Faster Multiplication, Powering of Polynomials

Lecture 5

Given 2 polys P and Q of size 2n

- Normal multiplication would take $(2n)^2$ cross multiplies, plus some amount of data manipulation (O(n^2) for dense multiplication)
- Karatsuba is O(n^{1.58}).
- Here's how.

"Karatsuba" multiplication

- Assume P and Q are of equal size and of odd degree.
- How bad a lie is this?
  - If they are not, pad them with 0 coefficients.
  - Divide them "in half"

"Karatsuba" multiplication

- $P = a_{2n-1}x^{2n-1} + \ldots + a_0 =$

$a_{2n-1}x^{2n-1} + \ldots + a_{n+1}x^{n+1} + a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0 =$

$(a_{2n-1}x^{n-1} + \ldots + a_{n+1}x^1 + a_n)x^n + a_{n-1}x^{n-1} + \ldots + a_0 =$

$= A*x^n + B$ where size(A) = size(B) = n

note that size(P) = 2n.

Same for $Q = Cx^n + D$

Compare:

- $P*Q = (A*C)(x^n)^2 + (A*D+B*C)(x^n) + (B*D)$
- There are 4 multiplies of size n polynomials, plus a size 2n polynomial addition: $O(4*(n^2))$. What we expected.
- The new idea: Instead, compute $R=(A+B)*(C+D) = A*C+A*D+B*C+B*D$
- $S=A*C$
- $T=B*D$
- Note that $U= R-S-T$ is $A*D+B*C$
- $P*Q = S*(x^n)^2 + U*x^n + T$.
- Thus there are 3 multiplies of size n polynomials plus a few additions of polys of size 2n.

Compare: Cost of multiplications, recursively

- M(2n) = 3*M(n) +cheaper adds O(n)....
- $M(k) = 3^k * M(1)$ where $k = \log_2(n)$

- $3^{\log_2(n)} = 2^{\log_2 3 \log_2 n} = n^{\log_2 3}$
- $= n^{1.58}$
- Does it work?
- Generally if P and Q are about the same size and dense, and "large but not too large" this is good.
- Larger sizes: better methods.
- How large are the coefficients? smallXbig or medXmed?

A peculiar way to multiply polynomials f(x) by g(x) (Sketch)

- Evaluate f(0), g(0), and trivially h(0)= f(0)*g(0) {one multiply}. This gives us the constant coefficient of h. What about the other coefficients?
- Repeat on 1,2,…n, that is, evaluate f(n), g(n), and store the value for h(n) which is f(n)*g(n)
- Interpolate: Find polynomial h(x) that assumes the value h(0) at 0, h(1) at 1, …. h(n) at n.

A peculiar way to multiply polynomials f(x) by g(x)

- The product h of f and g will have degree n=deg(h)=deg(f)*deg(g).
- Evaluate f(0), g(0), and trivially h(0)= f(0)*g(0)
- Repeat … until we have enough values (n+1)..
- Evaluate f(n), g(n), and h(n)= f(n)*g(n)
- Interpolate: What polynomial h(x) assumes the value h(0) at 0, h(1) at 1, …. h(n) at n?
- Questions:
  - Does this work? Yes
  - How do we choose n? See above

A peculiar way to multiply polynomials f(x) by g(x)

How much does this cost?

- 2*n evaluations of size(n) by Horner's rule is n^2.
- Interpolation by "Lagrange" or "Newton Divided Difference" is n^2

- $O(n^2)$ so it is a loser compared to Karatsuba.
- BUT....

What if we can do evaluation faster?

- Our choice of points 0,1,2,... n was arbitrary.
- What if we choose "better" points.
- e.g. evaluating a poly at 0 is almost free.
- evaluating a poly p(c) and p(-c) can be done "faster".. e.g. take even and odd coefficients separately.
- $P_{odd} := a_{2n+1}x^n + a_{2n-1}x^{n-1} + \ldots + a_1$
- $P_{even} := a_{2n}x^n + \ldots + a_0$
- $P(x) = P_{odd}(x^2)*x + P_{even}(x^2) = p1*x + p2$
- $P(-x) = -p1*x + p2$.
- So P(c) can be computed by evaluating
  - $p1 = P_{even}(c^2)$ and $p2 = P_{odd}(c^2)$
  - and returning $c*p1 + p2$,
- Finally, P(-c) is just $-c*p1 + p2$, nearly free.

What other points are better?

- Consider
  - some number r
    - real, complex,
    - finite field
    - other structure?,
  - some polynomial P, and
  - an integer n.
- Evaluate P at all 2^n points corresponding to powers of some 2^n root of r, namely 1, w, w^2, w^3, … $w^{2^n}=r$.
- Evaluating $P(-w^k)$ and $P(w^k)$ can be done "faster"..
  - take even and odd coefficients separately,
  - by the same trick as before, with $c^2 = w^{2k}$

Essentially, the numerical FFT is

- evaluating at "complex roots of unity."
- This reduces cost from n^2 to $O(n\log n)$;
- also same complexity for interpolation.
- Later we will return to this.

Computing powers of a polynomial

- Obvious methods
- $P^n = P*(P*(P.....*P))$

  repeated multiplication

$P^{2n} = (...(P^2)^2...)^2$ repeated squaring n times

(easily generalized to any power by occasional multiplications by P)

Under some models of cost for polynomial multiplication, repeated squaring is slower.

The essential observation is that if you have pre-computed say, p, …, p^5, and need p^{10}, you could

compute $(p^5)*(p^5)$ or

$p*(p* ….p^5)$.

The latter may be faster in practice. (Explain..)

Computing powers of a polynomial, other methods

Binomial expansion: let p=(a+<rest of terms>)

$(a+b)^n = a^n + n*a^{n-1}b + \ldots + b^n$

Multinomial expansion $(a+b+c+..)^n$

FFT: conceptually, evaluate p(), of degree d, at n*d+1 points (or more) and interpolate to get p^n (later)..