Tema 2

“Búsqueda no-informada”

Año académico 2019/20

Profesores:
Alberto Fernández Gil, Holger Billhardt

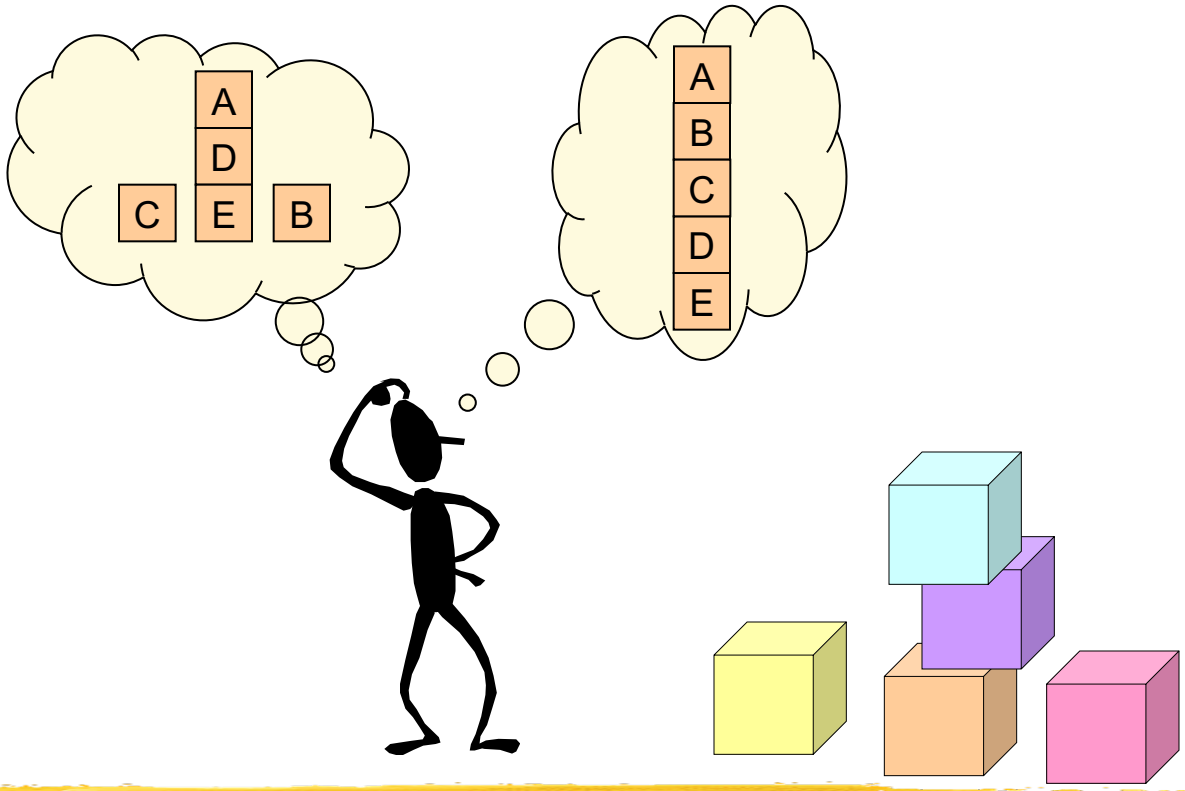
Entornos simples

Características de entornos simples (*problemas bien definidos*)

- discreto:
  - se puede concebir el mundo en estados
  - en cada estado hay un conjunto finito de percepciones y acciones
- accesible: el agente puede acceder a las características relevantes del entorno
  - puede determinar el estado actual del mundo
  - puede determinar el estado del mundo que le gustaría alcanzar
- estático y determinista: no hay presión temporal ni incertidumbre
  - el mundo cambia sólo cuando el agente actúa
  - el resultado de cada acción está totalmente definido y previsible

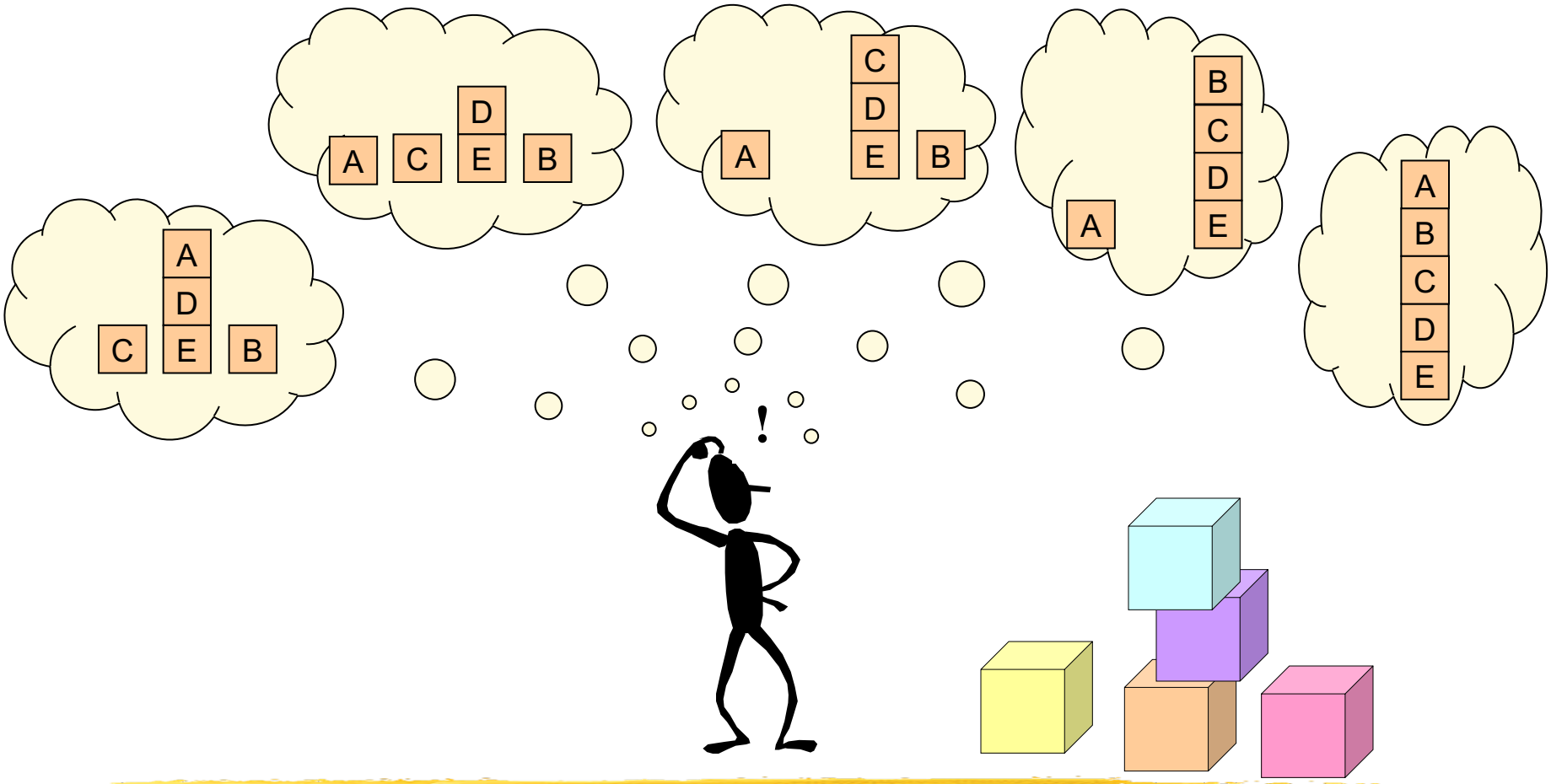
Agentes basados en búsqueda

- Mantienen un modelo *simbólico* del entorno
- y desean modificar el estado del entorno de acuerdo con sus *objetivos*.



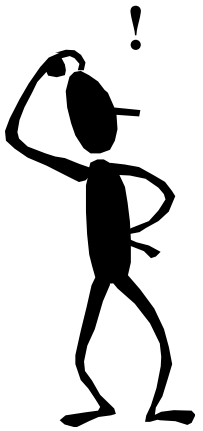
Agentes basados en búsqueda

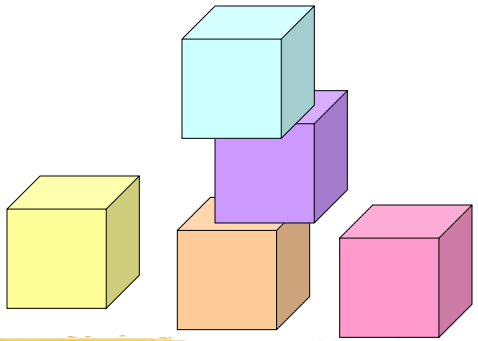
- Con tal fin, *anticipan* los efectos esperados de sus acciones sobre el modelo,
- generando *planes de actuación* en el modelo…



Agentes basados en búsqueda

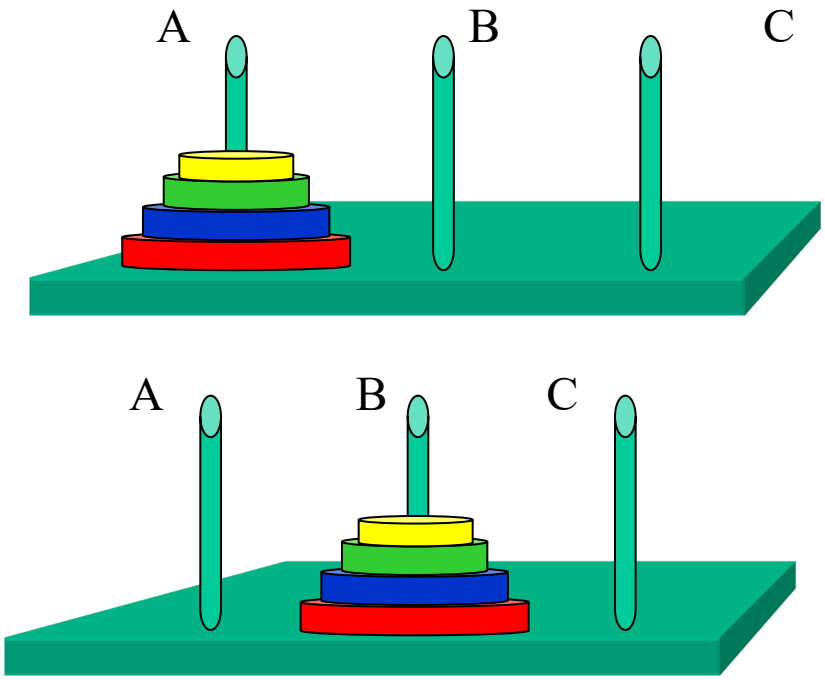
– … antes de ejecutar las acciones del plan en el entorno real





¿Por qué búsqueda?

Ejemplo: Torres de Hanoi



Objetivo:

- Trasladar los discos de la aguja *A* a *B* en el mismo orden

Restricción:

- un disco mayor nunca debe reposar sobre uno de menor tamaño

¿Cómo escribir el programa de agente correspondiente?

Solución 1: tablas de actuación

Tablas de actuación específicos del problema:

- para cada situación hay una entrada en una tabla de actuación; dicha entrada compila la secuencia de acciones a emprender:

  cuatro discos en A $\Rightarrow$

  disco 1 de A a C / disco 2 de A a B / disco 1 de C a B / disco 3 de A a C /
  disco 1 de B a A / disco 2 de B a C /disco 1 de A a C / disco 4 de A a B /
  disco 1 de C a B / disco 2 de C a A /disco 1 de B a A/ disco 3 de C a B /
  disco 1 de A a C / disco 2 de A a B/ disco 1 de C a B

- mejorar la flexibilidad: *aprender* nuevas entradas
- problema: limitaciones de memoria

Solución 2: algoritmo

Algoritmos específicos del problema:

- el diseñador del agente conoce un método para resolver el problema
- codifica este método en un algoritmo *particular para el problema*
- mejorar la flexibilidad: *parametrizar* el algoritmo
- problema: el diseñador ha de *anticipar* todos los escenarios posibles
- los entornos reales suelen ser demasiado complejos como para anticipar todas las posibilidades

```
PROCEDURE MoverDiscos(n:integer;
                      origen,destino,auxiliar:char);
{ Pre: n > 0
  Post: output = [movimientos para pasar n
                  discos de la aguja origen
                   a la aguja destino] }
BEGIN
  IF n = 0 THEN   {Caso base}
    writeln
  ELSE BEGIN      {Caso recurrente}
    MoverDiscos(n-1,origen,auxiliar,destino);
    write('Pasar disco',n,'de',origen,'a',destino);
    MoverDiscos(n-1,auxiliar,destino,origen)
  END; {fin ELSE}
END; {fin MoverDiscos}
```

Solución 3: búsqueda

Métodos independientes del problema :

- modelo simbólico del problema:
  - "inicialmente todos los discos reposan en *A* y su tamaño decrece de abajo hasta arriba"
  - "queremos que todos los discos estén en *C* en el mismo orden"
  - "podemos mover un disco *I* a la aguja *X*, si no hay otro disco por encima de *I* y, si actualmente hay discos en *X*, entonces dichos discos han de ser más grandes que *I*"
  - "cuanto menos movimientos de discos hagamos mejor"
- algoritmo de búsqueda *genérico*:
  - genera una solución a cualquier problema representado mediante el modelo simbólico
- mayor flexibilidad:
  - el diseñador no necesita conocer la solución de antemano
  - es más fácil adaptar el método a nuevas características del problema

Ejemplo: El mundo de los bloques

Situaciones:

- *n* bloques en una mesa de longitud ilimitada

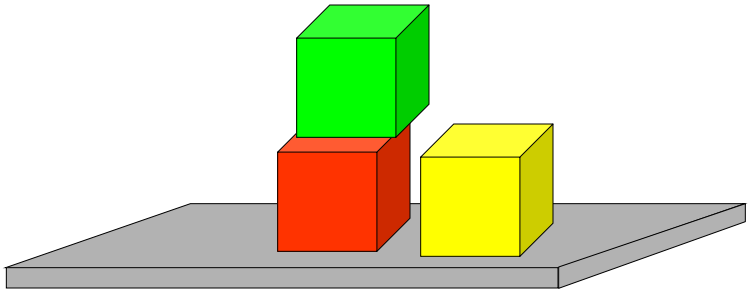
Acciones:

- *apilar*(*X*,*Y*): poner *X* encima de *Y*
  - Prec.: bloques *X* e *Y* están libres
  - Efecto: bloque *X* está encima de *Y*
- *quitar*(*Y*): poner *Y* en la mesa
  - Prec.: bloque *Y* está libre
  - Efecto: bloque *Y* está en la mesa

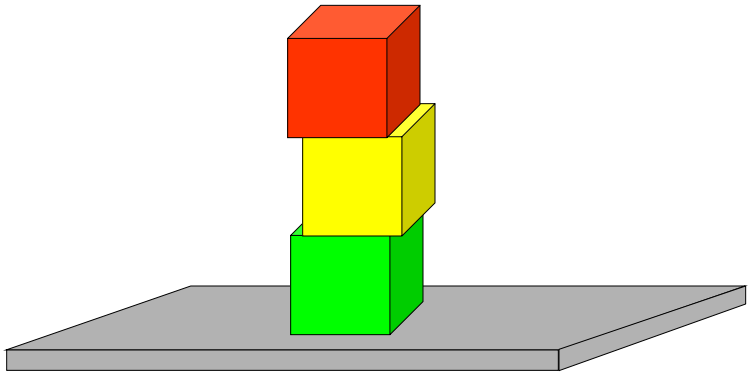
Actitud del agente:

- Objetivo: cierta configuración de bloques
- Sólo la posición *vertical* es relevante
- El coste de cada acción es uno

Situación presente



Objetivo



Búsqueda en espacios de estados

Espacio de estados: modelo del mundo representado por un grafo

mundo	modelo	representación
situacion	estado	nodo
situación presente	estado inicial	nodo inicial
acción y sus efectos	operador	arco
secuencia de acciones	plan	camino

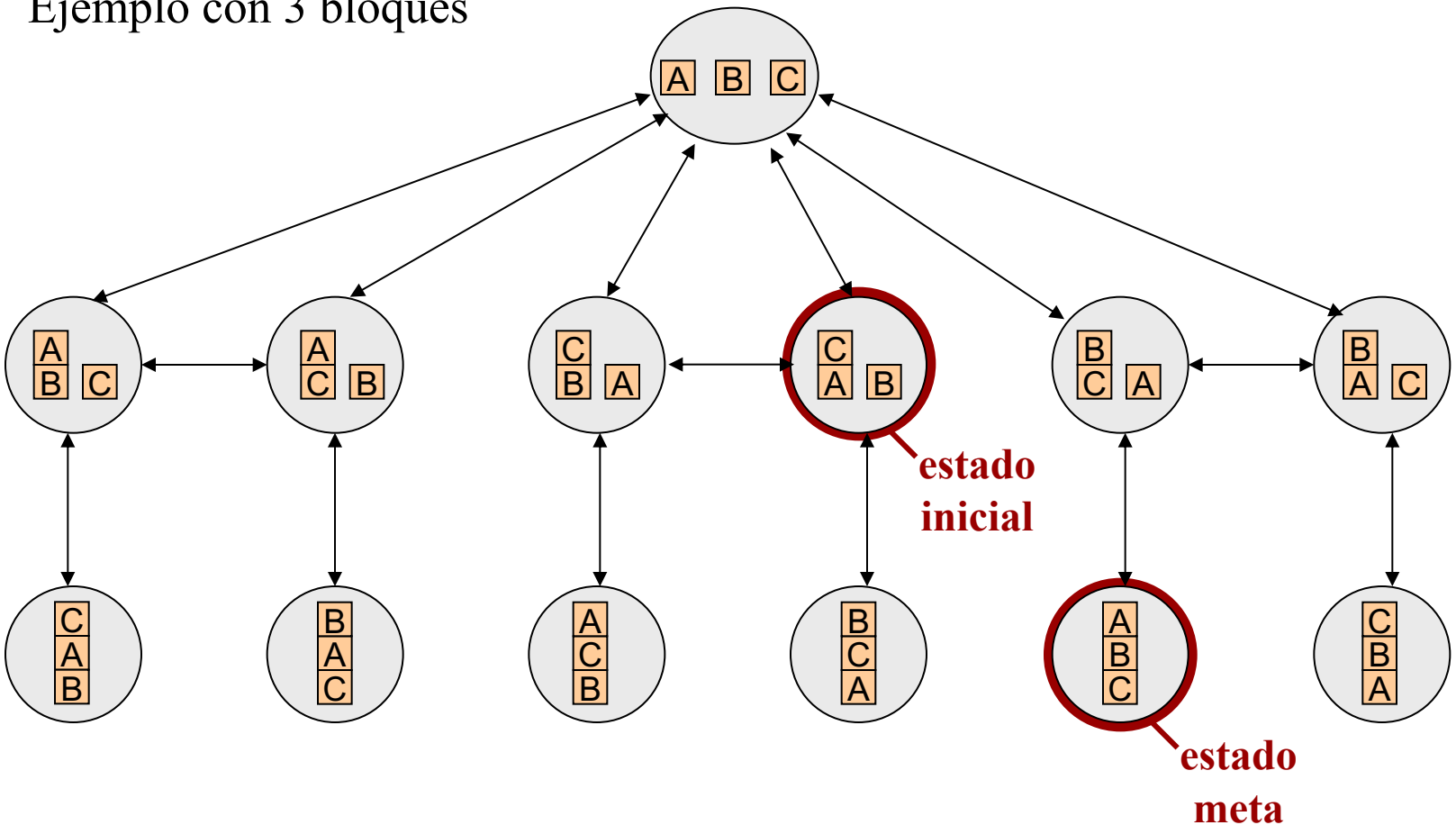
Problema de búsqueda: espacio de estados + actitud del agente

actitud	representación
estados meta	conjunto de nodos
eficiencia de un plan	coste de un camino

Objetivo: encontrar el plan más eficiente que lleve del estado inicial a un estado meta

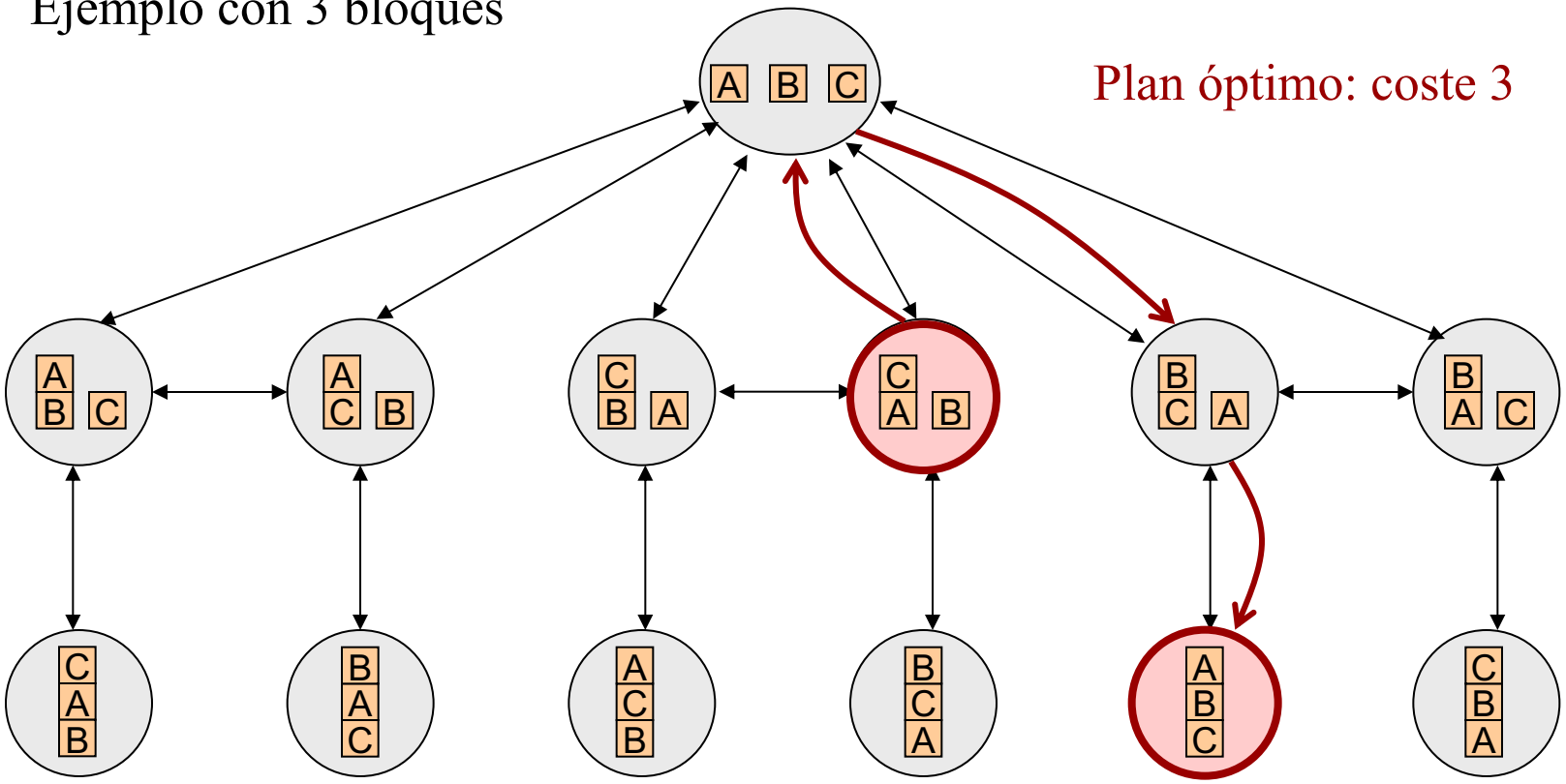
Mundo de los bloques: espacio de estados

Ejemplo con 3 bloques



Representación del problema de los bloques

Ejemplo con 3 bloques



Plan óptimo: coste 3

Agentes especializados

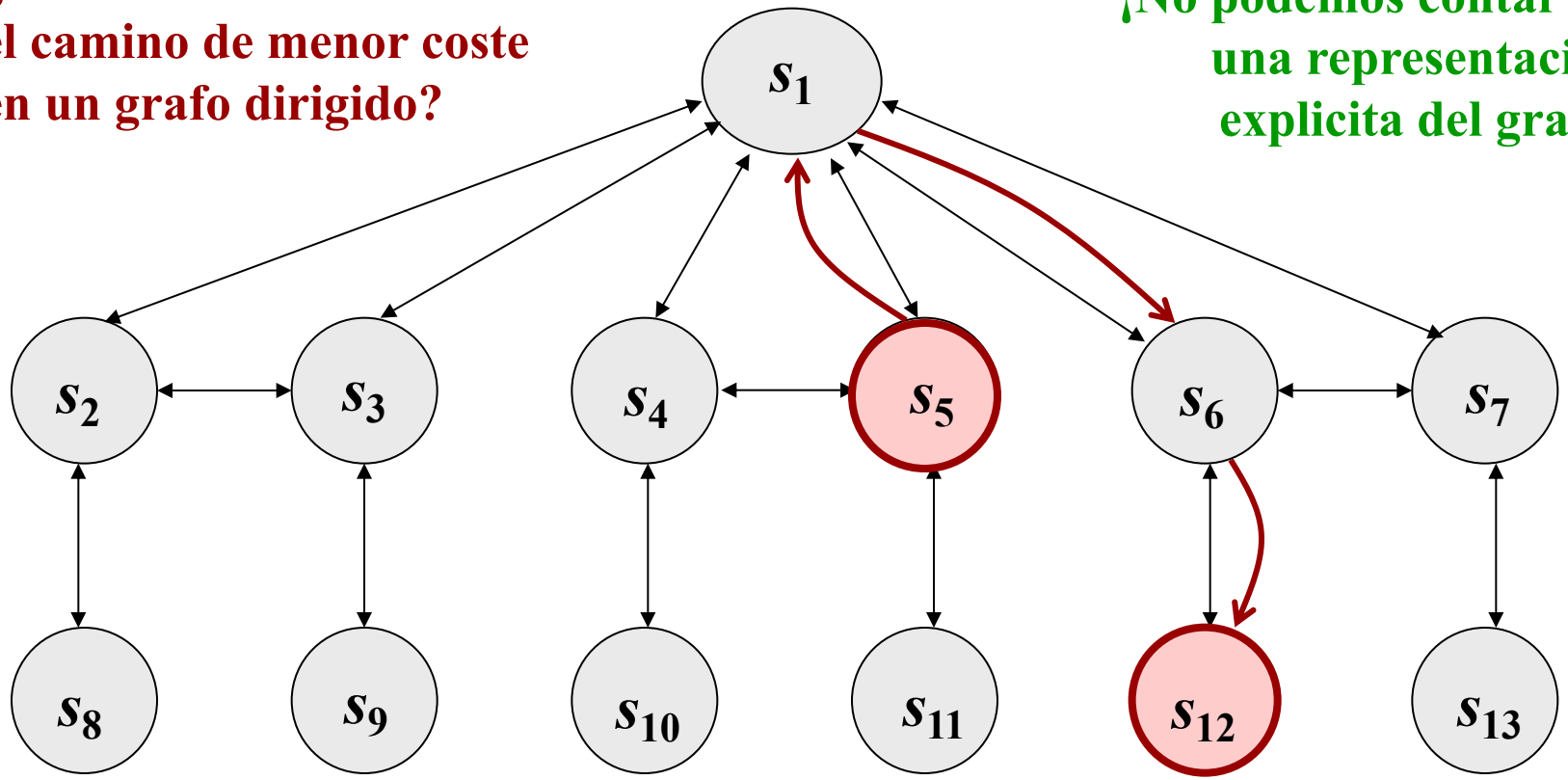
Tareas:

1. Definir el *modelo* y generar los *objetivos*
   - Conocimiento a priori del agente facilitado por el diseñador
2. Generar el espacio de estados
3. *Percibir* y *clasificar* la situación presente:
   - Agentes hardware: Visión artificial, Robótica
   - Agentes software: Algoritmos específicos del dominio
4. *Buscar* un *plan* de actuación
5. *Ejecutar* el plan de actuación
   - Agentes hardware: Robótica
   - Agentes software: Algoritmos específicos del dominio

Búsqueda en espacio de estados

¿Se trata "sólo" de encontrar el camino de menor coste en un grafo dirigido?

¡No podemos contar un una representación explicita del grafo!



Conocimientos “a priori” de nuestro agente

- Representación *implícita* del problema de búsqueda
- Conocimientos mínimos a priori de un agente:
  - s_0 — Estado *inicial*
  - *expandir*: $s \mapsto \{s_{i_1}, ..., s_{i_n}\}$ — Conjunto finito de *sucesores* de un estado
  - *meta?*: s $\mapsto$ verdad | falso — Prueba de *éxito* en un estado
  - c: $(s_i, s_j) \mapsto v$, $v \in \mathbb{R}$ — *Coste* de un operador
  - $c(s_{i_1}\ s_{i_2} \ldots s_{i_n}) = \sum_{k=1}^{n-1} c(s_{i_k}, s_{i_{k+1}})$ — *Coste* de un plan

Método de búsqueda

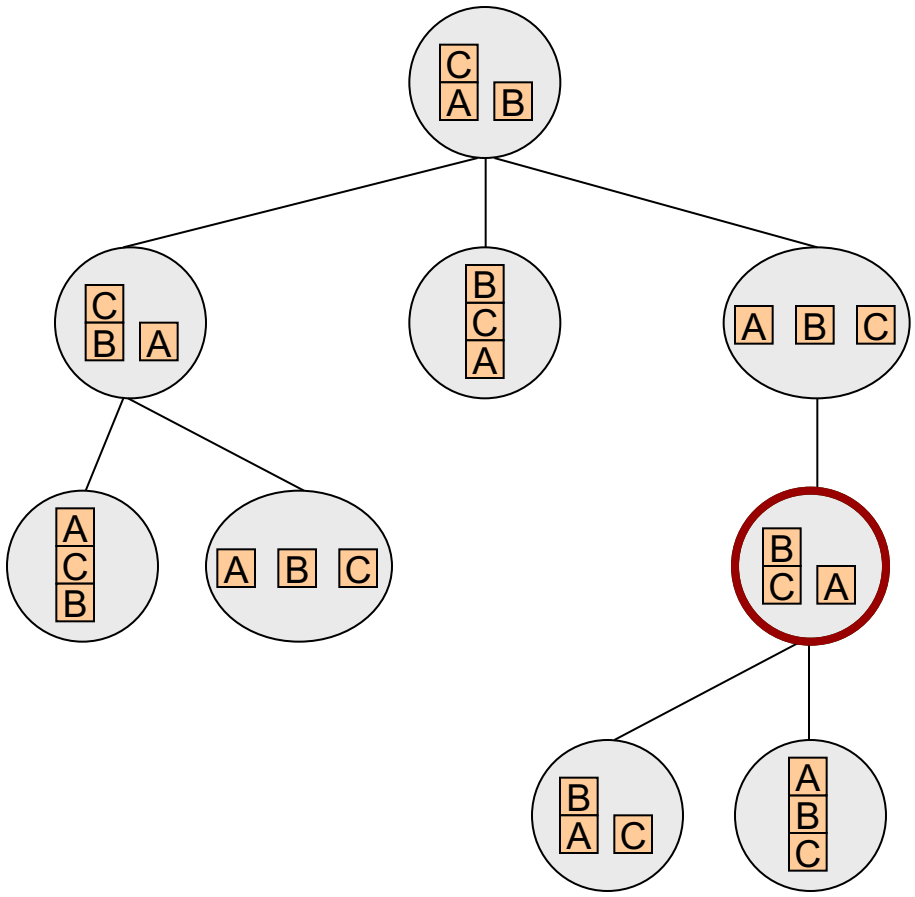
Método de búsqueda:

- *estrategia* para explorar el espacio de estados
- en cada paso se *expande* un estado
- se desarrolla sucesivamente un *árbol de búsqueda*

Método general de búsqueda:

1. seleccionar nodo hoja
2. comprobar si es nodo meta
3. *expandir* este nodo hoja

Arbol de búsqueda:



Algoritmo de búsqueda

Elementos del algoritmo

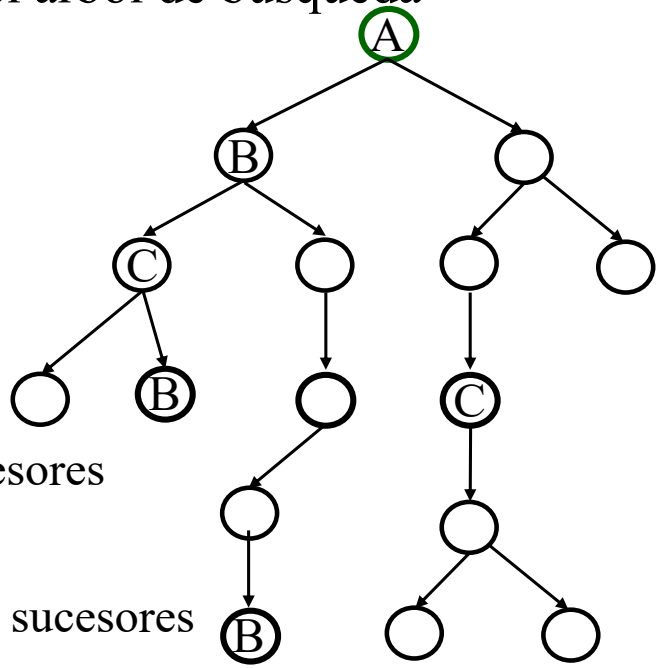
- el árbol se representa en base a un registro del tipo *nodo*
- *abierta* es una lista de nodos, con las hojas actuales del árbol
- *vacía?* determina si una lista es vacía
- *primero* quita el primer elemento de una lista
- *ordInsertar* añade un nodo a una lista, clasificado según una función de orden

```
{búsqueda general}
abierta ← s0
Repetir
    Si vacía?(abierta) entonces
        devolver(negativo)
    nodo ← primero(abierta)
    Si meta?(nodo) entonces
        devolver(nodo)
    sucesores ← expandir(nodo)
    Para cada n∈sucesores hacer
        n.padre ← nodo
        ordInsertar(n,abierta,<orden>)
Fin {repetir}
```

Estados repetidos

Problema:

- el mismo estado puede repetirse varias veces en el árbol de búsqueda
- puede generarse el mismo subárbol varias veces



Soluciones:

- ignorarlo
- evitar *ciclos simples*:
  - no añadir el *padre* de un nodo al conjunto de sucesores
- evitar *ciclos generales*:
  - no añadir un *antecesor* de un nodo al conjunto de sucesores
- evitar *todos los estados repetidos*:
  - no añadir *ningún nodo existente* en el árbol al conjunto de sucesores

Clasificación de métodos

Características:

- *Completitud*: se encuentra una solución si existe
- *Optimalidad:* se encuentra la *mejor* solución si hay varias
- *Complejidad en tiempo*: ¿cuánto se tarda en encontrar la solución?
- *Complejidad en espacio*: ¿cuánta memoria se utiliza en la búsqueda?

Tipos de métodos de búsqueda:

- *No informados* (Tema 2)
  - utilizan sólo los conocimientos mínimos (*búsqueda en amplitud*, *búsqueda de coste uniforme*, …)
- *Heurísticos* (Temas 3-4)
  - además utilizan información *aproximada*, y *específica* del problema, para guiar la búsqueda (*Algoritmo A** y extensiones, *búsqueda multiagente*)

Ejercicio

Problema de búsqueda / conocimiento del agente:

En una mesa se encuentran dos jarras, una con una capacidad de 3 litros (llamada *Tres*), y la otra con una capacidad de 4 litros (llamada *Cuatro*). Inicialmente, *Tres* y *Cuatro* están vacías. Cualquiera de ellas puede llenarse con el agua de un grifo *G*. Asimismo, el contenido tanto de *Tres* como de *Cuatro* puede vaciarse en una pila *P*. Es posible echar todo el agua de una jarra a la otra. No se dispone de dispositivos de medición adicionales. Se trata de encontrar una secuencia de operadores que deje exactamente dos litros de agua en *Cuatro*.

a) Modele este problema como un problema de *búsqueda*. Con tal fin, defina el estado inicial, el conjunto de estados meta, los operadores (especificando sus precondiciones y postcondiciones), así como el coste de cada operador.

b) Caracterice el *conocimiento a priori* del agente de resolución del problema correspondiente? Facilite ejemplos de los resultados de la función *expandir.*

c) Encuentre una solución al problema.

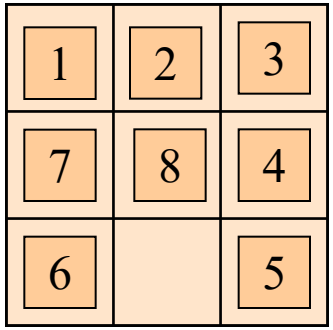
Ejemplo base: 8-Puzzle

Problema:

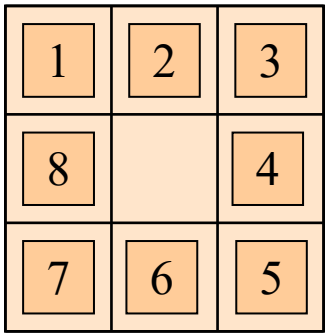
- **Estados**:
  - posición de las piezas
- **Acciones:**
  - mover pieza adyacente a la posición del "hueco"
  - de 2 a 4 operadores aplicables, según el estado
- **Objetivo:**
  - plan (óptimo) que lleva de un estado inicial al estado meta
- **Coste:**
  - La aplicación de cada operador vale *una* unidad

Instanciación:

Estado inicial



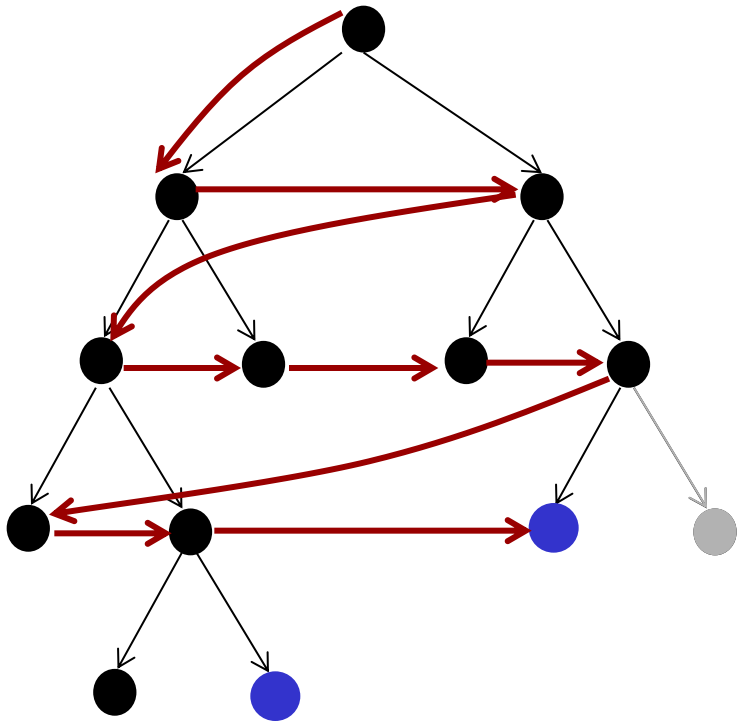
Estado meta



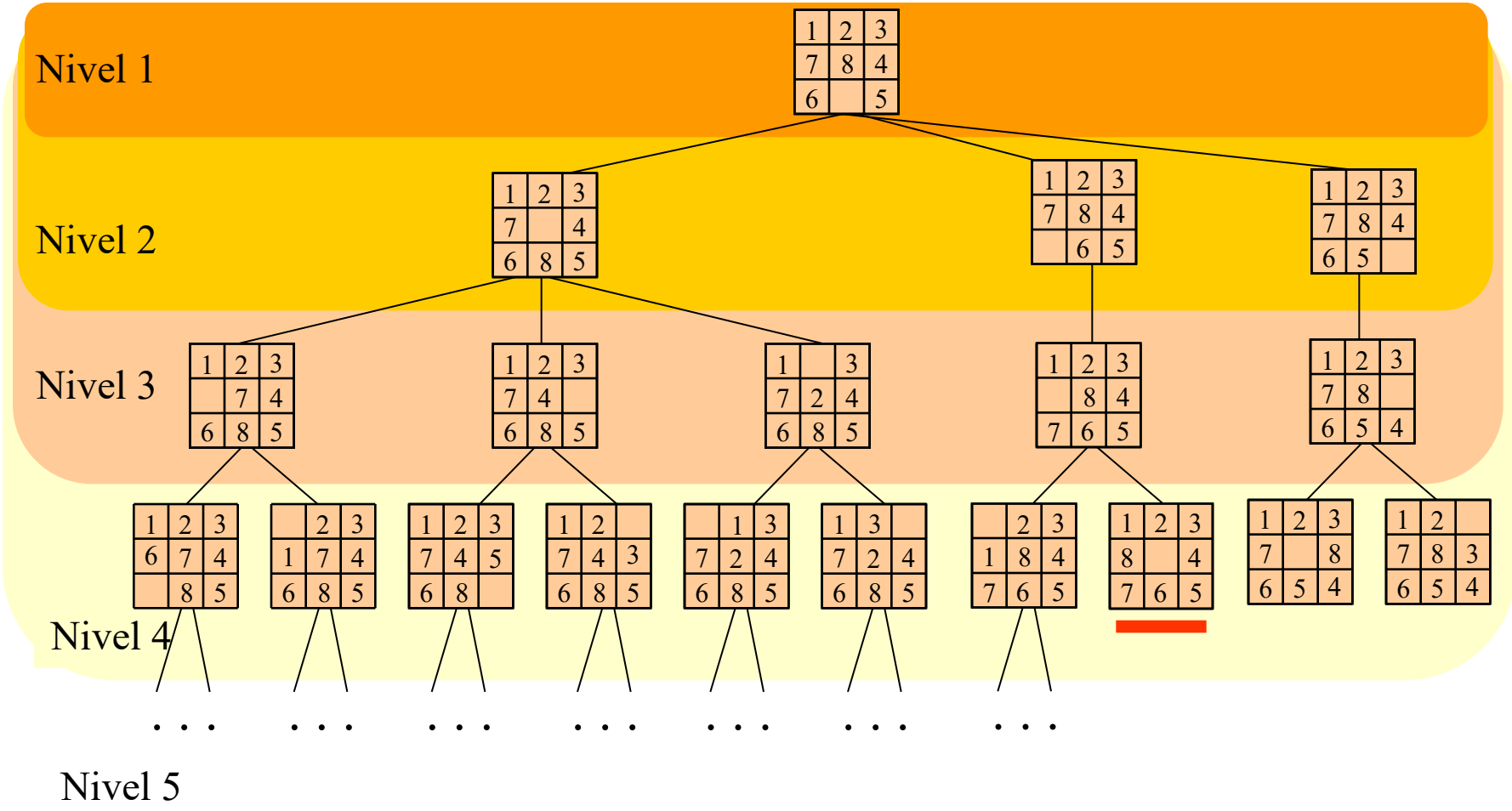
Búsqueda en amplitud

Búsqueda en amplitud:

- inglés: breadth first search
- Estrategia:
  - generar el árbol por niveles de profundidad
  - expandir todos los nodos de nivel *i*, antes de expandir nodos de nivel *i*+1
- Resultado:
  - considera primero todos los caminos de longitud 1, después los caminos de longitud 2, etc.
  - Se encuentra el estado meta de *menor profundidad*



Árbol de búsqueda en amplitud (evitando ciclos simples)



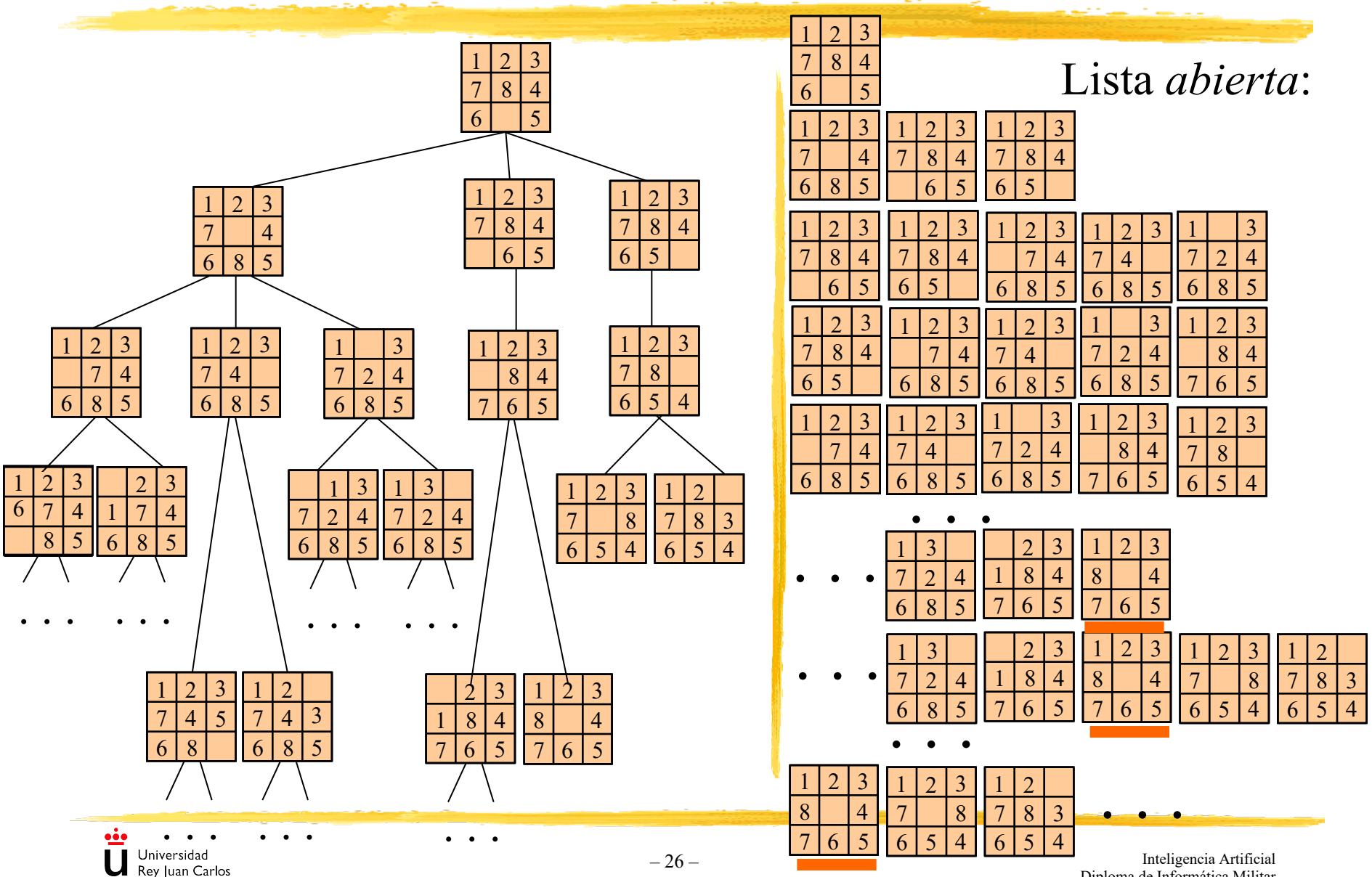
Algoritmo para búsqueda en amplitud

Algoritmo:

- usar el algoritmo general de búsqueda
- añadir nuevos sucesores al *final* de la lista *abierta*
- *abierta* funciona como *cola*
  - inserción al final
  - recuperación desde la cabeza
- estructura FIFO:
  - siempre expandir primero el nodo más antiguo (es decir: menos profundo)

{búsqueda en amplitud}
abierta ← s_0
Repetir
  Si *vacía?*(abierta) **entonces**
    devolver(negativo)
  nodo ← primero(abierta)
  Si *meta?*(nodo) **entonces**
    devolver(nodo)
  sucesores ← *expandir*(nodo)
  Para cada n∈ sucesores **hacer**
    n.padre ← nodo
    ordInsertar(n,abierta,*final*)
  Fin {repetir}

Árbol de búsqueda en amplitud



Complejidad

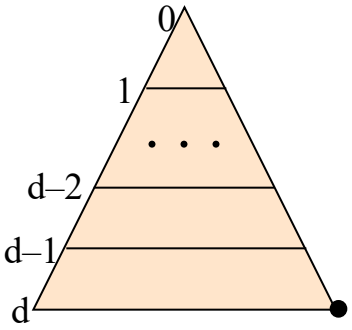
Complejidad en tiempo y espacio:

- proporcional al número de nodos expandidos

Suponemos que en el árbol de búsqueda

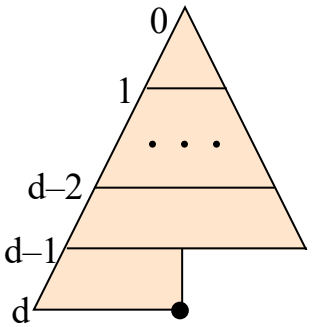
- el factor de ramificación es b
- el mejor nodo meta tiene profundidad d

Peor caso



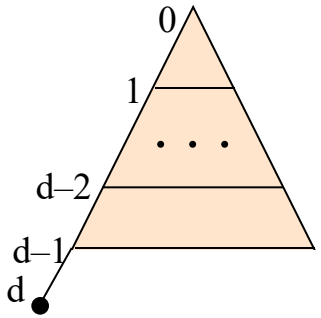
$1+b+\ldots+b^{d-1}+b^{d} \in O(b^{d})$

Caso medio



$1+b+\ldots+b^{d-1}+b^{d}/2 \in O(b^{d})$

Mejor caso



$1+b+\ldots+b^{d-1}+1 \in O(b^{d})$

Requerimientos de tiempo y memoria

Ejemplo: recursos requeridos por la búsqueda en amplitud en el *peor caso*

- factor de ramificación efectivo: 10
- tiempo: 1.000.000 nodos/segundo
- memoria: 1.000 bytes/nodo

d	nodos	tiempo	memoria
2	110	0,11 ms	107 KB
4	11.110	11 ms	10,6 MB
6	10^6	1,1 s	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 horas	10 TB
12	10^{12}	13 días	1.000 TB
14	10^{14}	3,5 años	99 PB
16	10^{16}	350 años	10.000 PB

Búsqueda en amplitud: análisis

Ventajas:

- completo:
  - siempre se encuentra un nodo meta si existe
- óptimo (para operadores de coste uno):
  - siempre se encuentra el nodo meta menos profundo

Problemas:

- complejidad
  - exponencial incluso en el mejor caso
  - los problemas de espacio son aún más graves que los problemas de tiempo

Ejercicio

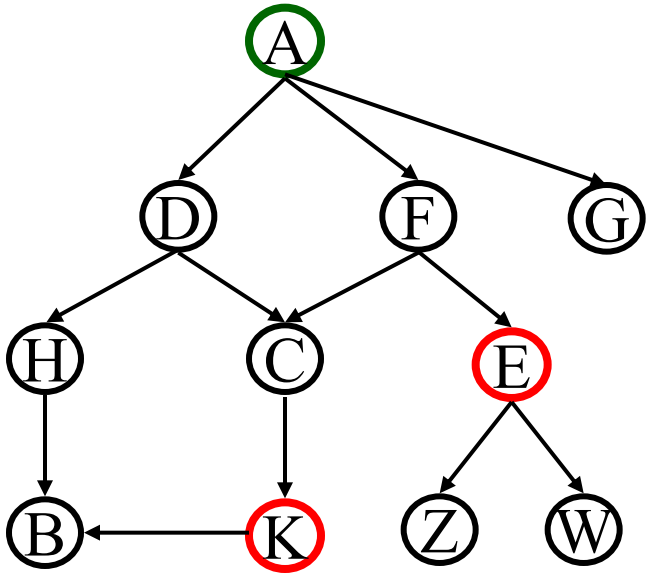
Búsqueda en amplitud:

El grafo que se muestra al lado determina un problema de búsqueda. Cada nodo representa un estado; los arcos modelan la aplicación de operadores. Suponga que *A* es el estado inicial y que *K* y *E* son estados meta

a) desarrolle el árbol de búsqueda que genera la búsqueda en amplitud. ¿Cuál de los nodos meta se encuentra primero?

b) indique el orden en que se expanden los nodos

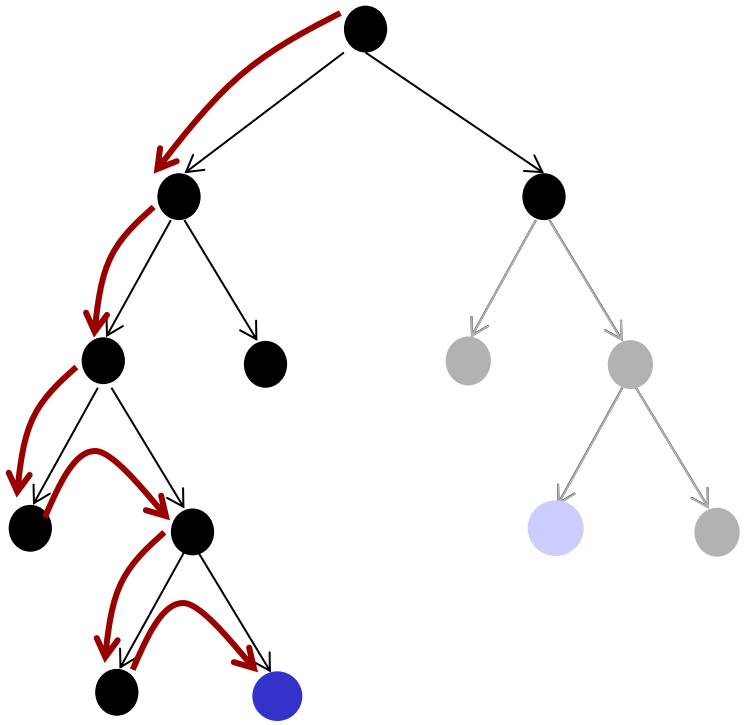
c) ponga el estado de la lista *abierta* en cada paso del algoritmo



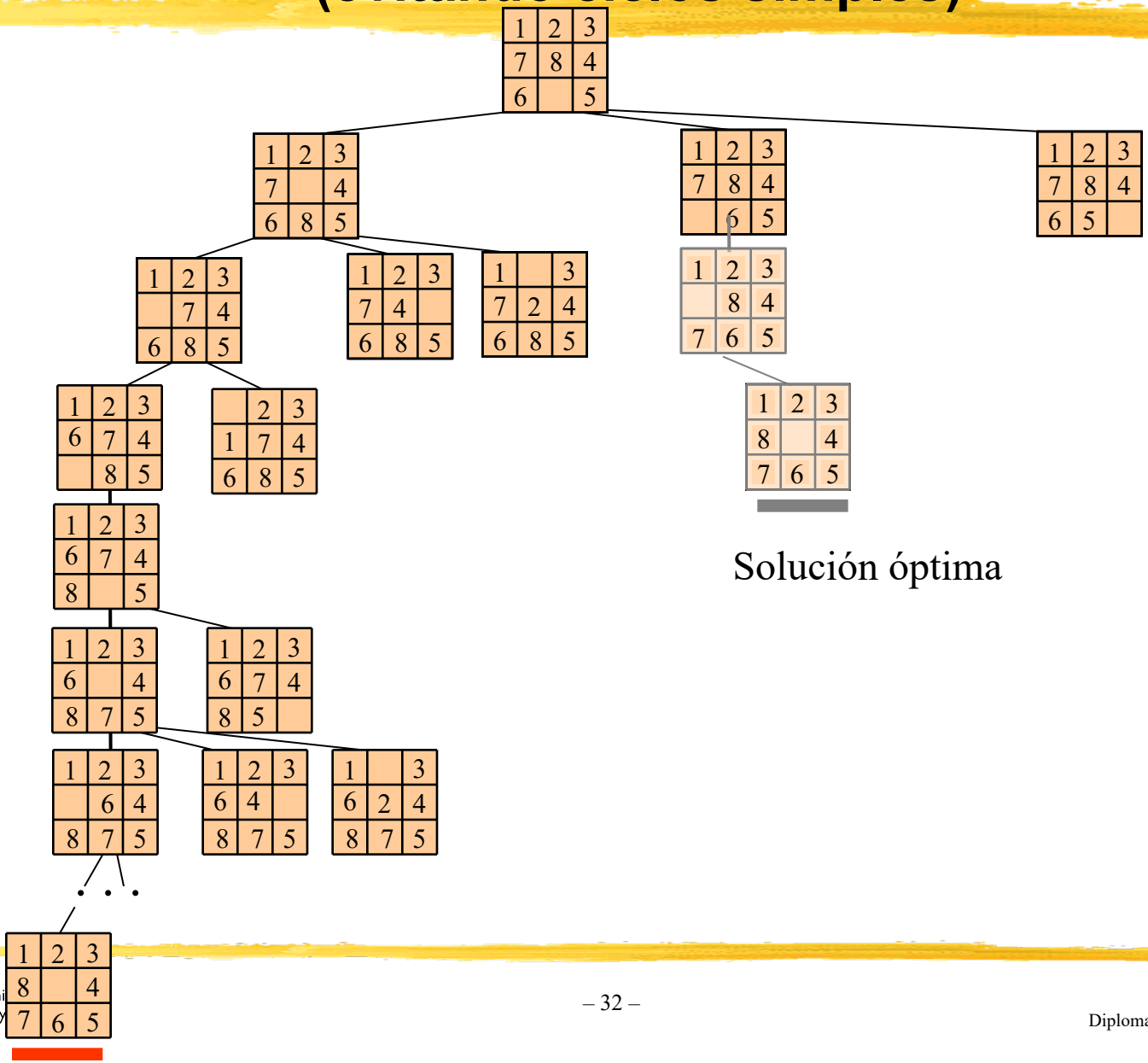
Búsqueda en profundidad

Búsqueda en profundidad:

- inglés: depth first search
- Estrategia:
  - expandir el árbol de "izquierda a derecha" (los nodos más profundos primero)
  - si se llega a un nodo sin sucesores, dar vuelta atrás y expandir el siguiente nodo más profundo
- Resultado:
  - el método va explorando un "camino actual"
  - no siempre se encuentra el nodo de *profundidad mínima*



Árbol de búsqueda en profundidad (evitando ciclos simples)



Solución óptima

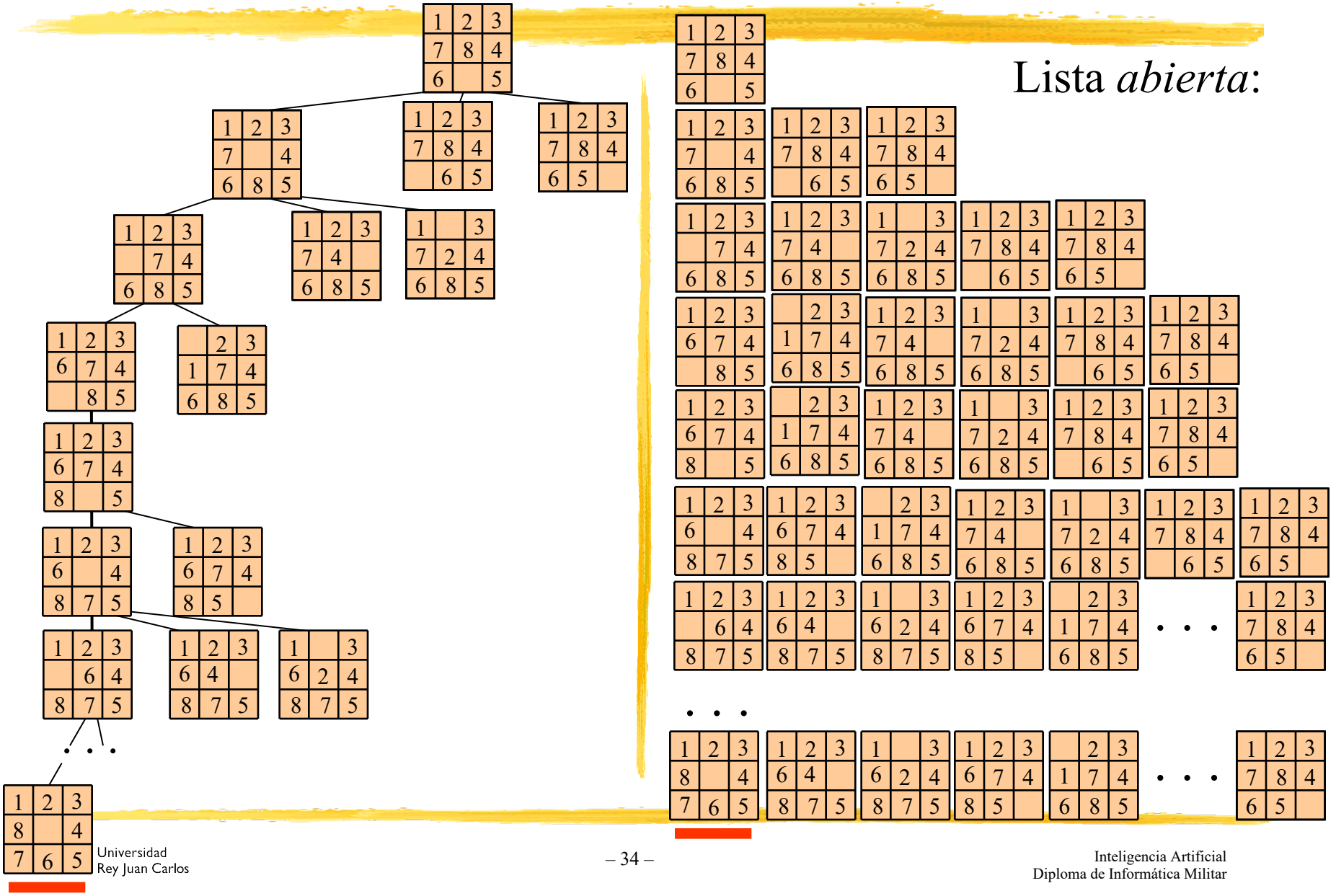
Búsqueda en profundidad

Algoritmo:

- usar el algoritmo general de búsqueda
- añadir nuevos sucesores *en la cabeza* de la lista *abierta*
- *abierta* funciona como *pila*
  - inserción en la cabeza de la lista
  - recuperación desde la cabeza
- estructura LIFO:
  - siempre expandir primero el nodo más reciente (es decir: el más profundo)
- al guardar *todos* los sucesores de un nodo expandido en *abierta*, se permite la "vuelta atrás"

{búsqueda en profundidad}
abierta ← s_0
Repetir
  Si *vacía?*(abierta) **entonces**
    devolver(negativo)
  nodo ← primero(abierta)
  Si *meta?*(nodo) **entonces**
    devolver(nodo)
  sucesores ← *expandir*(nodo)
  Para cada n∈ sucesores **hacer**
    n.padre ← nodo
    ordInsertar(n,abierta,*principio*)
  Fin {repetir}

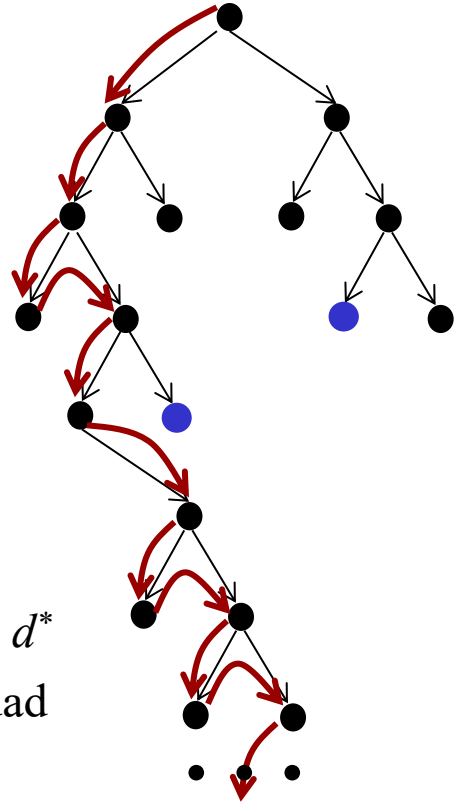
Árbol de búsqueda en profundidad



Límites de profundidad

Problema:

- si existen caminos infinitos sin nodo meta, es posible que la búsqueda en profundidad no termine
- la búsqueda en profundidad sólo es completa:
  - en el caso de árboles de búsqueda finitos
  - para espacios de búsqueda finitos; si se controla nodos repetidos en la misma rama del árbol

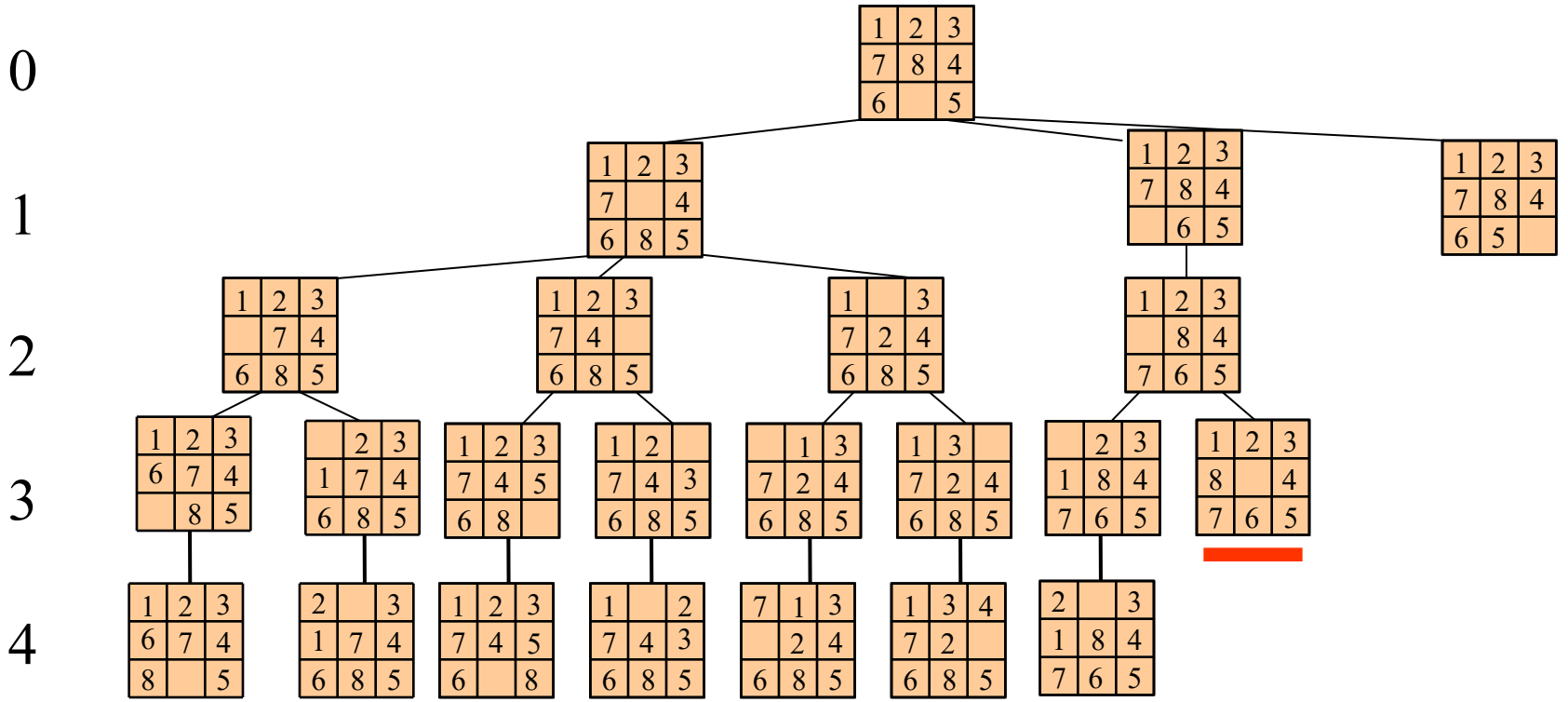


Solución:

- *búsqueda en profundidad limitada:*
  - inglés: depth limited search
  - búsqueda en profundidad con límite de profundidad d^*
  - antes de expandir un nodo, compruebe su profundidad
  - expandir sólo nodos con profundidad $d<d^*$
- incompleto si la profundidad del mejor nodo meta es mayor que d^*

Árbol de búsqueda en profundidad limitada (evitando ciclos simples)

limite d*=4



Búsqueda en profundidad limitada: complejidad

Complejidad en *tiempo*:

- proporcional al número de nodos expandidos
- factor de ramificación b / límite de profundidad d^* / nodo meta con profundidad $d \leq d^*$
- mejor caso:
  - $d \in O(d)$ (se expanden sólo los nodos del camino meta)
- peor caso:
  - $1+b+\ldots+b^{d^*-1} \in O(b^{d^*})$ (se expanden todos los nodos de prof. $< d^*$)

Complejidad en *espacio*:

- sólo los nodos del camino actual y sus "vecinos" (sucesores) necesitan almacenarse en la memoria
- *lineal* en la profundidad del árbol de búsqueda
  - mejor caso (si se encuentra la solución): $1+b \cdot d \in O(b \cdot d)$
  - peor caso (si no hay solución hasta profundidad d^*): $1+b \cdot d^* \in O(b \cdot d^*)$

Búsqueda en profundidad limitada: análisis

Ventajas:

- mejora significativa de la complejidad en espacio con respecto a la búsqueda en amplitud (lineal frente a exponencial):
- completo para límites de profundidad d^* adecuados

Problemas:

- *no* es *óptima*: el nodo meta que se encuentra puede *no* ser de profundidad mínima
- es común que unos límites "buenos" de profundidad sólo pueden establecerse cuando el problema ya haya sido resuelto
- en general, no se puede asegurar que la profundidad d de un nodo meta sea $d \leq d^*$, es decir no se puede garantizar la completitud.

Ejercicio

Búsqueda en profundidad:

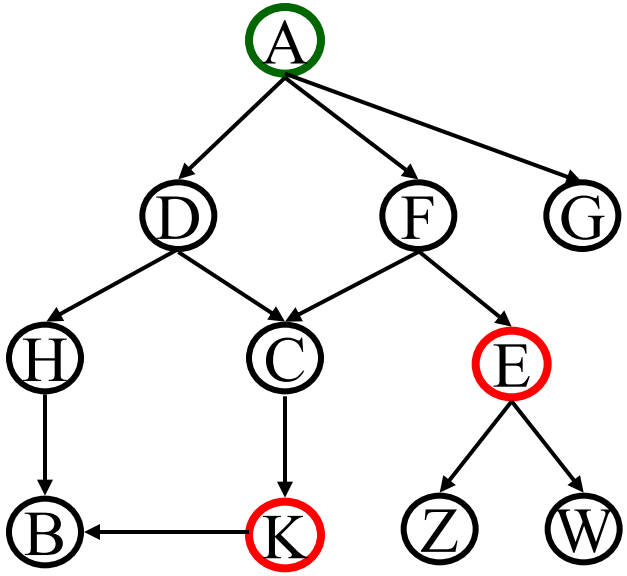
El grafo que se muestra al lado determina un problema de búsqueda. Cada nodo representa un estado; los arcos modelan la aplicación de operadores. Suponga que *A* es el estado inicial y que *K* y *E* son estados meta

a) desarrolle el árbol de búsqueda que genera la búsqueda en profundidad. ¿Cuál de los nodos meta se encuentra primero?

b) indique el orden en que se expanden los nodos

c) ponga el estado de la lista *abierta* en cada paso del algoritmo

d) ¿cómo cambiaría el proceso de búsqueda si aplicamos límites de profundidad, p.e. d^*=2?



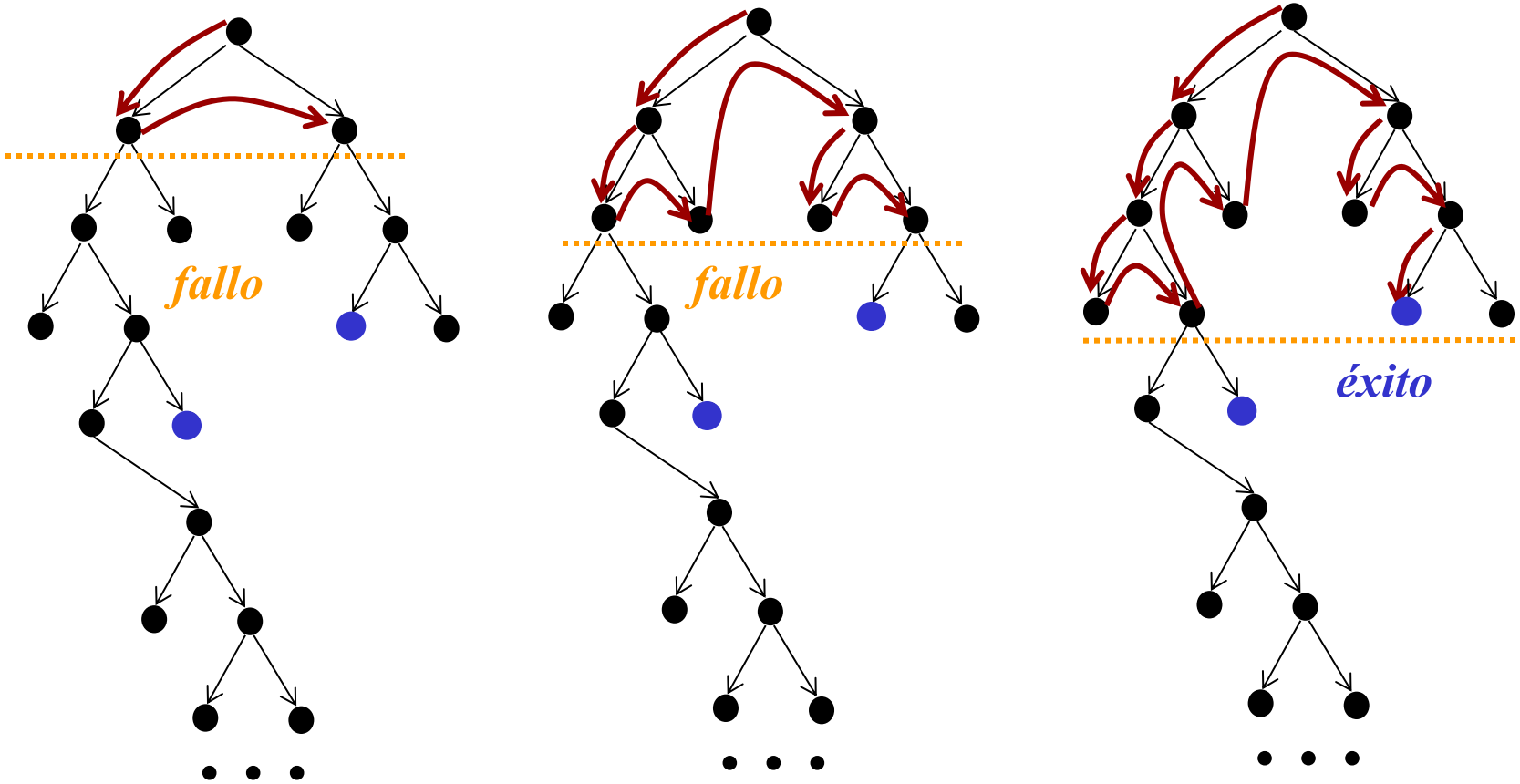
Búsqueda de profundización iterativa

- Inglés: iterative deepening search
- Idea:
  - esquivar el problema de elegir d^*, al probar *todos* los posibles límites de profundidad
- Estrategia:
  - enumerar todos los límites de profundidad d', empezando por 0
  - realizar búsqueda de profundidad limitada hasta d'
- Algoritmo:

{búsqueda de profundización iterativa}
abierta ← s_0
desde d´ ← 0 **hasta** ∝ **hacer**
    si *búsqueda-en-prof-limitada(problema, d´)* = éxito **entonces**
        devolver(nodo-meta)
    fin {desde}

Búsqueda de profundización iterativa

límite $d^*=1$ límite $d^*=2$ límite $d^*=3$



Búsqueda de profundización iterativa: análisis

Completo y óptimo

Complejidad en espacio:

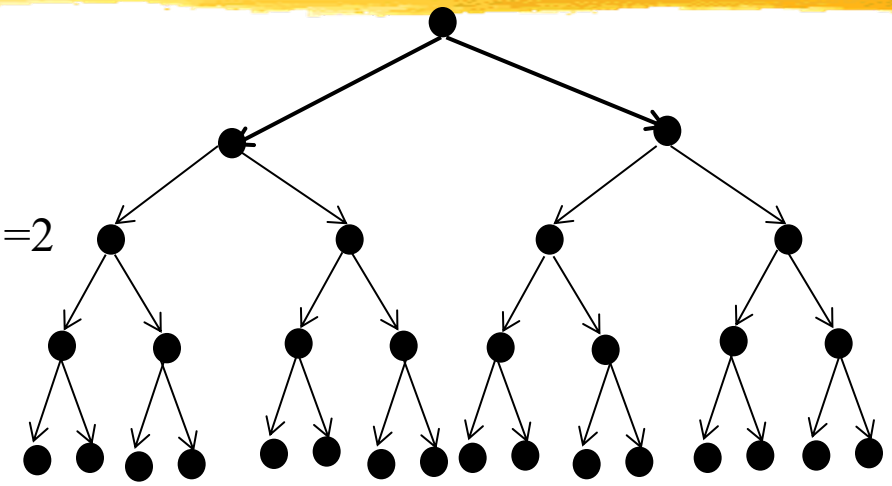
- igual que la búsqueda en profundidad: sólo se almacenan los nodos vecinos del camino actual
- lineal en la profundidad del árbol de búsqueda: todos los casos $O(b \cdot d)$

Complejidad en tiempo:

- Del mismo orden que en la búsqueda en amplitud/profundidad: en el peor caso $O(b^d)$
- normalmente el coste adicional es relativamente pequeño
- argumento intuitivo:
  - suponga un árbol de búsqueda de profundidad d
  - los nodos interiores (prof. $<d$) se expanden varias veces
  - los nodos hoja (prof. $= d$) se expanden sólo una vez
  - en un árbol de búsqueda grande "casi todos" los nodos son hojas

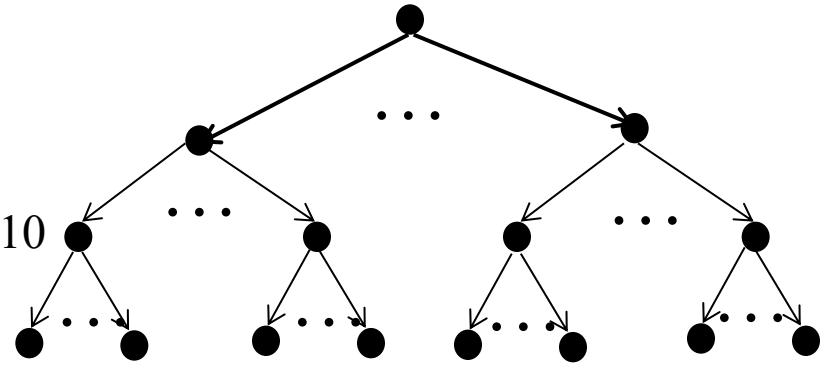
Búsqueda de prof. iterativa: complejidad en tiempo

factor de ramificación =2



1 nodo
2 nodos
4 nodos
8 nodos
15 nodos
16 nodos

factor de ramificación =10

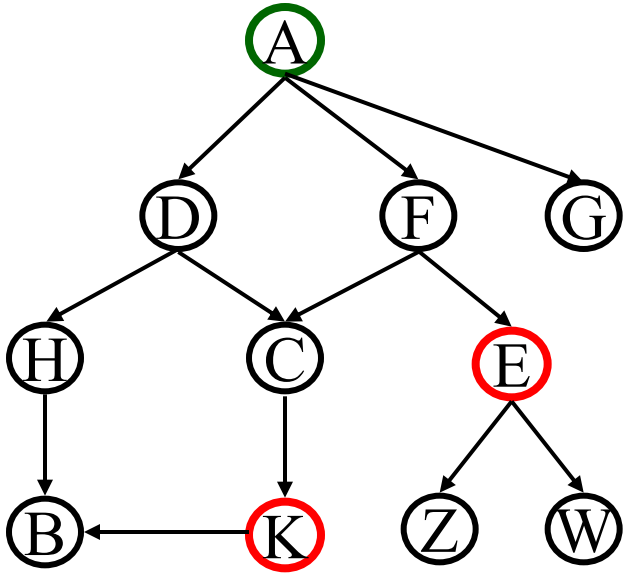


1 nodo
10 nodos
100 nodos
111 nodos
1000 nodos

Ejercicio

Búsqueda de profundización iterativa:

El grafo que se muestra al lado determina un problema de búsqueda. Cada nodo representa un estado; los arcos modelan la aplicación de operadores. Suponga que *A* es el estado inicial y que *K* y *E* son estados meta



a) desarrolle la secuencia de árboles de búsqueda generadas por la búsqueda de profundización iterativa, indicando para cada uno de ellos el orden en que se expanden los nodos

b) ¿Cuál de los nodos meta se encuentra primero?

Búsqueda no informada: resultados

Resultados del peor caso:

- factor de ramificación b / profundidad de la mejor solución d / límite de profundidad d^*

	búsqueda en amplitud	búsqueda en prof. limitada	búsqueda de prof. iterativa
complejidad en tiempo	$O(b^d)$	$O(b^{d^*})$	$O(b^d)$ *[> que bus. en amp.]*
complejidad en espacio	$O(b^d)$	$O(b \cdot d^*)$	$O(b \cdot d)$
óptimo? (op. de coste 1)	*sí*	*no*	*sí*
completo?	*sí*	*sólo si* $d \leq d^*$	*sí*

Método no informado preferido

Problema de encontrar rutas

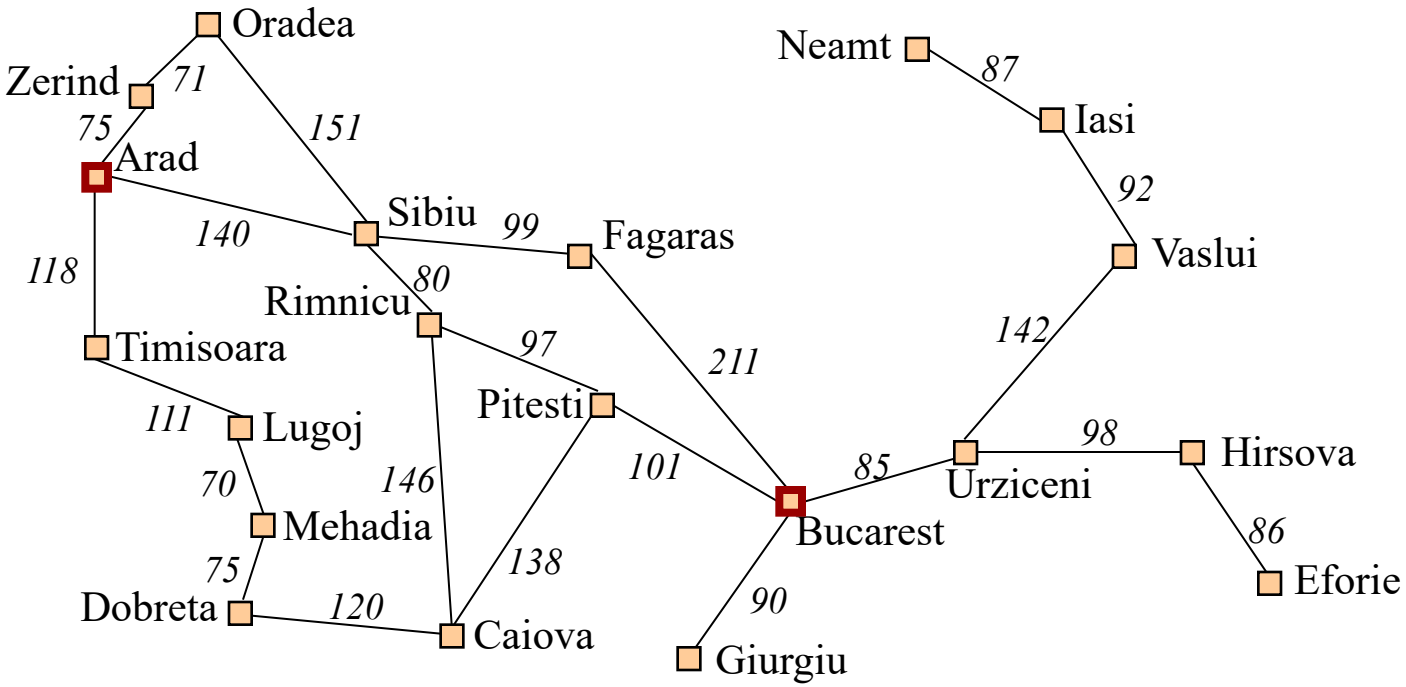
Estado: estancia en una ciudad

Acciones: ir a una ciudad vecina

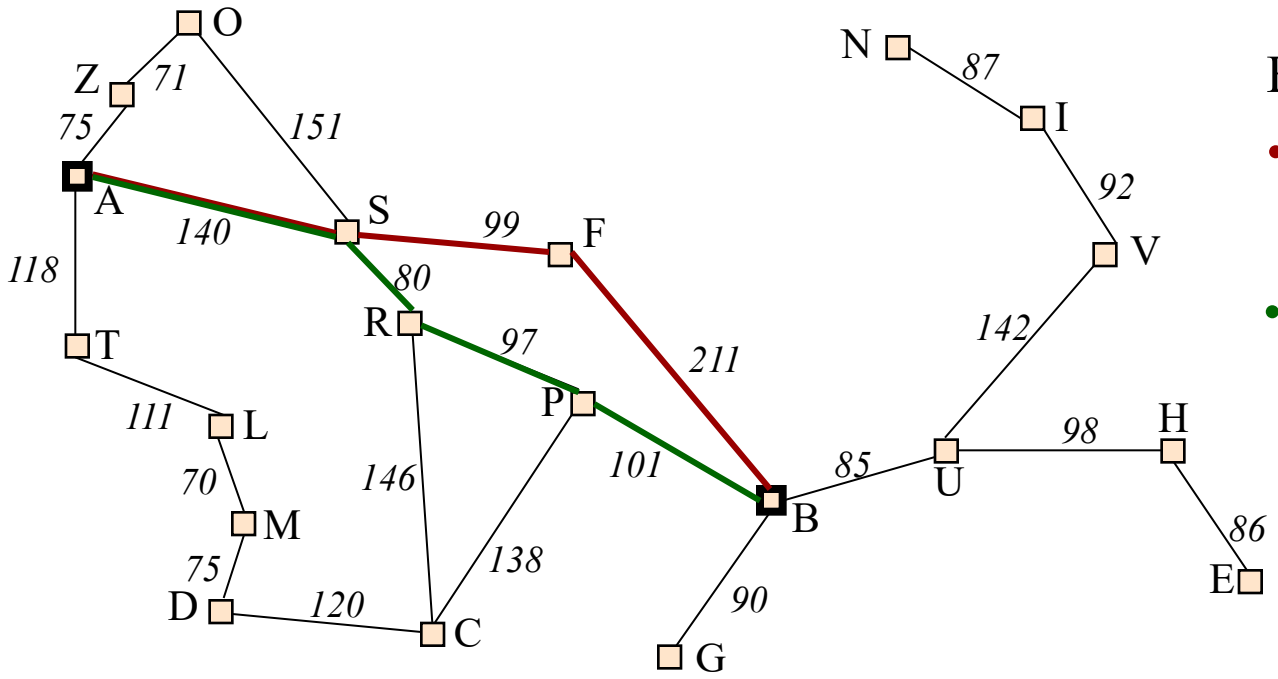
Objetivo: encontrar el camino mínimo

Coste de un plan: suma de distancias entre las ciudades visitadas

Coste de un operador: distancia por carretera a la ciudad vecina



Problema de encontrar rutas



Ejemplo:

- p_1 = A-S-F-B
  c(p_1) = 450
- p_2 = A-S-R-P-B
  c(p_2) = 418

Problema:

- los métodos de búsqueda vistos hasta ahora no tienen en cuenta los costes de las acciones (encuentran el nodo meta de menor profundidad)
- prof.(B_{p1}) = 3 < 4 = prof.(B_{p2}) / c($p1$) = 450 > 418=c($p2$)

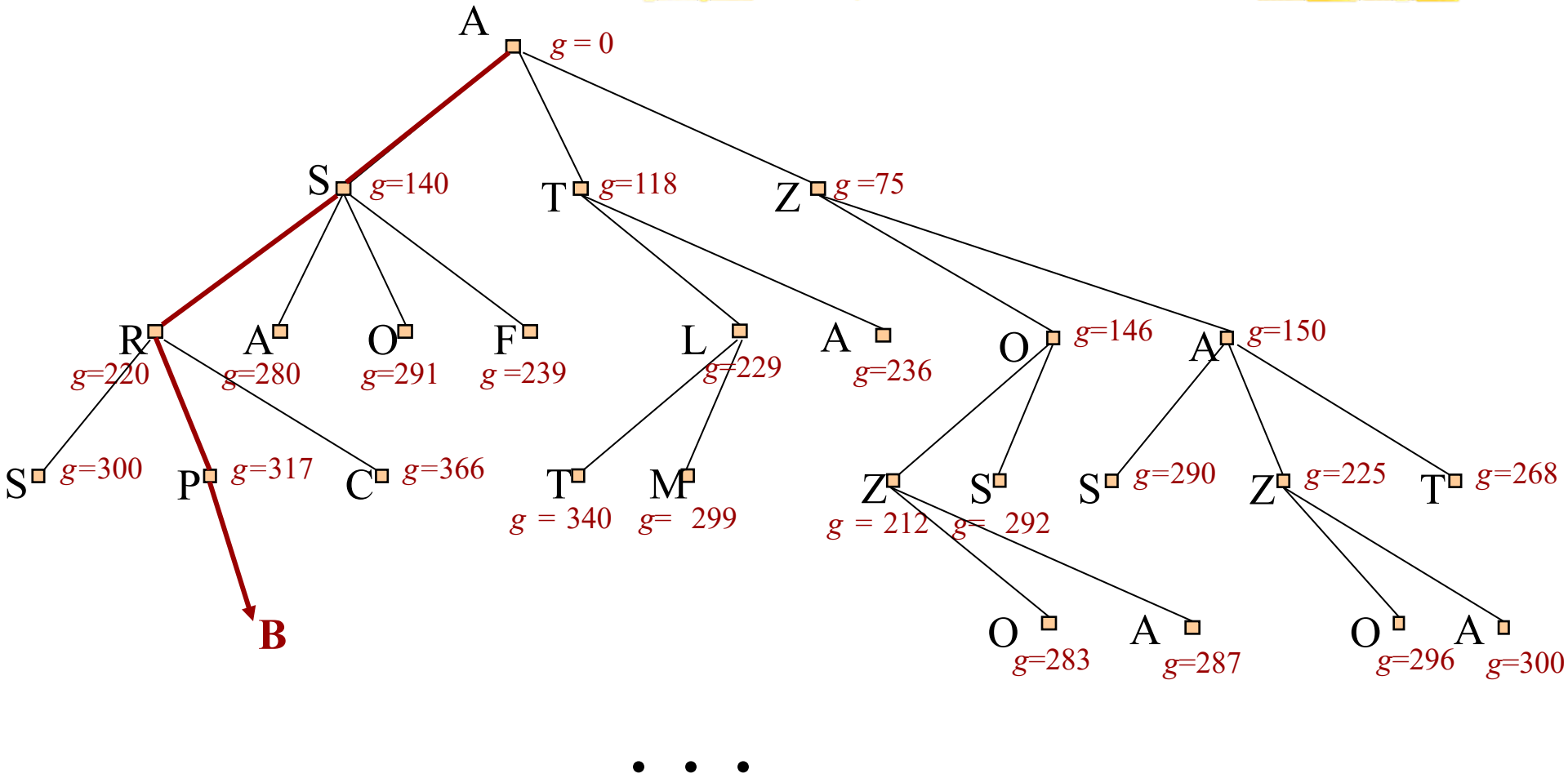
Búsqueda de coste uniforme

Búsqueda de coste uniforme:

- Inglés: uniform cost search
- Idea:
  - guiar la búsqueda por el *coste* de los operadores
- Método:
  - $g(n)$: coste mínimo para llegar del nodo inicial al nodo n
  - expandir siempre el nodo de menor coste g primero
- Algoritmo:
  - almacenar cada nodo con su valor g
  - insertar los nuevos nodos en *abierta* en orden ascendente según su valor g

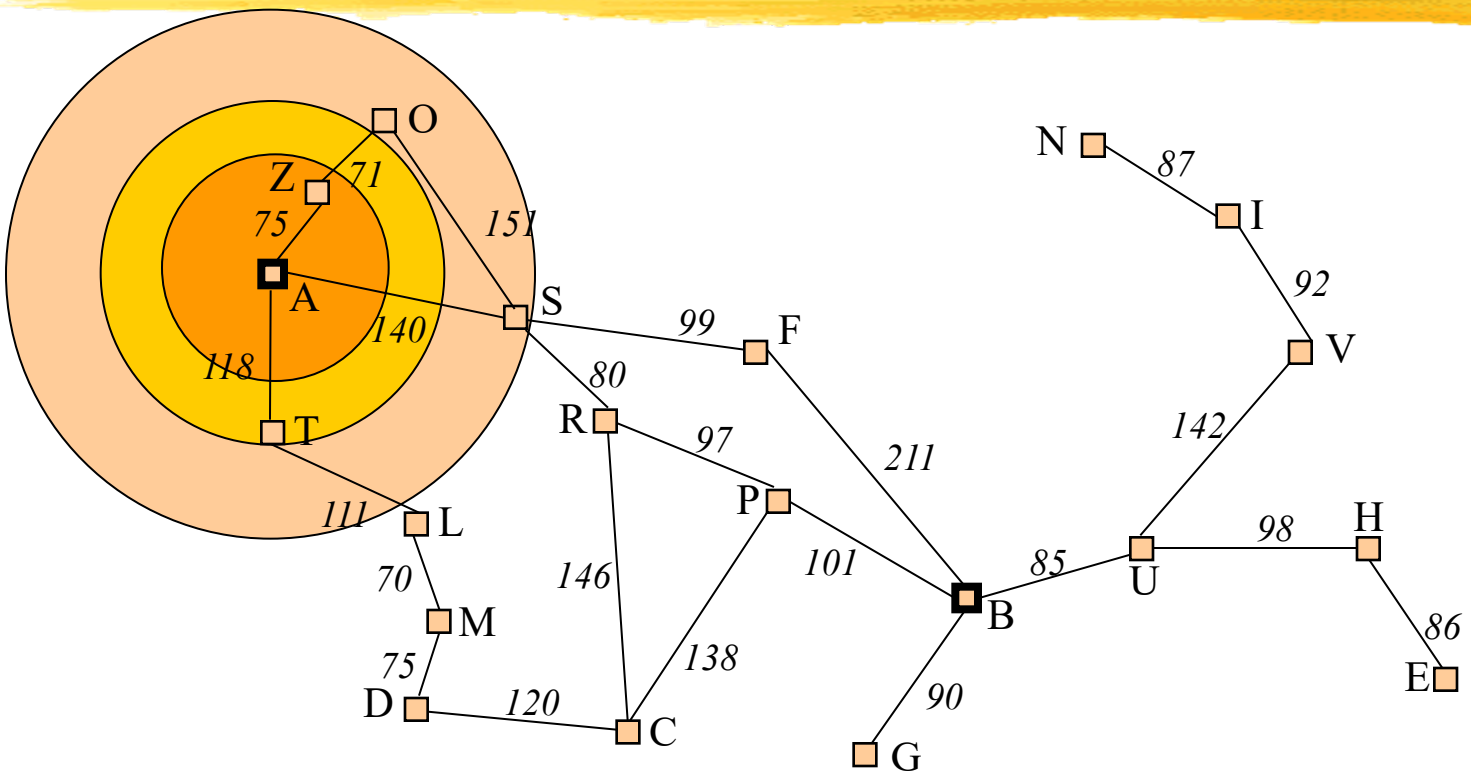
```
{búsqueda de coste uniforme}
abierta ← s0
Repetir
    Si vacío?(abierta) entonces
        devolver(negativo)
    nodo ← primero(abierta)
    Si meta?(nodo) entonces
        devolver(nodo)
    sucesores ← expandir(nodo)
    Para cada n∈ sucesores hacer
        n.padre ← nodo
        ordInsertar(n,abierta,g)
    Fin {repetir}
```

Ejemplo: Búsqueda de coste uniforme



• • •

Lógica de la búsqueda de coste uniforme



Características de la búsqueda de coste uniforme

Dinámica:

- la búsqueda de coste uniforme desarrolla sucesivamente todos los caminos por orden de valor g creciente
- igual que la búsqueda en amplitud si $g(n) = \text{prof.}(n)$ para todos los n

La búsqueda de coste uniforme es *completa*:

- sea n_m un nodo meta con $g(n_m) = k$
- suponga que no es encontrado por la búsqueda de coste uniforme
  - debe haber un número infinito de nodos n_i con $g(n_i) \leq k$
  - ya que el número de *sucesores* de un nodo es finito, debe haber un *camino* infinito p, tal que para todos los nodos n_i de p se cumple que $g(n_i) \leq k$
  - pero la función de coste c asigna un entero positivo a cada operador, y ninguna sucesión creciente de enteros positivos *tiene límite*
- contradicción; en consecuencia el nodo meta n_m será encontrado

Características de la búsqueda de coste uniforme

La búsqueda de coste uniforme es *óptima*:

- suponga que se encuentra un camino a un nodo meta n_m con $g(n_m) = k$
- la búsqueda de coste uniforme desarrolla sucesivamente todos los caminos por orden de valor g creciente
- Puesto que g crece de forma monótona, si hubiera un nodo meta n_m' con $g(n_m') < k$, éste se habría encontrado antes que n_m
- contradicción; en consecuencia n_m es el nodo meta de menor coste

Complejidad en tiempo y espacio:

- exponencial, al igual que la búsqueda en amplitud

Ejercicio

Búsqueda de coste uniforme:

Aplique la búsqueda de coste uniforme para encontrar una ruta de Pitesti (*P*) a Fagaras (*F*). Desarrolle el árbol de búsqueda generado por dicho algoritmo, asumiendo que se evitan ciclos simples. Indique el valor *g* de cada nodo, así como el orden en el que se expanden los nodos.