Razonamiento y planificación automática

María Aurora Martínez Rey

Tema 4: Búsqueda no informada

Índice de la clase

Tema 4

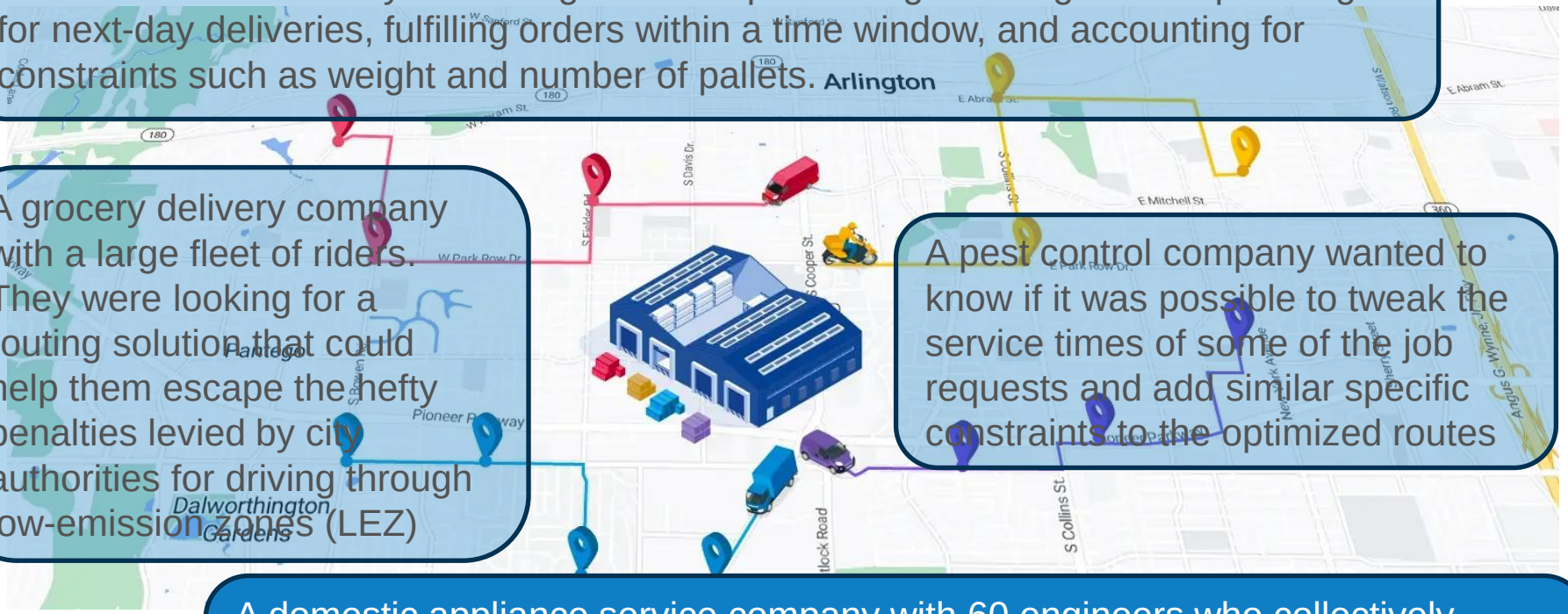
- Representación del problema para búsqueda
- Solución usando código
- Tipos de Búsqueda no informada
  - Búsqueda en amplitud
  - Búsqueda en profundidad
  - Búsqueda con coste uniforme

Resolución automática de problemas

Problemas reales de ejemplo

Fuente: https://nextbillion.ai/blog/end-to-end-delivery-route-optimization

A food delivery company with a fleet size of 22 vehicles (6 types of vehicles) that fulfills 250 deliveries/day on average. Their top 3 routing challenges were planning for next-day deliveries, fulfilling orders within a time window, and accounting for constraints such as weight and number of pallets.



A grocery delivery company with a large fleet of riders. They were looking for a routing solution that could help them escape the hefty penalties levied by city authorities for driving through low-emission zones (LEZ)

A pest control company wanted to know if it was possible to tweak the service times of some of the job requests and add similar specific constraints to the optimized routes

A domestic appliance service company with 60 engineers who collectively attend at least 250 service calls per day. The issue that they were looking to resolve was optimizing routes for their engineers for different time slots. They also wanted to reduce service response times and improve resource utilization in selected geographies/areas.

Problema de ejemplo

Una empresa tiene que asignar una ruta a un técnico que se desplaza por una ciudad. Tiene dos servicios que atender, y queremos conocer cuál es el servicio que puede realizar con mayor rapidez teniendo en cuenta los tiempos de desplazamiento entre diversos puntos de la ruta, que ya tenemos medidos y según lo que figura en la tabla siguiente. El punto de salida se denomina "S" y los dos clientes "G1" y "G2".

Está en:

Va a	S	A	B	C	D	E	F
A			1				
B	1			2			
C	4	1					
D				5		3	
E			7				
F					6	10	
G1				21			
G2					11		

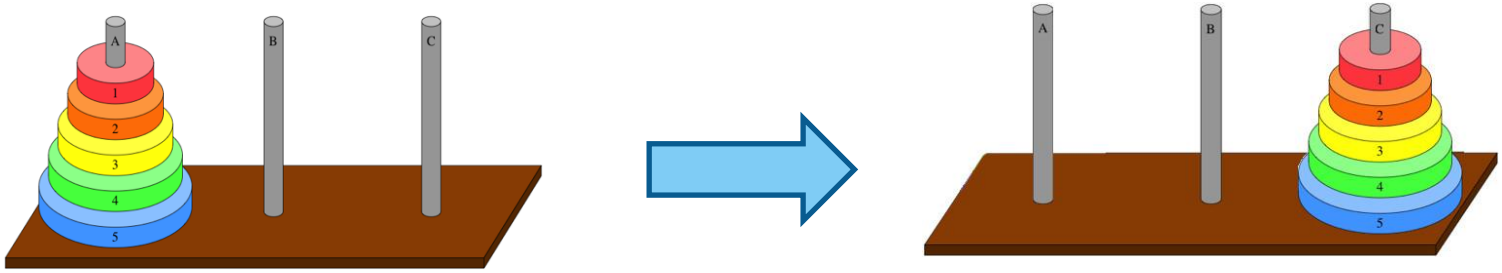
¿Para qué sirve la búsqueda?

- Encontrar un camino en un grafo
- Tomar decisiones "a futuro" entre un conjunto de opciones
- Encontrar una configuración (estado) que optimiza una función
- Buscar caminos entre puntos de un mapa
- Solucionar cualquier problema bien planteado
- Jugar a juegos de uno o más jugadores (con adversario)
- Es uno de los componentes internos de los planificadores

Recomendación: Russell & Norvig

Representación del problema

Problemas de búsqueda



- Dada una representación formal de un problema, encontrar la forma de transformar el estado inicial en un estado final deseado (puede haber una condición de estado final) utilizando para ello el conocimiento formal de las acciones (operadores) permitidos.

- Búsqueda no informada: BNI=<S,A,C> , donde:
  - S es el conjunto de estados {**s**} del problema
  - A el conjunto de acciones {a} que modifican **s** hacia un sucesor **s'**: a(**s**) → **s'**, a ∈ A, **s**,**s'** ∈ S
  - C=f(**s**,a,**s'**) el coste asociado a realizar una acción que transforma el estado **s** en uno de sus sucesores **s'**

Problema de ejemplo

Una empresa tiene que asignar una ruta a un técnico que se desplaza por una ciudad. Tiene dos servicios que atender, y queremos conocer cuál es el servicio que puede realizar con mayor rapidez teniendo en cuenta los tiempos de desplazamiento entre diversos puntos de la ruta, que ya tenemos medidos y según lo que figura en la tabla siguiente.

Está en:

Va a	S	A		B	C	D	E	F
A				1				
B	1				2			
C	4	1						
D					5		3	
E				7				
F						6	10	
G1					21			
G2						11		

Para cualquier problema:

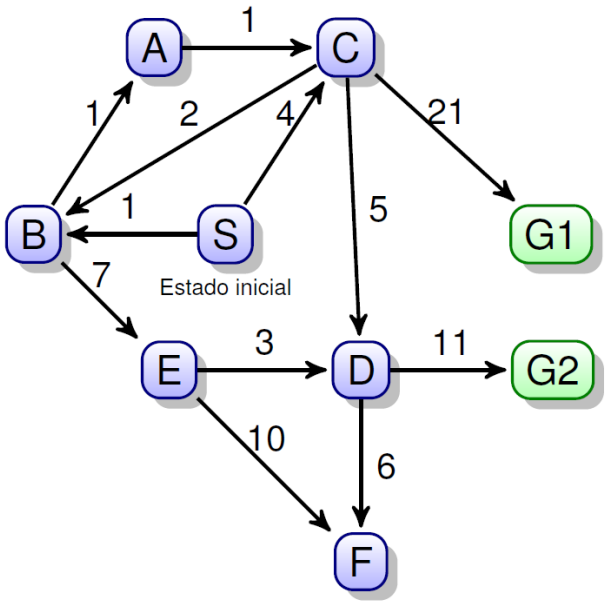
Estados: la ubicación en que está.
S= {S, A, B, … G1, G2}
Acciones: una sola, mover(?X,?Y)
Costes: los indicados en el grafo

Para un problema concreto:

Estado inicial: S
Estado final: G1 o G2

Grafo del problema

Una empresa tiene que asignar una ruta a un técnico que se desplaza por una ciudad. Tiene dos servicios que atender, y queremos conocer cuál es el servicio que puede realizar con mayor rapidez teniendo en cuenta los tiempos de desplazamiento entre diversos puntos de la ruta, que ya tenemos medidos y según lo que figura en la tabla siguiente.



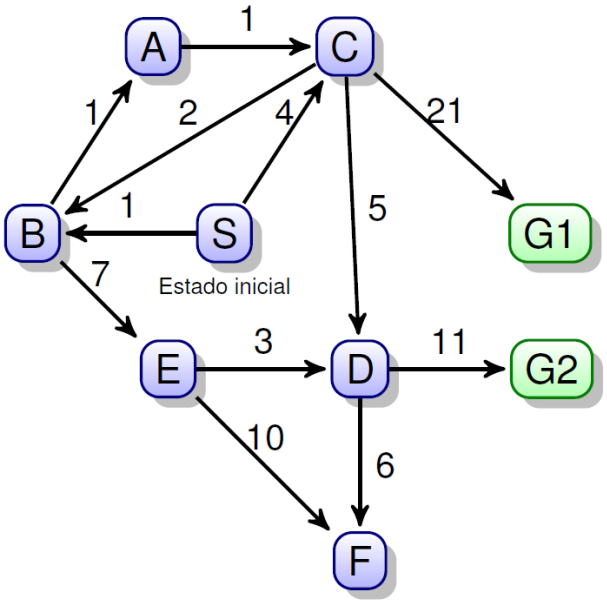
IMPORTANTE: Todo problema de este tipo puede representarse mediante un grafo, más un estado inicial, más una condición de estado objetivo.

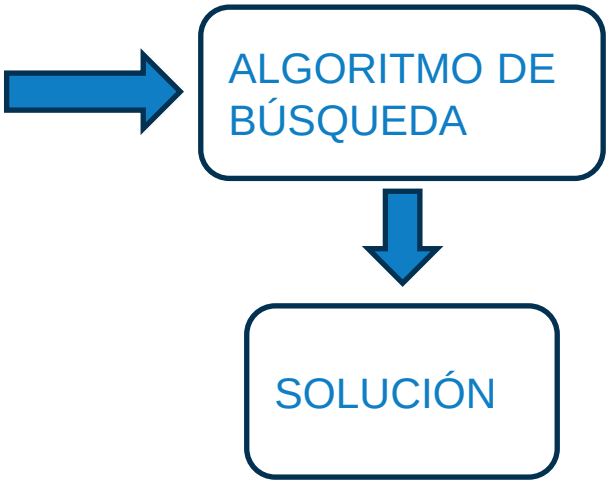
Grafo del problema:

Los nodos son estados, las flechas son la acción de movimiento, el coste viene sobre cada flecha. En verde objetivos, S es el inicio.

Forma de resolución del problema

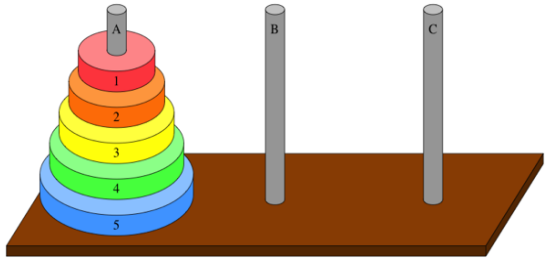
Una empresa tiene que asignar una ruta a un técnico que se desplaza por una ciudad. Tiene dos servicios que atender, y queremos conocer cuál es el servicio que puede realizar con mayor rapidez teniendo en cuenta los tiempos de desplazamiento entre diversos puntos de la ruta, que ya tenemos medidos y según lo que figura en la tabla siguiente.





Solución del problema

Algoritmos clásicos versus búsqueda




```
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}
```

ESPECÍFICO: sólo vale para las torres de Hanoi

Representación del estado específica: sólo vale para las Torres de Hanoi

Grafo del problema (estados y acciones)

GENÉRICOS (para cualquier problema)

	Sin costes		Con costes	
No informados	Búsqueda en Amplitud	Búsqueda en Profundidad	UCS	
Informados	Greedy Best First Search (GBFS)	Escalada	A*	

Algoritmo genérico de búsqueda (¡Importante!)

Es un algoritmo de búsqueda en grafos cualesquiera.


```
Input: Estado inicial S0, Estado final G

1: colaAbierta ← {S0}
2: mientras colaAbierta ≠ ∅
3:     nodo ← extraer primero de colaAbierta
4:     si meta(nodo) entonces
5:         retornar camino a nodo
6:     fin si
7:     sucesores ← expandir(nodo)
8:     para cada sucesor ∈ sucesores hacer
9:         sucesor.padre ← nodo
10:        colaAbierta ← colaAbierta ∪ sucesor
11:    fin para
12: fin mientras
13: retorna plan vacío o problema sin solución
```

Los algoritmos de búsqueda concretos que veremos serán casos particulares de este en los que se modifica la definición del paso 3 (otras posibilidades, ver libro de Rusell & Norvig)

Recordad el algoritmo general para los ejercicios (concepto de iteración, lugar en que se comprueba la meta, etc.)

Algoritmo genérico de búsqueda (¡Importante!)


```
Input: Estado inicial S0, Estado final G

1: colaAbierta ← {S0}
2: mientras colaAbierta ≠ ∅
3:     nodo ← extraer primero de colaAbierta
4:     si meta(nodo) entonces
5:         retornar camino a nodo
6:     fin si
7:     sucesores ← expandir(nodo)
8:     para cada sucesor ∈ sucesores hacer
9:         sucesor.padre ← nodo
10:        colaAbierta ← colaAbierta ∪ sucesor
11:    fin para
12: fin mientras
13: retorna plan vacío o problema sin solución
```

- Una sola solución: se detiene y entrega una única solución, pero ojo, puede haber "revisado" varias si está intentando encontrar la óptima.
- Opera por iteraciones: en cada iteración se expande un nodo.
- Memoria: al expandir un nodo se generan todos sus sucesores y estos se añaden a la llamada "lista abierta" (los que ya estaban, permanecen)
- No explora todos los caminos: en Amplitud parece que primero "hace crecer" todo el árbol y luego lo mira: no es así, esto es solo la "foto" final: va expandiendo nodos y finalizando cuando se cumple la condición

Representación y solución con código (simple-ai)

Ver cuaderno Jupyter de clase (04.2)

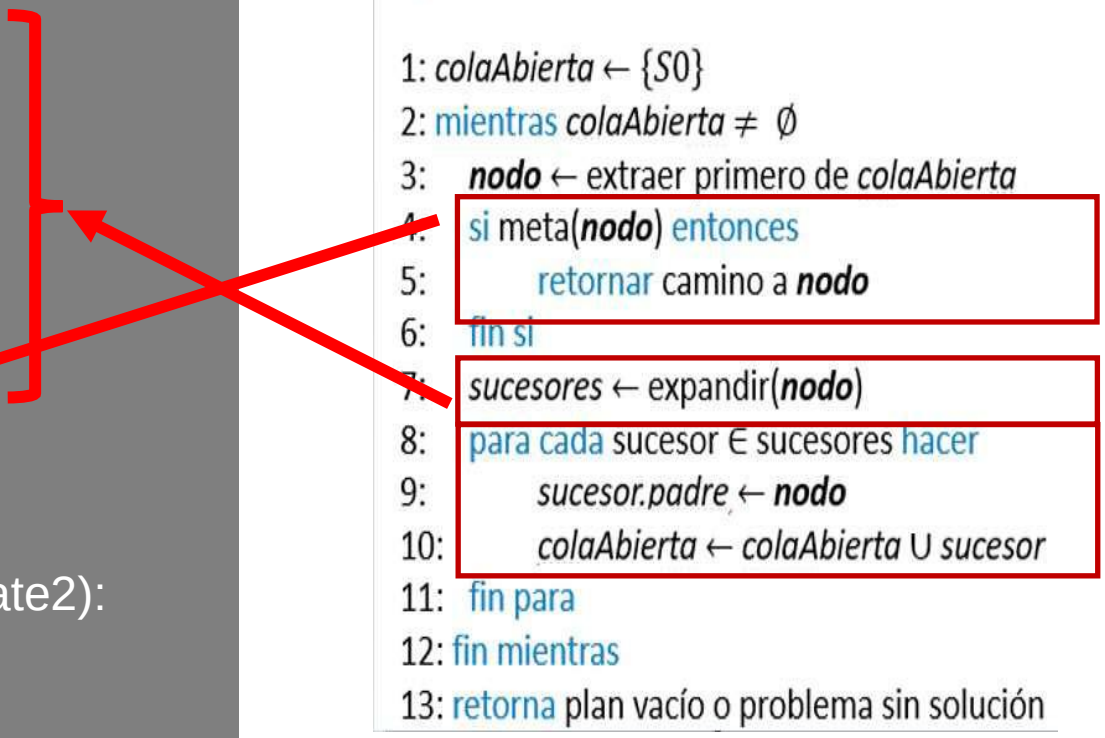
Código para resolver el problema

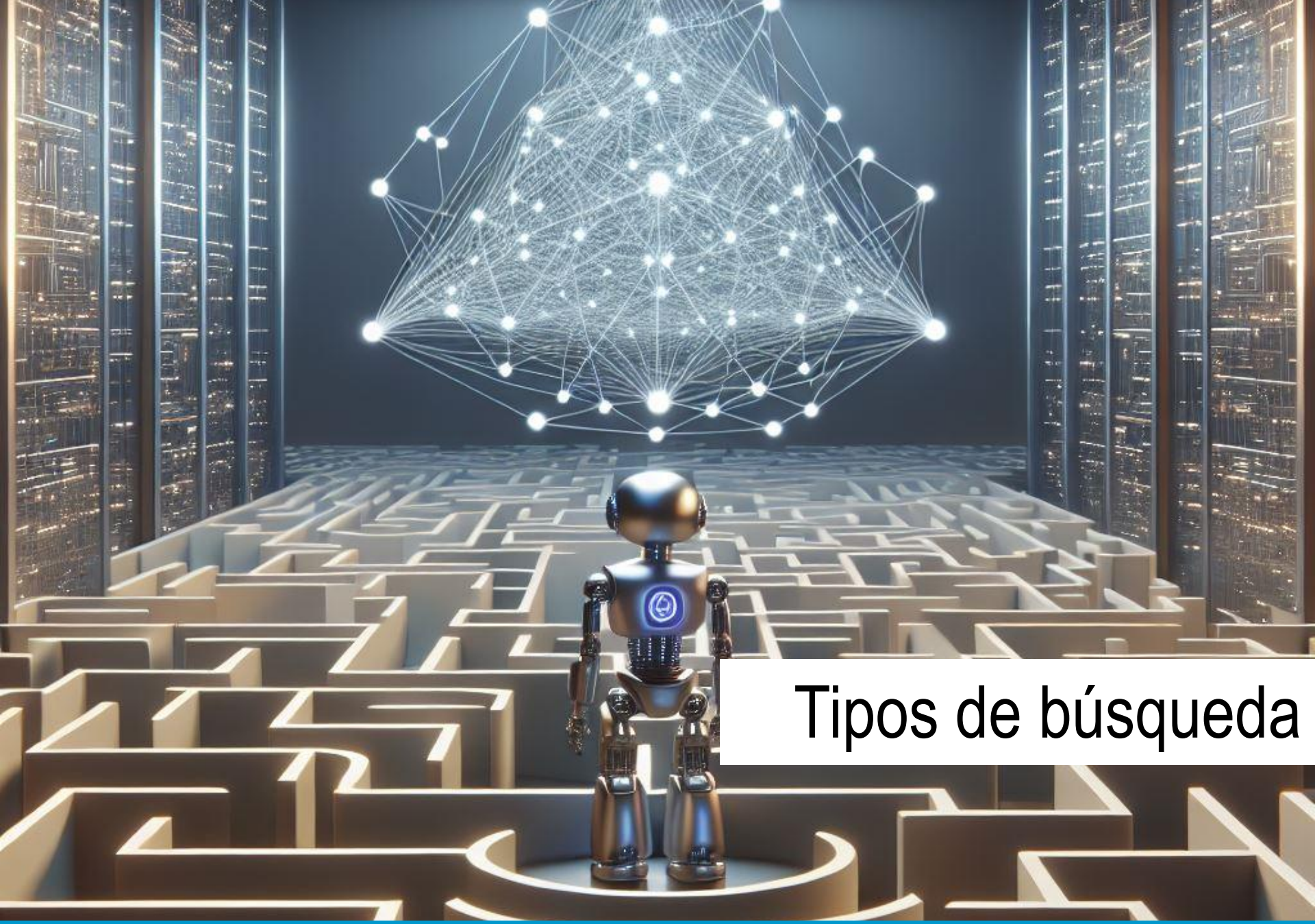
```
!pip install simpleai flask pydot graphviz
```

```
from simpleai.search import breadth_first


problem = StaticGraphSearch(problem_graph,initial,goals...)


used_viewer=ConsoleViewer()
result = breadth_first(problem, graph_search=True,viewer=used_viewer)
```

Representación del problema

```
class
StaticGraphSearch(SearchProblem):

    def actions(self, state):
      pass

    def result(self, state, action):
      pass

    def is_goal(self, state):
      pass

    def cost(self, state, action, state2):
      pass

    # Tema 5
    def heuristic(self,state):
      return 0
```





Tipos de búsqueda

Forma de realizar la búsqueda

- Offline: agentes deliberativos, las acciones se planean antes de ejecutarlas

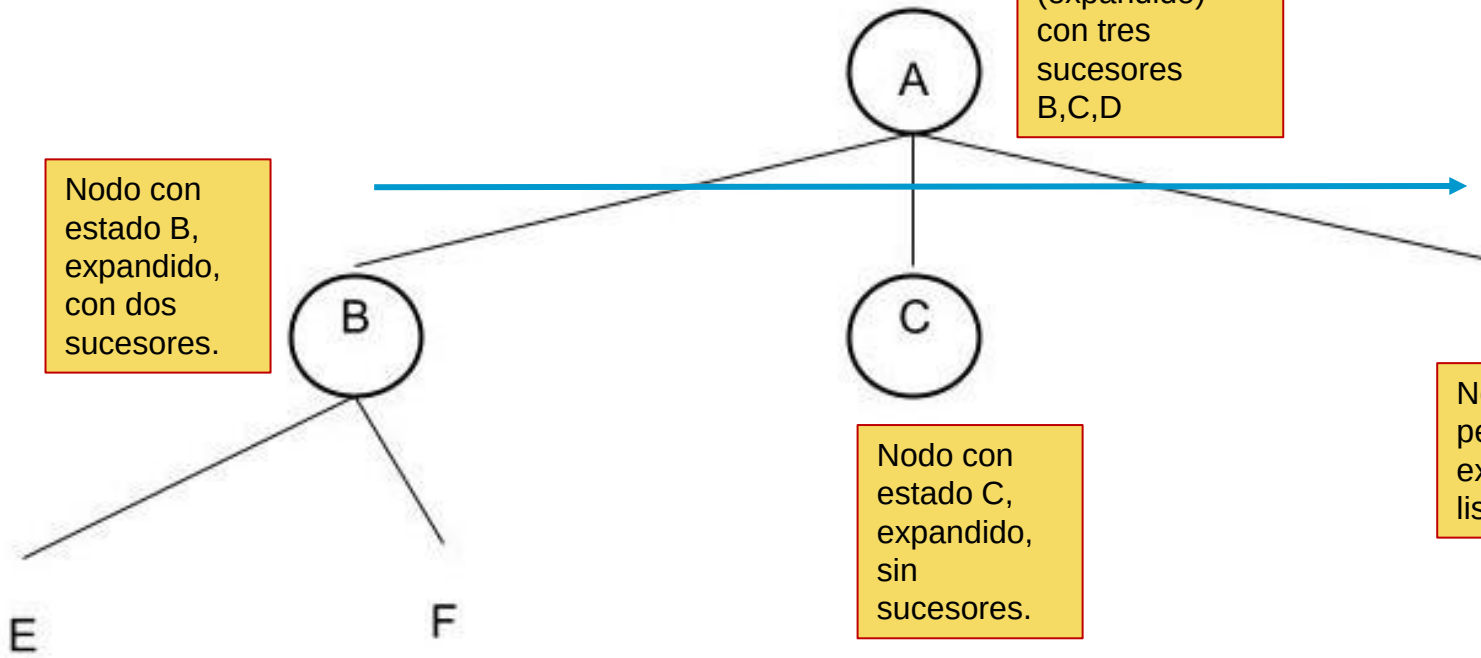
- Online: agentes reactivos, se ejecutan acciones a medida que se busca

En nuestro caso particular:

- Búsqueda offline
- Estado completamente visible (sin información oculta)
- Acciones deterministas e invariantes en el tiempo
- Función de detección de objetivo o bien estado objetivo definidos
- En general: buscaremos en el espacio de caminos y no en el de estados (que sería optimización)

Árbol de búsqueda (¡Notación para ejercicios!)

Es una forma particular de recorrer un grafo (será diferente para cada algoritmo particular)



Nodo con estado inicial A (expandido) con tres sucesores B,C,D

Nodo con estado B, expandido, con dos sucesores.

Nodo con estado C, expandido, sin sucesores.

Nodo pendiente de expandir (en lista abierta)

Nodo pendiente de expandir (en lista abierta)

Nodo pendiente de expandir (en lista abierta)

Notación: Al expandir, los sucesores los vamos anotando de izquierda a derecha, pero el algoritmo no siempre los expande en ese orden (otro algoritmo podría haber escogido D antes, y seguiría entonces por ese lado de la figura).

Factor de ramificación, profundidad, etc.

Estados: [N_{11},N_{12},N_{21},N_{22}]

Acciones (operadores): *Subir*, *Bajar*, *Derecha*, *Izquierda*

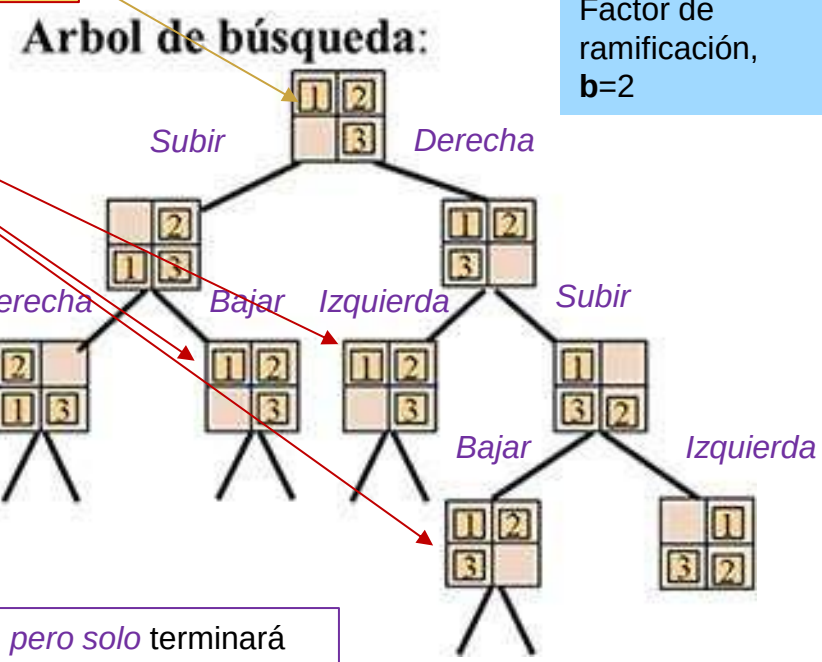
Costes (no se usan)

Nodo con estado inicial

Nodos con estado repetido

Profundidad de la solución menos profunda **d**=2

Nodo Meta



Factor de ramificación, **b**=2

Nodo:
estado + **camino recorrido** +
+ **información adicional** según el algoritmo

Profundidad máxima, **m** (para algunos algoritmos)

Nodos expandidos (hasta aquí): 4

Nodos generados (hasta aquí): 8

Se ve una solución: *Subir*, *Derecha* *pero solo* terminará cuando toque expandir ese nodo

NOTA: nuestros algoritmos ordenarán los sucesores con un orden fijo entre las acciones (no como en este árbol). Cada algoritmo decide qué nodo toca expandir con un criterio diferente (no siempre es el de más a la izquierda)

Estados repetidos

Al buscar, cada elemento del árbol se denomina nodo. En un árbol puede aparecer el mismo estado en distintos nodos (ej: dos formas distintas de llegar al mismo punto). Si no se controla, el algoritmo puede no terminar.

Estrategia: suele haber una típica para cada algoritmo, pero se podría variar.

- Ignorarlos: por extraño que parezca, algunos algoritmos no tienen problemas con esta solución debido a su propio orden de exploración o estructura del grafo.
- Evitar ciclos simples: evitando añadir el padre de un nodo al conjunto de sucesores.
- Evitar ciclos generales: de tal modo que ningún antecesor de un nodo se añada al conjunto de sucesores. (UCS, A*)
- Evitar todos los estados repetidos: no permitiendo añadir ningún nodo existente en el árbol al conjunto de sucesores. (Amplitud, profundidad)

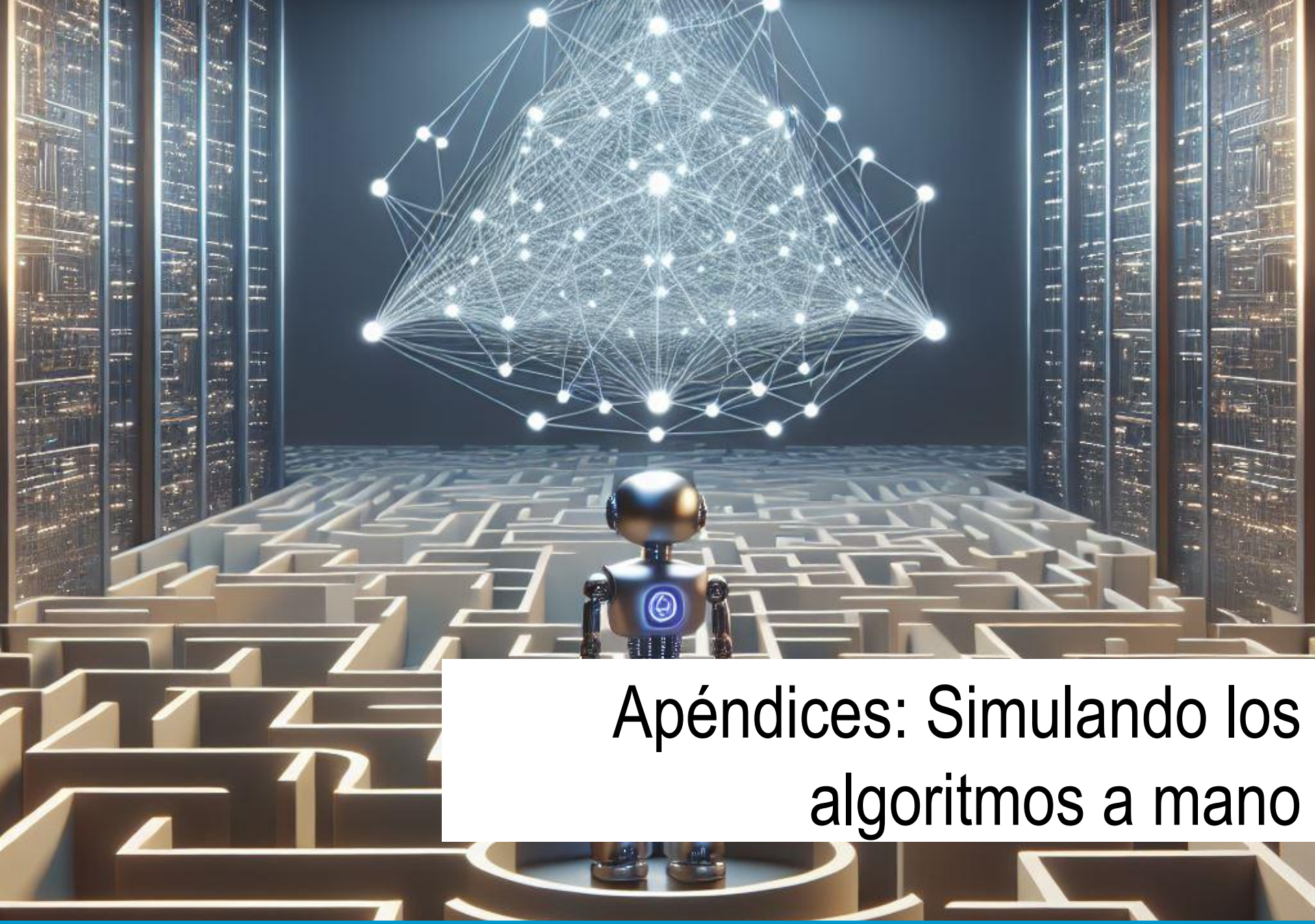
"*Algorithms that forget their history are doomed to repeat it*" – Russell & Norvig, p.82

Características de los algoritmos

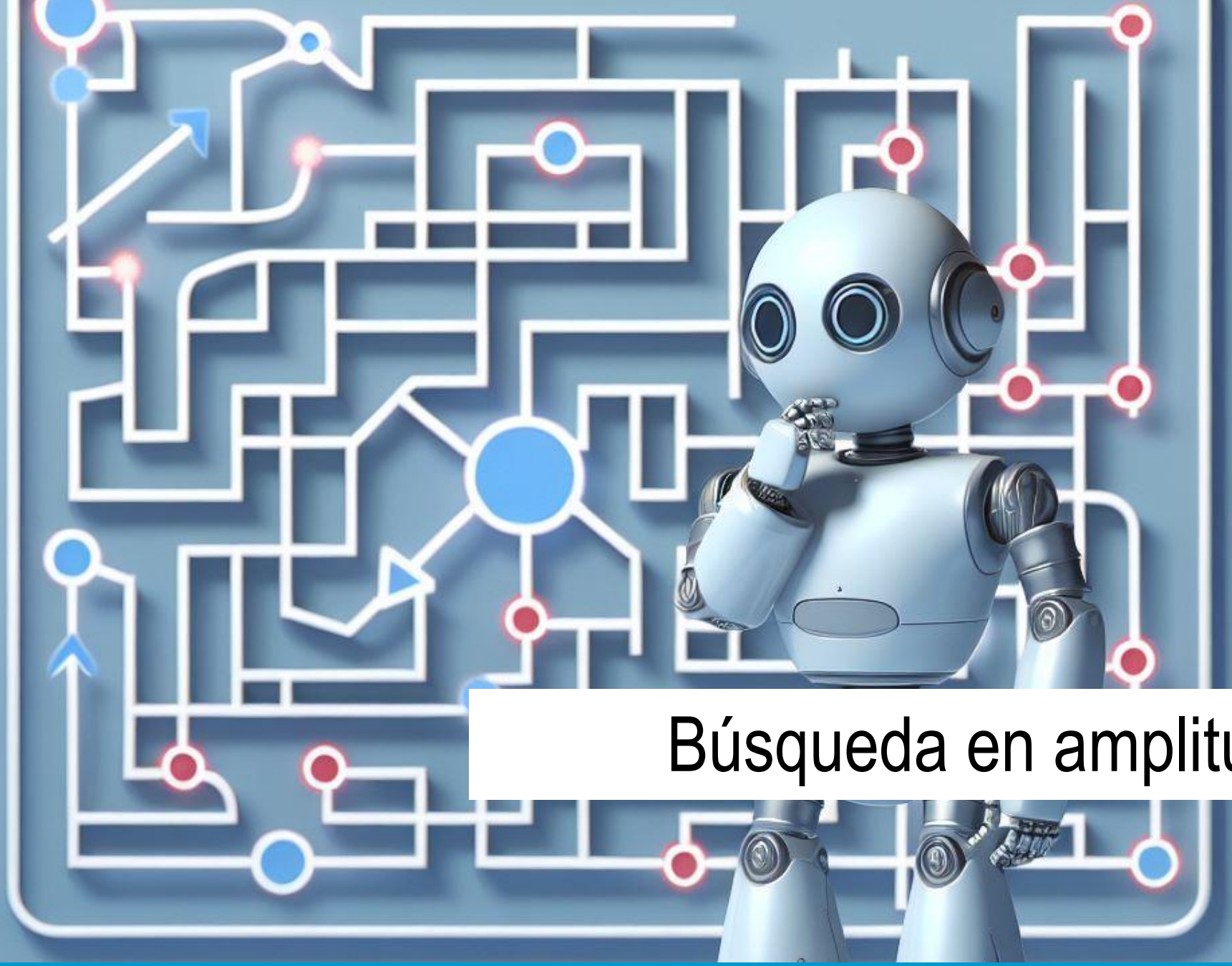
COMPLETITUD:
Al introducir el mecanismo de lista abierta, todos los algoritmos que mostramos van a ser completos (a costa de mayor complejidad espacial, es decir, memoria).
Ver Russell&Norvig para algoritmos no completos (profundidad pura)

OPTIMALIDAD DE LA SOLUCIÓN:
Sin costes: los algoritmos óptimos encuentran un solución de mínima profundidad
Con costes: los algoritmos óptimos encuentran la solución de mínimo coste

EFICIENCIA:
Medida práctica de la memoria necesaria o tiempo para la ejecución del algoritmo



Apéndices: Simulando los algoritmos a mano

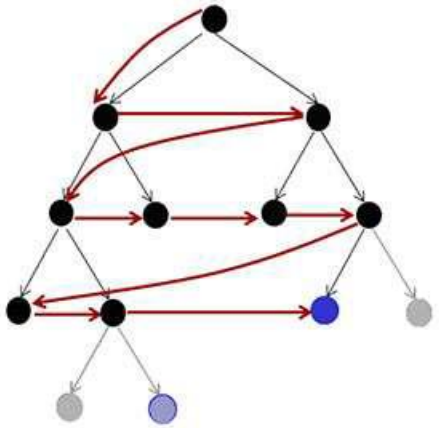


Búsqueda en amplitud

Búsqueda en Amplitud

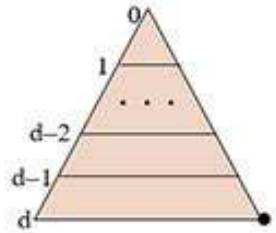
Usa el Algoritmo Genérico con las siguientes peculiaridades:

- Para añadir nuevos sucesores, lo haremos al final de la lista abierta.
- Por su parte, la lista abierta funciona como cola (insertando al final y recuperando al inicio), lo que conlleva que siempre se expandan primero aquellos nodos más antiguos (es decir, los menos profundos).
- Adicionalmente, evita todos los nodos que se han generado previamente.
- Termina en cuanto toca expandir un nodo meta



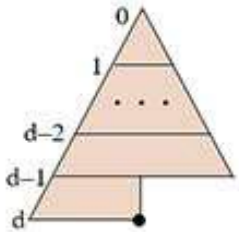
En rojo, orden en que se van expandiendo. Sólo se termina (nodo azul) cuando corresponde expandirlo.

Peor caso



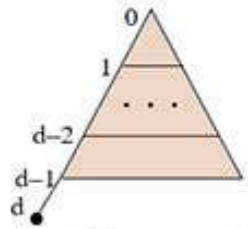
$1+b+\ldots+b^{d-1}+b^{d} \in O(b^{d})$

Caso medio



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

Mejor caso



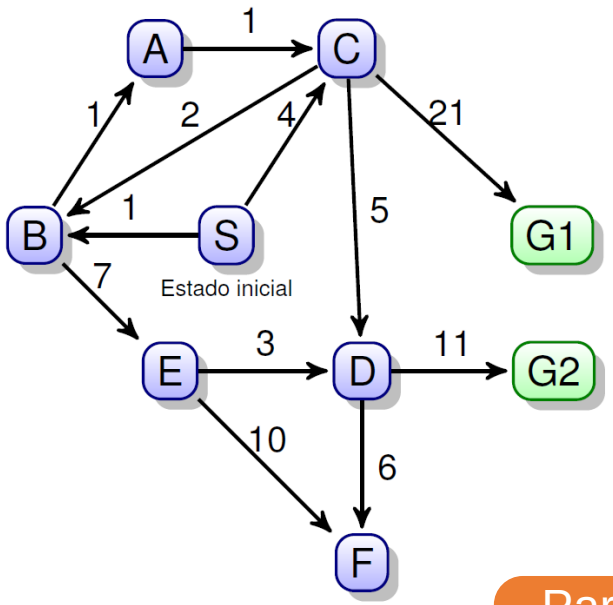
$1+b+\ldots+b^{d-1}+1 \in O(b^{d})$

Óptimo en número de acciones de la solución (profundidad)

Complejidad exponencial en espacio y tiempo

Ejercicio Búsqueda en Amplitud

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos. Al generar nuevos nodos, hacedlo por orden alfabético



Al generar nodos seguimos el orden alfabético indicado

Expandido	Genera	Abierta

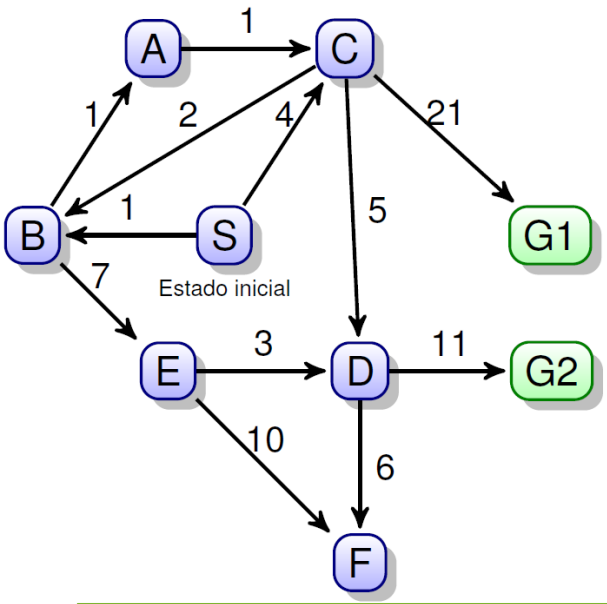
Iteraciones: 1 … 7

No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

Para examen: preguntamos por la tabla, completa, o por algún paso, o ponemos el árbol y pedimos completar la información en algún nodo (ejemplo: los sucesores)

Ejercicio Búsqueda en Amplitud (solución)

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos. Al generar nuevos nodos, hacedlo por orden alfabético



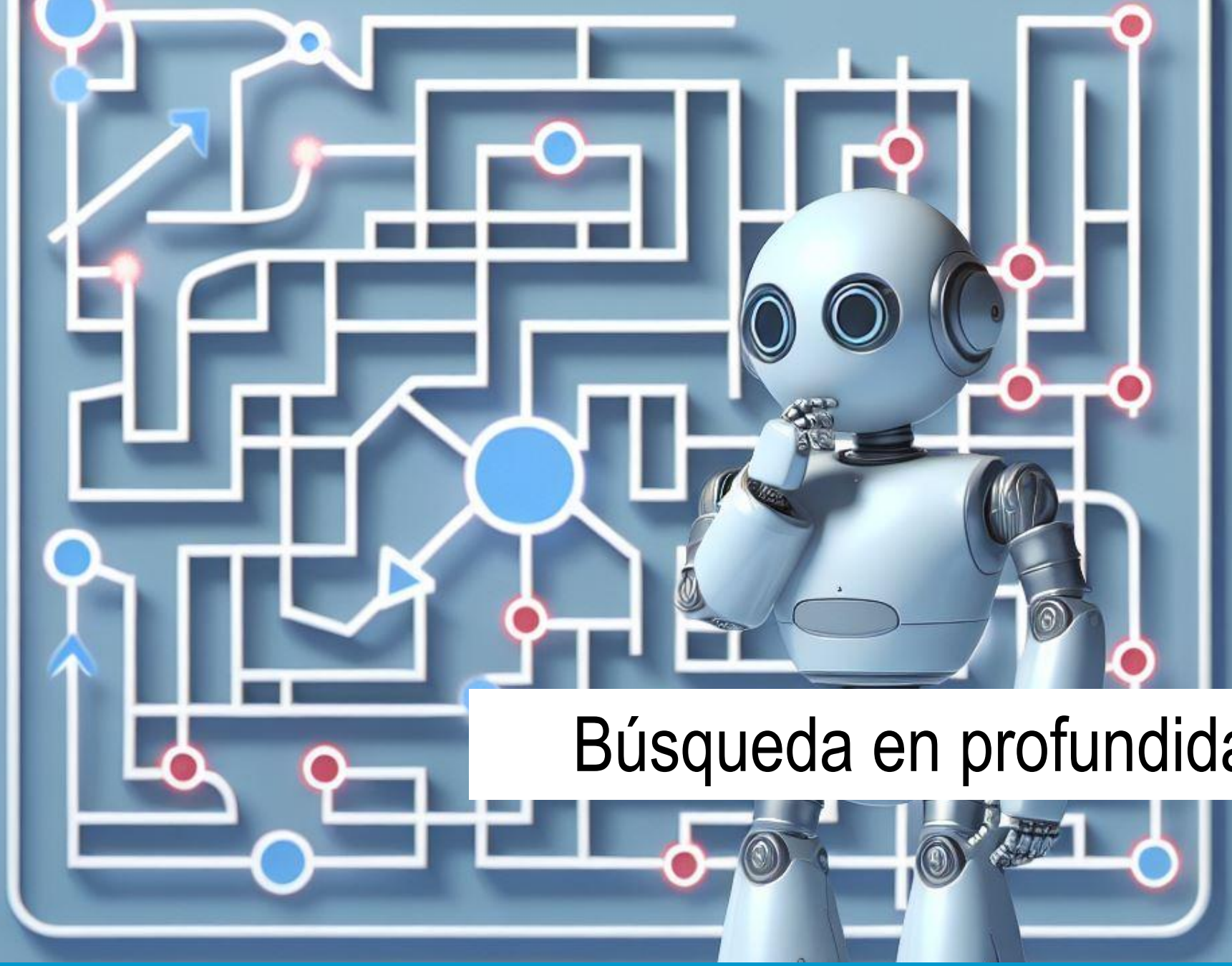
Al generar nodos seguimos el orden alfabético indicado

Expandido	Genera	Abierta
S	B, C	**B (SB)** , **C (SC)**
B (SB)	A, E	C (SC), **A (SBA)**, **E (SBE)**
C (SC)	D, G1 (B no se genera por repetido)	A (SBA), E (SBE) , **D (SCD)**, **G1 (SCG1)**
A (SBA)	(C no se genera por repetido)	E (SBE), D(SCD) , G1 (SCG1)
E (SBE)	F (D no se genera por repetido)	D (SCD) , G1 (SCG1) , **F (SBEF)**
D (SCD)	G2 (F no se genera por repetido)	G1 (SCG1), F (SBEF), **G2 (SCDG2)**
G1 (SCG1)	FIN	

1 Iteraciones 7

No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

Solución: *SC,CG1 .* Tenemos que retrazar a mano el camino porque no pintamos los arcos en la tabla, en el código se irían guardando. Se llega al objetivo **G1**.
Longitud solución: 2**, Expandidos**: 6 (columna 1), **Generados**: 8 (columna 2)

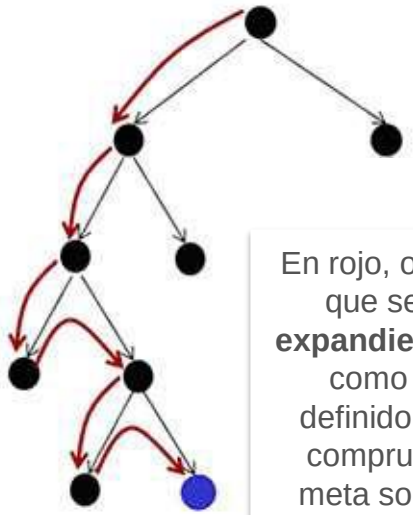


Búsqueda en profundidad

Búsqueda en Profundidad

Usa el Algoritmo Genérico con las siguientes peculiaridades:

- Los nuevos sucesores se añaden al inicio de la lista abierta
- La lista abierta funcionará como una pila (insertando al principio y extrayendo también del principio) y siempre extraeremos el nodo más profundo. Al guardar todos los sucesores de un nodo expandido en abierta, se permite la «vuelta atrás» o backtracking (completo).
- Adicionalmente, evita todos los nodos que se han generado previamente.
- Termina en cuanto toca expandir un nodo meta



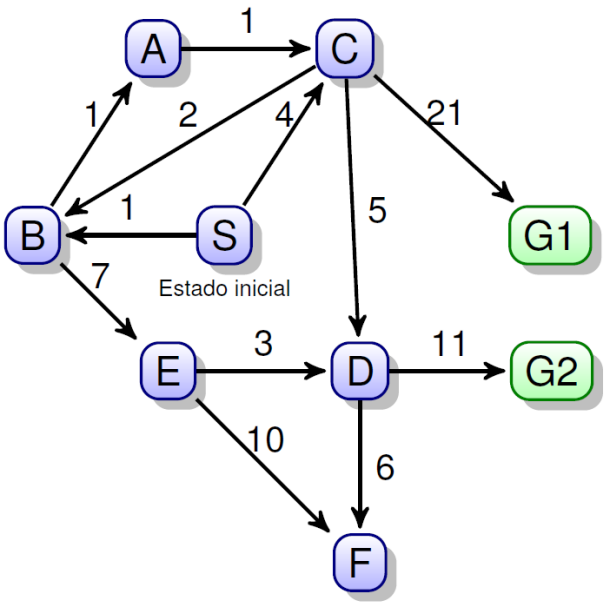
En rojo, orden en que se van **expandiendo**. Tal como está definido, no se comprueba la meta sobre los nodos laterales

No es óptimo en ningún sentido (podría haber mejores caminos)

Complejidad lineal en espacio pero exponencial en tiempo

Ejercicio Búsqueda en Profundidad

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos. Al generar nuevos nodos, ahora asumamos orden inverso al alfabético



Al generar nodos seguimos el orden indicado

Expandido	Genera	Abierta

1

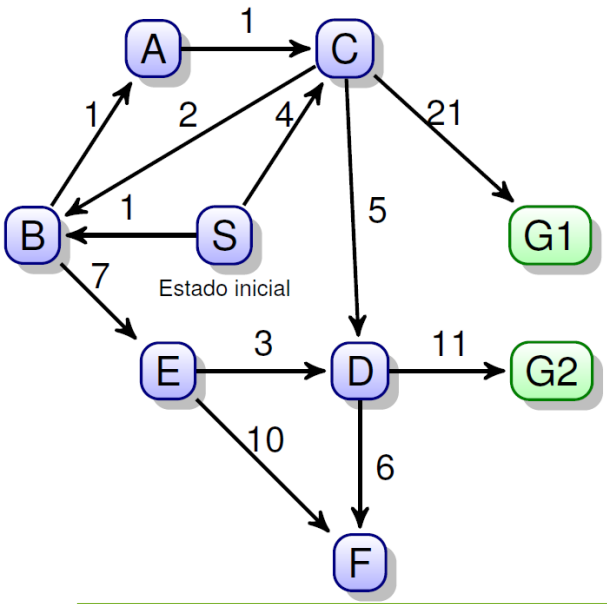
Iteraciones

6

No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

Ejercicio Búsqueda en Profundidad (solución)

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos. Al generar nuevos nodos, ahora asumamos orden inverso al alfabético



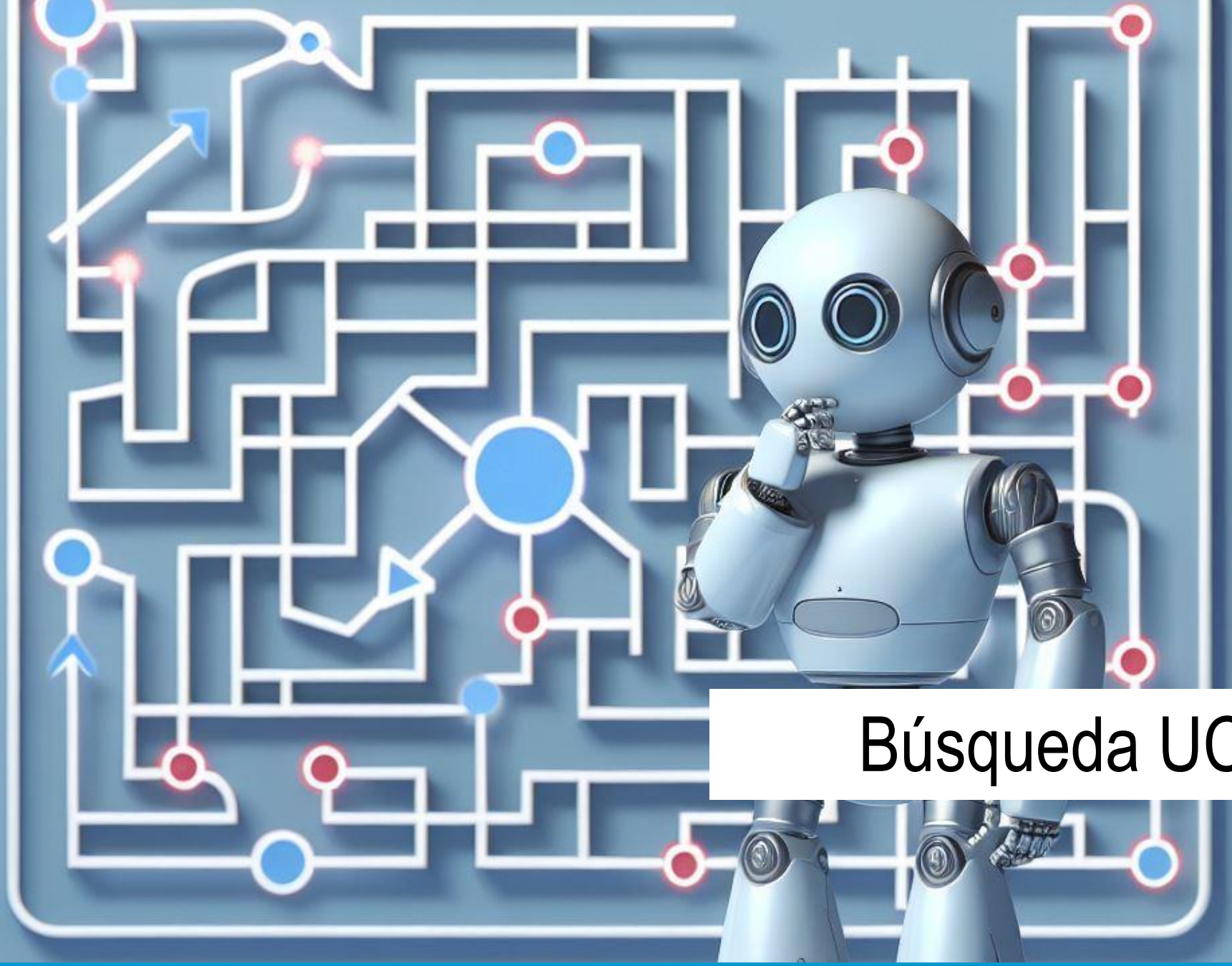
Al generar nodos seguimos el orden indicado

Expandido	Genera	Abierta
S	C, B	**B (SB)** , **C (SC)**
B (SB)	E, A	**A (SBA)** , **E (SBE)** , C (SC)
A (SBA)	(C no se genera por repetido)	E (SBE), C (SC)
E (SBE)	F , D	**D (SBED)**, **F (SBEF),** C (SC)
D (SBED)	G2 (F no se genera por repetido)	**G2 (SBEDG2),** F (SBEF), C (SC)
G2 (SB EDG2)	FIN	

Iteraciones: 1 … 6

No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

Solución: *SB, BE, ED, DG2 .* Tenemos que retrazar a mano el camino porque no pintamos los arcos en la tabla, en el código se irían guardando. Se llega al objetivo **G2**.
Longitud solución: 4**, Expandidos**: 5 (columna 1), **Generados**: 7 (columna 2)



Búsqueda UCS

Búsqueda con coste uniforme (UCS)

Usa el Algoritmo Genérico con las siguientes peculiaridades:

- En cada nodo anotamos la suma de los costes de las acciones que me han llevado al mismo, que denominamos *g*(N) (diferente según el camino)
- Los nuevos sucesores se añaden ordenados en función de dicho coste total.
- Se insertan en la lista Abierta nodos si y solo si:

a) su estado no esté en ella; o
b) estando ya en la lista, el coste *g* del nodo es menor (y reemplaza al anterior).

- La lista abierta funcionará como cola de prioridad ordenada según *g*.
- Adicionalmente, se controlan los estados repetidos con la política *evita ciclos generales* (no escribe nodos que se han generado previamente en el camino al nodo que se expande).
- Termina en cuanto toca expandir un nodo meta

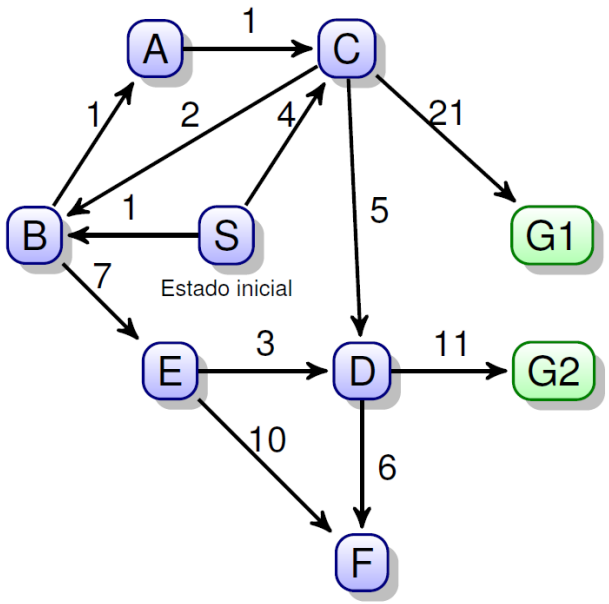
Óptimo en coste

Complejidad exponencial en espacio y tiempo

Notación: Como ahora el nodo que se expande ya no tiene que ver con el orden en que se generó, puede continuar por cualquier rama de las que haya creado (no necesariamente la de la izquierda).

Búsqueda con coste uniforme (UCS)

S: estado inicial, G1 y G2 metas, Costes en los arcos. Al generar nuevos nodos, el orden es alfabético, y también se usa este orden para empates en Abierta **(a igualdad de g)**



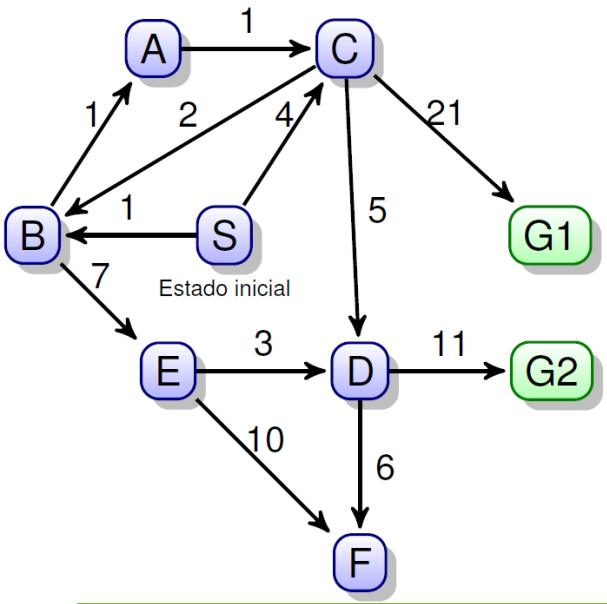
Expandido	Genera	Abierta	Cerrada

1

8

Búsqueda con coste uniforme (UCS) (solución)

S: estado inicial, G1 y G2 metas, Costes en los arcos. Al generar nuevos nodos, el orden es alfabético, y también se usa este orden para empates en Abierta **(a igualdad de g)**



Expandido	Genera	Abierta	Cerrada
S	B (g=1), C(g=4)	B (g=1), C(g=4)	S
B (g=1)	A (g=1+1=2), E (g=1+7=8)	A (g=2), C(g=4), E(g=8)	S, B (g=1)
A (g=2)	C (g=2+1=3)	C(g=3), ~~C(g=4),~~ E(g=8)	S, B (g=1), A(g=2)
C (g=3)	D (g=3+5=8), G1 (g=3+21=24) (B repetido en el camino a C)	D(g=8), E(g=8), G1 (g=24)	S, B (g=1), A(g=2), C(g=3)
D (g=8)	F (g=8+6=14), G2 (g=8+11=19)	E(g=8), F(g=14), G2 (g=19), G1 (g=24)	S, B (g=1), A(g=2), C(g=3), D(g=8)
E (g=8)	D (g=8+3=11) (peor que el de la lista cerrada), F (8+10=18) (peor)	F(g=14), G2 (g=19), G1 (g=24)	S, B (g=1), A(g=2), C(g=3), D(g=8),E(g=8)
F (g=14)	No tiene sucesores	G2 (g=19), G1 (g=24)	S, B (g=1), A(g=2), C(g=3), D(g=8),E(g=8),F(g=14)
** G2 (g=19)	FIN		

1

8

Solución: *SB(1),BA(1),AC(1),CD(5),DG2(11) .* Tenemos que retrazar a mano el camino de coste 19 porque no pintamos los arcos en la tabla, en el código se irían guardando.
Coste: 1+1+1+5+11=19, **Expandidos**: 7, **Generados**: 11 (aunque algunos nunca se insertaron en la lista abierta)

Comparativa de algoritmos de búsqueda no informada

📄 Comparativa de algoritmos de búsqueda no informada

Característica	Amplitud	Profundidad	UCS (Uniform Cost Search)
Completitud	✅ Sí (si el espacio es finito)	❌ No (puede quedar atrapado en ramas)	✅ Sí (si los costes son positivos)
Optimalidad	✅ Sí (mínima profundidad)	❌ No (puede encontrar soluciones largas)	✅ Sí (mínimo coste acumulado)
Estrategia	FIFO (cola)	LIFO (pila)	Menor coste acumulado (cola de prioridad)
Evita ciclos	✅ Sí (control de repetidos)	🔸 Opcional (mejor si se controla)	✅ Sí (requiere control de repetidos)
Complejidad temporal	$O(b^d)$	$O(b^m)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$
Complejidad espacial	$O(b^d)$	$O(m)$ sin repetidos	Alta: depende del número de caminos abiertos
Cuando usarlo	Soluciones cercanas y entornos finitos	Espacios muy grandes, poca memoria	Cuando hay costes variables y se busca óptimo

> ***b: factor de ramificación***
> ***d: profundidad de la solución***
> ***m: profundidad máxima del espacio***
> ***C*: coste de la solución óptima, ϵ: mínimo coste de acción***

Resumen y conceptos clave

- ¿Sabemos representar estados y operadores para un problema de búsqueda?
- ¿Conocemos los conceptos de: estado, nodo, meta (nodo o función), factor de ramificación, profundidad?
- ¿Sabemos cómo funciona el algoritmo de búsqueda en amplitud y en profundidad?¿en qué coinciden y en qué se diferencian?
- ¿Sabemos cómo funciona el algoritmo de coste uniforme?
- Examen: ¡importante! Hacerlos como tabla (en este formato), o completar algo, o ver si algún paso es correcto, cuáles son los sucesores de un nodo, qué nodo se escogería en UCS, etc.

www.unir.net