28 Feb 2018 The Fast Fourier Transform

Announcements:

- Prelims are graded, make-ups are not. Grades will be announced on Piazza after Thursday's make-up test.
- Fri, March 2. Guest lecture by Jon Kleinberg (Closest Pair of Points, Chapter 5.??)

Finishing $O(n^{1.58})$ integer multiplication...

Last lecture presented multiplication of degree $n-1$ polynomials using $O(n^{\log_2 3}) = O(n^{1.58})$ arithmetic operations on coefficients.

Standing assumption in this lecture and throughout theory of algorithms: when working on a problem with input size n bits, arithmetic operations $(+, *, <)$ on integers of size $O(\log n)$ bits take $O(1)$ time.

Multiply the binary number $a_{n-1}\ a_{n-2}\ a_{n-3}\ \dots\ a_0$ by $b_{n-1}\ b_{n-2}\ b_{n-3}\ \dots\ b_0$

Method: Define $A(x) = a_{n-1}x^{n-1} + \dots + a_0$, $B(x) = b_{n-1}x^{n-1} + \dots + b_0$

Compute $C(x) = A(x) \cdot B(x) = c_{2n-2}x^{2n-2} + \dots + c_0$ using $O(n^{1.58})$ arithmetic operations.

Coefficients of $C(x)$...

$$c_k = \sum_{i=0}^{k} a_i b_{k-i} \quad \text{"convolution"}$$

$0 \le c_k \le k$ $\quad c_k$ has $\le \log n$ bits.

The same applies to the intermediate polynomials used in the middle of the divide & conquer: all coefficients of all such polynomials are (signed) integers of absolute value $\le n$, hence $O(\log n)$ bits suffice. Every arithmetic op takes $O(1)$.

The input integers were $A(2)$, $B(2)$.
The output should be $C(2)$.

Last step of algorithm: compute $C(z) = \sum_{k=0}^{2n-2} c_k 2^k$.

Ordinary mult also.

n wide

n high

```
 1101
 0000
 1101
 1101
```

2n-2 high

Each summand only $O(\log n)$ bits wide.

```
 1101      c_0
 0101      c_1
  .
  .
  .
 1111      c_{2n-2}
```

total width $O(n)$

Summing up each column involves adding only $O(\log n)$ nonzero digits. So $O(n \log n)$ to sum up the whole table using ordinary add-and-carry.

Overall running time: $O(n^{1.58})$.

Explaining some notation: when doing $(a_1 x + a_0)(b_1 x + b_0)$

$$c_0 = a_0 b_0 \qquad c_1 = (a_0 + a_1)(b_0 + b_1) \qquad c_\infty = a_1 b_1$$

Evaluate	$a_1 x + a_0$	at	$x = 0$:	a_0
	$b_1 x + b_0$	at	$x = 0$:	b_0
	$(a_1 x + a_0)(b_1 x + b_0)$	at	$x = 0$:	c_0
Eval	$(a_1 x + a_0)(b_1 x + b_0)$	at	$x = 1$:	c_1
" "	" "	at	$x = \infty$:	$\approx c_\infty \cdot x^2 +$ lower order terms

Other choices of eval pts lead to different $O(n^{1.58})$ algs.

Eg.

$$c_0 = a_0 b_0 \qquad c_1 = (a_0 + a_0)(b_1 + b_0) \qquad c_{-1} = (-a_1 + a_0)(-b_1 + b_0)$$

$$(a_1 x + a_0)(b_1 x + b_0) = \tfrac{1}{2}(c_1 + c_{-1} - 2c_0) x^2 + \tfrac{1}{2}(c_1 - c_{-1}) x + c_0$$

An algorithm that multiplies degree $n-1$ polynomials using $O(n \log n)$ arithmetic ops... FFT!

Super useful because poly. mult. encodes convolution and convolution is everywhere!

Ex. X and Y are random vars, indep, taking $\{0, \dots, n-1\}$ values. Compute for all $0 \le k \le 2n-2$, $\Pr(X+Y=k)$...

$$\Pr(X+Y=k) = \sum_{i=0}^{k} \Pr(X=i) \Pr(Y=k-i)$$

Define $A(x) = \sum_{i=0}^{n-1} \Pr(X=i) x^i$ $\quad B(x) = \sum_{i=0}^{n-1} \Pr(Y=i) x^i$

then solving the probability problem above is just computing $A(x) \cdot B(x)$.

Ex 2. Given a sequence of numbers $a_0, \dots, a_{n-1}$

Compute the sequence of averages

$$c_i := \frac{1}{3}(a_{i-1} + a_i + a_{i+1}).$$

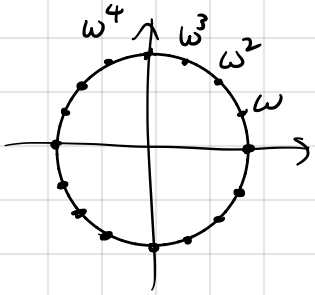
This is equiv't to multiply

$$A(x) := \sum a_i x^i \qquad B(x) = \frac{1}{3} + \frac{1}{3}x + \frac{1}{3}x^2.$$

A fast alg. for polynomial mult. is the following.

(1) Let $\omega = e^{2\pi i/2n}$

(2) Evaluate $A(\omega^k)$ for $k = 0, \dots, 2n-1$
Do the same for $B(\omega^k)$.



(3) Compute $C(\omega^k) := A(\omega^k) \cdot B(\omega^k)$

(4) Interpolate to find the coeffs of the unique degree $2n-1$ polynomial C whose values at $1, \omega, \omega^2, \dots, \omega^{2n-1}$ match the computed values.

FFT does both these steps in $O(n \log n)$

FFT problem. Given coeffs of polynomial $A(x)$, $\left(\omega = e^{\frac{2\pi i}{2n}}\right)$

evaluate it at $x = \omega^k$ $\forall\ k = 0, \ldots, 2n-1$.

Assume n is a power of 2.

$$A = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$$

$$= A_{even}(x^2) + x \cdot A_{odd}(x^2).$$

FFT alg. **Step 1.** Form polynomials A_{even}, A_{odd} $O(n)$

Step 2. Recursively evaluate $A_{even}(\omega^{2j}), A_{odd}(\omega^{2j})$ $\forall\ j = 0, \ldots, n-1$. Recursive $2 \cdot T(\frac{n}{2})$.

Step 3. Compute $A(\omega^k) = A_{even}(\omega^{2k}) + \omega^k \cdot A_{odd}(\omega^{2k})$ for each $k = 0, \ldots, 2n-1$. $O(n)$

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

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