Inteligencia Artificial

Búsqueda no informada

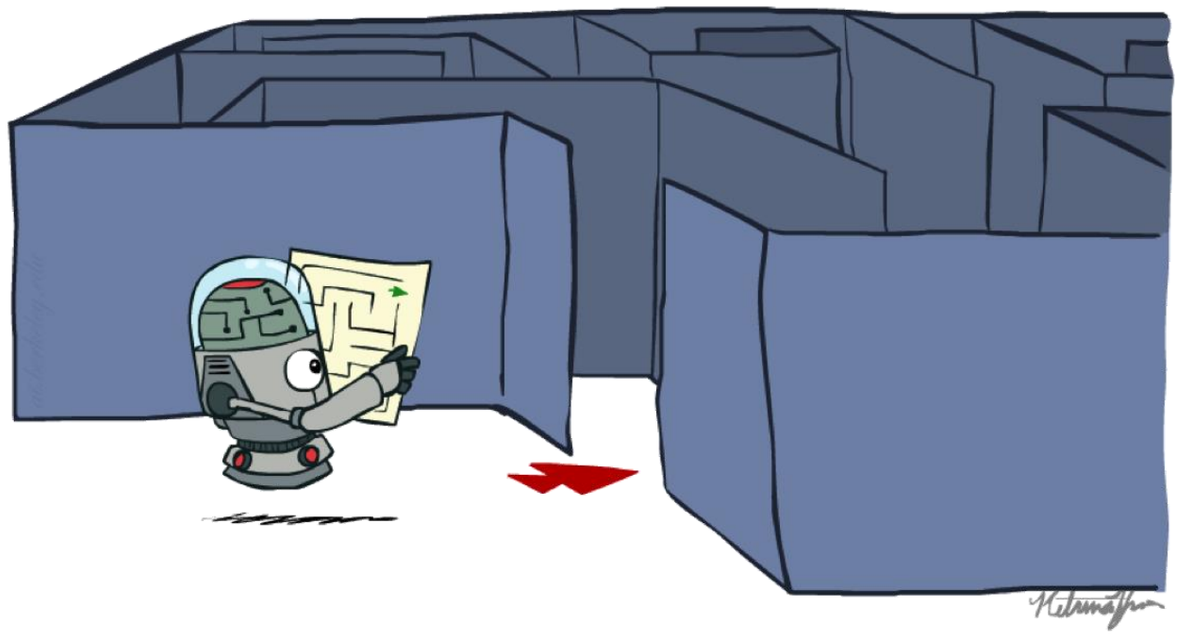
[Transparencias adaptadas de Dan Klein and Pieter Abbeel: CS188 Intro to AI, UC Berkeley (ai.berkeley.edu)]

Universidad Rey Juan Carlos

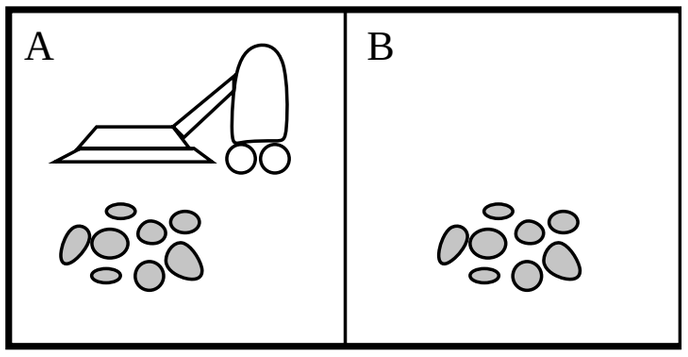
Contenido

- Agentes que planifican con antelación
- Problemas de búsqueda
- Estrategias de búsqueda no informada
  - Búsqueda primero en profundidad
  - Búsqueda primero en anchura
  - Búsqueda de coste uniforme

Estrategias de búsqueda no informada: son las que no reciben información sobre el problema más allá de su definición



Agente aspirador (recordatorio)

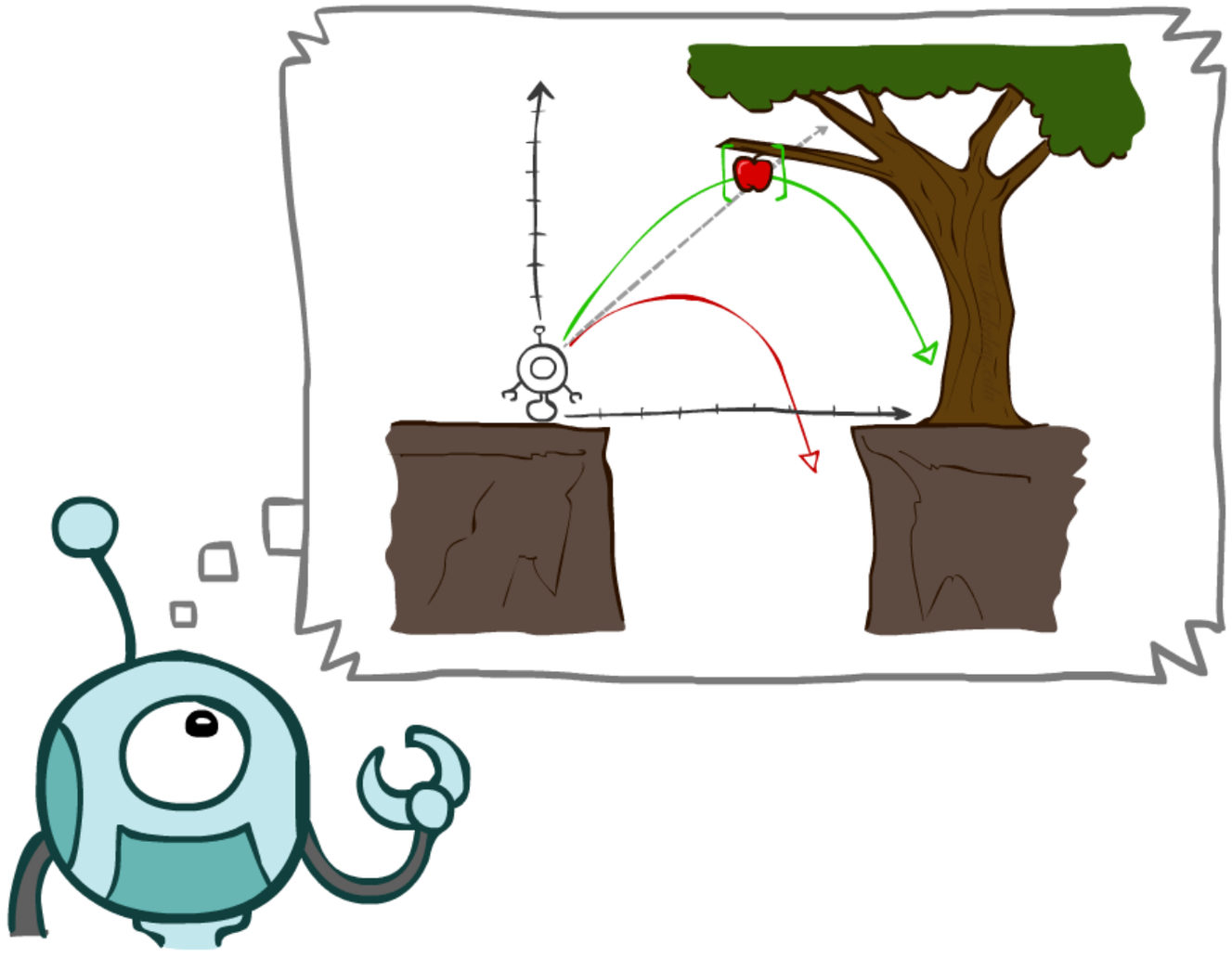


Programa del agente

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

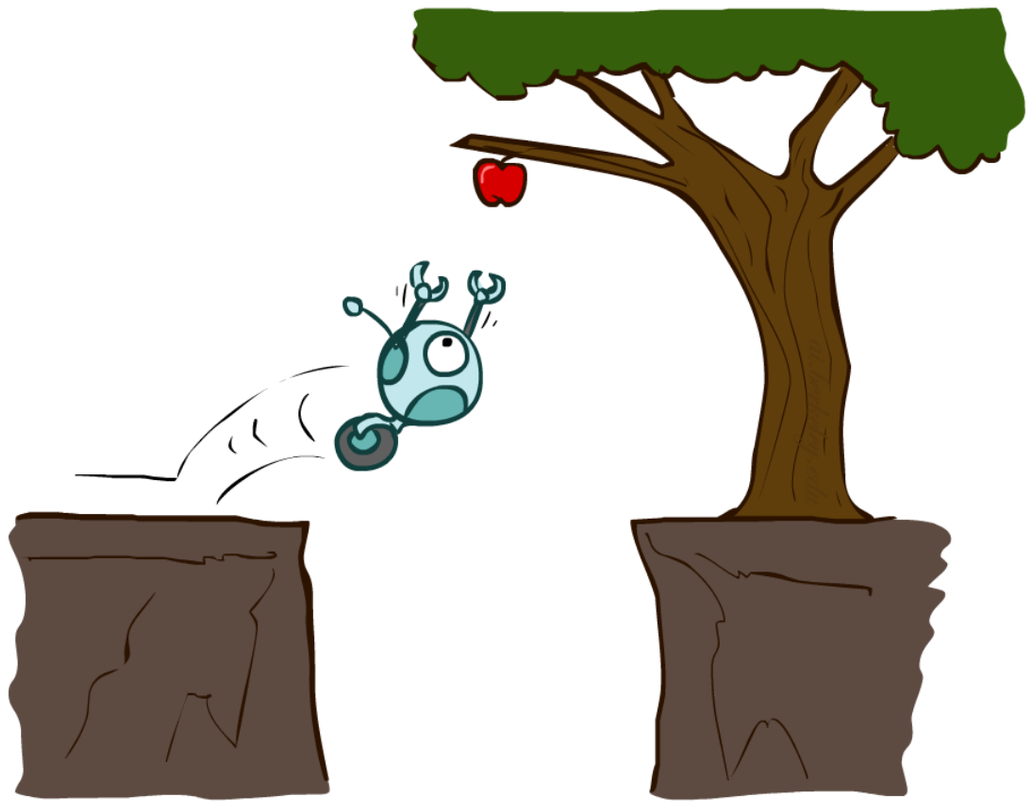
si *estado* = *Sucio* **entonces devolver** *Aspirar*
de otra forma, si *localización* = *A* **entonces devolver** *Derecha*
de otra forma, si *localización* = *B* **entonces devolver** *Izquierda*

Agentes que planifican con antelación

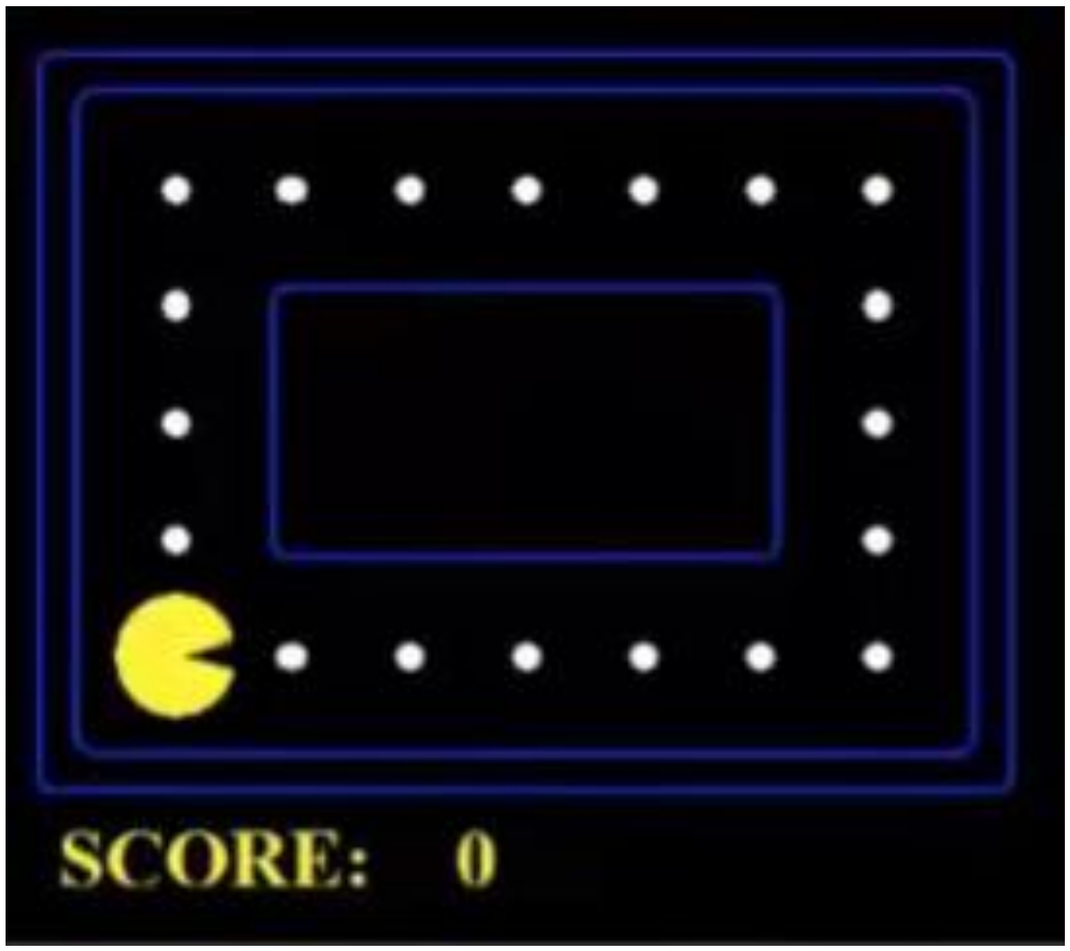


Agentes reactivos (simple y basado en modelo)

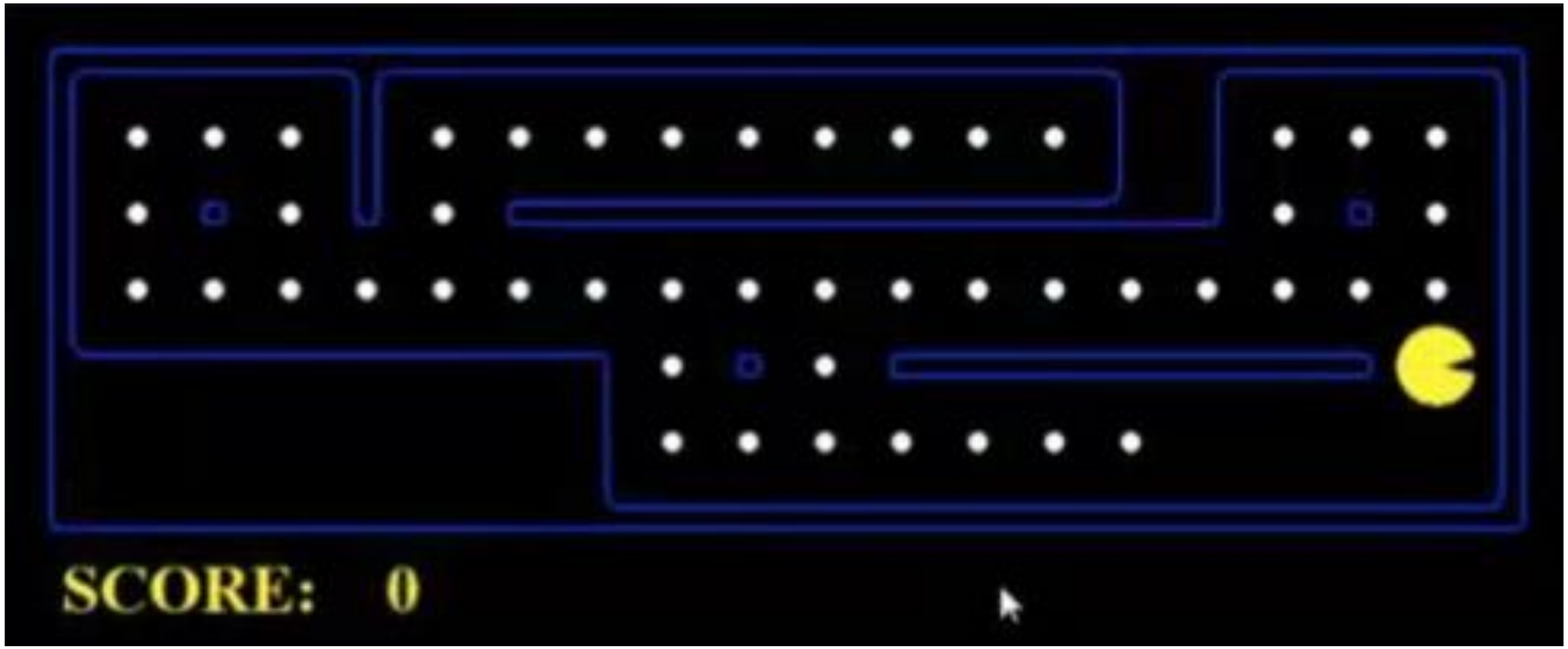
- Los agentes reactivos…
  - Eligen la acción basándose en la percepción actual (y puede que alguna de las anteriores)
  - Pueden tener memoria o un modelo del estado actual del mundo
  - No consideran las consecuencias futuras de sus acciones
  - Consideran **cómo es el mundo ahora**
- ¿Puede ser racional un agente reactivo?



Demo agente reactivo óptimo

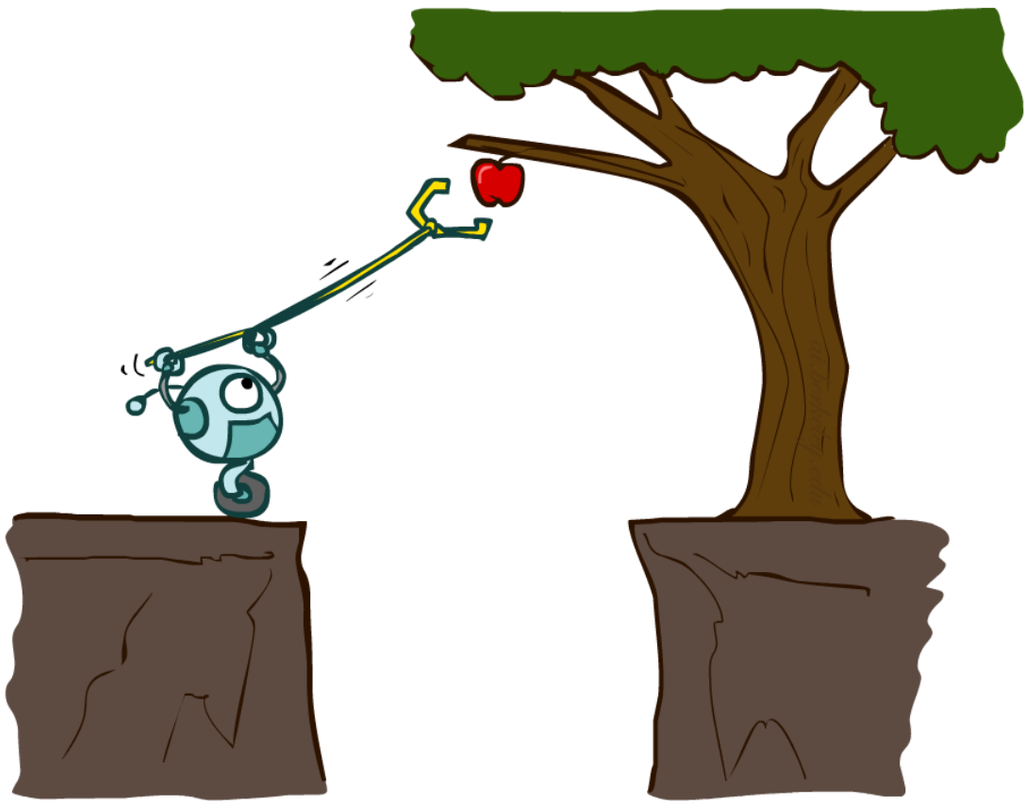


Demo agente reactivo no tan óptimo

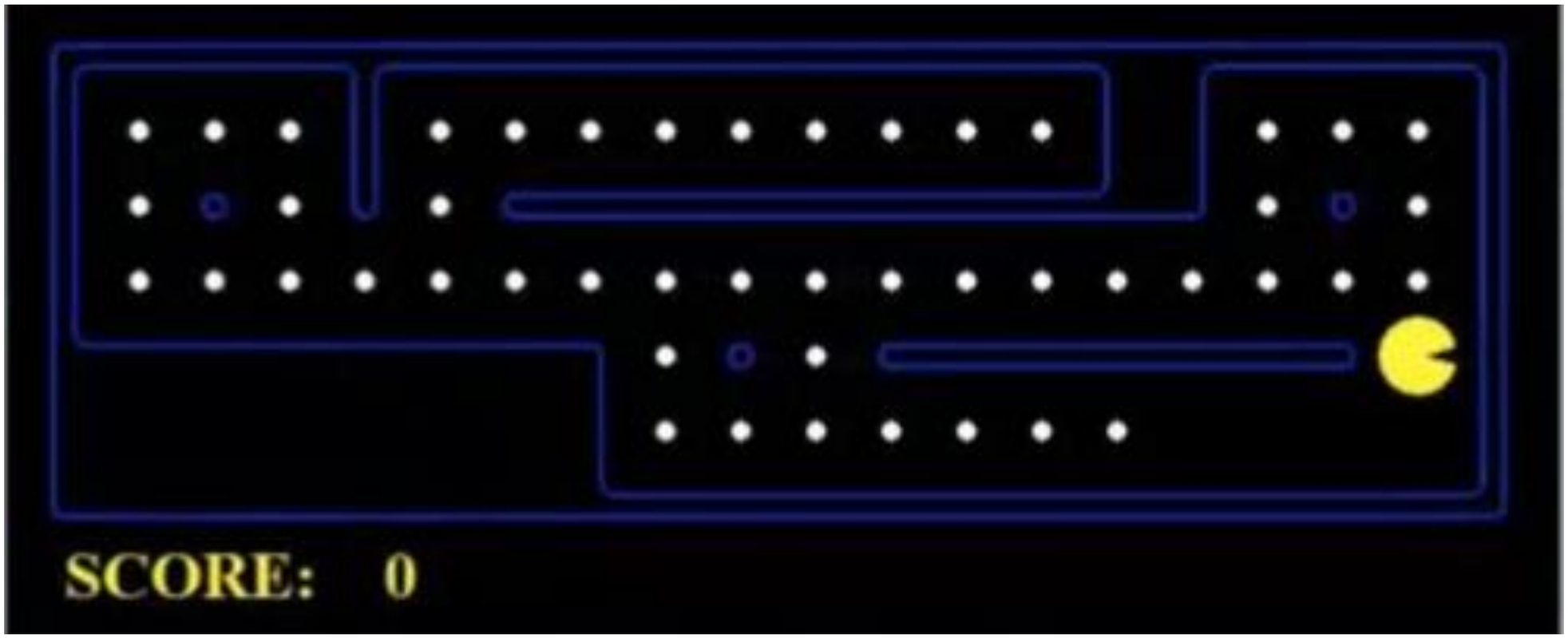


Agentes “planificadores” (objetivos/utilidad)

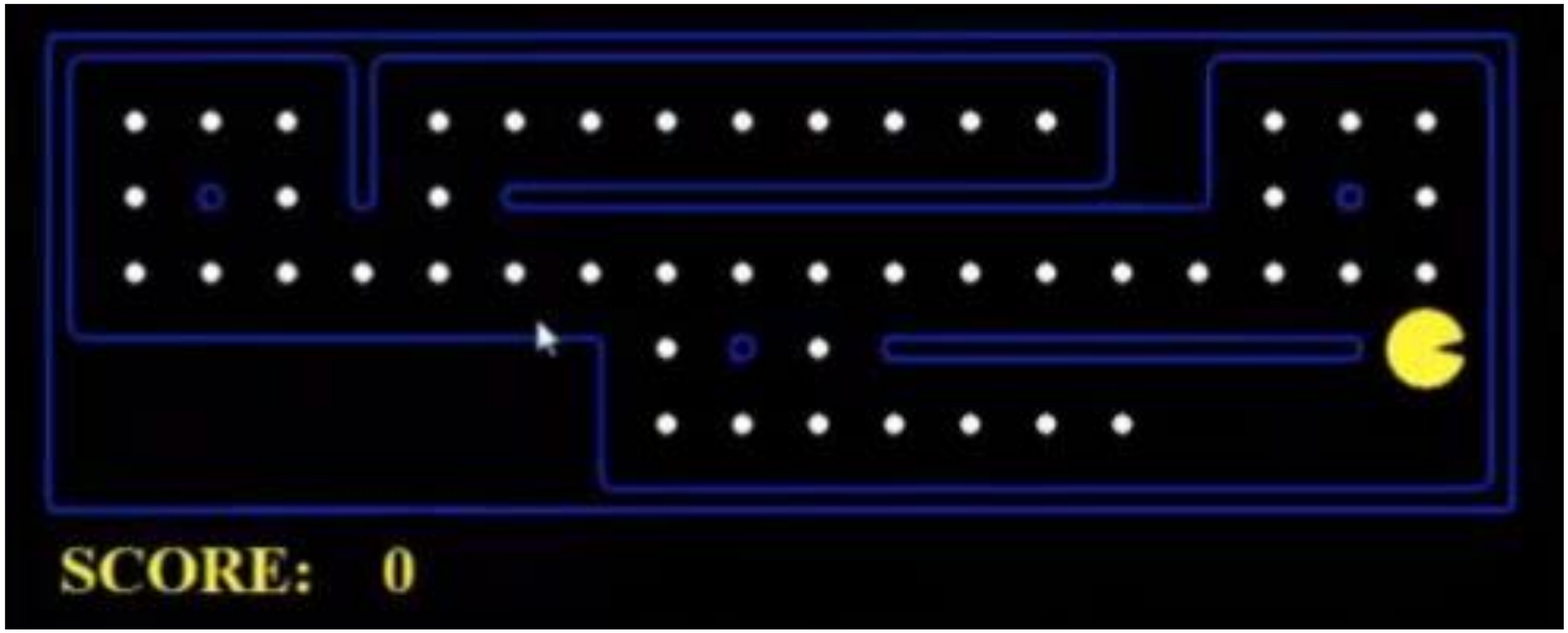
- Los agentes planificadores….
  - Se preguntan “y si”
  - Decisiones basadas en las (hipotéticas) consecuencias de sus acciones
  - Deben tener un modelo de cómo evoluciona el mundo en respuesta a las acciones
  - Deben formular un objetivo (test): estado o conjunto de estados del entorno
  - Consideran **cómo sería el mundo**



Demo agente “planificador”



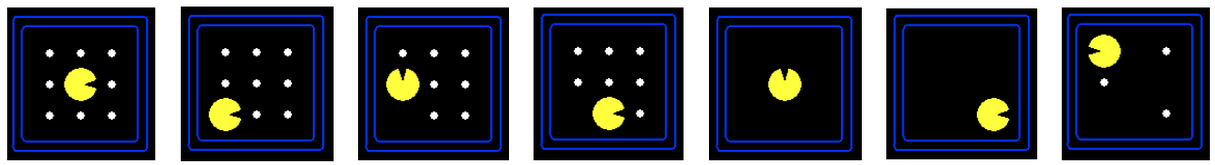
Demo agente que lo planifica todo



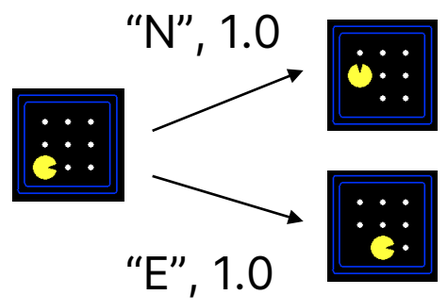
Problemas de búsqueda

Problemas de búsqueda

- Un **problema de búsqueda** consta de:
  - Un espacio de estados (estados alcanzables)



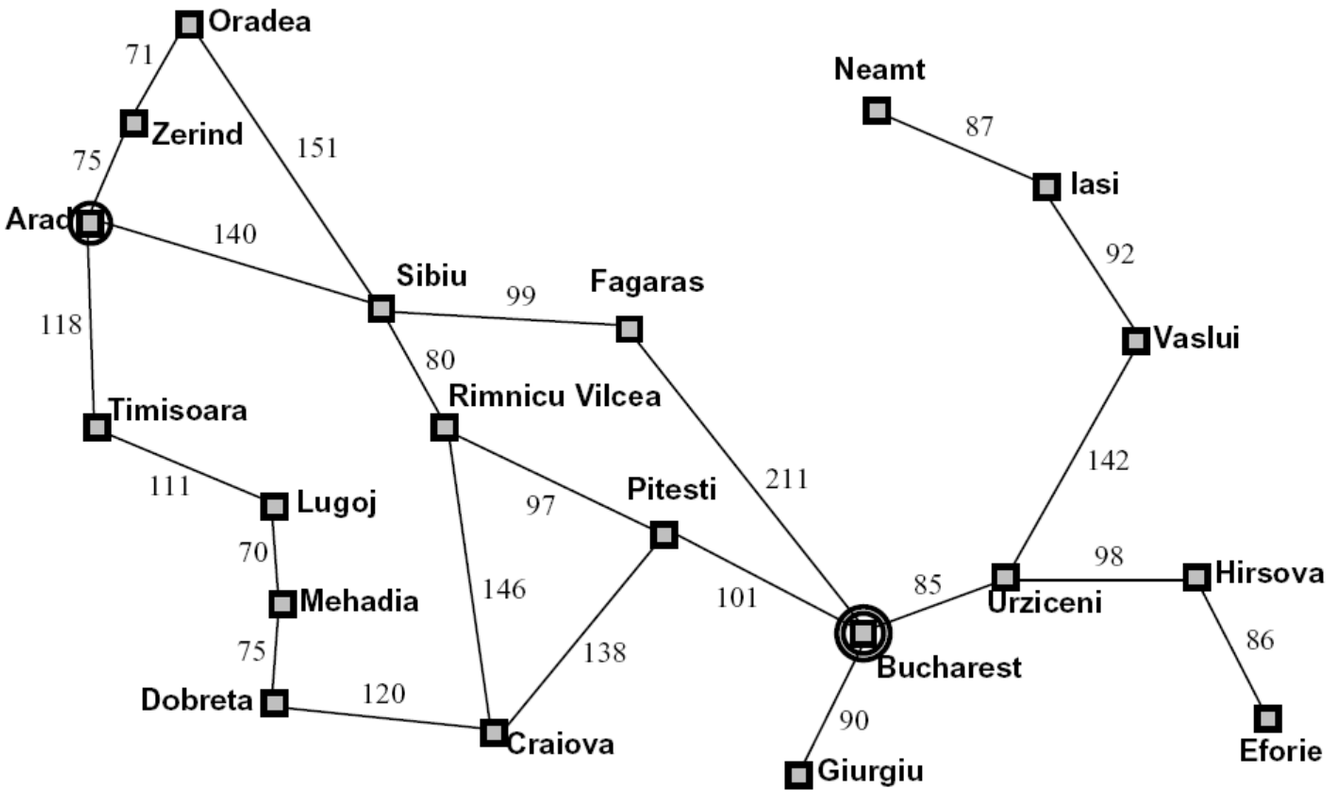
  - Una función sucesora (con acciones, costes)



    - Un estado inicial y un test objetivo (“¿este estado es el objetivo?”)
- Una **solución** es una secuencia de acciones (un plan) que transforma el estado inicial en un estado objetivo
- En este bloque: representación atómica del entorno y problemas que siempre se pueden resolver con una secuencia fija de acciones (no depende de percepciones futuras)

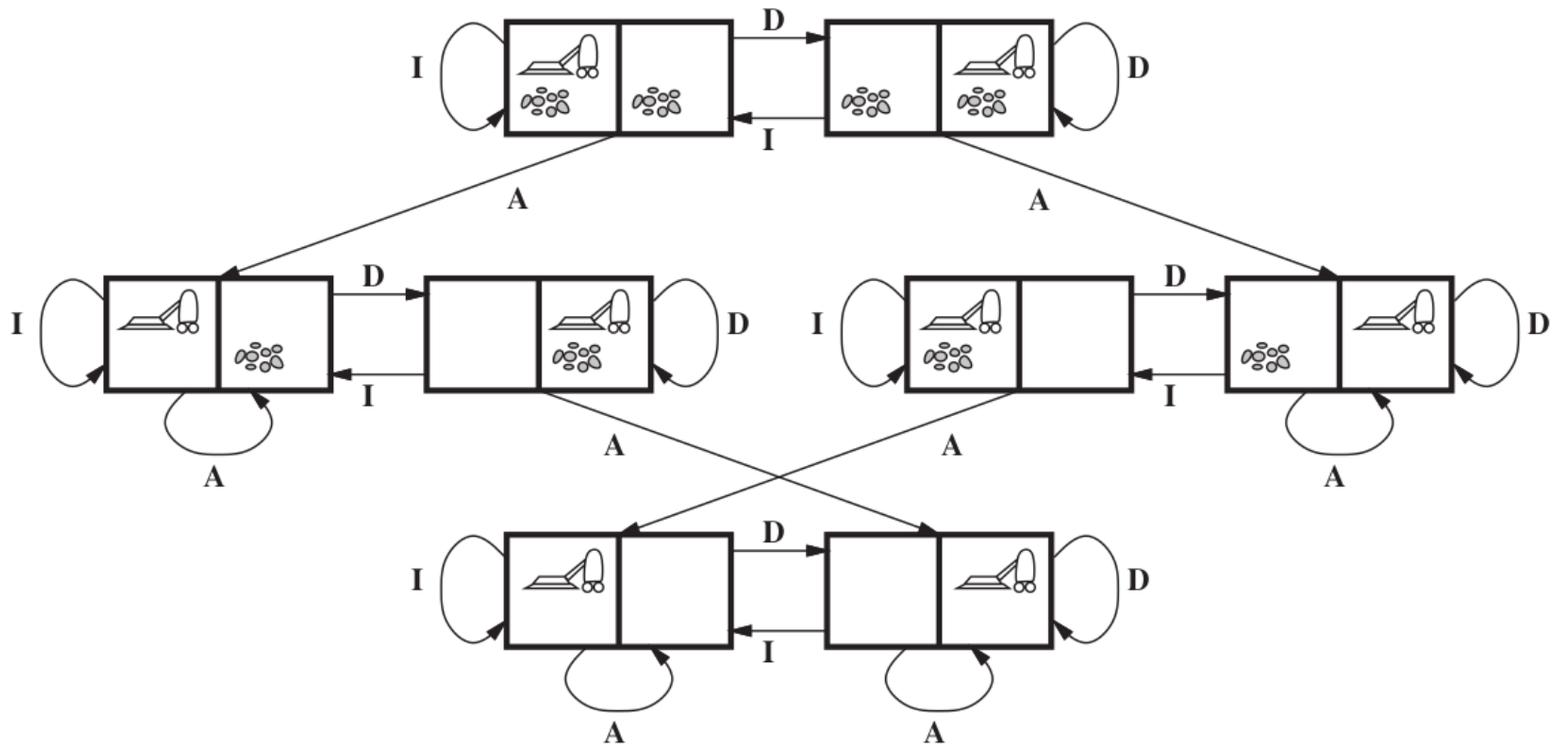
Los problemas de búsqueda son modelos

Ejemplo: viajar en Rumanía



- **Espacio de estados:**
  - Ciudades
- **Función sucesora:**
  - Carreteras: ir a la siguiente ciudad con coste = distancia
- **Estado inicial:**
  - Arad
- **Test objetivo:**
  - ¿Estado == Bucarest?
- **¿Solución?**

Ejemplo: el mundo de la aspiradora (vuelve)



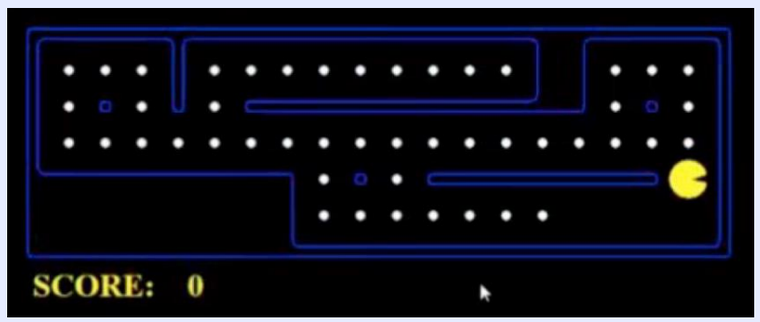
- **Espacio de estados:**
  - 2 localizaciones con dos configuraciones (sucio/limpio): 8 estados
- **Estado inicial:**
  - Cualquiera puede serlo
- **Función sucesora:**
  - Genera los estados legales que resultan al intentar las tres acciones (I, D, Aspirar)
- **Test objetivo:**
  - Todos los cuadrados limpios

Aplicaciones en el mundo real

- **Problemas de búsqueda de rutas:** sitios web, Google Maps, plataformas de viaje
- **Problemas tipo "del viajante" (*travelling salesman problem*):** visita todas las ciudades al menos una vez, no repitas ciudad
- **Creación de circuitos integrados:** colocar millones de componentes y conexiones en un chip para minimizar el área, los retardos, las capacitancias parásitas, etc.
- **Navegación de robots:** generalización de los problemas de búsqueda de rutas moviéndose por un espacio continuo

¿Qué hay en un espacio de estados?

El **estado del mundo** incluye hasta el último detalle del entorno

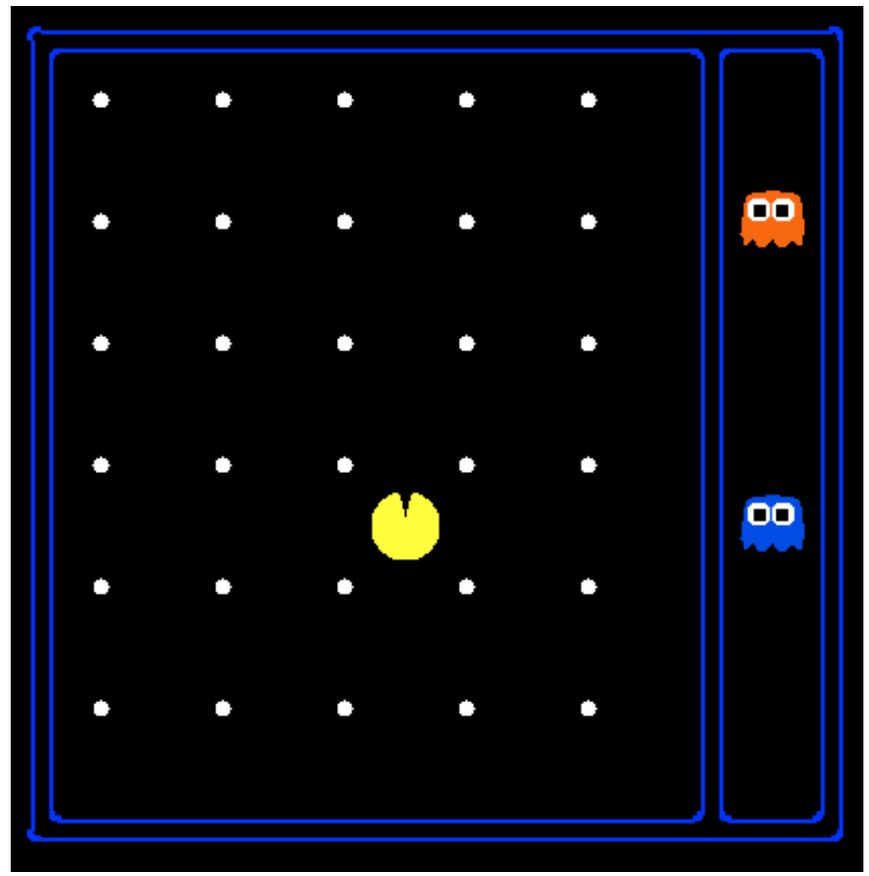


Un **estado de búsqueda** solo mantiene los detalles necesarios para planificar (abstracción)

- Problema: Calcular ruta
  - Estados: localización (x,y)
  - Acciones: NSEO
  - Sucesora: actualizar posición
  - Test objetivo: (x,y)==Final
- Problema: Comerse los puntos
  - Estados: {(x,y), booleanos pts.}
  - Acciones: NSEO
  - Sucesora: actualizar la localización y a veces un booleano de punto
  - Test objetivo: ptos==False

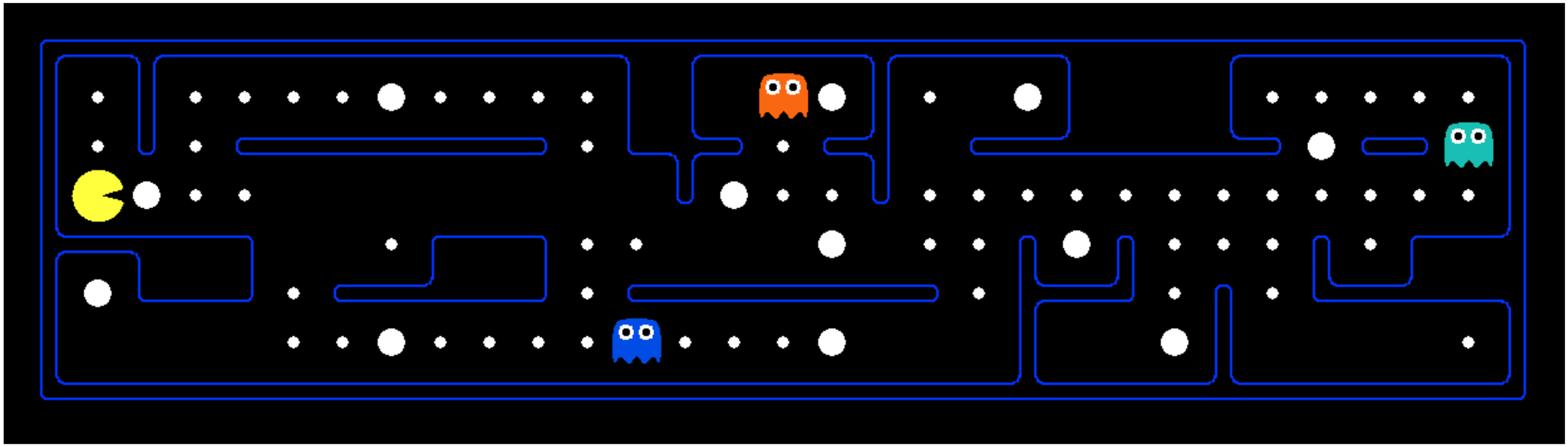
¿Tamaños del espacio de estados?

- **Estado del mundo:**
  - Posiciones del agente: 120
  - Comida (puntos): 30
  - Posiciones de los fantasmas: 12
  - Orientación del agente: NSEO



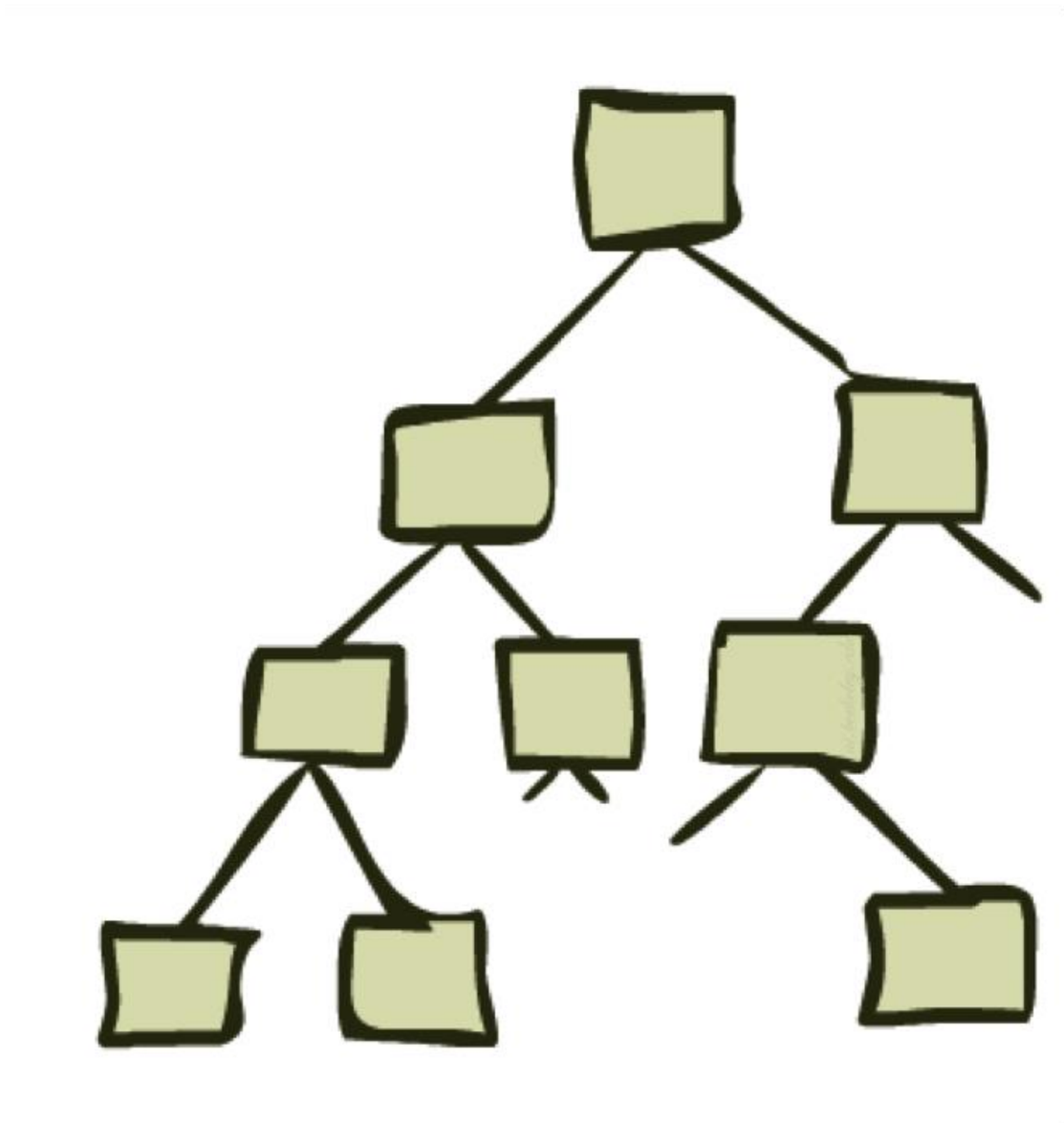
- **¿Cuántos…**
  - …estados del mundo?
    $120x(2^{30})x(12^{2})x4 = 7.4217035e+13$
  - …estados para “calcular ruta”?
    120
  - …estados para “comerse los puntos”?
    $120x(2^{30}) = 128.849.018.880$

Ejercicio: Paso seguro



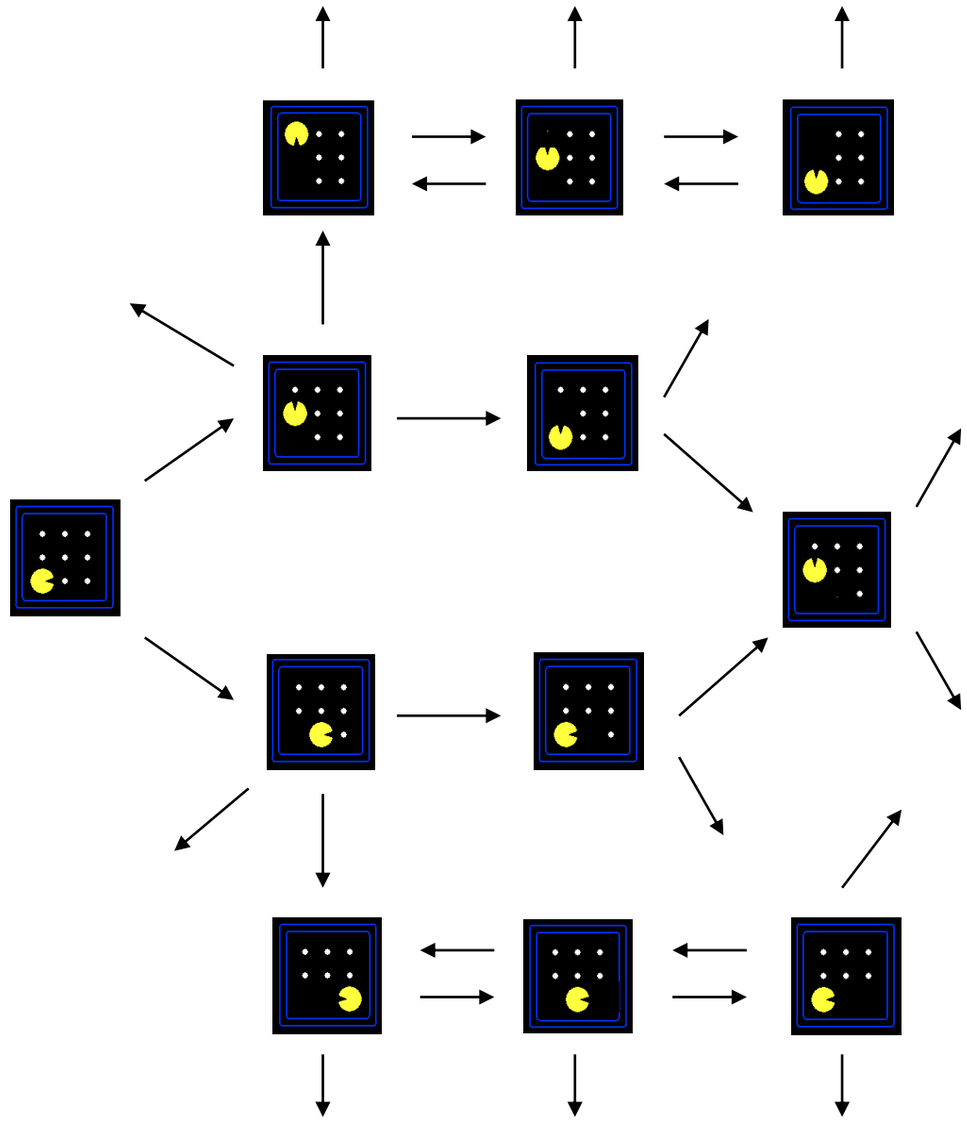
- Problema: comer todos los puntos mientras se mantiene a los fantasmas permanentemente asustados
- ¿Qué tiene que especificar el espacio de estados?
  - (position del agente, booleanos para los puntos, booleanos para las píldoras de poder, tiempo restante de poder)

Grafos del e. de estados y árboles de búsqueda



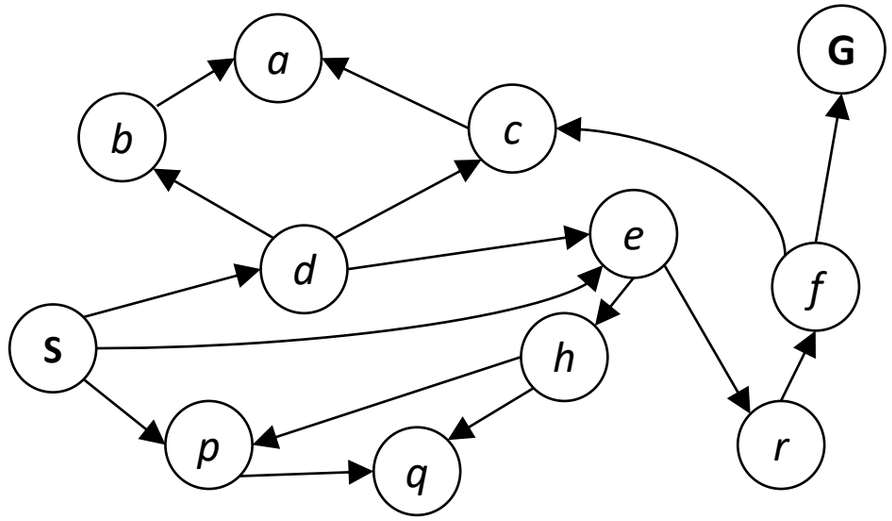
Grafos del espacio de estados

- **Grafo del espacio de estados:** una representación matemática de un problema de búsqueda
  - Los **nodos** son (abstracciones) de las configuraciones del mundo
  - Los **arcos** representan sucesores (resultados de las acciones)
  - El **test objetivo** es un conjunto de nodos objetivo (o solo uno)
- En un grafo del espacio de estados, cada estado ocurre **solo una vez**
- Raramente se puede construir entero en memoria (es muy grande), pero es útil como idea



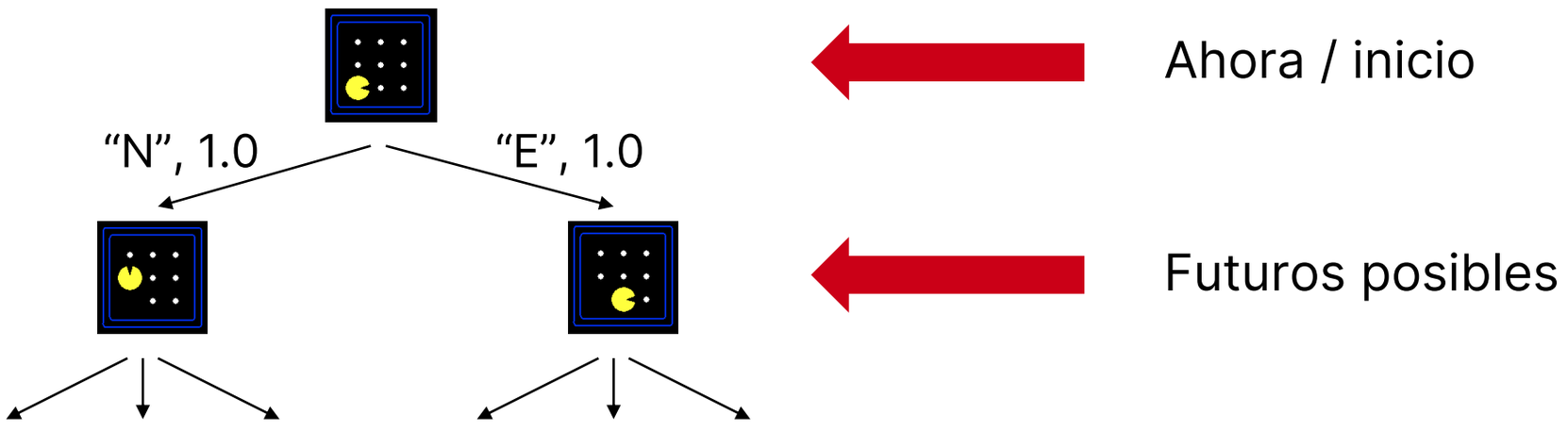
Grafos del espacio de estados

- **Grafo del espacio de estados:** una representación matemática de un problema de búsqueda
  - Los **nodos** son (abstracciones) de las configuraciones del mundo
  - Los **arcos** representan sucesores (resultados de las acciones)
  - El **test objetivo** es un conjunto de nodos objetivo (o solo uno)
- En un grafo del espacio de estados, cada estado ocurre **solo una vez**
- Raramente se puede construir entero en memoria (es muy grande), pero es útil como idea



Grafo del espacio de estados minúsculo para un problema de búsqueda minúsculo

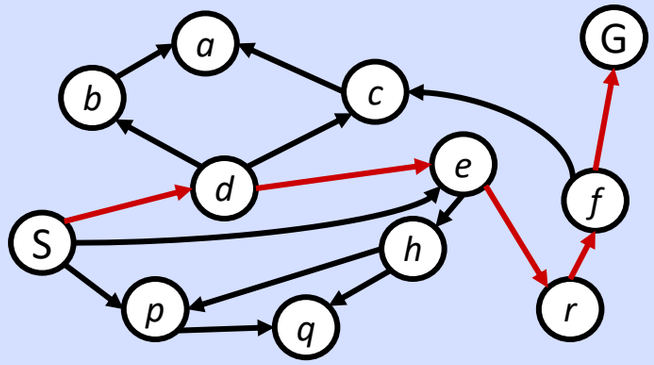
Árboles de búsqueda



- Un **árbol de búsqueda:**
  - Un árbol “y si” de planes y sus resultados
  - El estado inicial es el nodo raíz
  - Los hijos corresponden a sucesores
  - Los nodos muestran estados, pero corresponden a **planes** que alcanzan esos estados
  - Para la mayoría de problemas, es **inviable** construir el árbol completo

Grafos esp. de estados vs. árboles de búsqueda

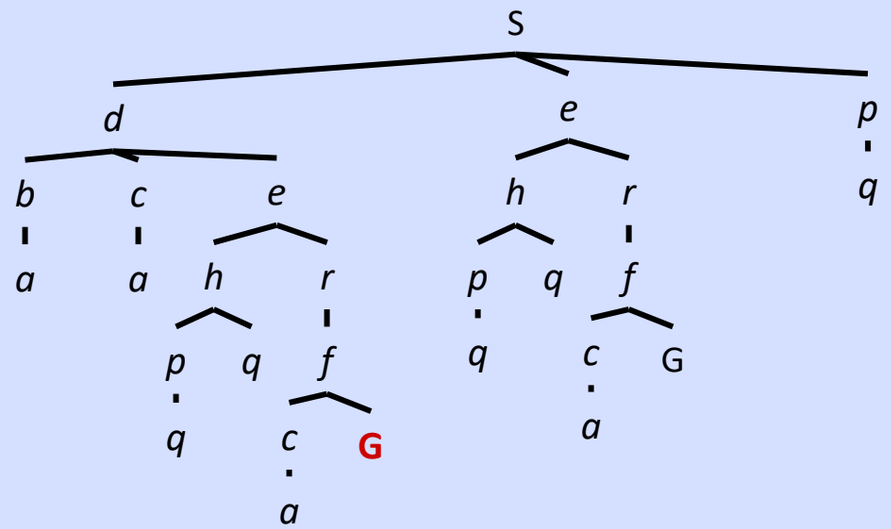
Grafo esp. estados



Cada NODO del árbol de búsqueda es un CAMINO completo en el grafo del espacio de estados.

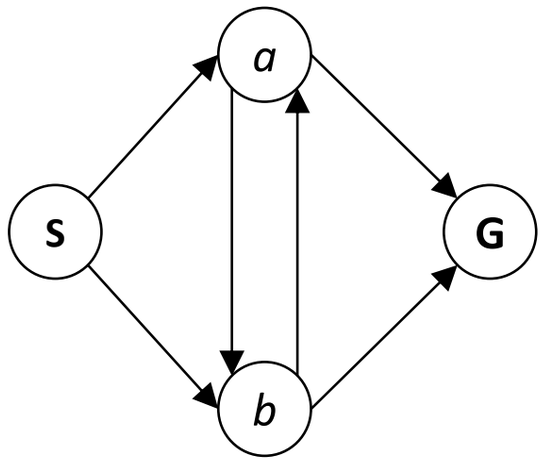
Construimos ambas cosas según se necesitan, y construimos lo mínimo posible.

Árbol de búsqueda

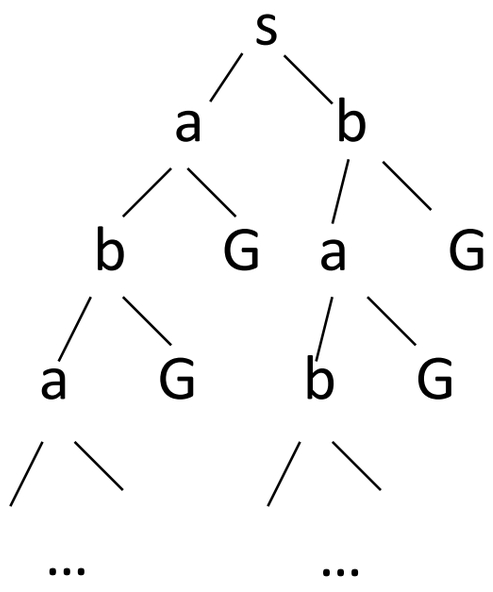


Ejercicio: esp. estados vs. árboles de búsqueda

Sea este grafo de cuatro estados:

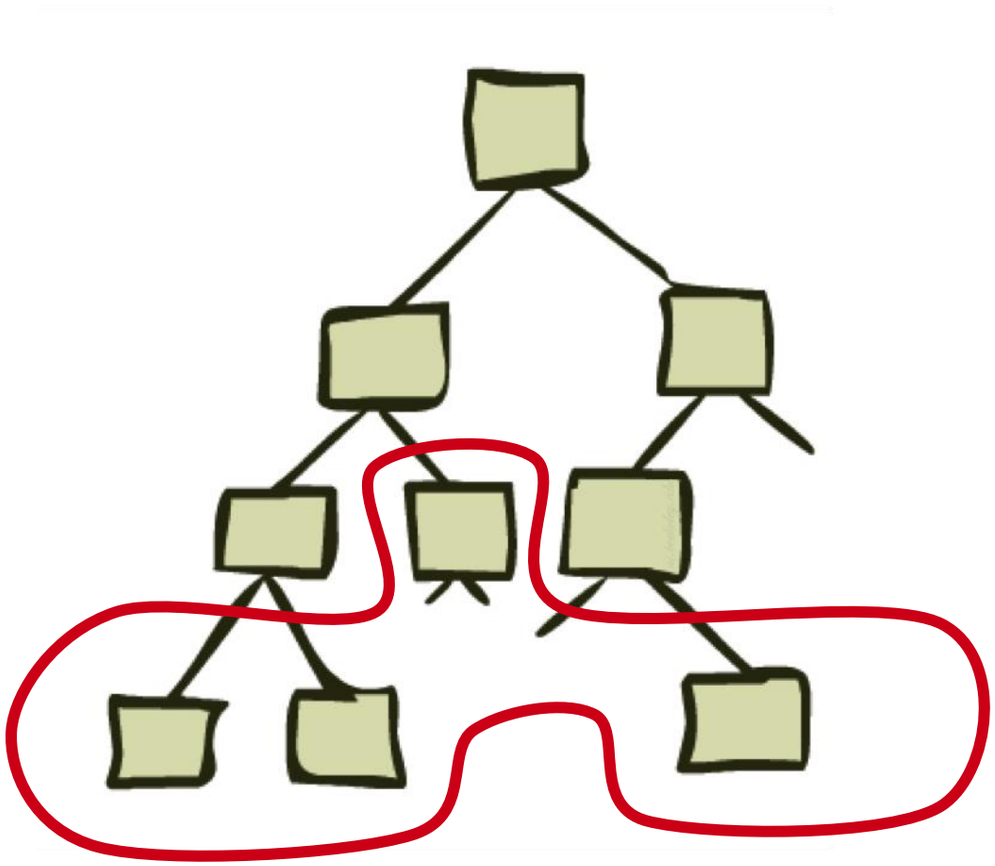


¿Cómo de grande es el árbol de búsqueda? (desde s)

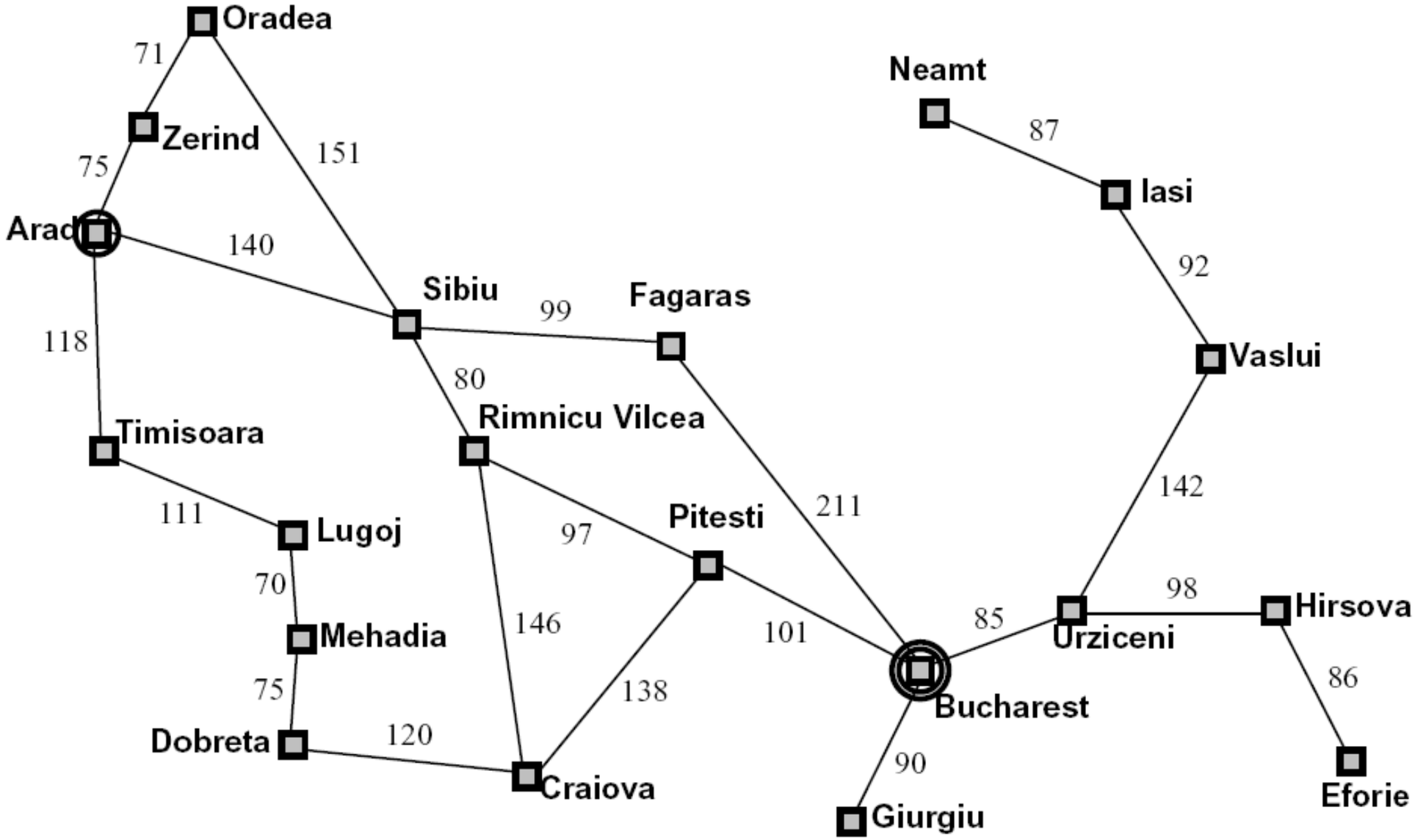


Importante: se repite mucha estructura en el árbol de búsqueda: caminos en bucle y caminos redundantes

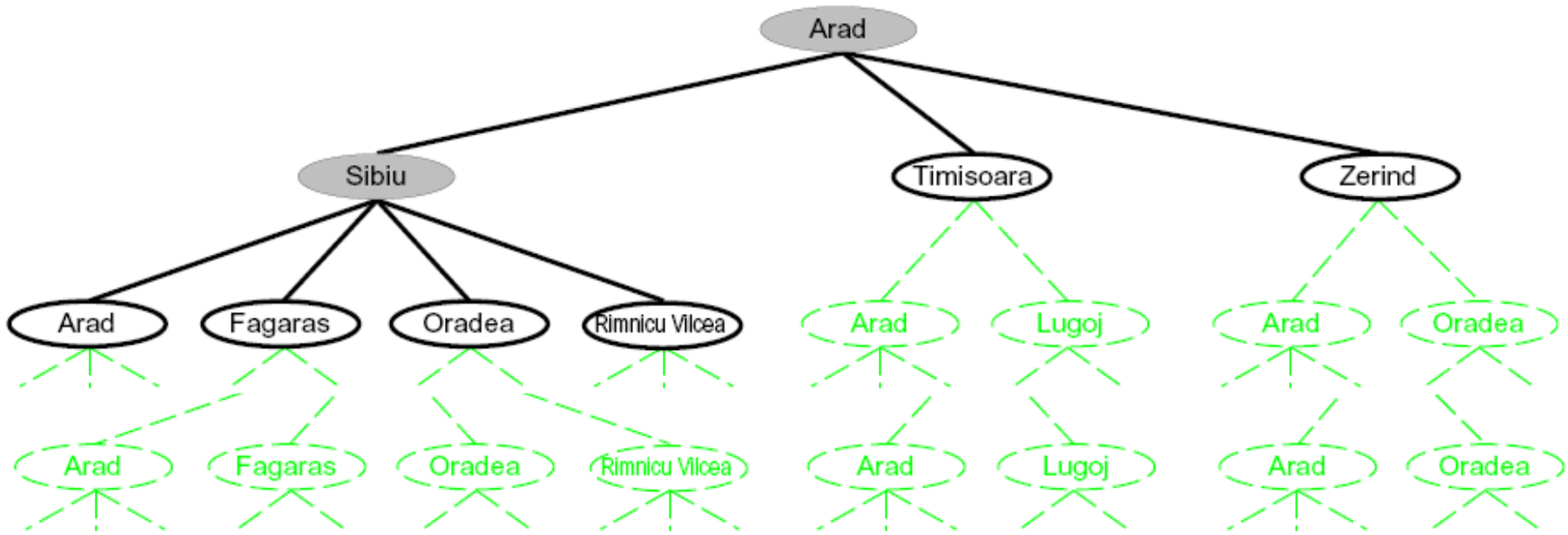
Búsqueda en árboles



Ejemplo de búsqueda: Rumania



Buscando con un árbol de búsqueda



- Búsqueda:
  - **Expandir** planes potenciales (nodos del árbol)
  - Mantener una **frontera** de planes parciales en estudio
  - Tratar de expandir tan pocos nodos como sea posible

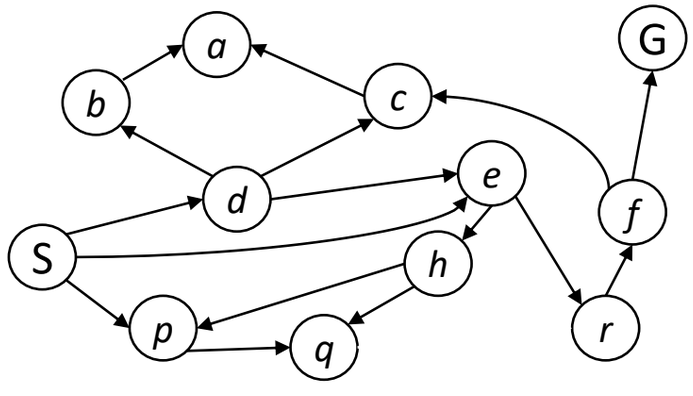
Algoritmo general de búsqueda en árboles

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 **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

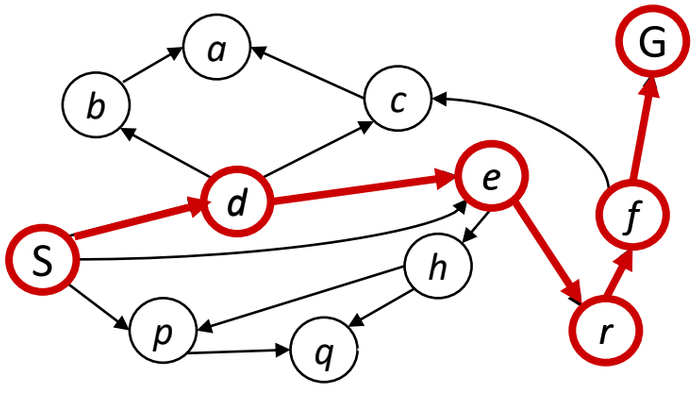
- Ideas importantes:
  - Frontera
  - Expansión
  - Estrategia de exploración

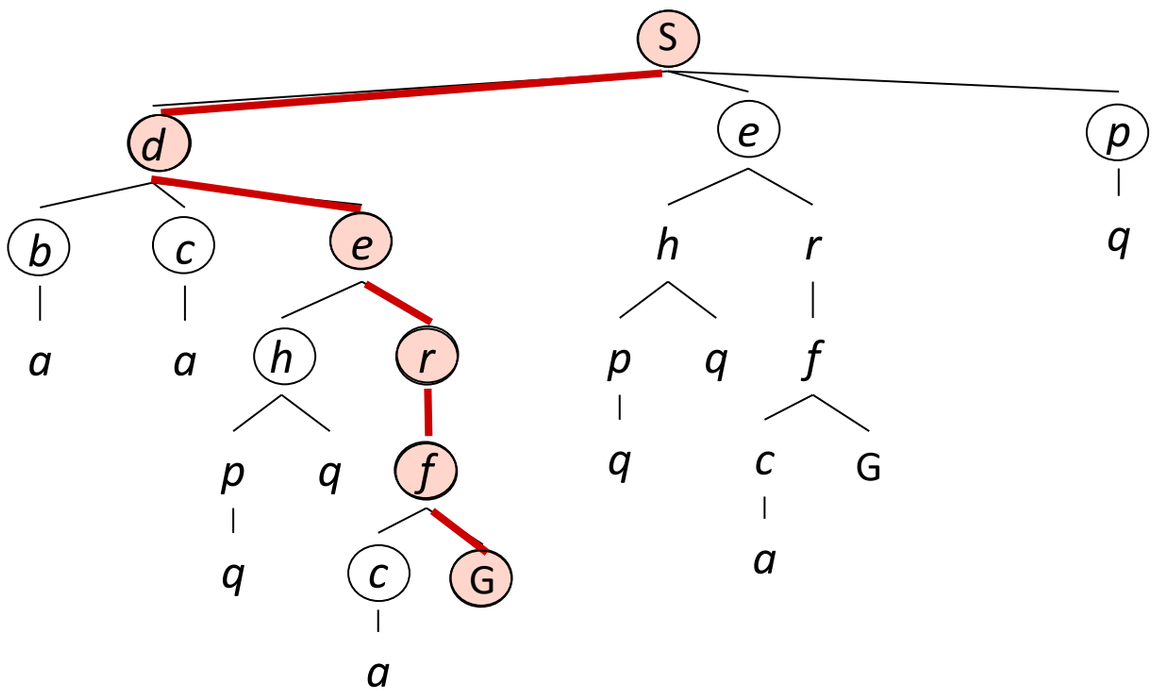
- Cuestión principal: ¿qué nodos de la frontera explorar?

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

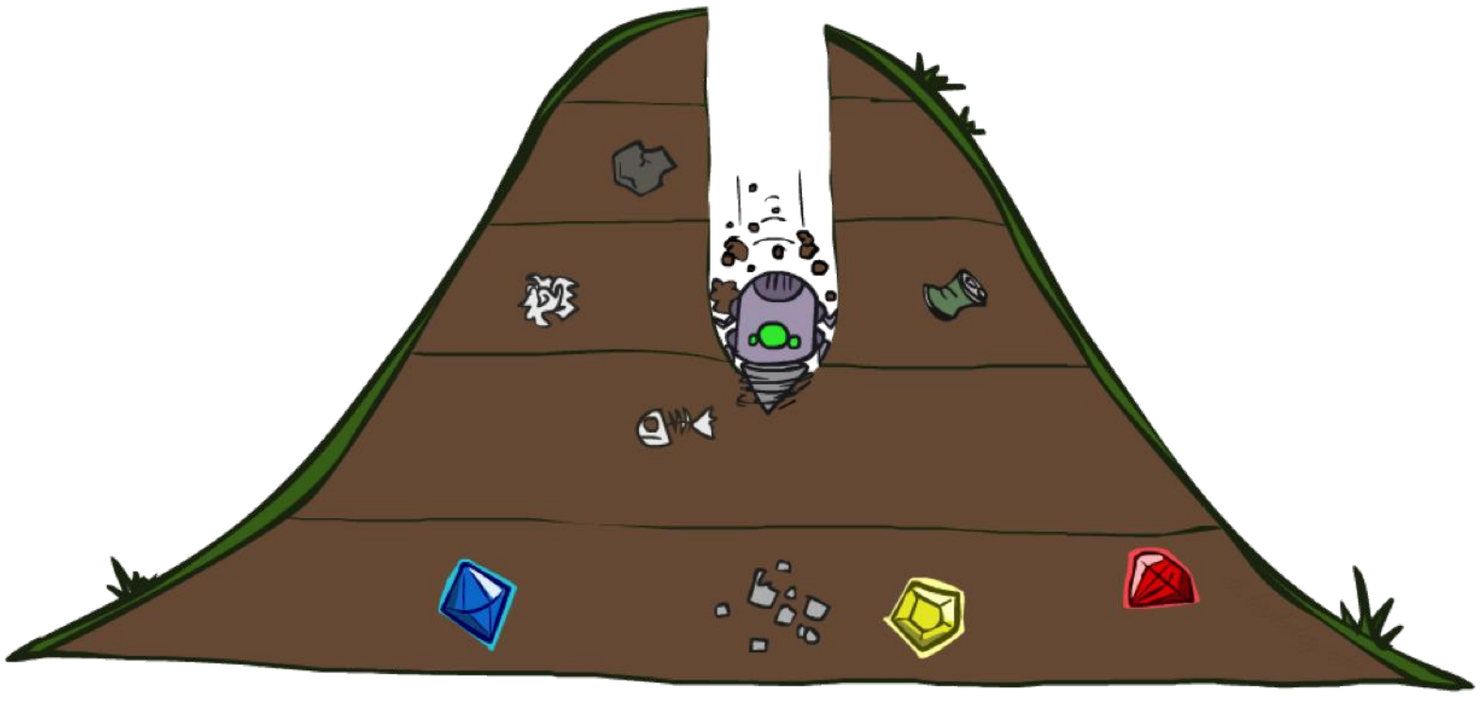
~~s → d → e → r~~

~~s → d → e → r → f~~

s → d → e → r → f → c

~~s → d → e → r → f → G~~

Búsqueda primero en profundidad (BPP)

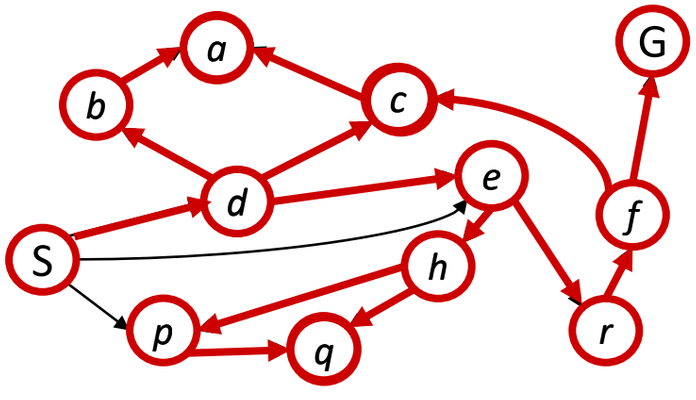


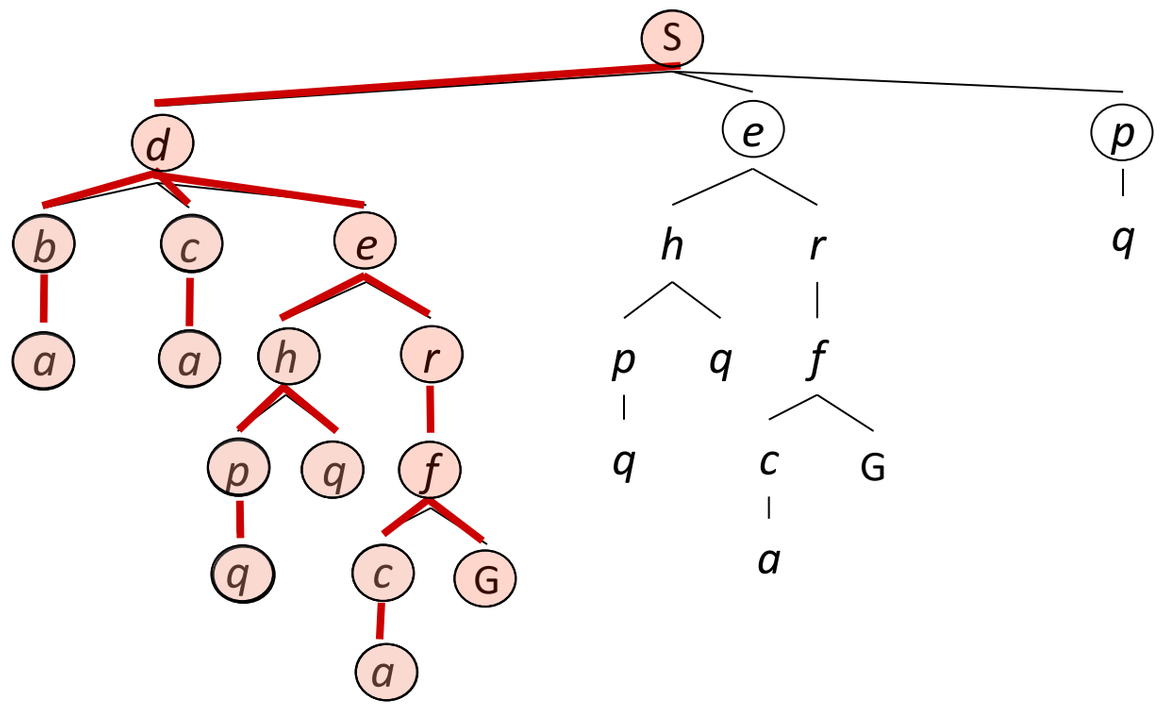
- Depth-First Search (DFS)

Búsqueda primero en profundidad

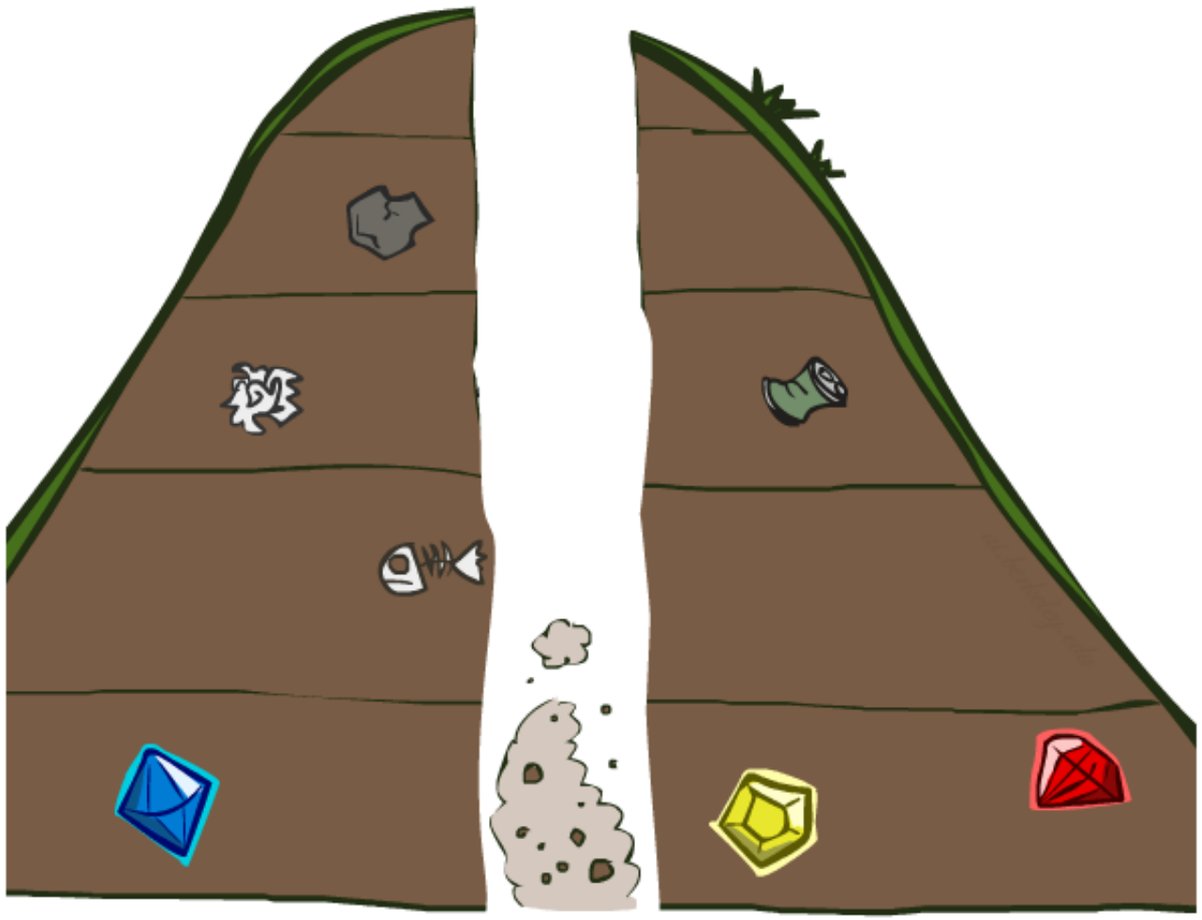
Estrategia: ampliar primero el nodo más profundo

Implementación: La frontera es una pila LIFO





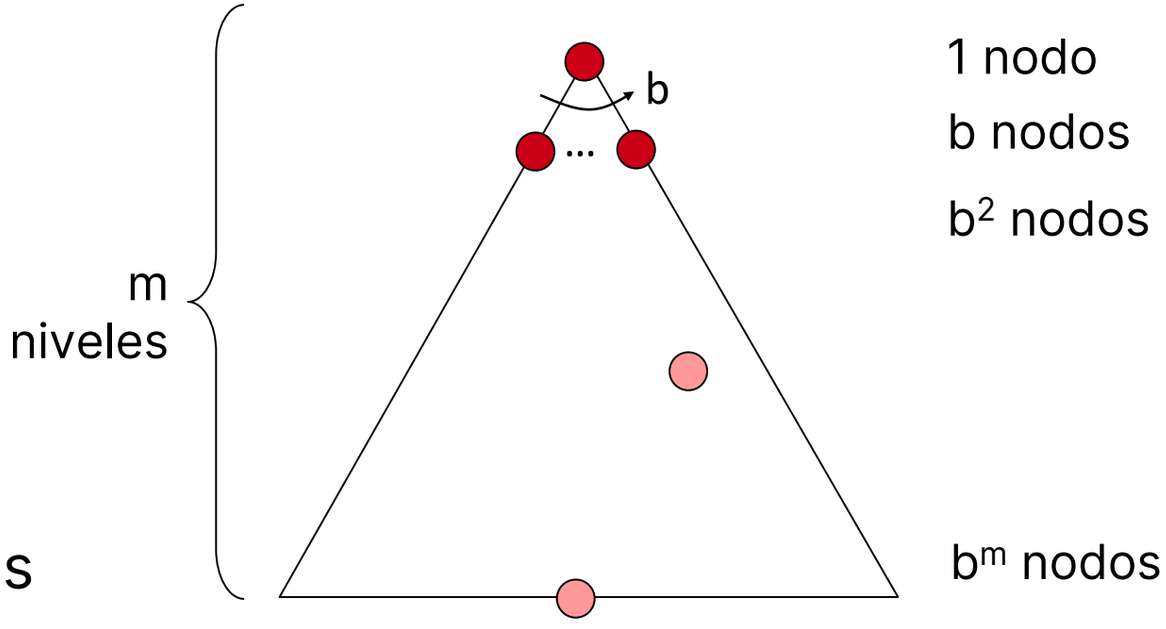
Estrategias de búsqueda: propiedades



Estrategias de búsqueda: propiedades

- Completo: ¿Garantiza encontrar una solución si existe?
- Óptimo: ¿Garantiza encontrar el camino de menor coste?
- ¿Complejidad temporal?
- ¿Complejidad espacial?

- Dibujo del árbol:
  - b es el factor de ramificación*
  - m es la máxima profundidad**
  - Soluciones a varias profundidades

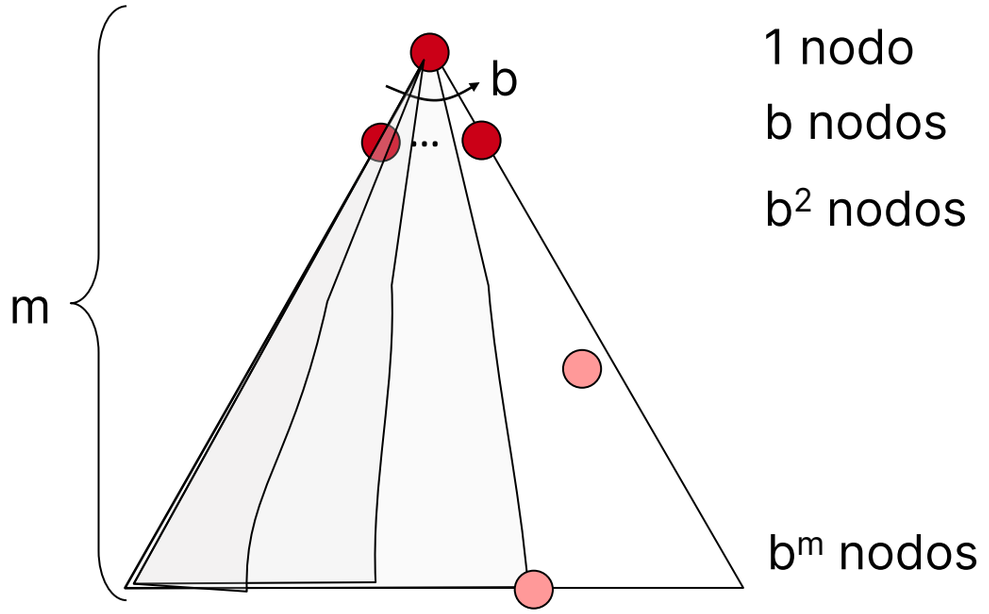


- ¿Número de nodos en el árbol entero?
  - $1 + b + b^2 + ... b^m = O(b^m)$

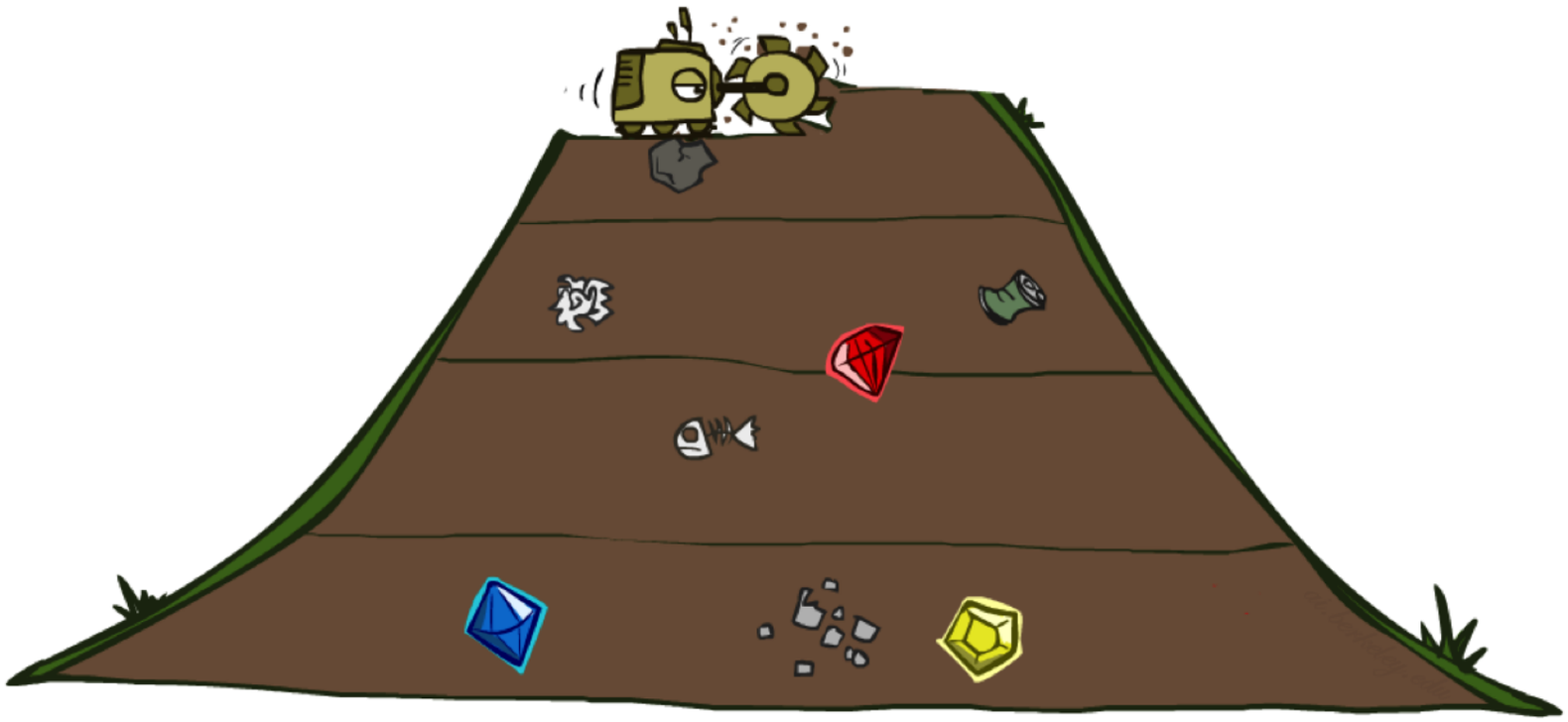
* Máximo número de sucesores que puede tener un nodo
** Longitud máxima que puede tener un camino en el espacio de estados

Propiedades primero en profundidad (BPP/DFS)

- ¿Qué nodos expande BPP?
  - El subárbol a la izquierda
  - Puede tener que procesar todo el árbol
  - Si m es finito, tarda $O(b^m)$
- ¿Cuánto espacio ocupa la frontera?
  - Sólo tiene hermanos en el camino a la raíz, por lo que O(bm)
- ¿Es completo?
  - m podría ser infinito, así que es completo **sólo si evitamos los ciclos** (lo veremos)
- ¿Es óptimo?
  - No, llega a la solución más “a la izquierda”, da igual la profundidad o el coste



Búsqueda primero en anchura (BPA)

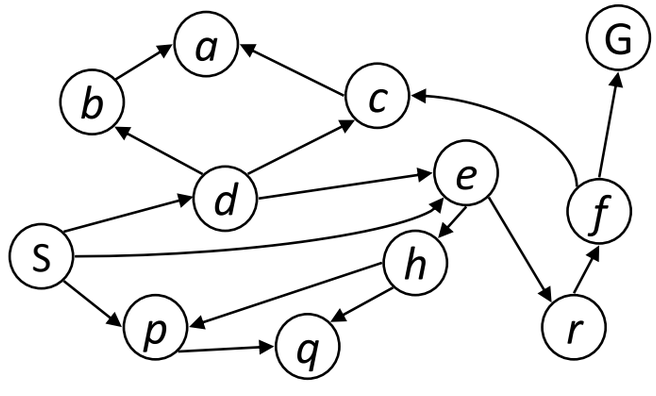


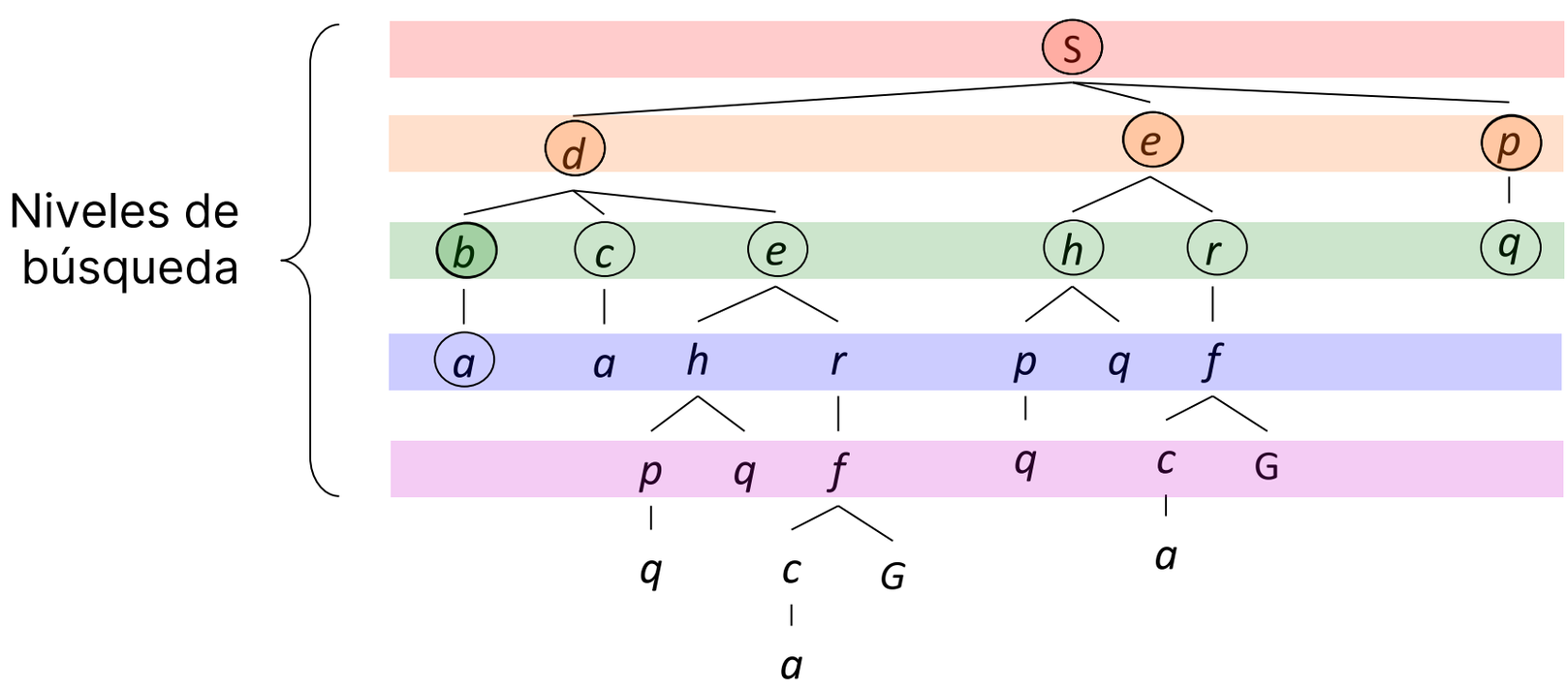
- Breadth-First Search (BFS)

Búsqueda primero en anchura

Estrategia: ampliar primero el nodo menos profundo

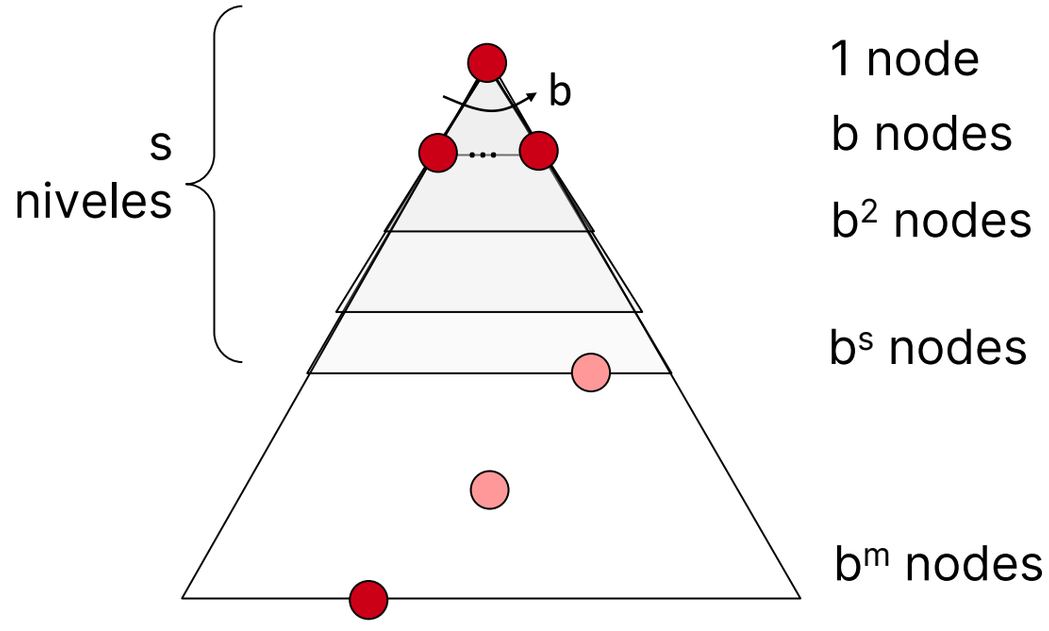
Implementación: La frontera es una cola FIFO



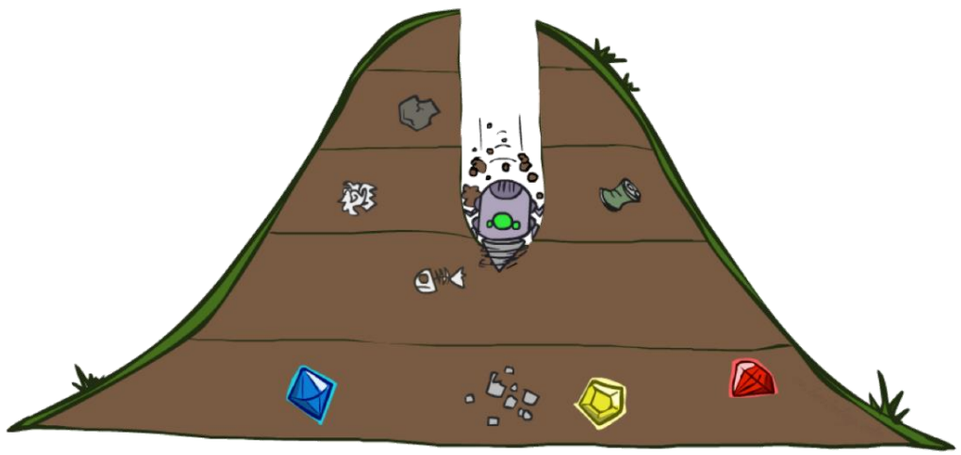


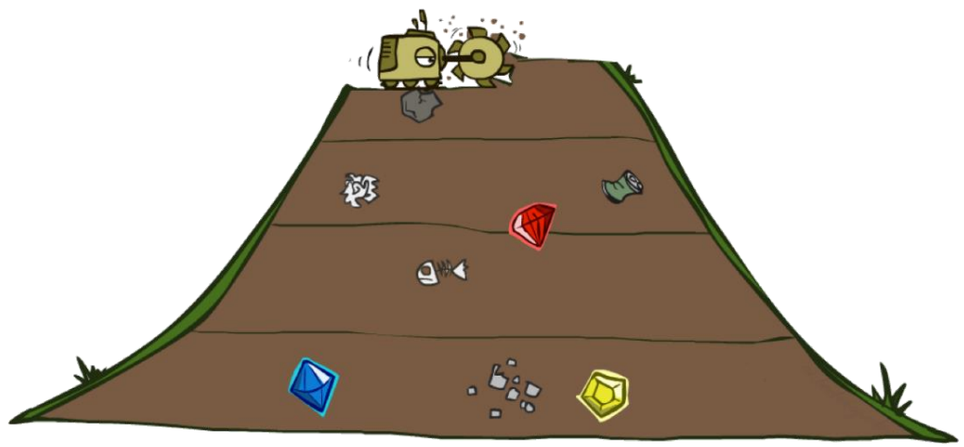
Propiedades primero en anchura (BPA/BFS)

- ¿Qué nodos expande BPA?
  - Procesa todos los nodos por encima de la solución más superficial
  - Sea s la profundidad de la solución
  - La búsqueda tarda $O(b^s)$
- ¿Cuánto ocupa la frontera?
  - Contiene aproximadamente el último nivel, por lo que $O(b^s)$
- ¿Es completa?
  - s debe ser finita si existe una solución, así que sí
- ¿Es óptima?
  - Sólo si todos los costes son 1 (veremos esto después)



Ejercicio: BPP vs BPA

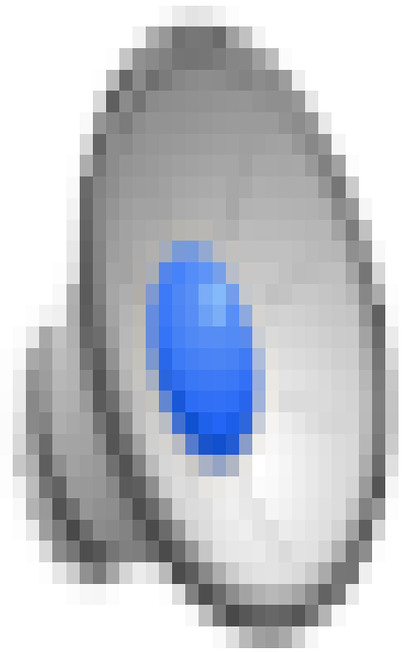




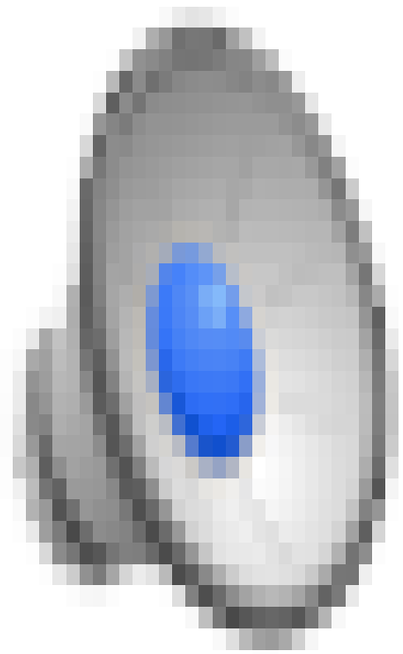
Ejercicio: BPP vs BPA

- ¿Cuándo funcionará mejor la búsqueda en anchura?
  - Cuando las soluciones tienden a ser poco profundas
- ¿Cuándo funcionará mejor la búsqueda en profundidad?
  - Cuando las soluciones tienden a ser profundas

Demo Laberinto Agua BPP/BPA (parte 1)

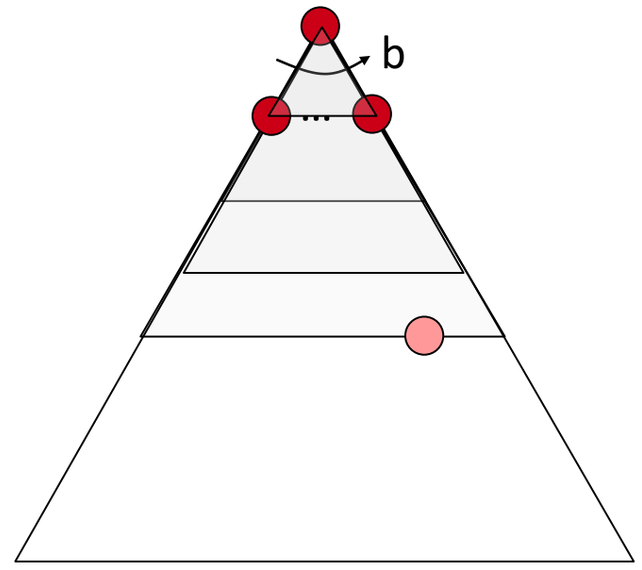


Demo Laberinto Agua BPP/BPA (parte 2)

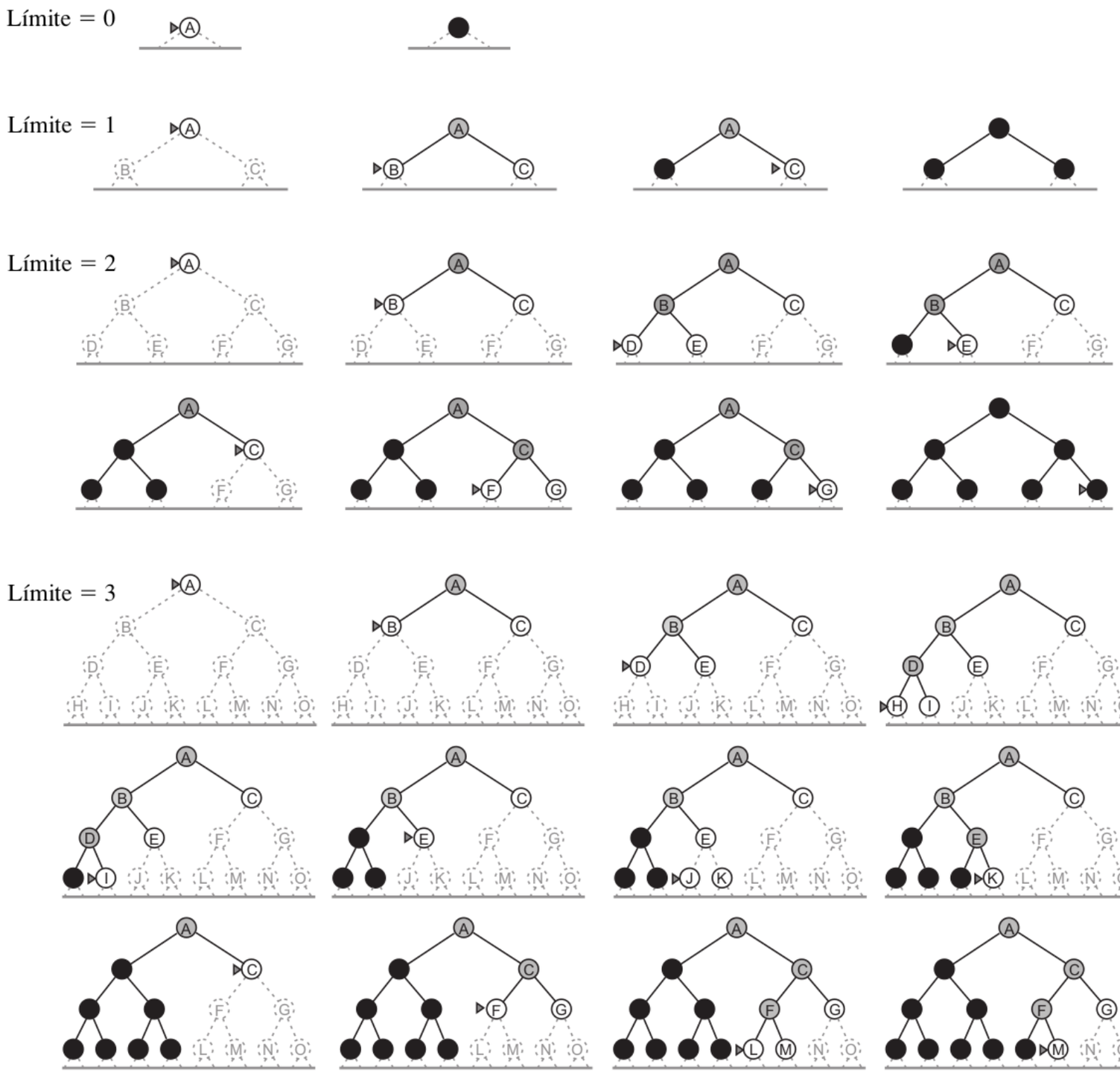


Profundidad iterativa

- Idea: obtener la ventaja de espacio de BPP con las ventajas de tiempo / solución superficial de BPA
  - Ejecutar BPP con límite de profundidad 1. Si no hay solución...
  - Ejecutar BPP con límite de profundidad 2. Si no hay solución...
  - Ejecutar BPP con límite de profundidad 3.

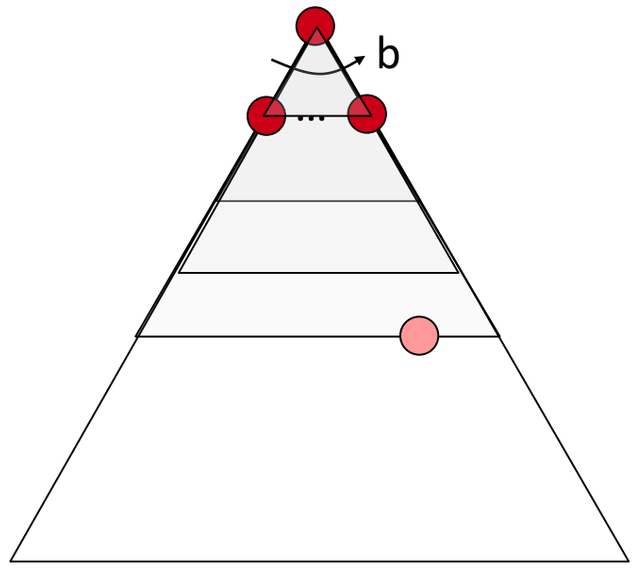


Profundidad iterativa



Profundidad iterativa

- Idea: obtener la ventaja de espacio de BPP con las ventajas de tiempo / solución superficial de BPA
  - Ejecutar BPP con límite de profundidad 1. Si no hay solución...
  - Ejecutar BPP con límite de profundidad 2. Si no hay solución...
  - Ejecutar BPP con límite de profundidad 3.

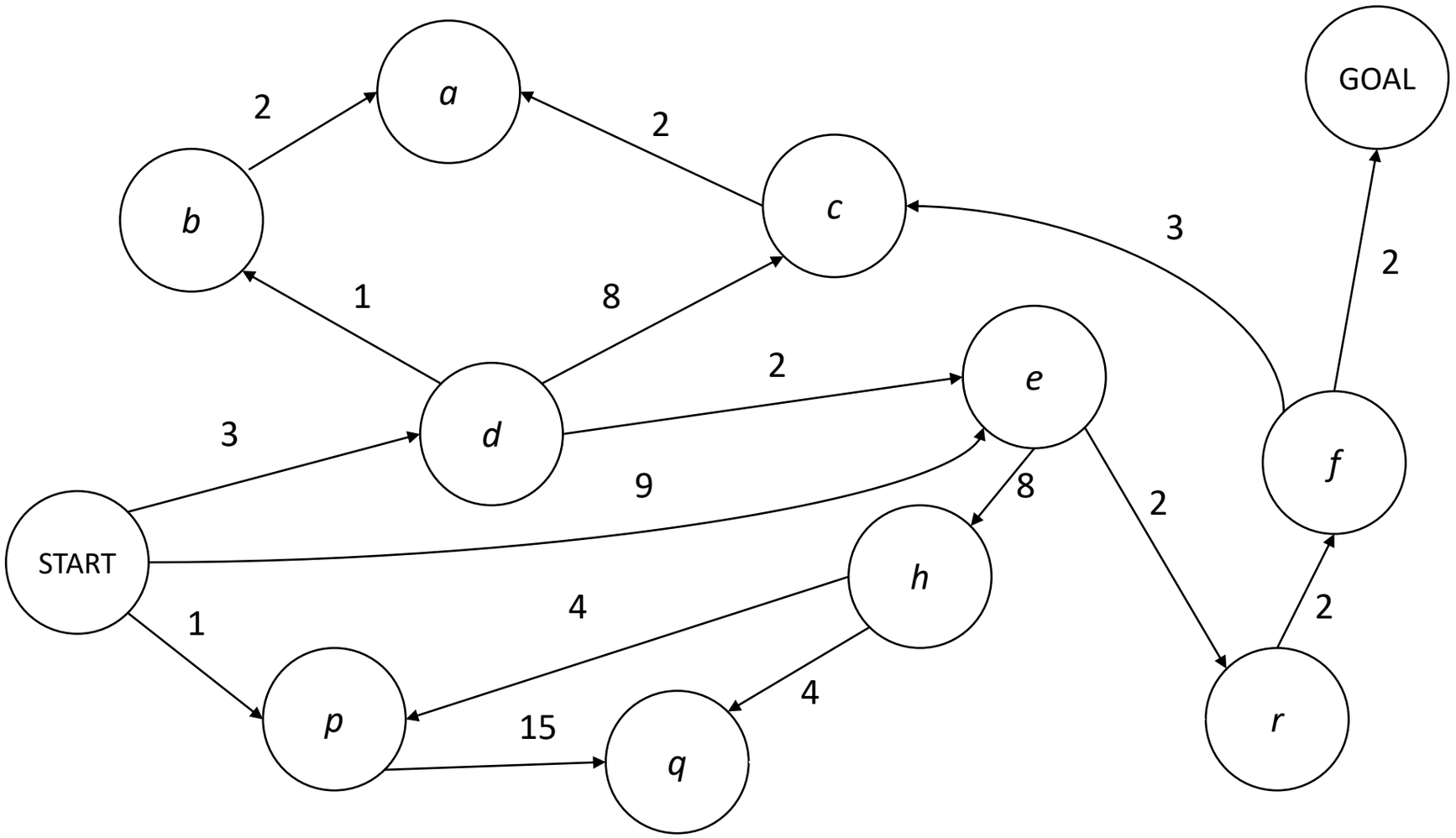


- ¿No es tremendamente redundante?
  - Por lo general, la mayor parte del trabajo se realiza en el nivel más bajo, así que no importa tanto generar los niveles superiores varias veces

Profundidad iterativa

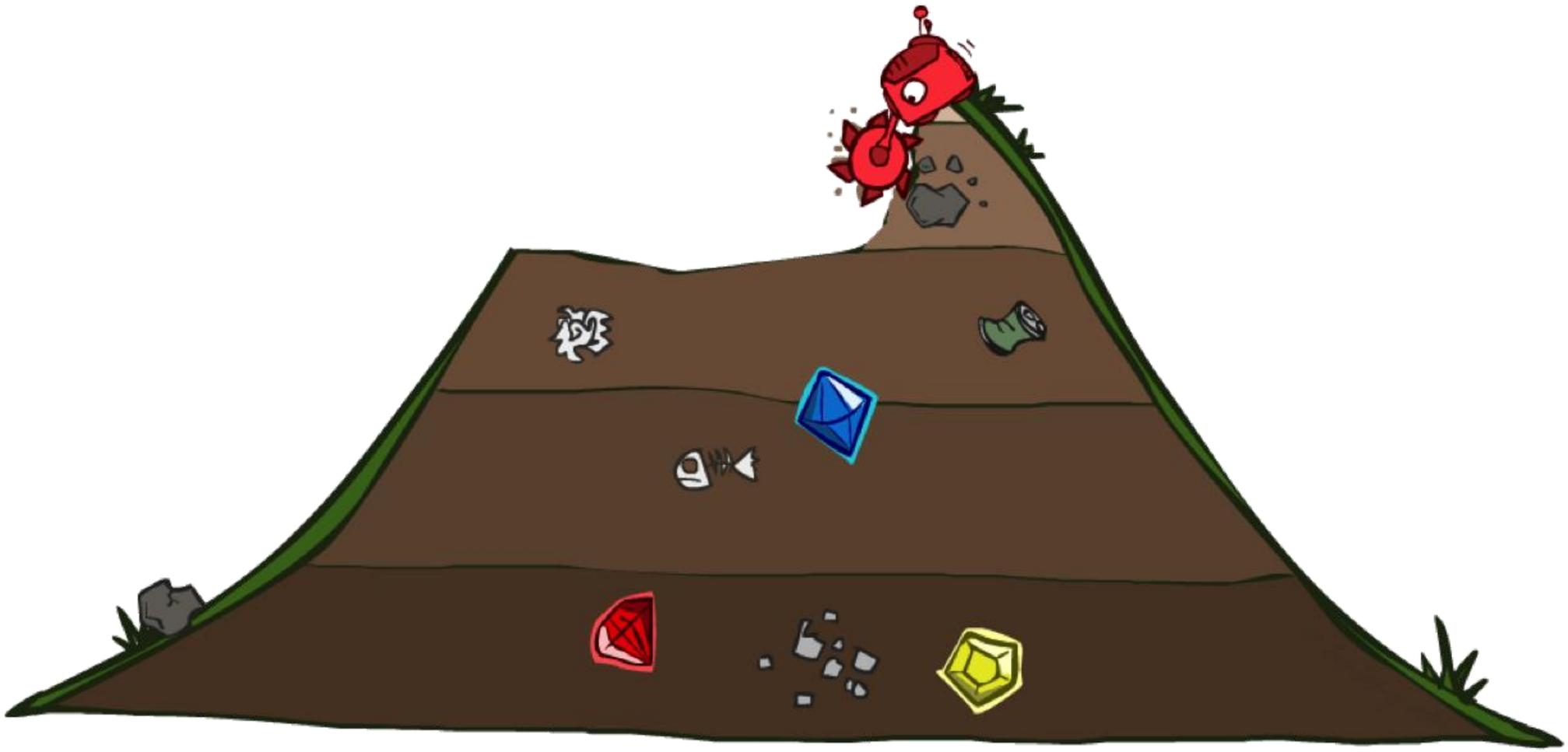
En general, la profundidad iterativa es el método de búsqueda no informada preferido cuando hay un espacio grande de búsqueda y no se conoce la profundidad de la solución

Búsqueda sensible al coste



BPA encuentra el camino más corto en términos de número de acciones.
No encuentra el camino de menor coste. Ahora veremos un algoritmo similar que sí encuentra el camino de menor coste.

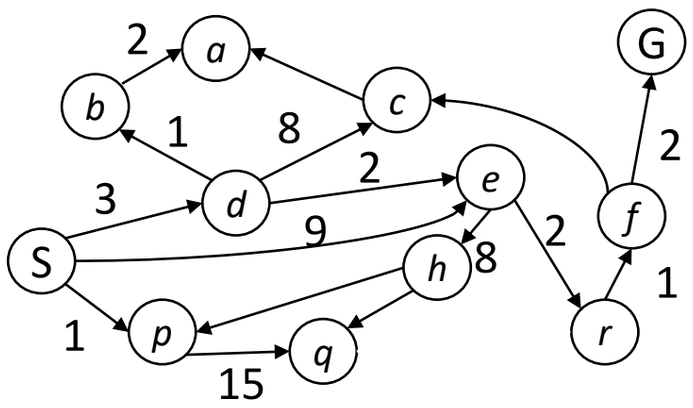
Búsqueda de coste uniforme



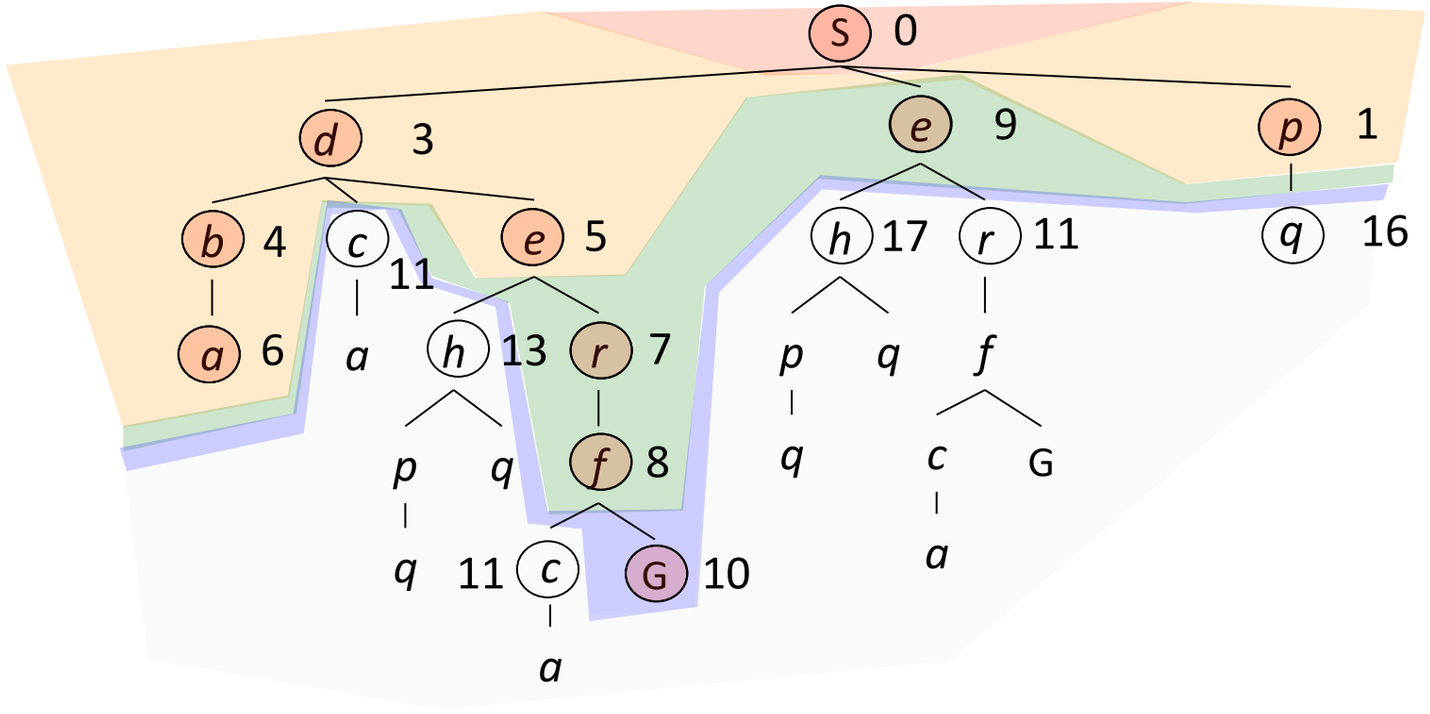
Búsqueda de coste uniforme

Estrategia: ampliar primero el nodo más barato:

La frontera es una cola de prioridades (prioridad: coste acumulado)

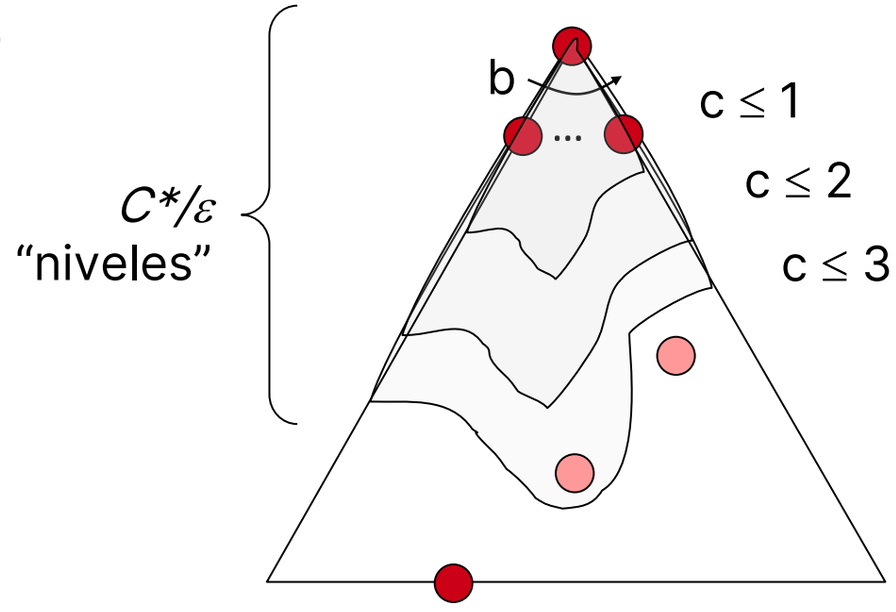


Contornos de coste



Propiedades coste uniforme (BCU/UCS)

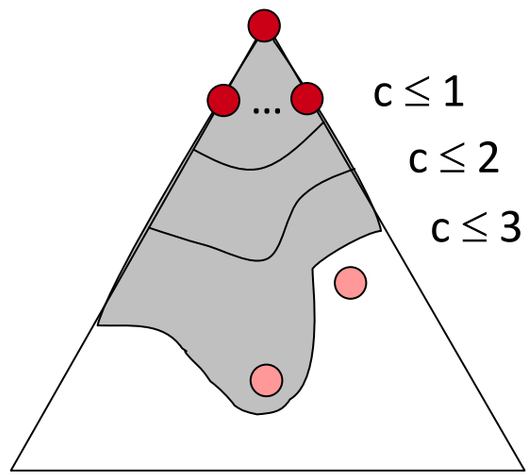
- ¿Qué nodos expande BCU?
  - Todos los nodos que cuestan menos que la solución más barata
  - Si esa solución cuesta C^* y las acciones cuestan al menos ε, entonces la "profundidad efectiva" es aprox. C^*/ε
  - Tarda $O(b^{C^*/\varepsilon})$ (exponencial en prof. efectiva)

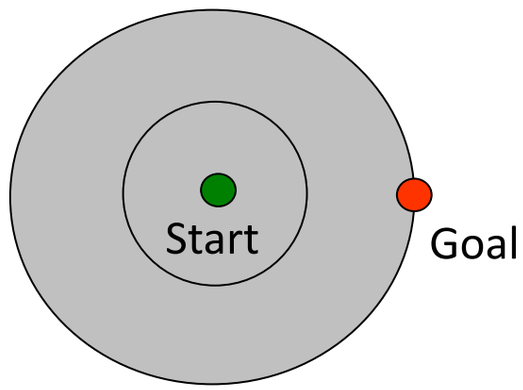


- ¿Cuánto ocupa la frontera?
  - Tiene aprox. el último nivel: $O(b^{C^*/\varepsilon})$
- ¿Es completa?
  - Asumiendo que la mejor solución tiene un coste finito y el coste mínimo de una acción es >0, sí
- ¿Es óptima?
  - ¡Sí! (lo demostraremos con el A*)

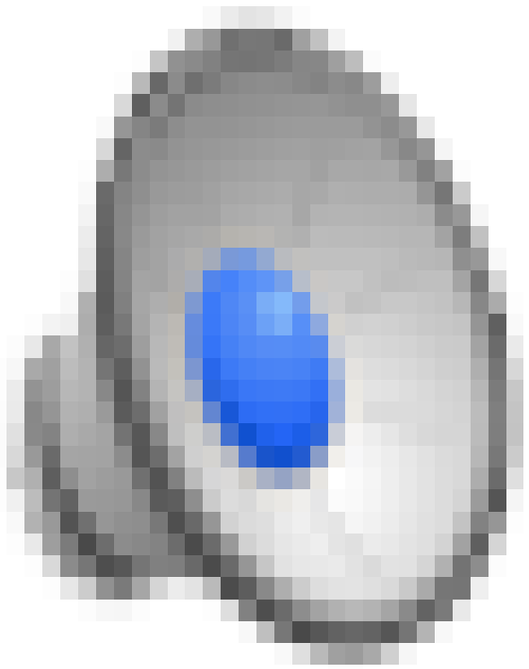
Problemas de la búsqueda de coste uniforme

- Recordatorio: la BCU explora contornos de coste incremental
- Lo bueno: es completa y óptima
- Lo malo:
  - Explora opciones en todas direcciones
  - No hay información sobre la posición del objetivo
- Lo arreglaremos pronto

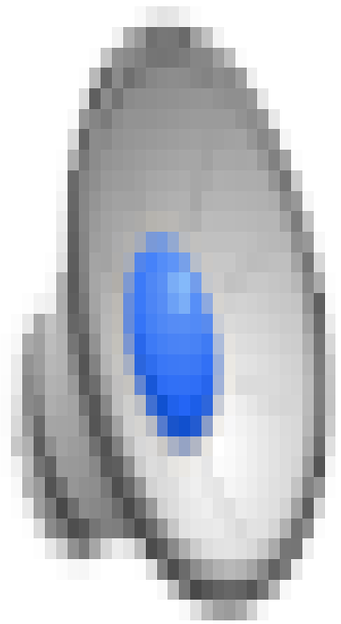




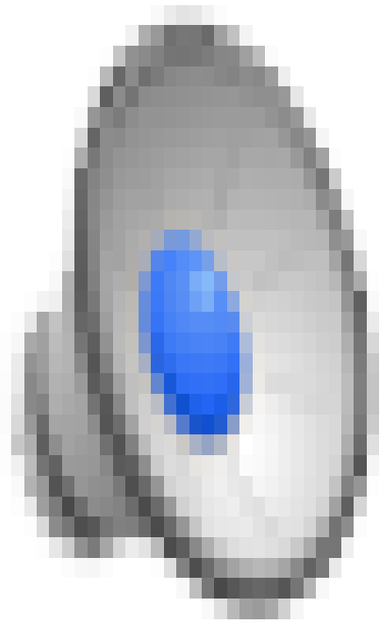
Demo Contornos BCU Pacman Laberinto S



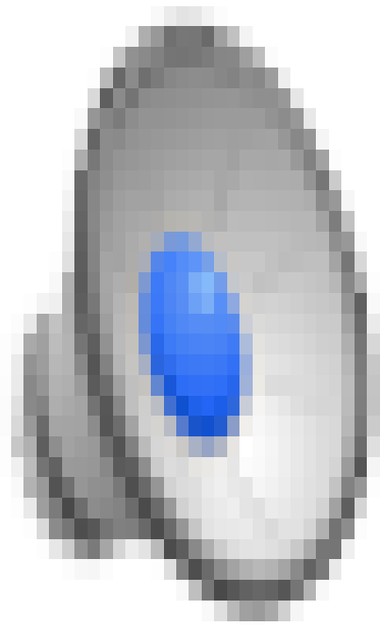
Demo BCU Vacío



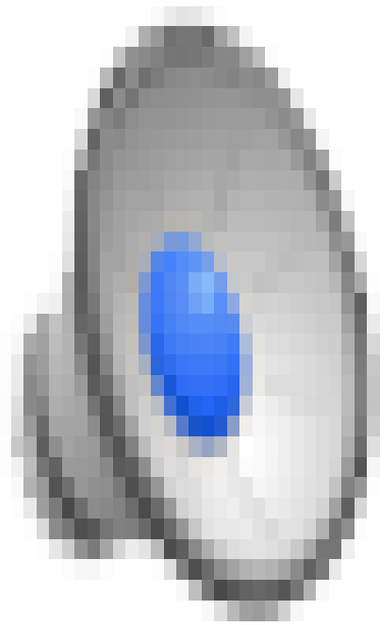
Demo Laberinto Agua Muy/Poco Profunda: BPP, BPA, o BCU



Demo Laberinto Agua Muy/Poco Profunda: BPP, BPA, o BCU



Demo Laberinto Agua Muy/Poco Profunda: BPP, BPA, o BCU



Resumen de propiedades

Criterio	Primero en anchura	Costo uniforme	Primero en profundidad	Profundidad iterativa
¿Completa?	Sí[a]	Sí[a, b]	No	Sí[a]
Tiempo	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^d)$
Espacio	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bd)$
¿Optimal?	Sí[c]	Sí	No	Sí[c]

[a] completa si b (factor de ramificación) es finito

[b] completa si los costes son ≥ ϵ para ϵ positivo

[c] óptima si los costes son iguales

Idea importante

- Todos estos algoritmos de búsqueda son iguales salvo en lo que respecta a las **estrategias de frontera**
  - La diferencia sólo está en cómo elegimos el siguiente nodo a expandir
  - Todas las fronteras pueden tratarse como colas de prioridades