

# Tema 2: Búsqueda

---

Rosa María Maza Quiroga – [rosammq@uma.es](mailto:rosammq@uma.es)

Departamento de Lenguajes y Ciencias de la Computación

Universidad de Málaga



# Relación con el Tema 1

---

¿AGENTE MÁS SIMPLE?

Agente reactivo.

¿AGENTE CON ÉXITO CONSIDERANDO  
ACCIONES FUTURAS?

**Agente basado en objetivos.**

También llamado:

**Agente resolvente de problemas.**

El que veremos en este tema.

# Resumen

---

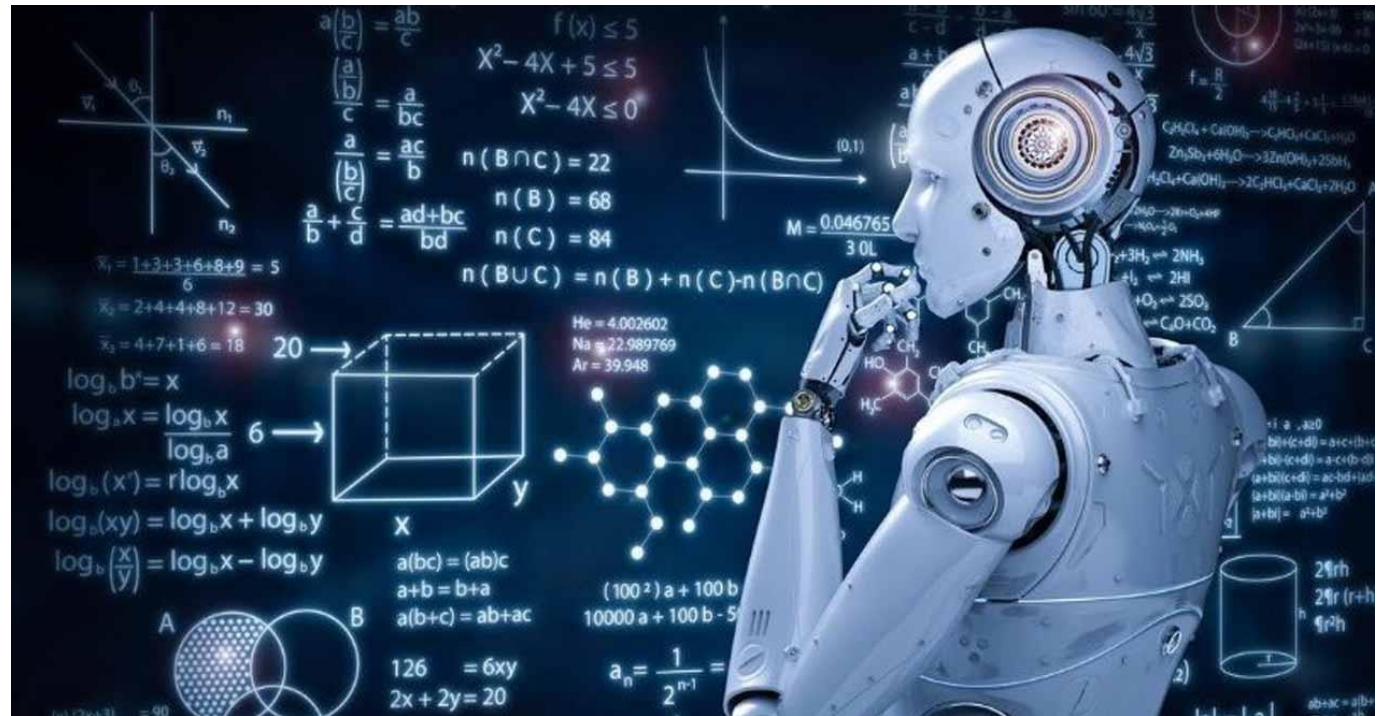
1. Agentes resolventes-problemas
2. Búsqueda de Soluciones
3. Estrategias Búsqueda: No informada
4. Estrategias de búsqueda: Informada

# 1. Agentes resolventes de problemas

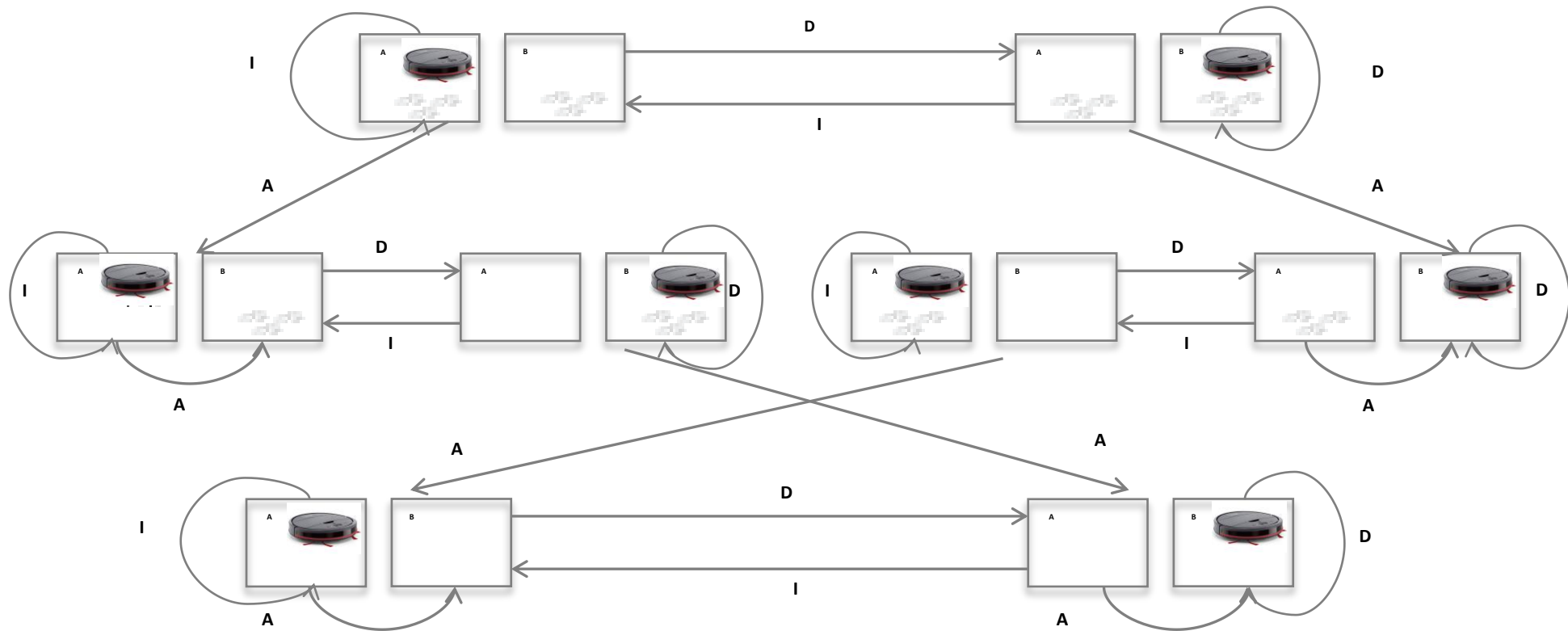
Espacio de estados

Proceso a seguir en problemas

Espacio de estados:  
implementación del algoritmo



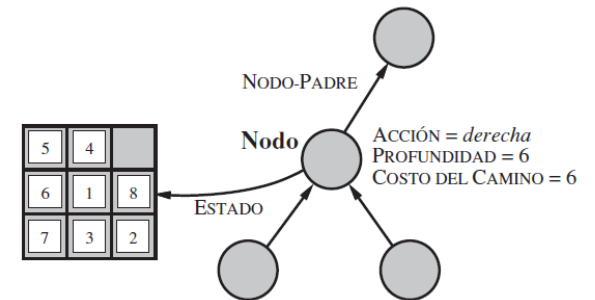
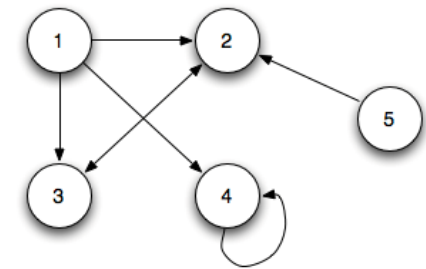
# ❖ Ejemplo mundo: Mundo Aspiradora



# 1. Agentes resolventes-problemas:

## Espacio de estados: agente individual

- ¿Cómo vamos a representar un mundo? Mediante un **espacio de estados**.
- **Espacio de estados:** es la formalización (representación de una idea mediante recursos formales de un sistema, en este caso, con forma simbólica) de cómo funciona una parte del mundo.
- Un espacio de estados está formado por:
  - Estados
  - Acciones posibles: precondiciones y efectos
- Un espacio de estados NO ES UNA SOLUCIÓN A UN PROBLEMA
- **Espacio de estados:** define implícitamente un **grafo**:
  - **Nodos:** estados
  - **Arcos:** acciones
  - Tipo de grafo: **Árbol**: entre 2 nodos sólo hay 1 camino posible.
- **Espacio de estados** se usa cuando la solución es un plan:
  - Percibo → **Construyo** → Razono → Devuelvo solución (plan)



# 1. Agentes resolventes-problemas: Cuando tenemos un problema...

---

- Problema: estar en un estado y querer cambiar a otro estado (como la vida ;) ).

Proceso a seguir:

1. ¿Cómo representamos un problema? Mediante **espacio de estados**.
2. ¿Cómo definimos el problema? Creamos una **instancia** del problema, formada por:
  1. Estado inicial
  2. Estado final o condición objetivo.
3. Construimos solución: elegir **estrategia** (algoritmo):

Al elegir una estrategia estamos eligiendo el algoritmo y la manera en que el algoritmo construye la solución:

  - ej. **Árbol de búsqueda**: estructura para encontrar la solución.
    - Existen varios caminos. La estrategia decide qué camino.
  1. **No informada** (ciega):
  2. **Informada**

# ❖ Ejercicio: PUZLE-8

---

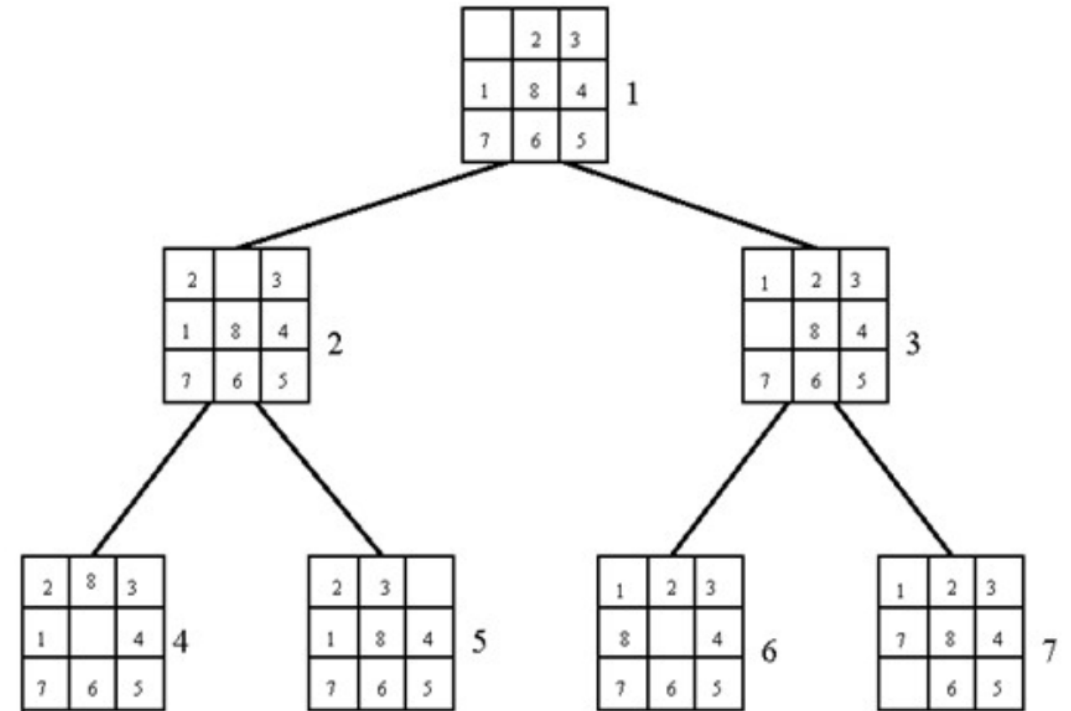
- Representación del puzle-8: Espacio de estados

5		8
4	2	1
7	3	6



# 1. Agentes resolventes-problemas: Solución a problemas

- Contemplar todos los estados: complicado (memoria).
- Por eso, se crean diferentes estrategias.



# 1. Agentes resolventes-problemas

---

1. Agentes inteligentes deben **maximizar** su rendimiento.
  1. Elegir objetivo y satisfacerlo, implica **formulación objetivo**.
  2. **Formulación problema**, dado un objetivo, es el proceso de decidir acciones y estados a considerar.
2. Agente con distintas opciones inmediatas de valores desconocidos puede decidir qué hacer, examinando las diferentes secuencias posibles de acciones que le conduzcan a estados de valores conocidos y entonces escoger la mejor secuencia.
3. **Búsqueda** es el proceso de hallar la secuencia.
  1. **Algoritmo de búsqueda** toma como entrada un problema y devuelve una solución (secuencia de acciones).

# 1. Agentes resolventes-problemas:

## Espacio de estados: Implementación del algoritmo

Para

1. Problema puede definirse con 4 componentes:

1. **Estado inicial**, donde comienza agente.

2. **Acciones** disponibles por el agente:

1. Formalización más común: **Función sucesor**: <acción, Estado sucesor>

3. **Test objetivo**, determina si un estado es objetivo.

4. **Costo camino**, asigna un costo numérico a cada camino.

Necesitamos conocer:  
precondición y efecto  
de cada acción

Clase:  
Método: devolverá los hijos

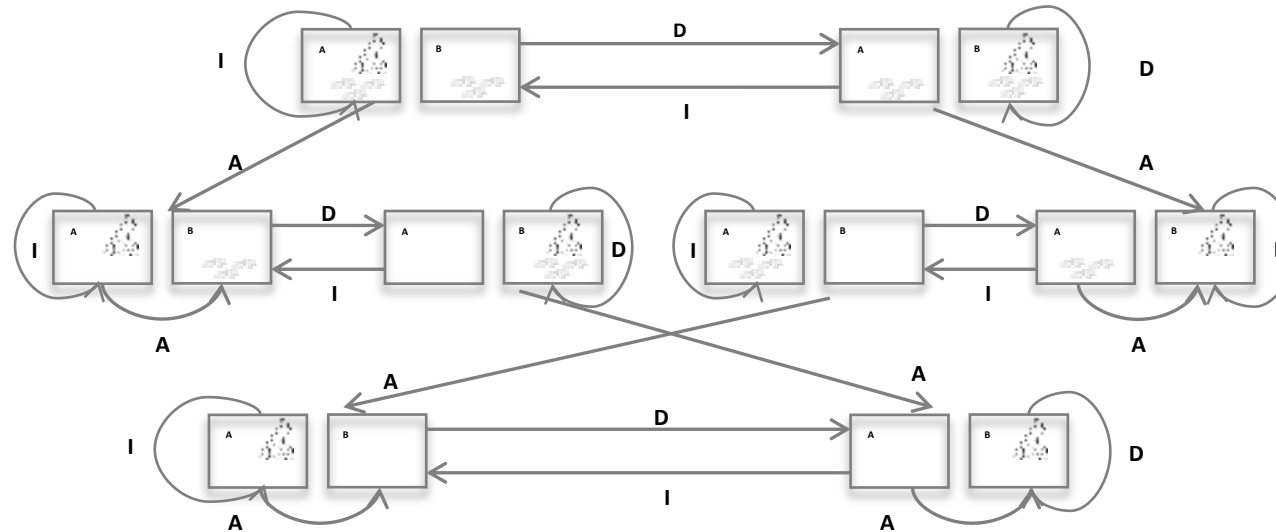
2. **Espacio de estados**: Todos los estados alcanzables desde el estado inicial. Definido por el estado inicial y la **función sucesor**.

3. **Solución**: Camino estado inicial a un estado objetivo.

1. **Calidad solución**: viene dada por el costo del camino.

2. **Solución Óptima**: coste mínimo de todas las soluciones.

# ❖ Ejemplo Problema: Mundo Aspiradora



- **Estados**: Dos localizaciones posibles y cada una puede estar sucia o no  $\rightarrow 2 \times 2^2=8$
- **Estado inicial**: Cualquier estado.
- **Función sucesor**: Tres **acciones** posibles (Izquierda, Derecha, Aspirar)
- **Test objetivo**, todos los estados cuadrados limpios.
- **Costo camino**, costo individual es 1, costo camino número pasos que lo componen.

# ❖ Ejemplo Problema: PUZLE-8

- **Estados**: Localización de las 8 fichas y el blanco.
- **Estado inicial**: Cualquier estado.
- **Función sucesor**: Cuatro acciones posibles (Izquierda, Derecha, Arriba, Abajo)
- **Test objetivo**, estados coincide con la configuración objetivo.
- **Costo camino**, costo individual es 1, costo camino número pasos que lo componen.

5		8
4	2	1
7	3	6

Posible estado inicial

1	2	3
4	5	6
7	8	

Estado final

## 2. Búsqueda de soluciones

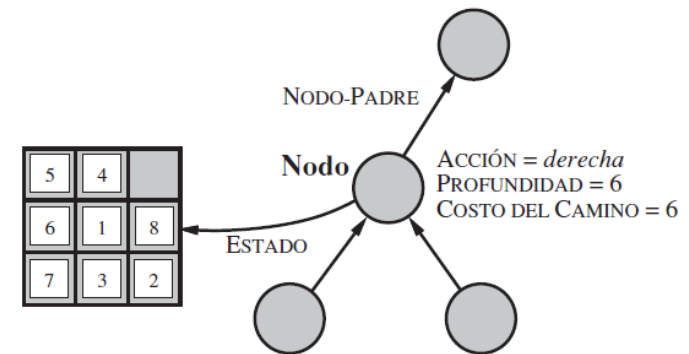
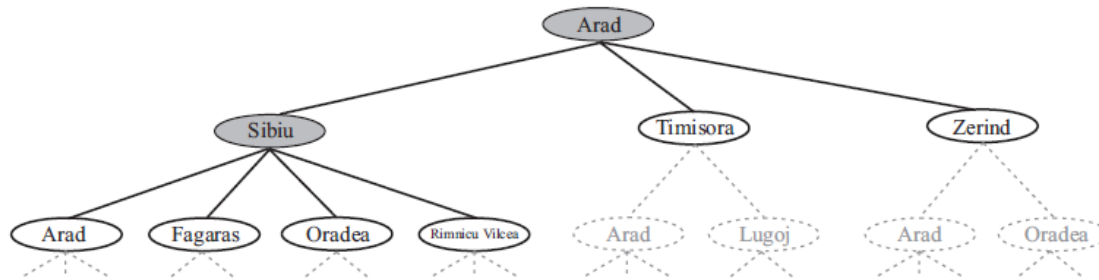
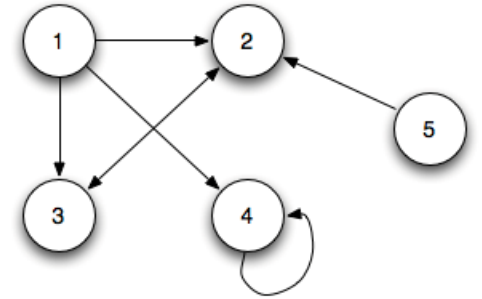
Resolución de problema

Rendimiento de la resolución



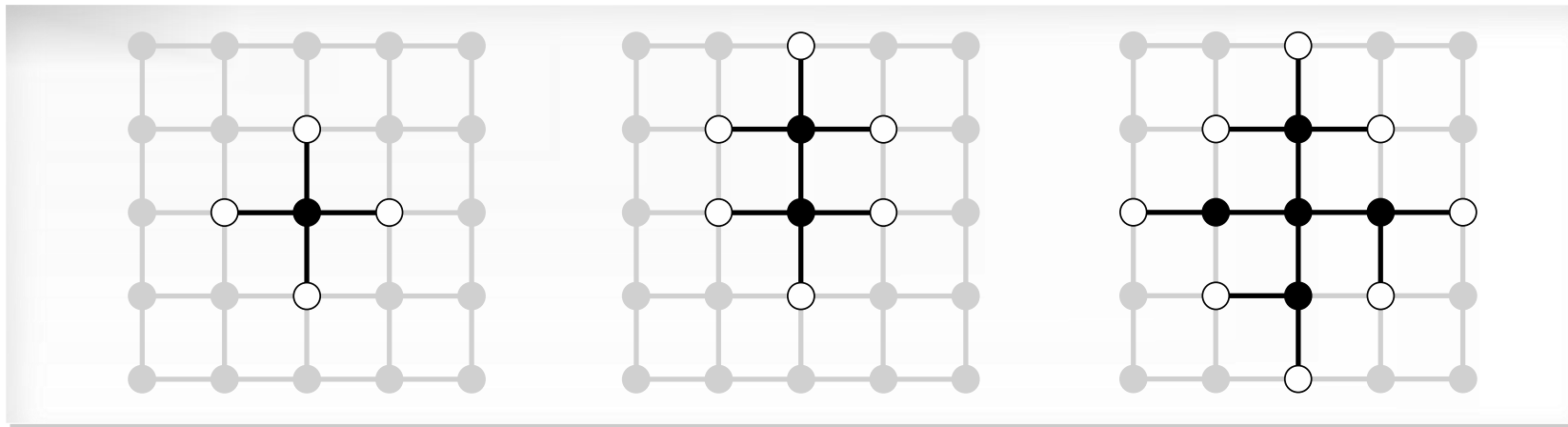
## 2. Búsqueda de soluciones: Resolución problema

- **Espacio de estados:** definición del mundo mediante un **grafo**:
  - **Nodos:** estados
  - **Arcos:** acciones
  - Tipo de grafo: **Árbol**: entre 2 nodos sólo hay 1 camino posible.



## 2. Búsqueda de soluciones: Resolución problema

- **Frontera:** La colección (conjunto) de nodos (nodos hoja) que se genera pero no se ha expandido.
  - La estrategia de búsqueda será una función que seleccione del conjunto el siguiente nodo a expandir.





## 2. Búsqueda de soluciones: Rendimiento de la resolución

---

- **Completitud:** garantizado algoritmo encuentra solución cuando existe.
- **Admisibilidad:** cuando encuentra la solución, ésta es óptima.
- **Optimización:** encuentra la estrategia la solución óptima.
- **Complejidad en tiempo:** cuanto tarda en encontrar la solución.
- **Complejidad en espacio:** cuanta memoria necesita para el funcionamiento de la búsqueda.
- **Eficiencia:**
  - **Coste de la búsqueda:** complejidad en tiempo y también uso memoria.
  - **Coste total:** coste de la búsqueda y coste del camino solución encontrado.

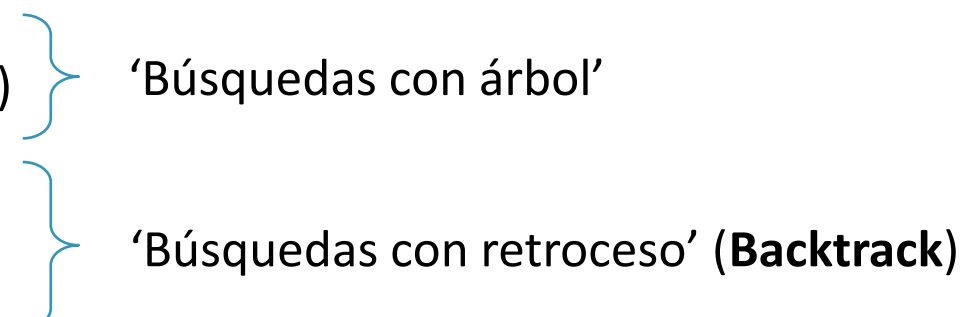
### 3. Estrategias de búsqueda

No informadas



# 3. Búsquedas no informadas

---

- No tienen **información adicional** acerca de los estados más allá de la que proporciona el problema.
- Generan sucesores.
- Distinguen si un estado es el objetivo o no.
- Estrategias (algoritmos):
  1. Primero en anchura (Breadth-first)
  2. Anchura con costes = Costo uniforme (Dijkstra)
  3. Primero en profundidad (Depth-first)
  4. Profundidad limitada
  5. Profundidad iterativa

‘Búsquedas con árbol’

‘Búsquedas con retroceso’ (**Backtrack**)
- Los problemas de búsqueda de complejidad-exponencial no pueden resolverse por métodos sin información, salvo casos pequeños.

### 3. Búsquedas no informadas: FIFO (First In First Out)

---

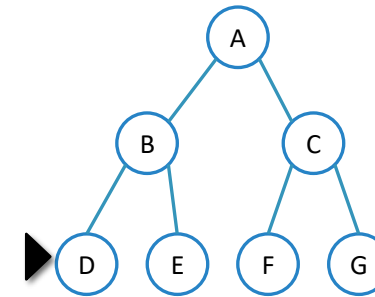
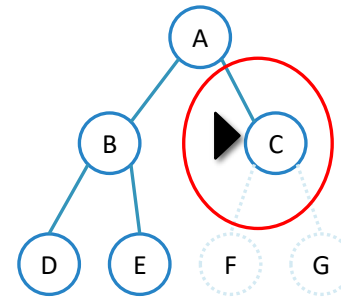
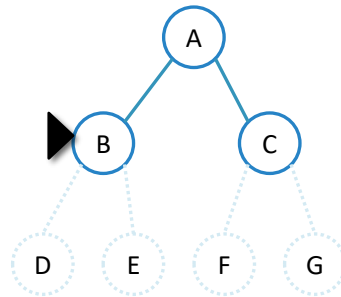
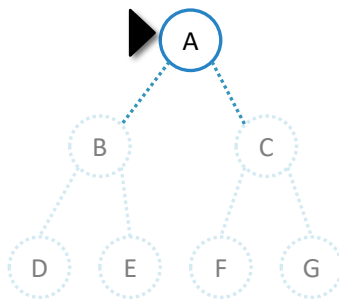


# 3. Búsquedas no informadas:

## 1. Primero en Anchura (FIFO)



- Expanden todos los nodos a una **profundidad** en el árbol de búsqueda antes de expandir cualquier nodo del **próximo nivel**.
- Algoritmo de **satisfacción** (no hay preferencia).
- **Óptima** si todas las acciones tienen mismo coste.
- **Problema:** grandes requisitos de memoria, más que tiempo de ejecución.



# 3. Búsquedas no informadas:

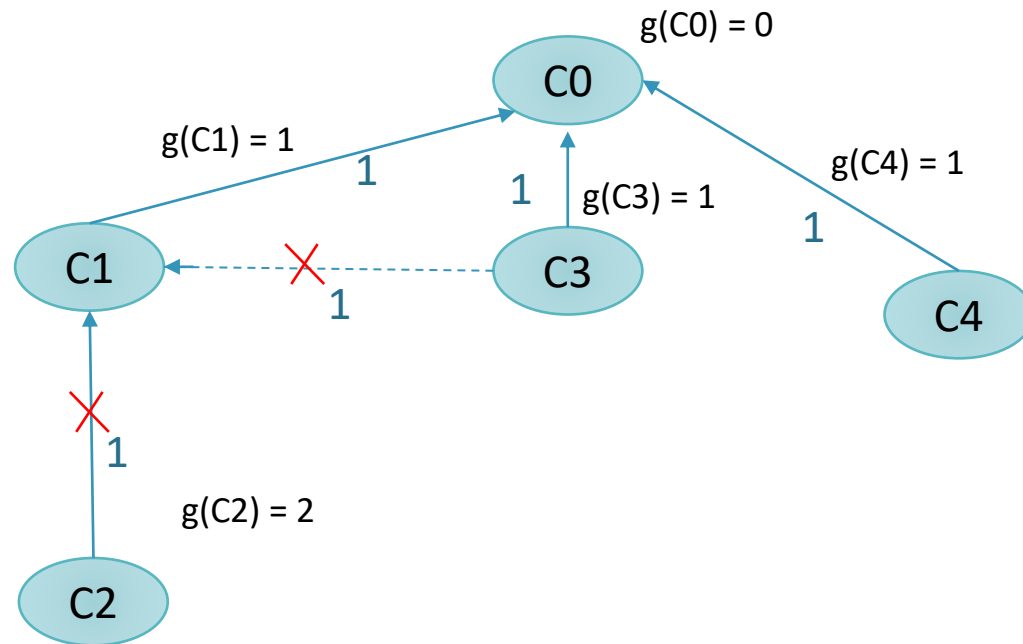
## 2. Anchura con costes (Dijkstra)



- Función  $g(n)$ : para cada nodo en el camino de búsqueda,  $g(n)$  será el **coste del camino guardado en el árbol desde el Estado inicial hasta  $n$** . Cola con prioridad.
  - Expande el nodo  $n$  con menor coste.
  - No importa el número de pasos que tiene un camino, pero sí su **coste total**.
  - **Óptima** con cualquier función de coste.
  - Analizamos el caso sin ciclos:
    - Podría darse bucle infinito si existe un ciclo de coste 0 que conduce al mismo estado.
- ❖ Ejercicio: Distancia en Km entre ciudades.

# ❖ Ejercicio: distancia entre ciudades

Aplicar el algoritmo de Dijkstra para encontrar la ruta más corta entre la ciudad C0 y la C3, dada la siguiente tabla que indica las ciudades conectadas y los Km entre ciudades.



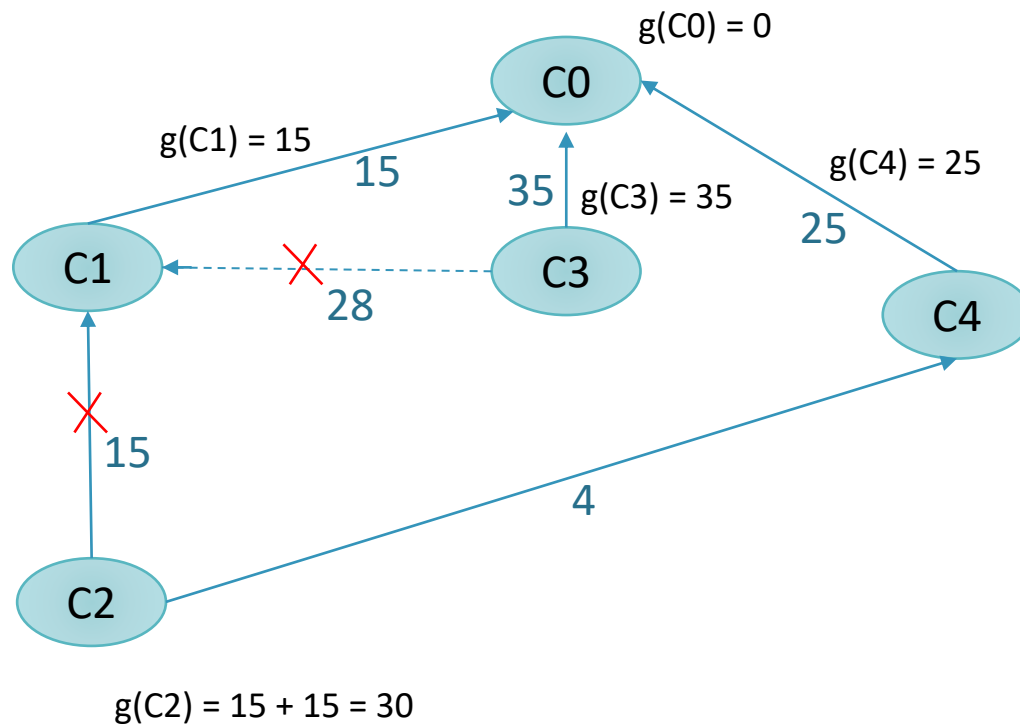
	C0	C2	C3	C4	C5
C0		1		1	1
C1	1		1	1	
C2		1			1
C3	1	1			
C4	1		1		

Iteraciones	Abiertos n(g(n))
1	C0(0) Seleccionamos C0 para expansión.
2	Ordenar en orden creciente: C1(1), C3(1), C4(1) Seleccionamos C1 para expansión.
3	Ordenamos en orden creciente: C3(1), C4(1), C2(2) Seleccionamos C3 para expansión.
4	C3 es seleccionado para expansión, y es el objetivo.
Solución	C0 – C3 Coste = 1

Resolver por Dijkstra con costes iguales ( $g(n)=0$ ) es lo mismo que hacer algoritmo primero en anchura.

# ❖ Ejercicio: distancia entre ciudades

Aplicar el algoritmo de Dijkstra para encontrar la ruta más corta entre la ciudad C0 y la C3, dada la siguiente tabla que indica las ciudades conectadas y los Km entre ciudades.



	c0	c1	c2	c3	c4
c0		15		35	25
c1	15		15	28	
c2		15			4
c3	35	28			
c4	25		4		

Iteraciones	Abiertos $n(g(n))$
1	C0(0) Seleccionamos C0 para expansión.
2	Ordenar en orden creciente: C1(15), C4(25), C3(35) Seleccionamos C1 para expansión.
3	Ordenamos en orden creciente: C4(25), C2(30), C3(35) Seleccionamos C4 para expansión.
4	Actualizar valor de C2. Ordenar creciente: C2(29), C3(35)
5	C3(35) es seleccionado para expansión, y es el objetivo.
Solución	C0 – C3 Coste = 35



### 3. Búsquedas no informadas: LIFO (Last In Last Out)

---



# 3. Búsquedas no informadas:

## Algoritmos Backtrack

---

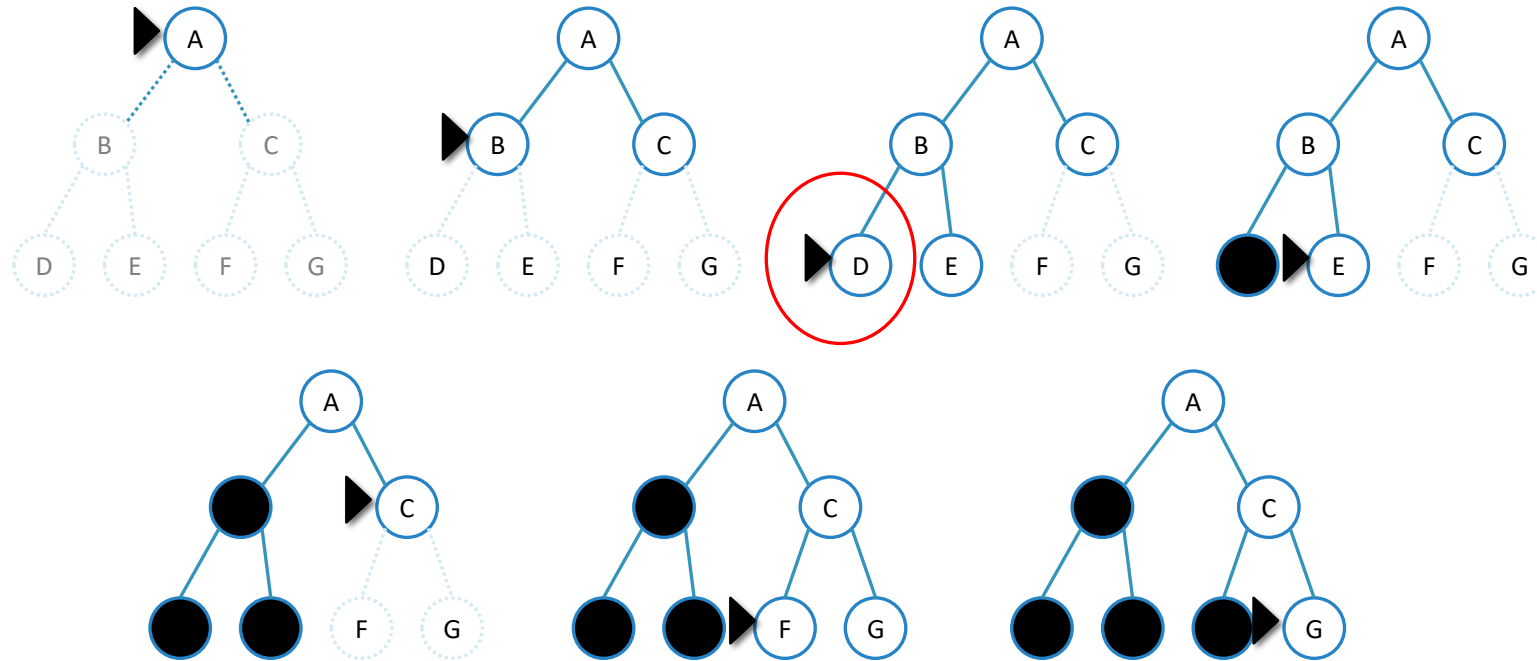
- Útiles para problemas que se representen como:
  - Árboles finitos.
  - Grafos acíclicos dirigidos finitos.
- Ver vídeo disponible en el *campus virtual*: [Algoritmos Backtrack](#)

# 3. Búsquedas no informadas:

## 3. Primero en Profundidad (LIFO)



- Expande el nodo más profundo en la frontera actual del árbol de búsqueda.



- Algoritmo no óptimo y no completo, ya que puede existir profundidad ilimitada.
  - Completo: únicamente en árboles o grafos acíclicos dirigidos y finitos.

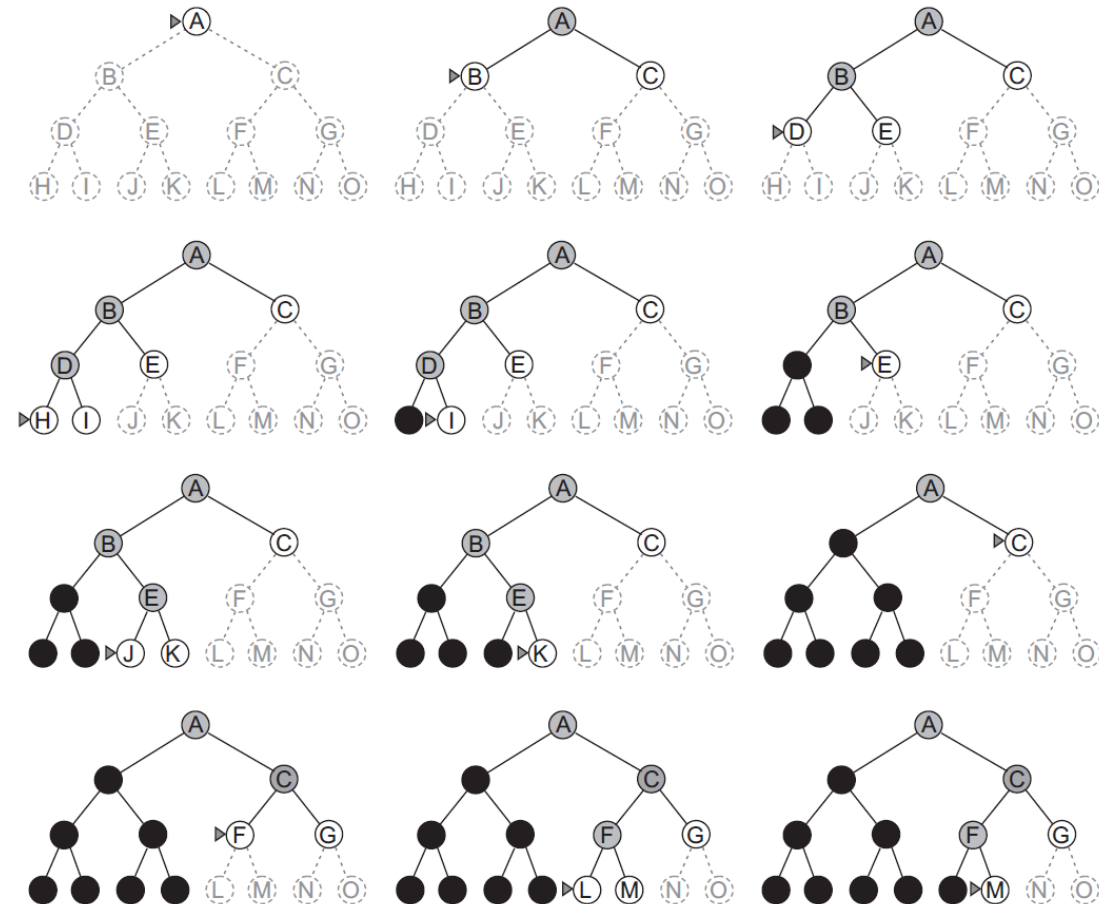
# 3. Búsquedas no informadas:

## 3. Primero en Profundidad (LIFO)



- **Memoria:** requisitos modestos.
- Almacena sólo UN camino desde la raíz a UN nodo hoja, junto con los nodos hermanos no expandidos para cada nodo del camino.
- Cuando el nodo ha sido expandido, se quita de la memoria cuando todos sus descendientes han sido explorados.
- **Problema:** camino muy largo previo al objetivo.

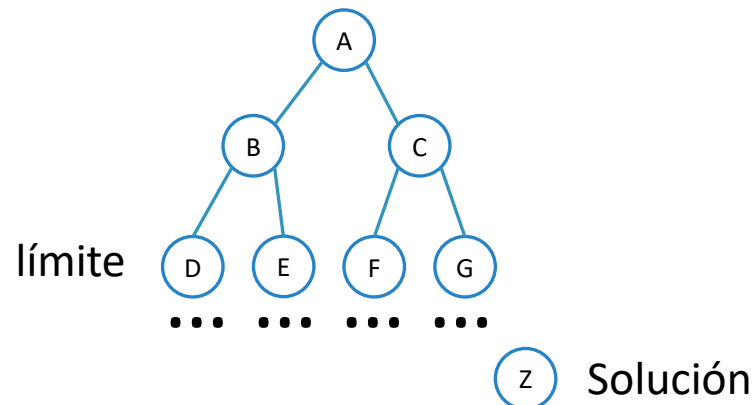
- ❖ En el ejemplo de la derecha:
  - Nodos de profundidad 3 no tienen sucesores.
  - M es el nodo objetivo.
  - Los nodos que han sido expandidos y no tienen descendientes en la frontera se quitan de la memoria (nodos en negro).



# 3. Búsquedas no informadas:

## 4. Profundidad limitada

- Se define límite de profundidad  $l$ .
  - Evita problema de caminos ilimitados o muy largos.
  - Nodos a profundidad  $l$  se les trata como si no tuviesen sucesor.
- **Problema:** Incompletitud si  $l < d$ , objetivo fuera alcance del límite profundidad.
- Se usa **cuando** tenemos **buena estimación** del **nivel** en el que está la **solución**.



# 3. Búsquedas no informadas:

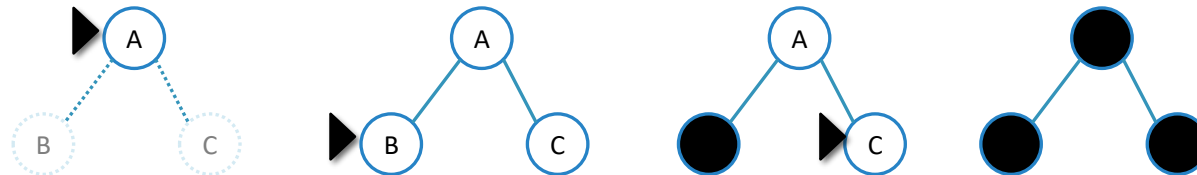
## 5. Profundidad Iterativa

- Si en Profundidad limitada no tenemos una buena estimación del nivel donde se encuentra el objetivo, utilizamos este algoritmo.
- Es el **preferido** cuando hay un espacio grande de búsqueda y no se conoce la profundidad de la solución.
- Combina ventajas de Primero en Profundidad y Primero en Anchura.
  - Llamamos a Primeo en profundidad para cada nivel hasta encontrar la solución.

Límite = 0

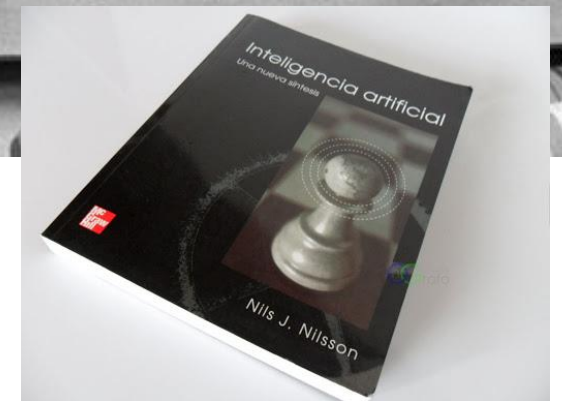
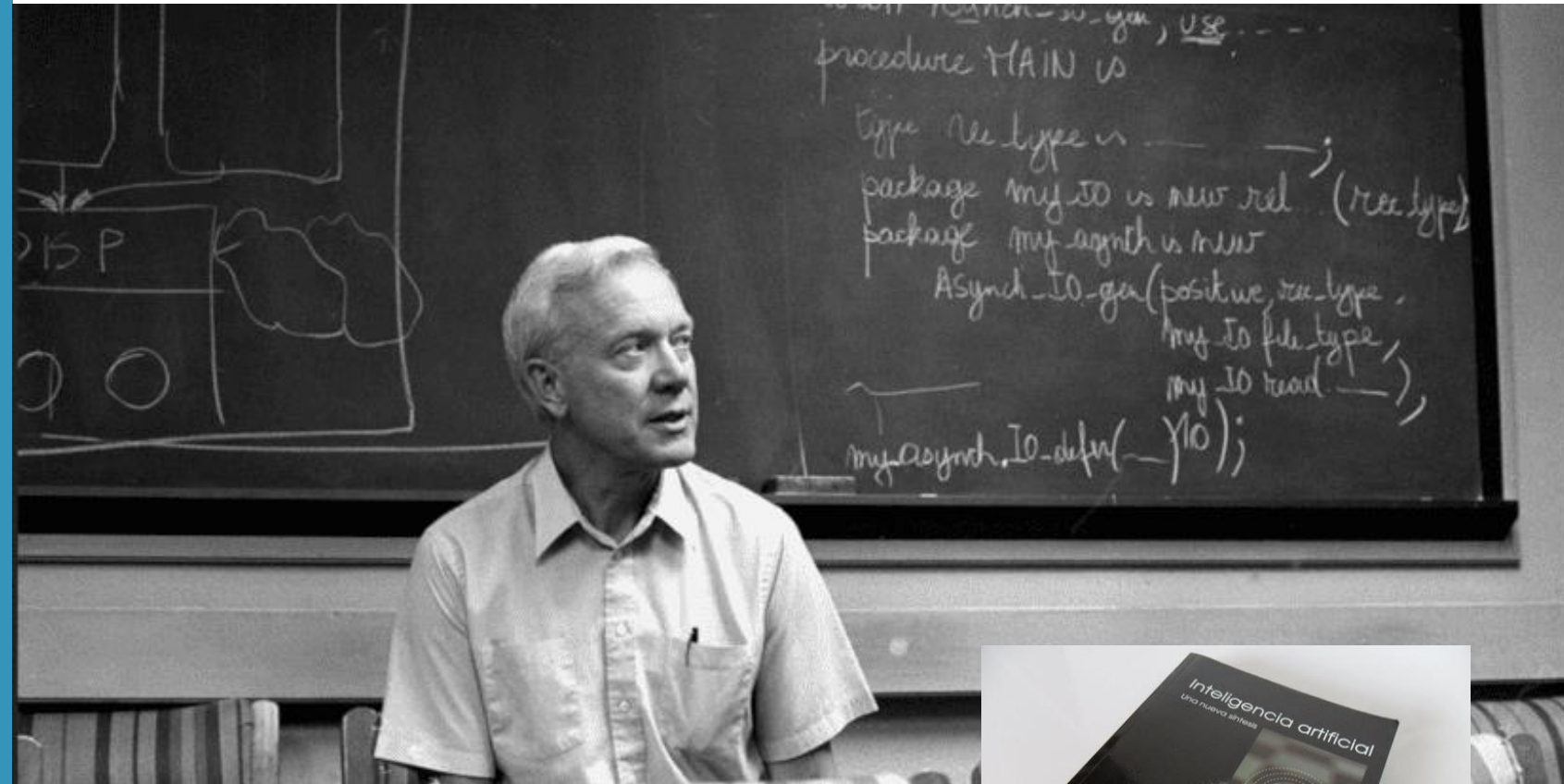


Límite = 1



## 4. Estrategias de búsqueda Informadas

Nils John Nilsson: pionero en robótica e IA (1933-2019) - coautor A\*



# 4. Búsquedas informadas

---

- **Estrategias de búsqueda informada** puede encontrar soluciones de manera más eficiente que una estrategia no informada.
- Selecciona un nodo para la expansión basada en **una función de evaluación  $f(n)$** .
  - Normalmente se expande el nodo con la evaluación más baja.
- **Función heurística  $h(n)$** , forma más común de transmitir el conocimiento adicional del problema al algoritmo de búsqueda.
  - Heurístico: es un proceso para llegar a un objetivo (eureka!: lo encontré!), ‘la cuenta la vieja’
  - Gran parte de nuestro conocimiento (como humano) es heurístico.
  - La mayoría de las veces ayuda a encontrar una solución buena.
  - **Propia de cada problema**





# 4. Búsquedas informadas

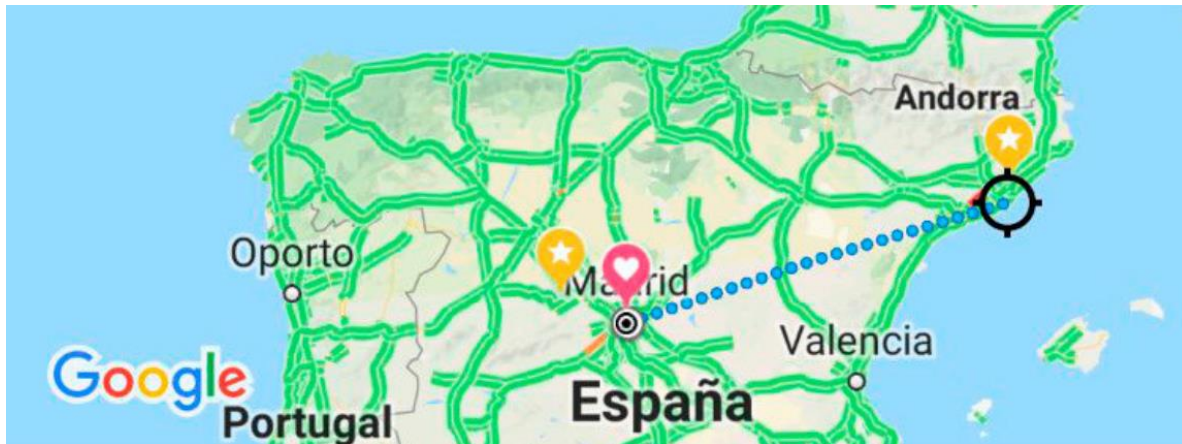
---

## ■ Estrategias (algoritmos):

1. A\*
  2. Voraz (greedy)  
(asignatura: previa a ésta)
  3. Poda alfa-beta  
(asignatura: previa a ésta)
  4. BID con heurística  
(asignatura: Inteligencia Artificial para juegos)
- ‘Búsquedas con árbol’
- ‘Búsquedas con retroceso’

# ❖ Ejemplo mapa de carreteras

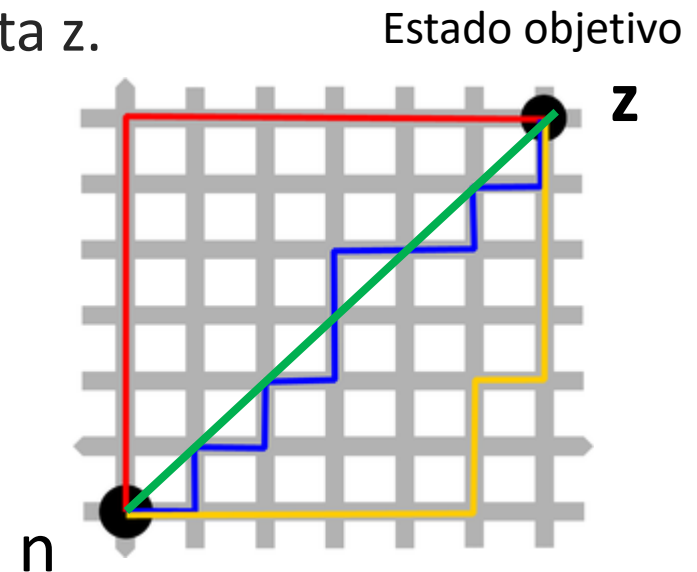
- **Objetivo**: ciudad z
- **Coste**: distancia en Km
- **Función heurística**:  $h(n)$  = distancia directa entre nodo n hasta z



- Caso real: los GPS miden sobre una esfera por lo que son arcos y no rectas.

# ❖ Ejemplo distancia en una malla

- **Objetivo**: ciudad z
- **Coste**: 1 por movimiento
- Movimientos (arriba, abajo, izquierda, derecha)
- Definimos función heurística para este problema.
- Función heurística  $h(n)$  = distancia directa del nodo n hasta z.
- Ejemplos de funciones heurísticas:
  - **Función heurística 1**:  $h_1(n)$  = distancia en **euclídea** a z
  - **Función heurística 2**:  $h_2(n)$  = **distancia de Manhattan** a z
  - ...
  - **Función heurística n**:  $h_n(n)$  ...



# ❖ Ejemplo distancia en una malla

- Definiendo las heurísticas:
- **Distancia euclídea:**

La distancia entre dos puntos es la raíz cuadrada de la suma de los cuadrados de las diferencias de las coordenadas.

$$h(n) = \sqrt{(F - F_o)^2 + (C - C_o)^2}$$

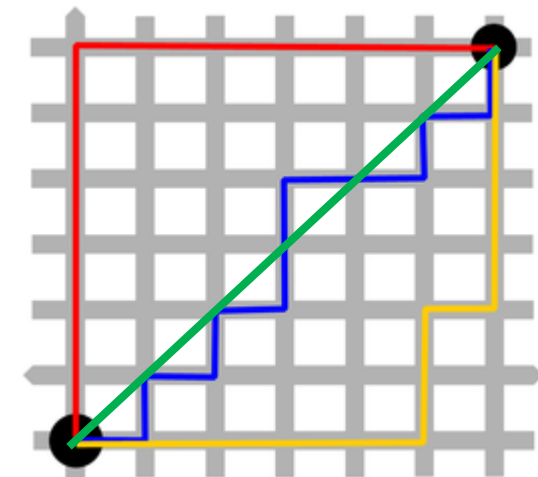
- **Distancia de Manhattan:**

La distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas

$$h(n) = |F - F_o| + |C - C_o|$$

Estado objetivo

$$Z = (F_o, C_o)$$



$$n = (F, C)$$



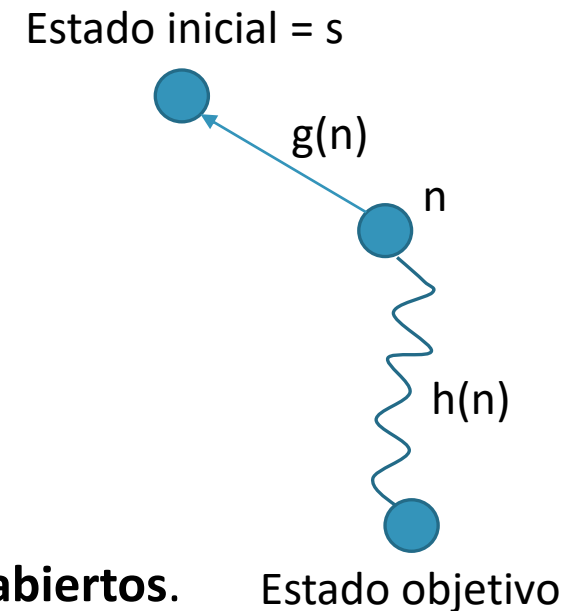
## 4. Búsquedas informadas: Voraz (greedy)

---

- Expandir nodo más cercano al objetivo, basándose que conduce rápidamente a una solución.
  - $f(n) = h(n)$ : coste estimado del camino más barato desde el nodo  $n$  a un nodo objetivo.
- Ejemplo  $h(n)$  = distancia en línea recta entre dos ciudades.
- Algoritmo no óptimo e incompleto.

# 4. Búsquedas informadas: Algoritmo A\*

- Evalúa los nodos con la función de evaluación  $f(n)$ :  $f(n) = g(n) + h(n)$ 
  - **$f(n)$  función de evaluación**: estimación del coste de un camino solución que pasa por  $n$ .
  - $g(n)$  **coste** en el árbol desde origen hasta  $n$ .
  - $h(n)$  **estimación** del coste desde  $n$  hasta el objetivo.
- Es un algoritmo de **optimización**.
- Selección de nodos abiertos = cola con prioridad.
- Dos caminos al mismo nodo: nos quedamos con el de menor  $g(n)$ .
- Ordena la lista de abiertos por el menor valor de  $f(n)$ .
- El objetivo lo encuentra **cuando el nodo objetivo está en la lista de abiertos**.



## ❖ Ejemplo Puzzle-8: Algoritmo A\*

---

- Resolver el problema con  $h_1(n)$  = número de piezas descolocadas.
- Resolver el problema con  $h_2(n)$  = suma de las distancias de Manhattan para cada pieza.

- $h_1$  vs  $h_2$  : **¿Qué heurístico es mejor?**

Informalmente:  $h_2$  es más preciso que  $h_1$ , porque  $h_2$  aproxima mejor la función de la realidad.

Formalmente: ver Propiedad más formado en propiedad 2 (a continuación).

- A\* encuentra la solución óptima para los dos casos anteriores. Pero...  
**¿encuentra siempre la solución óptima?**

# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 1 - Admisibilidad

- **Admisibilidad:** cuando encuentra una solución, ésta es óptima.

- Defino:

$h^*(n) \equiv$  coste mínimo REAL desde  $n$  hasta el objetivo (no lo conocemos).

$c^* \equiv$  coste de la solución óptima (no lo conocemos aún, lo conoceremos cuando resolvamos el problema de manera óptima).

- Supongamos:

- $h(n) = 0$  Dijkstra (no hay heurística, estrategia no informada (a ciegas) y  $h^*$  no lo conocemos, si lo conocemos no habría búsqueda.

- Defino **heurístico admisible** (también llamado optimista):

a)  $h(n) \leq h^*(n) \quad \forall n$

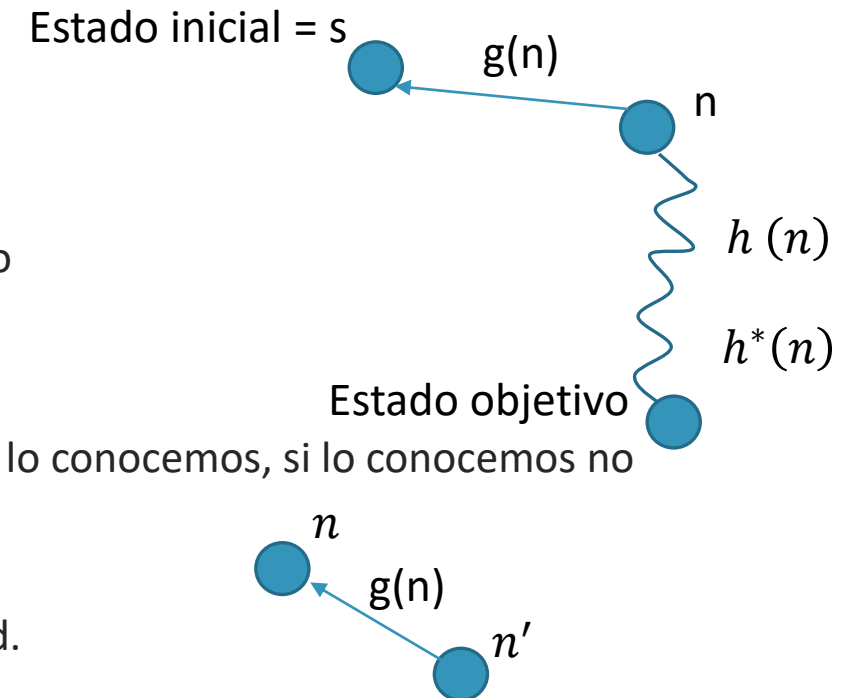
Será admisible si cumple esta propiedad.

- **Propiedad 1:**

$\forall \text{ arco}(n, n') \quad c(n, n') \geq \epsilon \geq 0$  y el heurístico es admisible, entonces A\* es admisible incluso en grafos infinitos.

- Si hay una solución, el algoritmo A\* **encuentra la óptima**.

- Si el camino crece infinitamente, el coste también (por eso se pone esta condición).





# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 1 - Admisibilidad

---

❖ Ejemplo Puzzle-8:

➤  $h_1$  ¿es un heurístico optimista?

Ej: si tengo 3 piezas mal colocadas, voy a tener que hacer 3 movimientos para resolver el puzle, o ¿puede que haga menos?

En cada movimiento sólo puedo mover una pieza.

Sí es optimista 😊

➤  $h_2$  ¿es un heurístico optimista?

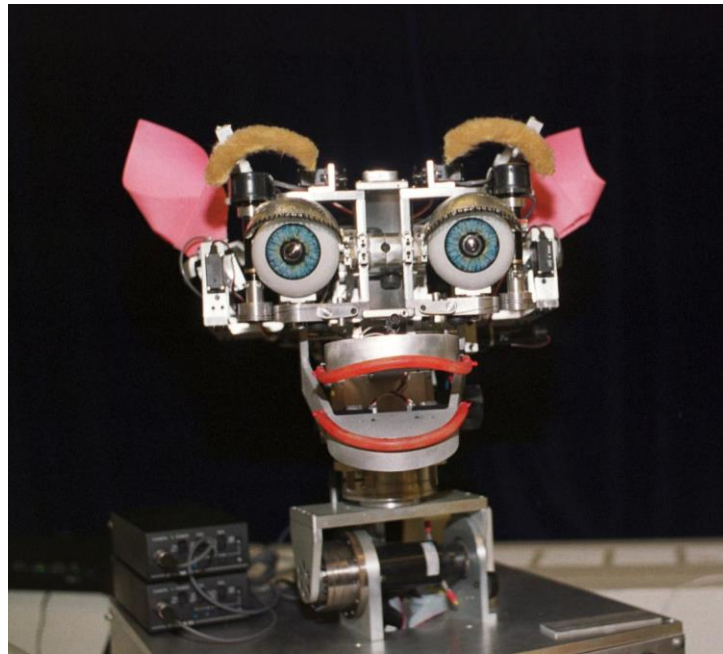
Si muevo una pieza, tendré que moverla tantas filas o columnas como de lejos esté.

Sí es optimista 😊

# Conclusión sobre heurística

---

Sé optimista y encontrarás el óptimo 😊



Kismet

# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 2

### Definición: **Propiedad más informado:**

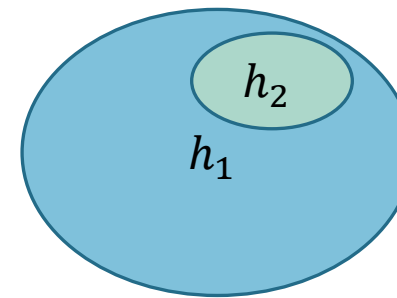
Dado 2 heurísticos admisibles (optimistas):

- $h_2$  está más informado que  $h_1$  si se cumple  $h_2(n) > h_1(n) \quad \forall n \neq E.obj.$

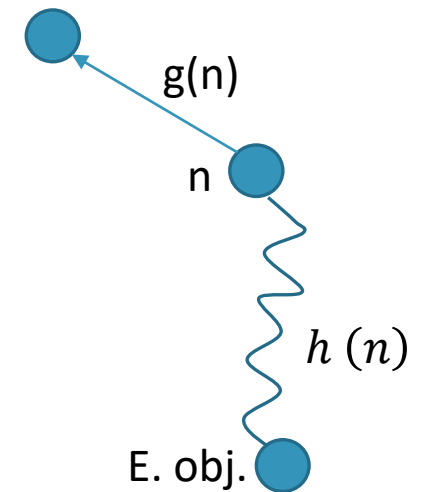
Si el heurístico es mejor en un nodo, ese heurístico es informado.

### Propiedad 2:

Si A\* selecciona  $n$  con  $h_2$ , también lo selecciona con  $h_1$ .

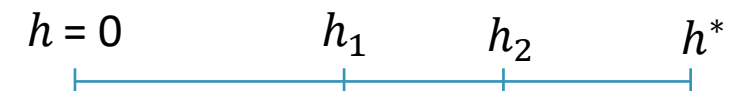


Estado inicial =  $s$



### ❖ Ejemplo Puzzle-8:

- $h_0 = 0$  (Dijkstra)
- $h_1$  = número piezas descolocadas.
- $h_2$  = suma de distancia de Manhattan de piezas mal colocadas.



# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 2

---

### ❖ Ejemplo Puzzle-8:

#### ➤ ¿Qué heurístico es el más informado?

- $h_0 = 0$  (*Dijkstra*)
- $h_1$  = número piezas descolocadas.
- $h_2$  = suma de distancia de Manhattan de piezas mal colocadas.

#### ➤ ¿ $h_2$ está más informado que $h_1$ ?

- No
- $h_2(n) \geq h_1(n) \quad \forall n$  Ante esta situación es mejor  $h_2$ , pero no podemos decir que sea más informado que  $h_1$

#### ➤ ¿Toda heurística siempre estará más informada que $h_0$ ?

No porque podemos definir una heurística que sea 0 antes de llegar al estado objetivo.

# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 3

- Definición: **Restricción de monotonía:**

- $h$  cumple la restricción de monotonía si  $h(n) \leq c + h(n')$

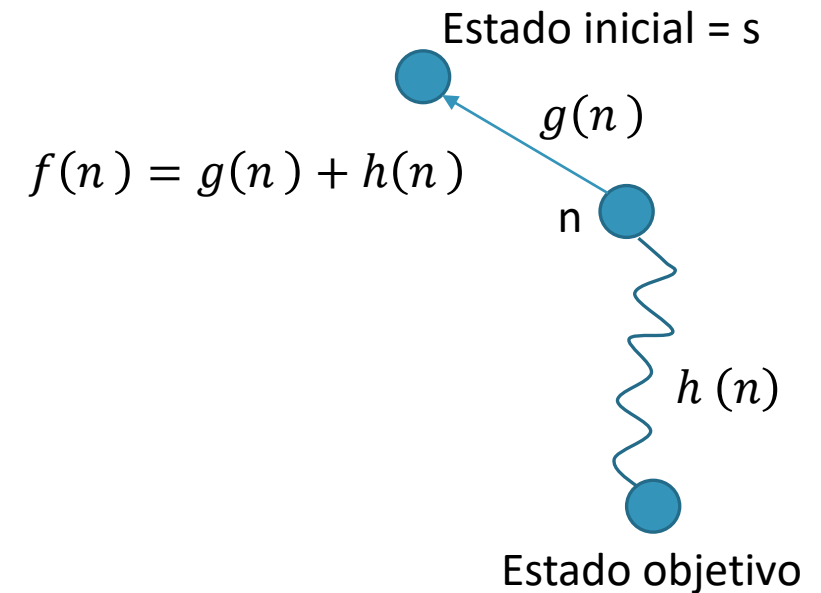
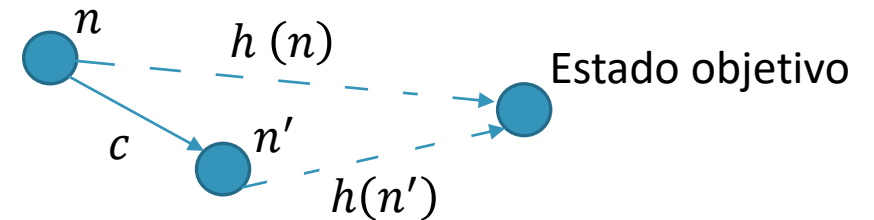
- Cuando esto se cumple, y tenemos un camino:

$s, n_1, n_2, n_3, \dots$  se cumple  $f(s) \leq f(n_1) \leq f(n_2) \leq f(n_3)$   
secuencia monótona no decreciente.

- **Propiedad 3:**

**Si  $h$  es monótono, A\* expande cada nodo como mucho una vez**  
(nunca encuentra un camino mejor a un nodo ya cerrado).

- Si  $h$  se obtuvo por relajación, entonces es monótono.
- Si es monótono entonces también es admisible (al revés no).
- Si hemos tenido que reabrir un nodo, no es monótono.



# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 3

---

- Forma de comprobar monotonía:  $h(n) - h(n') \leq c(n, n')$
- ❖ Ejemplo Puzzle-8 **comprobar monotonía de heurística**  $h_1$  = número de piezas mal colocadas (primer ejercicio hecho en clase). Suponemos que estamos en posición  $n$ , hay  $k$  piezas descolocadas y el coste de mover es 1.
  - ¿Qué puede ocurrir? Describo todos los casos posibles:
    - $p$  estaba descolocada y la coloco (acerco a solución)  
$$h(n) - h(n') = k - (k-1) = k - k + 1 = 1 \leq 1 \quad \text{Cumple monotonía}$$
    - $p$  estaba colocada y la descoloco (alejo de la solución)  
$$h(n) - h(n') = k - (k+1) = k - k - 1 = -1 \leq 1 \quad \text{Cumple monotonía}$$
    - $p$  estaba descolocada y lo sigue estando (me mantengo a la misma distancia).  
$$h(n) - h(n') = k - k = 0 \leq 1 \quad \text{Cumple monotonía}$$
  - Todos los casos posibles son monótonos, así que la heurística  $h_1$  es monótona.

# 4. Búsquedas informadas: Algoritmo A\*

## Propiedades: Propiedad 3

---

- Forma de comprobar monotonía:  $h(n) - h(n') \leq c(n, n')$
- ❖ Ejemplo Puzzle-8 comprobar **monotonía de heurística**  $h_2$  = distancia de Manhattan (segundo ejercicio hecho en clase). Suponemos que estamos en posición  $n$ , hay  $k$  piezas descolocadas y el coste de mover es 1.
  - ¿Qué puede ocurrir? Describo todos los casos posibles:
    - $p$  estaba descolocada y la coloco (acerco a solución)  
$$h(n) - h(n') = k - (k-1) = k - k + 1 = 1 \leq 1 \quad \text{Cumple monotonía}$$
    - $p$  estaba colocada y la descolo (alejo de la solución)  
$$h(n) - h(n') = k - (k+1) = k - k - 1 = -1 \leq 1 \quad \text{Cumple monotonía}$$
    - $p$  estaba descolocada y lo sigue estando (este caso no puede ocurrir con esta heurística).
  - Todos los casos posibles son monótonos, así que la heurística  $h_2$  es monótona.

# 4. Búsquedas informadas: Algoritmo A\*

## Conclusión tras las propiedades

---

1. Si el heurístico es monótono y admisible entonces A\* es el mejor que Dijkstra.
2. Esto me dice que se harán menos iteraciones que en Dijkstra (menor tiempo).
3. Función de evaluación: determina el criterio para continuar buscando:

Estrategia	Función evaluación
Dijkstra	$f(n) = g(n)$
Greedy	$f(n) = h(n)$
A*	$f(n) = g(n) + h(n)$

4. Cuando nos encontramos dos caminos a un nodo, elegir en función de  $f(n)$ .