Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

NOTA: Para la realización del trabajo se usó **Claude(IA)**, e hicimos los siguientes prompts para llegar a las respuestas:

PRIMERO PROMPT

Conoces el proyecto berkeley sobre la implementación de algoritmos de búsqueda para el juego de pacman, construido en Python 2.7.0. A continuación, se detallan las instrucciones para resolver el problema:

Instrucciones

Debes implementar la función depthFirstSearch. La implementación debe cumplir con los siguientes requisitos:

1. **Implementar DFS:**
   - Implementa la versión de búsqueda en profundidad (DFS) utilizando una búsqueda de grafos para evitar expandir estados ya visitados.
   - El algoritmo debe retornar una lista de acciones que lleven al agente desde el estado inicial hasta el objetivo. Asegúrate de que todas las acciones sean movimientos legales (no pasar a través de paredes).
2. **Uso de Estructuras de Datos:**
   - Utiliza las estructuras de datos Stack, Queue y PriorityQueue proporcionadas en util.py. Estas implementaciones tienen propiedades específicas necesarias para la compatibilidad con el sistema de calificación automática.

   Consideraciones:

   - Si usas una Stack como estructura de datos, la solución encontrada por tu algoritmo de DFS debes tener en cuenta que si apilas los sucesores en el orden proporcionado por la función problem.getSuccessors podrías obtener una solución no óptima.

Código de Inicio

Aquí tienes el código proporcionado para tinyMazeSearch y el esqueleto de depthFirstSearch:

```
python

def tinyMazeSearch(problem):
    """
    Returns a sequence of moves that solves tinyMaze.  For any other maze, the
    sequence of moves will be incorrect, so only use this for tinyMaze.
    """
    from game import Directions
    s = Directions.SOUTH
    w = Directions.WEST
    return  [s, s, w, s, w, w, s, w]


def depthFirstSearch(problem):
```

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```
    """
    Search the deepest nodes in the search tree first.
    Your search algorithm needs to return a list of actions that reaches the
    goal. Make sure to implement a graph search algorithm.
    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:
    "Start:", problem.getStartState()
    "Is the start a goal?", problem.isGoalState(problem.getStartState())
    "Start's successors:", problem.getSuccessors(problem.getStartState())
    """
    "* YOUR CODE HERE *"
```

Tareas a Realizar

- Completa la implementación de depthFirstSearch.
- Asegúrate de que la función devuelva la lista correcta y óptima de acciones para alcanzar el objetivo.

Código Realizado

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```python
def depthFirstSearch(problem):
    """
    Search the deepest nodes in the search tree first.

    Your search algorithm needs to return a list of actions that reaches the
    goal. Make sure to implement a graph search algorithm.

    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:

    print "Start:", problem.getStartState()
    print "Is the start a goal?", problem.isGoalState(problem.getStartState())
    print "Start's successors:", problem.getSuccessors(problem.getStartState())
    """
    "*** YOUR CODE HERE ***"

    # pila
    stack = util.Stack()
    # Conjunto visitados
    visited = set()
    # Añadir el estado inicial a la pila
    start_state = problem.getStartState()
    stack.push((start_state, []))
    while not stack.isEmpty():
        current_state, actions = stack.pop()
        # final objetivo
        if problem.isGoalState(current_state):
            return actions
        # Si el estado no ha sido visitado, explorarlo
        if current_state not in visited:
            visited.add(current_state)
            # Obtener los sucesores del estado actual
            for successor, action, _ in problem.getSuccessors(current_state):
                if successor not in visited:
                    # Añadir el sucesor a la pila con las acciones actualizadas
                    new_actions = actions + [action]
                    stack.push((successor, new_actions))
    # Si no se encuentra solución, devolver una lista vacía
    return []
```

SEGUNDO PROMPT

Desarrolla un algoritmo para Python 2.7.0 que implemente el algoritmo de búsqueda en anchura (Breadth First Search, BFS) en la función breadthFirstSearch en el archivo search.py. Sigue las instrucciones a continuación para resolver el problema:

Contexto del Problema

Debes implementar la función breadthFirstSearch siguiendo la estructura del DFS que creaste anteriormente. La implementación debe cumplir con los siguientes requisitos:

1. **Implementar BFS:**

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

- Implementa la búsqueda en anchura (BFS) utilizando una búsqueda de grafos para evitar expandir estados ya visitados.
- Asegúrate de que tu algoritmo devuelva una lista de acciones que lleven al agente desde el estado inicial hasta el objetivo. Las acciones deben ser movimientos legales (no pasar a través de paredes).

Código de Inicio

Aquí tienes el esqueleto de la función breadthFirstSearch que debes completar:

python

Copiar código

```
def breadthFirstSearch(problem):
    """
    Search the shallowest nodes in the search tree first.
    Your search algorithm needs to return a list of actions that reaches the
    goal. Make sure to implement a graph search algorithm.
    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:
    """
    "* YOUR CODE HERE *"
```

Tareas a Realizar

- Completa la implementación de breadthFirstSearch.
- Asegúrate de que la función devuelva la lista correcta de acciones para alcanzar el objetivo y **que sea capaz de encontrar la solución óptima.**

Código Realizado

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```python
def breadthFirstSearch(problem):
    """
    Search the shallowest nodes in the search tree first.
    """

    # cola
    queue = util.Queue()
    # Conjunto visitados
    visited = set()
    # Añadir el estado inicial a la cola
    start_state = problem.getStartState()
    queue.push((start_state, []))
    while not queue.isEmpty():
        current_state, actions = queue.pop()
        # fin objetivo
        if problem.isGoalState(current_state):
            return actions
        # Si el estado no ha sido visitado, explorarlo
        if current_state not in visited:
            visited.add(current_state)
            # Obtener los sucesores del estado actual
            for successor, action, _ in problem.getSuccessors(current_state):
                if successor not in visited:
                    # Añadir el sucesor a la cola con las acciones actualizadas
                    new_actions = actions + [action]
                    queue.push((successor, new_actions))
    # Si no se encuentra solución, devolver una lista vacía
    return []
```

TERCERO PROMPT

Desarrolla un algoritmo que implemente el algoritmo de búsqueda de costo uniforme (Uniform Cost Search, UCS) en la función uniformCostSearch. Sigue las instrucciones a continuación para resolver el problema:

Contexto del Problema

Debes implementar la función uniformCostSearch. Este algoritmo debe encontrar el camino con el menor costo total, considerando que el costo puede variar según el entorno del laberinto.

1. **Implementar UCS:**
    - Implementa la búsqueda de costo uniforme, que busca el camino con el costo total más bajo. A diferencia de BFS, UCS toma en cuenta el costo asociado a cada paso.
    - La función debe devolver una lista de acciones que lleven al agente desde el estado inicial hasta el objetivo, minimizando el costo total del camino.

Código de Inicio

Aquí tienes el esqueleto de la función uniformCostSearch que debes completar:

python

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

Copiar código

```
def uniformCostSearch(problem):
    """
    Search the lowest cost nodes in the search tree first.
    Your search algorithm needs to return a list of actions that reaches the
    goal with the lowest total cost. Make sure to implement a graph search algorithm.
    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:
    """
    "* YOUR CODE HERE *"
```

Tareas a Realizar

- Completa la implementación de uniformCostSearch.
- Asegúrate de que la función devuelva la lista correcta de acciones para alcanzar el objetivo con el costo total más bajo.

Código Realizado

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```python
def uniformCostSearch(problem):
    """
    Search the node of least total cost first.
    """

    # cola de prioridad
    pq = util.PriorityQueue()
    # Conjunto visitados
    visited = set()
    # Añadir el estado inicial a la cola de prioridad
    start_state = problem.getStartState()
    pq.push((start_state, [], 0), 0)
    while not pq.isEmpty():
        current_state, actions, current_cost = pq.pop()
        # fin objetivo
        if problem.isGoalState(current_state):
            return actions
        # Si el estado no ha sido visitado, explorarlo
        if current_state not in visited:
            visited.add(current_state)
            # Obtener los sucesores del estado actual
            for successor, action, step_cost in problem.getSuccessors(current_state):
                if successor not in visited:
                    # Calcular el nuevo costo y las nuevas acciones
                    new_cost = current_cost + step_cost
                    new_actions = actions + [action]
                    # Añadir el sucesor a la cola de prioridad
                    pq.push((successor, new_actions, new_cost), new_cost)
    # Si no se encuentra solución, devolver una lista vacía
    return []
```

CUARTO PROMPT

Desarrolla un algoritmo que implemente el algoritmo de búsqueda A* (A* Search) en la función aStarSearch. Sigue las instrucciones a continuación para resolver el problema:

Contexto del Problema

Debes implementar la función aStarSearch. A* es una búsqueda informada que utiliza una función heurística para estimar el costo desde el estado actual hasta el objetivo.

1. ***Implementar A:****
    - Implementa la búsqueda A* utilizando una función heurística. A* combina el costo del camino desde el inicio hasta el estado actual (g(n)) con una estimación del costo restante para llegar al objetivo (h(n)). La función heurística se pasa como argumento y debe evaluar el costo estimado desde el estado actual hasta el objetivo.
    - La función heurística debe tomar dos argumentos: el estado actual y el problema (para referencia adicional).
2. **Heurísticas:**
    - Puedes probar tu implementación de A* usando la heurística de Manhattan (manhattanHeuristic). Esta heurística mide la distancia Manhattan entre el estado actual y el objetivo.
3. **Nota Adicional:**

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

- Asegúrate de que tu función aStarSearch maneje adecuadamente los costos y la heurística para encontrar el camino más eficiente.

Código de Inicio

Aquí tienes el esqueleto de la función aStarSearch que debes completar:

python

Copiar código

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """
    Search the nodes with the lowest combined cost and heuristic value first.
    Your search algorithm needs to return a list of actions that reaches the
    goal with the lowest total cost plus heuristic estimate. Make sure to
    implement a graph search algorithm.
    The heuristic function must take two arguments: a state in the search
    problem and the problem itself.
    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:
    """
    "* YOUR CODE HERE *"
```

Tareas a Realizar

- Completa la implementación de aStarSearch.
- Asegúrate de que la función devuelva la lista correcta de acciones para alcanzar el objetivo con el costo total más bajo más la estimación heurística.

Código Realizado

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```python
def aStarSearch(problem, heuristic=nullHeuristic):
    """
    Search the node that has the lowest combined cost and heuristic first.
    """

    # cola prioridad
    pq = util.PriorityQueue()
    # Conjunto visitados
    visited = set()
    # Añadir el estado inicial a la cola de prioridad
    start_state = problem.getStartState()
    start_h = heuristic(start_state, problem)
    pq.push((start_state, [], 0), start_h)
    while not pq.isEmpty():
        current_state, actions, current_cost = pq.pop()
        # fin objetivo
        if problem.isGoalState(current_state):
            return actions
        # Si el estado no ha sido visitado, explorarlo
        if current_state not in visited:
            visited.add(current_state)
            # Obtener los sucesores del estado actual
            for successor, action, step_cost in problem.getSuccessors(current_state):
                if successor not in visited:
                    # Calcular el nuevo costo g (costo real hasta ahora)
                    new_cost = current_cost + step_cost
                    # Calcular el costo h (heurístico estimado hasta el objetivo)
                    h = heuristic(successor, problem)
                    # Calcular el costo f total (f = g + h)
                    f = new_cost + h
                    new_actions = actions + [action]
                    # Añadir el sucesor a la cola de prioridad
                    pq.push((successor, new_actions, new_cost), f)
    # Si no se encuentra solución, devolver una lista vacía
    return []
```

Autograder Resultados

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```
$ C:/Python27/python.exe "c:/Users/FAMILIA/Desktop/TODO MIGUE/Universidad Javeriana Cali/Asignaturas-Semestre/8.
/JuegoPacman/autograder.py"
Starting on 8-31 at 12:40:25

Question q1
===========

*** PASS: test_cases\q1\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'D', 'C']
*** PASS: test_cases\q1\graph_bfs_vs_dfs.test
***     solution:               ['2:A->D', '0:D->G']
***     expanded_states:        ['A', 'D']
*** PASS: test_cases\q1\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q1\graph_manypaths.test
***     solution:               ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***     expanded_states:        ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases\q1\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 130
***     nodes expanded:         146

### Question q1: 3/3 ###
```

```
Question q2
===========

*** PASS: test_cases\q2\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q2\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q2\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q2\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q2\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 3/3 ###
```

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```
Question q3
===========

*** PASS: test_cases\q3\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q3\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q3\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q3\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q3\ucs_0_graph.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q3\ucs_1_problemC.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269
*** PASS: test_cases\q3\ucs_2_problemE.test
***     pacman layout:          mediumMaze
***     solution length: 74
***     nodes expanded:         260
*** PASS: test_cases\q3\ucs_3_problemW.test
***     pacman layout:          mediumMaze
***     solution length: 152
***     nodes expanded:         173
*** PASS: test_cases\q3\ucs_4_testSearch.test
***     pacman layout:          testSearch
***     solution length: 7
***     nodes expanded:         14
*** PASS: test_cases\q3\ucs_5_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']

### Question q3: 3/3 ###
```

Miguel Angel Nivia
Willian Chapid Tobar
Daniel Vasquez Murillo

```
Question q4
===========

*** PASS: test_cases\q4\astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q4\astar_1_graph_heuristic.test
***     solution:               ['0', '0', '2']
***     expanded_states:        ['S', 'A', 'D', 'C']
*** PASS: test_cases\q4\astar_2_manhattan.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         221
*** PASS: test_cases\q4\astar_3_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q4\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q4\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###
```

```
Finished at 12:40:25

Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 0/3
Question q6: 0/3
Question q7: 0/4
Question q8: 0/3
------------------
Total: 12/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

FAMILIA (main *) JuegoPacman
$
```