CS 473: Fundamental Algorithms, Spring 2013
HW 0 (due Tuesday, at noon, January 22, 2013)

Version: 1.12.

This homework contains two problems. **Read the instructions for submitting homework on the course webpage**.

$\boxed{\boxed{\textbf{You also have to do quiz 0 online!}}}$

Collaboration Policy: For this homework, each student should work independently and write up their own solutions and submit them.

Read the course policies before starting the homework.

- Homework 0 and Quiz 0 test your familiarity with prerequisite material: big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction, to help you identify gaps in your background knowledge. You are responsible for filling those gaps. The course web page has pointers to several excellent online resources for prerequisite material. If you need help, please ask in headbanging, on Piazza, in office hours, or by email.
- Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
- Please carefully read the course policies on the course web site. If you have any questions, please ask in lecture, in headbanging, on Piazza, in offi ce hours, or by email. In particular:
  - Submit separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page, in the corresponding drop boxes outside 1404 Siebel.
  - You may use any source at your disposal: paper, electronic, human, or other, but you must write your solutions in your own words, and you must cite every source that you use (except for official course materials). Please see the academic integrity policy for more details.
  - No late homework will be accepted for any reason. However, we may forgive quizzes or homeworks in extenuating circumstances; ask Jeff for details.
  - Answering "I don't know" to any (non-extra-credit) problem or subproblem, on any homework or exam, is worth 25% partial credit.
  - Algorithms or proofs containing phrases like and so on or repeat this process for all n, instead of an explicit loop, recursion, or induction, will receive a score of 0.
  - Unless explicitly stated otherwise, every homework problem requires a proof.

---

1 Required problems

1. (50 PTS.) The multi-trillion dollar game.

You are given k piles having in total n identical coins (each coin is a trillion dollar coin, of course, see picture on the right). In every step of the game, you take a coin from a pile P, and you can put it in any pile Q, if Q has at least 2 less coins than P. In particular, if P has two coins, you are allowed to take one of the coins of P and create a new pile. If a pile has only a single coin, you are allowed to take the coin and remove it from the game[1]. The game is over when there are no more coins left. Note, that you can start a new pile by taking a coin from any pile that has at least 2 coins. As an example, consider the following sequence of moves in a game:

$$\begin{aligned}\{4,2,1\} &\implies \{4,1,1,1\} \implies \{3,2,1,1\} \implies \{2,2,2,1\} \implies \{2,2,1,1,1\}\\ &\implies \{2,1,1,1,1,1\} \implies \{2,1,1,1,1\} \implies \{2,1,1,1\} \implies \{2,1,1\}\\ &\implies \{2,1\} \implies \{2\} \implies \{1,1\} \implies \{1\} \implies \{\}.\end{aligned}$$

(A) (25 PTS.) Prove, formally, that this game always terminates after a finite number of steps.

(B) (25 PTS.) Unfortunately, the world economy is suffering from hyper-inflation. A trillion dollar bucks do not go as far as they used to go. To keep you interested in the game, the rules have changed – every time you remove a coin from a pile, you are required to insert two coins into two different piles (again, these new piles need to have 2 less coins than the original pile before the move) [you get the extra coin for such a move from the IMF]. Again, if a pile has a single coin, you can take the coin. Prove, formally, that this game always terminates after a finite number of steps. A valid game in this case might look like:

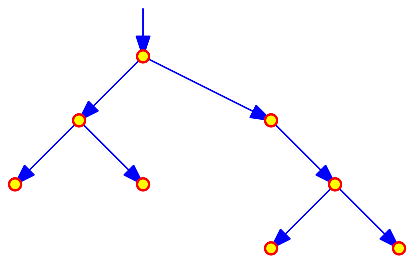
$$\begin{aligned}\{4,2,1\} &\implies \{3,3,2\} \implies \{3,2,2,1,1\} \implies \{3,2,2,1\} \implies \{3,2,2\}\\ &\implies \{3,2,1,1,1\} \implies \{2,2,2,2,1\} \implies \{2,2,2,2\} \implies \{2,2,2,1,1,1\}\\ &\implies \{2,2,2,1,1\} \implies \{2,2,2,1\} \implies \{2,2,2\} \implies \{2,2,1,1,1\}\\ &\implies \{2,2,1,1\} \implies \{2,2,1\} \implies \{2,2\} \implies \{2,1,1,1\}\\ &\implies \{1,1,1,1,1,1\} \implies \{1,1,1,1,1\} \implies \{1,1,1,1\} \implies \{1,1,1\}\\ &\implies \{1,1\} \implies \{1\} \implies \{\}.\end{aligned}$$

2. (50 PTS.) The baobab tree.

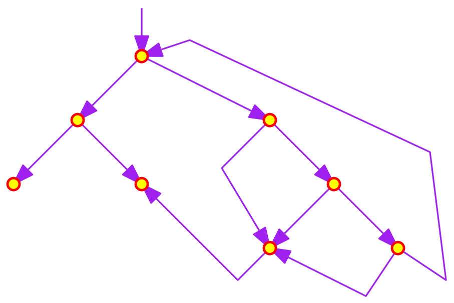
You are given a pointer to the root of a binary tree. The tree is stored in a read only memory, and you can not modify it in any way. Now, normally, each node in the tree has a pointer to its left child, right child, and its parent (these pointers are `NULL` if they have nothing to point to). Unfortunately, your tree might have been corrupted by a bug in somebody else code[2], so that some of the pointers, now either point to some other node in the tree or contain `NULL`,

[1] And then you can buy whatever banana republic you want with it.

[2] After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?



(A) Good tree.



(B) A corrupted tree.

Figure 1: A good & bad trees. For clarity, the figure does show the parent pointers that each node has
.

or vice versa (i.e., potentially any pointer in the tree [or potentially all of them] might be corrupted [including parent, left and right pointers]). See Figure 1.

Describe an algorithm[3] that determines whether the tree is corrupted or not. Your algorithm must not modify the tree. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the tree, and use $O(1)$ extra space (not counting the tree itself).

Observe that a regular recursive algorithm uses a stack, and the space it uses for the stack might be unbounded if the depth of the recursion is not bounded by a constant. Similarly, using arrays or stacks of unbounded size violates the requirement that the space used is $O(1)$.

[3]Since you've read the Homework Instructions, so you know what the phrase 'describe an algorithm' means. Right?