CS 473 ✧ Spring 2020

Homework 0

Due Wednesday, January 29, 2020 at 9pm

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms; fundamental graph problems and algorithms; and especially facility with recursion and induction. Notes on most of this prerequisite material are available on the course web page.
- **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students will be allowed to submit joint solutions.
- **Submit your solutions electronically on Gradescope as PDF files.**
  - Submit a separate file for each numbered problem.
  - You can find a $\LaTeX$ solution template on the course web site (soon); please use it if you plan to typeset your homework.
  - If you plan to submit scanned handwritten solutions, please use dark ink (not pencil) on blank white printer paper (not notebook or graph paper), and use a high-quality scanner or scanning app to create a high-quality PDF for submission (not a raw cell-phone photo).

☞ Some important course policies ☜

- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- **Avoid the Deadly Sins!** There are a few common writing (and thinking) practices that will be automatically penalized on every homework or exam problem. We're not just trying to be scary control freaks; history strongly suggests people who commit these sins are more likely to make other serious mistakes as well. So we're trying to break bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Every algorithm requires an English specification.
  - Greedy algorithms require formal correctness proofs.
  - Never use weak induction. Weak induction should die in a fire.

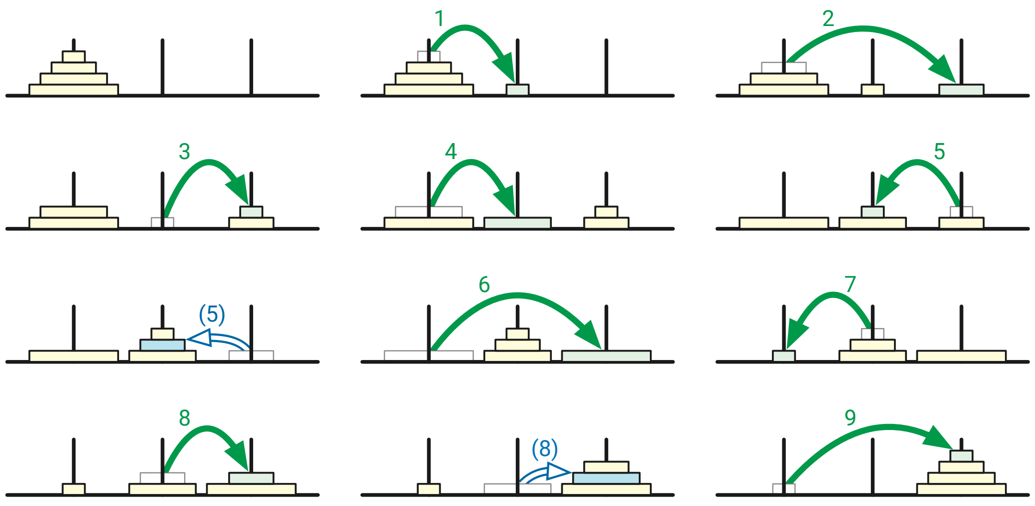
See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

---

1. The standard Tower of Hanoi puzzle consists of n circular disks of different sizes, each with a hole in the center, and three pegs. The disks are labeled $1, 2, \ldots, n$ in increasing order of size. Initially all n disks are on one peg, sorted by size, with disk n at the bottom and disk 1 at the top. The goal is to move all n disks to a different peg, by repeatedly moving individual disks. At each step, we are allowed to move the highest disk on any peg to any other peg, subject to the constraint that a larger disk is never placed above a smaller disk. A recursive strategy that solves this problem using exactly $2^n - 1$ moves is well known (and described in the textbook).

   This question concerns a variant of the Tower of Hanoi that I'll call the *Tower of Fibonacci*. In this variant, whenever any two disks $i-1$ and $i+1$ are adjacent, the intermediate disk i immediately teleports between them; otherwise, the setup, rules, and goal of the puzzle are unchanged. This teleport does *not* count as a move; on the other hand, the teleport is *not* optional. For example, the four-disk Tower of Fibonacci can be solved in nine moves (and two teleports, after moves 5 and 8) as follows:



   (a) Describe a recursive algorithm to solve the Tower of Fibonacci puzzle. Briefly justify why your algorithm is correct. (We don't need a complete formal proof of correctness; just convince us that you know why it works.)

   (b) How many *moves* does your algorithm perform, as a function of the number of disks? Prove that your answer is correct.

   (c) How many *teleports* does your algorithm induce, as a function of the number of disks? Prove that your answer is correct.

   For full credit, give *exact* answers to parts (b) and (c), not just $O(\,)$ bounds. Express your answers to parts (b) and (c) in terms of the Fibonacci numbers, defined as follows:

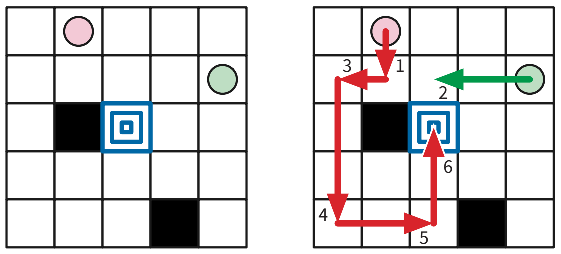
$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

2. Prove that every integer (positive, negative, or zero) can be written as the sum of distinct powers of −2. For example:

$$\begin{aligned}
17 &= (-2)^4 + (-2)^0 &&= 16 + 1 \\
-23 &= (-2)^5 + (-2)^4 + (-2)^3 + (-2)^0 &&= -32 + 16 - 8 + 1 \\
42 &= (-2)^6 + (-2)^5 + (-2)^4 + (-2)^3 + (-2)^2 + (-2)^1 &&= 64 - 32 + 16 - 8 + 4 - 2 \\
473 &= (-2)^{10} + (-2)^9 + (-2)^5 + (-2)^3 + (-2)^0 &&= 1024 - 512 - 32 - 8 + 1
\end{aligned}$$

[Hint: The empty set is a set, and weak induction should die in a fire.]

3. The famous puzzle-maker Kaniel the Dane invented a solitaire game played with two tokens on an $n \times n$ square grid. Some squares of the grid are marked as *obstacles*, and one grid square is marked as the *target*. In each turn, the player must move one of the tokens from its current position *as far as possible* upward, downward, right, or left, stopping just before the token hits (1) the edge of the board, (2) an obstacle square, or (3) the other token. The goal is to move either of the tokens onto the target square.



An instance of Kaniel the Dane's puzzle that can be solved in six moves.
Circles indicate initial token positions; black squares are obstacles; the center square is the target.

For example, we can solve the puzzle shown above by moving the red token down until it hits the obstacle, then moving the green token left until it hits the red token, and then moving the red token left, down, right, and up. The red token stops at the target on the 6th move *because* the green token is just above the target square.

Describe and analyze an algorithm to determine whether an instance of this puzzle is solvable. Your input consists of the integer n, a list of obstacle locations, the target location, and the initial locations of the tokens. The output of your algorithm is a single boolean: TRUE if the given puzzle is solvable and FALSE otherwise.

[Hint: Construct and search a graph. What are the vertices? What are the edges? Is the graph directed or undirected? Do the vertices or edges have weights? How long does it take to construct the graph? What problem do you need to solve on this graph? What textbook algorithm can you use to solve that problem? (Don't regurgitate the textbook algorithm; just point to the textbook!) What is the running time of that algorithm as a function of n?]