National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

CS-272 Artificial Intelligence

BSAI-1 A

Lab 6: Informed Search Strategies

CLO 3

Date: 15-10-2025

Lab Engineer: Mr Hafiz Arslan Ramzan
Instructor: Dr Seemab Latif

Lab 5: Informed Search Strategies

Introduction

The purpose of this lab is design and implement Uniform cost search, Greedy search, and A* search. This lab has three activities related to the search strategies.

Objectives

- To understand uninformed and informed search strategies.
- To implement search strategies for solving a problem.

Description

In this lab, you will develop search strategies for finding paths for the pac-man in three different maze sizes. In this game, pac-man is your agent. Your pac-man agent will find paths through the maze world to reach a particular location. You will build general search algorithms and apply them to many different pac-man scenarios. This scenario is same as Lab 2.

Assets/Data

Data and instructions are same as Lab 2. Kindly refer to Lab 2 handout.

Activities

Activity 1 – Uniform Cost Function:

While BFS will find a fewest-actions path to the goal, we might want to find paths that are "best" in other senses. Consider mediumDottedMaze and mediumScaryMaze.

By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response.

Implement the uniform-cost graph search algorithm in the uniformCostSearch function in search.py. We encourage you to look through util.py for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

Note: You should get very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions (see searchAgents.py for details).

SOLUTION:

CODE:

```python
def uniformCostSearchStats(problem):

    start = problem.getStartState() # initial
    frontier = util.PriorityQueue() # priority queue for ucs

    frontier.push((start, [] , 0), 0)  # state, actions, cost ; priority = cost

    frontier_puts = 1
    frontier_gets = 0   # keeping track of pushes and gets
    explored = dict()  # to avoid repetition
    while not frontier.isEmpty():
        state, actions, cost_so_far = frontier.pop()
        frontier_gets += 1
        if state in explored and explored[state] <= cost_so_far:    # if state already explored at equal or lower cost
            continue
        explored[state] = cost_so_far   # new
        if problem.isGoalState(state): # goal check
            return actions, cost_so_far, frontier_gets, frontier_puts
        for succ, action, stepCost in problem.getSuccessors(state): # negihbours
            new_cost = cost_so_far + stepCost
            if succ not in explored or new_cost < explored.get(succ, float('inf')):     #state not explroed or cheaper path
                frontier.push((succ, actions + [action], new_cost), new_cost)   # push with new cost as priority
                frontier_puts += 1
    return None, 0, frontier_gets, frontier_puts    # default
    util.raiseNotDefined()

def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    return uniformCostSearchStats(problem)[0]
```

OUTPUT:

```
PS D:\codes\fall25\ai\lab6> python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
PS D:\codes\fall25\ai\lab6>
```

```
PS D:\codes\fall25\ai\lab6> python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
PS D:\codes\fall25\ai\lab6> python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
PS D:\codes\fall25\ai\lab6>
```

As mentioned above, I am getting low and high costs for both of the last two agents respectively, highlighting the correctness of my code.

Activity 2 – A*:

Implement A* graph search in the empty function aStarSearch in search.py. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information).

The nullHeuristic heuristic function in search.py is a trivial example.

You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py).

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

You should see that A* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly). What happens on openMaze for the various search strategies?

SOLUTION:
CODE:

```python
def aStarSearchStats(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    start = problem.getStartState() # similar to ucs
    frontier = util.PriorityQueue()
    start_h = heuristic(start, problem) # calc hueristic
    frontier.push((start, [], 0), start_h)
    frontier_puts = 1
    explored = dict()  # dict because pointing state -> best g
    frontier_gets = 0
    while not frontier.isEmpty():
        state, actions, g = frontier.pop()
        frontier_gets += 1
        if state in explored and explored[state] <= g:  # already found one with <= cost
            continue
        explored[state] = g
        if problem.isGoalState(state):  # goal check
            return actions, g, frontier_gets, frontier_puts
        for succ, action, stepCost in problem.getSuccessors(state): # neighbours
            new_g = g + stepCost
            h = heuristic(succ, problem)
            f = new_g + h   # logic begind A*
            if succ not in explored or new_g < explored.get(succ, float('inf')):    # state not explored or lower cost
                frontier.push((succ, actions + [action], new_g), f)
                frontier_puts += 1
    return None, 0, frontier_gets, frontier_puts

def aStarSearch(problem, heuristic=nullHeuristic):
    """ Do not change this function"""
    return aStarSearchStats(problem, heuristic)[0]
```

OUTPUT:

```
PS D:\codes\fall25\ai\lab6> python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

ON “openMaze”:

```
PS D:\codes\fall25\ai\lab6> python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
PS D:\codes\fall25\ai\lab6>
```

On openMaze, A* expands far fewer nodes than UCS because the Manhattan heuristic perfectly matches the true path cost in open space. It goes almost straight to the goal, finding the optimal solution much faster (optimal and efficient).

Activity 3 – Greedy Search:

Implement Greedy graph search in the function GreedySearch **(define it yourself)** in search.py. Greedy search function should take a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information).

You can test your Greedy search implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py).

SOLUTION:

CODE:

```python
def GreedySearchStats(problem, heuristic=nullHeuristic):
    """Greedy Best-First Search: priority = heuristic(state)."""
    start = problem.getStartState() # very similar implementationto both of the above functions - like ucs and also A*
    frontier = util.PriorityQueue()
    frontier.push((start, [], 0), heuristic(start, problem))
    frontier_puts = 1
    explored = set()
    frontier_gets = 0
    while not frontier.isEmpty():
        state, actions, g = frontier.pop()
        frontier_gets += 1
        if state in explored:   # no additional check because cost not involved here only heuristic involved
            continue
        explored.add(state)
        if problem.isGoalState(state):
            return actions, g, frontier_gets, frontier_puts
        for succ, action, stepCost in problem.getSuccessors(state):
            if succ not in explored:
                frontier.push((succ, actions + [action], g + stepCost), heuristic(succ, problem))
                frontier_puts += 1
    return None, 0, frontier_gets, frontier_puts

def GreedySearch(problem, heuristic=nullHeuristic):
    return GreedySearchStats(problem, heuristic)[0]

# Abbreviations
bfs = breadthFirstSearch
dfs = depthFirstSearch
astar = aStarSearch
ucs = uniformCostSearch
greedy = GreedySearch
```

OUTPUT:

```
PS D:\codes\fall25\ai\lab6> python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=greedy,heuristic=manhattanHeuristic --frameTime 0
[SearchAgent] using function greedy and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 466
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
PS D:\codes\fall25\ai\lab6>
```

Submission and timeline:

1. Submit complete implementation folder for lab 5 containing activity 1, 2 and 3 on LMS (that includes all the files in .py) on 19 Oct 2025.
2. Submit the below table in word file on LMS on 19 Oct 2025 11:59PM.

	Uniform Cost Search			A*			Greedy Search		
	#nodes explored	**Solution length**	**Is it optimal?**	**#nodes explored**	**Solution length**	**Is it optimal?**	**#nodes explored**	**Solution length**	**Is it optimal?**
Tiny Maze	15	5	yes	14	8	yes	8	8	yes
Medium Maze	269	68	yes	221	68	yes	78	74	no
Big Maze	620	210	yes	549	210	yes	466	210	yes