Simple Graphs and Trees

Benjamin Cosman, Patrick Lin and Mahesh Viswanathan

Fall 2020

TAKE-AWAYS

- A *simple graph* is a set of vertices along with a set of (undirected) edges with no self-loops.
- Most concepts from directed graphs, like walks and paths, can be used almost unchanged for simple graphs. A cycle in a simple graph must be of length at least 3.
- The *degree* of a vertex is the number of edges *incident* to it. Two vertices connected by an edge are *adjacent*.
- A graph is *connected* if there is a path from each vertex to each other vertex.
- A connected acyclic graph is called a *tree*; its degree-1 vertices are *leaves*.
- A tree can also be directed; edges point from *parent* to *child* away from the *root*, and the *height* is the maximum distance from the root to a leaf.
- Two graphs are *isomorphic* if they have the same number of vertices, all connected in the same way.
- Some common graphs are the n-vertex line graph L_n, the n-vertex cycle graph C_n, the $(n+1)$-vertex wheel graph W_n, and the n-vertex complete graph K_n.
- A *k-coloring* in a graph is an assignment of k colors to vertices so that adjacent vertices always have different colors.
- A graph's *chromatic number* χ is the smallest number of colors needed to color it.
- A *bipartite* graph is one whose vertices can be split into two non-empty groups and all edges go from one group to the other.
- A *matching* in a bipartite graph is a way of pairing up all vertices in one group with adjacent vertices in the other.
- Colorings and matchings are useful for solving different kinds of allocation problems.

Simple Graphs and Trees

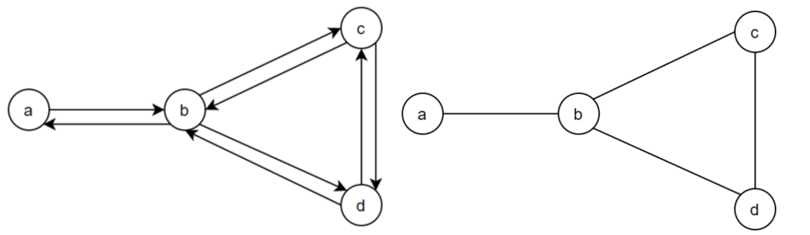

Figure 1: Left: A digraph of a symmetric, irreflexive relation. Right: The simple graph that conveys the same information.

We saw last week that any relation $R \subseteq A \times A$ can be represented by a directed graph. This week we will focus on symmetric, irreflexive relations - that is, relations where the directed graph would have no self-loops and each edge would have a matching edge going in the other direction. We could continue to use directed graphs to model these relations, but it is more convenient to replace all these pairs of edges with single, *undirected* edges, as in Figure 1.

Definition 1. A *simple graph* is a set of vertices $V(G)$ and a set of edges $E(G)$, where each edge $\langle u - v \rangle$ connects two different vertices u and v (there are no self-loops).

Most of our definitions for directed graphs work unchanged for simple graphs. A walk is still an alternating list of vertices and connecting edges, a path is still a walk with no repeat vertices, etc. One change is that a *cycle* in a simple graph must have length at least 3. This is because a cycle of length 1 is impossible now that we are disallowing self-loops entirely, and closed walks of length 2 are too uninteresting to call cycles since crossing *any* edge back and forth creates a length-2 closed walk.

Definition 2 (Degrees)**.** Two vertices connected by an edge are *adjacent*, and the edge is *incident* to the two vertices. The *degree* of a vertex is the number of edges incident to it.

Example 3. In the simple graph from Figure 1, vertex b has degree 3.

Definition 4. A graph is *connected* if there is a path from each vertex to each other vertex. A graph is a *tree* if it is both connected and acyclic. Degree-1 vertices in a tree are called *leaves*. (see Figure 2)

In a tree G, the following properties hold:

- There is exactly one path between any pair of vertices
- $|E(G)| = |V(G)| - 1$

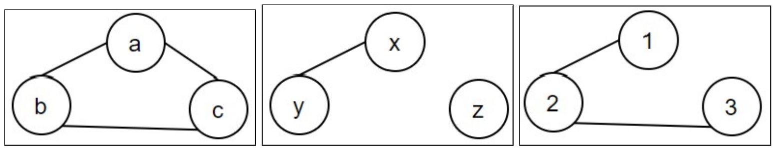

Figure 2: Left: A connected and cyclic graph. Center: A graph that is acyclic and not connected. Right: A tree (acyclic and connected) with 1 and 3 as leaves.

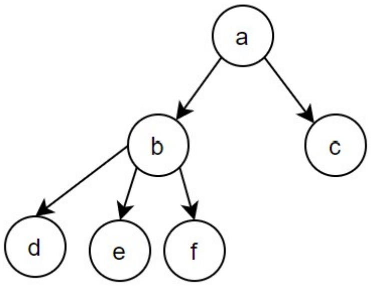

Figure 3: A (directed) tree of height 2. The vertex at the top is the root, and e.g. b is the parent of children d, e, and f.

Definition 5. A *directed tree* (Figure 3) is a digraph where there is one *root* vertex with indegree 0 (i.e. no incoming edges), and exactly one path from that root to each other vertex. Equivalently, a directed tree is what you would get if you designated one vertex in a simple tree to be the root, and then directed every edge to point from the vertex closer to the root to the one farther away. A directed tree is usually just called a tree - you have to figure out from context if the tree being discussed is directed or simple. Each vertex is called the *parent* of the vertices it has edges pointing to, and those vertices are its *children*. The height of a tree is the maximum distance from the root to a leaf.

Graph Isomorphism

Until now $V(G)$ has always been made explicit, i.e. we've always labeled the vertices of our graphs. However, we often want to talk about properties of a graph that depend only on its structure, in which case we don't care about the vertex labels. For example, the tree in Figure 2 would be a tree regardless of what the vertices are called. Thus we want an easy way to classify graphs as "having the same structure".

Definition 6 (Isomorphism)**.** An *isomorphism* between graphs G and H is a bijection $f : V(G) \to V(H)$ such that all edges are preserved, i.e. $\langle u - v \rangle$ is an edge in G iff $\langle f(u) - f(v) \rangle$ is an edge in H. Two graphs are *isomorphic* if there exists an isomorphism between them.

Example 7. G and H in Figure 4 are isomorphic. One isomorphism

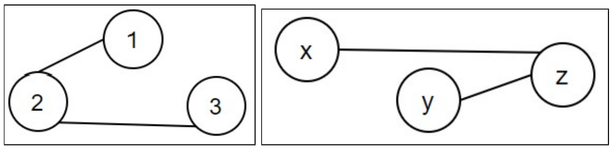

Figure 4: Two isomorphic graphs

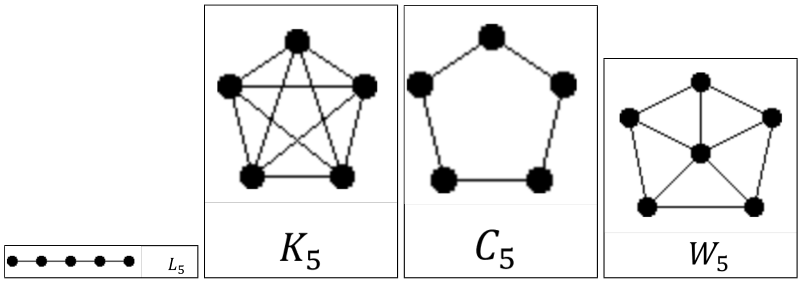

Figure 5: Examples of our common named graphs when $n = 5$. Notice that W_5 has 6 vertices.

would be $f(1) = x, f(2) = z, f(3) = y$. A function that would *not* work as an isomorphism would be $g(1) = x, g(2) = y, g(3) = z$, because edges are not mapped correctly: $\langle 1 - 2 \rangle$ is an edge in G, but $\langle g(1) - g(2) \rangle = \langle x - y \rangle$ is not an edge in H.

When vertex labels don't matter, we will frequently refer to a class of isomorphic graphs by a single name. For example, since both graphs in Figure 4 are 3 vertices connected in a line, we might refer to them both (and any other graph isomorphic to them) as "the graph L_3", rather than specifying which exact one we're talking about. Similarly, we frequently don't label the vertices on a graph at all if it doesn't matter in context.

Here are a few graphs whose names you will need to know:

Definition 8 (Specific named graphs)**.** See Figure 5 for examples of each:

- The line graph L_n is n vertices connected in a line.
- The complete graph K_n is n vertices and *all* possible edges between them.
- For $n \geq 3$, the cycle graph C_n is n vertices connected in a cycle.
- For $n \geq 3$, the wheel graph W_n is C_n with one extra vertex that is connected to all the others.

Colorings and Matchings

Simple graphs can be used to solve several common kinds of constrained-allocation problems. For example, suppose you want to schedule

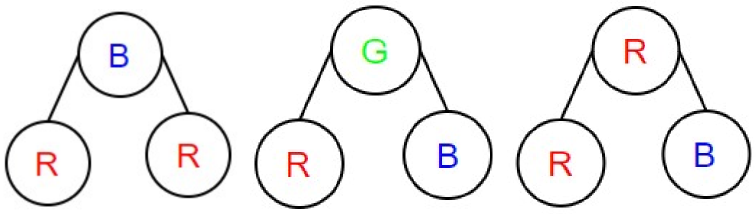

Figure 6: Three attempts at coloring L_3 using colors from $\{R, G, B\}$. Left: A 2-coloring. Center: A 3-coloring. Right: Not a coloring, because there are adjacent Red vertices (in our example, we'd be scheduling two overlapping classes in the same room R).

classes in various rooms; two classes can use the same room as long as they happen at different times, but they can't use the same room if their times overlap. We could solve this by first drawing a graph where the vertices are classes and there is an edge between two classes if they occur at overlapping times. Then we need to assign rooms to the vertices such that the endpoints of each edge are always different. This is an example of what we call a graph coloring:

Definition 9 (Coloring)**.** A *(k-)coloring* of a graph is an assignment of (k) colors[1] to its vertices such that no two adjacent vertices have the same color. (see Figure 6)

[1] "Colors" can be drawn from any set: $\{red, green, blue\}$ is fine but so is $\{1,2,3\}$

Another class of constrained-allocation problem solvable using graphs involves pairing things up. For example, a group of classes all need to be assigned *distinct* rooms, and only certain class-room pairings are acceptable (some classes need a big room, some need an AV system, etc). To solve this, we first draw a graph with classes on the left and rooms on the right, and acceptable class-room pairings as edges. Then we search for a set of edges whose endpoints are all distinct and include all classes (Figure 7). More generally:

Definition 10. A *bipartite* graph is a simple graph G whose vertices can be split into two non-empty sets $L(G)$ and $R(G)$, such that each edge has one endpoint in each set. A *matching* in a bipartite graph is an injective function assigning to each vertex in $L(G)$ an adjacent vertex in $R(G)$.

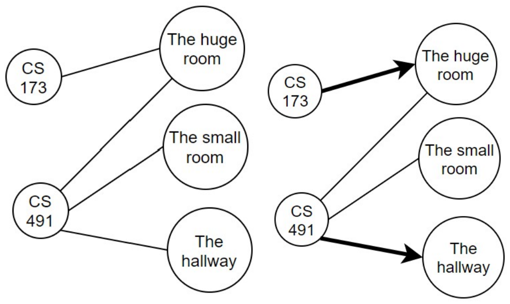

Figure 7: Left: a bipartite graph G where $L(G)$ are classes, $R(G)$ are rooms, and CS 173 needs a huge room while CS 491 can be held anywhere. Right: a matching in G assigning each class to a distinct acceptable meeting place.