Inteligencia Artificial

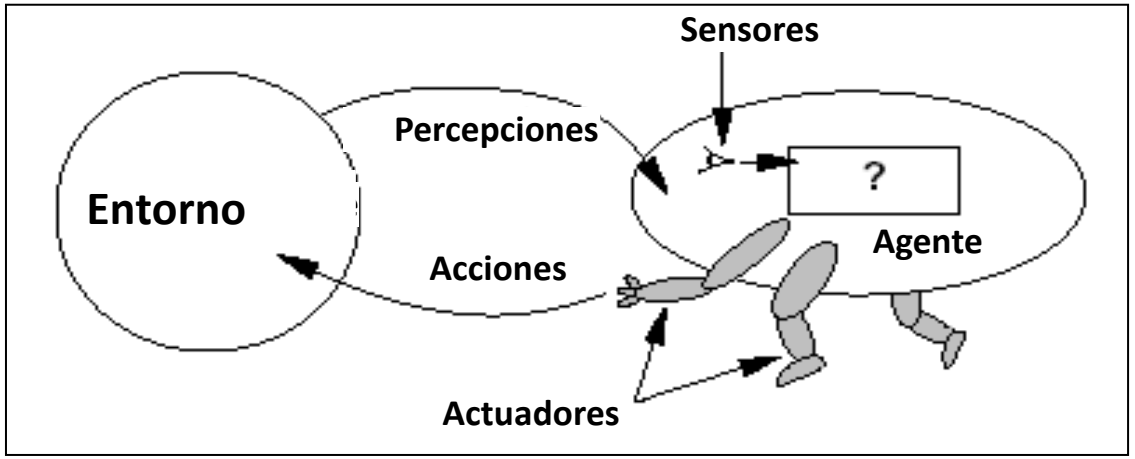
UNIVERSITAT DE BARCELONA

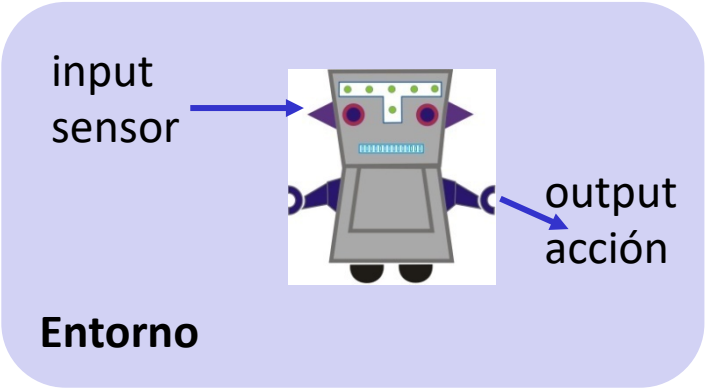
Resolución de Problemas Mediante Búsqueda: Búsqueda no Informada

Búsqueda no informada

- Agentes para la resolución de problemas
- Formulación de problemas
- Estrategias básicas de búsqueda
  - Búsqueda primero en anchura (BFS)
  - Búsqueda primero en profundidad (DFS)
  - Búsqueda de coste uniforme (UC)
  - ….
- Implementación

Agentes





Un agente es una entidad que percibe su entorno a través de sensores y actúa sobre el mismo mediante "actuadores".

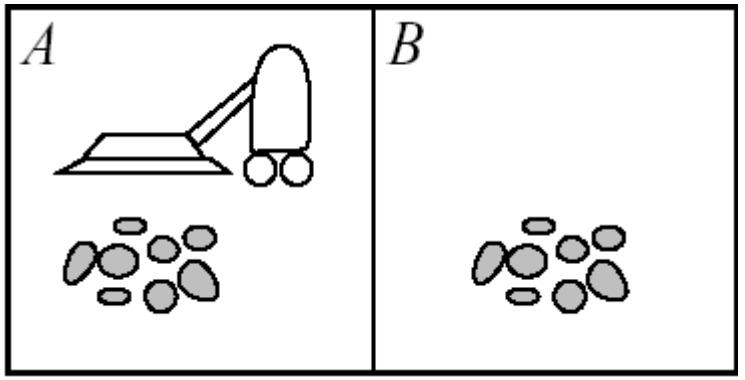
Los agentes incluyen a humanos, robots, softbots (programas), etc.

Modelizamos la función del agente t.q. proyecta una o varias percepciones en una acción:

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

Considerando agentes computacionales, el programa del agente se ejecuta sobre la arquitectura física para producir f.

Ejemplo: El agente aspirador



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

Acciones: *Izquierda, Derecha, Aspirar*

Agentes basados en objetivos

- Agentes inteligentes: Planifican antes de actuar:
  - Deben tener un modelo de cómo el mundo evoluciona en respuesta a sus acciones
  - Buscan secuencias de acciones que conducen a estados deseados (objetivos)
  - Son racionales

Agentes con objetivos

Etapas de resolución de problemas con objetivos:

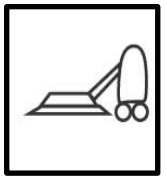
1. Formulación de objetivos: definir los estados objetivo y los factores que pueden influir su el grado de satisfacción.

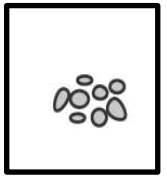
2. Formulación del problema: decidir qué acciones y estados considerar. Coste individual de las acciones ≥ 0.

3. Búsqueda de la solución: calcular la (mejor) secuencia de acciones que llevan a algún estado objetivo (a partir del inicial).

4. Ejecución: ejecutar las acciones calculadas. Tiene un coste total (aditivo).

Ejemplo: agente aspirador





- 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 aspirador es el agente que puede cumplirlo
- Las acciones que el robot aspirador 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: agente aspirador

- 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:

Ejecución:

Ejemplo: agente aspirador

- 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:

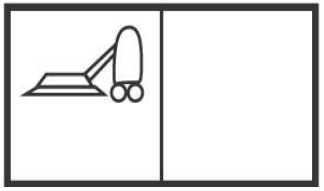
función sucesor

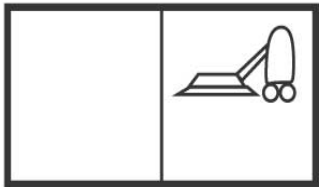
Búsqueda de la solución:

Ejecución:

Ejemplo: agente aspirador

- 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

Formulación del problema:

estados:

acciones:

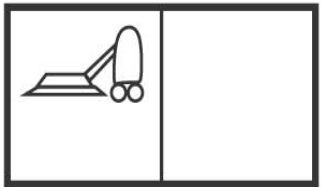
función sucesor

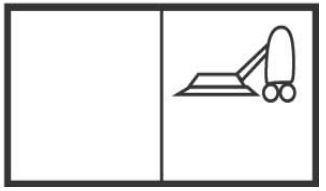
Búsqueda de la solución:

Ejecución:

Ejemplo: agente aspirador

- 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

Formulación del problema:

estados: ?

acciones:

función sucesor

Búsqueda de la solución:

Ejecución:

Ejemplo: agente aspirador

- 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:

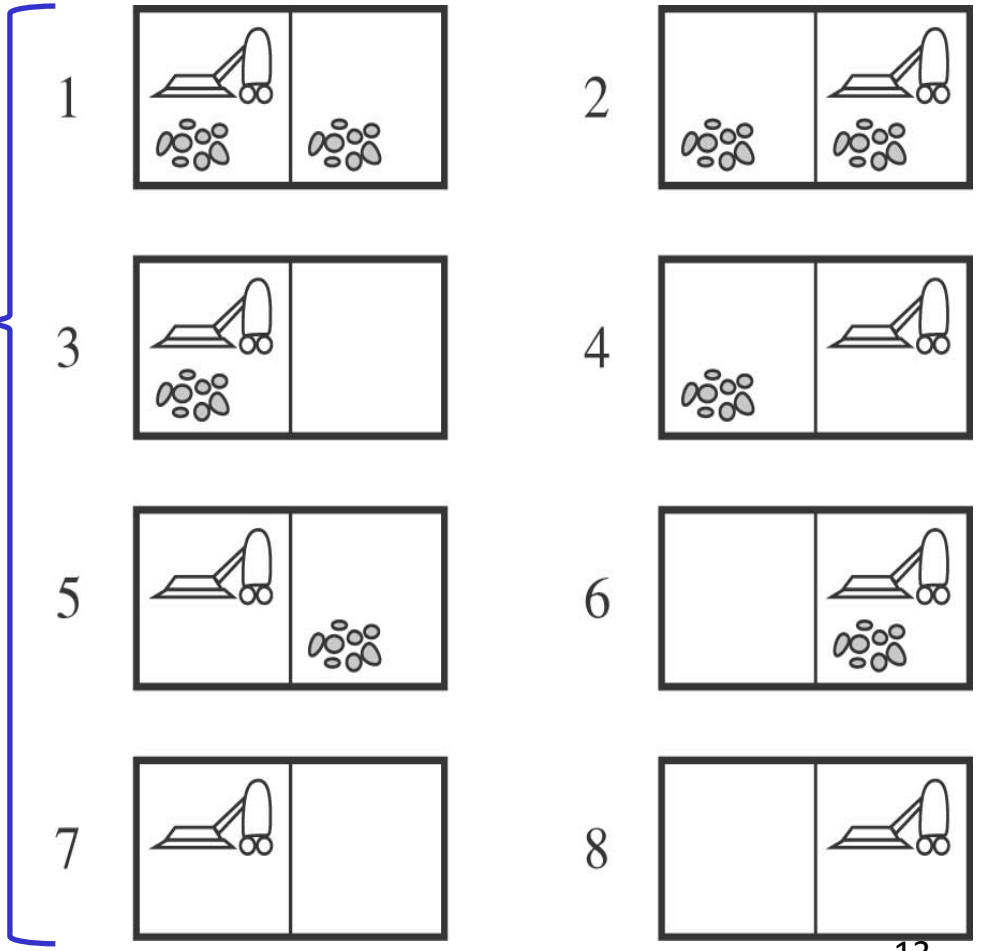
estados:

acciones:

función sucesor

Búsqueda de la solución:

Ejecución:



Ejemplo: agente aspirador

- 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:

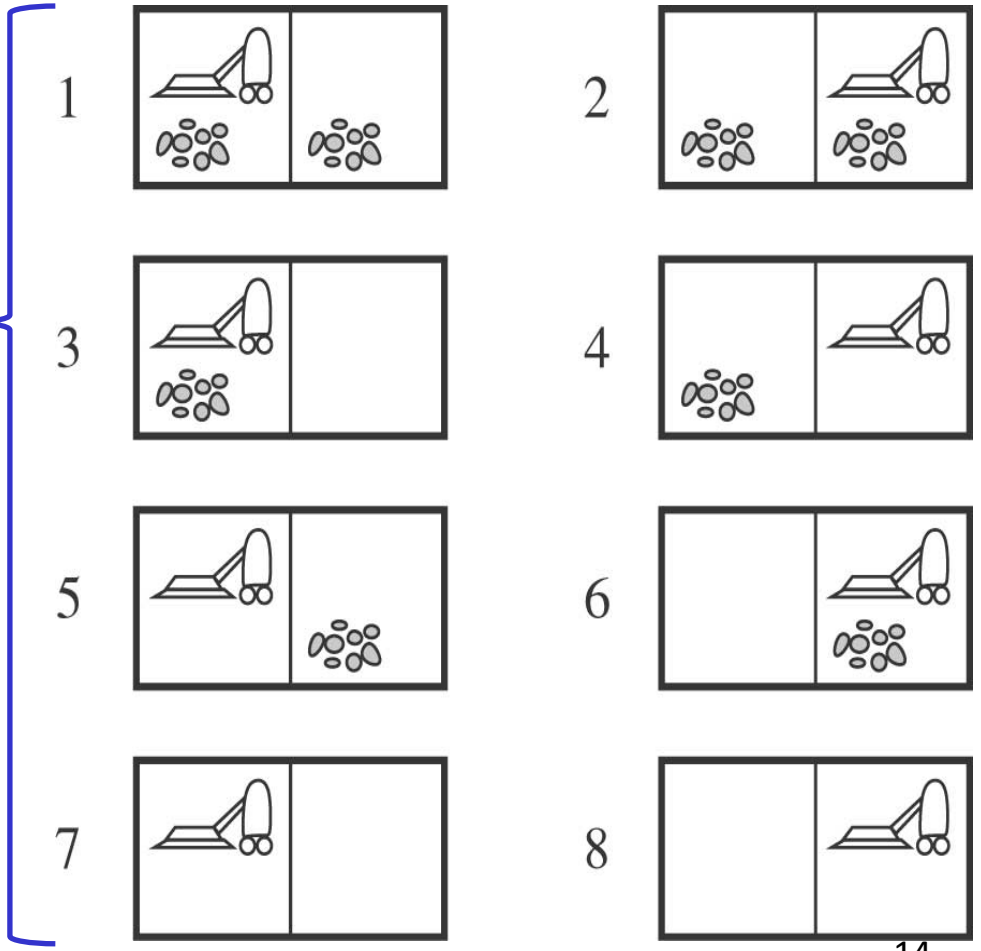
estados:

acciones: a ∈{ der, izq, asp} (coste 1u)

función sucesor ?

Búsqueda de la solución:

Ejecución:



Ejemplo: agente aspirador

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: a ∈{ der, izq, asp} (coste 1u)

función sucesor: S(e)={⟨a_i, e_i⟩} e∈{1,..,8}

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

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

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

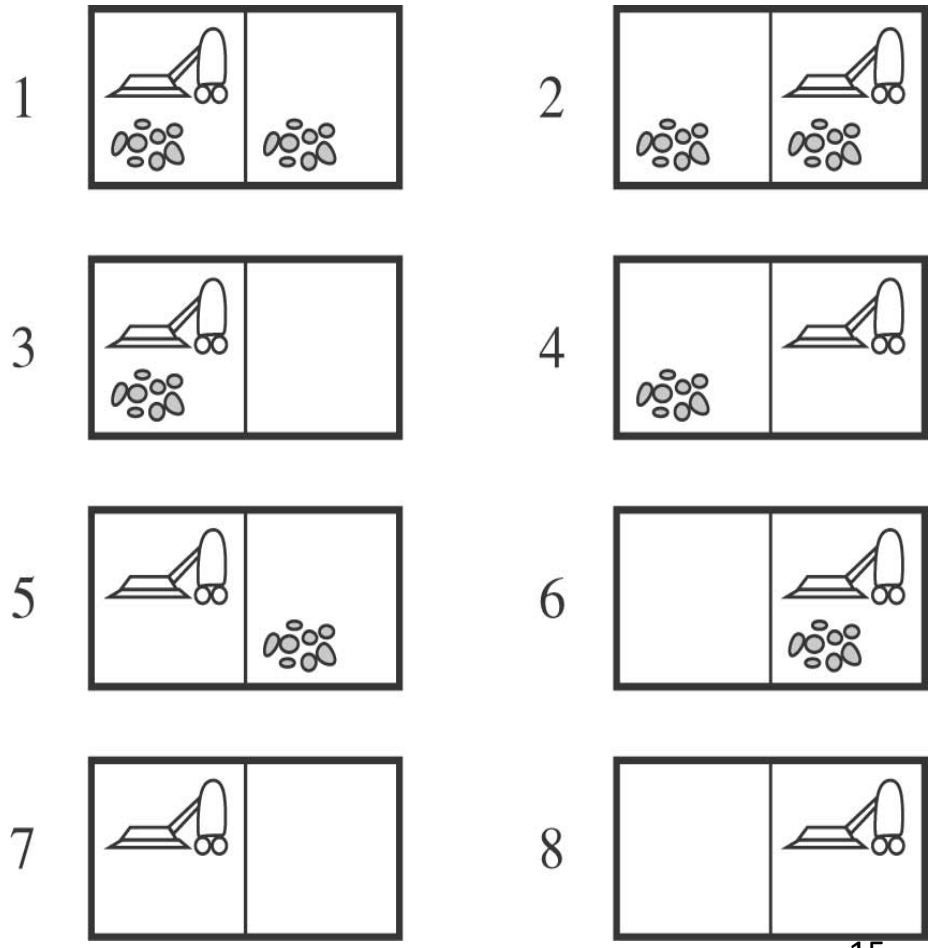
S(4)= { <der, 4>, <izq, 3>, <asp, 4>}

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>}



Por simplicidad, consideramos que la función sucesor siempre aplica las acciones en el mismo orden

Ejemplo: agente aspirador

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: a ∈{ der, izq, asp} (coste 1u)

función sucesor: S(e)={⟨a_i, e_i⟩} e∈{1,..,8}

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

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

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

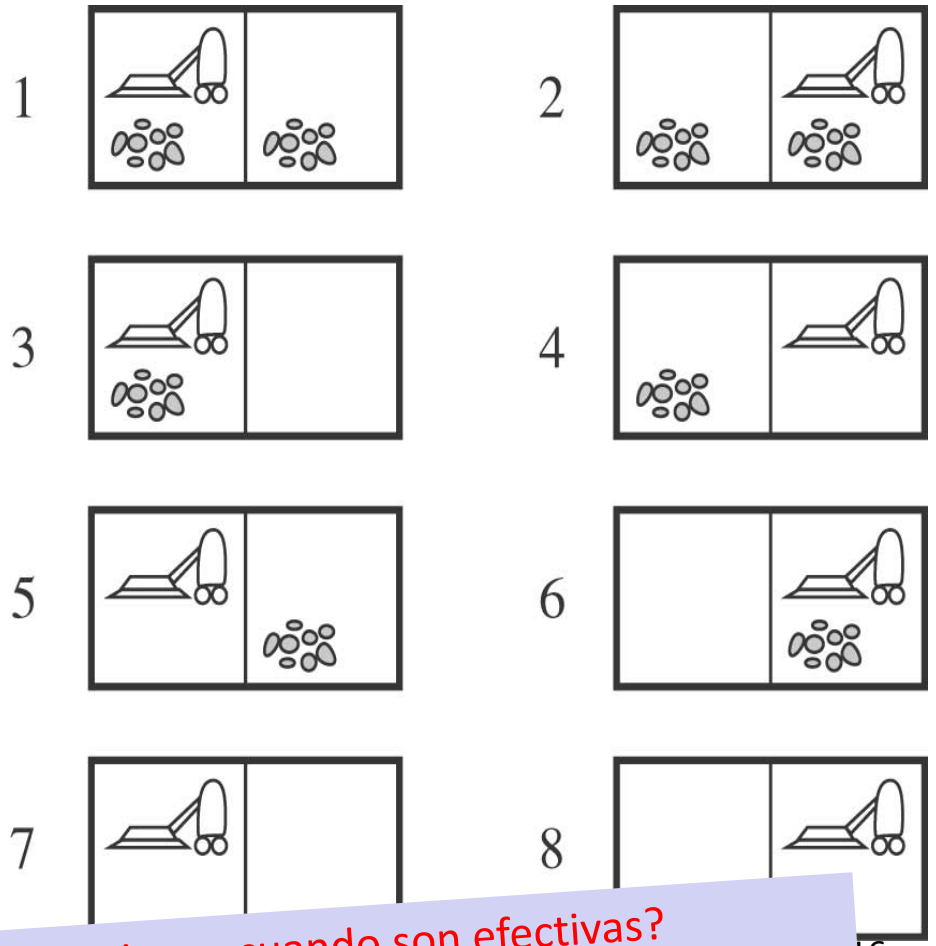
S(4)= { <der, 4>, <izq, 3>, <asp, 4>}

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: agente aspirador

Si sólo se pudieran hacer las acciones cuando son efectivas

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: a ∈{ der, izq, asp} (coste 1u)

función sucesor: S(e)={⟨a_i, e_i⟩} e∈{1,..,8}

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

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

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

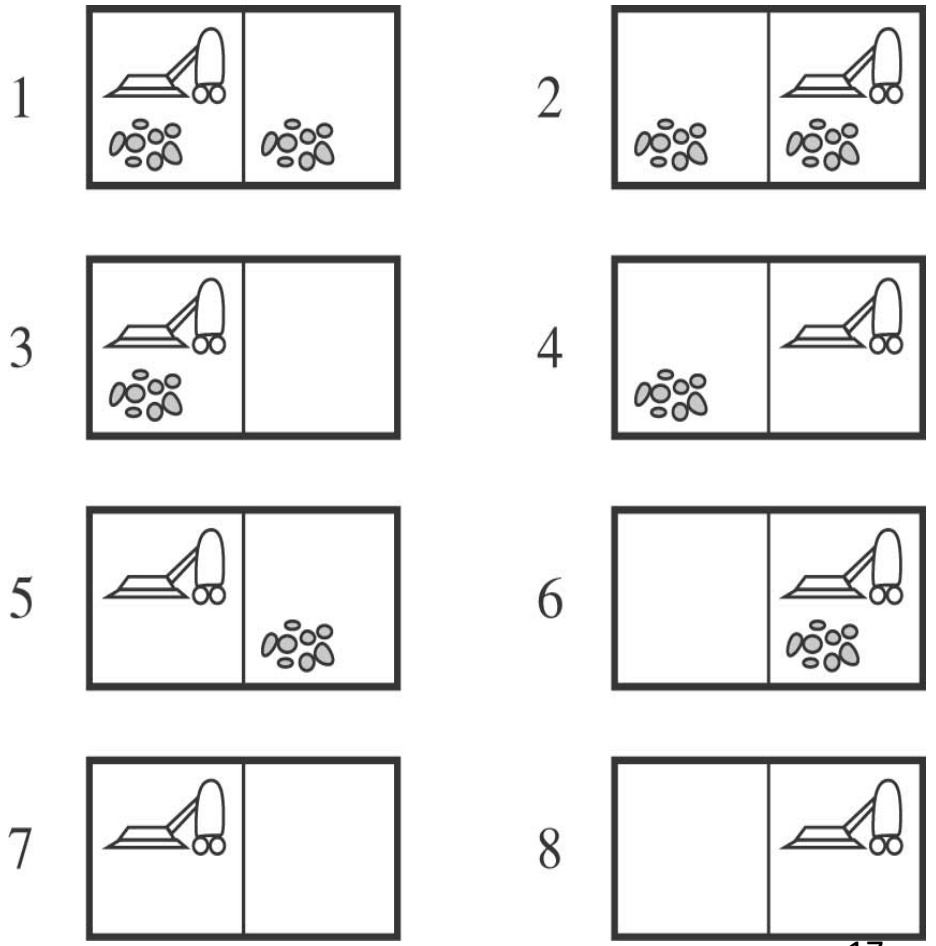
S(4) = {<izq, 3>}

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

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

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

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



Ejemplo: agente aspirador

- 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: a ∈{ der, izq, asp} (coste 1u)

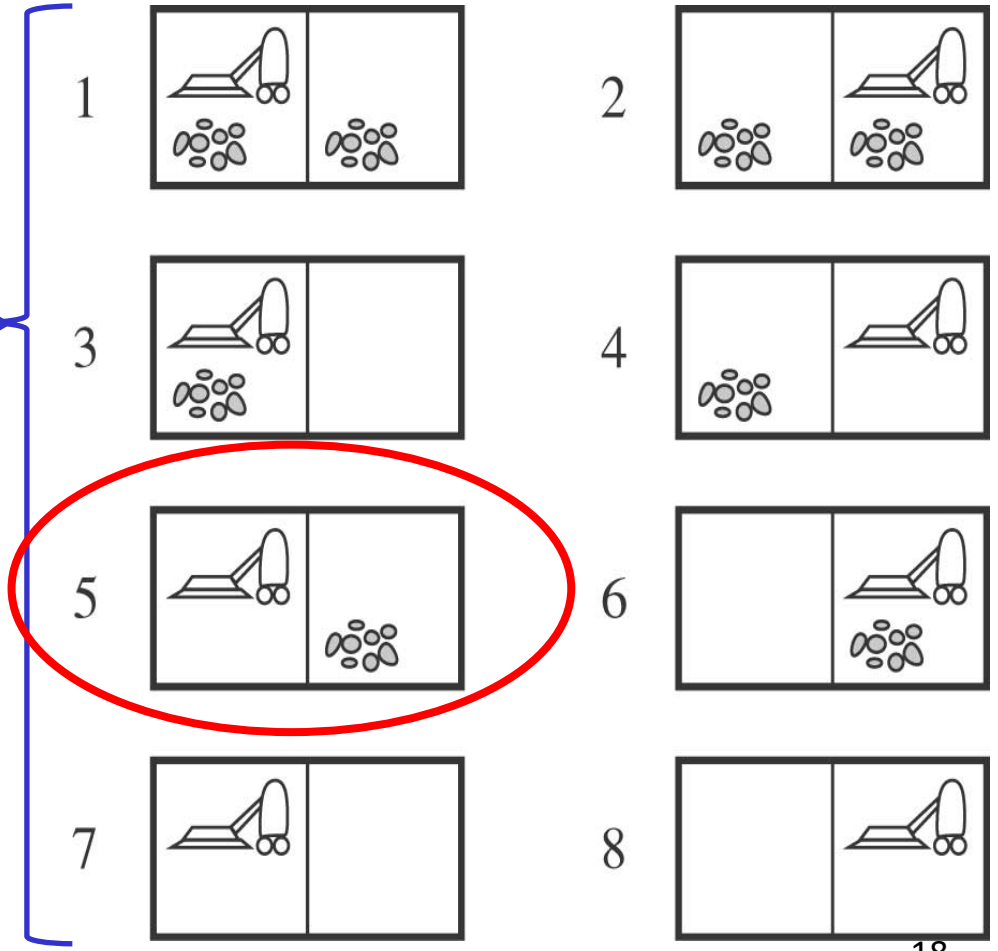
función sucesor: S(e)={⟨a_i, e_i⟩} e∈{1,..,8}

Búsqueda de la solución: ?

secuencia acciones

Ejecución:

Estado inicial



Ejemplo: agente aspirador

- 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:

función sucesor

Búsqueda de la solución:

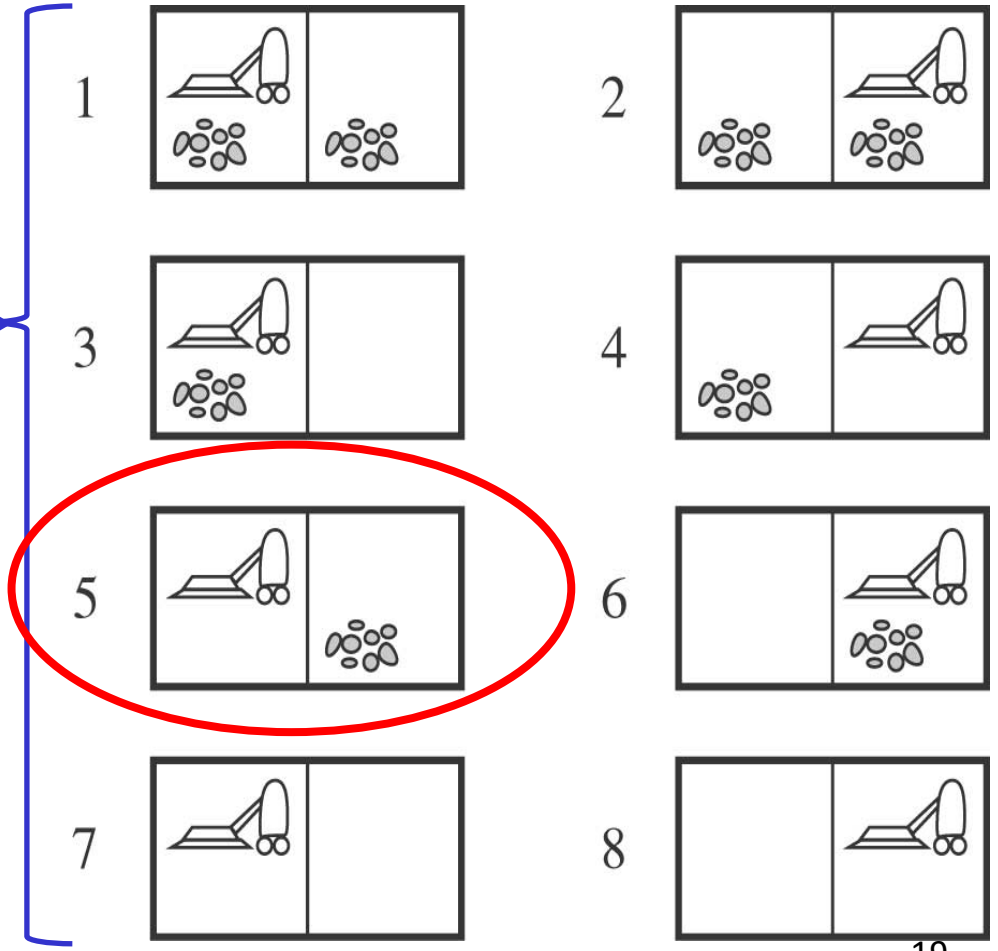
secuencia acciones

Sol= [*Derecha*, *Aspirar*]

A partir del estado inicial (5), ejecutar esta secuencia de acciones lleva al robot al estado 6 y luego al 8, que es uno de los estados objetivo

Ejecución: coste acciones

Estado inicial



Ejemplo: agente aspirador

- 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:

función sucesor

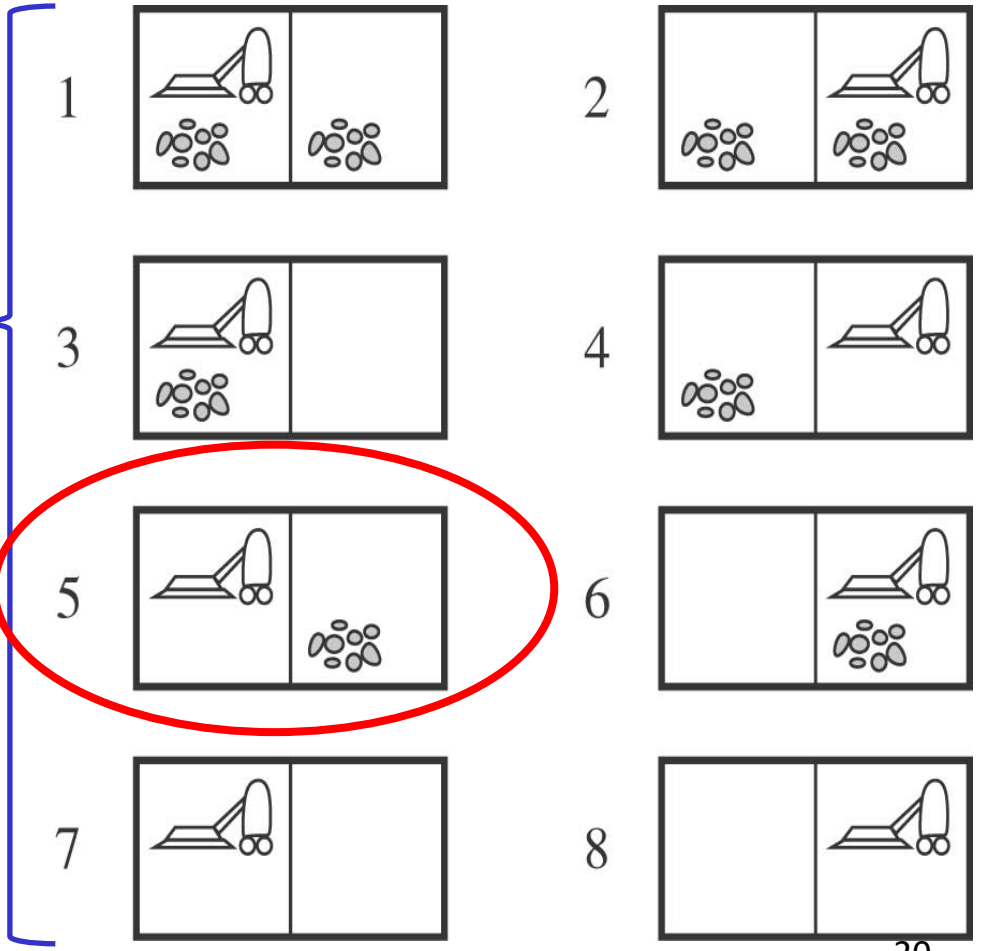
Búsqueda de la solución:

secuencia acciones

Sol= [*Derecha*, *Aspirar*]

A partir del estado inicial (5), ejecutar esta secuencia de acciones lleva al robot al estado 6 y luego al 8, que es uno de los estados objetivo

Ejecución: coste acciones ?



Ejemplo: agente aspirador

- 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

función sucesor

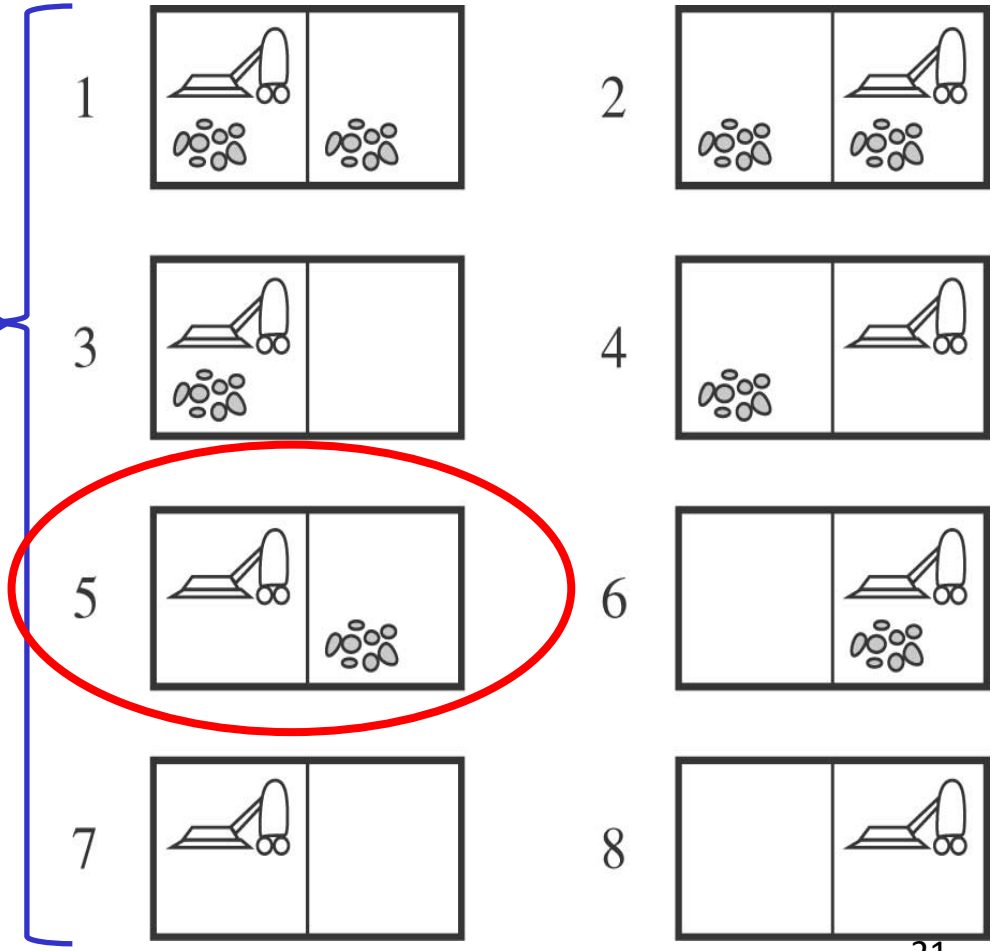
Búsqueda de la solución:

secuencia acciones

Sol= [*Derecha*, *Aspirar*]

A partir del estado inicial (5), ejecutar esta secuencia de acciones lleva al robot al estado 6 y luego al 8, que es uno de los estados objetivo

Ejecución: coste acciones 2 u

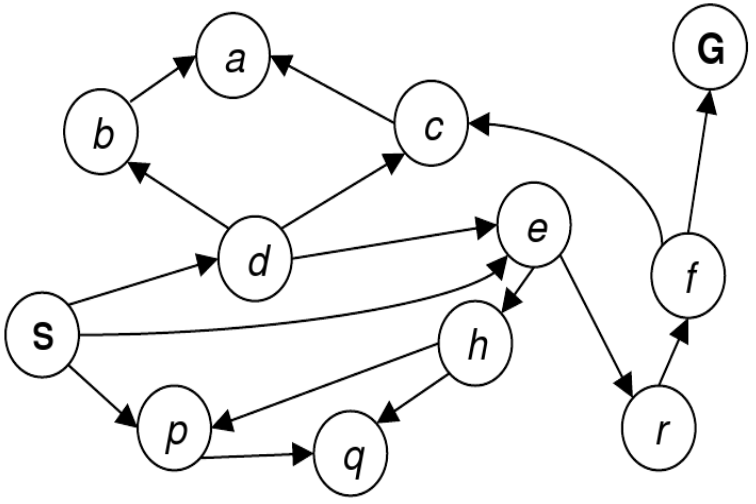


Video Pacman/Roomba

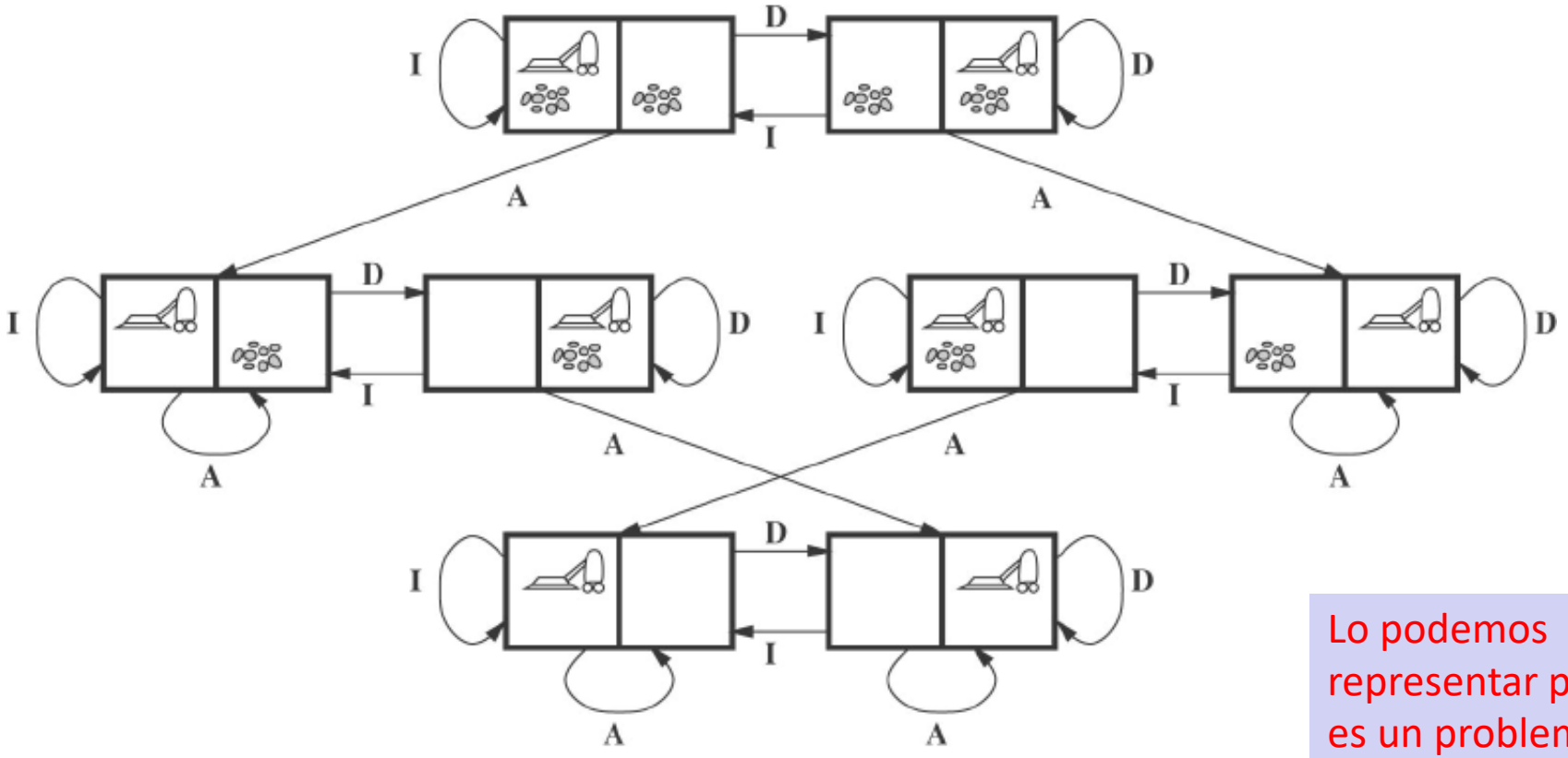
- http://pacman.elstonj.com/index.cgi?dir=videos&num=&perpage=&section=

Grafo del espacio de estados

- Todo problema de búsqueda tiene asociado un grafo de estados (vértices).
- La función sucesor se representa en base a los arcos (edges)
- En contadas ocasiones se puede construir este grafo en memoria



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
Acciones: *Izquierda (I), Derecha (D), Aspirar (A)*
Función sucesor Representada gráficamente por el grafo
Test objetivo: No suciedad
Coste del camino: 1u por acción

Algoritmo búsqueda en grafos
Best-First-Search

UNIVERSITAT DE BARCELONA

Grafo: se puede llegar al mismo vértice por más de un camino

función BEST-FIRST-SEARCH(*problema, f*) **devuelve** una solución o fallo

nodo ← CREAR-NODO(*problema.estado-inicial*), *alcanzados* ← ∅, *frontera* ← ∅

INSERTAR((*problema.estado-inicial,* nodo), *alcanzados*)

Lookup table

INSERTAR(nodo, *frontera*)

Priority queue, ordenada por *f*

bucle while not VACIA? (*frontera*) **hacer**

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

Pop

si ES-OBJETIVO (nodo.estado) **entonces devolver** SOLUCION(nodo)

bucle for each *sucesor* en EXPANDIR(nodo, *problema*) **hacer**

expansión : obtener sucesores

s ← *sucesor*.estado

si s no está en *alcanzados o* (*sucesor*.coste-camino < *alcanzados*[s].coste-camino) **entonces**

alcanzados[s] ← *sucesor*

INSERTAR(*sucesor*, *frontera*)

devolver fallo

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:

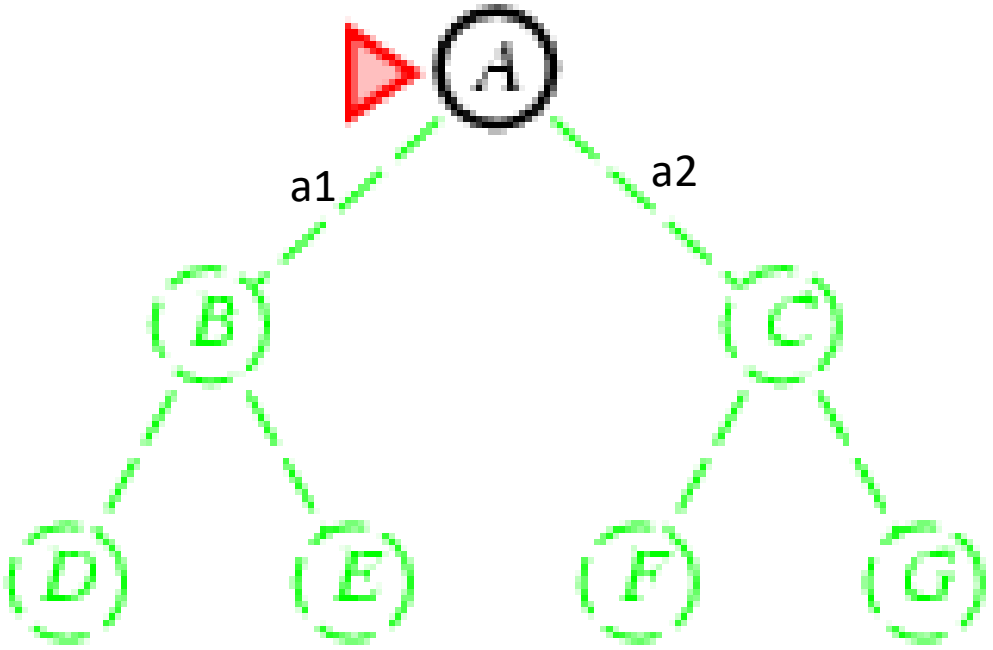
- **búsqueda primero en anchura (BFS: Breadth First Search)**
- búsqueda de coste uniforme
- búsqueda primero en profundidad (DFS: Depth First Search)
- búsqueda limitada en profundidad
- búsqueda por profundidad iterativa
- (búsqueda bidireccional)

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

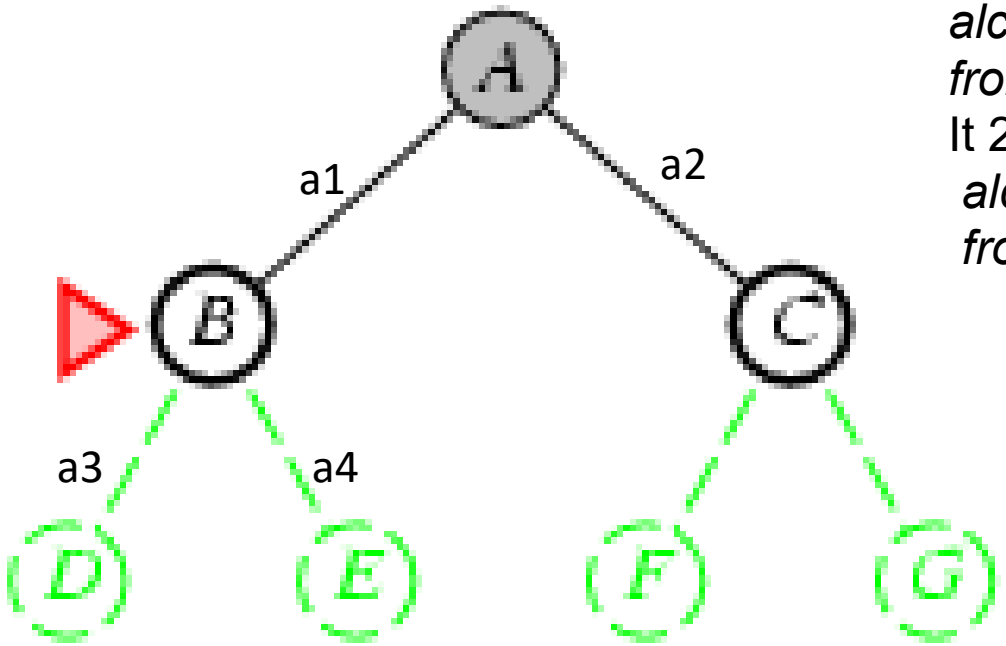


nodo = A
alcanzados={A}
frontera={A}
It 1: nodo=A ≠ D
alcanzados ={A,B,C}
frontera ={B, C}

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

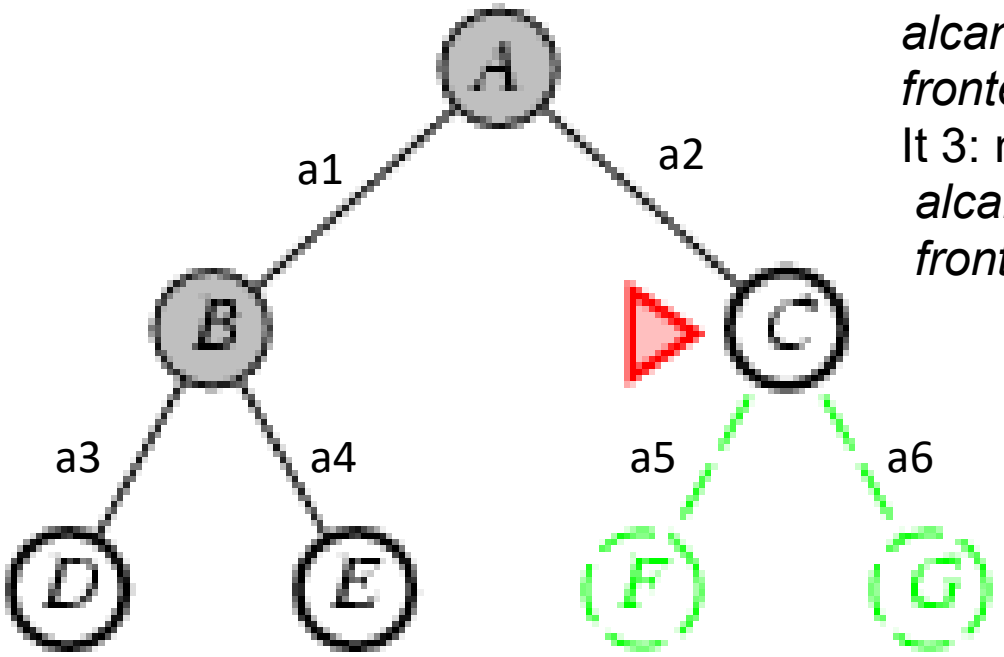


alcanzados ={A,B,C}
frontera ={B, C}
It 2: nodo=B ≠ D
alcanzados ={A,B,C,D,E}
frontera ={C, D, E}

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

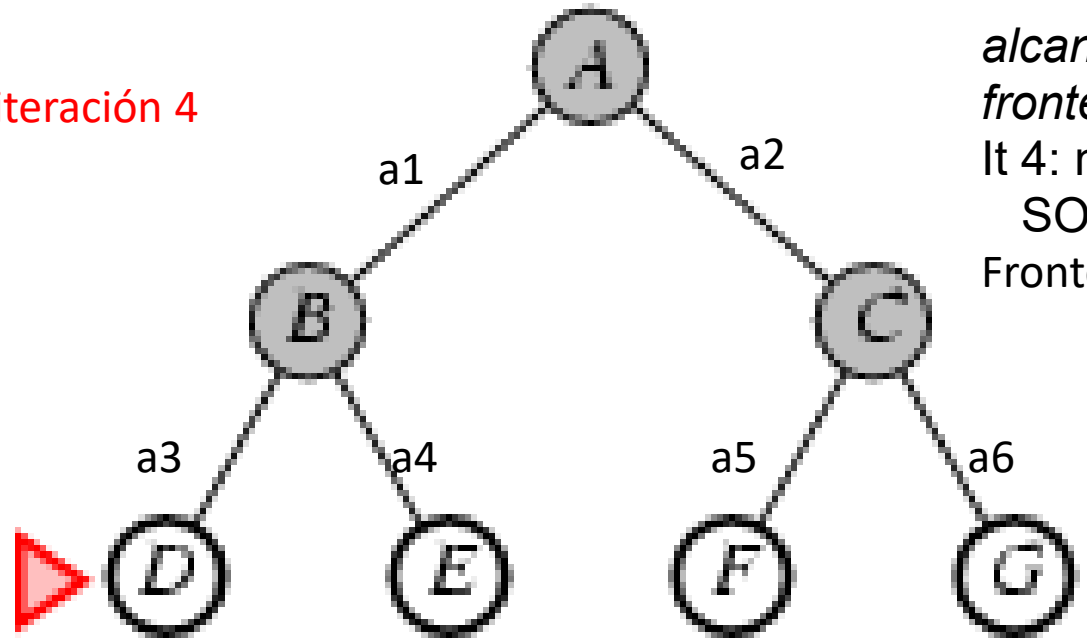


alcanzados ={A,B,C,D,E}
frontera ={C, D, E}
It 3: nodo=C ≠ D
alcanzados ={A,B,C,D,E,F,G}
frontera ={D, E, F, G}

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}



alcanzados ={A,B,C,D,E,F,G}
frontera ={D, E, F, G}
It 4: nodo=D
SOLUCION(D)
Frontera={E,F, G}

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 (UC: Uniform Cost)**
- **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

UNIVERSITAT DE BARCELONA

- Expande el nodo no expandido de menor coste
- Implementación:
  - La *frontera* es una cola ordenada por coste de camino

- Equivalente a la búsqueda primero en anchura si el coste de todos los pasos es igual.

- Pero: el coste de cada paso deberá ser >0 (≥ ε) para no quedar atrapado en un bucle infinito al expandir un nodo que tenga una acción de coste cero que vuelva al mismo estado (p.ej. NoOp)

Búsqueda de coste uniforme

UNIVERSITAT DE BARCELONA

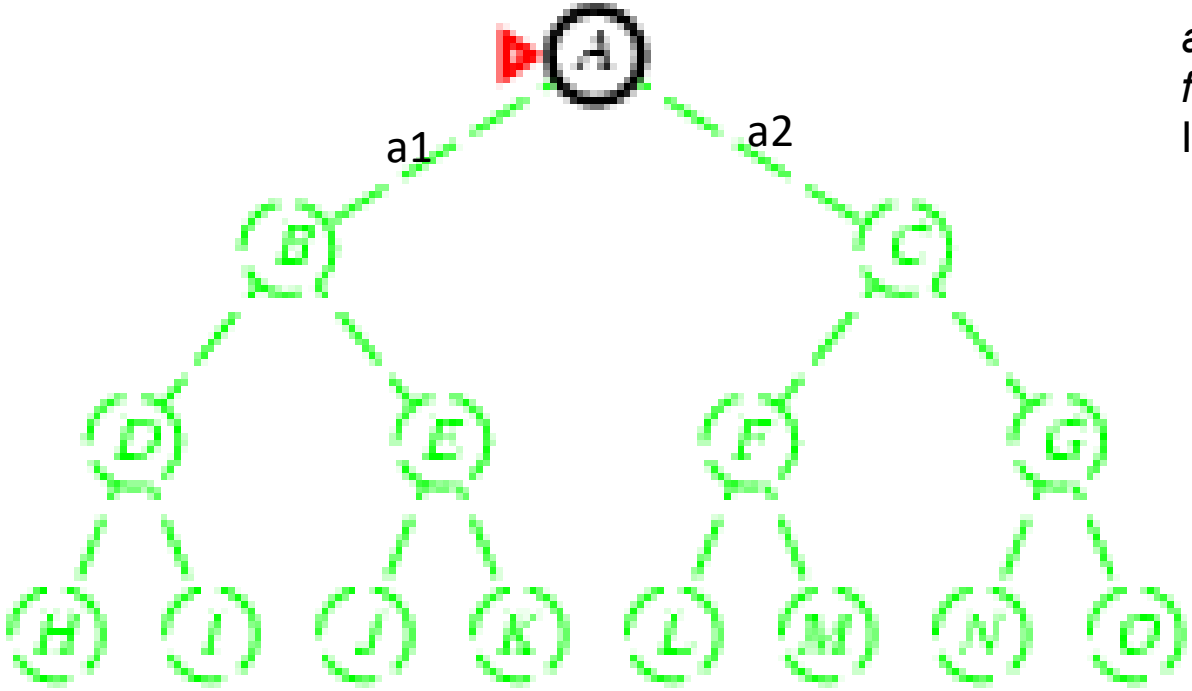
- Expande el nodo no expandido de menor coste
- Implementación:
  - La *frontera* es una cola ordenada por coste de camino

- Equivalente a la búsqueda primero en anchura si el coste de todos los pasos es igual. ¿Por qué?
- Pero: el coste de cada paso deberá ser >0 (≥ ε) para no quedar atrapado en un bucle infinito al expandir un nodo que tenga una acción de coste cero que vuelva al mismo estado (p.ej. NoOp)

Búsqueda primero en profundidad

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

Objetivo: M

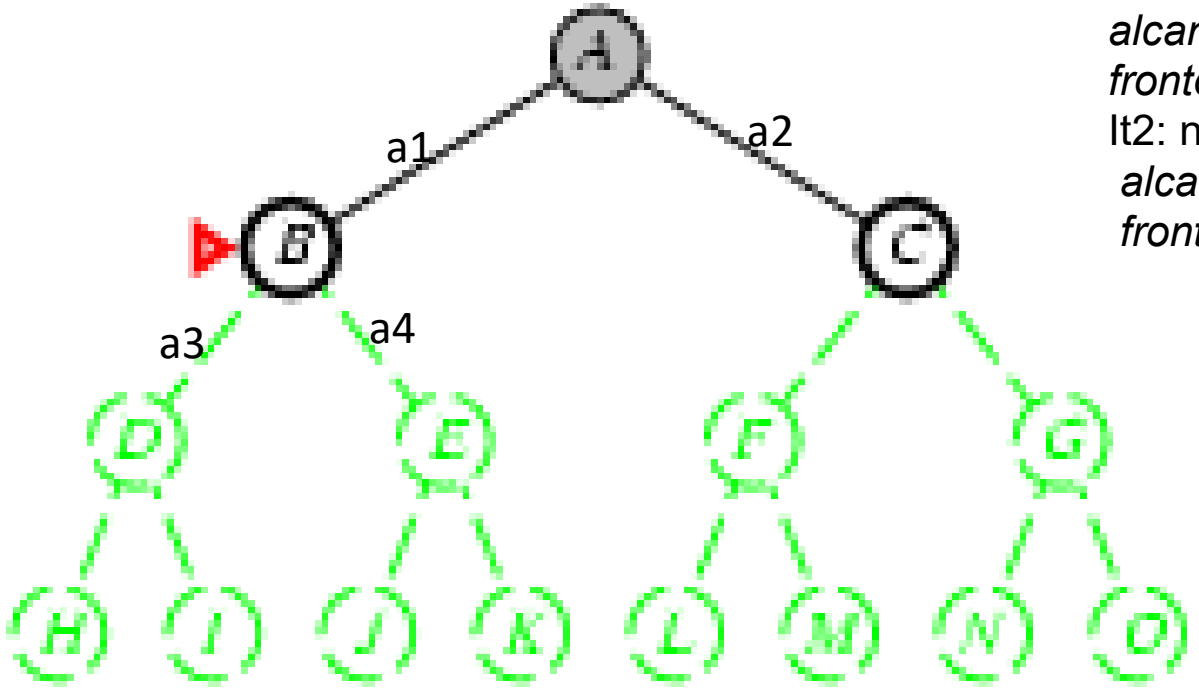


alcanzados ={A}
frontera={A}
It1: nodo=A ≠ M
alcanzados ={A,B,C}
frontera ={B, C}

Búsqueda primero en profundidad

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

Objetivo: M

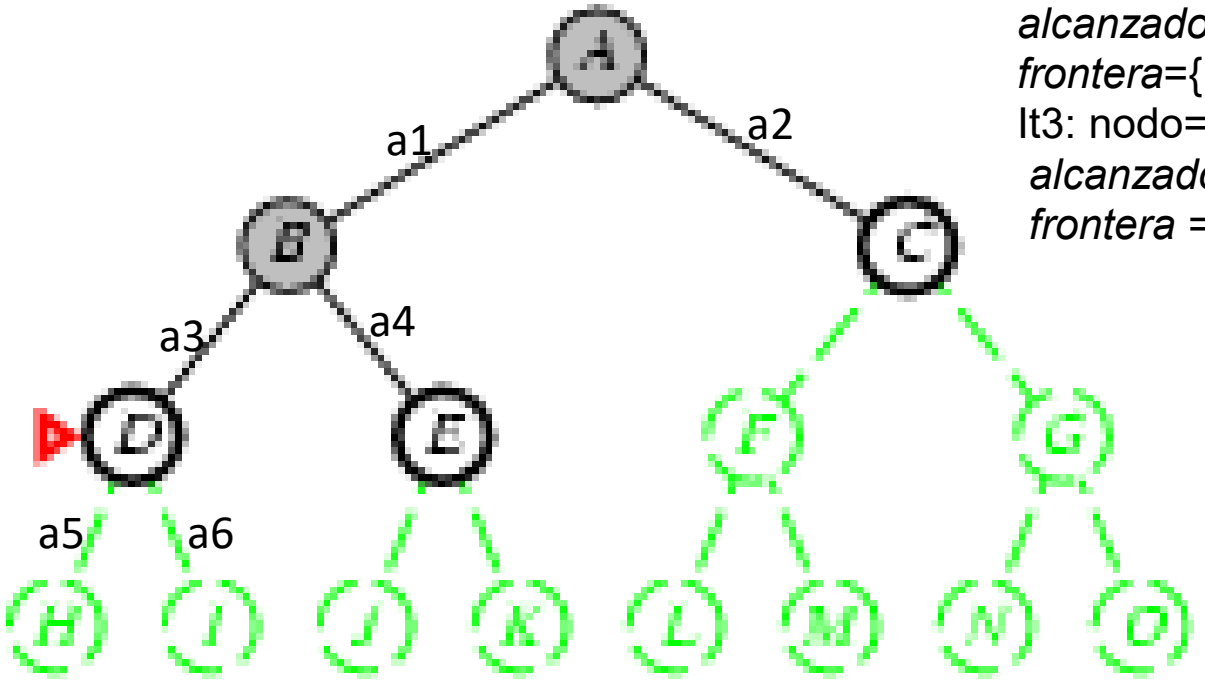


alcanzados ={A,B,C}
frontera={B,C}
It2: nodo=B ≠ M
alcanzados ={A,B,C,D,E}
frontera ={D, E, C}

Búsqueda primero en profundidad

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

Objetivo: M

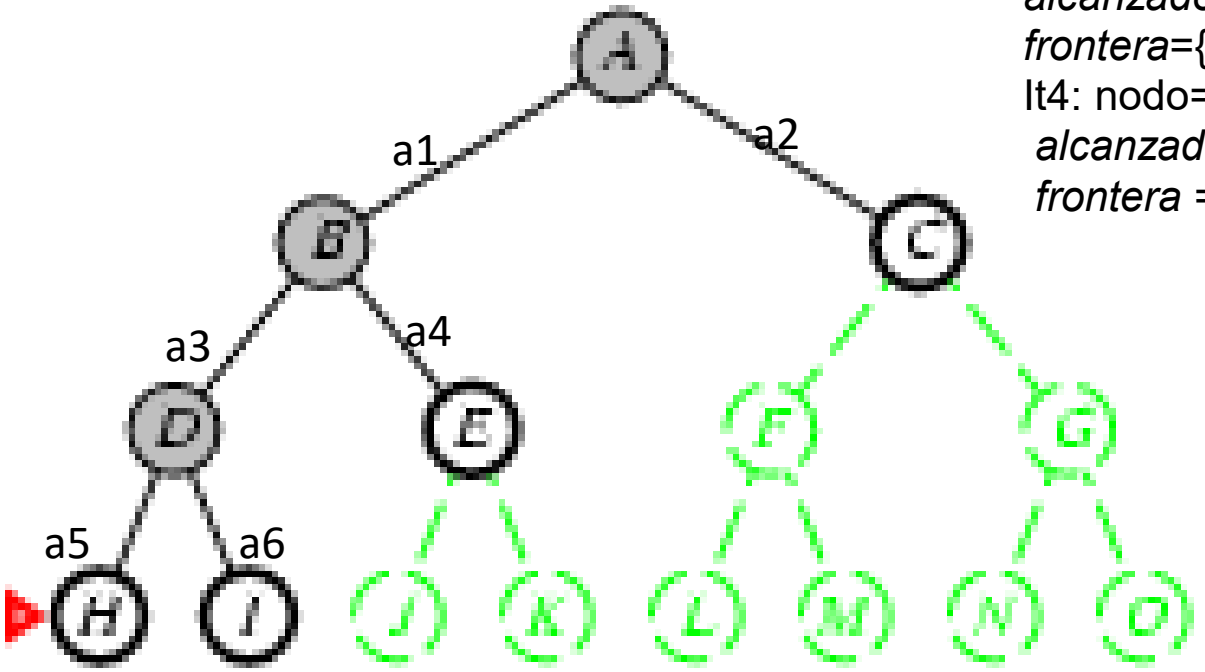


alcanzados ={A,B,C,D,E}
frontera={D,E,C}
It3: nodo=D ≠ M
alcanzados ={A,B,C,D,E,H,I}
frontera ={H,I, E, C}

Búsqueda primero en profundidad

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

Objetivo: M

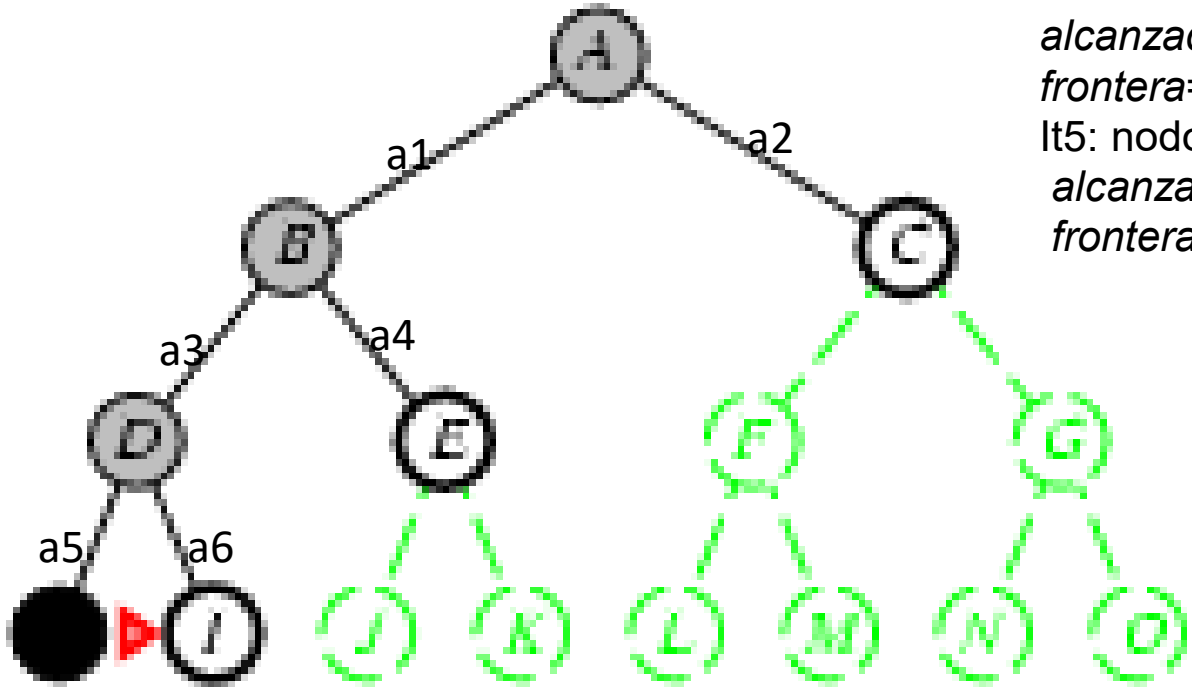


alcanzados ={A,B,C,D,E,H,I}
frontera={H,I,E,C}
It4: nodo=H ≠ M
alcanzados ={A,B,C,D,E,H,I}
frontera ={I, E, C}

Búsqueda primero en profundidad

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

Objetivo: M

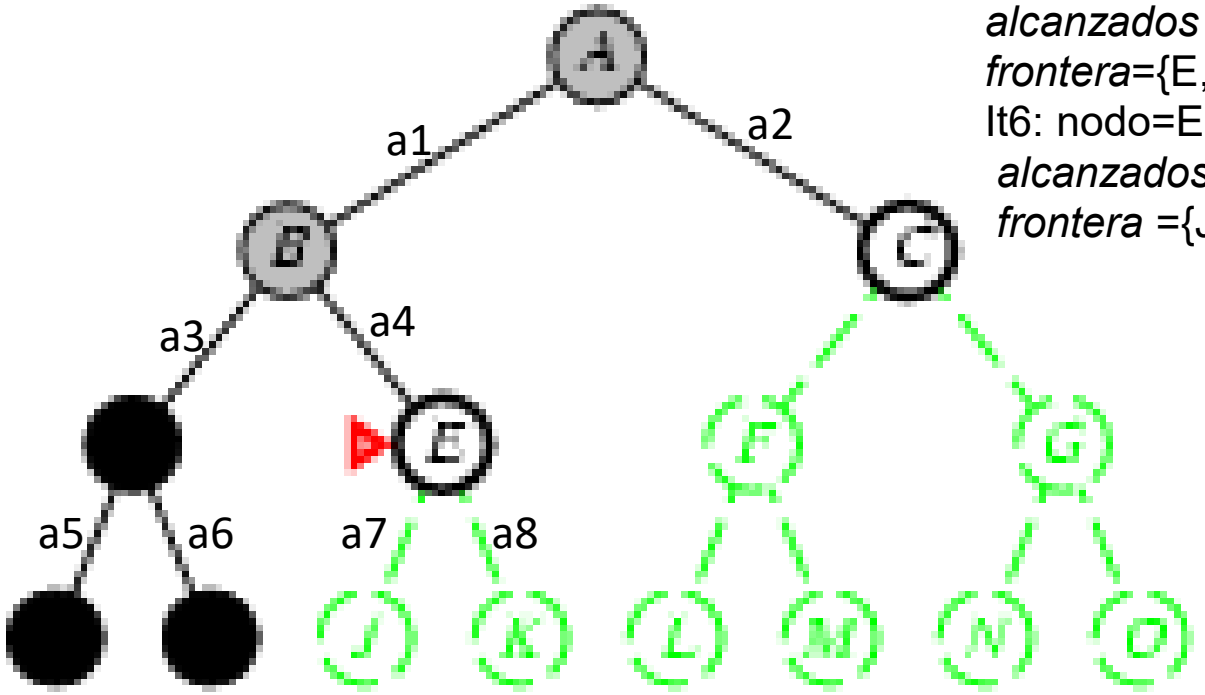


alcanzados ={A,B,C,D,E,H,I}
frontera={I,E,C}
It5: nodo=I ≠ M
alcanzados ={A,B,C,D,E,H,I}
frontera ={E, C}

Búsqueda primero en profundidad

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

Objetivo: M

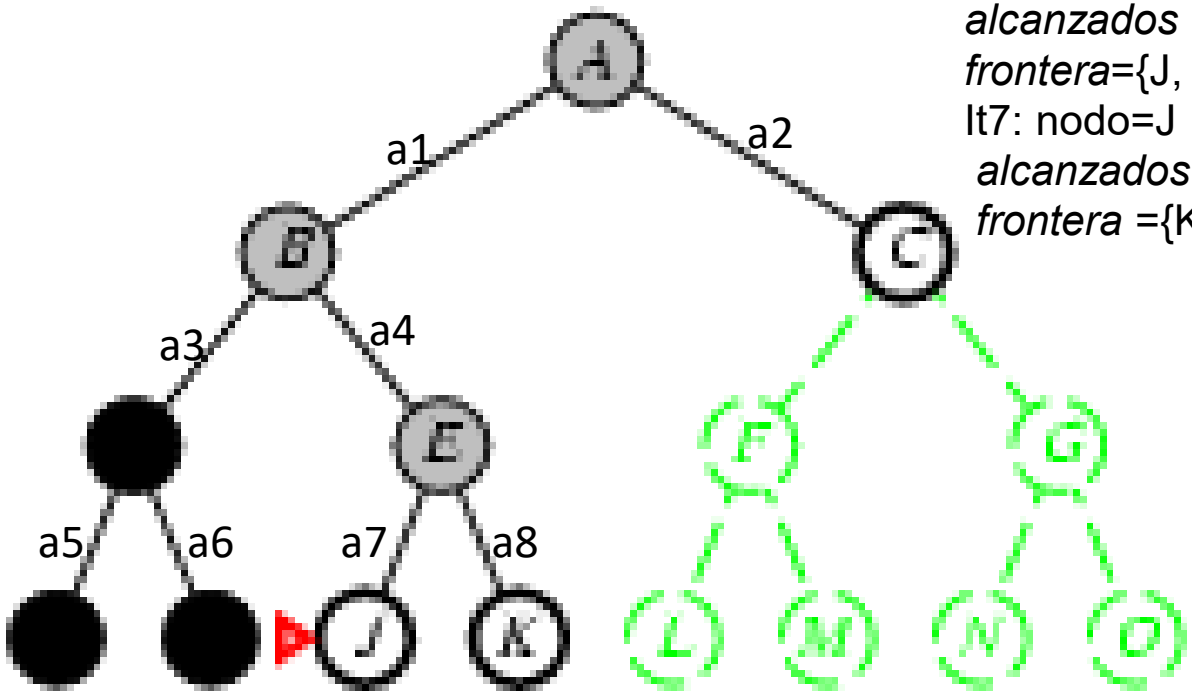


alcanzados ={A,B,C,D,E,H,I}
frontera={E,C}
It6: nodo=E ≠ M
alcanzados ={A,B,C,D,E,H,I,J,K}
frontera ={J, K, C}

Búsqueda primero en profundidad

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

Objetivo: M

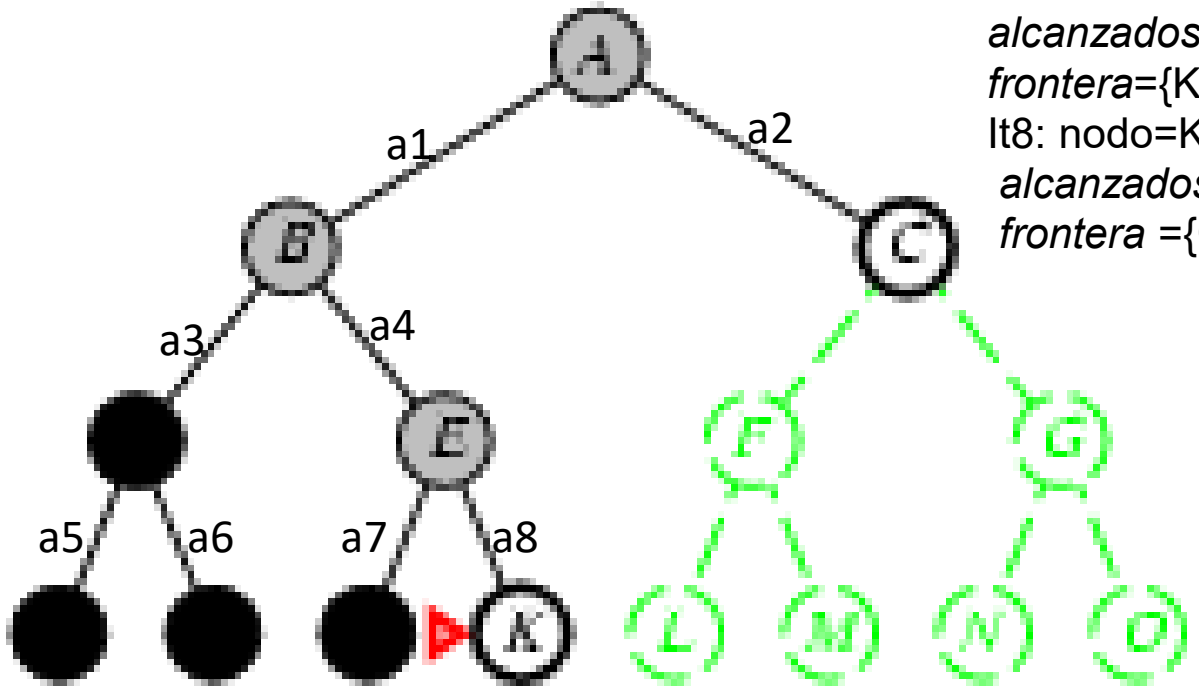


alcanzados ={A,B,C,D,E,H,I,J,K}
frontera={J, K, C}
It7: nodo=J ≠ M
alcanzados ={A,B,C,D,E,H,I,J,K}
frontera ={K, C}

Búsqueda primero en profundidad

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

Objetivo: M

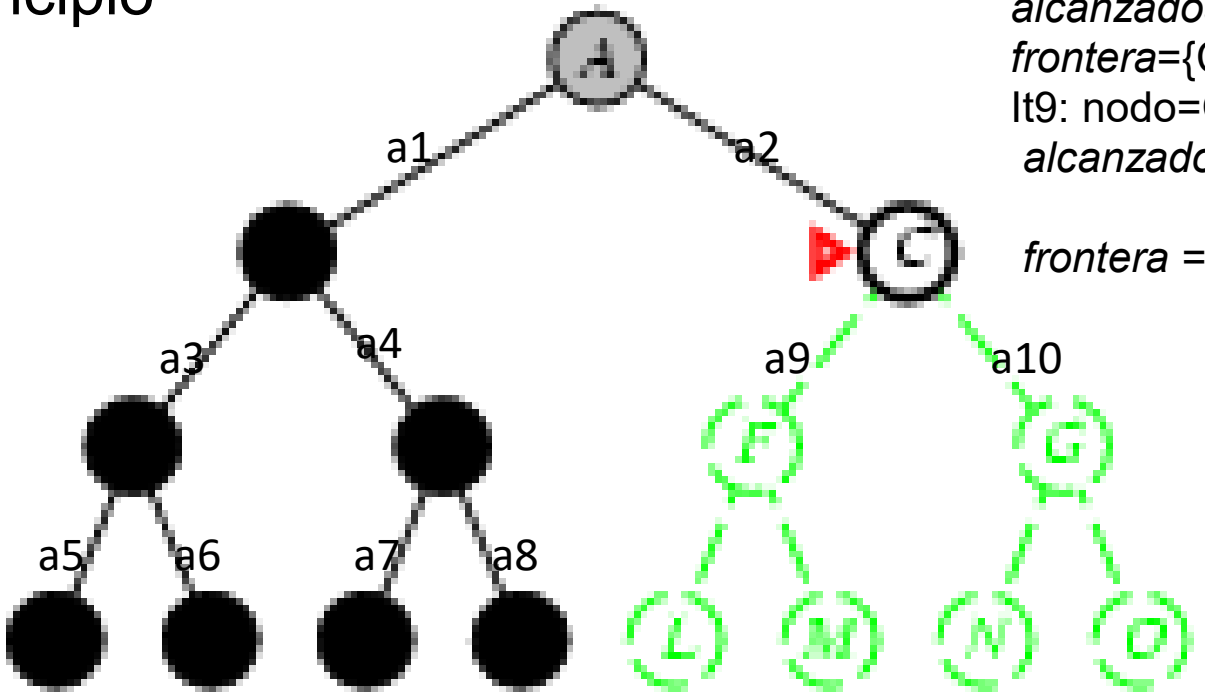


alcanzados ={A,B,C,D,E,H,I,J,K}
frontera={K, C}
It8: nodo=K ≠ M
alcanzados ={A,B,C,D,E,H,I,J,K}
frontera ={C}

Búsqueda primero en profundidad

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

Objetivo: M

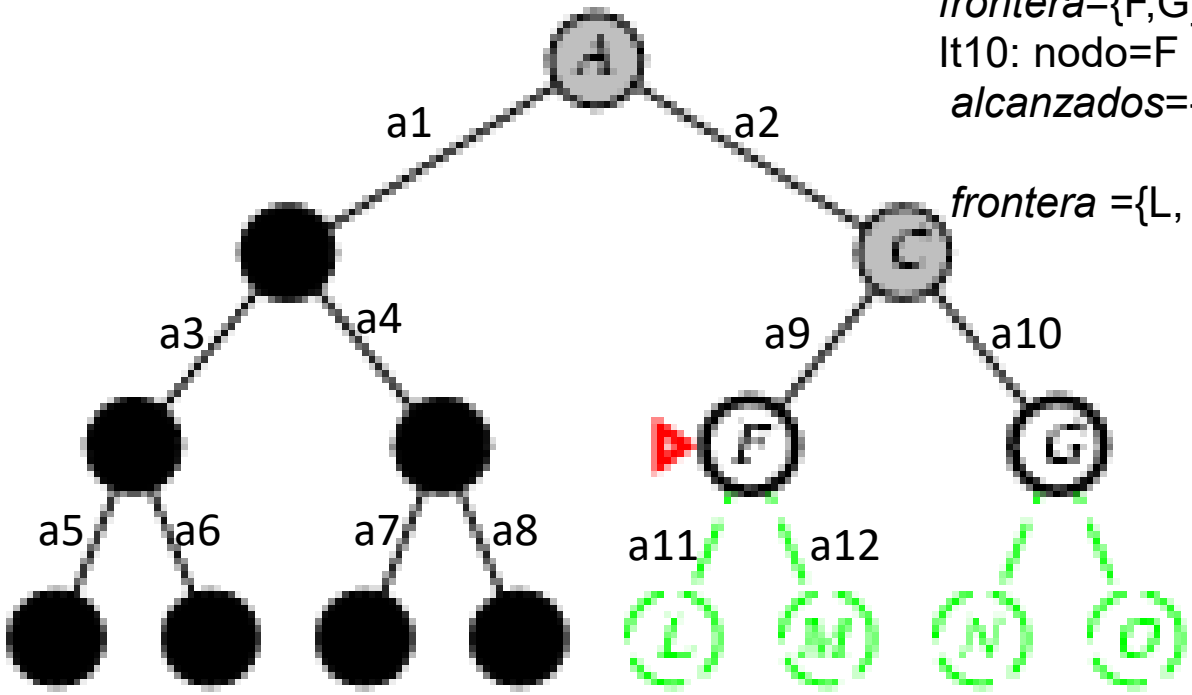


alcanzados ={A,B,C,D,E,H,I,J,K}
frontera={C}
It9: nodo=C ≠ M
alcanzados={A,B,C,D,E,H,I,J,K, F,G}
frontera ={F, G}

Búsqueda primero en profundidad

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

Objetivo: M



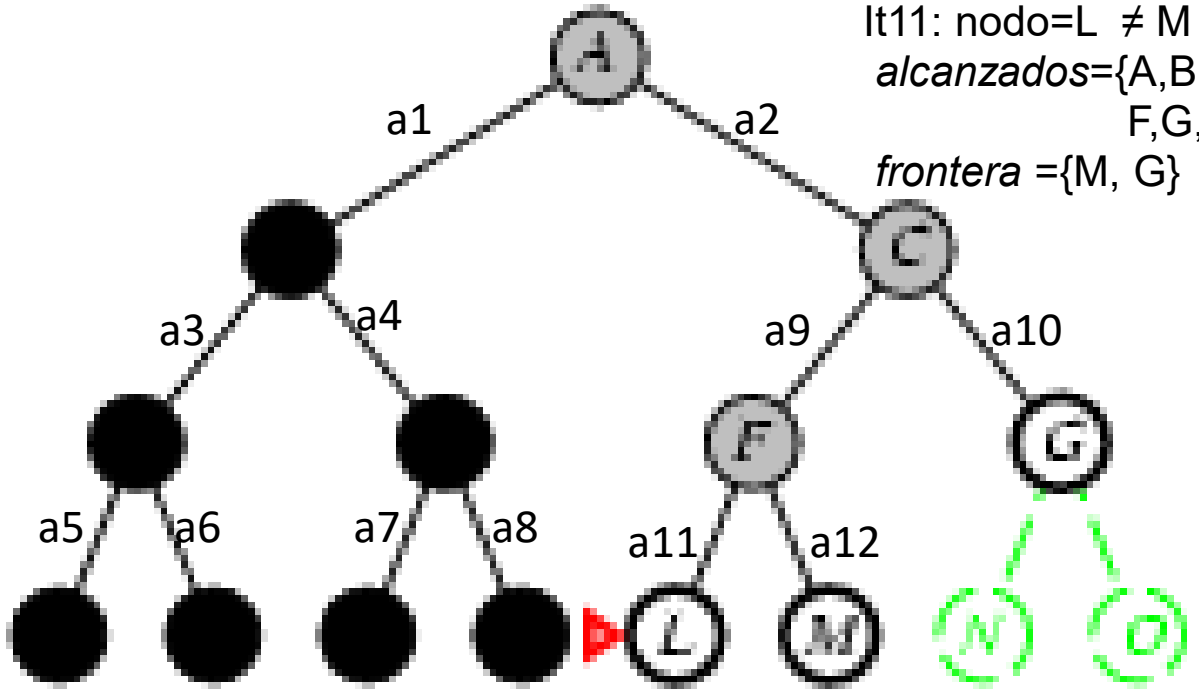
alcanzados ={A,B,C,D,E,H,I,J,K,F,G}
frontera={F,G}
It10: nodo=F ≠ M
alcanzados={A,B,C,D,E,H,I,J,K, F,G,L,M}
frontera ={L, M, G}

Búsqueda primero en profundidad

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

alcanzados ={A,B,C,D,E,H,I,J,K,F,G,L,M}
frontera={L,M,G}
It11: nodo=L ≠ M
alcanzados={A,B,C,D,E,H,I,J,K, F,G,L,M}
frontera ={M, G}

Objetivo: M

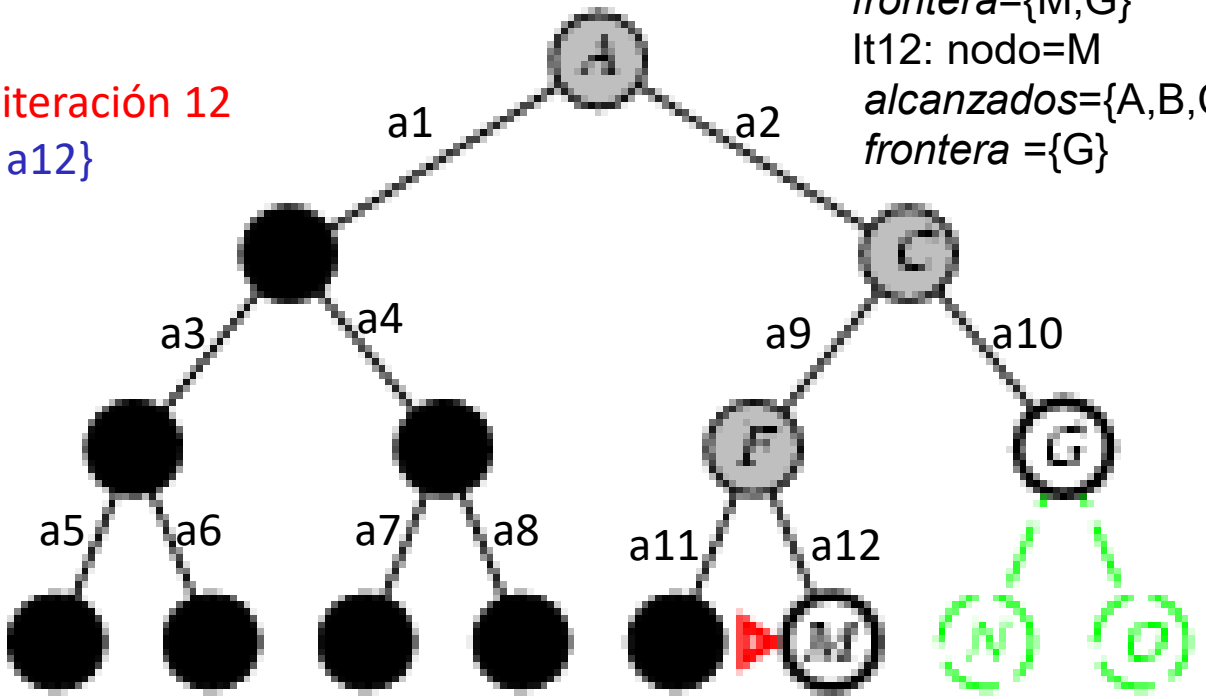


Búsqueda primero en profundidad

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

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

alcanzados ={A,B,C,D,E,H,I,J,K,F,G,L,M}
frontera={M,G}
It12: nodo=M
alcanzados={A,B,C,D,E,H,I,J,K,F,G,L,M}
frontera ={G}

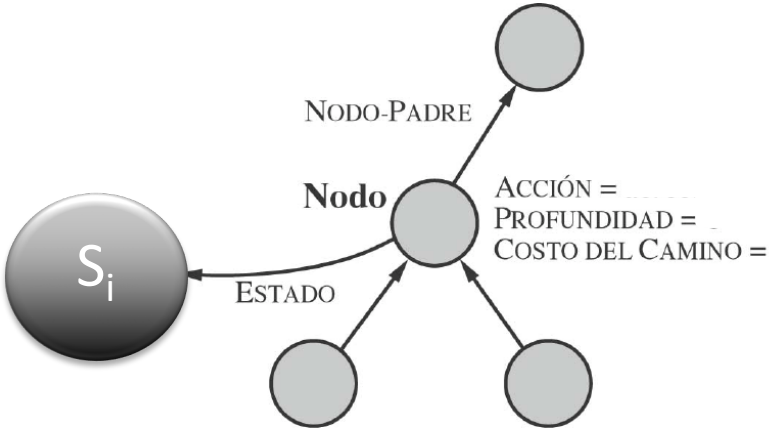


Implementación: estados frente a nodos

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 (función RESULTADO(s, *acción*) del problema) para obtener los estados correspondientes y con ellos crear nuevos nodos (rellenando los distintos campos)

Búsqueda en grafos

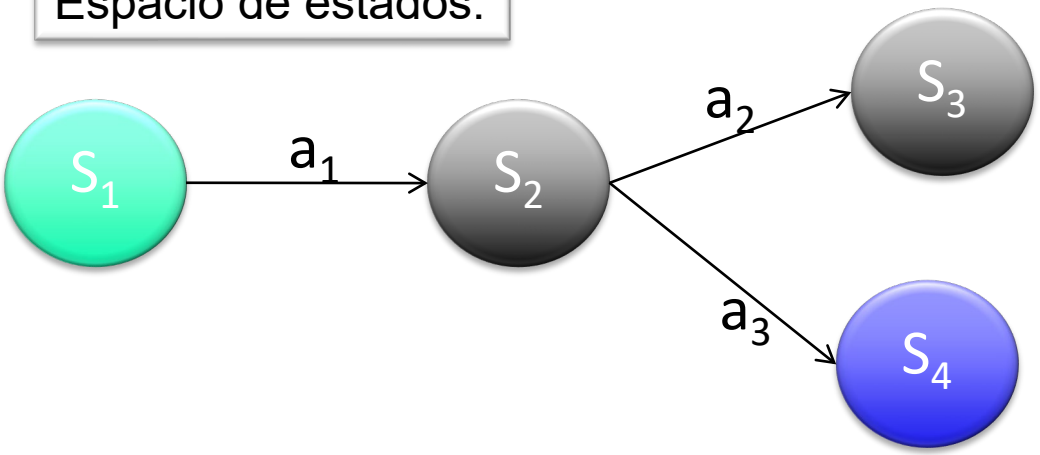

```
función BEST-FIRST-SEARCH(problema, f) devuelve una solución o fallo
    nodo ← CREAR-NODO(problema.estado-inicial), alcanzados ← ∅, frontera ← ∅
    INSERTAR((problema.estado-inicial, nodo), alcanzados)
    INSERTAR(nodo, frontera)
    bucle while not VACIA? (frontera) hacer
        nodo ←SACAR-BORRANDO-PRIMERO(frontera)
        si ES-OBJETIVO (nodo.estado) entonces devolver SOLUCION(nodo)
        bucle for each sucesor en EXPANDIR(nodo, problema) hacer
            s ← sucesor.estado
            si s no está en alcanzados o (sucesor.coste-camino < alcanzados[s].coste-camino) entonces
                alcanzados[s] ← sucesor
                INSERTAR(sucesor, frontera)
    devolver fallo
```

```
función EXPANDIR(nodo, problema) devuelve un conjunto de nodos
    s ← nodo.estado , sucesores ← ∅
    para cada acción en problema.ACCIONES(s) hacer
        s' ← problema.RESULTADO(s, acción)
        n ← un nuevo NODO
        ESTADO[n] ←s'
        NODO-PADRE[n] ←nodo
        ACCIÓN[n] ←acción
        COSTE-CAMINO[n] ← nodo.coste-camino +problema.COSTE-ACCION(s,acción,s')
        PROFUNDIDAD[n] ←PROFUNDIDAD[nodo] + 1
        añadir n a sucesores
    devolver sucesores
```

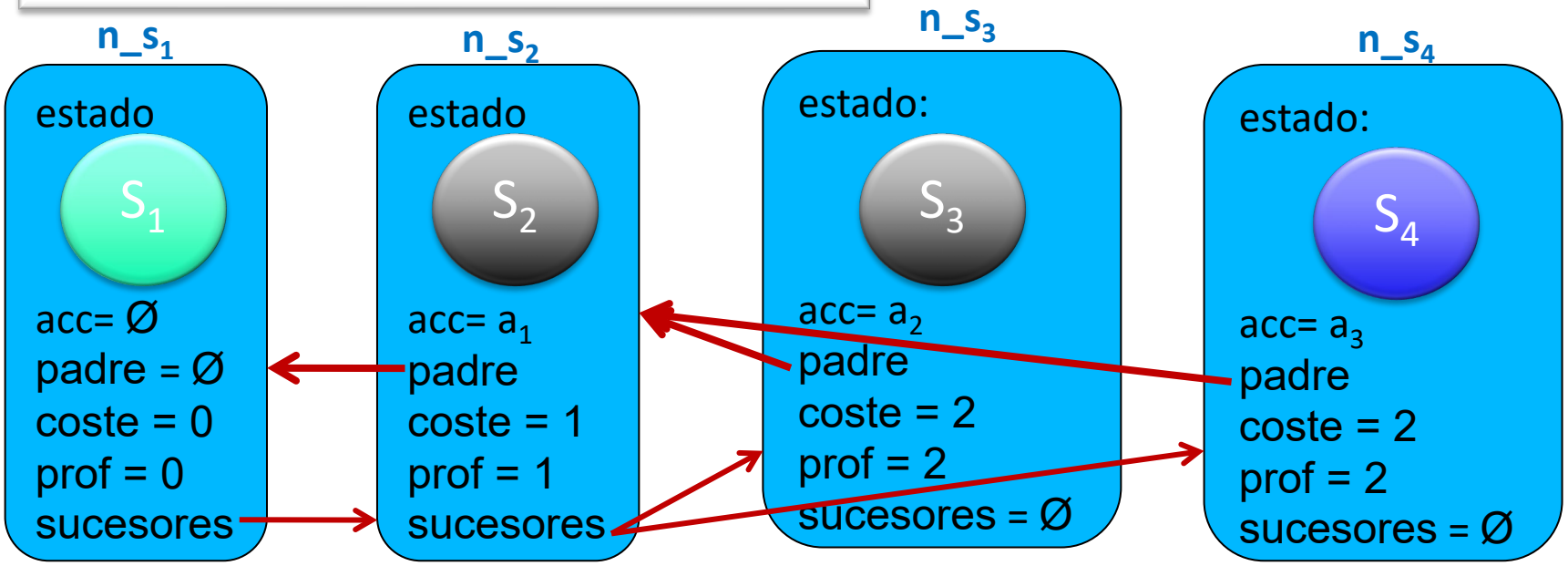
Acciones aplicables en s

Implementación: estados frente a nodos

Espacio de estados:

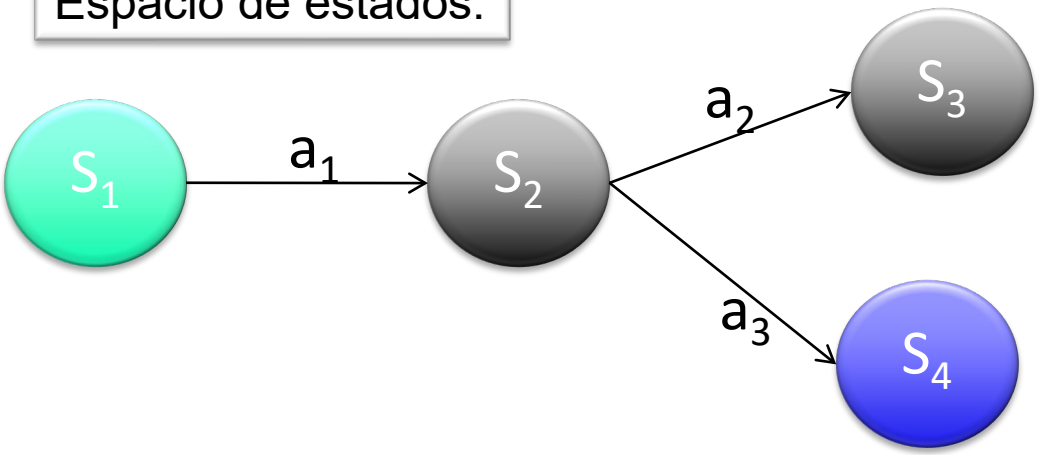


Estructura de nodos (árbol de búsqueda):



Implementación: estados frente a nodos

Espacio de estados:



Estructura de nodos (árbol de búsqueda):

n_s_1

estado 

acc= Ø
padre = Ø
coste = 0
prof = 0
sucesores={**n_s_2**}

n_s_2

estado 

acc= a_1
padre=**n_s_1**
coste = 1
prof = 1
sucesores = {**n_s_3** ,**n_s_4**}

n_s_3

estado: 

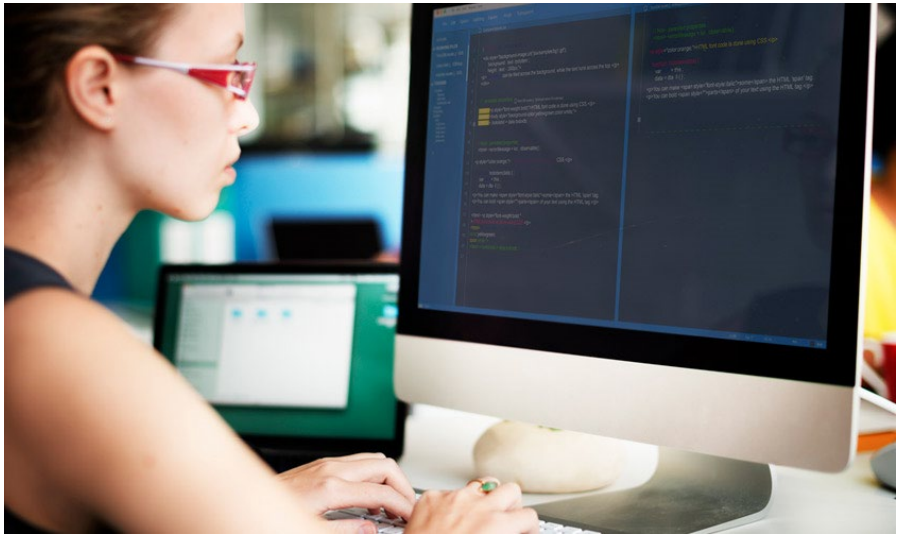
acc= a_2
Padre= **n_s_2**
coste = 2
prof = 2
sucesores = Ø

n_s_4

estado: 

acc= a_3
padre = **n_s_2**
coste = 2
prof = 2
sucesores = Ø

Detalles de implementación



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

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 expandidos: nodos sacados de la frontera para los que se ha obtenido sus sucesores
  - Nodos alcanzados: lista de nodos que han sido creados

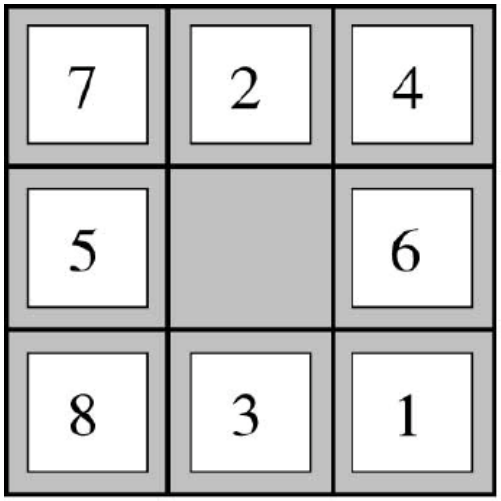
Diferencias con AV

- No partimos de un grafo (lista de nodos o de edges) dado, lo vamos construyendo
  - La estructura de datos que usamos para explorar es la frontera (cerrados nos evita ciclos)
- No recorremos todo el grafo,
  - Exploramos hasta encontrar la primera solución

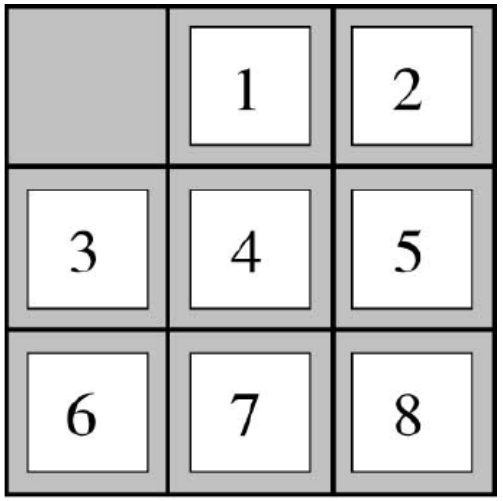
Más ejemplos

- 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/



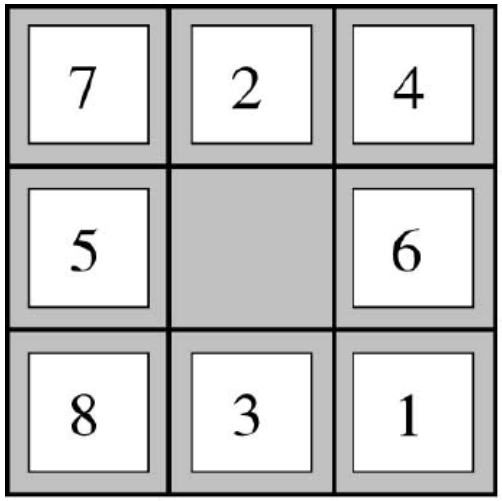
Estado Inicial



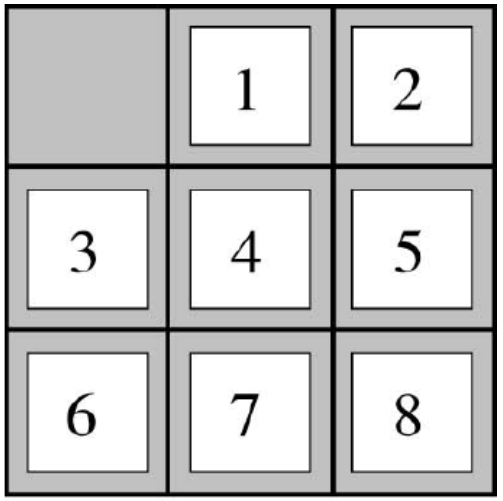
Estado Objetivo

¿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)

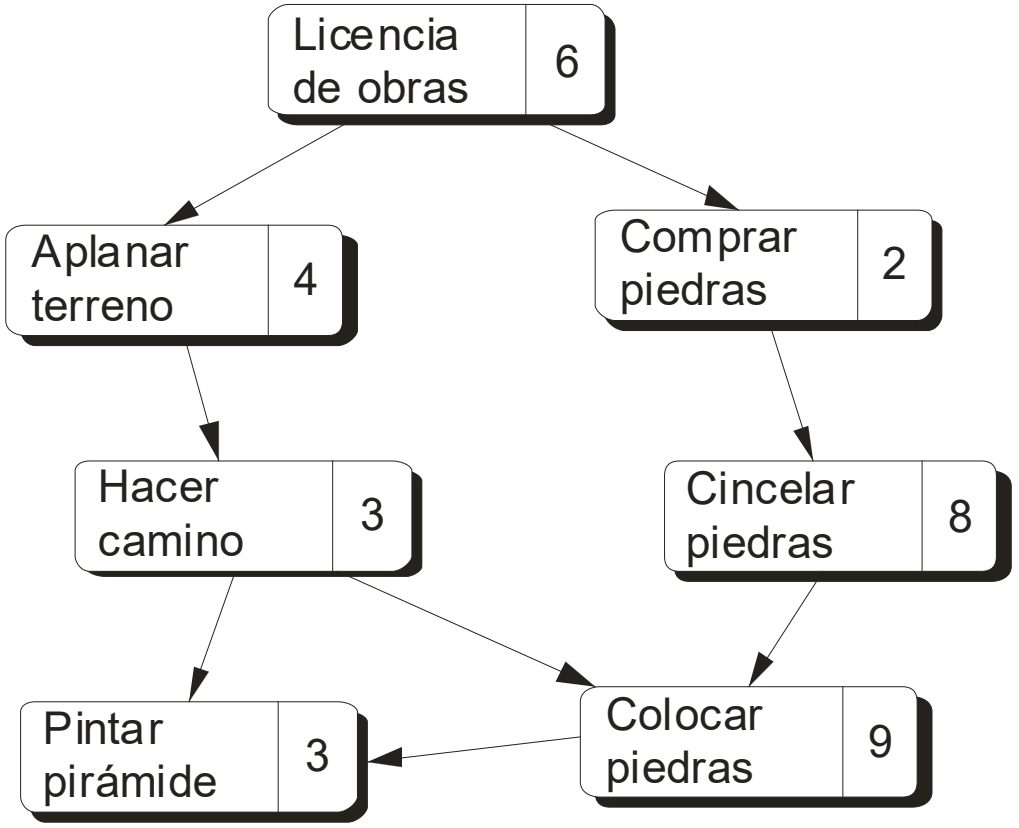
¿Nº estados?

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

¿Test objetivo? = estado objetivo (proporcionado)

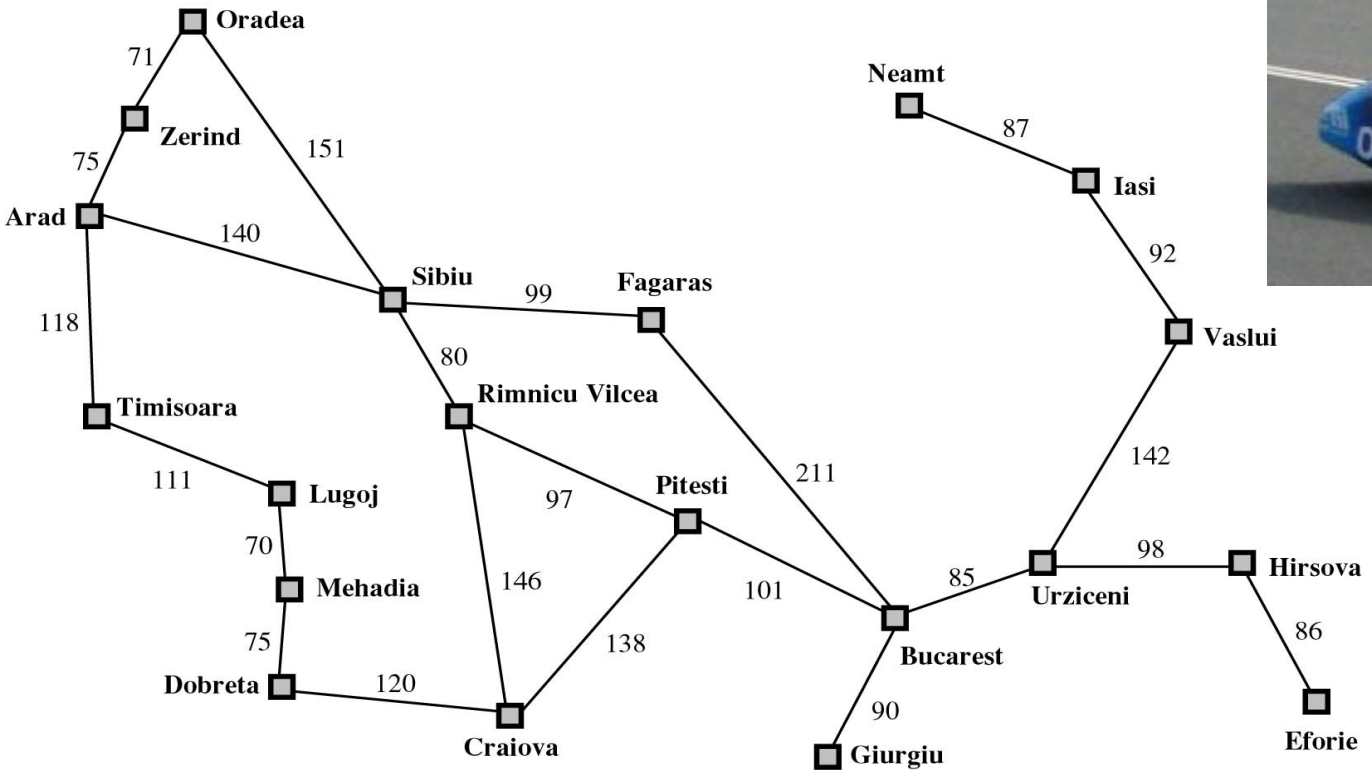
¿Coste del camino? 1 por movimiento

Ej.: planificación de tareas (ED)



Ejemplo: ruta 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.



Ejemplo: ruta 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. Etapas de la resolución del problema

Formulación del objetivo:

?

Formulación del problema:

estados: ?

acciones: ?

Búsqueda de la solución:

secuencia: ?

Ejecución:

?

Ejemplo: ruta 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

función sucesor para cada estado

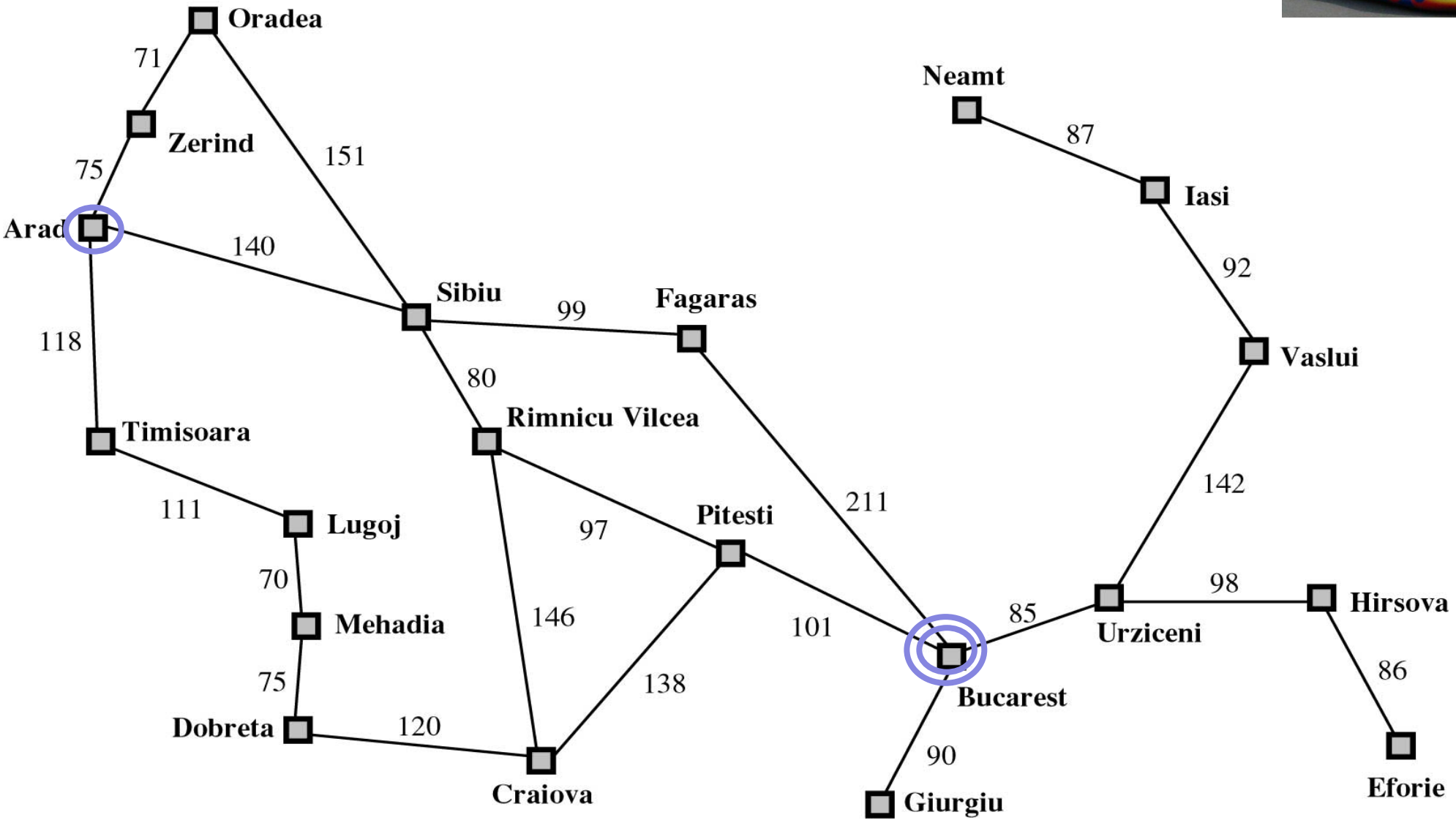
Búsqueda de la solución:

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

Ejecución:

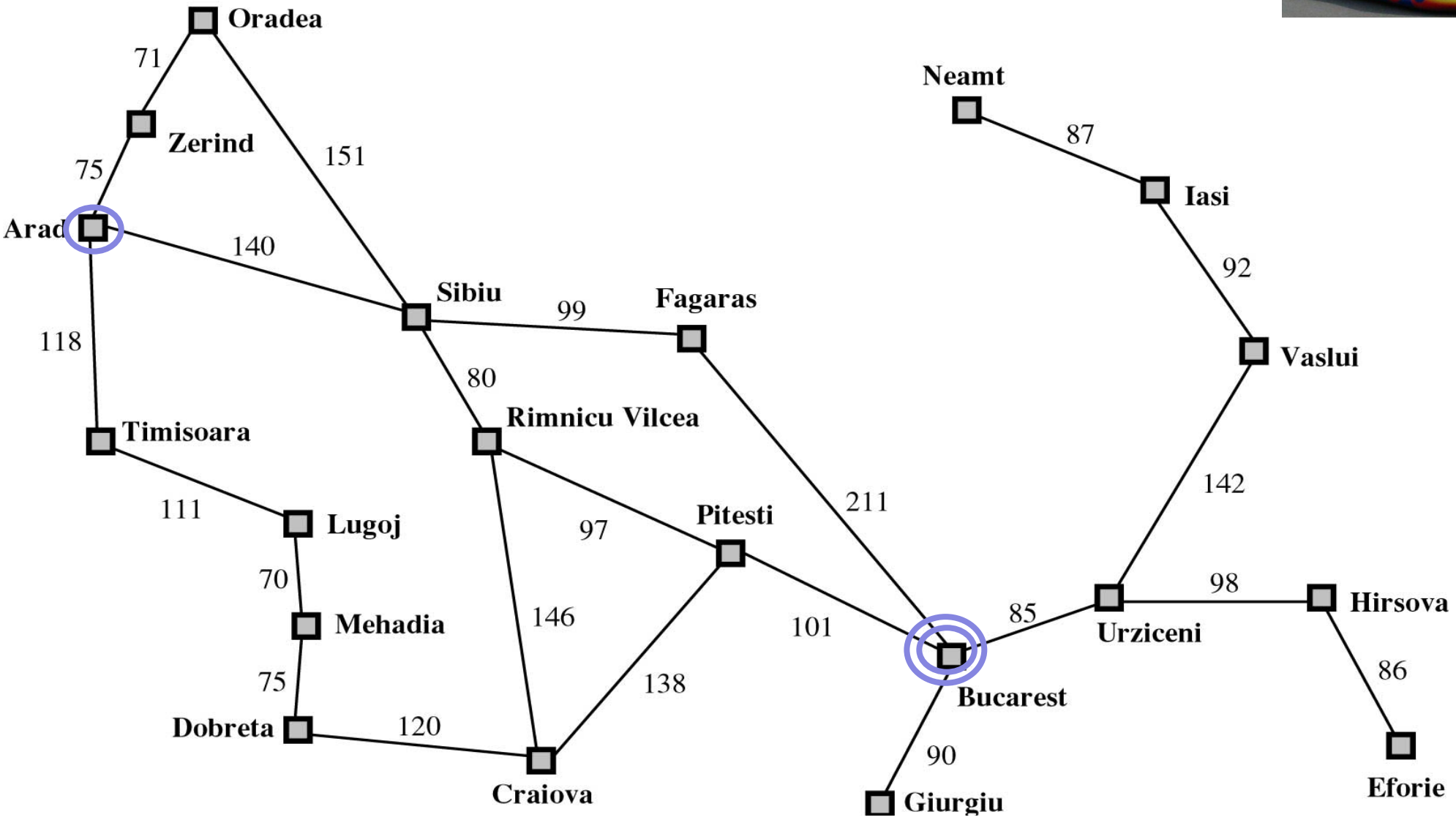
conducir por la secuencia de ciudades (i.e., las rutas entre ciudades)

Ejemplo: Hallar rutas en Rumanía



S(En(Arad)) ?

Ejemplo: Hallar rutas en Rumanía



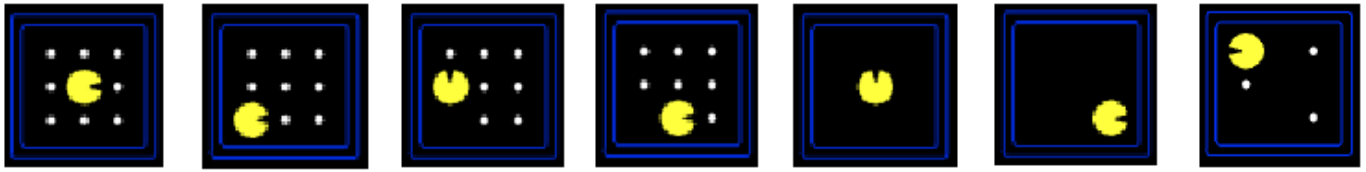
S(En(Arad))={ <Ir(Sibiu),En(Sibiu)>, <Ir(Timisoara), En(Timisoara)>, <Ir(Zerind),En(Zerind)>}

S(En(Sibiu)) ?

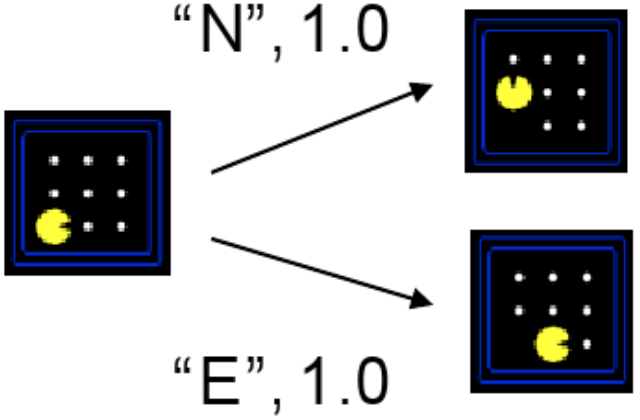
S(En(Neamt))

Problemas de búsqueda

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



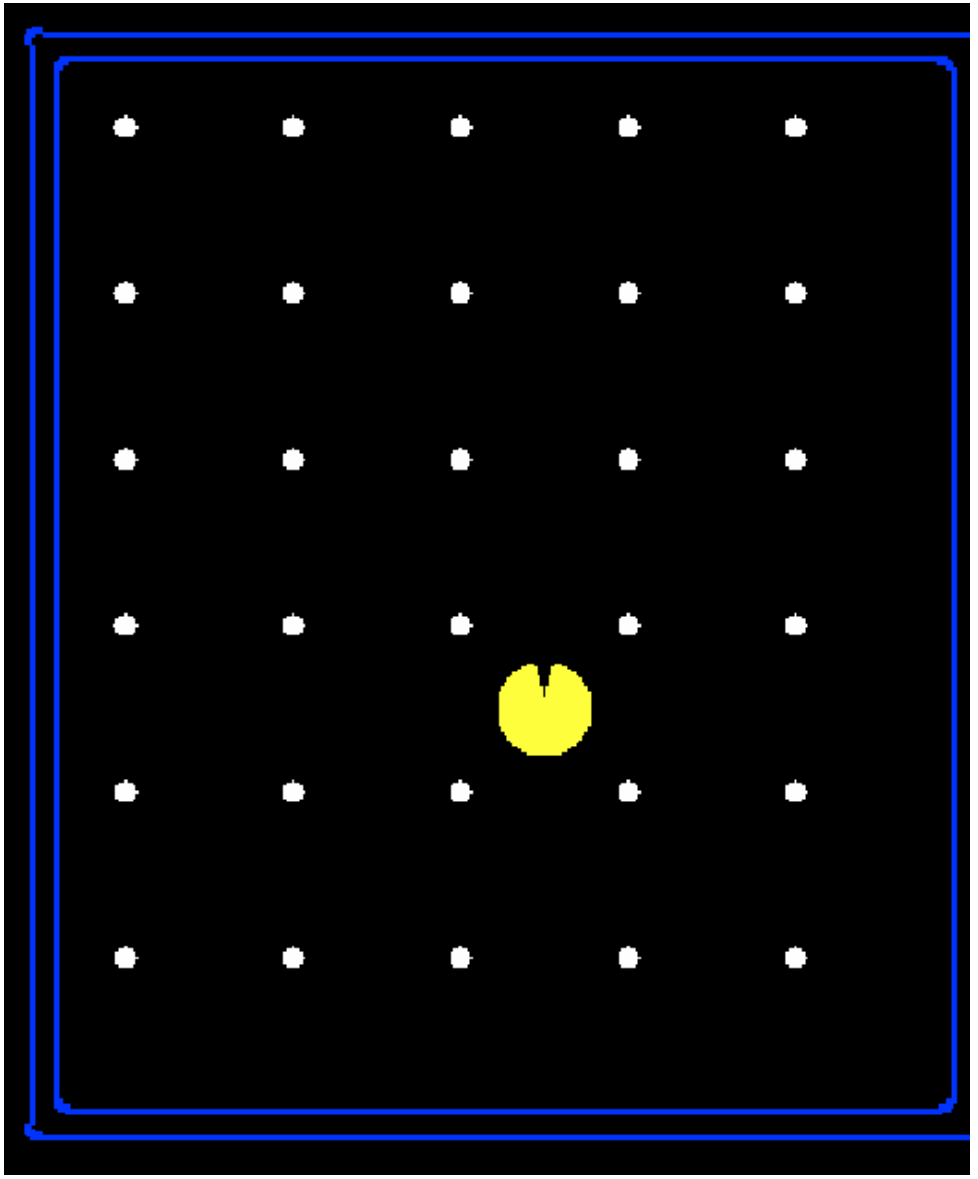
  - Una función **sucesor**:
  - Un **estado inicial** y un test que nos permita saber si hemos llegado al **objetivo**.



- 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



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.