

13 Graphs

In this chapter, we introduce *graphs*, a structure that we will continue to use heavily for the rest of the course.

13.1 A Whole Lot of Definitions

A **graph** is an ordered pair (V, E) , where V (called the “vertices”) is a non-empty finite set and E (called the “edges”) is a set of pairs of vertices from V , i.e. every element of E is a set $\{u, v\}$ of size 2, where $u, v \in V$. When discussing a single graph, we will use n to denote $|V|$, the number of vertices and m to denote $|E|$, the number of edges. We use $V(G)$ to denote the vertex set of a graph G and $E(G)$ to denote the edge set of G . We call u and v the **endpoints** of the edge $e = \{u, v\}$. We say that a vertex v is **incident** to an edge e if $v \in e$.

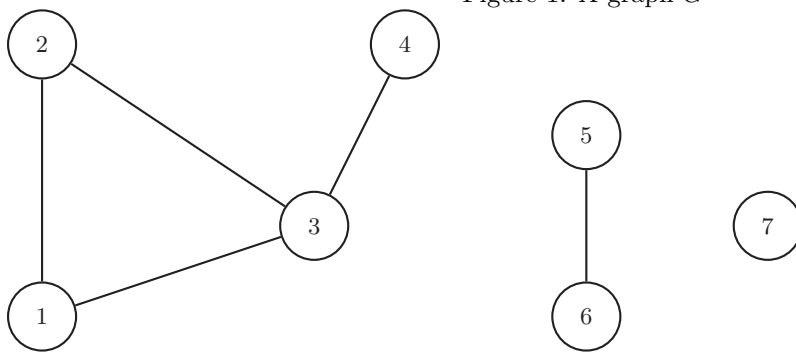


Figure 1: A graph G

We represent a graph pictorially by using circles for vertices and lines connecting them for edges. Figure 1 depicts a graph $G = (V, E)$ where the vertex set is $V := \{1, 2, 3, 4, 5, 6, 7\}$ and the edge set is $E := \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{3, 4\}, \{5, 6\}\}$. The placement of the vertices in space is arbitrary, all that matters is which vertices are connected by edges. Generally we try to draw a graph in a way which minimizes crossing edges to make this as easy to see as possible, but this is not a requirement. Figure 2 depicts the exact same graph G as Figure 1.

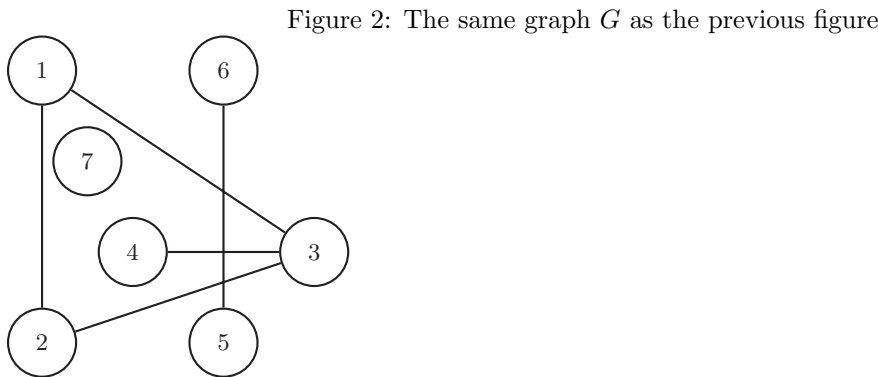


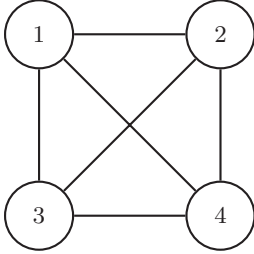
Figure 2: The same graph G as the previous figure

For a vertex (singular of vertices) v , we define the **degree** of v , written $\deg(v)$ as the number of edges incident to v . For example, in our graph G from the figures, $\deg(1) = \deg(2) = 2$, while $\deg(3) = 3$, $\deg(4) = \deg(5) = \deg(6) = 1$, and $\deg(7) = 0$. We call a vertex of degree 0 an **isolated vertex** (thus, 7 is an isolated vertex in G).

We define the **neighbors** of v as the vertices which share an edge with v (not including v itself). We use $N(v)$ to denote the set of neighbors of v . In our example G , $N(1) = \{2, 3\}$, $N(3) =$

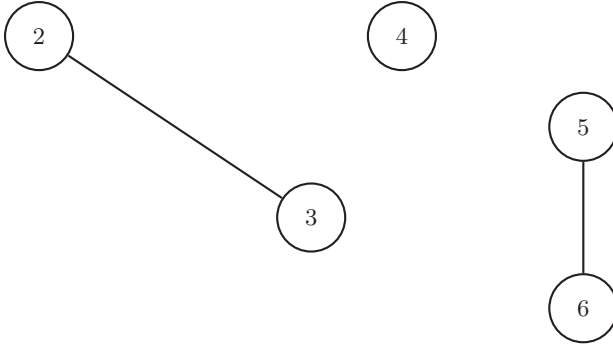
$\{1, 2, 4\}$, $N(4) = \{3\}$, and $N(7) = \emptyset$. Note that by definition $\deg(v) = |N(v)|$. We say that a graph is **k-regular** if all of its vertices have degree k . If G is an $(n - 1)$ -regular graph on n vertices, we call G a **clique**. We denote the clique on n vertices by K_n .

Figure 3: The four vertex clique K_4



We say that a graph G' is a subgraph of a graph G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For example, the graph in Figure 4 is a subgraph of the graph in Figure 1. Note that every graph is a subgraph of itself.

Figure 4: A subgraph of the graph from Figures 1 and 2



A **walk** is a sequence of vertices $v_1, \dots, v_k$ such that for every $1 \leq i < k$, $\{v_i, v_{i+1}\} \in E$ (i.e. there is an edge between each pair of consecutive vertices in the walk). A **path** is a walk which does not repeat any vertices. If a walk begins at u and ends at v , we call it a “walk from u to v ” or a $u \rightarrow v$ walk. Similarly a path that begins at u and ends at v is called a $u \rightarrow v$ path. The length of a walk or path is the number of edges it traverses, i.e. one less than the number of vertices in the sequence.

For example, in the graph from Figure 1, $1, 3, 4, 3, 1, 2, 3$ is a walk of length 6, which is not a path. $4, 3, 1, 2$ is a path of length 3. $5, 6$ is a path of length 1. Any single vertex, e.g. 2, forms a path of length 0. All of these paths are also walks.

We say that two vertices u and v are **connected** if there exists a $u \rightarrow v$ path. A **connected component** is a maximal set of vertices, all of which are connected to each other. The graph in Figure 1 has three connected components: $1, 2, 3, 4$, $5, 6$, and 7 . If all of the vertices of G are in the same connected component we say that G is **connected**.

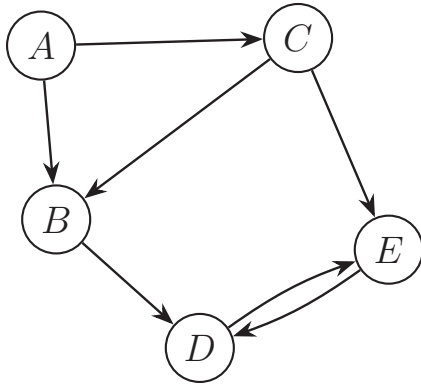
A **cycle** is a sequence of vertices $v_1, \dots, v_k, v_1$, such that $v_1, \dots, v_k$ is a path, and $\{v_k, v_1\}$ is an edge not connecting consecutive vertices in the path (note that this implies that a cycle in an undirected graph has at least 3 distinct vertices). The only cycle in the graph from Figure 1 is $1, 2, 3, 1$, which has length 3. Note that we could also write this cycle in other ways, such as $2, 1, 3, 2$. Since all these sequences correspond to the same set of vertices and edges, we consider all of them the same cycle. If G contains a cycle, we call G **cyclic**. Otherwise, we call G **acyclic**. The graph in Figure 1 is cyclic, while the graph in Figure 4 is acyclic.

13.2 Graph Variations

There are several important variants of the graph definition which occur commonly in computer science. In particular, we can add directions to the edges, in which case we have a **directed** graph, add weights to the edges, in which case we have a **weighted** graph, or allow multiple edges between the same pair of vertices, as well as self-loops, in which case we have a **multigraph**. These three variations can be used in any combination. We could, for example, have a directed multigraph. The type of graph we discussed in the previous section is sometimes called an undirected, unweighted simple graph to emphasize that it does not include any of these variations. In this course, however, we will generally omit these qualifiers. Whenever we talk about a graph we mean an undirected graph unless we explicitly specify a directed graph.

A directed graph is like an ordinary graph but instead of each edge being a set of two vertices, an edge is an ordered pair of vertices (u, v) . We represent a directed edge with an arrow from u to v . Note that while in an undirected graph we can only have one pair of edges between each pair of vertices, in a directed graph we can have two edges between the same vertices, because (u, v) and (v, u) are different edges. We depict an edge (u, v) in a directed graph using an arrow from u to v , as shown in Figure 5.

Figure 5: A directed graph



Many of the definitions from the previous section extend naturally to directed graphs, but there are a few differences. While a vertex in an undirected graph has a single set of neighbors, vertices in directed graphs have **in-neighbors** and **out-neighbors**, where $N_{in}(v) := \{u \in V \mid (u, v) \in E\}$ and $N_{out}(v) := \{u \in V \mid (v, u) \in E\}$. Similarly, we define the in-degree of v as $\deg_{in}(v) := |N_{in}(v)|$ and the out-degree of v as $|N_{out}(v)|$. In the graph from Figure 5, $N_{in}(C) = \{A\}$ and $N_{out}(C) = \{B, E\}$ so $\deg_{in}(C) = 1$ and $\deg_{out}(C) = 2$.

While a cycle in an undirected graph must have at least three vertices, in a directed graph it is possible to have a 2-vertex cycle in a directed graph. For example, D, E, D is a cycle in the graph in Figure 5. Unlike in an undirected graph, the existence of a $u \rightarrow v$ path in a directed graph does not imply the existence of a $v \rightarrow u$ path. There are several ways to extend the concept of connectivity to directed graphs. The one which will be relevant for this course is called strong connectivity. We say that two vertices u and v are **strongly connected** if G contains both a $u \rightarrow v$ path and a $v \rightarrow u$ path. A strongly connected component is a maximal set of vertices, every pair of which is strongly connected. The graph in Figure 5 contains four strongly connected components: $\{A\}$, $\{B\}$, $\{C\}$, and $\{D, E\}$.

A multigraph is like a graph but instead of being a set of edges, E is a multiset of edges, meaning that it can contain multiple copies of the same edge. We also allow a multigraph to contain loops: edges where both endpoints are the same vertex. Note an ordinary graph can be viewed as a special case of a multigraph. Some graph algorithms extend naturally to multigraphs.

A weighted graph is an ordered pair (G, w) where G is a graph and $w : E(G) \rightarrow \mathbb{R}$ is a function assigning a “weight” to each edge of G . What these weights represent will depend on the specific problem we are interested in.

13.3 Proofs with Graphs

In this section, we will see some examples of proofs involving graphs, and prove some useful graph properties.

There is a close relationship between the sum of the degrees of the vertices in a graph and the number of edges in that graph. The following theorem (often known as the “handshake lemma”) expresses this relationship. It was originally proven by Leonhard Euler in a paper which initiated the modern study of graph theory.

Theorem 61. 1. In every graph $G = (V, E)$, $\sum_{v \in V} \deg(v) = 2|E|$.

2. In every directed graph $G = (V, E)$, $\sum_{v \in V} \deg_{\text{out}}(v) = \sum_{v \in V} \deg_{\text{in}}(v) = |E|$.

Proof. We can prove each of these properties with a double counting argument, where we show that each edge is counted the same number of times on each side of the equation.

1. Let $G = (V, E)$ be a graph. We note that $\sum_{v \in V} \deg(v)$ counts each edge $\{u, v\}$ exactly twice—once in $\deg(u)$ and once in $\deg(v)$. Thus $\sum_{v \in V} \deg(v) = 2|E|$.
2. Let $G = (V, E)$ be a directed graph. We note that each edge is an out-edge of exactly one vertex, so $\sum_{v \in V} \deg_{\text{out}}(v)$ counts each edge exactly once. Similarly, each edge is an in-edge of exactly one vertex, so $\sum_{v \in V} \deg_{\text{in}}(v)$ also counts each edge exactly once. Thus, $\sum_{v \in V} \deg_{\text{out}}(v) = \sum_{v \in V} \deg_{\text{in}}(v) = |E|$.

□

We can answer other questions about the degrees in a graph easily using Theorem 61.

Theorem 62. *There does not exist a 3-regular graph with 5 vertices.*

Proof. In a 3-regular graph $G = (V, E)$ with n vertices, we have that $\sum_{v \in V} \deg(v) = \sum_{v \in V} 3 = 3n$, so by Theorem 61, $3n = 2|E|$, and thus $|E| = \frac{3}{2}n$. If $n = 5$, then $\frac{3}{2}n = \frac{15}{2} = 7.5$. Since the number of edges on a graph must be an integer, we conclude that a 3-regular graph with 5 vertices cannot exist. □

Now let’s prove some results about walks and paths.

Theorem 63. *Let $G = (V, E)$ be a graph and $u, v \in V$ be vertices. If G contains a $u \rightarrow v$ walk, then G contains a $u \rightarrow v$ path.*

Proof. Let $W = v_1, v_2, \dots, v_k$ be a *shortest* $u \rightarrow v$ walk in G (note that $v_1 = u$ and $v_k = v$). Assume for sake of contradiction that W is not a path. Then W must have some repeated vertex, i.e. there exist distinct integers $1 \leq i < j \leq k$ such that $v_i = v_j$. Then $W' := v_1, v_2, \dots, v_i, v_{j+1}, v_{j+2}, \dots, v_k$ is also a $u \rightarrow v$ walk (each pair of consecutive vertices in W' also occur consecutively in W , so since W is a walk there must be an edge between them). We note that W' is shorter than W , since we have omitted the vertices $v_{i+1}, \dots, v_j$ (and this is at least one vertex since $i < j$). Thus, W' contradicts our choice of W as the shortest $u \rightarrow v$ walk, and so our initial assumption that the shortest walk was not a path, must have been false. Therefore, there exists a $u \rightarrow v$ path in G . □

We note that Theorem 63 also holds for directed graphs with basically the same proof.

Theorem 64. *Let $G = (V, E)$ be a graph and $u, v, w \in V$ be distinct vertices. If G contains a $u \rightarrow v$ walk, and a $v \rightarrow w$ walk, then G contains a $u \rightarrow w$ path.*

Proof. By Theorem 63, since G has a $u \rightarrow v$ walk it has a $u \rightarrow v$ path $P_1 = u_1, u_2, \dots, u_k$, and since it has a $v \rightarrow w$ walk it has a $v \rightarrow w$ path $P_2 = v_1, v_2, \dots, v_\ell$. Let x be the first vertex of P_1 which also appears in P_2 (we know that P_1 and P_2 must have at least one common vertex since $u_k = v_1 = v$). Let i, j be integers such that $u_i = x = v_j$ (these are uniquely defined since P_1 and P_2 are paths and thus do not repeat vertices).

Then $P := u_1, u_2, \dots, u_{i-1}, x, v_{j+1}, v_{j+2}, \dots, v_\ell$ is a $u \rightarrow w$ path. In particular, we know that none of the vertices $u_1, \dots, u_{i-1}$ are the same as any of the vertices $v_{j+1}, \dots, v_\ell$ because we chose x as the *first* vertex which is in both P_1 and P_2 . □

13.4 Trees

A tree is a graph $T = (V, E)$ which satisfies the following three properties:

- T is connected
- T is acyclic
- $m = n - 1$ (the number of edges in T is exactly one less than the number of vertices).

In fact to show that a graph is a tree it suffices to show only two of these properties, because having any two implies having the third, we will prove this later.

You are likely familiar with *binary* trees from your data structures course, but trees in graph theory are not the same. By default, a tree is not rooted, meaning that there is no special root vertex and no notion of one vertex being a “child” of another. Furthermore, we put no restriction on the number of neighbors that a vertex in a tree can have. Figure 6 depicts my favorite tree, called the G-graph, because it is a graph in the shape of a giraffe (invented by one of my friends in undergrad).

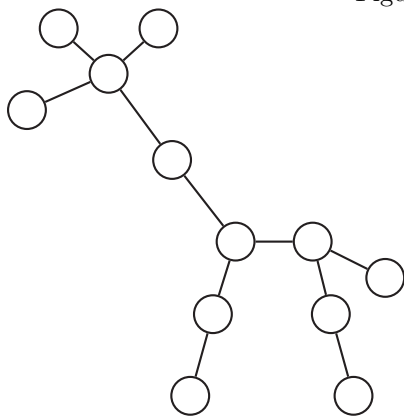


Figure 6: The G-graph, my favorite tree

We call a degree-1 vertex in a tree a **leaf**. The G-graph has 6 leaves.

Trees are important because they connect a set of vertices using as few edges as possible. We will discuss applications of this more when we talk about minimum spanning trees in a few chapters, for now we will just gain some practice working with trees by proving the following theorem.

Theorem 65. Any n -vertex graph $T = (V, E)$ which satisfies two of the following three properties satisfies the third one as well:

- T is connected
- T is acyclic
- $m = n - 1$ (the number of edges in T is exactly one less than the number of vertices).

Proof. We have three claims to prove:

1. First we show that if T is acyclic with $m = n - 1$, then T is connected.

Consider building T by adding its edges one at a time, starting from an empty graph (one with all of T 's vertices, but no edges).

Initially this graph has n connected components. Every time we add an edge which connects two vertices from different connected components, they merge and the number of connected components decreases by one.

If we were to add an edge connecting two vertices u and v from the same connected component, this would form a cycle: since u and v were connected, T contained a $u \rightarrow v$ path, and adding

the edge $\{v, u\}$ to this path gives a cycle. Since T is acyclic, we know that this will never happen. Every edge we add must merge two connected components.

Since T begins with n connected components and every edge we add merges two of them, after adding all $n - 1$ edges, T will have $n - (n - 1) = 1$ connected component, meaning that all of T is connected.

2. Next we prove that if T is connected with $m = n - 1$, then T is acyclic.

We use a similar argument to the previous part. Suppose we create T by adding edges 1 at a time starting from an empty graph. Each edge we add can merge at most two connected components. In order to get from n components down to 1 with only $n - 1$ edges, every single edge we add must merge two different connected components.

An edge $\{u, v\}$ that connects two different connected components cannot form a cycle (if it did that would imply that u and v were already connected via the rest of the cycle and therefore were in the same connected component). Thus, none of the edges we add can create a cycle and so T is acyclic.

3. Finally we show that if T is connected and acyclic then T has $n - 1$ edges.

To illustrate how to apply induction to graphs, we prove this by induction on n .

- Base Case: If $n = 1$, then the only possible graph consists of a single vertex and no edges. This graph is connected and acyclic and it has $n - 1$ edges.
- Induction Hypothesis: Suppose that for every $k < n$, every k -vertex graph which is connected and acyclic has exactly $k - 1$ edges.
- Induction Step: Let G be an n -vertex tree. Since G is acyclic and has at least one edge (since it is connected and $n > 1$) it must contain a vertex of degree 1 (otherwise we could obtain a cycle by walking around the graph arbitrarily exiting each vertex with a different edge than the one we used to enter until we repeat a vertex). Removing this degree 1 vertex yields a graph on $n - 1$ vertices which is still connected and acyclic, so by our IH, this graph has $n - 2$ edges. Adding back the removed vertex and edge we find that G has $n - 1$ edges as desired.

□

13.5 Representing Graphs

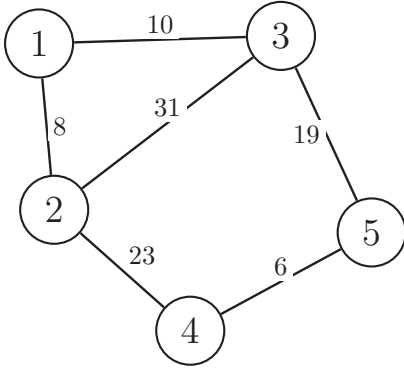
In order to analyze the runtime of algorithms on graphs, we need to define how we are encoding graphs as strings (since under our RAM model the input and output of every algorithm are just strings). There are two main ways of encoding graphs: adjacency matrices and adjacency lists. We will discuss both of them here (there are many other ways of encoding graphs which are useful in more specific applications, but we will not get into those in this course). In this section we will assume that the vertex set is $V := \{1, \dots, n\}$.

In the adjacency matrix representation, we encode $G = (V, E)$ with an $n \times n$ array, where $A[i, j] = 1$ if $\{i, j\} \in E$ and 0 otherwise. Note that the adjacency matrix of an undirected graph is symmetric. In contrast for a directed graph we define $A[i, j]$ to be 1 if $(i, j) \in E$, so $A[i, j] = A[j, i]$ only if the directed graph has directed edges between i and j in both directions.

For a weighted graph, we can compactly encode the weights by letting $A[i, j] = w(\{i, j\})$ if $\{i, j\} \in E$ (if our graph can include 0-weight edges, however, we will have to select some value other than 0 to represent “this edge does not exist”). For example, the weighted graph in Figure 7 can be represented by the adjacency matrix:

$$\begin{bmatrix} 0 & 8 & 10 & 0 & 0 \\ 8 & 0 & 31 & 23 & 0 \\ 10 & 31 & 0 & 0 & 19 \\ 0 & 23 & 0 & 0 & 6 \\ 0 & 0 & 19 & 6 & 0 \end{bmatrix}$$

Figure 7: A weighted graph



An adjacency matrix takes $O(n^2)$ space. It allows us to check whether a particular edge $\{u, v\}$ is present in the graph in time $O(1)$, we just check $A[u, v]$. Iterating through all of the neighbors of a vertex v takes $\Theta(n)$ time, however, since we need to loop over all of the entries in row v of the adjacency matrix.

Unfortunately for the humble adjacency matrix, iterating through the neighbors of a vertex is a much more common component of graph algorithms than checking for the presence of a random edge. Nevertheless, adjacency matrices are valuable to understand, because there are many contexts when an input which we want to treat as a graph is represented in an adjacency-matrix like form (e.g. if we are just given a weight function which takes two vertices and returns the weight of the connection between them, then we can treat this function as an adjacency matrix).

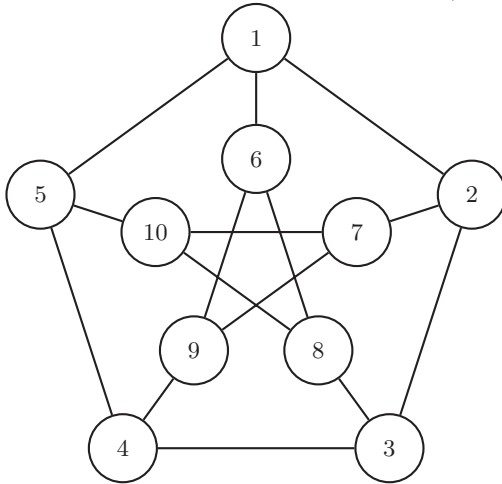
The more common way to represent graphs is using an adjacency list. Here we use an array A of size n , where $A[i]$ is a linked list containing all of the neighbors of vertex i . If G is a directed graph, then $A[i]$ is a linked list of the out-neighbors of i . If G is a weighted graph, then instead of the nodes in our linked lists storing just vertices, they store (vertex, weight) pairs, where the weight of the nodes storing j in $A[i]$ is the weight of the edge $\{i, j\}$. The graph in Figure 7 would be represented with the following adjacency list:

$$\begin{aligned}
 A[1] &: (2, 8) \rightarrow (3, 10) \\
 A[2] &: (1, 8) \rightarrow (3, 31) \rightarrow (4, 23) \\
 A[3] &: (1, 10) \rightarrow (2, 31) \rightarrow (5, 19) \\
 A[4] &: (2, 23) \rightarrow (5, 6) \\
 A[5] &: (3, 19) \rightarrow (4, 6)
 \end{aligned}$$

With an adjacency list, checking whether a particular edge $\{u, v\}$ exists takes $O(\deg(u))$ time, since we need to iterate through u 's entire linked list, which has $\deg(u)$ nodes. Iterating through the neighbors of v , however, takes only $O(\deg(v))$ time, a substantial improvement over adjacency matrices unless our graph is very dense (meaning the average degree is $\Omega(n)$). In this course when an algorithm outputs a graph or takes one as input, we will assume that it is represented as an adjacency list unless otherwise specified.

13.6 Comprehension Questions

1. Consider the pictured graph $G = (V, E)$ (this graph is known as the Petersen graph):



- How many vertices does this graph have?
 - How many edges does this graph have?
 - What is $N(6)$ in this graph?
 - What is $\deg(5)$ in this graph?
 - Give an example of a cycle in this graph. What is its length?
 - Give an example of a path of length 6 in this graph.
 - Give an example of a walk of length 12 in this graph.
 - Does this graph contain K_3 as a subgraph?
 - Is this graph 2-regular?
 - Represent G as an adjacency matrix
 - Represent G as an adjacency list
2. Let G be a graph with 400 edges. What is the sum of the degrees of all the vertices in G ?
3. Let $G = (V, E)$ be a directed graph and let $u, v \in V$. Which of the following are necessarily true? (Select all that apply)
- If G contains a $u \rightarrow v$ walk then G contains a $u \rightarrow v$ path
 - If G contains a $u \rightarrow v$ path then G contains a $u \rightarrow v$ walk
 - If G contains a $u \rightarrow v$ walk then G contains a $v \rightarrow u$ walk
 - If G contains a $u \rightarrow v$ path then G contains a $v \rightarrow u$ walk
4. Which of the following is NOT a property of every tree T ?
- T is connected
 - T is acyclic
 - T has a root
 - $m = n - 1$ (the number of edges in T is exactly one less than the number of vertices).
5. What is the runtime of checking whether an edge $\{u, v\}$ exists in a graph G ...
- ...assuming G is represented by an adjacency list?
 - ...assuming G is represented by an adjacency matrix?

13.7 Exercises

1. Prove that K_n has $\frac{n(n-1)}{2}$ edges.
2. Let G be a graph with minimum degree d (meaning that a vertex of G with the smallest degree has degree d). Prove that G has a path of length d .
3. Let T be a tree with at least two vertices. Prove that T has at least two leaves.
4. Let $G = (V, E)$ be a graph. Prove that G is a tree if and only if for every pair of distinct vertices $u, v \in V$ there is a unique path from u to v in G .
5. Let G be a graph with $n \geq 2$ vertices. Prove that G contains two vertices with the same degree.
6. Let G be a graph where every edge has one endpoint with even degree and one endpoint with odd degree. Prove that G has an even number of edges.
7. Prove that every 2-regular graph is composed of one or more disjoint cycles.
8. Prove that every tree with $n \geq 2$ vertices has $2 + \sum_{v \in V: \deg(v) \geq 3} (\deg(v) - 2)$ leaves.
9. Let G be a graph, let C be a cycle in G , let k be a positive integer and let P be a path of length k between two vertices of C . Prove that G contains a cycle of length at least $\sqrt{k}$.