Exercise #1-Part 1:

Solving N-Puzzle Problem Using the ID Algorithm

Attached are the following files:

1. stack.py – implementation the data structure: stack.
2. frontier.py – Implementing an ID data structure: an "extended" stack - a stack that stores states up to depth of search d. When it is empty it reloads the initial state and increases d by 1.
3. state.py – Implements an N-Puzzle.
4. search.py – Implements the search pseudocode from class.

You need to go through the attached files, understand them, and perform the following tasks:

1. Understand how search.py implements the search pseudocode taught in class.
2. Modify the data structure to save the number of states stored in the structure. As a result, the frontier will include 5 items and not 4:

]#stack, max. depth, init. state, try next level?, total items pushed]

1. Write a program that prints the maximum depth and number of states added (total items pushed) and checked (popped) as averaged from 100 runs. The output for search(2) can be:

Average depth: 1.83

Average number pushed: 6.39

Average number popped: 4.56

1. Run search (2), search (3) and search (4). What do you see?

\* Reminder from the lecture:

search(n)

f←frontier.create(state.create(n))

while not frontier.isEmpty(f) do

s←frontier.delNext(f)

if state.isTarget(s)

then return s

ns←state.getNext(s)

for i←1 to length(ns) do

frontier.insert(f,ns[i])

return null