Notes from Recitation 1, September 8, 2006

Horner’s Rule

We can specify an $n$th degree polynomial specified by giving $A[i]$, the coefficient of the $x^i$ term. Thus, a sequence of coefficients $A[0], A[1], A[2], \ldots A[n]$ corresponds to a polynomial


The problem is, given an array $A[0..n]$ and a value for $x$, compute $P(A[0..n], x)$. Some first attempts:

1. One straightforward way to compute $P(A[0..n], x)$ is to compute one term at a time, using $k$ multiplications to compute $A[k] \cdot x^k$. This algorithm takes $\Theta(n^2)$ time (the number of multiplications is $\sum_{i=1}^n i = O(n^2)$, and the number of additions is $n - 1$).

   Aside from the array of coefficients, this algorithm uses $O(1)$ additional space to store the current sum and temporary variables.

2. If we use a fast exponentiation algorithm to compute $x^n$, then computing $x^n$ requires $\Theta(\log n)$ operations. Then we can compute $P(A[0..n], x)$ in $\Theta(n \log n)$ time, using $O(1)$ additional space.

3. If we store the values of $x, x^2, \ldots x^n$ in a table, then we can do the computation in $\Theta(n)$ time, but using $\Theta(n)$ additional space.

Horner’s rule can do this computation in $\Theta(n)$ time, using $O(1)$ additional space.

Horner($A[0..n]$, $x$)

$$y \leftarrow A[n];$$

$$i \leftarrow n;$$

while ($i > 0$)

$$i \leftarrow i - 1;$$

$$y \leftarrow x \cdot y + A[i];$$

Follow the method for proving correctness of an iterative algorithm:

1. What is a good loop invariant $I$? Look at the value of $y$ after a few iterations:

$$i = n \implies y = A[n]$$

$$i = n - 1 \implies y = A[n] \cdot x + A[n - 1]$$

$$i = n - 2 \implies y = A[n] \cdot x^2 + A[n - 1] \cdot x + A[n - 2]$$

$$\vdots$$

Try a loop invariant

$$y = \sum_{j=1}^n A[j]x^{j-1}.$$ 

2. Proof the invariant holds by induction. For simplicity, we perform induction “backwards” on $i$, i.e., starting with a base case when $i = n$, and decreasing $i$. 

1
• **Base Case:** When \( i = n \), \( y = A[n] \).

• **Inductive Step:** Assume for \( i = k \) (or alternatively, for \( n \geq i \geq k \)), that \( y = \sum_{j=k}^{n} A[j]x^{j-k} \).

  We want to show for \( i = k - 1 \), that \( y = \sum_{j=k-1}^{n} A[j]x^{j-(k-1)} \).

  After the loop when \( i = k - 1 \), we compute a new \( y' = x \ast y + A[k - 1] \). This gives us

\[
\begin{align*}
  y' &= x \cdot \left( \sum_{j=k}^{n} A[j]x^{j-k} \right) + A[k - 1] \\
  &= \left( \sum_{j=k}^{n} A[j]x^{j-k+1} \right) + A[k - 1] \\
  &= \left( \sum_{j=k}^{n} A[j]x^{j-(k-1)} \right) + A[k - 1]x^0 \\
  &= \sum_{j=k-1}^{n} A[j]x^{j-(k-1)} \\
  &= \sum_{j=1}^{n} A[j]x^{j-1}
\end{align*}
\]

3. When the loop terminates, \( i = 0 \), so \( y = \sum_{j=0}^{n} A[j]x^{j} = P(A[0..n], x) \). This is exactly the value we were trying to compute.

4. The while loop always gets executed exactly \( n \) times, as long as \( n \) is nonnegative.