Chapter 4

Induction and Recursion

"To understand recursion, one must first understand recursion."
– Stephen Hawking

You have already read about and worked on one topic that involves "recursive structure," those recursively-defined sequences. It turns out that lots of other math- and computer-science-related objects feature recursive structure, too, and there is a certain family of proof-tecniques that are especially well suited for Propositions about them.

- Proof By Induction
  - Works very well for proving infinitely many propositions that each depend on natural numbers $n \in \mathbb{N}$
  - Key step involves relating the "next" proposition $P(k+1)$ to the "previous" one $P(k)$
  - Key examples: formulas for sums, inequalities, and recursive sequences (that have one initial condition)
- Proof by "Strong" Induction
  - Works very well for proving infinitely many propositions that each depend on natural numbers $n \in \mathbb{N}$
  - Key step involves relating the "next" proposition $P(k+1)$ to multiple previous ones, $P(k), P(k-1), \cdots, P(0)$
  - Key examples: formulas for sums, inequalities, and recursive sequences (that have multiple initial conditions)
- Proof by Structural Induction
  - Works very well for proving propositions about objects that enjoy general recursive structure
  - Key step involves relating a proposition about a general object to one about "smaller" or "simpler" objects
  - Key examples: facts and formulas about trees and subsets of $\mathbb{N} \times \mathbb{N}$

Even though we have different names for these three new proof techniques, they all work according to the same generic strategy, namely

$$\boxed{\text{True for “smaller” or “previous” cases } \Rightarrow \text{ True for “bigger” or “future” cases.}}$$

Many Mathematicians and Computer Scientists will simply use the term “Induction” to refer to any of these three approaches, but Computer Scientists are more likely to emphasize the third one – “Structural Induction” – as something special.

“Strong Induction” is more general than what we have labelled “Induction” (and what some other folks call “Weak Induction”), and “Structural Induction” is more general still. Before we launch into a detailed discussion of these proof techniques let’s first collect one or two more exaples of objects that exhibit “recursive structure” – OK, OK. *Three*, let’s discuss *three examples.*

1. Examples of Recursive Structure

Recursively defined sequences (like the Fibonacci Numbers) is one example of an object that features “recursive structure.” The following example presents a relatively important or famous instance of recursion, one that seemingly has nothing to do with sequences; this first example involves **iterating a function**.

Example 1.1. *The Famous Collatz Conjecture involves a function* $f : \mathbb{N} \to \mathbb{N}$ *given by the rule*

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases}$$

Mathematicians and Computer Scientists are interested in the behavior of this function when its outputs are “fed back in as inputs.” (This process is called ***iterating*** *or* ***iteration****.) For instance,* $f(3) = 10$ *and we can plug* 10 *back in to get* $f(10) = 5$*. Repeating this, we find* $f(5) = 16, f(16) = 8, f(8) = 4, f(4) = 2$ *and, finally,* $f(2) = 1$*. When we plug* 1 *back in we get* $f(1) = 4, f(4) = 2, f(2) = 1\ldots$ *creating an endless cycle of* 1*’s,* 4*’s and* 2*’s. We can notate this in a simplistic way as follows:*

$$3 \mapsto 10 \mapsto 5 \mapsto 16 \mapsto 8 \mapsto 4 \mapsto 2 \mapsto 1 \mapsto 4 \mapsto \boxed{2 \mapsto 1 \mapsto 4} \mapsto 2 \mapsto 1 \cdots$$

The boxed-portion repeats over-and over, and the actual Collatz Conjecture claims to answer this question: does this repeating-cycle behavior always happen, no matter which input we start with? We can try another example, and you should try your own, too!

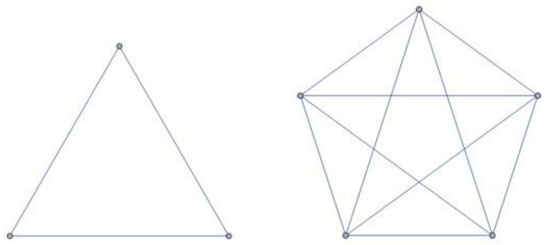
$$52 \mapsto 26 \mapsto 13 \mapsto 40 \mapsto 20 \mapsto 10 \mapsto 5 \mapsto 16 \mapsto 8 \mapsto \boxed{4 \mapsto 2 \mapsto 1}$$

Applying the function f *over-and-over again is where we find some “recursive structure;” in general,* ***iterating a function repeatedly*** *results in recursive structure. Future values of the process depend on the previous values in an explicit way.*

Fascinatingly enough, the Collatz Conjecture is still a conjecture – no one has a proof that its true and no one has a proof that it’s false. At least not yet. But you can bet that the "recursive structure" inherent in the problem will be used whenever a proof *is* discovered.

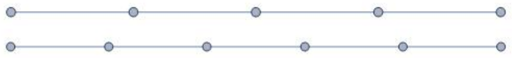
Also, did you happen to catch the secret sequence (or sequence*s*) that are lurking behind our Collatz example? If not, take a moment to think about the example (or re-read it – we'll wait). Did you spot the sequence yet? It's a recursively-defined sequence, and the initial condition is whatever number you first start with, call it a_0. All other terms in this sequence are then given by the recurrence equation $a_n = f(a_{n-1})$.

Example 1.2. *A* ***graph*** *is a set* G *consisting of* ***nodes*** *(also called* ***vertices****) and* ***edges****. Graphs are usually presented visually, with nodes displayed as dots and edges drawn as lines or curves joining them.*



The graph on the left has three vertices and three edges, while the graph on the right has five nodes and 20 edges. We will return to general graphs soon enough, but for this example we want to focus on a special type of graph, ones called ***Path Graphs*** *(or sometimes* ***Linear Graphs****).*

A ***path graph*** *is a graph whose vertices can be arranged as an ordered list,* $v_1, v_2, v_3, \cdots$*, and whose edges* only *join consecutive nodes. We will let* $PG(n)$ *denote the* ***Path Graph*** *with* n *vertices; examples of* $PG(5)$ *and* $PG(6)$ *are shown below.*



Here is a proposition about Path Graphs:

Proposition. $\forall\, n \in \mathbb{N},\;\; PG(n)$ *has* $n-1$ *edges.*

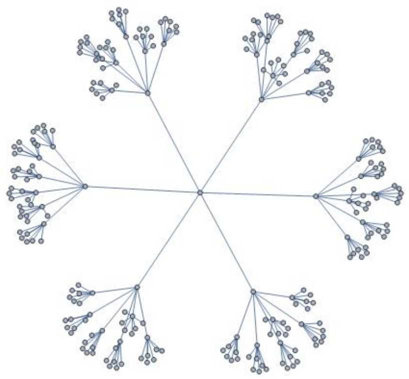
One might be able to prove this proposition directly, by contrapositive or perhaps by contradiction. However, it is natural to detect some "recursive structure" regarding Path-Graphs. In particular

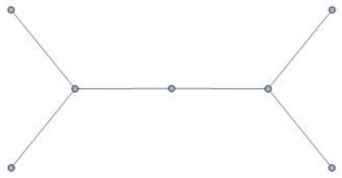
$$\boxed{PG(n) = PG(n-1) \text{ plus one extra edge \& one extra vertex.}}$$

Pause to make sure you understand and can visualize the meaning of the boxed sentence above; the fact that we can relate path-graphs with n *vertices to path-graphs with fewer vertices* ***is*** *the source of the "recusrive structure" here, and this simple observation can be used to help us prove Propositions (like the one above) in different ways!*

Example 1.3. *There is another type of graph that both Mathematicians and Computer Scientists* ***love to talk and think about****, and it has a nice name, too:* ***a tree****. It turns out that there are lots and lots of ways to define a tree, but here is perhaps the most accessible way.*

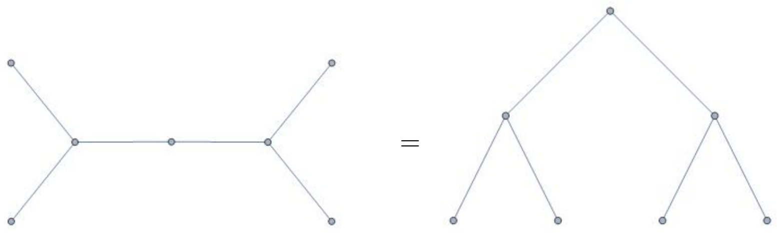
A ***tree*** *is a graph in which every vertex can be joined to any other vertex using exactly one sequence of edges. Here are some visual examples*





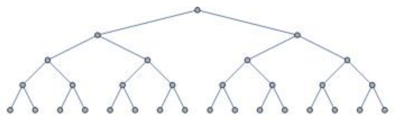
While the tree on the left is more complicated than the one on the right, note that both satisfy the defining-tree-property: there is only one way to "get" from one vertex to any other.

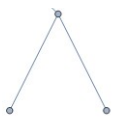
The tree up above on the right is an example of a ***binary tree****, one where each node has either 0, 1, or 2 edges joining it to other nodes. In fact, we can rearrange the placement of those nodes and edges for this graph so that it looks more "tree like:"*



In fact, the graph above is an example of a ***full binary tree****; this is a tree in which every node has exactly 0 or 2 edges joining it to other nodes. When we draw our trees so that there is one node or vertex "at the top," we call this vertex* ***the root****, and we refer to* ***parent nodes*** *and* ***children nodes*** *as we "move down" the graph.*

Here are two more examples of ***full binary trees****:*





Note the "recursive structure" on display here; the full binary tree on the left is made up of simpler or smaller full binary trees, like the one displayed on the right.