Problem Set 2

This second problem set is all about induction and the sheer breadth of applications it entails. By the time you're done with this problem set, you will have a much deeper understanding of how to think inductively.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 150 possible points. It is weighted at 7% of your total grade. The earlier questions serve as a warm-up for the later problems, so the difficulty of the problems increases over the course of this problem set.

Good luck, and have fun!

Checkpoint Questions Due Monday, October 7 at 2:15 PM
Remaining Questions Due Friday, October 11 at 2:15 PM

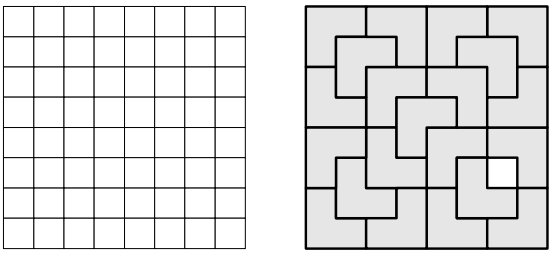
Write your solutions to the following problems and submit them by this Monday, October 7th at the start of lecture. As before, these problems are graded on a 0/12/25 scale based on whether or not you have made a good, honest effort to complete the problems. We will try to get these problems returned to you with feedback on your proof style this Wednesday, October 9th.

Please make the best effort you can when solving this problem. We want the feedback we give you on your solutions to be as useful as possible, so the more time and effort you put into them, the better we'll be able to comment on your proof style and technique.

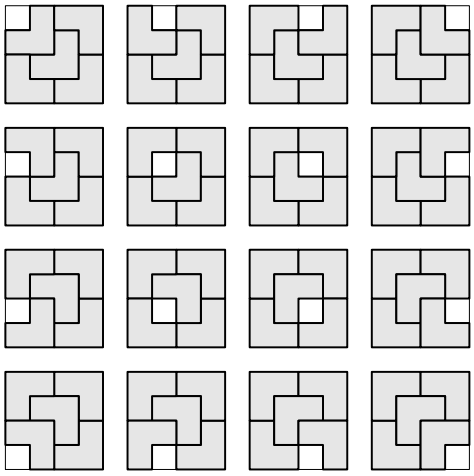
Checkpoint Question: Tiling with Triominoes* (25 Points if Submitted)

Recall from Problem Set One that a *right triomino* is an L-shaped tile that looks like this:

Suppose that you are also given a square grid of size $2^n \times 2^n$ and want to *tile* it with right triominoes by covering the grid with triominoes such that all triominoes are completely on the grid and no triominoes overlap. Here's an attempt to cover an 8×8 grid with triominoes, which fails because not all squares in the grid are covered:



Amazingly, it turns out that it is always possible to tile any $2^n \times 2^n$ grid that's missing exactly one square with right triominoes. It doesn't matter what n is or which square is removed; there is always a solution to the problem. For example, here are all the ways to tile a 4×4 grid that has a square missing:



Prove that any $2^n \times 2^n$ grid with one square removed can be tiled by right triominoes. (*Hint: Split the board up into four smaller boards.)*

---

* I originally heard this problem from David Gries of Cornell University.

The remainder of these problems should be completed and returned by Friday, October 11th at the start of class.

Problem One: Recurrence Relations (16 Points)

A *recurrence relation* is a recursive definition of the terms in a sequence. Typically, a recurrence relation specifies the value of the first few terms in a sequence, then defines the remaining terms from the previous terms.

For example, the *Fibonacci sequence* can be defined by the following recurrence relation:

$$F_0 = 0$$

$$F_1 = 1$$

$$F_{n+2} = F_n + F_{n+1}$$

The first terms of this sequence are $F_0 = 0$, $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, $F_6 = 8$, etc.

Some recurrence relations define well-known sequences. For example, consider the following recurrence relation:

$$a_0 = 1$$

$$a_{n+1} = 2a_n$$

The first few terms of this sequence are 1, 2, 4, 8, 16, 32, …, which happen to be powers of two.

i. Prove by induction that for any $n \in \mathbb{N}$, we have $a_n = 2^n$.

Minor changes to the recursive step in a recurrence relation can lead to enormous changes in what numbers are generated. Consider the following two recurrence relations, which are similar to the a_n sequence defined above but with slight changes to the recursive step:

$$b_0 = 1 \qquad\qquad c_0 = 1$$
$$b_{n+1} = 2b_n - 1 \qquad\qquad c_{n+1} = 2c_n + 1$$

ii. Find non-recursive definitions for b_n and c_n, then prove by induction that your definitions are correct.

Finding non-recursive definitions for recurrences (often called "solving" the recurrence) is useful in the design and analysis of algorithms. Commonly, when trying to analyze the runtime of an algorithm, you will arrive at a recurrence relation describing the runtime on an input of size n in terms of the runtime on inputs of smaller sizes. Solving the recurrence then lets you precisely determine the runtime. To learn more, take CS161, Math 108, or consider reading through *Concrete Mathematics* by Graham, Knuth, and Patashnik.

Problem Two: Nim (20 points)

Nim is a family of games played by two players. The game begins with several piles of stones that are shared between the two players. Players alternate taking turns removing any nonzero number of stones from any single pile of their choice. If at the start of a player's turn all the piles are empty, then that player loses the game.

Prove, by induction, that if the game is played with two piles of stones, each of which begins with the same number of stones, then the second player can always win the game if she plays correctly.

Problem Three: Contract Rummy (16 points)

Contract rummy is a card game for any number of players in which players are dealt a hand of cards and, through several iterations of drawing and discarding cards, need to accumulate *sets* and *sequences*. A set is a collection of three cards of the same value, and a sequence is a collection of four cards of the same suit that are in ascending order. The game proceeds in multiple rounds in which players need to accumulate a different number of sets and sequences. The rounds are:

- Two sets (six cards)
- One set, one sequence (seven cards)
- Two sequences (eight cards)
- Three sets (nine cards)
- Two sets and a sequence (ten cards)
- One set and two sequences (eleven cards)
- Three sequences (twelve cards)

Notice that in each round, the requirements are such that the number of cards required increases by one. It's interesting that it's always possible to do this, since the total number of cards must be made using just combinations of three cards and four cards.

Prove, by induction, that any natural number greater than or equal to six can be written as $3m + 4n$ for some natural numbers m and n. As a reminder, since m and n are natural numbers, it's important to show that $m \geq 0$ and $n \geq 0$.

Problem Four: Colored Cubes* (24 Points)

Suppose that you have cubes of n different colors. Collectively, you have a total of kn cubes for some natural number k. For example, you might have 30 cubes of six different colors, with $n = 6$ and $k = 5$. Alternatively, you might have 120 cubes of 30 different colors, with $n = 30$ and $k = 4$.

i. Prove that if $n \geq 1$, then there must be some color such that there are at least k cubes of that color and some color such that there are at most k cubes of that color. Note that these might be the same color. (You don't need to prove this by induction.)

ii. Prove by induction on n that for any $n \geq 0$, it is always possible to distribute the cubes into n boxes such that both of the following statements are true:

   - Each box contains exactly k cubes, and
   - Each box contains cubes of at most two different colors.

   (Hint: We suggest working through some examples and seeing if you spot a pattern before trying to prove this result. As a hint, inducting on n means that you should try to see if you can find a way to use up all cubes of one color to reduce the number of colors from $n+1$ to n.)

The result that you have proved in part (ii) forms the basis for the *alias method*, a fast algorithm for simulating rolls of a loaded die. This has applications in machine learning (simulating different outcomes of a random event for reinforcement learning), operating systems (allocating CPU time to processes with different needs), and computational linguistics (generating random sentences based on differently-weighted rules).

---

* This problem adapted from Exercise 3.4.1.7 of *The Art of Computer Programming, Third Edition, Volume II: Seminumerical Algorithms* by Donald Knuth.

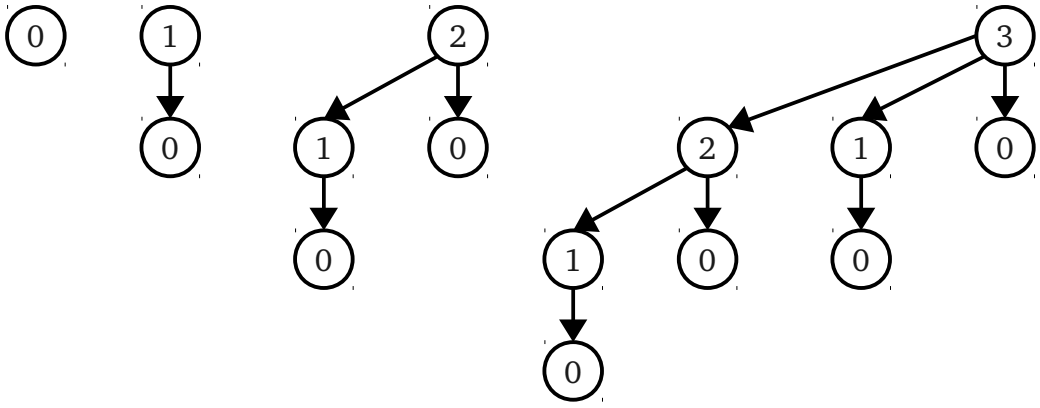
Problem Five: Binomial Trees (16 Points)

A ***directed tree*** is a special kind of directed graph with the following properties:

- There is a special node called the ***root node*** with no incoming edges.
- Every node other than the root node has exactly one incoming edge.

If a node u has an edge to a node v, we say that u is the ***parent*** of v. Similarly, v is a ***child*** of u.

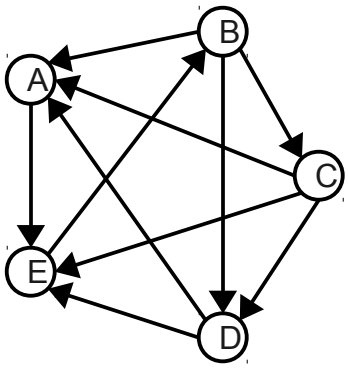
Binomial trees are a specific family of directed trees defined as follows: a binomial tree of order n is a single node with n children, which are binomial trees of order 0, 1, 2, …, $n - 1$. For example, here are pictures of binomial trees of orders 0, 1, 2, and 3:



Prove by induction that a binomial tree of order n has exactly 2^n nodes.

Binomial trees are used as a building block in the *binomial heap* data structure, which can be used to efficiently determine the smallest element out of a group of values. If you're interested in learning more about the binomial heap and its applications, consider picking up a copy of *Introduction to Algorithms, Second Edition* by Cormen, Leiserson, Rivest, and Stein or taking CS166.

Problem Six: Tournament Winners (28 points)



A *tournament* is a contest among $n > 0$ players. Each player plays a game against each other player, and either wins or loses the game (let's assume that there are no draws). A *tournament graph* is a graph representing the result of a tournament, where each node corresponds to a player and each edge (u, v) means that player u won her game against player v. A tournament for five players is shown to the left.

A *tournament winner* is a player in a tournament who, for each other player, either won her game against that player, or won a game against a player who in turn won against that player. For example, in the adjacent graph, B, C, and E are tournament winners. However, D is ***not*** a tournament winner, because he neither won against player C, nor won against anyone who in turn won against C. Although D won against E, who in turn won against B, who then won against C, under our definition D is not a tournament winner.

Prove, by induction, that every tournament graph has a tournament winner.

Problem Seven: Course Feedback (5 Points)

We want this course to be as good as it can be, and we'd really appreciate your feedback on how we're doing. For a free five points, please answer the following questions. We'll give you full credit no matter what you write (as long as you write something!), but we'd appreciate it if you're honest about how we're doing.

i. How hard did you find this problem set? How long did it take you to finish? Does that seem unreasonably difficult or time-consuming for a five-unit class?

ii. Do you think we should keep using Google Moderator to let people ask questions in lecture?

iii. Did you read the "Guide to Proofs" handout last week? If so, did you find it useful?

iv. How is the pace of this course so far? Too slow? Too fast? Just right?

v. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?

Extra Credit Problem: Egyptian Fractions (5 Points Extra Credit)

The Fibonacci sequence is named after Leonardo Fibonacci, an eleventh-century Italian mathematician who is credited with introducing Hindu-Arabic numerals (the number system we use today) to Europe in his book *Liber Abaci*. This book also contained an early description of the Fibonacci sequence, from which the sequence takes its name. *Liber Abaci* also described a notation for fractions called *Egyptian fractions*, a method for writing out fractions that has been employed since ancient times. An Egyptian Fraction is a sum of distinct fractions whose numerators are all one (these fractions are called *unit fractions*). For example:

$$\frac{2}{3} = \frac{1}{2} + \frac{1}{6} \qquad \frac{2}{15} = \frac{1}{10} + \frac{1}{30} \qquad \frac{7}{15} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120} \qquad \frac{2}{85} = \frac{1}{51} + \frac{1}{255}$$

One way of finding an Egyptian fraction representation of a rational number is to use a *greedy algorithm* that works by finding the largest unit fraction at any point that can be subtracted out from the rational number. For example, to compute the fraction for 42 / 137, we would start off by noting that 1 / 4 is the largest unit fraction less than 42 / 137. We then say that

$$\frac{42}{137} = \frac{1}{4} + \left(\frac{42}{137} - \frac{1}{4}\right) = \frac{1}{4} + \frac{31}{548}$$

We then repeat this process by finding the largest unit fraction less than 31 / 548 and subtracting it out. This number is 1/18, so we get

$$\frac{42}{137} = \frac{1}{4} + \left(\frac{42}{137} - \frac{1}{4}\right) = \frac{1}{4} + \frac{1}{18} + \left(\frac{31}{548} - \frac{1}{18}\right) = \frac{1}{4} + \frac{1}{18} + \frac{5}{4{,}932}$$

The largest unit fraction we can subtract from 5 / 4932 is 1 / 987:

$$\frac{42}{137} = \frac{1}{4} + \frac{1}{18} + \left(\frac{5}{4{,}932} - \frac{1}{987}\right) = \frac{1}{4} + \frac{1}{18} + \frac{1}{987} + \frac{1}{1{,}622{,}628}$$

And at this point we're done, because the leftover fraction is itself a unit fraction.

Prove that the greedy algorithm for Egyptian fractions always terminates for any rational number q in the range $0 < q < 1$ and always produces a valid Egyptian fraction. That is, the sum of the unit fractions should be the original number, and no unit fraction should be repeated. This shows that every rational number in the range $0 < q < 1$ has at least one Egyptian fraction representation.