EJERCICIOS DE TABLAS DE VERDAD

SIMBOLOS USADOS y aclaraciones

˅ **union**, (ó) Si A=1 y/o B=1 entonces A v B = 1, sino, = 0
^ **interseccion** (y) (A=1, B=1 aˆB--> 1, resto de casos aˆB--> 1)
~ **negado** (NO)

Inconsistencia: la solucion tiene tanto ceros como unos
Tautologia: siempre se cumple (todo unos)
Contradiccion: nunca se cumple (todo ceros)
Falacia: argumento inválido con apariencia de válido porque combina premisas verdaderas con una conclusión falsa.
Ejemplo falacia:
si llueve se moja la calle. La calle está mojada. Entonces llueve --> falacia

p	q	P=>Q	P<=>Q	P triangulo Q (disyuncion exclusiva)
0	0	1	1	0
0	1	1	0	1
1	0	0	0	1
1	1	1	1	0

EJERCICIO

Resolver un problema.
Un día mientras cuatro amigos jugaban póker, intentaban formar una escalera real de corazones y cada uno de ellos afirmó lo siguiente:

> Carlos afirmó: "si no tengo una carta de corazones, entonces lloverá"
> Daniel afirmó: "si tengo una carta de corazones, entonces no lloverá"
> Santiago afirmó: "si no tengo una carta de corazones, entonces no lloverá"

Todos tienen la misma posibilidad de tener la carta de corazones para formar la escalera real y terminado el juego comience a llover.
El último amigo que los escuchaba, David dijo: "uno de ustedes ha hecho una afirmación falsa".
¿Quién de los tres amigos mintió?

> p = “tengo una carta de corazones”
> q = “lloverá”

Formalizar expresiones. Formalizar expresiones y resolverlas con tablas de verdad. Indicar si son tautología, contradicción, o inconsistencia.
SOLUCIÓN PROPUESTA

Lo que dice Carlos: ~p => q

Lo que dice Daniel: p => ~q

Lo que dice Santi: ~p => ~q

Se implementa la tabla de verdad:

				Carlos	Daniel	Santiago
p	q	~p	~q	~p=>q	P=>~q	~p=>~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
				(i)	(i)	(i)

La fila marcada indica la combinación donde "se saca el corazón" (p=1) y "llueve" (q=1), con lo que Daniel miente.

Nota: 'i' = inconsistencia; 't'=tautología; 'c'=contradicción.

EJERCICIO

Ej) Si los elefantes volaran o supieran tocar la guitarra, entonces dejaría que me internaran en un psiquiátrico.
A=elefantes vuelan

B=elefantes saben tocar la guitarra

C=dejar que me internen en un psiquiátrico

Expresión: A v B => C

A	B	A v B	C	AvB=>C
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	0	0
0	0	0	1	1
0	1	1	1	1
1	0	1	1	1
1	1	1	1	1

EJERCICIO

Ej) Si y solo si viera un extraterrestre con mis ojos, creeria que hay vida en otros planetas

A=ver extraterrestre

B = creer que hay vida en otros planetas

Expresión A <=> B

A	B	A<=>B
0	0	1
0	1	0
1	0	0
1	1	1

EJERCICIO

Ej) Vi la serie aunque no leí la novela

Expresión: p ^ ~q

p	q	~q	P^~q
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

EJERCICIO

Ej) No me gusta madrugar ni trasnochar

Expresión: ~p ^~q

p	q	~p	~q	~p^~q
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

EJERCICIO

EJ) Un alumno suspende todas las asignaturas del master y en el telegram se escuchan lamentos...

Hulio Rosch dice: Ni Homunculo Ni yo hemos suspendido
Homunculo de Penfield dice: Hulio o Damasio han suspendido
Damasio el Somático dice: Los dos estais mintiendo como bellacos

La jefa de estudios ha dicho claramente que dos de estos individuos siempre dicen la verdad y el otro siempre miente.

A: Ha suspendido Hulio Rosch
B: Ha suspendido Homunculo de Penfield

C: Ha suspendido Damasio el Somático

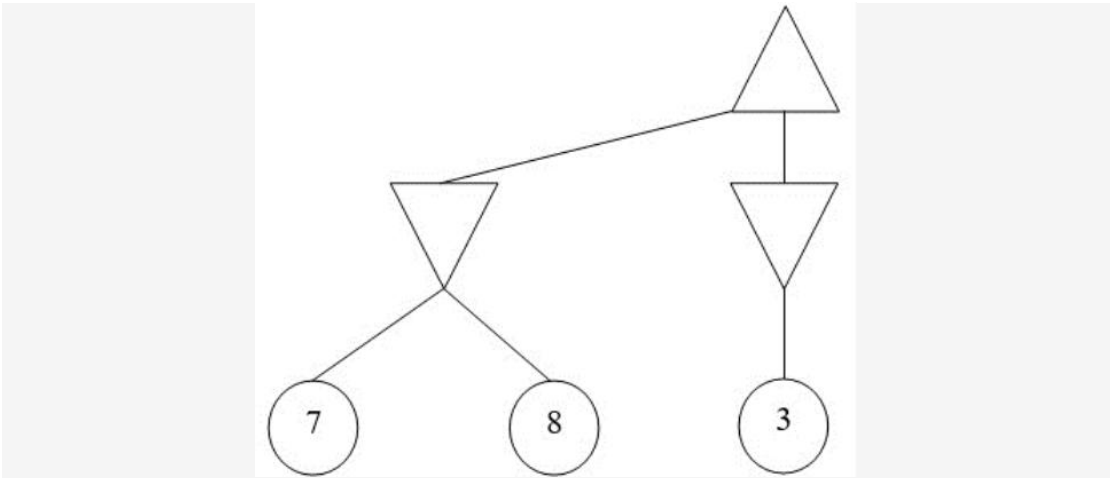
Expresiones:
Lo que dice Hulio Rosch : ~B v ~A
Lo que dice Homunculo de Penfield : A v C
Lo que dice Damasio el Somático: ~(dice Hulio) ^ ~(dice Homunculo)
=~(~B v~A)^~(AvC)

	a	b	c	~B	~A	Lo que dice Hulio	Lo que dice Homunculo	~(Lo que dice Hulio)	~(Lo que dice Homunculo)	Lo que dice Damasio
	0	0	0	1	1	1	0	0	1	0
B	0	1	0	0	1	0	0	1	1	1
A	1	0	0	1	0	0	1	1	0	0
	1	1	0	0	0	0	1	1	0	0
C	0	0	1	1	1	1	1	0	0	0
	0	1	1	0	1	0	1	1	0	0
	1	0	1	1	0	0	1	1	0	0
	1	1	1	0	0	0	1	1	0	0

CASO A (Suspende Hulio): Hulio miente, Homunculo verdad, Damasio miente
CASO B (Suspende Homunculo): Hulio miente, Homunculo miente. Damasio verdad
CASO C (Suspende Damasio): Hulio verdad, Homunculo verdad, Damasio miente

El caso C es el único que cumple que 2 dicen la verdad y uno miente, tal como afirma la jefa de estudios. Damasio el Somático suspende todas las asignaturas.

EJERCICIO GRAFO



Identificar el tipo de algoritmo de búsqueda con el que se ha generado
Completar el árbol asignando valores a los nodos que no tengan
Explicar paso a paso el funcionamiento del algoritmo

RESOLUCIÓN 1 MINIMAX:

Con minimax sencillo

Se construye de abajo hacia arriba

Triangulo con punta hacia abajo, es un minimo, coge valor minimo de sus sucesores.

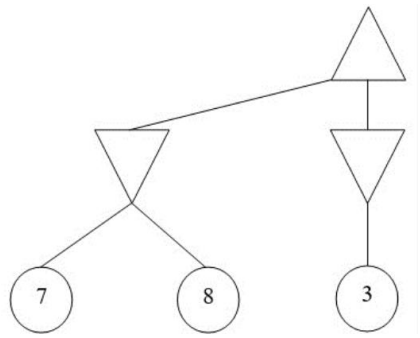
Triangulo con punta hacia arriba es un maximo, coge valor maximo de sus sucesores

RESOLUCIÓN 2 MINIMAX PODA ALFABETA:

Algoritmo: minimax con poda alfa – beta
Compleción del árbol:

PASOS

Triángulos vértice hacia arriba = Nivel Max; Triángulos vértice hacia abajo = Nivel Mi
Se inicializa alfa = -inf, beta = +inf, y se desplaza hacia abajo – izquierda, como sigue:



Se chequea en el nivel Min, con respecto a los nodos terminales, y los valores vigentes de Beta:

7 < Beta? = 7 < inf?

8 < Beta? = 8 < 7?

MejorUtilidad(min) = 7

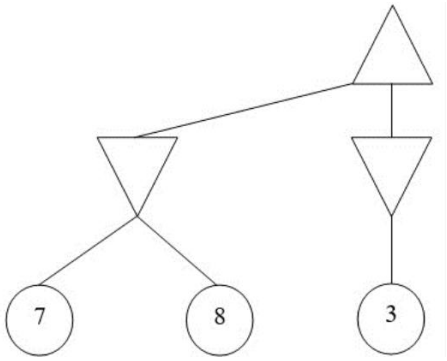
Se asciende al nivel Max, y se compara con los valores vigentes de Alfa:

7 > Alfa? = 7 > -inf?

MejorUtilidad = 7

Se actualiza grafo y valores, se desplaza Alfa y Beta al siguiente

sucesor:



Nuevamente, se inicia en el nivel de Min, por lo que se usa Beta para la comparación.

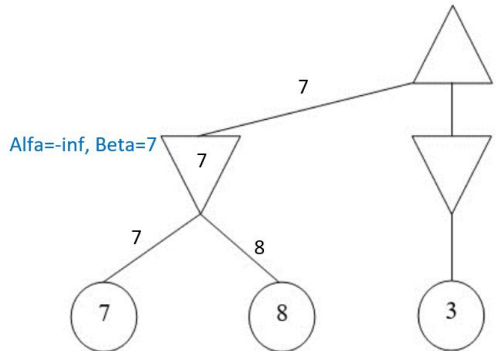
3 < Beta? = 3 < inf?

Condición de poda: Beta < Alfa? = 3 < 7?

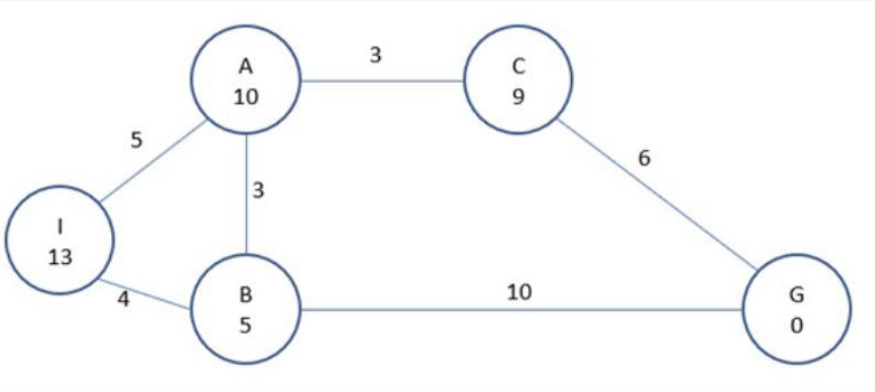
Se desplaza el valor de Utilidad hacia nivel Max: 3

Ya no quedan mas nodos, se escoge utilidad máxima en el nivel de max: 7

El grafo final queda como sigue:



EJERCICIO DADO ARBOL



- Aplicar correctamente al algoritmo X
- Indicar ciertas situaciones del algoritmo, por ejemplo, lista de nodos abiertos o cerrados en cada iteración.

RESOLUCION 1: COSTO UNIFORME

Algoritmo Costo Uniforme

Nodo inicial = [I]

Visitados: []

ColaAbierta: [(I,0)]

1)

Nodo = I; I es un estado final?

Visitados: [I]

ColaAbierta: [(B,4), (A,5)]

3)

Nodo = B; B es un estado final?

Visitados: [I, B]

ColaAbierta: [(A,5), (G,14)]

4)

Nodo = A; A es un estado final?

Visitados: [I, B, A]

ColaAbierta: [(C,8), (G,14)]

5)

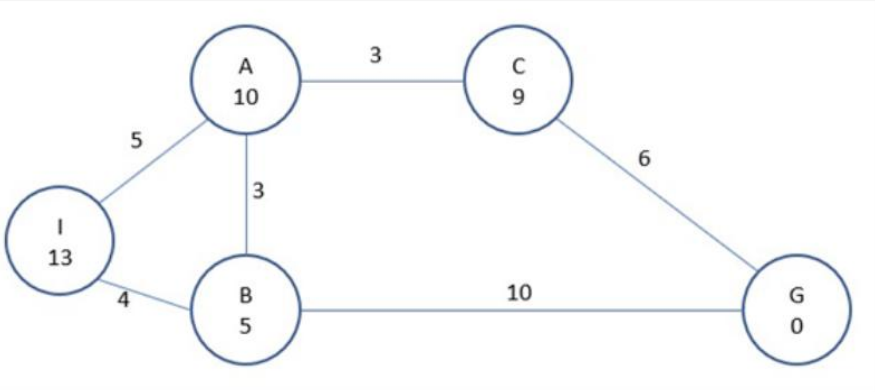
Nodo = C; C es un estado final?

Visitados: [I, B, A, C]

ColaAbierta: [(G,14)]

6)

Nodo = G; G es un estado final? retornar camino: I – B – A – C – G, Coste=14



- Aplicar correctamente al algoritmo X
- Indicar ciertas situaciones del algoritmo, por ejemplo, lista de nodos abiertos o cerrados en cada iteración.

RESOLUCION A*

Algoritmo A*

Nodo Inicial: I, Nodo final: G

colaAbierta = [I]; I.g = 0, I.h=13, I.f = I.g + I.h = 13

1)

Nodo = I; colaAbierta = []; colaCerrada = []

I es meta

Sucesores = [A,B]

Para ‘A’:
g = I.g + c(I,A) = 0 + 5 = 5
A.padre = I
A.g = g = 5
A.f = A.g + h’(A) = 5 + 10 = 15
A no está en colaCerrada con ‘f mejor’: colaAbierta = [A_{15}] (el subíndice es el valor de ‘f’)

Para ‘B’:
g = I.g + c(I,B) = 0 + 4 = 4
B.padre = I
B.g = g = 4
B.f = A.g + h’(B) = 4 + 5 = 9
B no está en cola cerrada con ‘f mejor’: colaAbierta = [B_9, A_{15}]
colaCerrada = [I_{13}]

2)
Nodo = B; colaAbierta = [A_{15}]
B es final?

Sucesores = [I,A,G]

Para I:
g = B.g + c(B,I) = 4 + 4 = 8
I.padre = B
I.g = g = 8
I.f = I.g + h'(I) = 8 + 13 = 21
I está en cola cerrada con 'f mejor' [21]

Para A:
g = B.g + c(B,A) = 4 + 3 = 7
A.padre = B
A.g = g = 7
A.f = A.g + h'(A) = 7 + 10
A no está en colaCerrada con 'f mejor' [, A17]

Para G:
g = B.g + c(B,G) = 4 + 10 = 14
G.padre = B
G.g = g = 14
G.f = G.g + h'(G) = 14
G no está en colaCerrada con 'f mejor' [, A5, A17]

colaCerrada = [I,B]

3)
Nodo = G ; colaAbierta = [A5, A17]
G es nodo final?

RESOLUCION COSTO UNIFORME
Algoritmo Costo Uniforme

No toma en cuenta heurística!

Nodo inicial = [I]
Visitados: []
ColaAbierta: [(I,0)]

1)
Nodo = I; I es final?
Visitados: [I]
ColaAbierta: [(B,4), (A,5)]

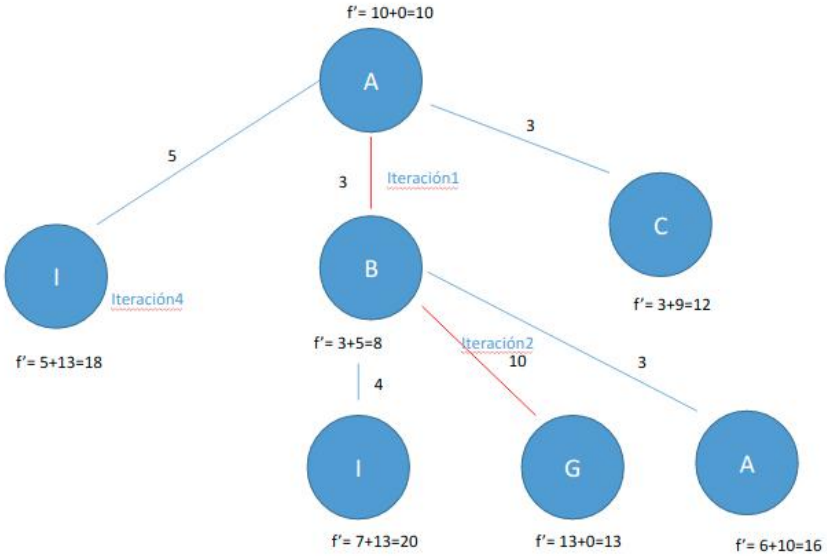
3)
Nodo = B; B es final?
Visitados: [I,B]
ColaAbierta: [(A,3) ~~(A,5)~~] (se reemplaza por coste menor)

4)

Nodo = A; A es final?
Visitados: [I,B, A]
ColaAbierta: [(C,3)]

5)
Nodo = C; C es final?
Visitados: [I,B,A,C]
ColaAbierta: [(A,3),(G,6)]

6)
Nodo = A; A YA FUE VISITADO, SE VA POR EL SIGUIENTE
Nodo = G; G es final? retornar camino: I – B – A – C – G



A*
Asumimos que el nodo Inicial es A y el final G
Asumimos que el número que está dentro de cada nodo en el enunciado es h'

- En cada nodo sumo el coste real de llegar a cada nodo (situado sobre las fechas en el diagrama) más el la heurística del nodo destino (dentro de cada nodo en el grafo del enunciado
- En cada paso selecciono el nodo de menor f' y lo expando hasta que llego al nodo G
- El resultado final marcado en rojo: A>B>G

En cada nodo sumo coste real de llegar a el
En cada paso selecciono menor f' y lo expando
Resultado: ABG

Algoritmo A*

A.g = 0
A.f = 10
Mete A en la cola de prioridades
Saca A de la cola de prioridades
Es A la meta? No
Expande A -> (I,B,C)

g= A.g + g(A,I) = 0+5=5
I.padre=A
I.g=g=5
I.f=g+h'(I)=5+13=18
I no está en la cola cerrada con mejor f': listaAbierta <- I

g= A.g + g(A,B) = 0+3=3
B.padre=A
B.g=g=3
B.f=g+h'(B)=3+5=8
B no está en la cola cerrada con mejor f': listaAbierta <- B

g= A.g + g(A,C) = 0+3=3
C.padre=A
C.g=g=3
C.f=g+h'(C)=3+9=12
C no está en la cola cerrada con mejor f': listaAbierta <- C

Mete A en la lista cerrada

listaAbierta(B,C,I)
listaCerrada(A)

Saca B de la cola de prioridades
Es B la meta? No
Expande B -> (G,I)
g= B.g + g(B,G) = 3+10=13
G.padre=B
G.g=g=13
G.f=g+h'(G)=13+0=13
G no está en la cola cerrada con mejor f': listaAbierta <- G

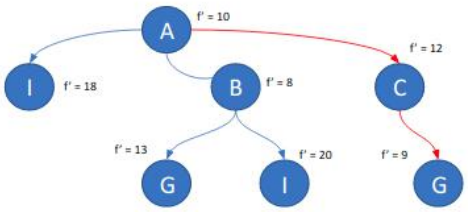
g= B.g + g(B,I) = 3+4=7
I.padre=B
I.g=g=7
I.f=g+h'(I)=7+13=20
I no está en la cola cerrada con mejor f': listaAbierta <- I
Mete B en la lista cerrada

listaAbierta(C,G,I,I)
listaCerrada(A,B)

Saca C de la cola de prioridades
Es C la meta? No
Expande C -> (G)
g= C.g + g(C,G) = 3+6=9
G.padre=C
G.g=g=9
G.f=g+h'(G)=9+0=9
G no está en la cola cerrada con mejor f': listaAbierta <- G
Mete C en la lista cerrada

listaAbierta(G,G,I,I)
listaCerrada(A,B,C)

Saca G de la cola de prioridades
G es la meta: devuelve camino hasta el nodo raiz



PROBLEMA DEL BUSCAMINAS

RESOLUCION CON DFS recursivo.

Se definiría una función recursiva cuyos atributos de entrada sea la matriz con el estado en un momento determinado y unas coordenadas (x,y), y un vector con los visitados.

Se comprueba el valor de la posición (x,y) es M. Si es así, se devolverá la matriz de entrada actualizada con el valor de la posición (x,y) a X

Si no es una mina, se marca el nodo como visitado y se calcularía el número de minas alrededor suyo.

Si el número es mayor que 0, se sustituye ese valor en la celda (x,y)

Si el número es igual a que 0, se sustituye el valor de la celda por 'B'. Se buscan los adyacentes a dicha celda y para cada uno de ellos, si no está visitado, se hace una llamada de nuevo a la función pasándole la matriz de estado y el vector de nodos visitados actualizados y la posición correspondiente al nodo sucesor.

Se devuelve la matriz de estado y el vector de nodos visitados actualizados.

Para contar el número de minas se mira las celdas adyacentes a una dada y se cuenta el número de M que hay en dichas celdas.

Ejemplo de ejecución.

Partiendo de la situación inicial:

V	V	V
M	V	V
M	V	V

en la posición (0,2) y con la lista de nodos visitados vacía.

Se revisaría el valor de la celda (0,2), al ser V, se marca como visitado, se calcula el número de M en las celdas adyacentes. En este caso es 0. Se sustituye el valor de la celda por B y se calculan los sucesores como posiciones adyacentes de la celda.

Visitados = [(0,2)]

V	V	B
M	V	V
M	V	V

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

Para cada nodo se llama a la función recursivamente:

(0,1):

Se calcula el valor de la celda (0,1) = V no M

Se marca como visitado Visitados = [(0,2), (0,1)]

Se calcula el número de M alrededor de la celda: 1, se sustituye por su valor.

Como no es 0 se devuelve el estado y los visitados

V	1	B
M	V	V
M	V	V

(1,1):

Se calcula el valor de la celda (1,1) = V no M

Se marca como visitado Visitados = [(0,2), (0,1), (1,1)]

Se calcula el número de M alrededor de la celda: 2, se sustituye por su valor.

Como no es 0 se devuelve el estado y los visitados

V	1	B
M	2	V
M	V	V

(1,2)

Se calcula el valor de la celda (1,2) = V no M

Se marca como visitado Visitados = [(0,2), (0,1), (1,1), (1,2)]

Se calcula el número de M alrededor de la celda: 0, se sustituye por B.

V	1	B
M	2	B
M	V	V

Se calculan los sucesores Sucesores =[(0,2),(0,1),(1,1),(2,1),(2,2)]

Para cada sucesor, no visitado, se llama a la función DFS

(0,2): visitado anteriormente, no se hace nada

(0,1): visitado anteriormente, no se hace nada

(1,1): visitado anteriormente, no se hace nada.

(2,1): no visitado

Se calcula el valor de la celda (2,1) = V, no M

Se marca como visitado Visitados = [(0,2), (0,1), (1,1), (1,2), (2,1)]

Se calcula el número de M alrededor de la celda: 2, se sustituye por su valor.

V	1	B
M	2	B
M	2	V

Como no es 0 se devuelve el estado y los visitados

(2,2): no visitado

Se calcula el valor de la celda (2,2) = V, no M

Se marca como visitado Visitados = [(0,2), (0,1), (1,1), (1,2), (2,1), (2,2)]

Se calcula el número de M alrededor de la celda: 0, se sustituye por B.

Como no es 0 se calculan los sucesores

Sucesores =[(1,2),(1,1),(2,1),(2,2)]

Para cada sucesor no visitado se vuelve a llamar a DFS:

(1,2): visitado, no se hace nada

(1,1): visitado, no se hace nada

(2,1): visitado, no se hace nada

(2,2): visitado, no se hace nada

V	1	B
M	2	B
M	2	B

Se devuelve el estado y los visitados

V	1	B
M	2	B
M	2	B

No hay mas nodos, se devuelve el estado y los visitados:

V	1	B
M	2	B
M	2	B

Se devolvería el estado final:

RESOLUCION CON BFS

1.

Elaboración del grafo en el que aparecerían las casillas que sería necesario explorar en cada casilla considerando que se marca la [0,2] a partir del estado inicial siguiente:

V	V	V
M	V	V
M	M	V

[0,2] -> [0,1] -> [0,0], [1,0], [1,1] , [1,2]

[0,2] -> [1,1] -> [0,0], [0,1], [1,0], [1,2], [2,0], [2,1], [2,2]

[0,2] -> [1,2] -> [0,1], [1,1], [2,1], [2,2]

2

Aplico el algoritmo BFS que me irá indicando cómo visitar las casillas:

2.1)

Cola: [2,2],

Visitados: [2,2]

2.2) Sucesores de [2,2]

Cola: ~~[2,2],~~ [0,1], [1,1], [1,2]

Visitados: [2,2], [0,1], [1,1], [1,2]

2.3) Sucesores [0,1]

Cola: ~~[0,1],~~ [1,1], [1,2], [0,0], [1,0]. No añado [1,1] , [1,2] porque ya están visitados

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0]

2.4) Sucesores [1,1]

Cola: ~~[1,1],~~ [1,2], [0,0], [1,0], [2,0], [2,1]. No añado [0,0], [0,1], [1,0], [1,2], [2,2] porque ya están visitados

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1].

2.5) Sucesores [1,2]

Cola: ~~[1,2],~~ [0,0], [1,0], [2,0], [2,1]. No añado [0,1], [1,1], [2,1], [2,2] porque ya están visitados

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1].

2.6) Sucesores [0,0]. No hay

Cola: ~~[0,0],~~ [1,0], [2,0], [2,1].

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1].

2.7) Sucesores [1,0]. No hay

Cola: ~~[1,0],~~ [2,0], [2,1].

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1].

2.8) Sucesores [2,0]. No hay

Cola: ~~[2,0],~~ [2,1].

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1].

2.9) Sucesores [2,1]. No hay

Cola: ~~[2,1].~~

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1].

RESULTADO

Órden final:

[0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1]2.

RESOLUCION CON BFS DE MODO MÁS VISUAL

1.

Elaboración del grafo en el que aparecerán las casillas que sería necesario explorar en cada casilla considerando que se marca la [0,2] a partir del estado inicial siguiente:

2.

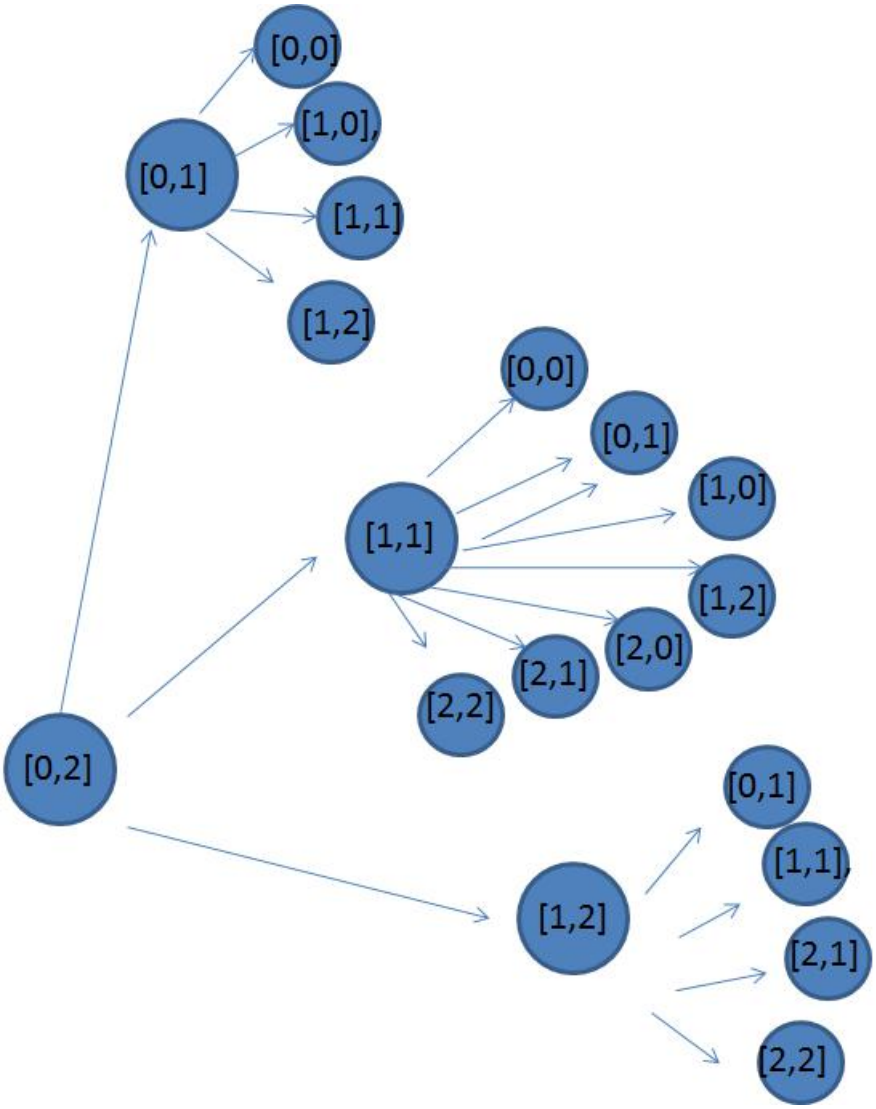
“**Matrix**”		y		
		0	1	2
x	0	**V**	**V**	**V**
	1	**M**	**V**	**V**
	2	**M**	**M**	**V**

[0,2] -> [0,1] -> [0,0], [1,0], [1,1] , [1,2]

[0,2] -> [1,1] -> [0,0], [0,1], [1,0], [1,2], [2,0], [2,1], [2,2]

[0,2] -> [1,2] -> [0,1], [1,1], [2,1], [2,2]

Visto en estructura de árbol:



1.

Aplico el algoritmo BFS que me irá indicando cómo visitar las casillas:

2.

2.1)

Cola: [0,2],

Visitados: [0,2]

2.2)

Nodo: [0,2]; NewMatrix(0,2) = B

Sucesores: [0,1], [1,1], [1,2]

Visitados: [2,2], [0,1], [1,1], [1,2]

Cola: [0,1], [1,1], [1,2]

“**NewMatrix**”		y		
		0	1	2
x	0			**B**
	1			
	2			

2.3)

Nodo: [0,1]; NewMatrix(0,1) = 1

Sucesores: [0,0], [1,0]

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0]

Cola: [1,1], [1,2], [0,0], [1,0]

“**NewMatrix**”		y		
		0	1	2
x	0		**1**	**B**
	1			
	2			

2.4)

Nodo: [1,1]; NewMatrix(1,1) = 3

Sucesores: [2,0], [2,1], [2,2]

Visitados: [2,2], [0,1], [1,1], [1,2], [0,0], [1,0], [2,0], [2,1], [2,2]

Cola: [1,2], [0,0], [1,0], [2,0], [2,1], [2,2]

“**NewMatrix**”		y		
		0	1	2
x	0		**1**	**B**
	1		**3**	
	2			

2.5)

Nodo: [1,2]; NewMatrix(1,2) = 1

Sucesores: ninguno

Visitados: sin cambios

Cola: [0,0], [1,0], [2,0], [2,1], [2,2]

“**NewMatrix**”		y		
		0	1	2
x	0		**1**	**B**
	1		**3**	**1**
	2			

2.6)

Nodo: [0,0]; NewMatrix(0,0) = 1

Sucesores: ninguno

Visitados: sin cambios

Cola: [1,0], [2,0], [2,1], [2,2]

“**NewMatrix**”		y		
		0	1	2
x	0	**1**	**1**	**B**
	1		**3**	**1**
	2			

2.7)

Nodo: [1,0]; NewMatrix(1,0) = X

Sucesores: ninguno

Visitados: sin cambios

Cola: [2,0], [2,1], [2,2]

“**NewMatrix**”		y		
		0	1	2
x	0	**1**	**1**	**B**
	1	**X**	**3**	**1**
	2			

2.8)

Nodo: [2,0]; NewMatrix(2,0) = X

Sucesores: ninguno

Visitados: sin cambios

Cola: [2,1], [2,2]

“**NewMatrix**”		y		
		0	1	2
x	0	**1**	**1**	**B**
	1	**X**	**3**	**1**
	2	**X**		

2.9)

Nodo: [2,1]; NewMatrix(2,1) = X

Sucesores: ninguno

Visitados: sin cambios

Cola: [2,2]

“**NewMatrix**”		y		
		0	1	2
x	0	**1**	**1**	**B**
	1	**X**	**3**	**1**
	2	**X**	**X**	

2.10)

Nodo: [2,2]; NewMatrix(2,2) = 1

Sucesores: ninguno

Visitados: sin cambios

Cola: []

“**NewMatrix**”		y		
		0	1	2
x	0	**1**	**1**	**B**
	1	**X**	**3**	**1**
	2	**X**	**X**	**1**

CONSIDERACIONES:

Se debe construir una función que se lleve la “Matrix” original, internamente se construye y eventualmente retorna una “NewMatrix” de las mismas dimensiones, con el problema resuelto. Dentro del algoritmo BFS, se debe eliminar la sección de ‘nodo=G’, y dejar que el algoritmo agote su cola por cuenta propia.

OTRA VERSION CON ANCHURA

Algoritmo Anchura

Nodo_inicial = (0,2)
Nodo_inicial.minas_vecinas = 0
Marca (0,2) con una B
colaAbierta.queue(Nodo_inicial)

Mientras colaAbierta contenga nodos:
  Nodo = colaAbierta.dequeue() = (0,2)
  listaCerrada.insert(0,2)
  Nodo.sucesores = [(0,1),(1,1),(1,2)]

    sucesor = (0,1)
    (0,1) no está en lista cerrada, entonces:
      sucesor.minas_vecinas = 1
      sucesor.minas_vecinas es distinto de 0:
        Marca (0,1) con un 1
        listaCerrada.insert(0,1)

    sucesor = (1,1)
    (1,1) no está en lista cerrada, entonces:
      sucesor.minas_vecinas = 3
      sucesor.minas_vecinas es distinto de 0:
        Marca (1,1) con un 3
        listaCerrada.insert(1,1)

    sucesor = (1,2)
    (1,2) no está en lista cerrada, entonces:
      sucesor.minas_vecinas = 1
      sucesor.minas_vecinas es distinto de 0:
        Marca (1,1) con un 1
        listaCerrada.insert(1,1)

colaAbierta []

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

colaAbierta ha quedado vacía: devuelve el tablero

Algoritmo Anchura

Nodo_inicial = (0,2)
Nodo_inicial.minas_vecinas = 0
Marca (0,2) con una B
colaAbierta.queue(Nodo_inicial)

Mientras colaAbierta contenga nodos:
Nodo = colaAbierta.dequeue() = (0,2)
listaCerrada.insert(0,2)
Nodo.sucesores = [(0,1),(1,1),(1,2)]

sucesor = (0,1)
(0,1) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 0
sucesor.minas_vecinas == 0:
Marca (0,1) con B
colaAbierta.queue((0,1))

sucesor = (1,1)
(1,1) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 1
sucesor.minas_vecinas es distinto de 0:
Marca (1,1) con un 1
listaCerrada.insert(1,1)

sucesor = (1,2)
(1,2) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 0
sucesor.minas_vecinas == 0:
Marca (1,2) con B
colaAbierta.queue((1,1))

colaAbierta [(0,1),(1,2)]
listaCerrada [(0,2),(1,1)]

Nodo = colaAbierta.dequeue() = (0,1)
listaCerrada.insert(0,1)
Nodo.sucesores = [(0,0),(1,0),(1,1),(1,2),(0,2)]

sucesor = (0,0)
(0,0) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 0
sucesor.minas_vecinas == 0:
Marca (0,0) con B
colaAbierta.queue((0,0))

sucesor = (1,0)
(1,0) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 1
sucesor.minas_vecinas es distinto de 0:
Marca (1,0) con un 1
listaCerrada.insert(1,0)

sucesor = (1,1)
(1,1) está en lista cerrada: no hago nada
sucesor = (1,2)
(1,2) está en la cola: no hago nada
sucesor = (0,2)
(0,2) está en lista cerrada: no hago nada

colaAbierta [(1,2),(0,0)]
listaCerrada [(0,2),(1,1),(1,0),(0,1)]

Nodo = colaAbierta.dequeue() = (1,2)
listaCerrada.insert(1,2)
Nodo.sucesores = [(0,2),(0,1),(1,1),(2,1),(2,2)]

sucesor = (0,2)
(0,2) está en lista cerrada: no hago nada
sucesor = (0,1)
(0,1) está en lista cerrada: no hago nada
sucesor = (1,1)
(1,1) está en lista cerrada: no hago nada

sucesor = (2,1)
(2,1) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 1
sucesor.minas_vecinas != 0:
Marca (2,1) con 1
listaCerrada.insert(2,1)

sucesor = (2,2)
(2,2) no está en lista cerrada, entonces:
sucesor.minas_vecinas = 0
sucesor.minas_vecinas == 0:
Marca (2,2) con B
colaAbierta.queue((2,2))

colaAbierta [((0,0),(2,2)]
listaCerrada [(0,2),(1,1),(1,0),(0,1),(1,2),(2,1)]

Nodo = colaAbierta.dequeue() = (0,0)
listaCerrada.insert(0,0)
Nodo.sucesores = [(0,1),(2,1),(1,0)]

sucesor = (0,1)
(0,1) está en lista cerrada: no hago nada
sucesor = (2,1)
(2,1) está en lista cerrada: no hago nada
sucesor = (1,0)
(1,0) está en lista cerrada: no hago nada

colaAbierta [(2,2)]
listaCerrada [(0,2),(1,1),(1,0),(0,1),(1,2),(2,1),(0,0)]

Nodo = colaAbierta.dequeue() = (2,2)

listaCerrada.insert(2,2)
Nodo.sucesores = [(1,1),(1,2),(2,1)]

sucesor = (1,1)
(1,1) está en lista cerrada: no hago nada
sucesor = (1,2)
(1,2) está en lista cerrada: no hago nada
sucesor = (2,1)
(2,1) está en lista cerrada: no hago nada
colaAbierta []
listaCerrada [(0,2),(1,1),(1,0),(0,1),(1,2),(2,1),(0,0),(2,2)]

La cola ha quedado vacía: devuelve tablero

EJERCICIO DE STRIPS

Enunciado:
Se tiene un problema simplificado del mundo de bloques en STRIPS [1]. En el que hay dos bloques A y B que están sobre una mesa M. El bloque B esta 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): sobre(B, A) ^ libre(B) ^ sobre(A, M)
Objetivo (G): sobre(B, M)
Operadores (O):
- MoverSobre (?bloque1, ?bloque2)
Precondiciones: libre(?bloque1) ^ libre(?bloque2)
Efectos: sobre(?bloque1,?bloque2) ^ ¬ libre(?bloque2)
- MoverSobreLaMesa (?bloque1, ?bloque2, ?Mesa)
Precondiciones: libre(?bloque1) ^ sobre(?bloque1, ?bloque2)
Efectos: ¬ sobre(?bloque1, ?bloque2) ^ sobre(?bloque1,?Mesa)
Se deben resolver las siguientes preguntas:

El objetivo está correctamente especificado teniendo en cuenta el enunciado?
Considera que los operadores están bien definidos?
Asumiendo que las acciones están bien definidas. Que acciones son aplicables en el estado inicial? Para cada acción aplicable genere el estado resultante.
Teniendo como nodo raíz el estado inicial, construye el árbol de búsqueda aplicando los algoritmos (este es un ejemplo, en el examen será solo aplicar un algoritmo, y se dirá exactamente cuál):
Anchura,
Profundidad

RESOLUCIÓN:

El objetivo está correctamente especificado teniendo en cuenta el enunciado?

No, requiere especificación / compleción del objetivo: **sobre(A, B) ^ libre(A) ^** sobre(B, M)

Considera que los operadores están bien definidos?

No, se requiere añadir el siguiente segmento:

- MoverSobreLaMesa (?bloque1, ?bloque2, ?Mesa)
Precondiciones: libre(?bloque1) ^ sobre(?bloque1, ?bloque2)
Efectos: ¬ sobre(?bloque1, ?bloque2) ^ sobre(?bloque1,?Mesa) **^ libre(?bloque2)**

Asumiendo que las acciones están bien definidas. Que acciones son aplicables en el estado inicial? Para cada acción aplicable genere el estado resultante.

Acción: MoverSobreLaMesa(B,A,M)

Estado resultante: sobre(B,M), sobre(A,M), libre(A), libre(B)

Teniendo como nodo raíz el estado inicial, construye el árbol de búsqueda aplicando los algoritmos (este es un ejemplo, en el examen será solo aplicar un algoritmo, y se dirá exactamente cuál):
Anchura,
Profundidad

Se representan todos los estados posibles como siguen:

Entonces: EstadoInicial = A, EstadoFinal = C

Algoritmo BFS:

S0 = A

Visitados = [A]

colaAbierta = [A]

nodo = A, colaAbierta = []

A == C?

Sucesores = [B]

Visitados = [A,B]

B.padre = A

colaAbierta = [B]

nodo = B, colaAbierta = []

B == C?

Sucesores = [A,C]

Visitados = [A,B,C]

C.padre = B

colaAbierta = [C]

nodo = C, colaAbierta = []

C == C? Retornar camino: A – B – C

Algoritmo DFS recursivo:

Nodo = A

Visitados = [A]

Sucesores = [B]

B.padre = A

Nodo = B

Visitados = [A,B]

Sucesores = [A,C]

C.padre = B

Nodo = C

Visitados = [A,B,C]

Sucesores = []

Camino = A – B – C