10 Evaluating Polynomials (March 10)

10.1 Horner's Rule

Suppose we want to evaluate an arbitrary univariate polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$. The obvious algorithm—computing all the relevant powers of x and then computing their weighted sum—requires $2n + 1$ multiplications and n additions in the worst case. We can reduce the number of multipplcations to n by using the following recurrence for $p(x) = p_n(x)$:

$$p_0(x) = a_n$$
$$p_i(x) = a_{n-i} + x \cdot p_{i-1}(x)$$

This recurrence is commonly known as *Horner's rule*. In the mid-1950's, Ostrowski asked whether this is the fastest possible algorithm for computing arbitrary polynomials. Specifically, is there a straight-line program that accepts $n + 2$ arbitrary real numbers $a_0, a_1, \ldots, z_n, x$ as input and computes $\sum_{i=0}^{n} a_i x^i$ using less than n additions (or subtractions) or less than n multiplications? Surprisingly, the answer is no!

10.2 ...is optimal for additions...

Ostrowski proved almost immediately that Horner's rule uses the minimum possible number of additions and subtractions, even if we know in advance that $x = 1$. In other words, any straight-line program that computes the scalar sum $\sum_{i=0}^{n} a_i$ contains at least n additions or subtractions.

We can prove Ostrowski's theorem by induction on n as follows. Suppose $n = 1$. If our straight-line program has no additions or subtractions, it can only compute expressions of the form $ca_0^i a_i^j$, where c is a real[1] constant and i and j are non-negative integers. Since the equation

$$ca_0^i a_i^j = a_0 + a_1$$

is not true for all values of a_0 and a_1, we must perform at least one addition or subtraction to compute $a_0 + a_1$.

Now let A be a straight-line program that computes $\sum_{i=0}^{n} a_i$ for some $n > 1$. Suppose the first addition or subtraction in A is

$$v_i \leftarrow v_j \pm v_k;$$

Both v_j and v_k must be of the form $ca_0^{i_0} \cdots a_n^{i_n}$. Without loss of generality, we can assume that a_n appears with non-zero exponent in v_k. Now suppose a Little Birdie tells us that $a_n = 0$. Then we can eliminate this instruction from our straight-line program (since it is equivalent to $v_i \leftarrow v_j$). If we simplify the rest of the program by replacing every instance of x_n with 0, we obtain a new straight-line program to compute the sum $\sum_{i=0}^{n-1} a_i$. By the inductive hypothesis, this new program has at least $n - 1$ additions and subtractions, and therefore A has at least n addditions and subtractions.

10.3 ...and multiplications!

Over a decade later, Pan proved the corresponding optimality result for multiplications, using what is now known as the *method of linear independence*. The key idea is to consider the space of linear combinations of the inputs a_i as a vector space $\mathbb{R}^{n+1}$. The coefficients of any intermediate polynomial in x that is computed by our straight-line program can be considered vectors in this

[1] Actually, this argument—in fact, the entire lecture—applies to computing polynomials over any field, but for simplicity, I'll phrase everything over the reals.

space. The method of linear independence analyzes the number of linearly independent vectors among these coefficients.

Throughout this section, I'll use the notation $\ell(a_1, \ldots, a_n)$ denote some linear combination of $a_1, \ldots, a_n$; that is,

$$\ell(a_1, \ldots, a_n) = \sum_{i=1}^{n} c_i a_i$$

for some real constants c_i. In other words, $\ell(a_1, \ldots, a_n)$ is an unspecified vector in the real vector space whose basis is $a_1, \ldots, a_n$. Recall that the *rank* of a set of vectors is the maximum number of linearly independent elements. The method of linear independence relies on the following fact (which I won't prove).

Lemma 1. *Let $\mathcal{L} = \{\ell_1(a_1, \ldots, a_n), \ldots, \ell_u(a_1, \ldots, a_n)\}$ be a set of linear combinations of $a_1, \ldots, a_n$, and let $\ell(a_2, \ldots, a_n)$ be some other linear combination of $a_2, \ldots, a_n$. For each i, define*

$$\ell_i'(a_2, \ldots, a_n) = \ell_i(\ell(a_2, \ldots, a_n), a_2, \ldots, a_n),$$

and let $\mathcal{L}' = \{\ell_1'(a_2, \ldots, a_n), \ldots, \ell_u'(a_2, \ldots, a_n)\}$. The rank of $\mathcal{L}'$ is at most one less than the rank of $\mathcal{L}$.

Now consider a striaght-line program to compute $p(x)$. Every multiplication instruction multiplies two polynomials in the ring $\mathbb{R}[a_1, \ldots, a_n, x]$. We'll say that a mutliplication is *insignificant* if either both arguments are in $\mathbb{R}[x]$ or if one argument is in $\mathbb{R}[a_1, \ldots, a_n]$ and the other argument is a scalar.

Theorem 2. *Let A be a straight-line rpgroam over $\{+, -, \times\}$ with input $a_0, \ldots, a_n, x$ that computes the expression*

$$\sum_{i=0}^{u} \ell_i(a_1, \ldots, a_n) x^i + r(x),$$

where each ℓ_i is a nontrivial linear combination of $a_1, \ldots, a_n$ and $r(x)$ is a polynomial in x. The number of significant multiplications in A is at least the rank of $\mathcal{L} = \{\ell_1, \ldots, \ell_u\}$.

Proof: The proof is by induction on the number of significant multiplications. If our program has no significant multiplications, it can only compute expressions of the form $\ell_0(a_1, \ldots, a_n) + r(x)$. In this case, $\mathcal{L}$ is actually empty, so it has rank 0.

Suppose our program A has k significant multiplications. The first one must be of the form

$$\left(\sum_{i=1}^{n} c_i a_i + s(x) \right) \times \left(\sum_{i=1}^{n} d_i a_i + t(x) \right),$$

where the c_i and d_i are scalars, with at least one of each not equal to zero, and s and t are polynomials in x. Without loss of generality, assume that $c_1 \neq 0$. For some constant c, define

$$\ell(a_2, \ldots, a_n) = \frac{c - \sum_{i=2}^{n} c_i a_i}{c_1} \quad \text{and} \quad p(x) = \frac{-s(x)}{c_1}.$$

Now consider the restriction of A to inputs where $a_1 = \ell(a_2, \ldots, a_n) + p(x)$. Under this restriction, our program actually computes

$$\sum_{i=0}^{u} \ell_i(\ell(a_2, \ldots, a_n) + p(x), a_2 \ldots, a_n) x^i + r(x) = \sum_{i=0}^{u} \ell_i'(a_2 \ldots, a_n) x^i + r'(x),$$

where the ℓ'_i are defined exactly as in Lemma 1 and $r'(x)$ is some other polynomial.[2] Moreover, the first significant multiplication in A is no longer significant under this restriction.

Thus, we obtain a new straight-line program A' with at most $k-1$ significant multiplcations. The inductive hypothesis implies that the rank of $\mathcal{L}' = \{l'_1, \ldots, \ell'_u\}$ is at most $k-1$. It now follows from Lemma 1 that $\mathcal{L}$ has rank at most k. □

This theorem immediately implies the desired lower bound of n multiplications, since the set of linear combinations $\{a_1, a_2, \ldots, a_n\}$ clearly has rank n.

10.4 Preprocessing Helps...

Now suppose we are told the coefficients $a_0, \ldots, a_n$ in advance, and asked to write a straight-line program to evaluate $p(x) = \sum_{i=0}^{n} a_i x^i$ given only the value of x as input. In other words, instead of a program to evaluate arbitrary polynomials, we want a program to evaluate some *specific* polynomial.

It's not hard to discover specific poylynomials that can be evaluated using less than n additions and n multiplications. For example, we can evaluate the polynomial x^n using at most $2\lceil \lg n \rceil$ multiplications (using prepeated squaring) and no additions at all. Somewhat surprisingly, *any* specific polynomial of degree n can be evaluated using roughly $n/2$ multiplications; even more surprisingly, this bound is tight in the worst case.

I'll only sketch the upper bound; for details, see Knuth II. For simplicity, assume that n is even. The basic idea is to write the polynomial $p(x)$ in the alternate form

$$p(x) = p_n(x) = (x^2 - a_n)p_{n-2}(x) + b_n x + c_n,$$

where $p_{n-2}(x)$ has degree at most $n-2$ and a_n, b_n, c_n are real constants. Continuing recursively, we obtain a series of polynomials $p_{n-4}(x), p_{n-6}(x), \ldots, p_0(x)$, where

$$p_i(x) = (x^2 - a_i)p_{i-2}(x) + b_i x + c_i$$

and $p_0(x) = c_0$. The key insight is that it is possible to choose the constants a_i and c_i so that $b_i = 0$ for all i. Once we know these constants, the previous recurrence gives us an algorithm that uses only $n/2 + 1$ multiplications; one to compute x^2, and one to compute each polynomial $p_i(x)$ from its predecessor $p_{i-2}(x)$.

10.5 ...but not very much

To prove that $n/2$ multiplcaitions are sometimes necessary, we use a generalization of the method of linear independence, called the method of *algebraic* independence.

A set of real numbers $\{a_1, a_2, \ldots, a_n\}$ is *algebraically dependent* (over the rationals) if there is a nonzero polynomial $p(x_1, x_2, \ldots, x_n)$ with rational coefficients such that $p(a_1, a_2, \ldots, a_n) = 0$. The *transcendence degree* of a set is the size of its largest algebraically independent subset. For example, the set $\{1, \sqrt{2}, \pi\}$ has transcendence degree 2.

[2] This is a lie.