CS 540: Introduction to Artificial Intelligence
Homework # 2

Assigned: 2/05
Due: 2/21 before class

Hand in your homework:

There are two programming questions in this homework. Please zip all your files into **HW2.zip** and submit it to the Canvas. The **HW2.zip** contains two files **Hanoi.java** and **Number.java** with **no package statements**. 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 make sure your programs are runnable from command line on a department Linux machine. We provide skeleton codes that you can optionally use, or you can write your own.

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 60-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., 6 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.

Collaborative 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.

Problem 1. Successor Function [30]

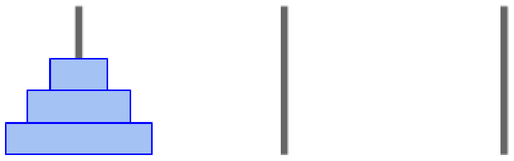
Tower of Hanoi is a famous game. In this game, there are some disks and rods and disk can be placed on other rods based on some rules. The rule of this game can be found at (https://en.wikipedia.org/wiki/Tower_of_Hanoi).

In this problem, we consider a modified version of Tower of Hanoi game. It has the following rules:

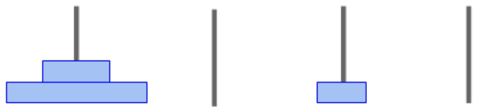
- In each step, only one disk can be moved.
- In each step, a disk can only be moved to the **adjacent rod(s)**.
- A disk can only be placed in an empty rod or on a larger disk.
- There can be more than three rods in this game. In the beginning, all disk can be randomly placed on different rods.

Write a program **Hanoi.java** to print the successor states of an input state.

- Input: n String $str1, str2, ...$ which represents n rods. Each string str represents the status of disks on this rod. We assume that the size of disk can only be 1 to 9. We use 0 to represent empy rod, and $s_1s_2..s_k$ represents that there are k disks in this rod and their size are s_1, s_2 ... s_k from top to bottom($s_i \neq s_j, \forall i, j$ and $k \leq 9$).
  For example, the following status can be represented as 123 0 0



  Another example as follows can be represented as 24 0 1 0



- Output: Print the list of successor states reachable in one step from the given input. Each line of output represents a state, the outputs follow the same format. The successors should not include the initial state itself or duplicates.

Here are some examples of running the program from the command line. The inputs and outputs are space-separated with no comma. Please follow the same input/output format.

- Example1:
```
$java Hanoi 123 0 0
23 1 0
```
  As we can only move between adjacent rods. Only the disk of size 1 can be moved from the 1st rod to the 2nd rod.
- Example 2:
```
$java Hanoi 24 0 1 0
4 2 1 0
24 0 0 1
24 1 0 0
```
  The disk of size 2 and disk of size 1 can be moved to the adjacent rods. The order of outputs does not matter. The following outputs are also correct.
```
4 2 1 0
24 1 0 0
24 0 0 1
```

Problem 2. Transforming Number[30]

In this problem, we are given two integers X and Y. Write a program **Number.java** to count the **least** steps we need to convert X to Y use the following rules:

- Increase X by 1.
- Decrease X by 1.
- Multiply X by 3.
- Get the square of X.
- You may assume that $0 \leq X < 100$, $0 \leq Y < 100$, and $X \neq Y$. During the transformation process, all numbers can not be less than 0 or greater than 99. That is, if the transformation is $X- > X_1- > X_2- > X_k ...- > Y$, we have $0 \leq X_i < 100$ and $i = 1, ...k$.

The input and output formats are as follow:

- Input: two integers X Y
- Output: one integer k. Indicate that the least number of steps is k to transform X to Y following the above rules.

- Example 1:
```
$java Number 1 5
3
```
The least number of steps is 3. We can transform like $1- > 2- > 6- > 5$, that needs three steps. However, we cannot do it in less than 3 steps.
- Example 2:
```
$java Number 1 19
4
```
We can transform the number like $1- > 2- > 6- > 18- > 19$. That needs 4 steps.
The transform paths may not be unique. However, we only output the least number of steps.

Hint: use the breadth-first search to solve this problem.