Project 1

Submission requirements

- Submit one file to Carmen: **`search.py`**
- Expected time commitment: 4-8 hrs
- Due: Friday, Sept. 7, 5:00PM

Directions

Download this ZIP file with the Pacman source code: search.zip. This code, and the idea for the assignment, comes from UC Berkeley.

- Unzip the code.
- Open up the Windows Command Line or Mac Terminal or Linux Terminal.
- Change your directory to the folder with the pacman code. You should see a file called `commands.txt` and two folders: `layouts` and `py`.
- Run some of these commands (as listed in `commands.txt`) to make sure your setup works.
    - `python pacman.py` (use your arrow keys to control pacman)
    - `python pacman.py --layout tinyMaze --pacman GoWestAgent`
- You can ignore most of the files in the folder, but you might want to look at these:
    - `pacman.py` — The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.
    - `game.py` — The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
    - `util.py` — Useful data structures for implementing search algorithms.
- Take a look at the generic search algorithm outlined on my website. This should give you a good idea of how to implement each of these search problems. The only thing that really needs to change is the particular data structure that you use (remember that these are implemented in `util.py`).
- Although we are only completing the first 4 problems, you can run the autograder to check your work using the following command:
    - `python autograder.py`
- Once completed, submit just the file `search.py` to carmen.

Task 1 (3 pts)

Open the file `py/search.py` and find the function `depthFirstSearch` (line 70). Take this template and finish the code so that depth-first search works. Test with:

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

Task 2 (2 pts)

(Note that this should be simple if you've completed task 1....)

Implement breadth-first search for pacman, in the `breadthFirstSearch` function. Test with:

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

Task 3 (3 pts)

Open the file py/search.py and find the function uniformCostSearch which reads:

Take the template and finish the code so that UCS works. (You might want to adapt your implementation of DFS or BFS.) Test your code with:

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

Task 4 (2 pts)

Finish the implementation of A* search. You can use the parameter called “heuristic” as a function like so:

```
dist = heuristic(state, problem)
```

You can test it with pacman by running this command:

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

Helpful Code:

```
def depthFirstSearch(problem):
    """
    Search the deepest nodes in the search tree first [p 85].

    Your search algorithm needs to return a list of actions that reaches
    the goal.  Make sure to implement a graph search algorithm [Fig. 3.7].

    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())
    """
    closedset = []
    openset = [problem.getStartState()] # openset starts with starting state
    parents = {}
    while len(openset) > 0:
      state = openset.pop() # get most-recently-added element from openset

      # ...

      if # ...
          print "Found goal!"

          # retrieve series of steps that brought us here (use the parents map)
          actions = []
          while state != problem.getStartState():
            # ...

          print actions # just to see the resulting actions
          return actions
      else:
          for (next_state, action, cost) in problem.getSuccessors(state):
            # next_state is something like (4, 2) (coordinates)
            # action is something like WEST
            # cost is not used for depth-first search
            # ...
```