Laboratorio PL1b

Unai Sainz de la Maza y Unai Salas

8 de Marzo de 2021

1 Corners Problem: Representación

En este apartado tenemos que implementar una codificación para el problema de las esquinas. El estado consistirá en un campo para la posición (dupla: x, y), y cuatros booleanos que representan las cuatro esquinas (c1, c2, c3, c4).

Utilizando la codificación anterior, las funciones $getStartState()$ y $isGoalState()$ nos quedarían de la siguiente forma:

- $getStartState()$: devuelve como posición la posición inicial ($self.startingPosition$) y cuatro campos booleanos seteados en $False$.
- $isGoalState()$: primero desempaqueta el campo state (codificado de la forma anteriormente mencionada), luego comprueba si todas las esquinas han sido visitadas ($c1 \wedge c2 \wedge c3 \wedge c4$).

2 Corners Problem: Heurístico

Se nos pide implementar un heurístico no trivial y consistente para el problema de las cuatro esquinas. Para implementar dicho heurístico en la función correspondiente ($cornersHeuristic()$), tenemos que utilizar la información de los corners ($problem.corners$), y la información que hemos codificado en el apartado anterior.

Para ello hemos organizado dicha información de la siguiente manera:

- Posición de los corners: hemos divido en cuatro variables la posición de las cuatro esquinas (c_p1, c_p2, c_p3, c_p4).
- Información del estado: desempaquetamos el estado codificado en cinco variables ($pos, c1, c2, c3, c4$), contiene la posición y las cuatro variables booleanas que representan si una esquina está visitada o no.
- Coordenadas: la posición (pos) la desempaquetamos en dos variables (x, y), esto lo hacemos para facilitar el trabajo a la hora de calcular el heurístico.

Heurístico:

- Para los corners no visitados (variables $c_x == False$, con $x = 1, 2, 3, 4$): hacemos la diferencia (en valor absoluto) de las coordenadas de los corners y las del estado, la diferencia se hace por componentes de las coordenadas, por ejemplo, $abs(c\_p1[0] - x)$.
  Sumamos las diferencias en valor absoluto anteriormente calculadas, por ejemplo, $(abs(c\_p1[0] - x) + abs(c\_p1[1] - y)$.
- Para los corners visitados (variables $c_x == True$, con $x = 1, 2, 3, 4$): devuelve 0.
- Devolvemos el máximo de la suma de diferencias, es decir, siendo $d1, d2, d3, d4$ las sumas de diferencias, $return \max([d1, d2, d3, d4])$.

Año: 2020-2021
Profesor(a): Ekaitz Jauregi
E-mail: ekaitz.jauregi@ehu.eus

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

3 Eating All The Dots

En este apartado se nos pide que implementemos un heurístico para comer toda la comida en el menor número de pasos posible. Para poder implementar el heurístico tenemos que tener en cuenta la información que se nos proporciona, es decir, saber qué información tenemos codificada para hacer uso de ella.

- *state*: este parámetro contiene una tupla $(pacmanPosition, foodGrid)$, la primera contiene la posición de pacman y la segunda es un grid con valores $True$ y $False$, aunque nosotros usaremos el método $foodGrid.asList()$ para obtener la lista de las coordenadas de la comida.

Nosotros hemos obtenido de dicho parámetro (*state*), la información y la hemos guardado en varias variables:

- *position*: guarda la información relativa a la posición del pacman.
- f_list: lista que contiene las coordenadas de la comida.

Heurístico: haciendo uso de la función $mazeDistance()$, devolvemos la mayor distancia entre las comidas y la posición del pacman.

- $mazeDistance(p1, p2, gameState)$: devuelve la distancia desde p1 a p2 (teniendo en cuenta las paredes, esto lo hace gracias al parámetro gameState), para el cálculo de la distancia utiliza el método $bfs()$ previamente implementado.

Proceso para el cálculo del heurístico:

- Inicializamos el valor a devolver (max_d) a cero.
- Recorremos la lista de las coordenadas de la comida (f_list) y en cada iteración calculamos la distancia desde el pacman a dichas coordenadas de la comida (llamando a $mazeDistance()$).
- Si la distancia calculada (d) es mayor a la última distancia guardada en max_d, cambiamos $max\_d = d$.
- Devolvemos max_d.

4 Suboptimal Search

Debemos implementar las funciones $findPathToClosestDot()$ y $isGoalState()$:

- $findPathToClosestDot()$: necesitamos que devuelva el camino más corto hasta un punto, esto lo conseguimos simplemente llamando al método $bfs()$, de esta forma encontramos el camino más corto hasta dicho punto.
- $isGoalState()$: para saber si hemos terminado (estamos en el *goalState*), comprobamos si que las coordenadas del pacman están en la lista de coordenadas de la comida (f_list). También podríamos comprobar si todos los campos en el grid ($foodGrid$) están seteados a $True$.

5 Resultados del autograder

Hemos conseguido la puntuación máxima:

- Q5: 3/3
- Q6: 3/3
- Q7: 5/4
- Q8: 3/3
- Puntuación total hasta la fecha (contando Q1, Q2, Q3 Y Q4): 26/25