Ejercicios Búsqueda no informada
Grado en Robótica. Curso 2022-2023.
Departamento de Electrónica y Computación

1. Ejercicio 1

Como es sabido, el algoritmo de búsqueda bidireccional consiste en desarrollar, simultaneamente, dos búsquedas: una desde el nodo inicial, *s*, y otra desde el nodo final, *t*. Se pide determinar si el algoritmo bidireccional resultante será o no completo y/o admisible en cada uno de los siguientes supuestos:

1. Cada una de las búsquedas se implementa en amplitud y los costes son siempre iguales a la unidad
2. Ambas búsquedas se implementan en amplitud y los costes son cantidades arbitrarias estrictamente positivas
3. Ambas búsquedas se implementan en profundidad con costes idénticos y siempre iguales a la unidad
4. Una búsqueda es en amplitud y la otra con profundización iterativa, con costes idénticos y siempre iguales a la unidad

dado que, en todos los casos, las búsquedas se alternan inmediatamente. Esto es, después de una expansión en una de ellas, se cambia inmediatamente a la dirección opuesta y así sucesivamente.

2. Ejercicio 2

1. Resolver el problema de las garrafas mediante el método de búsqueda en amplitud, teniendo en cuenta que:
   - el orden de selección de los operadores es:
     - `llenar grande`
     - `llenar pequeña`
     - `vaciar grande`
     - `vaciar pequeña`
     - `traspasar grande pequeña`
     - `traspasar pequeña grande`
   - suponer que cada vez que se genera un nodo, se elimina si en ese momento está en el árbol de búsqueda, es decir, si se detecta un ciclo
2. ¿Puedes garantizar que la solución encontrada anteriormente es óptima?
3. ¿Podrías encontrar una solución con el algoritmo de búsqueda en profundidad, para una una profundidad máxima de 4? En caso afirmativo, ¿te garantizaría la búsqueda en profundidad que la solución sea óptima?
4. ¿Podrías encontrar una solución con el algoritmo de búsqueda en profundidad, para una una profundidad máxima de 6? En caso afirmativo, ¿te garantizaría la búsqueda en profundidad que la solución sea óptima?
5. ¿Podrías encontrar una solución con el algoritmo de búsqueda en profundidad, para una una profundidad máxima de 8? En caso afirmativo, ¿te garantizaría la búsqueda en profundidad que la solución sea óptima?

6. ¿Podrías encontrar una solución con el algoritmo de búsqueda en profundidad iterativa, para una profundidad máxima inicial de 2, y un incremento de profundidad de 3? En caso afirmativo, ¿puedes garantizar que la solución sea óptima? En caso de no asegurar optimalidad, ¿puedes estimar en qué coste ha sido excedido el óptimo?

7. ¿Se puede resolver este problema con una búsqueda hacia atrás?

8. ¿Y con una búsqueda bidireccional?

3. Ejercicio 3

Se quiere determinar el número mínimo de movimientos que hacen falta para llegar desde cualquier estado del 15-Puzle hasta uno cualquiera de los estados finales mostrados en la figura 1. En la figura, el blanco está representado con una casilla sólida de color gris.

(a) Primer estado final (b) Segundo estado final

Figura 1: El 15-Puzle

1. Considerando que el coste de cada desplazamiento del blanco es siempre igual a la unidad, ¿cómo puede determinarse el número mínimo de movimientos para llegar desde cualquier estado hasta el estado final de la Figura 1(a)?

2. Si intercambiar el blanco con una casilla "rayada" tiene un coste nulo, mientras que el coste de desplazar el blanco a una casilla numerada es igual a la unidad, ¿cómo puede determinarse ahora el número mínimo de movimientos para llegar desde cualquier permutación con los mismos símbolos que hay en la Figura 1(b) hasta el estado final de la Figura 1(b)?

4. Ejercicio 4

Dado el árbol de la Figura 2 donde B y L son los 2 únicos nodos meta y A es el nodo inicial. Indica en qué orden se visitarían los nodos, distinguiendo nodos generados de nodos expandidos, para los siguientes algoritmos:

1. Amplitud

2. Profundidad

3. Mejor primero, tomando como mejor nodo en cada ramificicación aquel con menor orden alfabético

5. Ejercicio 5

Mi monovolumen nuevo es muy versátil. En principio, pueden montarse hasta 7 personas: 2 delante, 3 detras, y 2 en una tercera fila de asientos. Todos los asientos son individuales e independientes. Además, tiene un maletero de 200 litros. Todos los asientos son abatibles y se pueden ocultar en el suelo, o sacar del vehículo. Cuando se abate un asiento, se ganan 200 litros de capacidad. Además, los asientos de la segunda fila y del

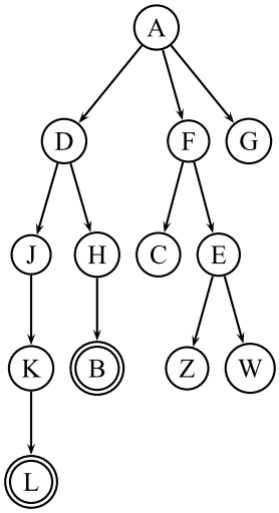

Figura 2: Árbol de búsqueda

copiloto se pueden desplazar hacia adelante, ganando un volumen de 50 litros por cada asiento que se mueve. Sin embargo, el asiento del copiloto solo se puede abatir o desplazar si se han abatido los dos asientos que tiene detrás (en la segunda fila, el central y el derecho), y nunca se puede dar la situación de que el asiento del copiloto esté abatido, y los dos que tiene detrás esté en posición normal. Los asientos solo se pueden abatir o desplazar si están en posición normal. Por último, en cada asiento solo se puede sentar una persona.

Todas las acciones de abatir y desplazar asientos son individuales. Es decir, no se pueden abatir ni desplazar dos asientos a la vez. Además, cada una de esas operaciones tiene un coste: abatir un asiento tiene coste 2, y desplazarlo tiene coste 1. Las operaciones inversas (desabatir y desplazar a posición original) tienen el mismo coste.

Se pide:

- Describir el espacio de estados del problema
- Representar un estado inicial consistente en que todos los asientos están en su posición inicial (sin desplazar ni abatir), excepto los dos asientos de la tercera fila, que estar an abatidos. Además, en ese estado inicial no habrá carga.
- ¿Cuál es el tamaño del espacio de estados?

6. Solución del ejecicio 1

Al contrario que ocurriera con la búsqueda unidireccional, en este caso es preciso considerar una cuestión adicional a la hora de resolver la completitud de una implementación bidireccional, esto es, si las dos búsquedas coincidirán o no. Si puede demostrarse que necesariamente lo harán en un nodo cualquiera, n, entonces puede concluirse, asimismo, que necesariamente encontrarán una solución (y, por ello, que el algoritmo bidireccional en cuestión es completo) que estará constituida, entonces, por la concatenación de los caminos desde el nodo inicial, s hasta n, $\langle s, \ldots, n\rangle$ y desde él hasta t, $\langle n, \ldots, t\rangle$: $\langle s, \ldots, n, \ldots, t\rangle$.

Por otra parte, la admisibilidad de un algoritmo bidireccional se resuelve, en cada caso, comprobando si el camino desde cada extremo (ya sea el nodo inicial, s, o el nodo final, t) hasta el nodo de coincidencia, n, es óptimo. Si lo fuera, la solución que resulta de la concatenación necesariamente lo será y con ello se habría probado la optimalidad de las soluciones encontradas o, equivalentemente, la admisibilidad del algoritmo bidireccional en cuestión.

Por lo tanto, en los siguientes apartados se discuten, para cada caso, estas cuestiones.

1. Si existe una solución, necesariamente existirá un camino desde el nodo inicial *s* hasta el nodo final o meta, *t*. Alternando la dirección de las búsquedas después de cada expansión, el nodo coincidente será el que se encuentra necesariamente en la mitad de este camino. Puesto que cada uno de los algoritmos de el primero en amplitud es completo, ambos encontrarán, irremediablemente, el nodo de coincidencia *n*. Por lo tanto, el algoritmo bidireccional resultante es necesariamente completo.

   Más aún, puesto que cada búsqueda de el primero en amplitud es admisible, ambas encontrarán el camino óptimo hasta el nodo de coincidencia *n*. Por supuesto, la concatenación de los caminos con el menor coste, resultará necesariamente en un camino óptimo y, por lo tanto, el algoritmo resultante es necesariamente admisible.

2. Como antes, el algoritmo bidireccional en cuestión es necesariamente completo. En realidad, la completitud de el algoritmo de el primero en amplitud, resulta del uso de una cola, puesto que así se garantiza que *"nunca se expande un nodo a profundidad* $(d + 1)$ *si antes no se han expandido todos los nodos a profundidad"*, independientemente del coste de los operadores. Como antes, la completitud de los algoritmos empleados en ambas direcciones, garantizan la completitud de la implementación bidireccional resultante.

   Sin embargo, ahora no sucede que cuando se genera un nodo cualquiera, *n*, por primera vez, se haga con el mínimo coste como ocurriera cuando los operadores tienen costes iguales a la unidad. Por lo tanto, el algoritmo de el primero en amplitud no es admisible en este caso. Si no lo es, no puede garantizarse que los caminos encontrados desde cada extremo hasta el nodo de coincidencia, *n*, sean óptimos y, por ello, que el camino resultante lo sea. Por lo tanto, el algoritmo bidireccional resultante no es admisible en este caso.

3. El algoritmo de el primero en profundidad no es completo. Si no es completo, se sigue inmediatamente que no es admisible, puesto que si no no puede garantizarse que encuentre una solución, ¡aún menos que encuentre la solución óptima! Por lo tanto, si no puede garantizarse que cada una de las búsquedas encontrará, eventualmente el nodo *n*, se sigue que esta implementación bidireccional no es completa.

   Como antes, no siendo completa, se sigue de inmediato que tampoco es admisible.

4. Como se sabe, el algoritmo de el primero en amplitud es completo. Por lo tanto, mientras que puede garantizarse que el algoritmo de el primero en amplitud encontrará el nodo de coincidencia, *n*, el algoritmo bidireccional resultante no es completo. Esto es así aunque, como se sabe, ¡el algoritmo de profundización iterativa es también completo, de modo que eventualmente alcanzará el mismo nodo *n*!

   El motivo es que no puede garantizarse que ambos nodos lleguen al nodo de coincidencia en el mismo instante, de modo que ambas búsquedas se cruzarían: cada vez que el algoritmo de profundización iterativa reinicia la búsqueda, creando una nueva pila en la que introduce únicamente el nodo inicial, el algoritmo de el primero en amplitud (en la dirección opuesta) expandirá unos cuantos nodos antes de que el algoritmo de profundización iterativa haya dejado de regenerar nodos para empezar a generar otros nuevos. Si el nodo n fuera alguno de los nodos expandidos por el algoritmo de el primero en amplitud mientras el algoritmo de profundidad iterativa llegaba a un nuevo área del espacio de estados, la coincidencia resultaría inadvertida para el algoritmo de profundización iterativa, que llegaría a él, pero más tarde.

7. Solución del ejercicio 2

1. El problema se resuelve expandiendo 40 nodos, en una profundidad de 6.

   A continuación, se desarrolla la ejecución del algoritmo. Para empezar, se introduce el nodo inicial, $\langle 0, 0\rangle$ en la lista abierta, y se comienza el proceso iterativo del algoritmo. En ese momento, el árbol de búsqueda tendrá el aspecto mostrado en la Figura 3, en el que se ha numerado el único nodo existente como el 00.

   El proceso iterativo comienza extrayendo el único nodo en la lista abierta, el 00, y generando todos sus hijos, tal y como se muestra en la Figura 4. Sólo hay dos operadores aplicables en ese momento, el

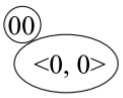

Figura 3: Árbol de búsqueda en el problema de las garrafas

de llenar garrafa grande y el de llenar garrafa peque˜na, generando dos hijos. Ninguno de los hijos es solución, y ninguno de ellos había sido generado anteriormente, por lo que se introducen en abierta.

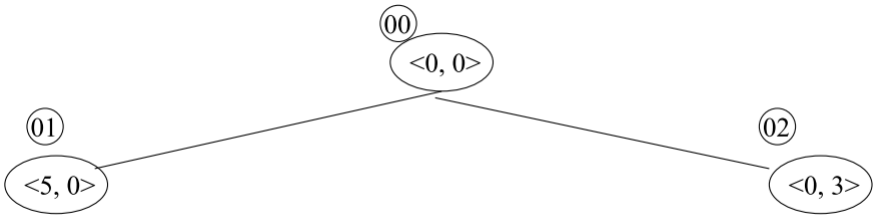

Figura 4: Árbol de búsqueda en el problema de las garrafas

En ese momento, la lista abierta queda como $[01, 02]$, que resulta de haber extraído el nodo 00, y haber incluído los dos descendientes en el orden en que se han generado. Como no se cumple la condición de fin (no se ha encontrado solución, y la lista abierta no está vacía), hay que expandir un nuevo nodo. Para ello, sacamos de abierta el primer nodo en modo FIFO, el 01, y se generan sus descendientes. Este nodo tiene tres descendientes, que se muestran en la Figura 5, que resultan de llenar la garrafa pequeña (nodo 03, estado $\langle 5, 3\rangle$), de vaciar la garrafa grande (nodo 04, estado $\langle 0, 0\rangle$) y de traspasar líquido de la garrafa grande a la peque˜na (nodo 05, estado $\langle 2, 3\rangle$). Se comprueba que el nodo 04 contiene un estado que ya se encuentra en el árbol, pues coincide con el abuelo, el nodo 00. Por tanto, ese nodo no se incluye en abierta, que queda como $[02, 03, 05]$. Nótese, que la introducción de los nodos en la lista se hace por detrás, siguiendo una estructura de cola FIFO.

De nuevo, habría que expandir un nuevo nodo, en este caso el 02, generando el árbol de búsqueda que se muestra en la Figura 6.

Los tres hijos que se generan son los resultantes de ejecutar los operadores de llenar garrafa grande, vaciar la peque˜na, y traspasar líquido de la peque˜na a la grande. Sólo el último, estado $\langle 0, 3\rangle$ en nodo 08, es un nuevo estado, y por tanto, es el único que se introduce en abierta, que queda como $[03, 05, 08]$.

2. Tras expandir otros 7 nodos, el árbol de búsqueda resultante es el mostrado en la Figura 7. En ese momento, abierta incluye sólo 2 nodos, el 27 y el 32.

   Al expandir el nodo 27 (estado $\langle 5, 2\rangle$), se generarían cuatro sucesores, resultantes de llenar la garrafa peque˜na, vaciar la garrafa grande, vaciar la garrafa peque˜na y traspasar líquido de la garrafa grande a la peque˜na, tal y como nos muestra la Figura 8. De estos 4 nodos, todos contienen estados repetidos excepto el nodo 38, que contiene el estado $\langle 4, 3\rangle$, que no sólo no había sido generado anteriormnte, sino que además es nodo meta. Por tanto, el algoritmo de búsqueda en amplitud pararía. La solución encontrada tendría un coste de 6, correspondiente con la profundidad del árbol, puesto que tenemos costes uniformes. En este árbolo de búsqueda habría que destacar el alto número de nodos repetidos que se generan cada vez que se expande un nodo. Esto hace que, aunque el número de operadores aplicable en cada nodo suele ser 3 o 4 operadores, el crecimiento del número de nodos expandidos en la lista abierta no es exponencial, sino que se hace constante en casi todo el árbol.

3. La solución es óptima, ya que la búsqueda en amplitud en problemas de costes uniforme es admisible.

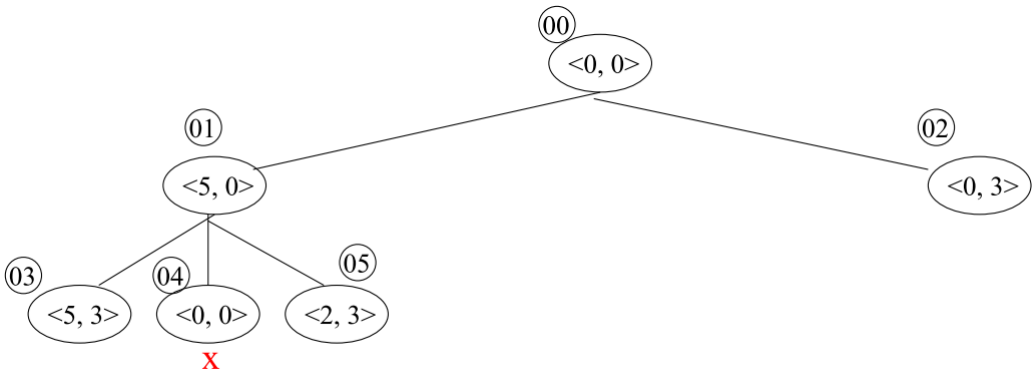

Figura 5: Árbol de búsqueda en el problema de las garrafas

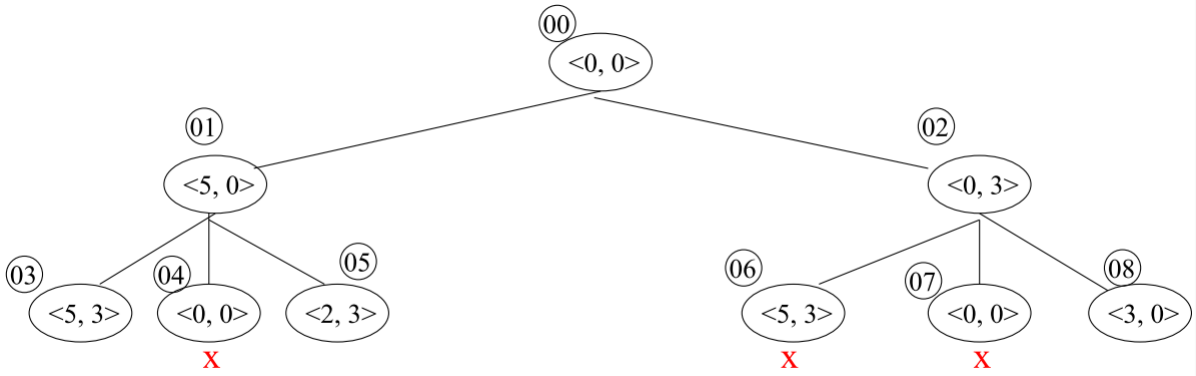

Figura 6: árbol de búsqueda en el problema de las garrafas

Nótese que anteriormente se ha dicho que la solución generada estaba en una profundidad de 6, que se igulaba al coste de la solución. Cualquier otra solución que se pudiera encontrar estaría en ese mismo nivel de profundidad (por tanto, con un mismo coste), o en un nivel de profundidad mayor (con un coste mayor).

4. Con una profundidad máxima de 4, no se puede encontrar una solución, ya que el coste de la solución es 6 y, por tanto, la mínima profundidad a la que se encontraría la solución sería 6.

5. Con una profundidad máxima de 6, sí se encontraría la solución. Sin embargo, el algoritmo de búsqueda en profundidad no es admisible, por lo que no garantizaría que la solución encontrada fuese óptima, a pesar de que en este caso sabemos que sí lo sería.

6. Con una profundidad máxima de 8, también se encontraría una solución, que podría tener un coste de 6, 7 u 8. Nuevamente no se garantizaría que fuese óptima.

7. El algoritmo encontraría una solución en su tercera iteración, con una profundidad máxima de 8. Este algoritmo sólo es admisible cuando la profundidad máxima inicial es 1, y el incremento de profundidad en cada iteración es también 1, por lo que no estaría garantizado que la solución encontrada fuese óptima.

8. Solución del ejercicio 3

1. Puesto que el coste de todos los operadores es indistinguible e igual a la unidad, cualquiera de los algoritmos de búsqueda no informada serviría para encontrar el número óptimo de pasos que transforma cualquier estado en el estado de la figura 1(a). En concreto:

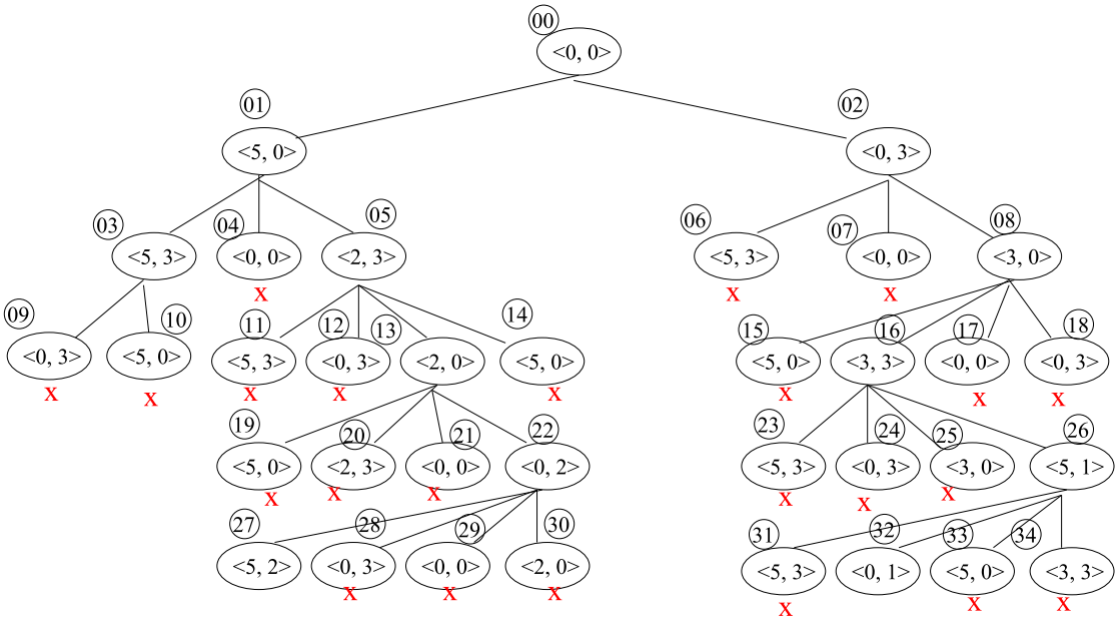

Figura 7: Árbol de búsqueda en el problema de las garrafas

- Puesto que estamos interesados en encontrar siempre soluciones, escogeremos un algoritmo completo
- Además, puesto que queremos calcular el coste óptimo que transforma cualquier configuración en el estado meta de la figura 1(a), el algoritmo elegido deberá ser admisible

Algoritmos que cumplen con estas propiedades serían, o bien amplitud o de profundización iterativa con profundidad inicial igual a 1 e incremento igual también a la unidad. Pero, además:

- Puesto que queremos obtener el valor mínimo que transforma cualquier permutación del 15-Puzle en el estado meta de la figura 1(a), será preciso razonar desde el estado meta hacia atrás.
- En consecuencia, si hay memoria suficiente, se podría usar un algoritmo de búsqueda en amplitud hacia atrás y, si no es así, el mismo problema podría resolverse entonces usando profundización iterativa con un incremento y profundidad inicial iguales a 1, también hacia atrás.

2. En este segundo caso, existen dos tipos de operadores: de una parte, los intercambios del blanco con las casillas 3, 7, 11, 12, 13, 14 y 15, con un coste igual a la unidad; de la otra, el intercambio con casillas rayadas, que no tiene coste asociado.

   Como antes, la mejor estrategia será aplicar un algoritmo de búsqueda hacia atrás para calcular el coste de llegar desde cualquier permutación de los símbolos anteriores hasta el estado de la figura 1(b) pero:

   - El algoritmo del primero en amplitud puede encontrar un nodo n a profundidad d n cuyo coste (suma del coste de cada operador desde la raíz hasta él) sea menor que el de otro nodo m, a una profundidad d_m, menor que la del primer nodo: $d_n > d_m$. El motivo es que ahora puede haber una cantidad arbitraria de movimientos con un coste nulo en cualquier camino desde el nodo raíz.

     Por ejemplo, el nodo n generado con la secuencia $\langle \circ\!-\!\bullet, \circ\!-\!\bullet, \overset{\circ}{\bullet}, \overset{\circ}{\bullet}, \bullet\!-\!\circ, \overset{\bullet}{\circ} \rangle$ (donde el punto blanco representa la posición del blanco y el punto sólido su destino), aparecerá a profundidad 6, mientras que su coste es, sin embargo, igual a 0, puesto que no altera la posición de ninguna casilla numerada. Por otra parte, el nodo m generado a profundidad 3 (menor que 6), con la secuencia $\langle \circ\!-\!\bullet, \circ\!-\!\bullet, \circ\!-\!\bullet \rangle$ tendrá un coste igual a la v, puesto que altera una única vez la posición de una casilla numerada, el 3.

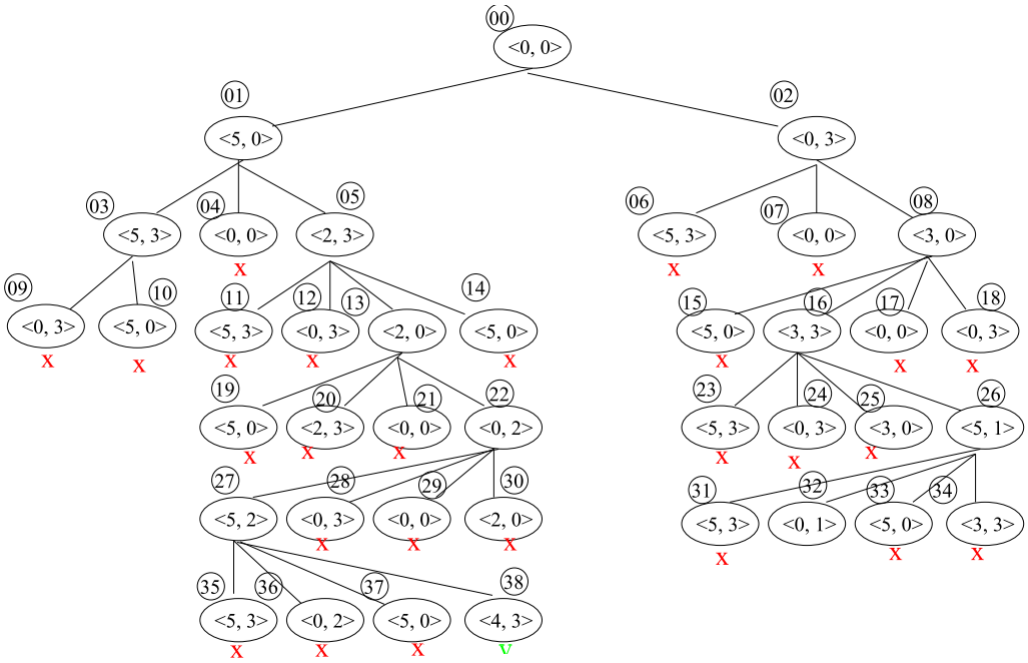

Figura 8: Árbol de búsqueda en el problema de las garrafas

- Algo parecido ocurre con el uso de un algoritmo de profundización iterativa: si se aplica el algoritmo usando como límite la profundidad-máxima, ocurrirá que pueden desestimarse soluciones con costes menores que otras generadas a esta profundidad.

  En definitiva, el problema es que, en cualquier caso, no existe ninguna relación entre la profundidad de un nodo y su coste.

Además, los estados posibles desde los que puede llegarse hasta el estado final de la figura 1(b) son todas las permutaciones de los 7 símbolos citados en cualesquiera de las 16 casillas del 15-Puzle. En total, se trata de: $V_{16,7} = \frac{16!}{(16-7)!} = 16 \times 15 \times 14 \times 13 \times 12 \times 11 \times 10 = 57_1657,600$ posiciones, que caben fácilmente en la memoria de cualquier ordenador, de modo que en lo sucesivo se considerará, únicamente, la adaptación del algoritmo del primero en amplitud para resolver este problema.

Para resolver este caso con el algoritmo del primero en amplitud, una alternativa consistiría en expandir los nodos terminales de la búsqueda generada con este algoritmo según el coste de los nodos terminales, en vez de por su profundidad. Sin embargo:

- Esta modificación no es suficiente para garantizar la optimalidad del algoritmo resultante —o, equivalentemente, su admisibilidad. Además, es preciso modificar la condición de terminación y acabar únicamente cuando se procede a la expansión de un nodo: puesto que los nodos se almacenarían ordenados por el coste de su camino hasta la raíz, cada vez que se procede a la expansión de un nodo es porque se trata del nodo (o uno de los nodos) de menor coste de modo que sólo entonces puede garantizarse que será el que llega con el menor coste (pero no necesariamente el menor número de pasos) hasta el nodo meta.
- La principal dificultad de este modelo consiste en que para mantener la cola convenientemente ordenada, después de cada expansión es preciso re-ordenar la cola abierta [1]

Afortunadamente, existe una solución muy elegante que no tiene el inconveniente se˜nalado. Para evitar la tarea extraordinaria de ordenar y mezclar listas, lo más fácil consiste en enhebrar dos algoritmos: el

[1] Esto se hace, típicamente, ordenando los sucesores del nuevo nodo por su coste y mezclando a continuación esa lista con la cola empleada por esta versión del algoritmo del primero en amplitud

algoritmo de el primero en amplitud que computará el coste mínimo (hacia atrás) para llegar hasta el estado final o meta de la figura 1(b) desde cualquier estado, según se van generando; y un algoritmo del primero en profundidad que recorre todos los caminos con coste nulo a partir de cualquier nodo.

Por lo tanto, aplicando el algoritmo del primero en amplitud hacia atrás, cada vez que se genere un nodo con coste igual a la unidad, el algoritmo procederá con normalidad, anotando que el coste para llegar hasta la raíz del árbol de búsqueda es igual al coste del nodo recién generado, puesto que el algoritmo del primero en amplitud resultante seguirá siendo completo y admisible. Sin embargo, cada vez que se genere un nodo con coste nulo, se invocará la ejecución de un algoritmo del primero en profundidad que recorrerá todos los caminos que haya con operadores de coste nulo a partir de él. El algoritmo del primero en profundidad se detendrá cada vez que se genere un nodo con coste igual a la unidad, de modo que, después de su ejecución, devolverá todos los nodos por debajo del nodo inicial que se generan con un coste estrictamente igual a la unidad.

El algoritmo generado de esta suerte, puede interpretarse como si se hubieran reescrito los operadores: el intercambio con las casillas numeradas se hace con normalidad, con coste igual a la unidad; el intercambio con casillas no numeradas devuelve todos los intercambios con casillas numeradas que pueden ocurrir por debajo de él, también con coste igual a la unidad. Puesto que ahora todos los costes son iguales a la unidad, un algoritmo de el primero en amplitud resolverá el problema planteado de forma admisible.

9. Solución del ejercicio 4

1. Búsqueda en Amplitud

Abierta	Expandir
A	A
D,F,G	D
F,G,J,H	F
G,J,H,C,E	G
J,H,C,E	J
H,C,E,K	H
C,E,K,B	

2. Búsqueda en Profundidad

Abierta	Expandir
A	A
D,F,G	D
J,H,F,G	J
K,H,F,G	K
L,H,F,G	

3. Búsqueda Mejor-primero

Abierta	Expandir
A	A
D,F,G	D
F,G,H,J	F
C,E,G,H,J	C
E,G,H,J	E
G,H,J,W,Z	G
H,J,W,Z	H
B,J,W,Z	

10. Solución del ejercicio 5

El espacio de estados se puede representar con una tupla de 7 posiciones: $< x_1, \ldots, x_7 >$, donde cada posición hace referencia a la posición de un asiento en el coche. x_1 y x_2 son los asientos del piloto y copiloto respectivamente. En la segunda fila de asientos, estarían x_3, x_4 y x_5, que corresponden con la ventanilla izquierda, el asiento central, y la ventanilla derecha respectivamente. Por último, tendríamos x_6 y x_7 en la tercera fila de asientos. Cada elemento de la tupla puede tomar un valor en el conjunto $\{n, a, d\}$, donde n significa posicion normal, a significa abatido, y d significa desplazado. A esta tupla se le podría unir el valor del espacio de carga disponible, aunque dado que es directamente calculable desde la configuración, no es necesario.

Dada esta representación cabría la duda de si incluir o no en cada estado la información sobre la capacidad de carga disponible. Sin embargo, este valor es calculable a partir de cada estado como una función, por lo que es preferible no gastar memoria con esa información, que se sabe es redundante.

Hay tres posibles acciones (donde la i en x_i puede tomar un valor entre 1 y 7):

- $abatir(x_i)$: abatir el asiento x_i
  - Precondiciones: $x_i = n$. Para $i = 2$ se debe cumplir que $x_4 = x_5 = a$.
  - Efectos: $x_i = a$
  - Coste: 2
- $desplazar(x_i)$: desplazar el asiento x_i
  - Precondiciones: $x_i = n$. Para $i = 2$ se debe cumplir que $x_4 = x_5 = a$.
  - Efectos: $x_i = d$
  - Coste: 1
- $normal(x_i)$: devolver a posición normal el asiento x_i
  - Precondiciones: $x_i \neq d$. Para $i = 4$ e $i = 5$ se debe cumplir también que $x_2 = n$
  - Efectos: $x_i = n$
  - Coste: depende de la situación de partida de x_i. Si inicialmente $x_i = d$, entonces el coste es igual a 1. Si inicialmente $x_i = a$, entonces el coste es igual a 2.

Nótese que la acción normal (x_i) se podría haber dividido en dos (una que desabatiese, y otra que eliminara el desplazamiento).

El estado inicial $< n, n, n, n, n, a, a >$.

Por simplificar, se podría decir que el tamaño del espacio de estados es 7^3 (todas las posibles combinaciones de los asientos), pero dado que hay configuraciones que no son aceptables es menor que ese valor. En concreto, hay que contar solo las configuraciones en que se cumpla que $x_2 = n$ y en su defecto, que $x_4 = x_5 = a$. En total, cumplen esa condición 11×5^3 estados. Este valor viene de tener en cuenta que de las 3^3 posibles posiciones que pueden ocupar x_2, x_4 y x_5, no hay problemas en los 9 casos en que $x_2 = n$. De los 18 casos restantes (2 posibles valores de x_2 por las 2^3 combinaciones de x_4 y x_5), no hay problema en 2 de ellos (cuando $x_4 = x_5 = a$). Por tanto, el espacio de estados tiene un tamaño de 11×5^3.