RELACIÓN DE EJERCICIOS TEMA 2

Ejercicio 1: Para los problemas de búsqueda planteados más abajo, contesta a las siguientes cuestiones y razona tus respuestas:

-Propón una representación adecuada de los estados.

-Describe el espacio de búsqueda.

-Detalla las acciones disponibles y sus costes.

-¿Cuál es la condición que debe cumplir un estado para ser final?

-¿Es el problema más adecuado para la búsqueda en árbol o para la búsqueda en grafo?

-¿Cuántos estados finales hay?

-¿Es posible dar un heurístico admisible/consistente para el problema? Si la respuesta es afirmativa, proporciona uno y demuestra su admisibilidad/consistencia.

-¿Qué algoritmo(s) de búsqueda serían adecuados para resolver el problema?

Problema 1.a (dos amigos):

Una persona sale de Gerona y otra de Cádiz. En cada etapa, ambas personas pueden moverse desde la capital de una provincia peninsular de España a la capital de una provincia vecina. La tarea es encontrar una ruta para cada persona de manera que se encuentren en alguna capital lo antes posible. El viaje hacia la siguiente capital lo empiezan ambas personas simultáneamente, y ambas deben llegar a sus destinos antes de empezar con la siguiente etapa. Suponemos que sabemos el tiempo necesario para viajar desde cualquier capital a cualquier otra capital vecina.

Problema 1.b (cruzar el puente):

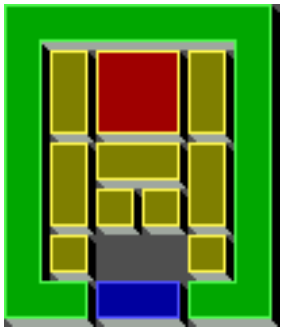
Un grupo de 5 personas quiere cruzar un viejo y estrecho puente por la noche. Es necesario usar una linterna para cruzarlo. Sólo hay una linterna disponible, cuya batería está baja: sólo quedan 5 minutos antes de que se apague. Las personas necesitan 10, 30, 60, 80 y 120 segundos para cruzar el puente, respectivamente. El puente sólo puede sostener a 2 personas como mucho al mismo tiempo, y la velocidad a la que se cruza es la de la persona más lenta. La linterna no se puede lanzar de un lado al otro, así que es necesario que una persona cruce el puente de nuevo para llevarla al otro lado. La tarea es encontrar el modo más rápido de que todo el mundo llegue al lado contrario, si esto es posible.

Problema 1.c (reparto de pan):

Una empresa de panadería tiene una furgoneta para realizar el reparto diario de pan, que puede cargar hasta P panes. Debe servir a N restaurantes, y el i-ésimo restaurante necesita c_i panes diariamente. La cantidad total de panes que repartir excede a P, pero hay panaderías de la empresa repartidas por toda la ciudad, donde la furgoneta puede recargar si es necesario. Supondremos que la furgoneta empieza el recorrido en alguna de las panaderías. La tarea es encontrar la ruta más corta para servir a todos los restaurantes, incluyendo las visitas a las panaderías que sean necesarias. Supondremos conocidas las distancias entre cualquier par de puntos de la ciudad.

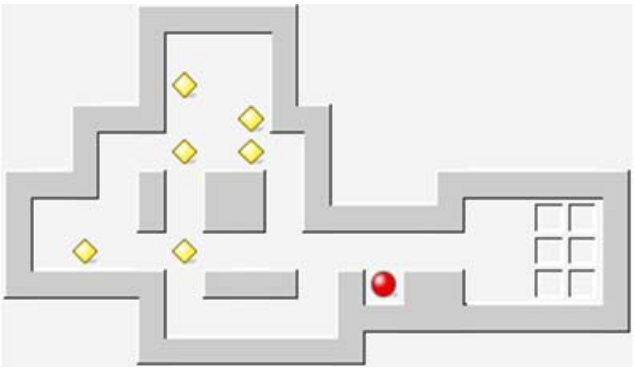
Problema 1.d (Klotski):

Dada la configuración de bloques del diagrama siguiente, el bloque más grande (el rojo) debe moverse hasta la posición central inferior (marcada en azul) con el menor número de movimientos de bloques posible. No está permitido quitar bloques; solamente se pueden deslizar los bloques horizontal y verticalmente.



Problema 1.e (Sokoban):

El agente (señalado con un círculo rojo) debe empujar los diamantes amarillos para llevarlos hasta los almacenes (señalados con recuadros grises huecos). Sólo se puede empujar un diamante cada vez. El agente no puede pasar por encima de los diamantes ni de las paredes. El problema está resuelto cuando todos los diamantes están en almacenes, y buscamos la solución con el menor número de movimientos del agente.



Problema 1.f (Rush hour):

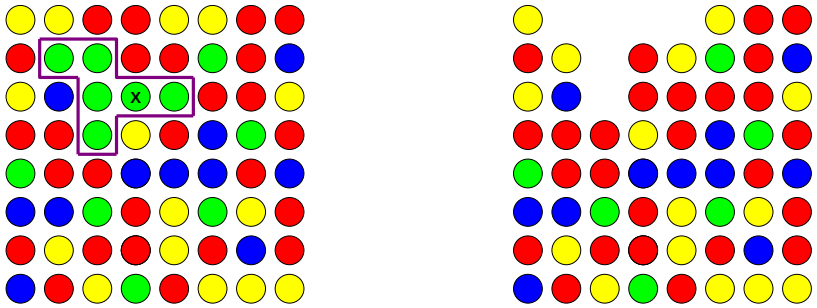
El objetivo es sacar el coche rojo de una rejilla 6x6 llena de automóviles, moviendo los otros vehículos fuera de su camino. Los coches y camiones sólo pueden moverse en línea recta sobre la rejilla (hacia atrás o hacia delante). No se pueden rotar. Buscamos la solución con el menor número de movimientos; puedes suponer que los coches y camiones sólo pueden moverse un recuadro cada vez.

Problema 1.g (juego de burbujas):

Tenemos una matriz con una burbuja en cada celda. Cada burbuja puede ser roja, amarilla, verde o azul. Una burbuja es una 4-vecina de otra burbuja, si la primera está

inmediatamente encima, debajo, a la izquierda o a la derecha de la otra. Un conjunto de burbujas del mismo color es un conjunto 4-conexo si para todo par de burbujas del conjunto existe una secuencia de burbujas 4-vecinas que las conecta.

Si tocamos una burbuja, entonces esa burbuja y todas las de su conjunto 4-conexo explotan, y las burbujas que hubiera encima caen. Como ejemplo, mostramos un posible estado inicial a la izquierda. La burbuja verde que va a ser tocada está señalada con una X, y su conjunto 4-conexo está señalado con una línea púrpura. El estado sucesor se muestra a la derecha; nótese cómo las burbujas que estaban encima de las que han explotado han caído.



La tarea es explotar todas las burbujas con el menor número de toques posible.

Ejercicio 2: Considera un problema de búsqueda en el que todo nodo del árbol de búsqueda tiene exactamente *b* hijos. Supongamos que sólo hay un estado final, que está a profundidad *d* (entendemos que el nodo raíz está a profundidad 0).

a) Calcula, en el mejor y peor caso, el número de nodos que serán analizados antes de encontrar el estado final cuando se usa la búsqueda primero en anchura. Haz lo mismo para la búsqueda primero en profundidad.

b) Calcula, en el mejor y peor caso, el máximo número de nodos que podrían estar esperando para ser expandidos cuando se usa la búsqueda primero en anchura. Haz lo mismo para la búsqueda por profundización progresiva.

Ejercicio 3: Da un ejemplo de un problema de búsqueda en el que la búsqueda en árbol nunca pueda entrar en un bucle infinito. Explica tu respuesta.

POSIBLES SOLUCIONES

Ejercicio 1:

Problema 1.a (dos amigos):

Un estado es un par ordenado (a,b), donde a es la posición actual de la primera persona, y b es la posición actual de la segunda persona. Ambas posiciones deben ser capitales de provincias peninsulares españolas.

El espacio de estados es el conjunto de todos los pares ordenados con la estructura anterior. Como hay 47 provincias peninsulares españolas, esto implica que el espacio de estados tiene tamaño 47^2.

Si el estado actual es (a,b) las acciones disponibles son moverse a otro estado (c,d) tal que a y c son capitales de provincias vecinas, y lo mismo ocurre con b y d. El coste de la etapa es *máx*(*distancia*(a,c), *distancia*(b,d)), dado que la persona que llega antes a su destino debe esperar a la otra.

Un estado es final si a=b, es decir, si ambas personas están en el mismo sitio.

Siempre es posible volver a un estado previamente visitado, con lo cual necesitamos usar búsqueda en grafo para este problema.

Hay 47 estados finales de la forma (a,a), donde a es la capital de una provincia peninsular española.

Un heurístico consistente es $h(a,b)$=*distancia*(a,b)/2, dado que la distancia entre los dos viajeros no puede reducirse en una etapa en más del doble del coste de la etapa:

$$\textit{distancia}(a,b)-\textit{distancia}(c,d) \leq 2\cdot \textit{máx}(\textit{distancia}(a,c), \textit{distancia}(b,d)) \Rightarrow$$
$$\textit{distancia}(a,b)/2 \leq \textit{máx}(\textit{distancia}(a,c), \textit{distancia}(b,d))+\textit{distancia}(c,d)/2 \Rightarrow$$
$$h(a,b) \leq \textit{coste}((a,b),(c,d))+ h(c,d)$$

El algoritmo A* sería particularmente adecuado para este problema, dado el heurístico consistente que acabamos de presentar.

Problema 1.b (cruzar el puente):

Un estado es una terna (A, B, c), donde A es el conjunto de personas en el lado inicial del puente, B es el conjunto de personas en el lado final del puente, y c es la ubicación (inicial/final) de la linterna. Notaremos a cada persona con el número de segundos que necesita para cruzar el puente.

El espacio de estados es el conjunto de todas las ternas con la estructura anteriormente citada, donde los dos conjuntos forman una partición del conjunto {10, 30, 60, 80, 120}, y $c \in$ {inicial, final}.

Si el coste de camino $g(n)$ desde el estado inicial ({10, 30, 60, 80, 120}, ∅, inicial) al estado actual n es mayor o igual que 300 segundos, entonces no hay ninguna acción disponible (porque el tiempo se ha agotado). En otro caso, dado un estado

consideraremos acciones consistentes en transferir una o dos personas desde el lado donde está la linterna hasta el otro lado. El resultado de una acción es un estado en el que la gente que ha cruzado el puente ha sido movida de un conjunto al otro, y la linterna ha cambiado de posición. El coste de una acción es el mayor de los valores de las personas que han cruzado, es decir, el número de segundos que tarda en cruzar la persona más lenta. Sin embargo, si el coste de camino desde el estado inicial hasta el estado resultante es estrictamente mayor que 300 segundos, entonces la acción no está disponible porque no hay tiempo para ejecutarla.

Para saber si un estado es final comprobamos si *B* tiene 5 elementos, es decir, si todas las personas están en el lado final del puente.

Es posible volver a un estado considerado anteriormente haciendo que algunas personas vayan y vuelvan por el puente. Sin embargo, esto consume tiempo, así que en algún momento el conjunto de acciones disponibles se reducirá al conjunto vacío. Así pues, es posible usar la búsqueda en árbol puesto que el árbol de búsqueda es finito, y no demasiado grande. Si queremos usar la búsqueda en grafo, debemos asegurarnos de que los nodos no sean expandidos hasta que se haya encontrado el camino óptimo hasta ellos (usando Dijkstra o A*), ya que un estado puede alcanzarse mediante diferentes caminos, cada uno con su propio coste.

Sólo hay un estado final: (∅, {10, 30, 60, 80, 120}, final). Nótese que no hay manera de que la linterna acabe en el lado inicial del puente si todo el mundo está en el lado final.

Se puede calcular un heurístico consistente ordenando los números del primer conjunto de estados en orden decreciente, y a continuación sumando los números que están en posiciones impares. Esto es consistente porque en un solo paso solo podemos transferir dos personas al lado final. Además, si la linterna está en el lado final y hay todavía gente en el lado inicial, podemos también sumar el número más pequeño del segundo conjunto del estado. Esto sigue siendo consistente porque si la linterna está en el lado final y no hemos llegado todavía al estado final sólo podemos continuar enviando al menos una persona al lado inicial.

Para este problema podemos usar A* o Dijkstra, dado que el espacio de búsqueda es relativamente pequeño. La profundización progresiva también funcionará, ya que es de esperar que haya una solución no demasiado profunda.

Problema 1.c (reparto de pan):

Un estado es un vector (*A*, *B*, *c*, *d*), donde *A* es el conjunto de restaurantes ya visitados, *B* es el conjunto de restaurantes por visitar, *c* es la posición actual de la furgoneta (que puede ser una panadería o un restaurante), y *d* es el número de panes que hay en la furgoneta. El estado inicial es (∅, *Todos-Los-Restaurantes*, *ningún-lugar*, 0), where *ningún-lugar* es una posición especial para indicar que todavía no hemos decidido dónde empezar.

El espacio de estados es el conjunto de todos los vectores que tienen la estructura mencionada más arriba.

Si *c=ningún-lugar* (estado inicial), las acciones disponibles son asignar el valor de *c* a alguna panadería y recargar la furgoneta (poner *d=P*). El coste es cero porque podemos empezar donde queramos. En otro caso, dado un estado las acciones disponibles son:

–Ir a un restaurante $j \in B$ tal que $d \geq c_j$. Entonces ponemos *c=j*; se mueve *j* desde *B* hasta *A*; y se asigna $d = d - c_j$.

–Si la furgoneta no está llena (*d<P*), podemos ir a una panadería *k* y poner *d=P*. En tal caso ponemos *c=k* y *d=P*, y se dejan *A* y *B* sin cambios.

El coste de una acción es la distancia desde la anterior posición de la furgoneta *c* a la posición que vamos a visitar.

Para saber si un estado es final comprobamos si *B* está vacío.

El problema es adecuado para la búsqueda en árbol, ya que no es posible volver a un estado considerado anteriormente. Esto se debe a que los elementos no se pueden mover desde *A* hasta *B*, y no podemos visitar dos panaderías en medio de dos restaurantes.

Todo posible vector de la forma (*A*, ∅, *c*, *d*) es un estado final. En este problema hay al menos *P*+1 estados finales, ya que tenemos *P*+1 valores posibles de *d*.

Un heurístico admisible es la distancia desde la posición actual *c* hasta el restaurante más lejano de *B*. Esto es admisible porque tenemos que llegar a ese restaurante en cualquier caso. Un heurístico mejor es el coste del árbol de unión mínimo del conjunto $B \cup \{c\}$ (véase el ejercicio 3.30 de la tercera edición del libro). Esto es admisible porque el árbol de unión da el coste de la manera más barata de conectar todas estas posiciones, ignorando el hecho de que podríamos tener que visitar algunas panaderías para obtener más panes.

El algoritmo A* sería particularmente adecuado para este problema, dado el heurístico consistente que acabamos de presentar, y el gran tamaño del espacio de búsqueda.

Problema 1.d (Klotski):

Un estado es una matriz 4x4 con la distribución actual de los bloques. Cada bloque se identifica mediante un número, pongamos 1…10. Podemos suponer que el bloque rojo es el número 1. Las celdas vacías contienen ceros.

El espacio de estados es el conjunto de todas las matrices 4x4 tales que todas sus celdas contienen números naturales en el intervalo 0…10, y tales que las formas de los bloques se respetan.

Las acciones disponibles en un cierto estado consisten en mover un bloque una celda hacia la izquierda, hacia la derecha, hacia arriba o hacia abajo, siempre que haya espacio para el movimiento. El coste de cada etapa es siempre 1.

Para ver si un estado es final comprobamos si el bloque rojo está en la parte central inferior, esto es, si las celdas (3,2), (3,3), (4,2) y (4,3) de la matriz valen 1.

Siempre es posible volver a un estado previamente visitado, con lo cual necesitamos búsqueda en grafo para este problema.

Toda matriz del espacio de búsqueda tal que el bloque rojo esté en la parte central inferior es un estado final. Por tanto, tenemos muchos estados finales.

Un heurístico consistente es la distancia Manhattan desde la posición actual de la celda más arriba y a la izquierda con valor 1 hasta la posición (3,2), que es la posición deseada de la parte superior izquierda del bloque rojo. En un paso esta función no puede decrecer en más de uno, que es el coste del paso, así que este heurístico es consistente.

El algoritmo A* sería particularmente adecuado para este problema, dado el heurístico consistente que acabamos de presentar.

Problema 1.e (Sokoban):

Un estado es una matriz del tamaño del mundo donde se mueve el agente, donde cada celda puede tener uno de estos valores: *vacía, diamante, agente*. Las coordenadas de las paredes y de los almacenes para los diamantes nunca cambian, así que no tiene sentido almacenarlas en el estado; las mantendremos en otro lugar.

El espacio de búsqueda es el conjunto de todas las matrices del tamaño del mundo tales que toda celda tiene uno de los valores permitidos, tales que sólo hay una celda marcada como agente, y que el número de diamantes sea el mismo que en el estado inicial.

Las acciones disponibles en un cierto estado consisten en mover el agente una celda a la izquierda, a la derecha, hacia arriba o hacia abajo, suponiendo que haya espacio para el movimiento. Si el movimiento implica empujar un diamante, debe haber una celda vacía detrás del diamante. El coste de paso es siempre 1.

Para ver si un estado es final comprobamos si todos los diamantes están en almacenes.

Hay muchas maneras de volver a un estado previamente visitado, así que necesitamos la búsqueda en grafo para este problema.

Los estados finales tienen todos los diamantes en almacenes, y el agente debe estar adyacente a uno de esos lugares. Por tanto, podemos tener varios estados finales, dependiendo de las coordenadas de los almacenes.

Un heurístico consistente es la suma de las distancias Manhattan desde cada diamante hasta el almacen más cercano. En un paso esta función no puede decrecer en más de uno, que es el coste del paso, así que el heurístico es consistente.

El algoritmo A* sería particularmente adecuado para este problema, dado el heurístico consistente que acabamos de presentar.

Problema 1.f (Rush hour):

Un estado es una matriz 6x6 con la distribución actual de los vehículos. Cada vehículo puede identificarse mediante un número, pongamos 1…*m*. Podemos suponer que el coche rojo es el número 1. Las celdas vacías contendrán ceros.

El espacio de estados es el conjunto de todas las matrices 6x6 tales que todas las celdas contienen números naturales del intervalo 0… *m*, y tales que las formas y orientaciones de los vehículos se respetan.

Las acciones disponibles en un cierto estado son desplazar un vehículo hacia la izquierda, hacia la derecha, hacia arriba o hacia abajo, suponiendo que haya sitio para el movimiento, y que el vehículo no se mueva de lado. El coste de paso es siempre 1.

Para ver si un estado es final comprobamos si el coche rojo está en la salida, es decir, si las celdas (3,5) y (3,6) de la matriz valen 1.

Hay muchas maneras de volver a un estado previamente visitado, así que necesitamos la búsqueda en grafo para este problema.

Toda matriz en el espacio de búsqueda tal que el coche rojo está en la salida es un estado final. Por tanto, tenemos muchos estados finales.

Un heurístico consistente es la distancia Manhattan desde la posición actual de la celda más arriba a la izquierda con valor 1 hasta la posición (3,5), que es la posición deseada de la parte izquierda del coche rojo. En un paso esta función no puede decrecer en más de uno, que es el coste del paso, así que el heurístico es consistente. Nótese que el coche rojo no puede moverse fuera de la fila 3, así que en realidad la distancia Maniatan se reduce a una distancia en columnas.

El algoritmo A* sería particularmente adecuado para este problema, dado el heurístico consistente que acabamos de presentar.

Problema 1.g (juego de burbujas):

Un estado es una matriz 8x8 con la configuración actual de las burbujas. Cada color se codifica con un número: 1…4. Las celdas vacías contendrán ceros.

El espacio de estados es el conjunto de todas las matrices 8x8 tales que todas sus celdas contienen números naturales en el intervalo 0…4.

Las acciones disponibles en un cierto estado consisten en tocar una de las burbujas restantes (codificadas como celdas no nulas en la matriz). Necesitaríamos un algoritmo de coloreado de globos (*blob coloring algorithm*) para hallar el conjunto 4-conexo correspondiente a la burbuja tocada. El resultado de la acción es un estado en el que todas las burbujas del conjunto 4-conexo han sido eliminadas, y las burbujas por encima de ellas han caído. El coste de paso siempre es 1.

Para ver si un estado es final comprobamos si todas las celdas de la matriz son cero.

No hay manera de volver a un estado anterior, dado que las burbujas explotadas no pueden ser recuperadas. Así pues, podemos usar búsqueda en árbol o búsqueda en grafo.

Sólo hay un estado final, que es una matriz 8x8 llena de ceros.

Se puede obtener un heurístico consistente calculando para cada color un vector booleano 8x1 que indique si hay al menos una burbuja de ese color en cada una de las 8 columnas. A continuación contamos el número de secuencias de valores *true* (1) del vector. Por ejemplo, para el vector (*false*,*true*,*true*,*true*,*false*,*false*,*true*,*true*) el recuento es 2 porque hay dos secuencias de valores *true* (hemos señalado la primera secuencia en *rojo* y la segunda secuencia en *verde*). El heurístico consistente se obtiene entonces sumando estos recuentos para los 4 colores. El heurístico es consistente porque en un paso (un toque) sólo podemos eliminar burbujas del mismo color que estén en columnas adyacentes, y además la caída de las burbujas no puede cambiar ninguna burbuja de una columna a otra. Así pues, el heurístico sólo puede permanecer invariable o bien decrecer en uno en cada paso, que es menor o igual que el coste de paso.

El algoritmo A* sería particularmente adecuado para este problema, dado el heurístico consistente que acabamos de presentar.

Ejercicio 2:

2.a)

La búsqueda primero en anchura explorará los niveles 0…d–1 por completo antes de llegar al estado final. Estos niveles contienen $b^0+b^1+\ldots+b^{d-1}$ nodos, y todos ellos deben ser explorados. Hay b^d nodos en el nivel d, y podría ocurrir que sólo necesitáramos analizar el primero de ellos, si tenemos suficiente suerte. Si tenemos mala suerte, podríamos tener que analizar todos los b^d nodos del nivel d. Por consiguiente, el número de nodos que analizará la búsqueda primero en anchura estará entre $b^0+b^1+\ldots+b^{d-1}+1$ nodos (mejor caso) y $b^0+b^1+\ldots+b^d$ nodos (peor caso).

La búsqueda primero en profundidad puede alcanzar la solución a profundidad d analizando únicamente los nodos del camino desde el estado inicial hasta el estado final. Por tanto, en el mejor caso la búsqueda primero en profundidad analiza d+1 nodos para llegar al estado final. En el peor caso, podríamos elegir un camino donde no está el estado final, con lo que el número de nodos explorados sería infinito. Es decir, el estado final nunca sería alcanzado.

2.b)

La búsqueda primero en anchura almacena en el conjunto de nodos que se deben explorar todos los nodos del siguiente nivel mientras está expandiendo los nodos del nivel actual. Así pues, cuando acabamos de expandir el último nodo del nivel d–1, tenemos b^d nodos en el conjunto de nodos que expandir. Este es el mejor caso, puesto que si el estado final es el último nodo del nivel d, entonces tendremos que almacenar en dicho conjunto todos los nodos del nivel d+1 excepto los hijos del estado final. Por tanto, en el peor caso tendríamos $b^{d+1}-b$ nodos en el conjunto de nodos que expandir.

La búsqueda por profundización progresiva sólo necesita almacenar los hijos sin expandir de los nodos del camino que se está examinando en un momento dado. Si la solución está a profundidad d, cuando entramos por primera vez en ese nivel tenemos b–1 nodos por expandir en todos los niveles desde el 1 al d–1. Así, en ese momento el

número de nodos del conjunto de nodos por ser expandidos es $(b-1)\cdot(d-1)$. Como los nodos del nivel d nunca serán expandidos, $(b-1)\cdot(d-1)$ es tanto el mejor como el peor caso para el número máximo de nodos por expandir.

Ejercicio 3:

Los problemas 1.c (reparto de pan) y 1.g (juego de burbujas) son tales que no hay manera de volver a un estado anterior (véase más arriba para más detalles). Por tanto, la búsqueda en árbol nunca puede generar un nodo con el mismo estado que uno de los ascendientes de ese nodo. ¿Puedes dar otros ejemplos?