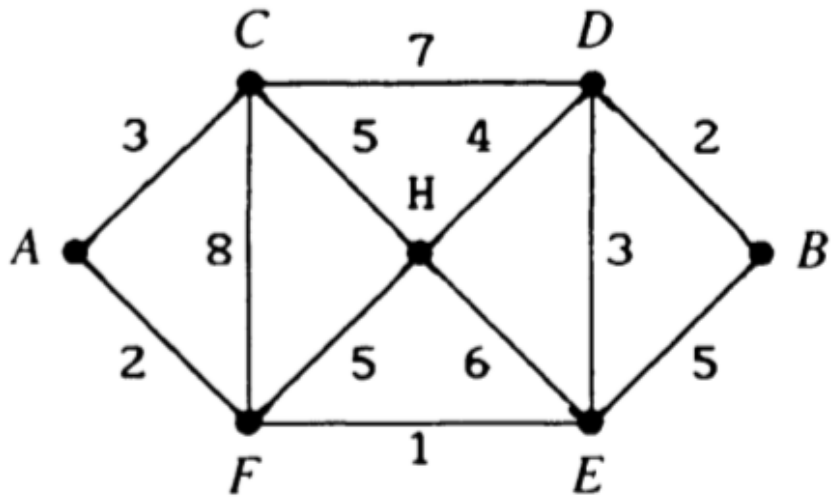
MATH 322 – Graph Theory
Fall Term 2021

Notes for Lecture 15

Thursday, October 28

Reminder: The Shortest Path Problem

In this problem, we are given a weighted connected graph G_0 of order n (here $n = 7$), and we are asked to find a minimum weight path connecting two specific vertices that we are interested in, say, vertices A and B below.



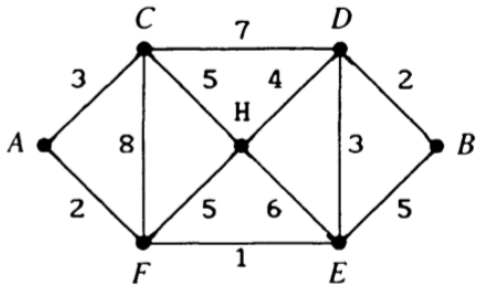
Weight matrix of a weighted graph

In the process of solving the shortest path problem, we need to consider the *weight matrix* W_{G_0} of G_0.

Assume that the vertex set of G_0 is $\{v_1, v_2, \ldots, v_n\}$; then $W_{G_0} = \left(w_{i,j}\right)_{i,j}$ is an $n \times n$ matrix satisfying

$$w_{i,j} = \begin{cases} \infty & \text{if } i = j, \text{ or the vertices } v_i \text{ and } v_j \text{ are non-adjacent} \\ \text{weight of the edge } \{v_i,\ v_j\} & \text{if } v_i \text{ and } v_j \text{ are adjacent vertices} \end{cases}$$

For the example above, we have



$$W_{G_0} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ H \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & H \end{array} \\ \begin{pmatrix} \infty & \infty & 3 & \infty & \infty & 2 & \infty \\ \infty & \infty & \infty & 2 & 5 & \infty & \infty \\ 3 & \infty & \infty & 7 & \infty & 8 & 5 \\ \infty & 2 & 7 & \infty & 3 & \infty & 4 \\ \infty & 5 & \infty & 3 & \infty & 1 & 6 \\ 2 & \infty & 8 & \infty & 1 & \infty & 5 \\ \infty & \infty & 5 & 4 & 6 & 5 & \infty \end{pmatrix} \end{array}.$$

An algorithm that returns the 'shortest possible distance' a path connecting any two fixed vertices (say, vertices A and B here) covers:

Dijkstra's algorithm

— The algorithm proceeds as follows: at each stage it assigns weights to the vertices (which are based in some sense on the weights of the edges).

— Some of these weights are called temporary *(in a sense, we're still testing out what the value for those vertices should be)*, while the rest have become permanent at some point *(and the algorithm cannot alter the latter anymore)*.

An algorithm that returns the 'shortest possible distance' a path connecting any two fixed vertices (say, vertices A and B here) covers:

Dijkstra's algorithm

— The algorithm proceeds as follows: at each stage it assigns weights to the vertices (which are based in some sense on the weights of the edges).

— Some of these weights are called temporary *(in a sense, we're still testing out what the value for those vertices should be)*, while the rest have become permanent at some point *(and the algorithm cannot alter the latter anymore)*.

— **At each stage a new vertex is allotted a permanent weight,** with vertex A being the first one to be allotted a permanent weight.

An algorithm that returns the 'shortest possible distance' a path connecting any two fixed vertices (say, vertices A and B here) covers:

Dijkstra's algorithm

— The algorithm proceeds as follows: at each stage it assigns weights to the vertices (which are based in some sense on the weights of the edges).

— Some of these weights are called temporary *(in a sense, we're still testing out what the value for those vertices should be)*, while the rest have become permanent at some point *(and the algorithm cannot alter the latter anymore)*.

— **At each stage a new vertex is allotted a permanent weight,** with vertex A being the first one to be allotted a permanent weight.

— The algorithm can be terminated as soon as vertex B (the vertex that we want our path to end at) is allotted a permanent weight.

Also, the permanent weight allotted to B is precisely the shortest distance covered by a path from A to B. *(more about this shortly)*

Dijkstra's algorithm

In more detail now,

- at the first stage, we assign permanent weight 0 to vertex A and temporary weight ∞ to every other vertex of G_0.

Dijkstra's algorithm

In more detail now,

- at the first stage, we assign permanent weight 0 to vertex A and temporary weight ∞ to every other vertex of G_0.
- At Stage r, let $v_{j_1}, v_{j_2}, \ldots, v_{j_{n-r+1}}$ be the vertices which still have temporary weight, while $v_{i_1}, v_{i_2}, \ldots, v_{i_{r-1}}$ are the vertices of G_0 which have already been allotted a permanent weight.

Dijkstra's algorithm

In more detail now,

- at the first stage, we assign permanent weight 0 to vertex A and temporary weight ∞ to every other vertex of G_0.
- At Stage r, let $v_{j_1}, v_{j_2}, \ldots, v_{j_{n-r+1}}$ be the vertices which still have temporary weight, while $v_{i_1}, v_{i_2}, \ldots, v_{i_{r-1}}$ are the vertices of G_0 which have already been allotted a permanent weight.
  - For each j_s, $1 \leqslant s \leqslant n-r+1$, and each i_t, $1 \leqslant t \leqslant r-1$, set

    $$a_{j_s,i_t} = \min\{\text{current weight of } v_{j_s},\ (\text{permanent weight of } v_{i_t}) + w_{j_s,i_t}\}.$$

    Furthermore, set $w_{j_s} = \min\{a_{j_s,i_t} : 1 \leqslant t \leqslant r-1\}$. This is the new temporary weight of the (arbitrary) vertex $v_{j_s} \in \{v_{j_1}, v_{j_2}, \ldots, v_{j_{n-r+1}}\}$.

Dijkstra's algorithm

In more detail now,

- at the first stage, we assign permanent weight 0 to vertex A and temporary weight ∞ to every other vertex of G_0.
- At Stage r, let $v_{j_1}, v_{j_2}, \ldots, v_{j_{n-r+1}}$ be the vertices which still have temporary weight, while $v_{i_1}, v_{i_2}, \ldots, v_{i_{r-1}}$ are the vertices of G_0 which have already been allotted a permanent weight.
  - For each j_s, $1 \leqslant s \leqslant n-r+1$, and each i_t, $1 \leqslant t \leqslant r-1$, set

    $$a_{j_s,i_t} = \min\{\text{current weight of } v_{j_s},\ (\text{permanent weight of } v_{i_t}) + w_{j_s,i_t}\}.$$

    Furthermore, set $w_{j_s} = \min\{a_{j_s,i_t} : 1 \leqslant t \leqslant r-1\}$. This is the new temporary weight of the (arbitrary) vertex $v_{j_s} \in \{v_{j_1}, v_{j_2}, \ldots, v_{j_{n-r+1}}\}$.
  - Find $s_0 \in \{1, 2, \ldots, n-r+1\}$ such that

    $$w_{j_{s_0}} = \min\{w_{j_s} : 1 \leqslant s \leqslant n-r+1\}$$

    (there might be two or more indices that work here, that is, the vertex that has minimum temporary weight at this point might not be unique; in such a case, just pick one index that works).

    Then vertex $v_{j_{s_0}}$ is the vertex that is allotted permanent weight at Stage r (with this weight being $w_{j_{s_0}}$).

Applying Dijkstra's algorithm to examples

From the Balakrishnan-Ranganathan book:

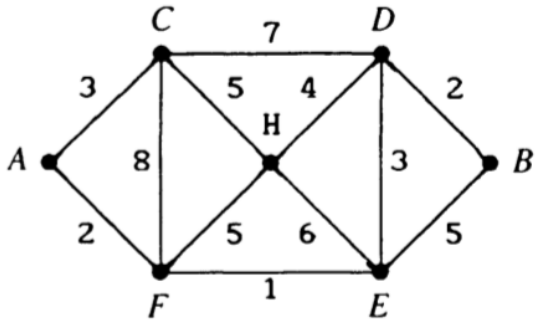


TABLE 10.5. *Steps of algorithm for shortest path from A to B*

	A	*B*	*C*	*D*	*E*	*F*	*H*
Iteration 0	0	∞	∞	∞	∞	∞	∞
Iteration 1	0	∞	3	∞	∞	2	∞
Iteration 2	0	∞	3	∞	3	2	7
Iteration 3	0	∞	3	10	3	2	7
Iteration 4	0	8	3	6	3	2	7
Iteration 5	0	8	3	6	3	2	7
Iteration 6	0	8	3	6	3	2	7

Understanding the output of the algorithm

- Suppose that v_0 is the vertex we start with, namely the vertex which is assigned permanent weight 0 (in the previous example, this is vertex A).

  For each vertex v which is allotted a permanent weight before the algorithm is terminated, **this permanent weight is precisely the shortest distance that can be covered by a path from the initial vertex v_0 to v.**

Understanding the output of the algorithm

- Suppose that v_0 is the vertex we start with, namely the vertex which is assigned permanent weight 0 (in the previous example, this is vertex A).

  For each vertex v which is allotted a permanent weight before the algorithm is terminated, **this permanent weight is precisely the shortest distance that can be covered by a path from the initial vertex v_0 to v.**

- In the case of the previous example, in which we were 'unlucky' (or 'lucky' from another point of view) and ended up needing to find permanent weights for all the vertices of the graph, we now know the shortest distance between vertex A and any other vertex of the graph! *For instance,*

  - *the shortest possible distance covered by a path from A to D is 6,*

Understanding the output of the algorithm

- Suppose that v_0 is the vertex we start with, namely the vertex which is assigned permanent weight 0 (in the previous example, this is vertex A).

  For each vertex v which is allotted a permanent weight before the algorithm is terminated, **this permanent weight is precisely the shortest distance that can be covered by a path from the initial vertex v_0 to v.**

- In the case of the previous example, in which we were 'unlucky' (or 'lucky' from another point of view) and ended up needing to find permanent weights for all the vertices of the graph, we now know the shortest distance between vertex A and any other vertex of the graph! *For instance,*
  - *the shortest possible distance covered by a path from A to D is 6,*
  - *while the shortest possible distance covered by a path from A to H is 7.*

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

  — We start moving 'backwards': 'standing at' vertex B, we look at its neighbours, which are the only vertices that could come right after B in the path *(or right before B, depending on which direction we traverse the path in)*.

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

  — We start moving 'backwards': 'standing at' vertex B, we look at its neighbours, which are the only vertices that could come right after B in the path *(or right before B, depending on which direction we traverse the path in)*.

Crucial Observation here

No matter what graph we are working with, at least some of the neighbours of vertex B will have been allotted a permanent weight **before B is allotted one**, and thus before the algorithm is terminated *(try to explain why this is so)*.

Let's say $N_1, N_2, \ldots, N_s$ are the neighbours of B that have been allotted a permanent weight, and suppose that $w_B, w_{N_1}, w_{N_2}, \ldots, w_{N_s}$ are these permanent weights.

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

  — We start moving 'backwards': 'standing at' vertex B, we look at its neighbours, which are the only vertices that could come right after B in the path *(or right before B, depending on which direction we traverse the path in)*.

Crucial Observation here

No matter what graph we are working with, at least some of the neighbours of vertex B will have been allotted a permanent weight **before B is allotted one**, and thus before the algorithm is terminated *(try to explain why this is so)*.

Let's say $N_1, N_2, \ldots, N_s$ are the neighbours of B that have been allotted a permanent weight, and suppose that $w_B, w_{N_1}, w_{N_2}, \ldots, w_{N_s}$ are these permanent weights.

Note that, for each $i \in \{1, 2, \ldots, s\}$, we will have

$$w_B - w_{N_i} \leqslant w_{B,N_i} = \text{weight of the edge } \{B, N_i\}.$$

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

  — We start moving 'backwards': 'standing at' vertex B, we look at its neighbours, which are the only vertices that could come right after B in the path *(or right before B, depending on which direction we traverse the path in)*.

Crucial Observation here

No matter what graph we are working with, at least some of the neighbours of vertex B will have been allotted a permanent weight **before B is allotted one**, and thus before the algorithm is terminated *(try to explain why this is so)*.

Let's say $N_1, N_2, \ldots, N_s$ are the neighbours of B that have been allotted a permanent weight, and suppose that $w_B, w_{N_1}, w_{N_2}, \ldots, w_{N_s}$ are these permanent weights.

Note that, for each $i \in \{1, 2, \ldots, s\}$, we will have

$$w_B - w_{N_i} \leqslant w_{B,N_i} = \text{weight of the edge } \{B, N_i\}.$$

If for some i_0 we have $w_B - w_{N_{i_0}} = w_{B,N_{i_0}}$, then the path of shortest total distance that we are trying to find can be chosen to move from B to N_{i_0}.

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

— 'Standing at' vertex N_{i_0}, we repeat the above process for N_{i_0} now: we look at all neighbours of N_{i_0}, except for vertex B, which have been allotted a permanent weight; let's write $M_1, M_2, \ldots, M_t$ for these neighbours of N_{i_0}.

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

— 'Standing at' vertex N_{i_0}, we repeat the above process for N_{i_0} now: we look at all neighbours of N_{i_0}, except for vertex B, which have been allotted a permanent weight; let's write $M_1, M_2, \ldots, M_t$ for these neighbours of N_{i_0}.

For any $j_0 \in \{1, 2, \ldots, t\}$ for which we have

$$w_{N_{i_0}} - w_{M_{j_0}} = w_{N_{i_0}, M_{j_0}} = \text{weight of the edge } \{N_{i_0}, M_{j_0}\},$$

we can choose the path of shortest total distance
that we are trying to find to move next to vertex M_{j_0}.

Understanding the output of the algorithm (cont.)

- **Important Question.** Once we have found the shortest distance from vertex A to, say, vertex B, how can we also find a path of shortest total distance from A to B (or in other words, a minimum weight path)?

— 'Standing at' vertex N_{i_0}, we repeat the above process for N_{i_0} now: we look at all neighbours of N_{i_0}, except for vertex B, which have been allotted a permanent weight; let's write $M_1, M_2, \ldots, M_t$ for these neighbours of N_{i_0}.

For any $j_0 \in \{1, 2, \ldots, t\}$ for which we have

$$w_{N_{i_0}} - w_{M_{j_0}} = w_{N_{i_0}, M_{j_0}} = \text{weight of the edge } \{N_{i_0}, M_{j_0}\},$$

we can choose the path of shortest total distance that we are trying to find to move next to vertex M_{j_0}.

— We continue like this until we reach vertex A, and thus find a full $A-B$ path of shortest total distance as we wanted *(we are guaranteed to reach vertex A after finitely many steps, because there are only finitely many vertices with permanent weights to consider).*

Finding minimum weight paths from A to B

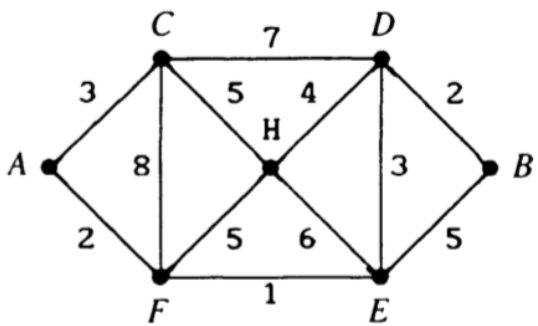
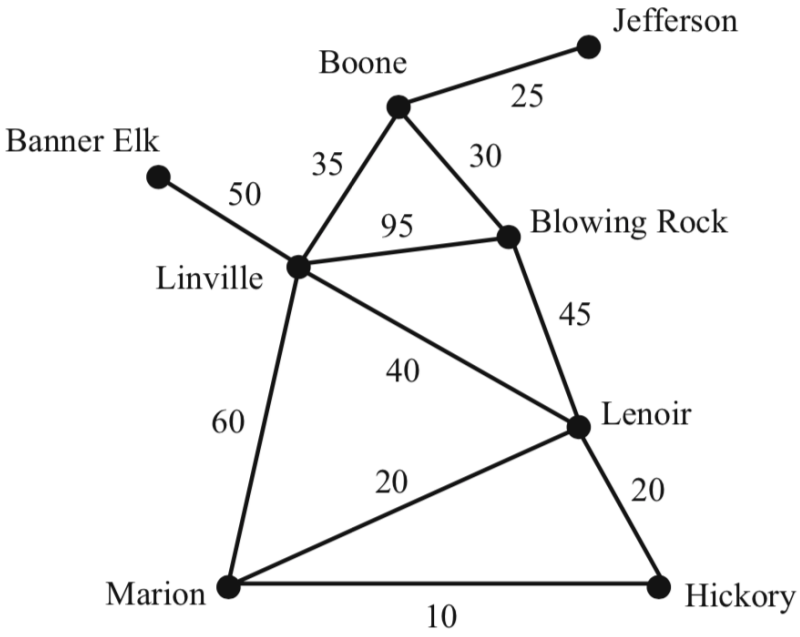


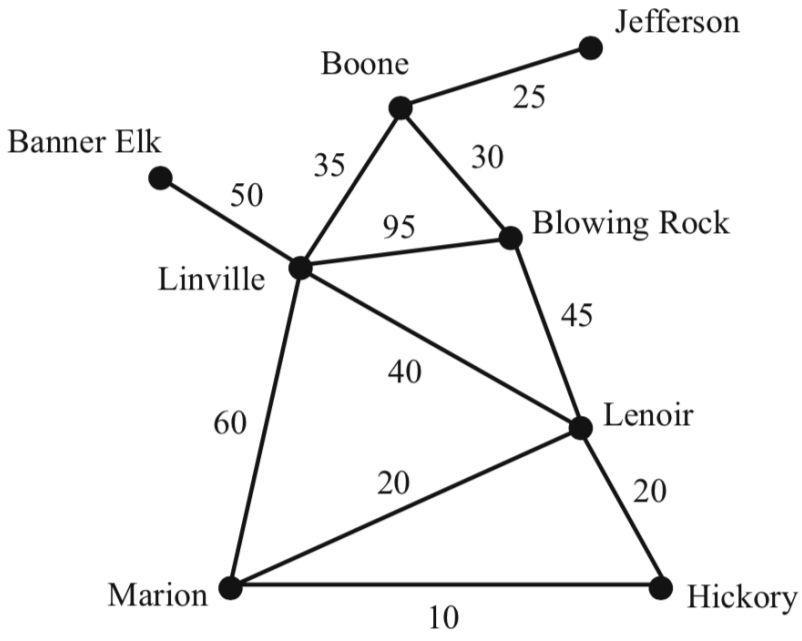
TABLE 10.5. *Steps of algorithm for shortest path from A to B*

	A	*B*	*C*	*D*	*E*	*F*	*H*
Iteration 0	[0]	∞	∞	∞	∞	∞	∞
Iteration 1	[0]	∞	[3]	∞	∞	[2]	∞
Iteration 2	[0]	∞	[3]	∞	[3]	[2]	[7]
Iteration 3	[0]	∞	[3]	[10]	[3]	[2]	[7]
Iteration 4	[0]	[8]	[3]	[6]	[3]	[2]	[7]
Iteration 5	[0]	[8]	[3]	[6]	[3]	[2]	[7]
Iteration 6	[0]	[8]	[3]	[6]	[3]	[2]	[7]

Applying Dijkstra's algorithm to examples (cont.)



Question. What is the shortest possible distance a path connecting the cities of Marion and Boone can cover? What about a path connecting the cities of Marion and Jefferson? Can you find minimum weight paths too?


Jefferson
Boone
25
Banner Elk
35
30
50
95
Blowing Rock
Linville
45
40
60
Lenoir
20
20
Marion
Hickory
10

What about the Travelling Salesman Problem, and the Euler-Königsberg Bridges Problem?

The travelling salesman problem

Let G_0 be a weighted connected graph whose vertices represent different cities that a salesman wants to visit, which are connected by, say, roads and highways, or by train routes, or by airline routes, represented by the edges of the graph *(with each edge weight capturing the cost or distance of travel from one city - endvertex to the other city - endvertex joined by the corresponding edge).*

Question 1. What is the most cost-efficient (or time-efficient) way for the salesman to visit all the cities and finally return to the city which he is supposed to start from?

Question 2. Is there a way for the salesman to visit all the cities but not pass by any city more than once (except perhaps in the case that he returns to the city where he starts from at the end of his trip)?

An efficient scenic route...

The city of Königsberg, Prussia was set on the Pregel River, and included two large islands that were connected to each other and the mainland by seven bridges.

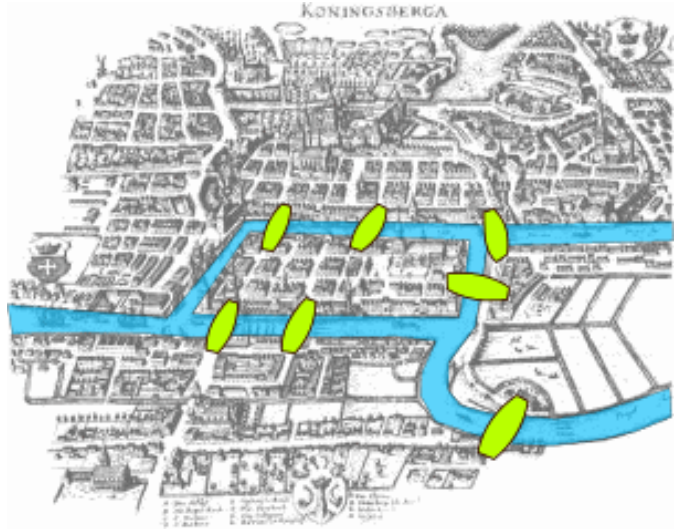


Image from Wikipedia: Map of the city in Leonhard Euler's time showing the actual layout of the seven bridges, and highlighting the river Pregel and the bridges.

People spent time trying to discover a way in which they could cross each bridge exactly once before returning to the point / place in the city that they started from.

Reminder: paths, cycles, trails and circuits

Let $G = (V, E)$ be a graph.

- walks A *walk* of length k in G is a sequence of (not necessarily distinct) vertices $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ from V, such that $v_i v_{i+1} \in E(G)$ for every $i = 0, 1, 2, \ldots, k-1$. The vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

Reminder: paths, cycles, trails and circuits

Let $G = (V, E)$ be a graph.

- walks A *walk* of length k in G is a sequence of (not necessarily distinct) vertices $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ from V, such that $v_i v_{i+1} \in E(G)$ for every $i = 0, 1, 2, \ldots, k-1$. The vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

  Recall that, since G here is a graph (and thus, according to the definitions in this course, it does not contain multiple edges), we can completely describe the walk by simply writing the vertices it passes through, one next to the other, in the correct order: $v_{i_0}\, v_{i_1}\, v_{i_2} \cdots v_{i_k}$.

Reminder: paths, cycles, trails and circuits

Let $G = (V, E)$ be a graph.

- walks A *walk* of length k in G is a sequence of (not necessarily distinct) vertices $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ from V, such that $v_i v_{i+1} \in E(G)$ for every $i = 0, 1, 2, \ldots, k-1$. The vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

  Recall that, since G here is a graph (and thus, according to the definitions in this course, it does not contain multiple edges), we can completely describe the walk by simply writing the vertices it passes through, one next to the other, in the correct order: $v_{i_0}\, v_{i_1}\, v_{i_2} \cdots v_{i_k}$.

- paths A path in G is simply a walk in which **all the vertices are distinct**.

Reminder: paths, cycles, trails and circuits

Let $G = (V, E)$ be a graph.

- walks A *walk* of length k in G is a sequence of (not necessarily distinct) vertices $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ from V, such that $v_i v_{i+1} \in E(G)$ for every $i = 0, 1, 2, \ldots, k-1$. The vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

  Recall that, since G here is a graph (and thus, according to the definitions in this course, it does not contain multiple edges), we can completely describe the walk by simply writing the vertices it passes through, one next to the other, in the correct order: $v_{i_0}\, v_{i_1}\, v_{i_2} \cdots v_{i_k}$.

- paths A path in G is simply a walk in which **all the vertices are distinct**.
- cycles A cycle is a 'closed path', that is, a walk in which all vertices are distinct except for the terminal vertex which coincides with the initial vertex.

Reminder: paths, cycles, trails and circuits

Let $G = (V, E)$ be a graph.

- walks A *walk* of length k in G is a sequence of (not necessarily distinct) vertices $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ from V, such that $v_i v_{i+1} \in E(G)$ for every $i = 0, 1, 2, \ldots, k-1$. The vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

  Recall that, since G here is a graph (and thus, according to the definitions in this course, it does not contain multiple edges), we can completely describe the walk by simply writing the vertices it passes through, one next to the other, in the correct order: $v_{i_0}\, v_{i_1}\, v_{i_2} \cdots v_{i_k}$.

- paths A path in G is simply a walk in which **all the vertices are distinct**.
- cycles A cycle is a 'closed path', that is, a walk in which all vertices are distinct except for the terminal vertex which coincides with the initial vertex.
- trails If **all the edges in a walk are distinct** (but not necessarily all the vertices), we call this walk a *trail*.

Reminder: paths, cycles, trails and circuits

Let $G = (V, E)$ be a graph.

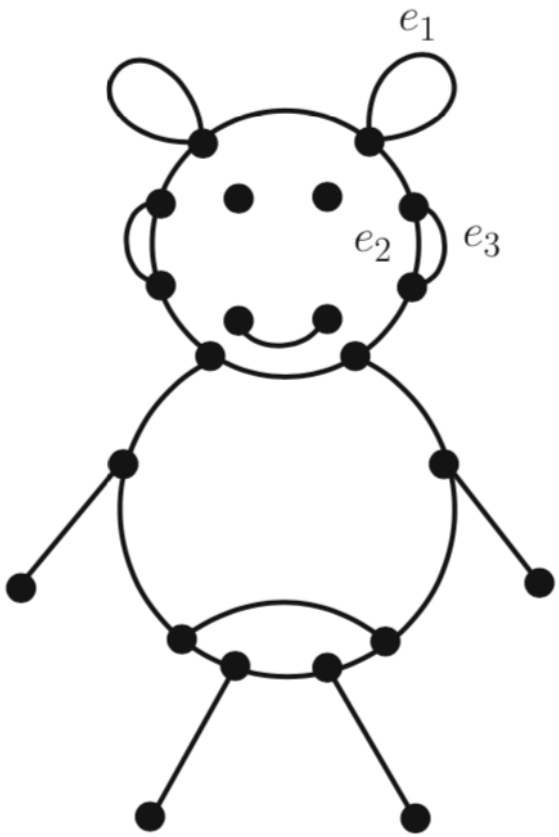
- walks A *walk* of length k in G is a sequence of (not necessarily distinct) vertices $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ from V, such that $v_i v_{i+1} \in E(G)$ for every $i = 0, 1, 2, \ldots, k-1$. The vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

  Recall that, since G here is a graph (and thus, according to the definitions in this course, it does not contain multiple edges), we can completely describe the walk by simply writing the vertices it passes through, one next to the other, in the correct order: $v_{i_0}\, v_{i_1}\, v_{i_2} \cdots v_{i_k}$.

- paths A path in G is simply a walk in which **all the vertices are distinct**.
- cycles A cycle is a 'closed path', that is, a walk in which all vertices are distinct except for the terminal vertex which coincides with the initial vertex.
- trails If **all the edges in a walk are distinct** (but not necessarily all the vertices), we call this walk a *trail*.
- circuits A *circuit* is a 'closed trail', that is, a walk in which **all edges are distinct**, and also the endvertices coincide.

Same objects in multigraphs?

Recall the definition of a 'multigraph':

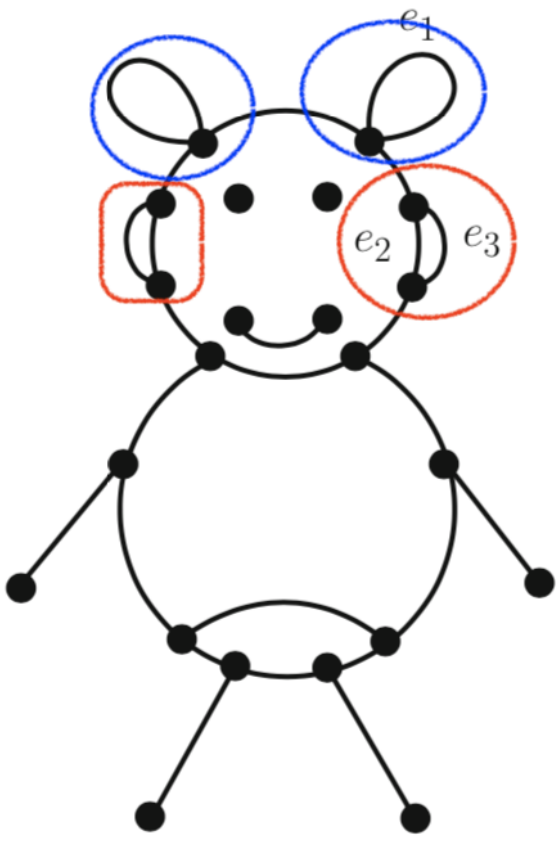


A *multigraph* G is an ordered pair $(V(G), E(G))$,

- where $V(G)$ is a non-empty set (whose elements are called the *vertices* or *nodes* of G),
- and where $E(G)$ is a **multiset** whose elements are taken from the set of 2-element and 1-element subsets of $V(G)$. The elements of $E(G)$ are called the *edges* of G

Same objects in multigraphs?

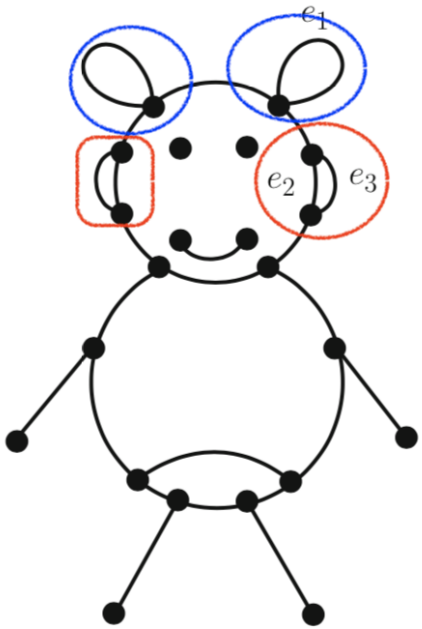
Recall the definition of a 'multigraph':



A *multigraph* G is an ordered pair $(V(G), E(G))$,

- where $V(G)$ is a non-empty set (whose elements are called the *vertices* or *nodes* of G),
- and where $E(G)$ is a **multiset** whose elements are taken from the set of 2-element and 1-element subsets of $V(G)$. The elements of $E(G)$ are called the *edges* of G (and now we can also have 'repeated' edges connecting the same pair of vertices, which we call groups of *multiple edges* or *parallel edges*, as well as *loops*).

Degree of a vertex in a multigraph

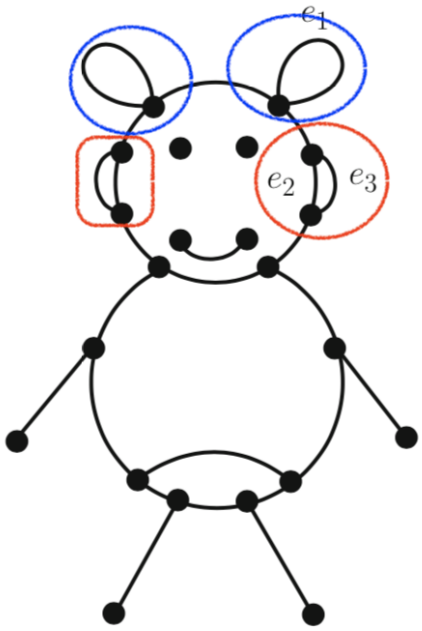


'Alternative' Definition

In a multigraph G, we define the degree of a vertex v_0 of G to be the number of edges which are incident to v_0.

By convention, if v_0 has loops attached to it, then each such loop contributes 2 to the degree of v_0.

Degree of a vertex in a multigraph



'Alternative' Definition

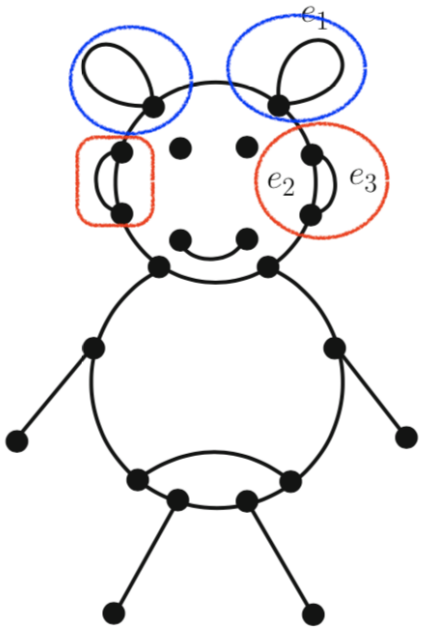
In a multigraph G, we define the degree of a vertex v_0 of G to be the number of edges which are incident to v_0.

By convention, if v_0 has loops attached to it, then each such loop contributes 2 to the degree of v_0.

One way to correctly determine the degree of a vertex v_0 in a multigraph is to examine how many different line segments / arcs we could 'walk' on moving away from v_0;

thinking in this way, it becomes clearer why each loop contributes 2 to the degree of v_0.

Degree of a vertex in a multigraph



'Alternative' Definition

In a multigraph G, we define the degree of a vertex v_0 of G to be the number of edges which are incident to v_0.

By convention, if v_0 has loops attached to it, then each such loop contributes 2 to the degree of v_0.

One way to correctly determine the degree of a vertex v_0 in a multigraph is to examine how many different line segments / arcs we could 'walk' on moving away from v_0;

thinking in this way, it becomes clearer why each loop contributes 2 to the degree of v_0.

With this definition, the Handshaking Lemma, as well as its first Corollary, that we saw earlier in the term, continue to hold in a multigraph.

Handshaking Lemma

Let G be a finite multigraph, with vertex set V and size $e(G)$. Then $\sum_{v_i \in V} \deg(v_i) = 2e(G)$.

If V_{odd} is the subset of the vertices in G which have odd degree, then V_{odd} must have **even cardinality**.

Adjacency matrix and incidence matrix for a multigraph?

Definition 1: Adjacency Matrix of a Multigraph

Let $G = (V, E)$ be a **multigraph** with vertex set $V = \{v_1, v_2, \ldots, v_n\}$. Then its *adjacency matrix* A_G is an $n \times n$ matrix which has the following properties:

- if $1 \leqslant i, j \leqslant n$ and $i \neq j$, then the (i, j)-th entry of A_G equals **the number of edges in G with endvertices v_i and v_j** *(of course if there are no edges joining v_i and v_j, the entry will be 0);*

Adjacency matrix and incidence matrix for a multigraph?

Definition 1: Adjacency Matrix of a Multigraph

Let $G = (V, E)$ be a **multigraph** with vertex set $V = \{v_1, v_2, \ldots, v_n\}$. Then its *adjacency matrix* A_G is an $n \times n$ matrix which has the following properties:

- if $1 \leqslant i, j \leqslant n$ and $i \neq j$, then the (i, j)-th entry of A_G equals **the number of edges in G with endvertices v_i and v_j** *(of course if there are no edges joining v_i and v_j, the entry will be 0);*
- for every $1 \leqslant i \leqslant n$, the diagonal (i, i)-th entry is **2 times the number of loops** at vertex v_i.

Adjacency matrix and incidence matrix for a multigraph?

Definition 1: Adjacency Matrix of a Multigraph

Let $G = (V, E)$ be a **multigraph** with vertex set $V = \{v_1, v_2, \ldots, v_n\}$. Then its *adjacency matrix* A_G is an $n \times n$ matrix which has the following properties:

- if $1 \leqslant i, j \leqslant n$ and $i \neq j$, then the (i, j)-th entry of A_G equals **the number of edges in G with endvertices v_i and v_j** *(of course if there are no edges joining v_i and v_j, the entry will be 0);*
- for every $1 \leqslant i \leqslant n$, the diagonal (i, i)-th entry is **2 times the number of loops** at vertex v_i.

Important Observation

For a(n) (undirected) multigraph, we note that the adjacency matrix is again a symmetric matrix (note however that it might no longer be a $0-1$ matrix; it will be so **if and only if** the multigraph G is also a(n) (ordinary) graph).

Adjacency matrix and incidence matrix for a multigraph? (cont.)

Definition 2: Incidence Matrix of a Multigraph

Let $G = (V, E)$ be a **multigraph** with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$ *(note that in the edge set we also include all the loops in G, if any exist)*.

Then the *incidence matrix* I_G of G is an $n \times \boldsymbol{m}$ matrix which has the following properties: for every $1 \leqslant i \leqslant n$ and every $1 \leqslant j \leqslant m$,

Adjacency matrix and incidence matrix for a multigraph? (cont.)

Definition 2: Incidence Matrix of a Multigraph

Let $G = (V, E)$ be a **multigraph** with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$ *(note that in the edge set we also include all the loops in G, if any exist)*.

Then the *incidence matrix* I_G of G is an $n \times \boldsymbol{m}$ matrix which has the following properties: for every $1 \leqslant i \leqslant n$ and every $1 \leqslant j \leqslant m$,

- **if e_j is NOT a loop of G,** then
  - the (i, j)-th entry of I_G is equal to 1 if vertex v_i is one of the endvertices of edge e_j,
  - and it is equal to 0 otherwise.

Adjacency matrix and incidence matrix for a multigraph? (cont.)

Definition 2: Incidence Matrix of a Multigraph

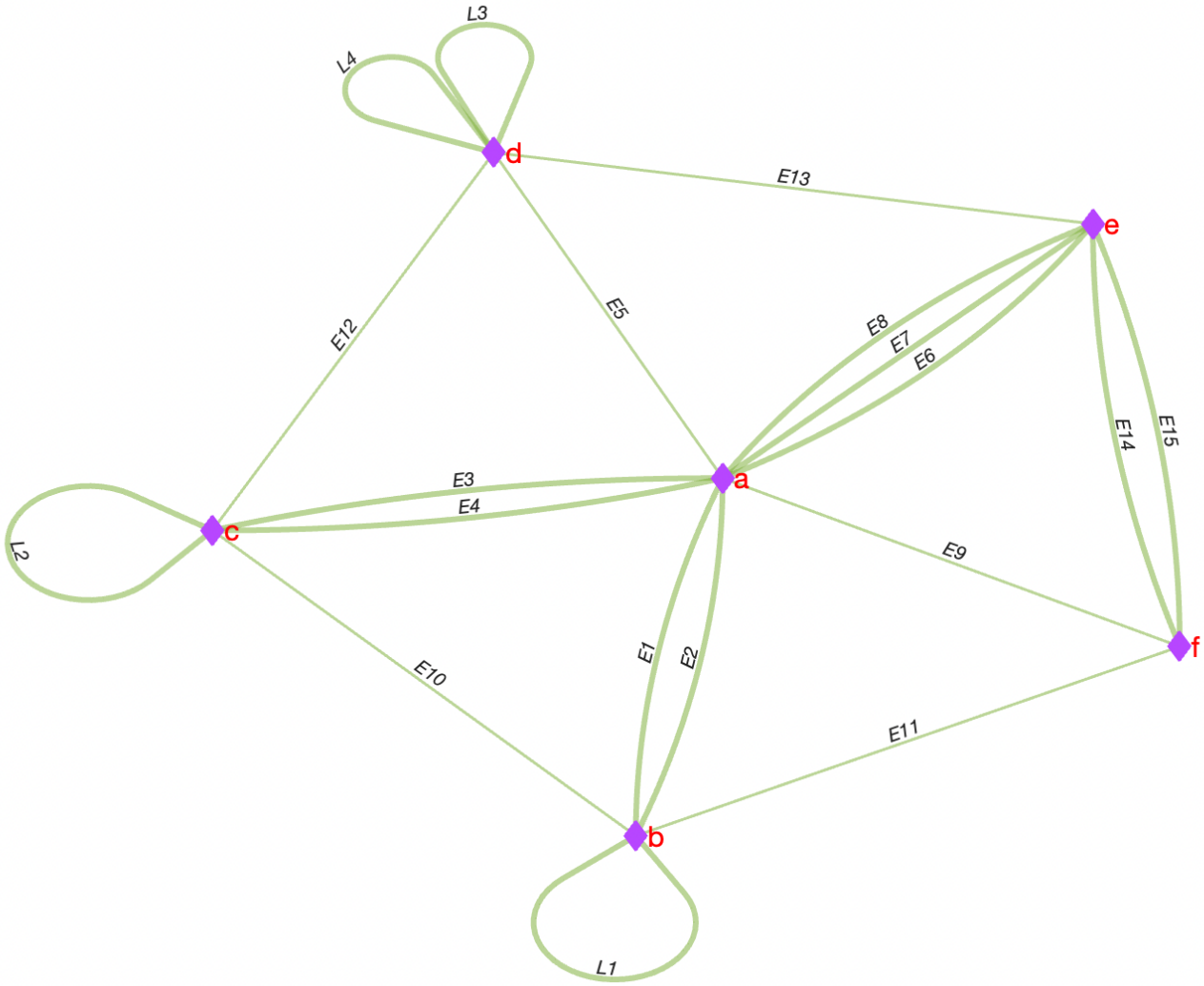
Let $G = (V, E)$ be a **multigraph** with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$ *(note that in the edge set we also include all the loops in G, if any exist)*.

Then the *incidence matrix* I_G of G is an $n \times \boldsymbol{m}$ matrix which has the following properties: for every $1 \leqslant i \leqslant n$ and every $1 \leqslant j \leqslant m$,

- **if e_j is NOT a loop of G,** then
  - the (i, j)-th entry of I_G is equal to 1 if vertex v_i is one of the endvertices of edge e_j,
  - and it is equal to 0 otherwise.
- **if e_j IS a loop of G,** then
  - the (i, j)-th entry of I_G is equal to 2 if e_j is a loop at vertex v_i,
  - and it is equal to 0 otherwise.

Practise on an example

Let G be the following multigraph (where the vertices and edges have all been labelled):



Practise on an example (cont.)

We have that

$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 2 & 2 & 1 & 3 & 1 \\ 2 & 2 & 1 & 0 & 0 & 1 \\ 2 & 1 & 2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 4 & 1 & 0 \\ 3 & 0 & 0 & 1 & 0 & 2 \\ 1 & 1 & 0 & 0 & 2 & 0 \end{pmatrix} \end{array}$$

and

$$I_G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccccccccccccccc} E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 & E9 & L1 & E10 & E11 & L2 & E12 & L3 & L4 & E13 & E14 & E15 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}.$$

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s};

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s}; note that now we may have cases where consecutive vertices are equal, that is, cases where $v_{i_{s-1}} = v_{i_s}$, and then e_{j_s} should be a loop at vertex v_{i_s}).

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s}; note that now we may have cases where consecutive vertices are equal, that is, cases where $v_{i_{s-1}} = v_{i_s}$, and then e_{j_s} should be a loop at vertex v_{i_s}).

As before, the vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

  where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s}; note that now we may have cases where consecutive vertices are equal, that is, cases where $v_{i_{s-1}} = v_{i_s}$, and then e_{j_s} should be a loop at vertex v_{i_s}).

  As before, the vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

- paths A path in H is simply a walk in which **all the vertices are distinct** *(note that, as a consequence of this definition, a path will not contain any loops)*.

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

  where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s}; note that now we may have cases where consecutive vertices are equal, that is, cases where $v_{i_{s-1}} = v_{i_s}$, and then e_{j_s} should be a loop at vertex v_{i_s}).

  As before, the vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

- paths A path in H is simply a walk in which **all the vertices are distinct** *(note that, as a consequence of this definition, a path will not contain any loops)*.
- cycles A cycle is a closed path.

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

  where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s}; note that now we may have cases where consecutive vertices are equal, that is, cases where $v_{i_{s-1}} = v_{i_s}$, and then e_{j_s} should be a loop at vertex v_{i_s}).

  As before, the vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

- paths A path in H is simply a walk in which **all the vertices are distinct** *(note that, as a consequence of this definition, a path will not contain any loops)*.
- cycles A cycle is a closed path.
- trails A trail in H is a walk in which **all the edges are distinct** (but not necessarily all the vertices).

Walks, paths, cycles, trails and circuits in a multigraph

Let $H = (V, E)$ be a multigraph in H, and suppose that $V = \{v_1, v_2, \ldots, v_n\}$, while $E = \{e_1, e_2, \ldots, e_m\}$ *(the latter set may include different edges which have the same pair of endvertices (and thus form groups of multiple edges), as well as loops; it's even possible that we have 'parallel' loops, that is, distinct loops which are attached to the same vertex, as e.g. we did in the previous example).*

- walks A *walk* of length k in H is now represented by a sequence of the form

$$v_{i_0}\ e_{j_1}\ v_{i_1}\ e_{j_2}\ v_{i_2} \cdots v_{i_{k-1}}\ e_{j_k}\ v_{i_k}$$

  where $v_{i_0}, v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are vertices from $V(H)$ (not necessarily distinct), $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ are edges from $E(H)$ (not necessarily distinct), **and for every $s = 1, 2, \ldots, k$ we have that $e_{j_s} = \{v_{i_{s-1}}, v_{i_s}\}$** (or in other words, e_{j_s} joins the vertices $v_{i_{s-1}}$ and v_{i_s}; note that now we may have cases where consecutive vertices are equal, that is, cases where $v_{i_{s-1}} = v_{i_s}$, and then e_{j_s} should be a loop at vertex v_{i_s}).

  As before, the vertices v_{i_0} and v_{i_k} are called the *endvertices* of the walk, and we sometimes say that this is a $v_{i_0} - v_{i_k}$ walk.

- paths A path in H is simply a walk in which **all the vertices are distinct** *(note that, as a consequence of this definition, a path will not contain any loops)*.
- cycles A cycle is a closed path.
- trails A trail in H is a walk in which **all the edges are distinct** (but not necessarily all the vertices).
- circuits Finally, a circuit is a closed trail.

Eulerian graphs (and multigraphs)

Definition

Let G be a connected graph (or multigraph) which is non-trivial, that is, it contains at least one edge.

Eulerian graphs (and multigraphs)

Definition

Let G be a connected graph (or multigraph) which is non-trivial, that is, it contains at least one edge.

- An *Euler trail* in G is a trail that passes by **all** edges in G (and hence, given that it is a trail, it passes by each edge exactly once).

Eulerian graphs (and multigraphs)

Definition

Let G be a connected graph (or multigraph) which is non-trivial, that is, it contains at least one edge.

- An *Euler trail* in G is a trail that passes by **all** edges in G (and hence, given that it is a trail, it passes by each edge exactly once).
- An *Euler circuit* in G is a circuit (that is, a closed trail) that passes by **all** edges in G.

Eulerian graphs (and multigraphs)

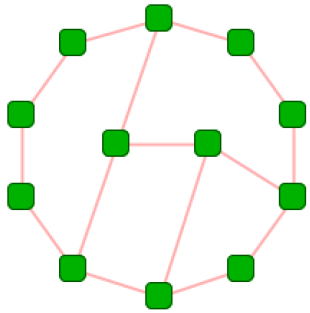
Definition

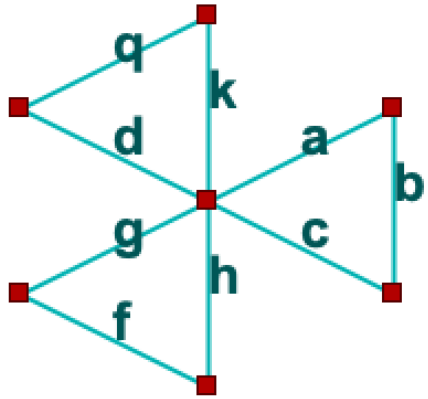
Let G be a connected graph (or multigraph) which is non-trivial, that is, it contains at least one edge.

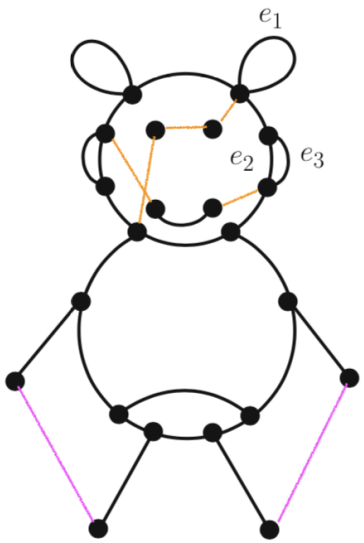
- An *Euler trail* in G is a trail that passes by **all** edges in G (and hence, given that it is a trail, it passes by each edge exactly once).
- An *Euler circuit* in G is a circuit (that is, a closed trail) that passes by **all** edges in G.

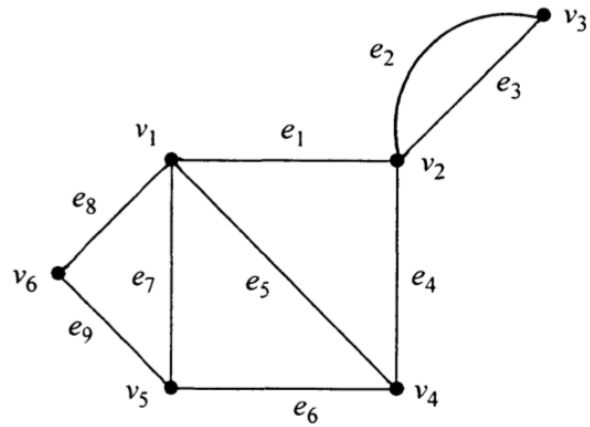
G is called *Eulerian* if we can find (at least) one Euler circuit in G.

Examples and non-examples









bottom row from the Balakrishnan-Ranganathan book (1st image modified)

Hamiltonian graphs

Definition

Let G be a connected graph.

Hamiltonian graphs

Definition

Let G be a connected graph.

- A *Hamilton path* in G is a path that passes through **all** vertices in G (and hence, given that it is a path, it passes through each vertex exactly once).

Hamiltonian graphs

Definition

Let G be a connected graph.

- A *Hamilton path* in G is a path that passes through **all** vertices in G (and hence, given that it is a path, it passes through each vertex exactly once).
- A *Hamilton cycle* in G is a cycle (that is, a closed path) that passes through **all** vertices in G.

Hamiltonian graphs

Definition

Let G be a connected graph.

- A *Hamilton path* in G is a path that passes through **all** vertices in G (and hence, given that it is a path, it passes through each vertex exactly once).
- A *Hamilton cycle* in G is a cycle (that is, a closed path) that passes through **all** vertices in G.

G is called *Hamiltonian* if we can find (at least) one Hamilton cycle in G.

Hamiltonian graphs

Definition

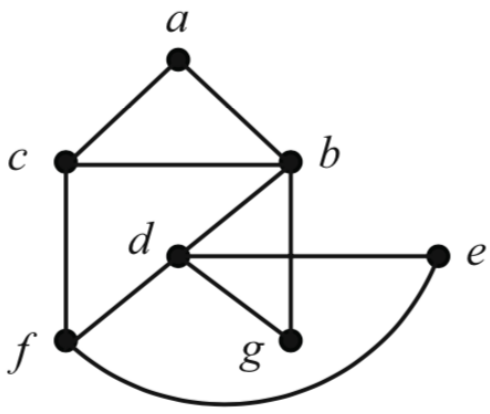
Let G be a connected graph.

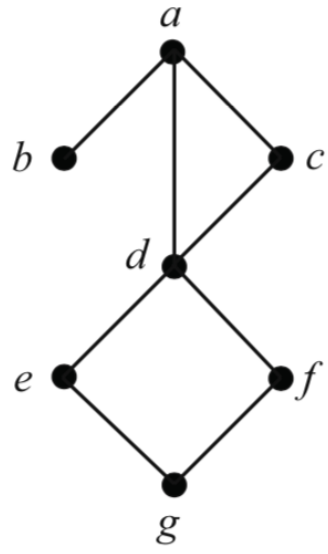
- A *Hamilton path* in G is a path that passes through **all** vertices in G (and hence, given that it is a path, it passes through each vertex exactly once).
- A *Hamilton cycle* in G is a cycle (that is, a closed path) that passes through **all** vertices in G.

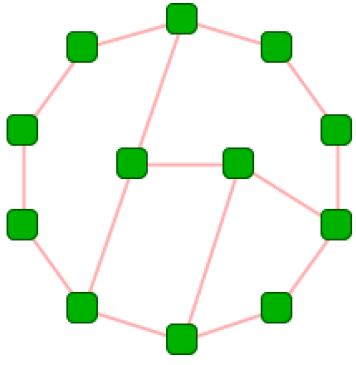
G is called *Hamiltonian* if we can find (at least) one Hamilton cycle in G.

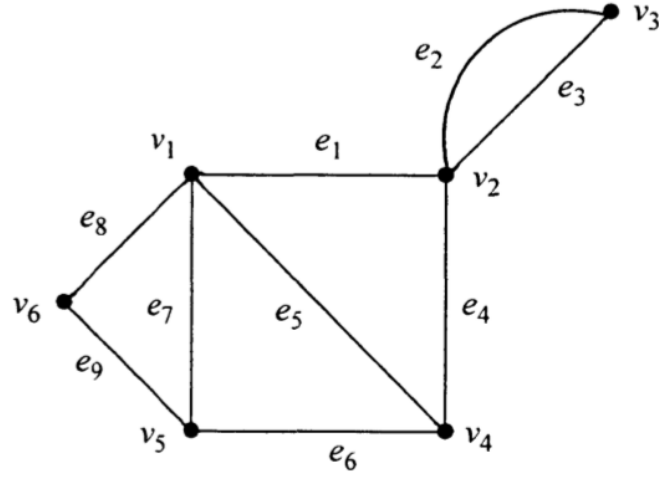
The name is in honour of the mathematician William Hamilton who introduced the idea of looking for Hamilton cycles in graphs (with the first graph he considered being (the 'frame' of) the solid dodecahedron) as a new board game!

Examples and non-examples









The notions are related

Theorem

Let G be a connected graph. If G is Eulerian, then the line graph $L(G)$ of G is both Hamiltonian and Eulerian.

The notions are related

Theorem

Let G be a connected graph. If G is Eulerian, then the line graph $L(G)$ of G is both Hamiltonian and Eulerian.

We will briefly discuss this theorem next time.

Eulerian graphs: necessary and sufficient conditions

Theorem 1

Let G be a (non-trivial) connected graph (or multigraph).

Then G is Eulerian **if and only if** every vertex of G has even degree.

Eulerian graphs: necessary and sufficient conditions

Theorem 1

Let G be a (non-trivial) connected graph (or multigraph).

Then G is Eulerian **if and only if** every vertex of G has even degree.

Proposition 1

Let G be a connected graph (or multigraph).

Then G has an Euler trail, but not an Euler circuit **if and only if** exactly two vertices of G have odd degree (and all other vertices have even degree).

Eulerian graphs: necessary and sufficient conditions

Theorem 1

Let G be a (non-trivial) connected graph (or multigraph).

Then G is Eulerian **if and only if** every vertex of G has even degree.

Proposition 1

Let G be a connected graph (or multigraph).

Then G has an Euler trail, but not an Euler circuit **if and only if** exactly two vertices of G have odd degree (and all other vertices have even degree).

We will see justifications for these results, as well as one more important theorem giving a necessary and sufficient condition for Eulerianity of a multigraph, next time.

Applying Theorem 1 and Proposition 1 to examples

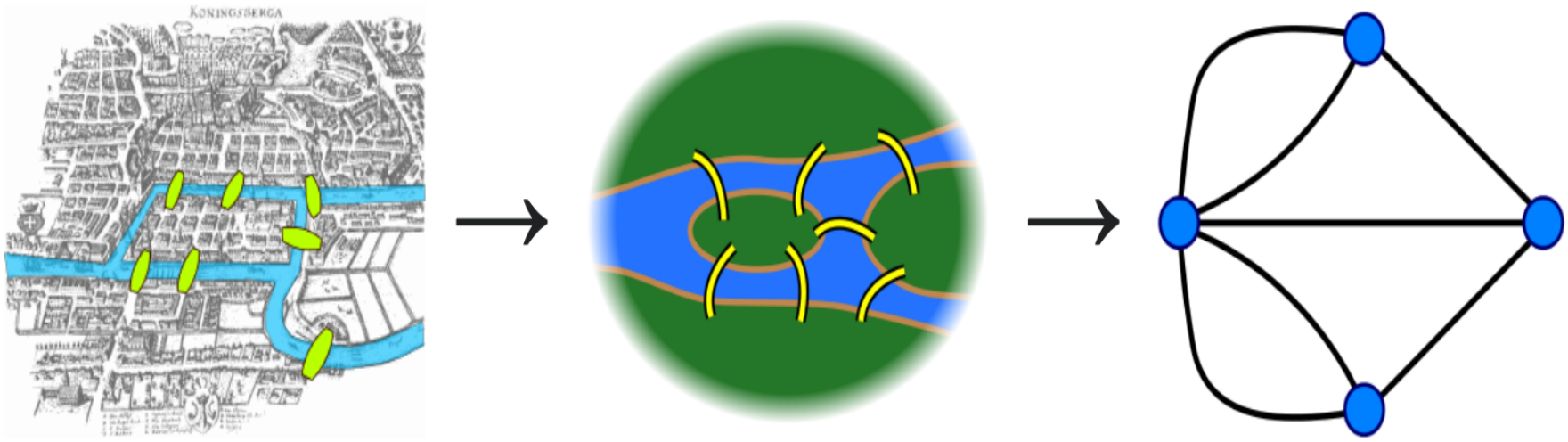
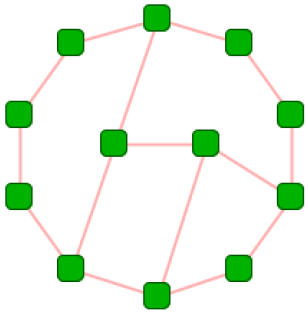
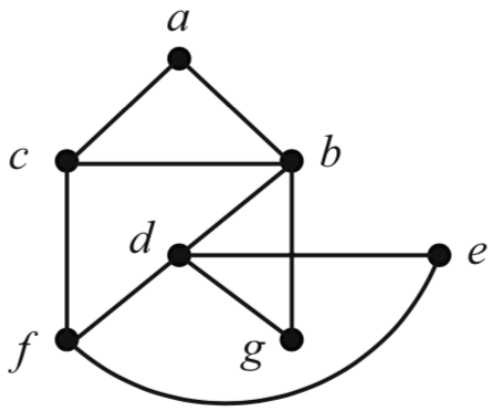
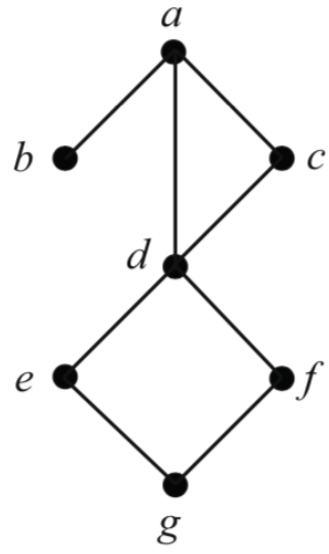


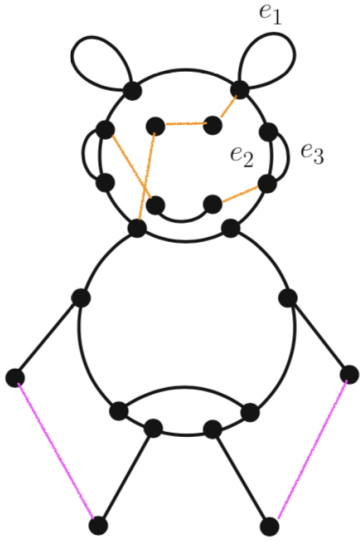
Image from Wikipedia

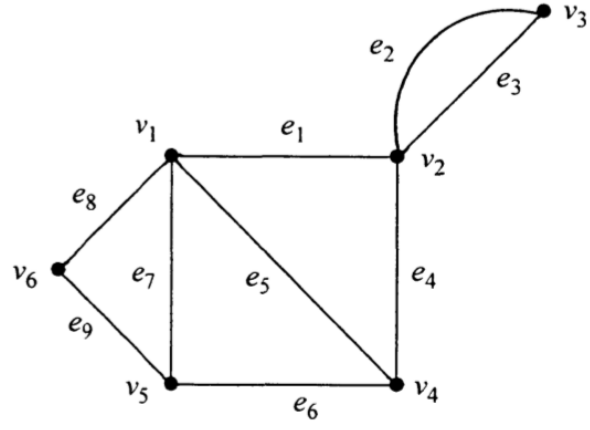
Applying Theorem 1 and Proposition 1 to examples



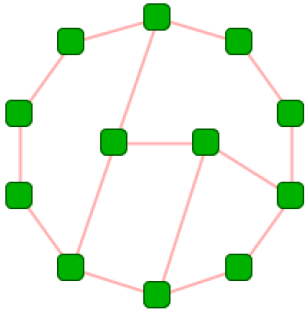


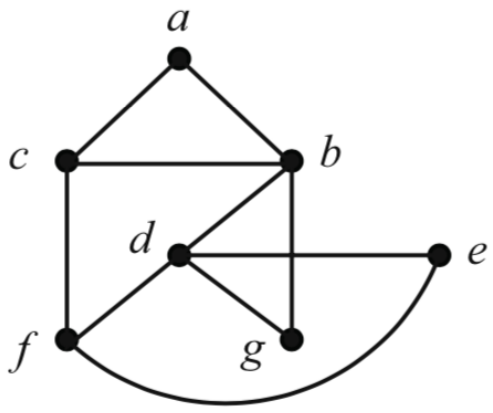


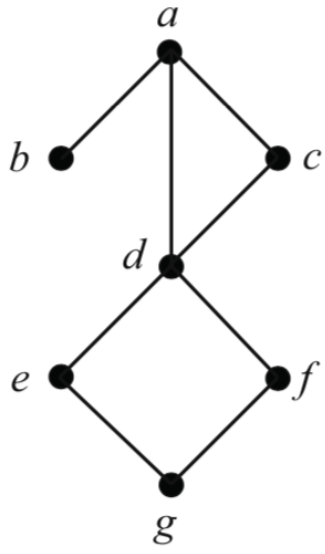




Applying Theorem 1 and Proposition 1 to examples







Applying Theorem 1 and Proposition 1 to examples