

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

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

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

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 $0 \leq d_{\text{in}} \leq 1$

Output: $C := \sum_0^{n-1} c_i \beta^i$ and $0 \leq d \leq 1$ 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: $s \leftarrow a_i + b_i + d$
 - 4: $(d, c_i) \leftarrow (s \text{ div } \beta, s \text{ mod } \beta)$
 - 5: **return** C, d .
-

$$\begin{array}{r} 10011 \\ 01001 \\ \hline 10000 \end{array}$$

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: $C \leftarrow C + \beta^j (A \cdot b_j)$
 - 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)

$$k \leftarrow \lceil n/2 \rceil$$

$$(A_0, B_0) := (A, B) \text{ mod } \beta^k, (A_1, B_1) := (A, B) \text{ div } \beta^k$$

$$s_A \leftarrow \text{sign}(A_0 - A_1), s_B \leftarrow \text{sign}(B_0 - B_1)$$

$$C_0 \leftarrow \text{KaratsubaMultiply}(A_0, B_0)$$

$$C_1 \leftarrow \text{KaratsubaMultiply}(A_1, B_1)$$

$$C_2 \leftarrow \text{KaratsubaMultiply}(|A_0 - A_1|, |B_0 - B_1|)$$

$$\text{return } C := C_0 + (C_0 + C_1 - s_A s_B C_2) \beta^k + C_1 \beta^{2k}.$$

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

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

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

Roughly,

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

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

$$T(n) = @T\left(\frac{n}{2}\right) + O(n)$$

$$\hookrightarrow O(n \log n) \approx n^{\frac{\log 3}{\log 2}}$$

$$= \underbrace{A_1 B_1}_{\sim} \beta^{2k} + (\underbrace{A_0 B_1 + A_1 B_0}_{\sim}) \beta^k + \underbrace{A_0 B_0}_{\sim}.$$

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

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

Question: Can we reduce the no. of multiplications?

$$C_0 = A_0 B_0$$

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

$$C_1 = A_1 B_1$$

$$= (A_0 - A_1) \cdot (B_0 - B_1). // \text{Say}$$

$$= A_0 B_0 + A_1 B_1 - (A_1 B_0 + B_1 A_0).$$

$$\Rightarrow A_0 B_1 + B_0 A_1 = C_0 + C_1 - C_2. \text{ Rather } C_0 + C_1 - S_A S_B C_2$$

to account for signs.

($\sim \frac{n}{2}$ sized)

Here we are using 3 multiplications instead of 4.
// Toom-Cook
r-way algorithm

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

$$\Rightarrow T(n) = O(n^{\log_2 3}) \approx O(n^{1.55}).$$

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

$$\Rightarrow A_0 B_1 + A_1 B_0 = C'_2 - (C_1 + C_2). \quad \text{lets us store smaller numbers.}$$

Schonhage-Strassen algorithm. $O(n \cdot \log n \cdot 2^{\log^* n})$

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

But we need more mathematical tools. De, Kaur Saha Saptharishi 2008.

Discrete Fourier Transform:

Assumptions:

1. n is a power of 2

2. \exists a multiplicative inverse for n.

That is $\exists m$ s.t $n \cdot m = 1$.

We are working over Rings $a+b=0$

$(R, +, \times, 0, 1)$

→ additive inverse
→ associativity.

$\left(\frac{1}{n}\right)$

$\mathbb{Z}/n\mathbb{Z} = \mathbb{Z} \bmod n$

3. \exists a primitive n^{th} root of 1 , say ω .
 That is $\omega^n = 1$ and $\forall k \in [1, n-1], \omega^k \neq 1$.

$$\checkmark \left\{ e^{j \frac{2\pi i}{n} k} \right\}_{k=0}^{n-1}$$

n^{th} roots of unity.

$$\omega = e^{j \frac{2\pi i}{n}}$$

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 :

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

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

Definition:

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

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

$$\begin{aligned} b_0 &= \begin{bmatrix} \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \dots \\ \omega^0 & \omega^2 & \omega^4 & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \\ b_n &= \\ b_{n-1} &= \end{aligned}$$

DFT of
 \vec{a} .

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

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:

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

$$b_i = \sum_{j=0}^{n-1} w^{ij} \cdot a_j = \left(\sum_{j=0}^{n_2-1} w^{ij} a_j \right) + w^{\frac{n}{2}i} \left(\sum_{j=0}^{\frac{n}{2}-1} w^{ij} a_{j+\frac{n}{2}} \right)$$

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

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

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

i is even

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

i is odd.

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

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

②

$$\left. \begin{array}{l} C_k = a_k + a_{k+\frac{n}{2}} \\ D'_k = a_k - a_{k+\frac{n}{2}} \end{array} \right\} \quad \begin{array}{l} \forall 0 \leq k \leq \frac{n}{2}-1. \\ i = 2p \\ i = 2p+1. \end{array}$$

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 . i = 2p

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

$w^{2p,j} \rightarrow (w^2)^{p,j}$

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

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

where $\underline{D'_{ik}} = D'_{ik} \cdot \underline{w^k}$
 $= (a_{ik} - a_{k+\frac{n}{2}}) \cdot w^k$

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

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

Rewriting this, we get that

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

and

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

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

$$\begin{aligned} & \xrightarrow{(F_{2n}(a') \odot F_{2n}(b'))} \\ & = F_{2n}(a * b) \end{aligned}$$

Theorem:

$$\underline{F}_{2n}^{-1} (F_{2n}(a') \odot F_{2n}(b')) = a * b .$$

$$(a_1, \dots, a_n) \odot (b_1, \dots, b_n) \\ = (a_1 b_1, \dots, a_n b_n)$$

↑ pointwise multiplication.

As a result $a * b$ can be computed in time $O(n \log n)$.
 $A = a_n x^{n-1} + \dots + a_0 ;$

Application: Univariate polynomial multiplication.

$$\text{Let } P_A(x) = a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_0$$

$$\underline{A} = \underline{P}_A(\underline{\beta})$$

$$\text{and } P_B(x) = b_{n-1} x^{n-1} + b_{n-2} x^{n-2} + \dots + b_0 .$$

$$\underline{B} = \underline{P}_B(\underline{\beta})$$

$$\text{Then } P_A(x) \cdot P_B(x) = \sum_{i=0}^{2n-1} (a * b)_i x^i .$$

$$A \cdot B = (P_A \cdot P_B)(\beta) .$$

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

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

$$\begin{array}{c} \text{---} \\ \mathbb{Z} \bmod z^m + 1 \\ \text{---} \\ \downarrow \\ \mathbb{Z} \bmod m \end{array}$$