Math 55 - Spring 2004 - Lecture notes #26 - April 29 (Thursday)

Goals for today: Generating functions

This material is covered by Lenstra and Rosen section 6.4.
We will not cover any other sections of chapter 6 of Rosen.

Def: Let P=(p(0), p(1), p(2), ...) be a (finite or infinite) sequence of real numbers. The generating function G of P is the (finite or infinite) series

$$G(x) = p(0) + p(1)x + p(2)x^2 + \ldots + p(i)x^i + \ldots$$

Note: if P is a finite sequence, G(x) is just a polynomial. But sometimes it is convenient to write

$$G(x) = \sum_{k=0 \text{ to infinity}} p(k)x^k$$

with the understanding that all p(k) = 0 for k large enough.

Note: if we have two sequences P and Q=(q(0),q(1),...), we will distinguish their generating functions by writing G_P(x) and G_Q(x)

G(x) can be used to compute useful properties of the sequence P.

EX: Let P = (p(0), p(1), ...) where
p(i) = probability of element i in a sample space.
In other words each p(i) >= 0 and their sum is one.
Then

$$G(1) = \sum_{k=0 \text{ to infinity}} p(k)1^k = \sum_{k=0 \text{ to infinity}} p(k) = 1$$

Now let f be a random variable such that P(f=i)=p(i)
(in particular f() is only allowed to have nonnegative integer values).
We can compute its expectation E(f) and variance V(f) using G(x) as follows:

$$G'(x) = \sum_{k=0 \text{ to infinity}} p(k)kx^{k-1}$$

so

```
       G'(1) = sum_{k=0 to infinity} p(k)*k*1^(k-1)
             = sum_{k=0 to infinity} p(k)*k
             = sum_{k=0 to infinity} P(f=k)*k
             = E(f)  ... by a Theorem about expectation

      Similarly
       G''(x) = sum_{k=0 to infinity} p(k)*k*(k-1)*x^(k-2)
      so
       G''(1) = sum_{k=0 to infinity} p(k)*k*(k-1)
              = sum_{k=0 to infinity} p(k)*(k^2 - k)
              = sum_{k=0 to infinity} p(k)*(k^2)
                - sum_{k=0 to infinity} p(k)*(k)
              = sum_{k=0 to infinity} P(f=k)*(k^2)
                - sum_{k=0 to infinity} P(f=k)*k
              = E(f^2) - E(f)
              = E(f^2) - G'(1)
      so
        V(f)  = E(f^2) - (E(f))^2
              = G''(1) + G'(1) - (G'(1))^2

   EX: Suppose we toss a biased coin n times, with P(Head) = p,
       and let p(i) = P(getting i Heads) = C(n,i)*p^i*(1-p)^(n-i).
       Then
         G(x) = sum_{i=0 to n} p(i)*x^i
              = sum_{i=0 to n} C(n,i)*(p*x)^i*(1-p)^(n-i)
              = (p*x+1-p)^n ... by the Binomial Theorem
       so if f = number of Heads in n tosses
         E(f) = G'(1) = n*p*(p*x+1-p)^(n-1) at x=1
                      = n*p ... as expected
ASK&WAIT: How else can we compute E(f)?
         V(f) = G''(1) + G'(1) - (G'(1))^2
              = n*(n-1)*p^2*(p*x+1-p)^(n-2) at x=1 + n*p - (n*p)^2
              = n*(n-1)*p^2 + n*p - (n*p)^2
              = n*p*(1-p) ... as expected
ASK&WAIT: How else can we compute V(f)?

   Theorem 1: Let G_P(x) = sum_{k=0 to infinity} p(k)*x^k and
                  G_Q(x) = sum_{k=0 to infinity} q(k)*x^k
              Then
                G_P(x) * G_Q(x)
                    = sum_{k=0 to infinity} c(k)*x^k
              where
```

```
            c(k) = sum_{j=0 to k} p(j)*q(k-j)

    proof: just multiplying polynomials (or power series)

Theorem 2: suppose f(x) and g(x) are independent random variables
           with Prob(f=i) = p(i) and Prob(g=i) = q(i).
           Let G_P(x) = sum_{k=0 to infinity} p(k)*x^k   and
           Let G_Q(x) = sum_{k=0 to infinity} q(k)*x^k
           denote their generating functions.
           Then the generating function of f(x)+g(x) is
             G_P(x)*G_Q(x)

   proof:  Prob(f(x)+g(x)=k) = sum_{j=0 to k} Prob(f(x)=j and g(x)=k-j)
                             = sum_{j=0 to k} Prob(f(x)=j)*Prob(g(x)=k-j)
                                  ... by independence of f and g
                             = sum_{j=0 to k} p(j)*q(k-j)
                             = c(k) ...  as defined above
            so the generating function of f+g is
              sum_{k=0 to infinity} c(k)*x^k
            = G_P(x)*G_Q(x)  ... by Theorem 1

 Ex: Let f(x) be the random variable that = 1 if a
     biased coin comes up H, and =0 if it comes up T.
     Then its generating function is G(x) = (1-p) + p*x.
     Now flip a coin n times, and let f_i(x) = 1 if the i-th flip
     comes up H and 0 otherwise. Then the generating function of
       f = f_1 + f_2 + ... + f_n
         = total number of Heads
     is
         G(x) * G(x) * ... * G(x)
           ... by Theorem 2, since each coin flip is independent
        = (1-p + p*x)^n
     This is the same answer as before, gotten by the binomial theorem.

     But Theorem 2 is more general, since we could use a different
     coin for each flip with a different probability p(i) of coming up H.
     The generating function of each f_i(x) would then be
        G_i(x) = (1-p(i) + p(i)*x)
     and the generating function of
        f = f_1 + ... + f_n = total number of Heads
     would be
        G_1(x) * ... * G_n(x) = prod_{i=1 to n} (1-p(i) + p(i)*x)
```

Ex: Recall the pictures from Lecture 23, of the probabilities of i heads after tossing a coin n times, or of getting a total of i after rolling a die n times and adding the results. These plots were computed using Theorem 2 as follows.

For tossing a fair coin n times, I computed the generating function after 1 toss G(x) = .5 + .5*x, and then multiplied this polynomial times itself n times to get

```
(G(x))^2 = 1/2^2 + 2/2^2 * x + 1/2^2 * x^2
(G(x))^3 = 1/2^3 + 3/2^3 * x + 3/2^3 * x^2 + 1/2^3 * x^3
...
(G(x))^n = 1/2^n + ...
```

from which I extracted the coefficients for plotting:

```
[1/4  2/4  1/4]
[1/8  3/8  6/8  3/8  1/8]
...
```

The point is that polynomial multiplication is a simple and systematic method to solve lots of probability problems. Polynomial multiplication is a built-in command in several programming environments (Matlab, Mathematica, Maple, ...). In Matlab the name of the command is "conv" which is short for "convolution". (If you have taken EECS 20 or similar course, you have probably encountered convolutions before. The same idea - computing probabilities of f+g, polynomial multiplication, convolution - comes up in many places!)

Now we turn to the use of generating functions for counting problems. The coefficients of the generating function will be integers, which will represent the number of objects of certain kinds.

Ex: The problem is to find the number of solutions to

```
      e1 + e2 + e3 = 16
```

where e1 can take on values from the set E1={2,3,5,6}
e2 can take on values from the set E2={3,5,6,7}, and
e3 can take on values from the set E3={2,5,8,9}, and
For example, e1+e2+e3 = 2+6+8 = 3+5+8 = 16 are two solutions.
How many solutions are there? We solve this by generating functions as follows. We represent

```
e1 by p1(x) = x^2 + x^3 + x^5 + x^6
e2 by p2(x) = x^3 + x^5 + x^6 + x^7
```

```
    e3 by p3(x) = x^2 + x^5 + x^8 + x^9
and multiply these polynomials together to get a bigger polynomial
    G(x) = p1(x)*p2(x)*p3(x)
Let c*x^16 be one term from G(x). It turns out that the
integer c is the answer to our problem.
Here is why. When you multiply these three polynomials out, you
get a contribution
      x^e1 * x^e2 * x^e3  = x^(e1 + e2 + e3)
for every e1 in E1, e2 in E2 and e3 in E3, corresponding to one
power of x from each polynomial p1(x), p2(x), p3(x). When e1+e2+e3=16,
you get a contribution of 1 to the constant c in c*x^16.
You get such a contribution for each triple (e1,e2,e3) that adds up
to 16, so that c counts the number of such triples, as desired.

It turns out that
    G(x) = x^22 + 3*x^21 + 4*x^20 + 4*x^19 + 6*x^18 + 8*x^17 +
           6*x^16 + ... + x^7
so the answer to our question is c=6. But in fact G(x) tells us
more: the coefficient of any x^k tell us the number of solutions
of e1+e2+e3 = k, i.e. it is the generating function for the number
of solutions to e1+e2+e3=k with e1 in E1, e2 in E2 and e3 in E3.
So for example, there is one solution to e1+e2+e3 = 22,
namely e1,e2,e3 all equalling their maximum values.
Similarly there is one solution to e1+e2+e3 = 7, when they all equal
their minimum values. This example obviously generalizes to the sum
of any number of ei lying in any sets Ei.

EX: How many ways can 8 cookies be distributed among 3 children, so that
    each child gets between 2 and 4 cookies? This is the same setup as
    above: We represent each child by the polynomial p(x) = x^2 + x^3 + x^4,
    since each child can get 2, 3 or 4 cookies, compute
       G(x) = (p(x))^3
    since there are 3 children, and look at the coefficient c of x^8 in G(x).
    c is the answer. In fact
       G(x) = x^12 + 3*x^11 + ... + 6*x^8 + ... + x^6
    so there are 6 ways to distribute 8 cookies.

EX: In the last two examples, the generating functions have been
    polynomials. Now we have an example were it is an infinite series.
    Our goal is to compute the number of r-combinations from a set
    with n-objects, where repetition is allowed. For example, from the
    set {1,2,3} with n=3 objects, the set of all 2-combinations with
```

repetition is
{{1,1}, {2,2}, {3,3}, {1,2}, {2,3}, {1,3}}
i.e. there are 6 possibilities.

Since the first item in the set may be chosen 1, 2, 3, ... times
we represent these choices by the infinite series
p(x) = 1+x+x^2+x^3+...
Since each of the n items in the set may be chosen, the complete
generating function is G(x) = (p(x))^n.
For example, with n=3 we get
G(x) = (1+x+x^2+x^3+...)^3
= 1 + 3*x + 6*x^2 + 10*x^3 + ...
But there is a much simpler way to write down G(x).
since p(x) is a geometric series we can sum it getting
p(x) = 1 + x + x^2 + ... = 1/(1-x) when |x| < 1
so in fact
G(x) = (p(x))^n = 1/(1-x)^n
which is a simple function.

Here is another way to get the answer, one that we figured out
before using "stars and bars". The way to represent all the
ways of choosing r items from n with repeated copies allowed is
to write down r stars and n-1 bars in any order. The bars
separate the stars into n groups, each with r1, r2, ... rn stars
such that r1+r2+...+rn = r. For example with
n=3 and r=2
|| represents {1,2}
|**| represents {2,2} etc.
As we showed in Chapter 4, the number of sequences of r stars and n-1
bars is C(r+n-1,r). Thus we have shown that
G(x) = 1/(1-x)^n
= sum_{r=0 to infinity} C(r+n-1,r)*x^r
which is the Taylor expansion of G(x) around 0.

EX: In our last example we choose a famous counting problem,
computing the "partition function", traditionally written p(n).
p(n) is the number of ways n can be written as a sum of
positive integers, where order doesn't matter.
For example
p(1) = 1 since 1 = 1 is the only way to do it
p(2) = 2 since 2 = 2 = 1+1
p(3) = 3 since 3 = 3 = 2+1 = 1+1+1

```
    p(4) = 5 since 4 = 4 = 3+1 = 2+2 = 2+1+1 = 1+1+1+1
    p(5) = 7 since 5 = 5 = 4+1 = 3+2 = 3+1+1
                     = 2+1+1+1 = 2+2+1 = 1+1+1+1+1
    p(10) = 42
    p(100) = 190,569,292
    p(200) ~ 4 * 10^12
It turns out that p(n) has a simple generating function

Theorem (Euler)  1 + sum_{n=1 to infinity} p(n)*x^n
                   = G(x) = prod_{m=1 to infinity} 1/(1-x^m)

Proof:
   Note that
     (1-x)^(-1)   = 1 + x + x^2 + x^3 + ...
   and
     (1-x^m)^(-1) = 1 + x^m + x^2m + x^3m + ...
   The factor (1-x^m)^(-1) in G(x) represents choosing the
   integer m once, twice, 3 times, ... in making up the
   sum for n. For example, to sum up to 5 or less, we'll
   only have 1,2,3,4 or 5 appearing in the sum. Thus
   p(5) is the coefficient of x^5 in
      (1-x)^(-1)*(1-x^2)^(-1)*(1-x^3)^(-1)*(1-x^4)^(-1)*(1-x^5)^(-1)*...
    = (1+x+x^2+x^3+x^4+x^5+...) *
       (1+x^2+x^4+...) *
        (1+x^3+...) *
         (1+x^4+...) *
          (1+x^5+...) * ...
    = 1 + x + 2*x^2 + 3*x^3 + 5*x^4 + 7*x^5 + ...
   The "..." in the above expression represents terms like x^6 or
   higher, which don't contribute to the x^5 or lower terms.
   The same idea works for any n, so the coefficient of x^n in
   prod_{m=1 to infinity} 1/(1-x^m)
     = (1-x)^(-1) * (1-x^2)^(-1) * (1-x^3)^(-1) * ...
     = (1 + x + x^2 + x^3 + ...) * (1 + x^2 + x^4 + x^6 + ... ) *
       (1 + x^3 + x^6 + x^9 + ...) * ...
   is p(n).

The partition function grows rapidly, and many formulas and relationships
have been studied for it (see problems 6.4-51 through 6.4-56 in Rosen.)
In particular, in analogy to Stirling's Formula, it is known that
for n large
     p(n) ~ exp(pi*sqrt(2/3)*sqrt(n))/(4*sqrt(3)*n)
```

```
i.e. the ratio between these two expressions approaches 1 as n grows.
This is hard to prove (Hardy and Ramanujan, 1917), but we will
show a weaker version, namely that p(n) <= exp(K*sqrt(n))
for some constant K.
By using the remarkable fact (without proof) that
    1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 + ... = pi^2/6
we will in fact show that K = pi*sqrt(2/3)

We start with the generating function G(x) for p(n) defined above.
and consider just 0 < x < 1. Then
    G(x) = 1 + p(1)*x + ... + p(n)*x^n
         > p(n)*x^n
Taking logs gets us
    log G(x) > log p(n) + log(x^n)
or
    log p(n) < log G(x) + n*log(1/x)
We estimate the two terms log G(x) and n*log(1/x) separately,
and then choose 0 < x < 1 to minimize the upper bound.
    log G(x) = log prod_{m=1 to infinity} (1-x^m)^(-1)
             = sum_{m=1 to infinity} log (1-x^m)^(-1)
             = - sum_{m=1 to infinity} log (1-x^m)
             = sum_{m=1 to infinity} sum_{n=1 to infinity} x^(mn)/n
                  ... substituting the Taylor expansion
                  ... log(1-z) = -z - z^2/2 - z^3/3 - z^4/4 + ...
                  ... with z = x^m
             = sum_{n=1 to infinity} sum_{m=1 to infinity} x^(mn)/n
                  ... summing in a different order
             = sum_{n=1 to infinity} 1/n * sum_{m=1 to infinity} x^(mn)
                  ... since n is a constant in the inner sum
             = sum_{n=1 to infinity} 1/n * x^n/(1-x^n)
                  ... geometric sum
    We want to get a simple upper bound for the summand that we can sum.
    Note that
         (1-x^n)/(1-x) = 1 + x + x^2 + ... + x^(n-1)
                              ... geometric sum
                       > n * x^(n-1)
                              ... since 0 < x < 1 means x^(n-1) is the
                              ... smallest of the n terms
    so
         x/(n^2*(1-x)) > x^n/(n*(1-x^n))
             ... multiplying both sides by x/(n^2*(1-x^n))
    Thus
```

```
  log G(x) = sum_{n=1 to infinity} 1/n * x^n/(1-x^n)
           < sum_{n=1 to infinity} 1/n^2 * x/(1-x)
           = x/(1-x) * sum_{n=1 to infinity} 1/n^2
Now sum_{n=1 to infinity} 1/n^2 is just a constant.
We'll call it K for now, and substitute K = pi^2/6 at the end.

Thus log p(n) < log G(x) + n log(1/x)
              < K*x/(1-x) + n log(1/x)
              < K*x/(1-x) + n * (1/x -1)
                   ... since log z < z-1 for all z > 1
                   ... to see why, look at the plots of log z and z-1
                   ... or at the Taylor expansion
                   ... log(z) = log(1-(1-z)) = z-1 - (z-1)^2/2 ...
              = K*[x/(1-x)] + n*[(1-x)/x]
              = K*t + n/t
                   ... where t = x/(1-x)
Finally we can minimize this as a function of t (or x).
Differentiating with respect to t and setting the derivative to 0
gets us
    0 = K - n/t^2
or
    t = sqrt(n/K)
or
    log p(n) < 2*sqrt(K)*sqrt(n)
or
    p(n) < exp(2*sqrt(K)*sqrt(n))
as desired. Finally, substituting K = pi^2/6 means
    p(n) < exp( pi * sqrt(2/3) * sqrt(n) )

Here is an intuitive argument that K = 1/1^2 + 1/2^2 + 1/3^2 + ... = pi^2/6.
Consider the polynomial with roots r1,r2,...,rn and constant term = 1:
  (1-x/r1)*(1-x/r2)*(1-x/r3)*...*(1/x/rn)
     = 1 - x*(1/r1 + 1/r2 + 1/r3 + ... + 1/rn) + x^2*(.) + ...
In other words, the coefficient of x is
the negative of the sum of the reciprocals of the roots r1,..,rn.
Now (making a mathematical leap) assume
that this idea work not just for a polynomial, but for a function
like f(x) = sin(sqrt(x))/sqrt(x). Note that f(x) has roots at
x = pi^2 , (2*pi)^2 , (3*pi)^2 , ...
so the sum of reciprocals of the roots is
    1/pi^2 + 1/(2*pi)^2 + 1/(3*pi)^2 + ...
  = (1/pi^2) * ( 1/1^2 + 1/2^2 + 1/3^2 + ... )
```

```
  = (1/pi^2) * K
Also by starting with the Taylor expansion
     sin x = x - x^3/3! + ...
we get
     sin(sqrt(x))/sqrt(x) = 1 - x/3! + ...
so equating the coefficient of x, namely -1/3! = -1/6, and
the negative of the sum of reciprocals of the roots, -(1/pi^2)*K,
we get -1/6 = -(1/pi^2)*K or K = pi^2/6 as desired.
```