CS 540-1: Introduction to Artificial Intelligence
Homework Assignment # 5

Assigned: 11/11
Due: 11/24 before 9:55 a.m.

Hand in your homework:

This homework includes a written portion and a programming portion. Please type the written portion and hand in the file in pdf format and name it as **WrittenPart.pdf**. The first page of the pdf must include a header with: **your name, Wisc username, class section, HW# , date and, if handed in late, how many days late it is**. The programming portion will be required to write in Java, and all files needed to run your program, including any support files but input/output/debug files, should be submitted. Finally, please create a folder named as <**Wisc username**>_**HW#**, put **all your codes** as well as **WrittenPart.pdf** into the folder and compress it as <**Wisc username**>_**HW#.zip**. This final zip file should then be uploaded to the appropriate place on the course Moodle website.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:55 a.m., and it is handed in between Wednesday 9:55 a.m. and Thursday 9:55 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline. A total of two (2) free late days may be used throughout the semester without penalty.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Search Algorithms [50]

Little Red Riding Hood wants to go to her grandmother's house in the forest to give her some cake. The forest is a dangerous place where the Big Bad Wolf resides. Little Red Riding Hood's mother has instructed her not to venture into the forest and to only go along the road. The following 5×5 grid shows the area that Little Red Riding Hood needs to traverse. She starts from square A (denoted by R). Her grandmother's house is located at square N (denoted by G). The forests are denoted by F (on squares F, I, M, R, U and Y). Your task is to help Little Red Riding Hood get to her grandmother's house. Assume that she can only move in four directions namely left, down, right and up. She cannot travel diagonally and will not go into the forests. **Also assume that** the successor function will cause legal moves to be examined in an **counter-clockwise** order: left; down; right; up. Note that not all of these moves may be possible for a given square (squares with forest on them will never be examined).

A R	B	C	D	E
F F	G	H	I F	J
K	L	M F	N G	O
P	Q	R F	S	T
U F	V	W	X	Y F

a) [10] Using **Depth-First Search**, list the squares in the order they are expanded (including the goal node if it is found). **Square A** is expanded first (hint: State B will be examined next). **Assume cycle checking is done** so that a node is not generated in the search tree if the grid position already occurs on the path from this node back to the root node (i.e., Path Checking DFS). Write down the list of states you expanded in the order they are expanded. Write down the solution path found (if any), or explain why no solution is found.

b) [10] Using **Breadth First Search** write down the list of states you expanded in the order they are expanded (until the goal node is reached). Use the same cycle checking as in the previous question.

c) [10] Using **Iterative Deepening Search**, draw the trees built at each depth until a solution is reached. Use the same cycle checking as in the previous questions.

d) [10] Consider the heuristic function $h(n)$ which is the Manhattan distance between a given square and the goal square. Is $h(n)$ an admissible heuristic?

e) [10] Perform A* search using the heuristic function $h(n)$. In the case of ties, expand states in alphabetical order. List each square in the order they are added to the OPEN list, and mark it with $f(n) = g(n) + h(n)$ (show f, g and h separately). When expanded (including square N), label a state with a number indicating when it was expanded (square A should be marked 1). Highlight the solution path found (if any), or explain why no solution is found.

Question 2: Programming Part: A* Search[50]

In this question, your task is to implement A* search algorithm to solve a game problem, the rotation game. Moreover, you will be asked to try different heuristic functions in you A* algorithm to understand the performance difference between them.

Game description

The rotation game uses a # shaped board, which can hold 24 pieces of square blocks (see Fig.1). The blocks are marked with symbols 1, 2 and 3, with exactly 8 pieces of each kind.

Initially, the blocks are placed on the board randomly. Your task is to move the blocks so that the eight blocks placed in the center square have the same marking symbol. There is only one type of valid move, which is to rotate one of the four lines, each consisting of seven blocks. That is, six blocks in the line are moved towards the head by one block and the head block is moved to the end of the line. The eight possible moves are marked with capital letters A to H. Figure 1 illustrates two consecutive moves, move A and move C from some initial state.

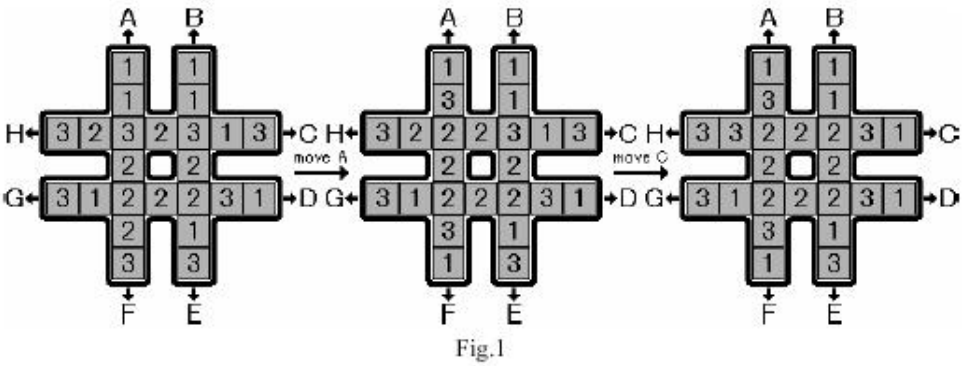

Figure 1: The Rotation Game.

The input file will comprise of only one line, which contains 24 digits corresponding to the symbols of the blocks in the initial state. The rows of blocks are listed from top to bottom. For each row the blocks are listed from left to right.

For each input file, your program should apply **a specified heuristic function**, and output three corresponding results for **the goal state with the minimum steps of moves**. The first result is the state for this goal. The second result is the operation sequence with letters ranging from 'A' to 'H', which represents the moves needed to reach this goal. If no moves are needed, you should output "No moves needed" instead. The third result is the number of popped out states from the queue (including the goal itself). **Notice that there could be multiple goal states with the minimum steps of moves, which leads to uncertainty in the grading process. In light of it, we will introduce the way to resolve this uncertainty in the next subsection.** The following shows a sample input and output for the heuristic function given by Eq.1, and it corresponds to the board game shown in Fig. 1.

Sample Input

```
111132323132231222312133
```

Sample Output

```
  1 1
  3 1
3322231
  2 2
3122231
  3 1
  1 3
AC
3
```

Methods to implement

In this programming question, you are only required to implement four methods of the class `AStarSearchImpl`, which are showed as follows:

```
public class AStarSearchImpl implements AStarSearch {
       public SearchResult search(String initConfig, int modeFlag);
       public boolean checkGoal(String config);
       public String move(String config, char op);
       public int getHeuristicCost(String config, int modeFlag);
}
```

In detail, you should implement the A* algorithm **strictly** according to the pseudo code shown in Alg.1, which is primarily based on the codes in Prof. Zhu's slides but with more details. As mentioned before, there might be several goal states with the minimum operation length, so there are some points that must be mentioned in order to ensure the final answer to be unique:

1. In line 3, it's possible that there might exist multiple nodes with the same least f value. In this situation, you should always pop the node with the least f value and smaller dictionary order with respect to node's operation sequence. In fact, we have already written a comparator function for this comparison criteria, as seen in `State.java`, so that you can use it to build the priority queue directly.
2. In line 8, you should generate the successor of n from operation A to operation H.
3. In line 16, you shouldn't add an extra copy of n' into OPEN. Combined with line 18, it implies that at any moment, there will only at most one copy for any possible node.

To implement the priority queue OPEN, we recommend (but not require) you to use the `PriorityQueue` in java with the comparator defined in `State.java`. Unfortunately, `PriorityQueue` in java does not support modification of its elements directly, so in order to implement line 16, you have to first use `PriorityQueue.remove` to remove the element in the queue with n', then add a new element with n' and new values into the queue. To use the function `remove` of the `PriorityQueue`, you need to override the `equals` function for the class of elements contained in the queue. Like the comparator function, we have also finished this part for you in `State.java`. However, be careful that this step does not increase the third result by one, namely the number of states popped out from the queue.

Finally, method `search` should return the result of class `SearchResult`, which comprises the final state, operation sequence and number of states popped out from the queue as mentioned.

For other parts of programming, you also have to implement the methods `checkGoal`, `move` and `getHeuristicCost`, whose prototypes are well commented in `AStarSearch.java`. Also to reduce your

programming burden, we have written I/O parts for you, so you are **NOT** responsible for any file input or console output. Please don't modify the codes for I/O part.

Algorithm 1 A* Algorithm

```
1: Put the start state S on the priority queue, called OPEN
2: while OPEN is not empty do
3:    Remove from OPEN and place in CLOSED a state n for which f(n) is the minimum
4:    if n is a goal state then
5:       exit(trace back pointers from n to S)
6:    end if
7:    Expand n, generating all its successors and attach to them pointers back to n.
8:    for each successor n' of n do
9:       if n' is not in OPEN or CLOSED then
10:         Estimate h(n'), g(n') = g(n) + c(n, n'), f(n') = g(n') + h(n')
11:         Place n' on OPEN.
12:      else
13:         if g(n') strictly less than its old g value in OPEN or CLOSED then
14:            Update pointers backward from n' to n
15:            if n' is in OPEN then
16:               Update g(n') in OPEN.
17:            else
18:               Remove n' from CLOSED and place it on OPEN with new g(n')
19:            end if
20:         end if
21:      end if
22:   end for
23: end while
24: Exit with failure.
```

Heuristic function

For this problem, you are required to try different heuristic functions to see their effects on A* algorithm. The first heuristic function is

$$h_1(s) = 8 - \max(n_1, n_2, n_3), \tag{1}$$

where n_1, n_2, n_3 are number of blocks in the central square. This formula is obtained by relaxing the constraints and allowing the blocks to switch remotely.

The second one is

$$h_2(s) = 0 \tag{2}$$

As a result, the A* algorithm with this heuristic will degenerate into BFS.

The last one should be designed by yourself. You should **write down your heuristic function h_3 and verify its correctness in WrittenPart.pdf**. We are happy to see if you can raise a better heuristic function, but since it is an open question, so we do not require your heuristic function to beat h_1. As long as you can show the validity of your heuristics and incorporate it into the code sketch correctly, then you will get full scores for this part.

How to test

We will test your program on multiple test cases, and the format of testing commands will be like this:
`java HW5 <inputFileName> <modeFlag>`
where `inputFielName` is the name of the input file which contains the initial state of the rotation game, and `modeFlag` is an integer ranging from 1 to 3, controlling which heuristic function your programm should use. Your program should read the initial state from the input file and then output three results sequentially:

- the goal state
- the operation sequence
- the number of popped out states from the queue

For all these three outputs, you results **must match exactly to the standard ones for `modeFlag`=1 or 2**. In cases of you may implement differently from Alg.1, we are providing you with three sample input files and six corresponding output files for `modeFlag`=1 or 2. They are `input0.txt` to `input2.txt`, and `output01.txt` to `output22.txt`, as seen in the zip file. For `modeFlag`=3, however, we will only check the validity of your result. That means **we will judge whether the output goal state can be generated under your operation sequence, and whether the number of moves is same as the one computed with given heuristic**. So an example command for testing for heuristic function h_1 could be:

```
java HW5 input1.txt 1
```

As part of our testing process, we will unzip the file you return to us, remove any class files, call `javac *.java` to compile your code, and call the main method of HW5 with certain parameters. Make sure that your code runs on one of the computers in the department because we will conduct our test on such computers.

Deliverables

1. Hand in (see the cover page for details) your modified version of the code skeleton we provide you with your implementation of the A* search algorithm. Also include any additional java class files needed to run the program.
2. Optionally, in the written part of your homework, add any comments about the program that you would like us to know.