IE170: Algorithms in Systems Engineering: Lecture 5

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

January 24, 2007

Taking Stock

Last Time

- In-Place, Out-of-Place
- Count Von Count
- Worst Case Analysis
- The world's easiest lab :-)

This Time

- Divide-and-Conquer
- The Master-Theorem

Divide and Conquer

A Simple Three Part Plan

1. Divide: the problem into a number of subproblems
2. Conquer: the subproblems by solving them recursively. (If subproblems are small enough, you solve them by "brute force." (constant time).
3. Combine: the subproblem solutions to give a solution to the original problem

The Towers of Hanoi

- An old famous puzzle mathematical puzzle
- It consists of three pegs, and a number of disks of different sizes which can slide onto any peg.
- The puzzle starts with the disks neatly stacked in order of size on one peg, smallest at the top, thus making a conical shape.
- The objective of the game is to move the entire stack to another peg

That's a Bit Too Easy

Three Rules

1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the pegs and sliding it onto another peg, on top of the other disks that may already be present on that peg.
3. No disk may be placed on top of a smaller disk.

The Legend

- At the beginning of the universe, three posts and 64 disks were created.
- An a secret monestary, monks have been moving these disks according to the rules of the puzzle.
- When the puzzle is completed, the world will end!

Who Says I Don't Teach You Anything Important?

After today, we will know when the world will end!

How clever are you?

- Label the pegs A, B, C
- Number the disks 1 (smallest), n (largest)

To move n disks from A to B

1. Move $n-1$ disks from A to C. This leaves the nth disk alone on peg A;
2. Move the nth disk from A to B
3. Move $n-1$ disks from C to B so they sit on the nth disk.

- This is a (canonical) example of a recursive algorithm: To do steps 1 and 3, just do the same algorithm, but for a problem of size $n-1$

Another explanation

- Move disk 1 to peg B
- Move disk 2 to peg C
- Move disk 1 from B to C, so it sits on 2
- (You now have 2 disks stacked correctly on peg C, peg B is empty again)
- Move disk 3 to peg B
- repeat the first 3 steps above to move 1 & 2 to sit on top of 3

Here it is in Java

```
public void hanoy(char a, char b, char c, int n)
{
  if (n==1) {
    moveit(1, a, b);
  }
  else {
    hanoy(a, c, b, n-1);
    moveit(n, a, b);
    hanoy(c, b, a, n-1);
  }
}
```

- Also posted on course web site

Count Von Count

Running Time of Towers of Hanoi Algorithm

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n-1) + \Theta(1) + T(n-1) & \text{if } n > 1 \end{cases}$$

Merge Sort

- Another sorting algorithm (that is very practical for your daily life) is based on recursion
- It is called Merge Sort and sorts the elements of a (sub)-array $A[p \ldots r]$ in the following manner:

```
MERGE-SORT$(A, p, r)$
1  if $p < r$
2     then $q \leftarrow \lfloor (p+r)/2 \rfloor$
3           MERGE-SORT$(A, p, q)$
4           MERGE-SORT$(A, q+1, r)$
5           MERGE$(A, p, q, r)$
```

Can you merge in linear time?

- The method MERGE(A, p, q, r) takes sorted subarrays $A[p \ldots q]$ and $A[q+1, \ldots r]$ and merges them into one sorted array
- How long does this take to do?

Let's Have Some Fun!

- Merge Sort
- Insertion Sort
- Selection Sort
- Bubble Sort

CountVonCount

Running Time of Merge-Sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Analyzing Recurrences

Deep Thoughts

To understand recursion, we must first understand recursion

- General methods for analyzing recurrences
  - Substitution: You learned this already
  - Master Theorem
  - Generating Functions: You won't learn this here
- Note that when we analyze a recurrence, we may not get or need an exact answer, only an asymptotic one
- We may prove the running time is in $O(f)$ or $\Theta(f)$

Good Stuff

- If you are only concerned about the asymptotic behavior of a recurrence, then
  1. You can ignore floors and ceilings: (Asymptotic behavior doesn't care if you round down or up)
  2. We assume that all algorithms run in $\Theta(1)$ (Constant time) for a small enough fixed input size n. This makes the base case of induction easy.

The Master Theorem

- Most recurrences that we will be interested in are of the form

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

- The Master Theorem tells us how to analyze recurrences of this form.
  - If $f \in O(n^{\log_b a - \varepsilon})$, for some constant $\varepsilon > 0$, then $T \in \Theta(n^{\log_b a})$.
  - If $f \in \Theta(n^{\log_b a})$, then $T \in \Theta(n^{\log_b a} \lg n)$.
  - If $f \in \Omega(n^{\log_b a + \varepsilon})$, for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and $n > n_0$, then $T \in \Theta(f)$.
- How do we interpret this?

Using the Master Theorem: Merge Sort

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

- We can analyze these using the Master Theorem.
- $T(n) = O(n \lg n)$

Towers of Hanoi

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n-1) + \Theta(1) + T(n-1) & \text{if } n > 1 \end{cases}$$

- We cannot analyze this using the Master Theorem
- Let's use guessing to establish that $T(n) = 2^n = 1$
- $2^{64} - 1 = 1.8847 \times 10^{19}$. I
- If the monks more one disk per second, this is 584.9 billion years. This is about 42 times older than the best estimate of the universe's current age.

The Call Stack

- The call stack of a program keeps track of the current sequence of function calls.
- When a new function call is made, data for the current one is saved on the call stack.
- When a function call returns, it returns to the next function on the top of the stack.
- The stack depth is the maximum number of functions on the stack at any one time.
- In a recursive program, the stack depth can be very large.
- This can create memory problems, even for simple recursive programs.
- There is also an overhead associated with each function call.

Next Time?

- Some more on the Master Theorem
- Some Data Structures
- Java implementations of data structures