Basics of Graph Theory

1 Basic notions

A *simple graph* $G = (V, E)$ consists of V, a nonempty set of vertices, and E, a set of unordered pairs of distinct elements of V called edges.

Simple graphs have their limits in modeling the real world. Instead, we use *multigraphs*, which consist of vertices and undirected edges between these vertices, with multiple edges between pairs of vertices allowed. A *multigraph* $G = (V, E)$ consists of a set V of vertices, a set E of edges, and a function g from E to $\{\{u, v\} : u, v \in V, u \in v\}$. The edges e_1 and e_2 are called multiple or parallel edges if $f(e_1) = f(e_2)$. Note that simple graphs are all multigraphs.

A computer network may contain a line from a computer to itself (it is called a loop). How can we model this? We cannot use multigraphs to model this, since loops, which are edges from a vertex to itself, are not allowed in multigraphs. Instead we use pseudographs, which are more general than multigraphs.

A *pseudograph* $G = (V, E)$ consists of a set V of vertices, a set E of edges, and a function g from E to $\{\{u, v\} : u, v \in V\}$. An edge if a loop if $f(e) = \{u, u\} = \{u\}$ for some $u \in V$.

How can we model a water pipe from A to B, and water can only flow from A to B. We use directed arrow $A \to B$. In graph theory, we also have *directed graphs*.

A *directed graph* $G = (V, E)$ consists of a set V of vertices, a set E of edges, that are ordered pairs of elements of V.

Similarly, we can have the following definition of directed multigraphs. A *directed multigraph* $G = (V, E)$ consists of a set V of vertices, a set E of edges, and a function f from E to $\{(u, v) : u, v \in V\}$. The edges e_1 and e_2 are multiple edges if $f(e_1) = f(e_2)$.

Now we introduce some basic terminology that describes the vertices and edges of undirected graphs.

Definition 1.1 *Two vertices u and v in an undirected graph G are called adjacent (or neighbors) in G if $\{u, v\}$ is an edge of G. If $e = \{u, v\}$, the edge e is called incident with the vertices u and v. They edge e is also said to connect u and v. The vertices u and v are called endpoints of the edge $\{u, v\}$.*

Definition 1.2 *The degree of a vertex in an (undirected) graph is the number of edges incident with it, except that a loop at a vertex contributes two to the degree of that vertex. The degree of the vertex v is denoted by $deg(v)$ or $d(v)$.*

Definition 1.3 *A vertex of degree zero is called an isolated vertex.*

What do we get when we add the degrees of all the vertices of a graph $G = (V, E)$? The following theorem is due to Euler.

Theorem 1.4 *(***The Handshaking Theorem***) Let $G = (V, E)$ be an undirected graph with e edges. Then*

$$2e = \sum_{v \in V} deg(v).$$

Question: How many edges are there in a graph with 10 vertices each of which of degree 6?

Corollary: An undirected graph has an even number of vertices of odd degree.

When (u, v) is an edge of the graph G with directed edges, u is said to be *adjacent to* v and v is said to be *adjacent from* u. The vertex u is called the initial vertex of (u, v), and v is called the *terminal* or *end* vertex of (u, v).

The initial vertex and terminal vertex of a loop are the same. Since the edges in graphs with directed edges are ordered pairs, the definition of the degree of a vertex can be defined to reflect the number of edges with this vertex as the initial vertex and as the terminal vertex.

Definition 1.5 *In a graph with directed edges the in-degree of a vertex v, denoted by $deg^-(v)$, is the number of edges with v as their terminal vertex. The out-degree of v, denoted by $deg^+(v)$, is the number of edges with v as their initial vertex.*

Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of this vertex.

Theorem 1.6 *Let $G = (V, E)$ be a graph with directed edges. Then*

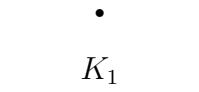
$$\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = |E|$$

where $|E|$ is the size of E.

1.1 Complete Graphs

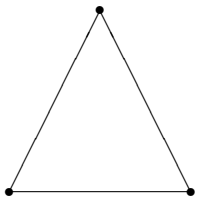
The complete graph of n vertices, denoted by K_n, is the simple graph that contains exactly one edge between each pair of distinct vertices.

The graph K_n for $n = 1, 2, 3, 4, 5, 6$ are displayed as follows:

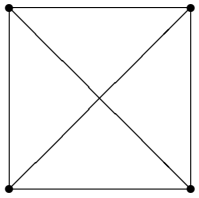


K_1

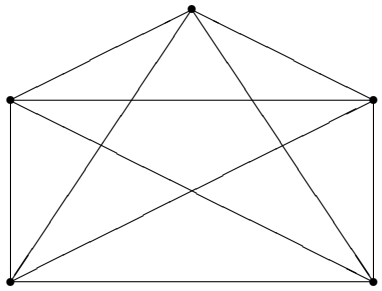
K_2



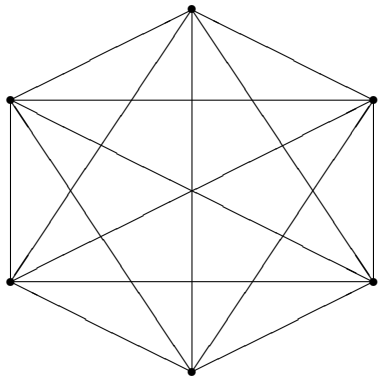
K_3



K_4



K_5



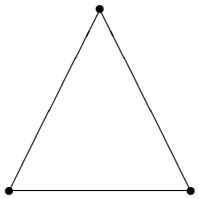
K_6

1.2 Cycles

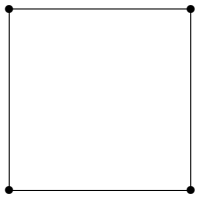
The cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \cdots, v_n$ and edges

$$\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \cdots, \{v_{n-1}, v_n\}, \text{and } \{v_n, v_1\}.$$

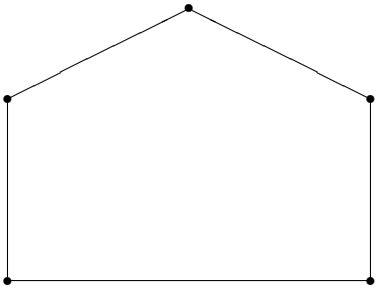
The cycles C_n for $n = 3, 4, 5, 6$ are displayed as follows:



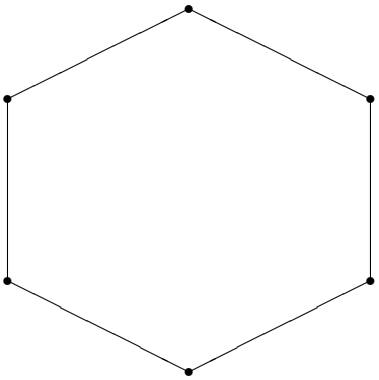
C_3



C_4



C_5

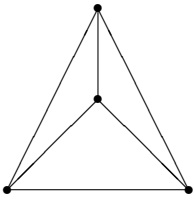


C_6

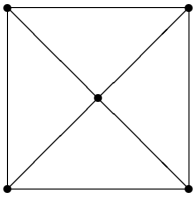
1.3 Wheels

We obtain the wheel W_n when we add an additional vertex to the cycle C_n, for $n \geq 3$, and connect this new vertex to each of the n vertices in C_n, by new edges.

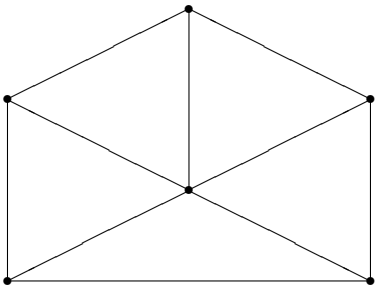
The wheels W_3, W_4, W_5 and W_6 are displayed as follows:



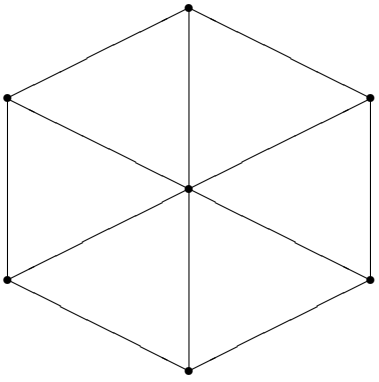
W_3



W_4



W_5



W_6

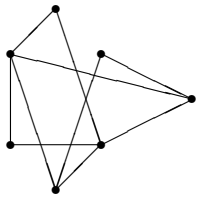
1.4 Bipartite graphs

Sometimes a graph has the property that its vertex set can be divided into two disjoint subsets such that each edge connects a vertex in one of these subsets to a vertex in the other subset.

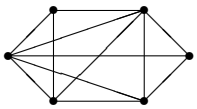
Definition 1.7 *A simple graph G is called bipartite if its vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex in V_2 (so that no edge in G connects either two vertices in V_1 or two vertices in V_2).*

Question: Among cycles C_3, C_4, C_5 and C_6, which are bipartite graphs?

Example: Are the following graphs bipartite?



G_1



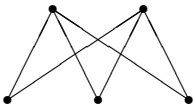
G_2

□

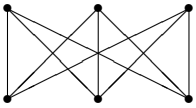
1.5 Complete bipartite graphs

The *Complete bipartite graph* $K_{m,n}$ is the graph that has its vertex set partitioned into two subsets of m and n vertices, respectively, and that there is an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.

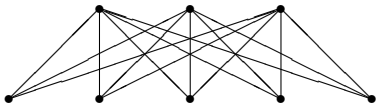
The complete bipartite graphs $K_{2,3}, K_{3,3}$ and $K_{3,5}$ are displayed as follows:



$K_{2,3}$



$K_{3,3}$



$K_{3,5}$

Can you figure out $K_{3,4}, K_{2,6}$?

1.6 From old to new

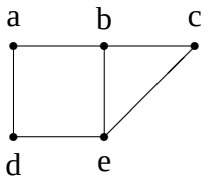
Definition 1.8 *A subgraph of a graph $G = (V, E)$ is a graph $H = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$.*

Example: K_3 and K_4 are subgraphs of K_5. □

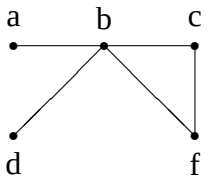
Two or more graphs can be combined in various ways. The new graph that contains all the vertices and edges of these graphs of these graphs is called the *union* of the graphs.

Definition 1.9 *The union of two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. The union of G_1 and G_2 is denoted by $G_1 \cup G_2$.*

Example: Find the union of the graphs G_1 and G_2 displayed as follows:

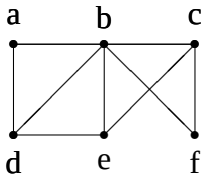


G_1



G_2

□

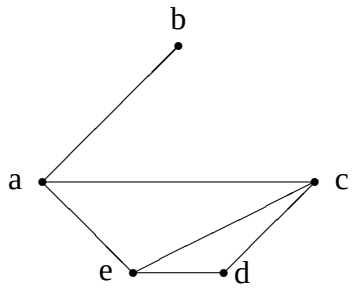


$G_1 \cup G_2$

1.7 Representation of graphs

There are many ways to represent graphs. One way to represent a graph without multiple edges is to list all the edges of this graph. Another way is to use *adjacency lists*, which specify the vertices that are adjacent to each vertex of the graph.

Example: Use adjacency lists to describe the simple graph given in the following figure:



□

The next table lists those vertices adjacent to each of the vertices of the graph.

Vertex	Adjacent Vertices
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

Carrying out graph algorithms using the representation of graphs by lists of edges, or by adjacency lists, can be cumbersome if there are many edges in the graph. To simplify computation, graphs can be represented using matrices. Two types of matrices commonly used to represent graphs will be presented here. One is based on the adjacency of vertices, and the other is based on incidence of vertices and edges.

1.8 Adjacency Matrices

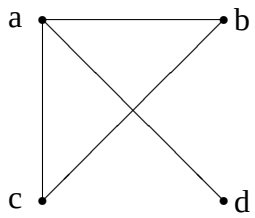
Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Suppose that the vertices of G are listed arbitrarily as $v_1, v_2, \cdots, v_n$. The *adjacency matrix* A (or A_G) of G, with respect to this listing of the vertices, is the $n \times n$ zero-one matrix with 1 as its (i, j)-th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when v_i and v_j are not adjacent. In other words, if its adjacency matrix is $A = [a_{ij}]$, then

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

1. Note that an adjacency matrix of a graph is based on the ordering chosen for the vertices. Hence there are as many as $n!$ different adjacency matrices for a graph with n vertices, since there are $n!$ different orderings of n vertices.

2. The adjacency matrix of a simple graph is symmetric, that is, $a_{ij} = a_{ji}$, since both of these entries are 1 when v_i and v_j are adjacent, and both are 0 otherwise. Furthermore, since a simple graph has no loops, each entry a_{ii}, $i = 1, \cdots, n$, is 0.

Example: Use an adjacency matrix to represent the following graph.



□

Solution: We order the vertices as a, b, c, d. The matrix representing this graph is:

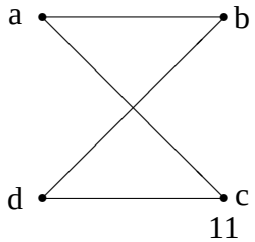
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Example: Draw a graph with the adjacency matrix

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

with respect to the ordering of vertices a, b, c, d. □

Solution: A graph with this adjacency matrix is:



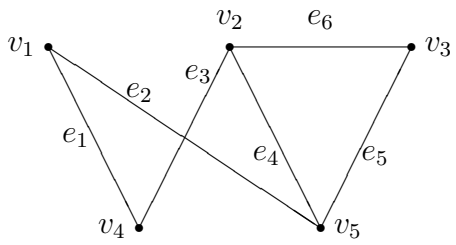
1.9 Incidence Matrices

Another common way to represent graphs is to use *incidence matrices*. Let $G = (V, E)$ be an undirected graph.

Suppose that $v_1, v_2, \cdots, v_n$ are the vertices and $e_1, e_2, \cdots, e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the $n \times m$ matrix $M = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_j, \\ 0 & \text{otherwise.} \end{cases}$$

Example: Represent the following graph with an incidence matrix:



□

Solution: The incidence matrix is

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

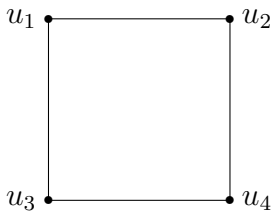
Incidence matrices can also be used to represent multiple edges and loops. Multiple edges are represented in the incidence matrix using columns with identical entries, since these edges are incident with the same pair of vertices. Loops are represented using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with this loop.

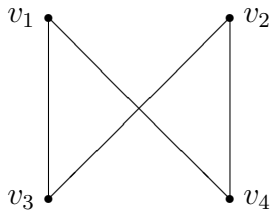
1.10 Isomorphisms of graphs

Definition 1.10 *The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a one-to-one and onto function f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2, for all a and b in V_1. Such a function is called an isomorphism.*

In other words, when two simple graphs are isomorphic, there is a one-to-one correspondence between vertices of the two graphs that preserves the adjacency relationship. Isomorphism of simple graphs is an equivalence relation.

Example: Show that the graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic.





□

Solution: The function f with $f(u_1) = v_1$, $f(u_2) = v_4$, $f(u_3) = v_3$, $f(u_4) = v_2$ is a one-one correspondence between V and V'.

We leave the verification that f is a one-one correspondence preserving adjacency to you.

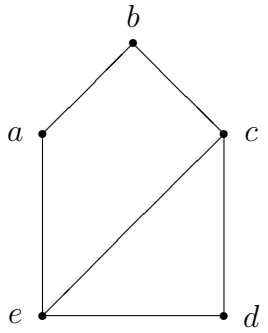
Example: We can show that two simple graphs are not isomorphic by showing that they do not share a property that isomorphic simple graphs graphs must

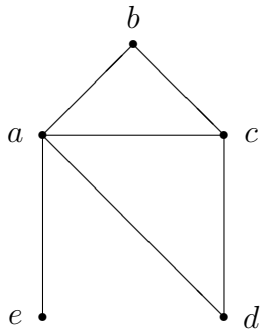
both have — such a property is called an *invariant* with respect to isomorphism of simple graphs:

1. isomorphic simple graphs must have the same number of vertices
2. isomorphic simple graphs must have the same number of edges
3. the degrees of the vertices in isomorphic simple graphs must be the same.

□

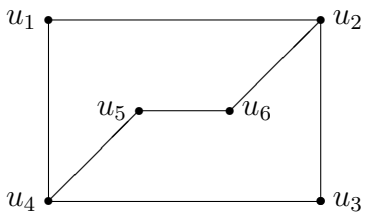
Example:
Show that the following two graphs are not isomorphic.

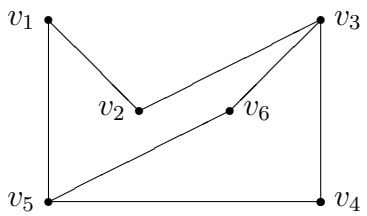




□

Example: Determine whether the following graphs G_1 and G_2 are isomorphic.





□

2 Connectivity

Many problems can be modeled with paths formed by traveling along the edges of graphs. For instance, the problem of determining whether a message can be sent between two computers using intermediate links can be studied with a graph model. Problems of efficiently planning routes for mail delivery, diagnostics in computer networks, etc., can be solved using models that involve paths in graphs.

2.1 Paths

Informally, a *path* is a sequence of edges that begins at a vertex of a graph and travels along edges of the graph, always connecting pairs of adjacent vertices.

Definition 2.1 *Let n be a nonnegative integer and G be an undirected graph. A path of length n from u to v in G is a sequence of n edges $e_1, e_2, \cdots, e_n$ of G such that $f(e_1) = \{x_0, x_1\}$, $f(e_2) = \{x_1, x_2\}$, $\cdots$, $f(e_n) = \{x_{n-1}, x_n\}$, where $x_0 = u$, and $x_n = v$. The path is a circuit if it begins and ends at the same vertex, that is, if $u = v$, and has length greater than 0.*

The path or circuit is said to pass through the vertices $x_1, x_2, \cdots, x_{n-1}$ or traverse the edges $e_1, e_2, \cdots, e_n$.

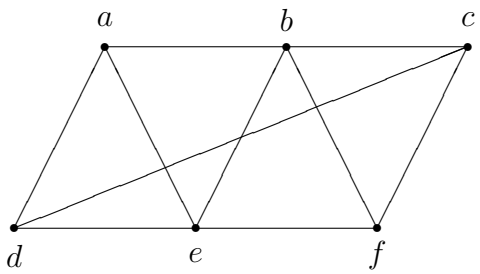
A simple path or circuit is simple if it does not contain the same edge more than once.

Remark: Note that if a graph G is a multigraph, there may be more than one path that passes through the same sequence of vertices. When it is not necessary to distinguish between multi edges, we will denote a path $e_1, e_2, \cdots, e_n$ where $f(e_i) = \{x_{i-1}, x_i\}$ for $i = 1, 2, \cdots, n$ by this vertex sequence $x_0, x_1, x_2, \cdots, x_n$. This notation identifies a path only up to the vertices it passes through. Thus, when the graph G is simple, we denote this path by its vertex sequence $x_0, x_1, x_2, \cdots, x_n$, because listing these vertices uniquely determines the path.

Example: In the simple graph shown below, a, d, c, f, e is a simple path of length 4 since $\{a, d\}$, $\{d, c\}$, $\{c, f\}$ and $\{f, e\}$ are all edges. However, d, e, c, a is not a path, since $\{e, c\}$ is not an edge.

b, c, f, e, b is a circuit of length 4 since $\{b, c\}$, $\{c, f\}$, $\{f, e\}$ and $\{e, b\}$ are edges, and this path begins and ends at b.

The path a, b, e, d, a, b is not simple since it contains the edge $\{a, b\}$ twice.



□

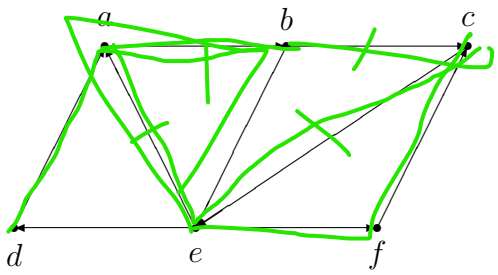
The following is the definition of paths and circuits for directed graphs.

Definition 2.2 *Let n be a nonnegative integer and G be a directed graph. A path of length n from u to v in G is a sequence of n edges $e_1, e_2, \cdots, e_n$ of G such that $f(e_1) = (x_0, x_1)$, $f(e_2) = (x_1, x_2)$, $\cdots$, $f(e_n) = (x_{n-1}, x_n)$, where $x_0 = u$, and $x_n = v$. The path is a circuit or cycle if it begins and ends at the same vertex, and has length greater than 0.*

A simple path or circuit is called simple if it does not contain the same edge more than once.

Remark: Again, when there are no multi edges in a directed path $e_1, e_2, \cdots, e_n$ where $f(e_i) = (x_{i-1}, x_i)$ for $i = 1, 2, \cdots, n$, we can denote this path by its vertex sequence $x_0, x_1, x_2, \cdots, x_n$.

Example: Find circuits of lengths 3, 4 and 5 in the following directed graph.



□

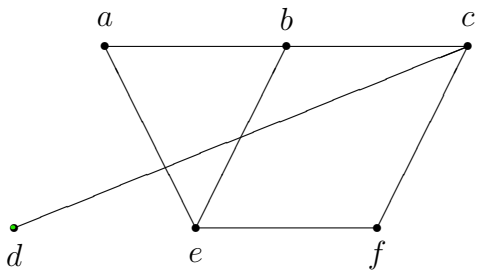
2.2 Connectedness

When does a computer network have the property that every pair of computers can share information, if messages can be sent through one or more intermediate computers? When a graph is used to represent this computer network, where vertices represent the computers and edges represent the communication links, this question becomes: when is there always a graph between two vertices in the graph?

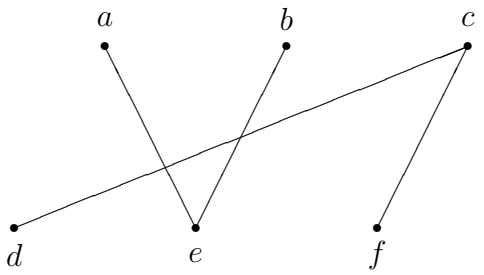
Definition 2.3 *An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph.*

Thus any two computers in the network can communicate if and only if the graph of this network is connected.

Example:



Connected graph: G_1



Non-connected graph: G_2

□

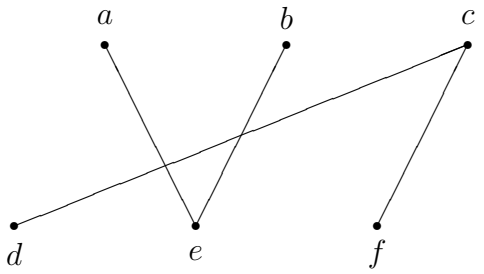
Theorem 2.4 *There is a simple path between every pair of distinct vertices of a connected undirected graph.*

Proof: Let u and v be two distinct vertices of the connected undirected graph $G = (V, E)$. Since G is connected, there is at least one path between u and v.

Let $u = v_0, v_1, \cdots, v_n = v$ be the vertex sequence of a path of least length. This path of least length is simple. To see this, suppose it is simple, then $v_i = v_j$ for some $i < j \leq n$. This means that there is a path from u to v of shorter length with vertex sequence $v_0, \cdots, v_{i-1}, v_j, \cdots, x_n$, obtained by deleting the edges corresponding to the vertex sequence $v_i, \cdots, v_{j-1}$. □

Definition 2.5 *A graph that is not connected is the union of two or more connected subgraphs, each pair of which has no vertex in common. These disjoint connected subgraphs are called the connected components of the graph.*

Example: Find the connected components of the following graph:



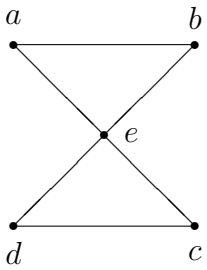
□

2.3 Euler circuits and Euler paths

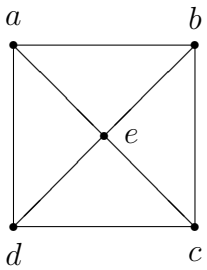
Can we travel along the edges of a graph starting at a vertex and returning to it by traversing each edge of the graph exactly once? Similarly, can we travel along the edges of a graph starting from a vertex and returning to it while visiting each vertex of the graph exactly once? Although these questions seem to be similar, the first question, which asks whether a graph has an *Euler circuit*, can be answered for all graphs, while the second question, which asks whether a graph has a *Hamiltonian circuit*, is quite difficult to solve.

Definition 2.6 *An Euler circuit in a graph G is a simple circuit containing every edge of G. An Euler path in G is a simple path containing every edge of G.*

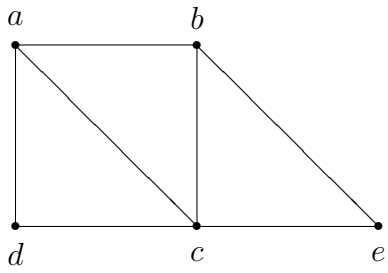
Example: Which of the following graphs have an Euler circuit?



G_1



G_2



G_3

□

There are simple criteria for determining whether a multigraph has an Euler circuit or an Euler path. Euler discovered them when he solved the famous K¨onigsberg bridge problem.

What can we say if a connected multigraph has an Euler circuit? What we can show is that every vertex must have even degrees.

To show this, first note that an Euler circuit begins with a vertex a and continues with an edge incident to a, say $\{a, b\}$. The edge $\{a, b\}$ contributes one to $deg(a)$. Each time the circuit passes through a vertex it contributes two to the vertex's degree since the circuit enters via an edge incident with this vertex and leaves via another such edge. Finally, the circuit terminates where it started, contributing one to $deg(a)$. Therefore, $deg(a)$ is even, because the circuit contributes one when it begins, one when it ends, and two every time it passes through a (if it ever does).

A vertex other than a has even degree because the circuit contributes two to its degrees each time it passes through the vertex.

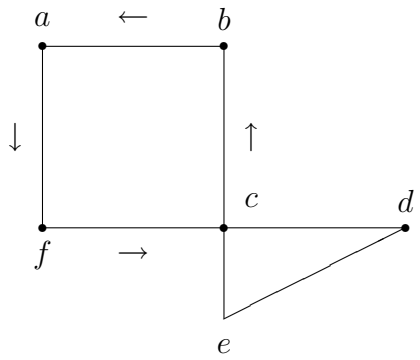
Thus, we can conclude that if a connected graph has an Euler circuit, then every vertex must have even degree.

Is this necessary condition for the existence of an Euler circuit also sufficient? That is, must an Euler circuit exist in a connected multigraph if all vertices have even degree? The answer is 'yes'.

Suppose that G is a connected multigraph and the degree of every vertex of G is even. We will form a simple circuit that begins at an arbitrary vertex a of G. Let $v_0 = a$.

First, we arbitrarily choose an edge $\{v_0, v_1\}$ incident with a. We continue by building a simple path $\{v_0, v_1\}$, $\{v_1, v_2\}$, $\cdots$, $\{x_{n-1}, x_n\}$ that is as long as possible. The path terminates since the graph has a finite number of edges. It begins at a with an edge of the form $\{a, v_1\}$, and it terminates at a with an edges of form $\{u, a\}$. This follows because each time the path goes through a vertex with even degree, it uses only one edge to enter this vertex, so that at least one edge remains for the path to leave the vertex.

This path may use all the edges, or it may not.



If all the edges have been used in the construction, then we know that an Euler circuit is constructed. Otherwise, consider the subgraph H obtained from G by deleting the edges already used and vertices that are not incident with any remaining edges.

Since G is connected, there is at least one vertex in common with the circuit that has been deleted. Let w be such a vertex.

Note that every vertex in H has even degree (why?). However, H may not be connected. Beginning with w, construct a simple path in H by choosing edges as long as possible, as was done in G. This path must terminate at w.

Continue this process, until all edges have been used. (The process must terminates since there are only a finite number of edges in G.) This produces an Euler circuit. The construction shows that if the vertices of a connected multigraph all have even degree, then the graph has an Euler circuit.

We summarize these results in the following theorem.

Theorem 2.7 *A connected multigraph has an Euler circuit if and only if each of its vertices has even degree.*

We can now argue why the graphs given in a previous example do not have Euler circuit. The following is an algorithm for the construction of Euler circuits.

Constructing Euler Circuits

procedure *Euler* (G: connected multigraph with all vertices of even degree)

circuit:= a circuit in G beginning at an arbitrarily chosen vertex with edges successively added to form a path that returns to this vertex

$H := G$ with the edges of this circuit removed

while H has edges

begin

subcircuit := a circuit in H beginning at a vertex in H that also is an endpoint of an edge of circuit

$H := H$ with edges of *subcircuit* and all isolated vertices removed

circuit := circuit with *subcircuit* inserted at the appropriate vertex

end { circuit is an Ruler circuit }

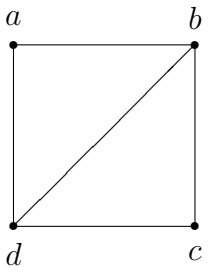
Now we show that a connected multigraph has an Euler path (and not an Euler circuit) if and only if it has exactly two vertices of odd degree.

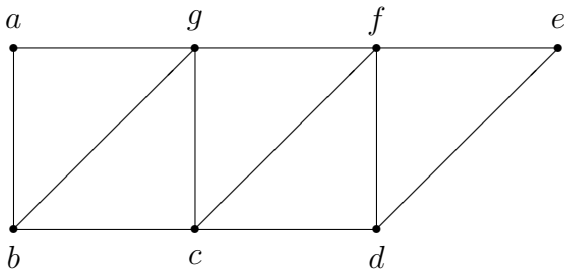
First suppose that a connected multigraph does have an Euler path from a to b, but not an Euler circuit. Then clearly, the first edge of the path contributes one to the degree of a, and the last edge in the path contributes one to the degree of b. Every time the path goes through a or b there is a contribution of two to the degree of a or b respectively. Consequently, both a and b had odd degree. Every other vertex has even degree, since the path contributes two to the degree of a vertex whenever it passes through it.

Now consider the converse. Suppose that a graph has exactly two vertices of odd degree, say a and b. Consider the larger graph made up of the original graph with the addition of an edge $\{a, b\}$. Every vertex of this larger graph has even degree, so that there is an Euler circuit. Then removal of the new edge produces an Euler path in the original graph.

Theorem 2.8 *A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.*

Example: Show that the graphs below have Euler paths.





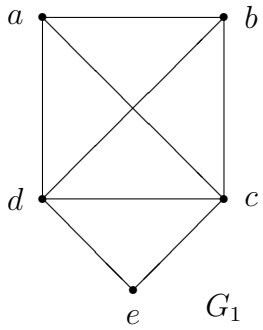
□

2.4 Hamilton paths and circuits

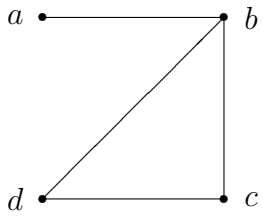
We have developed necessary and sufficient conditions for the existence of paths and circuits that contain every edge of a multigraph exactly once. Can we do the same for simple paths and circuits that contain every vertex of the graph exactly once?

Definition 2.9 *A path $v_0, v_1, \cdots, v_n$ in the graph $G = (V, E)$ is called a Hamilton path if $V = \{v_0, v_1, \cdots, v_{n-1}, v_n$ and $v_i \neq x_j$ for $0 \leq i < j \leq n$. A circuit $v_0, v_1, \cdots, v_{n-1}, v_n, v_0$ (with $n > 1$) in a graph $G = (V, E)$ is called a Hamilton circuit if $v_0, v_1, \cdots, v_{n-1}, v_n$ is a Hamilton path.*

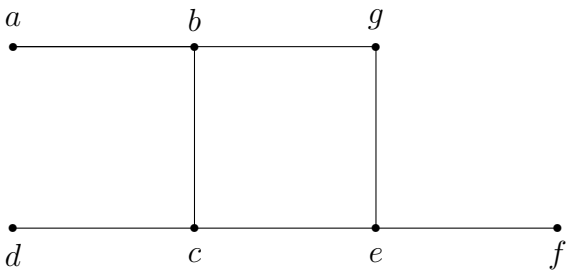
Example: Which of the following graphs have a Hamilton circuit or, if not, a Hamilton path?



G_1



G_2



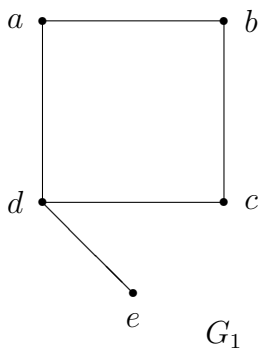
G_3

Solution: G_1 has a Hamilton circuit a, b, c, d, e, a. There is no Hamilton circuit in G_2 because any circuit containing every vertex must contain the edge $\{a, b\}$ twice. But G_2 does have a Hamilton path: a, b, c, d.

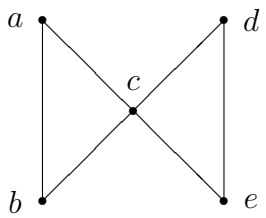
G_3 has neither a Hamilton circuit nor a Hamilton path, since any path containing all vertices must contain one of the edges $\{a, b\}$, $\{e, f\}$ and $\{c, d\}$ more than once. □

Is there a simple way to determine whether a graph has a Hamilton circuit or path? At first, it might seem that there should be an easy way to determine this, since there is a simple way to answer the similar question of whether a graph has an Euler circuit. Surprisingly, there are no known simple necessary and sufficient criteria for the existence of Hamilton circuits. However, there are some theorems known as giving sufficient conditions for the existence of Hamilton circuits. Also some properties can be used to show that a graph has no Hamilton circuit. For instance, a graph with a vertex of degree one cannot have a Hamilton circuit, since in a Hamilton circuit, each vertex is incident with two edges in the circuit. Moreover, if a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton circuit. Also note that when a Hamilton circuit is being constructed and this circuit has passed through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed from consideration. Furthermore, a Hamilton circuit cannot contain a smaller circuit within it.

Example: Show that the following two graphs do not contain Hamilton paths?



G_1



G_2

Solution: There is no Hamilton circuit in G_1 since G_1 has a vertex of degree one, namely, e.

Now consider G_2. Since the degrees of the vertices a, b, d, e are all two, every edge incident with these vertices must be part of any Hamilton circuit. It is now easy to see that no Hamilton circuit can exist in H, for any Hamilton circuit would have to contain four edges incident with c, which is impossible. □

Example: Show that K_n has a Hamilton circuit whenever $k \geq 3$.

Solution: We can form a Hamilton circuit in K_n beginning at any vertex. Such a circuit can be built by vertices in any order we choose, as long as the path begins and ends at the same vertex and visits each other vertex exactly once. This is possible since there are edges in K_n between any two vertices. □

At the end of this section, we state two sufficient conditions for the existence of Hamilton circuits. These two conditions were found by Gabriel A. Dirac in 1952 and Oystein Ore in 1960.

Theorem 2.10 *(Dirac) If G is a simple graph with n vertices with $n \geq 3$, such that the degree of each vertex is at least $n/2$, then G has a Hamilton circuit.*

Theorem 2.11 *(Ore) If G is a simple graph with n vertices with $n \geq 3$, such that $deg(u) + deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.*

2.5 Shortest-path problems—Dijkstra's algorithm

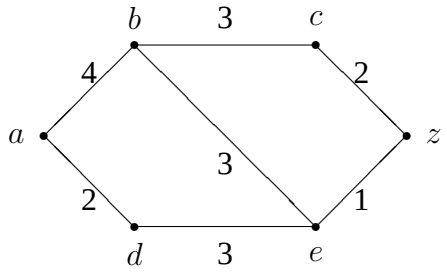
Many problems can be modeled using graphs with weights assigned to their edges. Graphs that have a number assigned to each edge are called *weighted graphs*. Weighted graphs are used to model computer networks. Communications costs (such as the monthly cost of leasing a telephone line), the response times of the computers over these lines, or the distance between computers, can all be studied using weighted graphs.

There are several different algorithms that find a shortest path between two vertices in a weighted graph. We will present an algorithm discovered by the Dutch mathematician Edsger Dijkstra in 1959. The version we will describe solves this problem in undirected weighted graphs where all the weights are positive. It is easy to adapt it to solve shortest path problems in directed graphs.

Before giving a formal presentation of the algorithm, we will give a motivating example.

Example:

What is the length of a shortest path between a and z in the weighted graph shown in the following graph?



Solution: Although a shortest path is easily found by inspection, we will develop some ideas useful in understanding Dijkstra's algorithm. We will solve this problem by fining the length of a shortest path from a to successive vertices, until z is reached.

The only paths starting at a that contain non vertex other than a (until the terminal vertex is reached) are a, b and a, d. Since the lengths of a, b and a, d are 4 and 2, respectively, it follows that d is the closest vertex to a.

We can find the next closet vertex by looking at all paths that go through only a and d (until the terminal vertex is reached). The shortest such path to b

is still a, b, with length 4, and the shortest such path to e is a, d, e, with length 5. Consequently, the next closest vertex to a is b.

To find the third closest vertex to a, we need to examine only paths that go through only a, d and b (until the terminal vertex is reached). There is a path of length 7 to c, namely, a, b, c and a path of length 6 to z, namely, a, d, e, z. Consequently, z is the next closest vertex to a, and the length of a shortest path to z is 6.

□

This example illustrates the general principles used in Dijkstra's algorithm. Note that a shortest path from a to z could have been found by inspection. However, inspection is impractical when a large number of edges are involved.

We will now consider the general problem of finding the length of a shortest path between a and z in an undirected connected simple weighted graph. Dijkstra's algorithm proceeds buy finding the length of a shortest path from a to a first vertex, the length of a shortest path from a to a second vertex, and so on, until the length of a shortest path from a to z is found.

The algorithm relies on a series of iterations. A distinguished set of vertices is constructed by adding one vertex at each iteration. A labeling procedure is carried out at each iteration. In this labeling procedure, a vertex w is labeled with the length of a shortest path from a to w that contains only vertices already in the distinguished set. The vertex added to the distinguished set is one with a minimal label among those vertices not already in the set.

We now give the details of Dijkstra's algorithm. It begins by labeling a with 0 and the other vertices with ∞. **We use the notation $L_0(a) = 0$ and $L_0(v) = \infty$ for these labels before any iterations have taken place (the subscript 0 stands for the 0-th iteration).** These labels are the lengths of shortest paths from a to the vertices, where the paths contain only the vertex a. (Since no path from a to a vertex different from a exists, ∞ is the length of a shortest path between a and this vertex.

Dijkstra's algorithm proceeds by forming a distinguished set of vertices. Let S_k denote this set after k iterations of the labeling procedure. We begin with $S_0 = \emptyset$. The set S_k is formed from S_{k-1} by adding a vertex u not in S_{k-1} with the smallest label. Once u is added to S_k, we update the labels of all vertices not in S_k, so that $L_k(v)$, the label of the vertex v at the kth stage, is the length of a shortest path from a to v that contains vertices only in S_k (that is, vertices that were already in the distinguished set together with u).

Let v be a vertex not in S_k. To update the label of v, note that $L_k(v)$ is the length of a shortest path from a to v containing only vertices in S_k. The updating can be carried out efficiently when this observation is used: a shortest path from a to v containing only elements of S_k is either a shortest path from a to v that contains only elements of S_{k-1} (that is, the distinguished vertices not including u), or it is a shortest path from a to u at the $(k-1)$-st stage with the edge (u, v) added. In other words,

$$L_k(a, v) = \min\{L_{k-1}(a, v), L_{k-1}(a, u) + w(u, v)\}.$$

This procedure is iterated by successively adding vertices to the distinguished set until z is added. When z is added to the distinguished set, its label is the length of a shortest path from a to z.

Dijkstra's algorithm

Procedure *Dijkstra* (G: weighted connected simple graph, with all weights positive)

{G has vertices $a = v_0, v_1, \cdots, v_n = z$ and weights $w(v_i, v_j)$
where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}

for $i := 1$ **to** n

$$L(v_i) := \infty$$

$L(a) := 0$

$S := \emptyset$

{ *the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set* }

while $z \notin S$

begin

$u := a$ vertex not in S with $L(u)$ minimal

$S := S \cup \{u\}$

for all vertices v not in S

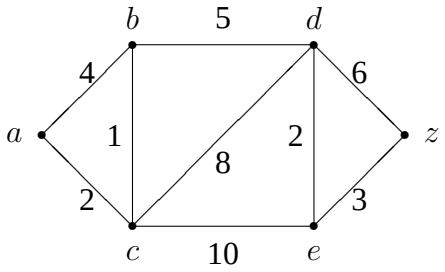
if $L(u) + w(u, v) < L(v)$ **then** $L(v) := L(u) + w(u, v)$

{ *this adds a vertex to* S *with minimal label and updates the labels of vertices not in* S }

end { $L(z) =$ length of a shortest path from a to z.

We illustrate Dijkstra's algorithm in the following example.

Example: Use Dijkstra's algorithm to find the length of a shortest path between the vertices a and z in the weighted graph:



□

Now we use tan inductive argument to show that Dijkstra's algorithm produces the length of a shortest path between two vertices a and z in an undirected connected weighted graph. Take as the induction hypothesis the following assertion: at the kth iteration

(i) the label of every vertex v in S is the length of a shortest path from a to the vertex, and

(ii) the label of every vertex not in S is the length of a shortest path from a to this vertex that contains only (besides the vertex itself) vertices in S.

When $k = 0$, before any iteration are carried out, $S = \emptyset$, so the length of a shortest path from a to a vertex other than a is ∞.

Assume that the inductive hypothesis holds for the kth iteration. Let v be the vertex added to S at the $(k + 1)$st iteration so that v is a vertex not in S at the

end of the kth iteration with the smallest label (in the case of ties, any vertex with smallest label may be used).

From the inductive hypothesis we see that the vertices in S before the $(k+1)$st iteration are labeled with the length of a shortest path from a. Also v must be labeled with the length of a shortest path to it from a. If this were not the case, at the end of the kth iteration there would be a path of length less than $L_k(v)$ containing a vertex not in S (because $L_k(v)$ is the length of a shortest path from a to v containing only vertices in S after the kth iteration). Let u be the first vertex not in S in such a a path. There is a path with length less than $L_k(v)$ from a to u containing only vertices of S. This contradicts the choice of v. Hence (i) is true at the end of the $(k+1)$st iteration.

Let u be a vertex not in S after $k+1$ iterations. A shortest path from a to u containing only elements of S either contains v or it does not. If it does not contain v, then by the inductive hypothesis, its length is $L_k(u)$. If it does contain v, then it must be made up of a path from a to v of shortest possible length containing elements of S other than v, followed by the edge from v to u. In this case, its length would be $L_k(v) + w(v,u)$. This shows that (ii) is true, since $L_{k+1}(u) = \min\{L_k(u), K_)k(v) + w(v,u)\}$.

We have proved the following theorem:

Theorem 2.12 *Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.*

The following part is optional. We can now estimate the computational complexity of Dijkstra's algorithm, in terms of addition and comparisons. The algorithm uses no more than $n-1$ iterations, since one vertex is added to the distinguished set at each iteration. We are done if we can estimate the number of operations used for each iteration. We can identify the vertex not in $S_)k$ with the smallest label using no more than $n-1$ comparisons. Then we use an addition and a comparison to update the label of each vertex not in S_k. It follows that no more than $2(n-1)$ operations are used at teach iteration, since there are no more than $n-1$ labels to update at each iteration. Since we use no more than $n-1$ iterations, each using no more than $2(n-1)$ operations we have the following theorem:

Theorem 2.13 *Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected simple undirected weighted graph with n vertices.*

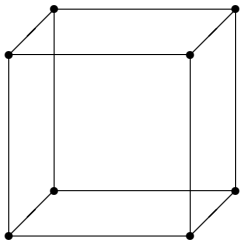
3 Planar graphs

In this lecture, we will study the question of whether a graph can be drawn in the plane without edges crossing.

Definition 3.1 *A graph is called planar if it can be drawn in the plane without edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a plannar representation of the graph.*

A graph may be planar even if it is usually drawn with crossings (e.g., K_4), since it may be possible to draw it in a different way without crossings.

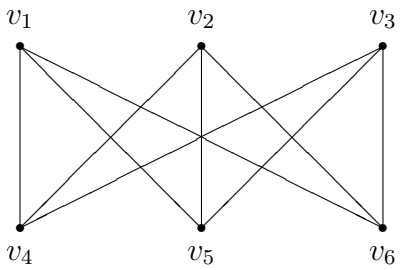
Example: The following graph is planar. Why?



□

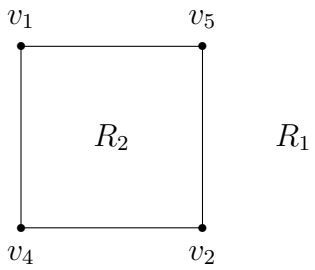
We can show that a graph is planar by displaying a planar representation. It is harder to show that a graph is nonplanar. We will give an example to show how this can be done in an ad hoc fashion.

Example: Show that $K_{3,3}$ is not planar.

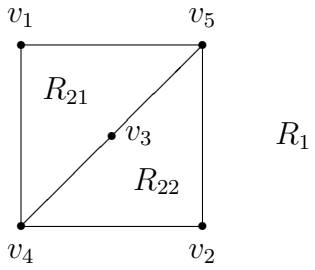


Solution: In any planar representation of $K_{3,3}$, the vertices v_1 and v_2 must be connected to both v_4 and v_5. These four edges form a closed curve that splits the

plane into two regions, R_1 and R_2. The vertex v_3 is either in R_1 or in R_2. See below.



If v_3 is in R_2, the inside of the closed curve, the edges between v_3 and v_4 and between v_3 and v_5 separate R_2 into two subregions, R_{21} and R_{22}.



Next, note that there is no way to place the final vertex v_6 without forcing a crossing. For if v_6 is in R_1, then the edge between v_6 and v_3 cannot be drawn without a crossing. If v_6 is in R_{21}, then the edge between v_6 and v_2 cannot be drawn without a crossing. If v_6 is in R_{22}, then the edge between v_6 and v_1 cannot be drawn without a crossing.

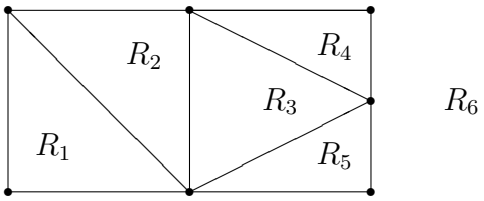
A similar argument can be used when v_3 is in R_1. We leave the verification of this case to you. Thus, $K_{3,3}$ cannot be represented on a plane. □

Planarity of graphs plays an important role in the design of electronic circuits. We can model a circuit with a graph by representing components of the circuit by vertices and connections between them by edges. We can print a circuit on a single board with no connections crossing if the graph representing the circuit is planar. When this graph is not planar, we must turn to more expensive options. For example, we can partition the vertices in the graph representing the circuit into planar subgraphs. We then construct the circuit using multiple layers. We

can construct the circuit using insulated wires whenever connections cross. In this case, drawing the graph with the fewest possible crossings is important.

3.1 Euler's formula

A planar representation of a graph splits the plane into regions, including an unbounded region. For instance, the planar representation of the graph shown in the following figure splits the plane into six regions. These are labelled in the figure.



Euler showed that all planar representations of a graph split the plane into the same number of regions. He accomplished this by finding a relationship among the number of regions, the number of vertices, and the number of edges of a planar graph.

Theorem 3.2 *(Euler's Formula) Let G be a connected planar simple with e edges and v vertices. Let r be the number of regions in a planar representation of G. Then*

$$r = e - v + 2.$$

Proof: First, we specify a planar representation of G. We will prove the theorem by constructing a sequence of subgraphs $G_1, G_2, \cdots, G_e = G$ successively by adding an edge at each stage. This is done using the following inductive definition.

- Arbitrarily pick one edge of G to obtain G_1.
- Having G_{n-1}, we construct G_n by arbitrarily adding an edge that is incident with a vertex already in G_{n-1}, adding the other vertex incident with this edge if it is not in G_{n-1}.

This construction is possible since G is connected. G is obtained after e edges are added. Let r_n, e_n and v_n represent the number of regions, edges, and vertices

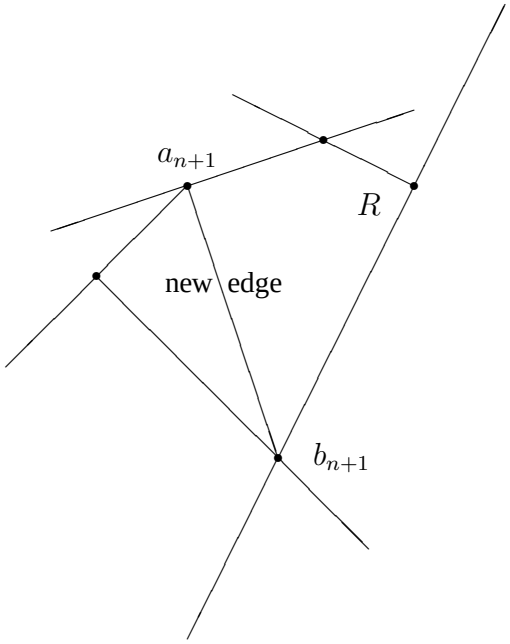
of the planar representation of G_n induced by the planar representation of G, respectively.

The proof will now proceed by induction.

The relationship $r_1 = e_1 - r_1 + 2$ is true for G_1 since $e_1 = 1, v_1 = 2$ and $r_1 = 1$.

Now assume that $r_n = e_n - v_n + 2$. Let $\{a_{n+1}, b_{n+1}\}$ be the edge that is added to G_n to obtain G_{n+1}. There are two possibilities to consider.

In the first case, both a_{n+1} and b_{n+1} are already in G_n. Then these two vertices must be on the boundary of a common region R, or else it would be impossible to add the edge $\{a_{n+1}, b_{n+1}\}$ to G_n without two edges crossing (and G_{n+1} is planar). The addition of this new edge splits R into two regions.

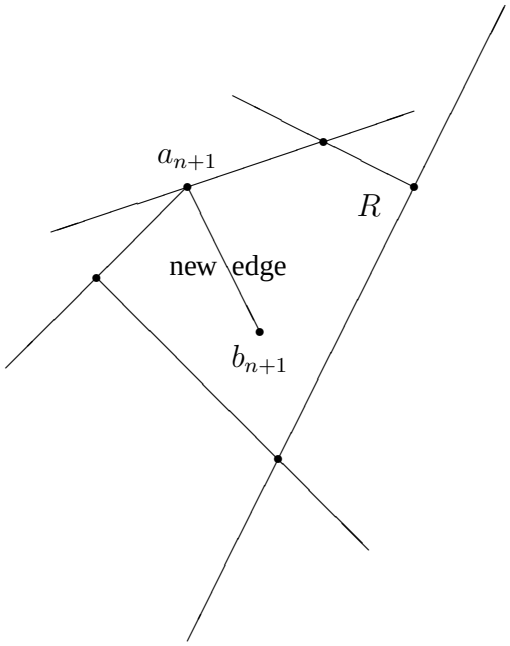


As a consequence, in this case,

- $r_{n+1} = r_n + 1$,
- $e_{n+1} = e_n + 1$, and
- $v_{n+1} = v_n$.

Therefore, $r_{n+1} = e_{n+1} - v_{n+1} + 2$.

In the second case, one of the two vertices of the new edge is not already in G_n. Suppose that a_{n+1} is in G_n but that b_{n+1} is not.



Adding this new edge does not produce any new regions, since b_{n+1} must be in a region that have a_{n+1} on its boundary. Consequently, $r_{n+1} = r_n$. Moreover, $e_{n+1} = e_n + 1$ and $v_{n+1} = v_n + 1$. Each side of the formula relating the number of regions, edges, and vertices remains the same, so the formula is still true. In other words, $r_{n+1} = e_{n+1} - v_{n+1} + 2$.

We have completed the induction argument. Hence $r_n = e_n - v_n + 2$ is true for all n. Since the original graph is the graph G_e, obtained after e edges have been added, the theorem is true. □

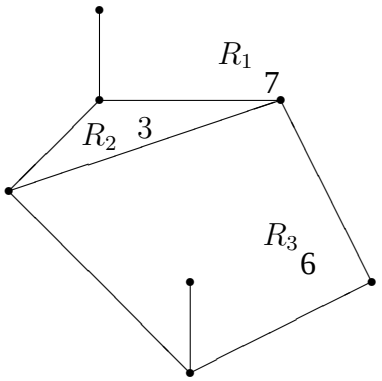
Euler's formula is illustrated in the following example.

Example: Suppose that a connected planar simple graph has 20 vertices, each of degree 3. Into how many regions does a representation of this planar graph split the plane? □

Euler's formula can be used to establish some inequalities that must be satisfied by planar graphs. One such inequality is given in the following corollary.

Corollary 3.3 *If G is connected planar simple graph with e edges and v vertices where $v \geq 3$, then $e \leq 3v - 6$.*

Proof: First we define the **degree** of a region as the number of edges on the boundary of this region. When an edge occurs twice on the boundary (so that it is traced out twice when the boundary is traced out), it contributes two to the degree.



Suppose a connected planar simple graph drawn in the plane divides the plane into r many regions. Then the degree of each region is at least three. (Since the graphs discussed here are simple graphs, no multi edges that could produce regions of degree two, or loops that could produces of degree one, are permitted.) In particular, note that the degree of the unbounded region is at least three since there are at least three vertices in the graph.

Note that the sum of the degrees of the regions is exactly twice the number of edges in the graph, because each edge occurs on the boundary of a region exactly twice (either in two different regions, or twice in the same region). Since each region has degree greater than or equal to three, it follows that

$$2e = \sum_{\text{all regions } R} deg(R) \geq 3r.$$

Hence $r \leq (2e)/3$. Using $r = e - r + 2$, we have

$$e - v + 2 \leq (2/3)e.$$

Immediately, we have $e \leq 3v - 6$. □

Corollary 3.4 *If G is a connected planar simple graph, then G has a vertex of degree not exceeding five.*

Proof: If G has one or two vertices, the result is true.

If G has at least three vertices, by 3.3, $e \leq 3v - 6$ and hence $2e \leq 6v - 12$. If the degree of every vertex were at least six, then since $2e = \sum_{v \in V} deg(v)$, we would have $2e \geq 6v$. But this contradicts the inequality $2e \leq 6e - 12$. If follows that there must be a vertex with degree no bigger than 5. □

From Corollary 3.5, we know that K_5 is not planar because the inequality $e \leq 3v - 6$ is not satisfied ($e = 10, v = 5, 3v - 6 = 9$.

We proved before that $K_{3,3}$ is not planar. Note however that this graph has 6 vertices and 9 edges, and the inequality $e = 9 < 12 = 3 \cdot 6 - 6$ is satisfied. Hence, we cannot use the inequality to prove that $K_{3,3}$ is not planar. However, we can have the following improvement.

Corollary 3.5 *If G is a connected planar simple graph with e edges and v vertices, $v \geq 3$, and no circuit in G can have length 3, then $e \leq 2v - 4$.*

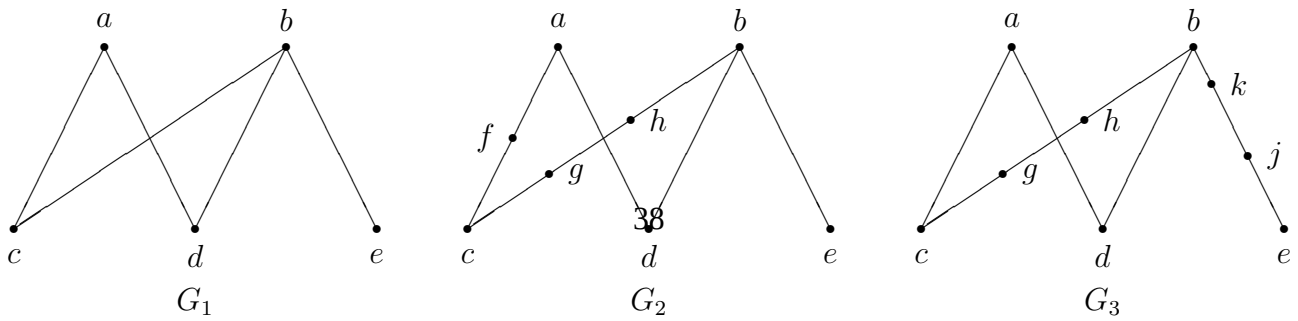
We leave the proof to you.

Since $K_{3,3}$ has no circuits of length 3, we can apply Corollary 3.5. Note that $K_{3,3}$ has 6 vertices and 9 edges, $e = 9 > 8 = 2v - 4$. By Corollary 3.5, $K_{3,3}$ is not planar.

3.2 Kuratowski's Theorem

We can see that $K_{3,3}$ and K_5 are not planar. Clearly, a graph is not planar if it contains either of these two graphs as a subgraph. Surprisingly, all nonplanar graphs must contain a subgraph that can be obtained from $K_{3,3}$ or K_5 using some permitted operations.

If a graph is planar, so will be any graph obtained by removing an edge $\{u, v\}$ and adding a new vertex w together with edges $\{u, w\}$ and $\{w, v\}$. Such an operation is called an *elementary subdivision*. The graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called *homeomorphic* if they can be obtained from the same graph by a sequence of elementary subdivisions.



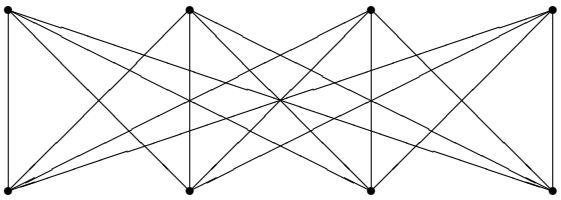
The three graphs displayed above are homeomorphic since all can be obtained from the first one by elementary subdivisions. (Can you figure out these elementary subdivisions?)

The following characterization of planar graphs using the concept of graph homeomorphism is due to Kuratowski, a mathematician form Poland.

Theorem 3.6 *A graph is nonplanar if and only if it contains a subgraph homeomorphic to $K_{3,3}$ or K_5.*

It is clear that a graph containing a subgraph homeomorphic to $K_{3,3}$ or K_5 is nonplanar. However, the proof of the converse, namely that every nonplanar graph contains a subgraph homeomorphic to $K_{3,3}$ or K_5, is complicated, and will not be given here.

Example: Determine whether the graph given below is planar.



□

The *crossing number* of a simple graph is the minimum number of crossings that can occur when this graph is drawn in the plane where no three arcs representing edges are permitted to cross at the same point. For example, $K_{3,3}$ has 1 as its crossing number.

Example: Find the crossing numbers of each of the following nonplanar graphs.

(a) K_5, (b) K_6, (c) K_7, (d) $K_{3,4}$, (e) $K_{4,4}$, (f) $K_{5,5}$ □

4 Graph coloring

Problems related to the coloring of maps of regions, such as maps of parts of the world, have generated many results in graph theory. When a map is colored, two regions with a common border are customarily assigned different colors. One way to ensure that two adjacent regions never have the same color is to use a different color for each region. However, this is inefficient, and on maps with many regions it would be hard to distinguish similar colors. Instead, a small number of colors should be used whenever possible. Consider the problem of determining the least number of colors that can be used to color a may so that adjacent regions never have the same color.

Each map in the plane can be represented by a graph. To set up this correspondence, each region of the map is represented by a vertex. Edges connect two vertices if the regions represented by these vertices have a common border. Two regions that touch at only one point are not considered adjacent. The resulting graph is called the *dual graph* of the map. By the way in which dual graphs of maps are constructed, it is clear that any map in the plane has a planar dual graph.

The problem of coloring the regions of a map is equivalent to the problem of coloring the vertices of the dual graph so that no two adjacent vertices in this graph have the same color. We now define a graph coloring.

Definition 4.1 *A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.*

A graph can be colored by assigning a different color to each of its vertices. However, for most graphs, a coloring can be found that uses fewer colors than the number of vertices in the graph. What is the least number of colors necessary?

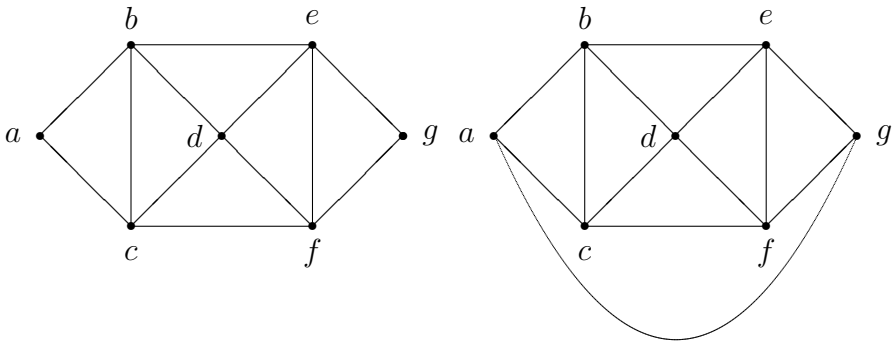
Definition 4.2 *The chromatic number of a graph is the least number of colors needed for a coloring of this graph.*

Note that asking for the chromatic number of a planar graph is the same as asking for the minimum number of colors required to color a planar map so that no two adjacent regions are assigned the same color. This question has been studied for more than 100 years, and was finally solved by the American mathematicians Kenneth Appel and Wolfgang Haken in 1976.

Theorem 4.3 *The chromatic number of a planar graph is no greater than 4.*

Example: What are the chromatic numbers of the following graphs: K_n, $K_{m,n}$, C_n? □

Example: What are the chromatic numbers of the following graphs?



□

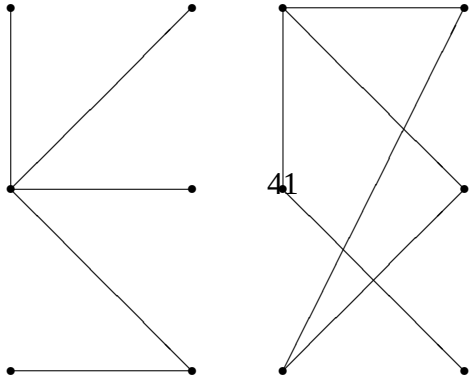
5 Trees

We have seen how graphs can be used to model and solve many problems. In this lecture, we will focus on a particular type of graph called a *tree*, so named because such graphs resemble trees.

Definition 5.1 *A tree is a connected undirected graph with no simple circuits.*

Since a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops. Therefore, any tree must be a simple graph.

Example: Which of the following are trees?



□

Any connected undirected graph with no simple circuits is a tree. What about graphs containing no simple circuits that are not necessarily connected? These graphs are called *forests* and have the property that each of their connected components is a tree.

Trees are often defined as undirected graphs with the property that there is a unique simple path between every pair of vertices. The following theorem shows that this alternative definition is equivalent to our definition.

Theorem 5.2 *An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.*

Proof: First assume that T is a tree. Then T is a connected graph with no simple circuits. Let x and y be two vertices of T. Since T is connected, there is a simple path between x and y (remember that we proved this before). Moreover, this path must be unique, for if there were a second such path, the path formed by combining the first path from x to y followed by the path from y to x obtained by reversing the order of the second path from x to y would form a circuit. This implies that there is a simple circuit in T. Hence, there is a unique simple path between any two vertices of a tree.

Now assume that there is a unique simple path between any two vertices of a graph T. Then T is connected since there is a path between any two of its vertices. Furthermore, T can have no simple circuits. To see this, suppose that T had a simple circuit that contained the vertices x and y. Then there would be two simple paths between x and y, since the simple circuit is made up of a simple path from x to y and a second simple path from y to x. Hence, a graph with a unique simple path between any two vertices is a tree. □

Theorem 5.3 *Any tree that has more than one vertex has at least one vertex of degree 1.*

A constructive way to prove this lemma is to imagine being given a tree T with more than one vertex. You pick a vertex v arbitrarily and then search outward along a path from v looking for a vertex of degree 1. As you reach each new vertex, you check whether it has degree 1. If it does, you are finished. If it does not, you exist from the vertex along a different edge from the one you entered on. Because T does not contain a simple circuit, the vertices included in the path never repeat. And since the number of vertices of T is finite, the process of building a

path must eventually terminate. When that happens, the final vertex v' of the path must have degree 1.

In many applications of trees, a particular vertex of a tree is designated as the *root*. Once we specify a root, we can assign a direction to each edge as follows. Since there is a unique path from the root to each vertex of the graph (by the previous theorem), we direct each edge away from the root. Thus a tree together with its root produces a directed graph called a *rooted tree*.

Definition 5.4 *A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.*

The terminology for tees has botanical and genealogical origins. Suppose that T is a rooted tree. If v is a vertex in T other than the root, the *parent* of v is the unique vertex u such that there is a directed edge from u to v (can you prove that such a vertex is unique?). When u is the parent of v, v is called a *child* of u. Vertices with the same parent are called *siblings*. The *ancestors* of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root (that is, its parent, its parent's parent, and so on, until the root is reached). The *descendants* of a vertex v are those vertices that have v as an ancestor.A vertex of a tree is called a *leaf* if it has no children. Vertices that have children are called *internal vertices*. The root is an internal vertex unless it is the only vertex in the graph, in which case it is a leaf. The *level* of a vertex is the number of edges along the unique path between it and the root. The *height* of a rooted tree is the maximum level to any vertex of the tree.

If a is a vertex in a tree, the *subtree* with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Theorem 5.5 *A tree with n vertices has $n - 1$ edges.*

Proof: We will use induction to prove this theorem. Note that for all the trees here we can choose a root and consider the tree rooted.

Basis step: When $n = 1$, a tree with $n = 1$ vertex has no edges. It follows that the theorem is true for $n = 1$.

Inductive step: The induction hypothesis states that every tree with k vertices has $k - 1$ edges, where k is a positive integer. Suppose that a tree T has $k + 1$ vertices and that v is a leaf of T (which must exist since the tree is finite), and let w be the parent of v. Removing from T the vertex v and the edge connecting w to v

produces a tree T' with k vertices, since the resulting graph is still connected and has no simple circuits. By the induction hypothesis, T' has $k-1$ edges. It follows that T has k-edges since it has one more edge than T', the edge connecting v and w. This completes induction step. □

Example: A graph G has ten vertices and twelve edges. Is it a tree? □

Example: Find all nonisomorphic trees with 4 vertices. □

Theorem 5.6 *If G is any connected graph, C is any nontrivial circuit in G, and one of the edges of C is removed from G, then the graph that remains is connected.*

Essentially, the main idea behind this theorem is that any two vertices in a circuit are connected by two distinct paths. It is possible to draw the graph so that one of these goes "clockwise" and the other goes "counterclockwise" around the circuit.

Suppose that an edge e is on this circuit, and a path from x to y contains e. Let v_1, v_2 be the endpoints of e. Also suppose that e is removed. It's easy to see that x and y are still connected. Why?

The following theorem follows immediately.

Theorem 5.7 *If G is any connected graph with n vertices and $n-1$ edges, where n is any positive integer, then G is a tree.*

Rooted trees with the property that all of their internal vertices have the same number of children are used in many different applications, such as searching, sorting and coding. When every vertex in a rooted tree has at most two children and each child is designated either the (unique) left child or the (unique) right child, the result is a *binary* tree.

Definition 5.8 *A binary tree is a rooted tree in which every parent has at most two children. Each child in a binary tree is designated either the (unique) left child or the (unique) right child (but not both), and every parent has at most one left child and one right child.*

A full binary tree is a binary tree in which each parent has exactly two children.

Given any parent v in a binary tree, the left subtree of v is the binary tree whose root is the left child of v, whose vertices consist of the left child of v and all its descendants, and whose edges consist of all those edges of T that connect the vertices of the left subtree.

The right subtree of v is defined similarly.

Example: Suppose that T is a full binary tree with k many internal vertices where k is a positive integer. Prove that T has a total of $2k + 1$ many vertices and has $k + 1$ terminal vertices.

The proof is based on the following facts:

- The set of all vertices of T can be partitioned into two disjoint subsets: the set of all vertices that have a parent and the set of all vertices that do not have a parent.
- There is only one vertex that does not have a parent, namely the root.
- In a full binary tree, every internal vertex has exactly two children. As a consequence, the number of vertices that have a parent is twice the number of parents.

□

Example: If T is a binary tree that has t terminal vertices and height h, then $t \leq 2^h$. □

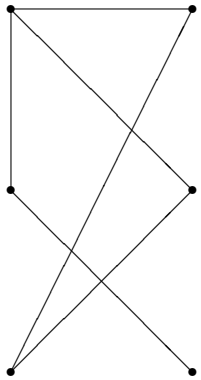
6 Spanning trees

Definition 6.1 *A spanning tree for a graph G is a subgraph of G that contains every vertex of G and is a tree.*

Theorem 6.2 *Every connected graph has a spanning tree.*

Note that any two spanning trees for a graph have the same number of edges.

Example: Find all spanning trees for the following graph.



G

Solution: The graph has one circuit of length 4 and removal of any edge of the circuit gives a tree. Thus, G has four spanning trees. □

Definition 6.3 *Let G be a weighted graph. We define the total weight of G, denoted as $w(G)$, as the sum of the weights of all the edges in G.*

A minimum spanning tree for a weighted graph is a spanning tree that has the least possible total weight compared to all other spanning trees for the graph.

The problem of finding a minimum spanning tree for a graph is certainly solvable. One solution is to list all spanning trees of the graph, compute the total weight of each, and choose one for which this total is a minimum. This solution, however, is inefficient in this use of computing time because the number of distinct spanning trees is so large. For instance, a complete graph with n vertices has n^{n-2} spanning trees.

Now we introduce two much more efficient algorithms: Kruskal's algorithm and Prim's algorithm.

Kruskal's algorithm

The algorithm we describe here was discovered by Kruskal in 1956. In this algorithm, the edges of a weighted graph are examined one by one in order of increasing weight. At each stage the edge being examine is added to what will become the minimum spanning tree, provided that this addition does not create a circuit. After $n-1$ edges have been added, where n is the number of vertices of the graph,

these edges, together with the vertices of the graph, for a minimum spanning tree for the graph.

Pseudo code for Kruskal's algorithm is given below.

procedure $Kruskal$ (G : weighted connected undirected graph with n vertices

T := empty graph

for $i = 0$ **to** $n - 1$

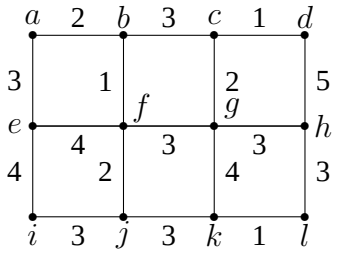
begin

e := any edge in G with smallest weight that does not form a simple circuit when added to T.

T := T with e added.

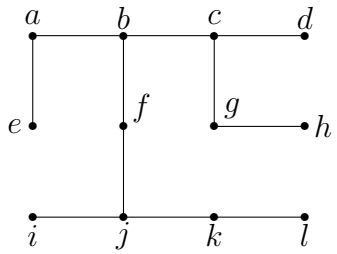
end {T is a minimum spanning tree of G}

Example: Apply Kruskal's algorithm to find a minimum spanning tree of the following graph.



Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{j, f\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{h, g\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
		24
		↑
		total

So a spanning tree is



□

Prim's algorithm

Prim's algorithm works differently from Kruskal's. It builds a minimum spanning tree T by expanding outward in connected links from some vertex. One edge and one vertex are added at each stage. The edge added is the one of least weight that connects the vertices already in T with those not in T, and the vertex is the endpoint of this edge that is not already in T.

Pseudo code for Prim's algorithm is given below.

procedure $Prim$ (G : weighted connected undirected graph with n vertices

T := a minimum weighted edge

for $i = 0$ **to** $n - 2$

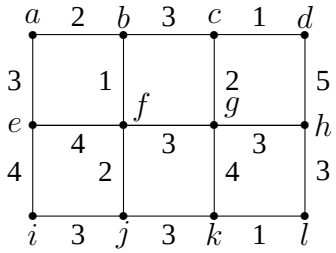
begin

e := any edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T.

T := T with e added.

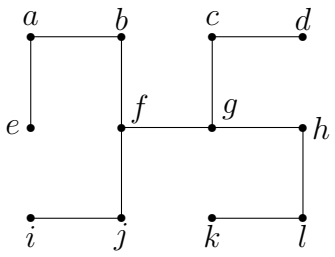
end {T is a minimum spanning tree of G}

Example: Apply Prim's algorithm to find a minimum spanning tree of the following graph.



Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{j, f\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{g, c\}$	2
8	$\{d, c\}$	1
9	$\{h, g\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
		24
		↑
		total

So a spanning tree obtaining by using Prim's algorithm is



□

We can actually prove that Prim's algorithm produces a minimum spanning tree. Let G be a connected weighted graph.

suppose that the successive edges chosen y Prim's algorithm are $e_1, e_2, \cdots, e_{n-1}$. Let S be the tree with $e_1, e_2, \cdots, e_{n-1}$ as its edges, and let S_k be the tree with $e_1, e_2, \cdots, e_k$ as its edges. Let T be a minimum spanning tree of G containing the edges $e_1, e_2, \cdots, e_k$, where k is the maximum integer with the property that a minimum spanning tree exists containing the first k edges chosen by Prim's algorithm. We need to prove that $S = T$ and we prove it by contradiction.

Suppose $S \neq T$, so that $k < n - 1$. Consequently, T contains $e_1, e_2, \cdots, e_k$ but not e_{k+1}. Consider the graph made up of T together with e_{k+1}. Since this graph is connected and has n edges, too many edges to be a tree, it must contain a simple circuit. This simple circuit must contain e_{k+1} since there was no simple circuit in T. Furthermore, there must be an edge in the simple circuit that does not belong to S_{k+1} since S_{k+1} is a tree. By starting at an endpoint of e_{k+1} that is also an endpoint of one of the edges $e_1, e_2, \cdots, e_k$, and following the circuit until it reaches an edge not in S_{k+1}, we can find an edge e not in S_{k+1} that has an endpoint that is also an endpoint of one of the edges $e_1, e_2, \cdots, e_k$. By deleting e from T and adding e_{k+1}, we obtain a tree T' with $n - 1$ edges (it is a tree since it has no simple circuits). Note that the tree T' contains $e_1, e_2, \cdots, e_k, e_{k+1}$. Furthermore, since e_{k+1} was chosen by Prim's algorithm at the $(k + 1)$-st step, and e was also available at that step, the weight of e_{k+1} is less than or equal to the weight of e. From this observation, it follows that T' is also a minimum spanning tree, since the sum of the weights of its edges does not exceed the sum of the weights of the edges of T. This contradicts the choice of k as the maximum integer so that a minimum spanning tree exists containing $e_1, e_2, \cdots, e_k$. Hence $k = n - 1$ and $S = T$. It follows that Prim's algorithm produces a minimum spanning tree.