Chapter 12-Recursive Definition

Thursday, January 5, 2023
1:12 AM

Recursive Definitions:

Similar to recursive functions in programming.
Has 2 parts:
A Base Case(s)
A Recursive Formula

Fibonacci Numbers Definition Example:

$$F_0 = 0$$

$$F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2}, \qquad \forall i \geq 2$$

Most recursive numerical formulas have a ***closed form***, or an equivalent expression that doesn't involve recursion.
(basically big brain ways to summarize a recursive function into a math formula)

Unrolling:

A Technique used to find the closed form.
(unrolls the recursive part, substituting in previous values)
Example:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(n-1) + 3, \qquad \forall n \geq 2 \end{aligned}$$

Unroll:

$$\begin{aligned} T(n) &= 2T(n-1) + 3 \\ &= 2(2T(n-2) + 3) + 3 \\ &= 2(2(2T(n-3) + 3) + 3) + 3 \\ &= 2^3T(n-3) + 2^2 \cdot 3 + 2 \cdot 3 + 3 \\ &= 2^4T(n-4) + 2^3 \cdot 3 + 2^2 \cdot 3 + 2 \cdot 3 + 3 \\ &\dots \\ &= 2^kT(n-k) + 2^{k-1} \cdot 3 + \dots + 2^2 \cdot 3 + 2 \cdot 3 + 3 \end{aligned}$$

$$\begin{aligned} &= 2^k T(n-k) + 3(2^{k-1} + \ldots + 2^2 + 2 + 1) \\ &= 2^k T(n-k) + 3\sum_{i=0}^{k-1}(2^i) \end{aligned}$$

But notice that there's still a k hanging in there (because we don't know for how long the function will go)

That's when we need to use the base case

From the definition, we know that n=1 is the base case

And so from the unrolled formula we know to substitute 1 into n-k. (and k = n-1)

$$\begin{aligned} T(n) &= 2^k T(n-k) + 3\sum_{i=0}^{k-1}(2^i) \\ &= 2^{n-1}T(1) + 3\sum_{i=0}^{n-2}(2^i) \\ &= 2^{n-1} + 3\sum_{k=0}^{n-2}(2^k) \\ &= 2^{n-1} + 3(2^{n-1} - 1) = 4(2^{n-1}) - 3 = 2^{n+1} - 3 \end{aligned}$$

Another Example:

$$\begin{aligned} S(1) &= c \\ S(n) &= 2S(n/2) + n, \qquad \forall n \geq 2 \;\; (n \text{ a power of } 2) \end{aligned}$$

$$\begin{aligned} S(n) &= 2S(n/2) + n \\ &= 2(2S(n/4) + n/2) + n \\ &= 4S(n/4) + n + n \\ &= 8S(n/8) + n + n + n \\ &\ldots \\ &= 2^i S(\frac{n}{2^i}) + in \end{aligned}$$

$$S(n) = 2^i S(\frac{n}{2^i}) + in = 2^{\log n}c + n\log n = cn + n\log n$$

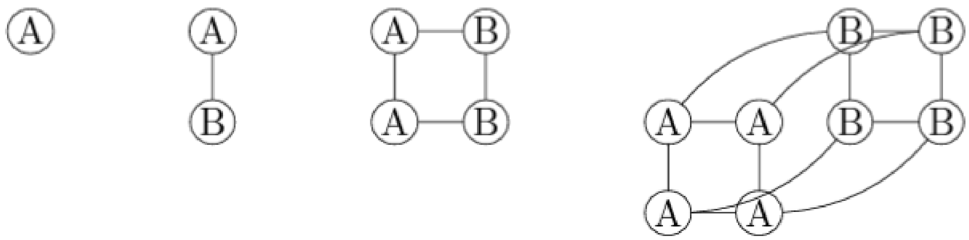
Hypercubes:

Defined Recursively as follows:

Q_0 is a single node with no edges

Q_n consists of two copies of Q_{n-1} with edges joining corresponding nodes, for any $n \geq 1$.

Diagrams:



Q^n has 2^n nodes. To compute the number of edges, we set up the following recursive definition for the number of edges $E(n)$ in the Q_n:

$$E(0) = 0$$

$$E(n) = 2E(n-1) + 2^{n-1}, \text{ for all } n \geq 1$$

Proofs with recursive definitions:

Claims involving recursive definitions often require proofs using a strong inductive hypothesis.

(Since it is often necessary to substitute in/refer to previous values in these proofs)

Example:

f(n) is defined to be:

$$f(0) = 2$$

$$f(1) = 3$$

$$\forall n \geq 1,\ f(n+1) = 3f(n) - 2f(n-1)$$

Claim 45 $\forall n \in \mathbb{N}, f(n) = 2^n + 1$

Proof: by induction on n.

Base: $f(0)$ is defined to be 2. $2^0+1 = 1+1 = 2$. So $f(n) = 2^n+1$ when $n = 0$.

$f(1)$ is defined to be 3. $2^1 + 1 = 2 + 1 = 3$. So $f(n) = 2^n + 1$ when $n = 1$.

Induction: Suppose that $f(n) = 2^n + 1$ for $n = 0, 1, \ldots, k$.

$f(k + 1) = 3f(k) - 2f(k - 1)$

By the induction hypothesis, $f(k) = 2^k+1$ and $f(k-1) = 2^{k-1}+1$. Substituting these formulas into the previous equation, we get:

$f(k+1) = 3(2^k+1)-2(2^{k-1}+1) = 3\cdot 2^k+3-2^k-2 = 2\cdot 2^k+1 = 2^{k+1} + 1$

So $f(k + 1) = 2^{k+1} + 1$, which is what we needed to show.