COMPSCI 230: Discrete Mathematics for Computer Science

Recitation 6: More Induction

Spring 2020 *Created By: David Fischer*

1. We begin with some example problems for structural induction - note that there are multiple ways to formulate these proofs, some of which are more clearly structural than the solutions provided.

   (a) An *n-ary* tree is a rooted tree where each node can have at most n children. Note that $n \in \mathbb{N}$. We will follow a process similar to that performed in class to show that the maximum number of nodes in an *n-ary* tree of depth d is n^d. By convention, the depth 0 tree consists of just the single root node. - note that this was proved in class for a binary tree, so, the students should be able to do this on their own. —– (15 minutes - 5 for attendance and preamble, 8 for students to do on their own, and 2 to go over the solution).

   Theorem 1. *Any* n-ary *with $n > 1$ tree of depth d has at most $\frac{n^{d+1}-1}{n-1}$ nodes.*

   Proof. For the purposes of the proof, assume $n > 1$. We begin with an obviously true fact.

   Claim 1. *A complete* n-ary *tree (for some fixed n) of depth d will always have at least as many nodes as any other* n-ary *tree of depth d.*

   Lemma 1. *The number of leaves in a complete* n-ary *tree of depth d is n^d.*

   Proof. By induction on depth. For $d = 0$, we simply have the single root node, and so $n^0 = 1$ is the number of leaves. Suppose for some depth k, the number of leaves in the complete *n-ary* tree of depth k is n^k. Then, for depth $k+1$, each previous leaf must have n nodes added to it. Each new node is a leaf since it has no children, and each old node is no longer a leaf since it now has children. This gives $n * n^k = n^{k+1}$, and so for depth $k+1$ there are n^{k+1} leaves. So, by induction lemma 1 is proved for all depths $d \geq 0$. □

   We finish by noting that the number of nodes in a complete *n-ary* tree of depth d is equal to the sum of the number of leaves over all depths $i \leq d$. By lemma 1 this gives $\sum_{i=0}^{d} n^d = \frac{1-n^{d+1}}{1-n} = \frac{n^{d+1}-1}{n-1}$ by multiplication by $\frac{-1}{-1}$. So, by claim 1 and this calculation, theorem 1 is shown. □

   (b) We now consider merge sort. We will prove the correctness of merge sort via structural induction. Begin by reviewing what merge sort is, and how it works.

   Theorem 2. *Merge sort is correct.*

   Proof. Students should say what this actually means - what would it mean for merge sort to be correct? (ans: no inversions in list after it runs) In the process of proving this, it will be useful to have the following fact.

   Lemma 2. *The merge operation called on two sorted arrays l_1 and l_2 of size m and n respectively results in a sorted array l_s of size $m + n$.*

Proof. We will show that for any two numbers in our lists, their relative order is preserved after the merge operation. Consider any pair of numbers a and b, in either of the two lists. We have four cases on the location of a and b. Suppose without loss of generality $a > b$. If a and b are both in l_1, then it is impossible for merge sort to place a before b in l_s, as the pointer looking at l_1 does not move to the next element in the list before placing the current element. The exact same logic holds if a and b are in l_2 (and so we could in fact have supposed without loss of generality that a and b were in l_1). If $a \in l_1$ and $b \in l_2$, then merge sort must place b before placing a. If this needs more convincing, suppose not. Then at some point the algorithm compares a and b since at every step of the merge operation it compares an element in one list to an element of the other list, places the smaller element and moves the pointer that pointed at the smaller element to the next element in its list. Then, since a was placed before b it must be that $a < b$, which is a contradiction. The case where $b \in l_1$ and $b \in l_2$ proceeds exactly as the case just stated. The case where $b > a$ is exactly the same as the previous four cases, with a and b swapped. So, in all cases, we have shown the order of two elements in the lists are preserved after performing the merge operation and so we are done. $\square$

We now proceed by induction on the size n of the unsorted input array. For a base case, suppose $n = 1$. Clearly the list is sorted. Now, supposed by strong induction that merge sort works on all inputs of size strictly smaller than k. Then note that the merge sort algorithm will call itself on two subarrays of size $\frac{k}{2}$. By the inductive hypothesis, merge sort will correctly sort these two subarrays, and by lemma 2, the merge operation will preserve the ordering of all elements in either list. So, by induction we have shown that merge sort is correct for all lists of size greater than or equal to one. $\square$

(c) Graph Coloring

Theorem 3. *Any graph with a max degree of k can be colored using at most $k + 1$ colors. Hint: use induction on the max degree.*

Proof. We proceed by induction on the number of nodes in a graph. Clearly for a graph with one node, our theorem holds, as we can color it using one color, which is one more than the max degree in the graph. Now, assume that for any graph with n nodes we can color it using $n + 1$ colors. Continue to let k be the max degree of the graph, and let $G = (V, E)$ with $|V| = n + 1$. Arbitrarily order the vertices $v_1, v_2, ..., v_n, v_{n+1}$. Then, note that the graph excluding v_{n+1} is an n node graph, with max degree k, and so can be colored using $k + 1$ colors. Now we just need to show that node v_{n+1} can be colored without using an additional color. To finish, note that the degree of node v_{n+1} must be less than or equal to d, so there must be some color remaining with which we can color node v_{n+1}. So, we have colored the graph with $n + 1$ nodes using at most $d + 1$ colors, where d is the max degree of any node in the graph and so our theorem is proved. $\square$

2. We end with an example of strong induction.

   (a) Binary representations

   Theorem 4. *Any integer can be written as a binary number. Hint: show that any integer can be written as a sum of (distinct) powers of two.*

   Proof. This is the same as saying, any number can be written as a sum of powers of 2. We will prove this using induction. Clearly $1_{10} = 1_2$ so $P(1)$ holds. Now, suppose by strong induction we have $P(k')$ for all integers $k' < k$. Let s be the largest power of two such that $s < k$. Clearly, s can be written as a sum of powers of two with one term, as it is itself a power of two. To finish the inductive step note that $k - s < k$, and so $k - s$ can be written as a sum of powers of two by the inductive hypothesis. Then, note that $k - s + s = k$, and so any positive integer k can be written as a sum of powers of two. By extension, negative numbers can as well by simply placing a negative sign in front of any positive binary representation (this is not a refresher on two's complement!). It just remains to show that the resultant set of powers of two that sum to k are distinct. For this, it suffices to show that $k - s < s$. Suppose not. Then, $k - s \geq s$. This means that $k = k - s + s \geq s + s = 2s$. This contradicts the definition of s as the largest power of two less than k, so we must have $k - s < s$, and we have found powers of two which add to k and have distinct elements. Since any integer can be written as a distinct sum of powers of two, we can write a binary number representing it, as shown in the following construction. Let t be an integer, and let S be the unordered set of powers of two which sum to t. As above, let x be the largest power of two smaller than t. Also, let P be the ordered set of all powers of two in increasing order. We construct b, the binary representation of t, as follows. For each $r \in P$, starting at the smallest and working up, concatenate a 1 to the left side of b if $r \in S$, and a 0 otherwise. By induction, we have shown that any number can be written as a sum of powers of two, and given such a set of powers of two, by construction we have show that any integer can be written in binary. $\square$