CS 2800: Discrete Math Sept. 26, 2011

Lecture

Lecturer: David Kempe *Scribe: Eunsol & June*

1 Directed v. Undirected Graphs

Today we introduced directed and undirected graphs. Graphs are a very powerful representation in computer science, as almost anything can be represented by graphs. Will provide examples of each kind and explore types of questions that can be answered.

1.1 Examples of Undirected Graphs

Undirected graphs have edges that represent symmetric relationships. Ie, if you are friends with someone, then they are friends with you. Examples naturally follow as:

1. Friendship Networks
2. Collaboration Networks
3. Roads connecting cities
4. Mathematical models
5. Computer Networks

1.2 Examples of Directed Graphs

Directed graphs have edges that do not represent symmetric relationships. Eg, if Jane is better at chess than Bob, then Bob is not better at chess than Jane and the edge will be directed to represent that. Examples follow as:

1. Sent Emails/ dialed phone calls
2. Transportation Networks - airlines(jet stream), trains(one way tracks), etc
3. Infection and Transmission of Disease - spread of epidemics
4. Finite State Machines
5. Predator and Prey Relationships
6. Boss Hierarchy

Note: Any undirected graph can be turned into a directed graph, just represent each edge in the undirected graph with two directed edges in a directed graph
Note: Depending on what you are representing, certain constraints will be properties of the representative graph. For example, in predator-prey graphs, you would not expect a triangle. In a boss hierarchy graph, you would expect the graph to be a tree (ie, contain no cycles).

1.3 Questions

For each of graphs, we can pose some interesting questions:

Questions for undirected graphs:

· Friendships: Who are the most influential people?

· Collaboration networks: What are the most useful collaborations?

· Roads between cities: What is the quickest path between city A and city B?

· Mathematical models: How does an earthquake wave travel through the earth?

· Computer Networks: How will your ip packet be routed?

Questions for directed graphs:

· Emails/phone calls: How does information travel?

· Transportation networks - airlines, trains, etc: How should the planes be used to cover as many destinations as possible?

· Infection and transmission of disease - epidemics: If an epidemic, like swine flu, is starting, how can we change the graph to minimize the infections?

· Finite State Machines: What is the set of all possible solutions?

· Predator and prey relationship: When will an animal's food supply disappearing?

· Boss hierarchy: How much redundancy is in a company's management system?

While these questions are application specific, they are, in essense, also examples of abstract cs problems. The following questions are not directly associated with a physical problem, but their solutions are handy in more advanced cs topics:

· Cover a graph. Find a cover set such that for every node, at least one of its neighbors is in the cover set.

· Find the shortest path that visits all nodes.

· Color a map with as few colors as possible, such that no two neighboring countries have the same color.

There is an advantage to studying abstract problems. Often times solving specific application problems just involves breaking down and transforming the application to fit into already studied abstract problems and their solutions.

Definition 1. *Graph, Nodes, and Edges*
A graph $G = (V, E)$ consists of V, a set of nodes, and E, a set of edges. Where if nodes $u, v \in V$ are connected in the graph, there there exists edge $e = (u, v) \in E$.

Corresponding to wether or not the graph G is un/directed, edges can be directed or undirected. When directed, $e = (u, v)$ and $e = (v, u)$ are different. When the graph is undirected, the two are equivalent, and if one direction exists in the graph, the other direction is implied.

You may come across many more ways to name edges and nodes, especially when working on applications. However, the widely accepted convention is to name these as nodes and edges.

convention	alternatives
nodes	vertices, points, people, cities
edges	lines, arcs, friendships, roads

So we have built up nodes and edges to create graphs, but we can attach more information. Edges can have weights. For example in building a graph of cities and their connecting roads, the weights can correspond to distance between cities. For nodes, we can attach properties to the nodes, such as city size. Often times the more information the more involved the questions we can ask.

Graphs have to be built in a way that reflects the problem. Suppose we have the issue of finding which sets of chemicals produce dramatic reactions. We could take all chemicals as nodes and add an edge if the two connecting chemicals react. But chemical reactions are not linear. If I find a set of 3 chemicals that all pairwise react, the combination of all 3 may not cause a reaction. Similarly, there exists sets of chemicals that will not react until a final ingrediant is added. Having a graph of the pairwise reactions will not account for these. An immediate response is, well, build a graph to represent all combinations of chemicals (a hypergraph). However, this should only be an absolute last resort(June: "Personally, I have never had to go that extreme.") The caution is that, while intriguing, they quickly grow beyond computable and comprehensible bounds.

A more likely complexity with graphs are self-edges.

Definition 2. *Self-edge, is an edge from a node, u, to itself. Ie., $e = (u, u)$.*

A common example of self-edges is a website that links to itself. Consider page-rank, used to sort websites returned in Google searches. Page rank is roughly ranking websites by the number of pages that link to said website. Self-edges are easy to create and bump a primative page rank algorithm into more highly valuing websites with self-edges.

Another complexity that can occur with the edges of a graph are multiple copies of an edge:

Definition 3. *Multigraph, if we allow multiple copies of edges and the number of copies is called the multiplicity of the graph.*

In multigraph, there can be multiple edges connecting the same pair of nodes. Ie., $e_1 = (u, v)$, $e_2 = (u, v)$, $e_3 = (u, v)$ are distinct, often by having different weights. In most cases, we implicitly assume graphs are not multigraph.

Often times the algorithms we can run over graphs depend on what type of graph we are dealing with, multi/non-multi, directed etc. We now provide some definitions that allow us to clarify in future what types of graphs there are.

Definition 4. *Adjacent nodes, or neighbors.*
If and only if there is an edge between a pair of nodes u, v, nodes u, v are neighbors and adjacent to one another.

Definition 5. *Degree.*
The degree of a node is the number of edges containing that node. In directed cases, in-degree of a node is the number of edges from that starts from the node, and the out-degree of a node is the number of edges that ends at the node.

Lemma 6. *The Handshake Lemma*
Given an undirected graph, with edges, $G = (V, E)$ where $|E|$ represents the number of edges in the graph.

$$\sum_{v \in V} deg(v) = 2|E|$$

Proof. Intuitively, each edge, $e = (u, v)$, contributes twice to the sum of degrees, once for u and once for v. Formally, we can change of the sums to be:

$$\begin{aligned}\sum_{v\in V} deg(v) &= \sum_{u\in V}\sum_{u\in V} 1, \text{ where there is an edge between u and v} \\ &= \sum_{e\in E}\sum_{v\in V, incident\ on\ edge\ e} 1 \\ &= \sum_{e\in E} 2 \text{ , the number of ends to an edge is 2} \\ &= 2|E|\end{aligned}$$

□

Corollary 7. *The number of vertices of odd degree is even.*

Note: this is because the degrees must sum up to something even.