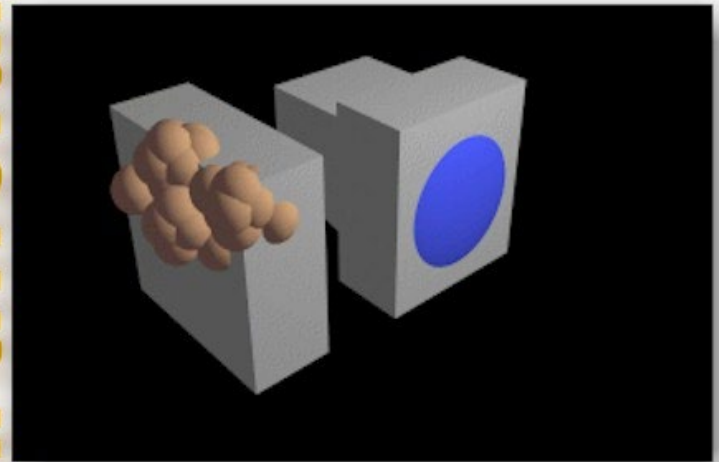


Tema 3: Búsqueda en espacios de estados



Inteligencia
Artificial



Objetivos

- Entender que la resolución de problemas en IA implica definir una representación del problema y un proceso de búsqueda de la solución.
- Conocer la representación de problemas basados en estados (estado inicial, objetivo y espacio de búsqueda) para ser resueltos con técnicas computacionales.
- Conocer las técnicas más representativas de búsqueda no informada en un espacio de estados (en profundidad, en anchura y sus variantes), y saber analizar su eficiencia en tiempo y espacio.
- Analizar las características de un problema dado y determinar si es susceptible de ser resuelto mediante técnicas de búsqueda. Decidir en base a criterios racionales la técnica más apropiada para resolverlo y saber aplicarla.
- Entender el concepto de heurística y analizar las repercusiones en la eficiencia en tiempo y espacio de los algoritmos de búsqueda.
- Conocer las técnicas más representativas de búsqueda informada en un espacio de estados (búsqueda local, algoritmo A*).

Estudia el tema en ...

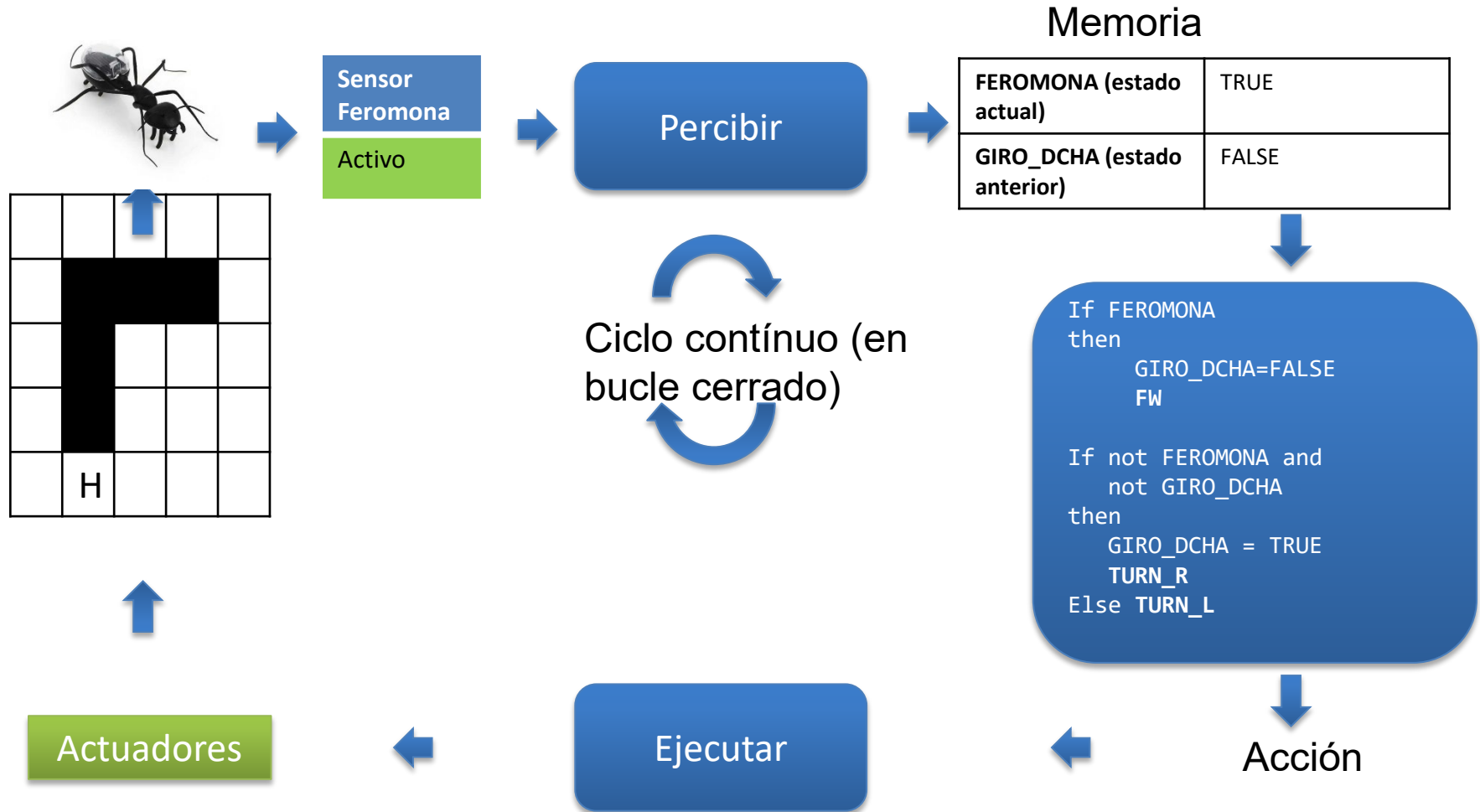
- Nils J. Nilsson, “*Inteligencia Artificial: Una nueva síntesis*”, Ed. Mc Graw Hill, 2000. pp. 53-62, 125-146, 163-174
- S. Russell, P. Norvig, *Artificial Intelligence: A modern Approach*, Tercera Edición, Ed. Pearson, 2010.

Contenido

- Diseño de un agente deliberativo: búsqueda
- Sistemas de búsqueda y estrategias
- Búsqueda sin información
- Búsqueda con información
- Problemas descomponibles y búsqueda

Limitaciones de un agente reactivo

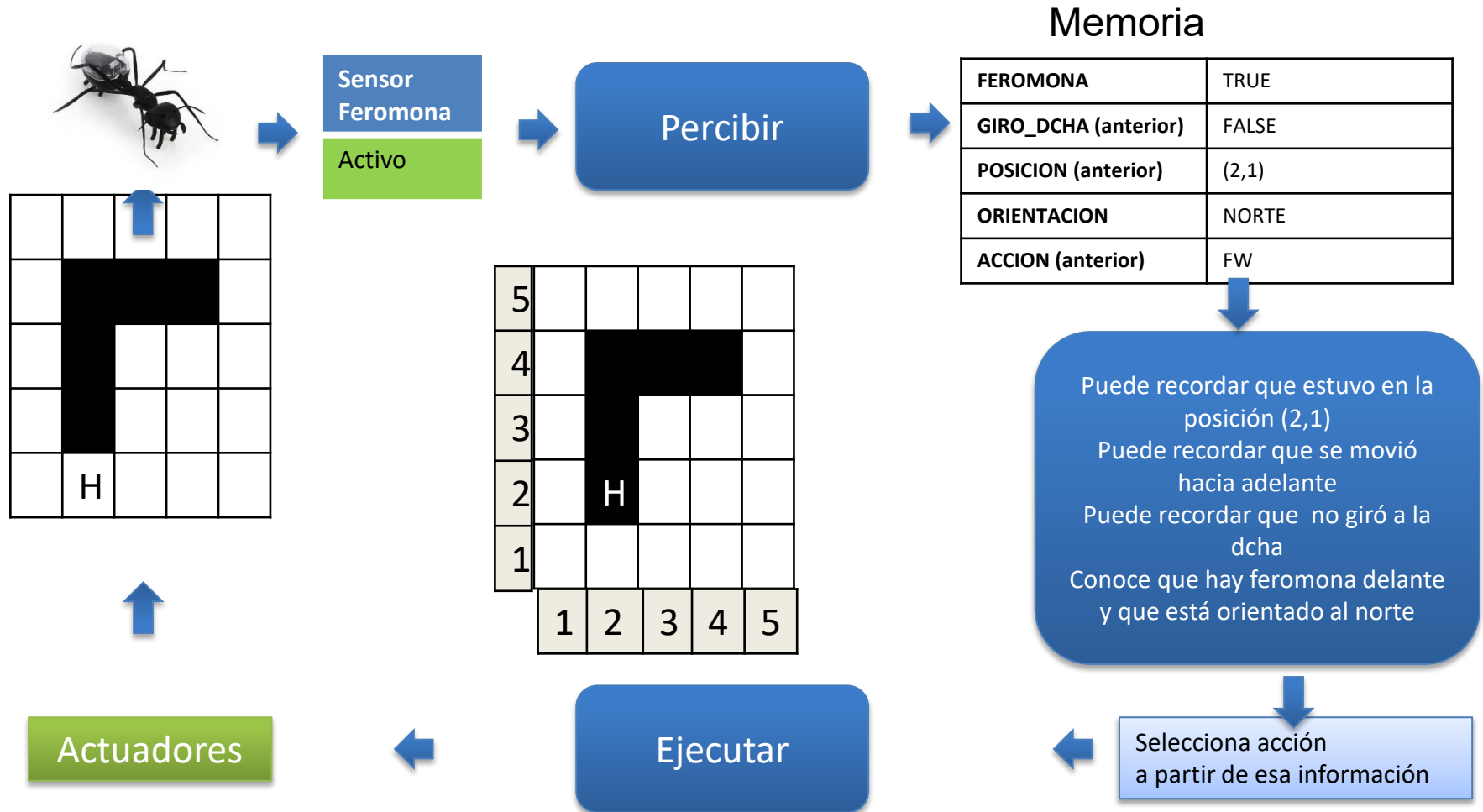
Limitaciones de un agente reactivo



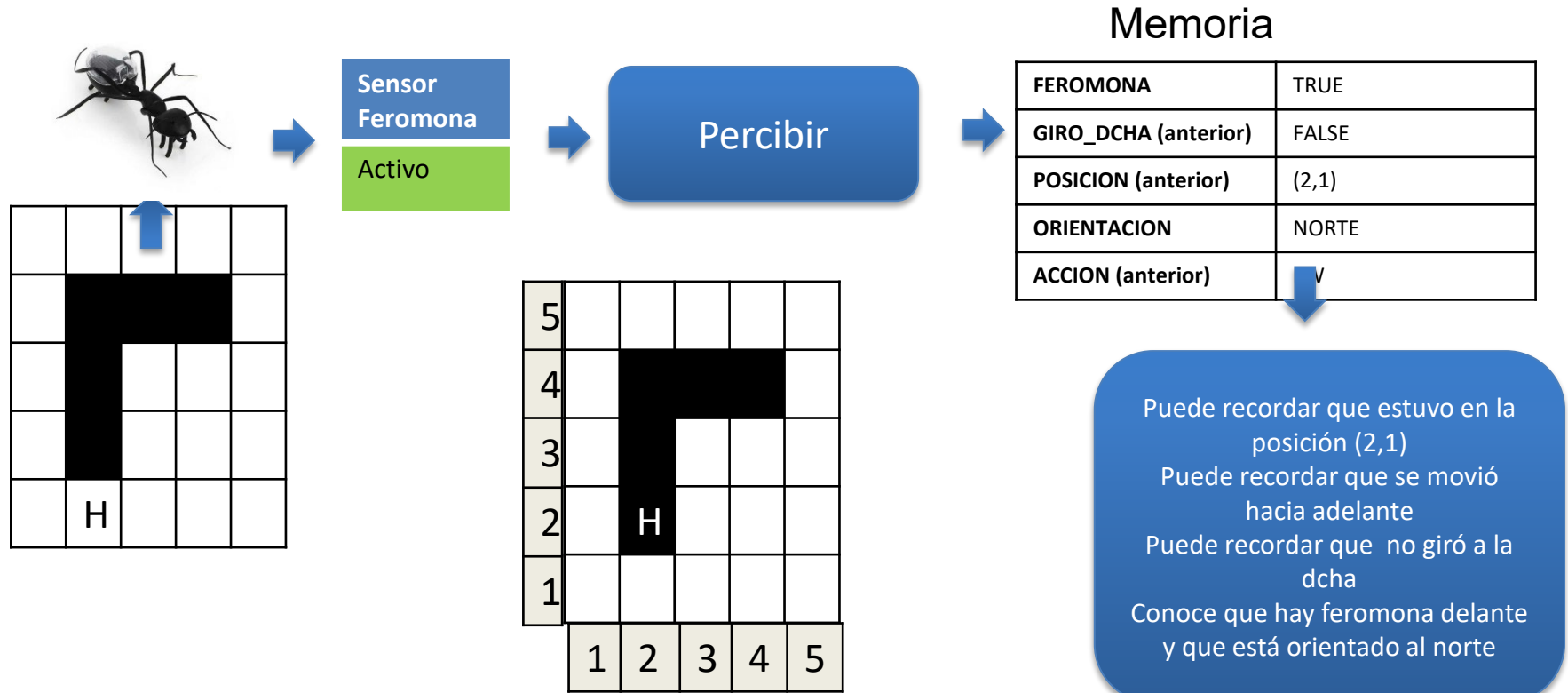
Limitaciones de un agente reactivo

Podemos complicar el
modelo de memoria
interna todo lo que
queramos

Limitaciones de un agente reactivo

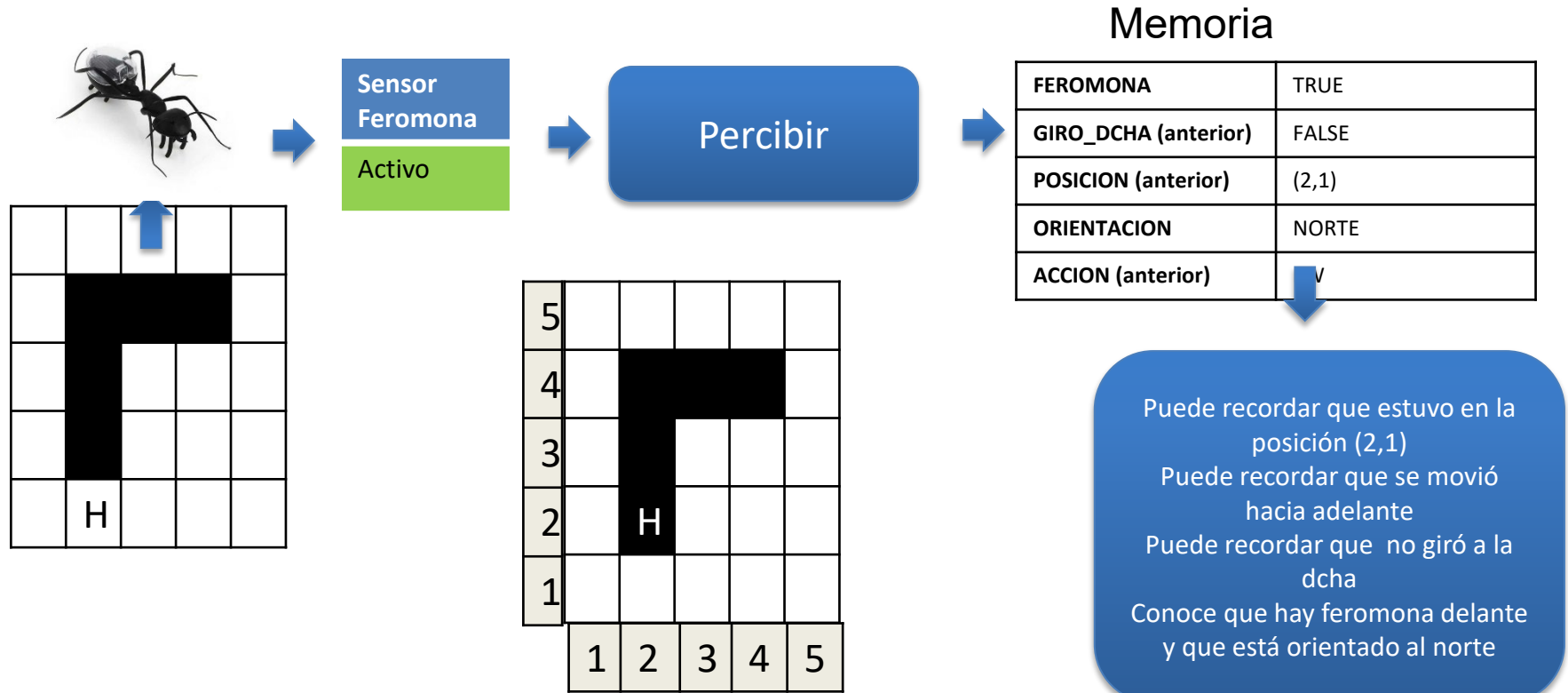


Limitaciones de un agente reactivo



No tiene capacidad para conocer de forma anticipada cómo se transforma el mundo mediante la ejecución de sus acciones

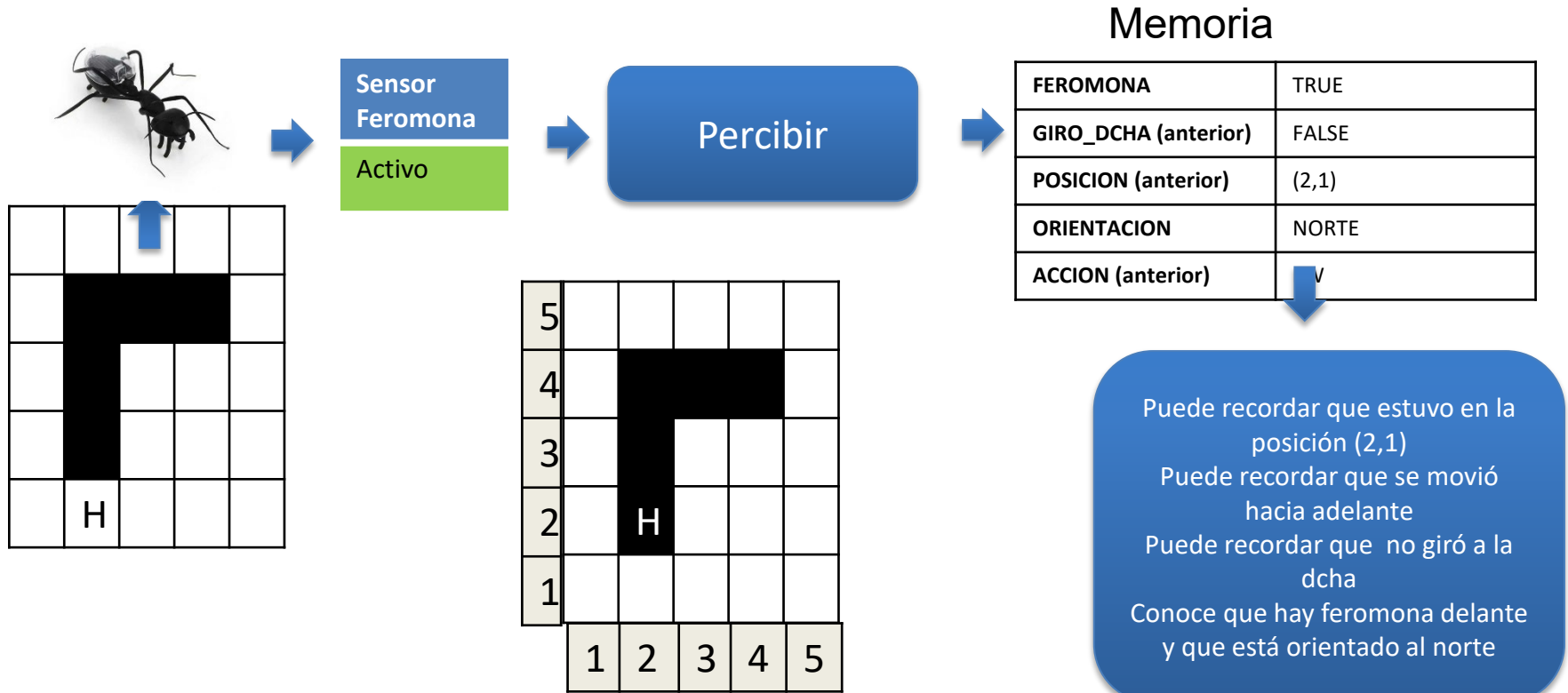
Limitaciones de un agente reactivo



No tiene capacidad para conocer de forma anticipada cómo se transforma el mundo mediante la ejecución de sus acciones

No puede saber que si está orientado al norte y está en la casilla (2,2) acaba estando en la casilla (2,3), y por tanto **no puede conocer qué alternativas de movimiento tiene.**

Limitaciones de un agente reactivo

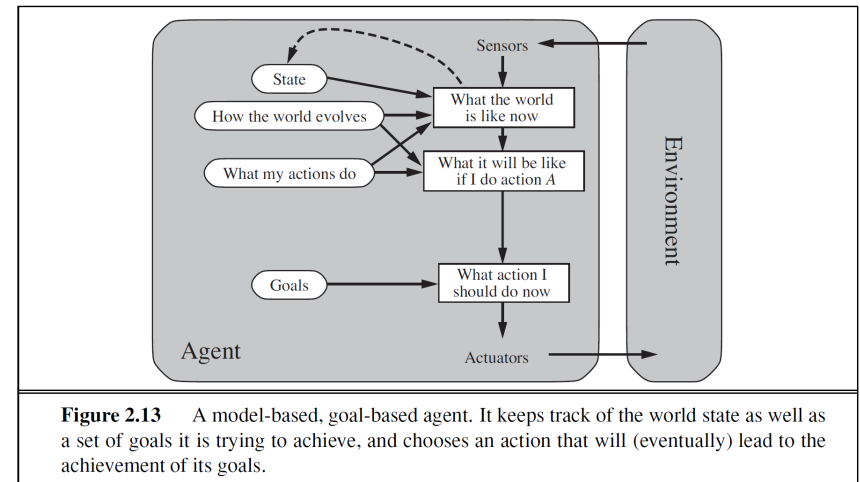


No tiene capacidad para conocer de forma anticipada cómo se transforma el mundo mediante la ejecución de sus acciones

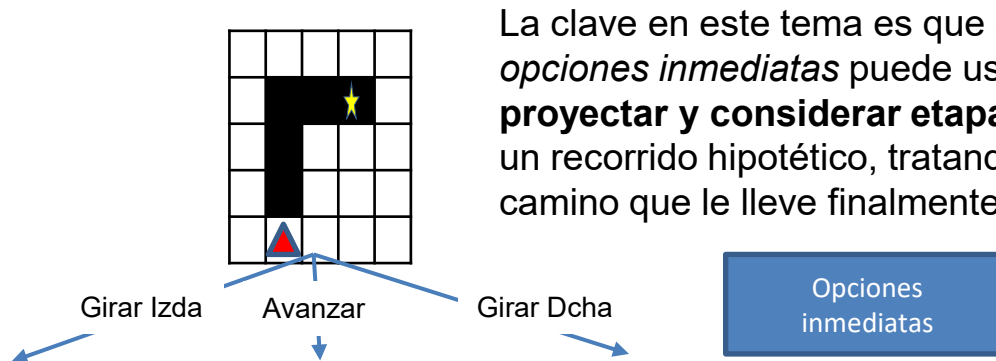
Además, no tiene capacidad para conocer cuál es su meta u objetivo para tomar una decisión correcta entre varias alternativas.

Diseño de un agente deliberativo

- El agente dispone de un modelo del mundo en el que habita
 - Por ejemplo un mapa del entorno y/o qué objetos hay en el entorno y qué propiedades tienen esos objetos
- El agente dispone de un modelo de los efectos de sus acciones sobre el mundo
 - P.e. sabe **anticipadamente** en qué casilla se encontrará cuando aplica una acción
- El agente es capaz de razonar sobre esos modelos para decidir que hacer para conseguir un objetivo
 - P.e., si la hormiga sabe que hay comida en (4,4), y conoce las consecuencias de sus acciones, en cada casilla puede tomar una decisión correcta para alcanzar su objetivo: “ir a la casilla (4,4) para comer”.

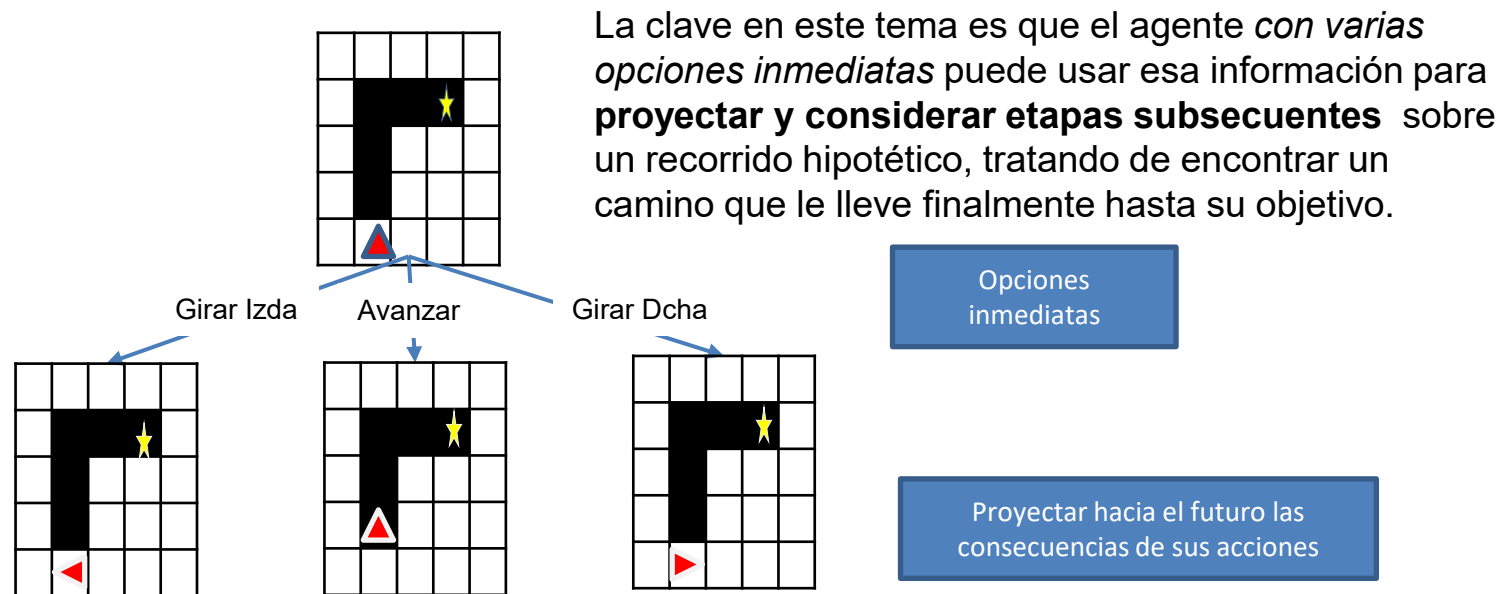


Diseño de un agente deliberativo



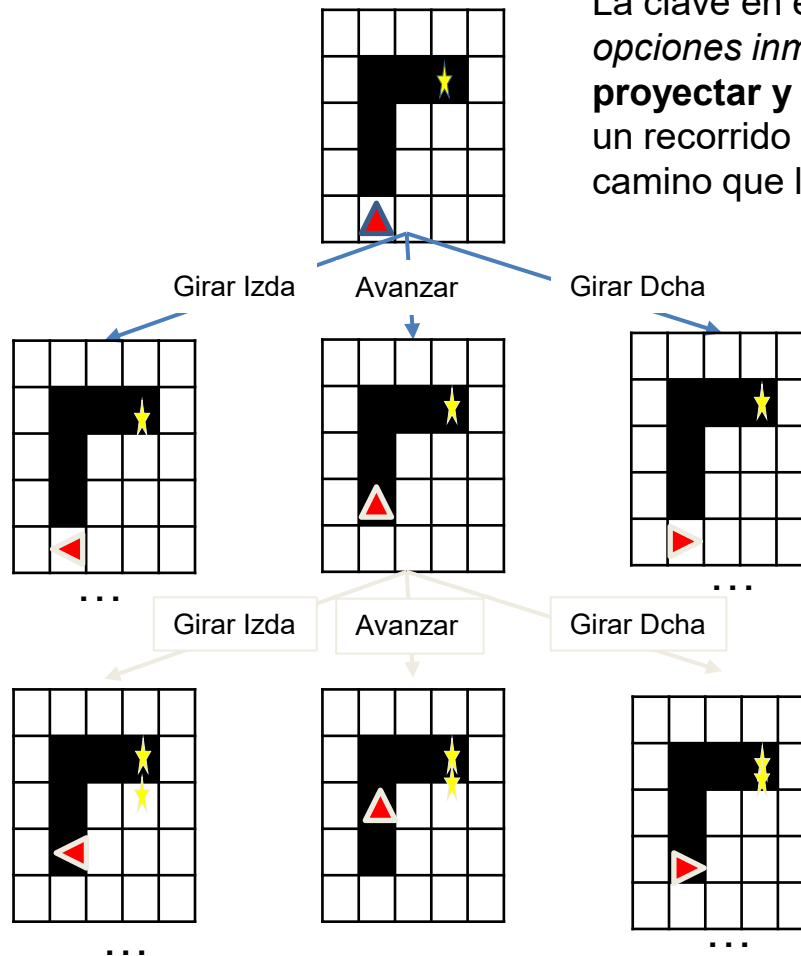
La clave en este tema es que el agente *con varias opciones inmediatas* puede usar esa información para **proyectar y considerar etapas subsecuentes** sobre un recorrido hipotético, tratando de encontrar un camino que le lleve finalmente hasta su objetivo.

Diseño de un agente deliberativo



Diseño de un agente deliberativo

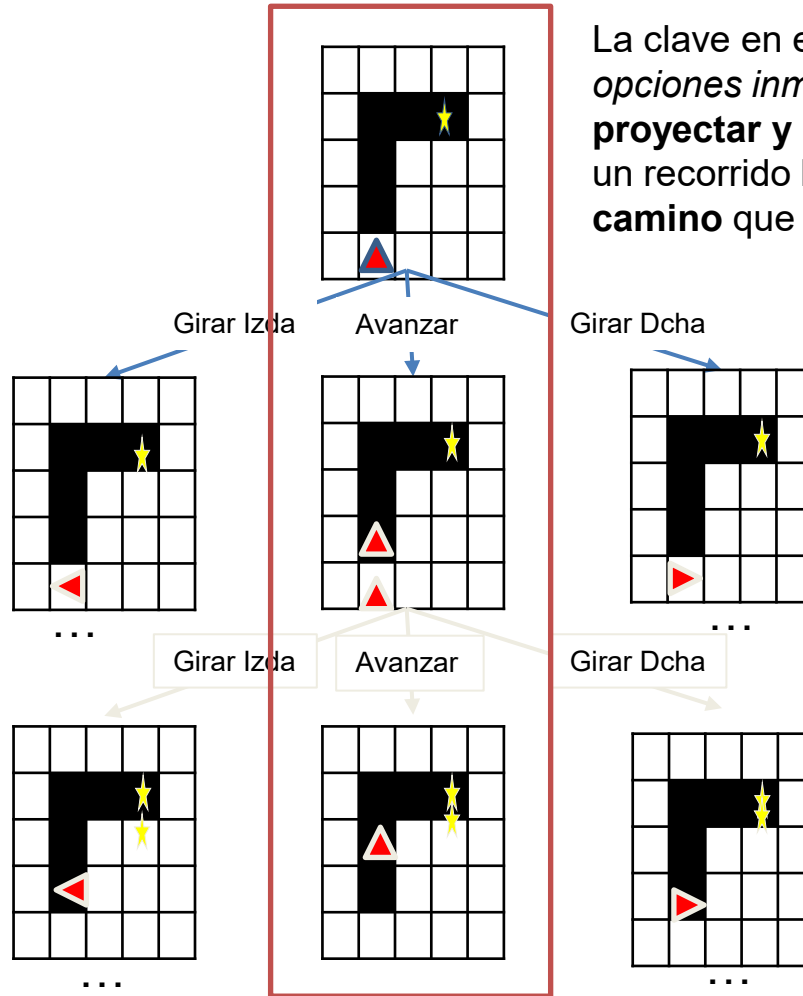
La clave en este tema es que el agente *con varias opciones inmediatas* puede usar esa información para **proyectar y considerar etapas subsecuentes** sobre un recorrido hipotético, tratando de encontrar un camino que le lleve finalmente hasta su objetivo.



Opciones inmediatas

Proyectar hacia el futuro las consecuencias de sus acciones

Diseño de un agente deliberativo



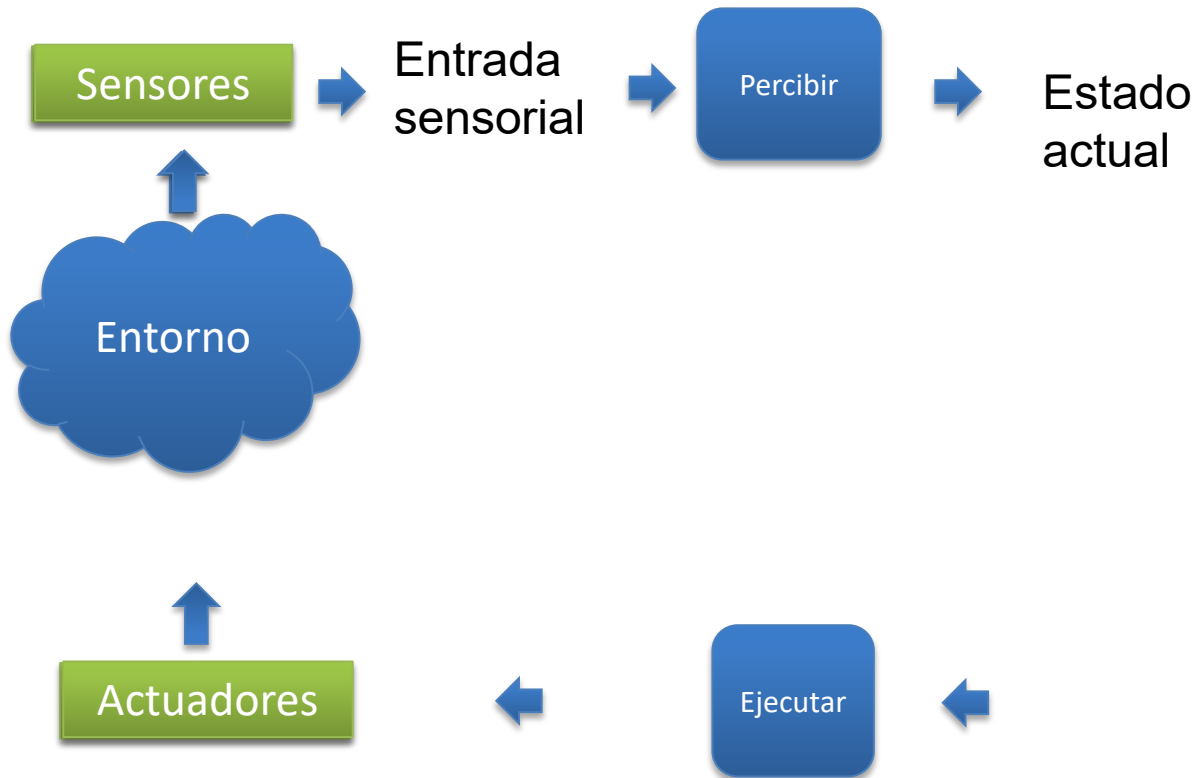
La clave en este tema es que el agente *con varias opciones inmediatas* puede usar esa información para **proyectar y considerar etapas subsecuentes** sobre un recorrido hipotético, tratando de **encontrar un camino** que le lleve finalmente hasta su objetivo.

Opciones inmediatas

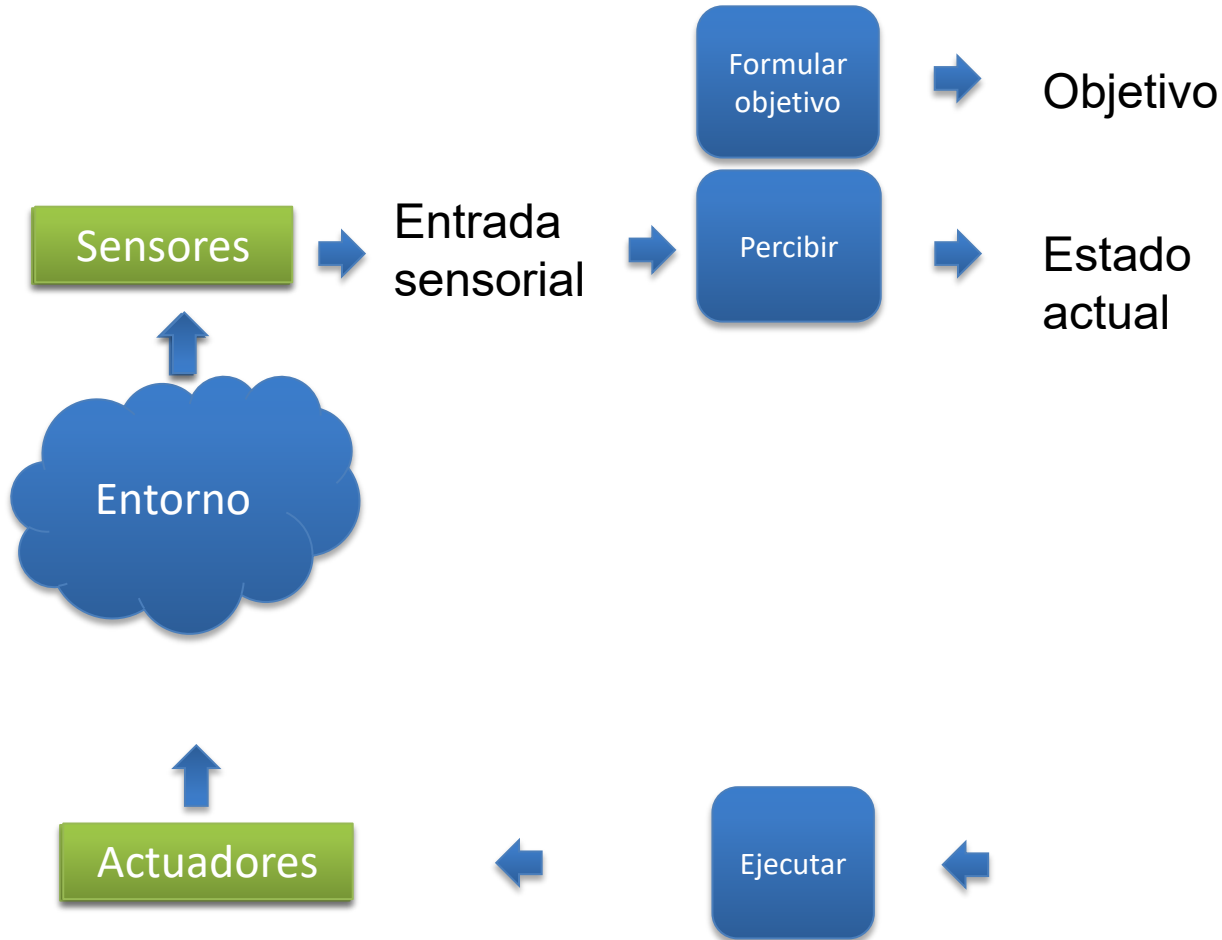
Proyectar hacia el futuro las consecuencias de sus acciones

Encontrar un camino

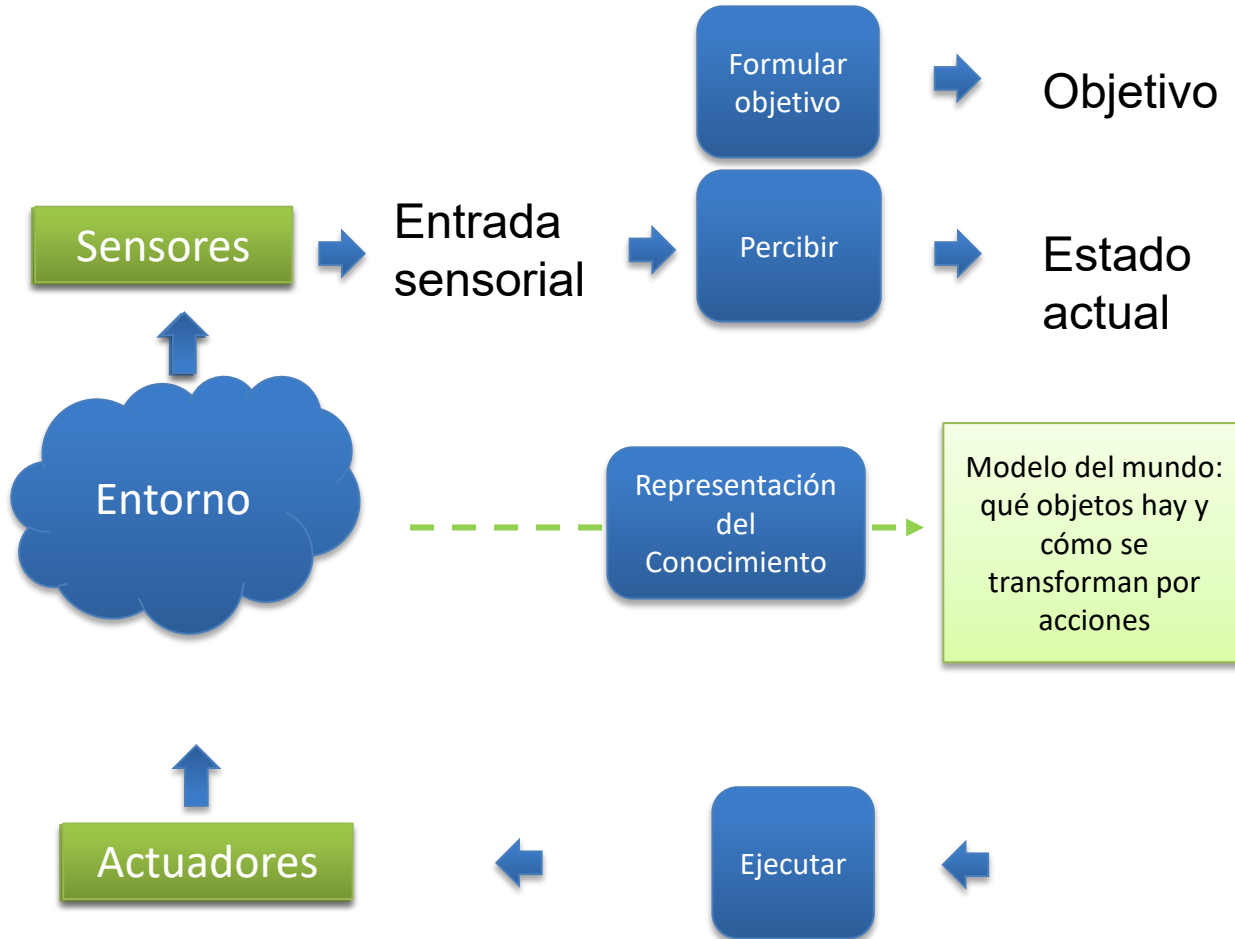
Agentes deliberativos



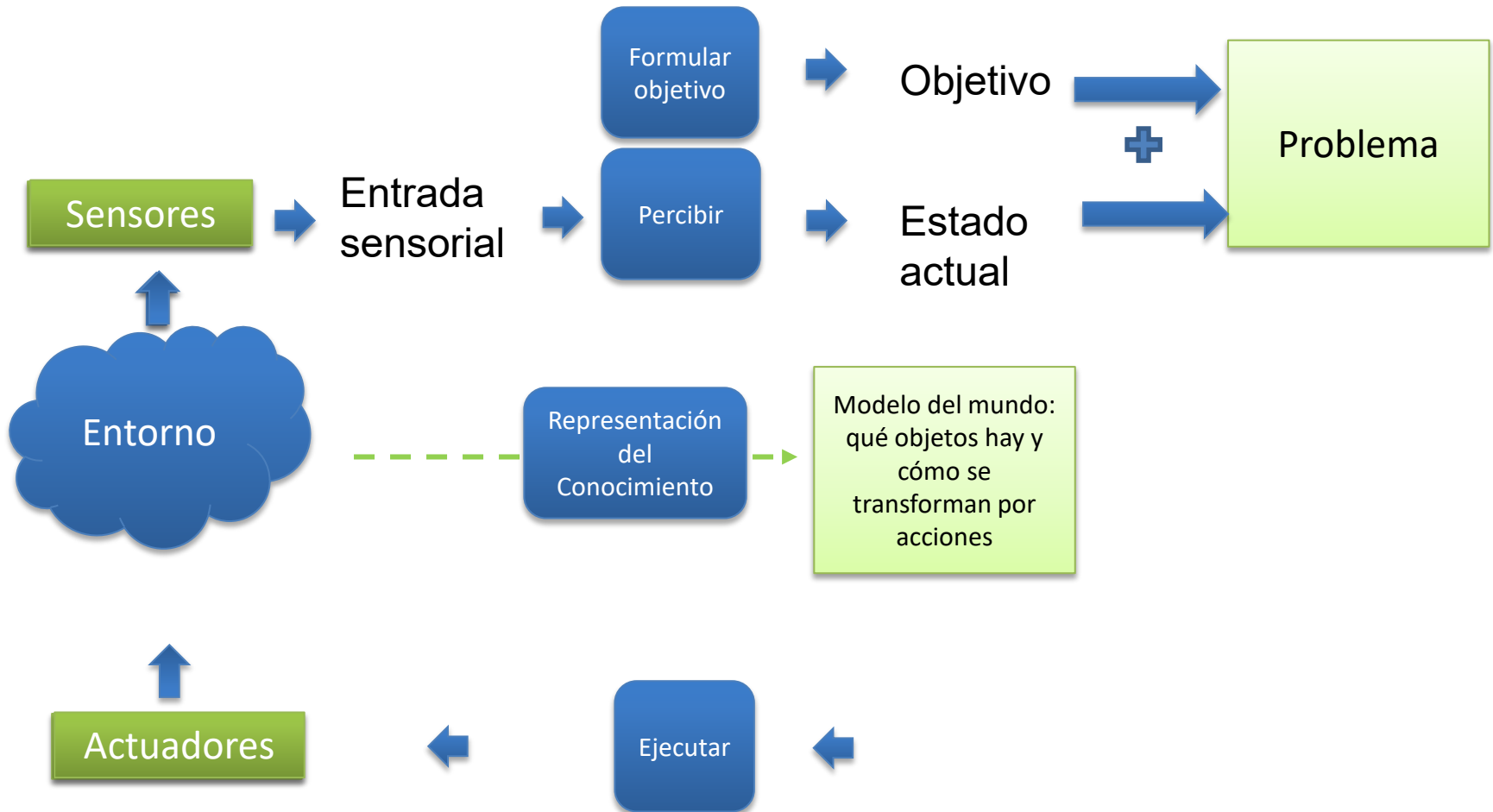
Agentes deliberativos



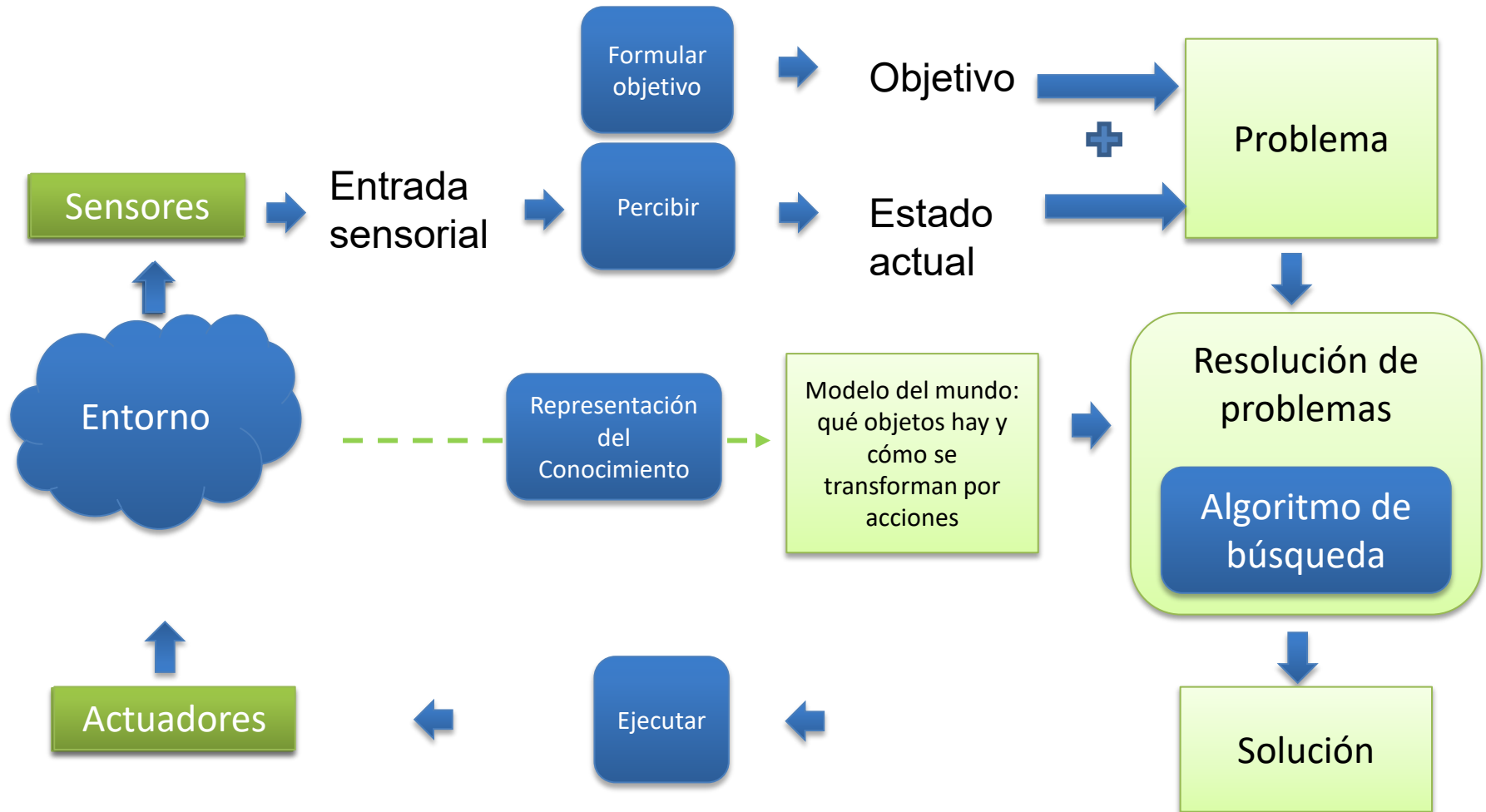
Agentes deliberativos



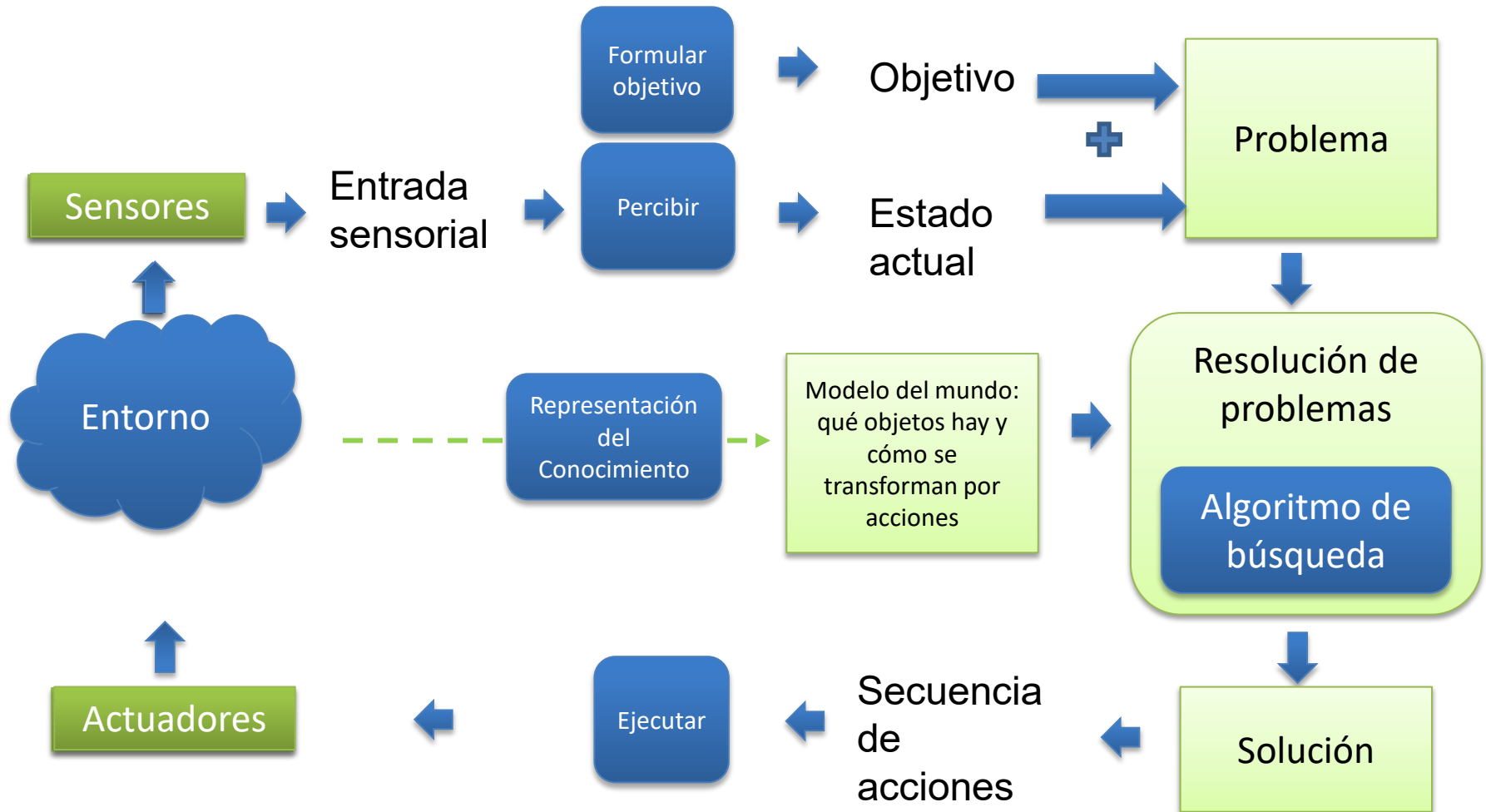
Agentes deliberativos



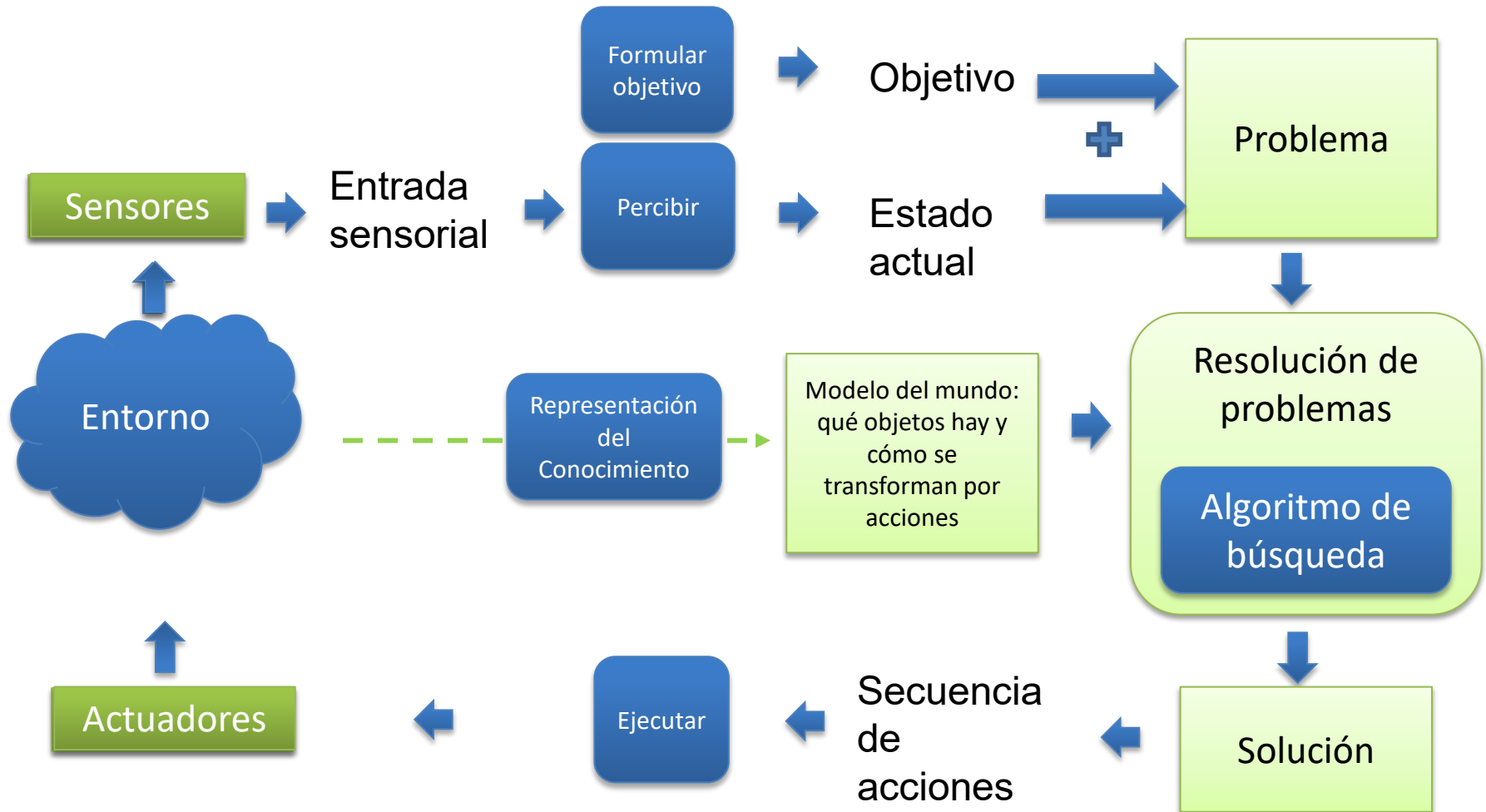
Agentes deliberativos





Agentes deliberativos



Arquitectura percepción/planificación/actuación





Componentes de la descripción de un problema

4				
3				
2				
1				
	1	2	3	4

Un agente está situado en un mundo cuadrado de dimensiones 4x4, inicialmente en la casilla (2,2). El agente se puede mover a las casillas vecinas que sean transitables. El agente debe alcanzar el premio que está situado en la casilla (4,4). ¿Cómo debe actuar el agente para alcanzar el premio?

Componentes de la descripción de un problema

4				
3				
2				
1				
	1	2	3	4

¿Un agente reactivo podría solucionarlo?



¿Y si le cambiamos el objetivo?

¿Y si le cambiamos el mapa?

Un agente está situado en un mundo cuadrado de dimensiones 4x4, inicialmente en la casilla (2,2). El agente se puede mover a las casillas vecinas que sean transitables. El agente debe alcanzar el premio que está situado en la casilla (4,4). ¿Cómo debe actuar el agente para alcanzar el premio?

Componentes de la descripción de un problema

- Estado inicial
- Objetivo
- Acciones
- Efectos de las acciones
- Espacio de estados
- Comprobación de objetivo
- Costo de las acciones

4				
3				
2				
1				
	1	2	3	4

Un agente está situado en un mundo cuadrulado de dimensiones 4x4, inicialmente en la casilla (2,2). El agente se puede mover a las casillas vecinas que sean transitables. El agente debe alcanzar el premio que está situado en la casilla (4,4). ¿Cómo debe actuar el agente para alcanzar el premio?

Componentes de la descripción de un problema

- **Estado inicial** : el agente está en la casilla (2,2)

- Predicado $\text{en}(x,y)$

- $\text{en}(2,2)$

- C++



- ```
struct posicion {x,y:int}
```

- ```
posicion estado_inicial ={2,2}
```

- ...

El estado inicial puede percibirse.

Modelo descriptivo!



4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

- **Estado objetivo:** el agente **deberá** estar en (4,4)
 - Predicado: $\text{en}(4,4)$
 - C++

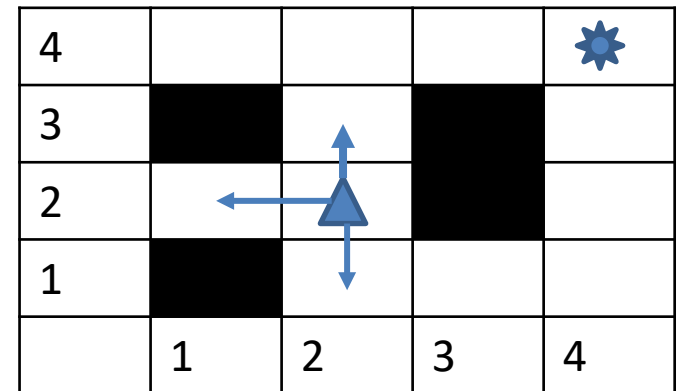
```
struct posicion {x,y:int}
posicion estado_final ={4,4}
```
 - ...

El objetivo puede proporcionarse como entrada al agente (un humano) o bien podemos tener un agente pro-activo que sea capaz formular objetivos para poder llegar a la autonomía plena.

4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

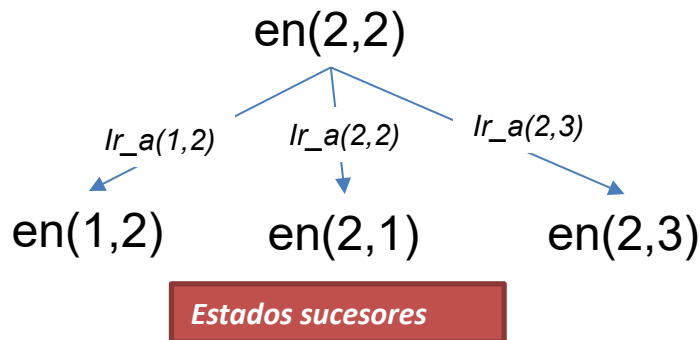
- **Acciones:** repertorio de acciones del agente
 - Para cada estado particular podemos representar las acciones que se pueden ejecutar
 - P.e: en el estado inicial
 - Ir_a(2,3)
 - Ir_a(2,1)
 - Ir_a(1,2)







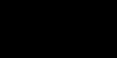
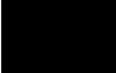



Componentes de la descripción de un problema

Carácter deliberativo

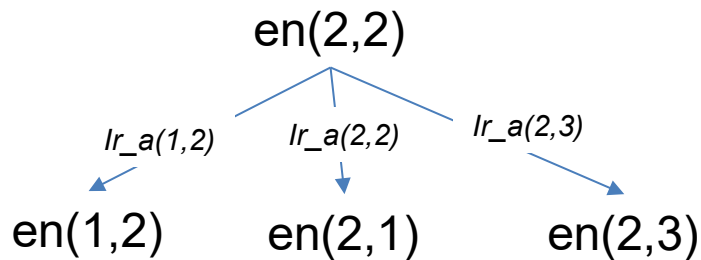
- **Efectos de las acciones:** se denomina también **modelo de transición**, una forma de especificar el estado resultante de aplicar una acción en un estado dado.
 - Para cada estado particular podemos conocer las acciones que se pueden ejecutar y su **Efecto**: estado resultante.



4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

- **Espacio de estados:** el conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones.

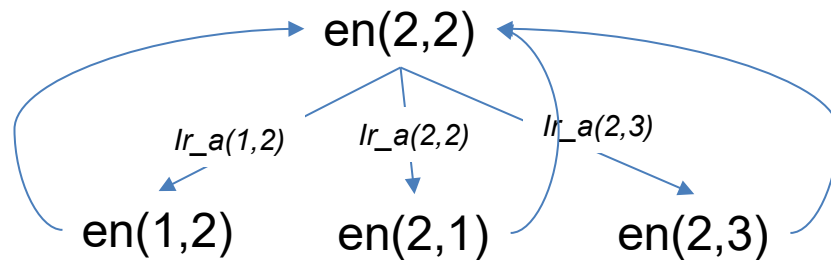







El estado inicial, el objetivo y el modelo de transición definen el espacio de estados de un problema

4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

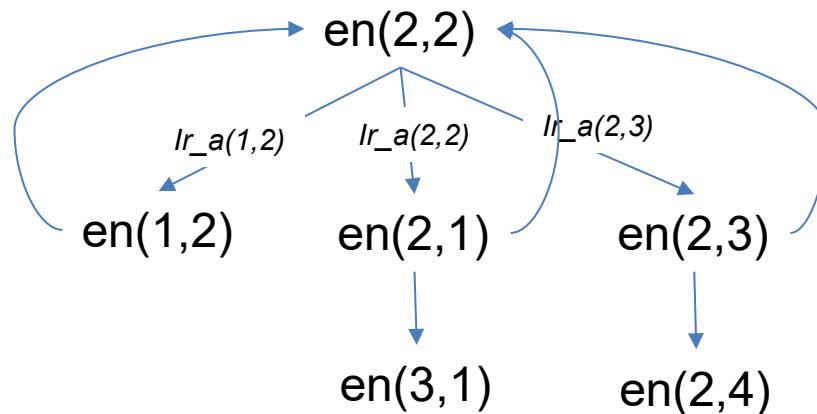
- **Espacio de estados:** el conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones.



4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

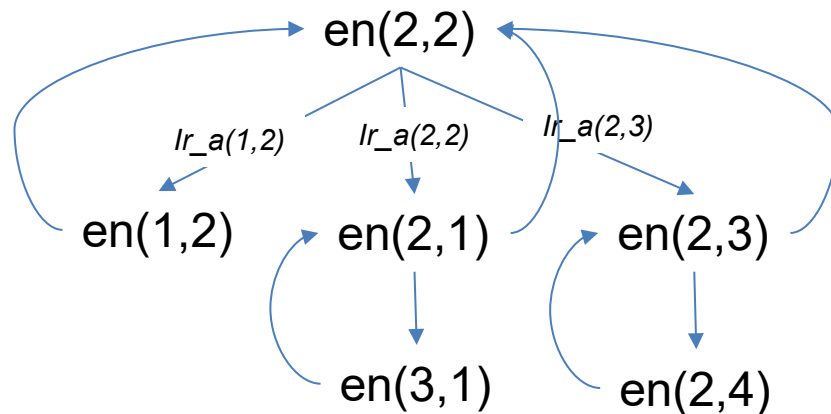
- **Espacio de estados:** el conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones.



4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

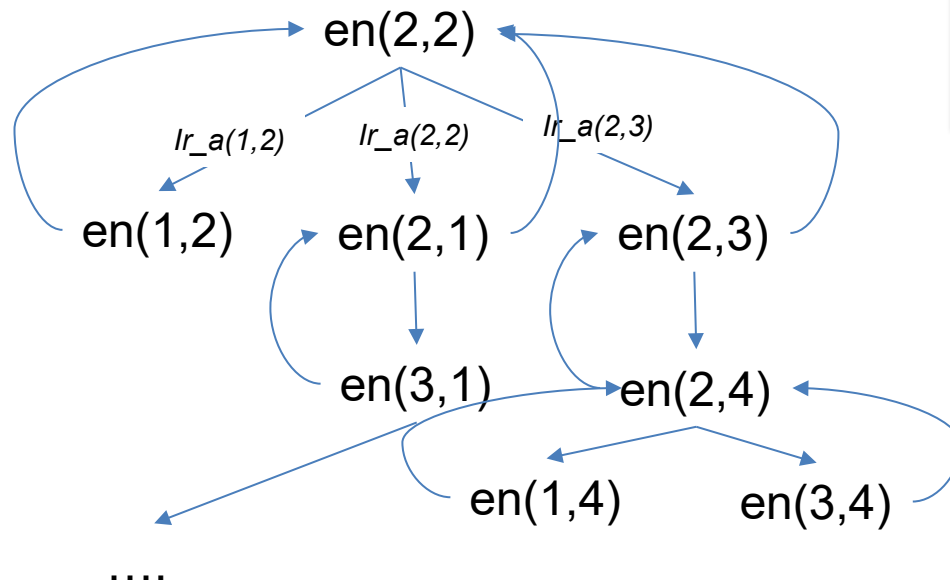
- **Espacio de estados:** el conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones.







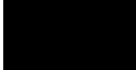

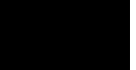


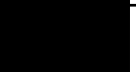
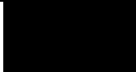


4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

- **Espacio de estados:** el conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones.



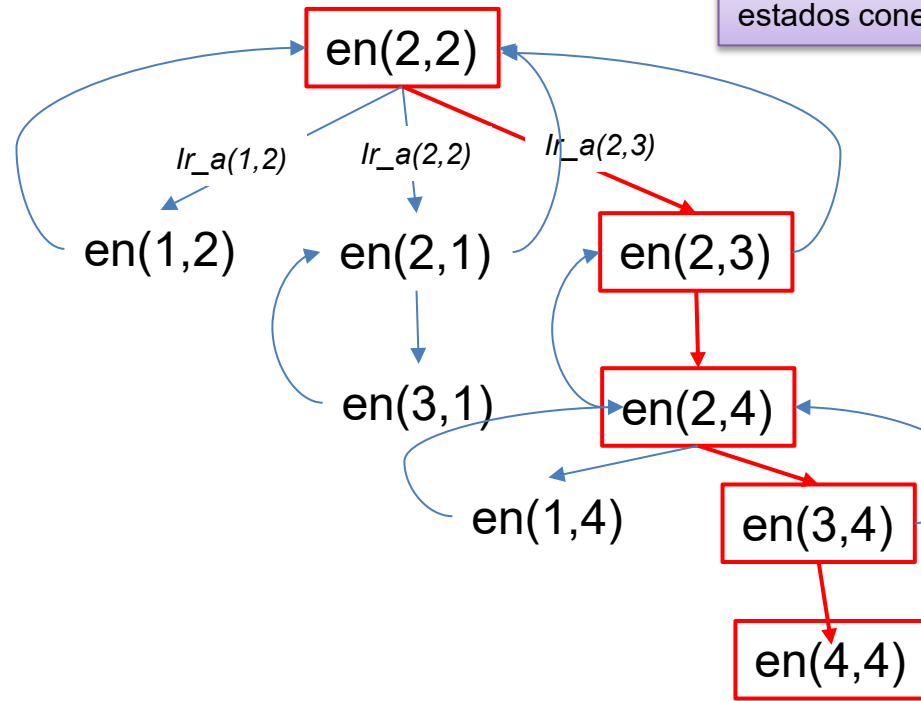
El espacio de estados TIENE LA FORMA de un grafo dirigido en el que cada nodo representa una casilla y cada arco una acción de movimiento entre casillas.

4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

- **Espacio de estados:** el conjunto de estados alcanzables desde el estado inicial mediante una secuencia de acciones.

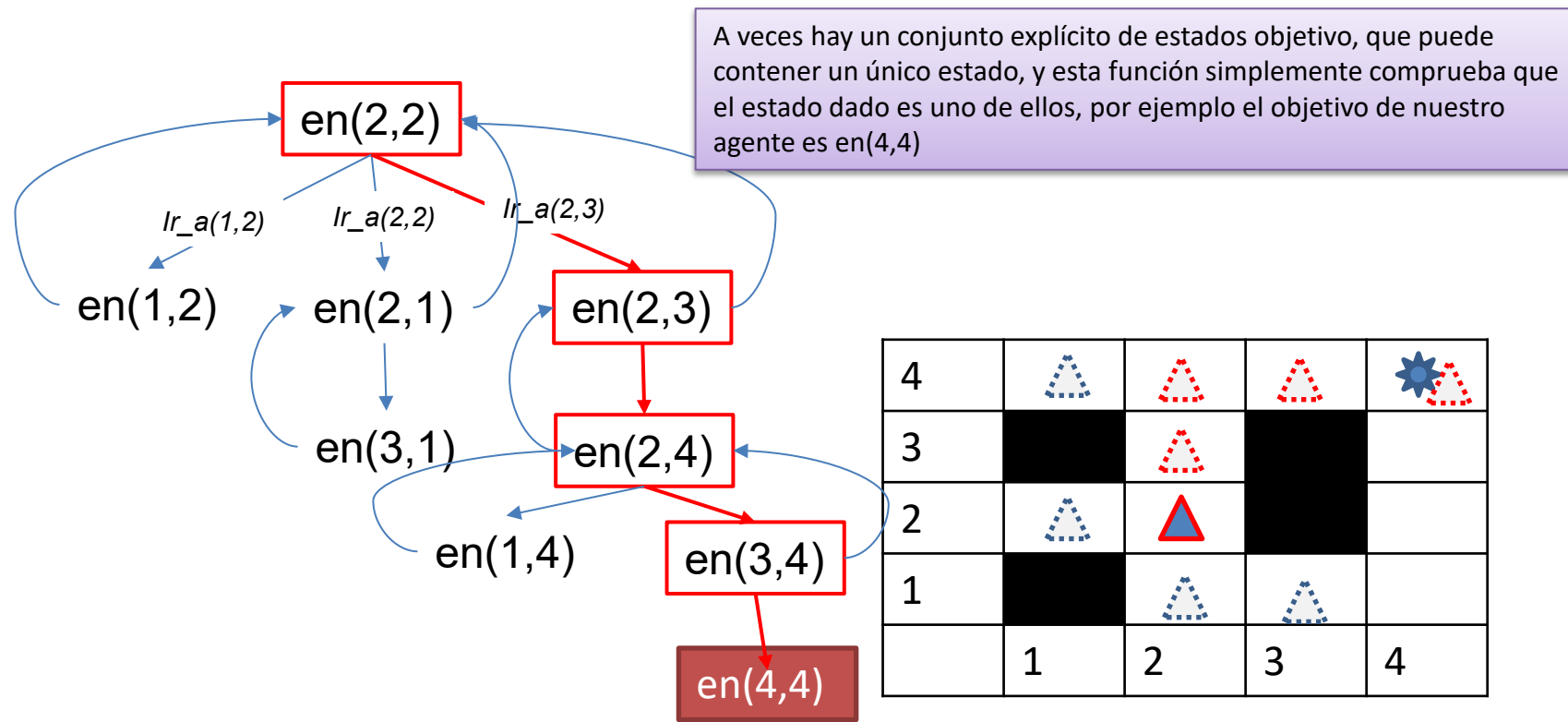
Un **camino** en el espacio de estados es una secuencia de estados conectados por una secuencia de acciones.



4				
3				
2				
1				
	1	2	3	4

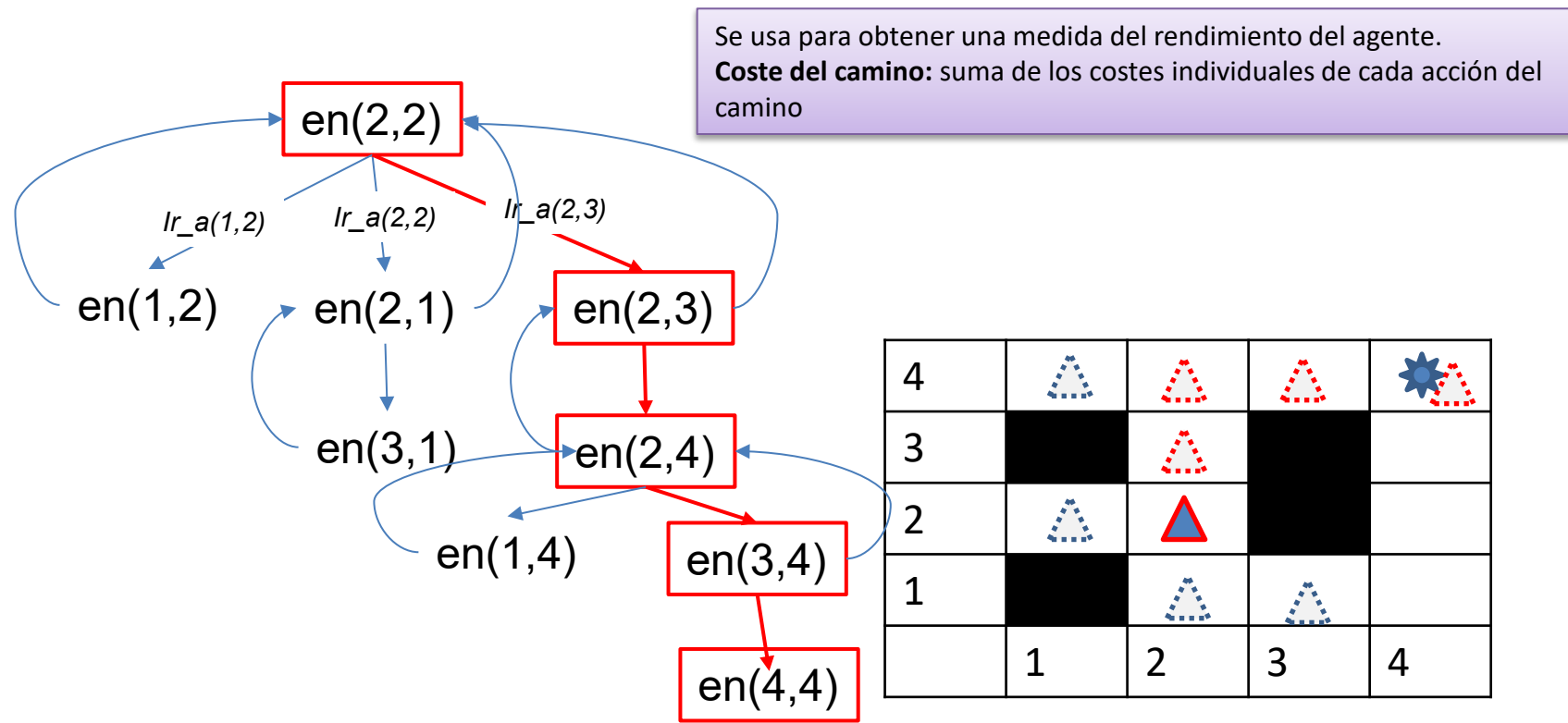
Componentes de la descripción de un problema

- **Comprobación de objetivo:** una función para comprobar que un estado dado es objetivo.



Componentes de la descripción de un problema

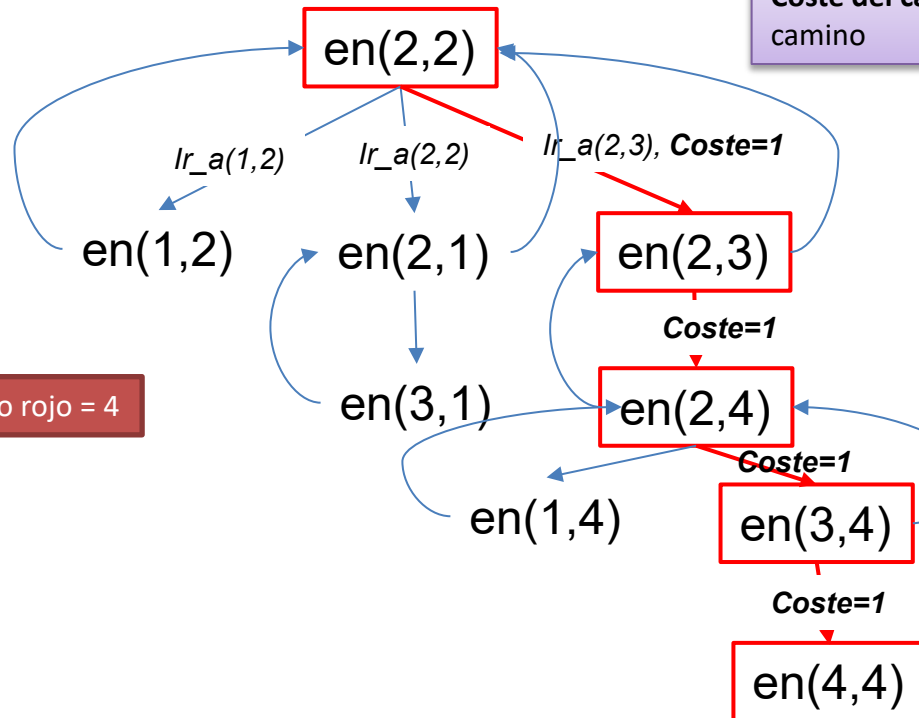
- **Coste del camino:** una función que asigna un coste numérico a cada camino.



Componentes de la descripción de un problema

- **Coste del camino:** una función que asigna un coste numérico a cada camino.

Se usa para obtener una medida del rendimiento del agente.
Coste del camino: suma de los costes individuales de cada acción del camino

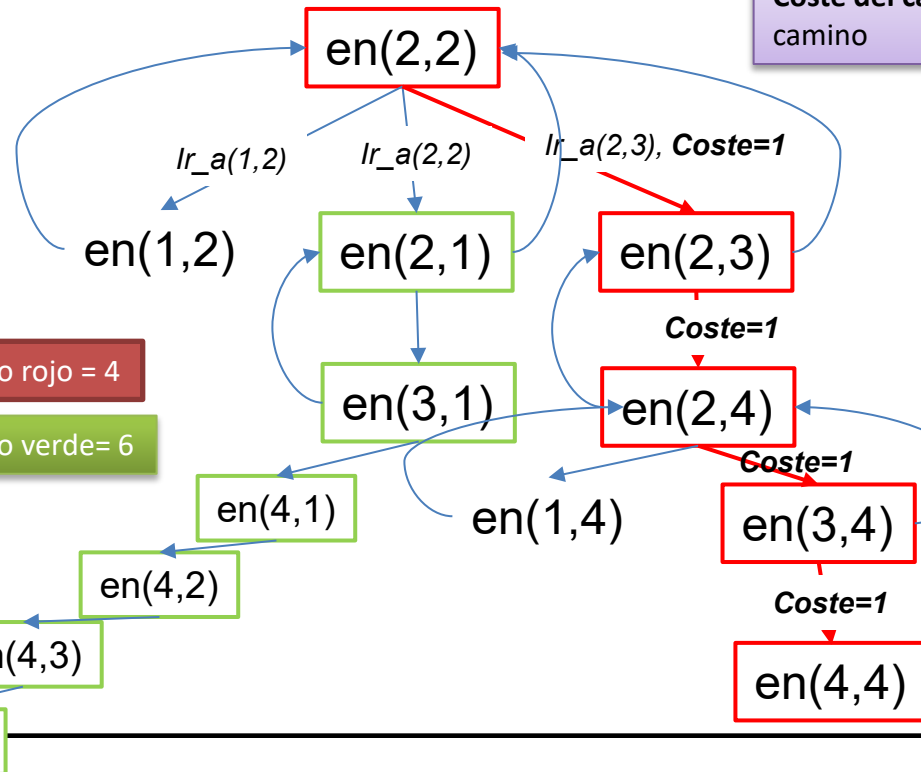


4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

- **Coste del camino:** una función que asigna un coste numérico a cada camino.

Se usa para obtener una medida del rendimiento del agente.
Coste del camino: suma de los costes individuales de cada acción del camino



4				
3				
2				
1				
	1	2	3	4

Componentes de la descripción de un problema

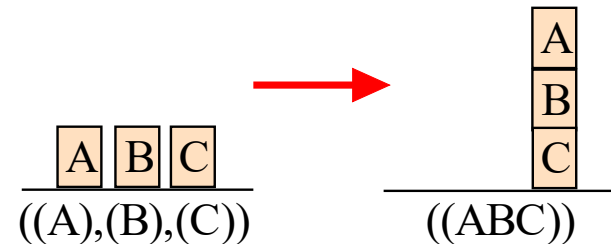
- Con estos elementos podemos definir un problema y podemos representarlo todo en una única estructura que proporcionamos a un algoritmo de resolución de problemas. Por tanto,
- una **solucion** a un problema es una secuencia de acciones que empieza en el estado inicial y permite alcanzar el estado objetivo.
- En los problemas en los que las acciones tengan un coste podemos hablar de **solución óptima** que consiste en el camino con el coste más bajo de entre todas las posibles soluciones

La búsqueda en un espacio de estados

- **IA = Representación del conocimiento + Búsqueda**
 - Estado, Estado Inicial, Estado Objetivo, Acciones, Coste, Espacio de estados
 - Representación del conocimiento a través de las acciones del agente
- Búsqueda en el espacio de estados
 - Resolución del problema mediante proyección de las distintas acciones y búsqueda del camino solución.

El mundo de bloques

- Supongamos un mundo cuadriculado con 3 bloques **A**, **B**, **C**.
- Inicialmente, todos los bloques están en el suelo.
- El objetivo es apilar los bloques de modo que **A** quede sobre **B**, **B** quede sobre **C**, y **C** esté en el suelo.
- En cada momento, se dispone de la operación **mover(x,y)** para poner **x** sobre **y**, donde **x**=**{A, B, C}** e **y**=**{A, B, C, Suelo}**.
- En cada momento, se conoce el estado del sistema. Lo modelamos con una secuencia de listas de objetos sobre objetos. Inicialmente, el estado es **((A), (B), (C))** y se desea llegar al estado **((ABC))**.
- Asumimos que se descartan los **operadores imposibles mover(A,A), mover(B,B), mover(C,C), etc.**, para cada estado.

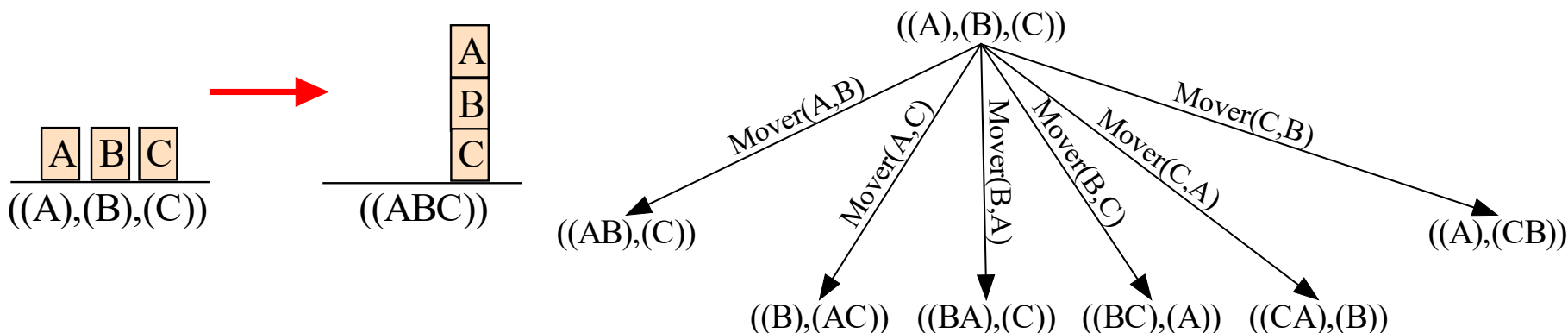


Descripción de un problema

- Estados
- Estado inicial
- Acciones
- Objetivo
- Costo de las acciones

El mundo de bloques

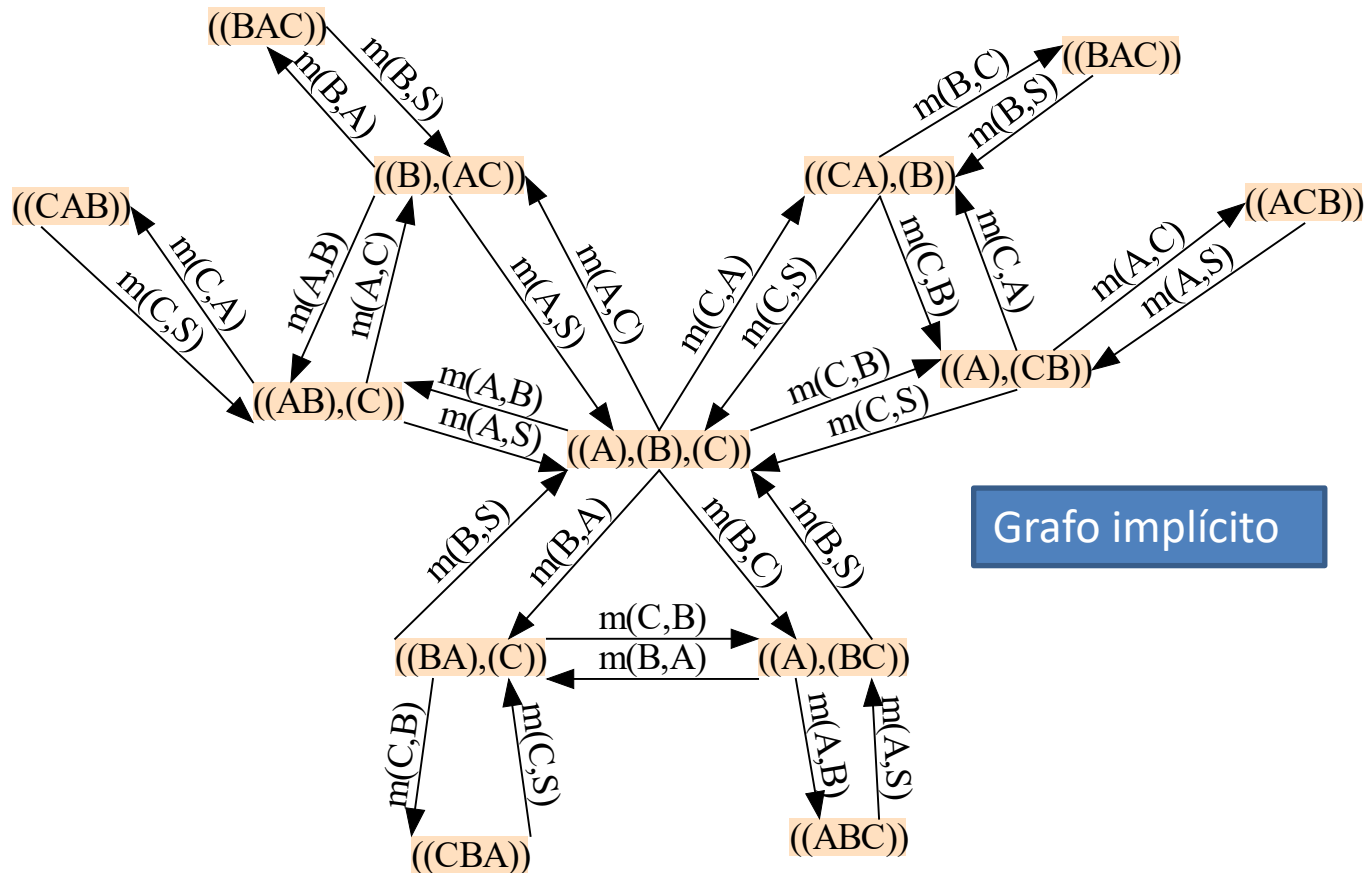
- Una estructura de **grafo dirigido** puede ser útil para buscar secuencias de acciones que nos lleven al objetivo final.
- En esta estructura, **un nodo representa un estado** del sistema y **un arco una posible acción**. La acción, aplicada al estado que representa al nodo origen, producirá el estado del nodo destino.
- Se denomina **grafo de estados**.



El mundo de bloques

- A la secuencia de acciones que lleva al agente desde un **estado inicial** hasta un **estado destino** se denomina **plan**.
- La búsqueda de dicha secuencia se denomina **planificación**.
 - Grafos explícitos: grafo que representa el espacio de estados explorado hasta el momento y que se almacena físicamente en memoria.
 - Grafos implícitos: grafo que representa el espacio de estados en su totalidad, no está almacenado físicamente en memoria.

Espacio de estados en el mundo de bloques

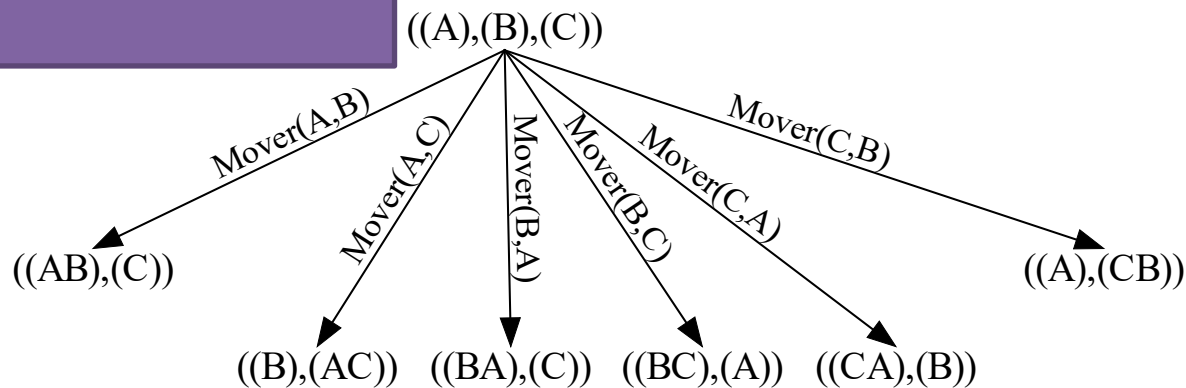


Búsqueda

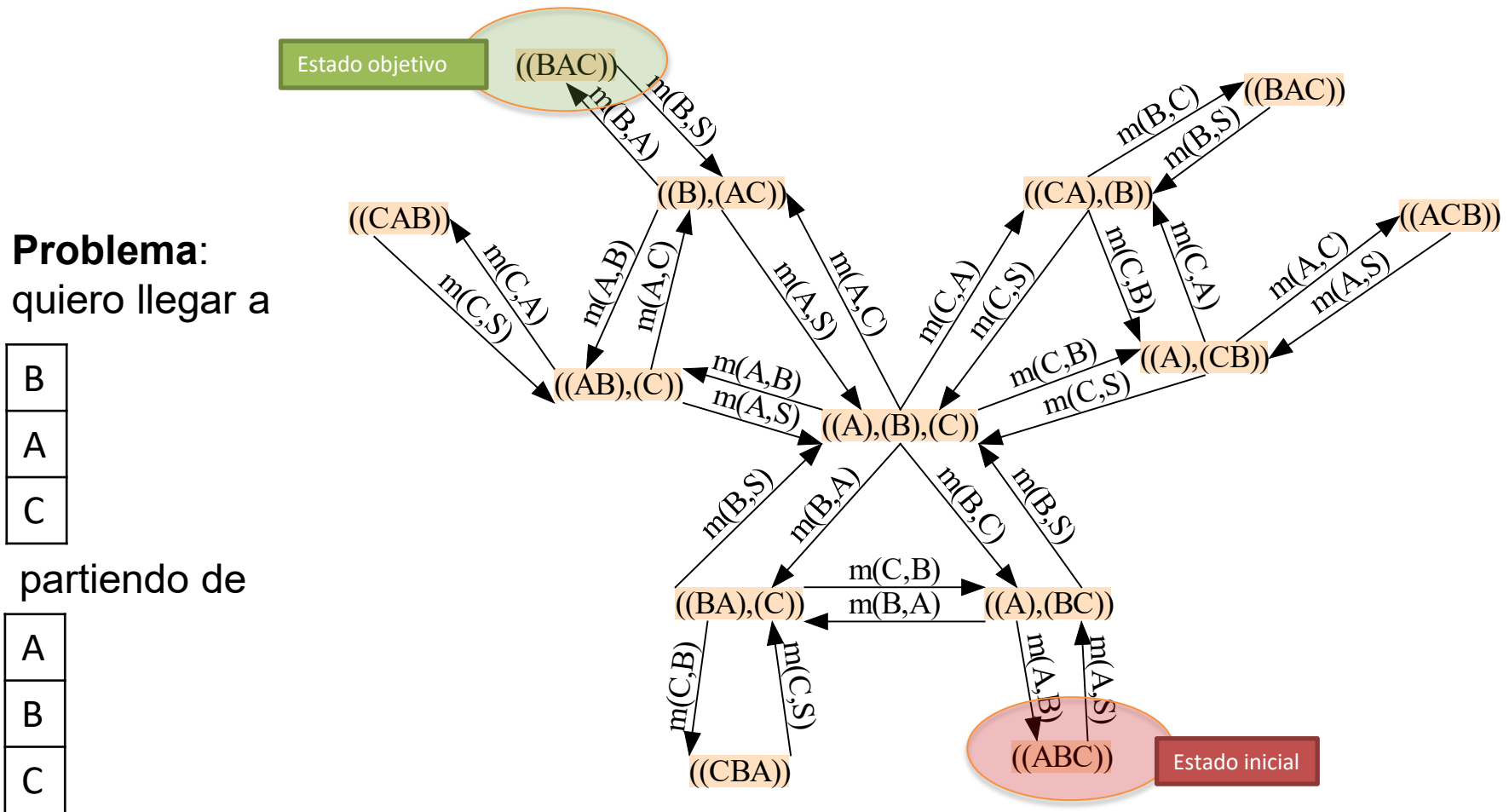
Razonamiento regresivo:

El grafo explícito se va construyendo por etapas conforme vamos “descubriendo” sus nodos aplicando acciones. En el primer ejemplo hemos visto cómo el grafo se va construyendo **progresivamente** a partir del estado inicial. Hay problemas en los que puede ser conveniente partir del objetivo y aplicar las acciones en sentido inverso, haciendo un proceso de exploración del grafo **regresivo**.

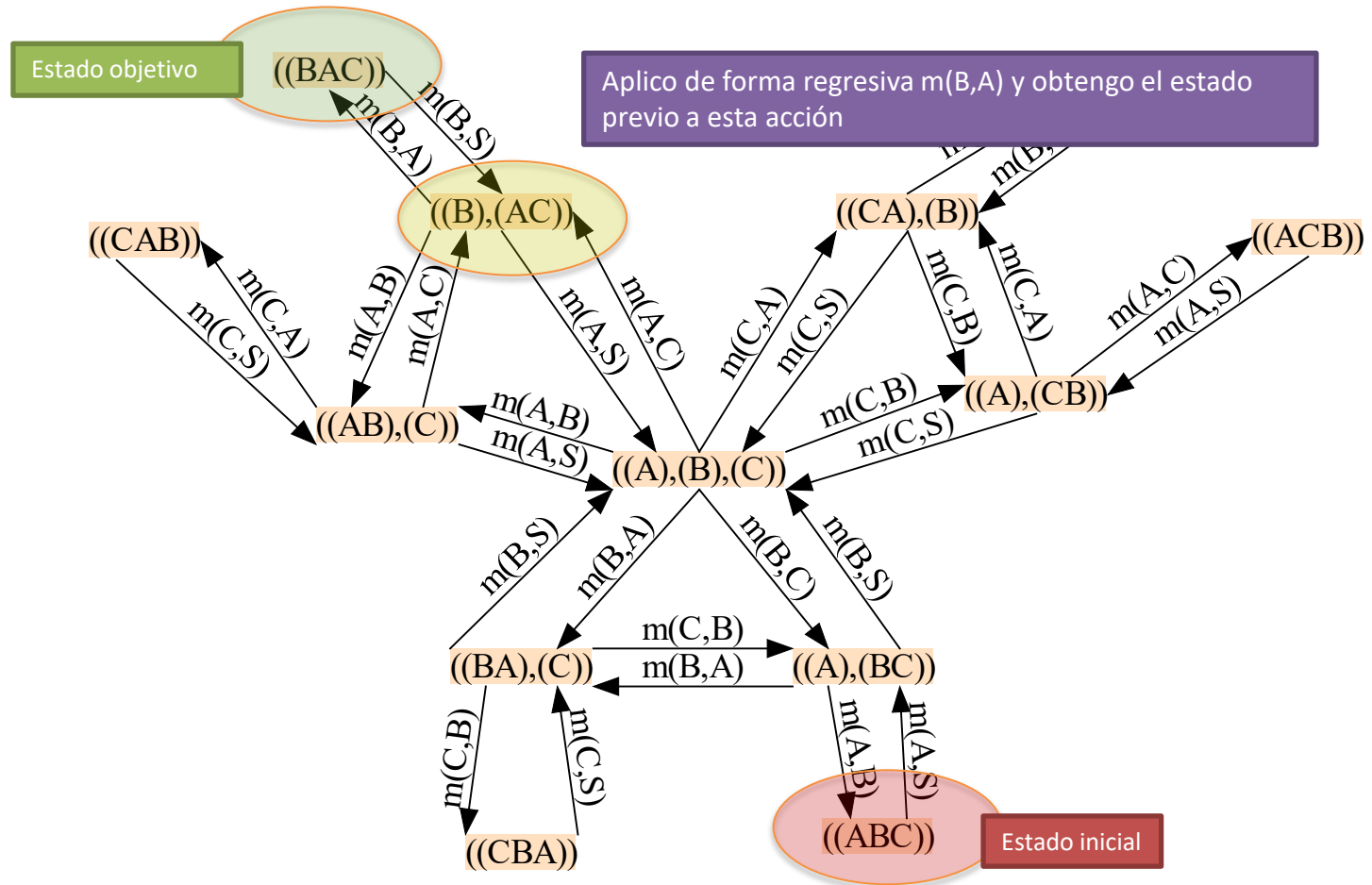
Ejemplo de aplicación regresiva de una acción:
para obtener el estado $((A,B),C)$ debería de aplicar $\text{Mover}(A,B)$



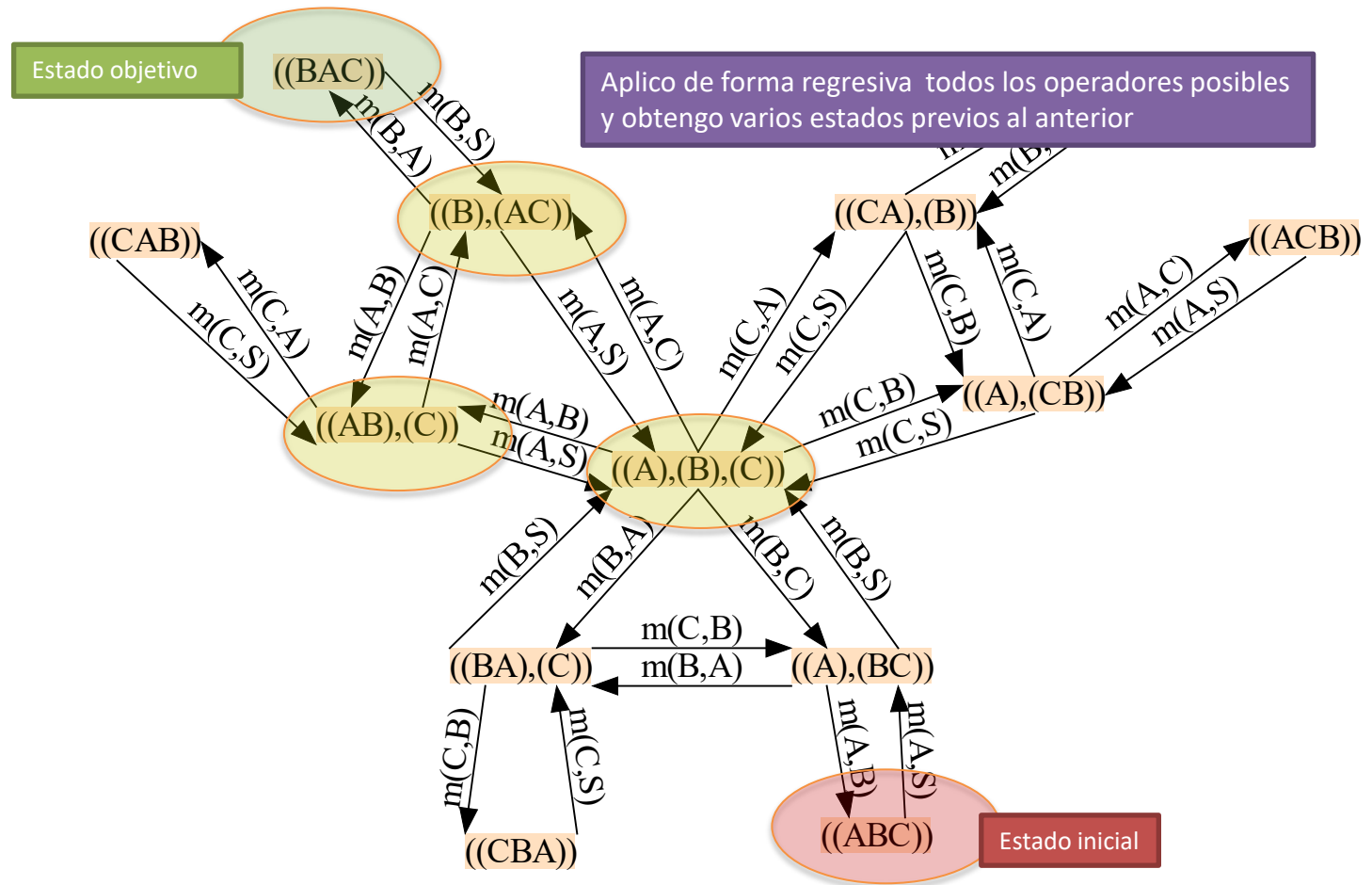
Espacio de estados en el mundo de bloques



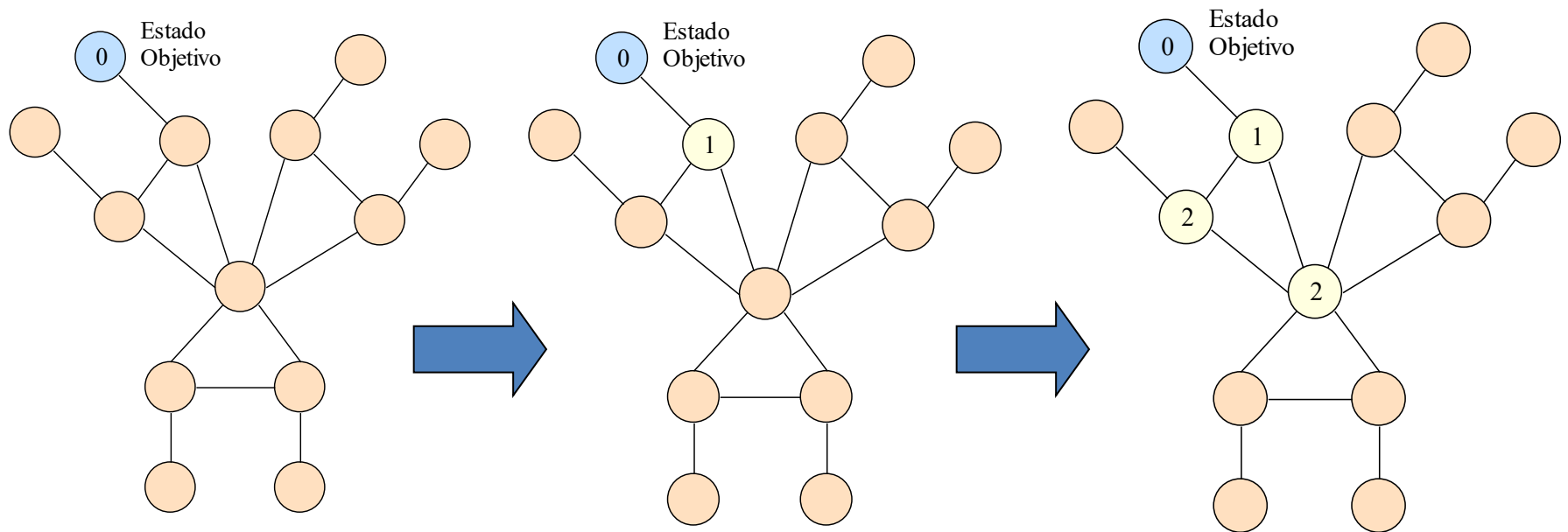
Espacio de estados en el mundo de bloques



Espacio de estados en el mundo de bloques

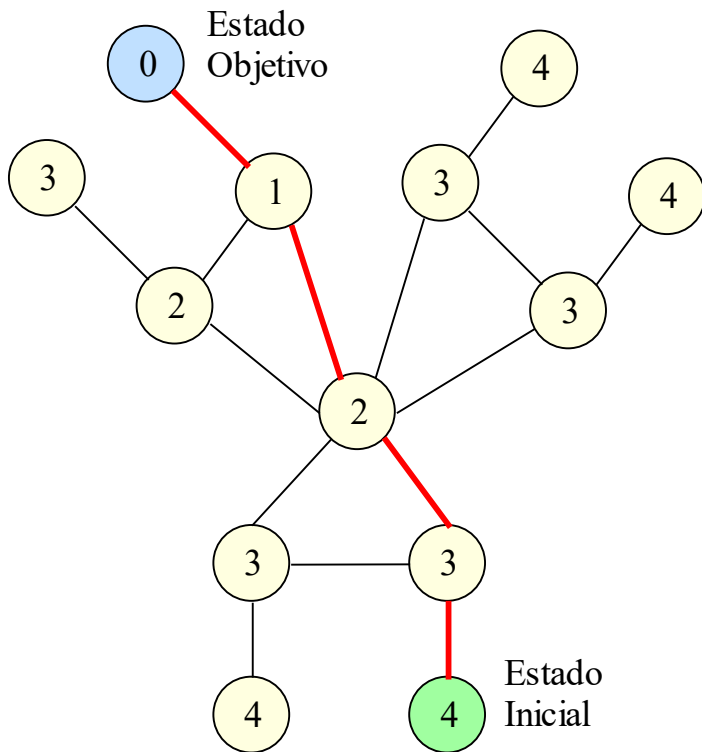


Búsqueda



Búsqueda

- **Ejemplo de planificación en el mundo de los bloques:**
Planificación de acciones.



Estado Inicial: **((ABC))**

Acción 1: **Mover(A, Suelo)**

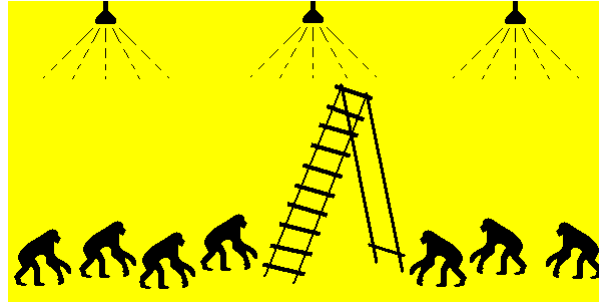
Acción 2: **Mover(B, Suelo)**

Acción 3: **Mover(A, C)**

Acción 4: **Mover(B, A)**

Estado objetivo alcanzado: **((BAC))**

Ejemplo de agente deliberativo: Problema del mono y los plátanos

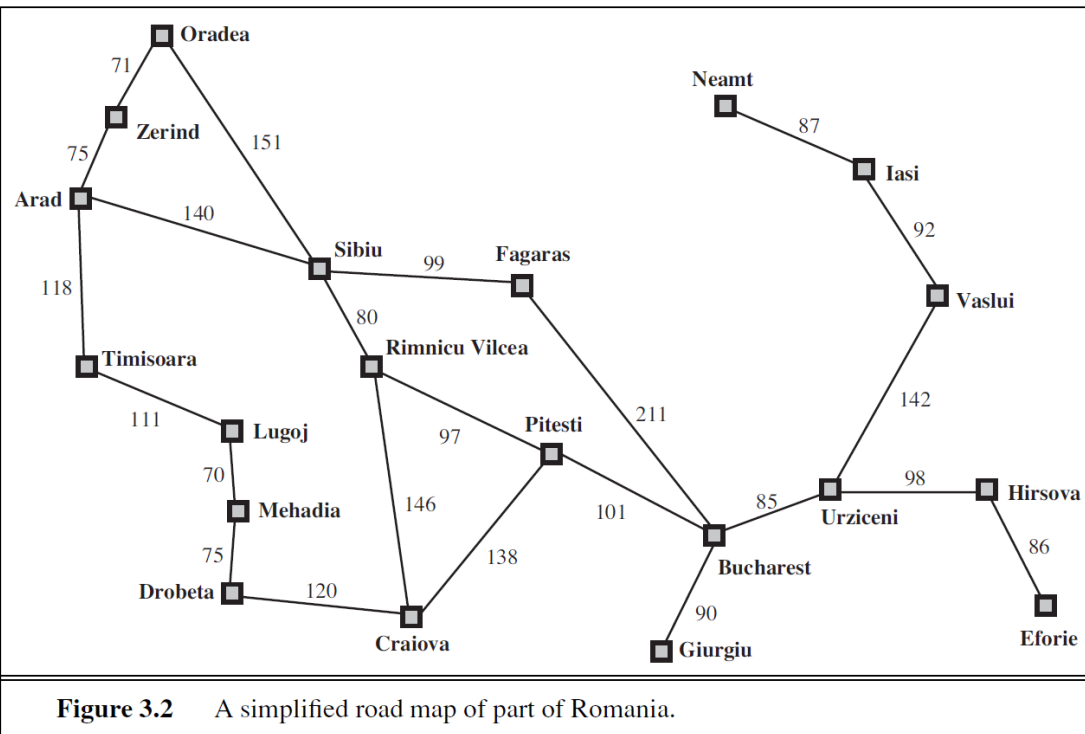


- “Un mono está en la puerta de una habitación. En el centro de la habitación hay un plátano colgado del techo, pero no puede alcanzarlo desde el suelo. En la ventana de la habitación hay una caja, que el mono puede mover y a la que puede encaramarse para alcanzar el plátano. El mono puede realizar las siguientes acciones: desplazarse de la puerta al centro, del centro a la ventana y viceversa; empujar la caja a la vez que se desplaza; subirse y bajarse de la caja; coger el plátano. El problema consiste en encontrar una secuencia de acciones que permita al mono coger el plátano.”

Problema del mono y los plátanos

- Como estado se puede utilizar una lista con cuatro elementos (X, Y, W, Z) donde:
 - X: posición del mono en la habitación (puerta, centro, ventana).
 - Y: situación del mono respecto a la caja (suelo, caja).
 - W: posición de la caja en la habitación (puerta, centro, ventana).
 - Z: posesión del plátano (tiene, no_tiene).
- Para describir todas las posibles acciones del mono, se necesitan seis operadores: andar, empujar, subir, bajar, coger y soltar. Sin embargo, para el problema que nos ocupa, son suficientes cuatro operadores: andar, empujar, subir y coger; nos limitaremos a estos 4 operadores .
- El operador andar se puede definir como:
 - (X, suelo, W, Z) -----> (Y, suelo, W, Z)
- para indicar que el mono se desplaza de la posición X a la posición Y.
- El estado se puede representar por la estructura *estado(X, Y, W, Z)*, que simplemente refleja la definición del estado.

Ejemplo de agente deliberativo: Mapa de Carreteras



Ejemplo de agente deliberativo: Mapa de Carreteras

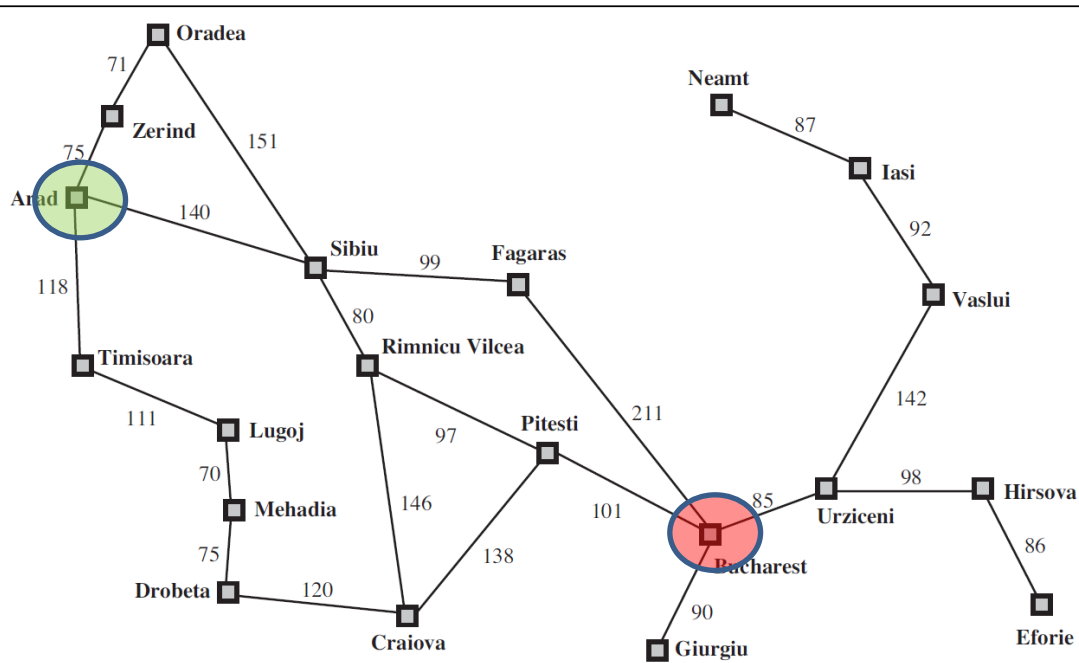


Figure 3.2 A simplified road map of part of Romania.

Estado inicial:

En(Arad)

Estado objetivo:

En(Bucharest)

Acciones aplicables: (en el estado *en(Arad)*)

{ir_a(Sibiu), ir_a(Timisoara), ir_a(Zerind)}

Modelo de transición:

RESULT(en(Arad), ir_a(Zerind)=en(Zerind))

Coste:

Cada arco (s,a,s') tiene representado un coste $c(s,a,s')$

Coste del camino: la suma de los costes individuales de cada arco

Ejemplo de agente deliberativo: Mapa de Carreteras

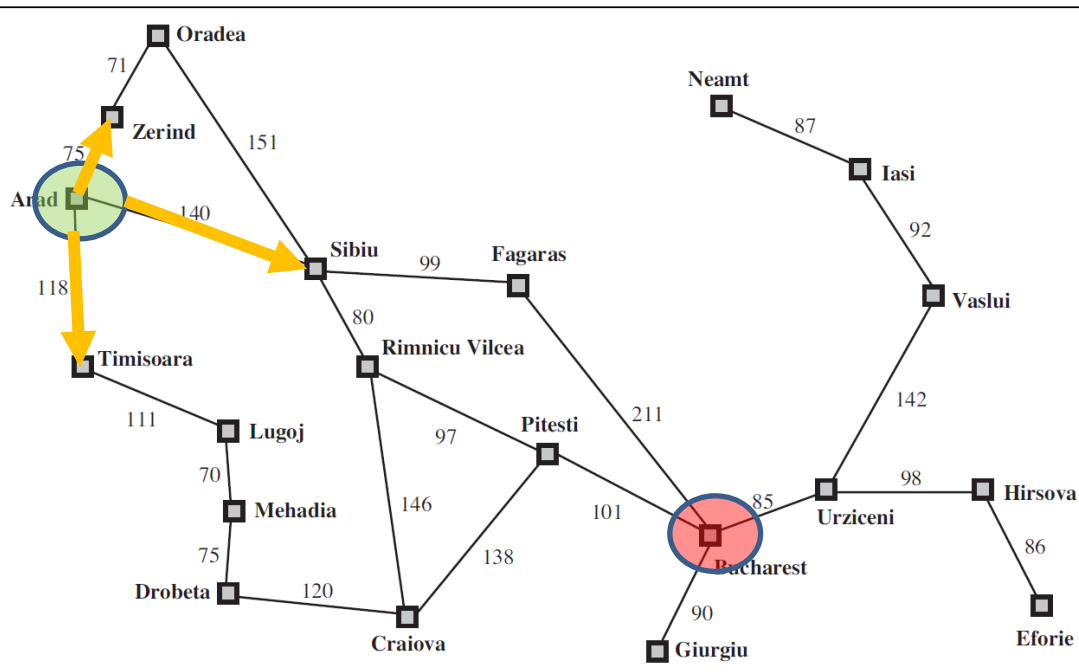


Figure 3.2 A simplified road map of part of Romania.

Estado inicial:

En(Arad)

Estado objetivo:

En(Bucharest)

Acciones aplicables: (en el estado *en(Arad)*)

{ir_a(Sibiu), ir_a(Timisoara), ir_a(Zerind)}

Modelo de transición:

RESULT(en(Arad), ir_a(Zerind)=en(Zerind))

Coste:

Cada arco (s,a,s') tiene representado un coste $c(s,a,s')$

Coste del camino: la suma de los costes individuales de cada arco

Ejemplo de agente deliberativo: Mapa de Carreteras

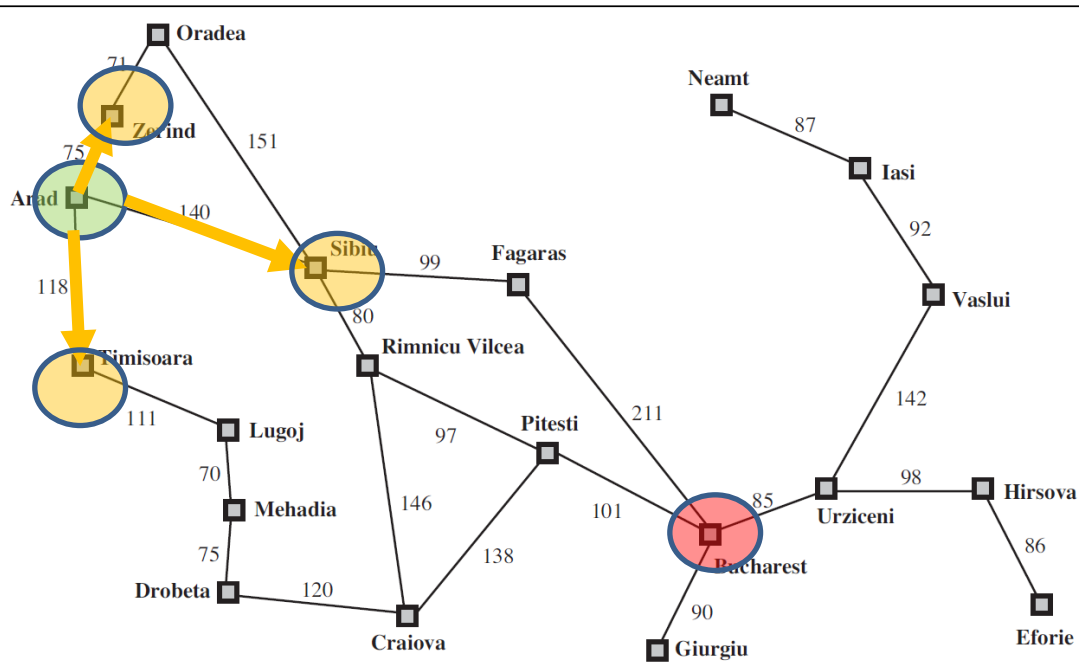


Figure 3.2 A simplified road map of part of Romania.

Estado inicial:

En(Arad)

Estado objetivo:

En(Bucharest)

Acciones aplicables: (en el estado *en(Arad)*)

{ir_a(Sibiu), ir_a(Timisoara), ir_a(Zerind)}

Modelo de transición:

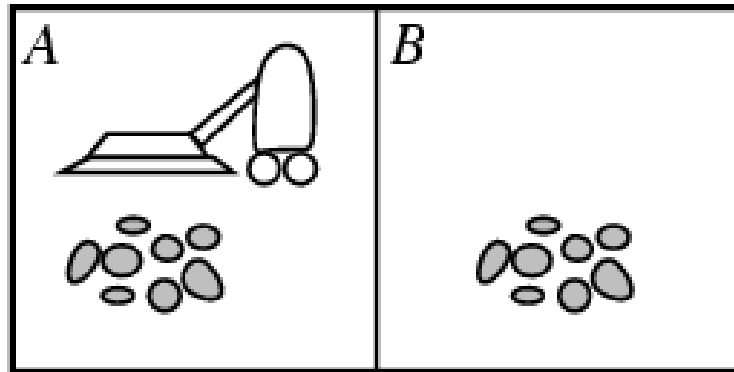
RESULT(en(Arad),ir_a(Zerind)=en(Zerind))

Coste:

Cada arco (s,a,s') tiene representado un coste $c(s,a,s')$

Coste del camino: la suma de los costes individuales de cada arco

Ejemplo de un agente deliberativo: Problema de la aspiradora



La aspiradora puede estar en dos ubicaciones y puede haber 2^2 configuraciones de basura.

Tamaño del espacio de estados = $2 \times 2^2 = 8$

Problema de la aspiradora

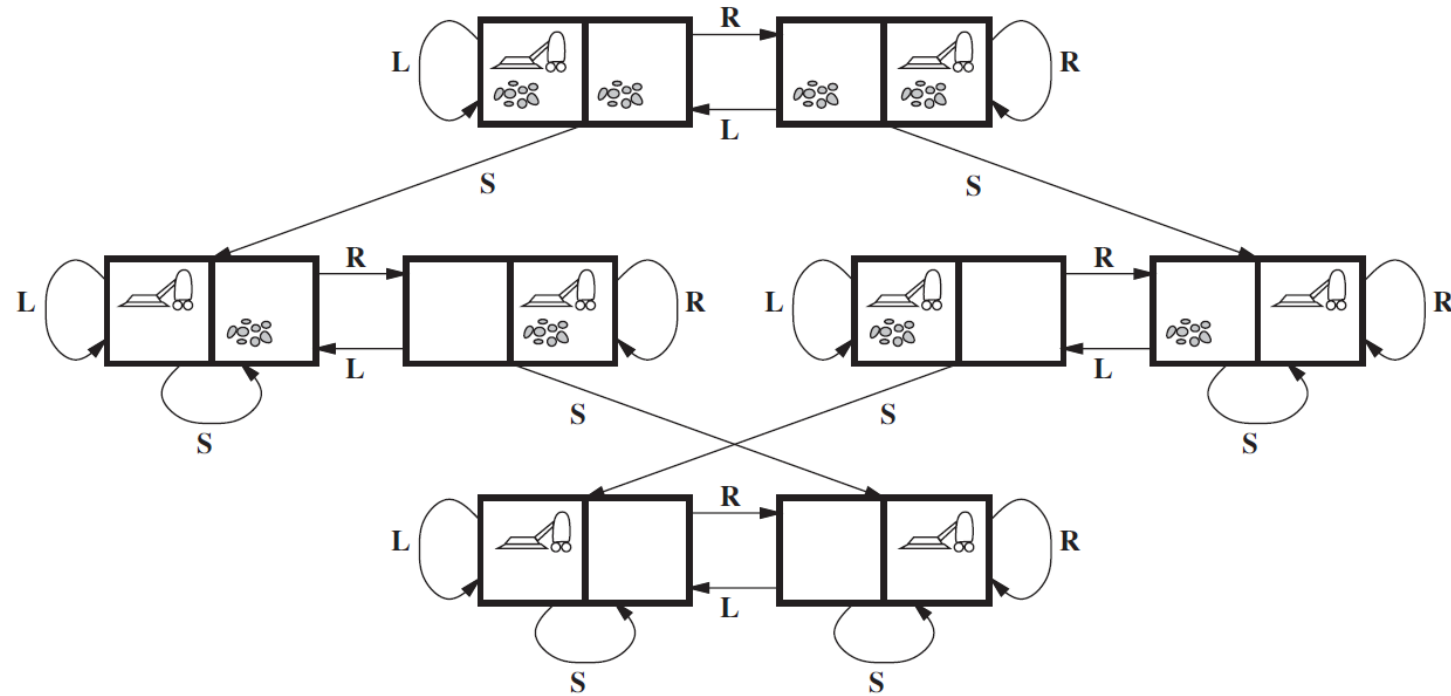


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Si tenemos n ubicaciones **Tamaño** = $n \cdot 2^n$

El 8-puzzle

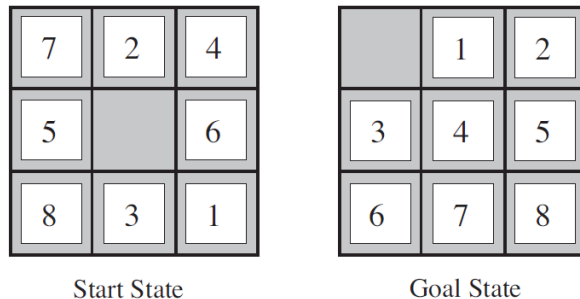


Figure 3.4 A typical instance of the 8-puzzle.

Estado: una matriz indicado la posición de cada casilla y de la blanca.

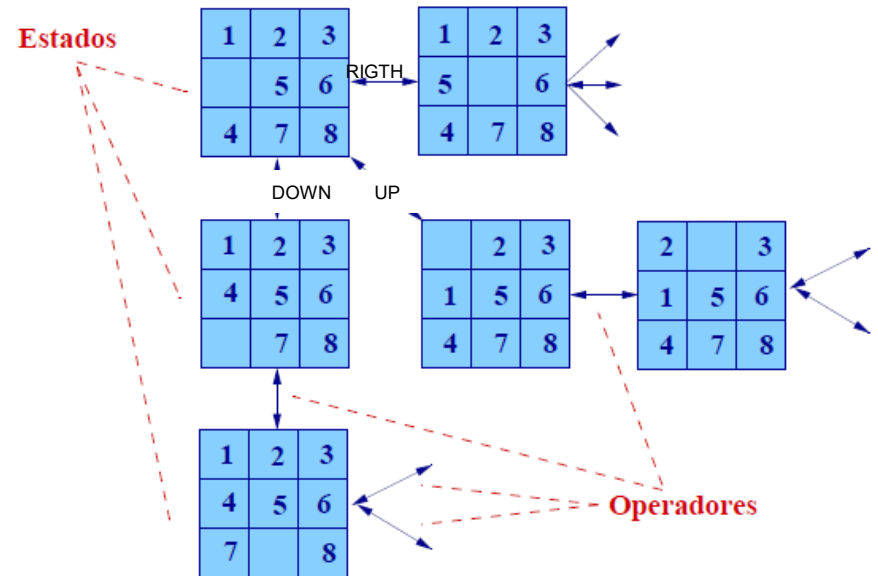
Estado inicial: Una matriz con valores específicos para el puzzle de la izda de la figura (Start State)

Estado objetivo: Una matriz con valores específicos para el puzzle de la dcha de la figura (Goal State)

Acciones aplicables: movimientos posibles de la casilla blanca.
Left, Right, Up, Down

Modelo de transición: dados un estado y una acción devuelve un estado representando una configuración de puzzle

Coste: Cada movimiento cuesta 1, luego el costo del camino es el número de movimientos o pasos para resolver un 8-puzzle



Tamaño espacio de estados 8-puzzle

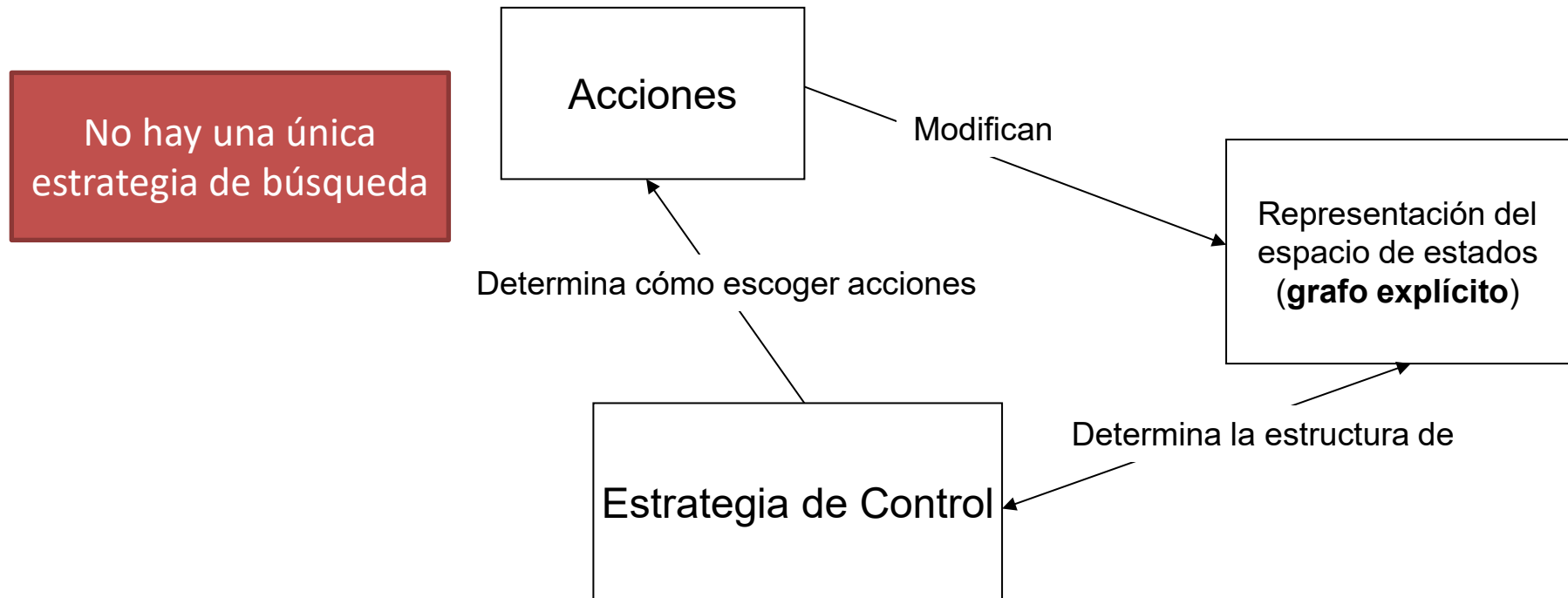
$9!/2 = 181440$ estados alcanzables.

Tamaño del 15-puzzle $\approx 1,3 \cdot 10^{12}$

Tamaño del 24-puzzle $\approx 10^{25}$

Sistemas de búsqueda y estrategias

Una vez que formulamos un problema, tenemos que **resolverlo**, con un **proceso de búsqueda** que tiene que encontrar una **secuencia de acciones** partiendo del **estado inicial** como **nodo raíz** del grafo explícito o del **grafo/árbol de búsqueda**, los arcos son acciones y los nodos son estados del espacio de estados del problema.



Procedimiento general de búsqueda

1. ***Grafo explícito*** ← Inicializar con estado inicial
2. **until** ***Grafo explícito*** contiene un nodo que cumple la condición de terminación (test de objetivo)
do
3. **begin**
4. **select** alguna acción A en el conjunto de acciones que pueda ser aplicada a ***Grafo explícito***
5. ***Grafo explícito*** ← resultado de aplicar A al grafo
6. **end**

Estrategias de control

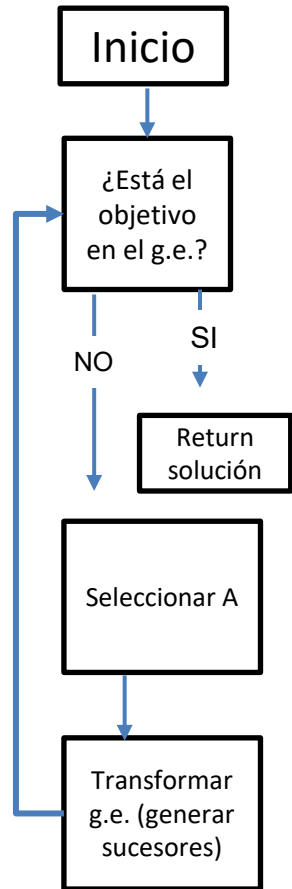
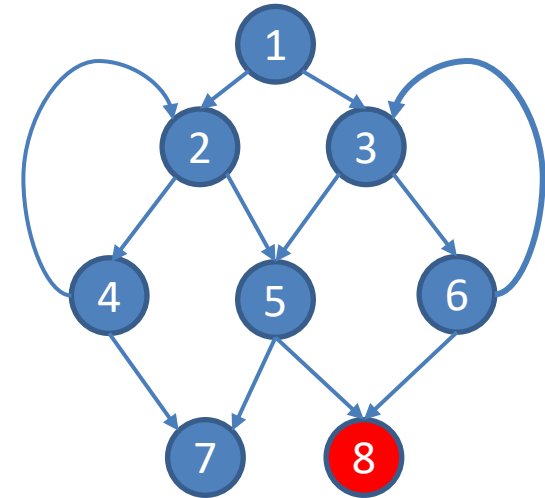
- Dependiendo de cómo representemos el grafo explícito podemos usar distintas estrategias para explorarlo
 - Estrategias irrevocables
 - Estrategias tentativas
 - Retroactivas
 - Búsqueda en grafos

Estrategias irrevocables

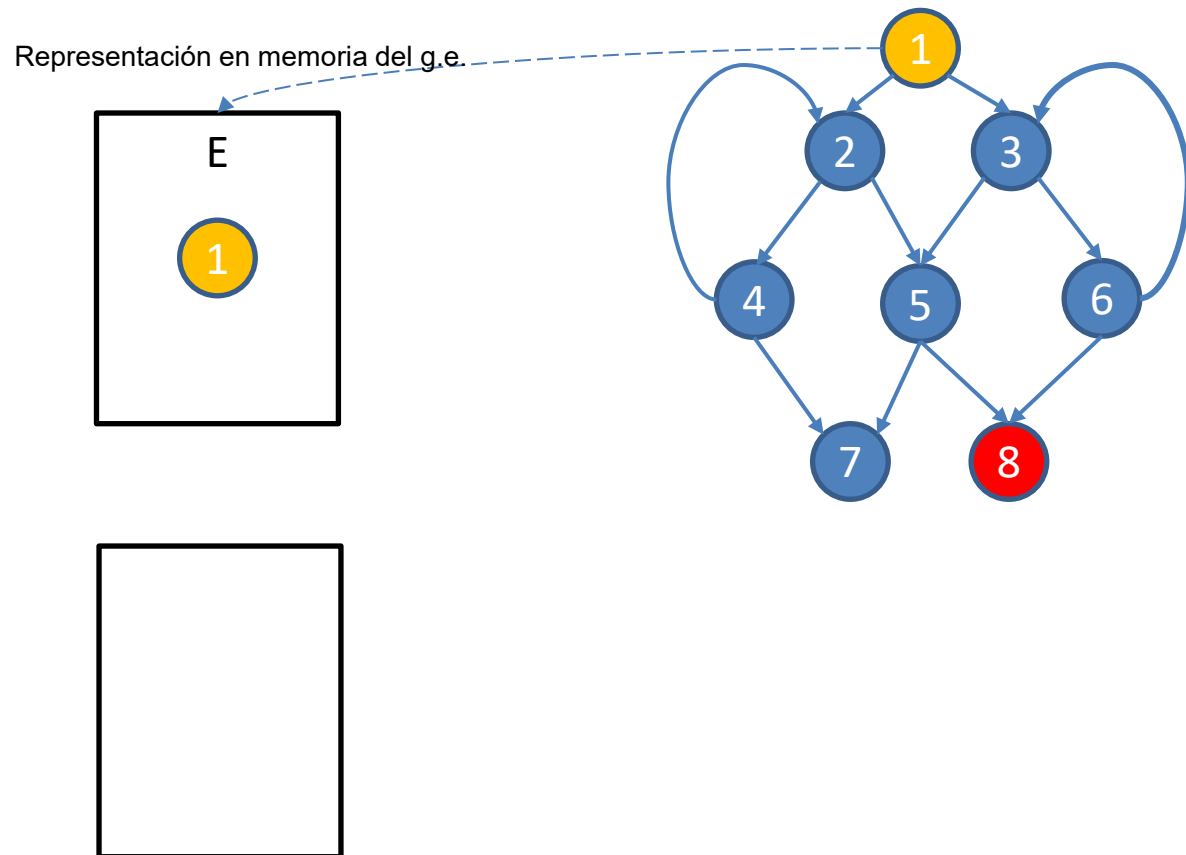
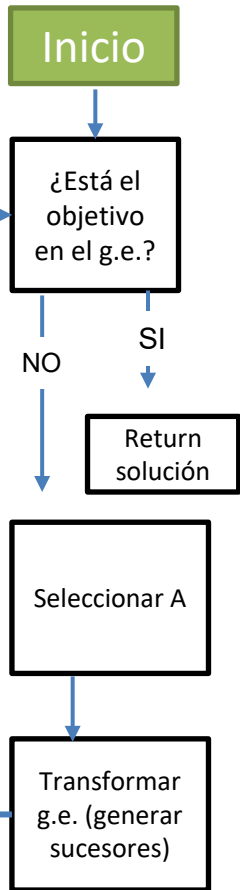
- En cada momento, el **grafo explícito** lo constituye **un único nodo**, E , que incluye la descripción completa del sistema en ese momento:
 - Se selecciona una acción A .
 - Se aplica sobre el estado del sistema E , para obtener el nuevo estado $E' = A(E)$.
 - Se borra de memoria E y se sustituye por E' .
- El proceso se detiene cuando E cumple la condición de estado objetivo.

Estrategias irrevocables

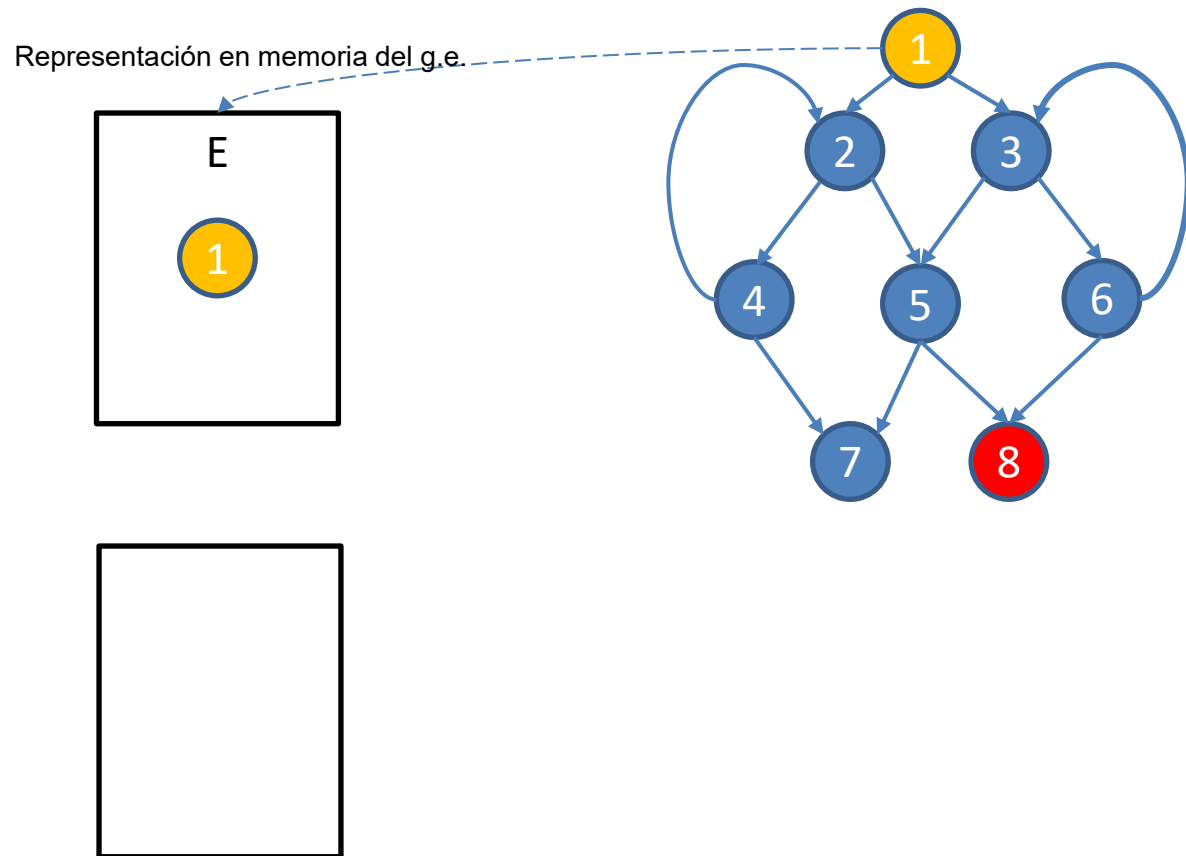
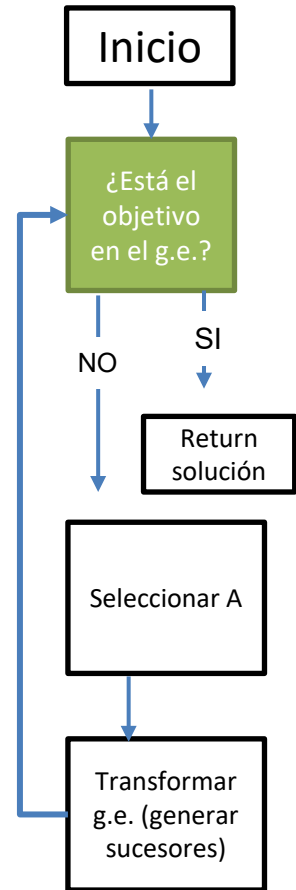
Representación en memoria del g.e.



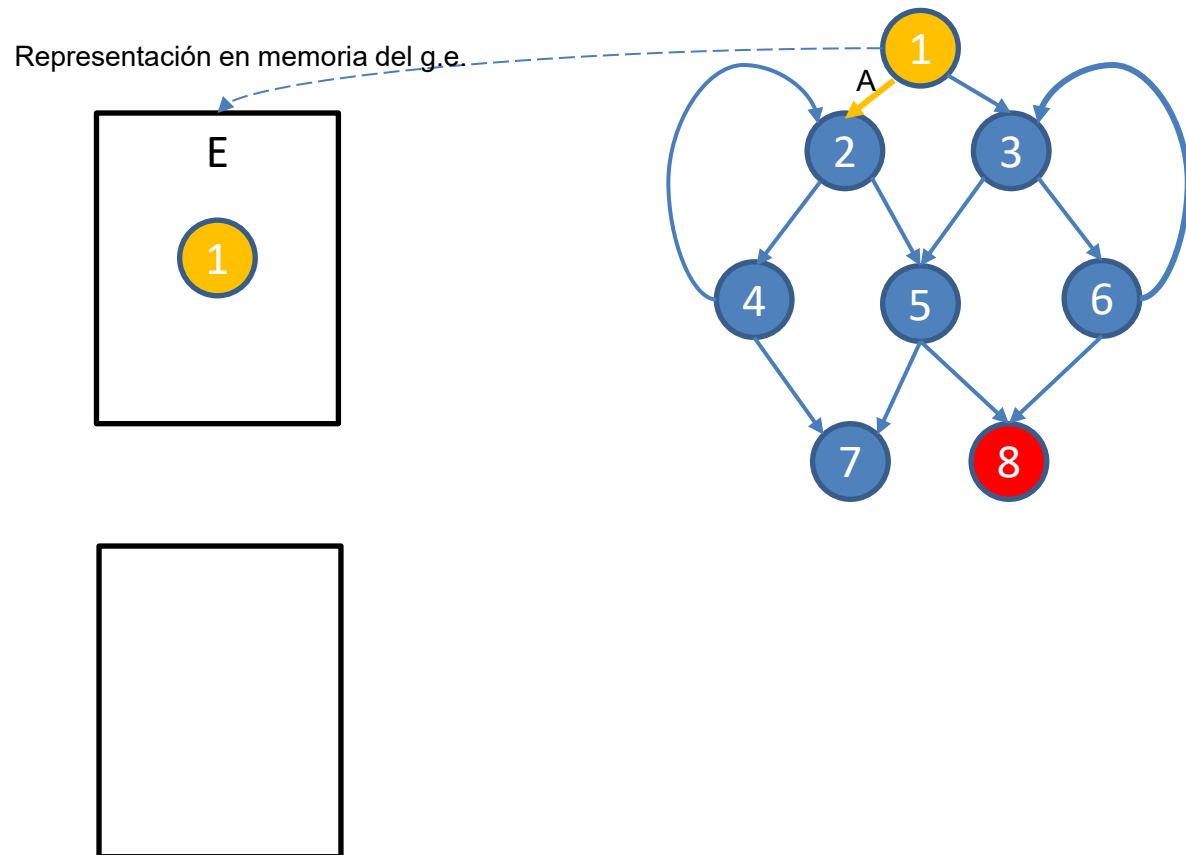
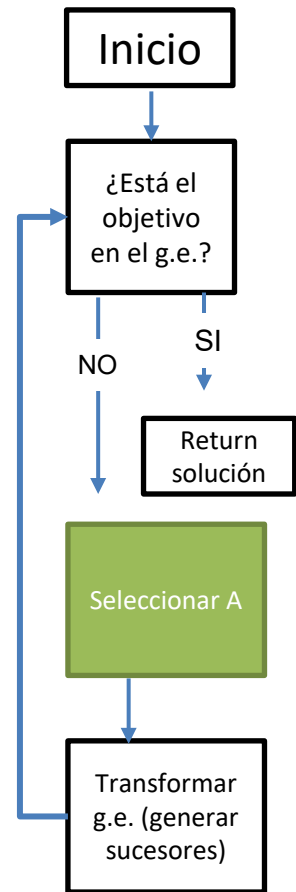
Estrategias irrevocables



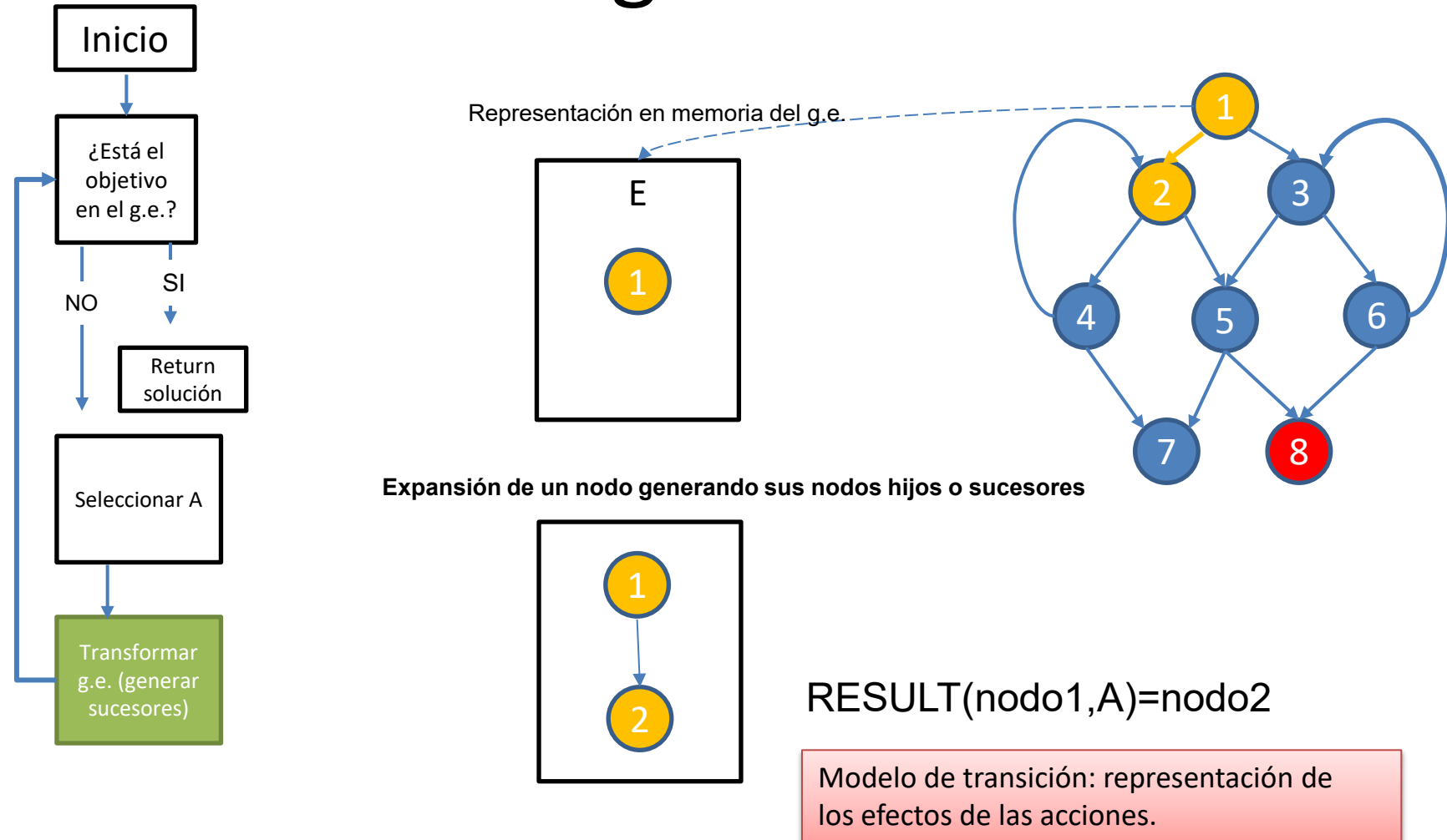
Estrategias irrevocables



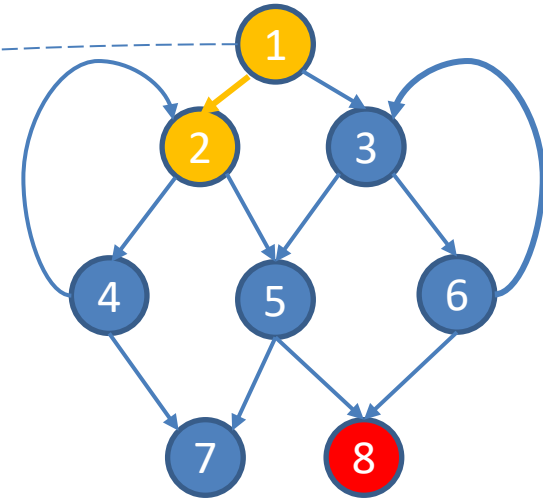
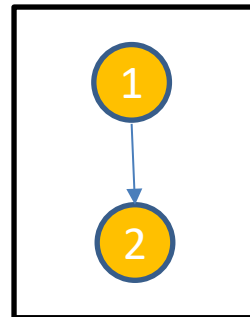
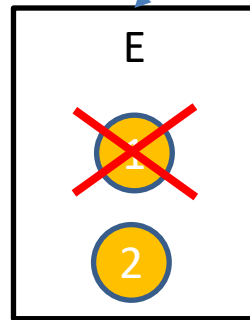
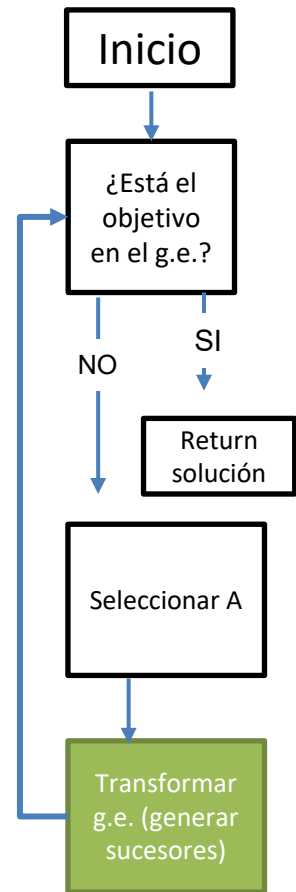
Estrategias irrevocables



Estrategias irrevocables

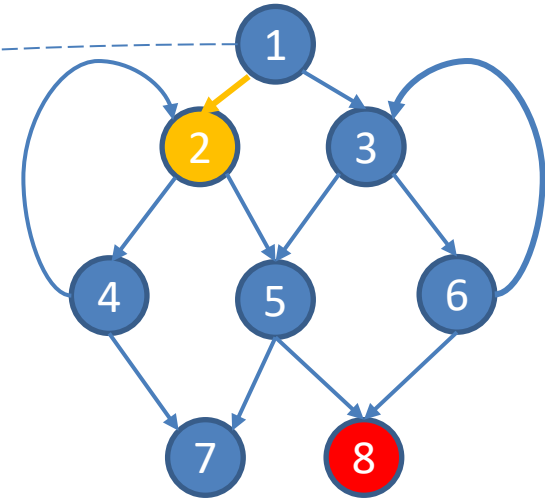
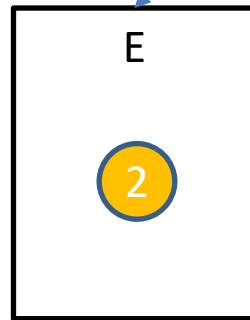
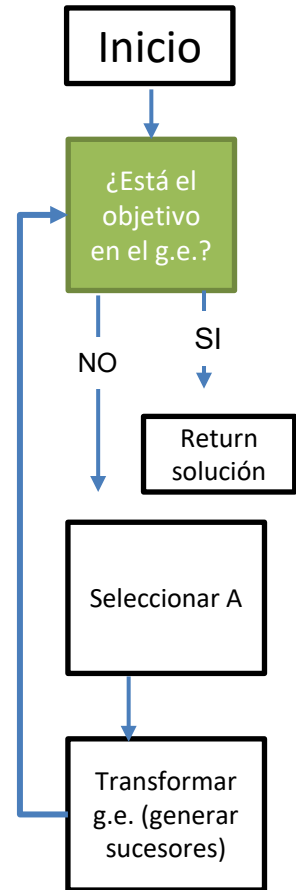


Estrategias irrevocables



RESULT(nodo1,A)=nodo2
 $E' = \text{nodo2}$
Hacer $E = E'$

Estrategias irrevocables

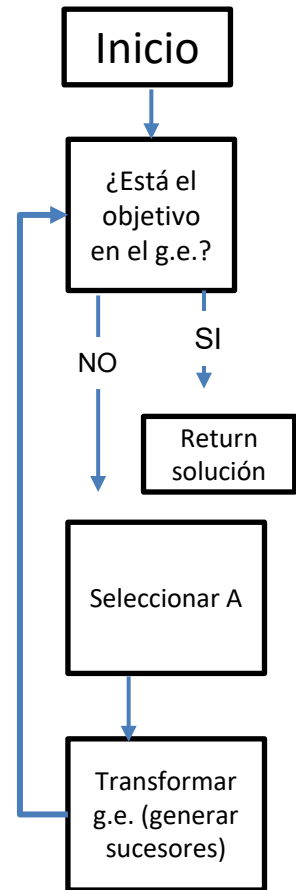


El nuevo nodo es lo único que se conoce del grafo explícito

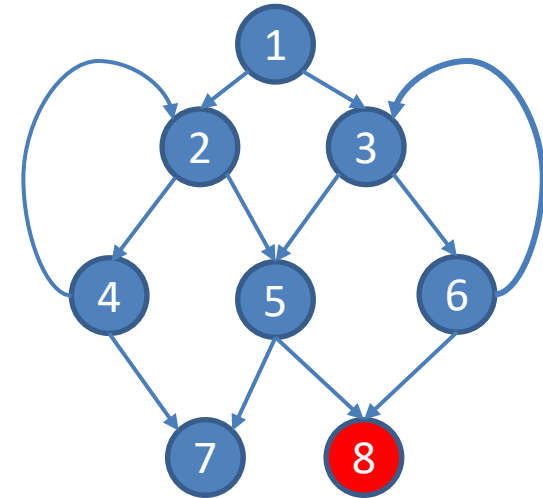
Estrategia retroactiva (Backtracking)

- En memoria sólo guardamos un hijo de cada estado; esto es, se mantiene el camino desde el estado inicial hasta el actual.
- El **grafo explícito**, por tanto, es realmente **una lista**.
- ¿Cuándo para el proceso? Cuando hemos llegado al objetivo y no deseamos encontrar más soluciones, o bien no hay más operadores aplicables al nodo raíz del espacio de búsqueda.

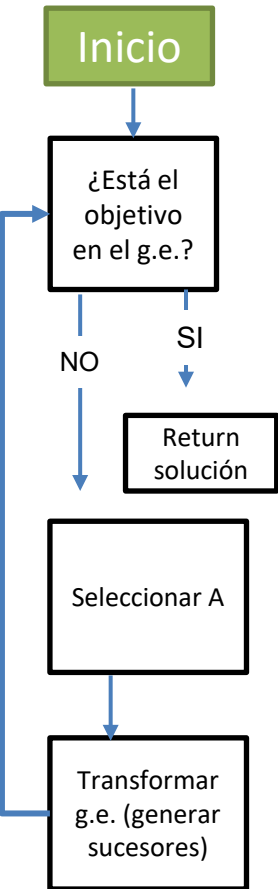
Estrategia retroactiva



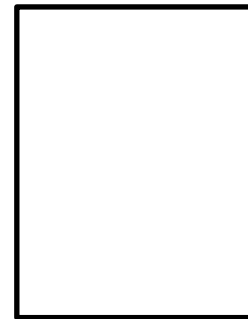
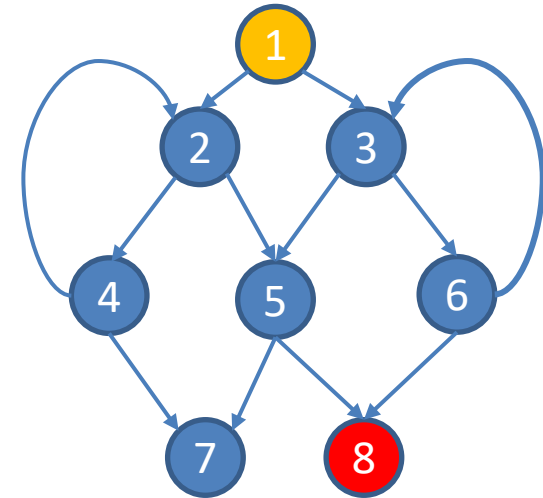
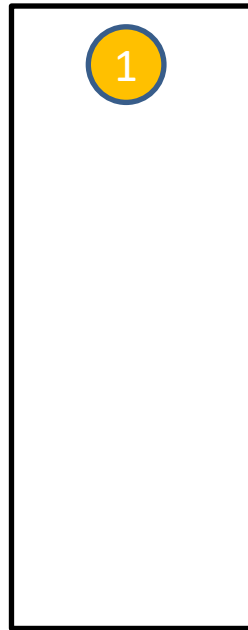
Representación en memoria del g.e.



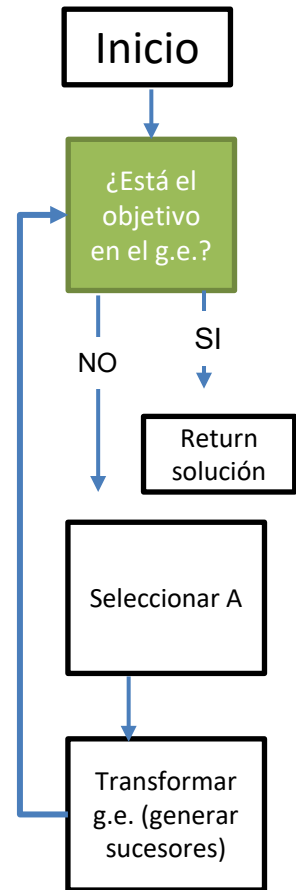
Estrategia retroactiva



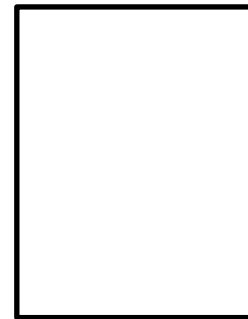
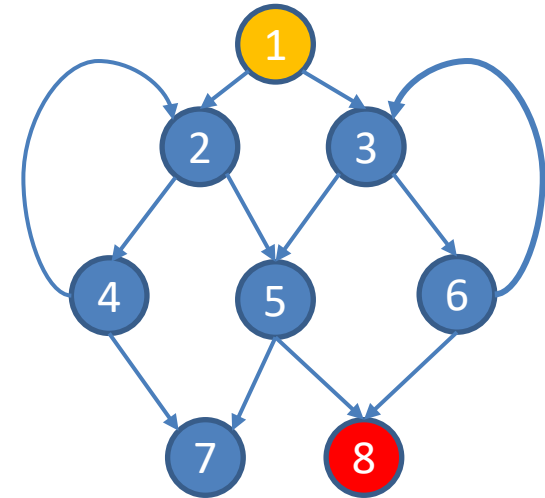
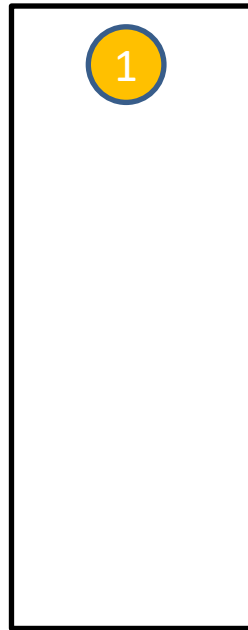
Representación en memoria del g.e.



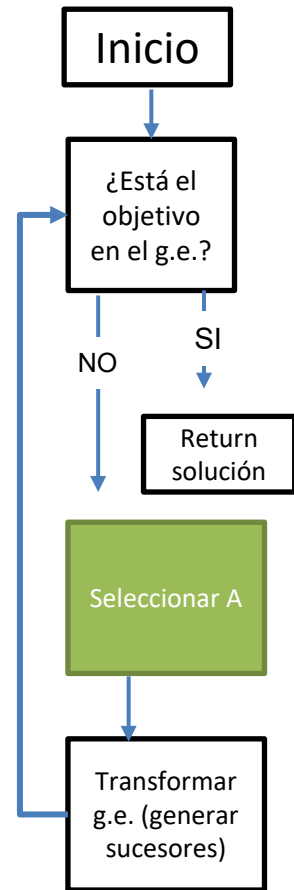
Estrategia retroactiva



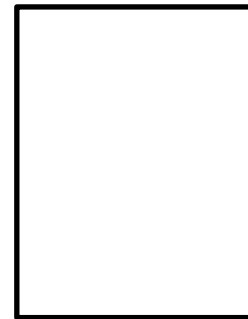
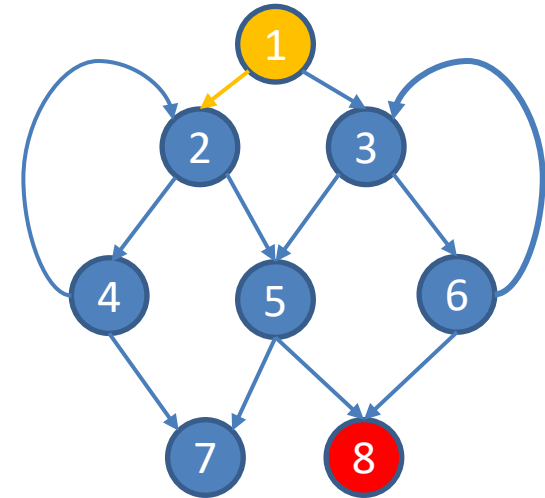
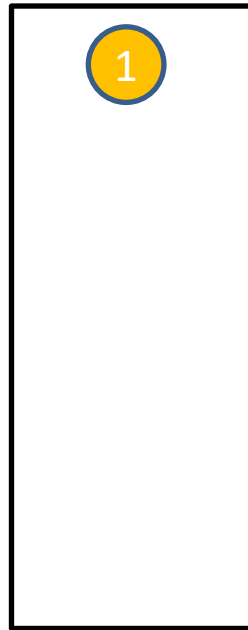
Representación en memoria del g.e.



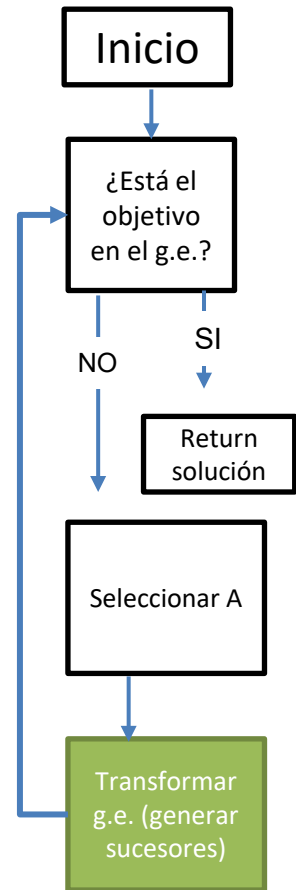
Estrategia retroactiva



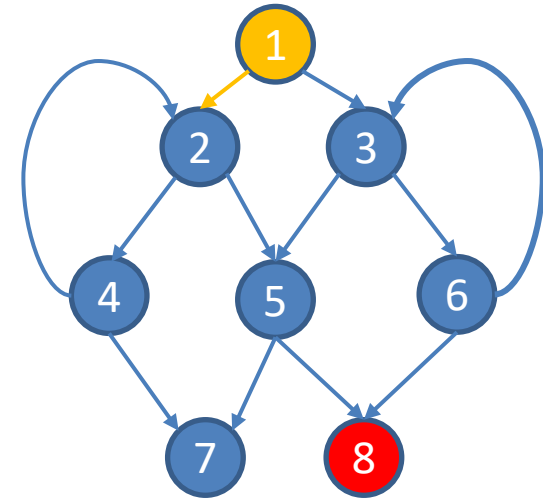
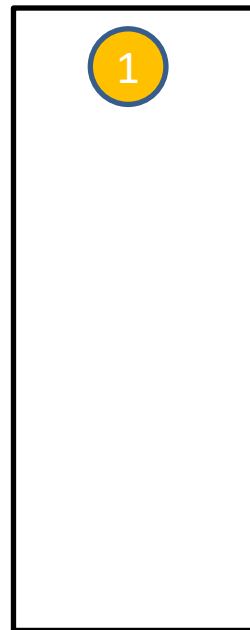
Representación en memoria del g.e.



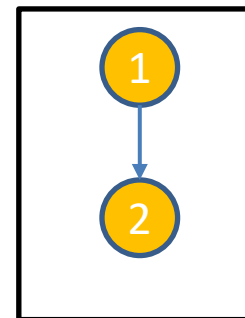
Estrategia retroactiva



Representación en memoria del g.e.



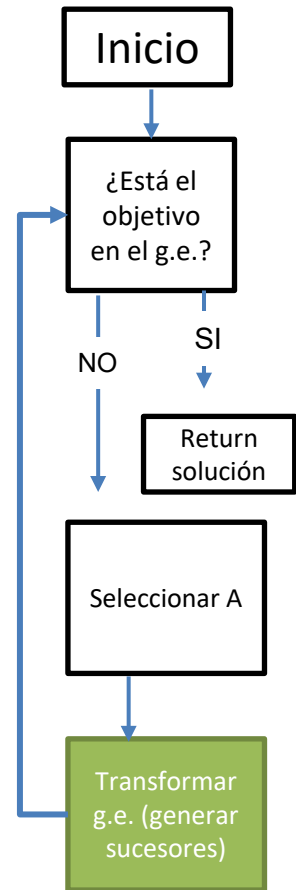
Expansión de un nodo generando sus nodos hijos o sucesores



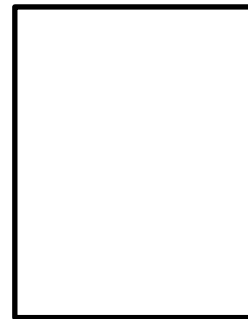
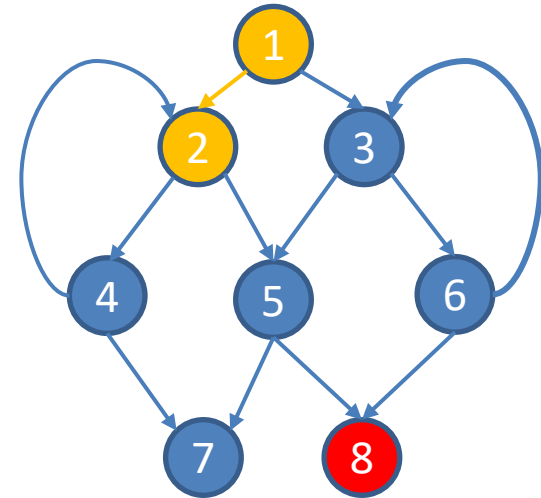
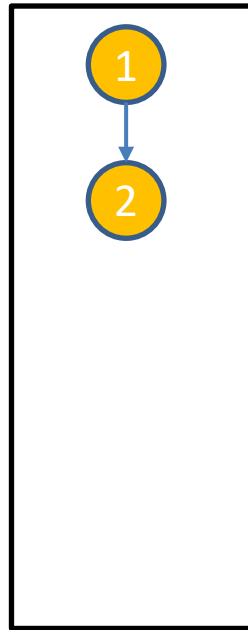
$\text{RESULT}(\text{nodo1}, A) = \text{nodo2}$

Modelo de transición: representación de los efectos de las acciones.

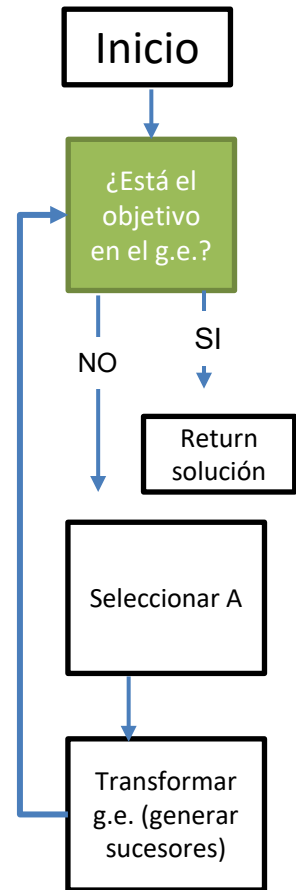
Estrategia retroactiva



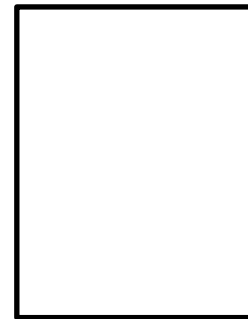
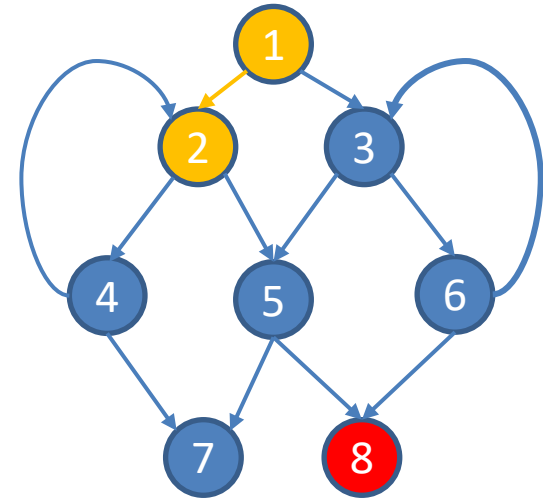
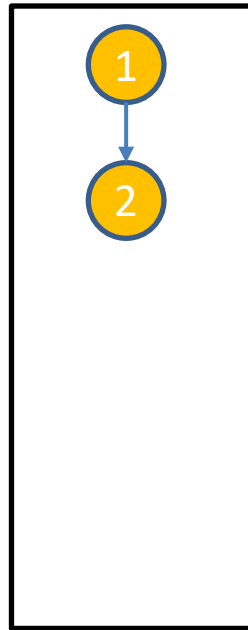
Representación en memoria del g.e.



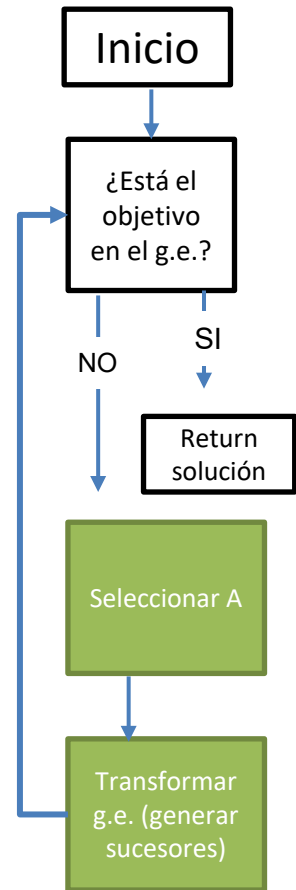
Estrategia retroactiva



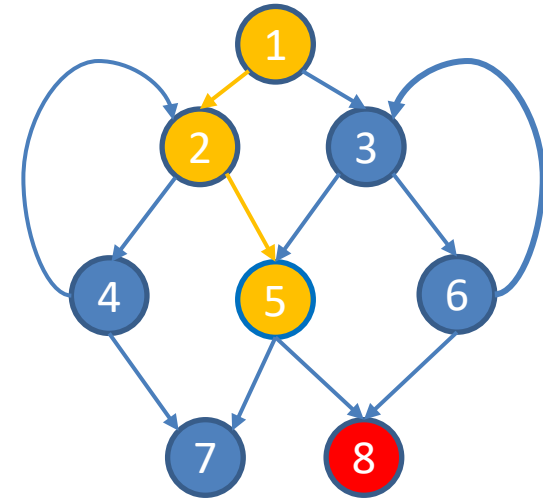
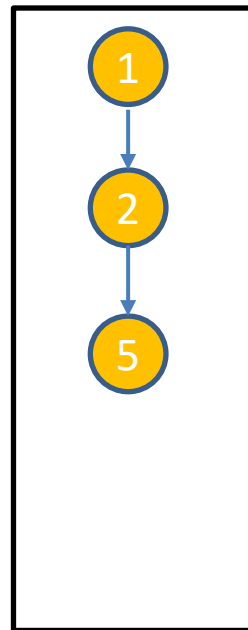
Representación en memoria del g.e.



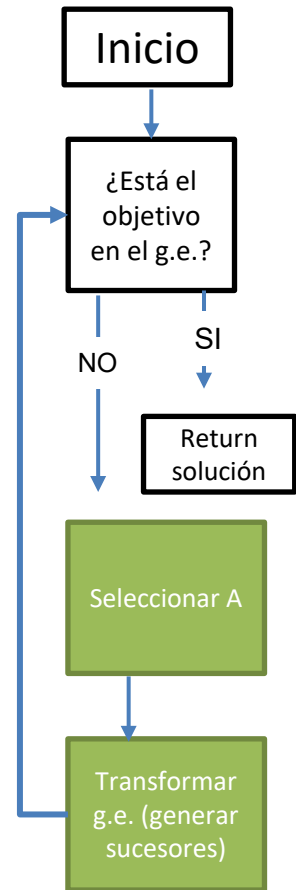
Estrategia retroactiva



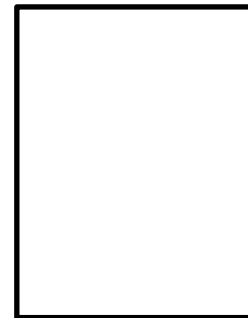
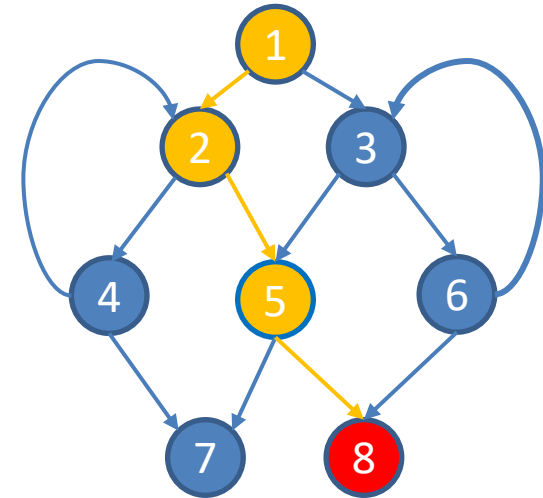
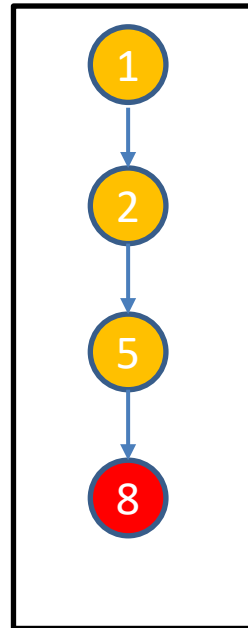
Representación en memoria del g.e.



Estrategia retroactiva



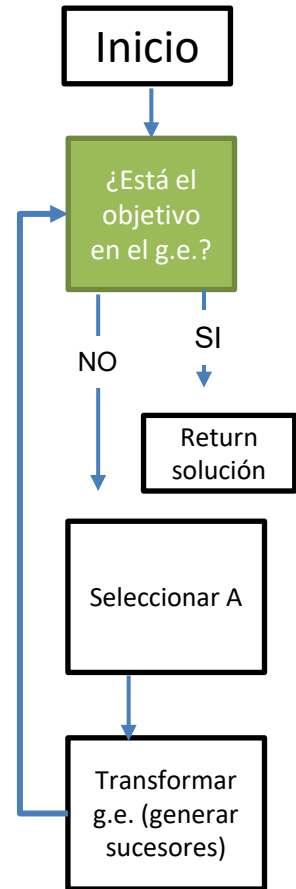
Representación en memoria del g.e.



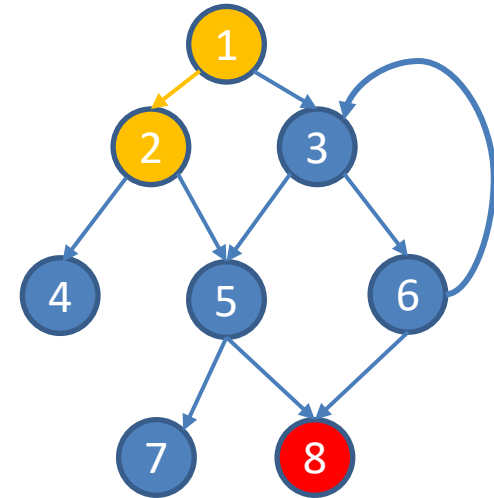
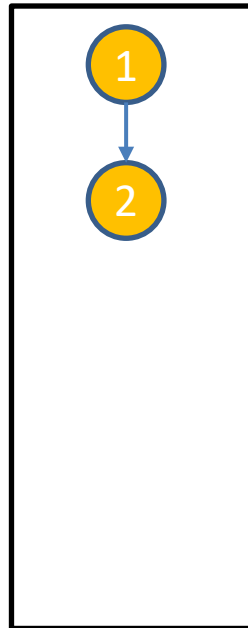
Estrategia retroactiva (Backtracking)

- ¿Cuándo se produce una vuelta atrás (o retroceso)?
 - Cuando se ha encontrado una solución, pero deseamos encontrar otra solución alternativa.
 - Cuando se ha llegado a un límite en el nivel de profundidad explorado o el tiempo de exploración en una misma rama.
 - Cuando se ha generado un estado que ya existía en el camino.
 - Cuando no existen reglas aplicables al último nodo de la lista (último nodo del grafo explícito).

Estrategia retroactiva



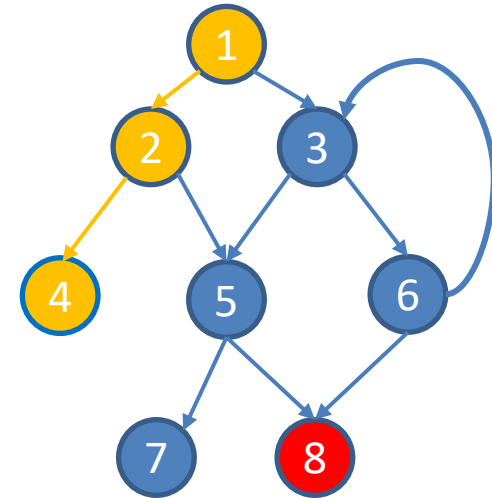
Representación en memoria del g.e.



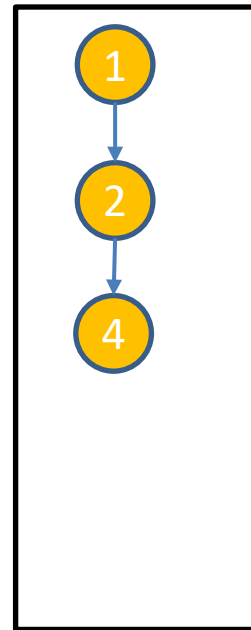
Considerar ahora esta situación donde el nodo4 es un nodo sin sucesores (hijos) en el grafo.

Estrategia retroactiva

Representación en memoria del g.e.



Supongamos que el algoritmo decide seleccionar la acción que lleva al nodo4. En esta situación realizaría un backtracking porque no hay acciones que se puedan aplicar en el nodo4



Selecciona una acción del último nodo insertado en memoria y que no haya sido seleccionada previamente para el nodo actual

Inicio

¿Está el objetivo en el g.e.?

NO

SI

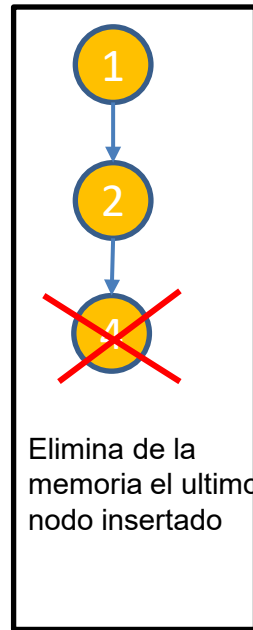
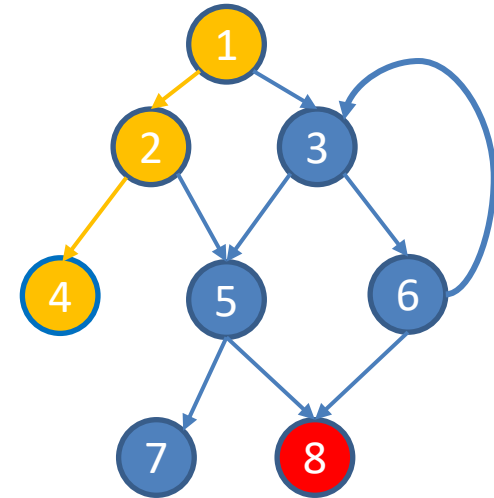
Return solución

Seleccionar A

Transformar g.e. (generar sucesores)

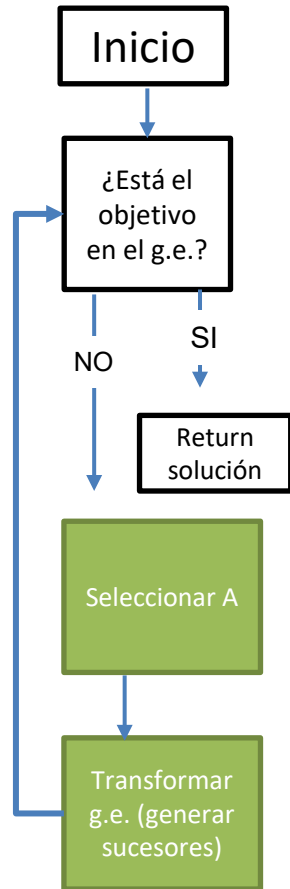
Estrategia retroactiva

Representación en memoria del g.e.

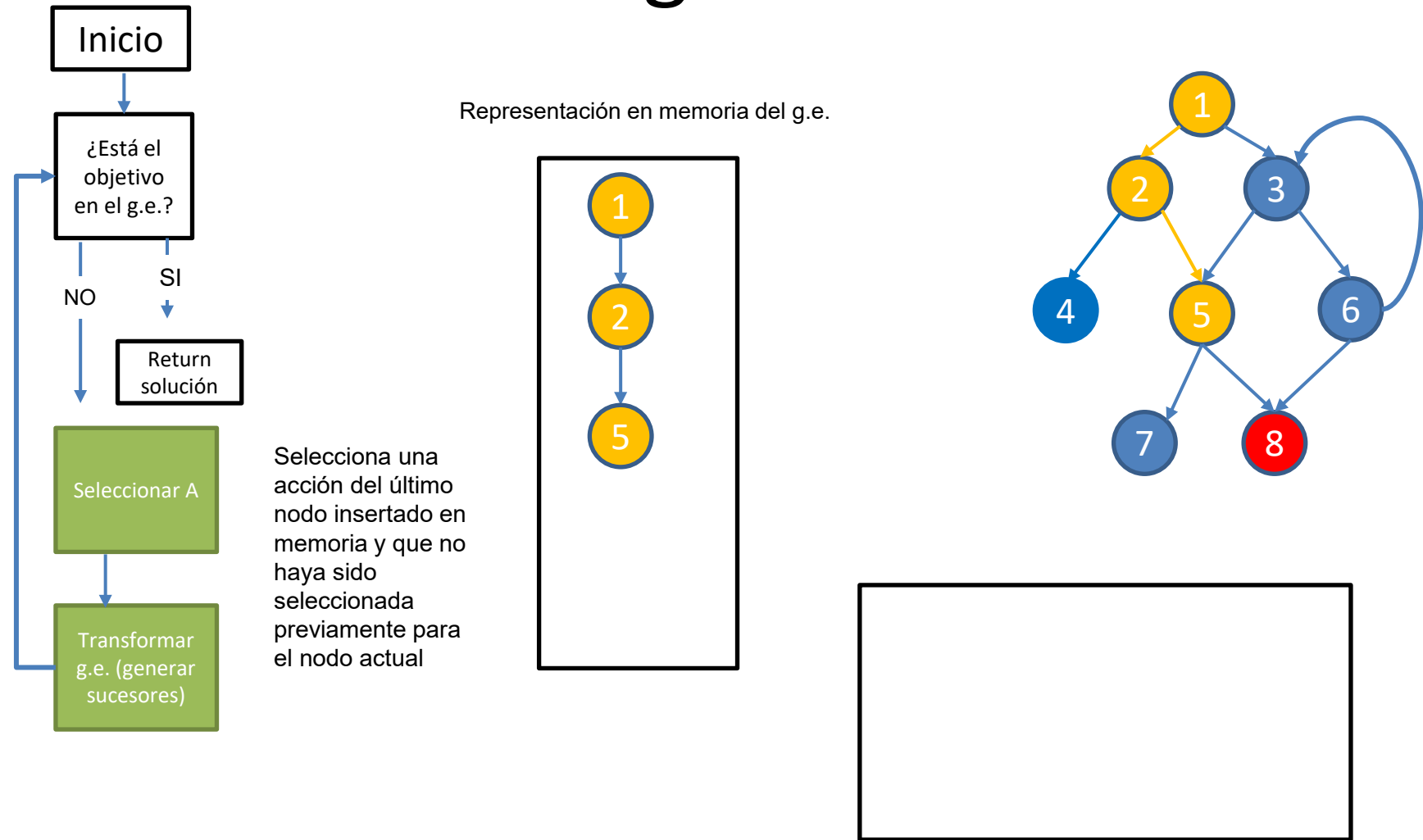


En esta situación realizaría un backtracking porque no hay acciones que se puedan aplicar en el nodo4

Selecciona una acción del último nodo insertado en memoria y que no haya sido seleccionada previamente para el nodo actual



Estrategia retroactiva



Búsqueda en grafos/árboles

- **Grafo explícito:** En memoria se guardan todos los estados (o nodos) generados hasta el momento, de forma que la búsqueda puede proseguir por cualquiera de ellos.
 - En **búsqueda en árboles** se usa una lista denominada **Frontera (o Abiertos)** donde se almacenan los nodos sucesores que se han generado en etapas del algoritmo y que están pendientes de explorar.

Búsqueda en grafos/árboles

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

Inicio

loop

Escoger nodo de
Frontera

El nodo es
objetivo?

NO

SI

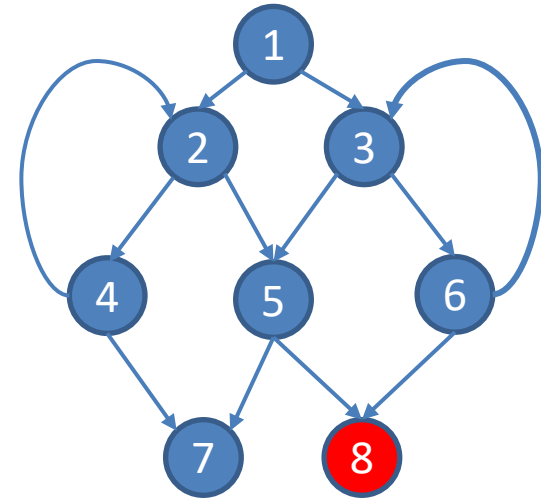
Return
solución

Seleccionar A

Transformar
g.e. (generar
sucesores)

Búsqueda en árboles

Lista de nodos Frontera (también llamada Abiertos).



```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

Inicio

loop

Escoger nodo de
Frontera

El nodo es
objetivo?

NO

SI

Return
solución

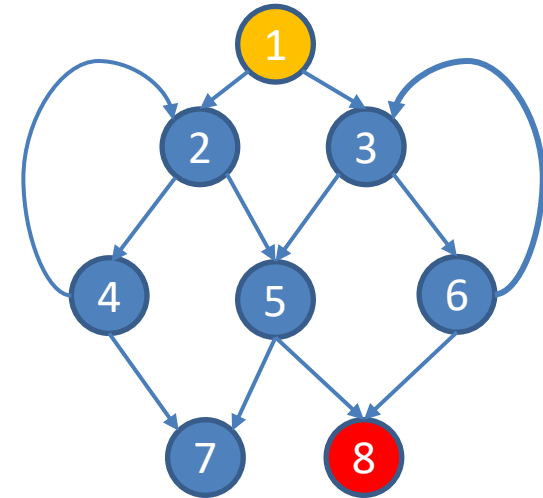
Seleccionar A

Transformar
g.e. (generar
sucesores)

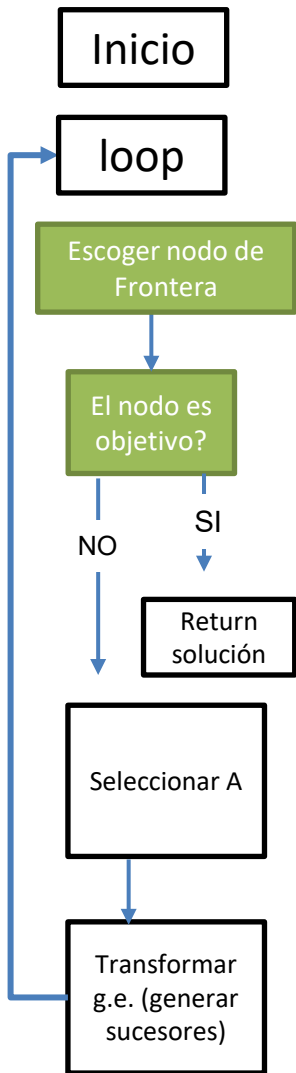
Búsqueda en árboles

Frontera (Abiertos)

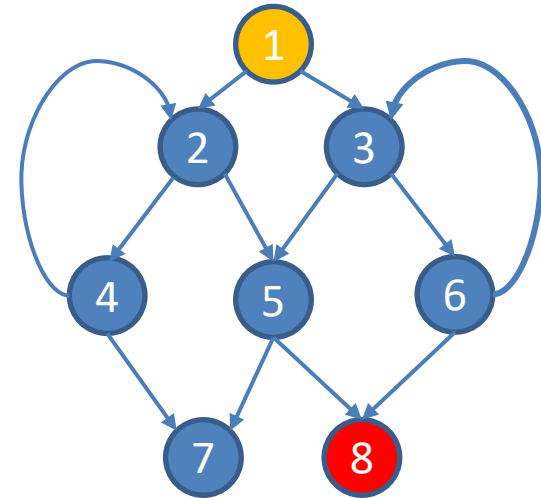
1



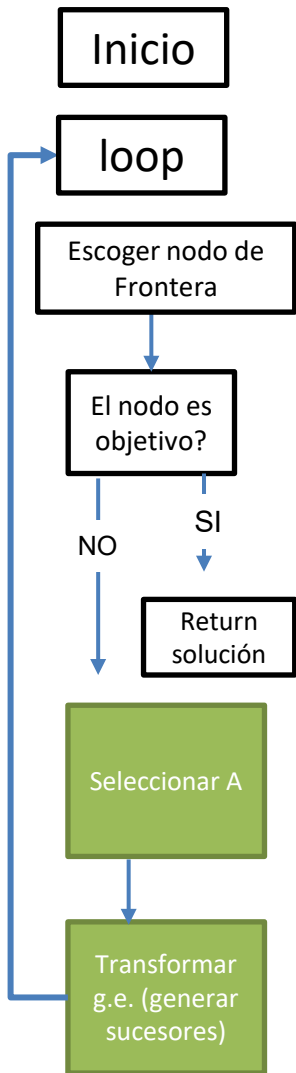
Búsqueda en árboles



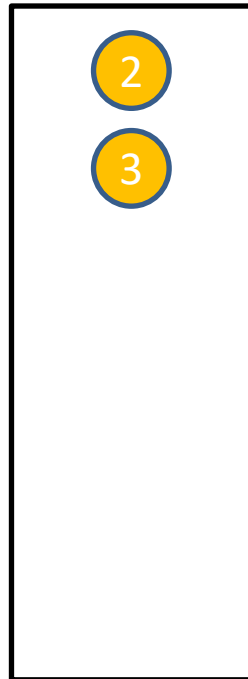
Frontier (Abiertos)



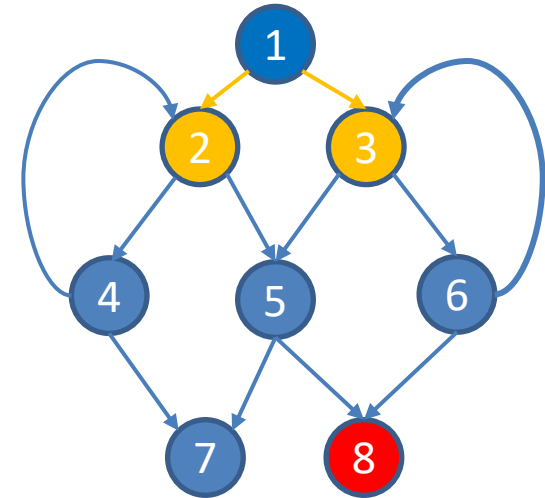
Búsqueda en árboles



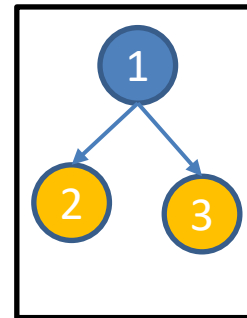
Frontera (Abiertos)



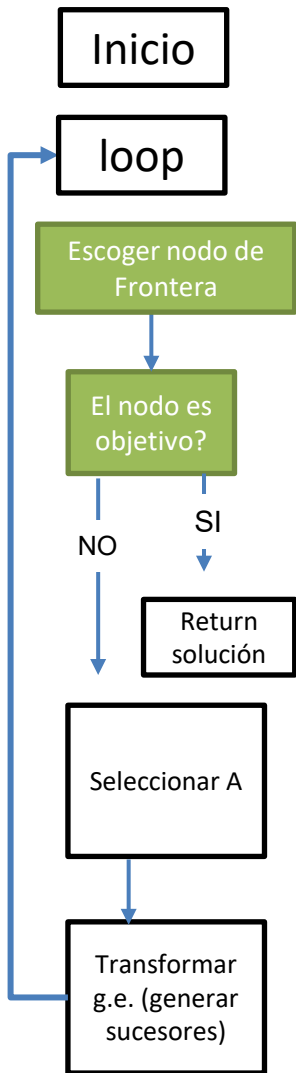
Selecciona todas las acciones del nodo escogido y añade todos los sucesores a Frontera (o Abiertos)



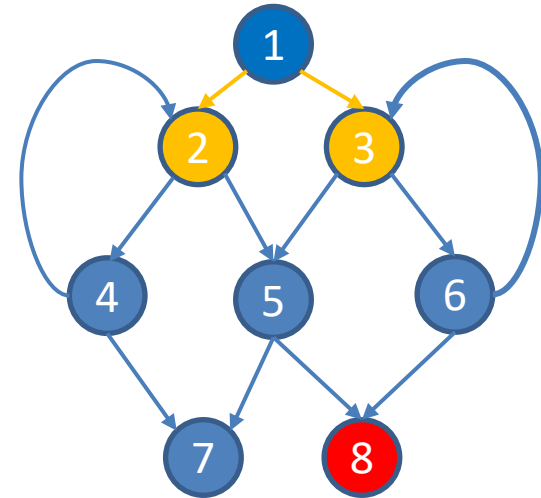
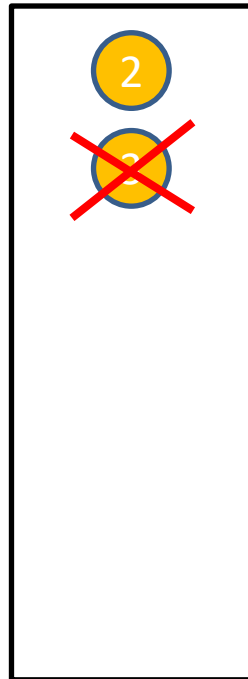
Expansión de un nodo generando sus nodos hijos o sucesores



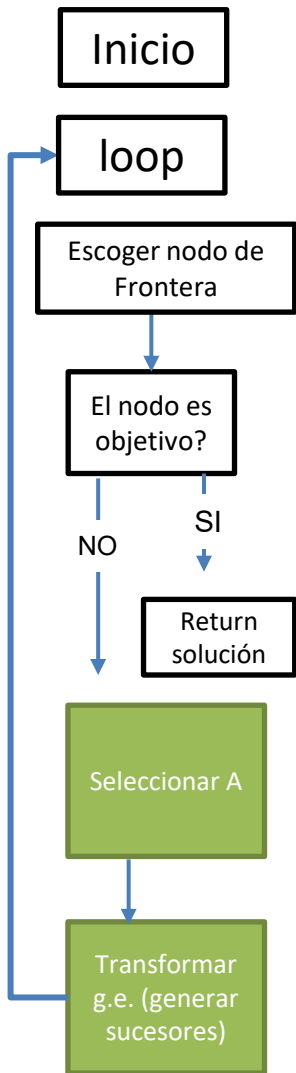
Búsqueda en árboles



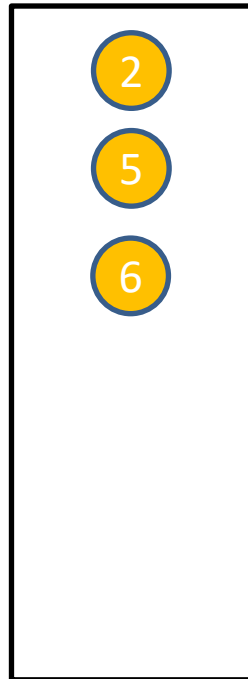
Frontera (Abiertos)



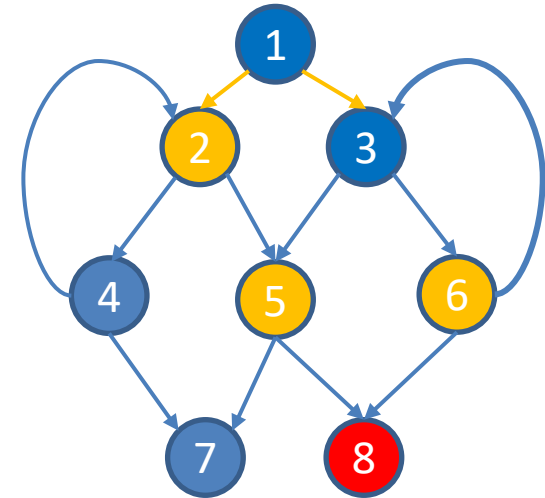
Búsqueda en árboles



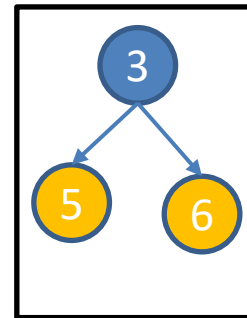
Frontera (Abiertos)



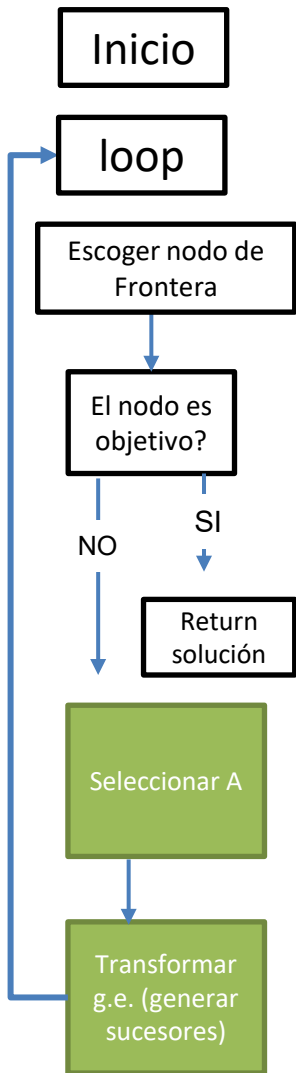
Selecciona todas las acciones del nodo escogido y añade todos los sucesores a Frontera (o Abiertos)



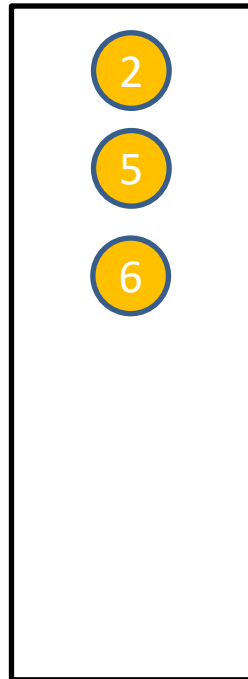
Expansión de un nodo generando sus nodos hijos o sucesores



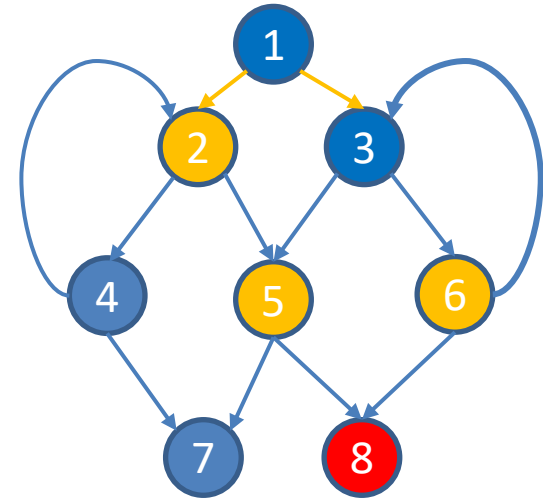
Búsqueda en árboles



Frontera (Abiertos)



Selecciona todas las acciones del nodo escogido y añade todos los sucesores a Frontera (o Abiertos)



Observar que tras escoger el nodo6, se insertará el nodo3 (otra vez). Puede haber un ciclo infinito

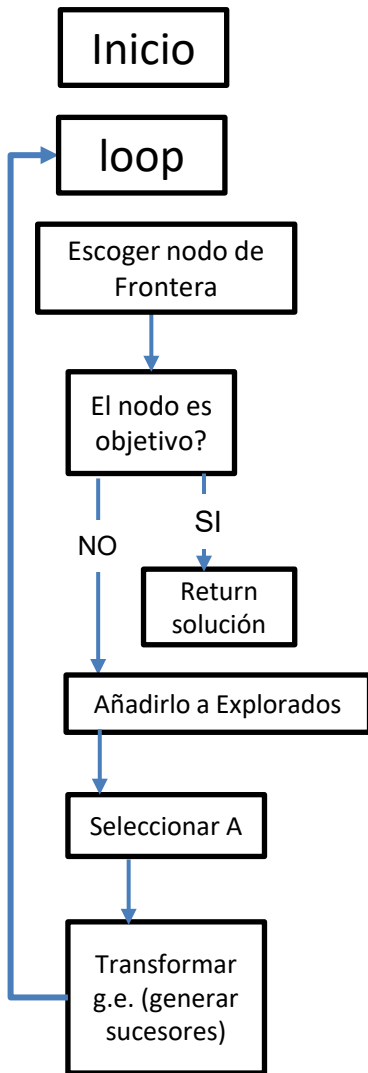
Búsqueda en grafos/árboles

- **Grafo explícito:** En memoria se guardan todos los estados (o nodos) generados hasta el momento, de forma que la búsqueda puede proseguir por cualquiera de ellos.
 - En **búsqueda en grafos** se usa una **lista adicional** llamada **Explorados (o Cerrados, o Visitados)** donde se almacenan los nodos que fueron frontera y que se han explorado (para llevar una traza de los nodos que ya se han visitado).

Búsqueda en grafos/árboles

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set
```

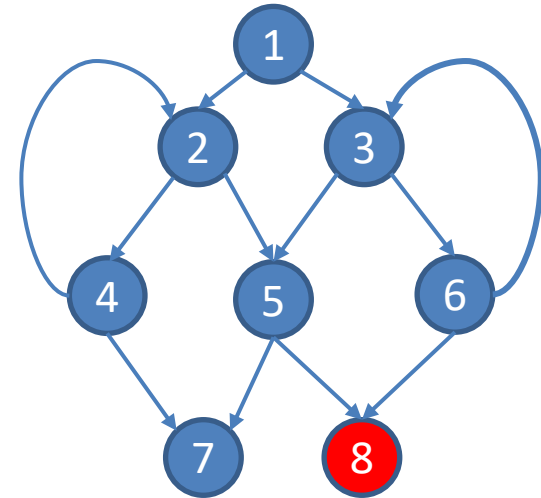
Búsqueda en grafos



Lista de nodos
Frontera
(también llamada
Abiertos).



Lista de nodos
Explorados
(también llamada
Cerrados o
Visitados).



```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set
```

Inicio

loop

Escoger nodo de
Frontera

El nodo es
objetivo?

NO

SI

Return
solución

Añadirlo a Explorados

Seleccionar A

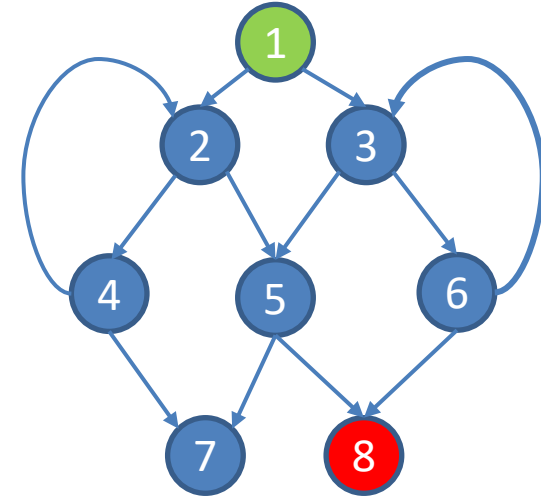
Transformar
g.e. (generar
sucesores)

Lista de nodos
Frontera
(también llamada
Abiertos).

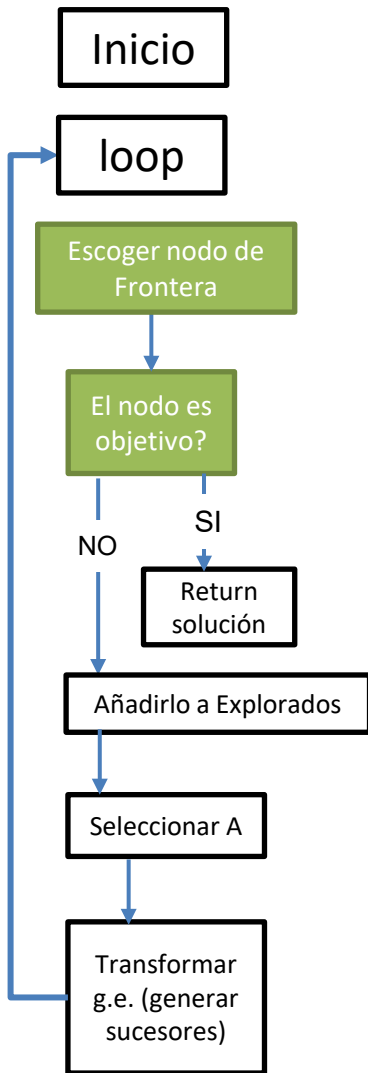
1

Lista de nodos
Visitados
(también llamada
Cerrados).

Búsqueda en grafos



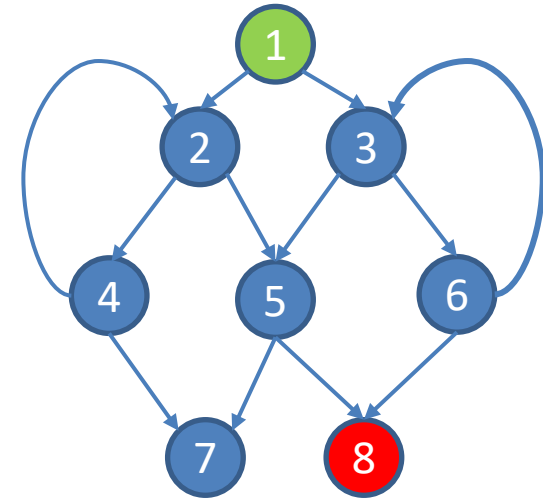
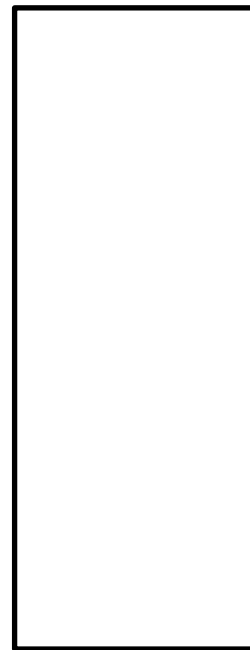
Búsqueda en grafos



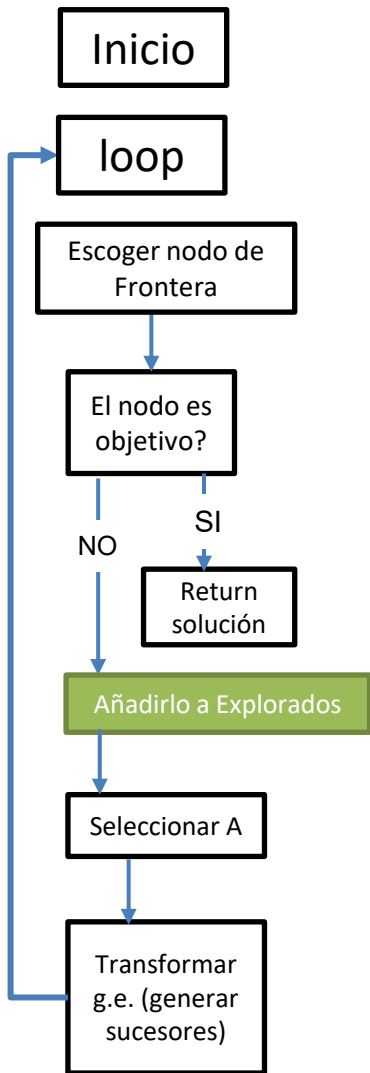
Lista de nodos
Frontera
(también llamada
Abiertos).



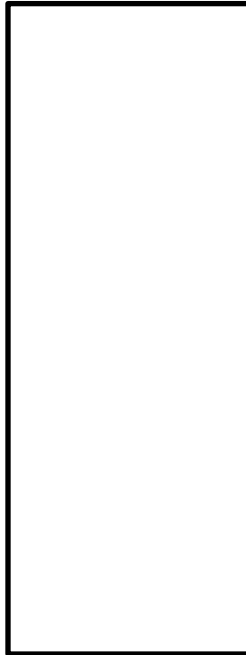
Lista de nodos
Visitados
(también llamada
Cerrados).



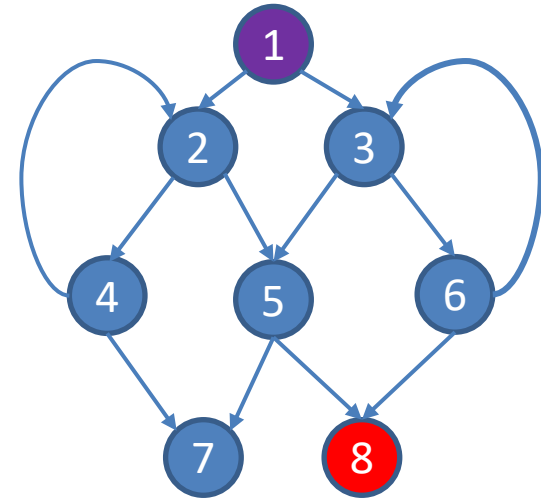
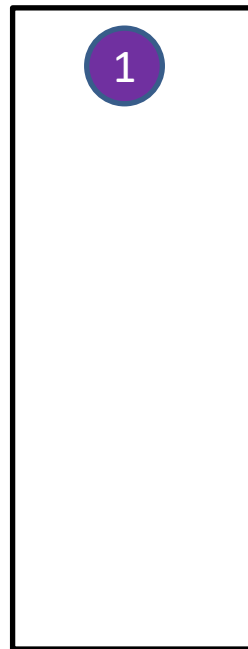
Búsqueda en grafos



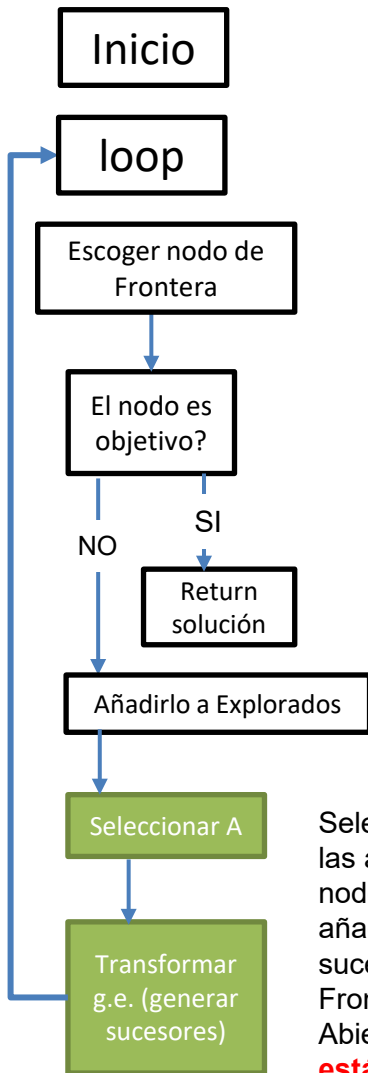
Lista de nodos
Frontera
(también llamada
Abiertos).



Lista de nodos
Visitados
(también llamada
Cerrados).



Búsqueda en grafos



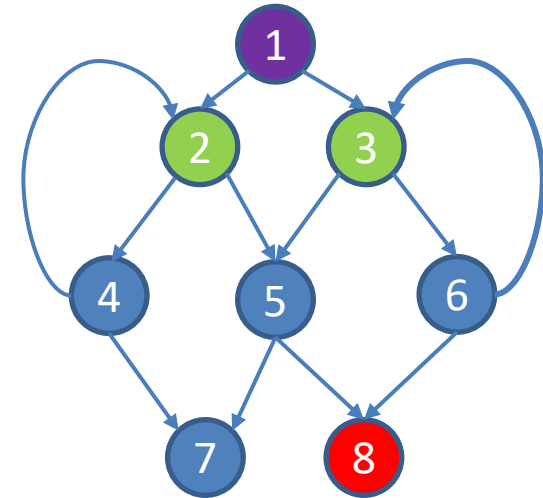
Lista de nodos
Frontera
(también llamada
Abiertos).



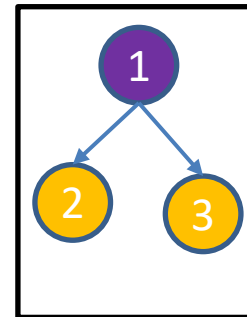
Lista de nodos
Visitados
(también llamada
Cerrados).



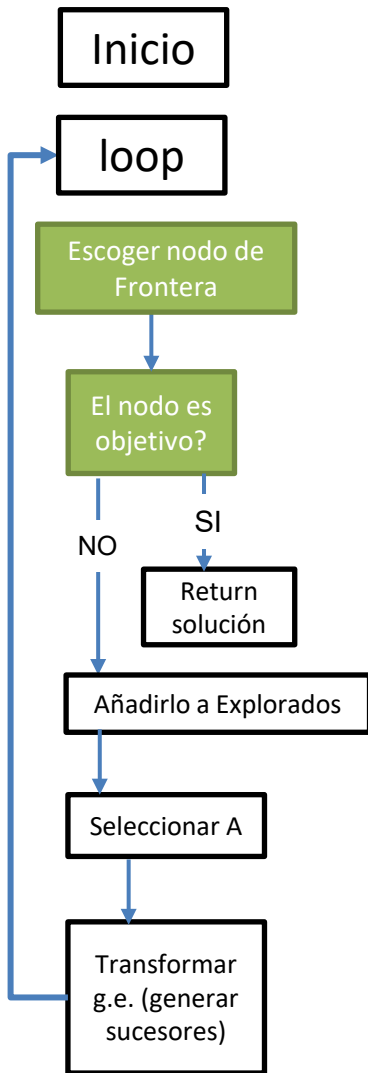
Selecciona todas
las acciones del
nodo escogido y
añade todos los
sucesores a
Frontera (o
Abiertos), **si no
están ya en
Explorados o en
Frontera.**



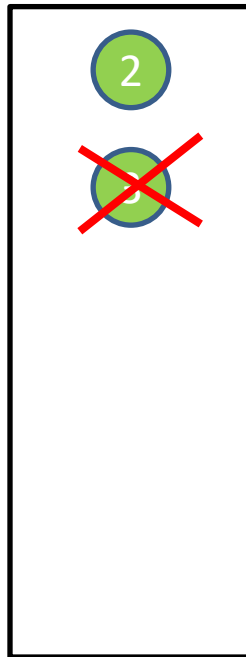
**Expansión de un nodo
generando sus nodos hijos
o sucesores**



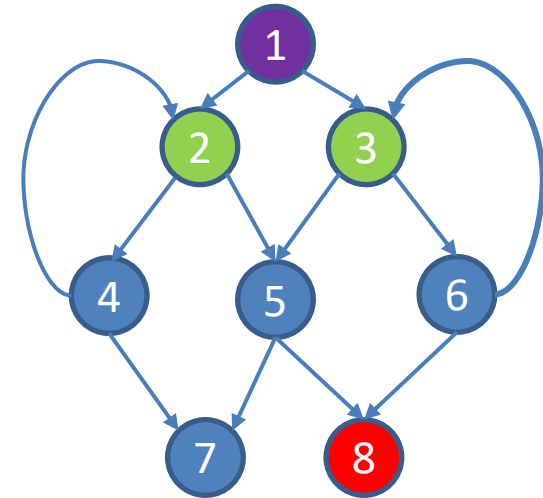
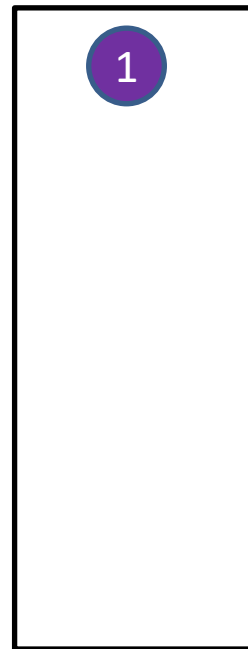
Búsqueda en grafos



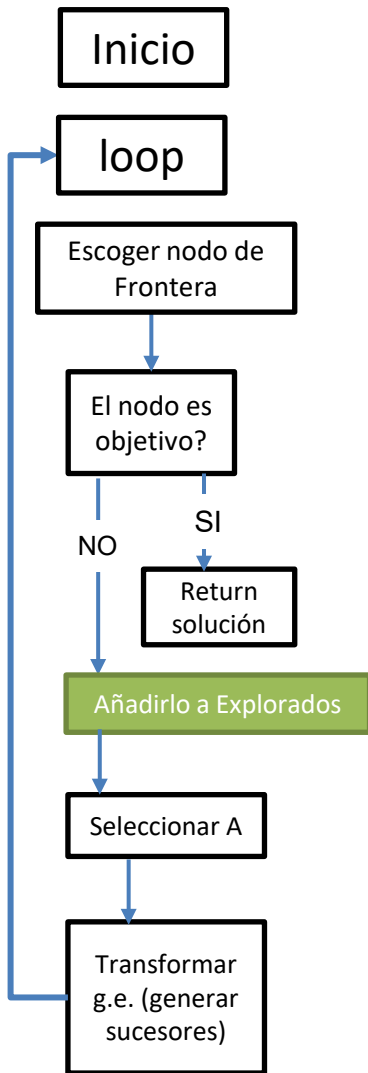
Lista de nodos
Frontera
(también llamada
Abiertos).



Lista de nodos
Visitados
(también llamada
Cerrados).



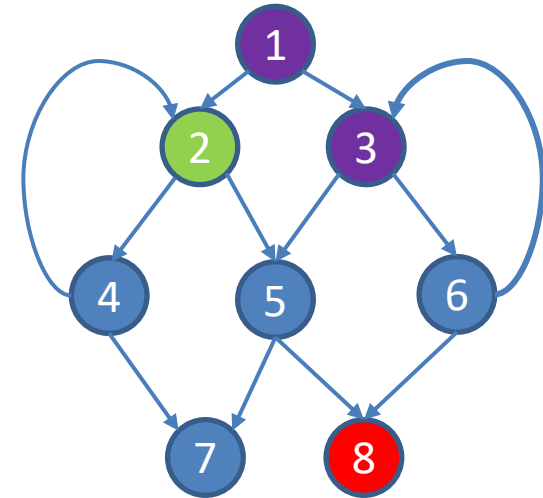
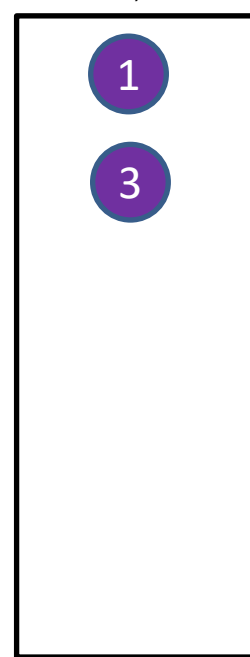
Búsqueda en grafos



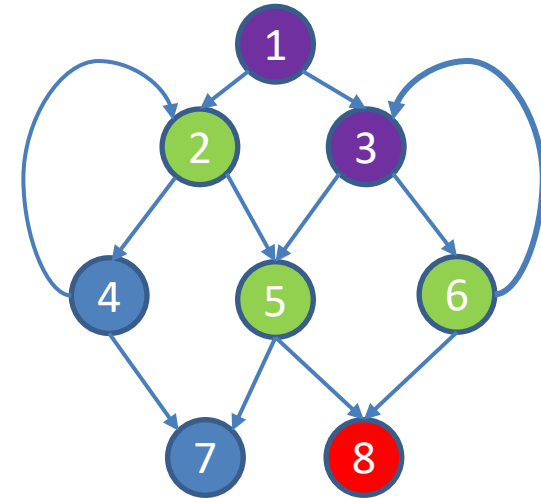
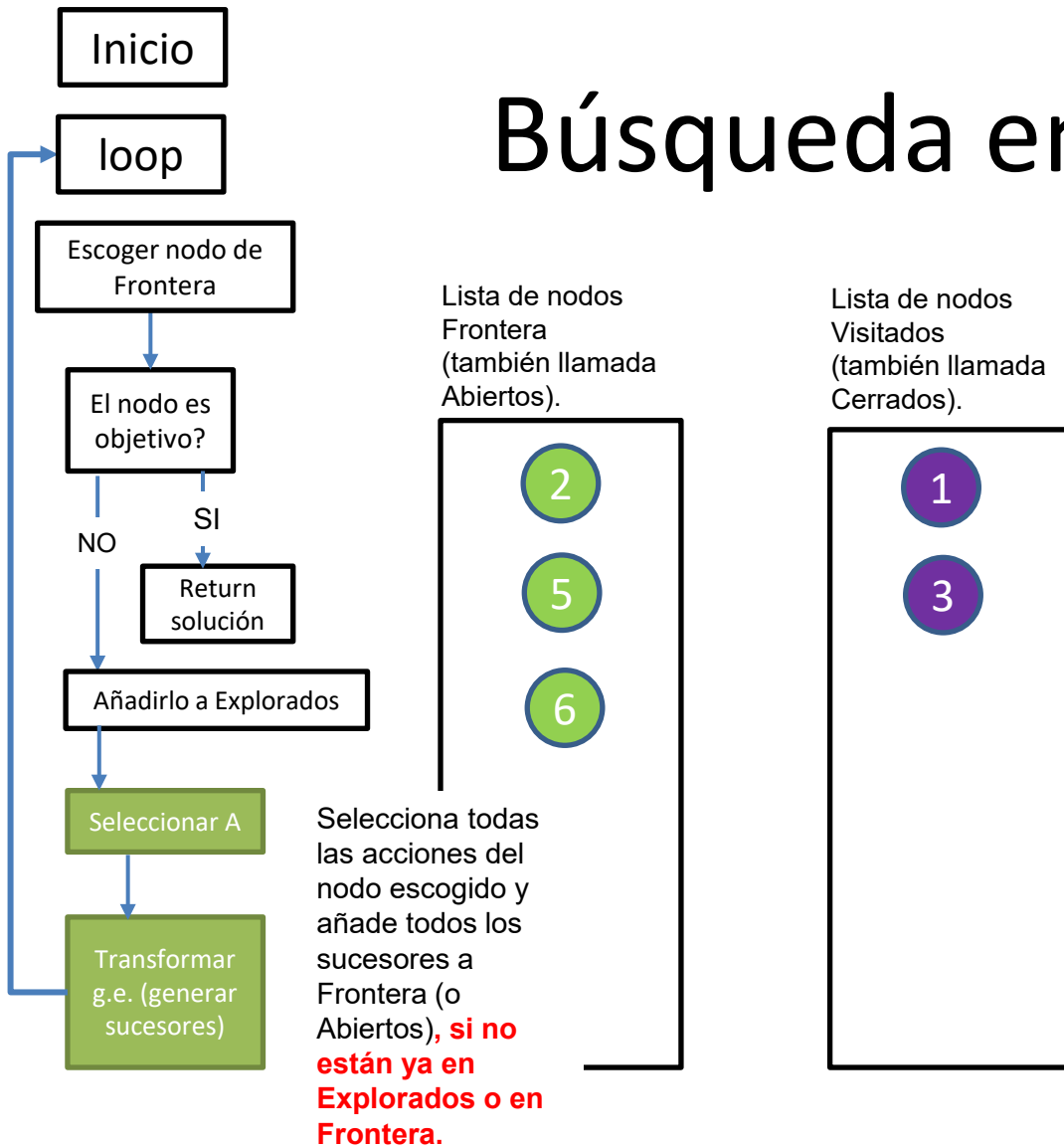
Lista de nodos
Frontera
(también llamada
Abiertos).



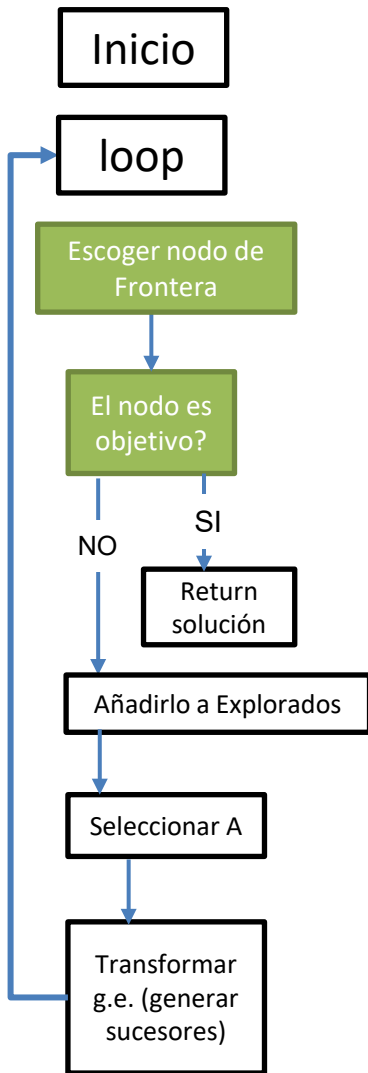
Lista de nodos
Visitados
(también llamada
Cerrados).



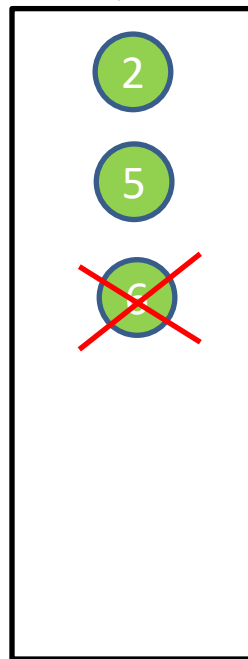
Búsqueda en grafos



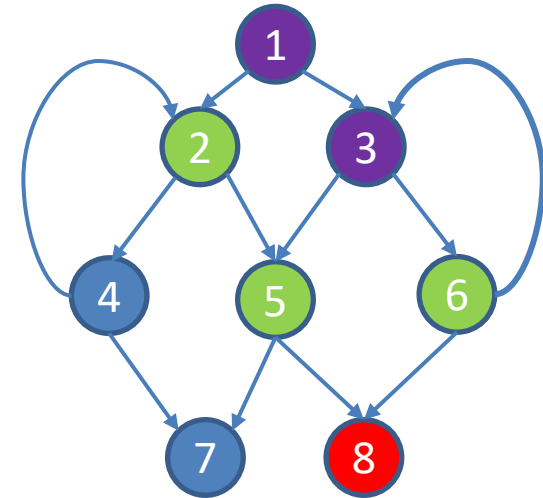
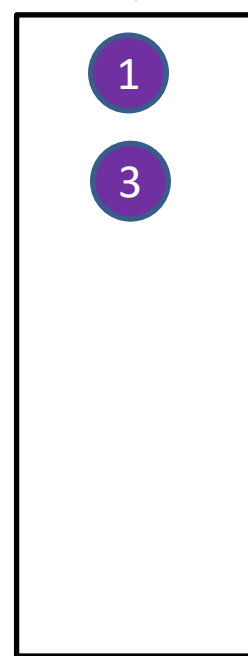
Búsqueda en grafos



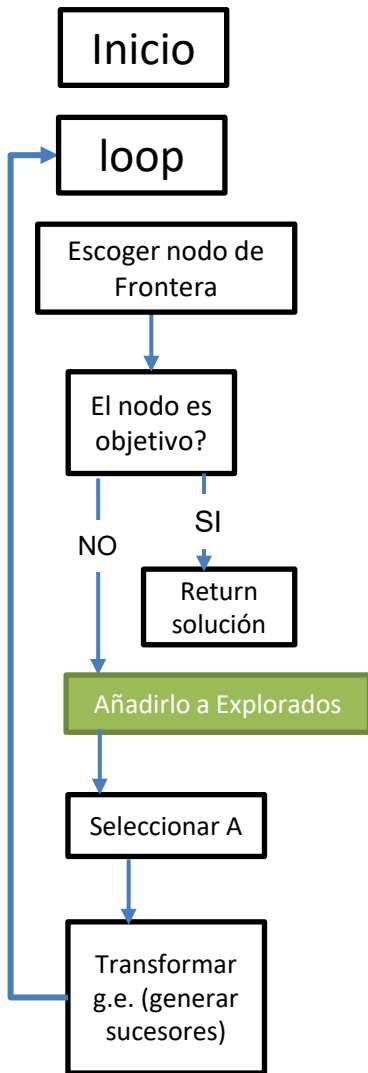
Lista de nodos
Frontera
(también llamada
Abiertos).



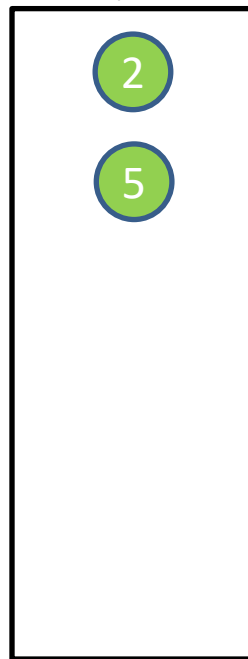
Lista de nodos
Visitados
(también llamada
Cerrados).



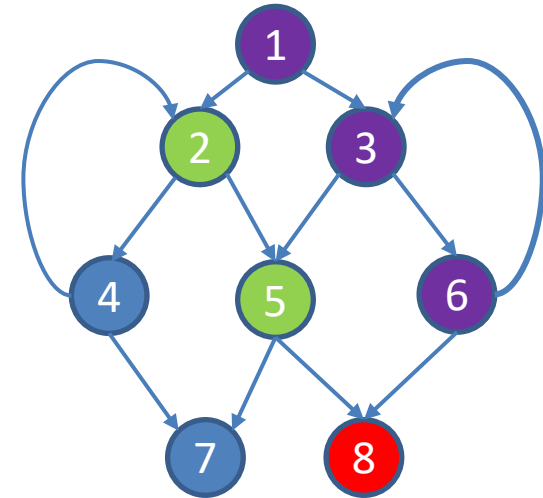
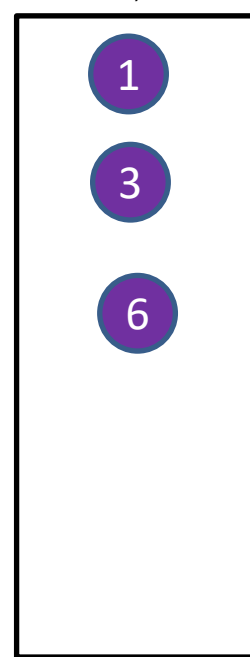
Búsqueda en grafos



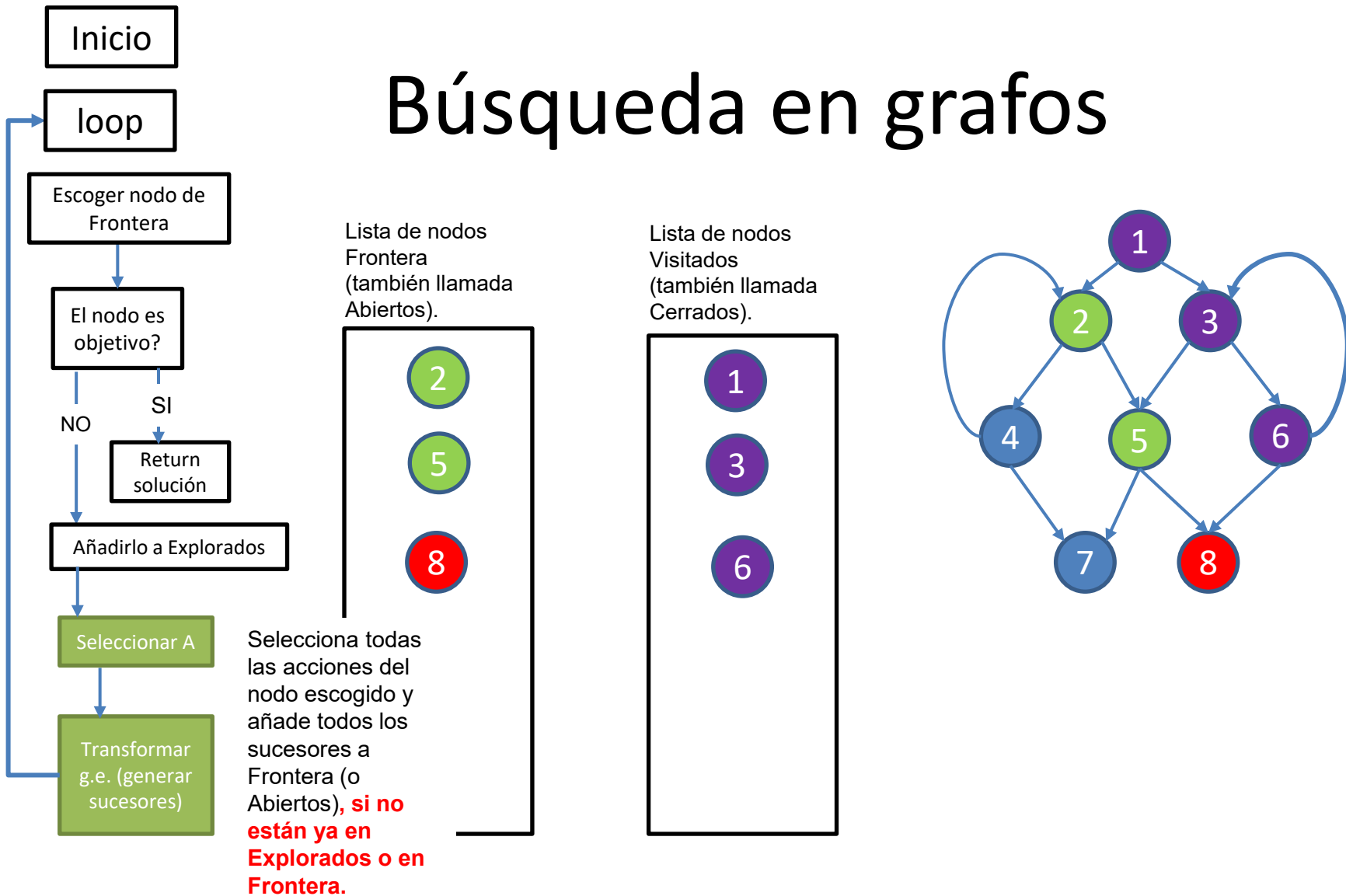
Lista de nodos
Frontera
(también llamada
Abiertos).



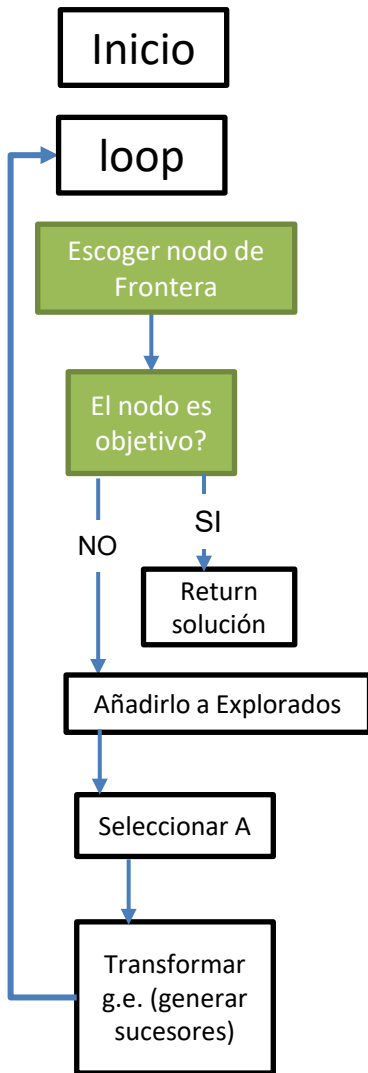
Lista de nodos
Visitados
(también llamada
Cerrados).



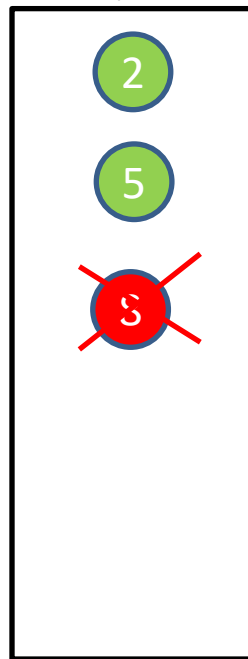
Búsqueda en grafos



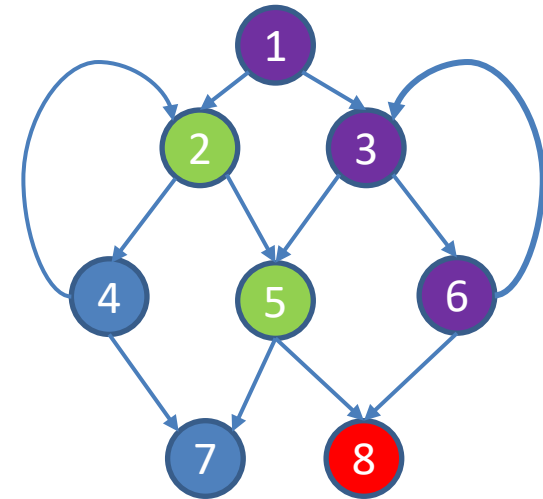
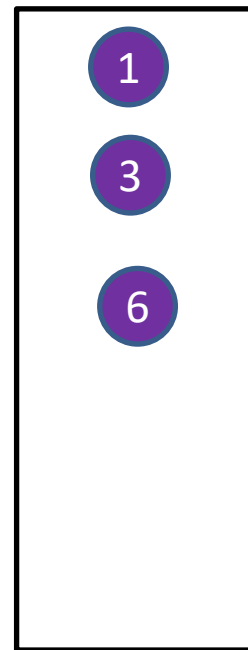
Búsqueda en grafos



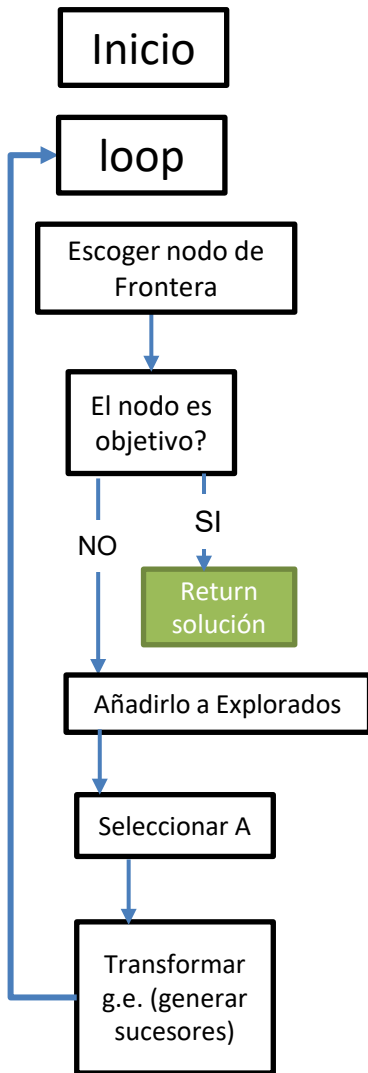
Lista de nodos
Frontera
(también llamada
Abiertos).



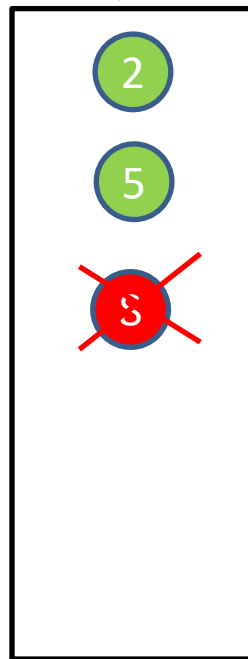
Lista de nodos
Visitados
(también llamada
Cerrados).



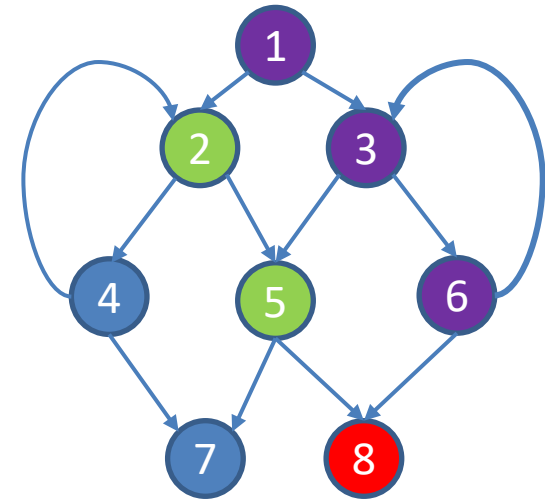
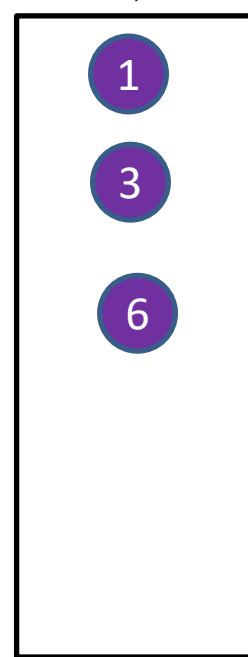
Búsqueda en grafos



Lista de nodos
Frontera
(también llamada
Abiertos).



Lista de nodos
Visitados
(también llamada
Cerrados).

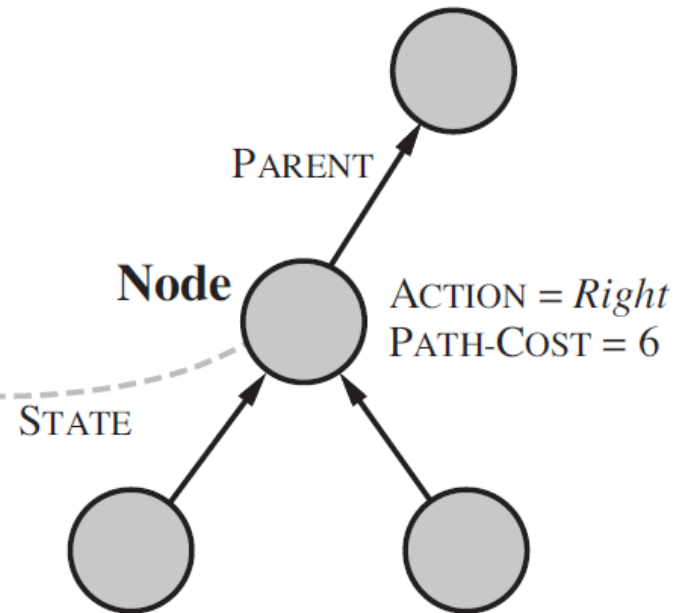


¿Qué devuelve como solución?

Infraestructura para los algoritmos de búsqueda

n.STATE
n.PARENT
n.ACTION
n.PATH-COST

5	4	
6	1	8
7	3	2



<https://tristanpenman.com/demos/n-puzzle/>

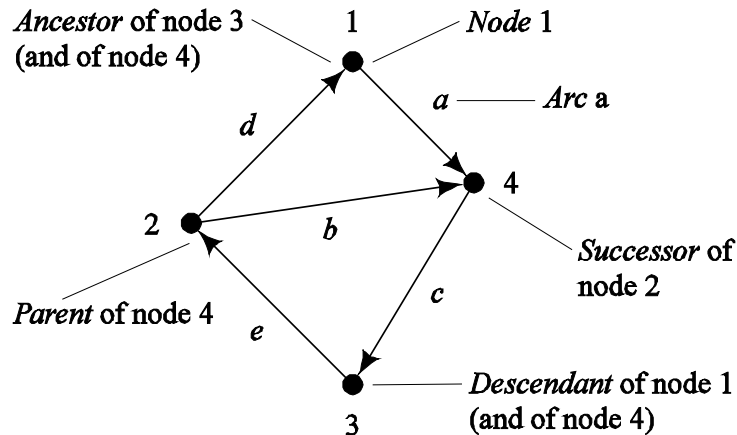
Infraestructura para los algoritmos de búsqueda

function CHILD-NODE(*problem, parent, action*) **returns** a node
return a node with
 STATE = *problem.RESULT(parent.STATE, action)*,
 PARENT = *parent*, ACTION = *action*,
 PATH-COST = *parent.PATH-COST* + *problem.STEP-COST(parent.STATE, action)*

Función que implementa el **modelo de transición**, e.d., la forma de calcular el **estado sucesor a uno dado**, o de otra forma, calcular los **efectos de aplicación de una acción**

Búsqueda en grafos/árboles

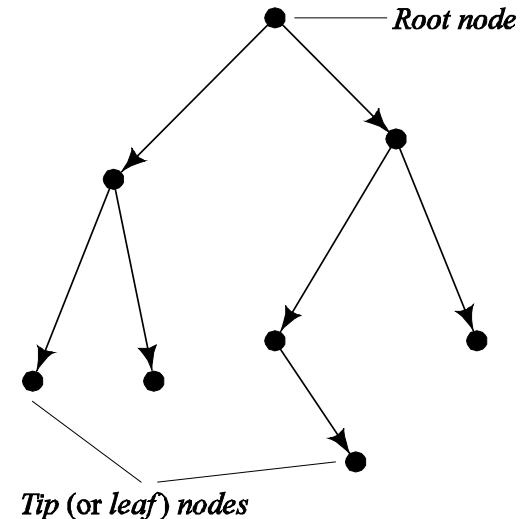
Graph notation



$c(a)$, alternatively $c(1, 4)$, is the *cost* of arc a
(d, a), alternatively $(2, 1, 4)$, is a *path* from node 2 to node 4

© 1998 Morgan Kaufman Publishers

Tree notation



Parámetros importantes que determinan el tamaño del espacio de estados:

Factor de ramificación = b

Profundidad del árbol de búsqueda = d

$$\text{Tamaño} \approx O(b^d)$$

Medidas del comportamiento de un sistema de búsqueda

- **Completitud:** hay garantía de encontrar la solución si esta existe
- **Optimalidad:** hay garantía de encontrar la solución óptima
- **Complejidad en tiempo:** ¿Cuánto tiempo se requiere para encontrar la solución?
- **Complejidad en espacio:** ¿Cuánta memoria se requiere para realizar la búsqueda?

Búsqueda sin información

- Búsqueda en anchura
- Búsqueda en profundidad
- Búsqueda con costo
- Descenso Iterativo

Búsqueda en anchura

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

node \leftarrow a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

frontier \leftarrow a FIFO queue with *node* as the only element

explored \leftarrow an empty set

loop do

if EMPTY?(*frontier*) **then return** failure

node \leftarrow POP(*frontier*) /* chooses the shallowest node in *frontier* */

 add *node*.STATE to *explored*

for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**

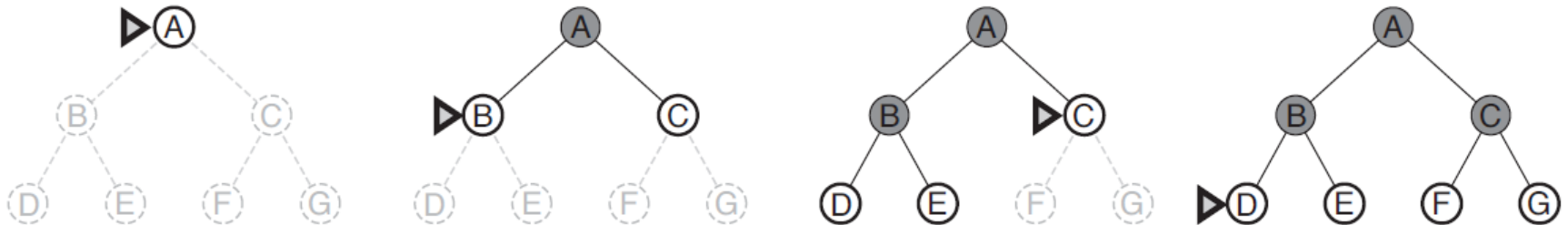
child \leftarrow CHILD-NODE(*problem*, *node*, *action*)

if *child*.STATE is not in *explored* or *frontier* **then**

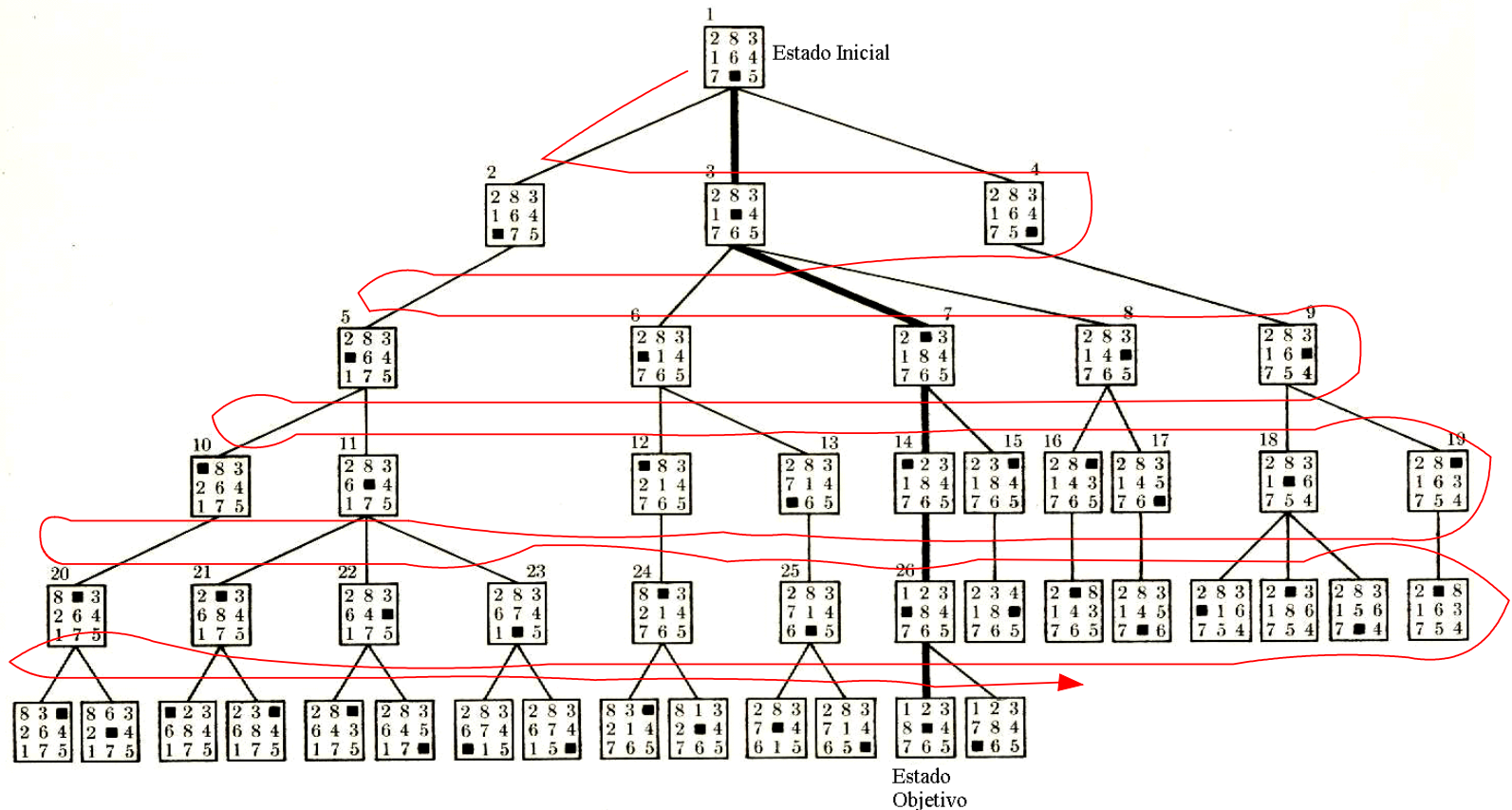
if *problem*.GOAL-TEST(*child*.STATE) **then return** SOLUTION(*child*)

frontier \leftarrow INSERT(*child*, *frontier*)

Búsqueda en anchura



Búsqueda en anchura



Características

- **Completo:** encuentra la solución si existe y el factor de ramificación es finito en cada nodo
- **Optimalidad:** si todos los operadores tienen el mismo coste, encontrara la solución óptima
- **Eficiencia:** buena si las metas están cercanas
- Problema: consume memoria exponencial

Búsqueda con costo uniforme

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier  $\leftarrow$  a priority queue ordered by PATH-COST, with node as the only element
  explored  $\leftarrow$  an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node  $\leftarrow$  POP(frontier) /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier  $\leftarrow$  INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

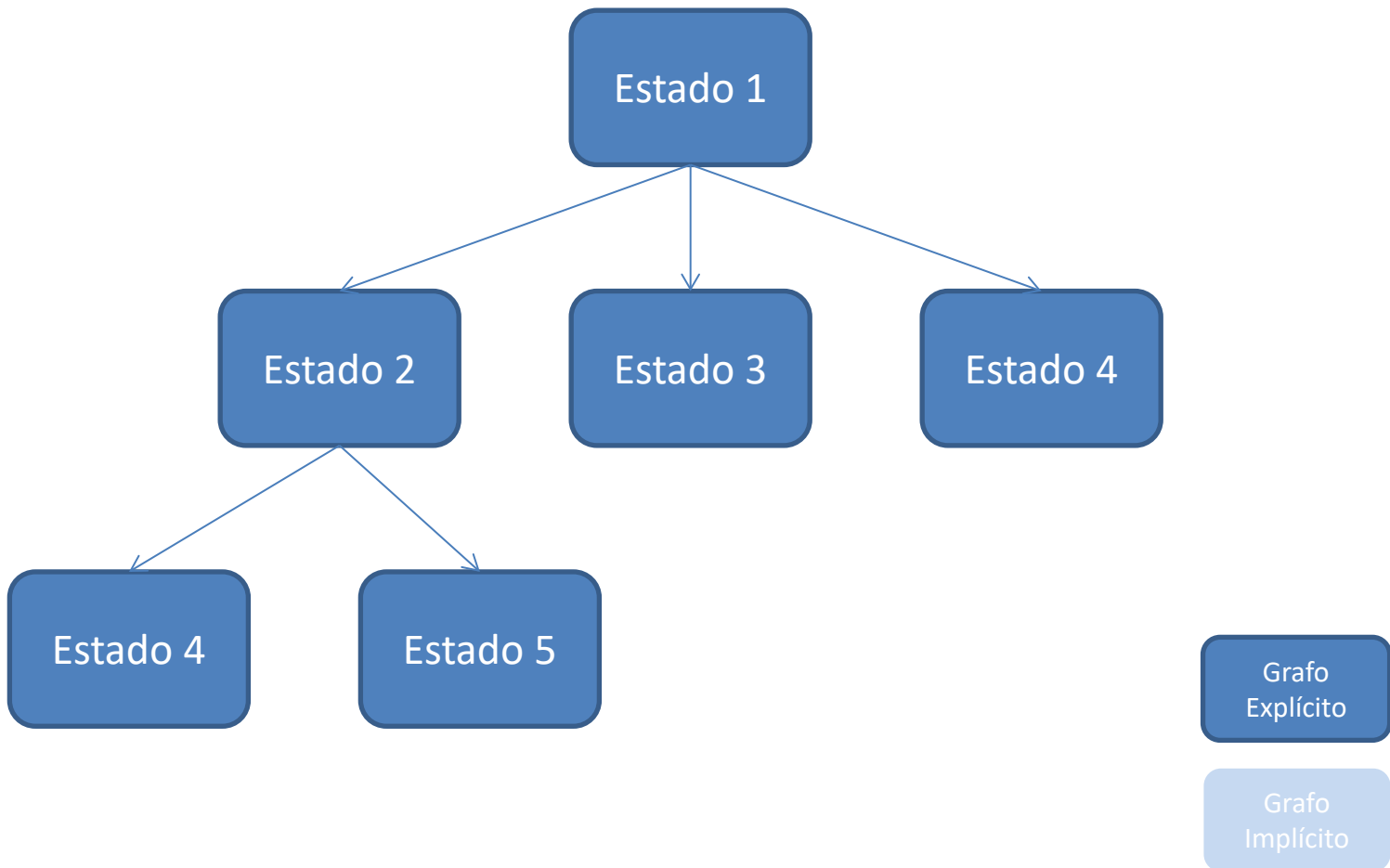
Búsqueda en profundidad

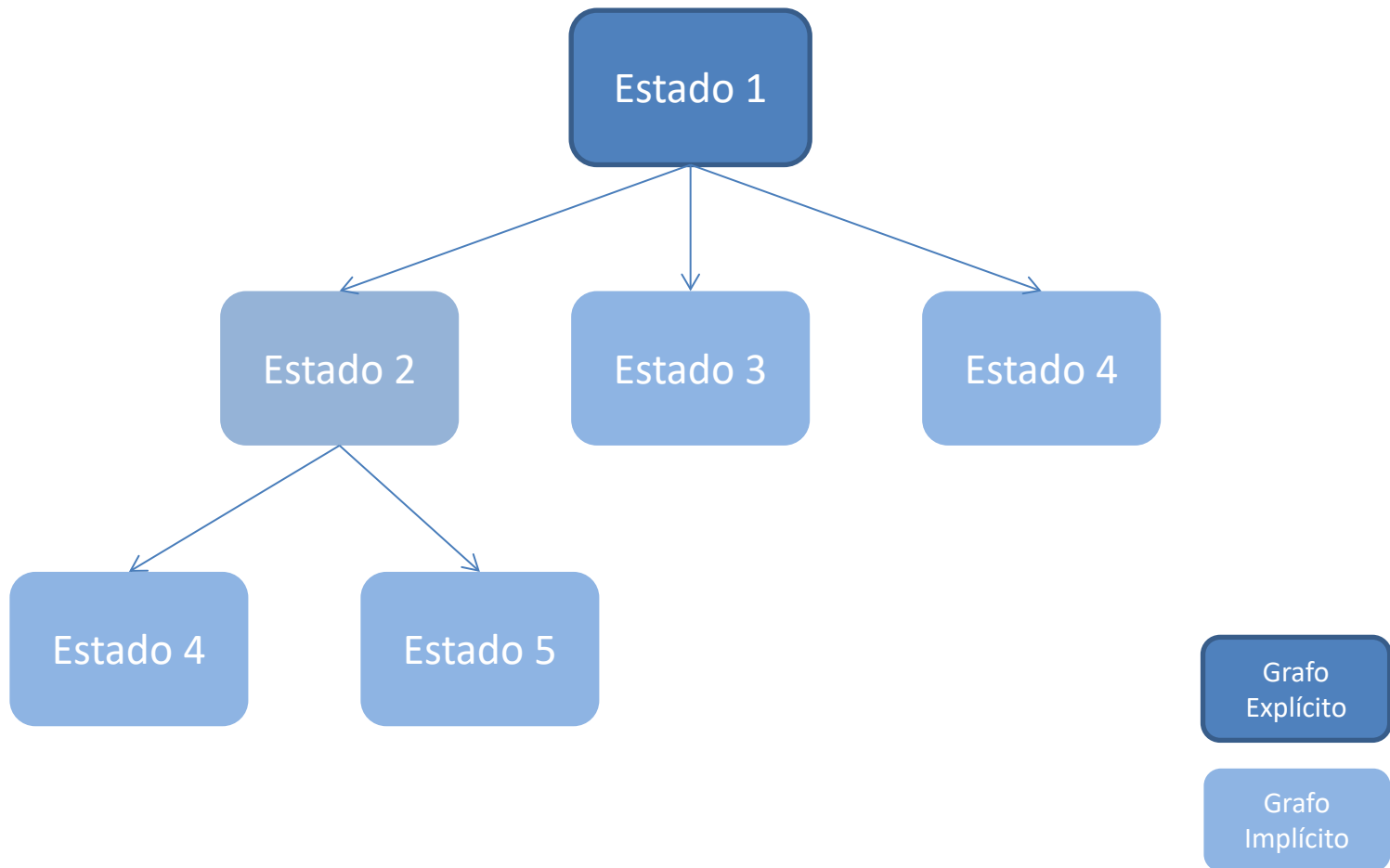
- Igual que la búsqueda en anchura cambiando FIFO por LIFO

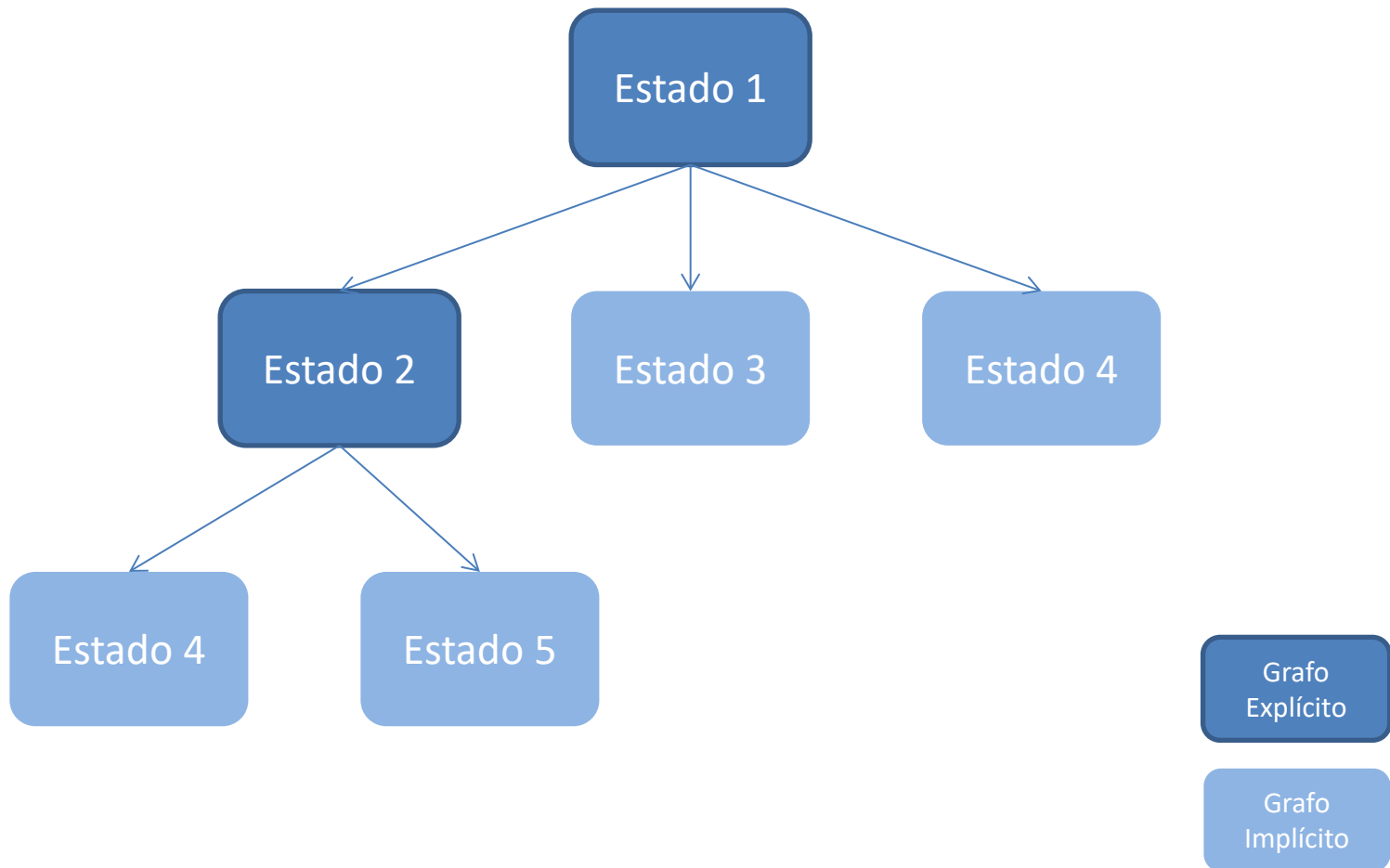
Búsqueda en profundidad retroactiva

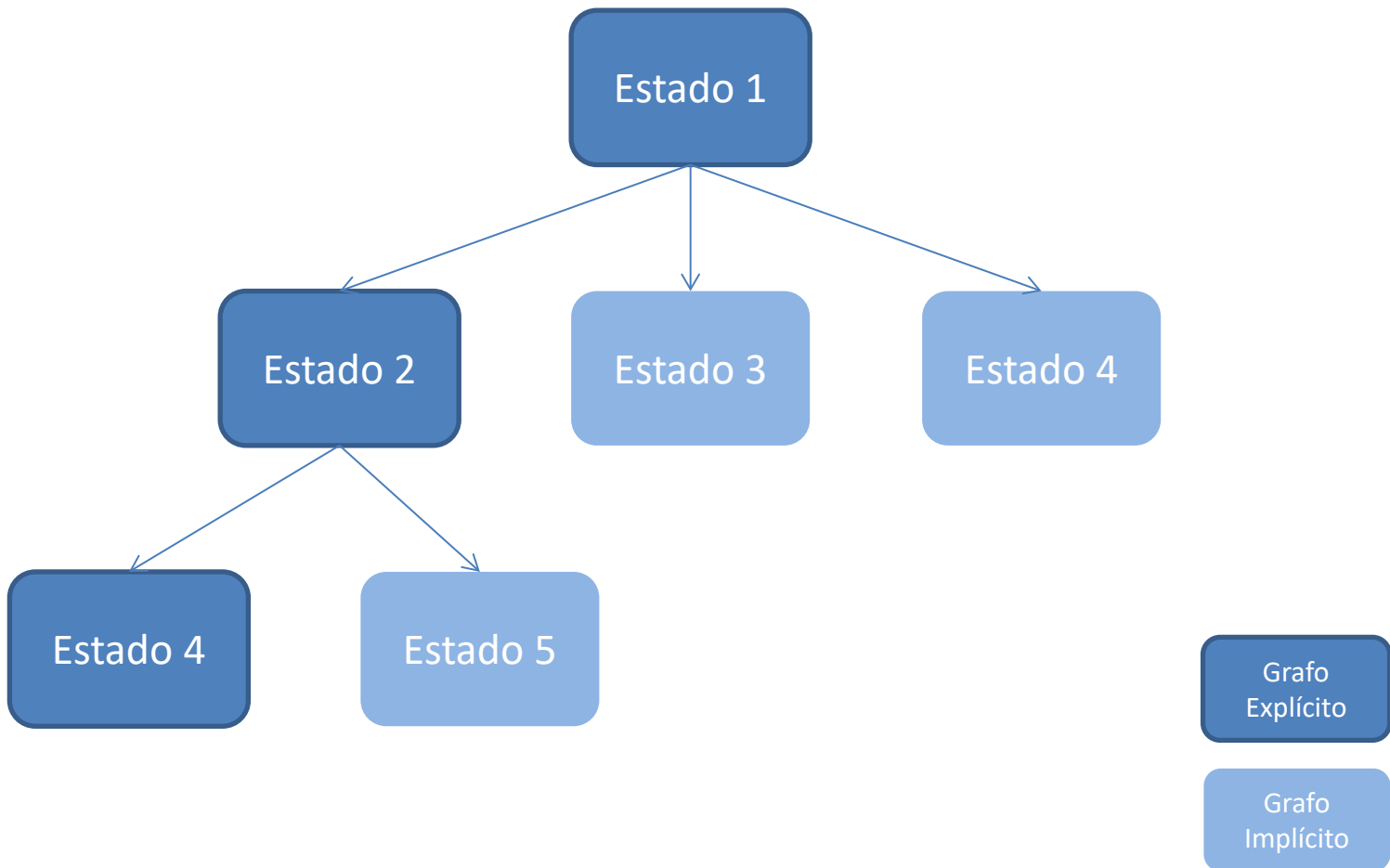
function DEPTH-LIMITED-SEARCH(*problem*, *limit*) **returns** a solution, or failure/cutoff
 return RECURSIVE-DLS(MAKE-NODE(*problem*.INITIAL-STATE), *problem*, *limit*)

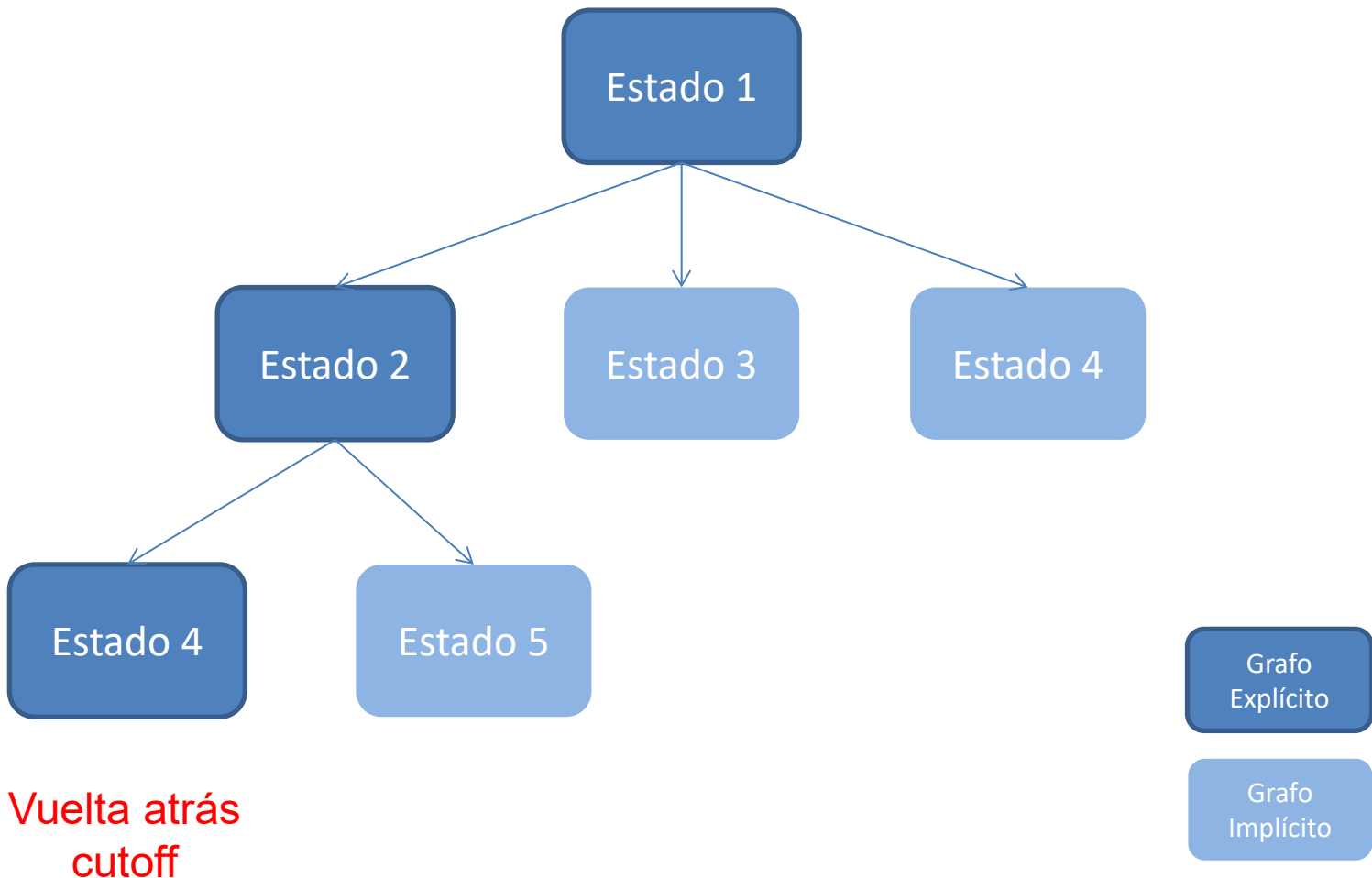
function RECURSIVE-DLS(*node*, *problem*, *limit*) **returns** a solution, or failure/cutoff
 if *problem*.GOAL-TEST(*node*.STATE) **then** **return** SOLUTION(*node*)
 else if *limit* = 0 **then** **return** *cutoff*
 else
 cutoff_occurred? \leftarrow false
 for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
 child \leftarrow CHILD-NODE(*problem*, *node*, *action*)
 result \leftarrow RECURSIVE-DLS(*child*, *problem*, *limit* - 1)
 if *result* = *cutoff* **then** *cutoff_occurred?* \leftarrow true
 else if *result* \neq failure **then** **return** *result*
 if *cutoff_occurred?* **then** **return** *cutoff* **else** **return** failure

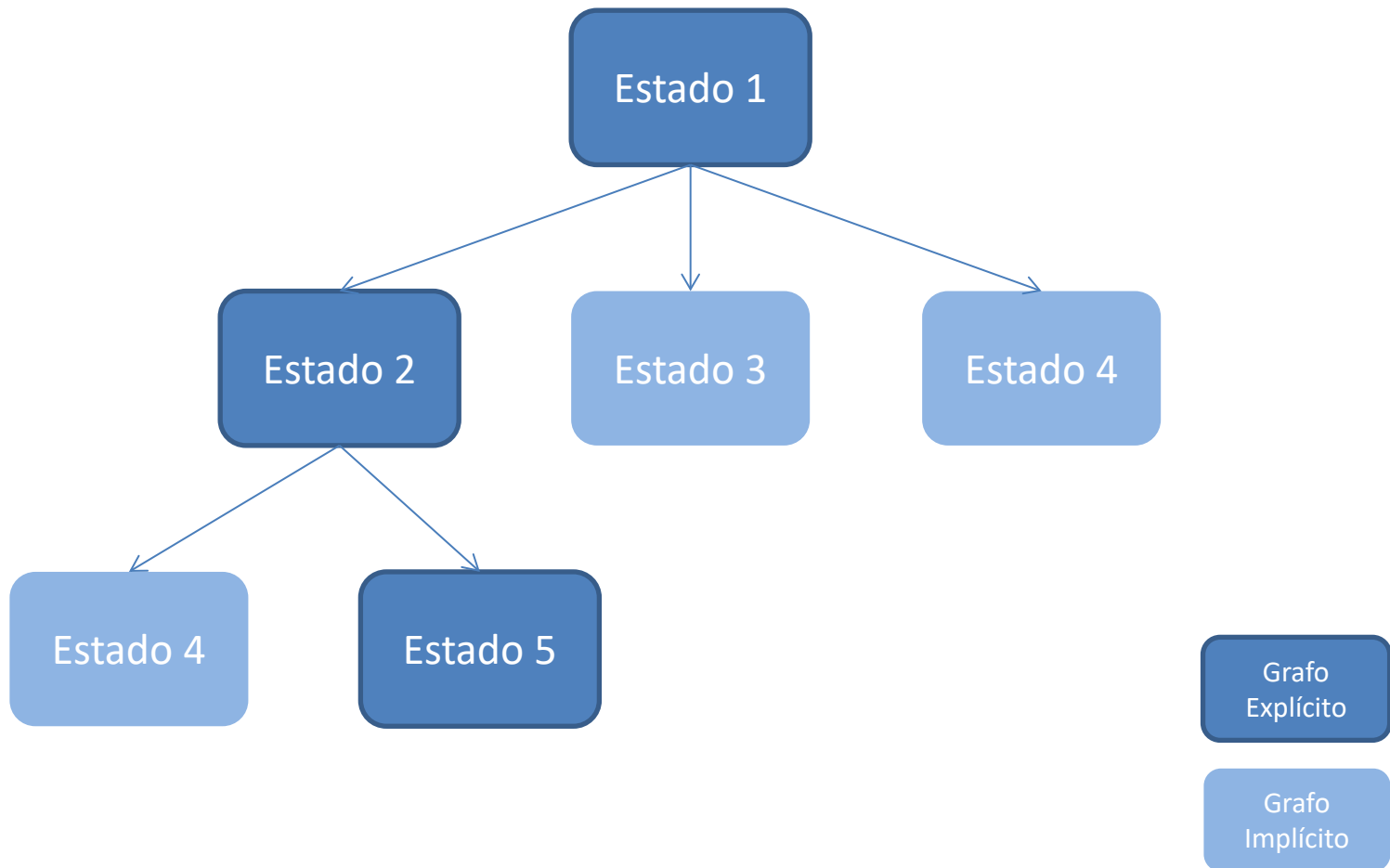


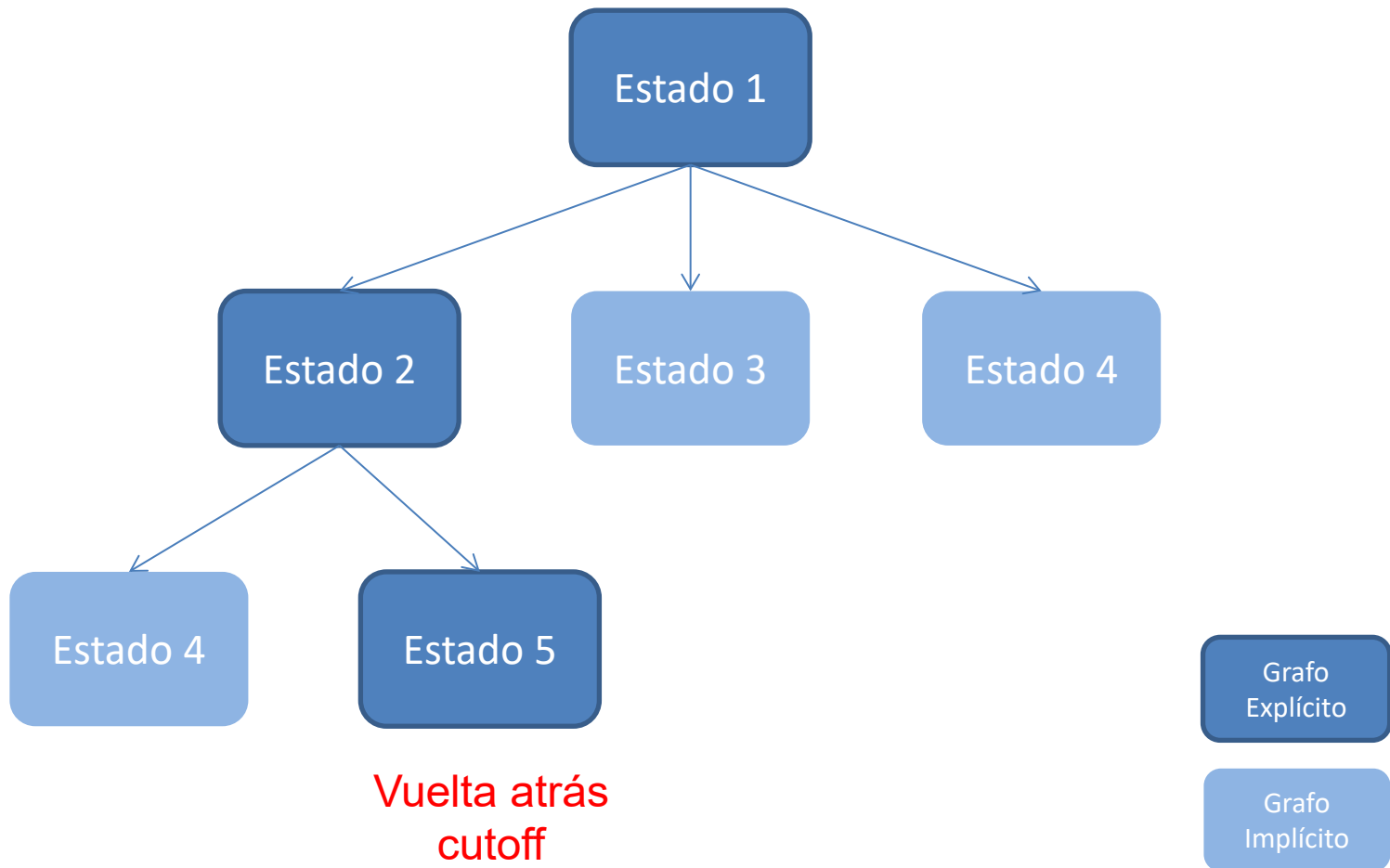


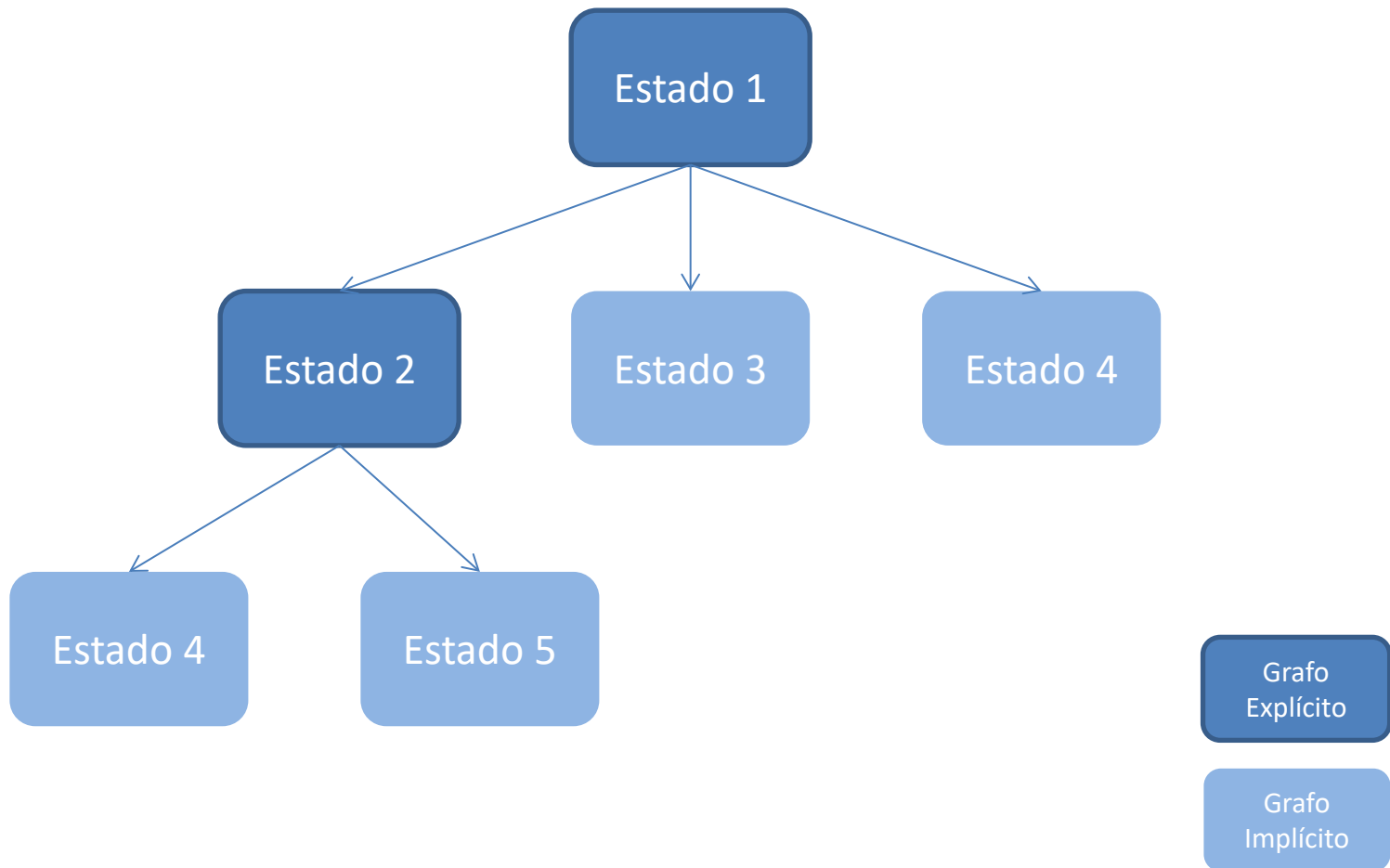


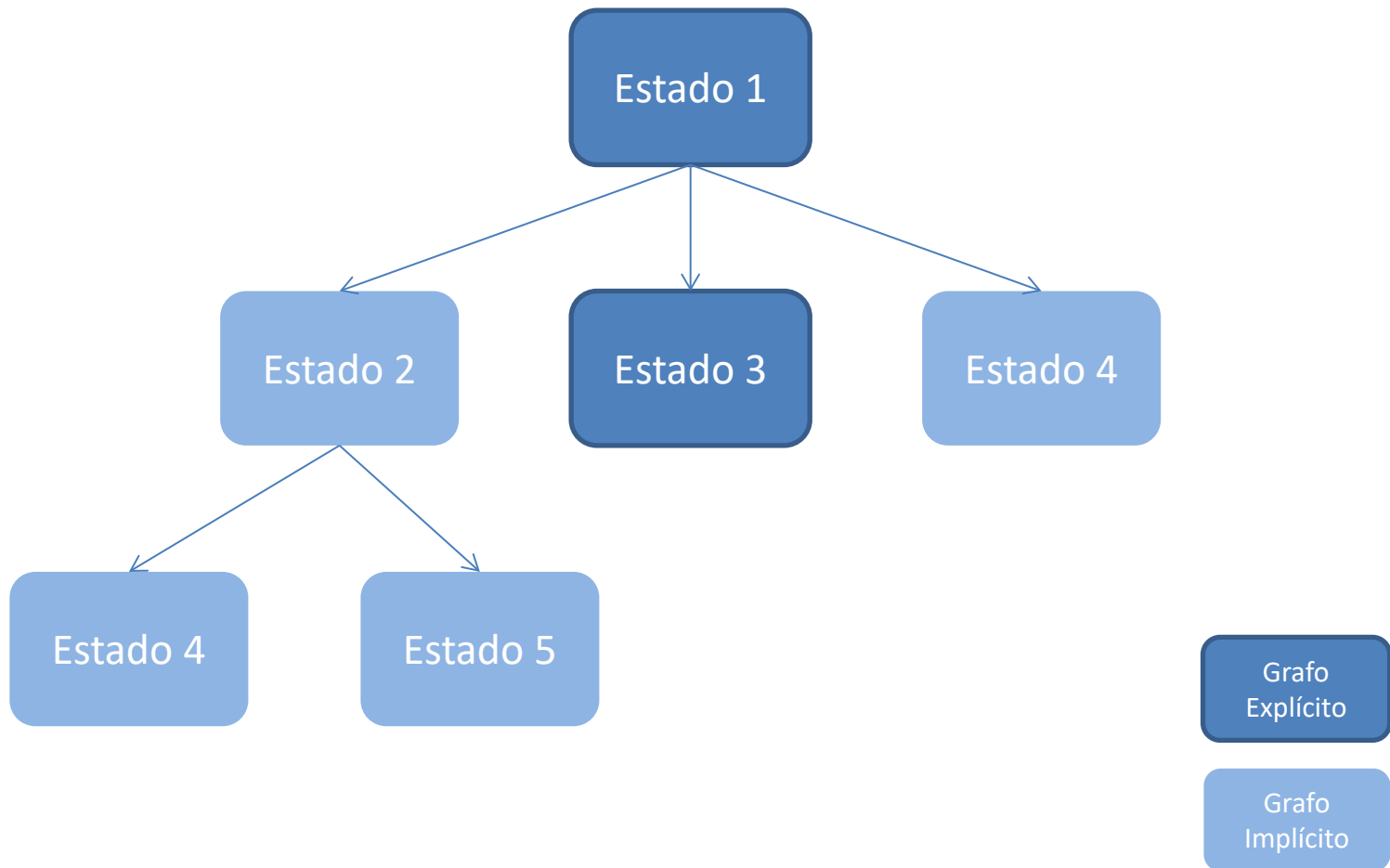


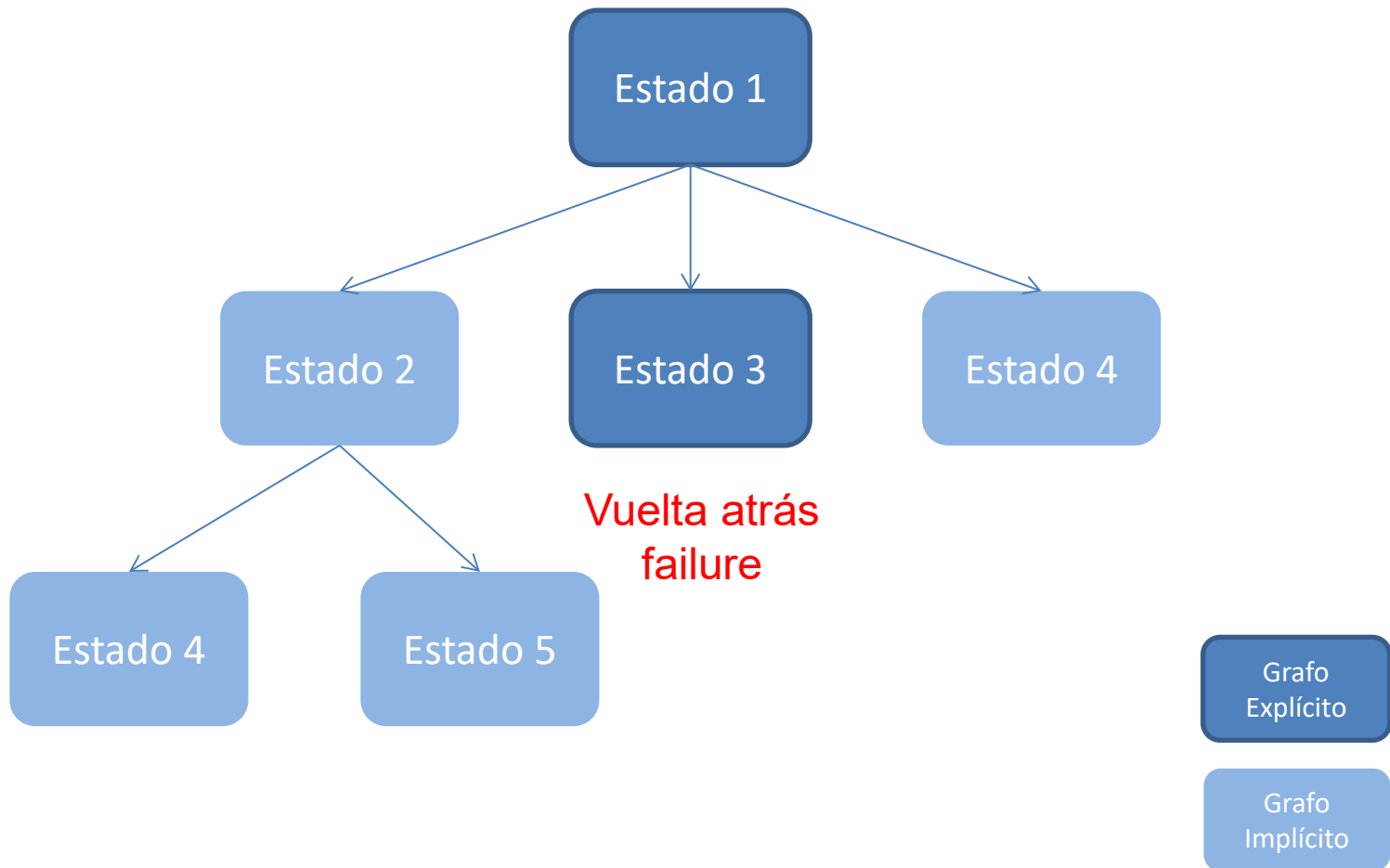








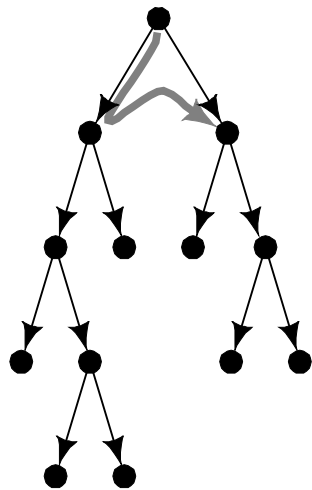




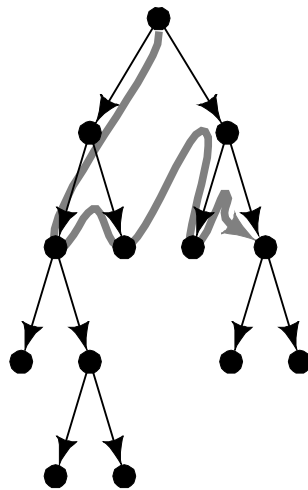
Características

- **Compleitud:** no asegura encontrar la solución
- **Optimalidad:** no asegura encontrar la solución óptima
- **Eficiencia:** bueno cuando las metas están alejadas del estado inicial, o hay problemas de memoria
- No es bueno cuando hay ciclos

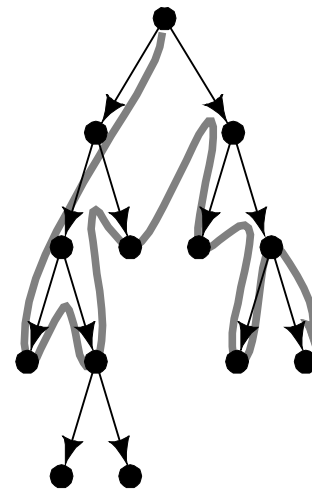
Descenso iterativo



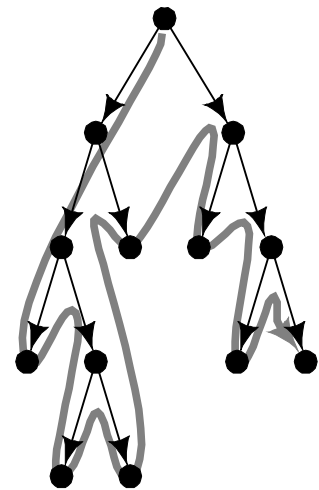
Depth bound = 1



Depth bound = 2



Depth bound = 3



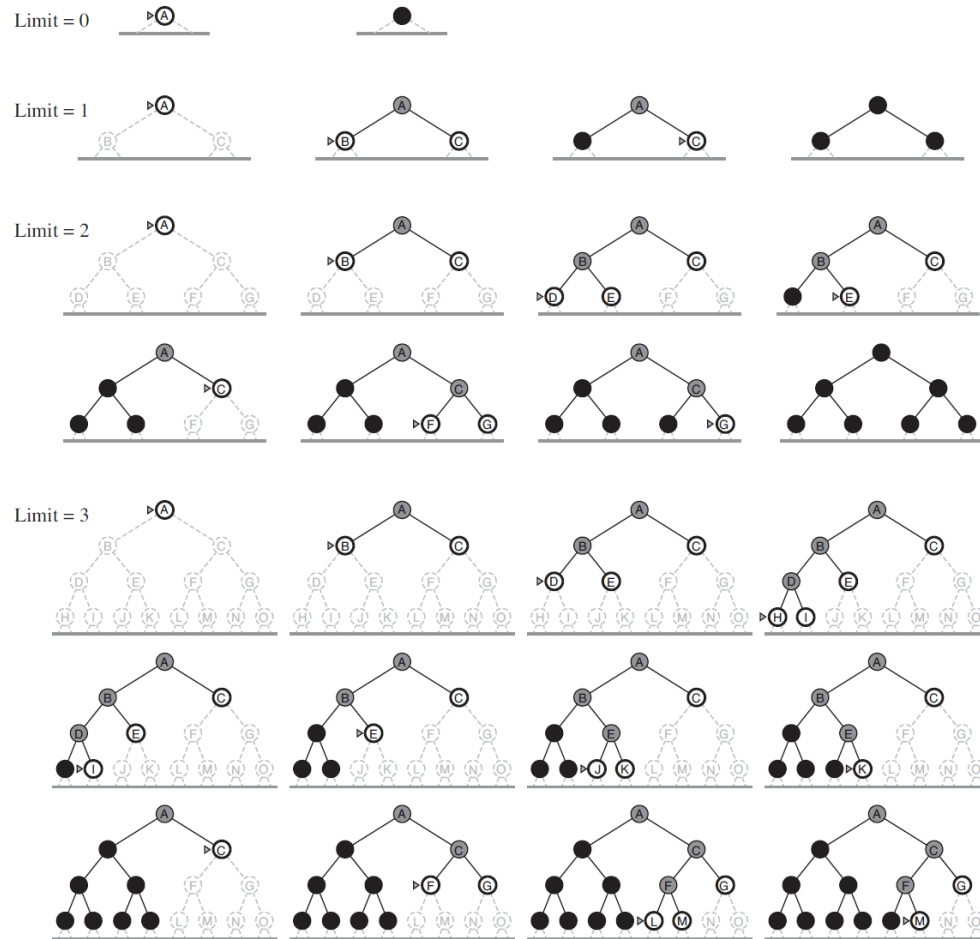
Depth bound = 4

© 1998 Morgan Kaufman Publishers

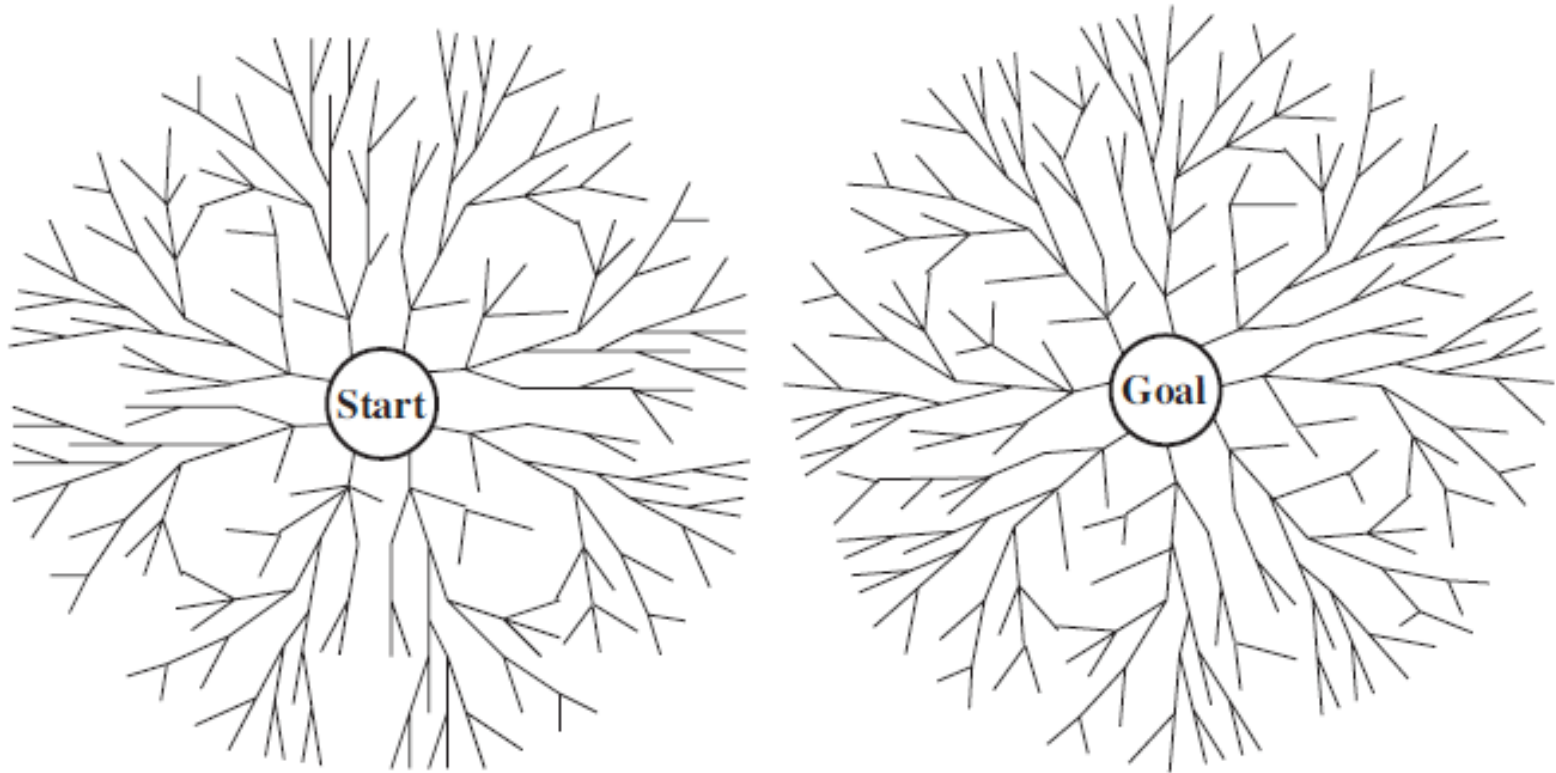
Descenso iterativo

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

Descenso iterativo



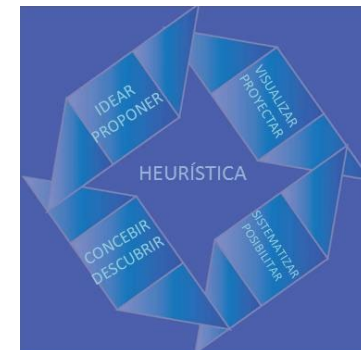
Búsqueda bidireccional



Búsqueda con información

- Heurísticas
- Métodos de escalada
- Búsqueda primero el mejor

SIMPLE
HEURISTICS
THAT MAKE US
SMART



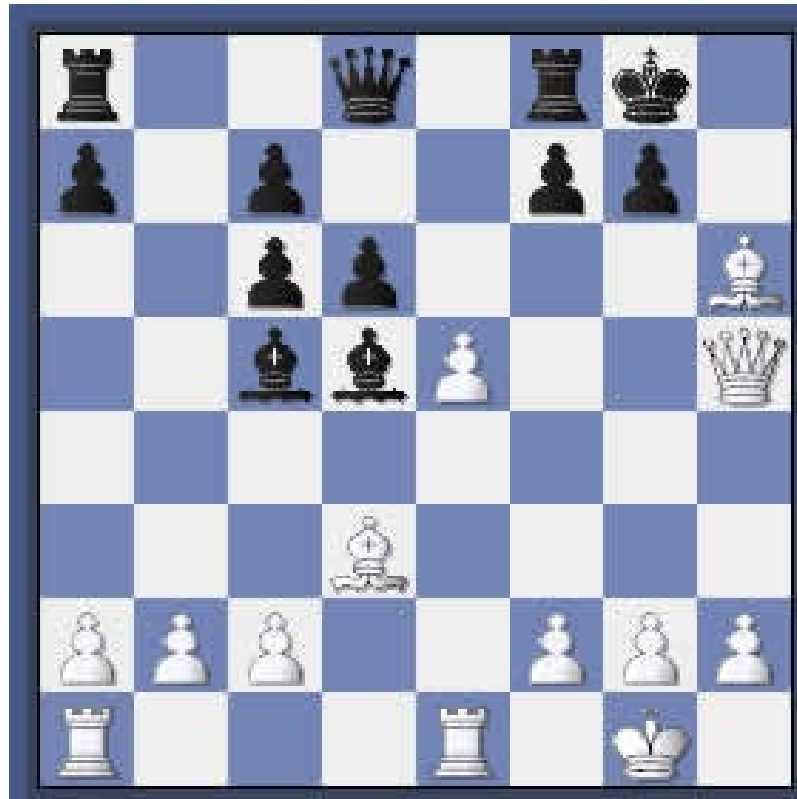
Heurísticas

- Si se tiene conocimiento perfecto : algoritmo exacto
- Si no se tiene conocimiento : búsqueda sin información
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego “heurisko” yo encuentro) conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio

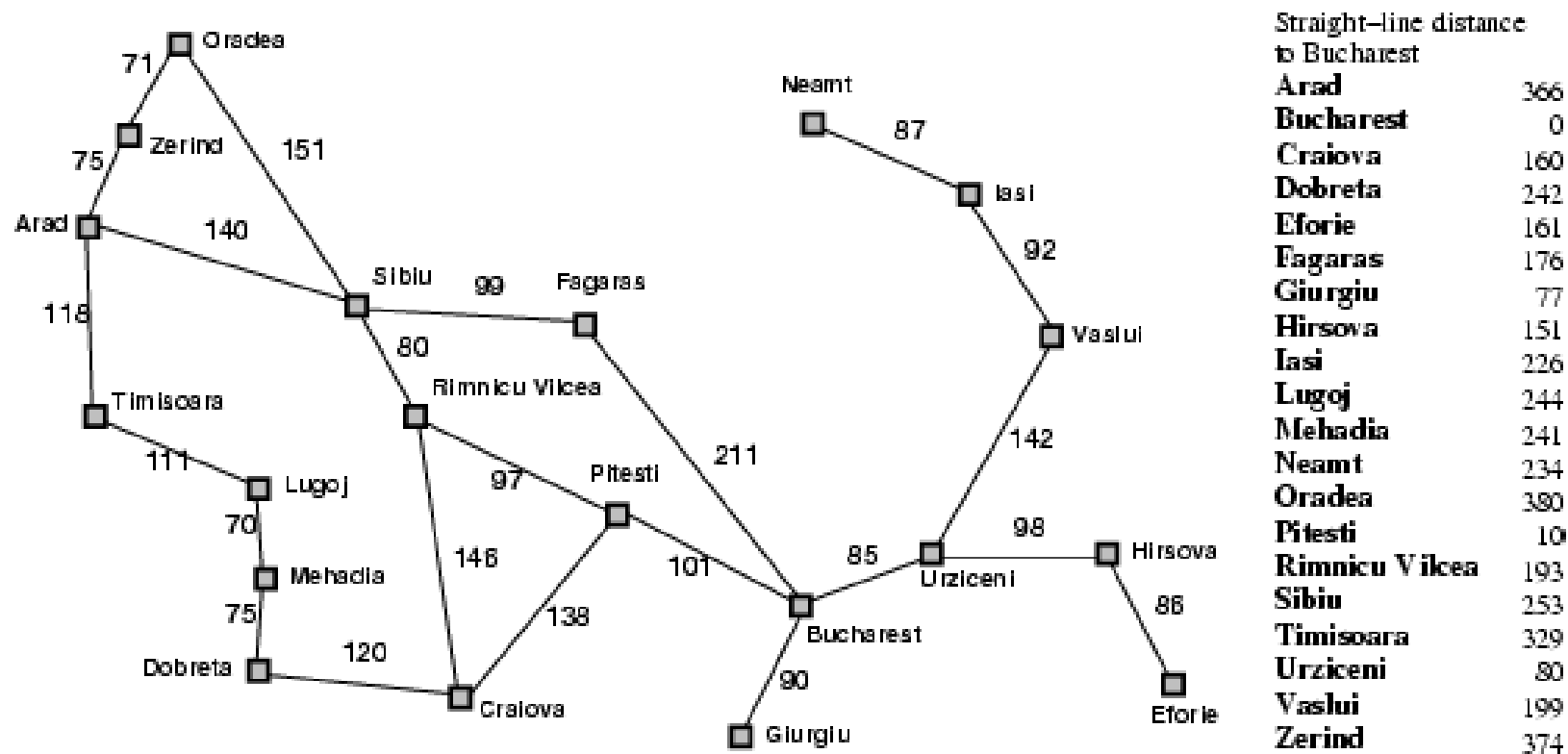
Heurísticas

- Las **heurísticas** son criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta
- En IA, entendemos por heurística un **método para resolver problemas** que en general **no garantiza la solución óptima**, pero que en media **produce resultados satisfactorios** en la resolución de un problema.
- Una heurística encapsula el conocimiento específico/experto que se tiene sobre un problema, y sirve de guía para que un algoritmo de búsqueda pueda encontrar una solución válida aceptable.
- Eventualmente, una heurística puede devolver siempre soluciones óptimas bajo ciertas condiciones (requiere demostración).

Ajedrez



Mapa de carreteras



8-puzzle

7	2	4
5		6
8	3	1

Start State

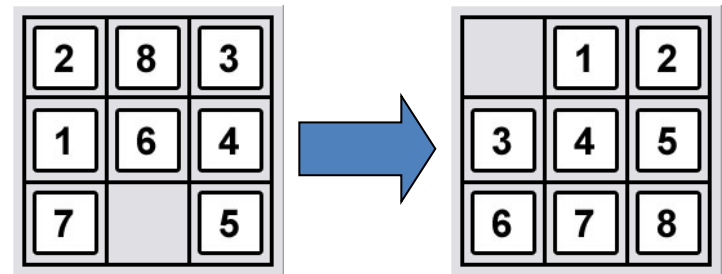
	1	2
3	4	5
6	7	8

Goal State

8-puzzle

- En IA, implementaremos heurísticas como funciones que devuelven un valor numérico, cuya maximización o minimización guiará al proceso de búsqueda a la solución.
- Ejemplo de heurística en el problema del 8-puzzle:
 - $f(n)$ = nº de fichas descolocadas en comparación con la posición objetivo a alcanzar.
 - **Objetivo:** Minimizar f .
 - n es un estado (posición de las piezas) del problema.
 - $f(n)$ es la *función heurística*.

$$f((2, 8, 3, 1, 6, 4, 7, 0, 5)) = 9$$

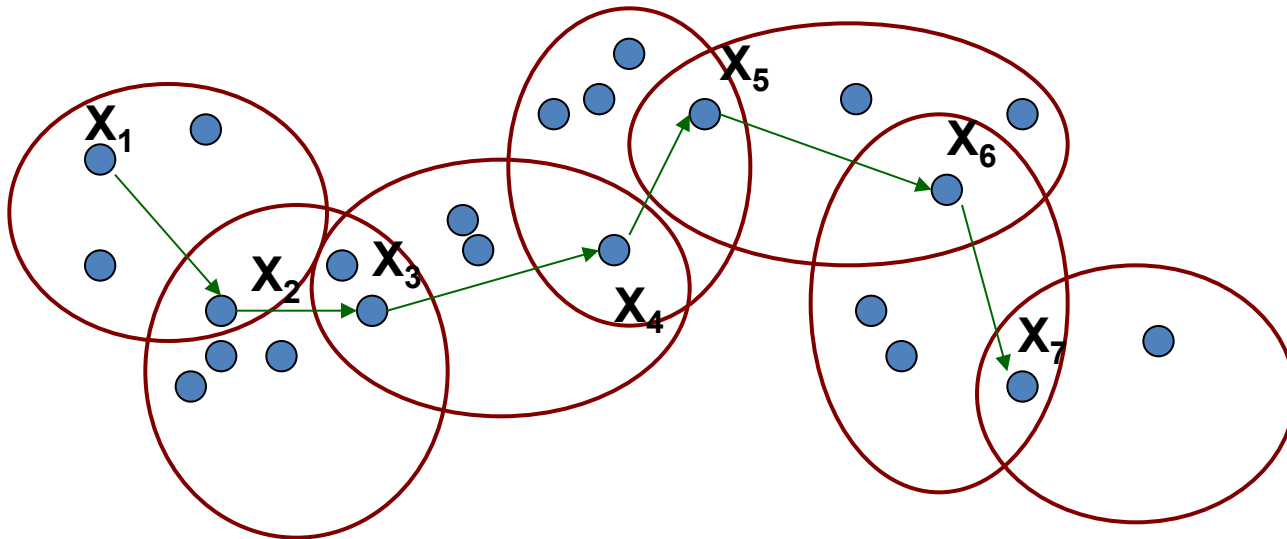


Métodos de escalada

- Algoritmo de escalada simple
- Algoritmo de escalada por la máxima pendiente
- Algunas variaciones estocásticas
- Algoritmos genéticos

Métodos de escalada

- Si dibujamos las soluciones como puntos en el espacio, una **búsqueda local** consiste en seleccionar la solución mejor en el vecindario de una solución inicial, e ir viajando por las soluciones del espacio hasta encontrar un óptimo (local o global).



Algoritmo de escalada simple

E: Estado activo

```
while (E no sea el objetivo
      y queden nodos por explorar a partir de E) {
    Seleccionar operador A para aplicarlo a E
    Evaluar  $f(A(E))$ 
    if ( $f(A(E)) > f(E)$ ) {
         $E = R(E)$ 
    }
}
```

Algoritmo de escalada por la máxima pendiente

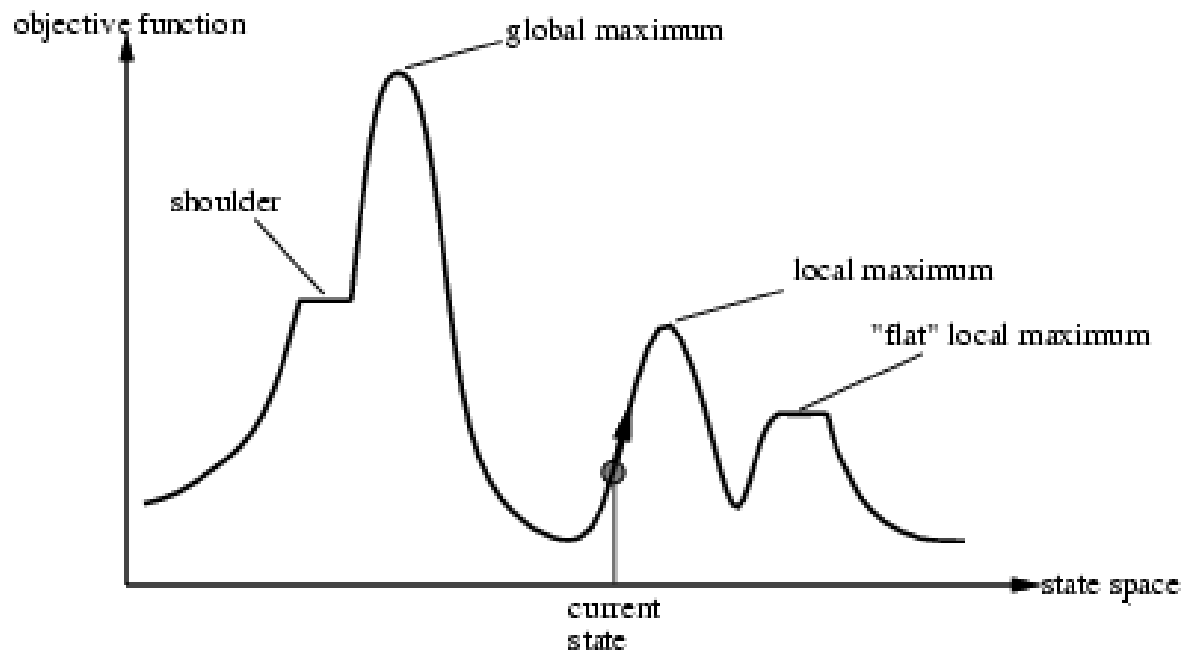
E: Estado activo

```
while (queden nodos por explorar a partir de E) {  
    Para todos los operadores  $A_i$ , obtener  $E_i = A_i(E)$   
    Evaluar  $f(E_i)$  para todos los estados  $E_i = A_i(E)$   
    Seleccionar  $E_{max}$  tal que  $f(E_{max}) = \max\{f(E_i)\}$   
    if ( $f(E_{max}) > f(E)$ ) {  
         $E = E_{max}$   
    } else return E  
}
```

Características

- **Completitud:** no tiene porque encontrar la solución
- **Admisibilidad:** no siendo completo, aun menos será admisible
- **Eficiencia:** rápido y útil si la función es monótona (de)creciente

Métodos de escalada



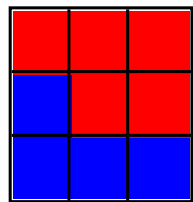
Ejemplo

- **Ejemplo:** Colorear una matriz de **filas*columnas** con **n** colores, de modo que cada celda tenga el mínimo número de celdas adyacentes del mismo color. Asumimos que las celdas adyacentes son las que se encuentran una casilla hacia arriba, abajo, izquierda o derecha de la casilla considerada.
 - **Función objetivo:** Minimizar la suma del número de pares de casillas adyacentes del mismo color.
 - **Definición del entorno:** El vecindario de una solución estará formado por aquellas soluciones cuyos colores varíen en una única posición de la solución dada.

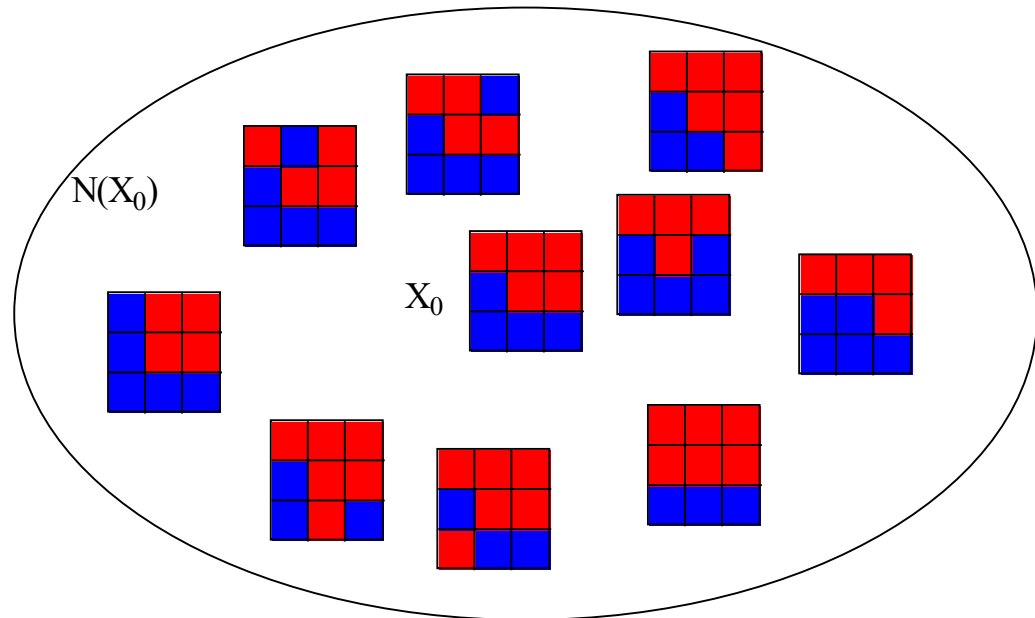
Ejemplo

- **Ejemplo:**

- **Solución inicial:** Generada de forma aleatoria. Supongamos que se ha generado la siguiente para una matriz de 3×3 , a rellenar con **2 colores rojo y azul**, con valor de función objetivo $v=8$:

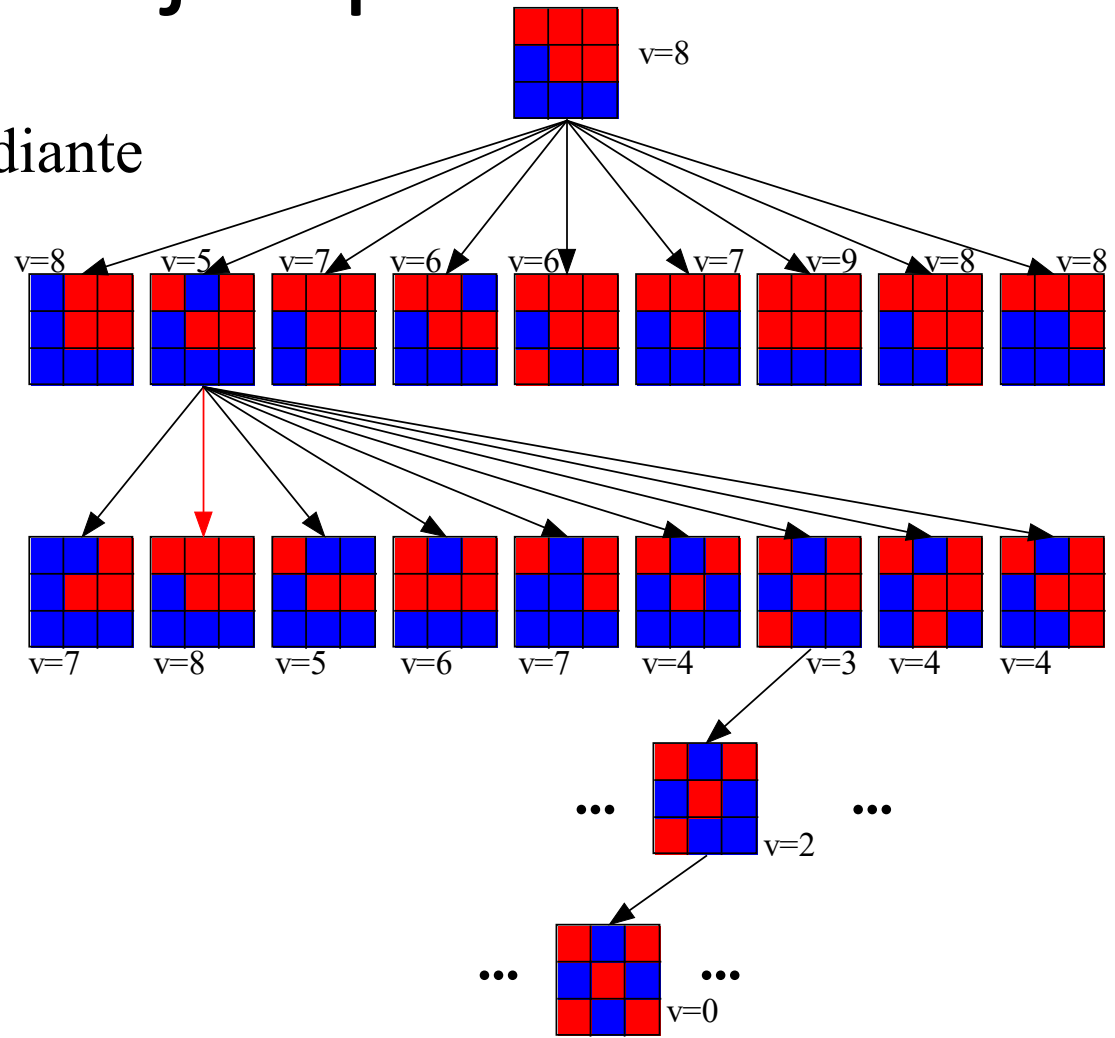


$v=8$



Ejemplo

- **Ejemplo:** Solución al problema anterior mediante ascensión de colinas.

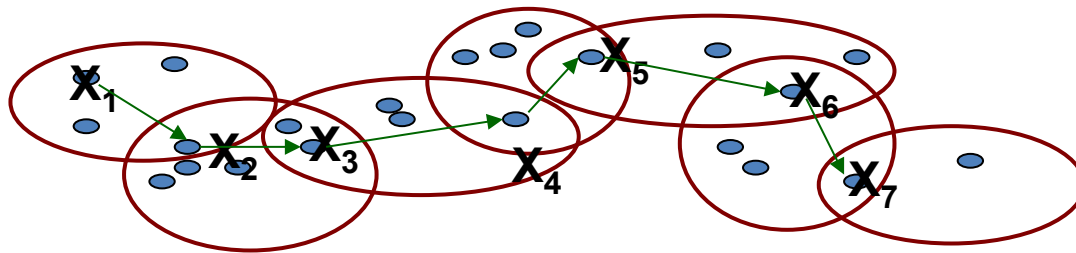


Algunas variaciones estocásticas

- Algoritmo de escalada estocástico
- Algoritmo de escalada de primera opción
- Algoritmo de escalada de reinicio aleatorio
- Enfriamiento simulado

Algoritmo de enfriamiento simulado

- Es un método de búsqueda local.
- Se basa en principios de Termodinámica.
- Al contrario que otros métodos de ascensión de colinas, permite visitar soluciones peores que la actual para evitar óptimos locales.



Algoritmo de enfriamiento simulado

- **Un poco de historia:**

- En el campo de la Termodinámica, en los años 50 se simuló el proceso de enfriamiento en sistemas de partículas hasta que se llegaba a un estado estable.
- El proceso simulaba la diferencia de energía del sistema, δE , y se quería verificar que la probabilidad de que el sistema tuviese el cambio δE seguía la siguiente fórmula (t es la temperatura actual del sistema; k es una constante física):

$$P[\delta E] = e^{-\frac{\delta E}{k \cdot T}}$$

Algoritmo de enfriamiento simulado

- **Analogía entre el proceso de enfriamiento y el algoritmo de enfriamiento simulado:**
 - Los **estados** por los que pasa el sistema físico de partículas equivalen a las **soluciones factibles** del algoritmo.
 - La **energía E del estado actual** del sistema es el valor de la **función objetivo de la solución actual**. Ambos tienen que minimizarse.
 - Un **cambio de estado** en el sistema equivale a **explorar el entorno de una solución y viajar a una solución vecina**.
 - El **estado final estable** (congelado) es la **solución final** del algoritmo.

Algoritmo de enfriamiento simulado

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow \text{schedule}(t)$

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow \text{next.VALUE} - \text{current.VALUE}$

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Algoritmo de enfriamiento simulado

- La **solución inicial** se puede generar de forma aleatoria, por conocimiento experto, o por medio de otras técnicas algorítmicas como *greedy*.
- La **actualización de temperatura** también es heurística, y hay varios métodos:
 - $T \leftarrow \alpha \cdot T$, con α en $(0,1)$
 - $T \leftarrow 1/(1+k)$, con k = número de iteraciones del algoritmo hasta el momento, etc.
- **Número de vecinos a generar:** Fijo $N(T) = \text{cte}$, dependiente de la temperatura $N(T) = f(T)$, etc.

Algoritmo de enfriamiento simulado

- Tanto la temperatura inicial como la temperatura final T_i y T_f son parámetros de entrada al algoritmo.
- Es difícil asignar un valor concreto a T_f , por lo que la condición de parada se suele sustituir por un número específico de iteraciones a realizar.

• **Ventajas:**

- Al ser un método probabilístico, tiene capacidad para salir de óptimos locales.
- Es eficiente.
- Es fácil de implementar.

Algoritmo de enfriamiento simulado

- **Inconvenientes:**

- Encontrar la temperatura inicial T_i , el método de actualización de temperatura α , el número de vecinos a generar en cada estado y el número de iteraciones óptimo es una tarea que requiere de **muchas pruebas de ensayo y error** hasta que ajustamos los parámetros óptimos.
- Pese a todo, el algoritmo puede proporcionar soluciones mucho mejores que utilizando algoritmos no probabilísticos.

Algoritmos genéticos

- La simulación de procesos naturales es un campo de investigación muy amplio en Inteligencia Artificial.
- Ejemplos son la **computación evolutiva**, **biocomputación**, **algoritmos bioinspirados**, etc.
- Si ha funcionado bien en la naturaleza, ¿porqué una simulación de estos procesos no iba a proporcionar buenos resultados en un computador?
- Ejemplos:
 - Algoritmos genéticos.
 - Algoritmos basados en Colonias de Hormigas.
 - Algoritmos basados en inteligencia de enjambres.

Algoritmos genéticos

- Son algoritmos de optimización basados en el proceso de la evolución natural de Darwin.
- En un proceso de evolución, existe una **población de individuos**. Los más adecuados a su entorno **se reproducen y tienen descendencia** (a veces **con mutaciones** que mejoran su idoneidad al entorno). Los más adecuados sobreviven para la siguiente generación.
- No necesitan partir de un nodo/estado inicial: ¡Hay toda una población!



- Su objetivo es encontrar una solución cuyo valor de función objetivo sea óptimo.

Algoritmos genéticos

- **Cromosoma** \leftrightarrow Vector representación de una solución al problema.
- **Gen** \leftrightarrow Característica/Variable/Atributo concreto del vector de representación de una solución
- **Población** \leftrightarrow Conjunto de soluciones al problema.
- **Adecuación al entorno** \leftrightarrow Valor de función objetivo (fitness).
- **Selección natural** \leftrightarrow Operador de selección.
- **Reproducción sexual** \leftrightarrow Operador de cruce.
- **Mutación** \leftrightarrow Operador de mutación.
- **Cambio generacional** \leftrightarrow Operador de reemplazamiento.

Algoritmos genéticos

- **Ejemplo:** Cromosoma que codifica una solución a un problema. Cada característica del problema es un valor 0/1.

Individuo (Solución)



Cromosoma
(Representación del individuo)

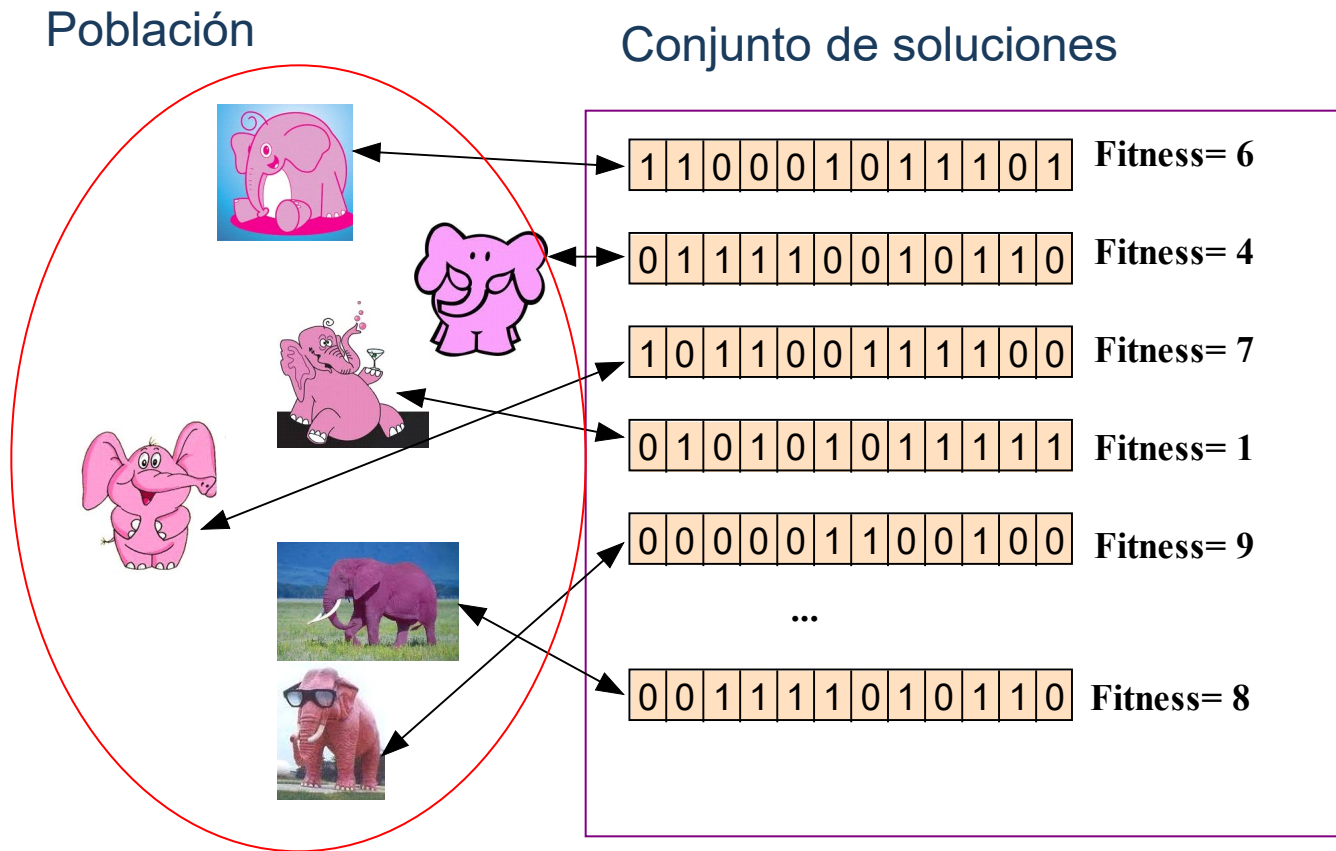
1	1	0	0	0	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---

...

Genes (codificación binaria)

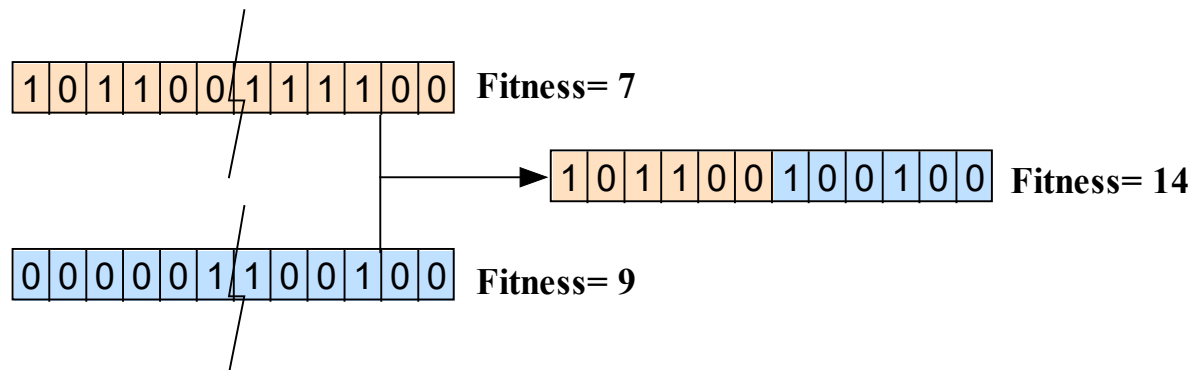
Algoritmos genéticos

- **Ejemplo:** Población. Conjunto de individuos (cada uno con su fitness).



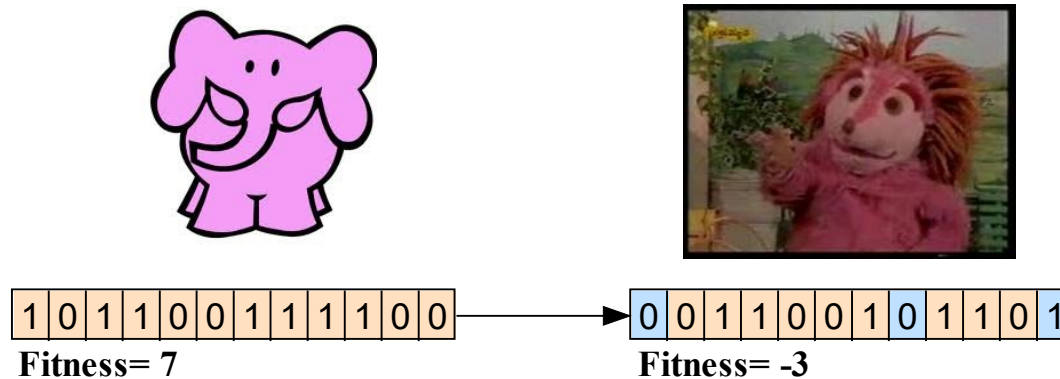
Algoritmos genéticos

- **Ejemplo:** Cruce. Combinación de soluciones de la población para generar descendientes.



Algoritmos genéticos

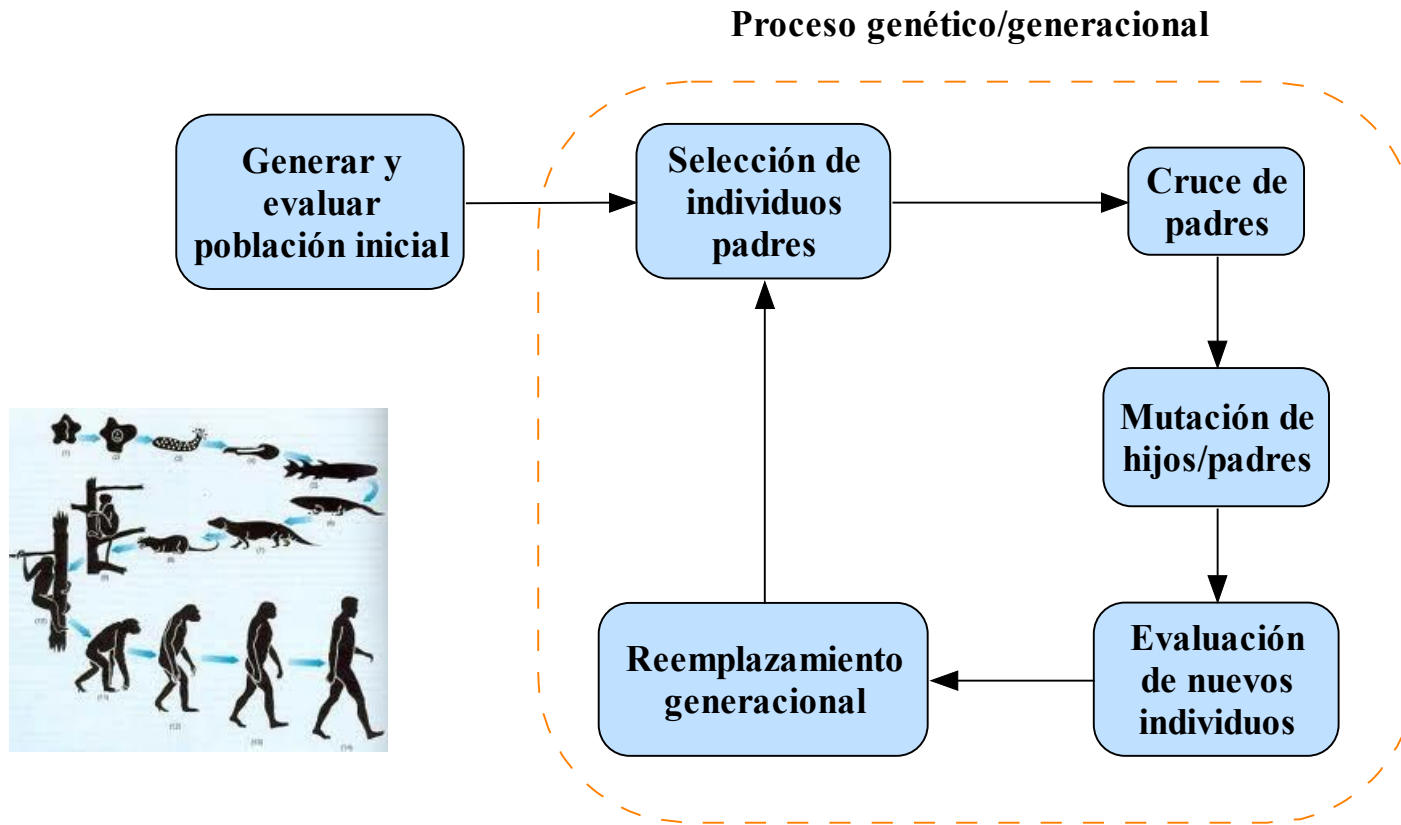
- **Ejemplo:** Mutación. Uno o más genes de un individuo pueden mutar para generar una nueva solución.



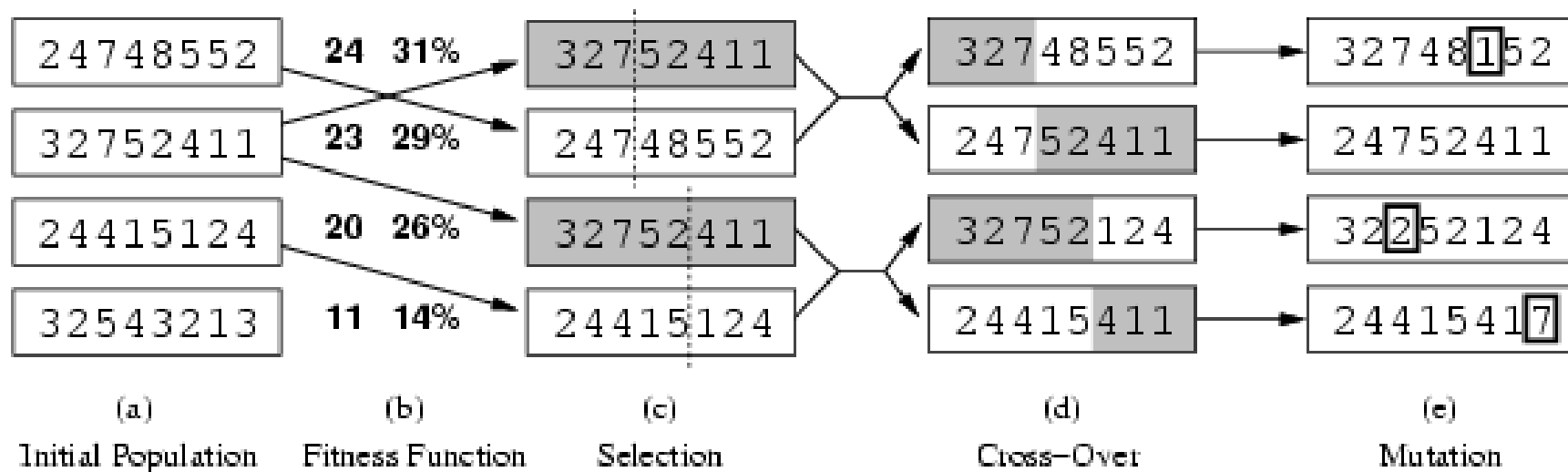
- En la población, hay una probabilidad dada a priori de que un individuo pueda mutar. A su vez, cuando un individuo muta, existe otra probabilidad de que cada gen mute o no.

Algoritmos genéticos

- Proceso de un algoritmo genético:



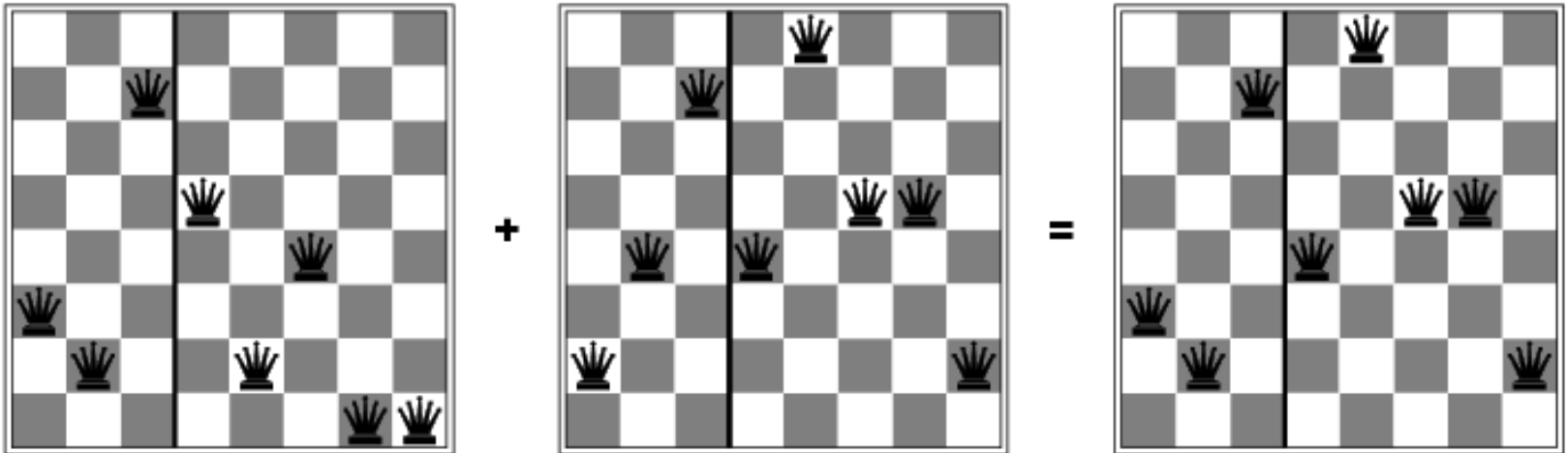
Ejemplo



Función de evaluación (8 reinas) = número de pares de reinas no atacadas

28 para una solución

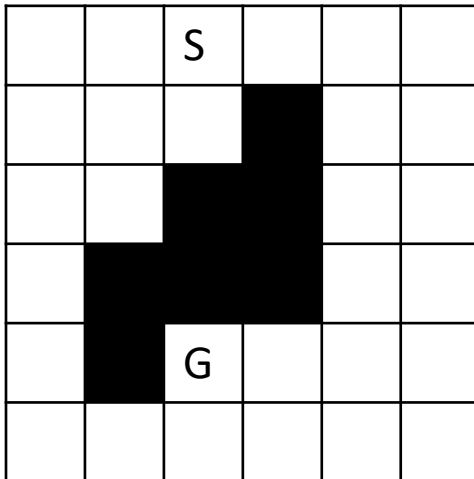
Ejemplo



Búsqueda primero el mejor

- Búsqueda primero mejor greedy (BFS)
- Algoritmo A*
- Búsqueda dirigida

Ejemplo motivador



- Solo movimiento horizontal y vertical.
- Estimación: distancia manhattan.
- Suma de la distancia vertical y horizontal medida en celdas

Calculemos el valor heurístico

		S	5	6	
	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- Solo movimiento horizontal y vertical.
- $h(n) = d(n, G)$
- Pensemos: ¿Cuál es el mejor camino?

Pensemos en escalada

		S	5	6	
	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- Solo movimiento horizontal y vertical.
- $h(n) = d(n, G)$
- ¿Cómo funcionaría escalda?

Pensemos en BFS

		S	5	6	
	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- Solo movimiento horizontal y vertical.
- $h(n) = d(n, G)$
- ¿Cómo funcionaría BFS greedy?

¿No es óptimo?

		S	5	6	
	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

- Solo movimiento horizontal y vertical.
- $h(n) = d(n, G)$
- ¿Cómo funcionaría BFS greedy?
- ¿Por qué no encuentra el camino óptimo?

A* puede encontrar óptimo

		S	5	6	
	4	3		5	
4	3			4	
3				3	
2		G	1	2	
3	2	1			

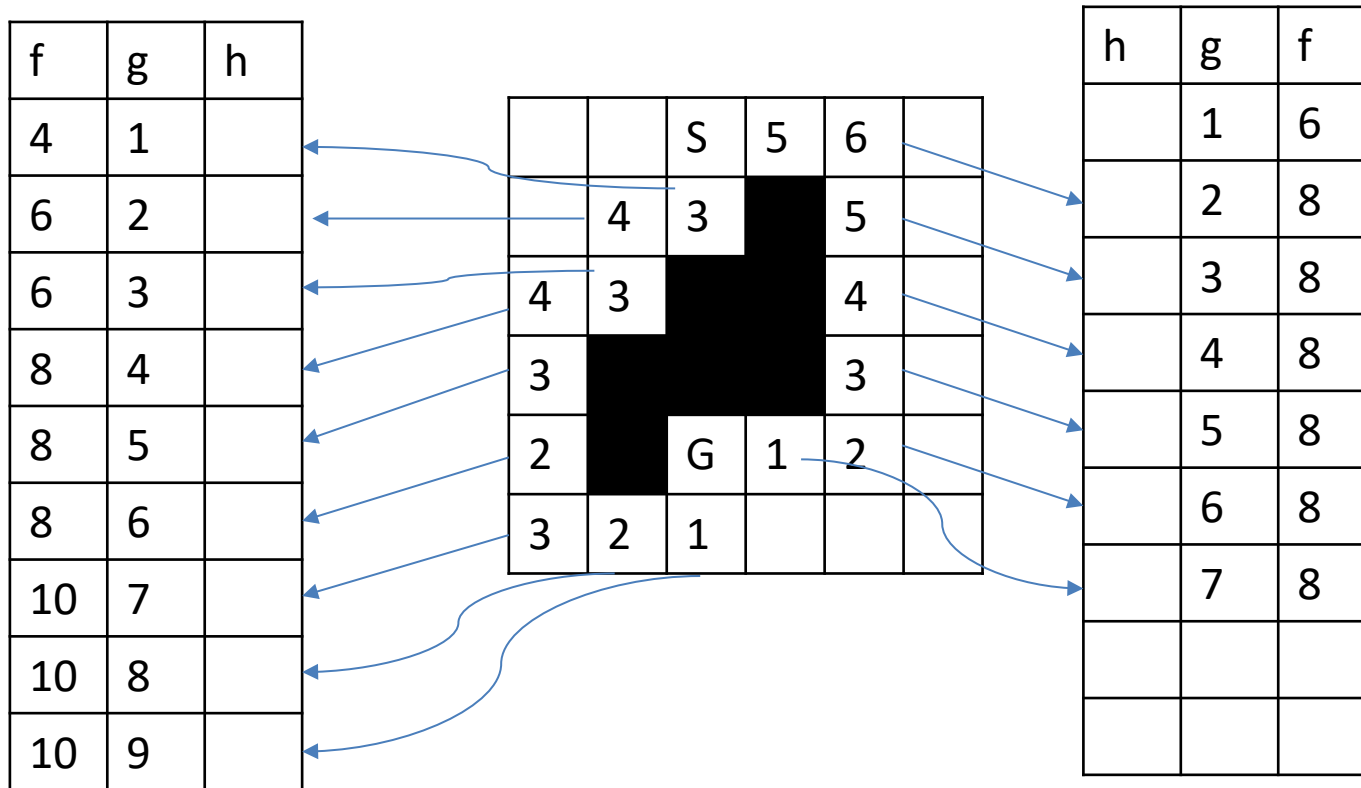
- Solo movimiento horizontal y vertical.
- $h(n) = d(n, G)$
- Para cada nodo n , A* tiene en cuenta la **estimación, $h(n)$** , al objetivo y el **coste real, $g(n)$** desde el inicio
- $f(n) = g(n) + h(n)$

Primera Aproximación a A*

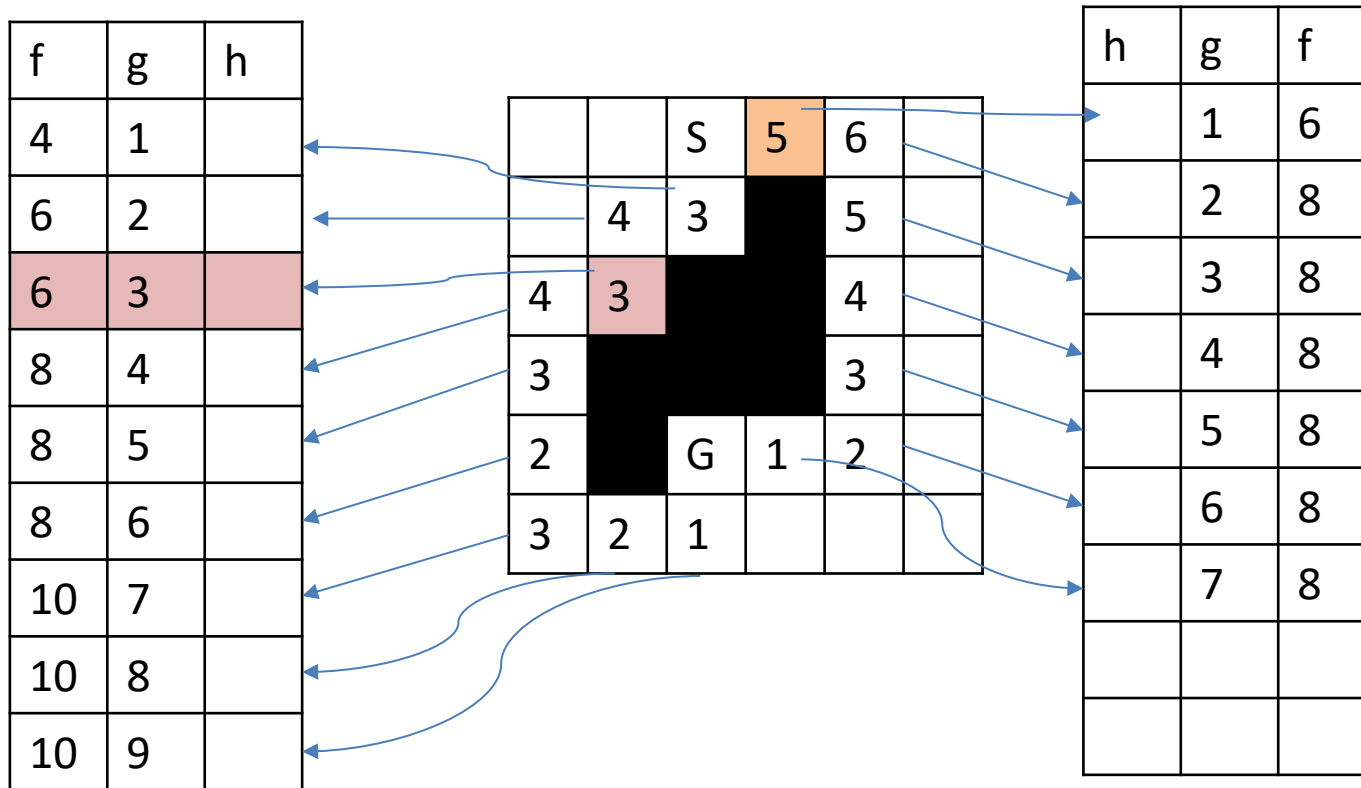
- Búsqueda en grafos donde abiertos es una cola con prioridad ordenada de acuerdo a $f(n)$.
- Recordar qué hay en un nodo:

estado	hijos	mejor_padre	g
			h





Sigamos la traza del algoritmo



Observar



Observar

- A* detecta que el camino inicial no tiene por qué ser el mejor cuando el nodo  genera su hijo y éste es peor que el nodo 
- Toda la descendencia de  no empeora (empatan) respecto al hijo de 
- Usamos como criterio de desempate el nodo más joven.

Algoritmo A*

- ABIERTOS contiene el nodo inicial, CERRADOS esta vacío
- Comienza un ciclo que se repite hasta que se encuentra solución o hasta que ABIERTOS queda vacío
 - Seleccionar el mejor nodo de ABIERTOS
 - Si es un nodo objetivo terminar
 - En otro caso se expande dicho nodo
 - Para cada uno de los nodos sucesores
 - Si está en ABIERTOS insertarlo **manteniendo la información del mejor padre**
 - Si está en CERRADOS insertarlo **manteniendo la información del mejor padre y actualizar la información de los descendientes**
 - En otro caso, insertarlo como un nodo nuevo

Nodos repetidos en abiertos

- Si un nodo n ya existe en abiertos hay que comprobar si el nuevo camino es mejor que el anterior.

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

El número indica el orden de generación de los nodos

Nodos repetidos en abiertos

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

El número indica el orden de generación de los nodos

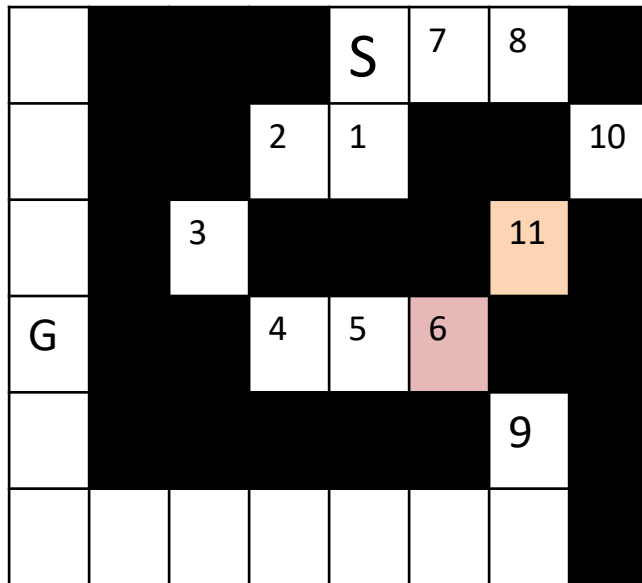
				S	9	11	13
		7	7	7			13
		7				13	13
G		7	9	11	13	15	

El número indica el valor de h

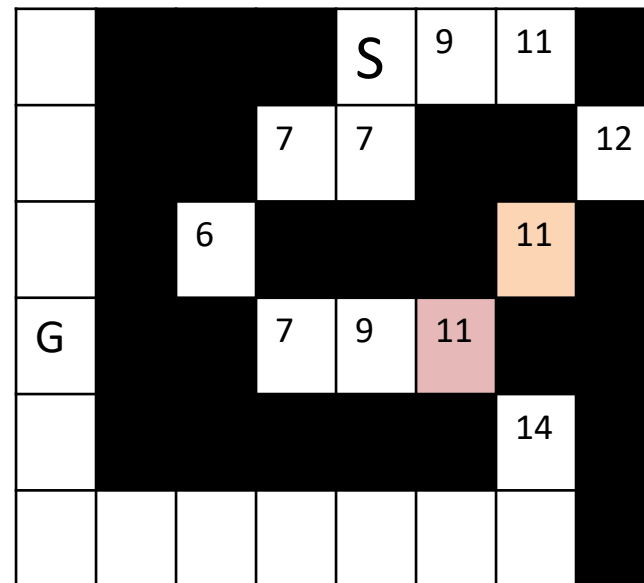
- Trazar a mano el proceso de A* y detenerse en generación 15.
- Observar:
 - en la generación 12, ¿Cuánto vale g y h?
 - en la generación 16, ¿Cuánto vale g y h?

Nodos repetidos en cerrados

- Si un nodo n ya existe en cerrados hay que hacer más cosas.
- **Vamos a asumir que se permiten movimientos en diagonal**
- **$h(n)$ = distancia manhattan**



El número indica el orden de generación de los nodos



El número indica el valor de h

- Hay que actualizar el padre de al nodo y hacerlo recursivamente por cada descendiente de en cerrados.
- **En la recursión pueden darse los siguientes casos.**

Nodos repetidos en cerrados

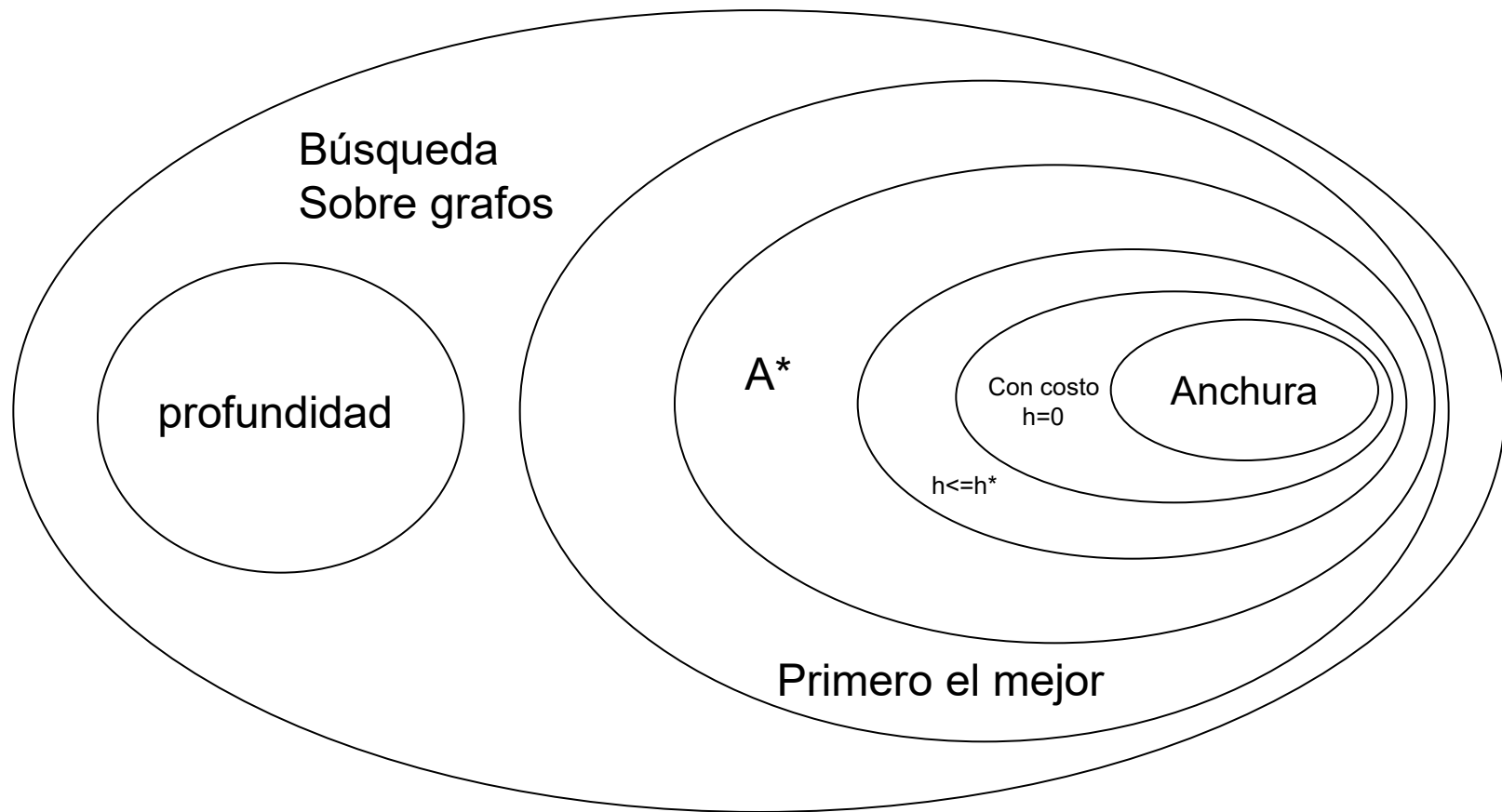
PROCESO RECURSIVO para un nodo n = un nodo en cerrados.

- $p(n)$ padre anterior de n
- $p_c(n)$ padre actual
- caso 1: $p_c(n)$ **NO ES MEJOR QUE** $p(n)$. FIN
- caso 2: $p_c(n)$ **ES MEJOR QUE** $p(n)$
 - actualizar $g(n)$ al coste actual del camino
 - actualizar el padre de n a $p_c(n)$. SEGUIR RECURSIÓN PARA CADA UNO DE SUS HIJOS QUE ESTÉ EN CERRADOS
- Si n tiene sucesor en ABIERTOS.
 - Si el coste actual del camino hasta el sucesor mejora el coste anterior del sucesor
 - actualizar el coste del sucesor
 - actualizar el padre del sucesor.
 - FIN.

Características

- **Completitud:** si existe solución, la encuentra
- **Admisibilidad:** si hay una solución óptima, la encuentra si
 - el numero de sucesores es finito para cada nodo,
 - $c(n_i, n_j) > \delta > 0$ en cada arco, y
 - La función $h(n)$ es admisible: $h(n) \leq h^*(n)$

Algoritmos de búsqueda



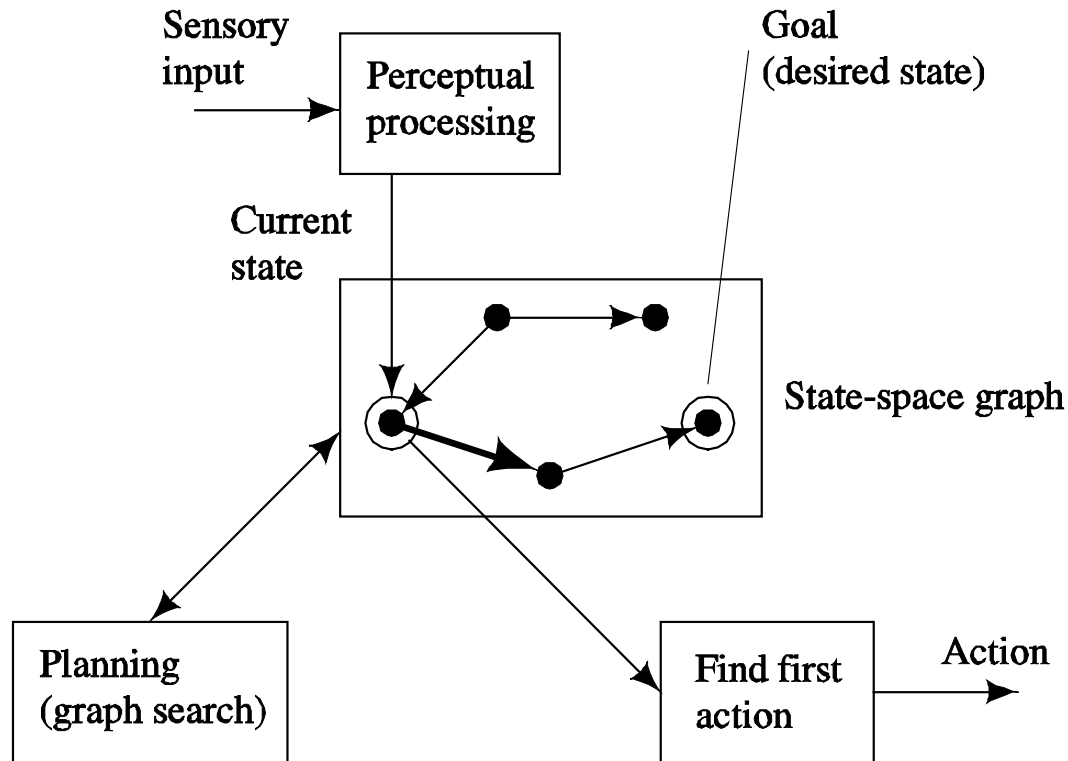
Dificultades del proceso

- Los procesos de percepción no siempre pueden obtener la información necesaria acerca del **estado** del entorno
- Las acciones pueden no disponer siempre de modelos de sus **efectos**
- Pueden haber otros procesos físicos, u **otros agentes**, en el mundo

Dificultades del proceso

- En el tiempo que transcurre desde la construcción de un plan, el mundo puede cambiar de tal manera que el plan ya no sea adecuado
- Podría suceder que se le requiriese al agente actuar antes de que pudiese completar una búsqueda de un estado objetivo
- Aunque el agente dispusiera de tiempo suficiente, sus recursos de memoria podrían no permitirle realizar la búsqueda de un estado objetivo

Arquitectura percepción/planificación/actuación

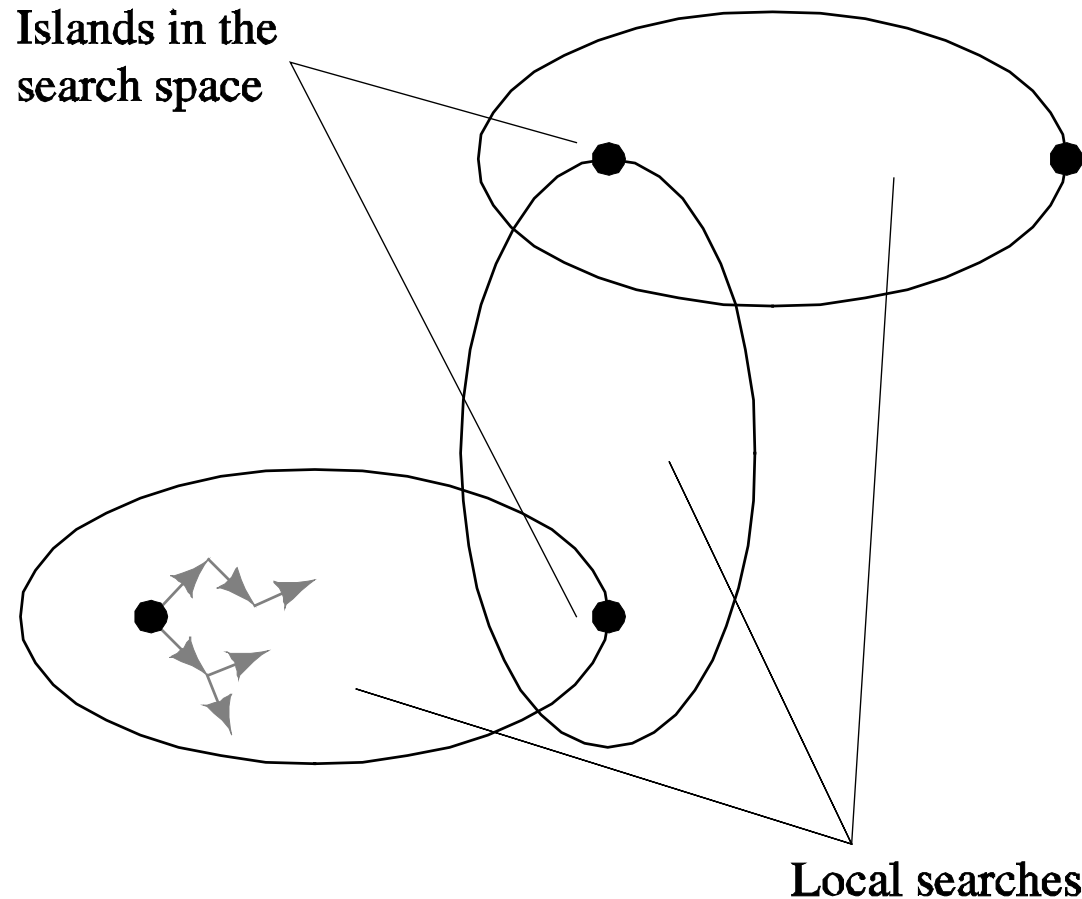


© 1998 Morgan Kaufman Publishers

Heurísticas sobre el proceso de búsqueda

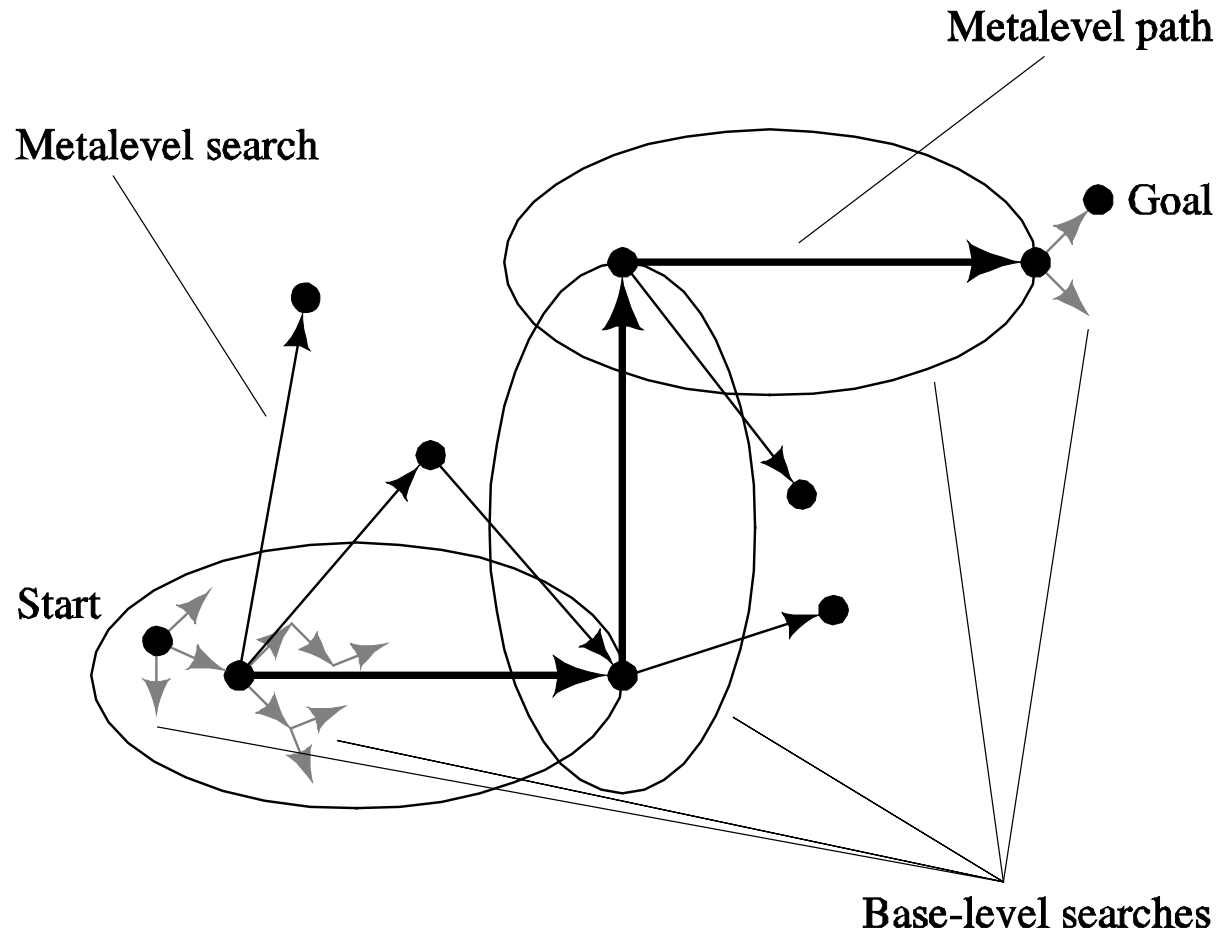
- Búsqueda orientada a subobjetivos
- Búsqueda con horizonte
- Búsqueda jerárquica

Búsqueda orientada a subobjetivos



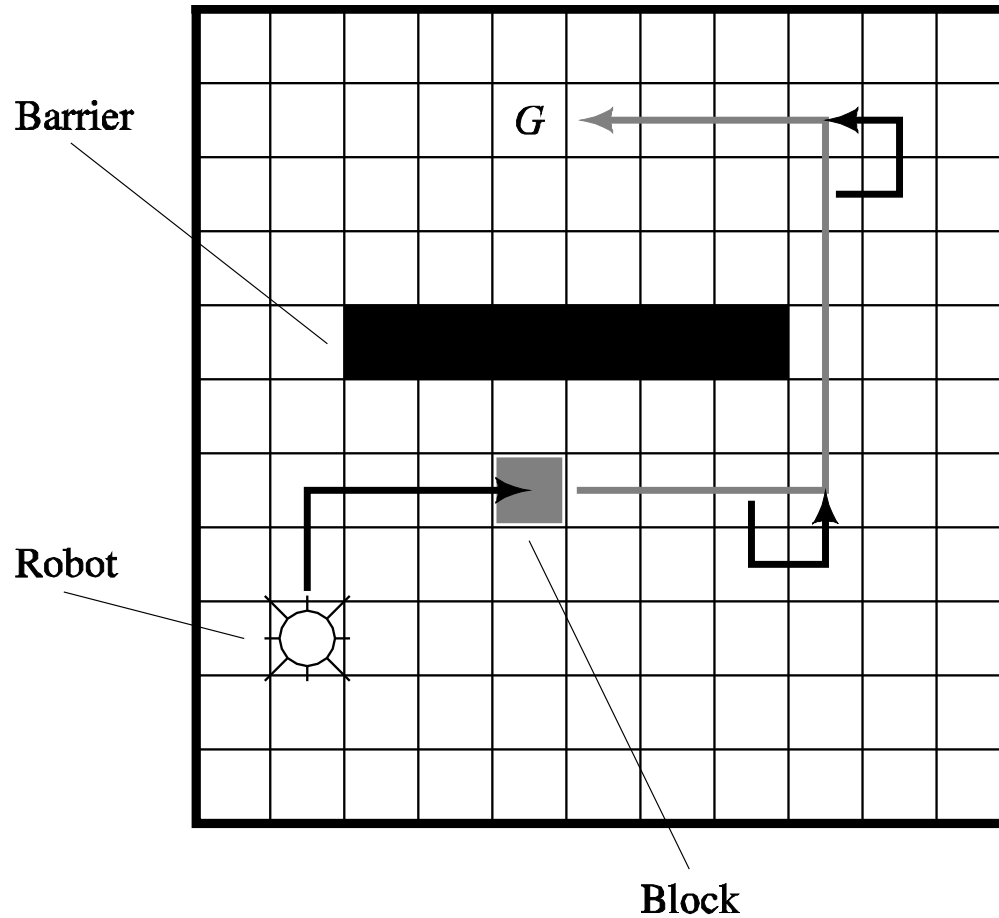
© 1998 Morgan Kaufman Publishers

Búsqueda jerárquica



© 1998 Morgan Kaufman Publishers

Búsqueda jerárquica



© 1998 Morgan Kaufman Publishers

Problemas descomponibles

- Base de datos inicial (C,B,Z)
- Operadores

R1: $C \longrightarrow (D,L)$

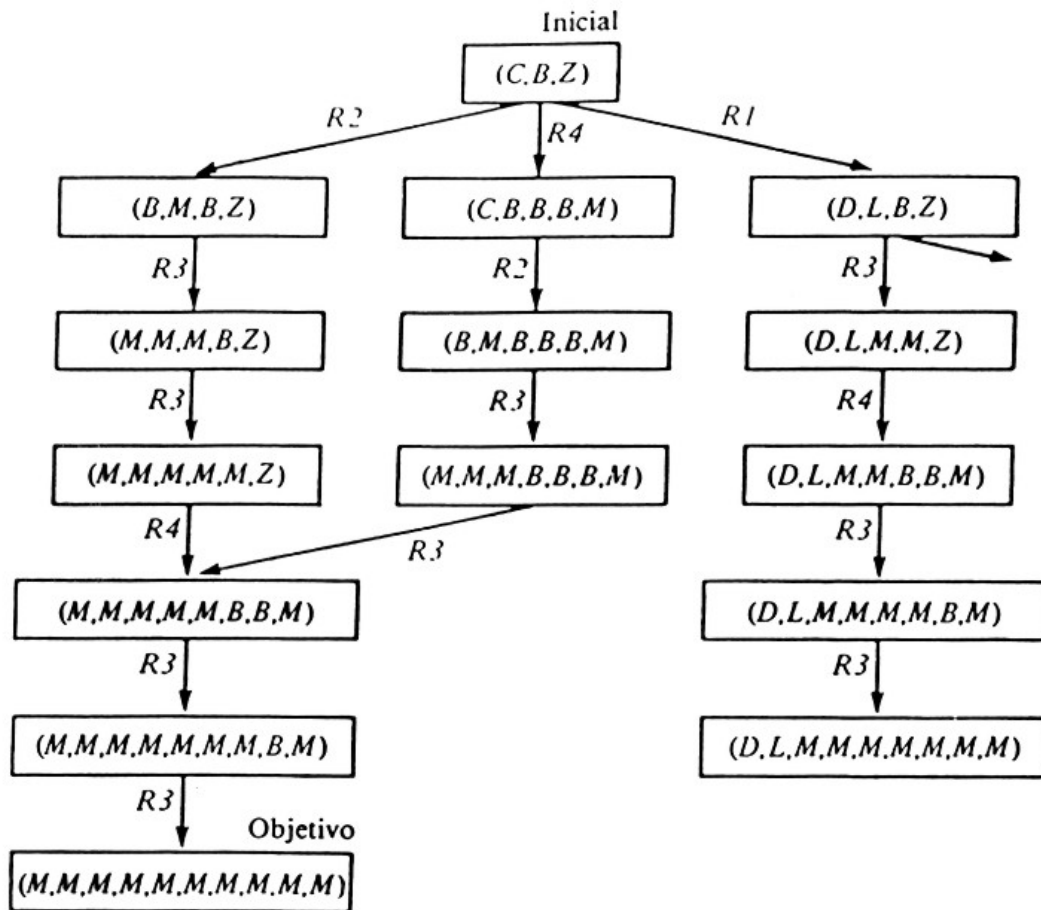
R2: $C \longrightarrow (B,M)$

R3: $B \longrightarrow (M,M)$

R4: $Z \longrightarrow (B,B,M)$

- Objetivo

Resolución del problema

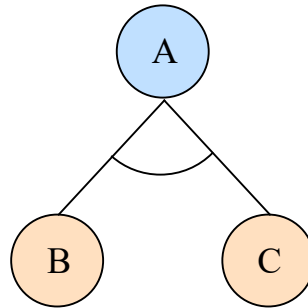


Grafo Y/O

- Descomposición de problemas: arcos Y
- Resolución de problemas: arcos O
- Concepto de solución: subgrafo solución

Grafo Y/O

- **Grafo Y:** Para completar el objetivo/tarea **A**, es necesario terminar antes los objetivos/tareas **B** y **C**.

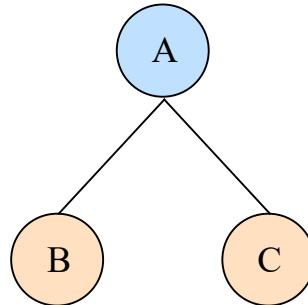


- En el cálculo proposicional, la expresión del grafo Y anterior correspondiente sería de la siguiente forma:

$$\mathbf{B \cdot C \rightarrow A}$$

Grafo Y/O

- **Grafo O:** Para completar el objetivo/tarea **A**, es necesario terminar antes o bien el objetivo/tarea **B**, o bien el objetivo/tarea **C**.

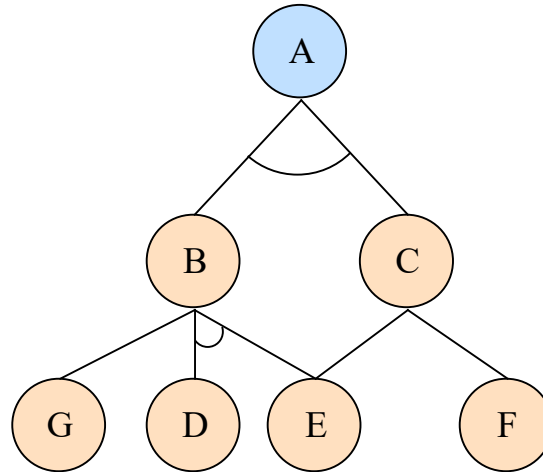


- En el cálculo proposicional, la expresión del grafo O anterior correspondiente sería de la siguiente forma:

$$\mathbf{B+C \rightarrow A}$$

Grafo Y/O

- **Grafo Y/O:** Combinación de grafos Y y grafos O que indican el orden de consecución de tareas a realizar para alcanzar el objetivo.



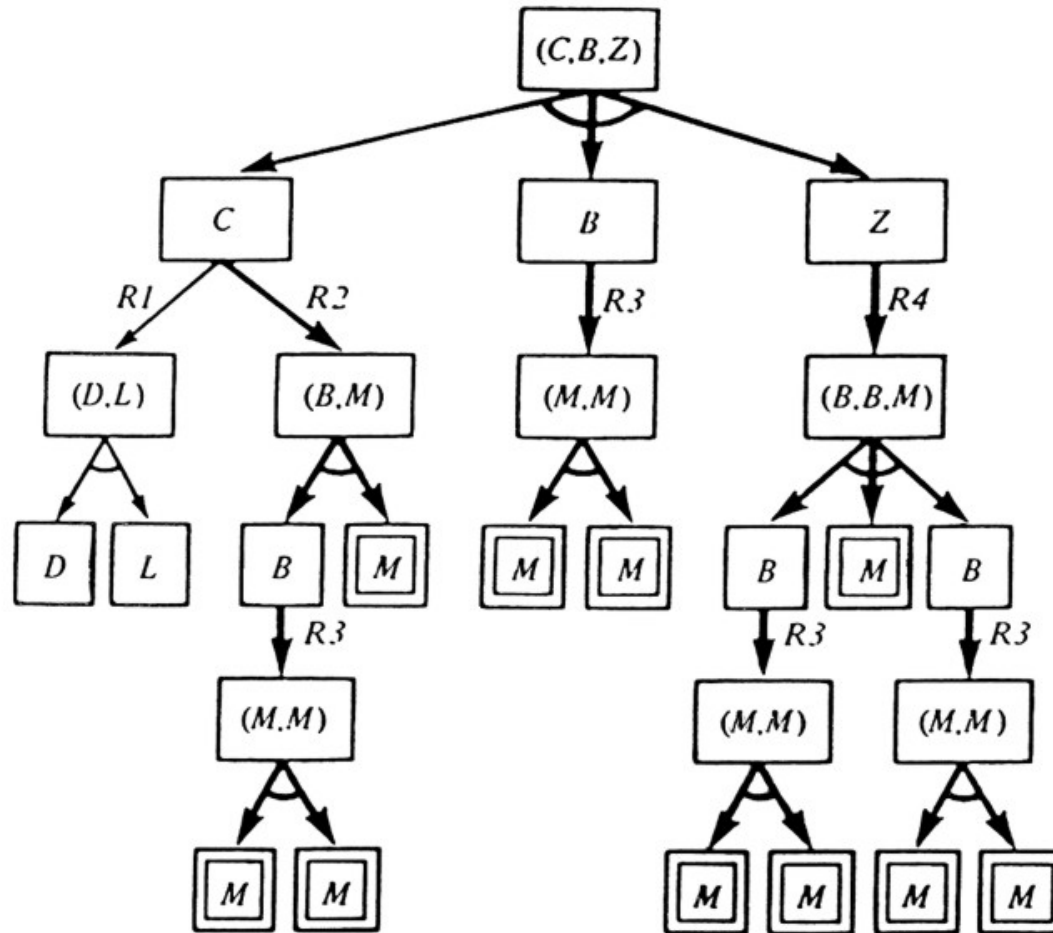
- En el cálculo proposicional, la expresión del grafo Y/O anterior correspondiente sería de la siguiente forma:

$$\mathbf{B \cdot C \rightarrow A; G + D \cdot E \rightarrow B; E + F \rightarrow C}$$

Grafo Y/O

- **Para resolver un grafo Y/O** cada nodo se resuelve de la siguiente manera:
 - Si es un nodo Y: Resolver todos sus hijos. Combinar la solución y solucionar el nodo. Devolver su solución.
 - Si es un nodo O: Resolver un hijo y ver si devuelve solución. En caso contrario, resolver el siguiente hijo, etc. Cuando ya esté resuelto algún hijo, combinar la solución en el nodo y devolverla.
 - Si es un nodo terminal: Resolver subproblema asociado y devolverla.
- **Mejora:** Para seleccionar el orden de resolución de nodos hijos, se puede utilizar alguna medida de estimación del coste de resolución.

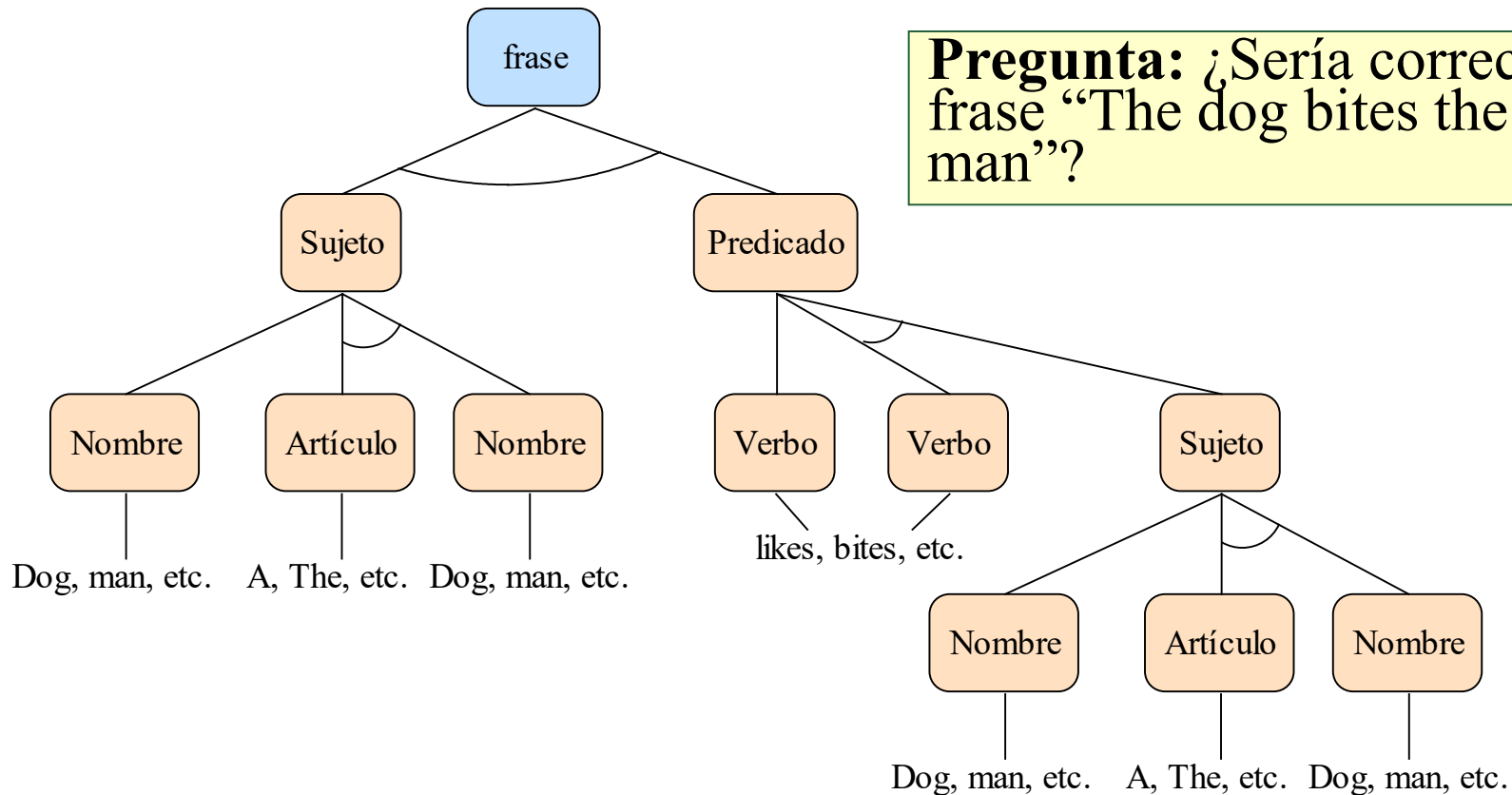
Nueva resolución del problema



Reconocimiento de frases de lengua inglesa

- Una frase está formada por un sujeto seguido de un predicado.
- El sujeto puede ser un sustantivo o un artículo seguido de un sustantivo.
- El predicado puede ser un verbo, o un verbo seguido de un complemento directo cuya estructura es idéntica a la del sujeto de la frase.

Reconocimiento de frases de lengua inglesa



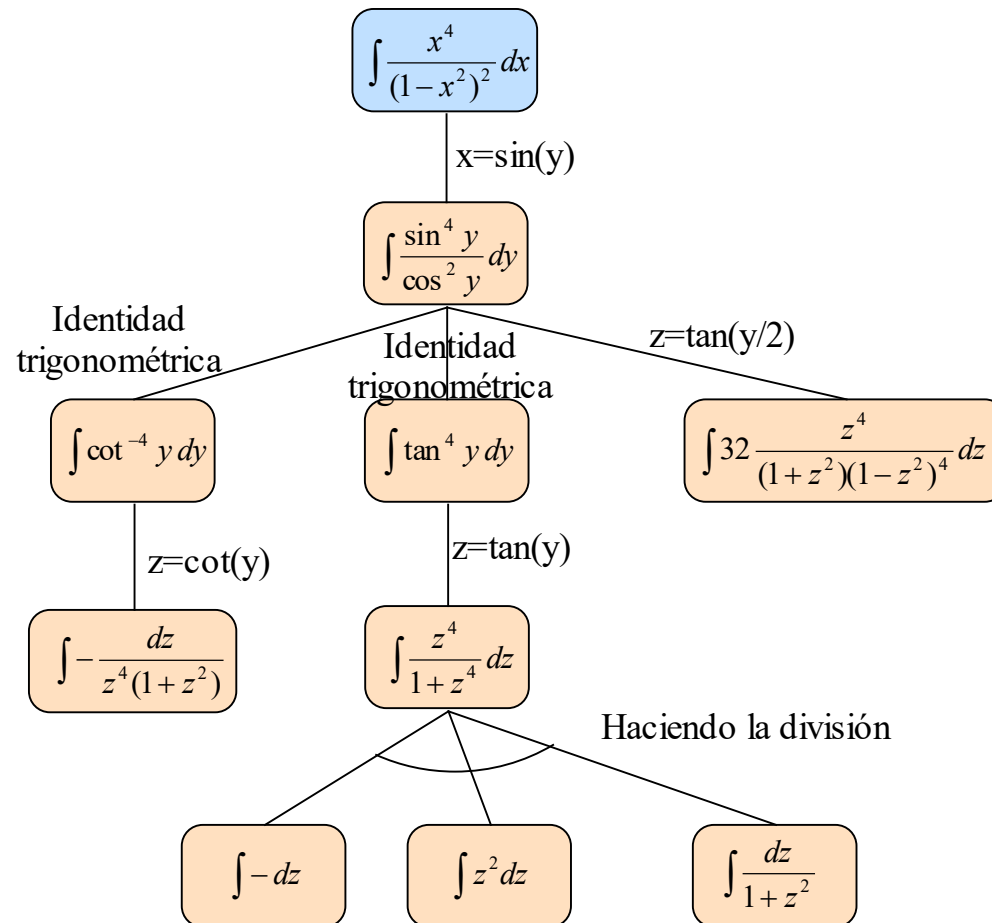
Pregunta: ¿Sería correcta la frase "The dog bites the man"?

Resolución de integrales

- Para simplificar, supongamos que el computador conoce las transformaciones y técnicas de integración, incluidas en una Base de Datos o de Conocimiento.
- Esta técnica es la que implementa el programa MACSYMA, muy utilizado por matemáticos.
- Supongamos que queremos hacer la siguiente integración:

$$\int \frac{x^4}{(1-x^2)^2} dx$$

Resolución de integrales



Búsqueda con acciones no deterministas

