Discrete Mathematics and Its Applications

Lecture 4: Advanced Counting Techniques: Divide-and-Conquer and Generating Functions

MING GAO

DaSE@ ECNU
(for course related communications)
mgao@dase.ecnu.edu.cn

Nov. 13, 2018

Outline

Divide-and-Conquer strategy

The divide-and-conquer strategy solves a problem P by:

1. Breaking P into subproblems that are themselves smaller instances of the same type of problem (Divide step);
2. Recursively solving these subproblems (Solve step);
3. Appropriately combining their answers (Conquer step).

Divide-and-Conquer strategy

The divide-and-conquer strategy solves a problem P by:

1. Breaking P into subproblems that are themselves smaller instances of the same type of problem (Divide step);
2. Recursively solving these subproblems (Solve step);
3. Appropriately combining their answers (Conquer step).

The real work to implement Divide-and-Conquer strategy is done piecemeal, where the key works lay in three different places:

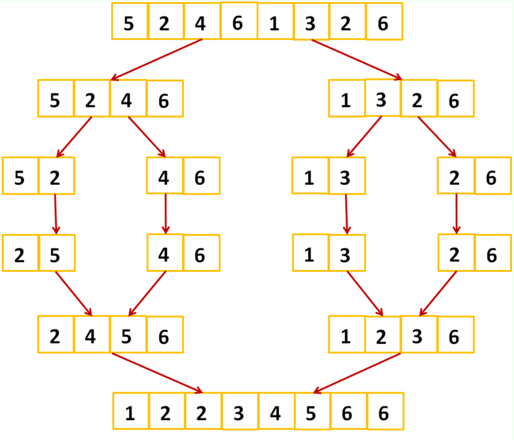
1. How to partition problem into subproblems;
2. At the very tail end of the recursion, how to solve the smallest subproblems outright;
3. How to glue together the partial answers.

Mergesort example

The classic divide and conquer recurrence is Merge-sort's $T(n) = 2T(n/2) + O(n)$, which divides the data into equal-sized halves and spends linear time merging the halves after they are sorted.

Mergesort example

The classic divide and conquer recurrence is Merge-sort's $T(n) = 2T(n/2) + O(n)$, which divides the data into equal-sized halves and spends linear time merging the halves after they are sorted.



Definition of DCR^2

Suppose that a recursive algorithm divides a problem of size n into a subproblems, where each subproblem is of size n/b (for simplicity, assume that n is a multiple of b; in reality, the smaller problems are often of size equal to the nearest integers either less than or equal to, or greater than or equal to, n/b). Also, suppose that a total of $g(n)$ extra operations are required in the conquer step of the algorithm to combine the solutions of the subproblems into a solution of the original problem. Then, if $f(n)$ represents the number of operations required to solve the problem of size n, it follows that f satisfies the recurrence relation

$$f(n) = af(n/b) + g(n).$$

This is called a **divide-and-conquer recurrence relation** (shorted in DCR^2).

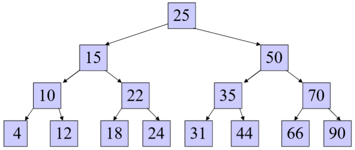
Binary search

Problem: We have an ordered sequence of numbers, $a_1, a_2, \cdots, a_n$. Given x, decide whether x is in the sequence or not. For example, $x = 17$ in the right tree.

Binary search

Problem: We have an ordered sequence of numbers, $a_1, a_2, \cdots, a_n$. Given x, decide whether x is in the sequence or not. For example, $x = 17$ in the right tree.

Solution: This binary search algorithm reduces the search for an element in a search sequence of size n to the binary search for this element in a search sequence of size $n/2$, when n is even.

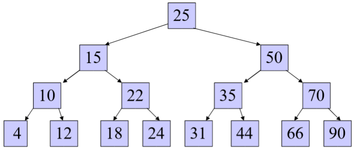


procedure *binary search* (x: integer, $a_1, a_2, \ldots, a_n$)
$i := 1$ {i is left endpoint of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ **then** $i := m + 1$
    else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*

Binary search

Problem: We have an ordered sequence of numbers, $a_1, a_2, \cdots, a_n$. Given x, decide whether x is in the sequence or not. For example, $x = 17$ in the right tree.

Solution: This binary search algorithm reduces the search for an element in a search sequence of size n to the binary search for this element in a search sequence of size $n/2$, when n is even.



procedure *binary search* (x: integer, $a_1, a_2, \ldots, a_r$)
$i := 1${i is left endpoint of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i+j)/2 \rfloor$
    if $x > a_m$ **then** $i := m + 1$
    else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*

Hence, the problem of size n has been reduced to one problem of size $n/2$. The DCR^2 for binary search is $T(n) = T(n/2) + 2$.

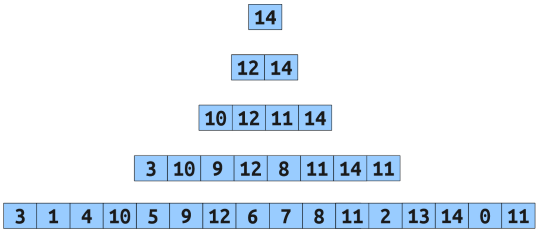
Find the maximum and minimum of a sequence

Let $T(n)$ be the total number of comparisons needed to find the maximum and minimum elements of the sequence with n elements.

Find the maximum and minimum of a sequence

Let $T(n)$ be the total number of comparisons needed to find the maximum and minimum elements of the sequence with n elements.

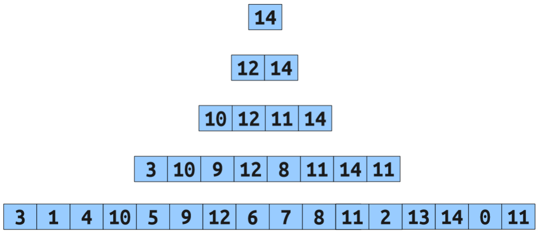
We have shown that a problem of size n can be reduced into two problems of size $n/2$, when n is even.



Find the maximum and minimum of a sequence

Let $T(n)$ be the total number of comparisons needed to find the maximum and minimum elements of the sequence with n elements.

We have shown that a problem of size n can be reduced into two problems of size $n/2$, when n is even.

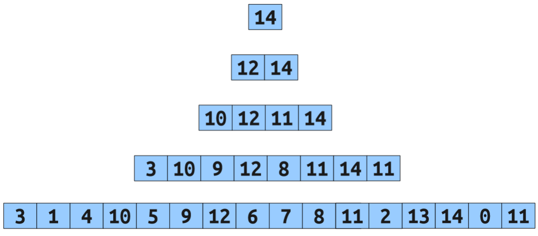


Using two comparisons, One compares the maxima of the two sequences, and the other compares the minima of the two sequences.

Find the maximum and minimum of a sequence

Let $T(n)$ be the total number of comparisons needed to find the maximum and minimum elements of the sequence with n elements.

We have shown that a problem of size n can be reduced into two problems of size $n/2$, when n is even.



Using two comparisons, One compares the maxima of the two sequences, and the other compares the minima of the two sequences. This gives the recurrence relation $T(n) = 2T(n/2) + 2$, when n is even.

Multiplication for integers

Suppose x and y are two n-bit integers, and assume for convenience that n is a power of 2.

Multiplication for integers

Suppose x and y are two n-bit integers, and assume for convenience that n is a power of 2.
As a first step toward multiplying x and y, we split each of them into their left and right halves, which are $n/2$ bits long, i.e.,

$$x = (\underbrace{a_{2n-1}a_{2n-2}\cdots a_n}_{x_L}\underbrace{a_{n-1}a_{n-2}\cdots a_0}_{x_R})_2,$$

$$y = (\underbrace{b_{2n-1}b_{2n-2}\cdots b_n}_{y_L}\underbrace{b_{n-1}b_{n-2}\cdots b_0}_{y_R})_2.$$

That is $x = 2^n x_L + x_R$ and $y = 2^n y_L + y_R$.
Thus, we have $xy = 2^{2n}x_L y_L + 2^n(x_L y_R + x_R y_L) + x_R y_R$. The significant operations are the four $n/2$-bit multiplications; these we can handle by four recursive calls, then evaluates the preceding expression in $O(n)$ time.

Multiplication for integers

Suppose x and y are two n-bit integers, and assume for convenience that n is a power of 2.

As a first step toward multiplying x and y, we split each of them into their left and right halves, which are $n/2$ bits long, i.e.,

$$x = (\underbrace{a_{2n-1}a_{2n-2}\cdots a_n}_{x_L}\underbrace{a_{n-1}a_{n-2}\cdots a_0}_{x_R})_2,$$

$$y = (\underbrace{b_{2n-1}b_{2n-2}\cdots b_n}_{y_L}\underbrace{b_{n-1}b_{n-2}\cdots b_0}_{y_R})_2.$$

That is $x = 2^n x_L + x_R$ and $y = 2^n y_L + y_R$.

Thus, we have $xy = 2^{2n}x_L y_L + 2^n(x_L y_R + x_R y_L) + x_R y_R$. The significant operations are the four $n/2$-bit multiplications; these we can handle by four recursive calls, then evaluates the preceding expression in $O(n)$ time. Thus, recurrence relation is $T(n) = 4T(n/2) + O(n)$, when n is even.

Using Gauss's trick

Three multiplications $x_L y_L$, $(x_L + x_R)(y_L + y_R)$, and $x_R y_R$ are suffice since

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R.$$

Using Gauss's trick

Three multiplications $x_L y_L$, $(x_L + x_R)(y_L + y_R)$, and $x_R y_R$ are suffice since

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R.$$

Thus, recurrence relation is $T(n) = 3T(n/2) + O(n)$, when n is even.

- The algorithm's recursive calls form a tree structure;
- At each successive level the subproblems get halved in size;
- At the $\log_2 n$th level, the subproblems get down to size 1, and so the recursion ends, i.e., the height is $\log_2 n$;
- The branching factor is 3: each problem recursively produces three smaller ones, with the result that at depth k in the tree there are 3^k subproblems, each of size $n/2^k$.

Using Gauss's trick

For each subproblem, the total time therefore spends at depth k in the tree is $3^k \times O(\frac{n}{2^k}) = (\frac{3}{2})^k \times O(n)$.

Using Gauss's trick

For each subproblem, the total time therefore spends at depth k in the tree is $3^k \times O(\frac{n}{2^k}) = (\frac{3}{2})^k \times O(n)$.

- At the very top level, when $k = 0$, we need $O(n)$;
- At the bottom, when $k = \log_2 n$, it is

$$O(3^{\log_2 n}) = O(n^{\log_2 3});$$

- Between these two endpoints, the work done increases geometrically from $O(n)$ to $O(n^{\log_2 3})$, by a factor of $\frac{3}{2}$ per level.

The sum of any increasing geometric series is, within a constant factor, simply the last term of the series. Therefore the overall running time is

$$O(n^{\log_2 3}) \approx O(n^{1.59}).$$

Solution for DCR^2

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 is a positive real number. Then

$$f(n) \text{ is } \begin{cases} O(n^{\log_b a}), & \text{if } a > 1; \\ O(\log n), & \text{if } a = 1. \end{cases}$$

Solution for DCR^2

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 is a positive real number. Then

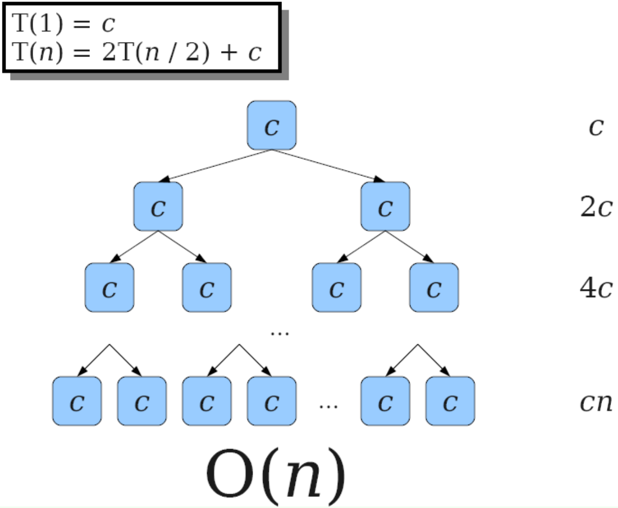
$$f(n) \text{ is } \begin{cases} O(n^{\log_b a}), & \text{if } a > 1; \\ O(\log n), & \text{if } a = 1. \end{cases}$$

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 $C_2 = -c/(a-1)$.

Example



Proof I

Suppose that f satisfies this recurrence relation whenever n is divisible by b. Let $n = b^k$, where k is a positive integer.

Proof I

Suppose that f satisfies this recurrence relation whenever n is divisible by b. Let $n = b^k$, where k is a positive integer. Then

$$\begin{aligned} f(n) &= af(n/b) + c \\ &= a^2 f(n/b^2) + ac + c \\ &= a^3 f(n/b^3) + a^2 c + ac + c \\ &= \cdots \\ &= a^k f(n/b^k) + \sum_{i=0}^{k-1} a^i c \end{aligned}$$

Proof I

Suppose that f satisfies this recurrence relation whenever n is divisible by b. Let $n = b^k$, where k is a positive integer. Then

$$\begin{aligned} f(n) &= af(n/b) + c \\ &= a^2 f(n/b^2) + ac + c \\ &= a^3 f(n/b^3) + a^2 c + ac + c \\ &= \cdots \\ &= a^k f(n/b^k) + \sum_{i=0}^{k-1} a^i c \end{aligned}$$

Because $n/b^k = 1$, it follows that

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

Proof II

Case I $a = 1$: We have $f(n) = a^k f(1) + \sum_{i=0}^{k-1} c$.

Proof II

Case I $a = 1$: We have $f(n) = a^k f(1) + \sum_{i=0}^{k-1} c$.
Because $n = b^k$, we have $k = \log_b n$. Hence,

$$f(n) = f(1) + c \log_b n.$$

Proof II

Case I $a = 1$: We have $f(n) = a^k f(1) + \sum_{i=0}^{k-1} c$.
Because $n = b^k$, we have $k = \log_b n$. Hence,

$$f(n) = f(1) + c \log_b n.$$

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 $f(n) \leq f(b^{k+1}) = f(1) + c(k+1) = (f(1) + c) + ck \leq (f(1) + c) + c \log_b n$.

Proof II

Case I $a = 1$: We have $f(n) = a^k f(1) + \sum_{i=0}^{k-1} c$.
Because $n = b^k$, we have $k = \log_b n$. Hence,

$$f(n) = f(1) + c \log_b n.$$

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 $f(n) \leq f(b^{k+1}) = f(1) + c(k+1) = (f(1) + c) + ck \leq (f(1) + c) + c \log_b n$.
Therefore, in both cases, $f(n)$ is $O(\log n)$ when $a = 1$.

Proof II

Case I $a = 1$**:** We have $f(n) = a^k f(1) + \sum_{i=0}^{k-1} c$.
Because $n = b^k$, we have $k = \log_b n$. Hence,

$$f(n) = f(1) + c \log_b n.$$

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 $f(n) \leq f(b^{k+1}) = f(1) + c(k+1) = (f(1) + c) + ck \leq (f(1) + c) + c \log_b n$.
Therefore, in both cases, $f(n)$ is $O(\log n)$ when $a = 1$.
Case II $a > 1$: First assume that $n = b^k$, where k is a positive integer. It follows that

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

Proof III

Because

$$\begin{aligned} a^k = a^{\log_b n} = b^{\log_b a^{\log_b n}} \\ = b^{(\log_b n)\cdot(\log_b a)} = b^{\log_b n^{\log_b a}} \\ = n^{\log_b a}, \end{aligned}$$

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.

Proof III

Because

$$\begin{aligned} a^k = a^{\log_b n} = b^{\log_b a^{\log_b n}} \\ = b^{(\log_b n)\cdot(\log_b a)} = b^{\log_b n^{\log_b a}} \\ = n^{\log_b a}, \end{aligned}$$

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.
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

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

Proof III

Because

$$\begin{aligned} a^k = a^{\log_b n} = b^{\log_b a^{\log_b n}} \\ = b^{(\log_b n)\cdot(\log_b a)} = b^{\log_b n^{\log_b a}} \\ = n^{\log_b a}, \end{aligned}$$

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.
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

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

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

Example I

Question: Let $f(n) = 5f(n/2) + 3$ and $f(1) = 7$. Find $f(2^k)$, where k is a positive integer. Also, estimate $f(n)$ if f is an increasing function.

Solution: From the proof of the theorem, with $a = 5$, $b = 2$, and $c = 3$, we see that if $n = 2^k$, then

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

Also, if $f(n)$ is increasing, the above theorem shows that $f(n)$ is $O(n^{\log_b a}) = O(n^{\log 5})$.

Example II

Binary search

Question: Give a big-O estimate for the number of comparisons used by a binary search.

Solution: We have known that $f(n) = f(n/2) + 2$ when n is even, where f is the number of comparisons required to perform a binary search on a sequence of size n.

Search maximum and minimum elements

Question: Give a big-O estimate for the number of comparisons used to locate the maximum and minimum elements in a sequence.

Solution: We have already known that $f(n) = 2f(n/2) + 2$ with $a = 2$, $b = 2$, and $c = 2$.

Hence, it follows that $f(n)$ is $O(n^{\log 2}) = O(n)$.

Example II

Binary search

Question: Give a big-O estimate for the number of comparisons used by a binary search.

Solution: We have known that $f(n) = f(n/2) + 2$ when n is even, where f is the number of comparisons required to perform a binary search on a sequence of size n.

Note that $a = 1$. Hence, from the above theorem, it follows that $f(n)$ is $O(\log n)$.

Example II

Binary search

Question: Give a big-O estimate for the number of comparisons used by a binary search.

Solution: We have known that $f(n) = f(n/2) + 2$ when n is even, where f is the number of comparisons required to perform a binary search on a sequence of size n.

Note that $a = 1$. Hence, from the above theorem, it follows that $f(n)$ is $O(\log n)$.

Search maximum and minimum elements

Question: Give a big-O estimate for the number of comparisons used to locate the maximum and minimum elements in a sequence.

Solution: We have already known that $f(n) = 2f(n/2) + 2$ with $a = 2$, $b = 2$, and $c = 2$.

Example II

Binary search

Question: Give a big-O estimate for the number of comparisons used by a binary search.

Solution: We have known that $f(n) = f(n/2) + 2$ when n is even, where f is the number of comparisons required to perform a binary search on a sequence of size n.

Note that $a = 1$. Hence, from the above theorem, it follows that $f(n)$ is $O(\log n)$.

Search maximum and minimum elements

Question: Give a big-O estimate for the number of comparisons used to locate the maximum and minimum elements in a sequence.

Solution: We have already known that $f(n) = 2f(n/2) + 2$ with $a = 2$, $b = 2$, and $c = 2$.

Hence, it follows that $f(n)$ is $O(n^{\log 2}) = O(n)$.

Master theorem

Theorem

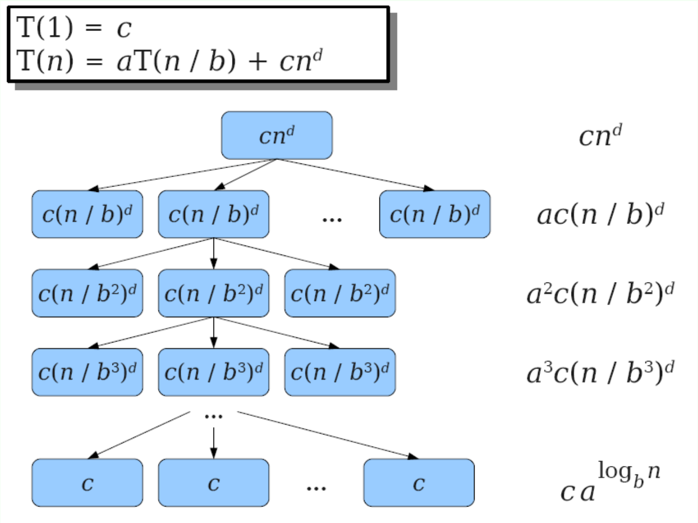
Let T be an increasing function that satisfies the recurrence relation

$$T(n) = aT(n/b) + cn^d$$

whenever n is divisible by b, where $a \geq 1$, b is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$T(n) \text{ is } \begin{cases} O(n^d), & \text{if } a < b^d; \\ O(n^d \log n), & \text{if } a = b^d; \\ O(n^{\log_b a}), & \text{if } a > b^d; \end{cases}$$

Master theorem Cont'd



Proof of Master theorem

At internal level k of the tree, the work done is

$$a^k c(n/b^k)^d$$

Proof of Master theorem

At internal level k of the tree, the work done is

$$a^k c(n/b^k)^d = cn^d(a/b^d)^k$$

Therefore

$$\begin{aligned} T(n) &= ca^{\log_b n} + \sum_{k=0}^{\log_b n-1} cn^d(\frac{a}{b^d})^k \\ &= ca^{\log_b n} + cn^d \sum_{k=0}^{\log_b n-1} (\frac{a}{b^d})^k \\ &= cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n-1} (\frac{a}{b^d})^k \end{aligned}$$

Case I proof

Case I

If $a/b^d = 1$, that is $a = b^d$ and $d = \log_b a$.

$$\begin{aligned} T(n) &= cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n - 1} (\frac{a}{b^d})^k \\ &= cn^d + cn^d \sum_{k=0}^{\log_b n - 1} 1 = cn^d + cn^d \log_b n \\ &= O(n^d \log n) \end{aligned}$$

Case II proof

Case II

If $a/b^d < 1$, that is $a < b^d$ and $d > \log_b a$.

$$\begin{aligned}
T(n) &= cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n-1} (\frac{a}{b^d})^k \\
&< cn^d + cn^d \sum_{k=0}^{\log_b n-1} (\frac{a}{b^d})^k \\
&< cn^d + cn^d \sum_{k=0}^{\infty} (\frac{a}{b^d})^k = cn^d (1 + \frac{1}{1 - a/b^d}) \\
&= O(n^d)
\end{aligned}$$

Case III proof

Case III

If $a/b^d > 1$, that is $a > b^d$ and $d < \log_b a$.

$$T(n) = cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n-1} (\frac{a}{b^d})^k = cn^d + cn^d \frac{(a/b^d)^{\log_b n} - 1}{(a/b^d) - 1}$$

$$< cn^{\log_b a} + cn^d (a/b^d)^{\log_b n} \frac{1}{(a/b^d) - 1}$$

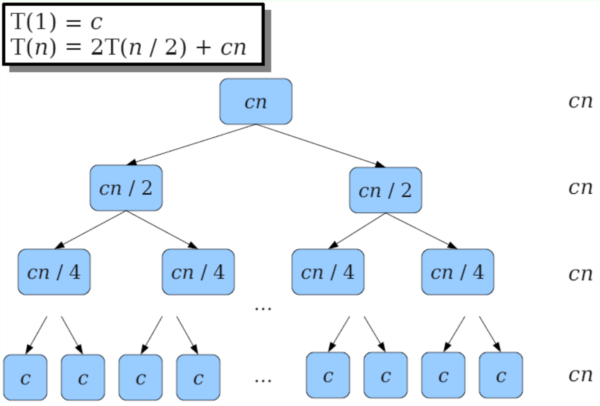
$$= cn^{\log_b a} + cn^d (a/b^d)^{\log_b n} \Theta(1)$$

$$= cn^{\log_b a} + cn^d (a^{\log_b n}/b^{d \log_b n}) \Theta(1)$$

$$= cn^{\log_b a} + cn^d (n^{\log_b a}/n^d) \Theta(1) = cn^{\log_b a} + cn^{\log_b a} \Theta(1)$$

$$= O(n^{\log_b a})$$

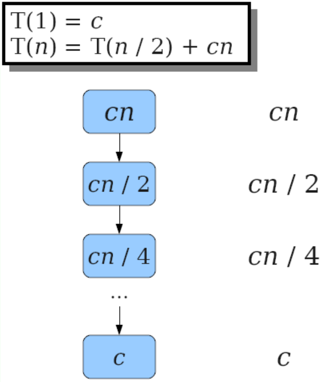
Example I of Master theorem



where $a = 2, b = 2$ and $d = 1$, that is $a = b^d$. Therefore, we have

$$T(n) = O(n \log n).$$

Example II of Master theorem



where $a = 1$, $b = 2$ and $d = 1$, that is $a < b^d$. Therefore, we have

$$T(n) = O(n^d) = O(n).$$

Example III

Question: Let # comparisons used by the Mergesort to sort a list of n elements be less than $M(n)$, where $M(n) = 2M(n/2) + n$.
Solution: By the master theorem, we find that $M(n)$ is $O(n \log n)$.

Example III

Question: Let # comparisons used by the Mergesort to sort a list of n elements be less than $M(n)$, where $M(n) = 2M(n/2) + n$.
Solution: By the master theorem, we find that $M(n)$ is $O(n \log n)$.

Question: Let $f(n)$ be # bit operations for multiplying two n-bit integers, where $f(n) = 3f(n/2) + Cn$.
Solution: Hence, from the master theorem, it follows that $f(n)$ is $O(n^{\log 3})$. Note that $\log 3 \approx 1.6$.

Example III

Question: Let # comparisons used by the Mergesort to sort a list of n elements be less than $M(n)$, where $M(n) = 2M(n/2) + n$.
Solution: By the master theorem, we find that $M(n)$ is $O(n \log n)$.

Question: Let $f(n)$ be # bit operations for multiplying two n-bit integers, where $f(n) = 3f(n/2) + Cn$.
Solution: Hence, from the master theorem, it follows that $f(n)$ is $O(n^{\log 3})$. Note that $\log 3 \approx 1.6$.

Question: Let $f(n)$ be # multiplications and additions required to multiply two $n \times n$ matrices, where $f(n) = 7f(n/2) + 15n^2/4$.
Solution: Hence, from the master theorem, it follows that $f(n)$ is $O(n^{\log 7})$. Note that $\log 7 \approx 2.8$.

Generating functions

Definition

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

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

Generating functions

Definition

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

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

Examples

$\{a_k\}$	$g(x)$
$a_k = 3$	$\sum_{k=0}^{\infty} 3x^k$
$a_k = k+1$	$\sum_{k=0}^{\infty}(k+1)x^k$
$a_k = 2^k$	$\sum_{k=0}^{\infty} 2^k x^k$
$a_k = 1(k=0,1,\cdots,5)$	$\sum_{k=0}^{5} x^k = \frac{x^6-1}{x-1}$
$a_k = C(m,k)(k=0,1,\cdots,m)$	$\sum_{k=0}^{m} C(m,k)x^k = (1+x)^m$

Operations of generating functions

Theorem

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

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

Operations of generating functions

Theorem

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

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

Examples of power series

$\{a_k\}$	$g(x)$
$a_k = 1$	$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ for $\lvert x\rvert < 1$
$a_k = a^k$	$\sum_{k=0}^{\infty} a^k x^k = \frac{1}{1-ax}$ for $\lvert ax\rvert < 1$
$a_k = k+1$	$\sum_{k=0}^{\infty} \big(\sum_{j=0}^{k} 1\big)x^k = \frac{1}{(1-x)^2}$

Extended binomial coefficient

Definition

Let u be a real number and k a nonnegative integer. Then the extended binomial coefficient $\binom{u}{k}$ is defined by

$$\binom{u}{k} = \begin{cases} \frac{u(u-1)\cdots(u-k+1)}{k!}, & \text{if } k > 0; \\ 1, & \text{if } k = 0. \end{cases}$$

Extended binomial coefficient

Definition

Let u be a real number and k a nonnegative integer. Then the extended binomial coefficient $\binom{u}{k}$ is defined by

$$\binom{u}{k} = \begin{cases} \frac{u(u-1)\cdots(u-k+1)}{k!}, & \text{if } k > 0; \\ 1, & \text{if } k = 0. \end{cases}$$

Examples

Question: Find the values of $\binom{-2}{3}$ and $\binom{1/2}{3}$.
Solution:

$$\binom{-2}{3} = \frac{(-2)(-3)(-4)}{3!} = -4;$$

$$\binom{1/2}{3} = \frac{(1/2)(1/2-1)(1/2-2)}{3!} = \frac{1}{16}.$$

Corollary

Let n and r are two positive integers, the extended binomial coefficient can be expressed as

$$\binom{-n}{r} = (-1)^r \binom{n+r-1}{r}.$$

Proof:

$$\begin{aligned}\binom{-n}{r} &= \frac{-n(-n-1)\cdots(-n-r+1)}{r!} \\ &= \frac{(-1)^r n(n+1)\cdots(n+r-1)}{r!} \\ &= \frac{(-1)^r (n+r-1)!}{r!(n-1)!} \\ &= (-1)^r \binom{n+r-1}{r}.\end{aligned}$$

The extended binomial theorem

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

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

The extended binomial theorem

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

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

Examples

Find the generating functions for $(1+x)^{-n}$ and $(1-x)^{-n}$ for $n \in \mathcal{Z}^+$.

Solution:

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

Replacing x by $-x$, we find that

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

Running example I

Question: Find the number of solutions of $e_1 + e_2 + e_3 = 17$, where e_1, e_2, and e_3 are nonnegative integers with $2 \leq e_1 \leq 5, 3 \leq e_2 \leq 6$, and $4 \leq e_3 \leq 7$.

Solution: The number of solutions with the indicated constraints is the coefficient of x^{17} in the expansion of

$$(x^2 + x^3 + x^4 + x^5)(x^3 + x^4 + x^5 + x^6)(x^4 + x^5 + x^6 + x^7).$$

Running example I

Question: Find the number of solutions of $e_1 + e_2 + e_3 = 17$, where e_1, e_2, and e_3 are nonnegative integers with $2 \leq e_1 \leq 5, 3 \leq e_2 \leq 6$, and $4 \leq e_3 \leq 7$.

Solution: The number of solutions with the indicated constraints is the coefficient of x^{17} in the expansion of

$$(x^2 + x^3 + x^4 + x^5)(x^3 + x^4 + x^5 + x^6)(x^4 + x^5 + x^6 + x^7).$$

Note that

Way	**# cases**
1: $x^5 \cdot x^6 \cdot x^6$	1
2: $x^5 \cdot x^5 \cdot x^7$	1
3: $x^4 \cdot x^6 \cdot x^7$	1

It is not hard to see that the coefficient of x^{17} in this product is 3. Hence, there are three solutions.

Running example II

Question: In how many different ways can eight identical cookies be distributed among three distinct children if each child receives at least two cookies and no more than four cookies?

Running example II

Question: In how many different ways can eight identical cookies be distributed among three distinct children if each child receives at least two cookies and no more than four cookies?

Solution: Because each child receives at least two but no more than four cookies, for each child there is a factor equal to $(x^2 + x^3 + x^4)$ in the generating function for the sequence $\{c_n\}$, where c_n is the number of ways to distribute n cookies.

Running example II

Question: In how many different ways can eight identical cookies be distributed among three distinct children if each child receives at least two cookies and no more than four cookies?

Solution: Because each child receives at least two but no more than four cookies, for each child there is a factor equal to $(x^2 + x^3 + x^4)$ in the generating function for the sequence $\{c_n\}$, where c_n is the number of ways to distribute n cookies. Because there are three children, this generating function is

$$(x^2 + x^3 + x^4)^3.$$

We need the coefficient of x^8 in this product.

Running example II

Question: In how many different ways can eight identical cookies be distributed among three distinct children if each child receives at least two cookies and no more than four cookies?

Solution: Because each child receives at least two but no more than four cookies, for each child there is a factor equal to $(x^2 + x^3 + x^4)$ in the generating function for the sequence $\{c_n\}$, where c_n is the number of ways to distribute n cookies. Because there are three children, this generating function is

$$(x^2 + x^3 + x^4)^3.$$

We need the coefficient of x^8 in this product.

Computation shows that this coefficient equals 6. Hence, there are six ways to distribute the cookies so that each child receives at least two, but no more than four, cookies.

Running example III

Question: Please determine # ways to insert tokens worth 1,2, and \$5 into a vending machine that costs r dollars not matter the orders.

Running example III

Question: Please determine # ways to insert tokens worth 1,2, and \$5 into a vending machine that costs r dollars not matter the orders.
Solution: The number of solutions with the indicated constraints is the coefficient of x^r in the expansion of

$$(1+x+x^2+x^3+\cdots)(1+x^2+x^4+x^6+\cdots)(1+x^5+x^{10}+x^{15}+\cdots).$$

Running example III

Question: Please determine # ways to insert tokens worth 1,2, and \$5 into a vending machine that costs r dollars not matter the orders.
Solution: The number of solutions with the indicated constraints is the coefficient of x^r in the expansion of

$$(1+x+x^2+x^3+\cdots)(1+x^2+x^4+x^6+\cdots)(1+x^5+x^{10}+x^{15}+\cdots).$$

When the order in which the tokens are inserted matters, the number of ways to insert exactly n tokens to produce a total of r dollars is the coefficient of x^r in $(x+x^2+x^5)^n$,

Running example III

Question: Please determine # ways to insert tokens worth 1,2, and \$5 into a vending machine that costs r dollars not matter the orders.
Solution: The number of solutions with the indicated constraints is the coefficient of x^r in the expansion of

$$(1+x+x^2+x^3+\cdots)(1+x^2+x^4+x^6+\cdots)(1+x^5+x^{10}+x^{15}+\cdots).$$

When the order in which the tokens are inserted matters, the number of ways to insert exactly n tokens to produce a total of r dollars is the coefficient of x^r in $(x+x^2+x^5)^n$,
Because any number of tokens may be inserted, the number of ways to produce r dollars using \$1, \$2, or \$5 tokens, when the order in which the tokens are inserted matters, is the coefficient of x^r in

$$1+(x+x^2+x^5)+(x+x^2+x^5)^2+\cdots=\frac{1}{1-(x+x^2+x^5)}.$$

Running example IV

Question: Find # k-combinations of a set with n elements. Assume that the binomial theorem has already been established.

Running example IV

Question: Find # k-combinations of a set with n elements. Assume that the binomial theorem has already been established.

Solution: Each of the n elements in the set contributes term $(1+x)$ to the generating function $f(x) = \sum_{k=0}^{n} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of k-combinations of a set with n elements. Hence, $f(x) = (1+x)^n$

Running example IV

Question: Find # k-combinations of a set with n elements. Assume that the binomial theorem has already been established.

Solution: Each of the n elements in the set contributes term $(1 + x)$ to the generating function $f(x) = \sum_{k=0}^{n} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of k-combinations of a set with n elements. Hence, $f(x) = (1 + x)^n$

But by the binomial theorem, we have

$$f(x) = \sum_{k=0}^{n} \binom{n}{k} x^k.$$

Running example IV

Question: Find # k-combinations of a set with n elements. Assume that the binomial theorem has already been established.

Solution: Each of the n elements in the set contributes term $(1+x)$ to the generating function $f(x) = \sum_{k=0}^{n} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of k-combinations of a set with n elements. Hence, $f(x) = (1+x)^n$

But by the binomial theorem, we have

$$f(x) = \sum_{k=0}^{n} \binom{n}{k} x^k.$$

Hence, $C(n,k)$ is the number of k-combinations of a set with n elements.

Running example V

Question: Find # r-combinations from a set with n elements when repetition of elements is allowed.

Running example V

Question: Find # r-combinations from a set with n elements when repetition of elements is allowed.

Solution: Each of the n elements in the set contributes term $(1 + x + x^2 + \cdots)$ to the generating function $f(x) = \sum_{k=0}^{\infty} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents # r-combinations from a set with n elements with repetition. Hence, $f(x) = (1 + x + x^2 + \cdots)^n$.

Running example V

Question: Find # r-combinations from a set with n elements when repetition of elements is allowed.

Solution: Each of the n elements in the set contributes term $(1 + x + x^2 + \cdots)$ to the generating function $f(x) = \sum_{k=0}^{\infty} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents # r-combinations from a set with n elements with repetition. Hence, $f(x) = (1 + x + x^2 + \cdots)^n$.

As long as $|x| < 1$, we have $1 + x + x^2 + \cdots = \frac{1}{1-x}$. Thus $f(x) = (1 - x)^{-n}$.

Running example V

Question: Find # r-combinations from a set with n elements when repetition of elements is allowed.

Solution: Each of the n elements in the set contributes term $(1 + x + x^2 + \cdots)$ to the generating function $f(x) = \sum_{k=0}^{\infty} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents # r-combinations from a set with n elements with repetition. Hence, $f(x) = (1 + x + x^2 + \cdots)^n$.

As long as $|x| < 1$, we have $1 + x + x^2 + \cdots = \frac{1}{1-x}$. Thus $f(x) = (1-x)^{-n}$.

Applying the extended binomial theorem, it follows that

$$(1-x)^{-n} = \sum_{r=0}^{\infty} \binom{-n}{r} (-x)^r.$$

Thus, we have $\binom{-n}{r}(-1)^r = (-1)^r C(n+r-1, r)(-1)^r = \binom{n+r-1}{r}$.

Running example VI

Question: Find # ways to select r objects of n different kinds if we must select at least one object of each kind.

Running example VI

Question: Find # ways to select r objects of n different kinds if we must select at least one object of each kind.
Solution: Each of the n elements in the set contributes term $(x + x^2 + \cdots)$ to the generating function $f(x) = \sum_{k=0}^{\infty} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of r objects of n different kinds with at least one object of each kind. Hence, $f(x) = (x + x^2 + \cdots)^n$.

Running example VI

Question: Find # ways to select r objects of n different kinds if we must select at least one object of each kind.

Solution: Each of the n elements in the set contributes term $(x + x^2 + \cdots)$ to the generating function $f(x) = \sum_{k=0}^{\infty} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of r objects of n different kinds with at least one object of each kind.

Hence, $f(x) = (x + x^2 + \cdots)^n$.

As long as $|x| < 1$, we have $f(x) = x^n(1 + x + x^2 + \cdots)^n = \frac{x^n}{(1-x)^n}$.

Running example VI

Question: Find # ways to select r objects of n different kinds if we must select at least one object of each kind.

Solution: Each of the n elements in the set contributes term $(x + x^2 + \cdots)$ to the generating function $f(x) = \sum_{k=0}^{\infty} a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of r objects of n different kinds with at least one object of each kind. Hence, $f(x) = (x + x^2 + \cdots)^n$.

As long as $|x| < 1$, we have $f(x) = x^n(1 + x + x^2 + \cdots)^n = \frac{x^n}{(1-x)^n}$.

Applying the extended binomial theorem, it follows that

$$f(x) = x^n \sum_{r=0}^{\infty} \binom{-n}{r} (-x)^r = \sum_{t=n}^{\infty} \binom{t-1}{t-n} x^t.$$

Hence, there are $C(r-1, r-n)$ ways to select r objects of n different kinds if we must select at least one object of each kind.

Running example VI

Question: Solve recurrence relation $a_k = 3a_{k-1}$ for $k = 1, 2, 3, \cdots$ and initial condition $a_0 = 2$.

Running example VI

Question: Solve recurrence relation $a_k = 3a_{k-1}$ for $k = 1, 2, 3, \cdots$ and initial condition $a_0 = 2$.

Solution: Let $f(x)$ be the generating function for the sequence $\{a_k\}$, that is, $f(x) = \sum_{k=0}^{\infty} a_k x^k$. First note that

$$xf(x) = \sum_{k=0}^{\infty} a_k x^{k+1} = \sum_{k=1}^{\infty} a_{k-1} x^k.$$

Running example VI

Question: Solve recurrence relation $a_k = 3a_{k-1}$ for $k = 1, 2, 3, \cdots$ and initial condition $a_0 = 2$.

Solution: Let $f(x)$ be the generating function for the sequence $\{a_k\}$, that is, $f(x) = \sum_{k=0}^{\infty} a_k x^k$. First note that

$$xf(x) = \sum_{k=0}^{\infty} a_k x^{k+1} = \sum_{k=1}^{\infty} a_{k-1} x^k.$$

Using the recurrence relation, we see that

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

Running example VI Cont'd

Thus,

$$f(x) - 3xf(x) = (1 - 3x)f(x) = 2,$$

i.e.,

$$f(x) = 2/(1 - 3x).$$

Running example VI Cont'd

Thus,

$$f(x) - 3xf(x) = (1 - 3x)f(x) = 2,$$

i.e.,

$$f(x) = 2/(1 - 3x).$$

Using the identity

$$1/(1 - ax) = \sum_{k=0}^{\infty} a^k x^k,$$

we have

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

Consequently,

$$a_k = 2 \cdot 3^k.$$

Running example VII

Question: Suppose that a valid codeword is an n-digit number in decimal notation containing an even number of 0s. Let an denote the number of valid codewords of length n. Note that $a_1 = 9$, and the recurrence relation is

$$a_n = 8a_{n-1} + 10^{n-1}.$$

Running example VII

Question: Suppose that a valid codeword is an n-digit number in decimal notation containing an even number of 0s. Let an denote the number of valid codewords of length n. Note that $a_1 = 9$, and the recurrence relation is

$$a_n = 8a_{n-1} + 10^{n-1}.$$

Solution: We extend this sequence by setting $a_0 = 1$, then

$$\begin{aligned} f(x) - 1 &= \sum_{n=1}^{\infty} a_n x^n = \sum_{n=1}^{\infty} (8a_{n-1}x^n + 10^{n-1}x^n) \\ &= 8x \sum_{n=1}^{\infty} a_{n-1}x^{n-1} + x \sum_{n=1}^{\infty} 10^{n-1}x^{n-1} \\ &= 8x \sum_{n=0}^{\infty} a_n x^n + x \sum_{n=0}^{\infty} 10^n x^n = 8xf(x) + \frac{x}{1-10x}. \end{aligned}$$

Running example VII Cont'd

That is,

$$f(x) - 1 = 8xf(x) + \frac{x}{1-10x}$$

i.e.,

$$f(x) = \frac{1-9x}{(1-8x)(1-10x)} = \frac{1}{2}\left(\frac{1}{1-8x} + \frac{1}{1-10x}\right).$$

Running example VII Cont'd

That is,

$$f(x) - 1 = 8xf(x) + \frac{x}{1-10x}$$

i.e.,

$$f(x) = \frac{1-9x}{(1-8x)(1-10x)} = \frac{1}{2}\left(\frac{1}{1-8x} + \frac{1}{1-10x}\right).$$

Furthermore,

$$f(x) = \sum_{k=0}^{\infty} \frac{1}{2}(8^k + 10^k)x^k.$$

Consequently,

$$a_n = \frac{1}{2}(8^n + 10^n).$$

Running example VIII

Statement: Let n is a positive integer, using generating functions to show that

$$\sum_{k=0}^{n} C(n,k)^2 = C(2n,n).$$

Running example VIII

Statement: Let n is a positive integer, using generating functions to show that

$$\sum_{k=0}^{n} C(n,k)^2 = C(2n,n).$$

Proof: First note that by the binomial theorem $C(2n,n)$ is the coefficient of x^n in $(1+x)^{2n}$.

Running example VIII

Statement: Let n is a positive integer, using generating functions to show that

$$\sum_{k=0}^{n} C(n,k)^2 = C(2n,n).$$

Proof: First note that by the binomial theorem $C(2n,n)$ is the coefficient of x^n in $(1+x)^{2n}$. However, we also have

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

Running example VIII

Statement: Let n is a positive integer, using generating functions to show that

$$\sum_{k=0}^{n} C(n,k)^2 = C(2n,n).$$

Proof: First note that by the binomial theorem $C(2n,n)$ 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 $\sum_{k=0}^{n} C(n,k)C(n,n-k) = \sum_{k=0}^{n} C(n,k)^2$.

Running example VIII

Statement: Let n is a positive integer, using generating functions to show that

$$\sum_{k=0}^{n} C(n,k)^2 = C(2n,n).$$

Proof: First note that by the binomial theorem $C(2n,n)$ 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 $\sum_{k=0}^{n} C(n,k)C(n,n-k) = \sum_{k=0}^{n} C(n,k)^2$.
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.

Take-aways

- Divide-and-Conquer Recurrence Relations
  - Definition and Examples of DCR^2
  - Master Theorem
- Generating Functions
  - Useful Facts About Power Series
  - Extended Binomial Coefficient
  - Counting Problems and Generating Functions
  - Using Generating Functions to Solve Recurrence Relations
  - Proving Identities via Generating Functions