In this lecture, we will see some of the power of polynomials in algorithm design.

You've probably all seen polynomials before: e.g., $3x^2 - 5x + 17$, or $2x - 3$, or $-x^5 + 38x^3 - 1$, or x, or even a constant 3. These are all polynomials over a single variable (here, x). The degree of a polynomial is the highest exponent of x that appears: hence the degrees of the above polynomials are $2, 1, 5, 1, 0$ respectively.

In general, a polynomial over the variable x of degree at most d looks like:

$$P(x) = c_d x^d + c_{d-1} x^{d-1} + .. + c_1 x + c_0$$

Note that the sequence of $d+1$ coefficients $\langle c_d, c_{d-1}, \ldots, c_0 \rangle$ completely describes $P(x)$.

Hence, if the coefficients were all drawn from the set $\mathbb{Z}_p$, then we have exactly p^{d+1} possible different polynomials. This includes the zero polynomial $0 = 0x^d + 0x^{d-1} + .. + 0x + 0$.

In this lecture, we will use properties of polynomials, to construct error correcting codes, and do other cool things with them.

1 Operations on Polynomials

Before we study properties of polynomials, recall the following simple operations on polynomials:

- Given two polynomials $P(x)$ and $Q(x)$, we can add them to get another polynomial $R(x) = P(x) + Q(x)$. Note that the degree of $R(x)$ is at most the maximum of the degrees of P and Q. (Q: Why is it not equal to the maximum?)

  $$(x^2 + 2x - 1) + (3x^3 + 7x) = 3x^3 + x^2 + 9x - 1$$

  The same holds for the difference of two polynomials $P(x) - Q(x)$, which is the same as $P(x) + (-Q(x))$.

- Given two polynomials $P(x)$ and $Q(x)$, we can multiply them to get another polynomial $S(x) = P(x) \times Q(x)$.

  $$(x^2 + 2x - 1) \times (3x^3 + 7x) = 3x^5 + 4x^3 + 6x^4 + 14x^2 - 7x$$

  The degree of $S(x)$ is equal to the sum of the degrees of P and Q.

- Note that $P(x)/Q(x)$ may not be a polynomial.

- We can also *evaluate* polynomials. Given a polynomial $P(x)$ and a value a, $P(a) := c_d \cdot a^d + c_{d-1} \cdot a^{d-1} + .. + c_1 \cdot a + c_0$. For example, if $P(x) = 3x^5 + 4x^3 + 6x^4 + 14x^2 - 7x$, then

  $$P(2) = 3 \cdot 2^5 + 4 \cdot 2^3 + 6 \cdot 2^4 + 14 \cdot 2^2 - 7 \cdot 2 = 266$$

  Of course, the multiplication between the c_i's and a must be well-defined, as should the meaning of a^i for all i. E.g., if the c_i's and a are reals, this is immediate.

  But also, if c_i's and a all belonged to $\mathbb{Z}_p$ for prime p, evaluation would still be well-defined. For instance, if we were working in $\mathbb{Z}_{17}$, then

  $$P(2) = 266 \pmod{17} = 11$$

- A *root* of a polynomial $P(x)$ is a value r such that $P(r) = 0$. For example, $P(x)$ above has three real roots $0, -1 + \sqrt{2}, -1 - \sqrt{2}$, and two complex roots.

Here, we were implictly working over $\mathbb{R}$, the field of real numbers. But almost everything today holds true for any field, say a finite field of integers modulo a prime ($\mathbb{F}_p$ for prime p).

2 How Many Roots?

Let's start with the following super-important theorem.

Theorem 1 (Few-Roots Theorem) *Any non-zero polynomial of degree at most d has at most d roots.*

This holds true, regardless of what field we are working over. When we are working over the reals (i.e., the coeffcients are reals, and we are allowed to plus in arbitrary reals for x), this theorem is a corollary of the fundamental theorem of Algebra. But it holds even if we are working over some other field (say $\mathbb{Z}_p$ for prime p).

Let's relate this to what we know. Consider polynomials of degree 1, also known as linear polynomials. Say they have real coefficients, this gives a straight line when we plot it. Such a polynomial has at most one root: it crosses the x-axis at most once. And in fact, any degree-1 polynomial looks like $c_1x + c_0$, and hence setting $x = -c_0/c_1$ gives us a root. So, in fact, a polynomial of degree exactly 1 has exactly one root.

What about degree 2, the quadratics? Things get a little more tricky now, as you probably remember from high school. E.g., the polynomial $x^2 + 1$ has no real roots, but it has two complex roots. However, you might remember that if it has one real root, then both roots are real. But anyways, a quadratic crosses the x-axis at most twice. At most two roots.

And in general, Theorem 1 says, any polynomial of degree at most d has at most d roots.

3 A New Representation for degree-d Polynomials

Let's prove a simple corollary of Theorem 1, which says that if we plot two polynomials of degree at most d, then they can intersect in at most d points—*unless they are the same polynomial* (and hence intersect everywhere)! Remember, two distinct lines intersect at most once, two distinct quadratics intersect at most twice, etc. Same principle.

Corollary 2 *Given $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$, where the a_i's are distinct, there is* at most *one polynomial $P(x)$ of degree at most d, such that $P(a_i) = b_i$ for all $i = 0, 1, \ldots, d$.*

Proof: For a contradiction, suppose there are two distinct polynomials $P(x)$ and $Q(x)$ of degree at most d such that for all i,

$$P(a_i) = Q(a_i) = b_i.$$

Then consider the polynomial $R(x) = P(x) - Q(x)$. It has degree at most d, since it is the difference of two polynomials of degree at most d. Moreover,

$$R(a_i) = P(a_i) - Q(a_i) = 0$$

for all the $d+1$ settings of $i = 0, 1, \ldots, d$. Once again, R is a polynomial of degree at most d, with $d+1$ roots. By the contrapositive of Theorem 1, $R(x)$ must be the zero polynomial. And hence $P(x) = Q(x)$, which gives us the contradiction. ■

To paraphrase the theorem differently, given two (i.e., $1+1$) points there is at most one linear (i.e., degree-1) polynomial that passes through them, given three (i.e., $2+1$) points there is at most one quadratic (i.e., degree-2) polynomial that passes through them, etc.

Can it be the case that for some $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$, there is *no* polynomial of degree at most d that passes through them? Well, clearly if $a_i = a_j$ but $b_i \neq b_j$. But what if all the a_i's are distinct?

Theorem 3 (Unique Reconstruction Theorem) *Given $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$ where the a_i's are distinct, there always exists a polynomial $P(x)$ of degree at most d, such that $P(a_i) = b_i$ for all $i = 0, 1, \ldots, d$.*

We will prove this theorem soon, but before that note some implications. Combining Corollary 2 with Theorem 3, we get that given $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$ with distinct a's, this means there is a *unique* polynomial of degree at most d that passes through them. Exactly one.

In fact, given $d+1$ numbers $b_0, b_1, \ldots, b_d$, there is a unique polynomial $P(x)$ of degree at most d such that $P(i) = b_i$. (We're just using the theorem with $a_i = i$.) Earlier we saw how to represent any polynomial of degree at most d by $d+1$ numbers, the coefficients. Now we are saying that we can represent the polynomial of degree at most d by a different sequence of $d+1$ numbers: its values at $0, 1, \ldots d$.

Two different representations for the same thing, cool! Surely there must be a use for this new representation. We will give at least two uses for this, but first let's see the proof of Theorem 3. (If you are impatient, you can skip over the proof, but do come back and read it—it is very elegant.)

3.1 The Proof of Theorem 3*

OK, now the proof. We are given $d+1$ pairs (a_i, b_i), and the a's are all distinct. The proof will actually give an algorithm to find this polynomial $P(x)$ with degree at most d, and where $P(a_i) = b_i$.

Let's start easy: suppose all the $d+1$ values b_i's were zero. Then $P(x)$ has $d+1$ roots, and now Theorem 1 tells us that $P(x) = 0$, the zero polynomial!

OK, next step. Suppose $b_0 = 1$, but all the d other b_i's are zero. Do we know a degree-d polynomial which has roots at d places $a_1, a_2, \ldots, a_d$. Sure, we do—it is just

$$Q_0(x) = (x - a_1)(x - a_2) \cdots (x - a_d).$$

So are we done? Not necessarily: $Q_0(a_0)$ might not equal $b_0 = 1$. But that is easy to fix! Just scale the polynomial by $1/Q_0(a_0)$. I.e., what we wanted was

$$\begin{aligned} R_0(x) &= (x - a_1)(x - a_2) \cdots (x - a_d) \cdot \frac{1}{Q_0(a_0)} \\ &= \frac{(x - a_1)(x - a_2) \cdots (x - a_d)}{(a_0 - a_1)(a_0 - a_2) \cdots (a_0 - a_d)} \quad . \end{aligned}$$

Again, $R_0(x)$ has degree d by construction, and satisfies what we wanted! (We'll call $R_0(x)$ the 0^{th} "switch" polynomial.)

Next, what if b_0 was not 1 but some other value. Easy again: just take $b_0 \times R_0(x)$. This has value $b_0 \times 1$ at a_0, and $b_0 \times 0 = 0$ at all other a_i's.

Similarly, one can define switch polynomials $R_i(x)$ of degree d that have $R_i(a_i) = 1$ and $R_i(a_j) = 0$ for all $i \neq j$. Indeed, this is

$$R_i(x) = \frac{(x-a_0)\cdots(x-a_{i-1})\cdot(x-a_{i-1})\cdots(x-a_d)}{(a_i-a_0)\cdots(a_i-a_{i-1})\cdot(a_i-a_{i-1})\cdots(a_i-a_d)} \quad .$$

So the polynomial we wanted after all is just a linear combination of these switch polynomials:

$$P(x) = b_0R_0(x) + b_1R_1(x) + \ldots + b_dR_d(x)$$

Since it is a sum of degree-d polynomials, $P(x)$ has degree at most d. And what is $P(a_i)$? Since $R_j(a_i) = 0$ for all $j \neq i$, we get $P(a_i) = b_iR_i(a_i)$. Now $R_i(a_i) = 1$, so this is b_i. All done.[1][2]

Example: Consider the tuples $(5,1),(6,2),(7,9)$: we want the unique degree-2 polynomial that passes through these points. So first we find $R_0(x)$, which evaluates to 1 at $x = 5$, and has roots at 6 and 7. This is

$$R_0(x) = \frac{(x-6)(x-7)}{(5-6)(5-7)} = \frac{1}{2}(x-6)(x-7)$$

Similarly

$$R_1(x) = \frac{(x-5)(x-7)}{(6-5)(6-7)} = -(x-5)(x-7)$$

and

$$R_2(x) = \frac{(x-5)(x-6)}{(7-5)(7-6)} = \frac{1}{2}(x-5)(x-6)$$

Hence, the polynomial we want is

$$P(x) = 1\cdot R_0(x) + 2\cdot R_1(x) + 9\cdot R_2(x) = 3x^2 - 32x + 86$$

Let's check our answer:

$$P(5) = 1, P(6) = 2, P(7) = 9.$$

Running Time: Note that constructing the polynomial $P(x)$ takes $O(d^2)$ time. (Can you find the simplified version in this time as well?) If you chose the points $a_0,\ldots,a_d$ carefully, you could do this in $O(d\log d)$ time. For this, you choose 2^r roots of unity for $r = 2^{\lceil \log_2(d+1)\rceil}$, and then use a divide-and-conquer idea: the resulting algorithm is called the Fast Fourier Transform.

4 Application: Error Correcting Codes

Consider the situation: I want to send you a sequence of $d+1$ numbers $\langle c_d, c_{d-1},\ldots,c_1,c_0\rangle$ over a noisy channel. I can't just send you these numbers in a message, because I know that whatever message I send you, the channel will corrupt up to k of the numbers in that message. For the current example, assume that the corruption is very simple: whenver a number is corrupted, it is replaced by a $\star$. Hence, if I send the sequence

$$\langle 5, 19, 2, 3, 2\rangle$$

[1] This algorithm is called Lagrange interpolation, after Joseph-Louis Lagrange, or Giuseppe Lodovico Lagrangia, depending on whether you ask the French or the Italians. (He was born in Turin, and both countries claim him for their own.) And to muddy things even further, much of his work was done at Berlin. He later moved to Paris, where he survived the French revolution—though Lavoisier was sent to the guillotine because he intervened on behalf of Lagrange. Among much other great research, he also gave the first known proof of Wilson's theorem, that n is a prime if and only if $n|(n-1)!+1$—apparently Wilson only conjectured Wilson's theorem.

[2] You can use an idea very similar to Lagrange interpolation to prove the Chinese Remainder Theorem.

and the channel decides to corrupt the third and fourth numbers, you would get

$$\langle 5, 19, \star, \star, 2\rangle.$$

On the other hand, if I decided to delete the fourth and fifth elements, you would get

$$\langle 5, 19, 2, \star, \star\rangle.$$

Since the channel is "erasing" some of the entries and replacing them with $\star$'s, the codes we will develop will be called *erasure* codes. The question then is: how can we send $d+1$ numbers so that the receiver can get back these $d+1$ numbers even if up to k numbers in the message are erased (replaced by $\star$s)? (Assume that both you and the receiver know d and k.)

A simple case: if $d = 0$, then one number is sent. Since the channel can erase k numbers, the best we can do is to repeat this single number $k+1$ times, and send these $k+1$ copies across. At least one of these copies will survive, and the receiver will know the number.

This suggests a strategy: no matter how many numbers you want to send, repeat each number $k+1$ times. So to send the message $\langle 5, 19, 2, 3, 2\rangle$ with $k = 2$, you would send

$$\langle 5, 5, 5, 19, 19, 19, 2, 2, 2, 3, 3, 3, 2, 2, 2\rangle$$

This takes $(d+1)(k+1)$ numbers, approximately dk. Can we do better?

Indeed we can! We view our sequence $\langle c_d, c_{d-1}, \ldots, c_1, c_0\rangle$ as the $d+1$ coefficients of a polynomial of degree at most d, namely $P(x) = c_d x^d + c_{d-1}x^{d-1} + .. + c_1 x + c_0$. Now we evaluate P at some $d+k+1$ points, say $0, 1, 2, \ldots, d+k$, and send these $d+k+1$ numbers

$$P(0), P(1), \ldots, P(d+k)$$

across. The receiver will get back at least $d+1$ of these numbers, which by Theorem 3 uniquely specifies $P(x)$. Moreover, the receiver can also reconstruct $P(x)$ using, say, Langrange interpolation.

Example: Here is an example: Suppose we want to send $\langle 5, 19, 2, 3, 2\rangle$ with $k = 2$. Hence $P(x) = 5x^4 + 19x^3 + 2x^2 + 3x + 2$. Now we'll evaluate $P(x)$ at $0, 1, 2, \ldots d+k = 6$. This gives

$$P(0) = 2, P(1) = 31, P(2) = 248, P(3) = 947, P(4) = 2542, P(5) = 5567, P(6) = 10676$$

So we send across the "encoded message":

$$\langle 2, 31, 248, 947, 2542, 5567, 10676\rangle$$

Now suppose the third and fifth entries get erased. the receiver gets:

$$\langle 2, 31, \star, 947, \star, 5567, 10676\rangle$$

So she wants to reconstruct a polynomial $R(x)$ of degree at most 4 such that $R(0) = 2, R(1) = 31, R(3) = 947, R(5) = 5567, R(6) = 10676$. (That is, she wants to "decode" the message.) By Langrange interpolation, we get that

$$R(x) = \frac{1}{45}(x-1)(x-3)(x-5)(x-6) - \frac{31}{40}x(x-3)(x-5)(x-6) + \frac{947}{36}x(x-1)(x-5)(x-6)$$
$$-\frac{5567}{40}x(x-1)(x-3)(x-6) + \frac{5338}{45}x(x-1)(x-3)(x-5)$$

which simplifies to $P(x) = 5x^4 + 19x^3 + 2x^2 + 3x + 2$!

Note on the Running Time: The numbers can get large, so you may want work in the field $\mathbb{F}_p$, as long as the size of the field is large enough to encode the numbers you want to send across. (Of course, if you are working modulo a prime p, both the sender and the receiver must know the prime p.)

> **Example:** Since we want to send numbers as large as 19, let's work in $\mathbb{Z}_{23}$. Then you'd send the numbers modulo 23, which would be
>
> $$\langle 2, 8, 18, 4, 12, 1, 4 \rangle$$
>
> Now suppose you get
>
> $$\langle 2, 8, \star, 4, \star, 1, 4 \rangle$$
>
> Interpolate to get
>
> $$R(x) = 45^{-1}(x-1)(x-3)(x-5)(x-6) - 5^{-1}x(x-3)(x-5)(x-6) + 9^{-1}x(x-1)(x-5)(x-6)$$
> $$-40^{-1}x(x-1)(x-3)(x-6) + 2 \cdot 45^{-1}x(x-1)(x-3)(x-5)$$
>
> where the multiplicative inverses are modulo 23, of course. Simplifying, we get $P(x) = 5x^4 + 19x^3 + 2x^2 + 3x + 2$ again.

4.1 Error Correction

One can imagine that the channel is more malicious: it decides to *replace* some k of the numbers not by stars but by other numbers, so the same encoding/decoding strategy cannot be used! Indeed, the receiver now has no clue which numbers were altered, and which ones were part of the original message! In fact, even for the $d = 0$ case of a single number, we need to send $2k + 1$ numbers across, so that the receiver knows that the majority number must be the correct one. And indeed, if you evaluate $P(x)$ at $n = d + 2k + 1$ locations and send those values across, even if the channel alters k of those numbers, there is a unique degree-d polynomial that agrees with $d + k + 1$ of these numbers (and this must be $P(x)$)—this is not hard to show. What is more interesting is that the receiver can also reconstruct $P(x)$ fast: this is known as the *Berlekamp-Welch* algorithm. We will cover this if we have time.

4.1.1 The Berlekamp-Welch Error-Correction Algorithm*

Let $[n] := \{0, 1, \ldots, n-1\}$. Suppose we send over the n numbers $s_0, s_1, \ldots, s_{n-1}$, where $s_i = P(i)$. We receive numbers $r_0, r_1, \ldots, r_{n-1}$, where at most k of these r_is are not the same as the s_is. Define a set Z of size k such that $Z \supseteq \{i \mid s_i \neq r_i\}$: it contains all the error locations.

Now define a degree-j "error" polynomial $E(x)$ such that

$$E(x) = \prod_{a \in Z} (x - a).$$

Observe that

$$P(x) \cdot E(x) = r_i \cdot E(x) \qquad \forall x \in [n].$$

Indeed, $E(x) = 0$ for all $x \in Z$ and $P(x) = r_i$ for all $x \in [n] \setminus Z$. Of course, we just received the r_is, and don't know $P(x)$. Nor do we know $E(x)$, since we don't know Z.

But we know that $E(x)$ looks like

$$E(x) = x^k + e_{k-1}x^{k-1} + \ldots + e_1 x + e_0. \tag{1}$$

for some values $e_{k-1}, e_{k-2}, \ldots, e_0$. Moreover, we know that $P(x) \cdot E(x)$ has degree $d+k$, so looks like

$$P(x) \cdot E(x) = x^{d+k} + f_{d+k-1}x^{d+k-1} + \ldots + f_1 x + f_0. \tag{2}$$

So we get $n = d + 2k + 1$ equalities that look like

$$x^k + e_{k-1}x^{k-1} + \ldots + e_1 x + e_0 = x^{d+k} + f_{d+k-1}x^{d+k-1} + \ldots + f_1 x + f_0, \tag{3}$$

one for each $x \in [n]$. The unknown are e_i and f_i values—there are $k + (d + k + 1) = d + 2k + 1$ unknowns. So we can solve for these using Gaussian elimination, and get $E(x)$ and $P(x) \cdot E(x)$. Dividing the latter by the former gives back $P(x)$. It's like magic.

Exercise: Show there is a unique solution to this system of n equalities in at most n variables.

5 Multivariate Polynomials and Matchings

Here's a very different application of polynomials in algorithm design. Now we'll consider multivariate polynomials, and use the fact that they also have "few" roots (where "few" is interpreted suitably) to get an unusual algorithm for finding matchings in graphs.

We can rephrase Theorem 1 as follows: if we fix a set S of values in the field we are working over (e.g., $\mathbb{R}$, or $\mathbb{F}_p$), and pick a random $X \in S$, the probability that $P(X) = 0$ is at most $d/|S|$. One can extend this to the following theorem for multivariate polynomials $P(\mathbf{x}) = P(x_1, x_2, \ldots, x_m)$. Recall that the degree of a monomial is the sum of exponents of the variables in it, and the degree of a polynomial is the maximum degree of any non-zero monomial in it.

Theorem 4 (Schwartz (1980), Zippel (1979)) *For any non-zero degree-d polynomial $P(\mathbf{x})$ and any subset S of values from the underlying field, if each X_i is chosen independently and uniformly from S, then*

$$\Pr[P(X_1, \ldots, S_m) = 0] \leq \frac{d}{|S|}.$$

This theorem is useful in many contexts. E.g., we get yet another algorithm for perfect matchings.

Definition 5 *For any graph $G = (V, E)$ with vertices $v_1, v_2, \ldots, v_n$, the* Tutte matrix[3] *and is a $|V| \times |V|$ matrix $M(G)$:*

$$M(G)_{i,j} = \begin{cases} x_{i,j} & \text{if } \{v_i, v_j\} \in E \text{ and } i < j \\ -x_{j,i} & \text{if } \{v_i, v_j\} \in E \text{ and } i > j \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$

This is a square matrix of variables $x_{i,j}$. And like any matrix, we can take its determinant, which is a (multivariate) polynomial $P_G(\mathbf{x})$ in the variables $\{x_{i,j}\}_{\{i,j\} \in E}$. The degree of this polynomial is at most $n = |V|$, the dimension of the matrix. Here is a surprising (and not difficult) fact:[4]

[3] Named after William T. (Bill) Tutte, pioneering graph theorist and algorithm designer. Recently it was discovered that he was one of the influential code-breakers in WWII, making crucial insights in breaking the Lorenz cipher.

[4] For bipartite graphs you can define the "Edmonds" matrix of $G = (U, V, E)$ is an $|U| \times |V|$ matrix $M_b(G)$ with the entry at row i and column j,

$$M_b(G)_{i,j} = \begin{cases} 0 & \text{if } (u_i, v_j) \notin E \\ x_{i,j} & \text{if } (u_i, v_j) \in E \end{cases}$$

Theorem 6 is easy to prove for this case. The proof for general graphs requires a little more care.

Theorem 6 (Tutte (1947)) *A graph G has a perfect matching if and only if $P_G(\mathbf{x})$, the determinant of the Tutte matrix, is not the zero polynomial.*

How do we check if $P_G(\mathbf{x})$ is zero or not? That's the problem: since we're taking a determinant of a matrix of variables, the usual way of computing determinants may lead to $n!$ terms, which eventually may all cancel out!

However, we can combine Theorems 4 and 6 together: take G, construct $M(\mathbf{x})$, and replace each variable by an independently uniform random value in some set S, and then compute the determinant of the resulting matrix of random numbers. This is exactly like plugging in the random numbers into $P_G(\mathbf{x})$. So if $P_G(\mathbf{x})$ was zero, the answer is zero for sure. And else, the answer is zero with probability at most $n/|S|$, which we can make as small as we want by choosing S large enough.

> **Exercise:** Think about how you would use this algorithm to find perfect matching (with high probability) in a graph, if one exists.
>
> **Exercise:** Given an algorithm to find perfect matchings in a graph (if one exists), use it to find maximum cardinality matchings in graphs.