CS 4349.003.19F Lecture 2—August 21, 2019

Main topics are #induction and #recursion including #example/prime_divisors and #example/tower_of_Hanoi.

Prime Divisors

- Today, we're going to discuss two connected but important topics, induction and recursion.
- This material should technically be review, but I may motivate it and explain it in a way you haven't seen before. The reading on induction has many many more examples that you should look over until you're really comfortable with it. We'll be doing **a lot** of induction proofs this semester.
- To help motivate what induction is, we're going to iterate over a couple proofs of a fundamental theorem in algebra (but not **the** fundamental theorem of algebra).
- Let n be a positive integer.
- A *divisor* of n is a positive integer p such that n / p is also an integer.
- Positive integer n is *prime* if it has exactly two divisors, n and 1.
- n is *composite* if it has more than two divisors.
- So, if n = 1, then it is neither prime nor composite.
- Theorem: Every integer n greater than 1 has a prime divisor.
- This is a universally quantified statement. We need to prove it about *all* the integers greater than one. And there are two ways to do that.

Direct proof: Let n be an arbitrary integer greater than 1.

. . . blah blah blah . . .

Thus, n has at least one prime divisor. □

Proof by contradiction: For the sake of argument, assume there is an integer greater than 1 with no prime divisor.
Let n be an arbitrary integer greater than 1 with no prime divisor.

. . . blah blah blah . . .

But that's just silly. Our assumption must be incorrect. □

- Proofs by contradiction are usually easier to discover, so let's start there.
- And to make things easier, we're going to pick a very specific counterexample. The *smallest* one.
- Proof by contradiction:
  - For the sake of argument, assume there is an integer greater than 1 with no prime divisor.
  - Let n be **the smallest** integer greater than 1 with no prime divisor.
    - Since n is a divisor of n and n has no prime divisors, n cannot be prime.

        - Thus, n must have at least one divisor d such that 1 < d < n.
    - Let d be an arbitrary divisor of n such that 1 < d < n.
        - **Because n is the smallest counterexample, d has a prime divisor.**
    - Let **p** be a prime divisor of d.
        - Because d / p is an integer, (n / d) * (d / p) = n / p is also an integer.
        - Thus, p is also a divisor of n.
        - But this contradicts our assumption that n has no prime divisors!
    - So our assumption must be incorrect.
- Great, we have a proof!

- But we can do better. Contradiction proofs are easy to write, but direct proofs are much easier to read. And when your readers include the people grading your homework, you probably want to make things easy to read.
- But something to notice about that previous proof: When we assumed n is the smallest counterexample, we, uh, assumed there are no smaller counterexamples.
- So let's just directly state our assumption that there are no smaller counterexamples and use it in a direct proof.
- Direct proof: Let n be an arbitrary integer greater than 1.
    - **Assume that for every integer k such that 1 < k < n, integer k has a prime divisor.**
    - There are two cases to consider: Either n is prime, or n is composite.
    - Suppose n is prime.
        - Then n is a prime divisor of n.
    - Suppose n is composite.
        - Then n has a divisor d such that 1 < d < n.
        - Let d be a divisor of n such that 1 < d < n.
        - By assumption (i.e. there are no smaller counterexamples), d has a prime divisor.
        - Let p be a prime divisor of d.
        - Because d / p is an integer, (n / d) * (d / p) = n / p is also an integer.
        - Thus, p is also a divisor of n.
    - In each case, we conclude that n has a prime divisor.
- This is called *proof by induction*.
- The assumption that there are no counterexamples smaller than n is called the *induction hypothesis*.
- The case we argued directly is called a *base case*. Yes, *every* prime number is a base case. While it is *usually* the true, nothing says the base cases have to be small. And there may even be an infinite number of them.
- The other case was the *inductive case*.
- Maybe you want to label the induction hypothesis, the base case(s), and the inductive case(s). That's up to you and how comfortable you feel doing these proofs.

- But until you're very comfortable doing these proofs, you do need to explicitly write out your induction hypothesis and make the case analysis obviously exhaustive.
- Again, I want to emphasize there is very little difference between this proof by induction and a proof by smallest counterexample except in readability.
- In a sense, our original argument relied on the observation that if n has no prime divisor, then there must be some smaller positive integer greater than 1 with no prime divisor.
- The direct proof relies on the contrapositive of this observation, "that smaller integer has a prime divisor => n has a prime divisor"

Proofs by Induction

- So what's the recipe here?
  1. **Write down the boilerplate (template).** Write down the universal invocation "Let n be an arbitrary…", the induction hypothesis, the conclusion, and leave blank space for the remaining details. **This is the easy part.** In fact, I'll just leave this here.

Theorem: $P(n)$ for every positive integer n.

Proof by induction: Let n be an arbitrary positive integer.
Assume that $P(k)$ is true for every positive integer $k < n$.
There are several cases to consider:

- Suppose n is *. . . blah blah blah . . .*
  Then $P(n)$ is true.
- Suppose n is *. . . blah blah blah . . .*
  The inductive hypothesis implies that *. . . blah blah blah . . .*
  Thus, $P(n)$ is true.

In each case, we conclude that $P(n)$ is true. □

or more abstractly:

Bellman's Theorem: Every snark is a boojum.

Proof by induction: Let X be an arbitrary snark.
Assume that for every snark younger than X is a boojum.
There are three cases to consider:

- Suppose X is the youngest snark.
  Then *. . . blah blah blah . . .*
- Suppose X is the second-youngest snark.
  Then *. . . blah blah blah . . .*
- Suppose X is older than the second-youngest snark.
  Then the inductive hypothesis implies *. . . blah blah blah . . .* and therefore *. . . blah blah blah . . .*

An all cases, we conclude that X is a boojum. □

  2. **Think big.** Don't think about small numbers like 1 or 5 or 10^100. And DO NOT think how to solve the problem all the way down to the ground. Think about how to reduce proofs about gigantic numbers (the inductive case) to claims about some other

smaller number or numbers. Seriously, the key thing that makes this all possible for a human is that we can compartmentalize. In a sense, we reduced the proof for n to claims for k < n. We don't have to worry about why things are true for those k, so just trust they are.**This is the hard part.**

3. **Look for holes.** Look for cases where your inductive argument breaks down (the base cases) and solve them directly. Don't be clever; be stupid but thorough. Doing base cases *after* the inductive cases makes it much easier to verify you really did handle the right set of base cases.
4. **Rewrite everything.** You probably didn't leave the correct amount of space. Rewrite the proof so the argument is easier for someone (the TA) to follow.

- All cases in induction proofs are either inductive cases or base cases. Base cases are *usually* a few small values of n, but they may be other things like all prime numbers. Induction proofs are usually easier to read if they describe the base cases first, but you should attempt to figure out the inductive cases first so you know which gaps need to be filled in with base cases.
- Until you are so comfortable with induction that you can just nod along thinking "ah yes, of course" whenever I make a claim based on an inductive assumption, I highly recommend you stick to full boilerplate for your proofs. **Write it down before you even start thinking.** I may even require the boilerplate for Homework 1.
- This boilerplate captures every kind of induction you might do. Claims about integers. Claims about graphs (where n is the number of vertices or edges). *Anything*.
- But here are two things you should **never, ever do** even if you're careful enough to make them technically correct.
- **Do not** make the induction hypothesis refer to k = n - 1. Use all k < n just in case you need them. Why would you tie n - 2 arms behind your back?
- **Do not** Assume the theorem for n and then attempt a proof for n + 1, especially when arguing about something "structural" like graphs. In doing so, you'll be tempted to build the (n + 1)st object instead of arguing about an arbitrary object directly. Now you have to prove that you were able to create any arbitrary object using your construction which is more work, harder to follow, and much easier to mess up.

Tower of Hanoi

- Alright, so induction is hard. Let's take a mental break. I borrowed this toy my kid used to play with. Maybe we can play with it too!
- The toy has three wooden pegs and a few disks. There's a puzzle we can try to solve that uses this toy called the Tower of Hanoi.
- French recreational mathematician Éduoard Lucas described the puzzle in 1883. A year later, Henri de Parville described the puzzle again, but after giving a remarkable backstory.

- In short, the great temple at Benares contains a version of this toy with three diamond needles. There are sixty-four golden disks on one of the needles stacked from biggest on bottom to smallest on top. Monks at the temple will move disks from one peg to another, but they may only move one disk at a time from needle to needle, and they may never stack a larger disk on top of a smaller one. When they move all the disks from the starting needle to a different needle, the temple will crumble to dust, and the world will end.
- That's pretty concerning! But do we really need to worry?
- Theorem: Moving n disks from one needle to another takes 2^n - 1 moves.
- Unsurprisingly, proving this theorem is going to require induction.
- But let's go one step further. To prove the theorem, it's actually going to be easiest to describe an *algorithm* for moving all the disks.
- And that means using the algorithmic analogue to induction which we call *recursion*.

Recursion

- On Monday, we discussed reductions, writing an algorithm for problem X by using an algorithm for a problem Y in a black-box manner.
- But there's nothing saying X and Y have to be different problems!
- *Recursion* is a special type of reduction where both X and Y are the same problem. So just like induction proofs,
  - if the problem instance can be solved directly, solve it directly.
  - Otherwise, reduce the problem instance to one or more **smaller instances of the same problem**.
- Remember, you want to treat the algorithm for Y as a black box without needing to understand how it works. That probably seems counterintuitive since we're writing the algorithm for X and Y right now.
- But it really is helpful for both designing and describing the algorithm to imagine that somebody else took your smaller instance and solved it for you. Following Erickson's lead, we'll call this person the *Recursion Fairy*.
- All the Recursion Fairy asks is that you give them a smaller instance of the problem. Then, they'll take care of things using methods **THAT ARE NONE OF YOUR BUSINESS**. Is it magic? Who knows? (other than the Recursion Fairy)
- Mathematically, it's the same as applying the induction hypothesis in an inductive proof. The simpler statement is just true, OK?

Tower of Hanoi Continued

- So how can we solve Tower of Hanoi recursively?
- It all comes down to two observations:

1. We need to move the n - 1 smaller disks off the largest disk so we can move the largest disk to the destination peg.
2. The location of the largest disk has no effect on the movement of the other n - 1 disks.

- Ah, so we can…
  - *recursively* move the smaller n - 1 disks off the bigger one, somehow
  - move the biggest disk
  - *recursively* stack the smaller n - 1 disks back on top, somehow
- Great! Now, remembering what I said about reductions and recursion and black boxes, what do we need to figure out and describe to move the n - 1 disks around?
- NOTHING. We ask the Recursion Fairy to move the n -1 disks **camera tricks**. Then we can move the biggest disk, and then we ask them to move the n - 1 disks **camera trick**. DONE. DON'T ASK WHAT HAPPENED WHEN I FROZE THE CAMERA. The Recursion Fairy is very shy. And they have a very large team of vicious IP attorneys.
- Our algorithm should work for any non-negative value n (moving zero disks around is still well-defined). So, like induction, we still have to figure out the *base cases* where our recursion strategy doesn't work. What cases are we missing here?
- Right, this algorithm does not make sense when n = 0, because there is no largest disk and n - 1 = -1. In this base case, we need to move the 0 disks from one peg to the other. In other words, we do nothing.
- OK, but just talking at you isn't the best way to describe the algorithm. Here's the pseudocode. Hanoi takes the number of disks, the source peg, the destination peg, and the other peg as parameters. Disks are numbered 1 to n in increasing order of size. The base case is taken care of just by skipping past the if block:

```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)    ⟨⟨Recurse!⟩⟩
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)    ⟨⟨Recurse!⟩⟩
```

- I really really want to emphasize that **we do not need to worry about what happens during those Hanoi(n - 1, …) calls**. Our **only** task is to reduce to a simpler instance. For even slightly more complicated problems, if we try to ask what happens in the simpler instance, it's only going to make things more confusing than they already are.
- So great. We have an algorithm for moving the disks, but we still need to prove we have the correct number of moves.
- Theorem: Hanoi(n, src, dst, tmp) moves n disks from src to dst in 2^n - 1 moves.
- Proof by induction:
  - Assume Hanoi(k, src, dst, tmp) moves k disks in 2^k - 1 moves from src to dst for all non-negative k < n.
  - If n = 0, Hanoi correctly moves nothing in 0 = 2^0 - 1 moves.

    - Suppose n = 1.
        - By the induction hypothesis, Hanoi(n - 1, src, tmp, dst) moves the top n - 1 disks from src to tmp.
        - src and dst are free of smaller disks, so we successfully move disk n from src to dst in one move.
        - By the induction hypothesis, Hanoi(n - 1, tmp, dst, src) moves the top n - 1 disks from tmp to dst and we are done.
        - The total number of moves is 2 * (2^{n - 1} - 1) + 1 = 2^n - 1.
- And what does this mean for the end of the world? Assuming I can move one disk per second, it will take just over two minutes to solve this toy. Had I brought a toy with 15 disks, it would take me over nine hours to solve the thing.
- And it will take the monks approximately 585 billion years to end the world with their 64 golden disks. So I guess we're safe.
- Next Monday, we'll go over a better way of analyzing algorithms than counting the exact number of steps to perform them like we did here. The following Wednesday, we'll begin discussing a special kind of recursive strategy called "divide-and-conquer" and see more examples of how induction and recursion are closely related.