CS 1010 Discrete Structures
Lecture 12:
Advanced Counting Techniques

Maria Francis

Feb 11, 2021

Merge Sort

- Algorithms textbooks traditionally claim that sorting is an important, fundamental problem in computer science.
- We will cover one sorting algorithm, Merge Sort. The analysis introduces another kind of recurrence.
- The input is a list of n numbers, and the output is those same numbers in nondecreasing order.
- There are two cases:
  1. If the input is a single number then the algorithm does nothing, because the list is already sorted.
  2. Otherwise,the first half and the second half of the list are each sorted recursively.
  3. Then the two halves are merged to form a sorted list with all n numbers.

Merge Sort

- Suppose we want to sort the list : $10, 7, 23, 5, 2, 8, 6, 9$.
- The first half $(10, 7, 23, 5)$ and the second half $(2, 8, 6, 9)$ are sorted recursively.
- The results of that are $5, 7, 10, 23$ and $2, 6, 8, 9$.
- All that remains is to merge these two lists.
- This is done by repeatedly emitting the smaller of the two leading terms.

Finding a Recurrence for Merge Sort

- Typically a question sorting algorithms asks is What is the maximum number of comparisons used in sorting n items? This is taken as an estimate of the running time.
- For merge sort we express it as an recurrence.
- Let T_n be the maximum number of comparisons used in Merge Sort when the input is list of n numbers.
- For easiness in analysis let us assume that n is a power of 2. This ensures that the input can be divided in half at every stage of the recursion.

Finding a Recurrence for Merge Sort

- If there is only one number in the list, then no comparisons are required, so $T_1 = 0$.
- Otherwise, T_n includes comparisons used in sorting the first half ($\leq T_{n/2}$), in sorting the second half ($\leq T_{n/2}$) and in merging the two halves.
- The number of comparisons in the merging step is at most $n - 1$.
  - This is because at least one number is emitted after each comparison and one more number is emitted at the end when one list becomes empty.
- Therefore, the maximum number of comparisons is given by the recurrence:

$$T_1 = 0$$
$$T_n = 2T_{n/2} + n - 1 \text{ for } n \geq 2.$$

How to Solve it?

- A closed form expression for such a recurrence.
- What about guessing?
- Here are the first few values: $T_1 = 0$, $T_1 = 2T_1 + 2 - 1 = 1$, $T_4 = 2T_2 + 4 - 1 = 5$, $T_8 = 2T_4 + 8 - 1 = 17$ and $T_{16} = 49$.
- Guessing the solution to this recurrence is hard because there is no obvious pattern.

What about Iteration?

$$
\begin{aligned}
T_n &= 2T_{n/2} + n - 1 \\
&= 2(2T_{n/4} + n/2 - 1) + (n-1) \\
&= 4T_{n/4} + (n-4) + (n-1) \\
&= 4(2T_{n/8} + n/4 - 1) + (n-2) + (n-1) \\
&= 8T_{n/8} + (n-4) + (n-2) + (n-1) \\
&= 8(2T_{n/16} + n/8 - 1) + (n-4) + (n-2) + (n-1) \\
&= 16T_{n/16} + (n-8) + (n-4) + (n-2) + (n-1)
\end{aligned}
$$

What about Iteration?

In particular, this formula seems holds:

$$\begin{aligned} T_n &= 2^k T_{n/2^k} + (n - 2^{k-1}) + (n - 2^{k-2}) + \cdots + (n - 2^0) \\ &= 2^k T_{n/2^k} + kn - 2^{k-1} - 2^{k-2} \ldots - 2^0 \\ &= 2^k T_{n/2^k} + kn - 2^k + 1. \end{aligned}$$

We collapsed the geometric sum in the last statement.
$2^0 + 2^1 + \cdots + 2^k = 2^{k+1} - 1$.
Verify with more terms.

What about Iteration?

- Write T_n using early terms with known values.
- If we let $k = log\ n$ then $T_{n/2^k} = T_1$ which we know is 0.

$$\begin{aligned} T_n &= 2^k T_{n/2^k} + kn - 2^k + 1 \\ &= 2^{log\ n} T_{n/2^{log\ n}} + nlog\ n - 2^{log\ n} + 1 \\ &= nT_1 + nlog\ n - n + 1 \end{aligned}$$

- We have a closed-form expression for the maximum number of comparisons. Guessing this formula is fairly complicated.

How to solve general divide and conquer recurrences

- Many recursive algorithms take a problem with a given input and divide it into one or more smaller problems.
- This reduction is successively applied until the solutions of the smaller problems can be found quickly.
- There are many examples in the textbook.
- Please read up on binary search, fast multiplication of integers from the textbook.
- These procedures follow an important algorithmic paradigm known as divide-and-conquer, and are called divide-and-conquer algorithms.
  - They divide a problem into one or more smaller instances of the same problem and they conquer the problem by using the solutions of the smaller problems to find a solution of the original problem.

A formula for f

- Recurrences of the form $f(n) = af(n/b) + g(n)$. Derive estimates of the size of functions.
- Let f satisfy this recurrence whenever n is divisible by b. and $n = b^k$, where $k \in \mathbb{N} \setminus \{0\}$. Then,

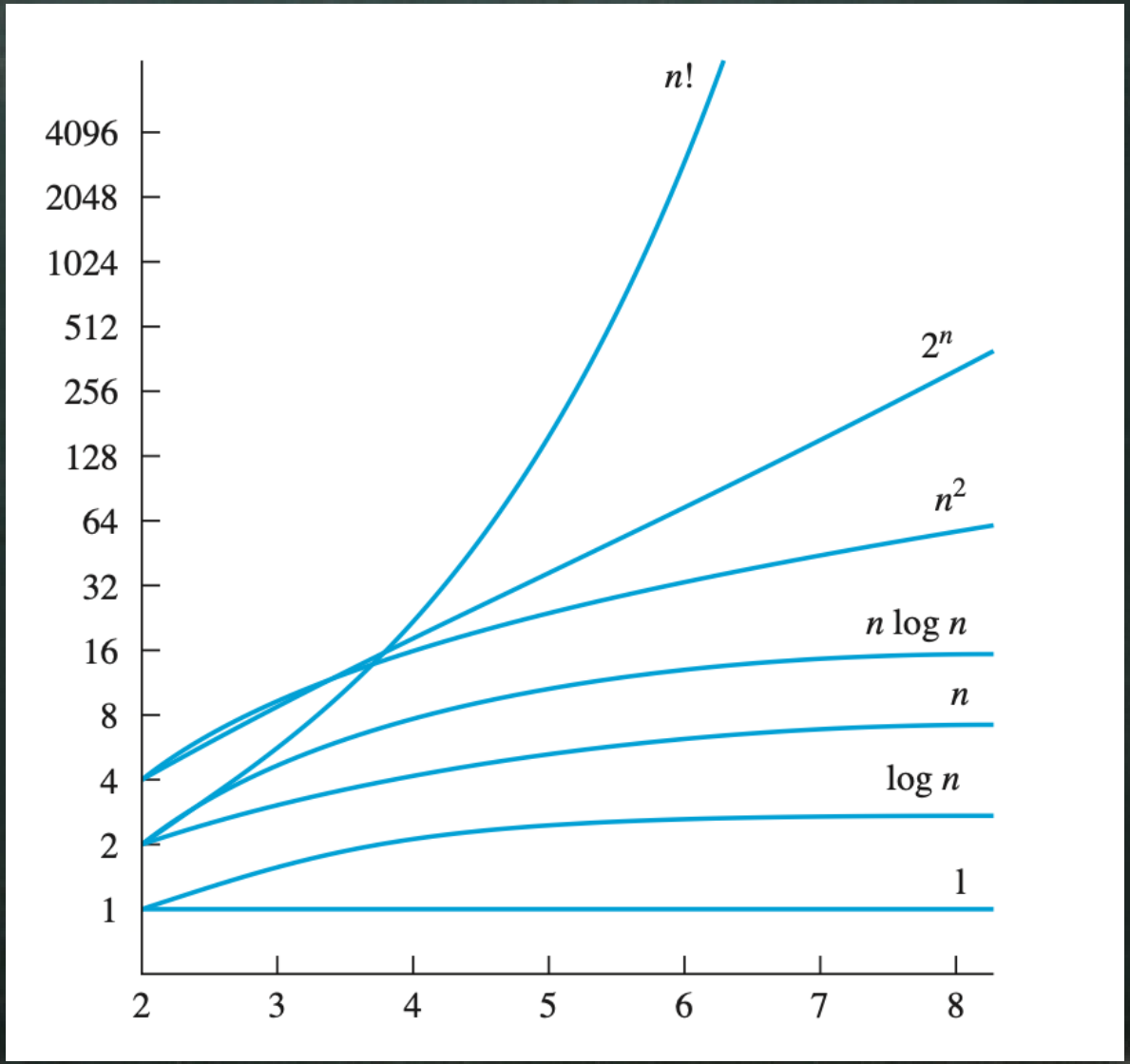
$$\begin{aligned} f(n) &= af(n/b) + g(n) \\ &= a^2 f(n/b^2) + ag(n/b) + g(n) \\ &= a^3 f(n/b^3) + a^2 g(n/b^2) + ag(n/b) + g(n) \\ &\vdots \\ &= a^k f(n/b^k) + \sum_{j=0}^{k-1} a^j g(n/b^j). \end{aligned}$$

- Because $n/b^k = 1$, $f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j g(n/b^j)$.

Estimating f

- We give solutions not in exact form but asymptotically, in big Oh, $\mathcal{O}$.
- What is $\mathcal{O}$? (Check out Section 3, Growth of Functions in the textbook)
  - If f, g be functions defined from $\mathbb{Z}$ or $\mathbb{R}$ to $\mathbb{R}$
  - We say $f(x) = \mathcal{O}(g(x))$ if there are constants C, k s.t. $|f(x)|leqC|g(x)|$ whenever $x > k$.
  - $f(x)is\mathcal{O}(g(x))$ says that $f(x)$ grows slower that some fixed multiple of $g(x)$ as x tends to infinity.
  - Some examples:
    - $f(x) = x^2 + 2x + 1$ is $\mathcal{O}(x^2)$.
    - $f(x) = 7x^2$ is $\mathcal{O}(x^3)$ but not $\mathcal{O}(n)$.
    - If $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, then $f(x)$ is $\mathcal{O}(x^n)$.
    - Also we have $n!$ is $\mathcal{O}(n^n)$ and $log\ n$ is $\mathcal{O}(n)$.

Growth of Functions



Solving Divide-and-Conquer Recurrences

Theorem

Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + c$$

whenever n is divisible by b, where $a \geq 1$, b is an integer greater than 1, and $c \in \mathbb{R}^+$. Then,

$$f(n) \text{ is } \begin{cases} \mathcal{O}(n^{\log_b a}) \text{ if } a \gneq 1 \\ \mathcal{O}(\log_b n) \text{ if } a = 1. \end{cases}$$

Furthermore, when $n = b^k$ and $a \neq 1$ where k is a positive integer, $f(n) = c_1 n^{\log_b a} + c_2$, where $c_1 = f(1) + c/(a-1)$ and $c_2 = -c/(a-1)$.

Proof

- Let $n = b^k$. From the expression for $f(n)$ with $g(n) = c$, we have,

$$f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j c = a^k f(1) + c \sum_{j=0}^{k-1} a^j.$$

- When $a = 1$ we have $f(n) = f(1) + ck$.
- $n = b^k$ implies $k = log_b\ n$ and so $f(n) = f(1) + clog_b\ n$.

Proof

- When n is not a power of b, we have $b^k < n < b^{k+1}$, for a positive integer k.
- Because f is increasing, it follows that $f(n) \leq f(b^{k+1})$ $= f(1) + c(k+1) = (f(1) + c) + ck \leq (f(1) + c) + clog_b n$.
- Therefore in both cases $f(n)$ is $O(logn)$ when $a = 1$.

Proof

- Suppose $a > 1$.
- First assume that $n = b^k$ where k is a positive integer.
- From the formula for the sum of terms of a geometric progression, it follows that

$$\begin{aligned} f(n) &= a^k f(1) + c(a^k - 1)/(a-1) \\ &= a^k[f(1) + c/(a-1)] - c/(a-1) \\ &= C_1 n^{\log_b a} + C_2 \end{aligned}$$

- $a^k = a^{\log_b n} = n^{\log_b a}$ - Exercise!,
- where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.

Proof

- Now suppose that n is not a power of b.
- Then $b^k < n < b^{k+1}$, where k is a nonnegative integer.
- Because f is increasing,

$$\begin{aligned} f(n) &\leq f(b^{k+1}) = C_1 a^{k+1} + C_2 \\ &\leq (C_1 a) a^{log_b n} + C_2 \\ &= (C_1 a) n^{log_b a} + C_2 \end{aligned}$$

because $k \leq log_b n < k + 1$. Hence, we have $f(n)$ is $O(n^{log_b\ a})$.

An Example

- $f(n) = 5f(n/2) + 3$ and $f(1) = 7$. Find $f(2^k)$ and estimate f.
- $a = 5, b = 2, c = 3$ and for $n = 2^k$ we have,

$$\begin{aligned} f(n) &= a^k[f(1) + c/(a-1)] + [-c/(a-1)] \\ &= 5^k[7 + (3/4)] - 3/4 \\ &= 5^k(31/4) - 3/4. \end{aligned}$$

- If $f(n)$ is increasing, we have from the above theorem, $f(n)$ is $\mathcal{O}(n^{log_b\ a}) = \mathcal{O}(n^{log 5})$.

Master Theorem

A more general result.

Theorem

Let f be an increasing function that satisfies $f(n) = af(n/b) + cn^d$ whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer ≥ 1 and c and d are real numbers with c positive and d nonnegative. Then,

$$f(n) \text{ is } \begin{cases} O(n^d) \text{ if } a < b^d \\ O(n^d \log n) \text{ if } a = b^d \\ O(n^{\log_b a}) \text{ if } a > b^d. \end{cases}$$

Proof is part of the assignment question.

Analysing Mergesort

- For mergesort, we have $T_n = 2T_{n/2} + n - 1$.
- It is less than M_n where $M_n = 2M_{n/2} + n$.
- By master's theorem we find that $M_n = \mathcal{O}(n log_2\ n)$ and that is what we got from our analysis.

Two issues with divide and conquer recurrences

- The asymptotic solution to a divide-and-conquer recurrence is independent of the boundary conditions.
  - Intuitively, if the bottom-level operation in a recursive algorithm takes, say, twice as long, then the overall running time will at most double.
  - We are looking for asympototic solutions since exact solutions maybe difficult to get or too complex to be useful.
- Second issue: dividing a problem of size n may create subproblems of non-integer size.
  - Fortunately, the asymptotic solution to a divide and conquer recurrenceis unaffected by floors and ceilings. I.e.the solution is not changed by replacing a term $T(b_i n)$ with either $T(\lceil b_i n \rceil)$ or $T(\lfloor b_i n \rfloor)$.
  - So leaving floors and ceilings makes sense in many contexts.

Overview of the recurrences

- Fibonacci : $T_n = T_{n-1} + T_{n-2}$ Solution: $T_n \approx (1.618)^{n+1}/\sqrt{5}$
- Towers of Hanoi : $T_n = 2T_{n-1} + 1$ Solution: $T_n \approx 2^n$
- Merge Sort : $T_n = 2T_{n/2} + n - 1$ Solution: $T_n \approx nlogn$

- Notice that the recurrence equations for Towers of Hanoi and Merge Sort are somewhat similar, but the solutions are radically different.
- Merge sort of $n = 64$ items takes a few hundred comparisons while moving $n = 64$ disks takes more than 10^{19} steps!

Overview of the recurrences

- In the Towers of Hanoi, we broke a problem of size n into two subproblem of size $n-1$ (which is large), but needed only 1 additional step (which is small).
- In Merge Sort, we divided the problem of size n into two subproblems of size $n/2$ (which is small), but needed $(n-1)$ additional steps (which is large).
- Yet, Merge Sort is faster by a mile!
- This suggests that generating smaller subproblems is far more important to algorithmic speed than reducing the additional steps per recursive call.

Generating Functions

- Generating functions are used to represent sequences efficiently by coding the terms of a sequence as coefficients of powers of a variable x in a formal power series.
- Generating functions can be used to solve many types of counting problems.
- It can be used to solve recurrence relations
  - First translate a recurrence relation for the terms of a sequence into an equation involving a generating function.
  - This equation can then be solved to find a closed form for the generating function.
  - From this closed form, the coefficients of the power series for the generating function can be found, solving the original recurrence relation.
- Generating functions can also be used to prove combinatorial identities.

Generating Functions

Definition

The generating function for the sequence $a_0, a_1, \ldots, a_k, \ldots$, of real numbers is the infinite series,

$$G(x) = a_0 + a_1 x + \cdots + a_k x^k + \cdots = \sum_{k=0}^{\infty} a_k x^k.$$

Sometimes also called ordinary generating function.

- OGF for the sequence $\{a_k\}$ with $a_k = 3$ is $\sum_{k=0}^{\infty} 3x^k$.
- $a_k = k + 1$ is $\sum_{k=0}^{\infty} (k+1)x^k$
- $a_k = 2^k$ is $\sum_{k=0}^{\infty} 2^k x^k$.

Generating Functions

- We can define generating functions for finite sequences: extend into an infinite sequence by setting $a_{n+1} = 0$, $a_{n+2} = 0$ and so on.
- The OGF is a polynomial of degree n, $G(x) = a_0 + a_1 x + \cdots + a_n x^n$.
- Example:
  - Let m be a positive integer. Let $a_k = C(m, k)$ for for $k = 0, 1, 2 \ldots, m$.
  - What is the generating function for the sequence $a_0, a_1, \ldots, a_m$?
  - $G(x) = C(m, 0) + C(m, 1)x + C(m, 2)x^2 + \cdots + C(m, m)x^m$.
  - From the binomial theorem we have $G(x) = (1 + x)^m$.

Power Series

- Generating Functions are usually considered to be formal power series.
- I.e. questions about the convergence of these series are ignored.
- However, to apply some results from calculus, it is sometimes important to consider for which x the power series converges.
- Generally we will not be concerned with questions of convergence or the uniqueness of power series in our discussions.

Power Series

- We look at some facts now. There will be a discussion of these results in calculus textbooks as well.
- The function $f(x) = 1/(1-x)$ is the generating function of the seqeunce $1, 1, 1, \ldots$ because

$$1/(1-x) = 1 + x + x^2 + \cdots$$

for $|x| < 1$.

- The function $f(x) = 1/(1-ax)$ is the generating function of the sequence $1, a, a^2, a^3, \ldots,$ because

$$1/(1-ax) = 1 + ax + a^2x^2 + \cdots$$

for $|ax| < 1$ or equivalently, for $|x| < 1/|a|$ for $a \neq 0$.

Adding and Multiplying OGFs

- Let $f(x) = \sum_{k=0}^{\infty} a_k x^k$ and $g(x) = \sum_{k=0}^{\infty} b_k x^k$. Then,

$$f(x)+g(x) = \sum_{k=0}^{\infty}(a_k+b_k)x^k \text{ and } f(x)g(x) = \sum_{k=0}^{\infty}(\sum_{j=0}^{k} a_j b_{k-j})x^k.$$

- Theorem 1 is valid only for power series that converge in an interval.
- However, the theory of OGFs is not limited to such series.
- In the case of series that do not converge, the statements in Theorem 1 can be taken as definitions.

How to apply these theorems?

- Let $f(x) = 1/(1-x)^2$.
- Find $a_0, a_1, a_2, \ldots$ in the expansion $f(x)$.
- We have from the result, $1/(1-x) = 1 + x + x^2 + x^3 + \cdots$.
- The product theorem can be used to get,

$$1/(1-x)^2 = \sum_{k=0}^{\infty} (\sum_{j=0}^{k} 1) x^k = \sum_{k=0}^{\infty} (k+1) x^k.$$

- This result also can be derived by differentiation.

Extended Binomial Coefficient

- To use OGFs to solve many important counting problems, we will need to apply the binomial theorem for exponents that are not positive integers.
- First, Let u be a real number and k a nonnegative integer. Then theextended binomial coefficeint $\binom{u}{k}$ is defined by

$$= \{u(u-1)\cdots(u-k+1)/k! \text{ if } k > 0$$
$$= 1 \text{ if } k = 0$$

Extended Binomial Coefficient

- Find the values $C(-2, 3)$
  - $u = -2$ and $k = 3$ gives us $C(-2, 3) = \frac{(-2)(-3)(-4)}{3!} = -4$.
- and $C(1/2, 3)$.
  - $u = 1/2$ and $k = 3$ gives us $C(1/2, 3) = \frac{(1/2)(1/2-1)(1/2-2)}{3!} = (1/2)(-1/2)(-3/2)/6 = 1/16$.

Extended Binomial Coefficient

- The top parameter is a negative integer.

$$\begin{aligned}
\binom{-n}{r} &= \frac{(-n)(-n-1)\cdots(-n-r+1)}{r!} \quad \text{by definition} \\
&= \frac{(-1)^r n(n+1)\cdots(n+r-1)}{r!} \\
&= \frac{(-1)^r (n+r-1)(n+r-2)\cdots n}{r!} \\
&= \frac{(-1)^r (n+r-1)!}{r!(n-1)!} \\
&= (-1)^r C(n+r-1, r)
\end{aligned}$$

Extended Binomial Theorem

Theorem

Let x be a real number with $|x| < 1$ and let u be a real number. Then,

$$(1+x)^u = \sum_{k=0}^{\infty} C(u,k)x^k.$$

It can be proved using the theory of Maclaurin series.

When u is a positive integer, the extended binomial theorem reduces to the binomial theorem presented before if we consider $C(u,k) = 0$ if $k > u$.

Standard OGFs

- Find the generating functions for $(1+x)^{-n}$ and $(1-x)^{-n}$, where n is a positive integer.
- By the extended binomial theorem it follows that

$$(1+x)^{-n} = \sum_{k=0}^{\infty} C(-n,k)x^k.$$

- By using the simple formula we derived for negative values,

$$(1+x)^{-n} = \sum_{k=0}^{\infty} (-1)^k C(n+k-1,k)x^k.$$

- Replacing x by $-x$ we find that

$$(1-x)^{-n} = \sum_{k=0}^{\infty} C(n+k-1,k)x^k.$$

Standard OGFs

- Please check the table below for OGFs that arise frequently.
- You should know these core formulae and learn how to derive other formulae from these ones.

$G(x)$	a_k
$(1+x)^n = \sum_{k=0}^{n} C(n,k)x^k = 1 + C(n,1)x + C(n,2)x^2 + \cdots + x^n$	$C(n,k)$
$(1+ax)^n = \sum_{k=0}^{n} C(n,k)a^k x^k = 1 + C(n,1)ax + C(n,2)a^2x^2 + \cdots + a^n x^n$	$C(n,k)a^k$
$(1+x^r)^n = \sum_{k=0}^{n} C(n,k)x^{rk} = 1 + C(n,1)x^r + C(n,2)x^{2r} + \cdots + x^{rn}$	$C(n,k/r)$ if $r \mid k$; 0 otherwise
$\frac{1-x^{n+1}}{1-x} = \sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$	1 if $k \le n$; 0 otherwise
$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + \cdots$	1
$\frac{1}{1-ax} = \sum_{k=0}^{\infty} a^k x^k = 1 + ax + a^2x^2 + \cdots$	a^k
$\frac{1}{1-x^r} = \sum_{k=0}^{\infty} x^{rk} = 1 + x^r + x^{2r} + \cdots$	1 if $r \mid k$; 0 otherwise

Standard OGFs

$\frac{1}{(1-x)^2} = \sum_{k=0}^{\infty}(k+1)x^k = 1+2x+3x^2+\cdots$	$k+1$
$\frac{1}{(1-x)^n} = \sum_{k=0}^{\infty} C(n+k-1,k)x^k$ $= 1+C(n,1)x+C(n+1,2)x^2+\cdots$	$C(n+k-1,k) = C(n+k-1,n-1)$
$\frac{1}{(1+x)^n} = \sum_{k=0}^{\infty} C(n+k-1,k)(-1)^k x^k$ $= 1-C(n,1)x+C(n+1,2)x^2-\cdots$	$(-1)^k C(n+k-1,k) = (-1)^k C(n+k-1,n-1)$
$\frac{1}{(1-ax)^n} = \sum_{k=0}^{\infty} C(n+k-1,k)a^k x^k$ $= 1+C(n,1)ax+C(n+1,2)a^2x^2+\cdots$	$C(n+k-1,k)a^k = C(n+k-1,n-1)a^k$
$e^x = \sum_{k=0}^{\infty}\frac{x^k}{k!} = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots$	$1/k!$
$\ln(1+x) = \sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}x^k = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\cdots$	$(-1)^{k+1}/k$

OGFs for solving recurrences

- Solve the recurrence relation $a_k = 3a_{k-1}$ for $k = 1, 2, 3, \ldots$ and initial condition $a_0 = 2$.
- Let $G(x)$ be the OGF for the sequence $\{a_k\}$ that is $G(x) = \sum_{k=0}^{\infty} a_k x^k$.
- $xG(x) = \sum_{k=0}^{\infty} a_k x^{k+1} = \sum_{k=1}^{\infty} a_{k-1} x^k$.
- Using the recurrence relation, we see that

$$\begin{aligned} G(x) - 3xG(x) &= \sum_{k=0}^{\infty} a_k x^k - 3 \sum_{k=1}^{\infty} a_{k-1} x^k \\ &= a_0 + \sum_{k=1}^{\infty} (a_k - 3a_{k-1}) x^k \\ &= 2. \end{aligned}$$

because $a_0 = 2$ and $a_k = 3a_{k-1}$.

OGFs for solving recurrences

- Thus,

$$G(x) - 3xG(x) = (1 - 3x)G(x) = 2.$$

- Solving for $G(x)$ shows that $G(x) = 2/(1 - 3x)$.
- Using the identity $1/(1 - ax) = \sum_{k=0}^{\infty} a^k x^k$, we have

$$G(x) = 2\sum_{k=0}^{\infty} 3^k x^k = \sum_{k=0}^{\infty} 2 \cdot 3^k x^k.$$

- Thus, $a_k = 2 \cdot 3^k$.

OGFs for solving recurrences

- Solve the recurrence, $a_{n+1} = 2a_n + (-1)^n$ with initial condition $a_0 = 0$.
- Let $G(x) = \sum_{k=0}^{\infty} a_k x^k$.
- We have $a_{k+1} - 2a_k x^k - (-1)x^k = 0$, i.e.

$$\frac{1}{x}\sum_{k=1}^{\infty} a_k x^k - 2\sum_{k=0}^{\infty} a_k x^k - \sum_{k=0}^{\infty}(-1)^k x^k = 0.$$

-

$$\frac{1}{x}(G(x) - a_0) - 2G(x) - \frac{1}{x+1} = 0.$$

$$G(x) = \frac{a_0}{1-2x} + \frac{1}{3}\frac{1}{1-2x} - \frac{1}{3}\frac{1}{1+x.}$$

OGFs for solving recurrences

- Equating $a_0 = 0$, we get $G(x) = \frac{1}{3}\frac{1}{1-2x} - \frac{1}{3}\frac{1}{1+x.}$.
- There are standard bits from which we get closed formula:

$$a_n = \frac{1}{3}2^n - \frac{1}{3}(-1)^n.$$

OGFs for counting problems

- There are 50 students in the International Mathematical Olympiad (IMO) training programme. 6 of them are to be selected to represent India in the IMO. How many ways are there to select 6 students?
- Each student is either selected or not selected.
- Hence each student contributes a factor of $1 + x$ to the gf.
- 1 (i.e. x^0) refers to the case when the student is not selected (student occupies 0 place)
- While the term x (i.e. x^1) refers to the case when the student is selected.

OGFs for counting problems

- Since there are 50 students the gf is $G(x) = (1 + x)^{50}$.
- Since 6 students are to be selected, the answer is the coefficient of x^6 in $G(x)$ which according to BT is $C(50, 6)$
- Basically one can use generating functions to find the number of k-combinations of a set with n elements.

OGFs for counting problems

- How is the generating function formed? By a sequence of $'+'$s and $'\times'$s corresponding to a sequence of *ORs* and *ANDs*
- For each student he is either selected OR not selected, so each student contributes a factor of $1 + x$.
- Now we need to do the same for the 1st student AND the 2nd student AND the 3rd student AND so on.
- That is why we multiply 50 copies of $1 + x$ together to form the generating function.

OGFs for counting problems

- There are 30 identical souvenirs, to be distributed among the 50 IMO trainees, and each trainee may get more than one souvenir. How many ways are there to distribute the 30 souvenirs among the 50 trainees?
- Each student may get 0 OR 1 OR 2 OR ... souvenirs, thus contributing a factor of $1 + x + x^2 + \cdots$.
- Since there are 50 students

$$G(x) = (1 + x + x^2 + \cdots)^{50} = \left(\frac{1}{1-x}\right)^{50} = (1-x)^{-50}.$$

- As there are 30 souvenirs the answer is the coefficient of x^{30} in $G(x)$.

OGFs for counting problems

- According to the extended BT, is equal to $C(-50, 30) = C(79, 30)$.
- You can argue that the term $1 + x + x^2 + \cdots$ should be replaced by $1 + x + x^2 + \cdots + x^{30}$ because each student may get ≤ 30 souvenirs. It turns out that this modification will not affect the final outcome.
- The general version of this example is Example 14 in Rosen.

OGFs for counting problems

- How many integer solutions to the equation $a + b + c = 6$ satisfy $-1 \leq a \leq 2$ and $1 \leq b, c \leq 4$?
- Since $-1 \leq a \leq 2$ the variable a contributes a term $x^{-1} + x^0 + x^1 + x^2$ to the gf.
- Similarly b and c contributes a term $x^1 + x^2 + x^3 + x^4$.
- Hence the OGF is

$$\begin{aligned} G(x) &= (x^{-1} + x^0 + x^1 + x^2)(x^1 + x^2 + x^3 + x^4)^2 \\ &= x(1 + x + x^2 + x^3)^3 \\ &= x(\frac{1 - x^4}{1 - x})^3 \\ &= x(1 - 3x^4 + 3x^8 - x^{12})(1 - x)^{-3} \\ &= (x - 3x^5 + 3x^9 - x^{13})(1 - x)^{-3} \end{aligned}$$

OGFs for counting problems

- The answer is the coefficient of x^6 in $G(x)$.
- To get an x^6 term we can multiply x with the x^5 term in $(1-x)^{-3}$ as well as multiply $-3x^5$ with the x term in $(1-x)^{-3}$.
- By the extended BT, the answer $C(7,5) - 3C(3,1) = 12$.

Proving Identities using OGFs

- S.T. $\sum_{k=0}^{n} c(n,k)^2 = C(2n,n)$.
- By the BT $C(2n,n)$ is the coefficient of x^n in $(1+x)^{2n}$.
- However we also have,

$$\begin{aligned}(1+x)^{2n} &= [(1+x)^n]^2 \\ &= [C(n,0) + C(n,1)x + C(n,2)x^2 + \cdots + C(n,n)x^n]^2.\end{aligned}$$

- The coefficient of x^n in this expression is $C(n,0)C(n,n) + C(n,1)C(n,n-1) + C(n,2)C(n,n-2) + \cdots + C(n,n)C(n,0)$.
- This equals $\sum_{k=0}^{n} C(n,k)^2$ because $C(n,n-k) = C(n,k)$.
- Because both $C(2n,n)$ and $\sum_{k=0}^{n} C(n,k)^2$ represent the coefficient of x^n in $(1+x)^{2n}$ they must be equal.

Proving Identities using OGFs

- BT tells us that $(1+z)^r$ is the GF for the sequence $(C(r,0), C(r,1), C(r,2), \ldots,)$: $(1+z)^r = \sum_{k\geq 0} C(r,k)z^k$.
- Similarly, $(1+z)^s = \sum_{k\geq 0} C(s,k)z^k$.
- Multiplying these together, we get another gf: $(1+z)^r(1+z)^s = (1+z)^{r+s}$.
- Equating coefficients of z^n on both sides:

$$\sum_{k=0}^{n} C(r,k)C(s,n-k) = C(r+s,n).$$

Vandermode's identity!