Razonamiento y planificación automática

Alejandro Cervantes Rovira

Tema 4: Búsqueda no informada

Índice de la clase

Tema 3

- Prolog

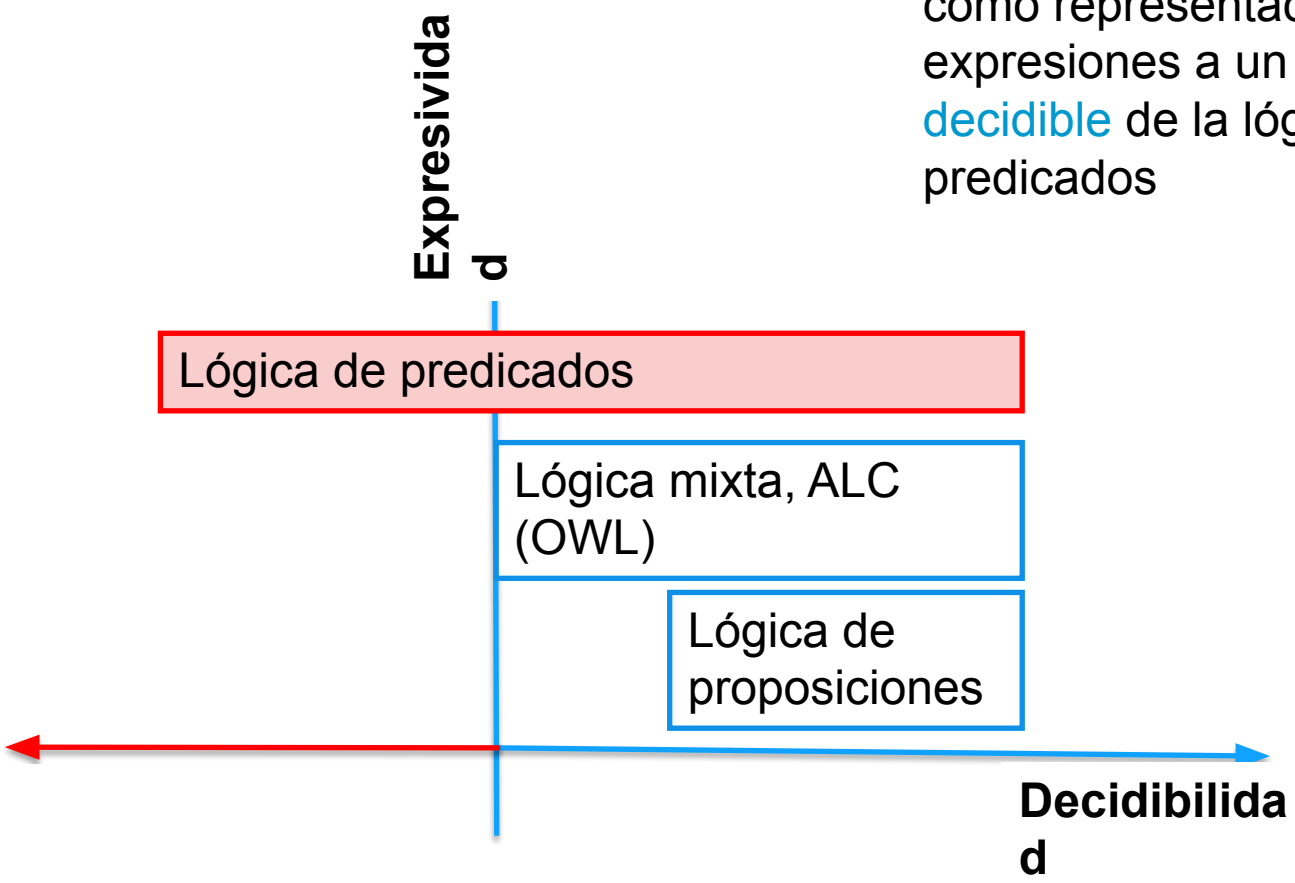
Tema 4

- Problemas de búsqueda
- Búsqueda no informada
  - Búsqueda en amplitud
  - Búsqueda en profundidad
  - Búsqueda con coste uniforme

Lógica y Prolog

Expresividad vs decidibilidad

Los sistemas que usan lógica como representación limitan las expresiones a un subconjunto decidible de la lógica de predicados



Lógica multivaluada y difusa

La lógica multivaluada

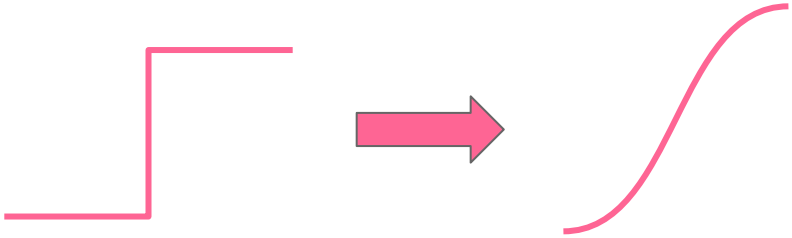
- permite valores intermedios (grande, tibio, lejos, pocos, muchos, etc.)
- se emplean más de dos valores de verdad para describir conceptos que van más allá de lo verdadero y lo falso
- ofrecen herramientas conceptuales que hacen posible describir formalmente la información difusa, vaga o incierta.

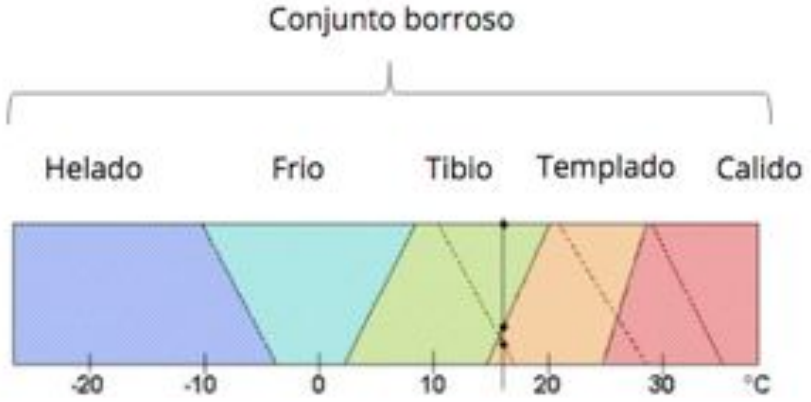
La lógica difusa (también llamada lógica borrosa) es una lógica multivaluada que permite representar matemáticamente la incertidumbre y la vaguedad, proporcionando herramientas formales para su tratamiento. El término «lógica difusa» aparece por primera vez en 1974.

Lógica difusa

Sistema matemático de valores continuos

Se reduce la cantidad de conocimiento previo





Prolog

Para mostrar un ejemplo que contiene elementos de un lenguaje de primer orden vamos a trabajar sobre la siguiente base de conocimientos:

Hechos:

1. Atlanta se encuentra en Georgia.
2. Houston y Austin se encuentran en Texas.
3. Toronto se encuentra en Ontario.

Que, usando el predicado **located_in**, podemos representar con las siguientes clausulas:

```
located_in(atlanta,georgia). % Clause 1
located_in(houston,texas).   % Clause 2
located_in(austin,texas).    % Clause 3
located_in(toronto,ontario). % Clause 4
```

Reglas:

1. Lo que está en Georgia o Texas, también está en USA.
2. Lo que está en Ontario, también está en Canadá.
3. Lo que está en USA o Canadá, también está en Norte América.

Que podemos representar con las siguientes clausulas (**geo.pl**):

```
located_in(X,usa) :- located_in(X,georgia).          % Clause 5
located_in(X,usa) :- located_in(X,texas).            % Clause 6
located_in(X,canada) :- located_in(X,ontario).       % Clause 7
located_in(X,north_america) :- located_in(X,usa).    % Clause 8
located_in(X,north_america) :- located_in(X,canada). % Clause 9
```

Observa que al estar trabajando con predicados que sí reciben argumentos ya no es necesario usar la directiva vista en el ejemplo anterior.

Vamos a ver cuál es el árbol de deducción que sigue Prolog para resolver la siguiente clausula:

```
located_in(toronto,north_america).
```

Base de hechos

Base de reglas (**implicaciones**). Variables en mayúsculas

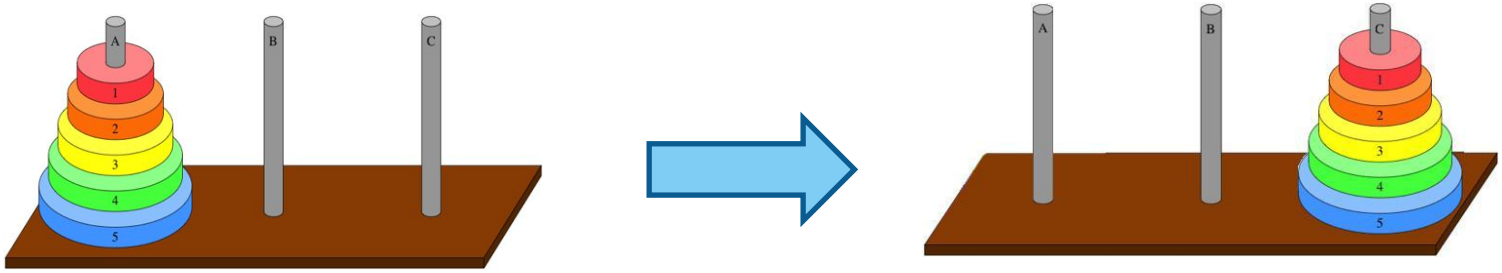
PROLOG online: https://swish.swi-prolog.org/

Consulta

Fuente: apuntes de clase

El 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'**

Tipos de búsqueda

- No informada: tal como está definida hasta ahora
- Informada: se dispone de un criterio adicional de selección que guía la búsqueda hacia un objetivo

- Sin coste: se ignora el término de coste
- Con coste uniforme: todas las acciones tienen el mismo coste
- Con costes: la función de coste existe, puede depender de uno, dos, o tres de los elementos (estado inicial, acción, o estado final)

Acciones (tb llamadas operadores)

- REGLAS: CONDICIONES → CONSECUENTE

- Ejemplo: Si un disco D1 no tiene nada encima, y en otro eje E2 el disco superior es de tamaño mayor (o no hay disco), se puede mover el disco incial del primer eje al segundo.

Torres de Hanoi

- R1: mover sobre otro disco o mesa. La constante **nada** es como un "falso disco" que usamos para saber si estamos moviendo el disco superior (habría que añadirlo a la representación):

```
  en(?E1,?D1) ∧ sobre(?D1,nada) ∧ sobre(?D2,?D1) ∧ en(?E2,?D3) ∧
  sobre (?D3,nada) ∧ tam(?D1,?T1) ∧ tam(?D3,?T3) ∧ (> ?T3 ?T1)
→
  -en(?E1,?D1), -sobre(?D3,nada),
  en(?E2,?D1), sobre(?D3,?D1), sobre(?D2,nada)
```

Otra regla similar trataría el caso de que se mueva a un eje ?E2 que no tiene ningún disco (puede haber condiciones negadas)

Forma de realizar la búsqueda

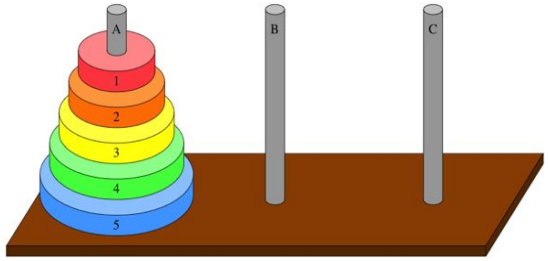
- Offline: agentes deliberativos, las acciones se planean antes de ejecutarlas

- 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

Algoritmos 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)

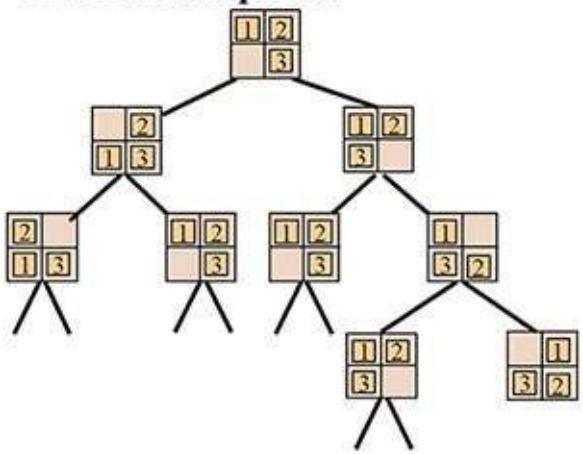
	Sin costes	Con costes
No informados	Búsqueda en Amplitud Búsqueda en Profundidad	UCS
Informados	Escalada	A*

Algoritmo genérico de búsqueda


```
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
```

Arbol de búsqueda:



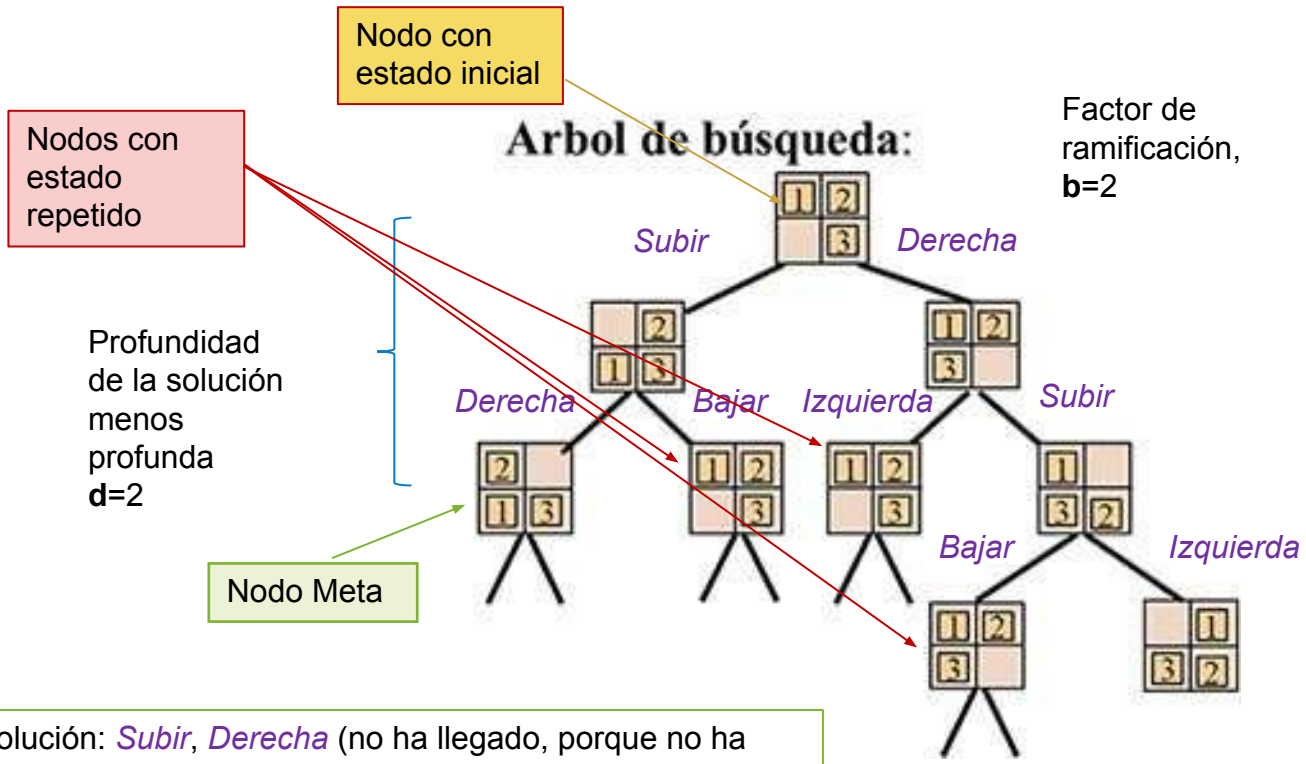
Recordad el algoritmo general para los ejercicios (concepto de iteración, lugar en que se comprueba la meta, etc.)

Árbol de de búsqueda

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

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

Costes (no se usan)



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

Solución: *Subir*, *Derecha* (no ha llegado, porque no ha **expandido** aún ningún nodo Meta)

NOTA: nuestros algoritmos ordenarán los sucesores siempre de la misma forma (no como en este árbol), y empezarán siempre a expandir por la opción que queda más a la izquierda

Estados repetidos

Al buscar, cada elemento del árbol es un nodo. En un árbol puede aparecer el mismo estado en distintos nodos (ej: dos formas distintas de llegar al mismo punto)

Estrategia: suele haber una típica para cada algoritmo, pero se puede adaptar

- 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.
- Evitar todos los estados repetidos: no permitiendo añadir ningún nodo existente en el árbol al conjunto de sucesores.

"*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)

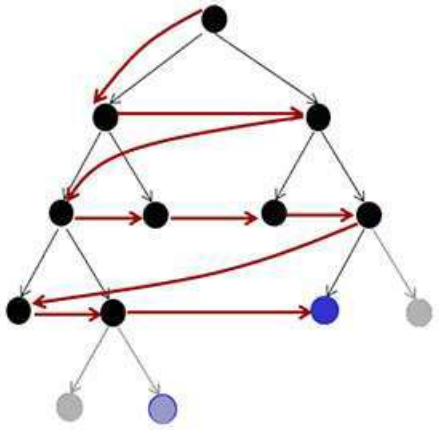
OPTIMALIDAD:
Sin costes: los algoritmos óptimos encuentran un solución de mínima profundidad
Con costes: los algoritmos óptimos encuentran la solución de mínimo coste

Búsqueda no informada

Búsqueda en Amplitud

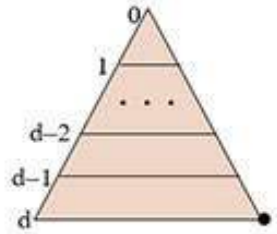
Usa el Algoritmo Genérico con las siguientes peculiaridades:

- 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



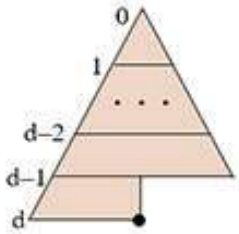
En rojo, orden en que se van expandiendo. Sólo se termina (nodo azul) cuando corresponde expandirlo.

Peor caso



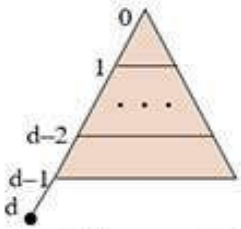
$1+b+\ldots+b^{d-1}+b^{d} \in O(b^{d})$

Caso medio



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

Mejor caso



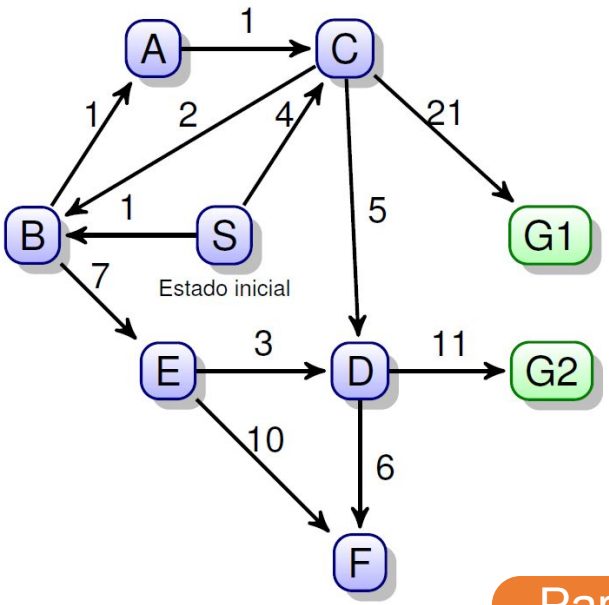
$1+b+\ldots+b^{d-1}+1 \in O(b^{d})$

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 expandir, el orden es alfabético



Al generar nodos seguimos el orden alfabético indicado

Expandido	Genera	Abierta

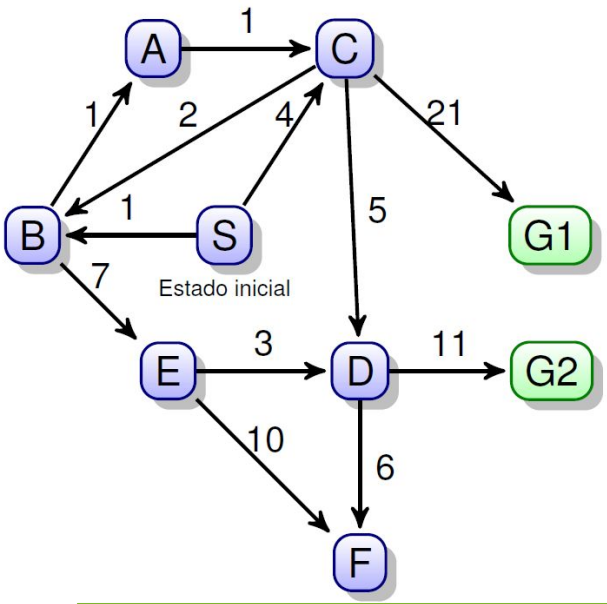
Iteraciones: 1 … 7

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

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos.
Al expandir, el orden es alfabético



Al generar nodos seguimos el orden alfabético indicado

Expandido	Genera	Abierta
S	B, C	B, C
B	A, E	C, A, E
C	D, G1 (anotamos que llegamos a G1 desde C)	A, E, D, G1
A	(C no se genera por repetido)	E, D, G1
E	F (D no se genera por repetido)	D, G1, F
D	G2 (F no se genera por repetido)	G1, F, G2
** G1	FIN	

Iteraciones: 1 … 7

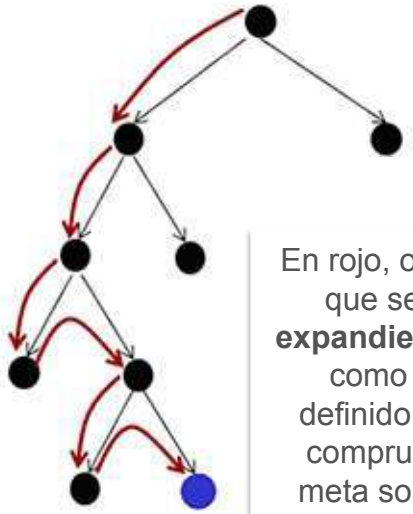
No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

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

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



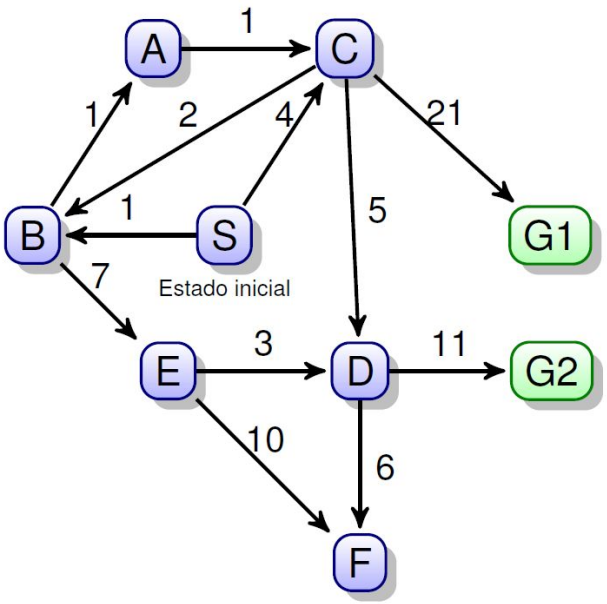
En rojo, orden en que se van **expandiendo**. Tal como está definido, no se comprueba la meta sobre los nodos laterales

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 expandir, asumamos orden inverso al alfabético



Al generar nodos seguimos el orden indicado

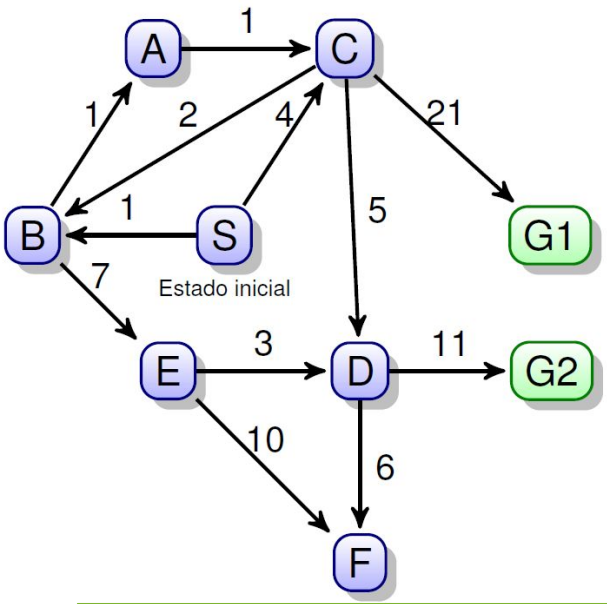
Expandido	Genera	Abierta

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)

Ejercicio Búsqueda en Profundidad

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



Al generar nodos seguimos el orden indicado

Expandido	Genera	Abierta
S	C, B	B, C
B	E, A	A, E , C
A	(C no se genera por repetido)	E, C
E	F, D	D, F, C
D	G2 (F no se genera por repetido)	G2, F, C
** G2	FIN	

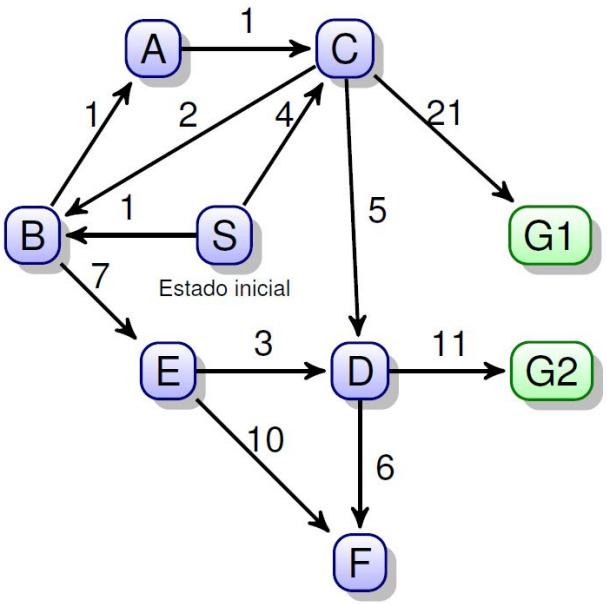
1 Iteraciones 6

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)

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)

Ejercicio Búsqueda en Profundidad

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos.
Al expandir, el orden es alfabético



Al generar nodos seguimos el orden alfabético indicado

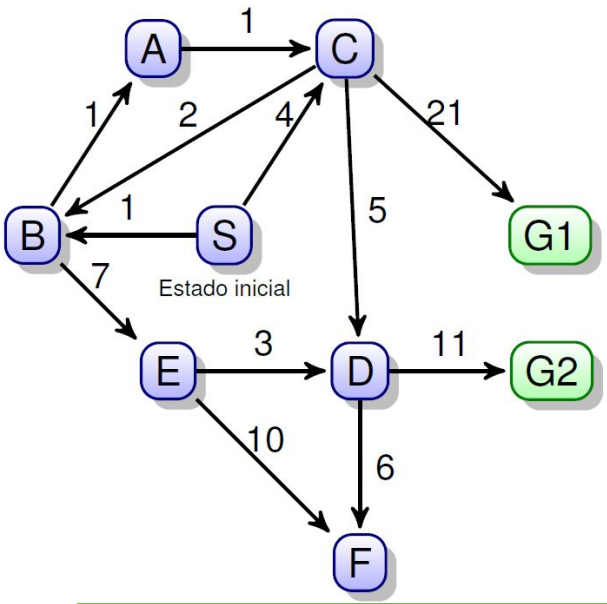
Expandido	Genera	Abierta

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)

Ejercicio Búsqueda en Profundidad

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos.
Al expandir, el orden es alfabético



Al generar nodos seguimos el orden alfabético indicado

Expandido	Genera	Abierta
S	B, C	B, C
C	B (repetido, lo omitimos), D, G1	A, E , C
A	(C no se genera por repetido)	E, C
E	D, F	D, F, C
D	G2 (F no se genera por repetido)	G2, F, C
** G2	FIN	

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)

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 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 g(N)
- Los nuevos sucesores se añaden ordenados en función del coste total hasta el nodo (g(N)).
- Se insertan en la lista Abierta[1] 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, 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

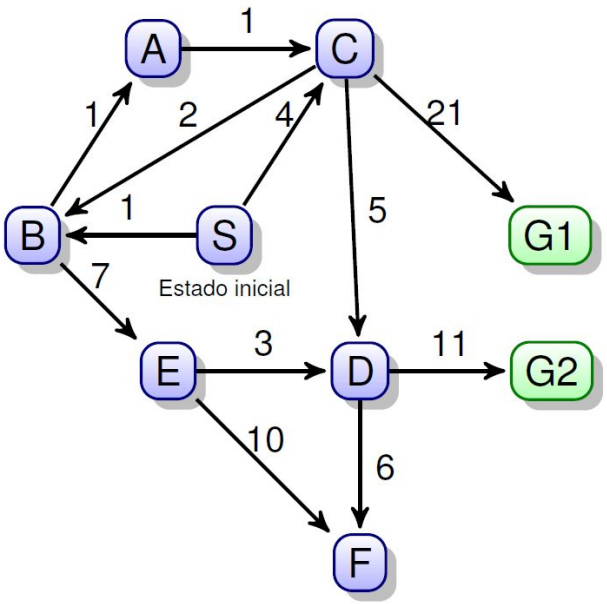
[1]Opcionalmente, usa lista cerrada para evitar repeticiones innecesarias: si un nodo estuvo en la lista abierta, y se expandió (desapareciendo), se guarda aquí. Se evita también introducir en Abierta nodos con el mismo estado, pero peor coste, que los que están en Cerrada.

Óptimo en coste

Complejidad exponencial en espacio y tiempo

Búsqueda con coste uniforme (UCS)

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



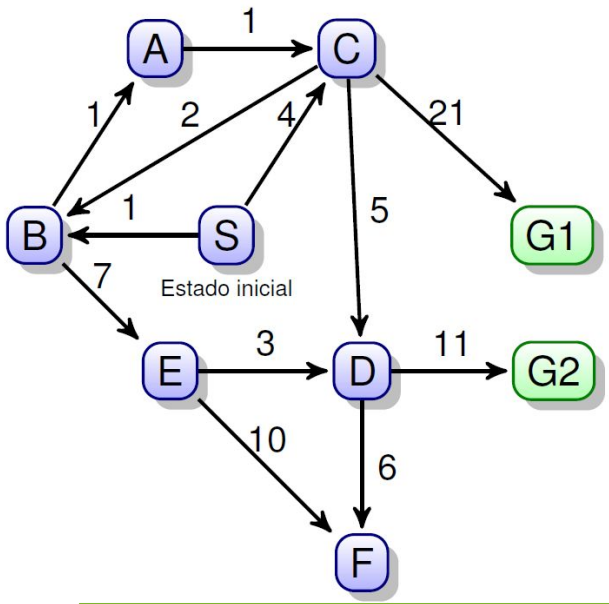
Expandido	Genera	Abierta	Cerrada

1

8

Búsqueda con coste uniforme (UCS)

S: estado inicial, G1 y G2 metas, Costes en los arcos. Al expandir, 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)

Autoevaluación

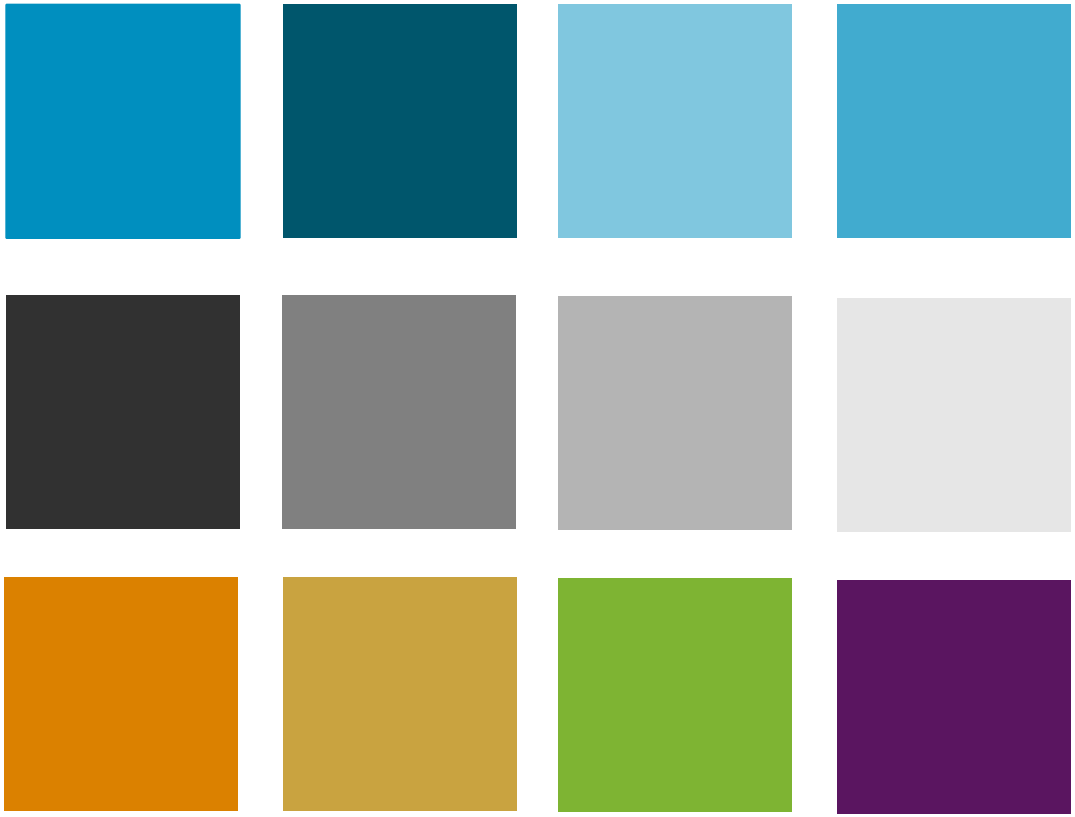
- ¿Sabemos representar estados y operadores para un problema de búsqueda?
- ¿Conocemos los conceptos de: estado, nodo, meta (nodo o función), factor de ramificación, profundidad?
- ¿Sabemos cómo funciona el algoritmo de búsqueda en amplitud y en profundidad?¿en qué coinciden y en qué se diferencian?
- ¿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

Reglas, hechos y motor de inferencia

- Los sistemas que utilizan inferencia lógica operan con una base de hechos (conocimiento del problema, afirmaciones que se consideran verdad) y una base de reglas (conocimiento del dominio, mecanismos para construir otras afirmaciones).
  - Los hechos representan el estado o situación del problema
  - Las reglas son implicaciones (A→B) y representan las formas por las que se puede resolver el problema
- Se opera entre hechos y reglas en ciclos, dirigidos por un motor de inferencia (que realiza el razonamiento según las reglas de la lógica)
- Hay dos formas de operar en estos sistemas:
  - encadenamiento hacia delante (se van generando conclusiones a partir de los hechos, hasta encontrar la buscada);
  - encadenamiento hacia atrás (se parte de la conclusión deseada y se ve si hay alguna regla que la pueda generar, y así hasta llegar a los hechos).

Procurar que los colores sean lo más similares a estos. Cuando se traten de gráficos, o imagenes sacadas de internet. Que predomine el azul, o gama de azules si es posible.



Formas de representación usadas en IA

PROLOG online: https://swish.swi-prolog.org/

Ejemplo PROLOG: https://www.cs.us.es/~fsancho/?e=73