Sliding Puzzles: Modeling

Eckel, TJHSST AI1, Fall 2022

Background & Explanation

We'll begin our study of AI with one of the simplest search algorithms – an algorithm you might have heard of before called breadth first search (BFS). But: we aren't going to learn our algorithms in the abstract; we will apply them to genuine problems. And the first step to solving any authentic problem with a program is to code a *model* of the problem. So before we learn about BFS, let's start with learning about the problem we're going to model and coding up a working representation of that problem.

Our first problem is solving sliding puzzles. Sliding puzzles are famous. (Google "Simon Tatham fifteen" for an example.) For this assignment, we will be working with square sliding puzzles of sizes 2x2, 3x3, 4x4, and 5x5.

Modeling the Board

Go ahead and try a sliding puzzle. The real-life sliding puzzle board is a grid, and each element of the grid can be a blank or a number.

Let's intelligently modify some of this while we make our model so that the data is easier to work with:

- We will use a period to represent the blank.
- To make representing the game state easier, we will use a single character to represent each tile instead of the two-digit numbers often used. Each character might be a digit from 1 to 9 or it might be a capital letter.
- Finally, it might seem counterintuitive, but we are going to use a single string to represent the gameboard, reading each row left to right and concatenating them all.

A couple of examples:

"63471.825"

"DJBAIGNE.CLFKOMH"

(Since this has 2-digit numbers, we'll use letters instead.)

You may think something like an *n* x *n* array of ints is more intuitive, but strings are *immutable* in Python, meaning every time one is modified, a new string is created. This prevents you from having to copy manually or accidentally pointing two different variables at the same instance. Debugging string manipulation is therefore *much* easier. Use a string, trust me.

If you find it difficult to think in one-dimensional indices, I recommend functions that convert back and forth between a matrix location (row, column) and a single index of that location in a state string. This is a good way to figure out how to do the conversion *generally* (not hard-coding a specific board size) and then not have to worry about it elsewhere!

File Input

On the course website, you should get the `slide_puzzle_tests.txt` file. It looks like this:

```
2 A.CB
2 .132
3 87436.152
3 .25187643
3 863.54217
4 AB.CEFGDIJKHMNOL
4 .BCDAEGHIFJLMNKO
5 ABCDEF.HIJKGMNOPLRSTUQVWX
5 FABCE.HIDJKGMNOPLRSTUQVWX
```

Note that each line contains two things:

- A number, indicating the size of the puzzle ("2" means 2x2, "3" means 3x3, etc).
- A sequence of characters containing a period representing the initial state of the puzzle. This is formatted exactly the way you should store your board state in your code, a single string of length size^2.

You've already seen this code to read in every line of a file and store each line in a list of strings.

```
with open("some_file.txt") as f:
    line_list = [line.strip() for line in f]
```

Two reminders:

- **The `.strip()` command is essential here**. It removes any leading or trailing whitespace on a string. In this case, it will remove the "\n" from each line that has one, but if the last line of the file doesn't end in "\n" it will still work without removing a random final character from the last word. If you forget, you'll end up having weird errors on input files that are missing the final "\n" or vice versa. This means you don't have to worry.
- Either way, one **absolutely necessary** guiding principle: when using any file, **you only want to read from the file ONCE.** Open it once, read it, and store the information in a data structure, then search or manipulate the data structure. Continually searching a file for information is **extremely slow** compared to searching data structures stored in RAM.

For this assignment, you'll also want **`.split()`** for the file of puzzles. The Python `split()` command splits on any whitespace by default, returning a list of substrings. This is much easier than alternatives involving finding indices and cutting piece by piece.

Modeling Tasks

1) First, make sure you can read the input and store the puzzles properly.

   Then, create a `print_puzzle` function that will take the same two pieces of information – the size of a board and its string representation – and print it nicely to the terminal as a grid. (I recommend putting an extra space in between each character horizontally; it looks nicer.)

2) Then, create a `find_goal` function that will take a board and return the goal state for that board. The goal state is always the characters in ascending order with a period in the bottom right space. (For example, the first board in the file is "`A.CB`". The goal is the characters in ascending order – "ABC" – with a period on the end, so "`ABC.`" The built in Python `sorted()` function might be of some use here.)

   By now, you should be able to read in each line in the puzzle and nicely print its starting state & goal state.

3) Now let's get the game to function. Design a `get_children` function that takes a state and returns a list of the boards one move away from that state. (Test with different size boards! Verify by hand! A small amount of annoying effort now will save a lot of pain later if this is wrong!)

   Remember that your code must work *generally*, with any size of board.

Get Your Code Ready to Turn In

When you turn your code in to me, you won't be using the filename `slide_puzzle_tests.txt` – you want to take the name of the file **on the command line**. Modify your code to do this.

Your code should open whatever file it gets, then, for each line, use the `print_puzzle` command from #1 to nicely output the state. Below that, as standard strings, print the goal state, and all of the children *of the start state*. Your output should be clearly labelled, like "`Line 0 start state:`", "`Line 0 goal state:`", "`Line 0 children:`", etc. **Carefully** compare your output with the sample run on the last page.

Specification

Submit a single Python script to the link on the course website.

This assignment is **complete** if:

- You follow the instructions on the submission form to format your submission properly.
- Your code does all of the following:
  - Accept a single **command line argument** specifying a file name. (Do not hardcode the file name!!)
  - Read puzzles from that file just like the example file – a size and board on each line.
  - Output the start state (nicely printed), the goal state (as a string), and the children of the start state (as a list of strings). See sample run on the last page.
- Runtime is effectively instantaneous.

Sample Run

```
>python.exe 2022_sp_modeling.py slide_puzzle_tests.txt
Line 0 start state:
A .
C B
Line 0 goal state: ABC.
Line 0 children: ['ABC.', '.ACB']

Line 1 start state:
. 1
3 2
Line 1 goal state: 123.
Line 1 children: ['31.2', '1.32']

Line 2 start state:
A B C
D E F
G . H
Line 2 goal state: ABCDEFGH.
Line 2 children: ['ABCD.FGEH', 'ABCDEF.GH', 'ABCDEFGH.']

Line 3 start state:
8 7 4
3 6 .
1 5 2
Line 3 goal state: 12345678.
Line 3 children: ['87.364152', '87436215.', '8743.6152']

Line 4 start state:
. 2 5
1 8 7
6 4 3
Line 4 goal state: 12345678.
Line 4 children: ['125.87643', '2.5187643']

Line 5 start state:
8 6 3
. 5 4
2 1 7
Line 5 goal state: 12345678.
Line 5 children: ['.63854217', '863254.17', '8635.4217']

Line 6 start state:
A B . C
E F G D
I J K H
M N O L
Line 6 goal state: ABCDEFGHIJKLMNO.
Line 6 children: ['ABGCEF.DIJKHMNOL', 'A.BCEFGDIJKHMNOL', 'ABC.EFGDIJKHMNOL']

Line 7 start state:
. B C D
A E G H
I F J L
M N K O
Line 7 goal state: ABCDEFGHIJKLMNO.
Line 7 children: ['ABCD.EGHIFJLMNKO', 'B.CDAEGHIFJLMNKO']

Line 8 start state:
A B C D E
F . H I J
K G M N O
P L R S T
U Q V W X
Line 8 goal state: ABCDEFGHIJKLMNOPQRSTUVWX.
Line 8 children: ['A.CDEFBHIJKGMNOPLRSTUQVWX', 'ABCDEFGHIJK.MNOPLRSTUQVWX', 'ABCDE.FHIJKGMNOPLRSTUQVWX',
'ABCDEFH.IJKGMNOPLRSTUQVWX']

Line 9 start state:
F A B C E
. H I D J
K G M N O
P L R S T
U Q V W X
Line 9 goal state: ABCDEFGHIJKLMNOPQRSTUVWX.
Line 9 children: ['.ABCEFHIDJKGMNOPLRSTUQVWX', 'FABCEKHIDJ.GMNOPLRSTUQVWX', 'FABCEH.IDJKGMNOPLRSTUQVWX']
```