CMPE540 – 2020-2021 - HW1: Search in some sliding tiles puzzle
Due: 26.04.2021, 23:59
Via moodle (repeated submission is not allowed – submit once)
Late submission policy: Grade-10*day^2

You will implement single-agent basic search strategies in this homework. The domain is sliding tiles domain, however, different from a standard n-tiles problem:

- There might be multiple empty cells. An empty cell is represented by dot (**.**)
- The numbers on the tiles are arbitrary (integer numbers, not necessarily consecutive, might be negative)
- There might be fixed tiles, represented by **x**
- Action cost corresponds to the integer value of the tile being moved.

An example initial puzzle configuration is as follows:

```
1 2 -1
. x -2
. .  .
```

An example goal puzzle configuration is as follows:

```
. .  .
. x -2
2 .  .
```

- There are several initial and goal configuration example files (and some solutions) attached.
- You can assume nxn square board configurations
- The cell contents are separated by white space (spaces/tabs in the same row; and newline for the next row). There won't be any surprise, make sure you can process the example input files.
- Note that goal configuration does not necessarily need to include all tile numbers. In the above example, the goal is satisfied when tiles 2 and -2 are moved to the indicated positions.
- You can assume that all problems have solutions (set recursion depth limit to 10K):
  `sys.setrecursionlimit(10000)`

I will run your program with the following command:

```
python3.9 id.py <search-type> <init-file> <goal-file> <soln-file>
```

where

- <search-type> might be bfs or dfs or ucs or astar0 or astar1 or my-astar-positive or my-astar-all
- <init-file> is a text file that includes initial puzzle configuration
- <goal-file> is a text file that includes goal puzzle configuration
- <soln-file> will include your solution

`<search-type>`

- bfs: you need to implement breath-first search
- dfs: you need to implement depth-first search

- ucs: you need to implement uniform-cost search
- astar0: you need to implement A* search with heuristic “number of misplaced tiles”
- astar1: you need to implement A* search with heuristic “Manhattan distance of the misplaced tiles”
- my-astar-positive: you need to implement a non-trivial consistent and admissible heuristic that assumes a board without any negatively numbered tile
- my-astar-all: you need to implement a non-trivial consistent and admissible heuristic for the negatively numbered tiles as well

Your heuristic functions will be evaluated relative to others.

`<soln-file>` has three parts:

- 1st part: the solution, i.e. the sequence of actions from the initial configuration to the goal configuration. Each action is written in a line, with the following format:
  - move <tile-number> <action-type>
  - <action-type> might be one of the following: left or right or up or down
- 2nd part: number of nodes inserted in the fringe:
  - nInsertedNodes: <number>
- 3r part: total path cost from the initial configuration to the final configurations
  - cost: <cost>

Please see the attached example solution files. I will use ‘diff’ command while automatically evaluating your homework, therefore make sure that there is no typo etc.

One example solution is as follows:

```
move 15 right
move 14 right
move 13 right
move 9 down
move 10 left
move 6 down
move 7 left
move 8 left
nInsertedNodes: 18720
cost: 82
```

Important implementation details:
Because your homeworks will be automatically graded, you should implement the following details:

- The order of expansion: In expanding a node, there will be multiple possible ordering choices for expansion. You need to check out whether each tile can move in different directions depending on whether the neighbouring cells in the movement directions are empty or not.
  - Consider the following tile order: Start from the top-left tile, then sweep the tiles in the first row one by one. After checking out all the tiles in the first row, continue with the second row. The last tile to possibly check out will be the bottom-right tile.
  - For each tile, you need to follow a specific action/movement order:
    - left right up down

- The order of removal from the fringe: DFS and BFS are straightforward. When there is a tie in the values of nodes in UCS and A*, remove first come first.
- State visitation: Do not insert states removed from the fringe into the fringe again.

Submission details: You will submit 2 files:

- <studentid>.py: your python implementation. You are allowed to use standard libraries. All search algorithms should be implemented by you.
  - The variable that stores the number of nodes inserted into the fringe should be increased in a single line in your code. You cannot change it in multiple other placed. Where it is incremented should be indicated in your report.
- <studentid>.pdf: A report that includes:
  - Whether my-astar-positive option is implemented or not. If implemented explain why it is non-trivial and why it is admissible and consistent.
  - Whether my-astar-all is implemented or not. If implemented explain why it is non-trivial and why it is admissible and consistent.

Enclosed, please see some example initial, goal files and some solution files. I run the following lines of code to obtain the solution files. There might be some other test cases. But make sure that solution file format matches with yours and there is no difference when 'diff' command is used:

```
python3 id.py bfs init_0.txt goal_0.txt soln_0_bfs.txt
python3 id.py dfs init_0.txt goal_0.txt soln_0_dfs.txt
python3 id.py ucs init_0.txt goal_0.txt soln_0_ucs.txt
python3 id.py astar0 init_0.txt goal_0.txt soln_0_astar0.txt
python3 id.py astar1 init_0.txt goal_0.txt soln_0_astar1.txt
python3 id.py bfs init_1.txt goal_1.txt soln_1_bfs.txt
python3 id.py dfs init_1.txt goal_1.txt soln_1_dfs.txt   → this took long time so I canceled
python3 id.py ucs init_1.txt goal_1.txt soln_1_ucs.txt
python3 id.py astar0 init_1.txt goal_1.txt soln_1_astar0.txt
python3 id.py astar1 init_1.txt goal_1.txt soln_1_astar1.txt
```

Important: Make sure you change the id with your own student-id. Correctly naming your python files and reports are very important!

Grading:

- All three parts (action sequence, nInsertedNodes and cost) should be correct in order to get points. If diff fails (there is difference between your solution and my solution files), you cannot receive any point from the corresponding test case.