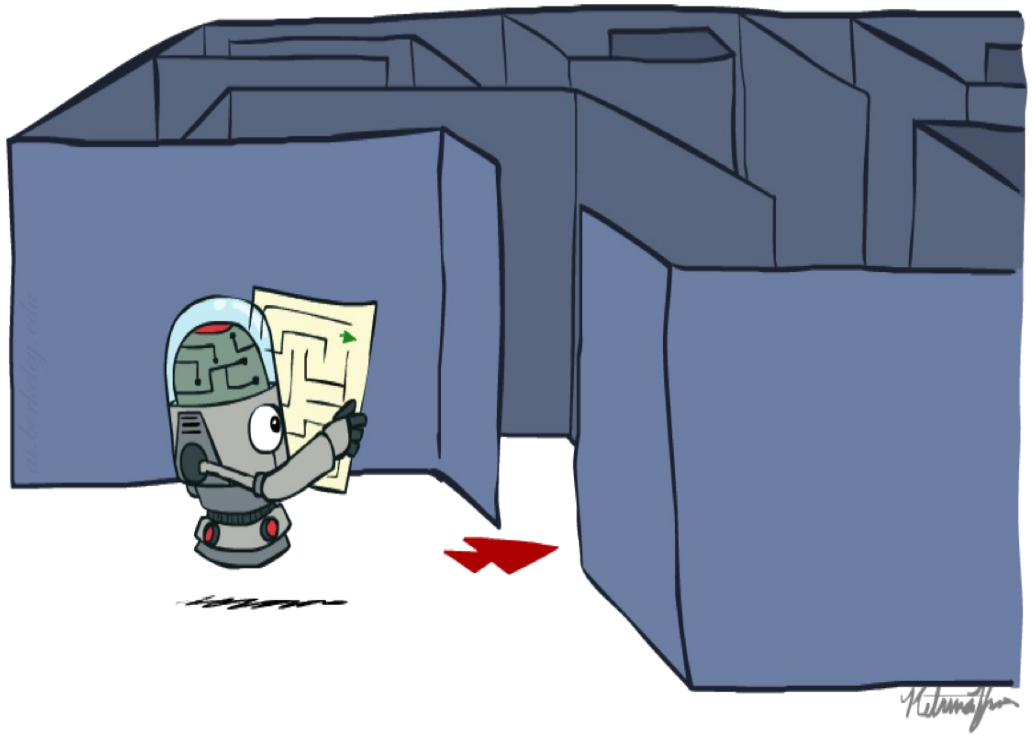
Técnicas de Inteligencia Artificial

Tema 3. Espacio de estados y búsqueda



Aitziber Atutxa

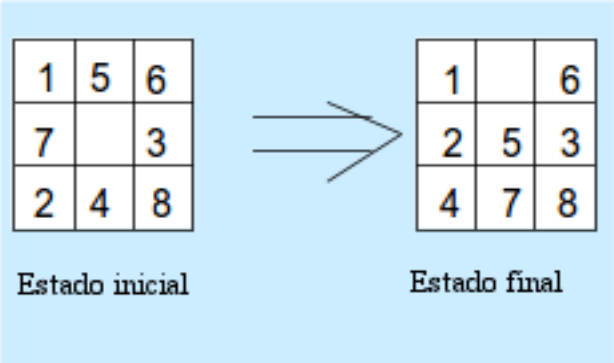
[transparencias de Koldo Gojenola y Ekaitz Jauregi adaptadas de Berkeley: Dan Klein, Pieter Abbeel]

Indice

- 3.1 Problemas y espacio de estados
- 3.2 Métodos de búsqueda no informados (Uninformed Search Methods):
  - Búsqueda en profundidad (Depth-First Search)
  - Búsqueda en anchura (Breadth-First Search)
  - Búsqueda de coste uniforme (Uniform-Cost Search)
- 3.3 Métodos de búsqueda informados:
  - Heurísticos
  - Búsqueda voraz (Greedy Search)
  - Búsqueda A* (A* or A star search)
- 3.4 Búsqueda adversarial
  - Minimax
  - Poda alfa-beta
  - Expectimax

Problemas y espacio de estados

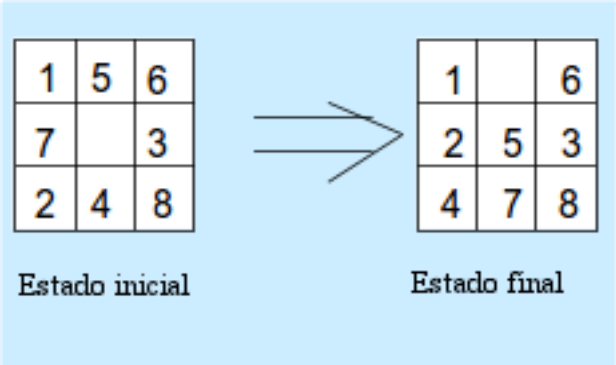
- Generalmente: problemas de IA, se modelan:
  - de forma simbólica y discreta
  - definiendo todas las configuraciones posibles (abstracción).
- Planteamiento del ***problema:***
  - encontrar una ***configuración objetivo***
  - partiendo de la ***configuración inicial***
  - aplicando ***acciones/transformaciones válidas*** según el modelo
  - La ***respuesta*** es la secuencia de acciones cuya aplicación sucesiva lleva al objetivo con el menor ***coste*** posible.



Problemas y espacio de estados

Modelo (Ejemplo típico: Juegos de tablero):

- Los estados (configuraciones posibles) = las configuraciones del tablero representado en forma simbólica.
- Las acciones permitidas, función de Sucesión: movimientos del juego, formalizadas como ***transiciones de estado, acción → nueva configuración (estado sucesor)***.
- Estado inicial del juego
- Función de Estado(s) finales



Espacio de estados: estados y operadores

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

Espacio de estados: estados y operadores

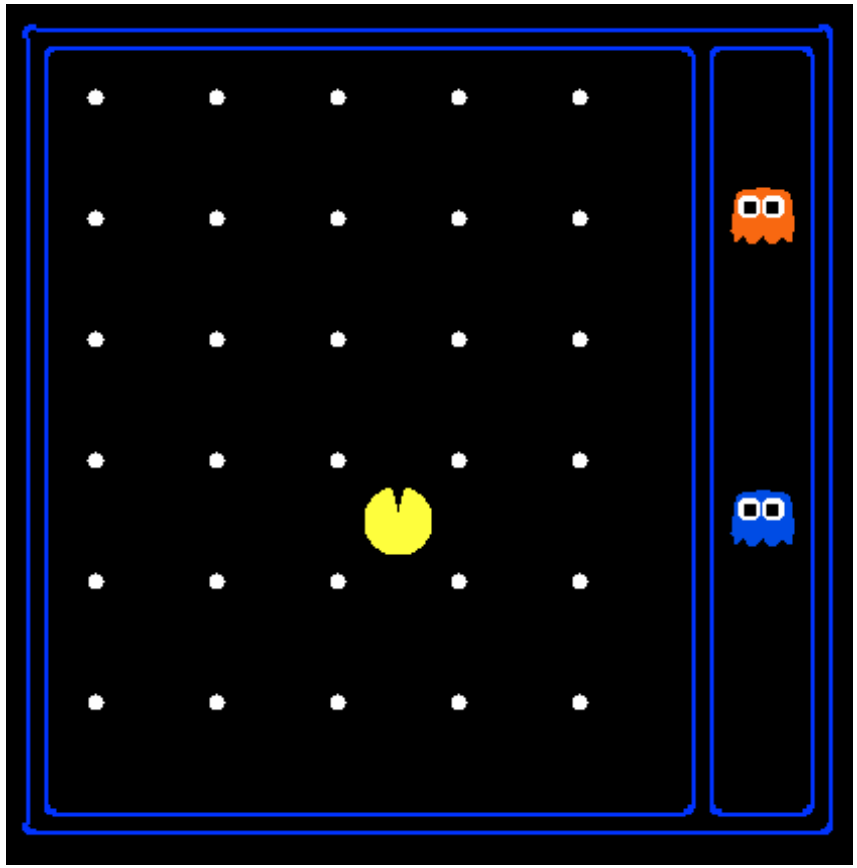
- **Una vez modelado** el problema, se puede modelar también su complejidad:
  - **Complejidad del espacio de estados:** aplicando la abstracción y sabiendo qué información es relevante para representar en el estado, ¿cuántas configuraciones posibles hay?
  - **Complejidad de los arboles de busqueda:** Recordad que en un árbol un mismo estado puede aparecer varias veces. **Factor de ramificación:** Es decir, ¿cuantas posibles acciones como máximo se va a poder llevar a cabo a partir de un estado?

Tamaño del espacio de estados

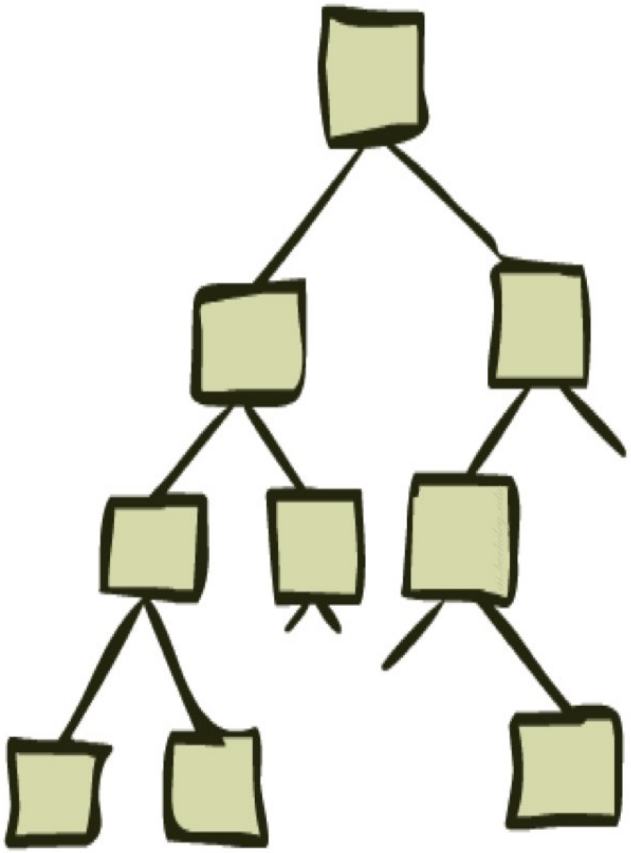
- Estados del mundo:
  - Posiciones del agente: 120
  - Número de comidas: 30
  - Posiciones de fantasma: 12

- ¿Cuántos
  - Estados del mundo?
    - $120x2^{30}x12^{2}(x4)_{\text{la posición de la boca/rotación del pacman}}$
  - Estados para buscar camino?

    $120(x4)_{\text{la posición de la boca/rotación del pacman}}$
  - Estados para eat-all-dots?
    - $120x2^{30}(x4)_{\text{la posición de la boca/rotación del pacman}}$

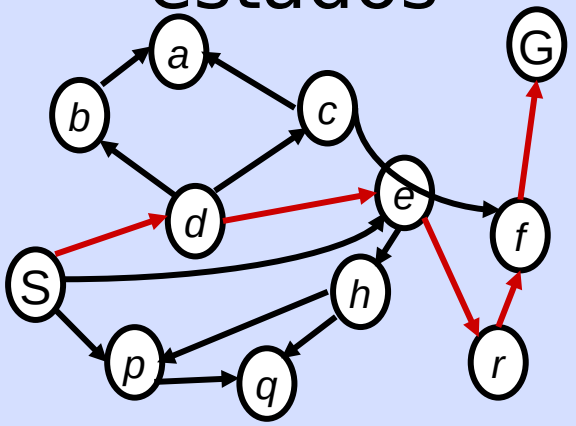


Grafos de espacio de estados y Árboles de búsqueda



Grafos de espacio de estados vs. Árboles de búsqueda

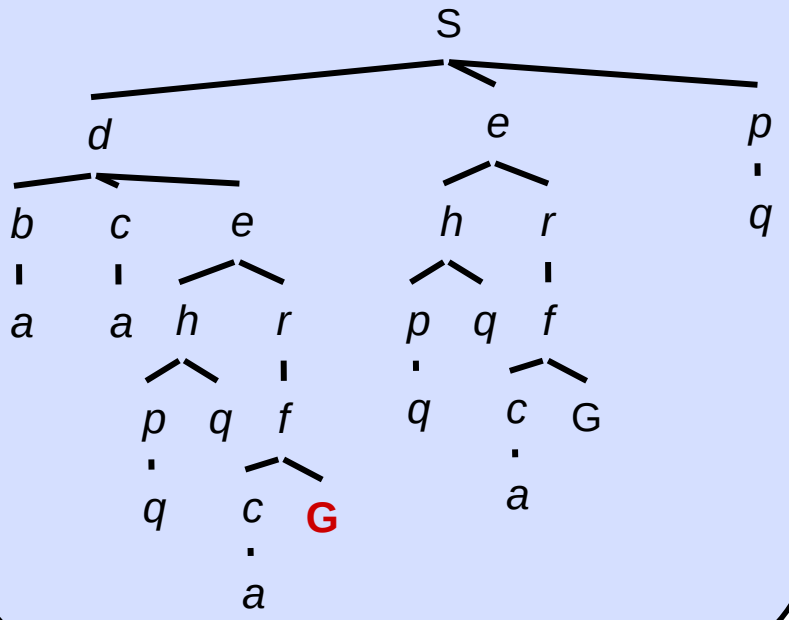
Grafo de espacio de estados



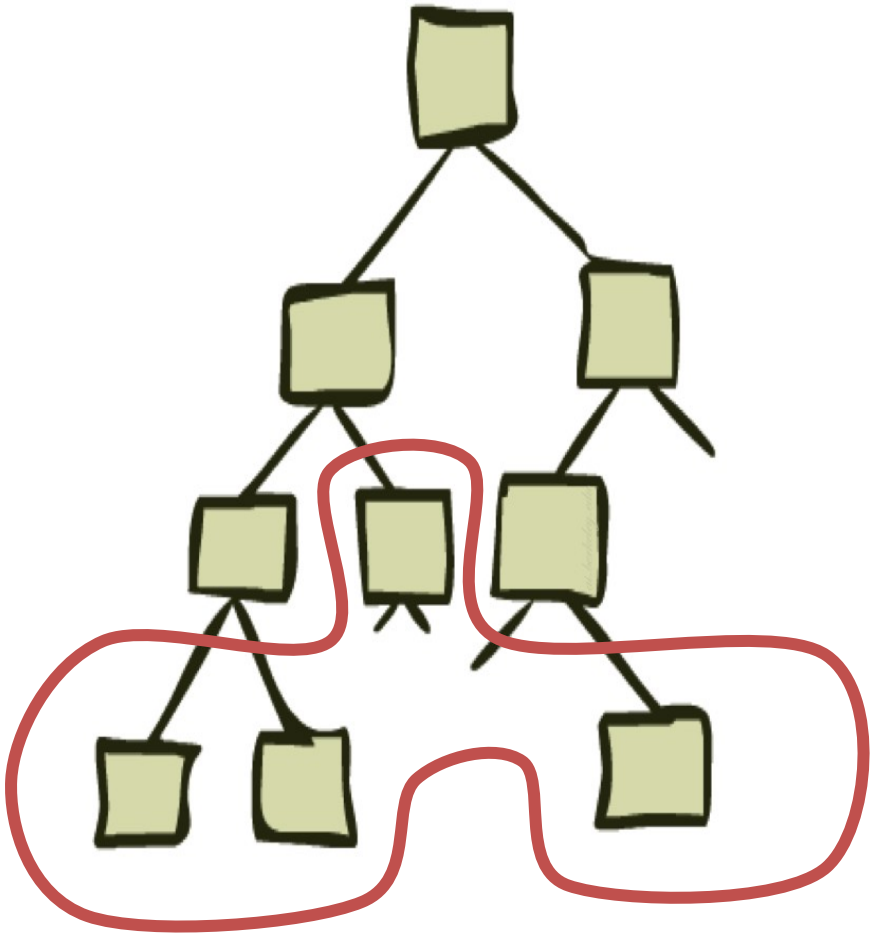
Cada nodo final en el árbol de búsqueda es un camino entero en el grafo

Se construyen a demanda – construyendo lo mínimo posible

Árbol de búsqueda



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)
  - Búsqueda de coste uniforme (Uniform-Cost Search)

- Métodos de búsqueda informados:
  - Heurísticos
  - Búsqueda voraz (Greedy Search)
  - Búsqueda A* (A* or A star search)

Búsqueda en árbol General

```
RECORRIDO NO SISTEMÁTICO(u)
    MIENTRAS haya elementos accesibles (elementos a expandir-frntera)
        nodoAct = obtener el siguiente nodo al azar de entre los accesibles
        SI nodoAct es meta:
            devolver el camino
        – SINO:
            FOR (nodoSuc, dir) ∈ sucesores DO
                introducir nodoSuc en el conjunto de los accesibles
```

- Supongamos que busco a una persona en este edificio y decido recorrerlo al azar… ¿Es el azar un buen sistema para hacer una busqueda o conviene ser sistemático?

- Ideas importantes:
  - Frontera ¿qué nodos me son accesibles? *fringe-elementos accesibles/candidatos a expansión. Recordemos que cada nodo me abre la puerta a otros nodos que serán a partir de entonces accesibles)*
  - Expansión *¿qué nodo transito posibilitandome el acceso a sus hijos?*
  - Estrategia de exploración *¿qué criterio empleo para decidir cúal es el siguiente nodo a expandir? Se une con el siguiente concepto o pregunta*
- Pregunta principal: ¿qué nodos de la frontera exploramos?

Búsqueda en árbol General

MIENTRAS haya elementos en la frontera (elementos a expandir)

nodoAct = obtener el siguiente nodo de la frontera

SI nodoAct no ha sido visitado:

SI nodoAct es meta:

devolver el camino

* SINO:

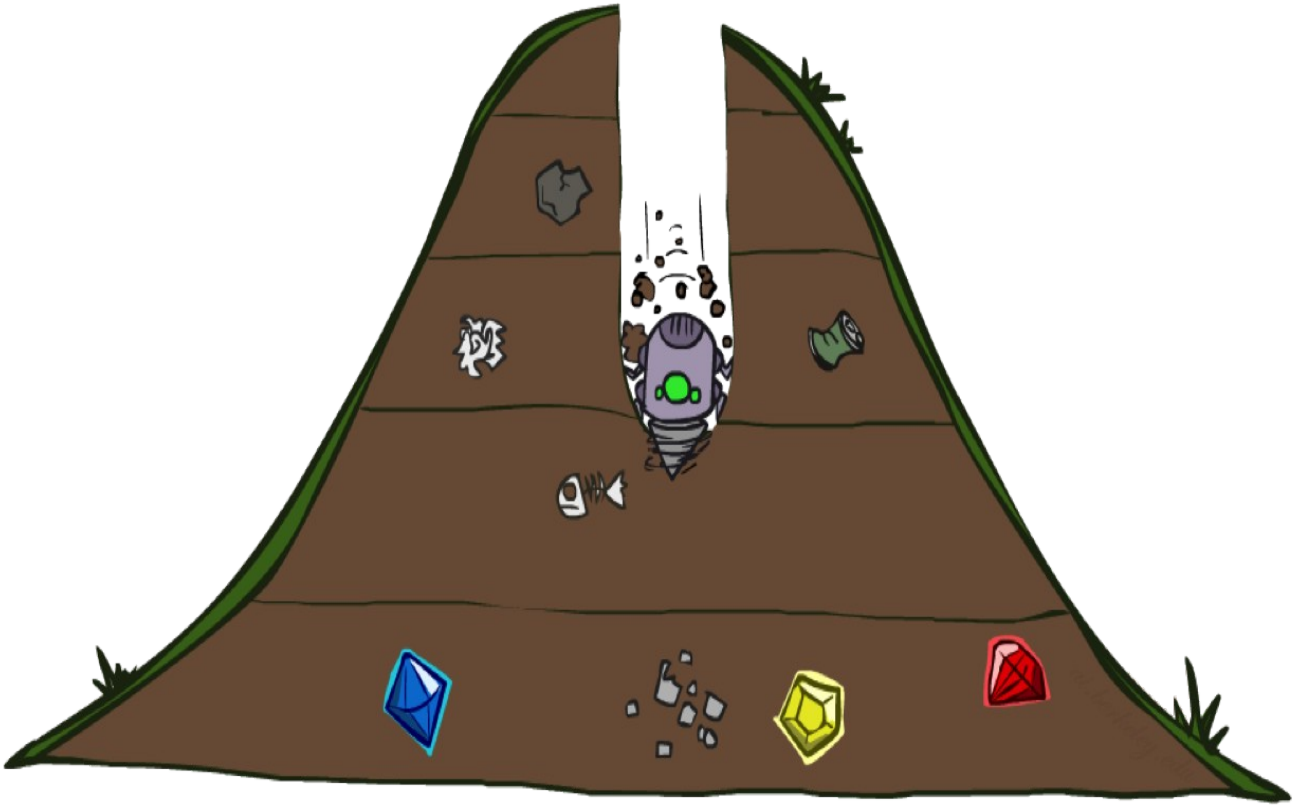
FOR $(nodoSuc, dir) \in sucesores$ DO

introducir nodoSuc en la frontera

EXPANSION DEL NODO

- Ideas importantes:
  - Frontera (fringe-candidatos a expansión)
  - Expansión
  - Estrategia de exploración
- Pregunta principal: ¿qué nodos de la frontera exploramos?

Búsqueda en profundidad (Depth-First Search)



Búsqueda en profundidad

- **Algoritmo: Una estrategia posible que me hace ser sistemático**

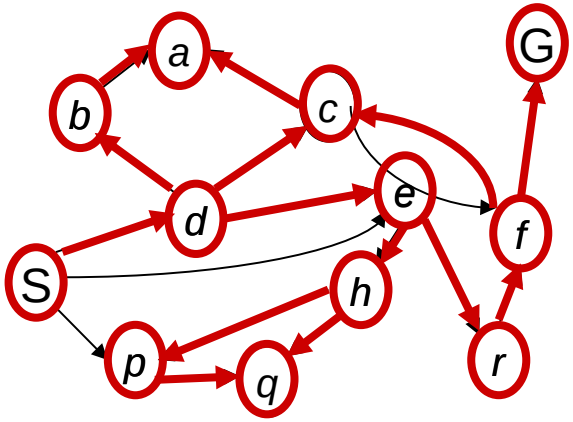
  "Los últimos serán los primeros…" expandir siempre el último nodo que se ha puesto accesible,

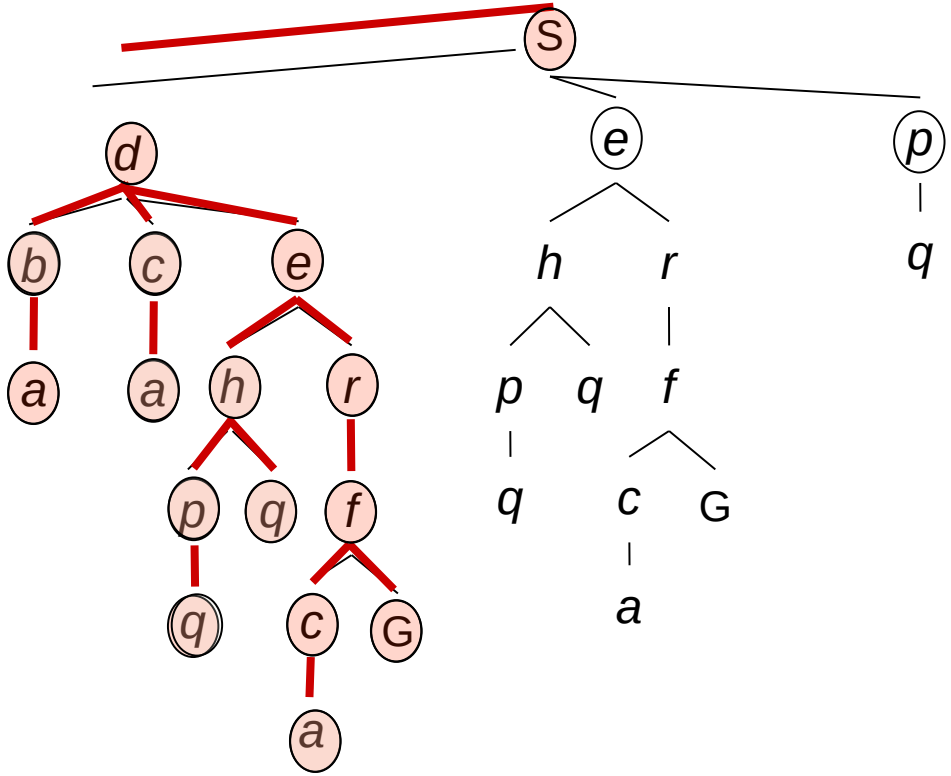
```
DFS(u)
    MIENTRAS haya elementos en la frontera (elementos a expandir)
        nodoAct = obtener el siguiente nodo de la frontera
        SI nodoAct no ha sido visitado:
            SI nodoAct es meta:
                devolver el camino
            * SINO:
                FOR (nodoSuc, dir) ∈ sucesores DO
                introducir nodoSuc en la frontera
```

- **Problema**
  - Caer en un camino infinitamente largo. **Atentos a las letras en negrita….**

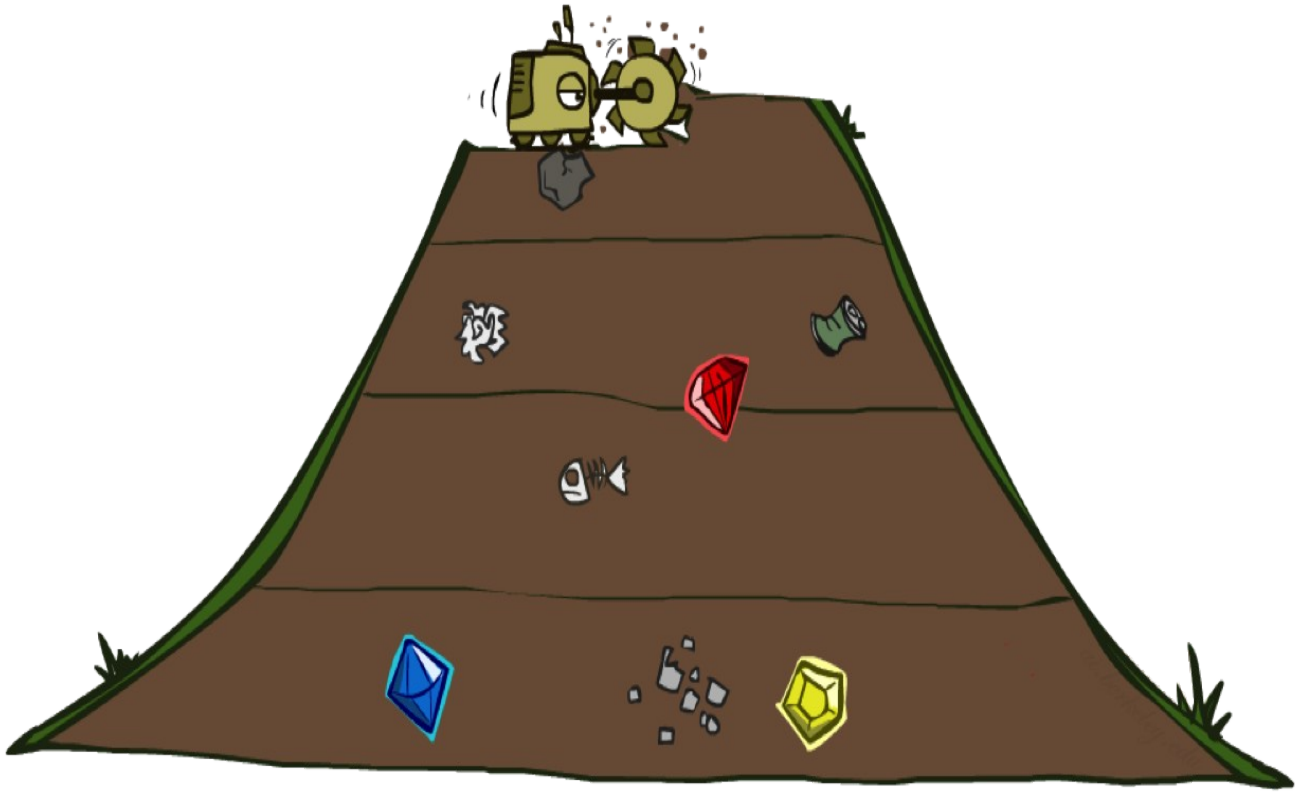
Búsqueda en profundidad

Estrategia: expandir un nodo más profundo primero
Implementación: **Fringe es una pila LIFO** *(es decir, los sucesores se ponen delante)*





Búsqueda en anchura (Breadth-First Search)



Búsqueda en anchura

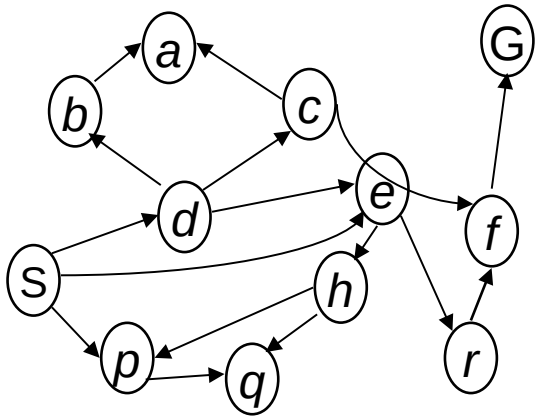
- **Algoritmo:** ¿Y si recorro en orden de llegada? *"los primeros serán los primeros"*

```
BFS(u)
    MIENTRAS haya elementos en la frontera (elementos a expandir)
        nodoAct = obtener el siguiente nodo de la frontera
        SI nodoAct no ha sido visitado:
            SI nodoAct es meta:
                devolver el camino
            * SINO:
                FOR (nodoSuc, dir) ∈ sucesores DO
                introducir nodoSuc en la frontera
```

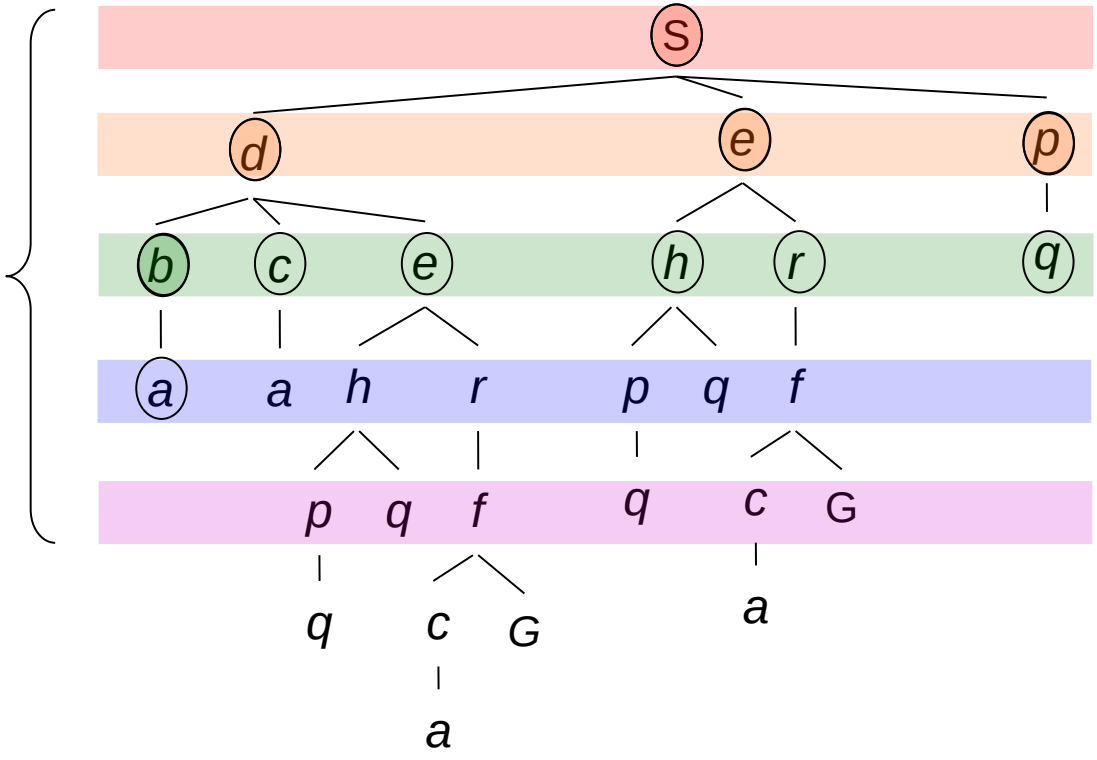
- **Problema**
  - Caer en un camino infinitamente largo

Busqueda en Anchura

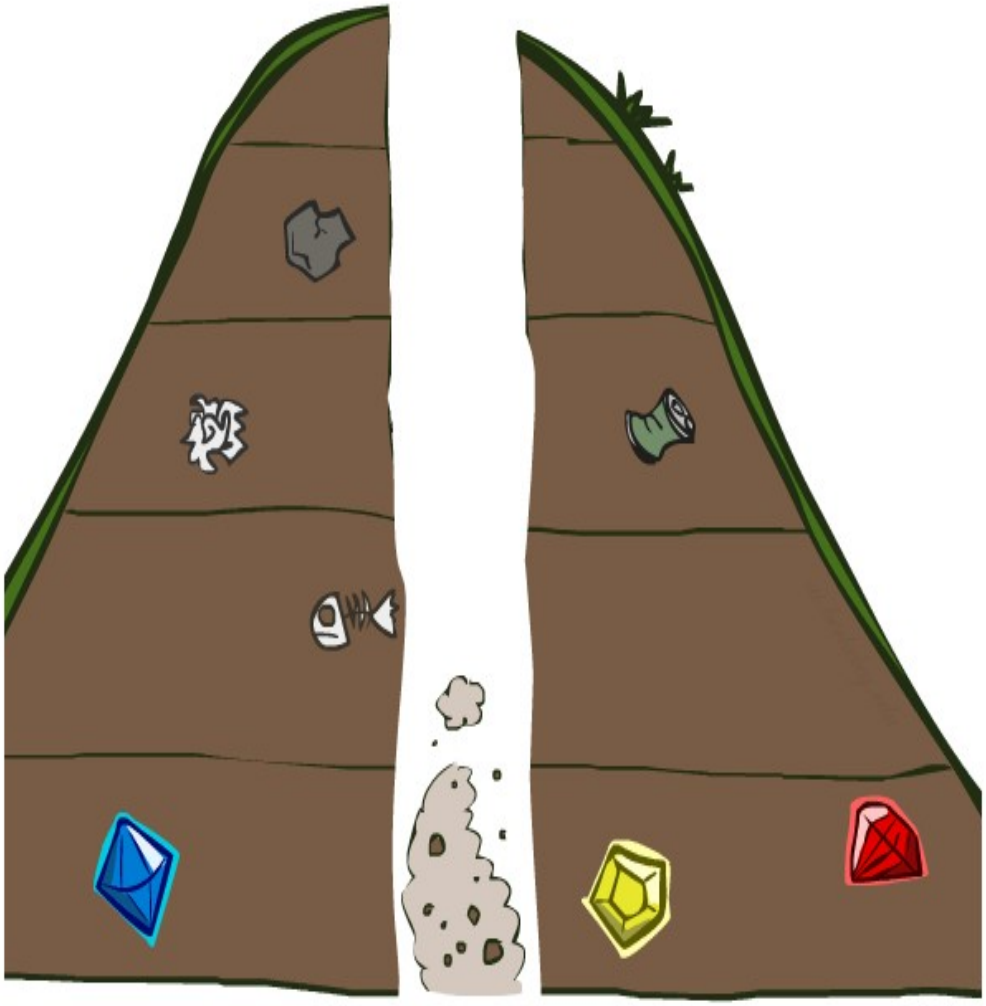
Estrategia: expandir primero un nodo del nivel más bajo
Implementación: el borde (fringe) es una cola FIFO



Niveles de búsqueda

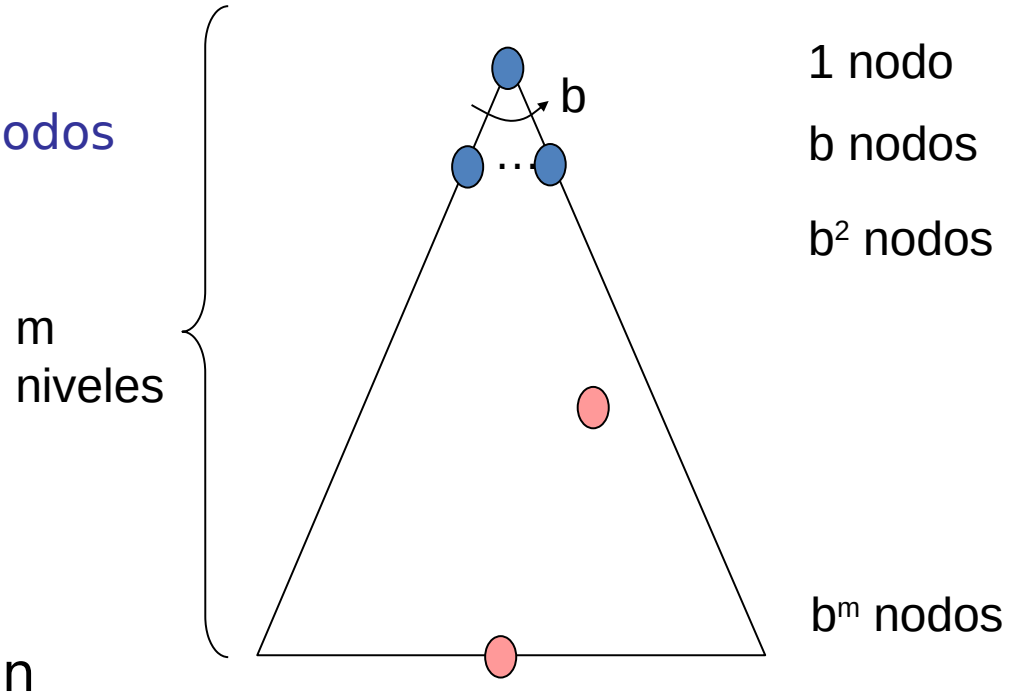


Propiedades de algoritmos de búsqueda



Propiedades de algoritmos de búsqueda

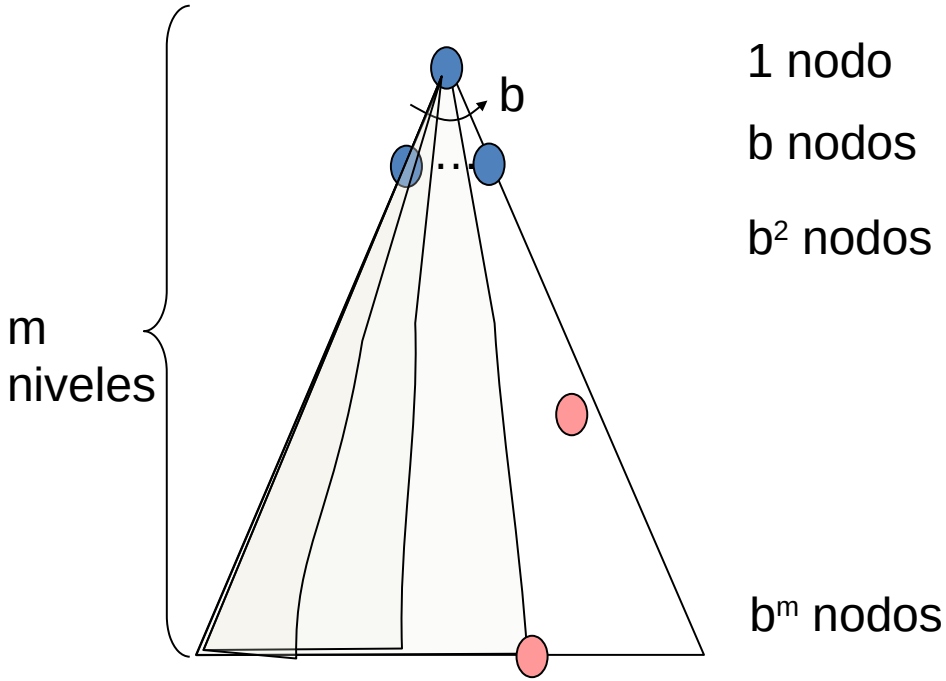
- Completo: ¿garantiza encontrar una solución en caso de que haya una?
- Óptimo: ¿garantiza encontrar el camino de coste mínimo?
- ¿Complejidad en tiempo de ejecución?
  ¿En el peor de los casos cuántos nodos habrá que recorrer?
- ¿Complejidad en espacio de memoria requerido?
  ¿La pila por cada nivel cuántos nodos guarda?



- Árbol de búsqueda:
  - b es el factor de ramificación
  - m la profundidad máxima
  - Soluciones a distintos niveles
- ¿Número de nodos del árbol?
  - $1 + b + b^2 + ... b^m = O(b^m)$ RESPUESTAS A CONTINUACIÓN

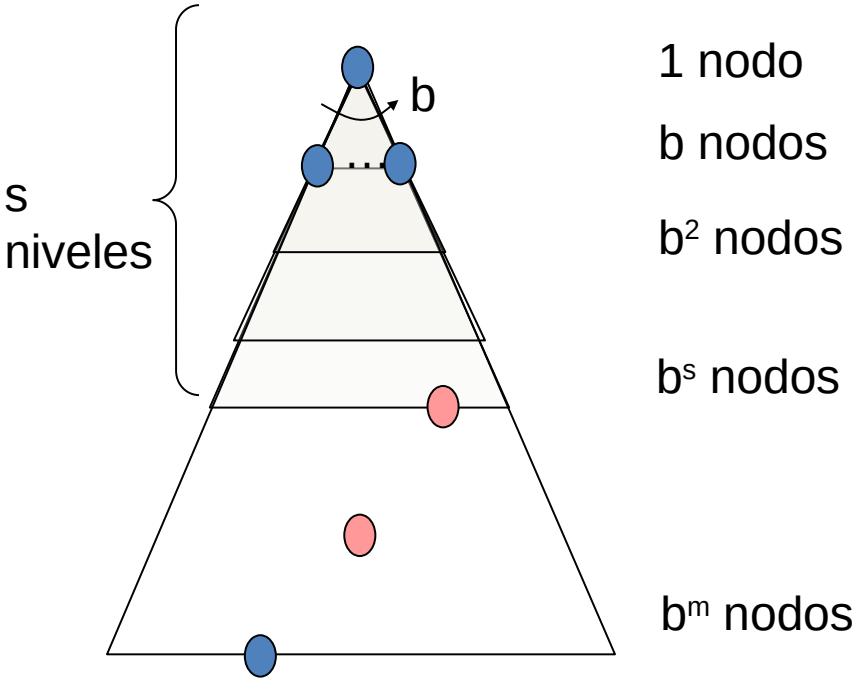
Propiedades de la Busqueda en Profundidad (DFS)

- ¿Qué nodos expande DFS?
  - Un prefijo izquierdo del árbol
  - ¡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(bm)
- ¿Es completo?
  - m puede ser infinito, por lo que solo si prevenimos ciclos (más después)
- ¿Es óptimo?
  - No, encuentra la solución “más a la izquierda”, sin tener en cuenta la profundidad o coste

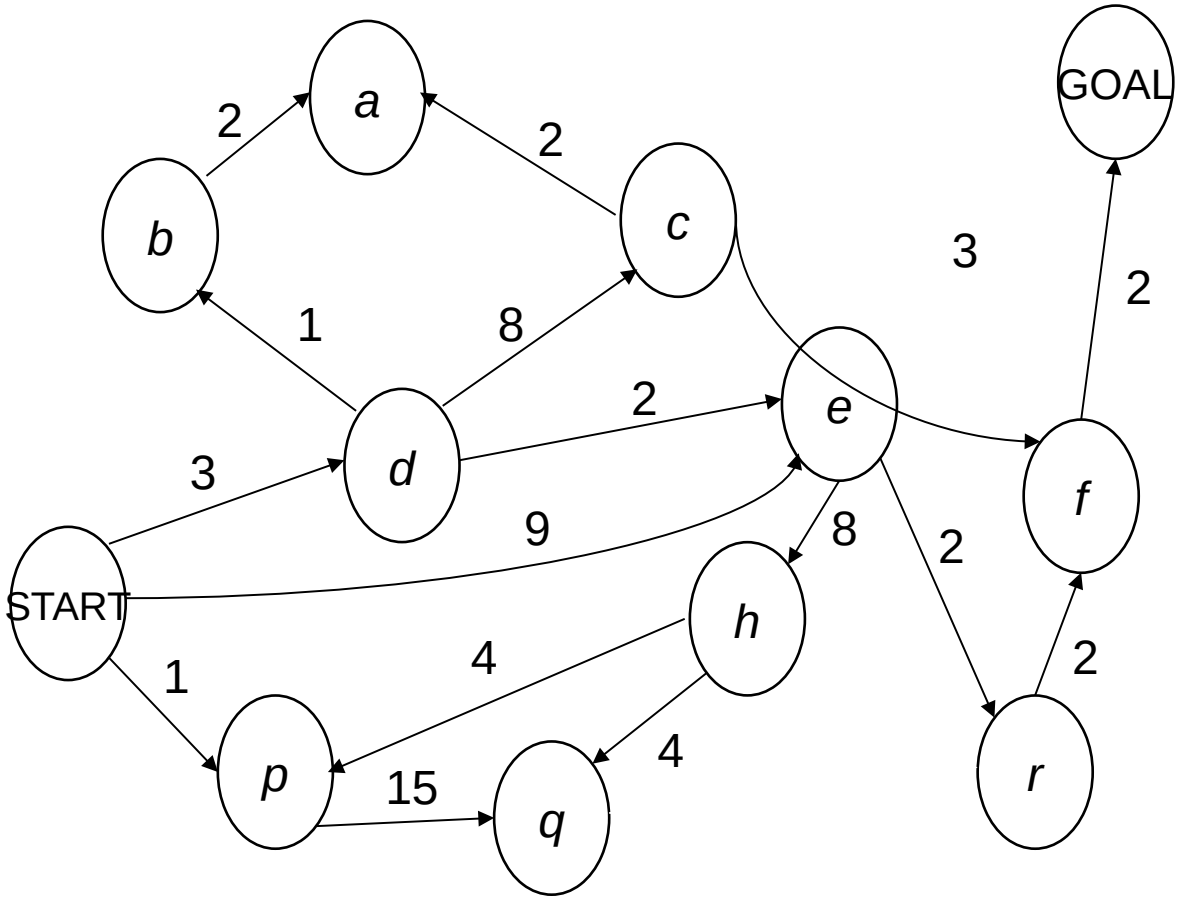


Propiedades de la Busqueda en Anchura(BFS)

- ¿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)

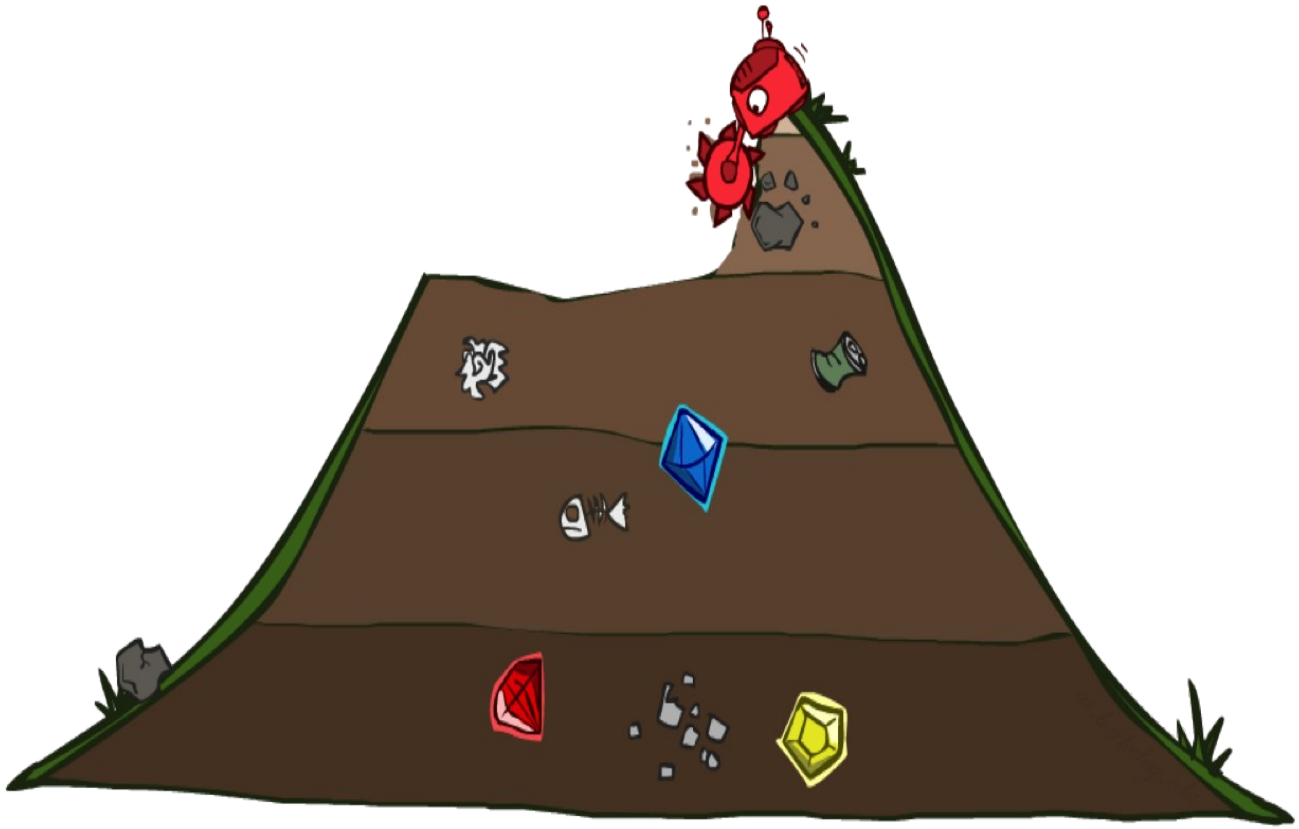


Búsqueda sensitiva al coste (Cost-Sensitive Search)



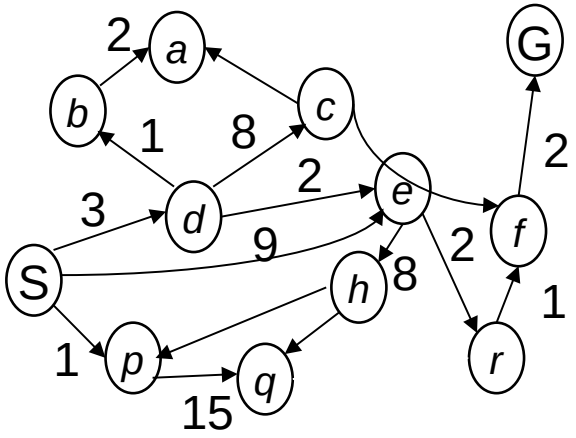
BFS encuentra el camino más corto en función del número de acciones (pasos), pero no encuentra el camino de coste mínimo. Examinaremos un algoritmo similar que encuentra el camino de coste mínimo.

Búsqueda de coste uniforme (Uniform Cost Search, UCS)

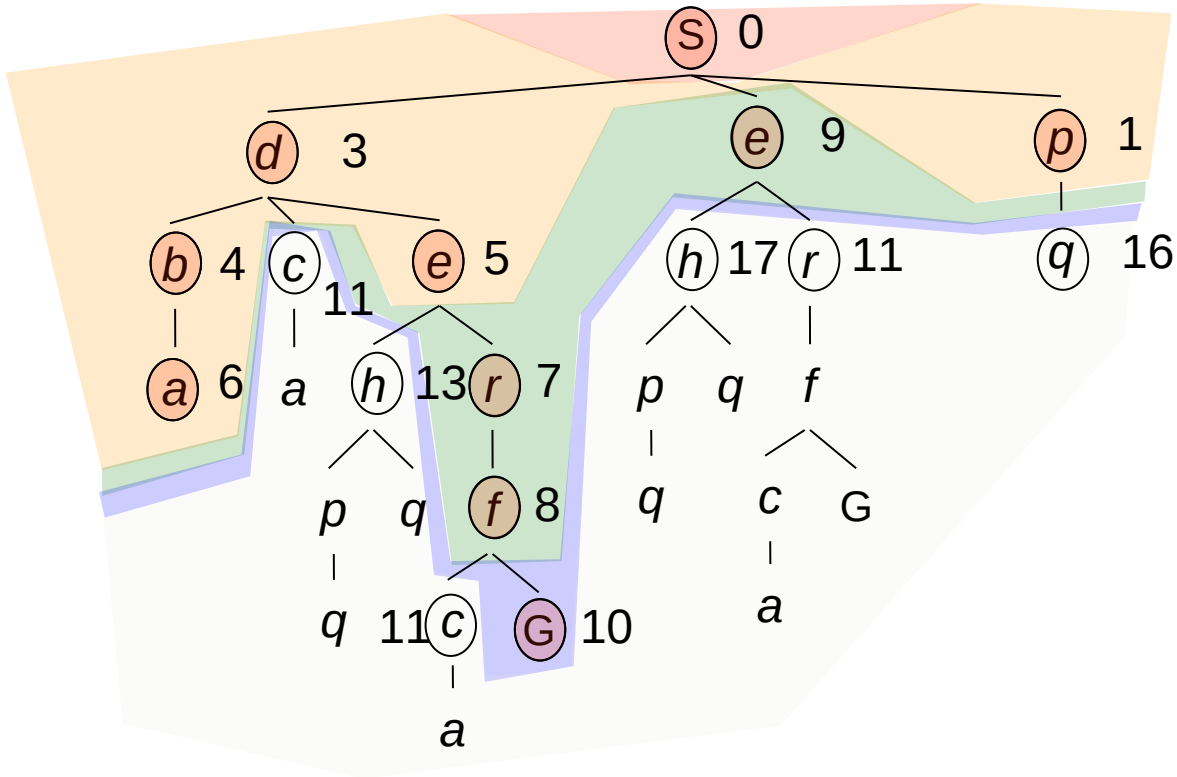


Uniform Cost Search

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



Costes (contornos)



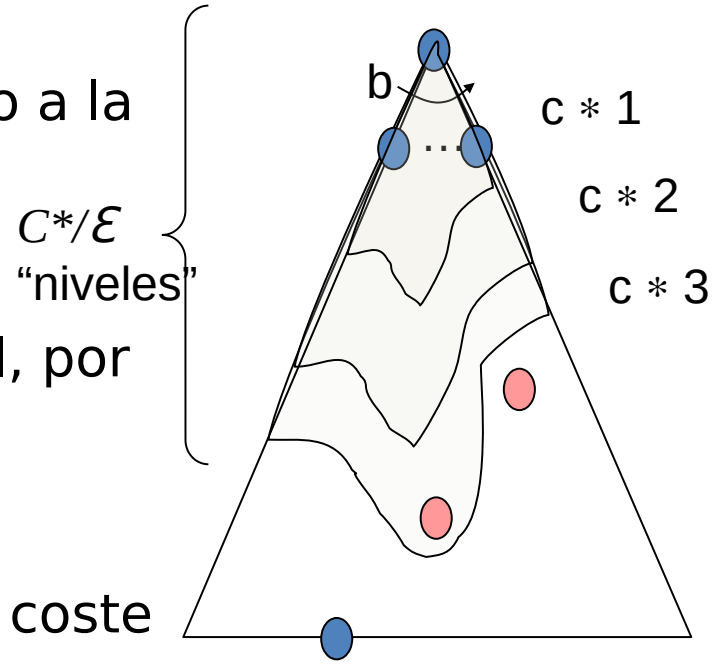
Propiedades de Uniform Cost Search (UCS)

- ¿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 $O(b^{C^*/\varepsilon})$ (exponencial respecto a la profundidad efectiva)

- ¿Cuánto espacio toma el borde (fringe)?
  - Contiene aproximadamente el último nivel, por ello $O(b^{C^*/\varepsilon})$

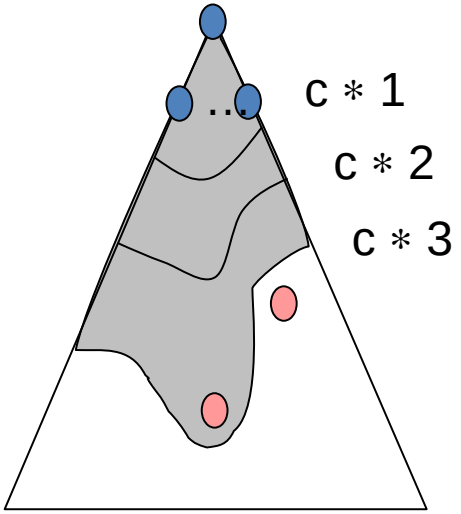
- ¿Es completo?
  - Asumiendo que la mejor solución tiene un coste finito y que el coste mínimo de un arco es positivo, sí

- ¿Es óptimo?
  - ¡Sí! (la prueba más tarde, con el algoritmo A*)



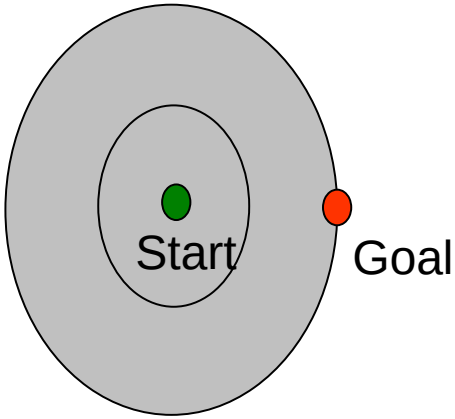
Aspectos de Uniform Cost

- A recordar: UCS explora contornos de coste de manera incremental



- La parte buena: UCS es completo y óptimo

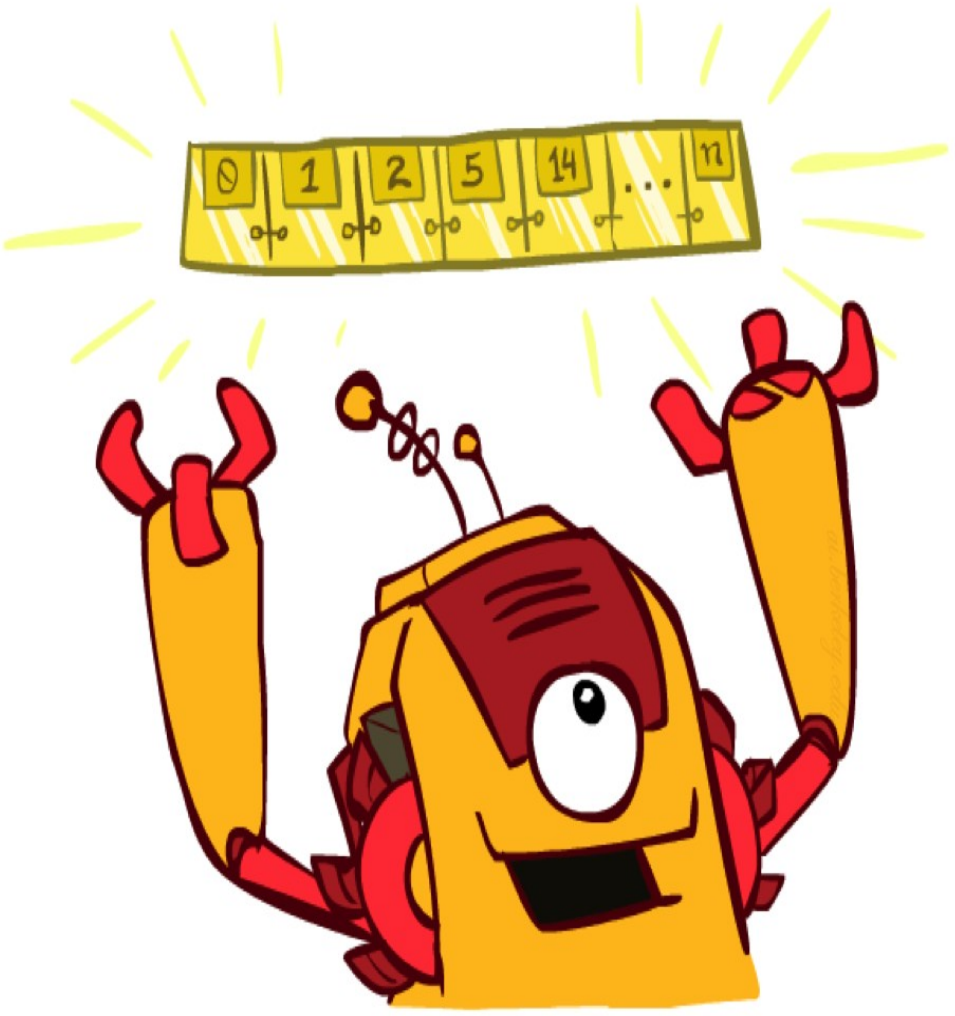
- Lo malo:
  - Explora opciones en cualquier “dirección”
  - No hay información acerca de la posición del objetivo



- Se arreglará con la búsqueda heurística

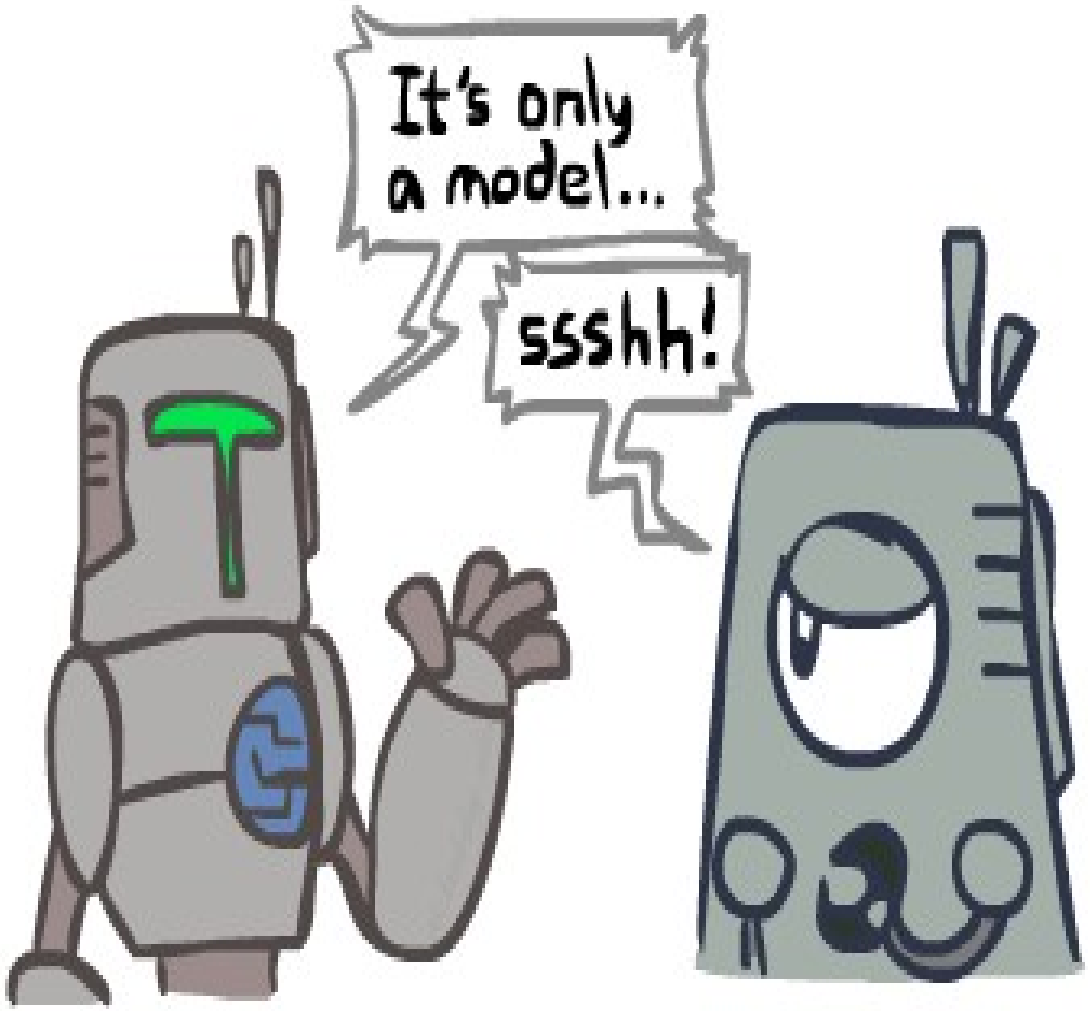
Un solo algoritmo + est.datos

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



Búsqueda y Modelos

- La búsqueda opera sobre modelos del mundo
  - El agente no prueba todos los planes en el mundo real
  - La planificación se hace en modo “simulación”
  - Nuestra búsqueda es tan buena como nuestros modelos…



Búsqueda No Informada: 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.