Resolución de problemas mediante búsqueda

Búsqueda ciega, no informada

Búsqueda heurística, informada

Agente simple de resolución de problemas: Tipos de problemas

- Agentes de resolución de problemas son un tipo de agentes basados en el objetivo
- Determinista, completamente observable → problema de estado simples:
  - » El agente sabe exactamente en qué estado se encuentra. La solución es una secuencia
- No-observable → problema sin sensores
  - » El agente no sabe dónde se encuentra. La solución es una secuencia.
- No determinista y/o parcialmente observable → problema con contingencias.
  - » Las percepciones proporcionan información actualizada sobre el estado actual.
- Espacio de estados desconocido → problema exploratorio

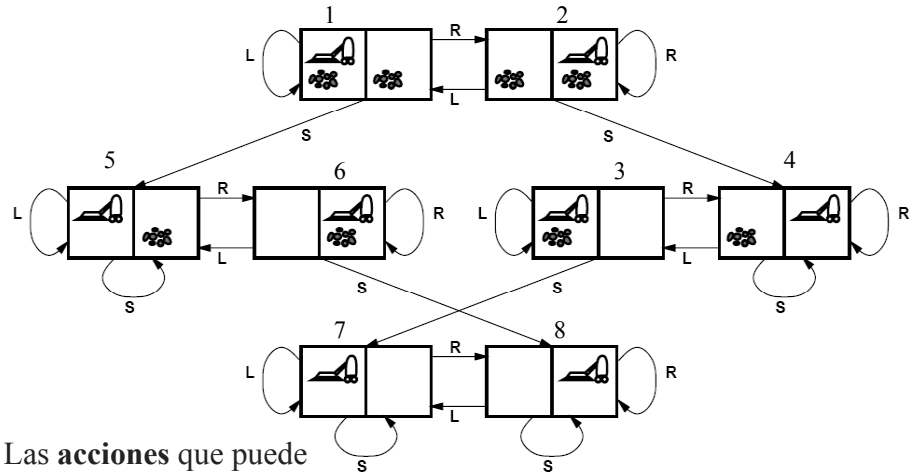
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Formulación de problemas, I
Problema de la aspiradora

- Se dispone de una aspiradora con acceso a dos habitaciones y con la capacidad de aspirar basura
  - » Entorno accesible y determinista: Problema de estados simples

El agente está en alguna posición

El mundo tiene dos alternativas {LIMPIO, SUCIO}



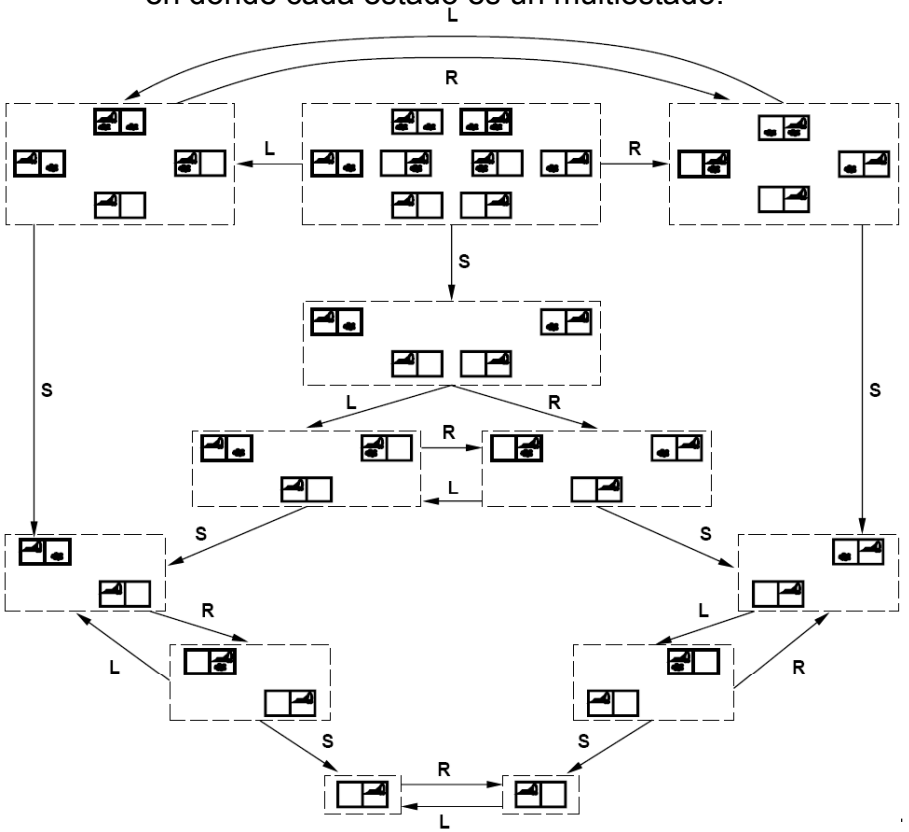
Las **acciones** que puede realizar el agente:
L: left (izquierda)
R: right (derecha)
S: suck (aspirar)

Objetivo: limpiar toda la suciedad.
Equivale al conjunto de estados {7,8}

Formulación de problemas, II
Problema de la aspiradora

» Entorno no accesible (parcialmente observable) y/o no determinista: Problema de estados múltiples

– DEF: Un problema de estados múltiples es un caso particular del caso de un problema de estado simple, en donde cada estado es un multiestado.



Formulación de problemas, III (definición)

- Abstracción de un problema
  - » DEF: Proceso de eliminar los detalles de la representación formal de un problema
- Problemas bien definidos
  - » La formulación de un problema requiere
    - – Especificación de **estados iniciales**: uno o más estados que describen las situaciones de partida
    - – Especificación de **estados objetivos**: uno o más estados que podrían ser soluciones admisibles del problema
      - **Función/test objetivo**: determina si un estado es un estado objetivo.
    - – Especificación del conjunto de **acciones/operadores** que pueden realizarse sobre cada estado.
      - **Función sucesor**: estando en un estado, aplicando un operador indica a qué estado se accede. S: x → S(x)
    - – Definición de un **espacio de estados** del problema
      - Conjunto de todos los estados alcanzables a partir del estado inicial aplicando cualquier secuencia de operadores
      - Determina un grafo: estados - arcos - caminos
    - – **Función de coste** de aplicación de los operadores

Formulación de problemas, V (Problema Bien Definido)

Estados? Posiciones de la suciedad y del robot → (1, AS, S), (2, S, AS), (3, AS,) (4, S, A), (5, A, S), (6, , AS) (7, A,), (8, , A)

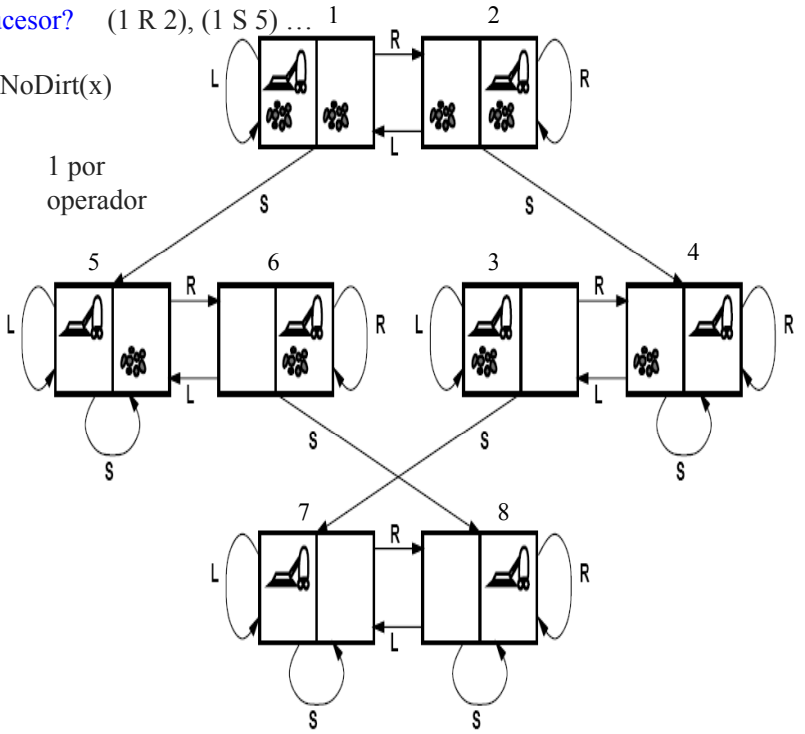
Estado inicial? El que se designe

Operadores? Left (L), right (R), suck (S)

Función sucesor? (1 R 2), (1 S 5) …

Objetivo? NoDirt(x)

Coste del camino? 1 por operador



Resolución mediante búsqueda, I

- La resolución de un problema de IA mediante búsqueda consiste en:
  - » Aplicar una determinada estrategia de control que conduzca a encontrar un camino desde el estado inicial hasta algún estado objetivo del espacio de estados
    - – Exige examinar las posibles secuencias de acciones
    - – Se debe seleccionar aquella secuencia que sea la mejor según un determinado criterio
- Los objetivos fundamentales de la resolución de un problema mediante búsqueda son:
  - » Encontrar una solución
  - » Que la solución tenga **coste total** mínimo:
    - » Coste de búsqueda (coste *offline*):
      - » Tiempo y memoria necesarios.
    - » Coste del camino solución (coste *online*).

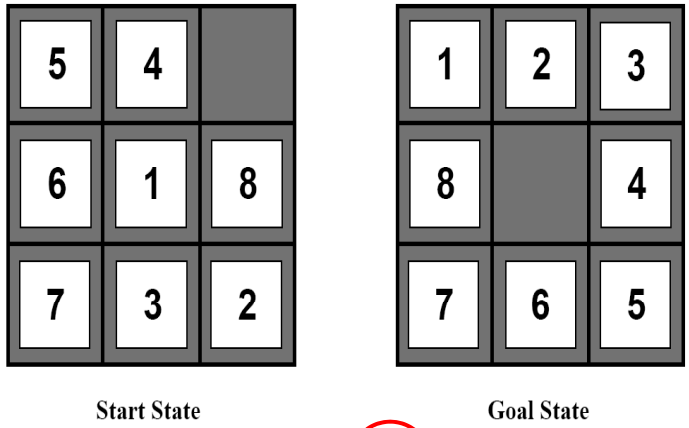
Resolución mediante búsqueda, II

- El entorno del problema influye sobre la secuencia de acciones solución
  - » Ejemplos
    - – Entorno determinista
      - Para cada estado inicial, hay una secuencia fija de operadores que llevan al objetivo
        - » Comenzar en #5
        - » Secuencia: [R, S]
    - – Entorno determinista no accesible
      - Para cada estado inicial, hay una secuencia fija de operadores que llevan al objetivo
        - » Comenzar en algún estado: {1 .. 8}
        - » Secuencia: [R, S, L, S]
    - – Entorno no determinista, semiaccesible
      - *La absorción deposita algunas veces suciedad, pero sólo cuando no hay suciedad*
        - » sensor de posición y sensor local de suciedad → Se percibe [L, Limpio].
        - » Comenzar en #5 ó #7 [L, Limpio]
        - » Secuencia: [R, If *Sucio* Then S]

Ejercicios propuestos

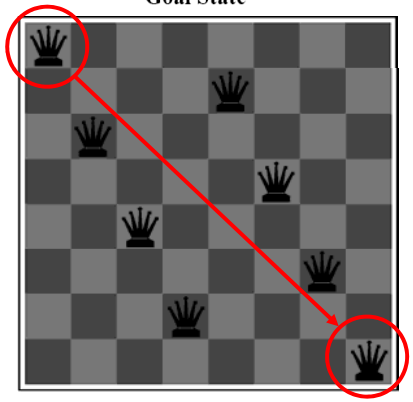
En ambos casos determinar:
(a) estados, (b) estado inicial, (c) operadores, (d) objetivo,
(e) coste del camino

1.- Dado un puzzle de 8 piezas, alcanzar el estado objetivo mediante sucesivos movimientos del hueco.



2.- Dado un tablero con N reinas, encontrar una configuración en la que no estén enfrentadas entre si

Solución no válida →



Soluciones a ejercicios propuestos

- Problema del *8-puzzle*
  - » Estados: posiciones de las piezas y hueco

```
(setf *estado0*
    '((0 5)(1 4)(2 nil)
      (3 6)(4 1)(5 8)
      (6 7) (7 3) (8 2))
```

  - » Operadores:
    - – HuecoA: Dcha – Izda – Arriba – Abajo
  - » Objetivo: (ver gráfico anterior)
  - » Coste operadores: 1
- Problema de las 8 reinas (en general de las N reinas/damas):
  - » Coste operadores: 1 (el camino solución siempre tiene coste 8).
  - » Posible representación (1):
    - – estado: N reinas en el tablero
    - – operadores: añadir una reina a una posición vacía.
  - » Posible representación (2):
    - – estado: N reinas en el tablero (no atacándose).
    - – Operadores: añadir una reina en la columna vacía más a la izquierda tal que no sea atacada por ninguna de las ya existentes.
    - – Menos operadores que en la representación (1)

Otros ejemplos, I

- Problemas de Criptoaritmética

```
   FORTY          29786
+    TEN     +     850
     TEN           850
  ------        ------
   SIXTY         31486
```

  - » Estados: algunas letras sustituidas por dígitos.
  - » Operadores: sustituir una letra por un dígito que no aparece ya dentro del estado.
  - » La solución se encuentra a profundidad conocida.
    - – Todas las soluciones son igualmente válidas luego el coste del camino es 0

Otros ejemplos, II

- Misioneros y caníbales
  - » Hay 3 misioneros y 3 caníbales en la orilla izquierda de un río. Un bote puede transportar a 1 ó 2 personas de una orilla a otra.
    - – Objetivo: pasar a todos a la otra orilla.
    - – Condición: No puede ocurrir nunca que si en una orilla hay algún misionero haya a la vez un número mayor de caníbales (se los comerían).
  - » Estados:
    - – Parámetros: número misioneros lado izquierdo, número caníbales lado izquierdo, posición bote (izquierda o derecha).
    - – Se debe verificar la condición.
  - » Operadores:
    - – Transportar 1 misionero.
    - – Transportar 1 caníbal.
    - – Transportar 2 misioneros.
    - – Transportar 2 caníbales.
    - – Transportar 1 misionero y 1 caníbal.
  - » Coste operador: 1

Otros ejemplos, III

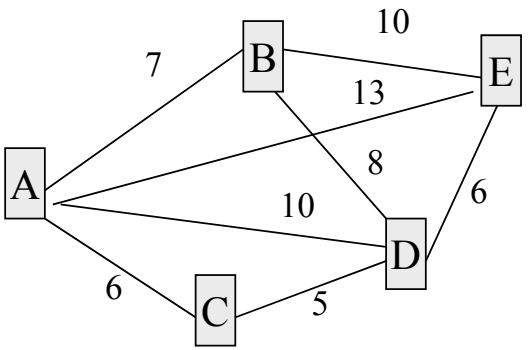
- Problema de mapa de carreteras
  - » Viajar de una ciudad a otra recorriendo la menor distancia posible
- Problema del viajante de comercio
  - » Un viajante debe viajar recorriendo un conjunto de ciudades. Debe partir de una ciudad inicial y, tras recorrer todas las ciudades, volver a la ciudad de inicio. Debe visitar exactamente 1 vez todas las ciudades (excepto la de inicio que la visita 2 veces).

Estado inicial: (A)

Estado final: (A ... A)

Estados:
(A C) (A D) (A B) (A E)
(A C D) (A C D E)
(A C D E B) ...

Operadores:
VisitarCiudadA = VA,
VisitarCiudadB = VB, ...
VisitarCiudadE = VE



Búsqueda en árboles, I

- Representación de un nodo:
  - » Estado: elemento del espacio de estados que corresponde con el nodo.
  - » Nodo padre: el nodo en el árbol de búsqueda que ha generado este nodo.
  - » Acción/Operador: operador que se aplicó al padre para generar este nodo.
  - » Coste del camino: el coste desde el nodo inicial. Denotado por *g(n)*.
  - » Profundidad en el árbol de búsqueda: número de pasos a lo largo del camino desde el nodo inicial.
- Distinguir los conceptos:
  - » Espacio de estados:
    - – Finito
  - » Árbol de nodos: se genera
    - – Finito o infinito

Búsqueda en árboles, II

- Algoritmo de búsqueda en árboles
- Algoritmo de expansión de nodos

```
function TREE-SEARCH( problem, fringe) returns 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 SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Búsqueda en árboles, III

- Algoritmo de búsqueda en árboles (descripción informal):

funcion búsqueda-árboles (problema, *frontera*)
devuelve una *solución* o *fallo*
inicializa árbol de búsqueda con estado inicial
bucle hacer
si no hay candidatos para expandir en *frontera*,
entonces devolver *fallo*
en otro caso
escoger de *frontera*, según estrategia, nodo para expandir y eliminar de ella
si el nodo es objetivo (contiene *estado objetivo*)
entonces devolver *solución*
en otro caso
expandir nodo insertando sucesores en la *frontera* (añadir nodos resultantes al árbol)

Búsqueda no informada *vs* búsqueda informada

- Búsqueda no informada o ciega:
  - » Sólo usan la información de la definición del problema
  - » no disponen de ninguna información adicional a la propia definición del mismo
- Estrategias:
  - » Búsqueda primero en anchura.
  - » Búsqueda primero en profundidad.
  - » Búsqueda limitada en profundidad.
  - » Búsqueda iterativa en profundidad.
  - » Búsqueda bidireccional.

- Búsqueda informada o heurística:
  - » Usan la información de definición del problema y el coste del estado actual al objetivo.
- Estrategias:
  - » *Best first*
  - » Búsqueda Avara
  - » A*
  - » IDA*
  - » Mejora iterativa

Estrategias de búsqueda ciega, I

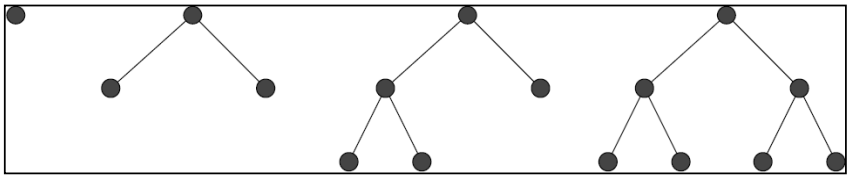
- Criterios de evaluación de estrategias:
  - » Completitud (encontrar solución)
  - » Optimización (encontrar la mejor solución)
  - » Complejidad espacial (memoria necesaria)
  - » Complejidad temporal (tiempo necesario)
- Estrategias de búsqueda:
  - » Hipótesis:
    - – Todos los operadores tienen el mismo coste (por ejemplo 1).
    - – El factor de ramificación es siempre finito.
  - » Las complejidades temporal y espacial se miden en términos de:
    - – **m** = profundidad máxima del árbol de búsqueda (puede ser infinito)
    - – **d** = profundidad de la mejor solución (de la de menor coste)
    - – **b** = factor de ramificación (máximo nº de sucesores de cualquier nodo del árbol de búsqueda)

Estrategias de búsqueda ciega, II

- Búsqueda en anchura (BFS):
  - » Completo y óptimo
  - » Complejidad espacial $O(b^{d+1})$
    - – Se almacenan todos los nodos en memoria
  - » Complejidad temporal =
    - – número de nodos expandidos =

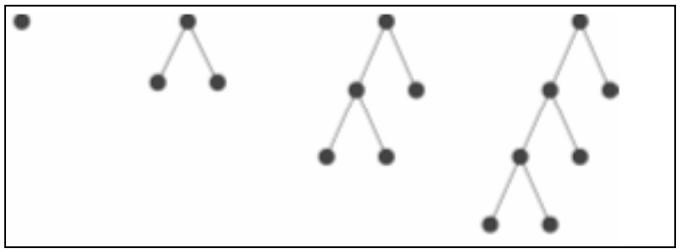
$$1 + b + b^2 + \ldots + b^d = \frac{1 - b^{d+1}}{1 - b} = O(b^{d+1})$$

  - » Frontera: Cola FIFO
    - – Número de nodos generados para b=10, 1000 nodos/segundo, 100 bytes/nodo:
      - » d=2, 111 nodos, 0.1 seg., 11 Kb
      - » d=6, 1.000.000 nodos, 18 minutos, 111 Mb
      - » d=12, 10^{12} nodos, 35 años, 111 Tb



Estrategias de búsqueda ciega, III

- Búsqueda en profundidad (DFS):
  - » No es óptimo
    - – Puede encontrar un camino peor
  - » No es completo
    - – Puede no acabar
  - » Frontera: Cola LIFO
  - » Complejidad temporal = $O(b^m)$
    - – Terrible si m>>d
    - – Más eficiente que BFS si la solución está accesible.
  - » Complejidad espacial: lineal
    - – número de nodos necesarios = un camino hasta una hoja y los hermanos de cada nodo del camino = $O(bm)$



Estrategias de búsqueda ciega, IV

- Búsqueda limitada en profundidad (DLS):
  - » Caso particular de *Búsqueda en profundidad*. Se utiliza un límite de profundidad (l). Los nodos a profundidad l no tienen sucesores.
  - » No es óptimo (idem DFS)
    - – Puede encontrar un camino peor
  - » No es completo, en general, aunque:
    - – sí es completo cuando

$$l \geq d$$

  - » Complejidad temporal = $O(b^l)$
  - » Complejidad espacial: lineal
    - – número de nodos necesarios = un camino hasta una hoja y los hermanos de cada nodo del camino = $O(bl)$

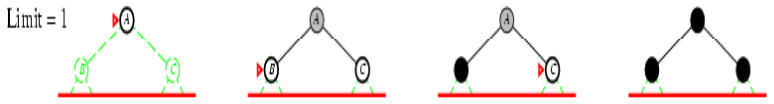
Estrategias de búsqueda ciega,V

- Búsqueda iterativa en profundidad (IDS):
  - » Son búsquedas en profundidad con límites 0, 1, 2, 3, 4, ...
  - » Es óptimo y completo
  - » Complejidad espacial = $O(bd)$
  - » Complejidad temporal
    - – número total de expansiones (los nodos con la profundidad de la mejor solución se expanden 1 vez; los siguientes 2 veces, los siguientes 3 veces, …) =

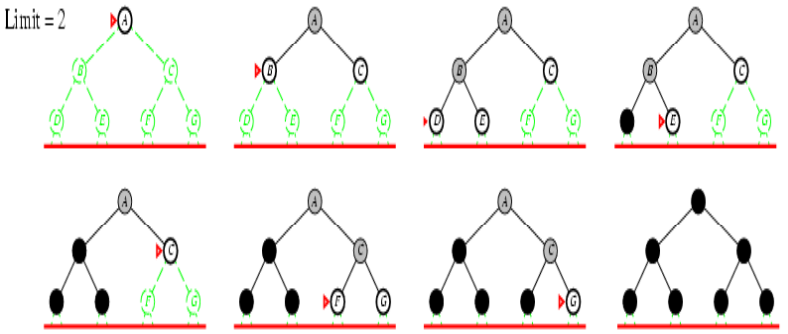
$$\begin{aligned} & 1b^d + 2b^{d-1} + 3b^{d-2} + ... \\ & +(d-1)b^2 + db + (d+1)b^0 \\ & = O(b^d) \end{aligned}$$

  - » Método preferido cuando no se conoce la profundidad de la solución.

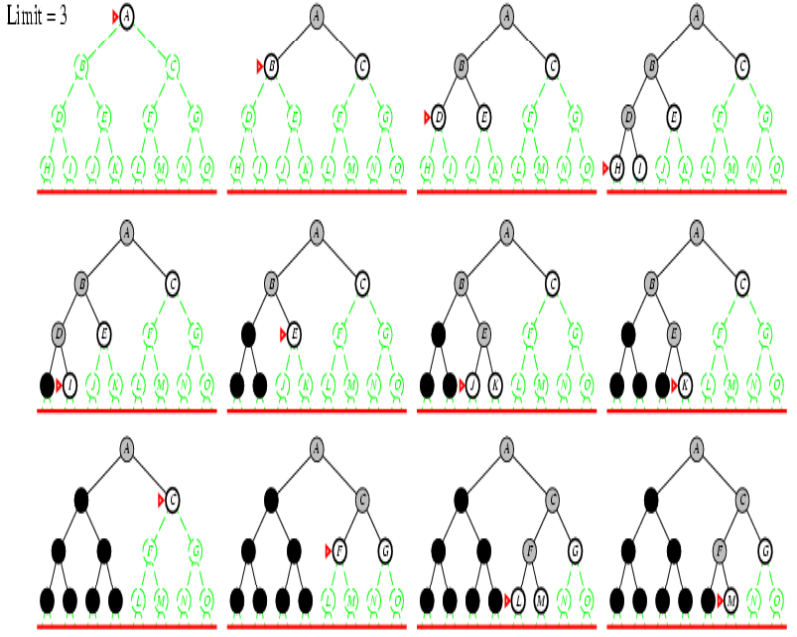
Iterative deepening search l =1



Iterative deepening search l =2



Iterative deepening search l =3



Estrategias de búsqueda ciega,VI

COMPARACIÓN DLS - IDS

- Para *b = 10*, *d = 5*
  - » $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111111$
  - » $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123456$

- Overhead = (123456 - 111111)/111111 = 11%

Estrategias de búsqueda ciega, VIII

- Búsqueda de coste uniforme:
  - » Los resultados anteriores pueden no verificarse cuando los costes de los arcos son variables. Hay que tener en cuenta los costes variables para los arcos:

$$coste(\Gamma) \geq k > 0, \forall \Gamma$$

  - » Para un nodo *n* se define:

    g(n) = coste desde nodo inicial
  - » Se expande el nodo con menor valor de *g*
  - » Completo y óptimo
  - » Si todos los arcos tienen el mismo coste, se tiene búsqueda en anchura (BFS).
    - – Si todos los arcos tienen el mismo coste, g(n)=profundidad(n)
  - » Complejidad espacial y temporal =

$$O(b^{\tilde{d}}), \tilde{d} = \frac{\text{coste} - de - mejor - solución}{minimo - valor - \text{coste}}$$

Estrategias de búsqueda ciega, VII

- Búsqueda bidireccional:
  - » Buscar simultáneamente desde estado inicial hasta objetivo y viceversa hasta que ambas búsquedas *se encuentren*.
  - » Optimo y completo.
  - » Complejidad espacial y temporal: $O(b^{d/2})$
    - – Espacial: Al menos uno de los árboles en memoria siempre
    - – Temporal: Comprobar que un nodo pertenece al otro árbol. Tiempo constante con tabla *hash*
  - » Ejemplo:
    - – d=6 y b=10 con BFS (anchura) en cada dirección
      - En el peor caso p=3 en cada dirección
      - 22.200 nodos generados
      - 11.111.100 si sólo hiciésemos BFS
  - » Dificultades
    - – Cálculo de predecesores.
    - – Varios estados objetivo.
    - – Significado de “encontrarse las búsquedas”
      - Abstracción de la función objetivo (*jaque mate*)
    - – Determinación del tipo de búsqueda en cada dirección.

Estrategias de búsqueda ciega, IX

- Cuadro resumen:

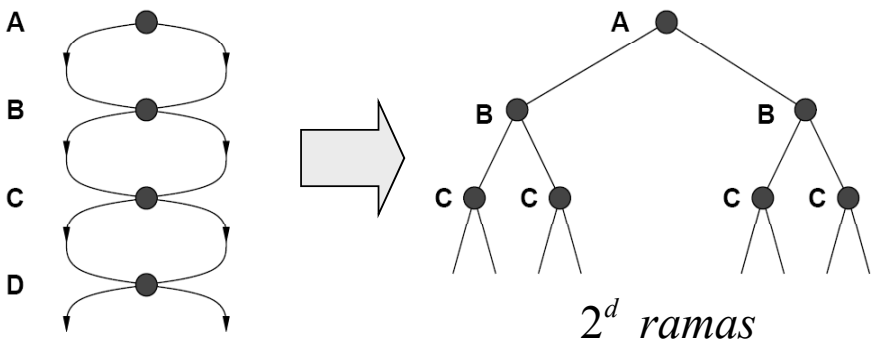
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Eliminación de estados repetidos, I

- La repetición de estados incrementa la complejidad de la estrategia de búsqueda
- Si la estrategia no los detecta (comparar el nodo a expandir con los ya expandidos), un problema resoluble puede llegar a ser irresoluble.
- Situación habitual en problemas de rutas y acciones reversibles
  - » Ejemplo: espacio con d+1 estados

Para los d+1 estados (d es la profundidad máxima)

El árbol de búsqueda contendrá 2^d ramas. Poda.



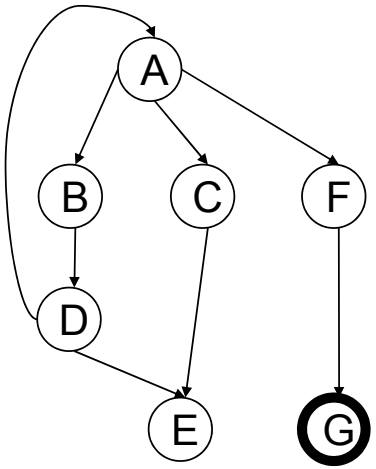
Eliminación de estados repetidos, II

- Para evitar que se repitan estados, se pueden considerar tres métodos:
  1. No generar un nodo hijo de un nodo si los dos pertenecen al mismo estado
  2. Evitar ramas con ciclos (en un camino desde el nodo inicial, hay dos nodos que pertenecen al mismo estado)

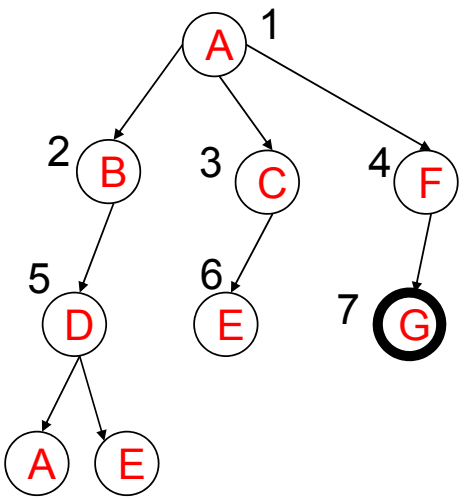
     El método 2) incluye al 1)
  3. Si al generar un nodo, su estado asociado, ya ha sido generado por otro nodo, eliminar el nodo peor (y sus descendientes) del árbol de búsqueda
     1. El método 3) incluye al 2) y, por tanto, al 1)
     2. Este método es el más caro (hay que mantener todos los nodos en memoria).
- Estructuras de datos
  - » Listas cerradas (nodos expandidos)
  - » Listas abiertas (frontera de nodos no expandidos)
- Algoritmo general de búsqueda en grafos
  - » (Russell, 2nd. Ed., sec. 3.5)

Ejemplo I
(Eliminación Estados Repetidos)

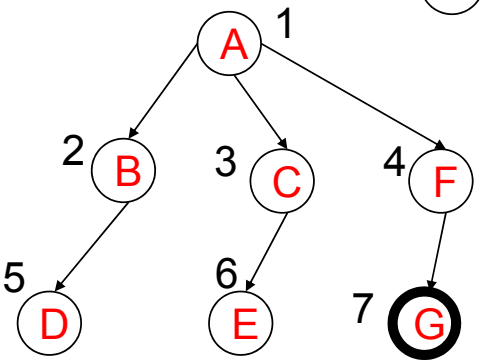
- Realizar búsqueda en anchura
  - » Costes unitarios. Estado inicial: {A} Estado objetivo: {G}



Sin EER

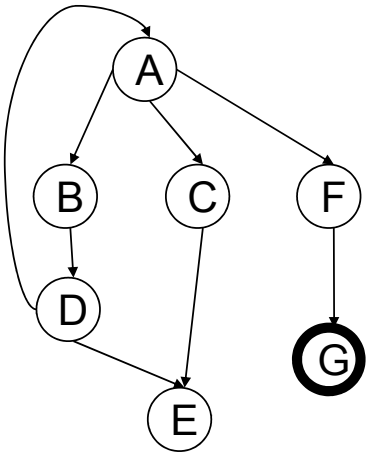


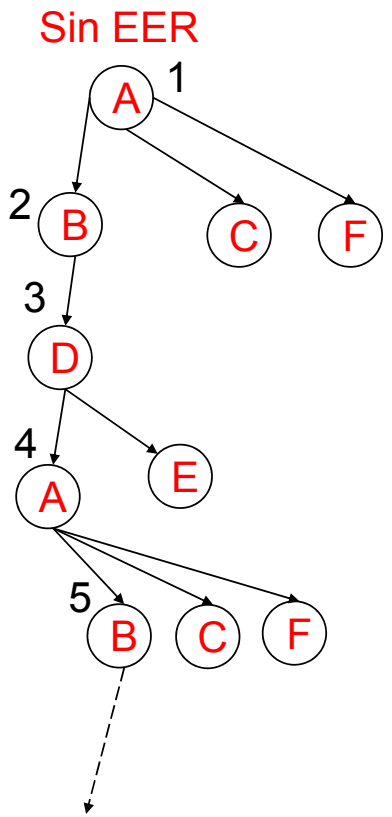
Con EER

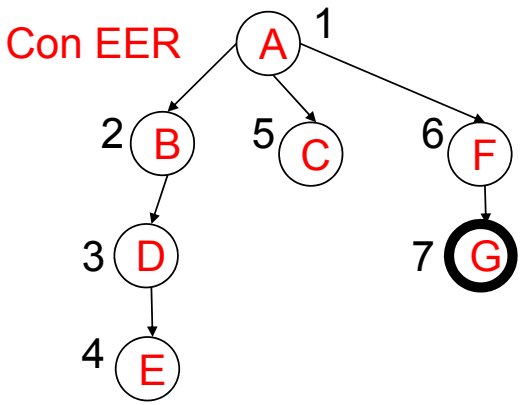


Ejemplo II
(Eliminación Estados Repetidos)

- Realizar búsqueda en profundidad
  - » Costes unitarios. Estado inicial: {A} Estado objetivo: {G}

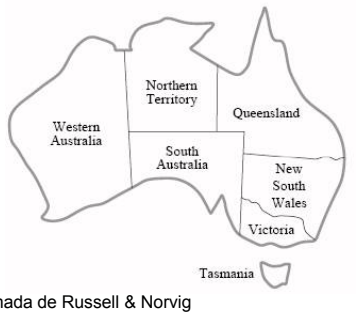






Problemas de satisfacción de restricciones, I

- *Constraint Satisfaction Problems (CSP)*
  - » Un conjunto de ***variables*** cuyos valores están definidos en un dominio (finito o infinito)
  - » Un conjunto de ***restricciones*** que involucran una o más variables del problema (ecuaciones lineales/no lineales)
  - » Los ***estados*** del problema que se definen mediante asignaciones variable – valor
  - » Una ***función objetivo*** que optimice la solución del CSP
- Grafos de restricciones: Representación de CSP
  - » Nodos: variables
  - » Arcos: restricciones



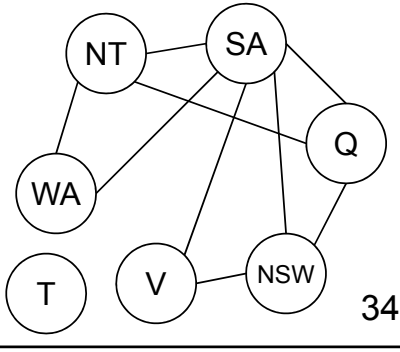


Imagen tomada de Russell & Norvig

Problemas de satisfacción de restricciones, II

- Estrategia de *backtracking*
  - » Búsqueda en profundidad
  - » Asigna valores a variables (una cada vez)
  - » Retrocede en el árbol cuando el dominio de asignación de una variable en el árbol es vacío
- Ejemplos
  - » Problema 8 damas.
  - » Criptoaritmética.
  - » Coloreo de mapas.

	WA	NT	Q	NSW	V	SA	T
Dominio inicial	RGB	RGB	RGB	RGB	RGB	RGB	RGB
WA = R	R	GB	RGB	RGB	RGB	GB	RGB
Q = G	R	B	G	R B	RGB	B	RGB
V = B	R	B	G	R	B		RGB

Problemas de satisfacción de restricciones, III

- Los problemas discretos (dominio finito) se pueden resolver utilizando búsqueda:
  - » Estado inicial
    - – todas las variables sin asignar
  - » Profundidad máxima
    - – número de variables = profundidad de todas las soluciones
    - – Se puede utilizar búsqueda en profundidad.
  - » Cardinal espacio búsqueda
    - – Producto de los cardinales de los dominios de las variables
  - » Se puede hacer:
    - – Eliminación de ramas en donde alguna restricción no se satisface (*backtracking*)
    - – Propagación de restricciones, para reducir los posibles valores de las variables por asignar.

Problemas de satisfacción de restricciones, IV

- Estados: 4 reinas en 4 columnas ($4^4 = 256$ estados)
- Acciones: mover reina en columna
  - » desplazar la reina que menos amenazas tenga
- Objetivo: que no existan amenazas
- Evaluación: *h(n)* = nº de amenazas

h = 5 h = 2 h = 0

Imagen tomada de Russell & Norvig

h(n)

4	2	4	3
4		3	
	3		4
3	4	2	4

h(n)

2		2	2
3	5	2	
	3		3
1	4	0	2