# Chapter 11

Additional Graph Theory

A **digraph** (or **directed graph**) D consists of a set V of vertices and a set E of **directed edges** (or **arrows**), each of which is represented as an ordered pair (u, v), where $u, v \in V$. We say that u is the **initial vertex** and v is the **terminal vertex** of the directed edge (u, v). We write $D = (V, E)$ as we did with undirected graphs. The **indegree** of a vertex v in a digraph, denoted $\boxed{\deg^-(v)}$, is the number of directed edges that have v as a terminal vertex while the **outdegree** of v, denoted $\boxed{\deg^+(v)}$, is the number of edges having v as an initial vertex.

Problem 11.1. Find the indegree and outdegree of each vertex in the following graph.
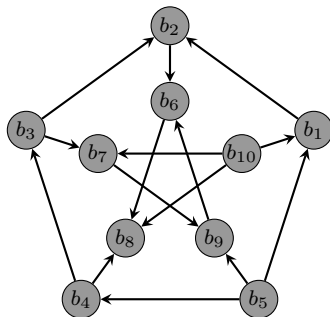
As expected, we have the following result that is analogous to the Handshake Lemma (Theorem 10.14).

Theorem 11.2. If $D = (V, E)$ is a digraph, then

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

Each graph/digraph is determined by its vertices and the manner in which they are connected by edges, not the way a graph/digraph might be sketched. We can represent a graph in a couple of ways.

The **adjacency list** of a simple graph lists all vertices in one column and all adjacent vertices in second column. For a digraph, the columns contain the initial vertices and the associated terminal vertices.

Problem 11.3. Make up a couple examples to explore adjacency lists for simple graphs and digraphs.

An $m \times n$ **matrix** A is a rectangular array of numbers with m rows and n columns. The entry in the ith row and jth column is indicated by $A_{i,j}$.

Example 11.4. The example below is a 2×3 matrix:

$$A = \begin{bmatrix} 2 & 3 & 5 \\ 10 & 6 & 7 \end{bmatrix}$$

In this example, $A_{1,2} = 3$.

The **adjacency matrix** A of a graph (respectively, digraph) G with vertices listed as $v_1, v_2, \ldots, v_n$ is the $n \times n$ matrix A whose entry $A_{i,j}$ in row i and column j is the number of edges connecting v_i and v_j (respectively, the number of edges from v_i to v_j).

Problem 11.5. Find the adjacency matrix for the following graph.
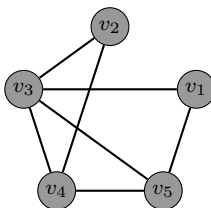
Problem 11.6. What properties will the adjacency matrix for a simple graph have?

Problem 11.7. Sketch a graph that has the following adjacency matrix.

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 1 & 2 & 0 \end{bmatrix}$$

Problem 11.8. Sketch a digraph that has the following adjacency matrix.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 0 \end{bmatrix}$$

Problem 11.9. What will the adjacency matrix for P_n look like, assuming the vertices are taken in the natural order (start at one end of the path and end at the other)? What about C_n? K_n?

Recall that a graph is not determined by a sketch since many sketches give the same graph. It may be hard to recognize from sketches whether two graphs are "essentially" the same even though the vertices may be different points. The notion of isomorphism (same form) gives us a way to deal with this. Two *simple* graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are the **isomorphic**, written $G_1 \cong G_2$, if there is a a bijection $f : V_1 \to V_2$ such that $\{u, v\}$ is an edge in G_1 if and only if $\{f(u), f(v)\}$ is an edge in G_2. The function f is called an **isomorphism**. For digraphs, we require that (u, v) is a directed edge in G_1 if and only if $(f(u), f(v))$ is a directed edge in G_2.

For $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, to show that $G_1 \cong G_2$:

1. State a vertex matching explicitly, and

2. Either

   (a) Check adjacency for each pair of vertices in G_1 and the corresponding pair in G_2 (a total of $\binom{|V_1|}{2}$ checks). This could also be as simple as providing sketches for each graph that clearly exhibit the correspondence of vertices and edges.

   (b) Demonstrate that the adjacency matrices of G_1 and G_2 are the same using an ordering that is compatible with the vertex matching.

Warning! The second method above usually involves much less writing, but be aware that the adjacency matrices may differ in one ordering but agree with a different ordering.

The simplest way to show that $G_1 \ncong G_2$ is to show that a feature preserved under isomorphism (called an **invariant**) holds for one graph but not the other. Here are a few isomorphic invariants:

(a) Order of the graph

(b) Number of edges in the graph

(c) Number of vertices of a given degree

(d) Degree sequence

(e) Vertices of degree k and ℓ are adjacent

(f) Subgraph that is isomorphic to C_n or P_n.

Problem 11.10. Determine whether the following graphs are isomorphic.
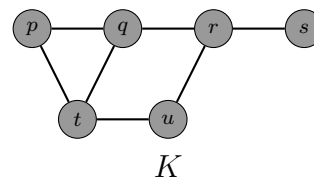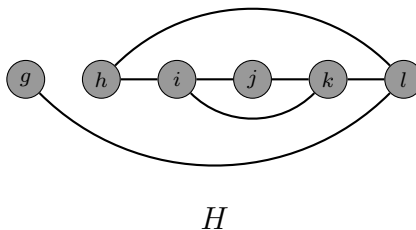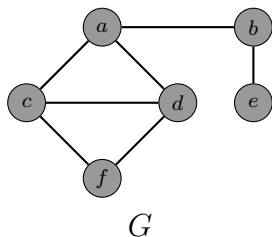

$$G \qquad\qquad H$$
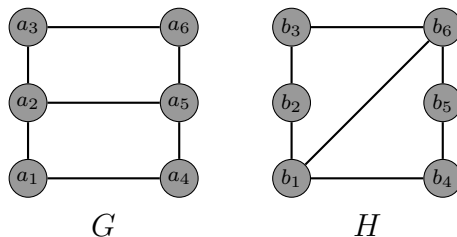
Problem 11.11. Let G be the graph with vertex set $V = \{a, b, c, d, e\}$ and edge set $E = \{\{a, b\}, \{a, c\}, \{a, e\}, \{b, d\}, \{b, e\}, \{c, d\}\}$ and let H be the following graph.
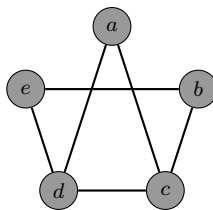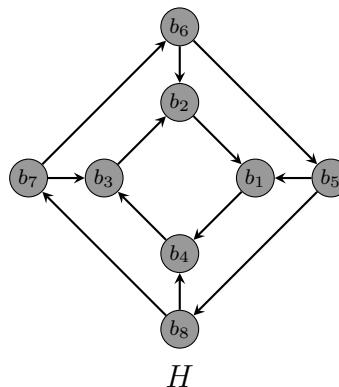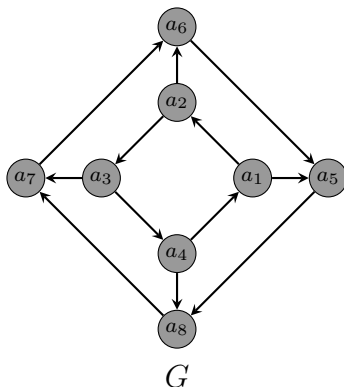
Determine whether G and H are isomorphic.

Problem 11.12. Determine which pairs of the following graphs are isomorphic.

$$G \qquad H \qquad K$$

Problem 11.13. Determine whether the following digraphs are isomorphic.

$$G \qquad H$$

We now introduce several new terms.

- A **walk** in a graph is an alternating sequence of vertices and edges that starts with a vertex and ends with a vertex such that consecutive vertices in the walk are the endpoints of the edge that separates them. In a simple graph, a walk can be specified by a sequence of vertices.

- The **length** of a walk is the number of edges in the walk.

- If the initial and terminal vertices of a walk are the same, then the walk is a **closed walk**.

- A **trail** is a walk with distinct edges (no repeated edges).

- A **circuit** is a closed trail, that is, a closed walk with no repeated edges.

- A **path** is a walk with distinct vertices. This is a subgraph isomorphic to P_n for some n.

- A **cycle** is a closed walk with distinct vertices except the initial and terminal vertices. This corresponds to a subgraph isomorphic to C_n for some n.

- A graph G is **connected** if for each pair of distinct vertices u and v, there is a walk from u to v. A **component** of a graph is a connected subgraph that is not contained in a larger connected subgraph.

- A **cut vertex** of a connected graph G is a vertex which when removed along with all incident edges results in a disconnected graph.

- A **bridge** (or **cut edge**) is an edge of a connected graph which when removed results in a disconnected graph.

The following theorem likely does not come as a surprise.

Theorem 11.14. A graph G is connected if and only if for each pair of distinct vertices u and v, there is a path from u to v.

A digraph is **strongly connected** if for each pair of distinct vertices u and v there is a (directed) walk from u to v. A digraph is **weakly connected** if the underlying undirected graph in which the direction of edges is removed is connected. Note that a strongly connected digraph will always be weakly connected. A **strongly connected component** of a digraph is a maximal strongly connected subgraph.

We now introduce a couple of important circuits that a graph may or may not possess. An **Euler circuit** in a graph G is a circuit that contains every edge of the graph. An **Euler trail** in a graph is a trail that contains every edge of the graph. Note that an Euler circuit is also an Euler trail as well.

Problem 11.15. Determine whether each of the following graphs has an Euler trail. How about an Euler circuit?


$$G \qquad\qquad H \qquad\qquad K$$
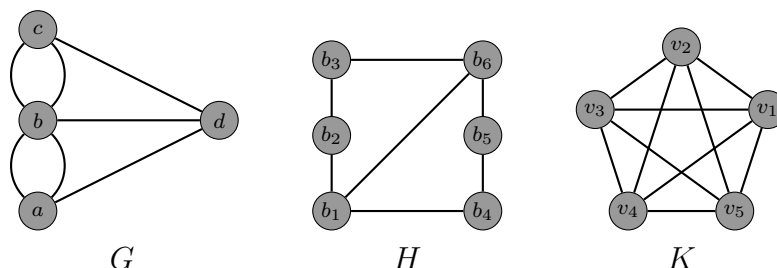
Theorem 11.16. If G is a connected graph of order $n \geq 2$, then G has an Euler circuit if and only if every vertex is even.

Corollary 11.17. If G is a connected graph of order $n \geq 2$, then G has an Euler trail that is not a circuit if and only if G has exactly two odd vertices.

Edges can be added to a connected graph in order to cause it to have an Euler circuit (or trail). In particular, an **Eulerization** of a connected graph is the addition of suitable multiple edges (i.e., duplicate existing edges) to permit an Euler circuit, mimicking what must be done to complete a circuit such as a postal route or other delivery/pick-up route.

Problem 11.18. Eulerize the following graph.
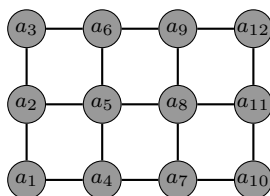
A cycle in a graph that passes through every vertex is a **Hamilton cycle**. This is often called a **Hamilton circuit**. A **Hamilton path** is a path in a graph that includes every vertex.

Problem 11.19. Determine whether each of the following graphs has a Hamilton circuit or a Hamilton path that is not a circuit.
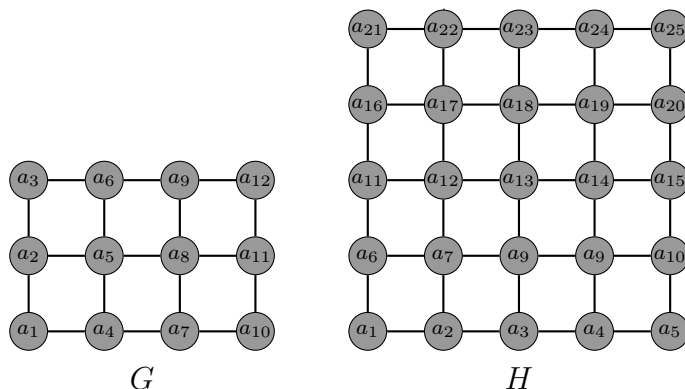
Unfortunately, unlike the situation for Euler circuits, there is no known simple necessary and sufficient condition for a Hamilton cycle to exist in a graph. We can state some simple cases when one cannot exist, and there are some theorems for the existence of a Hamilton cycle, but these do not cover all possibilities.

Theorem 11.20 (Dirac's Theorem). If G is a simple graph of order $n \geq 3$ in which $\deg(v) \geq n/2$ for each vertex, then G has a Hamilton cycle.

Theorem 11.21 (Ore's Theorem). If G is a simple graph of order $n \geq 3$ in which $\deg(u) + \deg(v) \geq n$ for each pair of vertices u and v, then G has a Hamilton cycle.

Problem 11.22. How many distinct Hamilton cycles does K_n have that start/end at a fixed vertex?

We now turn our attention to trees. A **tree** is a connected graph that has no cycles. A **forest** is a graph in which every connected component is a tree. Trees provide a useful structure for organizing data, for displaying organization, and for decision processes.

**Problem 11.23.** Is every tree necessarily a simple graph?

Theorem 11.24. A graph G is a tree if and only if for each pair of distinct vertices u and v, there is a unique path from u to v.

Theorem 11.25. Some properties of trees.

(a) A tree of order 2 or more has at least two vertices of degree 1.

(b) Every edge of a tree is a bridge.

(c) A connected graph in which every edge is a bridge is a tree.

(d) A tree of order n has $n - 1$ edges.

(e) A connected graph of order n with $n - 1$ edges is a tree.

A **spanning tree** of a simple graph G is a subgraph T of G such that T is a tree contains every vertex of G.

Problem 11.26. Find a spanning tree for each of the following.

G H
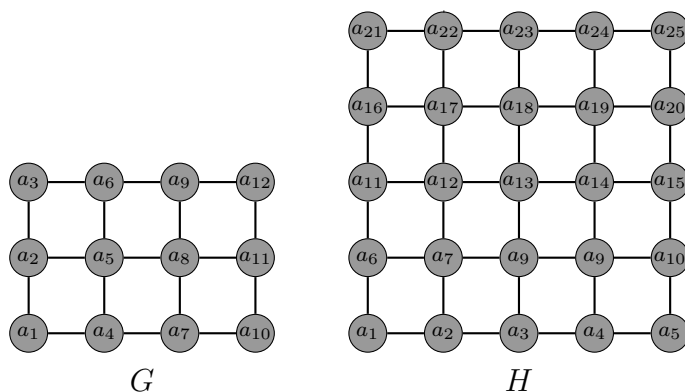
Problem 11.27. Do you think every connected simple graph contains a spanning tree?

In fact, we have the following theorem.

Theorem 11.28. A simple graph G is connected if and only if it contains a spanning tree.

Below, we provide informal descriptions of the depth-first and breadth-first algorithms for identifying a spanning tree in a connected simple graph G with vertices ordered as $v_1, v_2, \ldots, v_n$.

Depth-First Search. Initialize: $T = \{v_1\}$, $v = v_1$. As long as T does not contain all vertices, do the following:

- Choose the first vertex w in the ordered list that is adjacent to v and is not yet in T. If there are no vertices adjacent to v that are not yet in T, return to the most recently added previous vertex u and let $v = u$ and repeat this step with the revised v.

- Add w and edge $\{v, w\}$ to T.

- Set $v = w$.

Repeat the steps above (with revised v's) as often as needed.

Breadth-First Search. Initialize: $T = \{v_1\}$, $L = v_1$. As long as T does not contain all vertices, do the following.

- Put $v =$ the first vertex in L.

- Remove that vertex from L. (On paper, just mark out.)

- For all vertices adjacent to v not yet in T,

  - If w is the first vertex in order that is adjacent to v and not yet in T, add w and edge $\{v, w\}$ to T and put w at the end of list L.
  - Repeat until all vertices adjacent to v not yet in T have been examined.

Repeat the above steps as needed.

A **weighted graph** is a graph with a positive numerical values assigned to each edge of the graph. A **minimal spanning tree** of a connected weighted graph is a spanning tree that has the smallest possible sum of weights for its edges.

Minimal spanning trees arose in the practical matter of designing an efficient electrical network, and the concept applies to similar network notions in many areas such as transportation, utilities, and others. We will discuss one early, elementary, and effective algorithms for finding a minimal spanning tree in a connected undirected weighted graph.

Prim's Algorithm (Jarnik 1930, Prim 1957).   Assume G is a connected, undirected, weighted loopless graph of order n.

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.

2. Grow the tree by one edge: Of the edges that connect the tree to vertices not yet in the tree, find a minimum-weight edge, and transfer it to the tree.

3. Repeat step 2 (until all vertices are in the tree).

Prim's Algorithm is an example of a greedy algorithm.

Problem 11.29. Make up an example of a connected, undirected, weighted loopless graph and find a minimal spanning tree.