Assignment 2

AI 2016 – Minor Robotica

- **It will be graded.**
- **Execute in groups of one or two persons.**
- **Create one PDF per group with the answers to the questions on paper and include a short but clear explanation of what you programmed.**
- **You will only alter the search.py file**
- **Combine the PDF + your altered search.py code file in one ZIP file with the name:**
  AI2015 BFS DFS lastname1-studentnr1 lastname2-studentnr2.zip
  so e.g.:
  AI2015 BFS DFS deKok-500223133 Stolk-500131323.zip
- **Upload the zip-file to the DLWO-Dropbox of this course.**
  Other naming conventions will not be graded.
- **More info on the Python Pacman problem:**
  http://inst.eecs.berkeley.edu/~cs188/pacman/search.html

Note: the assignment works with Python 2.7!

You can use run.bat to execute all the different search runs. To disable certain parts, you can edit run.bat and place *REM* before the line(s) you want to disable.

Assignments:

You will implement Breadth Frist Search (BFS) and Depth First Search (DFS) strategies for both the 8puzzle as the Pacman route problem. The BFS function will work with both the 8-puzzle and the Pacman maze solver, DFS only for the Pacman maze solver.

Assignment 1 (6pt)

1a) Is cycle detection required when using:

- **BFS in solving the 8-puzzle problem?**
- **DFS in solving the 8-puzzle problem?**
- **BFS in solving the pacman route planning?**
- **DFS in solving the pacman route planning?**

Motivate your answers.

1b) Is cycle detection more important with BFS or DFS, or just as important,

- **when solving the 8-puzzle problem?**
- **when solving the pacman route planning?**

Motivate your answers.

Run:

```
python pacman.py --layout mediumMaze --pacman GoWestAgent
```

(e.g. by running run.bat)

Let's presume that:

- *Pacman starts from the top-right corner*
- *The goal is in the bottom-left corner*

All the following questions have to be answered as if the specified strategy is executed on the **`mediumMaze`** map.

1c) Explain how the search pattern would look like if you would implement a Breadth First Search to guide Pacman from the start to the goal in the `mediumMaze` map. Which nodes would be visited when and why?

1d) Would BFS always find the shortest route to the finish? Motivate your answer.

1e) Will BFS's search process will come across every possible position on the board before terminating? If so, why? If not, what nodes will not be part of the BFS' search?

1f) Do an estimation about the maximum number of open nodes at the frontier (open list) during the route. How did you come up with this number?

1g) Implement the `breadthFirstSearch` function in the `search.py` file. Make sure the BFS function works with both the eightpuzzle and the Pacman problem solver.

These commands should work now:

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

1h) Did your search algorithm find the route you were expecting? Motivate your answer.

Assignment 2 (4pt):

Run again:

```
python pacman.py --layout mediumMaze --pacman GoWestAgent
```

Let's presume that:

- *Pacman starts from the top-right corner*
- *The goal is in the bottom-left corner*

2a) Explain how the search pattern would look like if you would implement a Depth First Search to guide Pacman from the start to the goal. Which nodes would be visited when and why?

2b) Program the `depthFirstSearch` function in the `search.py` file.

These python commands should work now:

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

2c) Does the order in which the successor nodes (children) are provided by the function `getSuccessorNodes` make a large difference?

- **Can you speed up the search process by resorting them?**
- **What does this do to the route?**
- **Is this fair? Why (not)?**

Some pointers:

Remove the line `util.raiseNotDefined()` when you implemented the return value of your function.

A wise strategy is to first let it print the input to the search function and get to understand some basic provided functions by adding the lines in the `breadthFirstSearch` function:

```
print "Start:", problem.getStartState()
print "Is the start a goal?", problem.isGoalState(problem.getStartState())
print "Start's successors:", problem.getSuccessors(problem.getStartState())
```

As you can see, the call of `problem.getSuccessors(someState)` will provide you with the details you will have to use in your search routine.

For `eightpuzzle.py` it will return a list of child nodes with sets of:

```
(<state>, <action>, <costs>)

[(<object of possible next puzzle state 1>, <the action to get to
next state 1>, <the costs to take that step to 1>),
(<object of possible next puzzle state 2>, <the action to get to next
state 2>, <the costs to take that step to 2>)]
```

State can be any object or some list of variables and should not be analyzed by your program. You will just need to store them (entirely) in queues, stacks or whatever and compare them as a whole. Also you can just give it as an argument to the functions `problem.isGoalState(state)` and `problem.getSuccessors(state)`.
So, this would be possible:

```
X = problem.getStartState()
Y = problem.getSuccessors(X)          # Get all successors from the start state
if problem.isGoalState(Y[0][0]):      # Check if the first successor of the
```

```
                                                    # start state is the
goal
                                    # print "THIS IS THE GOAL STATE"
    return Y[0][1]                  # return the action that should be
                                    # taken by the main program to
                                    # get to the goal
```

Of course this will only work when there is one step to be taken and the first child of the start node is indeed the goal, but it gets you started in terms of syntax.

So, one successor will look like this: `(<__main__.EightPuzzleState instance at 0x02411760>, 'down', 1)`. As stated above you can you can just ignore the type of the first item (state). You can store objects just like any other value (e.g. an integer). So, states for the 8-puzzle and states for the Pacman problem can be handled just the same way.

For `pacman.py`, the `getSuccessors` function will return something like:

```
result = [((34, 15), 'South', 1), ((33, 16), 'West', 1)]
```

You can get certain elements from this returned value by:

```
result[0] = ((34, 15), 'South', 1)
result[1] = ((33, 16), 'West', 1)
result[1][0] = (33, 16) # is the state part of the second child
result[1][1] = 'West'   # the action that gets you from the parent of this
                        # node to this node
result[1][0][1]= 16     # Do not do this: the state should be kept as a
                        # whole! Just remember that, to keep your BFS/DFS
                        # function compatible with both the Pacman problem
                        # as the 8-puzzle problem, you should not interpret
                        # the states differently.
```

You can store composite objects as one variable by surrounding them with brackets. This may become handy when you want to save extra data about some state during the search process (maybe the entire list of actions that got you there?)

```
X = 1
Y = 2
Z = (X, Y)
Z     # will be (1, 2)
Z[0]  # will be (1)
Z[1]  # will be (2)
```

A convenient way of working with FIFOs (queue) and LIFOs (stack) in python is by using lists:

```
list = []                 # Creating an empty list
list.append(something)    # Add the variable (or object) something to
                          # the end of the list
```

```
list.insert(0, something) # Inserts something at the beginning of the
                          # list
list.pop()                # Gets the last variable from the list and
                          # removes it from the list
list.pop(0)               # Gets the first variable from the list and
                          # removes it from the list
list.remove(something)    # Remove the first item in the list whose
                          # value is equal to something
```

Note that you can make queue or stack behaviour by choosing the right combinations of `append`/`insert` and `pop(…)`.

The output of your search function should give something back in the form of a list of actions. By letting the `breadthFirstSearch` function seek trough the possible nodes in a BFS manner, it can find an optimal sequence of actions to reach the desired goal. In the case of the 8-puzzle, the output will look something like:

```
('down', 'down', 'left', 'up')
```

For pacman.py it would be something like:

```
('South', 'South', 'West', 'West')
```

Of course these are sequence of actions that your search got from the actions in the `problem.getSuccessors(state)` function.