Universidad Rey Juan Carlos

Práctica 1: Resolución de problemas por búsqueda

Test de modificaciones

Inteligencia Artificial

17 de octubre - Curso 2023/2024

Introducción

En este documento se proponen una serie de modificaciones al contenido de la Práctica 1. Se trabajará sobre la versión completa de la práctica incluida en el repositorio alojado en el GitLab de la EIF, que podrá ser descargado a través del siguiente comando:

```
git clone https://gitlab.eif.urjc.es/[nombre usuario]/[nombre repositorio].git
```

- Las modificaciones propuestas deben incorporarse sobre los archivos `search.py` y `searchAgents.py`, no siendo necesario aplicar ningún cambio sobre otros archivos. Una vez completadas, la versión **modificada** de los dos archivos anteriormente mencionados debe entregarse a través del espacio de entrega habilitado en el Aula Virtual.
- Se dispone de **1h 30 minutos** para abordar las modificaciones requeridas.
- El código tiene que ir obligatoriamente comentado explicando su funcionalidad. Debe ser legible y estar debidamente tabulado.
- Se utilizarán sistemas anticopia y se podrá requerir explicación individual de la práctica en caso de duda.

1. Búsqueda DFS de profundidad iterativa (3,5 ptos)

Una variante sencilla de la búsqueda primero en profundidad (DFS) se denomina "búsqueda de profundidad iterativa". Esta estrategia es una combinación de la búsqueda en profundidad y la búsqueda en amplitud que suele utilizarse cuando se desconoce la profundidad del árbol de búsqueda. La idea es realizar sucesivas búsquedas en profundidad en las que la máxima profundidad que se puede alcanzar está limitada, pero en las que se va aumentando gradualmente este límite hasta encontrar el objetivo.

El pseudocódigo de la búsqueda en profundidad con límite de profundidad es el que se proporciona a continuación:

```
frontier = {startNode}
expanded = {}
while frontier is not empty:
    node  =  frontier.pop()
    if isGoal(node):
        return path_to_node
    if node not in expanded and depth < depth_limit:
        expanded.add(node)
        for each child of node's children:
            frontier.push(child)
return None
```

Se resalta y subraya la parte que varía respecto a la búsqueda en profundidad normal.

Para llevar a cabo la búsqueda de profundidad iterativa, hay que llamar a la función de búsqueda en profundidad con límite de profundidad de forma iterativa, aumentando el límite en una unidad cada vez, hasta que devuelve un resultado. Este resultado será la solución.

Se pide:

a) Crear una función `depthLimitedDFS` que implemente el pseudocódigo proporcionado. Se puede (y se recomienda) reutilizar código ya escrito (1,5 ptos).

b) Implementar una función `iterativeDeepening` que llame iterativamente a `depthLimitedDFS`. Esta función debe ser intercambiable con las implementadas en la práctica para las distintas estrategias de búsqueda y, por tanto, aceptar los mismos parámetros de entrada y devolver el mismo tipo de salida que éstas (1,5 ptos). En ese caso, podrá probarse con:

```
python3 pacman.py -l tinyMaze -p SearchAgent -a fn=iterativeDeepening
python3 pacman.py -l mediumMaze -p SearchAgent -a fn=iterativeDeepening
python3 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=iterativeDeepening
```

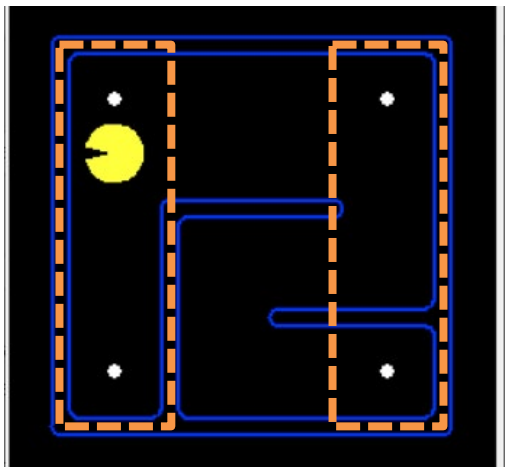
c) Incluir como comentario dentro de la función `iterativeDeepening` las diferencias observadas con las demás estrategias de búsqueda implementadas en la práctica (0,5 ptos).

Nota: Las funciones deben ubicarse junto a sus análogas implementadas en la práctica.

2. Problema de tocar los extremos (3,5 ptos)

En la práctica se ha formulado el problema de visitar todas las esquinas. En este ejercicio se pide incorporar un problema de búsqueda diferente que consiste en tocar los dos extremos laterales (izquierda y derecha) del laberinto.

Por ejemplo, en este laberinto:



Se quiere encontrar un camino que permita a Pacman entrar en **alguna** (no todas) de las casillas comprendidas dentro de cada uno de los rectángulos dibujados con línea discontinua en los extremos izquierdo y derecho del laberinto.

a) Crear una clase `SidesProblem` análoga a la `CornersProblem` en la que se defina el espacio de estados, el estado inicial, el test objetivo y la función sucesora rellenando los atributos y métodos correspondientes (2 ptos). Se puede (y se recomienda) empezar con una copia de `CornersProblem` y hacer las modificaciones pertinentes sobre ella. Si se descargan los ficheros tinySides.lay y mediumSides.lay y se colocan en la carpeta layouts del proyecto, se puede probar con:

```
python3 pacman.py -l tinySides -p SearchAgent -a fn=bfs,prob=SidesProblem
python3 pacman.py -l mediumSides -p SearchAgent -a fn=bfs,prob=SidesProblem
```

b) Crear una heurística `sidesHeuristic`, equivalente a `cornersHeuristic`, que calcule una heurística no trivial (1,5 pto). Esta heurística debe estar basada en la distancia y el único requisito es que expanda menos de 406 nodos al ejecutar el siguiente comando:

```
python3 pacman.py -l mediumSides -p SearchAgent -a
fn=astar,prob=SidesProblem,heuristic=sidesHeuristic
```

Notas: ver siguiente página.

Notas:

- Las funciones deben ubicarse junto a sus análogas implementadas en la práctica.
- Aunque no debería ser necesario, se puede probar con funciones de búsqueda distintas a las propuestas, excepto con la “primero en profundidad” (DFS), que no funcionará en los laberintos proporcionados.
- El siguiente código del constructor de `CornersProblem` puede eliminarse:

```
for corner in self.corners:
    if not startingGameState.hasFood(*corner):
        print('Warning: no food in corner ' + str(corner))
```