Polynomial representations

Lecture 4

Obvious representations of a polynomial in one variable x of degree d… DENSE

- Array of d+1 coefficients: $[a_0,\ldots,a_d]$ represents
  $a_0+a_1x+\ldots a_dx^d$
- Some other ordered structure of same coefficients. E.g. list.
- Also stored (in some fashion): "x" and d:
  - ["x",d, $[a_0,\ldots,a_d]$] -- d is just 1+length of array
- Why ordered? Consider division.. (this is relaxed later…)
- Assumption is that most of the a_i are non-zero.

Representations of a polynomial in 2 variables {x,y} of degree dx,dy… DENSE RECURSIVE

- We store: "x" and dx:
  - ["x",dx, [a_0,…,a_{dx}]]
  - But now each a_i is ["y",dy, [b_0,…,b_{dy}]]
- Assumption is (again) that b_{dy} and most of the b_i are non-zero.
- Also implicit is that there is some order $\phi(x)>\phi(y)$

Generalize to any number of variables {x,y,z}

- We could store this in some huge cube-like n-dimensional array where all degrees are the same maximum, but this seems wasteful: not all the dy, dz need be the same.
- For this to be a reasonable form, we hope most of the b_i are non-zero.
- Also required … $\phi(x)>\phi(y)>\phi(z)$…

Generalize to any coefficient?

- Array of coefficients might be an array of 32-bit numbers, or floats.
- Or an array of pointers to bignums.
- Or an array of pointers to polynomials in other variables. (= recursive !)
- Also required … $\phi(x)>\phi(y)>\phi(z)$; membership in the domain of coefficients must be easily decided, to terminate the recursion.

Aside: 'fat' vs. 'thin' objects

- Somewhere we record x,y,z and $\phi(x)>\phi(y)>\phi(z)$;
- Should we do this one place in the whole system, maybe even just numbering variables 0,1,2,3,…, and have relatively "thin" objects or
- Should we (redundantly) store x,y,z ordering etc, in each object, and perhaps within objects as well?
- A "fat" object might look something like (in lisp)

(poly (x y) (x 5 (y 2 3 4 0) (y 1 1 0)(y 0 2)(y 0 0) (y 1 1) (y 0 6))

Polynomial of degree 5 in x, $x^5(3y^2+4y)+x^4(y)+2x^3\ldots$

deg

coef of x^5

Aside: 'fat' vs. 'thin' objects

The fat version…

(poly (x y) (x 5 (y 2 3 4 0) (y 1 1 0)(y 0 2)(y 0 0) (y 1 1) (y 0 6))

Polynomial of degree 5 in x, $x^5(3y^2+4y)+$ …

An equivalent thin object might look like this, where it is understood globally that all polys have x as main variable, and y as next var; degree is always length of list –1:

((3 4 1) (1 0)(2)() (1) (6)) ;; used in Mathlab '68

$\hookrightarrow$ length 6 ⇒ degree 5

Operating on Dense polynomials

- Polynomial arithmetic on these guys is not too hard: For example, R=P+Q
  - Simultaneously iterate through all elements in corresponding places in P and Q
  - Add corresponding coefficients b_i
  - Produce new data structure R with answer
  - Or modify one of the input polynomials.
- P and Q may have different dx, dy, or variables, so size(R) <= size(P)+size(Q).

Operating on Dense polynomials

- R=P times Q
  - The obvious way: a double loop
  - For each monomial $a*x^n y^m$ in P and for each monomial in Q $b*x^r y^s$ produce a product $ab*x^{n+r}y^{n+s}$
  - Add each product into an (initially empty) new data structure R.
- degree(R) = degree(P)+degree(Q) (well, for one variable, anyway).
- Cost for multiplication? N=size(P),M=size(Q), O(NM) time, O(N+M) space.
- There are asymptotically faster ways than this. No one claims faster ways if N,M<30.

A Lisp program for dense polynomial multiplication

```
(defun make-poly (deg val)
  ;; make a polynomial of degree deg all of whose coefficients
  ;; are val.
  (make-array (1+ deg) :initial-element val))

(defun degree(x)(1- (length x)))

(defun times-poly(r s)
  (let ((ans(make-poly (+ (degree r)(degree s)) 0)))
    (dotimes (i (length r) ans)
      (dotimes (j (length s))
             (incf (aref ans (+ i j))
                   (* (aref r i)(aref s j)))))))
;; to make this more general, change "*" to recursively call this
```

Pro / Con for Dense polynomials

- Con: Most polynomials, especially with multiple variables, are sparse. $3x^{40}$+ … 0…+$5x^4$+3, so it tends to waste space.
- Con: Using a dense representation [3,0,0,0,0,…..] is slower than necessary for simple tasks.
- Pro: "Asymptotically fast" algorithms usually defined for dense formats
- Pro: Conversion between forms is O(D) where D is the size of the dense representation.

Sparse Polynomial Representation

- Represent only the non-zero terms.
- Favorable when algorithms depend more on the number of nonzero terms rather than the degree.
- Practically speaking, most common situation in "system" contexts where there are many variables.

Sparse Polynomials: expanded form

- Collection of monomials
  - For example, $34x^2y^3z + 500xyz^2$ has 2 monomial terms
  - Conceptually, each monomial is a pair:

  {coeff., exponent-vector}

  Multiplication requires collection. How to collect?
    - A list ordered by exponents (which order?)
    - A tree (faster insertion in random place: do we need this??)
    - A hash-table (faster than tree?) but unordered.

Sparse Polynomials: Ordered or not…

- If you multiply 2 polynomials with s, t terms, resp. then there are at most s*t resulting terms.
- The number of coefficient mults. is s*t.
- The cost to insert them into a tree or to sort them is O(s¢t log(s¢t)), so theoretically this n log n term dominates. Asymptotically fast methods don't work fast if s,t <<degrees.
- Insertion into a hash table is O(s¢t) probably.
- The hashtable downside: sometimes you want the result ordered (e.g. for division, GB)

Sparse Polynomials: recursive form

- Polynomials recursively sparse,
- A sparse polynomial in x with sparse polynomial coefficients:
  - $(3*x^{100}+x+1)z^{50} +4z^{10} +(5*y^{9}+4)z^{5}+5z+1$
- Ordering of variables important
  - Internally, given any 2 variables one is more "main variable"
- Representing constants or (especially zero) requires some thought. If you compute ...$0*x^{10}$ convert to 0.
- Programming issue: Is zero a polynomial with no terms, e.g. an empty hash table, or a hash table with a term $0*x^{0}*y^{0}$ ...

Some other representations

- Factored
- Straight Line
- Kronecker
- Modular

Factored form

- Choose your favorite other form, sparse or dense.
- Allow an outer layer … product or power of those other forms $p_1 £\ p_2^3$
- Multiplication is trivial. E.g mult by p_1: $p_1^2\ £\ p_2^3$
- Addition is not.
- Now common. Invented by SC Johnson for Altran (1970).
- Rational functions representation is simple generalization; allow exponents to be negative.

Straight-line program

- Sequence of program steps:
  - T1:=read(x)
  - T2:=3*T1+4
  - T3:=T2*T2
  - Write(T3)
- Evaluation can be easy, at least if the program is not just wasting time. Potentially compact.
- Many operations are trivial. E.g. to square a result, add a line to the above program, T4:=T3*T3.
- Testing for degree, or for zero is not trivial, may be done heuristically.

Examples: Which is better?

$$\left(y^3+x^5+x+1\right)^2$$

$$y^6+2\,x^5\,y^3+2\,x\,y^3+2\,y^3+x^{10}+2\,x^6+2\,x^5+x^2+2\,x+1$$

$$x^{10}+2\,x^6+\left(2\,y^3+2\right)\,x^5+x^2+\left(2\,y^3+2\right)\,x+y^6+2\,y^3+1$$

$$y^6+\left(2\,x^5+2\,x+2\right)\,y^3+x^{10}+2\,x^6+2\,x^5+x^2+2\,x+1$$

What is the coefficient of x^5y^3?
What is the coefficient of x^5?
What is the degree in x?
What is p(x=2,y=3)?

Which is better? (continued)

- Finding GCD with another polynomial
- Division with respect to x, or to y, or “sparse division”
- Storage
- Addition
- Multiplication
- Derivative (with respect to main var, other var).
- For display (for human consumption) we can convert to any other form, (which was done in the previous slide).

Recall: The Usual Operations

- Integer and Rational:
  - Ring and Field operations +- * exact quotient, remainder
- GCD, factoring of integers
- Approximation via rootfinding
- Polynomial operations
  - Ring operations, Field operations, GCD, factor
  - Truncated power series
  - Solution of polynomial systems
  - Interpolation: e.g. find p(x) such that p(0)=a, p(1)=b, p(2)=c Matrix operations (add determinant, resultant, eigenvalues, etc.)

Cute hack (first invented by Kronecker?) Many variables to one.

- Let x= t, y=t^{100} and z=t^{10000}.
- Then x+y+z is represented by t+t^{100}+t^{10000}
- How far can we run with this? Add, multiply (at least, as long as we don't overlap the exponent range).
- Alternative way of looking at this is 45*xyz is encoded as
  - [{x,y,z}, 45, [1,1,1]} where the exponent vector is bit-mapped into 1+100+10000. To multiply monomials with exponents we add the exponents, multiply the coefficients.
  - 20304 is $z^2y^3x^4$.
  - Bad news if x^{100} is computed since it will look like y. (Altran)

Kronecker again. One variable to NO variables

- Let x= t, y=t^{100} and z=t^{10000}.
- Then x+y+z is represented by t+t^{100}+t^{10000}
- Now evaluate this expression at t=some-big-number.
- How far can we run with this? Add, multiply (at least, as long as we don't overlap the exponent range).
- A hack used twice becomes a technique.
- A hack used three times becomes a method.
- A hack used four times becomes a methodology.

- (Eval down to 1 variable used for "heuristic GCD" first in Maple, used also in MuPAD but cannot be sole method)

What about polynomials in sin(x)?

- How far can we go by doing substitutions?
  - Let us replace sin(x) $\rightarrow$ s, cos(x) $\rightarrow$ c
  - Then sin(x)+cos(x) is the polynomial s+c.

- We must also keep track of simplifications that implement $s^2+c^2 \rightarrow 1$, derivative information such as ds/dx = c, and relations with sin(x/2) etc.

Modular representations

- Consider briefly a polynomial f(x) where coefficients are all reduced modulo some prime or a set of primes.{q1,q2,q3}
- What operations can be done by using one or more images?
- Compare to homework!
- Much more later.

What about polynomials in sqrt(2)?

- How far can we go by doing substitutions?
  - Let us replace sqrt(2) $\rightarrow$ u.

- We must also keep track of simplifications that implement $u^2 \rightarrow 2$, but the situation becomes rather more complicated because introduction of algebraic numbers, e.g. w=(1)^(1/8), leads to ambiguities: which root?
- Independence of simple algebraic extensions is not trivial; e.g. sqrt(6)/sqrt(3); or even
- w-w^3 = sqrt(2)
- w^4+1 = 0

What about polynomials in sqrt(x^2+y^2)?

- How far can we go by doing substitutions?
  - Let us replace sqrt(x^2+y^2) → u.
- We must also keep track of simplifications that implement u^2 → x^2+y^2, but the situation becomes rather more complicated again.

Logs and Exponential polynomials?

- Let exp(x) → E, log(x)→L.

- You must also allowing nesting of operations; then note that exp(-x)=1/exp(x)=1/E,
- And exp(log(x))=x, log(exp(x))= x+nπ i etc.
- We know that exp(exp(x)) is algebraically independent of exp(x), etc.
- Characterize "etc": what relations are there?
- Note that exp(1/2*log(x)) and sqrt(x) are similar.

Where next?

- We will see that most of the important efficiency breakthroughs in time-consuming algorithms can be found in polynomial arithmetic, often as part of the higher level representations.
- Tricks: evaluation and modular homomorphisms, Newton-like iterations, FFT
- Later, perhaps. Conjectures on e, π, independence.