Number theoretic algorithms

Representation: We choose an integral base $\beta > 1$.

Any positive integer A can be represented by a vector of length $n = \log A$, $(a_0, \dots, a_{n-1})$ such that

$$A = a_{n-1}\beta^{n-1} + a_{n-2}\beta^{n-2} + \cdots + a_1\beta + a_0$$

and $\forall\ 0 \le i \le n-1$; $0 \le a_i \le \beta - 1$.

Remark: What about sign?

Integer addition:

Algorithm 1.1 IntegerAddition

Input: $A = \sum_0^{n-1} a_i\beta^i$, $B = \sum_0^{n-1} b_i\beta^i$, carry-in $0 \le d_{\text{in}} \le 1$
Output: $C := \sum_0^{n-1} c_i\beta^i$ and $0 \le d \le 1$ such that $A + B + d_{\text{in}} = d\beta^n + C$

1: $d \leftarrow d_{\text{in}}$
2: **for** i **from** 0 **to** $n-1$ **do**
3: $\quad s \leftarrow a_i + b_i + d$
4: $\quad (d, c_i) \leftarrow (s \text{ div } \beta, s \bmod \beta)$
5: return C, d.

$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 10011 \\ 01001 \\ \hline 100 \end{array}$$

Remark: Note that if A and B are close to the maximum possible values that a computer can take then $A+B$ will be much larger than what the computer can handle.

Integer multiplication

A and B are again given in base β representation.

Algorithm 1.2 BasecaseMultiply

Input: $A = \sum_0^{m-1} a_i \beta^i,\ B = \sum_0^{n-1} b_j \beta^j$
Output: $C = AB := \sum_0^{m+n-1} c_k \beta^k$

Naive!

1: $C \leftarrow A \cdot b_0$
2: **for** j **from** 1 **to** $n-1$ **do**
3: $\quad C \leftarrow C + \beta^j (A \cdot b_j)$
4: return C.

In the worst case this takes $O(mn)$ operations.
This is roughly quadratic when $m \approx n$.

Question: Can we do better?

Yes! Karatsuba in 1960 gave a subquadratic time algorithm.

Algorithm 1.3 KaratsubaMultiply

Input: $A = \sum_0^{n-1} a_i \beta^i, B = \sum_0^{n-1} b_j \beta^j$
Output: $C = AB := \sum_0^{2n-1} c_k \beta^k$

→ **if** $n < n_0$ **then** return **BasecaseMultiply**(A, B)
$k \leftarrow \lceil n/2 \rceil$
$(A_0, B_0) := (A, B) \bmod \beta^k$, $(A_1, B_1) := (A, B) \operatorname{div} \beta^k$
$s_A \leftarrow \operatorname{sign}(A_0 - A_1)$, $s_B \leftarrow \operatorname{sign}(B_0 - B_1)$
$C_0 \leftarrow$ **KaratsubaMultiply**(A_0, B_0)
$C_1 \leftarrow$ **KaratsubaMultiply**(A_1, B_1)
$C_2 \leftarrow$ **KaratsubaMultiply**$(|A_0 - A_1|, |B_0 - B_1|)$
return $C := C_0 + (C_0 + C_1 - s_A s_B C_2)\beta^k + C_1 \beta^{2k}$.

Let $k = \lceil \frac{n}{2} \rceil$. $A_1 = A \operatorname{div} \beta^k$; $A_0 = A \bmod \beta^k$.

$$A = \underbrace{a_{n-1}\beta^{n-1} + a_{n-2}\beta^{n-2} + \cdots + a_{k+1}\beta^{k+1} + a_k\beta^k}_{A_1\beta^k} + \underbrace{\cdots + a_0}_{A_0}.$$

$$B = \underbrace{b_{n-1}\beta^{n-1} + b_{n-2}\beta^{n-2} + \cdots + b_{k+1}\beta^{k+1} + b_k\beta^k}_{B_1\beta^k} + \underbrace{\cdots + b_0}_{B_0}.$$

Roughly,

$$A = A_1\beta^k + A_0 \quad \text{and} \quad B = B_1\beta^k + B_0.$$

$$AB = (A_1\beta^k + A_0)(B_1\beta^k + B_0)$$

$$= A_1B_1\beta^{2k} + (A_0B_1 + A_1B_0)\beta^k + A_0B_0.$$

$T(n) : @T(\frac{n}{b}) + O(n)$

$\hookrightarrow O(n^{\log_b a}) = n^{\frac{\log a}{\log b}}$

$n^{\frac{\log 3}{\log 2}} \sim n^{\frac{\log 4}{\log 3}}$

There are 4 multiplications of size $\frac{n}{2}$.

Here $T(n) = 4T(\frac{n}{2}) + O(n)$. // $O(n^2)$ again.

Question: Can we reduce the no. of multiplications?

$C_0 = A_0B_0$

$C_1 = A_1B_1$

$C_2 = |A_0 - A_1| \cdot |B_0 - B_1|$.

$= (A_0 - A_1)\cdot(B_0 - B_1)$. // Says

$= A_0B_0 + A_1B_1 - (A_1B_0 + B_1A_0)$.

$\Rightarrow A_0B_1 + B_0A_1 = C_0 + C_1 - C_2$. Rather $C_0 + C_1 - S_AS_BC_2$

to account for signs.

Here we are using 3 multiplications ($\sim \frac{n}{2}$ sized) instead of 4.

// Toom-Cook r-way algorithm

$$\Rightarrow T(n) = 3T\left(\frac{n}{2}\right) + O(n).$$

$$\Rightarrow T(n) = O\left(n^{\log_2 3}\right) \approx O\left(n^{1+\epsilon}\right).$$

Instead of C_2 as above, we can also consider C_2 to be $(A_0 + A_1)\cdot(B_0 + B_1)$.

$$\Rightarrow A_0B_1 + A_1B_0 = C_2' - (C_1 + C_2).$$

// Using difference lets us store smaller numbers.

Schonhage-Strassen algorithm.

Runs in time $O(n \cdot \log n \cdot \log\log n)$

$O\left(n \cdot \log n \cdot 2^{O(\log^* n)}\right)$ Fürer 2007

$O\left(n \cdot \log n \cdot 2^{O(\log^* n)}\right)$ ↳ De, Kurur Saha Saptharishi 2008.

But we need more mathematical tools.

Discrete Fourier Transform:

Assumptions:

1. n is a power of 2
2. $\exists$ a multiplicative inverse for n. That is $\exists\, m$ s.t $n \cdot m = 1$. $\left(\frac{1}{n}\right)$

We are working over Rings $(R, +, \times, 0, 1)$ $a + b = 0$

↳ additive inverse

↳ associativity.

$\mathbb{Z}/n\mathbb{Z}_m = \mathbb{Z} \bmod m.$

3. $\exists$ a primitive n^{th} root of 1, say ω.

That is $\omega^n = 1$ and $\forall\, k \in [1, n-1]$, $\omega^k \neq 1$.

$\{e^{\frac{2\pi i \cdot k}{n}}\}_k$ ✓ — n^{th} roots of unity. $\omega = e^{\frac{2\pi i}{n}}$.

Obs: $\sum_{i=0}^{n-1} \omega^i = 0$. Further, $\sum_{i=0}^{n} \omega^{ij} = 0 \quad \forall\, j \neq 0$.

DFT matrix A:

$0 \le i, j \le n-1$, $\quad A_{ij} = \omega^{i \cdot j}$

$$(A^{-1})_{ij} = \frac{1}{n}\, \omega^{-i \cdot j}.$$

Definition:
Let $\vec{a} = (a_0, a_1, \ldots, a_{n-1})$. Then the discrete fourier Transform of $\vec{a}$ is given as follows:

$$F_n(\vec{a}) = A \cdot (\vec{a})^T \quad \leftarrow \text{transpose.}$$

$$\begin{matrix} b_0 \\ b_1 \\ \\ b_{n-1} \end{matrix} = \begin{bmatrix} \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 \cdots & \omega^{n-1} \\ \omega^0 & \omega^2 & \omega^4 & \cdots \\ \cdots & & \cdots & \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ \\ a_{n-1} \end{bmatrix}$$

} DFT of $\vec{a}$.

Inverse DFT of $\vec{a} = A^{-1} \cdot (\vec{a})^T$.

Note that $\text{DFT}(\text{Inv-DFT})(\vec{a}) = (\vec{a})^T$.

and $\text{Inv-DFT}(\text{DFT})(\vec{a}) = (\vec{a})^T$.

Qn: What is the complexity of computing $\text{DFT}(\vec{a})$?

Fast Fourier Transform: [Cooley-Tukey].

Let $(\vec{b})^T = A \cdot (\vec{a})^T$. // we want to obtain $(\vec{b})_i \ \forall i$.

$$b_i = \sum_{j=0}^{n-1} \underbrace{w^{ij}}_{(w^i)^j} \cdot a_j = \left(\sum_{j=0}^{\frac{n}{2}-1} w^{ij} a_j \right) + w^{\frac{n}{2} i} \left(\sum_{j=0}^{\frac{n}{2}-1} w^{ij} a_{j+\frac{n}{2}} \right)$$

$\sim$ $\frac{n}{2} + j' = j$

Note that $w^{\frac{n}{2}} = -1$.

$$\Rightarrow b_i = \sum_{j=0}^{\frac{n}{2}-1} w^{ij} \cdot \left(a_j + (-1)^i \cdot a_{j+\frac{n}{2}} \right)$$

$$\sum_{j=\frac{n}{2}}^{n-1} w^{ij} \cdot a_j = \sum_{j=0}^{\frac{n}{2}-1} w^{i(\frac{n}{2}+j)} \cdot a_{j+\frac{n}{2}}$$

i is even:

$$\sum_{j=0}^{\frac{n}{2}-1} w^{ij} \left(a_j + a_{j+\frac{n}{2}} \right) \quad (1)$$

i is odd:

$$\sum_{j=0}^{\frac{n}{2}-1} w^{ij} \left(a_j - a_{j+\frac{n}{2}} \right) \quad (2)$$

Let $C_k = a_k + a_{k+\frac{n}{2}}$

$D'_k = a_k - a_{k+\frac{n}{2}}$

$\forall \ 0 \le k \le \frac{n}{2} - 1$.

$i = 2p$

$i = 2p+1$.

Qn: Can we write (1) or (2) as Fourier Transforms of size $\frac{n}{2}$?

For that we need $\frac{n}{2}^{th}$ root of unity.

Obs: w^2 is a $\frac{n}{2}^{th}$ root of unity.

$$F_n(\vec{a}) = A(\vec{a})^T$$

Say $i = 2p$ for some p. $\quad i := 2p$

$$① = \sum_{j=0}^{\frac{n}{2}-1} \omega^{ij} C_j = \sum_{j=0}^{\frac{n}{2}-1} (\omega^2)^{p \cdot j} C_j = \left[F_{\frac{n}{2}}(\vec{C}) \right]_p$$

$\omega^{2p \cdot j} = (\omega^2)^{p \cdot j}$

Else say $i = 2p+1$ for some p.

$\omega^{(2p+1)j} = \omega^{2pj} \cdot \omega^j = (\omega^2)^{pj} \cdot \omega^j$

$$② = \sum_{j=0}^{\frac{n}{2}-1} \omega^{ij} D'_j = \sum_{j=0}^{\frac{n}{2}-1} (\omega^2)^{p \cdot j} \cdot (\omega^j \cdot D'_j) = \left[F_{\frac{n}{2}}(\vec{D}) \right]_p$$

where $D_k = D'_k \cdot \omega^k$

$= (a_k - a_{k+\frac{n}{2}}) \cdot \omega^k$

Putting it all together, we get that

$$b_i = \left[F_n(\vec{a}) \right]_i = \begin{cases} \left[F_{\frac{n}{2}}(\vec{C}) \right]_{\frac{i}{2}} & \text{if } i \text{ is even,} \\ \left[F_{\frac{n}{2}}(\vec{D}) \right]_{\frac{i-1}{2}} & \text{otherwise.} \end{cases}$$

$$\Rightarrow T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn$$

$$= O(n \cdot \log n).$$

Convolution of two vectors:

Let $\vec{a} = (a_0, \ldots, a_{n-1})$ and $\vec{b} = (b_0, \ldots, b_{n-1})$ then the convolution of a and b is as follows

$$\forall\, 0 \le i \le 2n-2, \quad (a * b)_i = \sum_{\substack{j+k=i \\ 0 \le j,k \le n-1}} a_j \, b_k .$$

Rewriting this, we get that

$$\forall\ i \le n-1,\quad (a*b)_i = \sum_{j=0}^{i} a_j \cdot b_{i-j}.$$

and

$$\forall\ i \ge n,\quad (a*b)_i = \sum_{j=i-(n-1)}^{n-1} a_j \cdot b_{i-j}.$$

Obs: Let $a' = (a_0, \dots, a_{n-1}, \overbrace{0, \dots, 0}^{n})$ and $b' = (b_0, \dots, b_{n-1}, \overbrace{0, \dots, 0}^{n})$.

Then $(a*b)_i = \sum_{j=0}^{i} a'_j \cdot b'_{i-j}$.

Theorem: $$F_{2n}^{-1}\left(F_{2n}(a') \odot F_{2n}(b')\right) = a*b.$$

$\left(F_{2n}(a') \odot F_{2n}(b')\right) = F_{2n}(a*b)$

$\odot$: pointwise multiplication.

$(a_1, \dots, a_{2n}) \odot (b_1, \dots, b_{2n}) = (a_1 b_1, \dots, a_{2n} b_{2n})$

↳ As a result $a*b$ can be computed in time $O(n \log n)$.

$A = a_{n-1}\beta^{n-1} + \dots + a_0$;

Application: Univariate polynomial multiplication.

Let $P_{\vec{a}}(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0$

and $P_{\vec{b}}(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_0$.

Then $P_{\vec{a}}(x) \cdot P_{\vec{b}}(x) = \sum_{i=0}^{2n-1} (a*b)_i\, x^i$.

$A \cdot B = (P_{\vec{a}} \cdot P_{\vec{b}})(\beta)$.

$A = P_{\vec{a}}(\beta)$

$B = P_{\vec{b}}(\beta)$

From above, we get that the product can be obtained in $O(n \log n)$ time.

Next class: DFT in mod-m rings; Chinese Remindering and Schönhage-Strassen algorithm.