Inteligencia Artificial

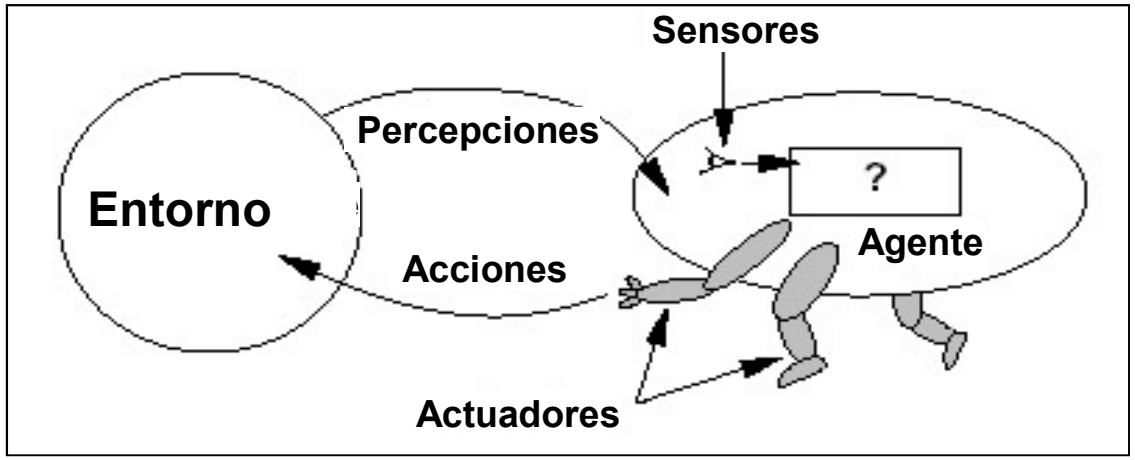
UNIVERSITAT DE BARCELONA

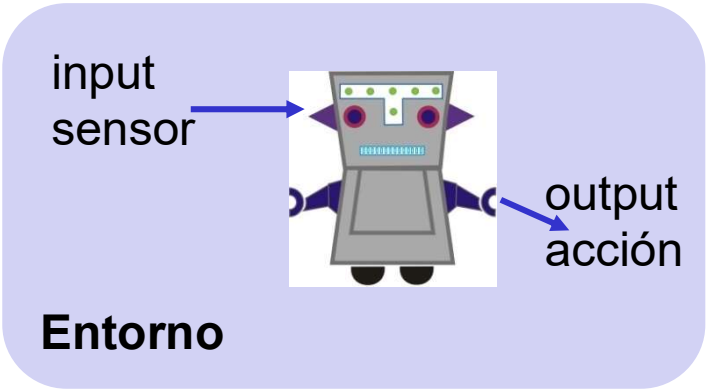
Resolución de Problemas Mediante Búsqueda: Búsqueda no Informada

Búsqueda no informada

1. Agentes para la resolución de problemas
2. Formulación de problemas
3. Problemas ejemplo
4. Búsqueda de soluciones
5. Estrategias básicas de búsqueda
   - Búsqueda primero en anchura
   - Búsqueda de coste uniforme
   - Búsqueda primero en profundidad
   - Búsqueda limitada en profundidad
   - Búsqueda por profundización iterativa
   - (Búsqueda bidireccional)

Agentes





Un agente percibe su entorno a través de sensores y actúa sobre el mismo mediante “actuadores”.

Los agentes incluyen a humanos, robots, softbots, termostatos, etc.

La función del agente proyecta una o varias percepciones en una acción:

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

El programa del agente se ejecuta sobre la arquitectura física para producir f.

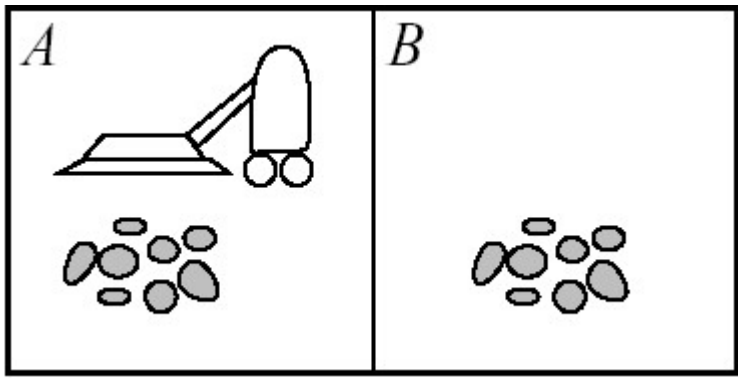
Agentes reactivos

- Agentes reactivos
  - Escogen una acción basándose en su actual percepción y en su memoria.
  - No piensan más allá del primer paso.
- ¿Puede un agente reactivo ser racional?

Agentes basados en objetivos

- Planifican con anterioridad
- Toman las decisiones en función de las consecuencias de sus acciones
- Deben tener un modelo de cómo el mundo evoluciona en respuesta a sus acciones

Ejemplo: El mundo aspiradora



Percepciones: localización y contenidos, por ejemplo, *[A, Sucio]*.

Acciones: *Izquierda, Derecha, Aspirar*

Un agente aspiradora

Secuencia de percepciones	Acción
[A, Limpio]	*Derecha*
[A, Sucio]	*Aspirar*
[B, Limpio]	*Izquierda*
[B, Sucio]	*Aspirar*
[A, Limpio], [A, Limpio]	*Derecha*
[A, Limpio], [A, Sucio]	*Aspirar*
⋮	⋮

función AGENTE-ASPIRADORA-REACTIVO([localización, estado]) **devuelve** una acción

si *estado* = *Sucio* entonces devolver *Aspirar*

si no, si *localización* = *A* **entonces devolver** *Derecha*

si no, si *localización* = *B* **entonces devolver** *Izquierda*

¿Cuál es la función que define la acción correcta?

Agentes para la resolución de problemas

- Los agentes reflejos simples tienen muchas limitaciones

- Los agentes para resolución de problemas basados en objetivos:
  - Agentes inteligentes: buscan secuencias de acciones que conducen a estados deseados.
  - Los objetivos permiten al agente:
    - dirigir su comportamiento limitando las acciones que intenta realizar.
    - actuar de forma que se maximice su medida del desempeño (racionalidad).

Agentes para Resolución de Problemas

Las **etapas** de la resolución de problemas con objetivos:

1. Formulación de objetivos: a partir de la situación actual, definir los estados objetivo y los factores que pueden influir en el grado de satisfacción de las distintas maneras de conseguirlo.

2. Formulación del problema: decidir qué acciones y estados considerar.

3. Búsqueda de la solución: decidir qué hacer examinando diferentes secuencias de acciones que llevan a estados objetivo y escogiendo la mejor.

4. Ejecución: ejecutar las acciones recomendadas.

Sencillo agente resolvedor de problemas

función AGENTE-SIMPLE-RESOLVEDOR-PROBLEMAS(percepción) **devuelve** una acción

entradas: *percepción*: una percepción

estático: *sec*: una secuencia de acciones, vacía inicialmente

estado: una descripción del estado actual del mundo

objetivo: un objetivo inicialmente nulo

problema: una formulación del problema

estado ← ACTUALIZAR-ESTADO(*estado, percepción*)

si *sec* está vacía **entonces hacer**

objetivo ← FORMULAR OBJETIVO(*estado*)

problema ← FORMULAR-PROBLEMA(*estado, objetivo*)

sec ← BÚSQUEDA(*problema*)

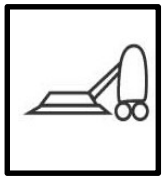
acción ← PRIMERO(*sec*)

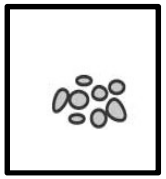
sec ← RESTO(*sec*)

devolver *acción*

OPEN-LOOP

Ejemplo: el mundo de la aspiradora





- Un mundo discreto de dos posiciones (celdas) adyacentes habitado por un robot
- El robot puede estar situado en cualquiera de las dos posiciones
- Las celdas pueden contener suciedad
- Se pretende llegar a una situación en la que el mundo esté limpio
- El robot aspiradora es el agente que puede cumplirlo
- Las acciones que el robot aspiradora puede hacer son:
  - Moverse a la derecha (si hay una celda a su derecha se mueve, si no se queda igual)
  - Moverse a la izquierda (equivalente a der)
  - Aspirar la suciedad de su celda (si no hay suciedad, se queda igual)
- Las percepciones del robot le permiten observar su posición y si hay suciedad o no en ella.

Ejemplo: el mundo de la aspiradora

- Mundo discreto de dos celdas adyacentes habitado por un robot que puede aspirar o moverse a derecha o izquierda
- Las celdas pueden contener suciedad, que se quiere eliminar
- Las percepciones del robot le permiten observar su posición y su entorno

Formulación del objetivo:

Formulación del problema:

estados:

acciones:

función sucesor

Búsqueda de la solución:

secuencia

Ejecución:

Ejemplo: el mundo de la aspiradora

- Mundo discreto de dos celdas adyacentes habitado por un robot que puede aspirar o moverse a derecha o izquierda
- Las celdas pueden contener suciedad, que se quiere eliminar
- Las percepciones del robot le permiten observar su posición y su entorno

Formulación del objetivo:

mundo limpio (estados 7 u 8)

Formulación del problema:

estados: ?

acciones: der, izq, asp

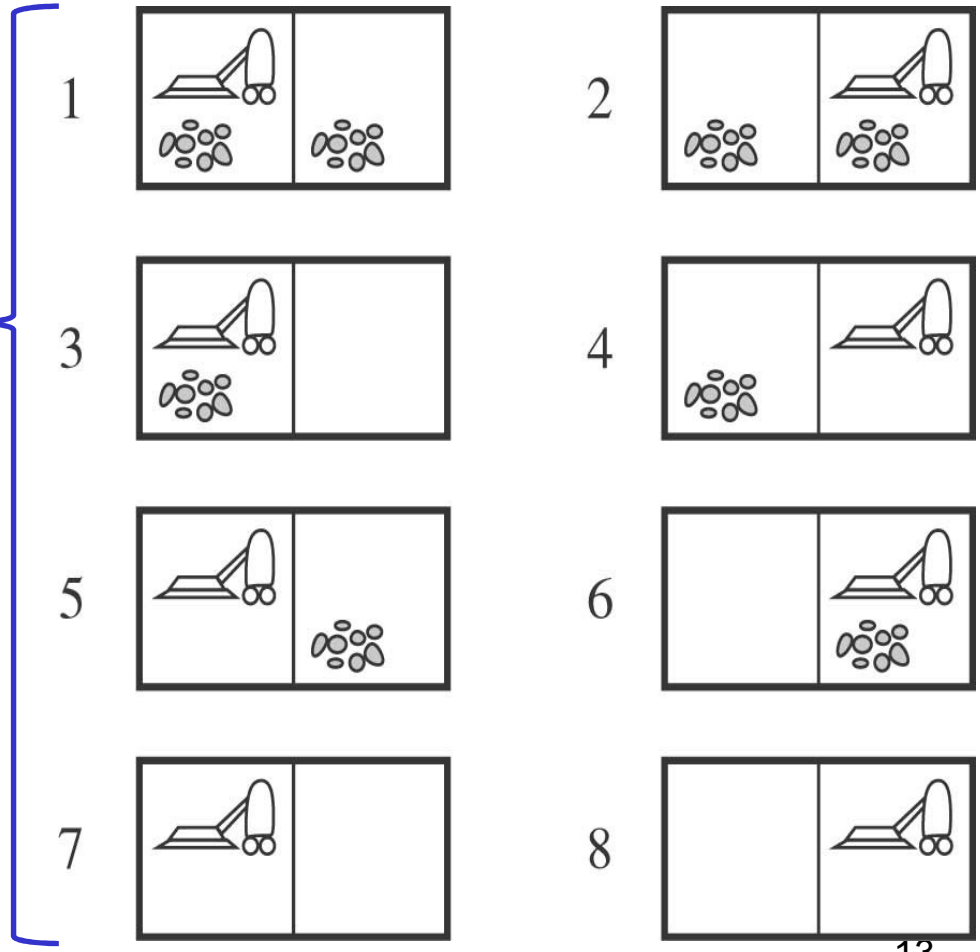
función sucesor ?

Búsqueda de la solución:

secuencia acciones

Ejecución:

ejecutar las acciones



Ejemplo: el mundo de la aspiradora

Formulación formal del problema:

Estados = { (posRobot, (suc1, suc2) } t.q. posRobot ∈{ I, D}, suc1, suc2 ∈{0, 1}

estado inicial : 5 =(I ,(0, 1))

estados finales: 7 =(I, (0,0)) y 8=(D, (0,0))

acciones: der, izq, asp

función sucesor:

S(1)={ <der, 2>, <izq, 1>, <asp, 5> }

S(2)= { <der, 2>, <izq, 1>, <asp, 4>}

S(3) ={ <der, 4>, <izq, 3>, <asp, 7>}

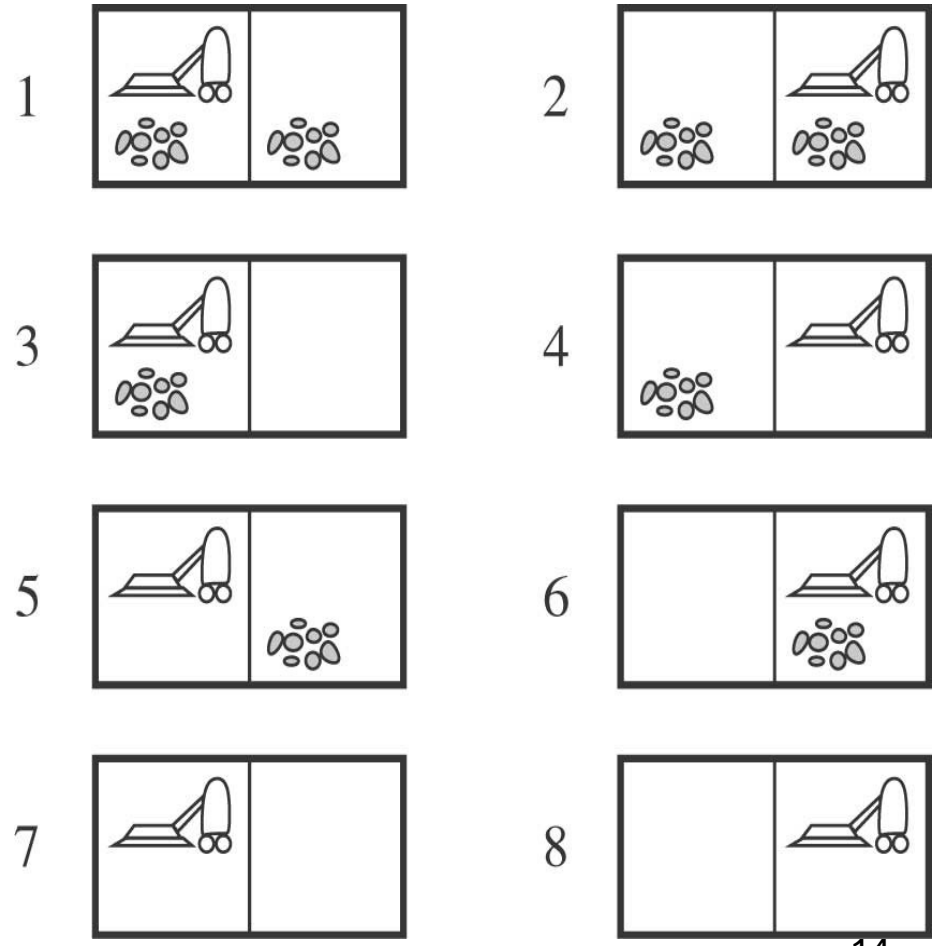
S(4)= { <der, 4>, <izq, 3>, <asp, 4>}

S(5)={ <der, 6> , <izq, 5> ,<asp, 5>}

S(6)= { <der, 6>, <izq, 5>, <asp, 8>}

S(7)={ <der, 8>, <izq, 7> ,<asp, 7>}

S(8)= { <der, 8>, <izq, 7> ,<asp, 8>}



Ejemplo: el mundo de la aspiradora

Formulación formal del problema:

Estados = { (posRobot, (suc1, suc2) } t.q. posRobot ∈{ I, D}, suc1, suc2 ∈{0, 1}

estado inicial : 5 =(I ,(0, 1))

estados finales: 7 =(I, (0,0)) y 8=(D, (0,0))

acciones: der, izq, asp

función sucesor:

S(1)={ <der, 2>, <izq, 1>, <asp, 5> }

S(2)= { <der, 2>, <izq, 1>, <asp, 4>}

S(3) ={ <der, 4>, <izq, 3>, <asp, 7>}

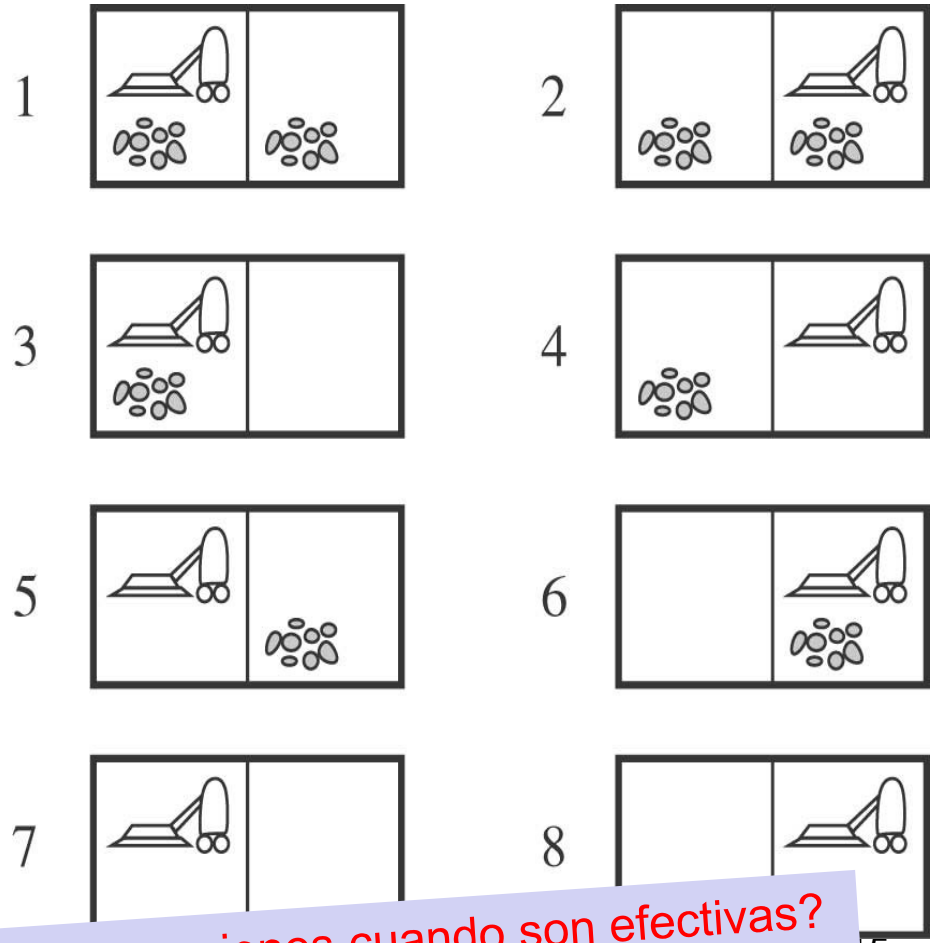
S(4)= { <der, 4>, <izq, 3>, <asp, 4>}

S(5)={ <der, 6> , <izq, 5> ,<asp, 5>}

S(6)= { <der, 6>, <izq, 5>, <asp, 8>}

S(7)={ <der, 8>, <izq, 7> ,<asp, 7>}

S(8)= { <der, 8>, <izq, 7> ,<asp, 8>}



¿Y si sólo se pudieran hacer las acciones cuando son efectivas?

Ejemplo: el mundo de la aspiradora

Si sólo se pudieran hacer las acciones cuando son efectivas

Formulación formal del problema:

Estados = { (posRobot, (suc1, suc2) } t.q. posRobot :pR∈{ I, D}, suc1, suc2 ∈{0,

estado inicial : 5 =(I ,(0, 1))

estados finales: 7 =(I, (0,0)) y 8=(D, (0,0))

acciones: der (si pR=I), izq (si pR=D),

asp (si pR=D&suc2=1 o si pR=I&suc1=1)

función sucesor:

S(1) = {<der, 2>, <asp, 5> }

S(2) = {<izq, 1>, <asp, 4>}

S(3) = {<der, 4>, <asp, 7>}

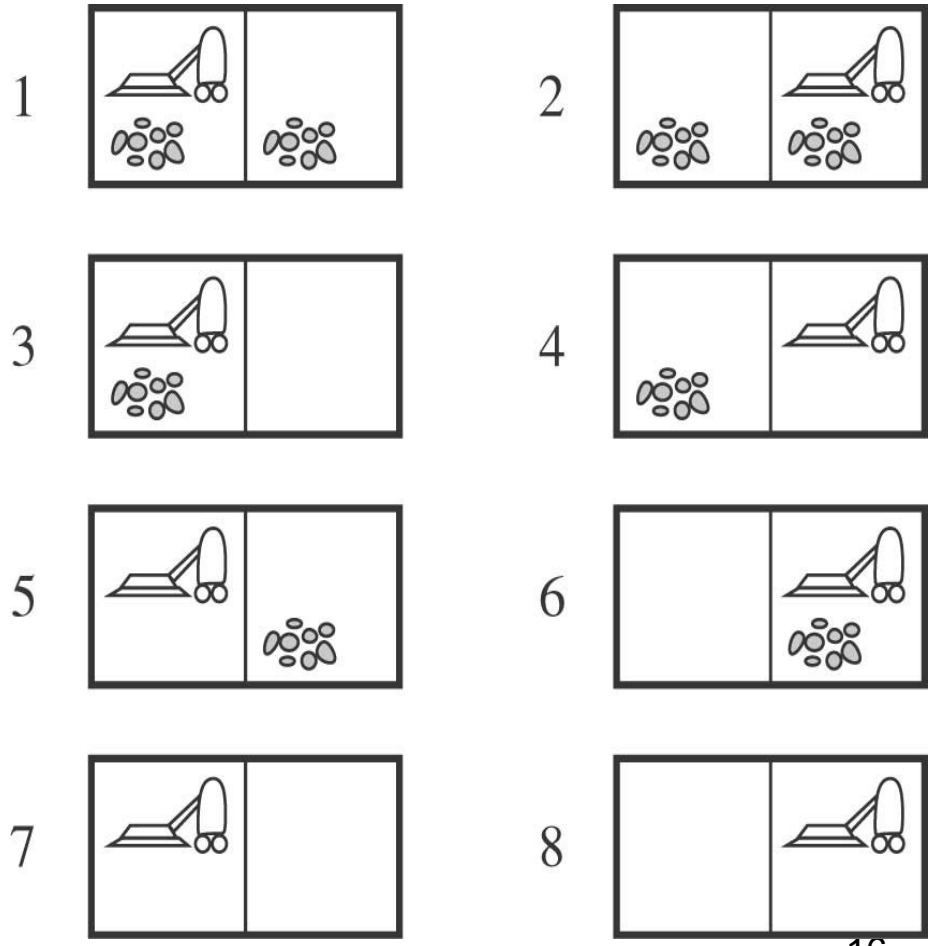
S(4) = {<izq, 3>}

S(5) = {<der, 6> }

S(6) = {<izq, 5>, <asp, 8>}

S(7) = {<der, 8>}

S(8) = {<izq, 7>}



Tipos de Problemas

Deterministas, completamente observables ⇒ problemas de estado simple
El agente sabe exactamente en qué estado estará; la solución es una secuencia.

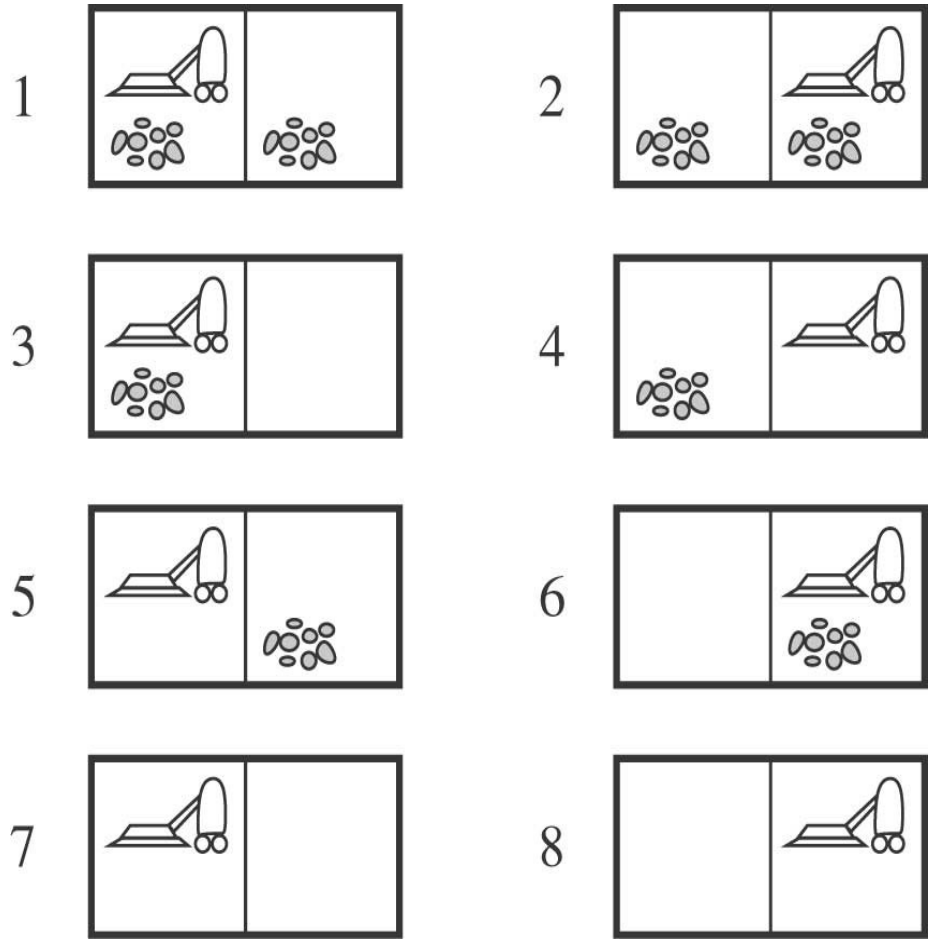
No observables ⇒ problemas conformados (conformant or sensorless)
El agente puede no saber dónde está; la solución (si la hay) es una secuencia.

No deterministas y/o parcialmente observables ⇒ problemas de contingencia
Las percepciones del agente proporcionan información nueva sobre el estado actual.
La solución es un árbol o una política.
A menudo intercalan la búsqueda y la ejecución.

Problemas de espacio-estado desconocido ⇒ problemas de exploración (“en línea”).

Ejemplo: el mundo de la aspiradora

Estado simple (det, compl. obs.)
estado inicial 5. ¿Solución?

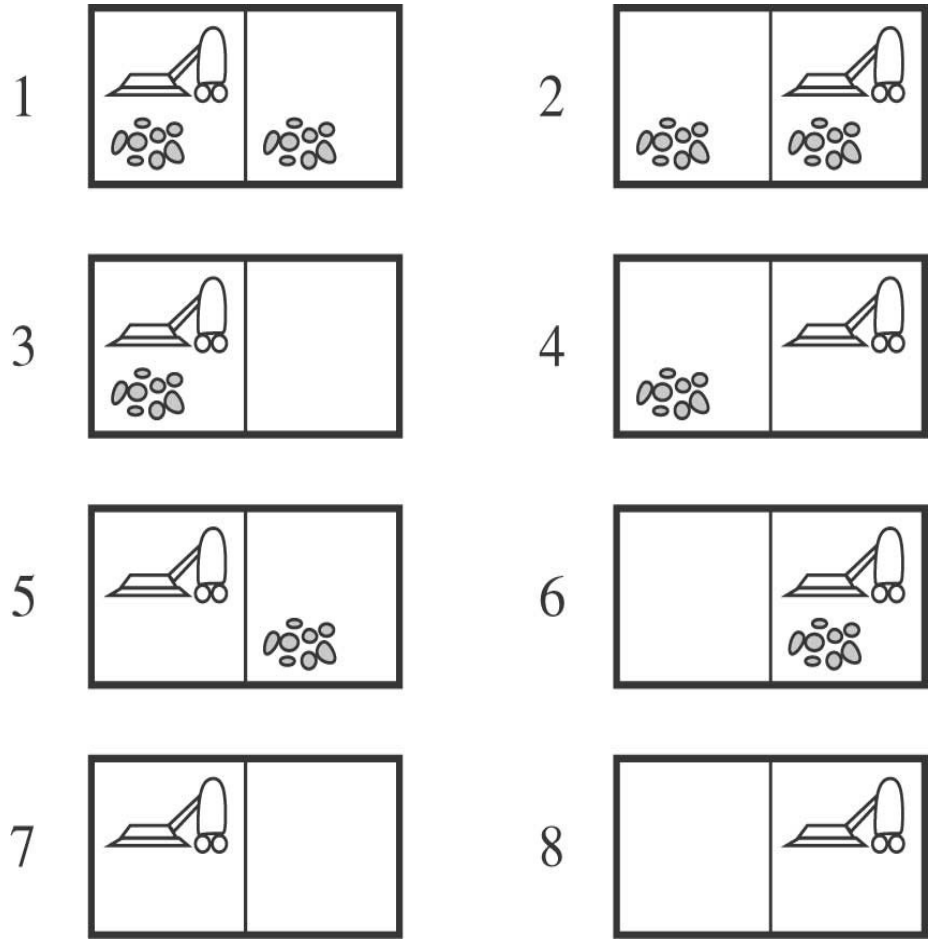


Ejemplo: el mundo de la aspiradora

Estado simple,
estado inicial 5.
¿Solución?

Sec= [*Derecha*, *Aspirar*]

(Es una solución porque al seguir esta secuencia de acciones el robot pasa, de estar en el estado inicial 5, al 6 y a continuación acaba en el 8, que es uno de los estados objetivo).



Formulación de Problemas de Estado Simple

Un *problema* se define por cuatro elementos:

estado inicial por ejemplo 5 =(I ,(0, 1))

función sucesor *S*(*x*) = conjunto de pares acción-estado
p. ej. S(5) ={ <der, 6> , <izq, 5> ,<asp, 5>}

Espacio de estados

test objetivo, puede ser
explícito, p. ej, x = (7 o 8) donde 7=(I ,(0, 0)), 8=(D ,(0, 0))
implícito, p. ej, NoSuciedad(x)

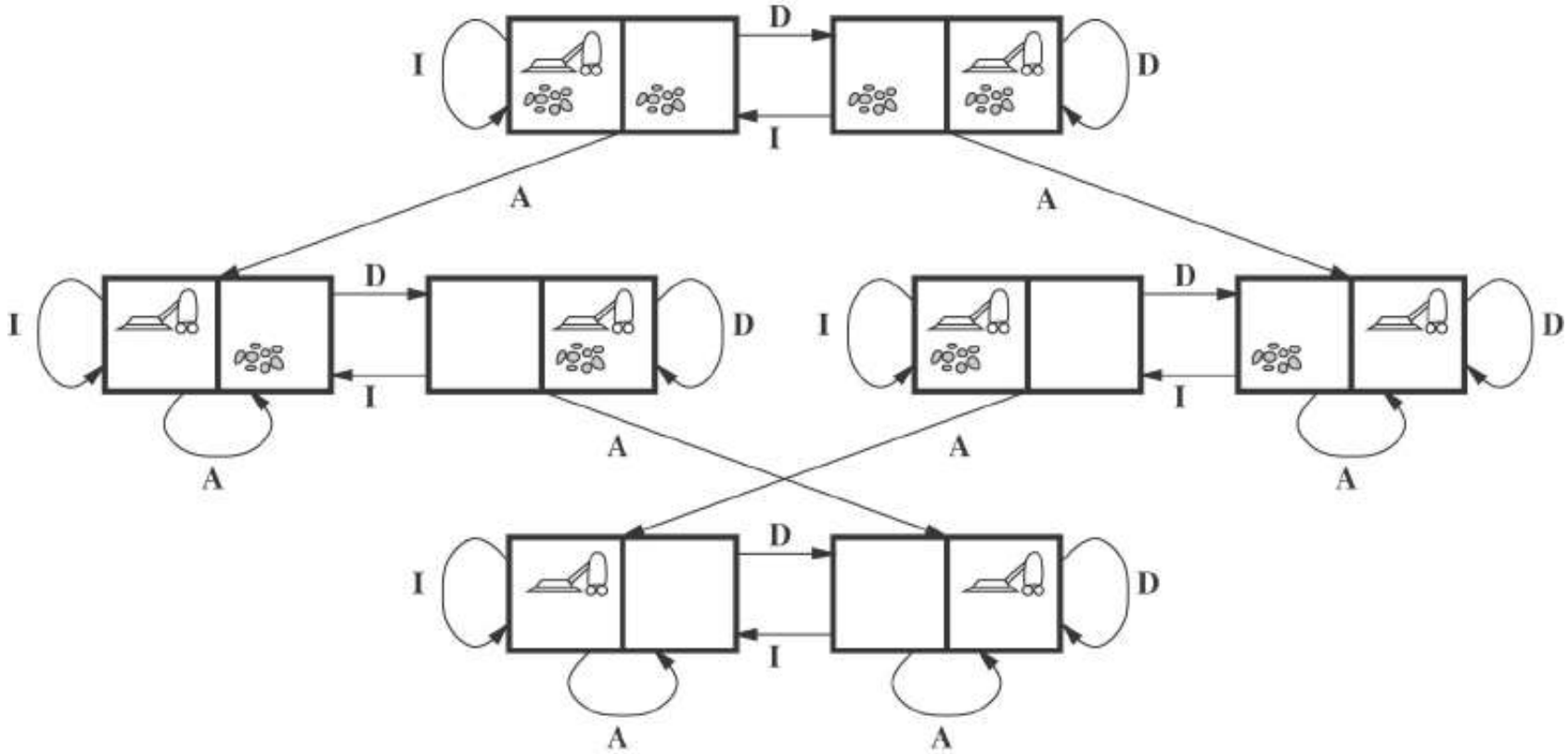
coste del camino (aditivo)
p.ej.: distancia total recorrida, número de acciones ejecutadas, etc.
c(*x, a, y*) es el coste individual, se supone que es ≥ 0

Una ***solución*** es una secuencia de acciones que parten desde un estado inicial hasta alcanzar un estado objetivo.

Ejercicio: dibujar el espacio de estados para el mundo de la aspiradora

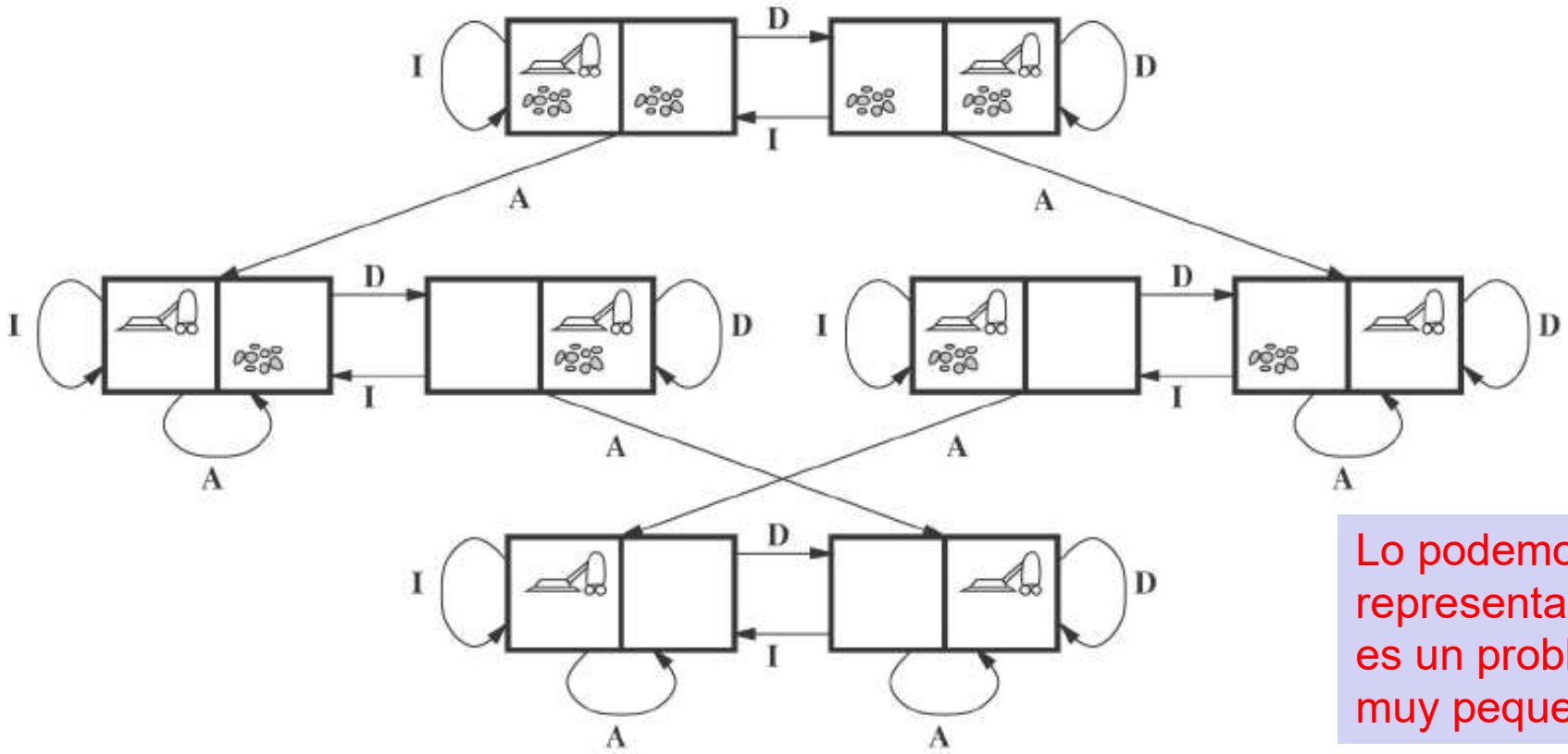
¿Estados?
¿Acciones?

Ejemplo: espacio de estados para el mundo de la aspiradora



¿Estados?
¿Acciones?
¿Test objetivo?
¿Coste del camino?

Ejemplo: espacio de estados para el mundo de la aspiradora



Lo podemos representar porque es un problema muy pequeño

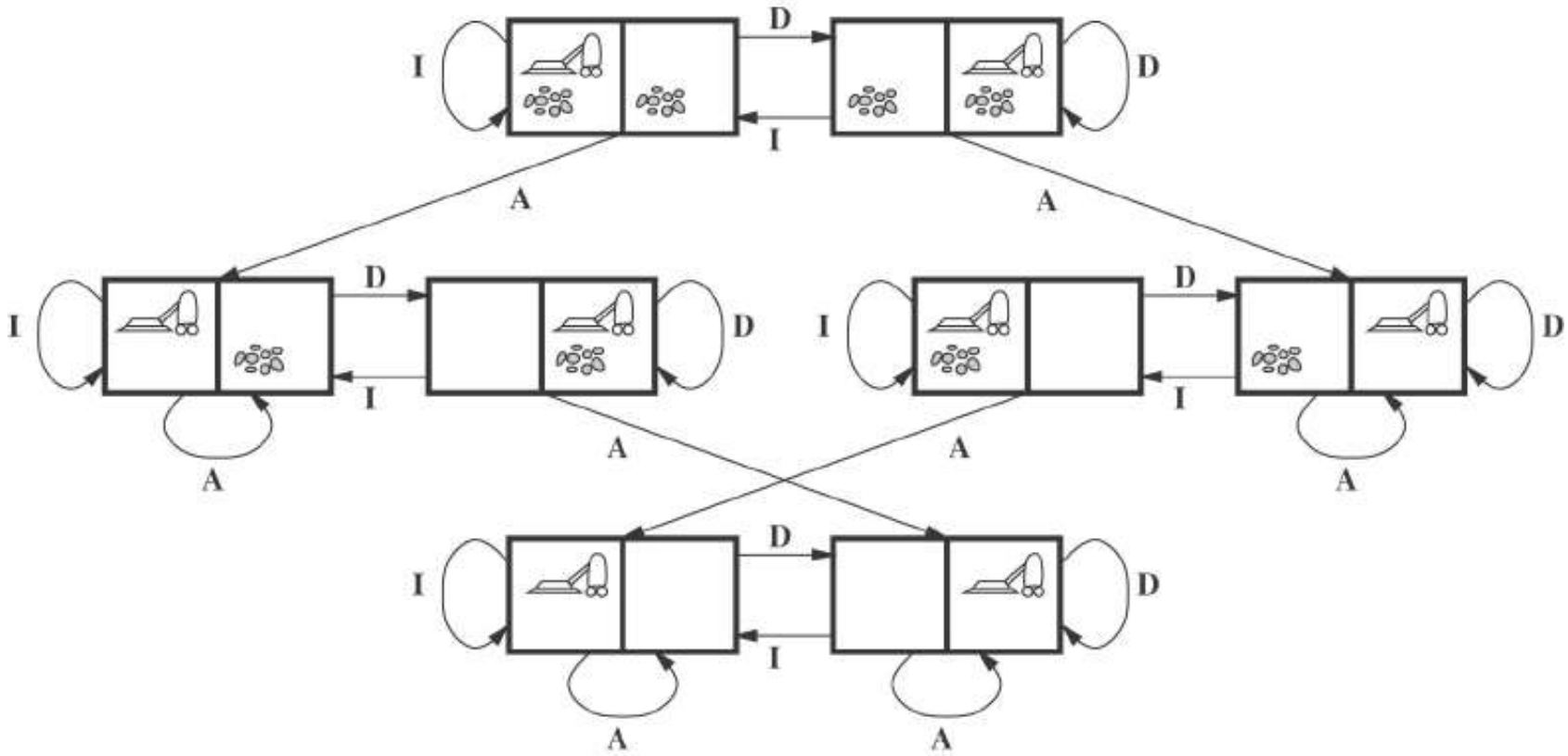
¿Estados? Suciedad completa y localizaciones de robot (ignorar *cantidades* de suciedad)

¿Acciones? *Izquierda (I), Derecha (D), Aspirar (A)*

¿Test objetivo? No suciedad

¿Coste del camino? 1 por acción

Ejemplo: espacio de estados para el mundo de la aspiradora



¿Estados? Suciedad completa y localizaciones de robot (ignorar *cantidades* de suciedad)

¿Acciones? *Izquierda (I), Derecha (D), Aspirar (A)*

¿Test objetivo? No suciedad

¿Coste del camino? 1 por acción

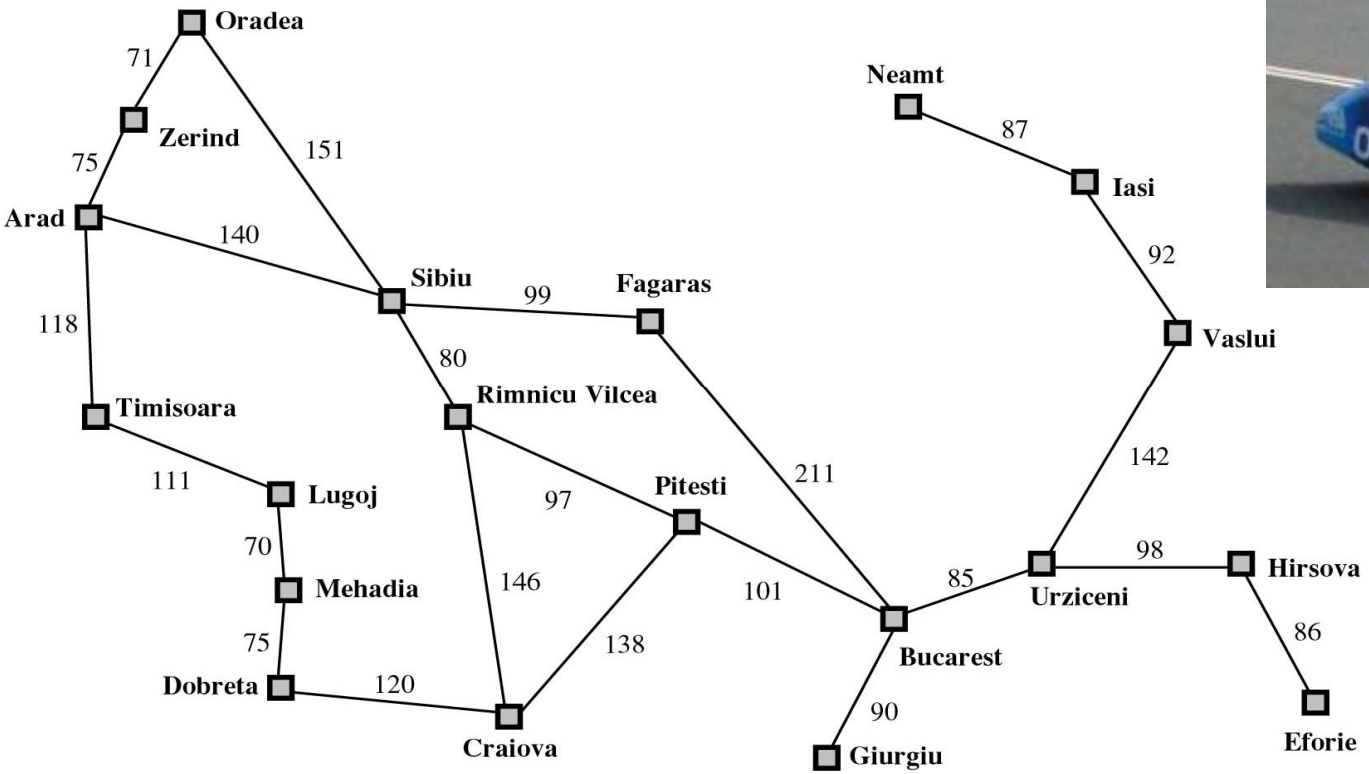
¿Y si sólo se pudieran hacer las acciones cuando son efectivas?

Video Pacman/Roomba

- http://pacman.elstonj.com/index.cgi?dir=videos&num=&perpage=&section=

Ejemplo: Hallar rutas en Rumanía

Un agente en la ciudad de Arad (Rumanía), que dispone de un coche y un mapa de carreteras entre las principales ciudades de Rumanía. Mañana sale un vuelo a Bucharest.



Formulación de Problemas de Estado Simple

Un *problema* se define por cuatro elementos:

estado inicial por ejemplo, En(Arad)

función sucesor $S(x)$ = conjunto de pares acción-estado
p. ej. S(En(Arad)) = { <Ir(Zerind), En(Zerind)>,...}

Espacio de estados

test objetivo, puede ser
explícito, p. ej, x = En(Bucharest)
implícito, p. ej, NoSuciedad(x)

coste del camino (aditivo)
p.ej.: suma de distancias, número de acciones ejecutadas, etc.
$c(x, a, y)$ es el coste individual, se supone que es ≥ 0

Una ***solución*** es una secuencia de acciones que parten desde un estado inicial hasta alcanzar un estado objetivo.

Ejemplo: Hallar rutas en Rumanía

Un agente en la ciudad de Arad (Rumanía), que dispone de un coche y un mapa de carreteras entre las principales ciudades de Rumanía. Mañana sale un vuelo a Bucharest.

Formulación del objetivo:

?

Formulación del problema:

estados: ?

acciones: ?

Búsqueda de la solución:

secuencia: ?

Ejecución:

?

Ejemplo: Hallar rutas en Rumanía

Un agente en Arad dispone de un coche y un mapa de carreteras para llegar a Bucharest.

Formulación del objetivo:

estar en Bucharest

Formulación del problema:

estados: cada estado es estar en una ciudad, #estados=#ciudades

acciones: conducir entre las ciudades siguiendo los tramos de carretera

Búsqueda de la solución:

secuencia de rutas entre ciudades, p.ej.:, ir de Arad a Sibiu, ir a Fagaras, ir a Bucharest.

Ejecución:

conducir por la secuencia de ciudades (i.e., las rutas entre ciudades)

Ejemplo: Hallar rutas en Rumanía

Un agente en Arad dispone de un coche y un mapa de carreteras para llegar a Bucharest

Formulación formal del problema:

Ciudades = {Oradea, Zerind, Arad, Sibiu, Timisoara, Lugoj, Mehaia, Drobeta, Craiova, Rimicu Vilcea, Fagaras, Pitesti, Bucharest, Giurgiu, Irrziceni, Hirsova, Eforie, Vaslui, Iasi, Neamt}

Estados: cada estado es estar en una ciudad,

Estados ={ En(x) | x ∈ Ciudades}

Estado inicial = En(Arad)

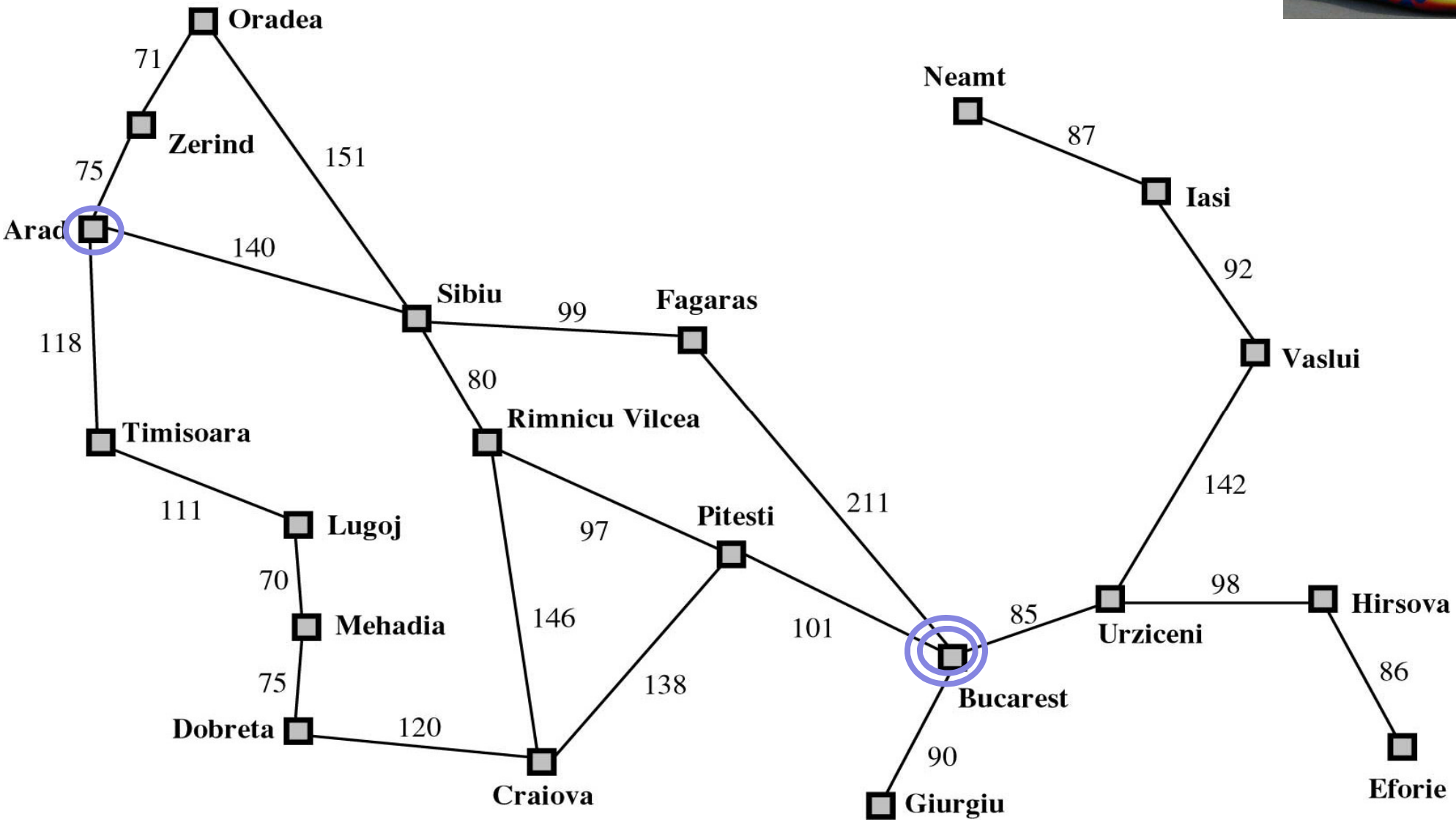
Estado objetivo = En(Bucharest)

Acciones: conducir de una ciudad a otra

Acciones= { Ir(x,y) | x,y ∈ Ciudades, x≠y, TramoMapa(x,y)}

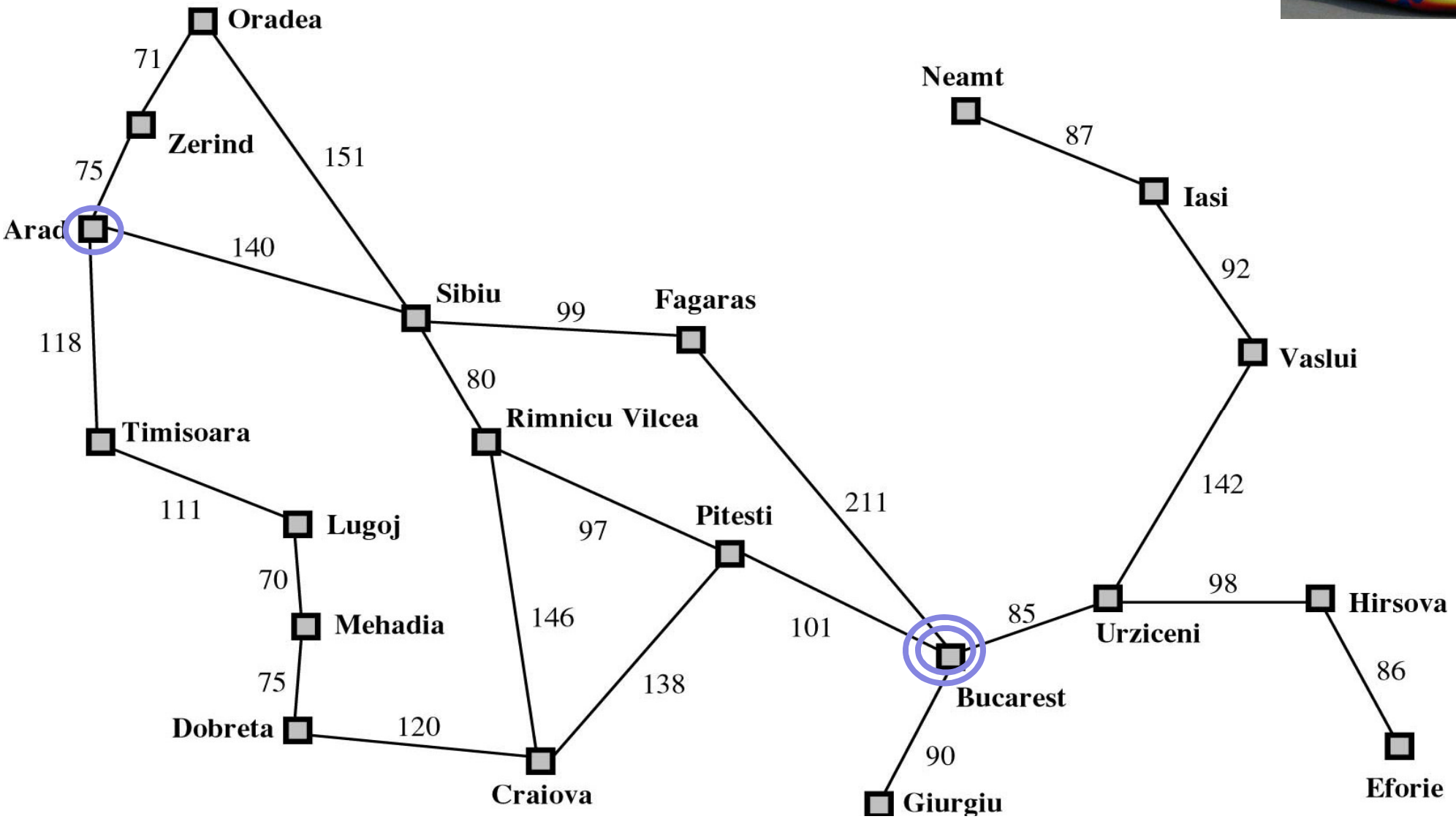
Función sucesor : S(En(x))= { < Ir(x,y), En(y) > }

Ejemplo: Hallar rutas en Rumanía



S(En(Arad)) ?

Ejemplo: Hallar rutas en Rumanía



S(En(Arad))={ <Ir(Sibiu),En(Sibiu)>, <Ir(Timisoara), En(Timisoara)>, <Ir(Zerind),En(Zerind)>}

S(En(Sibiu)) ?

S(En(Neamt))

Selección de un espacio de estados

El mundo real es tremendamente complejo

$\Rightarrow$ el espacio de estado se debe *abstraer* de la solución del problema

Estado (abstracto) = conjunto de estados reales

Acción (abstracta) = combinación compleja de acciones reales

p.e.: Ir(Zerind) representa un conjunto complejo de rutas posibles, tours, paradas de descanso, etc.

Para una consecución garantizada, cualquier estado real En(Arad) debe llegar a *algún* estado real En(Zerind).

Solución (abstracta) = conjunto de caminos reales que son soluciones en el mundo real

¡Cada acción abstracta debería ser “más fácil” que el problema original!

Algoritmos de búsqueda en árboles

Idea básica:
• exploración offline simulada del espacio de estados generado
• *expansión* de los estados: obtención de los sucesores de estados ya explorados

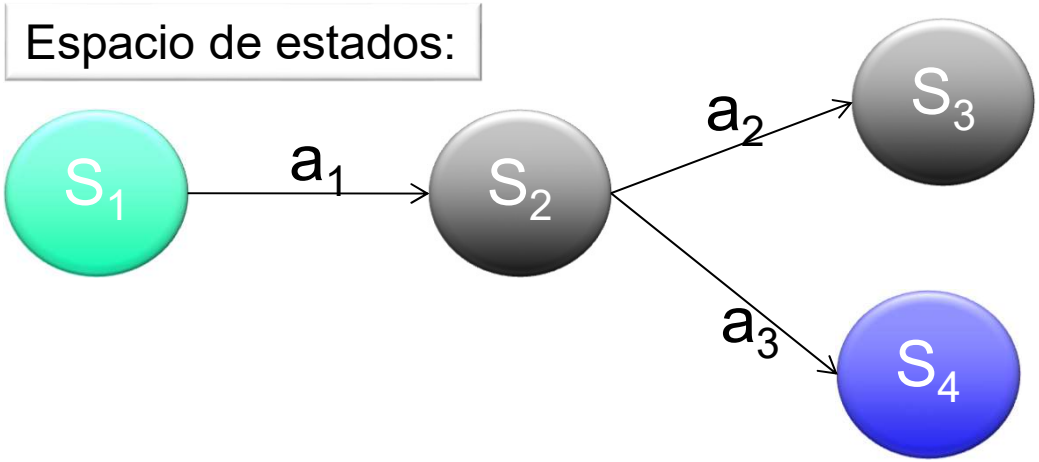
función BÚSQUEDA-ÁRBOLES(*problema*,*estrategia*) **devuelve** una solución o fallo

inicializa el árbol de búsqueda usando el estado inicial del *problema*

bucle hacer

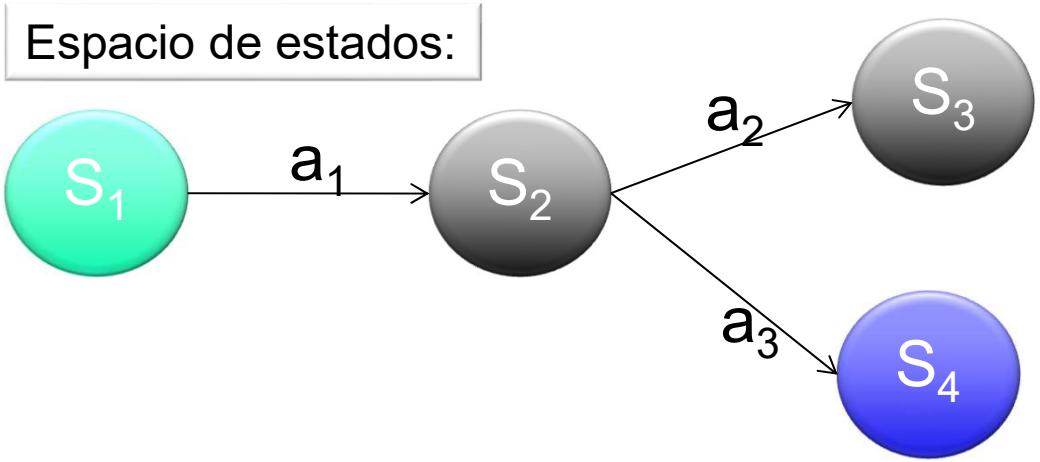
si no hay candidatos para expandir en el árbol de búsqueda **entonces devolver** fallo
escoger, de acuerdo a la *estrategia*, un nodo hoja para expandir
si el nodo contiene un estado objetivo **entonces devolver** la correspondiente solución
en otro caso expandir el nodo y añadir los nodos resultado al árbol de búsqueda

Ejemplo



- ¿Problema?
- ¿Búsqueda-árboles (problema, estrategia)?

Ejemplo: problema

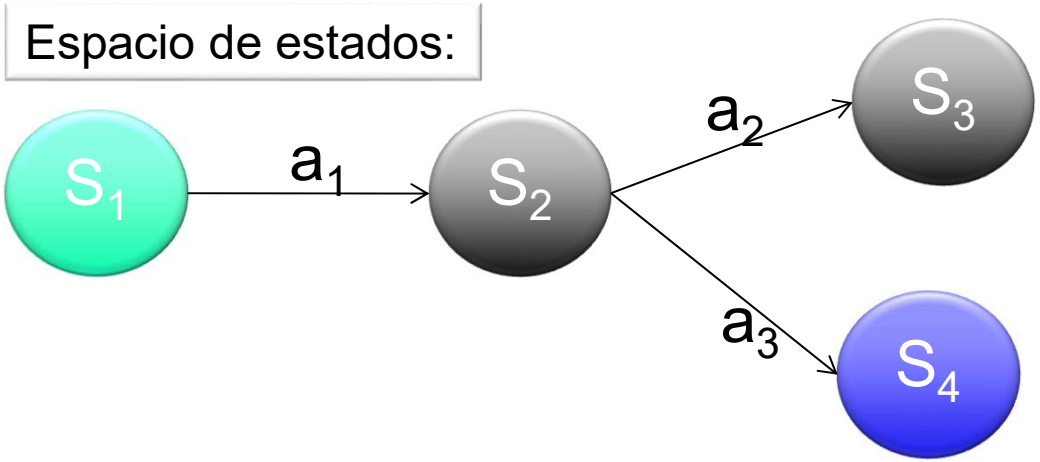


- Problema:

estado inicial
función sucesor :
test objetivo
coste del camino:

- Búsqueda-árboles (problema, estrategia)?

Ejemplo: problema



- Problema:

estado inicial s_1
función sucesor :
$S(s_1) = \{<a_1,s_2>\}$,
$S(s_2)=\{<a_2,s_3>, <a_3,s_4>\}$,
$S(s_3)=S(s_4)= \emptyset$
test objetivo, $x = s_4$
coste del camino:
(uniforme =1) $c(s_i,a_j,s_k)=1$ en las transiciones válidas

- Búsqueda-árboles (problema, estrategia)?

Ejemplo: ¿ejecución del algoritmo?

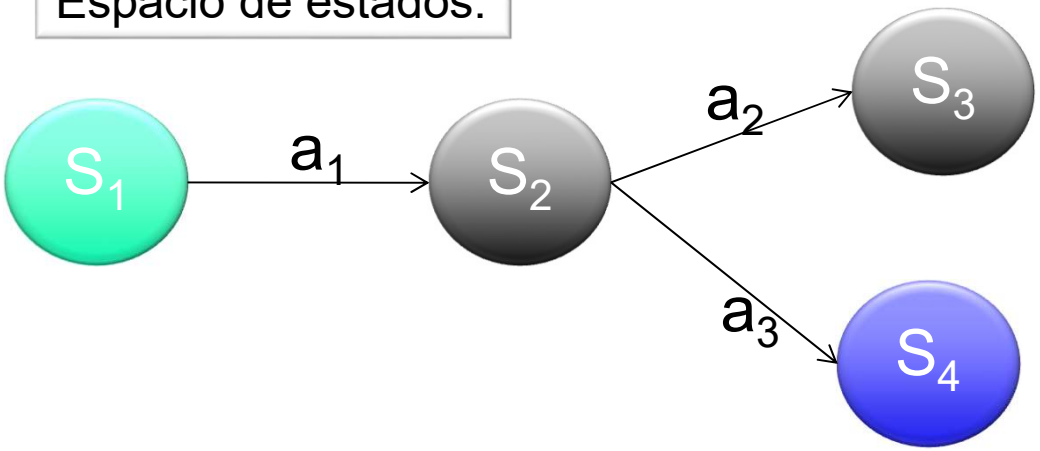
función BÚSQUEDA-ÁRBOLES(*problema*,*estrategia*) **devuelve** una solución o fallo

inicializa el árbol de búsqueda usando el estado inicial del *problema*

bucle hacer

- **si** no hay candidatos para expandir en el árbol de búsqueda **entonces devolver** fallo
- escoger, de acuerdo a la *estrategia*, un nodo hoja para expandir
- **si** el nodo contiene un estado objetivo **entonces devolver** la correspondiente solución
- **en otro caso** expandir el nodo y añadir los nodos resultado al árbol de búsqueda

Espacio de estados:



Ejecución del algoritmo ?

Ejemplo: ejecución del algoritmo

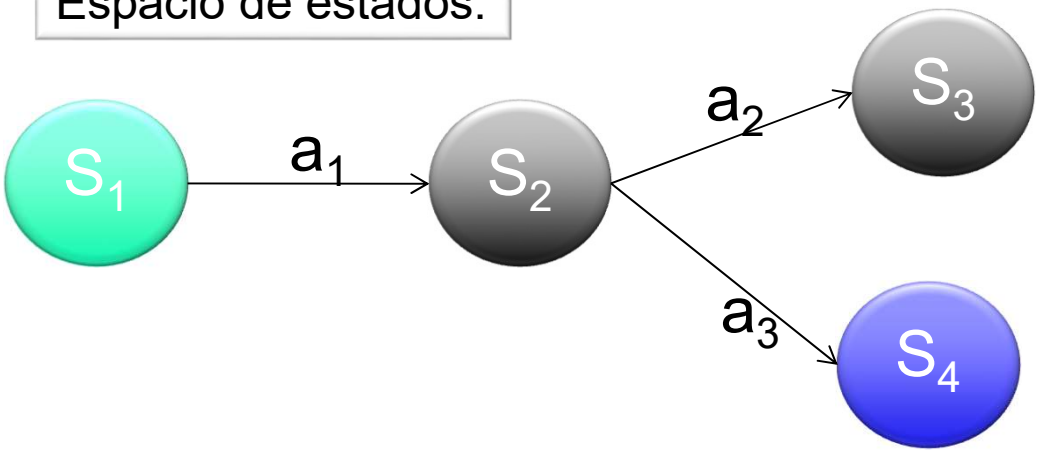
función BÚSQUEDA-ÁRBOLES(*problema*,*estrategia*) **devuelve** una solución o fallo

inicializa el árbol de búsqueda usando el estado inicial del *problema*

bucle hacer

- **si** no hay candidatos para expandir en el árbol de búsqueda **entonces devolver** fallo
- escoger, de acuerdo a la *estrategia*, un nodo hoja para expandir
- **si** el nodo contiene un estado objetivo **entonces devolver** la correspondiente solución
- **en otro caso** expandir el nodo y añadir los nodos resultado al árbol de búsqueda

Espacio de estados:



Inicialmente Arb= {n_s_1}
It 1: Arb ≠ Ø
act = n_s_1 (Arb ={})
act.est ≠ s_4 → Arb ={n_s_2}
It 2: Arb ≠ Ø
act = n_s_2 (Arb ={})
act.est ≠ s_4 → Arb ={n_s_3, n_s_4}
It 3: Arb ≠ Ø
act = n_s_3 (Arb ={n_s_4})
act.est ≠ s_4 → Arb ={n_s_4}
It 4: Arb ≠ Ø
act = n_s_4 (Arb ={})
act.est = s_4 → return solucion(n_s_4)

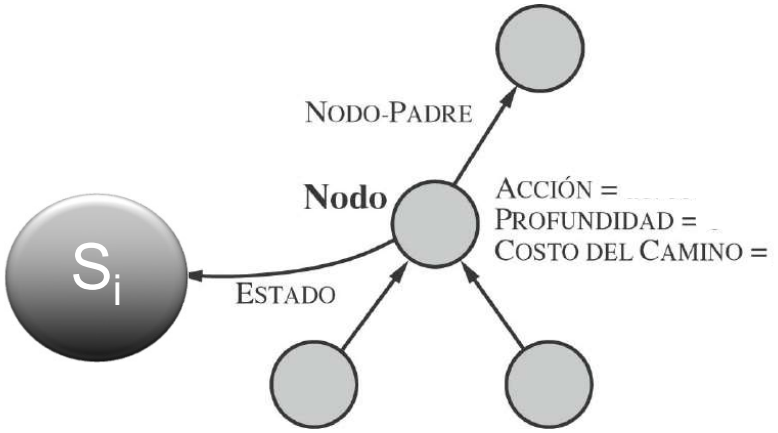
solucion(n_s_4) = {a_1, a_3}

Implementación: estados frente a nodos

Un *estado* es una (representación de) una configuración física.
Un *nodo* es una estructura de datos que forma parte de un árbol de búsqueda que incluye *estado*, *padre*, *hijos*, *profundidad*, *coste del camino* $g(x)$.

¡Los estados no tienen padres, hijos, profundidad o coste del camino!

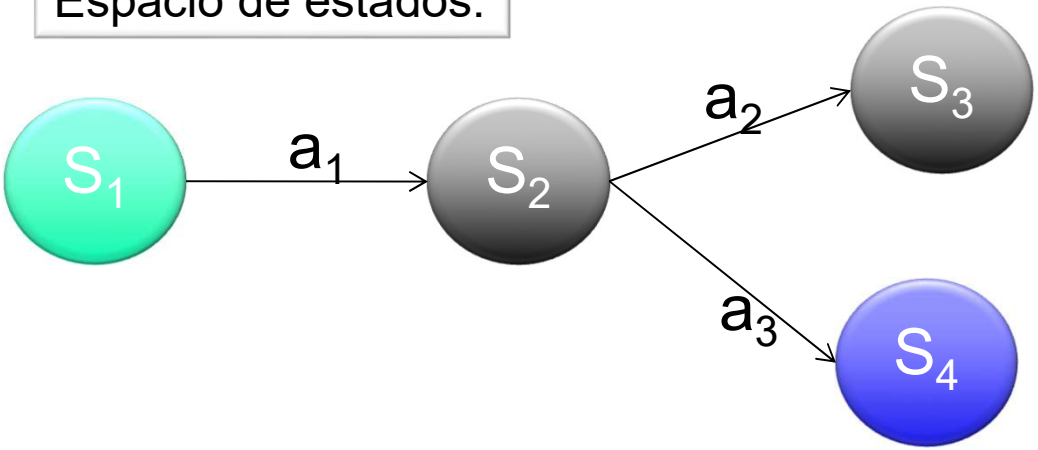
Notar que estado ≠ nodo del árbol de búsqueda:



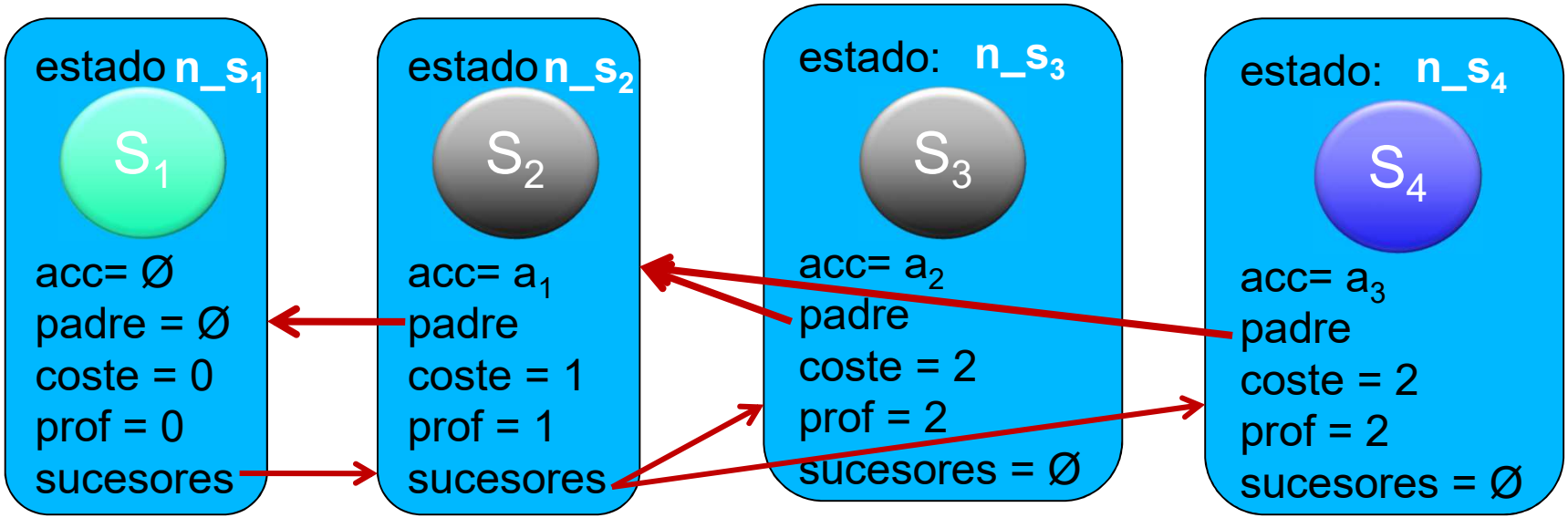
EXPANDIR un nodo consiste en utilizar la función sucesor (SUCESOR-EN) del problema para obtener los estados correspondientes y con ellos crear nuevos nodos (rellenando los distintos campos)

Implementación: estados frente a nodos

Espacio de estados:



Estructura de nodos (árbol de búsqueda):



Implementación: algoritmo general de búsqueda en árboles

función BÚSQUEDA-ÁRBOLES(*problema,frontera*) **devuelve** una solución o fallo

frontera ← INSERTA(HACER-NODO(ESTADO-INICIAL[*problema*]), *frontera*)

bucle hacer

si VACIA? (*frontera*) **entonces devolver** fallo

nodo ←SACAR-BORRANDO-PRIMERO(*frontera*)

si TEST-OBJETIVO[*problema*] aplicado al ESTADO[*nodo*] es cierto

entonces devolver SOLUCION(*nodo*)

frontera ← INSERTAR-TODO(EXPANDIR(*nodo, problema*), *frontera*)

---

función EXPANDIR(*nodo, problema*) **devuelve** un conjunto de nodos

sucesores ← conjunto vacío

para cada (*acción, resultado*) **en** SUCESOR-EN[*problema*](ESTADO[*nodo*]) **hacer**

s ←un nuevo NODO

ESTADO[s] ←*resultado*

NODO-PADRE*[s]* ←*nodo*

ACCIÓN[s] ←*acción*

COSTE-CAMINO[s] ←*COSTE-CAMINO[nodo]* + *COSTE-INDIVIDUAL(nodo.acción , s)*

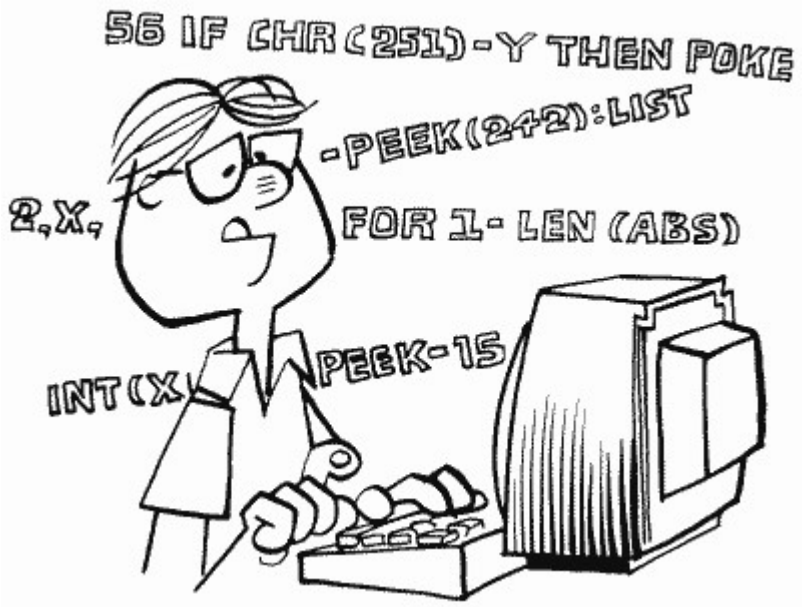
PROFUNDIDAD[s] ←*PROFUNDIDAD[nodo]* + *1*

añadir *s* a *sucesores*

devolver *sucesores*

Detalles de implementación

- Los nodos son conceptualmente caminos, pero es mejor representarlos con un estado, un coste, el valor de la última acción y una referencia al nodo padre.



Estrategias Básicas de Búsqueda

Vamos a ver estrategias que se denominan de búsqueda no informada (uninformed search) o de búsqueda ciega porque sólo usan la información de la definición del problema:

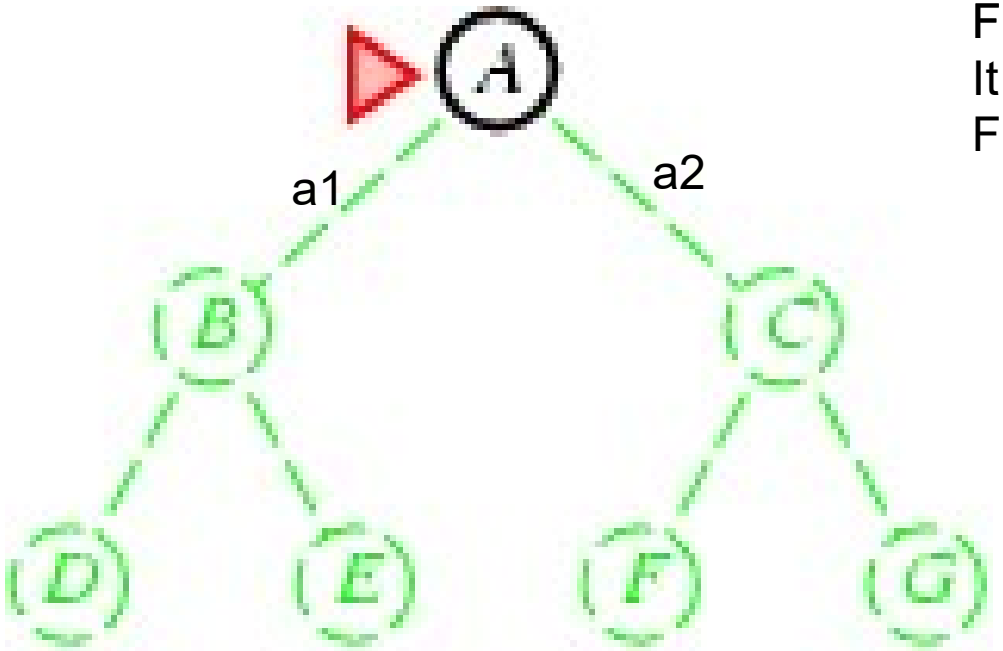
- búsqueda primero en anchura (BFS: Breadth First Search)
- búsqueda de coste uniforme
- búsqueda primero en profundidad (DFS: Depth First Search)
- búsqueda limitada en profundidad
- búsqueda por profundidad iterativa
- (búsqueda bidireccional)

Contrastan con las estrategias de búsqueda informada (informed search) o de búsqueda heurística que utilizan información del coste del estado actual al objetivo.

Búsqueda primero en anchura

- Expande el nodo no expandido cuya profundidad sea menor
- Implementación:
  - *La frontera* es una cola FIFO, es decir los nuevos sucesores se acumulan al final.

Objetivo: D

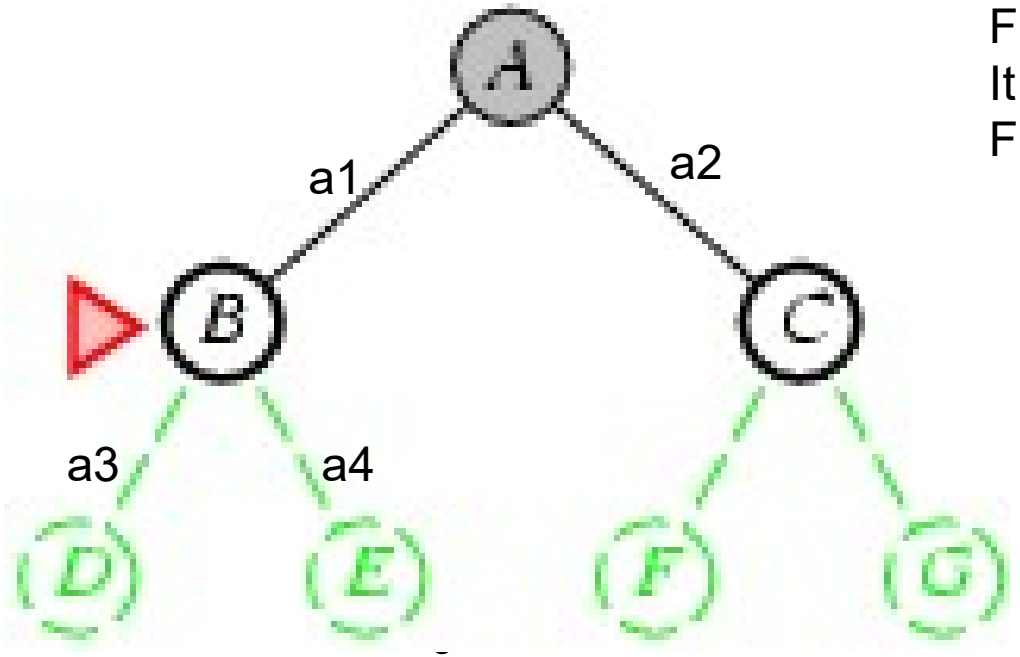


Frontera={A}
It 1: nodo=A ≠ D
Frontera={B,C}

Búsqueda primero en anchura

- Expande el nodo no expandido cuya profundidad sea menor
- Implementación:
  - *La frontera* es una cola FIFO, es decir los nuevos sucesores se acumulan al final.

Objetivo: D

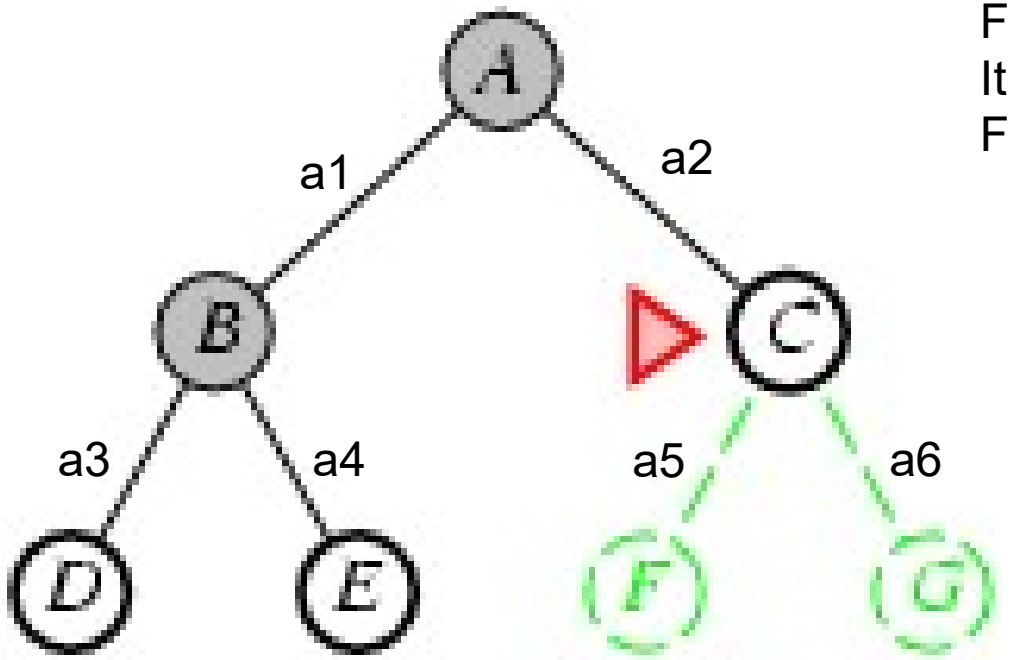


Frontera={B,C}
It 2: nodo=B ≠ D
Frontera={C, D, E}

Búsqueda primero en anchura

- Expande el nodo no expandido cuya profundidad sea menor
- Implementación:
  - *La frontera* es una cola FIFO, es decir los nuevos sucesores se acumulan al final.

Objetivo: D

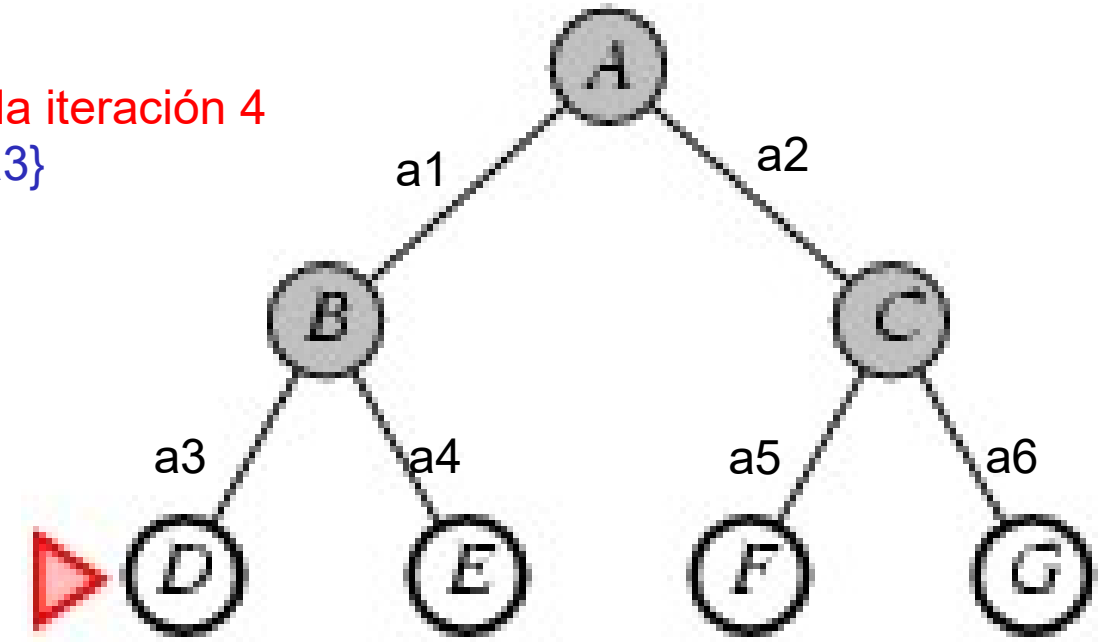


Frontera={C, D, E}
It 3: nodo=C ≠ D
Frontera={D, E,F, G}

Búsqueda primero en anchura

- Expande el nodo no expandido cuya profundidad sea menor
- Implementación:
  - *La frontera* es una cola FIFO, es decir los nuevos sucesores se acumulan al final.

Objetivo: D
Encontrado en la iteración 4
Solución: {a1, a3}



Frontera={D, E,F, G}
It 4: nodo=D
Frontera={E,F, G}

Evaluación de las Estrategias Básicas de Búsqueda

Una estrategia queda definida por el orden en que se expanden los nodos.

Vamos a evaluar las estrategias según cuatro aspectos:
Completitud
Complejidad temporal
Complejidad espacial
Optimalidad

Evaluación de las Estrategias Básicas de Búsqueda

Una estrategia queda definida por el orden en que se expanden los nodos.

Vamos a evaluar las estrategias según cuatro aspectos:
Completitud —¿garantiza encontrar una solución si existe una?
Complejidad temporal —¿cuántos nodos deben expandirse?
Complejidad espacial —¿cuántos nodos deben almacenarse en memoria?
Optimalidad —¿garantiza encontrar la mejor solución (la de menor coste) si existen varias?

Las complejidades temporal y espacial se miden en términos de:
b — máximo factor de ramificación del árbol de búsqueda
d — profundidad de la solución de menor coste
m — profundidad máxima del espacio de estados (puede ser ∞)

Propiedades de la búsqueda primero en achura

- Completa? Sí (si *b* es finita)
- Tiempo? $b+b^2+b^3+\ldots+b^d + (b^{d+1}-b) = O(b^{d+1})$
- Espacio? $O(b^{d+1})$ (mantiene cada nodo en memoria)
- Óptima? Sí (si el coste es igual para todas las acciones)

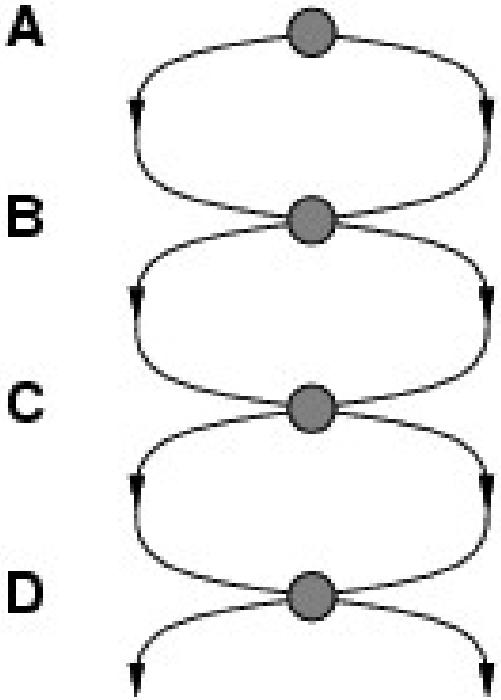
- El espacio representa un problema mayor que el tiempo

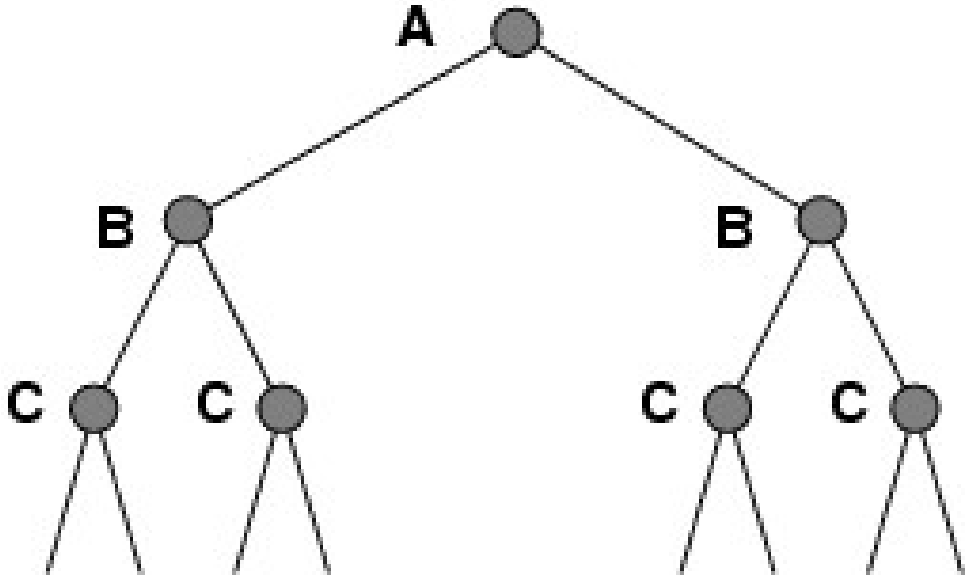
Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-firstsearch. The numbers shown assume branching factor b = 10; 10,000 nodes/second; 1000 bytes/node.

Estados repetidos

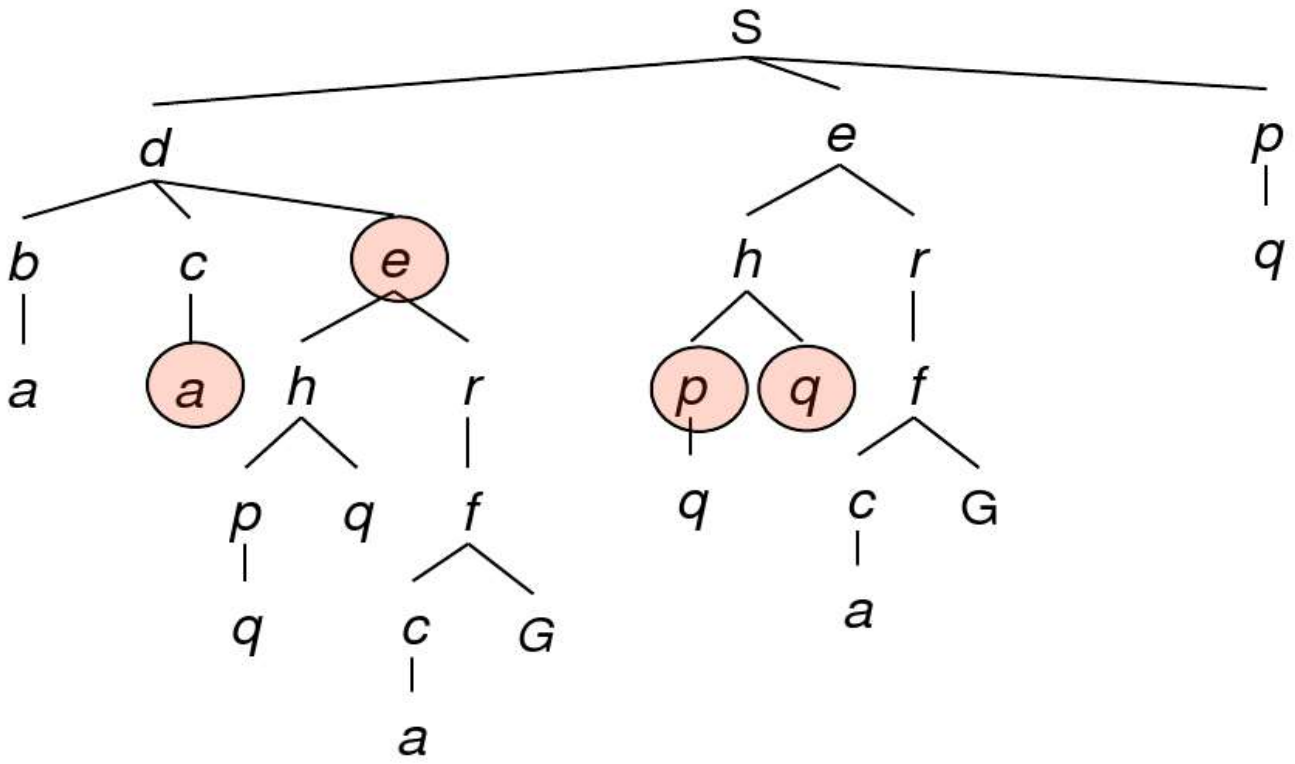
- Si no detectamos estados repetidos podemos convertir un problema lineal en un problema exponencial!!!





Búsqueda en grafos

- En una búsqueda primero en anchura no sería necesario expandir los nodos coloreados ¿por qué?

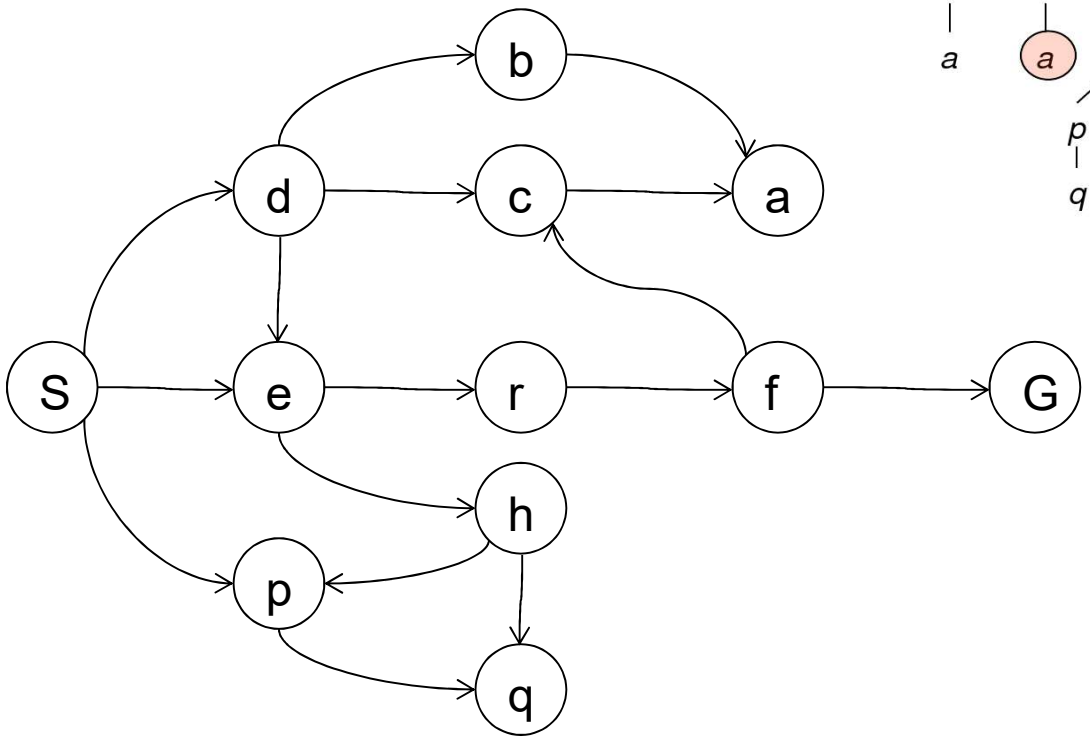


Búsqueda en grafos

- Qué espacio de estados puede estar explorando este BFS?

Búsqueda en grafos

- Qué espacio de estados puede estar explorando este BFS?



Algoritmos de búsqueda en grafos

Idea básica:

- Mecanismo de exploración igual que en árboles
- Evitar repeticiones mediante una lista de nodos cerrados (ya explorados)

función BÚSQUEDA-GRAFOS(*problema,frontera*) **devuelve** una solución o fallo

cerrados ← un conjunto vacío

frontera ← INSERTA(HACER-NODO(ESTADO-INICIAL[*problema*]), *frontera*)

bucle hacer

si VACIA? (*frontera*) **entonces devolver** fallo

nodo ←SACAR-BORRANDO-PRIMERO(*frontera*)

si TEST-OBJETIVO[*problema*] aplicado al ESTADO[*nodo*] es cierto

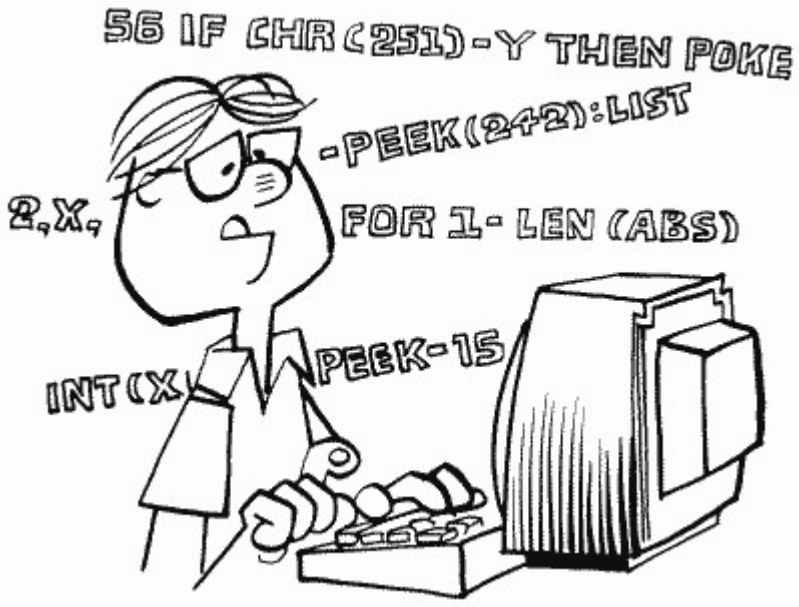
entonces devolver SOLUCION(*nodo*)

si ESTADO[*nodo*] no está en *cerrados* **entonces**

añadir ESTADO[*nodo*] a *cerrados*

frontera ← INSERTAR-TODO(EXPANDIR(*nodo, problema*), *frontera*)

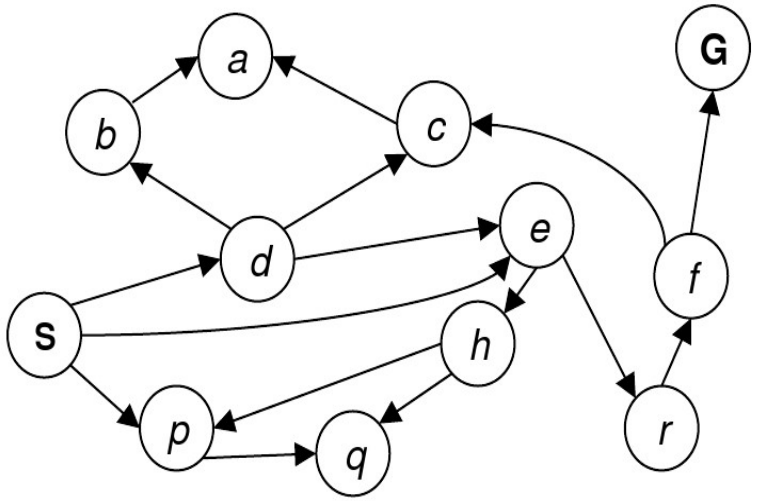
Detalles de implementación



- Utilizad un dict o set para implementar la lista de estados cerrados

Grafo del espacio de estados

- Todo problema de búsqueda tiene asociado un grafo de estados.
- La función sucesor se representa por arcos
- En contadas ocasiones se puede construir este grafo en memoria



Estrategias Básicas de Búsqueda

Vamos a ver estrategias que se denominan de búsqueda no informada (uninformed search) o de búsqueda ciega porque sólo usan la información de la definición del problema:

- búsqueda primero en anchura (BFS: Breadth First Search)
- búsqueda de coste uniforme
- búsqueda primero en profundidad (DFS: Depth First Search)
- búsqueda limitada en profundidad
- búsqueda por profundidad iterativa
- (búsqueda bidireccional)

Contrastan con las estrategias de búsqueda informada (informed search) o de búsqueda heurística que utilizan información del coste del estado actual al objetivo.

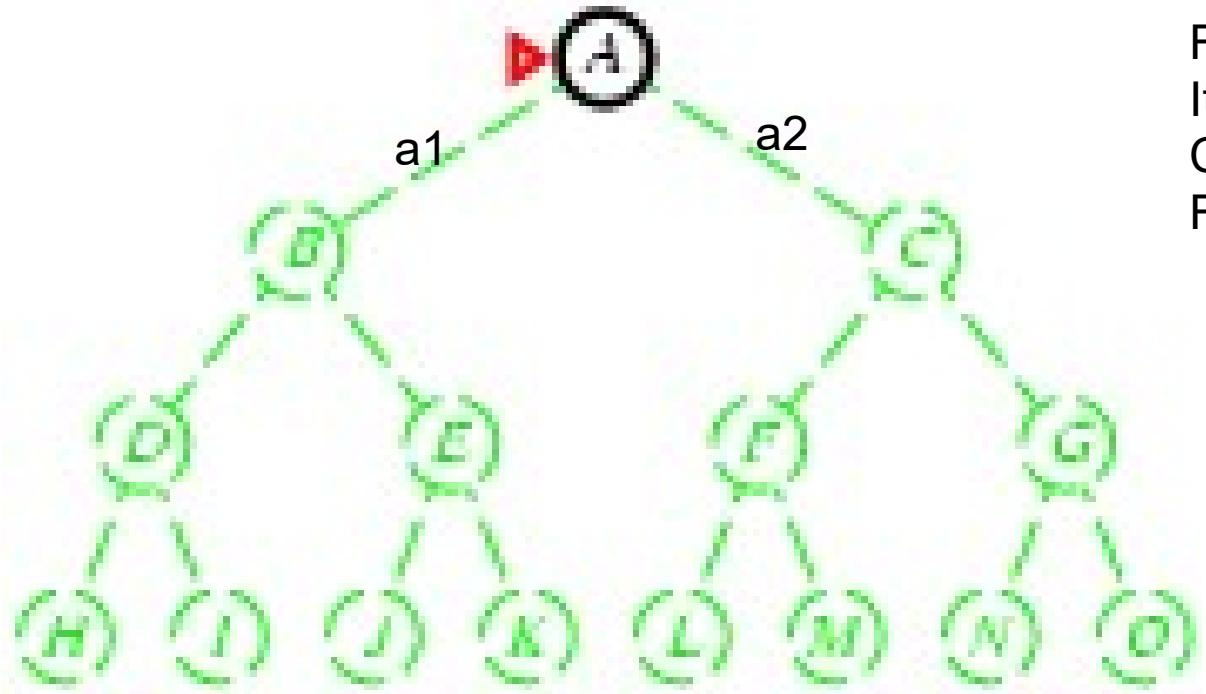
Búsqueda de coste uniforme

- Expande el nodo no expandido cuyo coste sea menor
- Implementación:
  - La *frontera* es un cola ordenada por coste de camino
- Equivalente a la búsqueda primero en anchura si el coste de todos los pasos es igual.
- Completo? Sí, si el coste de cada paso es ≥ ε >0
- Tiempo? # de nodos con $g \leq$ coste de la solución óptima, $O(b^{ceiling(C^*/\varepsilon)})$ donde C^* es el coste de la solución óptima
- Espacio? # de nodos con $g \leq$ coste de la solución óptima $O(b^{ceiling(C^*/\varepsilon)})$
- Óptima? Sí. Los nodos se expanden en orden creciente de g(n)

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- Implementación:
  - La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio

Objetivo: M

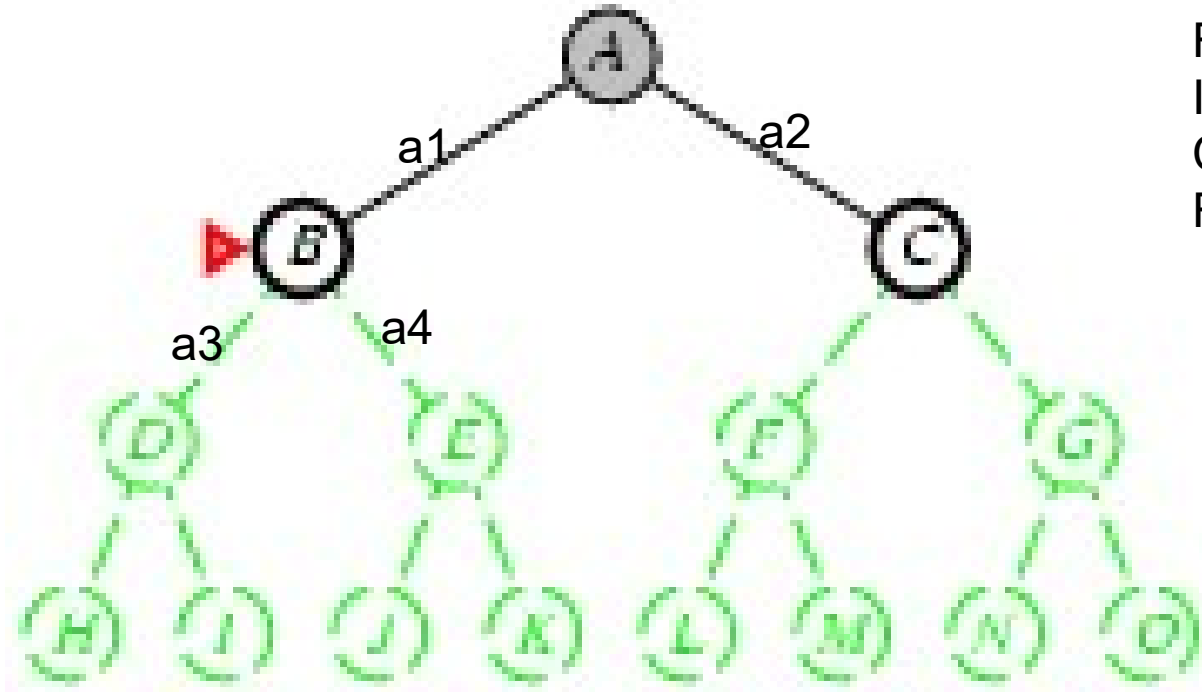


Frontera={ A}
It1: nodo=A ≠ M
Cerrados: {A}
Frontera={B, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- Implementación:
  - La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio

Objetivo: M

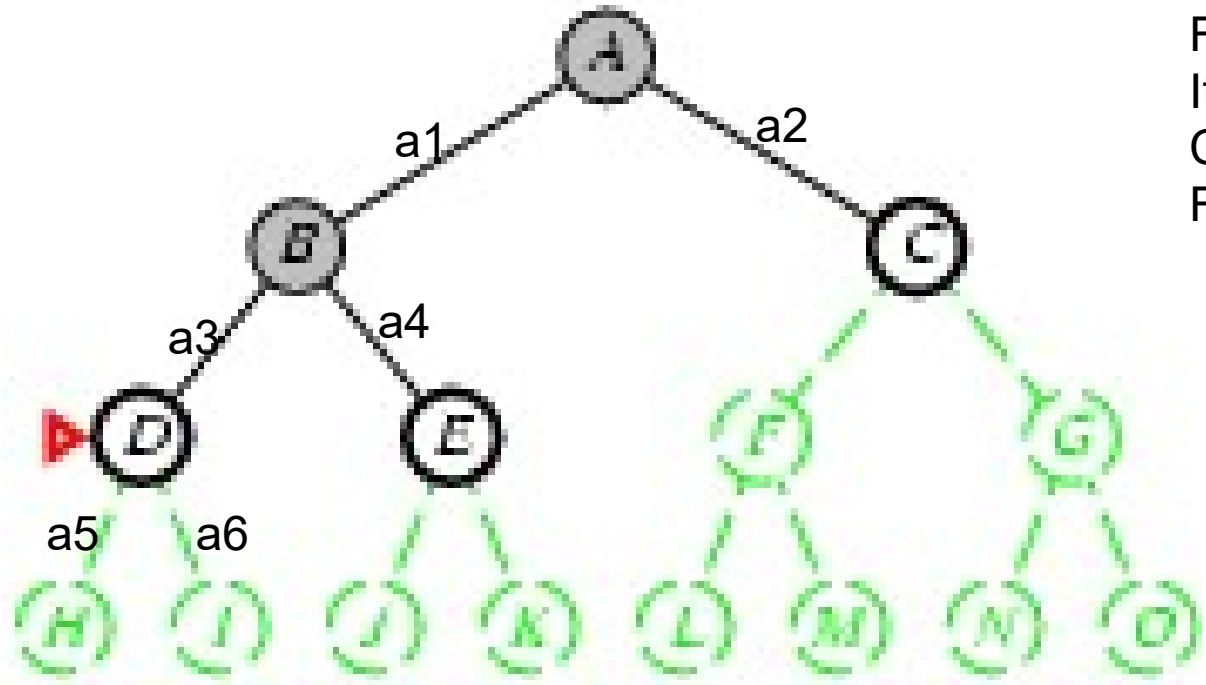


Frontera={B, C}
It 2: nodo=B ≠ M
Cerrados: {A, B}
Frontera={ D, E, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- Implementación:
  - La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio

Objetivo: M

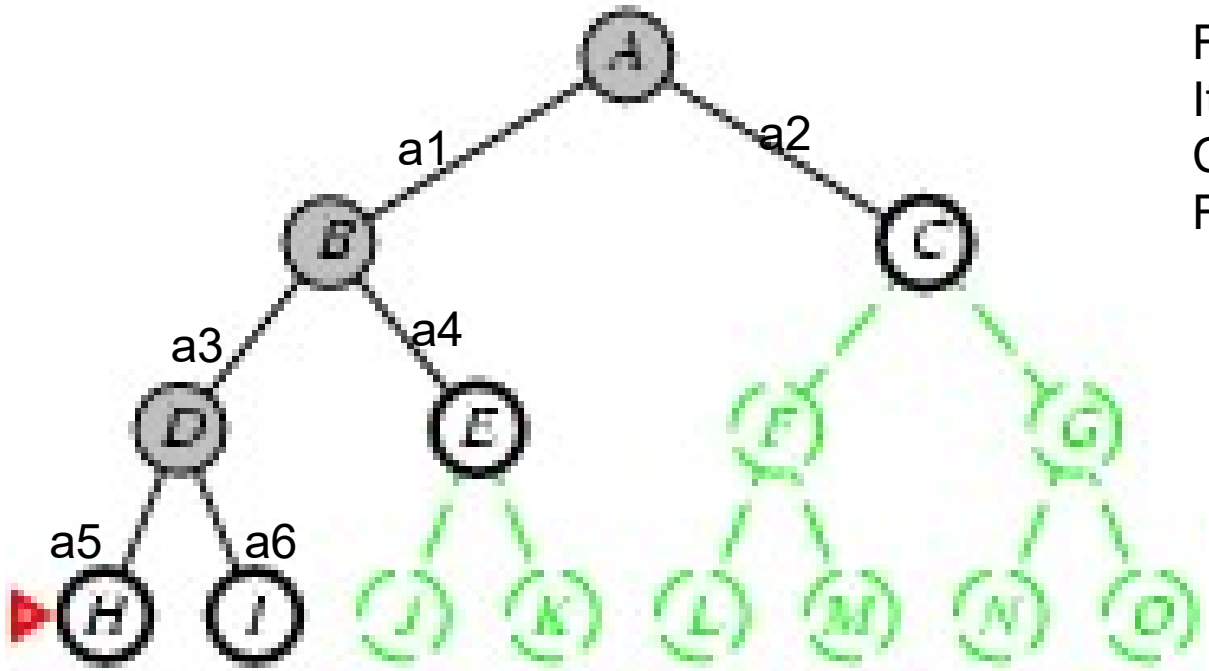


Frontera={ D, E, C}
It3: nodo=D ≠ M
Cerrados: {A, B, D}
Frontera={ H, I, E, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- Implementación:
  - La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio

Objetivo: M

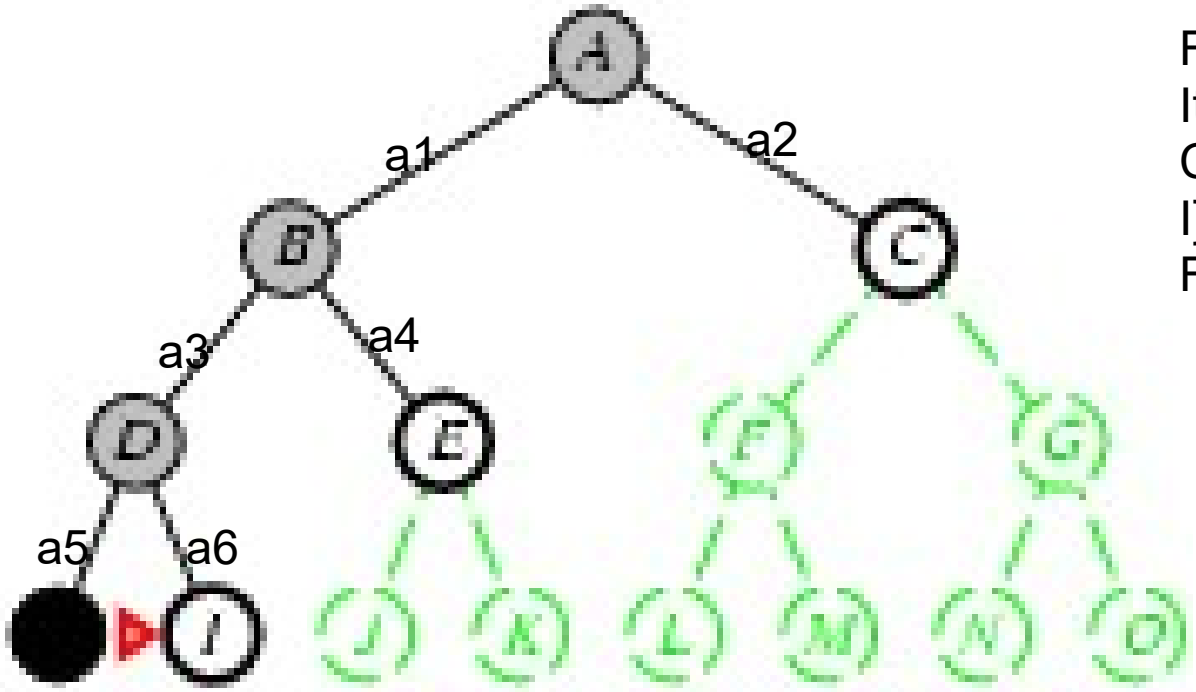


Frontera={ H, I, E, C}
It 4: nodo=H ≠ M
Cerrados: {A, B, D, H}
Frontera={ I, E, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- Implementación:
  - La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio

Objetivo: M

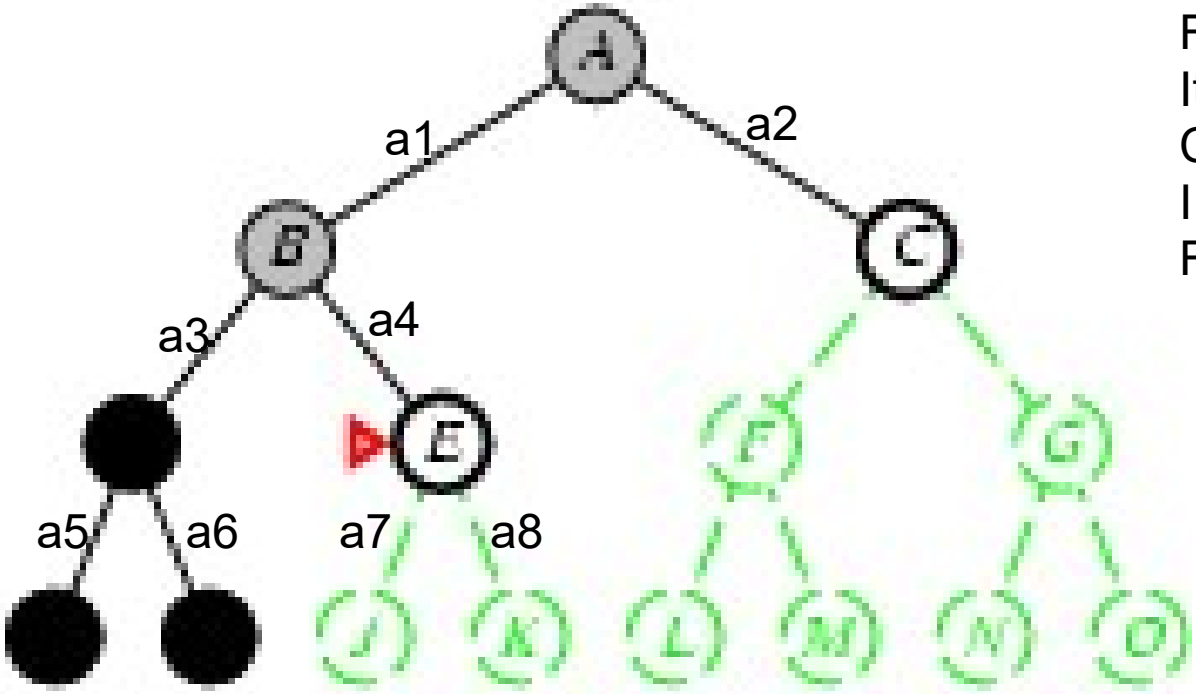


Frontera={ I, E, C}
It 5: nodo=I ≠ M
Cerrados: {A, B, D, H, I}
Frontera={ E, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M

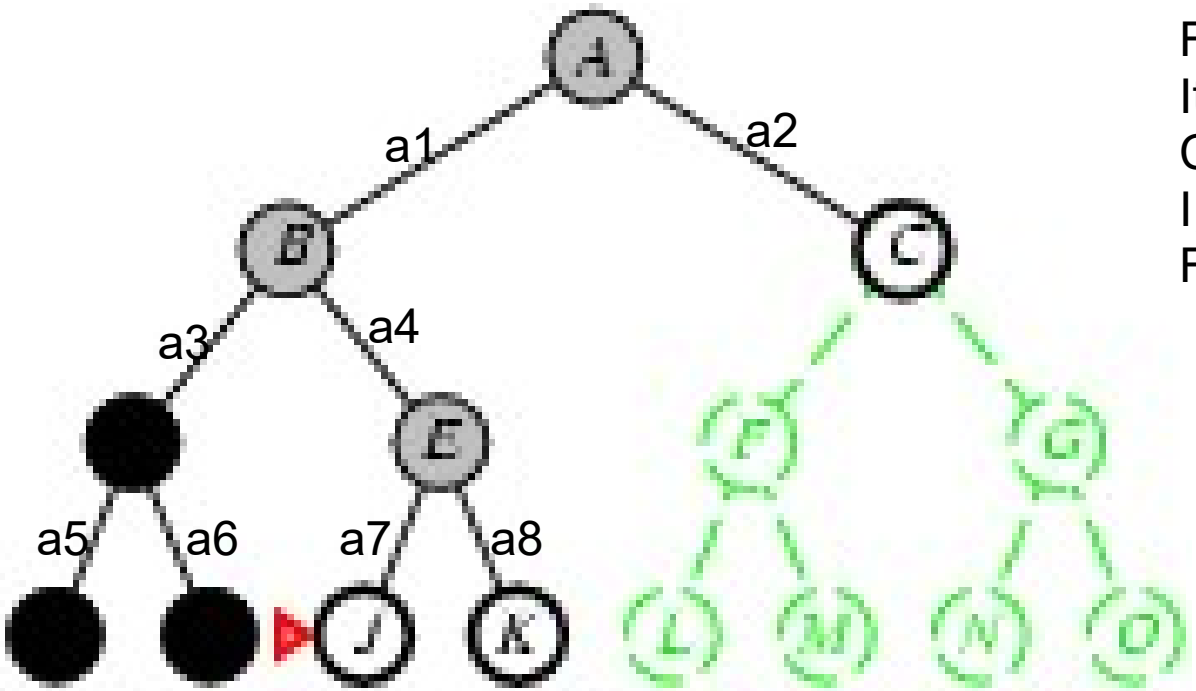


Frontera={ E, C}
It6: nodo=E ≠ M
Cerrados: {A, B, D, H, I, E}
Frontera={ J, K, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M

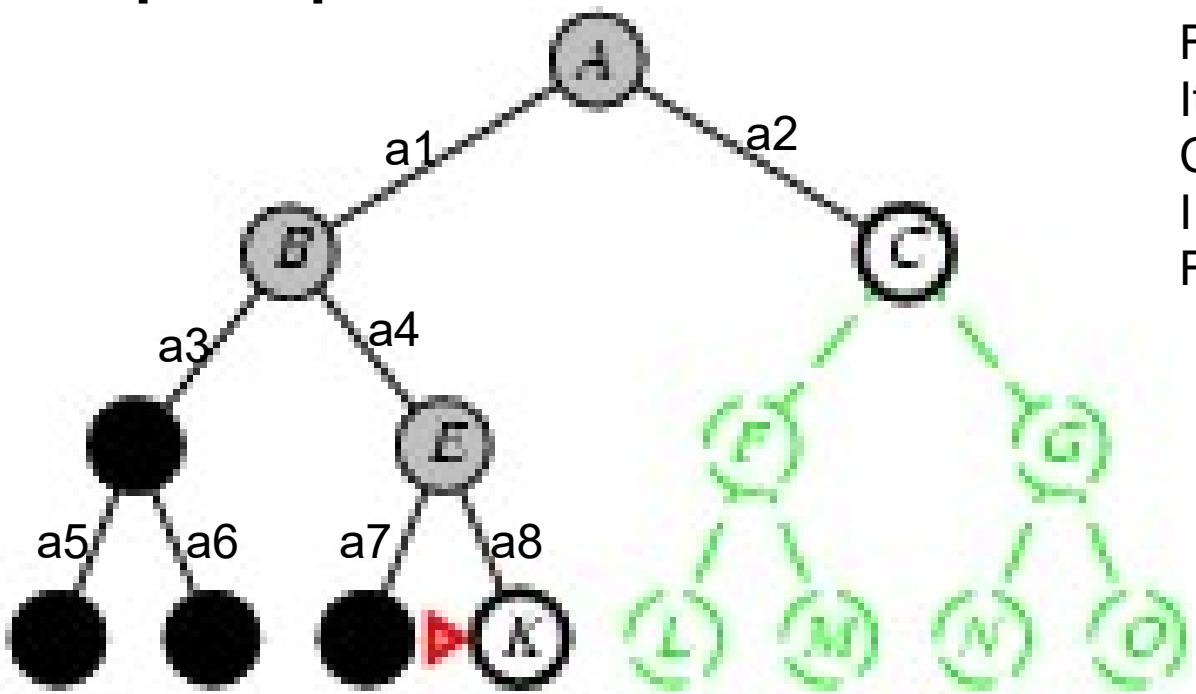


Frontera={ J, K, C}
It7: Nodo=J ≠ M
Cerrados: {A, B, D, H, I, E, J}
Frontera={K, C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M

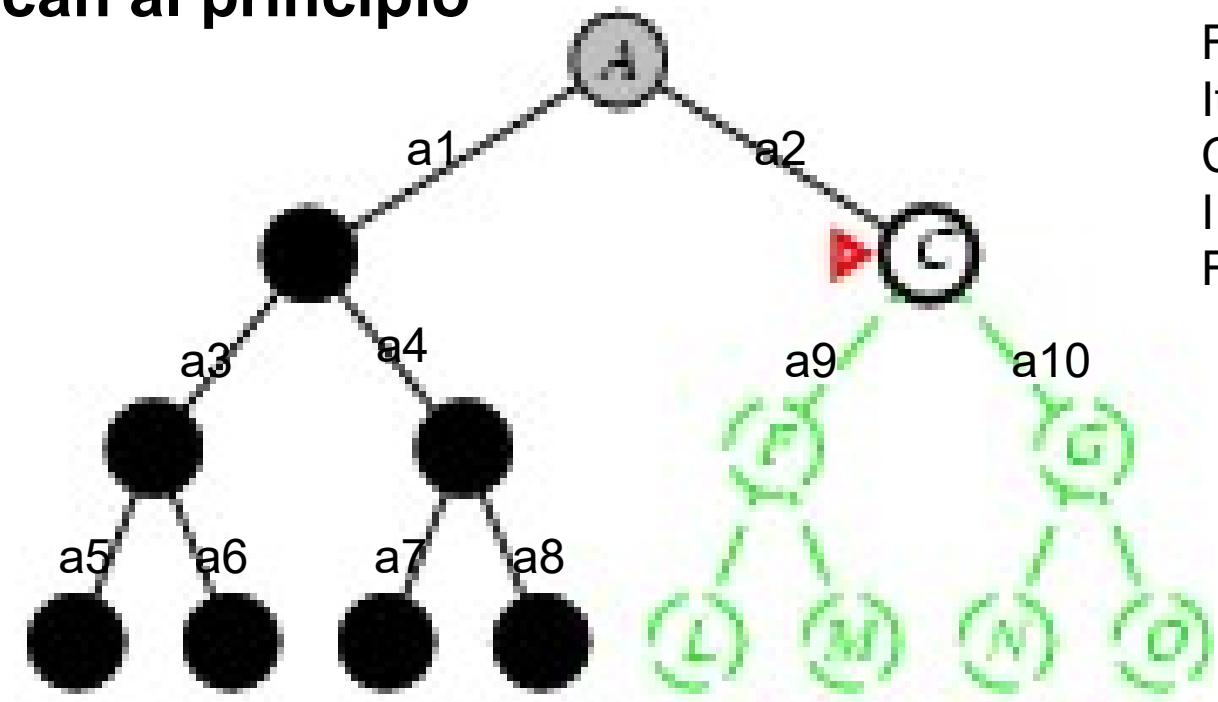


Frontera={K, C}
It 8: nodo=K ≠ M
Cerrados: {A, B, D, H, I, E, J, K}
Frontera={C}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M

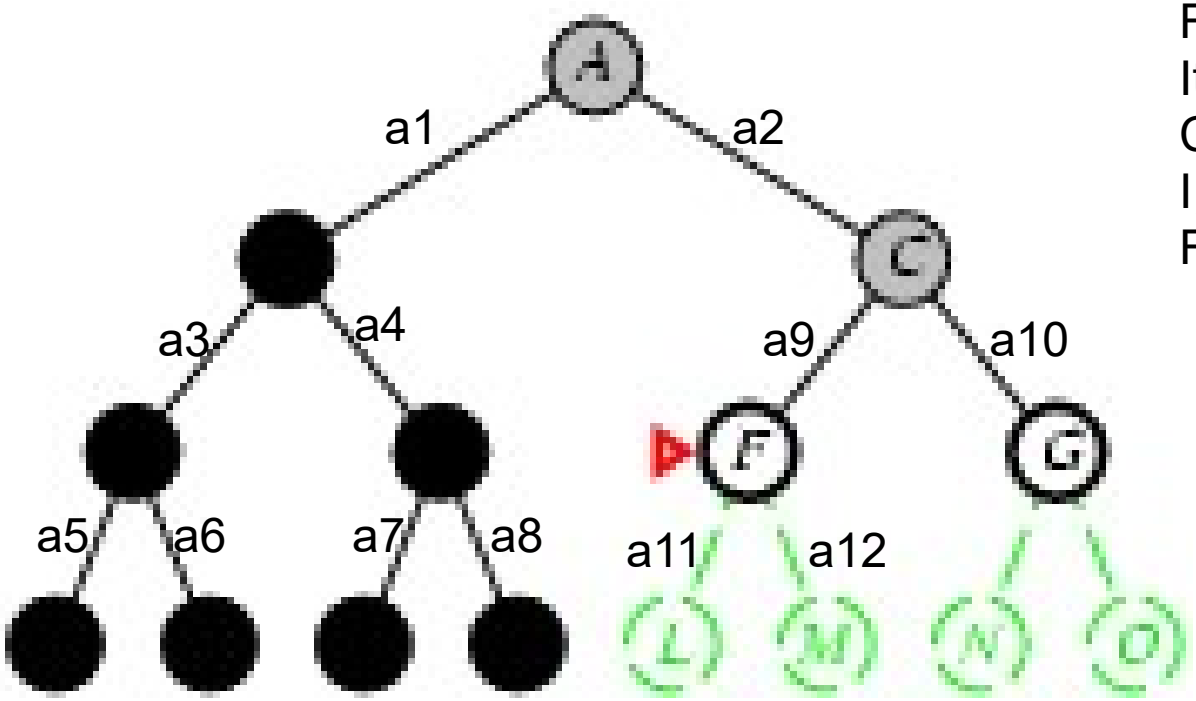


Frontera={C}
It 9: nodo=C ≠ M
Cerrados: {A, B, D, H, I, E, J, K, C}
Frontera={F, G}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M

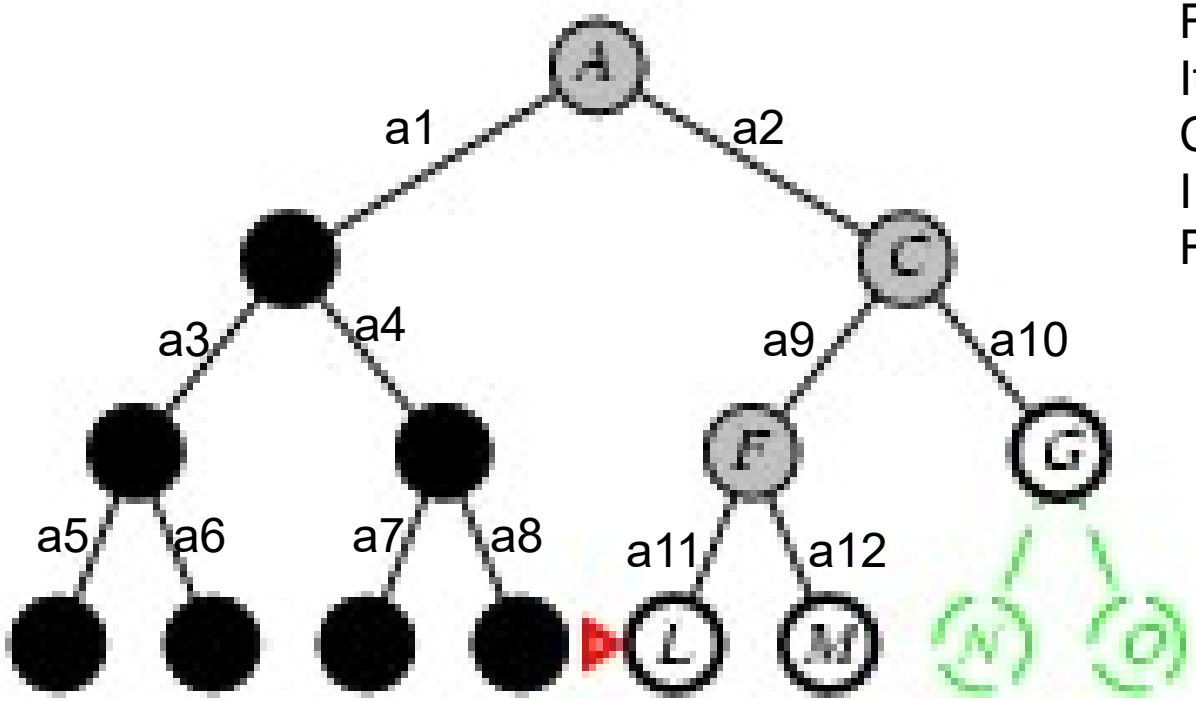


Frontera={F, G}
It 10: nodo=F ≠ M
Cerrados: {A, B, D, H, I, E, J, K, C, F}
Frontera={L, M, G}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M

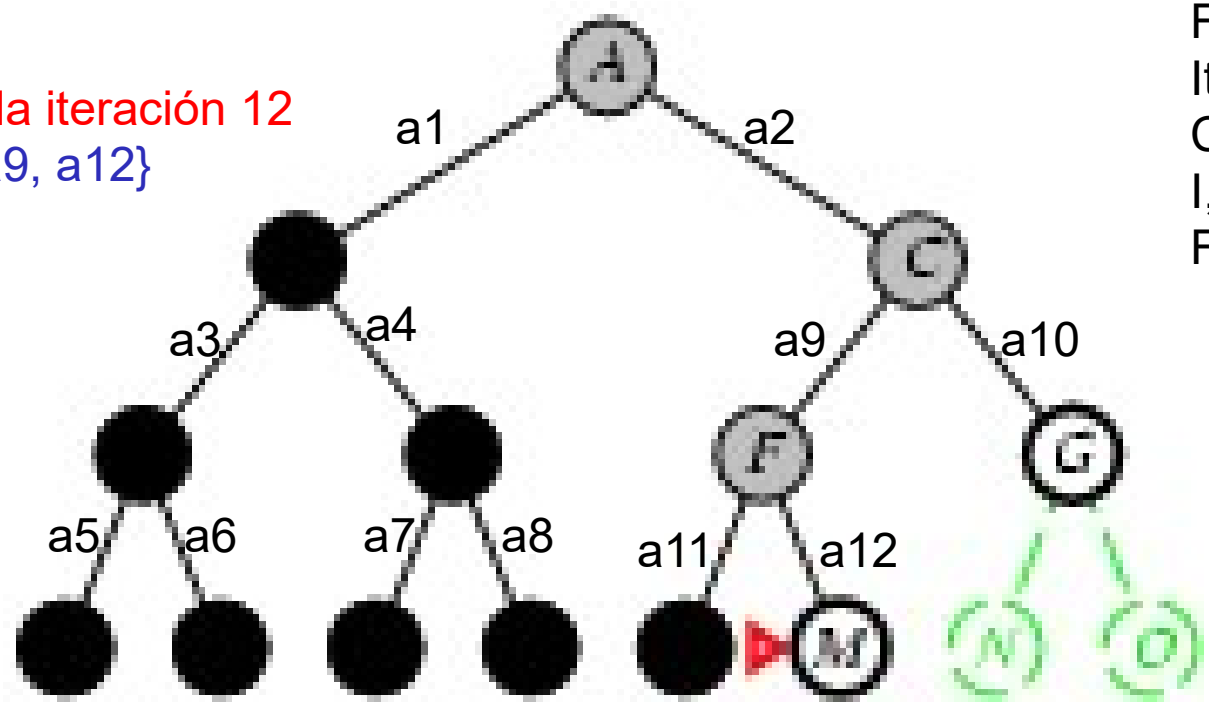


Frontera={L, M, G}
It 11: nodo=L ≠ M
Cerrados: {A, B, D, H, I, E, J, K, C, F, L}
Frontera={M, G}

Búsqueda primero en profundidad

- Se expande el nodo más profundo no expandido aún
- **Implementación:**
  - **La *frontera* es una cola LIFO: los nuevos sucesores se colocan al principio**

Objetivo: M
Encontrado en la iteración 12
Solución: {a2, a9, a12}



Frontera={M, G}
It 12: nodo= M
Cerrados: {A, B, D, H, I, E, J, K, C, F, L}
Frontera={G}

Propiedades de la búsqueda primero en profundidad

UNIVERSITAT DE BARCELONA

- Completa? No: falla en espacios de profundidad infinita, espacios con ciclos.
  - Si la modificamos para evitar estados repetidos en el camino actual → completa en espacios finitos
- Tiempo? $O(b^m)$: horrible si m es mucho mayor que d
  - Pero si las soluciones son densas, puede ser mucho más rápida que la búsqueda primero en anchura
- Espacio? $O(bm)$, es decir: espacio lineal!
- Óptima? No

Búsqueda de profundidad limitada

- Igual que la búsqueda primero en profundidad pero con un límite de profundidad l

Es decir: los nodos a profundidad l no tienen sucesores.

Búsqueda primero en profundidad con profundidad iterativa

UNIVERSITAT DE BARCELONA

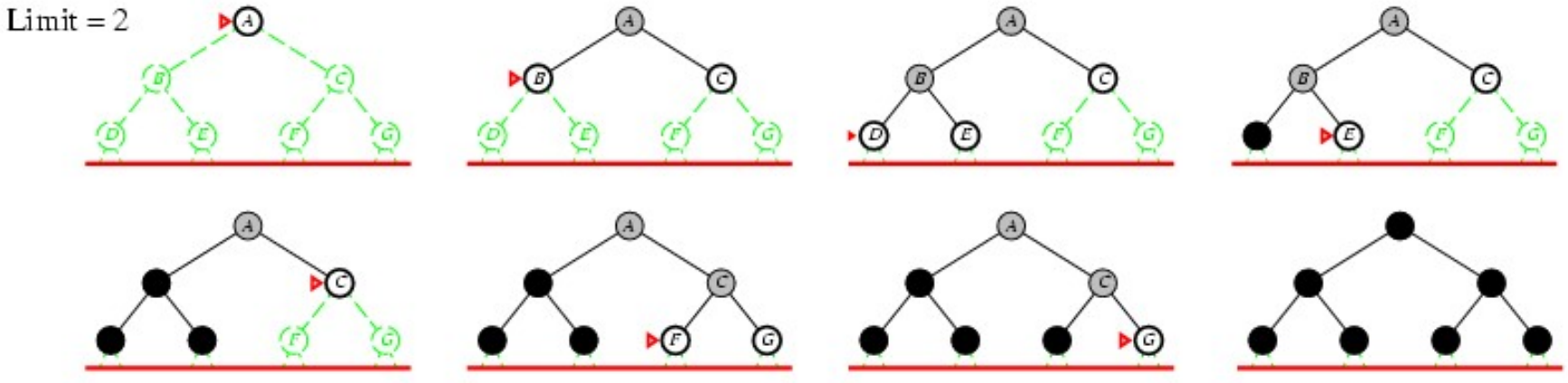
- Repite una búsqueda de profundidad limitada con
  - *l=0*
  - *l=1*
  - *l=2*
  - *...*

Búsqueda con profundidad iterativa l =0

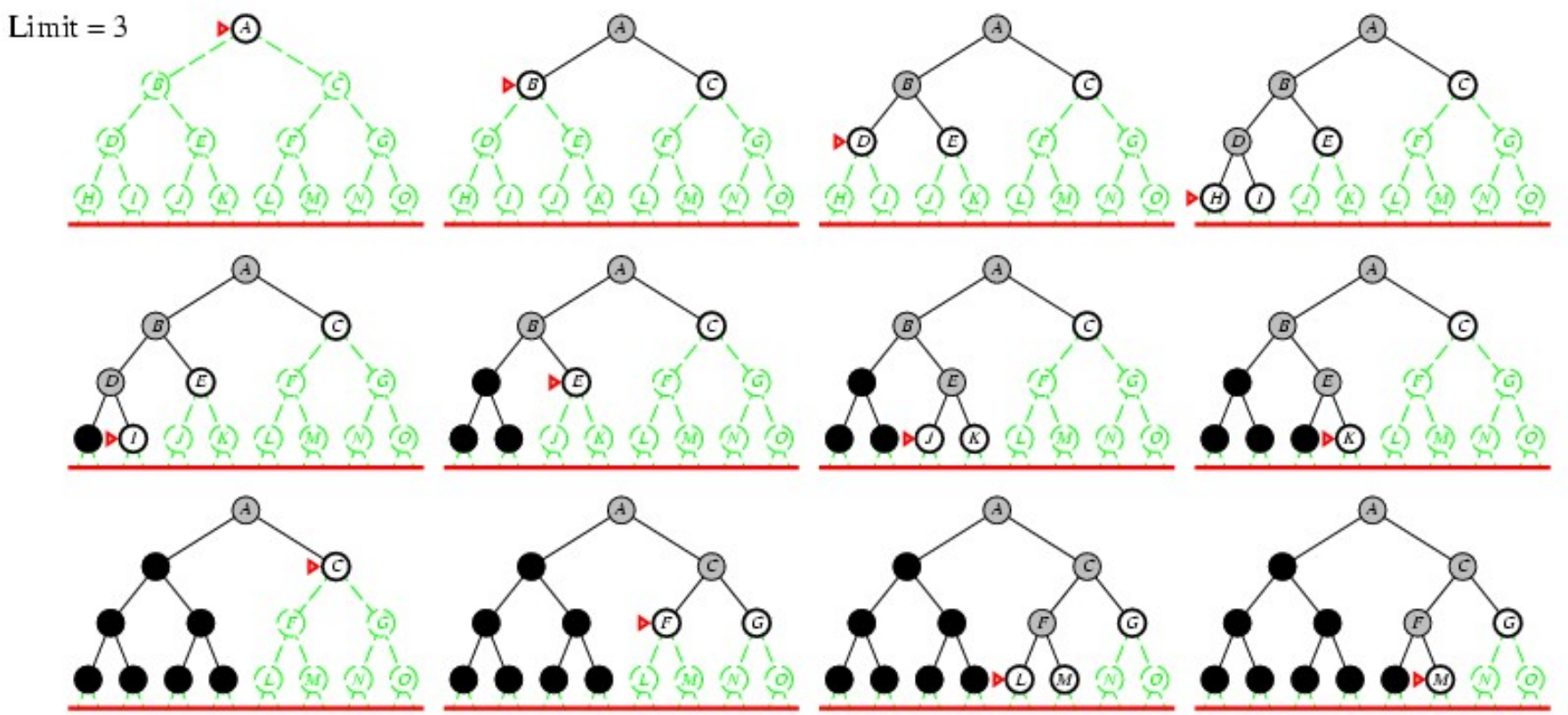
Limit = 0

Búsqueda con profundidad iterativa l =1

Búsqueda con profundidad iterativa l =2



Búsqueda con profundidad iterativa l =3



Búsqueda con profundidad iterativa

- Número de nodos generados por la búsqueda con profundidad **limitada** hasta profundidad *d* con factor de ramificación *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Número de nodos generados por la búsqueda con profundidad **iterativa** hasta profundidad *d* con factor de ramificación *b*:

$$N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- Para *b = 10*, *d = 5*,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$

- Sobrecarga = (123,456 - 111,111)/111,111 = 11%

Propiedades de la búsqueda en profundidad iterativa

- Completa? Sí
- Tiempo? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Espacio? $O(bd)$
- Óptima? Sí, (si el coste de todos los pasos es homogéneo).

Resumen de algoritmos

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Algoritmos de búsqueda en grafos

- Nomenclatura:
  - Espacio de estados: el espacio conceptual en el que se realiza la búsqueda.
  - Árbol/grafo de exploración: el que se va creando al realizar efectivamente la búsqueda (representa una parte del anterior)
  - Frontera: lista de los nodos visitados que aún no han sido expandidos
  - Nodos visitados: nodos que se han construido y se han metido en la frontera
  - Nodos expandidos: nodos sacados de la frontera para los que se ha obtenido sus sucesores
  - Cerrados: lista de nodos que han sido expandidos

Video

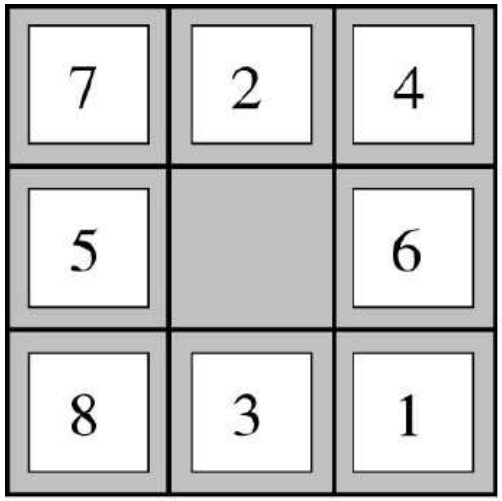
- TFC de planificación de rutas metro de madrid:
  - http://poiritem.wordpress.com/2009/10/26/1-introduccion-puesta-en-escena/

- https://www.youtube.com/watch?v=s8e6a-iMuQE comparativa dos búsquedas informadas en un laberinto

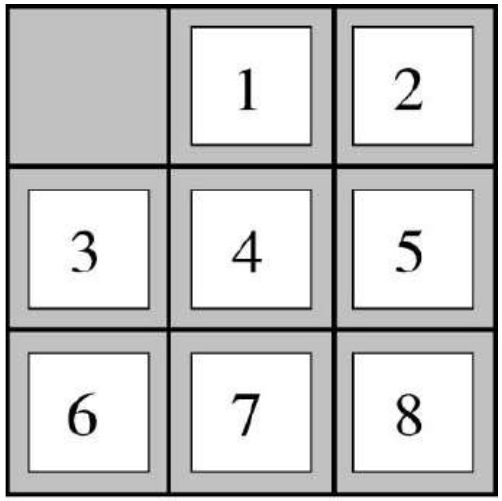
Más ejemplos

- Más ejemplos de problemas (apartado 3.2 libro)
  - "Toy problems":
    - El puzle de 8 piezas
    - http://en.wikipedia.org/wiki/Eight_queens_puzzle
  - Problemas reales
    - Planificar rutas
    - Problema del viajante de comercio
    - Diseño de circuitos integrados (VLSI layout)
    - Navegación de robots en espacios continuos
    - Ensamblaje de objetos complejos
    - Diseño de proteínas
    - Búsqueda en Internet

Ejemplo: el 8-puzle http://www.8puzzle.com/



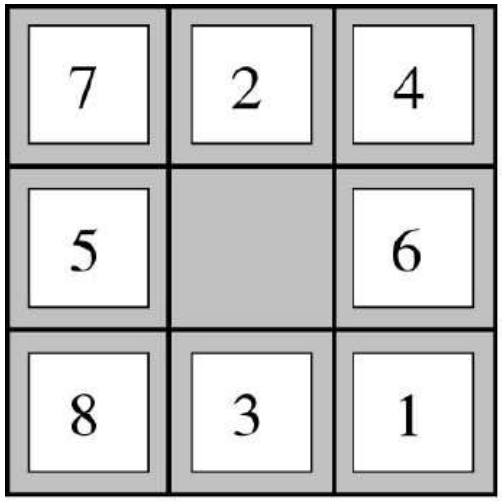
Estado Inicial



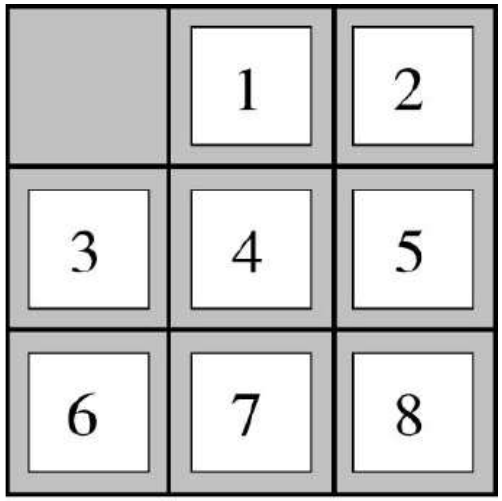
Estado Objetivo

¿Estados?
¿Acciones?
¿Test objetivo?
¿Coste del camino?

Ejemplo: el 8-puzle



Estado Inicial



Estado Objetivo

¿Estados? Localizaciones completas de las piezas (ignorar las posiciones intermedias)

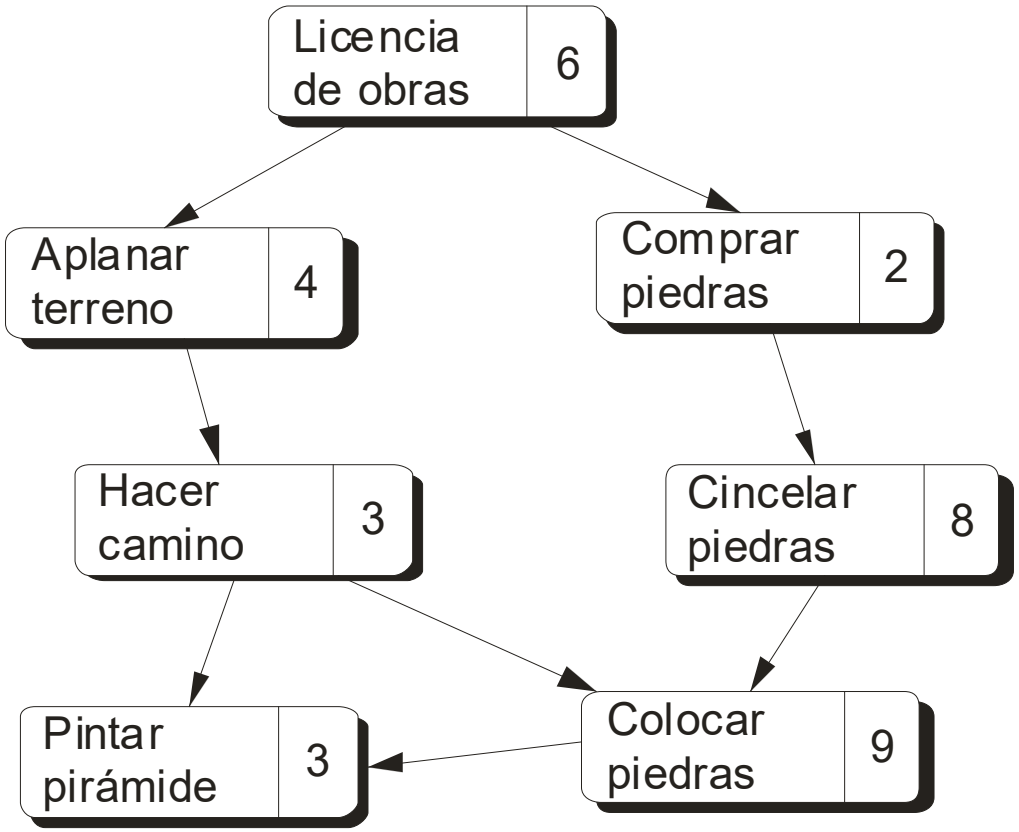
¿Nº estados?

¿Acciones? Mover el negro a la izquierda, derecha, arriba, abajo (ignorar los atascos, etc.)

¿Test objetivo? = estado objetivo (proporcionado)

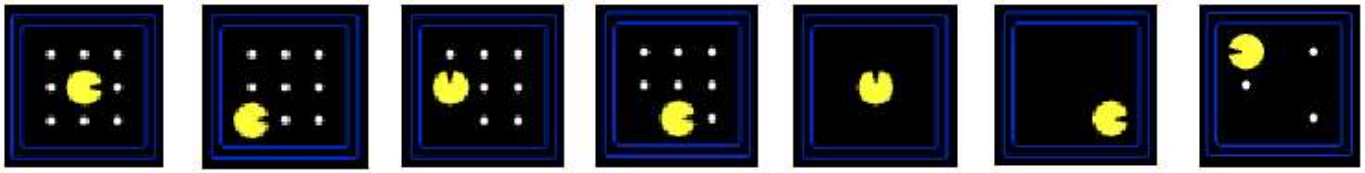
¿Coste del camino? 1 por movimiento

Ej.: planificación de tareas (ED)

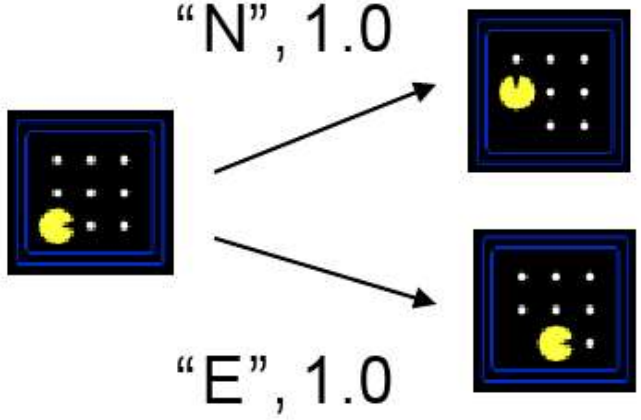


Problemas de búsqueda

- Un problema de búsqueda consiste en:
  - Un **espacio de estados**:



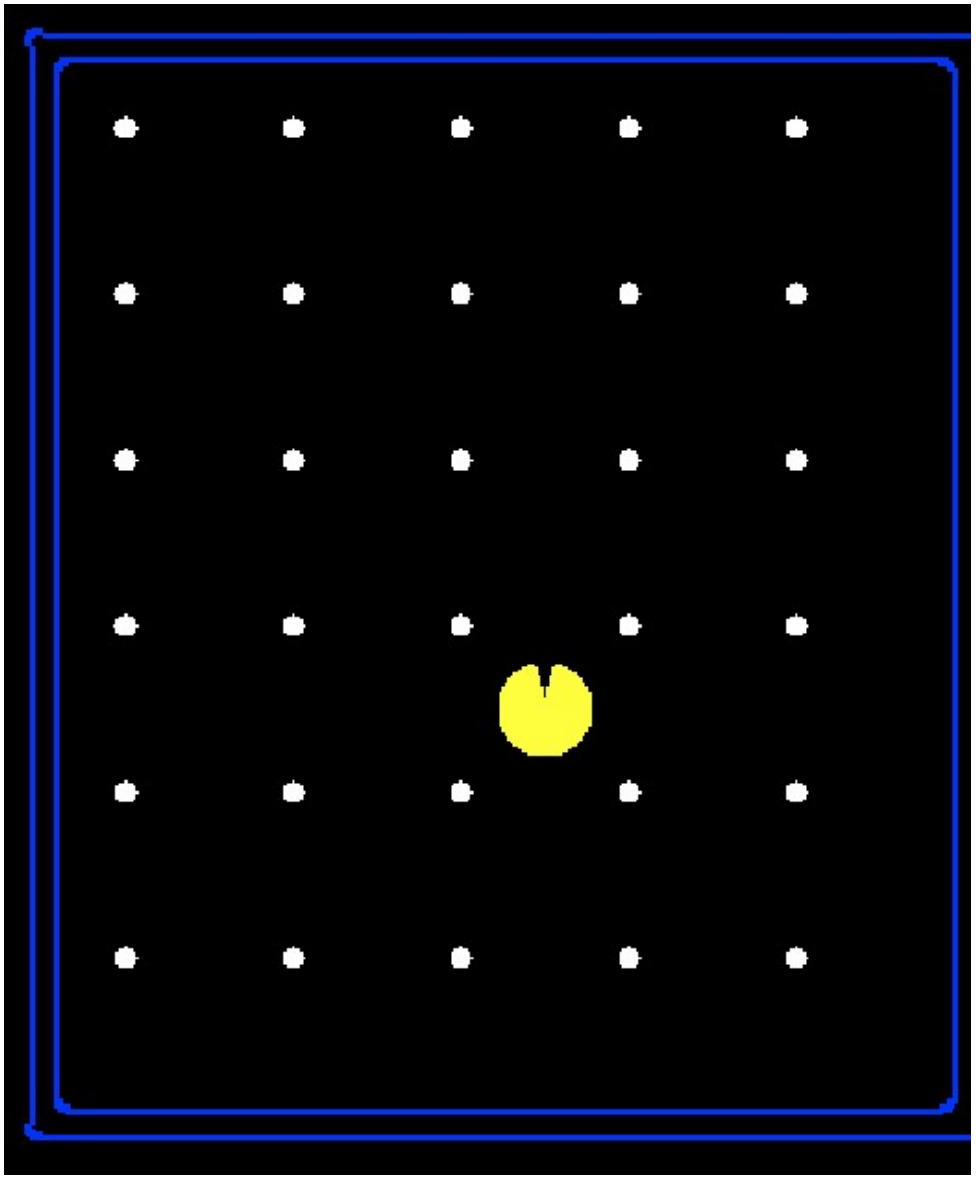
  - Una función **sucesor**:
  - Un **estado inicial** y un test que nos permita saber si hemos llegado al **objetivo**.



- Una **solución** es una secuencia de acciones que transforma un estado inicial en un estado objetivo.

¿Cómo de grande es el espacio de búsqueda?

- Problema de búsqueda:
  - Comer toda la comida
- Posiciones posibles de Pacman:
  - 10x12
- Número de copos:
  - 30



Tipos de Problemas

Deterministas, completamente observables ⇒ problemas de estado simple
El agente sabe exactamente en qué estado estará; la solución es una secuencia.

No observables ⇒ problemas conformados (conformant or sensorless)
El agente puede no saber dónde está; la solución (si la hay) es una secuencia.

No deterministas y/o parcialmente observables ⇒ problemas de contingencia
Las percepciones del agente proporcionan información nueva sobre el estado actual.
La solución es un árbol o una política.
A menudo intercalan la búsqueda y la ejecución.

Problemas de espacio-estado desconocido ⇒ problemas de exploración ("en línea").

Ejemplo: el mundo de la aspiradora

Estado simple, estado inicial 5.

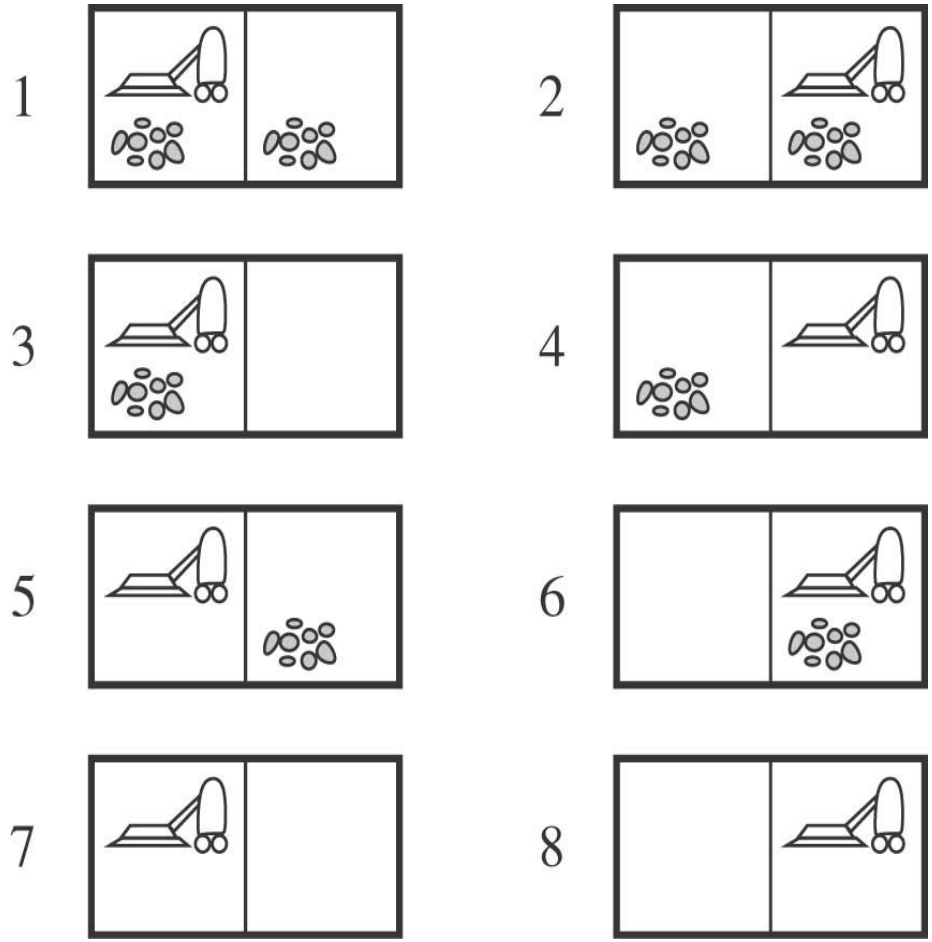
¿Solución?

Sec= [*Derecha*, *Aspirar*] (5 -> 6 ->8)

No observable, estado inicial
{1, 2, 3, 4, 5, 6, 7, 8}

Por ejemplo:
Derecha produce {2, 4, 6, 8}.

¿Solución?



Ejemplo: el mundo de la aspiradora

Estado simple, estado inicial 5.

¿Solución?

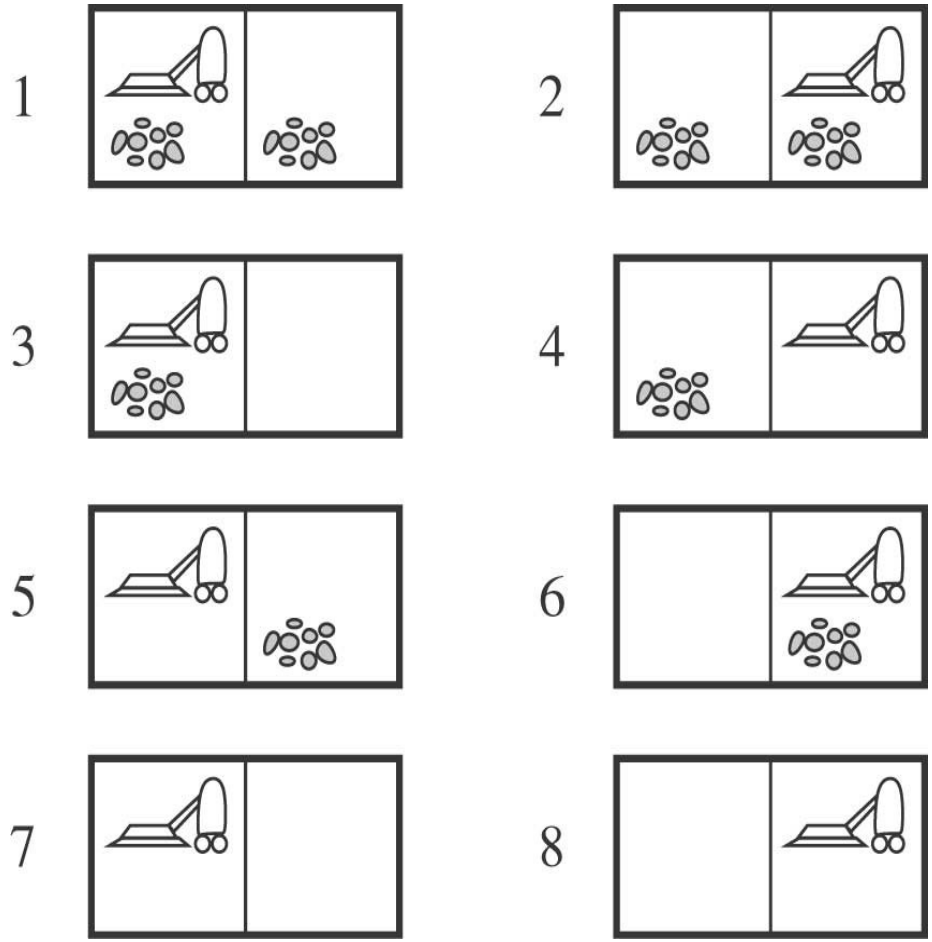
Sec= [[*Derecha*, *Aspirar*] (5 -> 6 ->8)

No observable, estado inicial
{1, 2, 3, 4, 5, 6, 7, 8}

Por ejemplo:
Derecha produce {2, 4, 6, 8}.

¿Solución?

Sec= [*Derecha, Aspirar, Izquierda, Aspirar*]

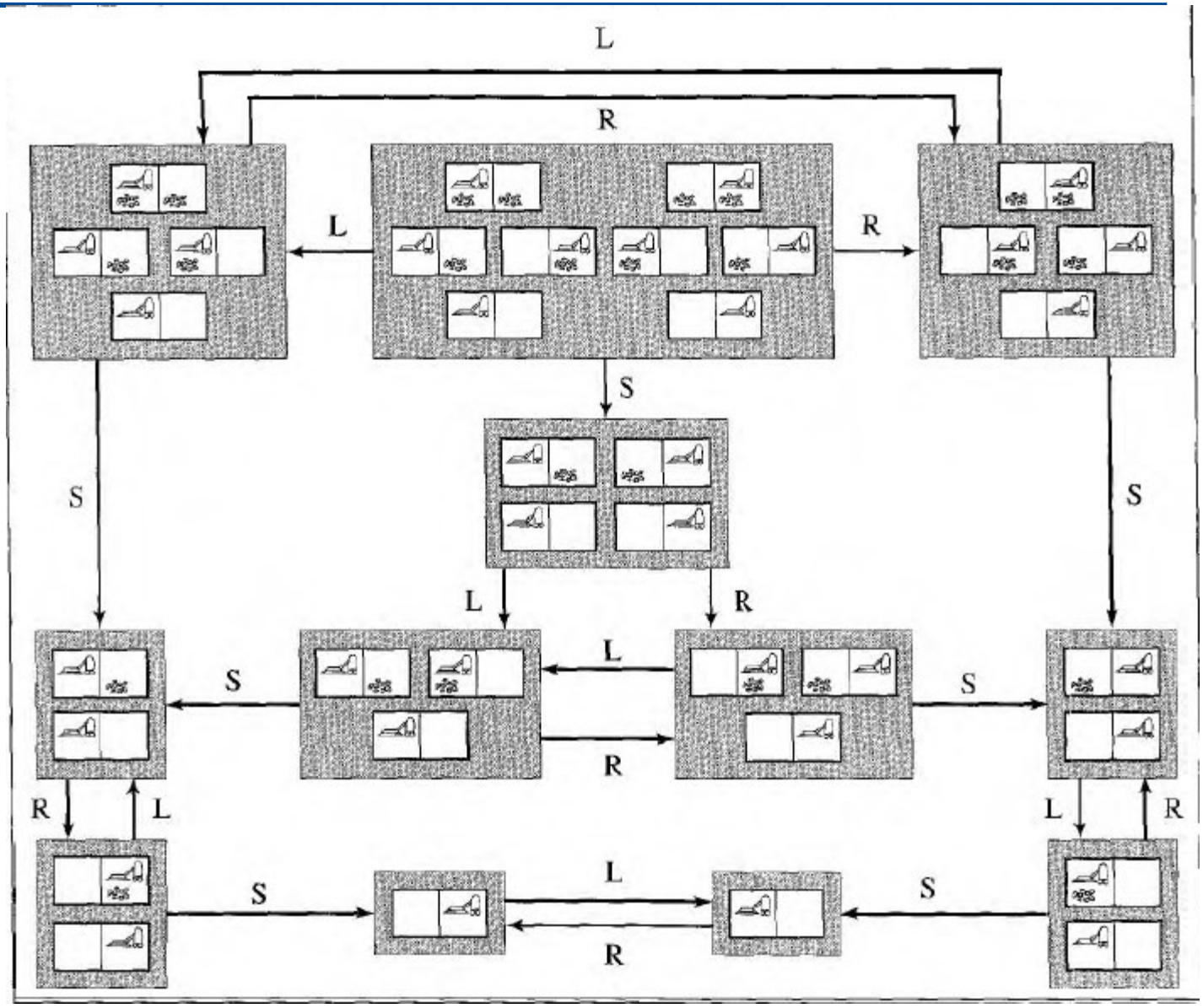


Ejemplo: el mundo de la aspiradora

- Problema

  No observable :

  Espacio de estados



Ejemplo: el mundo de la aspiradora

Estado simple, estado inicial 5.

¿Solución?

Sec= [*Derecha*, *Aspirar*] (5 -> 6 ->8)

No observable, estado inicial
{1, 2, 3, 4, 5, 6, 7, 8}

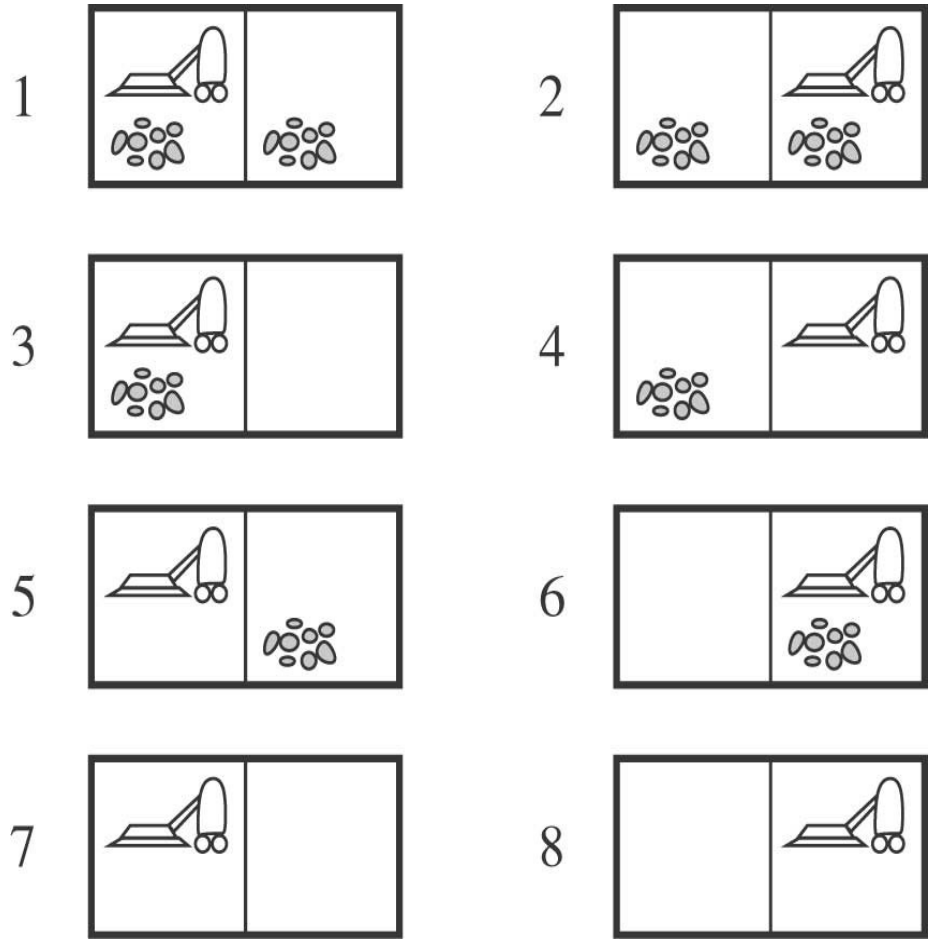
Por ejemplo:
Derecha produce {2, 4, 6, 8}.

¿Solución?

Sec= [*Derecha, Aspirar, Izquierda, Aspirar*]

Contingencia, estado inicial 5.
Ley de Murphy: Aspirar a veces deposita suciedad en la alfombra.
Sensores locales: suciedad, sólo en el lugar.

¿Solución?



Ejemplo: el mundo de la aspiradora

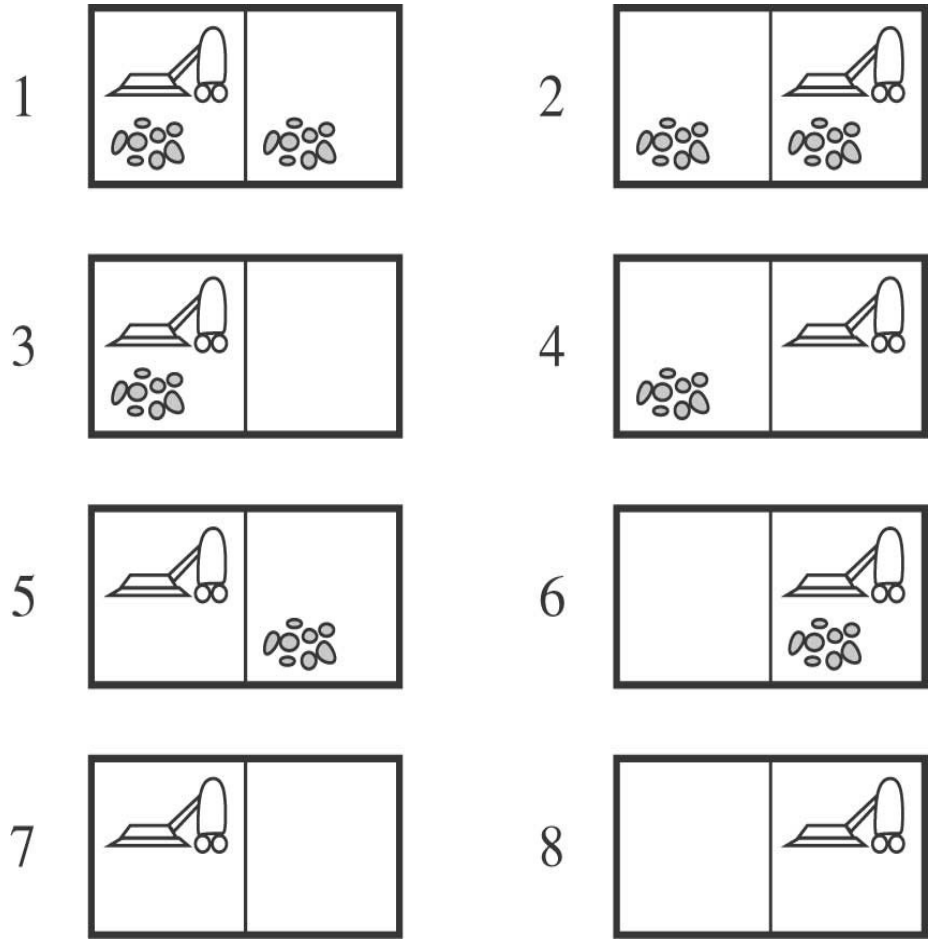
Estado simple, estado inicial 5.
¿Solución?
Sec= [*Derecha*, *Aspirar*]

Conformado, estado inicial
{1, 2, 3, 4, 5, 6, 7, 8}

Por ejemplo:
Derecha produce {2, 4, 6, 8}.

¿Solución?
Sec= [*Derecha, Aspirar, Izquierda, Aspirar*]

Contingencia, estado inicial 5.
Ley de Murphy: Aspirar a veces deposita suciedad en la alfombra.
Sensores locales: suciedad, sólo en el lugar.
¿Solución?
Sec= [*Derecha,* **si** *suciedad* **entonces** *Aspirar*]



Video

- https://www.youtube.com/watch?v=s8e6a-iMuQE comparativa dos búsquedas informadas en un laberinto

Resumen

- La formulación de problemas habitualmente requiere abstraer detalles del mundo real para definir un espacio de estados que pueda explorarse.
- Existe una variedad de estrategias de búsqueda no informada.
- La búsqueda en profundidad iterativa utiliza tan sólo un espacio lineal y no mucho más tiempo que otros algoritmos no informados.