Recurrences and Algorithmic Complexity

R. Sekar

$$\boxed{\text{Solving Recurrences}}$$

Solving Recurrences: Plug and Chug

- Expand the recurrence out for a few steps
- Identify the pattern
- Guess a solution based on the pattern
- Check the solution for a few small values of n
- Verify using induction

Plug and Chug for Tower of Hanoi Recurrence

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

Solving Linear Recurrences

- *Homogeneous* linear recurrences are of the form

$$f(n) = \sum_{i=1}^{d} a_i f(n-i)$$

  - Example: Fibonacci series $F(n) = F(n-1) + F(n-2)$
- They are known to have an *exponential* solution $f(n) = x^n$ for some x
  - Substitute this solution into the recurrence and solve for x:

$$\begin{aligned} x^n &= \sum_{i=1}^{d} a_i x^{n-i} \\ x^d &= \sum_{i=1}^{d} a_i x^{d-i} && \text{(Dividing all terms by } x^{n-d}\text{)} \\ \sum_{i=0}^{d} a_i x^{d-i} &= 0 && \text{(Rearrange terms to arrive at a polynomial, with } a_0 = 1\text{)} \end{aligned}$$

Solving Homogeneous Linear Recurrences (Contd.)

- Find the roots $r_1, \ldots, r_d$ of of this polynomial $\sum_{i=0}^{d} a_i x^{d-i} = 0$
- The general solution to the recurrence is

$$f(n) = \sum_{i=1}^{d} k_i r_i^n$$

- Solve for k_i using known values for $f(0)$ through $f(d-1)$.
- Note: if the polynomial has fewer than d roots, the general form of the solution gets more complicated — we will ignore this case here.

Solving Homogeneous Linear Recurrences: Fibonacci Example

$$f(n) = f(n-1) + f(n-2)$$

Solving Homogeneous Linear Recurrences: Fibonacci Example

$$f(n) = f(n-1) + f(n-2)$$

1. Substitute $f(n) = x^n$ in this equation, simplify to get *characteristic equation* $x^2 = x + 1$
2. Solve this quadratic equation to obtain roots $p = \frac{1+\sqrt{5}}{2}$ and $q = \frac{1-\sqrt{5}}{2}$
3. By the homogeneous linear recurrence method, the general solution is $f(n) = k_1 p^n + k_2 q^n$
4. Plug in $f(0) = 0$ and $f(1) = 1$ to obtain the following equations:
   - $k_1 p^0 + k_2 q^0 = f(0) = 0$
   - $k_1 p^1 + k_2 q^1 = k_1 \left(\frac{1+\sqrt{5}}{2}\right) + k_2 \left(\frac{1-\sqrt{5}}{2}\right) = f(1) = 1$

Solving Homogeneous Linear Recurrences: Fibonacci Example

$$f(n) = f(n-1) + f(n-2)$$

1. Substitute $f(n) = x^n$ in this equation, simplify to get *characteristic equation* $x^2 = x + 1$
2. Solve this quadratic equation to obtain roots $p = \frac{1+\sqrt{5}}{2}$ and $q = \frac{1-\sqrt{5}}{2}$
3. By the homogeneous linear recurrence method, the general solution is $f(n) = k_1 p^n + k_2 q^n$
4. Plug in $f(0) = 0$ and $f(1) = 1$ to obtain the following equations:
   - $k_1 p^0 + k_2 q^0 = k_1 + k_2 = f(0) = 0$ which means $k_2 = -k_1$
   - $k_1 p^1 + k_2 q^1 = k_1 \left(\frac{1+\sqrt{5}}{2}\right) + k_2 \left(\frac{1-\sqrt{5}}{2}\right) = (k_1 + k_2)/2 + \sqrt{5}(k_1 - k_2)/2 = f(1) = 1$
   - Substituting $k_2 = -k_1$ in this equation and simplifying, we get $k_1 = 1/\sqrt{5}$.
5. Thus, the solution is

$$f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Observations about Fibonacci Recurrence Solution

- All Fibonacci numbers are integers — it is mind-boggling that its closed form solution contains not just fractions, but *irrational* numbers!
  - No wonder that this solution was unknown for six centuries!
- Note that $|q| = |\frac{1-\sqrt{5}}{2}| = 0.6180 < 1$ so q^n rapidly approaches zero. For instance, $q^{20} \approx 0.00006$, and the error in $f(n)$ due to ignoring q is less than one in 10^{-8}.
- So, for larger n, $f(n)$ is determined almost entirely by the first term $\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$
  - $p^n/\sqrt{5}$ is very close to an integer value, although p is irrational!
- The ratio between successive Fibonacci numbers converges to $p = 1.618$, which is called the *golden ratio*

Asymptotic Complexity

- Expressing complexity in terms of "number of steps" is a simplification
  - Each such operation may in fact take a different amount of time
  - But it is too complex to worry about the details, esp. because they differ across programming languages, processor types, etc.

Asymptotic Complexity

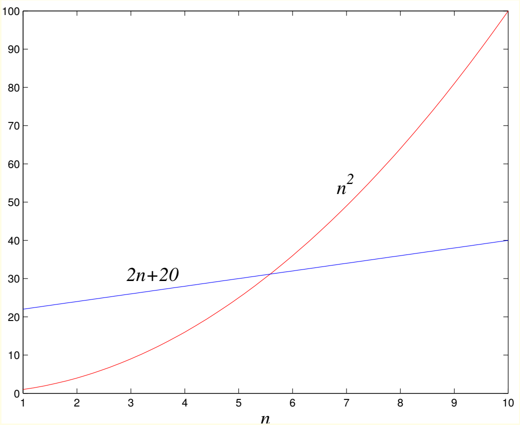
- Expressing complexity in terms of “number of steps” is a simplification
  - Each such operation may in fact take a different amount of time
  - But it is too complex to worry about the details, esp. because they differ across programming languages, processor types, etc.
- Why not simplify further?
  - Capture just the growth rate of $T(n)$ as a function of n
  - Ignore constant factors
    - No need to count operations in a loop (their number should be bounded by a constant)
  - Ignore exceptions from the formula for small values of n

Asymptotic Complexity: Big-O notation

Definition

Given functions $f, g : \mathbb{R} \longrightarrow \mathbb{R}$*, we say*
$f = O(g)$*, i.e., "f grows no faster than g,"*
iff

$$\lim_{x \to \infty} f(x)/g(x) < c \text{ for some constant } c$$



Big-O notation: Examples

- $10n = O(n)$
- $0.0001n^3 + n = O(n^3)$
- $2^n + 10^n + n^2 + 2 = O(10^n)$
- $0.0001n \log n + 10000n = O(n \log n)$

Solving Divide-and-Conquer Recurrences: Master Theorem

If $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$ for constants $a > 0$, $b > 1$, and $d \geq 0$, then

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

Solving Recurrences: Examples Using Master Theorem

$$T(n) = 2T(n/2) + n$$

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

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

Solving Recurrences: Examples Using Master Theorem

$$T(n) = 4T(n/2) + n^3$$

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

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

Solving Recurrences: Examples Using Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

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

$$T(n) = 3T(n/2) + n$$

Summary

- Recursion and induction
  - Examples
- Recurrence Solving Techniques
  - Plug-and-Chug
  - Homogeneous linear equations
- Recurrences for algorithm runtimes
- Asymptotic complexity
  - Divide-and-Conquer recurrences and Master theorem