Reading 12 : Graphs

Author: Dieter van Melkebeek (updates by Beck Hasti and Gautam Prakriya)

Graphs are discrete structures that model relationships between objects. Graphs play an important role in many areas of computer science. In this reading we introduce basic notions of graph theory that are important in computer science. In particular we look directed graphs, undirected graphs, trees and some applications of graphs.

12.1 Digraphs

We begin by discussing directed graphs, also known as digraphs.

The definition below uses the term multiset. A *multiset* is a collection very similar to a set, except duplicates of an element are allowed. Every set is also a multiset, but there are some multisets that are not sets. For example $\{1, 2, 3\}$ is a set and also a multiset. On the other hand, $\{1, 1, 2\}$ is a multiset, but not a set.

Definition 12.1. *A* digraph *G is a pair $G = (V, E)$, where V is a set of* vertices *and E is a multiset of elements of $V \times V$. Elements of E are called* edges.

We often use pictures to represent graphs. Each vertex is a labeled dot or a circle, and an edge is a line connecting two such dots, with an arrow indicating the direction of the edge.

Example 12.1: Consider the directed graph in Figure 12.1. We have

$$\begin{aligned} V &= \{a, b, c, d\} \\ E &= \{e_1, e_2, e_3, e_4, e_5, e_6\} \\ &= \{(a, b), (b, c), (c, d), (c, d), (d, b), (d, a)\} \end{aligned}$$

Notice that since E is a multiset, it is fine that it contains two copies of the pair (c, d). ⊠

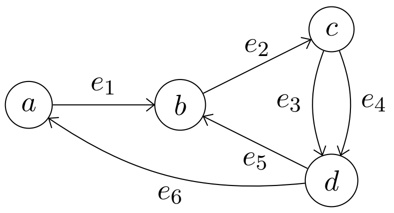


Figure 12.1: An example of a digraph.

If E is a set, then E is a binary relation on V. When this is the case, we say $G = (V, E)$ is a *simple* graph.

12.1.1 Paths

Another important notion is a path in a graph. Informally speaking, a path is a sequence of edges that "connect together".

Definition 12.2. *Let $G = (V, E)$ be a graph, and $u, v \in V$. A* path from u to v *is a sequence of edges such that the start vertex of the first edge is u, the end vertex of the last edge is v, and the start vertex of the $(i + 1)$st edge is the same as the end vertex of the ith edge. The* length *of a path is the number of edges.*

Example 12.2: We list some paths in the graph from Figure 12.1 in Table 12.1 and point out a few things.

The path P_1 is the empty sequence. Such a path has length zero and has the same start and end vertex. Following this path corresponds to starting at a and staying put.

There can be multiple paths between two vertices. For example, P_2 and P_3 are both paths from a to c. A path can also repeat edges. For example, the edge e_2 appears twice on path P_3.

Finally, observe that the path P_4 starts and ends at a. We call any non-empty path that starts and ends at the same vertex a *cycle*. ⊠

Path	Start vertex	End vertex	Sequence of edges	Length
P_1	a	a	Empty sequence	0
P_2	a	c	e_1, e_2	2
P_3	a	c	e_1, e_2, e_3, e_4, e_2	5
P_4	a	a	$e_1, e_2, e_3, e_5, e_2, e_4, e_6$	7

Table 12.1: Some paths in the graph from Figure 12.1.

12.1.2 Transitive Closure of a Directed Graph

The notion of a path also induces a relation R on V. In particular, uRv if there is a path from u to v in G. This relation is called the *transitive closure* of G.

Transitive closure is reflexive. This is because a path from $v \in V$ to itself is one where we start at v and stay put. This path has length zero and doesn't require any edges, which means it's present in every graph, even in one without any edges.

Transitive closure is not symmetric. The simplest counterexample is the graph with 2 vertices u and v and one edge (u, v). There is a path from u to v, but there is no path from v to u. We will discuss graphs for which the transitive closure is symmetric in the second part of this lecture.

Transitive closure is, as the name suggests, transitive. Indeed, let $u, v, w \in V$, and suppose there is a path $e_1, e_2, \ldots, e_r$ from u to v, and a path $f_1, f_2, \ldots, f_s$ from v to w. Then the sequence of edges $e_1, e_1, \ldots, e_r, f_1, f_2, \ldots, f_r$ is a path from u to w.

12.1.3 Directed Acyclic Graphs

A directed graph is *acyclic* if it contains no cycles. Se call such a graph a *directed acyclic graph*, or DAG for short.

Directed acyclic graphs describe many real life situations. For example, consider registering for classes. Some courses have other courses as prerequisites. We can model this relation on the set of courses using a directed graph. Each course has a corresponding vertex in the graph, and there is an edge from the vertex representing course c_1 to a vertex representing course c_2 if c_1 is a prerequisite for c_2. If there is a cycle in this graph, we won't be able to take any course whose corresponding vertex is on that cycle, or any course whose corresponding vertex lies on a path from some vertex in that cycle. Thus, the graph representing the "prerequisite" relation better be a directed acyclic graph.

Other examples of relations represented by directed acyclic graphs are dependencies between different pieces of software on a computer. Again, if there are circular dependencies, we would not be able to install some software.

The same thing goes for programming parts of a bigger project. We first need to code up the small parts before we can get the bigger parts to work. If there is a circular dependency, we may not be able to finish our program, or, at the very least, we will have to code up a significant portion of it before we can test its functionality (this often suggests that we designed the program poorly).

Example 12.3: Let's consider a more mundane problem: getting dressed. The graph of the "put this on before you put that on" relation is in Figure 12.2. Piece of clothing a is related to piece of clothing b if you need to put on a before putting on b. For example, you first need to put on your right sock before you put on your right shoe. This is indicated by an edge going from the vertex labeled "right sock" to the vertex labeled "right shoe" in Figure 12.2. Observe that the graph is acyclic. If it were cyclic, we would not be able to get dressed. ⊠

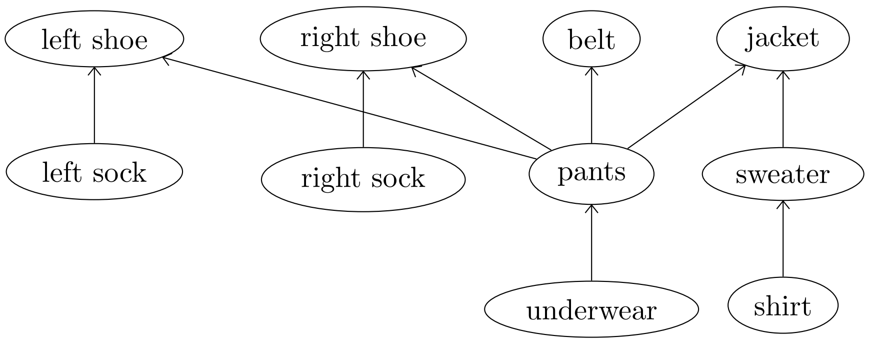


Figure 12.2: Graph of the clothing dependency relation.

A directed acyclic graph has no edges of the form (v, v) because such an edge would form a cycle of length 1. We now show that we can take any directed acyclic graph and arrange all its vertices on one line so that if $u, v \in V$ and $(u, v) \in E$, then the vertex u is "to the left" of the vertex v. In other words, all arrows in a picture representing that graph point from the left to the right. More formally, we show that every directed acyclic graph has a topological ordering.

Definition 12.3. *A* topological ordering *of a graph $G = (V, E)$ is a total order $\leq$ on V such that if $(u, v) \in E$ for two different vertices $u, v \in V$, then $u \leq v$.*

Theorem 12.4. *Every directed acyclic graph has a topological ordering.*

Before we prove Theorem 12.4, let's see an example involving the graph from Figure 12.2.

Example 12.4: Consider the rearrangement of the nodes of the graph from Figure 12.2 shown in Figure 12.3. Notice that all edges go from left to right, so we have a topological ordering of the graph from Figure 12.2.

More formally, the relation $\leq$ on the set of vertices defined by $u \leq v$ if u is to the left of v in Figure 12.3 satisfies $(u, v) \in E \Rightarrow u \leq v$ for all $u, v \in V$ such that $u \neq v$, and is a total order.

Notice that if we go through the vertices in the graph in Figure 12.3 from left to right and put on a piece of clothing when we reach its corresponding vertex, we successfully get dressed. ⊠

To prove Theorem 12.4, we need to define the following properties of vertices.

Definition 12.5. *Let $G = (V, E)$ be a graph, and let $v \in V$. The* in-degree *of v is the number of edges in E that have v as the endpoint. The* out-degree *of v is the number of edges in E that have v as the starting point.*

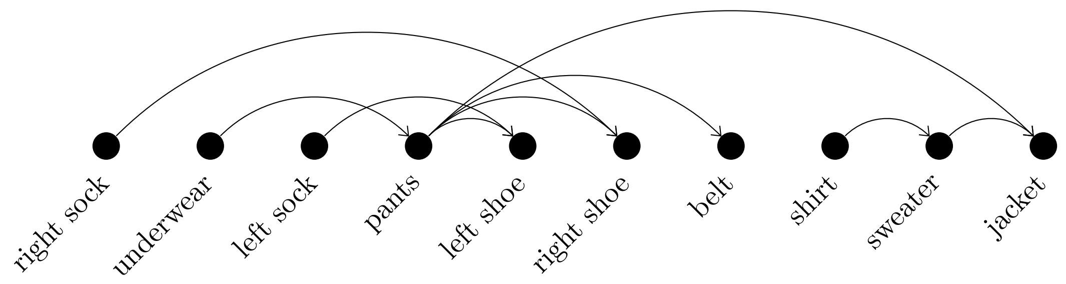


Figure 12.3: A topological ordering of the graph of clothing dependencies from Figure 12.2.

Example 12.5: Consider the graph in Figure 12.3. The vertex "right sock" has in-degree zero because there are no edges coming in, and has out-degree 1 because there is one edge leaving it (the one that goes to vertex "right shoe"). The vertex "pants" has in-degree 1 and out-degree 4. The vertex "right shoe" has in-degree 2 and out-degree 0. ⊠

Four vertices in the directed acyclic graph of Figure 12.2 have in-degree zero: right sock, underwear, left sock, and shirt. This observation is a key step towards proving Theorem 12.4.

Claim 12.6. *Every finite nonempty directed acyclic graph has a vertex of in-degree* 0.

We will prove Claim 12.6 in a moment. First let's see how it helps us prove Theorem 12.4.

Proof of Theorem 12.4. We prove by induction that every directed acyclic graph on n vertices has a topological ordering.

In the base case, we have a graph with one vertex. Its topological ordering $\leq$ is the empty relation. There are no edges in this graph, so $\leq$ satisfies $(\forall u, v \in V)\ \ (u, v) \in E \Rightarrow u \leq v$ trivially.

Now suppose that all directed acyclic graphs on n vertices have a topological ordering, and consider a directed acyclic graph G on $n+1$ vertices. By Claim 12.6, G has a vertex u of in-degree zero. Thus, for all vertices $v \neq u$ we define $u \leq v$ which guarantees that the implication $(u, v) \in E \Rightarrow u \leq v$ is satisfied by all vertices $v \in V$.

Now consider the subgraph G' of G obtained by removing u and all edges going from u. The subgraph G' is a directed acyclic graph because we obtained it from a directed acyclic graph by removing one vertex and some edges, which cannot introduce any cycles. Furthermore, G' has n vertices, so it has a topological ordering $\preceq$ by the induction hypothesis. Then for two vertices v, w different from u in G, we define $v \leq w \iff v \preceq w$.

We claim that $\leq$ is a topological ordering of G.

Consider any two vertices $v, w \in V$ such that $v \leq w$ and $w \leq v$. If both v and w are vertices of G', we also have $v \preceq w$ and $w \preceq v$ by definition of $\leq$. But $\preceq$ is antisymmetric by the induction hypothesis, so $v = w$ in that case. For the case $u = v$ (or $u = w$), recall that we defined $u \leq v$ for all vertices $v \neq u$, and did not set $v \leq u$ for any $v \in V$. Thus, we see that $\leq$ is antisymmetric.

Next, pick any three vertices $v, w, x \in V$ where $v \leq w$ and $w \leq x$. If all three are vertices of G', we also have $v \leq x$ because $\leq$ is the same as $\preceq$ on G', and $\preceq$ is transitive. Now suppose one of v, w, x is u. Then $v = u$ because no vertex $v \in V$ satisfies $v \leq u$, and $w, x \neq u$ for the same reason. Now $u \leq x$ for any vertex $x \neq u$, so the implication $u \leq w \wedge w \leq x \Rightarrow u \leq x$ is true regardless of what w is. Hence, $\leq$ is transitive.

We showed that $\leq$ is antisymmetric and transitive, so $\leq$ is an order relation. Also notice that for any pair $v, w \in V$ with $v \neq w$, we either have $v \leq w$ or $w \leq v$. If both v and w are vertices of

G', this follows because $\preceq$ is a total order, and if one of them is u, it follows because $u \leq v$ for any vertex $v \neq u$ by definition. Thus, $\leq$ is a total order. □

Now let's argue Claim 12.6. We give a proof using invariants.

Proof of Claim 12.6. We argue by contradiction. Suppose there are no vertices with zero in-degree in the graph $G = (V, E)$, and let $|V| = n$.

We construct a cycle in G in stages. At the beginning of stage i for $1 \leq i \leq n$, we have the following invariant: There are distinct vertices $v_1, \ldots, v_i \in V$ such that there is a path $(v_i, v_{i-1}), \ldots, (v_3, v_2), (v_2, v_1)$ from v_i to v_1.

When $i = 1$, we can pick any vertex as v_1. The path from v_1 to v_1 is the empty path. Thus, the invariant holds at the beginning of the first stage.

Now suppose the invariant holds at the beginning of stage $i < n$. The vertex v_i has in-degree at least 1 by assumption, so there is some vertex $u \in V$ such that $(u, v_i) \in E$. Now if $u = v_j$ for some $j \in \{1, \ldots, i\}$, this edge completes the cycle $(v_i, v_{i-1}), \ldots, (v_{j+1}, v_j), (v_j, v_i)$, and we get a contradiction with the fact that the graph is acyclic.

So assume $u \neq v_j$ for any $j \in \{1, \ldots, i\}$ (this is possible since $i < n$). Then define $v_{i+1} = u$ and move to stage $i+1$. Observe that the vertices $v_1, \ldots, v_{i+1}$ are distinct because $v_1, \ldots, v_i$ are distinct by the induction hypothesis and v_{i+1} is different from all of $v_1, \ldots, v_i$ by construction. Also, the path $(v_{i+1}, v_i), \ldots, (v_2, v_1)$ is present in the graph because the path $(v_i, v_{i-1}), \ldots, (v_2, v_1)$ is present by the induction hypothesis, and the edge (v_{i+1}, v_i) is present by construction. Thus, the invariant is maintained at the beginning of stage $i + 1$.

Now consider the situation at the beginning of stage n. There is a path $(v_n, v_{n-1}), \ldots, (v_2, v_1)$, and this path goes through all vertices of G. However, v_n has in-degree 1, so there is a vertex v_j such that $(v_j, v_n) \in E$. The edge (v_j, v_n) completes the cycle $(v_n, v_{n-1}), \ldots, (v_{j+1}, v_j), (v_j, v_n)$, so, again, we get a contradiction with the fact that G is acyclic.

Since all cases lead to a contradiction, we must reject the assumption that all vertices in G have nonzero in-degree. It follows that at least one vertex in G has in-degree zero. □

Now let's return to a fact we mentioned in the previous lecture. Consider the digraph $G = (V, E)$ that represents an order relation R. Recall that R is antisymmetric and transitive. Remove self loops (edges of the form (v, v)) from G to get a graph $G' = (V, E')$ that represents a relation R', and observe that R' is antireflexive in addition to being antisymmetric and transitive. Now let $u, v \in V$ be two different vertices. Because R is antisymmetric, at most one of (u, v), (v, u) is in E'. Also, there are no cycles in G' because we obtained G' from G by removing all self-loops, and there were no other cycles in G because G is the graph that corresponds to an order relation (a cycle $(v_1, v_2), \ldots, (v_r, v_1)$ in G would imply $v_1 R v_j$ and $v_j R v_1$ for some $v_j \neq v_1$ by transitivity, which would mean R is not antisymmetric). Then G' is a directed acyclic graph, and has a topological ordering $\leq$.

Now consider the following relation $\tilde{R}$. We have $(u, v) \in \tilde{R}$ if either $u \leq v$ in a topological ordering of G', or if $u = v$ and uRu. Thus, if uRu, also $u\tilde{R}u$. Also, if $u \neq v$ and uRv, we have $u \leq v$ in the topological ordering of G', and, therefore, $u\tilde{R}v$. Therefore, $\tilde{R}$ is an extension of R. It is also a total order because a topological ordering is a total order. Hence, we just proved the following consequence of Theorem 12.4.

Corollary 12.7. *Every order relation R can be extended to some total order $\tilde{R}$.*

12.2 Graphs

Now let's shift our attention to graphs where edges do not have direction. Sometimes people refer to such graphs as undirected graphs. We only use the term graphs, and drop the adjective "undirected".

A graph G consists of a vertex set V and an edge multiset E. We can view such a graph as a digraph where (u, v) is an edge if and only if (v, u) is an edge. We collapse those two edges into one edge $\{u, v\} \in E$ (where the curly braces indicate that order does not matter, which is consistent with the fact that the edge doesn't have a direction). If there is a self loop, we represent the edge as $\{v\}$. With this representation, E is a multiset consisting of one- and two-element subsets of V. If an edge e has a vertex v as one of its endpoints, we say e and v are *incident*.

12.2.1 Graphs Representing Relations

The fact that edges no longer have direction means that we cannot represent every relation using a graph with only one vertex for each element of the domain. An edge $\{u, v\}$ would not tell us whether uRv, vRu, or both. However, we can use a graph to represent any symmetric relation R since in that case we can interpret the edge $\{u, v\}$ unambiguously as both uRv and vRu.

It is still possible to represent an ordinary relation using a graph. We just need more vertices. Suppose R is a relation on A. We make disjoint vertex sets V_{left} and V_{right}, each labeled with elements of A. For each $a \in A$, we have vertices $a_{\text{left}} \in V_{\text{left}}$ and $a_{\text{right}} \in V_{\text{right}}$. The vertex set of G is then $V = V_{\text{left}} \cup V_{\text{right}}$. If aRb, there is an edge between a_{left} and b_{right}, and there are no other edges. The edge $\{a_{\text{left}}, b_{\text{right}}\}$ only indicates aRb, and not bRa.

We call any graph whose vertex set can be partitioned into two sets V_{left} and V_{right} such that no edge is a subset of V_{left} or V_{right} a *bipartite graph*.

12.2.2 Transitive Closure of a Graph

Observe that in a graph, there is a path from u to v if and only if there is a path from v to u (just traverse the edges on the path from u to v in reverse order). Thus, the transitive closure of a graph is symmetric. Note that transitive closure remains reflexive and transitive, so the transitive closure of a graph is an equivalence relation.

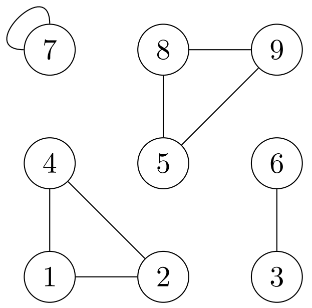
Definition 12.8. *Let $G = (V, E)$ be a graph and consider its transitive closure. Let $V_1, V_2, \ldots, V_r$ be the equivalence classes in the partition of V induced by the transitive closure. For each equivalence class V_i, the subgraph of G with vertex set V_i and edges from G that involve vertices in V_i is called a* connected component*. A graph is called* connected *if it has at most one connected component.*

Example 12.6: Consider the graph in Figure 12.4a. It has four connected components. We list them alongside the graph in Figure 12.4b. ☒

Since there is no direction to edges, we don't talk about in-degree and out-degree of a vertex in a graph. Instead, we only talk about the degree.

Definition 12.9. *Let $G = (V, E)$ be a graph, and let $v \in V$. The* degree *of v is the number of edges containing v, counting a self-loop (an edge $\{v\}$) twice. We use the notation* $\deg(v) = k$ *to denote that vertex v has degree k.*

Example 12.7: The vertex 1 in the graph in Figure 12.4a has degree 2 because it belongs to two edges, namely $\{1, 2\}$ and $\{1, 4\}$. The vertex 7 also has degree 2 because it belongs to the self-loop



(a) a graph with four connected components.

No.	Vertices	Edges
1	1, 2, 4	$\{1,2\}$, $\{1,4\}$, $\{2,4\}$
2	3, 6	$\{3,6\}$
3	5, 8, 9	$\{5,8\}$, $\{5,9\}$, $\{8,9\}$
4	7	$\{7\}$

(b) List of connected components of the graph in part (a).

Figure 12.4: An example illustrating the definition of a connected component.

$\{7\}$ which contributes 2 towards its degree. The only vertices of degree other than 2 are vertices 3 and 6, both of which have degree 1. ⊠

There is also a relationship between the sum of the degrees of all vertices of a graph and the number of that graph's edges. Before we state it, let's look at a few examples in Figure 12.5. The vertex labels in Figure 12.5 indicate the vertices' degrees. For each graph, we find the sum of the degrees of its vertices and the number of edges.

Theorem 12.10. *In every graph $G = (V, E)$,*

$$\sum_{v\in V} \deg(v) = 2|E|. \tag{12.1}$$

Proof. Consider the contribution of edge $e \in E$ to each side of (12.1).

First consider the contribution of e to the left-hand side of (12.1). If e is a self-loop $\{v\}$ for some $v \in V$, it contributes 2 to the degree of v, so its contribution to the left-hand side of (12.1) is 2. If e is not a self-loop, it has the form $\{u, v\}$ for some $u, v \in V$, and it contributes 1 to the degree of v and 1 to the degree of u for a total contribution of 2 to the left-hand side of (12.1).

Now consider e's contribution to the right-hand side of (12.1). Because e contributes 1 to $|E|$ and the right-hand side of (12.1) is $2|E|$, e contributes 2 to the right-hand side of (12.1).

We have shown that every edge gives the same contribution to both sides of (12.1), which means that (12.1) holds. □

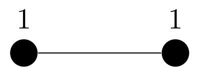
Theorem 12.10 has the following consequence.

Corollary 12.11. *The number of vertices of odd degree in a graph is even.*

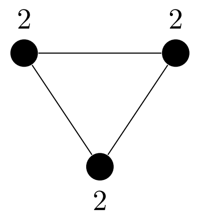
Proof. Consider the sets $V_1 = \{v \in V \mid v \text{ has odd degree}\}$ and $V_2 = \{v \in V \mid v \text{ has even degree}\}$. These sets form a partition of V, so we can write $\sum_{v\in V} \deg(v) = \sum_{v\in V_1} \deg(v) + \sum_{v\in V_2} \deg(v)$. Substituting that into (12.1) yields $\sum_{v\in V_1} \deg(v) + \sum_{v\in V_2} \deg(v) = 2|E|$, so

$$\sum_{v\in V_1} \deg(v) = 2|E| - \sum_{v\in V_2} \deg(v). \tag{12.2}$$

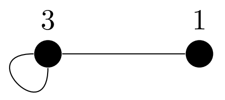
The right-hand side of (12.2) is even because $2|E|$ is even and $\sum_{v\in V_2} \deg(v)$ is a sum of even numbers, which is even. Thus, the left-hand side of (12.2), $\sum_{v\in V_1} \deg(v)$, is also even. Moreover, the latter sum is a sum of odd numbers. Only an even number of odd numbers can add up to an even number, so the left-hand side of (12.2) is a sum of an even number of terms, which implies that $|V_1|$ is even. □



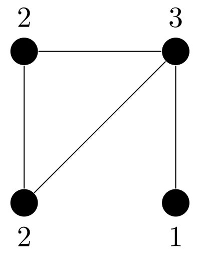
(a) $\sum_{v \in V} \deg(v) = 2$, $|E| = 1$



(b) $\sum_{v \in V} \deg(v) = 6$, $|E| = 3$



(c) $\sum_{v \in V} \deg(v) = 4$, $|E| = 2$. Recall that a self loop contributes 2 towards the degree of a vertex, which is why the degree of the vertex on the left is 3.



(d) $\sum_{v \in V} \deg(v) = 8$, $|E| = 4$

Figure 12.5: Finding the relationship between the sum of the degrees of vertices in a graph and the number of edges in that graph. In the pictures above, the vertex labels indicate the vertices' degrees.

12.3 Trees

Before we describe trees, we revisit the notion of a path.

We begin by describing *simple paths.* A simple path is just like any other path, except every edge appears on it at most once, and the only vertex that can be visited twice is the start vertex, but only if the second visit is at the very end of the path. If the start vertex is the same as the end vertex of a non-empty path, the path is called a *simple cycle.*

Example 12.8: Consider the graph in Figure 12.6. We show some simple paths as well as some paths that are not simple. ⊠

In a simple graph, there is at most one edge between any one pair of vertices. Thus, we can list the vertices on the path instead of listing the edges. If the path goes from vertex u to vertex v in one step, there is a unique edge this path can follow to achieve that. Hence, we can recover the sequence of edges from the sequence of vertices visited on the path.

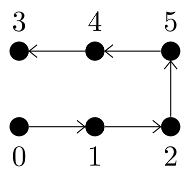
For example, we could describe the path in Figure 12.6a using the sequence $0, 1, 2, 5, 4, 3$ instead of the sequence $\{0, 1\}, \{1, 2\}, \{2, 5\}, \{5, 4\}, \{4, 3\}$.

And now we are ready to define what a tree is.

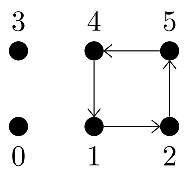
Definition 12.12. *A* tree *is a connected graph without simple cycles.*

We show three graphs in Figure 12.7. Only the first graph is a tree. The remaining two are not. One is not connected and one is not acyclic.

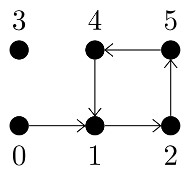
Now the graph in Figure 12.7a doesn't exactly look like a tree. So let's redraw it to make it look more like a tree. Pick an arbitrary node (in this case we pick the only one of degree 4),



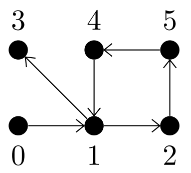
(a) A simple path from 0 to 3.



(b) A simple cycle that starts and ends at 1.

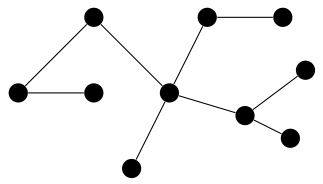


(c) A path from 0 to 1 that is not simple. The last vertex is a previously-visited vertex, but not the start vertex.

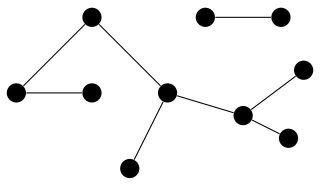


(d) A path from 0 to 3 that is not simple. The vertex 1 appears twice. The edge taken after the first visit to 1 is $(1, 2)$.

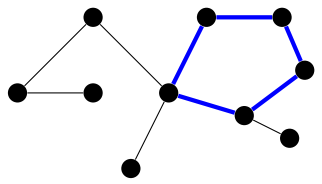
Figure 12.6: Examples illustrating the definition of a simple path. The arrows give the direction each edge is traversed in.



(a) A tree



(b) Not a tree because the graph has two connected components.



(c) Not a tree because there is a simple cycle in this graph. The edges on that cycles are highlighted using thick blue lines.

Figure 12.7: Examples illustrating the definition of a tree.

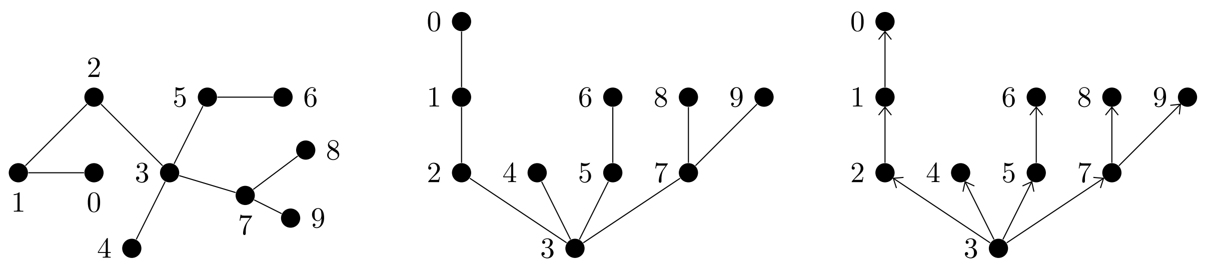
and consider it the *root* of the tree. Draw its neighbors above it, then draw the neighbors of the neighbors (except for the root which has already been drawn) one more level above, and keep going until the entire graph is drawn. We show this process in Figure 12.8, with the graph taking a tree form in Figure 12.8b.

Every vertex of degree 1 in a tree is called a *leaf*. For example, the leaves in Figure 12.8b are vertices 0, 4, 6, 8, and 9.

We declared the vertex 3 the root of the tree in Figure 12.8b, but that is only a product of how we drew the tree. To give the root some significance, we make all the edges directed, so that they all point away from the root. The resulting tree, called a *rooted tree*, is shown in Figure 12.8c. Now the concept of a *root* makes sense. It is the only vertex with in-degree zero. All the other vertices have in-degree 1.

12.3.1 Properties of Tress

Given their special structure, we can prove more facts about trees. In particular, we describe some relationships between the vertex and edge sets of trees.



(a) A tree (b) Now it looks more like a tree. (c) In a rooted tree, all edges point away from the root.

Figure 12.8: Pick node 3 as the root. Then draw all its neighbors in the next level. One level above, draw the neighbors of the root's neighbors. Keep going in this fashion until the entire picture is drawn.

Proposition 12.13. *Let $T = (V, E)$ be a tree. Then the following are true.*

(i) If $|V| \geq 2$, T has at least two leaves.

(ii) There is exactly one simple path between any two vertices.

(iii) If V is nonempty, $|V| = |E| + 1$.

We make some remarks about individual parts of Proposition 12.13, and then prove the individual parts.

If there is just one vertex in the graph, we consider the graph a tree if there are no edges, and we call the only vertex a leaf in that case. If there are at least two vertices, the characterization of the number of leaves from part (i) applies.

Proof of part (i) of Proposition 12.13. Consider a simple path P of maximum length in the tree T. Say P starts at u, ends in v, and the vertices on the path are $u = v_0, v_1, \ldots, v_{r-1}, v_r = v$. We claim that u and v have degree 1, which means they are leaves.

Suppose u does not have degree 1. Then there is another edge besides $\{u, u_1\}$ that's incident on u, say $\{w, u\}$. This edge does not appear on the path P because P is simple. There are two cases to consider, and each of them leads to a contradiction.

Case 1: w does not appear on the path P. Then the path $w, u, v_1, \ldots, v_r, v$ is a simple path longer than P, which is a contradiction because P is the longest simple path in T.

Case 2: w appears on the path P, say $w = v_i$. Then the path $w = v_i, v_{i+1}, \ldots, v_r, v_i$ is a simple cycle in T. But since T doesn't contain any simple cycles, this is a contradiction. □

Part (ii) of Proposition 12.13 actually gives us an equivalent definition of a tree. That is, a connected graph with exactly one simple path between any two vertices is a tree. We leave the proof of this fact as an exercise to the reader.

Because T is connected, there is a path between any two vertices in T. We first show that this implies the existence of a simple path between any two vertices in T. For example, consider the path in Figure 12.6d. We could turn it into a simple path by omitting the cycle $1, 2, 5, 4, 1$, and just going to 3 the moment we reach 1 for the first time. In fact, this strategy works in general. If a cycle is part of a path from u to v, the part of the path without the cycle is a shorter path from u to v. After removing some number of cycles in this fashion, we end with a simple path.

Lemma 12.14. *Let $G = (V, E)$ be a graph. If there is a path from vertex u to vertex v, there is also a simple path from vertex u to vertex v.*

Proof. We give a proof by induction on the path length.

In the base case, consider a path of length zero. This has only one vertex on it, namely the starting vertex. Thus, no vertices repeat, and the path is simple.

Now consider a path of length $n+1$ from u to v in some graph. Suppose that for every $x, y \in V$ if there is a path from x to y of length n or less, then there is also a simple path from x to y.

Let $e_1, e_2, \ldots, e_{n+1}$ be the edges on the path from u to v. If this path is simple, there is nothing to prove. Otherwise there is some vertex w that appears twice so that it is the endpoint of e_i and e_j with $i < j$. Then the edges $e_{i+1}, e_{i+1}, \ldots, e_j$ form a cycle that starts and ends at w. Moreover, the path $e_1, \ldots, e_i$ is a path from u to w and $e_{j+1}, \ldots, e_{n+1}$ is a path from w to v. Then the sequence of edges $e_1, \ldots, e_i, e_{j+1}, \ldots, e_{n+1}$ is a path from u to v. The path has length $i+(n+1)-(j+1)+1 = i+n-j+1$. Now $j \geq i+1$, so the length is at most $i+n-(i+1)+1 = n$. The induction hypothesis then implies there is a simple path from u to v. □

Proof of part (ii) of Proposition 12.13. Consider any two vertices u and v in T. There is a path from u to v in T because T is connected. Then there is a simple path from u to v by Lemma 12.14.

What remains to show is that there are not multiple simple paths from u to v. For the purpose of contradiction, assume there are different simple paths, P and Q, from u to v. The path P goes through vertices $u = v_0, v_1, \ldots, v_{r-1}, v_r = v$. The path Q starts at v, and at some point has to deviate from P. Say that at v_i for some $i < r$, Q goes to $w_1 \neq v_{i+1}$ next. At some point Q, has to rejoin P because both P and Q end at v. Say that Q continues from v_i by visiting vertices $w_1, w_2, \ldots, w_s$ that don't appear on P, and rejoins P after w_s by going to v_j with $0 \leq j \leq r$ and $i \neq j$ (the inequality holds because otherwise Q would not be a simple path). We show that the vertex v_j is involved in a simple cycle, which is a contradiction. There are two cases to consider.

Case 1: $j < i$. Consider the path from v_j to v_i using Q and then the path from v_i to v_j using P. Observe that w_k does not appear on the path P for any $k \in \{1, \ldots, s\}$ by assumption. Thus, the vertices $v_j, v_{j+1}, \ldots, v_i, w_1, \ldots, w_s$ are all distinct, and are, therefore, part of the simple cycle $v_j, v_{j+1}, \ldots, v_i, w_1, \ldots, w_s, v_j$.

Case 2: $j > i$. Consider the path from v_i to v_j using P and then the path from v_j to v_i obtained by following Q backwards from v_j (which we can do because edges don't have direction). Observe that w_k does not appear on the path P for any $k \in \{1, \ldots, s\}$ by assumption. Thus, the vertices $v_j, v_{j+1}, \ldots, v_i, w_1, \ldots, w_s$ are all disjoint vertices, and are part of the simple cycle $v_i, v_{i+1}, \ldots, v_j, w_s, \ldots, w_1, v_i$. □

Let us stress again that there are two common strategies for proving a fact about graphs by induction. Either induct on the number of vertices or the number of edges. We choose the former for the next proof.

Proof of (iii) of Proposition 12.13. We give a proof by induction on the number of vertices.

For the base case, consider a tree with one vertex. This tree has no edges because the only possible edge would be a self-loop, and this edge would form a simple cycle. Thus, in the base case, we have $|V| = 1$ and $|E| = 0$, so $|V| = |E| + 1$ holds.

Now consider a tree with vertex set V, and assume that every tree with $|V| - 1$ vertices has $|V| - 2$ edges. Note that $|V| \geq 2$ in this case, so T has a vertex of degree 1 by part (i) of Proposition 12.13. Call this vertex v.

Consider the subgraph $T' = (V', E')$ of T obtained from T by removing v and its incident edge. Suppose that u is the vertex connected to v by this edge. The situation is shown in Figure 12.9.

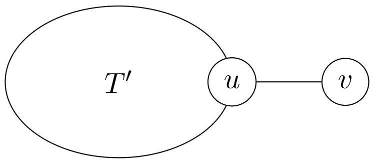
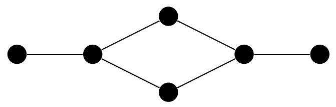


Figure 12.9: The tree T in the proof of part (iii) of Proposition 12.13.

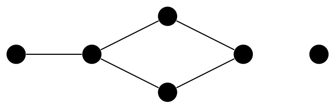
We claim that T' is a tree. First, T' doesn't contain any simple cycles because it's a subgraph of T, which does not have any simple cycles. Second, consider any two vertices $x, y \in V'$. Since $x, y \in V$, there is a simple path from x to y in T by part (ii) of Proposition 12.13. We claim that this path consists only of vertices in V'. If not, it would have to contain v. But the only way to get to v is from u, and the only way to continue the path from v is to go back to u. But this would traverse the edge $\{u, v\}$ twice, which would contradict the fact that the path is simple.

We have shown that T' is connected and doesn't have any simple cycles, so T' is a tree. Moreover, it consists of $|V'| = |V| - 1$ vertices, so $|V'| = |E'| + 1$ by the induction hypothesis. Since $|V'| = |V| - 1$ and $|E'| = |E| - 1$, we also get that $|V| = |V'| + 1 = |E|' + 1 + 1 = |E| + 1$, which completes the proof. □

We mentioned that part (ii) of Proposition 12.13 gave an equivalent definition of a tree. You may ask whether any graph satisfying part (i) or part (iii) is a tree. The answer is no. For example, the graph in Figure 12.10a has two vertices of degree 1 but also contains a simple cycle, and the graph in Figure 12.10b satisfies the conclusion of (iii), but is not connected. In fact, any graph $G = (V, E)$ with $|V| = |E| + 1$ that is not a tree is disconnected and contains a simple cycle.



(a) Not every graph with two vertices of degree 1 is a tree.



(b) Not every graph with $|V| = |E|+1$ is a tree. This graph has the right number of edges, but contains a simple cycle and is disconnected.

Figure 12.10: Parts (i) and (iii) of Proposition 12.13 are not equivalent definitions of a tree.

12.3.2 Spanning Trees

Suppose Madison gets hit by a giant blizzard. The road cleaning crew needs to decide which roads to plow first so that Madisonians can get from every intersection to any other intersection in town. (We assume that they can walk through the snow to the nearest intersection and can reach their actual destination by walking through the snow from the intersection nearest to it.)

Model the streets of Madison as a graph G. Intersections are vertices, and two vertices u and v are connected by an edge if there is a direct road segment between the intersections that correspond to u and v (that is, there are no other intersections on the way between those two intersections).

Now consider any connected subgraph C of G consisting of all vertices of G. If the road cleaning crew plows all the road segments corresponding to edges in C, they will have accomplished their goal. Now consider the situation when C is a tree. This tree consists of all vertices of G. Such a tree is called a *spanning tree* of G. The road cleaning crew can just find a spanning tree of G and plow the road segments corresponding to the edges in that spanning tree.

So, why should the road cleaning crew be able to find a spanning tree in the first place? The key observation is that the graph G is connected. If it were not connected, some Madisonians would be cut off from some parts of Madison, which would be a rather sorry state of affairs. Let's start with a formal definition of a spanning tree, and then prove a theorem that tells us the road cleaning crew can find a spanning tree.

Definition 12.15. *Let $G = (V, E)$ be a graph, and let $T = (V, E')$ be a connected subgraph of G. If T is a tree, we call it a* spanning tree *of G.*

Theorem 12.16. *Every connected graph G has a spanning tree.*

Proof. Start with the graph $T = G$. Now keep removing edges from T until it becomes a tree.

If T is a tree, it is a spanning tree of G. Otherwise T has a simple cycle in it. Pick any edge on that cycle, and remove it from T to obtain a graph T'. We claim that T' is connected. This is because if there is a path between vertices x, y in T such that the path passes through the edge that is removed, then we can obtain a path between vertices x, y in T' by traversing the rest of the cycle instead of the removed edge. Also, T' still contains all vertices of G (No vertices were deleted to obtain T'). Now replace T with T', and look for another edge to remove from T.

The process of removing edges stops when T has $|V| - 1$ edges by part (ii) of Proposition 12.13. At that point, T is a spanning tree because it consists of all vertices of G, it is connected and it contains no simple cycles. □

If a graph G is not connected, it doesn't have a spanning tree because there is a pair of vertices in G without a path between them. No subgraph of G can contain a path between those two vertices either, so G does not have a spanning tree.

The best we can hope for if G is not connected is to find a spanning tree for each connected component of G. This is possible by the theorem we just proved. The collection of those spanning trees is called a *spanning forest* of G.

12.4 Applications of Graphs

We have already seen some applications of graphs. For example, we used topological sorting to come up with a valid procedure for getting dressed in the morning, and we used spanning trees to help the Madison street cleaning crew rescue the city after a major blizzard. Let's now discuss some computer science oriented applications of graphs.

12.4.1 Graph Coloring

Every program written in a high-level language such as C needs to be compiled down to something a computer can run on its CPU. There may be many variables in a program, but the CPU can only store a constant number of them in its registers. One problem the compiler faces is to allocate a register on the CPU for each variable. It can reuse registers if the lifetimes (or scopes if you want) of two variables do not overlap. That is, if the lifetimes of two variables overlap, those two variables need to be stored in different registers. But if the lifetimes don't overlap, the compiler can use the same register for the variable that just started existing and the variable that already ceased to exist. If the compiler fails to allocate registers in the fashion we just described, it will be necessary to copy some data back and forth between the main memory (RAM) and the CPU's registers, which is going to slow down the execution of the program.

Let's model the situation the compiler is facing as a graph. There is a vertex corresponding to each variable, and two vertices are connected with an edge if their corresponding variables' lifetimes overlap. Our goal is to label each vertex with a name of a register so that no pair of vertices connected by an edge has the same label.

This is exactly the problem we face in graph coloring. There, we are given a graph, and our goal is to color its vertices so that if two vertices are connected by an edge, they don't have the same color. Of course, the goal is to use as few colors as possible. So just think of register names as colors, and register allocation becomes graph coloring.

Definition 12.17. *Let $G = (V, E)$ be a graph, and C a set of "colors" with $|C| = k$. A k-coloring of G is an assignment of "colors" in C to the vertices of G such that for every edge $\{u, v\}$ in G, the vertices u and v are assigned different colors.*

Definition 12.18. *The* chromatic number *of a graph G, denoted $\chi(G)$, is the smallest number k for which G has a k-coloring.*

We characterize graphs with low chromatic numbers (up to 2), and discuss some other results about the chromatic number of various graphs.

First, if $\chi(G) = 0$, we are not allowed to use a single color to color vertices of G. But every vertex must have some color, so the only graph that has chromatic number zero is the empty graph (a graph with zero vertices and no edges).

Now suppose $\chi(G) \leq 1$. Since we are allowed to use only one color, there cannot be any edges in the graph, for if there were an edge, both of its endpoints would be colored the same.

The situation becomes more interesting when $\chi(G) \leq 2$. First let's look at some incorrect characterizations of graphs whose chromatic number is at most 2. These incorrect characterizations come from various attempts to extend the characterization of graphs with chromatic number 1. A graph with chromatic number 1 has no edges, so one would be tempted to say that a graph with chromatic number 2 has at most 1 edge. Also, every vertex in a graph with chromatic number 1 has degree 0, so we may think that all vertices in a graph with chromatic number 2 must have degree at most 1. Neither of those two claims gives a full characterization of graphs with chromatic number 2. For example, the graph in Figure 12.11 has more than one edge, and also contains a vertex of degree more than 1.

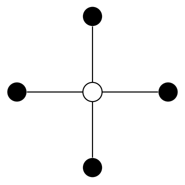


Figure 12.11: A 2-colorable graph. The vertex in the middle is colored white and all the other vertices are colored black. No edge has the same color on both ends, so the coloring shown here is a valid 2-coloring.

Suppose we have two colors, red and green, at our disposal. Each vertex gets one of those colors. Regardless of whether the color assignment is a coloring, it induces a partition of the vertex set. One partition class is the set of vertices colored red, and the other partition class is the set of vertices colored green. If the color assignment is actually a 2-coloring, there are no edges between a pair of vertices colored by the same color. But then the graph is bipartite. Let V_{left} be the set of red vertices, and V_{right} be the set of green vertices. Since the coloring is valid, there are no edges that are subsets of V_{left} or V_{right}.

Conversely, if a graph is bipartite, it is 2-colorable. Just color all vertices in V_{left} red and all vertices in V_{right} green. Since all edges involve one vertex of V_{left} and one vertex of V_{right}, their endpoints have different colors. Thus, we have proved the following theorem.

Theorem 12.19. *A graph is 2-colorable if and only if it is bipartite.*

There is a more general correspondence between a k-coloring of a graph G and a partition of its vertex set, V, into k subsets $V_1, \ldots, V_k$ such that no edge is a subset of V_i for any $i \in \{1, \ldots, k\}$ (we call such a graph *k-partite*).

If the graph has a k-coloring, we can define partition classes by grouping together all vertices with the same color, i.e., $V_c = \{v \in V \mid v \text{ is colored with color } c \text{ in the coloring}\}$. Note that no edge in G can be a subset of V_c because that would make both of its endpoints have the same color.

Conversely, if we have a partition of V into sets V_1 through V_k, we can color all vertices in V_1 with one color, all vertices in V_2 with another color, and so on.

The chromatic number 2 case is interesting because we can easily tell whether a graph is bipartite, i.e. we can easily tell if a graph is 2 -colorable.

There is no good characterization of graphs with chromatic number at most 3. In fact, a good characterization that is computable by an efficient algorithm would be a major breakthrough.

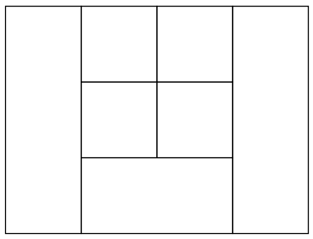
There is also no good characterization of graphs with chromatic number at most 4, but an important class of graphs has chromatic number at most 4.

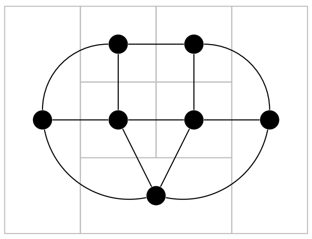
Definition 12.20. *A planar graphs is a graphs that can be drawn on a plane in such a way that no two edges intersect each other.*

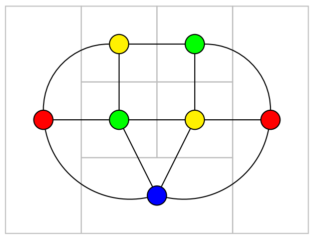
Theorem 12.21 (Four color theorem)**.** *If G is a planar graph, $\chi(G) \leq 4$.*

The original proof of the four color theorem has an interesting place in history because it was done by a computer. The authors of the proof, Appel and Haken, designed a program that generated a list of cases such that every planar graph falls in at least one of them. The program then verified for each case that it was four-colorable. Of course, since the proof was done by a computer, it is necessary to prove correctness of the program. Since then, the proof has been simplified. In particular, the number of cases was reduced and it is now possible, but still tedious, to verify the entire proof by hand.

The four color theorem has an interesting consequence. Consider a map in which every country is colored so that two countries that share a border have different colors. Note that we can draw the map in a plane. So place a vertex in each country, and connect vertices corresponding to two countries that share a border with edges (here it is important that they share a border and not just one point in the corner). This gives us a planar graph. The four color theorem then tells us we can color the resulting graph with four or fewer colors. Now just color a country on the map with the color of its corresponding vertex in the graph to get the desired coloring of the map. We illustrate this process in Figure 12.12.


(a) Start with a map.


(b) Make a planar graph.


(c) Find a 4-coloring.

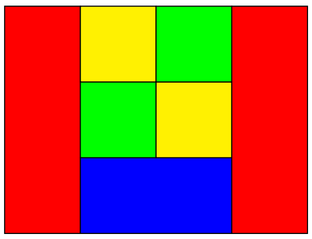

(d) Use the 4-coloring to color the map's regions.

Figure 12.12: Four-coloring a map