

Lecture 2

*Lecturer: Andre Wibisono**Scribe: John Lazarsfeld*

1 Overview

This lectures continues a review of some mathematical preliminaries needed for the course, and we also introduce our first algorithmic problem setting. Specifically, these lecture notes cover:

- a review on asymptotics
- a review of graphs
- a brief introduction to the Stable Matching Problem (this is covered more fully in Lecture 3).

2 Asymptotics

We start with a brief review of asymptotic notation (which you have likely encountered before). See [KT, 2.2] for a more complete treatment.

To begin with some motivation, in this class we will often be concerned with the running times¹ of algorithms, with a particular focus on how these running times behave as the size of an algorithm's input grows. Usually, we can associate the “size” of an input with a natural number $n \in \mathbb{N}$. For example, n might be the input's *length* (e.g., if the input is a one-dimensional array), but it could also represent some other input dimension (e.g., the number of vertices in a graph, or the number of rows in a matrix). Generally, the running time of an algorithm depends on n (intuitively, more computation is required as the input size grows). Now suppose we have two algorithms, and their running times on inputs of size n are described by the functions $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N} \rightarrow \mathbb{N}$. In this class, we usually only want to *compare* $f(n)$ and $g(n)$ up to some meaningful behavior in a way that ignores low-order or constant terms. For this, we use standard asymptotic (big-O) notation:

Definition 1 (Asymptotic Upper Bounds (“big-O”)). *For functions f and g , we say $f(n) = O(g(n))$ if there exists constants $C > 0$ and $N \in \mathbb{N}$ such that for all $n \geq N$, we have $f(n) \leq C \cdot g(n)$.*

Definition 2 (Asymptotic Lower Bounds (“big-Omega”)). *For functions f and g , we say $f(n) = \Omega(g(n))$ if there exists constants $C > 0$ and $N \in \mathbb{N}$ such that for all $n \geq N$, we have $f(n) \geq C \cdot g(n)$.*

¹Unless specified otherwise, we use “running time” to refer to the number of *pseudo-code* steps that are performed by an algorithm. See [KT 2.2, pg 35-36] for some discussion on this.

Definition 3 (Asymptotic Equivalence (“Theta”)). *For functions f and g , we say $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.*

Observe that the existential quantification of C and N in these definitions imply that we don’t “care” about constant factors or lower-order terms when comparing functions asymptotically. For example if $f(n) = 3n^2 + \log n$, the additive $\log n$ is a lower-order term since it grows more slowly with n than $3n^2$. Additionally, we don’t care much about constant factor 3. So in this case, we would say $f(n) = \Theta(n^2)$. Note that when comparing functions $f(n)$ and $g(n)$, we sometimes will write $f = O(g)$ or $f = \Omega(g)$ for convenience.

2.1 Examples

We include a few examples comparing functions f and g to illustrate how to prove/verify the asymptotic relationship given the definitions above.

1. Let $f(n) = 2n + 20$ and $g(n) = n^2$.

Claim: $f = O(g)$

Proof: $2n + 20 \leq n^2$ for all $n \geq 5$ (So Definition 1 is satisfied for $C = 1$, $N = 5$).

Note that many combinations of C , N can satisfy the definition. For example, we could have written:

$$\begin{aligned} f(n) &= 2n + 20 \\ &\leq 2n + 20n && \text{(for all } n \geq 1) \\ &\leq 2n^2 + 20n^2 && \text{(for all } n \geq 1) \\ &= 22n^2 \end{aligned}$$

which satisfies the definition using $C = 22$, $N = 1$.

In general, larger C values will correspond to smaller N values, and vice versa.

2. Let $f(n) = 2n^2 + 5$ and $g(n) = n^2 + 2n + 25$.

Claim: $f = \Theta(g)$

Proof: first observe $f(n) \leq 2g(n) = 2n^2 + 4n + 25$ for all $n \geq 1$. For the other direction, we have

$$\begin{aligned} f(n) &= 2n^2 + 5 = n^2 + n^2 + 5 \\ &\geq n^2 + 2n + 25, \end{aligned}$$

where the last inequality holds when $n^2 + 5 \geq 2n + 25$, which is true for all $n \geq 5$. So $f = O(g)$ and $f = \Omega(g)$, and thus $f = \Theta(g)$.

2.2 Orders of Growth

There are some standard orders of growth that you will need to familiarize yourself with until they are second nature. We first state and remark on a few standard types of growth, and then give some additional “rules” that will help you be able to prove/verify asymptotic relationships more intuitively and quickly.

- **Polynomial growth:** $n, n^2, n^3, \dots, n^k (k \geq 0)$.

When n doubles, the output is multiplied by a constant: $(2n)^k = 2^k \cdot n^k$.

- **Exponential Growth:** $2^n, 3^n, e^n, b^n (b > 1)$, etc.

When n doubles, the output is squared: $b^{2n} = (b^n)^2$.

For all $b > 1$, and $m, n > 0$, the following rules hold:

$$\begin{aligned}b^{m+n} &= b^m \cdot b^n \\(b^m)^n &= b^{mn} \\b^n &= e^{n \log b}\end{aligned}$$

- **Logarithmic growth:** $\log_2 n, \log n, \log_b n (b > 0)$.

When n doubles, the output increases by an additive constant: $\log(2n) = \log n + \log 2$.

Note that logarithms are the inverse functions of exponentials. For any $b > 0$:

$$\log_b n = x \iff b^x = n.$$

By default, \log refers to $\log_e = \ln$, but we will often use \log_2 (ask yourself why this is often true in computer science contexts?) However, note that for any $a, b > 1$:

$$\log_b n = \frac{\log_a n}{\log_a b}.$$

Because $\log_a b$ is a constant wrt n , $\log_b n = O(\log_a n)$. Since this holds for all constants a, b , it follows that $\log_b n = \Theta(\log_a n) = \Theta(\log n)$. This is why we usually write \log without being too concerned about the specific (constant) base.

Also note that $\log(mn) = \log m + \log n$ for $m, n \geq 1$.

In general, exponentials dominate polynomials which dominate logarithms. Here, “dominate” means “will have larger function value when n is sufficiently large.” When using asymptotic notation, it will be helpful to build these notions into your intuition. We summarize these and **other rules to help simplify calculations** when reasoning about asymptotic relationships:

- multiplicative constants can be omitted: $365n^2$ is equivalent to n^2
- n^a dominates n^b if $a > b$ (read as: $n^a = \Omega(n^b)$). For example: n^2 dominates n .
- any exponential dominates any polynomial: b^n dominates n^{10} for $b > 1$
- any polynomial dominates any logarithm: $n^{0.1}$ dominates $(\log n)^{10}$.

As an exercise, you should convince yourself that each of these statements is true.

2.3 Recurrences and Counting

We also state some standard recurrences and their corresponding asymptotics. Assume for all $f : \mathbb{N} \rightarrow \mathbb{N}$ below that $f(0) = f(1) = 0$. Again, you should verify as an exercise that each of the asymptotic characterizations is correct:

- if $f(n) = f(n - 1) + 1$, then $f(n) = \Theta(n)$.
- if $f(n) = f(n - 1) + n$, then $f(n) = \Theta(n^2)$.
- if $f(n) = 2f(n - 1) + n$, then $f(n) = \Theta(2^n)$.
- if $f(n) = f(n/2) + 1$, then $f(n) = \Theta(\log n)$.

Finally, throughout the course we will need to know some combinatorial quantities associated with sets. To that end, let S be a set with n elements, and recall the following:

- Factorials: $n! = n \cdot (n - 1) \cdots 2 \cdot 1$

This is the number of *permutations* of the elements of S .

- 2^n : is the number of subsets (of any size) of S .
- n^2 : is the number of *ordered pairs* of elements of S .

Recall the set of ordered pairs of S can be written as $\{(i, j) : i, j \in S\}$.

- $\binom{n}{2} = \frac{n(n-1)}{2}$: is the number of *unordered pairs* of elements of S .

The set of unordered pairs can be written as $\{\{i, j\} : i, j \in S, i \neq j\}$.

- $\lceil \log_2 n \rceil$: the number of times needed to halve n to reach 1.

3 Graphs

We now review some basics on graphs, which will be prevalent in many of the problem settings in future lectures. For a more comprehensive review (and in particular for a refresher on representing graphs in computer memory), see [KT, Ch3]. We start by recalling some basic definitions, and then we will review a variety of special types of graphs (complete, connected, cycles, trees, bipartite).

Main Definition: A graph $G = (V, E)$ is composed of:

- A collection of vertices $V = \{1, \dots, n\}$
- A collection of edges $E \subseteq V \times V$ between some vertices

Figure 1 shows a graph with 7 vertices and 12 edges.

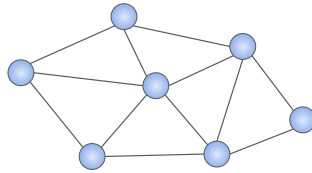


Figure 1: A graph $G = (V, E)$ with 7 vertices

Unless indicated otherwise, we will assume for graphs $G = (V, E)$ that

- Edges are *undirected*: If $(i, j) \in E$, then $(j, i) \in E$. Also write $\{i, j\} \in E$.
- No self-loops: $(i, i) \notin E$.
- Edges are *unweighted*: All $(i, j) \in E$ have weight 1.

Observe that if a graph $G = (V, E)$ has n vertices, then the maximum number of edges in G is

$$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)}{2}.$$

In particular, this says that the number of edges in a graph is at most quadratic in n (i.e., $O(n^2)$). Graphs that actually have $\binom{n}{2}$ edges are known as *complete graphs*.

Complete Graphs: Let K_n denote the complete graph on n vertices, where every pair of vertices shares an edge. Figure 2 shows the sequence of complete graphs for $n = 2 \dots 6$. We sometimes say that a complete graph is *fully connected*.

Connected Graphs: In general, we say a (not-necessarily complete) graph $G = (V, E)$ is *connected* if *every* pair of vertices $i, j \in V$ is connected via a sequence of edges:

$$(i_0, i_1), (i_1, i_2), \dots, (i_{n-1}, i_n) \in E$$

with $i_0 = i$ and $i_n = j$. We call such a sequence of edges a *path* from vertex i to j . If there exists some $i, j \in V$ for which no such path exists, then the graph is *not connected*. Figure 3 shows an example of this.

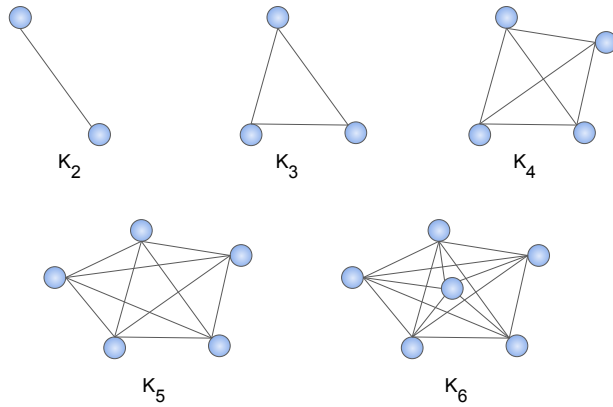


Figure 2: Complete graphs K_n for $n = 2 \dots 6$.

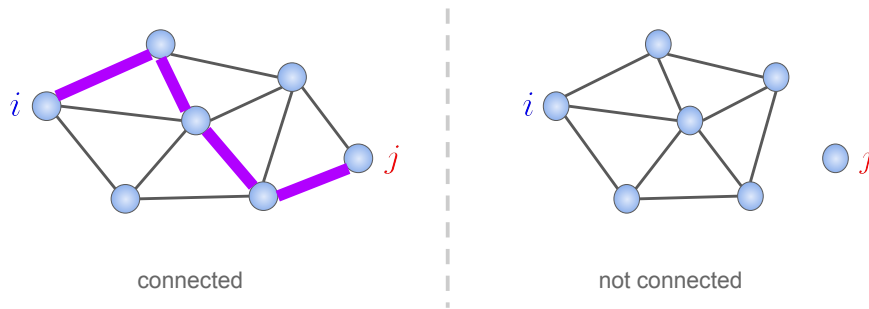


Figure 3: A connected graph (left), and a disconnected graph (right).

Cycle Graphs: Let $C_n = (V, E)$ denote the *cycle graph* on n vertices. For $n \geq 2$, and letting $V = \{1, \dots, n\}$, the set E is defined by

$$E = \{\{i, i + 1\} : 1 \leq i \leq n - 1\} \cup \{\{1, n\}\} .$$

Figure 4 shows the sequence of C_n graphs for $n = 2 \dots 6$. Note that a graph can be *drawn* in multiple ways, and each C_n in the figure can be re-visualized as a “ring” where every pair of adjacent vertices on the ring shares an edge.

In general, we say a graph with n vertices *contains a cycle* if some C_k ($3 \leq k \leq n$) exists as a *subgraph* of G .

Trees: A graph $G = (V, E)$ is a *tree* if G is *connected* and has *no cycles*. Figure 5 shows an example of a tree (left) and a graph that is not a tree since it contains a cycle (right).

We sometimes call a tree a *spanning tree*, and a tree on n vertices has exactly $n - 1$ edges (this is a question on PS01). If a graph G with no cycles (sometimes called *acyclic*) but is *not* connected is called a *forest*.

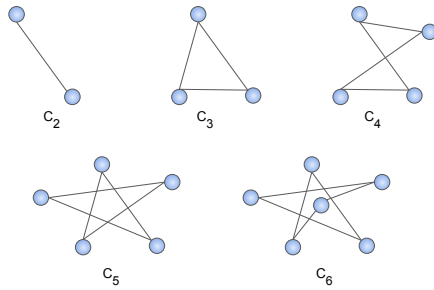


Figure 4: Cycle graphs C_n for $n = 2 \dots 6$.

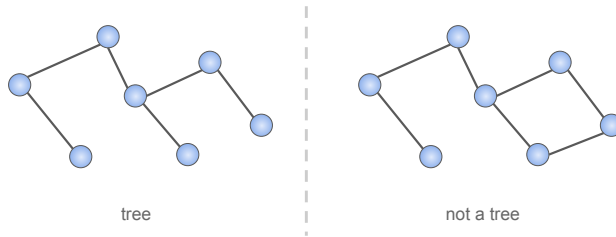


Figure 5: Example of a tree (left) and not a tree (right).

Bipartite Graphs: A graph $G = (V, E)$ is *bipartite* if the vertices can be partitioned into disjoint subsets $A \cup B = V$ such that every $e \in E$ connects a vertex in A to a vertex in B . Equivalently, no two vertices in A share an edge, and no two vertices in B share an edge. Figure 6 gives an example of a bipartite graph (and also a graph that is not bipartite).

A graph G is bipartite if and only if it has no cycles of odd length (this is a problem in PS01). In general, to prove that a graph is bipartite, it is sufficient to identify the sets A, B and show that no pair of vertices within the same set share an edge. To prove a graph is not bipartite, it suffices to identify an odd cycle. (Can you identify an odd cycle in the graph on the right in Figure 6?)

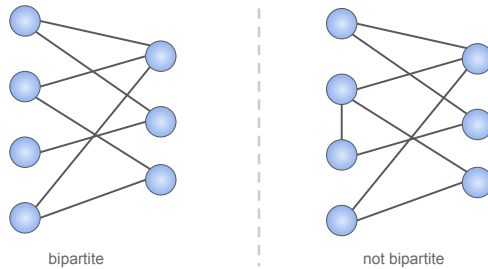


Figure 6: Example of a bipartite graph (left) and not a bipartite graph (right).

Complete Bipartite Graphs: A complete bipartite graph, denoted by $K_{m,n}$, is a bipartite graph on $m + n$ vertices, with every vertex in a set A of m vertices sharing an edge with every vertex in a set B of n vertices, and no other edges. Figure 7 illustrates the graphs $K_{3,1}$, $K_{3,2}$, and $K_{4,3}$.

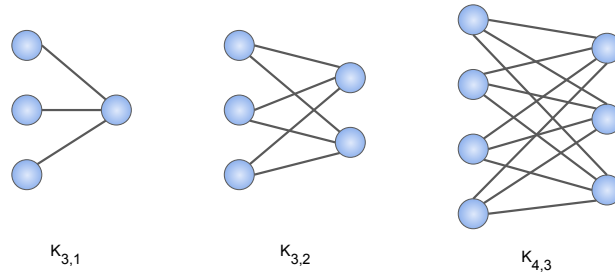


Figure 7: Examples of complete bipartite graphs $K_{m,n}$.

Matchings in Bipartite Graphs: Let G be a bipartite graph $G = (V, E)$ on $2n$ vertices with a partition $A \cup B = V$, both of size n (i.e., $|A| = |B| = n$). A *perfect matching* on G is a set of n edges from E such that no two edges in the set share a common vertex, (and every vertex in V is included in exactly one edge).

Not every bipartite graph G has a perfect matching, and Figure 8 shows an example of this. In general, Hall's Marriage Theorem gives conditions for determining when a graph *does* contain a perfect matching, but we will not cover this result in the course.

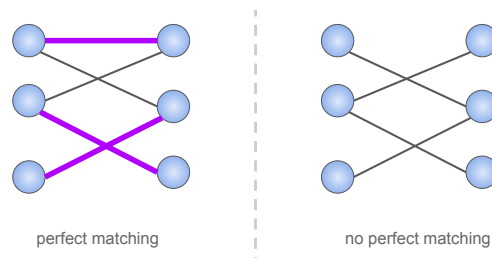


Figure 8: Examples of bipartite graphs with (left) and without (right) a perfect matching.

Matchings in Complete Bipartite Graphs: Let $K_{n,n}$ denote the *complete* bipartite graph (using the definition from above) on $2n$ vertices. Figure 9 shows several examples of these graphs. Complete bipartite graphs always contain a perfect matching, and in fact *many different* such matchings exist. In particular, there are $n!$ distinct perfect matchings (since fixing the order of vertices in the left part of the graph, there are n possible matching options from the right part for the first vertex, $n - 1$ remaining options for the second vertex, etc.).

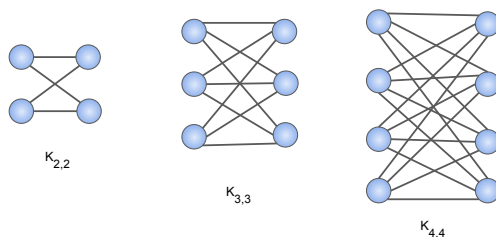


Figure 9: Examples of *complete* bipartite graphs $K_{n,n}$.

4 Introduction to the Stable Matching Problem

Our first problem setting is the *Stable Matching Problem*, which we only briefly began introducing in this lecture (see the Lecture 3 notes for the continued introduction).

First, recall from our definition of perfect matchings in complete bipartite graphs that no restrictions are placed on which vertices can be paired together to form a matching. However, suppose that vertices on each “side” (each set of the bipartition) have *preferences* over the n vertices in the opposite group (who they’d like to be matched with). Figure 10 shows an example of preferences on a $K_{3,3}$ graph. In the figure, vertex A prefers being matched to vertex 1 to vertex 2 to vertex 3. The preferences associated with all other vertices can be interpreted similarly.

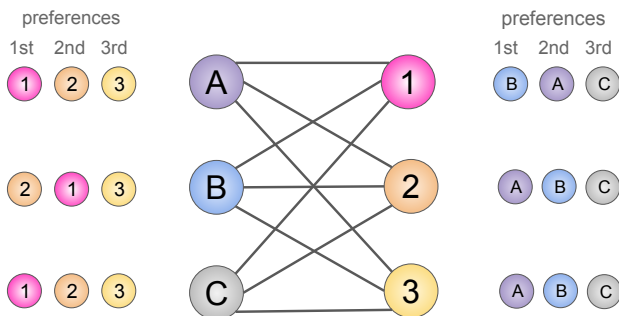


Figure 10: Example of $K_{3,3}$ with preferences

Given these set of preferences, the question is whether we can find a perfect matching that makes “everyone happy”? The next lecture will begin by giving some historical background and motivation for this problem, as well as introduce our notion of “happiness” in this setting.