Special points — n^{th} complex root of 1.

FAST FOURIER TRANSFORM

$(x - x_i)^n = 1$ $\qquad i = \sqrt{-1}$

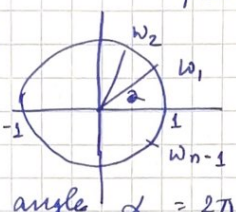
1 st root $\{1\}$ $\qquad 1^1 = 1$

2nd root $\{-1, 1\}$ $\qquad 1^2 = -1^2 = 1$

4th roots $\{-1, 1, i, -i\}$ $\qquad 1^2 = -1^2 = i^2 = (-i)^2 = 1$

nth roots of 1 = equally spaced points in complex plane in disc around 0

$z = x + iy$



nth the roots of unity = $\{\omega_0, \omega_1 \text{---} \omega_n\}$

$\omega_{n,j} \quad \forall j = (0, n-1)$

General form angle $\alpha = \frac{2\pi}{n} j$

$$\omega_{n,j} = \left\{ \underbrace{\cos\left(\frac{2\pi}{n} j\right)}_{x} + i \underbrace{\sin\left(\frac{2\pi}{n} j\right)}_{y} \right\} = e^{i\left(\frac{2\pi}{n} j\right)}$$

Properties ⓪ $\omega_{n,j}^n = 1$ ① $\omega_{n,0} = 1$ ② $\omega_{n,j}^2 = \omega_{n/2,j}$

③ $\omega_{n,j+n/2} = -\omega_{n,j}$ ④ $(\omega_{n,1})^K = (\omega_{n,K})$

nth roots of $1 = \{\omega_{n,1}^0, \omega_{n,1}, \omega_{n,2}^2, \omega_{n,3}^3 \ldots \omega_{n,n-1}^{n-1}\}$

Q- How to do multipoint evaluations using nth root of unity?

Given polynomial $P(x) = a_0 + a_1 x + a_2 x^2 \text{---}$ (n coefficients)
the task is to compute $P(\omega_{n,0})$ $P(\omega_{n,1}) \ldots$

assume $n = 2^K$ (pad with $0 \times^{K+1}$ if needed)

17.11.2023 **Idea** – Split the coefficients into 2 polynomials, with even & odd coefficients.

$$P^{even}(x) = a_0 + a_2 x + a_4 x^2 \text{---} a_n x^{n/2-1} = \sum_{K=0}^{n/2-1} a_{2K} x^K$$

$$P^{odd}(x) = a_1 + a_3 x + a_5 x^2 \text{---} a_{n-1} x^{n/2-1} = \sum_{K=0}^{n/2-1} a_{2K+1} x^K$$

$\therefore P(x)$ can be rewritten as

$$P(x) = P^{even}(x^2) + x P^{odd}(x^2)$$

$$P(\omega_{n,j}) = P^{even}(\omega_{n,j}^2) + \omega_{n,j} \cdot P^{odd}(\omega_{n,j}^2)$$

$$\Rightarrow P(\omega_{n,j}) = P^{even}(\underbrace{\omega_{n/2,j}}_{\text{using Property ②}}) + \omega_{n,j} P^{odd}(\omega_{n/2,j}) \quad \text{— (A)}$$

Similarly

$$P(-\omega_{n,j}) = P(\omega_{n,j+n/2}) = P^{even}(\omega_{n/2,j}) + \omega_{n,j} P^{odd}(\omega_{n/2,j}) \quad \text{— (B)}$$

Property ③

with $j \le \frac{n}{2}$, we write (A) & (B) as first half & second half

Therefore we reduced the polynomial to compute $\frac{n}{2}$ roots of 1. and to evaluate P on $\omega_{n,j}$ for $(j = 0, 1, \ldots, n-1)$ we evaluate p_{odd} and p_{even} on $\omega_{n/2,j}$ $(j = 0, 1, \ldots, n/2-1)$

The problem is now become Recursive. We will see this in algorithm

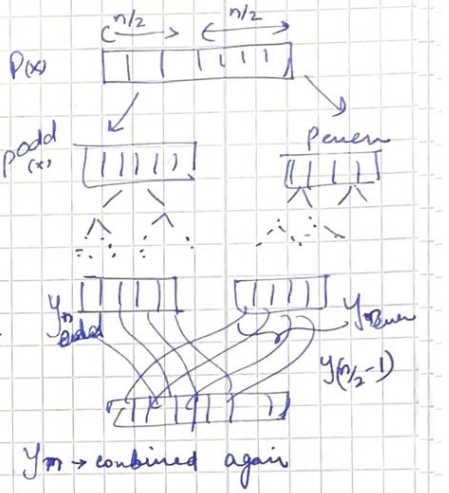
FFT algorithm $([a_0, a_1, \ldots, a_{n-1}])$:

if $n = 1$ return $[a_0]$

Recursive $\begin{cases} [y_0^{even}, \ldots, y_{n/2-1}^{even}] = FFT[a_0, a_2, \ldots, a_{n-2}] \\ [y_0^{odd}, \ldots, y_{n/2-1}^{odd}] = FFT[a_1, a_3, \ldots, a_{n-1}] \end{cases}$ ← using expressions A ← and B

for $(j = 0, \ldots, n/2 - 1)$

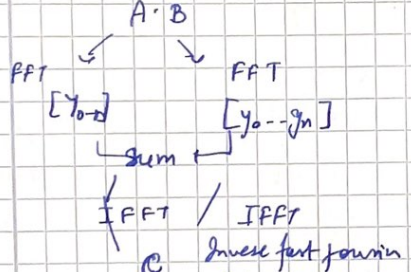
Compute $\begin{cases} y_j = y_j^{even} + \omega_{n,j}\, y_j^{odd} \\ y_{j+n/2} = y_j^{even} - \omega_{n,j}\, y_j^{odd} \end{cases}$



Running Time $t(n) = 2t\left(\frac{n}{2}\right) + c \cdot n$ ← divide & conquer technique.

$\Rightarrow t(n) \in \Theta(n \log n)$

Multiplication of 2 polynomials ⇒



Remarks

① How do we work with Complex numbers ? — gives the issue of precision $n \log n \cdot (\log \log n)$ due to precisions

② Method also works with modulo P instead of complex numbers.

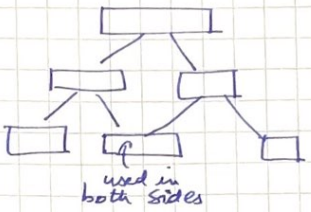
$P = 7$, Z_7 modulo $1^2 = 1 \pmod 7$ $6^2 = 1 \pmod 7$ so we can have more roots

③ Reverse is essentially the same Inverse FFT

④ Many applications such as image processing etc

⑤ Related to Classic Fourier Tranform.

DYNAMIC PROGRAMMING



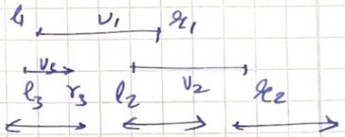
directed acyclic graph.

Ex.0

Given n jobs, job i has start time l_i and end time r_i, value $= v_i$

Goal → find set of non-overlapping jobs with maximum total value.

i.e. pick jobs to execute that maximizes value and do not overlap.



[• if $v_i = 1$, problem is trivial, simple greedy algorithm works]

Solution:

→ Denote jobs $J_1 \dots J_n$ by right endpoint.

→ Consider right most J_n

J_n — take it — add v_n, optimize $(J_1 \dots J_i)$ where $i < J_n\, l$ (left)

— not take it — compute Optimum on remaining.

$OPT(J_1 \dots J_n)$

$$\max \{ OPT(J_1 \dots J_{n-1}),\ v_n + OPT(J_1 \dots J_i) \}$$

J_i is last such that $r_i < l_n$

↗ Take maximum of both cases to maximize value.

Base case $OPT(\phi) = 0$, $OPT(J_n) = v_1$

Running time $t(n) \leq 2t(n-1) \Rightarrow t(n) \in O(2^n)$ [very bad]

Further Analysis

We need to compute $OPT(J_1 \dots J_K)$. There are 2 ways to do it.

top down — one by one from top

bottom up — draw dependence diagrams

$OPT(J_1 \dots J_n)$

↓

$OPT(J_1 \dots J_{n-1})$

⋮

$OPT(J_K) = v_K$

for $K = 1 \dots n$

compute $OPT(J_1 \dots J_K) = \max \{ OPT(J_1 \dots J_{K-1}),\ v_K + OPT(J_1 \dots J_i) \}$

$\exists i$ s.t. $r_i < l_K$

$s_i \leq l_k$ ← can be found using binary search algorithm
$\sim O(\log n)$

∴ Running time

Sorting $O(n \log n)$

for loop → $n \times O(\log n)$

$\therefore T(n) \in O(n \log n)$ ← much better than 2^k

at each step we save the OPT( ) value in a table to be used in other parts of program. Hence it is called dynamic programming.