UNIVERSITY OF ARKANSAS

CSCE 46103/56103: Introduction to Artificial Intelligence

Homework #2

Submission Deadline: 11:59 PM, September 25th, 2024

Submission Instruction

Read all the instructions below carefully before you start working on the homework and before submission.

Homework Overview

This homework contains 4 sets of questions, from Question 1 to Question 4. You are given a PDF file **AI_Homework2.pdf** (this one) containing the submission instructions and the questions from 1 to 3.

Additionally, you are provided with two Python notebooks:

- **AI_Homework2_Question2.ipynb**, which contains the programming question for Question 2.
- **AI_Homework2_Question3.ipynb**, which contains the programming question for Question 3.

What to Submit

You are required to submit the following files:

- A single PDF file containing your solutions to all questions, including your name.
- The Python notebook **AI_Homework2_Question2.ipynb** containing your code for Question 2.
- The Python notebook **AI_Homework2_Question3.ipynb** containing your code for Question 3.

- **Question4** a zip file with your source code implementation.

Submission Format

Place all the files to be submitted in a single folder named in the format:

```
{LASTNAME}_{FIRSTNAME}_homework2
```

Ensure that the folder includes all necessary files before submitting.

Collaboration

You may discuss the homework with your classmates; however, you must not copy each other's work. Each student is expected to submit their own original solutions.

Question 1: Informed Search [20 pts]

For this homework, the objective is to find the shortest path from state **S** to state **G** in Figure 1. Note that if there is a tie between nodes with the same heuristic value, **alphabetical order** is the default order.

Answer the following questions based on the graph provided:

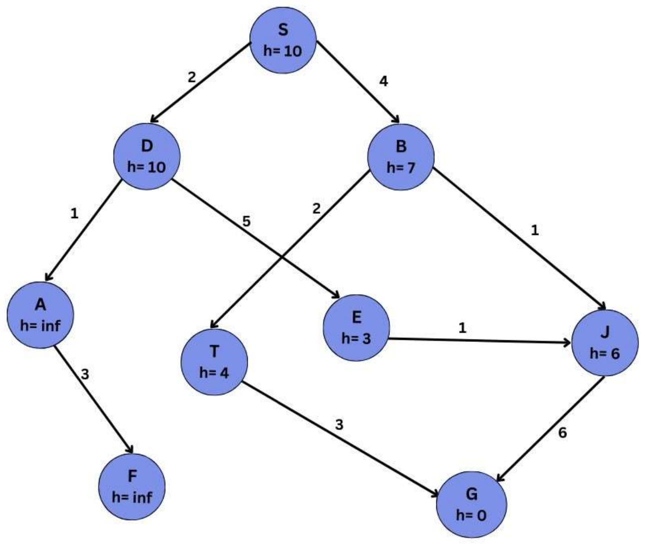

Figure 1: Graph with 9 states. The heuristic value is provided under the name of each state.

(a) Greedy Search:

[(a.1)] [2.5 pts] What is the path returned by Greedy Search?

S -> B -> T -> G

[(a.2)] [2.5 pts] What is the node expansion order for Greedy Search?

(S: 10) -> (B,S | 7), (D,S | 10) -> (T,B,S|4), (J,B,S|6), (D,S|10) -> (G,T,B,S:0), (J,B,S|6), (D,S|10)

(b) A* Search:

[(b.1)][2.5 pts] What is the path returned by A* Search

S -> B -> T -> G

[(b.2)] [2.5 pts] What is the node expansion order for A* Search

(S|10) -> (B,S|11), (D,S|12) -> (T,B,S|6), (J,B,S|7), (D,S|12) -> (G,T,B,S|3)

(c) Heuristic Modification: [5 pts]

Are the heuristic values provided in Figure 1 admissible. Include your explaination for each value wherether it is admissible.

If they are not admissible, modify the values to make them admissible, ensuring that the heuristic never overestimates the true cost to reach the goal.

They are NOT admissible. An example would be the S node as the true cost is 9 while the heuristic value is 10.

Modified:

S = 9

D = 12

B = 5

A = Inf

T = 3

E = 7

J = 6

F = inf

G = 0

(d) (5 pts)A* Search with Admissible Heuristic:

Using the modified heuristic values from part (c), run A* Search again on the new graph.

[(d.1)] [2.5 pts] What is the path returned by A* Search?

S->B->T->G

[(d.2)] [2.5 pts] What is the node expansion order for A* Search?

(S|9) -> (B,S|9), (D,S|12) -> (T,B,S|5), (J,B,S|7), (D,S|12) -> (G,T,B,S|3)

Problem 2: Programming Informed Search [20 pts]

Implement **Greedy Search** and **A* Search** in Python on the graph (with admissible heuristic) given in Problem 1. Please refer to `AI_Homework2_Question2.ipynb` for more details.

Problem 3: Sudoku Solver [30 pts]

In this assignment, you will create a program using Constraint Satisfaction Problems (CSP) techniques to solve the famous Sudoku problem. An example of Sudoku is shown in Figure 2. Please refer to `AI_Homework2_Question3.ipynb` for more details.

	1	2	3	4	5	6	7	8	9
A	3			8		1			2
B	2		1		3		6		4
C				2		4			
D	8		9				1		6
E		6			1			5	
F	7		2				4		9
G				5		9			
H	9		4		8		7		5
I	6			1		7			3

	1	2	3	4	5	6	7	8	9
A	3	4	6	8	9	1	5	7	2
B	2	9	1	7	3	5	6	8	4
C	5	7	8	2	6	4	3	9	1
D	8	5	9	4	7	3	1	2	6
E	4	6	3	9	1	2	8	5	7
F	7	1	2	6	5	8	4	3	9
G	1	3	7	5	4	9	2	6	8
H	9	2	4	3	8	6	7	1	5
I	6	8	5	1	2	7	9	4	3

Figure 2: Sudoku puzzle and its solution.

Question (4): Pacman Problem

In this pacman problem, you will implement an algorithm that teaches the Pacman agent to find the path in the maze to collect food. You are given a map, a start position, and a food position. Your task is to implement a search algorithm to find a path from the start position to the food position.

Provided Files

The code for this assignment includes multiple Python files. The purpose of each file is described below:

Filename	**Purpose**
`search.py`	You will implement your algorithm in this file.
`searchAgents.py`	This file will be used to call your implemented algorithms.
`pacman.py`	This file is the main program that runs the Pacman game.
`util.py`	This file consists of useful data structures.
`game.py`	This file defines how the Pacman is played. It describes supporting types such as `AgentState`, `Agent`, `Direction`, and `Grid`.
`graphicsDisplay.py` `graphicsUtils.py` `textDisplay.py` `ghostDisplay.py` `keyboardAgents.py` `layout.py`	These files are used to render the Pacman game. You can ignore these files.

Acknowledgement: The base code is borrowed from the Introduction to AI course of the University of California, Berkeley.

Instructions

Environment Setup

We will use Python 2.7 for this assignment. To set up a virtual environment, run the following commands:

```
virtualenv –p python2 .7 pacman_env
source pacman_env/bin/ activate
```

You can test the Pacman game and play using the arrow keys:

```
python pacman. py
```

Searching Function Format

The searching function should follow this format:

```
def search_fn(problem):
    # Your task is required to implement a searching function
    # The function returns a list of actions
```

problem is an instance of the class`PositionSearchProblem`(defined in searchagents.py). The problem instance has 3 important instance methods

- `problem.getStartState()` - Returns the start position.

- `problem.isGoalState(current)` - Returns `True` (if the current position contains food.)
- `problem.getSuccessors(current)` - Returns a list of successors of current position.

Now, take a look on the example

```
#search.py def
tinyMazeSearch(problem): from game
import Directions s =
Directions.SOUTH w =
Directions.WEST return [s, s, w,
s, w, w, s, w]
```

- The tinyMazeSearch function will return a fixed path. s and w represent for the action going the South and going to the West. You will have for basic actions: North, South, West, East.
- To test this searching function, we will run the Pacman game by the command as follows.

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

Implement search algorithms (30 points)

Note: You can use the Queue, Priority Queue, and Stack that have already been implemented in the `util.py`

a. Depth First Search (DFS) - 10 points

You are required to implement a depth-first search function in the search.py script that leads Pacman to find his food.

```
def depthFirstSearch(problem):
    "***␣YOUR␣CODE␣HERE␣***"
```

You can test your depth-first search in 3 mazes: a tiny maze, a medium maze, and a big maze.

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
python pacman.py -l bigMaze -p SearchAgent -a fn=dfs
```

b. Breadth First Search (BFS) - 10 points

You are required to implement a breadth-first search function in the search.py script that leads Pacman finds his food

```
def breadthFirstSearch(problem):
    "***␣YOUR␣CODE␣HERE␣***"
```

You can test your breadth-first search in 3 mazes: a tiny maze, a medium maze and a big maze

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs
```

c. Uniform Cost Search (UCS) - 10 points

You are required to implement a uniform cost search function in the search.py script that leads Pacman finds his food

```
def uniformCostSearch(problem):
    "***␣YOUR␣CODE␣HERE␣***"
```

You can test your uniform cost search in 3 mazes: a tiny maze, a medium maze and a big maze.

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