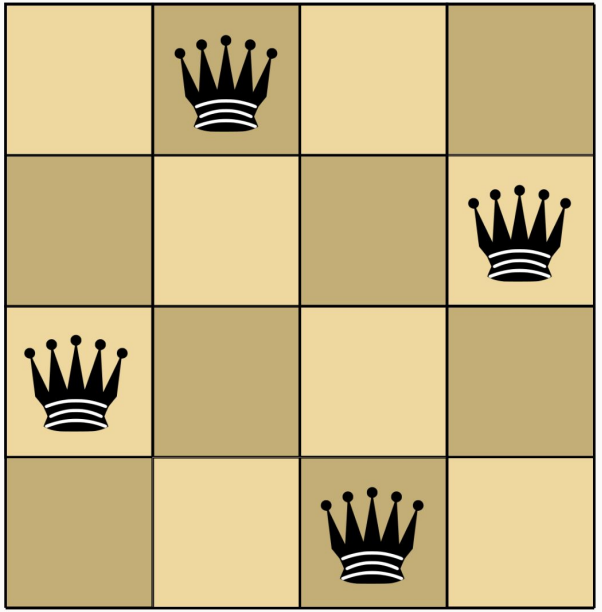


Ejemplo 2: 4 Queens Puzzle

s_init



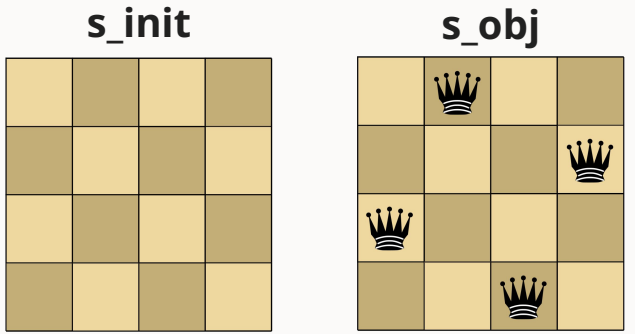
s_obj



Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open



Acciones:

Colocar una reina en una casilla

Restricciones:

- Solo colocar en casillas vacías
- Casilla no puede ser atacada por otra reina

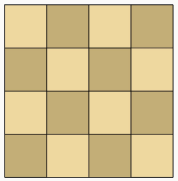
Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

Nodo actual

Open:



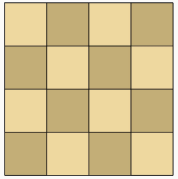
Closed:

Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

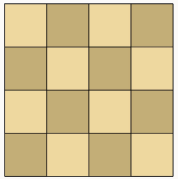
1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

Nodo actual



Open:

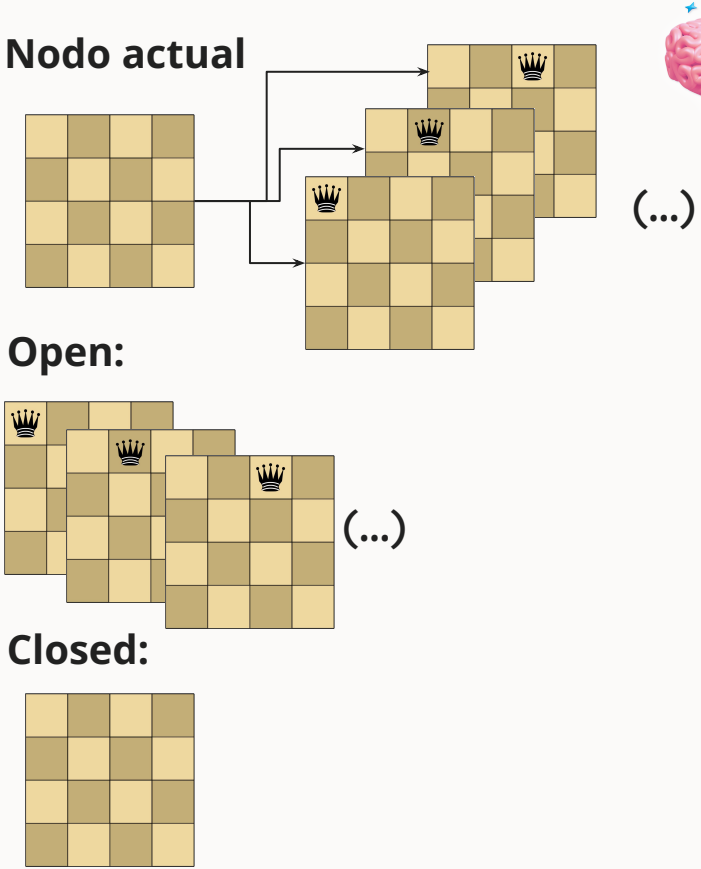
Closed:



Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

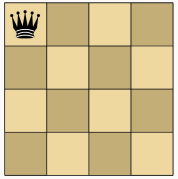


Input: Un problema de búsqueda (S, A, s_{init}, G)

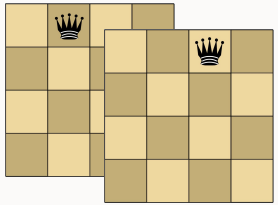
Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

Nodo actual

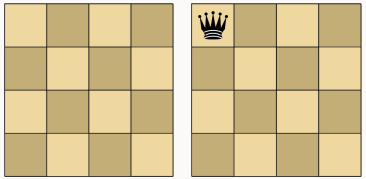


Open:



(...)

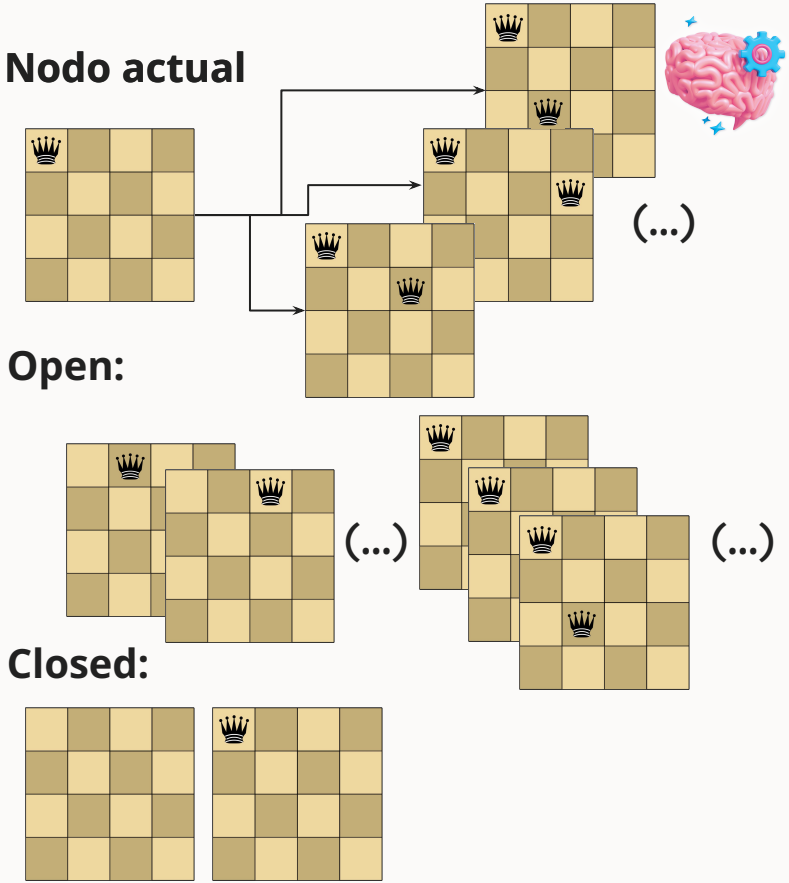
Closed:



Input: Un problema de búsqueda (S, A, s_{init}, G)

Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

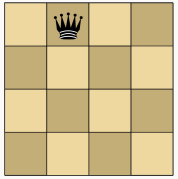


Input: Un problema de búsqueda (S, A, s_{init}, G)

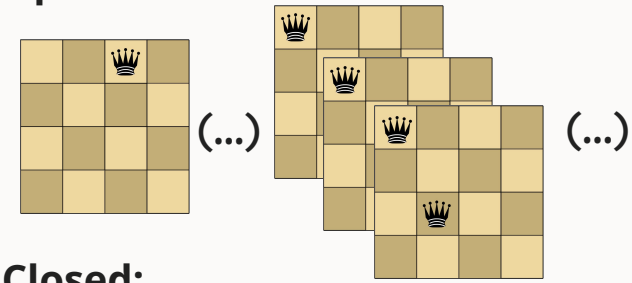
Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open $\neq \varnothing$
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

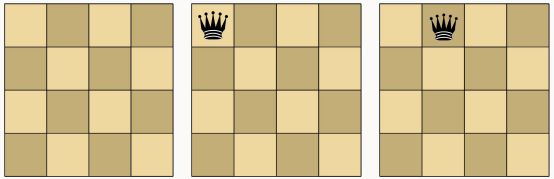
Nodo actual



Open:



Closed:

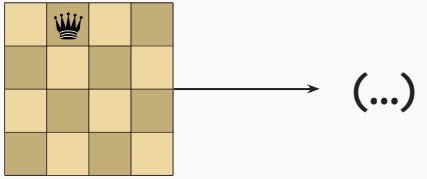


Input: Un problema de búsqueda (S, A, s_{init}, G)

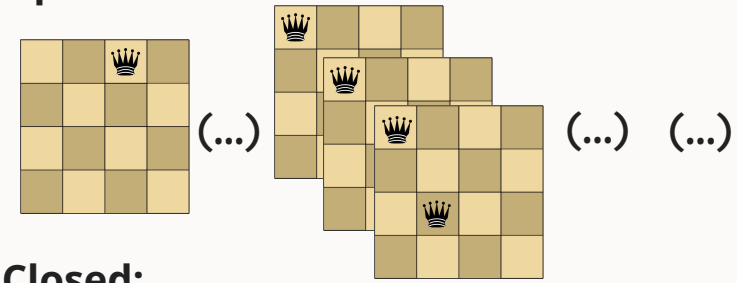
Output: Un nodo objetivo

1. *Open* es un contenedor vacío
2. *Closed* es un conjunto vacío
3. Inserta s_{init} a Open
4. parent(s_{init}) = null
5. **while** Open ≠ ∅
6. $u \leftarrow$ Extraer(Open)
7. Inserta u en Closed
8. **for each** $v \in Succ(u) \setminus (Open \cup Closed)$
9. parent(v) = u
10. **if** $v \in G$ **return** v
11. Inserta v a Open

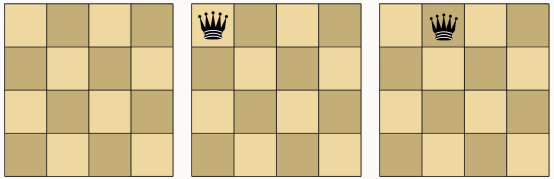
Nodo actual



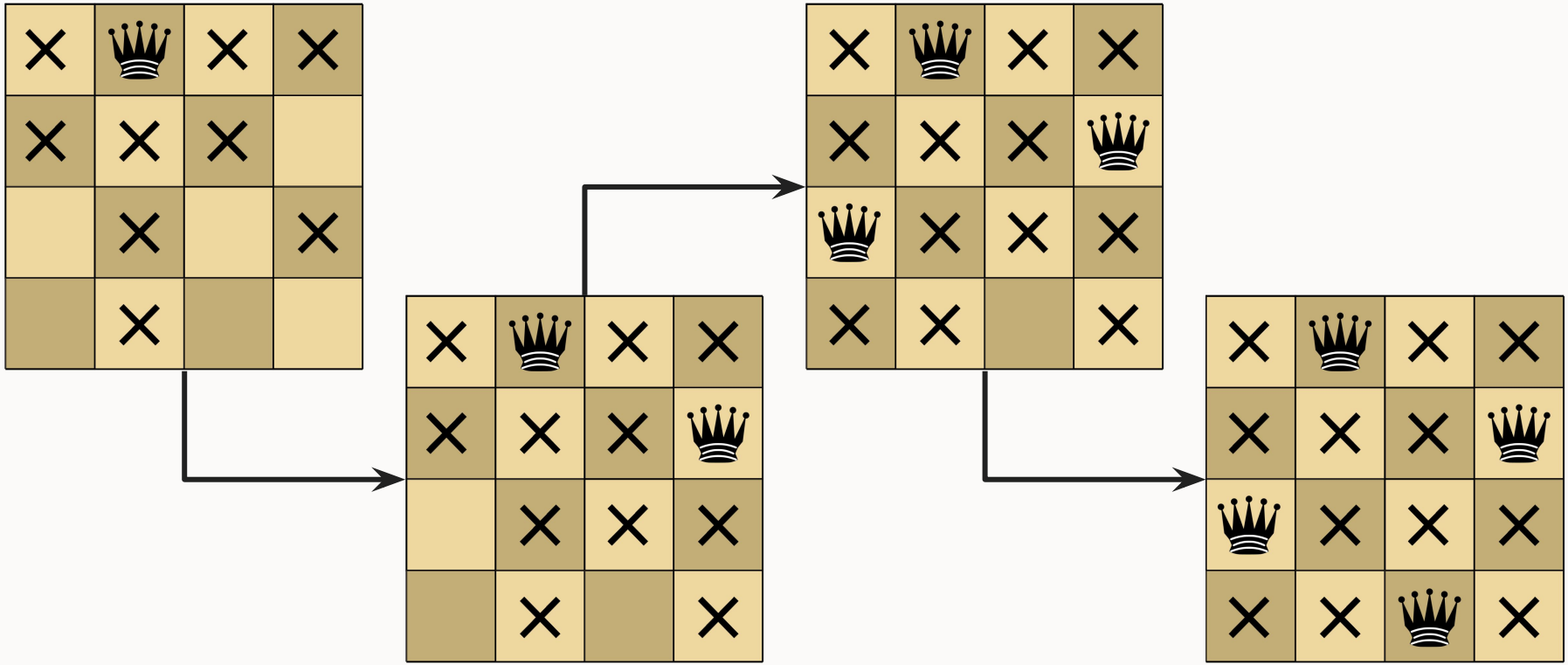
Open:



Closed:



Siguiendo el algoritmo:



Ayudantía 3

Introducción a la búsqueda

Por Martín Lagies y Felipe Espinoza

08 de abril 2024