Data Structures and Algorithms — CSCI 230
Discrete Mathematics Overview — Part B

Proof by Induction

- Mathematical induction is used to prove statements true for all integers greater than or equal a certain base value n_0. Here are three example statements we will prove in class using mathematical induction:
  - $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ for $n \geq 0$ and $a \neq 1$.
  - $n! < n^n$, for $n \geq 2$.
  - A complete binary tree of height $h \geq 0$ contains exactly $2^{h+1} - 1$ nodes.
- An inductive proof always has three main components:

  Basis case: Prove the statement is true for $n = n_0$ (and perhaps $n = n_0 + 1$, etc.).

  Inductive hypothesis: Assume the statement is true for all k, $n_0 \leq k < n$.

  Induction step: Prove that for all $n \geq n_0$, the inductive hypothesis implies that the statement is true for n.
- The principle of mathematical induction states that if the basis case and induction step are both proved, then the statement holds for all $n \geq n_0$.
- Notes:
  - At first proofs by induction appear to be circular, but in fact they are not. Instead, think of a set of dominoes standing on end: the basis case is the act of pushing over the first domino (or two); the induction step is the relationship between the positions of the dominoes showing that for every domino, if its precedecessors fall over, it will fall over as well.
  - Sometimes the inductive hypothesis is written $n_0 \leq k \leq n$ and the induction step proves that the inductive hypothesis shows the statement to be true for $n + 1$.
  - At other times, the inductive step is to show for all $n \geq n_0$, if the statement is true for n then it is true for $n + 1$. Students more comfortable with this method of inductive proof are welcome to use it here.
  - Very often the inductive hypothesis is not written out explicitly.

Let's see how the three examples mentioned above are proved using mathematical induction.

1. $\displaystyle\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ for $n \geq 0$ and $a \neq 1$.

Basis case: When $n = 0$, the summation reduces to $a^0 = 1$, and the right side of the equation is $(a^{0+1} - 1)/(a - 1) = 1$. Since these are both 1, the basis case is proved.

Induction step: For any given $n \geq 1$, if the statement is true for $0 \leq k < n$, then

$$
\begin{aligned}
\sum_{i=0}^{n} a^i &= \textstyle\sum_{i=0}^{n-1} a^i \ + \ a^n \\
&= \tfrac{a^{(n-1)+1}-1}{a-1} \ + \ a^n \qquad \text{by the induction hypothesis} \\
&= \tfrac{a^n-1}{a-1} \ + \ a^n \tfrac{a-1}{a-1} \\
&= \tfrac{a^n-1+a^{n+1}-a^n}{a-1} \\
&= \tfrac{a^{n+1}-1}{a-1}
\end{aligned}
$$

completing the induction step.

2. $n! < n^n$, for $n \geq 2$.

Basis case: For $n = 2$, the left side of the inequality evaluates to 2 and the right side evaluates 4. Since $2 < 4$, the basis case is established.

Induction step: For $n > 2$, if $k! < k^k$ for $2 \leq k < n$ then in particular

$$(n-1)! < (n-1)^{n-1}.$$

Multiplying both sides of this inequality by n shows that

$$n! < n \cdot (n-1)^{n-1}.$$

But, since $n - 1 < n$ (obviously),

$$n! < n \cdot (n-1)^{n-1} < n \cdot n^{n-1}.$$

So,

$$n! < n^n.$$

3. A complete binary tree of height $h \geq 0$ contains exactly $2^{h+1} - 1$ nodes.

Basis case: Any binary tree of height $h = 0$ is complete and has exactly one node. Since $2^{0+1} - 1 = 2 - 1 = 1$, the basis case is established.

Induction step: For $h > 0$, assume any complete binary tree of height k, $0 \leq k \leq h-1$, contains $2^{k+1} - 1$ nodes, and consider any complete binary tree, T, of height h. Removing the root of T creates two complete binary trees of height $h-1$, one formed from the left subtree of T and the other formed from the right. By the inductive hypothesis each of these has $2^{(h-1)+1} - 1 = 2^h - 1$ nodes. Therefore, T had

$$(2^h - 1) + (2^h - 1) + 1$$

nodes to start with (the 1 at the end is for the root node). Adding these values shows that T had

$$2^{h+1} - 1$$

nodes, which completes the proof.

Class Exercises on Induction

1. Prove each of the following by mathematical induction.

   (a) $\displaystyle\sum_{i=1}^{n}(2i-1) = n^2$ for $n \geq 1$.

   Proof:

   Basis: For $n = 1$, the left hand side is $2 \times 1 - 1 = 1$ and the right hand side is $1^2 = 1$, so the basis case is established.

   Induction step: Assume $\sum_{i=1}^{n}(2i-1) = n^2$ for some $n \geq 1$. We show that $\sum_{i=1}^{n+1}(2i-1) = (n+1)^2$, as follows:

$$\begin{aligned}
\sum_{i=1}^{n+1}(2i-1) &= \sum_{i=1}^{n}(2i-1) + 2(n+1) - 1 \\
&= \sum_{i=1}^{n}(2i-1) + 2n + 1 \\
&= n^2 + 2n + 1 \quad \text{by the induction hypothesis} \\
&= (n+1)^2.
\end{aligned}$$

   (b) $n < 2^n$ for $n \geq 0$.

   Proof:

   Basis: For $n = 0$, we have $0 < 1 = 2^0$, which establishes the basis case.

Induction step: Assume $n < 2^n$ for some $n \geq 0$, we attempt to show that $n+1 < 2^{n+1}$:

$$\begin{array}{rcll} 2^{n+1} & = & 2 \times 2^n & \\ & > & 2n & \text{by the induction hypothesis} \end{array}$$

Hmmm, we need to know this is $\geq n+1$, but in fact $2n \not\geq n+1$ when $n = 0$. What to do?

Answer: We must go back and prove another basis case, for $n = 1$.

Basis: For $n = 1$, we have $1 < 2 = 2^1$, which establishes the second basis case. Next we have to restate and prove the induction step:

Induction step: Now assume $n < 2^n$ for some $n \geq 1$; we show that $n+1 < 2^{n+1}$:

$$\begin{array}{rcll} 2^{n+1} & = & 2 \times 2^n & \\ & > & 2n & \text{by the induction hypothesis} \\ & \geq & n+1 & \text{since } n \geq 1. \end{array}$$

(c) Any set containing n elements has 2^n different possible subsets.
Hint: In the inductive step, consider a particular element a of the set and count all the subsets that do contain a and all those that do not.

Basis: For $n = 0$, we have a set of 0 elements; i.e., the empty set, which has 1 subset, itself. Since $1 = 2^0$, this establishes the basis case.

Induction step: Let S be a set with $n > 0$ elements. Let a be a particular element of S. Let $\mathcal{T}$ be the set of all subsets of S and $\mathcal{T}_\dashv$ be the set of all subsets of $S - \{a\}$. By the induction hypothesis, $|T_a| = 2^{n-1}$. Every set in $\mathcal{T}_\dashv$ is also a subset of S, so $\mathcal{T}$ has at least 2^{n-1} elements. For every set U in $\mathcal{T}_\dashv$, we also have $U \cup \{a\} \subseteq S$, which gives 2^{n-1} more subsets of S in $\mathcal{T}$, for a total of $2^{n-1} + 2^{n-1} = 2^n$.

2. What is wrong with the following inductive proof showing that all horses are the same color? (Or, more specifically, in any set H of n horses, each horse in H has the same color as every other horse in H.)

Basis: $n = 1$. One horse is trivially the same color as itself.

Induction Step: We have to prove for each $n \geq 2$ that if in all sets containing fewer than n horses all horses are the same color, then in any set of n horses all horses are the same color. Let H be any set containing n horses, and label the horses $h_1, h_2, \ldots, h_n$. Let $H_1 = H - \{h_n\}$ and let $H_2 = H - \{h_1\}$. By the inductive hypothesis, since $|H_1| = n - 1$, all horses in H_1 are the same color; call it c_1. Also by the inductive hypothesis, since $|H_2| = n - 1$, all horses in H_2 are the same color; call it c_2. Also, since $H_1 \cap H_2 = \{h_2, \ldots, h_{n-1}\}$, horses $h_2, \ldots, h_{n-1}$ are each both colors c_1 and c_2. Hence, $c_1 = c_2$. Therefore, all horses in H are the same color.

Recursion

Here is an outline of five steps I find useful in writing and debugging recursive functions:

1. Handle the base case(s) first, at the start of the function.
2. Define the problem solution in terms of smaller instances of the problem. This defines the necessary recursive calls.
3. Figure out what work needs to be done before making the recursive call(s).
4. Figure out what work needs to be done after the recursive call(s) complete(s) to finish the solution.
5. Assume the recursive calls work correctly, but make sure they are progressing toward the base case(s)!

Here are two example recursive functions: factorial and mergesort.

```
int Fact(int n)
{
  if ( n == 0 || n == 1 )
    return 1;
  else
    return n * Fact(n-1);
}

template <class T>
void MergeSort(T * pts, int n)
{
  MergeSort(pts, 0, n-1);
}

templage <class T>
void MergeSort( T * pts, int low, int high )
{
  if ( low == high ) return;

  int mid = (low + high) / 2;
  MergeSort(T, low, mid);
  MergeSort(T, mid+1, high);

  //  At this point the lower and upper halves
  //  of "pts" are sorted. All that remains is
  //  to merge them into a single sorted list.
  //  We will discuss the details of this later.
}
```

Class Exercises on Recursion

1. Assume the following structure for a node in a binary tree:

```
struct TreeNode {
  int value;
  TreeNode * left_child;
  TreeNode * right_child;
};
```

   Write a recursive function to count the number of nodes in the binary tree whose root is pointed to by `T`. Note the structural similarity between your solution and the inductive proof about complete binary trees.

2. The following is a recursive version of binary search. Will it work correctly? Why or why not?

```
template <class T>
bool RBinSearch(T * pts, int low, int high,
               T point, int& loc)
{
  if (high == low) {
    loc = low;
    return pts[loc] == point;
  }
  int mid = (low + high) / 2;
  if (pts[mid] <= point)
    return RBinSearch(pts, mid, high, point, loc);
  else
    return RBinSearch(pts, low, mid-1, point, loc);
}
```

3. The Fibonacci numbers are defined recursively as

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2}, \qquad n \geq 2 \end{aligned}$$

   (The notation f_n means the same thing as $f(n)$, i.e. f is a function of n.)

   (a) Write a recursive function to calculate f_n.

   (b) How many recursive calls will your function make to calculate f_3, f_4, f_5, etc? Can you write a recursive expression similar to the recursive formula for the Fibonacci numbers to count the number of recursive calls?

Review Problems

Here are a few review problems which have appeared on homeworks or tests in previous semesters. Practice writing solutions carefully and then compare to solutions provided on-line. If you can solve these problems and the problems we worked on in class then you are ready for the chapter quiz!

1. Give an inductive proof showing that for all integers $n \geq 5$, $4n + 4 < n^2$.
2. Give an inductive proof showing that for all positive integers n,
$$\sum_{i=1}^{n} \frac{1}{(2i-1)(2i+1)} = \frac{n}{2n+1}.$$
3. Give an inductive proof showing that for all $n \geq 1$,
$$\sum_{i=1}^{n} i(i!) = (n+1)! - 1$$
4. Evaluate the following summations.
   (a) $\displaystyle\sum_{i=0}^{100} (-1)^i$
   (b) $\displaystyle\sum_{i=5}^{2n} i$
   (c) $\displaystyle\sum_{i=0}^{n} (a^i - 2i + n)$
   (d) $\displaystyle\sum_{i=0}^{n-1} \big(c + \sum_{j=0}^{i-1} (j+2)\big)$
5. Prove using mathematical induction that $f_n > (3/2)^n$, for $n \geq 5$. Here, f_n is the nth Fibonacci number. Use the definition of the Fibonacci numbers given in the text on page 6. Study the example inductive proof on page 6 carefully.
6. Write the code necessary to accomplish the merging step in `MergeSort`. This code should follow the comments in the main `MergeSort` function.