Jade Yu Cheng
ICS 311
Homework 14
Oct 30, 2008

Question for lecture 18

Problem 30-1 on p. 844

Divide-and-conquer multiplication

a. Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only three multiplications.

Answer: First let's consider the naïve method of multiplying polynomials. We would do the operations as follow:

$$
\begin{aligned}
&(a \cdot x + b)(c \cdot x + d) \\
&= ac \cdot x^2 + (bc + ad) \cdot x + bd
\end{aligned}
$$

Multiplication #: 4
Operations: ac, bc, ad, bd

Now, let's consider the modified method of doing the same polynomial multiplication:

$$
\begin{aligned}
&(a \cdot x + b)(c \cdot x + d) \\
&= ac \cdot x^2 + [(a + b)(c + d) - ac - db] \cdot x + bd \\
&= ac \cdot x^2 + (bc + ad) \cdot x + bd
\end{aligned}
$$

Multiplication #: 3
Operations: $(a + b)(c + d)$, ac, bd

This way we converted the calculation of looking for the coefficients for the output polynomial into three times of multiplications and a bunch of summations and subtraction. As we know the summation and subtraction operations are considerably cheaper than multiplications. For polynomial summation or subtraction, it takes $O(n)$, which would always be the lower term in comparison with polynomial multiplication. Let's write the equation above more clear:

If
$$A=(a+b)(c+d)$$
$$B=ac$$
$$C=ad$$

Then
$$(a\cdot x+b)(c\cdot x+d)$$
$$=B\cdot x^2+(A-B-C)\cdot x+C$$

Multiplication operations: A,B,C

b. Give two divide-and-conquer algorithms for multiplying two polynomials of degree-bound n that run in time $O(n^{lg3})$ The first algorithm should divide the input polynomial coefficients into a high half and a low half, and the second algorithm should divide them according to whether their index is odd or even.

Answer: First let's consider the naïve method of multiplying polynomials. We would do the operations as follow:

$$A(x)=a_1\cdot x^{n-1}+a_2\cdot x^{n-2}+\cdots+a_{n-1}\cdot x+a_n$$
$$B(x)=b_1\cdot x^{n-1}+b_2\cdot x^{n-2}+\cdots+b_{n-1}\cdot x+b_n$$
$$A(x)\cdot B(x)=(a_1\cdot x^{n-1}+a_2\cdot x^{n-2}+\cdots+a_{n-1}\cdot x+a_n)\cdot(b_1\cdot x^{n-1}+b_2\cdot x^{n-2}+\cdots+b_{n-1}\cdot x+b_n)$$

Multiplication #: n^2
Operations: $a_i\cdot b_j$ while $i,j=\{1,2,\cdots n\}$

Algorithm 1:

Now, let's consider the modified method of doing the same polynomial multiplication:

$$A(x)\cdot B(x)=(a_1\cdot x^{n-1}+a_2\cdot x^{n-2}+\cdots+a_{n-1}\cdot x+a_n)\cdot(b_1\cdot x^{n-1}+b_2\cdot x^{n-2}+\cdots+b_{n-1}\cdot x+b_n)$$
$$=[(a_1\cdot x^{n-1}+a_2\cdot x^{n-2}+\cdots+a_{n/2}\cdot x^{n/2})+(a_{n/2+1}\cdot x^{n/2-1}+a_{n/2+2}\cdot x^{n/2-2}+\cdots+a_n)]\times$$
$$[(b_1\cdot x^{n-1}+b_2\cdot x^{n-2}+\cdots+b_{n/2}\cdot x^{n/2})+(b_{n/2+1}\cdot x^{n/2-1}+b_{n/2+2}\cdot x^{n/2-2}+\cdots+b_n)]$$
$$=[x^{n/2}\cdot(a_1\cdot x^{n/2-1}+a_2\cdot x^{n/2-2}+\cdots+a_{n/2})+(a_{n/2+1}\cdot x^{n/2-1}+a_{n/2+2}\cdot x^{n/2-2}+\cdots+a_n)]\times$$
$$[x^{n/2}\cdot(b_1\cdot x^{n/2-1}+b_2\cdot x^{n/2-2}+\cdots+b_{n/2})+(b_{n/2+1}\cdot x^{n/2-1}+b_{n/2+2}\cdot x^{n/2-2}+\cdots+b_n)]$$

If

$$f_1(x) = a_1 \cdot x^{n/2-1} + a_2 \cdot x^{n/2-2} + \cdots + a_{n/2}$$
$$f_2(x) = a_{n/2+1} \cdot x^{n/2-1} + a_{n/2+2} \cdot x^{n/2-2} + \cdots + a_n$$
$$f_3(x) = b_1 \cdot x^{n/2-1} + b_2 \cdot x^{n/2-2} + \cdots + b_{n/2}$$
$$f_4(x) = b_{n/2+1} \cdot x^{n/2-1} + b_{n/2+2} \cdot x^{n/2-2} + \cdots + b_n$$

Then

$$A(x) = x^{n/2} \cdot f_1(x) + f_2(x)$$
$$B(x) = x^{n/2} \cdot f_3(x) + f_4(x)$$

The objective function: $A(x) \cdot B(x) = \left[x^{n/2} \cdot f_1(x) + f_2(x)\right] \times \left[x^{n/2} \cdot f_3(x) + f_4(x)\right]$

By looking at the last equation, if we consider $f_1(x), f_2(x), f_3(x), f_4(x)$ as coefficients, we see that the problem is turned into a $(a \cdot x + b)(c \cdot x + d)$ form, which was discussed in part 1 of this paper. The conclusion was polynomial multiplication of this from can be accomplished with 3 operations.

If we look at the form of $A(x)$ and $B(x)$, we divide the problem by half of it size and added a summation operation. As we discussed, the polynomial summation and subtraction takes $O(n)$ time to compute. Therefore, we have a recurrence solution for our problem:

$$T[n] = 3 \cdot T\left[\frac{n}{2}\right] + O(n)$$

We apply master method to solve this recurrence. We obtain the final run time of the polynomial multiplication using this method. Since $O\left(n^{\lg_2 3}\right)$ is higher term than $O(n)$. Algorithm run in $O\left(n^{\lg_2 3}\right)$ time

Algorithm 2:

Let's divide the objective function in a different way. Instead of a higher half and a lower half, we can also do odd index half and the even index half:

$$A(x) \cdot B(x) = \left(a_1 \cdot x^{n-1} + a_2 \cdot x^{n-2} + \cdots + a_{n-1} \cdot x + a_n\right) \cdot \left(b_1 \cdot x^{n-1} + b_2 \cdot x^{n-2} + \cdots + b_{n-1} \cdot x + b_n\right)$$
$$= \left[\left(a_1 \cdot x^{n-1} + a_3 \cdot x^{n-3} + \cdots + a_{n-1} \cdot x\right) + \left(a_2 \cdot x^{n-2} + a_4 \cdot x^{n-4} + \cdots + a_n\right)\right] \times$$
$$\left[\left(b_1 \cdot x^{n-1} + b_3 \cdot x^{n-3} + \cdots + b_{n-1} \cdot x\right) + \left(b_2 \cdot x^{n-2} + b_4 \cdot x^{n-4} + \cdots + b_n\right)\right]$$
$$= \left[x \cdot \left(a_1 \cdot x^{n-2} + a_3 \cdot x^{n-4} + \cdots + a_{n-1}\right) + \left(a_2 \cdot x^{n-2} + a_4 \cdot x^{n-4} + \cdots + a_n\right)\right] \times$$
$$\left[x \cdot \left(b_1 \cdot x^{n-2} + b_3 \cdot x^{n-4} + \cdots + b_{n-1}\right) + \left(b_2 \cdot x^{n-2} + b_4 \cdot x^{n-4} + \cdots + b_n\right)\right]$$

If

$$g_1(x) = a_1 \cdot x^{n-2} + a_3 \cdot x^{n-4} + \cdots + a_{n-1}$$
$$g_2(x) = a_2 \cdot x^{n-2} + a_4 \cdot x^{n-4} + \cdots + a_n$$
$$g_3(x) = b_1 \cdot x^{n-2} + b_3 \cdot x^{n-4} + \cdots + b_{n-1}$$
$$g_4(x) = b_2 \cdot x^{n-2} + b_4 \cdot x^{n-4} + \cdots + b_n$$

Then

$$A(x) = x \cdot g_1(x) + g_2(x)$$
$$B(x) = x \cdot g_3(x) + g_4(x)$$

The objective function: $A(x) \cdot B(x) = [x \cdot g_1(x) + g_2(x)] \times [x \cdot g_3(x) + g_4(x)]$

As we see, we again converted the objective function into a $(a \cdot x + b)(c \cdot x + d)$ form. The run time analysis for this algorithm, therefore, is also $O(n^{\lg_2 3})$

c. Show that two n-bit integers can be multiplied in $O(n^{\lg_2 3})$ steps, where each step operates on at most a constant number of 1-bit values.

Answer: We have two n-bit integers, $k_n k_{n-1} \cdots k_2 k_1$ and $l_n l_{n-1} \cdots l_2 l_1$, where $0 \le k_i \le 10$ and $0 \le l_i \le 10$, k_i, l_i represent the number on every digit, $i = \{1, 2, \cdots, n\}$. Obviously, we can also write them as:

$$k_n \cdot 10^{n-1} + k_{n-1} \cdot 10^{n-2} + \cdots + k_2 \cdot 10 + k_1$$
$$l_n \cdot 10^{n-1} + l_{n-1} \cdot 10^{n-2} + \cdots + l_2 \cdot 10 + l_1$$

Therefore the multiplication of these two integers is turned into a polynomial multiplication problem. As we discussed above, by using divide-and-conquer, we designed two algorithms to solve the polynomial multiplication problem of degree-bound n with a run time $O(n^{\lg_2 3})$. For each operation here, we only deal with either k or l. They are integers with only 1-bit values. Therefore we can multiplicity two n-bit integers with $O(n^{\lg_2 3})$ steps, where each step operates on at most a constant number of 1-bit values.