Royal Institute of Technology
DD2440 Advanced Algorithms
Lecturer: Johan Håstad

Lecture 7

Notes by Carl-Johan Backman

2014-11-12

1 Arithmetics

In this lecture we want to examine how to perform fast arithmetic (multiplication and division) of n-bit integers.

Table 1: Some examples of time complexity for performing arithmetic operations

Operation	Complexity
Addition and subtraction	$\mathcal{O}(n)$
Naive multiplication	$\mathcal{O}(n^2)$
Naive division	$\mathcal{O}(n^2)$

The naive division is defined as: given $a, b \in \mathbb{N}$, n-bit integers, find a decimal number x such that $|a/b - x| \leq 2^{-n}$.

1.1 Multiplication

We can think about an integer as a polynomial with different coefficients. Consider the following equations

$$a = \sum_{i=0}^{n-1} a_i 2^i \tag{1}$$

$$P_A(x) = \sum_{i=0}^{n-1} a_i x^i \tag{2}$$

$$\tag{3}$$

and note that they are almost the same. Use $\beta = 2^t$ for some t and we have

$$\sum_{i=0}^{\frac{n}{t}-1} a_i \beta^i. \tag{4}$$

Thus, we can understand that multiplying two integers is approximately the same thing as multiplying two polynomials.

How should we perform the polynomial multiplication $P_A \cdot P_B$, of two degree $d-1$ polynomials? Instead of multiplying them coefficient by coefficient we use the following method

1. Evaluate at $2d-1$ point x_i, where d is a constant.
2. $y_i = P_A(x_i) \cdot P_B(x_i)$
3. Interpolate y_i with $P(x_i) = y_i$.

Karatsuba uses $d = 2$. Using this method the two steps 1 and 3 each takes $\mathcal{O}(n)$ and the complexity is dominated by the recursive call in step 2. The overall complexity turns out to be $O(n^\gamma)$ where $\gamma = \log_d 2d - 1$.

For small d (2,3 etcetera) the choice of good points affect constants in running time. But if d is close to n we need to be more careful.

1.1.1 Discrete Fourier Transform

Definition 1. *Let $f_0, f_1, \ldots, f_{m-1}$ be a sequence of length m, then the* discrete Fourier transform (DFT) *of this sequence is*

$$\hat{f}_j = \sum_{k=0}^{m-1} f_k w^{kj}, \tag{5}$$

where $j = 0, ..., m-1$ and w is an m^{th} root of unity, i.e. $w^m = 1$, $w^i \neq 1$ for $1 \leq i \leq m-1$.

Example 1. *If we have $w = e^{2\pi i/m}$, w will be an m^{th} root of unity. In the figure below the these roots are shown in the unit circle when $m = 12$.*

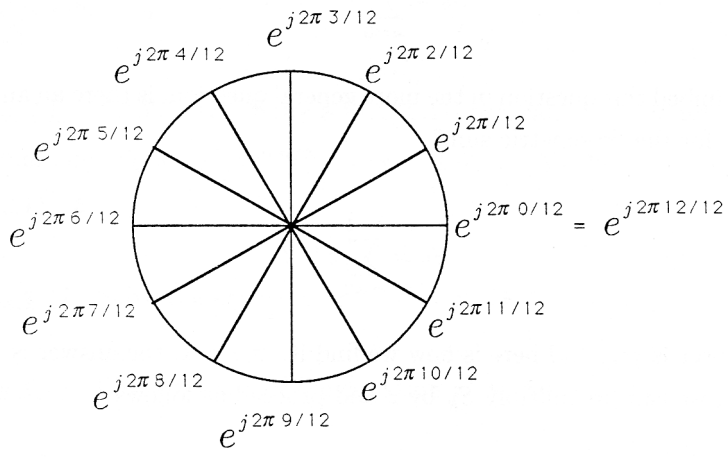

Figure 1: The roots of $w = e^{2\pi i/m}$ when $m = 12$.

There are some important computational advantages when using DFT, namely (they will be described in more detail below)

1. Easy to compute (with fast Fourier transform (FFT))
2. Polynomial evaluation: $f \approx f_0 + f_1 \cdot x + f_2 \cdot x^2 + \cdots + f_{m-1} \cdot x^{m-1}$ then $\hat{f}_j = f(w^j)$.
3. Easy to invert

3. The DFT is easy to invert, because it is almost its own inverse.

Theorem 1.

$$\sum_{j=0}^{m-1} \hat{f}_j w^{-ij} = m \cdot f_i. \tag{6}$$

Proof.

$$\sum_{j=0}^{m-1}\sum_{k=0}^{m-1} f_k w^{jk} w^{-ij} = \sum_{k=0}^{m-1} f_k \left(\sum_{j=0}^{m-1} w^{kj-ij} \right) = m \cdot f_j, \tag{7}$$

because sum inside the parenthesis is m if $(k = i)$ and 0 otherwise. □

1. The DFT is easy to compute with FFT.

Theorem 2. *If m is a power of 2 then DFT can be computed in $\mathcal{O}(m \log m)$ operations. This is called the fast Fourier transform.*

Let $m = 2^+$, and w an m^{th} root of unity. Now we will think of this as making $m/2$ dimensional FFT:s of odds and evens. Setting $\sigma = w^2$ we see that σ is $m/2^{\text{th}}$ root of unity. Let

$$f_j^e = \sum_{k=0}^{\frac{m}{2}-1} f_{2k} \sigma^{kj}. \tag{8}$$

be the transform of the even elements. Note that as $w^{2kj} = \sigma^{kj}$ this is in fact a partial sum from the full transform. Similarly we have for the odd elements

$$f_j^o = \sum_{k=0}^{\frac{m}{2}-1} f_{2k+1} \sigma^{kj}. \tag{9}$$

By identification of terms we then have the complete transform

$$\sum_{k=0}^{m-1} f_k w^{kj} = \sum_{k=0}^{m/2-1} f_{2k} w^{2kj} + \sum_{k=0}^{m/2-1} f_{2k+1} w^{(2k+1)j} = f_j^e + w^j f_j^o, \tag{10}$$

and

$$\hat{f}_j = \hat{f}_j^e + w^j f_j^o = \hat{f}_{j-m/2}^e + w^j \hat{f}_{j-m/2}^o. \tag{11}$$

where $j = m/2, m/2+1, \ldots, m-1$.

Thus we can calculate the Fourier transform with the help of a divide and conquer strategy. By recursion we compute the transforms of the odd and even terms and then we use (10) and (11) to complete the calculation. Since the latter costs $\mathcal{O}(m)$ arithmetic operations, if we let $T(m)$ be the number of operations needed for a transform of size m we get

$$T(m) = 2T(m/2) + \mathcal{O}(m), \tag{12}$$

which solves to $T(m) = \mathcal{O}(m \log m)$, as stated in Theorem 2.

1.1.2 Some Final Notes on the Fourier Transform

When solving the Fourier transform we want integer result. All the intermediate results are floating point complex numbers (pairs of reals). The rule is to do calculations with enough accuracy to make rounding to closest integer the correct result. $\mathcal{O}(\log m)$ bits is enough. In practice, even double is good enough.

For multiplication $d = m = n/\log n$, for each coefficient $\log n$ bits. $M(n) = O(m \log m\, M(\log m)) = O(nM(\log n))$, i.e. n operations on $\mathcal{O}(\log n)$ bit numbers. Some efforts in increasing the efficiency has been made

- Schönhage-Strassen 1971: Do not use complex numbers $\Rightarrow M(n) = \mathcal{O}(n \log n \log\log n)$
- Fürer 2005: $M(n) = \mathcal{O}(n \log n\, 2^{O(\log^* n)})$

1.2 Division

Compute $1/y$ with n bits of accuracy when $1 \leq y \leq 2$. x/y is found by inversion and multiplication.

Put $x_0 = 1/2$, and let

$$x_{i+1} = 2x_i - yx_i^2. \tag{13}$$

x_i will then become better and better approximations of $1/y$. $|x_i - 1/y| \leq 2^{-2^i}$ i.e. x_i will be $1/y$ with 2^i bits of accuracy. If we massage the product $x_{i+1}\, y$ we can rewrite it as

$$x_{i+1}\, y - 1 = 2x_i y - x_i^2 y^2 - 1 = -(x_i y - 1)^2. \tag{14}$$

Now we have that $|x_0 y - 1| \leq 1/2$ as $x_0 = 1/2$ and $1 \leq y \leq 2$. Then, by induction

$$|x_{i+1} y - 1| = |x_i - 1|^2 \leq (2^{-2^i})^2 = 2^{-2^{i+1}}. \tag{15}$$

So $log\, n$ iterations gives n bits of accuracy! Each iteration cost $\mathcal{O}(M(n))$, i.e. the cost of multiplication. Thus, we get the following theorem

Theorem 3. *Cost of division is $\mathcal{O}(log\, n\ M(n))$, where $M(n)$ is the cost of multiplication.*

We can do better yet, tough, since x_i will have error $\approx 2^{-2^i}$ we compute it with $2^i + 5$ bits of accuracy. Then the cost of division is $\sum_{i=0}^{log\, n} M(2^i + 5) = \mathcal{O}(M(n))$.