CS 441/ ECE 530 Artificial Intelligence
Assignment 1

Deadline: 24th of October, 5pm
How to submit: send an email to serhan.aksoy@antalya.edu.tr

In this assignment you will create an agent to solve the 8-**puzzle** game. You may visit **mypuzzle.org/sliding** for a refresher of the rules of the game. You will implement and compare several search algorithms and collect some statistics related to their performances. Please read all sections of the instructions carefully:

An instance of the 8-puzzle game consists of a **board** holding N = 8 distinct movable tiles, plus an empty space. The tiles are numbers from the set {1, 2, …, 8}. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, we will represent the blank space with the number 0.

Given an initial **state** of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order:

```
1 2 3
4 5 6
7 8 0
```

The search space is the set of all possible states reachable from the initial state. The cost of moving from one configuration of the board to another is the same and equal to one. Thus, the total cost of path is equal to the number of moves made from the initial state to the goal state.

We provide you with a baseline code 8pzHw.py. Please first have a look at the functions.

Algorithm Review

Recall from the lectures that searches begin by visiting the root node of the search tree, given by the initial state. Among other book-keeping details, three major things happen in sequence in order to visit a node:

- First, we **remove** a node from the frontier set.
- Second, we **check** the state against the goal state to determine if a solution has been found.
- Finally, if the result of the check is negative, we then **expand** the node. To expand a given node, we generate successor nodes adjacent to the current node (generateMoves function already does this), and add them to the frontier set. Note that if these successor nodes are already in the frontier, or have already been visited, then they

should not be added to the frontier again (i.e., you need to implement graph search not tree search).

This describes the life-cycle of a visit, and is the basic order of operations for search agents in this assignment—(1) remove, (2) check, and (3) expand. In this assignment, we will implement algorithms as described here**. Note that the breadth first implementation that I showed you during the lecture applies the goal test when a node is generated rather than expanded. In this assignment, to comply with the other search algorithms, do not apply the goal test when you generate a node. Instead, you should apply it when you expand a node (i.e. remove the node from the frontier).**

What you need to submit
Your job in this assignment is to complete 8pzHw.py, which solves any 8-puzzle board when given an arbitrary starting configuration. You need to implement three search algorithms: Breadth-First Search, Depth-First Search, A-Star Search.

When executed, your program should create / write to a file called `output.txt`, containing the following statistics:
`path_to_goal`: the sequence of moves taken to reach the goal
(the direction variable of the puzzle object would be useful for recovering path_to_goal)
`cost_of_path`: the number of moves taken to reach the goal
`nodes_expanded`: the number of nodes that have been expanded
`search_depth`: the depth within the search tree when the goal node is found
`max_search_depth`: the maximum depth of the search tree in the lifetime of the algorithm

For example, if we run BFS from the initial state:

```
1 2 5
3 4 0
6 7 8
```

The output file should contain the following lines:
path_to_goal: ['up', 'left', 'left']
cost_of_path: 3
nodes_expanded: 10
search_depth: 3
max_search_depth: 4

The path would look like this:

```
1 2 5
3 4 0 (start)
6 7 8

1 2 0
3 4 5 (after up)
```

```
6 7 8

1 0 2
3 4 5 (after left)
6 7 8

0 1 2
3 4 5 (after left)
6 7 8
```

Similarly if we run DFS from the same initial state

The output file should contain the following lines:

path_to_goal: ['Up', 'Left', 'Left']
cost_of_path: 3
nodes_expanded: 181437
search_depth: 3
max_search_depth: 66125

For `Astarsearch` you need to implement the `h` function which takes as input a puzzle object and returns an integer which is an estimate of the cost of the path from the state of the puzzle object to the goal state. For undergraduate students a single heuristic function is enough. For MSc students, three heuristic functions must be implemented.

References:
CS 188 course in UC Berkeley
CSMM.101x Artificial Intelligence course in Columbia University.