Induction and Recursion

IE 496 Lecture 5

Reading for this lecture

- Miller and Boxer, Chapters 2 and 3
- Aho, Hopcroft, and Ullman, Sections 2.5-2.9

Induction and Recursion

Mathematical Induction

- Induction is a technique for proving statements about consecutive integers.
- Principle of Mathematical Induction
  - Let *P(n)* be a predicate that we want to prove TRUE for all positive *n*.
  - Method:
    1. Prove *P*(1).
    2. Prove $P(k) \Rightarrow P(k+1) \ \forall k \geq 1$.
- Example: Prove $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- What does induction have to do with programming?

Recursion

- Definition (Mathematics): An expression, each term of which is determined by application of a formula to the preceding terms.
- In CS, a function that calls itself is called *recursive*.
- Recursion allows us to process large data sets based on our knowledge of smaller ones.
- The correctness and complexity of recursive algorithms can be proven by induction.
- Example: The *factorial* function.

Analyzing Recursive Algorithms

- For recursive algorithms, running times can also be defined recursively.
- Example:

  $T(0) = 1$
  $T(n) = T(n\text{-}1) + 1$

- What is T(n)?
- In general, we have something like $T(n) = aT(n/b - c) + f(n)$.

Example: Binary Search

- This example is in Miller and Boxer, p.37

Divide and Conquer

- A common approach is to divide the problem into smaller parts and solve each part independently.
- In this case, our recurrence relation looks like

$$T(n) = S(n) + aT(n/b) + C(n)$$

- $S(n)$ = time to split
- $C(n)$ = time to combine

Example: Merge Sort

- This example is in Miller and Boxer, p. 41

Balancing

- When splitting, it usually makes sense to split in such a way that the two resulting subproblems are of approximately equal size.
- If not, the recurrence relation may not hold, and efficiency will decrease.
- However, balancing is not always easy to do.
- Improper balancing can cause problems for parallel algorithms (more on this later).

Example: Multiplying *n*-bit numbers

- This example is in AHU, p. 62.

The Master Theorem

- The Master Theorem provides the solution to the most commonly occurring recursions of the form:

$$T(n) = f(n) + T(n/b)$$

- The solution to such recursions is given by

1. Suppose $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$. Then $T(n) = \Theta(n^{\log_b a})$.
2. Suppose $f(n) = \Theta(n^{\log_b a})$. Then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. Suppose $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and there are constants $c < 1$ and $N > 0$ such that $(n/b) > N \Rightarrow af(n/b) \leq cf(n)$. Then $T(n) = \Theta(f(n))$.

Implications for Parallelism

- Recursive algorithms have natural parallel formulations.
  - Split the problem until there is one part per processor.
  - Perform a sequential algorithm on each part in parallel.
  - Combine the results.
- Must be cognizant of the overhead involved in splitting and combining.
- The splitting and combining steps are essentially sequential steps that can harm parallelism.