Induction

1 Introduction

See also the video lecture recording: Induction on Canvas.

Induction is a powerful proof technique that is widely used in computer science and mathematics. It has many variations, and we shall look at some of them.

- Ordinary induction over $\mathbb{N}$.
- Course-of-values induction over $\mathbb{N}$.

2 Induction over $\mathbb{N}$

Imagine an infinite sequence of dominoes standing on an infinite table, with Domino $n+1$ standing just behind Domino n, and someone pushes Domino 0. Then Domino 0 falls, causing Domino 1 to fall, causing Domino 2 to fall, causing Domino 3 to fall. . . What about Domino $10^{10^{100}}$? It will eventually fall. Indeed it is obvious that *each domino will fall*.

This is the idea behind induction over $\mathbb{N}$. Let P be a property of natural numbers. Suppose that $P(0)$—this is called the *base case*. Suppose also that, for any natural number $\mathbb{N}$, the statement $P(n)$ implies $P(n+1)$—this is called the *inductive step*, and the hypothesis $P(n)$ is called the *inductive hypothesis*. From these two facts, we may conclude that *every* natural number satisfies P, even big ones like $10^{10^{100}}$.

Example. Let's prove $0 + \ldots + (n-1) = \frac{1}{2}(n-1)n$ by induction on $n \in \mathbb{N}$. Clearly this is true for $n = 0$, since the sum of no numbers is defined to be 0. Assuming it's true for n, let's show that it's true for $n + 1$.

$$\begin{aligned}
0 + \cdots + ((n+1) - 1) &= 0 + \cdots + (n-1) + n \\
&= \frac{1}{2}(n-1)n + n \quad \text{(by the inductive hypothesis)} \\
&= \frac{1}{2}n^2 - \frac{1}{2}n + n \\
&= \frac{1}{2}n^2 + \frac{1}{2}n \\
&= \frac{1}{2}n(n+1) \\
&= \frac{1}{2}((n+1) - 1)(n+1) \quad \text{as required.}
\end{aligned}$$

Note: induction is not the only way to prove this fact. You might be able to see another.

3 Variations

Here are some variations. Let P be a property of natural numbers.

- Suppose we have proved that P holds for 0, 1 and 2, and also that, if it holds for n, $n+1$ and $n+2$, then it also holds for $n+3$. We now know that P holds for all natural numbers.
- Suppose we have proved that P holds for 1 and 3, and also that, if it holds for n and $n+2$, then it also holds for $n+4$. We now know that P holds for all odd natural numbers.
- Suppose we have proved that P holds for 1, and also that, if it holds for n, then it also holds for $2n$. We now know that P holds for every power of 2.

4 Course-of-values induction

When we give a proof by ordinary induction, the inductive step proves that $P(1)$ follows from $P(0)$, that $P(2)$ follows from $P(1)$, that $P(3)$ follows from $P(2)$, and so on. But surely, when proving $P(3)$, it should be acceptable to assume not just $P(2)$ but also $P(1)$ and $P(0)$. This thinking leads to *course-of-values induction* (also called "strong induction").

The principle is as follows. Let P be a property of natural numbers. Suppose that, for any natural number n, the statement $P(n)$ holds if P holds for all natural numbers less than n. (The latter assumption is called the *inductive hypothesis*.) This means that

- $P(0)$
- if $P(0)$, then $P(1)$
- if $P(0)$ and $P(1)$, then $P(2)$
- if $P(0)$ and $P(1)$ and $P(2)$, then $P(3)$
- etc.

From this fact, we may conclude that *every* natural number satisfies P, even big ones like $10^{10^{100}}$.

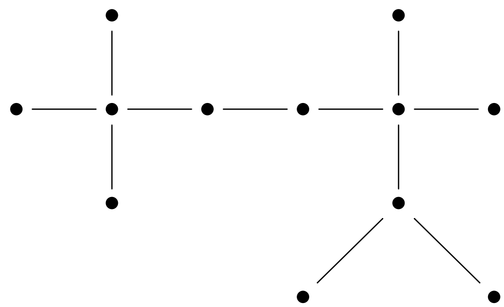
Example. The *merge sort* algorithm is the following recursively defined algorithm for sorting a list p.[1]

- If the length of p is 0 or 1, return p.
- If the length of p is $2k$ where $k > 0$, then sort the left part of length k, and sort the right part of length k, and merge the results.
- If the length of p is $2k+1$ where $k > 0$, then sort the left part of length k, and sort the right part of length $k+1$, and merge the results.

How do we know that this algorithm terminates, and returns a list that is a sorted version of p? By course-of-values induction on the length of the list. In each of the three cases, it is easy to see that the algorithm yields a sorted version of p, assuming that it works correctly on shorter lists.

[1] The version given here is intended to return the sorted version of the list. The version for sorting an *array* with in-place update is slightly different, but the idea is the same.

Example. A *tree* is an undirected graph that is connected and acyclic. For example:



The empty undirected graph is not considered to be connected, so a tree has at least one vertex.

Let's show that every tree T with t vertices has $t-1$ edges. We proceed by course-of-values induction on t.

- If $t = 1$, then there are no edges, so we're done.
- If $t > 1$, then there's at least one edge (because there are two distinct vertices, connected by a path). Pick an edge e, whose endpoints are x and y. When we remove e from T, what remains is a graph T' in which each vertex z is connected to x or to y but not both. (Proof: in T, there's a unique path from z to x. If it doesn't end in e, then it lies in T', and there's no path in T' from z to y. If it does end in e, then there's a path in T' from z to y, but not from z to x.)

  Let R be the part of T' whose vertices are connected to x, and S be the part whose vertices are connected to y. These are both trees. Let r be the number of vertices in R, and s the number of vertices in S. Since $r < t$ and $s < t$, the inductive hypothesis tells us that R has $r-1$ edges and S has $s-1$ edges. So the number of edges in T is $(r-1)+(s-1)+1 = r+s-1 = t-1$, as required.

Note: induction is not the only way to prove this fact. You might be able to see another. As a hint, pick one vertex and call it the "root".

Example Recall the first step of the proof of Kleene's theorem: showing that every regexp E has a corresponding εNFA. This can be seen as a course-of-values induction on the *length* of E. In other words, we prove that the statement is true for E assuming that it is true for all *shorter* expressions.

In fact, all we need to assume is that the property holds for *subexpressions*. For example, to prove the property for E_0E_1, we need only assume that it's true for the subexpressions E_0 and E_1. This kind of argument often appears in computer science, and is called *structural* induction.