Applied Symbolic Computation
(CS 300)

Karatsuba's Algorithm for Integer Multiplication

Jeremy R. Johnson

Introduction

- **Objective: To derive a family of asymptotically fast integer multiplication algorithms using polynomial interpolation**
  - **Karatsuba's Algorithm**
  - **Polynomial algebra**
  - **Interpolation**
  - **Vandermonde Matrices**
  - **Toom-Cook algorithm**
  - **Polynomial multiplication using interpolation**
  - **Faster algorithms for integer multiplication**

References: Lipson, Cormen et al.

Karatsuba's Algorithm

- **Using the classical pen and paper algorithm two n digit integers can be multiplied in $O(n^2)$ operations. Karatsuba came up with a faster algorithm.**
- **Let A and B be two integers with**
  - $A = A_1 10^k + A_0$, $A_0 < 10^k$
  - $B = B_1 10^k + B_0$, $B_0 < 10^k$
  - $C = A*B = (A_1 10^k + A_0)(B_1 10^k + B_0)$
    $= A_1 B_1 10^{2k} + (A_1 B_0 + A_0 B_1)10^k + A_0 B_0$

Instead this can be computed with 3 multiplications

- $T_0 = A_0 B_0$
- $T_1 = (A_1 + A_0)(B_1 + B_0)$
- $T_2 = A_1 B_1$
- $C = T_2 10^{2k} + (T_1 - T_0 - T_2)10^k + T_0$

Complexity of Karatsuba's Algorithm

- **Let T(n) be the time to compute the product of two n-digit numbers using Karatsuba's algorithm. Assume $n = 2^k$. $T(n) = \Theta(n^{lg(3)})$, $lg(3) \approx 1.58$**

- **$T(n) \leq 3T(n/2) + cn$**

  $\leq 3(3T(n/4) + c(n/2)) + cn = 3^2T(n/2^2) + cn(3/2 + 1)$

  $\leq 3^2(3T(n/2^3) + c(n/4)) + cn(3/2 + 1)$

  $= 3^3T(n/2^3) + cn(3^2/2^2 + 3/2 + 1)$

  …

  $\leq 3^iT(n/2^i) + cn(3^{i-1}/2^{i-1} + \ldots + 3/2 + 1)$

  ...

  $\leq cn[((3/2)^k - 1)/(3/2 - 1)]$ --- Assuming $T(1) \leq c$

  $\leq 2c(3^k - 2^k) \leq 2c3^{lg(n)} = 2cn^{lg(3)}$

Divide & Conquer Recurrence

Assume $T(n) = aT(n/b) + \Theta(n)$

- **$T(n) = \Theta(n)$ [a < b]**
- **$T(n) = \Theta(n\log(n))$ [a = b]**
- **$T(n) = \Theta(n^{\log_b(a)})$ [a > b]**

Polynomial Algebra

- **Let F[x] denote the set of polynomials in the variable x whose coefficients are in the field F.**
- **F[x] becomes an algebra where +, * are defined by polynomial addition and multiplication.**

$$A(x) = \sum_{i=0}^{m} a_i x^i, B(x) = \sum_{j=0}^{n} b_j x^j$$

$$C(x) = A(x)B(x) = \sum_{k=0}^{m+n} c_k x^k,$$

$$c_k = \sum_{k=i+j} a_i b_j = \sum_{i=\max(0,k-n)}^{\min(k,m)} a_i b_{k-i}$$

Interpolation

- **A polynomial of degree n is uniquely determined by its value at (n+1) distinct points.**

Theorem: Let A(x) and B(x) be polynomials of degree m. If $A(\alpha_i) = B(\alpha_i)$ for i = 0,…,m, then A(x) = B(x).

Proof.

Recall that a polynomial of degree m has m roots.

$A(x) = Q(x)(x-\alpha) + A(\alpha)$, if $A(\alpha) = 0$, $A(x) = Q(x)(x-\alpha)$, and deg(Q) = m-1

Consider the polynomial C(x) = A(x) - B(x). Since $C(\alpha_i) = A(\alpha_i) - B(\alpha_i) = 0$, for m+1 points, C(x) = 0, and A(x) must equal B(x).

Lagrange Interpolation Formula

- **Find a polynomial of degree m given its value at (m+1) distinct points. Assume $A(\alpha_i) = y_i$**

$$A(x) = \sum_{i=0}^{m} \left(\prod_{j \neq i} \frac{(x - \alpha_j)}{(\alpha_i - \alpha_j)} \right) y_i$$

- **Observe that**

$$A(\alpha_k) = \sum_{i=0}^{m} \left(\prod_{j \neq i} \frac{(\alpha_k - \alpha_j)}{(\alpha_i - \alpha_j)} \right) y_i = y_k$$

Matrix Version of Polynomial Evaluation

- **Let $A(x) = a_3x^3 + a_2x^2 + a_1x + a_0$**

- **Evaluation at the points α, β, γ, δ is obtained from the following matrix-vector product**

$$\begin{bmatrix} A(\alpha) \\ A(\beta) \\ A(\gamma) \\ A(\delta) \end{bmatrix} = \begin{bmatrix} 1 & \alpha^1 & \alpha^2 & \alpha^3 \\ 1 & \beta^1 & \beta^2 & \beta^3 \\ 1 & \gamma^1 & \gamma^2 & \gamma^3 \\ 1 & \delta^1 & \delta^2 & \delta^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Matrix Interpretation of Interpolation

- **Let $A(x) = a_n x^n + \ldots + a_1 x + a_0$ be a polynomial of degree n. The problem of determining the (n+1) coefficients $a_n,\ldots,a_1,a_0$ from the (n+1) values $A(\alpha_0),\ldots,A(\alpha_n)$ is equivalent to solving the linear system**

$$\begin{bmatrix} A(\alpha_0) \\ A(\alpha_1) \\ \ldots \\ A(\alpha_n) \end{bmatrix} = \begin{bmatrix} 1 & \alpha_0^1 & \ldots & \alpha_0^n \\ 1 & \alpha_1^1 & \ldots & \alpha_1^n \\ \ldots & \ldots & \ldots & \ldots \\ 1 & \alpha_n^1 & \ldots & \alpha_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_3 \end{bmatrix}$$

Vandermonde Matrix

$$V(\alpha_0,\ldots,\alpha_n)=\begin{bmatrix} 1 & \alpha_0^1 & \cdots & \alpha_0^n \\ 1 & \alpha_1^1 & \cdots & \alpha_1^n \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \alpha_n^1 & \cdots & \alpha_n^n \end{bmatrix}$$

$$\det(V)=\prod_{i<j}(\alpha_j-\alpha_i)$$

$V(\alpha_0,\ldots,\alpha_n)$ is non-singular when $\alpha_0,\ldots,\alpha_n$ are distinct.

Polynomial Multiplication using Interpolation

- **Compute C(x) = A(x)B(x), where degree(A(x)) = m, and degree(B(x)) = n. Degree(C(x)) = m+n, and C(x) is uniquely determined by its value at m+n+1 distinct points.**

- **[Evaluation] Compute A(α_i) and B(α_i) for distinct α_i, i=0,…,m+n.**

- **[Pointwise Product] Compute C(α_i) = A(α_i)*B(α_i) for i=0,…,m+n.**

- **[Interpolation] Compute the coefficients of C(x) = $c_n x^{m+n}$ + … + $c_1 x + c_0$ from the points C(α_i) = A(α_i)*B(α_i) for i=0,…,m+n.**

Interpolation and Karatsuba's Algorithm

- **Let $A(x) = A_1x + A_0$, $B(x) = B_1x + B_0$, $C(x) = A(x)B(x) = C_2x^2 + C_1x + C_0$**
- **Then $A(10^k) = A$, $B(10^k) = B$, and $C = C(10^k) = A(10^k)B(10^k) = AB$**
- **Use interpolation based algorithm:**
  - **Evaluate $A(\alpha)$, $A(\beta)$, $A(\gamma)$ and $B(\alpha)$, $B(\beta)$, $B(\gamma)$ for $\alpha = 0$, $\beta = 1$, and $\gamma = \infty$.**
  - **Compute $C(\alpha) = A(\alpha)B(\alpha)$, $C(\beta) = A(\beta)B(\beta)$, $C(\gamma) = A(\gamma)B(\gamma)$**
  - **Interpolate the coefficients C_2, C_1, and C_0**
  - **Compute $C = C_2 10^{2k} + C_1 10^k + C_0$**

Matrix Equation for Karatsuba's Algorithm

- **Modified Vandermonde Matrix**

$$\begin{bmatrix} C(0) \\ C(1) \\ C(\infty) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

- **Interpolation**

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_0 B_0 \\ (A_0 + A_1)(B_0 + B_1) \\ A_1 B_1 \end{bmatrix}$$

Integer Multiplication Splitting the Inputs into 3 Parts

- **Instead of breaking up the inputs into 2 equal parts as is done for Karatsuba's algorithm, we can split the inputs into three equal parts.**

- **This algorithm is based on an interpolation based polynomial product of two quadratic polynomials.**

- **Let $A(x) = A_2x^2 + A_1x + A_0$, $B(x) = B_2x^2 + B_1x + B$, $C(x) = A(x)B(x) = C_4x^4 + C_3x^3 + C_2x^2 + C_1x + C_0$**

- **Thus there are 5 products. The divide and conquer part still takes time = O(n). Therefore the total computing time $T(n) = 5T(n/3) + O(n) = \Theta(n^{\log_3(5)})$, $\log_3(5) \approx 1.46$**

Asymptotically Fast Integer Multiplication

- **We can obtain a sequence of asymptotically faster multiplication algorithms by splitting the inputs into more and more pieces.**

- **If we split A and B into k equal parts, then the corresponding multiplication algorithm is obtained from an interpolation based polynomial multiplication algorithm of two degree (k-1) polynomials.**

- **Since the product polynomial is of degree 2(k-1), we need to evaluate at 2k-1 points. Thus there are (2k-1) products. The divide and conquer part still takes time = O(n). Therefore the total computing time $T(n) = (2k-1)T(n/k) + O(n) = \Theta(n^{\log_k(2k-1)})$.**

Asymptotically Fast Integer Multiplication

- **Using the previous construction we can find an algorithm to multiply two n digit integers in time $\Theta(n^{1+\varepsilon})$ for any positive ε.**
  - **$\log_k(2k-1) = \log_k(k(2-1/k)) = 1 + \log_k(2-1/k)$**
  - **$\log_k(2-1/k) \leq \log_k(2) = \ln(2)/\ln(k) \rightarrow 0$.**
- **Can we do better?**
- **The answer is yes. There is a faster algorithm, with computing time $\Theta(n\log(n)\log\log(n))$, based on the fast Fourier transform (FFT). This algorithm is also based on interpolation and the polynomial version of the CRT.**