ECE358: Foundations of Computing

Aman Bhargava

September-December 2020

Contents

0.1 Introduction and Course Information

This document offers an overview of the ECE358 course. They comprise my condensed course notes for the course. No promises are made relating to the correctness or completeness of the course notes. These notes are meant to highlight difficult concepts and explain them simply, not to comprehensively review the entire course.

Course Information

- Professors: Prof. Andreas Veneris and Prof. Zissis Poulos

- Course: Engineering Science, Machine Intelligence Option
- Term: 2020 Fall

Chapter 1

Math Review

1.1 Logarithms

Logarithm Rules:

- **Definition:** $a = b^c$ iff $log_b a = c$.
- $a = b^{\log_b a}$.
- $\log_c(ab) = \log_c a + log_c b$.
- $\log_c(a^n) = n \log_c(a)$.
- $\log_b(a) = \log_a(b)$.
- $\log(1/a) = -\log a$.
- $\log(a/c) = \log a - \log c$.
- $a^{\log n} = n^{\log a}$.

Variations on Logarithm

$$\log^{(i)} n = \begin{cases} n & \text{iff } i = 0 \\ \log(\log^{(i)} n) & \text{otherwise} \end{cases} \tag{1.1}$$

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + log^*(\log n) & \text{otherwise} \end{cases} \tag{1.2}$$

1.2 Sequences and Series

Theorem 1 *Fibonnaci Definition*

$$F_i = F_{i-1} + F_{i-2} \tag{1.3}$$

Where $F_0 = 0,\ F_1 = 1.$

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}} \tag{1.4}$$

Where $\phi = \frac{1+\sqrt{5}}{2}$, $\hat{\phi} = \frac{1-\sqrt{5}}{2}$

Theorem 2 *Arithmetic Series*

$$\sum_{i=1}^{n} = \frac{n(n+1)}{2} = \Theta(n^2) \tag{1.5}$$

Theorem 3 *Geometric Series*

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} \tag{1.6}$$

Theorem 4 *Infinite Series*

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \text{ iff } |x| < 1 \tag{1.7}$$

Theorem 5 *Telescoping Series*

$$\sum_{i=1}^{n} a_i - a_{i-1} = a_n - a_0 \sum_{i=1}^{n} a_i - a_{i+1} = a_0 - a_n \tag{1.8}$$

1.3 Combinatorics

Theorem 6 *Binomial Coefficient*

$$(x+y)^r = \sum_{i=0}^{r} \binom{r}{i} x^i y^{r-i} \tag{1.9}$$

Where $\binom{r}{i}$ *is the binomial coefficient which equals*

$$\frac{r!}{i!(r-i)!}$$

Chapter 2

Asymptotics

What are Asymptotics? Analysis method for performance and complexity of an algorithm (space/memory, time/clock cycle complexity).

- Tight Bound: $\Theta()$.
- Worst Case: $O()$.
- Best Case: $\Omega()$
- Average & Expected Case: *Randomized + probabilistic analysis. Not a focus for the course.*
- Amortized Case: Discused later on. "Average worst case".

Useful Facts

- Θ bound is not always possible to find.
- **Transitivity:** if $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $f(n) = \Theta(h(n))$.
- **Symmetry:** $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.
- **Transpose:** $f(n) = O(g(n))$ if and only if $g(n)\Omega(f(n))$.
- $n^a \in O(n^b)$ iff $a \leq b$.
- $\log_a(n) \in O(\log_b(n)) \, \forall a, b$.
- $c^n \in O(d^n)$ iff $c \leq d$.

- If $f(n) \in O(f'(n))$ and $g(n) \in O(g'(n))$ then

$$f(n) \cdot g(n) \in O(f'(n) \cdot g'(n))$$

$$f(n) + g(n) \in O(\max(f'(n), g'(n))$$

Theorem 7 *Big O Definition $f(n) = O(g(n))$ if and only if: There exists $c > 0$ and $n_0 > 0$ such that*

$$0 \leq f(n) \leq cg(n)\, \forall n \geq n_0 \tag{2.1}$$

Theorem 8 *Big Ω Definition*

"Lower Bound" Big Ω Definition: *$f(n) = \Omega(h(n))$ if and only if: There exists $c_1, n_1 > 0$ such that*

$$0 \leq c_1 h(n) \leq f(n)\, \forall n \geq n_1 \tag{2.2}$$

Theorem 9 *Big Θ Definition*

"Tight Bound" Big Θ Definition: *$f(n) = \Theta(g(n))$ if and only if:*
There exists $c_1, c_2, n_0 > 0$ such that

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\, \forall n > n_0 \tag{2.3}$$

Chapter 3

Graphs

3.1 Definitions

Theorem 10 *Graph Definition A* ***graph*** *is a collection of* ***vertices*** *and* ***edges*** *represented by*

$$G = (V, E) \tag{3.1}$$

Graphs can be:

- *Directed or Undirected.*
- *Weighted or Unweighted.*

Path: A sequence of **edges** between adjacent vertices.

- **Simple Path:** No vertex is repeated in the path.
- **Cycle:** A path that starts and ends with the same vertex.
- A **connected** graph has a path between any two vertices (else it is **disconnected**)

Bipartite Graphs: A graph is **bipartite** if and only if vertices V can be divided into V_1, V_2 such that:

1. $V_1 \cap V_2 = \emptyset$.
2. $V_1 \cup V_2 = V$.
3. Adjacencies exist only between elements and V_1 and V_2 (i.e., vertices within each set are disconnected from eachother, but are connected to elements from the other set).

Vertex Degree is the number of **edges** adjacent to a vertex (includes incoming and outgoing edges).

- **In-degree** is the number of incoming edges.
- **Out-degree** is the number of outgoing edges.

Clique: For all $v_1, v_2 \in V$, there exists an edge connecting them (i.e. a fully-connected set of vertices).

Representations for Graphs: We either use an **adjacency matrix** or an **adjacency matrix**.

- **Adjacency List:** Each element in the list represents a vertex and "has" a collection of **pointers** connecting them to other elements of the list.
  - **Time Complexity:** $O(n)$ – Determining if E_{v_1,v_2} exists would, at worst, require you to check n pointers arising from vertex v_1 and/or v_2.
  - **Space Complexity:** $O(|E|)$ – with worst case that $|E| = n^2$ for clique.
- **Adjacency Matrix:** Matrix contains weights connecting v_i to v_j at index $M_{i,j}$.
  - **Time Complexity:** $O(1)$ – Just check address $M_{i,j}$ and $M_{j,i}$.
  - **Space Complexity:** $O(n^2$ every time.

Chapter 4

Trees

4.1 Definitions

Theorem 11 *Tree Definition A **tree** is a **connected, acyclic, undirected** graph.*

- ***Root node*** *is usually defined.*
- ***Parent node*** *is the 'next node up' going toward the root.*
- ***Child node*** *is one of the 'next node(s) down' going away from the root.*
- ***Binary tree:*** ≤ 2 *children per node.*
- ***Depth of node:*** *Length of path from **root** $\rightarrow$ **node**.*
- ***Height of node:*** *Number of edges on the **longest path** from the node $\rightarrow$ leaf.*
- ***Complete k-ary tree:*** *Every internal node has k children and all leaves are at the same depth.*

Theorem 12 *One-for-all Tree Theorem **If one of these is true, they all are:***

- *G is a tree.*
- *Every pair of vertices $v_1, v_2 \in G$ is connected by a **unique, simple path**.*
- *G is disconnected, but becomes disconnected when one edge is removed.*

- *G is connected with $|E| = |V| - 1$.*
- *G is acyclic.*
- *If an edge is added G* ***becomes cyclic****.*

Chapter 5

Proof Methods

5.1 Induction

Basic Idea: To prove by induction, we show that the statement holds for some base case $n = 1$, then show that the statement holding for aritrary n implies that the statement holds for $n + 1$. That concludes the proof.

Basis: We prove that the statement holds for some set values of n (usually $n = 1$ or $n = 0, 1, ..., 4$).

Hypothesis: We hypothesize that the thing we are trying to prove is true.

Inductive Step: We "plug in" the $n+1$ case to the hypothesis and show that it boils down to the n case and some algebraic equivalence. Then you can make the claim that it holds for all n.

5.2 Contradiction

Basic Idea: Given a true or false proposition P, we assume that $\neg P$ ("not P") holds. After some clever algebra and symbol-shunting, we get a contradiction. This implies that P must hold instead.

Illustrative Example: P: If $x^2 - 5x + 4 < 0$, then $x > 0$.

- To prove P, we **assume towards a contradiction** (ATaC) that $\neg P$ holds. That is, we assume that if $x^2 - 5x + 4 < 0$ then $x \leq 0$.

- But then:

$$\begin{aligned} x^2 &< 5x - 4 \\ x^2 &< 0 \end{aligned} \tag{5.1}$$

Results in a contradiction! Therefore P must hold.

5.3 Master Theorem

This provides a "hammer" to prove the time complexity of recurrent functions.

Theorem 13 *The Master Theorem Let $a \geq 1$ and $b \geq 1$ and $f(n)$ be some function.*

$$T(n) = aT(\frac{n}{b}) + f(n) \tag{5.2}$$

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ *for some* $\epsilon > 0$. *Then*

$$T(n) = \Theta(n^{\log_b a})$$

Case 2: $f(n) = \Theta(n^{\log_b a})$. *Then*

$$T(n) = \Theta(n^{\log_b a} \log n)$$

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ *for some* $\epsilon > 0$ ***AND*** $af(n/b) \leq cf(n)$ *for* $0 < c < 1$. *Then*

$$T(n) = \Theta(f(n))$$

.

5.4 Substitution

Substitution is a method for determining the **closed from** runtime of an algorithm via induction.

Example – Mergesort: The runtime for mergesort is recursively defined as $T(n) = 2T(\lceil n/2 \rceil) + n$.

1. We start by **guessing** $T(n) = O(n \log n)$.

2. We **hypothesize** that $T(n) = O(n \log n)$ for all cases $\leq n$, meaning that
$$T(n/2) \leq c\lfloor n/2 \rfloor \log\lfloor n/2 \rfloor$$

3. **Inductive step:** We prove that
$$T(n) \leq cn \log n$$
Which is pretty obvious from there.