CS 473 ✧ Spring 2017

☙ Homework 0 ❧

Due Wednesday, January 25, 2017 at 8pm

- **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 must submit scanned handwritten solutions, please use dark ink (not pencil) on blank white printer paper (not notebook or graph paper), use a high-quality scanner (not a phone camera), and print the resulting PDF file on a black-and-white printer to verify readability before you submit.

☞ 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.
- The answer ***"I don't know"*** (and *nothing* else) is worth 25% partial credit on any problem or subproblem, on any homework or exam, except for extra-credit problems. We will accept synonyms like "No idea" or "WTF" or "¯\(•_•)/¯", but you must write *something*.
- **Avoid the Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an ***automatic zero*** on any homework or exam problem. Yes, really.
  - Always give complete solutions, not just examples.
  - Every algorithm requires an English specification.
  - Greedy algorithms require formal correctness proofs.
  - Never use weak induction.

See the course web site for more information.

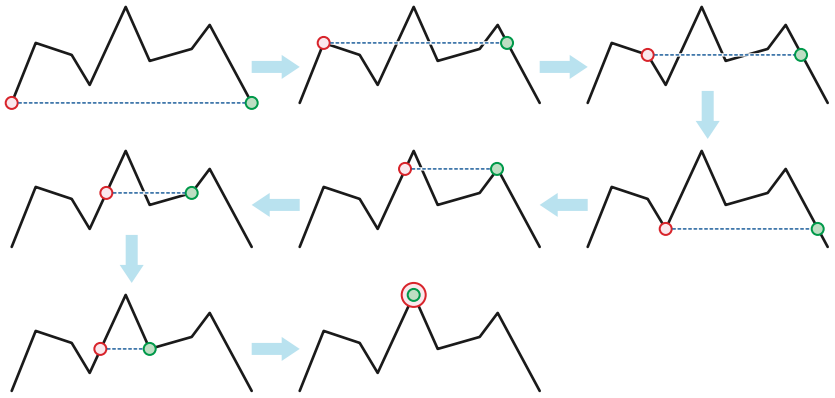
If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

---

1. Every cheesy romance movie has a scene where the romantic couple, after a long and frustrating separation, suddenly see each other across a long distance, and then slowly approach one another with unwavering eye contact as the music rolls in and the rain lifts and the sun shines through the clouds and the music swells and everyone starts dancing with rainbows and kittens and chocolate unicorns and. . . .[1]

   Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters their favorite park at the east and west entrances and immediately establish eye-contact. They can't just run directly to each other; instead, they must stay on the path that zig-zags through the part between the east and west entrances. To maintain the proper dramatic tension, Alice and Bob must traverse the path so that they always lie on a direct east-west line.

   We can describe the zigzag path as two arrays $X[0..n]$ and $Y[0..n]$, containing the x- and y-coordinates of the corners of the path, in order from the southwest endpoint to the southeast endpoint. The X array is sorted in increasing order, and $Y[0] = Y[n]$. The path is a sequence of straight line segments connecting these corners.



Alice and Bob meet. Alice walks backward in step 2, and Bob walks backward in steps 5 and 6.

   (a) Suppose $Y[0] = Y[n] = 0$ and $Y[i] > 0$ for every other index i; that is, the endpoints of the path are strictly below every other point on the path. Prove that under these conditions, Alice and Bob can meet.

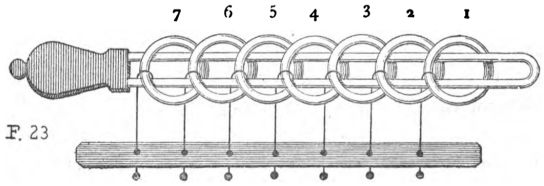
   [Hint: Describe a graph that models all possible locations and transitions of the couple along the path. What are the vertices of this graph? What are the edges? What can you say about the degrees of the vertices?]

   (b) If the endpoints of the path are *not* below every other vertex, Alice and Bob might still be able to meet, or they might not. Describe an algorithm to decide whether Alice and Bob can meet, without either breaking east-west eye contact or stepping off the path, given the arrays $X[0..n]$ and $Y[0..n]$ as input.

   *[Hint: Build the graph from part (a). (How?) What problem do you need to solve on this graph? Call a textbook algorithm to solve that problem. (Do **not** regurgitate the textbook algorithm.) What is your overall running time as a function of n?]*

[1] Fun fact: Damien Chazelle, the director of *Whiplash* and *La La Land*, is the son of Princeton computer science professor Bernard Chazelle.

2. The Tower of Hanoi is a relatively recent descendant of a much older mechanical puzzle known as the Chinese rings, Baguenaudier (a French word meaning "to wander about aimlessly"), Meleda, Patience, Tiring Irons, Prisoner's Lock, Spin-Out, and many other names. This puzzle was already well known in both China and Europe by the 16th century. The Italian mathematician Luca Pacioli described the 7-ring puzzle and its solution in his unpublished treatise *De Viribus Quantitatis*, written between 1498 and 1506;[2] only a few years later, the Ming-dynasty poet Yang Shen described the 9-ring puzzle as "a toy for women and children".



A drawing of a 7-ring Baguenaudier, from *Récréations Mathématiques* by Édouard Lucas (1891)

The Baguenaudier puzzle has many physical forms, but it typically consists of a long metal loop and several rings, which are connected to a solid base by movable rods. The loop is initially threaded through the rings as shown in the figure above; the goal of the puzzle is to remove the loop.

More abstractly, we can model the puzzle as a sequence of bits, one for each ring, where the ith bit is `1` if the loop passes through the ith ring and `0` otherwise. (Here we index the rings from right to left, as shown in the figure.) The puzzle allows two legal moves:

- You can always flip the 1st (= rightmost) bit.
- If the bit string ends with exactly i `0`s, you can flip the $(i+2)$th bit.

The goal of the puzzle is to transform a string of n `1`s into a string of n `0`s. For example, the following sequence of 21 moves solve the 5-ring puzzle:

$$\texttt{11111} \xrightarrow{1} \texttt{11110} \xrightarrow{3} \texttt{11010} \xrightarrow{1} \texttt{11011} \xrightarrow{2} \texttt{11001} \xrightarrow{1} \texttt{11000} \xrightarrow{5} \texttt{01000}$$
$$\xrightarrow{1} \texttt{01001} \xrightarrow{2} \texttt{01011} \xrightarrow{1} \texttt{01010} \xrightarrow{3} \texttt{01110} \xrightarrow{1} \texttt{01111} \xrightarrow{2} \texttt{01101} \xrightarrow{1} \texttt{01100} \xrightarrow{4} \texttt{00100}$$
$$\xrightarrow{1} \texttt{00101} \xrightarrow{2} \texttt{00111} \xrightarrow{1} \texttt{00110} \xrightarrow{3} \texttt{00010} \xrightarrow{1} \texttt{00011} \xrightarrow{2} \texttt{00001} \xrightarrow{1} \texttt{00000}$$

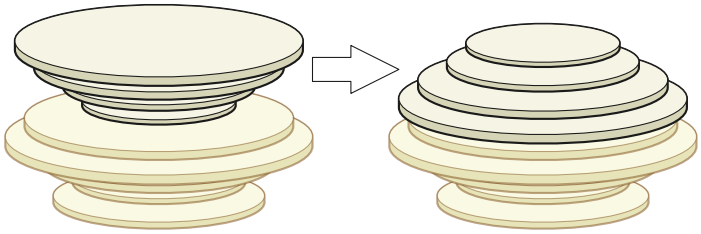
(a) Describe an algorithm to solve the Baguenaudier puzzle. Your input is the number of rings n; your algorithm should print a sequence of moves that solves the n-ring puzzle. For example, given the integer 5 as input, your algorithm should print the sequence $1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1$.

(b) *Exactly* how many moves does your algorithm perform, as a function of n? Prove your answer is correct.

(c) ***[Extra credit]*** Call a sequence of moves *reduced* if no move is the inverse of the previous move. Prove that for any non-negative integer n, there is *exactly one* reduced sequence of moves that solves the n-ring Baguenaudier puzzle. *[Hint: See problem 1!]*

[2] *De Viribus Quantitatis* [*On the Powers of Numbers*] is an important early work on recreational mathematics and perhaps the oldest surviving treatise on magic. Pacioli is better known for *Summa de Aritmetica*, a near-complete encyclopedia of late 15th-century mathematics, which included the first description of double-entry bookkeeping.

3. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.



Flipping the top four pancakes.

   (a) Describe an algorithm to sort an arbitrary stack of n pancakes using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

   (b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of n pancakes, so that the burned side of every pancake is facing down, using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

*[Hint: This problem has **nothing** to do with the Tower of Hanoi!]*