Recitation 10. Induction

Introduction. Induction is useful in proving properties of recursive algorithms, like their execution times. It is also the basis for understanding loops in terms of loop invariants and bound functions and understanding recursive methods in terms of what a recursive call does based the specification of the method called instead of how the call is executed. We present the fundamentals of mathematical induction here. Study Weiss's chapter on induction as well!

Terminology. A natural number is a nonnegative integer --a member of {0, 1, 2, …}. A positive integer is an integer that is greater than 0.

For a formula e = f, the LHS is the lefthand side, e, and the RHS is the righthand side, f.

First formulation of proof by induction. To prove a property P(n) for all natural numbers, do this:

(1) Prove the base case: P(0)

(2) Prove the inductive case. For arbitrary positive integer k: assume that P(0), P(1), …, P(k-1) are true and prove P(k).

Alternative formulation. To prove a property P(n) holds for all natural numbers, do the following:

(1) Prove the base case: P(0)

(2) Prove the inductive case. For arbitrary natural number k: assume that P(0), P(1), …, P(k) all are true and prove P(k+1).

Note on the first formulation. Sometimes, it helps to have several bases cases, e.g. prove P(0), P(1), and P(2) separately; then, for arbitrary k>2, assume P(0), P(1), …, P(k-1) and prove P(k).

Note. If we want to prove something only about integers {3, 4, …}, then we (1) use the base case P(3) and (2) prove the inductive case: for arbitrary k>3, assume P(0), …, P(k-1) and prove P(k).

Note: We have defined what is called "strong induction". In "weak induction", one assumes only P(k-1) and proves P(k) in the inductive case. However, the two are equivalent, so there is no sense in making a big deal about the difference between them.

Theorem. For all n>= 0, P(n) holds, where,

$$P(n): \quad \sum_{1<=i<=n} (2i-1) = n^2$$

Proof.

Base case. For n=0, the LHS is 0 and the RHS is 0.

Inductive case: For k>=0, we assume P(0), …, P(k) and prove P(k+1): We start with the LHS of P(k+1) and prove it equal to $(k+1)^2$:

$$\begin{aligned}
& \sum_{1<=i<=k+1} (2i-1) \\
= & \quad \text{<break off the term for i=k+1>} \\
& \sum_{1<=i<=k} (2i-1) + 2(k+1)-1 \\
= & \quad \text{<inductive hypothesis P(k)>} \\
& k^2 + 2(k+1)-1 \\
= & \quad \text{<arithmetic P(k)>} \\
& k^2 + 2k+1 \\
= & \quad \text{<arithmetic P(k)>} \\
& (k+1)^2
\end{aligned}$$

Note on the format for doing calculations. Between each pair of successive formulas, we write "='' followed by an indented hint; the hint says what we used in showing that the first formula equals the second. Always put such hints in, because it will help you later in reading your own proof and because it helps anyone else who reads your proof --e.g. a grader.

Why a proof by induction works. Students often ask how a proof by induction shows that P(n) holds for all natural numbers n. We show why.

Suppose we have proved:

(1) The base case: P(0)

(2) The inductive case. For arbitrary positive integer k: assume P(0), P(1), …, P(k-1) and prove P(k).

Now, give us any integer, like 99, and we can prove (if we have the time) that P(99) is true. Here's how.

Step 0. We know that P(0) holds

Step 1. P(0) holds. Therefore, we can use the inductive case to prove that P(1) holds.

Step 2. P(0) and P(1) hold. Therefore, we can use the inductive case to prove that P(2) holds.

Step 3. P(0), P(1), and P(2) hold. Therefore, we can use the inductive case to prove that P(3) holds.

…

Step 99. P(0), P(1), P(2, …, P(98) hold. Therefore, we can use the inductive case to prove that P(99) holds.

A calculational format is not needed. The proof on page 1 calculated something in the inductive case. This example shows another style.

Suppose we have a currency that consists of 2-cent and 5-cent coins. Prove that any amount above 3 cents can be made using these coins.

We write P(n) as

P(n): Some bag of 2-cent and 5-cent coins has sum n.

We prove that P(n) holds for all n>=4.

Base case n= 4. A bag that contains 2 2-cent coins and 0 5-cent coins sums to 4.

Inductive case. We prove P(k+1), for k>=4, assuming P(k). Since P(k) holds, there is a bag of 2-cent and 5-cent coins that sums to k. Consider two cases: the bag contains a 5-cent coin or it does not.

Case 1: the bag contains a 5-cent coin. Take the 5-cent coin out and put in 3 2-cent coins. The bag now sums to k+1. Case proved.

Case 2: the bag doesn't contain a 5-cent coin. The bag contains only 2-cent coins. Since k>=4, the bag contains at least 2 2-cent coins. Take 2 out and throw in a 5-cent coin. The bag now sums to k+1. Case proved.

Two important hints on proving by induction.

1. State the theorem. NEVER start proving something by induction without first writing down what P(n) is and stating the theorem in the form

for all n, n>=0 , P(n) holds.

If you don't say what P(n) is, how can you prove that it holds? If you don't state the range of n beforehand, (e.g. n>=0), how can anyone know what you are proving. Don't EVER in this course forget this hint.

2. Exposing the inductive hypothesis. In proving the inductive case, you have to use at least one of P(0), P(1), …, P(k) in proving P(k+1). Therefore, when you start with (part of) P(k+1), your goal should be to manipulate it to expose one of the formulas P(1), …, P(k) --i.e. to change it so that you see part of P(k) so you can replace it. We did this in the first problem. We changed the sum over i in the range 0..k+1 to a sum over i in the range 0..k, because that is what the LHS of P(k) contains.

Make your development of the proof of the inductive case goal-oriented; strive to expose one of P(0), …, P(k) so that you can use it.

Your understanding of recursive methods depends, actually, on mathematical induction. To see this, let's take the definition of n!, for n>=0, as

$$n! = \prod_{1 \le i \le n} i \quad \text{(which is } 1*2*\ldots*n\text{)}$$

Note: $\prod_{1 \le i \le 0} i = 1$ (by definition)

Here's method fact:

```
// = n! (for n>=0)
public int fact(int n) {
    if (n=0)
        {  return 1; }
    return fact(n-1)*n;
}
```

We prove that, for all n>=0,

P(n): fact(n) = n!

Base case: For n=0, fact(0) = 1, which we see by inspection of the method body. But 1 = 0!, so the base case holds.

Inductive case: For k>0, we assume P(k-1) and prove P(k):

```
    fact(k)
=     <inspect method body --this exposes P(k-1)>
    fact(k-1)*k
=     <Use assumption P(k-1)>
    (k-1)!*k
=     <arithmetic, definition of n!>
    k!
```

Throughout the course, we have told you to understand a recursive method in terms of the recursive calls doing what the specification of the method says they will do. And that's what we did in this more formal proof,

Proving things about "inductive definitions". An inductive or recursive definition is a definition of something in terms of itself. For example, we can define the notation b^n, for n>=0, inductively as follows:

$b^0 = 1$

$b^n = b*b^{n-1}$ for n>0

We can prove facts about such inductive definitions using mathematical induction. When we do this, we generally do a case analysis using the cases given by the inductive definition.

For example, you should prove that, for n>=0 and m>=0,

$b^{m+n} = b^m * b^n$

Caution. You can't do this properly unless you first put the theorem in the form "for all k, k>=0, P(k)", and state precisely what P(k) is.

Proofs about binary trees. We have defined binary trees inductively, as follows:

(1) null is a binary tree, called the empty binary tree

(2) (left, v, right) is a binary tree, where left and right are binary trees and v is any value, called the root value of the binary tree.

We also call (left, r, right) a node.

We deal only with finite binary trees, which means that the number of nodes in it is finite.

Because we have defined binary trees inductively, we can define various properties of a tree inductively:

The number of nodes in tree t, written #t, is defined by:

#null = 0

#(l, v, r) = 1 + #l + #r

The height of a binary tree, height(t), is defined by

height(null) = 0

height (l, v, r) = 1 + max(height(l), height (r))

The level of a node n in a binary tree t is defined by:

level(n,t) = 0 if n is the root of t

= 1 + level(n,t.left) if n is in subtree t.left

= 1 + level(n,t.right) if n is in subtree t.right

We can also define:

•A full binary tree is a binary tree in which each node has 0 or 2 children.

•A complete binary tree is a binary tree in which all leaf nodes are at level n or n-1 (for some n) and all leaves at level n are toward the left.

• A perfect binary tree is a complete binary tree in which all leaves are at the same level.

• A binary tree is linear if each node has at most 1 child.

The exercises show you a number of properties of binary trees that can be proved inductively.

Exercises.

1. Prove by induction that, for n>= 0,

$$\sum_{0<=i<n} 2^i = 2^n - 1$$

2. Prove by induction that, for n>= 0,

$$\sum_{0<=i<n} 3^i = (3^n-1)/2$$

3. Prove by induction that, for n>=0,

$$\sum_{1<=i<=n} i = n*(n+1)/2$$

4. Prove by induction that, for n>=3, $2*n+1 < 2^n$.

5. Prove by induction that, for n>=0, $4^n - 1$ is divisible by 3. Hint. When trying to prove something about $4^{n+1} - 1$, you have to use the fact that it holds for $4^n - 1$. So, start with $4^{n+1} - 1$ and expose the formula $4^n - 1$ by adding it to and subtracting it from $4^{n+1} - 1$.

6. Prove by induction that, for n>=0, $10^n - 1$ is divisible by 9. Hint. See hint on previous question.

7. Prove by induction that, for n>=0 and x != y, $x^n - y^n$ is divisible by x-y. Hint:In starting with "$x^{n+1} - y^{n+1}$ is divisible by x-y", you have to expose the inductive hypothesis "$x^n - y^n$ is divisible by x-y", To be able to expose it, add and subtract the formula xy^n from the formula $x^{n+1} - y^{n+1}$.

8. Prove by induction that any amount greater than 14 can be obtained using 3-cent and 8-cent coins.

9. A convex polygon is a polygon in which the line joining any two points on its perimeter is within the polygon. Prove by induction that, for n>=3, the sum of the angles of a convex polygon with n sides is (n-1)*180. Use the fact that the sum of the angles of a triangle is 180.

The next few questions deal with Fibonacci numbers, which are defined by:

$F_0 = 0$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$, for n>=2

Also, phi = (1+sqrt(5))/2, and it is known that $phi^2 = phi + 1$.

10. Prove that $F_n < 2n$, for n<=0.

11. Prove that, for n>=1, $phi^{n-2} <= F_n <= phi^{n-1}$

12. Prove that, for n>=0 and m>=1,
$F_{n+m} = F_m F_{n+1} + F_{m-1}F_n$

13. Prove that, for n>=0, F_{3n} is even, F_{3n+1} is odd, and F_{3n+2} is odd.

14. Prove that, for n>= 0, $F_1 + F_2 + \ldots + F_n = F_{n+2} - 1$.

15. The definition of F_n given earlier can be transformed mechanically into a Java method Fib for calculating F_n. Prove by induction that the total number of calls on Fib made to calculate F_n, for n>= 3, is $F_{n+2} + F_{n-1} - 1$. This is a huge number --to calculate F(30) requires over 2 million calls, and calculating F(100) takes over 21^{10} calls! It's an inefficient way to calculate F(n).

16. Below are two definitions of the reverse of a String; in it, s is supposed to be a String and c a character. Operation + is catenation.

revf("") = ""
revf(c + s) = revf(s) + c

refb("") = ""
revb(s+c) = c + revb(s)

Prove that, for all Strings s, revf(s) = revb(s)

17. Below is a definition of the reverse of a String. Prove that rev(s) = rev1(s), for all Strings s, where c1 and c2 are arbitrary characters and where rev1 is given in the previous exercise. You can make use of the result of the previous exercise.

rev("") = ""
rev(c1) = c1
rev(c1 + s + c2) = c2 + rev(s) + c1

18. Define m_0 inductively as follows:

$m_0 = 0$

$m_{0+1} = 2m_0 + 1$, for n>=0

Prove by induction that, for n>=0, $m_0 = 2^n - 1$.

Proofs about binary trees.

19. Prove by induction that the number of nodes in a perfect binary tree of height h is 2h+1-1.

20. Prove that the number of leaves in a perfect binary tree of height h is 2^{h-1}.

21. Prove by induction that the number of nodes in a linear binary tree of height h is h.

22. Prove by induction that the number of leaves in a linear binary tree of height h is 1.

23. Prove that every nonempty complete binary tree has an odd number of nodes.

Proofs about sets.

24. Let P(s) be the power set of set s --the set of all subsets of s. Define #s to be the size of a set. Prove by induction that $\#P(s) = 2^{\#s}$. Hint. The base case is easy. Consider a set {e} u s, where element e is not in s. Then P({e} u s) consists of sets that contain e and sets that do not contain e. How many are there of each?

25. Jack claims that he is exactly one-third Spanish. (For example, a person is 1/4 Spanish if 1 grand-parent was Spanish and 3 were not). Prove that Jack is lying by relating the problem to the following set and showing that 1/3 is not in the set.

0 is in S
1 is in S
if s and y are in S, then so is (x+y)/2.