Homework Assignment 2

Fall 2024

IE 523 Financial Computing Due on Friday 09/27 (at 11:59 pm)

In the following two questions, you are asked to implement a solution procedure based on **recursion**. The first recursion you write will be naïve: in a sense, you are trying every possible solution *recursively*. The second recursion you work on will be a little more involved and it is based on our Hanoi towers puzzle we saw in class. Both questions require the creation of one class each.

Question 1: A recursive Sudoku solver

Sudokus have been a popular puzzle for quite some time (at least since the 19th century). That said, their recent popularity in Japan is due to a publication in 1986 (under the name of Sudoku – meaning a "single number"), as well as in the United States due to newspapers picking up the game in the early 2000s. The game now appears prominently in puzzle books and online game portals.

In this first question, we ask you to use recursion to find a solution to a given Sudoku. First, let us describe the basics. Sudoku is a combinatorial, number-placing game, played on a partially completed 9×9 grid. The grid is further divided into 9 3×3 "sub-grids" or sub-squares. The goal is then to fill every square in the grid with a number between 1 and 9. The catch?

- Every row needs to contain every number between 1 and 9 exactly once.
- Every column needs to contain every number between 1 and 9 exactly once.
- There are 9 "sub-grids"/sub-squares (rows 1-3 and columns 1-3, rows 1-3 and columns 4-6, rows 1-3 and columns 7-9, rows 4-6 and columns 1-3, and so on). Every "sub-grid"/sub-square also needs to contain every number between 1 and 9 exactly once.

We can solve the Sudoku puzzle in a recursive manner. First, identify the first empty grid spot, beginning from the upper left corner. Then, fill it with a number that is not already present in the row, column, or "sub-grid"/sub-square. Finally, proceed to the next empty grid spot and perform the same! Note that if at any point you are unable to add any number to the grid spot (that is, you have made a mistake in one of your previous entries), you *recursively* return false and call on the previous function to try again!

Write a class Sudoku that allows for the creation of a Sudoku puzzle, as read by a file (see the provided `sudoku.dat` file provided). Then, the Sudoku should possess proper functionality to solve the Sudoku puzzle provided recursively. We provide an example in Figure 1. That said, there could be alternative solutions – so please verify that your obtained solution forms a valid Sudoku! You can achieve that by defining an `isValid()` functionality which checks if all rules of the Sudoku are satisfied.

Question 2: Generalized Hanoi towers

During our recursion lecture, we saw one possible way of solving the Hanoi towers puzzle: a recursive procedure that was shown to be optimal for 3 towers. Our code was simple:

1. Move $n-1$ discs from the source tower to the middle tower.
2. Move the top disc (the only disc) from the source tower to the destination tower.

Figure 1: A Sudoku puzzle (left) and one of its possible solutions (right).

		6		8		5		
9			3				2	
	4				1			7
					5	6		
	2			4			1	
3								9
		1	9				3	
				2		4		
	5				7			

3	1	6	7	8	2	5	9	4
9	8	7	3	5	4	1	2	6
5	4	2	6	9	1	3	8	7
7	9	8	1	3	5	6	4	2
6	2	5	8	4	9	7	1	3
1	3	4	2	7	6	8	5	9
4	7	1	9	6	8	2	3	5
8	6	9	5	2	3	4	7	1
2	5	3	4	1	7	9	6	8

3. Finally, move again the $n-1$ discs from the middle tower to the destination tower.

Take a look at the code we saw during class for a better idea of how this work.

Now, we ask you to write a *similar* recursion for the Hanoi towers puzzle with 4 towers. Assume you have 4 towers, numbered as 1, 2, 3, and 4, and multiple discs (let it be n) in the first tower. The goal is to get all of the discs in the last tower (tower 4) with the smallest number of moves. The recursive procedure you are asked to implement is similar. First, pretend you only have 3 pegs. Then:

1. Move $n-k$ discs from the source tower to one of the middle towers (let this be tower m).
2. Move the remaining k discs from the source tower to the destination using the middle tower that is not occupied (that is, you cannot use tower m).
3. Finally, move again the $n-k$ discs from the middle tower m to the destination tower.

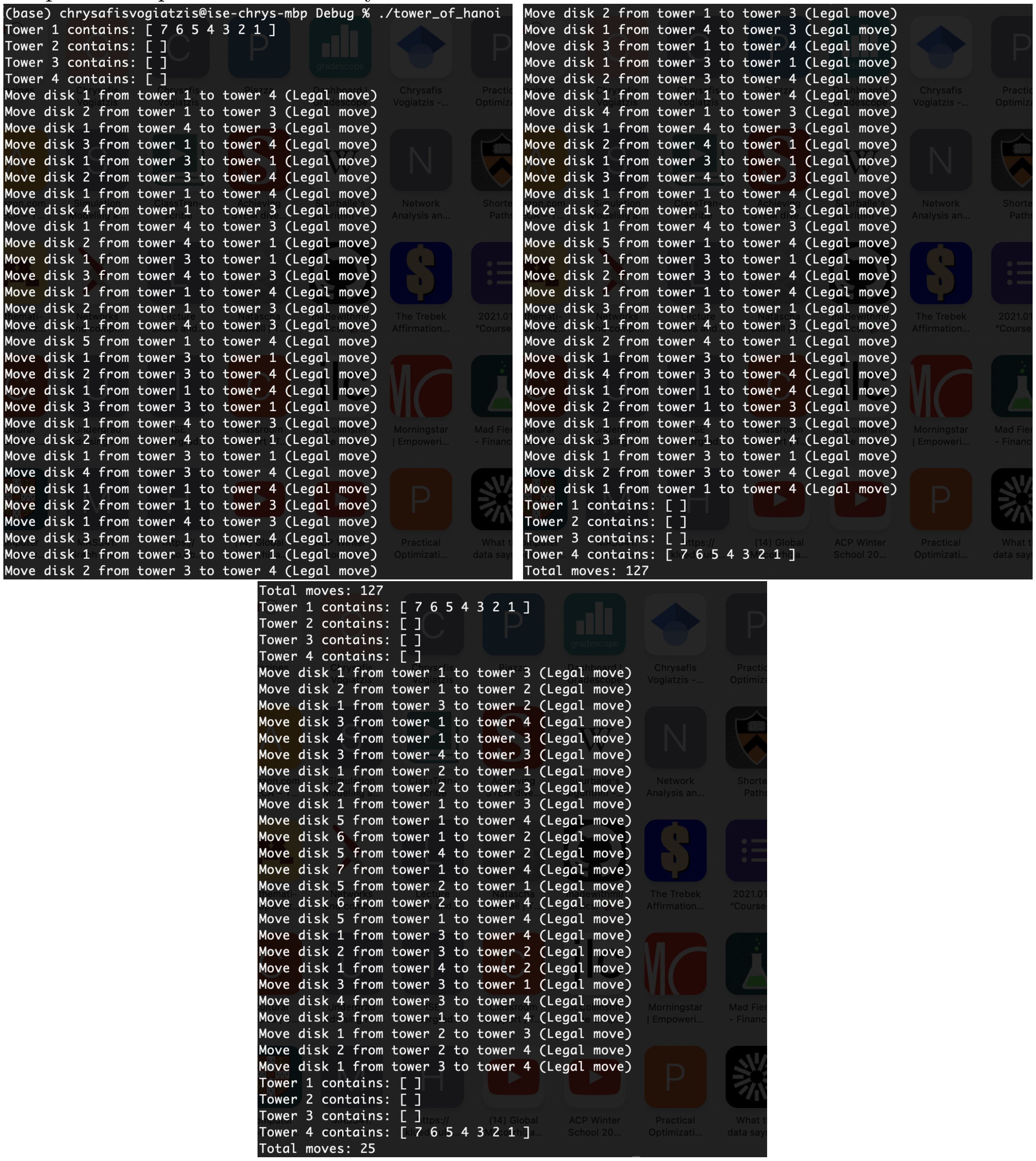
We need to clarify a couple of points. The second step is the same as moving discs from a source to a destination using a single middle tower! This is exactly the move we did (using recursion) in class! Secondly, we note how k is an unknown value. It can be any integer number between 1 and $n-1$. It can be shown that $k = \lfloor\sqrt{2n}\rfloor$ (that is the integer part of $\sqrt{2n}$, if you round the square root down) is optimal, so use that in your calculation.

Write a class Hanoi that allows for the creation of a Hanoi tower puzzle with multiple discs and 4 towers. The class should allow for solving the problem in a recursive manner. The class should also allow for counting the number of moves your recursive solution took. Finally, we want you to compare two approaches:

1. The number of moves to solve a 7 discs on 4 towers problem using the approach described here.
2. The number of moves to solve a 7 discs on 4 towers problem by using only the 3 towers (that is, using the code we saw in class).

What do you observe? We show an example (indicative) of the output expected in Figure 2.

Figure 2: The indicative output produced when using the `main.cpp` source code provided. It begins by printing the 4 towers (with all discs in tower 1 – the first tower), then it proceeds to make valid moves using only 3 towers, and finishes by printing that it required 127 moves. The second part of the code prints again the new re-initialized Hanoi tower (again, all discs are in the first tower), proceeds to make valid moves using all 4 towers, and finishes by printing that the process required 25 moves only!



```
(base) chrysafisvogiatzis@ise-chrys-mbp Debug % ./tower_of_hanoi
Tower 1 contains: [ 7 6 5 4 3 2 1 ]
Tower 2 contains: [ ]
Tower 3 contains: [ ]
Tower 4 contains: [ ]
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 2 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 3 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 2 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 4 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 2 from tower 4 to tower 1 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 3 from tower 4 to tower 3 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 2 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 5 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 2 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 3 from tower 3 to tower 1 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 2 from tower 4 to tower 1 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 4 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 2 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 3 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 2 from tower 3 to tower 4 (Legal move)
```

```
Move disk 2 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 3 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 2 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 4 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 2 from tower 4 to tower 1 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 3 from tower 4 to tower 3 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 2 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 5 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 2 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 3 from tower 3 to tower 1 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 2 from tower 4 to tower 1 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 4 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Move disk 2 from tower 1 to tower 3 (Legal move)
Move disk 1 from tower 4 to tower 3 (Legal move)
Move disk 3 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 1 (Legal move)
Move disk 2 from tower 3 to tower 4 (Legal move)
Move disk 1 from tower 1 to tower 4 (Legal move)
Tower 1 contains: [ ]
Tower 2 contains: [ ]
Tower 3 contains: [ ]
Tower 4 contains: [ 7 6 5 4 3 2 1 ]
Total moves: 127
```

```
Total moves: 127
Tower 1 contains: [ 7 6 5 4 3 2 1 ]
Tower 2 contains: [ ]
Tower 3 contains: [ ]
Tower 4 contains: [ ]
Move disk 1 from tower 1 to tower 3 (Legal move)
Move disk 2 from tower 1 to tower 2 (Legal move)
Move disk 1 from tower 3 to tower 2 (Legal move)
Move disk 3 from tower 1 to tower 4 (Legal move)
Move disk 4 from tower 1 to tower 3 (Legal move)
Move disk 3 from tower 4 to tower 3 (Legal move)
Move disk 1 from tower 2 to tower 1 (Legal move)
Move disk 2 from tower 2 to tower 3 (Legal move)
Move disk 1 from tower 1 to tower 3 (Legal move)
Move disk 5 from tower 1 to tower 4 (Legal move)
Move disk 6 from tower 1 to tower 2 (Legal move)
Move disk 5 from tower 4 to tower 2 (Legal move)
Move disk 7 from tower 1 to tower 4 (Legal move)
Move disk 5 from tower 2 to tower 1 (Legal move)
Move disk 6 from tower 2 to tower 4 (Legal move)
Move disk 5 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 4 (Legal move)
Move disk 2 from tower 3 to tower 2 (Legal move)
Move disk 1 from tower 4 to tower 2 (Legal move)
Move disk 3 from tower 3 to tower 1 (Legal move)
Move disk 4 from tower 3 to tower 4 (Legal move)
Move disk 3 from tower 1 to tower 4 (Legal move)
Move disk 1 from tower 2 to tower 3 (Legal move)
Move disk 2 from tower 2 to tower 4 (Legal move)
Move disk 1 from tower 3 to tower 4 (Legal move)
Tower 1 contains: [ ]
Tower 2 contains: [ ]
Tower 3 contains: [ ]
Tower 4 contains: [ 7 6 5 4 3 2 1 ]
Total moves: 25
```