Introduction to
Artificial Intelligence and Machine Learning
Homework 1 - Search

2018.9.26

Question 1~4 ~ Graph Search

- Motivation: By abstracting the problems, we can solve them using this **problem-independent** method.
- To abstract a problem, we define

  - the initial state

  - the goal state

  - the successors (or children) of each state

  - (optional) the cost of each action (i.e. state transition) of the problem.

Question 1~4 ~ Graph Search

- It is recommended that you begin with a function "graphSearch" defined by yourself
- def graphSearch(problem, search):

```
GRAPH-SEARCH(problem)

initialize the frontier using the initial state of problem
initialize the explored set to be empty
repeat
    if the frontier is empty
        return failure
    choose a leaf node and remove it from the frontier.
    if the node contains a goal state
        return the corresponding solution
    add the node to the explored set
    expand the chosen node
    if (not in the frontier) or (explored set)
        add the resulting nodes to the frontier
```

Question 1~4 ~ Graph Search

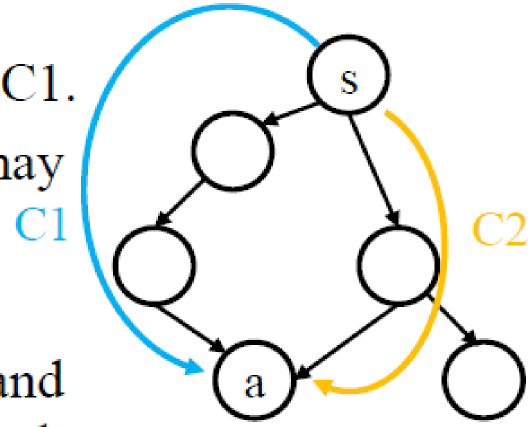
- **graphSearch(problem, search):(in search.py)**
  - Frontier.push(problem.getStartState())
  - node = problem.getStartState()
  - return …(hint: return a list)

GRAPH-SEARCH(*problem*)

```
initialize the frontier using the initial state of problem
initialize the explored set to be empty
repeat
    if the frontier is empty
        return failure
    choose a leaf node and remove it from the frontier.
    if the node contains a goal state
        return the corresponding solution
    add the node to the explored set
    expand the chosen node
    if (not in the frontier) or (explored set )
        add the resulting nodes to the frontier
```

Question 1~4 ~ Graph Search

- Question 1: DFS – Stack
- Question 2: BFS – Queue
- Question 3: UCS – Priority Queue, Tricky!
  - The first time when node “a” is explored, the cost is C1.
  - Since we are considering graph search, we may encounter “a” second time, with cost C2.
  - What if C2<C1?
  - My method: A dict which remembers states and corresponding min cost. When lower cost confirmed, update the dict and push the node with new cost in.
  - Or define your PriorityQueue in ***search.py***
  - Or any method you like.
- Question 4: A* – with heuristic



Map	
State	Cost
…	…
a	~~C1~~ C2
…	…

Question 1~4 ~ Graph Search

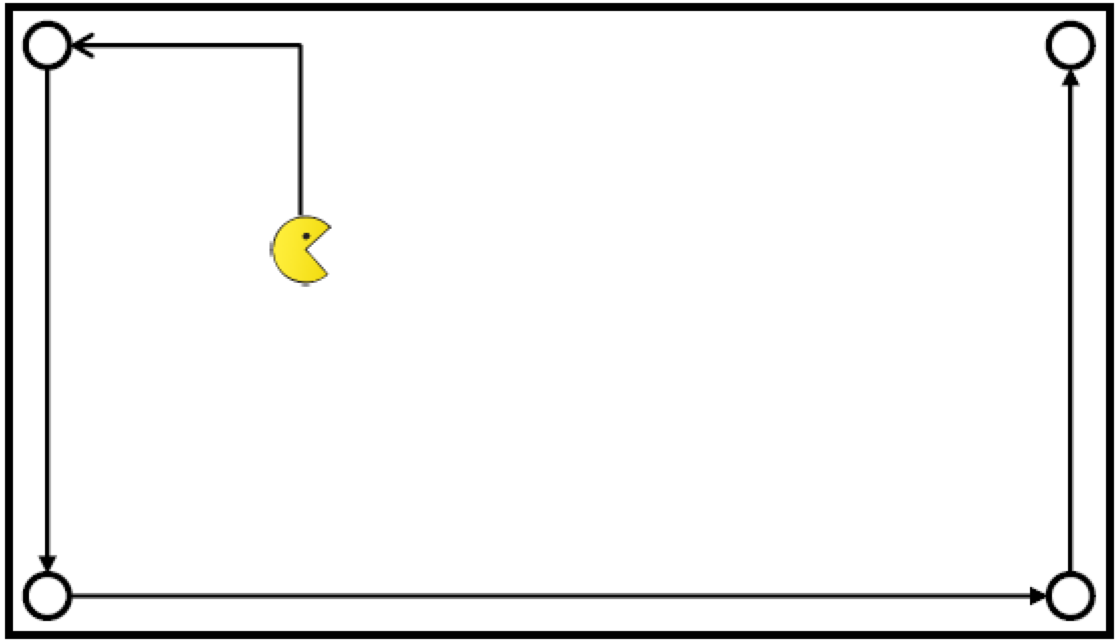
- Reminder: Please make sure your graph search program is ***problem-independent***!
- –By testing “python eightpuzzle.py”

Question 5 – Defining Game States

- *searchAgents.py*
- Objective: Abstract the Corner Problem
  - There are four foods at each corner at the beginning.
  - Once the Pacman wanders to a corner with food, the food will be eaten and no longer exist.
  - The goal is the Pacman eating up all foods, which declares the end of the game.
- You don’t need to change codes after line 459 in *searchAgents.py*

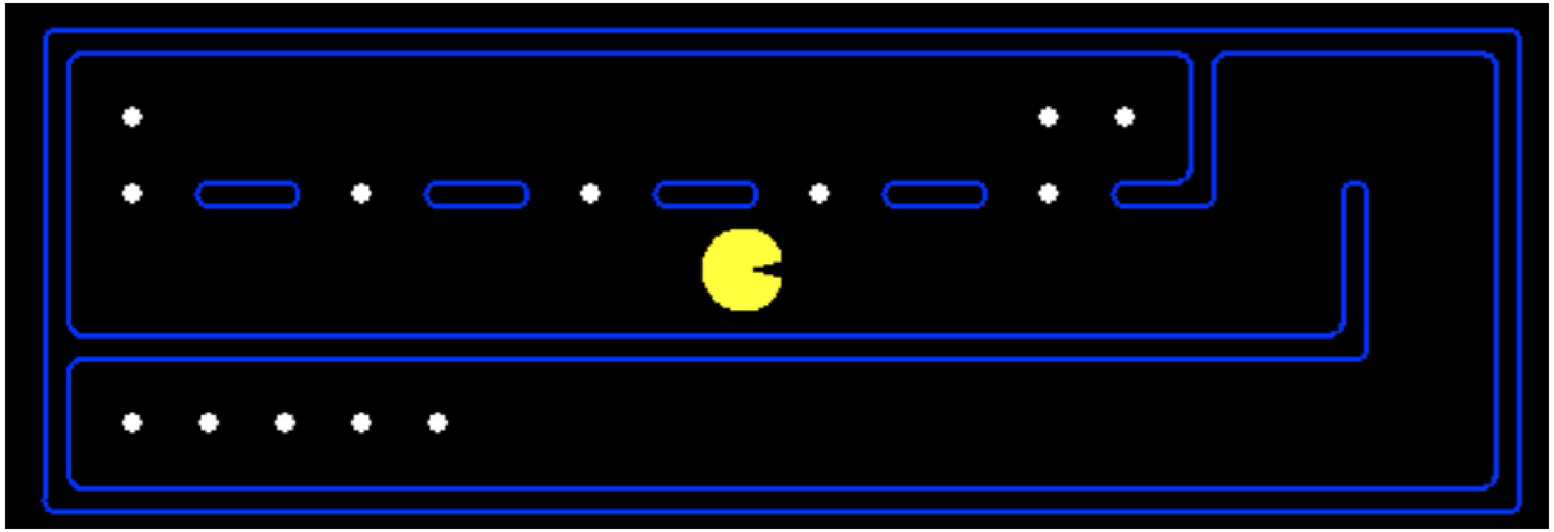
Question 6~7 – Designing a Heuristic

- Reminder: Design a consistent heuristic for the Corner Problem ***for all possible game states***.
- Question: Is min{manh(p, $c1$), manh(p, $c2$), manh(p, $c3$), manh(p, $c4$)}+manh($c1$, $c2$)+manh($c2$, $c3$)+manh($c3$, $c4$) consistent?



Question 6~7 – Designing a Heuristic

- For question 7, you may search the closest food first.
- However, if you only use it as your heuristic, you cost will be higher after you eat a food than not.
- mazeDistance(point1, point2, gameState) may be useful
- You may focus on this “tricky” search.



Deadline

- 2018/10/17 27:00 (2018/10/18 03:00)
- Allow late submission until 2018/10/24 27:00 (2018/10/25 03:00)