



Inteligencia Artificial

Espacio de estados y búsqueda

Índice

3. Espacio de estados y búsqueda

3.1 Métodos de búsqueda no informados (Uninformed Search Methods):

- Búsqueda en anchura (Breadth-First Search)
- Búsqueda en profundidad (Depth-First Search)
- British Museum
- Búsqueda de coste uniforme (Uniform-Cost Search)

3.2 Métodos de búsqueda informados: (Informed Search Methods):

- Heurísticos
- Búsqueda voraz (Greedy Search)
- Búsqueda en haz (Beam-Search)
- Búsqueda A* (A* or A star search)
- Grafos AND / OR

3.3 Búsqueda adversarial

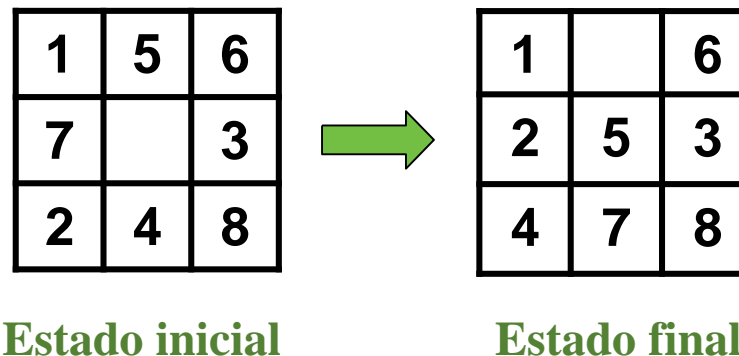
- Minimax
- Alfa-beta
- Expectimax

Problemas y espacio de estados

- Muchos problemas que pueden resolverse aplicando técnicas de Inteligencia Artificial, se modelan en forma **simbólica** y **discreta**, definiendo las configuraciones posibles del universo que describe el problema.
- El *problema* se plantea en términos de encontrar una *configuración objetivo* a partir de una *configuración inicial* dada, aplicando *transformaciones válidas* según el modelo del universo. La *respuesta* es la secuencia de transformaciones cuya aplicación sucesiva lleva a la configuración deseada.

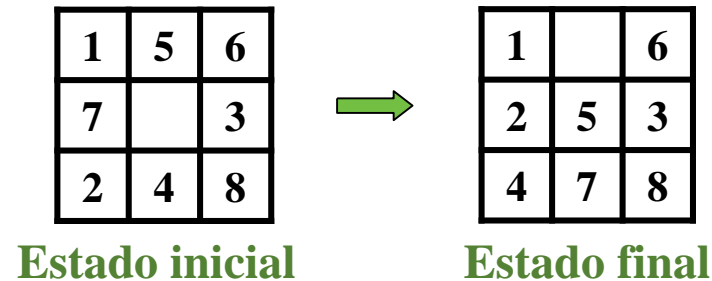
Estados y reglas del problema

- Los **ejemplos más característicos** de esta categoría de problemas **son los juegos** (que son universos restringidos fáciles de modelar).
- En un juego, las configuraciones del **universo** corresponden directamente a las configuraciones del **tablero**. Cada configuración es un **estado** que puede ser esquematizado gráficamente y representado en forma simbólica.
- Las transformaciones permitidas corresponden a **las reglas o movimientos** del juego, formalizadas como **transiciones de estado**.



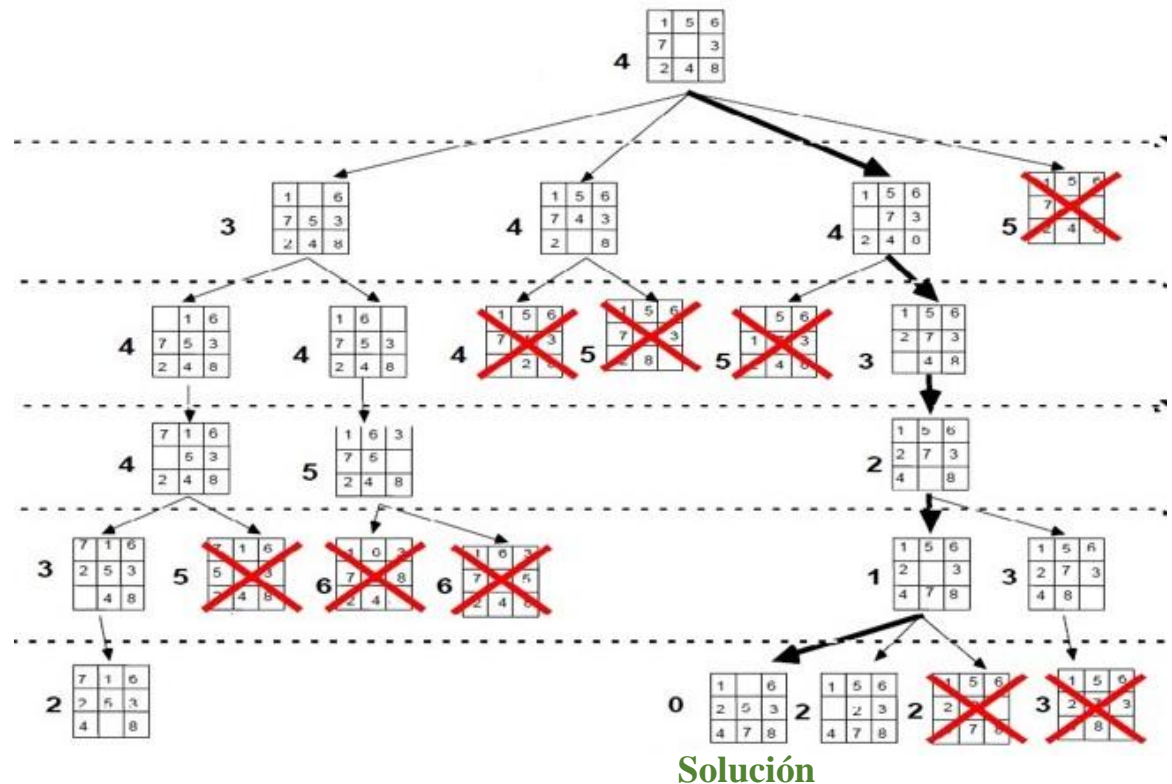
Soluciones de los problemas

➤ Problemas con más de una solución válida:



➤ Entre las soluciones posibles ...

- buscar la menos costosa
- buscar la más rápida



Estados y operadores

- Formalmente, un *espacio de estados* se define por una cuádrupla $[N, A, I, F]$ donde:
- N es un conjunto de nodos que representan estados en el proceso de resolución de un problema
 - A es un conjunto de arcos entre nodos, que corresponden a los posibles pasos (aplicación de un *operador*) en el proceso de resolución de un problema
 - I es un subconjunto no vacío de N que contiene el/los *estados iniciales* del problema
 - F es un subconjunto no vacío de N que contiene el/los *estados finales* del problema

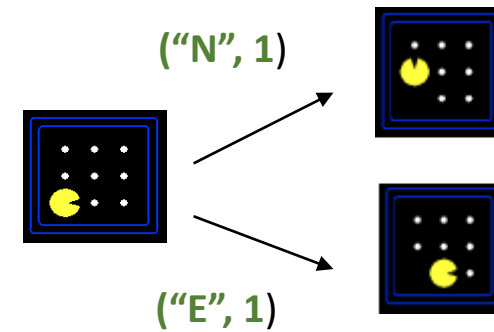
Problemas de búsqueda

➤ Un **problema de búsqueda** consiste en:

- Un espacio de estados



- Una **función** sucesor (con **acciones**, **costes**)



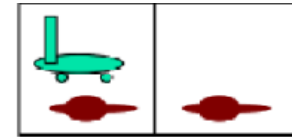
- Un estado **inicial** y un **test** de haber llegado al objetivo (**goal**)

➤ Una solución **es una secuencia de acciones** (un plan) que **transforma el estado inicial en un estado objetivo** (goal state)

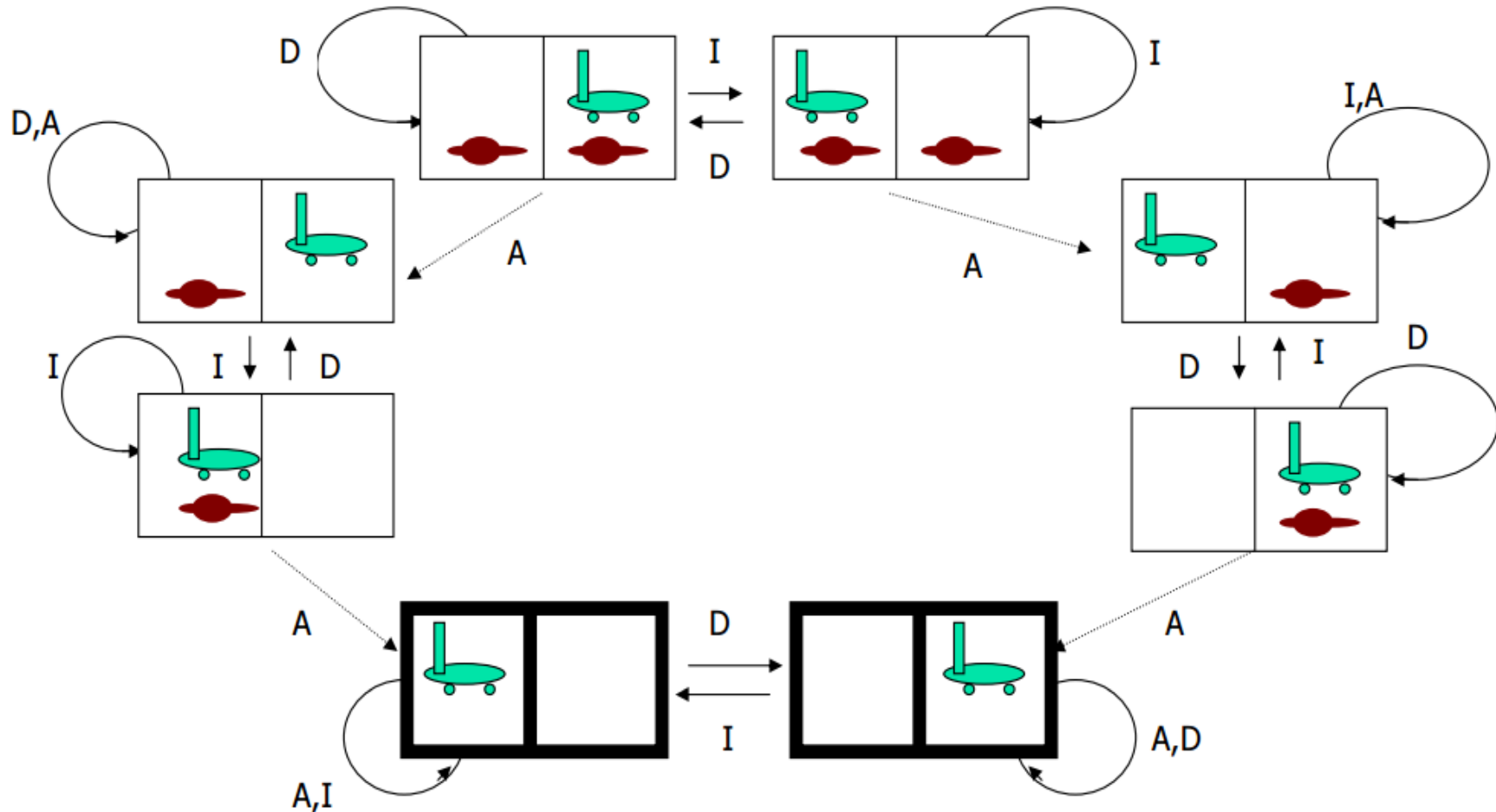
Ejemplo: aspiradora

➤ Dos casillas. Cada casilla:

- **Limpia o sucia**
- **Estado:** (situación de casillas, aspiradora)
- **Operaciones:**
 - Izquierda / Derecha / Aspirar
- **Objetivo:** limpiar todo
- **Estado inicial:** aspiradora en cualquier casilla



Aspiradora: conjunto de estados



Caracterización de los problemas

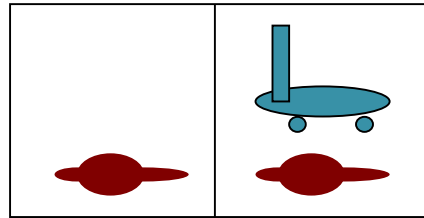
- ¿El problema **tiene siempre solución**?
- ¿**En qué consiste** realmente **la solución** del problema?
- ¿Basta con encontrar **una solución cualquiera** o se busca la **solución óptima**?
- ¿**Se puede descomponer** el problema?
- ¿**Es posible** ignorar o **deshacer** pasos en la solución?
- ¿Pueden darse **ciclos** en el camino hacia la solución?
- ¿**Es predecible** el universo del **problema**?
- ¿Cuál es el **papel del conocimiento**?
- ¿**Requiere** el problema la **interacción** con una **persona**?

¿En qué consiste realmente la solución del problema?

- **La solución del problema** consiste en alcanzar el estado final del problema
- **Una solución** para un problema **viene dada por la secuencia de operadores** que se han aplicado **para pasar del estado inicial** hasta el **estado objetivo**
 - El espacio de estados del mundo de la aspiradora es un **autómata finito**
 - Por ejemplo, las infinitas cadenas **$D^* A D^* I I^* A (D U I)^*$** son soluciones del problema a partir de la configuración inicial

¿Basta con encontrar una solución cualquiera o se busca la solución óptima?

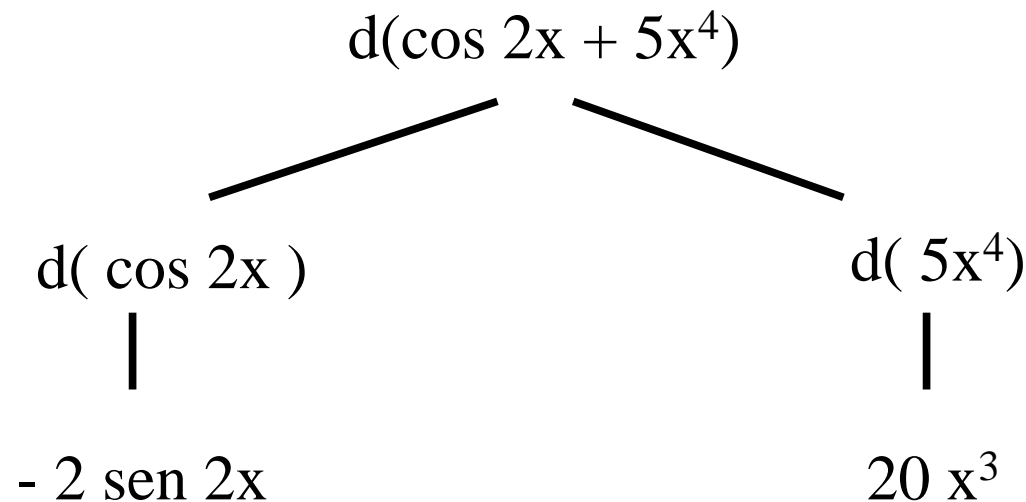
- En el mundo de la aspiradora asignamos un **coste de 1 a cada operación**, que por ejemplo, **representa el coste de la electricidad** gastada en el movimiento.



- La **solución óptima**, es **AIA** con **coste 3** a partir de la configuración inicial:
- Cualquier solución más larga tendrá un coste mayor.

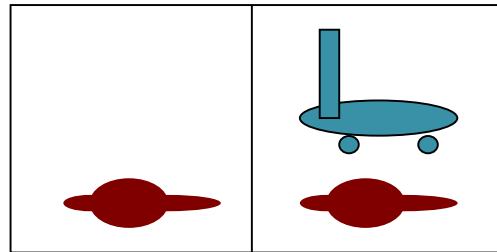
¿Se puede descomponer el problema?

- Existe un determinado conjunto de problemas (a los que **llamaremos divisibles**) a los cuales se puede aplicar la técnica de descomposición ó “divide y vencerás”.
- Por ejemplo, el problema de **hallar una derivada**



¿Pueden darse ciclos en el camino hacia la solución?

- El sistema **debe memorizar los estados** por los que ha pasado anteriormente **para evitar ciclos innecesarios**.
- Ejemplo del mundo de la aspiradora: **(I+D+)*** son movimientos que **dejan a la aspiradora en la misma posición**



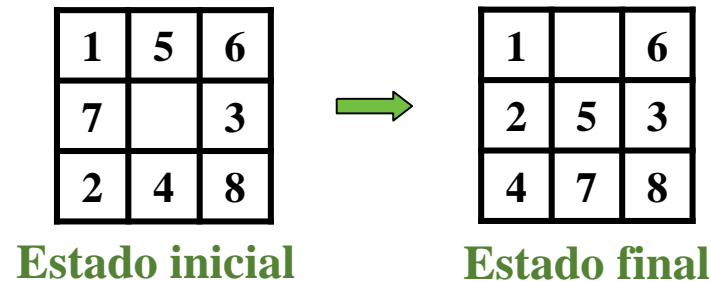
¿Es posible ignorar o deshacer pasos de la solución?

- Problemas en los cuales, **los pasos dados** en la búsqueda hacia la solución **se pueden deshacer**: por ejemplo, recorrer un laberinto.
- Problemas en los que **una operación es irreversible**: por ejemplo, realizar una jugada en cualquier tipo de juego.
 - Carta en la mesa pesa!
 - Ficha movida ... movida está!

¿Es predecible el universo del problema?

➤ Problemas en los que **puede** planearse la secuencia de movimientos que llevarán hasta un **estado objetivo**, puesto que **se sabe perfectamente como actúan los operadores** sobre un estado.

– Ejemplo: el problema del **puzzle**



➤ Problemas en los que **no se conoce con precisión** cual va a ser el resultado de cada **movimiento** que se hace:

– Ejemplo: el juego del **mus** ➡ interviene el azar y los movimientos del resto de los jugadores

¿Cuál es el papel del conocimiento?

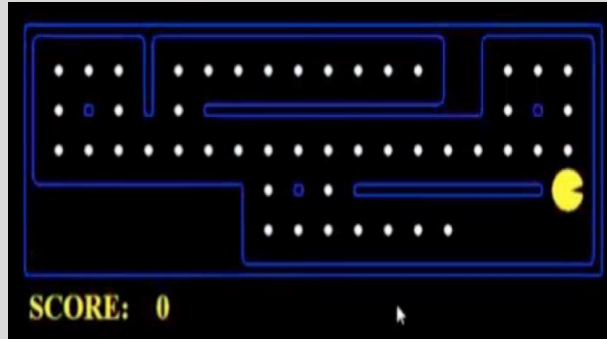
- En algunos problemas **es importante el conocimiento** para restringir la búsqueda de la solución.
 - Ejemplo del **ajedrez**: conocimiento plasmado en tácticas y estrategias basadas en la experiencia de los jugadores.
- En otros problemas **se precisa un conocimiento muy amplio** tan sólo **para poder reconocer la solución**.
 - Ejemplo: responder a una pregunta formulada en lenguaje natural.

¿Requiere el problema la interacción con una persona?

- Problemas en los que la computadora recibe una descripción del problema y produce una respuesta **sin necesitar ningún dato adicional** ni interacción alguna con el usuario.
 - Ejemplo: **El puzzle** de las ocho piezas tan solo **requiere** del exterior **una configuración inicial y otra configuración final**, y nos devuelve una secuencia de movimientos que son la solución.
- Conversacionales: habrá comunicación entre una persona y el ordenador, bien para **proporcionar asistencia adicional** a la computadora, bien para proporcionar **información adicional** al usuario, **o ambas**.
 - Ejemplo: tutores inteligentes

¿Qué tiene un espacio de estados?

El **estado del mundo** incluye todos los detalles del entorno



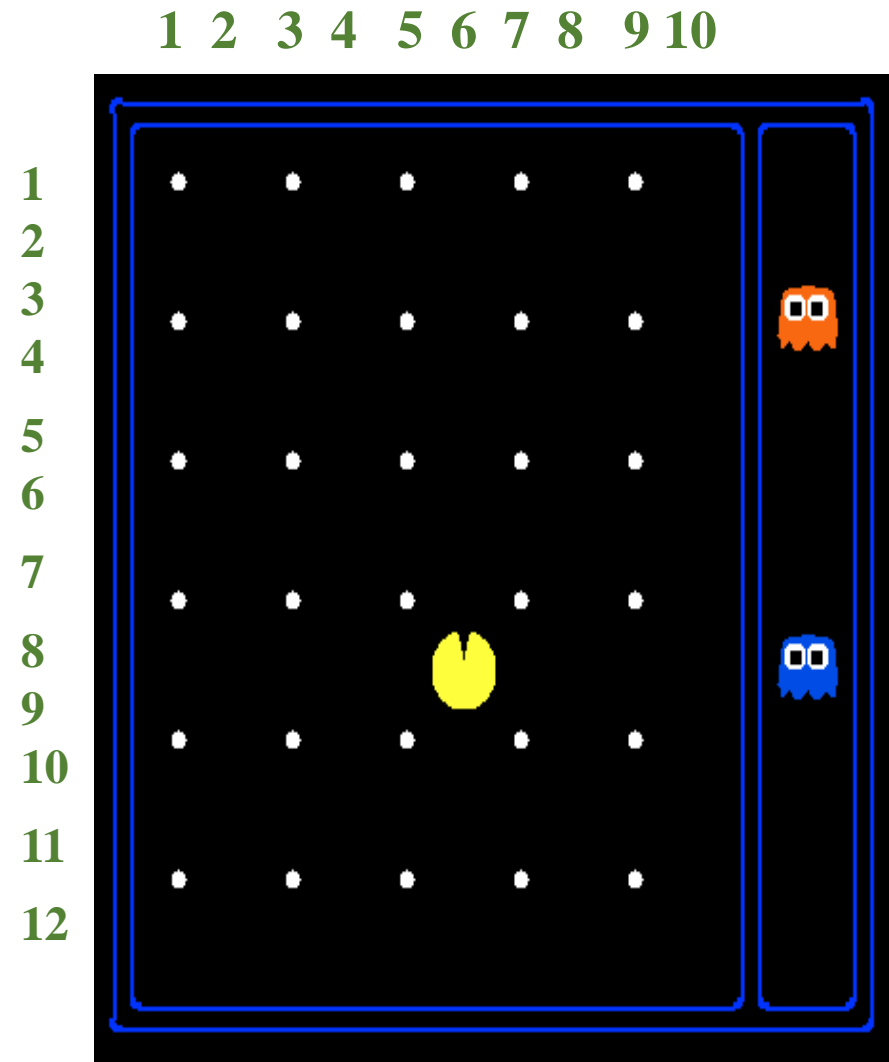
Un **estado de búsqueda contiene únicamente** los detalles necesarios para la planificación (abstracción)

- **Problema: buscar camino**
 - Estados: (x,y) localización
 - Acciones: NSEW
 - Sucesor: cambiar localización
 - Test de objetivo: $\text{is } (x,y) = \text{END}$
- **Problema: comer todas las bolitas**
 - Estados: $\{(x,y), \text{booleanos (puntos)}\}$
 - Acciones: NSEW
 - Sucesor: cambiar localización y posiblemente un booleano (punto)
 - Test de objetivo: todos los puntos son false

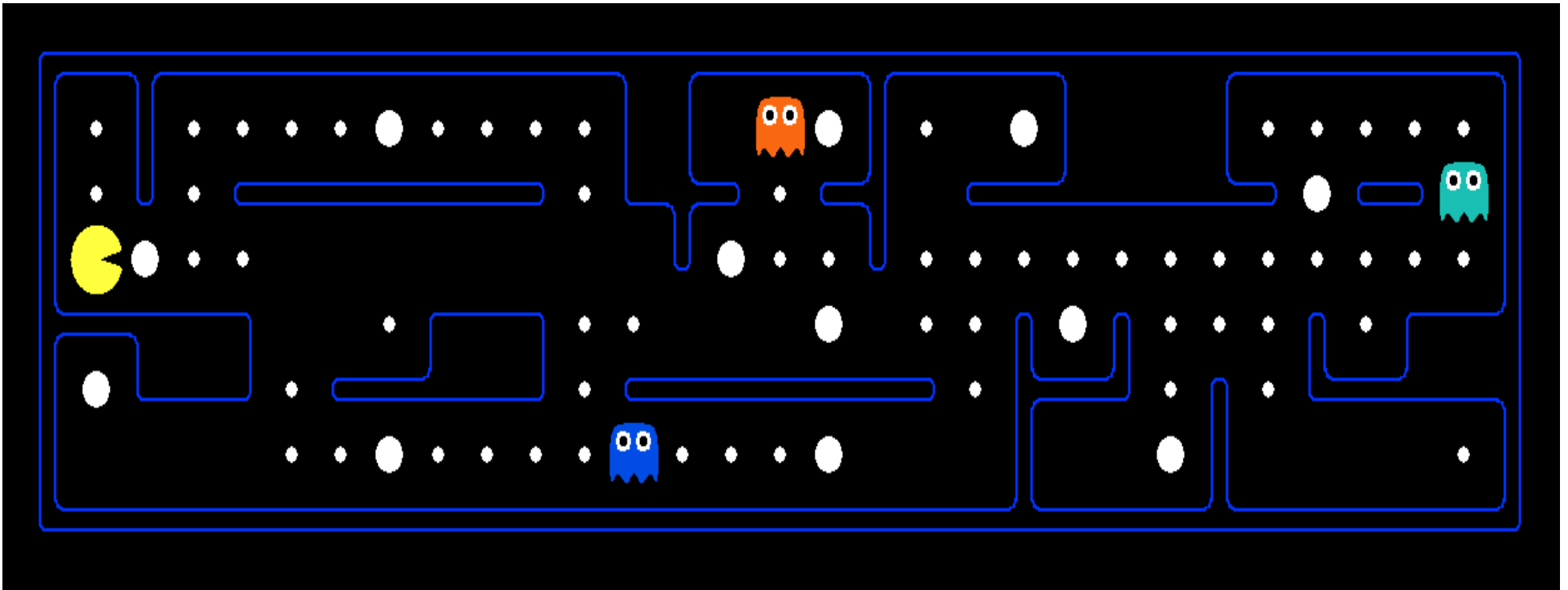
¿Tamaño del espacio de estados?

➤ Estados del mundo:

- Posiciones del agente: **120**
- Número de comidas: **30**
- Posiciones de fantasma: **12**

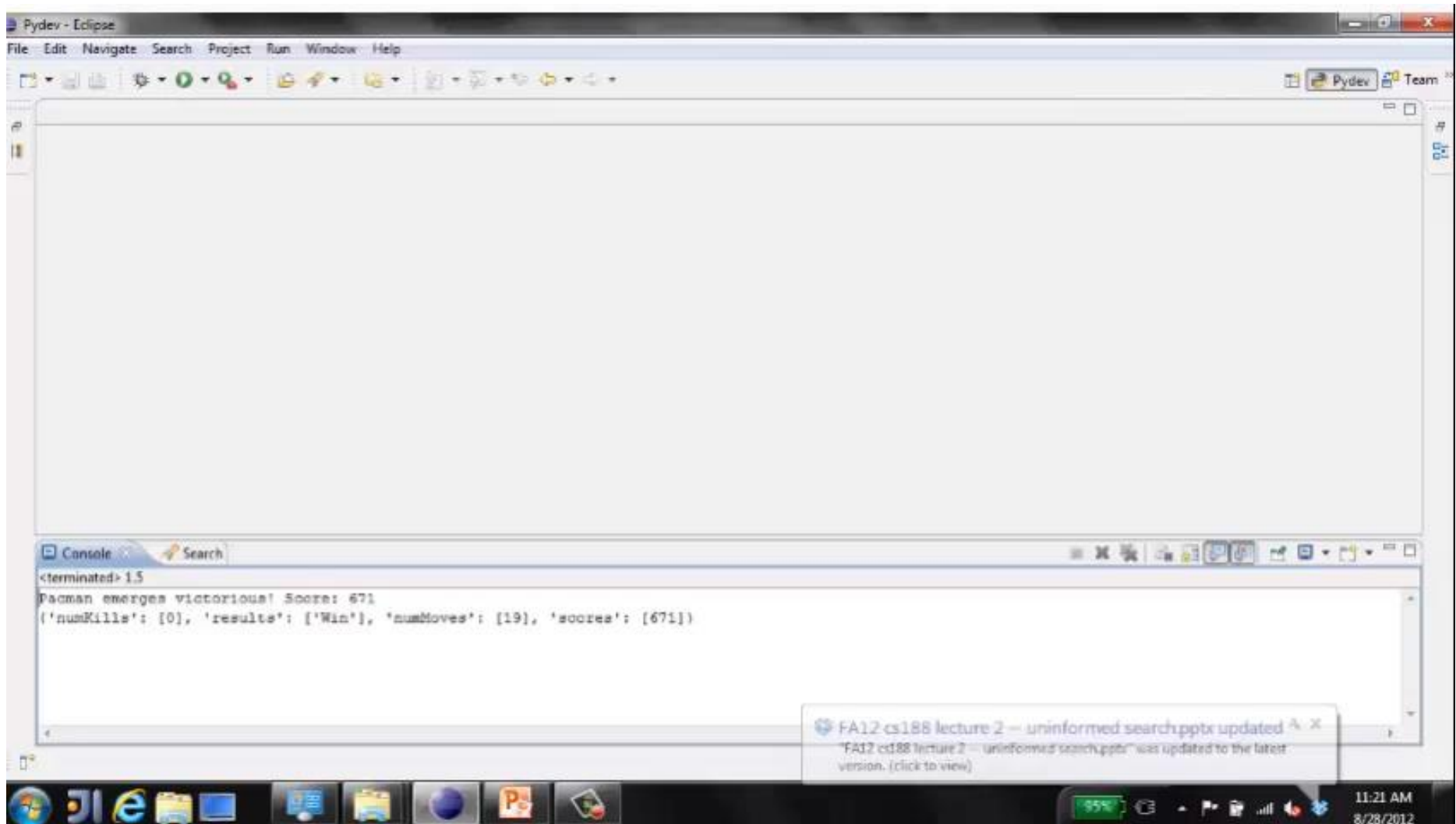


Quiz: Camino libre



- Problema: **comer todos los puntos**
- ¿Qué debe especificar el espacio de estados?
 - (posición del agente, booleanos para puntos, booleanos para bolitas de poder, tiempo restante de susto)

Vídeo de un agente reactivo

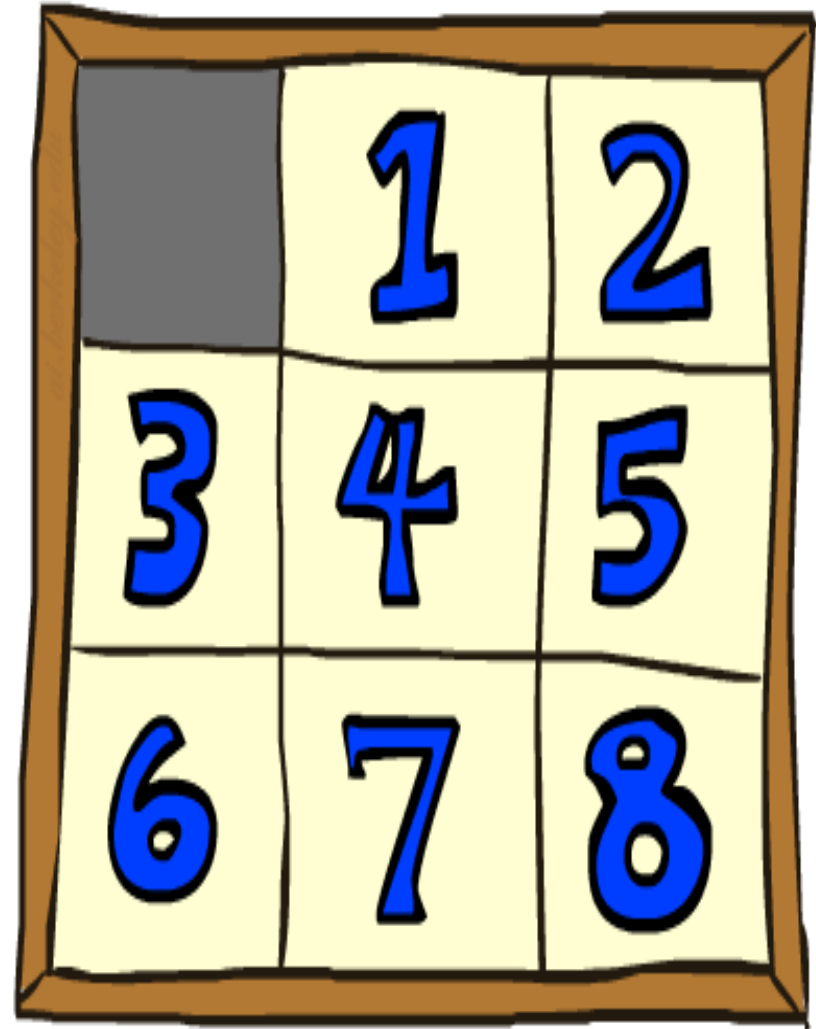
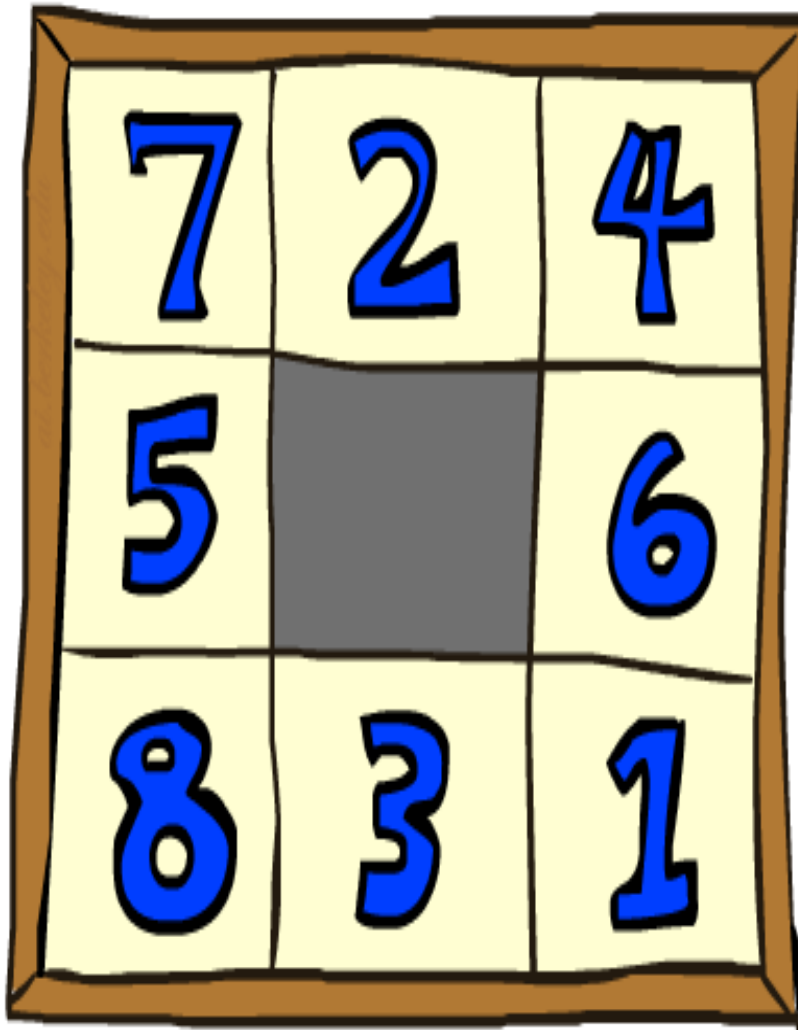


The screenshot shows the Pydev Eclipse IDE interface. The main editor area is empty. The console window at the bottom displays the following output:

```
<terminated> 1.5  
Pacman emerges victorious! Score: 671  
{'numKills': [0], 'results': ['Win'], 'numMoves': [19], 'scores': [671]}
```

A system notification bubble is visible in the bottom right corner of the screen, stating: "FA12 cs188 lecture 2 - uninformed search.pptx updated" and "FA12 cs188 lecture 2 - uninformed search.pptx was updated to the latest version. (click to view)". The Windows taskbar at the bottom shows the time as 11:21 AM on 8/28/2012.

Grafos de espacio de estados y árboles de búsqueda



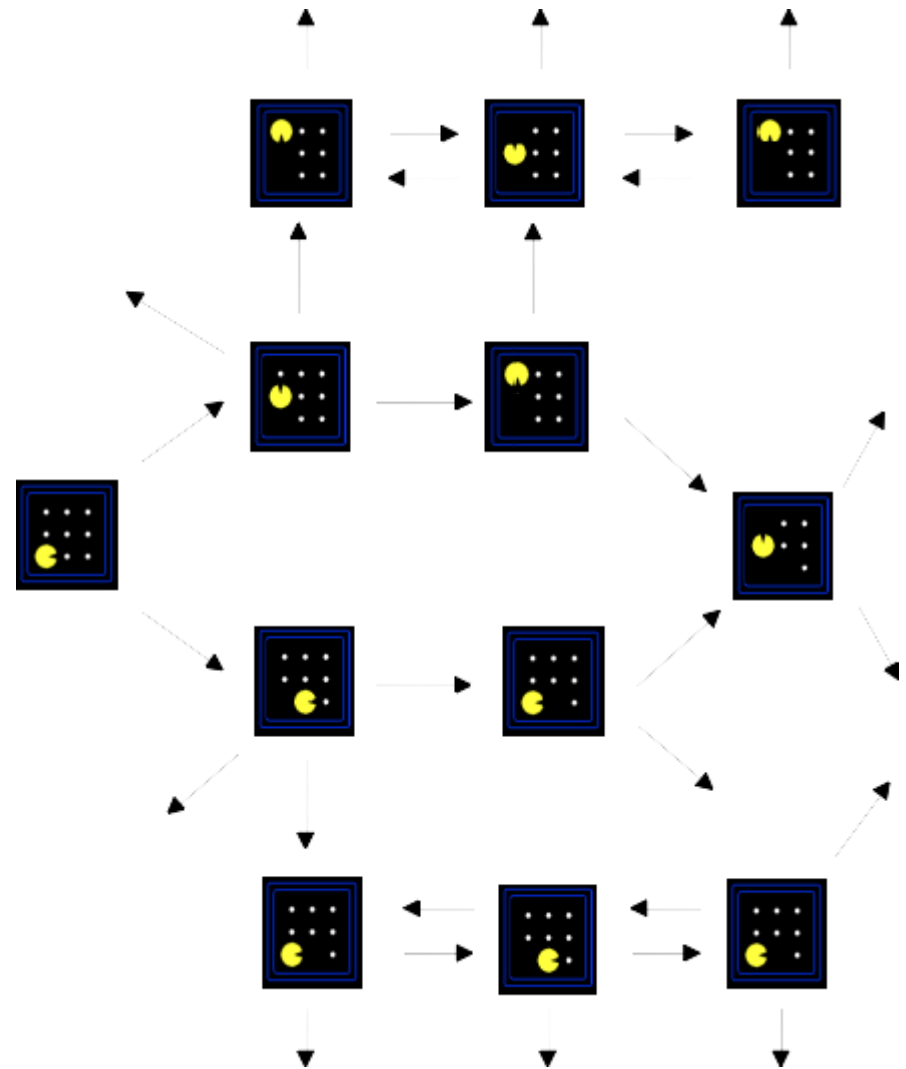
Representación: grafos de espacio de estados

➤ Grafo de espacio de estados: **representación matemática de un problema de búsqueda**

- Los **nodos** son configuraciones (abstractas) del mundo
- Los **arcos** representan sucesores (resultados de acción)
- El **test de objetivo** es un conjunto de nodos objetivo (puede ser solo uno)

➤ En un **grafo de estados**, cada estado aparece una sola vez

➤ Pocas veces podremos construir este **grafo** en memoria (**demasiado grande**), pero es una idea útil.



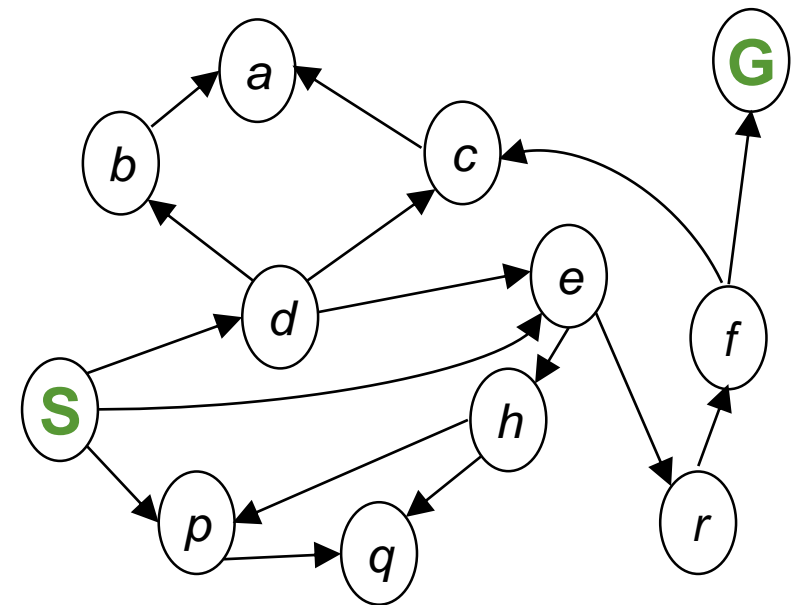
Representación: grafos de espacio de estados

➤ Grafo de espacio de estados: **representación matemática de un problema de búsqueda**

- Los **nodos** son configuraciones (abstractas) del mundo
- Los **arcos** representan sucesores (resultados de acción)
- El **test de objetivo** es un conjunto de nodos objetivo (puede ser solo uno)

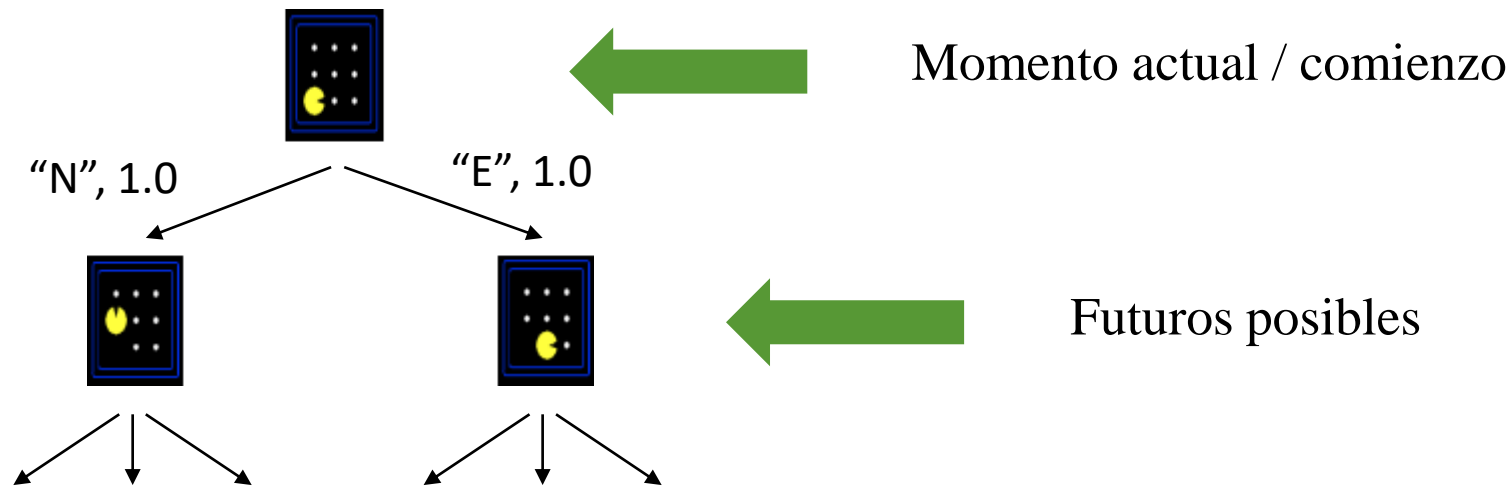
➤ En un **grafo de estados**, cada estado aparece una sola vez

➤ Pocas veces podremos construir este **grafo** en memoria (**demasiado grande**), pero es una idea útil



*Grafo de estados
pequeño para un
problema de búsqueda
pequeño*

Representación: árboles de búsqueda

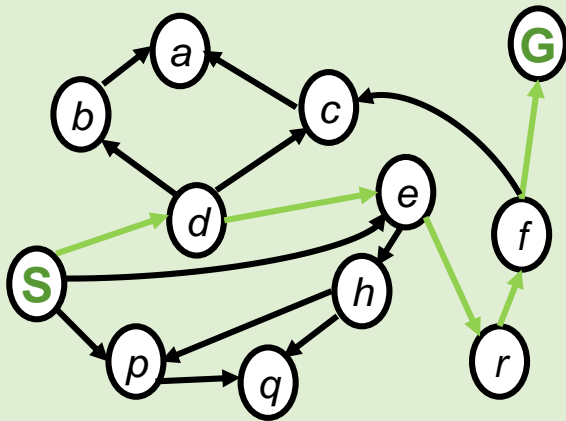


➤ Árbol de búsqueda:

- El **estado inicial** es el **nodo raíz**
- Los **hijos** son los **sucesores**
- Los **nodos muestran estados**, pero corresponden a **PLANES** que llevan a esos estados
- Para la mayoría de problemas, **nunca podremos construir realmente ese árbol (demasiado grande)**

Grafos de espacio de estados vs. árboles de búsqueda

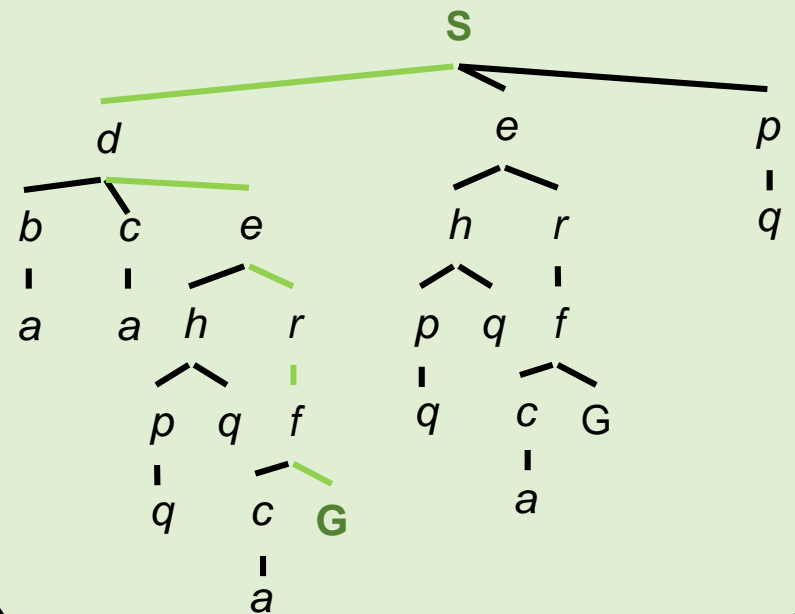
Grafo de espacio de estados



Cada nodo en el árbol de búsqueda es un camino entero en el grafo

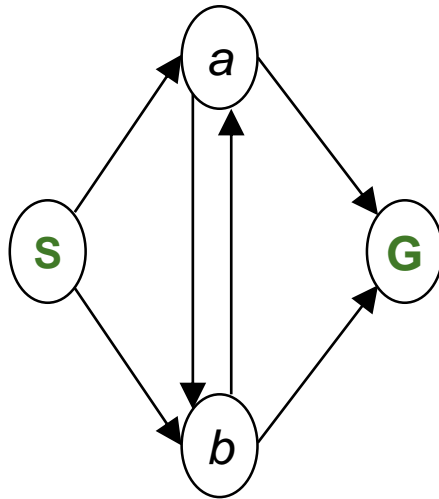
Se construyen a demanda – construyendo lo mínimo posible

Árbol de búsqueda

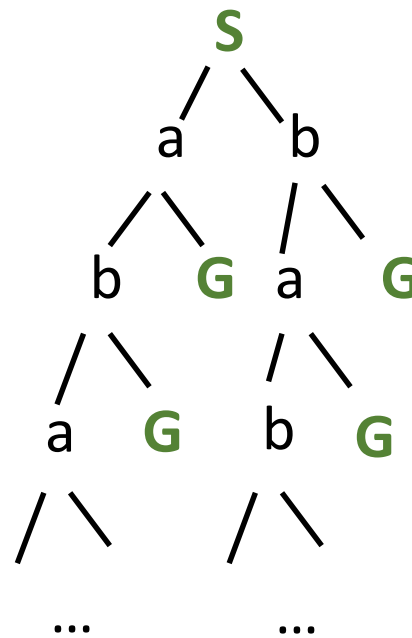


Grafos de espacio de estados vs. árboles de búsqueda

Consideremos este grafo con 4 estados:

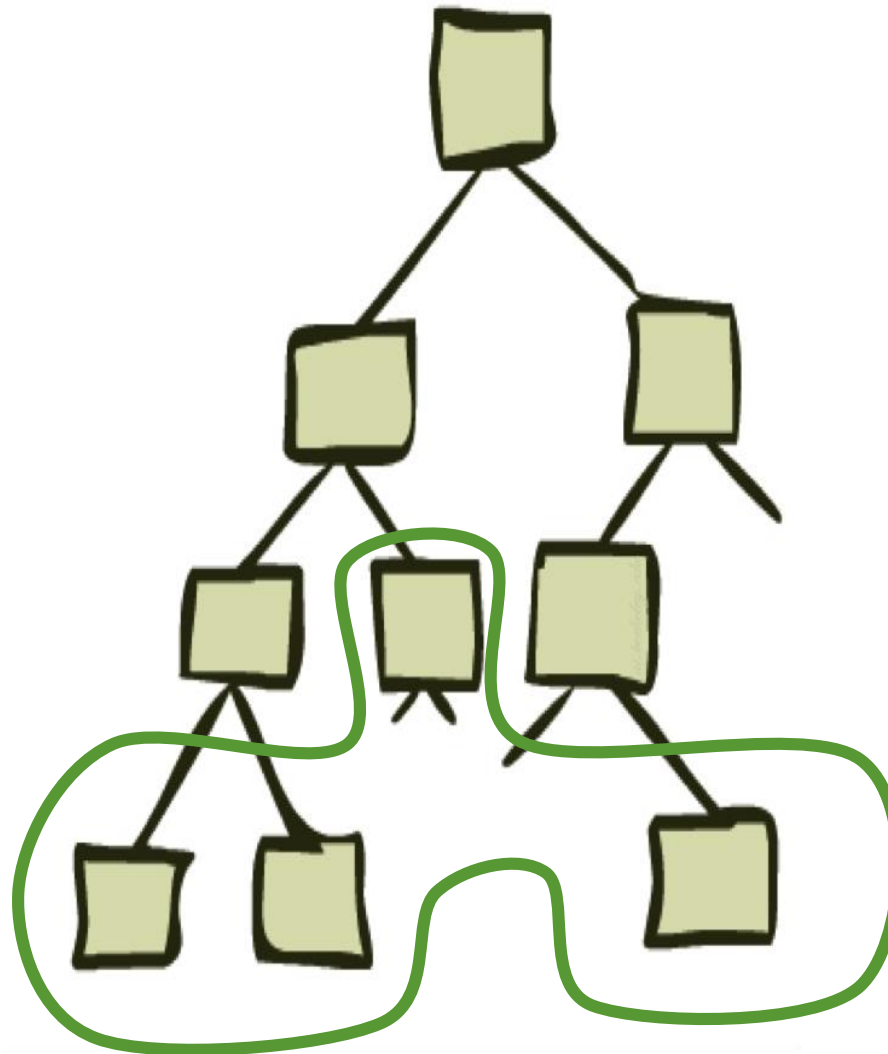


¿Cómo de grande es su árbol de búsqueda?



Importante: ¡El árbol de búsqueda tiene muchas estructuras repetidas!

Búsqueda en árbol

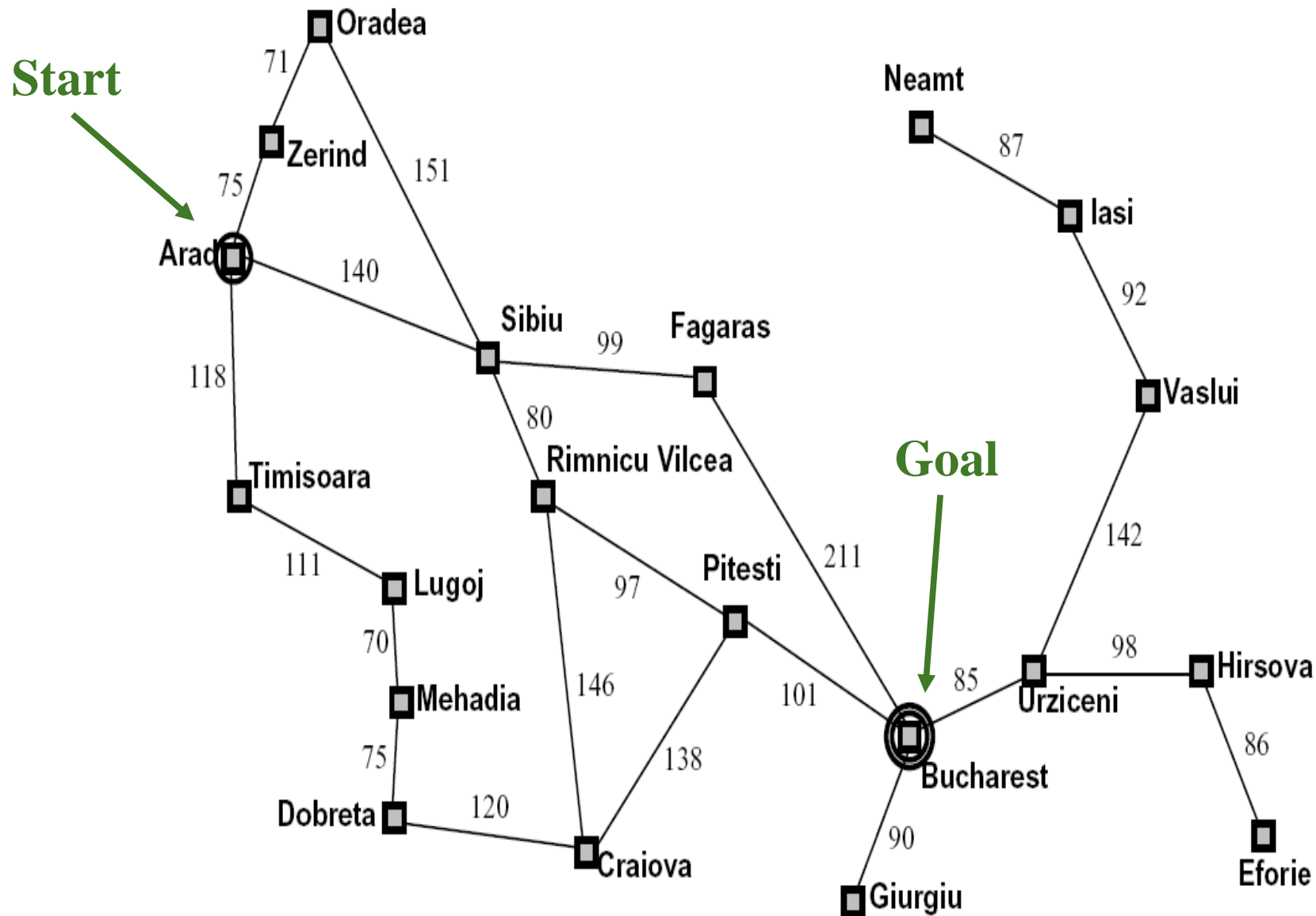


Métodos de resolución de problemas.

Técnicas de búsqueda

- Métodos de búsqueda no informados (**Uninformed Search Methods**):
 - Búsqueda en profundidad (Depth-First Search)
 - Búsqueda en anchura (Breadth-First Search)
 - British Museum
 - Búsqueda de coste uniforme (Uniform-Cost Search)

Ejemplo de búsqueda: Rumanía



Búsqueda en árbol

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe ← INSERT(child-node, fringe)
    end
  end
end
```

➤ Idea:

- Expandir planes potenciales (nodos del árbol)
- Mantener un **borde (fringe)** de planes parciales en consideración
- Intentar expandir el **mínimo número posible de nodos del árbol**

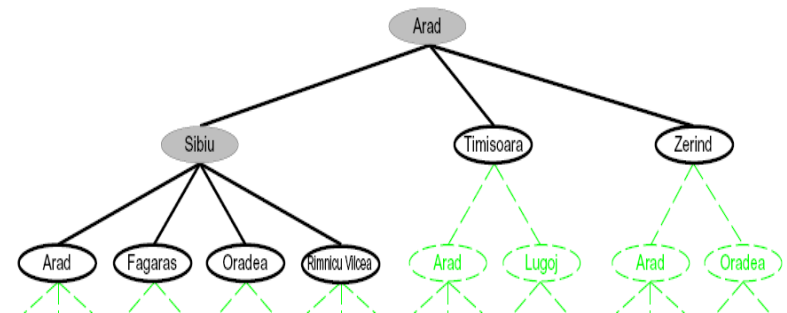
Búsqueda en grafos

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
      end
    end
  end
```

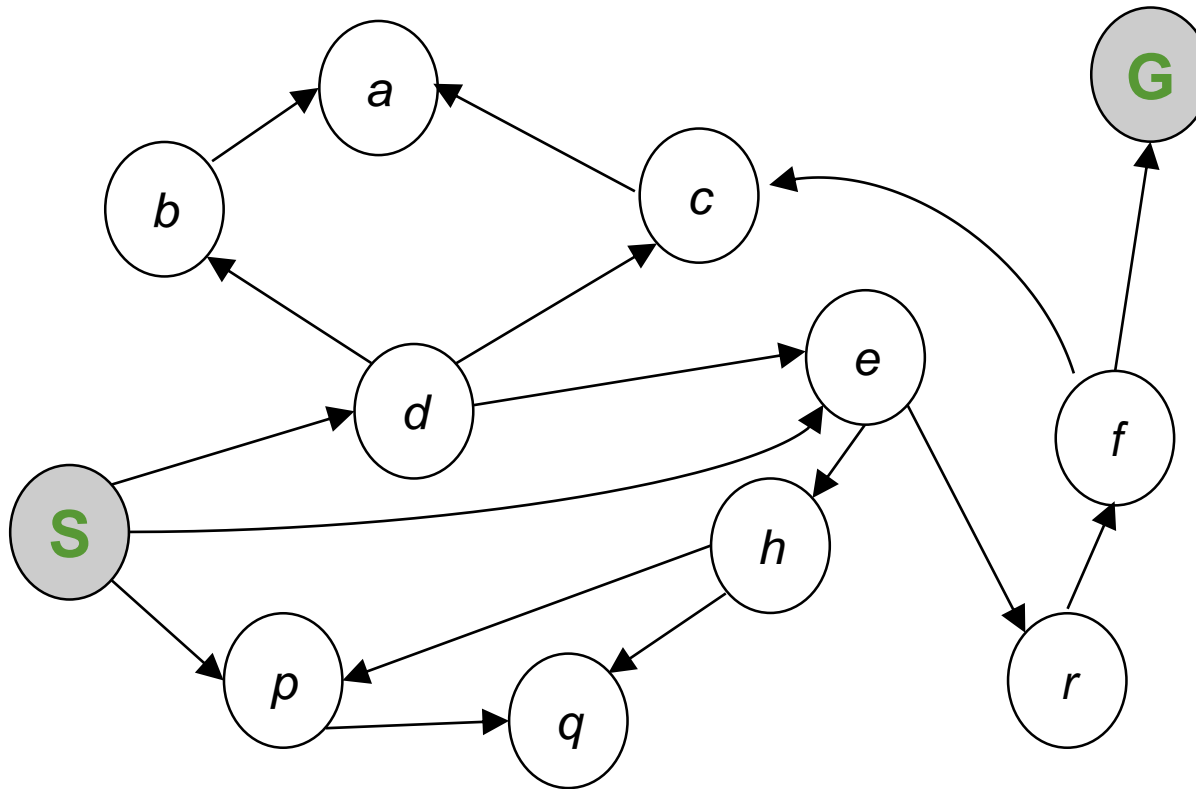
➤ Idea:

- Expandir planes potenciales (nodos del árbol)
- Mantener un **borde (fringe)** de planes parciales en consideración
- Intentar expandir el **mínimo número posible de nodos del árbol**

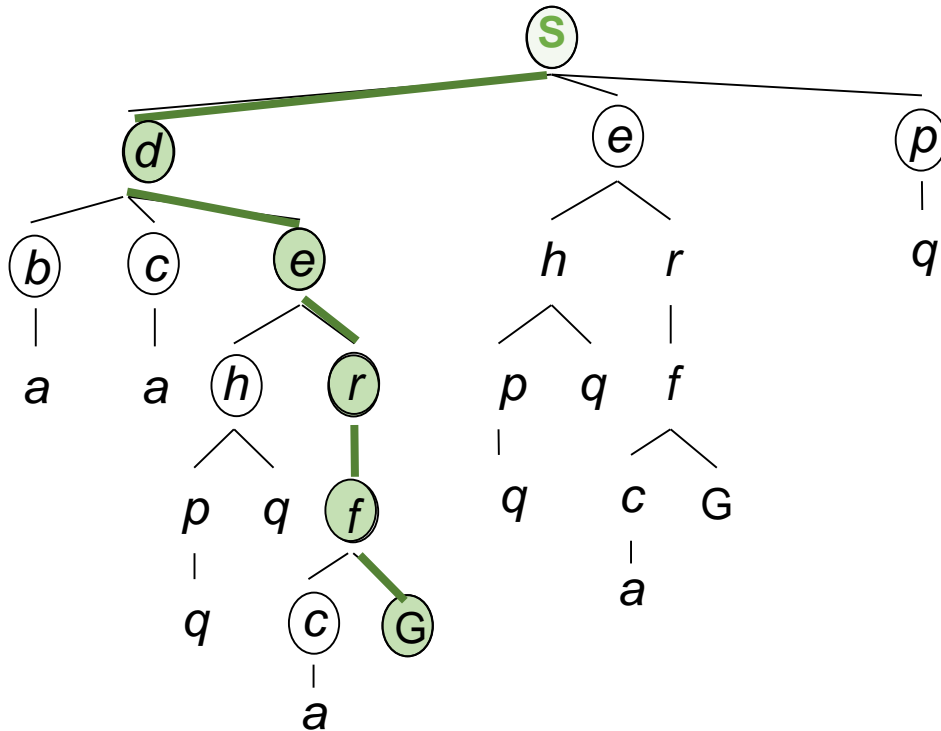
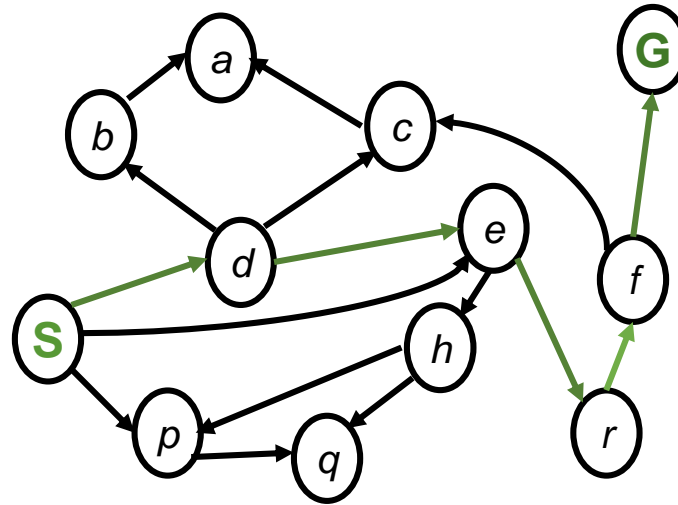
➤ **Cuidado! Se repiten estados visitados!**



Ejemplo: búsqueda en árbol

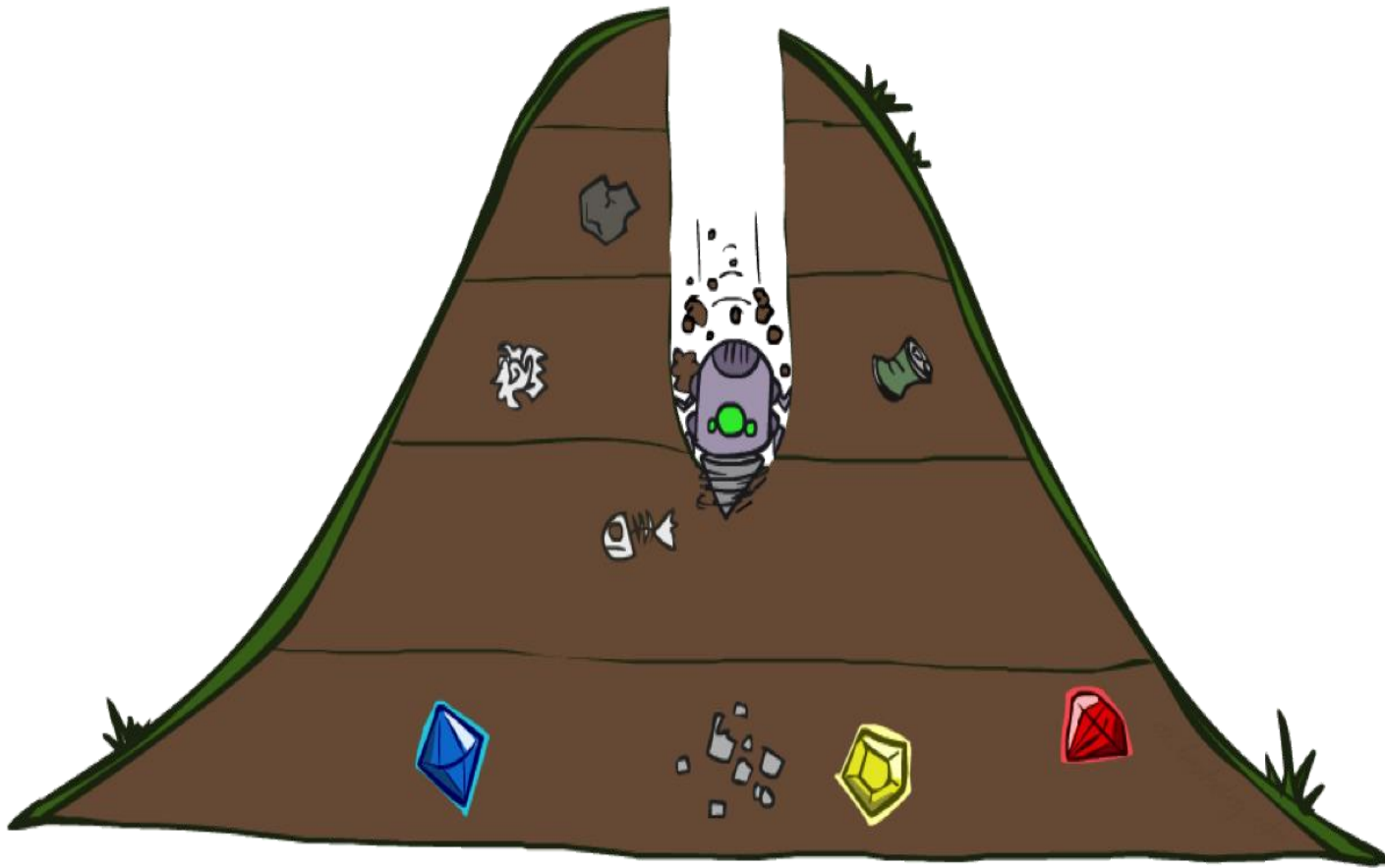


Ejemplo: búsqueda en árbol



~~s~~
~~s → d~~
 s → e
 s → p
 s → d → b
 s → d → c
~~s → d → e~~
 s → d → e → h
~~s → d → e → r~~
~~s → d → e → r → f~~
 s → d → e → r → f → c
~~s → d → e → r → f → G~~

Búsqueda en profundidad (Depth-First Search)



Búsqueda en profundidad

➤ Algoritmo

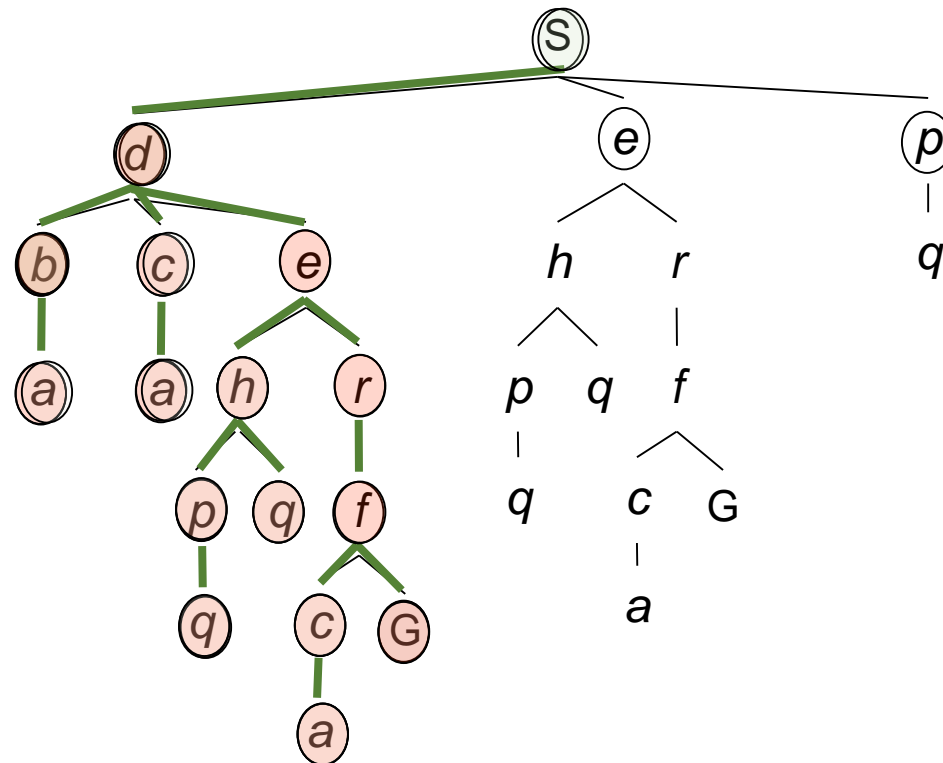
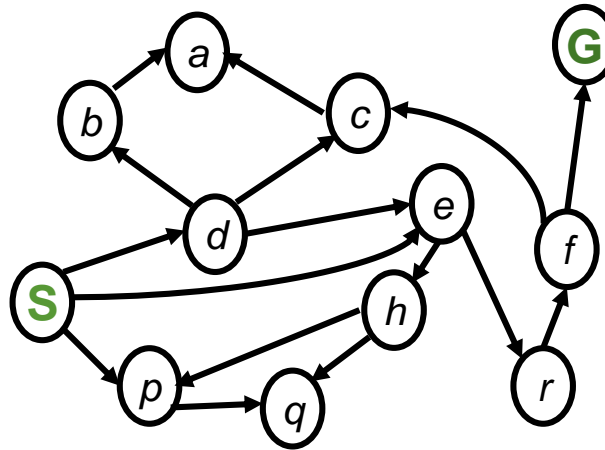
- Construir una lista con el nodo raíz como único elemento.
- Hasta que la lista esté vacía o el primer elemento de la lista sea el elemento objetivo:
 - Eliminar el primer elemento de la lista y añadir los hijos de este elemento (si los hubiera) **al principio de la lista**.
- Si se ha encontrado el nodo objetivo, anunciar éxito, si no, fallo.

➤ Problema

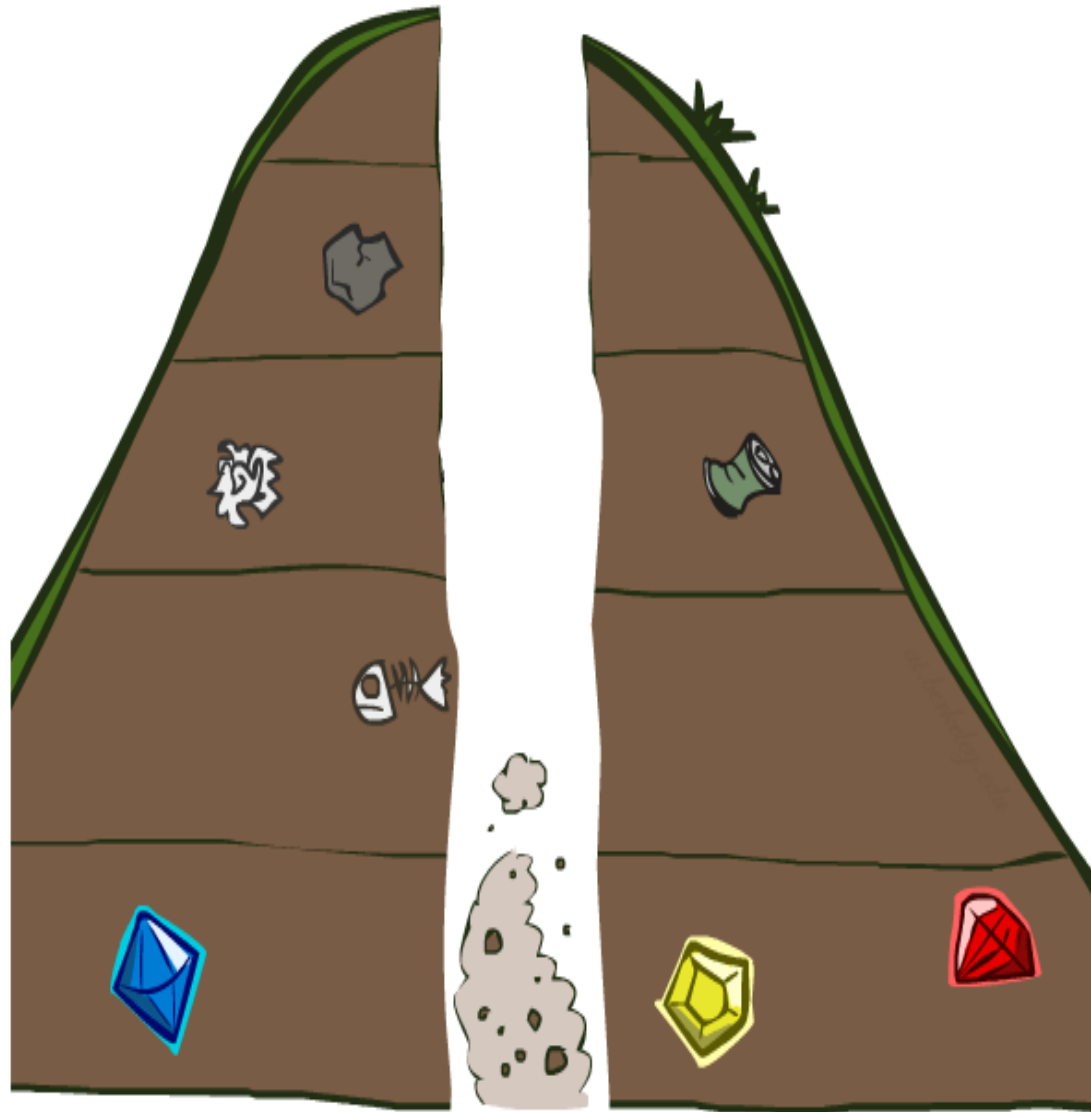
- **Caer en un camino infinitamente largo**

Búsqueda en profundidad

Estrategia: expandir un nodo más profundo primero
Implementación: Fringe es una pila LIFO



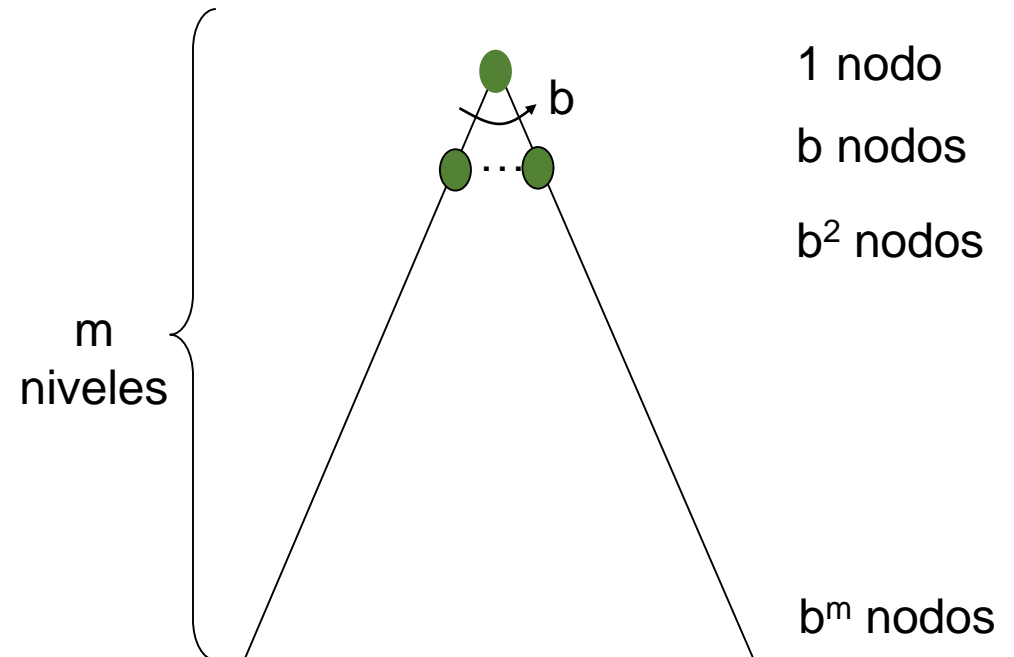
Propiedades de algoritmos de búsqueda



Propiedades de algoritmos de búsqueda

- **Completo:** ¿garantiza encontrar una solución en caso de que haya una?
- **Óptimo:** ¿garantiza encontrar el camino de coste mínimo?
- **¿Complejidad en tiempo?**
- **¿Complejidad en espacio?**

- Árbol de búsqueda:
 - **b** es el factor de ramificación
 - **m** la profundidad máxima

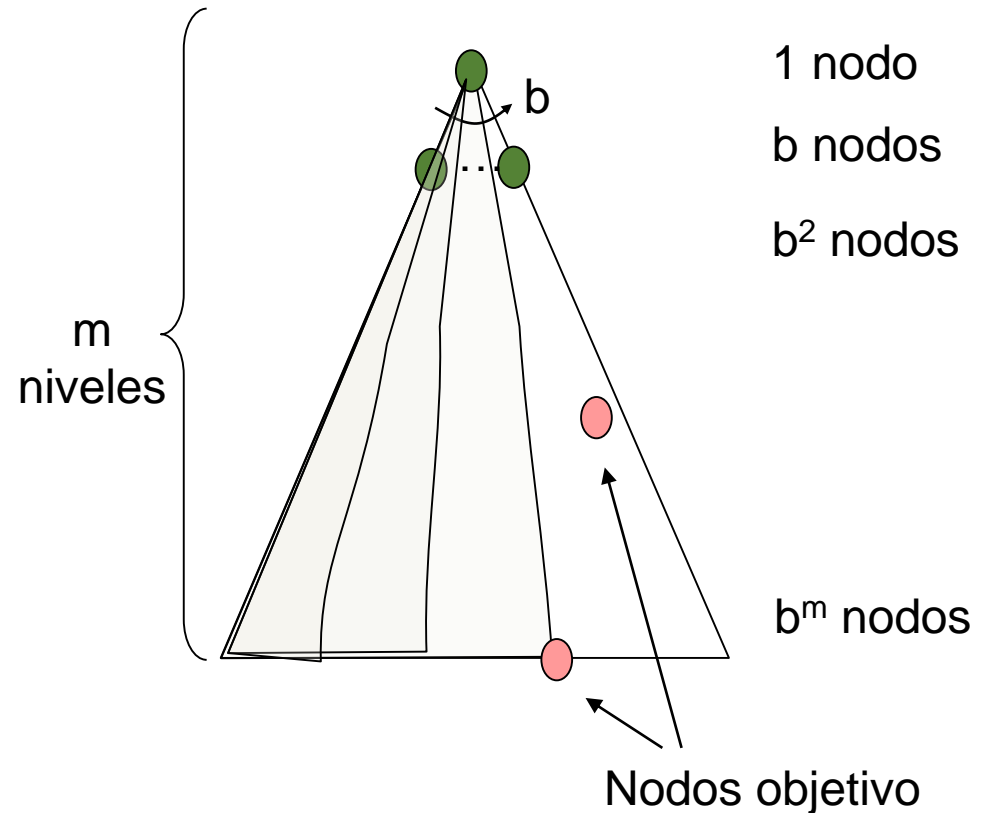


- ¿Número de nodos del árbol?

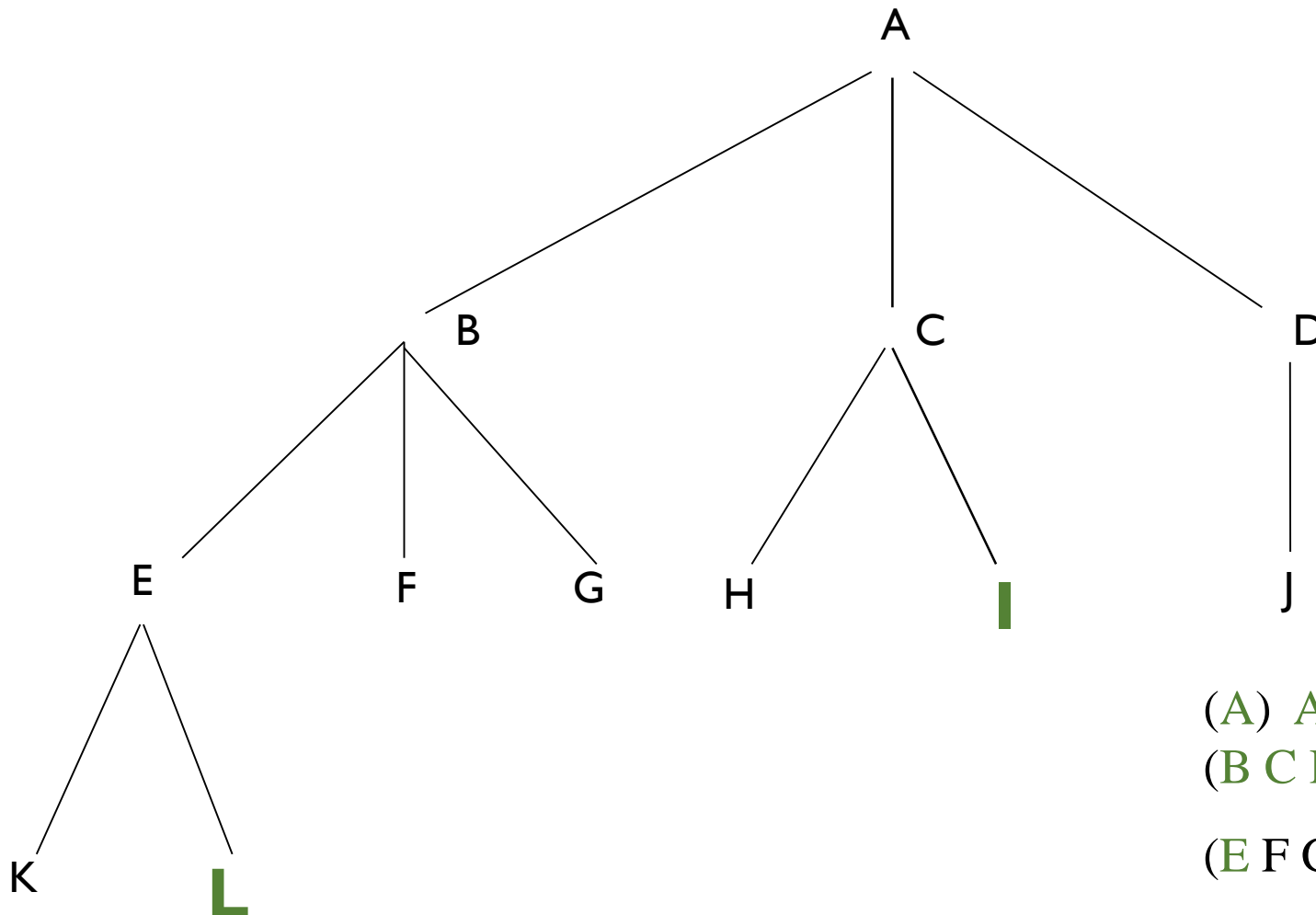
$$1 + b + b^2 + \dots + b^m = O(b^m)$$

Propiedades de Depth-First Search (DFS)

- ¿Qué nodos expande DFS?
 - ¡Podría procesar el árbol entero!
 - Si m es finito, toma tiempo $O(b^m)$
- ¿Cuánto espacio toma el borde (fringe)?
 - Solo contiene los hermanos/as en el camino a la raíz, por lo que $O(b^m)$
- ¿Es completo?
 - m puede ser infinito, por lo que solo si prevenimos ciclos
- ¿Es óptimo?
 - **No**, encuentra la solución “**más a la izquierda**”, sin tener en cuenta la profundidad o coste



Búsqueda en profundidad



(A) A prolongable

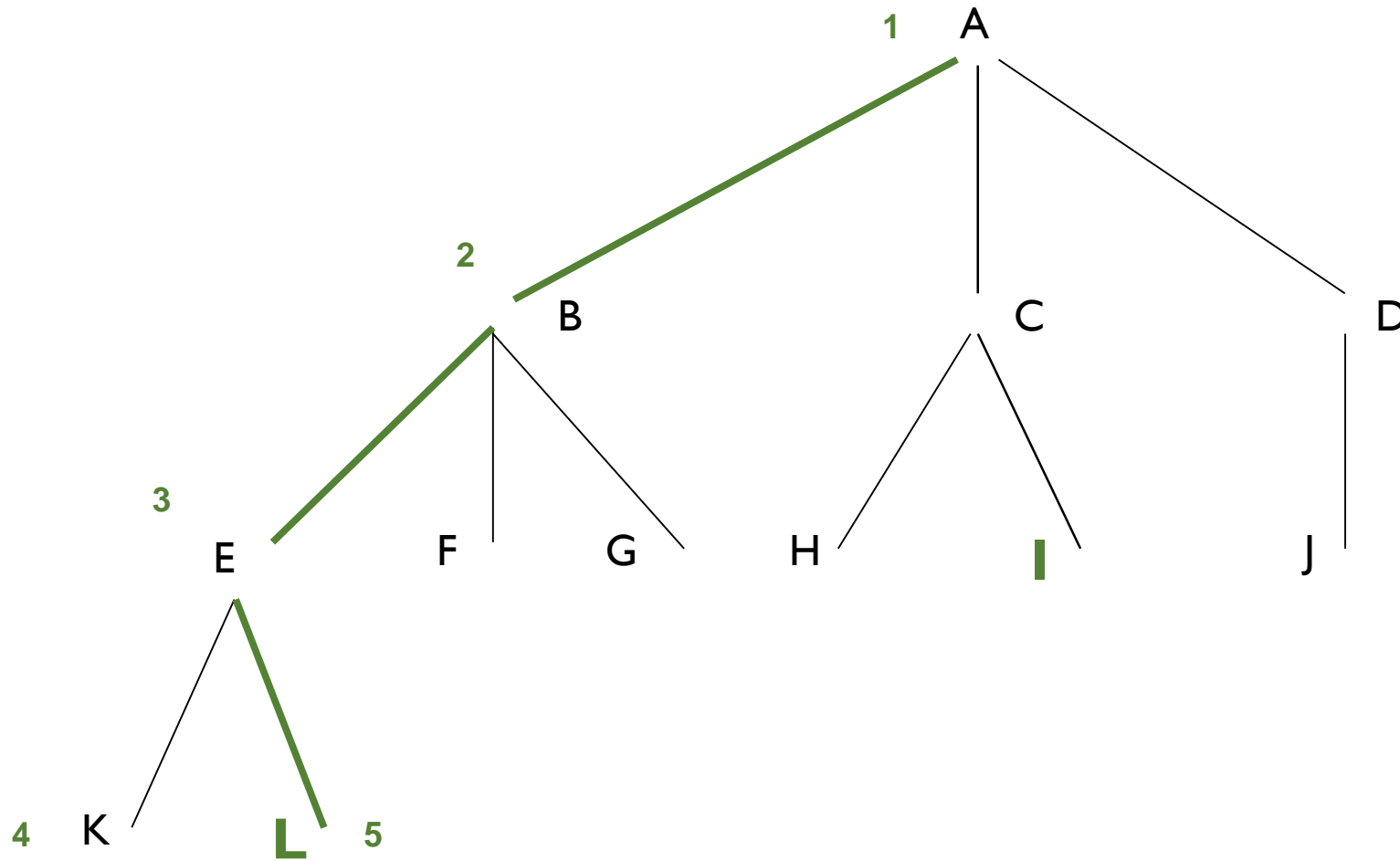
(B C D) B prolongable

(E F G C D) E prolongable

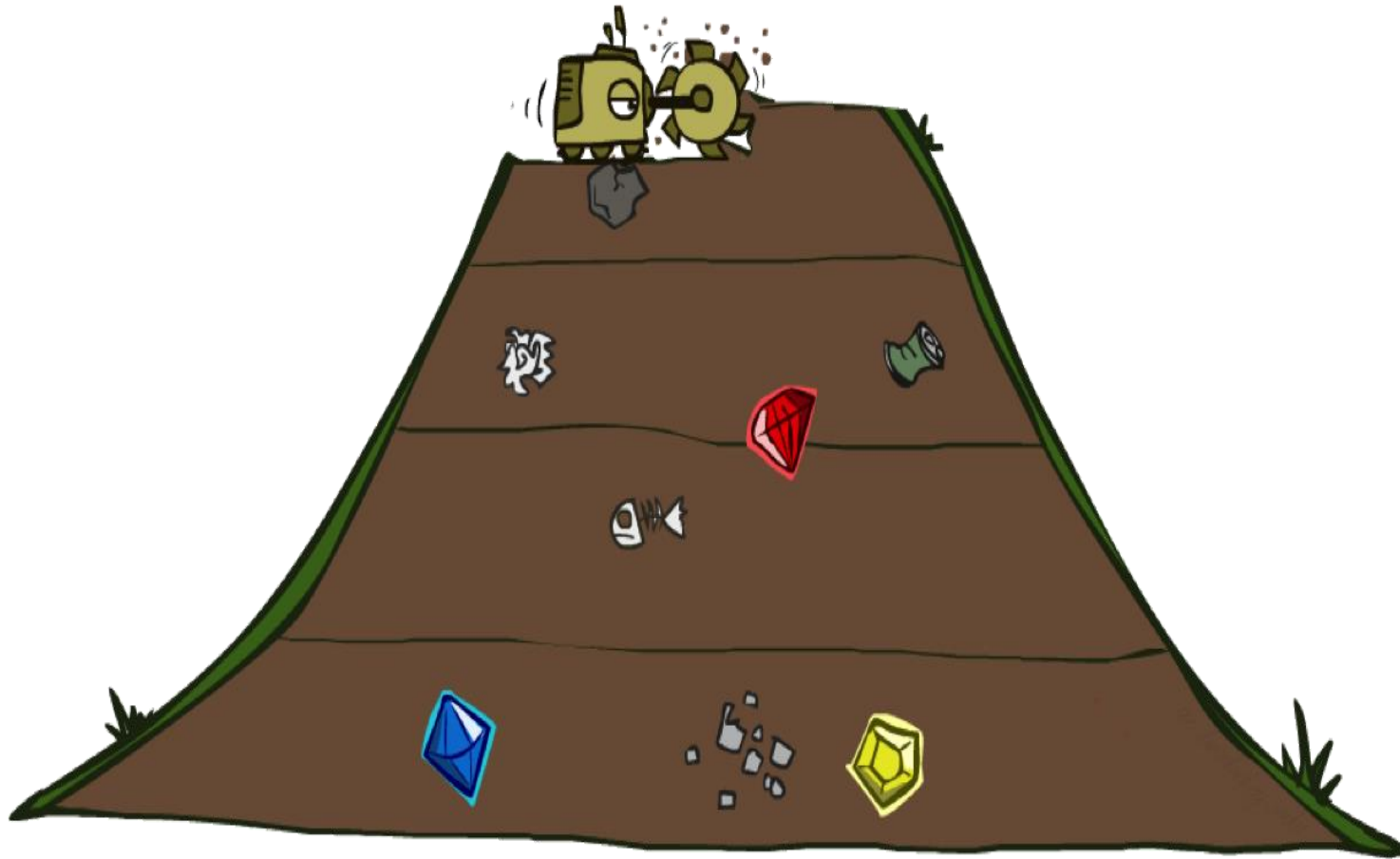
(K L F G C D) K fracaso

(L F G C D) → ÉXITO L

Búsqueda en profundidad



Búsqueda en anchura (Breadth-First Search)



Búsqueda en anchura

➤ Algoritmo

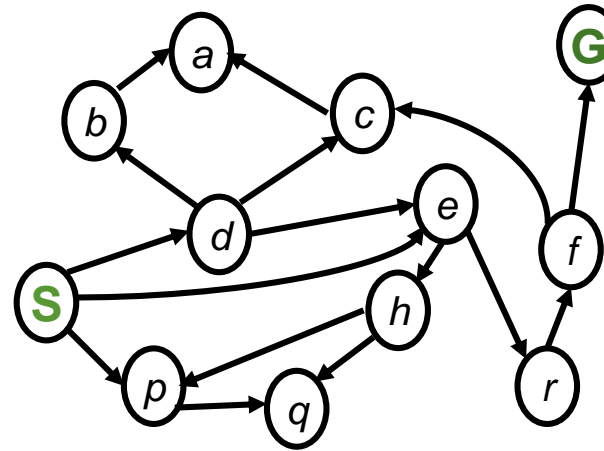
- Construir una lista con el nodo raíz como único elemento.
- Hasta que la lista esté vacía o el primer elemento de la lista sea el elemento objetivo:
 - Eliminar el primer elemento de la lista y añadir los hijos de este elemento (si los hubiera) **al final de la lista**.
- Si se ha encontrado el nodo objetivo, anunciar éxito, si no, fallo.

➤ Problema

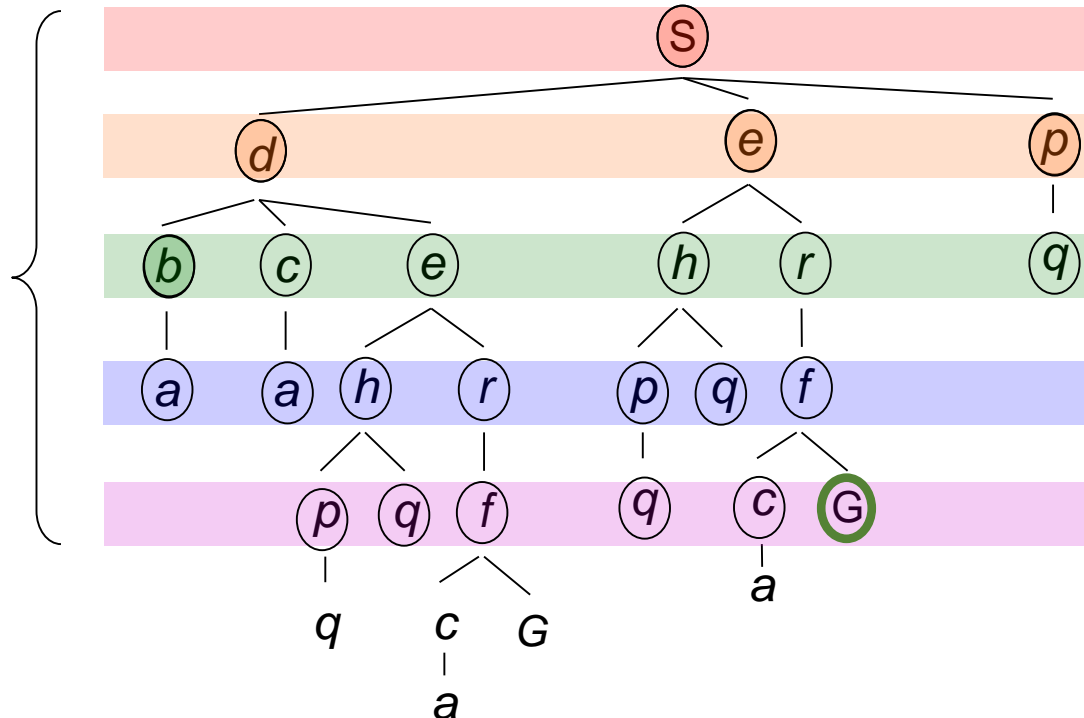
- **Caer en un camino infinitamente largo**

Breadth-First Search

*Estrategia: expandir primero
un nodo del nivel más bajo
Implementación: el borde (fringe)
es una cola FIFO*

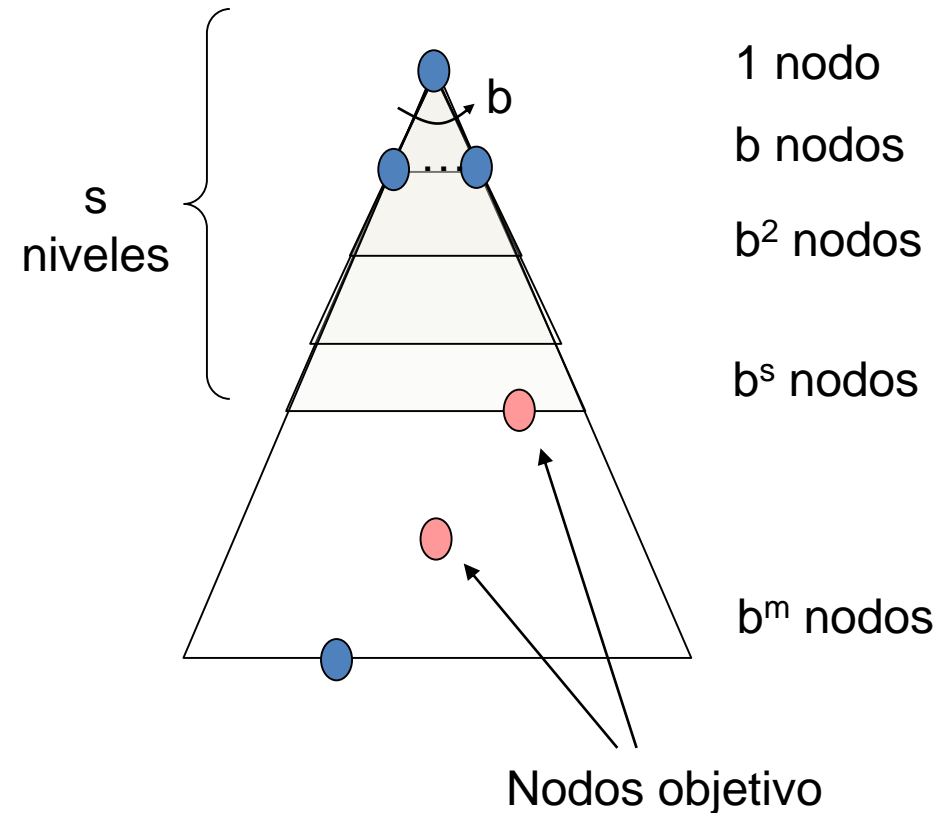


Niveles de
búsqueda

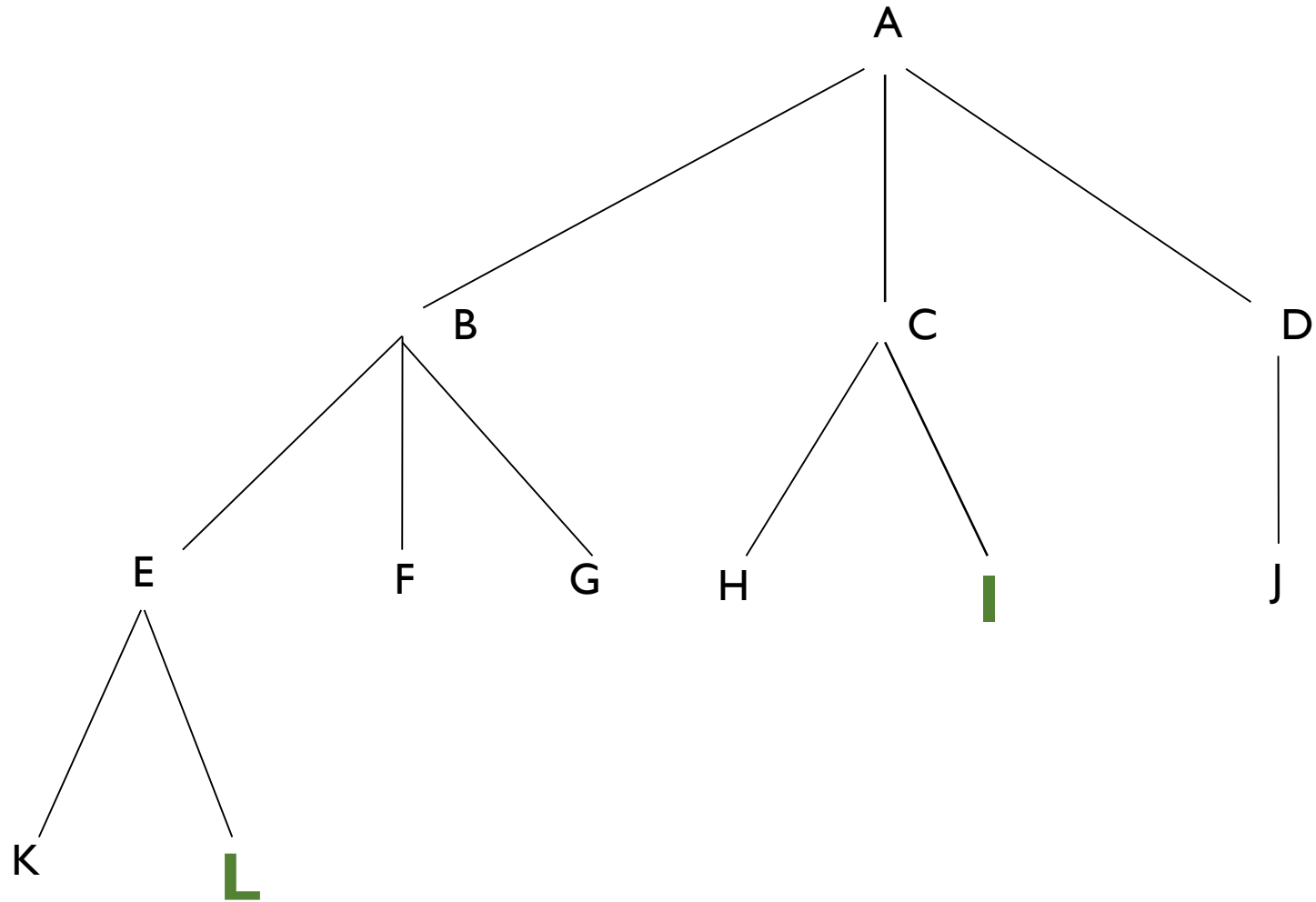


Propiedades de Breadth-First Search (BFS)

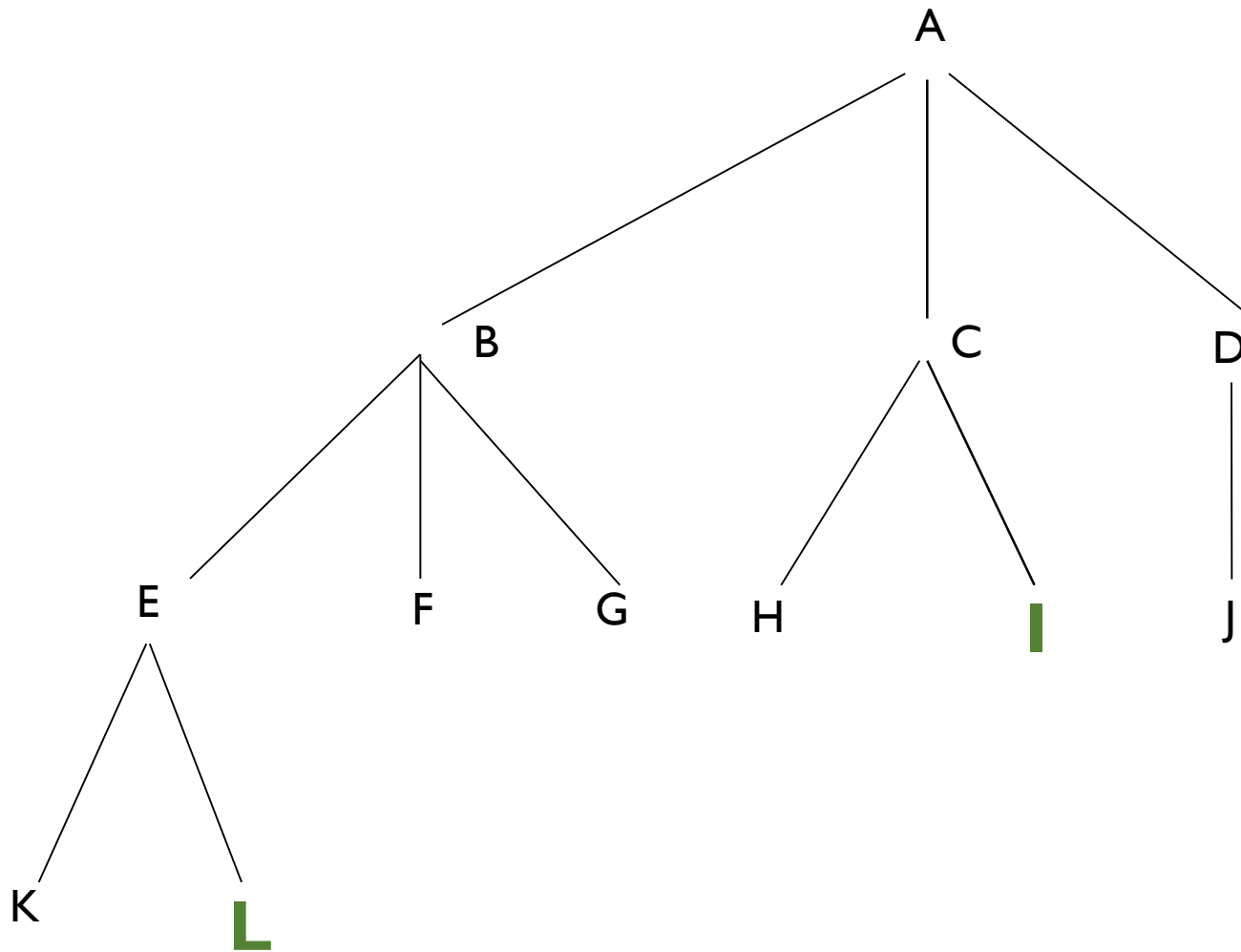
- ¿Qué nodos expande BFS?
 - Procesa todos los nodos por encima de la solución de menor nivel
 - Sea s la profundidad de la solución de menor nivel
 - La búsqueda toma un tiempo $O(b^s)$
- ¿Cuánto espacio toma el borde (fringe)?
 - Aproximadamente el último nivel, $O(b^s)$
- ¿Es completo?
 - s debe ser finito si existe una solución, por lo que sí
- ¿Es óptimo?
 - **Solo si todos los costes son 1** (más sobre esto después)



Búsqueda en anchura

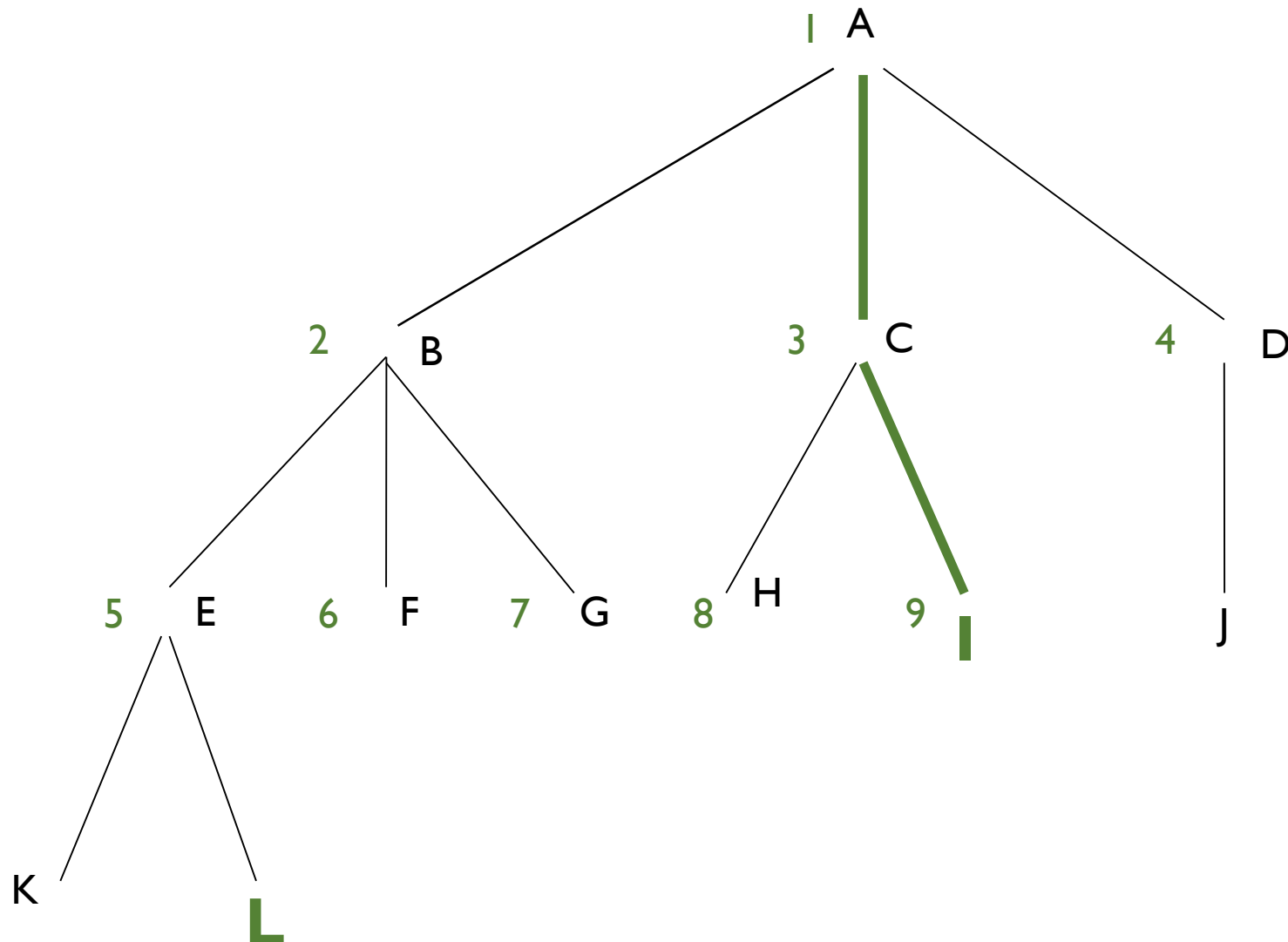


Búsqueda en anchura



(A) A prolongable
(B C D) B prolongable
(C D E F G) C prolongable
(D E F G H I) D prolongable
(E F G H I J) E prolongable
(F G H I J K L) F fracaso
(G H I J K L) G fracaso
(H I J K L) H fracaso
(I J K L) → ÉXITO I

Búsqueda en anchura



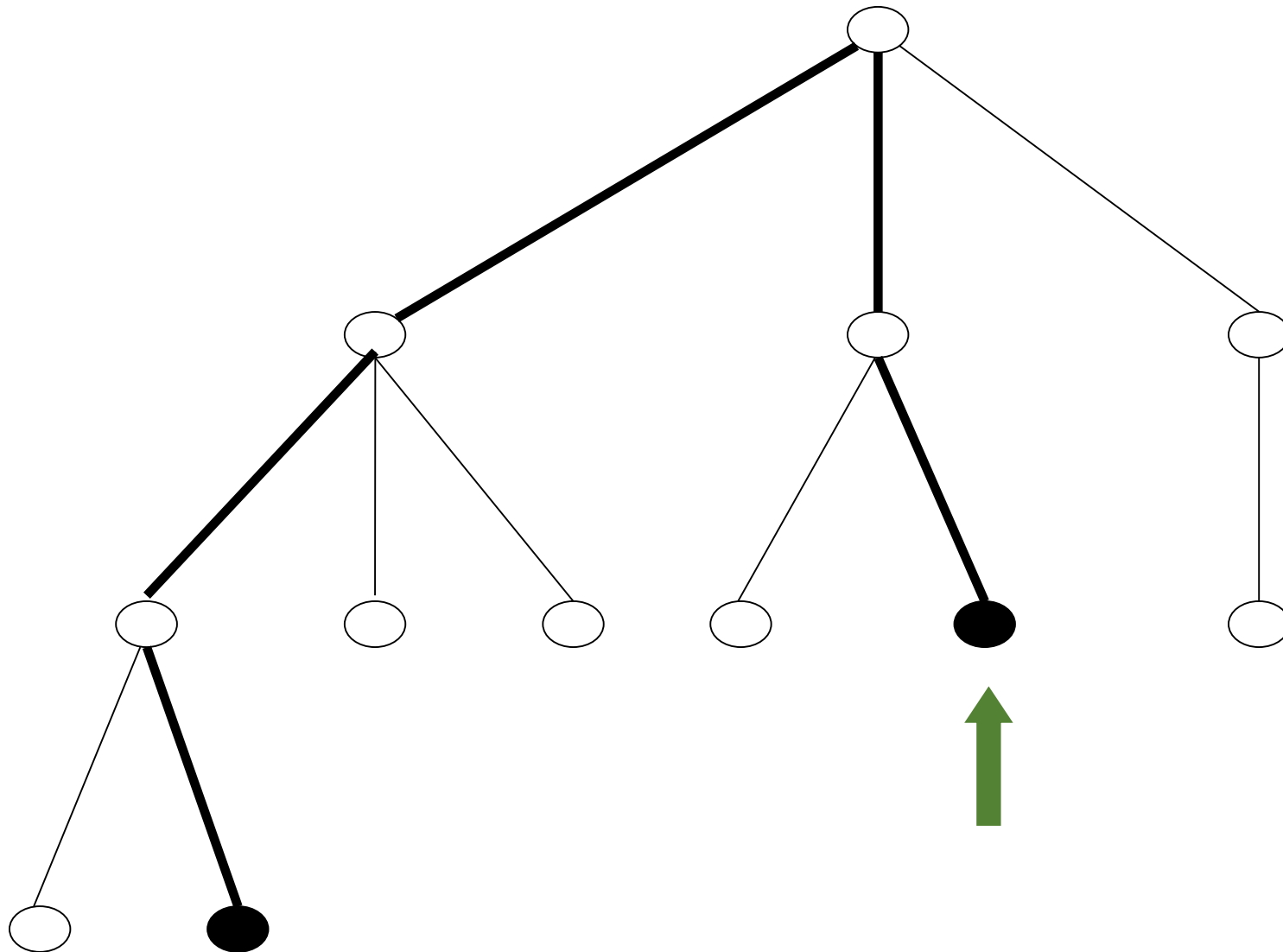
➤ Algoritmo

- Busca todos los posibles caminos y selecciona el mejor entre ellos
 - Puede implementarse modificando cualquiera de los dos anteriores

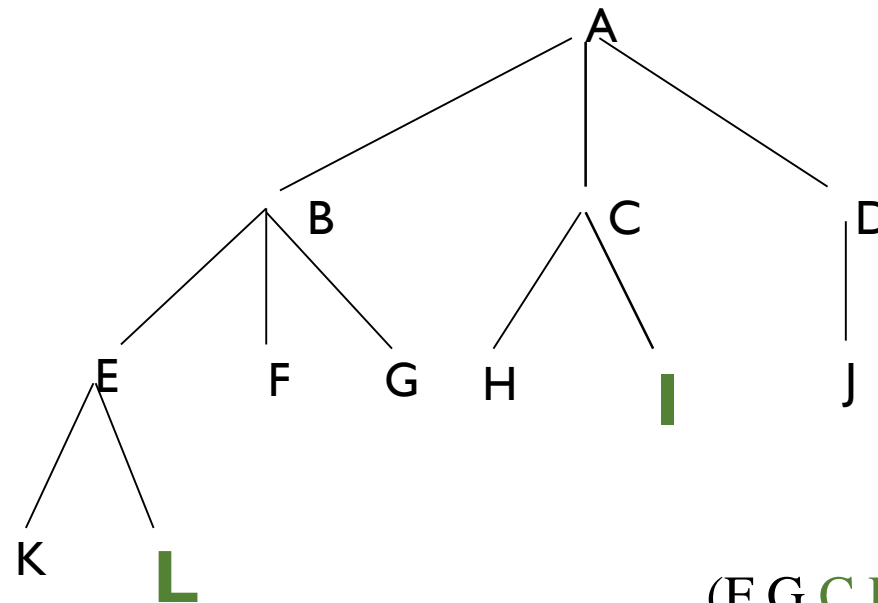
➤ Problema

- Ineficiente en espacios de estado grandes
- Inaplicable en problemas con explosión combinatoria

British museum



Como variante de la búsqueda en profundidad



(A) A prolongable

(B C D) B prolongable

(E F G C D) E prolongable

(K L F G C D) K fracaso

(L F G C D) → **ÉXITO L**

Se podría seguir a partir de aquí si se quieren todas las soluciones, tratando el nodo éxito como otro nodo cualquiera

(F G C D) F fracaso

(G C D) G fracaso

(C D) C prolongable

(H I D) H prolongable

(I D) → **ÉXITO I**

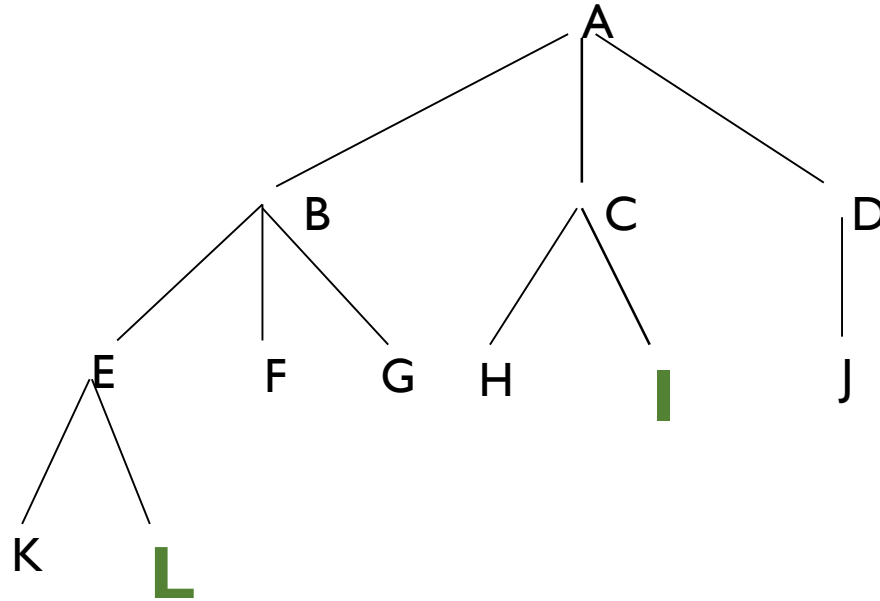
(D) D prolongable

(J) J fracaso

() FALLO

NO HAY MÁS SOLUCIONES

Como variante de la búsqueda en anchura



(A) A prolongable

(B C D) B prolongable

(C D E F G) C prolongable

(D E F G H I) D prolongable

(E F G H I J) E prolongable

(F G H I J K L) F fracaso

(G H I J K L) G fracaso

(H I J K L) H fracaso

(I J K L) → ÉXITO I

Se pueden encontrar todas las soluciones tratando el nodo éxito como otro nodo cualquiera.

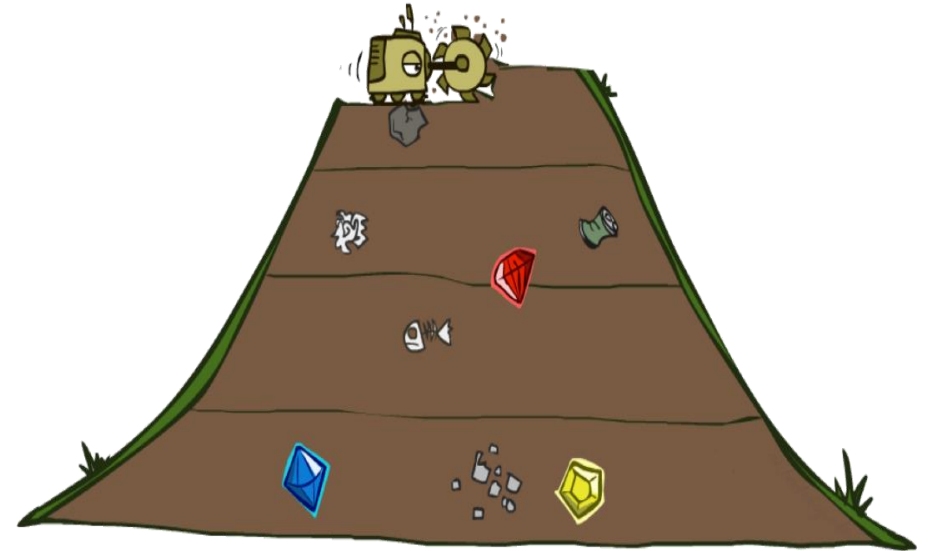
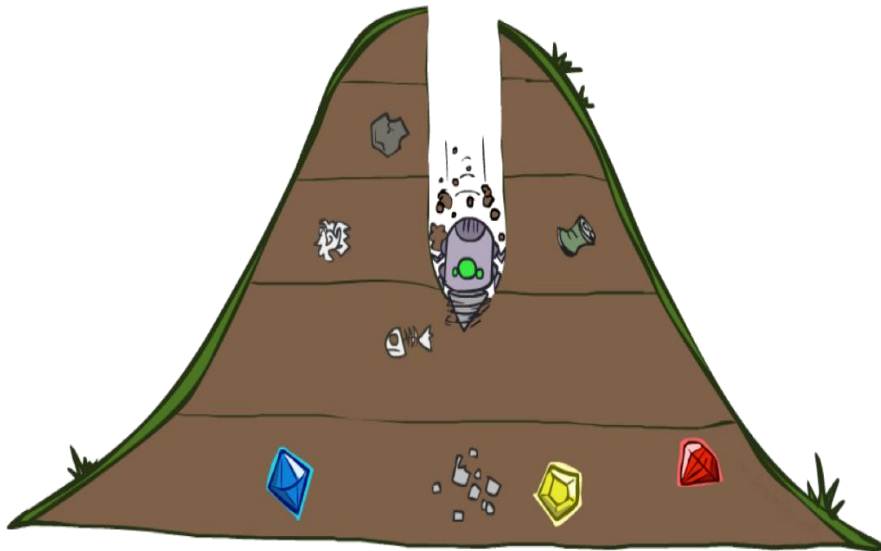
(J K L) J fracaso

(K L) K fracaso

(L) → ÉXITO L

() FALLO

Quiz: DFS vs BFS

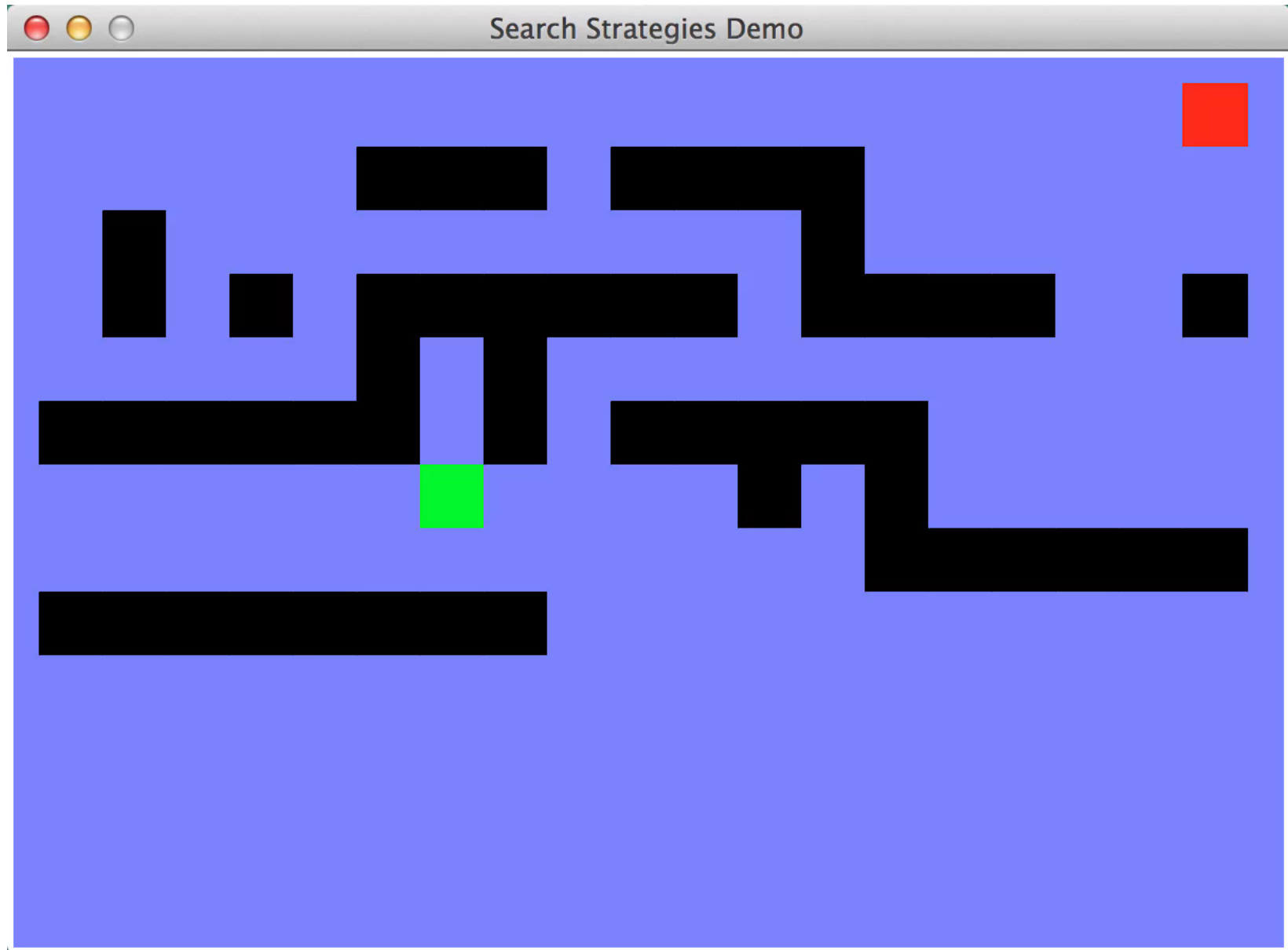


Quiz: DFS vs BFS

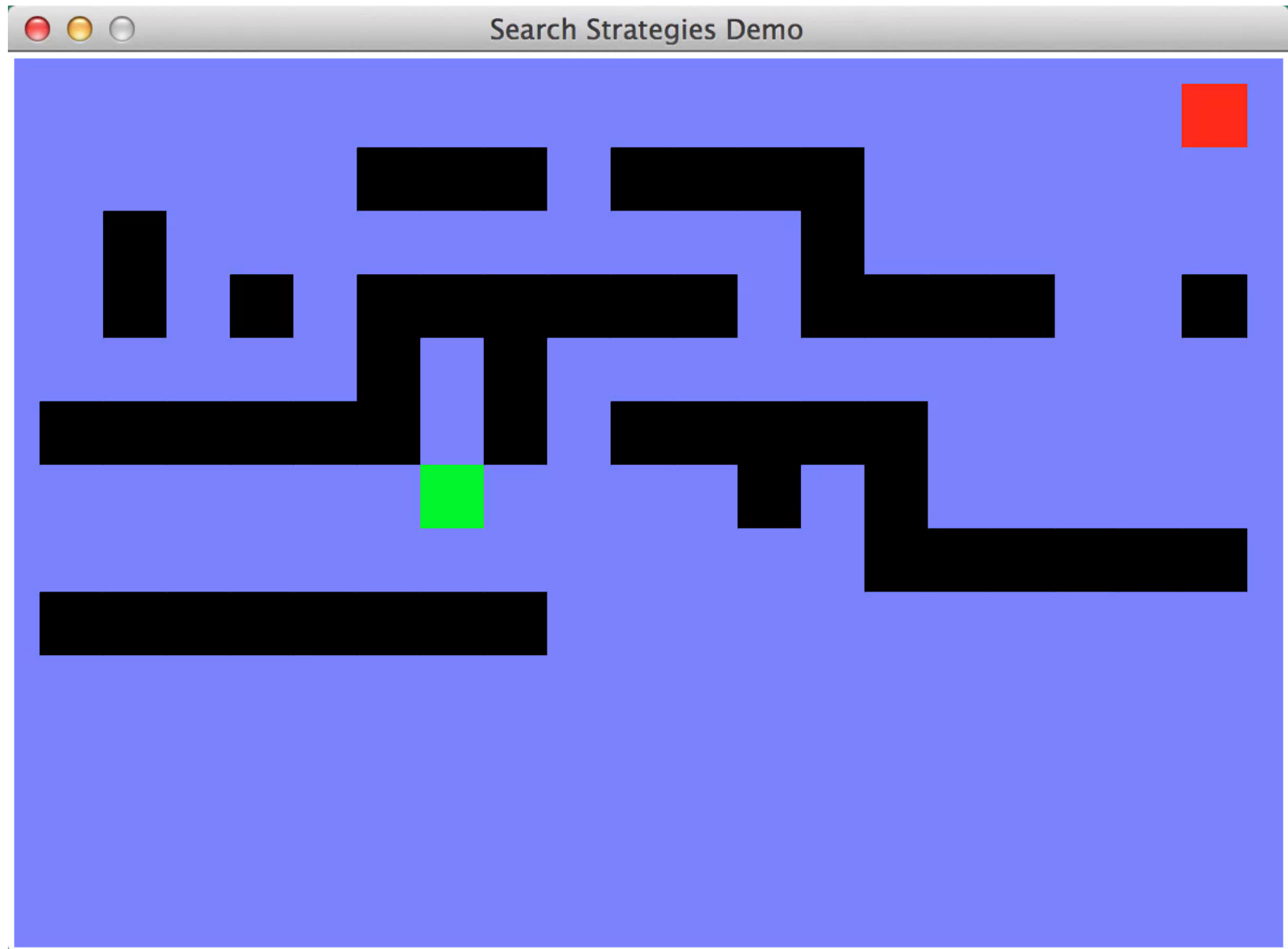
➤ ¿Cuándo supera BFS a DFS?

➤ ¿Cuándo supera DFS a BFS?

Video of Demo Maze Water DFS/BFS (part 1)

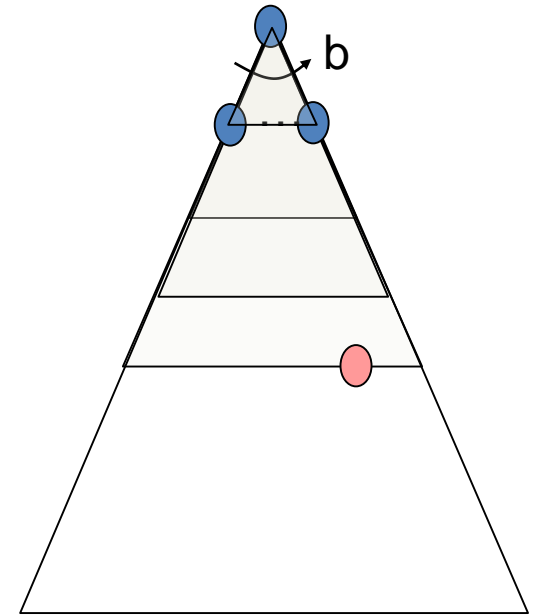


Video of Demo Maze Water DFS/BFS (part 2)

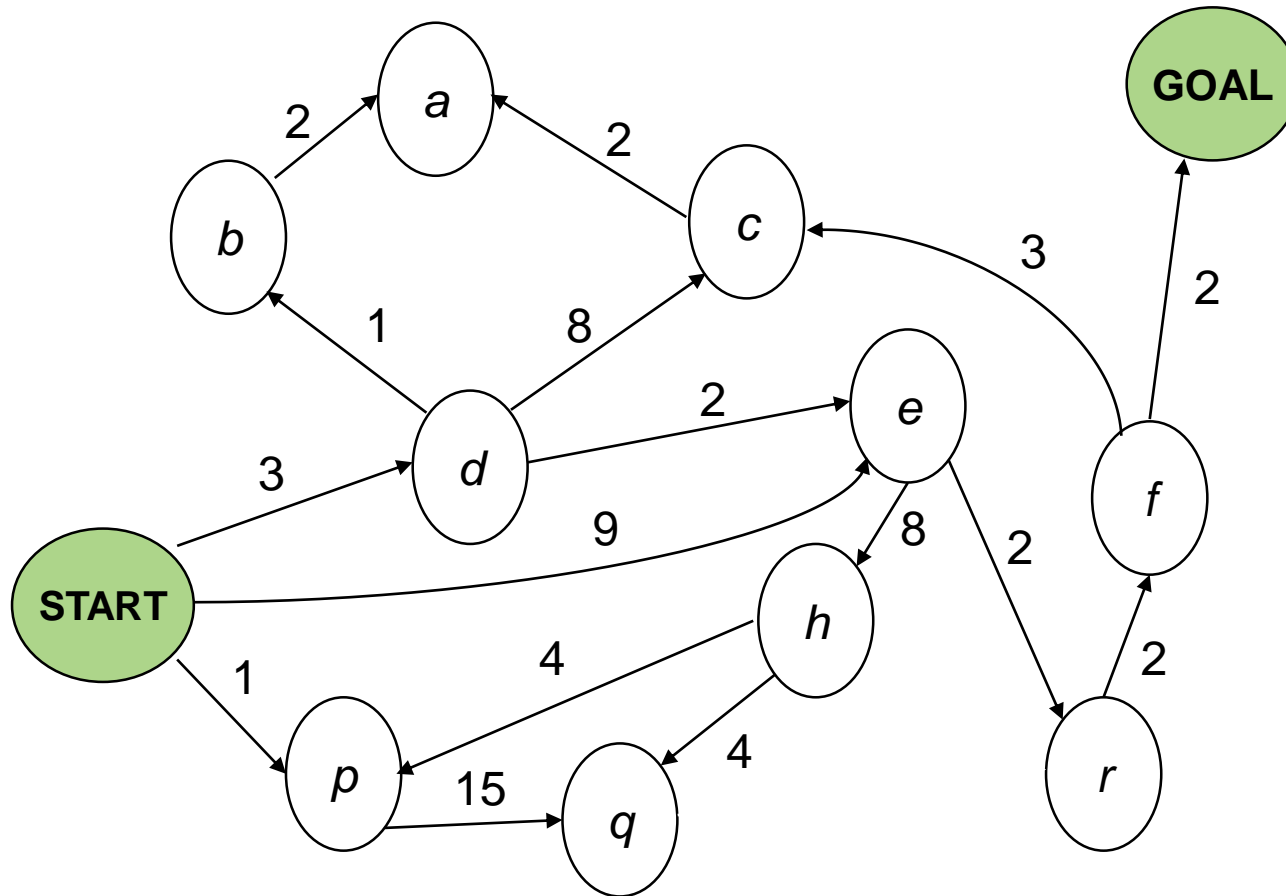


Profundización Iterativa (Iterative Deepening)

- Idea: aprovechar la ventaja en espacio de DFS con el tiempo de BFS / (ventajas de la solución de menor nivel)
 - Ejecutar DFS con límite de profundidad 1. Si no hay solución ...
 - Ejecutar DFS con límite de profundidad 2. Si no hay solución ...
 - Ejecutar DFS con límite de profundidad 3.
- ¿No es totalmente redundante e ineficiente?
 - En general, la mayoría del tiempo se gasta en el último nivel, por lo que no está mal!

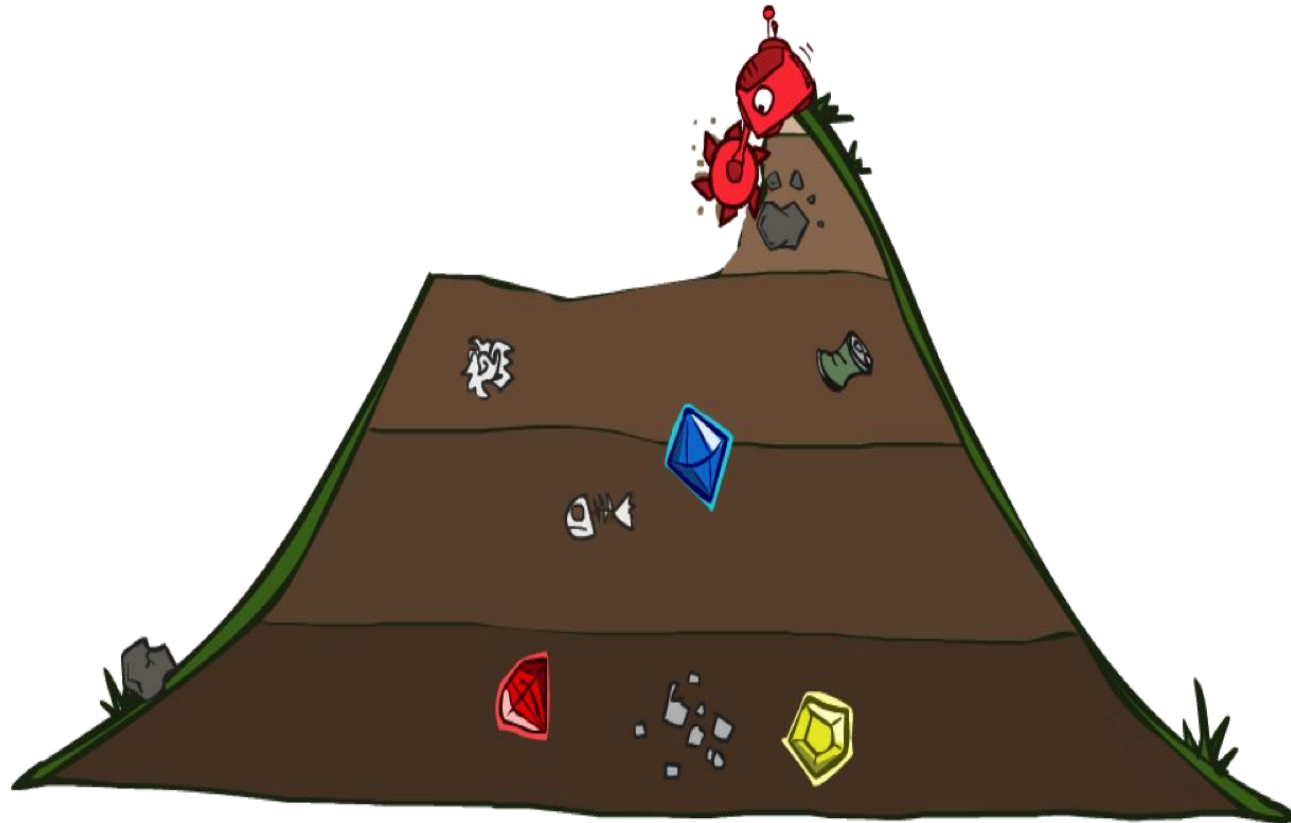


Búsqueda sensitiva al coste (Cost-Sensitive Search)



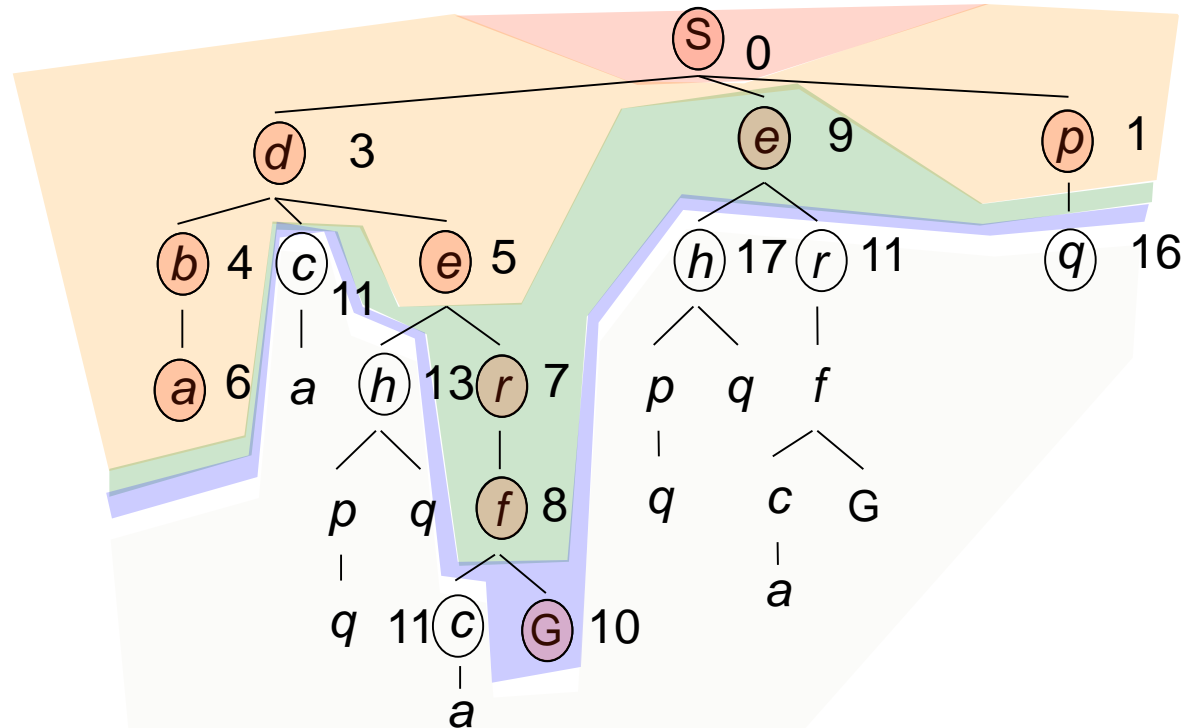
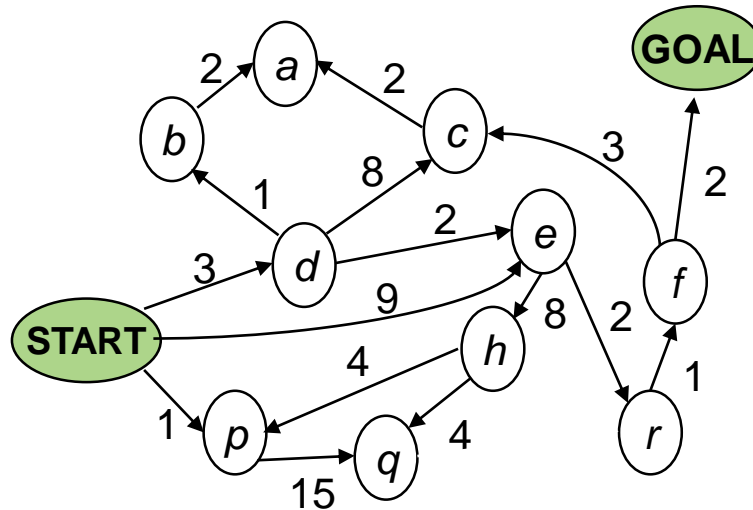
- **BFS encuentra el camino más corto en función del número de acciones** (pasos), pero no encuentra el camino de coste mínimo. Examinaremos un algoritmo similar que encuentra el camino de **coste mínimo**.

Búsqueda de coste uniforme (Uniform Cost Search, UCS)



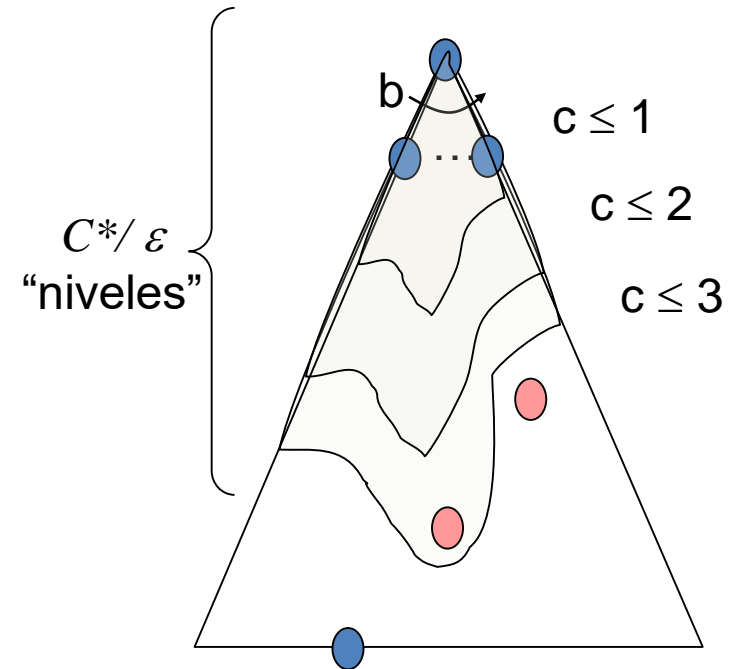
Uniform Cost Search

*Estrategia: expandir el
nodo más barato primero:
El borde (Fringe) es una
cola de prioridad
(prioridad: coste
acumulativo)*



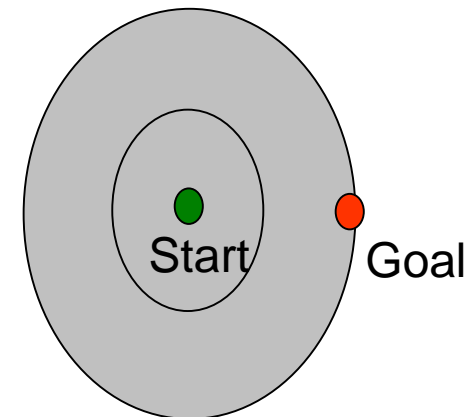
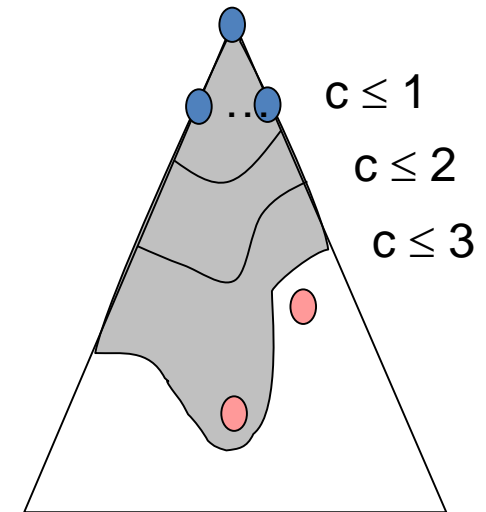
Propiedades de Uniform Cost Search (UCS)

- ¿Qué nodos expande UCS?
 - ¡Procesa todos los nodos con coste menor al de la solución de menor coste!
 - Si esa solución cuesta C^* y los arcos cuestan al menos ϵ , entonces la “profundidad efectiva” es aproximadamente C^*/ϵ
 - Toma tiempo $O(b^{C^*/\epsilon})$ (exponencial respecto a la profundidad efectiva)
- ¿Cuánto espacio toma el borde (fringe)?
 - Contiene aproximadamente el último nivel, por ello $O(b^{C^*/\epsilon})$
- ¿Es completo?
 - Asumiendo que la mejor solución tiene un coste finito y que el coste mínimo de un arco es positivo, **sí**
- ¿Es óptimo?
 - **¡Sí!** (la prueba más tarde, con el algoritmo A^*)

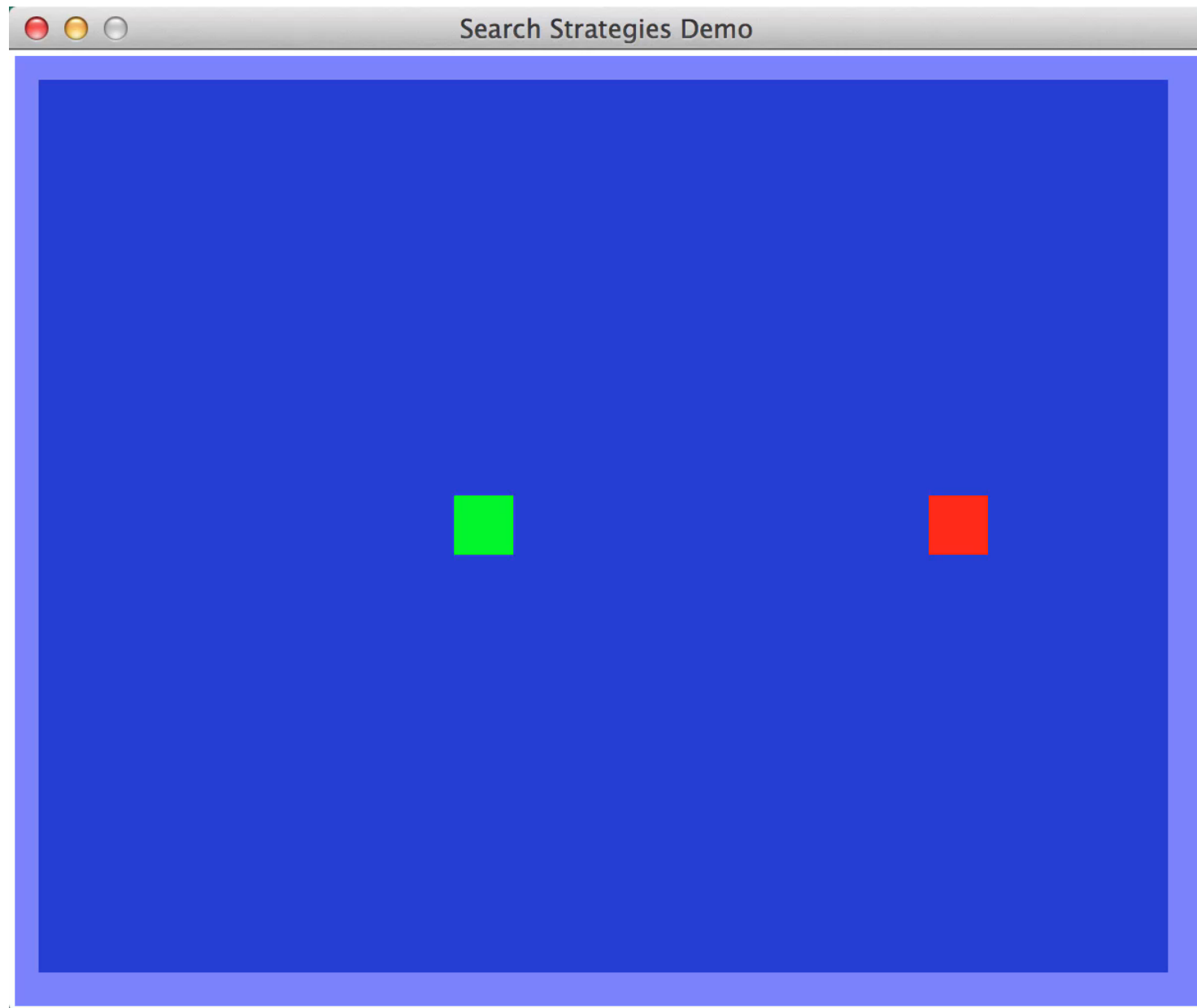


Aspectos de Uniform Cost

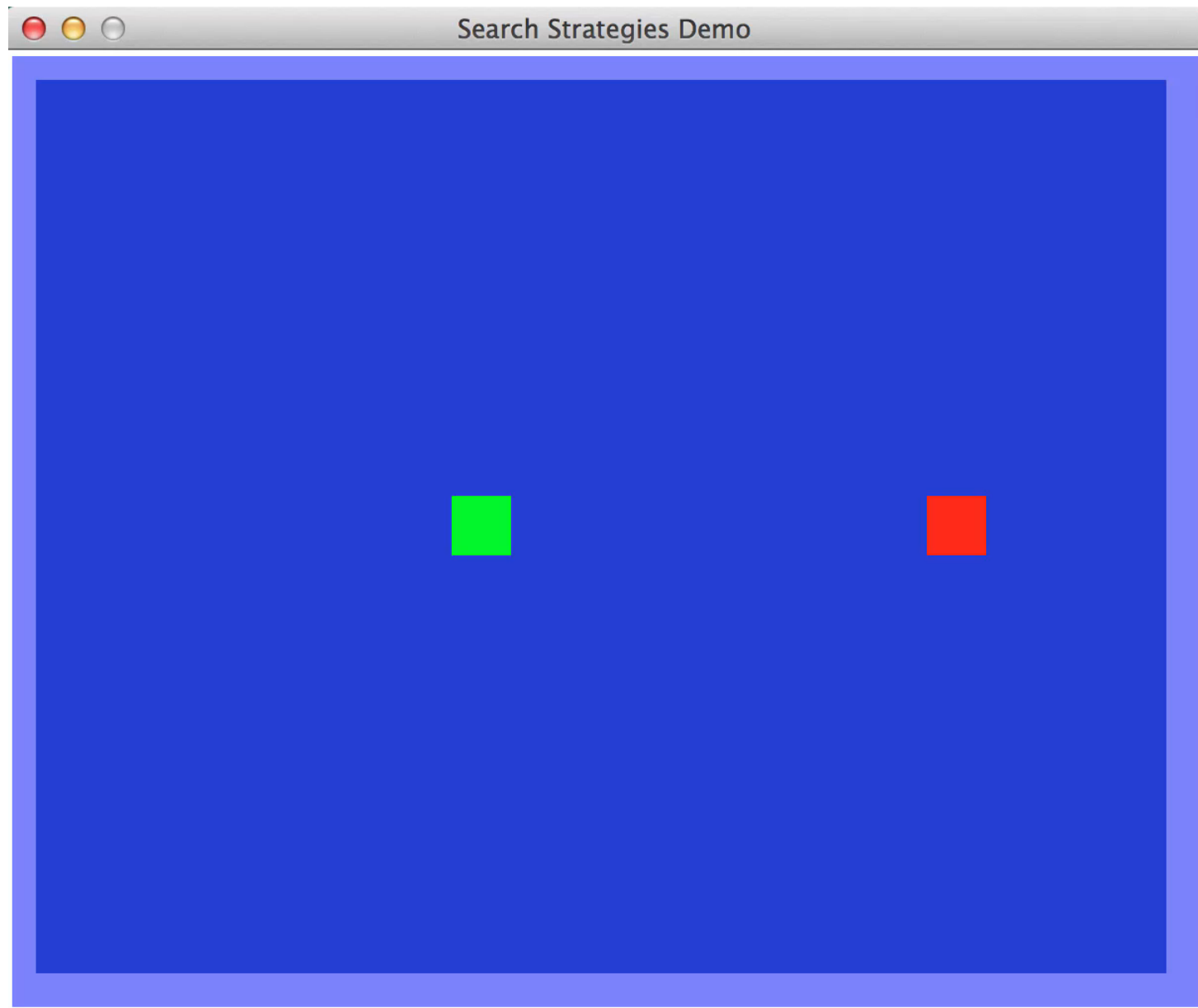
- A recordar: UCS explora contornos de coste de manera incremental
- La parte buena: UCS es completo y óptimo
- Lo malo:
 - Explora opciones en cualquier “dirección”
 - No hay información acerca de la posición del objetivo
- Se arreglará con la búsqueda heurística



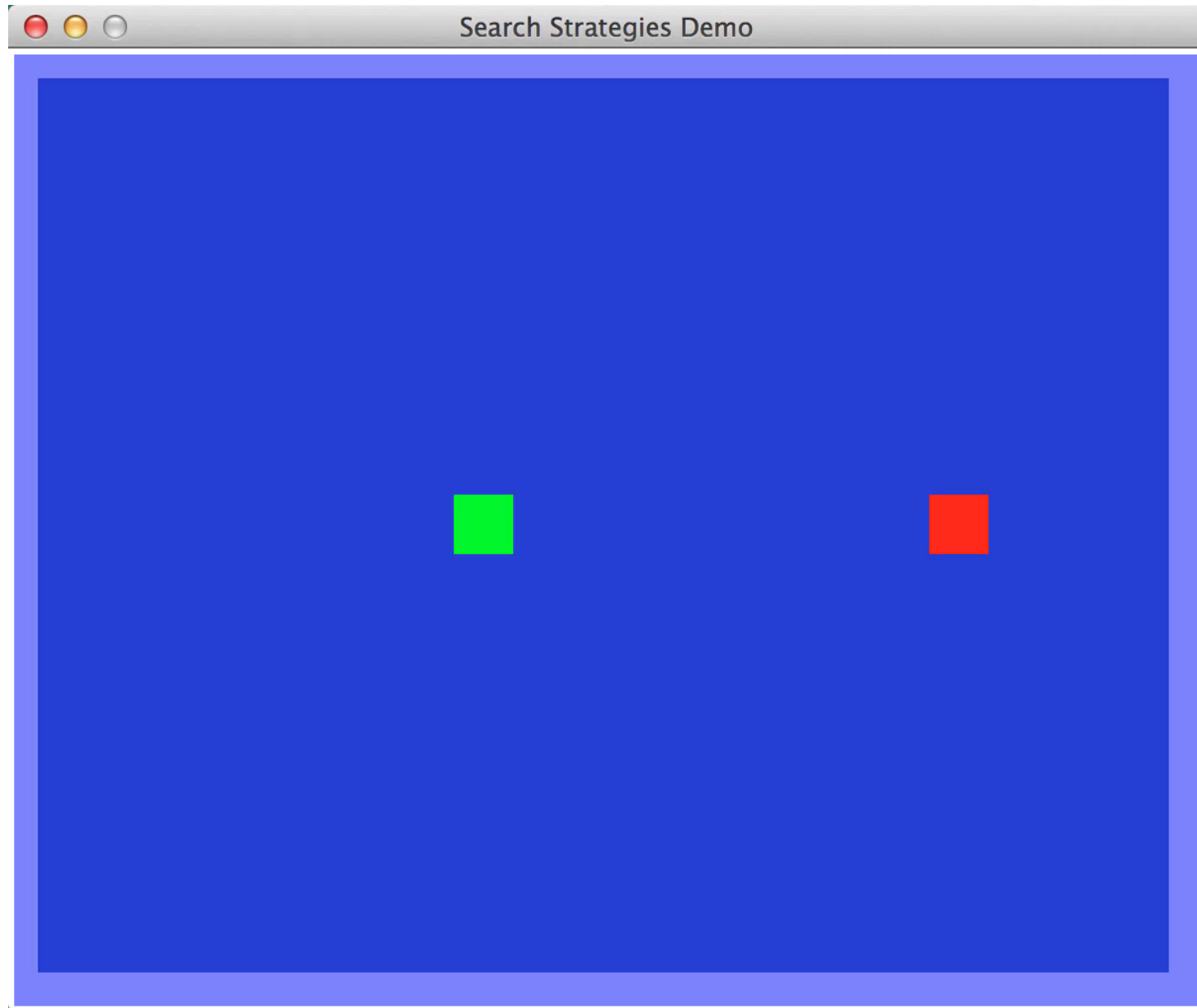
Otro vídeo de DFS



Otro vídeo de BFS

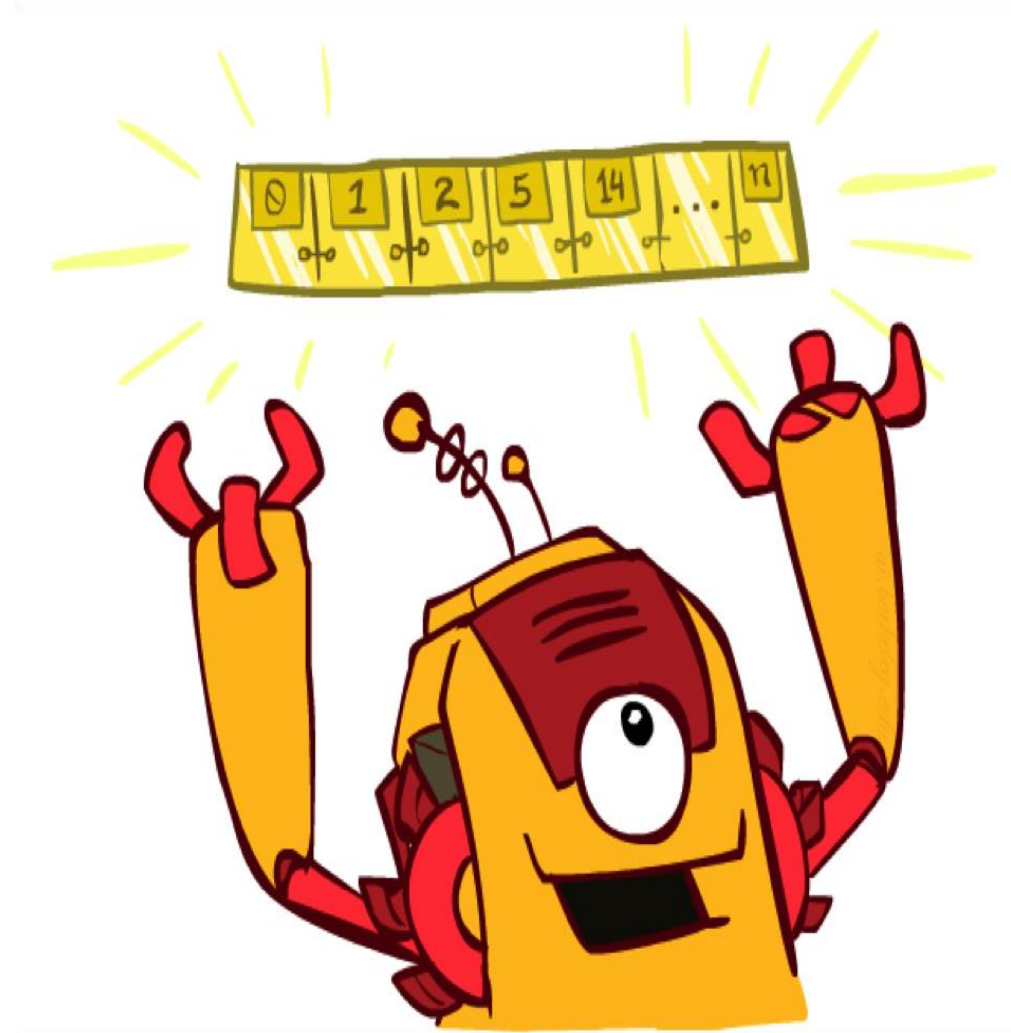


Vídeo de UCS



Una sola cola

- Los 3 algoritmos (DFS, BFS, UCS) son el mismo **excepto por la estrategia de tratamiento del borde**
 - Conceptualmente, todos los bordes son colas de prioridad (es decir, colecciones de nodos con prioridades asignadas)
 - Se puede codificar una sola implementación que toma como parámetro un objeto de tipo cola



Búsqueda Exhaustiva: Conclusión

➤ Conclusión:

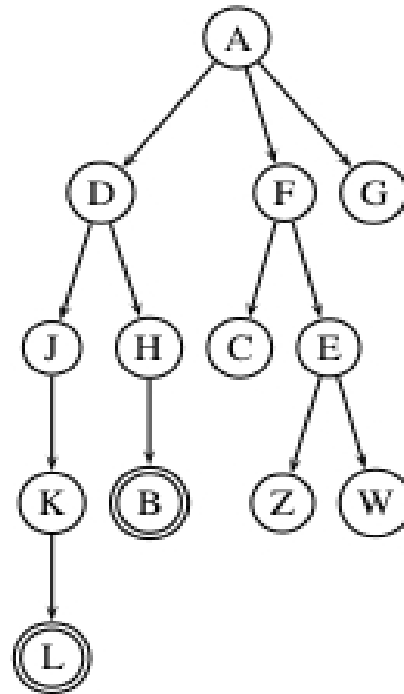
- **Pueden** encontrar soluciones en problemas generando sistemáticamente nuevos estados y comparándolos con el objetivo.
- Son increíblemente ineficientes **en la mayoría de los casos.**

➤ ¿Mejoras? **Búsqueda informada**

- Utilizar el conocimiento específico del problema para intentar encontrar soluciones de una forma más eficiente.

Ejercicio búsqueda exhaustiva

- Dado el árbol de la figura 2 donde B y L son los 2 únicos nodos meta y A es el nodo inicial.



- Indica en qué orden se visitarían los nodos, distinguiendo nodos generados de nodos expandidos, para los siguientes algoritmos:
 1. Anchura (amplitud)
 2. Profundidad

Ejercicio búsqueda exhaustiva

1. Búsqueda en Amplitud:

abierto	Explorar
A	A
D,F,G	D
F,G,J,H	F
G,J,H,C,E	G
J,H,C,E	J
H,C,E,K	H
C,E,K,B	
E,K,B	E
K,B,Z,W	K
B,Z,W,L	

2. Búsqueda en Profundidad:

abierto	Explorar
A	A
D,F,G	D
J,H,F,G	J
K,H,F,G	K
L,H,F,G	