CS 540-1: Introduction to Artificial Intelligence
Homework Assignment # 3

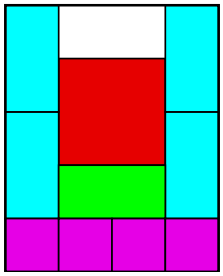
Assigned: 9/20
Due: 9/27 before class

Question 1: The Flying Klotski [100 points]

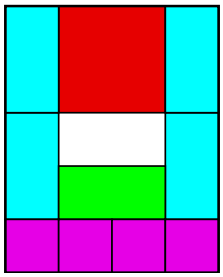
This is a programming question. The solution to the programming problem should be coded in Java, **and you are required to use only built-in libraries** to complete this homework. Please submit a single zip file named hw3.zip, which should contain a source code file named **Klotski.java** with no package statements, and make sure your program is runnable from command line on a department Linux machine. We provide a skeleton Klotski.java code that you can optionally use, or you can write your own.

The goal of this assignment is to become familiar with A* search. The assignment tests your understanding of AI concepts, and your ability to turn conceptual understanding into a computer program. All concepts needed for this homework have been discussed in class, but there may not be existing pseudo-code for you to directly follow.

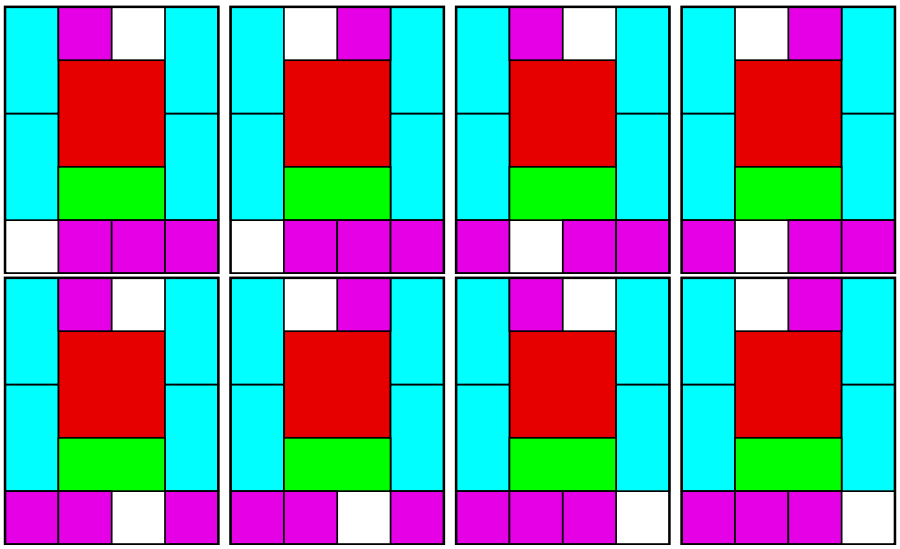
Klotski is a sliding block puzzle thought to have originated in the early 20th century. Ten pieces (colored) are placed inside a 5×4 box, among them one 2×2 block, four 2×1 blocks, one 1×2 block and four 1×1 blocks. The empty spaces are white. The following is a possible initial state S_1:



Our rules for Klotski is slightly different from the original ones. The player is not allowed to remove blocks. Except for 1×1 blocks, larger blocks can only move horizontally or vertically one square at a time. So S_1 has this successor by moving the 2×2 block:

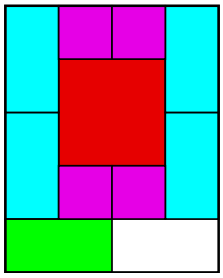


However, we allow a 1×1 block to *fly* to any empty space, no matter if the empty space is adjacent to it or not. This also counts as one move. Because of this new rule, S_1 has 8 additional successors by flying a 1×1 block:

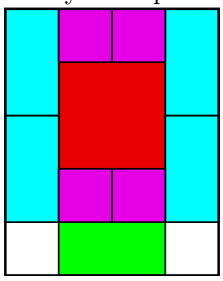


So S_1 has 9 successors in all.

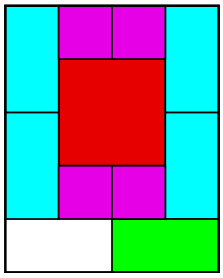
Let us look at another example with a different state S_2:



To emphasize, larger blocks can only move by one square in one step. This is a valid successor of S_2:

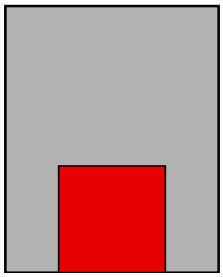


But this is not:



Of course, S_2 has 8 additional successors resulting from flying the 1×1 blocks, and one more successor by sliding down the lower right cyan block.

The goal of the game is to move the 2×2 red block to the center of the lowest row (the "exit"): We do not care about the position of other blocks (omitted in the gray area). Therefore, you have a simple goal test that corresponds to multiple goal states:



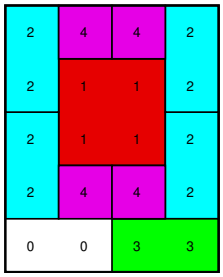
For this question, you will use A* search to find the shortest path from an initial state to a goal state. Write a program **Klotski.java** with the following command line format:

```
$java Klotski FLAG tile1 tile2 tile3 ... tile20
```

FLAG is an integer that specifies the behavior and output of the program (see below). In the command line, tile1 – tile20 specify the initial state in the natural reading order (left to right, top to bottom). We use 0 to represent empty block, 1 to represent 2×2 block, 2 to represent 2×1 block, 3 to represent 1×2 block and 4 to represent 1×1 block. For example, the following command line

```
$java Klotski 100 2 4 4 2 2 1 1 2 2 1 1 2 2 4 4 2 0 0 3 3
```

represents this initial state:



1. When FLAG=100, generate, sort, and print all successors of the initial state. For each state, print 5 lines of 4 digits per line, with no space between each digit; then print an empty line.

   The successors should be sorted according to the following rule. If we view the whole board as a 20-digit number in the natural reading order (from left to right, top to bottom), then print the successors sorted from small to large. Hint: you may also implement these 20-digit integers as strings and compare them according to dictionary order.

   Please see examples `example*100` in the attached files; the corresponding inputs are in file `input`.

2. Implement A* search as defined on slide 21 of informed search (see course webpage). Use a constant heuristic function $h = 0$ for all states. This reduces A* to uniform cost search. For tie breaking when popping from the priority queue, pop the state with the smallest 20-digit ID among states with the minimum f score.

   When FLAG=200, in each iteration (please see examples `example*200` in the attached files; so on and so forth for other FLAGs):

   - right after step 3 print a line *iteration* i where i is the iteration number (starting from 1), then print the node n;

- right after step 5 print the word *OPEN*, print the nodes in OPEN, print the word *CLOSED*, then print the nodes in CLOSED. The nodes do not need to be sorted.

Each node should be printed with the following format:

- the unique 20-digit node ID
- the state as a 5×4 board
- its f, g, h values separated by space
- its back pointer, i.e. the ID of its parent. If there is no parent, print the word *null*.

3. When FLAG=300, print the following:
   (a) The solution path from the initial state to the goal state, namely the sequence of states. Each state should be printed in the same format as with FLAG=100, i.e. just the 5×4 board and an empty line;
   (b) The word *goalCheckTimes* followed by the total number of times you performed goal-check;
   (c) The word *maxOPENSize* followed by the maximum numbers of states in your Open at any moment in your search;
   (d) The word *maxCLOSEDSize* followed by the maximum numbers of states in your CLOSED at any moment in your search;
   (e) The word *steps* followed by the length (number of edges, i.e. number of states minus 1) of your solution path.
4. Add the following admissible heuristic h to your A* algorithm: For a state s, let $h(s)$ be the Manhattan distance between the 2×2 block and where it should be in a goal state. When FLAG=400, print the same things as in FLAG=200 but with this Manhattan h.
5. When FLAG=500, print the same things as in FLAG=300 but with this Manhattan h.