

Discussion 2

Sunday, September 9, 2018

11:39 AM

Topics: FFT, Graphs

FFT

- polynomial review:

2 ways to represent degree n polynomial with $O(n)$ parameters:

coefficients $\xrightleftharpoons[\text{interpolation}]{\text{evaluation}}$ points \leftarrow takes linear time to multiply two polynomials (nice!)

Idea: given coefficients of two polynomials, I will

① translate to points by evaluation

$\boxed{\theta(n^2)}$?? too slow
 $\theta(n)$

② multiply the values

③ translate back to coefficients by interpolation

Can we do better?

Try Divide-and-Conquer $\frac{n}{2}$

- FFT: a fast way to evaluate polynomials

Idea: We're free to choose which points we're evaluating the polynomial at, so we're going to choose those points wisely so that we can use Divide-and-Conquer.

We can evaluate degree $\leq n-1$ polynomial $A(x)$ at $\pm x_1, \dots, \pm x_{\frac{n}{2}}$.

by ① split A into even power terms $A_e(x)$ and odd power terms $A_o(x)$

② Evaluate $A_e(x_i^2)$ and $A_o(x_i^2)$

③ For each i ,

$$A(x_i) = A_e(x_i^2) + x_i A_o(x_i^2)$$

$$A(-x_i) = A_e(x_i^2) - x_i A_o(x_i^2)$$

$$\text{Runtime: } T(n) = 2T(n/2) + O(n)$$

$$\Rightarrow T(n) = O(n \log n)$$

However, this assumes that at the next level,

$x_1^2, \dots, x_{\frac{n}{2}}^2$ have to be plus-minus pairs...

It's okay! We can use complex numbers!

Notice for example we want to choose 8 points so that they're roots of $z^8 = 1$.

(They're plus-minus pairs since $z^8 = (-z)^8$)

Then, at the next level, our points would be roots of $z^4 = 1$,

so they'll still be plus-minus pairs. (Yayyy!!)

function FFT(A, ω):

A: coefficients representation of a polynomial A(x)

ω: nth root of unity

if ω=1: return A(1)

split A into A_e, A_o

V_e = FFT(A_e, ω²) evaluate A_e and store values in V_e

A_e = FFT(A_o, ω²) evaluate A_o and store values in V_o

for j=0 to n-1:

 compute A(ω^j) = V_e[j] + ω^jV_o[j]

return A(ω⁰), ..., A(ωⁿ⁻¹)

• FFT in matrix:

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \dots & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

values "x⁰" "x¹" "x²" ... "xⁿ⁻¹" coefficients

• values = FFT(coefficients, ω)

Don't forget our original motivation is multiplying to polynomials,

so we still need to convert the points back to coefficients

$$\text{coefficients} = \frac{1}{n} \text{FFT}(\text{values}, \omega^{-1})$$

Graphs

- procedure DFS (G):

for all $v \in V$:

visited (v) = False

for all $v \in V$:

if not visited (v):

explore (G, v)

- procedure explore (G, v):

visited (v) = true

previsit (v) depends on your task

→ e.g. find connected components ;
pre/post orderings

for each $(v, u) \in E$:

if not visited (u): explore (G, u)

postvisit (v) depends on your task

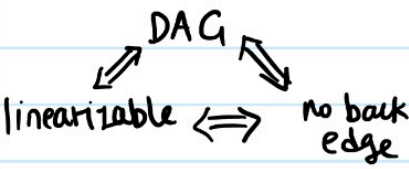
- runtime: $O(|E| + |V|)$

- using pre/post ordering to determine edge type:

for edge (u, v) :

[[]]	Tree / Forward	u on stack the entire time when v on stack
[[]]	Back	v on stack the entire time when u on stack
[] []	Cross	v got explored first

DAG

- 

linearizable \iff no back edge
- linearize a DAG: using decreasing post order

SCC

motivation: why if a graph isn't a DAG?

- all directed graph is a DAG of its SCC