Técnicas de Búsqueda Ciega

1.- La formulación **medianamente formal** (el lenguaje natural no es formal) de los estados, los estados iniciales, los estados objetivo y las reglas de transición de alguno de los apartados, a tu elección, del ejercicio 2 de la relación de ejercicios de búsqueda.

Para este ejercicio he escogido el problema del mono, los plátanos y la caja. Para la representación de sus estados he seguido el vídeo de moodle del siguiente enlace.

El problema consiste en que un mono está en una habitación donde hay unos plátanos colgados del techo, los cuales no alcanza, y se hace ayuda de una caja para llegar a ellos.

Lo primero sería representar los estados posibles, los cuales serán representados con variables booleanas:

bool cajaBajoPlatanos;//esta variable representa si la caja está o no debajo de los plátanos.
bool monoEnCaja;//esta variable representa si el mono está encima de la caja.
bool monoPlatanos;//esta variable representa si el mono tiene los pátanos.

Una vez representados los estados continuamos con los estados inicial y final:

Para el estado inicial solo vamos a suponer que el mono no tiene los plátanos, ya que así el sistema tiene más libertad de empezar de distintas formas (la caja bajo los plátanos o no o el mono encima de la caja o no): monoPlatanos=false.

Para el estado final más de lo mismo, para que el sistema tenga más libertad solo expondremos que el mono consigue los plátanos y no el como: monoPlatanos=true.

A continuación representaremos las reglas de transición que son las posibilidades que tiene el sistema para jugar a este juego, es decir que el mono se suba a la caja está permitido, pero no que el mono vuele hacia los plátanos:

Lo primero que necesitamos es que el mono se suba a la caja, parta representar esto lo haremos como en el lenguaje clips visto en prácticas, con unas condiciones que llevan a unas acciones.

Puede pasar que mientras estamos exponiendo las condiciones se nos ocurran más estados que debemos tener, por ejemplo en este caso para que el mono se suba a la caja necesitamos que no esté subido ya y además que esté cerca de ésta, por lo tanto añadimos ese estado.

Estado nuevo: bool monoCercaCaja;//representa si el mono está cerca de la caja.

Regla de transición: ((monoEnCaja==false)^(monoCercaCaja==true)) → (monoEnCaja==true).

Aunque queramos que el mono se suba a la caja cuando esta esté bajo los plátanos no lo ponemos como condición para que, una vez más, el sistema tenga libertad en sus acciones.

La siguiente regla será que el mono coja los plátanos y se representará así: ((monoPlatanos==false)^(cajaBajoPlatanos==true)^(monoEnCaja==true)) → (monoPlatanos=true).

Otra regla sería que el mono se baje de la caja, por si está subido a ella pero lejos de los plátanos deberá bajarse para moverlos: (monoEnCaja==true) → (monoEnCaja==false).

Una regla más sería que el mono se acerque a la caja para subirse en ella: (monoCercaCaja==false) → (monoCercaCaja==true)

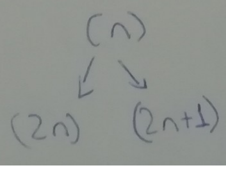
Y por último que el mono mueva la caja, por si esta no está bajo los plátanos, la moverá hasta que lo esté: ((monoCercaCaja==true)^(monoEnCaja==false)^(CajaBajoPlatanos==false)) → (CajaBajoPlatanos==true)

Con esto tendríamos representado el problema mediante sus estados y sus reglas de transición.

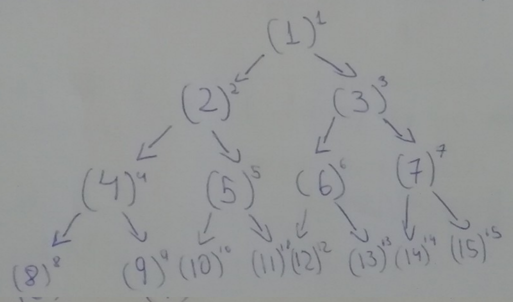
2.

2. Estado Inicial = 1

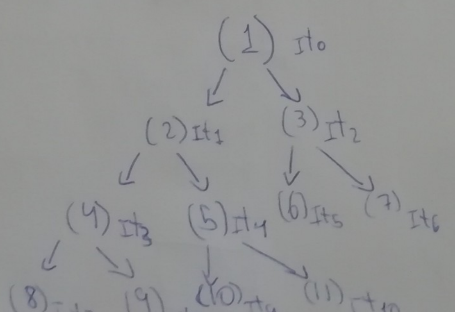
2.1. Cada estado padre genera dos hijos tal que, si el padre es n, su hijo nº1 es 2n y su hijo nº2 es 2n+1.



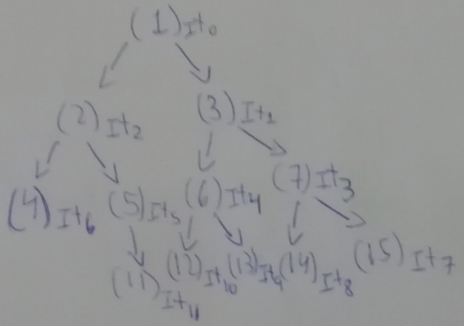
2.2. A continuación dibujaré desde el estado 1 al 15.
(Los estados se numeran encima a la derecha)



2.3. Si aplicamos la búsqueda en anchura los estados se visitan en orden si la hacemos de izquierda a derecha {(1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11)} teniendo 11 iteraciones al final.



Y si la aplicamos de derecha a izquierda a derecha las iteraciones serían 12 {(1),(3),(2),(7),(6),(5),(4),(15),(14),(13),(12),(11)}



2.3. b) Aplicando búsqueda en profundidad iniciando por la rama izquierda las iteraciones serían:
La nomenclatura es (nodo, coste, padre).

It0

F{(1,0,-)}

E{-}

It1

F{(2,1,1),(3,1,1)}

E{(1,0,-)}

It2

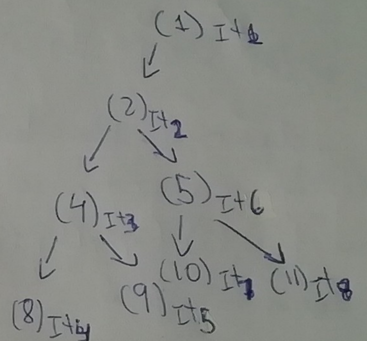
F{(4,2,2),(5,2,2)(3,1,1)}

E{(1,0,-),(2,1,1)}

It3

F{(8,3,4),(9,3,4),(5,2,2),(3,1,1)}

E{(1,0,-),(2,1,1),(4,2,2)}



Uno las iteraciones 4 y 5 porque ni el nodo 8 ni el 9 tienen hijos por lo que solo se pasarán a exploradas

It4/5

F{(5,2,2),(3,1,1)}

E{(1,0,-),(2,1,1),(4,2,2),(8,3,4),(9,3,4)}

It_6

F{(10,3,5), $\boxed{(11,3,5)}$, (3,1,1)}

E{(1,0,-), (2,1,1), (4,2,2), (8,3,4), (9,3,4), (5,2,2)}

Con esta iteración ya tenemos el nodo 11 en frontera, por lo cual hemos acabado, nos ha costado 6 iteraciones, necesitaríamos 9 nodos en memoria en esta iteración, a no ser que borremos los caminos innecesarios ((4), (8), (9), (10)) y el camino es:

(1) -> (2) -> (5) -> (11).

Si empezásemos por la rama derecha sería igual pero nos daría más iteraciones.

2.3.c) Como el límite es tres nunca se encontraría la solución puesto que está en la capa 4.

2.3.d) Como la búsqueda en profundidad iterativa es con límite pero aumentándolo pasaríamos por todos los nodos de las 3 primeras capas, y al aumentar el límite encontraríamos la solución. Las primeras iteraciones serían:

(1) -> (2) -> (4); (2) -> (5); (3) -> (6); (3) -> (7)

Comenzaremos con la It_8:

It_8

F{(8,3,4), (9,3,4)}

E{(1,0,-), (2,1,1), (4,2,2), (5,2,2), (3,1,1), (6,2,3), (7,2,3)}

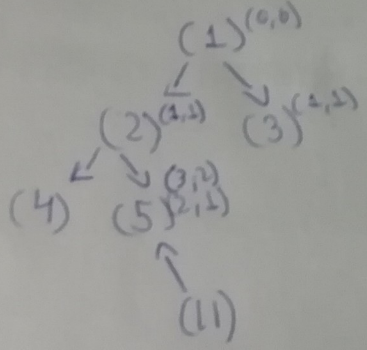
It 9/10 (Las unimos porque no tienen hijos los nodos 8 y 9)

F{(10,3,5), $\boxed{(11,3,5)}$}

E{(1,0,-), (2,1,1), (4,2,2), (5,2,2), (3,1,1), (6,2,3), (7,2,3), (10,3,5), (11,3,5)}

En la iteración 11 se encontraría la solución con 11 nodos en memoria y el camino sería (1) → (2) → (5) → (11)

2.4. Se puede aplicar la búsqueda bidireccional si suponemos que ya tenemos el grafo generado o conocemos el estado final, aunque en este problema es muy simple ya que cada hijo solo tiene un padre. A la derecha anotaremos la iteración y el coste.



El coste es 3, con 3 iteraciones y el camino es

(1) → (2) → (5) → (11)

2.5. Sí porque conocemos el proceso de creación de hijos de una forma no gráfica, la cual es la fórmula de

$n \to 2n$

$ \to 2n+1$.