Artificial Intelligence

Sliding Brick Puzzle

Part II: Searching for Solutions (12 points)

In the previous part, we implemented key components of a solver for the puzzle game *Sliding Brick Puzzle*. In this part, we continue with this implementation and significantly extend it to find puzzle solutions using various methods.

Please start with your code from part 1 and extend it as directed below. As before, the code for this part should be written in Python to run on tux.cs.drexel.edu, and we will use the same **run.sh** shell script for testing. Again, **you may only use built-in standard libraries (e.g., math, random, etc.); you may NOT use any external libraries or packages** (if you have any questions at all about what is allowable, please email the instructor).

Important notes: Notice that the search space is a **graph**, so you will have to keep track of all the states visited so far, and make sure your algorithm does not get stuck in loops.

For all the search functions that you will implement in this assignment, the output should print the following:

- the list of moves required to solve the state,
- the final state of the puzzle,
- the number of nodes that were explored,
- the time that the function took to find the solution in seconds with two decimal digits,
- and the length of the solution found.

Breadth-First Search (BFS) (3pts)

Write a method that does a breadth-first search from the given state from a file, returning the sequence of moves found that reaches the solution state. Add a "**bfs**" command-line command that allows a user to perform the BFS on a given file representing the state like before.

Here is an example (**pay attention to spaces and newlines**):

```
> sh run.sh bfs " SBP-level0.txt"

(2,left)
(4,down)
(3,right)
(2,up)
(2,up)

5,4,
 1, 2, 2, 1, 1,
 1, 0, 0, 3, 1,
 1, 0, 0, 4, 1,
 1, 1, 1, 1, 1,

16
0.00
5
```

Depth-Frist Search (DFS) (3pts)

Similar to BFS, write a method that does a depth-limited search from a given cube, returning the first sequence of moves found that reaches the solution state. Like the previous part, add a "**dfs**" command-line command that allows a user to perform the DFS on a given file representing the state:

```
> sh run.sh dfs " SBP-level0.txt"

(2,left)
(3,left)
(2,right)
(3,down)
(4,left)
(4,left)
(2,up)
(3,right)
(3,right)
(4,down)
(2,left)
(2,up)

5,4,
 1, 2, 2, 1, 1,
 1, 0, 0, 3, 1,
 1, 0, 0, 4, 1,
 1, 1, 1, 1, 1,

15
0.00
12
```

Iterative Deepening Search (IDS) (3pts)

Write a method that does an iterative deepening search from a given state, returning the first sequence of moves found that reaches the solution state. Add an "**ids**" command-line command that allows a user to perform this search:

```
> sh run.sh ids " SBP-level0.txt"

(3,left)
(2,left)
(4,down)
(3,right)
(3,right)
(2,up)
(2,up)

5,4,
 1, 2, 2, 1, 1,
 1, 0, 0, 3, 1,
 1, 0, 0, 4, 1,
 1, 1, 1, 1, 1,

19
0.02
7
```

A Search (3pts)*

Lastly, write a method that does an A* search from a given state, returning the first sequence of moves found that reaches the solution state. Add an "**astar**" command-line command that allows a user to perform this task. Note that as part of this process, you will need to choose and implement an *admissible* heuristic function $h(n)$, such that A* can reasonably estimate the minimum cost from a given state to the goal state. As a heuristic, use the Manhattan distance between the master brick and the goal. Also, note that the A* and BFS should have a similar length which is the length of an optimal solution for a state.

Here is an example:

```
> sh run.sh bfs " SBP-level0.txt"

(2,left)
(4,down)
(3,right)
(2,up)
(2,up)

5,4,
 1, 2, 2, 1, 1,
 1, 0, 0, 3, 1,
 1, 0, 0, 4, 1,
 1, 1, 1, 1, 1,

12
0.00
5
```

Extra Credit (up to 2 pts)

For extra points, you can try to improve the run-time of your algorithm more and compete against the rest of the class (shorter run-time is better). We will check each competitor's timing with more complex puzzles: *SBP-bricks-level1.txt*, *SBP-bricks-level2.txt*, *SBP-bricks-level3.txt*, *SBP-bricks-level4.txt*, *SBP-bricks-level5.txt*, *SBP-bricks-level6.txt*, *SBP-bricks-level7.txt*.

The top 5 agents will receive extra credit of up to 2 points. Remember that to have a chance to win the competition, your function should be able to complete the run in less than 5 minutes, even for the solutions at level 7.

There are a few ways that you can improve the algorithm. The first is to work on improving your data structure and search heuristics. Second, you could implement a variant of A* like Bidirectional A* or IDA*.

If you would like to enter the competition for extra credit, you must specifically mention that in your document and add a "**competition**" command-line command that, just as before, receives a file and prints the default output.

Academic Honesty

Please remember that you must write all the code for this (and all) assignments by yourself, on your own, without help from anyone except the course TA or instructor.

Submission

Remember that your code must run on **tux.cs.drexel.edu**—that's where we will run the code for testing and grading purposes. Code that doesn't compile or run there will receive a grade of zero.

For this assignment, you must submit:

- Your Python code for this assignment.
- Your **run.sh** shell script that can be run as noted in the examples.
- We should be able to run all the functions mentioned above (**bfs**, **dfs**, **ids**, **astar**, and **competition** if you would like to enter the competition) using your **run.sh** shell script.
- A PDF document with written documentation containing a few paragraphs
    - explaining your program,
    - analyzing the time and space complexity of algorithms and comparison (one or two paragraphs is enough),
    - explaining the heuristic you used for A* if it is anything other than the one explained above,
    - results showing testing of your routines,
    - and if you did anything extra as well as anything that is not working.

Please use a compression utility to compress your files into a single ZIP file (NOT RAR, nor any other compression format). The final ZIP file must be submitted electronically using Blackboard—do not email your assignment to a TA or instructor! If you are having difficulty with your Blackboard account, you are responsible for resolving these problems with a TA or someone from IRT before the assignment is due. If you have any doubts, complete your work early so that someone can help you.