Intel·ligència Artificial

PAC1- Representació de problemes

Luis Enrique Arribas Zapater 17745245D

1.a

Nos encontramos ante un problema de planificación, por tanto la manera más sencilla de tratarlo es considerar a cada una de las posibles situaciones de ocupación del parking como un estado y las acciones de los operadores que se definan llevarán al sistema de un estado válido a otro estado válido.

Consideremos que cada plaza de parking puede hallarse a su vez en tres únicos estados válidos distintos: *ocupada por un abonado, ocupada por un no abonado o vacía*.

Podemos codificar estos tres estados como sigue:

vacío	no abonado	abonado
-1	0	1

Por otro lado necesitamos tener un control de la distancia o coste de recorrido de la plataforma desde la entrada hasta cada plaza. Para ello no nos basta con una lista de plazas y su estado, sino que modelaremos el parking como una matriz de 8 filas y 10 columnas como vemos a continuación:

	9	8	7	6	5	4	3	2	1	0
0										EXIT
1										
2										
3										
4										
5										
6										
7										

Por tanto el parking será modelado en 10 listas de 8 plazas, que pueden estar cada una en solo un estado de tres válidos posibles.

1.b

El espacio de estados será de 3^{80} estados, si bien no todos son accesibles desde otro estado. De hecho, sólo habrá 80 estados alcanzables desde cualquier otro estado, ya que sólo una de las plazas podría cambiar su valor en cada movimiento de la plataforma.

El sistema puede representarse siguiendo el siguiente esquema:

parking::== [columna i [piso j]]

Siendo *parking* una lista de columnas y *columna* una lista de plazas la representación pedida sería:

[0[1,1,1,1,1,1,1,1], 1[0,0,0,0,0,0,0,0], 2[1,1,1,1,1,1,1,1], 3[0,0,0,0,0,0,0,0], 4[1,1,1,1,1,1,1,1], 5[0,0,0,0,0,0,0,0] , 6[1,1,1,1,1,1,1,1], 7[0,0,0,0,0,0,0,0], 8[1,1,1,1,1,1,1,1], 9[0,0,0,0,0,0,0,0]]

2.

insertar_coche(es_abonado?)

Consideremos que la plataforma siempre se encuentra la plaza 0,0 ya que es el punto de entrada y salida del parking y es donde se recoge el coche.

Para diseñar el algoritmo de búsqueda de la plaza de menor coste veamos el esquema del parking donde cada plaza tiene asignada su coste, que es igual a la suma aritmética de los valores de fila y columna.

	9	8	7	6	5	4	3	2	1	0
0	9	8	7	6	5	4	3	2	1	0
1	10	9	8	7	6	5	4	3	2	1
2	11	10	9	8	7	6	5	4	3	2
3	12	11	10	9	8	7	6	5	4	3
4	13	12	11	10	9	8	7	6	5	4
5	14	13	12	11	10	9	8	7	6	5
6	15	14	13	12	11	10	9	8	7	6
7	16	15	14	13	12	11	10	9	8	7

por tanto un algoritmo para localizar la plaza libre más próxima a la salida debe recorrer todas las plazas candidatas siguiendo un orden de coste estrictamente creciente.

Considerando cada plaza como una tupla [i,j] donde i representa la columna y j la fila el algoritmo realizará una consulta del estado de las plazas como sigue:

[0,1] [1,0] [0,2] [1,1] [2,0] [0,3] [1,2] [2,1] [3,0] [0,4] [1,3] [2,2] [3,1] [4,0]…[9,7]

El algoritmo realizará una operación de comparación en cada plaza de la matriz siguiendo el orden definido hasta encontrar una valor menor que cero. Entonces detendrá la búsqueda y cambiará el valor de -1 al valor pasado como parámetro.

retirar_coche(columna, piso)

La ubicación de la plataforma no nos interesa en este caso, ya que debe retirarse el vehículo solicitado por el usuario y no otro (en base a criterios de eficiencia por ejemplo).

El algoritmo cambiará el valor de la tupla [i,j] pasada como parámetro por -1, indicando al sistema que la plaza está libre.

Como ejemplo tomemos como estado actual del sistema el de la respuesta de la pregunta 1.b y efectuamos la operación

retirar_coche(3,5)

Y el estado final del sistema será:

[0[1,1,1,1,1,1,1,1], 1[0,0,0,0,0,0,0,0], 2[1,1,1,1,1,1,1,1], 3[0,0,0,0,0,-1,0,0], 4[1,1,1,1,1,1,1,1], 5[0,0,0,0,0,0,0,0] , 6[1,1,1,1,1,1,1,1], 7[0,0,0,0,0,0,0,0], 8[1,1,1,1,1,1,1,1], 9[0,0,0,0,0,0,0,0]]

3.

Definiremos un nuevo operador ***optimizar_espacio()*** no parametrizado.

Será necesario añadir una nueva lista al sistema para modelar la plaza candidata al intercambio de menor coste con la forma *[columna, piso, valor]* que inicializaremos con valores nulos.

Una posible representación del sistema sería ahora:

[0[1,-1,1,1,1,1,1,1], 1[0,0,0,0,0,0,0,0], 2[1,1,1,1,1,1,1,1], 3[0,0,0,0,0,0,0,0], 4[1,1,1,1,1,1,1,1], 5[0,0,0,0,0,0,0,0] , 6[1,1,1,1,1,1,1,1], 7[0,0,0,0,0,0,0,0], 8[1,1,1,1,1,1,1,1], 9[0,0,0,0,0,0,0,0], [0, 1, -1]]

Realizaremos una operación de comparación del atributo "valor" de la plaza candidata en cada plaza durante el recorrido del parking y si la plaza candidata tiene un valor menor que la plaza recorrida realizaremos una operación de comparación de coste (que corresponde a la respectivas sumas de los valores de columna y piso) y si la plaza candidata tiene menor coste realizaremos una operación de intercambio de valores en la estructura del parking entre ambas plazas y la plaza de las dos que haya quedado con menor valor pasará a ser la plaza candidata.

Para el intercambio de dos plazas ocupadas necesitaremos siempre una plaza vacía pero la garantía de que existe una plaza (81) siempre vacía con esta función nos permite obviar esta necesidad la hora de modelar la solución. Por esto podemos simplemente intercambiar los valores de dos plazas ocupadas sin preocuparnos de cómo hará el sistema para intercambiar los coches.

4.a

El estado objetivo será tal que las 32 plazas de menor coste tengan un valor 1, las siguientes 32 de menor coste un valor 0 y el resto un valor -1.

Para ello consideremos el estado inicial:

[0[1,1,1,1,1,1,1,1], 1[0,0,0,0,0,0,0,0], 2[1,1,1,1,1,1,1,1], 3[0,0,0,0,0,0,0,0] 4[1,1,1,1,1,1,1,1], 5[0,0,0,0,0,0,0,0] , 6[1,1,1,1,1,1,1,1], 7[0,0,0,0,0,0,0,0], 8[1,1,1,1,1,1,1,1], 9[0,0,0,0,0,0,0,0] [NULL, NULLL, NULL]]

Veamos una representación gráfica del estado objetivo para mayor claridad. Las plazas que serán ocupadas por abonados en amarillo y las de no abonados en azul. Nótese que en realidad hay otros estados objetivo que cumplirían las condiciones descritas pero este sería el alcanzable usando el algoritmo de recorrido definido en la pregunta 2.

	9	8	7	6	5	4	3	2	1	0
0	9	8	7	6	5	4	3	2	1	0
1	10	9	8	7	6	5	4	3	2	1
2	11	10	9	8	7	6	5	4	3	2
3	12	11	10	9	8	7	6	5	4	3
4	13	12	11	10	9	8	7	6	5	4
5	14	13	12	11	10	9	8	7	6	5
6	15	14	13	12	11	10	9	8	7	6
7	16	15	14	13	12	11	10	9	8	7

Por tanto, la representación formal del estado objetivo del sistema será:

[0 [1,1,1,1,1,1,1,1], 1[1,1,1,1,1,1,1,0], 2[1,1,1,1,1,1,0,0], 3[1,1,1,1,1,0,0,0], 4[1,1,1,0,0,0,0,0], 5[1,1,0,0,0,0,0,-1], 6[1,0,0,0,0,0,-1,-1], 7[0,0,0,0,0,-1,-1,-1], 8[0,0,0,0,-1,-1,-1,-1], 9[0,0,-1,-1,-1,-1,-1,-1], [5, 7, -1]]

4.b

En una búsqueda no informada el algoritmo recorrerá las plazas como hemos visto anteriormente en orden ascendente de coste, esto es, diagonales de abajo a arriba comenzando por la plaza EXIT, lo que sería una búsqueda en anchura (BFS) de menor a mayor coste.

5.

Una posible representación sería la sustitución de los valores atómicos de la lista columna por tuplas (listas) de dos elementos, representando ambos lados de la plataforma. De este modo nos garantizamos que el coste de desplazamiento de cada plaza es igual que la representada por el otro elemento de la tupla.

[0 [[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 1[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 2[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 3[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 4[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 5[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[-1,1]], 6[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 7[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 8[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], 9[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]], [5, 7, 0, -1]]

La lista de plaza candidata debería poder permitir representar si la plaza es la que está a un lado o a otro, parra lo que añadimos un atributo cuyos estados válidos son 0 y 1, representando izquierda y derecha.

En el operador retirar_coche(columna, piso), será necesario añadir este nuevo valor binario, de modo que los parámetros pasasen a ser tres.

En el operador *insertar_coche(valor) y en el el operador* ***optimizar_espacio()***
la búsqueda de plaza debe tener en cuenta la nueva estructura del sistema y realizar la comparación de valor con ambos miembros de la tupla de cada plaza.