UNIVERSITY OF SOUTHERN MAINE
Department of Computer Sciences

COS 485
Department of Computer Sciences

Design/Analysis of Comp Algorithms
Department of Computer Sciences

General Notation

- $\neg$ → Not
- $\exists$ → There exists
- $:$ → Such that
- $\forall$ → For all
- $\equiv / \Leftrightarrow$ → Equivalent
- $\implies$ → Implies
- $\in$ → Is in/Belongs to
- $\wedge$ → And
- $\vee$ → Or
- $\nearrow$ → Strictly increasing
- $\searrow$ → Strictly decreasing
- $\nearrow$ → Eventually increasing
- $\searrow$ → Eventually decreasing

Sets of Numbers

- $\mathbb{Z}$ → Integers
- $\mathbb{R}$ → Real numbers
- $\mathbb{Q}$ → Rational numbers
- $\mathbb{N}$ → Natural numbers
- $\mathbb{P}$ → Prime numbers
- U → Universal set (everything)
- $\varnothing$, $\{\}$ → Empty set (nothing)

Common Operations

- **Floor** $\lfloor n \rfloor$ → Truncate n
- **Ceiling** $\lceil n \rceil$ → Round up n
- **Factorial** $n!$ → $1 \times 2 \times 3 \times 4 \times \cdots \times n$
- **Summation** $\sum_{k=1}^{n} f(k)$ → $\sum_{k=1}^{5} 2k = 2 + 4 + 6 + 8 + 10$
- **Binomial** $\binom{n}{k}$ → $\frac{n!}{k!(n-k)!}$
- **Divides** $n \mid k$ → iff $\exists c \in \mathbb{Z} : k = nc$

Rules of Logarithms

- $\log_b x = y \iff b^y = x$
- $\mathrm{b}^{\log_b x} \iff x$
- $\log_b(b^x) \iff x$
- $\log_b(xy) \iff \log_b x + \log_b y$
- $\log_b(x/y) \iff \log_b x - \log_b y$
- $\mathrm{x}^{\log_b n} = n^{\log_b x}$
- $\log_b(x^y) \iff y \times \log_b x$
- $\mathrm{x}^{\log_b n} = n^{\log_b x}$

Big-O, o, Ω and Θ $(\forall n > n_0)$

- $f(n) \in O(g(n)) \rightarrow \exists C : 0 \le f(n) \le Cg(n)$
- $f(n) \in o(g(n)) \rightarrow \forall C > 0 : 0 \le f(n) < Cg(n)$
- $f(n) \in \Omega(g(n)) \rightarrow \exists C : 0 \le Cg(n) \le f(n)$
- $f(n) \in \Theta(g(n)) \rightarrow \exists A, B : Ag(n) \le f(n) \le Bg(n)$

Common Time-Complexities

- **Constant** → $O(1)$
- **Logarithmic** → $O(\log n)$
- **Linear** → $O(n)$
- **Quasi-Linear** → $O(n \log n)$
- **Quadratic** → $O(n^2)$
- **Cubic** → $O(n^3)$
- **Exponential** → $O(2^n)$
- **Factorial** → $O(n!)$

Proof Types (For $P \implies Q$)

- **Direct** → Assume P, then use rules of logic to prove Q
- **Cases/Exhaustion** → $P_1 \vee P_2 \vee \cdots \vee P_n \implies Q$
- **Contradiction** → Assume P and derive $P \not\Rightarrow Q$
- **Inductive** → $P(1) \wedge P(k+1) \implies P(k)$

Common Series

- $\sum_{k=1}^{n} k \rightarrow 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$
- $\sum_{k=1}^{n} k^2 \rightarrow 1 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{k=1}^{n} k^3 \rightarrow 1 + 2^3 + 3^3 + \cdots + n^3 = \left[\frac{n(n+1)}{2}\right]^2$
- $\sum_{k=1}^{n} ar^{k-1} \rightarrow \mathrm{ar}^0 + ar^1 + ar^2 + \cdots + ar^{n-1} = a\left(\frac{1-r^n}{1-r}\right)$
- $\sum_{k=0}^{n} \binom{n}{k} \rightarrow \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n$

Programs

BS Computer Science

Cheat Sheet

Published ???, 2024.

Contact

Dr. James Quinlan

UNIVERSITY OF SOUTHERN MAINE

Department of Computer Sciences

COS 485

Department of Computer Sciences

Design/Analysis of Comp Algorithms

Department of Computer Sciences

Solving Recurrence Relations

The general steps for solving a recurrence relation are

- Find an explicit solution using the Characteristic Equation
- Verify solution using an inductive proof

First use Algebra to take a simple Recurrence Relation and turn it into a LHRRCC

$T_n = T_{n-1} + T_{n-2}$

LHRRCC is a Linear Homogeneous Recurrence Relation with Constant Coefficients.

$T_n - T_{n-1} - T_{n-2} = 0$

Use the Characteristic Equation, where C is the coefficient from the LHRRCC:

$C_0 X^k + C_1 X^{k-1} + C_2 X^{k-2} + \cdots + C_k = 0$

For this example it would be

$x^2 - x - 1 = 0 \implies x = \frac{1 \pm \sqrt{5}}{2}$

Then solve for x and substitute into the General Solution

$f_n = A x_1^n + B x_1^n$

Next, use the initial conditions to solve for the explicit solution

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

Finally, veryify the explicit solution that was found using an inductive proof.

Divide and Conquer

If $f(n)$ is the number of operations required to solve an intial problem, then a Divide and Conquer Recurrence Relation will look like

$$f(n) = \underbrace{a}_{\text{subproblems}} \; f \; \underbrace{\overbrace{\left(\tfrac{n}{b}\right)}^{\text{input size}}}_{\text{branching factor}} + \underbrace{g(n)}_{\text{work per level}}$$

where $n = b^k$, $k \geq 1$

Theorem: Suppose $f \nearrow$ (*is strictly increasing*), $b \mid n$, $a \geq 1$, $b \geq 1$, $c \in^{+}$: $f(n) = af\left(\frac{n}{b}\right) + O(1)$, then

$$f(n) \in \begin{cases} O(n^{\log_b a}) & \text{if } a > 1 \\ O(\log_b n) & \text{if } a = 1 \end{cases}$$

Master Theorem

Suppose a complexity function $T(n)$ is eventually non-decreasing and satisfies

$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$ for $n > 1$, n a power of b

$T(1) = c$

Where $b \geq 2$ and $d \geq 0$ are constant *integers*, and a and c are constant such that $a > 0$ and $c > 0$.

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \lg n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Greedy Algorithms

Greedy algorithms are a class of algorithms that make **locally optimal** choices at each step with the hope of finding a **global optimum** solution. The idea is to select the best local choice at each step, leading to a solution that may or may not be the most optimal but is often good enough. Problems that are solved using Greedy Algorithms include

- Knapsack Problem
- Shortest Path (Graph)
- Minimal Spanning Tree

Dynamic Programming (DP)

DP solves problems by breaking them down into simpler **subproblems**. By solving each subproblem only once and storing the results, it avoids repeating computations and is more efficient. Dynamic Programming algorithms can use either a 'Top-Down' approach or a 'Bottom-Up' approach which use **memoization** or **tabulation** to store subproblems, respectively. Some examples of problems that use Dynamic Programming are:

- Fibonacci Sequence
- Shortest Path (Graph)
- Chain Matrix Multiplication

Programs

BS Computer Science

Cheat Sheet

Published ???, 2024.

Contact

Dr. James Quinlan