Bloque 2: Resolución de Problemas en IA. Búsqueda.

Tema 2. Resolución de problemas en IA.
Búsqueda de soluciones en un espacio de estados. Formulación y tipos.
Búsqueda no informada. Costes

Tema 3. Búsqueda heurística. Algoritmo A, algoritmo A*. Propiedades.

Tema 4. Diseño y evaluación de funciones heurísticas.
Coste computacional de la resolución de problemas.

Tema 5. Búsqueda entre adversarios. Algoritmo Minimax, Alfa-Beta.

Bibliografía Básica:

- Rusell, Norvig. ***'Inteligencia Artificial: Un enfoque moderno'***. Prentice Hall, 2004. Cap. 3, 4 y 5.
- Nilsson. ***"Inteligencia Artificial: una nueva síntesis"***. McGraw Hill, 2000. Cap. 7-9
- Palma, Marín. ***"IA. Técnicas, métodos y aplicaciones"***. McGraw Hill, 2008. Cap. 8-9

Bibliografía Complementaria:

E. Rich, K. Knight. ***'Inteligencia Artificial'***. McGraw Hill , 1994, Capítulo 2.
P.H. Winston. ***'Inteligencia Artificial'***. Addison-Wesley Iberomaericana, 1994.
M. Ginsberg ***"Essentials of Artificial Intelligence"***. Morgan Kaufmann Publishers, 1993.

Tema2.- Resolución de problemas en IA. Formulación.

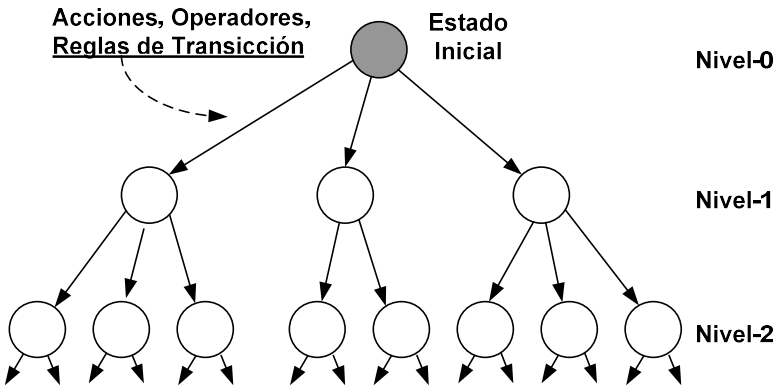
La Resolución de Problemas en IA, se modela generalmente como un proceso de **búsqueda** a través de un **espacio de estados**.

Definición del problema:

a) Definir una representación para los **posibles estados** del problema.
b) Determinación ***estado inicial***.
c) Determinación estado ***final meta***, explícitamente o condiciones de estado meta.
d) **Operadores**, reglas de transición o de movimiento: Permiten transformar un estado del problema en otro.

Objetivo: Encontrar una secuencia de reglas (solución) que aplicadas a una descripción del problema (estado inicial) lo transformen en otra (estado meta), mediante un proceso de búsqueda en un espacio de estados.

Espacio de estados: Conjunto de estados del problema alcanzables a partir del estado inicial, aplicando reglas de transición.



Estado Inicial

Estado inicial del problema

Conjunto de acciones, operadores, reglas de transición:

- Descripción del estado (o estados) que se alcanza, desde un estado concreto, al ejecutar la acción.
- Cada acción tiene un coste > 0

Estado Meta, o Prueba de objetivo:
Descripción de estado para saber si es un estado meta u objetivo

Espacio de Estados = Estado Inicial + Conjunto Operadores

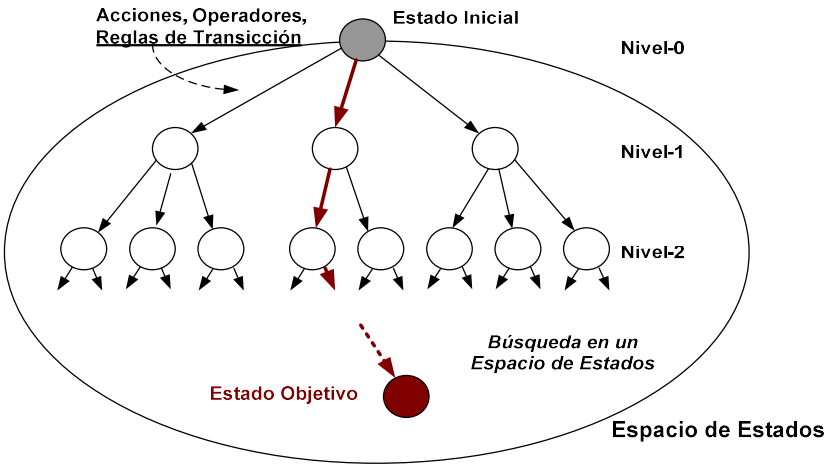
Conjunto de todos los estados que pueden alcanzarse desde el estado inicial mediante cualquier secuencia valida de acciones

Camino: secuencia de acciones que permiten pasar de un estado a otro

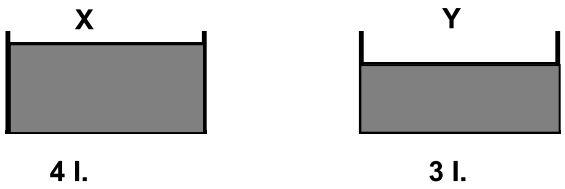
Coste del camino(g): Σ coste de cada acción del camino.

La representación de problemas como un espacio de estados es la base de muchos métodos de IA.

- Permite definir formalmente el problema.
- Permite definir el proceso de resolución de un problema como una combinación de técnicas conocidas y búsqueda.



El Problema de las Jarras de Agua



Queremos obtener 2 litros en X

Especificación del problema:

Representación Estados: (x,y) / $x \in \{0,1,2,3,4\}$, $y \in \{0,1,2,3\}$

Estado inicial: (0,0)

Estado final: (2, N)

Operadores:

R1: (x,y) ^ x<4 → (4,y) *llenar x*

R2: (x,y) ^ y<3 → (x,3) *llenar y.*

R3: (x,y) ^ x>0 → (0,y) *vaciar x*

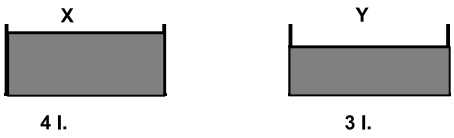
R4: (x,y) ^ y>0 → (x,0) *vaciar y*

R5: (x,y) ^ x+y≥4 ^ y>0 ^ x<4 → (4,y-(4-x)) *llenar x desde y*

R6: (x,y) ^ x+y ≥ 3 ^ x>0 ^ y<3 → (x-(3-y),3) *llenar y desde x*

R7: (x,y) ^ x+y ≤ 4 ^ y>0 → (x+y,0) *vaciar y en x*

R8: (x,y) ^ x+y ≤ 3 ^ x>0 → (0,x+y) *vaciar x en y*



Estado Inicial (0, 0)

R1: (x,y) ^ x<4 → (4,y) llenar x. **R2:** (x,y) ^ y<3 → (x,3) llenar y.

R3: (x,y) ^ x>0 → (0,y) vaciar x. **R4:** (x,y) ^ y>0 → (x,0) vaciar y

R5: (x,y) ^ x+y≥4 ^ y>0 ^ x<4 → (4,y-(4-x)) llenar x desde y

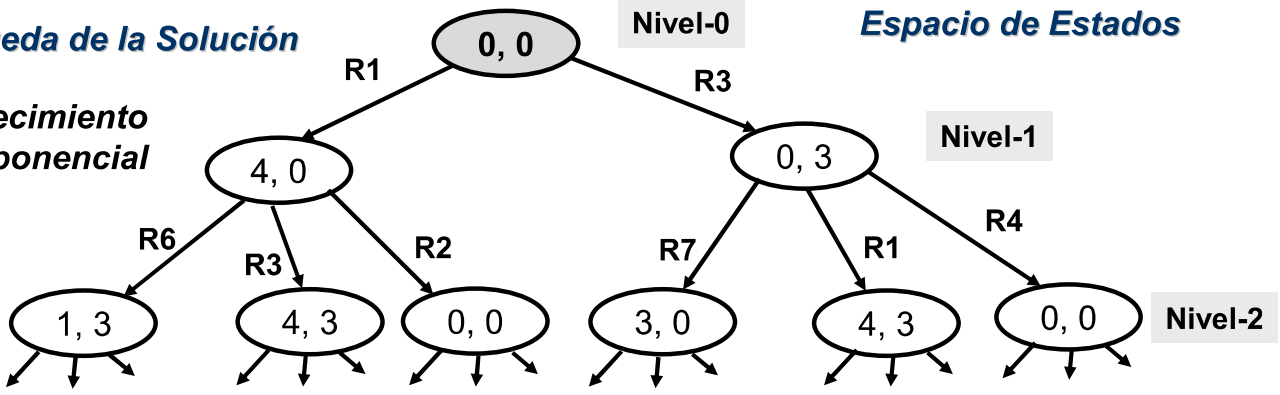
R6: (x,y) ^ x+y ≥ 3 ^ x>0 ^ y<3 → (x-(3-y),3) llenar y desde x

R7: (x,y) ^ x+y ≤ 4 ^ y>0 → (x+y,0) vaciar y en x

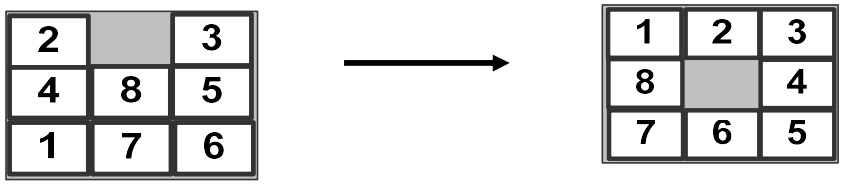
R8: (x,y) ^ x+y ≤ 3 ^ x>0 → (0,x+y) vaciar x en y

Búsqueda de la Solución ***Espacio de Estados***

Crecimiento exponencial



El Problema del Puzle



Especificación del problema:

Representación Estados:

Opción-1:

Matriz 3x3 / $x_{ij} \in \{B,1,2,3,4,5,6,7,8\}$,

Estado inicial: (2,B,3,4,8,5,1,7,6) Estado final: (1,2,3,8,B,4,7,6,5)

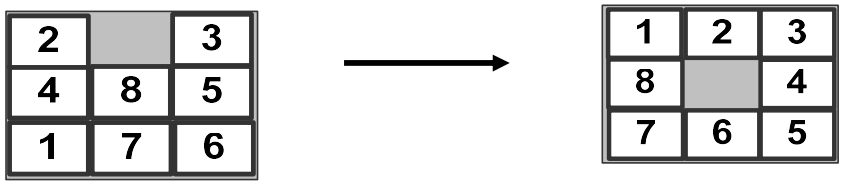
Opción-2:

$\{Pos(2,x_2,y_2), Pos(B,x_0,y_0), Pos(3,x_3,y_3), \ldots.\}$, $x_i, y_i \in \{1,2,3\}$, $i \in \{0,1,2,..,8\}$,

Estado inicial: {Pos(2,1,1), Pos(B,1,2), Pos(3,1,3), Pos (4, 2,1),…}

Estado final: {Pos(1,1,1), Pos(2,1,2), Pos(3,1,3), Pos (4, 2, 3),…}

El Problema del Puzle



Especificación del problema:

Estados: $\{Pos(2,f_2,c_2), Pos(B,f_0,c_0), Pos(3,f_3,c_3), \ldots.\}$, $f_i, c_i \in \{1,2,3\}$, $i \in \{0,1,2,..,8\}$,

Estado inicial: {Pos(2,1,1), Pos(B,1,2), Pos(3,1,3), Pos (4, 2,1),…}

Estado final: {Pos(1,1,1), Pos(2,1,2), Pos(3,1,3), Pos (4, 2,3),…}

Operadores:

Opción-1: Mover cada ficha en cada una de las 4 direcciones: 8 x 4=32 reglas

Opción-2: Mover el blanco en cada una de las 4 direcciones: 4 reglas

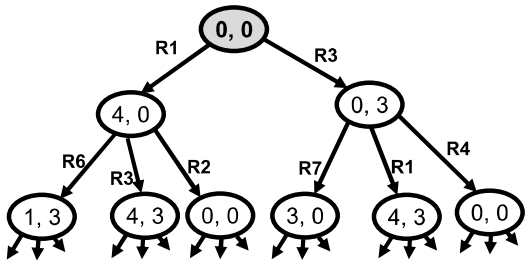
Ejemplo: R↑: Pos (x, f, c), Pos (B, f-1, c) ^ f ≥ 2 → Pos (x, f-1, c), Pos (b, f, c)

Es importante la especificación del problema

Búsqueda de soluciones en un espacio de estados: CONCEPTOS

Objetivo: Obtención de una secuencia de operadores que aplicados al Estado-Inicial conduzcan a un Estado-Meta: *Senda solución*.

Expansión de un estado: aplicar operadores a un estado-padre para generar un nuevo conjunto de estados sucesores: *Estados generados, Estados expandidos*.



Nodo Raíz: Nodo origen de la búsqueda

Nodos hoja: En cada momento, nodos frontera de la búsqueda (aplicación de la estrategia).

Estrategia de búsqueda: elección del estado que se desea expandir en cada momento.

Árbol de búsqueda: Conjunto de estados generados en la búsqueda.

Espacio de estados ≠ árbol de búsqueda

Criterios para medir la eficiencia de una estrategia de búsqueda:

- ¿Permite encontrar una solución?
- ¿Es una buena solución?
- Coste de la búsqueda: tiempo y memoria
- Coste total = coste de la búsqueda + coste del camino

Proceso General de Búsqueda

Función búsqueda (problema, estrategia) **devuelve** {solución, fallo}

Inicializar el árbol de búsqueda con el estado inicial de *problema*

bucle hacer

si no hay estados candidatos para la expansión **devuelve** fallo
si no escoger un nodo a expandir en función de la *estrategia;*

si el nodo describe un estado objetivo **devuelve** solución
si no expandir el nodo y añadir los nodos resultantes al árbol de búsqueda;

fin bucle

fin función

Función Búsqueda_General (problema, estrategia) **devuelve** {nodo, fallo}

{Nodos} := {Estado_Inicial [problema]}:

bucle hacer

si Está_Vacía {nodos} **devolver** fallo;

nodo := eliminar_primero {nodos};

si Objetivo (nodo) = éxito **devolver** nodo;

nodos := Ordenar_estrategia ({nodos}, Expandir(nodo, operadores[problema]));

fin bucle

fin función

2.2.- Caracterización de la Búsqueda de Soluciones

Caracterización Estrategia de Búsqueda:

- Completitud: garantía de encontrar una solución si existe.
- Complejidad temporal: tiempo necesario para encontrar una solución.
- Complejidad espacial: memoria necesaria para realizar la búsqueda.
- Admisibilidad: garantía de encontrar la solución óptima.

Tipos de Búsqueda:

- sin información o búsqueda ciega (orden de expansión de nodos).
- informada o búsqueda heurística (expansión 'inteligente' de nodos).

Estrategia de control: criterios para seleccionar operadores y para anotar aquellas secuencias de operadores ya aplicados y los estados que éstos han producido.

- Estrategia de búsqueda: selección del operador ó siguiente nodo a expandir (anchura, profundidad, heurística). *Determina por donde progresa la búsqueda*.
- Régimen de control: criterios para determinar el conjunto de estados visitados. Irrevocable, Tentativo (backtracking, grafos).

Direccionamiento de la Búsqueda: Forward, Backward, Bi-direccional.

Régimen de Control

Determina si se pueden deshacer pasos (etapas de búsqueda realizadas) y desarrollar la búsqueda por otros nodos, en caso de llegar a un nodo donde no se puede progresar o es poco 'prometedor'.

Los regímenes de control se clasifican en:

Irrevocables: Cuando se decide aplicar un operador en un estado, no se vuelve a reconsiderar a aplicación de otro operador en ese estado.

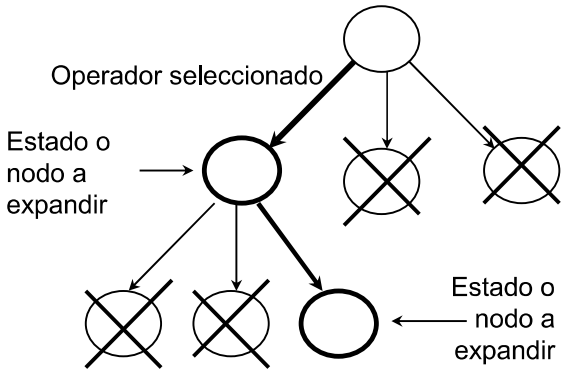
- **Simples: No requieren registrar los estados ya visitados.** En todo momento, se mantiene un único estado en la lista de nodos a expandir, ya que el resto de nodos "hermanos" no se guardan en la lista.
- **Inconvenientes: Ciclos, Estados repetidos, Bloqueos, etc.**

Tentativos:

- **Requieren registrar las secuencias** de operadores aplicados y los estados o nodos ya visitados.
- Permiten control de **ciclos.**

a) Regimen de Control Irrevocable

Cuando se decide aplicar un operador en un estado, **no se vuelve a reconsiderar** la aplicación de otro operador en ese estado.



- Se define una función *inteligente* de escalada f(n) sobre cada estado del problema, máxima/mínima en el estado final.
- En cada estado, se selecciona el operador que maximiza/minimiza dicha función: a través de máximos locales se pretende obtener un máximo global (meta).

Ventajas: Simples. No requieren registrar los estados ya visitados. Se mantiene un único estado en la lista de nodos a expandir, el resto de nodos "hermanos" no se guardan en la lista.

Inconvenientes: Ciclos, Estados repetidos, Bloqueos, etc.

Aplicables, cuando

- El control de estrategia tiene suficiente información en cada estado, para determinar, irrevocablemente, que operador aplicar.
- La aplicación de un operador no debe impedir la aplicación posterior de otro operador que

b) Regimen de Control Tentativo

Los regímenes de control tentativos seleccionan un operador a aplicar (o un estado a expandir) en cada paso mediante la aplicación de una estrategia de búsqueda determinada.

Pero se toman las medidas precisas para poder retornar a ese punto del proceso y aplicar entonces otro operador alternativo al seleccionado previamente.

- Requiere guardar la búsqueda realizada, las diversas decisiones tomados y todos los estados producidos en la búsqueda, permitiendo estudiar globalmente los efectos de la aplicación de secuencias alternativas de operadores.
- El régimen de control tentativo más común es la **exploración en grafos**, independientemente de la estrategia de búsqueda a emplear en la selección del nodo (anchura, profundidad, heurística, …)

Ventajas: Mayor visibilidad de la búsqueda, evita ciclos, estados repetidos y bloqueos.

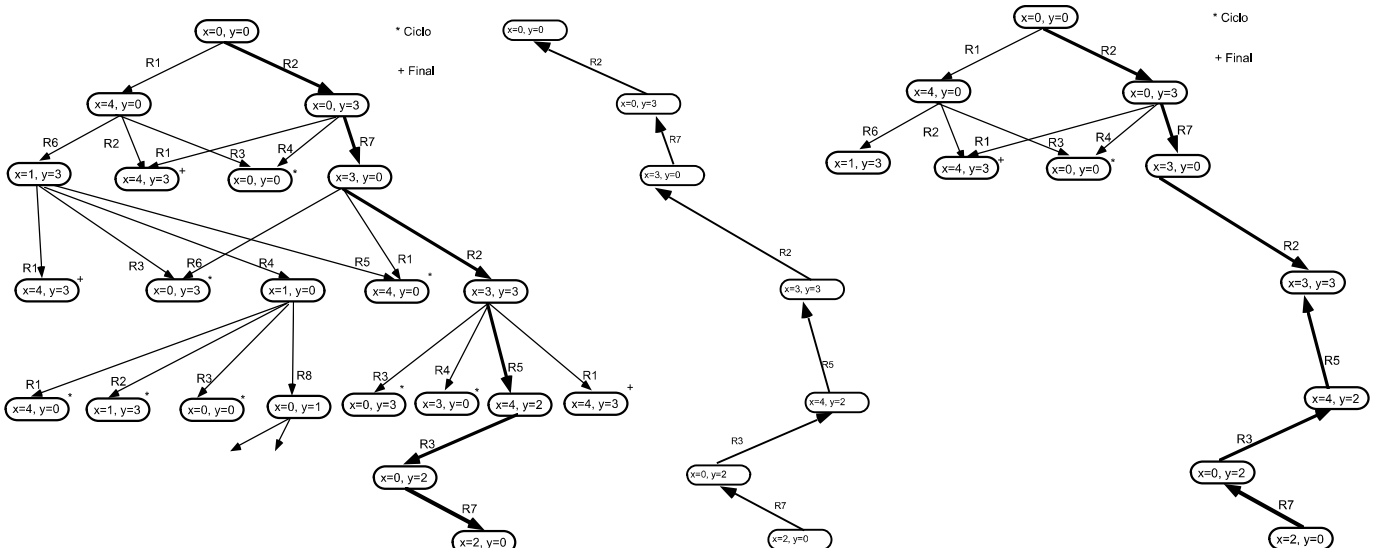
Inconvenientes: Requieren registrar los estados ya visitados: Incremento de Memoria.

Direccionamiento de la Búsqueda

Dirigida por los datos (Forward): Aplicación F-Operadores, No necesita estados meta explicitos

Dirigida por el objetivo (Backward): Aplicación B-Operadores, Necesita metas explicitas, Parecido conducta humana

Bidireccional: La EC decide si se aplican F o B-Operadores



Búsqueda de la Solución en un Espacio de Estados

Problema: {$E_{inicial}$, Operadores, E_{meta}}

Régimen de Control:

- **Irrevocable (Función de Escalada, Voraz): Rápido, Memoria⇓, Ciclos.**
- **Tentativo:**
  - **Backtracking: evita ciclos en senda, Memoria⇒)**
  - **Búsqueda en grafos (primero el mejor, A*): Evita repetir búsqueda. Memoria⇑**

Estrategia de Búsqueda *(elección de la exploración)*:

- **No Informada (Anchura, Profundidad, Coste Uniforme, Prof. Iterativa)**
- **Informada (Función de escalada, A, A*, Agenda). Variantes A*.**

Encadenamiento *(qué obtener)*: (hacia delante, Hacia Atrás, Bidireccional)

Características

- **Coste (Temporal, Espacial). Coste de la búsqueda / de la estrategia.**
- **Eficiencia Heurística (experimental: P, B).**
- **Completitud.**
- **Admisibilidad.**

2.3.- Estrategias de búsqueda no informadas.

Búsqueda en anchura.

Búsqueda por coste uniforme.

Búsqueda en profundidad o backtracking

Búsqueda por profundización iterativa.

```
Función Búsqueda_General (problema, estrategia) devuelve {nodo, fallo}
    {Nodos} := {Estado_Inicial [problema]}
    bucle hacer
        si Está_Vacía {nodos} devolver fallo
        nodo := eliminar_primero {nodos}
        si Objetivo (nodo) = éxito devolver nodo
        nodos := Ordenar_estrategia ({nodos}, Expandir(nodo, operadores[problema]))
    fin bucle
fin función
```

a) Búsqueda en anchura (BA)

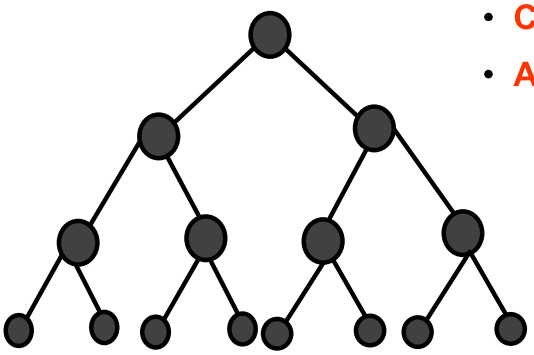
Ordenar_estrategia ({nodos}: Prioritario el nodo más antiguo

Nivel-0

Nivel-1

Nivel-2

Nivel-3



- **Completa**
- **Admisible**

  (si coste de operadores unitario y no negativo)

Coste ***(factor de ramificación: b, nivel nodo solución: d):***

nodos expandidos $1 + b + b^2 + b^3 + b^4 + \ldots\ldots\ldots + b^d$ **$O(b^d)$**

nodos generados $1 + b + b^2 + b^3 + b^4 + \ldots\ldots\ldots + b^d + (b^{d+1} - b)$ **$O(b^{d+1})$**

- coste espacial = coste temporal
- coste espacial más crítico que el temporal
- coste temporal inviable para b↑ y d ↑

Búsqueda en anchura (BA)

b = 10 1000 nodos/segundo 100 bits/nodo

Profundidad	Nodos	Tiempo	Memoria
0	1	1 ms	100 b
2	111	.1 s	11 kb
4	11.111	11 s	1mb
6	10^6	18 m	111 mb
8	10^8	31 h	11 gb
10	10^{10}	128 d	1 tb
12	10^{12}	35 a	111 tb
14	10^{14}	3500 a	11.111 tb

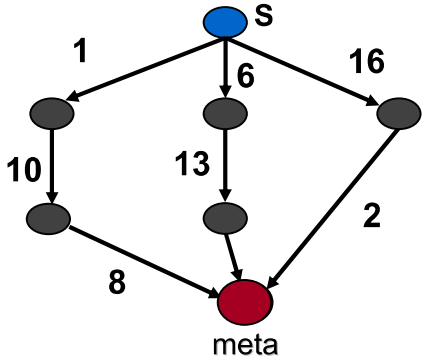
b) Búsqueda por coste uniforme (BCU)

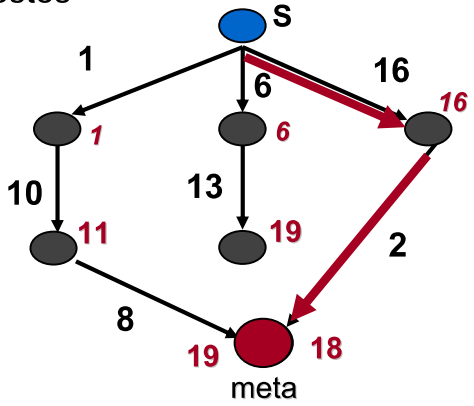
Ordenar_estrategia ({nodos}: Prioritario el nodo de menor coste dede raíz

Cada rama tiene asociado un coste > 0.

Cada nodo tiene un coste desde la raíz: g(n)=Σcostes

Se expande el nodo con menor g(n)





Coste (ramificación: b, coste solución óptima: C*, coste mínimo de rama ε): **O($b^{(C^*/\varepsilon)}$)**

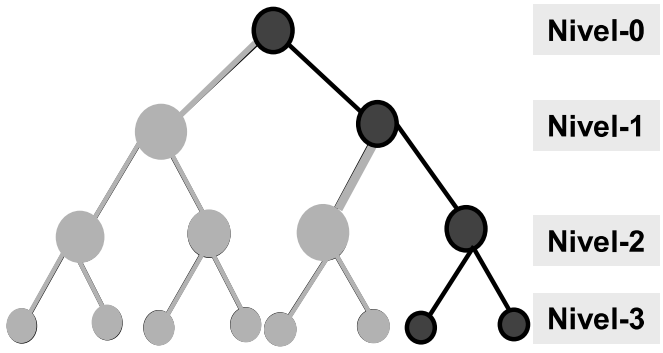
Completa

Admisible si se cumple g(sucesor(n)) > g(n) *(coste positivo de las ramas)*

c) Búsqueda en profundidad o backtracking (BP)

Ordenar_estrategia ({nodos}: Prioritario el nodo más nuevo

- ✓ Nuevos estados a la cabeza de la lista.
- ✓ Vuelta atrás cuando:
  - a) nodo no meta sin hijos,
  - *b) Se alcanza una profundidad máxima* (variante BPL)
- ✓ Puntos muertos (árboles muy profundos o infinitos, bucles).
- ✓ No completa.
- ✓ No admisible.



Alcanzada profundidad Máxima (Nivel-3)

Costes *(factor de ramificación: b , máx profundidad: d)*

- *Coste espacial moderado* (sólo se guarda camino desde nodo raíz hasta nodo hoja y nodos aún no expandidos) **O(b·d)**
- Coste temporal **$O(b^d)$**, aunque en término resulta *medio menos costosa que amplitud*

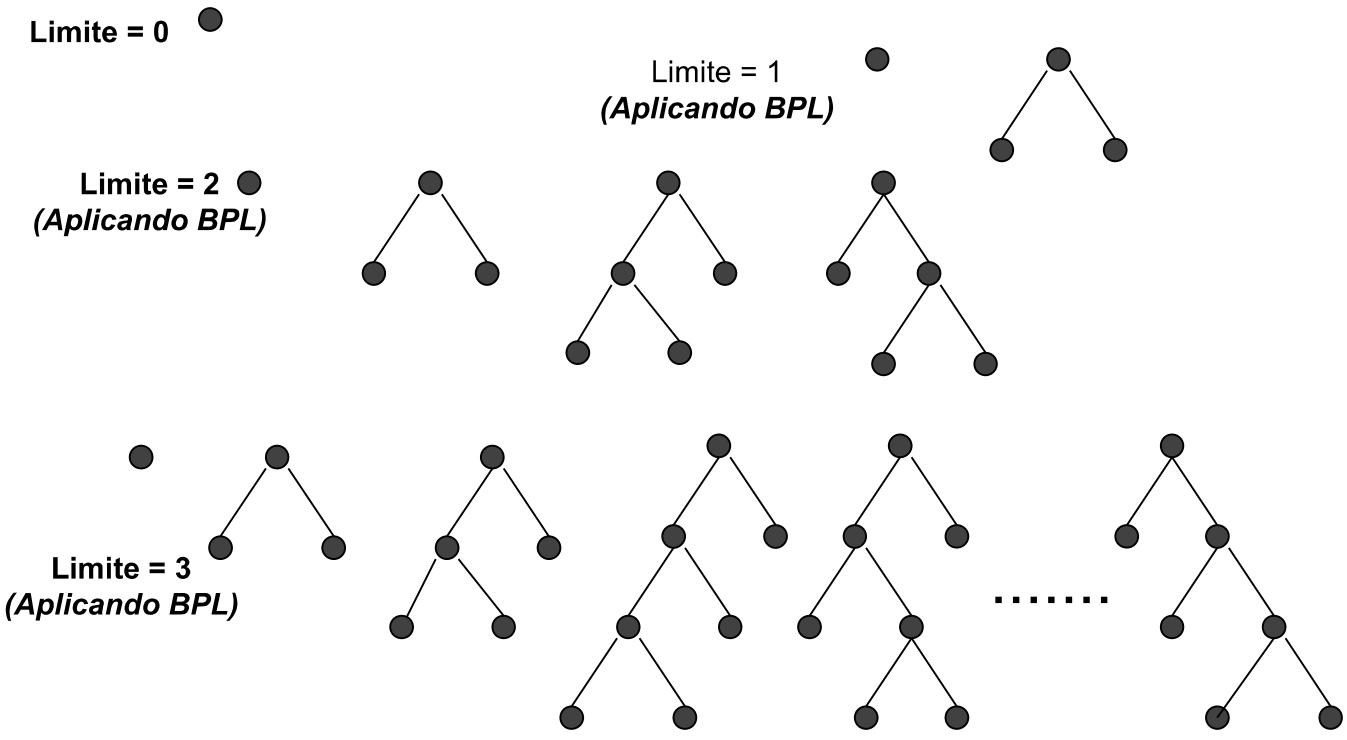
d) Búsqueda por profundización iterativa (BPI)

```
Función Búsqueda_Profundización_Iterativa (problema )
        para profundidad = 0 hasta ∞ hacer
            resultado = Búsqueda_Limitada_Profundidad(problema, profundidad)
            si resultado <> fallo devolver resultado
        fin para
        devolver fallo
fin función
```

Realiza **iterativamente** una **búsqueda limitada en profundidad**, desde una profundidad-máxima 0 hasta ∞.

- Resuelve la dificultad de elección del límite adecuado para BLP.
- Combina ventajas de búsqueda primero en amplitud y primero en profundidad.
- **Completa y admisible.**
- Complejidad temporal $\mathbf{O(b^d)}$, complejidad espacial $\mathbf{O(b \cdot d)}$

Búsqueda por profundización iterativa (BPI)



Búsqueda por profundización iterativa (BPI)

BPI puede parecer ineficiente por generar estados repetidamente, pero realmente no es así:

- En un árbol de búsqueda con ramificación similar en cada nivel, la mayor parte de los nodos está en el nivel inferior.
- Los nodos de nivel inferior (d) son generados una sola vez, los anteriores dos veces, etc. Los hijos de la raiz se generan d veces.

Nodos generados **(ramificación b = 10 , profundidad d=5)**

BPI:	$(d)\cdot b + (d-1)\cdot b^2 + (d-2)\cdot b^3 + \ldots\ldots\ldots + 1\cdot b^d$	123.456
En Anchura:	$1 + b + b^2 + \ldots\ldots\ldots\ldots + b^{d-2} + b^{d-1} + b^d + (b^{d+1} - b)$	1.111.100

- La búsqueda en anchura generará algunos nodos en profundidad d+1, mientras que BPI no lo hace. Por ello, BPI es en realidad más rápida que BA.

BPI es la método de búsqueda no informada preferido cuando el espacio de búsqueda es grande y no se conoce a priori la profundidad de la solución (en otro caso, BP sería la elección)

Comparación de Estrategias de Búsqueda no informadas.

Criterio	BA	BCU	BP	BPI
Tiempo	b^d	b^d	b^m	b^d
Espacio	b^d	b^d	$b\cdot m$	$b\cdot d$
¿Óptima?	si	si	no	si
¿Completa?	si	si	no	si

El problema de los estados repetidos y/o ciclos

Problema: Visitar estados ya visitados y expandidos anteriormente en algún otro camino

- Debido a Operadores reversibles: S1 → S2, S2 → S1
- Da lugar a una búsqueda infructuosa
- **También produce ciclos infinitos**

Problemas sin ciclos (depende formulación).

Eliminar repetición de estados → reducción exponencial coste búsqueda

Métodos para evitar estados repetidos (de más a menos eficientes y coste temporal):

- No regresar al estado del que se acaba de llegar (no generar un sucesor que sea su propio padre). *Requiere guardar solo el padre.*
- No crear caminos con ciclos (no generar nodo que sea su propio antecesor): *Requiere guardar solo la senda actual.*
- No generar un estado que se haya generado previamente en la búsqueda (coste espacial $O(b^d)$). *Requiere guardar todos los estados alcanzados.*