

Algoritmo general de búsqueda

Input: Estado inicial S_0 , Estado final G

```

1: colaAbierta  $\leftarrow \{S_0\}$ 
2: mientras  $colaAbierta \neq \emptyset$ 
3:    $nodo \leftarrow$  extraer primero de  $colaAbierta$  Eleges una hoja
4:   Si  $G \subseteq nodo$  entonces
5:     return camino a  $nodo$  verifica si es un estado objetivo
6:   fin si
7:    $sucesores \leftarrow$  expandir ( $nodo$ )
8:   para cada  $sucesor \in sucesores$  hacer
9:      $sucesor.padre \leftarrow nodo$  si no lo es, expandes el nodo
10:     $colaAbierta \leftarrow colaAbierta \cup sucesor$ 
11:   fin para
12: fin mientras
13: return plan rta o problema sin solución
  
```

En la cola abierta se encuentran aquellos estados y caminos

El principal problema son los estados repetidos (los cuales pueden provocar errores como entrar en bucles infinitos). Para evitar esto:

- Ignorarlo
- Evitar ciclos simples: Evitando añadir el padre de un nodo al conjunto de sucesores
- Evitar ciclos generales: Evitando que ningún nodo anterior de un nodo se añada al conj. nro de sucesores
- Evitar todos los estados repetidos: No permitir añadir ningún nodo existente en el árbol al conjunto de sucesores

Búsqueda en amplitud o anchura (BFS) breadth first search

Input: Grafo, estado inicial S_0 , estado final G

```

1: definir colaAbierta como cola
2: marcar  $S_0$  como visitado
3: colaAbierta.add ( $S_0$ )
4: mientras  $colaAbierta \neq \emptyset$ 
5:    $nodo =$  colaAbierta.remove() Apesar de que el nodo que vamos a extraer en la iteración
6:   Si  $G \subseteq nodo$  entonces
7:     return camino
8:   fin si
9:    $sucesores \leftarrow$  Grafo.hijos ( $nodo$ ) (se expande)
10:  para cada  $sucesor \in sucesores$  hacer
11:    si  $sucesor$  no está marcado como visitado:
12:      marcar  $sucesor$  como visitado
13:       $sucesor.padre = nodo$ 
14:      colaAbierta.add ( $sucesor$ )
15:    fin si
16:  fin para
17: fin mientras
  
```

Características:

- El algoritmo es **completo** (encuentra solución si existe)
- El algoritmo es **óptimo** (encuentra la mejor solución) (menor costo)
- Deficiente en complejidad en tiempo y en complejidad en espacio (memoria ocupada)

- Es no recursivo
- Utiliza una cola (FIFO): "First in, first out", es decir, se van visitando nodos según el orden que entran en la lista
- Para marcar como visitado:
 - Guardar en otro conjunto (Ej: otra lista)
 - Utilizar un atributo del nodo que indique si se ha visitado ya
- Atributo **padre** es importante para recuperar el camino correcto.

Búsqueda en profundidad (DFS) Depth-first search

Input: Grafo, estado inicial S_0 , estado final G

```

1: definir pilaAbierta como pila
2: pilaAbierta.push ( $S_0$ )
3: mientras  $pilaAbierta \neq \emptyset$ 
4:    $nodo =$  pilaAbierta.pop()
5:   Si  $G \subseteq nodo$  entonces
6:     return nodo
7:   fin si
8:   si  $nodo$  no está marcado como visitado
9:     marcar  $nodo$  como visitado
10:     $sucesores \leftarrow$  Grafo.hijos ( $nodo$ )
11:    para cada  $sucesor \in sucesores$  hacer
12:       $sucesor.padre = nodo$ 
13:      pilaAbierta.push ( $sucesor$ )
14:    fin para
15:  fin si
16: fin mientras
  
```

Características:

- El algoritmo es **completo** (si no hay estados repetidos dentro de la misma rama)
- No es óptimo (no garantiza que se encuentre la solución que este a menor profundidad)

- Es no recursivo
- Utiliza una pila (LIFO). "Last in, first out" es decir, se van visitando los nodos últimos que entraron en la pila.
- Verifica que un nodo ha sido visitado hasta que lo saca de la pila

Búsqueda de coste uniforme

Input: Grafo, estado inicial S_0 , estado final G

```
1: definir colaAbierta como cola de prioridad
2: colaAbierta.add( $S_0$ )
3: mientras colaAbierta ≠ Ø
4:   nodo = colaAbierta.pop()
5:   si  $G \subseteq$  nodo entonces
6:     retornar camino a nodo
7:   fin si
8:   marcar nodo como visitado
9:   sucesores ← Grafo.hijos(nodo)
10:  para cada sucesor en sucesores hacer
11:    si sucesor no está marcado como visitado:
12:      si sucesor no está en colaAbierta:
13:        sucesor.padre = nodo
14:        colaAbierta.add(sucesor)
15:      sino:
16:        reemplazar sucesor si el costo es menor
17:        (habrá que cambiar el padre)
18:    fin si
19:  fin para
20: fin mientras
```

características

• El algoritmo es {completo} {óptimo}

• Utiliza una cola de prioridades
Se trata de una cola en la que el primer elemento en salir será el de menor costo asociado
• Por tanto la cola ordena los elementos de menor a mayor costo

• Se verifica que el nodo no haya sido ya procesado

• Para marcar como visitado un nodo:
- se añade en otro conjunto (ej. otra lista)
- se utilizar un atributo del nodo.

Búsqueda con algoritmo A*

// Inicialmente todos los valores de g y f en los nodos son ∞

Input: Grafo, Estado inicial S_0 , Estado final G

```
1: Definir colaAbierta como cola de prioridad
2: Definir listaCerrada como lista
3:  $S_0.g \leftarrow 0$ 
4:  $S_0.f \leftarrow h(\text{Grafo}, S_0, G)$ 
5: colaAbierta ← { $S_0$ }
6: listaCerrada ← { }
7: mientras colaAbierta ≠ Ø
8:   nodo ← extrae el de menor  $f$  de colaAbierta
9:   si  $G \subseteq$  nodo
10:    retornar camino a nodo
11:   fin si
12:   sucesores ← expandir(nodo)
13:   para cada sucesor en sucesores hacer
14:     tentativa.g ← nodo.g + c(nodo, sucesor)
15:     si tentativa.g < sucesor.g
16:       sucesor.padre ← nodo
17:       sucesor.g ← tentativa.g
18:       sucesor.f ← sucesor.g + h(Grafo, sucesor, G)
19:       si sucesor en listaCerrada con mejor valor de  $f$ 
20:         continuar
21:       fin si
22:       colaAbierta ← colaAbierta ∪ sucesor
23:     fin para
24:   listaCerrada ← listaCerrada ∪ nodo
25: fin mientras
26: retorna plan vacío o problema sin solución
```

• El algoritmo es {completo} {óptimo} (si h es óptima)

Ejercicios de Examen Razonamiento y planificación automática

Tipo (Tabla de verdad)

Problemas

Formalizar expresiones

- ① Un día (mientras) cuatro amigos (jugaban) al póker, intentaban formar una excusa real de rocamboles y cada uno de ellos ofirió lo siguiente:

Carlos: "Si no tengo una carta de los jefes, enton~~ce~~ Moreí!"
Daniel: "Si tenemos una carta!"

Daniel: "Sí tengo una carta de corazones, entonces no lo veas".
Sanhego: "Sí, no lo veas".

Santiago: "Si no tengo una carta de corazon, entonces no llovere".

Todos tienen la misma posibilidad de tener la carta de corazones para formar la escalera real y
terminado el juego comienza a llorar

El último amigo que encuchó, David dijo: "Uno de ustedes ha dicho una afirmación falsa". ¿Quién de los tres amigos mintió?

P = "Tengo una carta de corazon",
q = "Illo verá"

Si --- entonces

P	q	$P \rightarrow q$
0	0	1
0	1	1
1	0	1
1	1	1

(Se debe desarrollar tabla de verdad completa)

				Carlos		Daniel		Santiago	
P	q	$\neg P$	$\neg q$	$\neg P \rightarrow q$	$P \rightarrow \neg q$	$\neg P \rightarrow \neg q$	$\neg P \rightarrow \neg \neg q$		
0	0	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1	0	0
1	0	0	1	1	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1

Asumiendo que el enunciado dice que todos tienen carta de corazones y que comienza a llorar, entonces tenemos $P = 1$ y $q = 1$. Esto en la última fila de la tabla de verdad, donde podemos ver que Daniel es el que ha mentido ya que tiene un 0 en la casilla.

- ② Formalizar expresiones y resolverla con tablas de verdad. Indicar si son tautología, contradicción o inconsistencia

1) Vi la sere aungue [no] leí la novela 2) Vi la sene y [no] leí la novela

2) No me gusta madrugar ni traenocher ~ No me gusta madrugar no me gusta traenocher

1) $p = \text{"ver la serie"}$; $q = \text{"leer la novela"}$

2) P = "me gusta madrugar"; q = "me gusta trasnochar"

P	q	$\neg q$	$P \wedge (\neg q)$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Es inconsistente

P	q	ㄱp	ㄱq	ㄱp ∧ ㄱq
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Es inconsistencia

Ejercicios de formalización

- En los polos el frío es intenso únicamente $\boxed{\text{si los planetas giran en torno al sol}}$

$P = \text{"en los polos el frío es intenso"}$ y $\boxed{P \rightarrow q}$ ($\neg q \rightarrow P$)
 $q = \text{"los planetas giran en torno al sol"}$ y $\boxed{\neg q \rightarrow P}$ ($P \rightarrow q$)

- Siempre que los herbívoros corren o el frío en los polos es intenso, los planetas giran en torno al sol

$P = \text{"los herbívoros corren"}$

$q = \text{"el frío en los polos es intenso"}$

$r = \text{"los planetas giran en torno al sol"}$

$$\boxed{(P \vee q) \rightarrow r}$$

- Juan es francés $\boxed{\text{si nació el 23 de febrero}}$. Si es bretón, entonces es más bien bajo. Ahora bien, nació el 23 de febrero $\neg q$ es bretón. Por consiguiente, es francés o es más bien bajo.

$P = \text{"Juan es francés"}$

$q = \text{"Juan nació el 23 de febrero"}$

$r = \text{"Juan es bretón"}$

$s = \text{"Juan es más bien bajo"}$

(el por consiguiente se representa \vdash)

$$\boxed{q \rightarrow P, r \rightarrow s, q \vee r \vdash P \vee s}$$

- Si es cierto que Aristóteles nació en Estagira $\neg q$ que fue tutor de Alejandro Magno y, ademá, que $\boxed{\text{si nació en Estagira era macedonio por su nacimiento, entonces era efectivamente macedonio}}$.

$P = \text{"Aristóteles nació en Estagira"}$

$q = \text{"Aristóteles fue tutor de Alejandro Magno"}$

$r = \text{"Aristóteles era macedonio por su nacimiento"}$

$$\boxed{(P \wedge q) \wedge (P \rightarrow r) \rightarrow r}$$

- Un solo proveedor \neg puede afectar los precios $\neg P$ el mercado es libre. Si un solo proveedor no puede afectar los precios, es que hay un gran número de proveedores. Es así que no hay un gran número de proveedores, luego, no es libre el mercado

$P = \text{"un solo proveedor puede afectar los precios"}$

$q = \text{"El mercado es libre"}$

$r = \text{"Hay gran número de proveedores"}$

$$\boxed{q \rightarrow \neg P, \neg P \rightarrow r, \neg r \vdash \neg q}$$

- Si a es un número par $\neg b$ es un número impar, entonces c es igual a a . Ahora bien c no es igual a a (menos que sea mayor que b). Pero c no es mayor que b . Ademá, a es un número par. Luego, b no es un número impar \vdash Una unor

$P = \text{"a es un número par"}$

$q = \text{"b es un número impar"}$

$r = \text{"c es igual a a"}$

$s = \text{"c es mayor que b"}$

$$\neg P \vee s$$

$$\boxed{(P \wedge q) \rightarrow r, \neg s \rightarrow \neg r, \neg s, P \vdash \neg q}$$

Tema 3

Lógica matemática

Proposiciones:

	Conectores lógicos	Símbolos
y		\wedge
o		\vee
Si... entonces		\rightarrow
Si y solo si		\leftrightarrow
no		\neg

Tablas de verdad

Negación

P	$\neg P$
0	1
1	0

Conjunción (AND)

P	q	$P \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Disyunción no (OR)

P	q	$P \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Disyunción exclusiva (XOR)

P	q	$P \Delta q$
0	0	0
0	1	1
1	0	1
1	1	0

Condicional

P	q	$P \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Solo da verdadero cuando la primera proposición verdadera y la segunda falsa

Bicondicional

P	q	$P \leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Es Tautología

Es Tautología
(Da siempre 1 o
verdad en la expresión)

NOTA: Que sea Tautología no depende de los valores de verdad de los proposiciones que lo forman, sino de la manera en la que se establecen relaciones sintácticas de una proposición con otras. Ej: aquí al combinar una expresión verdadera con \neg una falsa, debido a la relación sintáctica, siempre se obtiene una verdadera

Ejemplo 1

$P = \text{"El auto es rojo"}$, $\neg P = \text{"el auto no es rojo"}$

Expresión: "el auto es rojo \neg el auto no es rojo"

P	$\neg P$	$P \vee \neg P$
0	1	1
1	0	1

Ejemplo 2

$P = \text{"voy al cine"}$, $q = \text{"voy a cenar"}$, $r = \text{"me quedo en casa"}$

Expresión: "Si voy al cine \wedge voy a cenar, entonces voy al cine \neg no me quedo en casa"

P	q	r	$\neg r$	$\neg r \wedge q$	$P \vee \neg r \wedge q$	$(P \wedge q) \rightarrow (P \vee \neg r \wedge q)$
0	0	0	1	0	1	1
0	0	1	0	0	0	1
0	1	0	1	0	1	1
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1

Es Tautología

Hay 3 proposiciones por lo que partimos de $2^3 = 8$ en la tabla de verdad

Ejemplo 3

$P = \text{"Estudio en casa"}$, $q = \text{"Estudio en la biblioteca"}$

Expresión: "estudio en casa \neg estudio en la biblioteca \wedge \neg estudio en casa \wedge \neg estudio en la biblioteca"

P	q	$\neg P$	$\neg q$	$P \vee q$	$\neg P \wedge \neg q$	$(P \vee q) \wedge (\neg P \wedge \neg q)$
0	0	1	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Es contradicción (Da siempre 0 o falso en la expresión). Al igual que en la tautología, no depende de los valores de las proposiciones (veras que las hay verdaderas y falsas), sino de las relaciones sintácticas que se establecen entre ellas.

Ejemplo 4

$P =$ "Estudio en casa", $q =$ "Estudio en la biblioteca"

Expresión: "Estudio en casa \wedge \neg estudio en la biblioteca" \vee \neg "estudio en casa \wedge \neg estudio en la biblioteca"

P	q	$\neg P$	$\neg q$	or		Expresión
				$P \vee q$	$\neg P \vee \neg q$	
0	0	1	1	0	1	0
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	1	0	0	1	0	0

Es inconsistencia (Tiene ilógico contingente o falacioso)

Aquella ~~expresión~~ que puede ser verdadera o falsa, combinando tanto la contradicción, según los valores de las proposiciones que la integran.

Ejemplo de examen: Un día mientras 4 amigos jugaban al poker, intentaban formar una escalera real de corazones y cada uno de ellos afirmó lo siguiente:

- Carlos afirmó: "Si \neg tengo una carta de corazones, entonces neveré"
- Daniel afirmó: "Si tengo una carta de corazones, entonces \neg neveré"
- Santiago afirmó: "Si \neg tengo una carta de corazones, entonces \neg neveré"

Todos tienen la misma posibilidad de tener la carta de corazones para formar la escalera real y terminar el juego ~~común~~ coincide a never

El último amigo que se encargaba, David, dijo: "Uno de ustedes ha dicho una afirmación falsa"

¿Quién de los 3 amigos mintió?

$P =$ "Tengo una carta de corazones"

$q =$ "Neveré"

condicional

P	q	$P \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Analizaremos las expresiones

Carlos Daniel Santiago

P	q	$\neg P$	$\neg q$	$\neg P \rightarrow q$	$P \rightarrow \neg q$	$\neg P \rightarrow \neg q$
0	0	1	1	0	1	1
0	1	1	0	1	1	0
1	0	0	1	1	1	1
1	1	0	0	1	0	1

Ahora con los datos del enunciado evaluemos la pila producida y veremos quién ha dicho la verdad.

Del enunciado ~~no~~ se da muy claro, pero ~~supongamos~~ que dice que ~~que~~ empieza a never

Entonces (nos fijamos en filas 2 y 4 de la tabla)

Del enunciado no se da muy claro, pero ~~supongamos~~ que dice que todos consiguen la carta de corazones y que comienza a never (ya que no dan info individual de si se ha pasado)

Entonces $P=1$ (verdadero) y $q=1$

Fila 4 de la tabla \rightarrow [El que miente es Daniel]

(Este hecho con la suposición que se ha puesto ~~que~~ no se dice claramente que ocurre por saber quién ha mentido)

Si se elevan los precios o los salarios habrá inflación. Si hay inflación, el gobierno ha de regularla o el pueblo sufrirá. Si el pueblo sufre, los gobernantes se hacen más impopulares. Pero es así que el gobierno no regulará la inflación y que, sin embargo, los gobernantes no se harán más impopulares. Entonces es que no subirán los salarios.

P = "se elevan los precios"

q = "se elevan los salarios"

r = "hay inflación"

s = "el gobierno regula la inflación"

t = "el pueblo sufre"

u = "los gobernantes se hacen impopulares"

$$(P \vee q) \rightarrow r, r \rightarrow (s \wedge t), t \rightarrow u, \neg s \wedge \neg u \vdash \neg q$$

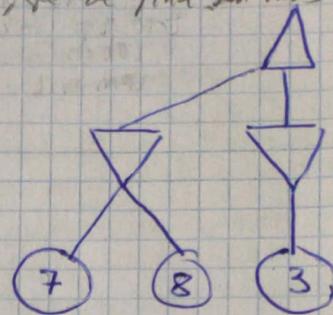
Ejercicio tipo - Árbol de búsqueda

OJO: Dice que no nos piden en la geometría que el final son nodos

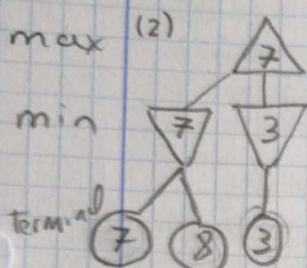
Dado el siguiente árbol:

Aquí pongo los iconos para ayudar y que se vea que es en minimax

- (1) Identifica el tipo de algoritmo con el que se ha generado
- (2) Completar el árbol asignando valores a los nodos que no tienen
- (3) Explicar paso a paso como funciona el algoritmo



(1) Dicho árbol se ha generado con el algoritmo de búsqueda minimax. En él, la figura Δ representa a max, ∇ representa a min y \circ son los nodos terminales



En los nodos terminales se encuentra la utilidad. Teniendo en cuenta que min busca minimizar la utilidad, en cada uno de los nodos se incluye el valor mínimo de las ramas que conducen a ese nodo (de abajo a arriba).

Por tanto \rightarrow

En el caso de max es al revés, como trata de maximizar, se toma el valor del nodo de mayor valor, en este caso el de 7

- (3) El funcionamiento se resume en elegir el mejor movimiento para uno mismo partiendo de la suposición de que el contrincante elegirá el peor para el adversario

Se considera que se está utilizando la versión general del algoritmo y la que aparece representando es el árbol completo (ya que otra interpretación es que se está jugando minimax con suspensión, en el cual la profundidad del árbol se limita a un valor determinado).

El algoritmo utiliza dos funciones mutuamente recursivas. Ejemplo paso a paso:

1) Se genera el árbol completo (o prof determinada si se trata de minimax con suspensión)

2) Comienza max:

- Evalúa si es nodo terminal \rightarrow si lo es devuelve la utilidad (En este caso no lo es)
- Expande sucesores
- Se inicializa el valor de $\alpha \rightarrow -\infty$
- Para cada sucesor:
 - Se asigna a α el valor max entre α y MinValor (sucesor)

Inicializar $\alpha \rightarrow -\infty$

\hookrightarrow Esto es la llamada a la función de min

3) Al haber llamado a min (se realizan estos pasos a cada sucesor generado en 2)

- Evalúa si es nodo terminal \rightarrow si lo es devuelve la utilidad (En este caso no lo es)
- Expande sucesores
- Se inicializa $\beta \rightarrow +\infty$
- Para cada sucesor

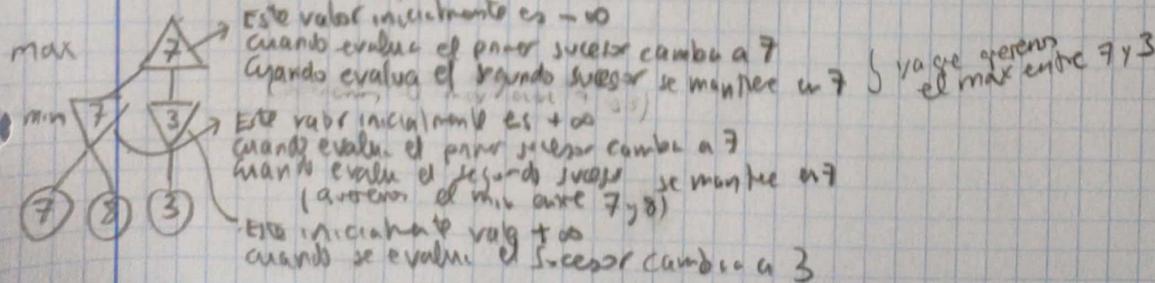
Iniciarizar $\beta \rightarrow +\infty$

- Se asigna a β el valor min entre β y MaxValor (sucesor)

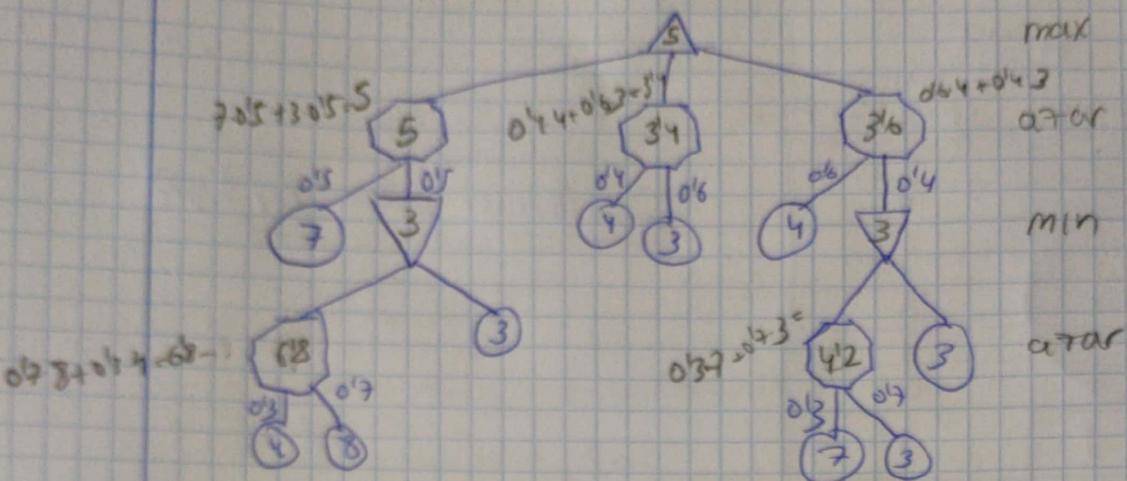
\hookrightarrow Esto es llamada a función de max

4) Al haber llamado a max (se realizan estos pasos a cada sucesor generado en 3)

- Evalúa si es nodo terminal \rightarrow Lo es, a sigue se devuelve utilidad



- ① a) El árbol está generado con el algoritmo expectimax ya que se pueden identificar los turnos de min, max y azar
- b) Completar el árbol asignando los valores en los nodos no terminales $\circ \rightarrow$ Nodos terminales



- d) Mencionar y explicar la principal desventaja del algoritmo

La principal desventaja de este método es la complejidad añadida al introducir el azar ya que el árbol crece y se ramifica mucho

- c) Explicar paso a paso y con sus palabras el funcionamiento del árbol

1) Se genera el árbol completo con la profundidad especificada

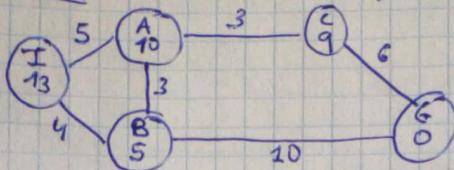
2) Empieza max

- al inicialmente vale $\rightarrow -\infty$
para cada hijo de max se tome

$$x = \max (\alpha, \text{expectimax} (\text{hijo}, \text{profundidad} - 1, \text{turno siguiente}, \text{azar}))$$

3) Se toca a ~~min~~ azar

Ejercicio Grafo | Resolución con algoritmo A*



Iteraciones:

- Definimos colaAbierta como colaPrioridad
- Definimos listaCerrada como lista
- $I.g = 0$, $I.h = 13$, $I.f = I.g + I.h$
- Introducimos I en colaAbierta
- Como la colaAbierta no está vacía, sacamos su primer elemento (que al ser de prioridad será el de menor coste). En este caso I
- I no es nodo objetivo, asique no se retorna nada
- Se expande I
- Para los sucesores de I (A, B):
 - Para A : $\stackrel{0}{\text{I}}$ $\stackrel{5}{\text{A}}$ \rightarrow Unir de I, A
 - $g_{\text{tentativo}} = I.g + C(I, A) = 5$
 - Como $g_{\text{tentativo}} < A.g$ (vale inicialmente ∞)
 - I padre (A)
 - $A.g = g_{\text{tentativo}} = 5$
 - $A.f = A.g + h'(A) = 5 + 10 = 15$
 - A no está en listaCerrada por lo que se mete en colaAbierta A con su mejor valor de f , que es 15
- Para B
 - $g_{\text{tentativo}} = I.g + C(I, B) = 4$
 - Como $g_{\text{tentativo}} < B.g$ (vale inicialmente ∞)
 - I padre (B)
 - $B.g = g_{\text{tentativo}} = 4$
 - $B.f = B.g + h'(B) = 4 + 5 = 9$
 - B no está en listaCerrada por lo que se mete B en colaAbierta con su mejor valor de f , que es 9

- Introducimos I en listaCerrada

- Estado
- | |
|--------------------------------------|
| colaAbierta: $(B, 9), (A, 15)$ |
| listaCerrada: $(I, 13)$ |
| I padre (B), I padre (A) |

- Como la colaAbierta no está vacía, sacaremos su primer elemento (B)
 - B no es objetivo asique no se retorna el camino
 - Se expande B
 - Para los sucesores de B (I, A, G)
 - Para I :
 - $\text{tentativo_g} = B.g + C(B, I) = 4$
 - $\text{tentativo_g} = I.g \rightarrow$ no más acciones
 - Para A :
 - $\text{tentativo_g} = B.g + C(B, A) = 7$
 - $\text{tentativo_g} \geq A.g \rightarrow$ no más acciones
 - Para G :
 - $\text{tentativo_g} = B.g + C(B, G) = 14$
 - $\text{tentativo_g} \leq G.g \rightarrow$ vale inicialmente ∞
 - B padre (G)
 - $G.g = g_{\text{tentativo}} = 14$
 - $B.f = B.g + h'(G) = 14$
 - G no está en listaCerrada por lo que se mete G en colaAbierta con su mejor valor de f , que es 14
 - Introducimos B en listaCerrada

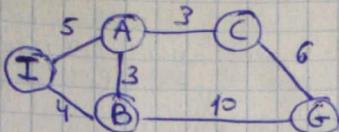
Consideraciones

Estado inicial: I
Estado final: G
coste $\rightarrow f = g + h$

Inicialmente g y f de todos los nodos son ∞
 Los h no son incluidos en el coste

- Estado
- | |
|---|
| colaAbierta: $(G, 14), (A, 15)$ |
| listaCerrada: $(I, 13), (B, 9)$ |
| I padre (B), I padre (A), B padre (G) |
- como colaAbierta no está vacía, sacamos su primer elemento (G)
 - G es nodo objetivo \rightarrow retornamos el camino
 - $I \rightarrow B \rightarrow G \quad f = 14$ (coste)
 - El camino se saca como G .padre = B , B .padre = I

Ejercicio gráfico - Resolución con búsqueda de coste uniforme



Consideraciones

nodo inicial: I
nodo final: G

Como costes se toman solo los costes de las transiciones (los costes internos de los nodos no se toman en cuenta por el A*)

Iteraciones:

- 1) Creamos cola y la definimos como cola de prioridad
- creamos lista para meter los nodos visitados
 - Anadimos I a la cola
 - Como la cola no está vacía, sacamos su primer elemento (I)
 - Como I no es nodo objetivo no retornamos el camino
 - Anadimos I a la lista de nodos visitados
 - Expandimos I
 - Para los sucesores de I (A, B):
 - A → No está en lista de visitados y no está en cola abierta por lo que se pega I como padre de A y se añade A a la cola
 - B → No está en lista de visitados y no está en cola abierta por lo que se pega I como padre de B y se añade B a la cola

Estado

cola: (B, 4), (A, 5)	(cola de prioridad, se asigna el costo)
lista de visitados: (I, 0), (B, 4)	(ordenados por costo)
I padre (B), I padre (A)	

- 2) Como la cola no está vacía sacamos el primer elemento (B), que es el de menor costo
- Como B no es objetivo no retornamos el camino
 - Anadimos B a la lista de nodos visitados
 - Expandimos B
 - Para los sucesores de B (I, A, G):
 - I → está visitado → se ignora
 - A → no está visitado pero está en cola. Se comprueba el costo: $I \rightarrow A \quad \text{costo 5}$
 $I \rightarrow B \rightarrow A \quad \text{costo 9}$
No se modifica
 - G → no está visitado ni en la cola, por lo que se pega B como padre de G y se añade G a la cola → 4+10 = 14 **OJO**

Estado

cola: (A, 5), (G, 14)
lista de visitados: (I, 0), (B, 4)
I padre (B), I padre (A), B padre (G)

- 3) Como la cola no está vacía sacamos el primer elemento (A), que es el de menor costo
- Como A no es el objetivo no retornamos camino
 - Anadimos A a la lista de nodos visitados
 - Expandimos A
 - Para los sucesores de A (I, B, C):
 - I, B → ya visitados → se ignoran
 - C → no visitado y no en cola. Se pega A como padre de C y se mete a C en la cola

Estado

cola: (C, 8), (G, 14)
lista de visitados: (I, 0), (B, 4), (A, 5)
I padre (B), I padre (A), B padre (G)
A padre (C)

- 4) - Como cola no está vacía, sacamos el primer elemento (C)

- Como C no es objetivo no retornamos el camino

- Anadimos C a la lista de nodos visitados

- Expandimos C

- Para los sucesores de C (A, G):

A → ya visitado → se ignora
G → no visitado, por lo que se pega en la cola. Se comprueba el costo: $I \rightarrow B \rightarrow G \quad \text{costo 14}$

$I \rightarrow A \rightarrow C \rightarrow G \quad \text{costo 14}$
Como no mejora respecto al que había no se modifica

Estado

cola: (G, 14)

lista de visitados: (I, 0), (B, 4), (A, 5), (C, 8)

I padre (B), I padre (A), B padre (G), A padre (C)

- 5) - Como cola no está vacía sacamos el primer elemento (G)

- Como G es objetivo, retornamos camino: $I \rightarrow B \rightarrow G \quad (\text{costo 14})$

(N.B. deberíamos retornar el otro camino porque B está igual → primero como padre de G, y por costo al revés la ruta C-G no desciende y no se modifica el padre)

Ejercicios Típicos - Problema

Dada la matriz de caracteres de 2D que representa el tablero del juego buscaminas, donde \boxed{H} → mina no revelada, \boxed{V} → celda vacía no revelada, \boxed{B} → celda en blanco ~~revelada~~ que no tiene minas adyacentes (arriba, abajo, izq, der y 4 diagonales), Digito del 1 al 8 indica el n° de minas adyacentes a la celda revelada, \boxed{X} → mina revelada.

Se debe partir de una posición dada (índice de fila y columna) y entre todos los cuadros ~~revelados~~ (M o V), retornar el tablero resuelto después de revelar esta posición de acuerdo a las siguientes reglas:

- (1) Si la posición dada revela una mina (H), entonces el juego ha terminado. Cambiar a X
- (2) Si la posición revela un cuadro vacío (V) sin minas adyacentes, cambiar \boxed{V} a cuadro blanco revelado (B) y todos sus cuadros adyacentes no revelados deben revelarse de forma recursiva
- (3) Si la posición revela un cuadro vacío (V) con al menos una mina adyacente, se debe cambiar a un digito (1 a 8) que represente el n° de minas adyacentes
- (4) Cuando no se revelan más casillas se retorna el tablero.

Para los siguientes ejemplos se hace click en la casilla $(0, 2)$

Ej 1:

	0	1	2
1	\boxed{V}	\boxed{V}	\boxed{V}
2	\boxed{M}	\boxed{V}	\boxed{V}
3	\boxed{M}	\boxed{M}	\boxed{V}

click en $(0, 2)$
Retorna

\boxed{V}	1	\boxed{B}
\boxed{M}	3	1
\boxed{M}	\boxed{M}	\boxed{V}

Se aplica (2) y como hay vacío hay que revelar de forma recursiva los vacíos

Ej 2:

\boxed{V}	\boxed{V}	\boxed{V}
\boxed{V}	\boxed{V}	\boxed{V}
\boxed{M}	\boxed{V}	\boxed{V}

click en $(0, 2)$
Retorna

\boxed{B}	\boxed{B}	\boxed{B}
1	1	\boxed{B}
\boxed{M}	1	\boxed{B}

Al aplicar 2 revelando vacío B para 2 de los adyacentes, los que no se han aplicado se aplican de forma recursiva

Ej 3:

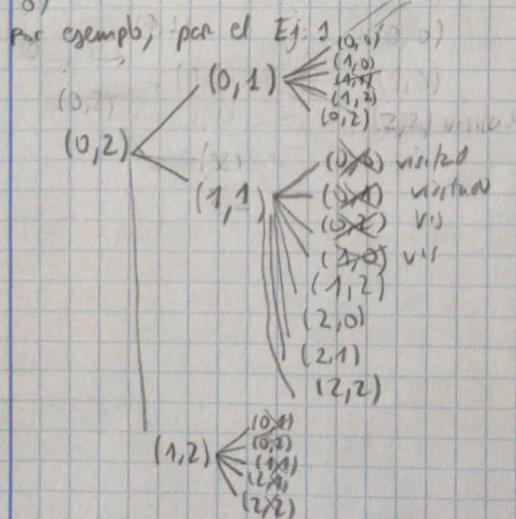
\boxed{V}	\boxed{V}	\boxed{V}
\boxed{V}	\boxed{V}	\boxed{V}
\boxed{V}	\boxed{V}	\boxed{V}

click en $(0, 2)$
Retorna

\boxed{B}	\boxed{B}	\boxed{B}
\boxed{B}	\boxed{B}	\boxed{B}
\boxed{B}	\boxed{B}	\boxed{B}

- Completar los retornos (el no lo pide en enunciado)
- Identificar el tipo de algoritmo de búsqueda con el que se pide resolver el problema.
- Explicar paso a paso y con sus palabras el funcionamiento del algoritmo

Necesitamos generar un arbol que por cada nodo descubierto, compruebe todos sus adyacentes (puede tener hasta 8)



Para solucionarlo sería necesario aplicar algoritmo de búsqueda en profundidad

Utilización del algoritmo BFS (Te iré indicando las posiciones a visitar)

$$\begin{pmatrix} V & V & V \\ M & V & V \\ M & M & V \end{pmatrix} \rightarrow \begin{pmatrix} V & 1 & 0 \\ M & 3 & 1 \\ M & M & V \end{pmatrix}$$

Estado inicial

0) cola = [(0,2)]

Estado final

1) cola = [(0,1), (1,1), (1,2)]

visitado = [(0,2)]

(0,2) padre [(0,1), (1,1), (1,2)]

2) cola = [(1,1), (1,2), (0,0), (1,0)]

visitado = [(0,2), (0,1)]

(0,2) padre [(0,1), (1,1), (1,2)]

(0,1) padre [(0,0), (1,0)]

3) cola = [(1,2), (0,0), (1,0), (2,0), (1,1), (2,2)]

visitado = [(0,2), (0,1), (1,1)]

(0,2) padre [(0,1), (1,1), (1,2)]

(0,1) padre [(0,0), (1,0)]

(1,1) padre [(2,0), (2,1), (2,2)]

4) cola = [(0,0), (1,0), (2,0), (2,1), (2,2)]

visitado = [(0,2), (0,1), (1,1), (1,2)]

⊗

5) cola = [(1,0), (2,0), (2,1), (2,2)]

visitado = [(0,2), (0,1), (1,1), (1,2), (0,0)]

⊗

6) cola = [(2,0), (2,1), (2,2)]

visitado = [(0,2), (0,1), (1,1), (1,2), (0,0), (1,0)]

⊗

7) cola = [(2,1), (2,2)]

visitado = [(0,2), (0,1), (1,1), (1,2), (0,0), (1,0), (2,0)]

⊗

8) cola = [(2,2)]

visitado = [(0,2), (0,1), (1,1), (1,2), (0,0), (1,0), (2,0), (2,1)]

⊗

9) cola = \emptyset

visitado = [(0,2), (0,1), (1,1), (1,2), (0,0), (1,0), (2,0), (2,1), (2,2)]

⊗

Luego creo que lo mejor haría por búsqueda en profundidad más que en anchura

Algoritmo utilizado

Input: Casilla en la que hacen click (I0)

1: definir cola como cola

2: definir visitado como lista

3: cola.add (I0)

4: mientras cola $\neq \emptyset$

5:

Exercicio tipo - Problema en STRIPS

Dados dos bloques A y B situados sobre una mesa M. El bloque B está sobre A y el objetivo es colocar el bloque A sobre B.

Se define el problema formalmente en STRIPS de la siguiente manera:

Estado inicial (I): $\text{sobre}(B, A) \wedge \text{libre}(B) \wedge \text{sobre}(A, M)$

Objetivo (G): $\text{sobre}(B, M)$

Operadores (O):

- $\text{MoverSobre}(\text{?bloque1}, \text{?bloque2})$

Precondiciones: $\text{libre}(\text{?bloque1}) \wedge \text{libre}(\text{?bloque2})$

Efectos: $\text{sobre}(\text{?bloque1}, \text{?bloque2}) \wedge \neg \text{libre}(\text{?bloque2})$

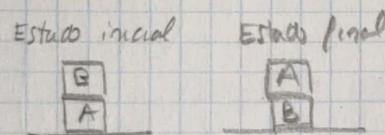
- $\text{MoverSobreLaMesa}(\text{?bloque1}, \text{?bloque2}, \text{?Mesa})$

Precondiciones: $\text{libre}(\text{?bloque1}) \wedge \text{sobre}(\text{?bloque1}, \text{?bloque2})$

Efectos: $\neg \text{sobre}(\text{?bloque1}, \text{?bloque2}) \wedge \text{sobre}(\text{?bloque1}, \text{?mesa})$

Se debe responder

- (1) • ¿El objetivo está correctamente especificado teniendo en cuenta el enunciado?
- (2) • ¿Consideras que los operadores están bien definidos?
- (3) • Asumiendo que las acciones están bien definidas. ¿De acciones son aplicables en el estado inicial? Para cada acción aplicable genera el estado resultante
- (4) • Teniendo como nodo raíz el estado inicial, construye el árbol de búsqueda aplicando los algoritmos:
 - Ancho
 - Profundidad
 (en un examen debes solo un algoritmo el que haya que aplicar)



- (1) No, habría que indicar también que A está sobre B y que A queda libre por encima. Sería de la siguiente manera:

$\text{sobre}(B, M) \wedge \text{sobre}(A, B) \wedge \text{libre}(A)$

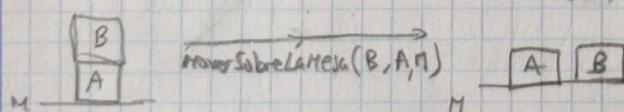
- (2) Mover Sobre si esto bien

MoverSobreMesa no está bien, faltaria liberar el bloque B. El efecto sería:

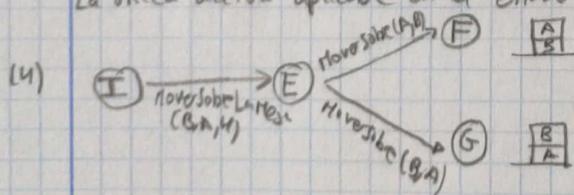
$\neg \text{sobre}(\text{?bloque1}, \text{?bloque2}) \wedge \text{sobre}(\text{?bloque1}, \text{?mesa}) \wedge \text{libre}(\text{?bloque2})$

- (3) Estado inicial

Estado generado: $\text{sobre}(B, M) \wedge \text{sobre}(A, M) \wedge \text{libre}(A) \wedge \text{libre}(B)$
(E)



La única acción aplicable en el estado inicial es $\text{MoverSobreMesa}(B, A, M)$



El BFS comienza en I y desarrolla todas las opciones posibles, ya que solo hay 1 iría a E. Despues de expandir E tendré que evaluar F y G, pero al llegar a F veré que es la meta y retornaré el camino.

El DFS comienza en I y desarrolla la primera de las expansiones posibles (en este caso solo hay 1 (E)). Despues en E seguirá por una de las ramas, si va por F entonces llega a la estado meta y retorna el camino. En caso de no ir por E irá por G que no lleva a la meta y que no hay más caminos así que volverá atrás a evaluar F.

Problema Strips

Paquete P y camión C en ciudad A

Objetivo en que el paquete se localice en ciudad B y el camión C termine en ciudad A

Se define:

Estado Inicial (I): posición (P, A) \wedge posición (C, A)

Objetivo (G): posición (P, B)

Operadores:

- mover (?camión, ?origen, ?destino)

precond: posición (?camión, ?origen)

efectos: posición (?camión, ?destino) \wedge \neg posición (?camión, ?origen)

- cargar (?paquete, ?camión, ?ciudad)

precond: posición (?paquete, ?ciudad) \wedge posición (?camión, ?ciudad)

efectos: en (?paquete, ?camión) \wedge \neg posición (?paquete, ?ciudad)

- descargar (?paquete, ?camión, ?ciudad)

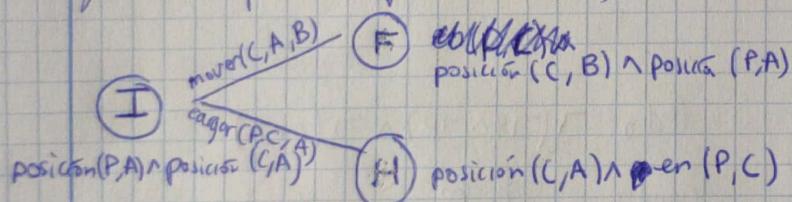
precond: posición (?camión, ?ciudad) \wedge en (?paquete, ?camión)

efectos: posición (?paquete, ?ciudad) \wedge \neg en (?paquete, ?camión)

(1) El objetivo no está correctamente especificado ya que en el anuncio se indica que el camión debe ~~se~~ terminar en la ciudad A y esto no aparece en G. Debería ser:
Objetivo (G): posición (P, B) \wedge posición (C, A)

(2) Acciones aplicables desde I, ¿por qué son aplicables? Indicar estado resultante por cada acción aplicable

Partiendo de I las acciones posibles son mover y cargar ya que son las únicas en las que se cumplen las precondiciones



(3) Aplicar algoritmo en anchura controlando estados repetidos hasta llegar al nodo G

Estados posibles

I: pos(P, A) \wedge pos(C, A)

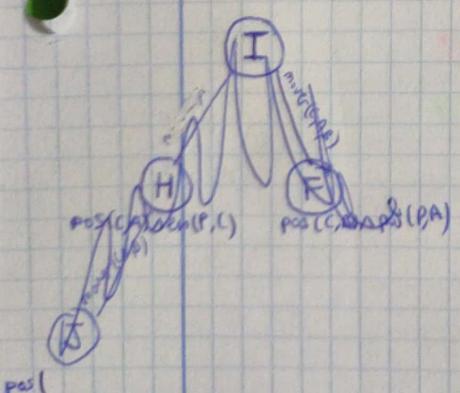
H: pos(C, A) \wedge en(P, C)

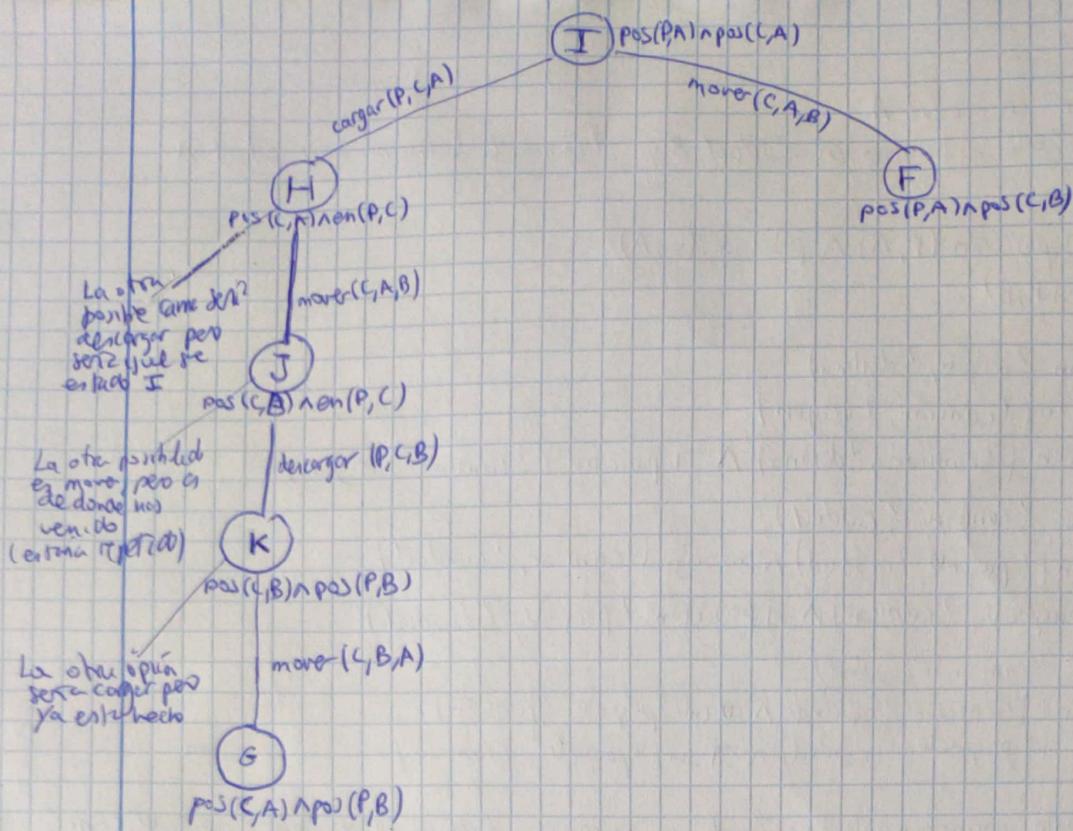
F: pos(C, B) \wedge pos(P, A)

J: pos(C, B) \wedge en(P, C)

K: pos(C, B) \wedge pos(P, B)

G: pos(C, B) \wedge pos(P, B)





Roberto: "Ni camila ni yo lo hicimos"

Camila: "Roberto o Pedro lo hicieron"

Pedro: "Ustedes dos están mintiendo"

2 dicen la verdad y 1 miente

P : Roberto robó

q : Camila robó

r : Pedro robó

(1) Expresiones en forma simbólica

Roberto: $\neg q \wedge \neg P$ Camila: $P \vee r$

Pedro: $\neg(\neg q \wedge \neg P) \wedge \neg(P \vee r)$

(2) Tablas de verdad

P	q	r	$\neg q$	$\neg P$	$\neg q \wedge \neg P$	$P \vee r$	$\neg(\neg q \wedge \neg P) \wedge \neg(P \vee r)$
0	0	0	1	1	1	0	0
0	0	1	1	1	1	1	0
0	1	0	0	1	0	0	1
0	1	1	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	1	1	0	0	1	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	1	0
							0

olvidarse de r

adicionalmente y 1 miente

Para que Pedro diga la verdad (1 en la tabla) implica que es Camila la que robó ($q=1$ en la fila 10)

Si dos dicen la verdad y una miente considerando el enunciado habrá que coger la pila en la que los (columnas) Roberto, Camila y Pedro haga dos 1 y un 0.

Esto se da en fila 2 } Roberto y Camila dicen la verdad } Pedro miente

En fila 2 } $P=0$ } $q=0$ } $r=1$ } Pedro robó

② Formaliza expresiones, resuelve tabla de verdad e indica si se trata de tautología, contradicción o inconsistencia

- Si los elefantes vuelan \square supieran tocar la guitarra, entonces dejaría que me internaran en un psiquiátrico

p: los elefantes vuelan q: los elefantes saben tocar la guitarra r: dejar que me internen en un psiquiátrico

p	q	r	$p \vee q$	Expresión
0	0	0	0	$(p \vee q) \rightarrow r$
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Se trata de una inconsistencia
(hay 0 y 1 en la 6^a columna)

- Si y sólo si viera un extraterrestre con mis propios ojos, creería que hay vida en otro planeta

$$\text{Si y sólo si} \Leftrightarrow \begin{array}{c|c|c} p & q & p \Leftrightarrow q \\ \hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$$

p: ver un extraterrestre con mis propios ojos

q: creer en la vida en otro planeta

p	q	$p \Leftrightarrow q$	Expresión
0	0	1	
0	1	0	
1	0	0	
1	1	1	

Se trata de inconsistencia

- Vi la serie aunque no leí la novela

p: ver la serie q: leer la novela

p	q	$\neg q$	$p \wedge \neg q$	Expresión
0	0	1	0	
0	1	0	0	
1	0	1	1	
1	1	0	0	

Inconsistencia

- No me gusta madrugar \neg me gusta trastocar

p: me gusta madrugar q: me gusta trastocar

p	q	$\neg p$	$\neg q$	$\neg p \wedge \neg q$	Expresión
0	0	1	1	1	
0	1	1	0	0	
1	0	0	1	0	
1	1	0	0	0	

Inconsistencia

Matriz bono (1 verdadero 0 falso)

edificio $\rightarrow \frac{1}{0}$
vacío $\rightarrow 0$

Bono: lo forman un grupo de edificios (1s conectados entre sí en las ocho direcciones)

Ejemplo

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

y bonos (si edificio está se lo cuenta como bono)

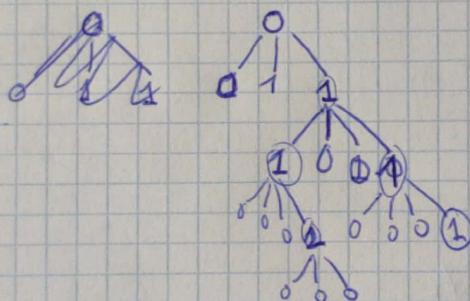
Matriz

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

(2) Algoritmo de búsqueda con el que se puede resolver
debido a que hay que encontrar los contiguos usando búsqueda en profundidad. Además viene un
lote ~~algoritmo~~ por almacenar los estados visitados y así no repetir

(3) La principal desventaja de este método de búsqueda no es óptimo (no garantiza solucionar a menor profundidad)

(2) El algoritmo partira del (0,0) y se expandirá con sus tres descendientes



Habrá que generar hasta
atras y luego unir