CSE 3521: Survey of Artificial Intelligence I

Project #1: Problem solving by searching

Due: 9/23 (Wed) before the start of class

Instructions:

- Use C++, JAVA, or Python only.
- You are allowed to read, copy and modify the codes from *http://aima.cs.berkeley.edu/code.html*.
- You are **NOT** allowed to search any other source online or offline.
- Comment amply on the source code.
- Double-check if your code compiles on stdlinux without errors.
- Compress the source files and the compiling instructions into a single zip file, and upload it onto the Carmen dropbox.

8-puzzle: you will solve 8-puzzle using IDDFS and A*.

1. (20 points) Problem formulation. Start writing codes for solving this task.

    - Define a suitable variable/class for states, represented by a sequence of tile numbers from left to right of the top row, then left to right of the middle row, and so on. (For example (7,2,4,5,0,6,8,3,1) in the slides, where 0 mean a blank.)
    - Define a variable/class for actions {‘up’, ‘down’, ‘left’, ‘right’} for the blank tile. You must use THIS ACTION ORDER when generating child nodes.
    - Define a Node variable/class with the attributes (state, parent node, action, path-cost, depth, cost2go). The cost2go attribute is used in A*, but unnecessary for IDDFS.
    - Assume unit step cost, that is, path-cost is the total number of actions performed.
    - Define frontier and explored set. Use standard C++/JAVA/Python libraries for priority queues.
    - Write the function ChildNode(node, action [,goalstate]) which returns a child node or None if the action is not allowed at the state. The goalstate might be needed for A*.

- Define a goal check function. The goal state is (0,1,2,3,4,5,6,7,8).
- Define the function Solution(node) which returns the solution (= sequence of nodes from initstate to goalstate) by recursively backtracking node.parent and its parent, and so on. .

2. (20 points) DFS

- Implement IDDFS. (Use either the BFS pseudocode in the slides and change it, or use the recursive version).
- Your script should start with the initial state (0,3,5,4,2,7,6,8,1) and output the sequence of state and action on the screen in this format:

```
Step 0 : None
0 3 5
4 2 7
6 8 1
Step 1 : right
3 0 5
4 2 7
6 8 1

… omitted …

Step 12 : up
0 1 2
3 4 5
6 7 8
```

3. (20 points) A*

- Implement the heuristic function Cost2Go(state, goalstate) which returns the number of misplaced tiles (as explained in class).
- Implement A* search. Note that your A* code will be very similar to BFS/DFS except that A* uses the path-cost + cost-to-go as priority.

- Your script should return results in the same format as problem 2.

Hints and expected outcome:

- Submit separate binaries for IDDFS and A*. To get the full score, both algorithms of yours should find solutions in 12 steps.
- If you have trouble debugging, try first with a simple initial state such as (1,2,0,3,4,5,6,7,8), which can be solved by two ‘left’ actions.
- Your code should contain definitions of class Node, ChildNode(), Solution(), and Cost2Go().
- You can implement IDDFS with a recursion, or by making the following changes to DFS:
    a. Implement DLS (depth-limited DFS), which is the same as DFS except that 1) it will take "cutoff" as an additional argument and 2) it will not add a child node to the frontier if the node.depth > cutoff. Naturally, if the solution isn't found at the current cutoff, the frontier will go empty and DLS will return with an empty solution.
    b. Now range through cutoff = 0,...,20 (maxcutoff) and call DLS with the cutoff value, and stop if DLS finds a solution. It HAS to find the 12-step solution for this 8-puzzle.