Lecture 14: Graph theory (formal)

We now have all the tools to formally set up graph theory. But before we discuss graphs – one last topic with regards to order relations.

Topological sort

The idea of *sorting* a set of jobs/objects is an important component in several CPRE applications, including instruction scheduling in CPUs, compiler design, and concurrency control. In each of these cases, a central controller has a list of *jobs*, and needs to to make decisions about how to schedule different jobs in a consistent manner. The idea is that sorting it *correctly* is important: if Job #1 requires (as input) the output of Job #2, then necessarily #2 must be scheduled such that it is completed before #1 is launched; else disaster ensues.

The catch is that usually in systems, a precise schedule is not given a priori — only a rough specification. It is up to the designer to figure out the rest! That's where Topological Sort (top-sort) for short comes into play.

Top-sort is a way to design a consistent schedule (i.e., a **total order** in which the jobs are performed) given a partial schedule (i.e., a **partial order**).

Formally, top-sort is defined as the following problem:

> given a set of objects and a partial order relation R, produce a total order (i.e., sort the elements) that is compatible with R.

The algorithm for top-sort is surprisingly simple:

1. Draw the Hasse diagram of the partial order relation and set $i = 0$.
2. Pick any minimal element from the Hasse diagram, copy in position i in the total order, remove (pop) from Hasse diagram, increment i.
3. Repeat Step 2 until no more elements remain

Here is a toy (but practical) application. The typical CPRE course schedule has the following prerequisites for different courses:

- ComS 227 for ComS 228
- ComS 228 for CPRE 288
- ComS 228 for CPRE 310
- CPRE 288 for CPRE 310
- CPRE 288 for CPRE 381
- CPRE 310 for CPRE 308
- CPRE 310 for ComS 311

Assuming that a student takes courses one at a time, the goal is to design a consistent (linear) order of the courses such that all prerequisites are satisfied at the time of taking any particular course.

The way to solve this is as follows: (1) model the prereqs as a partial order relation, and draw the directed graph; (2) convert it into the Hasse diagram representation; (3) apply the above algorithm.

We won't do the entire working of the above algorithm (that is left to you as an **exercise**), but one of the outputs of applying top-sort on the above is given as follows (abbreviating course names by numbers):

227, 228, 288, 381, 310, 308, 311

A perfectly valid (but distinct) output would be:

227, 228, 288, 310, 308, 311, 381

This is because the minimal element in a Hasse diagram need not be unique, so different outputs for top-sort may emerge; all that matters is that the order in any given output list respects the constraints given by the input specifications. See if you can produce other potential top-sort outputs for the above

Graphs

We began our discussion on sets, functions, relations, etc via an informal introduction to graphs and their applications. We will now use these mathematical tools to revisit graphs, except in a more formal manner.

A (directed) graph G is a set V, together with a set E that is a subset of the Cartesian product $V \times V$. The elements of V are called *nodes* while the elements of E are called edges. Concisely, we use the notation $G = (V, E)$.

Observe that the set of edges is nothing but a relation defined from V to itself. This mirrors our intuition that graphs and relations are morally the same.

If the relation defined above is *symmetric*, then we call the graph *undirected*. By convention, the edge between nodes v_1 and v_2 will only be counted once. We use the set notation $\{v_1, v_2\}$ to denote an undirected edge between nodes v_1 and v_2 since the order of the nodes do not matter.

Simple graphs are special classes of undirected graphs where no self-loops or multiple edges between nodes are allowed.

The *degree* of a node in a simple graph is the number of edges in the graph touching that node.

In a directed graph, there are two notions of degree for each node: the *in-degree* (number of incoming edges) and the *out-degree* (number of outgoing edges).

Given any graph $G = (V, E)$, we can *traverse* this graph by hopping from node to node along the given edges. There are different types of graph traversals:

- A *walk* is any sequence of hops along a given graph. If the start point is u, and the end point is w, then any walk can be represented as:

$$ue_1v_1e_2v_2 \dots e_nw$$

  where $e_1, e_2, \dots, e_n$ denote the different edges traversed. In a generic walk, both edges and nodes can be visited one or more times.

- A *path* is any walk that does not contain repeated edges. (However, nodes can be repeated.)
- A *simple path* is any path that does not contain repeated vertices.

Representing graphs

It is always useful to draw a graph in terms of a picture (where nodes corresponds to dots and edges correspond to arrows/lines between dots). But drawing graphs can become cumbersome, especially for very large graphs. For that, we need more formal methods.

There are at least 3 different ways to mathematically write down the description of any given graph:

- Edge lists: just enumerate all the edges in the graph.
- Adjacency lists: loop through all the nodes, and for each node $i \in V$, store an array of vertices that i points to (in the case of directed graphs) or adjacent to i (in the case of undirected graphs).
- Adjacency matrices: if n is the number of nodes, construct an $n \times n$ array (or matrix) A. The $(i, j)^{th}$ entry of this array is 1 if node i points to node j, and 0 if not.

All three representations are equivalent. Observe that lists take lesser memory to store than matrices.

There is no *unique* way to represent a given graph. Indeed, it is difficult to figure out when two graphs are referring to the same thing. Checking for equivalence of graphs is called the *graph isomorphism* problem, which we will not discuss very much; see Section 2.6.2 of the textbook if you are interested.

Degree theorems

The degree of a node in a given graph is a simple quantity to compute, but conveys a considerable amount of structural information in the graph. Degrees of nodes in graphs also satisfy a number of interesting properties, which we prove as Theorems below.

First, we prove the *Degree Theorem for undirected graphs*. It states that:

> In any undirected graph,
> $$\sum_{v \in V} \deg(v) = 2|E|.$$

This can be proved using counting in two ways. First, let us focus on the left hand side. For any fixed v, $\deg(v)$ denotes the number of edges connected to v. Therefore, as we enumerate all nodes and add up their degrees, we are listing all the edges present in the graph. However, notice that each edge is counted exactly twice, since every edge is incident to exactly two nodes. Therefore, the sum of the degrees of the vertices is twice the number of edges. This ends the proof.

Second, we prove the *Handshaking Lemma*. It states that:

> In every party, the number of people who have shaken hands an odd number of times is even.

This can be proved using the Degree Theorem. Model people as a set of nodes V, and handshakes as edges E. Then, we get a graph $G = (V, E)$. The degree of each node v is the number of handshakes that person v makes with other people. We partition the people in V into two groups, V_o and V_e, depending on whether people have shaken hands an *odd* or an *even* number of times. Splitting the left hand side of the equation in the Degree Theorem into two parts, we get:

$$\sum_{v \in V_o} \deg(v) + \sum_{v \in V_e} \deg(v) = 2|E|.$$

Rewriting, we get:

$$\sum_{v \in V_o} \deg(v) = 2|E| - \sum_{v \in V_e} \deg(v).$$

The right hand side of the equation is even, since $2|E|$ is even and $\sum_{v \in V_e} \deg(v)$ is a sum of even degrees (by definition of V_e), which is also even. Therefore the left hand side of the equation is even. However, the left hand side is the sum of a certain set of *odd* numbers, and can be even only if the number of elements in that set is even. Therefore, the cardinality of V_o is even (or in words, the number of people who have an odd number of handshakes is even.) Done!

Third, we prove a second Degree Theorem for the special case of directed graphs. In such cases, where edges have arrows, there is a slightly more subtle notion of degree: depending on how many arrows are pointing inwards or outwards from a given vertex, we can define an *in-degree* $\deg^+(v)$ as well as an *out-degree* $\deg^-(v)$. The *Degree Theorem for directed graphs* states that:

In a directed graph,

$$\sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |E|.$$

This is again a consequence of the fact that each edge has precisely one initial vertex and a terminal vertex, and therefore counting the edges will involve adding up all of the in-degrees, or equivalently, all of the out-degrees.