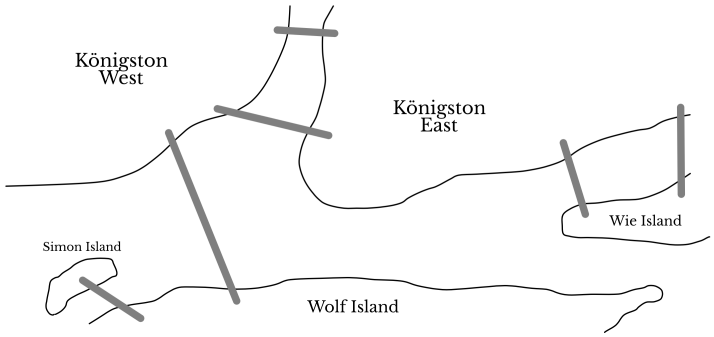
Queen's University
School of Computing

CISC 203: Discrete Mathematics for Computing II
Lecture 5: Graph Theory
Winter 2019

1 Traveling Around Königston

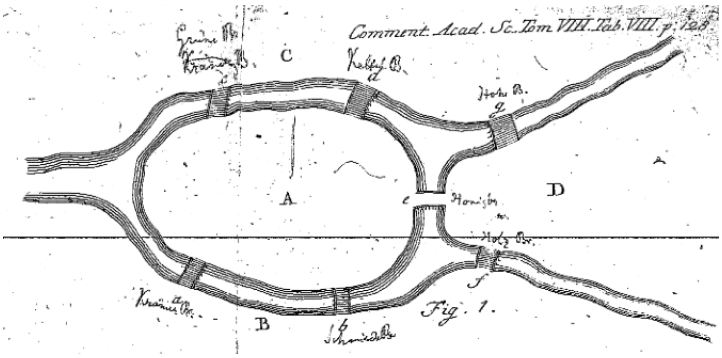
Imagine that, one day, you take a trip to the City of Königston. (This city looks a lot like Kingston, but it has a large German population, hence the name.) Königston consists of five regions: Königston West, Königston East, Wolf Island, Wie Island, and Simon Island. They are arranged as follows:



To move between these five regions, you can take either a bridge or a ferry. Since you want to get the full Königston experience, but you don't want to stay out too late, you decide to cross each bridge and ride each ferry exactly once.

During your trip, the local government announces a new ferry service between Wolf Island and Wie Island. Naturally, you feel the need to redo your tour. However, with this new ferry, it is no longer possible for you to cross each bridge and ride each ferry exactly once! How could such a small change have such a big effect?

In 1736, the Swiss mathematician Leonhard Euler thought about a remarkably similar problem called the "Bridges of Königsberg". The downtown area of the City of Königsberg (now known as Kaliningrad, Russia) is divided into four parts by a river, and each of these parts are connected to one another by a total of seven bridges. In his paper, Euler drew the following map to illustrate the layout of the city:



Interestingly, Euler found that, no matter what path you take across the bridges, you cannot cross each of the bridges exactly once. With this simple puzzle, Euler inadvertently kicked off the entire study of **graph theory**—something that affects both mathematicians and tourists to this day.

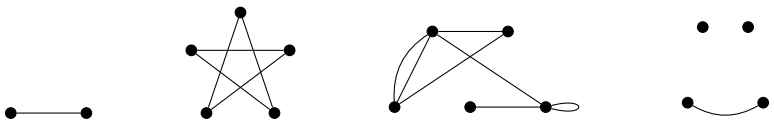
2 Graphs

At its core, a **graph** is not a complicated concept. In fact, graphs are only made up of two sets.

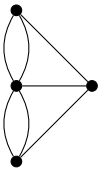
Definition 1 (Graph)**.** A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, where each element of E is a pair $\{u, v\}$ of vertices $u, v \in V$.

In other words, a graph is just a set of elements (which we call **vertices**) and a set of relations that join those elements (which we call **edges**).

Example 2. Each of the following are graphs:



Example 3. We can model the Bridges of Königsberg problem with the following graph:



2.1 Properties of Graphs

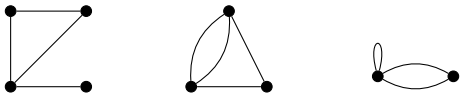
You may have noticed that Definition 1 is worded in a very general way, and it allows for strange things that might make our graphs complicated, like multiple edges between the same vertices or edges joining a vertex to itself (called **loops**). If we like, we can refine our definition to consider restricted graphs that don't allow for such strange things. We call such graphs **simple**; these are the kinds of graphs we will be dealing with most of the time.

Definition 4 (Simple graph)**.** A graph $G = (V, E)$ is simple if,

1. given two elements $\{u_1, v_1\}, \{u_2, v_2\} \in E$, both $u_1 \neq u_2$ and $v_1 \neq v_2$ (no multiple edges); and
2. if $\{u, v\} \in E$, then $u \neq v$ (no loops).

In the literature, graphs with multiple edges are called **multigraphs** and graphs with both multiple edges and loops are called **pseudographs**, but we will not use such terminology here.

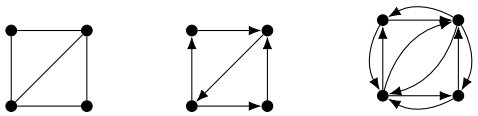
Example 5. The first graph is a simple graph. Neither the second nor third graphs are simple graphs. (The second graph is a multigraph and the third graph is a pseudograph.)



Both Definitions 1 and 4 assume that the set of edges E consists of unordered pairs, so if an edge exists between vertices u and v, then another edge implicitly exists between v and u. If we instead require that E consists of ordered pairs, then the existence of an edge between vertices u and v does not necessarily imply the existence of an edge between v and u. We say that such graphs are **directed**, because the direction of each edge in the set E matters when we move between vertices. Directed graphs are occasionally referred to by the shorthand name of "digraphs".

As you would expect, we say that graphs where the direction of edges does not matter are **undirected**.

Example 6. Out of the following graphs, the first graph is undirected and both the second and third graphs are directed. The third graph is the directed equivalent of the first graph.

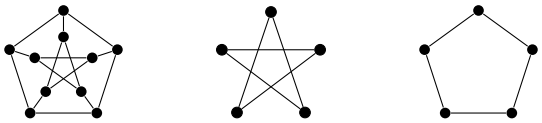


Finally, if we have two graphs $G = (V, E)$ and $H = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$, then we say that H is a **subgraph** of G. In other words, a subgraph is a copy of G with vertices or edges (or both) removed. Note that if we remove a vertex, then we cannot leave behind any edges that touched the removed vertex.

If H is obtained from G by deleting only edges, then H is sometimes called a **spanning subgraph** of G. Likewise, if H is obtained from G by deleting only vertices, then H is an **induced subgraph** of G. For simplicity's sake, we will not use either of these terms; we will only refer to subgraphs in general.

Subgraphs are particularly useful when we discuss certain problems in graph theory, since we may be able to reduce a given graph to a subgraph that is easier to use or understand.

Example 7. The leftmost graph, known as the **Petersen graph**, is used frequently as an example or counterexample when proving results in graph theory. We will see the Petersen graph again later in these notes, but for now, observe that the Petersen graph contains the two rightmost graphs as subgraphs.



2.2 Properties of Vertices and Edges

Now that we have established terminology for talking about graphs in general, let's look at some terminology for talking about particular properties of a graph. In this section, assume that we are given an arbitrary graph $G = (V, E)$.

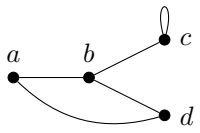
Given vertices $u, v \in V$, if $\{u, v\} \in E$ (that is, if an edge exists between u and v), then we say that u and v are **adjacent**, and we also say that the edge $\{u, v\}$ is **incident** to the vertices u and v. The set of all vertices adjacent to some vertex u is called the **neighbourhood** of u.

We say that a vertex u is of **degree** k if it is adjacent to k other vertices in the graph. We denote the degree of a vertex u by $\deg(u)$. We may formally define the degree of a vertex u as

$$\deg(u) = |\{v \in V \mid \{u, v\} \in E\}|.$$

Remark. In the case where a vertex u has an undirected loop, we must add two to the degree of u. The reason we do this is because if we rewrite the undirected loop as two directed loops—one for each direction—then u ends up being adjacent to itself twice.

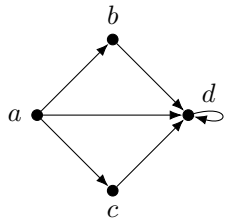
Example 8. In the following graph, we see that vertex a is adjacent to vertices b and d; vertex b is adjacent to vertices a, c, and d; vertex c is adjacent to vertices b and c; and vertex d is adjacent to vertices a and b. From this, we see that $\deg(a) = 2$, $\deg(b) = 3$, $\deg(c) = 3$ (since the loop adds two to the degree), and $\deg(d) = 2$.



For directed graphs in particular, we can define specific kinds of degrees. Namely, the **indegree** of a vertex u (denoted $\deg^-(u)$) is the number of vertices with edges that lead to u, and the **outdegree** of u (denoted

$\deg^+(u)$) is the number of vertices with edges that lead away from u. Intuitively, we can think of the indegree of u as the number of arrow heads touching u and, similarly, we can think of the outdegree of u as the number of arrow tails touching u.

Example 9. In the following graph, we see that $\deg^-(a) = 0$, $\deg^+(a) = 3$, $\deg^-(b) = \deg^+(b) = 1$, $\deg^-(c) = \deg^+(c) = 1$, $\deg^-(d) = 4$, and $\deg^+(d) = 1$.



If each vertex of an undirected graph G has the same degree, then G is a **regular** graph. A regular graph G is sometimes called ***k*-regular** if the degree of each vertex is k. A directed graph is regular only if the indegree and outdegree of each vertex are equal to one another.

Using what we know about vertices, edges, and degrees, we can now prove our first small result in graph theory: if we sum the degrees of every vertex in an undirected graph G, our total will be exactly twice the number of edges of G.

Lemma 10 (Handshake lemma)**.** *Given an undirected graph $G = (V, E)$,*

$$\sum_{u \in V} \deg(u) = 2 \cdot |E|.$$

Proof. Each edge $\{u, v\} \in E$ contributes one to the degrees of the vertices u and v. In the case where the edge is not a loop, adding one to both $\deg(u)$ and $\deg(v)$ double-counts that edge. In the case where the edge is a loop, adding two to $\deg(u)$ also double-counts that edge. Therefore, the summation of the degrees of all vertices in G will be exactly twice the number of edges in G. □

We call the previous result the "handshake lemma" because of the fun observation that, if n people shake hands with one another at a party, then exactly $2n$ hands will be shaken. (As an exercise, verify this at the next party you attend.)

There is an analogue to the handshake lemma for directed graphs, which relates the number of edges to both the indegree and the outdegree of each vertex. We will not prove it here, but we will state it out of interest.

Lemma 11 (Handshake lemma—directed graphs)**.** *Given a directed graph $G = (V, E)$,*

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

Proof. Omitted. □

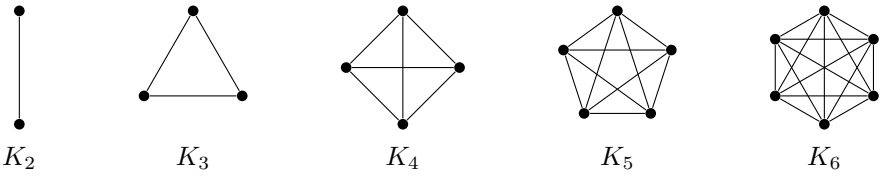
2.3 Special Graphs

Throughout our study of graph theory, we will see certain special graphs come up often in definitions, theorems, and examples. Let us look at a few of these special graphs now.

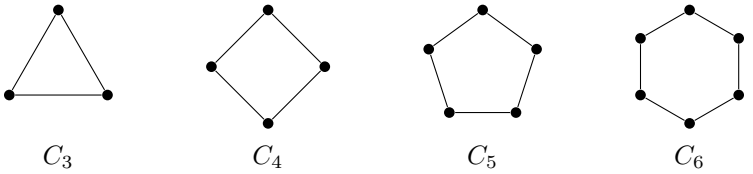
The **null graph** is the graph containing zero vertices and zero edges. The **singleton graph** is the graph containing one vertex and zero edges.

null graph singleton graph

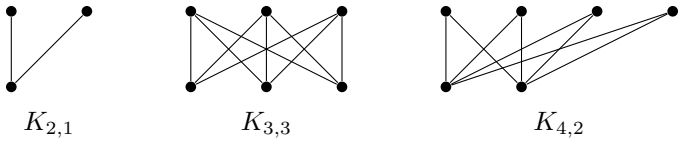
The **complete graph** on n vertices, denoted K_n, consists of n vertices and $\frac{n(n-1)}{2}$ edges joining each vertex to every other vertex exactly once. (Note that the null and singleton graphs are equivalent to the complete graphs K_0 and K_1, respectively.)



The **cycle graph** on $n \geq 3$ vertices, denoted C_n, consists of n vertices and n edges that join each vertex u_i to the vertex u_{i+1} for all $1 \leq i \leq n$. The vertex u_n is joined to the vertex u_1 to complete the cycle.



A **bipartite graph** is a graph whose vertex set V can be partitioned into two subsets V_1 and V_2 such that each edge joins a vertex in V_1 to a vertex in V_2. If V_1 contains m vertices and V_2 contains n vertices, and if each vertex in V_1 is joined to every vertex in V_2, then we say that the graph is a **complete bipartite graph**, denoted $K_{m,n}$.



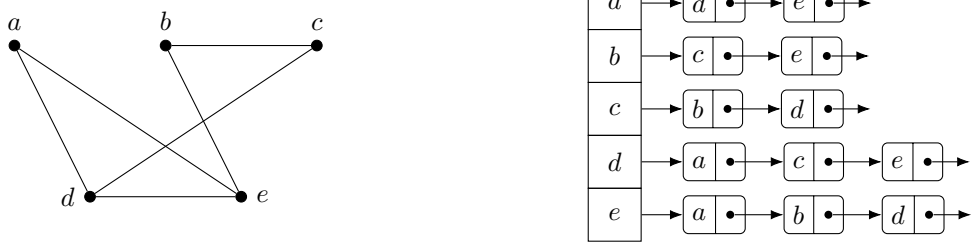
2.4 Representing Graphs

There are a number of methods of representing graphs apart from listing vertices and edges as explicit sets. Depending on how we are using graphs in an application, representing graphs using such methods could lead to ease-of-use or efficiency gains; for instance, storing vertices and edges in a data structure will result in faster access times compared to storing such data in separate locations on a disk.

The first method of representing graphs is known as an **adjacency list**. This method uses both an array data structure and a linked list data structure: vertices are represented using indices of the array, and for each array entry, the linked list stored at that index records all vertices adjacent to the vertex at that index.

Using adjacency lists, we can access vertices in constant time by indexing into the array, and we may determine all of the vertices adjacent to a given vertex by reading the elements of the linked list, which can be done in time proportional to the degree of the vertex.

Example 12. The graph at left can be represented by the adjacency list at right.



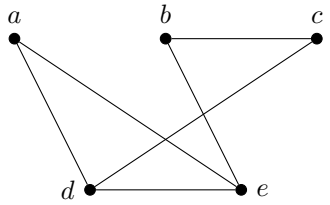
The second method of representing graphs is known as an **adjacency matrix**. As the name suggests, this method uses a matrix, or a two-dimensional array, to store vertex and edge data. Given a graph G with n vertices, the adjacency matrix of G (denoted $\mathbf{A}_G$) is an $n \times n$ matrix where the entry at row i and column j is equal to 1 when vertices u_i and u_j are adjacent and equal to 0 otherwise.

Remark. Adjacency matrices can represent loops on a vertex u_i by storing a 1 entry at position (i, i) of the matrix. We can modify our implementation of an adjacency matrix to handle graphs with multiple edges by storing the number of edges between vertices u_i and u_j at position (i, j) of the matrix.

If the graph G is undirected, then the associated adjacency matrix $\mathbf{A}_G$ will be symmetric; that is, the entries at positions (i, j) and (j, i) will be equal for all i and j. This is not always the case if G is a directed graph.

An adjacency matrix allows us to determine adjacent vertices by scanning a given row or column, which can be done in time proportional to the number of vertices, $|V|$. For graphs with many vertices and few edges, adjacency matrices perform worse than adjacency lists. However, adjacency matrices are easier to implement than adjacency lists.

Example 13. The graph at left (identical to the graph from Example 12) can be represented by the adjacency matrix at right.



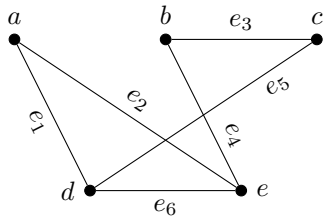
	a	b	c	d	e
a	0	0	0	1	1
b	0	0	1	0	1
c	0	1	0	1	0
d	1	0	1	0	1
e	1	1	0	1	0

The third method that we will see in this section also uses a matrix, but in a slightly different way. Instead of using rows and columns to represent vertices in a graph as we did with the adjacency matrix, we represent all vertices as rows and all edges as columns. The resulting structure is known as an **incidence matrix**. Given a graph G with n vertices and m edges, the incidence matrix of G (denoted $\mathbf{E}_G$) is an $n \times m$ matrix where the entry at row i and column j is equal to 1 if edge e_j is incident to vertex u_i and equal to 0 otherwise.

Remark. Just like with adjacency matrices, we can represent loops and multiple edges in an incidence matrix by making the appropriate changes to entries of the matrix. If we want to represent directed edges, we may use positive or negative entries depending on whether an edge is pointing toward or away from a given vertex.

In order to determine adjacent vertices in an incidence matrix, we must scan both the edge columns (to determine edges incident to our chosen vertex) and the vertex rows (to determine other vertices to which our chosen vertex is adjacent). This can be done in time proportional to the size of the matrix, $|V| \cdot |E|$, making it extremely inefficient compared to our previous methods. However, incidence matrices have their advantages; as an example, incidence matrices can distinguish multiple edges, while adjacency matrices cannot.

Example 14. The graph at left (identical to the graph from Example 12 with added edge labels) can be represented by the incidence matrix at right.



	e_1	e_2	e_3	e_4	e_5	e_6
a	1	1	0	0	0	0
b	0	0	1	1	0	0
c	0	0	1	0	1	0
d	1	0	0	0	1	1
e	0	1	0	1	0	1

Finally, keep in mind that this section is not exhaustive: there are many other methods to represent graphs, and certain methods lend themselves better to certain graphs. However, the three methods presented here are among the most commonly used methods of graph representation.