COMP 6721 Applied Artificial Intelligence (Fall 2023)

Lab Exercise #2: State Space Search

Question 1 Once upon a time a farmer went to the market and purchased a fox, a goose, and a bag of beans. On his way home, the farmer came to the bank of a river and rented a boat. But in crossing the river by boat, the farmer could carry only himself and a single one of his purchases – the fox, the goose, or the bag of the beans.

If left alone, the fox would eat the goose, and the goose would eat the beans. The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact.

Represent this problem as a search problem. Choose a representation for the problem's states and:

(a) Write down the initial state

(b) Write down the goal state

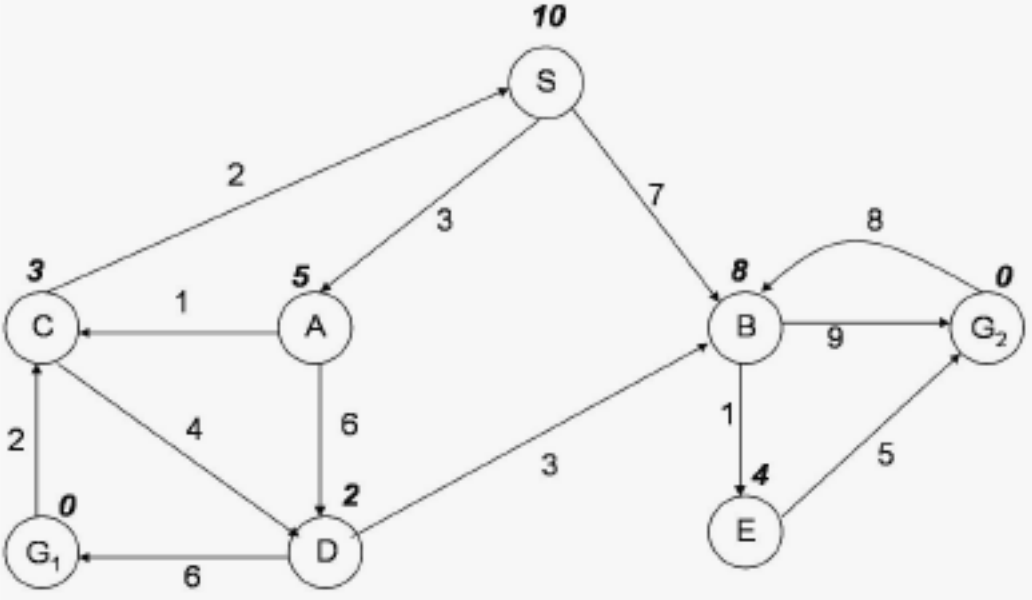
(c) Write down all illegal states

(d) Write down the possible actions

(e) Draw the state space for this problem

(f) Find a series of moves to solve this problem

Question 2 Assume that S is the initial state and G_1 and G_2 are the goal states. The possible actions between states are indicated by arrows. The number labelling each arc is the actual cost of the action. For example, the cost of going from S to A is 3. The number in bold italic near each state is the value of the heuristic function h at that state. For example, the value of h at state C is 3. When all else is equal, expand states in alphabetical order.



For the following search strategies, show the states visited, along with the open and closed lists at each step (where it applies).

(a) Breadth-first search

(b) Depth-first search

(c) Iterative deepening depth-first search

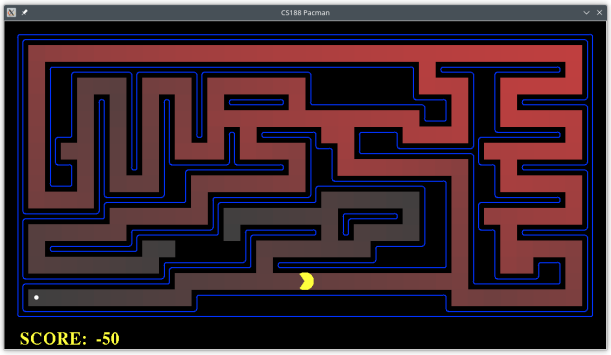
(d) Uniform cost search

(e) Hill climbing

(f) Best-first search

(g) Algorithm A

Question 3 *Help Pacman to find his food!*[1] In this question, your goal is to implement the depth-first search and breadth-first search algorithms shown in the lecture. To make it more interesting, we'll use an existing framework where you write generic search algorithms and run them in a Pacman simulator environment.



For the source code download and a list of files to edit, please refer to the supplemental lab page on Moodle.

(a) Finding a Fixed Food Dot using Depth First Search.
In the file `search.py`, find the function `depthFirstSearch`:

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

Your search algorithm needs to return a list of actions that reaches
the goal. Make sure to implement a graph search algorithm.
...
"*** YOUR CODE HERE ***"
```

One of the first things you'll probably want to do is to check if Pacman found the goal state:

```
if(problem.isGoalState(problem.getStartState())):
   return []
```

Remember that DFS uses a *stack* data structure. Make sure you use the data structure that come with the provided `util.py`; you can just use:

```
open = util.Stack()
```

and use the stack functions as you'd expect (`push, pop` etc.).

Read the code carefully to understand the data your function must return: Pacman expects a list of *actions* to perform (move left, right, up, down):

```
return actions
```

[1] Based on UC Berkeley's Pacman AI project, http://ai.berkeley.edu

You'll find possible actions when you create successor states from each state; use the commented out example debug code:

```
print("Start's successors:", problem.getSuccessors(problem.getStartState()))
```

to see these; thus, a legal result would be a list of actions like `['West', 'South', 'South', 'East', 'West']`. Make sure you return the *solution path* here (path from start to goal), not the *search path* (including all the states you visited while searching). Make sure you read the note on the lab page regarding where to put the *loop check* in your implementation.

When you're done, you can test your implementation on the three mazes shown on the Moodle page. You should also run the unit tests to check your code for correctness using:

```
python autograder.py -q q1
```

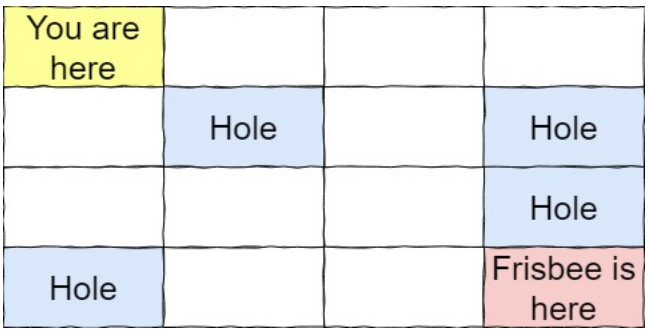
If everything is correct, you'll receive a score of 3/3 (this score is just fyi, there is no submission for this lab exercise).

(b) Breadth First Search.
Like above, but now we search with BFS. Remember that the only difference to DFS is the data structure used to manage the open list (fringe). Test your code using `python autograder.py -q q2`

(c) Solving the 8-Puzzle.
But what about the 8-puzzle we discussed in the lecture? If you implemented your search algorithms correctly, they don't include anything specific about Pacman, so you can run them with the provided 8-puzzle code `python eightpuzzle.py` to find the solution for a random puzzle:

```
(comp6721) :~/COMP6721/Pacman/search> python eightpuzzle.py
A random puzzle:
-------------
| 5 | 4 | 1 |
-------------
| 3 | 7 | 2 |
-------------
| 6 |   | 8 |
-------------
BFS found a path of 11 moves: ['up', 'up', 'left', 'down', 'right',
  'up', 'right', 'down', 'left', 'left', 'up']
After 1 move: up
-------------
| 5 | 4 | 1 |
-------------
| 3 |   | 2 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
```

Question 4 *Bonus take-home exercise from OpenAI:*[2] Winter is here. You and your friends were tossing around a Frisbee at the park when you made a wild throw that left the Frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international Frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc as soon as you can. The surface is described using a rectangular grid like the figure below:



(a) What are the initial and goal state?

(b) What are the illegal states?

(c) What are the possible actions and what should be their costs?

(d) Draw the state space for this problem.

(e) Find a series of moves to solve this problem.

(f) Perform the following search strategies, show the states visited, along with the open and closed lists at each step.

I Breadth-first search

II Uniform cost search

III Depth-first search

[2] https://gym.openai.com/envs/FrozenLake-v0/