CIS 121—Data Structures and Algorithms with Java—Spring 2017

Review—Tuesday (9/6)

Learning Goals

During this lab, you will:

- Review common proof techniques
- Build intuition for the connection between induction and recursion
- Walk through solving and testing a recursive problem
- Review some graph theory

Proof Techniques

- **Fundamental proof techniques.** For contradiction and contrapositive, try writing out the truth tables to prove to yourself that these statements are logically equivalent to a direct proof.
  - Direct: $p \implies q$
  - Contrapositive: $\neg q \implies \neg p$
  - Contradiction: $p \wedge \neg q \implies C$
  - Induction: To prove $P(n)\ \forall n \geq b$, show $P(b)$ and $\forall n \geq b$, $P(n) \implies P(n+1)$
- **A basic contradiction proof.** Prove that $\forall$ integers, if n^2 is odd then n is odd. Suppose for contradiction that n^2 is odd but n is even. We can express n as $n = 2k$, where $k \in \mathbb{Z}$.

$$\begin{aligned} n &= 2k \\ n^2 &= (2k)^2 \\ &= 4k^2 \\ &= 2(2k^2) \end{aligned}$$

  Since $2k^2 \in \mathbb{Z}$ (why?), it must be that n^2 is even. But we assumed that n^2 is odd! Contradiction. Now try to prove this using the contrapositive. It should be almost the same proof, just worded differently.

- **A basic combinatorial proof.** Pascal's formula:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{1}$$

  Solution: We show how to choose k people from a group of n people in two ways. LHS: n choose k. RHS: designate one person to be special. We break into two cases: 1. The special person is included in the group, so we choose $k-1$ more out of $n-1$. 2. The special person is not included in the group, so we choose k people out of $n-1$. Now try to prove this identity algebraically!

- **A linearity of expectation problem.** What is the expected number of fixed points in a random permutation of $(1, 2, 3, ..., n)$?

  Remember that linearity of expectation simply means that the expectation of a sum is equal to the sum of the expectations. That is: $E[X + Y] = E[X] + E[Y]$. It's like you can "distribute" the expectation. We often use L.O.E. with indicator random variables. Now, to solve the problem.

Let X be a random variable representing the number of fixed points in a permutation. Let X_i be an indicator random variable that is 1 if spot i is fixed and 0 otherwise. We know that $E[X_i] = P(X_i = 1) = \frac{1}{n}$ (why?). Now we apply L.O.E.:

$$\begin{aligned} E[X] &= E[\sum_{i=0}^{n} X_i] \\ &= \sum_{i=0}^{n} E[X_i] \\ &= E[X_1] + E[X_2] + ... + E[X_n] \\ &= n * E[X_i] \\ &= n * \frac{1}{n} = 1 \end{aligned}$$

So the expected number of fixed spots is independent of the length of the sequence being permuted!

Induction and Recursion

- First an induction proof: Prove that a balanced binary tree of height $lg(n)$ has at most n leaves. We will solve this by doing induction on the height of the balanced binary tree.

  Base Case: n = 0

  In this situation we have a tree of height 0. Clearly the tree has at most 1 leaf.

  Strong Induction Hypothesis: For all $1 \leq j \leq k$ A balanced binary tree of height $lg(j)$ has at most $2^{lg(j)}$ leaves.

  Induction Step: We want to show that a balanced binary tree of height $lg(k) + 1$ has at most $2^{lg(k)+1} = 2k$ leaves. Let us consider the root of the binary tree. We can use the induction hypothesis on the left half because it has a height that is lg(k), and the induction hypothesis on the right half because if has a height that is lg(k) as well. Therefore, we know that the left half as at most k leaves and the right half also has at most k leaves. Therefore, combining them together means that the entire tree has at most 2k leaves.

- Now, let us write pseudocode that would count the leaves of a balanced binary tree.

```
function leaves(node){
if right == null and left == null then
|  return 1;
end
if right != null then
|  total += leaves(right);
end
if left != null then
|  total += leaves(left);
end
return total
}
```

Notice the incredibly similarity between induction and recursion!

Solving and testing a recursive problem

Let us use induction to prove that the binary search algorithm will run in $O(lgn)$ time on a sorted array. First, to do this we must define a recurrence relation, which is the following:

$$T(n) = \begin{cases} T(n/2) + c, & \text{if } n \geq 1 \\ O(1), & \text{otherwise} \end{cases}$$

Now we must prove that $T(n) = O(lgn)$

Base Case: n = 1 If you have an array of size 1, then you can, in constant time, determine whether or not the number is the number that you are looking for. Therefore, trivially the algorithm runs in $O(lgn)$ time.

Strong Induction Hypothesis: For all $1 \leq j \leq k$ Binary search will determine whether or not the number you are looking for is in the list in $O(lg(j))$ time, where j is the size of the list.

Induction Step: Given that you have a list of size k + 1, we want to show that the algorithm runs in $O(lg(k+1))$ time. Thus we want to show that $T(n) \leq c * lg(k+1)$ for some c, and for all $n > n_0$.

Note that with how binary search works, we check the middle element of the subarray against the element we are searching for. If it is greater than the element we recurse on the right half, otherwise we recurse on the left half. Therefore, this is 1 operation to check. Also there is 1 operation to call the function again on the respective half. Using strong induction, we thus know that the runtime of binary search on the half is bounded by $c * lg(\frac{k+1}{2})$.

$$c * lg(\frac{k+1}{2}) + 2$$

**** For this proof to work therefore, c must be ≥ 2 ****

$$c * (1 + lg(\frac{k+1}{2}))$$

$$c * (1 - lg(2) + lg(k+1))$$

$$c * lg(k+1) \leq c * lg(k+1)$$

Therefore proof is correct when c = 2 for any n_0.
Now let us write pseudocode for binary search.

```
function binarySearch(number, list){
if size(list) == 1 then
  | return list[0] == number;
end
if number > list[n/2] then
  | return binarySearch(number, list[(n/2 + 1) ... n]) ;
else
  | return binarySearch(number, list[1...(n/2 - 1])) ;
end
}
```

Testing:

- Base case
- Recursive step
- Edge case behaviors
  1. In list / Not in list
  2. Doesn't go past the upper and lower bound.

Graph Theory

- **Terminology.** vertex/node, edge, degree, walk, path, cycle, connected, connected component, adjacent, tree, leaf, root, undirected vs. directed graph
- **Prove the handshake lemma.** That is, that the sum of degrees in a graph is twice the number of edges.

  Proof: Each edge contributes twice to the sum of the degrees of all the vertices, an indegree and an outdegree. Now, prove this corollary: Every graph has an even number of vertices of odd degree.

Putting It All Together

Prove the following: A connected, undirected graph with n vertices contains at least $n-1$ edges.

- What does it mean for a graph to be connected? You'll want to use the precise definition here.
- What type of proof technique do you want to use here? n can be arbitrarily large, and it's an integer value...
- "At least" is a hint that the negation might be easier to work with.

Solution: We will prove this using induction on the number of vertices (you must induct on a specific variable while holding the other variables constant!).
Base case: $n = 0$ or $n = 1$. The graph cannot have any edges, so the claim holds.
Induction hypothesis: Assume that the statement is true for a graph with k vertices, $k \geq 0$.
Induction step: We will show it's true for a graph with $k+1$ vertices. Let G be an arbitrary graph with $k+1$ vertices. Suppose for contradiction that G is connected and contains at most $k-1$ edges. We know by the handshake lemma that the total degree of the graph must be at most $2(k-1) = 2k-2$. Thus there must exist some vertex v that has at most degree 1. If $\delta(v) = 0$, G is disconnected and we are done. Suppose $\delta(v) = 1$. Then remove v from the graph to form $G\prime = G - \{v\}$. It must be that $G\prime$ is connected (why?). Note that $G\prime$ has k vertices. However, it has at most $k-2$ edges, contradicting our induction hypothesis! Thus, if G is connected, then it must contain at least k edges, and we've proved our claim.