CS 214: Artificial Intelligence Lab

Common Instructions: You have to submit a code without any syntax errors. Don't copy from others; write your own piece of code with comments as much as possible. Try to Run with the test cases before submitting the code.

Assignment 1

NOTE:

- Download *search.zip*, unzip and rename it with your group number as "group_2" for example.
- Test your code and submit it on Moodle (`https://moodle.iitdh.ac.in/user/index.php?id=992`).
- Deadline for submission is January 17 2024 by (11:59 PM)
- We will run a plagiarism check for all the submissions; if found penalty will be applied

Lab Instructions:

In this project, your Pacman agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. You will build general search algorithms and apply them to Pacman scenarios.

This project includes an autograder for you to grade your answers on your machine. This can be run with the command: ***python autograder.py***

The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment and some of which you can ignore. You can download all the code and supporting files as a zip archive.

File to edit:

- **search.py:** where you will have the functions for the search algorithms. you have to complete these functions in order to implement the algorithms.

Can look at files:

- **Pacman.py:** The main file that runs Pacman game. This file describes a Pacman GameState type, which you use in this project.
- **game.py:** This file describes several supporting types like AgentState, Agent, Direction, and Grid.
- **util.py:** Useful data structures for implementing search algorithms.

You can ignore all other supporting files

Files to Edit and Submit:
You will fill in portions of **search.py** and **searchAgents.py** during the assignment. You should submit these files with complete code and comments. Please do not modify other files in this distribution.

Evaluation:
Your code will be autograded for technical correctness. **Please do not change the names of any provided functions** or classes within the code, or your code can not be evaluated. However, the

marks will be awarded based on the judgment of autograder and manual evaluation.

The Pacman Game:

After downloading the code (search.zip), unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line:

python pacman.py

Pacman has to navigate the corridors and get the treats efficiently. The simplest agent in searchAgents.py is the GoWestAgent, which always goes to the west and gets the treats. This agent can occasionally win.Run the following command to see how GoWestAgent works

python pacman.py –layout testMaze –pacman GoWestAgent

But this agent can't turn back if there are treats other side. check using the command:

python pacman.py –layout tinyMaze –pacman GoWestAgent

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal.

Question 1: Finding a Fixed Food Dot using Depth First Search

The treat will be at a fixed place on the maze. You will write the code to navigate the Pacman through the corridors to the treat. For which you need to Implement the **depth-first search (DFS)** algorithm in the **depthFirstSearch** function in *search.py*. To make your algorithm complete, write the graph search version of DFS, which avoids expanding any already visited states.

First, test that the SearchAgent is working correctly by running:

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

The command above tells SearchAgent to use tinyMazeSearch as its search algorithm, which is implemented in *search.py*. Pacman should navigate the maze successfully.

After implementing the **depthFirstSearch** function, your code should quickly find the solution for:

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 -z .5 -p SearchAgent -a fn=dfs

Hint: If you use a Stack as your data structure, the solution found by your DFS algorithm for mediumMaze should have a length of 130 (provided you push successors onto the fringe in the order provided by getSuccessors; you might get 246 if you push them in the reverse order).

Question 2: Finding a Fixed Food Dot using Breadth First Search

The treat will be at a fixed place on the maze. You will write the code to navigate the Pacman through the corridors to the treat. For which you need to Implement the **breadth-first search (BFS)** algorithm in the **breadthFirstSearch** function in *search.py*. To make your algorithm complete, write the graph search version of BFS, which avoids expanding any already visited states.

After implementing the **breadthFirstSearch** function, Your code should quickly find the solution for:

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs

Hint: Use a Queue as your data structure and compare the solution cost you get in BFS with the ones you get in DFS to know which algorithm gives the least cost solution.