CS 540: Introduction to Artificial Intelligence
Homework Assignment # 3

Assigned: 2/19
Due: 2/26 before class

Hand in your homework:

If a homework has programming questions, please hand in the Java program. If a homework has written questions, please hand in a PDF file. Regardless, please zip all your files into hwX.zip where X is the homework number. Go to UW Canvas, choose your CS540 course, choose Assignment, click on Homework X: this is where you submit your zip file.

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 9:30 a.m., and it is handed in between Wednesday 9:30 a.m. and Thursday 9:30 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.

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 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: Torus 8-Puzzle [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 **Torus.java** with no package statements, and make sure your program is runnable from command line on a department Linux machine. We provide a skeleton Torus.java code that you can optionally use, or you can write your own.

The goal of this assignment is to become familiar with iterative deepening depth-first search (DFS). 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. We ask you to implement your own stack for DFS as we did in class, rather than writing a recursive DFS program.

Recall the standard 8-puzzle is a sliding game, where the top, bottom, left, or right neighboring tile can slide into the empty square. However, if the empty square is not in the middle, not all four neighbors exist.

A torus 8-puzzle is a variant with the following change: when the empty square is at the border, the tile at the opposite end can slide into it, too. Here is an example:

1	2	3
4	5	
6	7	8

This state has four successors in torus 8-puzzle. Three of them are the standard:

1	2	
4	5	3
6	7	8

1	2	3
4	5	8
6	7	

1	2	3
4		5
6	7	8

However, the fourth successor allows the tile 4 to slide out to the left and re-enter from the right into the empty square:

1	2	3
	5	4
6	7	8

You can think of that row as a ring, where a tile going out of the board from one end automatically enters from the other end. This is true for all rows and all columns. Technically (in topology), this is known as a torus – hence the name. (Caution: If you search torus puzzle online, you may come across the paper "How to Solve the Torus Puzzle." That game is *different*.)

Here is another example state:

	1	2
3	4	5
6	7	8

It has four successor states. Two are standard:

1		2
3	4	5
6	7	8

3	1	2
	4	5
6	7	8

The third one comes from the torus row ring movement:

2	1	
3	4	5
6	7	8

And the fourth one comes from the torus column ring movement:

6	1	2
3	4	5
	7	8

It is easy to see that *any* state has exactly four successors in the torus 8-puzzle.

For this question, you will use iterative deepening depth-first search to find the shortest path from an input state to the goal state (which will always be this state):

1	2	3
4	5	6
7	8	

Write a program **Torus.java** with the following command line format:

```
$java Torus FLAG tile1 tile2 tile3 tile4 tile5 tile6 tile7 tile8 tile9
```

where FLAG is an integer that specifies the output of the program (see below). Tile1 − tile9 specify the initial state in the natural reading order (left to right, top to bottom). These take values in integer 1–8 for the 8 tiles, plus 0 for the empty space. For example, if the initial state is our very first example and FLAG=100, the command line would be

```
$java Torus 100 1 2 3 4 5 0 6 7 8
```

(Part a, 20 points) When FLAG=100, print out the four successor states of the initial state, in the order they are pushed into the stack (see below). Each successor state should be printed as 9 numbers on a single line. For example,

```
$java Torus 100 1 2 3 4 5 0 6 7 8
1 2 0 4 5 3 6 7 8
1 2 3 0 5 4 6 7 8
1 2 3 4 0 5 6 7 8
1 2 3 4 5 8 6 7 0
```

Important: We ask you to implement the following order among successors. If we view the 9 numbers as a 9-digit integer, then there is a natural order among states. Whenever you push successors into the stack, push them from small 9-digit to large 9-digit. This means that when we later pop them out, the largest 9-digit successor will be goal-checked before the other successors. This order will be used throughout this program, so that the output is well-defined.

(Part b, 20 points) When FLAG=2XX, perform a depth-limited depth-first search with cutoff depth XX (i.e. this is one outer-loop of iterative deepening). For example, if FLAG=200, the cutoff depth is 0. In DFS you will push the initial state in the stack, pop it out, do a goal test, but will NOT expand it. If FLAG=201, the cutoff depth is one. In DFS you will expand the initial state (i.e. put its four successors into the stack in the order we specified in Part a). You will pop each successor out, print it, and perform goal-check (and terminate the program if goal-check succeeds). But you will not expand any of these successors.

XX can be 00 to 99. If depth-limited DFS finds a goal before the cutoff, it should stop.

You will need to implement both backpointers and path-checking cycle prevention. Note: do not use the whole CLOSED set for cycle prevention, which is not memory efficient for DFS. Use a "prefix path" instead. Recall in the prefix path you only need to record the path from the initial state to the current state being goal-checked. Specifically, you can implement the prefix path-checking as follows:

- Use a data structure that supports linear ordering of states, such as a list.
- When you pop out a state s for goal-checking, it comes with a parent backpointer. Say its parent state is p. Look into the prefix list, it should contain p somewhere as in $initial, \ldots, p, \ldots$. Remove everything after p and put s there, so your prefix list now looks like $initial, \ldots, p, s$.
- At the very beginning when s is the initial state, just put it in the empty prefix data structure.
- Whenever you generate a successor t, you want to check if t is in the current prefix list. If no, push t to the DFS stack; if yes, do not push it. This is path-checking cycle prevention.

For this part, print out the states in the order of goal-test (i.e. when you pop them out of the stack). For example:

```
$java Torus 201 1 2 3 4 5 0 6 7 8
1 2 3 4 5 0 6 7 8
1 2 3 4 5 8 6 7 0
1 2 3 4 0 5 6 7 8
1 2 3 0 5 4 6 7 8
1 2 0 4 5 3 6 7 8
```

(Part c, 20 points) When FLAG=3XX, perform a depth-limited depth-first search with cutoff depth XX like in part b. But this time, also print out the backpointers. That is, each printed state should be followed by the word "parent," then the parent state (all space-separated). Use all-zero for the parent of the initial state. For example,

```
$java Torus 301 1 2 3 4 5 0 6 7 8
1 2 3 4 5 0 6 7 8 parent 0 0 0 0 0 0 0 0 0
1 2 3 4 5 8 6 7 0 parent 1 2 3 4 5 0 6 7 8
1 2 3 4 0 5 6 7 8 parent 1 2 3 4 5 0 6 7 8
1 2 3 0 5 4 6 7 8 parent 1 2 3 4 5 0 6 7 8
1 2 0 4 5 3 6 7 8 parent 1 2 3 4 5 0 6 7 8
```

(Part d, 20 points) When FLAG=4XX, perform a depth-limited depth-first search with cutoff depth XX like in part b. But this time, only print out the prefix path (starting from the initial state) for the very first state you goal-check at depth XX+1 (recall the initial state is goal-checked at depth 1). For example,

```
$java Torus 400 1 2 3 4 5 0 6 7 8
1 2 3 4 5 0 6 7 8
$java Torus 401 1 2 3 4 5 0 6 7 8
1 2 3 4 5 0 6 7 8
1 2 3 4 5 8 6 7 0
$java Torus 402 1 2 3 4 5 0 6 7 8
1 2 3 4 5 0 6 7 8
1 2 3 4 5 8 6 7 0
1 2 3 4 5 8 6 0 7
```

(Part e, 20 points) When FLAG=500, perform iterative deepening (which can go beyond depth cutoff 99 if necessary). Print out the following output:

1. The solution path from the initial state to the goal state (one state per line)
2. The phrase Goal-check followed by the total number of times you perform goal-check. A state can be goal-checked multiple times as you gradually increase the depth cutoff, and should be counted multiple times.
3. The phrase Max-stack-size followed by the maximum number of states in your DFS stack (N.B. not the prefix) at any moment in your search.

For example,

```
java Torus 500 1 2 3 4 5 0 6 7 8
1 2 3 4 5 0 6 7 8
1 2 3 0 5 4 6 7 8
1 2 3 6 5 4 0 7 8
1 2 3 6 5 4 7 0 8
1 2 3 6 0 4 7 5 8
1 2 3 6 4 0 7 5 8
1 2 3 0 4 6 7 5 8
1 2 3 4 0 6 7 5 8
1 2 3 4 5 6 7 0 8
1 2 3 4 5 6 7 8 0
Goal-check 40517
Max-stack-size 19
```

Finally, we provide two additional examples to help you develop your code.

```
Extra Example 1:
Part A)
Input:
$java Torus 100 8 7 6 5 4 3 2 1 0
Output:
8 7 0 5 4 3 2 1 6
8 7 6 5 4 0 2 1 3
8 7 6 5 4 3 0 1 2
8 7 6 5 4 3 2 0 1

Part B)
Input:
$java Torus 201 8 7 6 5 4 3 2 1 0
Output:
8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 0 1
8 7 6 5 4 3 0 1 2
8 7 6 5 4 0 2 1 3
8 7 0 5 4 3 2 1 6
```

```
Part C)
Input:
$java Torus 301 8 7 6 5 4 3 2 1 0
Output:
8 7 6 5 4 3 2 1 0 parent 0 0 0 0 0 0 0 0 0
8 7 6 5 4 3 2 0 1 parent 8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 0 1 2 parent 8 7 6 5 4 3 2 1 0
8 7 6 5 4 0 2 1 3 parent 8 7 6 5 4 3 2 1 0
8 7 0 5 4 3 2 1 6 parent 8 7 6 5 4 3 2 1 0

Part D)
Input:
$java Torus 401 8 7 6 5 4 3 2 1 0
Output:
8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 0 1

Part E)
Input:
$java Torus 500 8 7 6 5 4 3 2 1 0
Output:
8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 0 1
8 7 6 5 4 3 0 2 1
8 7 6 5 4 3 1 2 0
8 7 6 5 4 0 1 2 3
8 7 6 5 0 4 1 2 3
8 7 6 5 2 4 1 0 3
8 0 6 5 2 4 1 7 3
0 8 6 5 2 4 1 7 3
1 8 6 5 2 4 0 7 3
1 8 6 5 2 4 7 0 3
1 0 6 5 2 4 7 8 3
1 2 6 5 0 4 7 8 3
1 2 6 0 5 4 7 8 3
1 2 6 4 5 0 7 8 3
1 2 0 4 5 6 7 8 3
1 2 3 4 5 6 7 8 0
Goal-check 42689480
Max-stack-size 32



Extra Example 2:
Part A)
```

```
Input:
$java Torus 100 4 3 8 5 1 6 7 2 0
Output:
4 3 0 5 1 6 7 2 8
4 3 8 5 1 0 7 2 6
4 3 8 5 1 6 0 2 7
4 3 8 5 1 6 7 0 2
```

Part B)

```
Input:
$java Torus 202 4 3 8 5 1 6 7 2 0
Output:
4 3 8 5 1 6 7 2 0
4 3 8 5 1 6 7 0 2
4 3 8 5 1 6 0 7 2
4 3 8 5 0 6 7 1 2
4 0 8 5 1 6 7 3 2
4 3 8 5 1 6 0 2 7
4 3 8 5 1 6 2 0 7
4 3 8 0 1 6 5 2 7
0 3 8 5 1 6 4 2 7
4 3 8 5 1 0 7 2 6
4 3 8 5 0 1 7 2 6
4 3 8 0 1 5 7 2 6
4 3 0 5 1 8 7 2 6
4 3 0 5 1 6 7 2 8
4 3 6 5 1 0 7 2 8
4 0 3 5 1 6 7 2 8
0 3 4 5 1 6 7 2 8
```

Part C)

```
Input:
$java Torus 302 4 3 8 5 1 6 7 2 0
Output:
4 3 8 5 1 6 7 2 0 parent 0 0 0 0 0 0 0 0 0
4 3 8 5 1 6 7 0 2 parent 4 3 8 5 1 6 7 2 0
4 3 8 5 1 6 0 7 2 parent 4 3 8 5 1 6 7 0 2
4 3 8 5 0 6 7 1 2 parent 4 3 8 5 1 6 7 0 2
4 0 8 5 1 6 7 3 2 parent 4 3 8 5 1 6 7 0 2
4 3 8 5 1 6 0 2 7 parent 4 3 8 5 1 6 7 2 0
4 3 8 5 1 6 2 0 7 parent 4 3 8 5 1 6 0 2 7
4 3 8 0 1 6 5 2 7 parent 4 3 8 5 1 6 0 2 7
0 3 8 5 1 6 4 2 7 parent 4 3 8 5 1 6 0 2 7
4 3 8 5 1 0 7 2 6 parent 4 3 8 5 1 6 7 2 0
4 3 8 5 0 1 7 2 6 parent 4 3 8 5 1 0 7 2 6
```

```
4 3 8 0 1 5 7 2 6 parent 4 3 8 5 1 0 7 2 6
4 3 0 5 1 8 7 2 6 parent 4 3 8 5 1 0 7 2 6
4 3 0 5 1 6 7 2 8 parent 4 3 8 5 1 6 7 2 0
4 3 6 5 1 0 7 2 8 parent 4 3 0 5 1 6 7 2 8
4 0 3 5 1 6 7 2 8 parent 4 3 0 5 1 6 7 2 8
0 3 4 5 1 6 7 2 8 parent 4 3 0 5 1 6 7 2 8

Part D)
Input:
$java Torus 402 4 3 8 5 1 6 7 2 0
Output:
4 3 8 5 1 6 7 2 0
4 3 8 5 1 6 7 0 2
4 3 8 5 1 6 0 7 2

Part E)
Input:
$java Torus 500 4 3 8 5 1 6 7 2 0
Output:
4 3 8 5 1 6 7 2 0
4 3 0 5 1 6 7 2 8
4 0 3 5 1 6 7 2 8
4 1 3 5 0 6 7 2 8
4 1 3 0 5 6 7 2 8
0 1 3 4 5 6 7 2 8
1 0 3 4 5 6 7 2 8
1 2 3 4 5 6 7 0 8
1 2 3 4 5 6 7 8 0
Goal-check 18088
Max-stack-size 17
```