Sistemas Inteligentes

Agentes Inteligentes que resuelven problemas

Dr. Diego Oliva

Contenido

2. Agentes Inteligentes que resuelven problemas
   - Formulación de agentes que resuelven problemas
   - Algoritmos de búsqueda con y sin información
   - Aplicación de algoritmos de búsqueda en problemas de satisfacción de restricciones
   - Aplicación de algoritmos de búsqueda al desarrollo de juegos inteligentes

Agentes Inteligentes que resuelven problemas

Un agente que resuelve problemas es aquel que decide que hacer para encontrar el conjunto de acciones que lo lleven a los estados deseables.

Pasos para solucionar un problema:

- Formular un objetivo en base a la situación actual y la medida de rendimiento del agente.
- Formulación del problema: definir que acciones y estados se deben considerar para encontrar la solución.
- Búsqueda de la secuencia de acciones que lleven a una solución.
- Solución.
- Ejecución de las acciones encontradas en la búsqueda.

Agentes Inteligentes que resuelven problemas

- Agentes
  - Como actúan para alcanzar la meta
    - Secuencia de acciones para alcanzarla
    - Agentes para la solución de problemas
- Problema, se define como:
  - Una meta u objetivo
  - Conjunto de medios que permiten alcanzar la meta

- Búsqueda
  - Procedimiento de exploración para determinar que soluciones se pueden obtener

Agentes Inteligentes que resuelven problemas

- Formulación de metas (objetivos)
  - Una meta es un conjunto de estados del ambiente
  - A través de las acciones, un agente pasa de un estado a otro

- Acciones
  - Generan transiciones de un estado a otro
  - El agente tiene que determinar que acciones permiten obtener el estado de la meta

Agentes Inteligentes que resuelven problemas

- Formulación del problema:

Consiste en decidir que acciones y estados habrán de considerarse para llegar a los objetivos o metas.

Aquí se define:

- ¿Qué condiciones son necesarias?
- ¿Qué sucede si no hay forma de discernir que camino nos lleva a la meta?
- ¿Qué decisión tomar en una situación especifica?

Agentes Inteligentes que resuelven problemas

- Búsquedas

Se definen como la evaluación de las diversas secuencias de acciones que permiten pasar de un estado a otro.

En términos generales, cuando un agente tiene ante si diversas opciones cuyo valor ignora, éstas se tienen que evaluar de alguna forma.

Agentes Inteligentes que resuelven problemas

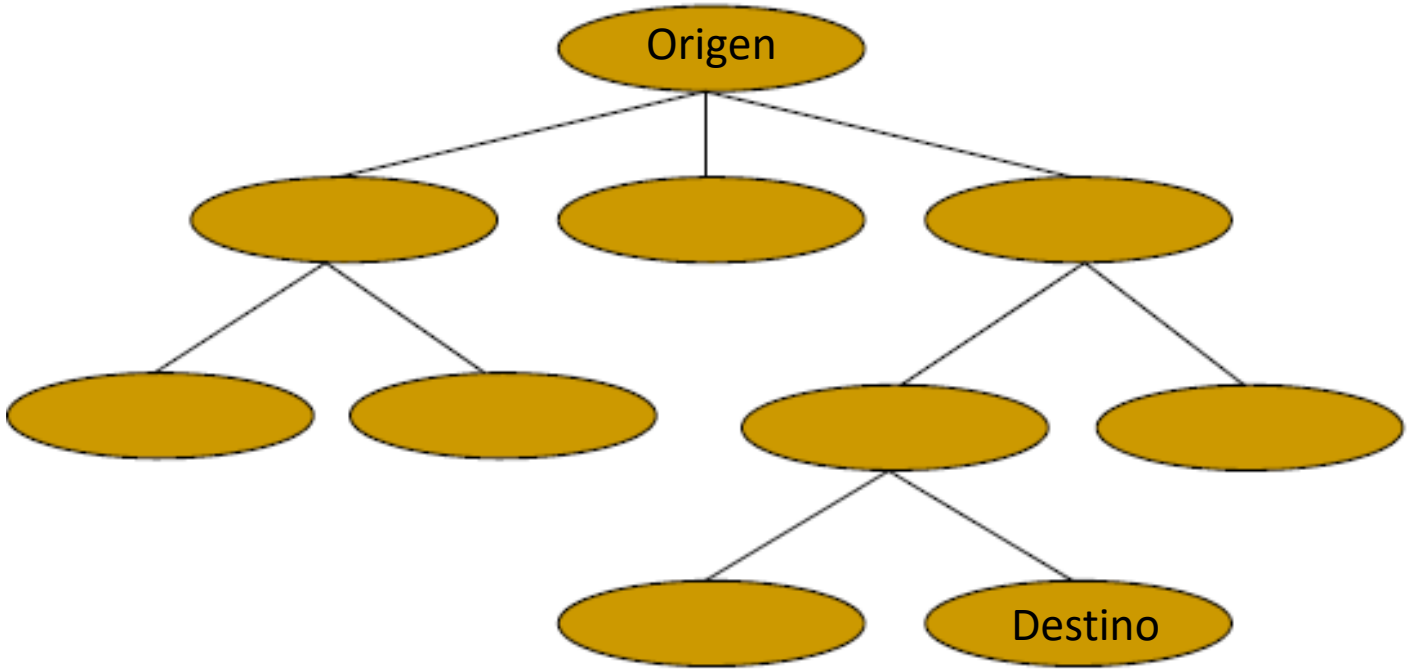
Algoritmo de búsqueda

- Entrada: un problema
- Salida: solución que adopta la forma de una secuencia de acciones

Una vez encontrada una solución, se procede a ejecutar las acciones

Agentes Inteligentes que resuelven problemas

Algoritmo de búsqueda



Agentes Inteligentes que resuelven problemas

Un Agente que Resuelve Problemas debe:

- Formular: decidir que acciones y estados deberán considerarse
- Buscar: proceso para hallar las secuencias de acciones que conduzcan a una meta
- Ejecutar: fase donde se llevan a cabo las acciones que conducen a la meta

Agentes Inteligentes que resuelven problemas

Un Agente que Resuelve Problemas:

```
función AGENTE-SENCILLO-RESOLVENTE-PROBLEMAS(percepción) devuelve una acción
  entradas: percepción, una percepción
  estático: sec, una secuencia de acciones, vacía inicialmente
            estado, una descripción del estado actual del mundo
            objetivo, un objetivo inicialmente nulo
            problema, una formulación del problema
  estado ← ACTUALIZAR-ESTADO(estado, percepción)
  si sec está vacía entonces hacer
        objetivo ← FORMULAR OBJETIVO(estado)
        problema ← FORMULAR-PROBLEMA(estado, objetivo)
        sec ← BÚSQUEDA(problema)
      acción ← PRIMERO(secuencia)
      sec ← RESTO(secuencia)
    devolver acción
```

Agentes Inteligentes que resuelven problemas

Ejemplo:

Imagine un agente en la ciudad de Arad, Rumanía, disfrutando de un viaje de vacaciones. Mañana sale un vuelo a Bucarest y solo puede llegar a esta ciudad por carretera conduciendo un automóvil.

Formulación del objetivo:

- Estar en Bucarest antes que salga el vuelo

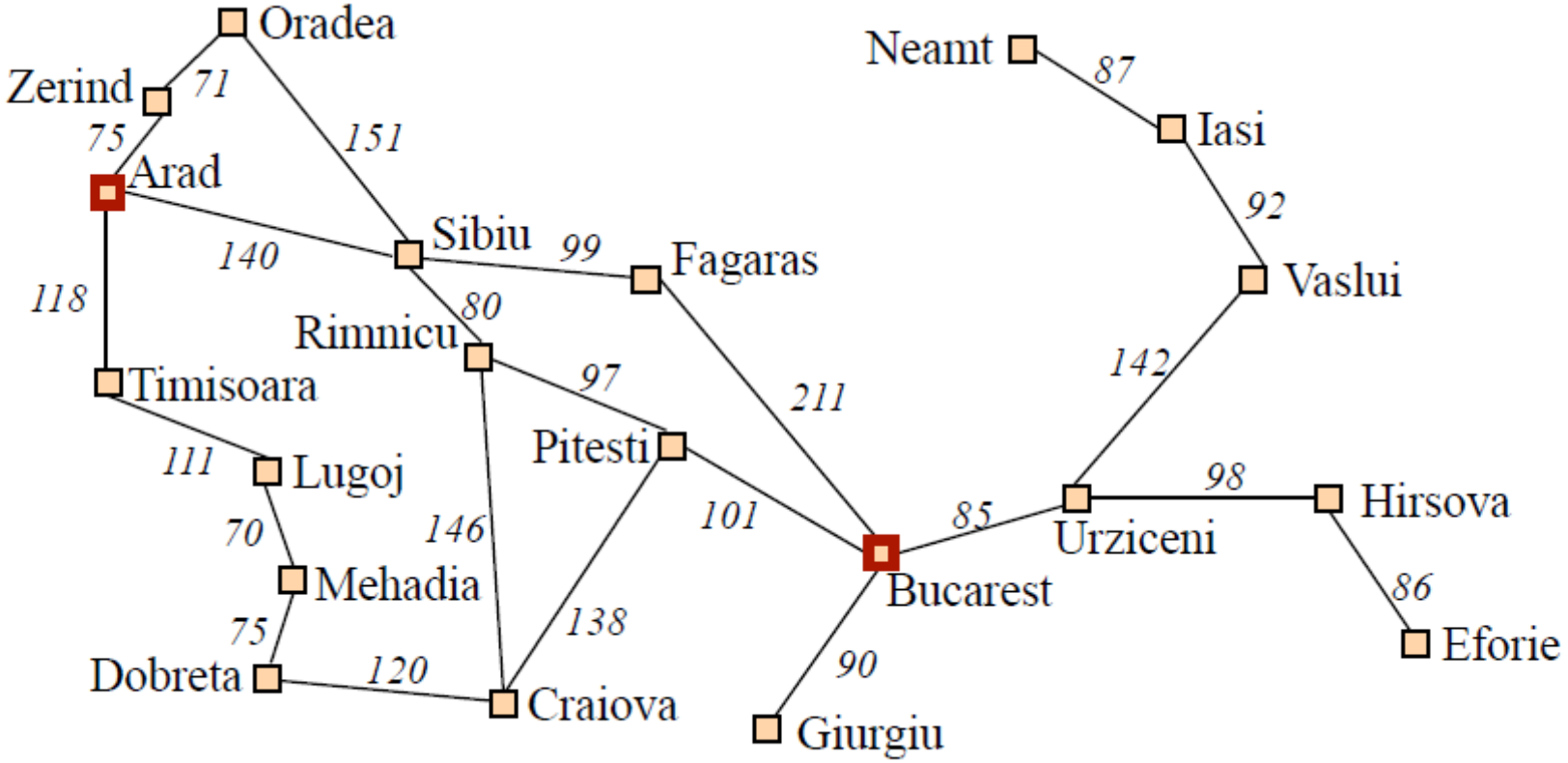
Formulación del problema:

- *estados*: varias ciudades
- *acciones*: conducir entre las ciudades

Encontrar solución:

- secuencia de ciudades, por ejemplo, Arad, Sibiu, Fagaras, Bucarest.

Agentes Inteligentes que resuelven problemas



Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos

Un problema se define por cuatro elementos básicos:

1. El **estado inicial** en el que comienza el agente.
2. **Función sucesor**, es una descripción de las posibles acciones disponibles por el agente.
   - **Espacio de estados**, es el conjunto de todos los estados alcanzables desde el estado inicial. (Grafo en el cual los nodos son estados y los arcos entre los nodos son acciones)
3. El **test objetivo,** el cual determina si un estado es un objetivo. En ocasiones existe un conjunto de explícito de estados posibles y estados objetivos, en este caso solo se comprueba si es o no objetivo.
4. **Costo del camino**, aquí se asigna un costo numero a cada camino. El agente elige una función de costo que represente la medida de rendimiento.

Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos

Ejemplo:

- *Estado inicial* , “Estoy en Arad”
- *Función sucesor, S*(*x*) = conjunto de pares acción-estado,

  S(*Arad*) = {<*Arad* → *Zerind*>,< *Zerind* → Ordea>,...}
- *Prueba de meta*, descripción para decidir si se trata de un estado meta

  explícito, por ejemplo, *x* = “en Bucarest”
- *Costo del camino, se asignan costos a una ruta*

  distancias, acciones ejecutadas, etc.
- Una *solución* es una secuencia de acciones que parten desde un estado inicial hasta alcanzar un estado objetivo.

Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos

Costos

- Mediante una ruta se conectan los conjuntos de estados
  - La solución es una ruta que conduce a estados meta

Espacio de conjunto de estados

- ¿Cuál es el mejor camino?
  - ¿Permite encontrar una solución? (decisión)
  - ¿Es una buena solución? (optimización)
  - ¿Cuál es el costo de búsqueda correspondiente al tiempo y memoria necesarios para encontrar la solución?

Costo total
C.T. = Costo del Camino + Costo de la Búsqueda

Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos

Ejemplo:

Estado inicial

- *n* bloques en una mesa de longitud ilimitada

Acciones:

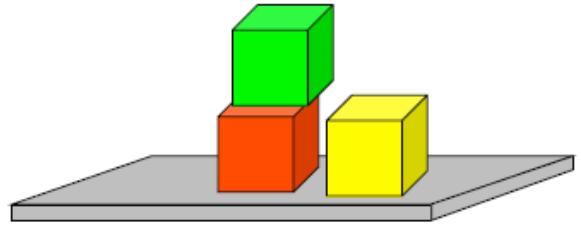
- *apilar*(*X*,*Y*): poner *X* encima de *Y*
  - Prec.: bloques *X* e *Y* están libres
  - Efecto: bloque *X* está encima de *Y*
- *quitar*(*Y*): poner *Y* en la mesa
  - Prec.: bloque *Y* está libre
  - Efecto: bloque *Y* está en la mesa

Prueba de meta

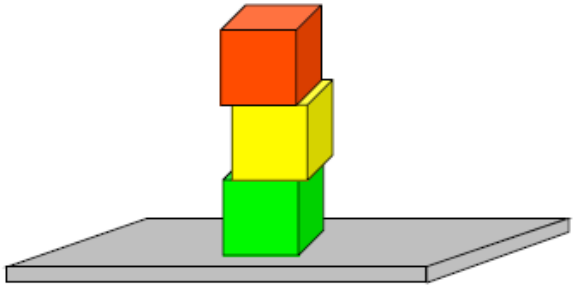
- Objetivo: cierta configuración de bloques
- Sólo la posición *vertical* es relevante

El cost0 de cada acción es uno

Situación presente

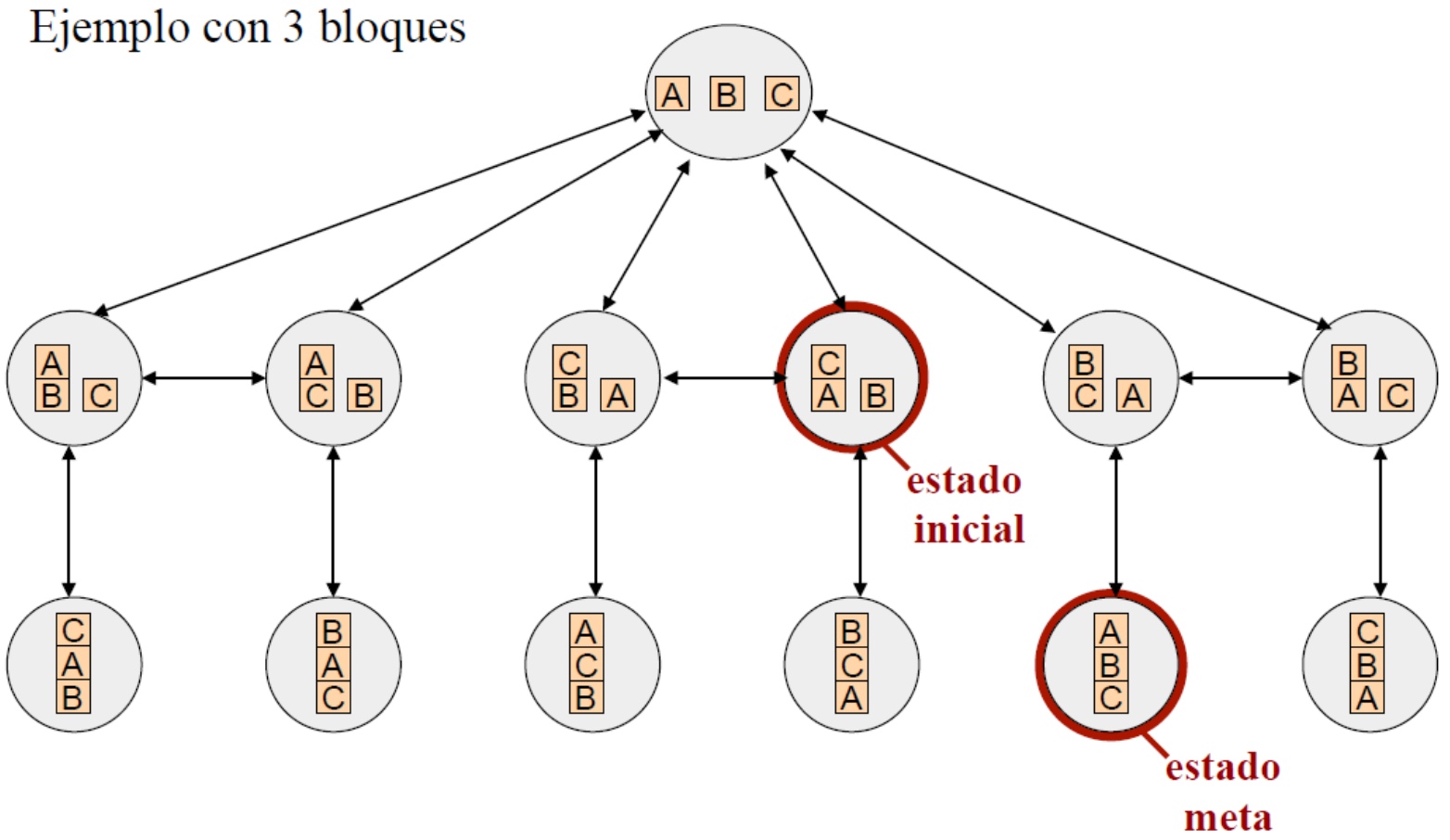


Objetivo



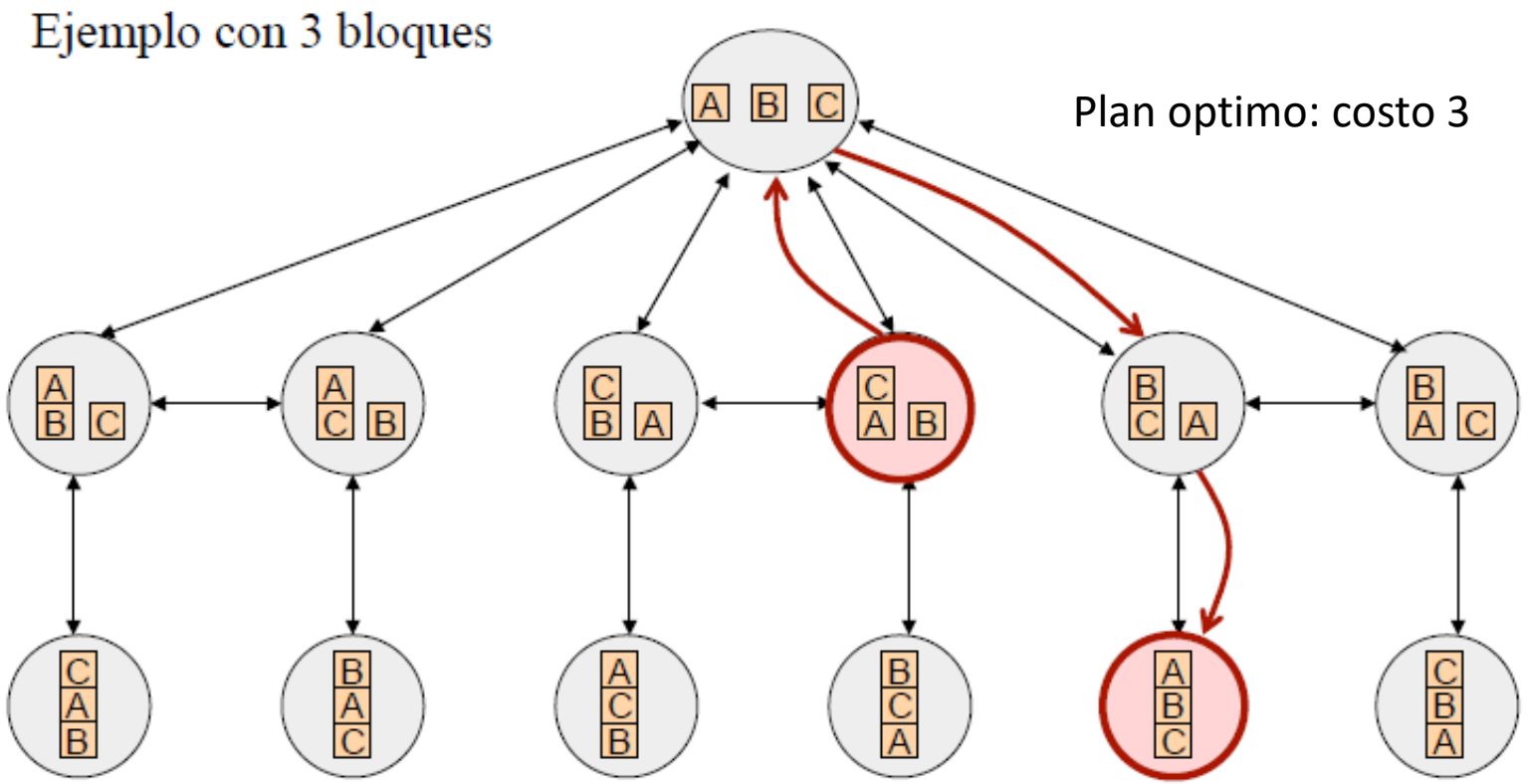
Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos



Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos



Agentes Inteligentes que resuelven problemas

Problemas y soluciones bien definidos

Tipos de problemas:

- **Problemas de juguete**: se emplean para ilustrar o entrenar los métodos planteados. Estos problemas se pueden describir de forma exacta, comúnmente se emplean para comparar el rendimiento de los algoritmos.
- **Problema del mundo real:** es aquel en el que el ser humano se preocupa por solucionar. No tienen una sola descripción, pero se realizan generalizaciones que lleven a soluciones optimas.

Agentes Inteligentes que resuelven problemas

<u>Problemas de juguete</u>

Problema de la aspiradora

- Estado simple, estado inicial 5. ¿Solución?

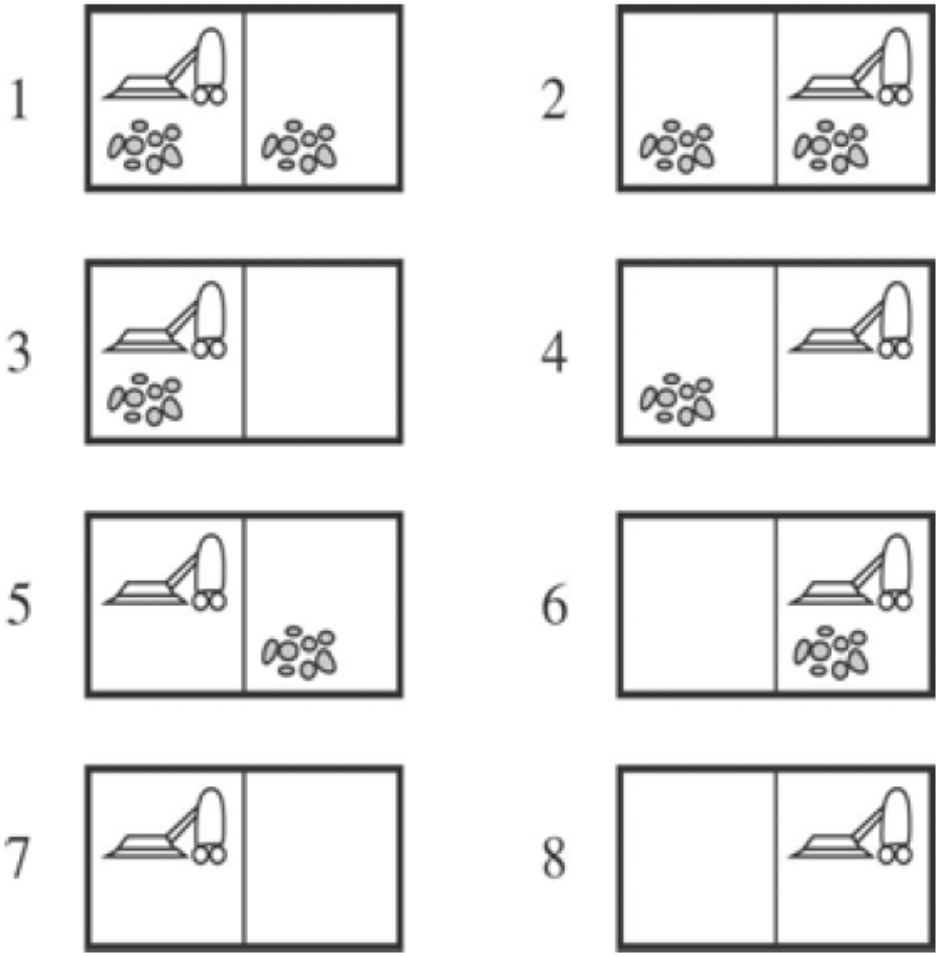
  [*Derecha*, *Aspirar*]

- Estado múltiple, estado inicial

  {1, 2, 3, 4, 5, 6, 7, 8}

¿Qué sucede si se elije la acción Derecha?

La acción *Derecha* produce {2, 4, 6, 8}



Agentes Inteligentes que resuelven problemas

<u>Problemas de juguete</u>

Problema de la aspiradora

- Estado múltiple, estado inicial

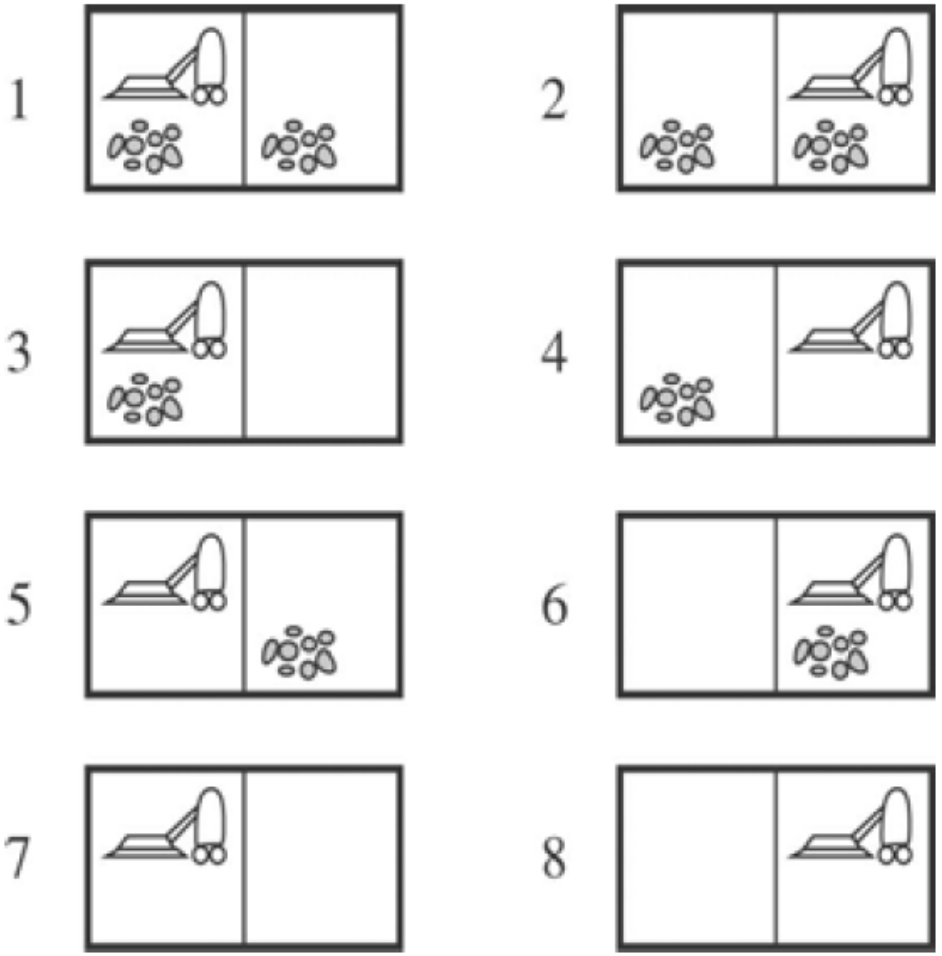
  {1, 2, 3, 4, 5, 6, 7, 8}

¿Qué sucede si se elije la acción Derecha?

La acción *Derecha* produce {2, 4, 6, 8}

¿Qué pasa con la suciedad que hay en los estados {1,3,5,7}?

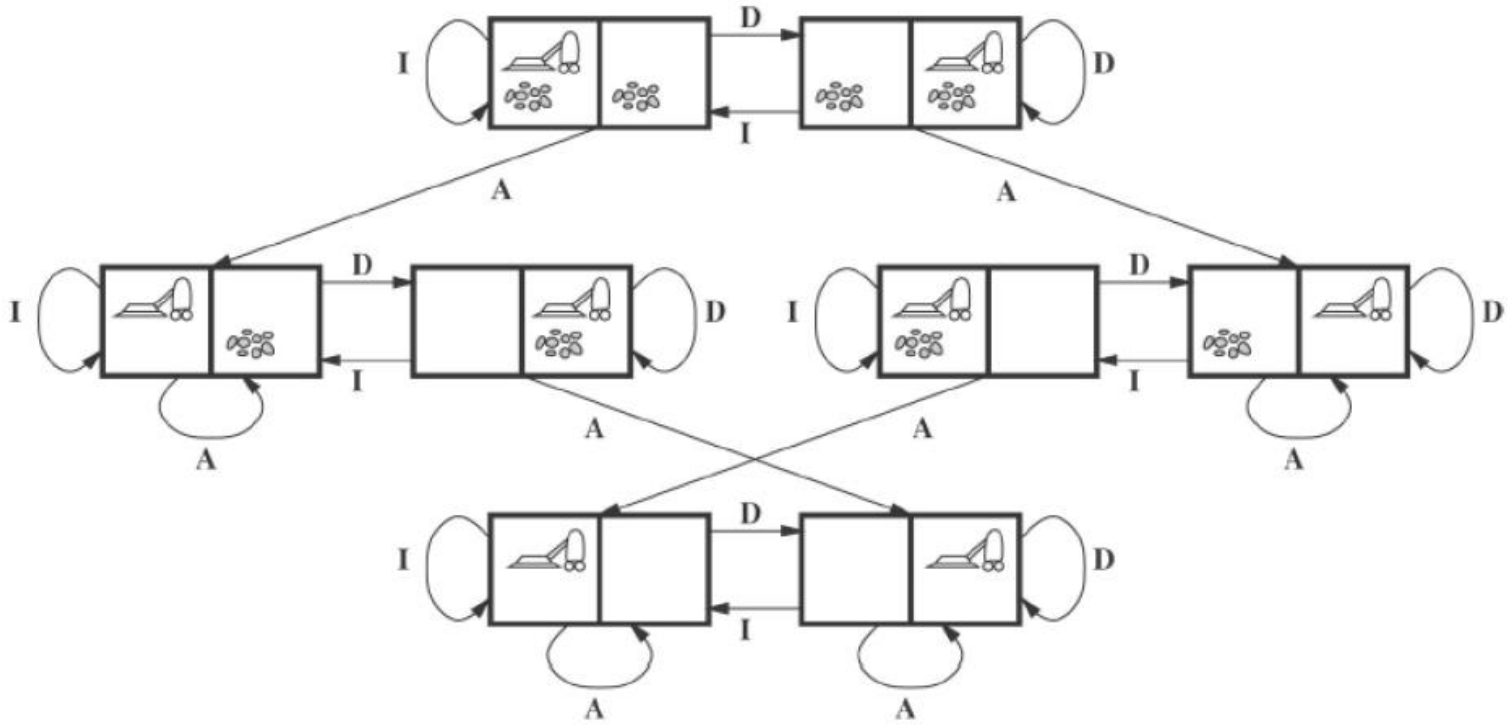
[*Derecha, Aspirar, Izquierda, Aspirar*]



Agentes Inteligentes que resuelven problemas

Problemas de juguete

Problema de la aspiradora



Definir:

- ¿Estados?
- ¿Acciones?
- ¿Prueba de meta?
- ¿Costo del camino?

Agentes Inteligentes que resuelven problemas

Problemas de juguete

Problema de la aspiradora

- ¿Estados?

  Suciedad completa y localizaciones
  de robot (ignorar *cantidades* de suciedad)
  Cualquier estado puede ser inicial

- ¿Acciones? (Función sucesor)

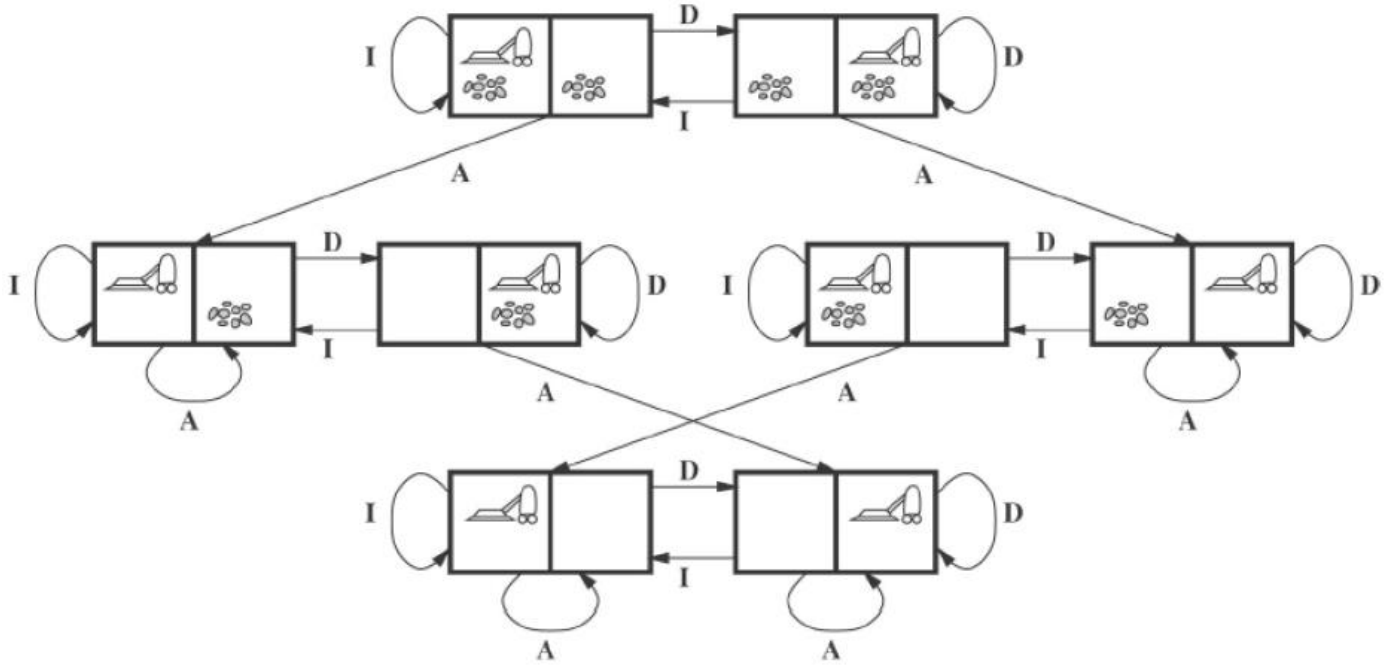
  Izquierda, Derecha, Aspirar, NoOp

- ¿Test objetivo?

  No suciedad, todos los cuadros limpios

- ¿Costo del camino?

  1 por acción (0 por *NoOp*)



Agentes Inteligentes que resuelven problemas

Problemas de juguete

Problema 8-puzzle

- ¿Estados?

  Localizaciones completas de las piezas

  Cualquier estado puede ser inicial

- ¿Acciones? (Función sucesor)

  Mover el negro a la izquierda, derecha, arriba, abajo

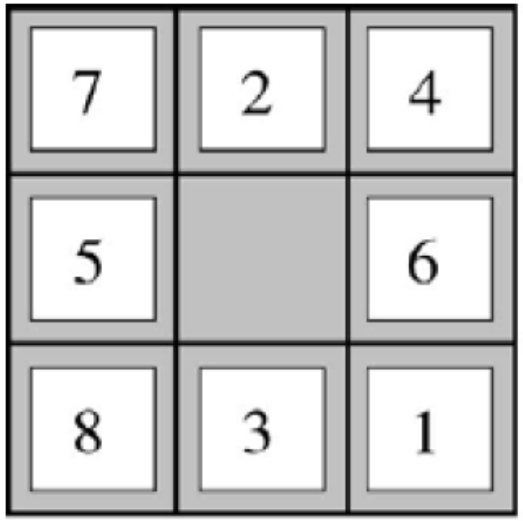
- ¿Test objetivo?

  Comprobar si se llego al estado objetivo

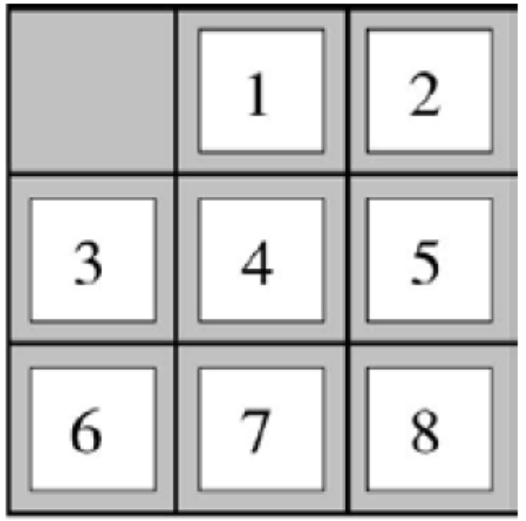
- ¿Costo del camino?

  1 por movimiento

[Nota: solución óptima de la familia del *n*-puzzle es NP-Completo]



Estado Inicial



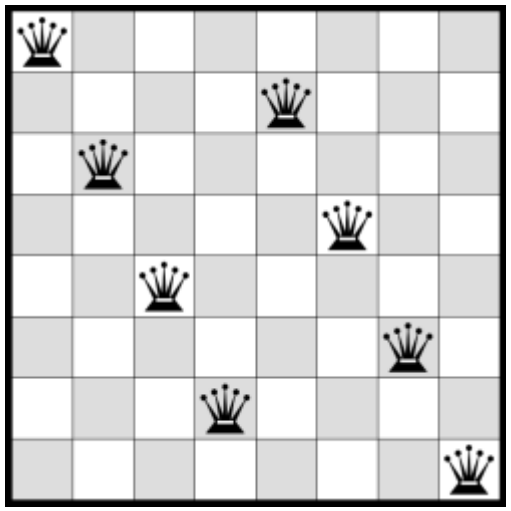
Estado Objetivo

Agentes Inteligentes que resuelven problemas

Problemas de juguete

Problema de las 8 Reinas

Colocar las 8 reinas en el tablero de manera que ninguna ataque a la otra.



- ¿Estados?

  Cualquier combinación de 0 a 8 Reinas

  Estado inicial: ninguna reina sobre el tablero

- ¿Acciones? (Función Sucesor)

  Añadir una reina a cualquier casilla vacía

- ¿Test objetivo?

  8 reinas sobre el tablero sin que se ataque ninguna

- ¿Costo del camino?

  No tiene relevancia en este problema

¿Cómo se puede simplificar el problema?

Estados: combinaciones de 8 reinas, una por columna desde la columna mas a la izquierda sin que exista posibilidad de atacarse.

Función sucesor: añadir una reina en la columna mas a la izquierda sin evitando ataques entre ellas.

Agentes Inteligentes que resuelven problemas

Problemas del mundo real:

Búsqueda de rutas en un aeropuerto:

Estados: Localización y hora actual

Estado inicial: es especificado por el problema

Función sucesor: devuelve los estados que resultan de tomar cualquier vuelo programado desde el aeropuerto actual a otro, que salgan a la hora actual mas el tiempo de transito.

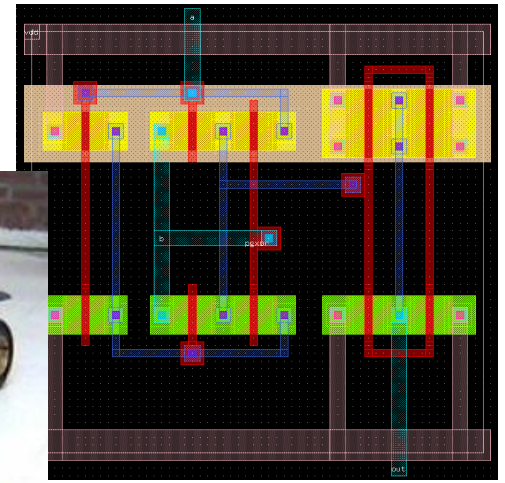
Test objetivo: ¿Se tiene el destino para cierta hora especificada?

Costo: depende del problema puede ser, dinero, tiempo de espera, tiempo de vuelo, calidad del vuelo, etc.

Agentes Inteligentes que resuelven problemas

Problemas del mundo real:

- Diseño electrónico
- Navegación de robots
- Secuencias de ensamble automático
- Búsquedas en internet



Búsqueda de soluciones

Árbol de búsqueda

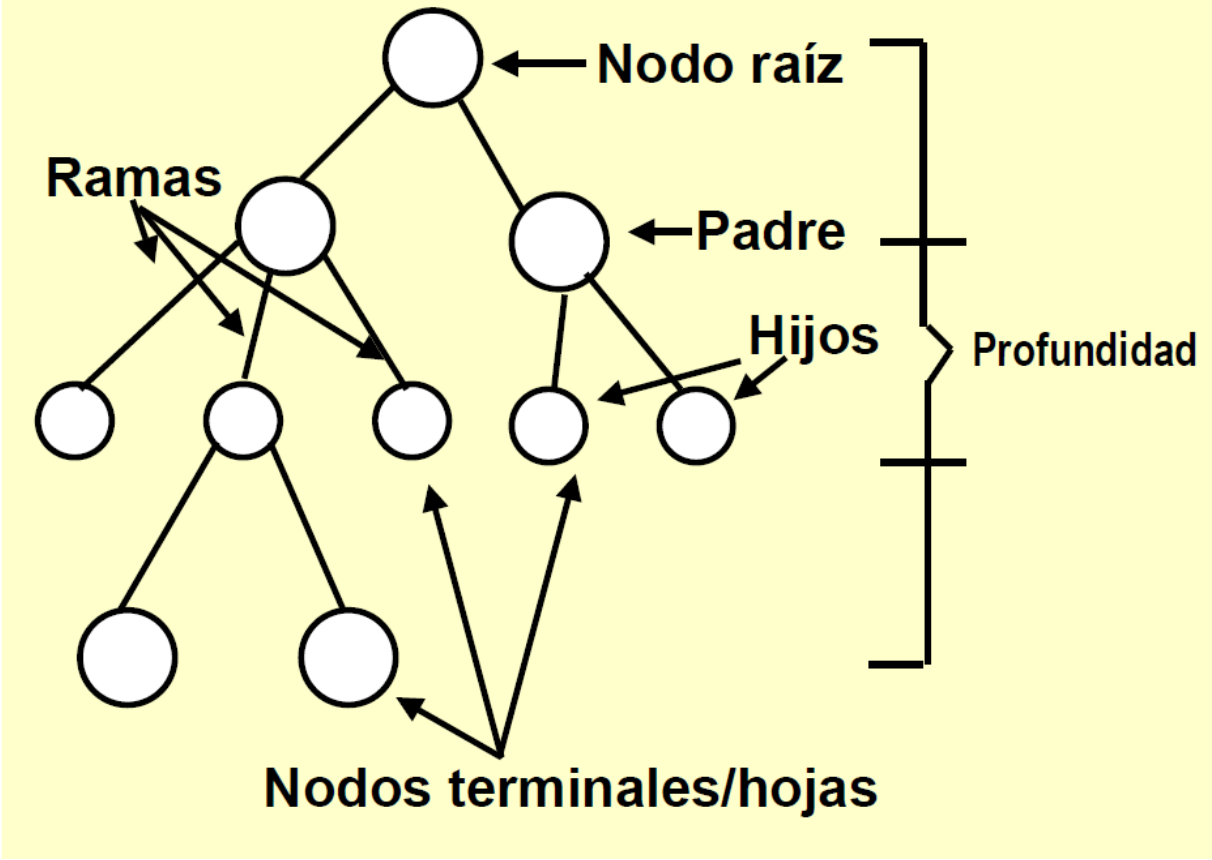
Idea general: Explorar las diferentes ramas de un árbol, con el objetivo de encontrar un camino desde la raíz a una hoja que represente un estado final.

Se debe considerar la siguiente terminología:

nodo, árbol, hoja, nodo-raíz, nodo-terminal, *branching factor* (factor de arborescencia), ramas, padres, hijos, árbol uniforme, etc.

Búsqueda de soluciones

Árbol de búsqueda



Búsqueda de soluciones

Árbol de búsqueda:

Definiciones:

- **Estado:** es extraído del espacio de estados y corresponde al nodo.
- **Nodo padre:** es el nodo en el árbol que genera un nuevo nodo.
- **Acción:** es aplicada al padre para generar un nuevo nodo.
- **Costo del camino:** es el camino que va desde el estado inicial al nodo actual.
- **Profundidad:** numero de paso a lo largo del camino desde el estado inicial.

Búsqueda de soluciones

Árbol de búsqueda:

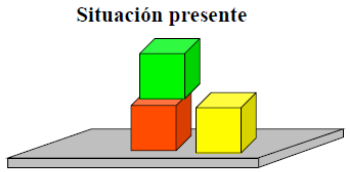
función BÚSQUEDA-ÁRBOLES(*problema*,*estrategia*) **devuelve** una solución o fallo

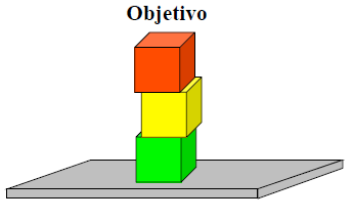
inicializa el árbol de búsqueda usando el estado inicial del *problema*

Hacer ciclo

si no hay candidatos para expandir **entonces devolver** fallo
escoger, de acuerdo a la *estrategia*, un nodo hoja para expandir
si el nodo contiene un estado objetivo **entonces devolver** la correspondiente solución
en otro caso expandir el nodo y añadir los nodos resultado al árbol de búsqueda

Búsqueda de soluciones





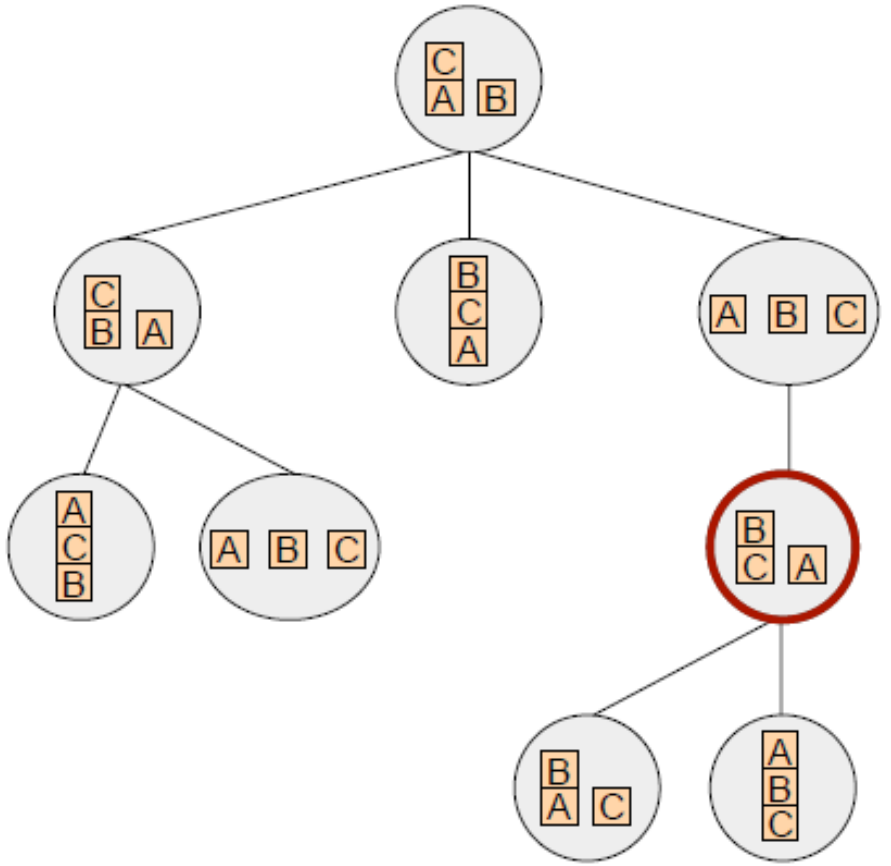
Árbol de búsqueda:

Ejemplo: problema de los bloques

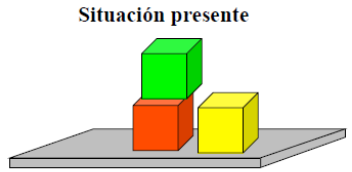
Método de búsqueda:
- *estrategia* para explorar el espacio de estados
- en cada paso se *expande* un estado
- se desarrolla sucesivamente un *árbol de búsqueda*

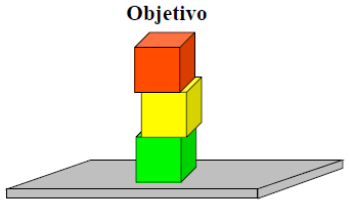
Método general de búsqueda:
1. seleccionar nodo hoja
3. *expandir* este nodo hoja
2. comprobar si es nodo meta

Árbol de búsqueda



Búsqueda de soluciones





Árbol de búsqueda:

Ejemplo: problema de los bloques

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
- p*rimero* quita el primer elemento de una lista
- *ordInsertar* añade un nodo a una lista, clasificado según una función de orden
- *expandir* devuelve los hijos de un nodo

{búsqueda general}

1. abierta ← *s0*
2. **Repetir**
3. **Si** *vacía?*(abierta) **entonces**
4. **devolver**(negativo)
5. nodo ← primero(abierta)
6. **Si** *meta?*(nodo) **entonces**
7. **devolver**(nodo)
8. sucesores ← *expandir*(nodo)
9. **Para cada** n∈sucesores **hacer**
10. n.padre ← nodo
11. *ordInsertar*(n,abierta,<orden>)
12. **Fin** {repetir}

Búsqueda de soluciones

Medir el rendimiento de las soluciones:

La salida de un algoritmo de búsqueda suele ser un fallo o una solución, para verificar el rendimiento de las soluciones se proponen cuatro técnicas:

1. **Completitud**: ¿se garantiza que el algoritmo encuentre la solución si esta existe?
2. **Optimización**: ¿se ha encontrado la solución optima?
3. **Complejidad en tiempo**: ¿cuánto se tarda en encontrar una solución?
4. **Complejidad en espacio**: ¿cuánta memoria se requiere para la búsqueda?

Estrategias de búsqueda no informada

- **Búsqueda no informada:**

Este término significa que ellas no tienen información adicional acerca de los estados más allá de la que se proporciona por la propia definición del problema.

- **Búsqueda informada:**

Estas estrategias saben si un estado no objetivo es mas prometedor que otro en relación a la función objetivo.

Estrategias de búsqueda no informada

<u>Búsqueda en amplitud:</u>

Inglés: breadth first search

Estrategia:

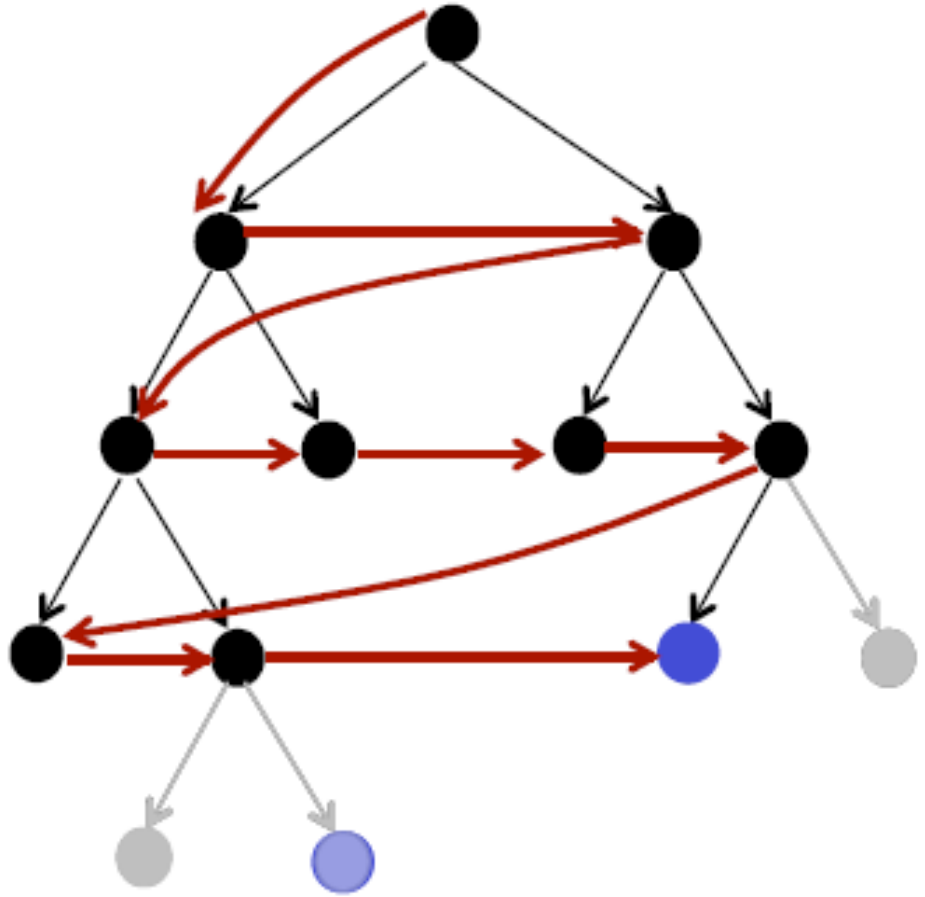
- Generar el árbol por niveles de profundidad
- Expandir todos los nodos de nivel *i*, antes de expandir nodos de nivel *i*+1

Resultado:

- Considera primero todos los caminos de longitud 1, después los caminos de longitud 2, etc.
- Se encuentra el estado meta de *menor profundidad*

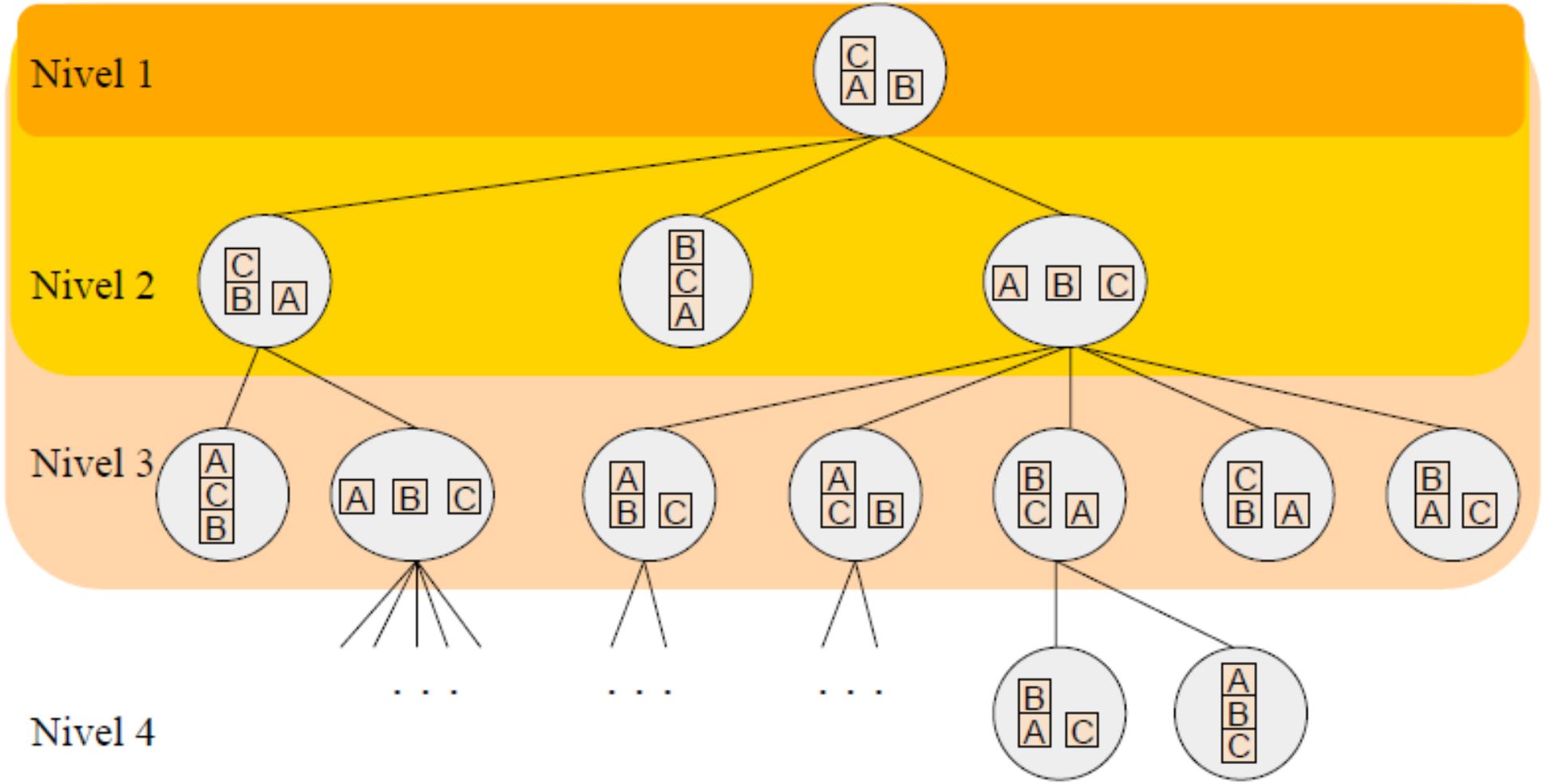
Estrategias de búsqueda no informada

Búsqueda en amplitud:



Estrategias de búsqueda no informada

Búsqueda en amplitud:



Estrategias de búsqueda no informada

Búsqueda en amplitud:

Algoritmo:

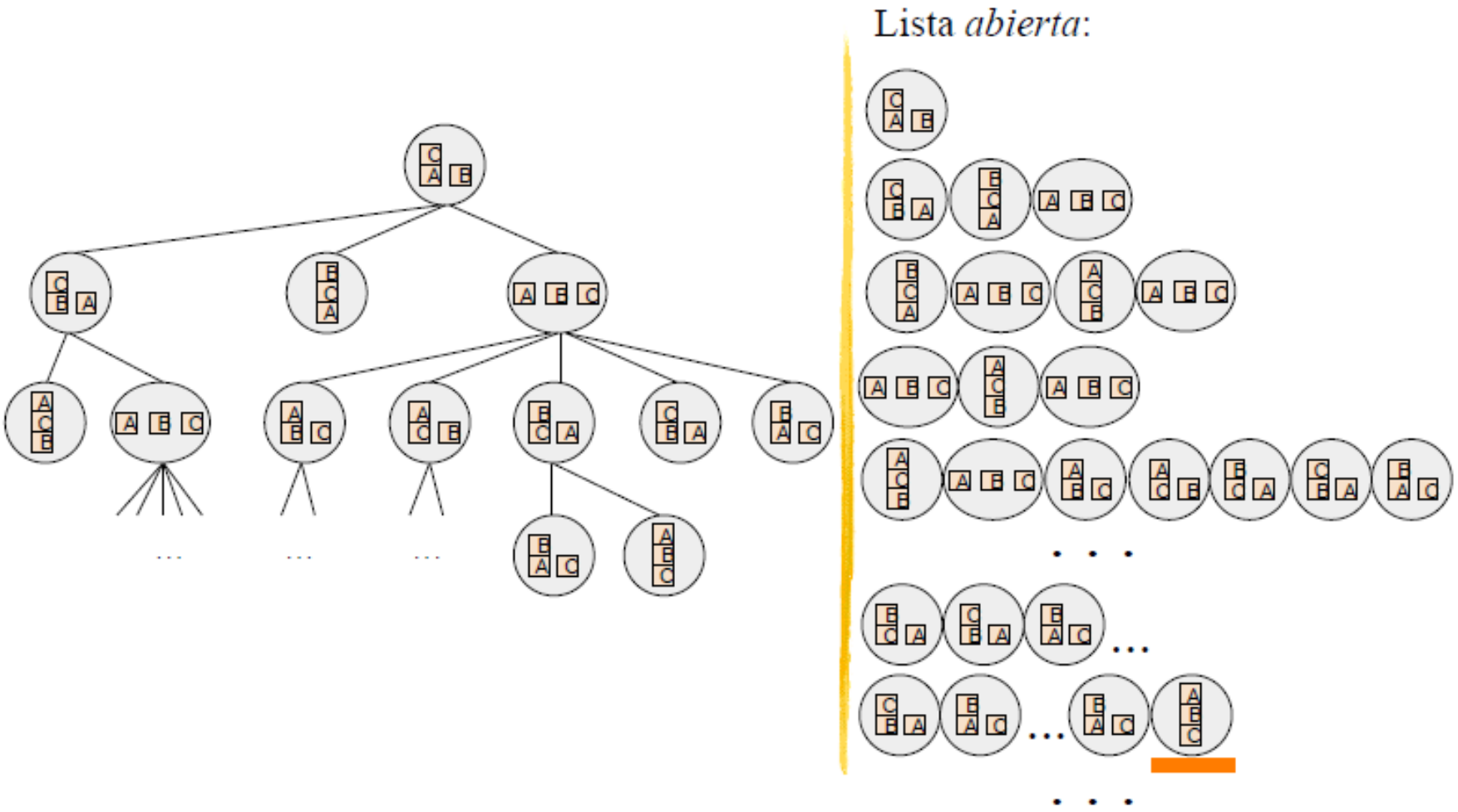
- Usar el algoritmo general de búsqueda
- Añadir nuevos sucesores al *final* de la lista *abierta*
- *abierta* funciona como *cola*
  - inserción al final
  - recuperación desde la cabeza
- estructura FIFO:
  - siempre expandir primero el nodo más antiguo

{búsqueda en amplitud}

1. abierta ← *s0*
2. **Repetir**
3. **Si** *vacía?*(abierta) **entonces**
4. **devolver**(negativo)
5. nodo ← primero(abierta)
6. **Si** *meta?*(nodo) **entonces**
7. **devolver**(nodo)
8. sucesores ← *expandir*(nodo)
9. **Para cada** n∈ sucesores **hacer**
10. n.padre ← nodo
11. *ordInsertar*(n,abierta,*final*)
12. **Fin** {repetir}

Estrategias de búsqueda no informada

Búsqueda en amplitud:



Estrategias de búsqueda no informada

Búsqueda en amplitud:

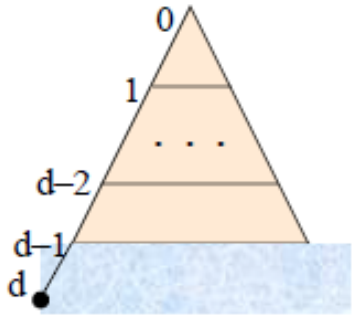
Complejidad en tiempo y espacio:

- proporcional al número de nodos expandidos

Suponemos que en el árbol de búsqueda

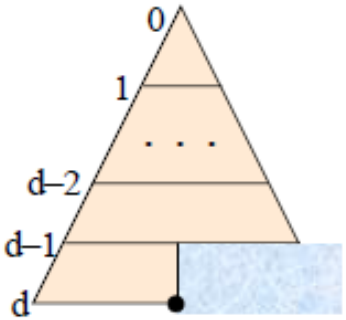
- el factor de ramificación es b
- el mejor nodo meta tiene profundidad d

Mejor caso



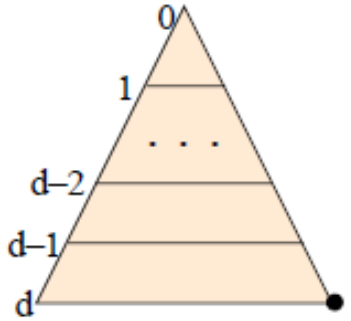
$1+b+\ldots+b^{d-1}+1 \in O(b^d)$

Caso medio



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

Peor caso



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

Estrategias de búsqueda no informada

Búsqueda en amplitud:

- Requerimientos de recursos de una búsqueda en amplitud exponencial factor de ramificación efectivo: 10
- tiempo: 1000 nodos/segundo
- memoria: 100 bytes/nodo

d	nodos	tiempo	memoria
0	1	1 ms	100 Bytes
2	111	100 ms	11 KB
4	11.111	11 s	1 MB
6	10^6	18 min	111 MB
8	10^8	31 horas	11 GB
10	10^{10}	128 días	1 TB
12	10^{12}	35 años	111 TB
14	10^{14}	3500 años	11.111 TB

Estrategias de búsqueda no informada

Búsqueda en amplitud:

Ventajas:

- Método completo:
  - Siempre se encuentra un nodo meta si existe
- Método 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

Estrategias de búsqueda no informada

<u>Búsqueda en profundidad:</u>

Inglés: depth first search

Estrategia:

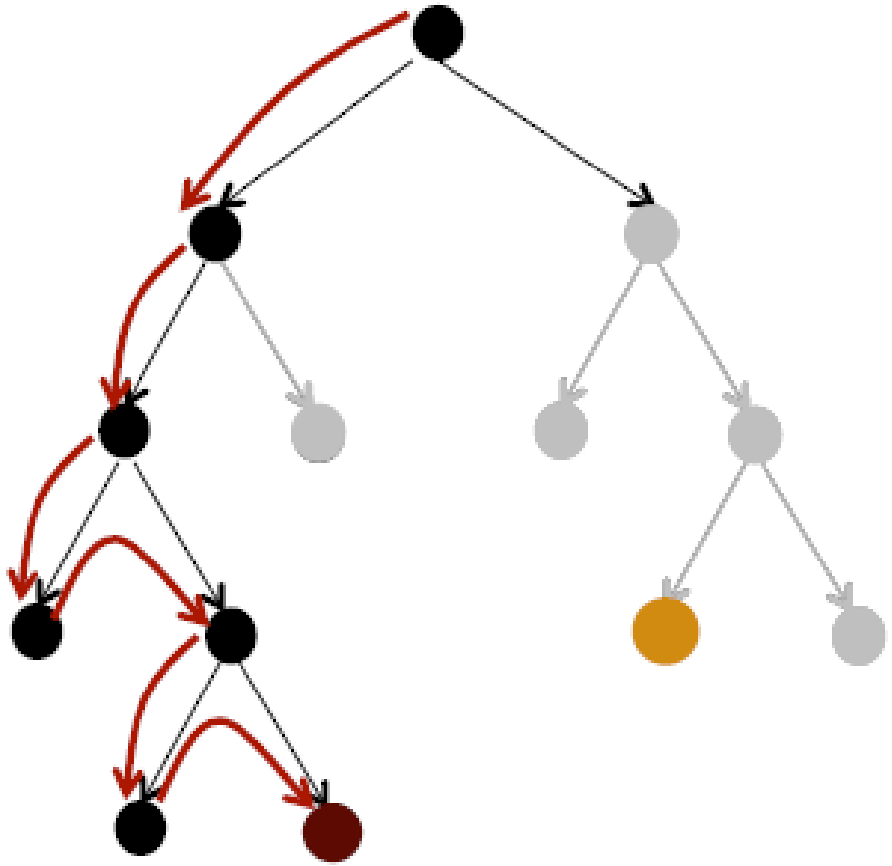
- Expandir los nodos más profundos primero
- Si se llega a un nodo sin sucesores, dar vuelta atrás y expandir el siguiente nodo más profundo

Resultado:

- El método va explorando un “camino actual”
- No siempre se encuentra el nodo de *profundidad mínima*

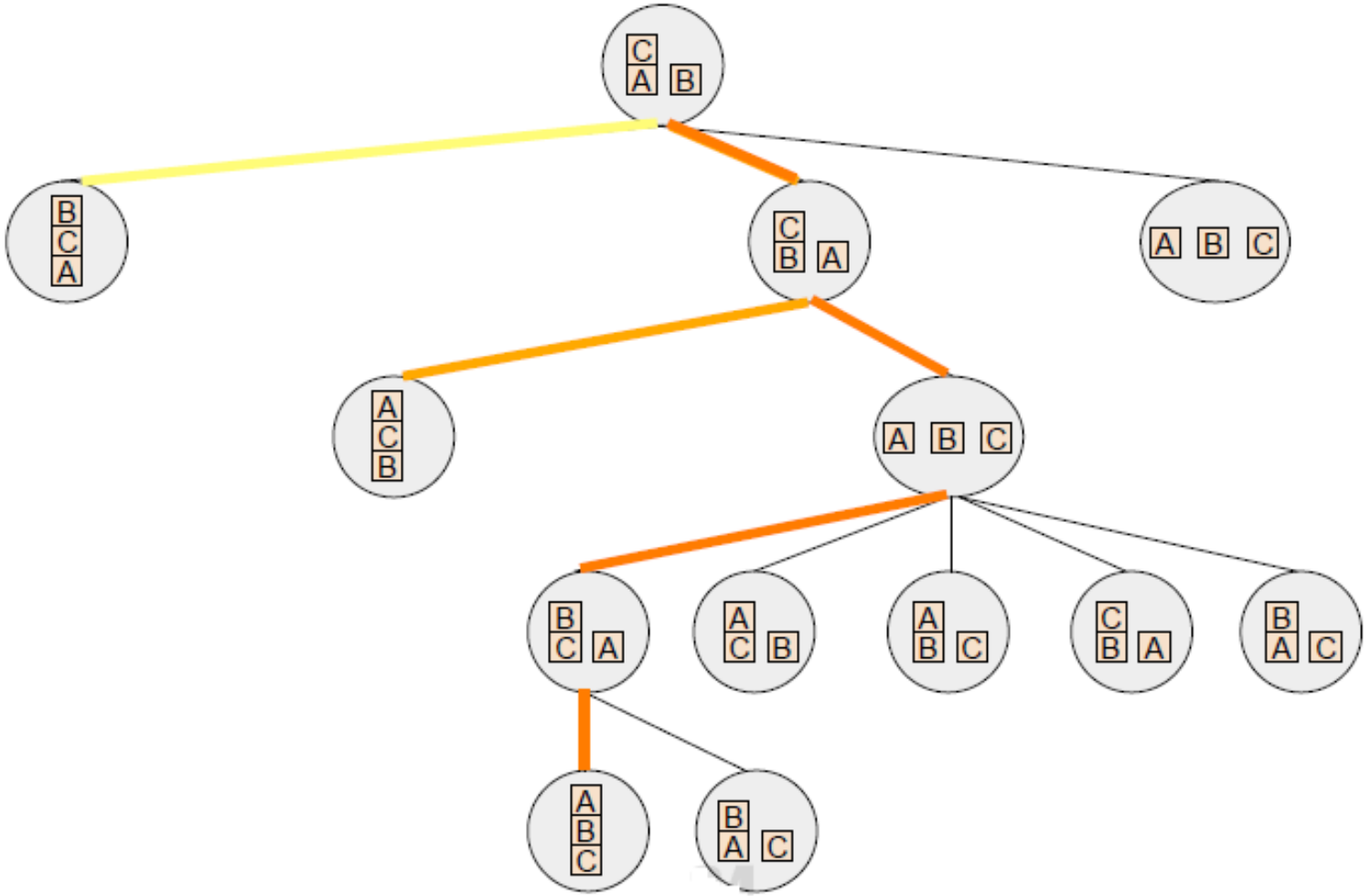
Estrategias de búsqueda no informada

Búsqueda en profundidad:



Estrategias de búsqueda no informada

Búsqueda en profundidad:



Estrategias de búsqueda no informada

Búsqueda en profundidad:

Algoritmo:

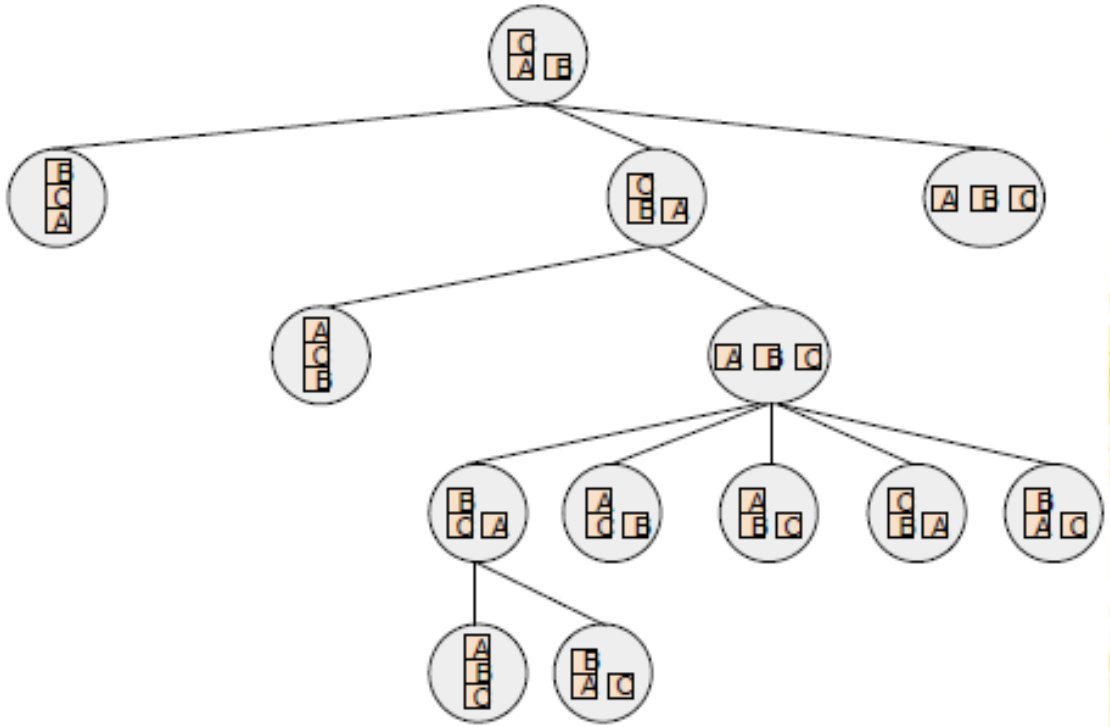
- Usar el algoritmo general de búsqueda
- Añadir nuevos sucesores *en la cabeza* de la lista *abierta*
- *abierta* funciona como *pila*
  - – inserción en la cabeza de la lista
  - – recuperación desde la cabeza
- Estructura LIFO:
  - – Siempre expandir primero el nodo más reciente (es decir: el más profundo)
- Al guardar *todos* los sucesores de un nodo expandido en *abierta*, se permite la "vuelta atrás"

{búsqueda en profundidad}

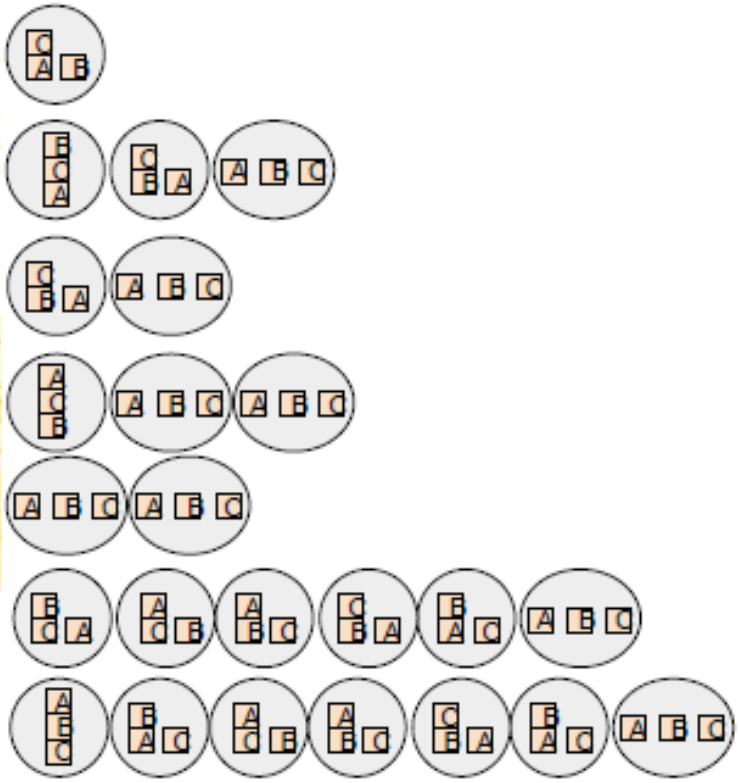
1. abierta ← *s0*
2. **Repetir**
3. **Si** *vacía?*(abierta) **entonces**
4. **devolver**(negativo)
5. nodo ← primero(abierta)
6. **Si** *meta?*(nodo) **entonces**
7. **devolver**(nodo)
8. sucesores ← *expandir*(nodo)
9. **Para cada** n∈ sucesores **hacer**
10. n.padre ← nodo
11. *ordInsertar*(n,abierta,*cabeza*)
12. **Fin** {repetir}

Estrategias de búsqueda no informada

Búsqueda en profundidad:



Lista *abierta*:

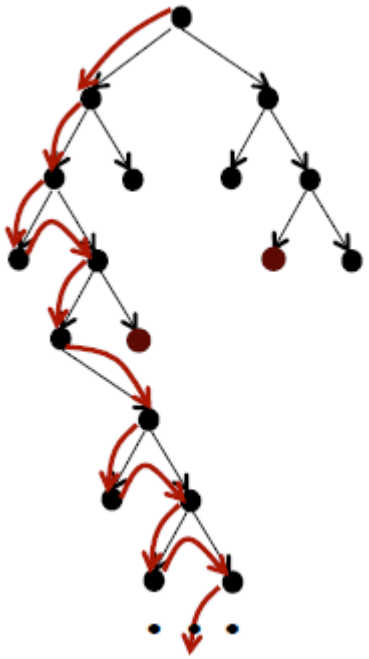


Estrategias de búsqueda no informada

Búsqueda en profundidad:

Problema:

- La búsqueda en profundidad sólo es completa en el caso de árboles de búsqueda finitos.
- Si existen caminos infinitos sin nodo meta, es posible que la búsqueda en profundidad no termine.



Solución:

- B*úsqueda en profundidad limitada:*
  - inglés: depth limited search
  - búsqueda en profundidad con límite de profundidad d^*
  - expandir sólo nodos con profundidad $d \leq d^*$
- Incompleto si la profundidad del mejor nodo meta es mayor que d^*

Estrategias de búsqueda no informada

Búsqueda en profundidad:

Complejidad en *tiempo*:

- Proporcional al número de nodos expandidos
- Factor de ramificación b / límite de profundidad d* / nodo meta con profundidad d≤d*
- Mejor caso: O(d) (se expanden sólo los nodos del camino meta)
- Peor caso: O(bd*) (se expanden todos los nodos de profundidad ≤ 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: $O(b \cdot d)$ / peor caso: $O(b \cdot d^*)$

Estrategias de búsqueda no informada

Búsqueda en profundidad:

Ventajas:

- Mejora significativa de la complejidad en espacio con respecto a la búsqueda en amplitud (lineal frente a exponencial):
    - Método 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.

Estrategias de búsqueda no informada

Búsqueda con profundidad 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´*

Estrategias de búsqueda no informada

Búsqueda con profundidad iterativa:

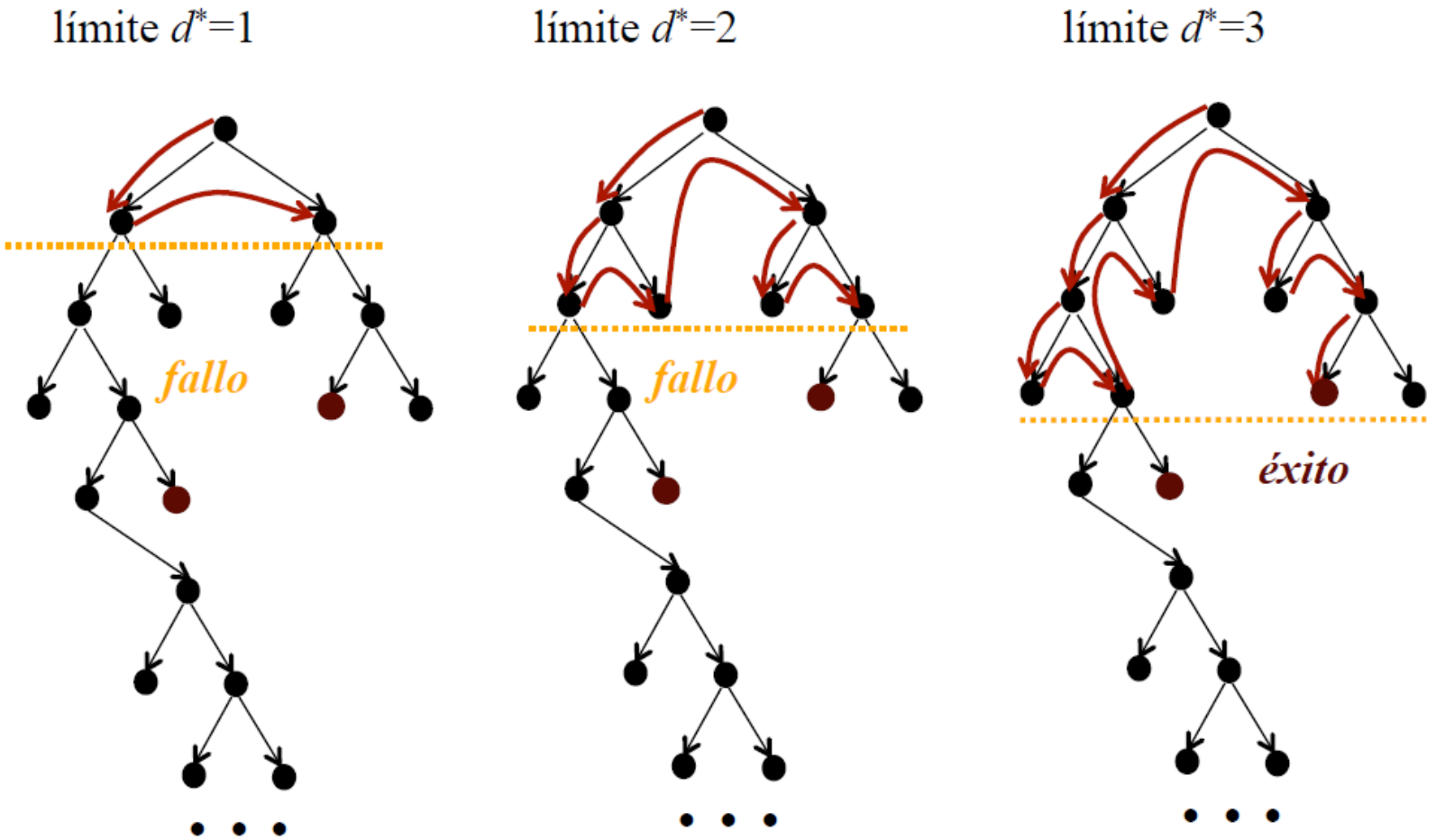
Algoritmo:

{búsqueda de profundización iterativa}

1. abierta ← *s0*
2. **desde** d´ ← 0 **hasta** ∝ **hacer**
   3. **si** *búsqueda-en-prof-limitada(problema, d´)* = éxito **entonces**
   4. **devolver**(nodo-meta)
5. **fin** {desde}

Estrategias de búsqueda no informada

Búsqueda con profundidad iterativa:



Estrategias de búsqueda no informada

Búsqueda de costo uniforme:

- Inglés: uniform cost search

Idea:

- Guiar la búsqueda por el *costo* de los operadores

Método:

- $g(n)$: costo mínimo para llegar del nodo inicial al nodo n
- Expandir siempre el nodo de menor coste g primero

Estrategias de búsqueda no informada

Búsqueda de costo uniforme:

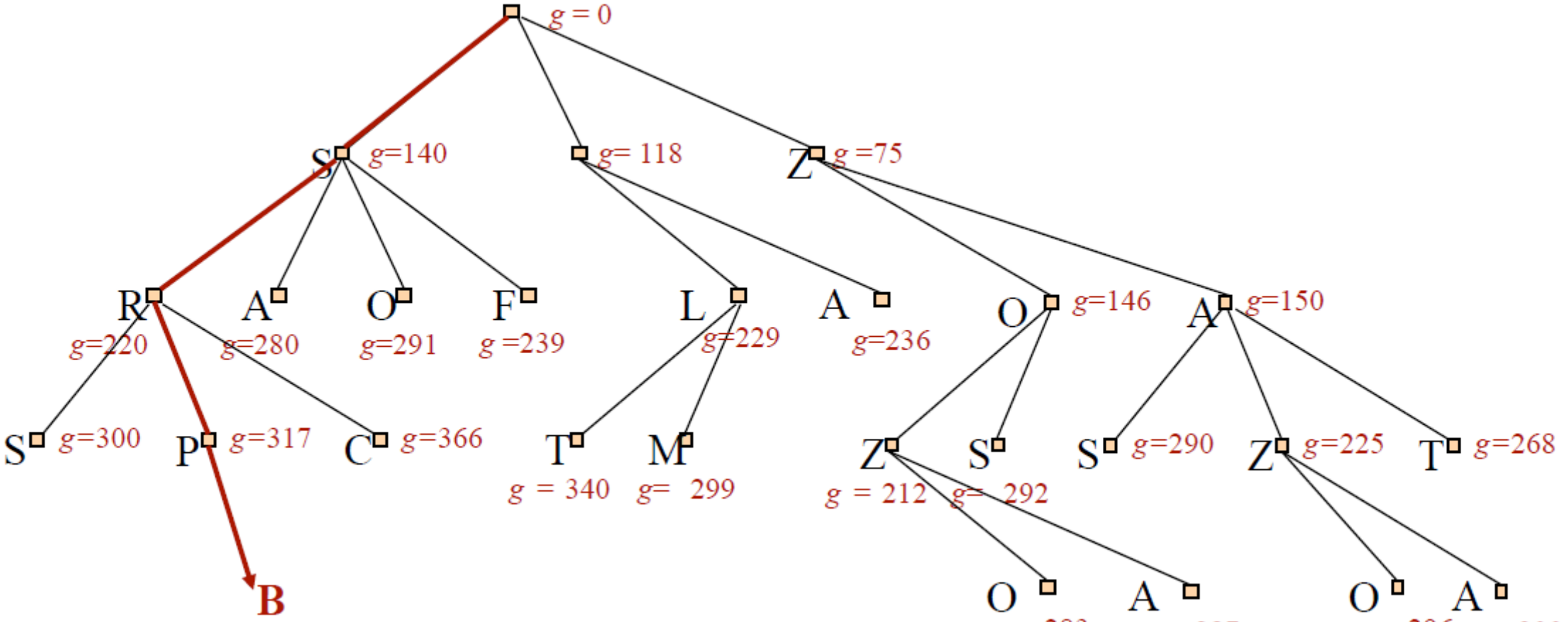
Algoritmo:

- Almacenar cada nodo con su valor *g*
- Insertar los nuevos nodos en *abierta* en orden ascendente según su valor *g*

```
{búsqueda de coste uniforme}
1.  abierta ← s0
2.  Repetir
3.      Si vacío?(abierta) entonces
4.          devolver(negativo)
5.          nodo ← primero(abierta)
6.      Si meta?(nodo) entonces
7.          devolver(nodo)
8.          sucesores ← expandir(nodo)
9.          Para cada n∈ sucesores hacer
10.             n.padre ← nodo
11.             ordInsertar(n,abierta,g)
12. Fin {repetir}
```

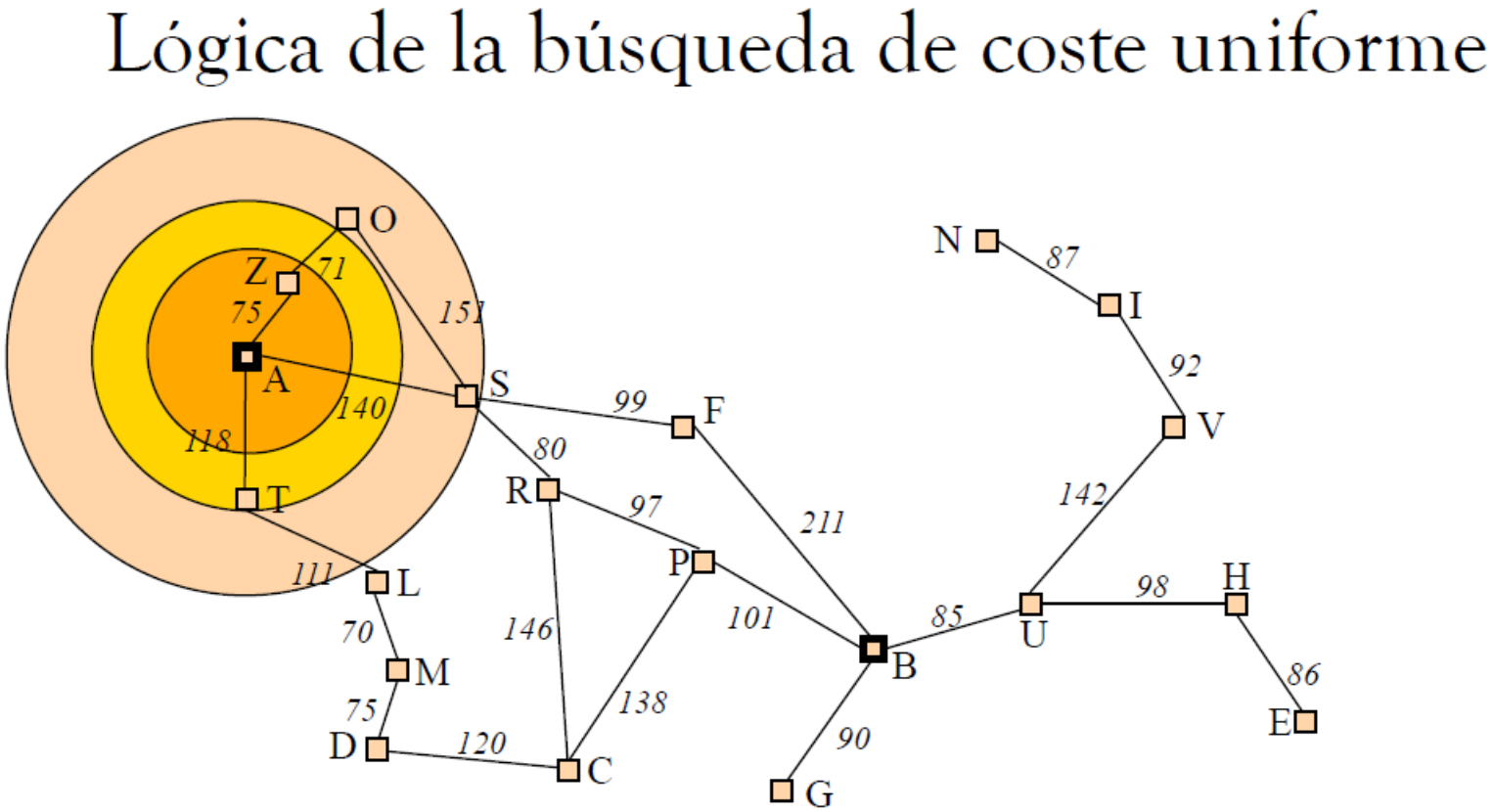
Estrategias de búsqueda no informada

Búsqueda de costo uniforme:



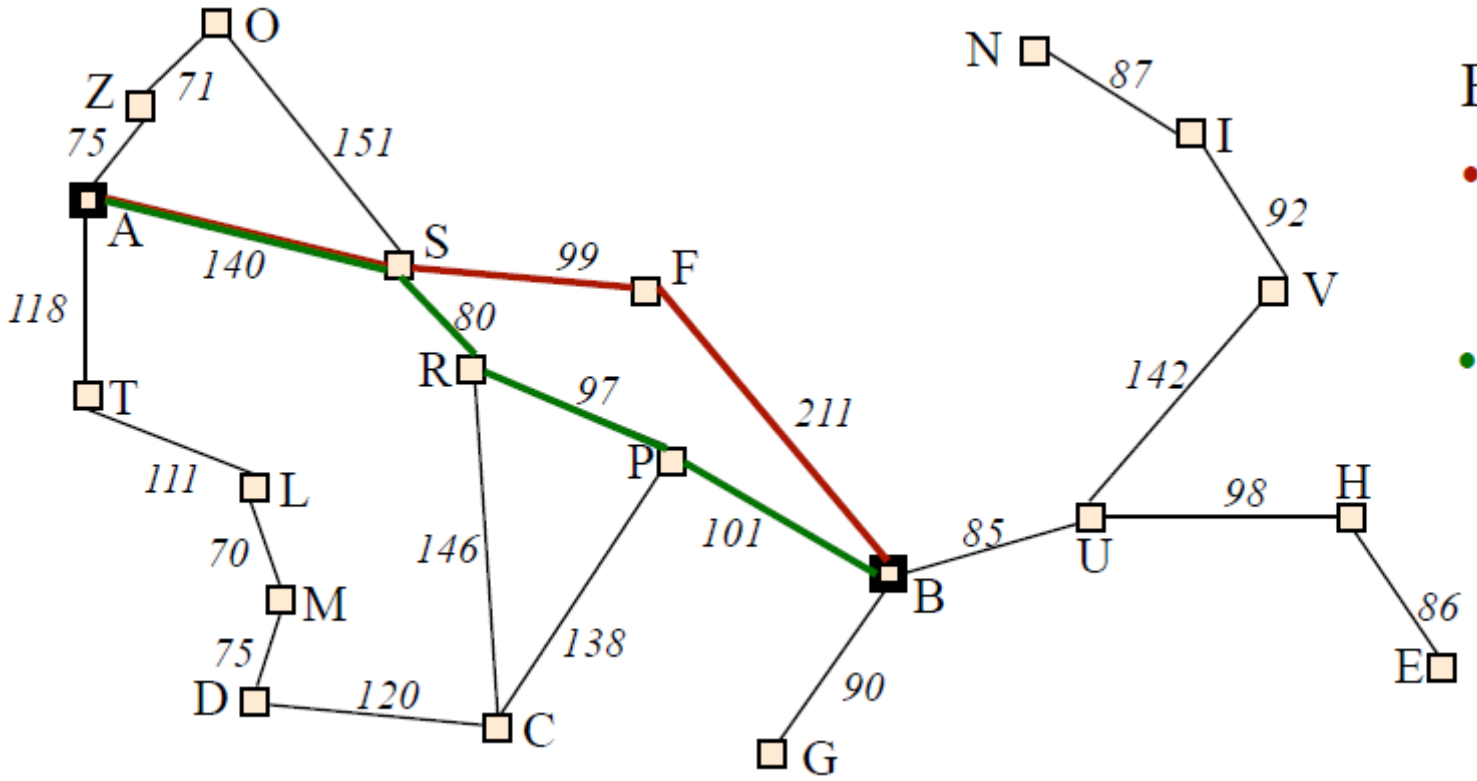
Estrategias de búsqueda no informada

Búsqueda de costo uniforme:



Estrategias de búsqueda no informada

Ejemplo de búsqueda de trayectorias



Ejemplo:

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

Estrategias de búsqueda no informada

Ejemplo de búsqueda de trayectorias

Problema:

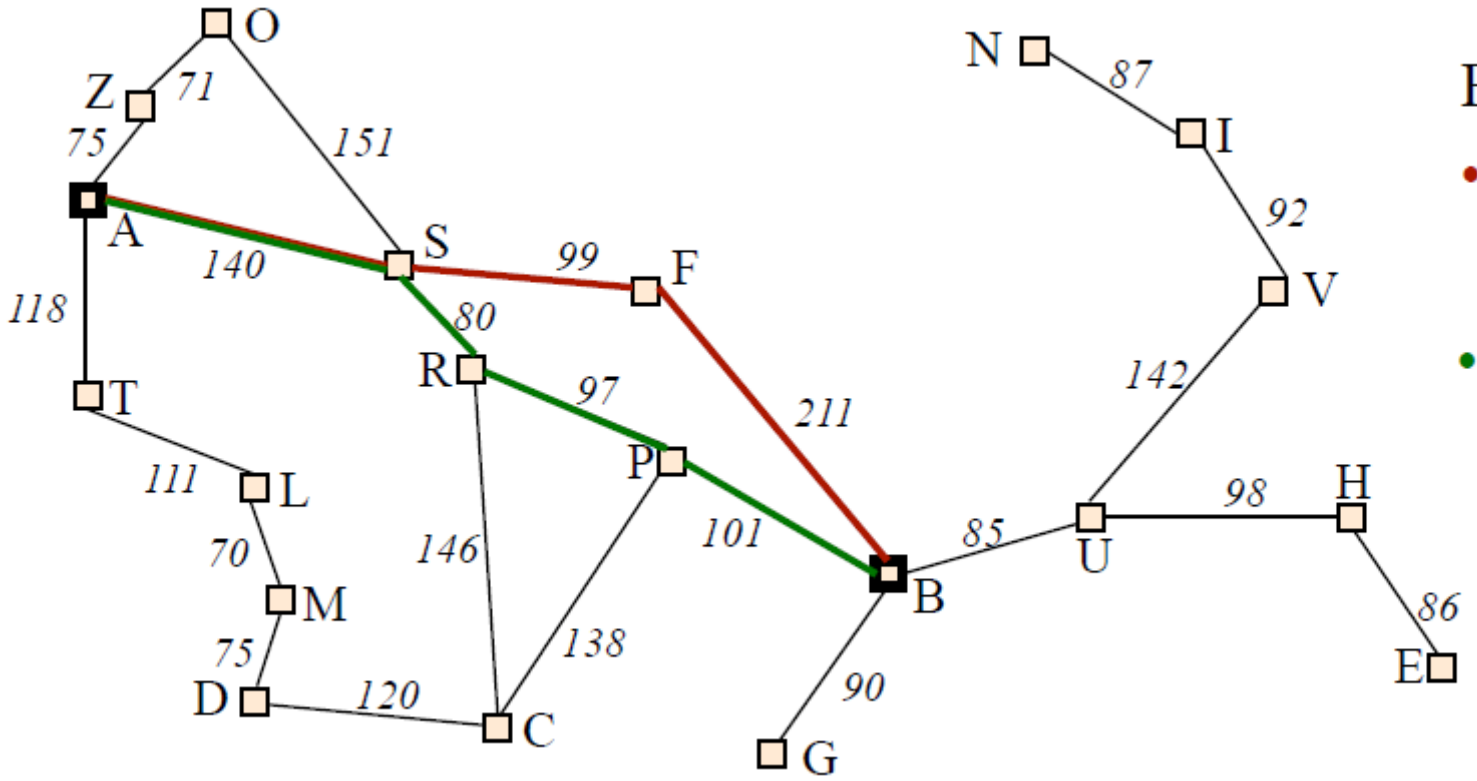
- Los métodos de búsqueda no informados encuentran el nodo meta de menor profundidad; éste puede *no* ser el nodo meta de coste mínimo

- Profundidad (*Bp*1) = 3
- Profundidad (*Bp*2) = 4

- Costo (*p*1) = 450
- Costo (*p2)* =418

Estrategias de búsqueda no informada

Ejemplo de búsqueda de trayectorias



Ejemplo:

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

Estrategias de búsqueda no informada

Practica 2

Realizar las actividades que se describen en el documento “Practica 2_ SI”.

Se debe adjuntar el reporte y el ejecutable de los programas.