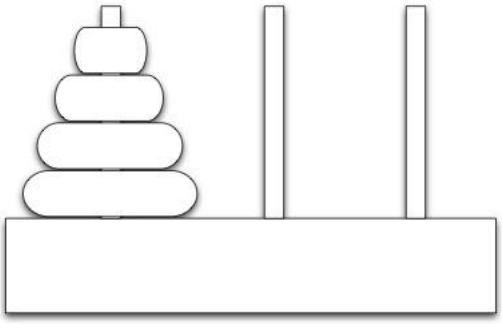
Inteligencia Artificial Problemas de Búsqueda

1. Torres de Hanoi



Las Torres de Hanoi es un famoso problema para estudiar la recursividad en informática y las ecuaciones de recurrencia en matemática discreta. Empezamos con N discos de distintos tamaños en un poste (apilados en orden según su tamaño) y dos postes vacíos. Se nos permite mover un disco de un poste a otro, pero nunca se nos permite mover un disco más grande encima de un disco más pequeño. El objetivo es mover todos los discos hasta el poste situado más a la derecha (véase la figura anterior).

En este problema, formularemos las Torres de Hanoi como un problema de búsqueda.

(a) Proponer una representación de estados para el problema.

Una posible representación del estado sería almacenar tres listas, correspondientes a qué discos están en cada poste. Si suponemos que los N discos están numerados en orden creciente 1, ..., n, entonces podemos representar cada poste como una lista de números enteros correspondientes a los discos que están en esa clavija. El orden en el que estos se encuentran es conocido: los mayores siempre irán debajo.

Otra representación alternativa podría ser asignar a cada disco 1, ..., n un número representando el número de poste en el que se encuentra.

(b) ¿Cuál es el tamaño del espacio de estados?

Si hay k clavijas y n discos, el tamaño del espacio de estados es k^n. Una forma de verlo es que, si hay n discos y cada uno puede estar en k posiciones, habrá k estados posibles para el disco 1, otros k para el disco 2, … hasta el disco n.

Para esta configuración concreta, el tamaño es 3^N.

(c) ¿Cuál es el estado inicial?

Usando la primera representación de estados, ([1, ..., n], [], []).

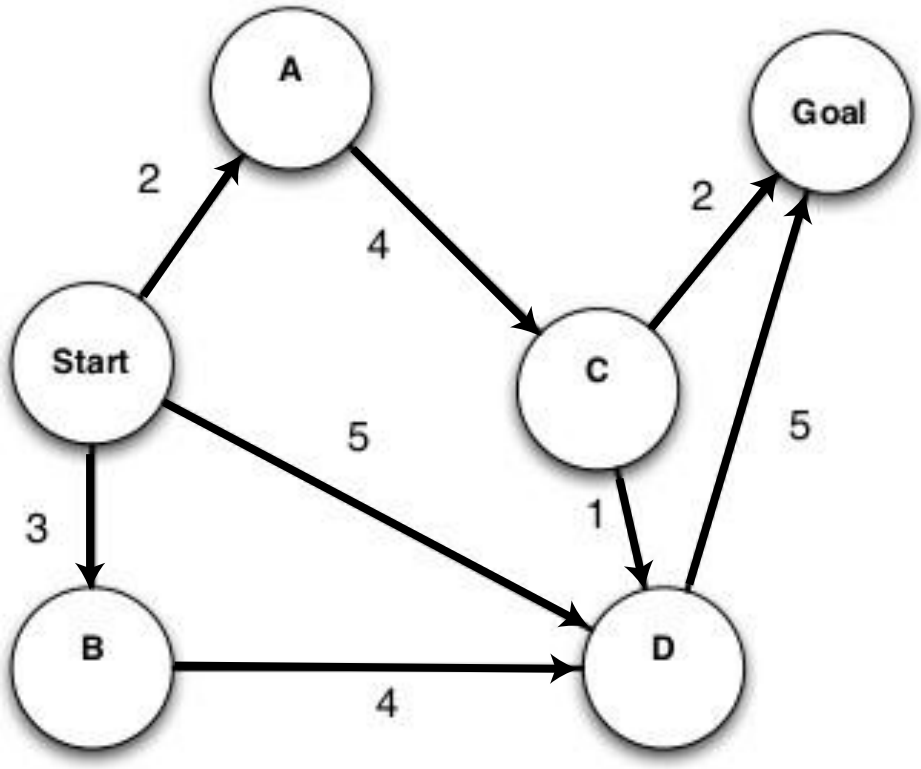
(d) Desde cualquiera de los estados, ¿qué acciones son legales?

Podemos sacar el primer entero de cualquier lista (es decir, poste) y empujarlo al principio de otra lista (clavija), siempre y cuando sea menor que el entero que se encuentra al principio de la lista a la que se empuja (es decir, la clavija a la que se mueve). Los enteros también se pueden mover siempre a cualquier lista vacía.

(e) ¿Cuál es el test objetivo?

¿Estado == ([], [], [1, ..., n])?

2. Algoritmos de búsqueda en acción



Para cada una de las siguientes estrategias de **búsqueda en grafos**, calcule el orden en que se expanden los estados, así como la ruta devuelta por la búsqueda en grafos. En todos los casos, supongamos que los lazos se resuelven de tal manera que los estados con orden alfabético anterior se expanden primero. Recuerde que, en la búsqueda en grafos, un estado se expande solo una vez.

(a) Búsqueda primero en profundidad

En caso de empate, orden alfabético

Inicio – Frontera: S

Se expande S – Frontera: SA, SB, SD – Set explorado: S

Se expande SA – Frontera: SB, SD, SAC – Set explorado: S, A

Se expande SAC – Frontera: SB, SD, SACD, SACG – Set explorado: S, A, C

Se expande SACD – Frontera: SB, SD, SACG, SACDG – Set explorado: S, A, C, D

Se expande SACDG – Objetivo alcanzado

Resultado: S→A→C→D→G

(b) Búsqueda primero en anchura

En caso de empate, orden alfabético

Inicio – Frontera: S

Se expande S – Frontera: SA, SB, SD – Set explorado: S

Se expande SA – Frontera: SB, SD, SAC – Set explorado: S, A

Se expande SB – Frontera: SD, SAC, SBD – Set explorado: S, A, B

Se expande SD – Frontera: SAC, SBD, SDG – Set explorado: S, A, B, D

Se expande SAC – Frontera: SBD, SDG, SACD, SACG – Set explorado: S, A, B, C, D

Si estuviéramos haciendo búsqueda en el árbol, se expandiría SBD, pero en la búsqueda en el grafo, comprobamos que D ya se ha expandido antes.

Se expande SDG – Objetivo alcanzado

Resultado: S→D→G

(c) Búsqueda de coste uniforme

En caso de empate, orden alfabético. Coste del camino entre paréntesis.

Inicio – Frontera: S

Se expande S – Frontera: SA (2), SB (3), SD (5) – Set explorado: S

Se expande SA – Frontera: SB (3), SD (5), SAC (6) – Set explorado: S, A

Se expande SB – Frontera: SD (5), SAC (6), SBD (7) – Set explorado: S, A, B

Se expande SD – Frontera: SAC (6), SBD (7), SDG (10) – Set explorado: S, A, B, D

Se expande SAC – Frontera: SBD (7), SDG (10), SACD (7), SACG (8) – Set explorado: S, A, B, C, D

Si estuviéramos haciendo búsqueda en el árbol, se expandiría SBD o SACD, pero en la búsqueda en el grafo, comprobamos que D ya se ha expandido antes.

Se expande SACG – Objetivo alcanzado

Resultado: S→A→C→G (coste 8)

3. El lobo, la oveja y la col

Considere el siguiente problema:

> *Un hombre se encuentra en la orilla izquierda de un río junto con un lobo, una oveja y una col. Quiere cruzar el río llevando consigo el lobo, la oveja y la col. En la barca sólo hay dos plazas, una de las cuales debe ir ocupada por el hombre. Cada uno de los restantes pasajeros (lobo, oveja, col) ocupa una plaza, de tal modo que sólo uno puede acompañar al hombre en cada viaje. Además, no puede dejar solos en una orilla al lobo con la oveja, ni a la oveja con la col (ni los tres), porque la primera se comería a la segunda en cada paso*

a) Proponga una representación de estados para el problema, e indique los estados inicial y objetivo

Una posibilidad: cada estado se describe como un par de conjuntos, indicando quién(es) se encuentran en cada orilla del río (c = col; o = oveja; l = lobo; h = hombre). Por tanto, el estado inicial del problema sería ({c,o,l,h},{}), y el estado objetivo ({},{c,o,l,h}).

b) Simule la estrategia de búsqueda en anchura para este problema, asumiendo que se filtran todos los estados repetidos. Expanda los sucesores de los nodos siguiendo las preferencias del hombre, las cuales se ordenan (de mayor a menor) como sigue: viajar con la oveja, viajar con la col, viajar sólo, viajar con el lobo. Dibuje el árbol de búsqueda correspondiente e indique el orden en el que se exploran los nodos.

En caso de empate, orden alfabético. Por simplicidad, se representan los planes por su estado final.

Inicio – Frontera: ({c,o,l,h},{})

Se expande ({c,o,l,h},{}) – Frontera: ({c,l},{o,h}) – Set explorado: ({c,o,l,h},{})

Se expande ({c,l},{o,h}) – Frontera: ({c,o,l,h},{}), ({c,l,h},{o}) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h})

Filtramos el estado ya explorado

Se expande ({c,l,h},{o}) – Frontera: ({l},{c,o,h}), ({c, l},{o,h}), ({c},{o,l,h}) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o})

Prefiere viajar con la col (expandimos ese nodo). Añadimos entre paréntesis la profundidad.

Se expande ({l},{c,o,h}) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({o,l,h},{c})(5), ({c,l,h},{o})(5) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h})

Hasta aquí es igual para BPA y BPP.

En BPA:

Se filtra ({c, l},{o,h})(4)

Se expande ({c},{o,l,h})(4) – Frontera: ({o,l,h},{c})(5), ({c,l,h},{o})(5), ({c,o,h},{l})(5), ({c,l,h},{o})(5) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({c},{o,l,h})

Se expande ({o,l,h},{c})(5)) – Frontera: ({c,l,h},{o})(5), ({c,o,h},{l})(5), ({c,l,h},{o})(5), ({l},{c,o,h})(6), ({o},{c,l,h})(6) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({c},{o,l,h}), ({o,l,h},{c})

Se filtra y luego se expande ({c,o,h},{l})(5)) – Frontera: ({c,l,h},{o})(5), ({l},{c,o,h})(6), ({o},{c,l,h})(6), ({o},{c,l,h})(6), ({c},{o,l,h})(6), ({o},{c,l,h})(6) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({c},{o,l,h}), ({o,l,h},{c}), ({c,o,h},{l})

Se filtra y luego se expande ({o},{c,l,h})(6) – Frontera: ({o},{c,l,h})(6), ({c},{o,l,h})(6), ({o},{c,l,h})(6), {c,o,h},{l})(7), {o,h},{c,l})(7), {o,l,h},{c})(7) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({c},{o,l,h}), ({o,l,h},{c}), ({c,o,h},{l}), ({o},{c,l,h})

Se filtra y luego se expande ({o,h},{c,l})(7) – Frontera: ({o,l,h},{c})(7), ({},{c,o,l,h})(8), ({o},{h,c,l})(8) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({c},{o,l,h}), ({o,l,h},{c}), ({c,o,h},{l}), ({o},{c,l,h})

Se filtra y luego se expande ({},{c,o,l,h})(8) – Objetivo alcanzado.

Camino: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o},{c,l,h}), ({o,h},{c,l}), ({},{c,o,l,h}) (7 pasos)

c) Simule ahora la estrategia de búsqueda en profundidad para este problema. Dibuje el árbol de búsqueda correspondiente e indique el orden en el que se exploran los nodos, asumiendo que se filtran todos los estados repetidos y siguiendo las preferencias del apartado anterior.

En BPP:

Se expande ({o,l,h},{c})(5) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({c,l,h},{o})(5), ({l},{c,o,h})(6), ({o},{c,l,h})(6) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o,l,h},{c})

Se filtra y luego se expande ({o},{c,l,h})(6) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({c,l,h},{o})(5), ({c,o,h},{l})(7), ({o,h},{c,l})(7), ({o,l,h},{c})(7) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o,l,h},{c}), ({o},{c,l,h})

Se expande ({c,o,h},{l})(7) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({c,l,h},{o})(5), ({o,h},{c,l})(7), ({o,l,h},{c})(7), ({c},{o,l,h})(8), ({o},{c,l,h})(8) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o,l,h},{c}), ({o},{c,l,h}), ({c,o,h},{l})

Se expande ({c},{o,l,h})(8) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({c,l,h},{o})(5), ({o,h},{c,l})(7), ({o,l,h},{c})(7), ({o},{c,l,h})(8), ({c,o,h},{l})(9), ({c,l,h},{o})(9) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o,l,h},{c}), ({o},{c,l,h}), ({c,o,h},{l}), ({c},{o,l,h})

Se filtra y luego se expande ({o},{c,l,h})(8) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({c,l,h},{o})(5), ({o,h},{c,l})(7), ({o,l,h},{c})(7), ({c,o,h},{l})(9), ({o,h},{c,l})(9) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o,l,h},{c}), ({o},{c,l,h}), ({c,o,h},{l}), ({c},{o,l,h}), ({o},{c,l,h})

Se filtra y luego se expande ({o,h},{c,l})(9) – Frontera: ({c, l},{o,h})(4), ({c},{o,l,h})(4), ({c,l,h},{o})(5), ({o,h},{c,l})(7), ({o,l,h},{c})(7), ({},{c,o,l,h})(10), ({o},{c,l,h})(10) – Set explorado: ({c,o,l,h},{}), ({c,l},{o,h}), ({c,l,h},{o}), ({l},{c,o,h}), ({o,l,h},{c}), ({o,l,h},{c}), ({o},{c,l,h}), ({c,o,h},{l}), ({c},{o,l,h}), ({o},{c,l,h}) , ({o,h},{c,l})

Se expande ({},{c,o,l,h})(10) – Objetivo alcanzado.

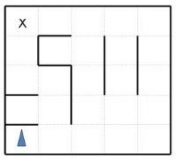
Resultado: ({c,o,l,h},{}) → ({c,l},{o,h}) → ({c,l,h},{o}) → ({l},{c,o,h}) → ({o,l,h},{c}) → ({o},{c,l,h}) → ({c,o,h},{l}) → ({o},{c,l,h}) → ({o,h},{c,l}) → ({},{c,o,l,h}) (9 pasos)

d) Suponga ahora que no se filtra ningún estado repetido. ¿La búsqueda en anchura y la búsqueda en profundidad seguirían siendo completas? Razone brevemente su respuesta

La BPA sí, pero la BPP no porque aparecen caminos en bucle.

4. Búsqueda e heurísticas

Imagine que un agente similar a un automóvil desea salir de un laberinto como el que se muestra a continuación:



1. El agente es direccional y en todo momento mira hacia alguna dirección $d \in (N,S,E,O)$.
2. Con una sola acción, el agente puede avanzar a una velocidad ajustable *v o* girar. Las acciones de giro son *izquierda* y *derecha*, que cambian la dirección del agente en 90 grados. El giro solo está permitido cuando la velocidad es cero (y la deja en cero).
3. Las acciones de movimiento son *rápido* y *lento*. *Rápido* incrementa la velocidad en 1 y *lento* disminuye la velocidad en 1; en ambos casos, el agente mueve un número de cuadrados igual a su NUEVA velocidad ajustada (ver ejemplo a continuación).
4. Una consecuencia de esta formulación es que es **imposible** que el agente se mueva a la misma velocidad distinta de cero durante dos pasos de tiempo consecutivos.
5. Cualquier acción que resulte en una colisión con una pared o reduzca *v por* debajo de 0 / por encima de una velocidad máxima $V_{máx}$ es ilegal.
6. El objetivo del agente es encontrar un plan que lo aparque (estacionario) en la plaza de salida utilizando la menor cantidad posible de acciones (pasos de tiempo).

Como ejemplo: si en el paso de tiempo t la velocidad actual del agente es 2, al tomar la acción *rápido*, el agente primero aumenta la velocidad a 3 y se mueve 3 cuadrados hacia adelante, de modo que en el paso de tiempo t + 1 la velocidad actual del agente será 3 y estará a 3 casillas de distancia de donde estaba en el paso de tiempo t. Si, en cambio, el agente toma la acción *lento*, primero disminuye su velocidad a 1 y luego mueve 1 cuadrado hacia adelante, de modo que en el paso de tiempo t + 1 la velocidad actual del agente será 1 y estará a 1 casilla de distancia de donde estaba en el paso de tiempo t. Si, con una velocidad instantánea de 1 en el paso de tiempo t + 1, toma la acción *lento* nuevamente, la velocidad actual del agente se convertirá en 0 y no se moverá en el paso de tiempo t + 1. Luego, en el paso de tiempo t + 2, será libre de girar si lo desea. Tenga en cuenta que el agente no podría haber girado en el paso de tiempo t + 1 cuando tenía una velocidad de 1, porque tiene que estar estacionario para girar.

1. Si la cuadrícula *es M* por *N*, ¿cuál es el tamaño del espacio de estados? Justifique su respuesta. Debe suponer que todas las configuraciones son accesibles desde el estado inicial.

La representación de los estados es (dirección, x, y, velocidad).

El tamaño del espacio de estados es, por tanto, $4MN(V_{max}+1)$ (direcciones × ancho del laberinto × alto del laberinto × velocidades posibles)

Nótese que la velocidad puede tomar cualquier valor en $\{0, ..., V_{max}\}$.

2. ¿Es admisible la distancia de Manhattan desde la ubicación del agente hasta la ubicación de la salida? ¿Por qué o por qué no?

No, la distancia Manhattan no es una heurística admisible. El agente puede moverse a una velocidad media superior a que 1 (primero acelerando hasta V_{max} y luego frenando hasta 0 al llegar a la meta), por lo que puede alcanzar la meta en menos pasos de tiempo de los que hay cuadrados entre él y la meta. Un ejemplo concreto: En un paso de tiempo 0, el agente comienza inmóvil en la casilla 0 y el objetivo se encuentra a 9 casillas de distancia, en la casilla 9. En el paso de tiempo 0 el agente realiza la acción "rápido" y termina en la casilla 1 con una velocidad de 1. En el paso de tiempo 1, el agente realiza la acción "rápido" y acaba en la casilla 3 con una velocidad de 2. En el paso de tiempo 2, el agente realiza la acción "rápido" y termina en la casilla 6 con una velocidad de 3. En el paso de tiempo 3, el agente realiza la acción "lento" y llega a la casilla 8 con una velocidad de 2. En el paso de tiempo 4, el agente realiza la acción "lento" y llega al cuadrado 9 con una velocidad de 1. En el paso de tiempo 5, el agente realiza la acción "lento" y se queda en el cuadrado 9 con una velocidad de 0. Por lo tanto, la acción lenta no se realiza. Por lo tanto, el agente puede moverse 9 casillas realizando 6 acciones.

3. Declarar y justificar una heurística admisible no trivial para este problema que no es la distancia de Manhattan a la salida.

Hay muchas respuestas a esta pregunta. He aquí algunas, en orden de más débil a más fuerte:

(a) El número de giros necesarios para que el agente quede orientado hacia la meta.

(b) Considere una relajación del problema donde no hay paredes, el agente puede girar, cambiar de velocidad arbitrariamente y mantener la velocidad constante. En este problema relajado, el agente se movería con V_{max}, y luego se detendría de repente en la meta, tardando así $d_{Manhattan}/V_{max}$ unidades de tiempo.

Para pensar (no es el nivel que se pedirá en el examen):

También podemos mejorar la relajación anterior teniendo en cuenta la dinámica de aceleración y desaceleración. En este caso, el agente tendrá que acelerar desde 0 desde el estado inicial, mantener una velocidad constante de V_{max}, y desacelerar hasta 0 cuando esté a punto de alcanzar la meta. Nótese que esta heurística siempre siempre devolverá un valor mayor que el anterior, pero sigue sin sobreestimar el coste real de alcanzar la meta. Podemos decir que esta heurística domina a la anterior.

En concreto, supongamos que $d_{Manhattan}$ es mayor e igual que la distancia que se tarda en acelerar hasta y desacelerar desde V_{max} (En el caso de que $d_{Manhattan}$ sea menor que esta distancia, podemos seguir usando $d_{Manhattan}/V_{max}$ como heurístico). Podemos dividir $d_{Manhattan}$ en tres partes: d_{acel}, d_{Vmax}, d_{decel}. El agente viaja una distancia de d_{acel} cuando acelera de 0 a V_{max} y una distancia de d_{decel} cuando desacelera de

V_{max} a 0. Entre la aceleración y la desaceleración, el agente viaja una distancia de $d_{Vmax} = d_{Manhattan} - d_{acel} - d_{decel}$. Para las dos primeras, tenemos sumas aritméticas:

$$d_{acel} = 1 + 2 + 3 + \ldots + V_{max} = \frac{V_{max}(V_{max} + 1)}{2}$$

$$d_{decel} = (V_{max} - 1) + (V_{max} - 2) + \cdots + 1 + 0 = \frac{V_{max}(V_{max} - 1)}{2}$$

Donde se ha usado la fórmula general para la suma de los primeros n números naturales:

$$s_n = \frac{n}{2}(término_{primero} + término_{último})$$

Y la fórmula general para la suma de los primeros n números naturales en orden descendente desde un valor V_{max} hasta 0:

$$s_n = \frac{n}{2}(2 \cdot término_{primero} - (n - 1))$$

Con esto, tenemos:

$$d_{Vmax} = d_{Manhattan} - \frac{V_{max}(V_{max} + 1)}{2} - \frac{V_{max}(V_{max} - 1)}{2} = d_{Manhattan} - V_{max}^2$$

Se requieren V_{max} pasos para viajar d_{acel}, $\frac{d_{Manhattan} - V_{max}^2}{V_{max}}$ para viajar los d_{Vmax} pasos, y V_{max} pasos para viajar d_{decel}. Por tanto, la heurística es:

$$h(n) = \begin{cases} \frac{d_{Manhattan}}{V_{max}}, & si\ d_{Manhattan} \leq V_{max}^2 = d_{acel} + d_{decel} \\ \frac{d_{Manhattan}}{V_{max}} + V_{max}, & si\ d_{Manhattan} > V_{max}^2 = d_{acel} + d_{decel} \end{cases}$$

4. Si usáramos una heurística inadmisible en la búsqueda en el grafo A*, ¿sería completa? ¿Sería óptima?

Si la función heurística está acotada, entonces la búsqueda en grafos A* visitaría todos los nodos eventualmente, y encontraría un camino hacia el estado objetivo si existe.

Una heurística inadmisible no garantiza la optimalidad, ya que puede hacer que el buen objetivo óptimo parezca estar muy lejos, y llevarte a un objetivo subóptimo.

5. Si utilizamos una heurística *admisible* en la búsqueda en el grafo A*, ¿se garantiza que devolverá una solución óptima? ¿Y si la heurística fuera consistente? ¿Qué pasaría si estuviéramos usando la búsqueda de árbol A* en lugar de la búsqueda en el grafo A*?

Aunque las heurísticas admisibles garantizan la optimalidad para la búsqueda en el árbol A*, no garantizan necesariamente la optimalidad para la búsqueda en el grafo A*; sólo se garantiza que devuelvan una solución óptima si también son consistentes.

En la búsqueda en el árbol, si la heurística es admisible, la solución será óptima.

6. Dar una ventaja general que una heurística inadmisible podría tener sobre una admisible.

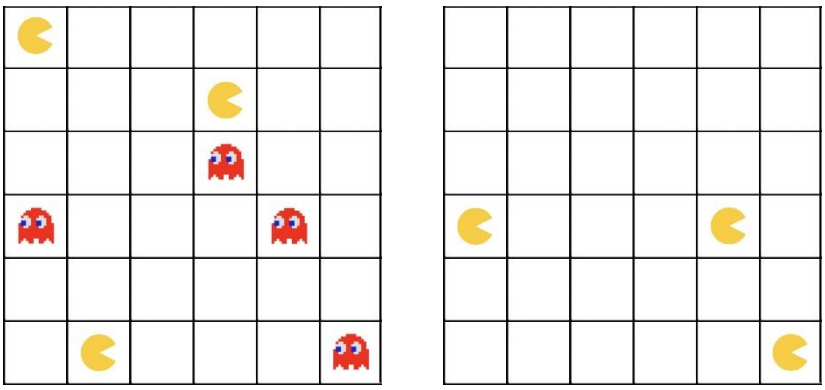
El tiempo para resolver un problema de búsqueda A* es función de dos factores: el número de nodos expandidos y el tiempo empleado por nodo. Una heurística inadmisible puede ser más rápida de calcular, lo que conduce a una solución que se obtiene más rápidamente debido al menor tiempo empleado por nodo. También puede ser una estimación más cercana a la función de coste real (¡aunque a veces la sobreestime!), con lo que se expanden menos nodos. Perdemos la garantía de optimalidad al utilizar una heurística inadmisible. Pero a veces podemos estar de acuerdo con encontrar una solución subóptima a un problema de búsqueda

5. Pacamigos unidos

Pacman y sus Pacamigos han decidido combinar fuerzas e ir a la ofensiva, ¡y ahora están persiguiendo fantasmas! En una cuadrícula de tamaño M por N, Pacman y $P - 1$ de sus Pacamigos se están moviendo para eliminar colectivamente **a todos los** fantasmas en la cuadrícula pisando la misma casilla que cada uno de ellos. Pasar a la misma casilla que un fantasma lo eliminará de la cuadrícula y moverá al Pacman a esa casilla.

En cada turno, Pacman y sus Pacamigos pueden elegir una de las siguientes cuatro acciones: *izquierda, derecha, arriba, abajo,* pero no pueden chocar entre sí. En otras palabras, cualquier acción que resulte en dos o más Pacmen ocupando la misma casilla resultará en ningún movimiento ni para Pacman ni para los Pacamigos. Además, Pacman y sus Pacamigos son **indistinguibles** entre sí. Hay un total de G fantasmas, que son indistinguibles entre sí y no pueden moverse.

Tratando esto como un problema de búsqueda, consideramos que cada configuración de la cuadrícula es un estado, y el estado objetivo es la configuración donde **todos los** fantasmas han sido eliminados del tablero. A continuación, se muestra un ejemplo de estado inicial, así como un ejemplo de estado de objetivo:



(a) Posible estado inicial (b) Posible estado objetivo

Suponga que cada una de las siguientes subpartes es **independiente** entre sí. **También suponga que, independientemente de cuántos Pacmans se muevan en un turno, el coste total del movimiento sigue siendo 1.**

(a) Supongamos que Pacman no tiene Pacamigos, por lo que $P = 1$.

(i) ¿Cuál es el tamaño de la representación mínima del espacio de estados dada esta condición?

Dado que P = 1, sólo necesitamos llevar la cuenta de la posición de Pacman, así como de si cada uno de los G fantasmas ha sido eliminado. Podemos hacer un seguimiento de esto con MN y 2^G respectivamente; por lo tanto, el tamaño de la representación mínima es el producto de los dos, que es $MN(2)^G$. Tened en cuenta que no necesitamos hacer un seguimiento de las posiciones de los fantasmas, ya que son estacionarios.

Para cada una de las siguientes heurísticas, seleccione si la heurística es sólo admisible, sólo consistente, ninguna o ambas. Recordemos que $P = 1$.

Es importante señalar que, si es consistente, tiene que ser admisible, por lo que no puede ser sólo consistente.

(ii) $h(n) =$ la suma de las distancias de Manhattan de Pacman a cada fantasma.

En este caso, podemos considerar un ejemplo ficticio, tal que todos los fantasmas estén adyacentes y alineados horizontalmente, y Pacman esté una casilla a su izquierda. Si hay tres fantasmas, entonces la suma de sus distancias Manhattan desde Pacman es 1 + 2 + 3 = 6, que es una sobreestimación del coste real 3, ya que el Pacman sólo necesita moverse tres espacios para eliminar realmente a todos los fantasmas. Por tanto, ni admisible ni consistente.

(iii) $h(n) =$ el número de fantasmas multiplicado por la distancia máxima de Manhattan entre Pacman y cualquiera de los fantasmas.

Considerando el mismo ejemplo que en la parte anterior, esta heurística también sobreestima el coste, ya que el número de fantasmas × distancia Manhattan entre Pacman y el fantasma más alejado es 3×3 = 9 > 3. Esta pregunta también podría ser "distancia Manhattan mínima", pero la respuesta sigue siendo ninguna. Consideremos el caso en el que Pacman está muy lejos del fantasma más cercano, pero todos los fantasmas están uno al lado del otro. La heurística nos dará una sobreestimación para recoger todos los fantasmas, porque ir al valor de la distancia mínima a cada fantasma es una sobreestimación una vez que has llegado al fantasma más cercano. Por tanto, ni admisible ni consistente.

(iv) $h(n) =$ el número de fantasmas restantes.

Sin sus Pacamigos, Pacman sólo puede eliminar como máximo un fantasma en cada turno, por lo que el número de fantasmas será un límite inferior al número de movimientos necesarios para eliminar todos los fantasmas, lo que hace que esta heurística sea admisible. Además, como entre un movimiento y otro, el número de fantasmas sólo puede reducirse como máximo en uno, la heurística también es consistente.

Explicación más detallada de la consistencia:

Para demostrar que la heurística h(n) = el número de fantasmas restante es consistente, debemos verificar que cumple con la propiedad de la desigualdad de triángulo, que es una propiedad esencial para que una heurística sea consistente.

La desigualdad de triángulo establece lo siguiente para cualquier estado n, estado sucesor n', y coste c(n, n') de llegar de n a n':

$h(n) - h(n') \leq c(n, n')$

En este caso, h(n) representa el número de fantasmas restantes en el estado n, y c(n, n') es el coste de moverse de n a n', que es 1 en este problema.

Para demostrar la consistencia de h(n) = el número de fantasmas restantes, observemos lo siguiente:

1. h(n) es siempre no negativo porque el número de fantasmas restantes nunca puede ser negativo.
2. Cuando Pacman se mueve a un estado sucesor n' desde el estado n, puede eliminar a uno o más fantasmas, pero no puede aumentar el número de fantasmas restantes en n' en comparación con n. Esto se debe a que no hay nuevos fantasmas que aparezcan durante el movimiento de Pacman, y Pacman solo puede eliminar fantasmas. Por lo tanto, h(n) - h(n') será 0 o positivo.
3. Dado que el coste c(n, n') de moverse de n a n' es 1, h(n) - h(n') ≤ c(n, n') siempre se cumple.

En resumen, h(n) = el número de fantasmas restantes cumple con la desigualdad de triángulo y, por lo tanto, es una heurística consistente para este problema de búsqueda en el que Pacman persigue fantasmas.

(b) Supongamos que Pacman tiene exactamente un Pacamigo menos que el número de fantasmas; por lo tanto, $P = G$. Recordemos que Pacman y sus Pacamigos son indistinguibles entre sí.

(i) (Opcional) ¿Cuál es el tamaño de la representación mínima del espacio de estado dada esta condición? Recordemos que $P = G$.

Para pensar (los números combinatorios quedan fuera del alcance de la asignatura):

Como P puede ser cualquier número de Pacman y Pacamigos, arbitrariamente igual a G, necesitamos llevar la cuenta de la posición de todos los Pacmans, así como de si cada uno de los G fantasmas ha sido eliminado; sin embargo, como los Pacmans son indistinguibles entre sí, podemos considerar iguales los estados en los que diferentes Pacmans han intercambiado sus posiciones. Por lo tanto, podemos hacer un seguimiento de las configuraciones únicas de los Pacmans utilizando $\binom{MN}{P}$ que es el número de maneras de seleccionar P posiciones Pacman de una cuadrícula de M por N. Por lo tanto, el tamaño de la representación mínima es el producto de esto y 2^G, que es $\binom{MN}{P}2^G$

Para cada una de las siguientes heurísticas, seleccione si la heurística es sólo admisible, sólo coherente, ninguna o ambas. Recordemos que $P = G$.

(ii) $h(n) =$ la mayor de las distancias de Manhattan entre cada Pacman y su fantasma más cercano.

Consideremos el caso en el que tenemos dos Pacmans y dos fantasmas, y un Pacman está muy cerca de ambos fantasmas, mientras que el otro está muy lejos. Esta heurística sobreestimaría el coste ya que la distancia entre el Pacman más alejado y los fantasmas como métrica, cuando el Pacman más cercano podría eliminar ambos fantasmas más rápidamente.

(iii) $h(n) =$ la más pequeña de las distancias de Manhattan entre cada Pacman y su fantasma más cercano.

Como mínimo, se necesitará la menor de las distancias entre cada Pacman y su fantasma más cercano para que cualquiera de los fantasmas sea eliminado. Por tanto, esta heurística siempre subestimará el coste real y, por tanto, es admisible. Para ver que esto es consistente, podemos decir que la diferencia entre h(a) - h(b) para ir

de cualquier estado al siguiente es una subestimación del coste entre estados. De hecho, consideremos el escenario con dos Pacmans y dos fantasmas, donde un fantasma está al lado de un Pacman y lejos del otro, mientras que el otro fantasma está lejos de ambos. La heurística para este estado será 1, ya que uno de los fantasmas está adyacente a uno de los comecocos, pero en cuanto ese comecocos elimine a ese fantasma, el estado siguiente tendrá un valor heurístico mucho mayor, ya que la distancia Manhattan mínima entre cualquiera de los comecocos y el fantasma restante será muy alta; incluso en este caso límite, h(A) - h(B) será negativo, lo que significa que sigue siendo una subestimación del coste entre estados. Por lo tanto, la heurística es consistente.

(iv) $h(n) =$ número de fantasmas restantes

Dado que múltiples fantasmas podrían ser eliminados en un movimiento porque hay más de un Pacman eliminándolos, esta heurística puede sobreestimar el coste para el estado objetivo, y por lo tanto no es consistente ni admisible.

(v) $h(n) =$ número de fantasmas restantes / P

Como se nos da que P = G, el valor máximo de esta heurística es 1, ya que h(n) es como máximo G. Por tanto, esta heurística es admisible ya que tiene un valor siempre menor o igual que 1, el coste de dar una vuelta, y cuando no quedan fantasmas, la heurística se evalúa a 0, por lo que subestimará el coste hasta el estado objetivo. También es consistente, ya que el valor de la heurística disminuye fraccionalmente en incrementos inferiores a 1, por lo que siempre subestima el coste entre dos estados.

6. Problema de los bloques

Considere el problema de los bloques cuyo estado inicial y estado objetivo se muestran en la siguiente figura:

Las acciones posibles son:

- Situar un bloque libre (sin nada encima) en la mesa
- Situar un bloque libre (sin nada encima) sobre otro bloque libre

Desarrolle el árbol de búsqueda que expande el algoritmo A^*, utilizando la siguiente heurística:

$$h(n) = \textit{número de bloques descolocados}$$

Con tal fin, considere que un bloque está descolocado si *por debajo* no tiene el elemento correcto (bien el bloque deseado o bien la mesa). Utilizando la búsqueda en el grafo, indique el orden de expansión de los estados y muestre en cada paso los valores de *f*, *g* y *h*. Suponga que el coste de cada movimiento es 1.

Representación de estados: ({bloques formando torre, especificando orden},{bloques aislados en la mesa sin especificar orden})

Estado inicial: ({B-C-A},{})

Estado final: ({A-B-C},{})

Desarrollamos el árbol de búsqueda (entre paréntesis, g(n)+h(n)=f(n), en A* se expande el nodo con menos valor f(n)):

Se expande ({B-C-A},{}) (0+2=2) – Frontera: ({C-A},{B}) (1+3=4) – Set explorado: ({B-C-A}, {})

Se expande ({C-A},{B}) (1+3=4) – Frontera: ({C-B},{A}) (2+3=5), ({B-C-A},{}) (2+2=4), ({},{A, B, C}) (2+2=4) – Set explorado: ({B-C-A}, {}), ({C-A},{B})

Hay empate. Si optamos por expandir ({B-C-A},{}) vemos que está en el set explorado, por lo que no se expande.

Se expande ({},{A,B,C}) (2+2=4) – Frontera: ({C-B},{A}) (2+3=5), ({C-A},{B}) (3+3=6), ({C-B},{A}) (3+3=6), ({B-A},{C}) (3+2=5), ({B-C},{A}) (3+1=4), ({A-B},{C}) (3+1=4) – Set explorado: ({B-C-A}, {}), ({C-A},{B}), ({},{A,B,C})

Vuelve a haber empate. Se expande, por ejemplo, ({A-B},{C}) (3+1=4) – Frontera: ({C-B},{A}) (2+3=5), ({C-A},{B}) (3+3=6), ({C-B},{A}) (3+3=6), ({B-A},{C}) (3+2=5), ({B-C},{A}) (3+1=4), ({C-A-B},{})

(4+2=6), ({A-C},{B}) (4+2=6), ({},{A,B,C}) (4+2=6) – Set explorado: ({B-C-A}, {}), ({C-A},{B}), ({},{A,B,C}), ({A-B},{C})

Se expande ({B-C},{A}) (3+1=4) – Frontera: ({C-B},{A}) (2+3=5), ({C-A},{B}) (3+3=6), ({C-B},{A}) (3+3=6), ({B-A},{C}) (3+2=5), ({C-A-B},{}) (4+2=6), ({A-C},{B}) (4+2=6), ({},{A,B,C}) (4+2=6), ({B-A},{C}) (4+2=6), ({A-B-C},{}) (4+0=4), ({},{A,B,C}) (4+2=6) – Set explorado: ({B-C-A}, {}), ({C-A},{B}), ({},{A,B,C}), ({A-B},{C}), ({B-C},{A})

Se expande ({A-B-C},{}) (4+0=4) – Objetivo alcanzado.

Resultado: ({B-C-A},{}) → ({C-A},{B}) → ({},{A,B,C}) → ({B-C},{A})→ ({A-B-C},{}) (4 pasos)