MUPPANA TEJASWINI

2320030271

# AIML-PYTHON CODE ASSIGNMENT-1

1)BREADTH FIRST SEARCH:

def breadth\_first\_search(graph, start, goal):

"""Breadth-First Search for shortest path."""

queue = [(start, [start])]

visited = set()

while queue:

node, path = queue.pop(0)

if node == goal:

return path

visited.add(node)

for neighbor in graph.get(node, {}):

if neighbor not in visited:

visited.add(neighbor)

queue.append((neighbor, path + [neighbor]))

return None

2)DEPTH FIRST SEARCH:

def depth\_first\_search(graph, start, goal):

"""Depth-First Search for path."""

stack = [(start, [start])]

visited = set()

while stack:

node, path = stack.pop()

if node == goal:

return path

visited.add(node)

for neighbor in graph.get(node, {}):

if neighbor not in visited:

visited.add(neighbor)

stack.append((neighbor, path + [neighbor]))

return None

3)UNIFORM COST SEARCH:

def uniform\_cost\_search(graph, start, goal):

"""Uniform-Cost Search for shortest path."""

heap = [(0, start, [start])]

visited = set()

while heap:

cost, node, path = heapq.heappop(heap)

if node == goal:

return path

visited.add(node)

for neighbor, edge\_cost in graph.get(node, {}).items():

if neighbor not in visited:

visited.add(neighbor)

heapq.heappush(heap, (cost + edge\_cost, neighbor, path + [neighbor]))

return None

4)GREEDY BEST FIRST SEARCH:

def greedy\_best\_first\_search(graph, start, goal, heuristic):

"""Greedy Best-First Search for path."""

heap = [(heuristic(start, goal), start, [start])]

visited = set()

while heap:

\_, node, path = heapq.heappop(heap)

if node == goal:

return path

visited.add(node)

for neighbor in graph.get(node, {}):

if neighbor not in visited:

visited.add(neighbor)

heapq.heappush(heap, (heuristic(neighbor, goal), neighbor, path + [neighbor]))

return None

5)A\*SEARCH:

def a\_star\_search(graph, start, goal, heuristic):

"""A\* Search for shortest path."""

heap = [(heuristic(start, goal), 0, start, [start])]

visited = set()

while heap:

\_, cost, node, path = heapq.heappop(heap)

if node == goal:

return path

visited.add(node)

for neighbor, edge\_cost in graph.get(node, {}).items():

if neighbor not in visited:

visited.add(neighbor)

heapq.heappush(heap, (cost + edge\_cost + heuristic(neighbor, goal), cost + edge\_cost, neighbor, path + [neighbor]))

return None

6)MINIMUM SEARCH:

def minimax(game, depth, maximizing\_player):

"""Minimax algorithm for adversarial search."""

if depth == 0 or game.is\_terminal():

return game.evaluate()

if maximizing\_player:

max\_eval = float('-inf')

for move in game.get\_legal\_moves():

game.make\_move(move)

eval = minimax(game, depth - 1, False)

game.undo\_move()

max\_eval = max(max\_eval, eval)

return max\_eval

else:

min\_eval = float('inf')

for move in game.get\_legal\_moves():

game.make\_move(move)

eval = minimax(game, depth - 1, True)

game.undo\_move()

min\_eval = min(min\_eval, eval)

return min\_eval