Recurrences

We've already seen recursive definitions, for example $C(n,k)$.

Similarly, let $f(0) = 1$

$f(n) = n \cdot f(n-1)$ for $n \in \mathbb{N}$. $\longrightarrow$ a "recursive" definition.

Then $f(n) = n \cdot (n-1) \cdots 1 = n!$

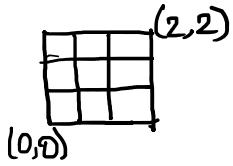
Consider the following slightly more involved example.

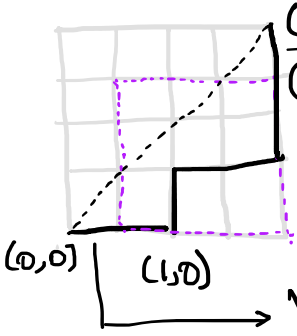
How many paths are there in the coordinate grid from $(0,0)$ to (n,n) if

- At each step, you can move one unit right or one unit up.
- You can never cross over to the $y > x$ region.

We can construct such a path as:

- Pick the minimum $k > 0$ s.t. (k,k) is reached.





You have to move up just before reaching (k,k).

You have to move right in the beginning

Let us call such a path a Catalan path.

Then $(0,0) \longrightarrow\!\!\!\!\rightarrow (n,n)$ (↳ Catalan path) $\equiv$ $(0,0) \rightarrow (1,0) \longrightarrow\!\!\!\!\rightarrow (k,k-1) \rightarrow (k,k) \longrightarrow\!\!\!\!\rightarrow (n,n)$

((k,k) is the first time we touch the diagonal)

$\therefore$ Denoting C_n as the number of Catalan paths from $(0,0)$ to (n,n),

$$C_n = \sum_{k=1}^{n} C_{k-1} C_{n-k} \quad (\text{and } C_0 = 1)$$

1, 1, 2, 5, 14, 42, 132, ...

The Fibonacci sequence is defined by

$F_0 = 0$

$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$ for $n > 1$.

Q. How many ternary strings of length n are there which do not have "00" as a substring?

Set up a recurrence.

$A(n)$ = no. of such strings which start with 0.

$B(n)$ = no. of such strings not starting with 0.

Now, $A(n) = B(n-1)$ (the second digit cannot be 0)

$B(n) = 2(A(n-1) + B(n-1))$

(the first digit is 1 or 2 and the second digit can be anything)

Also, $A(0) = 0$, $B(0) = 1$ (the empty string)

$B(n) = 2B(n-1) + 2B(n-2)$, $B(0) = 1$ and $B(1) = 2$

The required count is $B(n) + B(n-1)$.

Recursion and Induction

When we want to prove something about recursively defined expressions, we typically use induction.

Claim F_{3n} is even for $n \geq 0$.

Proof. $F_0 = 0$ and $F_3 = 2$ are even.

Suppose F_{3n} is even for $0 \leq n \leq k-1$

Then $F(3k) = F(3k-1) + F(3k-2) = \underbrace{2 \cdot F(3k-2) + F(3k-3)}_{\text{even.}}$.

In fact, it even holds that F_n is even iff $n = 3k$ for some $k \geq 0$.

Ideally, we would want to get a closed form expression for recursively defined quantities.

For example $f(0) = 0$ and $f(n) = n + f(n-1)$ for $n > 0$.

Then $f(n) = \frac{n \cdot (n-1)}{2}$

Sometimes, we just give it a name - like $n!$, C_n and F_n.

F_n and C_n even have closed form expressions.

Consider the class of recurrence relations

$f(0) = c$

$f(1) = d$

$f(n) = a \cdot f(n-1) + b f(n-2)$ for $n \geq 2$

(F_n has $a=1, b=1, c=0, d=1$)

Suppose

$x^2 - ax - b$ has two distinct complex solutions x and y.

We claim that there exist p, q such that for all n,

$$f(n) = px^n + yq^n.$$

Let $p = \frac{d - cy}{x - y}$ and $q = \frac{d - cx}{x - y}$ (for the base cases $n=0$ and $n=1$)

For all $k \geq 2$,

suppose it holds for all $0 \leq k \leq n-1$.

then

$$\begin{aligned} f(k) &= a(px^{k-1} + qy^{k-1}) + b(px^{k-2} + qy^{k-2}) \\ &= px^{k-2}(ax + b) + qy^{k-2}(ay + b) \\ &= px^k + qy^k. \end{aligned}$$

If $x^2 - ax - b$ has only one solution, say $x \neq 0$. Then there are p, q such that for all n,

$$f(n) = (p + qn)x^n.$$

This can be checked similarly.

Later, we shall study the above method more in detail.

(Generating Functions)

Unrolling a recursion

It is often helpful to "unroll" a recursion to see what is going on.
For example,

$$T(0) = 0, \quad T(n) = T(n-1) + n^2$$

Then $$T(n) = \sum_{k=0}^{n} k^2$$

Another example,

$$T(1) = 0$$
$$T(n) = T(\lfloor n/2 \rfloor) + 1$$

If $n = 2^k$,

$$T(n) = 1 + T(2^{k-1})$$
$$= 1 + 1 + \cdots + 1 = k$$

In general,

$$T(n) = \lfloor \log_2 n \rfloor \text{ (or just } \log n)$$

A rooted tree is a tree with a special node designated as the root.

We define u to be v's parent if the path from root to v contains the edge $\{u, v\}$.

(Any non-root node has a unique parent)

If v has parent u, then v is called a child of u.

A leaf is then defined as a node with no child.

Any non-leaf node is called an internal node.

u is said to be an ancestor of v and v a descendant of u if the root-v path passes through u.

Note that the "is ancestor" and "is descendant" relations define partial orders on the set of a nodes

The subtree rooted at u is the subtree containing all of u's descendants.

The depth of a node is its distance from the root.
The height of a rooted tree is the maximum depth.

Level i is the set of nodes at depth i.
Note that edges are only present between adjacent levels.

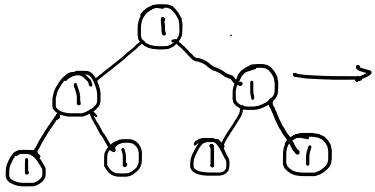
We define the arity of a rooted tree by the maximum number of children possessed by a node.
We call a tree m-ary if the arity is $\leq m$.
(binary / ternary tree)

In a full m-ary tree, every internal node has exactly m children.
A complete and full m-ary tree is a full m-ary tree with all leaves at the same level.
→ Such a tree has m^i nodes at level i.
→ There are $\left(\frac{m^{h+1}-1}{m-1}\right)$ nodes in all. (where h is the height)

→ If $m=2$, there are 2^h leaves and 2^h-1 internal nodes.

Now, consider the recurrence (see: Towers of Hanoi)

$$M(1)=1 \qquad M(n)=2M(n-1)+1$$



→ create two copies of $M(k-1)$ and add 1.

$M(n)$ is just the number of nodes in a complete and full binary tree of height $n-1$, which is 2^n-1.

Generating Functions

A generating function is an alternate representation of an infinite sequence.

The sequence $f(0), f(1), \cdots$ is represented as

$$G_f(x) = f(0) + xf(1) + x^2 f(2) + \cdots$$

That is, for $f: \mathbb{N}_0 \to \mathbb{R}$,

$$G_f(x) = \sum_{k \geq 0} f(k) x^k$$

Generating functions often have a succinct representation.

For example, if $f(k) = a^k$ for some a, then

$$G_f(x) = \frac{1}{1-ax} \quad \text{for sufficiently small } |x|.$$

This enables us to manipulate and analyze sequences more easily.

For $a \in \mathbb{R}$, define $\binom{a}{k} = \frac{a(a-1)\cdots(a-k+1)}{k!}$ for $k \in \mathbb{N}$

and $\binom{a}{0} = 1$

Theo. For $|x|<1$ and $a \in \mathbb{R}$,

$$(1+x)^a = \sum_{k \geq 0} \binom{a}{k} x^k. \qquad \text{[Extended Binomial Theorem]}$$

We do not prove the above.

However, it is useful when we want to find a closed form for f given certain G_f.

For example, if $G_f = \frac{1}{1-x}$,

$$\binom{-1}{k} = (-1)^k \Rightarrow (1-x)^{-1} = \sum_{k \geq 0} x^k \Rightarrow f(k) = 1$$

(we have $(-1)^k \cdot (-1)^k = 1$)

If $G_f = \frac{1}{(1-x)^2}$, $f(k) = k+1$

More generally,

$$G_f = \frac{1}{(1-ax)^b} \quad \text{for} \quad f(k) = (-a)^k \cdot \binom{-b}{k} = \binom{b+k-1}{k} \cdot a^k$$

We have:

- $G_{f+g} = G_f + G_g$
- $G_g(x) = x \cdot G_f(x)$ where $g(0) = 0$ and $g(k+1) = f(k)$ for $k \geq 0$.
  (you can similarly shift for multiplication by x^r)

So if we have the generating function, we can often get closed form expressions.
But how do we get the generating function in the first place?

Generating Functions from Recurrence Relations

Consider the Fibonacci sequence

$$f(0) = 0, \quad f(1) = 1, \quad f(n) = f(n-1) + f(n-2) \text{ for all } n \geq 2$$

We have

$$f(n) \cdot x^n = x \cdot f(n-1) \cdot x^{n-1} + x^2 \cdot f(n-2) \cdot x^{n-2}$$

If we sum this up over $n \geq 2$, we get

$$\sum_{n \geq 2} f(n) \cdot x^n = x \cdot \sum_{n \geq 1} f(n) x^n + x^2 \cdot \sum_{n \geq 0} f(n) \cdot x^n$$

$$G_f(x) - f(0) - f(1) \cdot x = x \cdot (G_f(x) - f(0)) + x^2 G_f(x)$$

$$G_f(x)\,(1 - x - x^2) = f(0) + (f(1) - f(0)) \cdot x$$

$$\Rightarrow G_f(x) = \frac{x}{1 - x - x^2} \qquad \text{(Substituting } f(0) + f(1)\text{)}$$

More generally, if we have

$$f(0) = c, \quad f(1) = d, \quad f(n) = a \cdot f(n-1) + b \cdot f(n-2) \text{ for all } n \geq 2$$

$$G_f(x) = \frac{c + (d - ac)x}{1 - ax - bx^2}$$

(Linear) recurrence relations result quite naturally in recurrence relations.

Now, suppose

$$g(k) = \sum_{j=0}^{k} f(j)$$

What is G_g in terms of G_f?

Recursively, we can define g by

$$g(0) = f(0)$$
$$g(n) = g(n-1) + f(n-1) \quad \text{for} \quad n \geq 1.$$

Similar to the Fibonacci example, we can do

$$g(k)\, x^k = g(k-1) \cdot x^{k-1} \cdot x + f(k) \cdot x^k$$

$$G_g(x) = g(0) + x \cdot G_g(x) + [G_f(x) - f(0)]$$

$$G_g(x) = x \cdot G_g(x) + G_f(x)$$

$$\rightarrow G_g(x) = \frac{G_f(x)}{1-x}$$

Asymptotic analysis

In the running time of an algorithm, we are usually interested in how it scales, not the exact time.
Further, the exact time also depends on various parameters such as the system it is running on, the compiler etc.

If the time is, say, $3n^2+9$, we only care about n^2 part of it.
For a doubling of the input size, it grows by roughly 4 times.
It grows like n^2.

Def. Given functions $T(n)$ and $O(n)$, we write $T(n) = O(f(n))$ if there exist $c, k > 0$ such that for all $n \geq k$, $0 \leq T(n) \leq cf(n)$

Note that we require T and f to be (eventually) non-negative.

We're actually supposed to write $T(n) \in O(f(n))$, this is notational abuse.

We typically consider $T, f : \mathbb{N} \to \mathbb{R}$.
Note that c does not depend on n, it is a constant factor.

For example, $3n^2+9 = O(n^2)$

$$2^n = O(2^n \log n)$$

We usually use the O notation to analyse the worst-case running time of an algorithm.

$$|\log(n!) - \log(n^n)| = O(n)$$

We can do even better:

$$|\log(n!) - \log((n/e)^n)| = O(\log n)$$

even better:

$$|\log(n!) - \log((n/e)^n) - \tfrac{1}{2}\log n| = O(1)$$

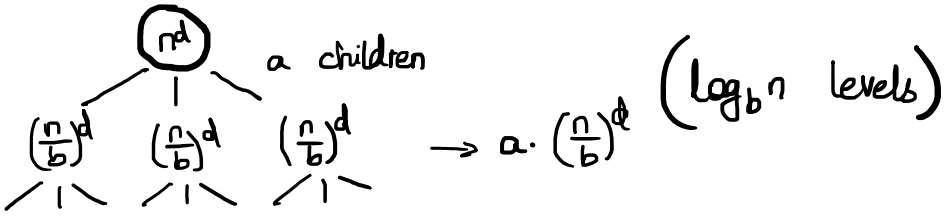
We could do even better if we have an $O(f(n))$ where f is decreasing - an error that decreases as the size grows.

- If $T(n) = O(f(n))$ and $R(n) = O(f(n))$, then $T(n) + R(n) = O(f(n))$
  If c_T, k_T, c_R, k_R, consider $c = c_T + c_R$ and $k = \max(k_T, k_R)$
- If eventually $R(n) \leq T(n)$, then $T(n) - R(n) = O(T(n))$
  For $n \geq \max(k_T, k_R)$,
  $$0 \leq T(n) - R(n) \leq T(n)$$
- If $T(n) = O(g(n))$ and $g(n) = O(f(n))$, then $T(n) = O(f(n))$.
  If c_T, k_T, c_g, k_g, consider $c = c_T c_g$ and $k = \max(k_T, k_g)$
- If $T(n)$ is upperbounded by a degree d polynomial with a positive coefficient for n^d, $T(n) = O(n^d)$.

→ $T(n) = O(1)$ ≡ $T(n)$ is bounded above.
There exists $c > 0$ s.t. $T(n) \leq c$ for all n.

→ $T(n) = O(\log n)$ Quite slow growth. If n doubles, it increases by 1.
→ $T(n) = O(n)$ $T(n)$ is "linear" in n.
→ $T(n) = O(n^2)$ $T(n)$ is "quadratic" in n.
→ $T(n) = O(n^d)$ for some fixed d
$T(n)$ is "polynomial" in n.
→ $T(n) = O(2^{d \cdot n})$ for some fixed d
$T(n)$ is "exponential" in n.

Suppose $T(n) = a\,T(n/b) + cn^d$ for some $a, b \in \mathbb{N}$, $c > 0$ and $d \geq 0$ and $T(1) = 1$. (Divide and conquer)
(Say $n = b^k$ so only integers are encountered)
We want to analyze T using the O notation.



The total of the i^{th} level is $a^i \cdot \left(\frac{n}{b^i}\right)^d$

$$\Rightarrow T(n) = O\left(n^d\left(1 + \left(\frac{a}{b^d}\right) + \left(\frac{a}{b^d}\right)^2 + \cdots + \left(\frac{a}{b^d}\right)^k\right)\right)$$

$(k = \log_b n)$

If $a = b^d$, then
$T(n) = O(n^d \log n)$

If $a < b^d$, then
$T(n) = O(n^d)$

If $a > b^d$, then
$T(n) = O(n^{\log_b a})$

} Master Recurrence Theorem

$$\left(O(a^k) = O\left(2^{\log n \cdot \log a / \log b}\right) = O(n^{\log_b a})\right)$$

Now, big O notation does not give a tight upper bound. For this, we introduce the Θ notation.

Def. We write $T(n) = \Theta(f(n))$ if $T(n) = O(f(n))$ and $f(n) = O(T(n))$.

For example, $3n^2 - n = \Theta(n^2)$

- If $T(n) = \Theta(f(n))$ and $R(n) = \Theta(f(n))$, then $T(n) + R(n) = \Theta(f(n))$

We say that f and g are asymptotically equal and write $f(n) \simeq g(n)$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

If there exists $c > 0$ s.t. $f(n) \simeq c \cdot g(n)$, then $f(n) = \Theta(g(n))$.
(and f, g are eventually non-negative)

We say that f is asymptotically much smaller than g and write $f(n) \ll g(n)$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

If $f(n) \ll g(n)$, then $f(n) = O(g(n))$ but $f(n) \neq \Theta(g(n))$
(for eventually positive f, g)

Note that Θ and O do not require the limit to exist.
($f(n) = 2n$ for odd n and n for even n. Then
$f(n) = \Theta(n)$ but $\lim_{n \to \infty} \frac{f(n)}{n}$ does not exist.)