

Graph Theory (MT-3001)

Lecturer

USAMA ANTULEY

LECTURE # 01

Course Details

□ Text Book:

1. Saoub, Karin R. **Graph Theory: An Introduction to Proofs, Algorithms, and Applications.** CRC Press, 2021.
2. **Graph theory: undergraduate mathematics / by Khee Meng Koh, Fengming Dong, Kah Loon Ng, Eng Guan Tay. Bondy, John Adrian, and Uppaluri Siva Ramachandra Murty**

□ Reference Book (s):

1. Bondy, John Adrian, and Uppaluri Siva Ramachandra Murty. ***Graph theory with applications.*** Vol. 290. London: Macmillan, 1976.
2. West, Douglas Brent. ***Introduction to graph theory.*** Vol. 2. Upper Saddle River: Prentice Hall, 2001.

Marking Division

S. No	Particulars	% Marks
1.	Assignment (at least 2)	05
2.	Quizzes (at least 2) + Presentation/Project	15
3.	Sessional 1 & 2	30
4.	Final Exam	50
	Total	100

Airlines Graph

Consider a small country with five cities: A, B, C, D, E .

There are six flights:

$$A - B, A - C, A - E, \\ B - D, C - D, C - E.$$

Is there a direct flight from A to D?
With one stop?
With two stops?

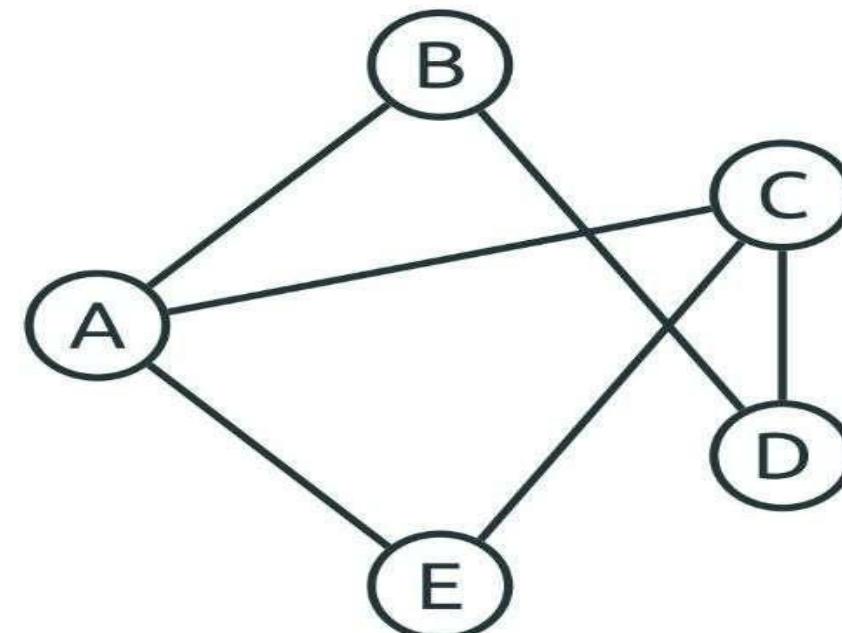
Airlines Graph

Consider a small country with five cities: A, B, C, D, E .

There are six flights:

$A - B, A - C, A - E,$
 $B - D, C - D, C - E.$

Is there a direct flight from A to D?
With one stop?
With two stops?



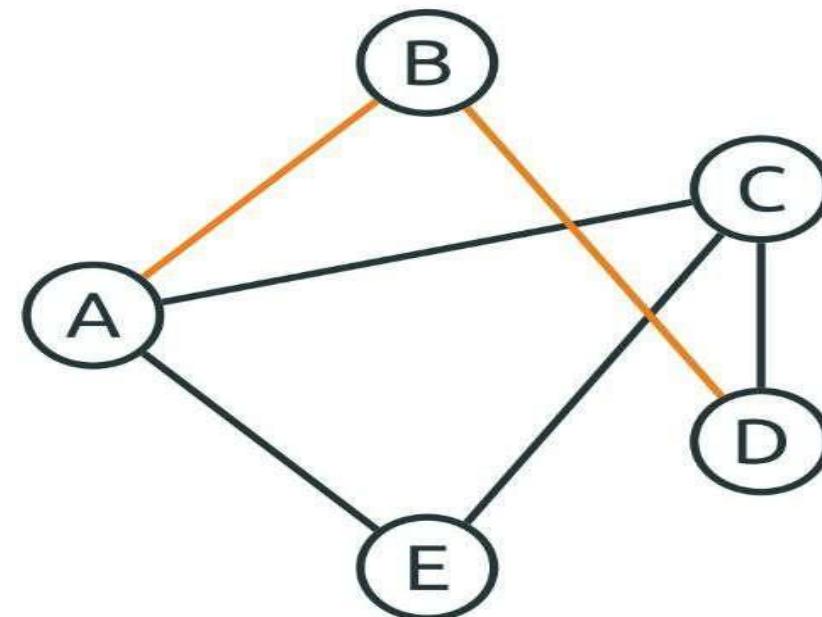
Airlines Graph

Consider a small country with five cities: A, B, C, D, E .

There are six flights:

$A - B, A - C, A - E,$
 $B - D, C - D, C - E.$

Is there a direct flight from A to D?
With one stop?
With two stops?



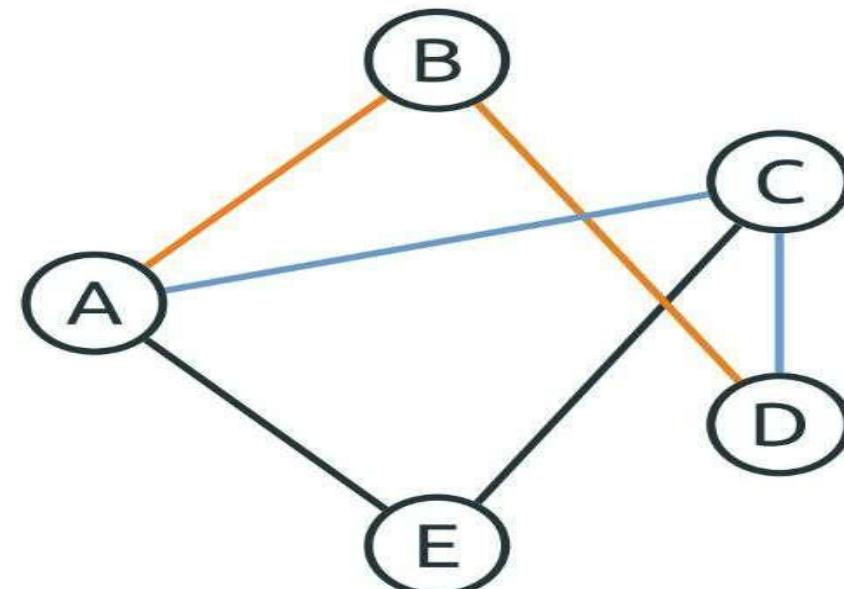
Airlines Graph

Consider a small country with five cities: A, B, C, D, E .

There are six flights:

$A - B, A - C, A - E,$
 $B - D, C - D, C - E.$

Is there a direct flight from A to D?
With one stop?
With two stops?



Outline

What is a Graph?

Graph Examples

Graph Applications

Graphs

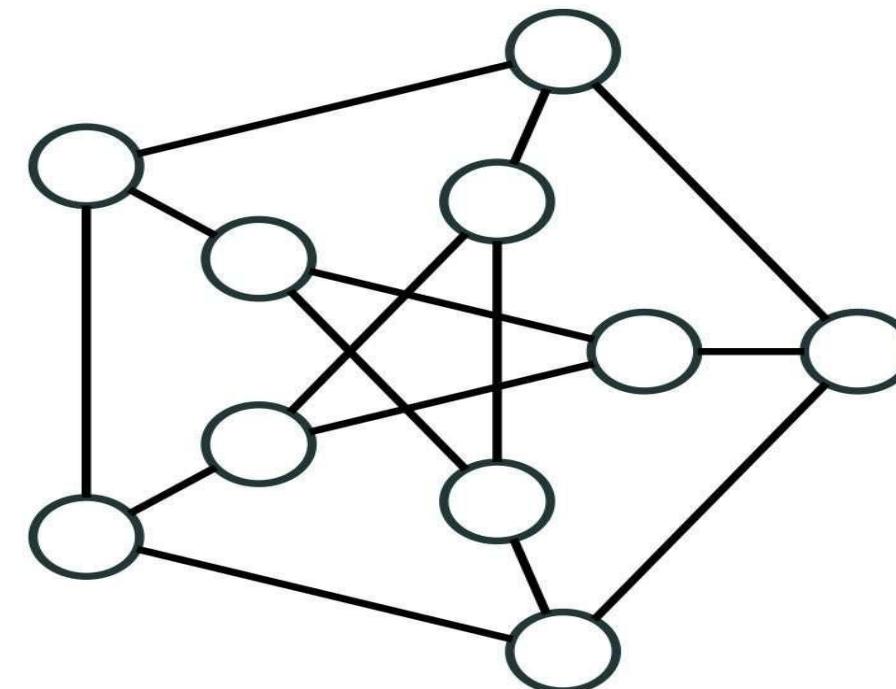
- A set of Objects

Graphs

- A set of **Objects**
- **Relations** between pairs of objects

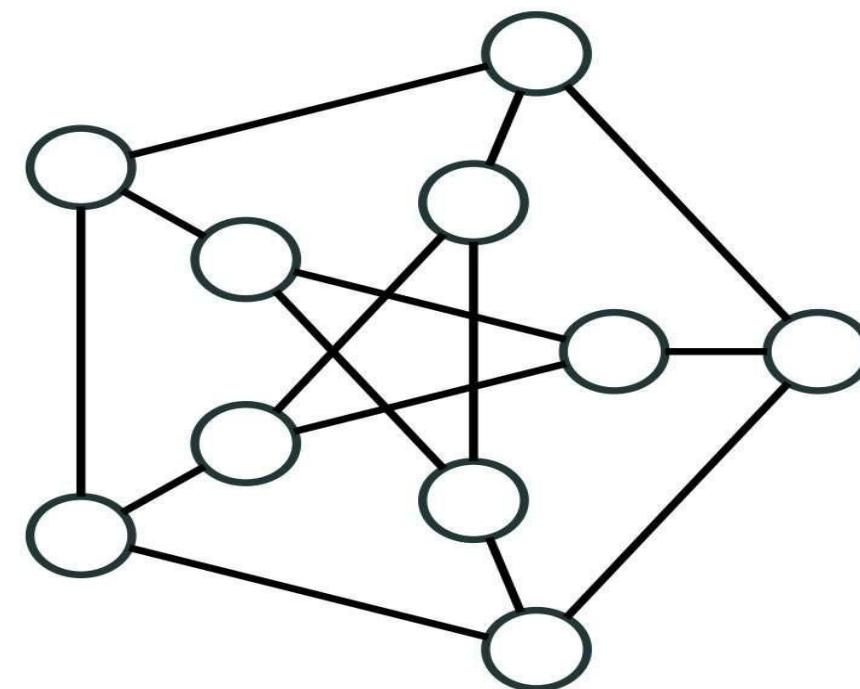
Graphs

- A set of **Objects**
- **Relations** between pairs of objects



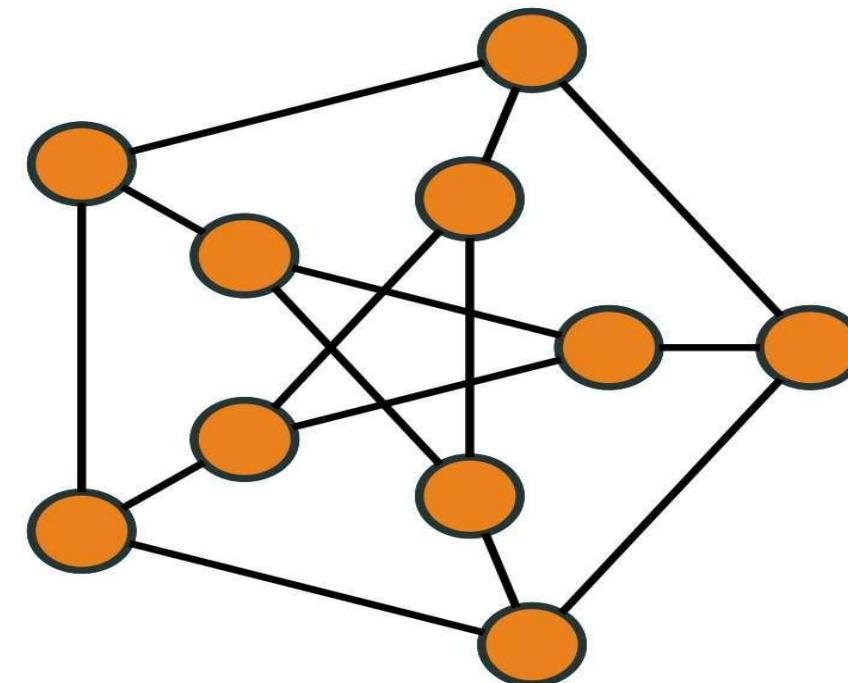
Definition

- A **Graph** $G = (V, E)$



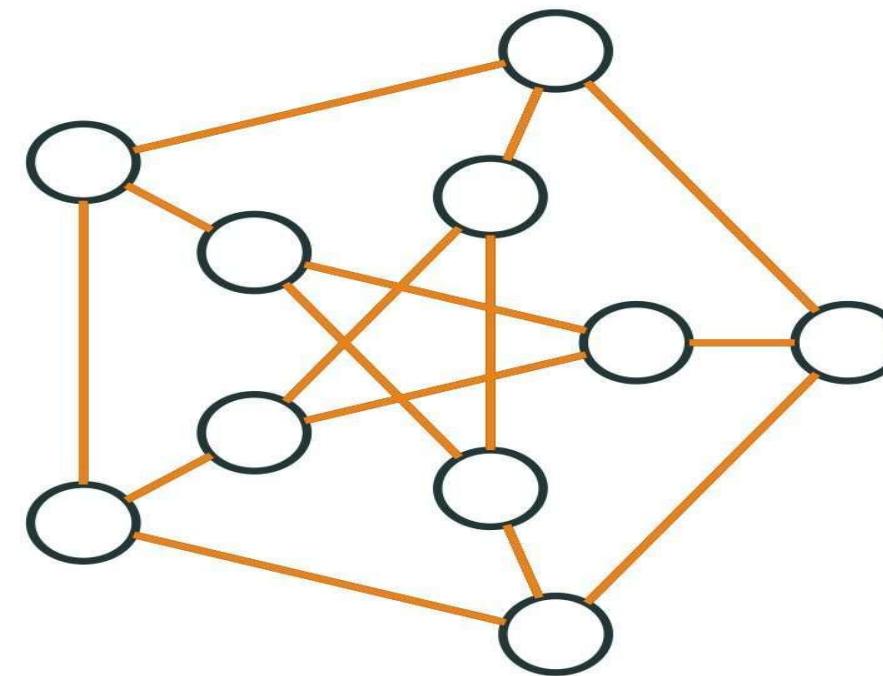
Definition

- A **Graph** $G = (V, E)$
- A set V of **Vertices/Nodes**



Definition

- A **Graph** $G = (V, E)$
- A set V of **Vertices/Nodes**
- A set E of **Edges**



Vocabulary



Vocabulary



- We can name individual vertices and edges

Vocabulary



- We can name individual vertices and edges

Vocabulary



- We can name individual vertices and edges
- e **Connects** u and v

Vocabulary



- We can name individual vertices and edges
- e Connects u and v
- u and v are End Points of e

Vocabulary



- We can name individual vertices and edges
- e **Connects** u and v
- u and v are **End Points** of e
- u and e are **Incident**

Vocabulary



- We can name individual vertices and edges
- e **Connects** u and v
- u and v are **End Points** of e
- u and e are **Incident**
- u and v are **Adjacent**

Vocabulary



- We can name individual vertices and edges
- e Connects u and v
- u and v are End Points of e
- u and e are Incident
- u and v are Adjacent
- u and v are Neighbors

Drawing a Graph

Objects: {A,B,C,D}

Relations: {{A,C},{D,A},{B,D},{C,B}}

Drawing a Graph

Objects: {A,B,C,D}

Relations: {{A,C},{D,A},{B,D},{C,B}}

B

C

A

Drawing a Graph

Objects: {A,B,C,D}

Relations: {{A,C},{D,A},{B,D},{C,B}}

(B)

(C)

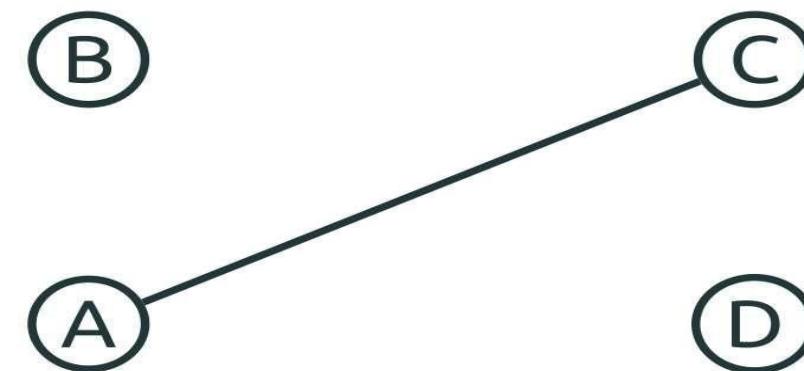
(A)

(D)

Drawing a Graph

Objects: {A,B,C,D}

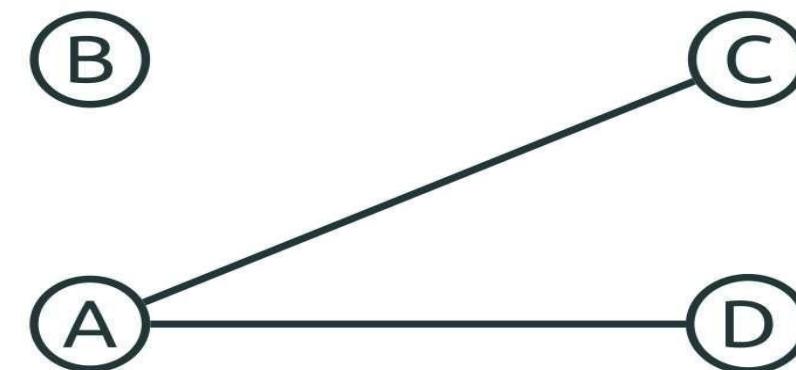
Relations: {{A,C},{D,A},{B,D},{C,B}}



Drawing a Graph

Objects: {A,B,C,D}

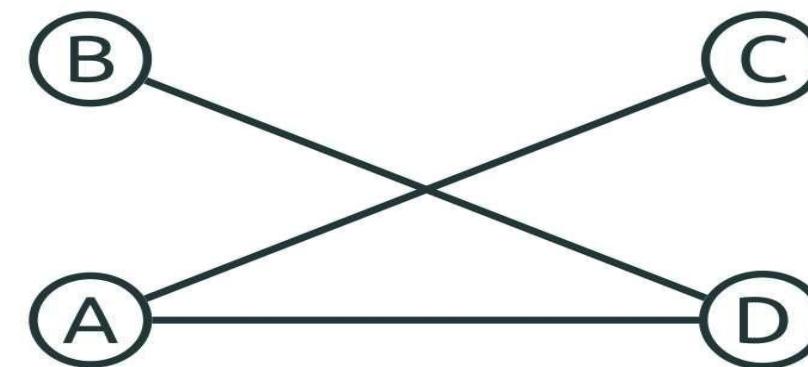
Relations: {{A,C},{D,A},{B,D},{C,B}}



Drawing a Graph

Objects: {A,B,C,D}

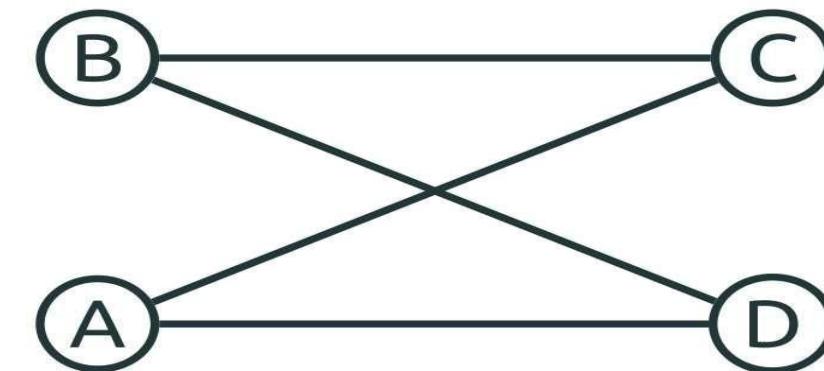
Relations: {{A,C},{D,A},{B,D},{C,B}}



Drawing a Graph

Objects: {A,B,C,D}

Relations: {{A,C},{D,A},{B,D},{C,B}}



Drawing a Graph

Objects: {A,B,C,D}

Relations: {{A,C},{D,A},{B,D},{C,B}}



Drawing a Graph

Objects: {A,B,C,D}

Relations: {{A,C},{D,A},{B,D},{C,B}}



Directed Graph



- It is often convenient to consider **Directed Edges (Arcs)**

Directed Graph



- It is often convenient to consider **Directed Edges (Arcs)**
- They describe **asymmetric** relations

Directed Graph



- It is often convenient to consider **Directed Edges (Arcs)**
- They describe **asymmetric** relations
- There is a flight from A to B, but not the other way around

Directed Graph



- It is often convenient to consider **Directed Edges (Arcs)**
- They describe **asymmetric** relations
- There is a flight from A to B, but not the other way around
- Such a graph is called **Directed**

Drawing a Directed Graph

Objects: {A,B,C,D}

Relations: {(A,C),(D,A),(B,D),(C,B)}



Drawing a Directed Graph

Objects: {A,B,C,D}

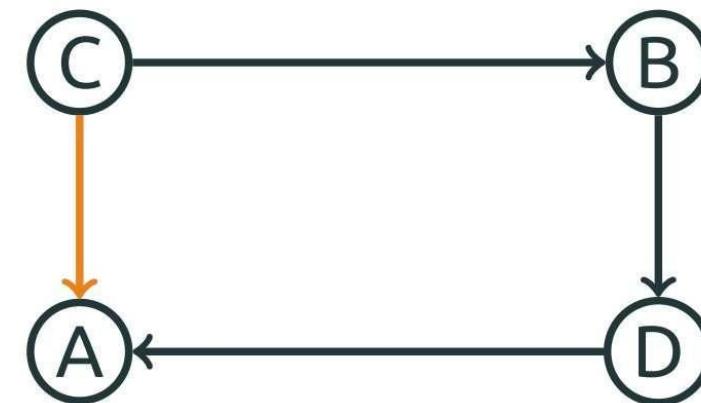
Relations: {(A,C),(D,A),(B,D),(C,B)}



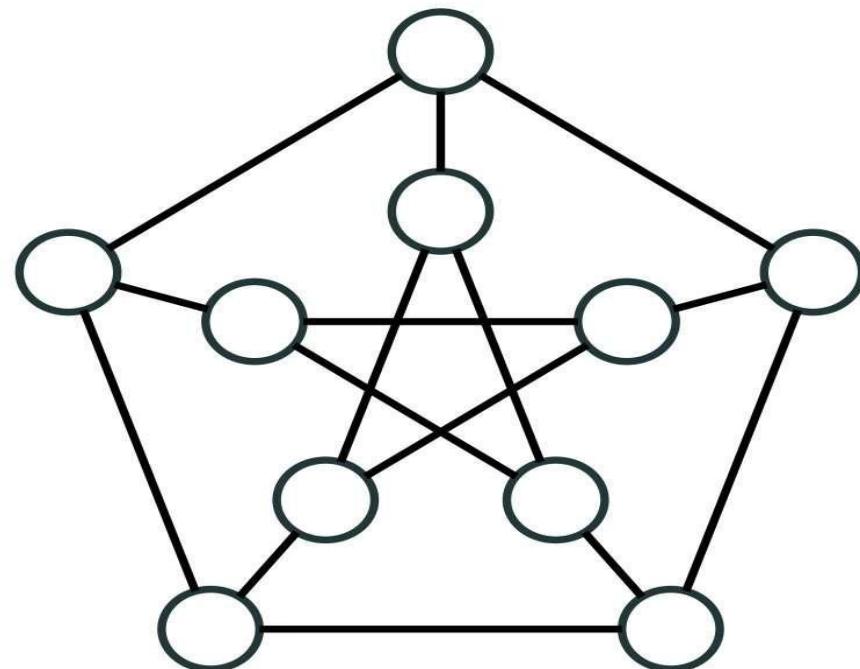
Drawing a Directed Graph

Objects: {A,B,C,D}

Relations: {(C,A),(D,A),(B,D),(C,B)}

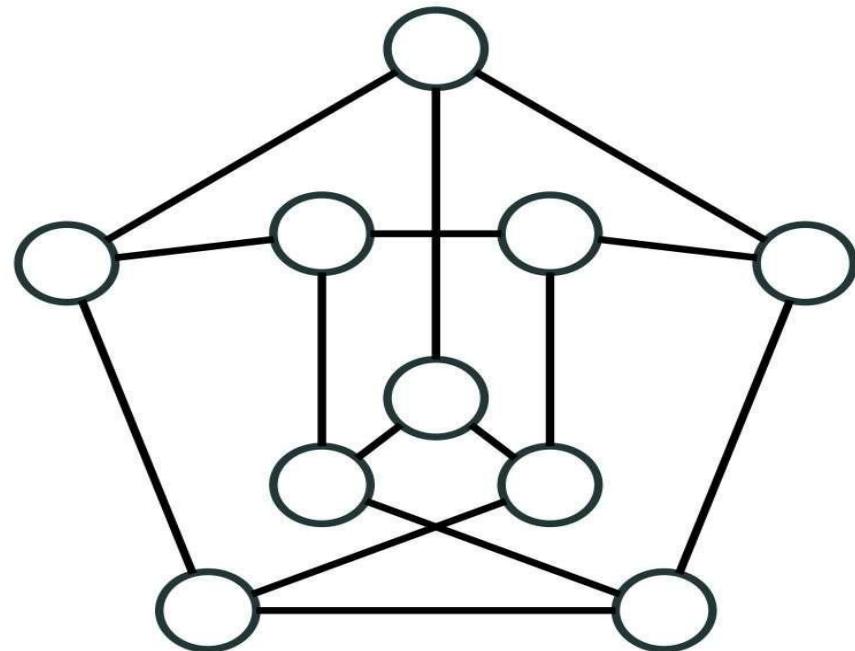
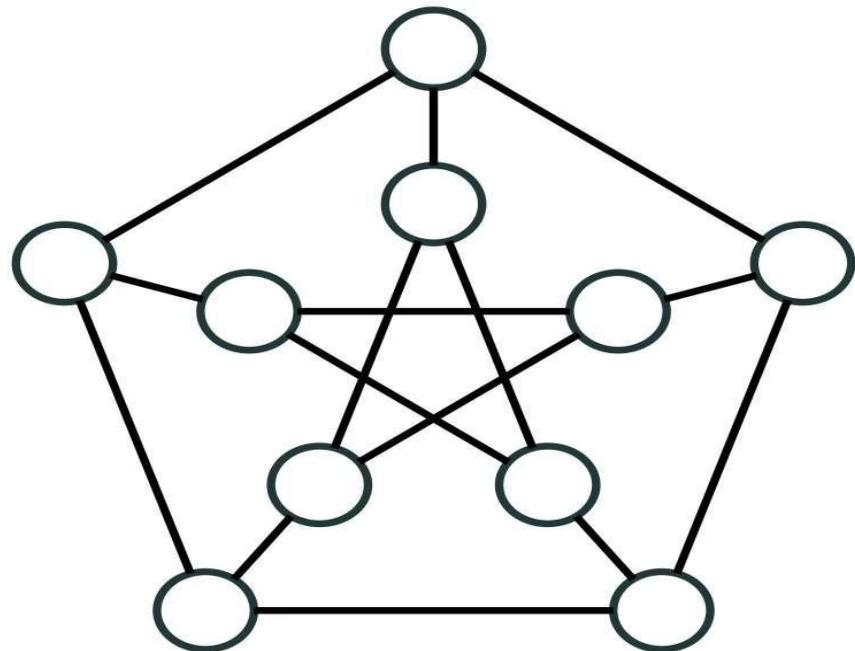


Many Ways to Draw a Graph



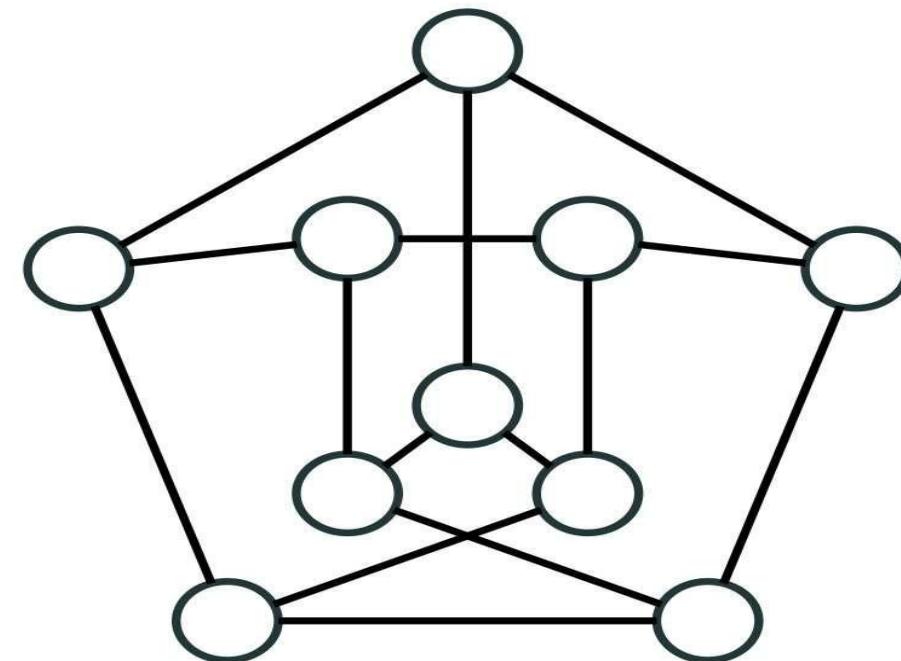
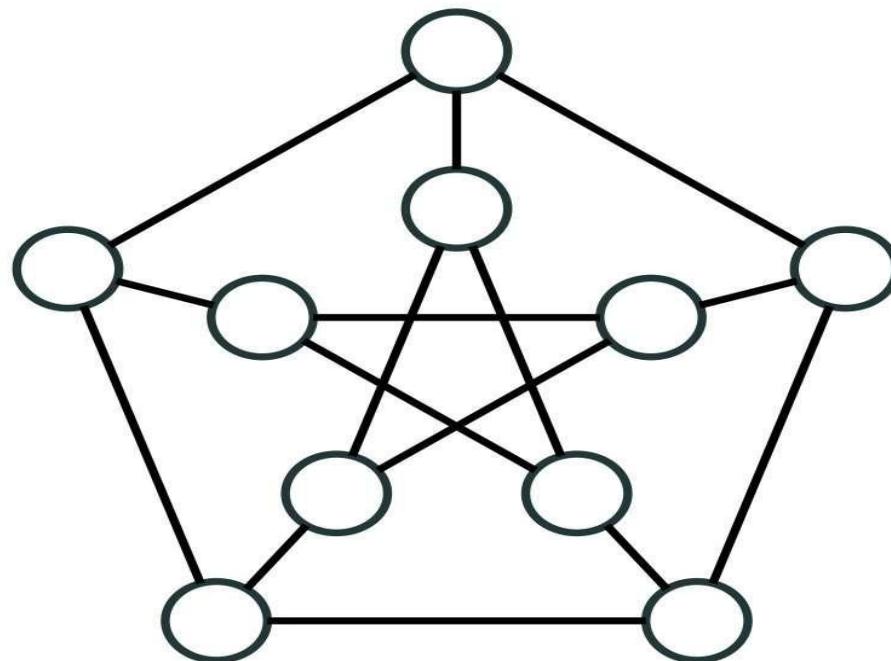
Many Ways to Draw a Graph

Are these graphs the same?



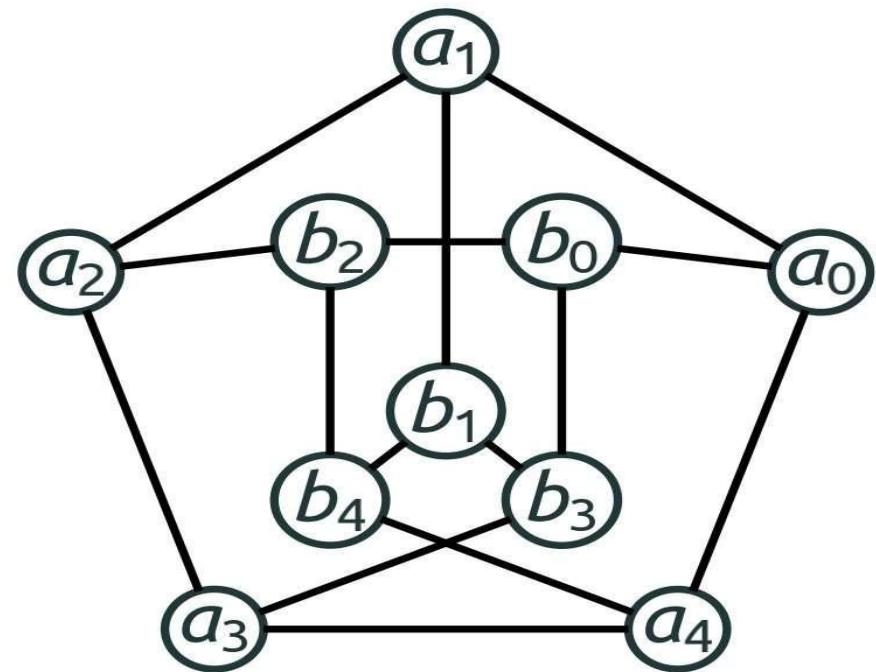
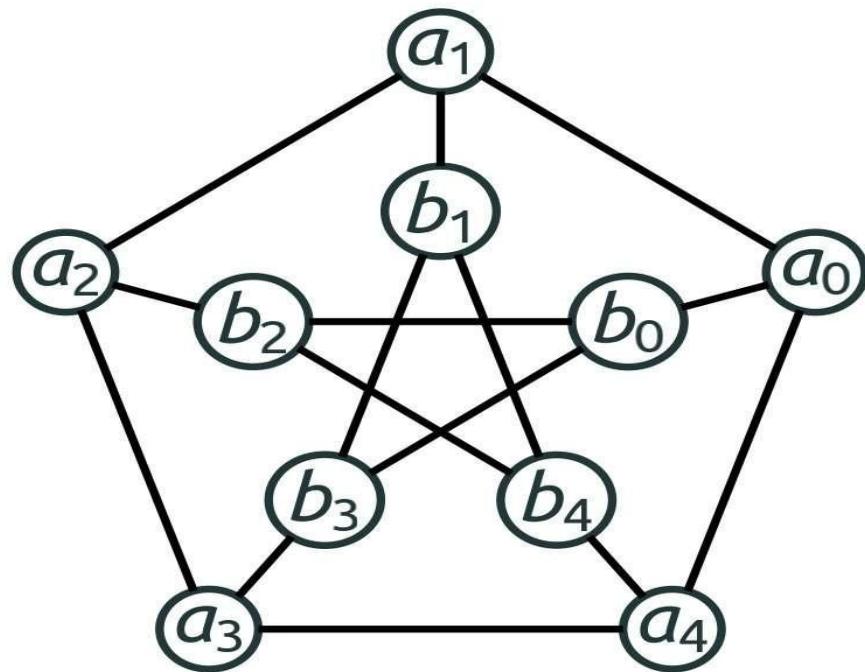
Many Ways to Draw a Graph

10 vertices and 15 edges?



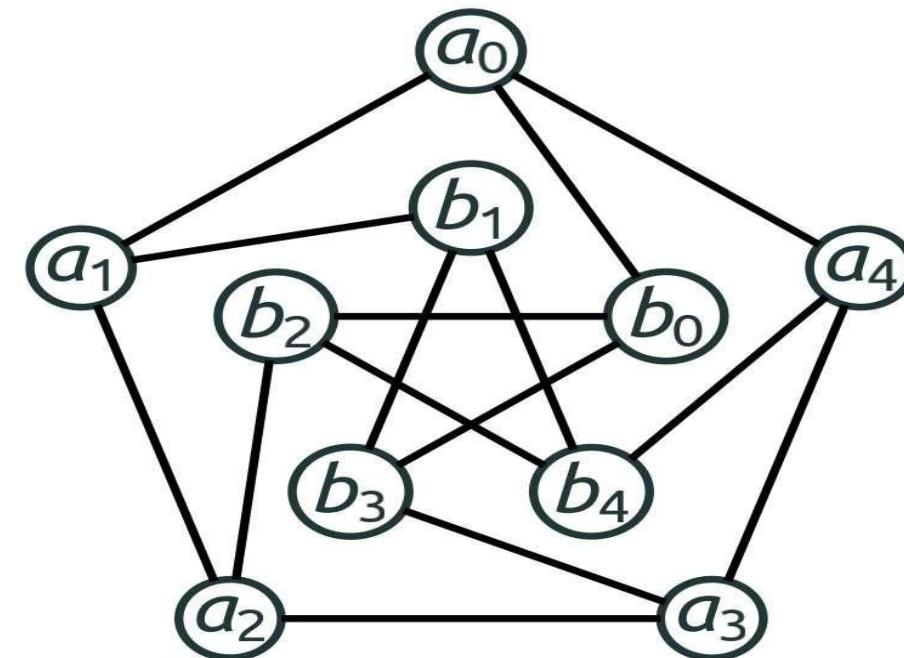
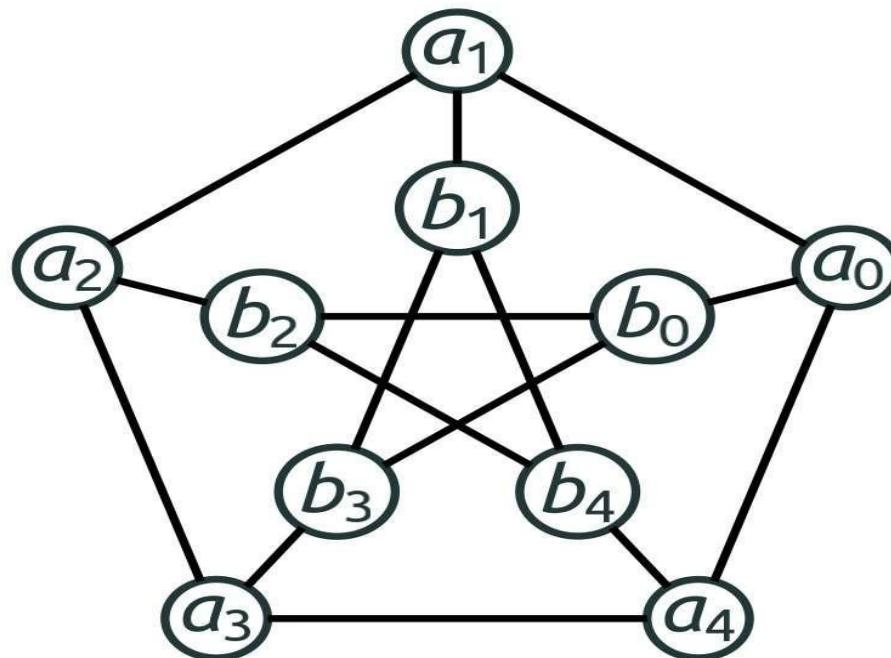
Many Ways to Draw a Graph

Are these graphs the same?



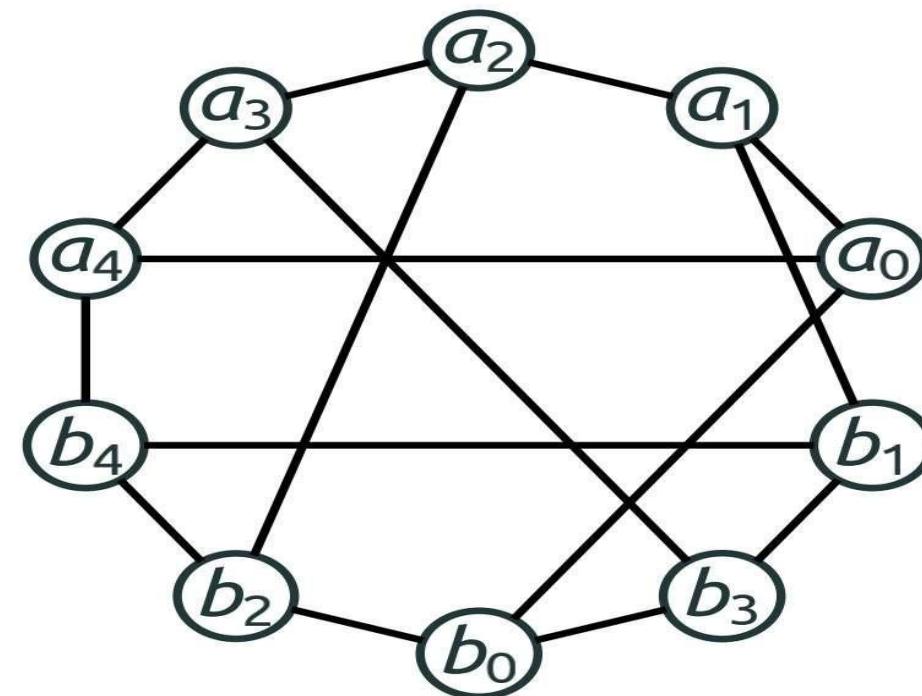
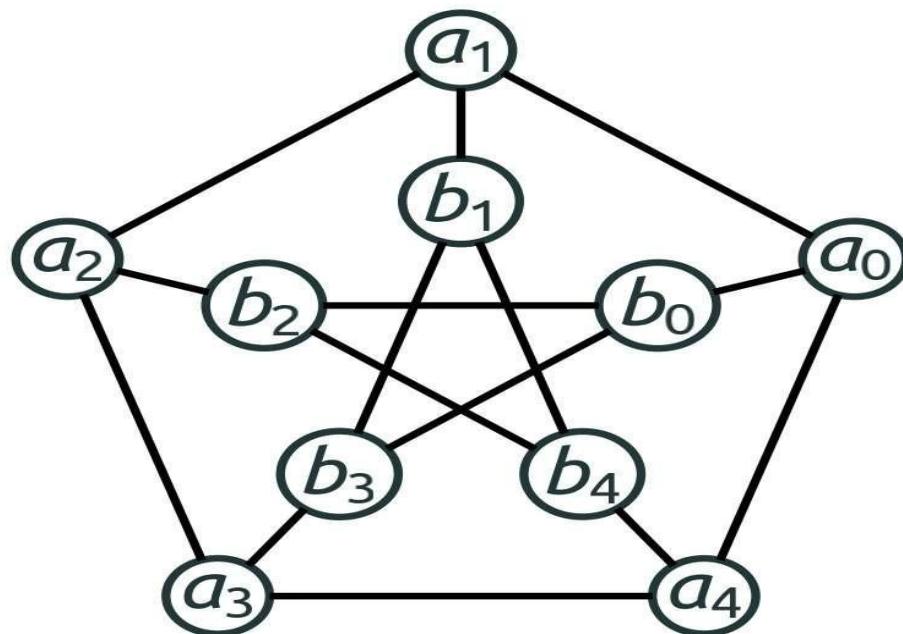
Many Ways to Draw a Graph

Are these graphs the same?

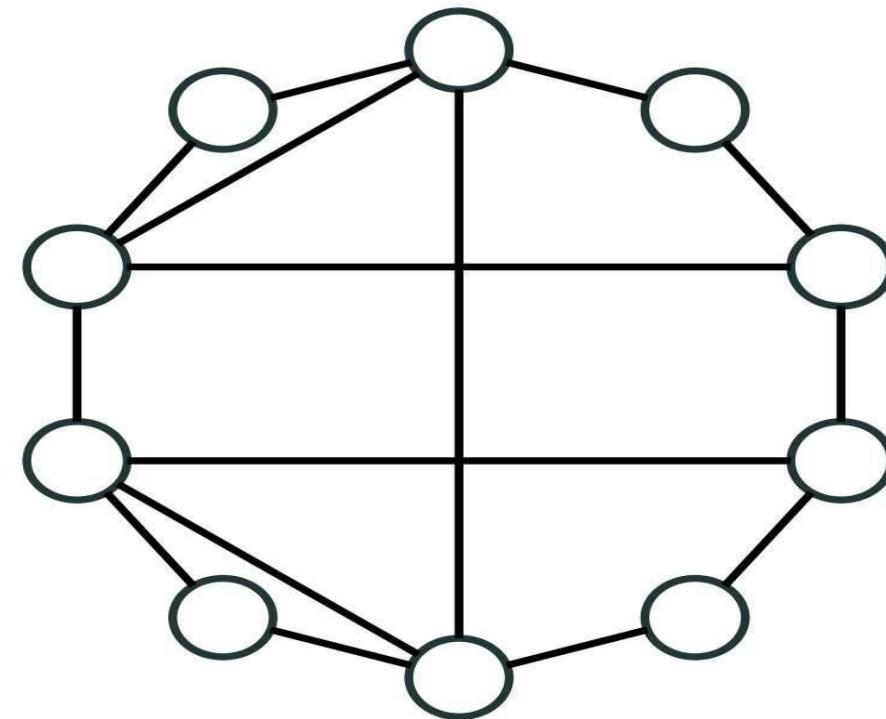
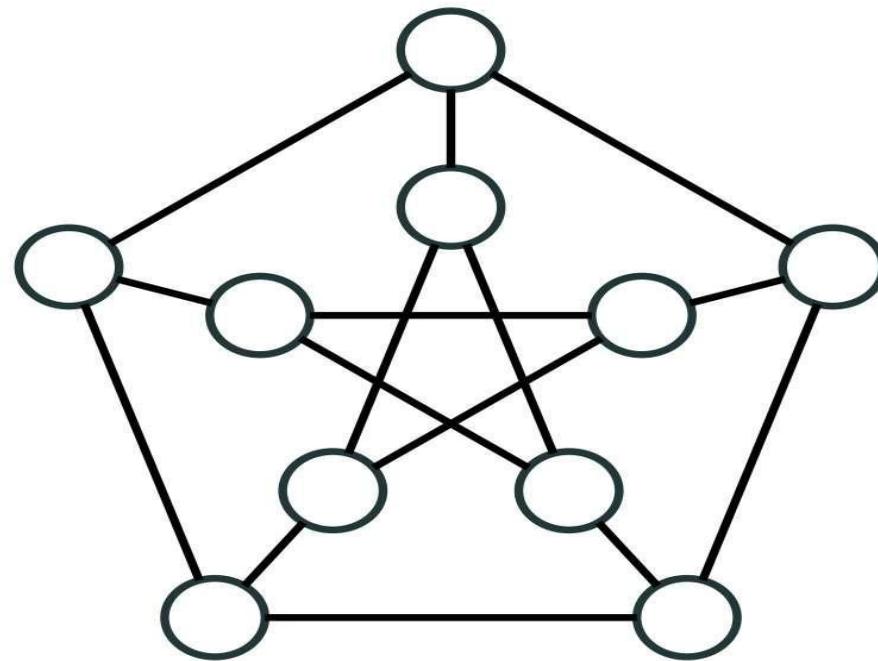


Many Ways to Draw a Graph

Are these graphs the same?

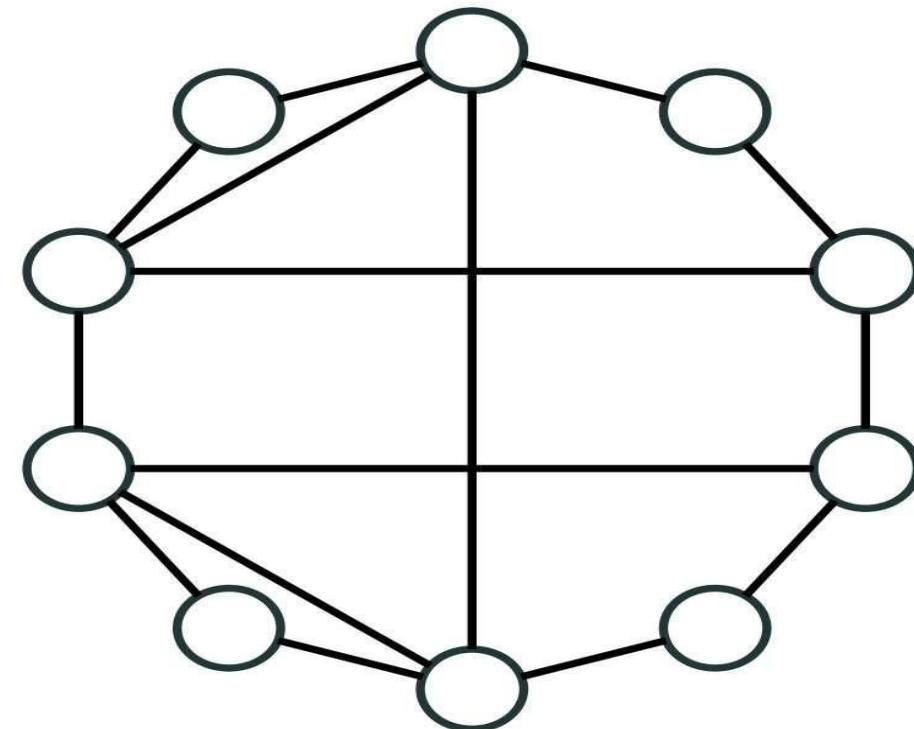
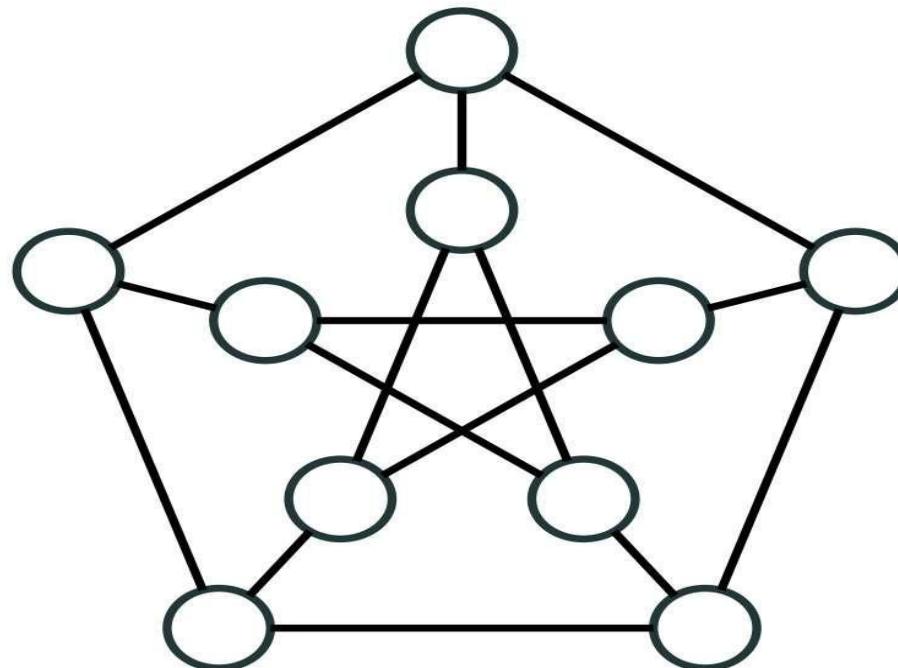


Are These Graphs the Same?



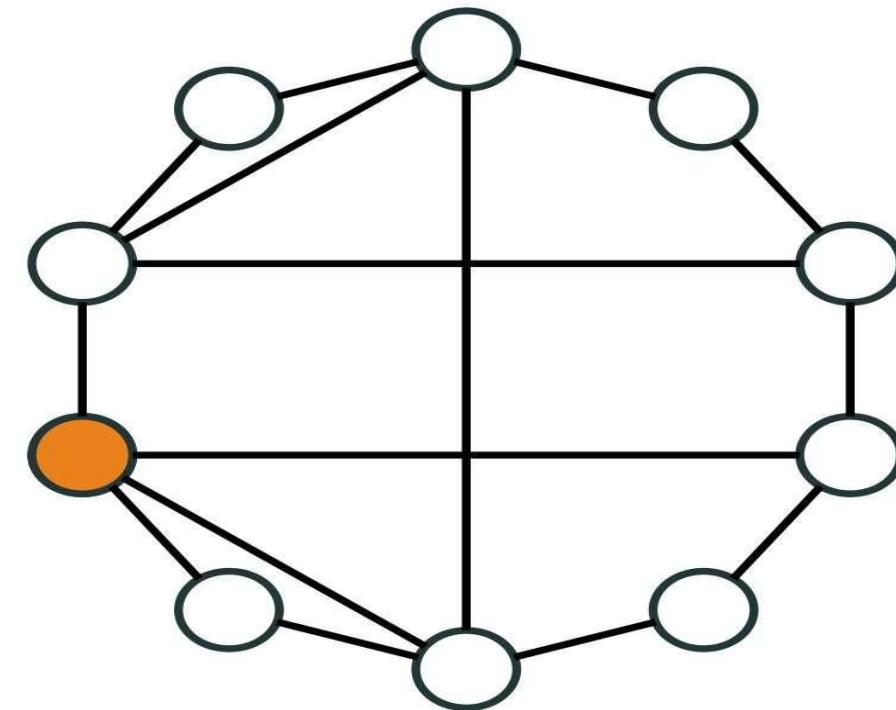
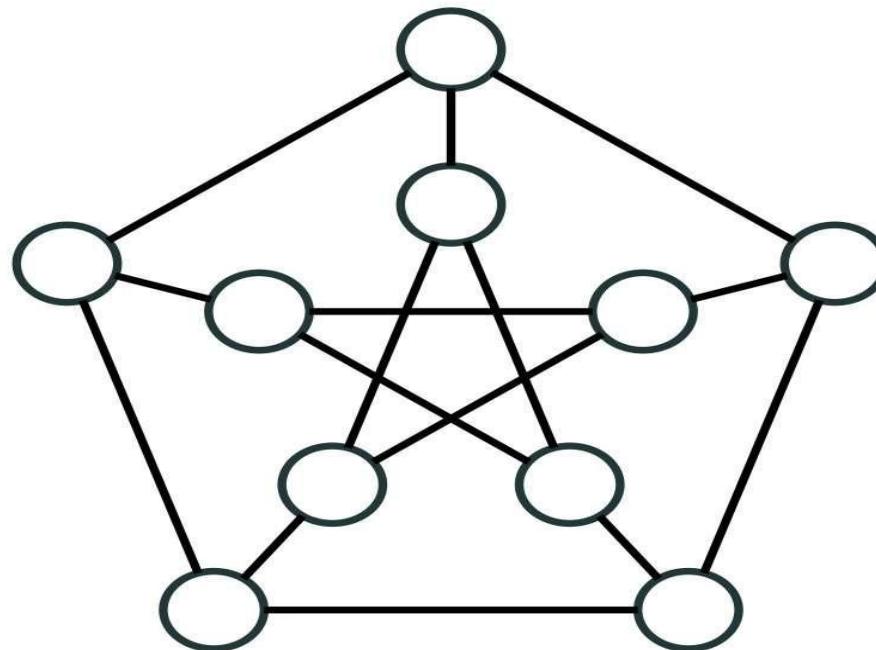
Are These Graphs the Same?

10 vertices and 15 edges?



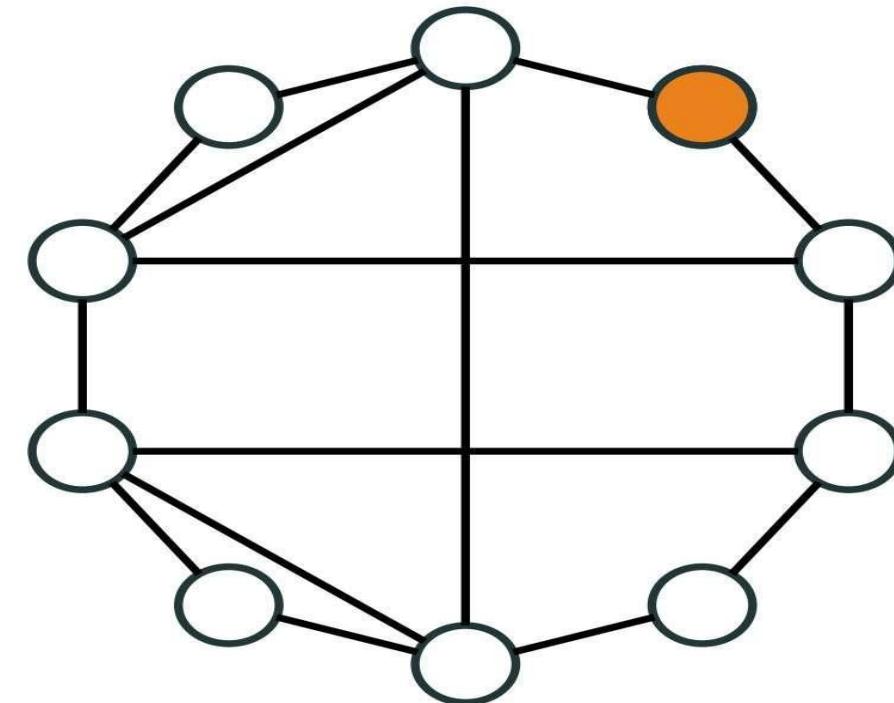
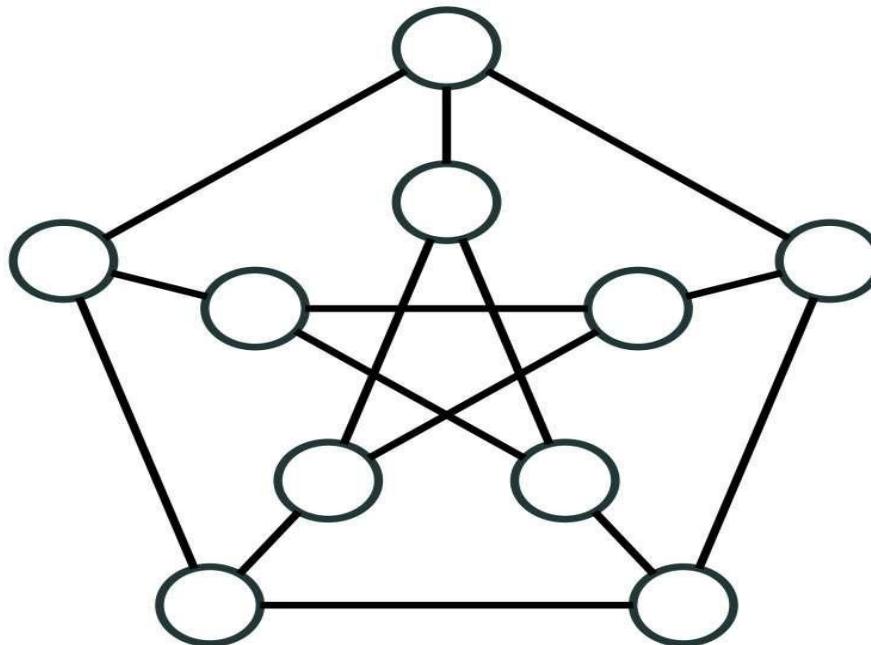
Are These Graphs the Same?

10 vertices and 15 edges?



Are These Graphs the Same?

10 vertices and 15 edges?



Graph Drawing is Beautiful!



Donald E. Knuth

*Graph drawing is the best possible field I can think of:
It merges aesthetics, mathematical beauty and wonderful algorithms.
It therefore provides a harmonic balance between the left and right brain parts.*

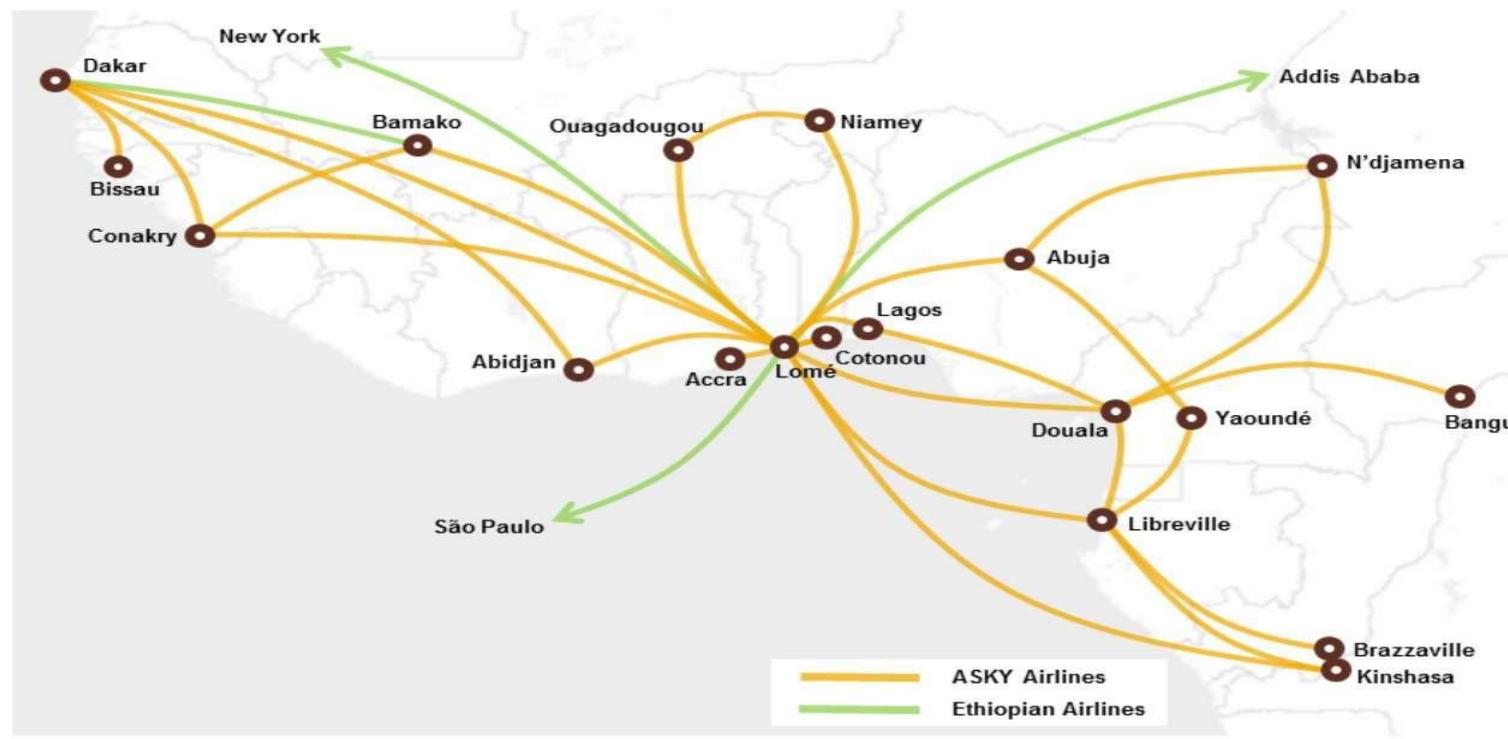
Outline

What is a Graph?

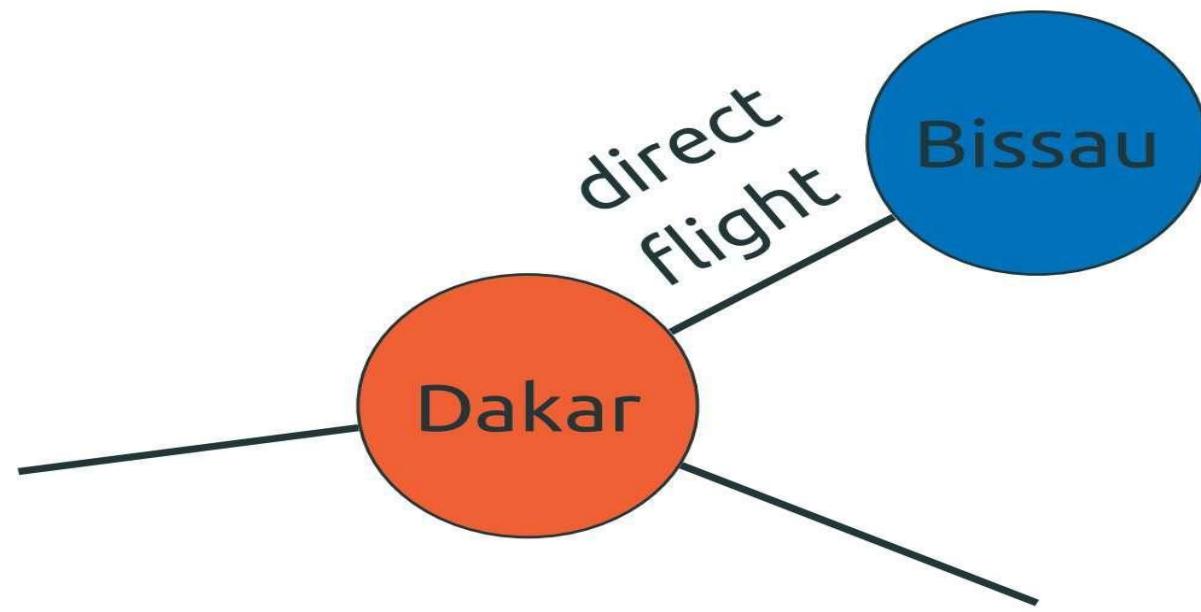
Graph Examples

Graph Applications

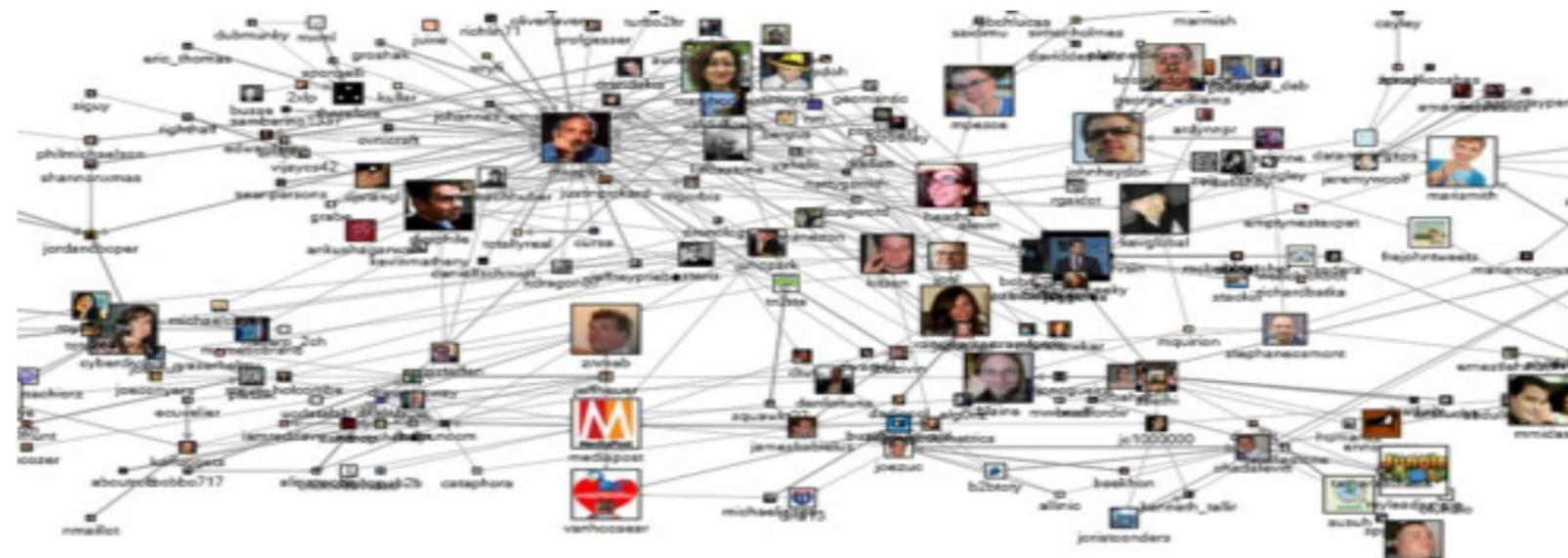
Airlines Graph



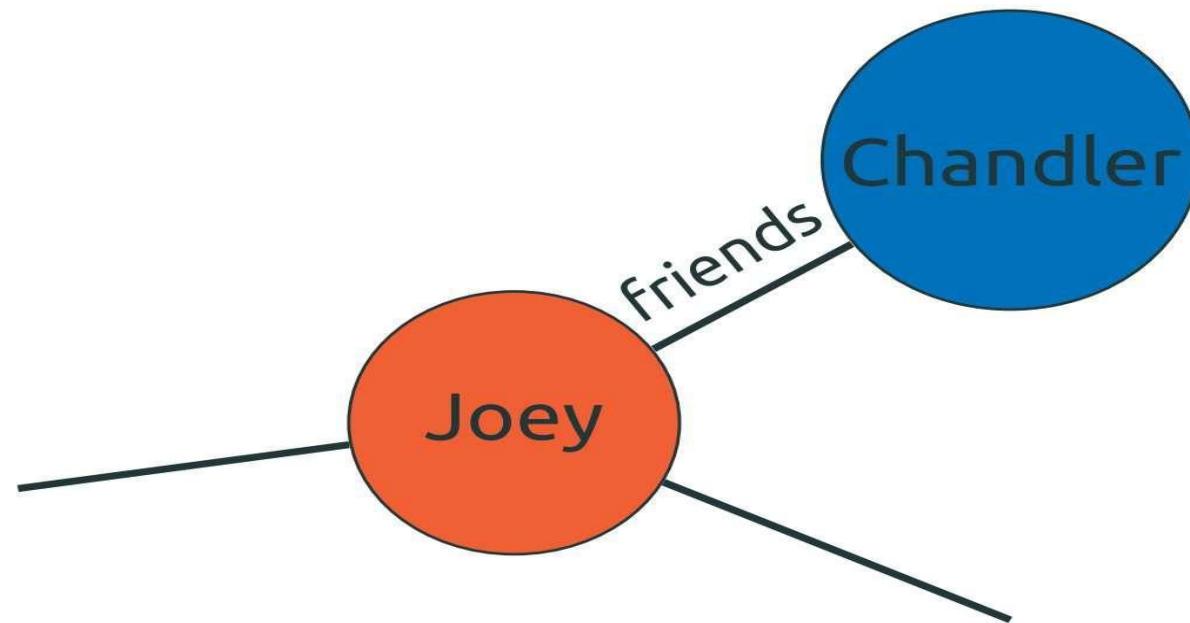
Airlines Graph



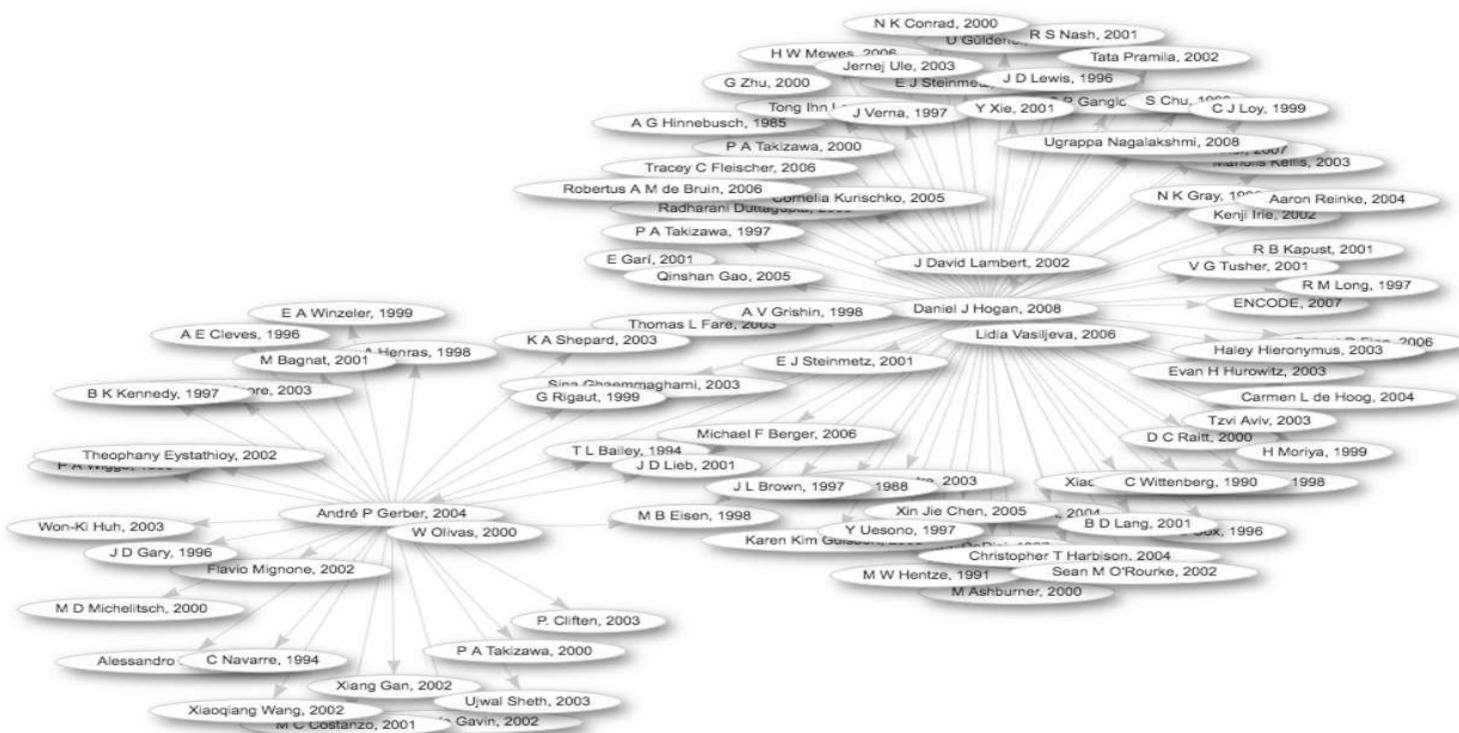
Facebook Graph



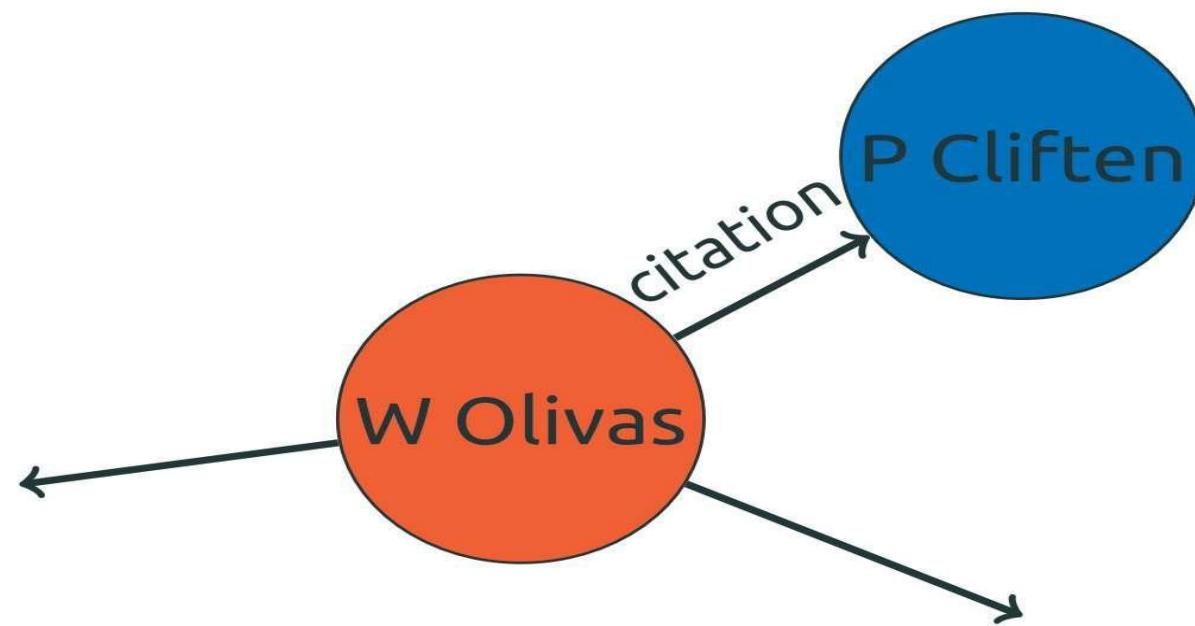
Facebook Graph



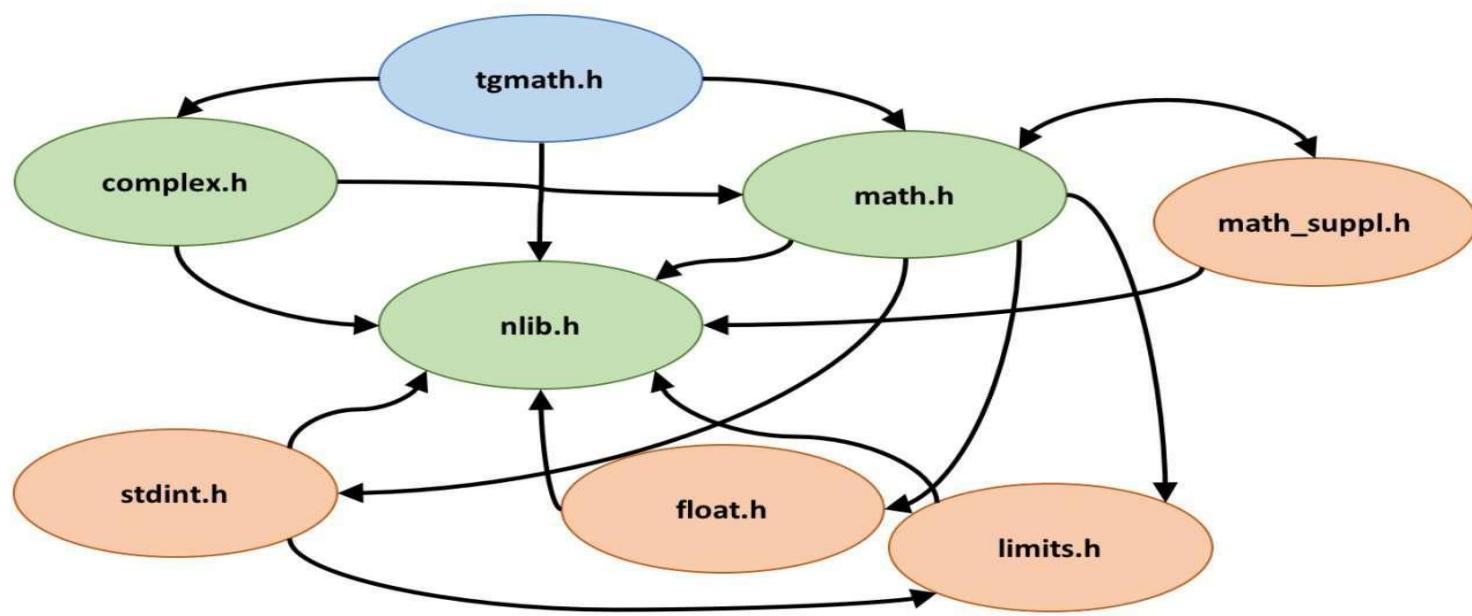
Citation Graph For a Paper



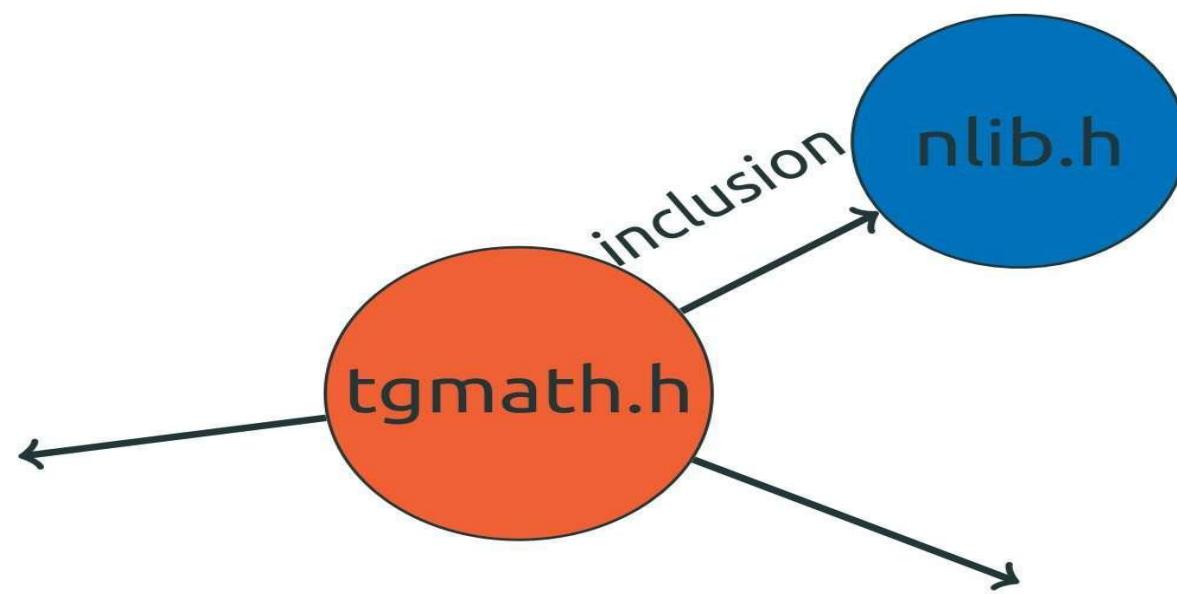
Citation Graph For a Paper



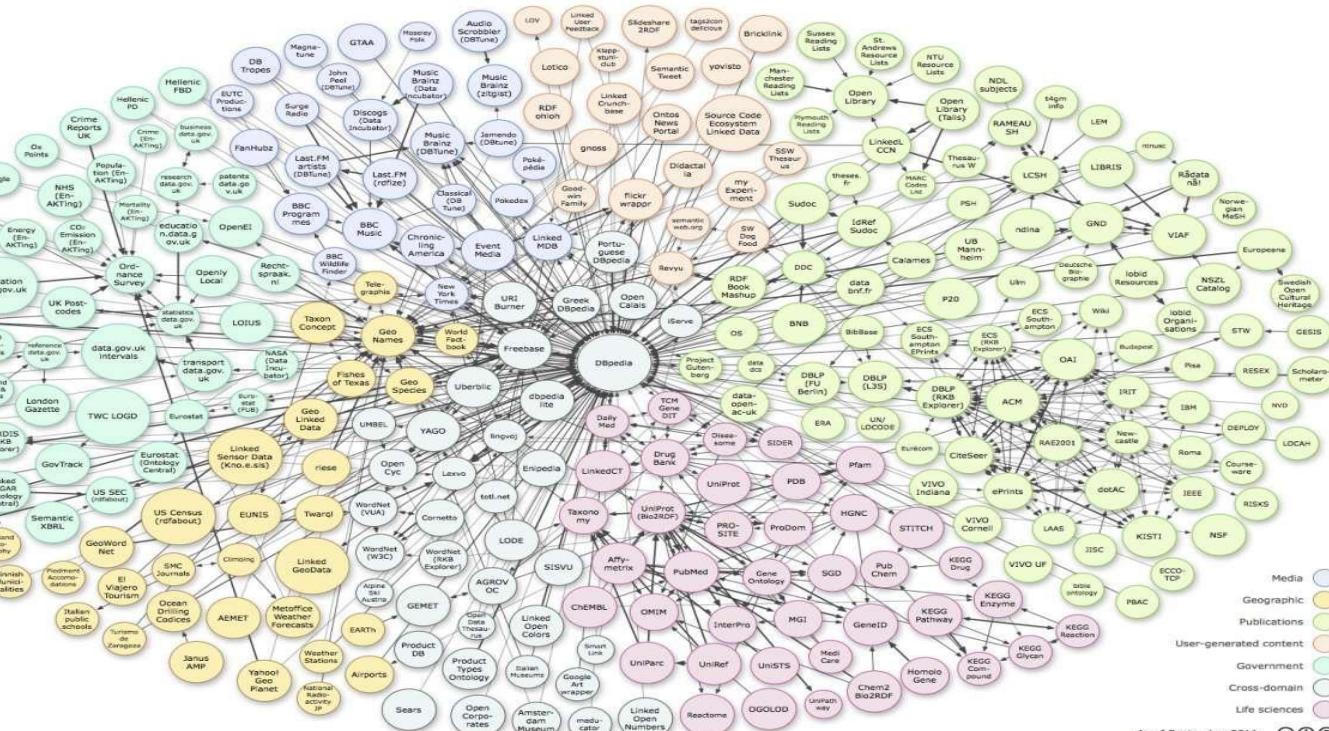
Dependency Graph



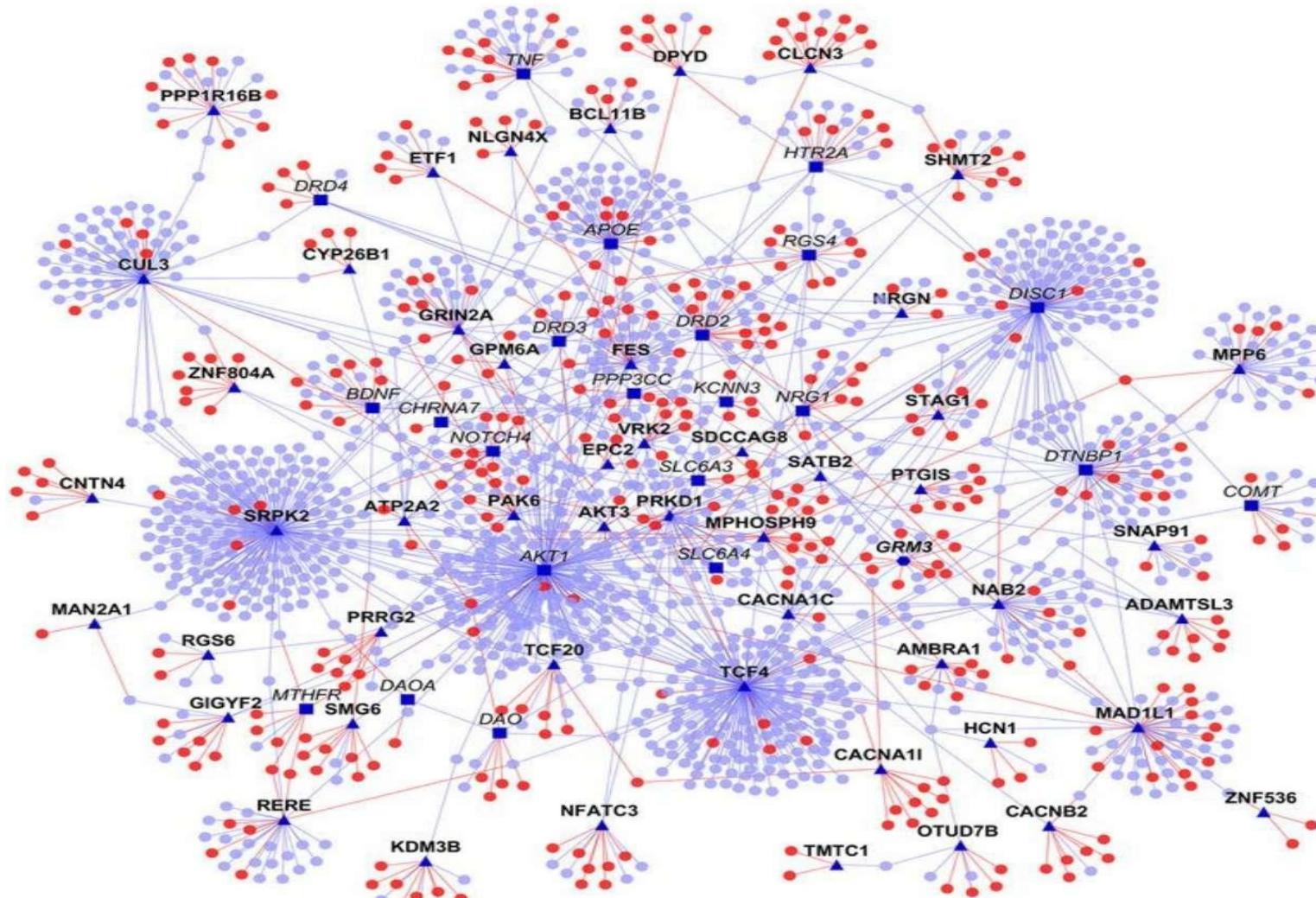
Dependency Graph



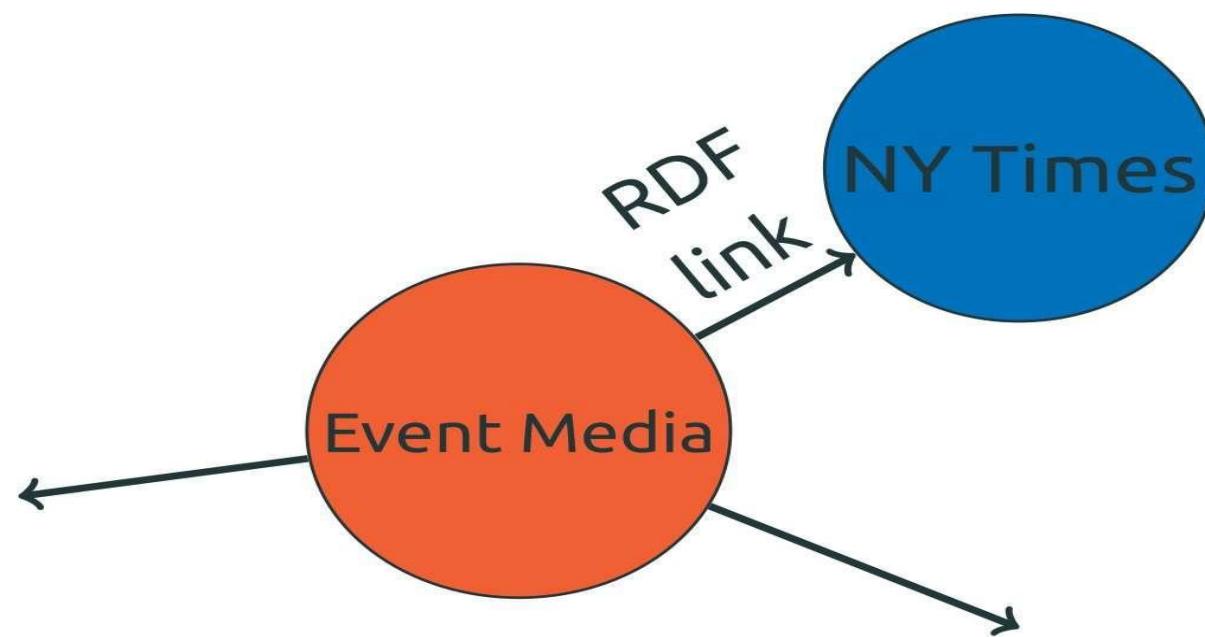
Linked Open Data Diagram



Schizophrenia Protein–Protein Interaction



Linked Open Data Diagram



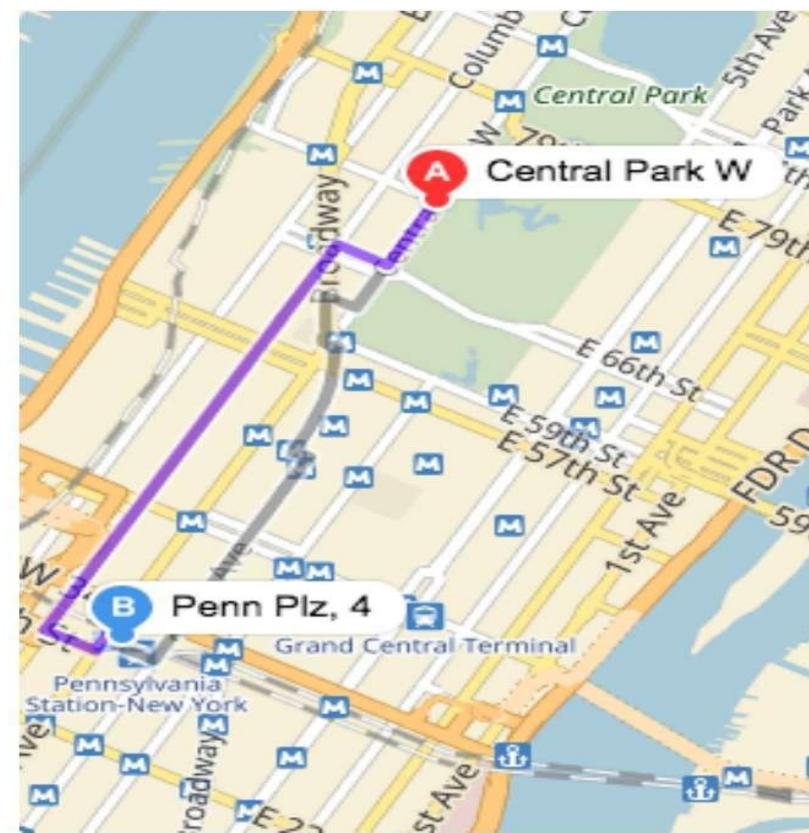
Outline

What is a Graph?

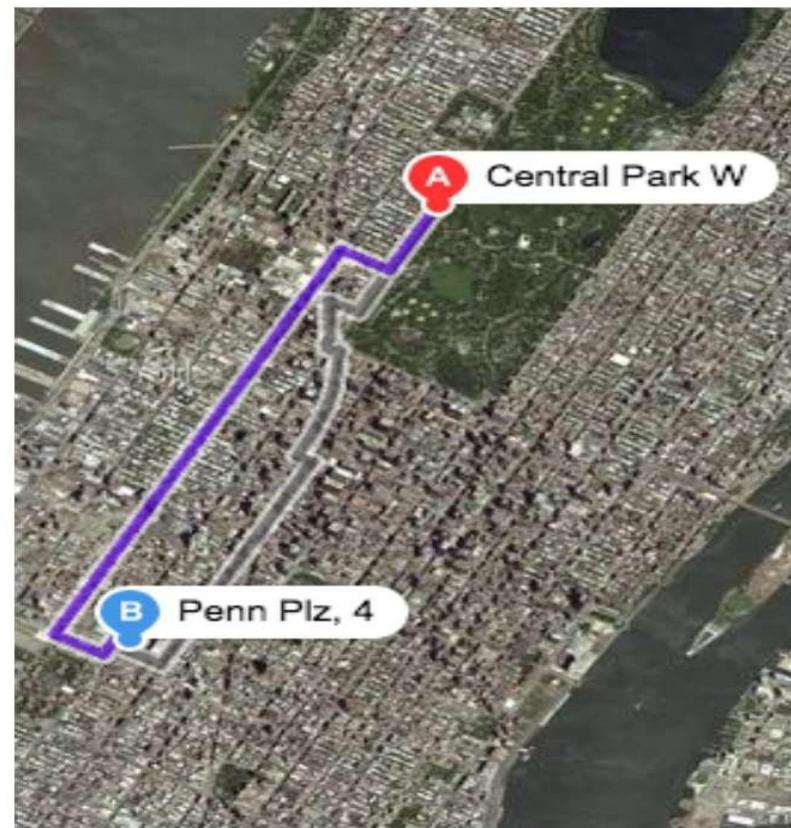
Graph Examples

Graph Applications

Navigation



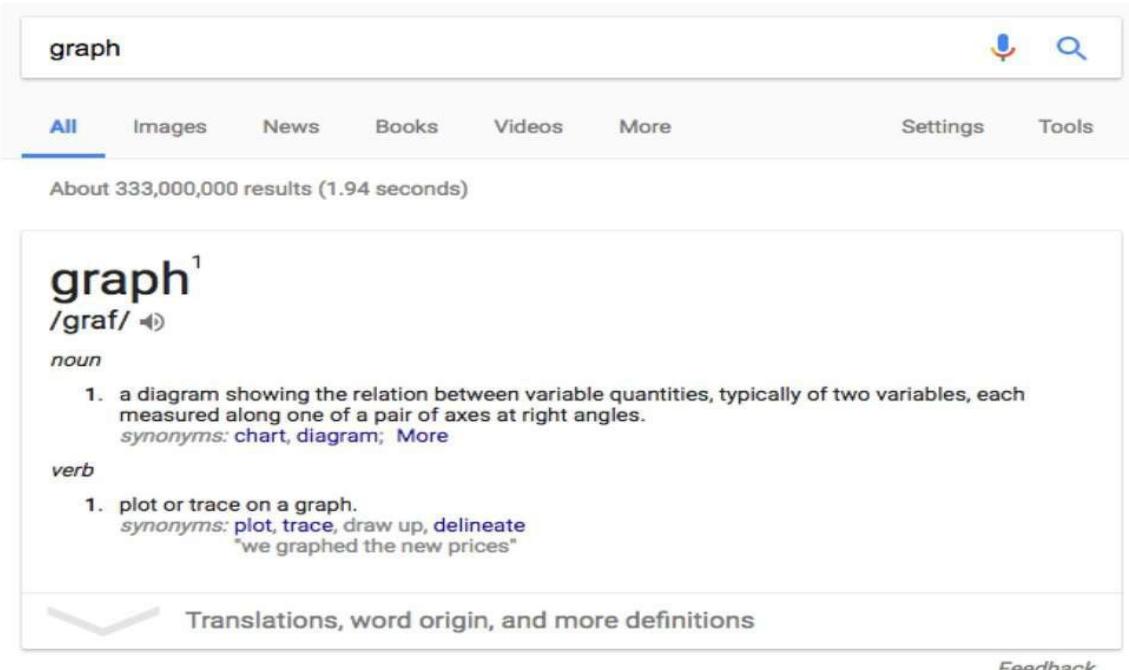
Navigation



PageRank



PageRank



A screenshot of a search results page from a search engine. The search term "graph" is entered in the search bar. Below the search bar, there are tabs for "All", "Images", "News", "Books", "Videos", "More", "Settings", and "Tools". A message indicates "About 333,000,000 results (1.94 seconds)". The main content area shows the first search result for "graph". The result title is "graph¹" with the pronunciation "/graf/". It is defined as a "noun" with one definition: "a diagram showing the relation between variable quantities, typically of two variables, each measured along one of a pair of axes at right angles." Below this, it is defined as a "verb" with one definition: "plot or trace on a graph." There are also "synonyms": "chart, diagram; More" and an example sentence: "we graphed the new prices". At the bottom of the result, there is a link to "Translations, word origin, and more definitions" and a "Feedback" button.

Graph - Wikipedia

<https://en.wikipedia.org/wiki/Graph> ▾

Graph (topology), a topological space resembling a graph in the sense of discrete mathematics. Graph of a function. Chart, a means of representing data (also called a graph).

Graph of a function - Wikipedia

https://en.wikipedia.org/wiki/Graph_of_a_function ▾

PageRank

Graph TV

graphtv.kevininformatics.com/

Graph Ratings of Your Favorite TV Shows. Visualize IMDb ratings and trends of TV shows by episode. Have you seen *Mad Men*, *Breaking Bad*, or *Battlestar* ...

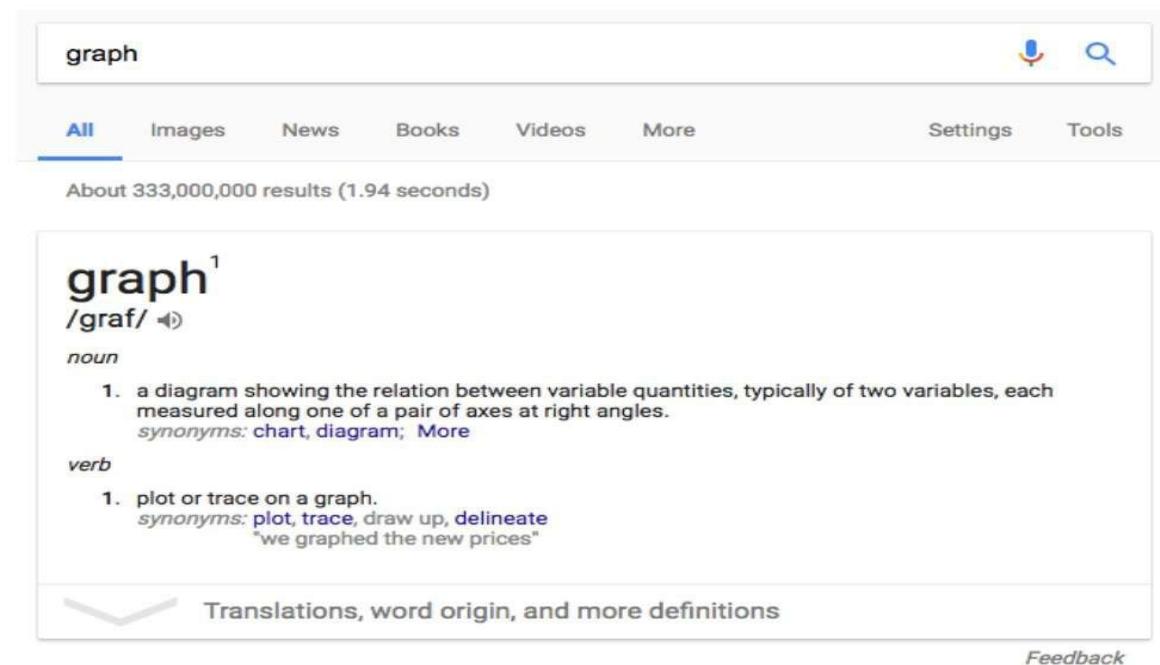
< Goooooooooooooogle >

Previous

1 2 3 4 5 6 7 8 9 10

Next

PageRank



A screenshot of a search results page from a search engine. The search bar at the top contains the word "graph". Below the search bar, there are tabs for "All", "Images", "News", "Books", "Videos", "More", "Settings", and "Tools". A message indicates "About 333,000,000 results (1.94 seconds)". The main content area shows the definition of "graph".
graph¹
/graf/ ⓘ
noun
1. a diagram showing the relation between variable quantities, typically of two variables, each measured along one of a pair of axes at right angles.
synonyms: chart, diagram; [More](#)
verb
1. plot or trace on a graph.
synonyms: plot, trace, draw up, delineate
"we graphed the new prices"
A "Translations, word origin, and more definitions" button is visible below the main content, and a "Feedback" link is at the bottom right.

Graph - Wikipedia

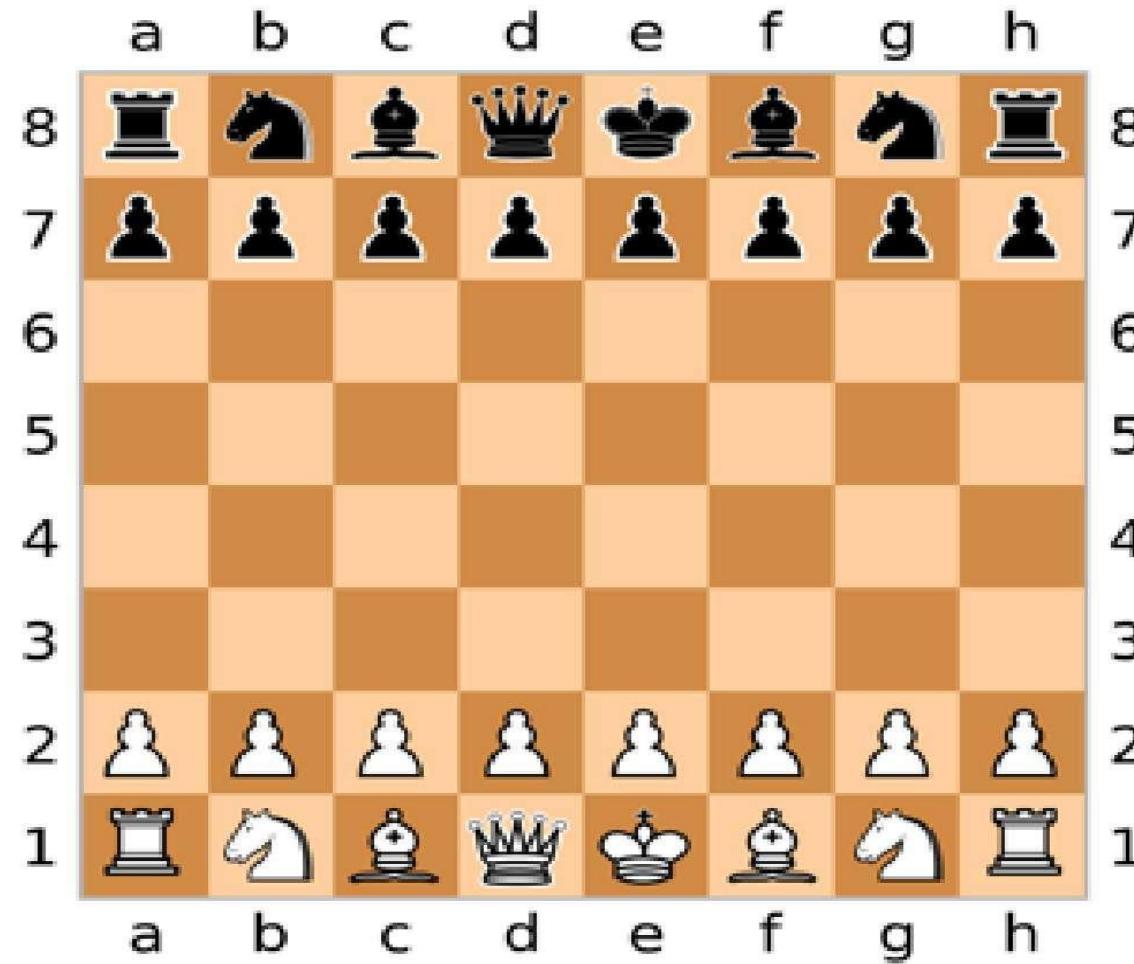
<https://en.wikipedia.org/wiki/Graph> ▾

Graph (topology), a topological space resembling a graph in the sense of discrete mathematics. Graph of a function. Chart, a means of representing data (also called a graph).

Graph of a function - Wikipedia

https://en.wikipedia.org/wiki/Graph_of_a_function ▾

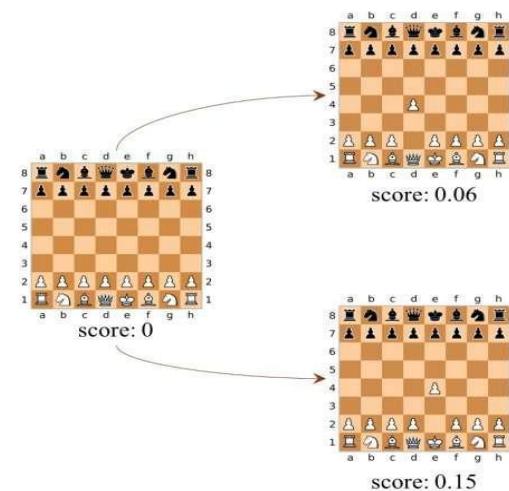
Game Strategies



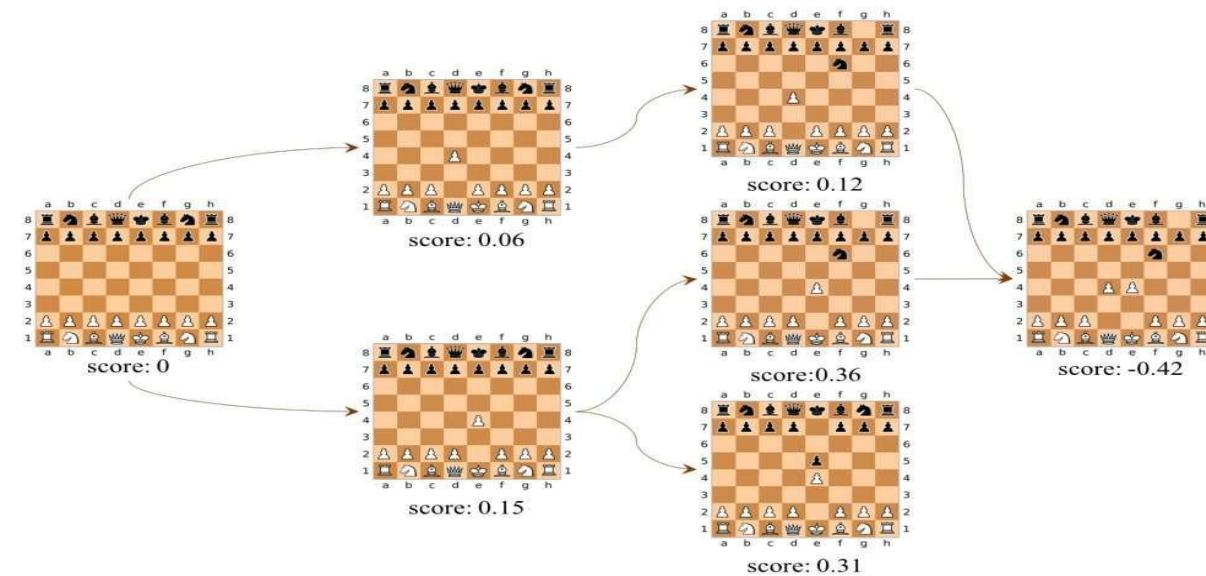
Game Strategies



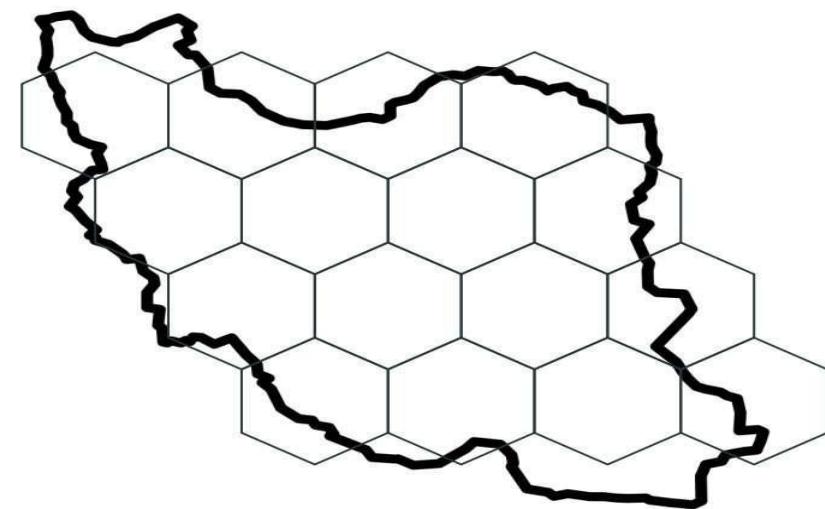
Game Strategies



Game Strategies

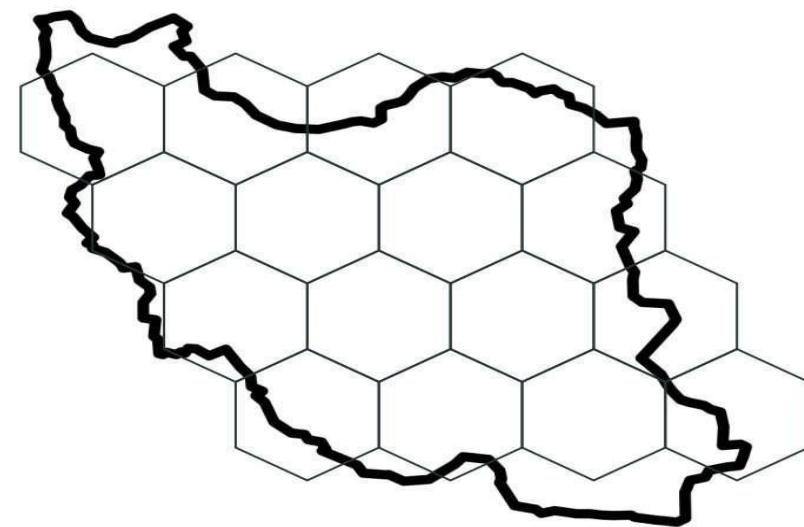


GSM



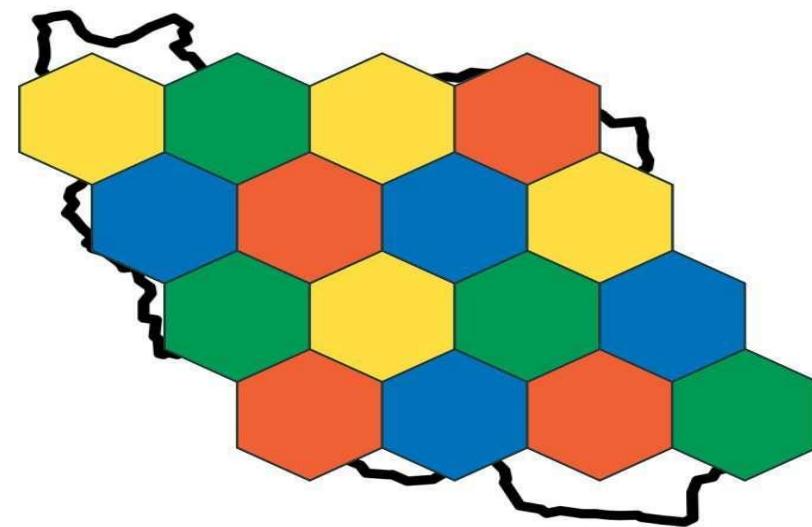
GSM

4 Frequency
Ranges

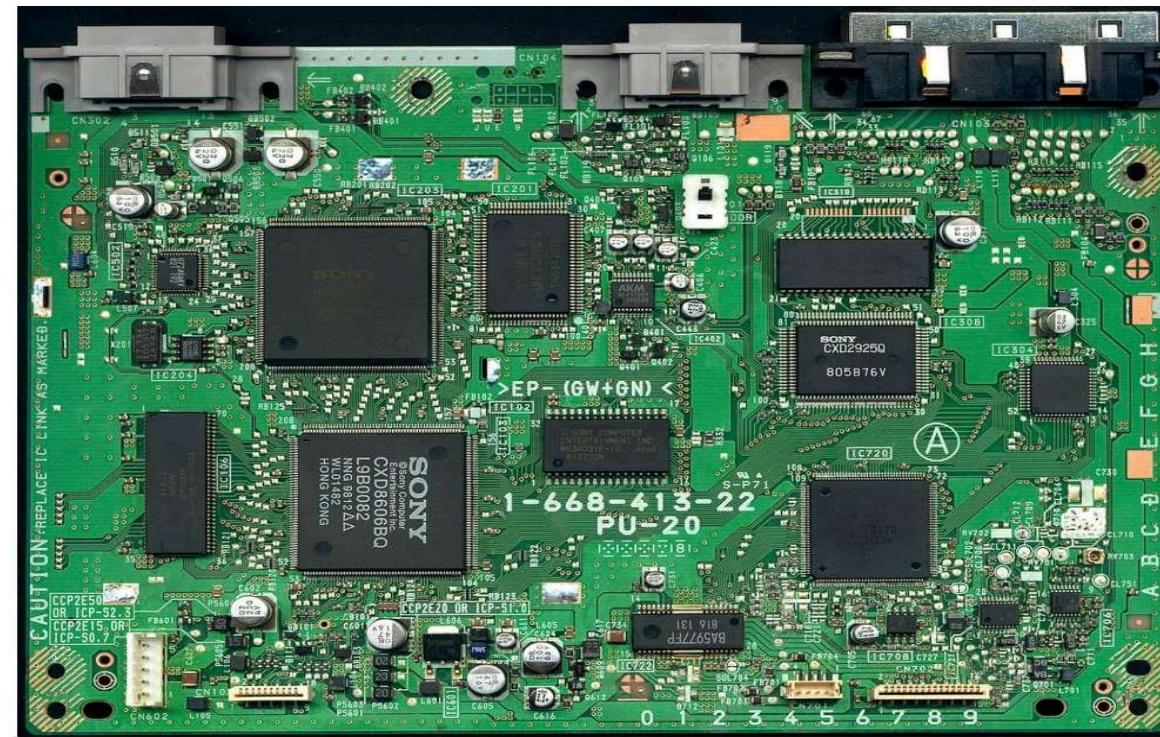


GSM

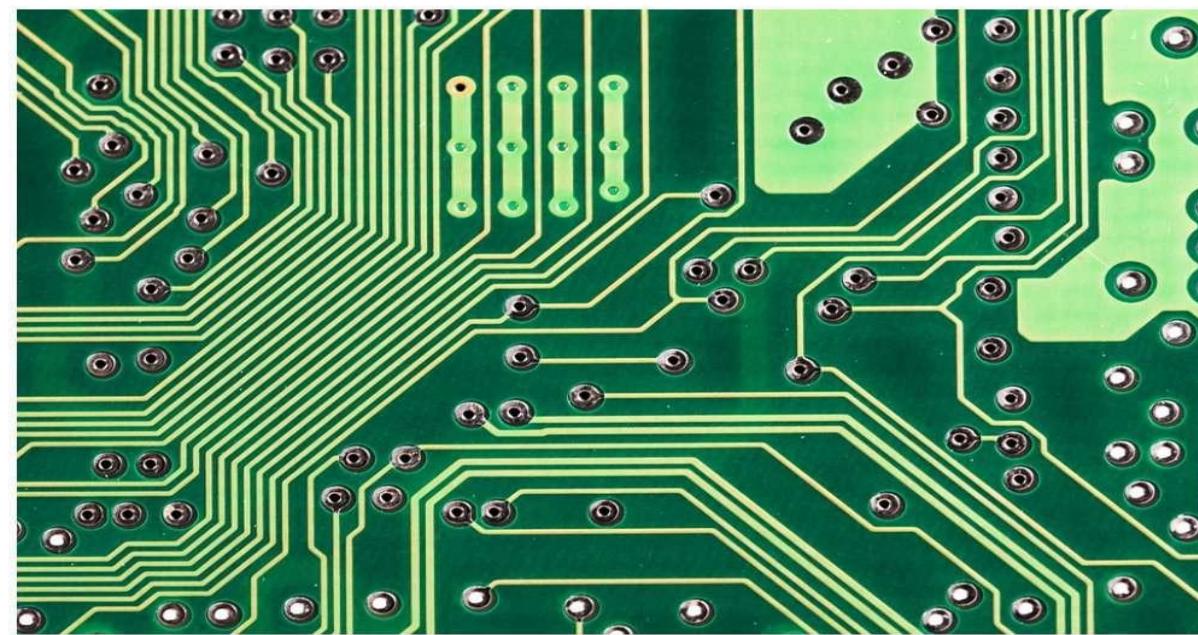
4 Frequency
Ranges



Computer Chips



Computer Chips



LECTURE # 02

Some Basic Definitions

The Degree of a Vertex

The number of friends



The Degree of a Vertex

- The **Degree** of a vertex is the number of its incident edges

The Degree of a Vertex

- The **Degree** of a vertex is the number of its incident edges
- I.e., the **Degree** of a vertex is the number of its neighbors

The Degree of a Vertex

- The **Degree** of a vertex is the number of its incident edges
- I.e., the **Degree** of a vertex is the number of its neighbors
- The degree of a vertex v is denoted by $\deg(v)$

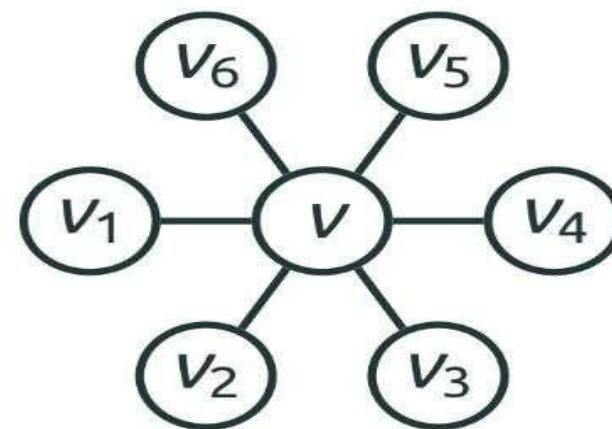
The Degree of a Vertex

- The **Degree** of a vertex is the number of its incident edges
- I.e., the **Degree** of a vertex is the number of its neighbors
- The degree of a vertex v is denoted by $\deg(v)$
- The **degree of a graph** is the maximum degree of its vertices

The Degree of a Vertex: Examples

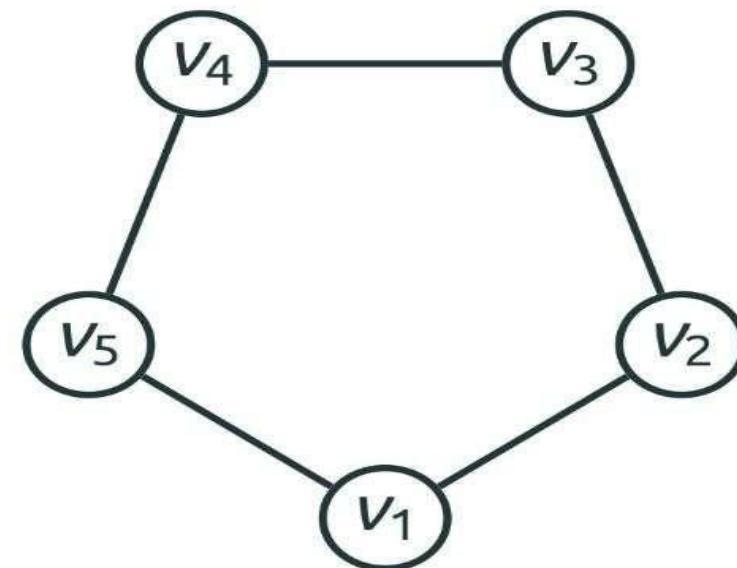
The degree of v is 6: $\deg(v) = 6$

The degree of v_6 is 1: $\deg(v_6) = 1$

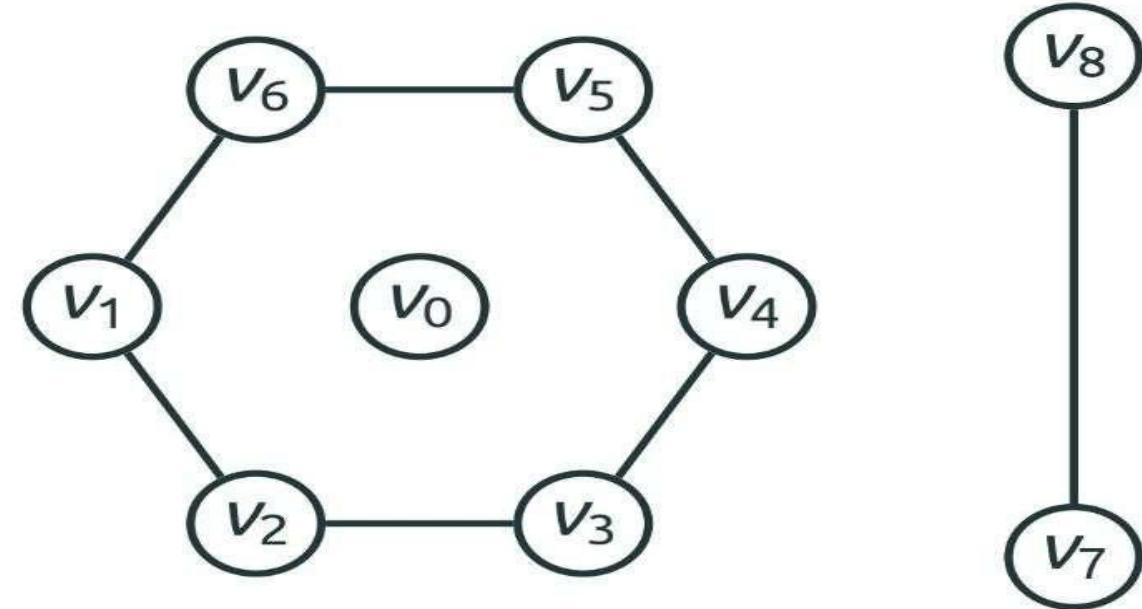


The Degree of a Vertex: Examples

The degree of every vertex is 2: $\forall i, \deg(v_i) = 2$

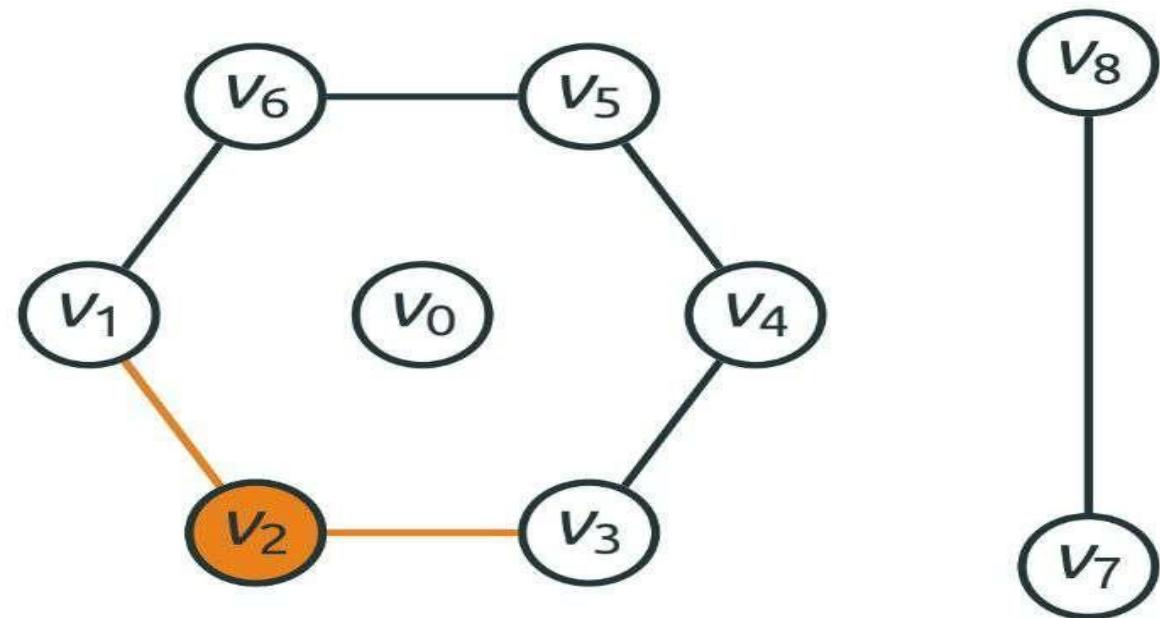


Isolated Vertices



Isolated Vertices

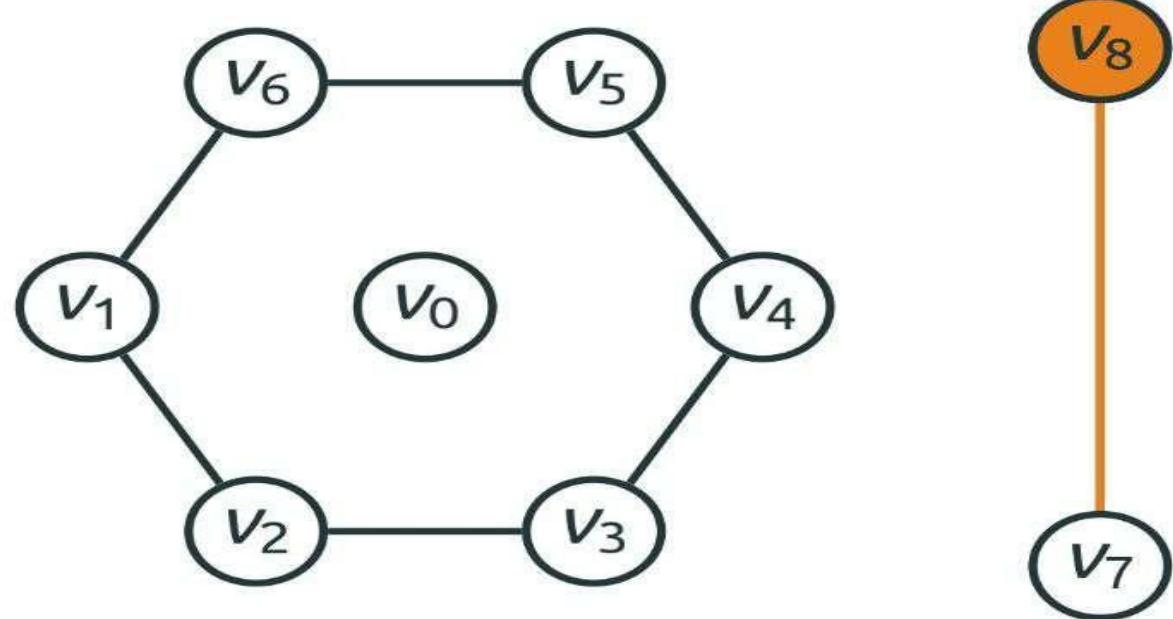
$$\deg(v_2) = 2$$



Isolated Vertices

$$\deg(v_2) = 2$$

$$\deg(v_8) = 1$$

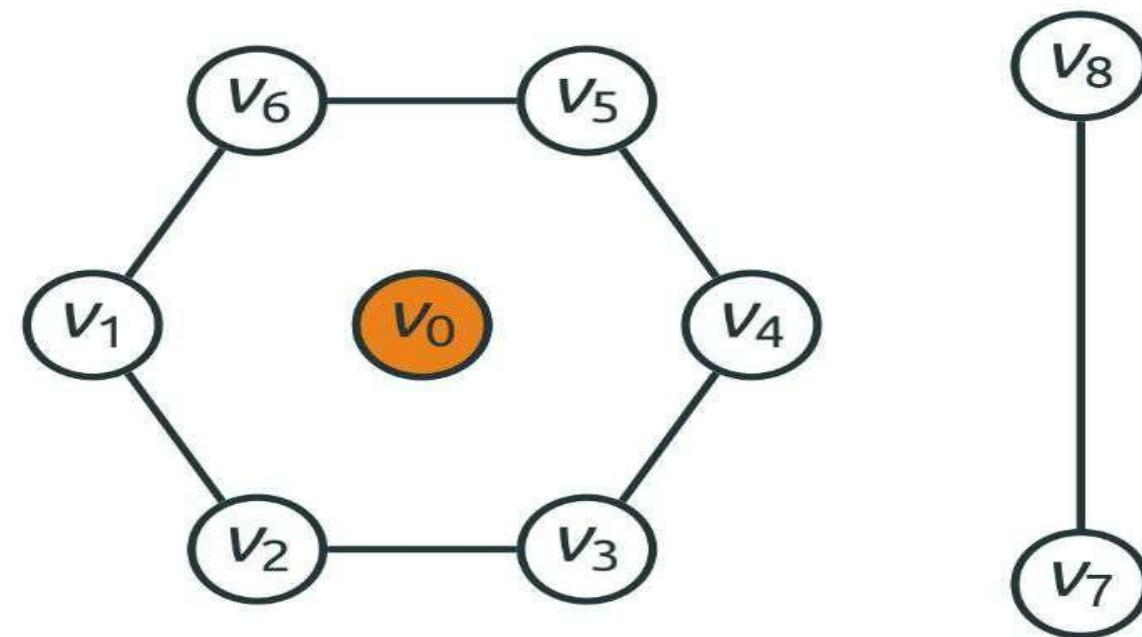


Isolated Vertices

$$\deg(v_2) = 2$$

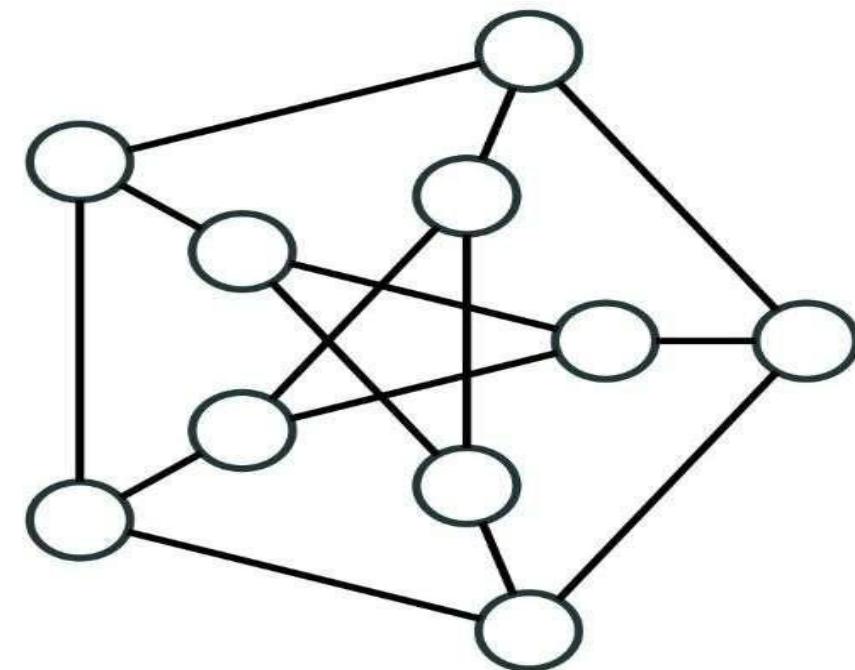
$$\deg(v_8) = 1$$

$\deg(v_0) = 0$. v_0 is an Isolated Vertex



Regular Graphs

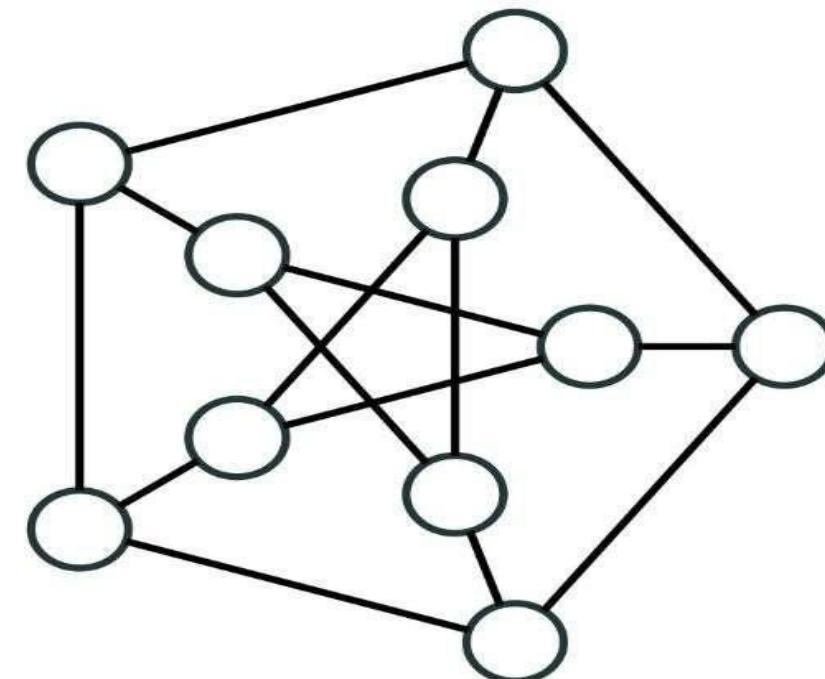
A **Regular** graph is a graph where each vertex has the same degree



Regular Graphs

A **Regular** graph is a graph where each vertex has the same degree

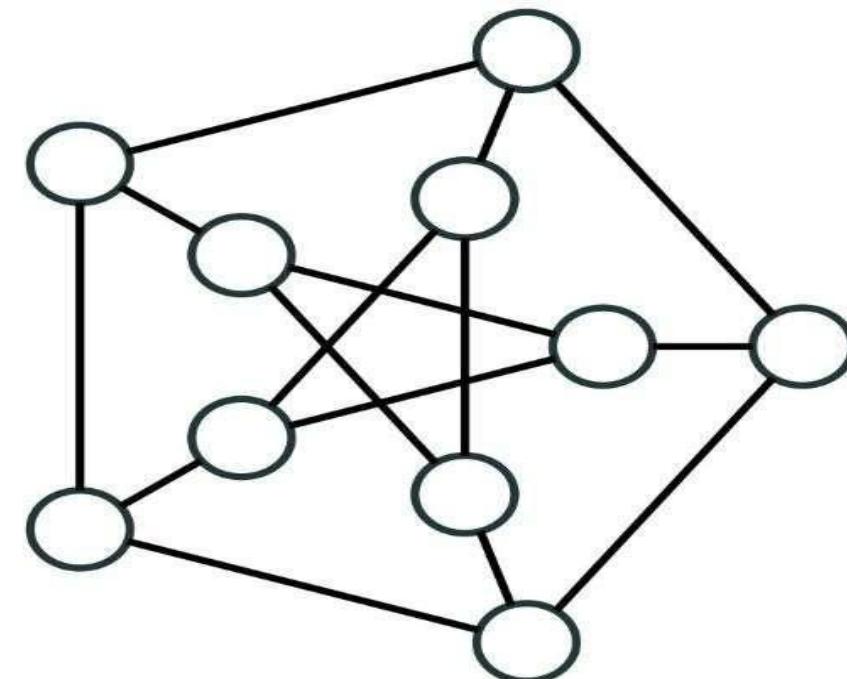
A regular graph of degree k is also called **k -Regular**



Regular Graphs

A **Regular** graph is a graph where each vertex has the same degree

A regular graph of degree k is also called **k -Regular**
E.g., this graph is **3-Regular**



Complement Graph

- The **Complement** of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ on the same set of vertices V and the following set of edges:

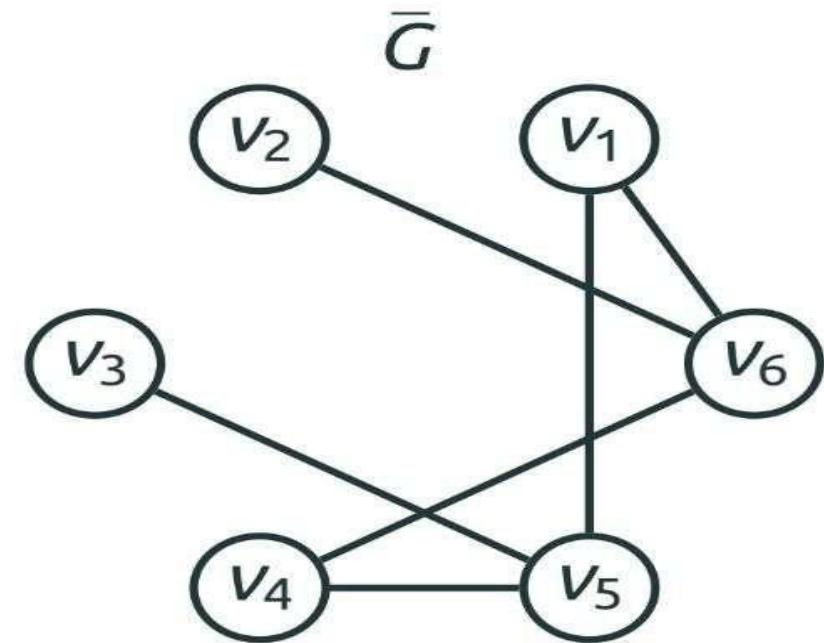
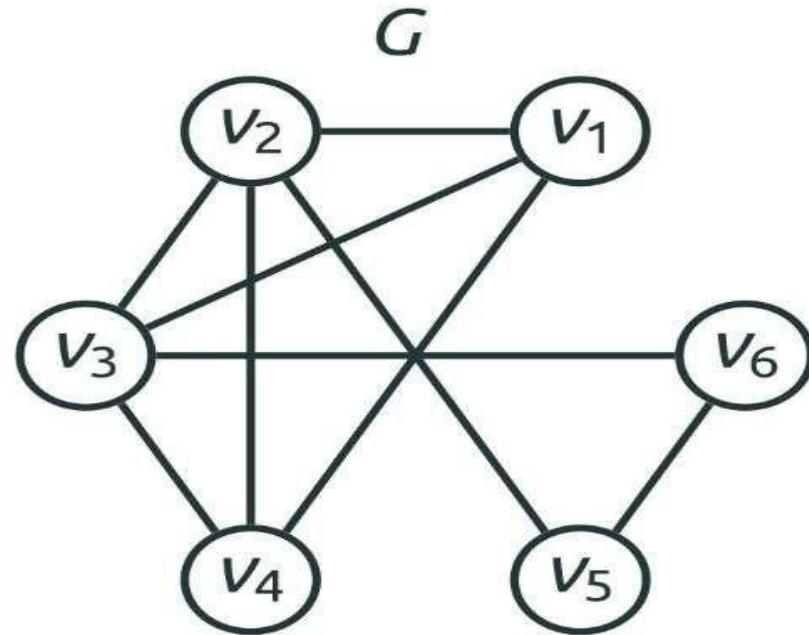
Complement Graph

- The **Complement** of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ on the same set of vertices V and the following set of edges:
- Two vertices are connected in \bar{G} **if and only if** they are not connected in G

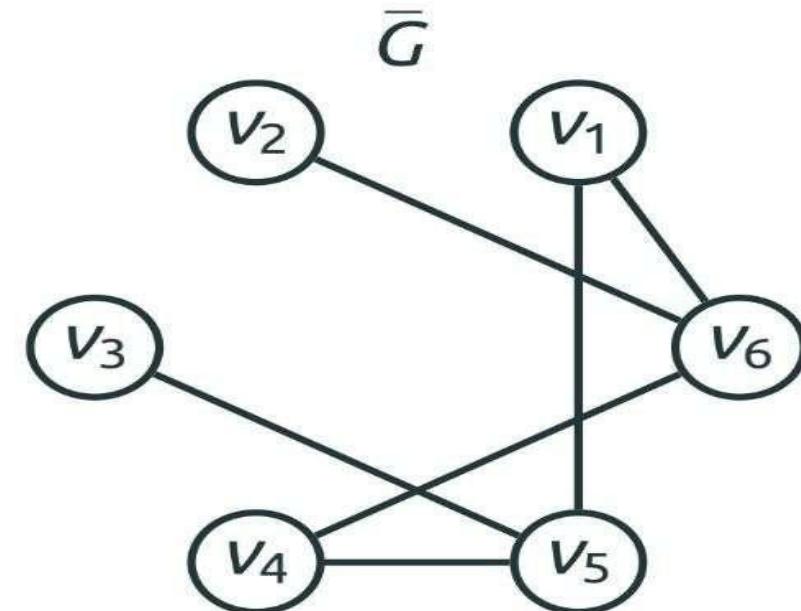
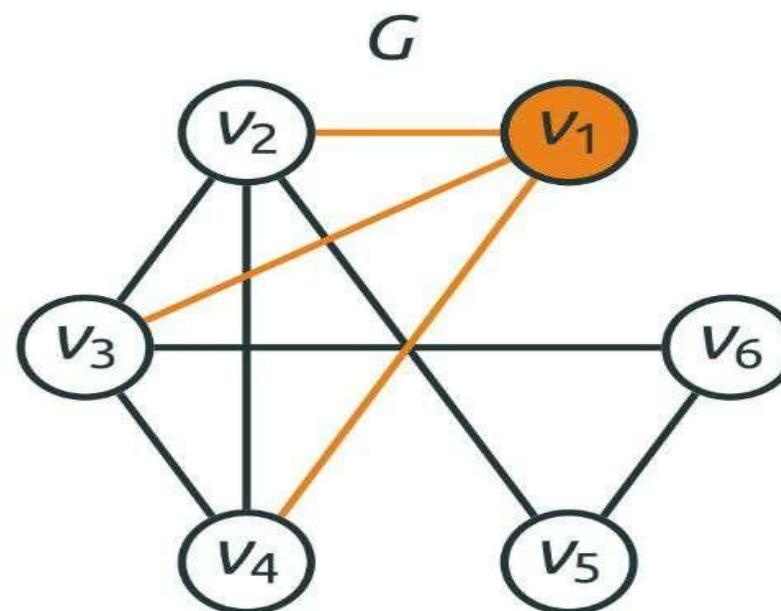
Complement Graph

- The **Complement** of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ on the same set of vertices V and the following set of edges:
- Two vertices are connected in \bar{G} **if and only if** they are not connected in G
- I.e., $(u, v) \in \bar{E}$ **if and only if** $(u, v) \notin E$

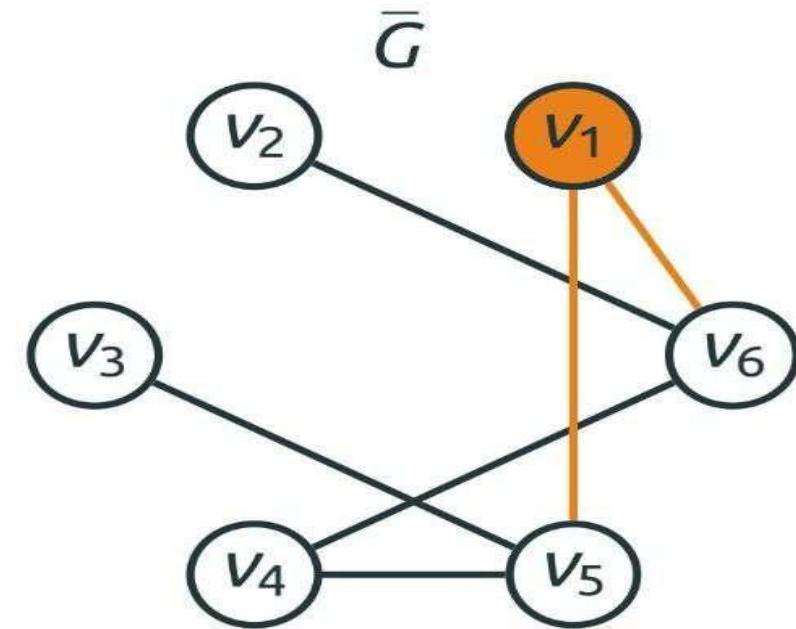
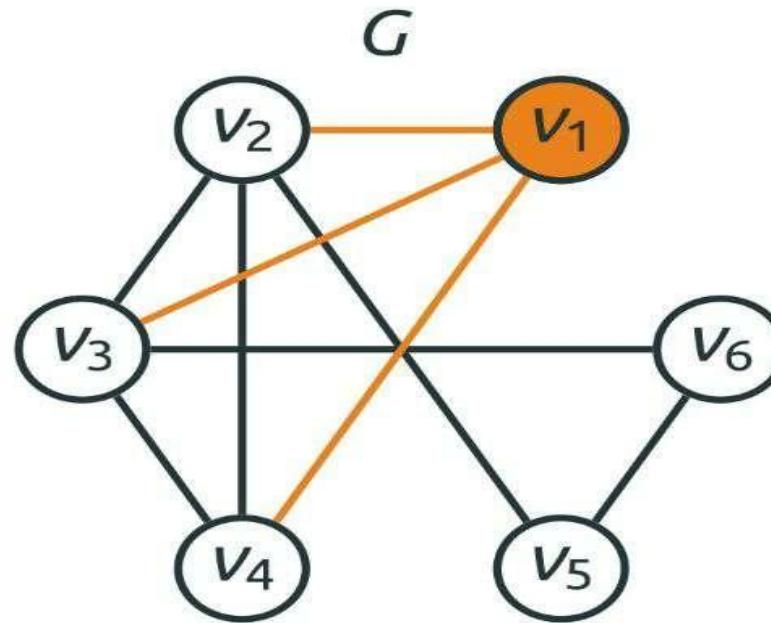
Complement Graph



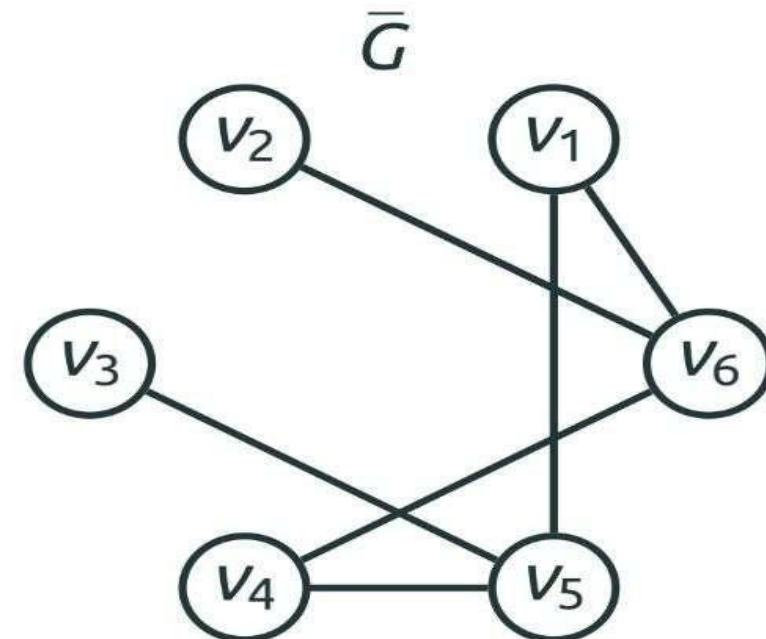
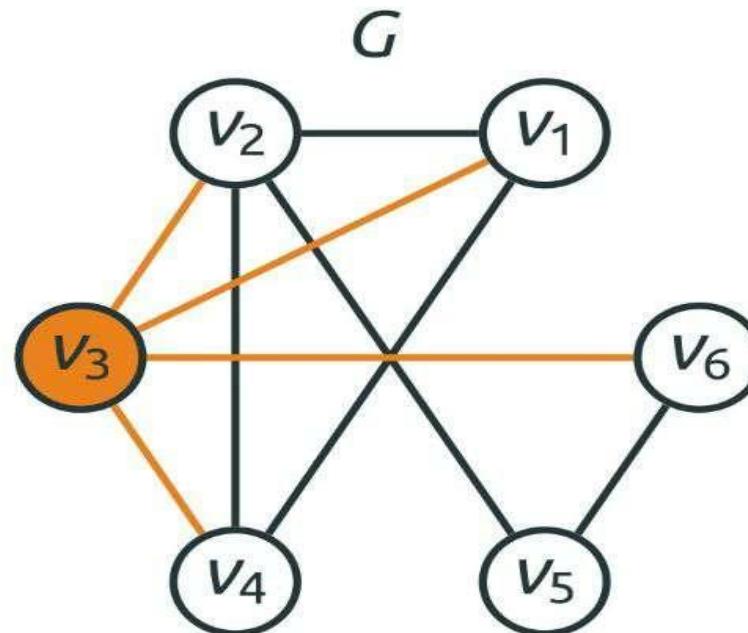
Complement Graph



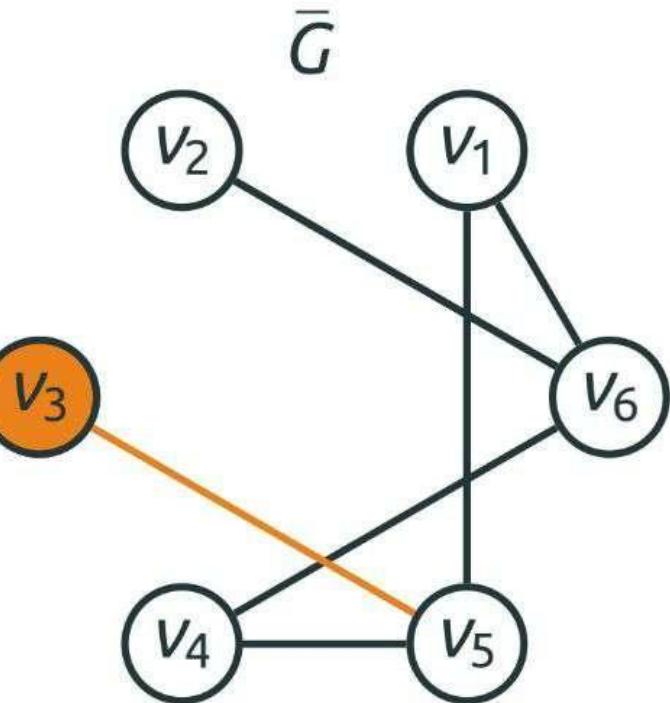
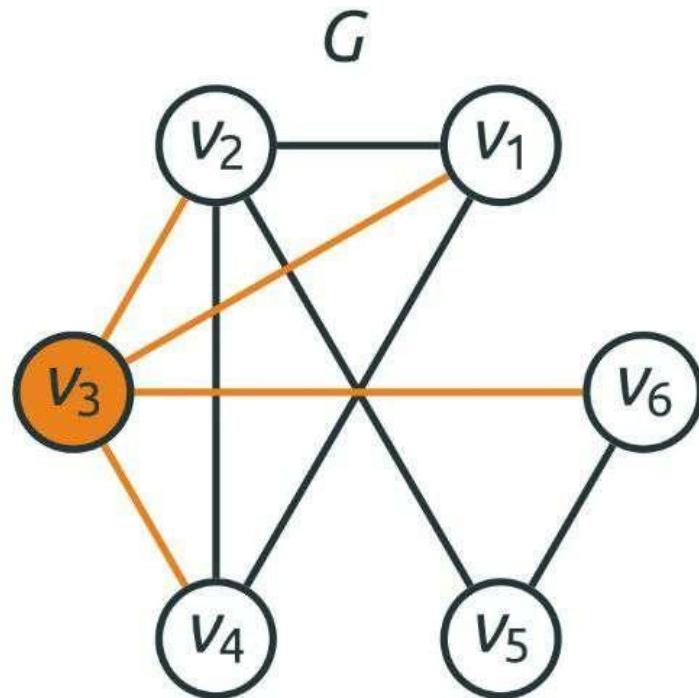
Complement Graph



Complement Graph

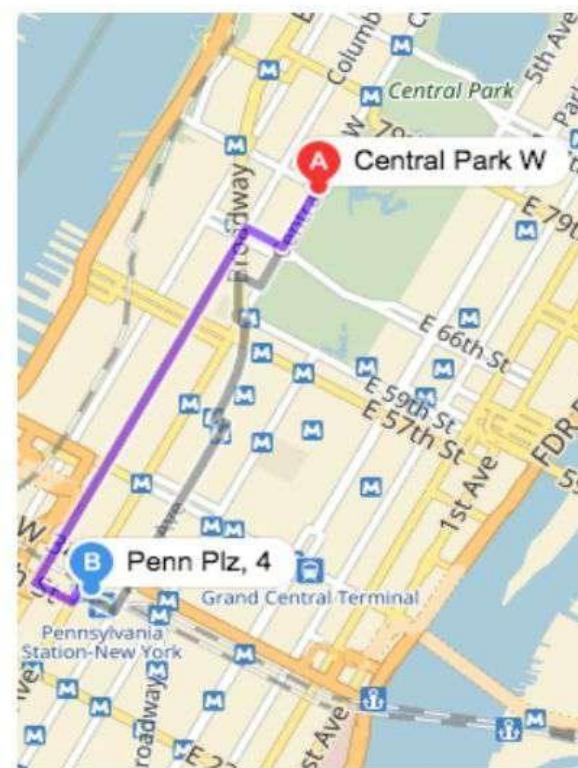


Complement Graph



Paths

Is there a path from one point to another?



Walks

- A **Walk** in a graph is a sequence of edges, such that each edge (except for the first one) starts with a vertex where the previous edge ended

Walks

- A **Walk** in a graph is a sequence of edges, such that each edge (except for the first one) starts with a vertex where the previous edge ended
- The **Length** of a walk is the number of edges in it

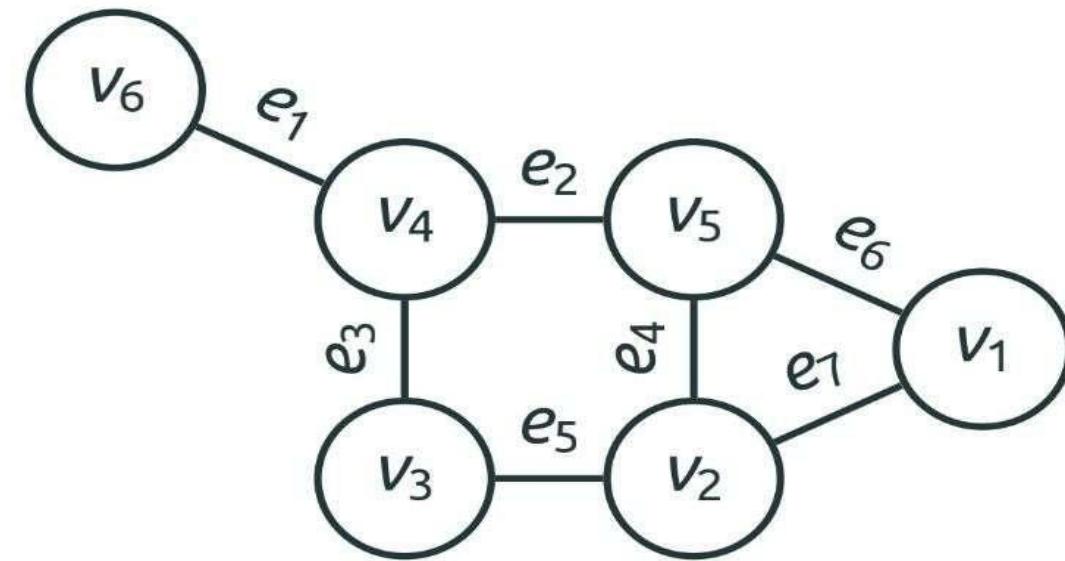
Walks

- A **Walk** in a graph is a sequence of edges, such that each edge (except for the first one) starts with a vertex where the previous edge ended
- The **Length** of a walk is the number of edges in it
- A **Path** is a walk where all edges are distinct

Walks

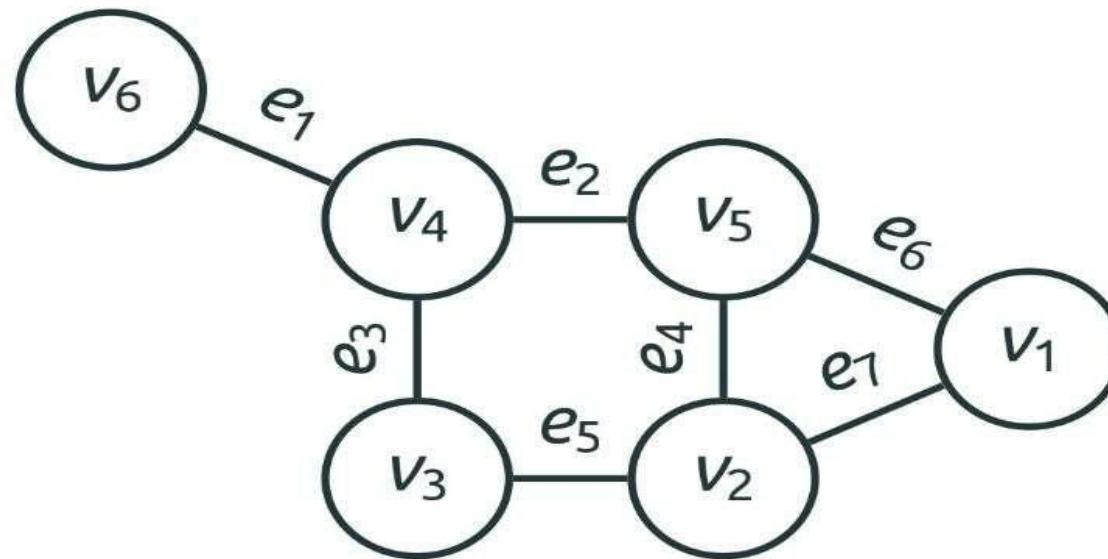
- A **Walk** in a graph is a sequence of edges, such that each edge (except for the first one) starts with a vertex where the previous edge ended
- The **Length** of a walk is the number of edges in it
- A **Path** is a walk where all edges are distinct
- A **Simple Path** is a walk where all vertices are distinct

Walks: Examples



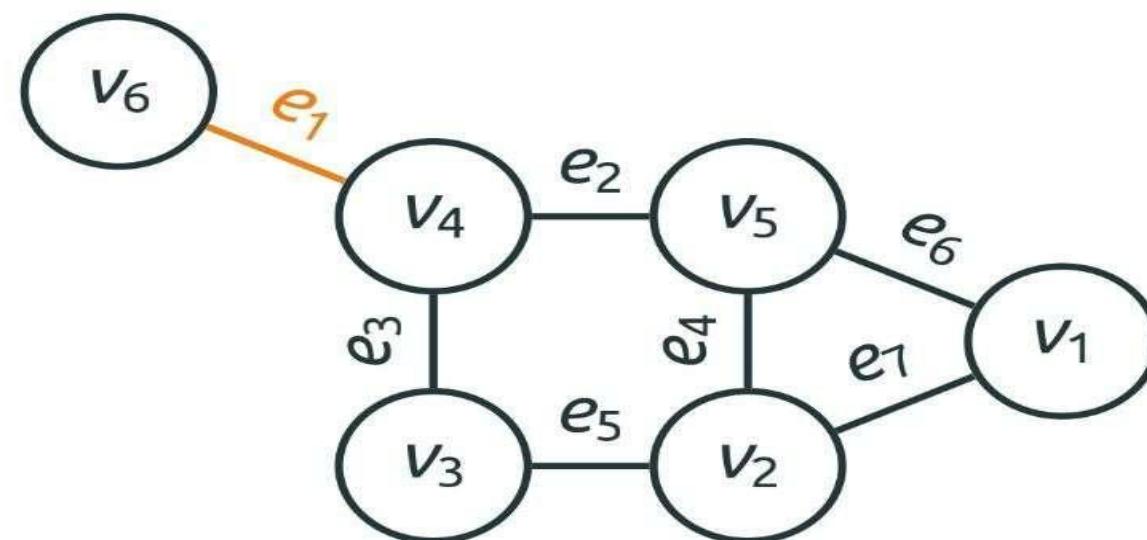
Walks: Examples

A **walk** of length 6: $(e_1, e_2, e_4, e_5, e_3, e_1)$



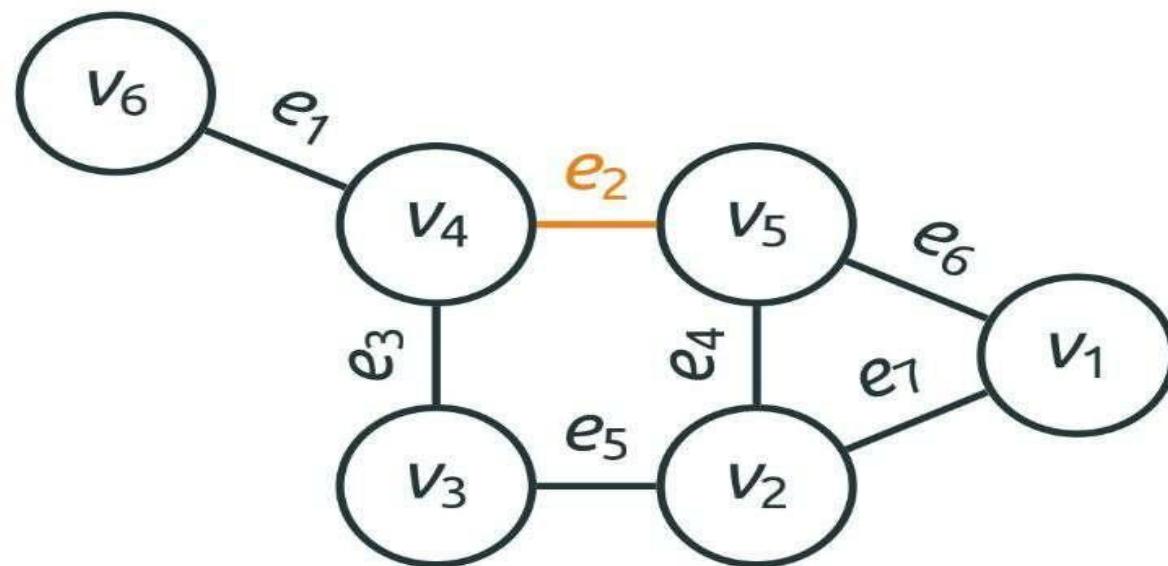
Walks: Examples

A **walk** of length 6: ($e_1, e_2, e_4, e_5, e_3, e_1$)



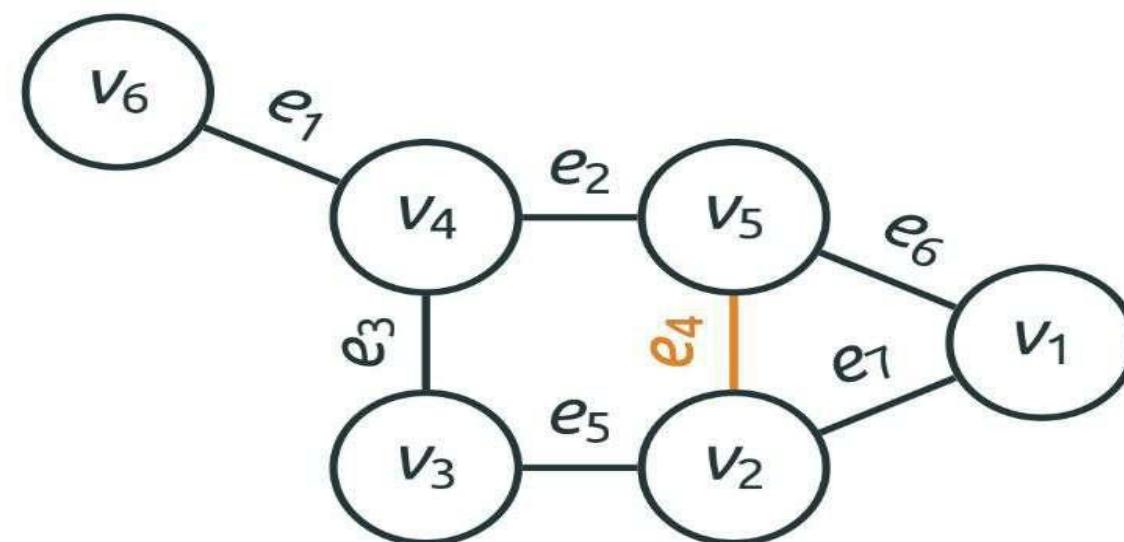
Walks: Examples

A **walk** of length 6: $(e_1, e_2, e_4, e_5, e_3, e_1)$



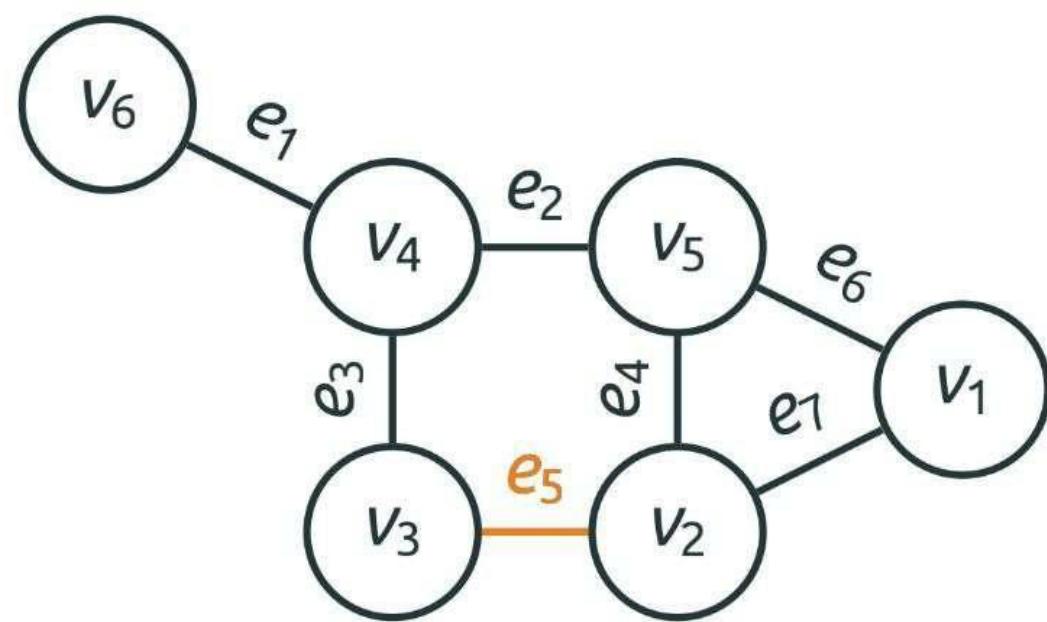
Walks: Examples

A **walk** of length 6: $(e_1, e_2, e_4, e_5, e_3, e_1)$



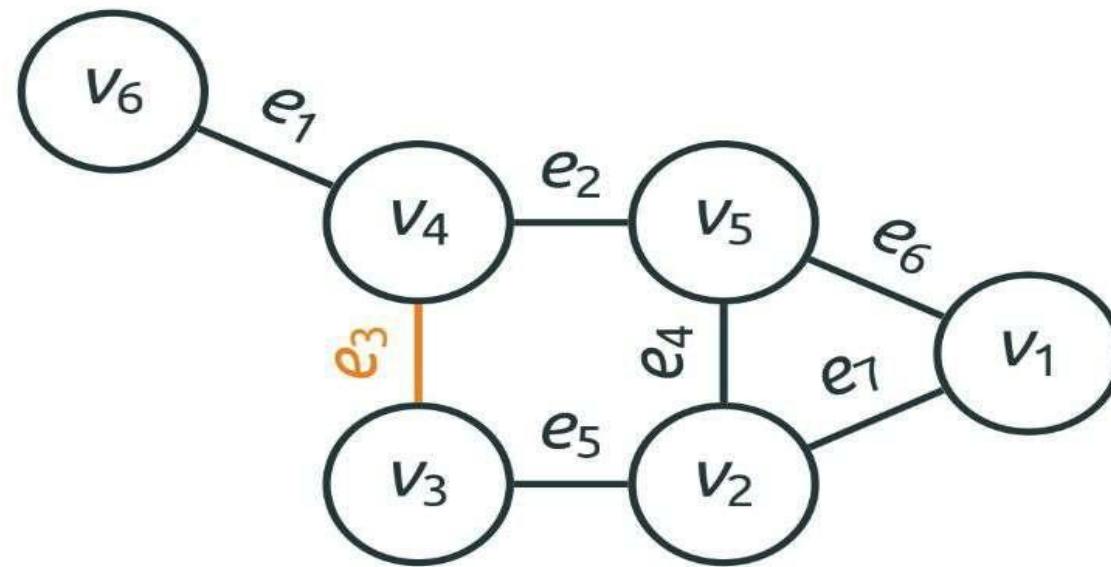
Walks: Examples

A **walk** of length 6: $(e_1, e_2, e_4, e_5, e_3, e_1)$



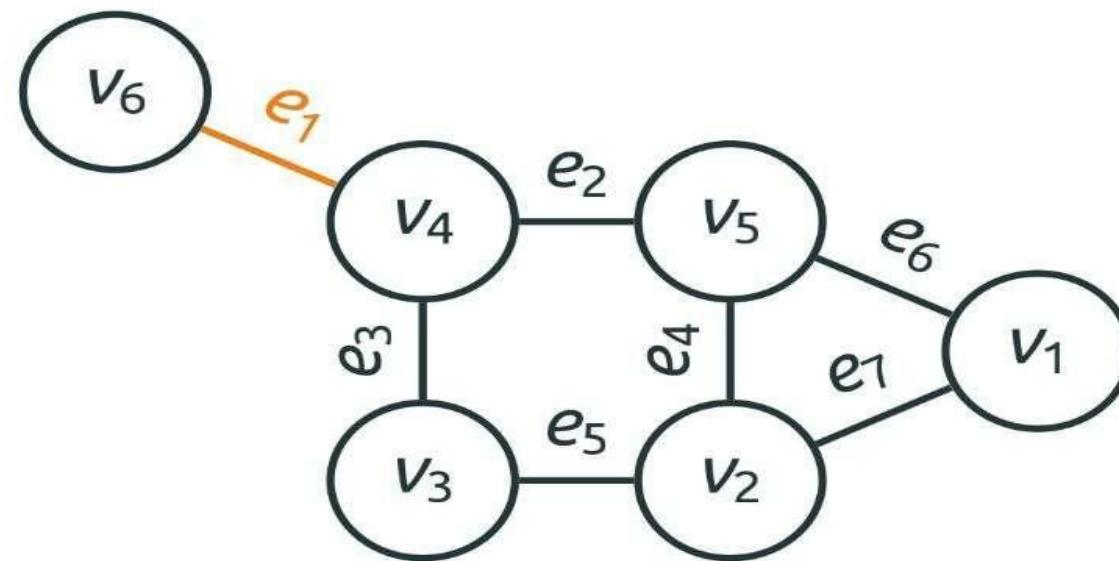
Walks: Examples

A **walk** of length 6: $(e_1, e_2, e_4, e_5, \textcolor{orange}{e}_3, e_1)$



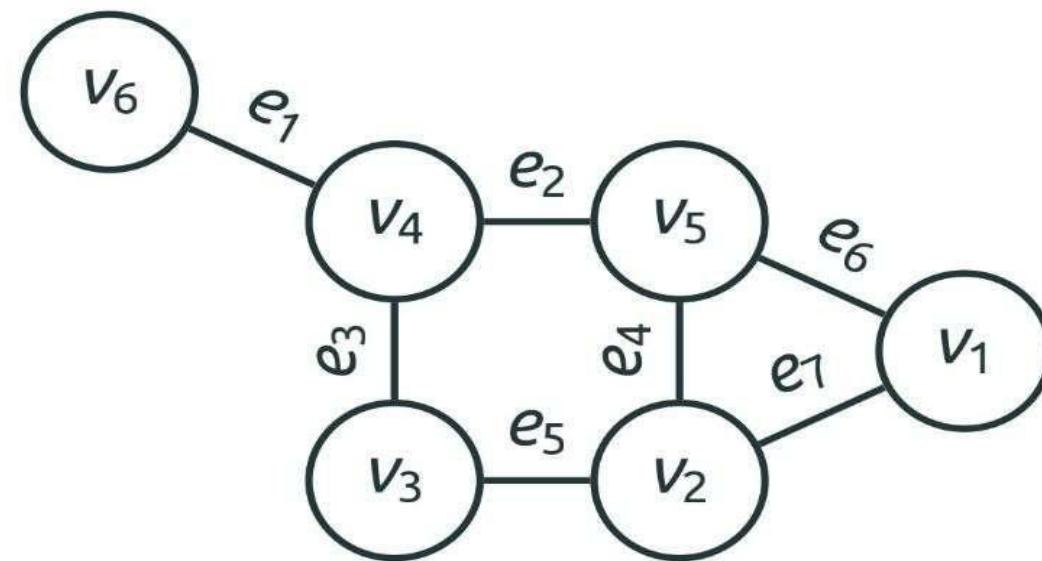
Walks: Examples

A **walk** of length 6: $(e_1, e_2, e_4, e_5, e_3, e_1)$



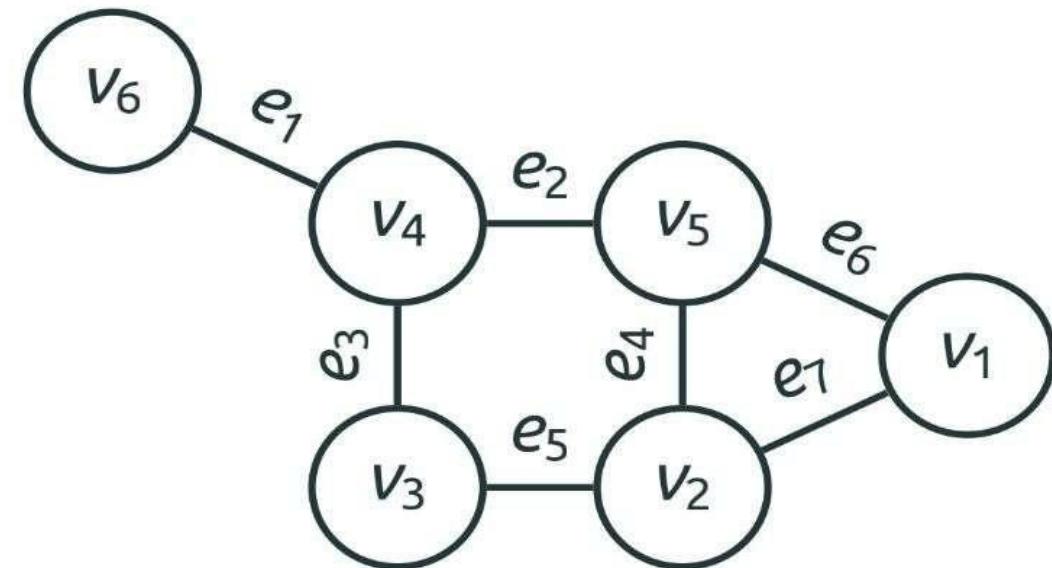
Walks: Examples

A **walk** of length 6: ($e_1, e_2, e_4, e_5, e_3, e_1$)
Not a **path**: uses e_1 twice



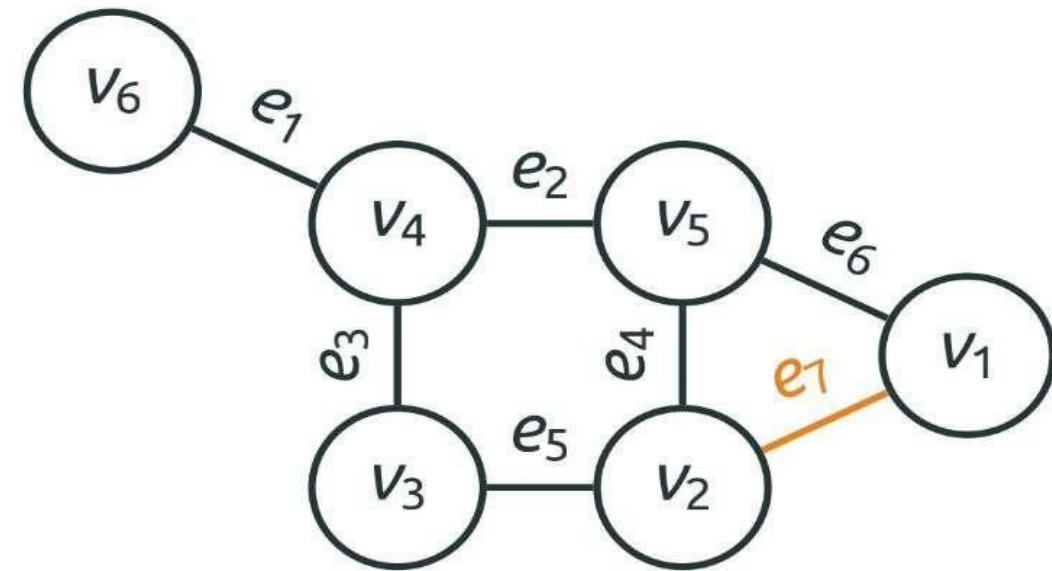
Walks: Examples

A **path** of length 4: (e_7, e_6, e_4, e_5)



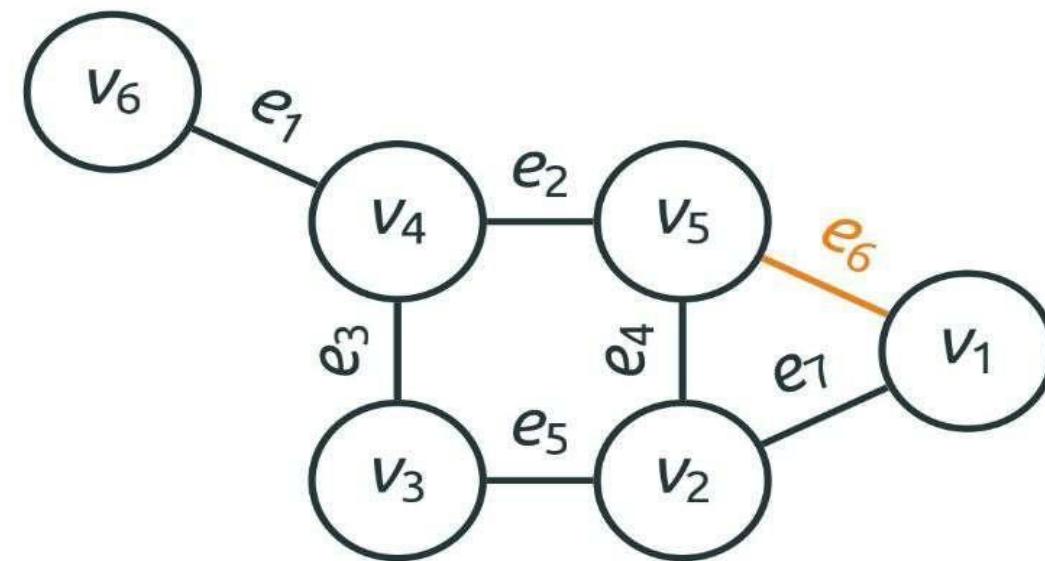
Walks: Examples

A **path** of length 4: (e_7, e_6, e_4, e_5)



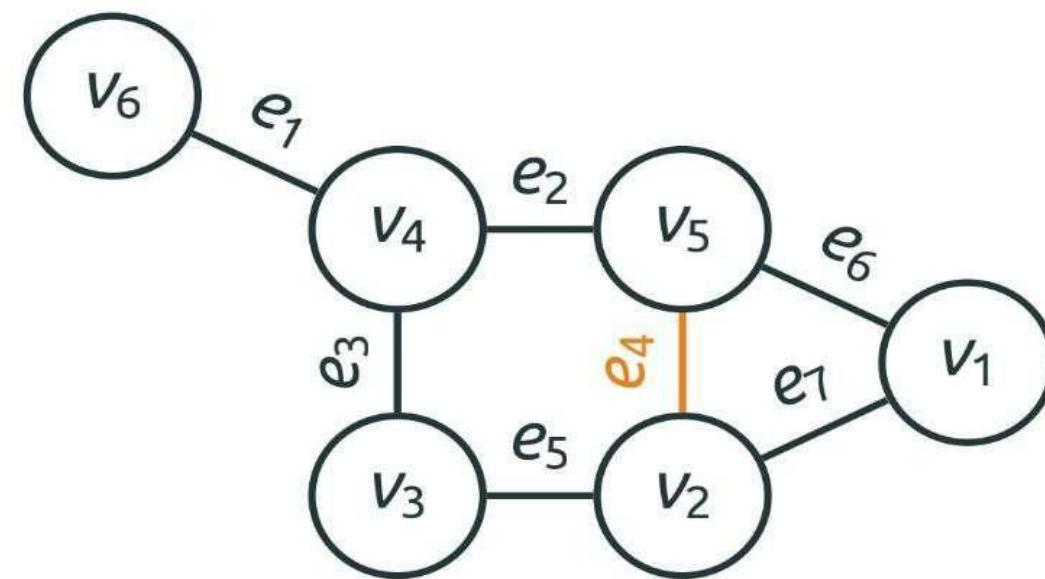
Walks: Examples

A **path** of length 4: (e_7, e_6, e_4, e_5)



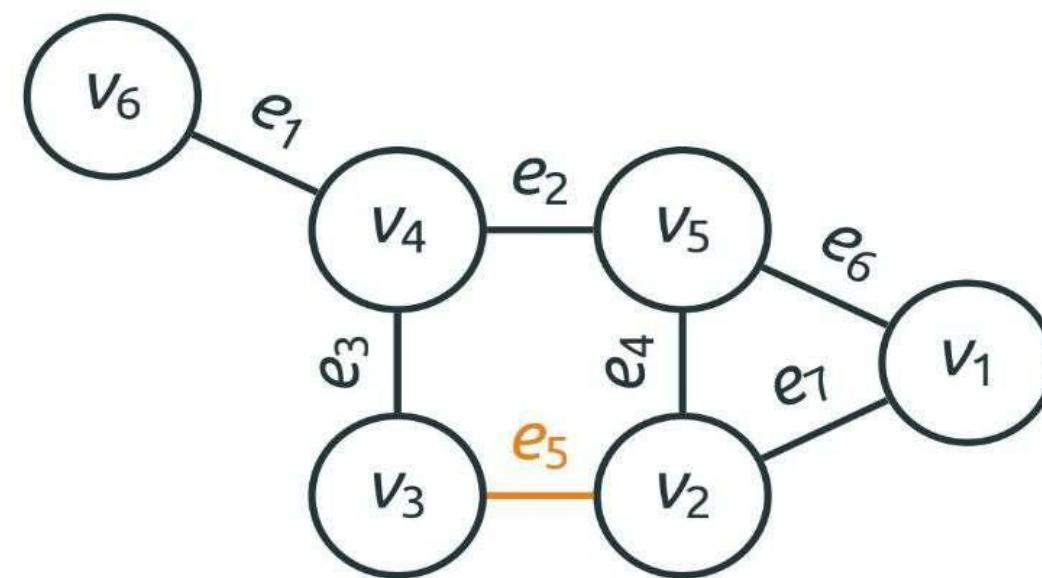
Walks: Examples

A **path** of length 4: (e_7, e_6, e_4, e_5)



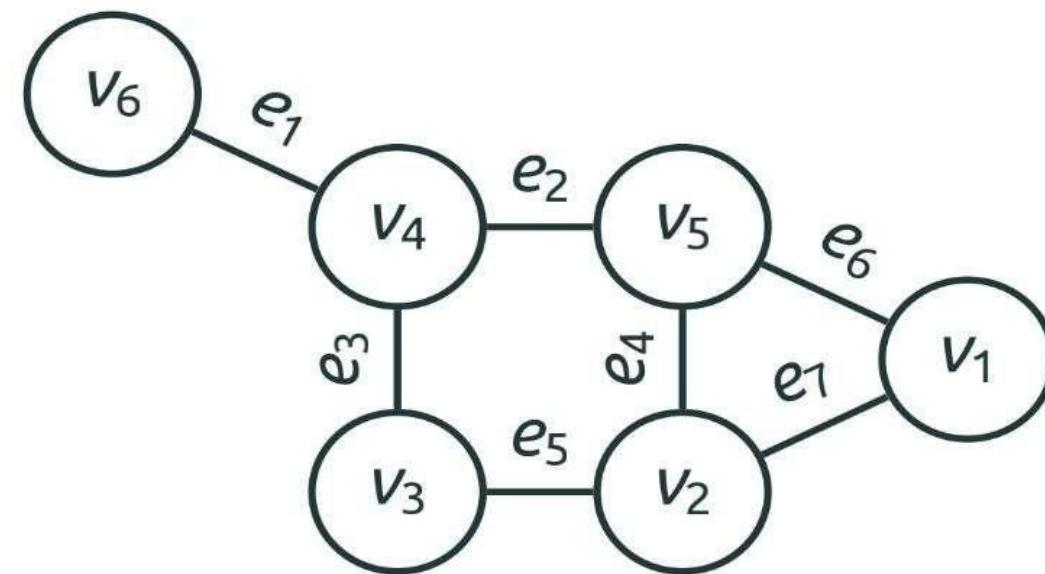
Walks: Examples

A **path** of length 4: (e_7, e_6, e_4, e_5)



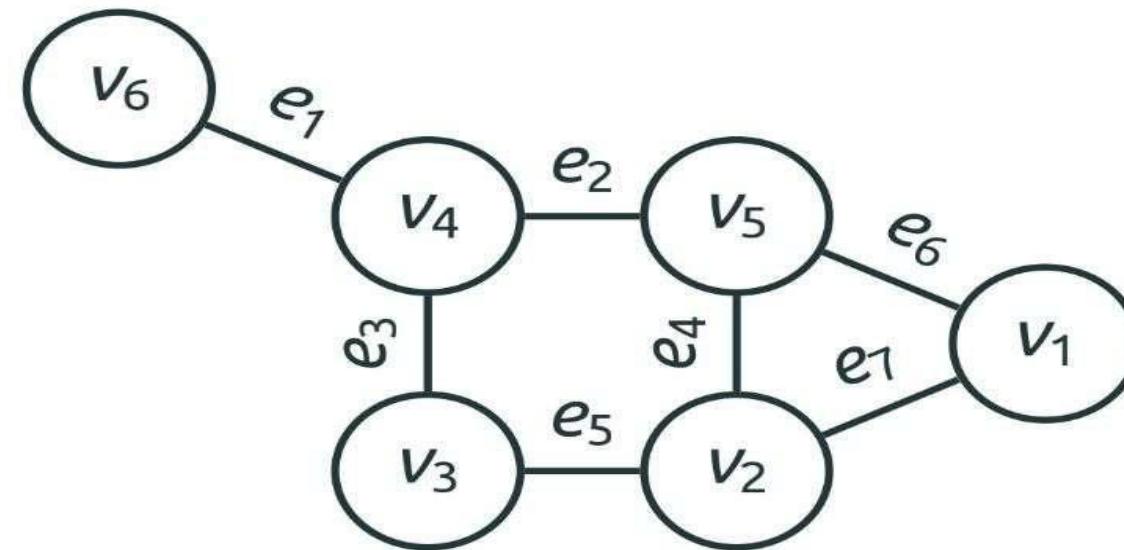
Walks: Examples

A **path** of length 4: (e_7, e_6, e_4, e_5)
Not a **simple path**: visits v_2 twice



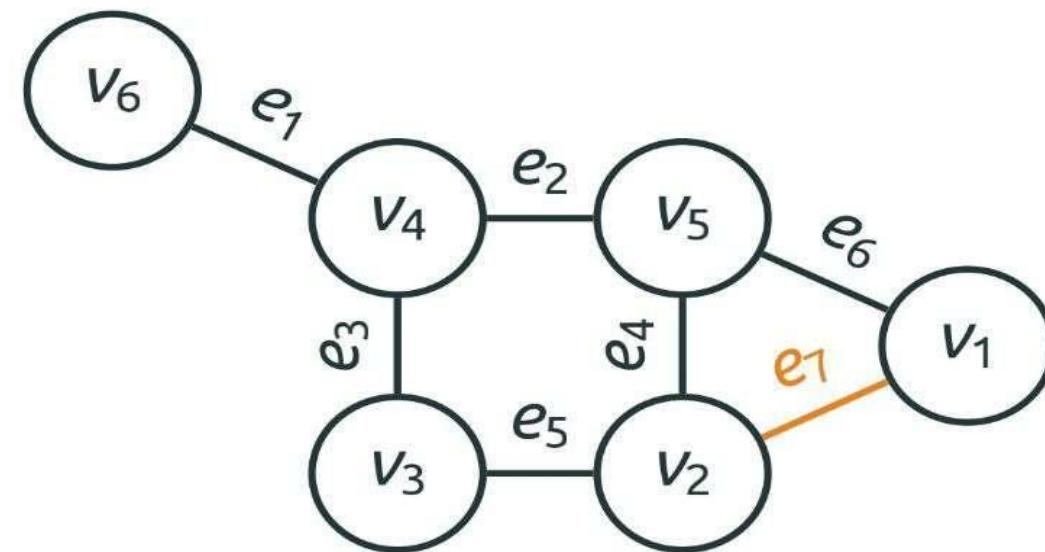
Walks: Examples

A **simple path** of length 4: (e_7, e_6, e_2, e_3)



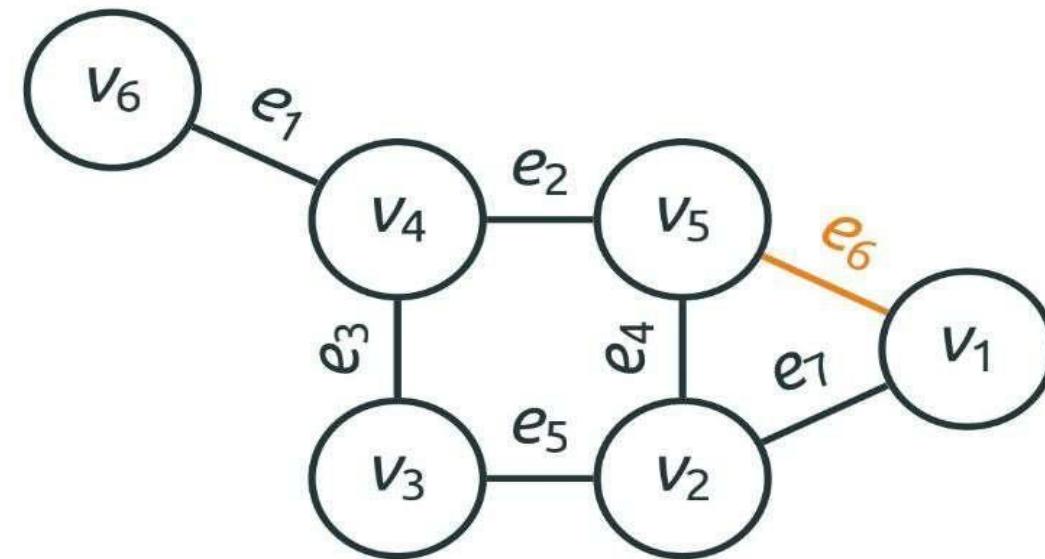
Walks: Examples

A **simple path** of length 4: (e_7, e_6, e_2, e_3)



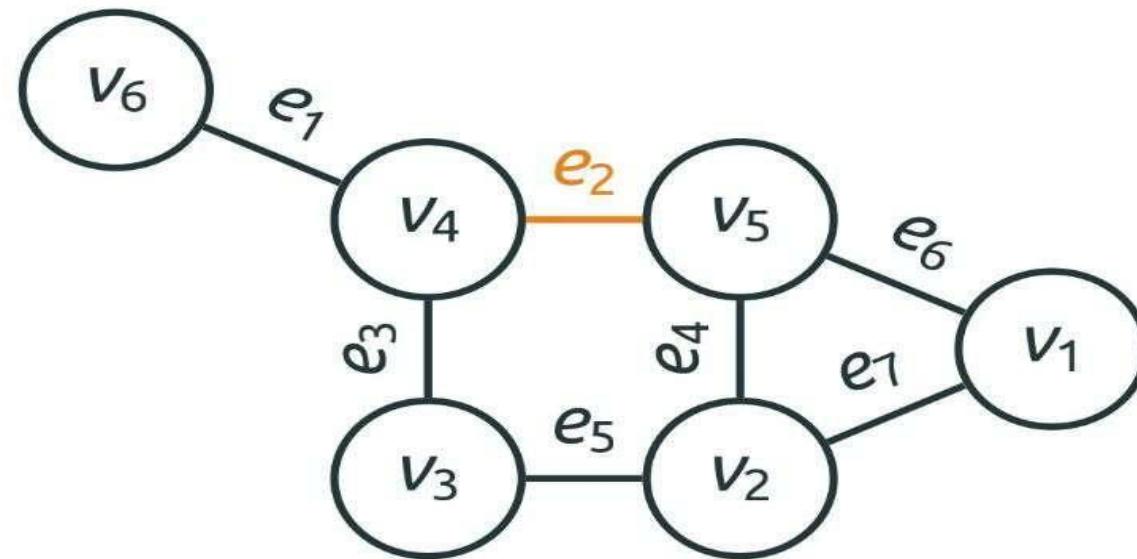
Walks: Examples

A **simple path** of length 4: (e_7, e_6, e_2, e_3)



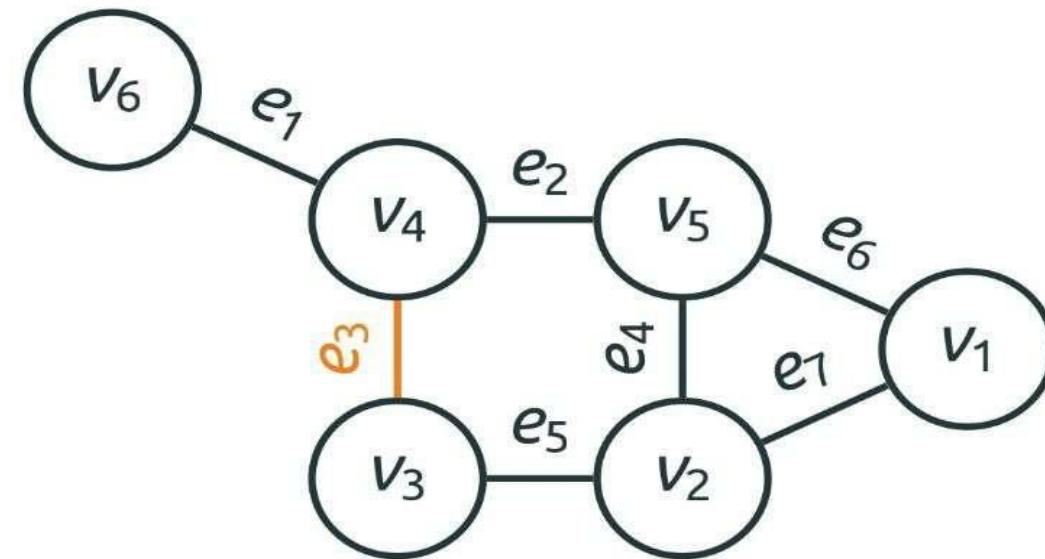
Walks: Examples

A **simple path** of length 4: (e_7, e_6, e_2, e_3)



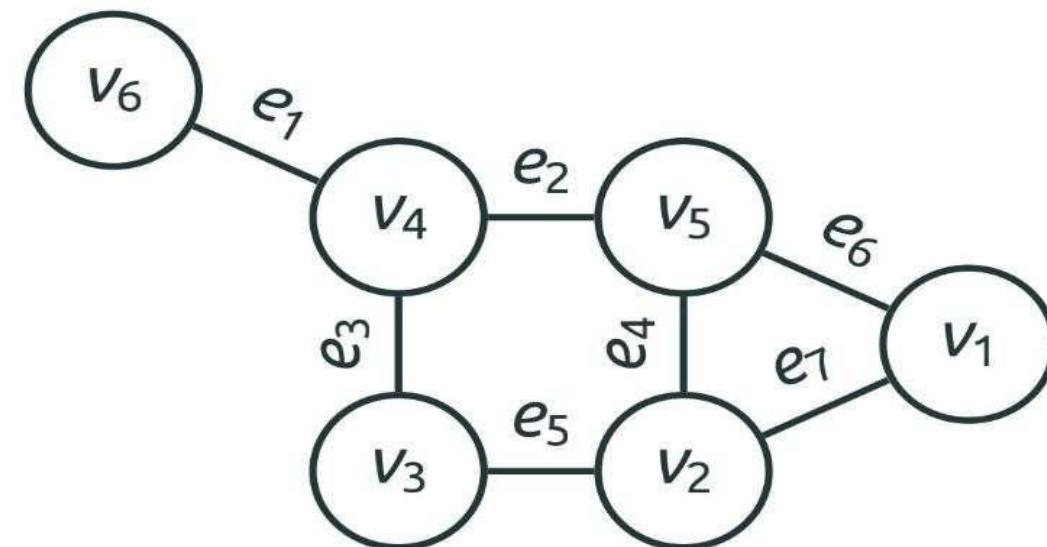
Walks: Examples

A **simple path** of length 4: (e_7, e_6, e_2, e_3)



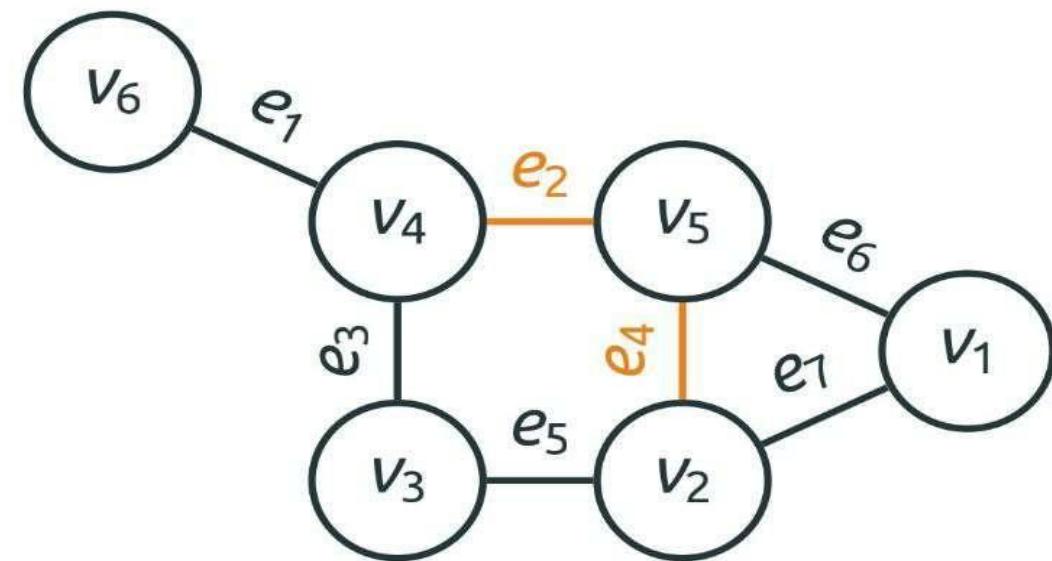
Walks: Examples

It is sometimes convenient to specify a path (walk) by a list of its vertices



Walks: Examples

(v_4, v_5, v_2) is a path of length 2



Cycles

- A **Cycle** in a graph is a path whose first vertex is the same as the last one

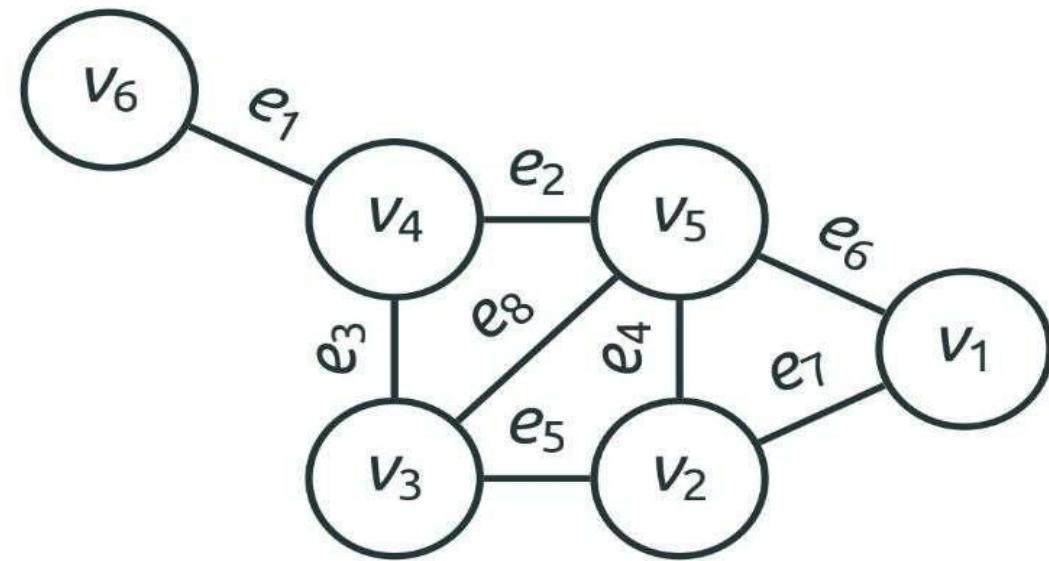
Cycles

- A **Cycle** in a graph is a path whose first vertex is the same as the last one
- In particular, all the edges in a **Cycle** are distinct

Cycles

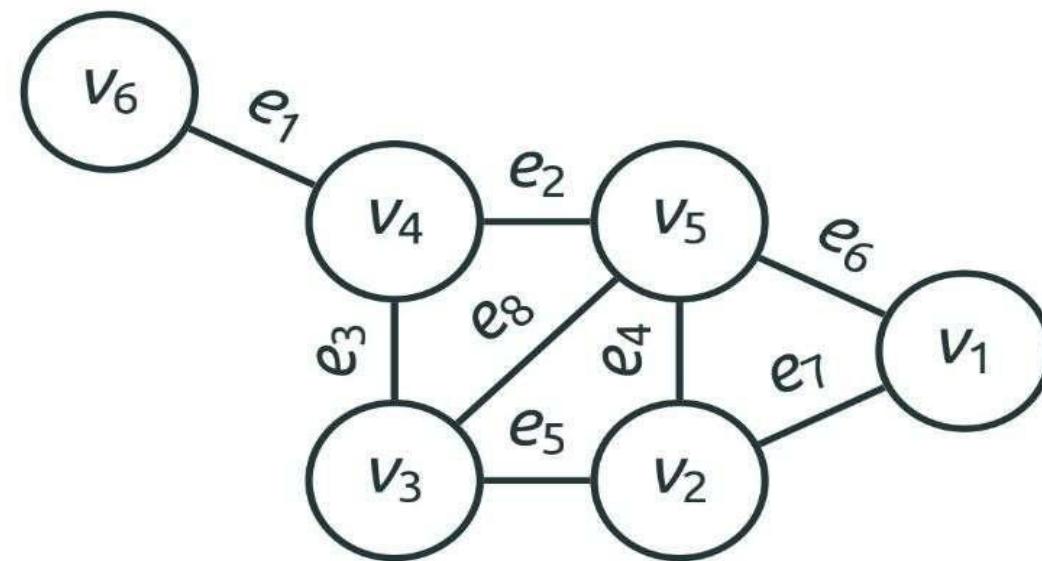
- A **Cycle** in a graph is a path whose first vertex is the same as the last one
- In particular, all the edges in a **Cycle** are distinct
- A **Simple Cycle** is a cycle where all vertices except for the first one are distinct. (And the first vertex is taken twice)

Cycles: Examples



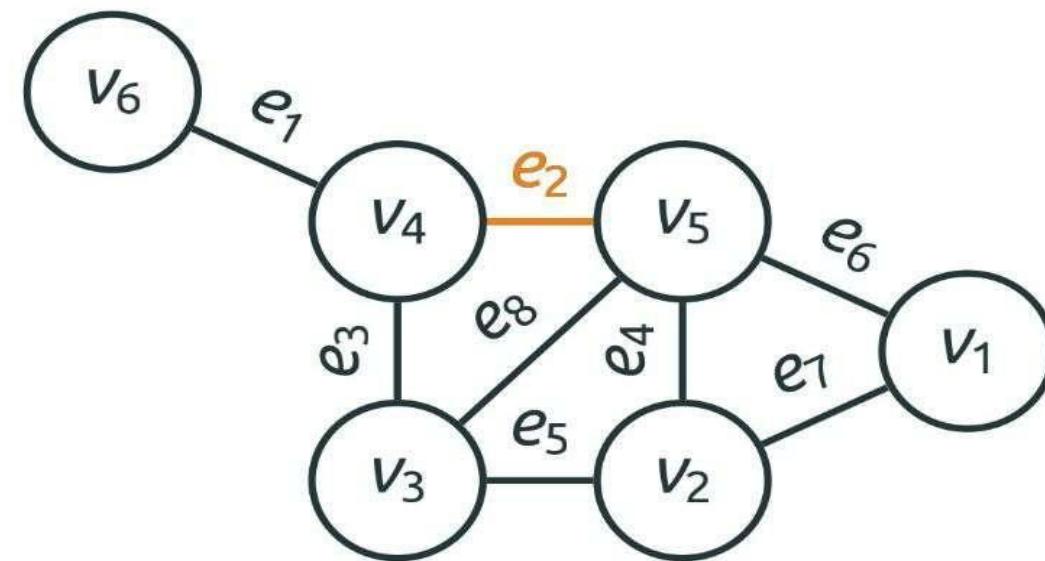
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$



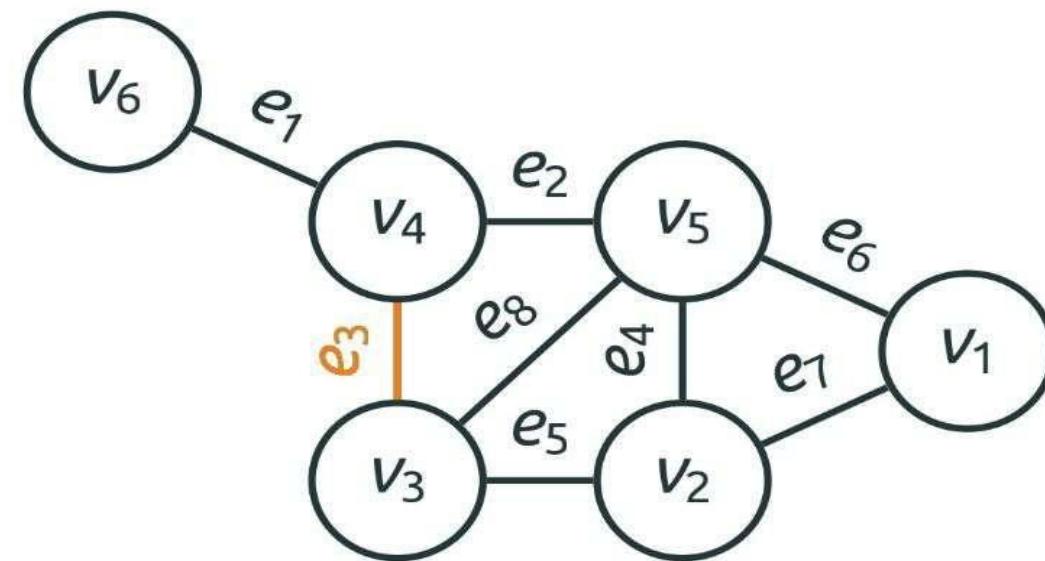
Cycles: Examples

A **cycle** of length 6: ($e_2, e_3, e_8, e_4, e_7, e_6$)



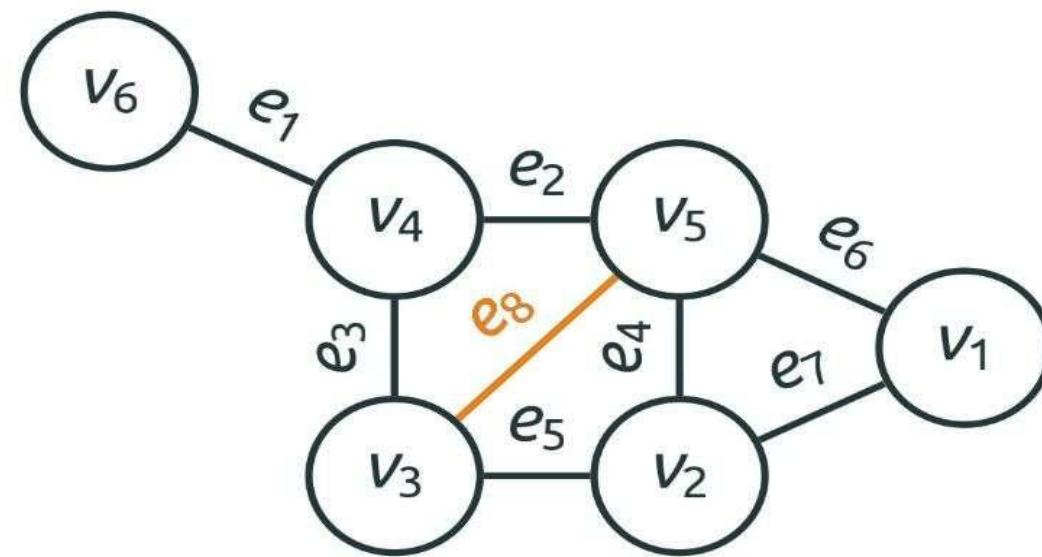
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$



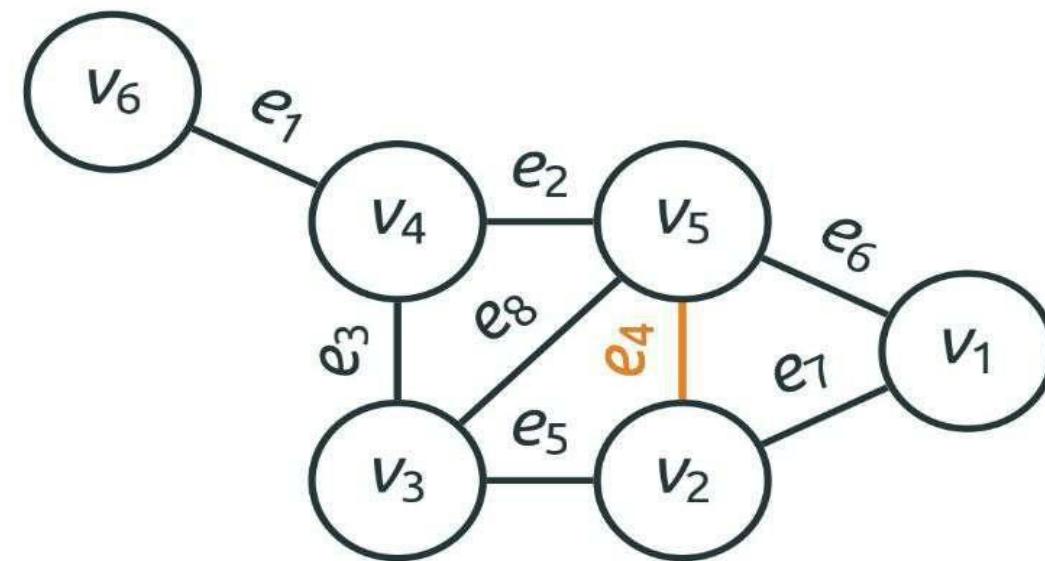
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$



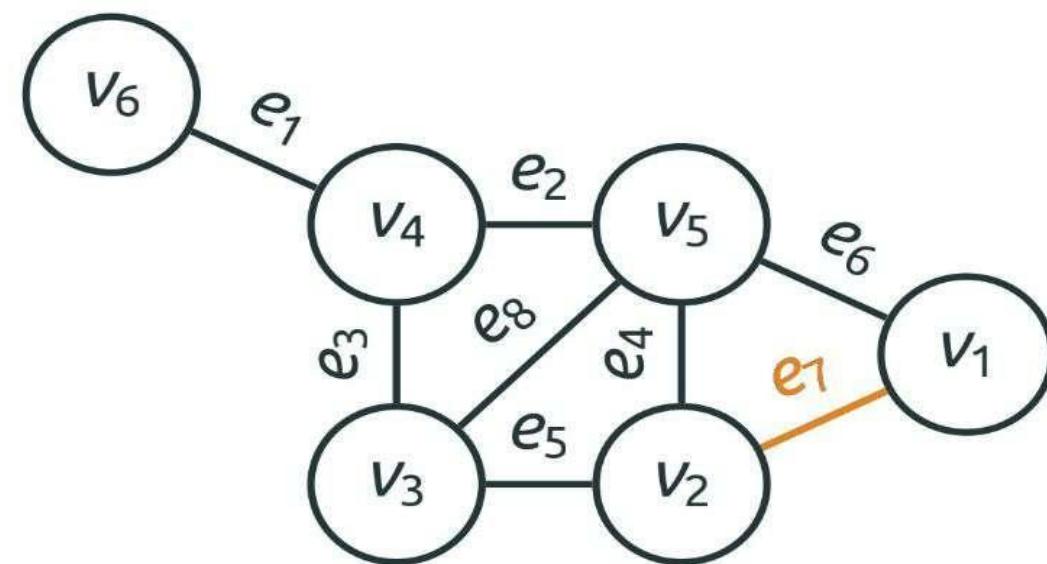
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$



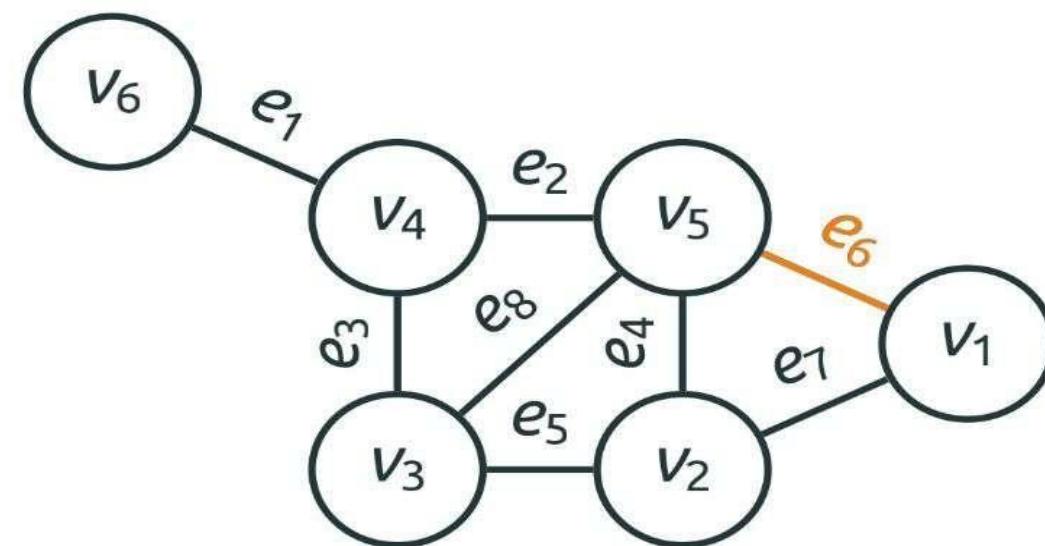
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$



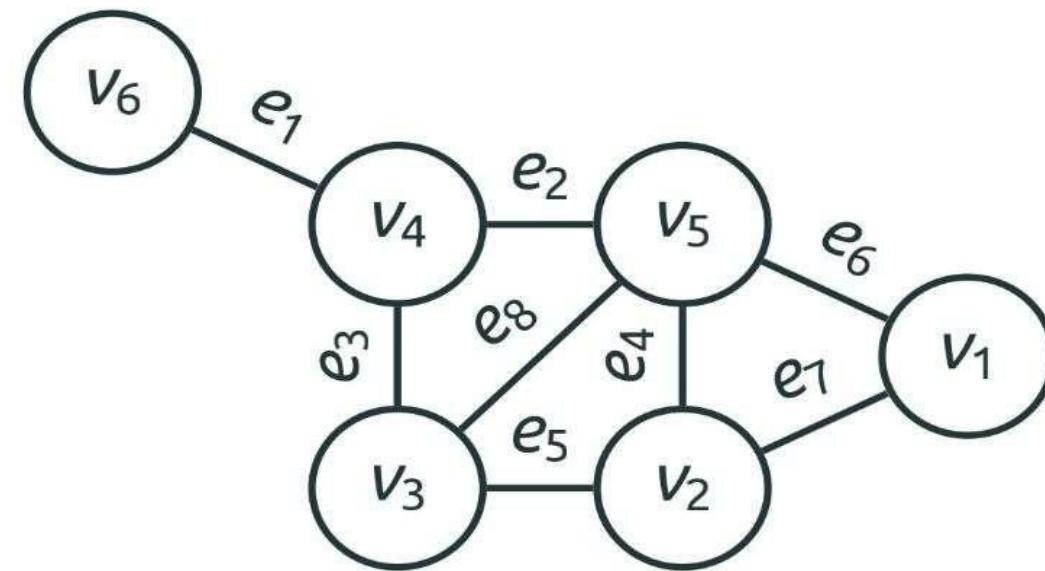
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$



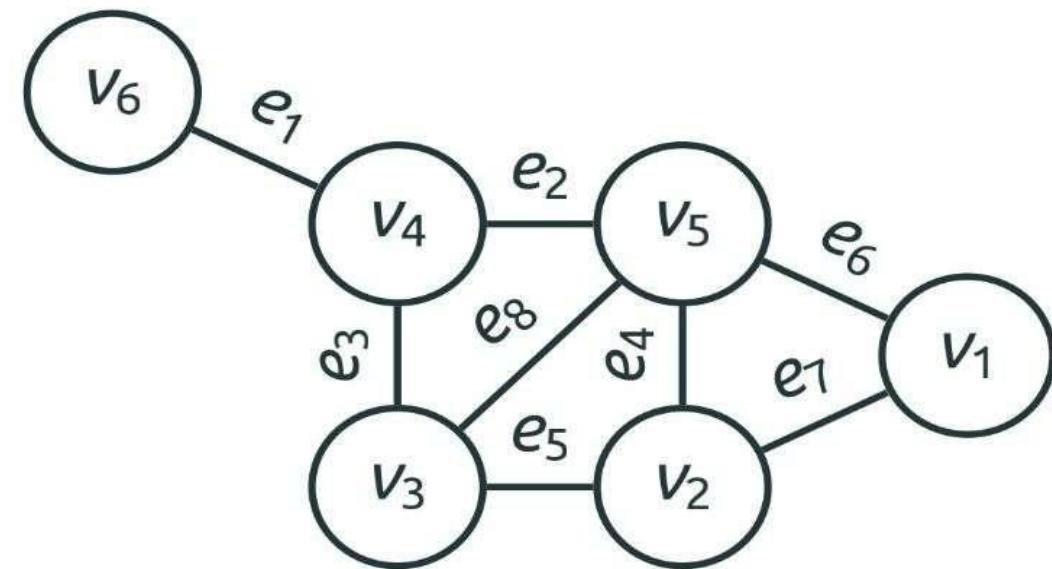
Cycles: Examples

A **cycle** of length 6: $(e_2, e_3, e_8, e_4, e_7, e_6)$
Not a simple cycle: visits v_5 three times



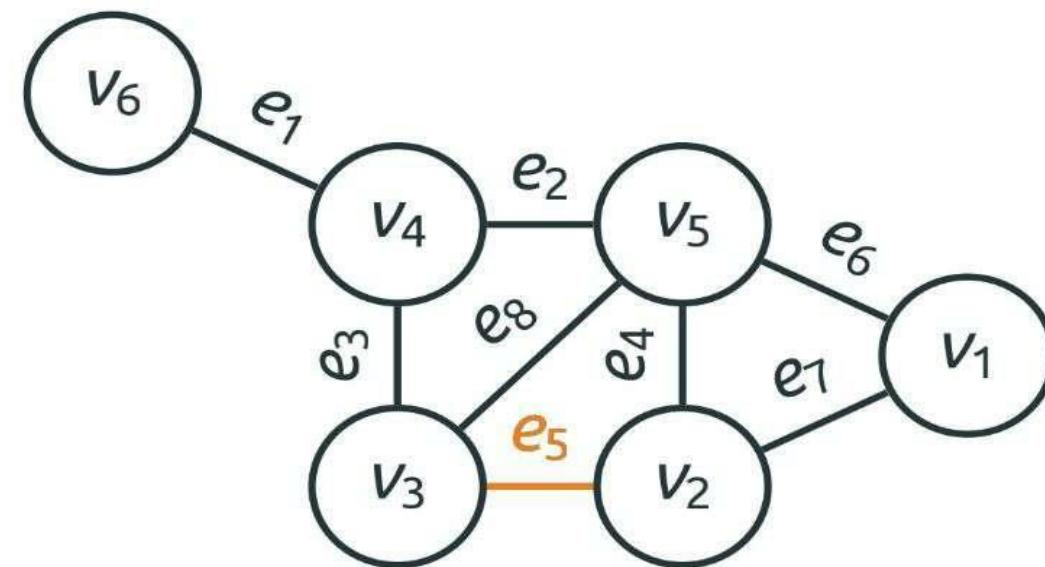
Cycles: Examples

A **simple cycle** of length 4: (e_5, e_4, e_2, e_3)



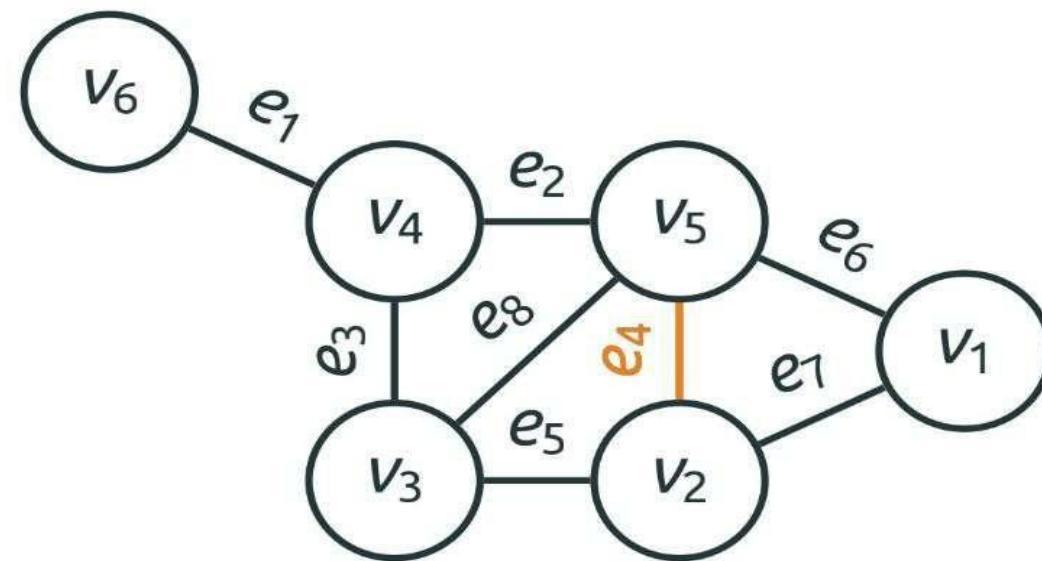
Cycles: Examples

A **simple cycle** of length 4: (e_5, e_4, e_2, e_3)



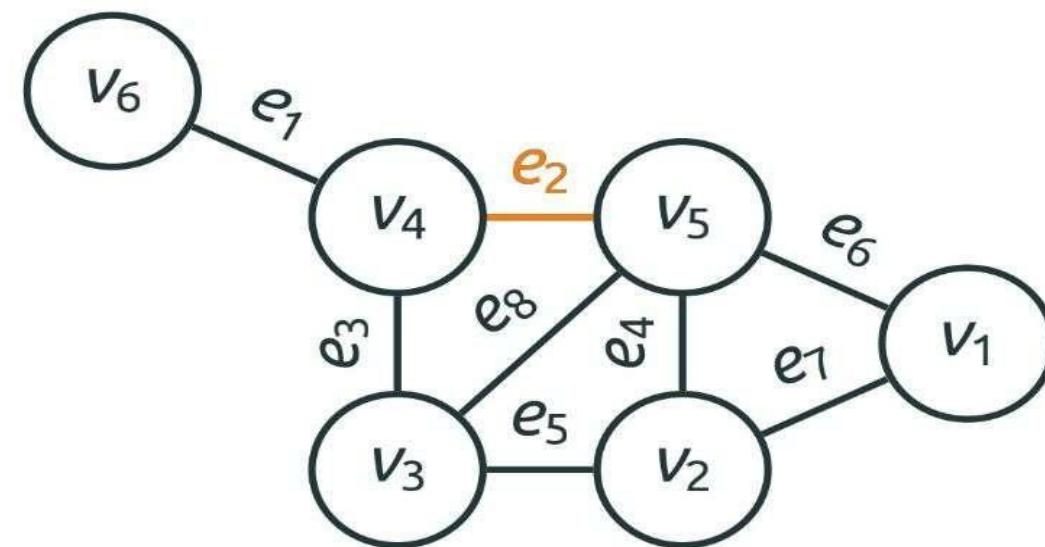
Cycles: Examples

A **simple cycle** of length 4: (e_5, e_4, e_2, e_3)



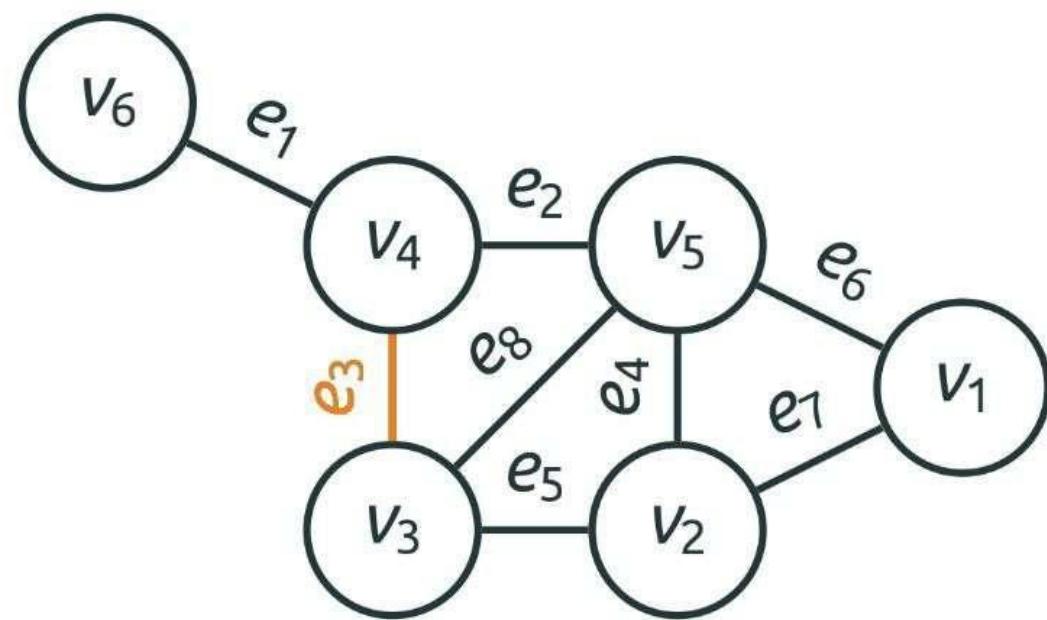
Cycles: Examples

A **simple cycle** of length 4: (e_5, e_4, e_2, e_3)



Cycles: Examples

A **simple cycle** of length 4: (e_5, e_4, e_2, e_3)



Directed Edge (Arc)

Arc (u, v)



Directed Edge (Arc)

Arc (u, v)

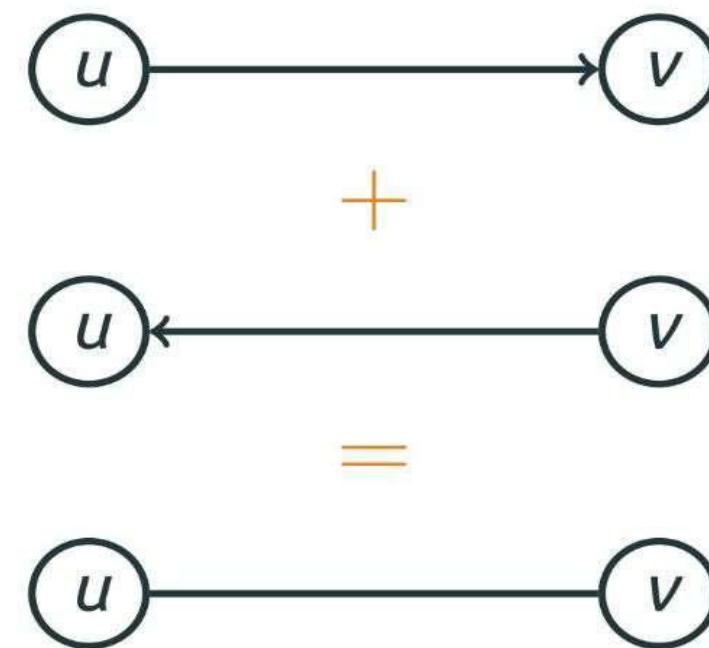


+



Directed Edge (Arc)

Arc (u, v)

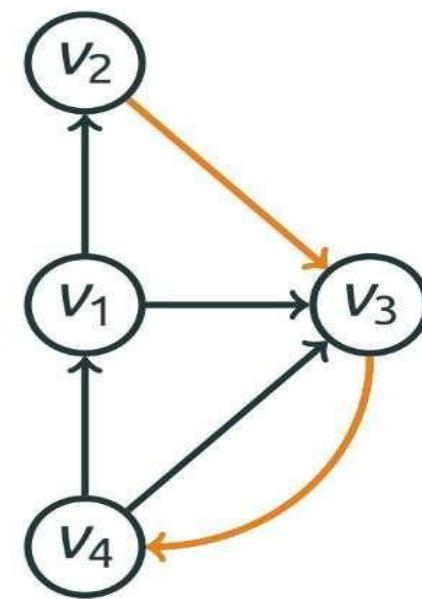


The Degree of a Vertex

- The **Indegree** of a vertex v is the number of edges ending at v
- The **Outdegree** of a vertex v is the number of edges leaving v

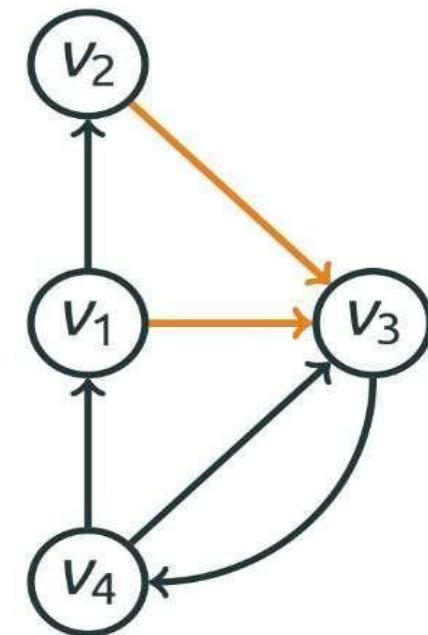
Directed Paths

(v_2, v_3, v_4) is a
Path of length 2



Directed Paths

(v_1, v_3, v_2) is **not** a Path

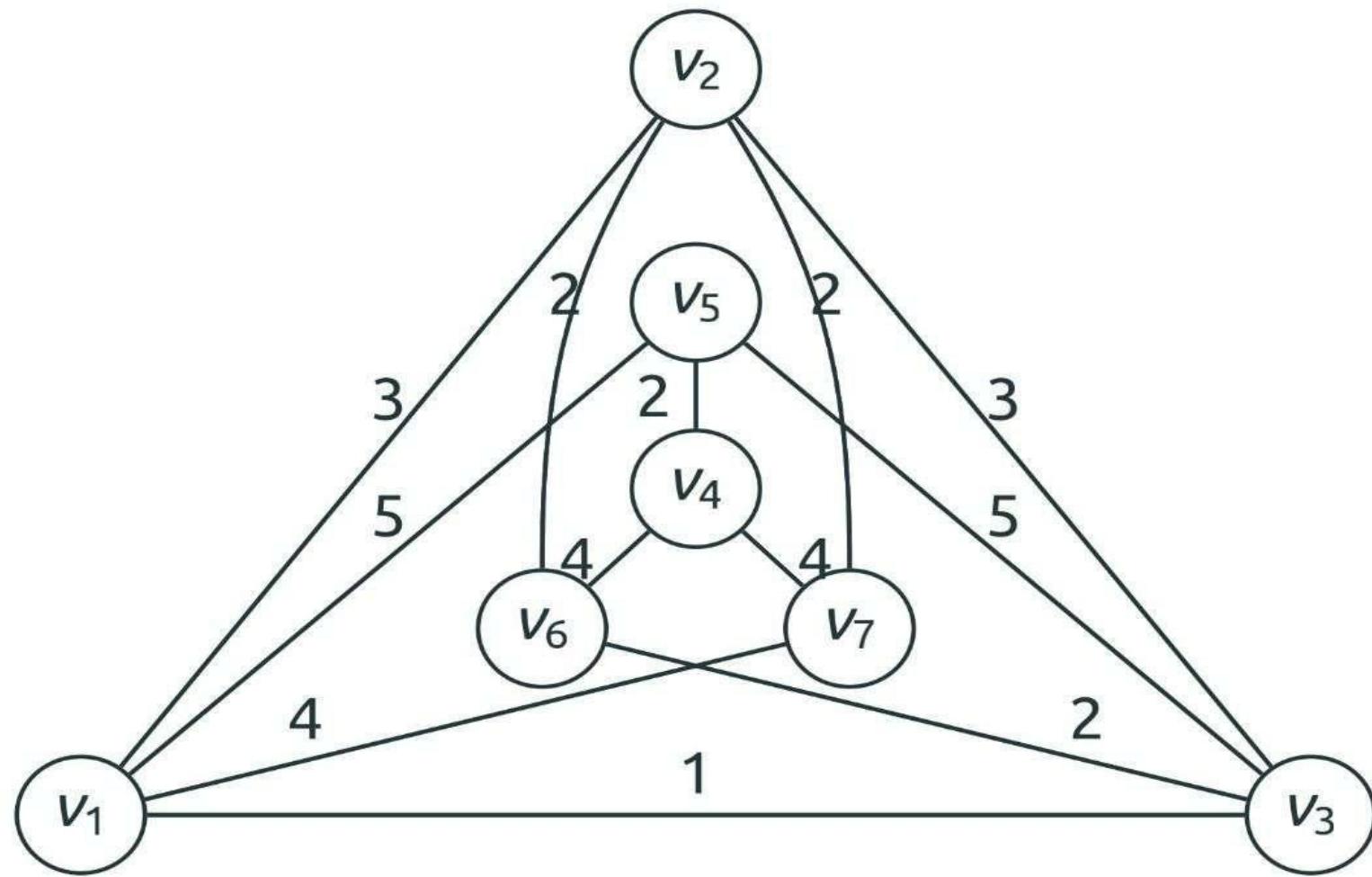


Weighted Graphs

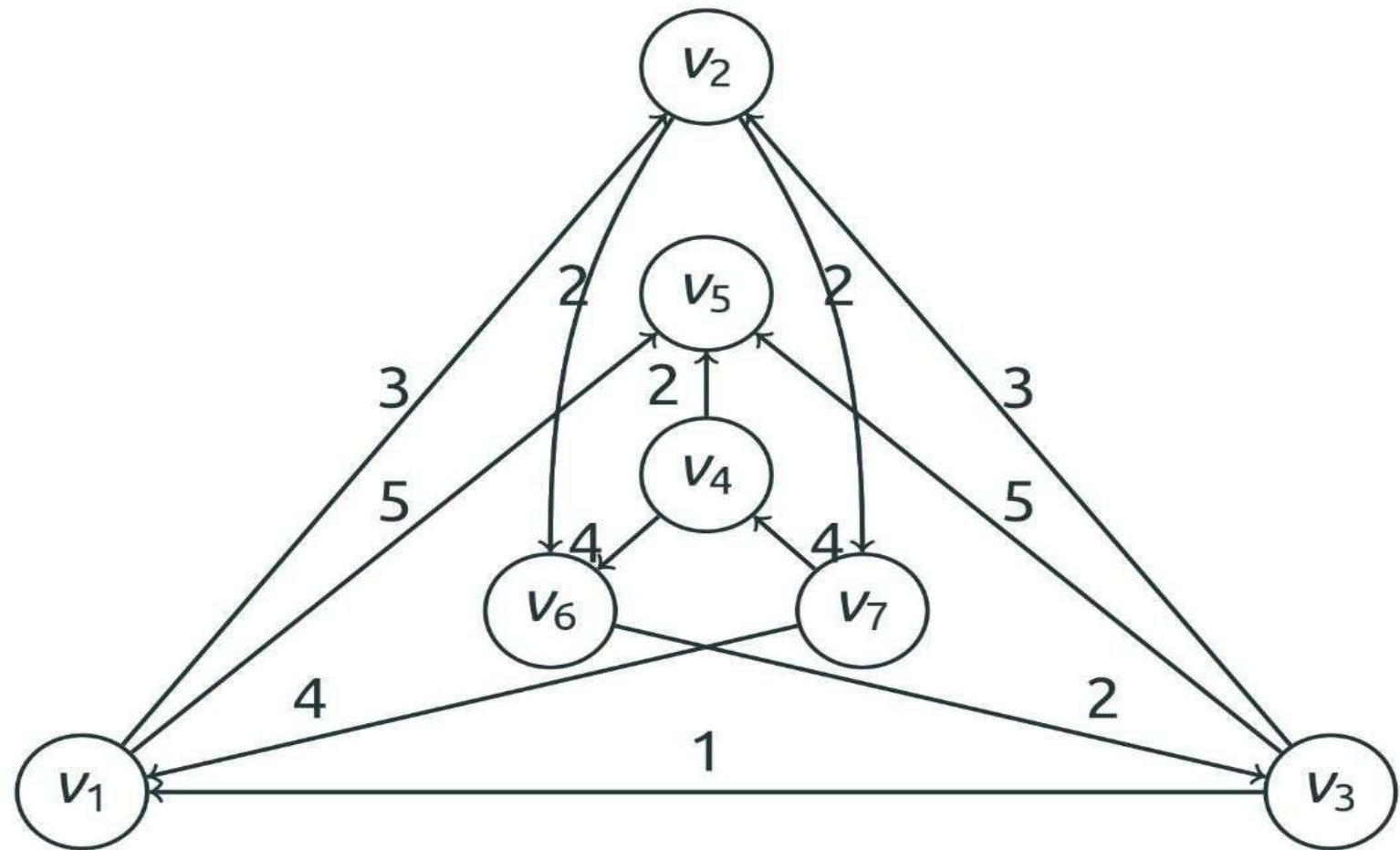
Distance, Driving Time, etc.



Weighted Graphs: Examples



Weighted Graphs: Examples

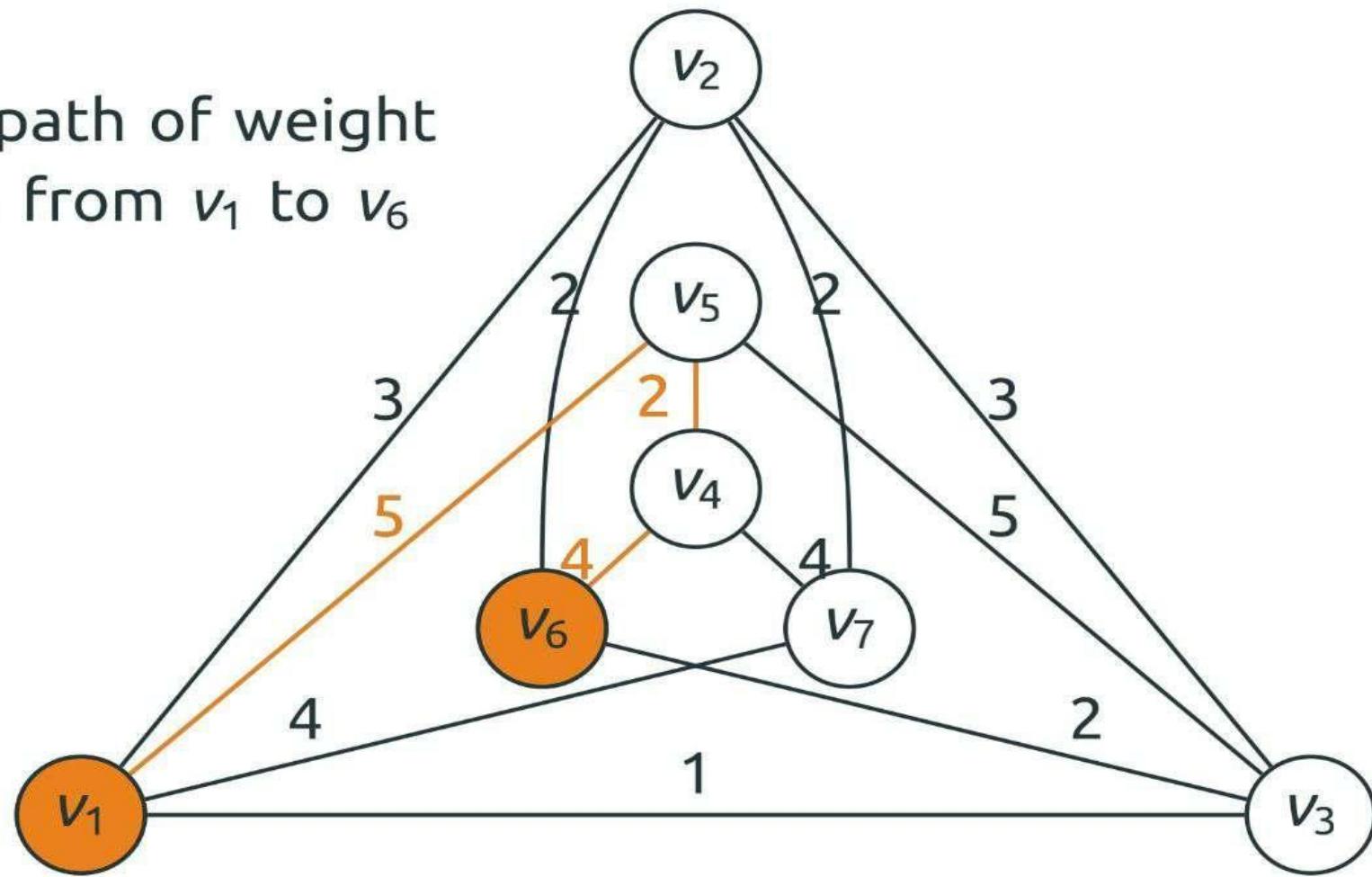


Weighted Paths

- A **Weighted Graph** associates a weight with every edge
- The **Weight** of a path is the sum of the weights of its edges
- A **Shortest Path** between two vertices is a path of the minimum weight
- The **Distance** between two vertices is the length of a shortest path between them

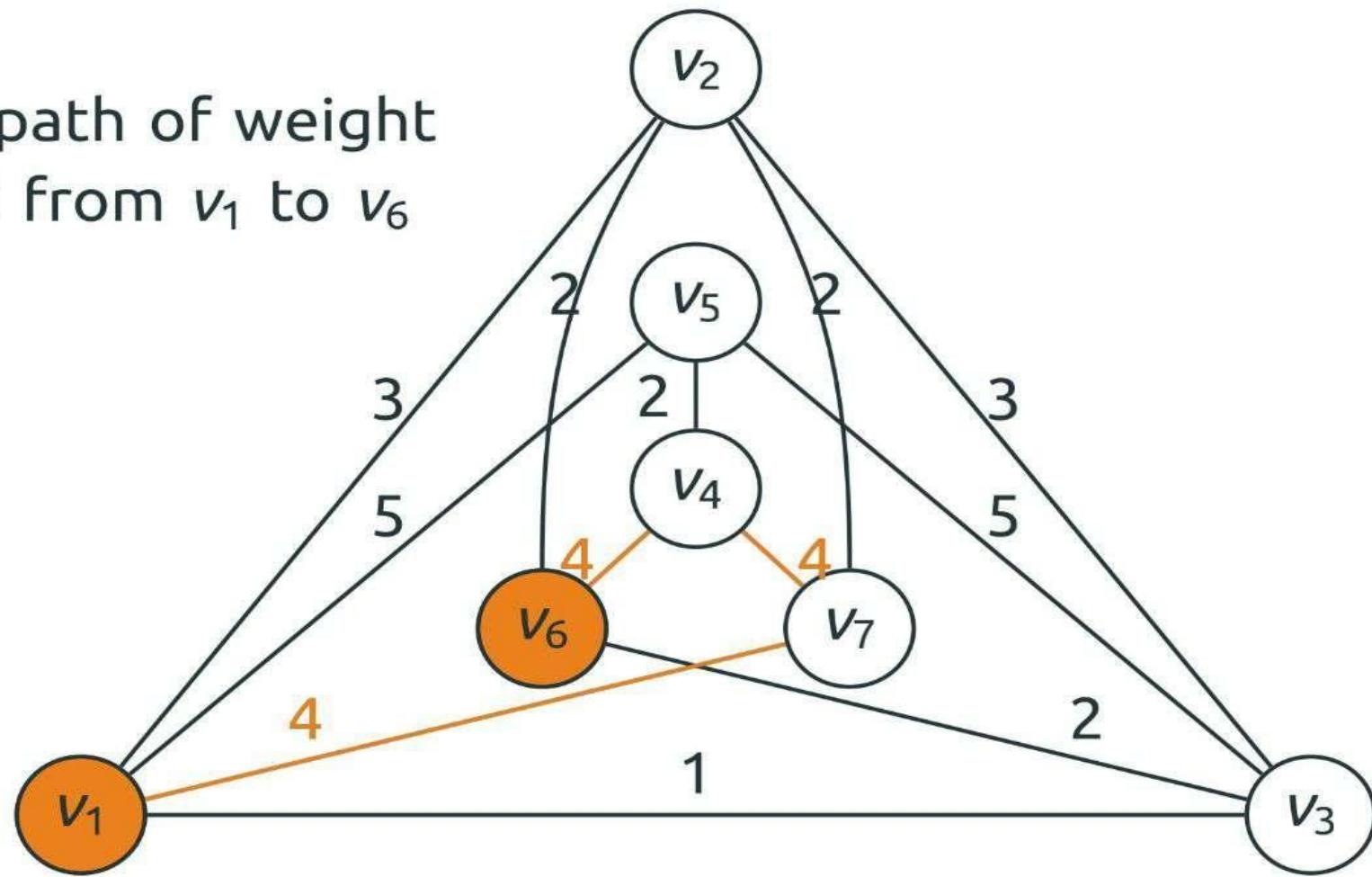
Weighted Paths: Examples

A path of weight
11 from v_1 to v_6



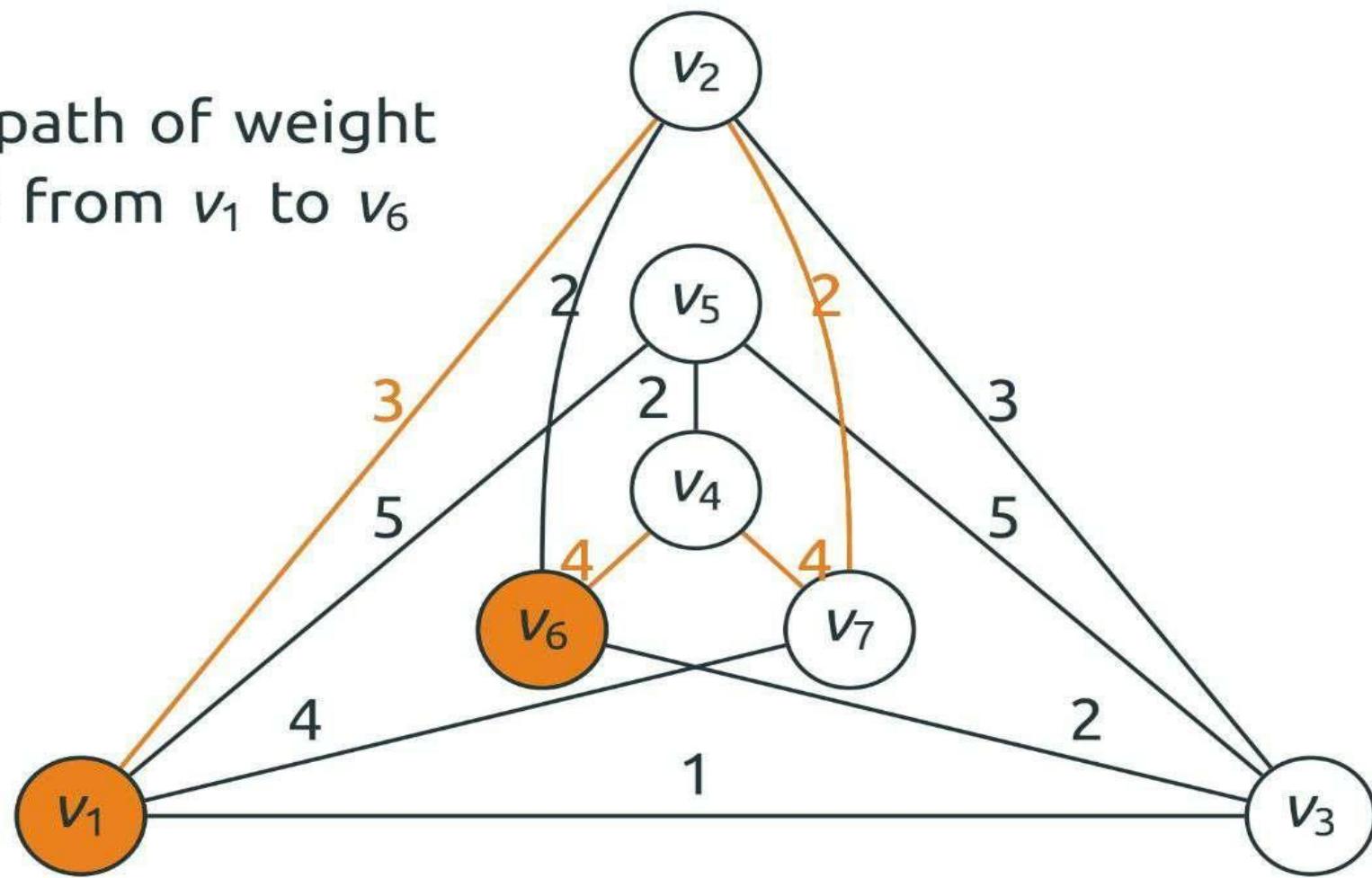
Weighted Paths: Examples

A path of weight
12 from v_1 to v_6



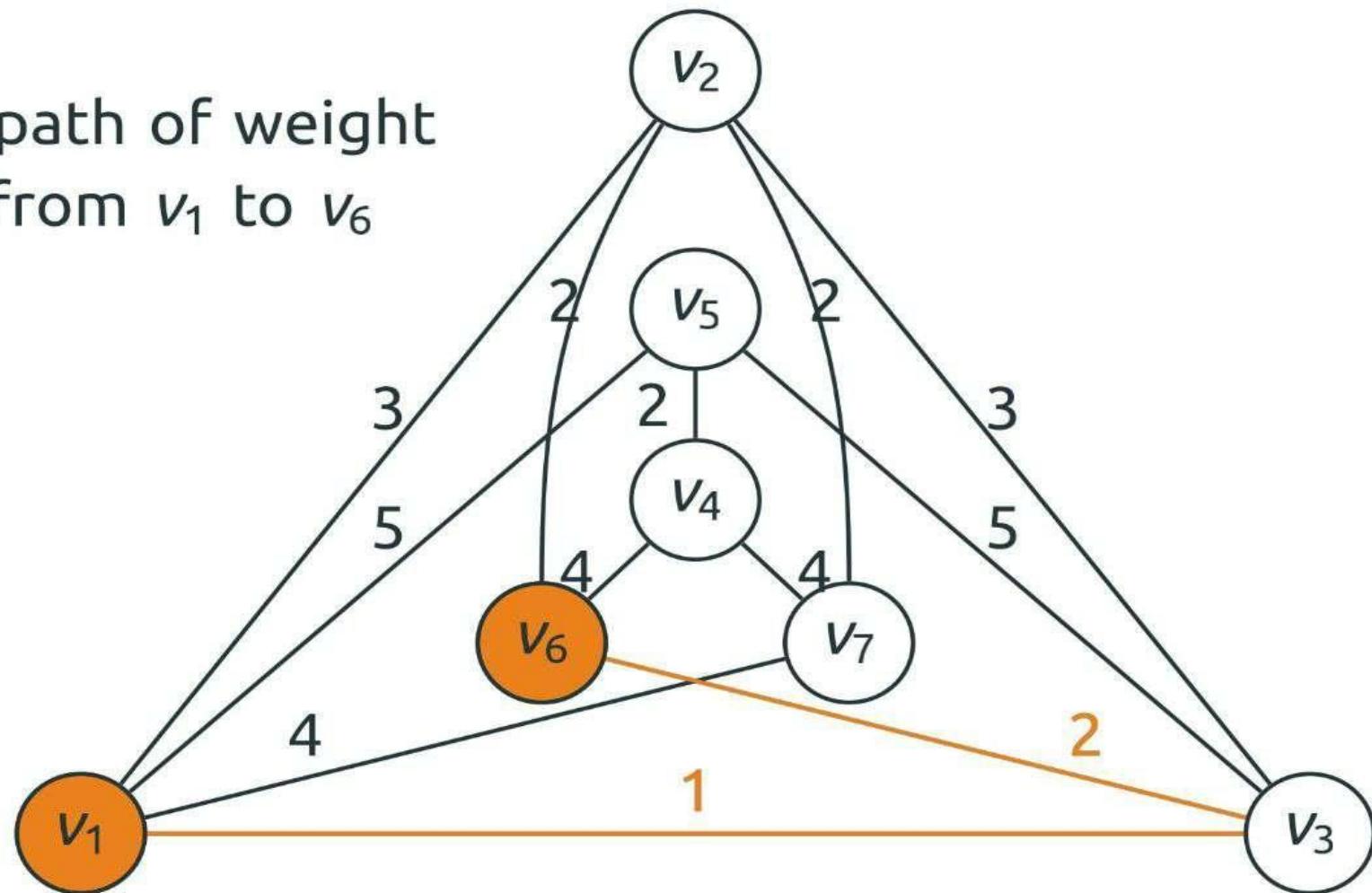
Weighted Paths: Examples

A path of weight
13 from v_1 to v_6



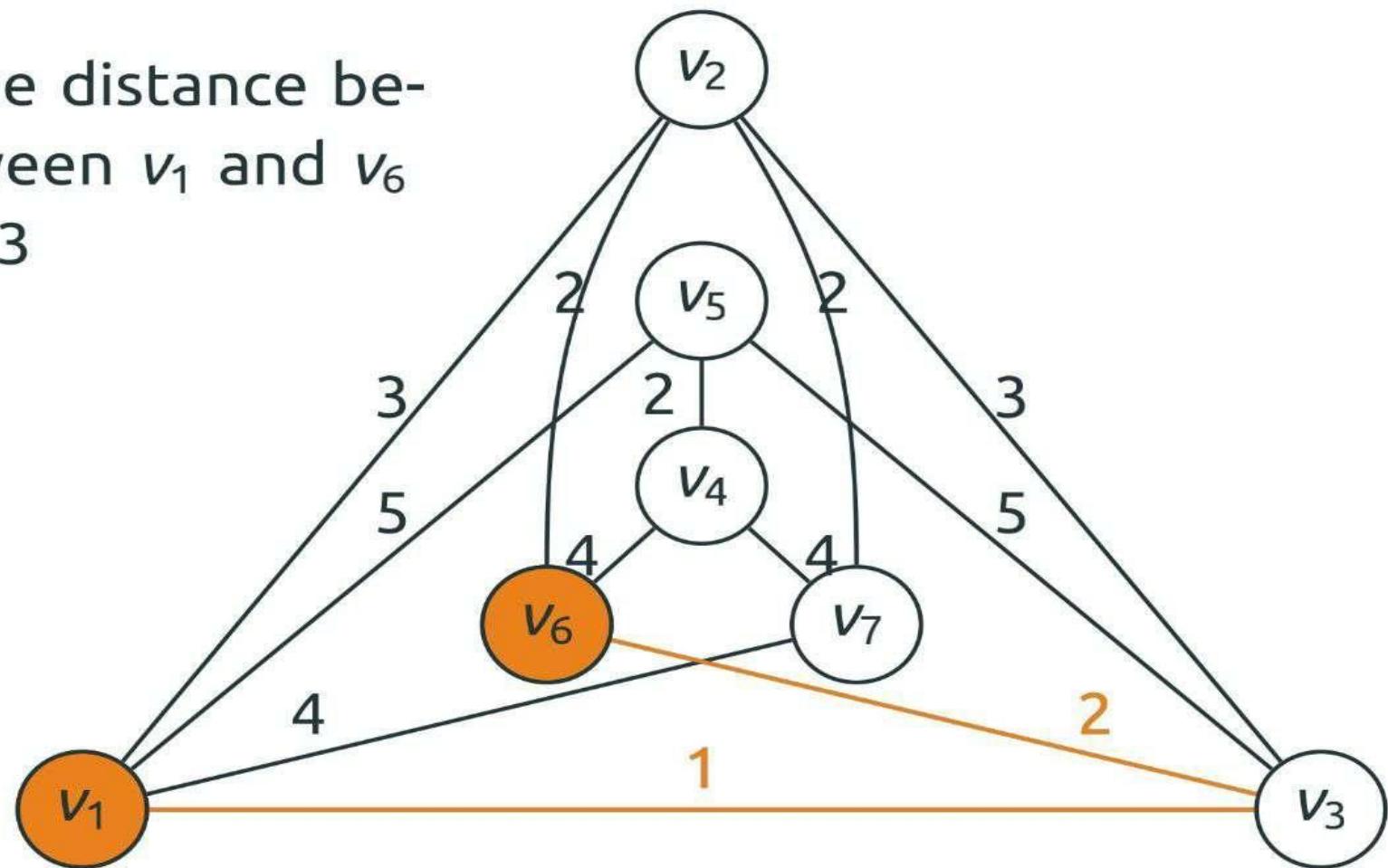
Weighted Paths: Examples

A path of weight
3 from v_1 to v_6



Weighted Paths: Examples

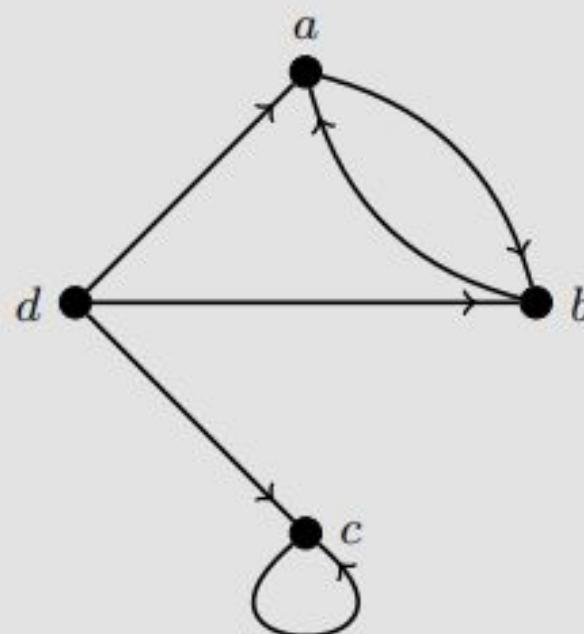
The distance between v_1 and v_6 is 3



LECTURE # 03

Definition 1.6 A *directed graph*, or *digraph*, is a graph $G = (V, A)$ that consists of a vertex set $V(G)$ and an *arc set* $A(G)$. An *arc* is an ordered pair of vertices.

Example 1.6 Let G_5 be a digraph where $V(G_5) = \{a, b, c, d\}$ and $A(G_5) = \{ab, ba, cc, dc, db, da\}$. A drawing of G_5 is given below.



Path Graphs

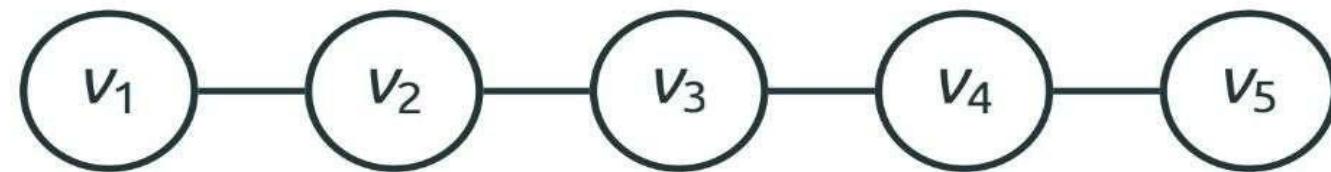
The Path Graph P_n , $n \geq 2$, consists of n vertices v_1, \dots, v_n and $n - 1$ edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$



Path Graphs

The Path Graph P_n , $n \geq 2$, consists of n vertices v_1, \dots, v_n and $n - 1$ edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$

The Graph P_5



Path Graphs

The **Path Graph** P_n , $n \geq 2$, consists of n vertices v_1, \dots, v_n and $n - 1$ edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$

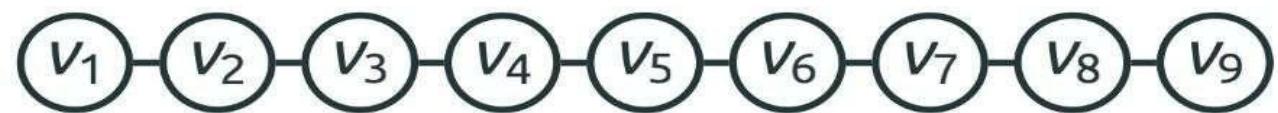
The Graph P_2



Path Graphs

The **Path Graph** P_n , $n \geq 2$, consists of n vertices v_1, \dots, v_n and $n - 1$ edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$

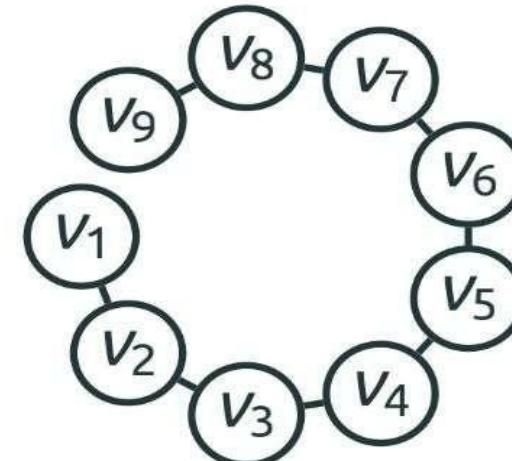
The Graph P_9



Path Graphs

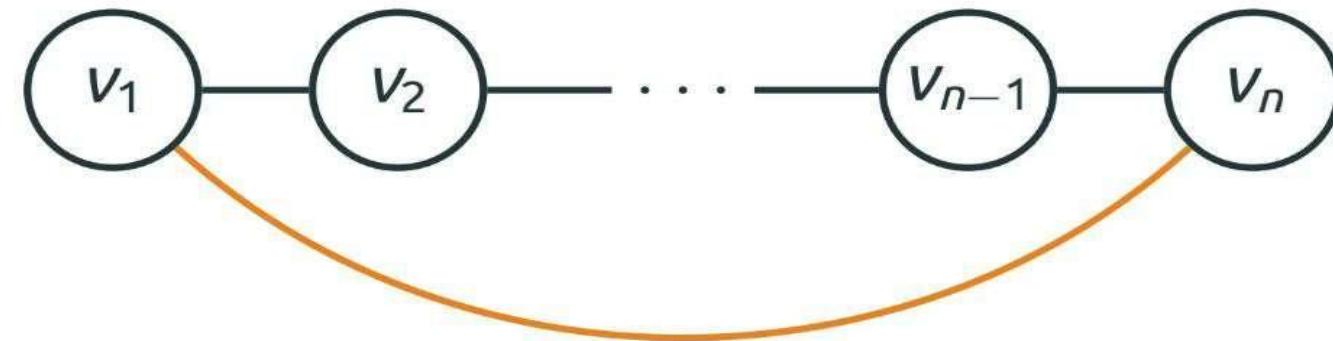
The **Path Graph** P_n , $n \geq 2$, consists of n vertices v_1, \dots, v_n and $n - 1$ edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$

The Graph P_9



Cycle Graphs

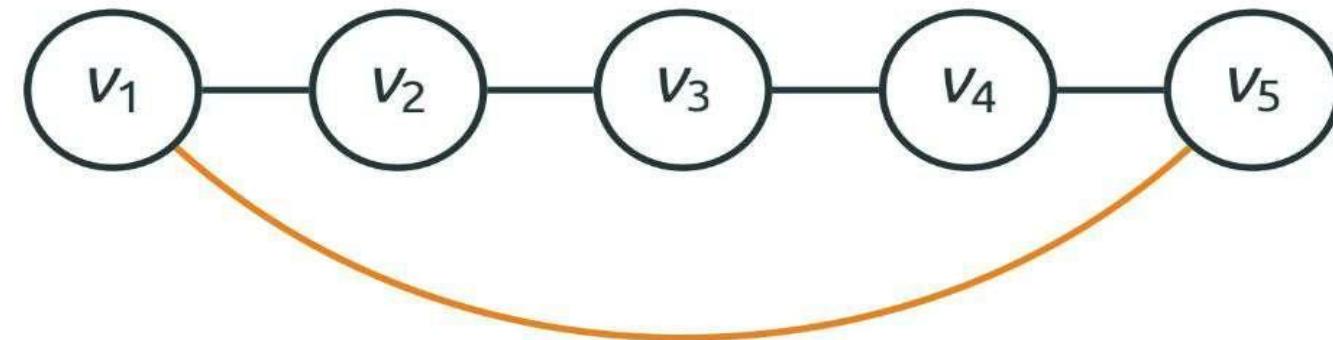
The **Cycle Graph** C_n , $n \geq 3$, consists of n vertices v_1, \dots, v_n and n edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$



Cycle Graphs

The **Cycle Graph C_n** , $n \geq 3$, consists of n vertices v_1, \dots, v_n and n edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$

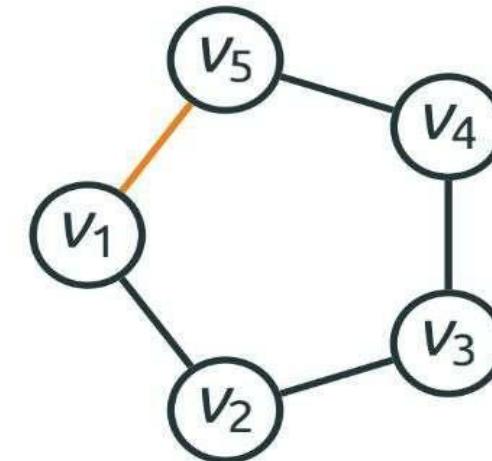
The Graph C_5



Cycle Graphs

The **Cycle Graph** C_n , $n \geq 3$, consists of n vertices v_1, \dots, v_n and n edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$

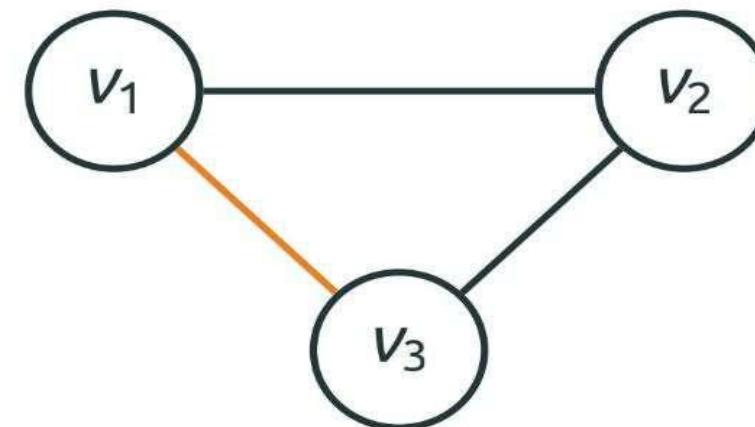
The Graph C_5



Cycle Graphs

The **Cycle Graph** C_n , $n \geq 3$, consists of n vertices v_1, \dots, v_n and n edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$

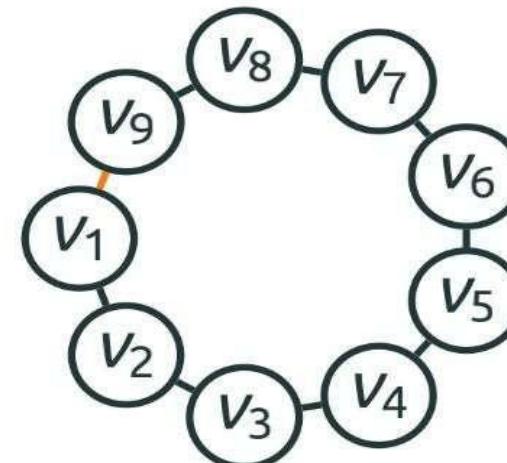
The Graph C_3



Cycle Graphs

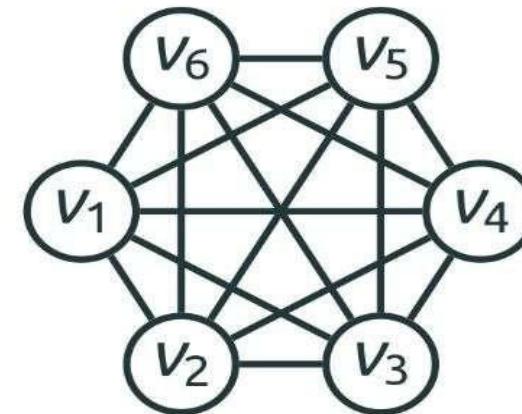
The **Cycle Graph** C_n , $n \geq 3$, consists of n vertices v_1, \dots, v_n and n edges $\{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$

The Graph C_9



Complete Graphs

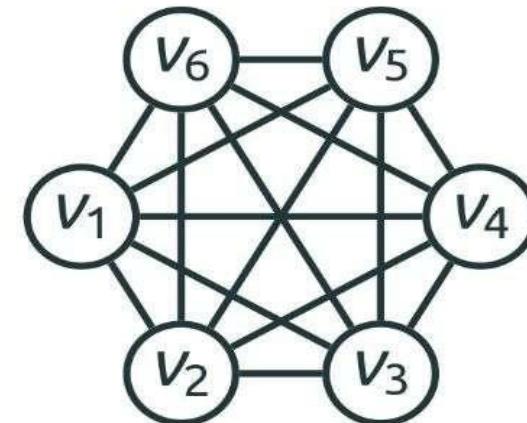
The **Complete Graph (Clique) K_n** , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)



Complete Graphs

The **Complete Graph (Clique) K_n** , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)

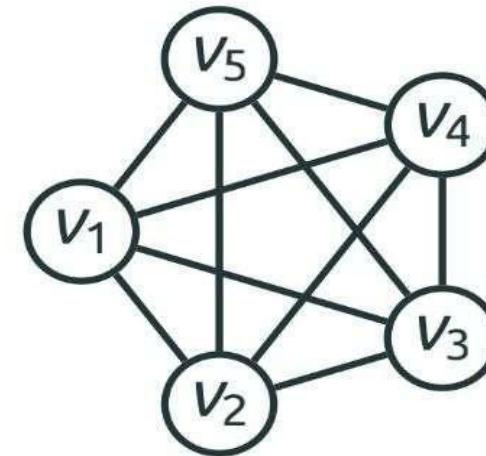
The Graph K_6



Complete Graphs

The **Complete Graph (Clique) K_n** , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)

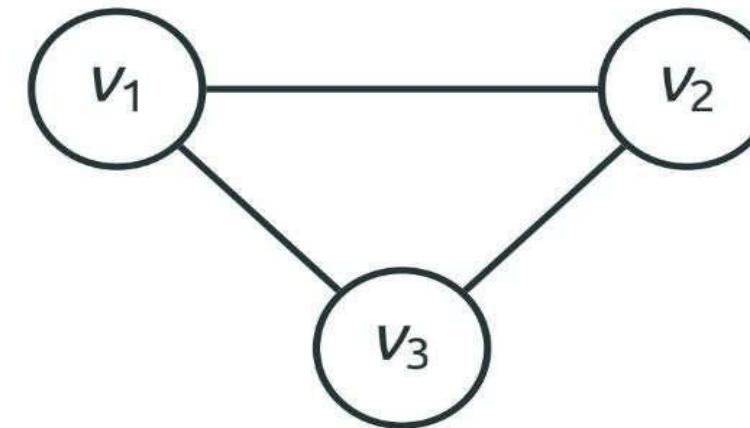
The Graph K_5



Complete Graphs

The **Complete Graph (Clique) K_n** , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)

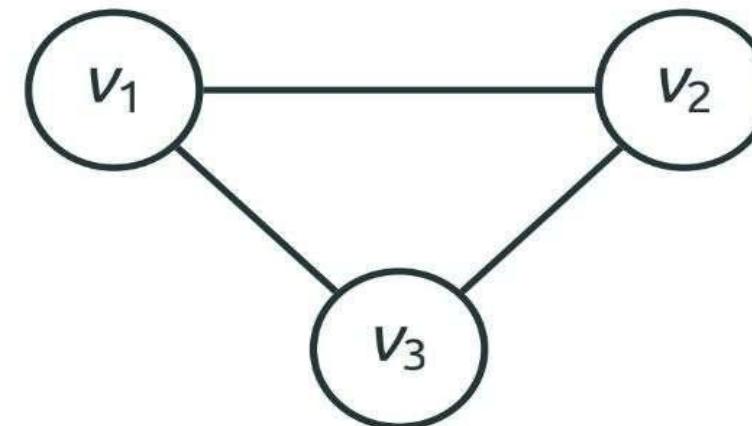
The Graph K_3



Complete Graphs

The **Complete Graph (Clique)** K_n , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)

The Graph $K_3 = C_3$



Complete Graphs

The **Complete Graph (Clique) K_n** , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)

The Graph K_2



Complete Graphs

The **Complete Graph (Clique)** K_n , $n \geq 2$, contains n vertices v_1, \dots, v_n and all edges between them ($n(n - 1)/2$ edges)

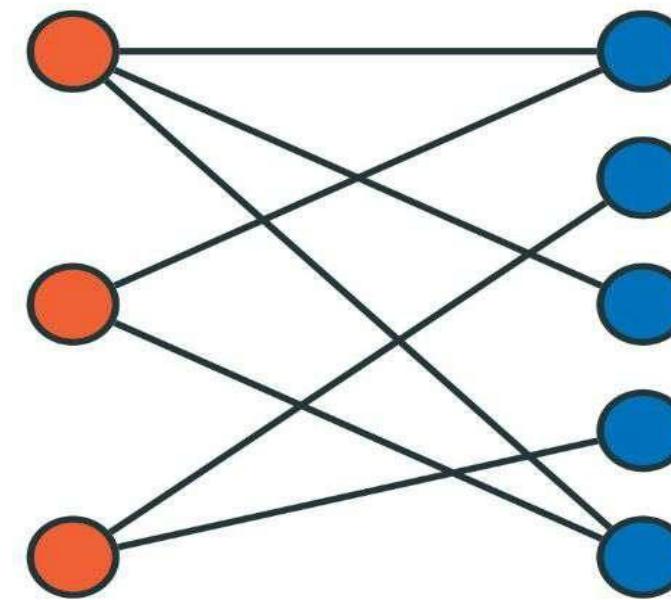
The Graph $K_2 = P_2$



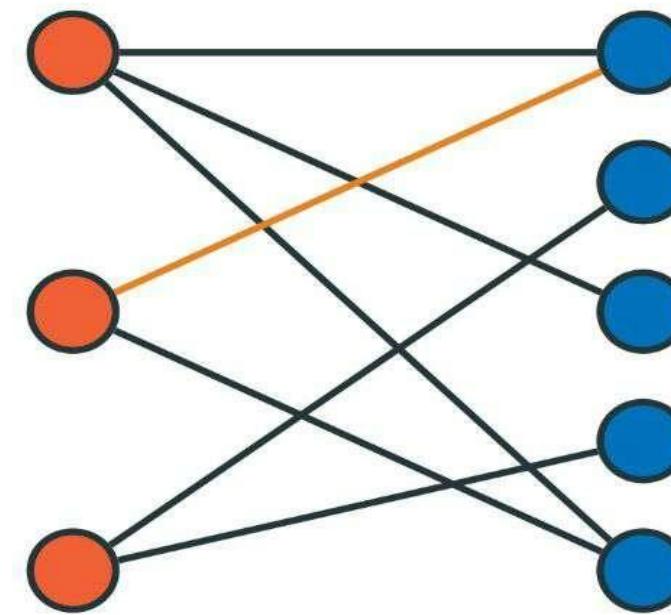
Bipartite Graphs

- A graph G is **Bipartite** if its vertices can be partitioned into **two disjoint sets** L and R such that
 - Every edge of G connects a vertex in L to a vertex in R
 - I.e., no edge connects two vertices from the same part
- L and R are called the **parts** of G

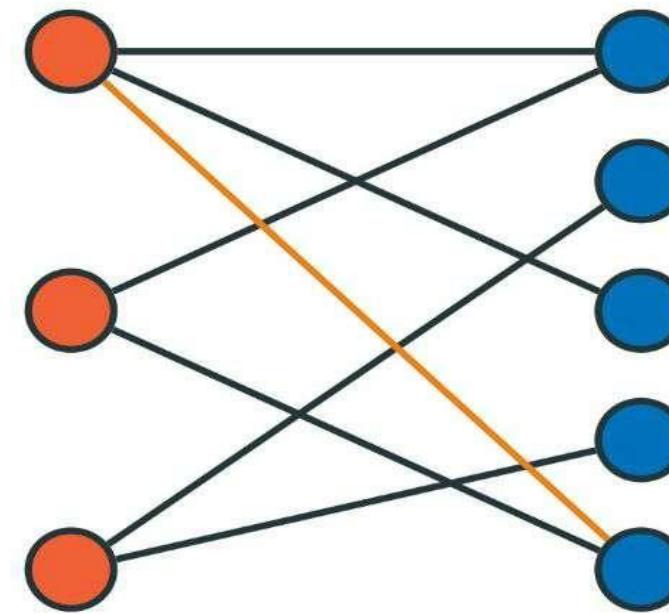
Bipartite Graphs: Examples



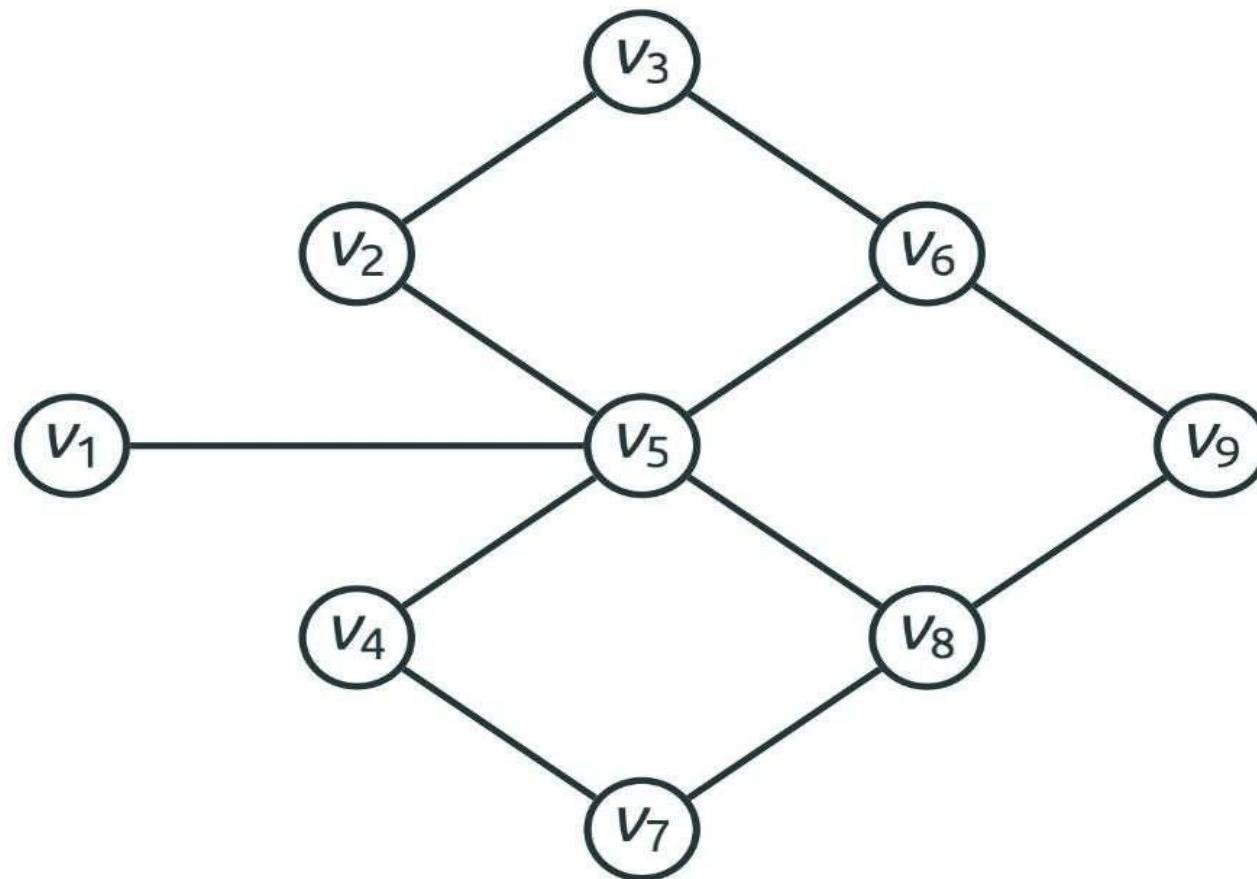
Bipartite Graphs: Examples



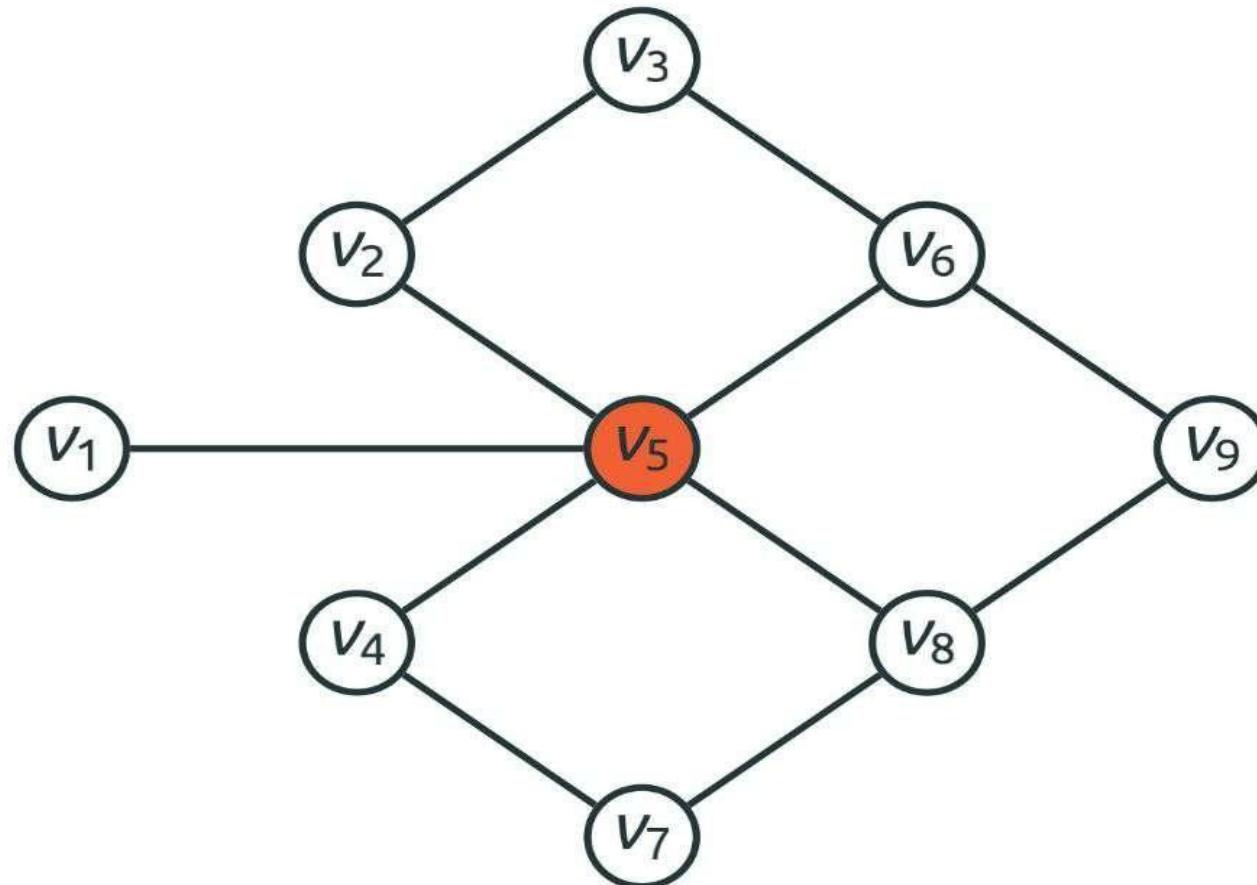
Bipartite Graphs: Examples



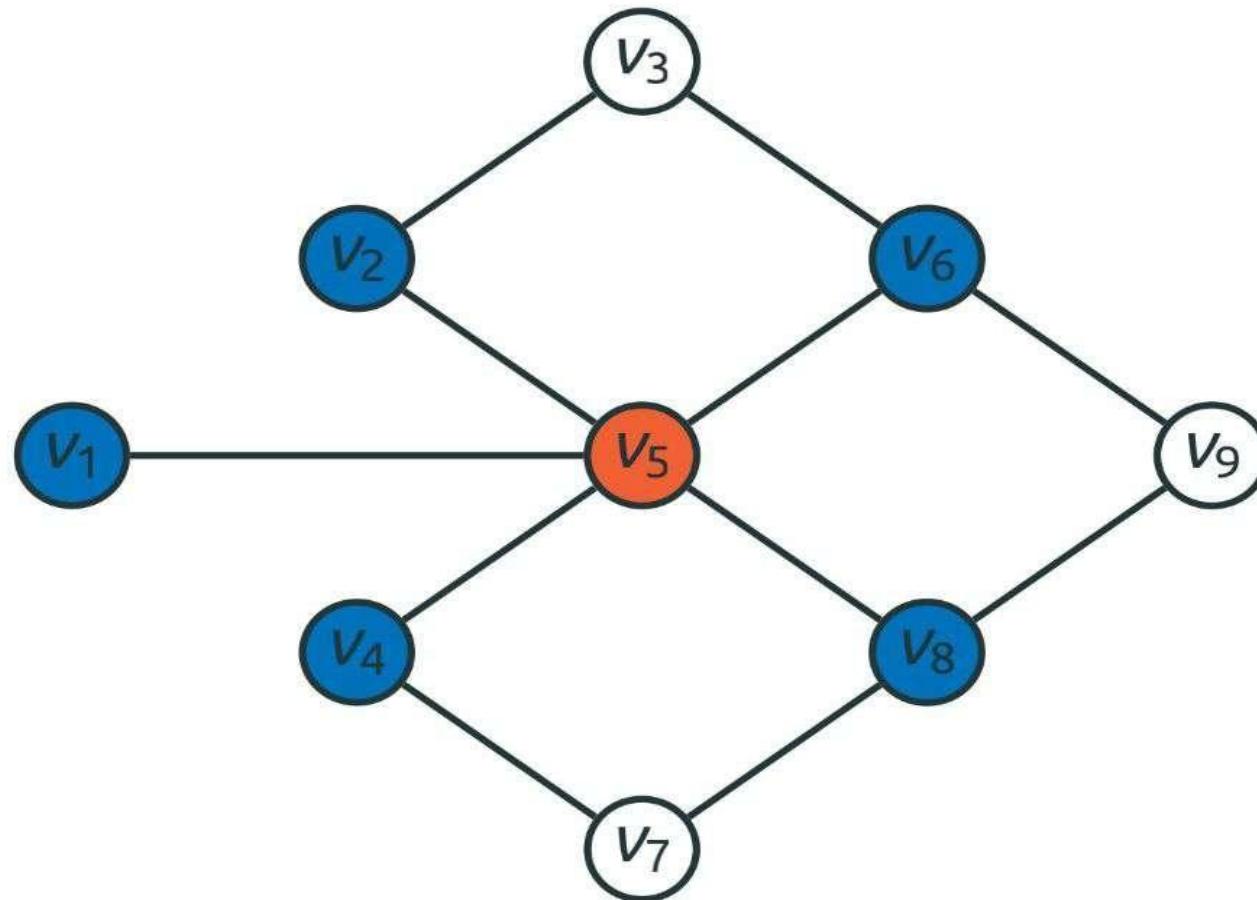
Bipartite Graphs: Examples



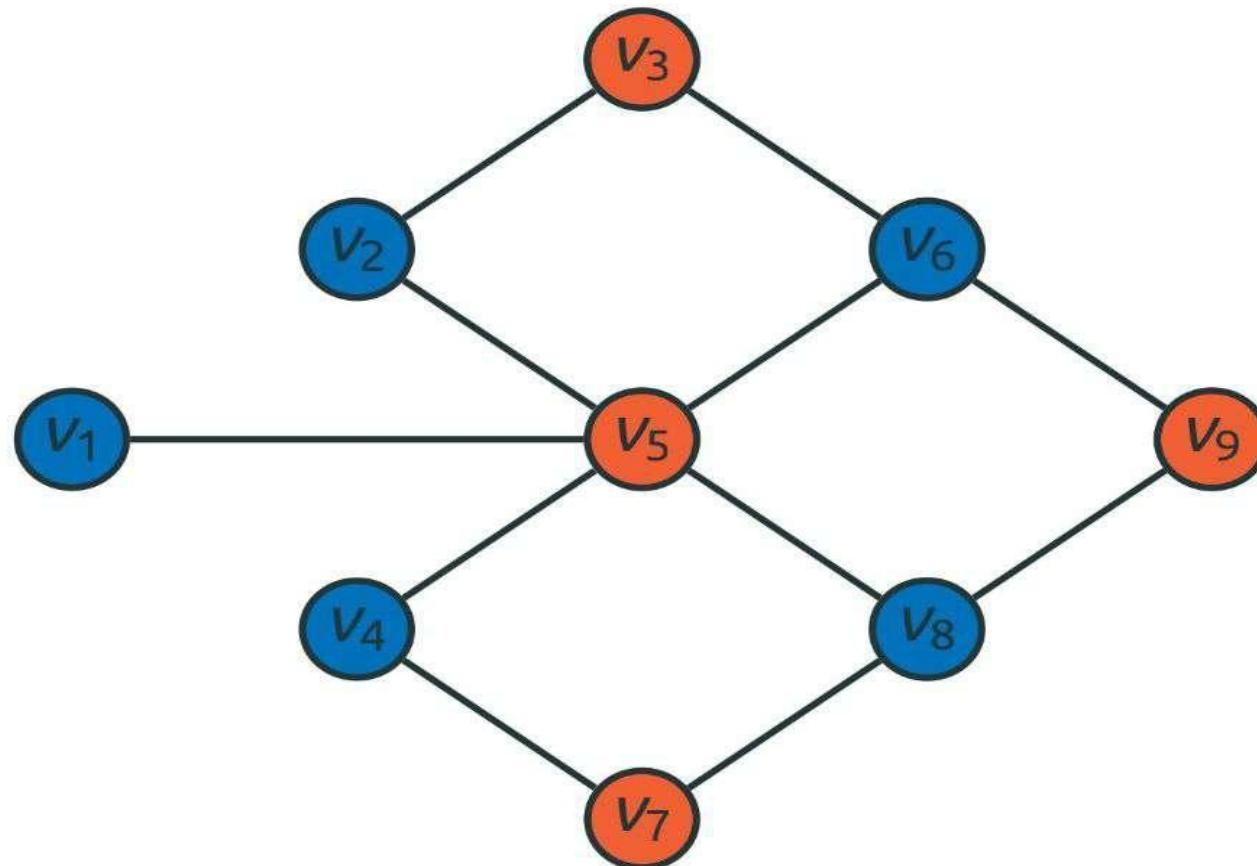
Bipartite Graphs: Examples



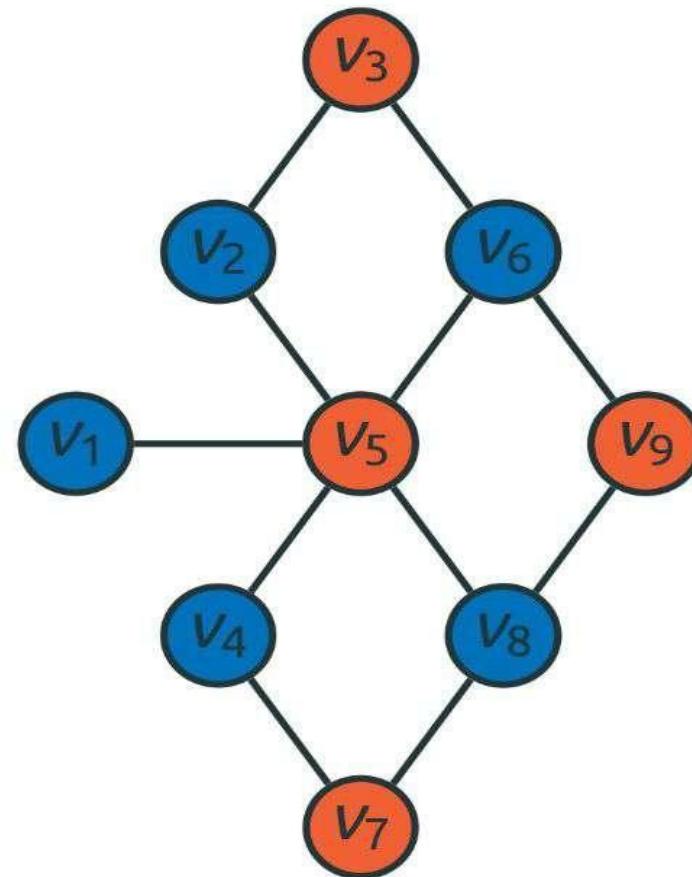
Bipartite Graphs: Examples



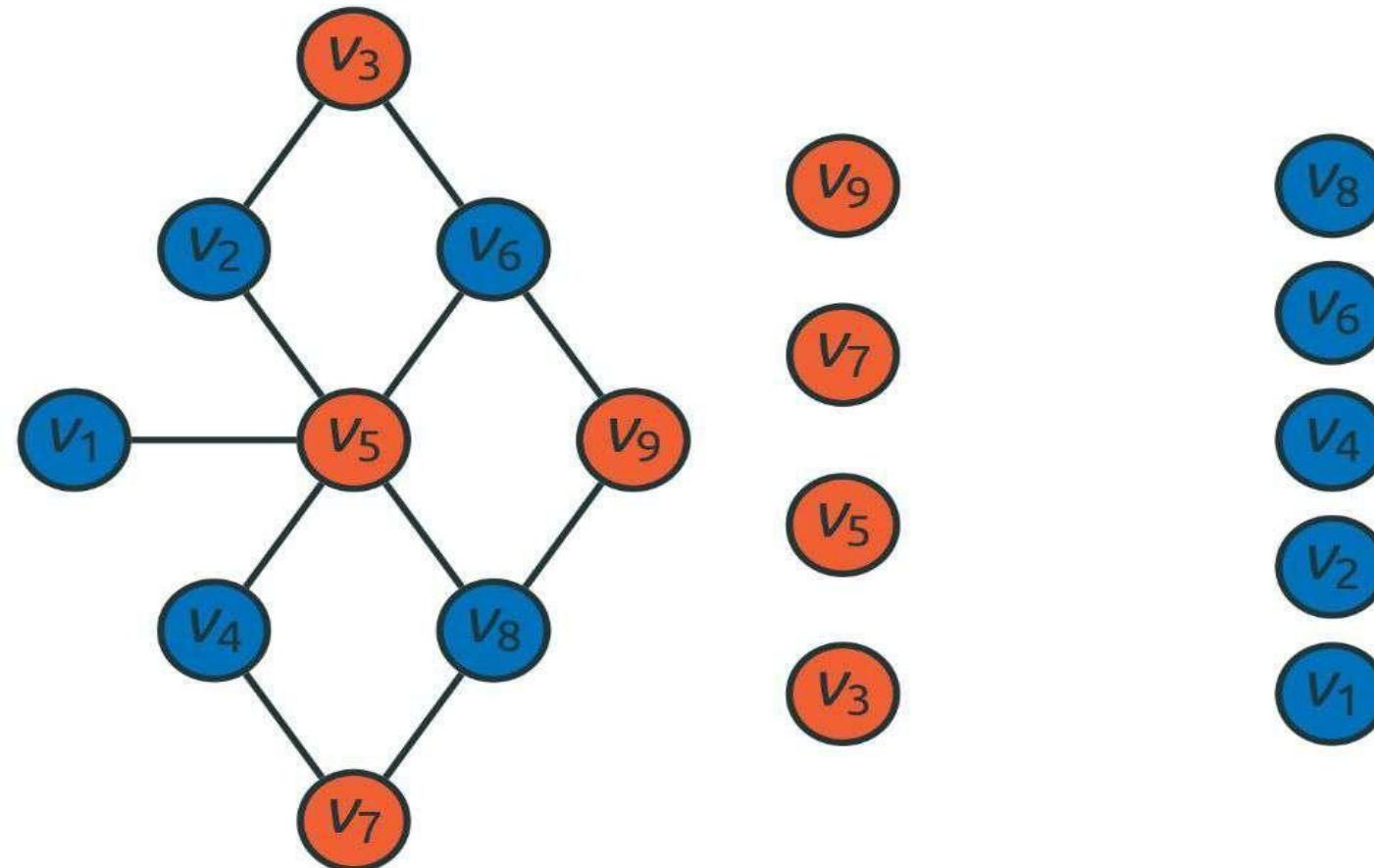
Bipartite Graphs: Examples



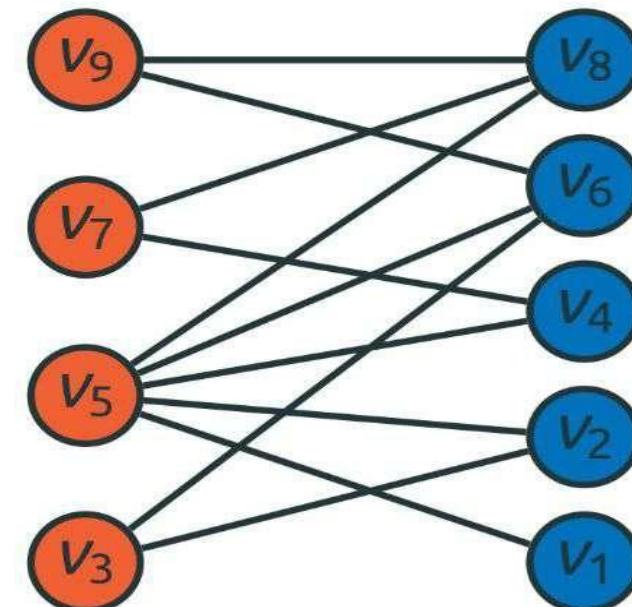
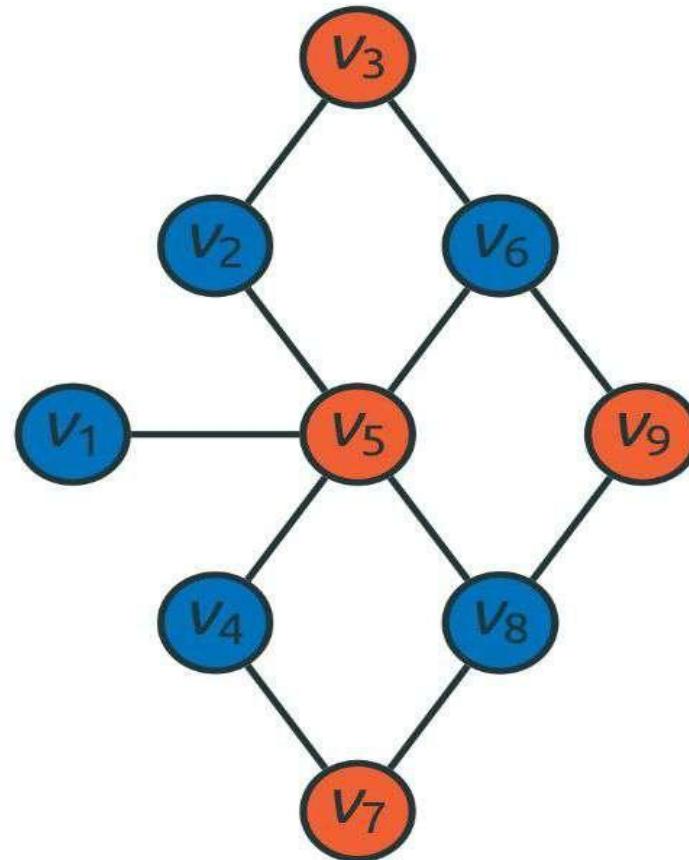
Bipartite Graphs: Examples



Bipartite Graphs: Examples

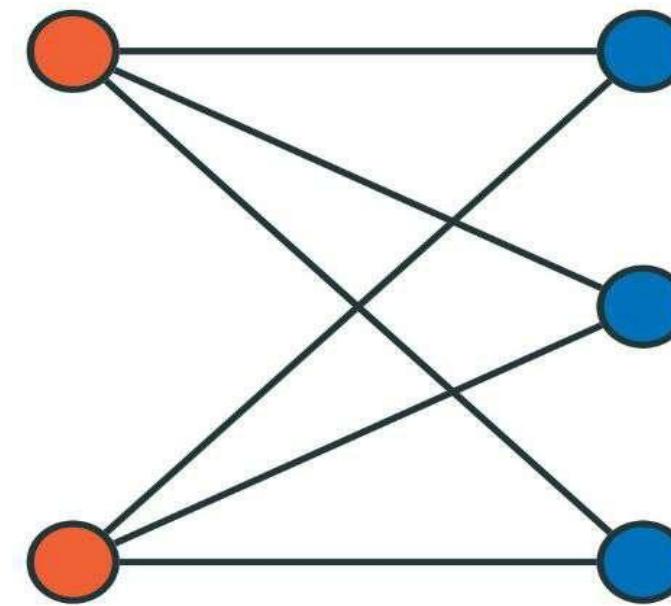


Bipartite Graphs: Examples



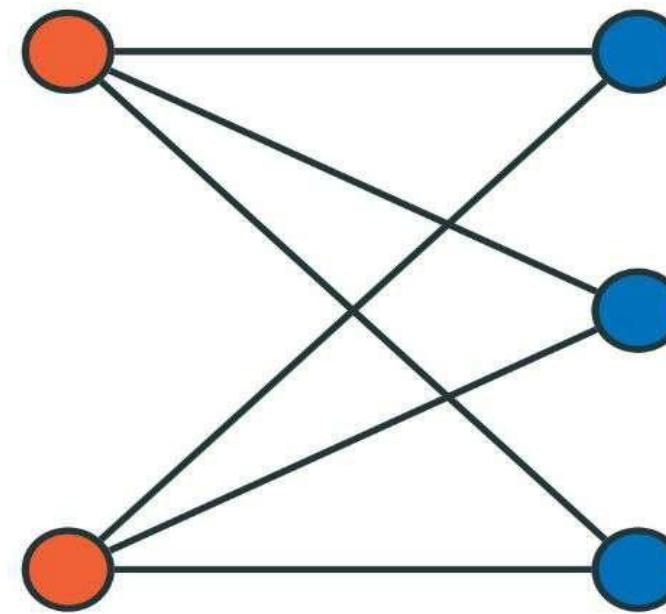
Complete Bipartite Graphs

Complete bipartite graph

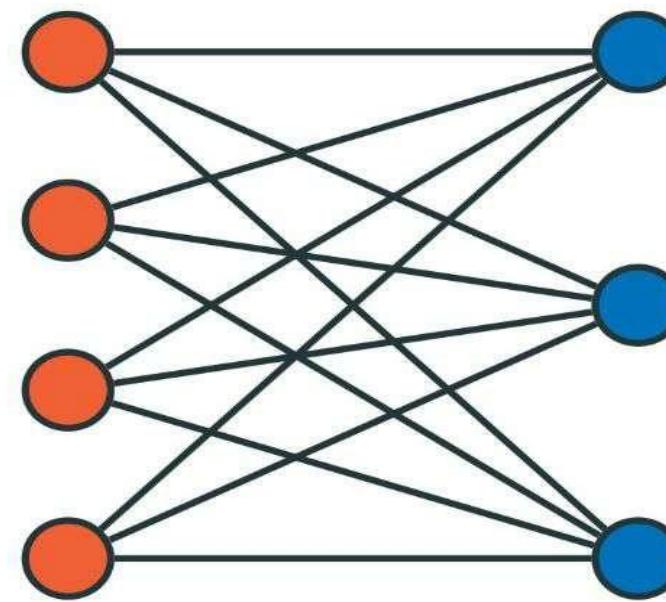


Complete Bipartite Graphs

Complete bipartite graph $K_{2,3}$

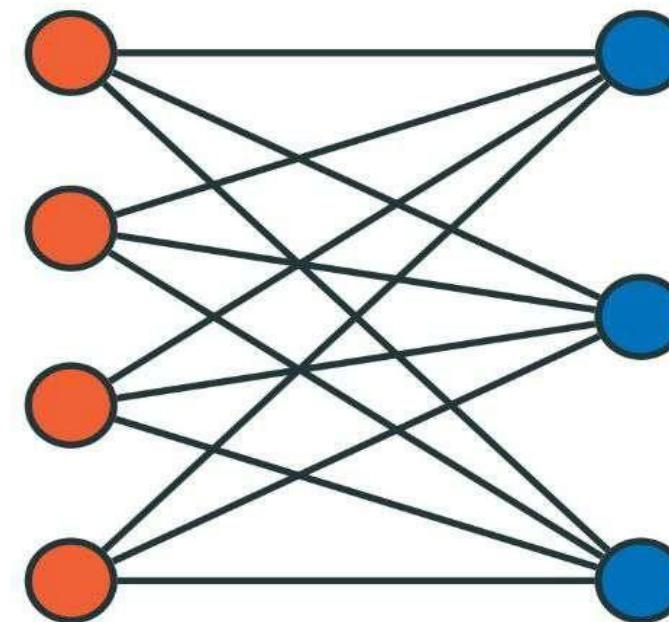


Complete Bipartite Graphs



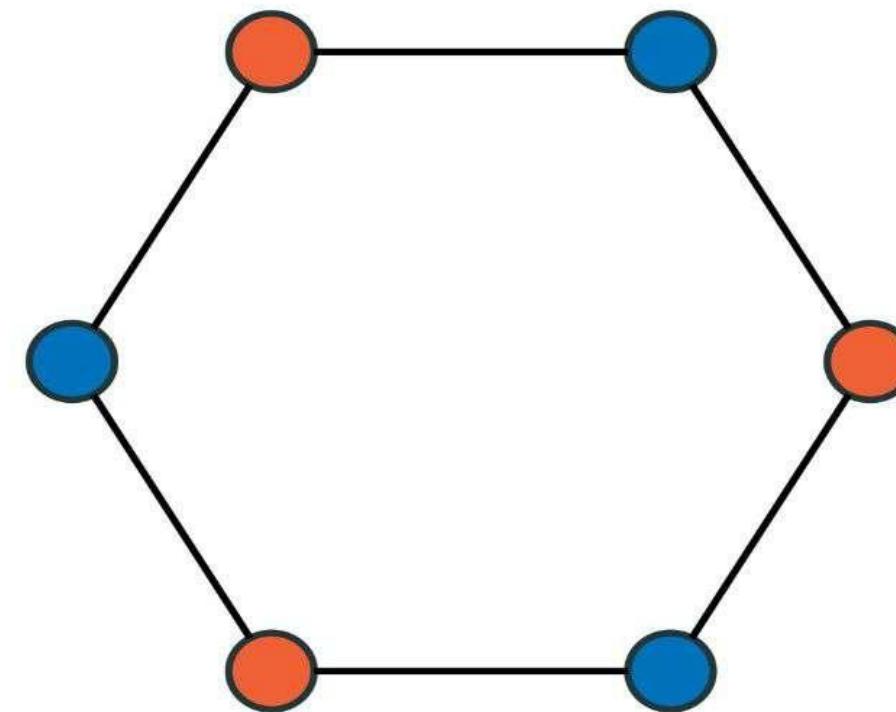
Complete Bipartite Graphs

Complete bipartite graph $K_{4,3}$



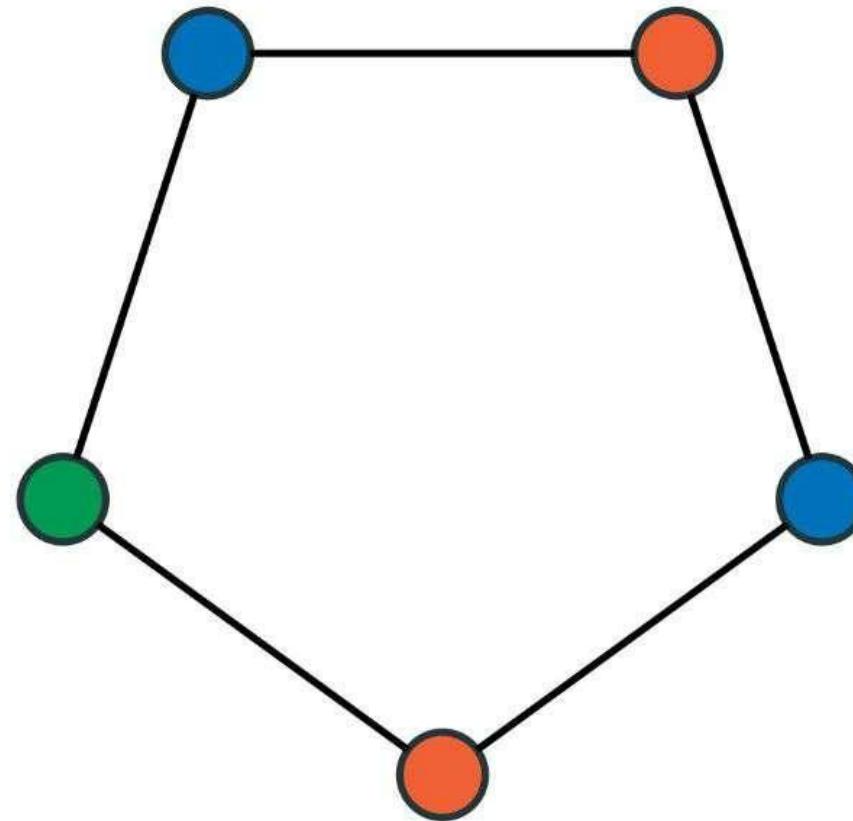
Cycle Graphs

For even n , C_n is **bipartite**



Cycle Graphs

For odd $n > 2$, C_n is **not bipartite**



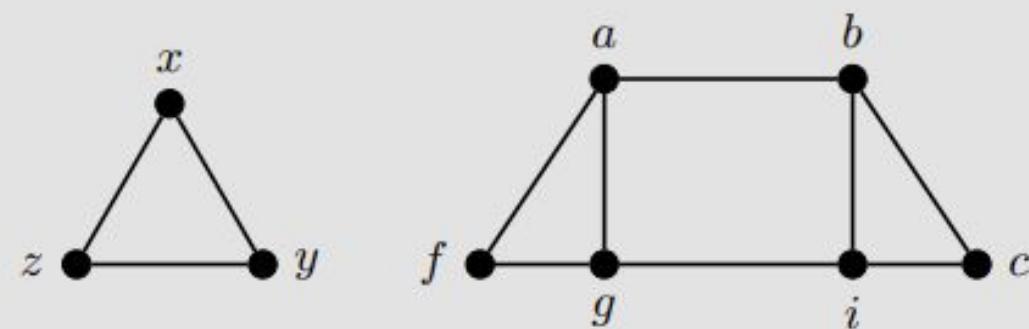
Graph Combinations

Definition 1.15 Given two graphs G and H the ***union*** $G \cup H$ is the graph with vertex-set $V(G) \cup V(H)$ and edge-set $E(G) \cup E(H)$.

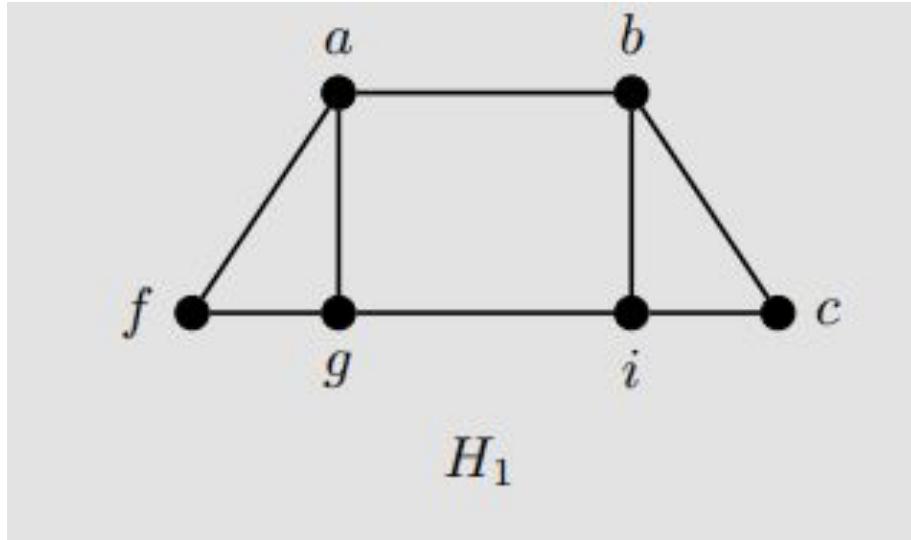
If the vertex-sets are disjoint (that is $V(G) \cap V(H) = \emptyset$) then we call the disjoint union the ***sum***, denoted $G + H$.

Example 1.12 Find the sum $K_3 + H_1$ and the union $H_1 \cup H_4$ using the graphs from Examples 1.3 and 1.4.

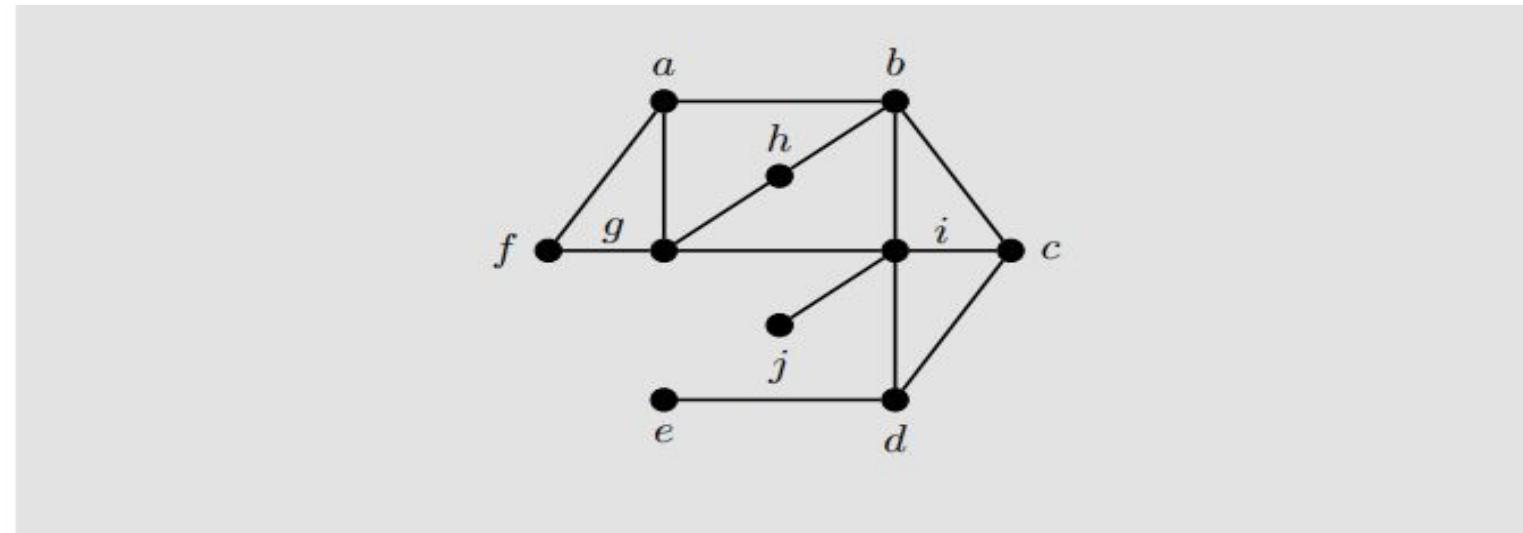
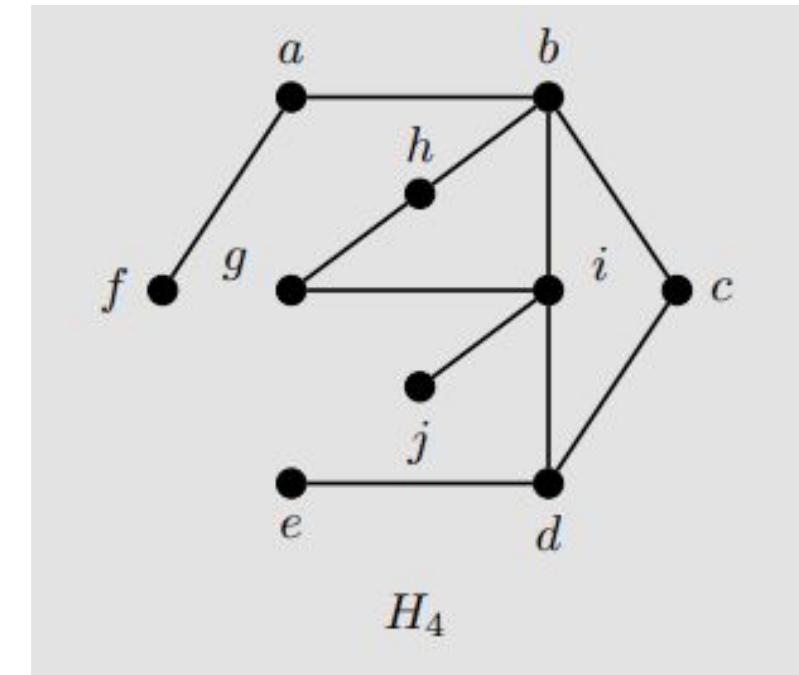
Solution: First note that, since we are finding the sum $K_3 + H_1$, we are assuming the vertex sets are disjoint. Thus the resulting graph is simply the graph below.



Next, since H_1 and H_4 are subgraphs of the same graph and have some edges in common, their union will consist of all the edges in at least one of H_1 and H_4 , where we do not draw (or list) an edge twice if it appears in both graphs, as shown in the following graph.

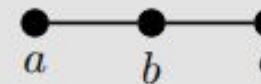


U

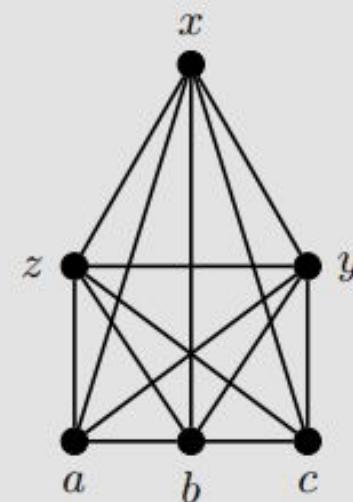


Definition 1.16 The *join* of two graphs G and H , denoted $G \vee H$, is the sum $G + H$ together with all edges of the form xy where $x \in V(G)$ and $y \in V(H)$.

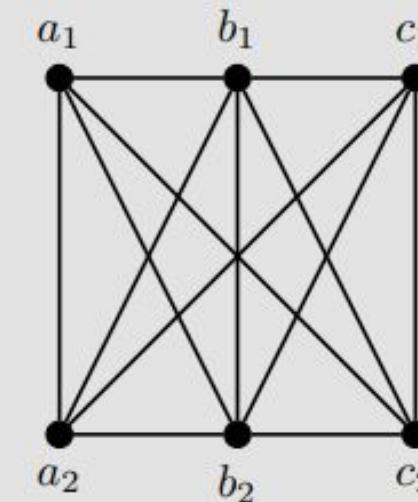
Example 1.13 Find the join of K_3 and the graph G below consisting of three vertices and two edges, as well as the join $G \vee G$.



Solution: The join $K_3 \vee G$ is shown below on the left. Note that every vertex from K_3 is adjacent to all those from G , but this is not K_6 since the edge ac is missing. The join $G \vee G$ is on the right below.



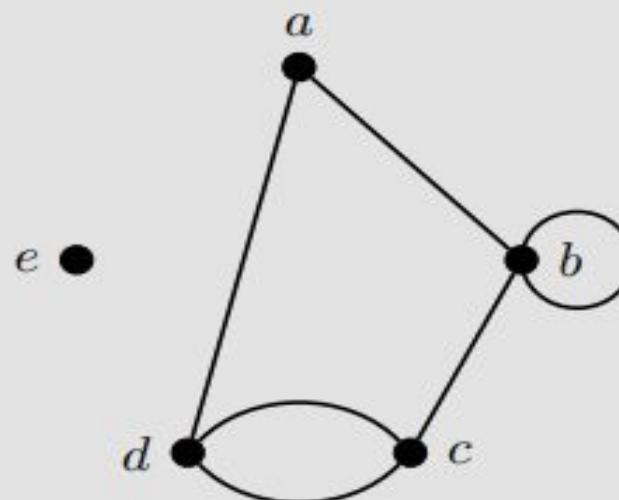
$K_3 \vee G$



$G \vee G$

Quick Review

Example 1.1 Let G_4 be a graph where $V(G_4) = \{a, b, c, d, e\}$ and $E(G_4) = \{ab, cd, cd, bb, ad, bc\}$. Although G_4 is defined by these two sets, we generally use a visualization of the graph where a dot represents a vertex and an edge is a line connecting the two dots (vertices). A drawing of G_4 is given below.

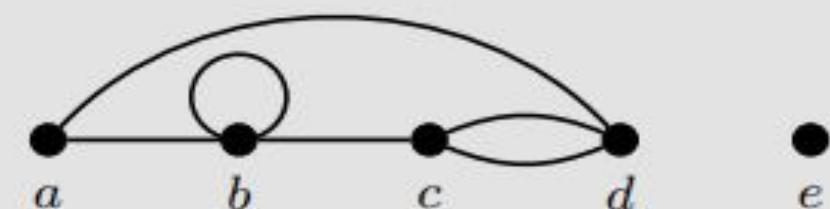
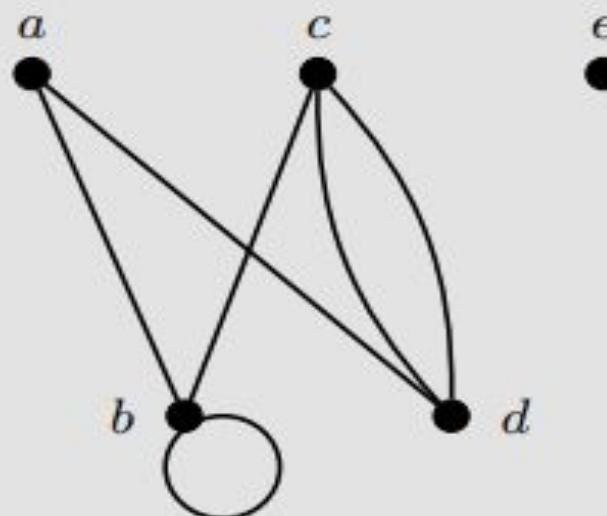


Note that two lines were drawn between vertices c and d as the edge cd is listed twice in the edge set. In addition, a circle was drawn at b to indicate an edge (bb) that starts and ends at the same vertex.

Definition 1.2 The number of vertices in a graph G is denoted $|V(G)|$, or more simply $|G|$. The number of edges is denoted $|E(G)|$ or $\|G\|$.

Definition 1.4 A *subgraph* H of a graph G is a graph where H contains some of the edges and vertices of G ; that is, $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Example 1.2 Consider the graph G_4 from Example 1.1. Below are two different drawings of G_4 .



To verify that these drawings represent the same graph from Example 1.1, we should check the relationships arising from the vertex set and edge set. For example, there are two edges between vertices c and d , a loop at b , and no edges at e . You should verify the remaining edges.

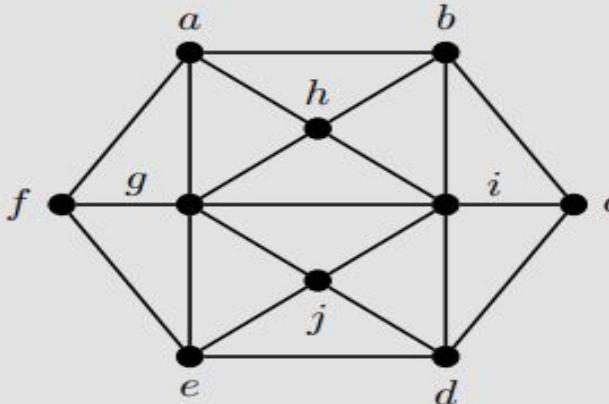


Definition 1.3 Let G be a graph.

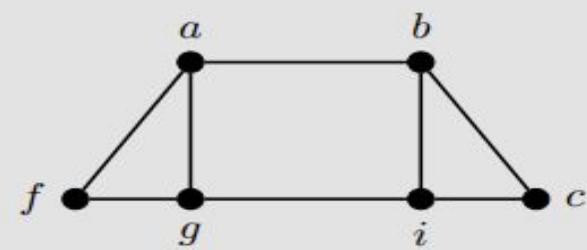
- If xy is an edge, then x and y are the ***endpoints*** for that edge. We say x is ***incident to*** edge e if x is an endpoint of e .
- If two vertices are incident to the same edge, we say the vertices are ***adjacent***, denoted $x \sim y$. Similarly, if two edges share an endpoint, we say they are adjacent. If two vertices are adjacent, we say they are ***neighbors*** and the set of all neighbors of a vertex x is denoted $N(x)$.
 - ab and ad are adjacent edges in G_4 since they share an endpoint, namely vertex a

- $a \sim b$, that is a and b are adjacent vertices as ab is an edge of G_4
- $N(d) = \{a, c\}$ and $N(b) = \{a, b, c\}$
- If two vertices (or edges) are not adjacent then we call them ***independent***.
- If a vertex is not incident to any edge, we call it an ***isolated vertex***.
 - e is an isolated vertex of G_4
- If both endpoints of an edge are the same vertex, then we say the edge is a ***loop***.
 - bb is a loop in G_4
- If there is more than one edge with the same endpoints, we call these ***multi-edges***.
 - cd is a multi-edge of G_4
- If a graph has no multi-edges or loops, we call it ***simple***.
- The ***degree*** of a vertex v , denoted $\deg(v)$, is the number of edges incident to v , with a loop adding two to the degree. If the degree is even, the vertex is called ***even***; if the degree is odd, then the vertex is ***odd***.
 - $\deg(a) = 2$, $\deg(b) = 4$, $\deg(c) = 3$, $\deg(d) = 3$, $\deg(e) = 0$
- If all vertices in a graph G have the same degree k , then G is called a ***k-regular*** graph. When $k = 3$, we call the graph ***cubic***.

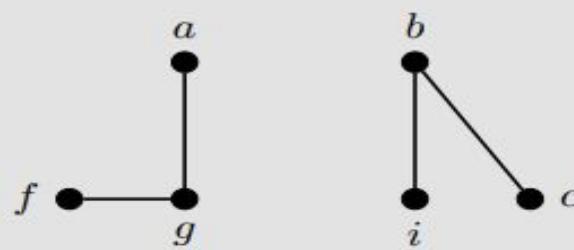
Example 1.3 Consider the graph G below. Find two subgraphs of G , both of which have vertex set $V' = \{a, b, c, f, g, i\}$.



Solution: Two possible solutions are shown below. Note that the graph H_1 on the left contains every edge from G amongst the vertices in V' , whereas the graph H_2 on the right does not since some of the available edges are missing (namely, ab , af , ci , and gi).



H_1



H_2

The graph shown on the left above is a special type, called an **induced subgraph**, since all the edges are present between the chosen vertices. Another special type of subgraph, called a **spanning subgraph**, includes all the vertices of the original graph.

Definition 1.5 Given a graph $G = (V, E)$, an *induced subgraph* is a subgraph $G[V']$ where $V' \subseteq V$ and every available edge from G between the vertices in V' is included.

We say H is a *spanning subgraph* if it contains all the vertices but not necessarily all the edges of G ; that is, $V(H) = V(G)$ and $E(H) \subseteq E(G)$.

Definition 1.8 A *weighted graph* $G = (V, E, w)$ is a graph where each of the edges has a real number associated with it. This number is referred to as the *weight* and denoted $w(xy)$ for the edge xy .

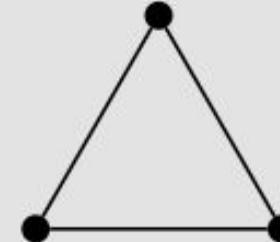
See Example # 1.8

Definition 1.9 A simple graph G is *complete* if every pair of distinct vertices is adjacent. The complete graph on n vertices is denoted K_n .

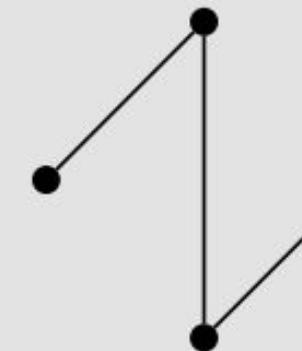
Properties of K_n

- (1) Each vertex in K_n has degree $n - 1$.
- (2) K_n has $\frac{n(n - 1)}{2}$ edges.
- (3) K_n contains the most edges out of all simple graphs on n vertices.

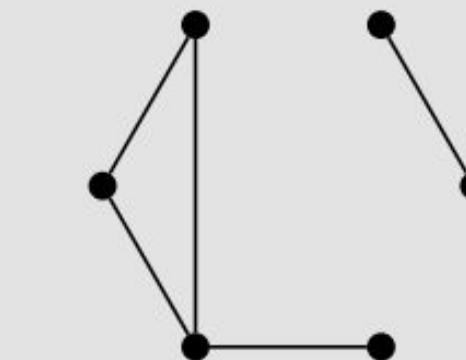
Example 1.10 Find the complements of each graph shown below.



G_1



G_2

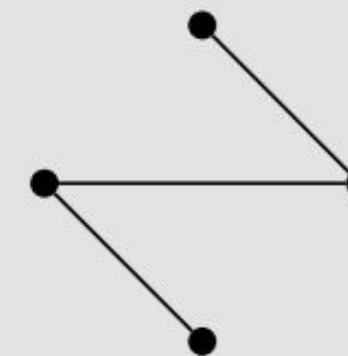


G_3

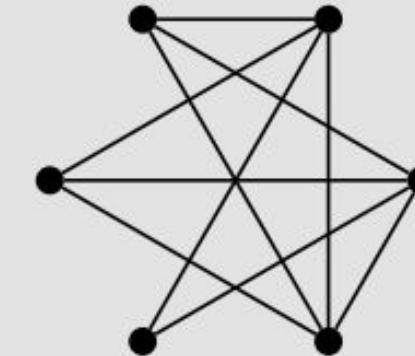
Solution: For each graph we simply add an edge where there wasn't one before and remove the current edges.



\overline{G}_1



\overline{G}_2



\overline{G}_3

Definition 1.12 A graph G is **bipartite** if the vertices can be partitioned into two sets X and Y so that every edge has one endpoint in X and the other in Y .

Definition 1.13 $K_{m,n}$ is the **complete bipartite graph** where $|X| = m$ and $|Y| = n$ and every vertex in X is adjacent to every vertex in Y .

NOTE:

A **directed graph** or **digraph** consists of a finite set V of vertices and a set A of *ordered pairs* of distinct vertices called **arcs**. If the ordered pair $\{u, v\}$ is an arc a , we say that the arc a is **directed from u to v** . In this context, arc a is **adjacent from** vertex u and is **adjacent to** vertex v . In a **mixed graph**, there will be at least one edge and at least one arc. If each arc of a digraph is replaced by an edge, the resulting structure is a graph known as the **underlying graph** of the digraph. On the other hand, if each edge of a simple graph is replaced by an arc, the resulting structure is a digraph known as an **orientation** of the simple graph. Any orientation of a complete graph is known as a **tournament**.

SUMMARY OF WEEK # 01

A **graph** G consists of a set V of **vertices** and a collection E (not necessarily a set) of unordered pairs of vertices called **edges**. A graph is symbolically represented as $G = (V, E)$. In this book, unless otherwise specified, both V and E are finite. The **order** of a graph is the number of its vertices, and its **size** is the number of its edges. If u and v are two vertices of a graph and if the *unordered pair* $\{u, v\}$ is an edge denoted by e , we say that e **joins** u and v or that it is an edge between u and v . In this case, the vertices u and v are said to be **incident on** e and e is **incident to** both u and v . Two or more edges that join the same pair of distinct vertices are called **parallel** edges. An edge represented by an unordered pair in which the two elements are not distinct is known as a **loop**. A graph with no loops is a **multigraph**. A graph with at least one loop is a **pseudograph**. A **simple graph** is a graph with no parallel edges and loops. The term *graph* is used in lieu of *simple graph* in many places in this book. The **complete graph** K_n is a graph with n vertices in which there is exactly one edge joining every pair of vertices. The graph K_1 with one vertex and no edge is known as the **trivial graph**. A **bipartite graph** is a simple graph in which the set of vertices can be partitioned into two sets X and Y such that every edge is between a vertex in X and a vertex in Y ; it is represented as $G = (X, Y, E)$. The **complete bipartite graph** $K_{m,n}$ is the graph (X, Y, E) with m vertices in X and n vertices in Y in which there is an edge between every vertex in X and every vertex in Y . The **union** of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G = G_1 \cup G_2 = (V, E)$, where V is the union of V_1 and V_2 and E is the union of E_1 and E_2 .

EX #: 1.8

Problems:

1.1-1.7, 1.12, 1.14, 1.15, 1.16, 1.17, 1.20, 1.22

LECTURE # 04 & 05

Theorem 1.33 Let $G = (V, A)$ be a digraph and $|A|$ denote the number of arcs in G . Then both the sum of the in-degrees of the vertices and the sum of the out-degrees equals the number of arcs; that is, if $V = \{v_1, v_2, \dots, v_n\}$, then

$$\begin{aligned}\deg^-(v_1) + \cdots + \deg^-(v_n) &= |A| \\ &= \deg^+(v_1) + \cdots + \deg^+(v_n)\end{aligned}$$

Isomorphisms

Two graphs G and H are said to be **isomorphic** if there exists a one-one and onto mapping $f : V(G) \rightarrow V(H)$ such that two vertices u, v are adjacent in G when and only when their images $f(u)$ and $f(v)$ under f are adjacent in H (i.e., **the adjacency is preserved under f**).

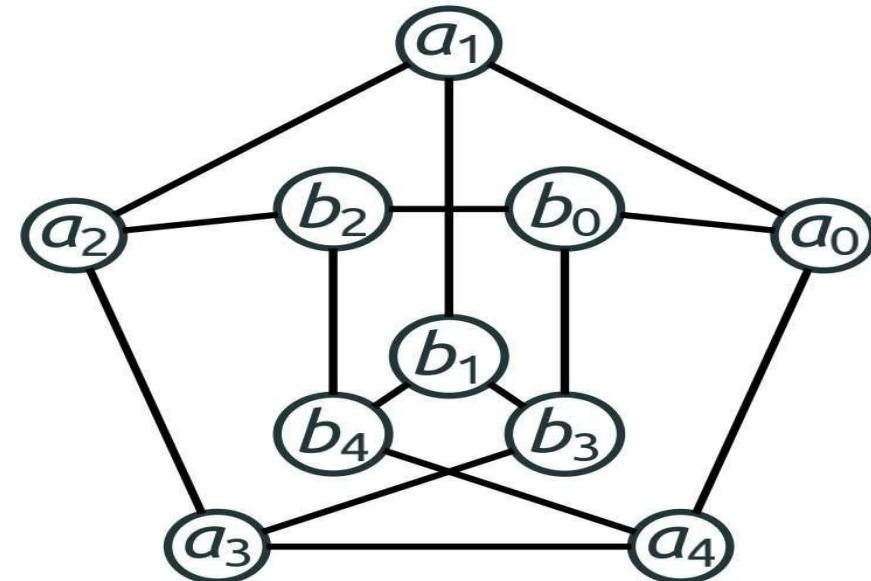
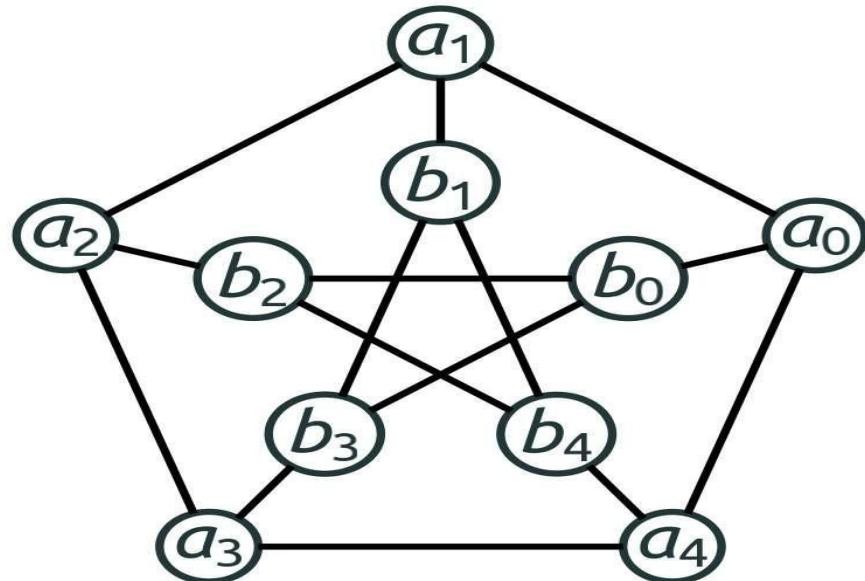
In this case, we shall write $G \cong H$ and call the mapping f an **isomorphism** from G to H .

- (1) The phrase ‘when and only when’ used above means that if u and v are adjacent in G , then $f(u)$ and $f(v)$ **must be** adjacent in H ; and if u and v are **not** adjacent in G , then $f(u)$ and $f(v)$ **must not be** adjacent in H .
- (2) The word **isomorphism** is derived from the Greek words *isos* (meaning ‘equal’) and *morphe* (meaning ‘form’).

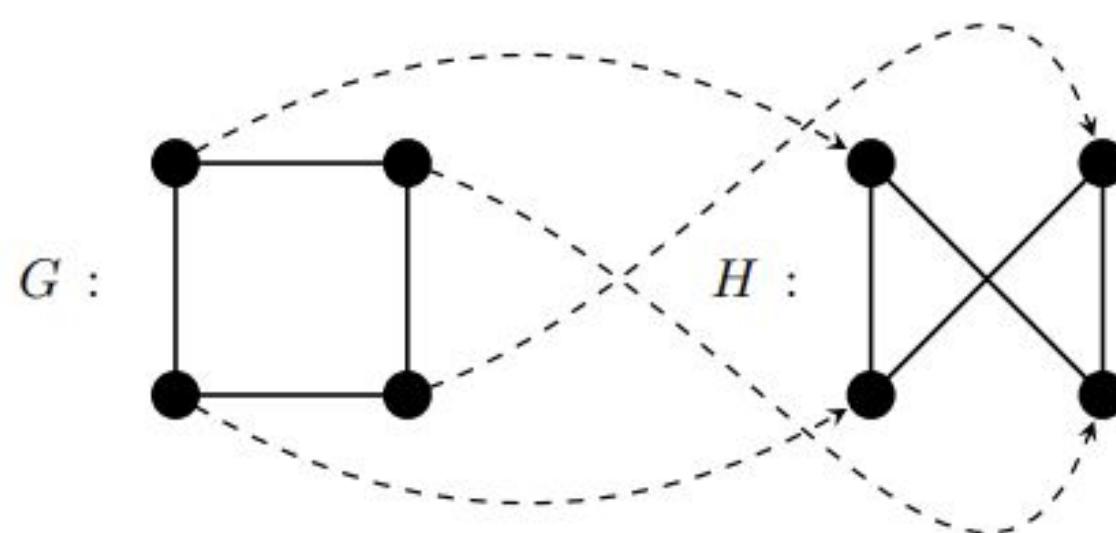
Recall Lecture # 01

Many Ways to Draw a Graph

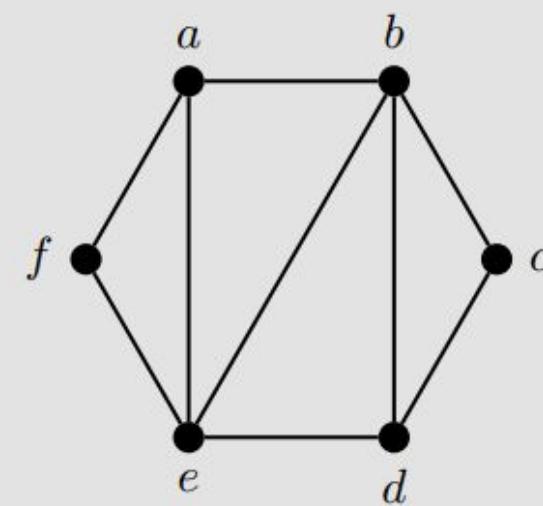
Are these graphs the same?



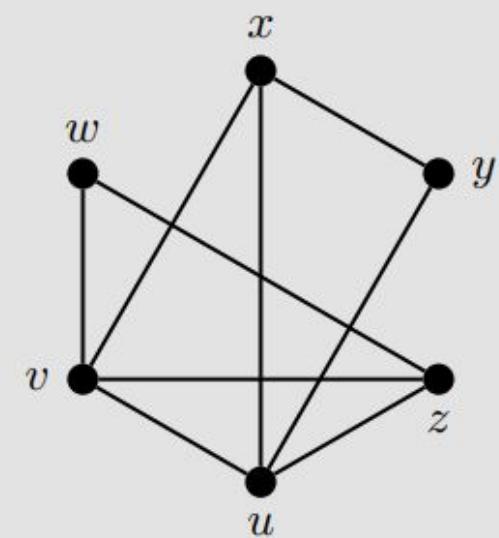
- number of vertices
- number of edges
- vertex degrees



Example 1.14 Determine if the following pair of graphs are isomorphic. If so, give the vertex pairings; if not, explain what property is different among the graphs.

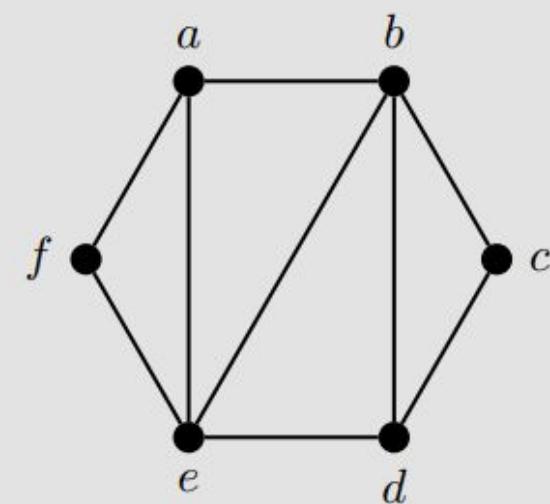


G_1

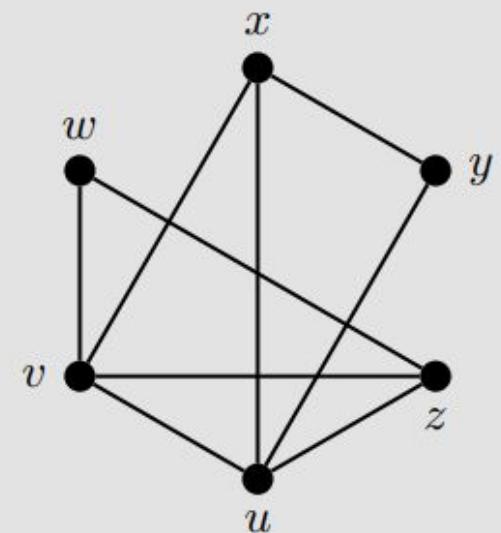


G_2

- number of vertices
- number of edges
- vertex degrees

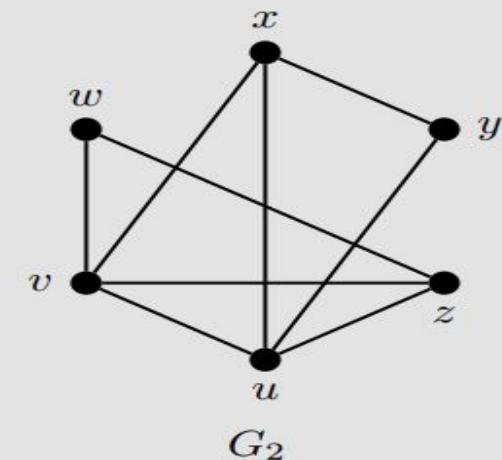
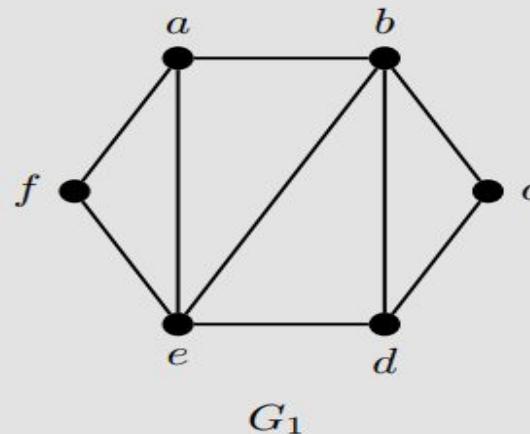


G_1



G_2

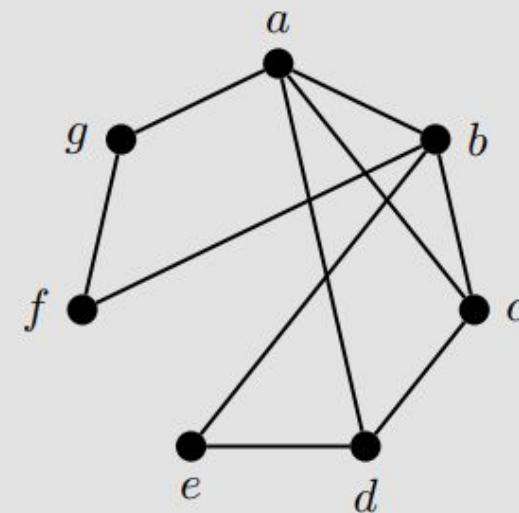
Solution: First note that both graphs have six vertices and nine edges, with two vertices each of degrees 4, 3, and 2. Since corresponding vertices must have the same degree, we know b must map to either u or v . We start by trying to map b to v . By looking at vertex adjacencies and degree, we must have e map to u , c map to w , and a map to x . This leaves f and d , which must be mapped to y and z , respectively. The chart below show the vertex pairings and checks for corresponding edges.



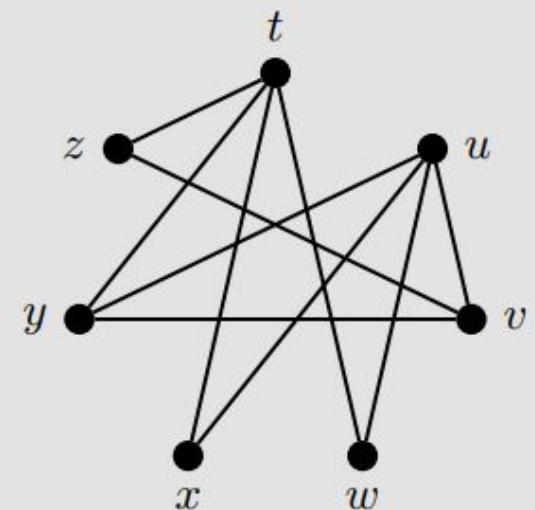
$V(G_1) \leftrightarrow V(G_2)$	Edges	
$a \leftrightarrow x$	$ab \leftrightarrow xv$	✓
$b \leftrightarrow v$	$ae \leftrightarrow xu$	✓
$c \leftrightarrow w$	$af \leftrightarrow xy$	✓
$d \leftrightarrow z$	$bc \leftrightarrow vw$	✓
$e \leftrightarrow u$	$bd \leftrightarrow vz$	✓
$f \leftrightarrow y$	$be \leftrightarrow vu$	✓
	$cd \leftrightarrow wz$	✓
	$de \leftrightarrow zu$	✓
	$ef \leftrightarrow uy$	✓

Since all edge relationships are maintained, we know G_1 and G_2 are isomorphic.

Example 1.15 Determine if the following pair of graphs are isomorphic. If so, give the vertex pairings; if not, explain what property is different among the graphs.

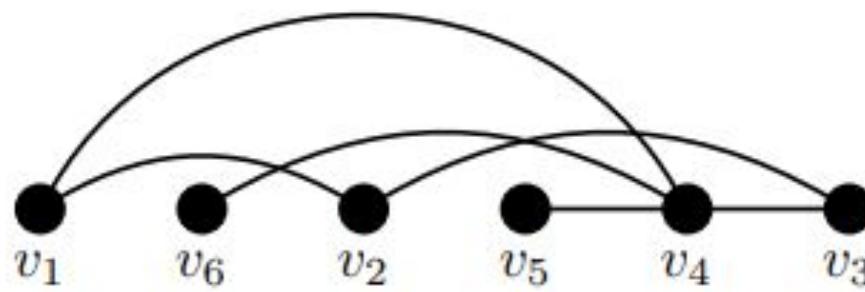


G_3

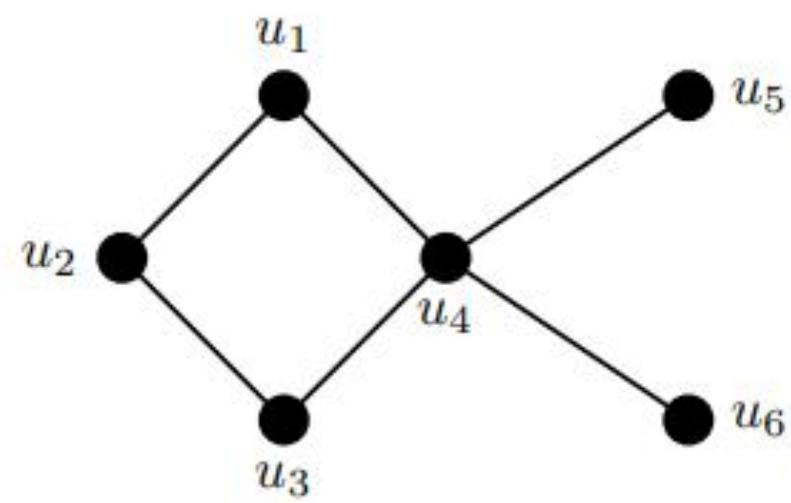


G_4

Example 1.15 Determine if the following pair of graphs are isomorphic. If so, give the vertex pairings; if not, explain what property is different among the graphs.



(a) G



(b) H

Theorem 1.18 Assume G_1 and G_2 are isomorphic graphs. Then G_1 and G_2 must satisfy any of the properties listed below; that is, if G_1

- is connected
- has n vertices
- has m edges
- has m vertices of degree k
- has a cycle of length k (see [Section 2.1.2](#))
- has an eulerian circuit (see [Section 2.1.3](#))
- has a hamiltonian cycle (see [Section 2.2](#))

then so too must G_2 (where n, m , and k are non-negative integers).

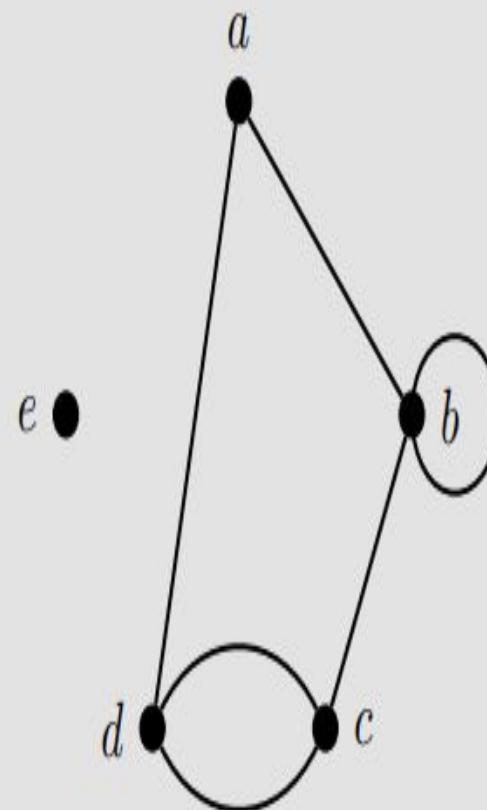
Matrix Representation

WHY Matrix Representation??

Matrix Representation

Definition 1.19 The *adjacency matrix* $A(G)$ of the graph G is the $n \times n$ matrix where vertex v_i is represented by row i and column i and the entry a_{ij} denotes the number of edges between v_i and v_j .

Example 1.16 Find the adjacency matrix for the graph G_4 from Example 1.1.

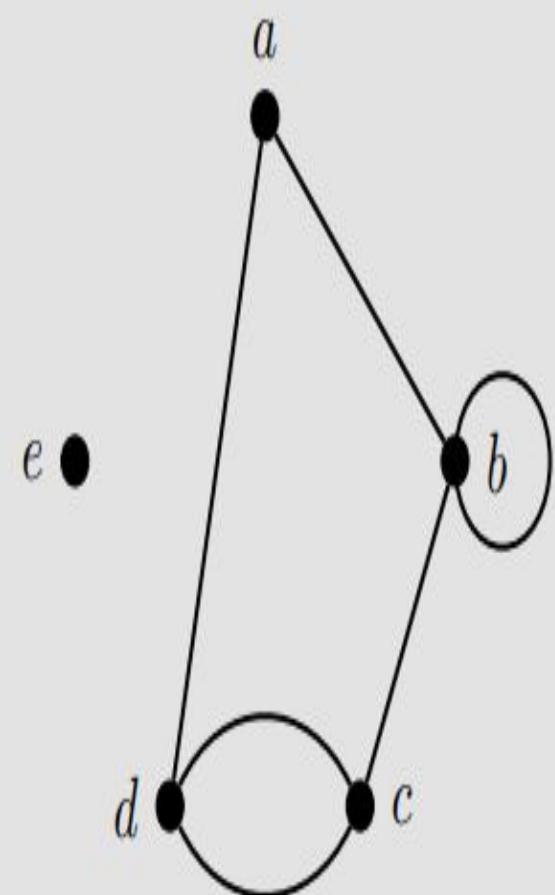


Example 1.16 Find the adjacency matrix for the graph G_4 from Example 1.1.

Solution:

$$\begin{array}{cc}
 & \begin{matrix} a & b & c & d & e \end{matrix} \\
 \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \left[\begin{matrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \right]
 \end{array}$$

Note that the entry $(2, 2)$ represents the loop at b and the entries $(3, 4)$ and $(4, 3)$ show that there are two edges between c and d . The column for e has all 0's since e is an isolated vertex.



NOTE:

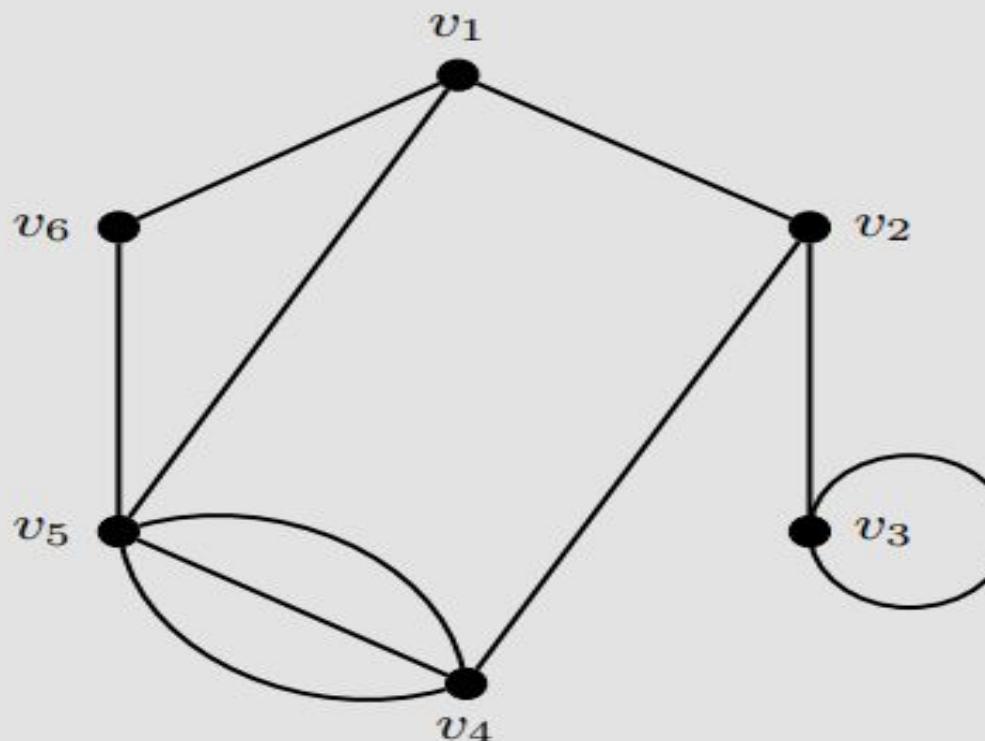
A few interesting properties of the adjacency matrix can be seen. First, the matrix is symmetric along the main diagonal since if there is an edge $v_i v_j$ then it will be accounted for in both the entry (i, j) and (j, i) in the matrix. Second, the main diagonal represents all loops in the graph. Finally, the degree of a vertex can be easily calculated from the adjacency matrix by adding the entries along the row (or column) representing the vertex but double any item along the diagonal. In the matrix above, we would get $\deg(a) = 2$ and $\deg(b) = 4$, which matches the graph representation from Example 1.1.

Example 1.17 Draw the graph whose adjacency matrix is shown below.

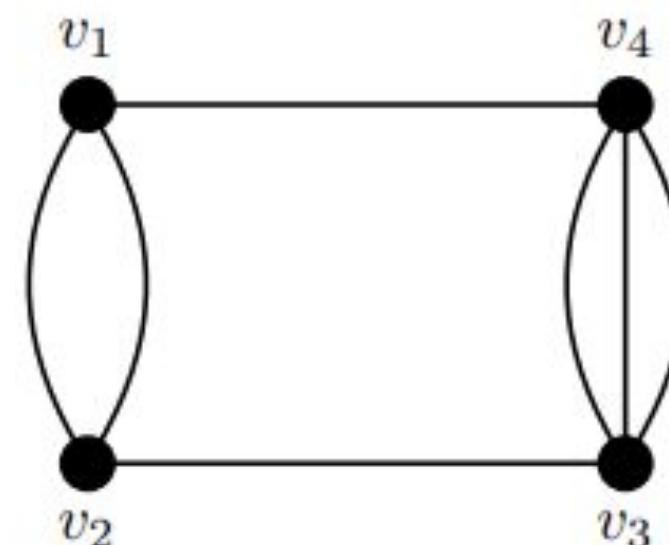
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 \\ 1 & 0 & 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Example 1.17 Draw the graph whose adjacency matrix is shown below.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 \\ 1 & 0 & 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Example 1.2.11. Let G be the multigraph shown in Fig. 1.2.9, where its four vertices are named v_1 , v_2 , v_3 and v_4 .



$$A = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 \\ 1 & 0 & 3 & 0 \end{pmatrix}.$$

Let G be an (n, m) -multigraph with

$$V(G) = \{v_1, v_2, \dots, v_n\} \text{ and } E(G) = \{e_1, e_2, \dots, e_m\}.$$

The **incidence matrix** of G is the $n \times m$ matrix

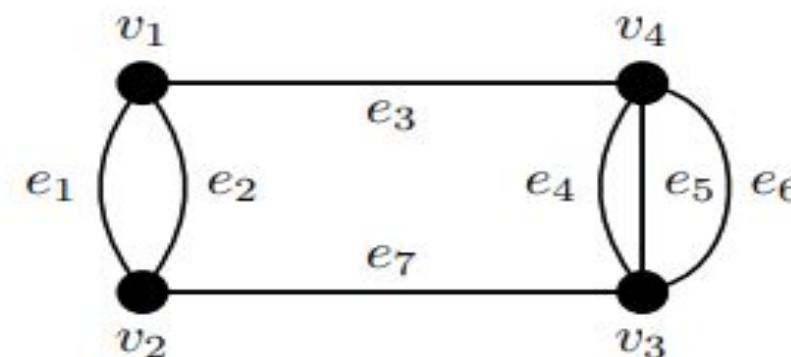
$$\mathbf{B}(G) = (b_{ij})_{n \times m},$$

where

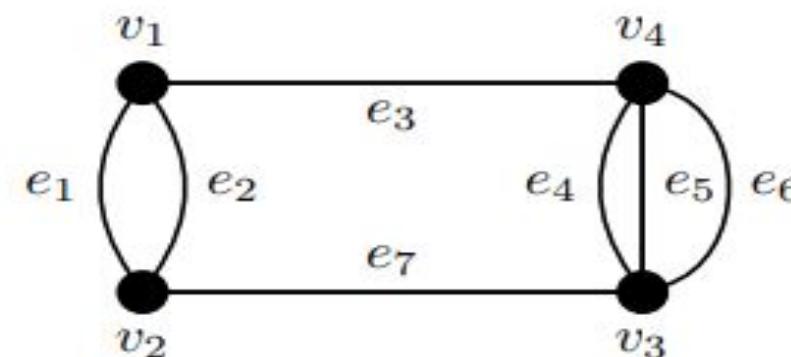
$$b_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is incident with } e_j, \\ 0 & \text{otherwise.} \end{cases}$$

2 For Loop

Example 1.2.14. Let G be the multigraph discussed in Example 1.2.11, where its seven edges are named e_1, e_2, \dots, e_7 as shown in Fig. 1.2.11:



Example 1.2.14. Let G be the multigraph discussed in Example 1.2.11, where its seven edges are named e_1, e_2, \dots, e_7 as shown in Fig. 1.2.11:



$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

PROOF TECHNIQUES

Handshaking



- Before a business meeting, some people shook hands. Then the number of people who made an odd number of handshakes is even
- In graph terms: A graph has an even number of odd nodes

PROOF TECHNIQUES

Proposition 1.20 The sum of two odd integers is even.

Proof: Assume x and y are odd integers. Then there exist integers n and m such that $x = 2n+1$ and $y = 2m+1$. Thus $x+y = (2n+1)+(2m+1) = 2(n+m+1) = 2k$, where k is the integer given by $n+m+1$. Therefore $x+y$ is even.

Degree Sum Formula

Lemma

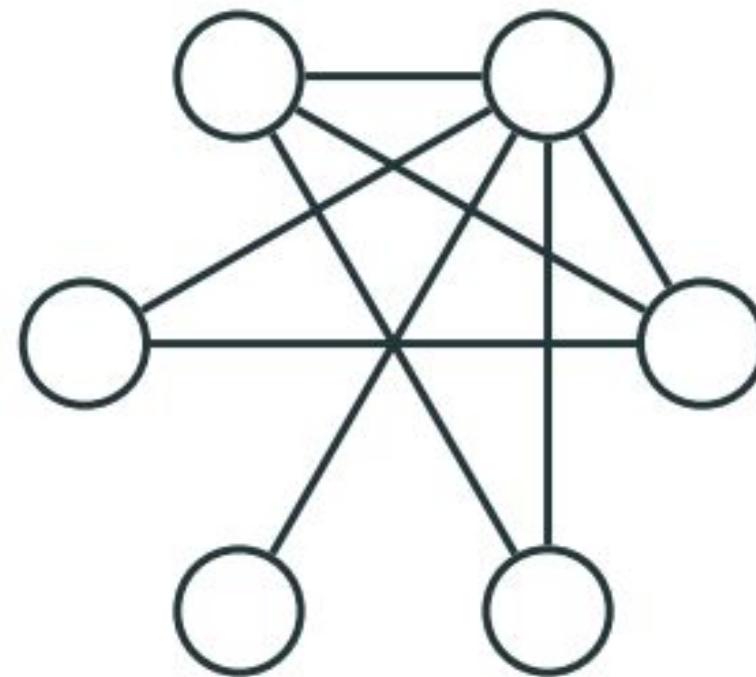
For any graph $G(V, E)$, the sum of degrees of all its nodes is twice the number of edges:

$$\sum_{v \in V} \text{degree}(v) = 2 \cdot |E|.$$

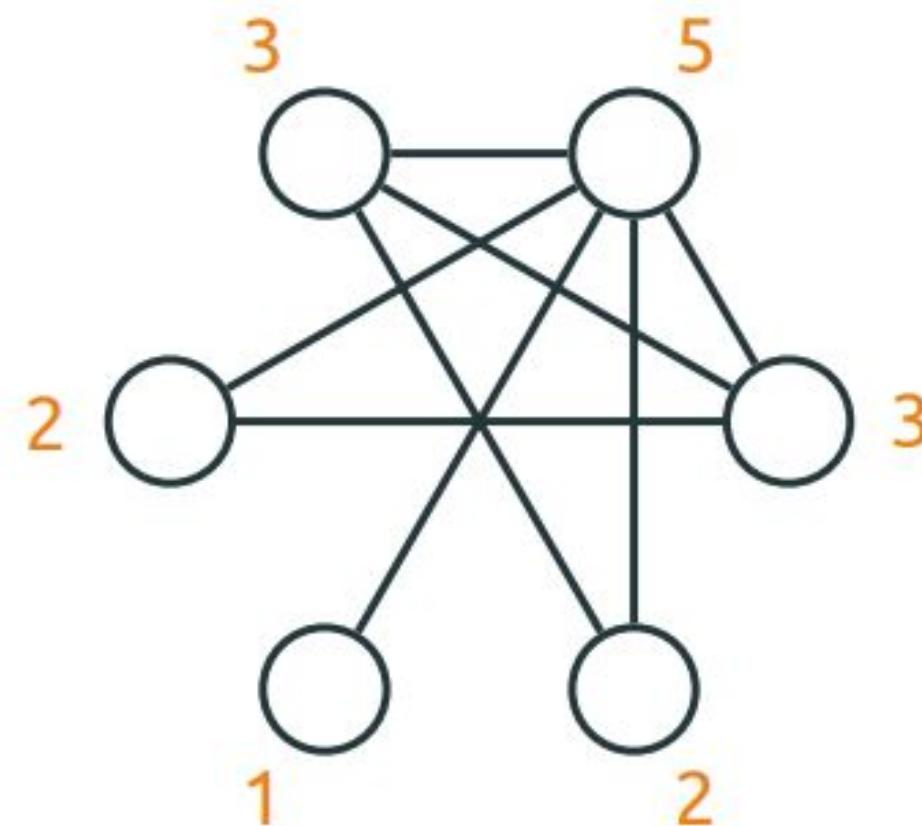
Theorem 1.21 (Handshaking Lemma) Let $G = (V, E)$ be a graph and $|E|$ denote the number of edges in G . Then the sum of the degrees of the vertices equals twice the number of edges; that is if $V = \{v_1, v_2, \dots, v_n\}$, then

$$\sum_{i=1}^n \deg(v_i) = \deg(v_1) + \deg(v_2) + \dots + \deg(v_n) = 2|E|.$$

Example

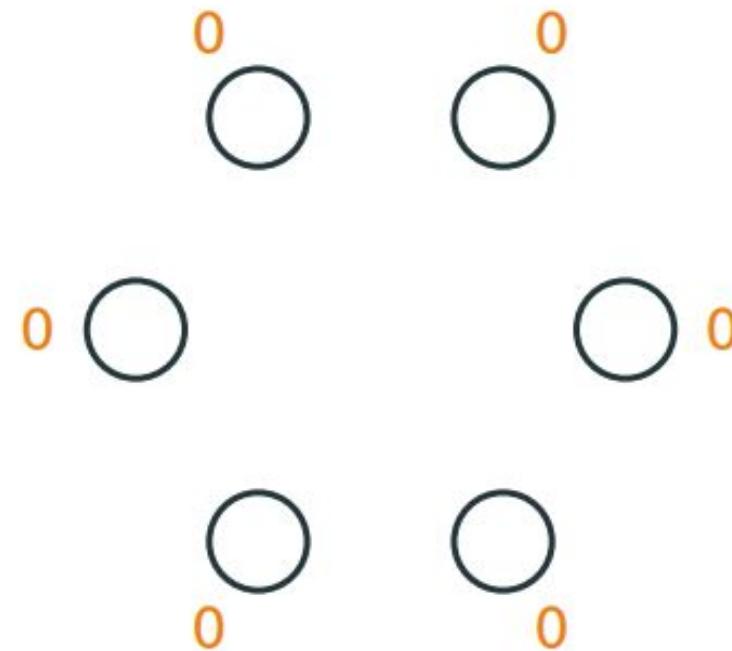


Example



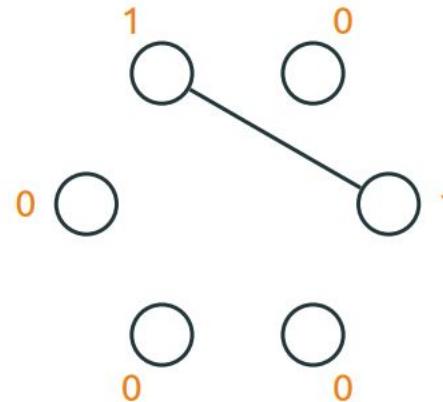
$$3 + 5 + 3 + 2 + 1 + 2 = 2 \cdot 8$$

Proof



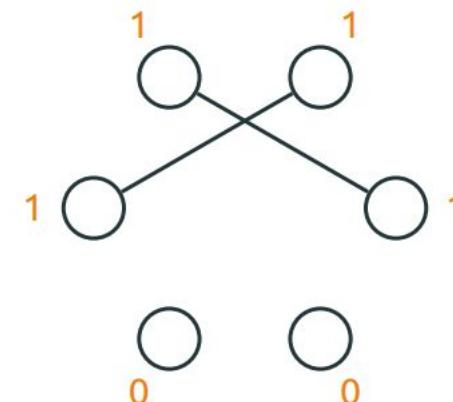
now, start adding them back, one by one

Proof



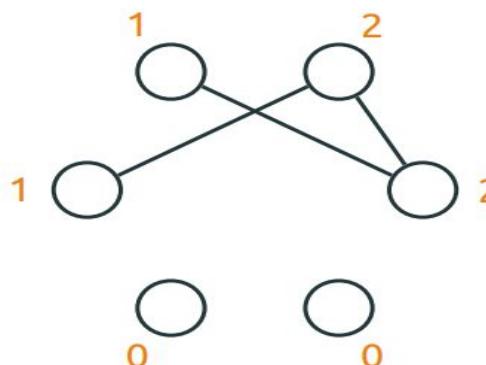
now, start adding them back, one by one

Proof



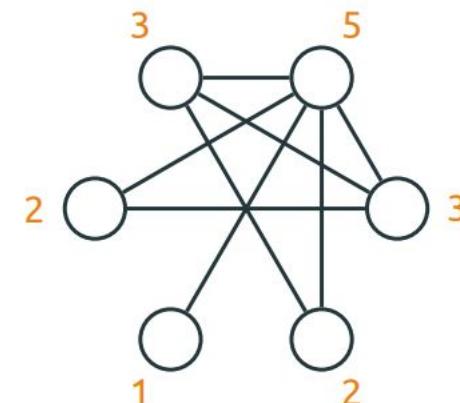
now, start adding them back, one by one

Proof



now, start adding them back, one by one

Proof



each edge contributes 2 to the sum of degrees

Summary

- Essentially, we proved the formula by induction on the number of edges
- Base case: the formula holds if there are no edges (all degrees are equal to 0)
- Induction step: when we add an edge, the sum of degrees increases by 2

Theorem 1.4 (The First Theorem of Graph Theory) *If G is a graph of size m , then*

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

Proof. When summing the degrees of the vertices of G , each edge of G is counted twice, once for each of its two incident vertices. ■

The sum of the degrees of the vertices of the graph G of Figure 1.4 is 12, which is twice the size 6 of G , as expected from Theorem 1.4. The **average degree** of a graph G of order n and size m is

$$\frac{\sum_{v \in V(G)} \deg v}{n} = \frac{2m}{n}.$$

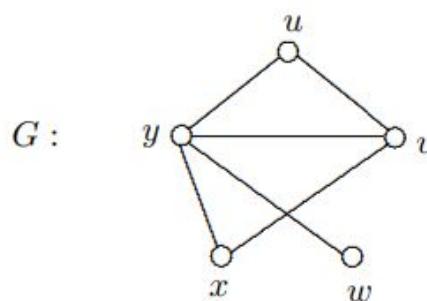


Figure 1.4: A graph

Even and Odd Vertices

A vertex in a graph G is **even** or **odd**, according to whether its degree in G is even or odd. Thus, the graph G of Figure 1.4 has three even vertices and two odd vertices. While a graph can have either an even or an odd number of even vertices, this is not the case for odd vertices.

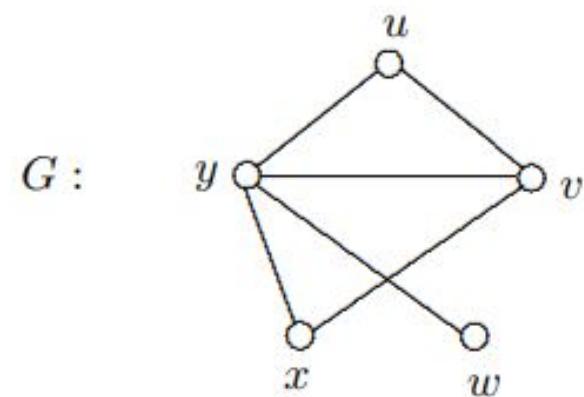


Figure 1.4: A graph

Try to understand it!

Corollary 1.22 Every graph has an even number of vertices of odd degree.

Prove Theorem 1.2: Every graph has an even number of odd vertices.

Solution. Suppose the sum of the degrees of the odd vertices is x and the sum of the degrees of the even vertices is y . The number y is even, and the number $x + y$, being twice the number of edges, is also even. So x is necessarily even. If there are p odd vertices, the even number x is the sum of p odd numbers. So p is even.



Proposition 1.23 For any integer n , if n^2 is odd then n is odd.

Proof: Suppose for a contradiction that n^2 is odd but n is even. Then $n = 2k$ for some integer k and $n^2 = (2k)^2 = 4k^2 = 2j$ where j is the integer $2k^2$. Thus n^2 is both even and odd, a contradiction. Therefore if n^2 is odd then n is also odd.

Proof: Suppose n is not odd. Then n is even and $n = 2k$ for some integer k . Then $n^2 = (2k)^2 = 4k^2 = 2j$ where j is the integer $2k^2$, and so n^2 is even. Thus if n^2 is odd, it must be that n is also odd.

Proposition 1.24 For every simple graph G on at least 2 vertices, there exist two vertices of the same degree.

Proof: Suppose for a contradiction that G is a simple graph on n vertices, with $n \geq 2$, in which no two vertices have the same degree. Since there are no loops and each vertex can have at most one edge to any other vertex, we know the maximum degree for any vertex is $n - 1$ and the minimum degree is 0. Since there are exactly n integers from 0 to $n - 1$, we know there must be exactly one vertex for each degree between 0 and $n - 1$. But the vertex of degree $n - 1$ must then be adjacent to every other vertex of G , which contradicts the fact that a vertex has degree 0. Thus G must have at least two vertices of the same degree.

Mathematical Induction

Proposition 1.25 The complete graph K_n has $\frac{n(n - 1)}{2}$ edges.

Proof: Argue by induction on n . If $n = 1$ then K_1 is just a single vertex and has $0 = \frac{1(0)}{2}$ edges.

Suppose for some $n \geq 1$ that K_n has $\frac{n(n - 1)}{2}$ edges. We can form K_{n+1} by adding a new vertex v to K_n and adjoining v to all the vertices from K_n . Thus K_{n+1} has n more edges than K_n and so by the induction hypothesis has

$$n + \frac{n(n - 1)}{2} = \frac{2n + n(n - 1)}{2} = \frac{n(2 + n - 1)}{2} = \frac{n(n + 1)}{2} = \frac{n(n-1)+2n}{2}$$

edges.

Thus by induction we know K_n has $\frac{n(n - 1)}{2}$ edges for all $n \geq 1$.

The power of induction is that we are proving a statement that holds for an infinite number of objects but only need to prove two very specific items.

Definition 1.26 The *degree sequence* of a graph is a listing of the degrees of the vertices. It is customary to write these in decreasing order. If a sequence is a degree sequence of a simple graph then we call it *graphical*.

Remark 2.5.1. All graphs considered for graphic sequences need not be connected, but are always assumed to be simple.

Any graph G of order n with $V(G) = \{v_1, v_2, \dots, v_n\}$ has its **degree sequence** defined to be $(d(v_1), d(v_2), \dots, d(v_n))$. We usually assume that the sequence is non-increasing, that is, $d(v_1) \geq d(v_2) \geq \dots \geq d(v_n)$.

A non-increasing sequence $\mathbf{d} = (d_1, d_2, \dots, d_n)$ of non-negative integers is called a **graphic sequence** if \mathbf{d} is the degree sequence of some graph. If \mathbf{d} is the degree sequence of a graph G , we say that \mathbf{d} is **representable** by G or G **represents** \mathbf{d} .



Remark 2.5.1. All graphs considered for graphic sequences need not be connected, but are always assumed to be simple.

Example 1.18 Explain why neither $4, 4, 2, 1, 0$ nor $4, 4, 3, 1, 0$ can be graphical.

Example 1.18 Explain why neither $4, 4, 2, 1, 0$ nor $4, 4, 3, 1, 0$ can be graphical.

Solution: The first sequence sums to 11, but we know the sum of the degrees of a graph must be even by the **Handshaking Lemma**. Thus it cannot be a **degree sequence**.

The second sequence sums to 12, so it is at least even. However, in a simple graph with 5 vertices, a vertex with **degree 4 must be adjacent to all the other vertices**, which would mean **no vertex could have degree 0**. Thus the second sequence cannot be a degree sequence.

Which of the following sequences are graphic?

- (i) (0, 0, 0, 0)
- (iii) (1, 1, 0, 0)
- (v) (1, 1, 1, 1)
- (vii) (2, 2, 0, 0)
- (ix) (2, 1, 1, 1, 1)
- (xi) (4, 3, 2, 1, 0)

- (ii) (1, 0, 0, 0)
- (iv) (1, 1, 1, 0)
- (vi) (2, 1, 1, 0)
- (viii) (2, 1, 1, 1, 0)
- (x) (3, 2, 2, 1, 1)
- (xii) (4, 3, 2, 2, 1)

Which of the following sequences are graphic?

- | | |
|----------------------|------------------------|
| (i) (0, 0, 0, 0) | (ii) (1, 0, 0, 0) |
| (iii) (1, 1, 0, 0) | (iv) (1, 1, 1, 0) |
| (v) (1, 1, 1, 1) | (vi) (2, 1, 1, 0) |
| (vii) (2, 2, 0, 0) | (viii) (2, 1, 1, 1, 0) |
| (ix) (2, 1, 1, 1, 1) | (x) (3, 2, 2, 1, 1) |
| (xi) (4, 3, 2, 1, 0) | (xii) (4, 3, 2, 2, 1) |

It is clear that the sequences (ii), (iv), (viii) and (x) are non-graphic.
The reason is that the number of odd vertices in any graph must be even.

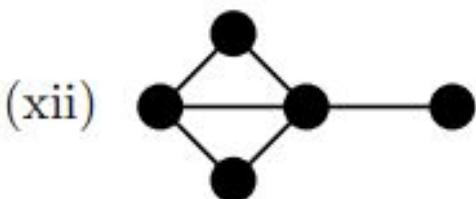
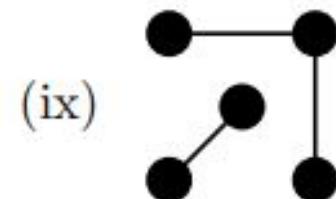
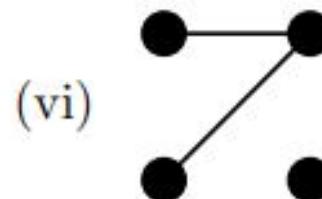
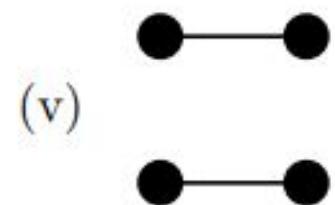
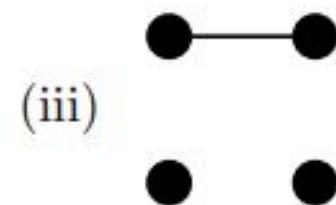
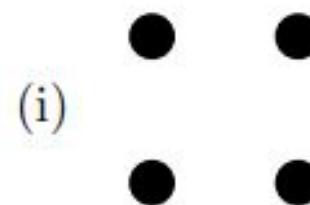
Which of the following sequences are graphic?

- | | |
|----------------------|------------------------|
| (i) (0, 0, 0, 0) | (ii) (1, 0, 0, 0) |
| (iii) (1, 1, 0, 0) | (iv) (1, 1, 1, 0) |
| (v) (1, 1, 1, 1) | (vi) (2, 1, 1, 0) |
| (vii) (2, 2, 0, 0) | (viii) (2, 1, 1, 1, 0) |
| (ix) (2, 1, 1, 1, 1) | (x) (3, 2, 2, 1, 1) |
| (xi) (4, 3, 2, 1, 0) | (xii) (4, 3, 2, 2, 1) |

The sequence (vii) is not graphic. For if it were representable by, say G , then G would have two isolated vertices, and thus have its remaining two vertices joined by ‘2’ parallel edges, which is not allowed for ‘simple’ graphs.

- | | |
|----------------------|------------------------|
| (i) (0, 0, 0, 0) | (ii) (1, 0, 0, 0) |
| (iii) (1, 1, 0, 0) | (iv) (1, 1, 1, 0) |
| (v) (1, 1, 1, 1) | (vi) (2, 1, 1, 0) |
| (vii) (2, 2, 0, 0) | (viii) (2, 1, 1, 1, 0) |
| (ix) (2, 1, 1, 1, 1) | (x) (3, 2, 2, 1, 1) |
| (xi) (4, 3, 2, 1, 0) | (xii) (4, 3, 2, 2, 1) |

The remaining sequences are graphic, and the graphs representing them



Theorem 1.27 (Havel-Hakimi Theorem) An increasing sequence $S : s_1, s_2, \dots, s_n$ (for $n \geq 2$) of nonnegative integers is graphical if and only if the sequence

$$S' : s_2 - 1, s_3 - 1, \dots, s_{s_1} - 1, s_{s_{n+1}}, \dots, s_n$$

is graphical.

LECTURE # 07, 08 & 09

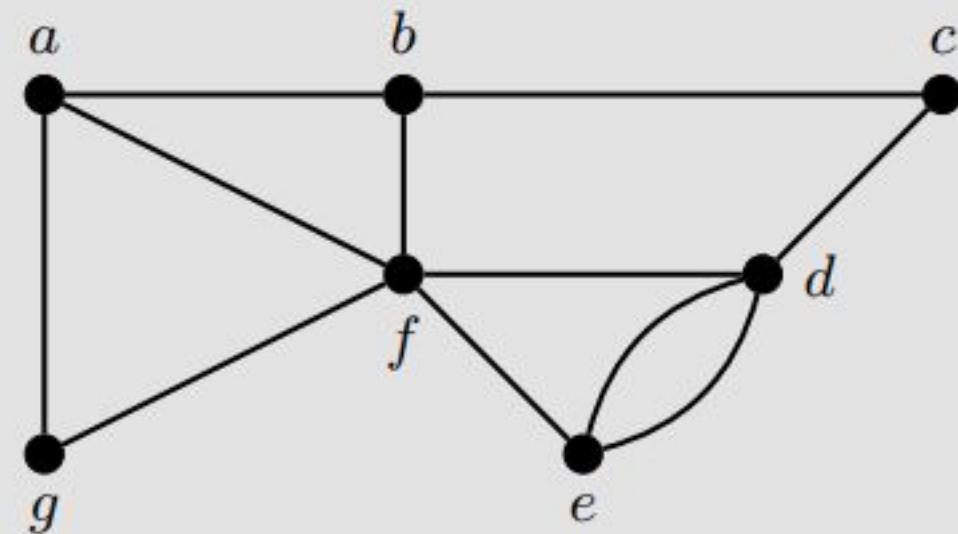
Definition 2.1 Let G be a graph.

- A ***walk*** is a sequence of vertices so that there is an edge between consecutive vertices. A walk can repeat vertices and edges.

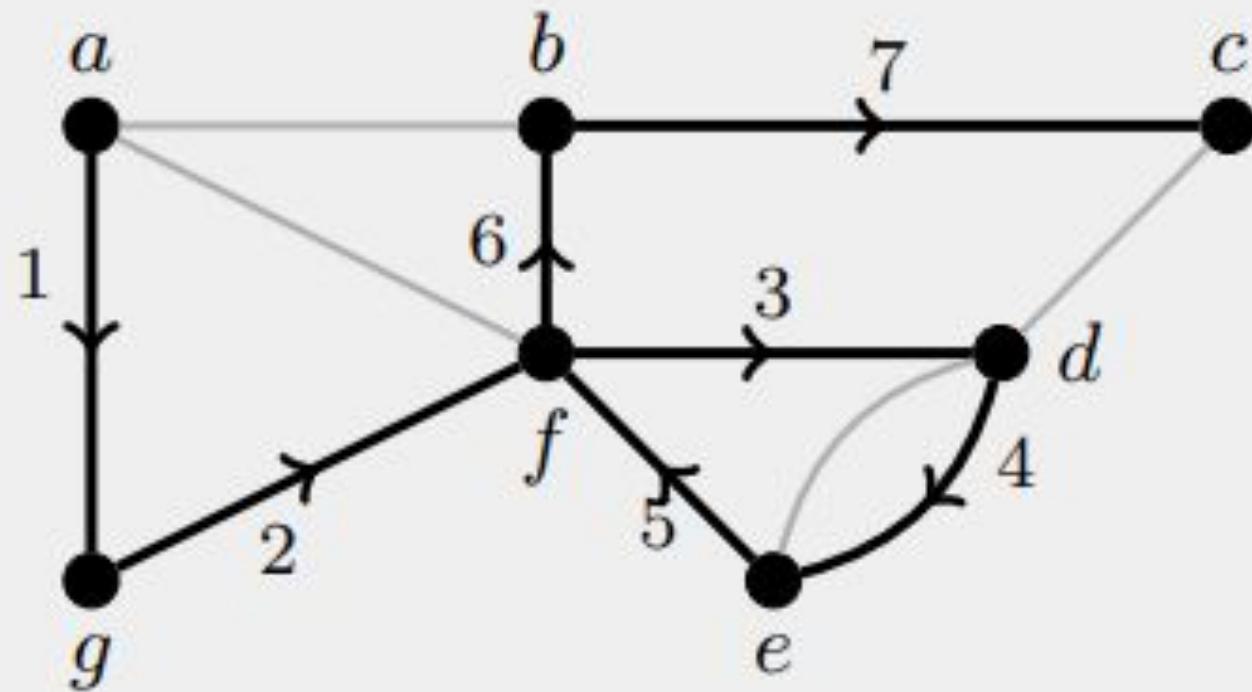
- A ***trail*** is a walk with no repeated edges. A trail can repeat vertices but not edges.
- A ***path*** is a trail with no repeated vertex (or edges). A path on n vertices is denoted P_n .
- A ***closed walk*** is a walk that starts and ends at the same vertex.
- A ***circuit*** is a closed trail; that is, a trail that starts and ends at the same vertex with no repeated edges though vertices may be repeated.
- A ***cycle*** is a closed path; that is, a path that starts and ends at the same vertex. Thus cycles cannot repeat edges or vertices. Note: we do not consider the starting and ending vertex as being repeated since each vertex is entered and exited exactly once. A cycle on n vertices is denoted C_n .

The ***length*** of any of these tours is defined in terms of the number of edges. For example, P_n has length $n - 1$ and C_n has length n .

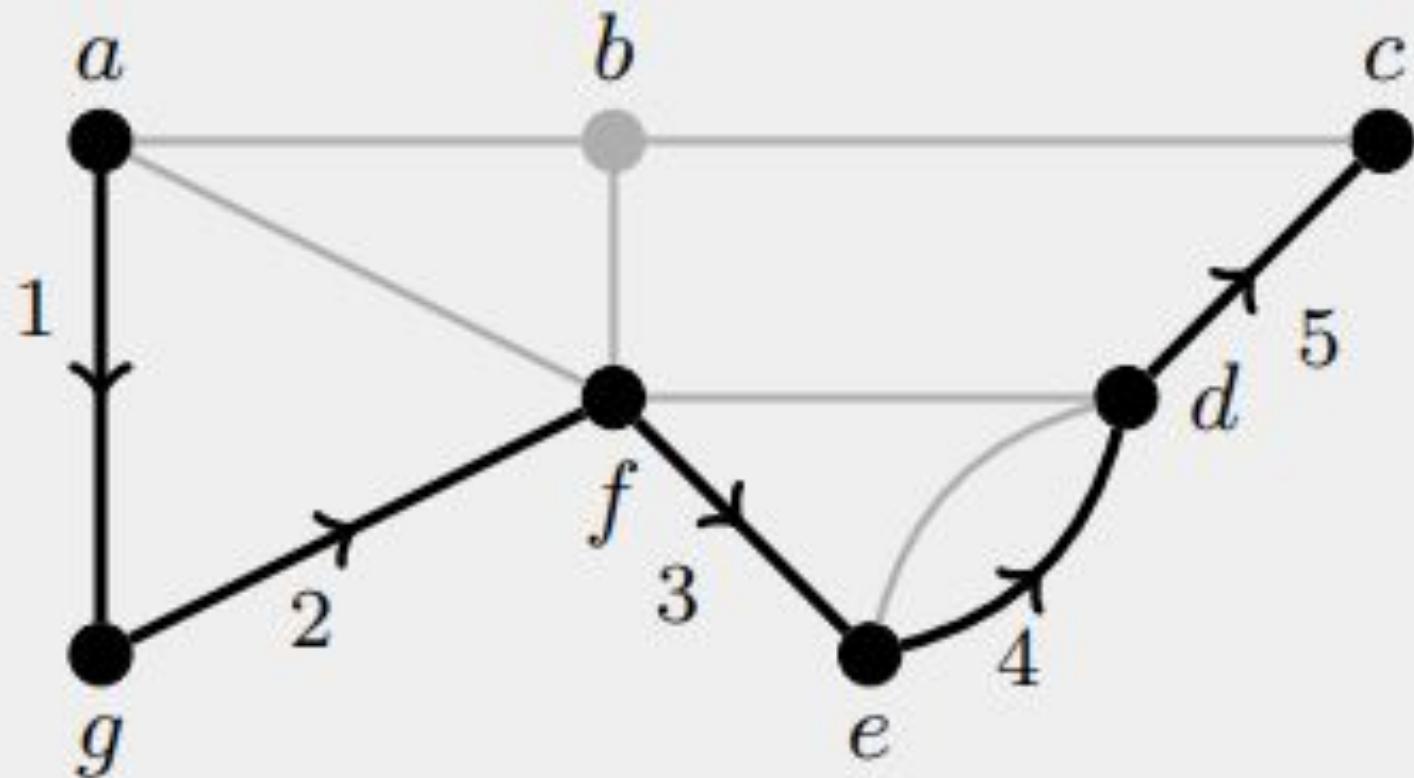
Example 2.1 Given the graph below, find a trail (that is not a path) from a to c , a path from a to c , a circuit (that is not a cycle) starting at b , and a cycle starting at b .



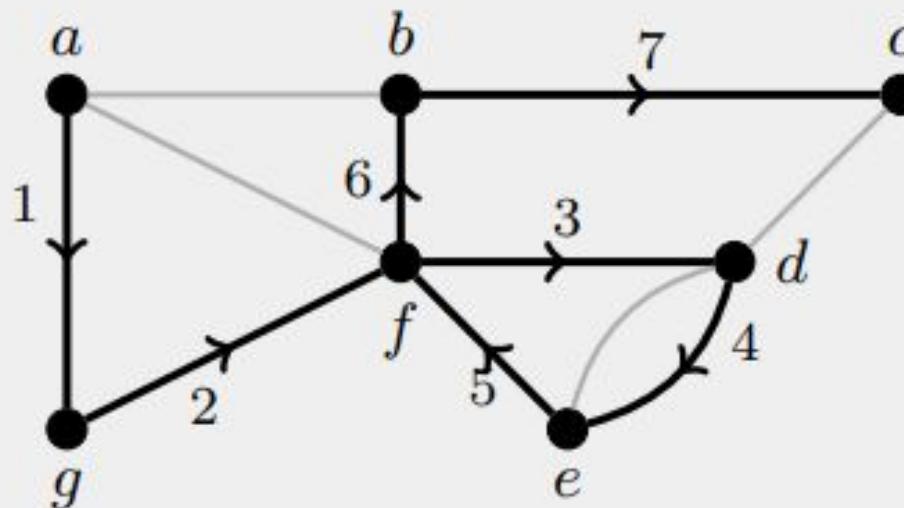
Trail from a to c



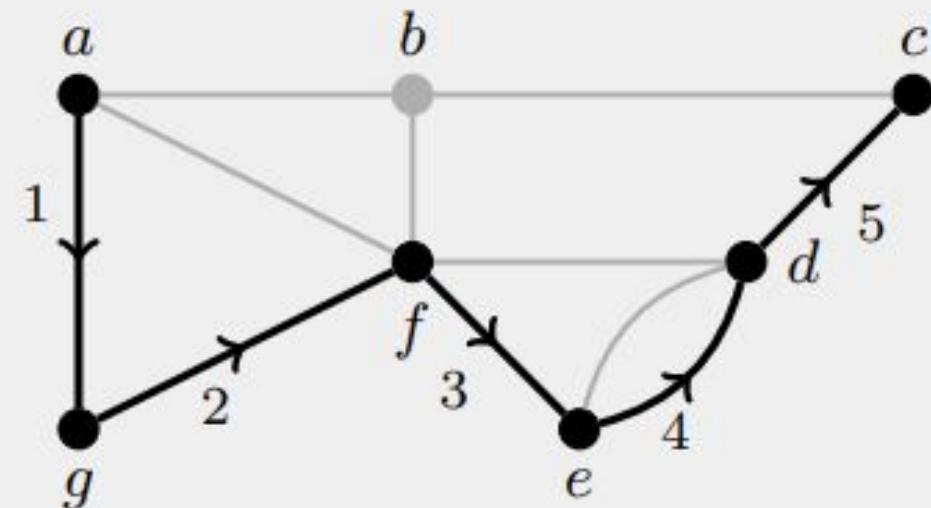
Path from a to c



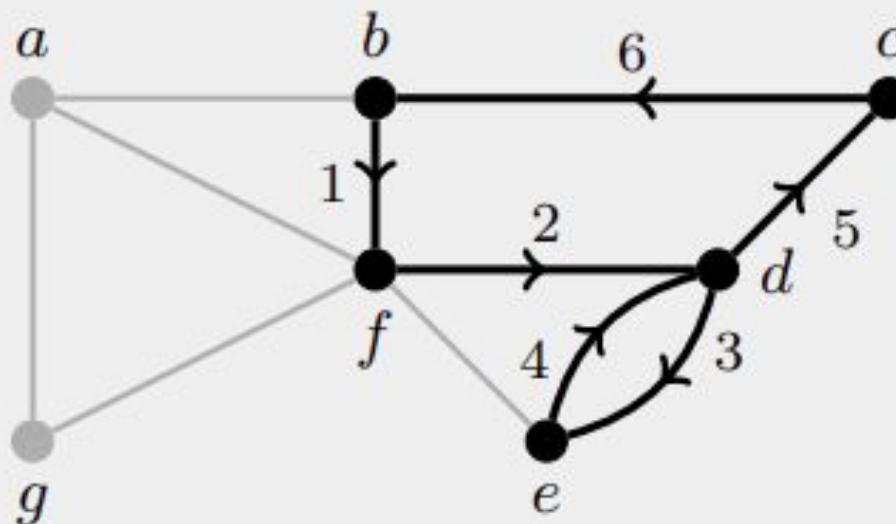
Trail from a to c



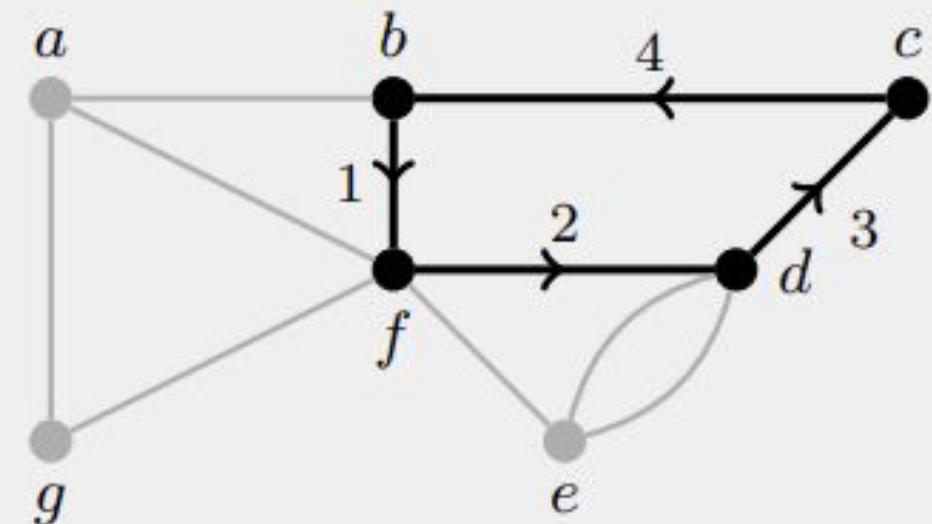
Path from a to c



Circuit starting at b



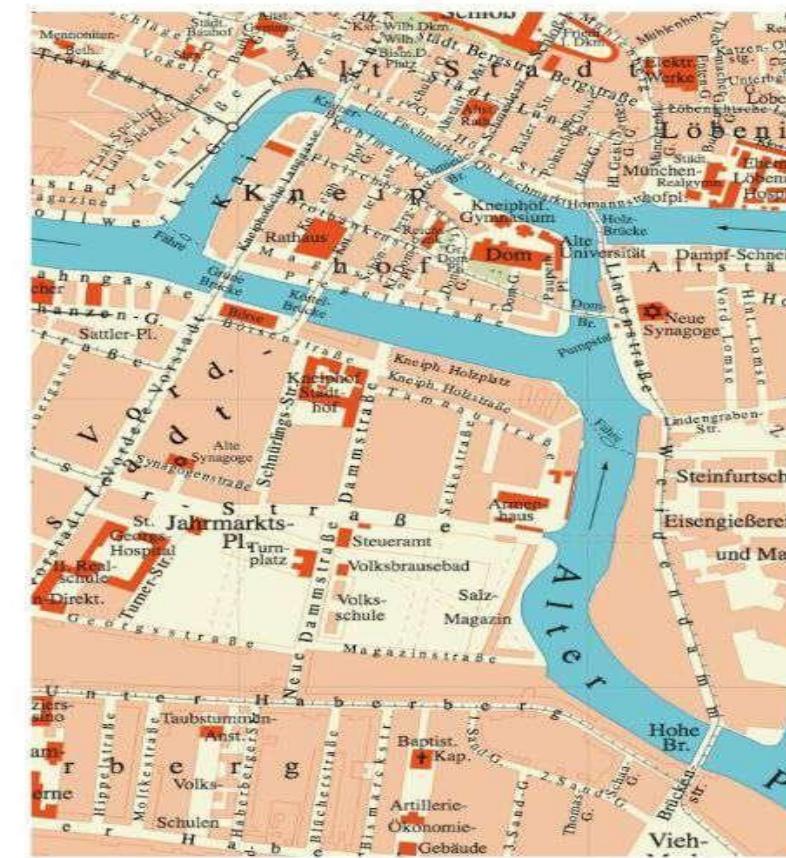
Cycle starting at b



Definition 2.2 Let G be a graph. Two vertices x and y are ***connected*** if there exists a path from x to y in G . The graph G is ***connected*** if every pair of distinct vertices is connected.

Seven Bridges of Königsberg

Königsberg, Prussia, 1735



Seven Bridges of Königsberg

Königsberg, Prussia, 1735

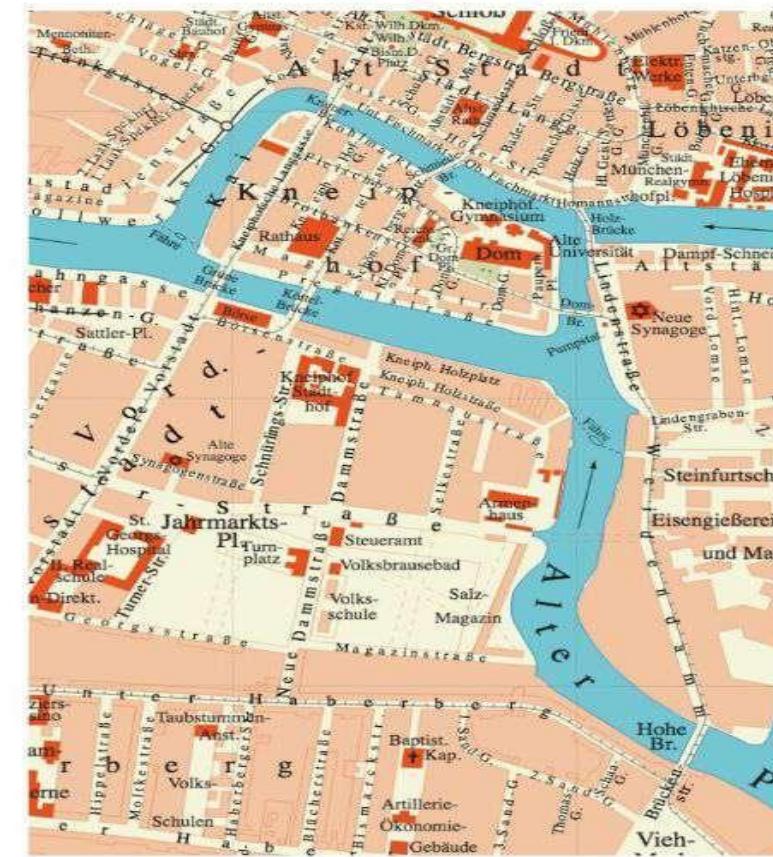
Walk through Königsberg

Cross each bridge
exactly once

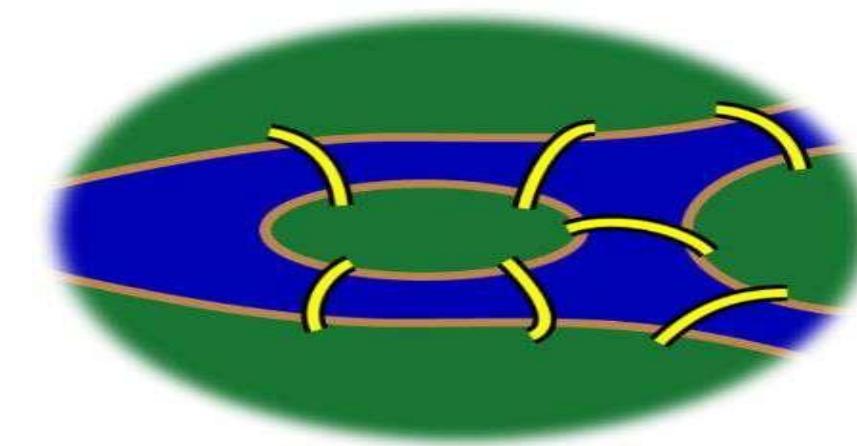
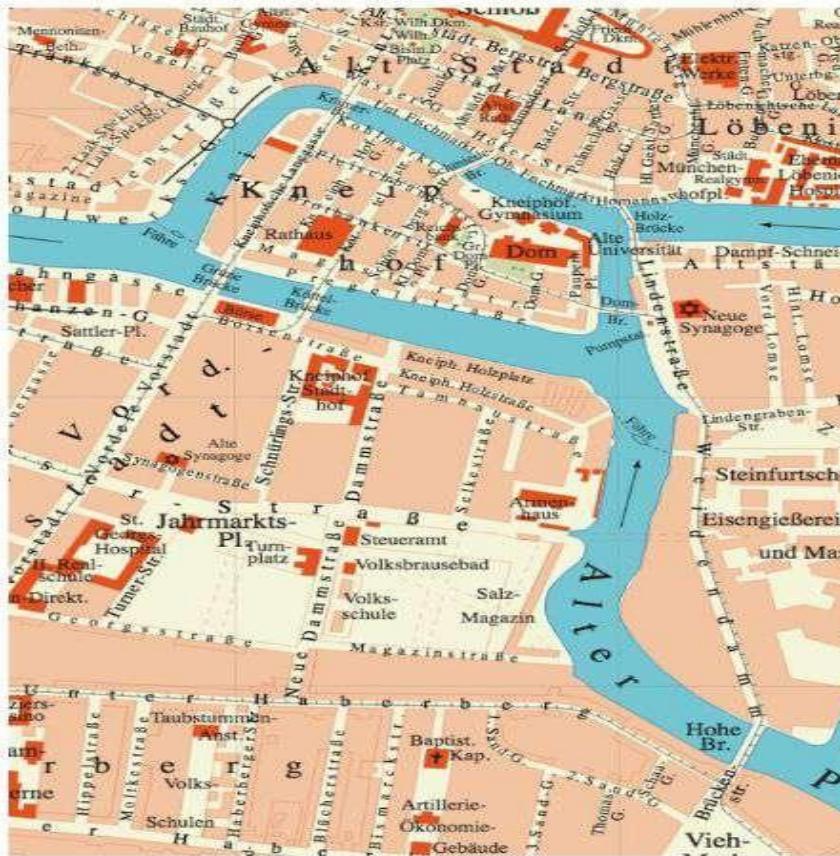


Impossible!

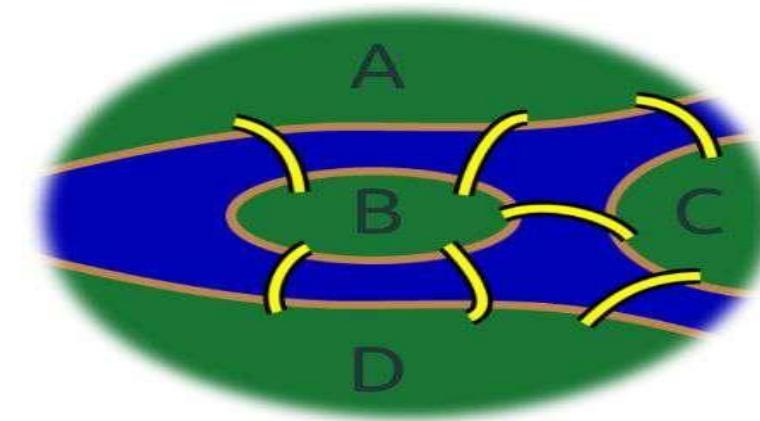
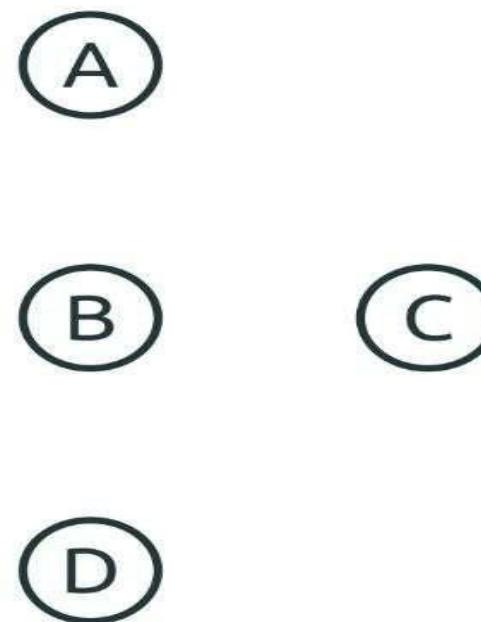
Leonhard Euler



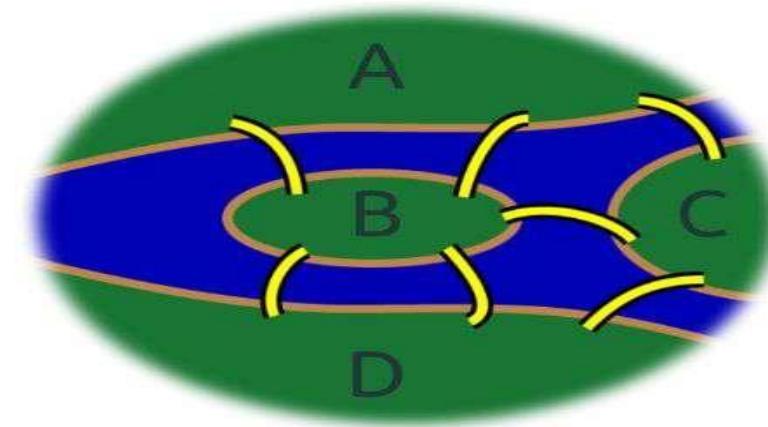
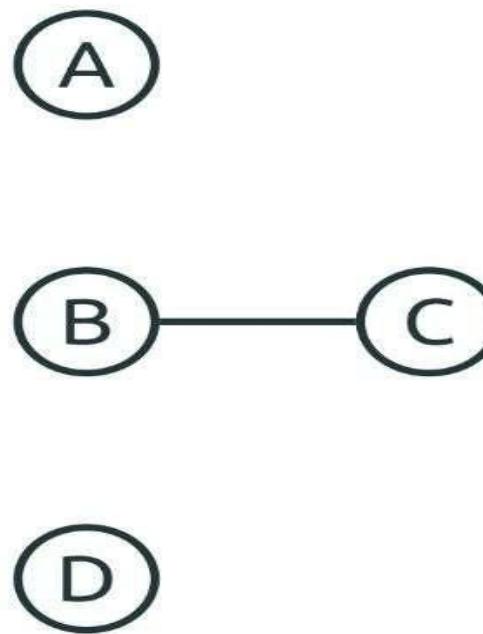
Bridges of Königsberg. Graph



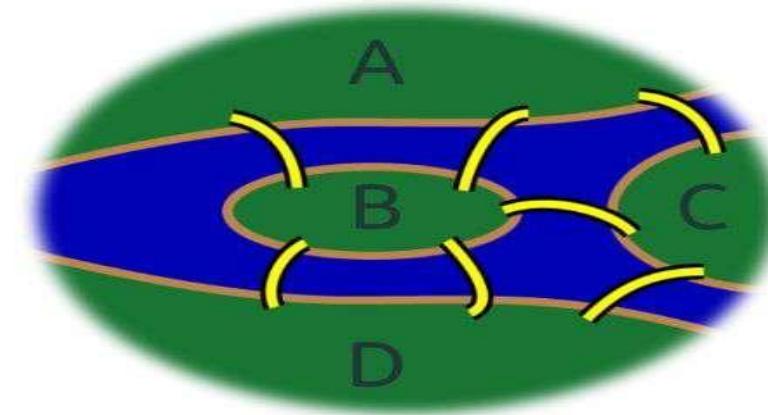
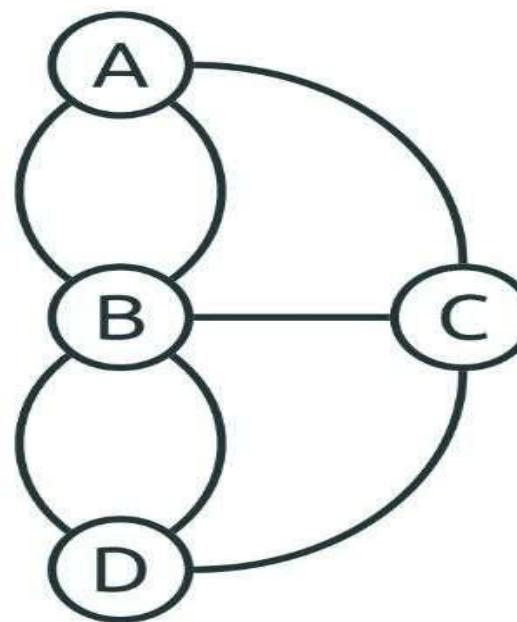
Bridges of Königsberg. Graph



Bridges of Königsberg. Graph



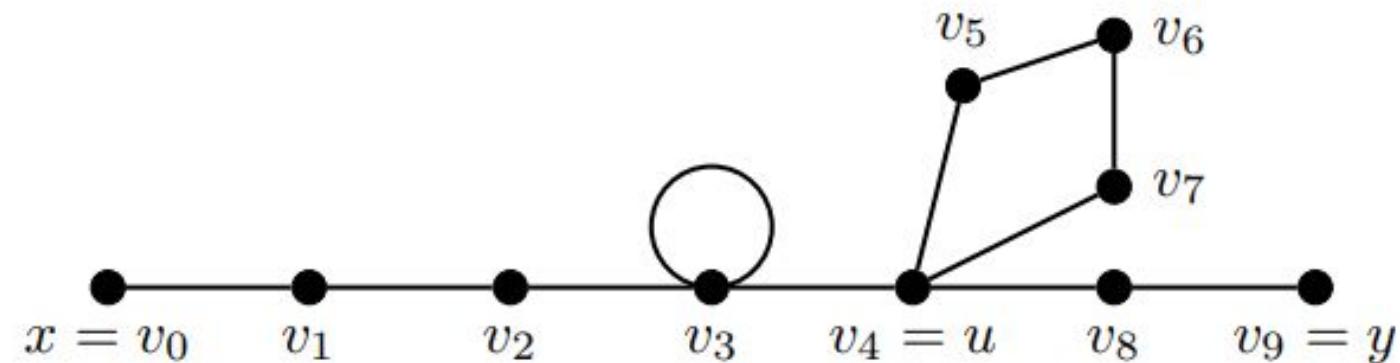
Bridges of Königsberg. Graph



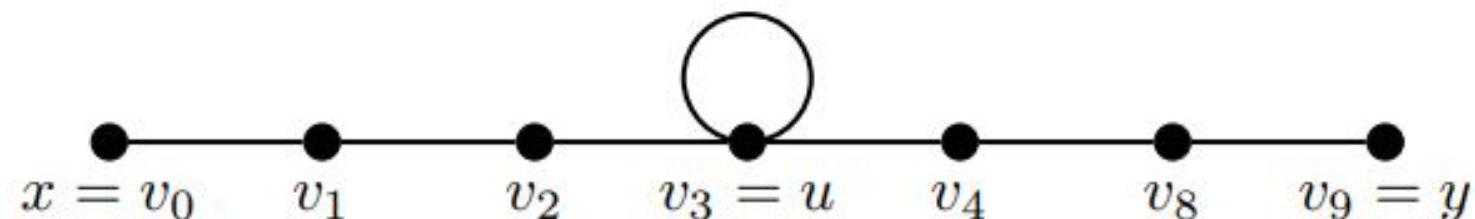
Theorem 2.3 Every $x - y$ walk contains an $x - y$ path.

Consider a walk as shown below. Using the technique from the proof above, we begin by identifying a repeated vertex, namely v_4 .

$$W : v_0 \ v_1 \ v_2 \ v_3 \ v_3 \boxed{v_4 \ v_5 \ v_6 \ v_7 \ v_4} \ v_8 \ v_9$$

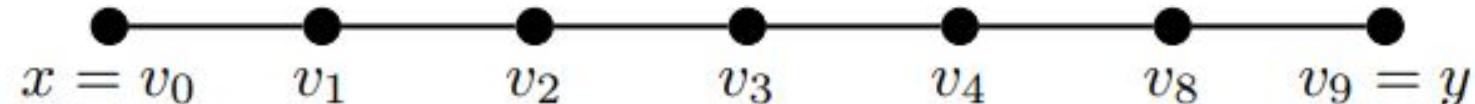


$$W' : v_0 \ v_1 \ v_2 \boxed{v_3 \ v_3} \ v_4 \ v_8 \ v_9$$



The final graph below does not contain any repeated vertices and so can be considered a path from v_0 to v_9 , all of whose vertices and edges were contained in the original walk W .

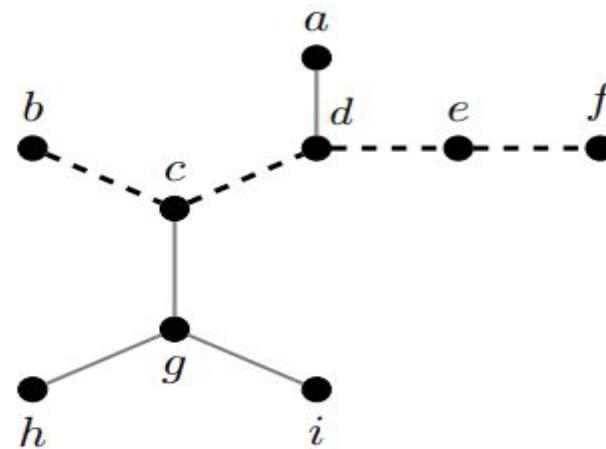
$$W'' : v_0 \ v_1 \ v_2 \ v_3 \ v_4 \ v_8 \ v_9$$



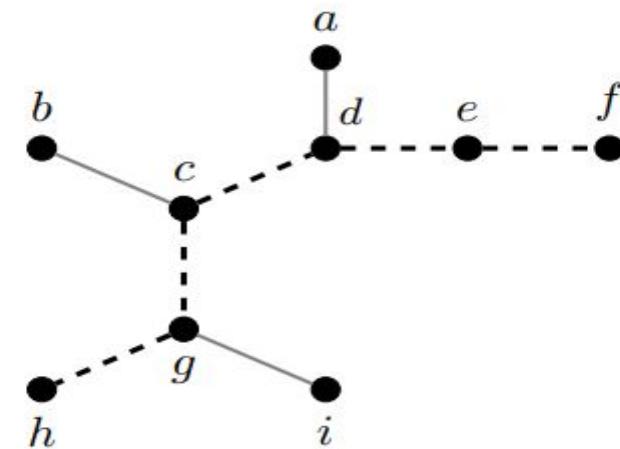
Definition 2.4 An object X is **maximum** if it is the largest among all objects under consideration; that is, $|X| \geq |A|$ for all $A \in \mathcal{U}$.

An object X is **maximal** if it cannot be made larger.

In certain scenarios, the adjectives maximum and maximal may be applied to the same object; however, this need not be true. Consider the graphs shown below. The path $b - c - d - e - f$ (highlighted on the left) is maximal because we cannot add any additional vertices and keep it a path. However, it is not maximum since a longer path can be found, namely $h - g - c - d - e - f$ shown on the right. Note that this second path is a maximum path within the graph shown.



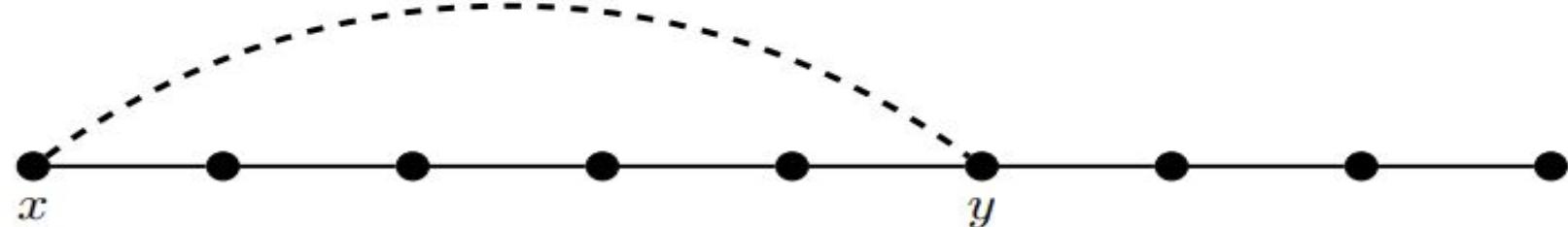
Maximal $b - f$ path



Maximum path in G

Theorem 2.5 If every vertex of a graph has degree at least 2 then G contains a cycle.

Proof: Let P be a maximal path in G and let x be an endpoint of P . Since P is maximal, it cannot be extended and so every neighbor of x must already be a vertex of P . Since x has degree at least 2, we know there must be another neighbor y of x in $V(P)$ via an edge not a part of the path.



Then the edge xy completes a cycle with the portion of P from x to y .

Definition 2.6 Let G be a graph. An **eulerian circuit** (or **trail**) is a circuit (or trail) that contains every edge and every vertex of G .

If G contains an eulerian circuit it is called **eulerian** and if G contains an eulerian trail but not an eulerian circuit it is called **semi-eulerian**.

Theorem 2.7 A graph G is eulerian if and only if

- (i) G is connected and
- (ii) every vertex has even degree.

Corollary 2.8 A graph G is semi-eulerian if and only if

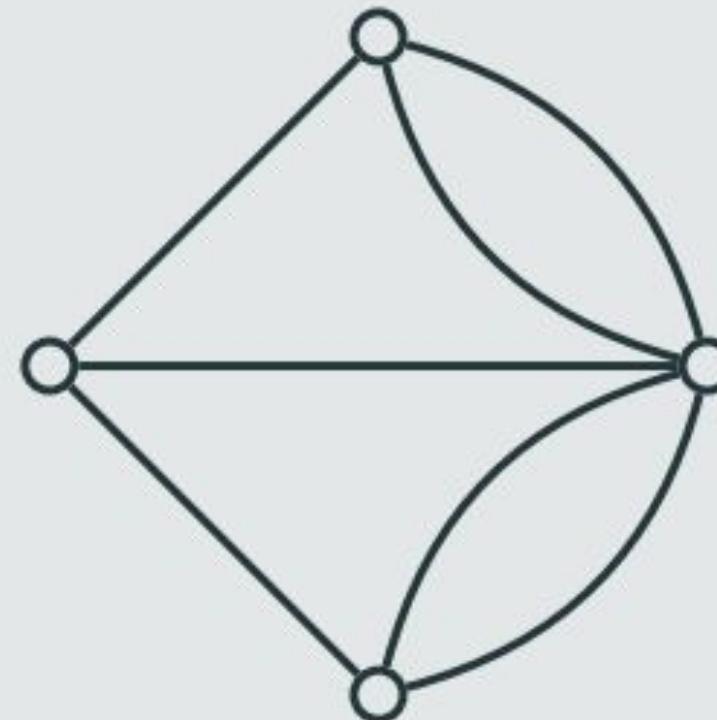
- (i) G is connected and
- (ii) exactly two vertices have odd degree.

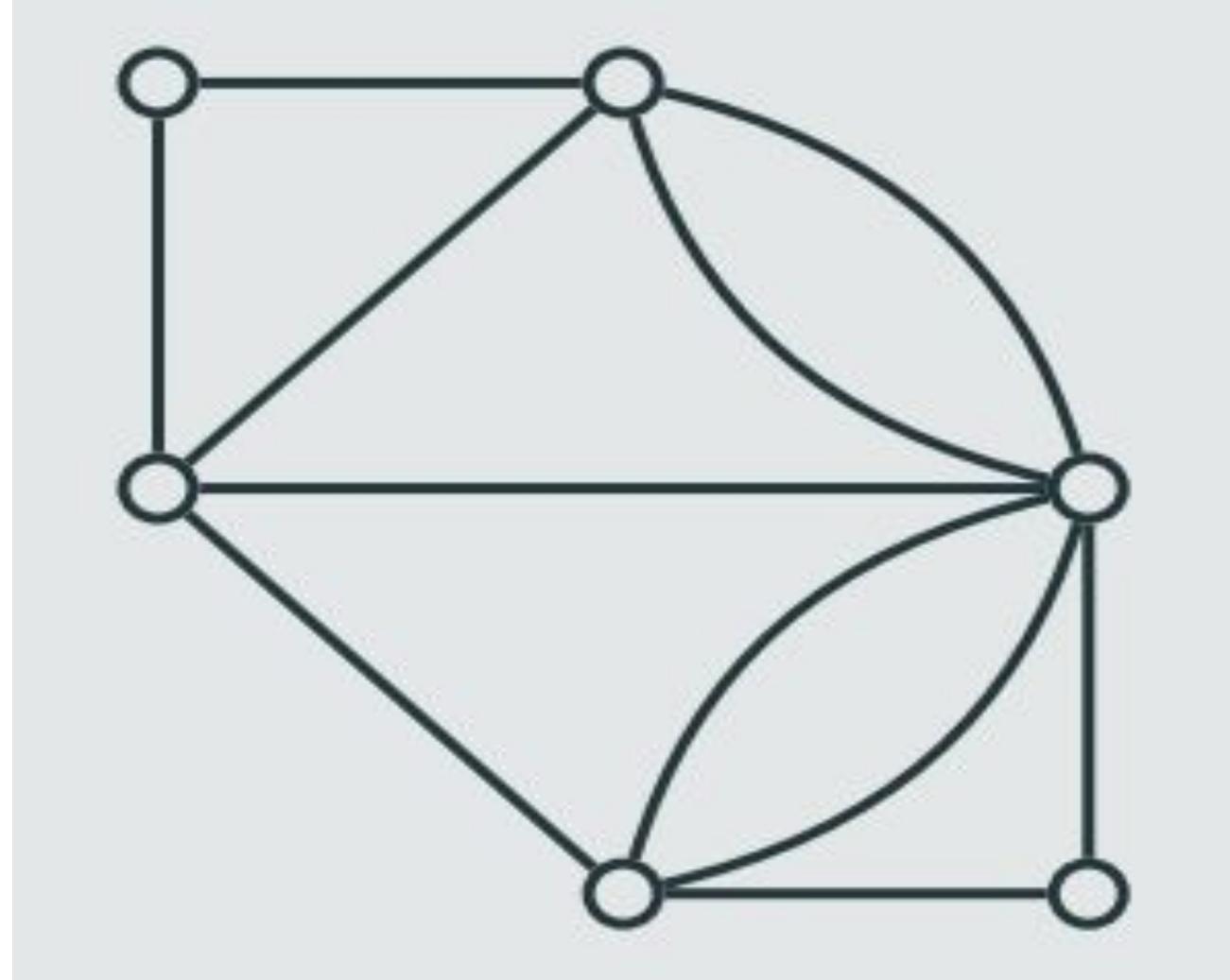
Criteria

Theorem

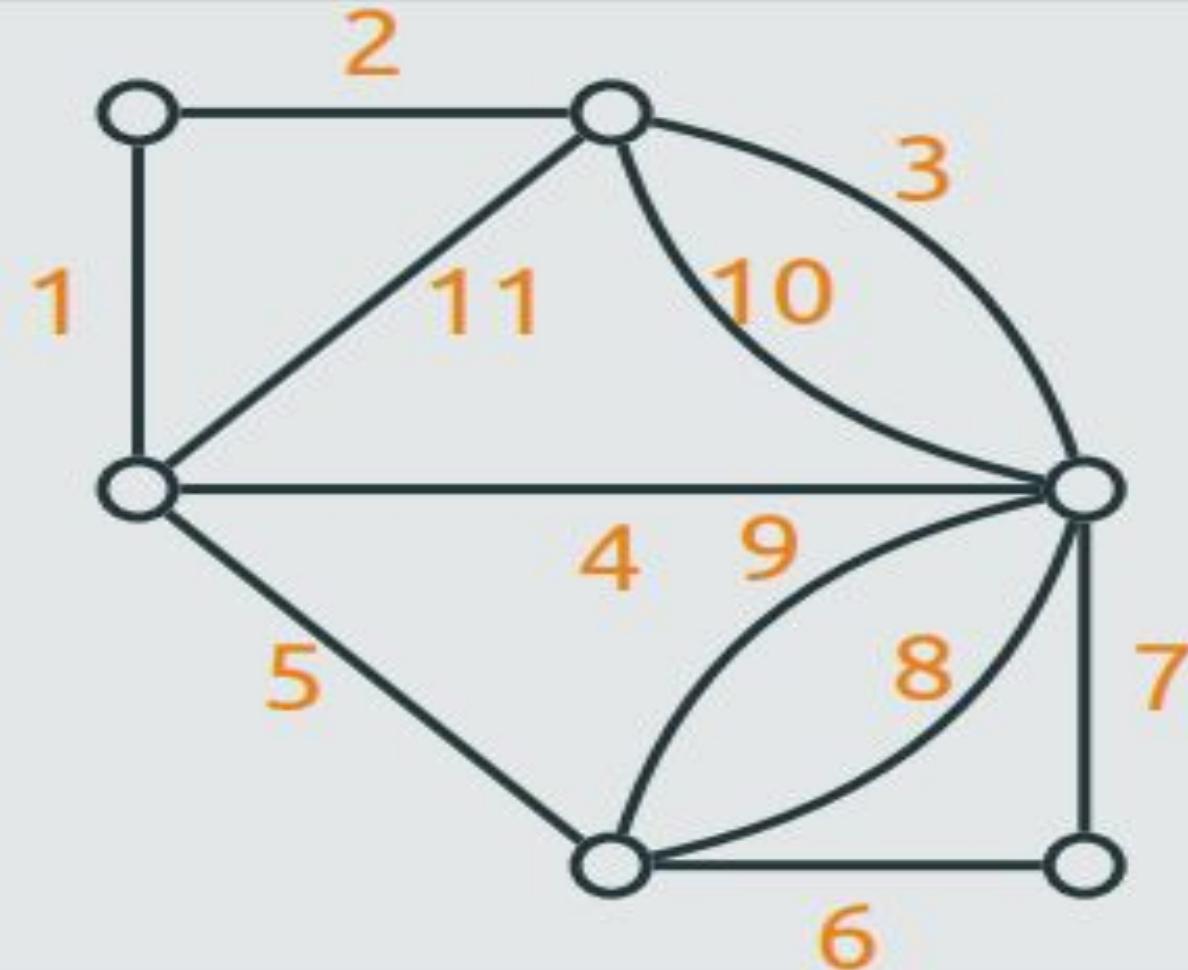
A connected *undirected* graph contains an Eulerian cycle, if and only if the degree of every node is even.

Non-Eulerian graph





Eulerian graph



Criteria

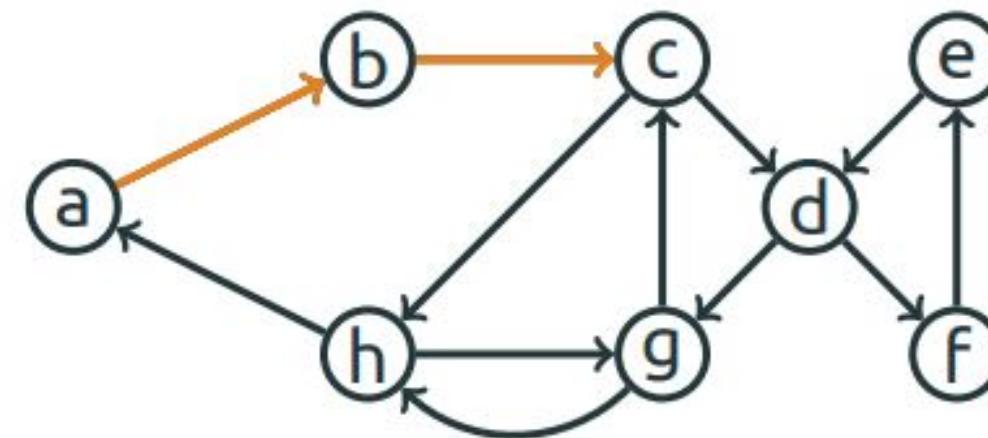
Theorem

A connected *undirected* graph contains an Eulerian cycle, if and only if the degree of every node is even.

Theorem

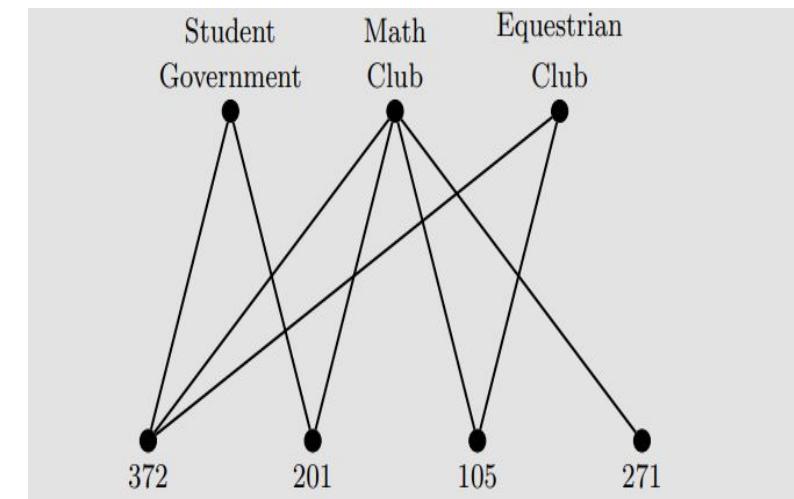
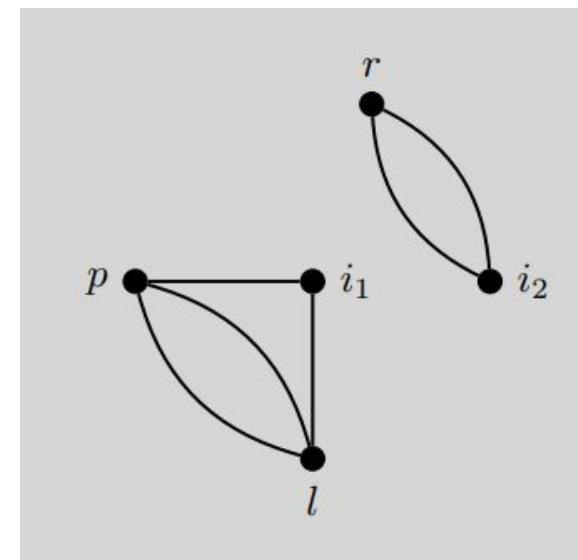
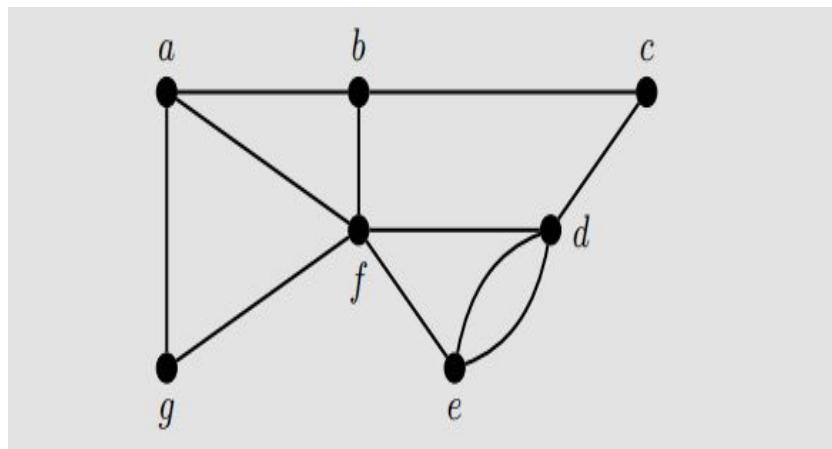
A strongly connected *directed* graph contains an Eulerian cycle, if and only if, for every node, its in-degree is equal to its out-degree.

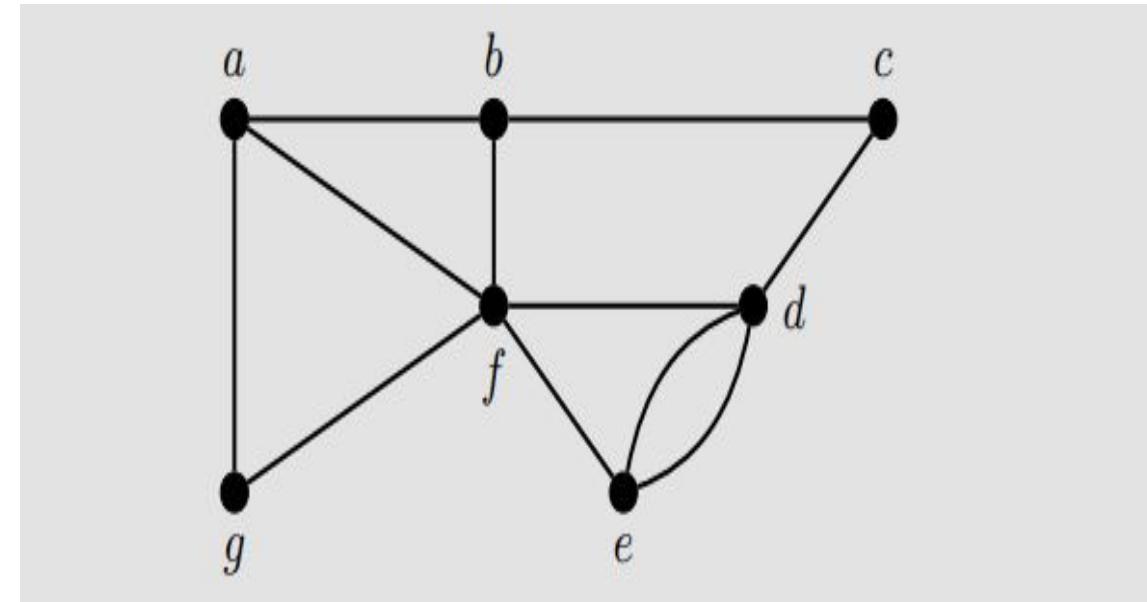
Proof (Directed Case)



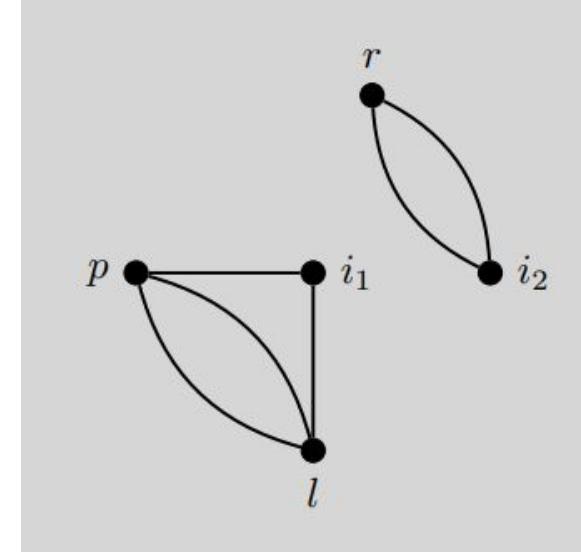
since the graph is balanced, at some point
we'll return back to the starting node

Example 2.3 Consider the graphs appearing in the examples from this and the previous chapter. Which ones are eulerian? semi-eulerian? neither?

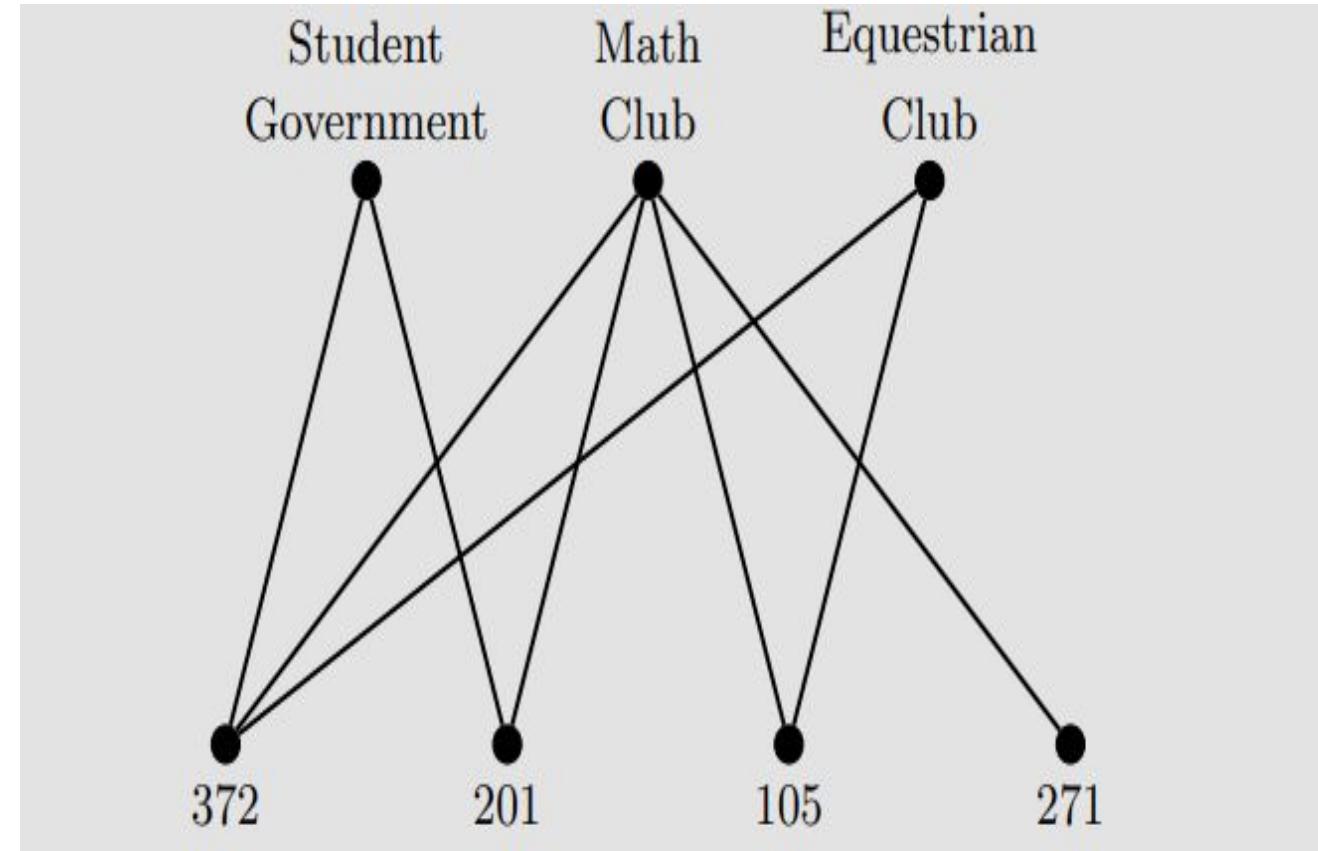




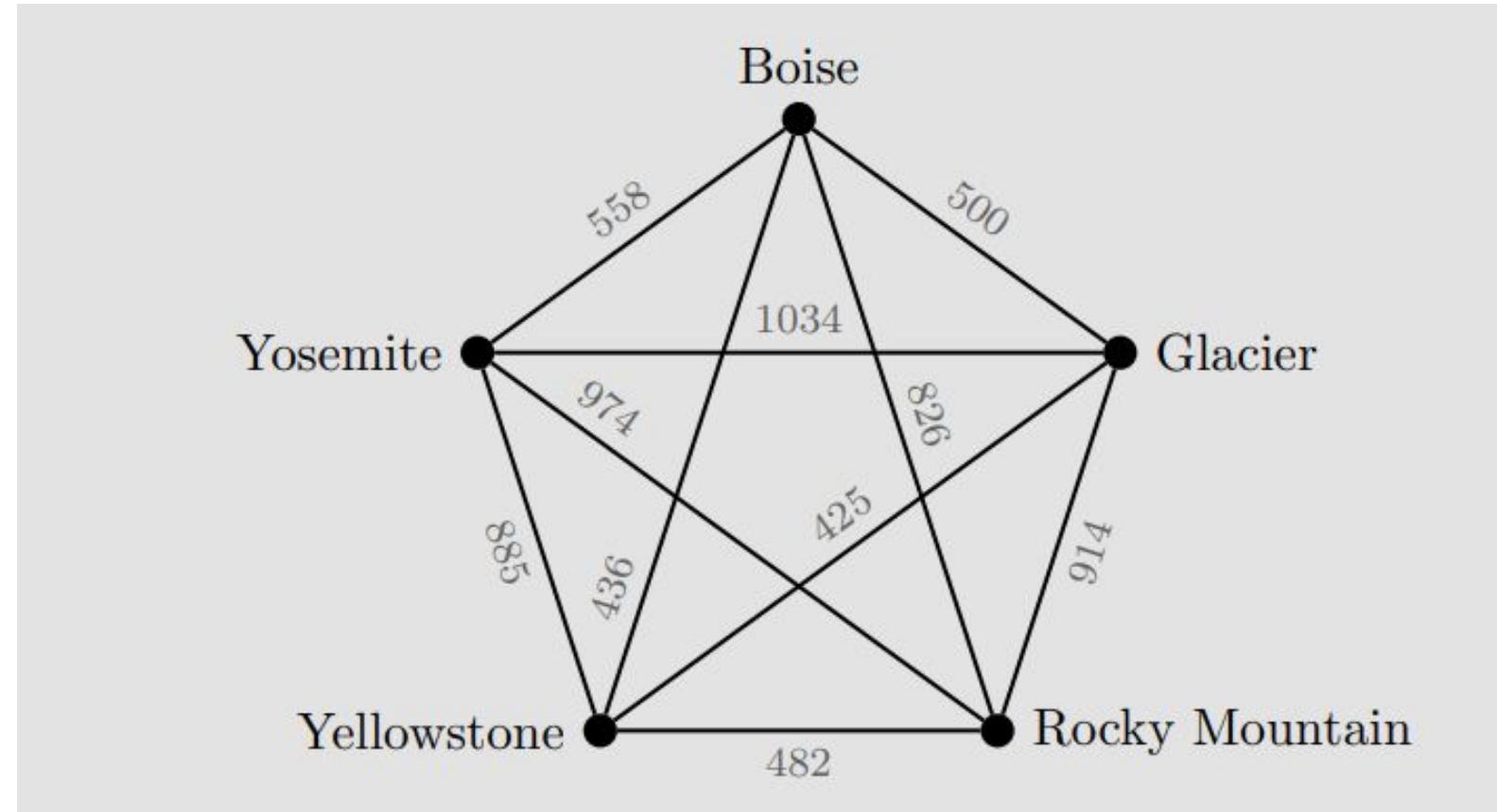
Even though the graph in Example 2.1 is connected, it is neither eulerian nor semi-eulerian since it has more than two odd vertices (namely, a, b, e , and f).

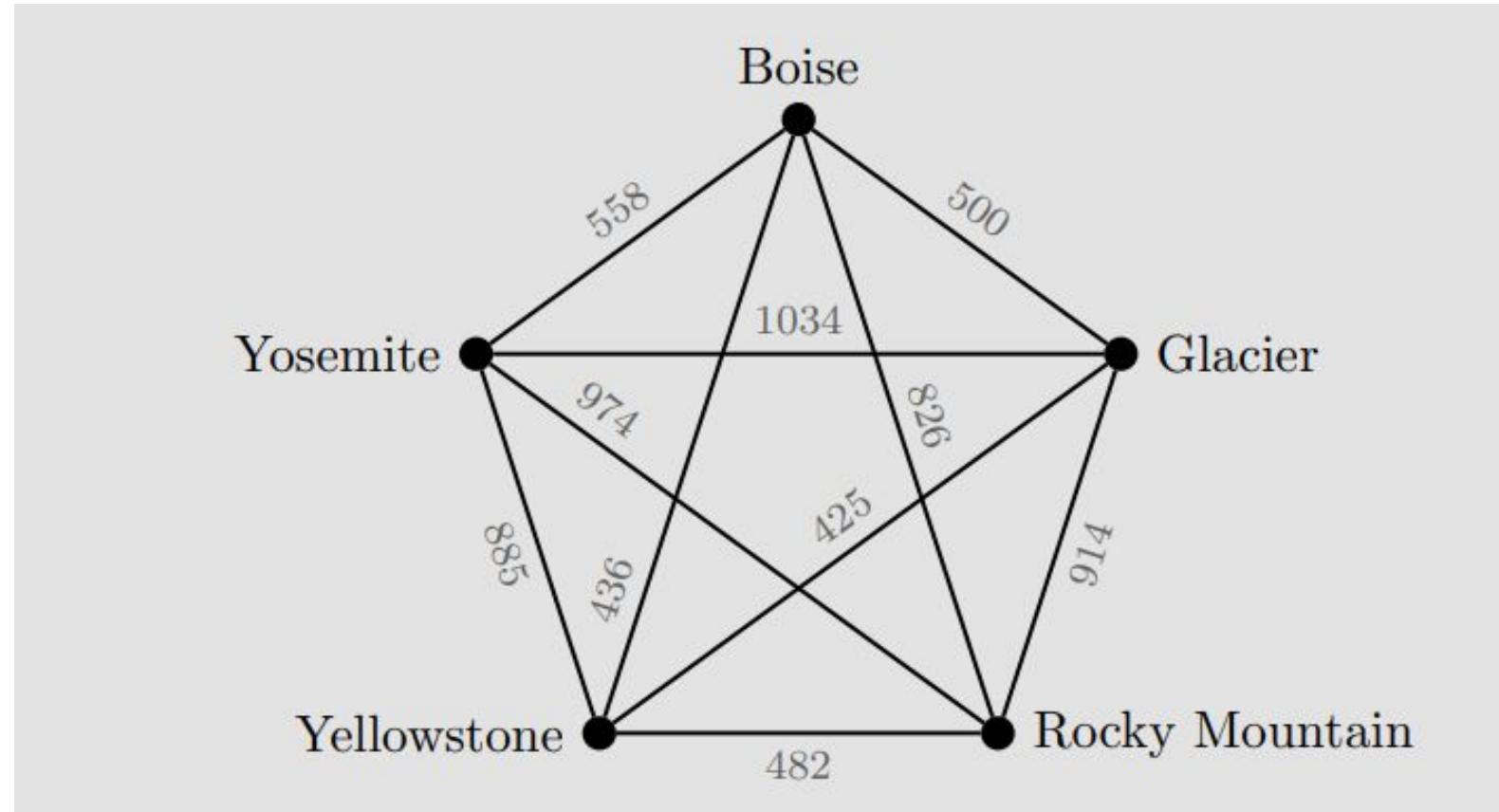


- The graph representing Island City in Example 2.2 is not connected, so it is neither eulerian nor semi-eulerian.



- The graph in Example 1.11 is semi-eulerian since it is connected and exactly two vertices are odd (namely, 372 and 271).





- The graph in Example 1.7 is eulerian since it is connected and all the vertices have degree 4.

Algorithms of Finding Eulerian Circuit (or trails)

Fleury's Algorithm

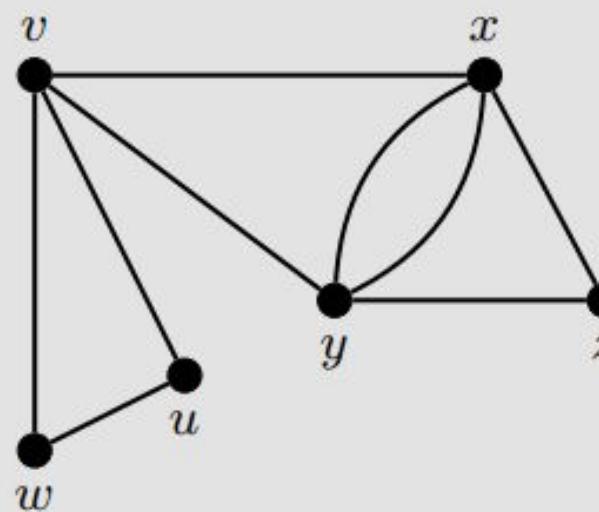
Input: Connected graph G where zero or two vertices are odd.

Steps:

1. Choose a starting vertex, call it v . If G has no odd vertices, then any vertex can be the starting point. If G has exactly two odd vertices, then v must be one of the odd vertices.
2. Choose an edge incident to v that is unlabeled and label it with the number in which it was chosen, ensuring that the graph consisting of unlabeled edges remains connected.
3. Travel along the edge to its other endpoint.
4. Repeat Steps (2) and (3) until all edges have been labeled.

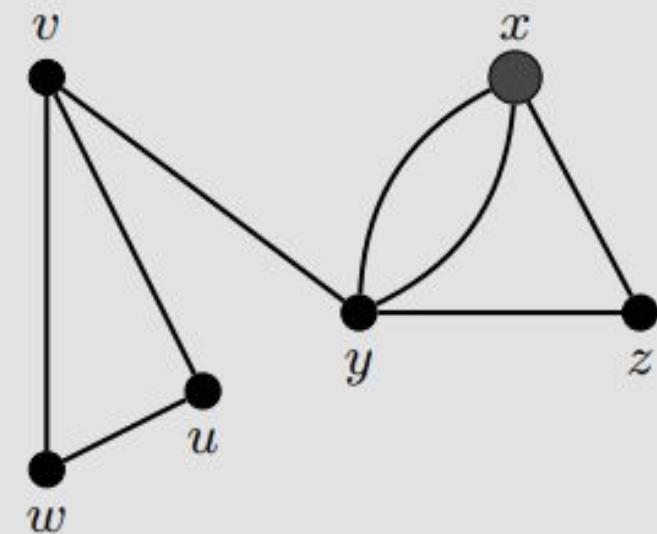
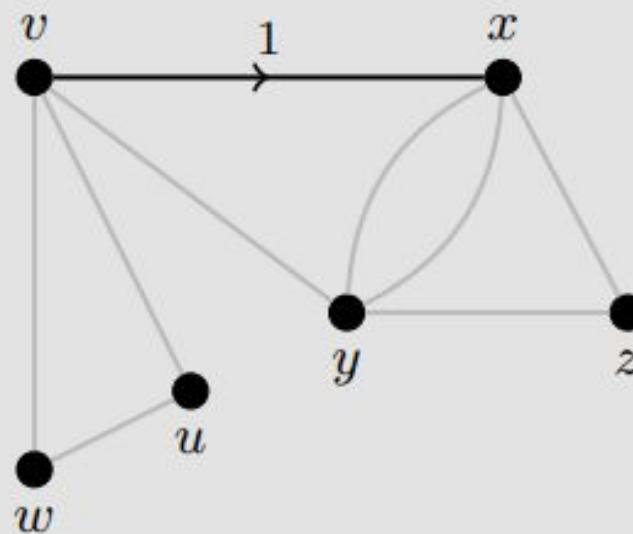
Output: Labeled eulerian circuit or trail.

Example 2.4 Input: A connected graph (shown below) where every vertex has even degree. We are looking for an eulerian circuit.

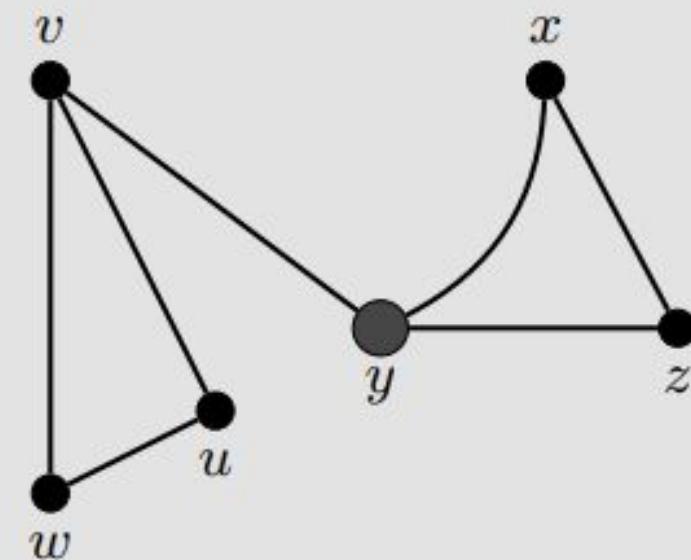
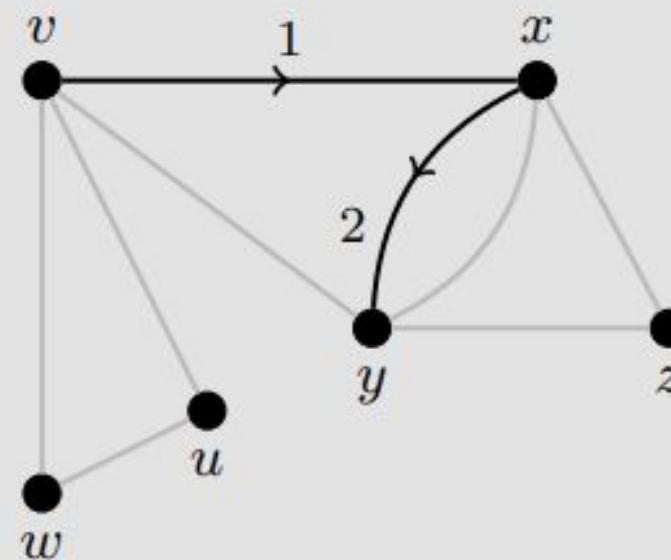


Step 1: Since no starting vertex is explicitly stated, we choose vertex v to be the starting vertex.

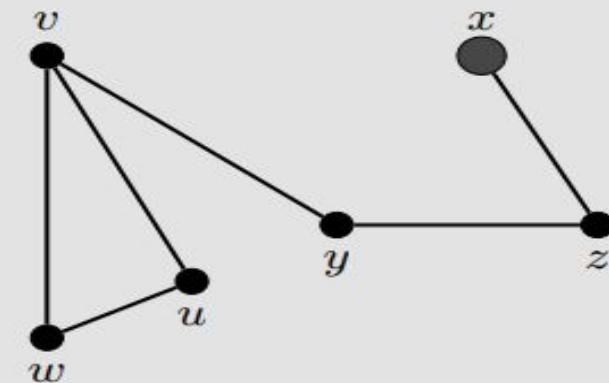
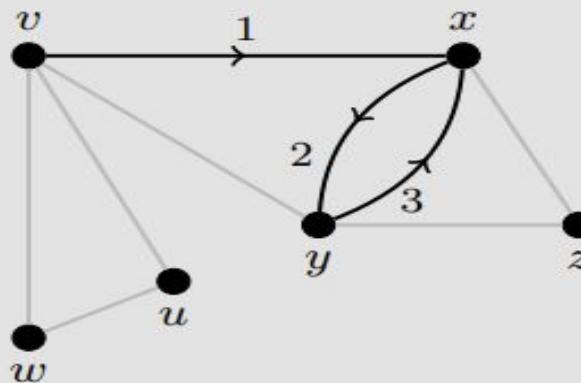
Step 2: We can choose any edge incident to v . Here we chose vx . The labeled graph is on the left and the unlabeled portions are shown on the right with edges removed that have already been chosen.



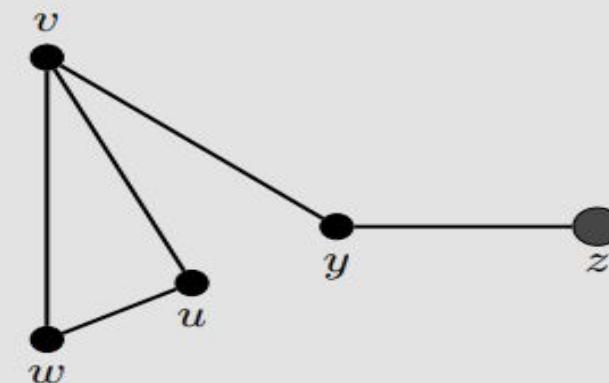
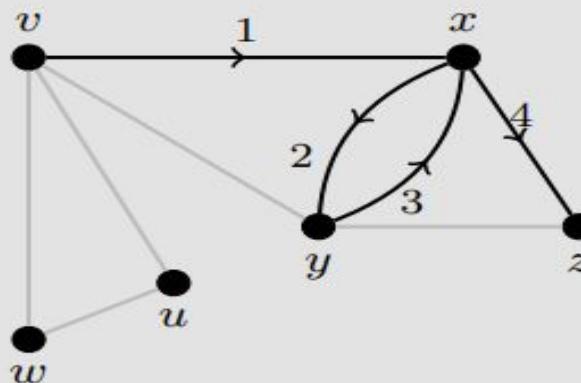
Step 3: Looking at the graph to the right, we can choose any edge out of x . Here we chose xy . The labeled and unlabeled graphs have been updated below.



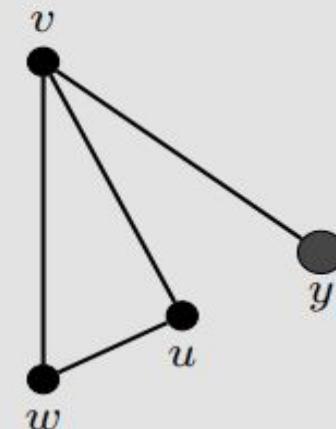
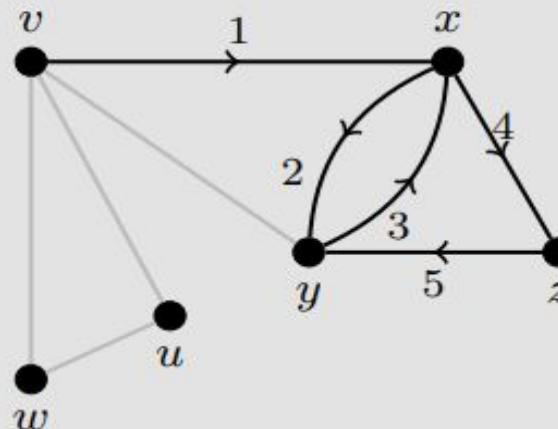
Step 4: At this point we cannot choose yv , as its removal would disconnect the unlabeled graph shown on the right in Step 3. However, yx and yz are both valid choices. Here we chose yx .



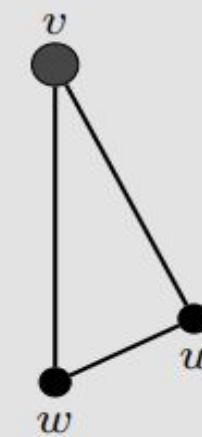
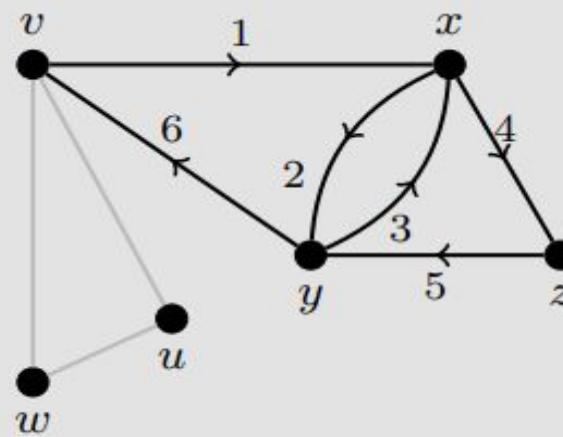
Step 5: There is only one available edge xz .



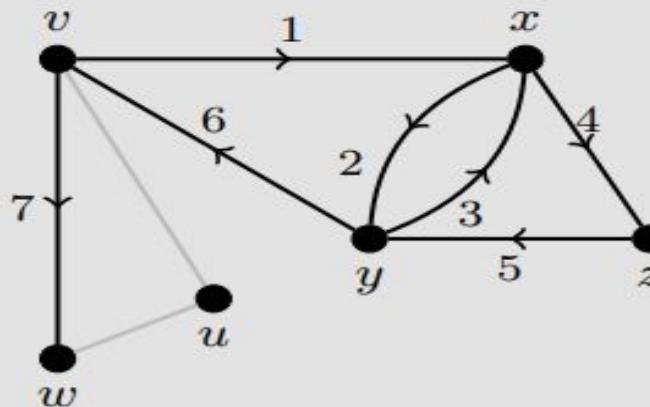
Step 6: There is only one available edge zy .



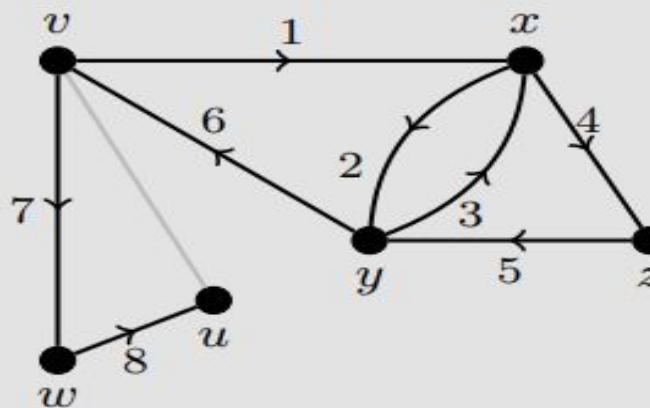
Step 7: There is only one available edge yv .



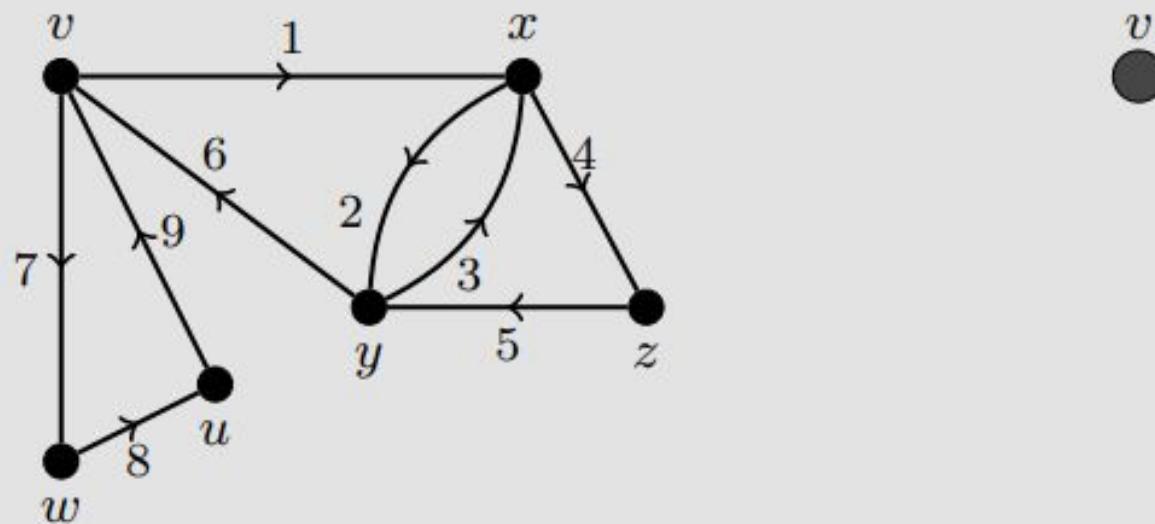
Step 8: Both vw and vu are valid choices for the next edge. Here we chose vw .



Step 9: There is only one available edge wu .



Step 10: There is only one available edge uv .



Output: The graph above on the left has an eulerian circuit labeled, starting and ending at vertex v .

Hierholzer's Algorithm

Input: Connected graph G where all vertices are even.

Steps:

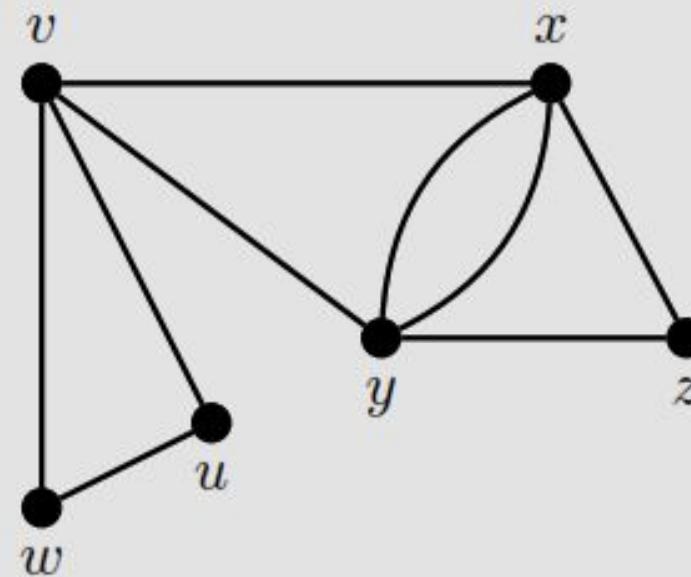
1. Choose a starting vertex, call it v . Find a circuit C originating at v .
2. If any vertex x on C has edges not appearing in C , find a circuit C' originating at x that uses two of these edges.
3. Combine C and C' into a single circuit C^* .
4. Repeat Steps (2) and (3) until all edges of G are used.

Output: Labeled eulerian circuit.

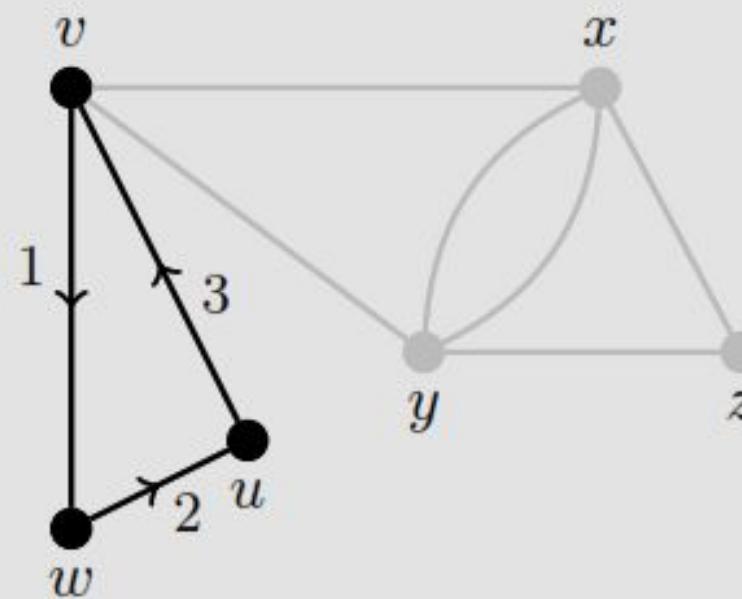
NOTE:

Hierholzer's Algorithm requires the graph to be **eulerian**,
whereas **Fleury's Algorithm** allows for the graph to be **eulerian or semi-eulerian**.

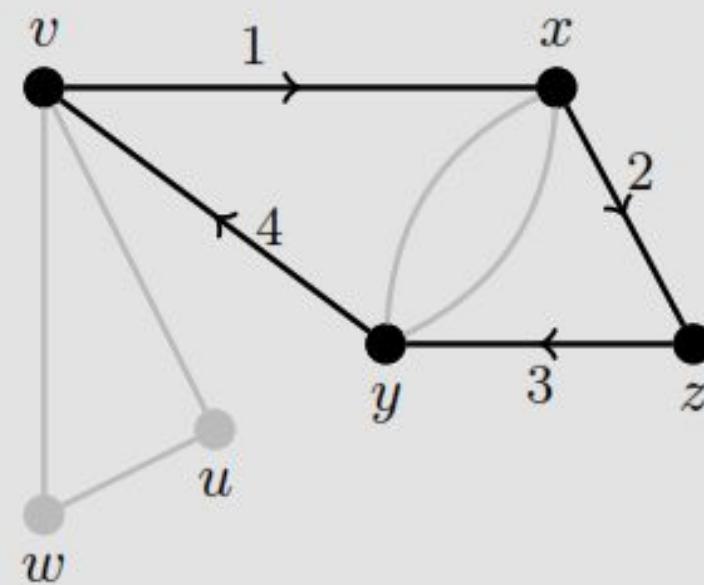
Example 2.5 Input: A connected graph where every vertex has even degree. We are looking for an eulerian circuit.



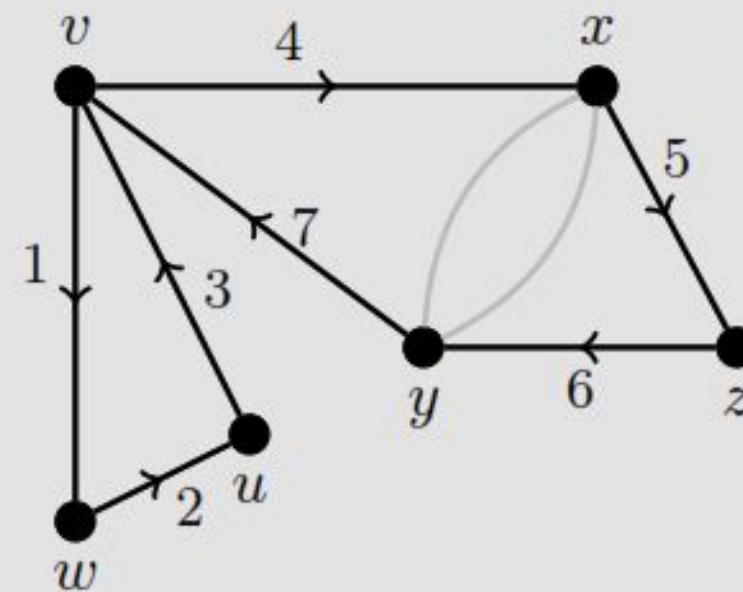
Step 1: Since no starting vertex is explicitly stated, we choose v and find a circuit originating at v . One such option is highlighted below.



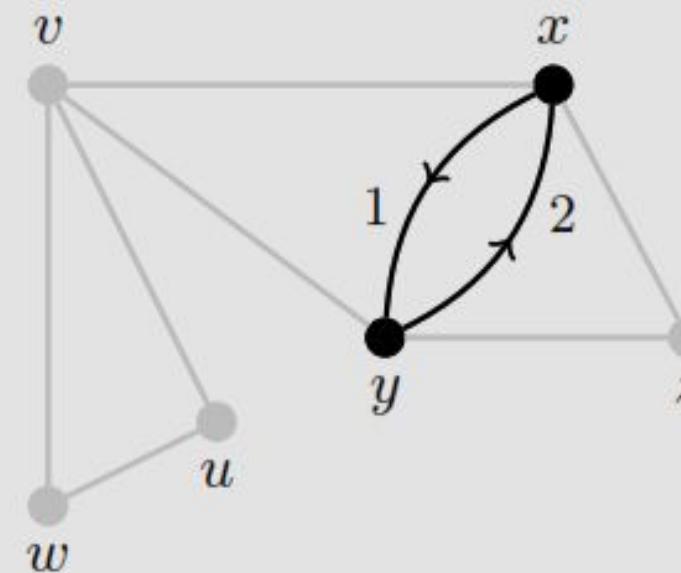
Step 2: As $\deg(v) = 4$ and two edges remain for v (shown in gray in the previous figure), a second circuit starting at v is needed. One option is shown below.



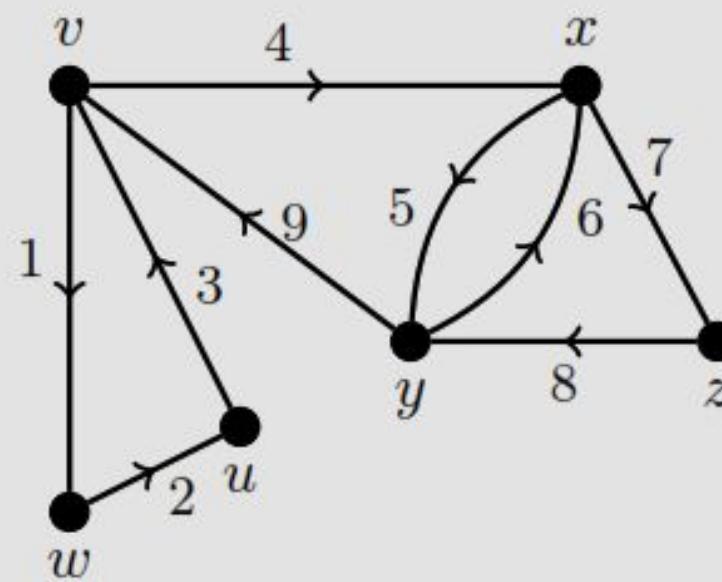
Step 3: Combine the two circuits from Step 1 and Step 2. There are multiple ways to combine two circuits, but it is customary to travel the first circuit created and then travel the second.



Step 4: As $\deg(x) = 4$ and two edges remain for x (shown in gray above), a circuit starting at x is needed. It is shown below.



Step 5: Combine the two circuits from Step 3 and Step 4.



Output: The graph above gives a labeled eulerian circuit originating at v .

- *There are advantages and disadvantages for these two methods; in particular, both Fleury's and Hierholzer's Algorithms will find an eulerian circuit when one exists, whereas only Fleury's can be used to find an eulerian trail (see Exercise 2.10 for a modification of Hierholzer's that will find an eulerian trail)*
- *Try a few more examples and make additional comparisons. In practice, either algorithm is a good choice for finding an eulerian circuit. Pick the method that works best for you.*

Definition 2.10 A cycle in a graph G that contains every vertex of G is called a *hamiltonian cycle*. A path that contains every vertex is called a *hamiltonian path*. A graph that contains a hamiltonian cycle is called *hamiltonian*.

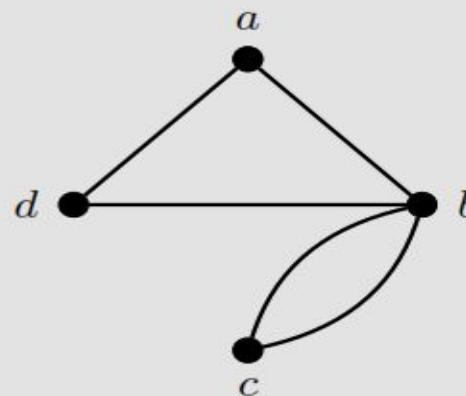
Hamiltonian Cycle

A **Hamiltonian cycle** visits every node of a graph exactly once.

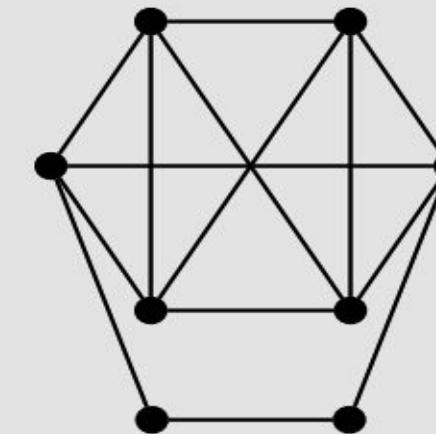
Simple Criteria?

- No simple criteria is known for the Hamiltonian cycle problem

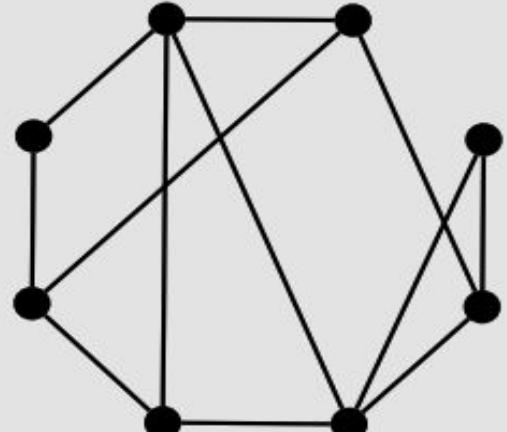
Example 2.8 For each of the graphs below, determine if they have hamiltonian cycles (and paths) and eulerian circuits (and trails).



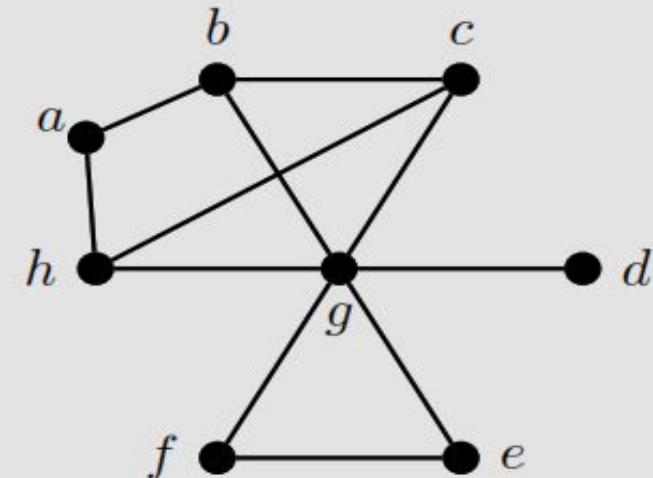
G_1



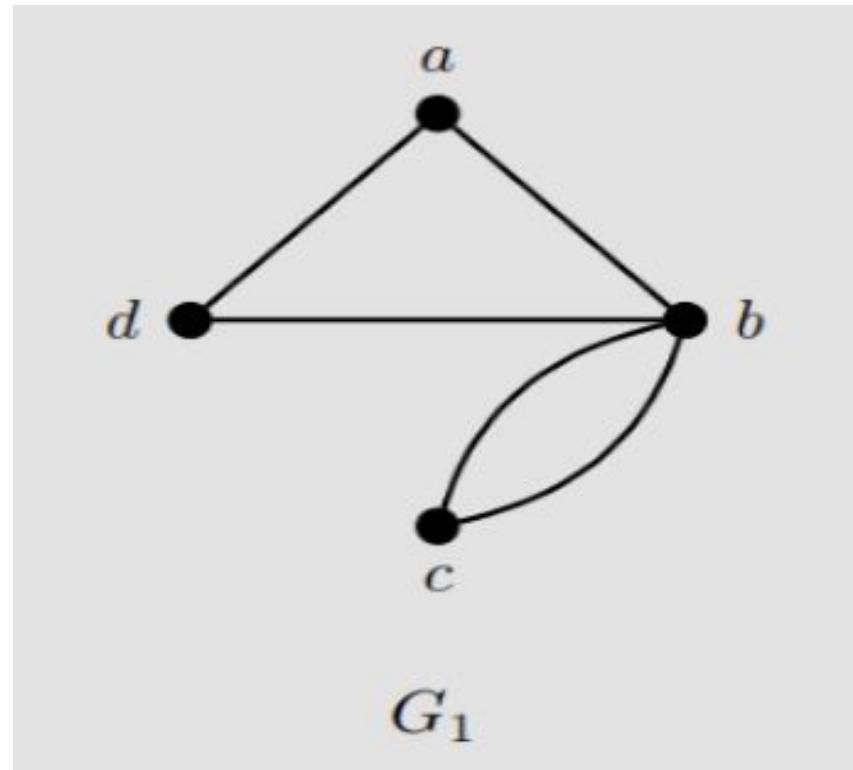
G_2



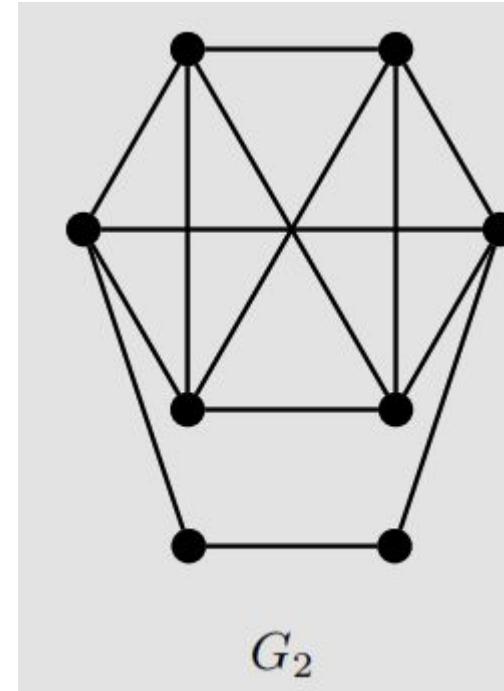
G_3



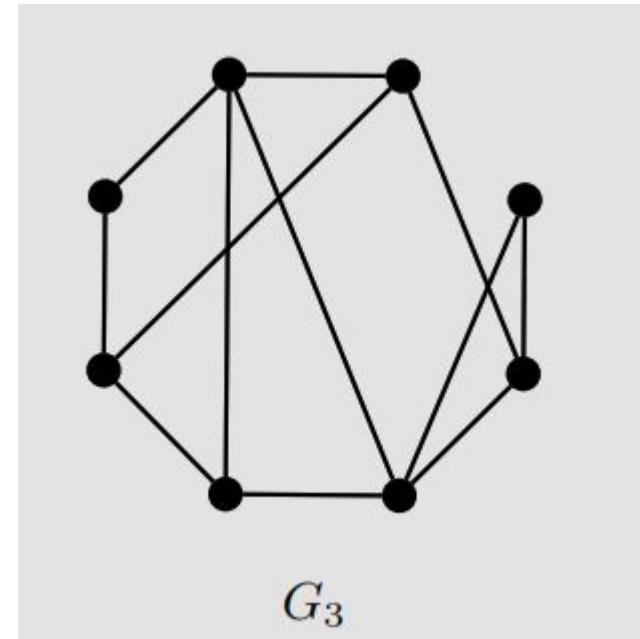
G_4



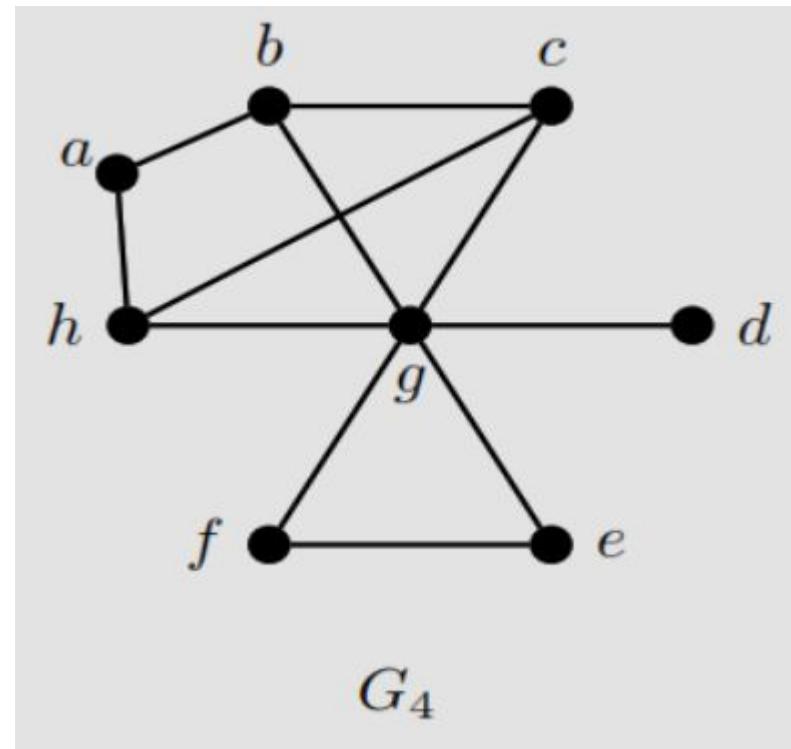
Solution: Since G_1 is connected and all vertices are even, we know it has an **eulerian circuit**. There is no hamiltonian cycle since we need to include c in the cycle and by doing so we have already passed through b twice, making it impossible to visit a and d .



Since G_2 is connected and all vertices are even, we know it is eulerian. Hamiltonian cycles and hamiltonian paths also exist. To find one such path, remove any one of edges from a hamiltonian cycle.



Some vertices of G_3 are odd, so we know it is not eulerian. Moreover, since more than two vertices are odd, the graph is not semi-eulerian. However, this graph does have a **hamiltonian cycle** (and so also a hamiltonian path). Can you find it?

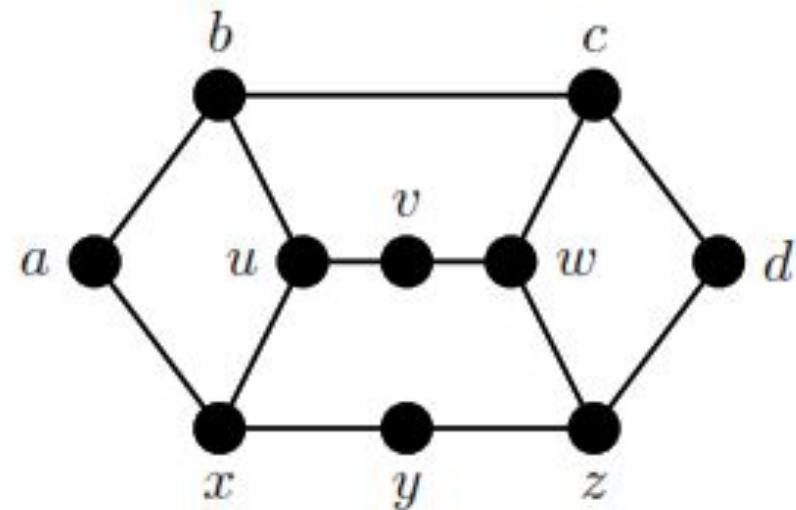


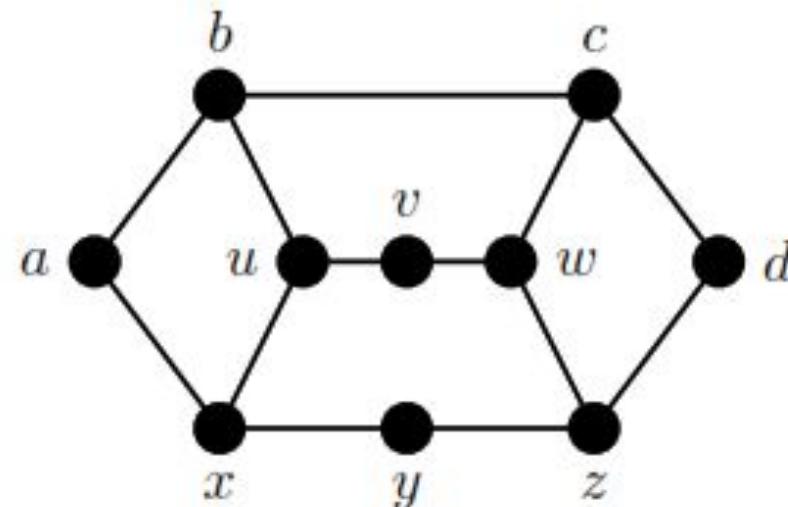
Four vertices of G_4 are odd, we know it is neither eulerian nor semi-eulerian. This graph does not have a hamiltonian cycle since d cannot be a part of any cycle. Moreover, this graph does not have a hamiltonian path since any traversal of every vertex would need to travel through g multiple times.

- If a graph has a hamiltonian cycle, it automatically has a hamiltonian path (just leave off the last edge of the cycle to obtain a path).
- If a graph has a hamiltonian path, it may or may not have a hamiltonian cycle.

Properties of Hamiltonian Graphs

- (1) G must be connected.
- (2) No vertex of G can have degree less than 2.
- (3) G cannot contain a ***cut-vertex***, that is a vertex whose removal disconnects the graph.
- (4) If G contains a vertex x of degree 2 then both edges incident to x must be included in the cycle.
- (5) If two edges incident to a vertex x must be included in the cycle, then all other edges incident to x cannot be used in the cycle.
- (6) If in the process of attempting to build a hamiltonian cycle, a cycle is formed that does not span G , then G cannot be hamiltonian.



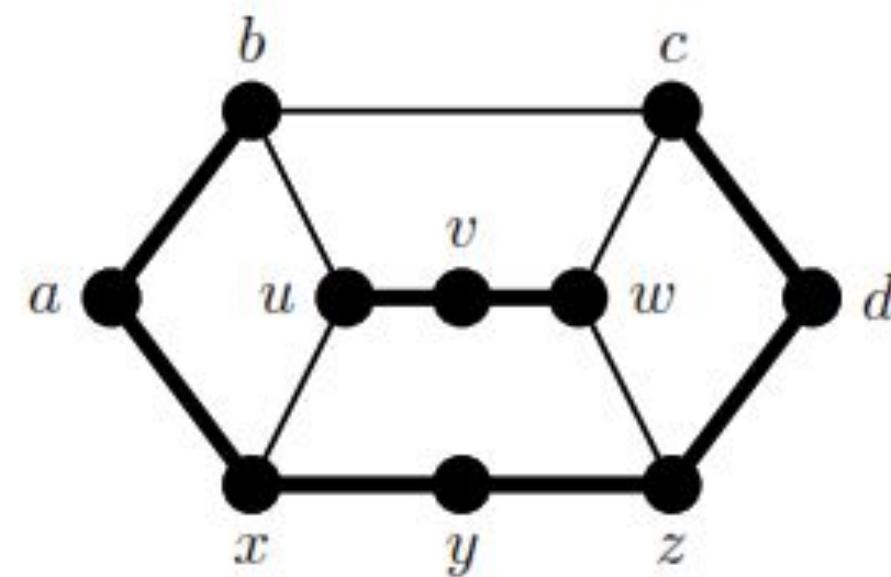


There are four vertices of degree two in G , namely, a, d, v and y . If G has a Hamiltonian cycle C , then by rule (i), the eight edges $ab, ax, cd, dz, uv, vw, xy$ and yz are required edges in $E(C)$ as indicated in bold in Fig. 5.1.4(b).

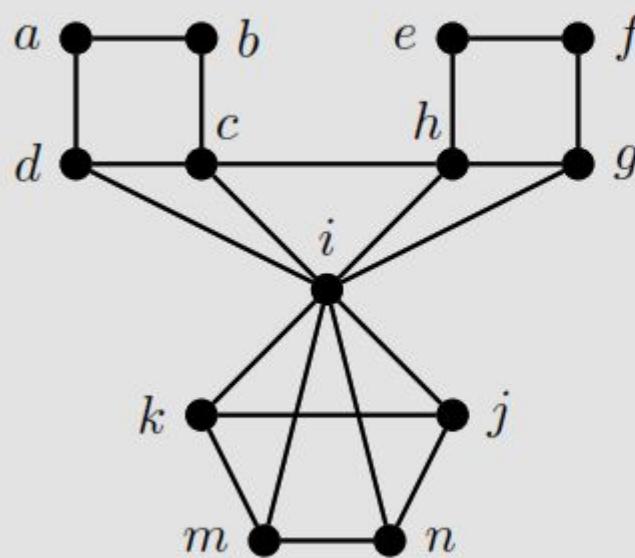
It is now obvious that the set of eight required edges can be expanded to form such a C by adding bu and cw . Thus G is a Hamiltonian graph.

There are four vertices of degree two in G , namely, a, d, v and y . If G has a Hamiltonian cycle C , then by rule (i), the eight edges $ab, ax, cd, dz, uv, vw, xy$ and yz are required edges in $E(C)$ as indicated in bold in Fig. 5.1.4(b).

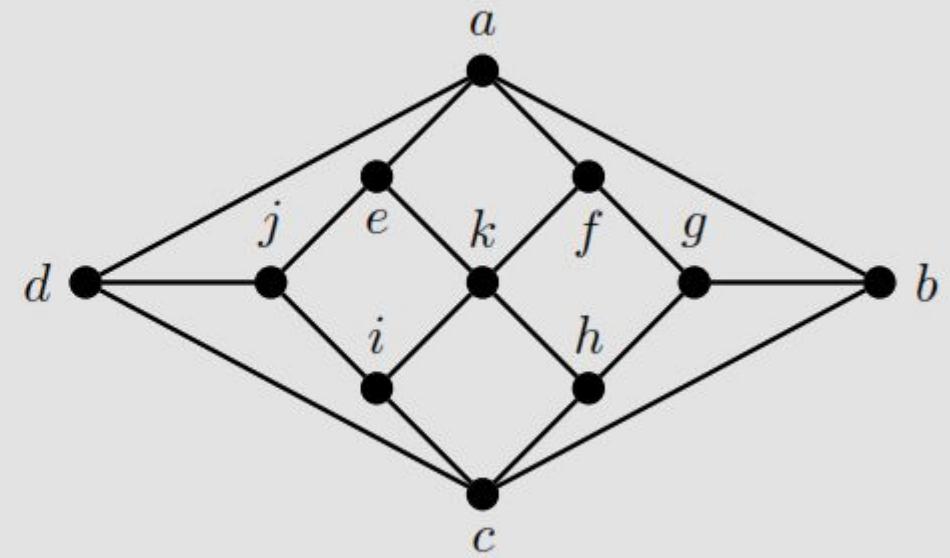
It is now obvious that the set of eight required edges can be expanded to form such a C by adding bu and cw . Thus G is a Hamiltonian graph.



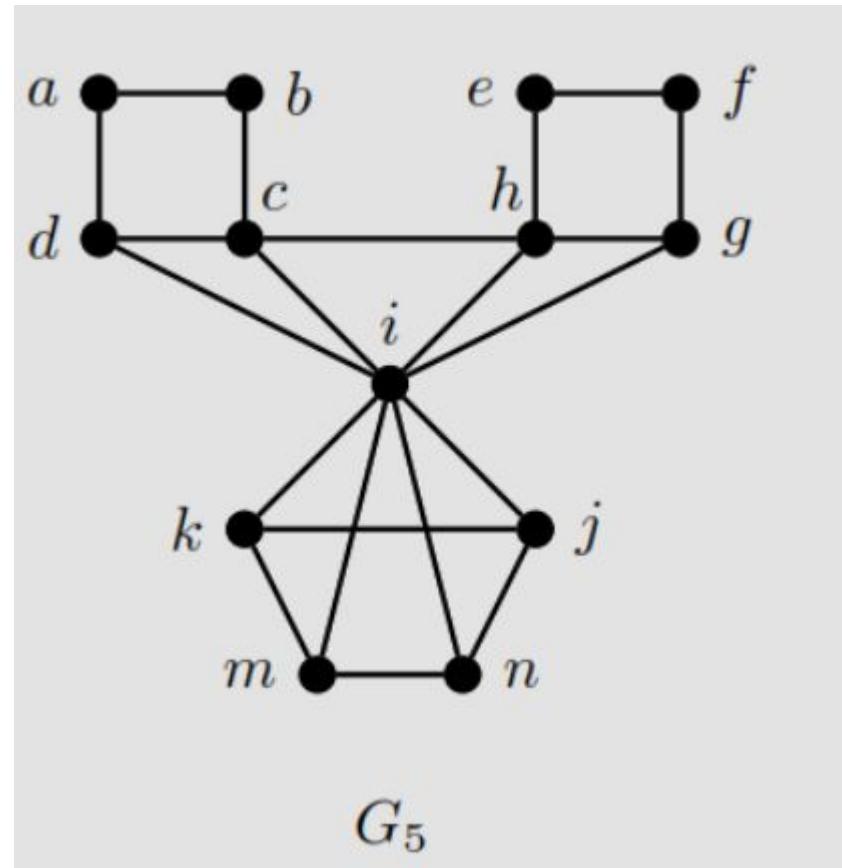
Example 2.9 Use the properties listed above to show that the graphs below are not hamiltonian.



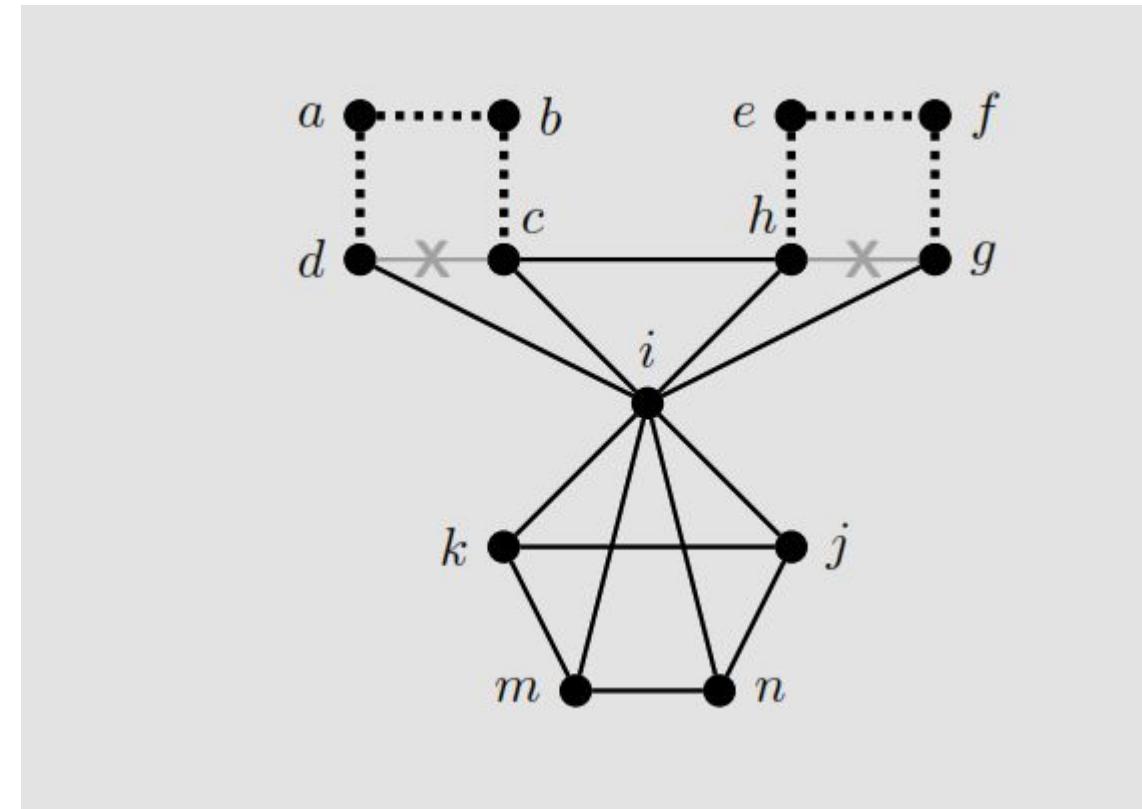
G_5



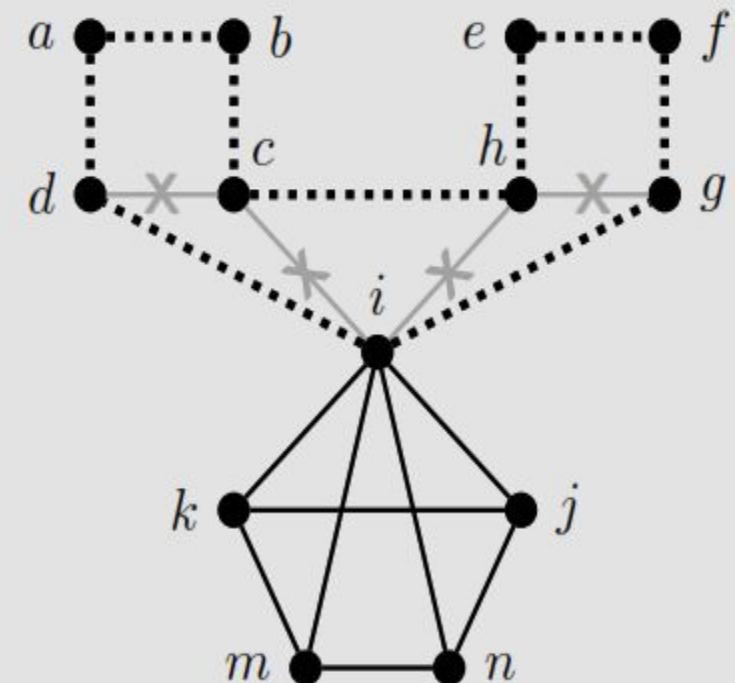
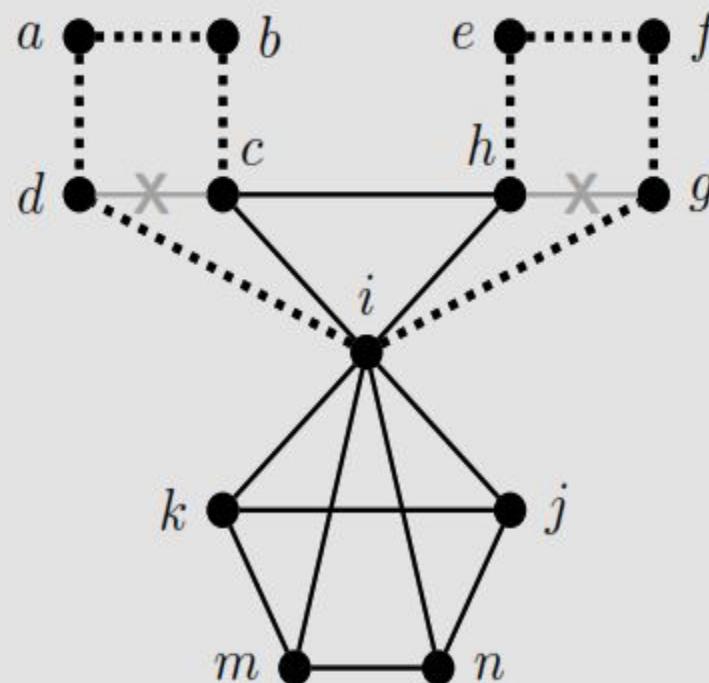
G_6



Solution: For G_5 , notice that vertices a , b , e , and f all have degree 2 and so all the edges incident to these vertices must be included in the cycle. But then edges cd and hg cannot be a part of a cycle since they would create smaller cycles that do not include all of the vertices in G_5 .



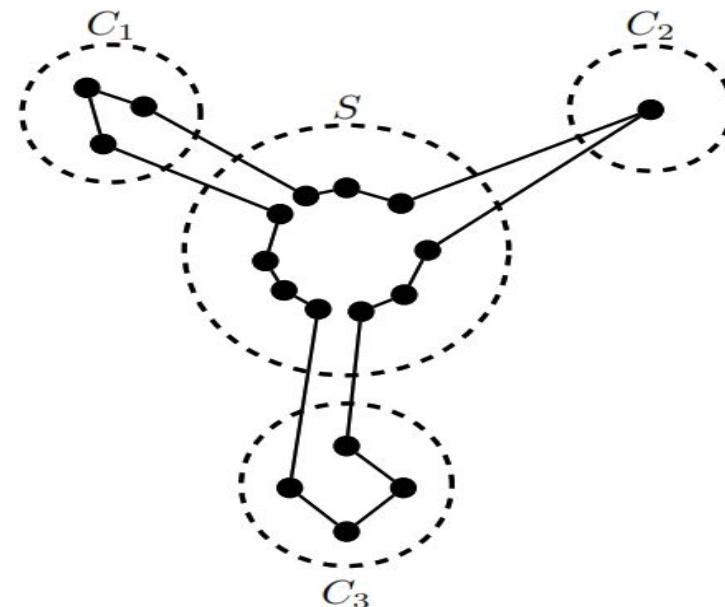
Since only one edge remains incident to d and g , namely di and gi , then these must be a part of the cycle. But in doing so, we would be forced to use ch since the other edges incident to i could not be chosen. This creates a cycle that does not span G_5 , and so G_5 is not hamiltonian.



LECTURE # 10, 11 & 12

Proposition 2.11 If G is a graph with a hamiltonian cycle, then $G - S$ has at most $|S|$ components for every nonempty set $S \subseteq V$.

Proof: Assume G has a hamiltonian cycle C and S is a nonempty subset of vertices. Then as we traverse C we will leave and return to S from distinct vertices in S since no vertex can be used more than once in a cycle. Since C must span $V(G)$, we know S must have at least as many vertices as the number of components of $G - S$.

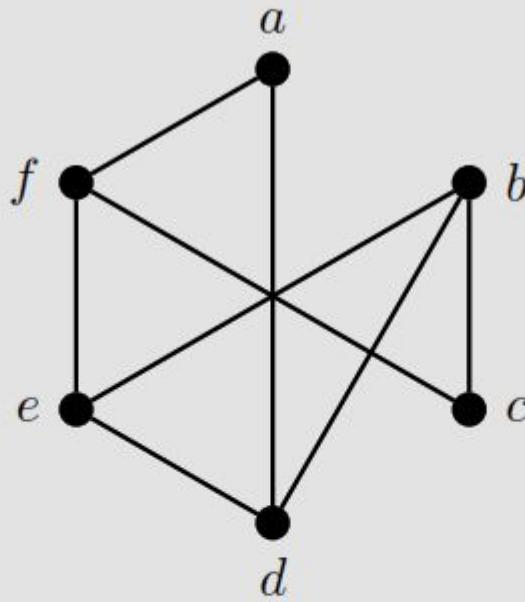


Theorem 2.12 (Dirac's Theorem) Let G be a graph with $n \geq 3$ vertices. If every vertex of G satisfies $\deg(v) \geq \frac{n}{2}$, then G has a hamiltonian cycle.

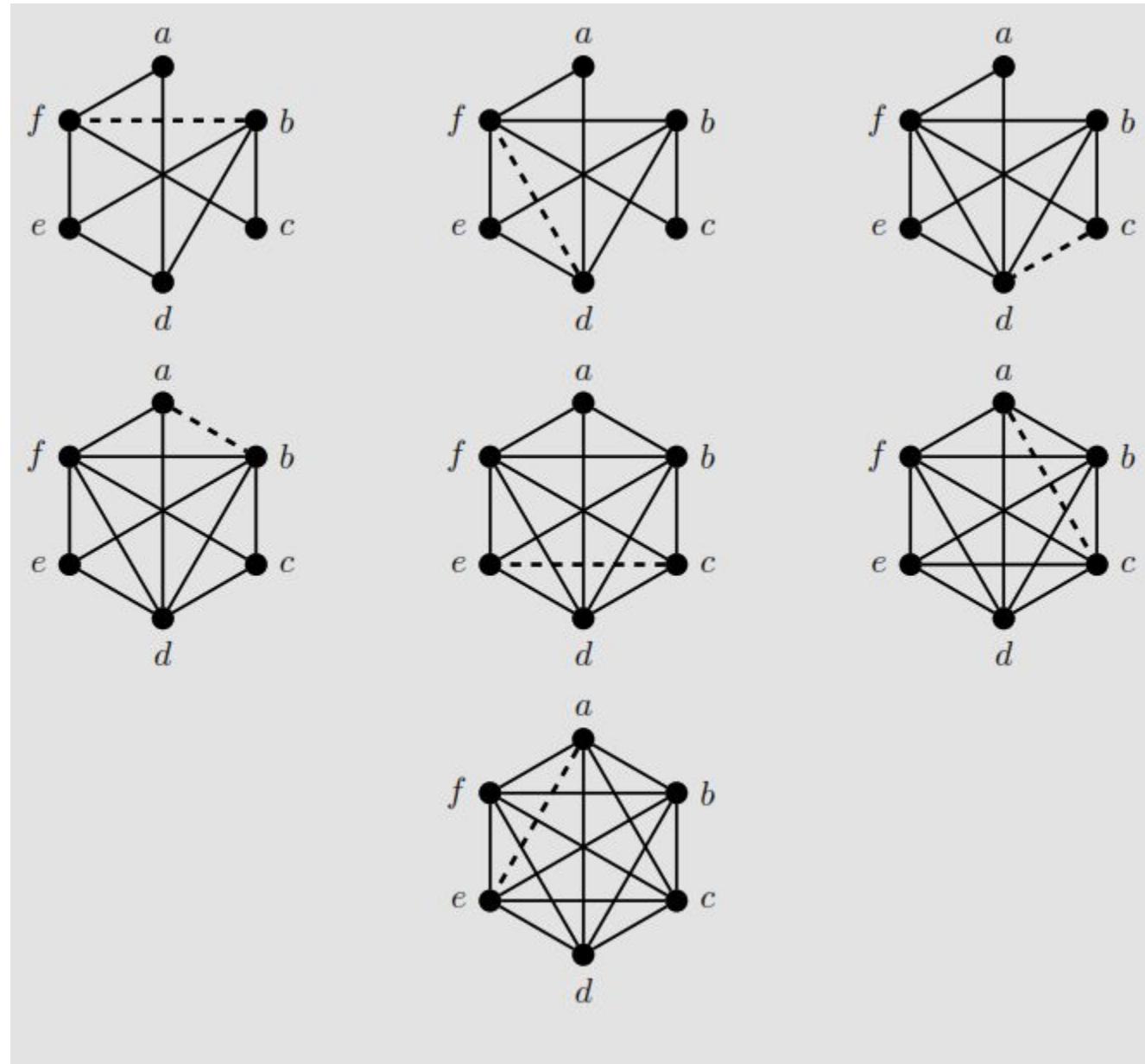
Theorem 2.13 (Ore's Theorem) Let G be a graph with $n \geq 3$ vertices. If $\deg(x) + \deg(y) \geq n$ for all distinct nonadjacent vertices, then G has a hamiltonian cycle.

Definition 2.14 The *hamiltonian closure* of a graph G , denoted $cl(G)$, is the graph obtained from G by iteratively adding edges between pairs of nonadjacent vertices whose degree sum is at least n , that is we add an edge between x and y if $\deg(x) + \deg(y) \geq n$, until no such pair remains.

Example 2.10 Find a hamiltonian closure for the graph below.



Solution: First note that the degrees of the vertices are 2, 3, 2, 3, 3, 3 (in alphabetic order). Since the closure of G is formed by putting an edge between nonadjacent vertices whose degree sum is at least 6, we must begin with edge bf or df . We chose to start with bf . In the sequence of graphs shown on the next page, the edge added at each stage is a thick dashed line.



Complete graphs K_n , $n \geq 3$, are Hamiltonian.

Theorem 2.16 A graph G is hamiltonian if and only if its closure $cl(G)$ is hamiltonian.

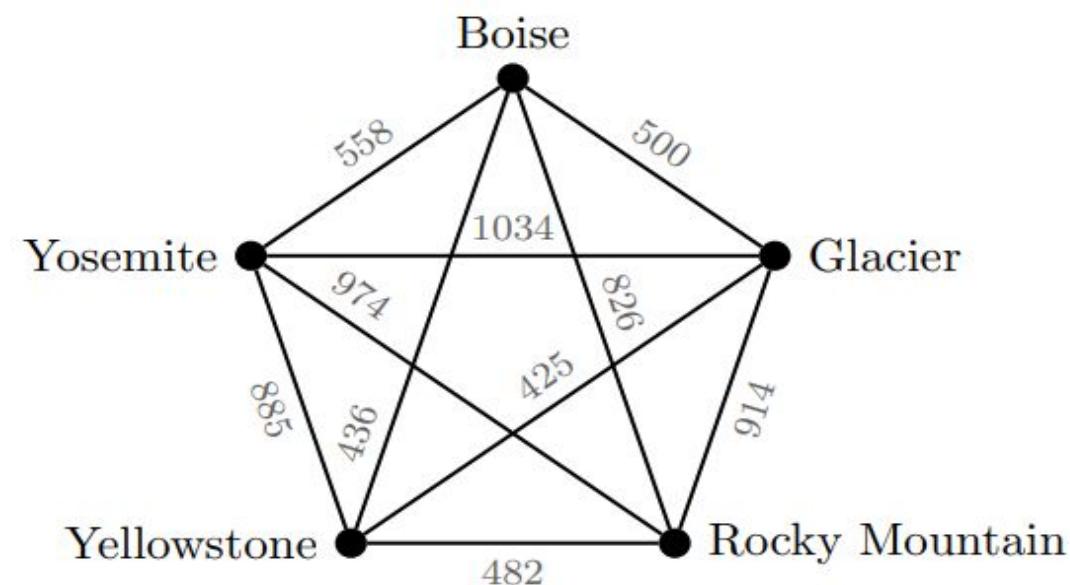
Lemma 2.17 If G is a graph with at least 3 vertices such that its closure $cl(G)$ is complete, then G is hamiltonian.

Theorem 2.18 Let G be a simple graph where the vertices have degree d_1, d_2, \dots, d_n such that $n \geq 3$ and the degrees are listed in increasing order. If for any $i < \frac{n}{2}$ either $d_i > i$ or $d_{n-i} \geq n-i$, then G is hamiltonian.

2.2.1 The Traveling Salesman Problem

The discussion above should make clear the difficulty in determining if a graph is hamiltonian. But what if a graph is known to have a hamiltonian cycle? For example, every complete graph K_n (for $n \geq 3$) must contain a hamiltonian cycle since it satisfies the criteria of Dirac's Theorem. In this scenario, finding a hamiltonian cycle is quite elementary, and so, as mathematicians do, we generalize the problem to one in which the edges are no longer equivalent and have a weight associated to them. Then instead of asking whether a graph simply *has* a hamiltonian cycle, we can now ask how do we find the *best* hamiltonian cycle.

Historically, the extensive study of hamiltonian circuits arose in part from a simple question: A traveling salesman has customers in numerous cities; he must visit each of them and return home, but wishes to do this with the least total cost; determine the cheapest route possible for the salesman. In fact,

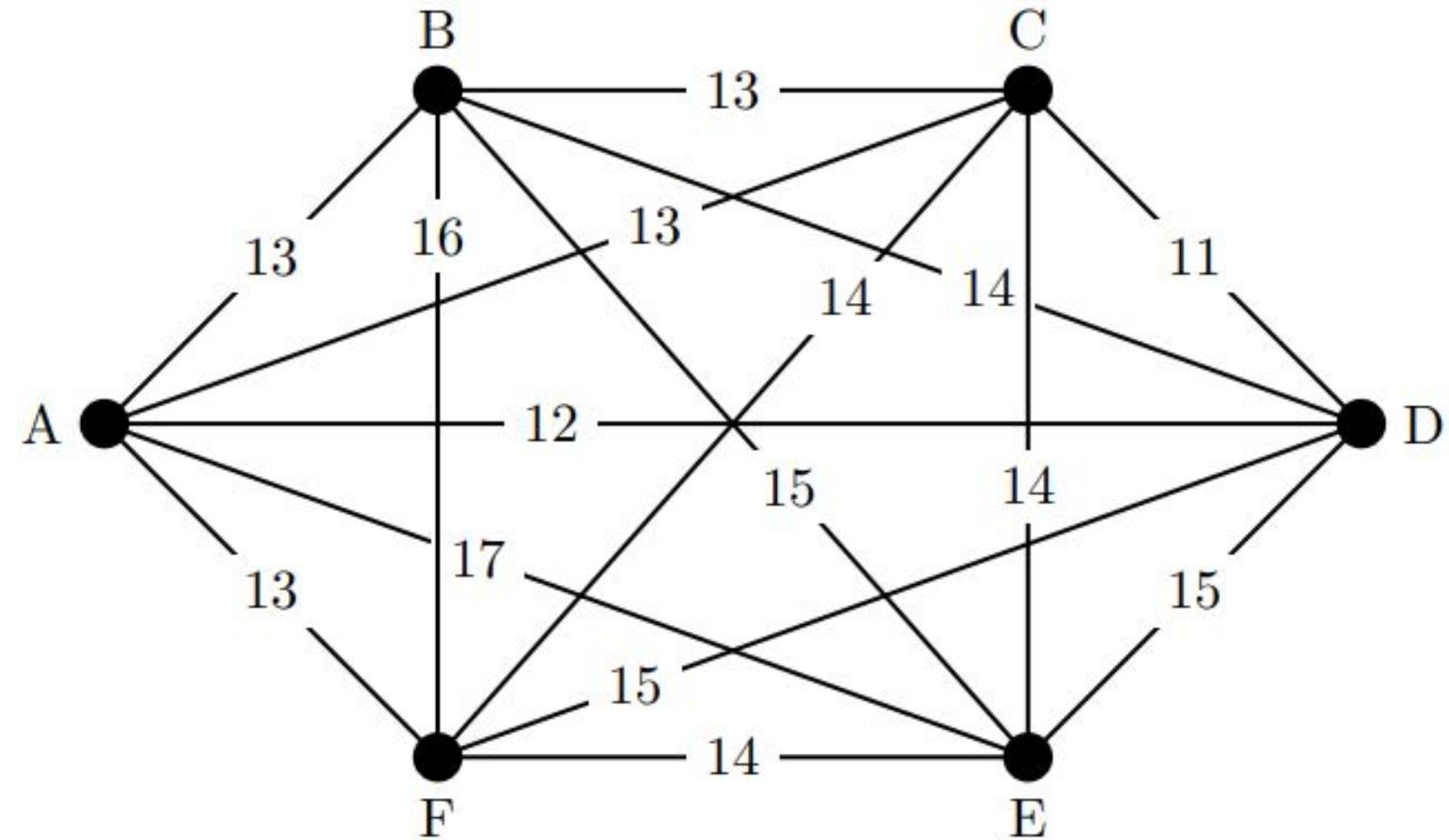


The graph that models the general Traveling Salesman Problem (TSP) is a *weighted complete graph*, such as the one shown above from Example 1.7.

Example 5.4.1. Mr. Lee is a salesman living in town A. Every day, he needs to travel from his house to all the other 5 towns (B to F) to promote his products before returning home in the evening. The 6 towns are well connected by public transport and there are direct buses between each pair of towns. The time (in minutes) it takes for Mr. Lee to travel between each pair of towns is shown in the table below.

Between	A	B	C	D	E	F
A	-	13	13	12	17	13
B	-	-	13	14	15	16
C	-	-	-	11	14	14
D	-	-	-	-	15	15
E	-	-	-	-	-	14
F	-	-	-	-	-	-

How should Mr. Lee plan his route from his home to the other 5 towns so as to minimize his total traveling time?



Clearly, G has many spanning cycles, each representing a route that Mr. Lee can take. The total traveling time would be the sum of all the weights of edges in a particular spanning cycle. The question now reduces to finding a spanning cycle with the **minimum weight**.

The Traveling Salesman Problem. (TSP) Let G be a weighted complete graph of order n . Find a spanning cycle C in G such that

$$w(C) = \sum_{e \in E(C)} w(e)$$

is the minimum among all spanning cycles of G .

Remark 5.4.2.

- (1) Unlike the shortest path problem, minimum spanning tree problem, chinese postman problem, there are currently no efficient algorithms for solving the traveling salesman problem.
- (2) Research questions relating to the traveling salesman problem are still being worked on by many mathematicians and computer scientists. Interested readers may visit <http://www.tsp.gatech.edu/>.
- (3) Some instances of the traveling salesman problem have the weighted complete graphs G satisfying the triangle inequality

$$w(xy) \leq w(xz) + w(zy)$$

for any vertices x, y, z in G .

Algorithms for finding Hamiltonian Cycle with least possible cost

Brute Force Algorithm

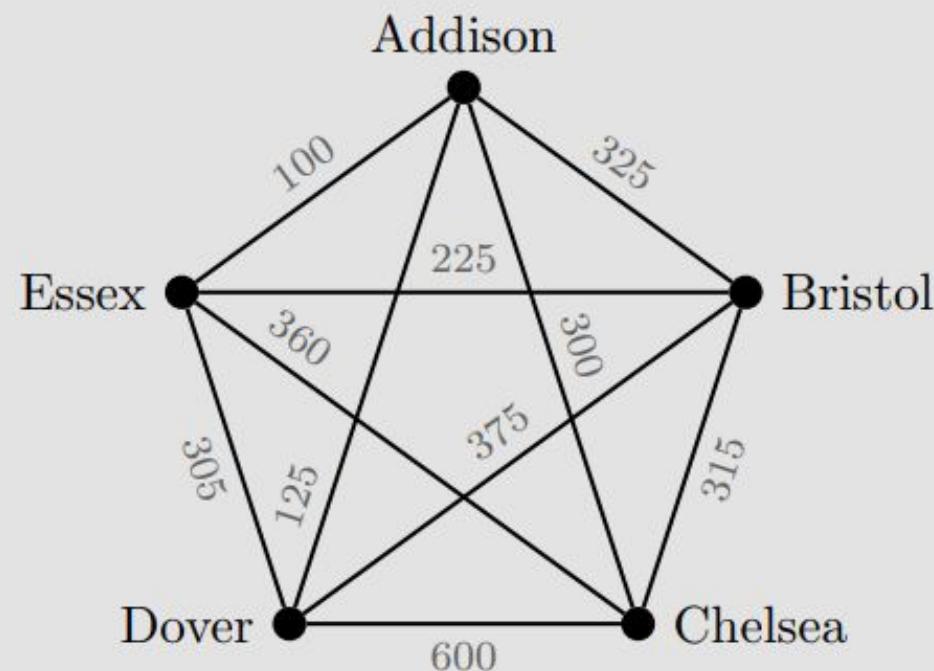
Input: Weighted complete graph K_n .

Steps:

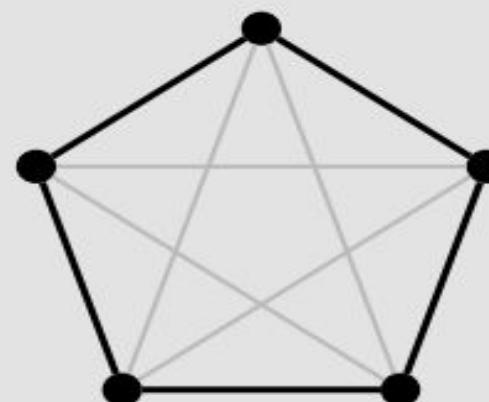
1. Choose a starting vertex, call it v .
2. Find all hamiltonian cycles starting at v . Calculate the total weight of each cycle.
3. Compare all $(n-1)!$ cycles. Pick one with the least total weight. (Note: there should be at least two options).

Output: Minimum hamiltonian cycle.

Example 2.11 Sam is planning his next business trip from his hometown of Addison and has determined the cost for travel between any of the five cities he must visit. This information is modeled in the weighted complete graph on the next page, where the weight is given in terms of dollars. Use Brute Force to find all possible routes for his trip.

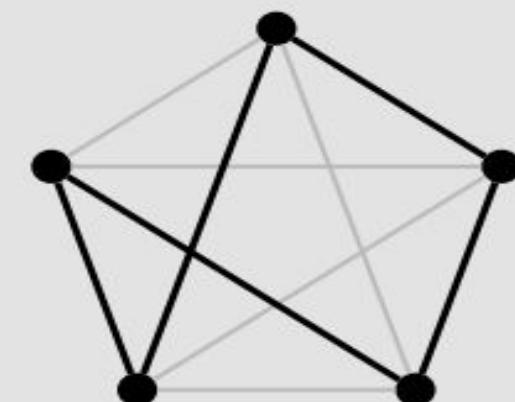


Solution: One method for finding all hamiltonian cycles, and ensuring you indeed have all 24, is to use alphabetical or lexicographic ordering of the cycles. Note that all cycles must start and end at Addison and we will abbreviate all cities with their first letter. For example, the first cycle is $a b c d e a$ and appears first in the list below. Below each cycle is its reversal and total weight.



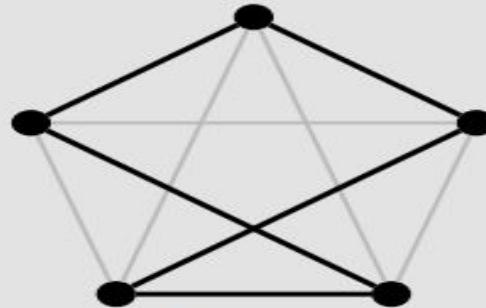
$a b c d e a$
 $a e d c b a$

1645

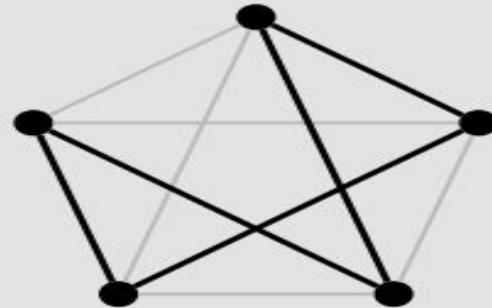


$a b c e d a$
 $a d e c b a$

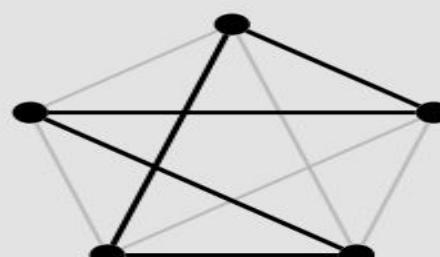
1430



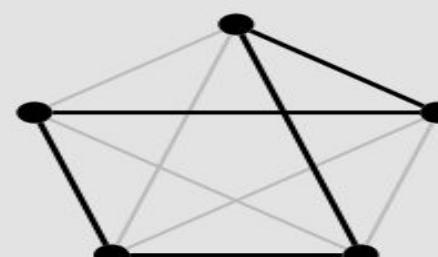
abdcea
aecdबa
1760



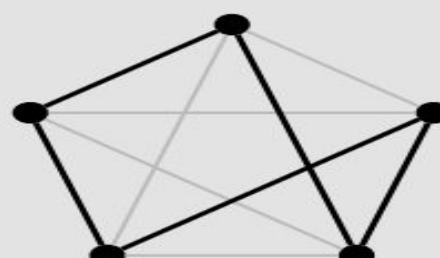
abdeca
acedba
1665



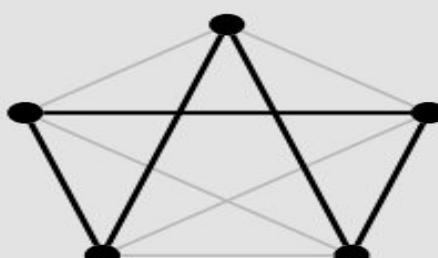
abecda
adceba
1635



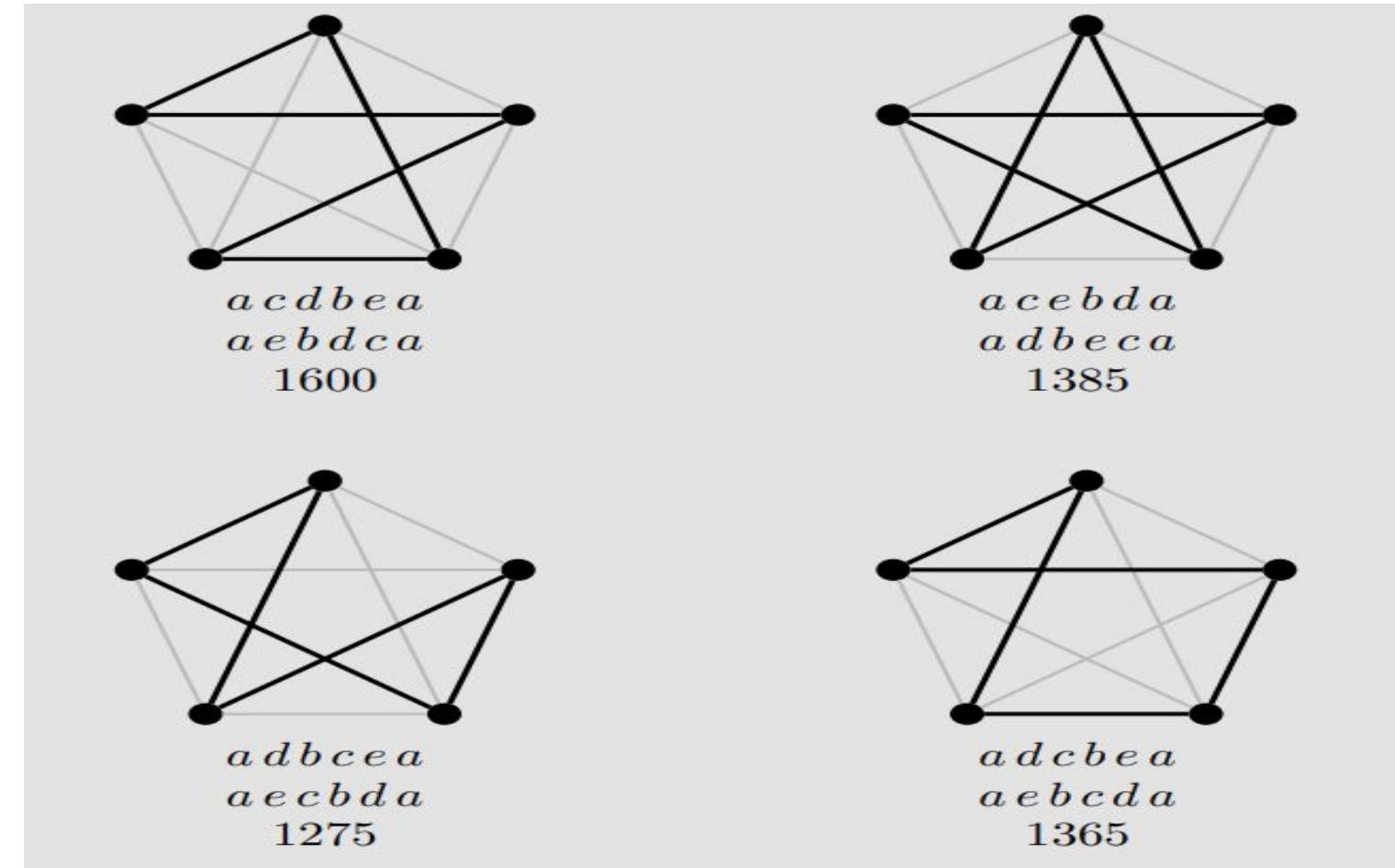
abedca
acdeba
1755



acbdea
aedbca
1395



acbeda
adebca
1270



From the data above we can identify the optimal cycle as *acbeda*, which provides Sam with the optimal route of Addison to Chelsea to Bristol to Essex to Dover and back to Addison, for a total at a cost of \$1270.

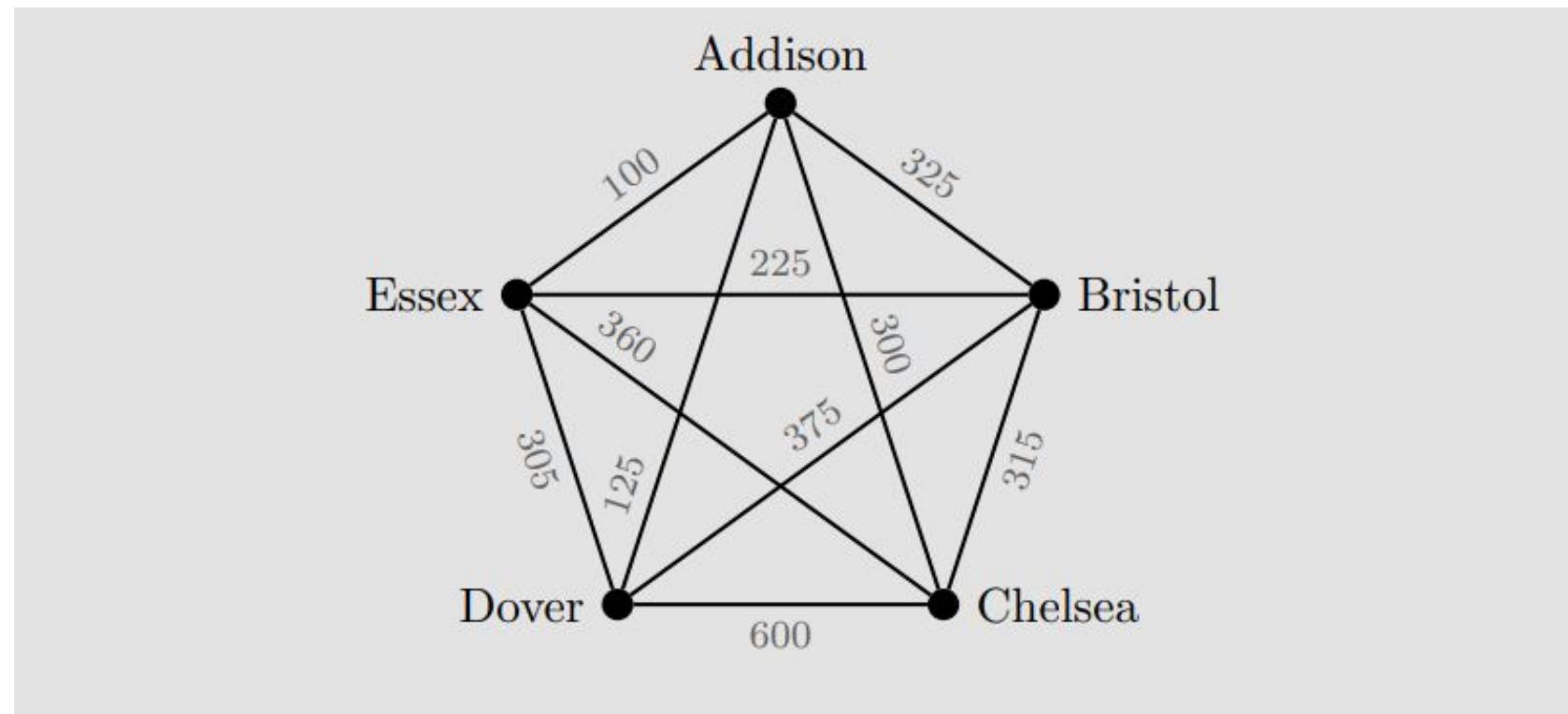
Nearest Neighbor Algorithm

Input: Weighted complete graph K_n .

Steps:

1. Choose a starting vertex, call it v . Highlight v .
2. Among all edges incident to v , pick the one with the smallest weight. If two possible choices have the same weight, you may randomly pick one.
3. Highlight the edge and move to its other endpoint u . Highlight u .
4. Repeat Steps (2) and (3), where only edges to unhighlighted vertices are considered.
5. Close the cycle by adding the edge to v from the last vertex highlighted. Calculate the total weight.

Output: hamiltonian cycle.

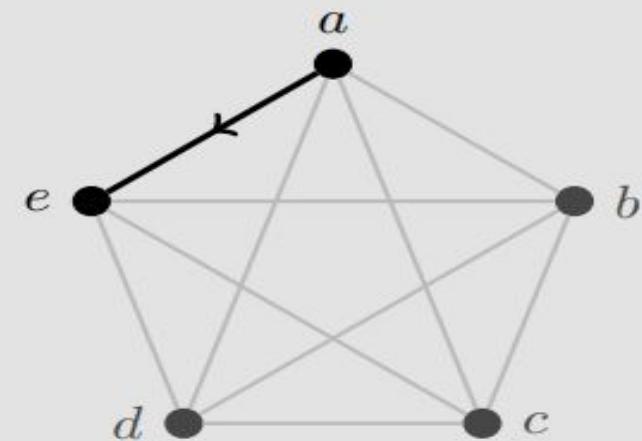
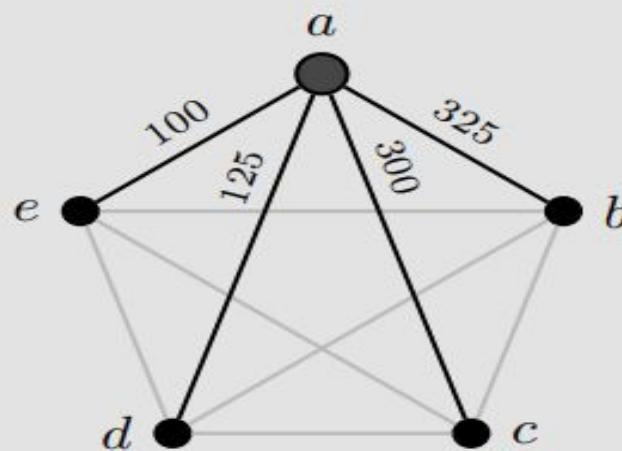


Example 2.12 Apply the Nearest Neighbor Algorithm to the graph from Example 2.11.

Solution: At each step we will show two copies of the graph. One will indicate the edges under consideration and the other traces the route under construction.

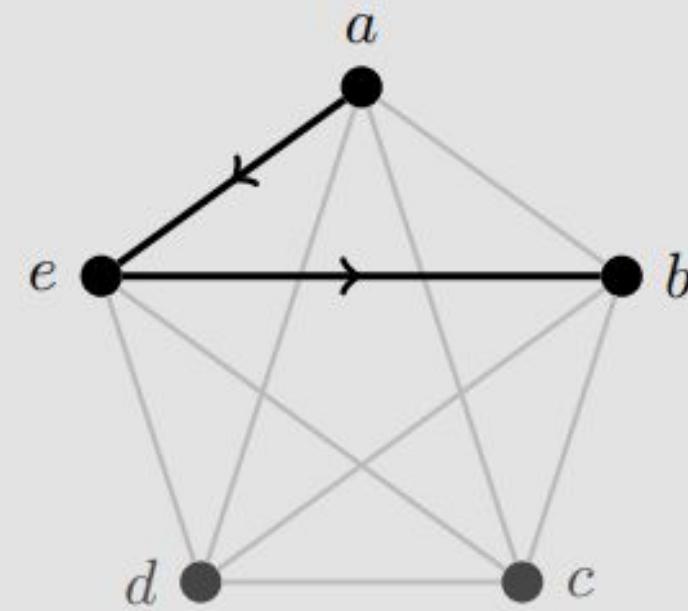
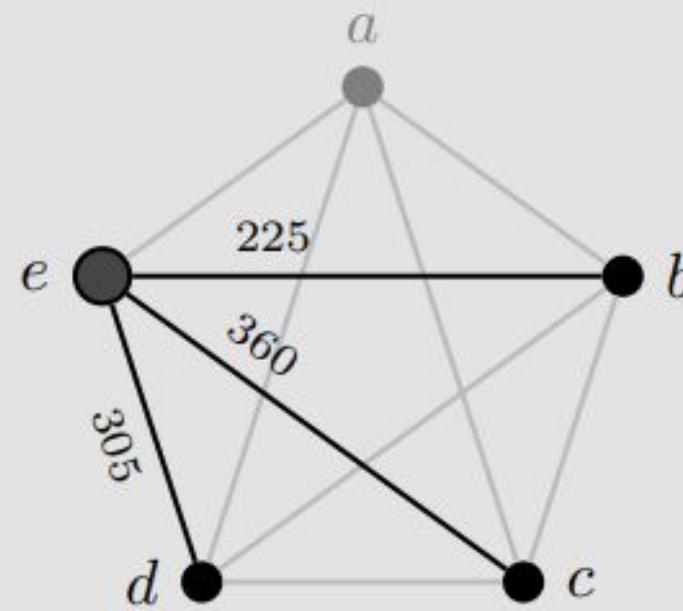
Step 1: The starting vertex is a .

Step 2: The edge of smallest weight incident to a is ae with weight 100.

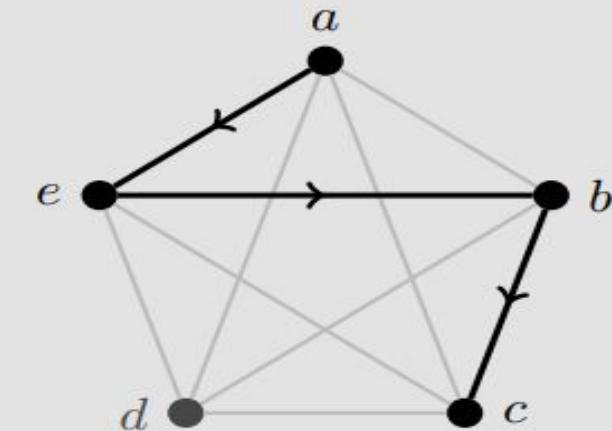
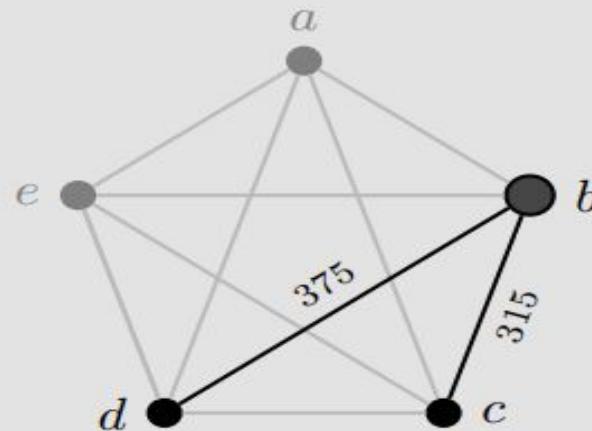


Step 3: From e we only consider edges to b, c , or d . Choose edge eb with weight 225.

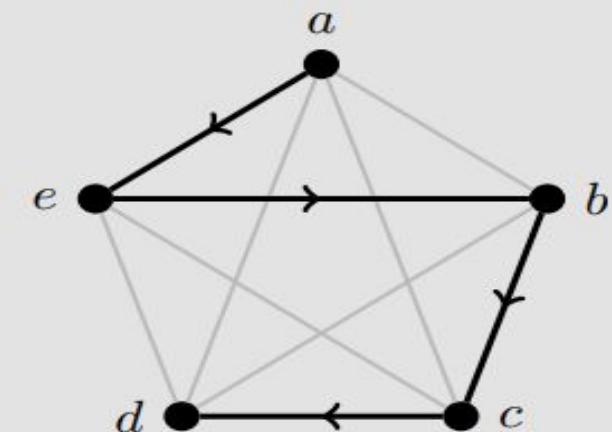
Step 3: From e we only consider edges to b, c , or d . Choose edge eb with weight 225.



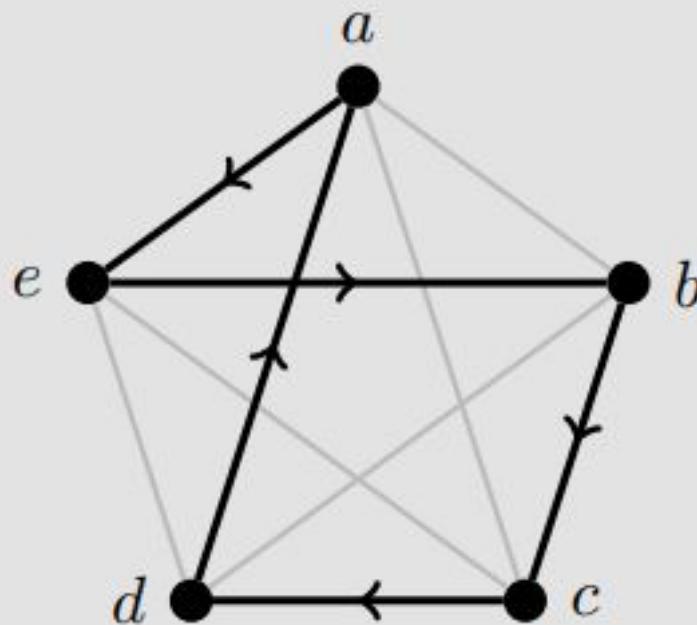
Step 4: From b we consider the edges to c or d . The edge of smallest weight is bc with weight 315.



Step 5: Even though cd does not have the smallest weight among all edges incident to c , it is the only choice available.



Step 6: Close the circuit by adding da .



Output: The circuit is $aebcda$ with total weight 1365.

In the previous example, the final circuit was Addison to Essex to Bristol to Chelsea to Dover and back to Addison. Although Nearest Neighbor did not find the optimal circuit of total cost \$1270, it did produce a fairly good circuit with total cost \$1365. Perhaps most important was the speed with which Nearest Neighbor found this circuit.

Repetitive Nearest Neighbor Algorithm

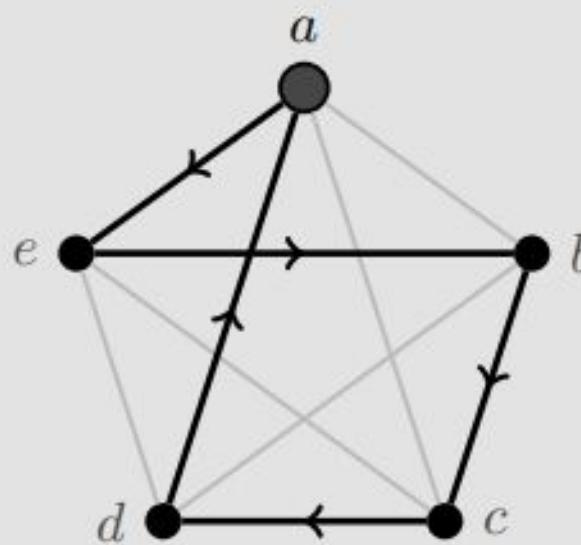
Input: Weighted complete graph K_n .

Steps:

1. Choose a starting vertex, call it v .
2. Apply the Nearest Neighbor Algorithm.
3. Repeat Steps (1) and (2) so each vertex of K_n serves as the starting vertex.
4. Choose the cycle of least total weight. Rewrite it with the desired reference point.

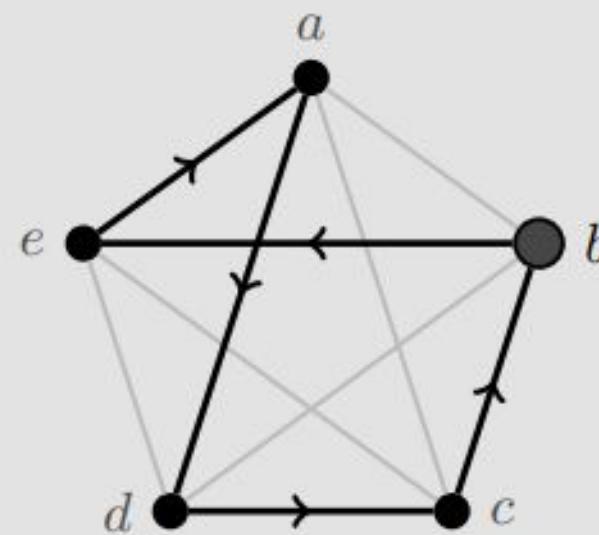
Output: hamiltonian cycle.

Solution: The five cycles are shown below, with the original name, the rewritten form with a as the reference point, and the total weight of the cycle. You should notice that the cycle starting at d is the same as the one starting at a , and the cycle starting at b is their reversal.



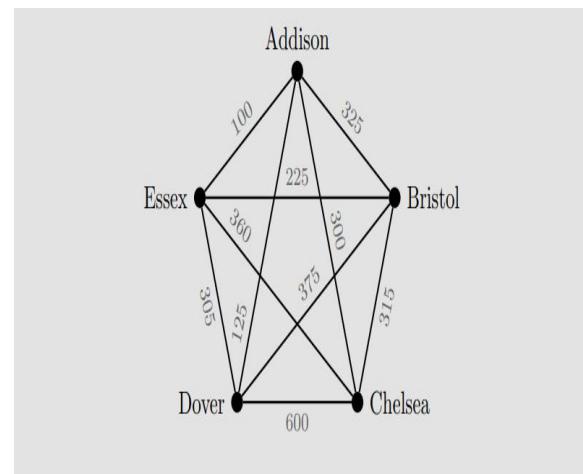
$aebcda$
 $aebcda$

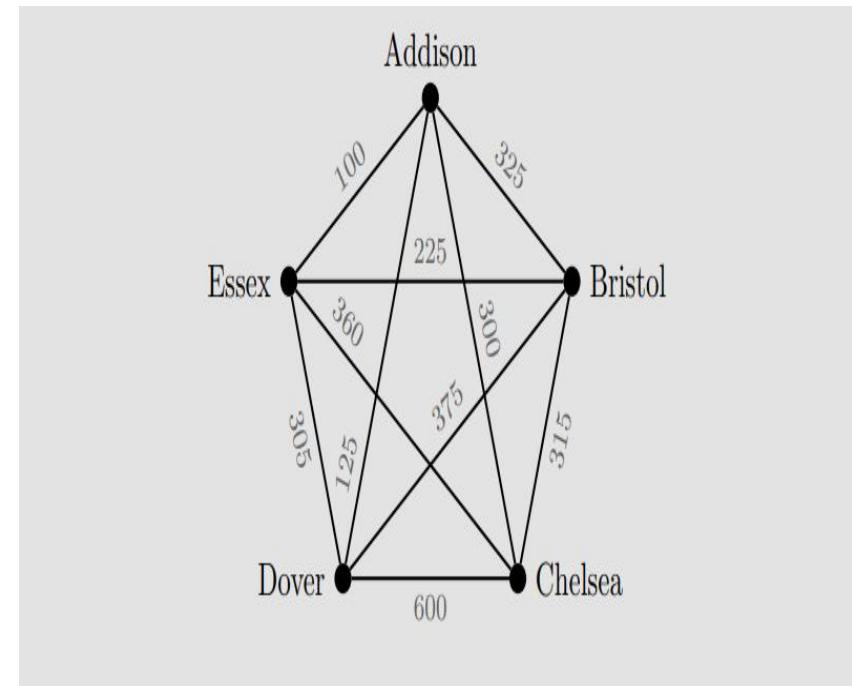
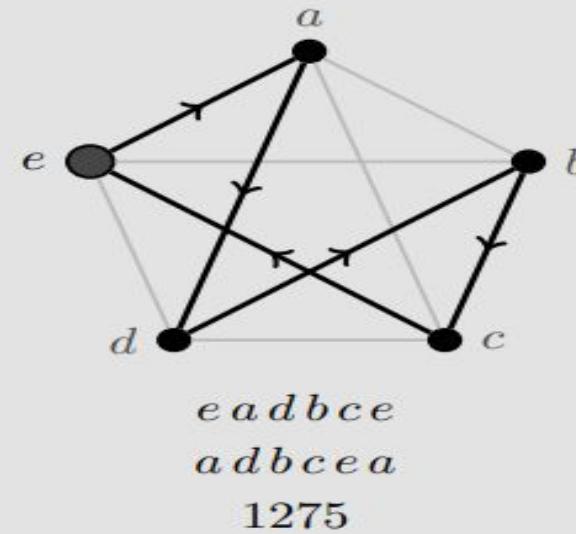
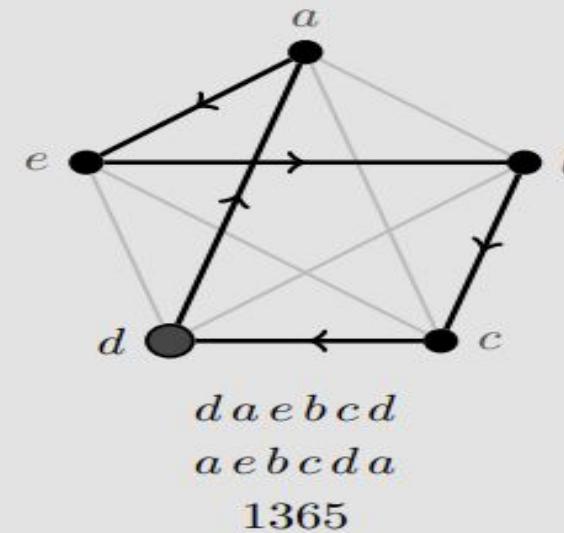
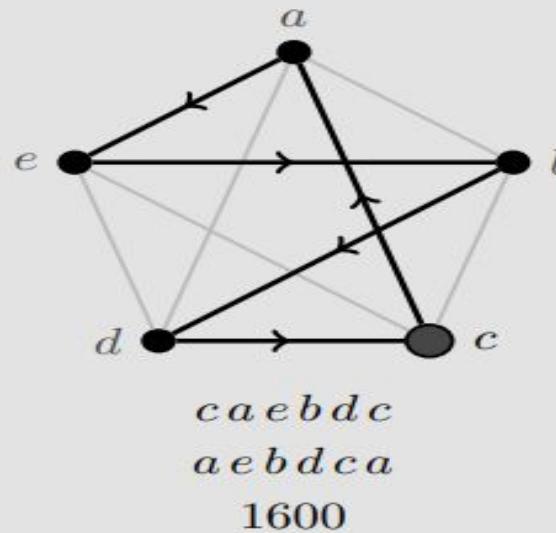
1365



$beadcb$
 $adcbea$

1365





It should come as no surprise that Repetitive Nearest Neighbor performs better than Nearest Neighbor; however, there is no guarantee that this improvement will produce the optimal cycle

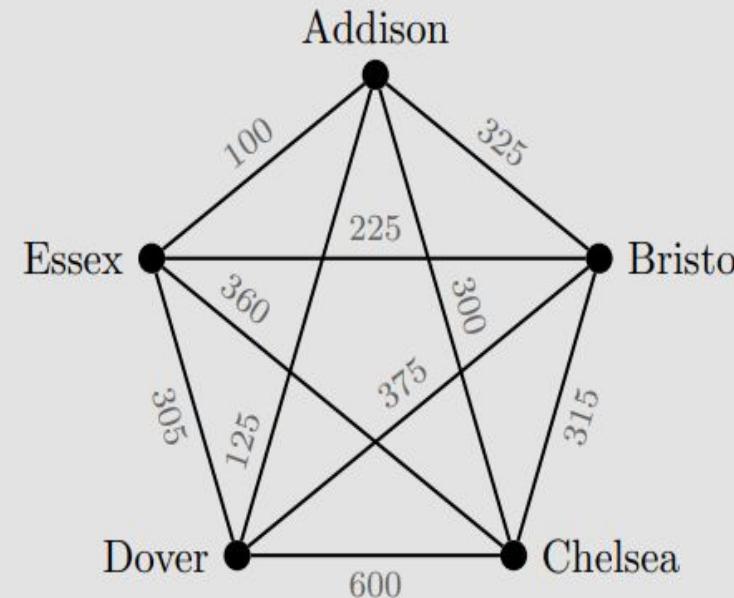
Cheapest Link Algorithm

Input: Weighted complete graph K_n .

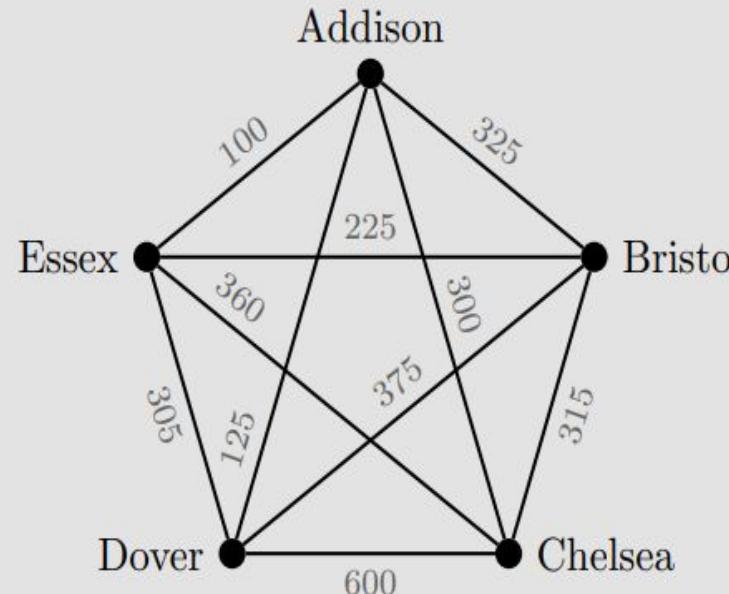
Steps:

1. Among all edges in the graph pick the one with the smallest weight. If two possible choices have the same weight, you may randomly pick one. Highlight the edge.
2. Repeat Step (1) with the added conditions:
 - (a) no vertex has three highlighted edges incident to it; and
 - (b) no edge is chosen so that a cycle closes before hitting all the vertices.
3. Calculate the total weight.

Output: hamiltonian cycle.



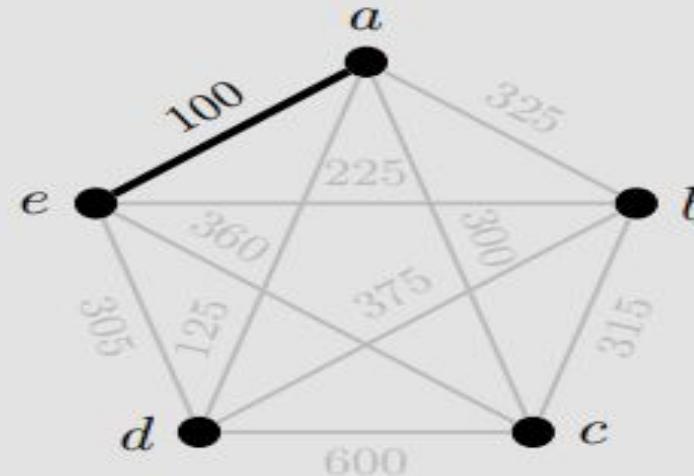
Example 2.14 Apply the Cheapest Link Algorithm to the graph from Example 2.11.



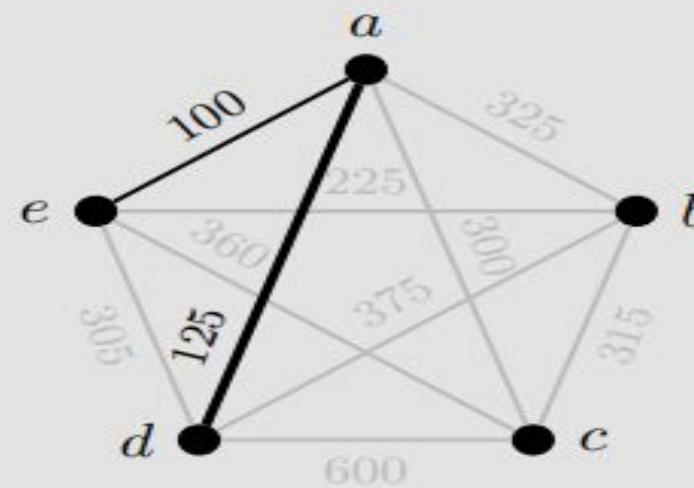
Example 2.14 Apply the Cheapest Link Algorithm to the graph from Example 2.11.

Solution: In each step shown, unchosen edges are shown in gray, previously chosen edges are in black, and the newly chosen edge in bold. An edge that is skipped will be marked with an X.

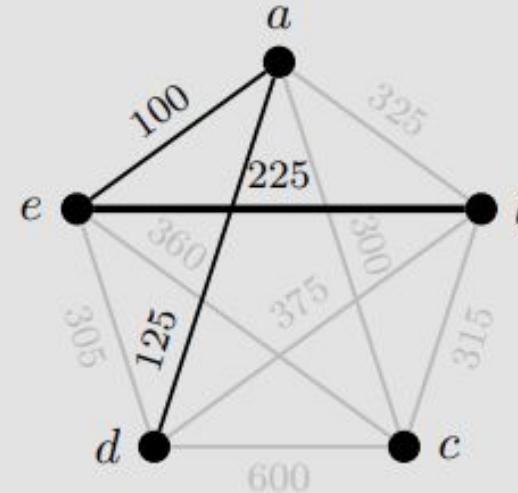
Step 1: The smallest weight is 100 for edge ae .



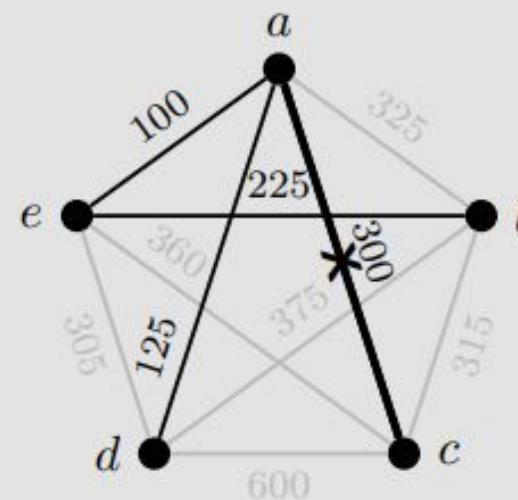
Step 2: The next smallest weight is 125 with edge ad .



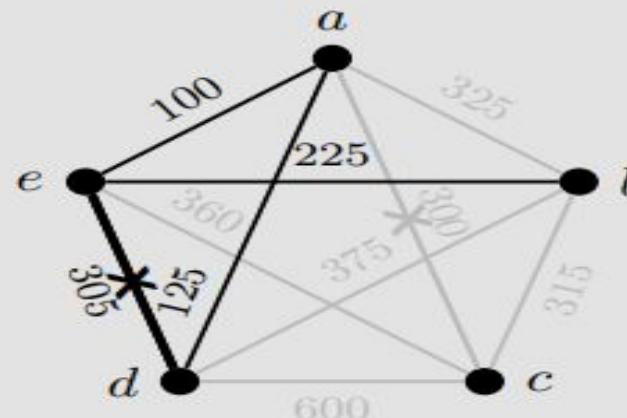
Step 3: The next smallest weight is 225 for edge be .



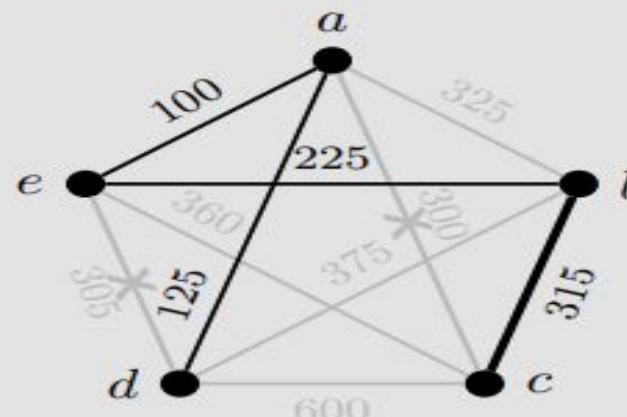
Step 4: Even though ac has weight 300, we must bypass it as it would force a to have three incident edges that are highlighted.



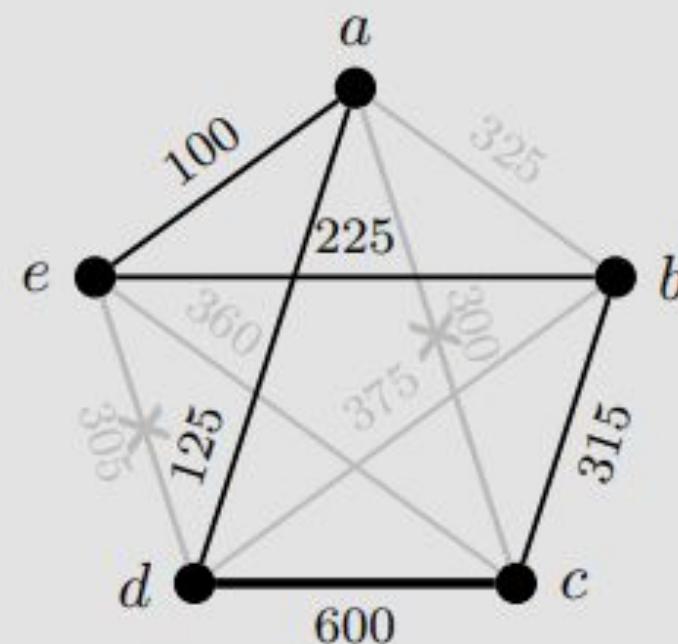
Step 5: The next smallest weight is 305 for edge ed , but again we must bypass it as it would close a cycle too early as well as force e to have three incident edges that are highlighted.



Step 6: The next available is bc with weight 315.



Step 7: At this point, we must close the cycle with the only one choice of cd .



Output: The resulting cycle is $a \rightarrow e \rightarrow b \rightarrow c \rightarrow d \rightarrow a$ with total weight 1365.

Nearest Insertion Algorithm

Input: Weighted complete graph K_n .

Steps:

1. Among all edges in the graph, pick the one with the smallest weight. If two possible choices have the same weight, you may randomly pick one. Highlight the edge and its endpoints.
2. Pick a vertex that is closest to one of the two already chosen vertices. Highlight the new vertex and its edges to both of the previously chosen vertices.
3. Pick a vertex that is closest to one of the three already chosen vertices. Calculate the increase in weight obtained by adding two new edges and deleting a previously chosen edge. Choose the scenario with the smallest total. For example, if the cycle obtained from (2) was $a - b - c - a$ and d is the new vertex that is closest to c , we calculate:

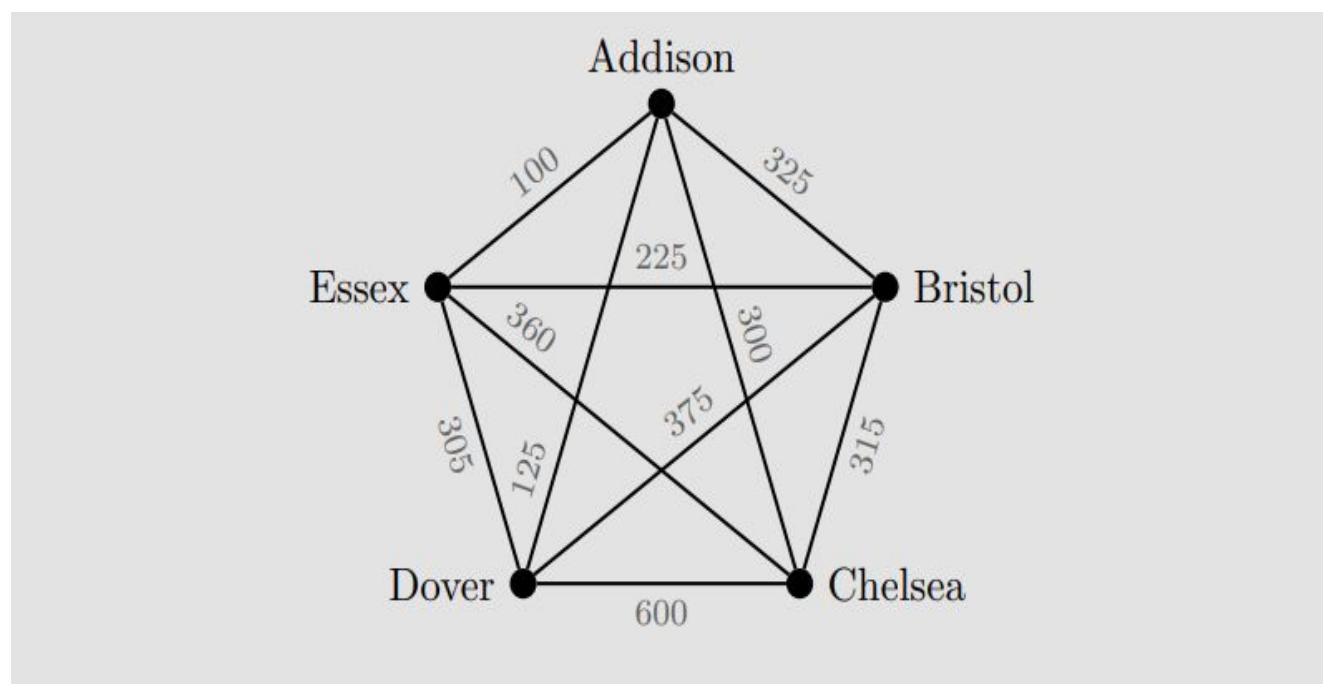
$$w(dc) + w(db) - w(cb) \text{ and } w(dc) + w(da) - w(ca)$$

and choose the option that produces the smaller total.

4. Repeat Step (3) until all vertices have been included in the cycle.
5. Calculate the total weight.

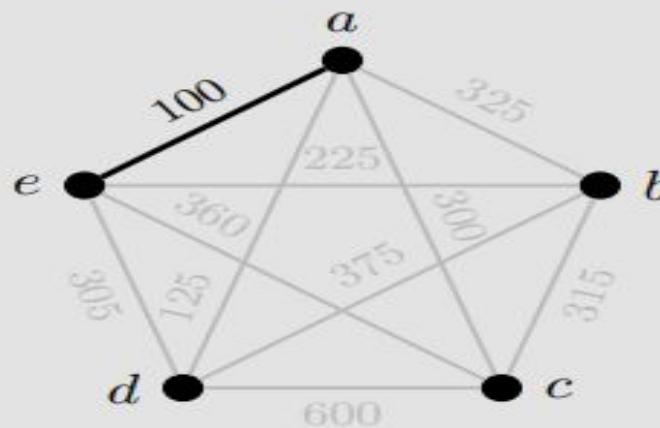
Output: hamiltonian cycle.

Example 2.15 Apply the Nearest Insertion Algorithm to the graph from Example 2.11.

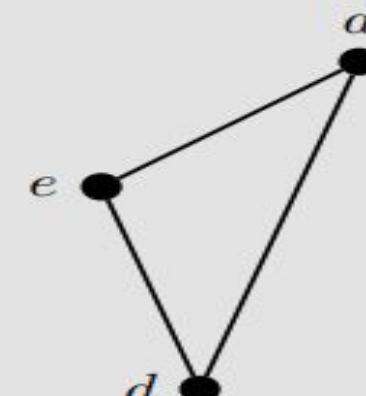
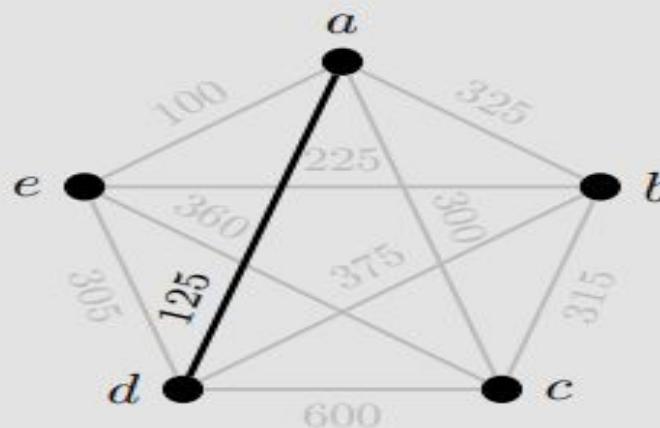


Solution: At each step shown, the graph on the left highlights the edge being added and the graph on the right shows how the cycle is built.

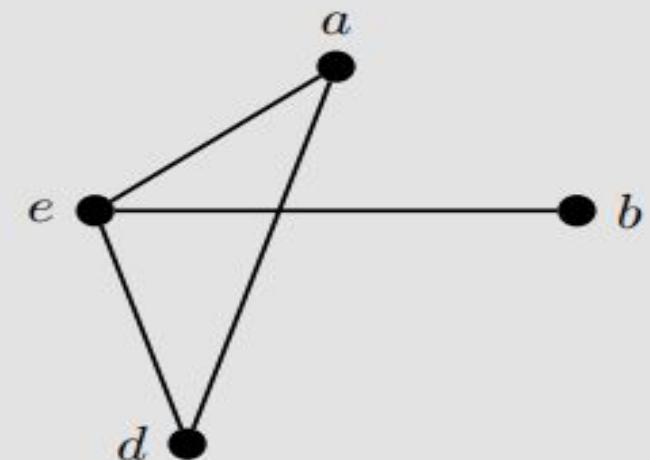
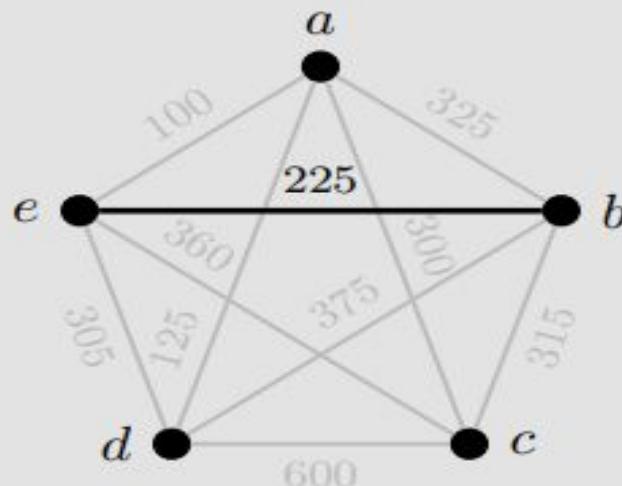
Step 1: The smallest weight edge is ae at 100.



Step 2: The closest vertex to either a or e is d through the edge ad of weight 125. Form a cycle by adding ad and de .



Step 3: The closest vertex to any of a, d , or e is b through the edge be with weight 225.

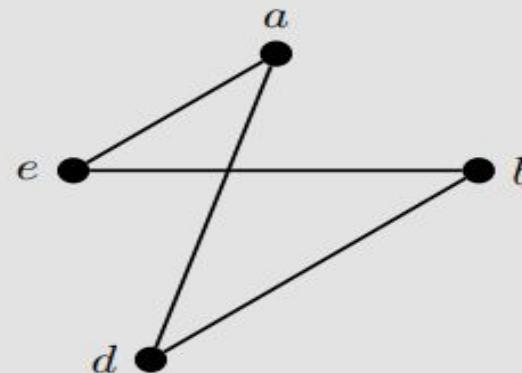


In adding edge be , either ae or de must be removed so that only two edges are incident to e . To determine which is the better choice, compute the following expressions:

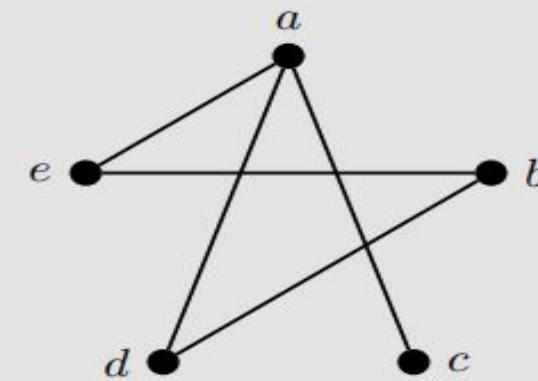
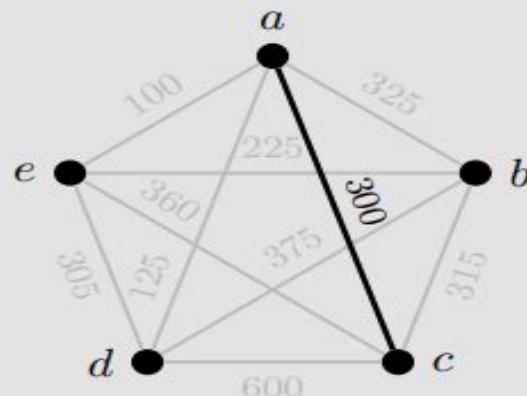
$$be + ba - ea = 225 + 325 - 100 = 450$$

$$be + bd - ed = 225 + 375 - 305 = 295$$

Since the second total is smaller, we create a larger cycle by adding edge bd and removing ed .



Step 4: The only vertex remaining is c , and the minimum edge to the other vertices is ac with weight 300.

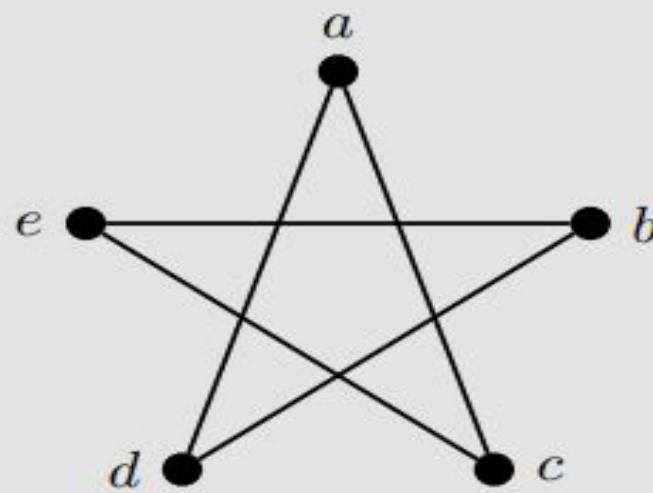


Either ae or ad must be removed. As in the previous step, we compute the following expressions:

$$ca + cd - ad = 300 + 600 - 125 = 775$$

$$ca + ce - ae = 300 + 360 - 100 = 560$$

The second total is again smaller, so we add ce and remove ae .



Output: The cycle is $acebda$ with total weight 1385.

In the example above, Nearest Insertion performed slightly worse than Cheapest Link and Repetitive Nearest Neighbor. Among all three algorithms, Repetitive Nearest Neighbor found the cycle closest to optimal. In general, when comparing algorithm performance we focus less on absolute error ($1275 - 1270 = 5$) but rather on relative error.

Definition 2.19 The *relative error* for a solution is given by

$$\epsilon_r = \frac{\text{Solution} - \text{Optimal}}{\text{Optimal}}$$

Example 2.16 Find the relative error for each of the algorithms performed on the graph from Example 2.11.

Solution:

- Repetitive Nearest Neighbor: $\epsilon_r = \frac{1275 - 1270}{1270} = 0.003937 \approx 0.39\%$
- Cheapest Link: $\epsilon_r = \frac{1365 - 1270}{1270} = 0.074803 \approx 7.48\%$
- Nearest Insertion: $\epsilon_r = \frac{1385 - 1270}{1270} = 0.090551 \approx 9.05\%$

Shortest Path

Dijkstra's Algorithm

Input: Weighted connected simple graph $G = (V, E, w)$ and designated *Start* vertex.

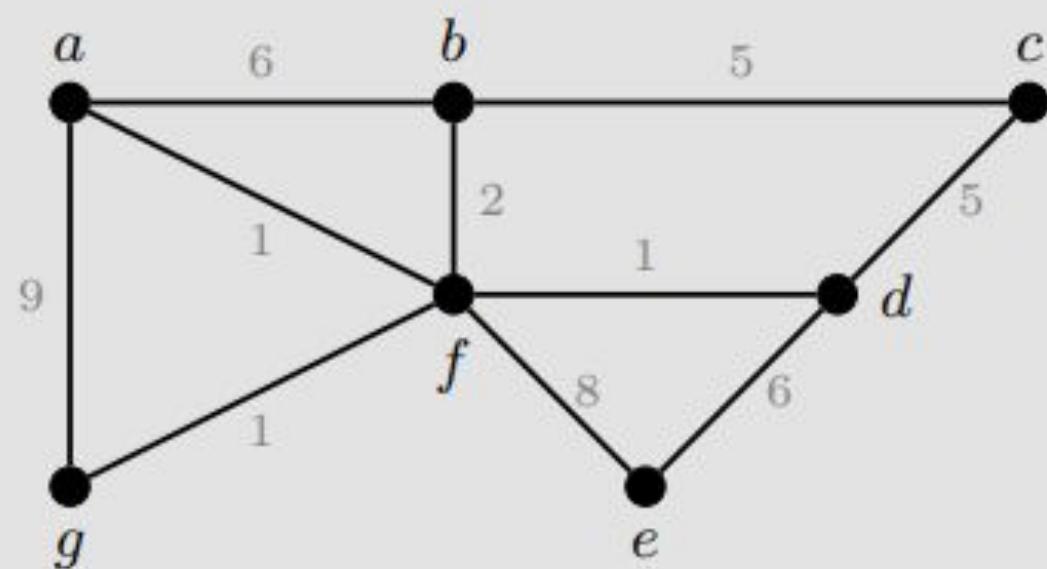
Steps:

1. For each vertex x of G , assign a label $L(x)$ so that $L(x) = (-, 0)$ if $x = \text{Start}$ and $L(x) = (-, \infty)$ otherwise. Highlight *Start*.
2. Let $u = \text{Start}$ and define F to be the neighbors of u . Update the labels for each vertex v in F as follows:

if $w(u) + w(uv) < w(v)$, then redefine $L(v) = (u, w(u) + w(uv))$
otherwise do not change $L(v)$
3. Highlight the vertex with lowest weight as well as the edge uv used to update the label. Redefine $u = v$.
4. Repeat Steps (2) and (3) until each vertex has been reached. In all future iterations, F consists of the un-highlighted neighbors of all previously highlighted vertices and the labels are updated only for those vertices that are adjacent to the last vertex that was highlighted.
5. The shortest path from *Start* to any other vertex is found by tracing back using the first component of the labels. The total weight of the path is the weight given in the second component of the ending vertex.

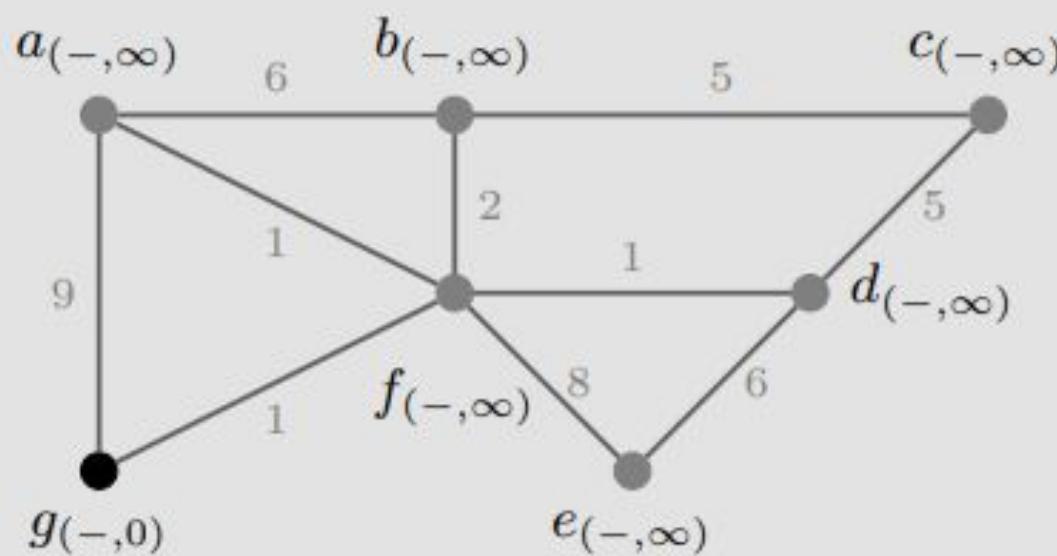
Output: Highlighted path from *Start* to any vertex x of weight $w(x)$.

Example 2.17 Apply Dijkstra's Algorithm to the graph below where $Start = g$.



Solution: In each step, the label of a vertex will be shown in the table on the right.

Step 1: Highlight g . Define $L(g) = (-, 0)$ and $L(x) = (-, \infty)$ for all $x = a, \dots, f$.



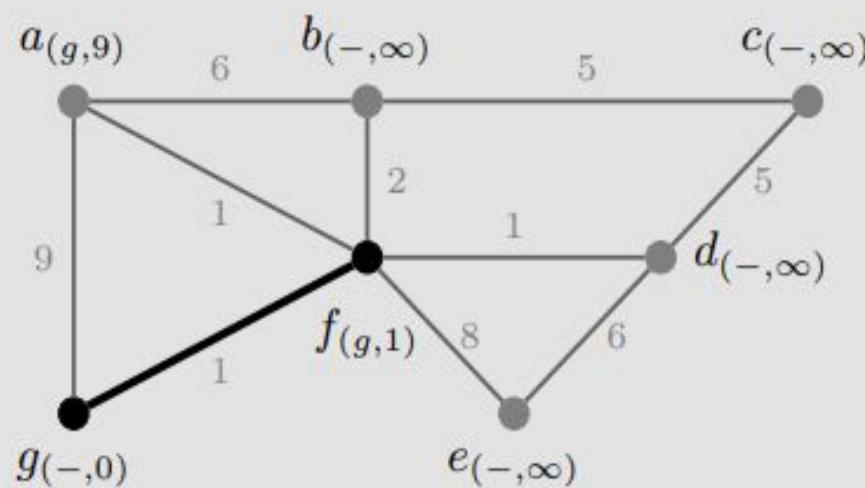
$F = \{\}$	
a	$(-, \infty)$
b	$(-, \infty)$
c	$(-, \infty)$
d	$(-, \infty)$
e	$(-, \infty)$
f	$(-, \infty)$
g	$(-, 0)$

Step 2: Let $u = g$. Then the neighbors of g comprise $F = \{a, f\}$. We compute

$$w(g) + w(ga) = 0 + 9 = 9 < \infty = w(a)$$

$$w(g) + w(gf) = 0 + 1 = 1 < \infty = w(f)$$

Update $L(a) = (g, 9)$ and $L(f) = (g, 1)$. Since the minimum weight for all vertices in F is that of f , we highlight the edge gf and the vertex f .



$F = \{a, f\}$	
a	$(-, \infty) \rightarrow (g, 9)$
b	$(-, \infty)$
c	$(-, \infty)$
d	$(-, \infty)$
e	$(-, \infty)$
f	$(-, \infty) \rightarrow (g, 1)$
g	$(-, 0)$

Step 3: Let $u = f$. Then the neighbors of all highlighted vertices are $F = \{a, b, d, e\}$. We compute

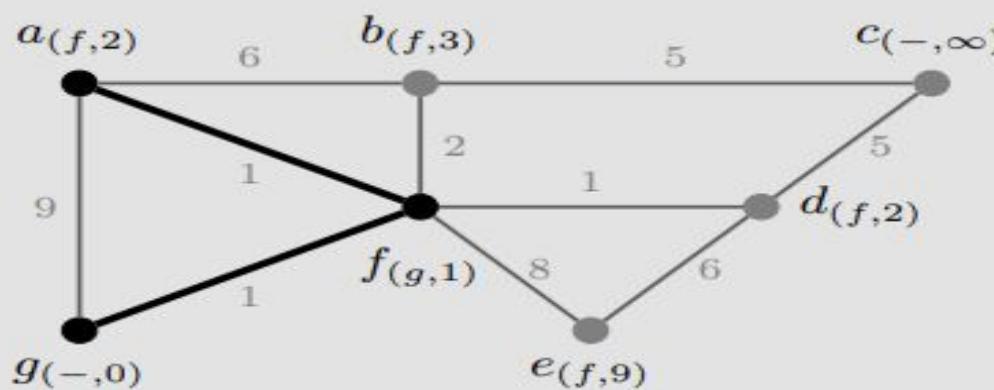
$$w(f) + w(fa) = 1 + 1 = 2 < 9 = w(a)$$

$$w(f) + w(fb) = 1 + 2 = 3 < \infty = w(b)$$

$$w(f) + w(fd) = 1 + 1 = 2 < \infty = w(d)$$

$$w(f) + w(fe) = 1 + 8 = 9 < \infty = w(e)$$

Update $L(a) = (f, 2)$, $L(b) = (f, 3)$, $L(d) = (f, 2)$ and $L(e) = (f, 9)$. Since the minimum weight for all vertices in F is that of a or d , we choose to highlight the edge fa and the vertex a .

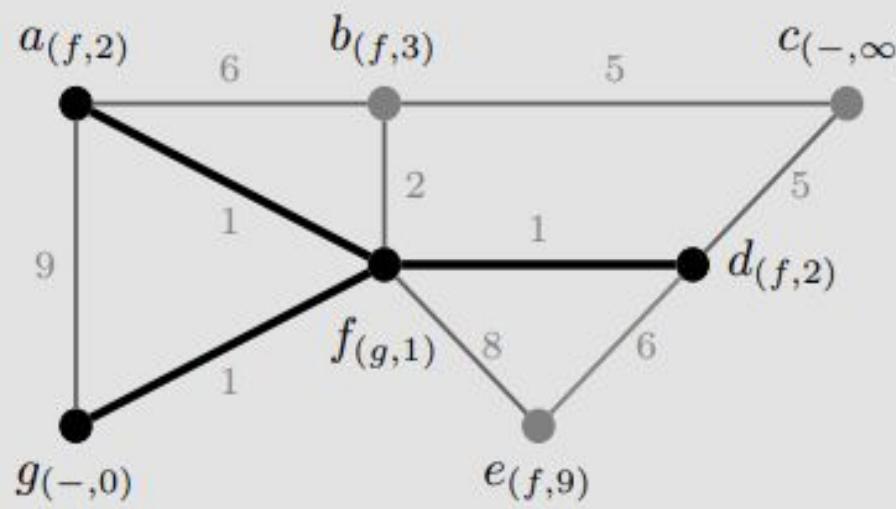


$F = \{a, b, d, e\}$	
a	$(g, 9) \rightarrow (f, 2)$
b	$(-, \infty) \rightarrow (f, 3)$
c	$(-, \infty)$
d	$(-, \infty) \rightarrow (f, 2)$
e	$(-, \infty) \rightarrow (f, 9)$
f	$(g, 1)$
g	$(-, 0)$

Step 4: Let $u = a$. Then the neighbors of all highlighted vertices are $F = \{b, d, e\}$. Note, we only consider updating the label for b since this is the only vertex adjacent to a , the vertex highlighted in the previous step.

$$w(a) + w(ba) = 2 + 6 = 8 \not< 2 = w(b)$$

We do not update the label for b since the computation above is not less than the current weight of b . The minimum weight for all vertices in F is that of d , and so we highlight the edge fd and the vertex d .



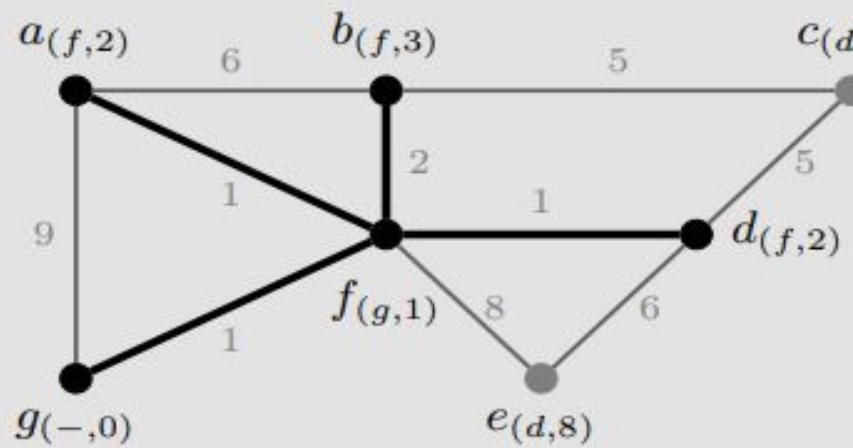
$F = \{b, d, e\}$
$a (f, 2)$
$b (f, 3)$
$c (-, \infty)$
$d (f, 2)$
$e (f, 9)$
$f (g, 1)$
$g (-, 0)$

Step 5: Let $u = d$. Then the neighbors of all highlighted vertices are $F = \{b, c, e\}$. We compute

$$w(d) + w(dc) = 2 + 5 = 7 < \infty = w(c)$$

$$w(d) + w(de) = 2 + 6 = 8 < 9 = w(e)$$

Update $L(c) = (d, 7)$ and $L(e) = (d, 8)$. Since the minimum weight for all vertices in F is that of b , we highlight the edge bf and the vertex b .

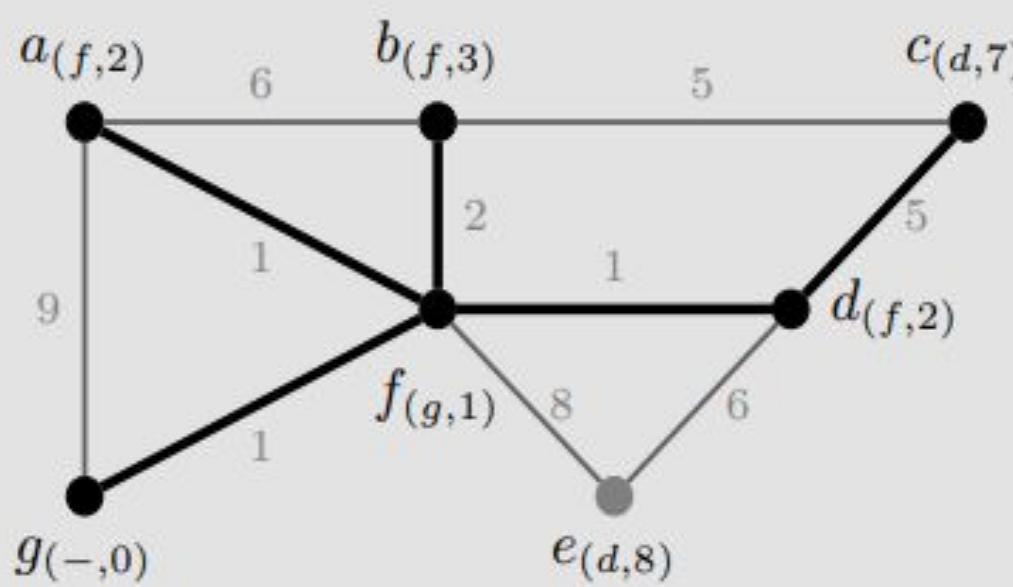


$F = \{b, c, e\}$	
a	$(f, 2)$
b	$(f, 3)$
c	$(-, \infty) \rightarrow (d, 7)$
d	$(f, 2)$
e	$(f, 9) \rightarrow (d, 8)$
f	$(g, 1)$
g	$(-, 0)$

Step 6: Let $u = b$. Then the neighbors of all highlighted vertices are $F = \{c, e\}$. However, we only consider updating the label of c since e is not adjacent to b . Since

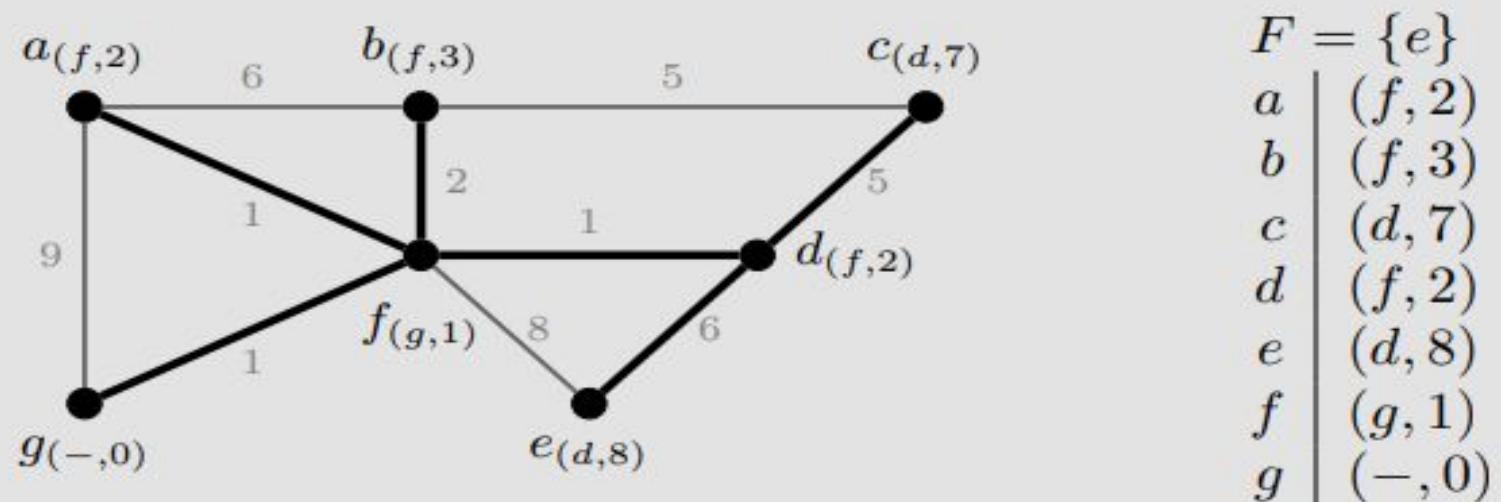
$$w(b) + w(bc) = 3 + 5 = 8 \not< 7 = w(c)$$

we do not update the labels of any vertices. Since the minimum weight for all vertices in F is that of c we highlight the edge dc and the vertex c . This terminates the iterations of the algorithm since our ending vertex has been reached.



$F = \{c, e\}$	
a	$(f, 2)$
b	$(f, 3)$
c	$(d, 7)$
d	$(f, 2)$
e	$(d, 8)$
f	$(g, 1)$
g	$(-, 0)$

Step 7: Let $u = c$. Then the neighbors of all highlighted vertices are $F = \{e\}$. However, we do not need to update any labels since c and e are not adjacent. Thus we highlight the edge de and the vertex e . This terminates the iterations of the algorithm since all vertices are now highlighted.



Output: The shortest paths from g to all other vertices can be found highlighted above. For example the shortest path from g to c is $g f d c$ and has a total weight 7, as shown by the label of c .

Lec # 13,14 & 15

Walk using Matrices

Use for the adjacency matrix is to count the number of walks between two vertices within a graph

Theorem 2.24 Let G be a graph with adjacency matrix A . Then for any integer $n > 0$ the entry a_{ij} in A^n counts the number of walks from v_i to v_j .

Example 1.4.3. Some walks in H of Fig. 1.4.2 and their respective lengths are shown in the table below.

	sequence	walk	length
(1)	$bf_3xf_4wf_5xf_4wf_{10}zf_{10}w$	$b - w$	6
(2)	$bf_3xf_4wf_5xf_7y$	$b - y$	4
(3)	$bf_3xf_4wf_{11}yf_9c$	$b - c$	4
(4)	$bf_2cf_8xf_4wf_5xf_3b$	$b - b$	5
(5)	$bf_2cf_9yf_{12}zf_6xf_3b$	$b - b$	5

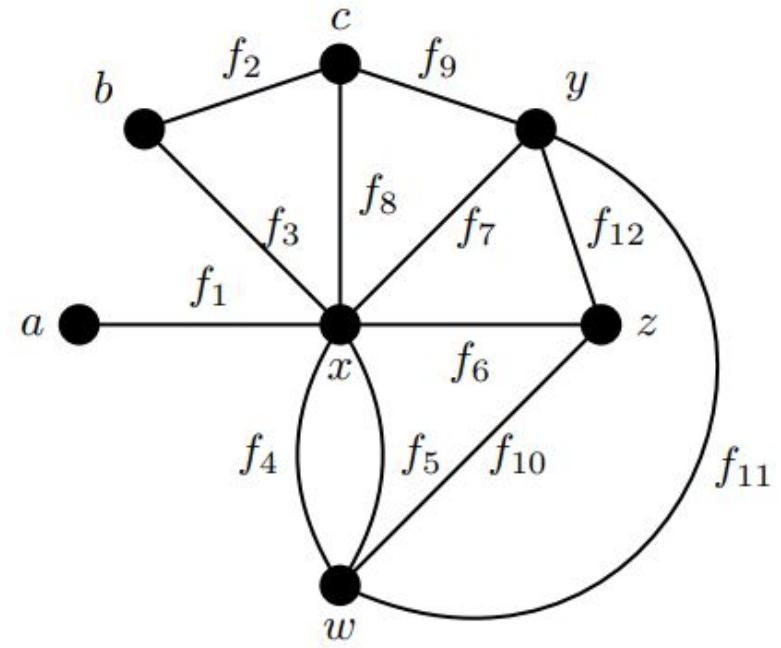
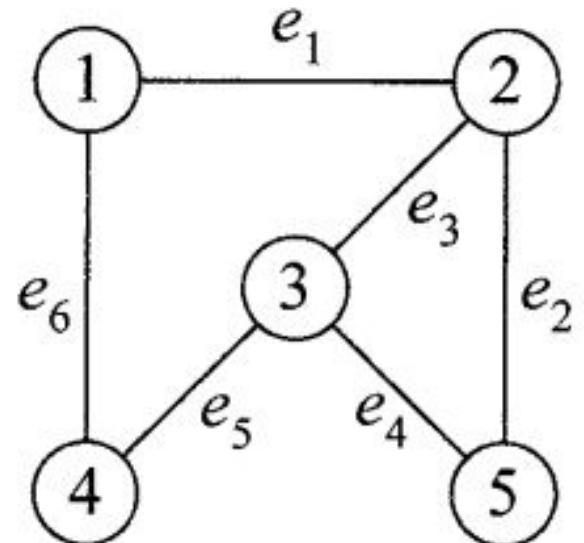
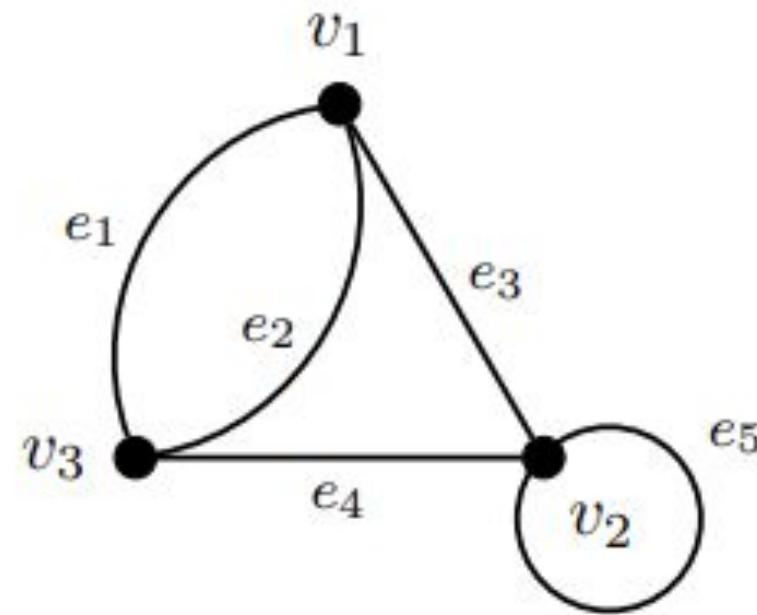


Fig. 1.4.2

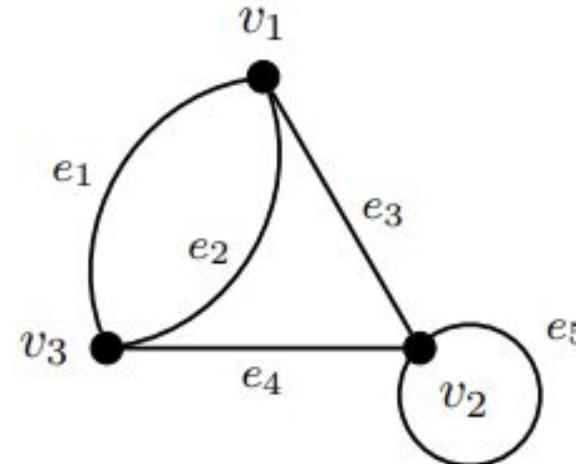
$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad A^2 = \begin{bmatrix} 2 & 0 & 2 & 0 & 1 \\ 0 & 3 & 1 & 2 & 1 \\ 2 & 1 & 3 & 0 & 1 \\ 0 & 2 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix}, \quad \text{and} \quad A^4 = \begin{bmatrix} 9 & 3 & 11 & 1 & 6 \\ 3 & 15 & 7 & 11 & 8 \\ 11 & 6 & 15 & 3 & 8 \\ 1 & 11 & 3 & 9 & 6 \\ 6 & 8 & 8 & 6 & 8 \end{bmatrix}$$

The degrees of the vertices 1, 2, 3, 4, 5 are 2, 3, 3, 2, 2, respectively, in agreement with the diagonal entries of A^2 . The (1, 5) entry in A^4 is 6, indicating that there are six *different* walks of length 4 between 1 and 5. These walks are $1 - 4 - 1 - 2 - 5$, $1 - 2 - 1 - 2 - 5$, $1 - 4 - 3 - 2 - 5$, $1 - 2 - 5 - 2 - 5$, $1 - 2 - 3 - 2 - 5$, and $1 - 2 - 5 - 3 - 5$.

**Fig. 2-2**



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



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

Now consider the walks from v_1 to v_2 . There is only one walk of length 1, and yet three of length 2:

$$v_1 \xrightarrow{e_3} v_2 \xrightarrow{e_5} v_2$$

$$v_1 \xrightarrow{e_1} v_3 \xrightarrow{e_4} v_2$$

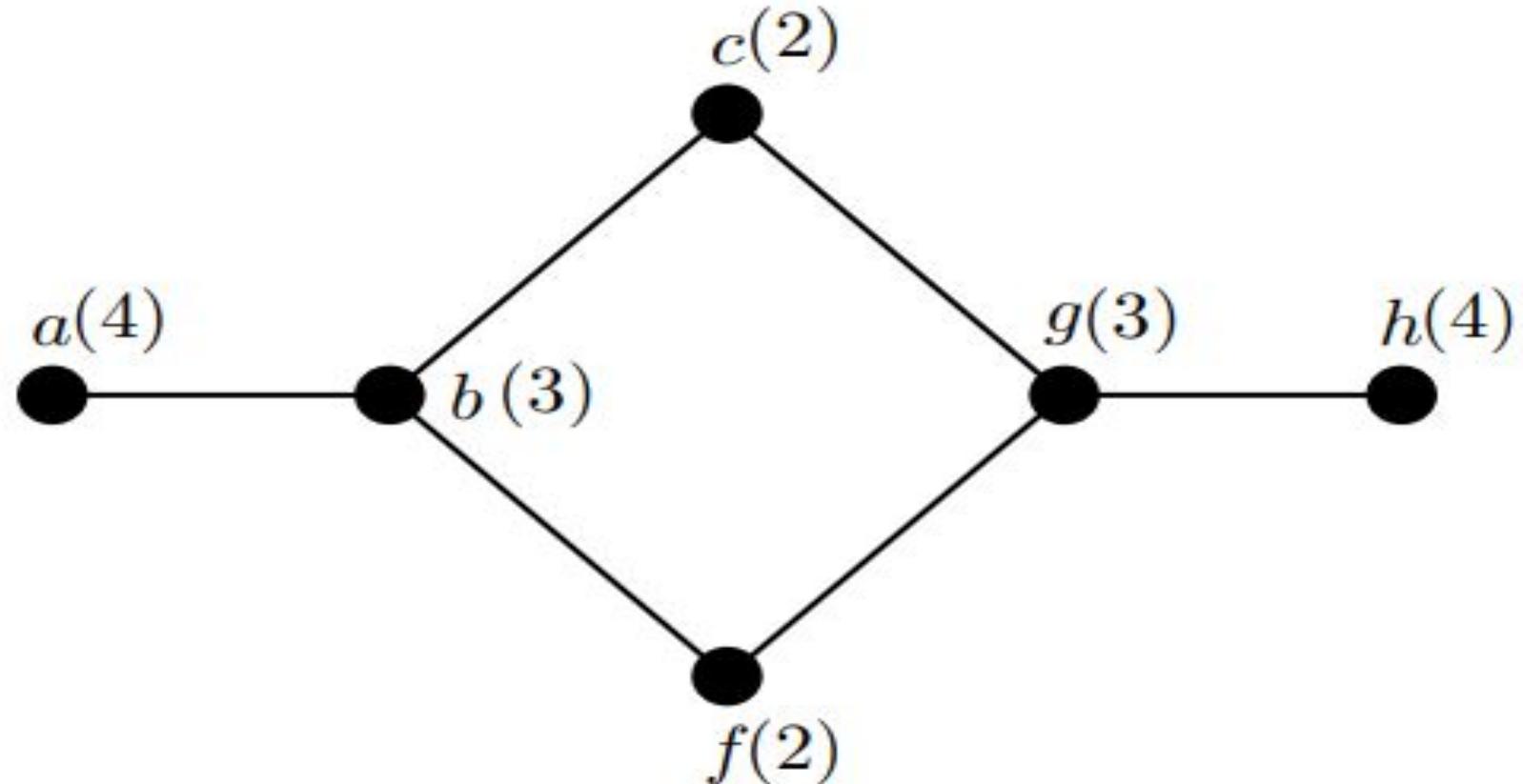
$$v_1 \xrightarrow{e_2} v_3 \xrightarrow{e_4} v_2$$

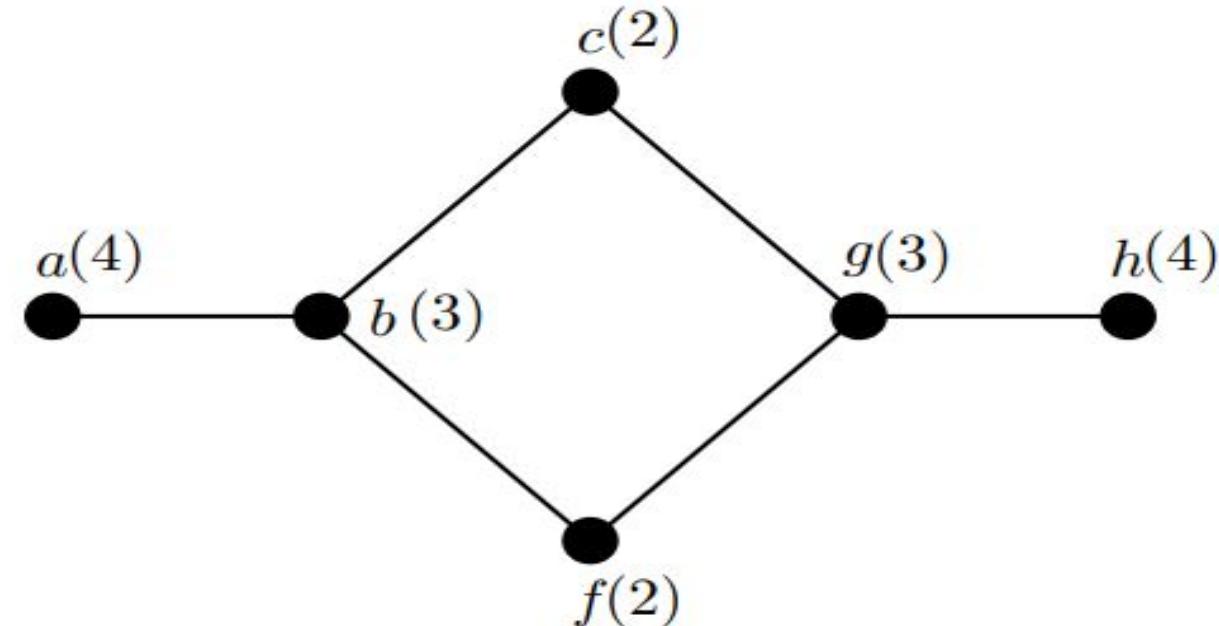
$$A^2 = \begin{bmatrix} 5 & 3 & 1 \\ 3 & 3 & 3 \\ 1 & 3 & 5 \end{bmatrix}$$

Definition 2.25 Given two vertices x, y in a graph G , define the *distance* $d(x, y)$ as the length of the shortest path from x to y . The *eccentricity* of a vertex x is the maximum distance from x to any other vertex in G ; that is $\epsilon(x) = \max_{y \in V(G)} d(x, y)$.

The *diameter* of G is the maximum eccentricity among all vertices, and so measures the maximum distance between any two vertices; that is $\text{diam}(G) = \max_{x, y \in V(G)} d(x, y)$. The *radius* of a graph is the minimum eccentricity among all vertices; that is $\text{rad}(G) = \min_{x \in V(G)} \epsilon(x)$.

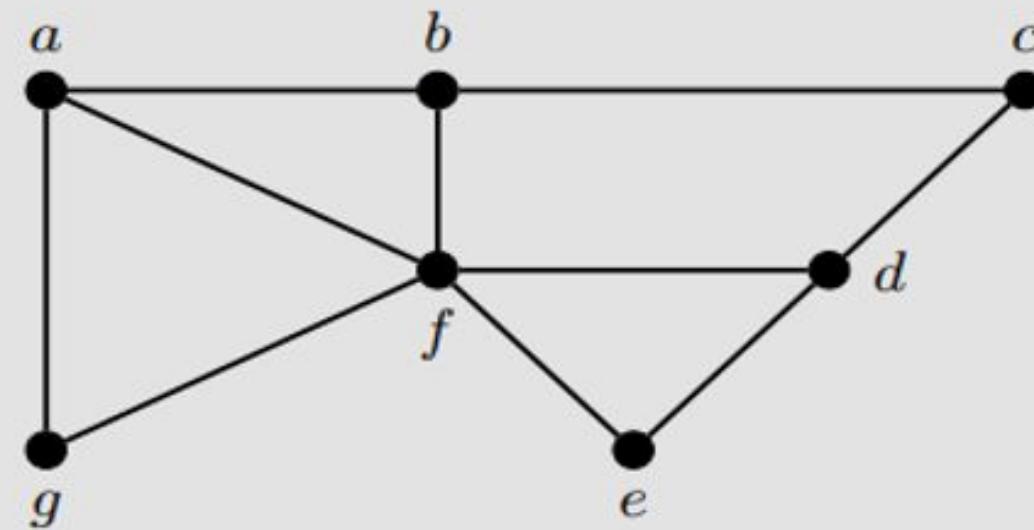
Radius & Diameter?





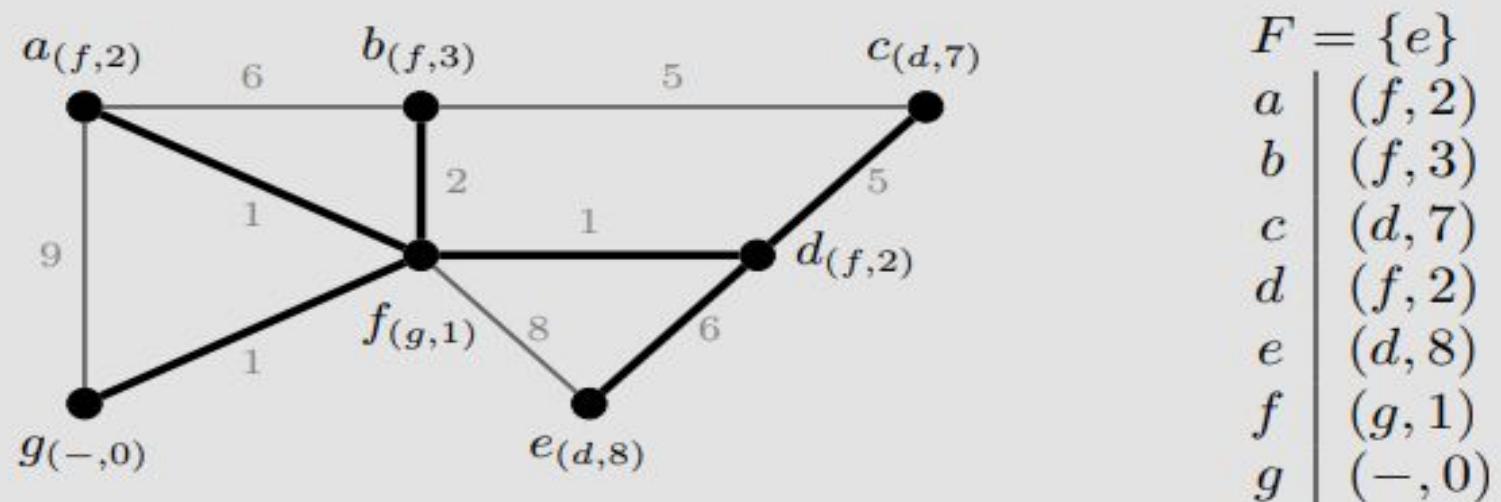
Among the six eccentricities shown in the figure, we notice that ‘2’ is the smallest while ‘4’ is the largest. In this situation, we say that the **radius** of G is 2 and the **diameter** of G is 4. Note also that there are two vertices in G , namely c and f , with least eccentricity (i.e., $e(c) = e(f) = 2$). Each of them is called a **central** vertex, and the set of these two central vertices is called the **center** of G .

Example 2.18 Find the diameter and radius for the graph below.



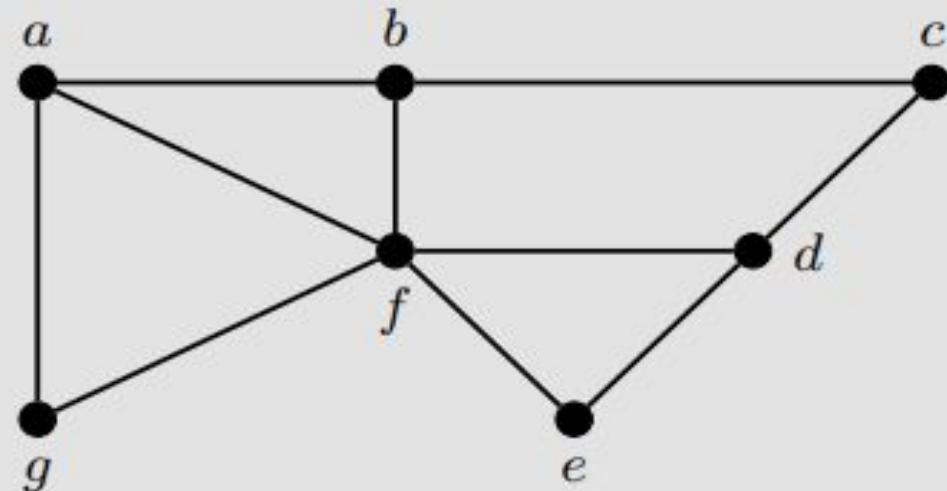
Remember!

Step 7: Let $u = c$. Then the neighbors of all highlighted vertices are $F = \{e\}$. However, we do not need to update any labels since c and e are not adjacent. Thus we highlight the edge de and the vertex e . This terminates the iterations of the algorithm since all vertices are now highlighted.

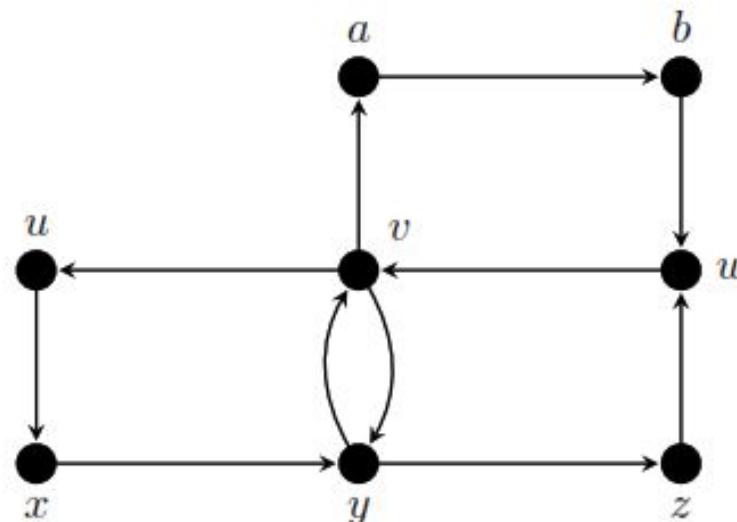


Output: The shortest paths from g to all other vertices can be found highlighted above. For example the shortest path from g to c is $g f d c$ and has a total weight 7, as shown by the label of c .

Example 2.18 Find the diameter and radius for the graph below.



Solution: Note that f is adjacent to all vertices except c , but there is a path of length 2 from f to c . As no vertex is adjacent to all other vertices, we know the radius is 2. The longest path between two vertices is from g to c , and is of length 3, so the diameter is 3.



Example 10.2.6. For the digraph D in Example 10.2.4 (see Fig. 10.2.3), we have:

$$\begin{aligned} e(z) &= \max\{d(z, a), d(z, b), d(z, u), d(z, v), d(z, w), d(z, x), d(z, y), d(z, z)\} \\ &= \max\{3, 4, 3, 2, 1, 4, 3, 0\} = 4. \end{aligned}$$

It can be verified that the eccentricities of the eight vertices in D are shown in the following table:

Vertex	a	b	u	v	w	x	y	z
Eccentricity	5	4	5	3	3	4	3	4

Example 10.2.6. For the digraph D in Example 10.2.4 (see Fig. 10.2.3), we have:

$$\begin{aligned} e(z) &= \max\{d(z, a), d(z, b), d(z, u), d(z, v), d(z, w), d(z, x), d(z, y), d(z, z)\} \\ &= \max\{3, 4, 3, 2, 1, 4, 3, 0\} = 4. \end{aligned}$$

It can be verified that the eccentricities of the eight vertices in D are shown in the following table:

Vertex	a	b	u	v	w	x	y	z
Eccentricity	5	4	5	3	3	4	3	4

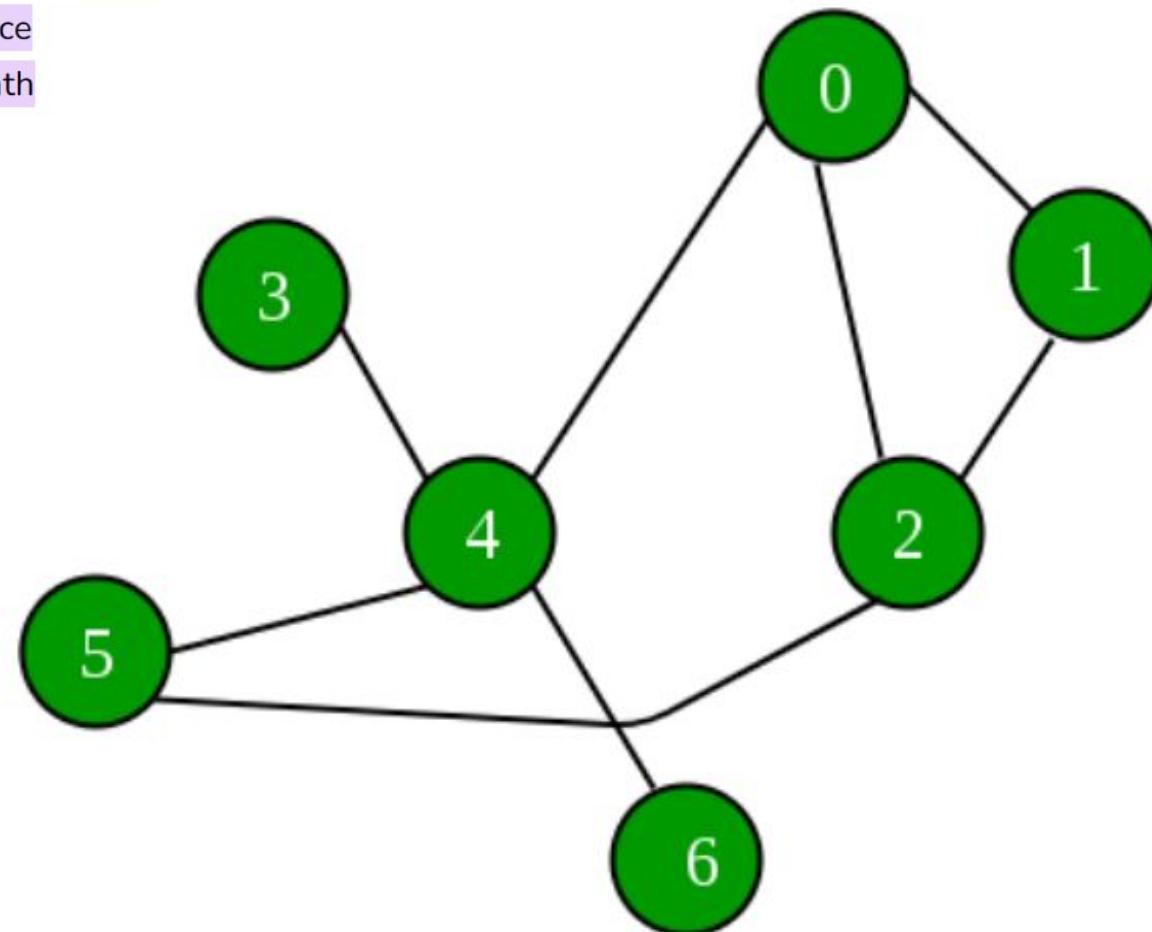
For the digraph D in Example 10.2.4,

$$\text{diam}(D) = 5 = d(a, x) = d(a, z) = d(u, b) = \max\{d(p, q) \mid p, q \in V(D)\}.$$

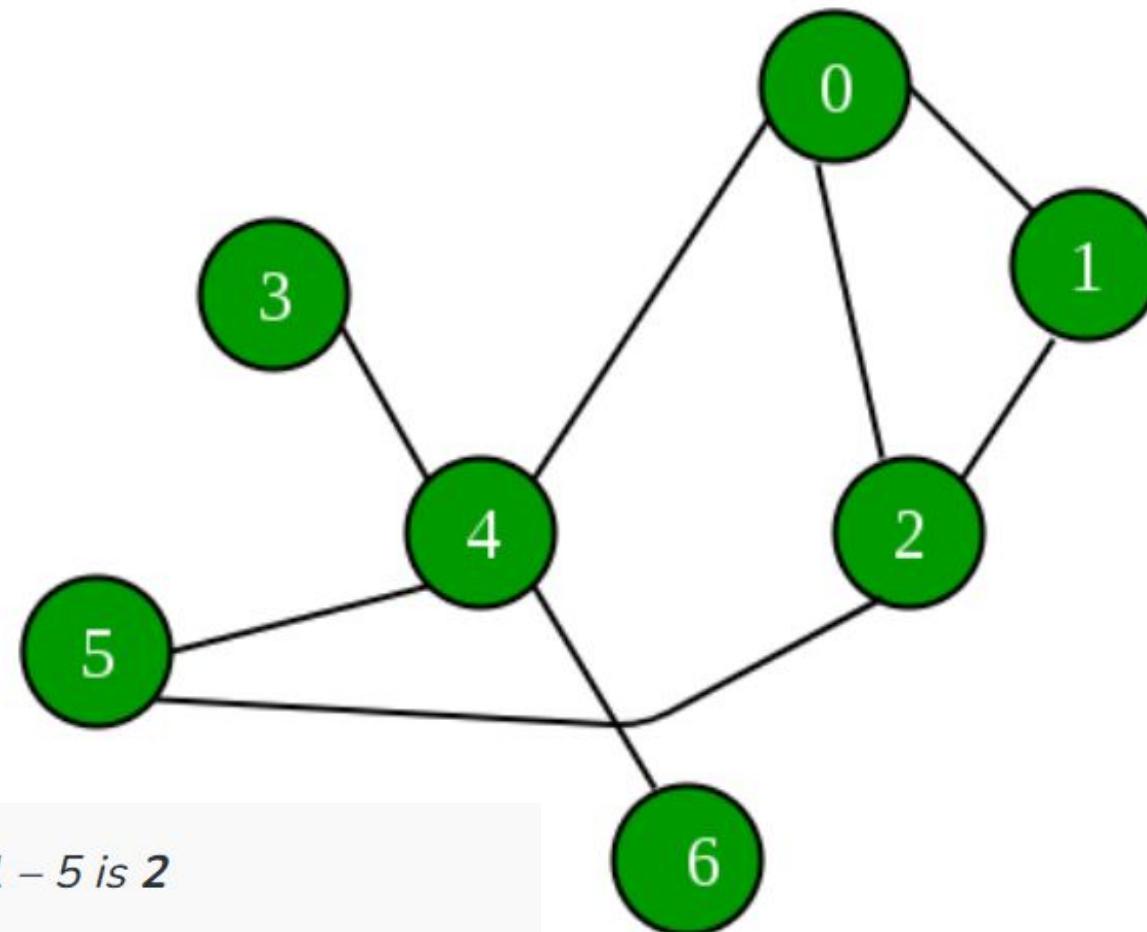
Distance from 1-5

2. The distance between two Vertices –

The distance between two vertices in a graph is the number of edges in a shortest or minimal path. It gives the available minimum distance between two edges. There can exist more than one shortest path between two vertices.



Distance from 1-5



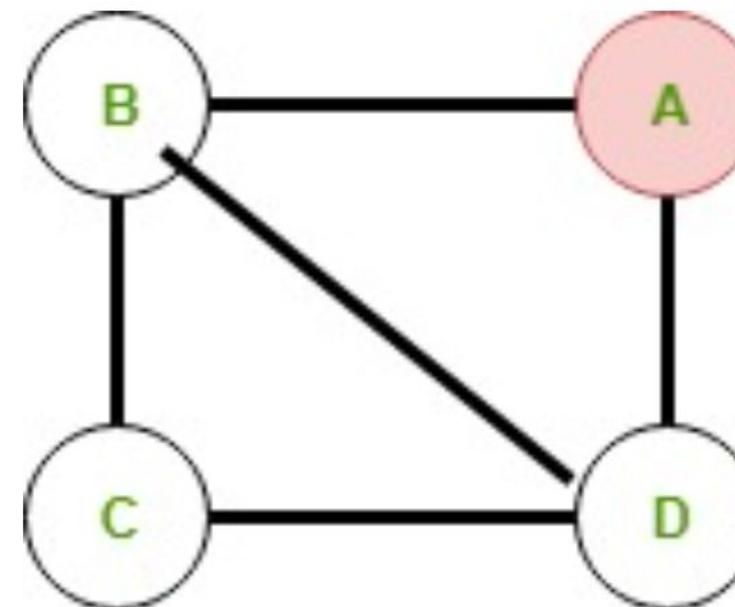
Shortest Distance between 1 – 5 is 2

$1 \rightarrow 2 \rightarrow 5$

3. Eccentricity of graph –

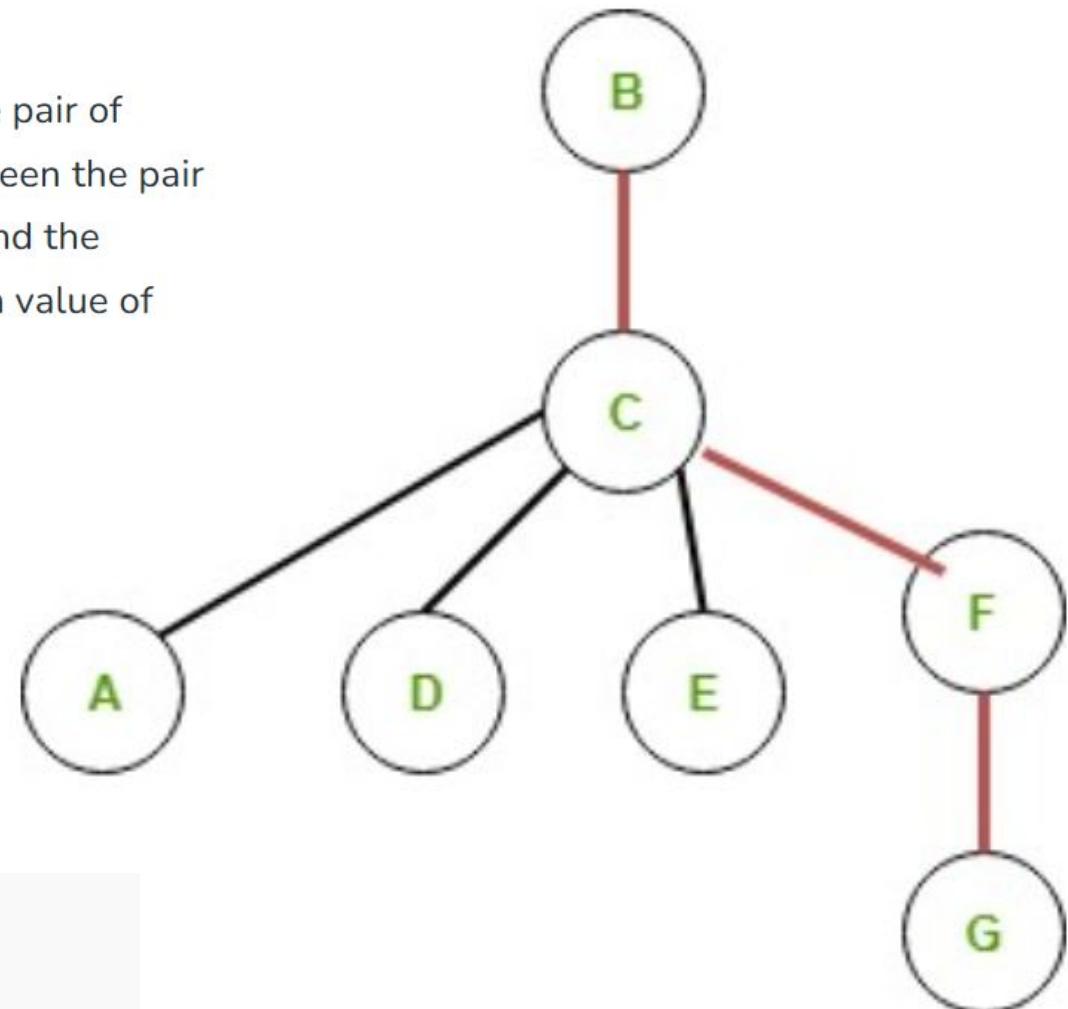
It is defined as the maximum distance of one vertex from other vertex.

The maximum distance between a vertex to all other vertices is considered as the eccentricity of the vertex. It is denoted by $e(V)$.



4. Diameter of graph –

The diameter of graph is the maximum distance between the pair of vertices. It can also be defined as the maximal distance between the pair of vertices. Way to solve it is to find all the paths and then find the maximum of all. It can also be found by finding the maximum value of eccentricity from all the vertices.



Diameter: 3

$BC \rightarrow CF \rightarrow FG$

Here the eccentricity of the vertex B is 3 since $(B,G) = 3$. (Maximum Eccentricity of Graph)

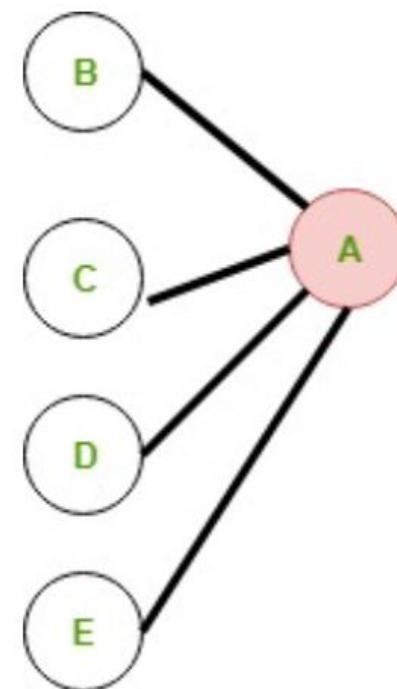
6. Centre of graph –

It consists of all the vertices whose eccentricity is minimum. Here the eccentricity is equal to the radius. For example, if the school is at the center of town it will reduce the distance buses has to travel. If eccentricity of two vertex is same and minimum among all other both of them can be the center of the graph.

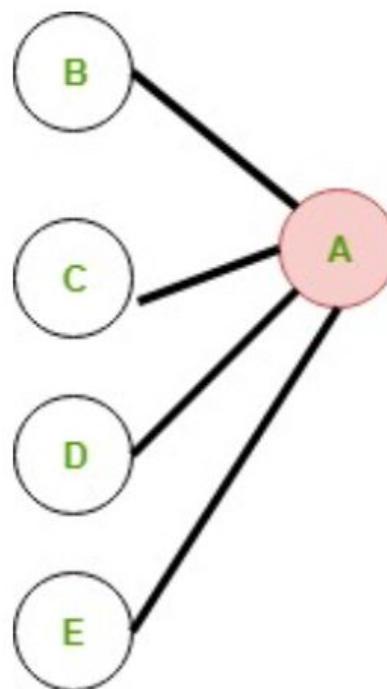
Centre: A

Inorder to find the center of the graph, we need to find the eccentricity of each vertex and find the minimum among all of them.

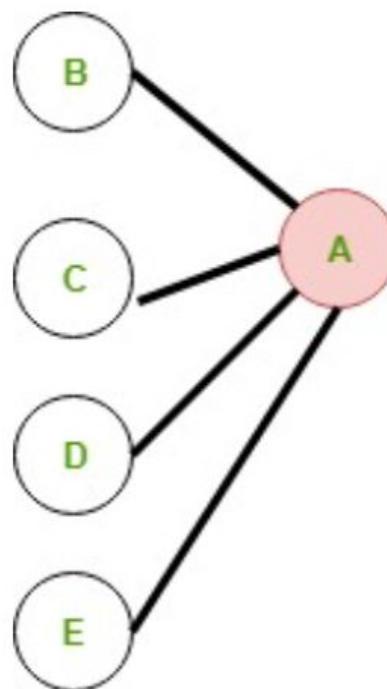
The minimum eccentricity vertex will be considered as the center.



Definition 2.29 Let G be a graph with $\text{rad}(G) = r$. Then x is a **central vertex** if $\epsilon(x) = r$. Moreover, the **center** of G is the graph $C(G)$ that is induced by the central vertices of G .



Definition 2.29 Let G be a graph with $\text{rad}(G) = r$. Then x is a **central vertex** if $\epsilon(x) = r$. Moreover, the **center** of G is the graph $C(G)$ that is induced by the central vertices of G .



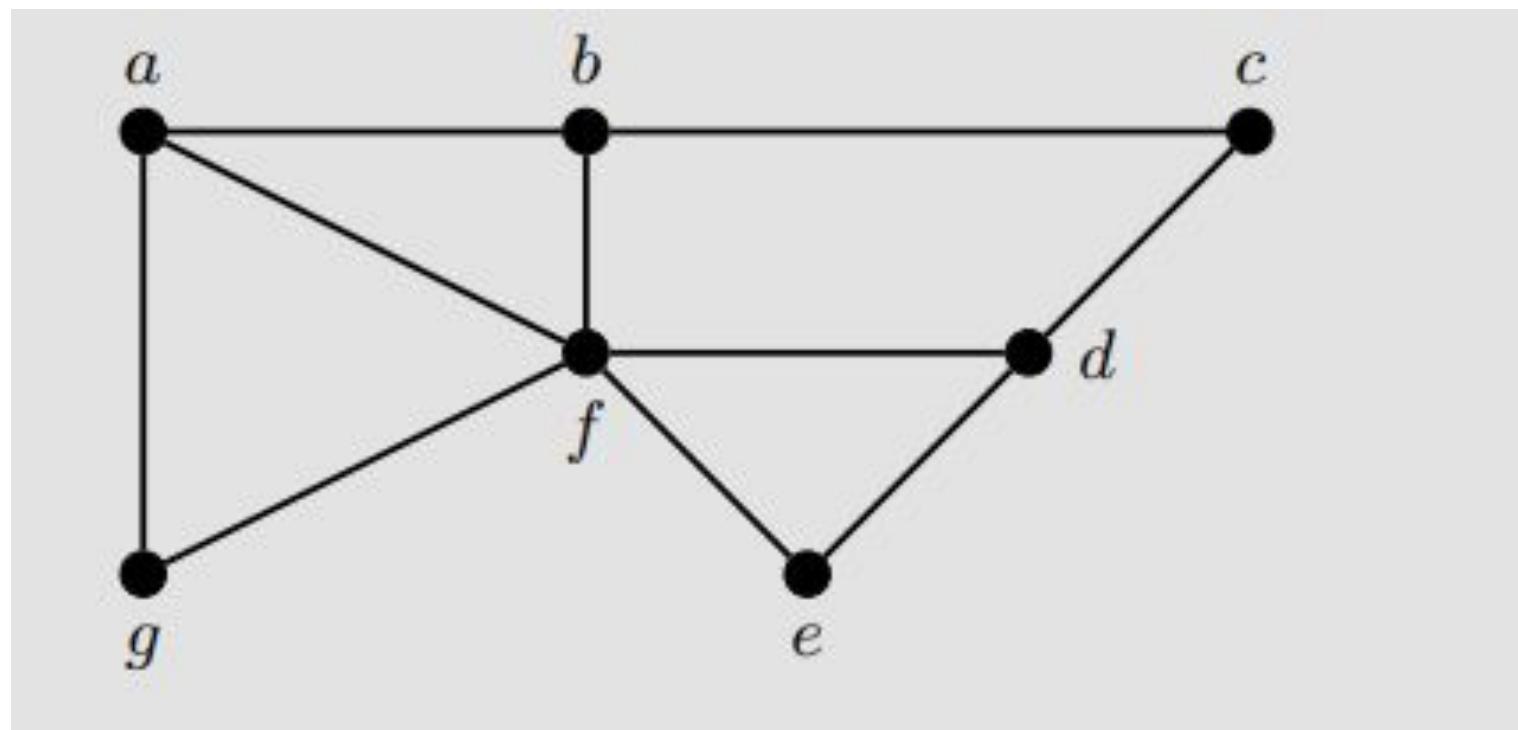
Theorem 2.26 If G is disconnected then \overline{G} is connected and $diam(\overline{G}) \leq 2$.

Theorem 2.27 For a simple graph G if $rad(G) \geq 3$ then $rad(\overline{G}) \leq 2$.

Theorem 2.28 For any simple graph G , $rad(G) \leq diam(G) \leq 2rad(G)$.

Definition 2.30 Given a graph G , the *girth* of G , denoted $g(G)$, is the minimum length of a cycle in G . The *circumference* of G is the maximum length of a cycle.

Example 2.20 Find the girth and circumference for the graph from Example 2.18.



Example 2.20 Find the girth and circumference for the graph from Example 2.18.

Solution: Since the graph is simple, we know the girth must be at least 3, and since we can find triangles within the graph we know $g(G) = 3$. Moreover, the circumference is 7 since we can find a cycle containing all the vertices (try it!).

EX, #:2.4

Problems: 2.1-2.9, 2.15, 2.16, 2.27, 2.28

Tree

Definition 3.1 A graph G is

- *acyclic* if there are no cycles or circuits in the graph.
- a *tree* if it is both acyclic and connected.
- a *forest* if it is an acyclic graph.

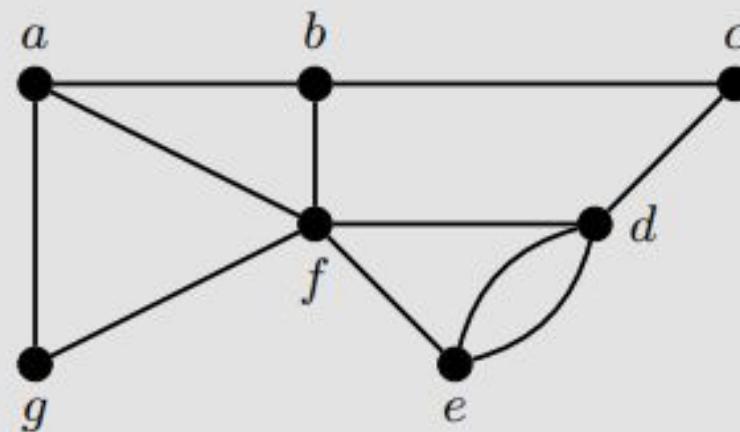
In addition, a vertex of degree 1 is called a *leaf*.

- A **tree** is a connected graph without cycles
- A **tree** is a connected graph on n vertices with $n - 1$ edges
- A graph is a **tree** if and only if there is a unique simple path between any pair of its vertices

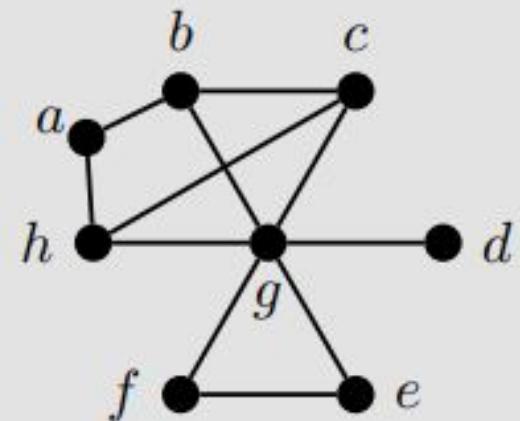
Definition 3.2 A *spanning tree* is a spanning subgraph that is also a tree.

- A **Spanning Tree** of a graph G , is a subgraph of G which is a tree and contains all vertices of G
- A **Minimum Spanning Tree** of a weighted graph G is a spanning tree of the smallest weight

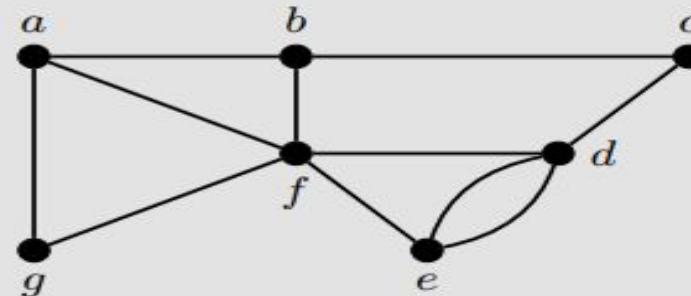
Example 3.1 For each of the graphs below, find a spanning tree and a subgraph that does not span.



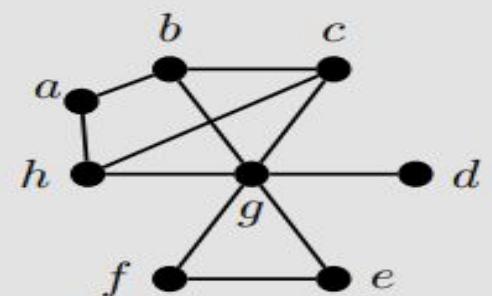
G_1



G_2

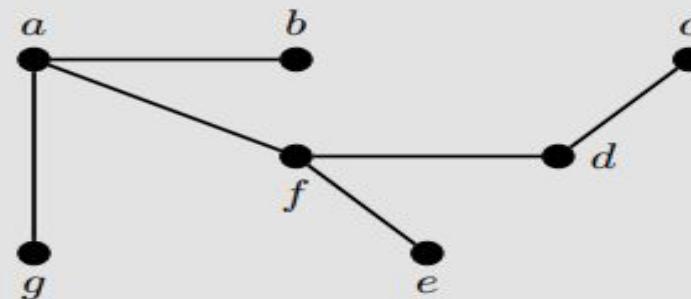


G_1

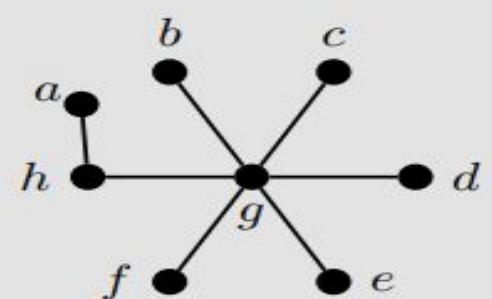


G_2

Solution: To find a spanning tree, we must form a subgraph that is connected, acyclic, and includes every vertex from the original graph. The graphs T_1 and T_2 below are two examples of spanning trees for their respective graphs; other examples exist.

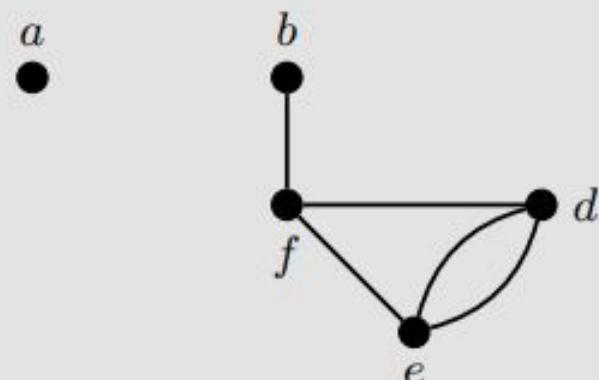


T_1

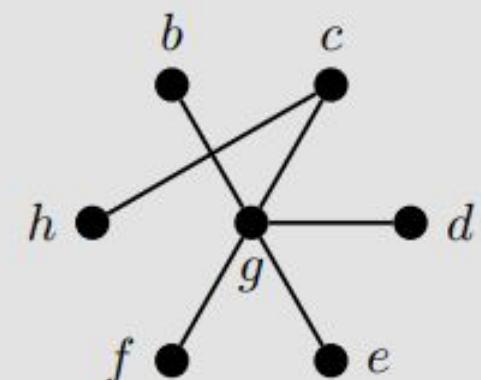


T_2

The subgraph H_1 below is neither spanning nor a tree since some vertices from G_1 are missing and there is a multi-edge (and hence a circuit) between d and e . The subgraph H_2 below is not spanning since it does not contain vertex a , but it is a tree since no circuits or cycles exist. As above, these are merely examples and other non-spanning subgraphs exist.



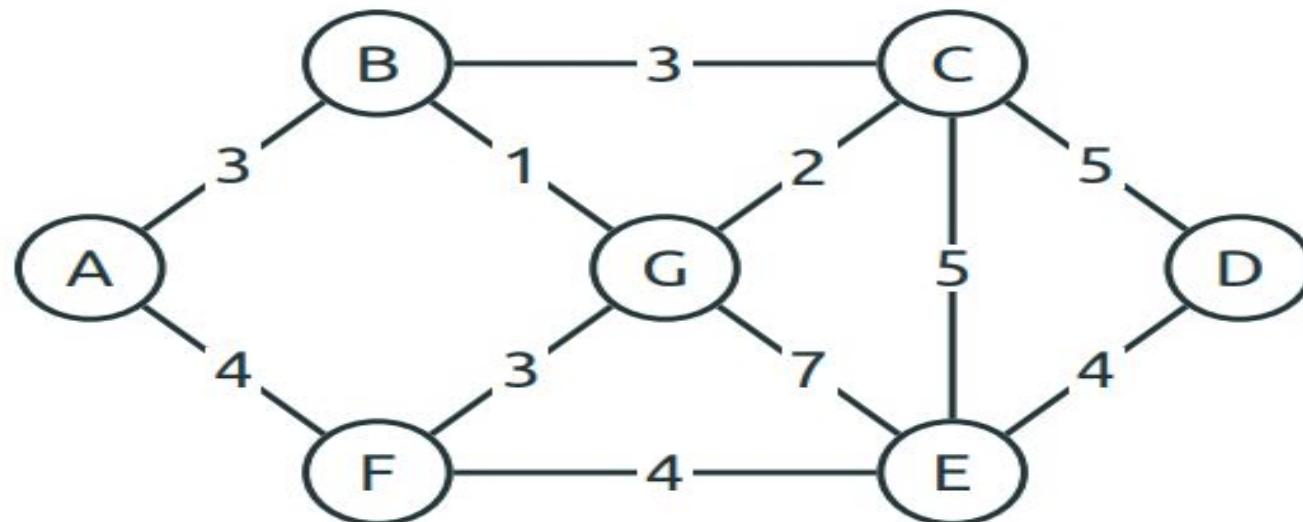
H_1



H_2

Definition 3.3 Given a weighted graph $G = (V, E, w)$, T is a **minimum spanning tree**, or MST, of G if it is a spanning tree with the least total weight.

Minimum Spanning Tree: Examples



Kruskal's Algorithm

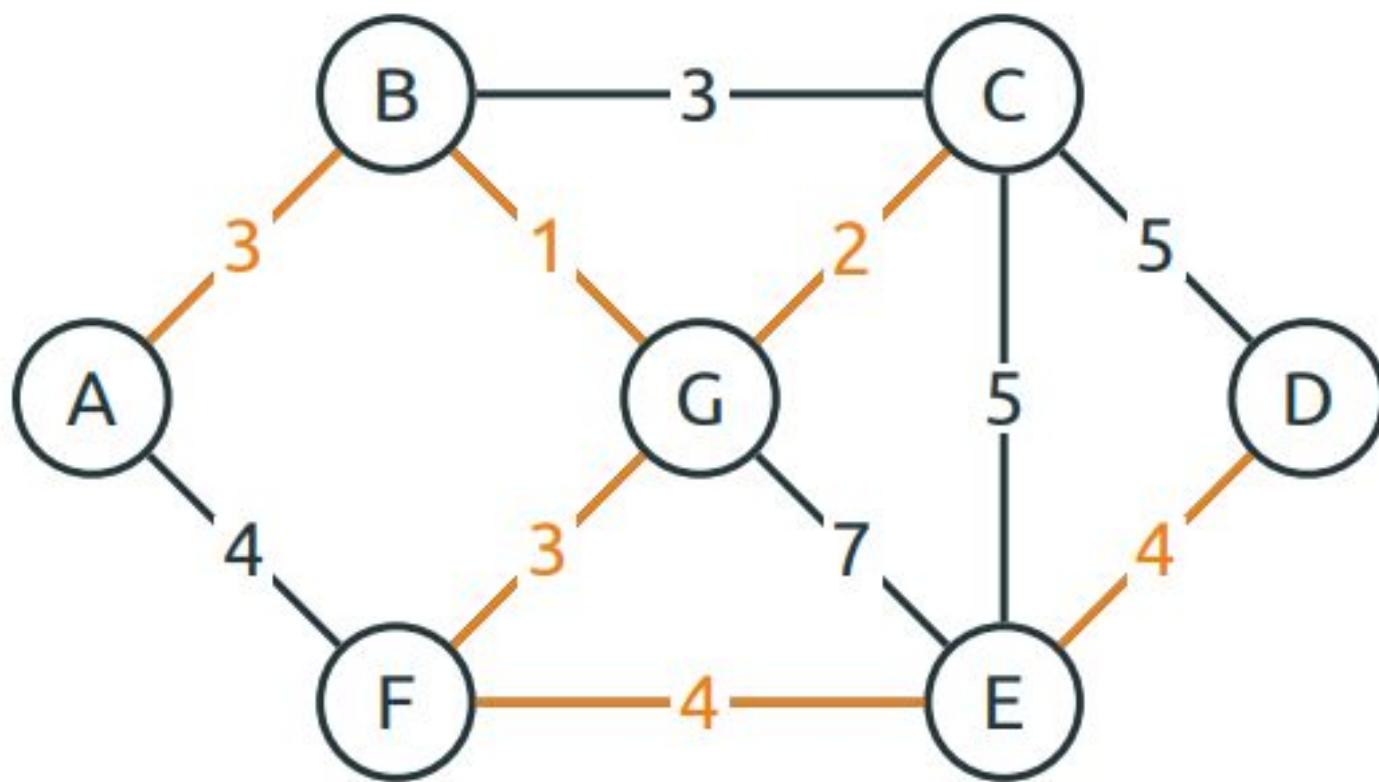
Input: Weighted connected graph $G = (V, E)$.

Steps:

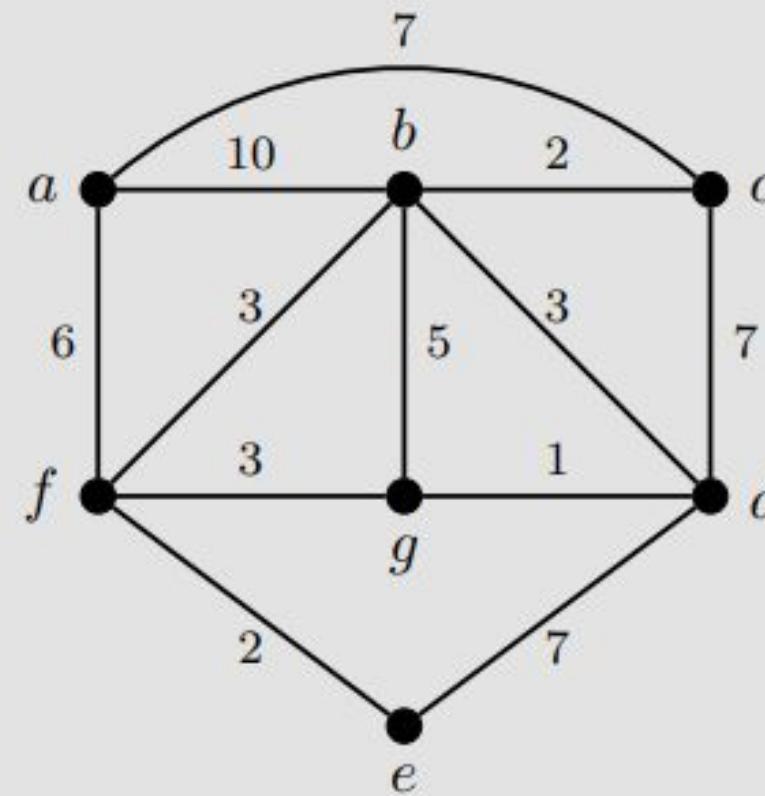
1. Choose the edge of least weight. Highlight it and add it to $T = (V, E')$.
2. Repeat Step (1) so long as no circuit is created. That is, keep picking the edges of least weight but skip over any that would create a cycle in T .

Output: Minimum spanning tree T of G .

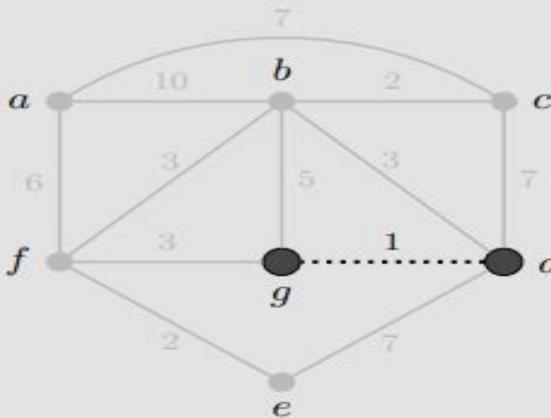
Minimum Spanning Tree: Examples



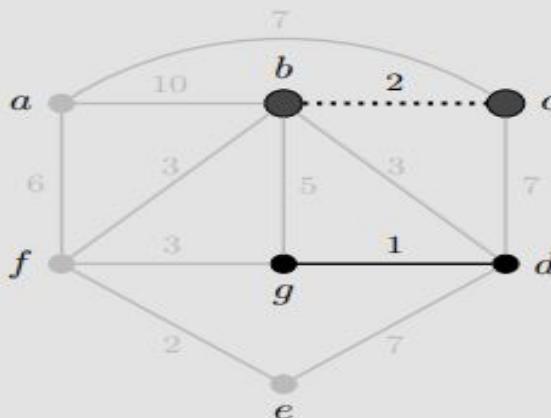
Example 3.2 Find the minimum spanning tree of the graph G below using Kruskal's Algorithm.

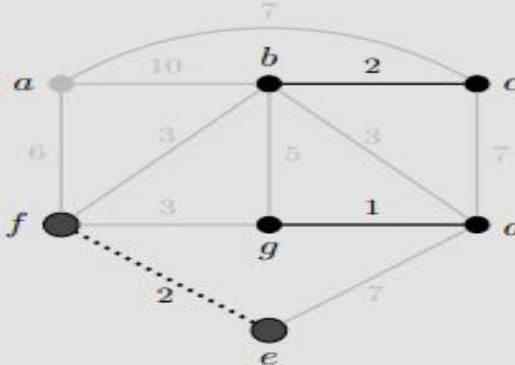


Step 1: Pick the smallest edge, gd and highlight it.

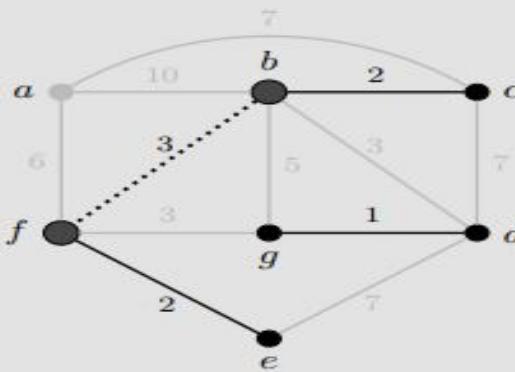


Step 2: Pick the next smallest edge. There are two edges of weight 2 (bc and ef). Either is a valid choice. We choose bc .

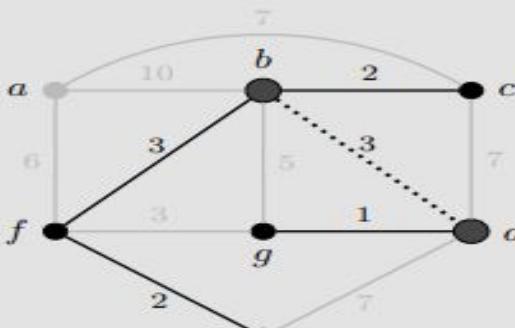


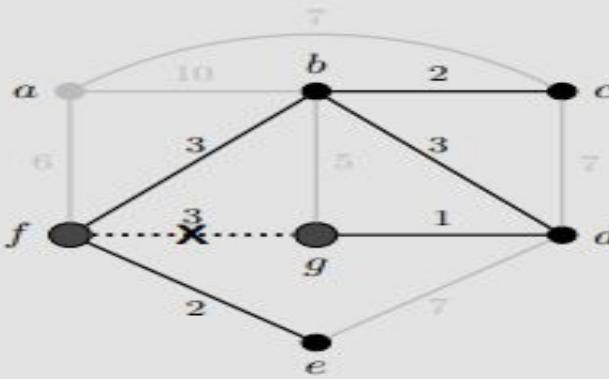


Step 4: The next smallest edge weight is 3, and there are 3 edges to choose from (bd , bf , and fg) . We randomly pick bf .

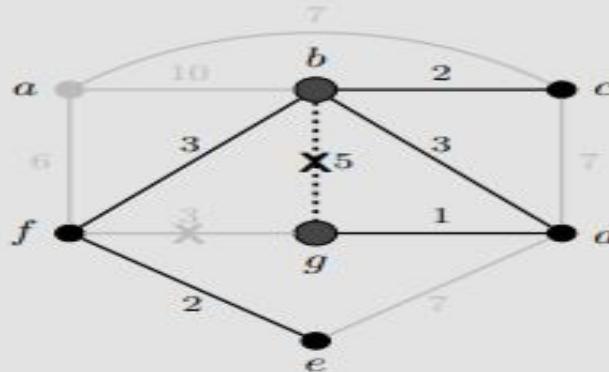


Step 5: Both of the other edges of weight 3 are still available. We choose bd .

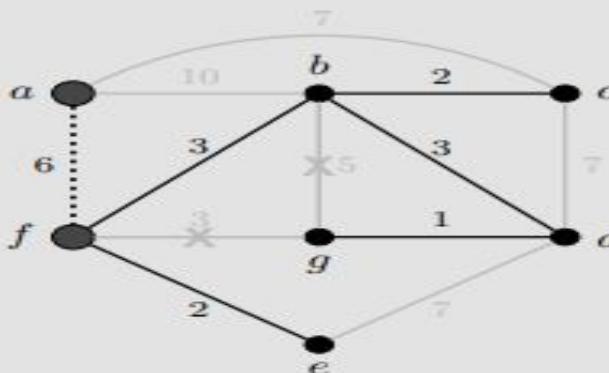


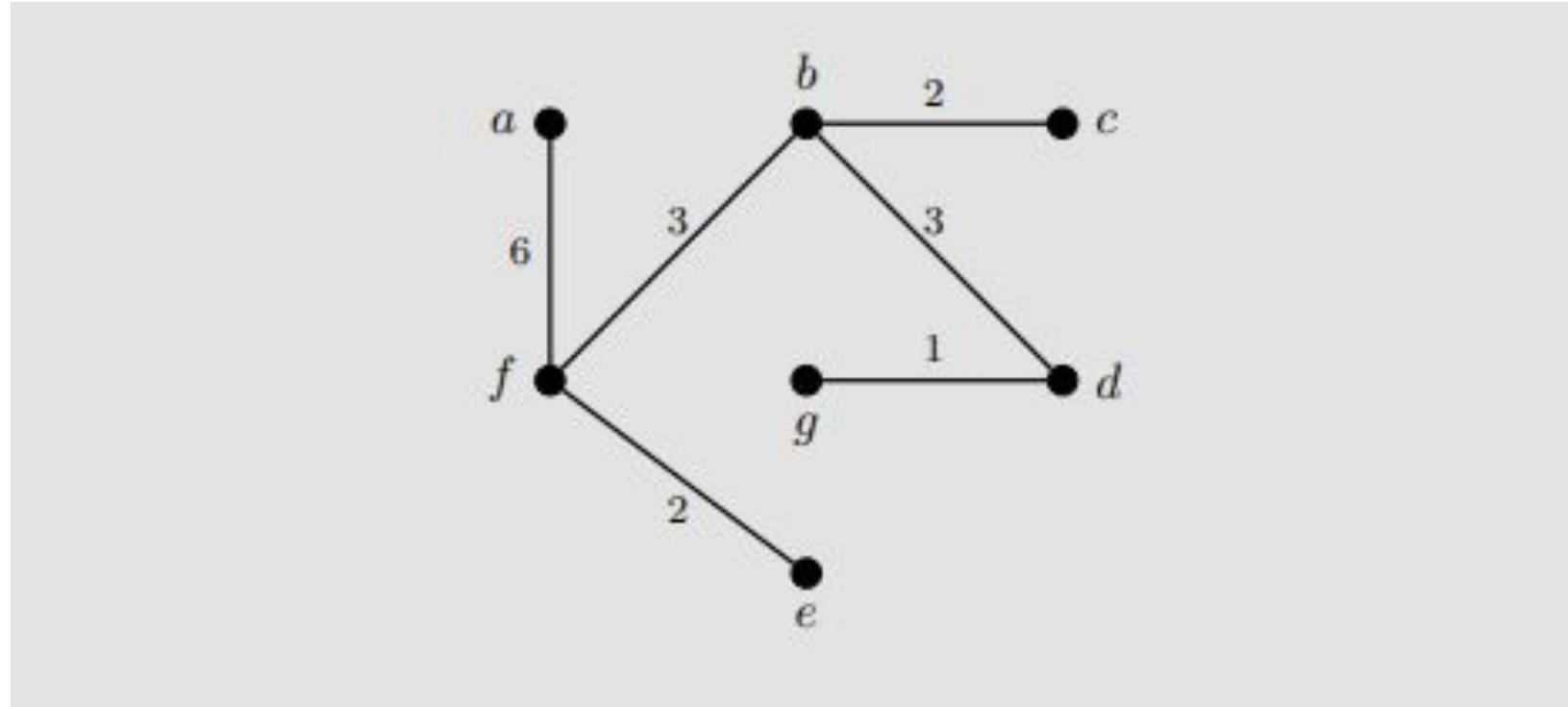


The next smallest edge is bg of weight 5. Again, we cannot choose this edge since it would create a circuit ($b d g b$).



The next available edge is af of weight 6. This is also the last edge needed since we now have a tree containing all the vertices of G .





Output: The following tree with total weight 17.

Prim's Algorithm

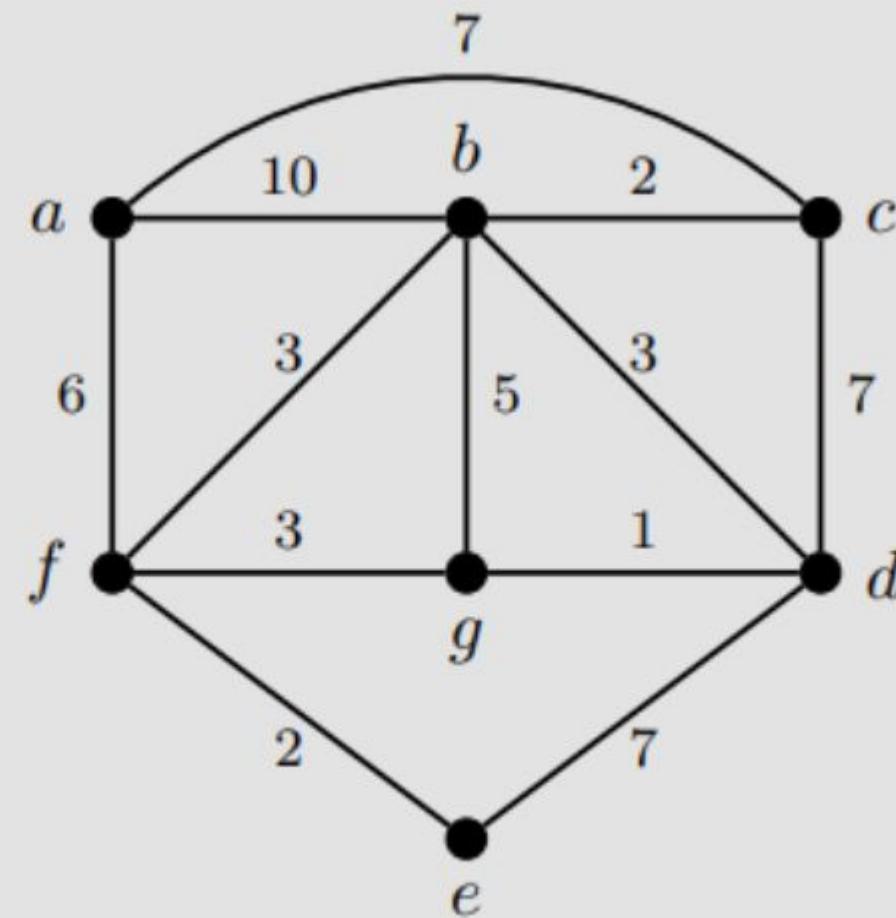
Input: Weighted connected graph $G = (V, E)$.

Steps:

1. Let v be the root. If no root is specified, choose a vertex at random. Highlight it and add it to $T = (V', E')$.
2. Among all edges incident to v , choose the one of minimum weight. Highlight it. Add the edge and its other endpoint to T .
3. Let S be the set of all edges with exactly endpoint from $V(T)$. Choose the edge of minimum weight from S . Add it and its other endpoint to T .
4. Repeat Step (3) until T contains all vertices of G , that is $V(T) = V(G)$.

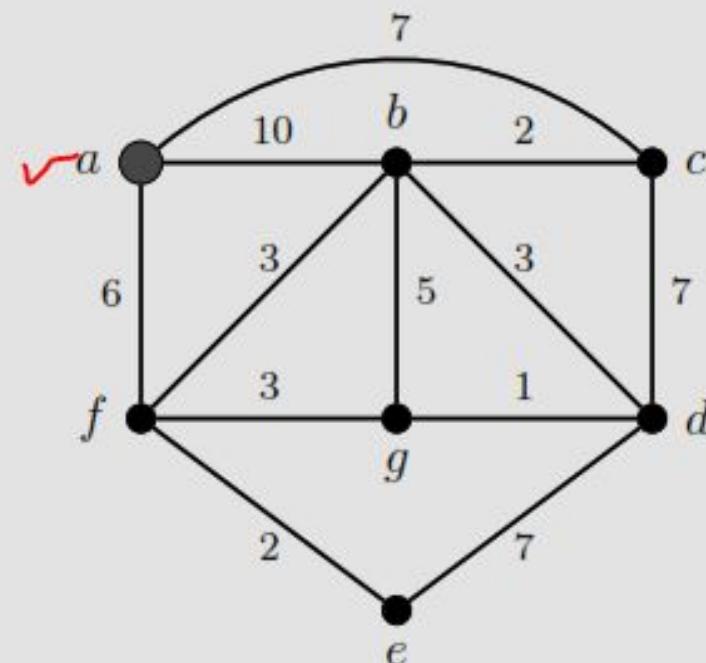
Output: Minimum spanning tree T of G .

Example 3.3 Use Prim's algorithm to find a minimum spanning tree for the graph given in Example 3.2.

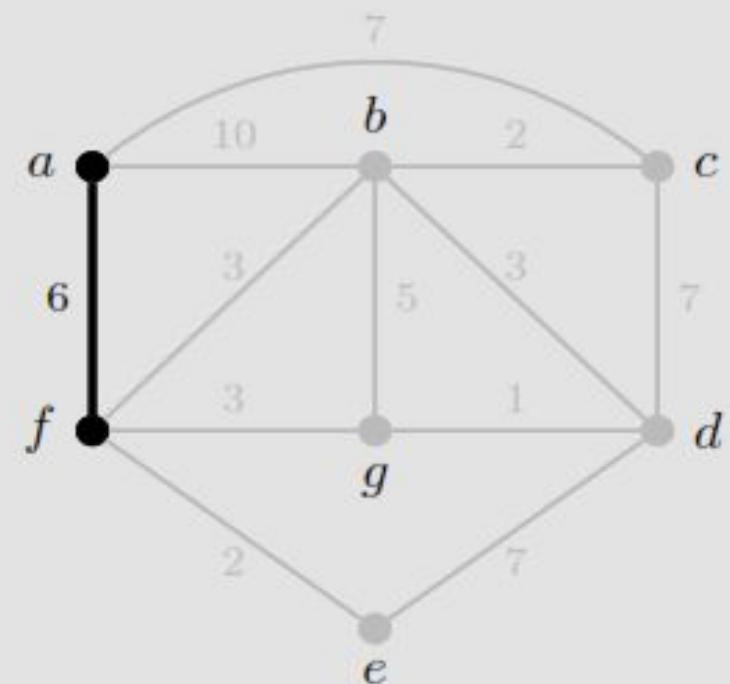
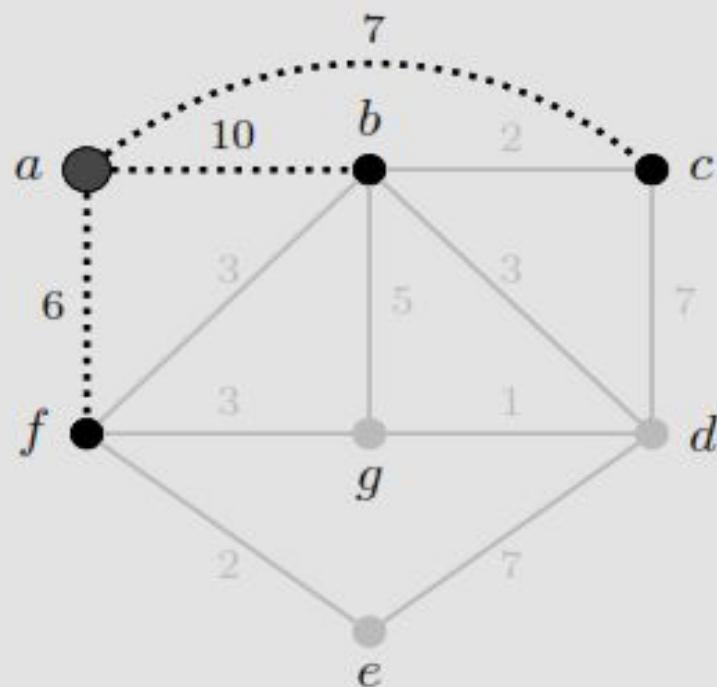


Solution: As before, the newest edges under consideration will be shown as dotted lines and the previously chosen edges will be in black. Unchosen edges will be shown in gray. The vertices in T at each stage will be highlighted as well

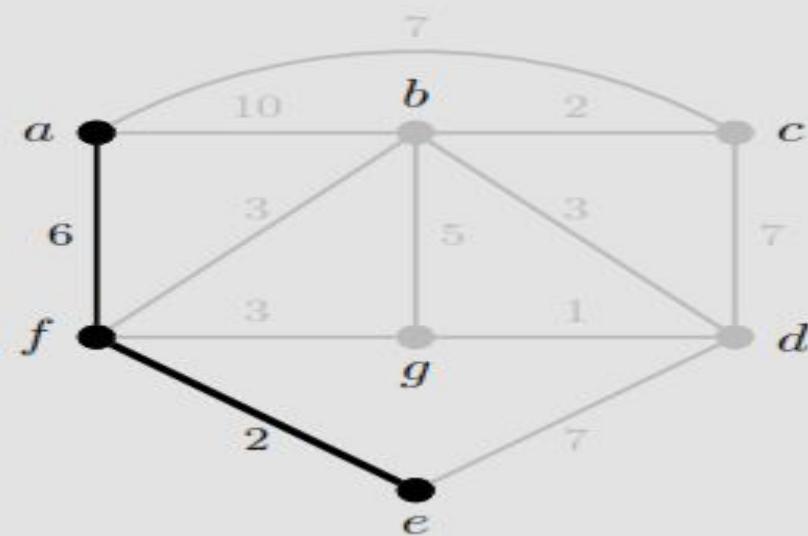
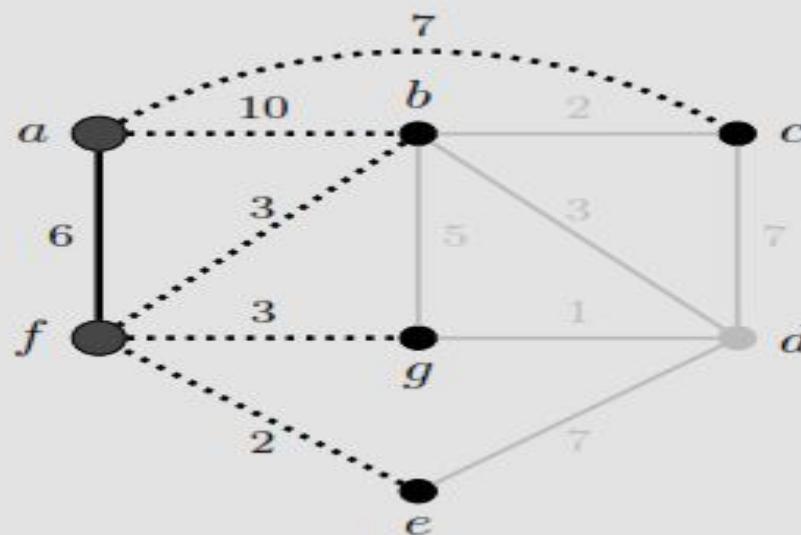
Step 1: Since no root was specified, we choose a as the starting vertex.



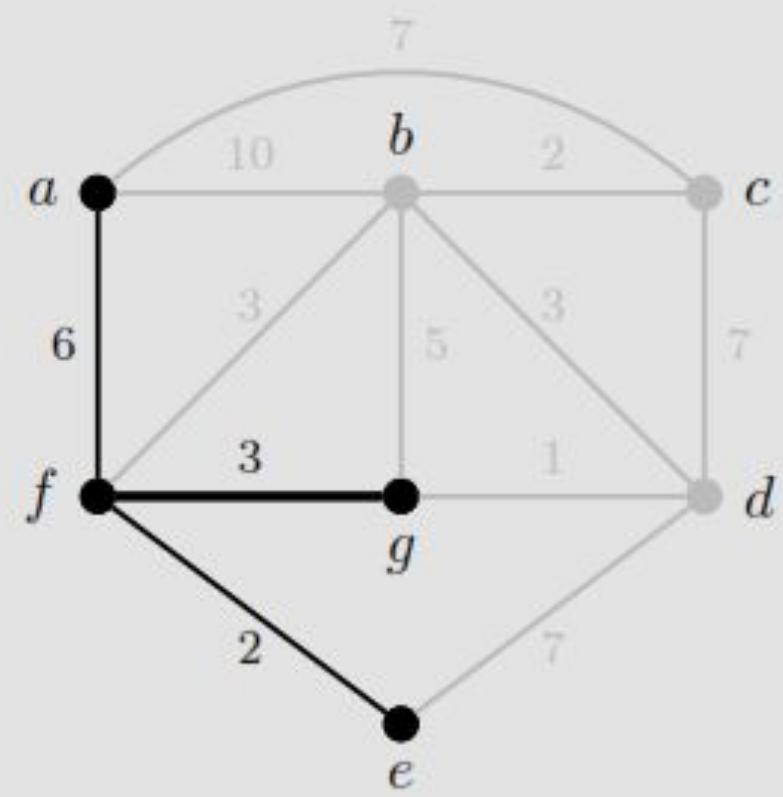
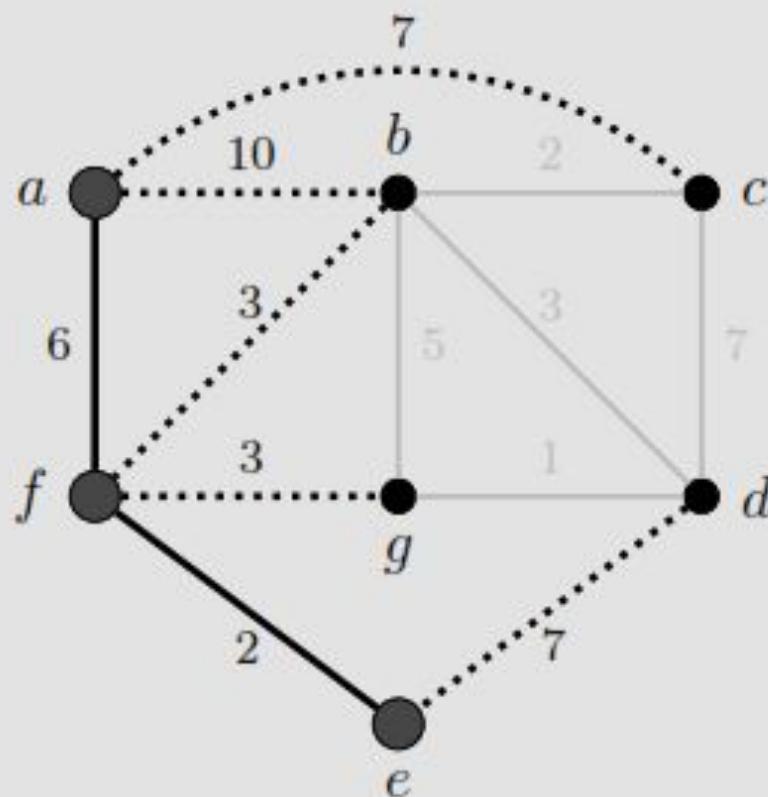
Step 2: We consider the edges incident to a , namely ab , ac , and af . These are shown as dotted lines in the graph on the left. The edge of least weight is af . This is added to the tree, shown in bold on the right.



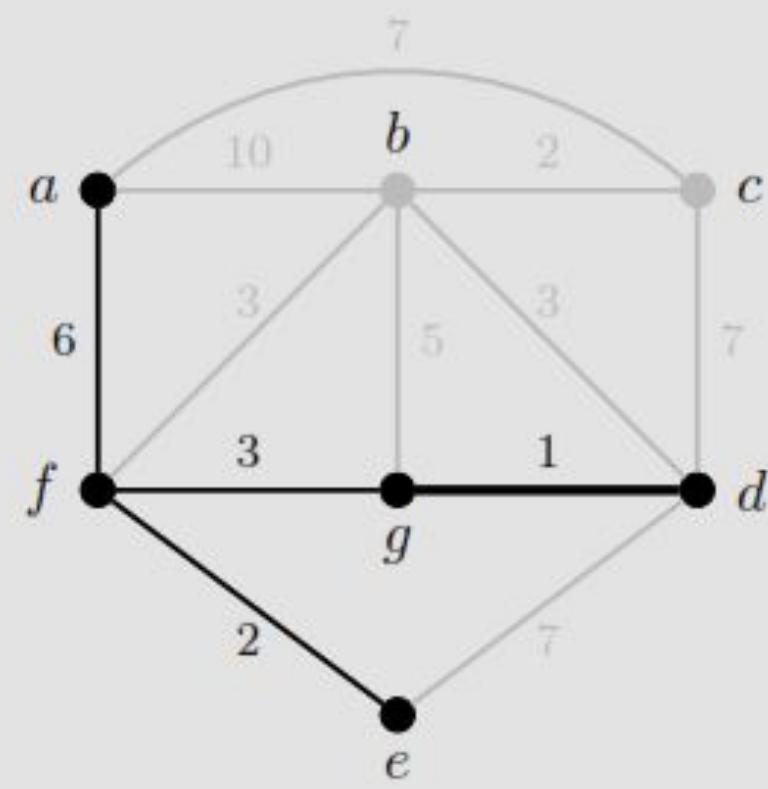
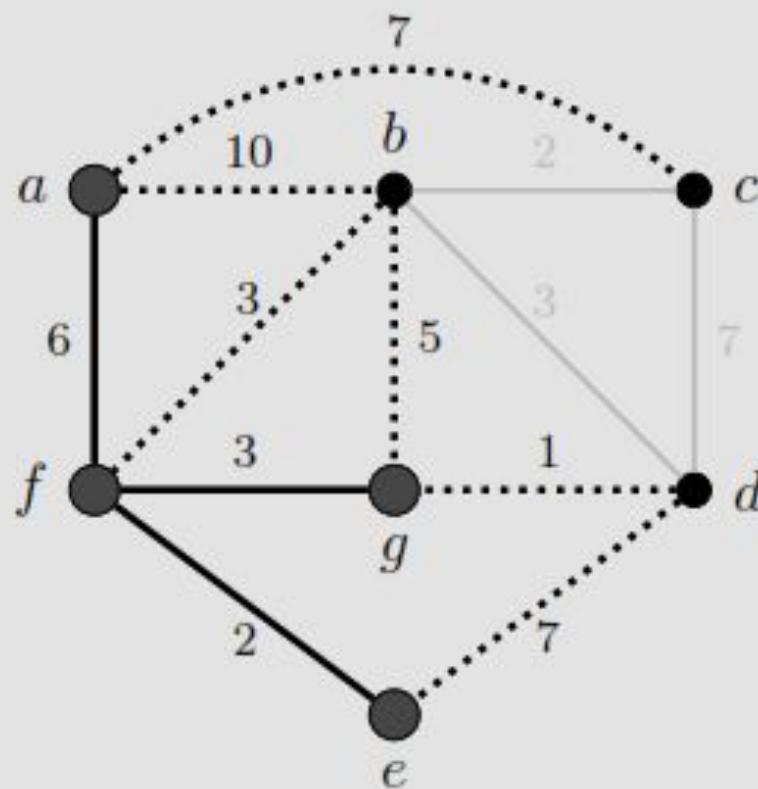
Step 3: The set S consists of edges with one endpoint as a or f , as shown in the graph to the left. The edge of minimum weight from these is ef . This is added to the tree, as shown on the right.



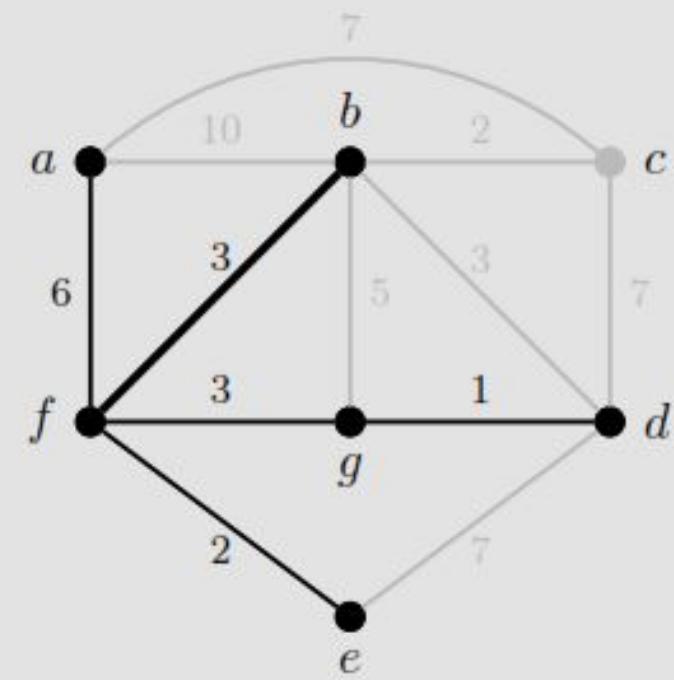
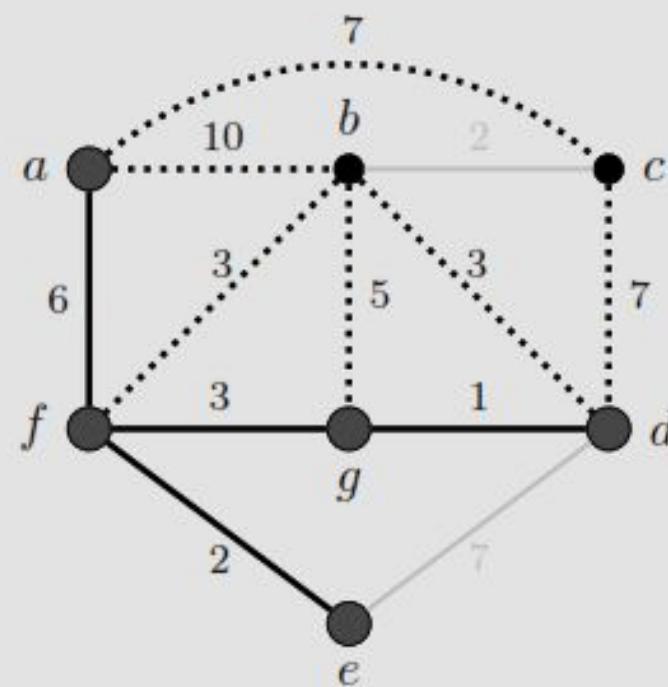
Step 4: The new set S consists of edges with one endpoint as a , e , or f . The next edge added to the tree could either be fg or fb . We choose fg .



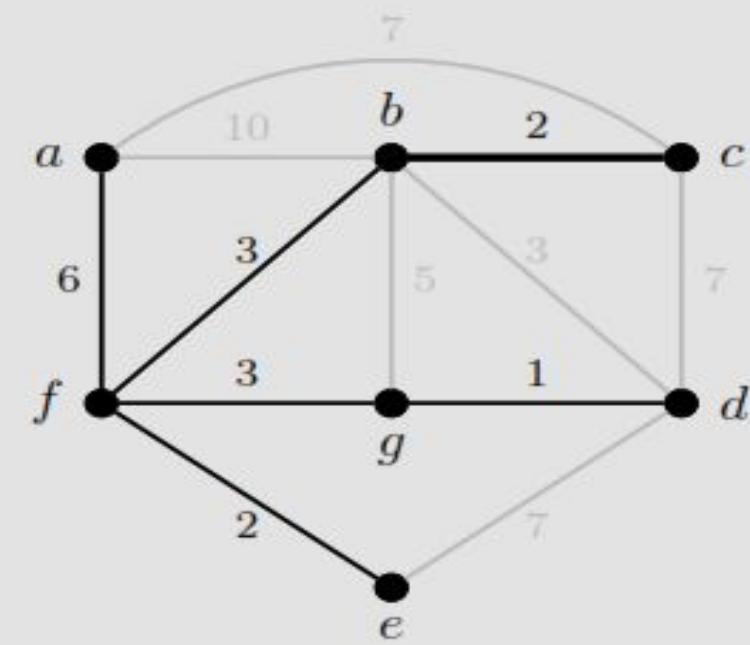
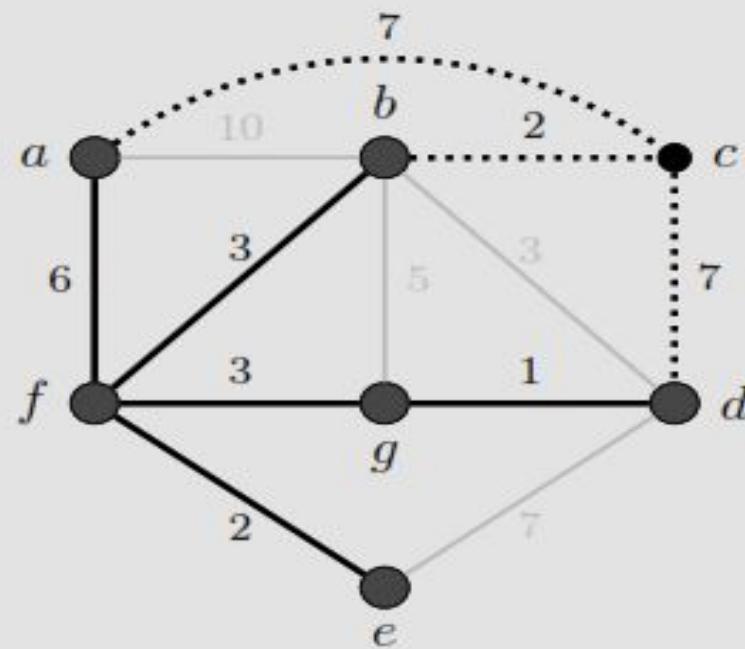
Step 5: We consider the edges where exactly one endpoint is from a, e, f , or g . The next edge to add to the tree is dg .



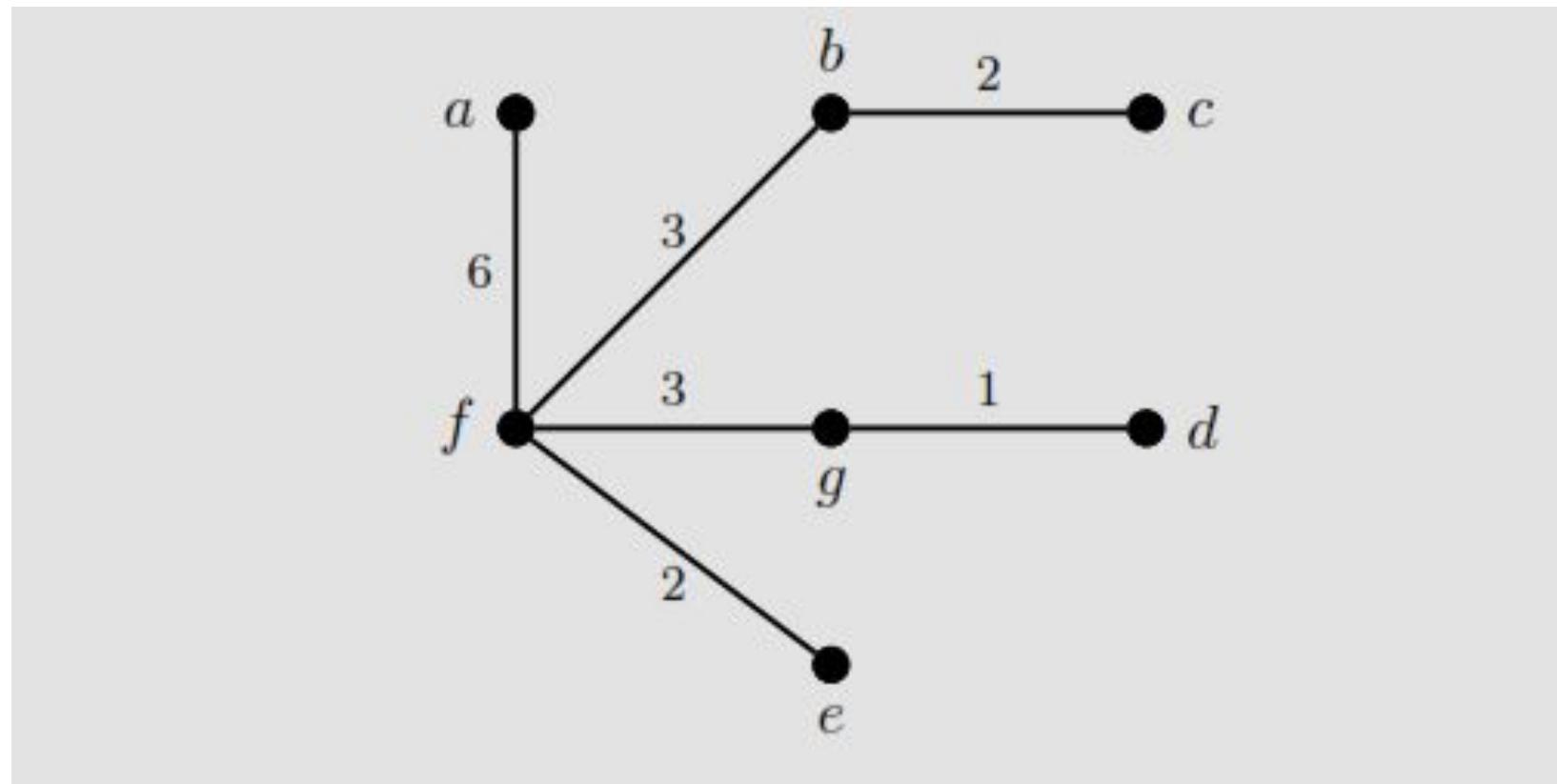
Step 6: The edges to consider must have exactly one endpoint from a, d, e, f , or g . Note that de is no longer available since both endpoints are already part of the tree (and its addition would create a cycle). There are two possible minimum weight edges, bf or bd . We choose bf .



Step 7: The only edges we can consider are those with one endpoint of c since this is the only vertex not part of our tree. The edge of minimum weight is bc .



Output: A minimum spanning tree of total weight 17.



Theorem 3.4 Every tree with at least two vertices has a leaf.

Proof: Suppose for a contradiction that there exists a tree T with at least two vertices that does not contain a leaf. Since T must be connected, we know no vertex has degree 0, and therefore every vertex of T must have degree at least 2. But then by Theorem 2.5 we know T must have a cycle, which contradicts that T is acyclic. Thus T must contain a leaf.

Lemma 3.5 Given a tree T with a leaf v , the graph $T - v$ is still a tree.

Theorem 3.6 A tree with n vertices has $n - 1$ edges for all $n \geq 1$.

Corollary 3.7 The total degree of a tree on n vertices is $2n - 2$.

Proposition 3.8 Let T be a tree. Then for every pair of distinct vertices x and y there exists a unique $x - y$ path.

Proposition 3.9 Every tree is minimally connected, that is the removal of any edge disconnects the graph.

Proposition 3.10 If any edge e is added to a tree T then $T + e$ contains exactly one cycle.

Theorem 3.12 Kruskal's Algorithm produces a minimum spanning tree.

Theorem 3.11 Let T be a graph with n vertices. The following conditions are equivalent:

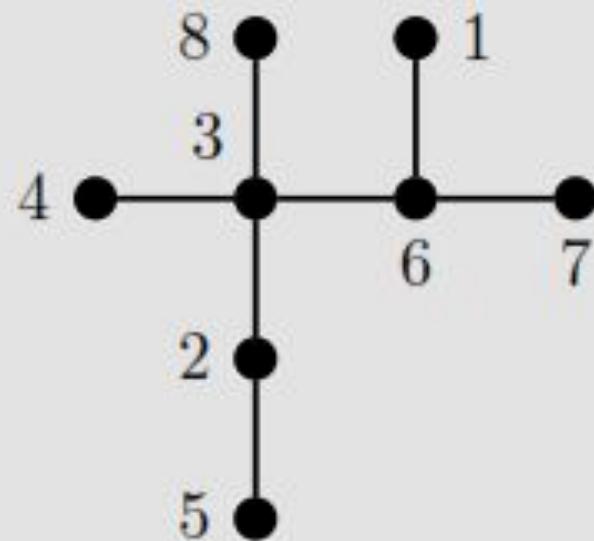
- (a) T is a tree.
- (b) T is acyclic and contains $n - 1$ edges.
- (c) T is connected and contains $n - 1$ edges.
- (d) There is a unique path between every pair of distinct vertices in T .
- (e) Every edge of T is a bridge.
- (f) T is acyclic and for any edge e from T , $T + e$ contains exactly one cycle.

Lec # 16, 17 & 18

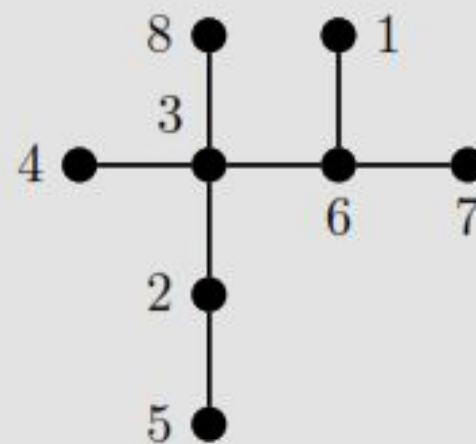
Definition 3.13 Given a tree T on $n > 2$ vertices (labeled $1, 2, \dots, n$), the *Prüfer sequence* of T is a sequence $(s_1, s_2, \dots, s_{n-2})$ of length $n - 2$ defined as follows:

- Let l_1 be the leaf of T with the smallest label.
- Define T_1 to be $T - l_1$.
- For each $i \geq 1$, define $T_{i+1} = T_i - l_{i+1}$, where l_{i+1} is the leaf with the smallest label of T_i .
- Define s_i to be the neighbor of l_i .

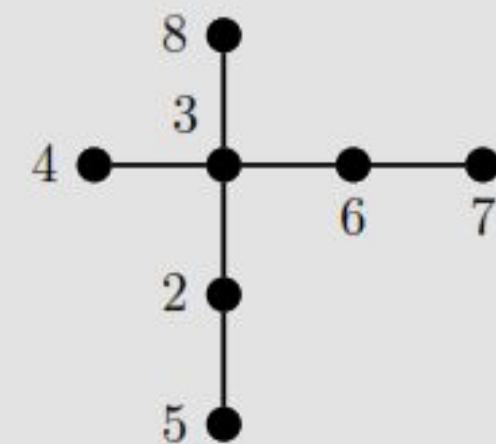
Example 3.4 Find the Prüfer sequence for the tree below.



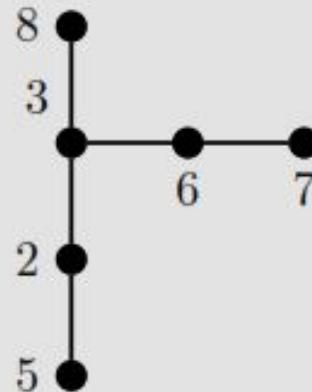
Solution: The pruning of leaves is shown below, with l_i and s_i listed for each step.



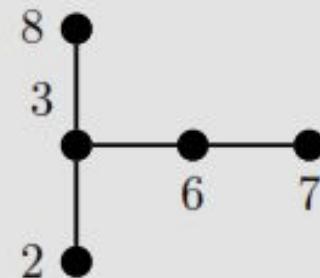
$$T : l_1 = 1 \quad s_1 = 6$$



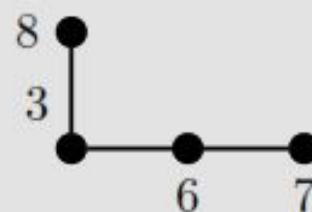
$$T_1 : l_2 = 4 \quad s_2 = 3$$



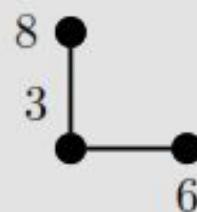
$$T_2 : l_3 = 5 \quad s_3 = 2$$



$$T_3 : l_4 = 2 \quad s_4 = 3$$



$$T_4 : l_5 = 7 \quad s_5 = 6$$



$$T_5 : l_6 = 6 \quad s_6 = 3$$

The Prüfer sequence for the tree T is $(6, 3, 2, 3, 6, 3)$.

Example 3.5 Find the tree associated to the Prüfer sequence $(1, 5, 5, 3, 2)$.

Example 3.5 Find the tree associated to the Prüfer sequence $(1, 5, 5, 3, 2)$.

Solution: First note that the sequence is of length 5, so the tree must have 7 vertices. At every stage we will consider the possible leaves of a subtree created by the earlier pruning. Initially our set of leaves is $L = \{4, 5, 7\}$, since these do not appear in the sequence and so must be the leaves of the full tree. To begin building the tree, we look for the smallest value not appearing in the sequence, namely 4. This must be adjacent to 1 by entry s_1 , as shown on the next page.



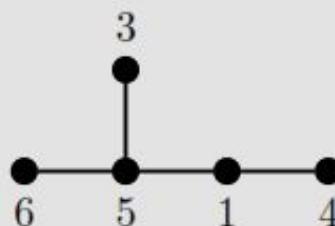
Next, we remove s_1 from the Prüfer sequence and consider the subsequence $(5, 5, 3, 2)$. Now our set of leaves is $L = \{1, 6, 7\}$ since 4 has already been placed as a leaf and 1 is no longer appearing in the sequence. Since 1 is the smallest value in our set L , we know it must be adjacent to $s_2 = 5$, as shown below.



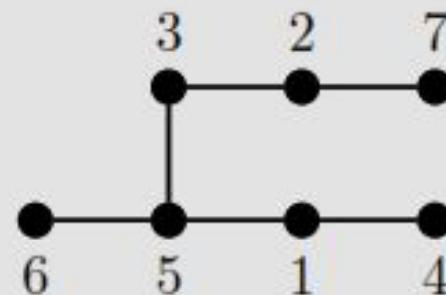
Repeating this process, we see that our subsequence is $(5, 3, 2)$ and $L = \{6, 7\}$. Thus 6 is the leaf adjacent to 5. See the graph below.



Our next subsequence is $(3, 2)$ and $L = \{5, 7\}$, meaning 5 is the smallest value not appearing in the sequence that hasn't already been placed as a “leaf,” so there must be an edge from 5 to 3.



Finally we are left with the single entry (2). We know that $L = \{3, 7\}$ and both of these must be adjacent to 2.



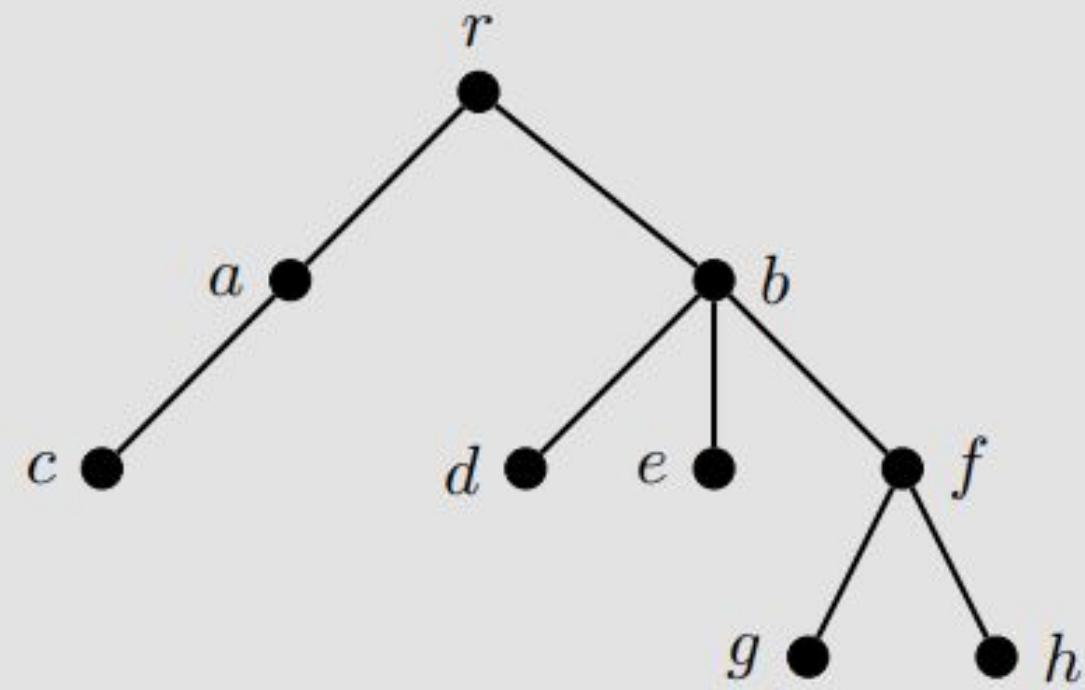
To verify we have the correct tree, we can use the process shown in Example 3.4 above to find the Prüfer sequence of our final tree and verify it matches the one given.

Theorem 3.14 (Cayley's Theorem) There are n^{n-2} different labeled trees on n vertices.

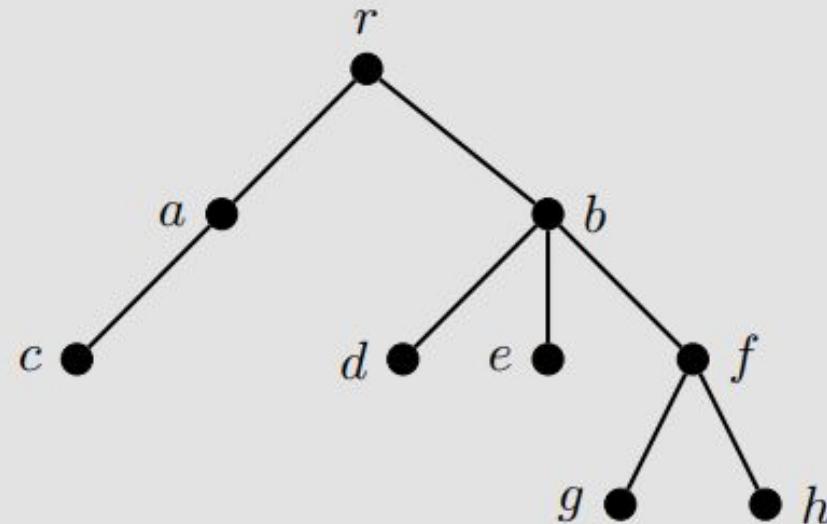
Proof: First, note that there are n^{n-2} possible sequences of length $n - 2$ since each spot on the sequence has n options. Each of these sequences can be viewed as a Prüfer sequence and will uniquely determine a tree as shown above. Thus there are n^{n-2} different labeled trees on n vertices.

Definition 3.15 A *rooted tree* is a tree T with a special designated vertex r , called the *root*. The *level* of any vertex in T is defined as the length of its shortest path to r . The *height* of a rooted tree is the largest level for any vertex in T .

Example 3.6 Find the level of each vertex and the height of the rooted tree shown below.



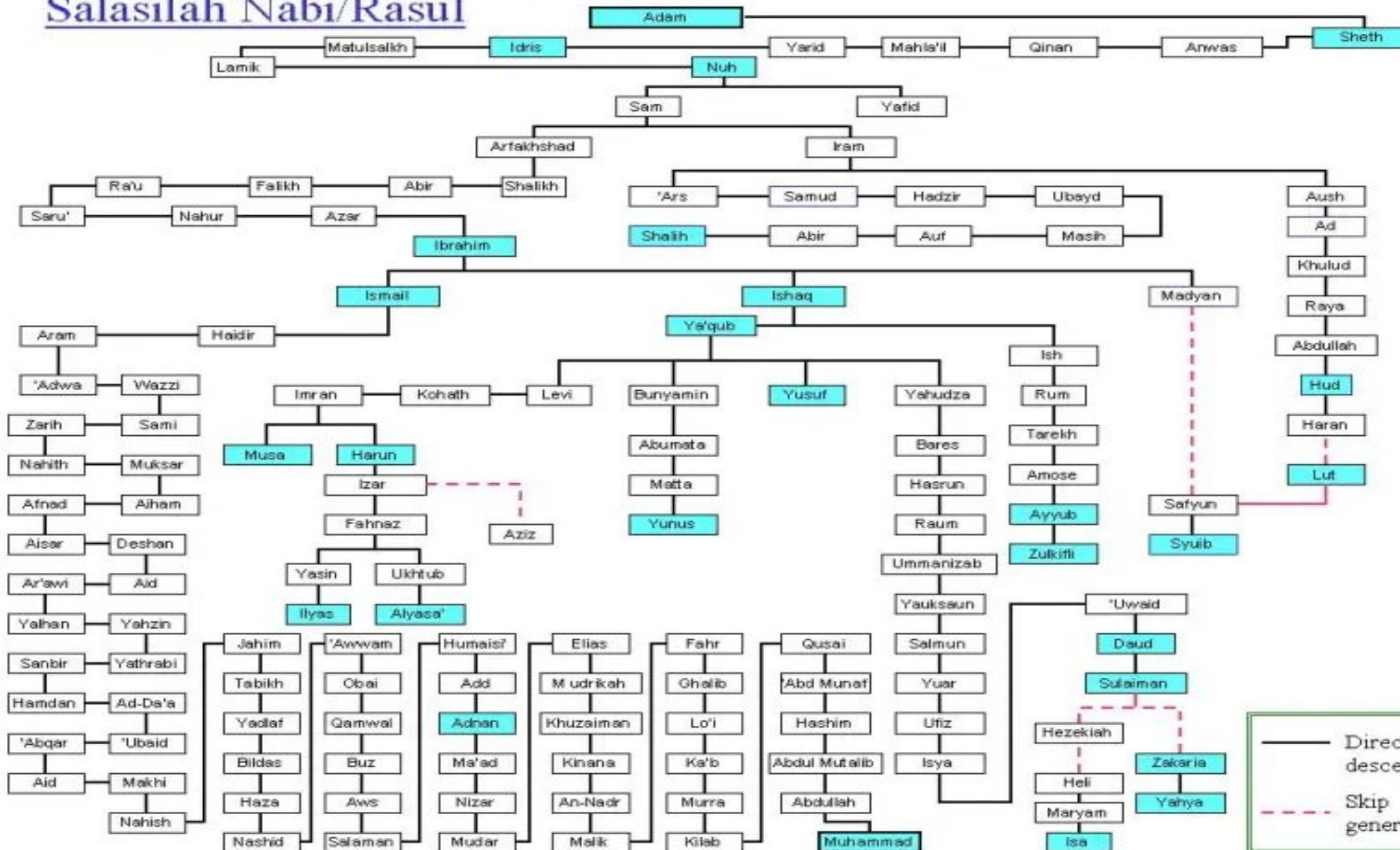
Example 3.6 Find the level of each vertex and the height of the rooted tree shown below.



Solution: Vertices a and b are of level 1, c, d, e , and f of level 2, and g and h of level 3. The root r has level 0. The height of the tree is 3.

Just Example. Not verified

Salasilah Nabi/Rasul



Definition 3.16 Let T be a tree with root r . Then for any vertices x and y

- x is a ***descendant*** of y if y is on the unique path from x to r ;
- x is a ***child*** of y if x is a descendant of y and exactly one level below y ;
- x is an ***ancestor*** of y if x is on the unique path from y to r ;
- x is a ***parent*** of y if x is an ancestor of y and exactly one level above y ;
- x is a ***sibling*** of y if x and y have the same parent.

Definition 3.17 A tree in which every vertex has at most two children is called a *binary tree*. If every parent has exactly two children we have a *full binary tree*. Similarly, if every vertex has at most k children then the tree is called a *k -nary tree*.

Example 3.7 Trees can be used to store information for quick access. Consider the following string of numbers:

4, 2, 7, 10, 1, 3, 5

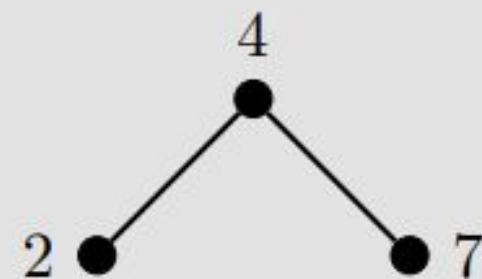
We can form a tree by creating a vertex for each number in the list. As we move from one entry in the list to the next, we place an item below and to the left if it is less than the previously viewed vertex and below and to the right if it is greater. If we add the restriction that no vertex can have more than two edges coming down from it, then we are forming a binary tree.

4, 2, 7, 10, 1, 3, 5

For the string above, we start with a tree consisting of one vertex, labeled 4 (see T_1). The next item in the list is a 2, which is less than 4 and so its vertex is placed on the left and below the vertex for 4. The next item, 7, is larger than 4 and so its vertex is placed on the right and below the vertex for 4 (see T_2).



T_1

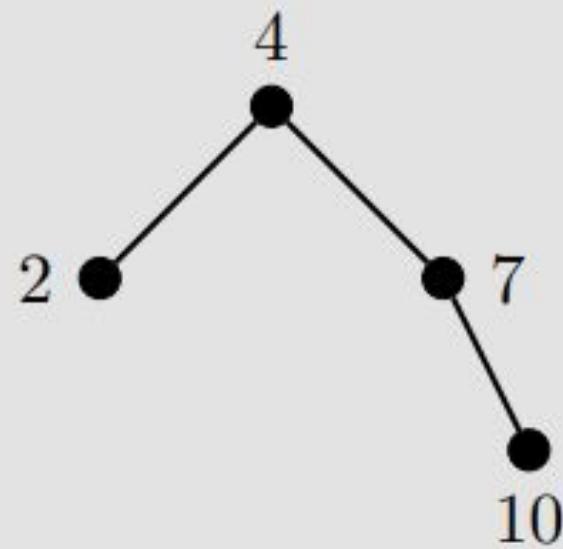


T_2

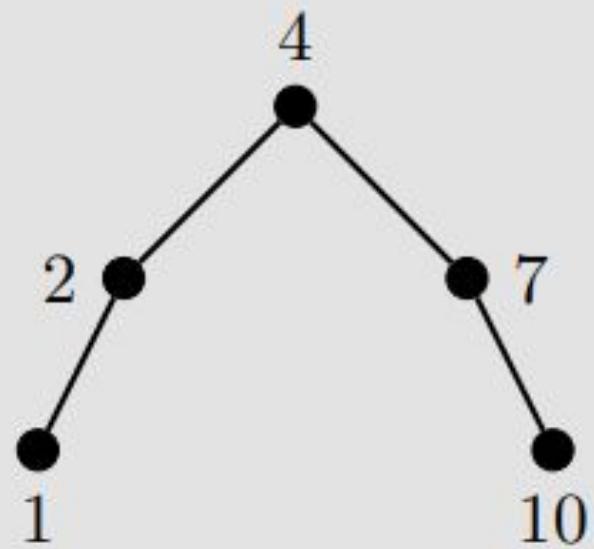
4, 2, 7, 10, 1, 3, 5

The next item in the list is 10. Since 4 already has two edges below it, we must attach the vertex for 10 to either 2 or 7. Since 10 is greater than 4, it must be placed to the right of 4 and since 10 is greater than 7, it must be placed to the right of 7 (see T_3). A similar reasoning places 1 to the left and below 2 (see T_4).

4, 2, 7, 10, 1, 3, 5



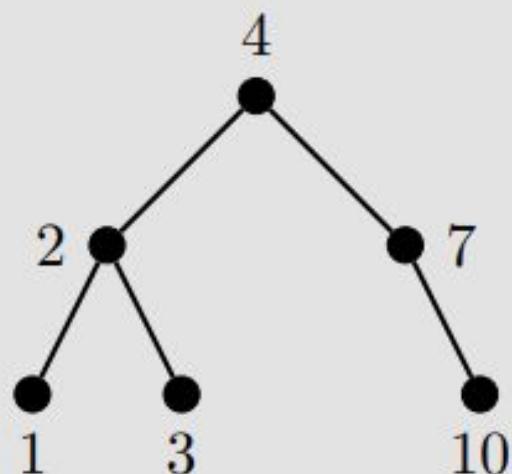
T_3



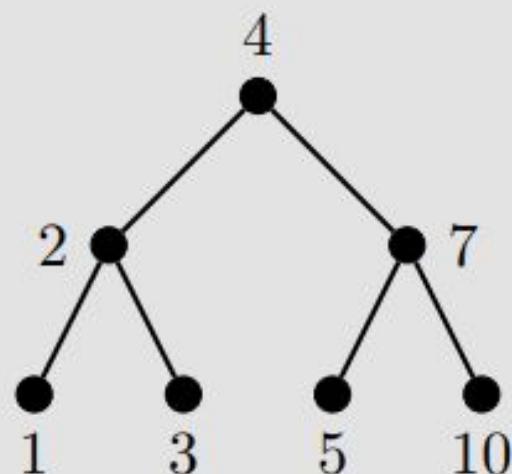
T_4

4, 2, 7, 10, 1, 3, 5

The next item is 3, which is less than 4 and so must be to the left of 4. Since 3 is greater than 2, it must be placed to the right of 2 (see T_5). The final item is 5, which is greater than 4 but less than 7, placing it to the right of 4 but to the left of 7 (see T_6).



T_5



T_6

Theorem 3.18 Let T be a binary tree with height h and l leaves. Then

- (i) $l \leq 2^h$.
- (ii) if T is a full binary tree and all leaves are at height h , then $l = 2^h$.
- (iii) if T is a full binary tree, then $n = 2l - 1$.

Depth-First Search Tree

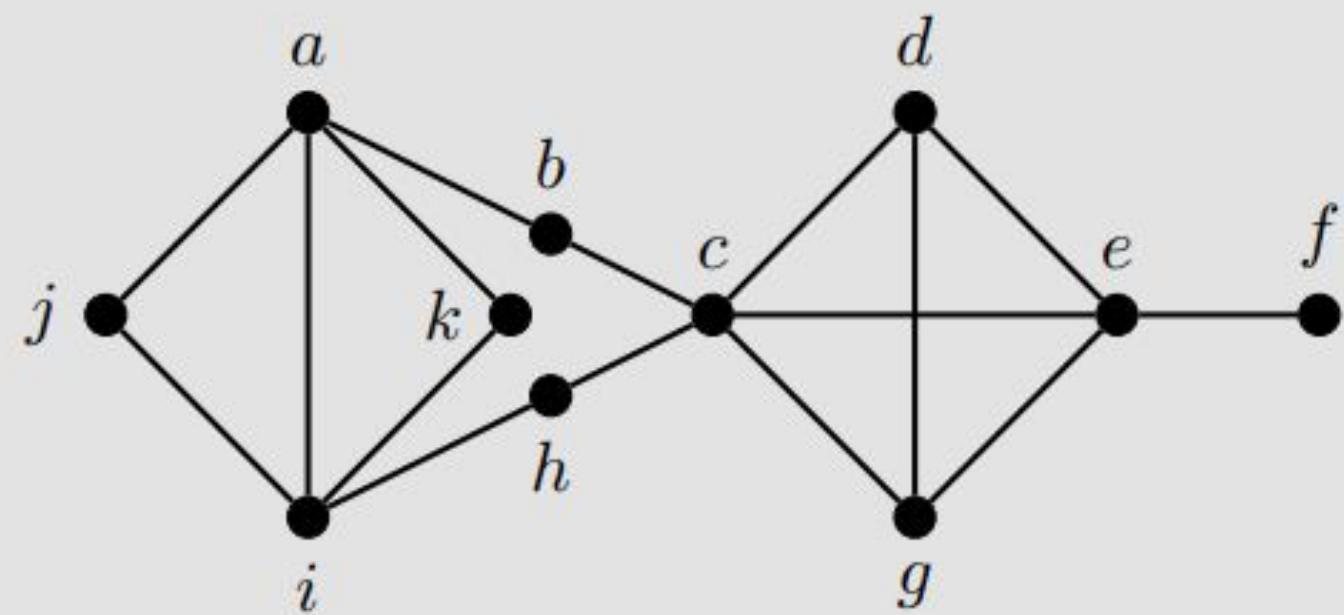
Input: Simple graph $G = (V, E)$ and a designated root vertex r .

Steps:

1. Choose the first neighbor x of r in G and add it to $T = (V, E')$.
2. Choose the first neighbor of x and add it to T . Continue in this fashion—picking the first neighbor of the previous vertex to create a path P . If P contains all the vertices of G , then P is the depth-first search tree. Otherwise continue to Step (3).
3. Backtrack along P until the first vertex is found that has neighbors not in T . Use this as the root and return to Step (1).

Output: Depth-first search tree T .

Example 3.8 Find the depth-first search tree for the graph below with the root a .

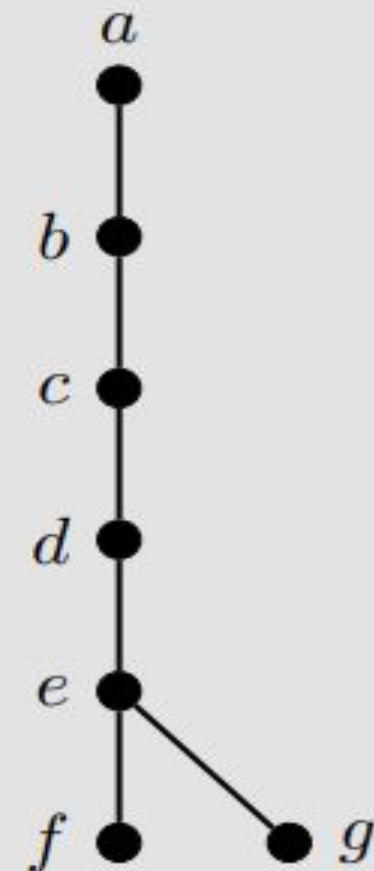


Solution:

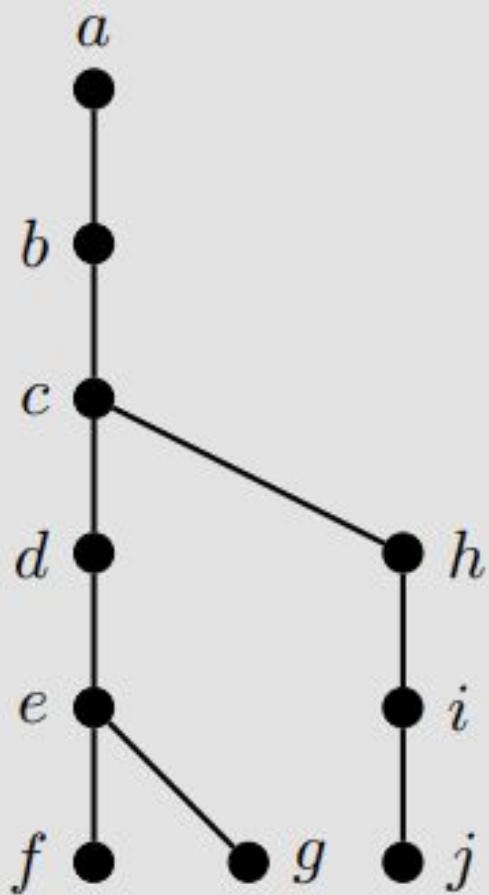
Step 1: Since a is the root, we add b as it is the first neighbor of a . Continuing in this manner produces the path shown below. Note this path stops with f since f has no further neighbors in G .



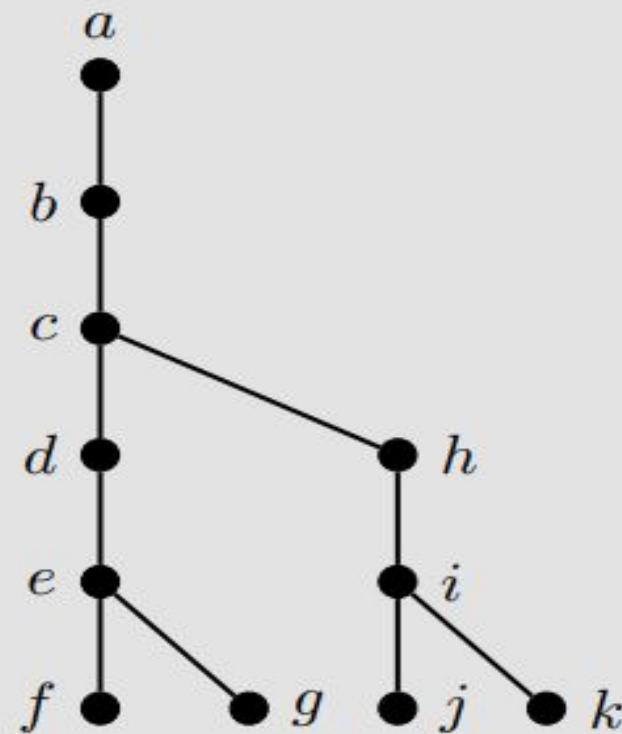
Step 2: Backtracking along the path above, the first vertex with an unchosen neighbor is e . This adds the edge eg to T . No other edges from g are added since the other neighbors of g are already part of the tree.



Step 3: Backtracking again along the path from Step 1, the next vertex with an unchosen neighbor is c . This adds the path $chij$ to T .



Step 4: Backtracking again along the path from Step 3, the next vertex with an unchosen neighbor is i . This adds the edge ik to T and completes the depth-first search tree as all the vertices of G are now included in T .



Output: The tree above is the depth-first search tree.

Breadth-First Search Tree

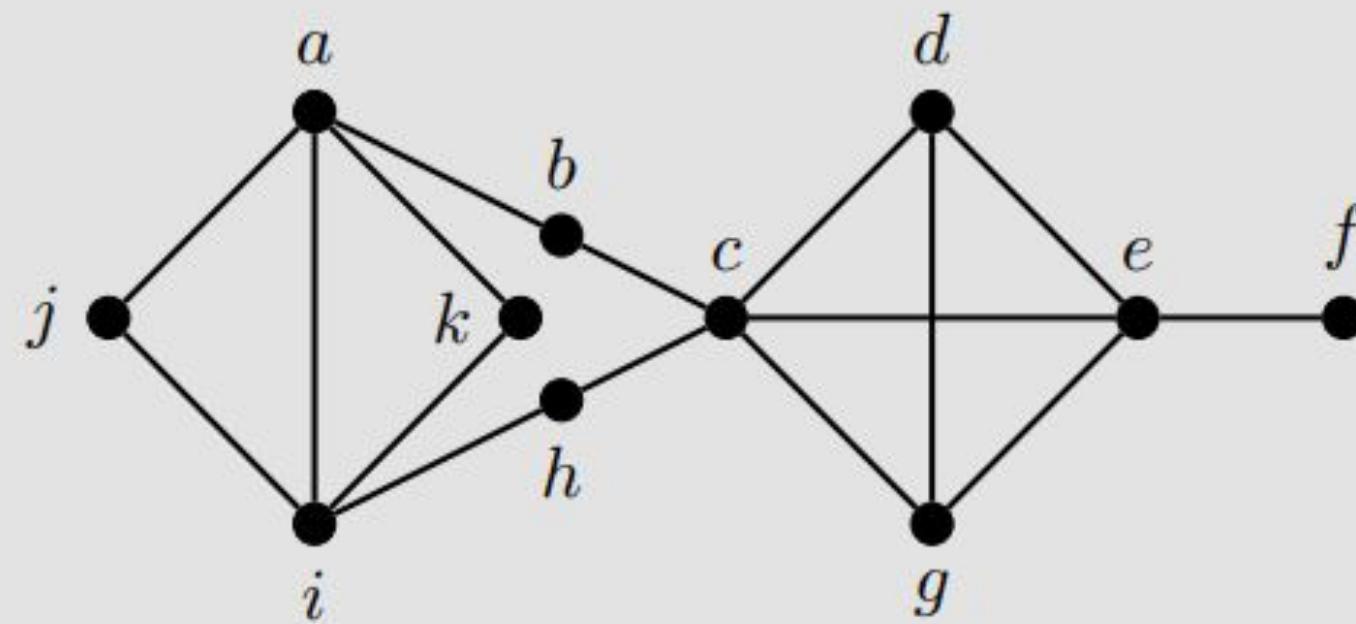
Input: Simple graph $G = (V, E)$ and a designated root vertex r .

Steps:

1. Add all the neighbors of r in G to $T = (V, E')$.
2. If T contains all the vertices of G , then we are done. Otherwise continue to Step (3).
3. Beginning with x , the first neighbor of r that has neighbors not in T , add all the neighbors of x to T . Repeat this for all the neighbors of r .
4. If T contains all the vertices of G , then we are done. Otherwise repeat Step (3) with the vertices just previously added to T .

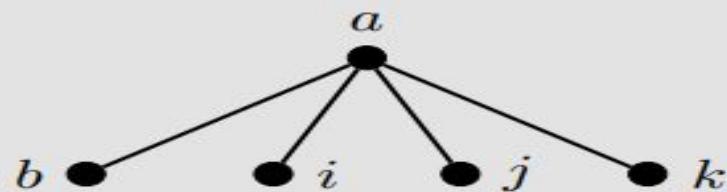
Output: Breadth-first tree T .

Example 3.9 Find the breadth-first search tree for the graph below with the root *a*.

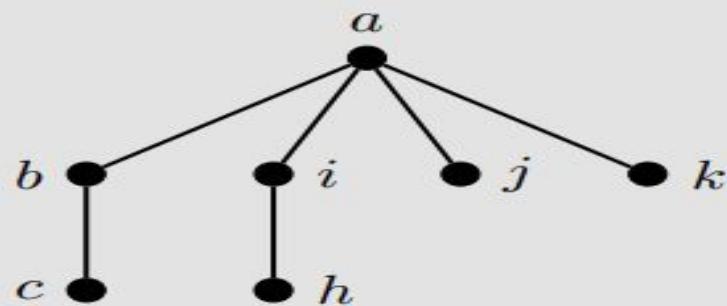


Solution:

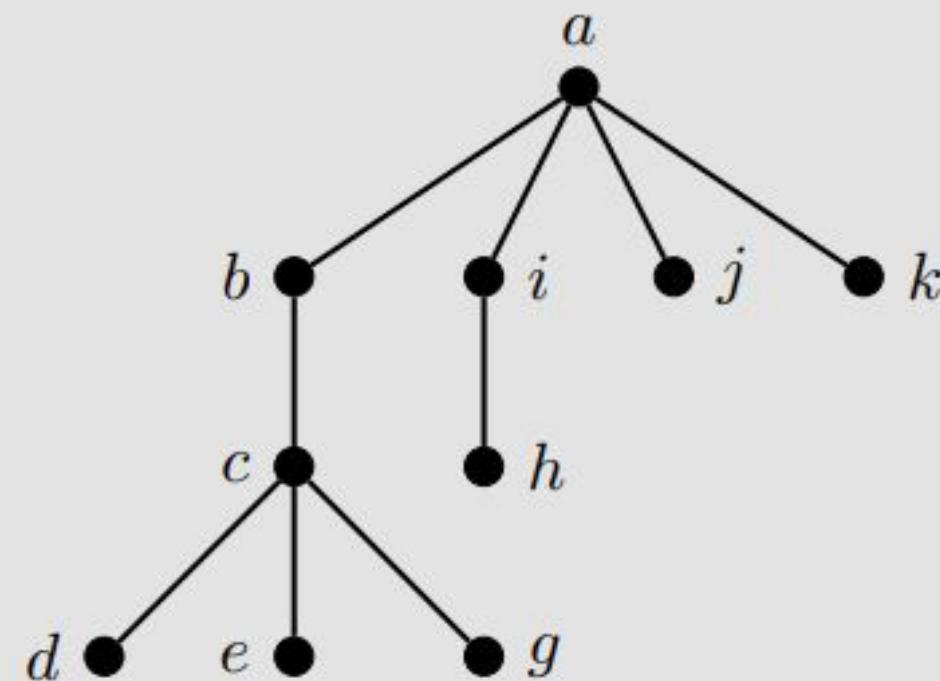
Step 1: Since a is the root, we add all of the neighbors of a to T .



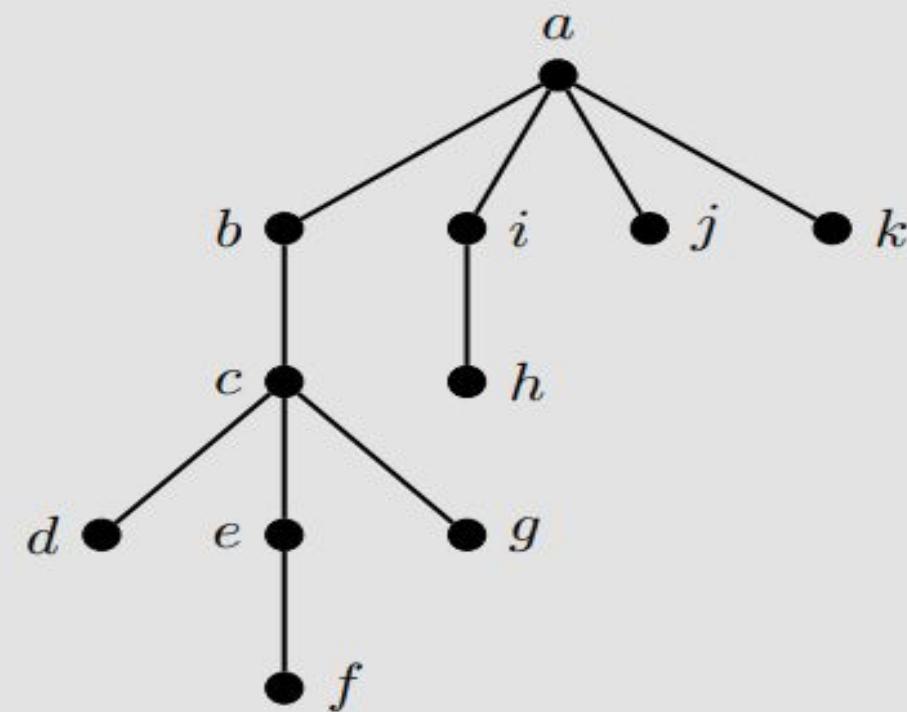
Step 2: We next add all the neighbors of b, i, j , and k , that are not already in T , beginning with b as it is the first neighbor of a that was added in Step 1. This adds the edge bc . Moving to i we add the edge ih . No other edges are added since j and k do not have any unchosen neighbors.



Step 3: We next add all the neighbors of c not in T , namely d, e , and g . No other vertices are added since all the neighbors of h are already part of the tree T .



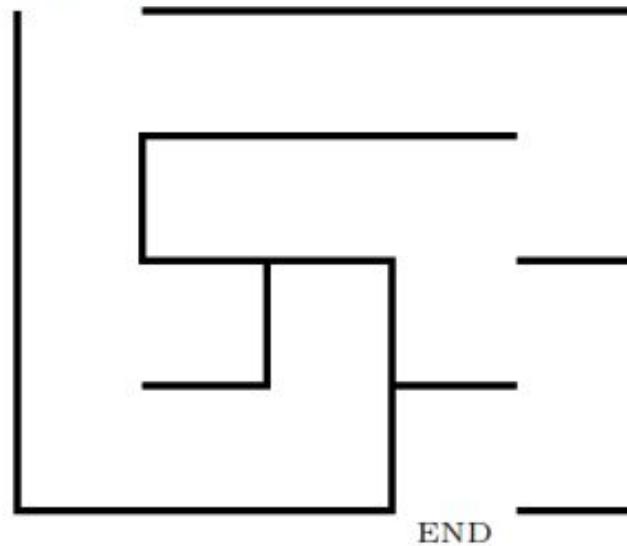
Step 4: Since d has no unchosen neighbors, we move ahead to adding the unchosen neighbors of e . This completes the breadth-first search tree as all the vertices of G are now included in T .



Output: The tree above is the breadth-first search tree.

Mazes

START



v_1

v_2

v_9

v_{10}

v_8

v_3

v_7

v_4

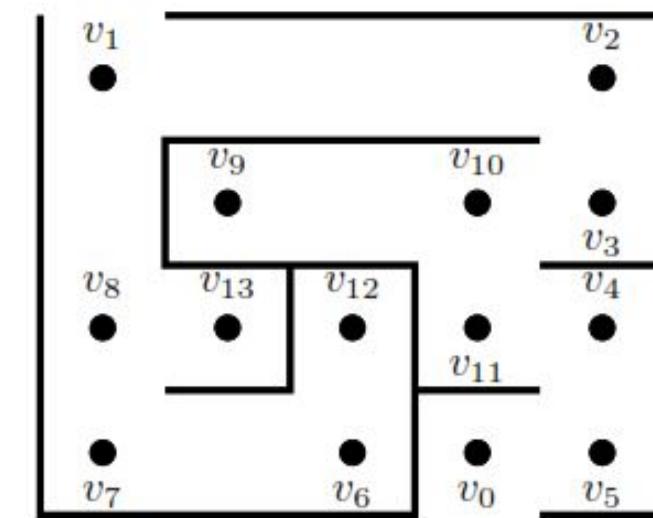
v_{13}

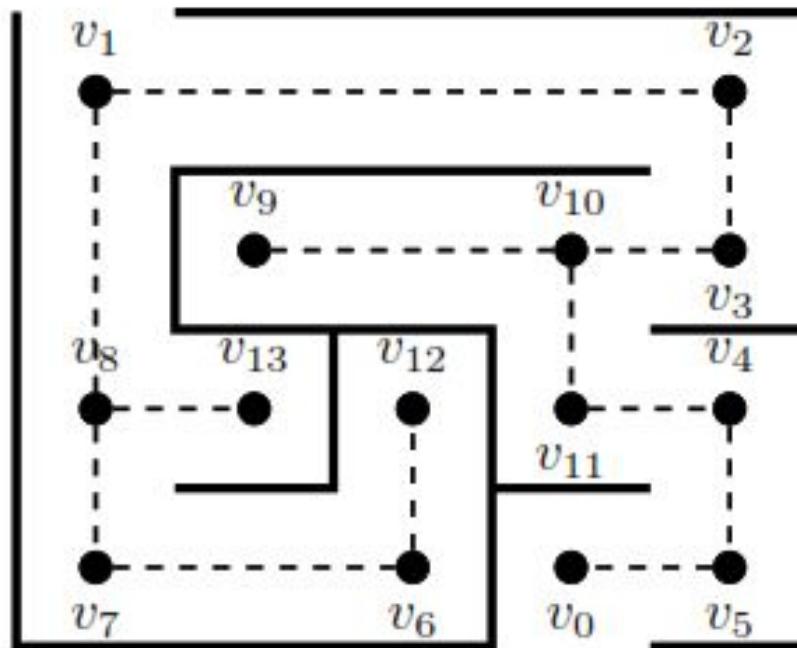
v_{11}

v_6

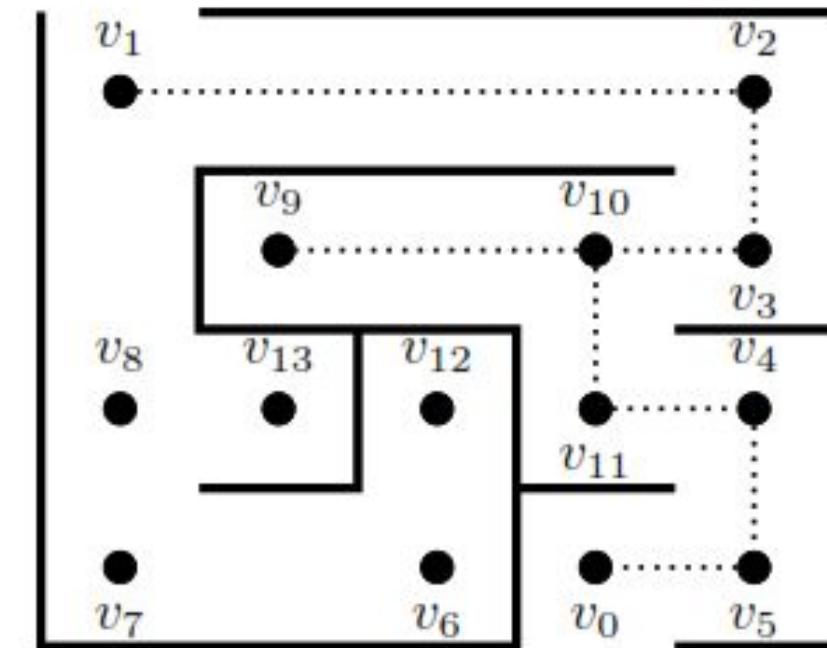
v_0

v_5

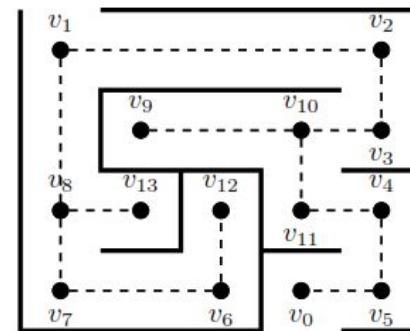




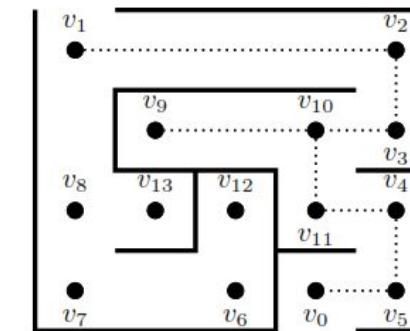
Breadth-First Search



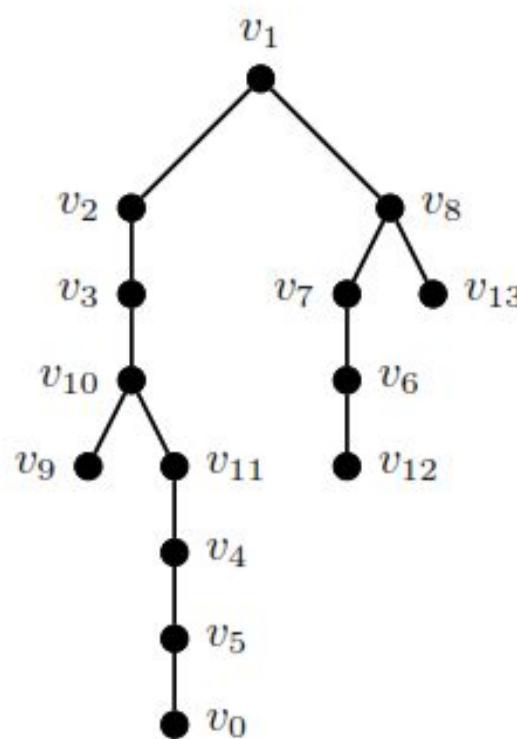
Depth-First Search



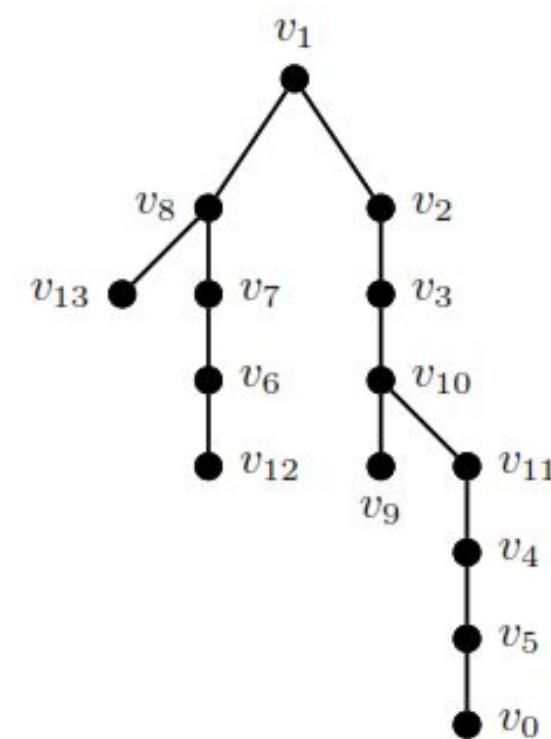
Breadth-First Search



Depth-First Search



Maze Breadth-First Tree



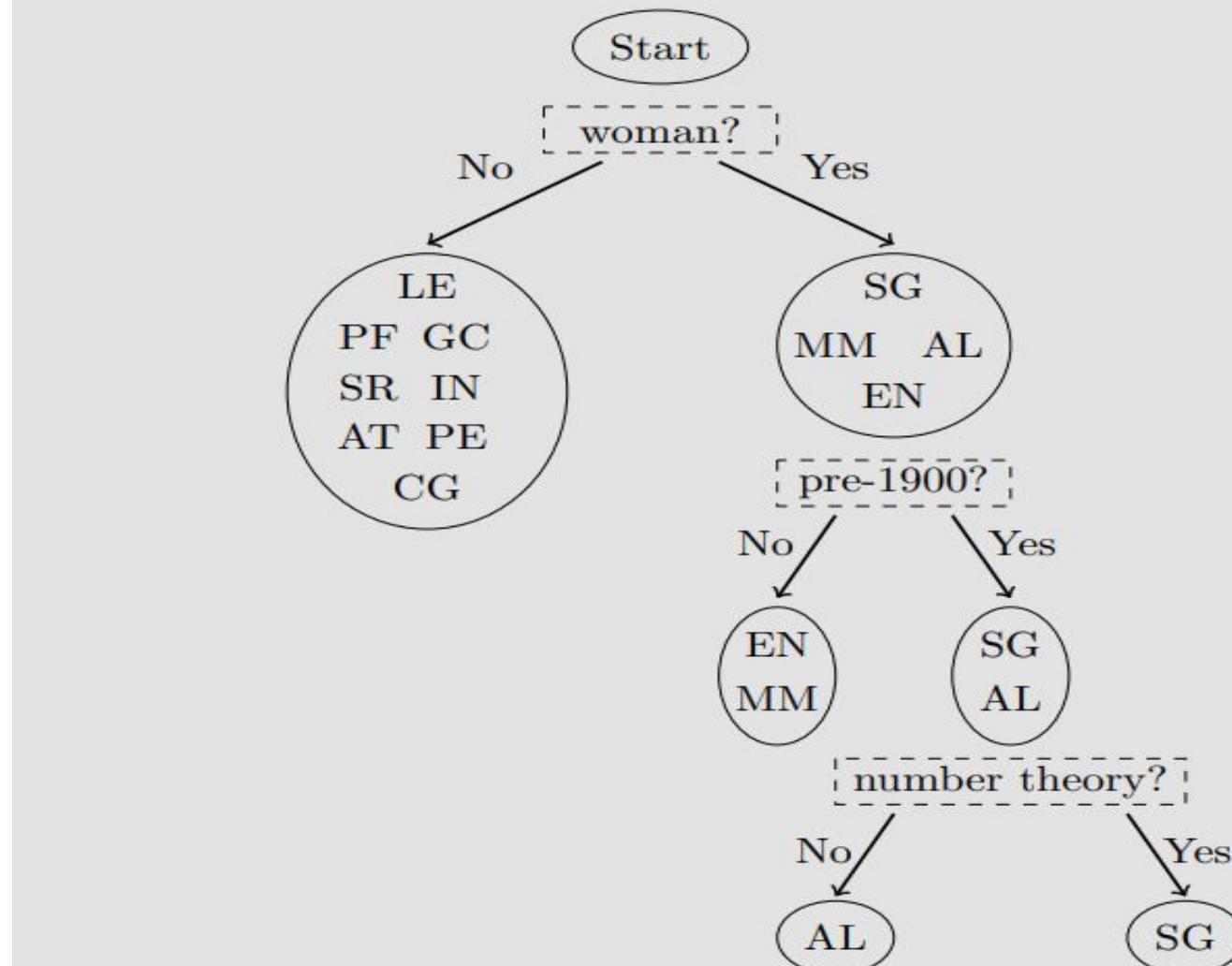
Maze Depth-First Tree

Decision Trees

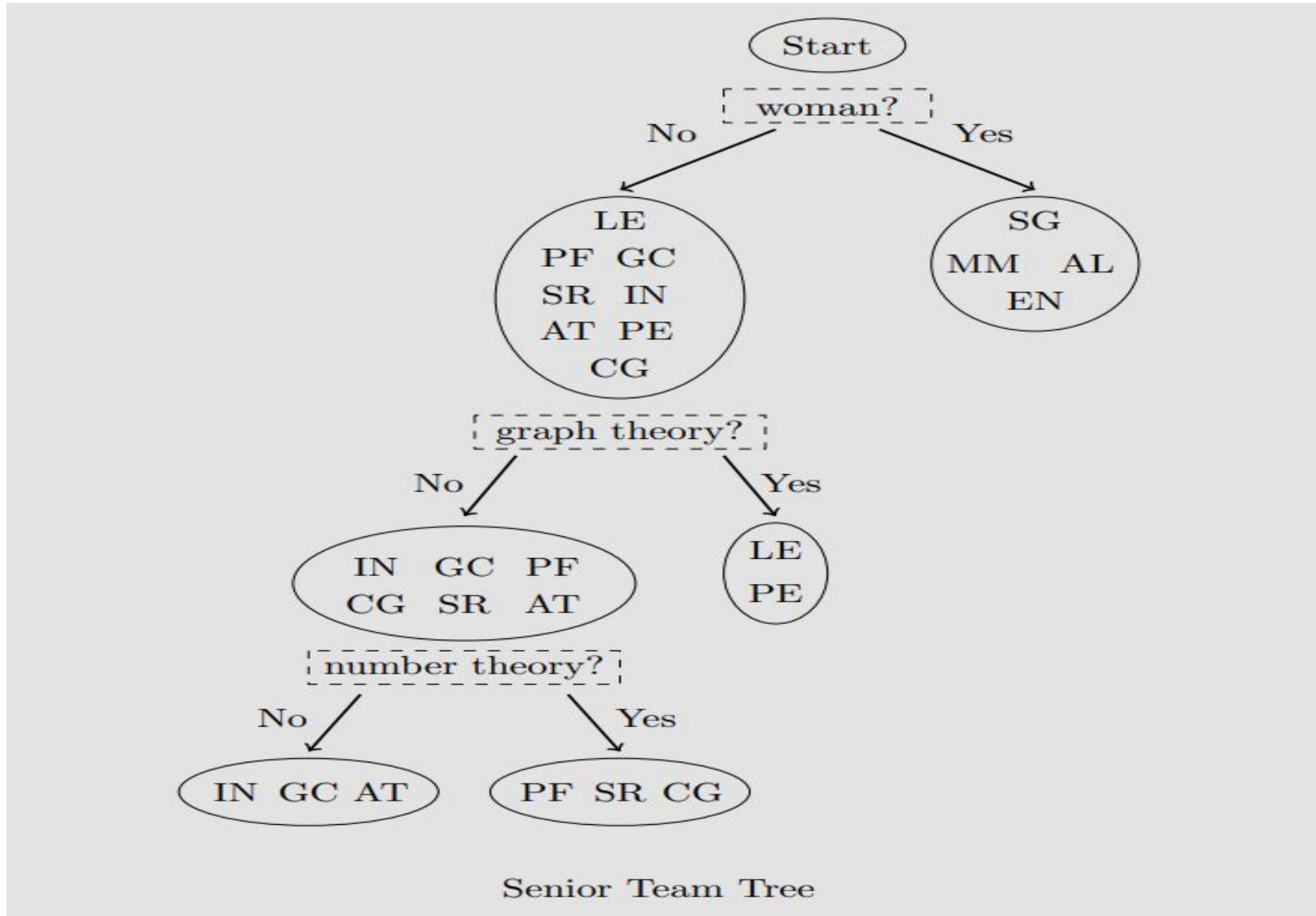
Example 3.11 The junior and senior math majors are facing off on a Mathematician Guess Who game. Below is a list of the cards. The junior team picked Pierre de Fermat and asked the questions (in order) “Is your person a woman? From pre-1900’s? Known for Number Theory?” The senior team picked Sophie Germain and asked “Is your person a woman? Known for Graph Theory? Known for Number Theory?” Use a decision tree to determine which team won.

Pierre de Fermat (PF)	Carl Gauss (CG)	Leonhard Euler (LE)
Srinivasa Ramanujan (SR)	Alan Turing (AT)	Emmy Noether (EN)
Maryam Mirzakhani (MM)	Georg Cantor (GC)	Isaac Newton (IN)
Sophie Germain (SG)	Paul Erdős (PE)	Ada Lovelace (AL)

Solution: Below are two trees representing the sequence of questions asked by each team. Since only the junior team has a single person in their final set, we know they won the game after 3 questions.



Junior Team Tree

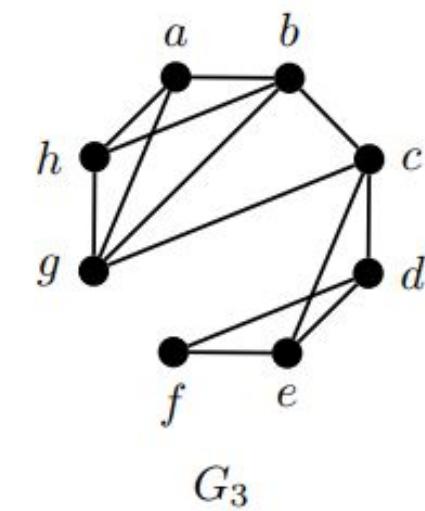
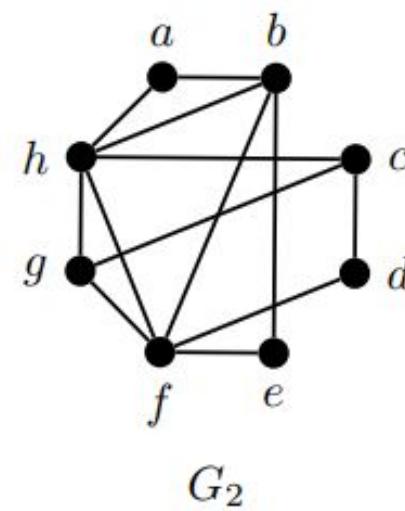
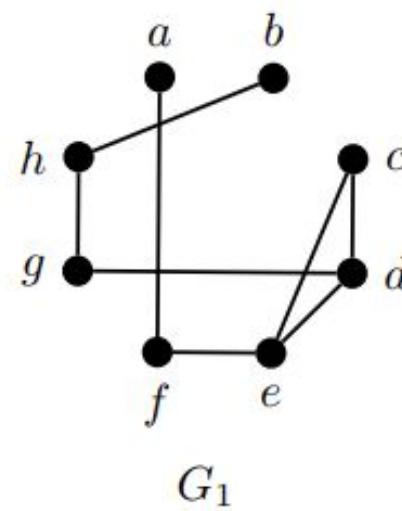


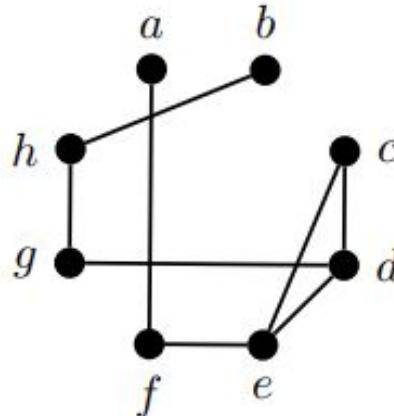
EX #: 3.5

Problems: 3.1-3.8, 3.13-3.17, 3.23

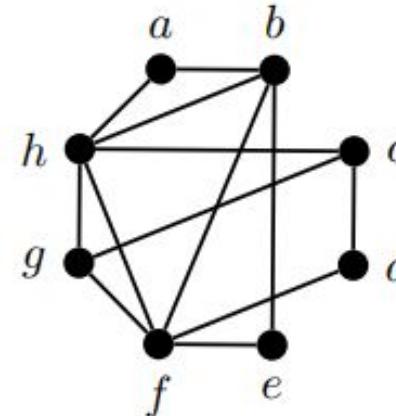
Connectivity Measures

Definition 4.1 A ***cut-vertex*** of a graph G is a vertex v whose removal disconnects the graph, that is, G is connected but $G - v$ is not. A set S of vertices within a graph G is a ***cut-set*** if $G - S$ is disconnected.

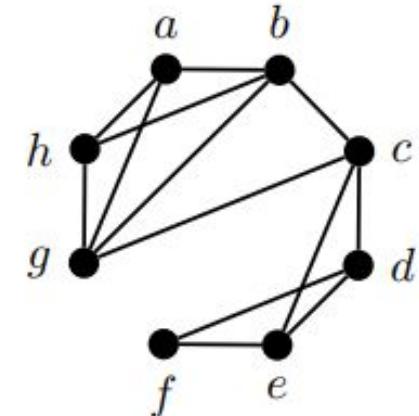




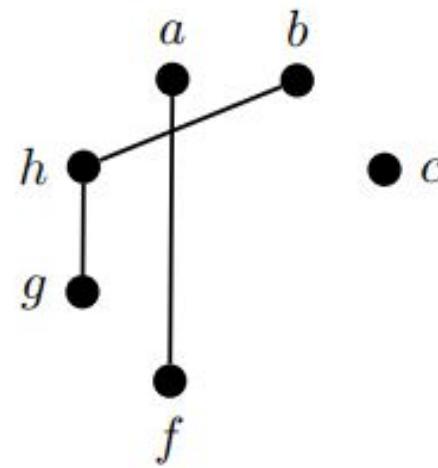
G_1



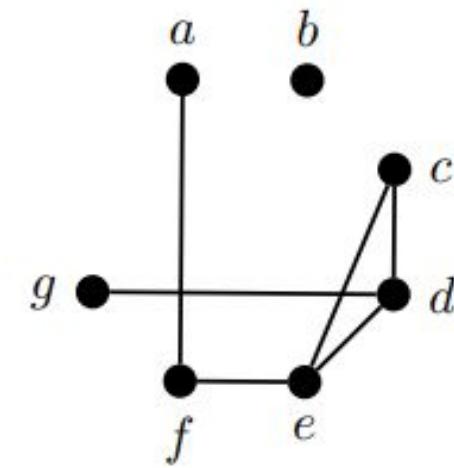
G_2



G_3



$G_1 - \{d, e\}$

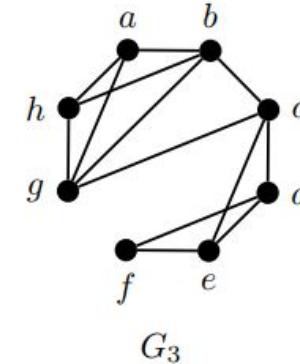
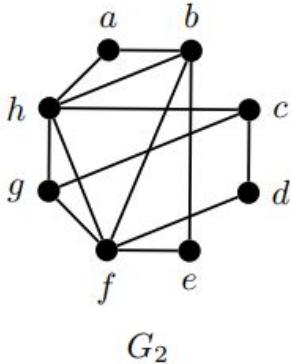
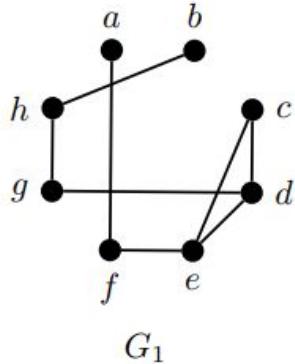


$G_1 - \{h\}$

4.1.1 k -Connected

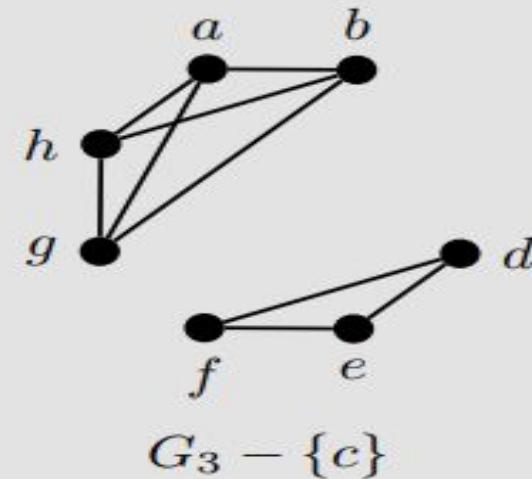
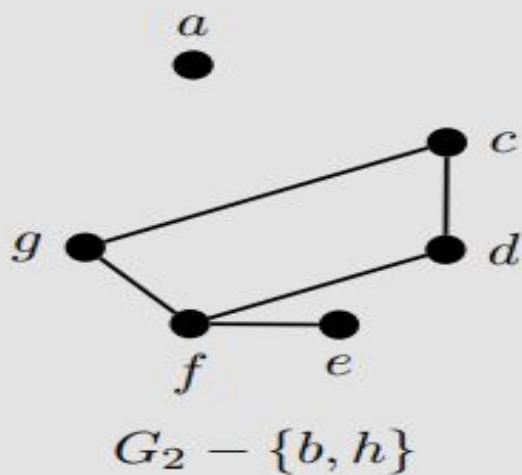
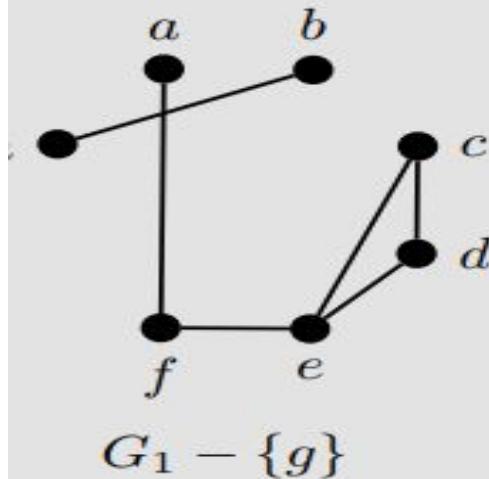
Definition 4.2 For any graph G , we say G is **k -connected** if the smallest cut-set is of size at least k .

Define the connectivity of G , $\kappa(G) = k$, to be the maximum k such that G is k -connected, that is there is a cut-set S of size k , yet no cut-set exists of size $k - 1$ or less. Define $\kappa(K_n) = n - 1$.



Example 4.1 Find $\kappa(G)$ for each of the graphs shown above on page 169.

Solution: The removal of any one of d, e, f, g , or h in G_1 will disconnect the graph, so $\kappa(G_1) = 1$. Similarly, $G_3 - c$ has two components and so $\kappa(G_3) = 1$. However, $\kappa(G_2) = 2$ since the removal of any one vertex will not disconnect the graph, yet $S = \{b, h\}$ is a cut-set. Note this means G_2 is both 1-connected and 2-connected, but not 3-connected.



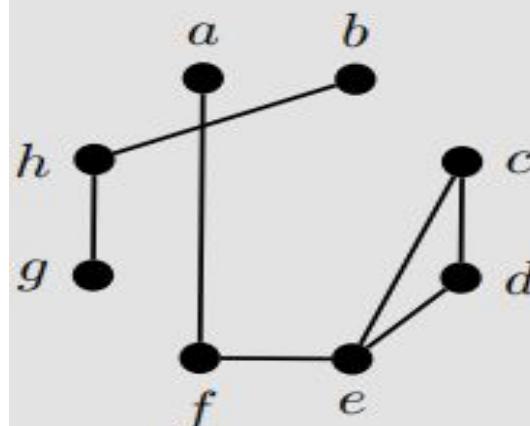
Definition 4.3 A **bridge** in a graph $G = (V, E)$ is an edge e whose removal disconnects the graph, that is, G is connected but $G - e$ is not. An **edge-cut** is a set $F \subseteq E$ so that $G - F$ is disconnected.

Definition 4.4 We say G is **k -edge-connected** if the smallest edge-cut is of size at least k .

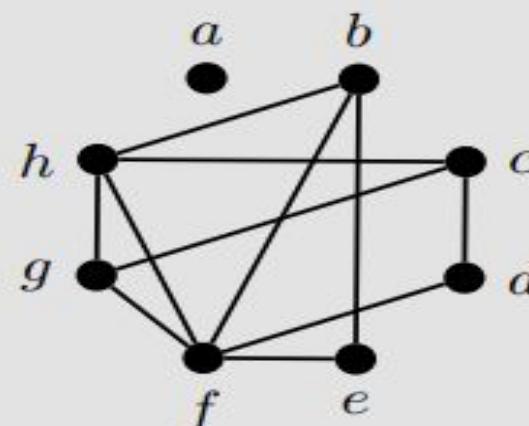
Define $\kappa'(G) = k$ to be the maximum k such that G is k -edge-connected, that is there exists a edge-cut F of size k , yet no edge-cut exists of size $k - 1$.

Example 4.2 Find $\kappa'(G)$ for each of the graphs shown on page 169.

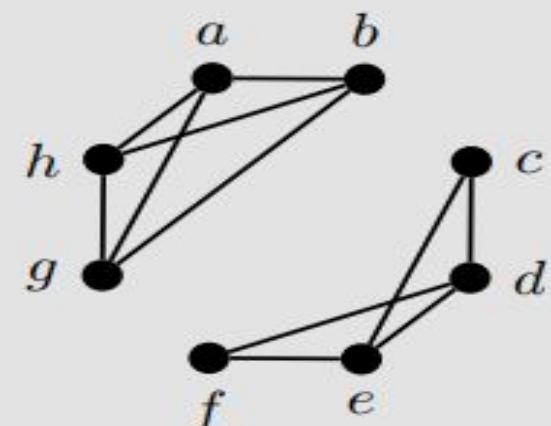
Solution: There are many options for a single edge whose removal will disconnect G_1 (for example af or dg). Thus $\kappa'(G_1) = 1$. For G_2 , no one edge can disconnect the graph with its removal, yet removing both ab and ah will isolate a and so $\kappa(G_2) = 2$. Similarly $\kappa'(G_3) = 2$, since the removal of bc and cg will create two components, one with vertices a, b, g, h and the other with c, d, e, f .



G₁ - dg



$$G_2 - \{ab, ah\}$$



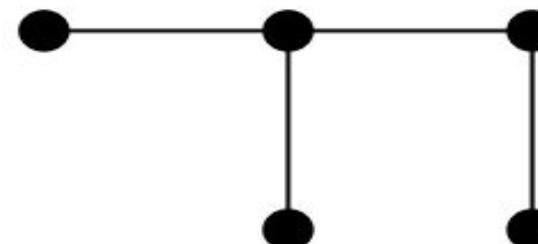
$$G_3 - \{bc, cg\}$$

Vertex-connectivity

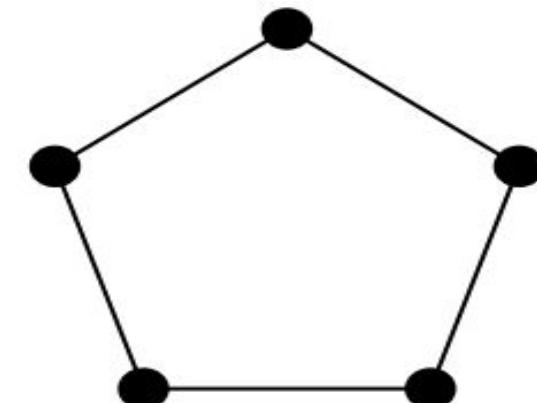
The **vertex-connectivity** (or simply the **connectivity**) of G , denoted by $\kappa(G)$, is defined by

$$\kappa(G) = \begin{cases} n - 1, & \text{if } G \cong K_n; \\ \min\{|S| \mid S \text{ is a cut of } G\}, & \text{if } G \not\cong K_n. \end{cases}$$

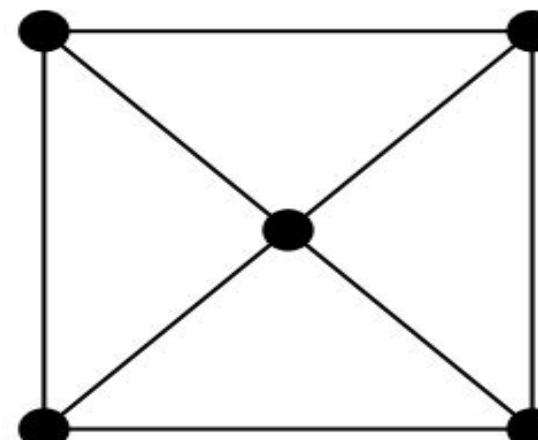
In other words, $\kappa(G)$ is the minimum number of vertices in G whose removal results in either a disconnected graph or a trivial graph (a singleton).



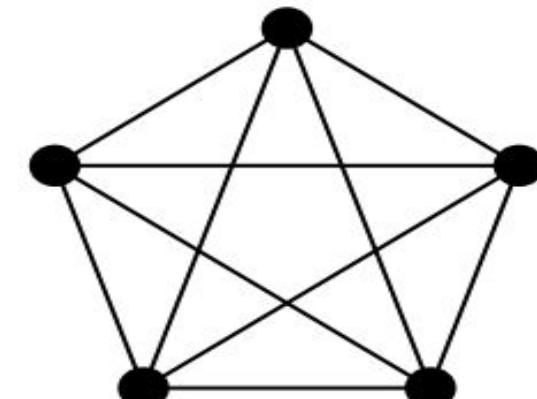
(a) Tree



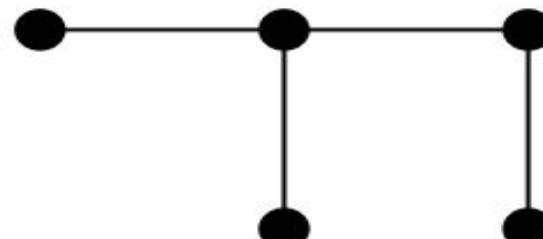
(b) Cycle C_5



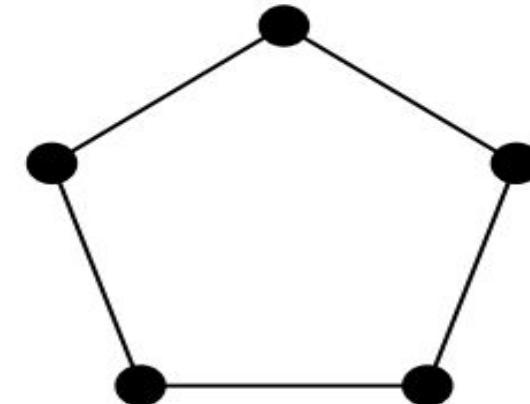
(c) Wheel W_5



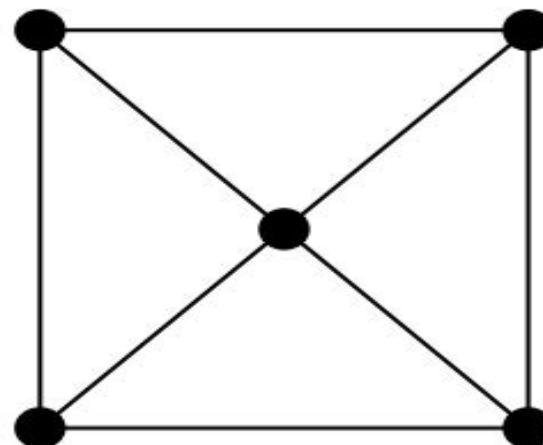
(d) Complete graph K_5



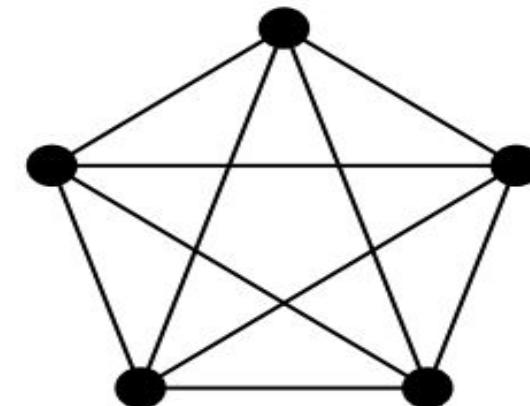
(a) Tree



(b) Cycle C_5



(c) Wheel W_5



(d) Complete graph K_5

- (a) $\kappa(G) = 1$, (b) $\kappa(C_5) = 2$, (c) $\kappa(W_5) = 3$, and (d) $\kappa(K_5) = 4$.

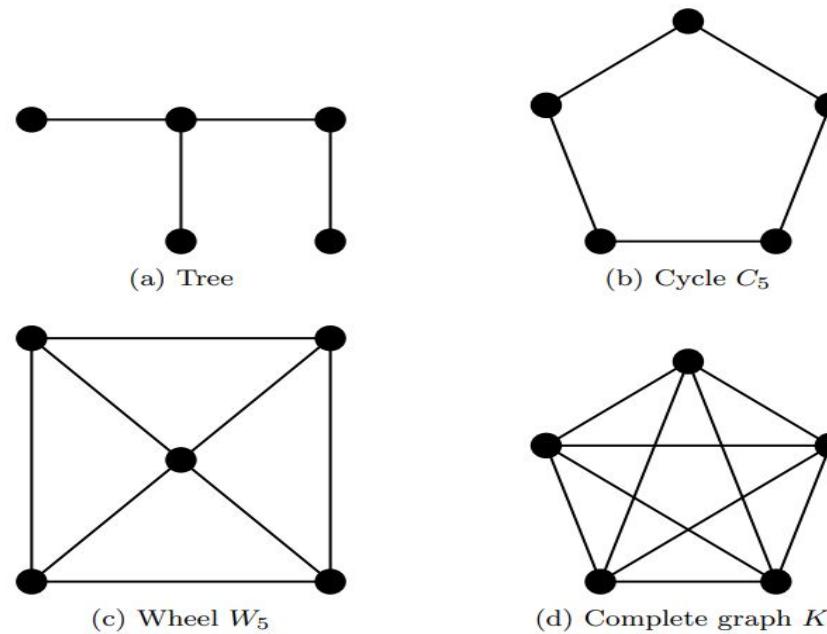
- (1) When we remove any r vertices ($1 \leq r \leq n - 1$) from the complete graph K_n , the resulting subgraph is never disconnected; it is a trivial graph if $r = n - 1$. By definition, $\kappa(K_n) = n - 1$.
- (2) We define, for convention, $\kappa(G) = 0$ if G is disconnected.

Edge-connectivity

A subset F of E is called an **edge-cut** of G if $G - F$ is disconnected. Thus, an edge f in G is a bridge of G if and only if the singleton $\{f\}$ is an edge-cut of G . The **edge-connectivity** of G , denoted by $\kappa'(G)$, is defined by

$$\kappa'(G) = \begin{cases} 0, & \text{if } v(G) = 1; \\ \min\{|F| \mid F \text{ is an edge-cut of } G\}, & \text{if } v(G) \geq 2. \end{cases}$$

That is, $\kappa'(K_1) = 0$, and if $n \geq 2$, then $\kappa'(G)$ is the minimum number of edges in G whose removal results in a disconnected graph. Any edge-cut F of G with $|F| = \kappa'(G)$ is called a **minimum** edge-cut of G . Likewise, we define $\kappa'(G) = 0$ if G is disconnected.



- (a) $\kappa'(G) = 1$;
 (b) $\kappa'(C_5) = 2$;

- (c) $\kappa'(W_5) = 3$;
 (d) $\kappa'(K_5) = 4$.

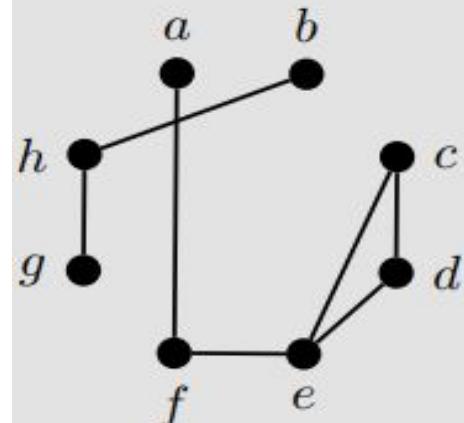
Lec # 20, 21 & 22

Example 6.4.3.

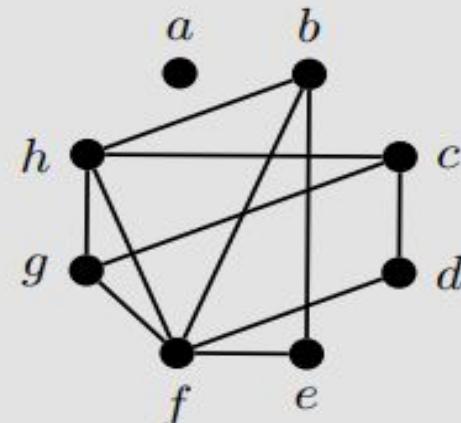
- (i) The graph K_2 or any other connected graph containing a cut-vertex is 1-connected but not 2-connected.
- (ii) Every cycle C_n , $n \geq 3$, is k -connected, for $k = 1, 2$, but not 3-connected.
- (iii) Every wheel W_n , $n \geq 4$, is k -connected, for $k = 1, 2, 3$, but not 4-connected.

Example 4.2 Find $\kappa'(G)$ for each of the graphs shown on page 169.

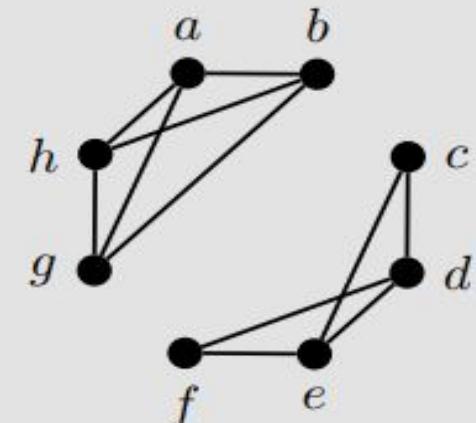
Solution: There are many options for a single edge whose removal will disconnect G_1 (for example af or dg). Thus $\kappa'(G_1) = 1$. For G_2 , no one edge can disconnect the graph with its removal, yet removing both ab and ah will isolate a and so $\kappa(G_2) = 2$. Similarly $\kappa'(G_3) = 2$, since the removal of bc and cg will create two components, one with vertices a, b, g, h and the other with c, d, e, f .



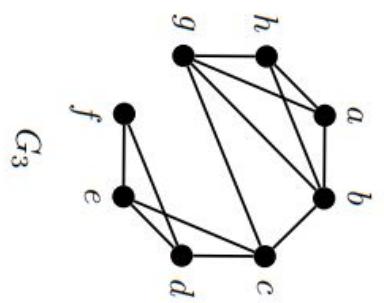
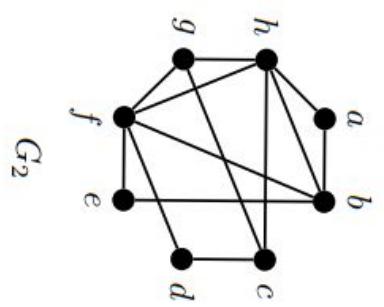
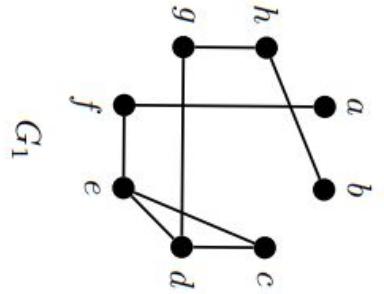
$G_1 - dg$



$G_2 - \{ab, ah\}$



$G_3 - \{bc, cg\}$



Theorem 4.5 (Whitney's Theorem) For any graph G , $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.

Theorem 6.3.1. (Whitney (1932)) *For any graph G of order $n \geq 1$,*
 $0 \leq \kappa(G) \leq \kappa'(G) \leq \delta(G) \leq n - 1$.

Connectivity & Paths

Theorem 4.6 A vertex v is a cut-vertex of a graph G if and only if there exist vertices x and y such that v is on every $x - y$ path.

Proof: First suppose v is a cut-vertex in a graph G . Then $G - v$ must have at least two components. Let x and y be vertices in different components of $G - v$. Since G is connected, we know there must exist an $x - y$ path in G that does not exist in $G - v$. Thus v must lie on this path.

Conversely, let v be a vertex and suppose there exist vertices x and y such that v is on every $x - y$ path. Then none of these paths exist in $G - v$, and so x and y cannot be in the same component of $G - v$. Thus G must have at least two components and so v is a cut-vertex.

Theorem 4.7 An edge e is a bridge of G if and only if there exist vertices x and y such that e is on every $x - y$ path.

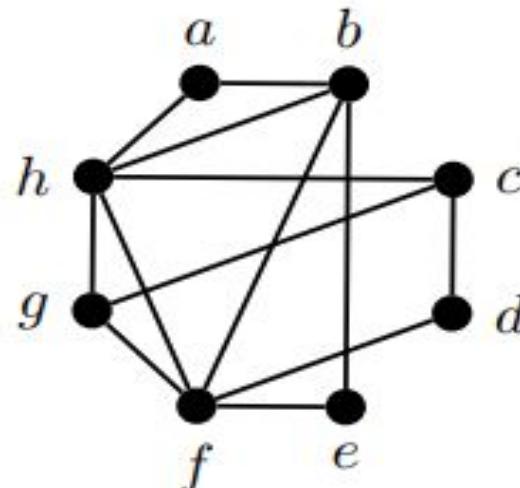
Theorem 4.8 Every nontrivial connected graph contains at least two vertices that are not cut-vertices.

Theorem 4.9 An edge e is a bridge of G if and only if e lies on no cycle of G .

Definition 4.10 Let P_1 and P_2 be two paths within the same graph G . We say these paths are

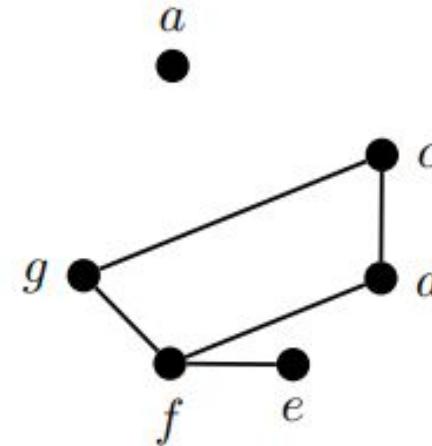
- ***disjoint*** if they have no vertices or edges in common.
- ***internally disjoint*** if the only vertices in common are the starting and ending vertices of the paths.
- ***edge-disjoint*** if they have no edges in common.

Definition 4.11 Let x and y be two vertices in a graph G . A set S (of either vertices or edges) ***separates*** x and y if x and y are in different components of $G - S$. When this happens, we say S is a separating set for x and y .

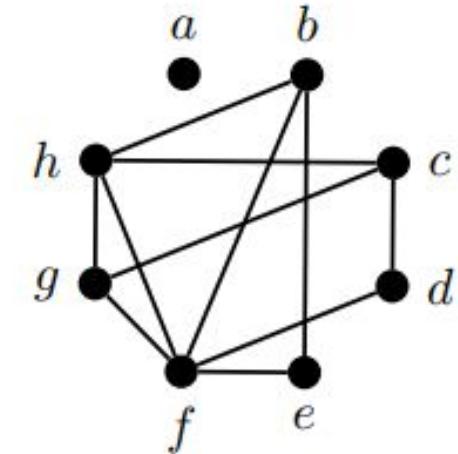


G_2

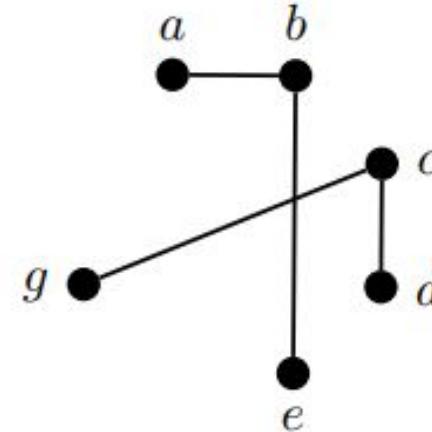
Note that a cut-set may or may not be a separating set for a specific pair of vertices. Consider the graph G_2 from page 169 (and reproduced on the next page). We have already shown that $\{b, h\}$ is a cut-set and $\{ab, ah\}$ is an edge-cut. But if we want to separate b and c then we cannot use b in the separating set, and using the edges ab and ah will only isolate a , leaving b and c in the same component. However, we can separate b and c using the vertices $\{f, h\}$ and the edges $\{cd, cg, ch\}$. Note that you cannot separate b and c with fewer vertices or edges (try it!).



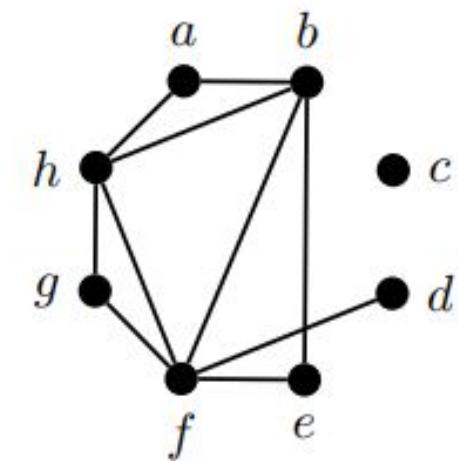
$$G_2 - \{b, h\}$$



$$G_2 - \{ab, ah\}$$



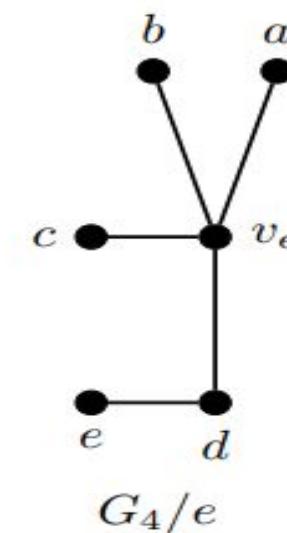
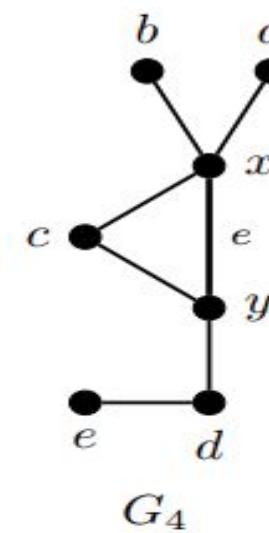
$$G_2 - \{f, h\}$$



$$G_2 - \{cd, cg, ch\}$$

Contraction

Definition 4.12 Let $e = xy$ be an edge of a graph G . The **contraction** of e , denoted G/e , replaces the edge e with a vertex v_e so that any vertices adjacent to either x or y are now adjacent to v_e .



Contracting an edge creates a smaller graph, both in terms of the number of vertices and edges, but keeps much of the structure of a graph in tact. In particular, contracting an edge cannot disconnect a graph (see Exercise 4.22).

Menger's Theorem

Theorem 4.13 (Menger's Theorem) Let x and y be nonadjacent vertices in G . Then the minimum number of vertices that separate x and y equals the maximum number of internally disjoint $x - y$ paths in G .

Theorem 4.14 A nontrivial graph G is k -connected if and only if for each pair of distinct vertices x and y there are at least k internally disjoint $x - y$ paths.

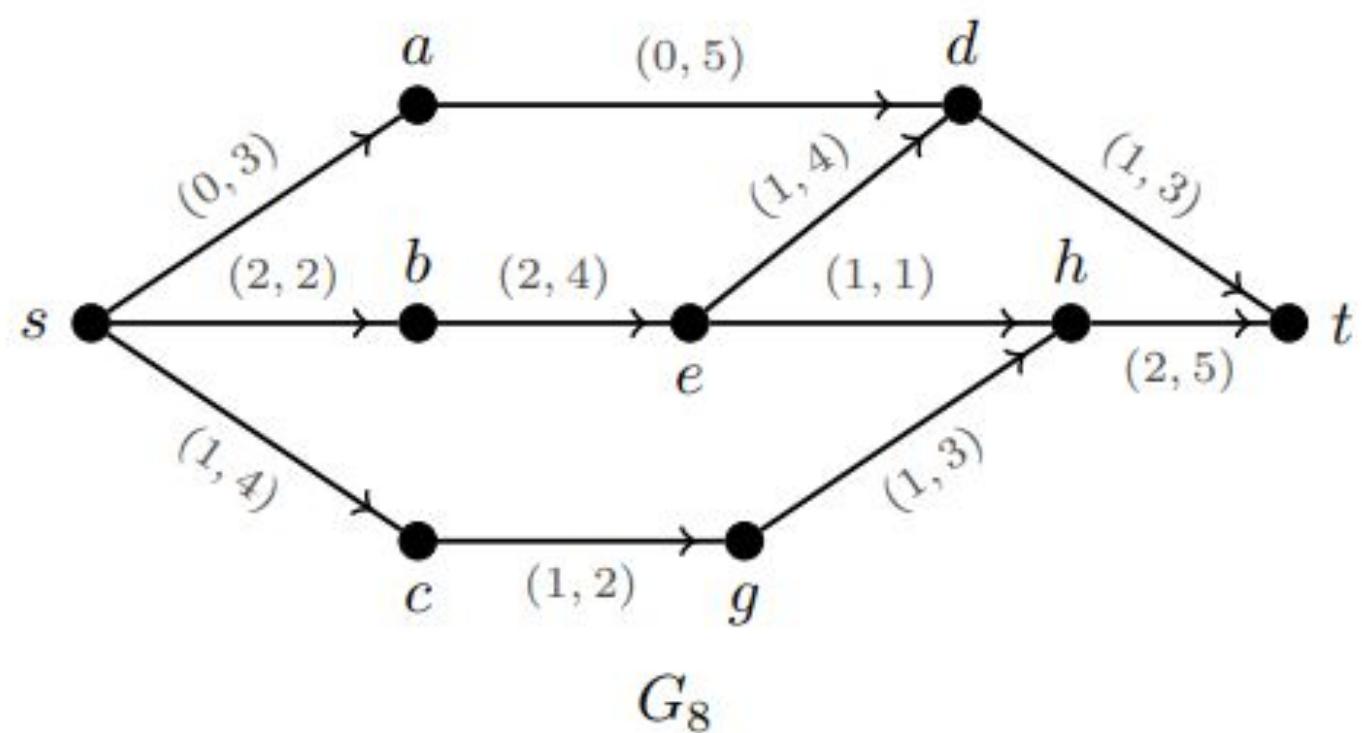
Theorem 4.15 Let x and y be distinct vertices in G . Then the minimum number of edges that separate x and y equals the maximum number of edge-disjoint $x - y$ paths in G .

Theorem 4.16 A nontrivial graph G is k -edge-connected if and only if for each pair of distinct vertices x and y there are at least k edge disjoint $x - y$ paths.

Network Flow

Definition 4.27 A *network* is a digraph where each arc e has an associated nonnegative integer $c(e)$, called a *capacity*. In addition, the network has a designated starting vertex s , called the *source*, and a designated ending vertex t , called the *sink*. A *flow* f is a function that assigns a value $f(e)$ to each arc of the network.

Below is an example of a network. Each arc is given a two-part label. The first component is the flow along the arc and the second component is the capacity.



Definition 4.28 For a vertex v , let $f^-(v)$ represent the total flow entering v and $f^+(v)$ represent the total flow exiting v . A flow is **feasible** if it satisfies the following conditions:

- (1) $f(e) \geq 0$ for all edges e
- (2) $f(e) \leq c(e)$ for all edges e
- (3) $f^+(v) = f^-(v)$ for all vertices other than s and t
- (4) $f^-(s) = f^+(t) = 0$

Definition 4.29 The *value* of a flow is defined as $|f| = f^+(s) = f^-(t)$, that is, the amount exiting the source which must also equal the flow entering the sink. A *maximum flow* is a feasible flow of largest value.

Definition 4.30 Let f be a flow along a network. The *slack* k of an arc is the difference between its capacity and flow; that is, $k(e) = c(e) - f(e)$.

Definition 4.31 A *chain* K is a path in a digraph where the direction of the arcs are ignored.

Augmenting Flow Algorithm

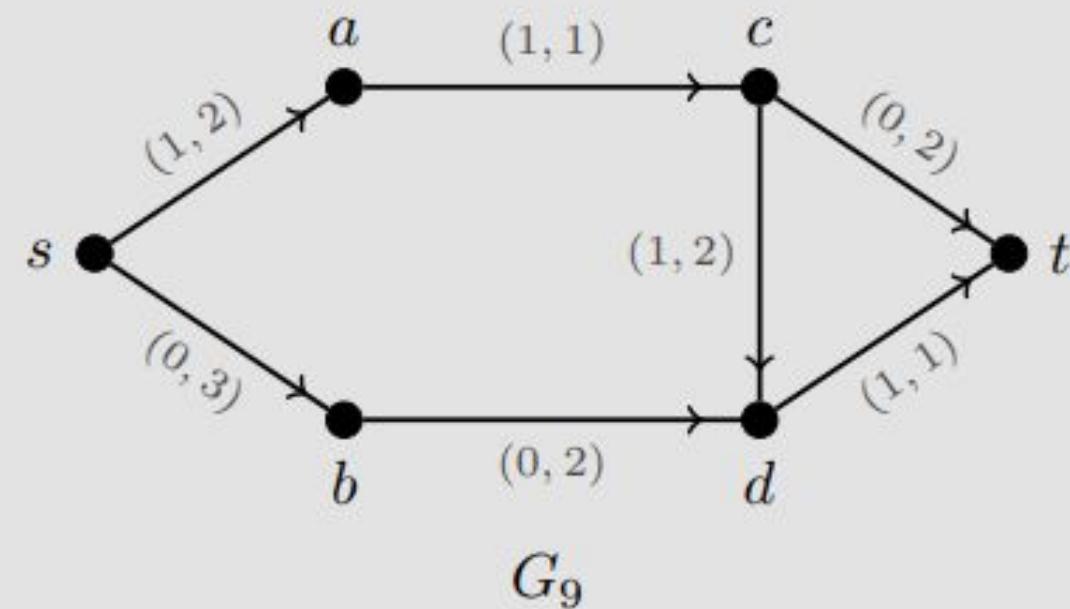
Input: Network $G = (V, E, c)$, with designated source s and sink t , and each arc is given a capacity c .

Steps:

1. Label s with $(-, \infty)$
2. Choose a labeled vertex x .
 - (a) For any arc yx , if $f(yx) > 0$ and y is unlabeled, then label y with $(x^-, \sigma(y))$ where $\sigma(y) = \min\{\sigma(x), f(yx)\}$.
 - (b) For any arc xy , if $k(xy) > 0$ and y is unlabeled, then label y with $(x^+, \sigma(y))$ where $\sigma(y) = \min\{\sigma(x), k(xy)\}$.
3. If t has been labeled, go to Step (4). Otherwise, choose a different labeled vertex that has not been scanned and go to Step (2). If all labeled vertices have been scanned, then f is a maximum flow.
4. Find an $s - t$ chain K of slack edges by backtracking from t to s . Along the edges of K , increase the flow by $\sigma(t)$ units if they are in the forward direction and decrease by $\sigma(t)$ units if they are in the backward direction. Remove all vertex labels except that of s and return to Step (2).

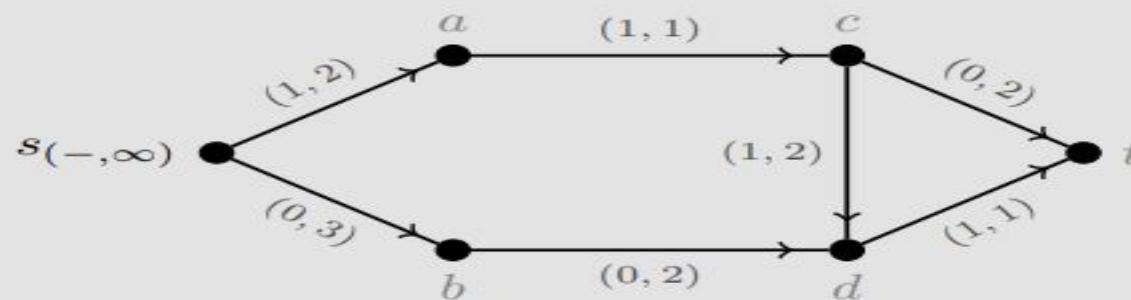
Output: Maximum flow f .

Example 4.5 Apply the Augmenting Flow Algorithm to the network G_9 shown below.

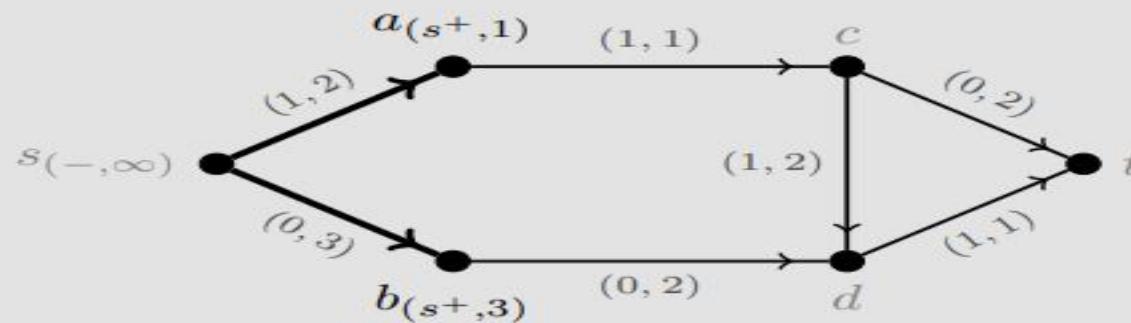


Solution: We will show edges under consideration in bold.

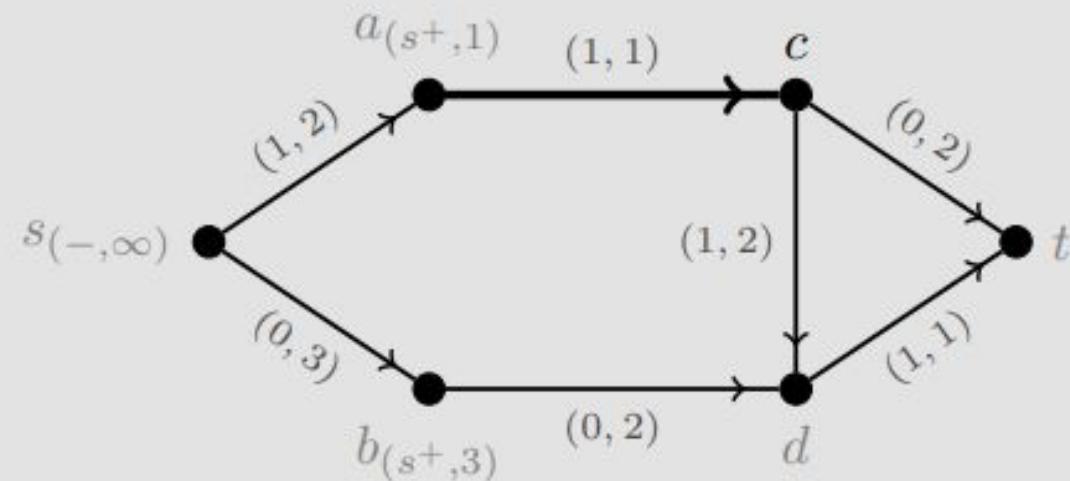
Step 1: Label s as $(-, \infty)$.



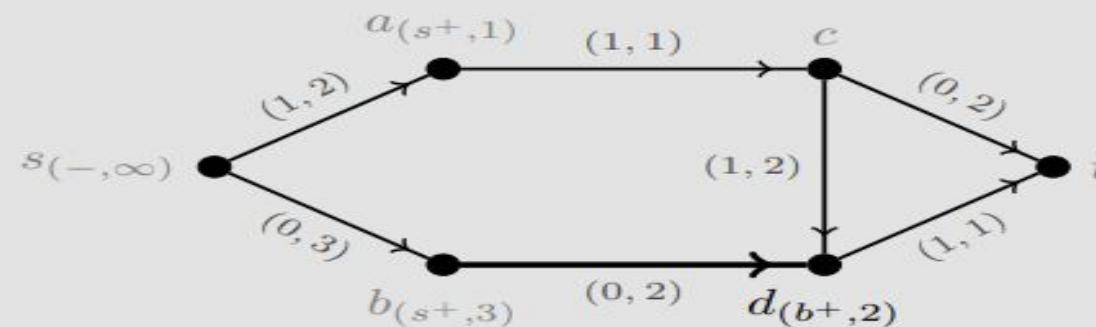
Step 2: Let $x = s$. As there are no arcs to s we will only consider the arcs out of s , of which there are two: sa and sb , which have slack of 1 and 3, respectively. We label a with $(s^+, 1)$, b is labeled $(s^+, 3)$.



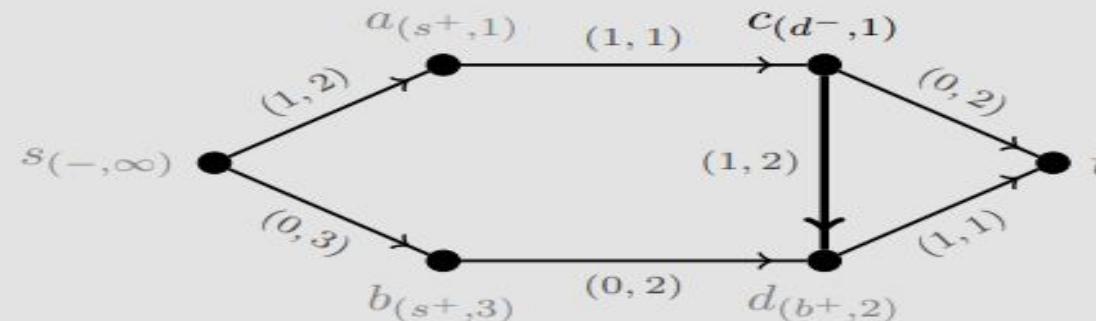
Step 3: As t is not labeled, we will scan either a or b ; we choose to start with a , (so $x = a$ in the algorithm). Since the only arc going into a is from a labeled vertex, we need only consider the edges out of a , of which there is only one, ac , which has no slack. Thus c is left unlabeled at this stage.



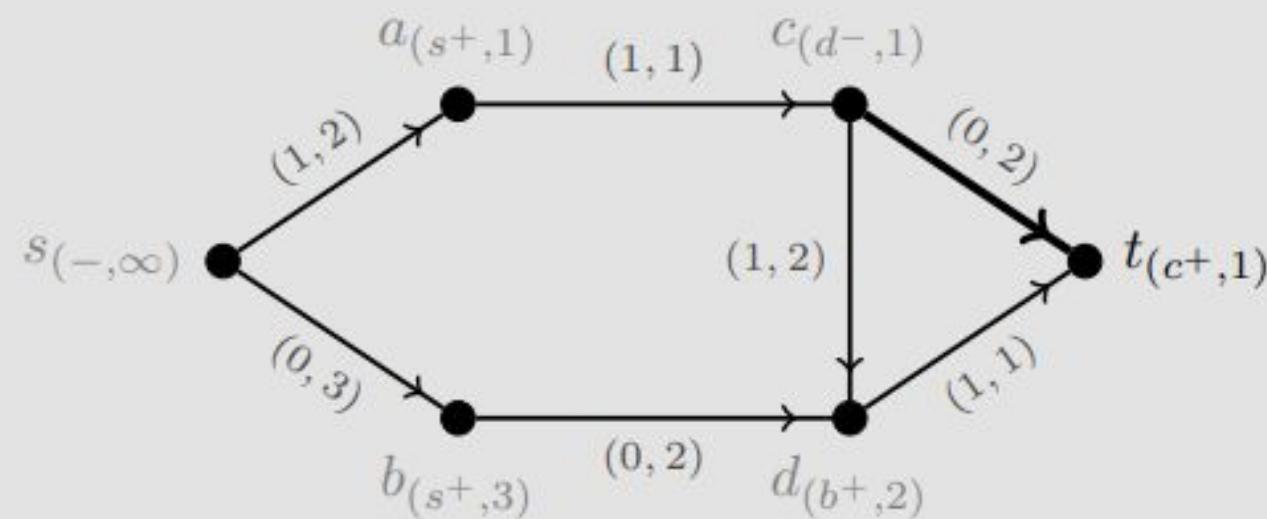
Step 4: As t is not labeled, we will scan b (so $x = b$ in the algorithm). Since the only arc going into b is from a labeled vertex, we need only consider the edges out of b , of which there is only one, bd , with slack of 2. Label d as $(b^+, 2)$ since $\sigma(d) = \min\{\sigma(b), k(bd)\} = \min\{3, 2\} = 2$.



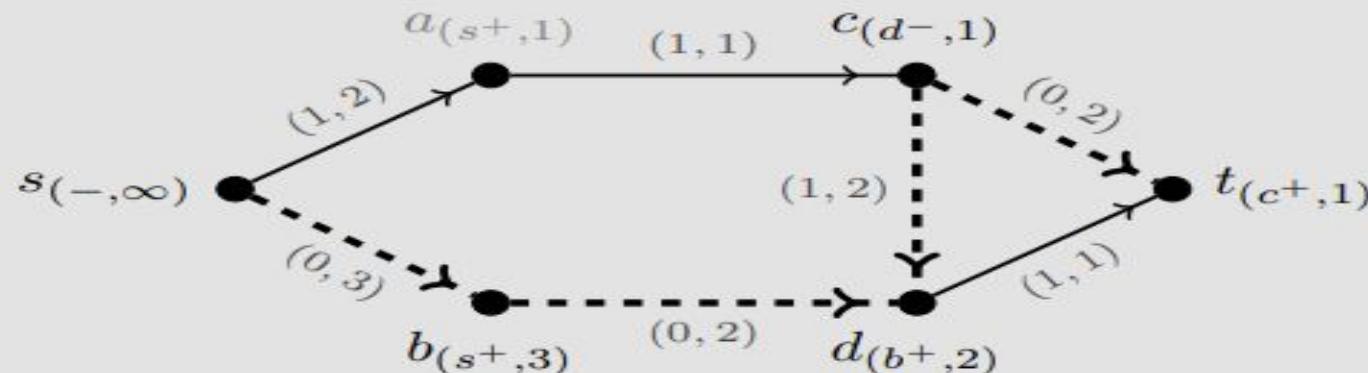
Step 5: As t is not labeled, we will scan d (so $x = b$ in the algorithm). There is one unlabeled vertex with an arc going into d , namely c , which gets a label of $(d^-, 1)$ since $\sigma(c) = \min\{\sigma(d), f(cd)\} = \min\{2, 1\} = 1$. The only arc out of d is dt , which has no slack.



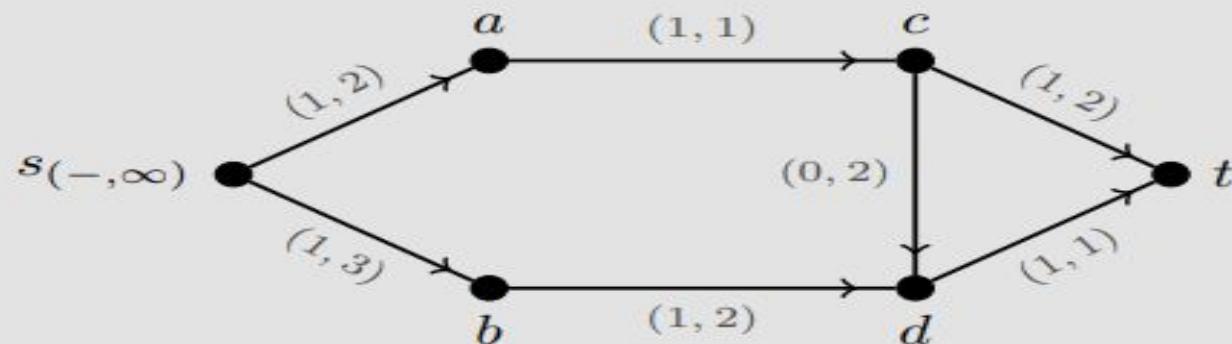
Step 6: As t is not labeled, we will scan c (so $x = c$ in the algorithm). Since the only arc going into c is from a labeled vertex, we need only consider the edges out of c to an unlabeled vertex, of which there is only one, ct , with slack of 2. Label t as $(c^+, 1)$ since $\sigma(t) = \min\{\sigma(c), k(ct)\} = \min\{1, 2\} = 1$.



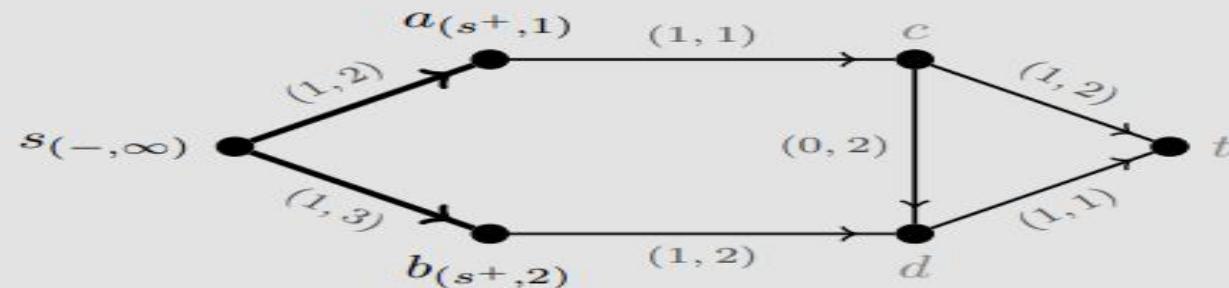
Step 7: Since t is now labeled, we find an $s - t$ chain K of slack edges. Backtracking from t to s gives the chain $sbdc$.



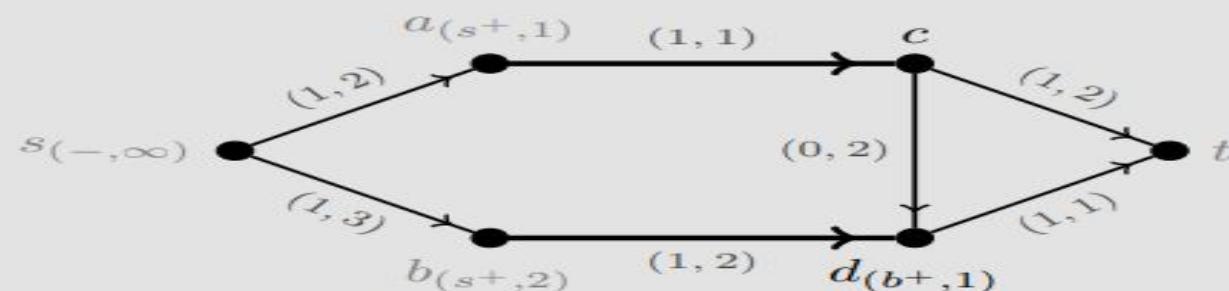
We increase the flow by $\sigma(t) = 1$ units along each of the edges in the forward direction (namely sb, bd, ct) and decrease by 1 along the edges in the backward direction (namely cd). We update the network flow and remove all labels except that for s .



Step 8: As before we will only label the vertices whose arcs from s have slack. We label a with $(s^+, 1)$ and b with $(s^+, 2)$.



Step 9: We scan either a or b ; we begin with a . The only arc from a is to c , but since there is no slack we do not label c . Scanning b we only consider the arc bd , which has slack 1. Label d with $(b^+, 1)$ since $\sigma(d) = \min\{\sigma(b), k(bd)\} = \min\{2, 1\} = 1$.



Scanning d will not assign a label to t since there is no slack along the arc dt , and c also remains unlabeled since cd has no flow. At this point there are no further vertices to label and so f must be a maximum flow of G_9 , with a value of 2.

Lec # 23, 24 & 25

Definition 4.32 Let P be a set of vertices and \bar{P} denote those vertices not in P (called the complement of P). A ***cut*** (P, \bar{P}) is the set of all arcs xy where x is a vertex from P and y is a vertex from \bar{P} . An ***s – t cut*** is a cut in which the source s is in P and the sink t is in \bar{P} .

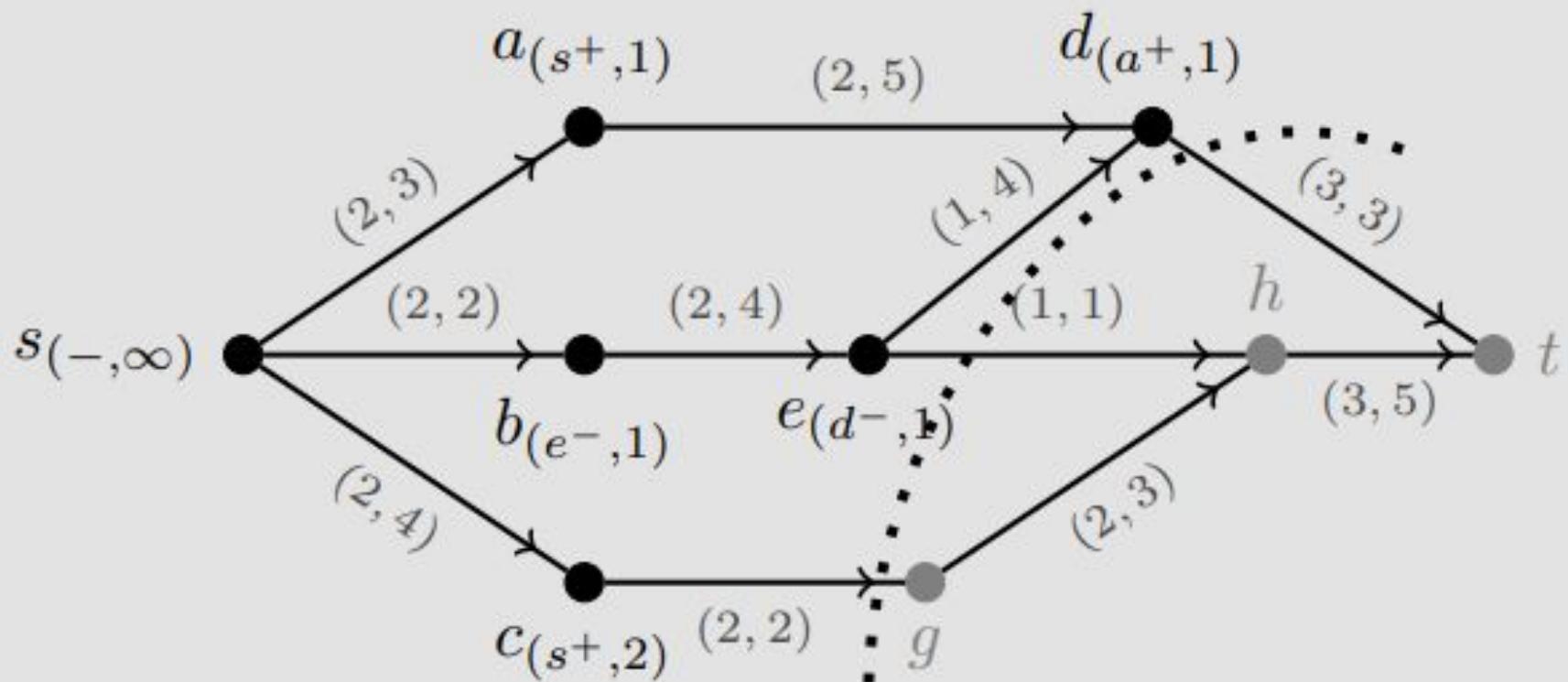
Definition 4.33 The ***capacity*** of a cut, $c(P, \bar{P})$, is defined as the sum of the capacities of the arcs that comprise the cut.

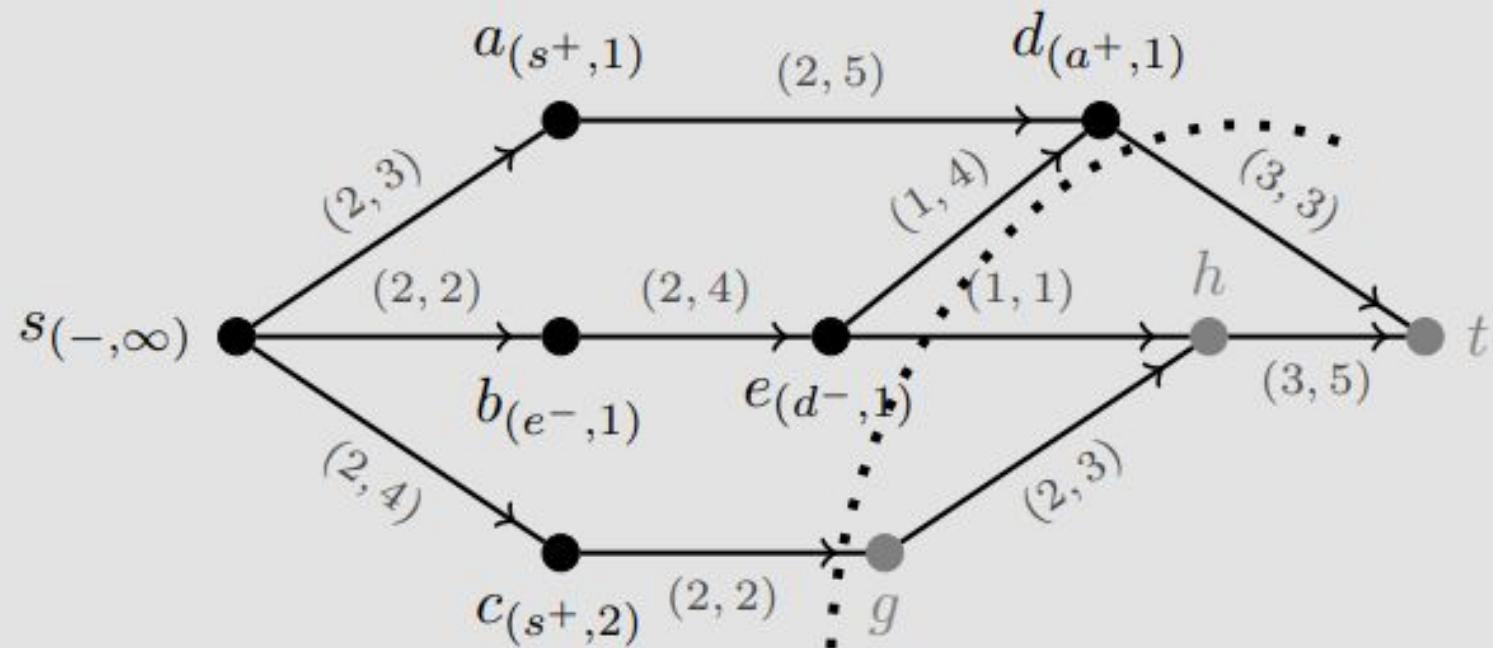
Theorem 4.34 (Max Flow–Min Cut) In any directed network, the value of a maximum $s – t$ flow equals the capacity of a minimum $s – t$ cut.

Min-Cut Method

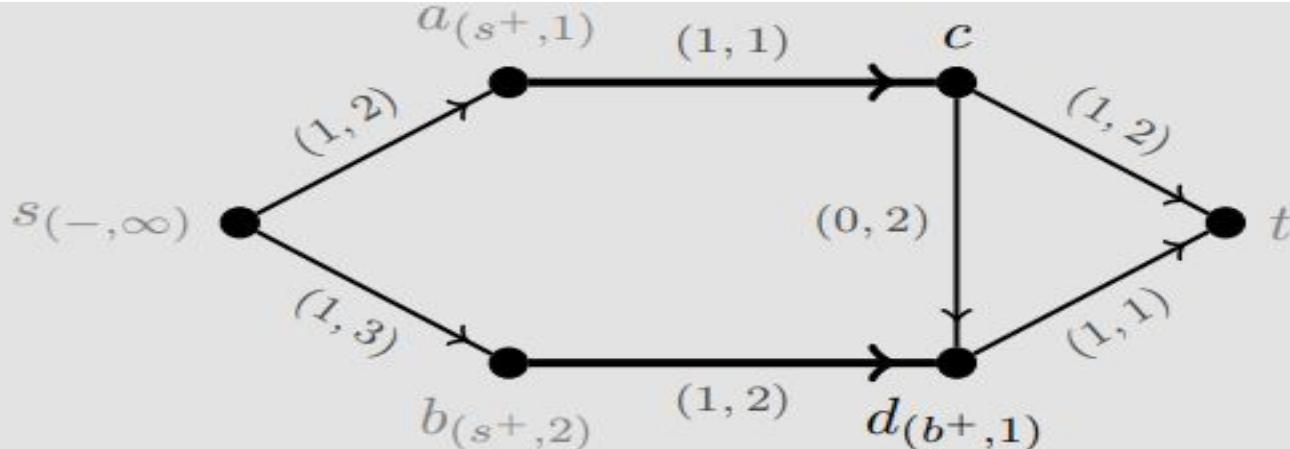
1. Let $G = (V, A, c)$ be a network with a designated source s and sink t and each arc is given a capacity c .
2. Apply the Augmenting Flow Algorithm.
3. Define an $s - t$ cut (P, \bar{P}) where P is the set of labeled vertices from the final implementation of the algorithm.
4. (P, \bar{P}) is a minimum $s - t$ cut for G .

Example 4.7 Use the Min-Cut Method to find a minimum $s - t$ cut for the network G_8 on page 189 and the network G_9 from Example 4.5.

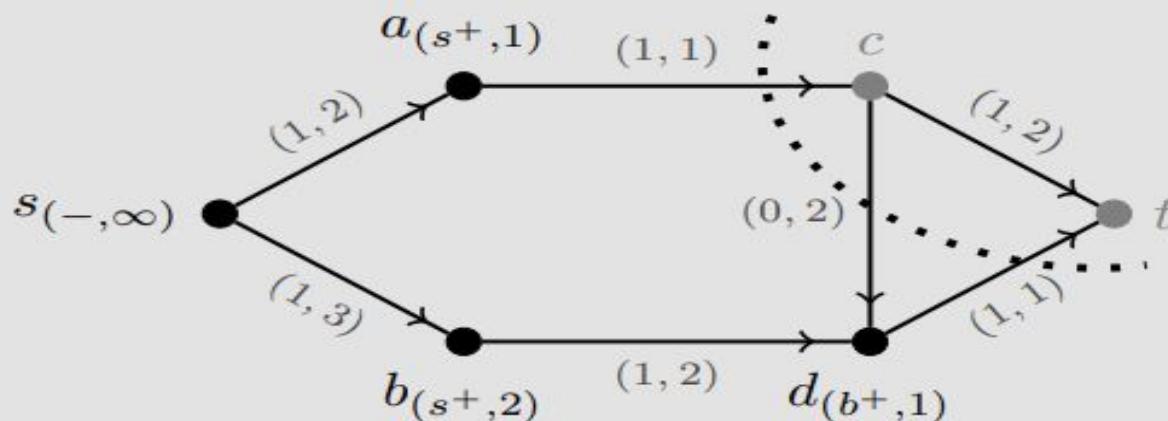




The Min-Cut Method sets $P = \{s, a, b, c, d, e\}$ and $\bar{P} = \{g, h, t\}$. The arcs in the cut are $\{dt, eh, cg\}$, making the capacity of this cut $c(P, \bar{P}) = 3 + 1 + 2 = 6$. Since we have found a flow and cut with the same value, we know the flow is maximum and the cut is minimum.



The final labeling in the network G_9 from Example 4.5 gives $P = \{s, a, b, d\}$ and $\bar{P} = \{c, t\}$. The arcs in the cut set are $\{ac, dt\}$. Note, cd is not in the cut since the arc is in the wrong direction. The capacity of this cut is $c(P, \bar{P}) = 2$, and since we have found a flow and cut with the same value, we know the flow is maximum and the cut is minimum.



EX #:4.6

Problems: 4.1-4.6, 4.9, 4.10, 4.13, 4.14, 4.15, 4.17, 4.20.

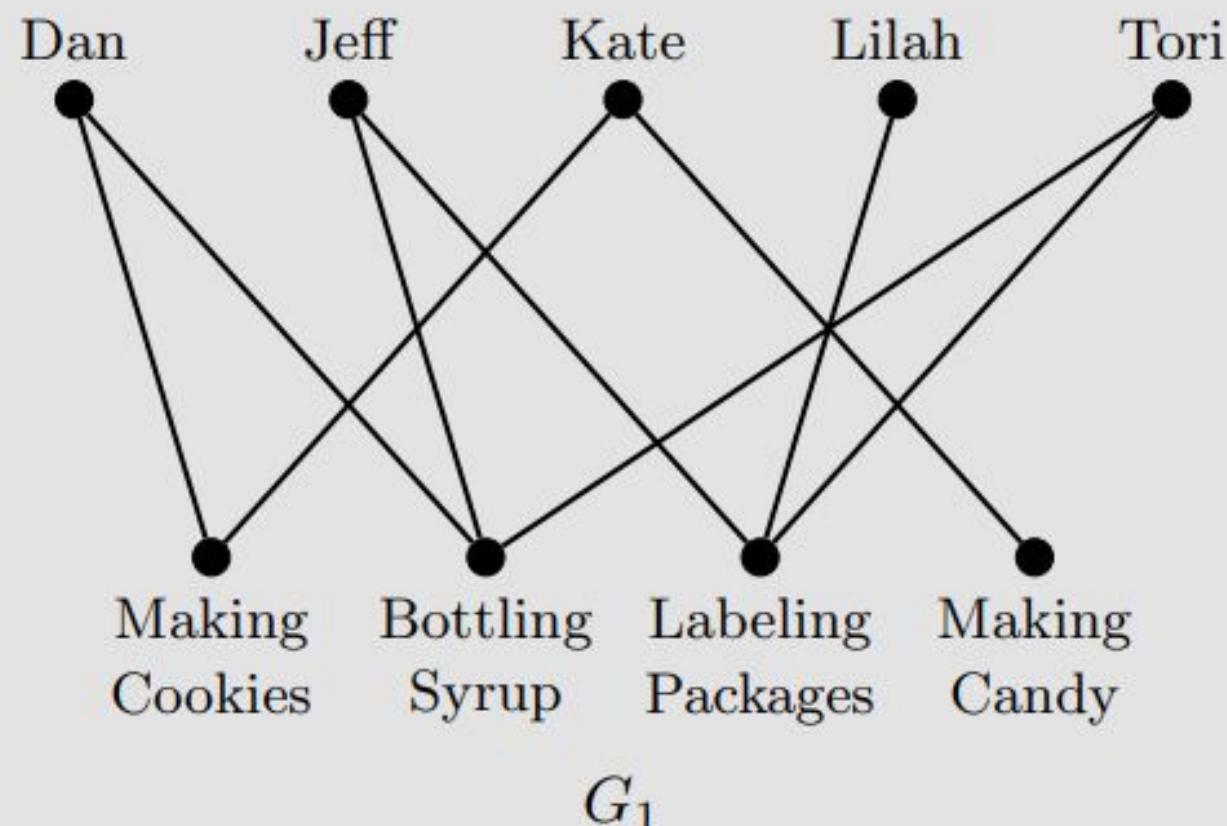
Matching in Bipartite Graph

Definition 5.1 Given a graph $G = (V, E)$, a *matching* M is a subset of the edges of G so that no two edges share an endpoint. The size of a matching, denoted $|M|$, is the number of edges in the matching.

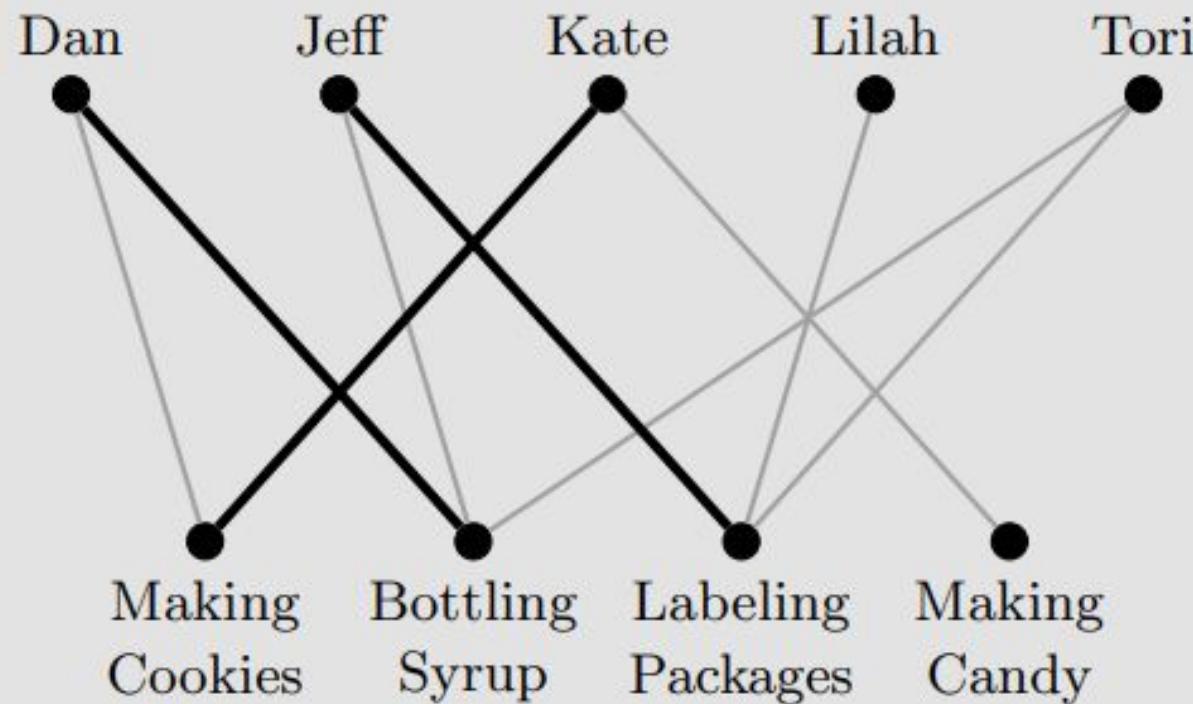
Example 5.1 The Vermont Maple Factory just received a rush order for 6-dozen boxes of maple cookies, 3-dozen bags of maple candy, and 10-dozen bottles of maple syrup. Some employees have volunteered to stay late tonight to help finish the orders. In the chart below, each employee is shown along with the jobs for which he or she is qualified. Draw a graph to model this situation and find a matching.

Employee	Task	
Dan	Making Cookies	Bottling Syrup
Jeff	Labeling Packages	Bottling Syrup
Kate	Making Candy	Making Cookies
Lilah	Labeling Packages	
Tori	Labeling Packages	Bottling Syrup

Solution: Model using a bipartite graph where X consists of the employees and Y consists of the tasks. We draw an edge between two vertices a and b if employee a is capable of completing the task b , creating G_1 below.

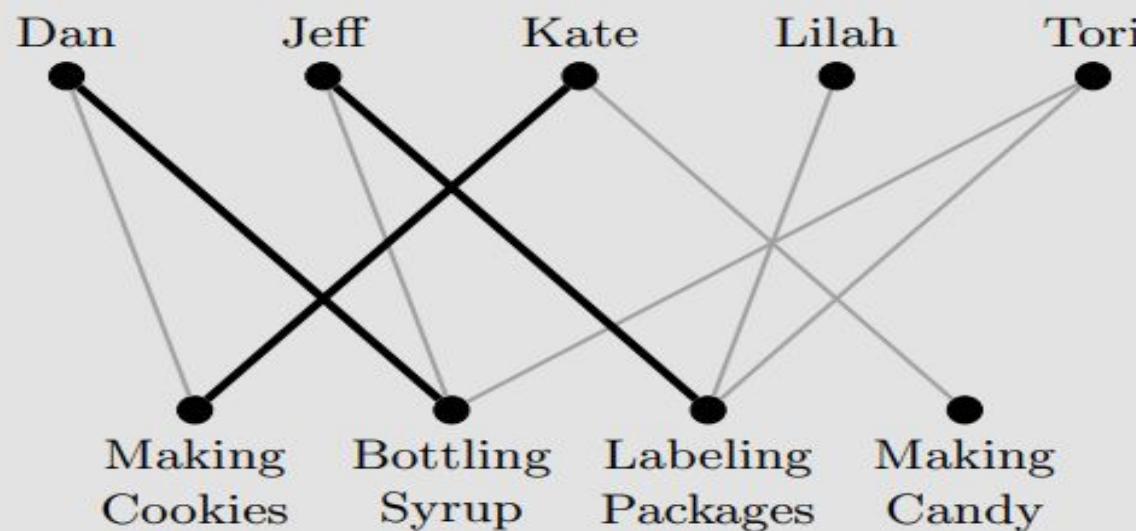


A matched edge, which is shown in bold below, represents the assignment of a task to an employee. One possible matching is shown below.



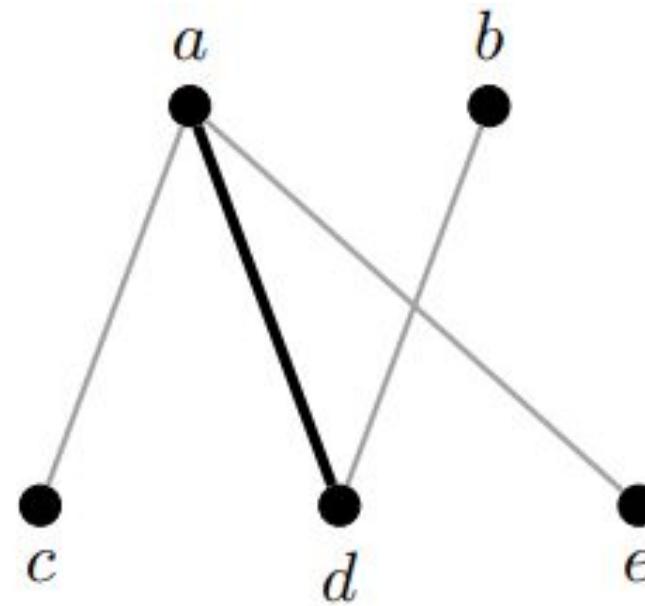
Definition 5.2 A vertex is *saturated* by a matching M if it is incident to an edge of the matching; otherwise, it is called *unsaturated*.

A matched edge, which is shown in bold below, represents the assignment of a task to an employee. One possible matching is shown below.

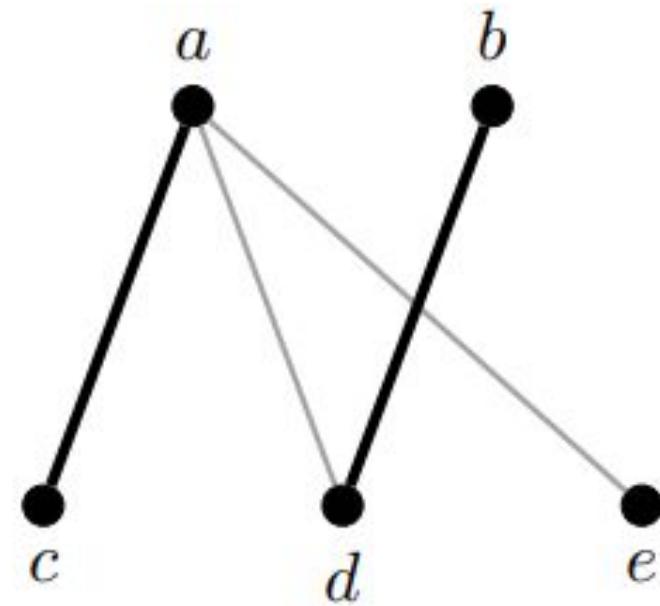


Definition 5.3 Given a matching M on a graph G , we say M is

- ***maximal*** if M cannot be enlarged by adding an edge.
- ***maximum*** if M is of the largest size amongst all possible matchings.
- ***perfect*** if M saturates every vertex of G .
- an ***X-matching*** if it saturates every vertex from the collection of vertices X (a similar definition holds for a Y -matching).



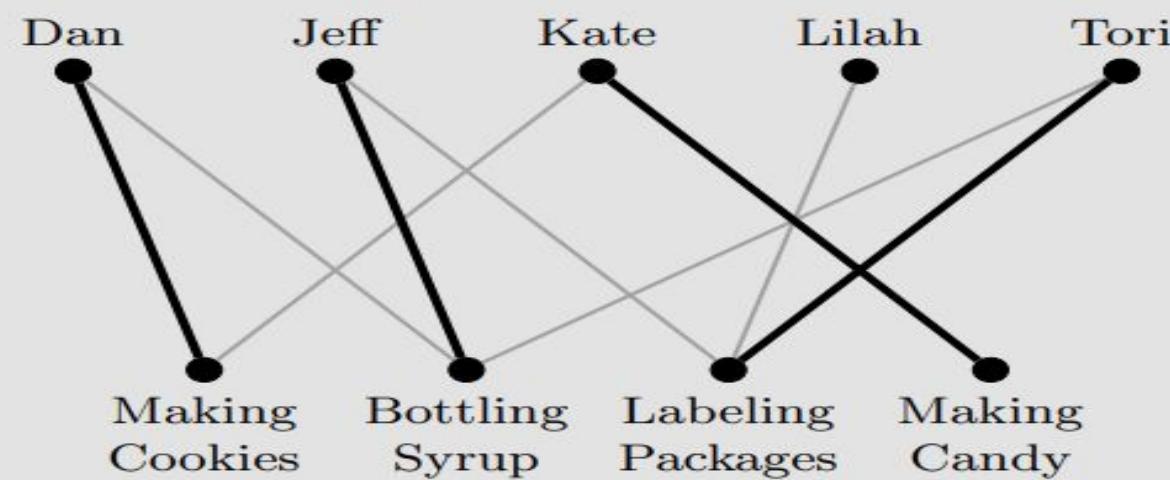
Maximal Matching
of G_2



Maximum Matching
of G_2

Example 5.2 Determine and find the proper type of matching for the Vermont Maple Factory graph G_1 from Example 5.1.

Solution: Since we need the tasks to be completed but do not need every employee to be assigned a task, we must find an X -matching where X consists of the vertices representing the tasks. An example of such a matching is shown below.



Note that all tasks are assigned to an employee, but not all employees have a task (Lilah is not matched with a task). In addition, this is not the only matching possible. For example, we could have Lilah labeling packages and Tori bottling syrup with Jeff having no task to complete.

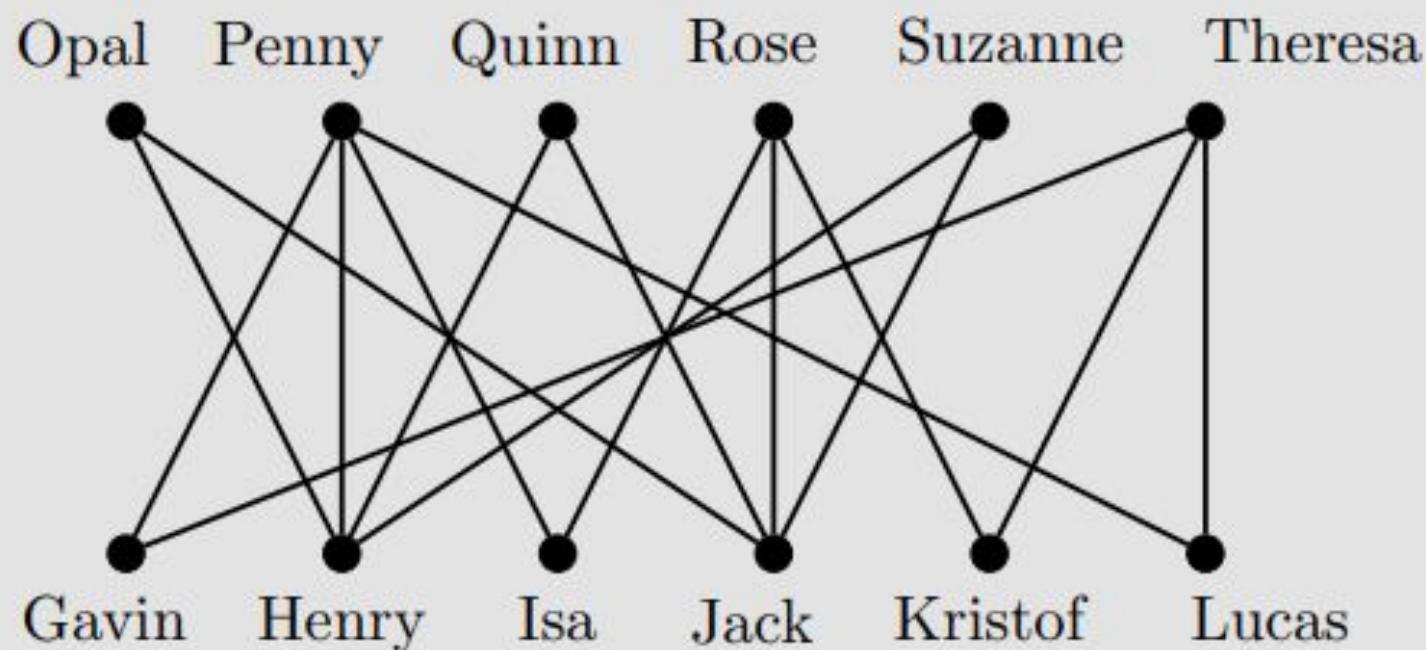
Theorem 5.4 (Hall's Marriage Theorem) Given a bipartite graph $G = (X \cup Y, E)$, there exists an X -matching if and only if $|S| \leq |N(S)|$ for any $S \subseteq X$.



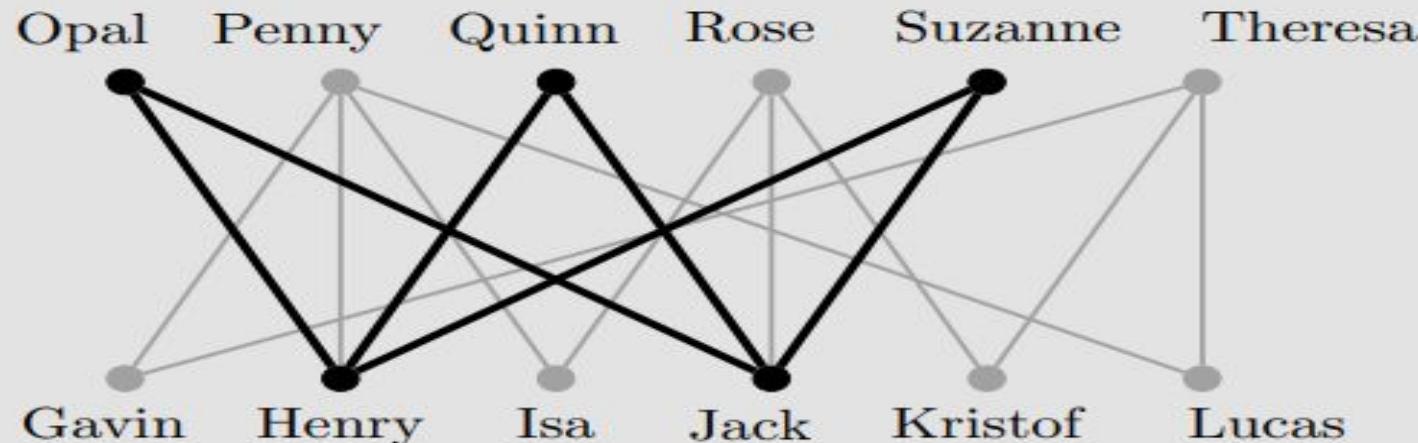
Example 5.3 In a small town there are 6 boys and 6 girls whose parents wish to pair into marriages where the only requirement is that a girl must like her future spouse (pretty low standards in my opinion). The table below lists the girls and the boys she likes. Find a pairing with as many marriages occurring as possible.

Girls	Boys She Likes			
Opal	Henry	Jack		
Penny	Gavin	Isa	Henry	Lucas
Quinn	Henry	Jack		
Rose	Kristof	Isa	Jack	
Suzanne	Henry	Jack		
Theresa	Gavin	Lucas	Kristof	

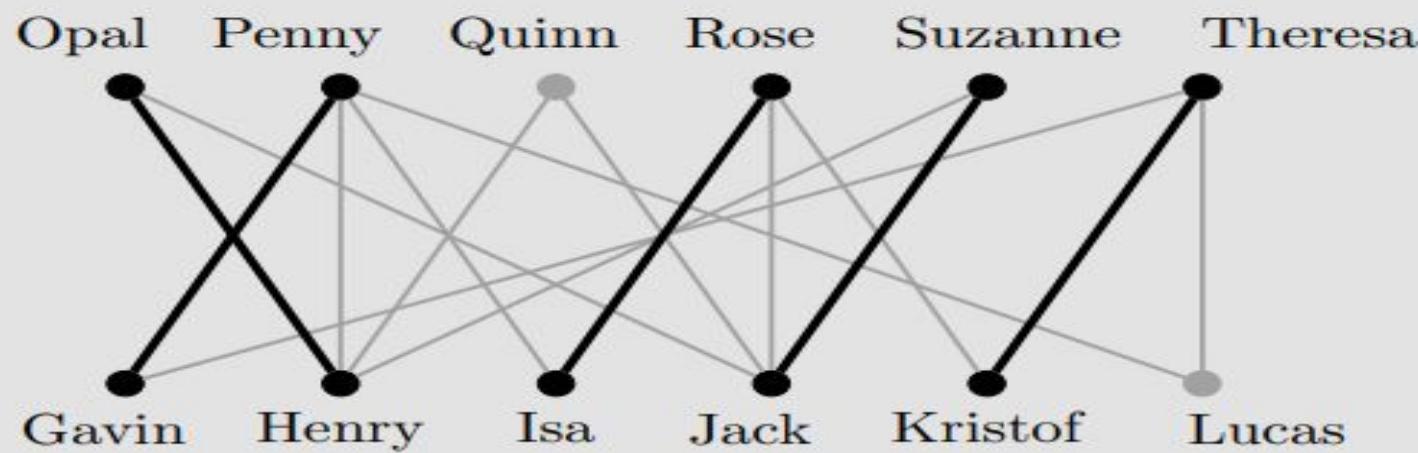
Solution: The information from the table will be displayed using a bipartite graph, where X consists of the girls and Y consists of the boys.



Notice that Opal, Quinn, and Suzanne all only like the same two boys (Henry and Jack), so at most two of these girls can be matched, as shown below in the following graph.



This means at most 5 marriages are possible; one such solution is shown below.



Corollary 5.5 Every k -regular bipartite graph has a perfect matching for all $k > 0$.

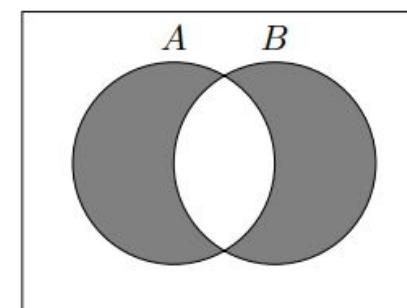
Definition 5.6 Given a matching M of a graph G , a path is called

- **M -alternating** if the edges in the path alternate between edges that are part of M and edges that are not part of M .
- **M -augmenting** if it is an M -alternating path and both endpoints of the path are unsaturated by M , implying both the starting and ending edges of the path are not part of M .

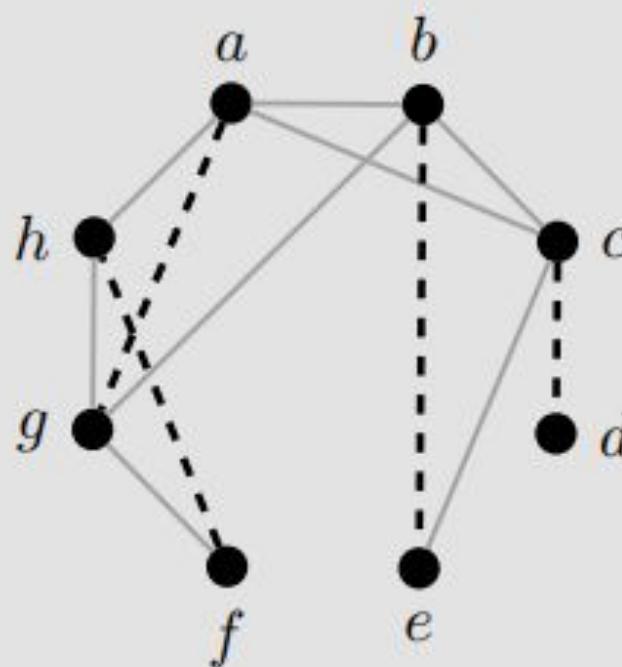
Theorem 5.7 (Berge's Theorem) A matching M of a graph G is maximum if and only if G does not contain any M -augmenting paths.

Definition 5.8 Let A and B be two sets. Then the *symmetric difference* $A \Delta B$ is all those elements in exactly one of A and B ; that is, $A \Delta B = (A - B) \cup (B - A)$.

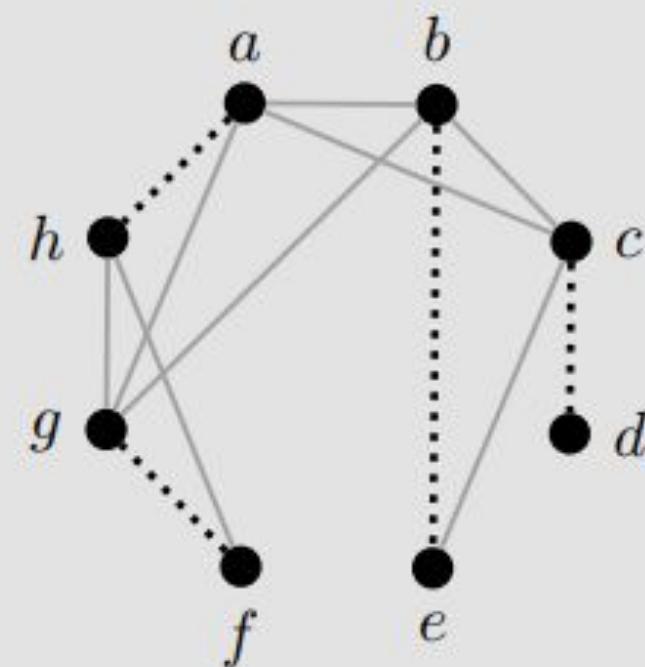
$$A \Delta B = A \cup B - A \cap B.$$



Example 5.4 Below are two different matchings of a graph G . Find $M_1 \triangle M_2$.

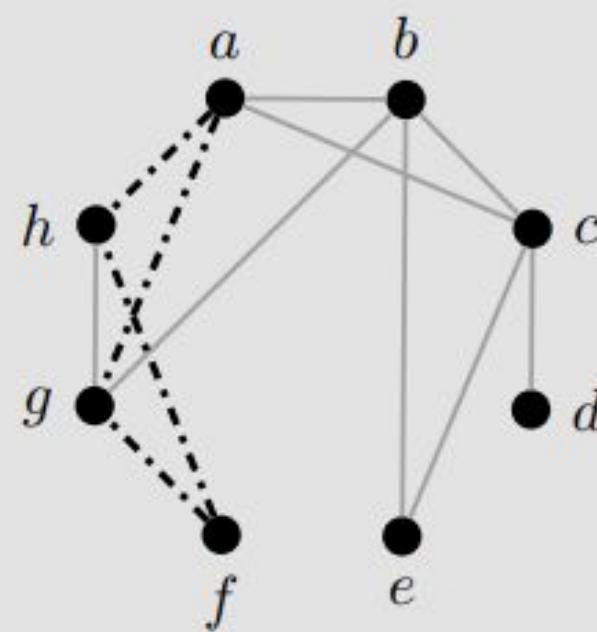


M_1



M_2

Solution: Note that the symmetric difference will only show those edges involved in exactly one of the two matchings. Thus if an edge is in both matchings or left unmatched in both matchings, it does not appear in $M_1 \Delta M_2$.



$$M_1 \Delta M_2$$

Lemma 5.9 Let M_1 and M_2 be two matchings in a graph G . Then every component of $M_1 \Delta M_2$ is either a path or an even cycle.

Theorem 5.7 (Berge's Theorem, restated) A matching M of a graph G is not maximum if and only if G contains some M -augmenting path.

Lec # 26, 27 & 28

Augmenting Path Algorithm

Input: Bipartite graph $G = (X \cup Y, E)$.

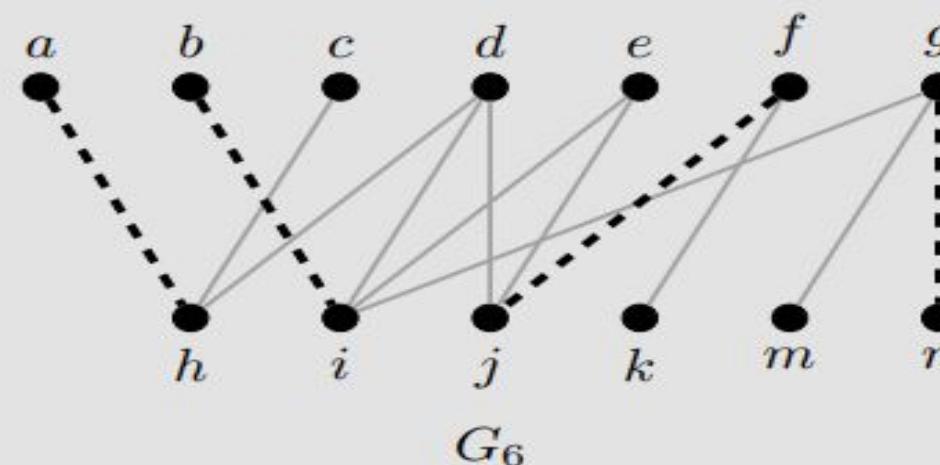
Steps:

1. Find an arbitrary matching M .
2. Let U denote the set of unsaturated vertices in X .
3. If U is empty, then M is a maximum matching; otherwise, select a vertex x from U .
4. Consider y in $N(x)$.
5. If y is also unsaturated by M , then add the edge xy to M to obtain a larger matching M' . Return to Step (2) and recompute U . Otherwise, go to Step (6).
6. If y is saturated by M , then find a maximal M -alternating path from x using xy as the first edge.

- (a) If this path is M -augmenting, then switch edges along that path to obtain a larger matching M' ; that is, remove from M the matched edges along the path and add the unmatched edges to create M' . Return to Step (2) and recompute U .
 - (b) If the path is not M -augmenting, return to Step (4), choosing a new vertex from $N(x)$.
7. Stop repeating Steps (2)–(4) when all vertices from U have been considered.

Output: Maximum matching for G .

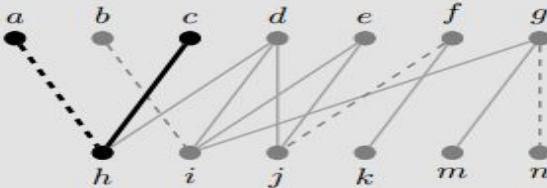
Example 5.5 Apply the Augmenting Path Algorithm to the bipartite graph G_6 below, where $X = \{a, b, c, d, e, f, g\}$ and $Y = \{h, i, j, k, m, n\}$, with an initial matching shown as dashed lines.



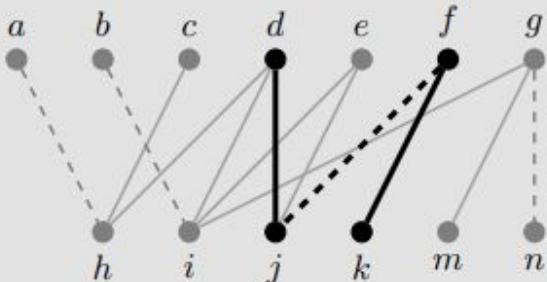
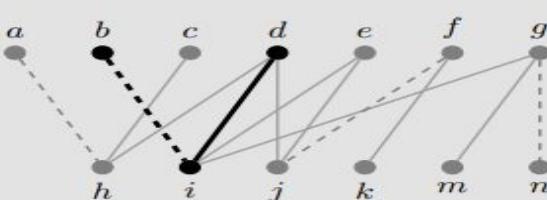
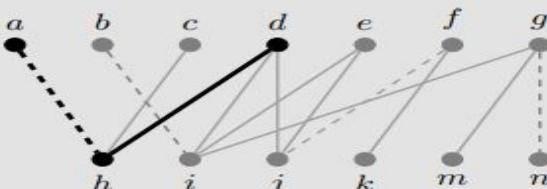
Solution: During each step shown below, the path under consideration will be in bold, with the matching shown as dashed lines throughout.

Step 1: The unsaturated vertices from X are $U = \{c, d, e\}$.

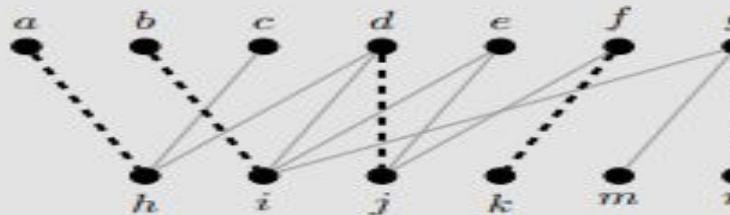
Step 2: Choose c . The only neighbor of c is h , which is saturated by M . Form an M -alternating path starting with the edge ch . This produces the path cha , as shown on the next page, which is not augmenting.



Step 3: Choose a new vertex from U , say d . Then $N(d) = \{h, i, j\}$. Below are the alternating paths originating from d .

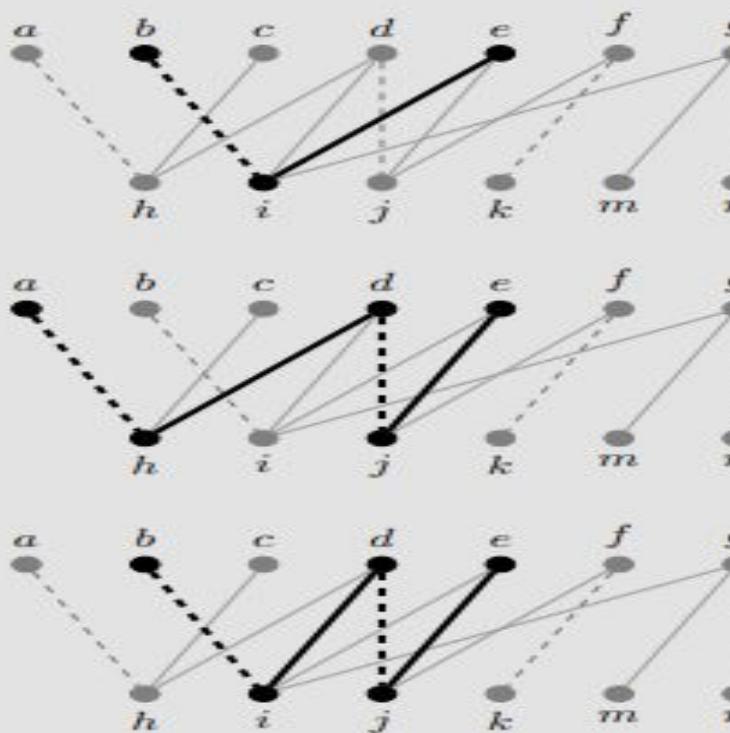


Note that the last path $(d j f k)$ is M -augmenting. Form a new matching M' by removing edge fj from M and adding edges dj and fk , as shown in the following graph.



Step 4: Recalculate $U = \{c, e\}$. We must still check c since it is possible for the change in matching to modify possible alternating paths from a previously reviewed vertex; however, the path obtained is $c h a$, the same as from Step 2.

Step 5: Check the paths from e . The alternating paths are shown below.



None of these paths are augmenting. Thus no M' -augmenting paths exist in G and so M' must be maximum by Berge's Theorem.

Output: The maximum matching $M' = \{ah, bi, dj, fk, gn\}$ from Step 3.

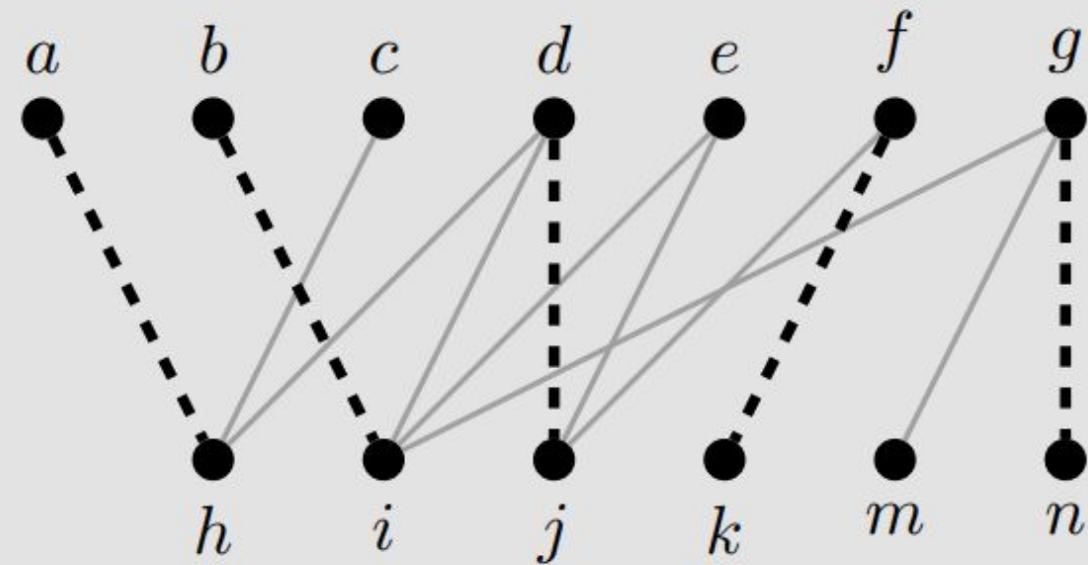
Definition 5.10 A *vertex cover* Q for a graph G is a subset of vertices so that every edge of G has at least one endpoint in Q .

Theorem 5.11 (König-Egerváry Theorem) For a bipartite graph G , the size of a maximum matching of G equals the size of a minimum vertex cover for G .

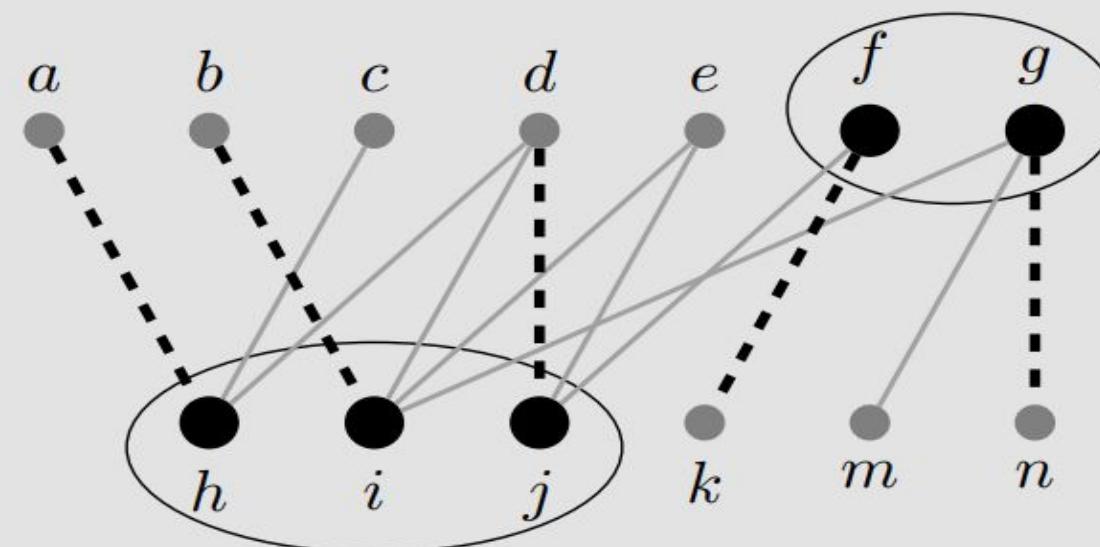
Vertex Cover Method

1. Let $G = (X \cup Y, E)$ be a bipartite graph.
2. Apply the Augmenting Path Algorithm and mark the vertices considered throughout its final implementation.
3. Define a vertex cover Q as the unmarked vertices from X and the marked vertices from Y .
4. Q is a minimum vertex cover for G .

Example 5.6 Apply the Vertex Cover Method to the output graph from Example 5.5.



Solution: Recording the vertices considered throughout the last step of the Augmenting Path Algorithm, the marked vertices from X are a, b, c, d , and e , and the marked vertices from Y are h, i , and j . This produces the vertex cover $Q = \{f, g, h, i, j\}$ of size 5, shown below. Recall that the maximum matching contained 5 edges.



Matching in General Graphs

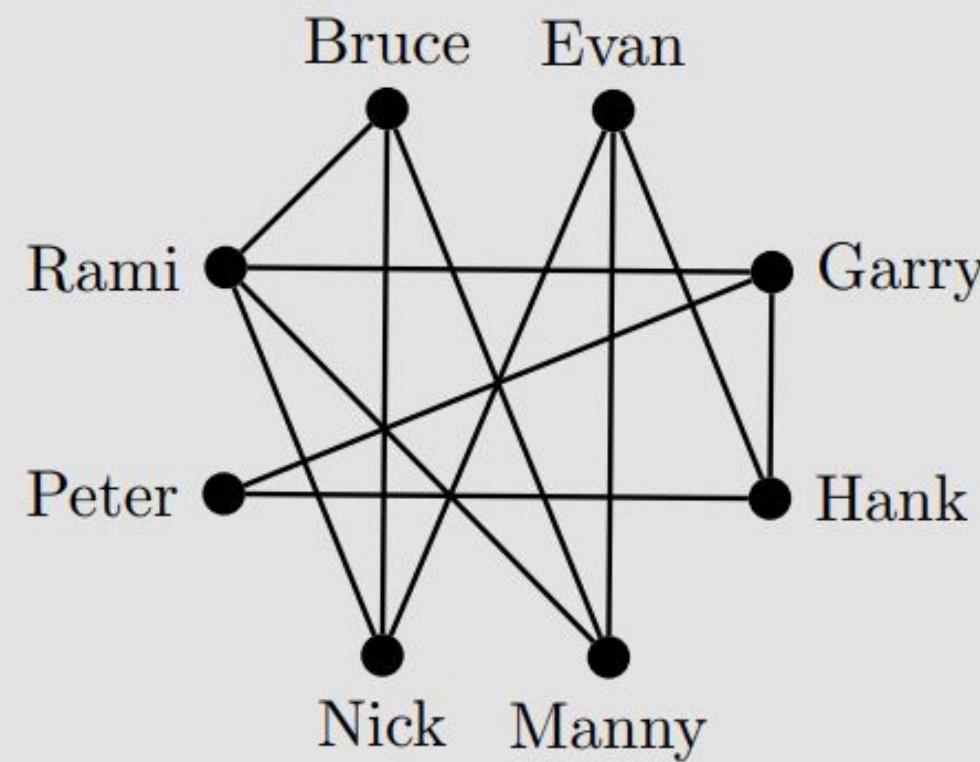
Theorem 5.7 (Berge's Theorem) A matching M of a graph G is maximum if and only if G does not contain any M -augmenting paths.

Bruce, Evan, Garry, Hank, Manny, Nick, Peter, and Rami decide to go on a week-long canoe trip in Guatemala. They must divide themselves into pairs, one pair for each of four canoes, where everyone is only willing to share a canoe with a few of the other travelers.

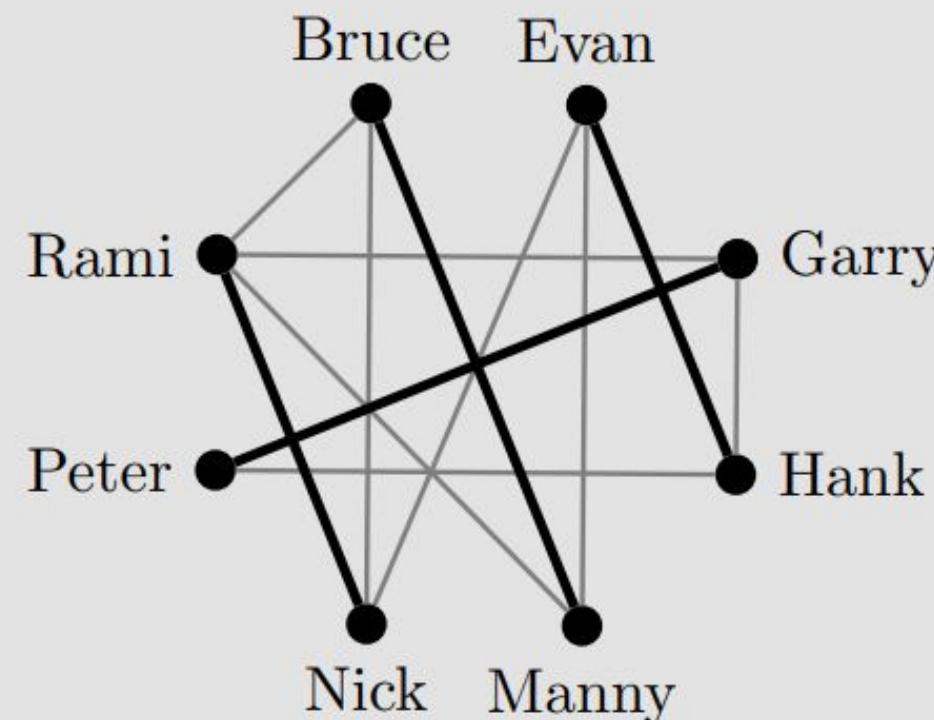
Example 5.9 The group of eight men from above have listed who they are willing to share a canoe with. This information is shown in the following table, where a Y indicates a possible pair. Note that these relationships are symmetric, so if Bruce will share a canoe with Manny, then Manny is also willing to share a canoe with Bruce. Model this information as a graph. Find a perfect matching or explain why no such matching exists.

	Bruce	Evan	Garry	Hank	Manny	Nick	Peter	Rami
Bruce	Y	Y	.	Y
Evan	.	.	.	Y	Y	Y	.	.
Garry	.	.	.	Y	.	.	Y	Y
Hank	.	Y	Y	.	.	.	Y	.
Manny	Y	Y	Y
Nick	Y	Y	Y
Peter	.	.	Y	Y
Rami	Y	.	Y	.	Y	Y	.	.

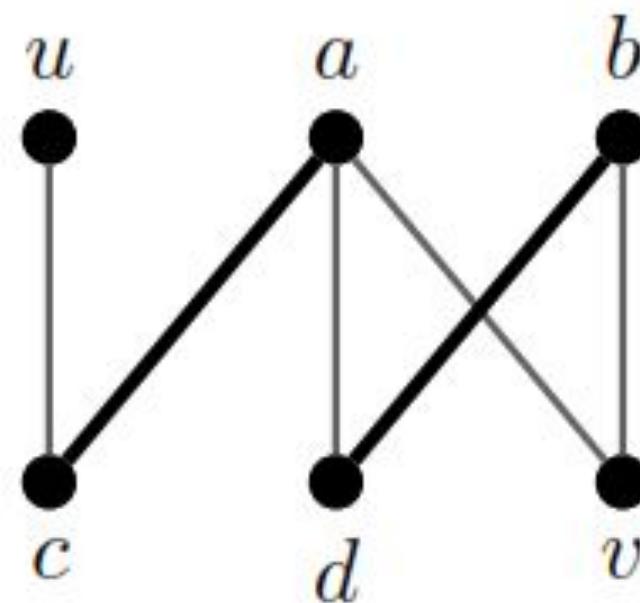
Solution: The graph is shown below where an edge represents a potential pairing into a canoe.



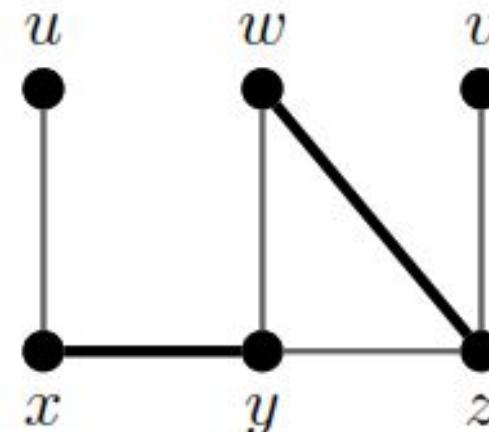
Note that Peter can only be paired with either Garry or Hank. If we choose to pair Peter and Garry, then Hank must be paired with Evan, leaving Nick and Manny to each be paired with one of Rami and Bruce. One possible matching is shown below. Since all people have been paired, we have a perfect matching.



Corollary 5.16 Every cubic graph without any bridges has a perfect matching.



G_8

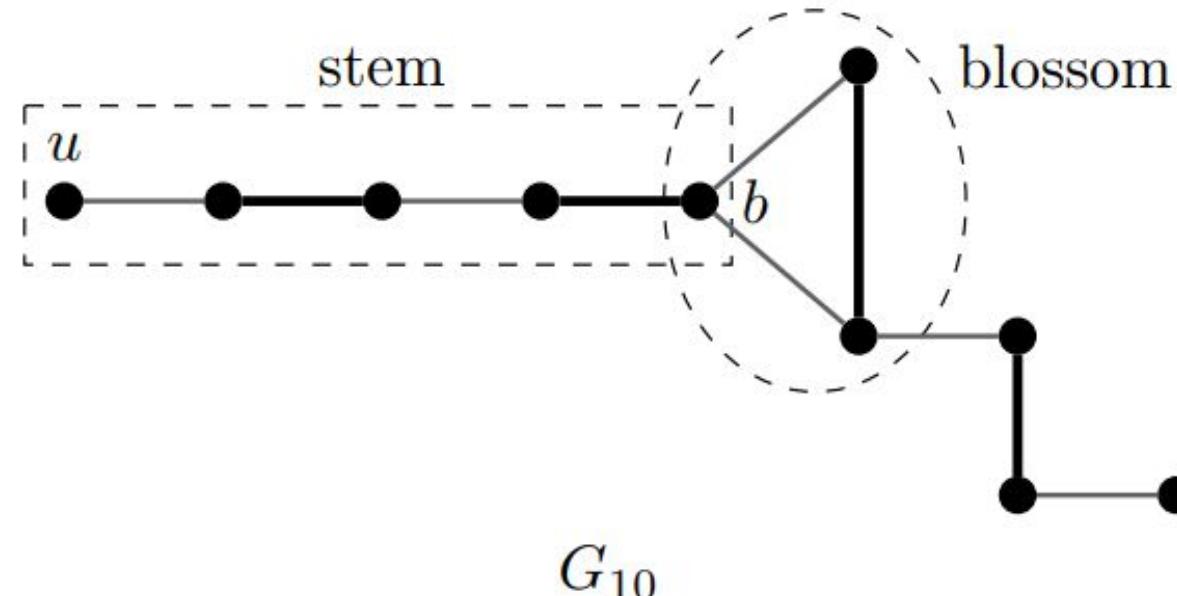


G_9

For example, we can find an alternating path of **length 4** from u to z , namely $u x y w z$, but another alternating path from u to z exists of **length 3** where the last edge is not matched, namely $u x y z$. This is caused by the **odd cycle occurring** between y , w and z . Note that the vertex from which these two paths diverge (namely y) is entered by a matched edge (xy) and has two possible unmatched edges out (yw and yz). This configuration is the basis behind the **blossom**.

Edmonds' Blossom Algorithm

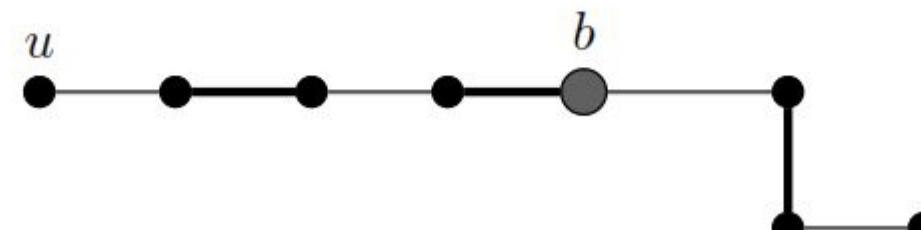
Definition 5.17 Given a graph G and a matching M , a *flower* is the union of two M -alternating paths from an unsaturated vertex u to another vertex v where one path has odd length and the other has even length. The *stem* of the flower is the maximal common initial path out of u , that ends at a vertex b , called the *base*. The *blossom* is the odd cycle that is obtained by removing the stem from the flower.



A few additional details can be discerned from these definitions. First, the stem must be of even length since the last edge must be from the matching M . Next, the blossom is an odd cycle C_{2k+1} where exactly k edges are from M , since the two edges from the base on the cycle must both be unmatched edges. Moreover, every vertex of the blossom must be saturated by M since traveling some direction along the cycle will end with an edge from M . Finally, the only edges that come off the blossom that are also from M must be from the stem.

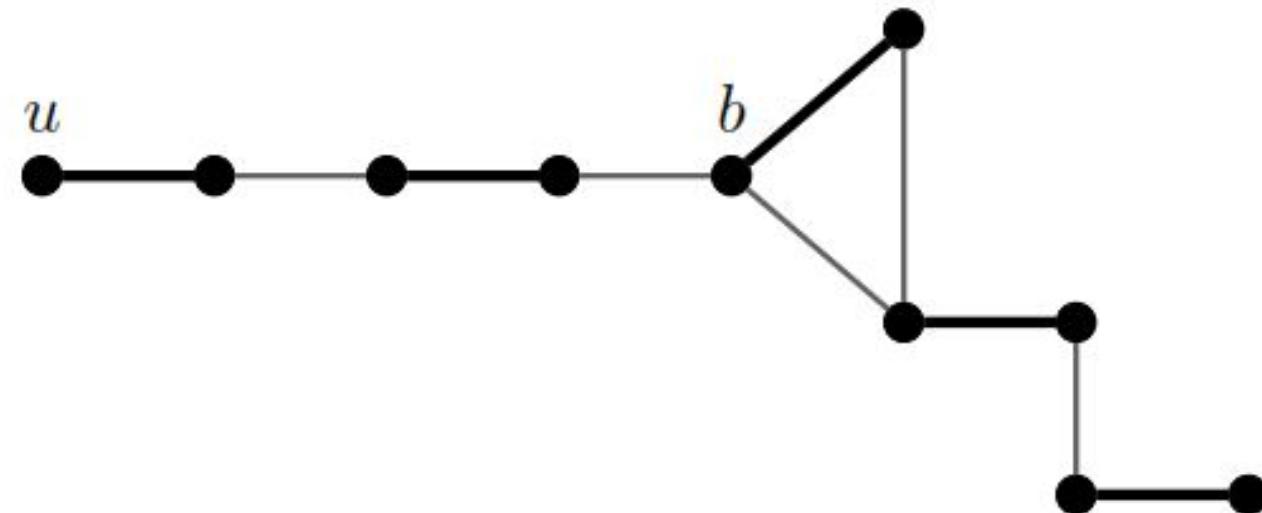
Edmonds' Blossom Algorithm starts in the same way as the Augmenting Path Algorithm, where we examine alternating paths originating from an unsaturated vertex. Where these algorithms differ is **when a blossom is encountered**. When all alternating paths from u are being explored, if two different paths to a vertex are found to end in different types of edges (namely matched or unmatched), then a blossom has been discovered. We can then

Point 1 **contract the blossom**, much in the same way we contract an edge from Chapter 4 in the proof of Menger's Theorem. The contraction of the blossom from G_{10} above is shown below.



G_{10} with blossom contracted

If we find an augmenting path in the contracted graph, then we can find an augmenting path in the original graph by choosing the correct direction along the blossom. Now just like in the Augmenting Path Algorithm we can swap edges along this path, creating a larger matching while still maintaining the properties of a matching.



G_{10} with matching swapped

Edmonds' Blossom Algorithm

A **blossom** is a set of nodes and edges starting at an open vertex, with a “stem” of even number of edges (matched and unmatched edges alternating), and a cycle of odd number of edges (again with alternating matched and unmatched edges, but with the two edges adjacent to the stem unmatched). A illustration is shown in Figure 8.1(a).

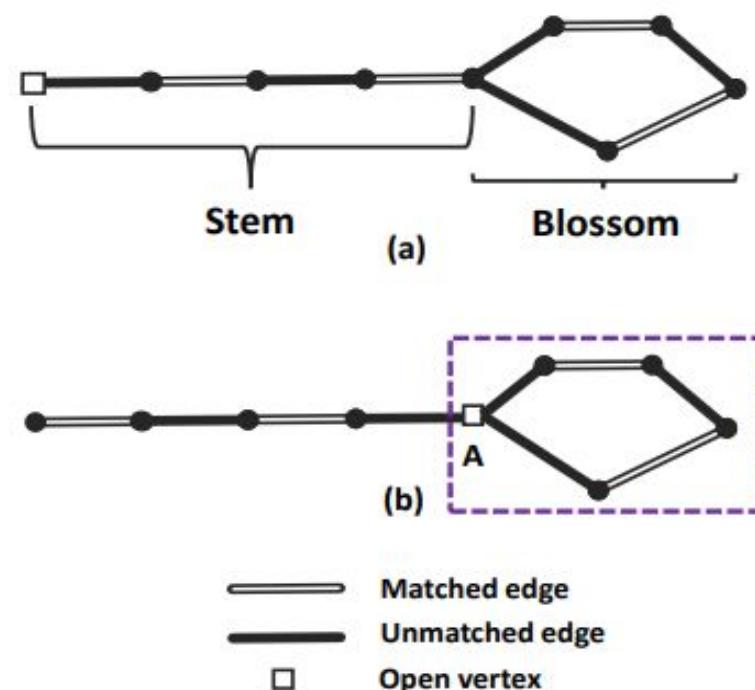
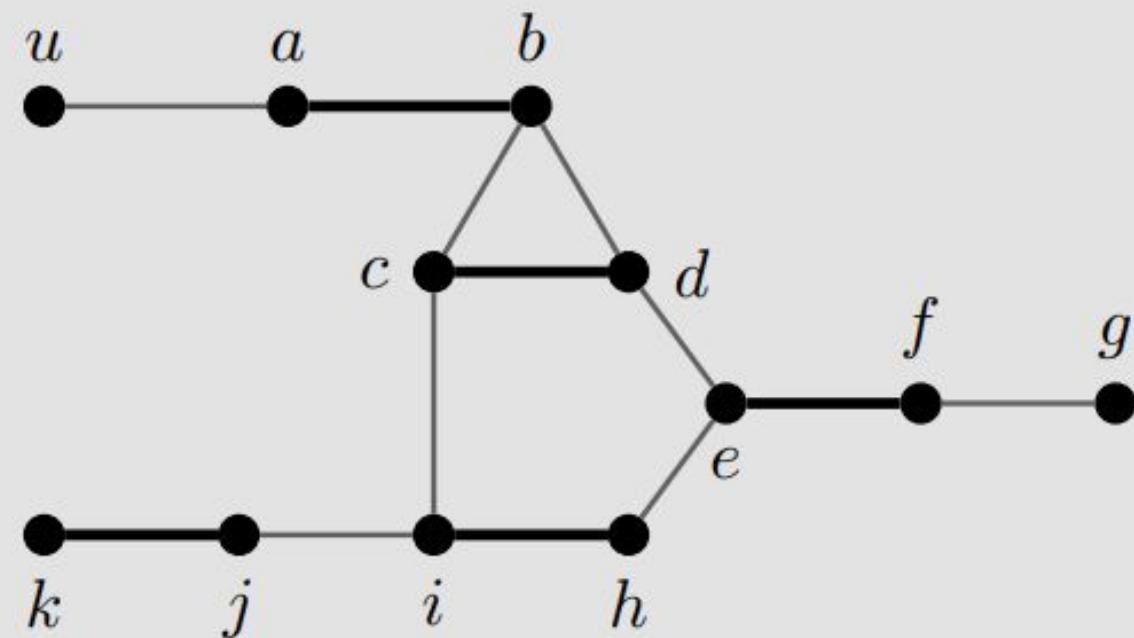
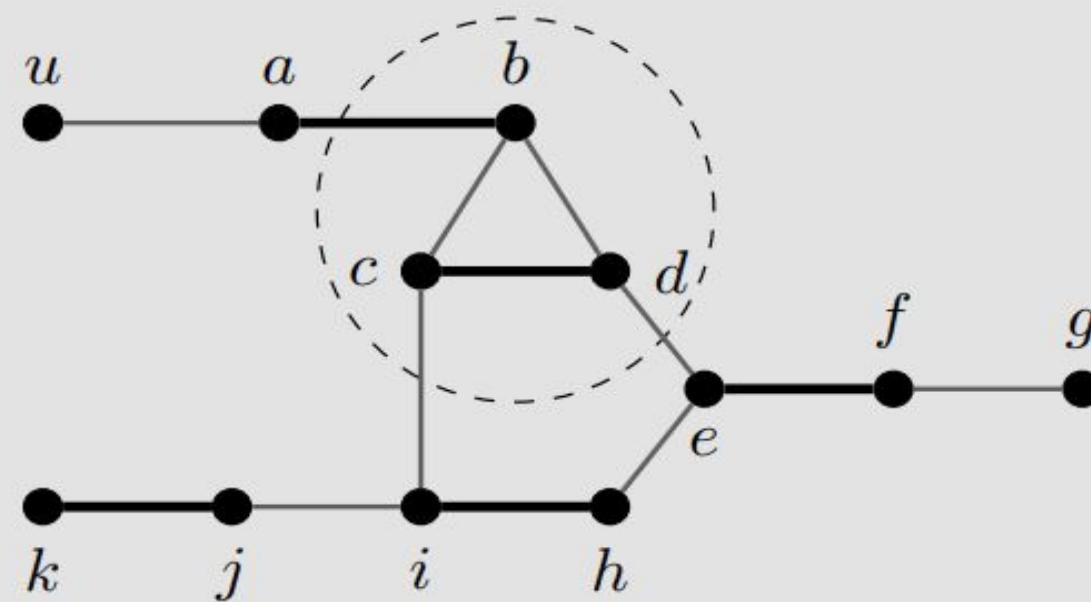


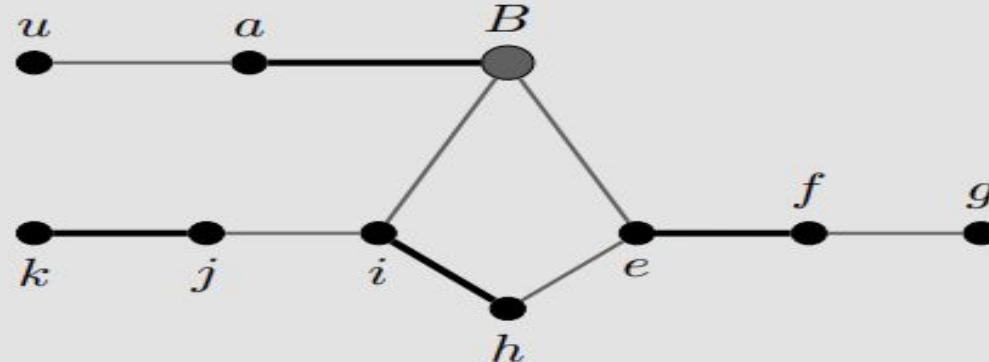
Figure 8.1: An example of blossom and the toggling of the stem.

Example 5.11 Use Edmonds' Blossom Algorithm to find a maximum matching on the graph below, where the initial matching is shown in bold.

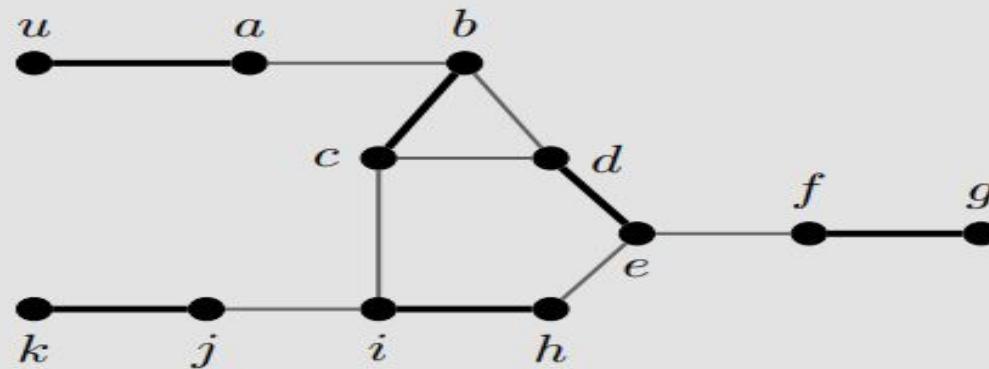


Solution: We begin by looking for alternating paths out of u . Note that we will explore all paths simultaneously (essentially building a Breadth-First Tree). At b we have two options for finding an alternating path, namely $uabc$ and $uabd$. If we take either path out to their next vertex, we get the ending vertex of the other previous path (such as $uabdc$ and vertex c from the path $uabc$). This implies that we have found a blossom, with odd cycle bcd and base vertex b . We contract this blossom to a vertex B as shown below.





Again we build out alternating paths from u . In doing so, we find the path $uaBefg$, which is augmenting since it is alternating with both endpoints being unsaturated. Thus we retrace this path in the original graph as $uabcdefg$ and swap all edges along this path to obtain a larger matching.



The matching shown above is maximum since all vertices are saturated (and so is also a perfect matching).

Definition 5.18 A perfect matching is *stable* if no unmatched pair is *unstable*; that is, if x and y are not matched but both rank the other higher than their current partner, then x and y form an unstable pair.

Stable Matching Problem

Given two sets $R = \{r_1, \dots, r_n\}, H = \{h_1, \dots, h_n\}$
each agent ranks **every** agent in the other set.

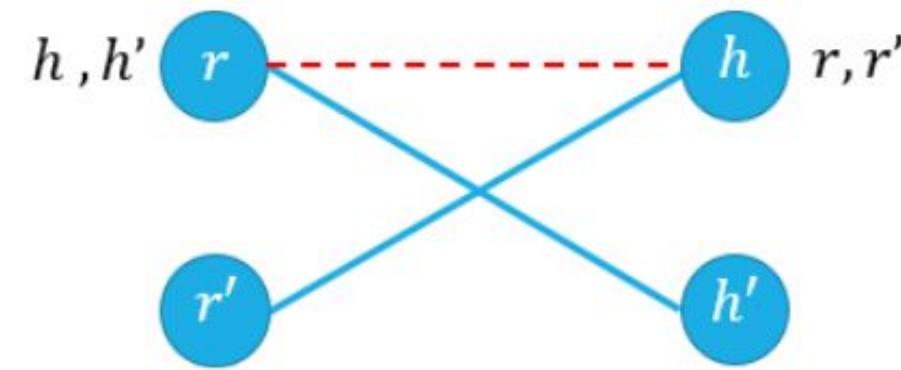
Stable Matching Problem

Given two sets $R = \{r_1, \dots, r_n\}, H = \{h_1, \dots, h_n\}$
each agent ranks **every** agent in the other set.

Goal: Match each agent to **exactly one** agent in the other set, respecting their preferences.

How do we “respect preferences”?

Avoid **blocking pairs**: unmatched pairs (r, h) where r prefers h to their match, and h prefers r to its match.



Stable Matching, More Formally

Perfect matching:

- Each rider is paired with exactly one horse.
- Each horse is paired with exactly one rider.

Stability: no ability to exchange

an unmatched pair $r-h$ is **blocking** if they both prefer each other to current matches.

Stable matching: perfect matching with no blocking pairs.

Algorithm Example

h_1, h_2, h_3  r_1

 h_1 r_2, r_3, r_1

h_1, h_3, h_2  r_2

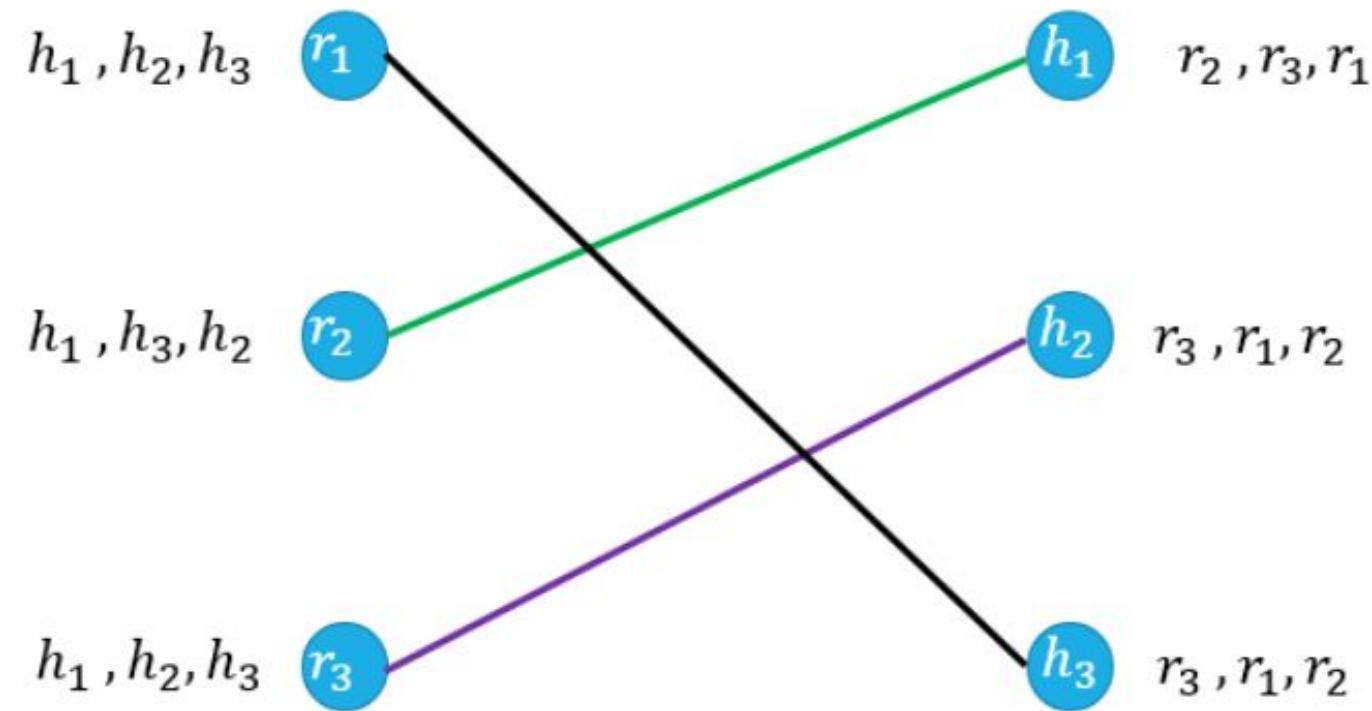
 h_2 r_3, r_1, r_2

h_1, h_2, h_3  r_3

 h_3 r_3, r_1, r_2

Proposals: $r_1, r_2, r_1, r_3, r_3, r_1$

Algorithm Example



Proposals: $r_1, r_2, r_1, r_3, r_3, r_1$

Gale-Shapley Algorithm

Initially all r in R and h in H are free

While there is a free r

 Let h be highest on r 's list that r has not proposed to

 if h is free, then match (r, h)

 else // h is not free

 suppose (r', h) are matched

 if h prefers r to r'

 unmatch (r', h)

 match (r, h)

Example 5.13 Four men and four women are being paired into marriages. Each person has ranked the members of the opposite sex as shown below. Draw a bipartite graph and highlight the matching Anne–Rob, Brenda–Ted, Carol–Stan, and Diana–Will. Determine if this matching is stable. If not, find a stable matching and explain why no unstable pair exists.

Anne: $t > r > s > w$

Brenda: $s > w > r > t$

Carol: $w > r > s > t$

Diana: $r > s > t > w$

Rob: $a > b > c > d$

Stan: $a > c > b > d$

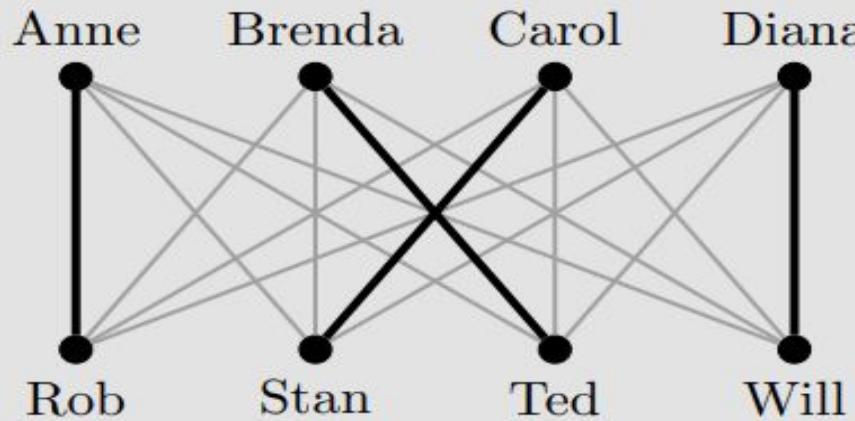
Ted: $c > d > a > b$

Will: $c > b > a > d$

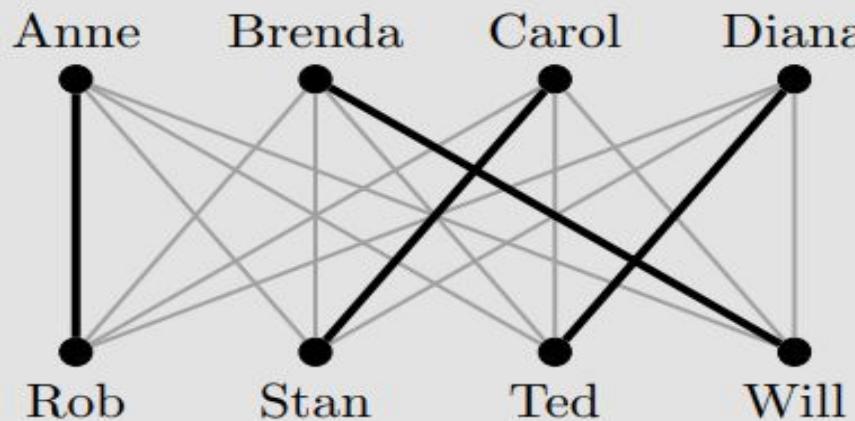
Solution: The complete bipartite graph appears on the next page with the matching in bold. This matching is not stable since Will and Brenda form an unstable pair, as they prefer each other to their current mate.

Anne: t > r > s > w
 Brenda: s > w > r > t
 Carol: w > r > s > t
 Diana: r > s > t > w

Rob: a > b > c > d
 Stan: a > c > b > d
 Ted: c > d > a > b
 Will: c > b > a > d



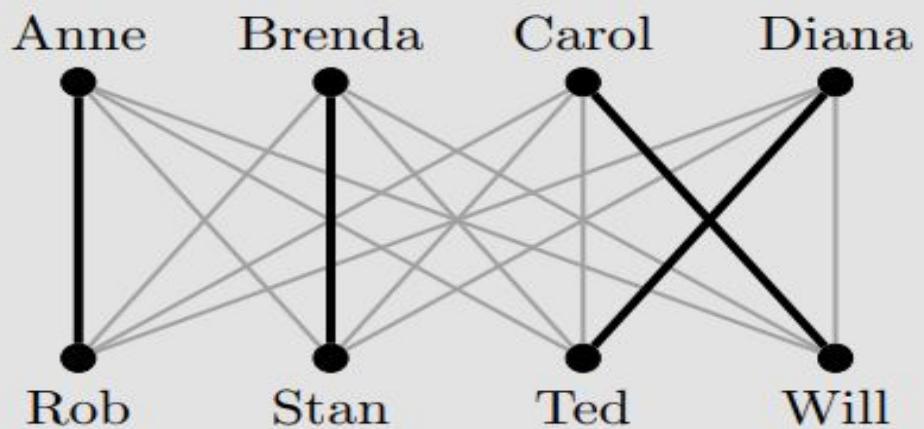
Switching the unstable pairs produces the matching below
 $(\text{Anne} \leftrightarrow \text{Rob}, \text{Brenda} \leftrightarrow \text{Will}, \text{Carol} \leftrightarrow \text{Stan}, \text{and Diana} \leftrightarrow \text{Ted})$.



Anne:	t > r > s > w
Brenda:	s > w > r > t
Carol:	w > r > s > t
Diana:	r > s > t > w

Rob:	a > b > c > d
Stan:	a > c > b > d
Ted:	c > d > a > b
Will:	c > b > a > d

Note that this matching is not stable either since Will and Carol prefer each other to their current mate. Switching the pairs produces a new matching ($\text{Anne} \leftrightarrow \text{Rob}$, $\text{Brenda} \leftrightarrow \text{Stan}$, $\text{Carol} \leftrightarrow \text{Will}$, $\text{Diana} \leftrightarrow \text{Ted}$) shown below.



This final matching is in fact stable due to the following: Will and Carol are paired, and each other's first choice; Anne only prefers Ted over Rob, but Ted does not prefer Anne over Diana; Brenda does not prefer anyone over Stan; Diana prefers either Rob or Stan over Ted, but neither of them prefers Diana to their current mate.

Gale-Shapley Algorithm

Input: Preference rankings of n women and n men.

Steps:

1. Each man proposes to the highest ranking woman on his list.
2. If every woman receives only one proposal, this matching is stable. Otherwise move to Step (3).
3. If a woman receives more than one proposal, she
 - (a) accepts if it is from the man she prefers above all other currently available men and rejects the rest; or,
 - (b) delays with a maybe to the highest ranked proposal and rejects the rest.
4. Each man now proposes to the highest ranking unmatched woman on his list who has not rejected him.
5. Repeat Steps (2)–(4) until all people have been paired.

Output: Stable Matching.

Instance of the Stable Marriage Problem

An instance of the stable marriage problem can be specified either by two sets of preference lists or by a ranking matrix, as in the example below.

men's preferences

1st 2nd 3rd

Bob: Lea Ann Sue

Jim: Lea Sue Ann

Tom: Sue Lea Ann

women's preferences

1st 2nd 3rd

Ann: Jim Tom Bob

Lea: Tom Bob Jim

Sue: Jim Tom Bob

ranking matrix

Ann Lea Sue

Bob 2,3 1,2 3,3

Jim 3,1 1,3 2,1

Tom 3,2 2,1 1,2

{(Bob, Ann) (Jim, Lea) (Tom, Sue)} is unstable

{(Bob, Ann) (Jim, Sue) (Tom, Lea)} is stable

Stable Marriage Algorithm (Gale-Shapley)

Step 0 Start with all the men and women being free

Step 1 While there are free men, arbitrarily select one of them and do the following:

Proposal The selected free man m proposes to w , the next woman on his preference list

Response If w is free, she accepts the proposal to be matched with m . If she is not free, she compares m with her current mate. If she prefers m to him, she accepts m 's proposal, making her former mate free; otherwise, she simply rejects m 's proposal, leaving m free

Step 2 Return the set of n matched pairs

Example

Free men:
Bob, Jim, Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Bob proposed to Lea
Lea accepted

Free men:
Jim, Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Jim proposed to Lea
Lea rejected

Free men:
Jim, Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Jim proposed to Sue
Sue accepted

Free men:
Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Tom proposed to Sue
Sue rejected

Free men:
Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Tom proposed to Lea
Lea replaced Bob
with Tom

Free men:
Bob

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Bob proposed to Ann
Ann accepted

Analysis of the Gale-Shapley Algorithm

- ❑ The algorithm terminates after no more than n^2 iterations with a stable marriage output
- ❑ The stable matching produced by the algorithm is always *man-optimal*: each man gets the highest rank woman on his list under any stable marriage. One can obtain the *woman-optimal* matching by making women propose to men
- ❑ A man (woman) optimal matching is unique for a given set of participant preferences
- ❑ The stable marriage problem has practical applications such as matching medical-school graduates with hospitals for residency training

Practice

EX #:4.6

Problems: 4.1-4.6, 4.9, 4.10, 4.13, 4.14, 4.15, 4.17, 4.20.

EX #: 5.5

Problems: 5.1-5.14, 5.17-5.19, 5.24, 5.25

Lec # 30, 31 & 32

Gale-Shapley Algorithm

Input: Preference rankings of n women and n men.

Steps:

1. Each man proposes to the highest ranking woman on his list.
2. If every woman receives only one proposal, this matching is stable. Otherwise move to Step (3).
3. If a woman receives more than one proposal, she
 - (a) accepts if it is from the man she prefers above all other currently available men and rejects the rest; or,
 - (b) delays with a maybe to the highest ranked proposal and rejects the rest.
4. Each man now proposes to the highest ranking unmatched woman on his list who has not rejected him.
5. Repeat Steps (2)–(4) until all people have been paired.

Output: Stable Matching.

Example 5.17 Four women are to be paired as roommates. Each woman has ranked the other three as shown below. Find all possible pairings and determine if any are stable.

Emma: l > m > z

Leena: m > e > z

Maggie: e > z > l

Zara: e > l > m

Solution: There are three possible pairings, only one of which is stable.

- Emma \leftrightarrow Leena and Maggie \leftrightarrow Zara

This is stable since Emma is with her first choice and the only person Leena prefers over Emma is Maggie, but Maggie prefers Zara over Leena.

- Emma \leftrightarrow Maggie and Leena \leftrightarrow Zara

This is not stable since Emma prefers Leena over Maggie and Leena prefers Emma over Zara.

- Emma \leftrightarrow Zara and Leena \leftrightarrow Maggie

This is not stable since Emma prefers Maggie over Zara and Maggie prefers Emma over Leena.

Example 5.18 Before the four women from Example 5.17 are paired as roommates, Maggie and Zara get into an argument, causing them to adjust their preference lists. Determine if a stable matching exists.

Emma: l > m > z

Leena: m > e > z

Maggie: e > l > z

Zara: e > l > m

Solution: There are three possible pairings, none of which are stable.

- Emma \leftrightarrow Leena and Maggie \leftrightarrow Zara

This is not stable since Leena prefers Maggie over Emma and Maggie prefers Leena over Zara.

- Emma \leftrightarrow Maggie and Leena \leftrightarrow Zara

This is not stable since Emma prefers Leena over Maggie and Leena prefers Emma over Zara.

- Emma \leftrightarrow Zara and Leena \leftrightarrow Maggie

This is not stable since Emma prefers Maggie over Zara and Maggie prefers Emma over Leena.

Factors & Factorization of the graph

Definition 5.19 Let G be a graph with spanning subgraph H and let k be a positive integer. Then H is a ***k-factor*** of G if H is a k -regular.

Examples of Peterson & K_4

Factor

- A **factor** of a graph G is a spanning subgraph of G , not necessarily connected.
- G is the sum of factors G_i , if:
 - G is the edge-disjoint union of G_i 's.
Such a union is called **factorization**.
- **n-factor**: A regular factor of degree n .
- If G is the sum of n -factors:
 - The union of n -factors is called **n-factorization**.
 - G is **n-factorable**.

1-factor

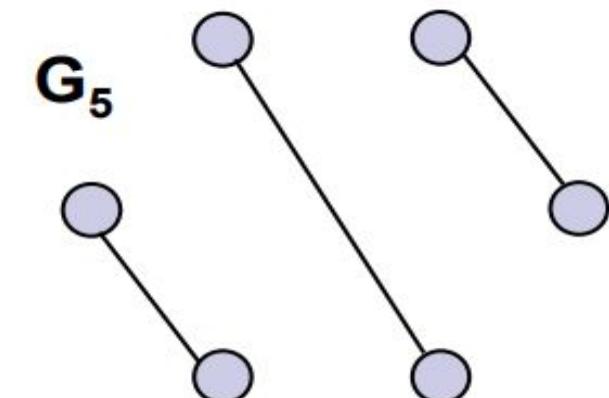
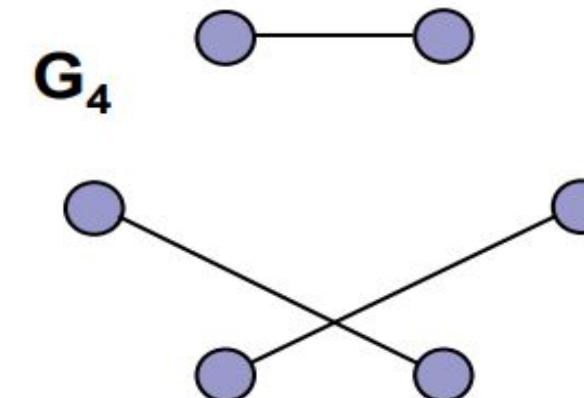
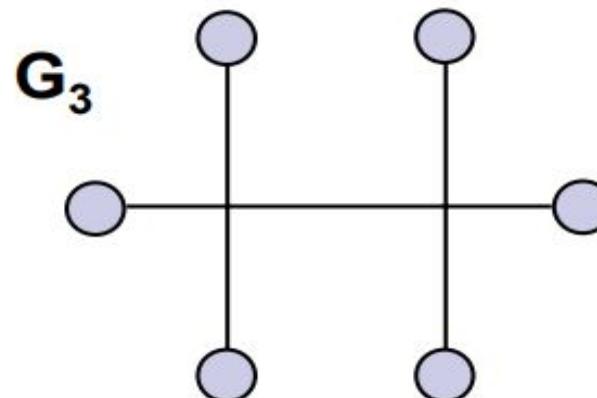
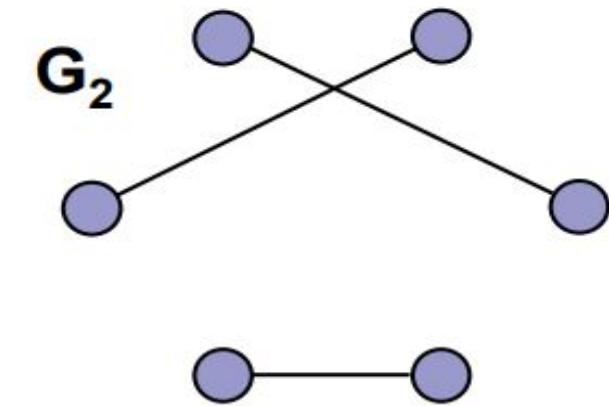
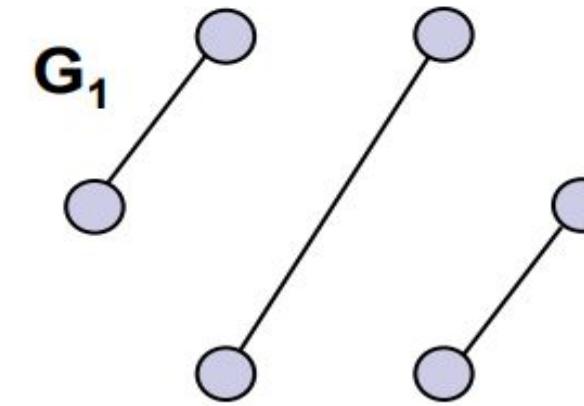
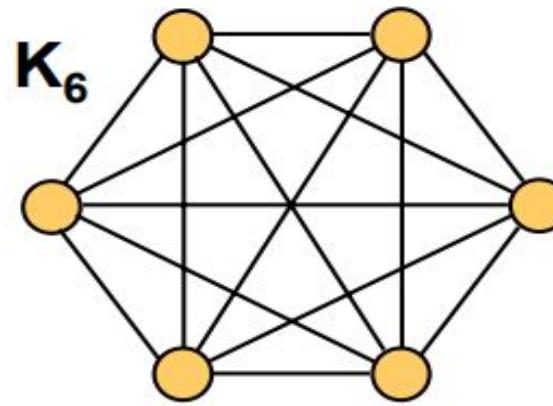
- When G has a 1-factor, G_1 ,
 - $|V|$ is even.
 - The edges of G_1 are edge disjoint.
- K_{2n+1} cannot have a 1-factor, but K_{2n} can.

Theorem: The complete graph K_{2n} is 1-factorable.

We need to display a partition of the set E of edges of K_{2n} into $(2n - 1)$ 1-factors.

- Denote the vertices: v_1, v_2, \dots, v_{2n}
- Define for $i = 1, 2, \dots, 2n - 1$
The sets $E_i = \{v_i v_{2n}\} \cup \{v_{i-j} v_{i+j} \mid j = 1, 2, n - 1\}$
 $i+1$ and $i-j$ are modulo($2n - 1$) operations.

Example

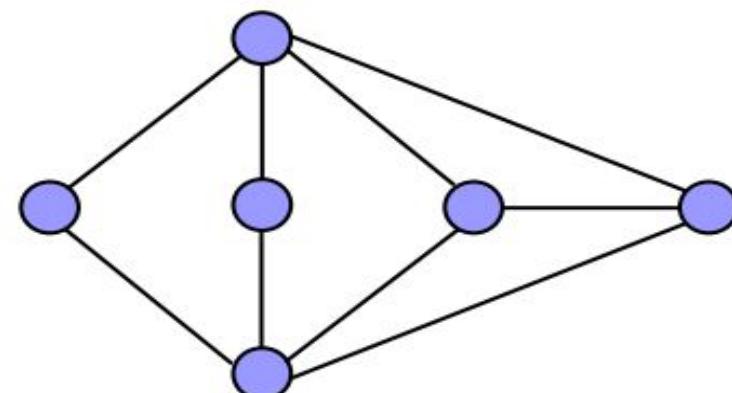
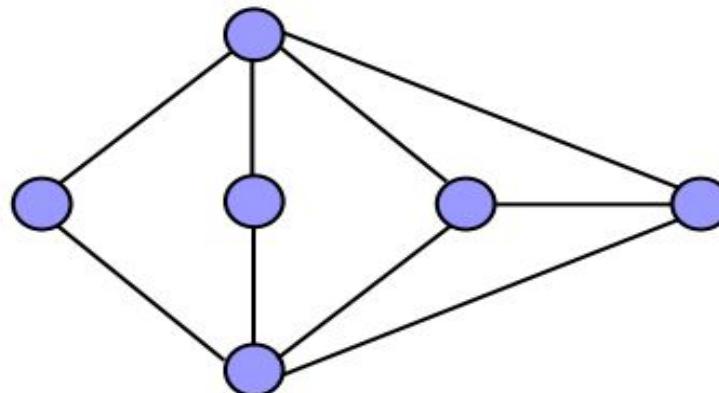


1-factors

- Complete bipartite graphs $K_{m,n}$ have no 1-factor if $n \neq m$.

Theorem: Every regular bipartite graph $K_{n,n}$ is 1-factorable.

Theorem: If a 2-connected graph has a 1-factor, then it has at least two different 1-factors.

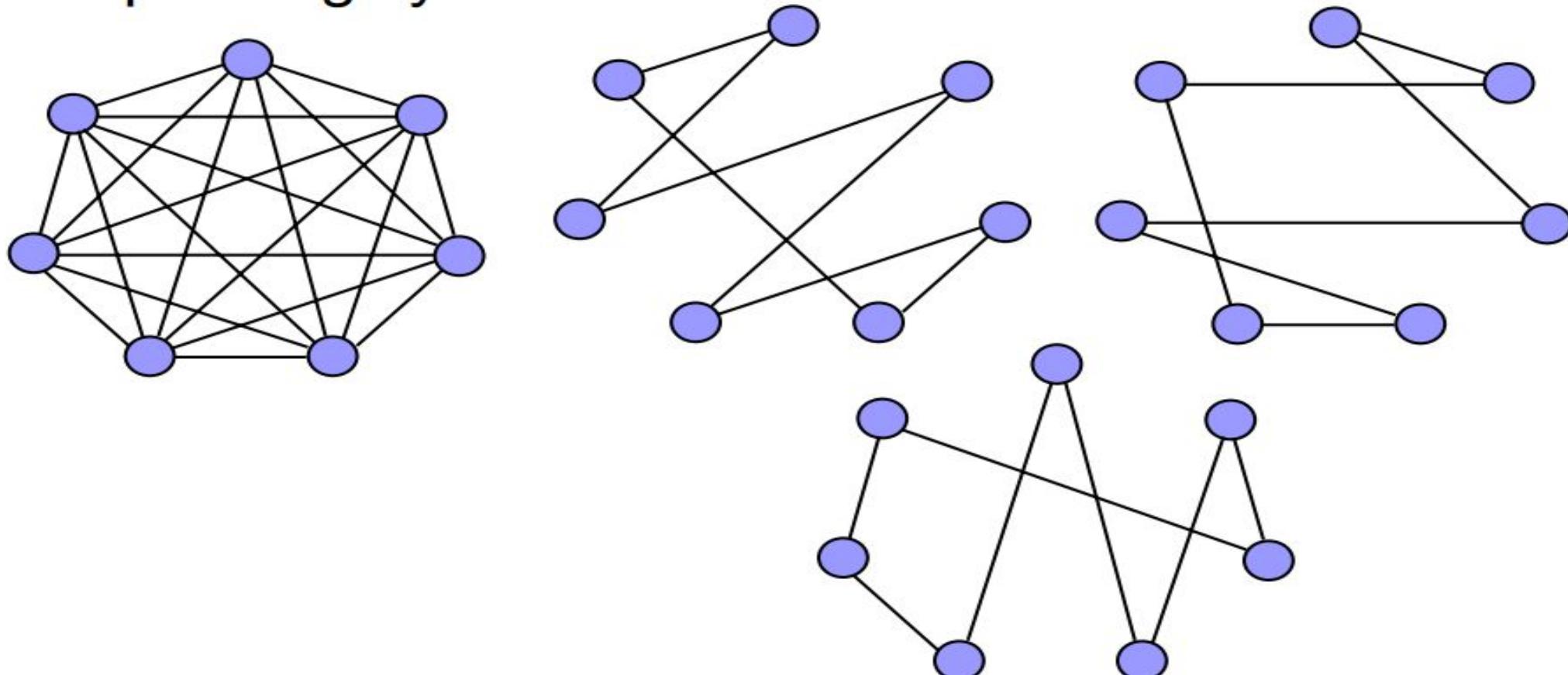


2-factorization

- If a graph is 2-factorable, then each factor is a union of disjoint cycles.
- If a 2-factor is connected, it is a spanning cycle.
- A 2-factorable graph must have all vertex degrees even.
- Complete graphs K_{2n} are not 2-factorable.
- K_{2n-1} complete graphs are 2-factorable.

2-factors

Theorem: The graph K_{2n+1} is the sum of n spanning cycles.



2-factors

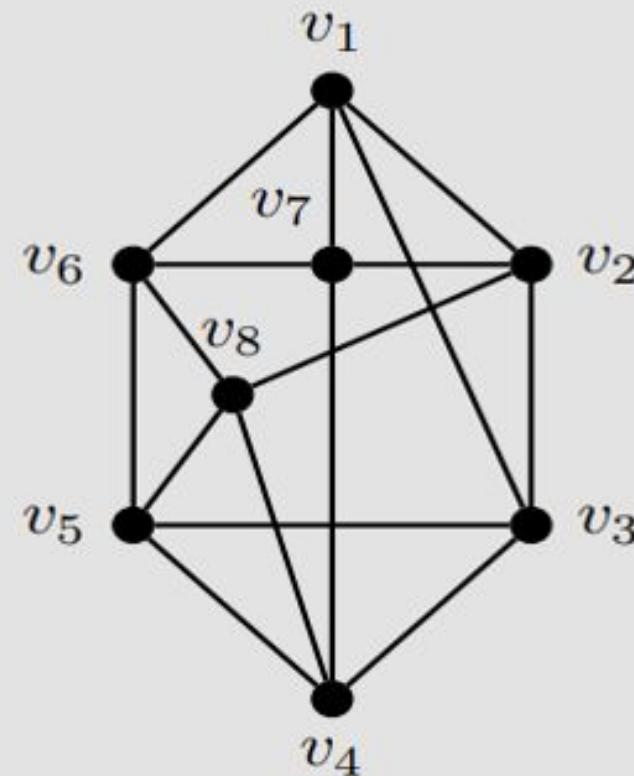
Theorem: The complete graph K_{2n} is the sum of a 1-factor and $n - 1$ spanning cycles.

- If every component of a regular graph G of degree 2 is an even-length cycle, then G is also 1-factorable.
 - It can be represented as the sum of two 1-factors.

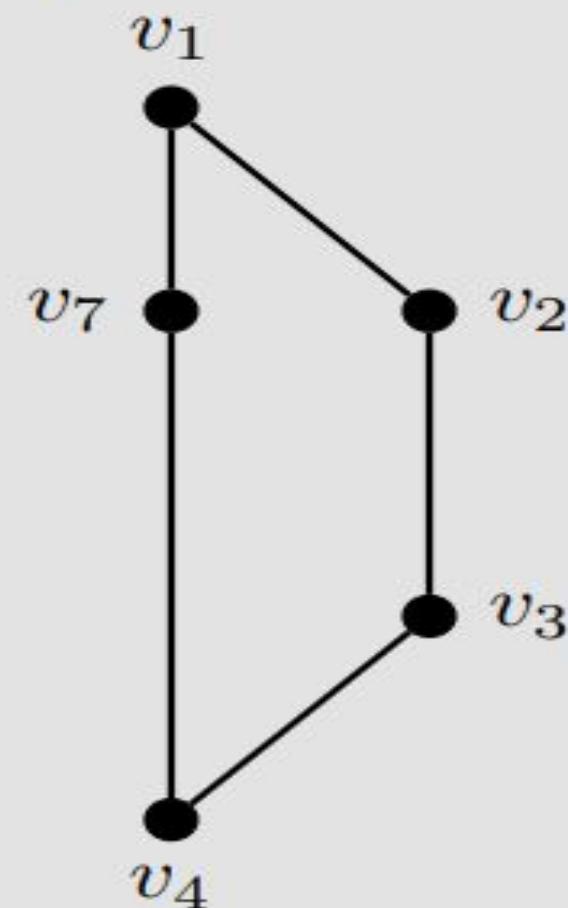
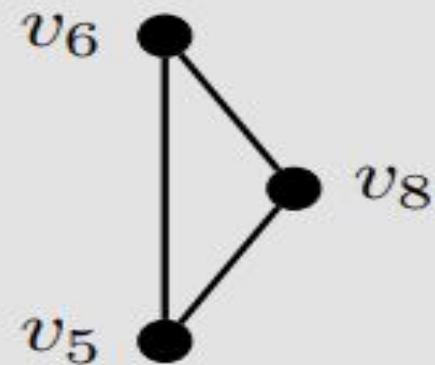
Theorem: Every bridgeless cubic graph is the sum of a 1-factor and a 2-factor.

- Example: Petersen graph.

Example 5.19 Find a 2-factor for the graph shown below.



Solution: More than one 2-factor exists for the graph above, as we simply need to find a spanning subgraph that is 2-regular. One such solution is given below. Note that this solution consists of two components, one of which is isomorphic to C_3 and the other to C_5 .



Do it by your own

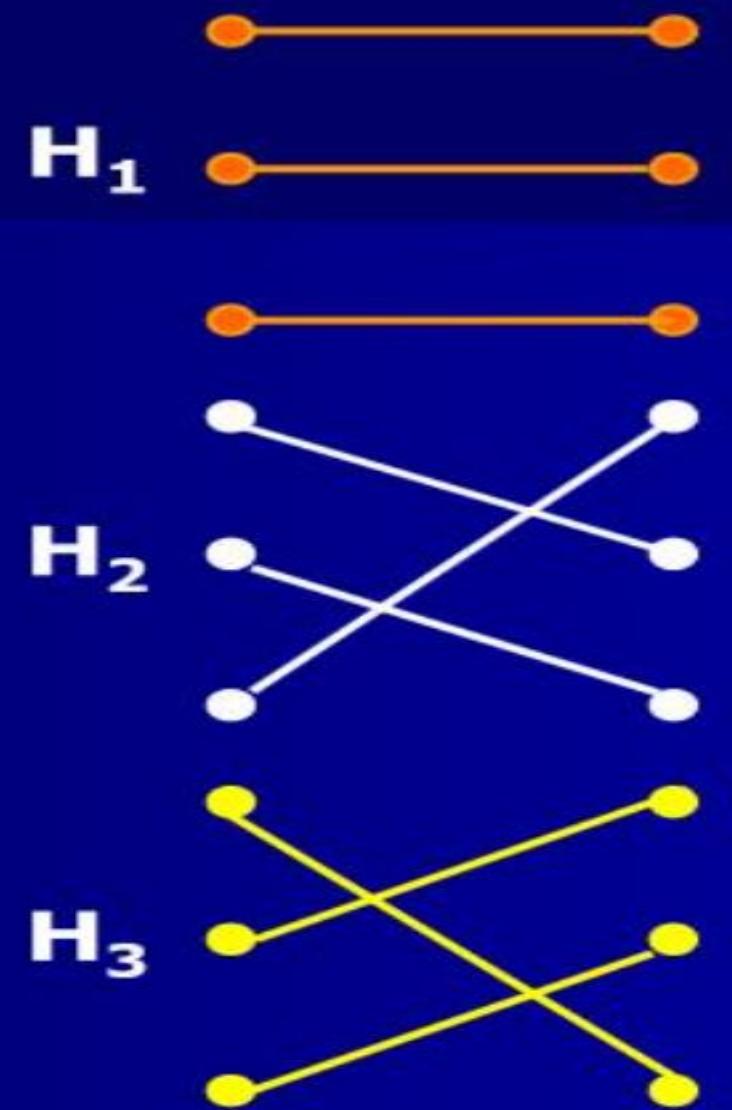
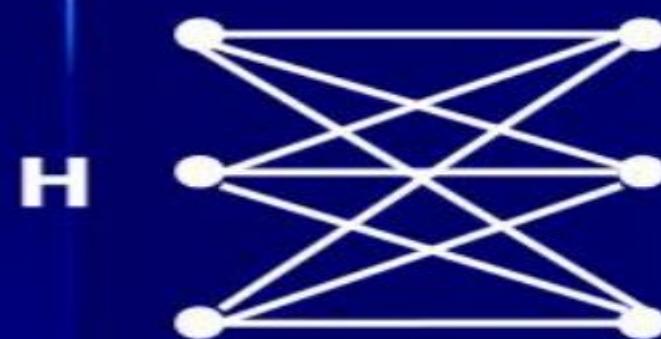
7.42 Show that an Eulerian graph cannot have a bridge.

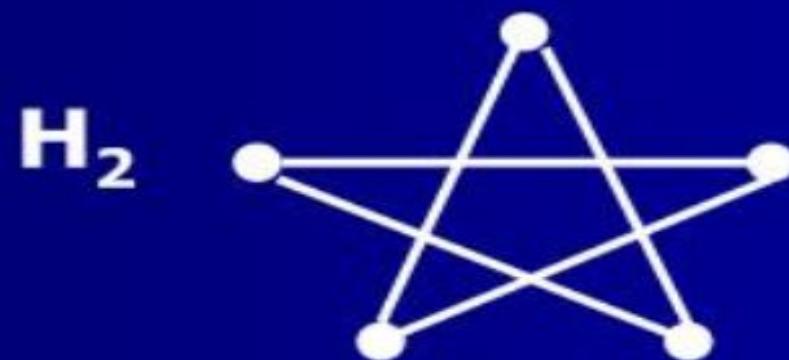
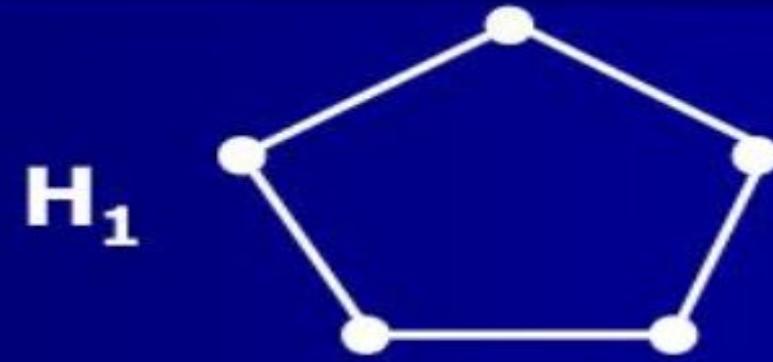
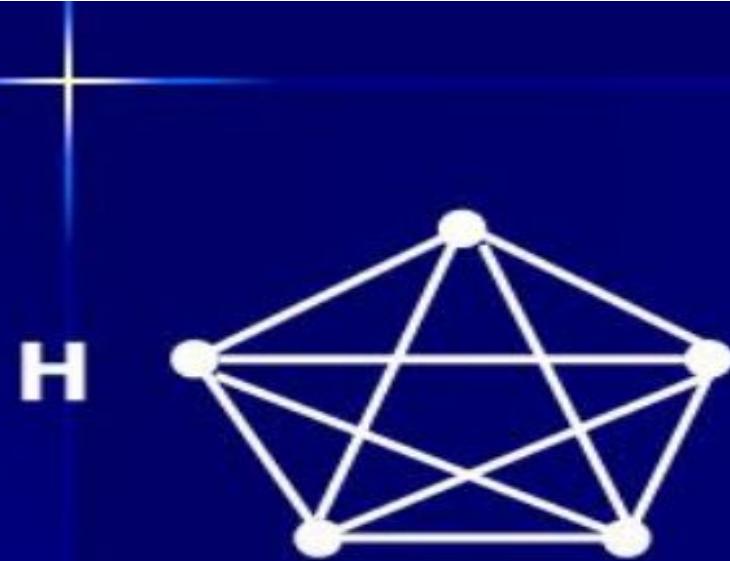
Solution. If a graph is Eulerian, it is a connected graph in which the degree of each vertex is even. Suppose it has a bridge. If this bridge is deleted, there will be two components, and in each component will be exactly one vertex of odd degree. But in any graph, the number of odd vertices is always even.

7.43 If a cubic graph has a bridge, it is not 1-factorable.

Solution. Suppose a cubic graph with a bridge is 1-factorable. The bridge will be in one of the three 3-factors. The removal of the bridge gives two components. Each component will have an even number of odd vertices and a vertex of degree 2. So there cannot be one more 1-factor since the number of vertices in each component is odd.

1-factorization of $K_{3,3}$





Theorem 5.20 If G is a $2k$ -regular graph, then G has a 2-factor.

Definition 5.21 A *k-factorization* of G is a partition of the edges into disjoint k -factors.

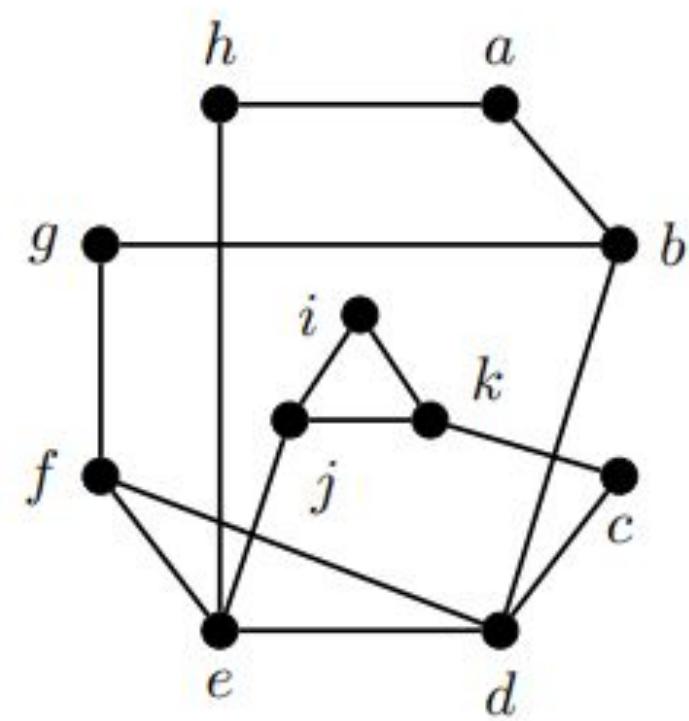
Proposition 5.22 Every k -regular bipartite graph has a 1-factorization for all $k \geq 1$.

Theorem 5.23 A graph G has a k -factorization if and only if G is $2k$ -regular.

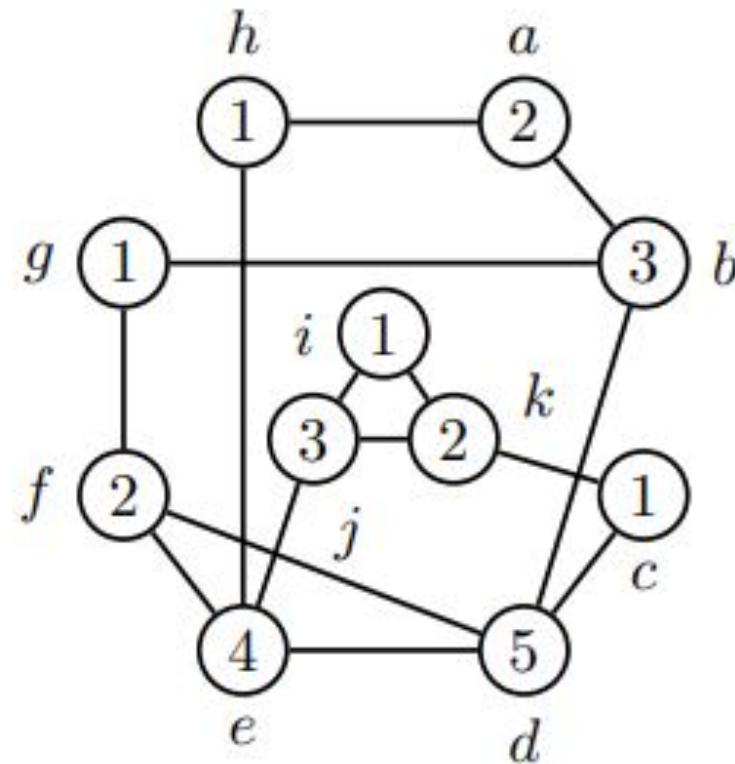
Lec # 34, 35 & 36

Graph Coloring

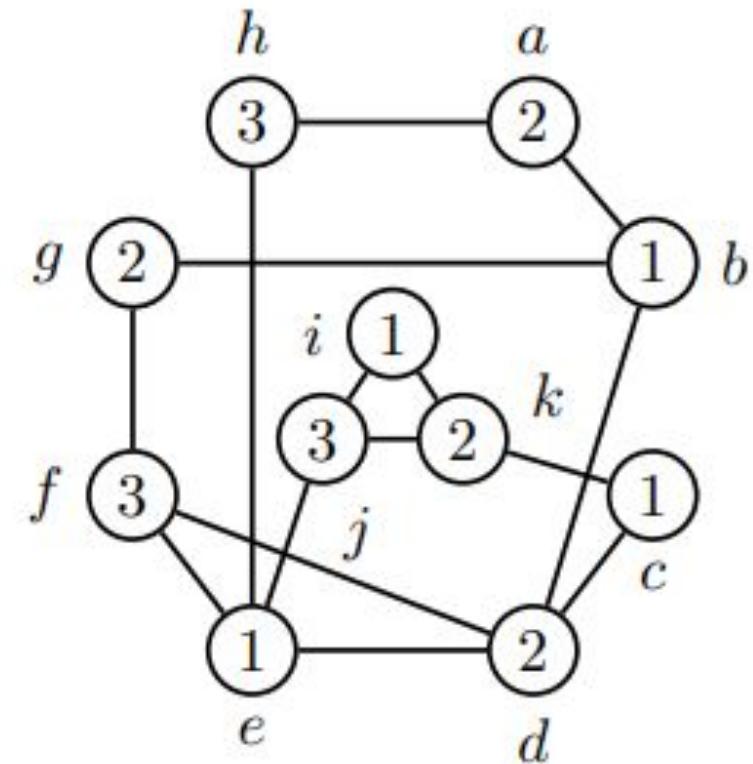
Definition 6.1 A proper *k-coloring* of a graph G is an assignment of colors to the vertices of G so that no two adjacent vertices are given the same color and exactly k colors are used.



G_1



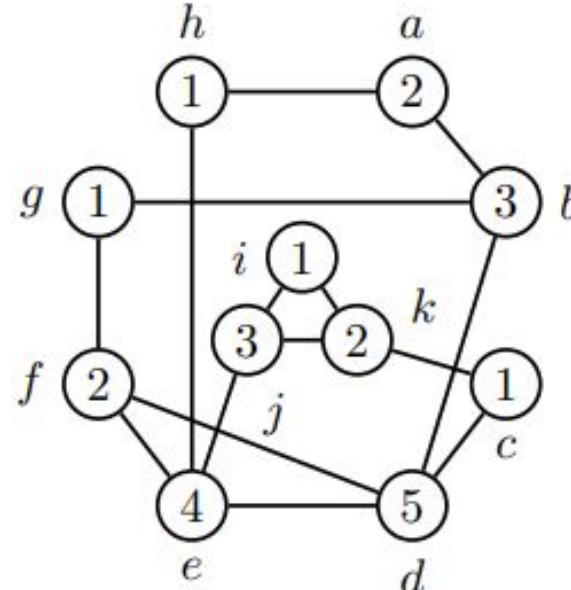
proper 5-coloring



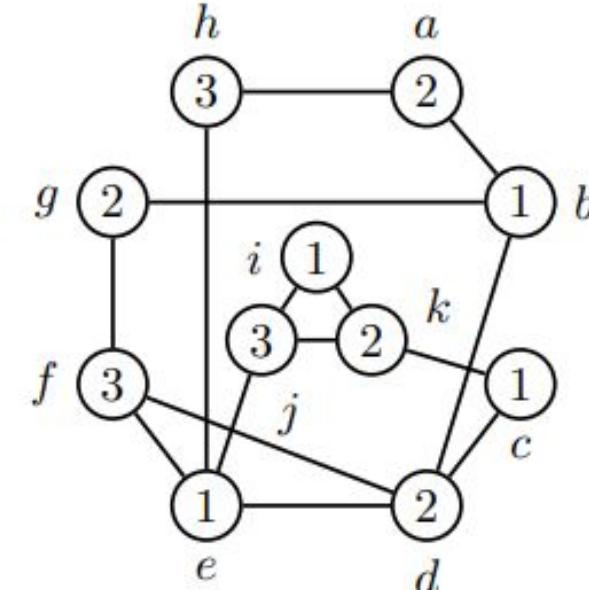
proper 3-coloring

Definition 6.2 Given a proper k -coloring of G , the ***color classes*** are sets S_1, \dots, S_k where S_i consists of all vertices of color i .

Definition 6.3 The ***independence number*** of a graph G is $\alpha(G) = n$ if there exists a set of n vertices with no edges between them but every set of $n + 1$ vertices contains at least one edge.



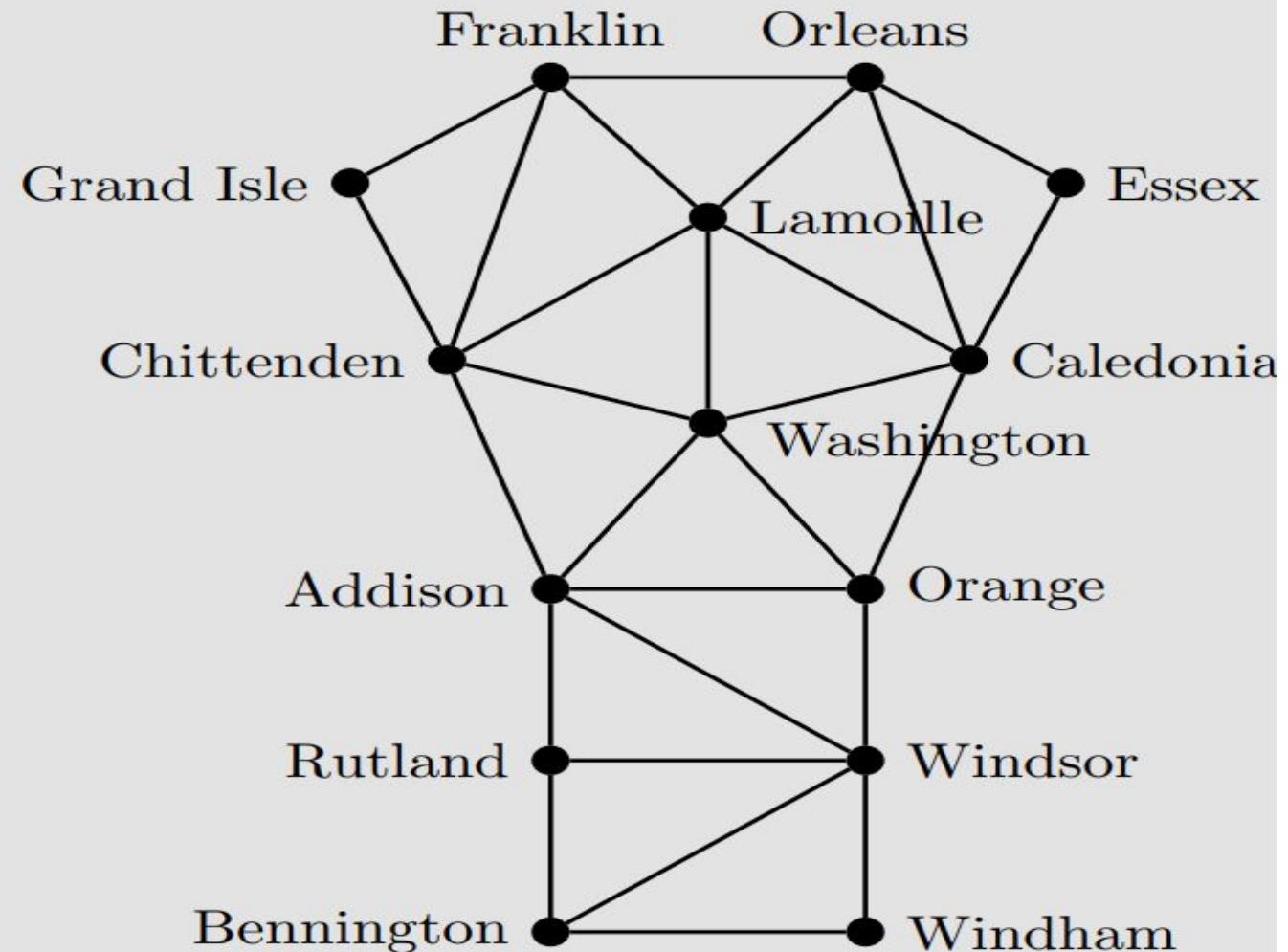
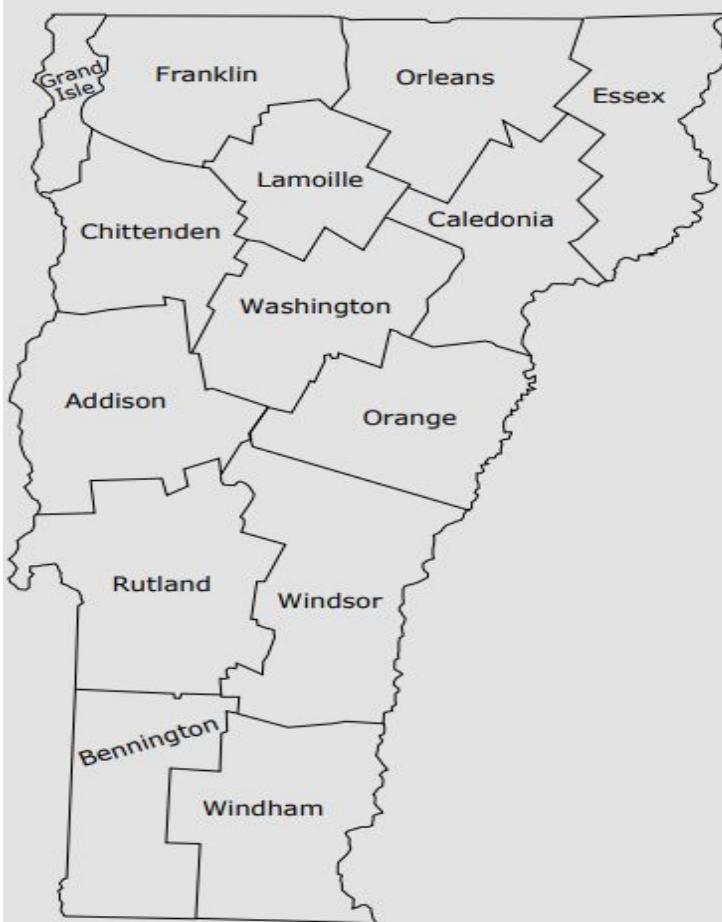
proper 5-coloring

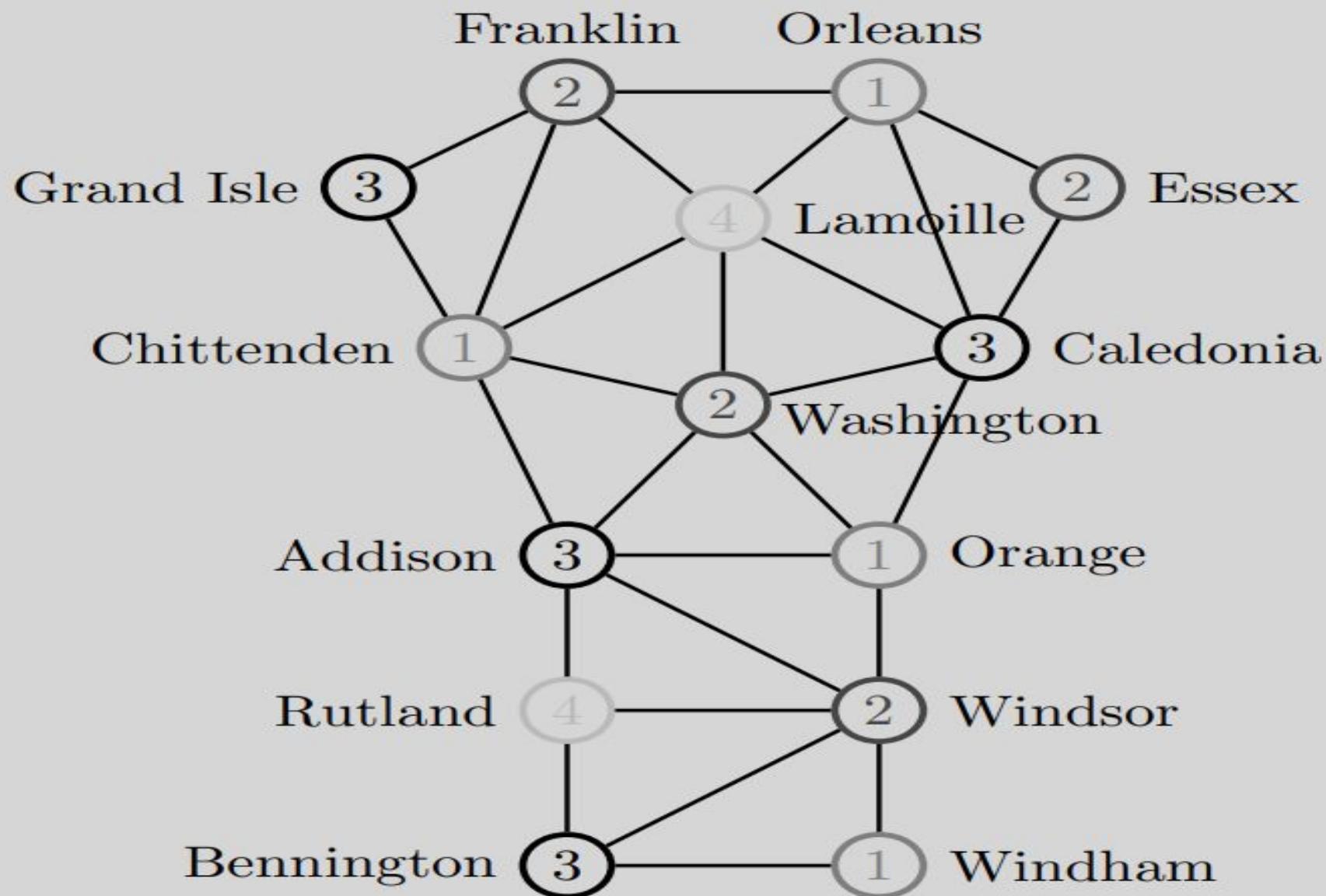


proper 3-coloring

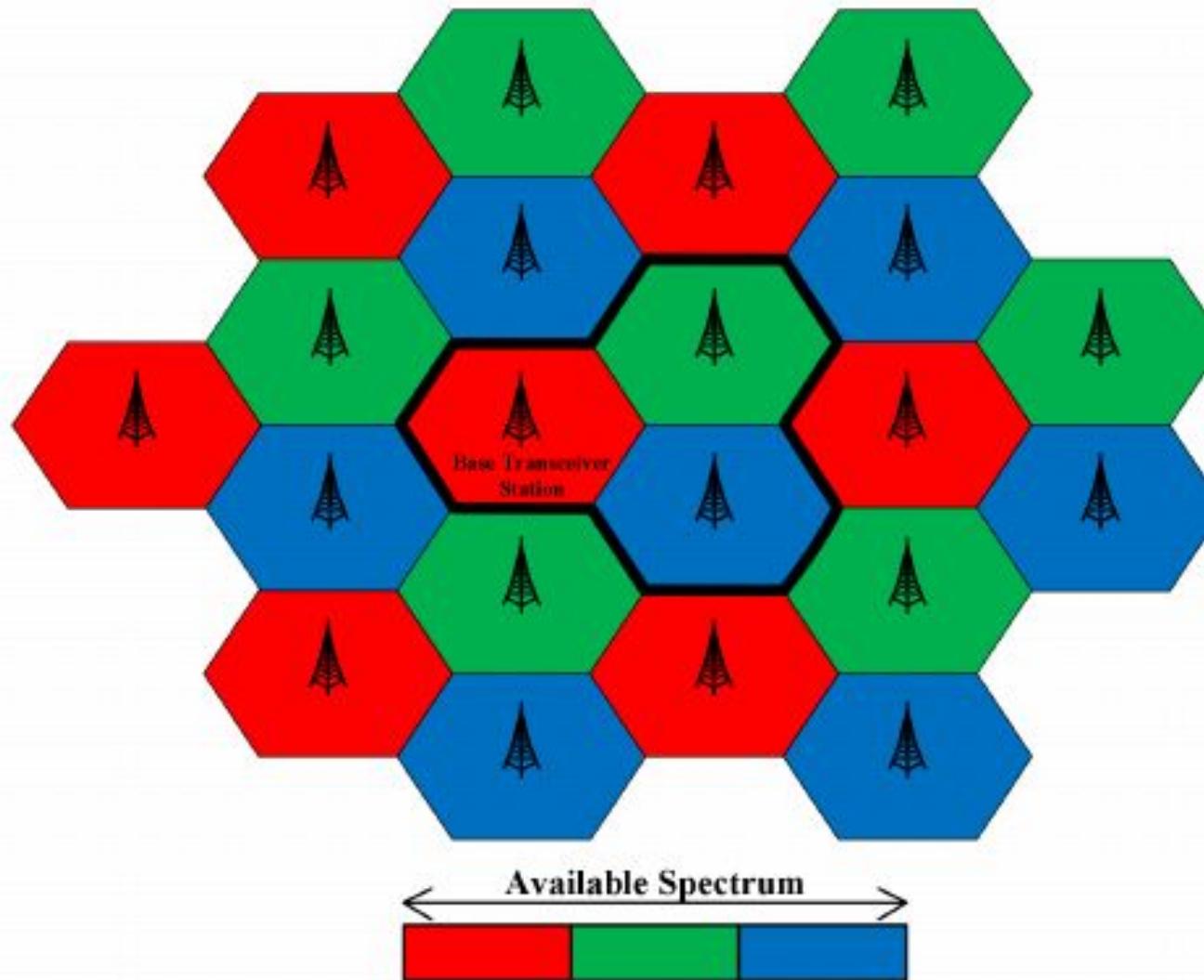
The first coloring of G_1 has color classes $S_1 = \{c, g, h, i\}$, $S_2 = \{a, f, k\}$, $S_3 = \{b, j\}$, $S_4 = \{e\}$, $S_5 = \{d\}$ and the second coloring has color classes $S_1 = \{b, c, e, i\}$, $S_2 = \{a, d, g, k\}$, $S_3 = \{f, h, j\}$. Recall that two vertices are independent if they have no edges between them. Thus a color class must consist of independent vertices. We will see there is a relationship between independent sets and coloring a graph.

Example 6.1 Find a coloring of the map of the counties of Vermont and explain why three colors will not suffice.

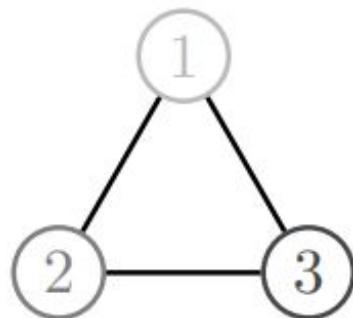




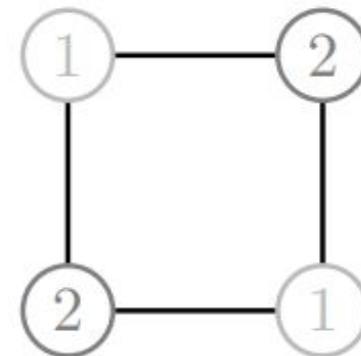
Note that Lamoille County is surrounded by five other counties. If we try to alternate colors amongst these five counties, for example Orleans – 1, Franklin – 2, Chittenden – 1, Washington – 2, we still need a third color for the fifth county (Caledonia – 3). Since Lamoille touches each of these counties, we know it needs a fourth color.



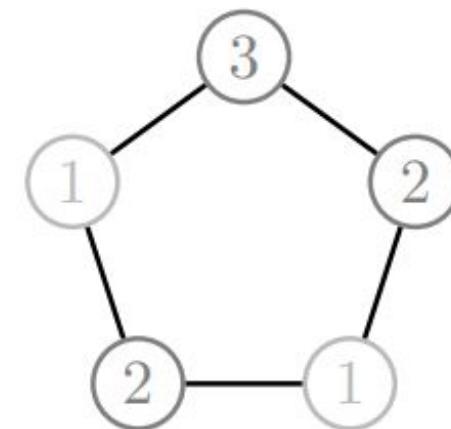
Definition 6.4 The *chromatic number* $\chi(G)$ of a graph is the smallest value k for which G has a proper k -coloring.



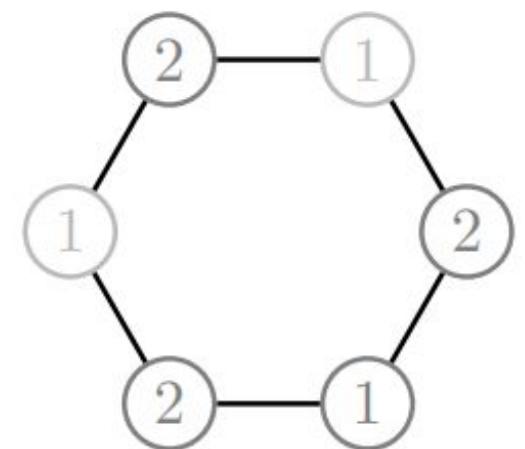
C_3



C_4

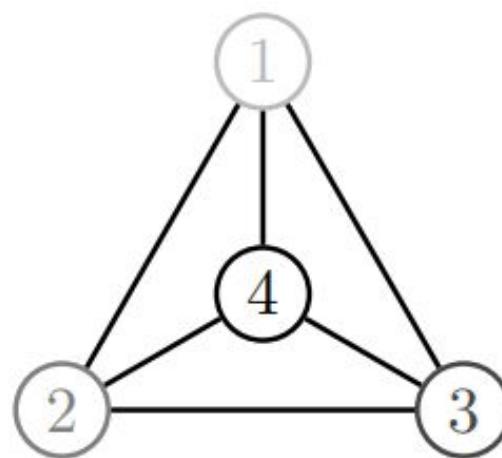


C_5

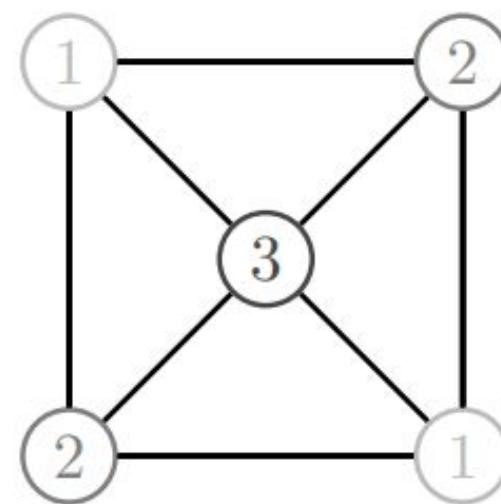


C_6

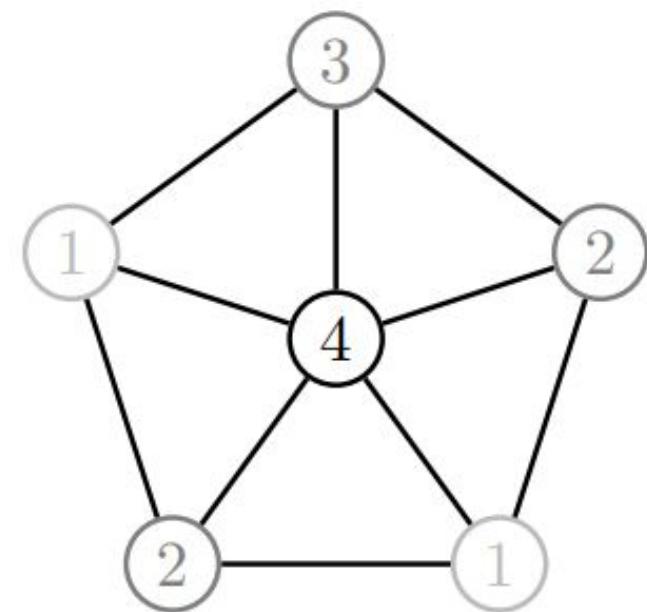
Definition 6.5 A *wheel* W_n is a graph in which n vertices form a cycle around a central vertex that is adjacent to each of the vertices in the cycle.



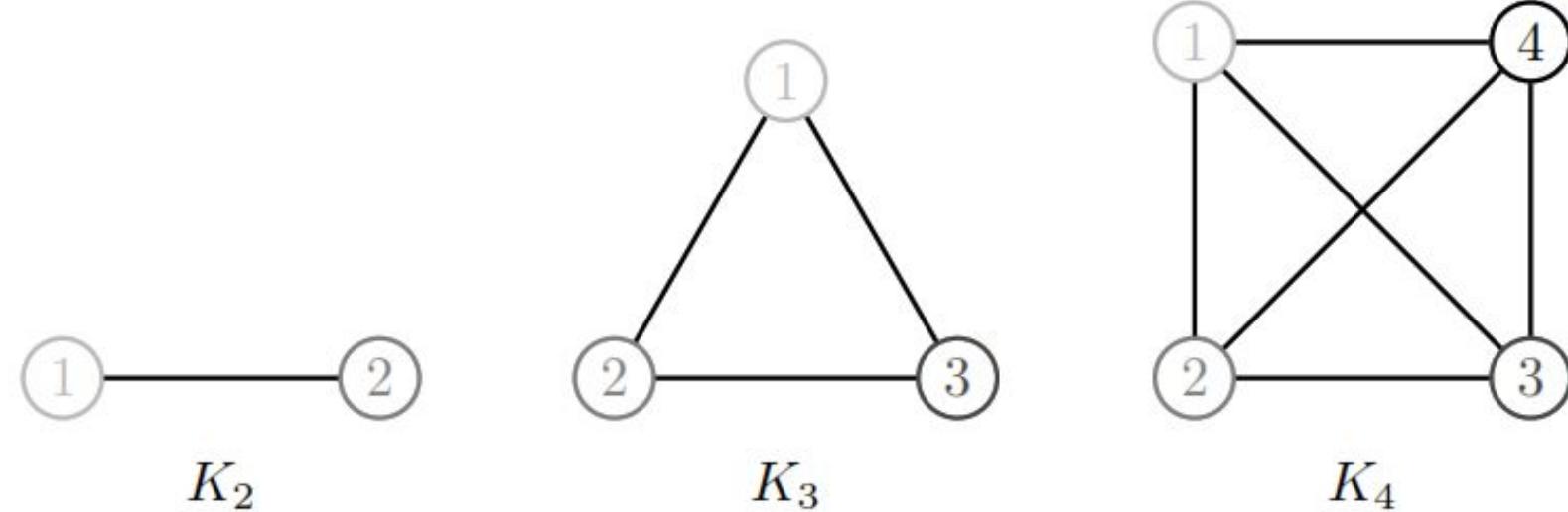
W_3



W_4



W_5



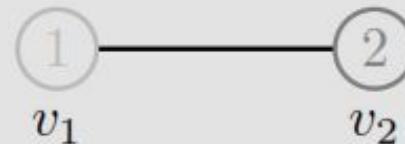
When a complete graph appears as a subgraph within a larger graph, we call it a *clique*.

Special Classes of Graphs with known $\chi(G)$

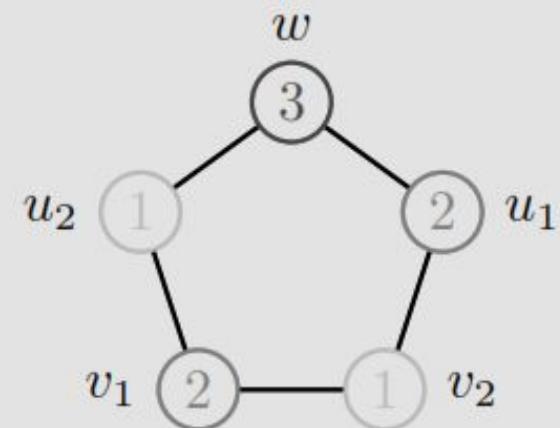
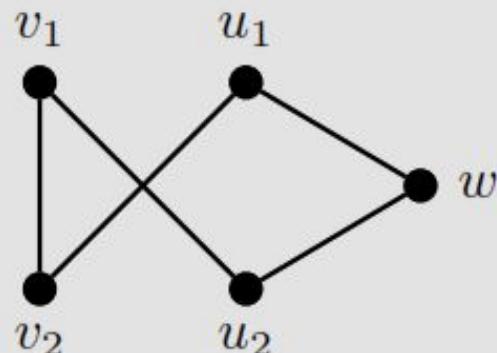
- $\chi(C_n) = 2$ if n is even ($n \geq 2$)
- $\chi(C_n) = 3$ if n is odd ($n \geq 3$)
- $\chi(K_n) = n$
- $\chi(W_n) = 4$ if n is odd ($n \geq 3$)

Example 6.2 Mycielski's Construction is a well-known procedure in graph theory that produces triangle-free graphs with increasing chromatic numbers. The idea is to begin with a triangle-free graph G where $V(G) = \{v_1, v_2, \dots, v_n\}$ and add new vertices $U = \{u_1, u_2, \dots, u_n\}$ so that $N(u_i) = N(v_i)$ for every i ; that is, add an edge from u_i to v_j whenever v_i is adjacent to v_j . In addition, we add a new vertex w so that $N(w) = U$; that is, add an edge from w to every vertex in U . The resulting graph will remain triangle-free but need one more color than G . If you perform enough iterations of this procedure, you can obtain a graph with $\omega(G) = 2$ and $\chi(G) = k$ for any desired value of k .

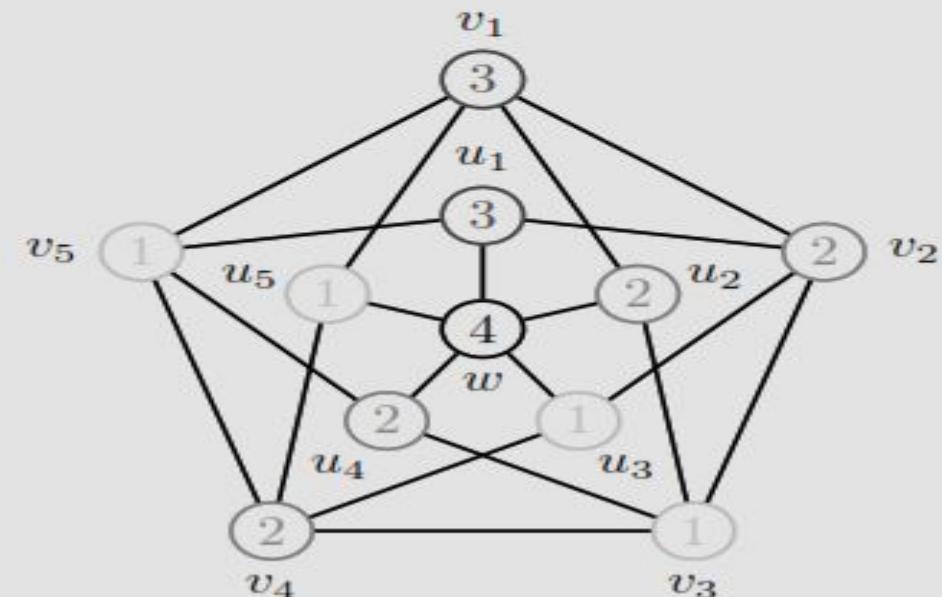
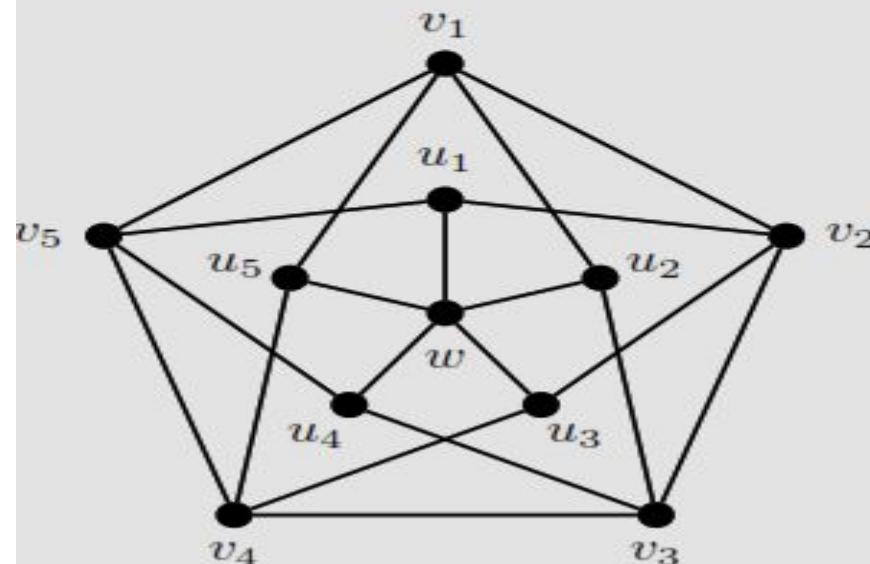
Consider G to be the complete graph on two vertices, K_2 , which is clearly triangle free and has chromatic number 2, as shown in the following graph.



After the first iteration of Mycielski's Construction, we get the graph shown below on the left. Notice that u_1 has an edge to v_2 since v_1 is adjacent to v_2 . Similarly, u_2 has an edge to v_1 . In addition, w is adjacent to both u_1 and u_2 . The graph on the right below is an unraveling of the graph on the left. Thus we have obtained C_5 , which we know needs 3 colors.



After the second iteration, we obtain the graph shown below. The outer cycle on 5 vertices represents the graph obtained above in the first iteration. The inner vertices are the new additions to the graph, with u_1 adjacent to v_2 and v_5 since v_1 is adjacent to v_2 and v_5 . Similar arguments hold for the remaining u -vertices and the center vertex w is adjacent to all of the u -vertices. A coloring of the graph is shown below on the right. Note that the outer cycle needs 3 colors, as does the group of u -vertices. This forces w to use a fourth color. In addition, no matter which three vertices you choose, you cannot find a triangle among them, and so the graph remains triangle-free.



If we continue this procedure through one more step, we obtain a graph needing 5 colors with a clique size of 2.

Theorem 6.7 (Brooks' Theorem) Let G be a connected graph and Δ denote the maximum degree among all vertices in G . Then $\chi(G) \leq \Delta$ as long as G is not a complete graph or an odd cycle. If G is a complete graph or an odd cycle then $\chi(G) = \Delta + 1$.

Prove Theorem 9.1: $\chi(G) \leq \Delta(G) + 1$ for any graph G .

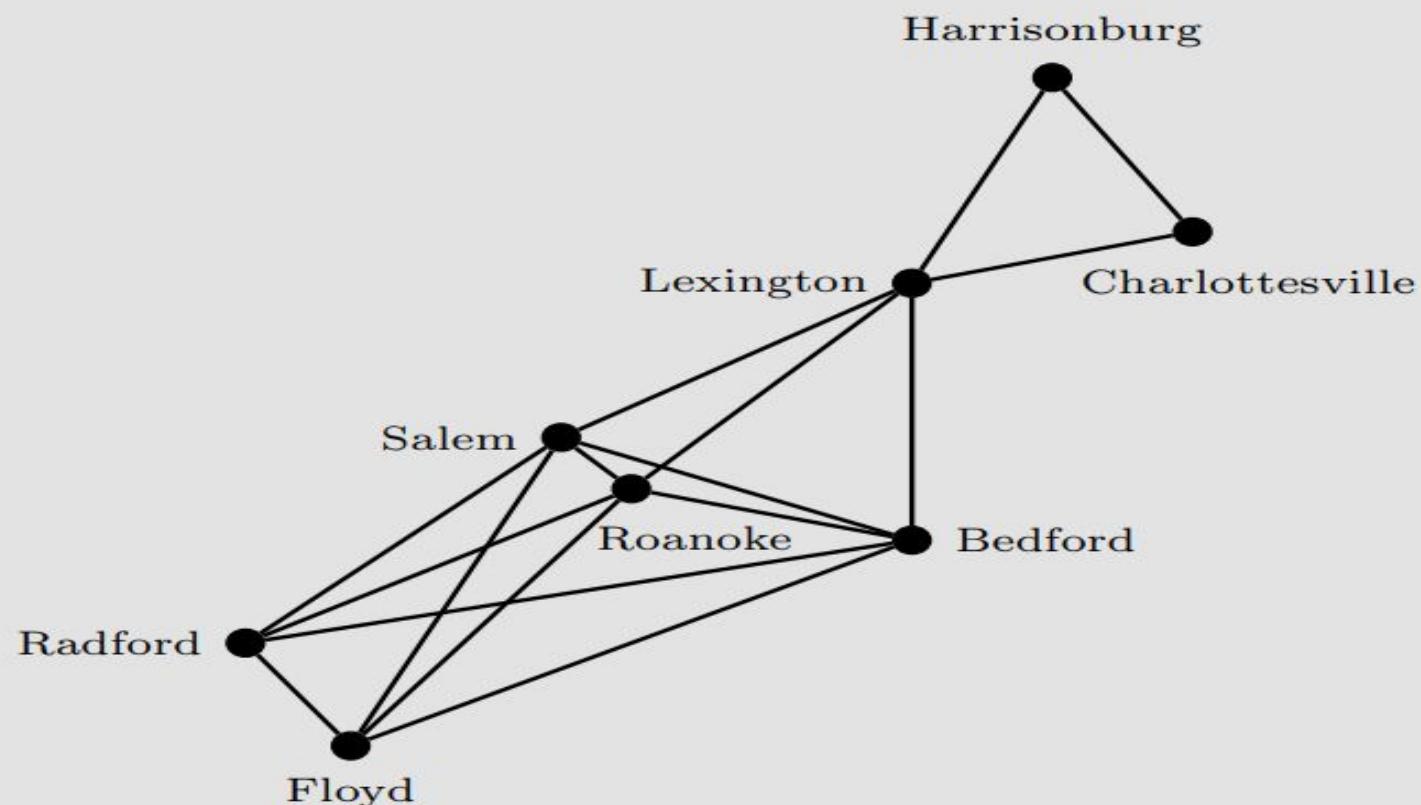
Solution. Let n be number of vertices of G . The proof is by induction on n . Let G' be the graph obtained by deleting any vertex v from G . Then $\Delta(G') \leq \Delta(G)$. By the induction hypothesis, $\chi(G') \leq \Delta(G') + 1 \leq \Delta(G) + 1$. We can assign a color to v (in G) that is different from the colors (assigned in G') to the vertices adjacent to it. In that case, we do not need more than $\Delta(G) + 1$ colors to color the vertices of G since the number of vertices adjacent to v is at most $\Delta(G)$. Hence, $\chi(G) \leq \Delta(G) + 1$.



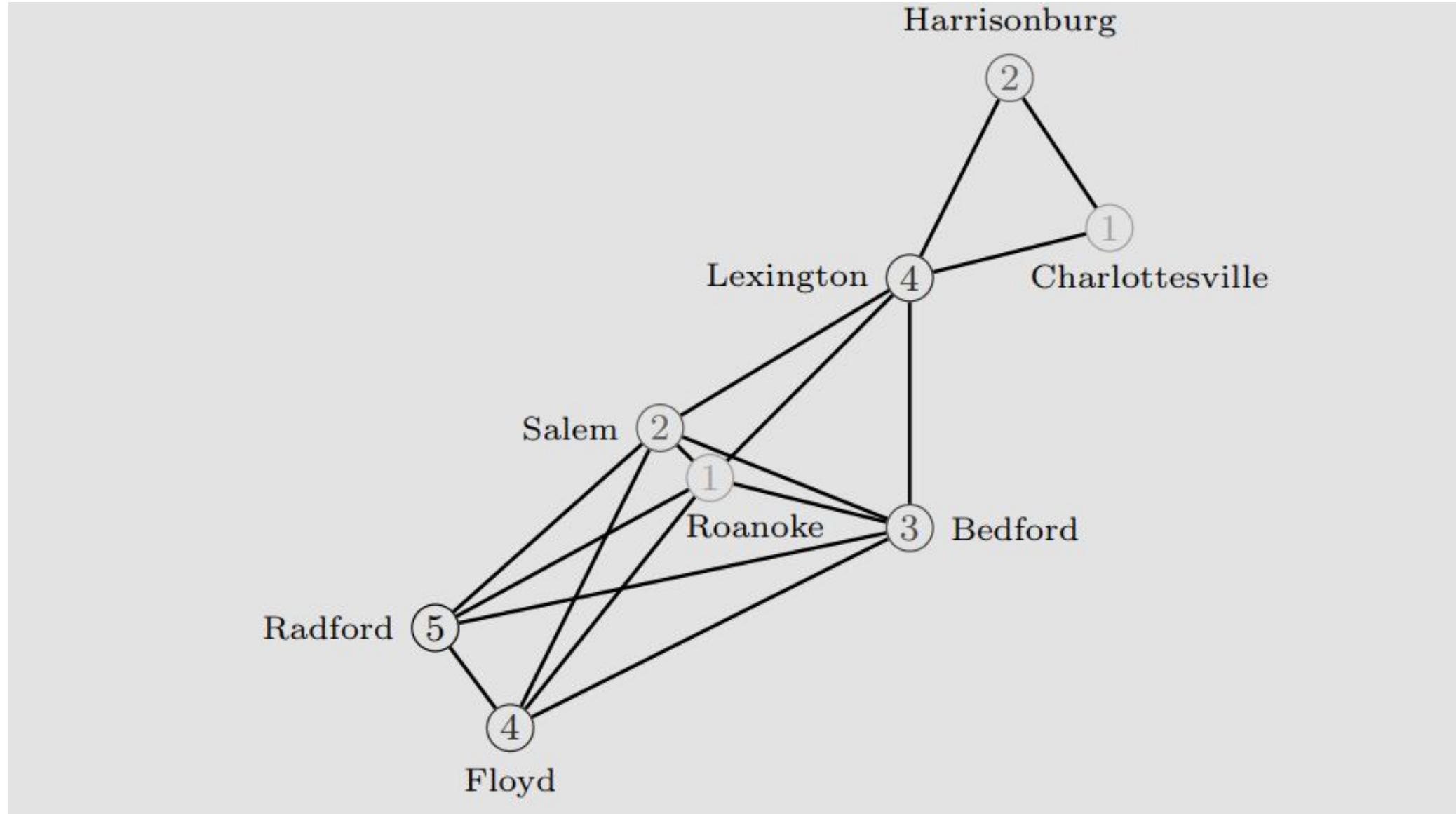
Basic Coloring Strategies

- Begin with vertices of high degree.
- Look for locations where colors are forced (cliques, wheels, odd cycles) rather than chosen.
- When these strategies have been exhausted, color the remaining vertices while trying to avoid using any additional colors.

Example 6.5 Due to the nature of radio signals, two stations can use the same frequency if they are at least 70 miles apart. An edge in the graph below indicates two cities that are at most 70 miles apart, necessitating different radio stations. Determine the fewest number of frequencies need for each city shown below (not drawn to scale) to have its own municipal radio station.



Solution: Each vertex will be assigned a color that corresponds to a radio frequency. This graph has $\chi = 5$ since we have a 5-coloring, as shown below, and fewer than 5 colors will not suffice as there is a K_5 among the vertices representing the cities of Roanoke, Salem, Bedford, Floyd, and Radford.



Proposition 6.9 Let G be a graph with m edges. Then

$$\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$$

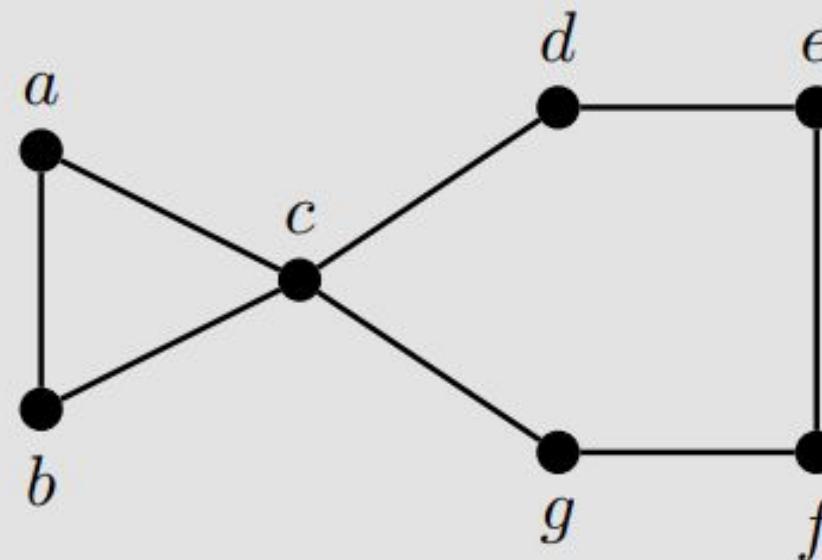
Proposition 6.10 Let G be a graph and $l(G)$ be the length of the longest path in G . Then $\chi(G) \leq 1 + l(G)$.

Definition 6.11 Given a graph $G = (V, E)$, an *induced subgraph* is a subgraph $G[V']$ where $V' \subseteq V$ and every available edge from G between the vertices in V' is included.

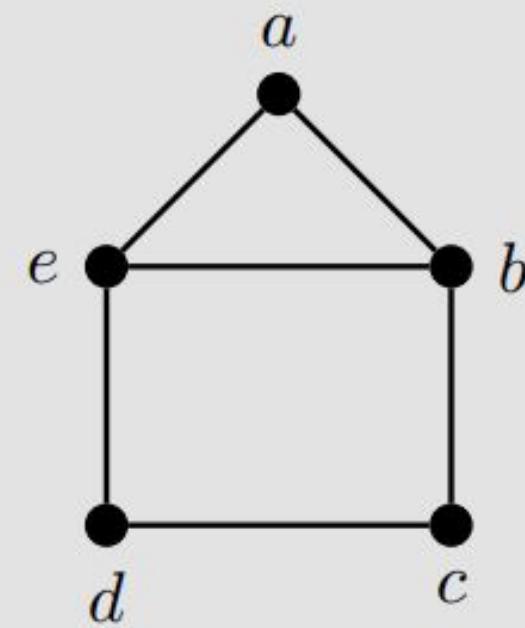
Definition 6.13 A graph G is *perfect* if and only if $\chi(H) = \omega(H)$ for all induced subgraphs H .

Definition 1.5 Given a graph $G = (V, E)$, an *induced subgraph* is a subgraph $G[V']$ where $V' \subseteq V$ and every available edge from G between the vertices in V' is included.

Example 6.6 Determine if either of the two graphs below are perfect.



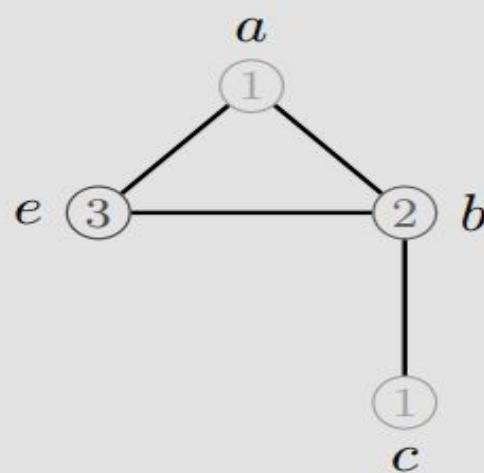
G_2



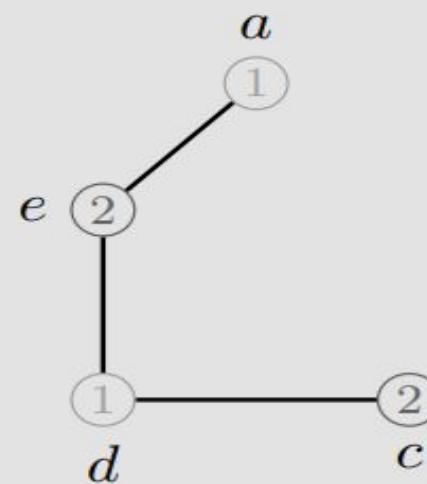
G_3

Solution: Without much work we can see that both graphs above satisfy $\chi(G) = \omega(G)$. However, if we look at the subgraph H induced by $\{c, d, e, f, g\}$ in G_2 we see that H is just C_5 and so $\chi(H) = 3$ even though $\omega(H) = 2$. Thus G_2 is not perfect.

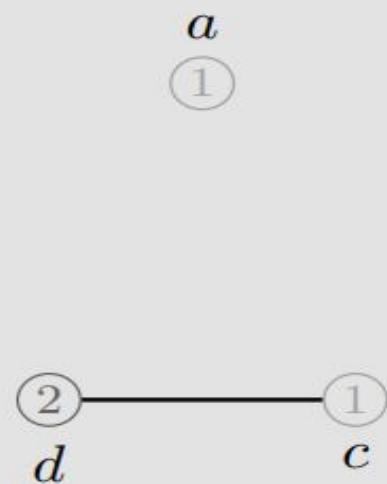
However, G_3 is in fact perfect. If an induced subgraph H contains $\{a, b, e\}$, then $\omega(H) = 3 = \chi(H)$; otherwise one of $\{a, b, e\}$ will not be in H and so $\omega(H) \leq 2$ and without much difficulty we can show $\omega(H) = \chi(H)$. A few illustrative induced subgraphs are shown below.



$G_3[a, b, c, e]$



$G_3[a, c, d, e]$



$G_3[a, c, d]$

Theorem 6.14 A graph G is perfect if and only if \overline{G} is perfect.

Theorem 6.15 A graph G is perfect if and only if no induced subgraph of G or \overline{G} is an odd cycle of length at least 5.

Perfect Graphs

The following classes of graphs are known to be perfect:

- Trees
- Bipartite graphs
- Chordal graphs
- Interval graphs

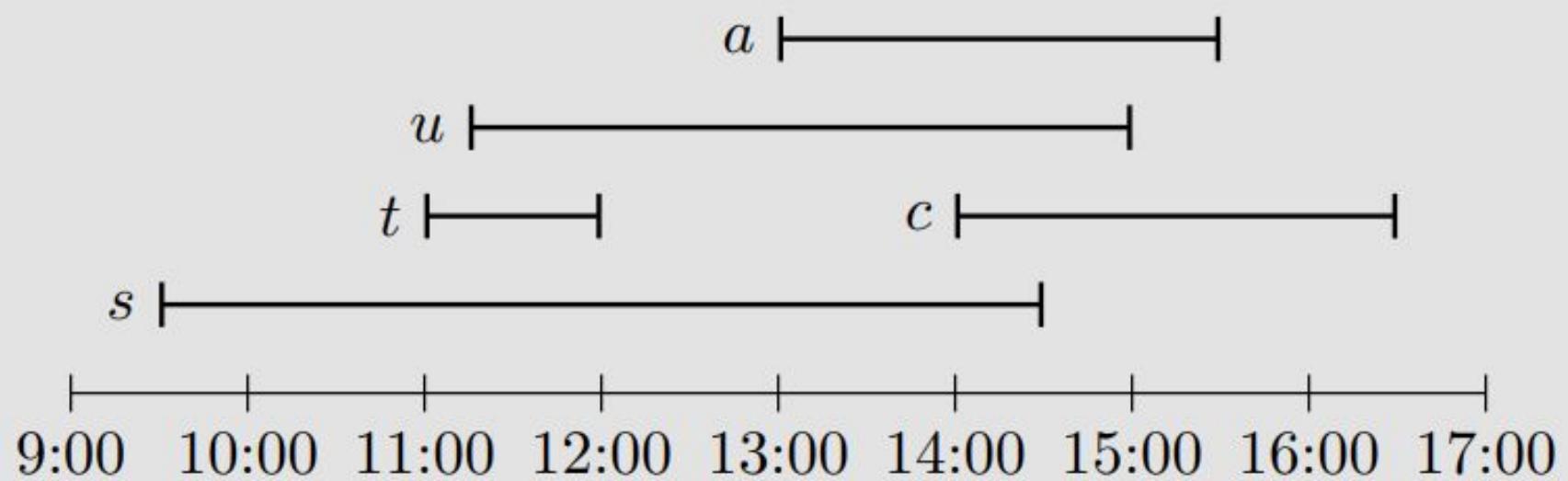
Definition 6.16 A graph G is *chordal* if any cycle of length four or larger has an edge (called a chord) between two nonconsecutive vertices of the cycle.

Definition 6.17 A graph G is an *interval graph* if every vertex can be represented as a finite interval and two vertices are adjacent whenever the corresponding intervals overlap; that is, for every vertex x there exists an interval I_x and xy is an edge in G if $I_x \cap I_y \neq \emptyset$.

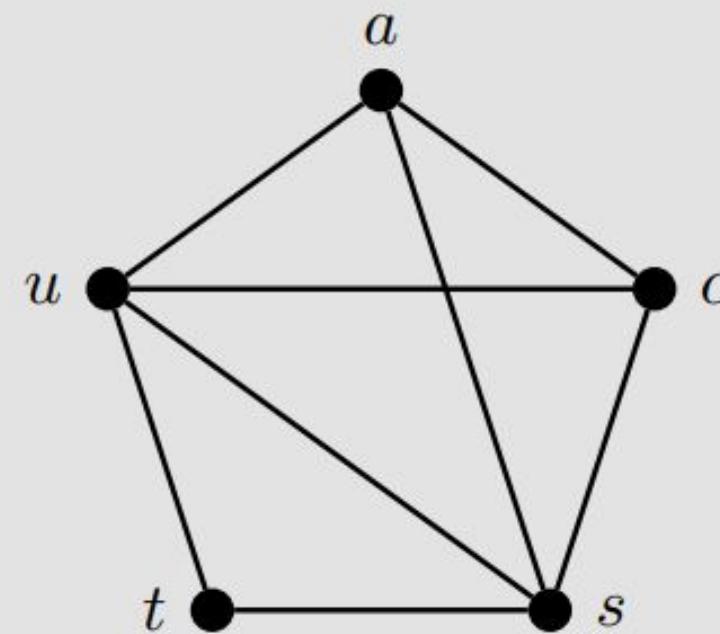
Example 6.7 Five student groups are meeting on Saturday, with varying time requirements. The staff at the Campus Center need to determine how to place the groups into rooms while using the fewest rooms possible. The times required for these groups is shown in the table below. Model this as a graph and determine the minimum number of rooms needed.

Student Group	Meeting Time
Agora	13:00–15:30
Counterpoint	14:00–16:30
Spectrum	9:30–14:30
Tupelos	11:00–12:00
Upstage	11:15–15:00

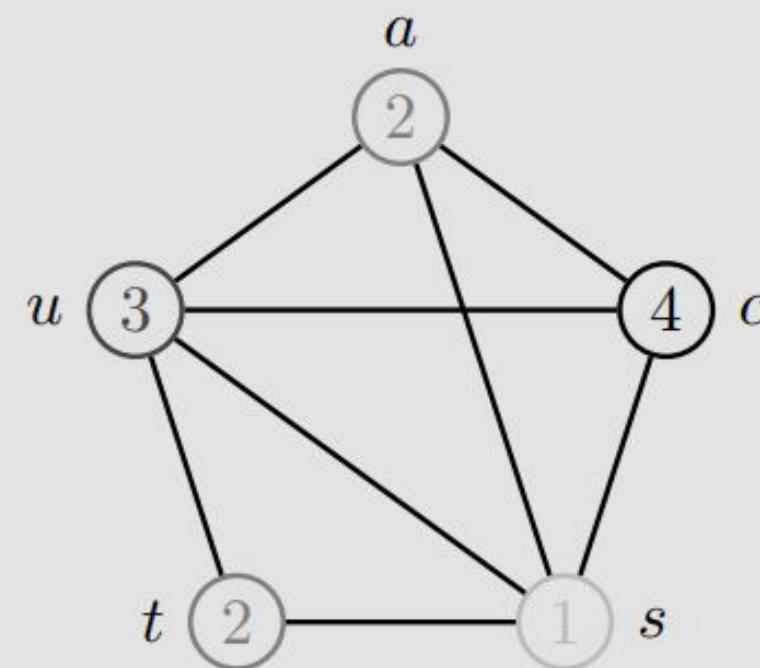
Solution: First we display the information in terms of the intervals. Although this step is not necessary, sometimes the visual aids in determining which vertices are adjacent.



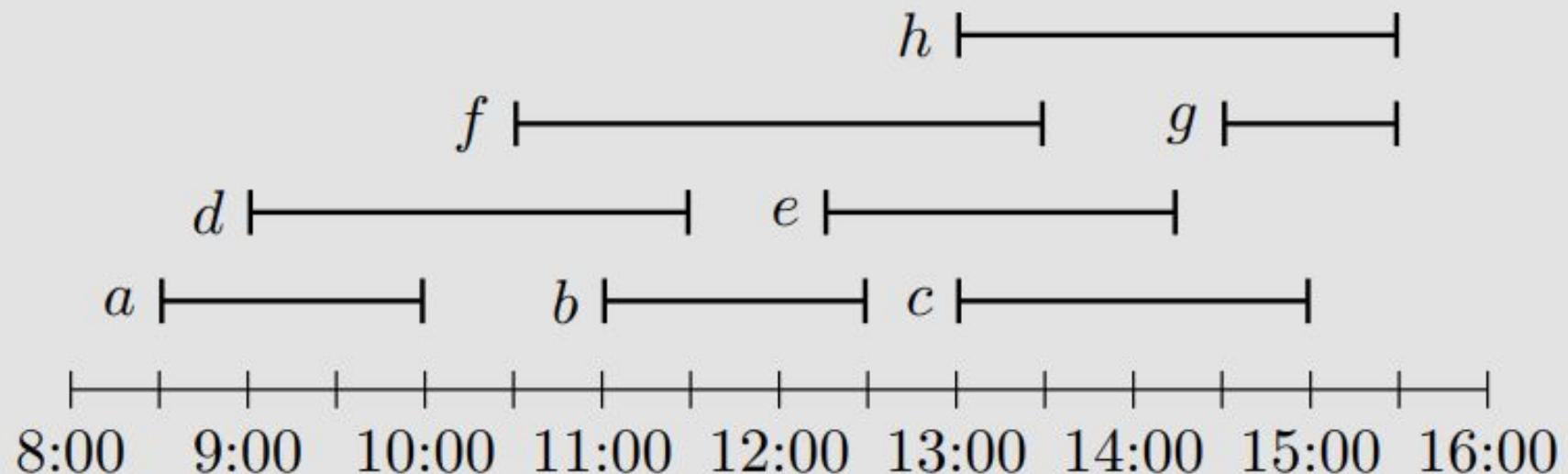
Below is the graph where each vertex represents a student group and two vertices are adjacent if their corresponding intervals overlap.



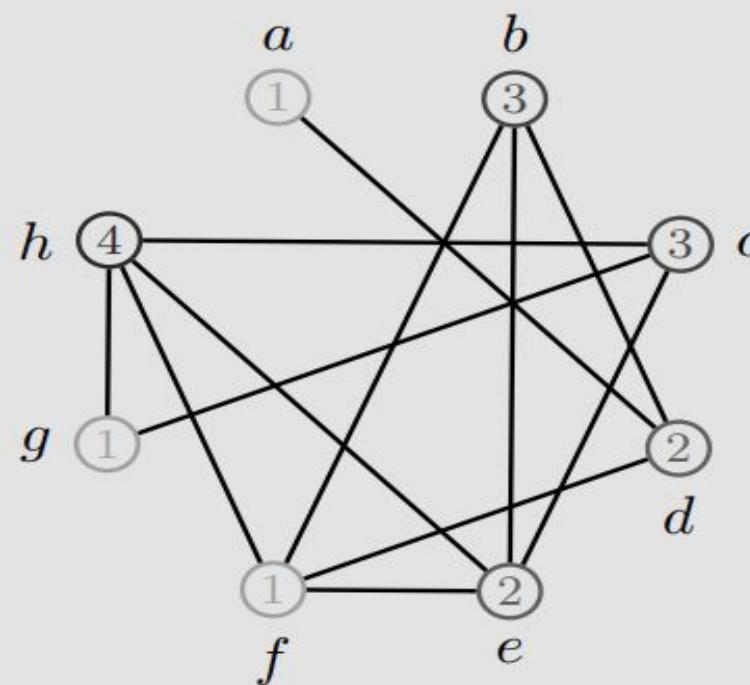
A proper coloring of this graph is shown below. Note that four colors are required since there is a K_4 subgraph with a, c, s , and u .



Example 6.8 Eight meetings must occur during a conference this upcoming weekend, as noted below. Determine the minimum number of rooms that must be reserved.



Solution: Each meeting is represented by a vertex, with an edge between meetings that overlap and colors indicating the room in which a meeting will occur. If we color the vertices according to their start time (so in the order a, d, f, b, e, c, h, g), we get the coloring below.



Note that four meeting rooms are needed since there is a point at which four meetings are all in session, which is demonstrated by the K_4 among the vertices c, e, f , and h .

Lec # 38, 39 & 40

Edge Coloring

This section focuses on a different aspect of graph coloring where instead of assigning colors to the vertices of a graph, we will instead ***assign colors to the edges of a graph.*** Such colorings are called edge-colorings and have their own set of definitions and notations, many of which are analogous to those for vertex colorings from above

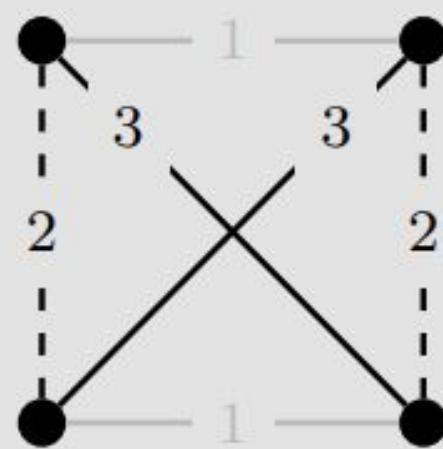
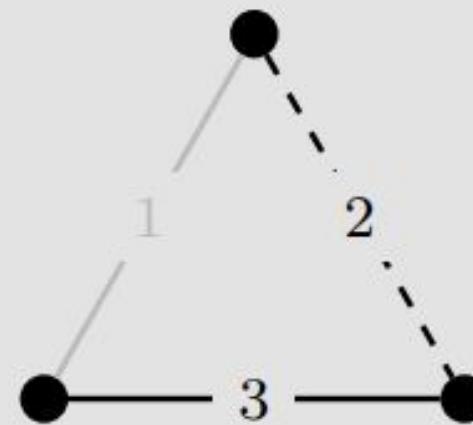
Definition 6.18 Given a graph $G = (V, E)$ an *edge-coloring* is an assignment of colors to the edges of G so that if two edges share an endpoint, then they are given different colors. The minimum number of colors needed over all possible edge-colorings is called the *chromatic index* and denoted $\chi'(G)$.

Example 6.9 Recall that the chromatic number for any complete graph is equal to the number of vertices. Find the chromatic index for K_n for all n up to 6.

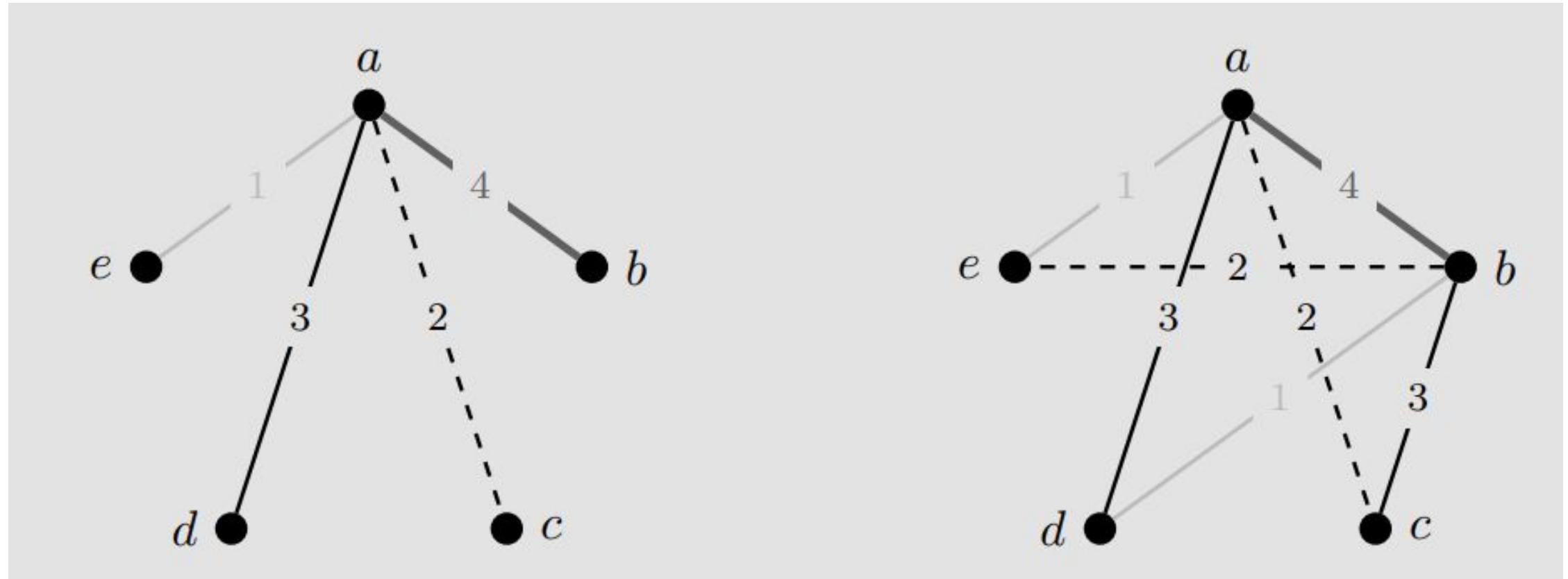
Example 6.9 Recall that the chromatic number for any complete graph is equal to the number of vertices. Find the chromatic index for K_n for all n up to 6.

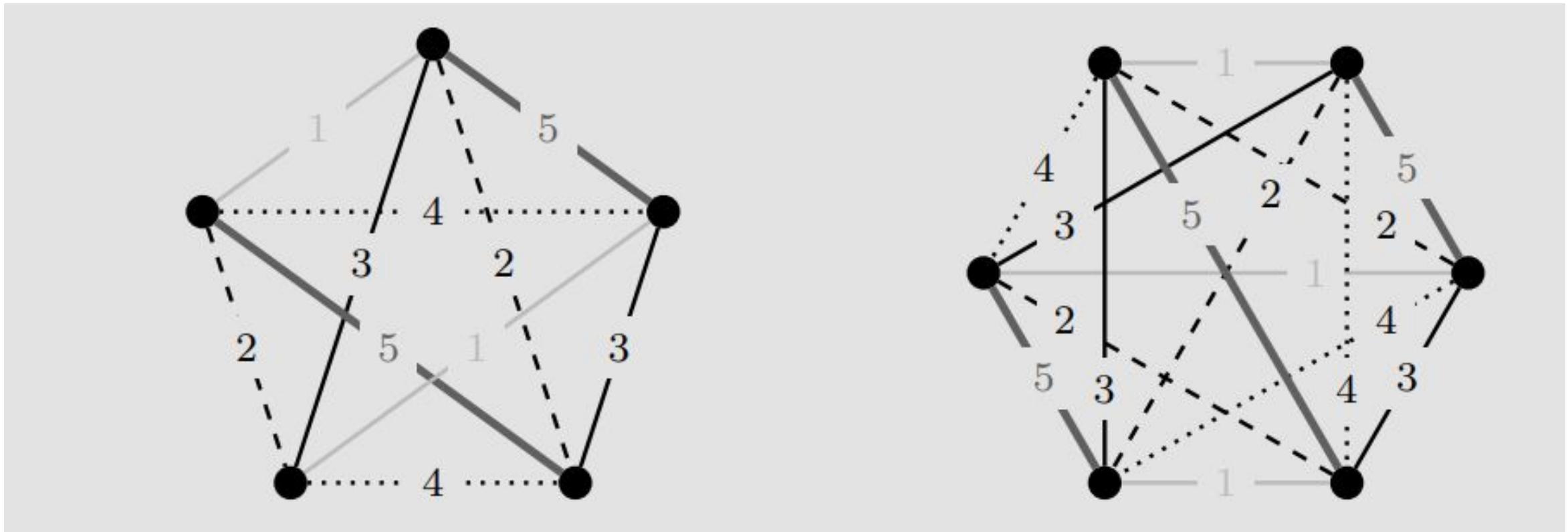
Solution: Since K_1 is a single vertex with no edges and K_2 consists of a single edge, we have $\chi'(K_1) = 0$ and $\chi'(K_2) = 1$. Due to their simplicity,

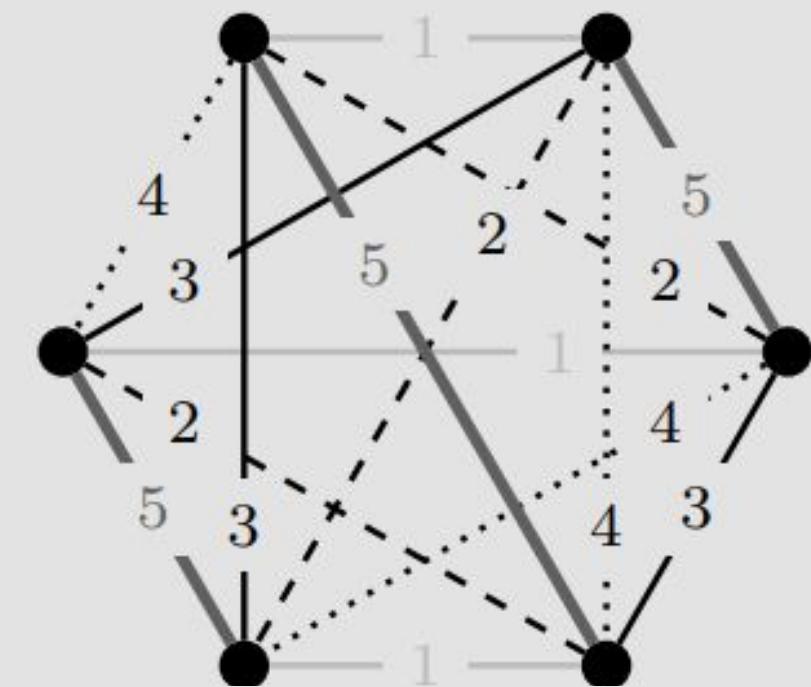
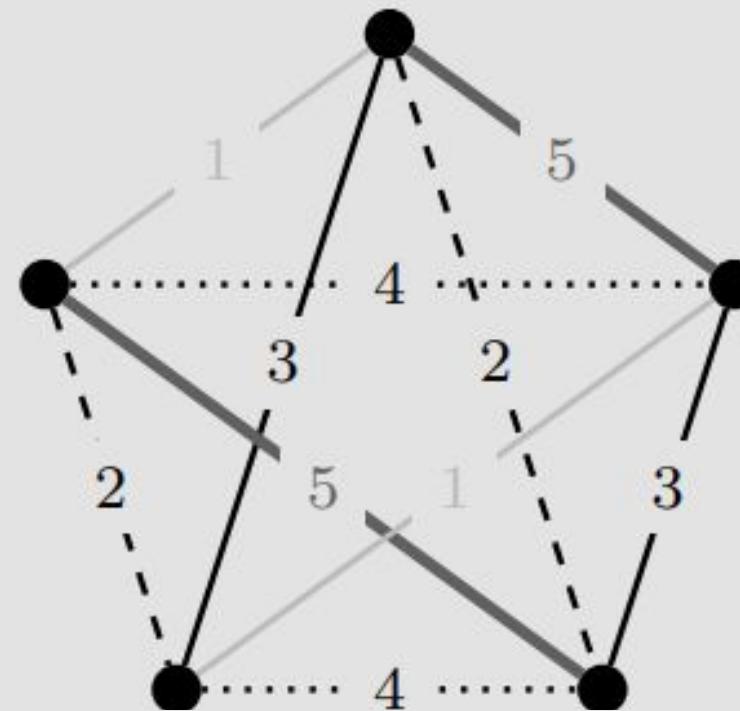
Example 6.9 Recall that the chromatic number for any complete graph is equal to the number of vertices. Find the chromatic index for K_n for all n up to 6.



For K_3 since any two edges share an endpoint, we know each edge needs its own color and so $\chi'(K_3) = 3$. For K_4 we can color opposite edges with the same color, thus requiring only 3 colors. Optimal edge-colorings for K_3 and K_4 are shown below.



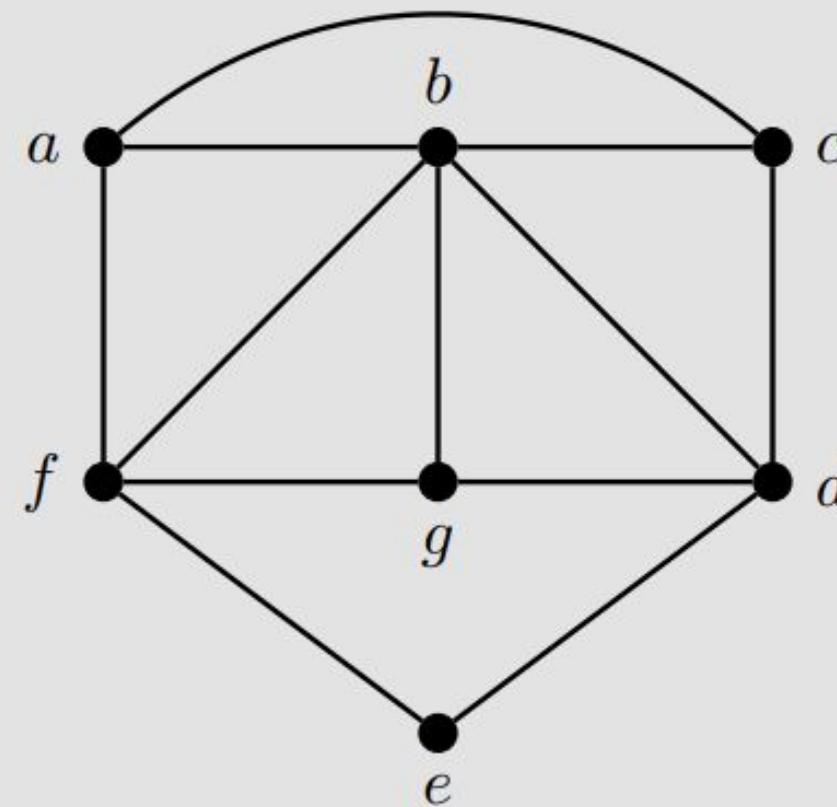




In general, $\chi'(K_n) = n - 1$ when n is even and $\chi'(K_n) = n$ when n is odd.

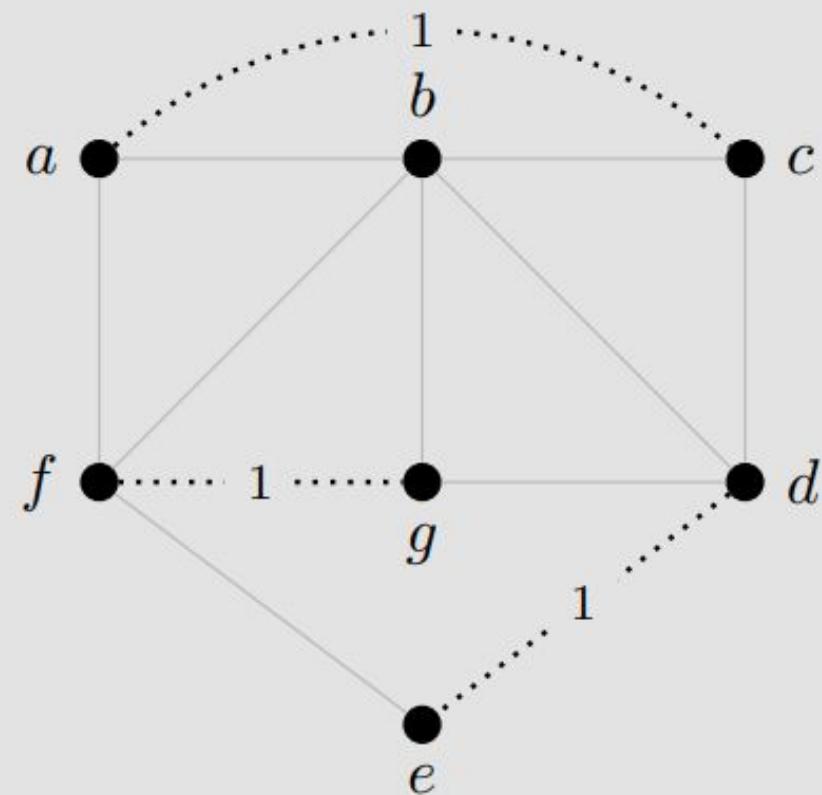
Theorem 6.19 (Vizing's Theorem) $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$ for all simple graphs G .

Example 6.10 Consider the graph G_4 below and color the edges in the order $ac, fg, de, ef, bc, cd, dg, af, bd, bg, bf, ab$ using a greedy algorithm.



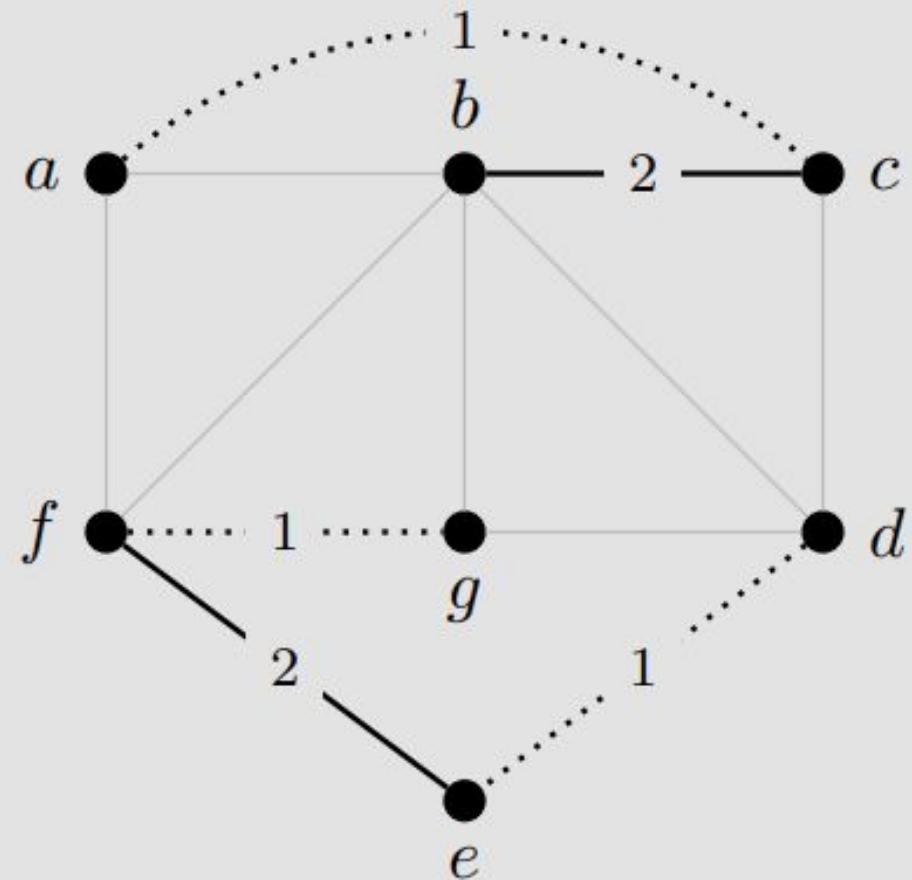
order $ac, fg, de, ef, bc, cd, dg, af, bd, bg, bf, ab$ using a greedy algorithm.

Step 1: Since the first three edges ac, fg , and de are not adjacent, we give each of them the first color.



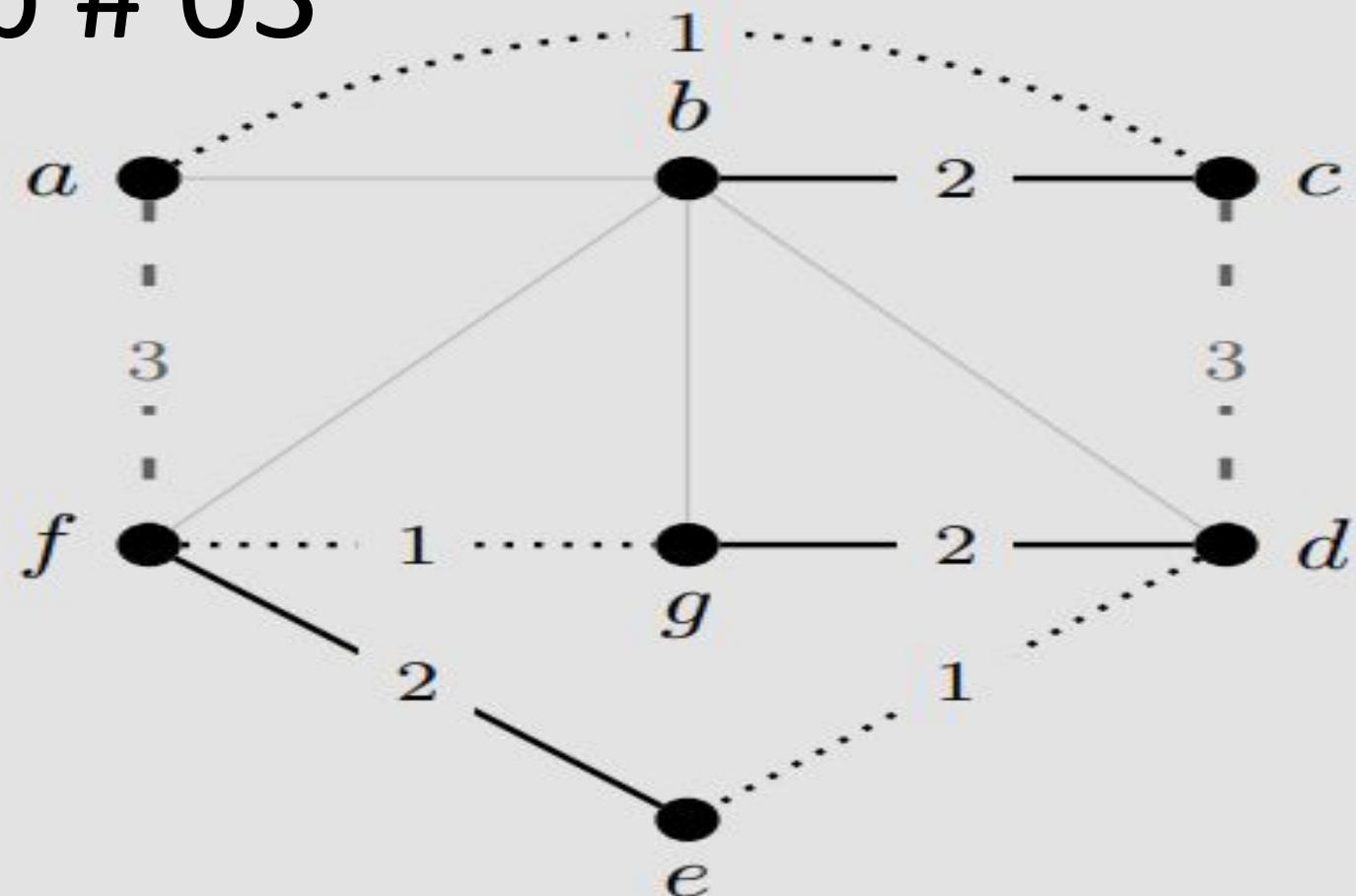
order $ac, fg, de, ef, bc, cd, dg, af, bd, bg, bf, ab$ using a greedy algorithm.

Step # 02



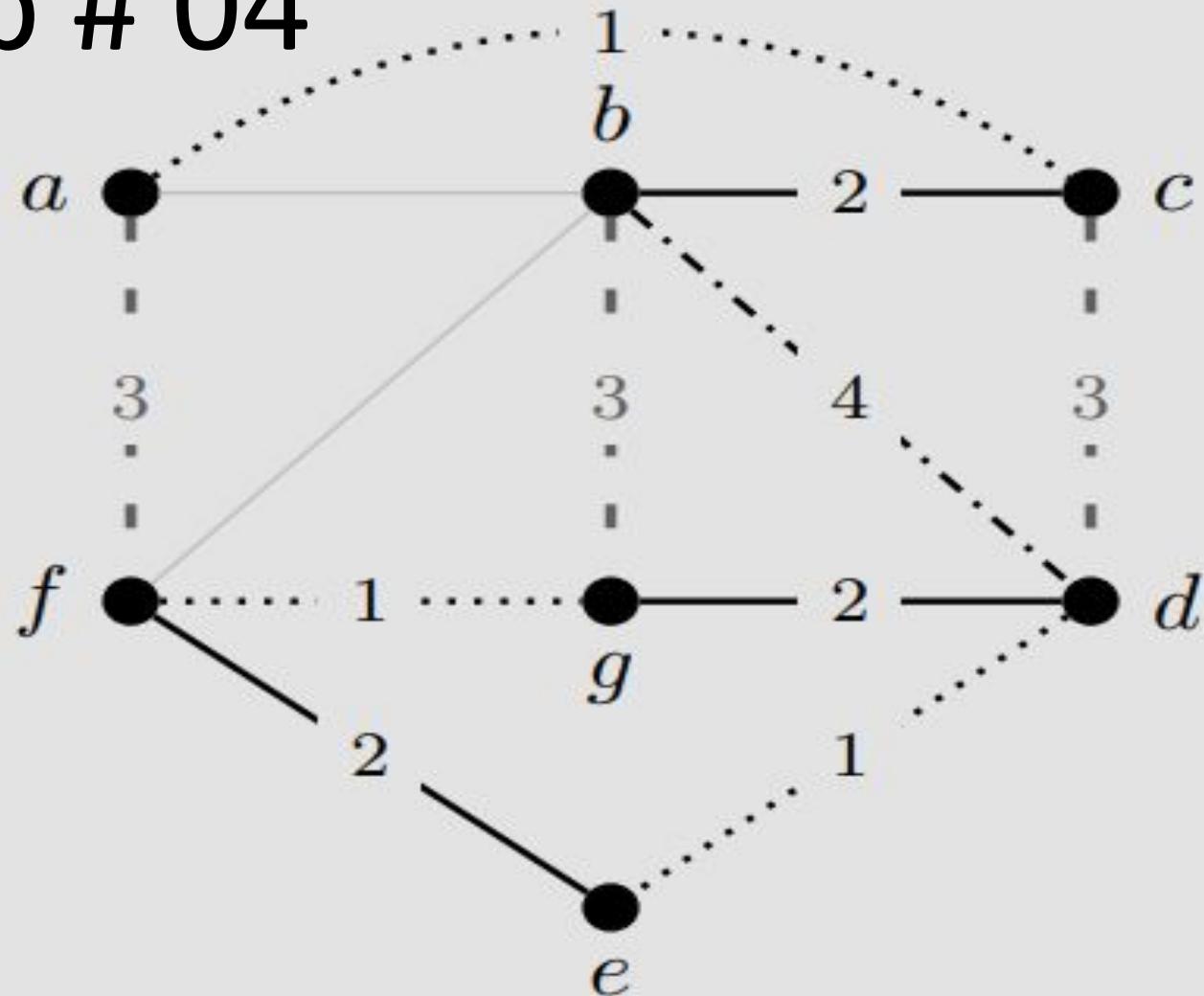
order $ac, fg, de, ef, bc, cd, dg, af, bd, bg, bf, ab$ using a greedy algorithm.

Step # 03



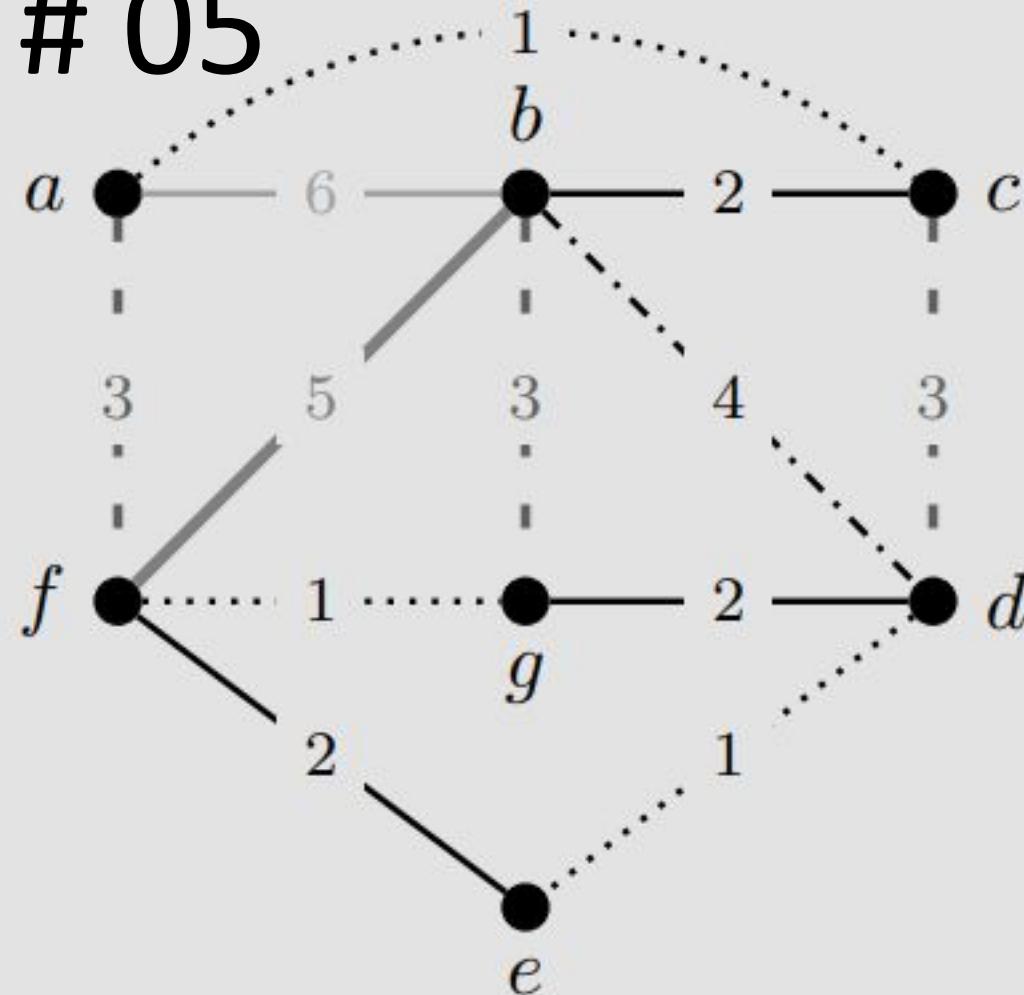
order $ac, fg, de, ef, bc, cd, dg, af, bd, bg, bf, ab$ using a greedy algorithm.

Step # 04

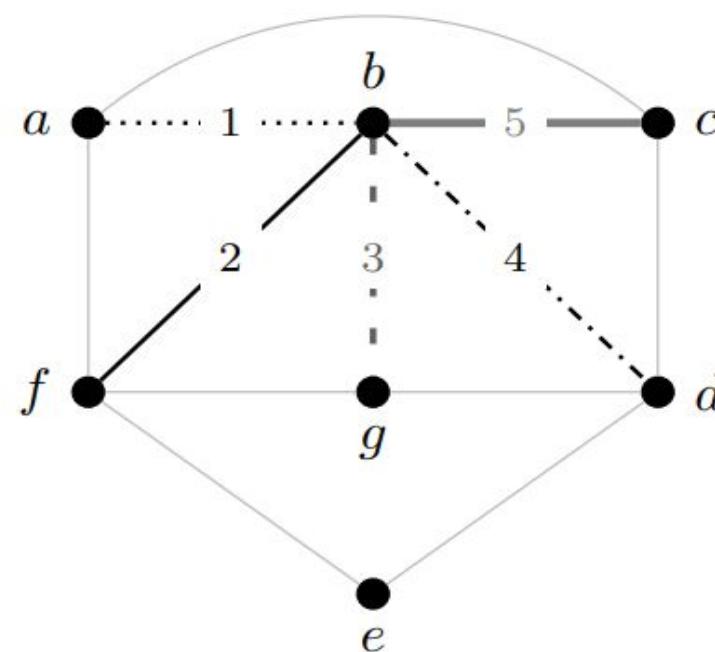


order $ac, fg, de, ef, bc, cd, dg, af, bd, bg, bf, ab$ using a greedy algorithm.

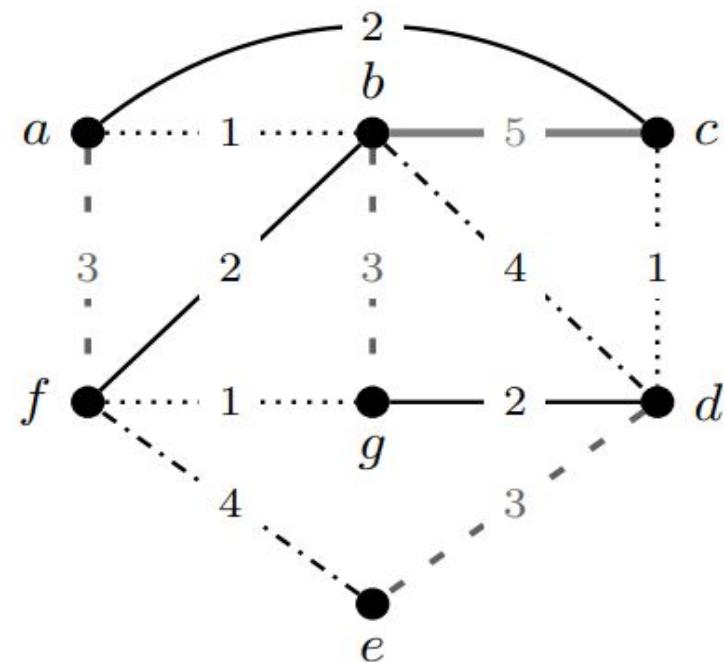
Step # 05



In the example above $\Delta(G_4) = 5$, and the edge-coloring above uses 6 colors; however $\chi'(G_4) = 5$. In general, starting with the vertex of highest degree and coloring its edges has a better chance of success in avoiding unnecessary colors, as shown below on the left. Once we have the minimum number of colors established, we attempt to fill in the remaining edges without introducing an extra color; one possible solution is shown below on the right.



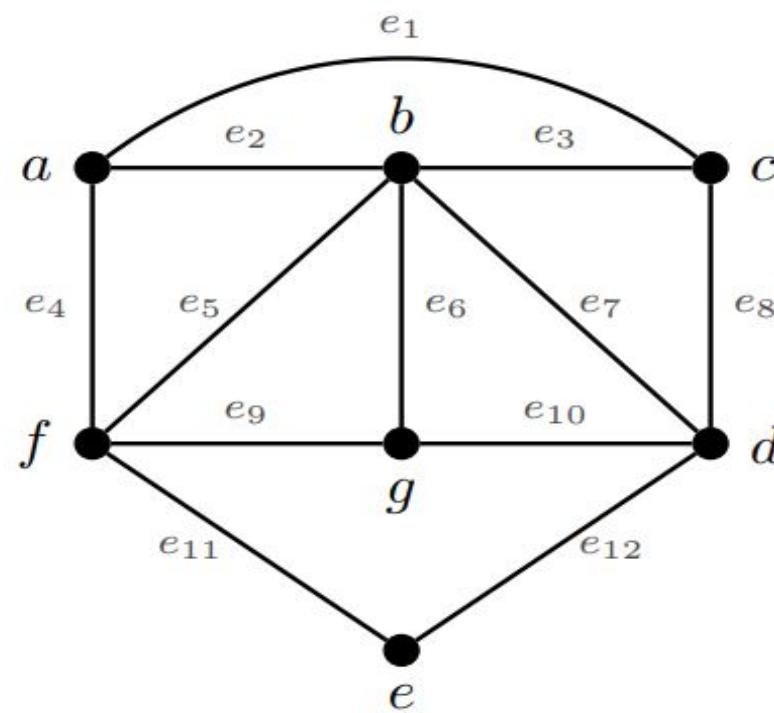
initial coloring of G_4



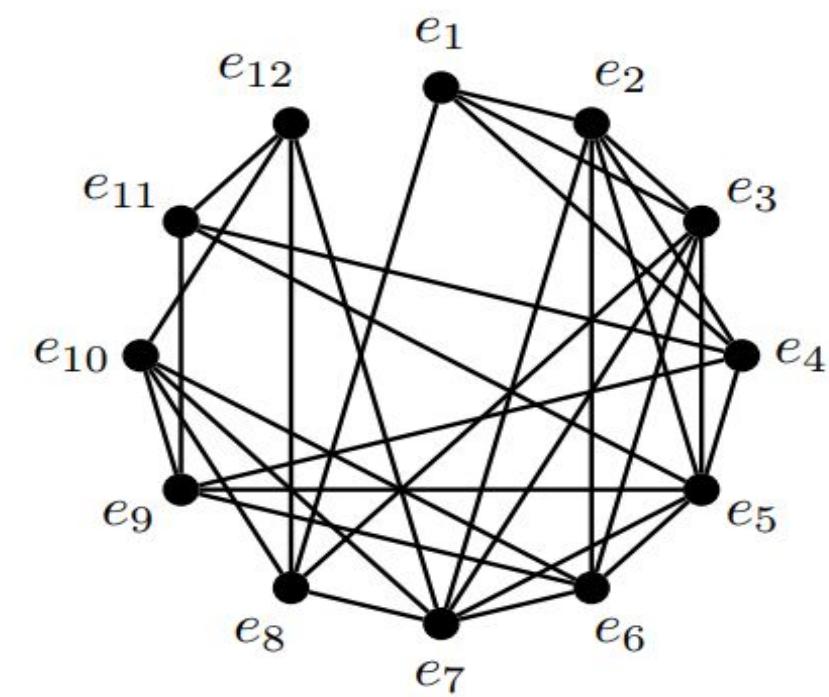
optimal edge-coloring of G_4

Definition 6.20 Given a graph $G = (V, E)$, the *line graph* $L(G) = (V', E')$ is the graph formed from G where each vertex x' in $L(G)$ represents the edge x' from G and $x'y'$ is an edge of $L(G)$ if the edges x' and y' share an endpoint in G .

Below is the graph G_4 from Example 6.10 and its line graph. Notice that the vertex e_1 in $L(G_4)$ is adjacent to e_2 and e_4 through the vertex a in G_4 and e_1 is adjacent to e_3 and e_8 through the vertex c in G_4 .



G_4

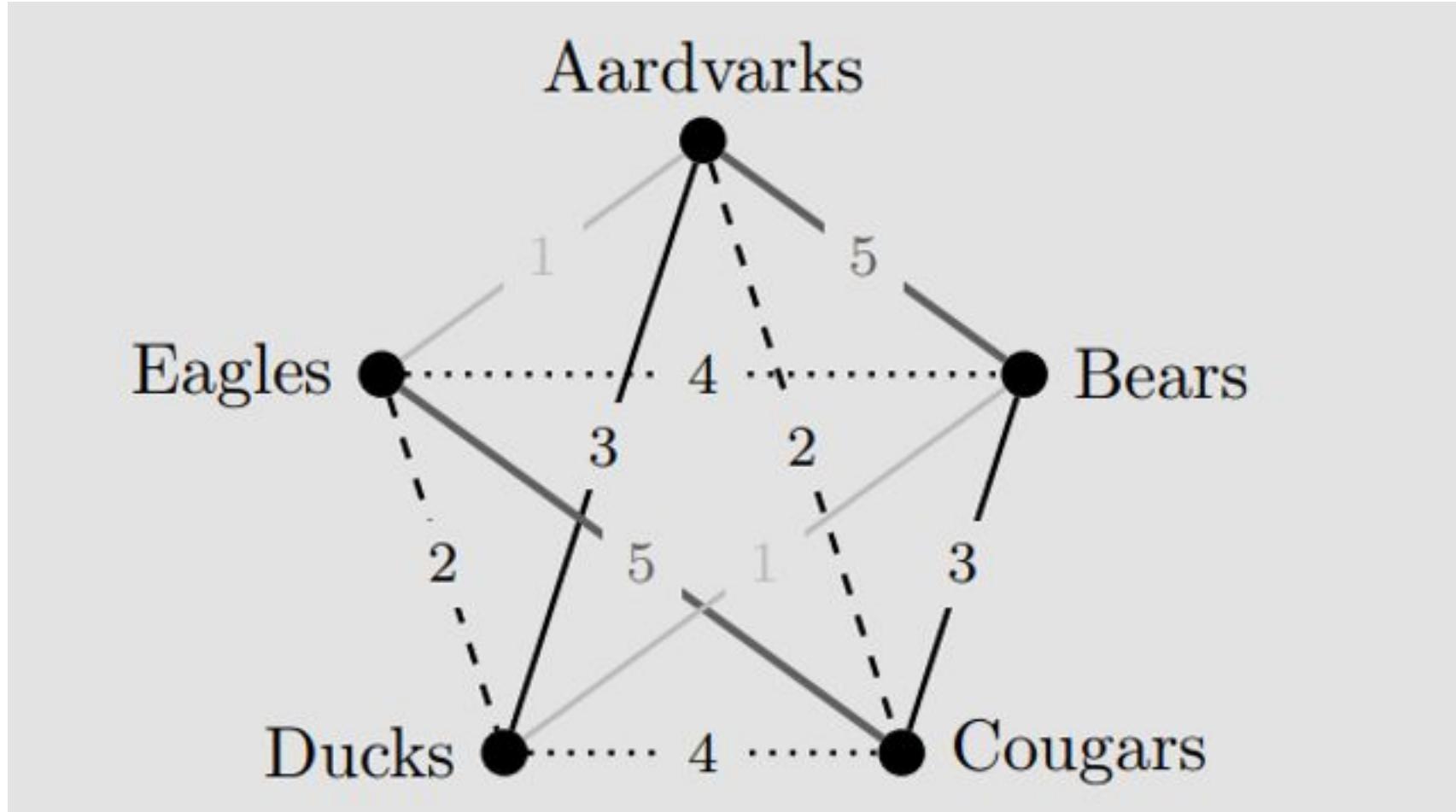


$L(G_4)$

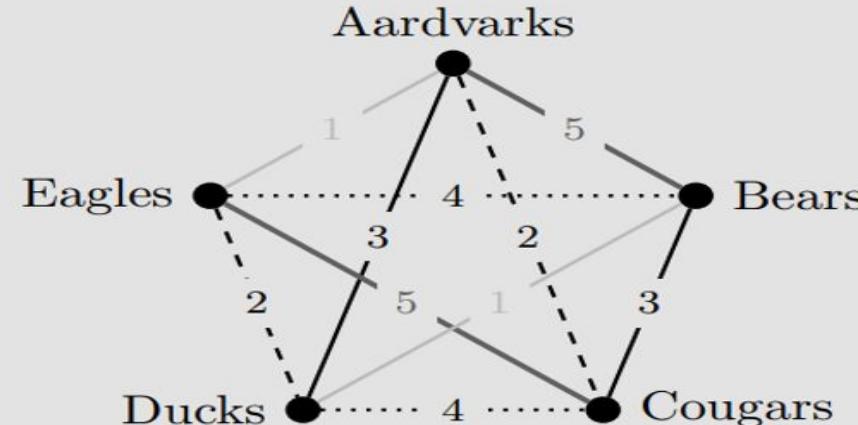
From this definition, it should be clear how edge-coloring and vertex-coloring are related. In particular, if edge e_1 is given the color blue in G , this would correspond to coloring vertex e_1 in $L(G)$ with blue. This correspondence provides the following result.

Theorem 6.21 Given a graph G with line graph $L(G)$, we have $\chi'(G) = \chi(L(G))$.

Example 6.11 The five teams from Section 1.1 (Aardvarks, Bears, Cougars, Ducks, and Eagles) need to determine the game schedule for the next year. If each team plays each of the other teams exactly once, determine a schedule where no team plays more than one game on a given weekend.



In general, $\chi'(K_n) = n - 1$ when n is even and $\chi'(K_n) = n$ when n is odd.



Week	Games	
1	Aardvarks vs. Bears	Cougars vs. Eagles
2	Aardvarks vs. Cougars	Ducks vs. Eagles
3	Aardvarks vs. Ducks	Bears vs. Cougars
4	Aardvarks vs. Eagles	Bears vs. Ducks
5	Bears vs. Eagles	Cougars vs. Ducks

Coloring Variations

This section will highlight additional variations of graph coloring, specifically vertex colorings. Within each of these we will see applications of graph coloring that can inform why this new version of coloring is worthy of study.

On-line Coloring

Definition 6.24 Consider a graph G with the vertices ordered as $x_1 \prec x_2 \prec \dots \prec x_n$. An *on-line algorithm* colors the vertices one at a time where the color for x_i depends on the induced subgraph $G[x_1, \dots, x_i]$

which consists of the vertices up to and including x_i . The maximum number of colors a specific algorithm \mathcal{A} uses on any possible ordering of the vertices is denoted $\chi_{\mathcal{A}}(G)$.

A greedy algorithm called ***First-Fit*** that uses the first available color for a new vertex

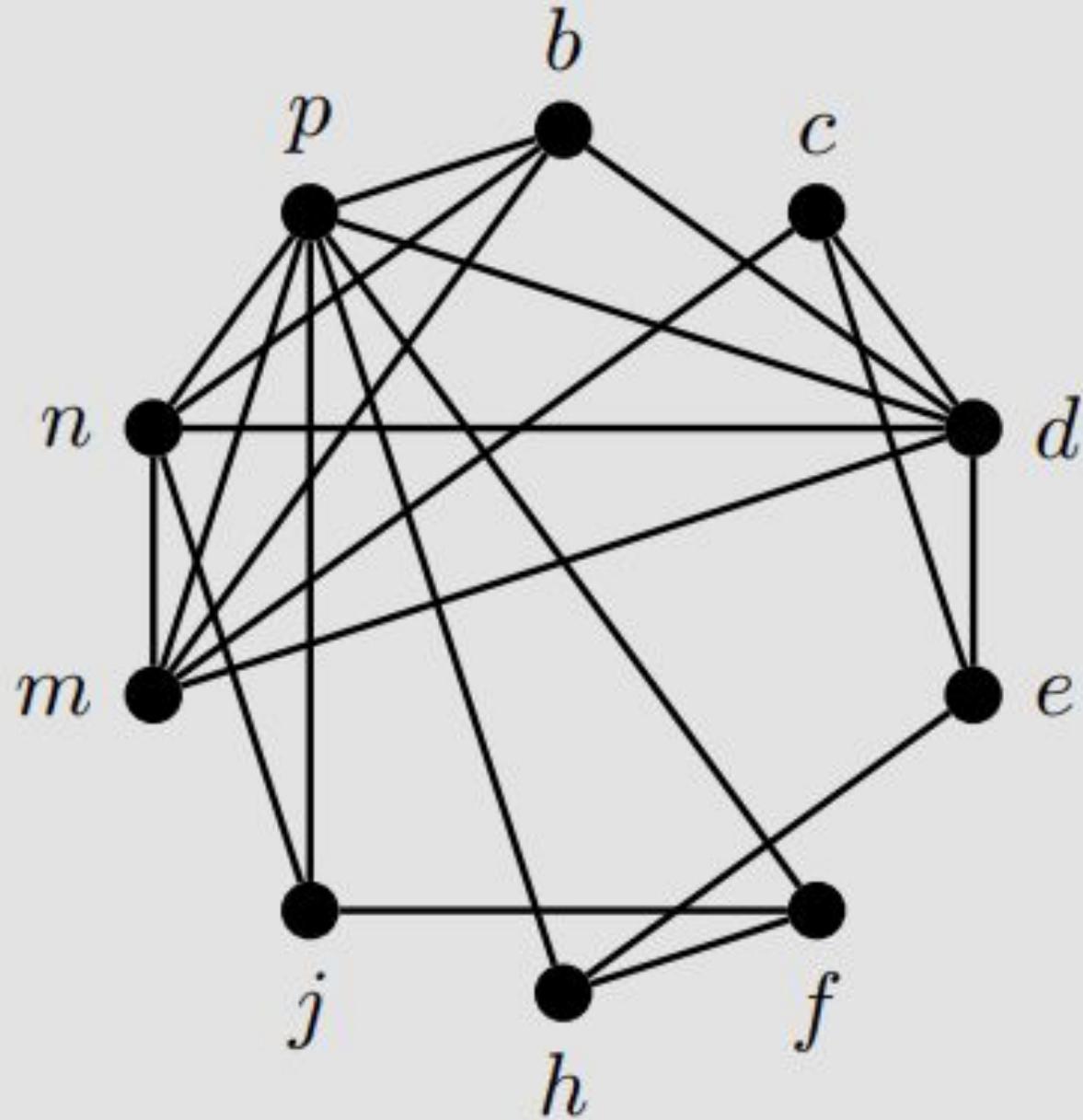
First-Fit Coloring Algorithm

Input: Graph G with vertices ordered as $x_1 \prec x_2 \prec \dots \prec x_n$.

Steps:

1. Assign x_1 color 1.
2. Assign x_2 color 1 if x_1 and x_2 are not adjacent; otherwise, assign x_2 color 2.
3. For all future vertices, assign x_i the least number color available to x_i in $G[x_1, \dots, x_i]$; that is, give x_i the first color not used by any neighbor of x_i that has already been colored.

Output: Coloring of G .



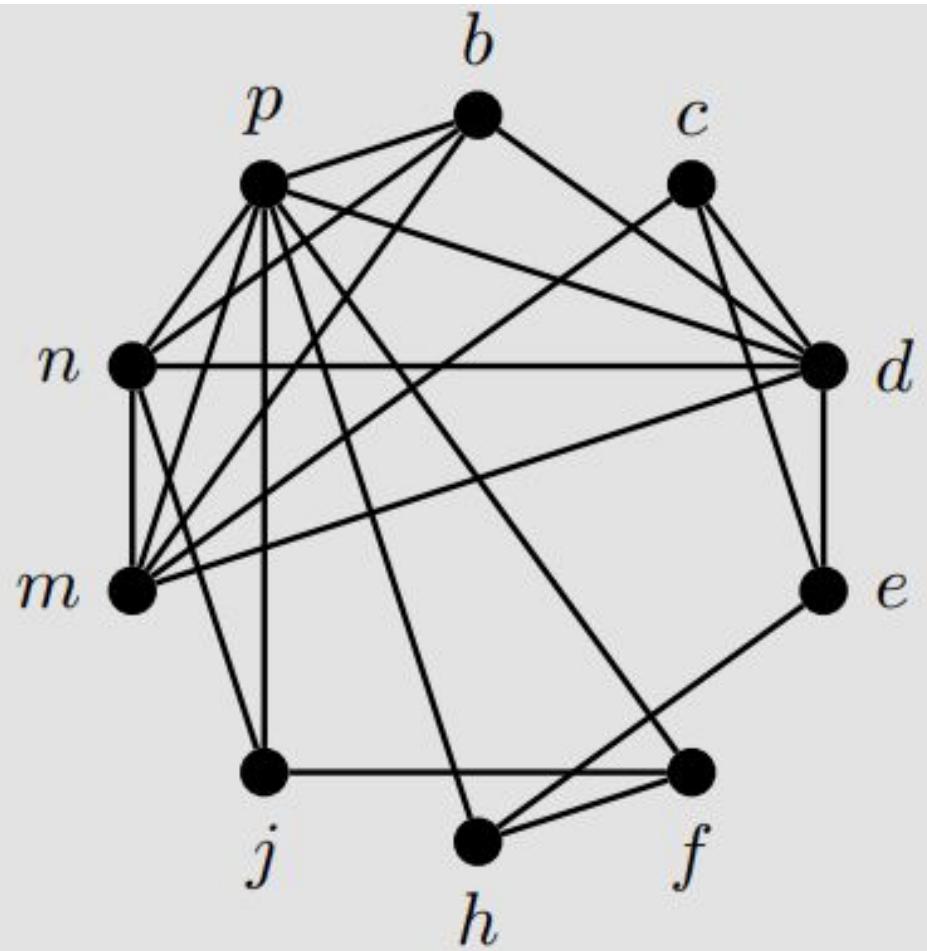
First-Fit Coloring Algorithm

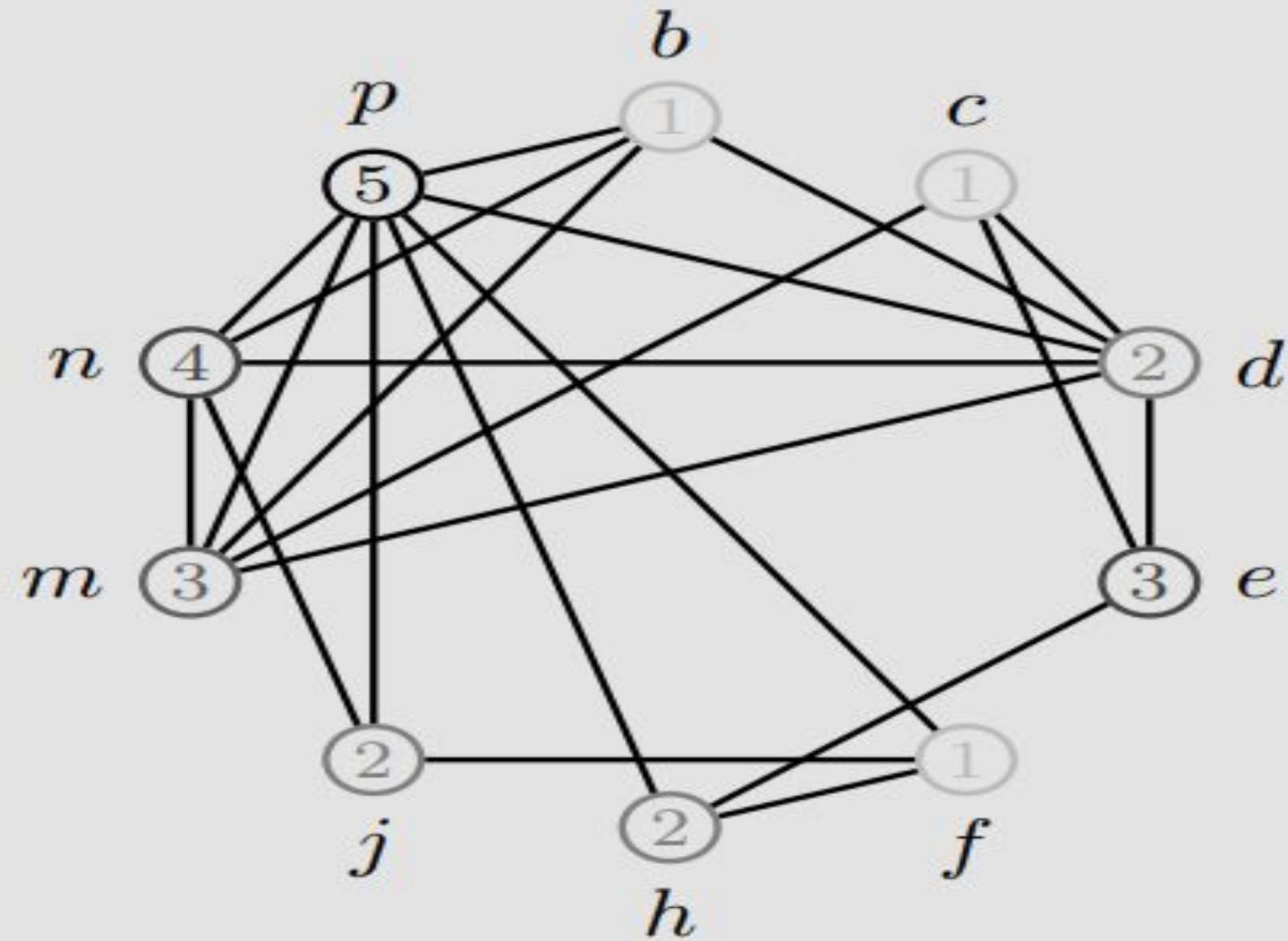
Input: Graph G with vertices ordered as $x_1 \prec x_2 \prec \dots \prec x_n$.

Steps:

1. Assign x_1 color 1.
2. Assign x_2 color 1 if x_1 and x_2 are not adjacent; otherwise, assign x_2 color 2.
3. For all future vertices, assign x_i the least number color available to x_i in $G[x_1, \dots, x_i]$; that is, give x_i the first color not used by any neighbor of x_i that has already been colored.

Output: Coloring of G .



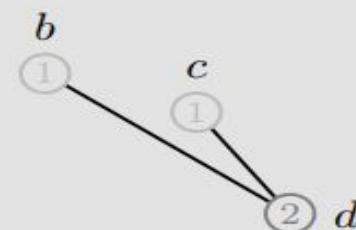


Hint:

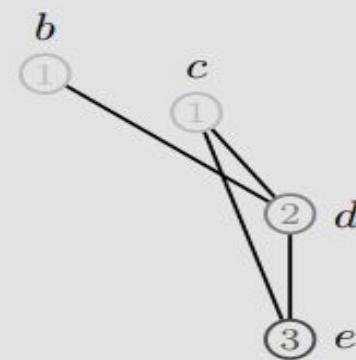
Step 2: Color c with 1 since b and c are not adjacent.



Step 3: Color d with 2 since d is adjacent to a vertex of color 1.



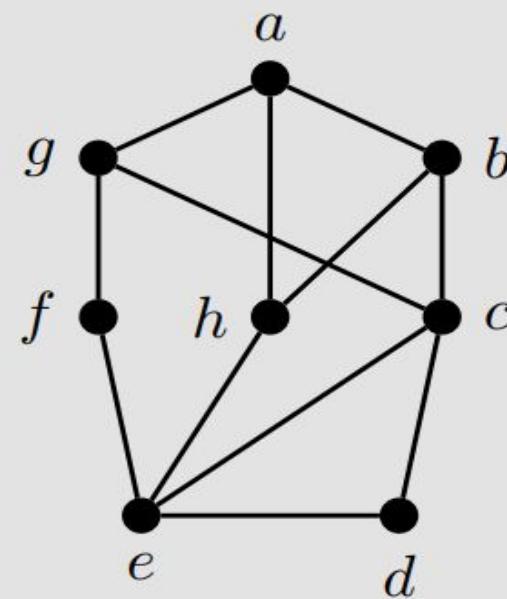
Step 4: Color e with 3 since e is adjacent to a vertex of color 1 (c) and a vertex of color 2 (d).



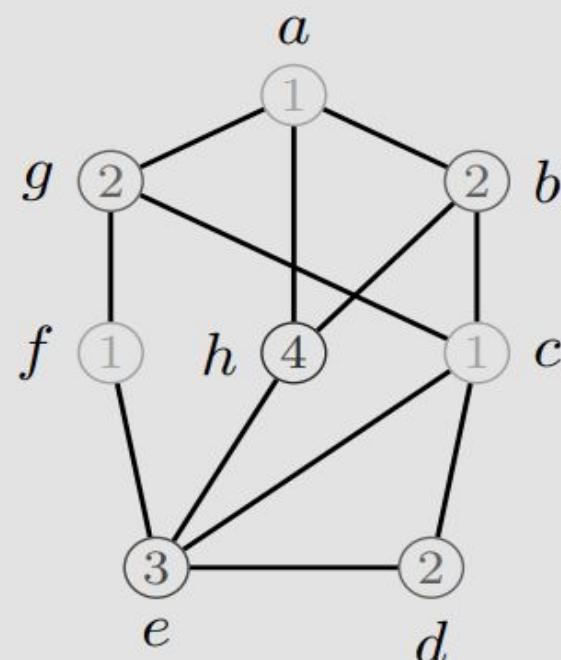
Do Example 6.15 & 6.16

Example 6.16 Color the graph below using First-Fit using the two different vertex orders listed and determine if either finds the optimal coloring for the graph.

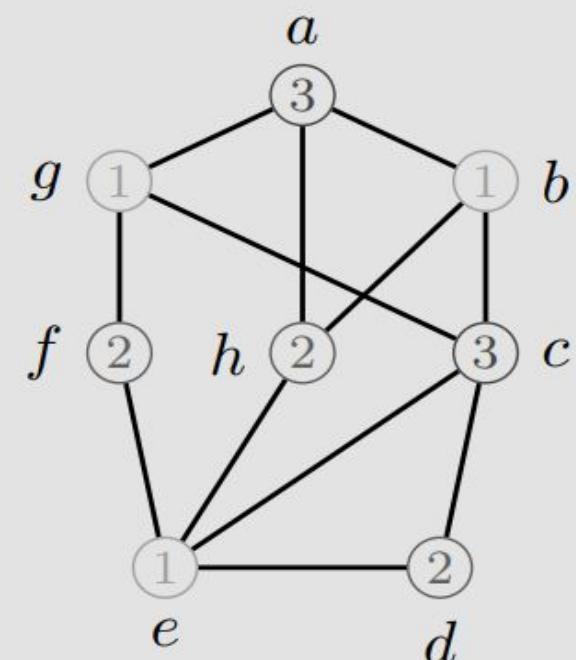
- (a) $a \prec b \prec c \prec d \prec e \prec f \prec g \prec h$
- (b) $e \prec h \prec b \prec d \prec c \prec g \prec f \prec a$



Solution: Below are the two First-Fit colorings of the graph based on the order given. Only the vertex order from (b) gives the optimal coloring since $\omega(G) = 3$.



vertex order (a)



vertex order (b)

H.

Example 6.17 Prove there exists an interval graph G using at least $3\omega(G) - 2$ colors.

Weighted Coloring

Ten families need to buy train tickets for an upcoming trip. The families vary in size but each of them needs to sit together on the train. Determine the minimum number of seats needed to accommodate the ten family trips.

Weighted Coloring

Definition 6.25 Given a weighted graph $G = (V, E, w)$, where w assigns each vertex a positive integer, a proper ***weighted coloring*** of G assigns each vertex a set of colors so that

- (i) the set consists of consecutive colors (or numbers);
- (ii) the number of colors assigned to a vertex equals its weight; and
- (iii) if two vertices are adjacent, then their set of colors must be disjoint.

Note that in some publications, weighted colorings are referred to as interval colorings (since an interval of colors is being assigned to each vertex). To avoid the confusion between interval colorings and interval graphs, we use the term weighted coloring.

Example 6.18 Find an optimal weighted coloring for the graph below where the vertices have weights as shown below.

$$w(a) = 2$$

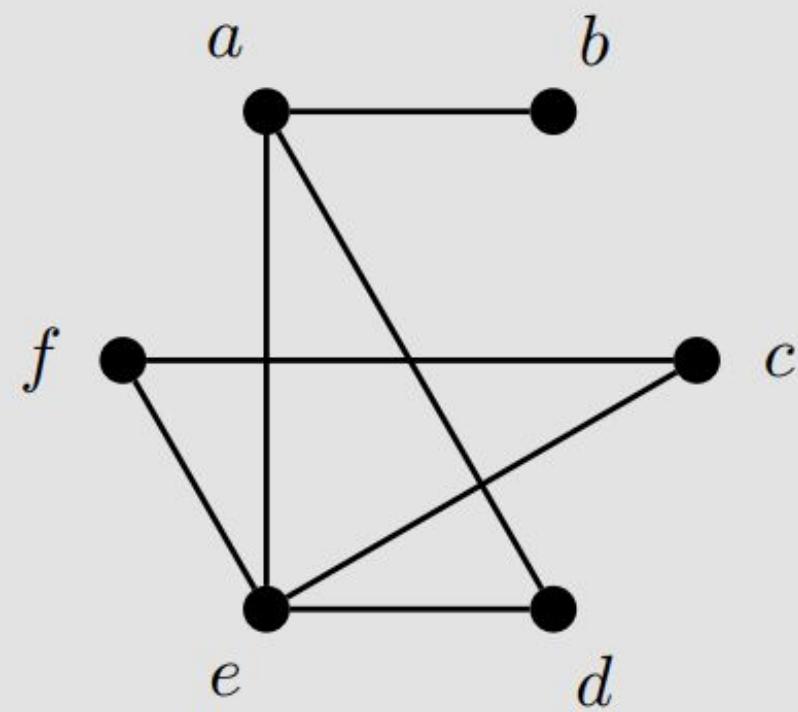
$$w(b) = 1$$

$$w(c) = 4$$

$$w(d) = 2$$

$$w(e) = 2$$

$$w(f) = 4$$



The biggest change will be to focus on locations that have large weighted cliques, which is found by adding the weights of the vertices within a complete subgraph. Thus if we have two different cliques on three vertices, one with total weight 8 and the other with total weight 10, we should initially focus on the one with the higher total weight.

$$w(a) = 2$$

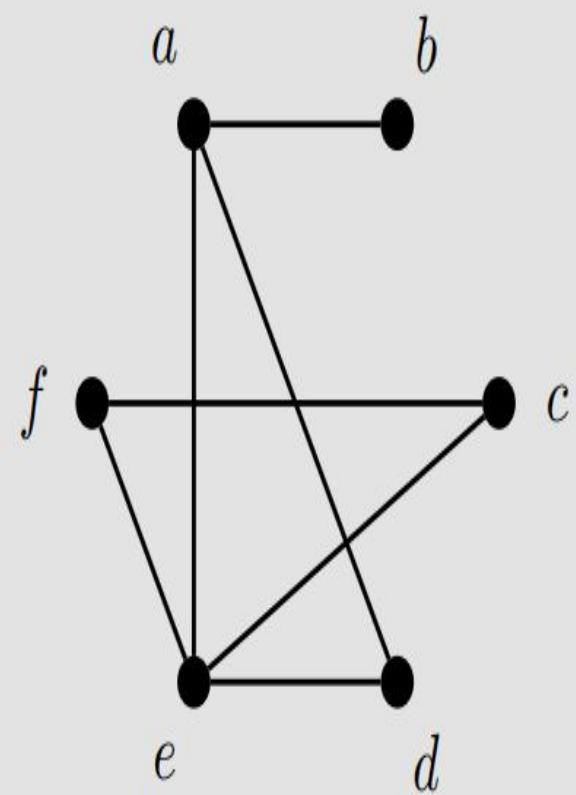
$$w(b) = 1$$

$$w(c) = 4$$

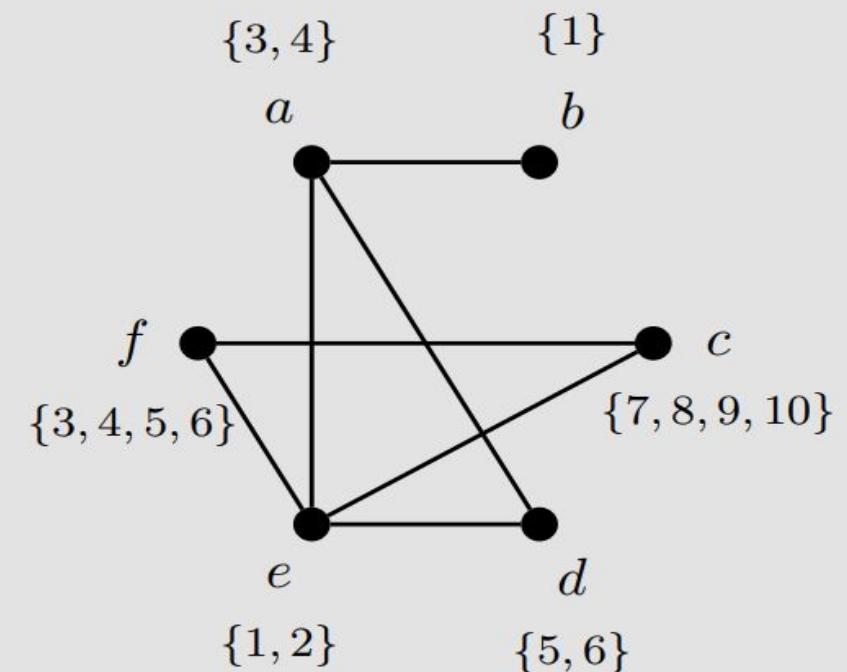
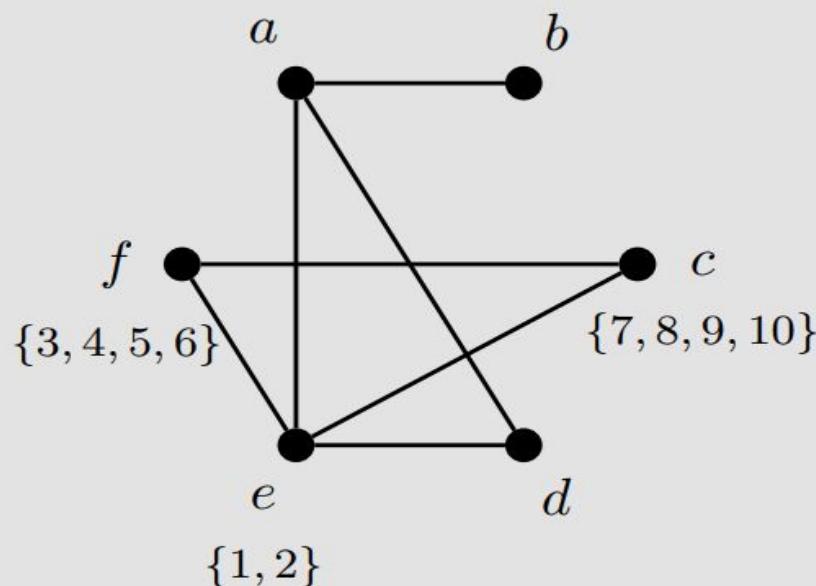
$$w(d) = 2$$

$$w(e) = 2$$

$$w(f) = 4$$



6
Solution: Note that a, d , and e form a K_3 with total weight 8, and c, e , and f form a K_3 with total weight 10. We begin by assigning weights to the vertices from the second K_3 , and since e appears twice we assign it the first set of colors $\{1, 2\}$. From there we are forced to use another 4 colors on f ($\{3, 4, 5, 6\}$) and another 4 colors on c ($\{7, 8, 9, 10\}$), as shown on the left below. We can color the remaining three vertices using colors 1 through 10 as shown on the right.



Example 6.19 Suppose the ten families needing train tickets have the same underlying graph as that from Example 6.15 and the size of each family is noted below. Determine the minimum number of seats needed to accommodate everyone's travels.

$$w(a) = 2$$

$$w(b) = 5$$

$$w(c) = 2$$

$$w(d) = 1$$

$$w(e) = 4$$

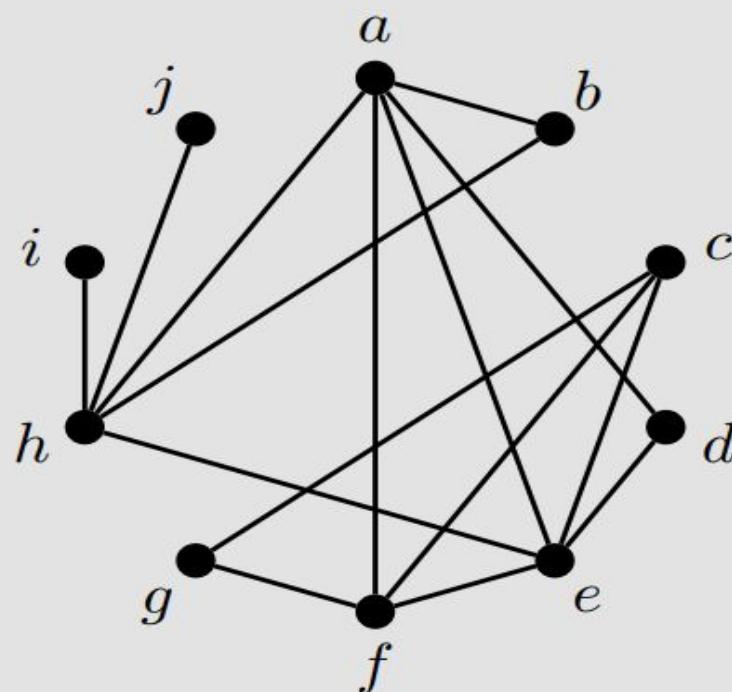
$$w(f) = 3$$

$$w(g) = 1$$

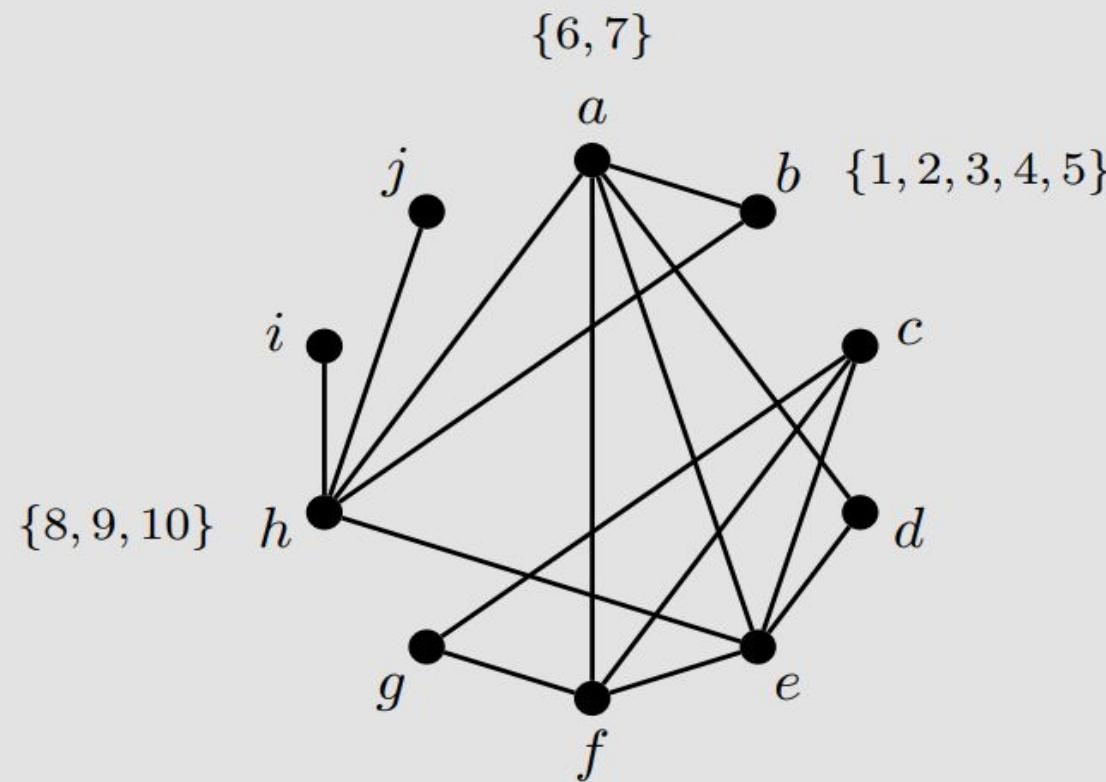
$$w(h) = 3$$

$$w(i) = 4$$

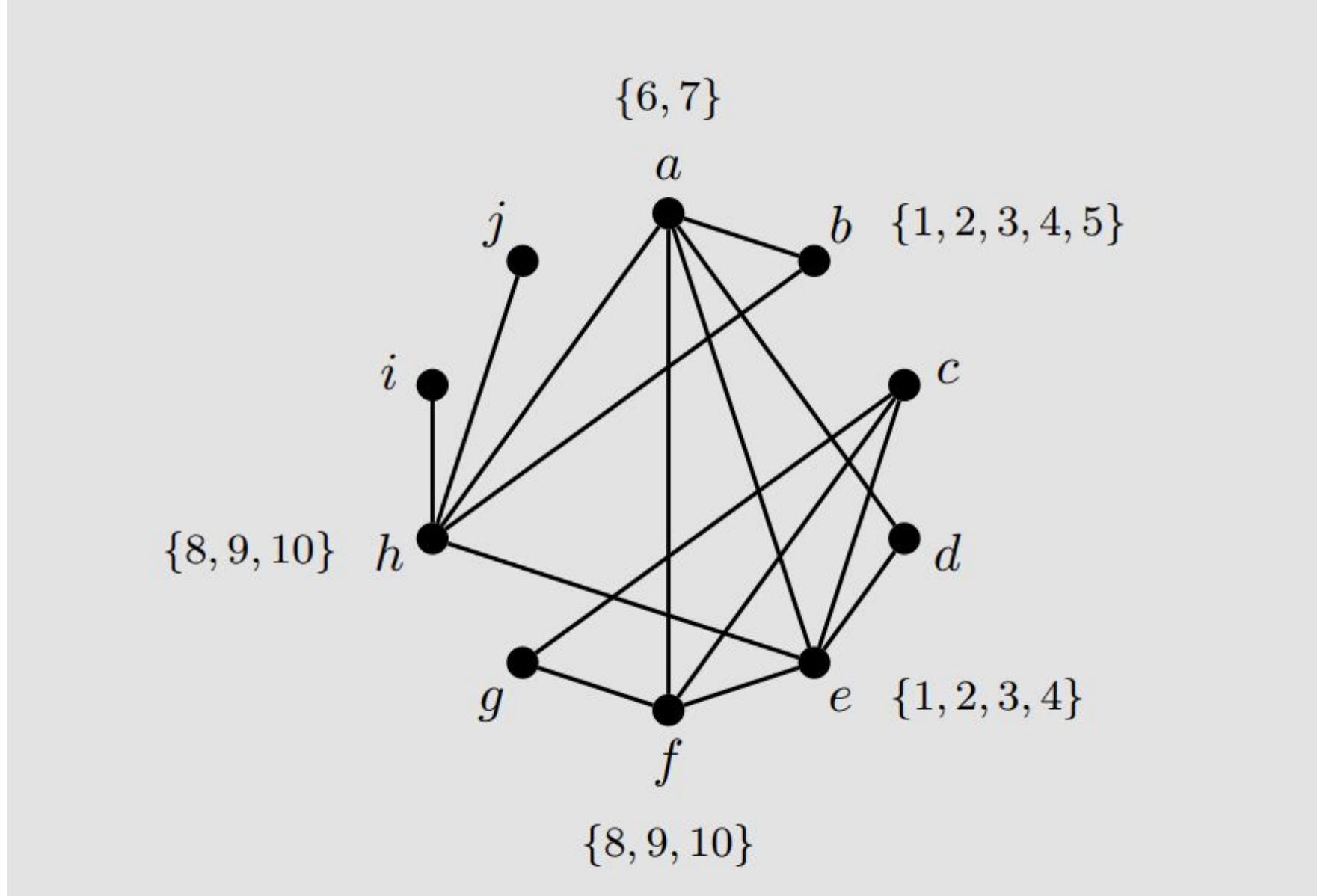
$$w(j) = 5$$

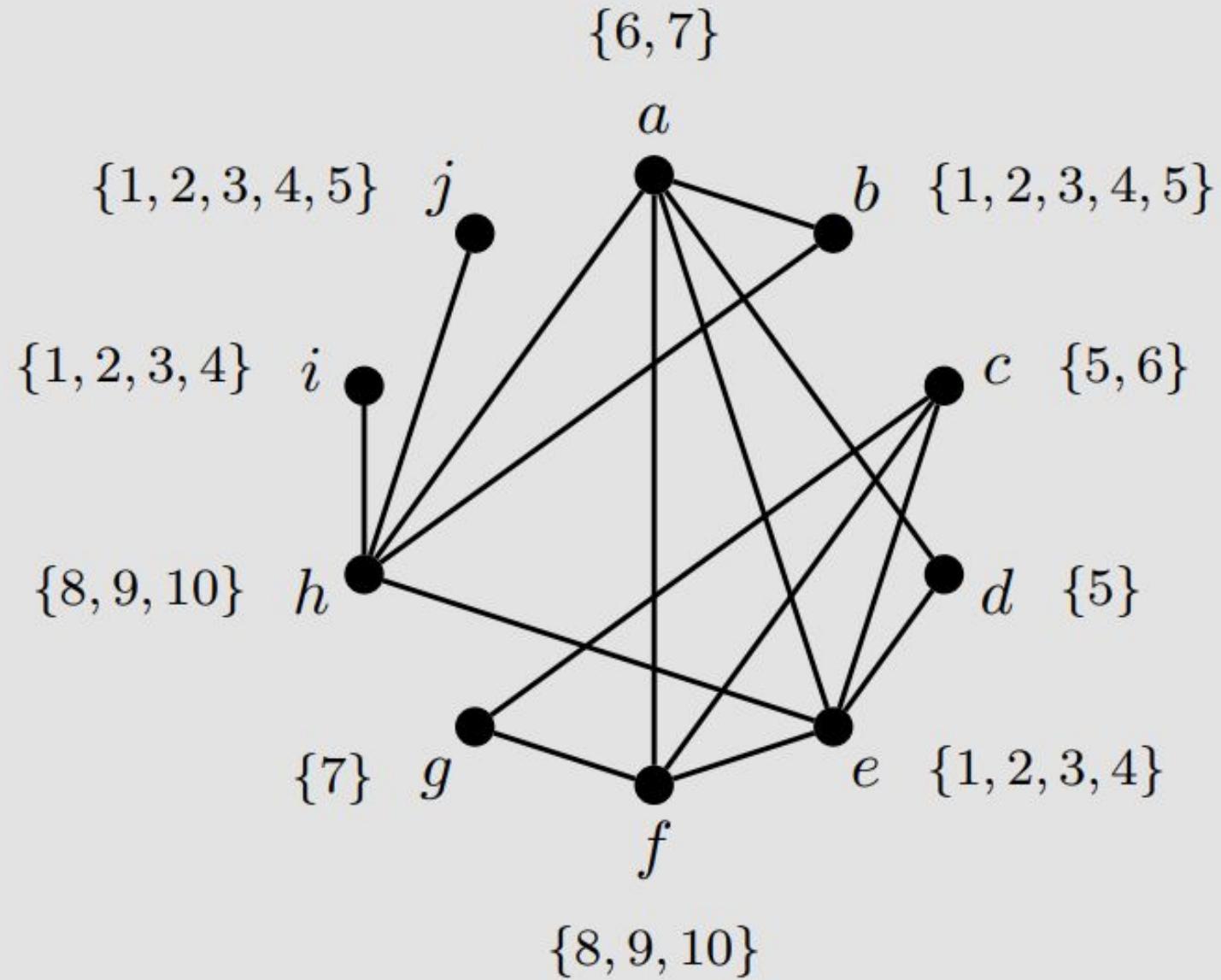


Solution: As with the example above, we begin by looking for the largest total weight for a clique. Since the clique size is three, we want to find K_3 subgraphs with high total weight. The largest of these is with a, b , and h with total 10. Since b has the largest weight among these, we give it colors 1 through 5, assign a colors 6 and 7, and colors 8, 9, and 10 to h .



The next two cliques with a high total weight (of 9) are formed by a, e, f and c, e, f . Since a has already been assigned colors, we begin with the first clique. We can fill in the colors for e and f without introducing new colors, as shown below. Note that since e is adjacent to both a and h it could only use four consecutive colors chosen from those used on b .





The combination of interval graphs, on-line algorithms, and weighted colorings have been extensively studied due to a very specific application, called **Dynamic Storage Allocation, or DSA**. The storage allocation refers to assigning **variables to locations within a computer's memory**, where each variable has a size associated to it. This can be thought of as the weight of a vertex. The location a variable is assigned is the set of colors a vertex is given. The dynamic part of DSA refers to variables being in use for specific intervals of time and only the previously used (or in-use) variables locations are known. Thus each vertex is represented by an interval and the coloring must use an on-line algorithm. In total, DSA can be modeled as an on-line coloring of a weighted interval graph. Modifications to the known performance

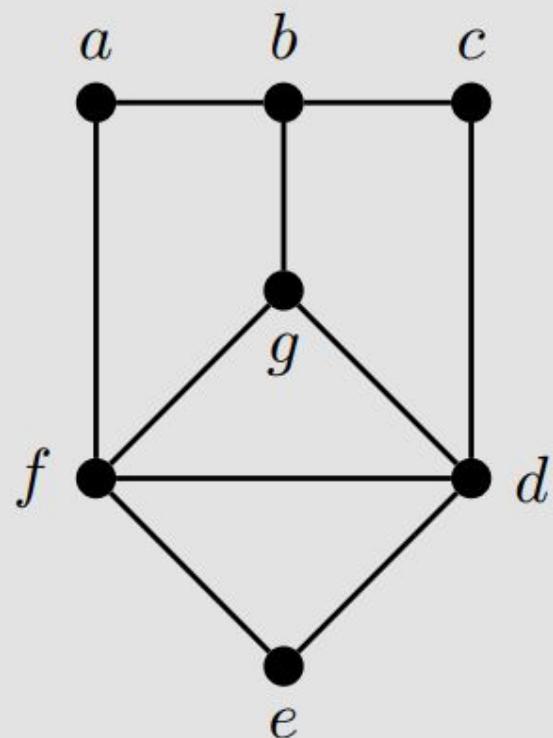
Lec # 41, 42 & 43

List Coloring

List coloring is a more modern approach to graph coloring and was introduced independently by Vizing in 1976.

In list coloring we are concerned with assigning colors to vertices (or edges) with the added restriction that the colors must come from some predefined list.

Example 6.20 The table on the right below shows two different lists for the vertices in the graph on the left. Find a proper coloring for each version of the lists or explain why none exists.



Vertex	List 1	List 2
a	{1, 2}	{1}
b	{1, 3, 5}	{1, 4, 5}
c	{3, 4}	{3, 4}
d	{3, 4}	{3, 4}
e	{2, 3}	{2, 3}
f	{1, 2, 3}	{1, 2, 3}
g	{4, 5}	{4, 5}

Solution: A possible coloring from the first set of lists can be given as

$$a - 1, b - 4, c - 3, d - 4, e - 3, f - 2, g - 5.$$

However, there is no proper coloring possible from the second set of lists. First note a must be given color 1 and so f can only be colored 2 or 3. But since those are the same colors available for e , we know one will be given color 2 and the other color 3. In either case, since d is adjacent to both f and e we know it cannot be given color 3, and so must be given color 4, forcing g to be color 5 and c to be color 3. But now b is adjacent to vertices of color 1, 3, and 5, and so cannot be given a color from its list.

Definition 6.26 Let G be a graph where each vertex x is given a list $L(x)$ of colors. A proper *list coloring* assigns to x a color from its list $L(x)$ so that no two adjacent vertices are given the same color.

If for every collection of lists, each of size k , a proper list coloring exists then G is ***k-choosable***. The minimum value for k for which G is k -choosable is called the ***choosability*** of G and denoted $ch(G)$.

Proposition 6.27 For any simple graph G , $ch(G) \geq \chi(G)$.

Proof: Let G satisfy $\chi(G) = k$ and give each vertex of G the list $\{1, 2, \dots, k\}$. Then there is a proper coloring for G from these lists, namely the one exhibited by the fact that $\chi(G) = k$. However, if we remove the same one element from each of these lists, then G cannot be colored since otherwise $\chi(G) < k$.

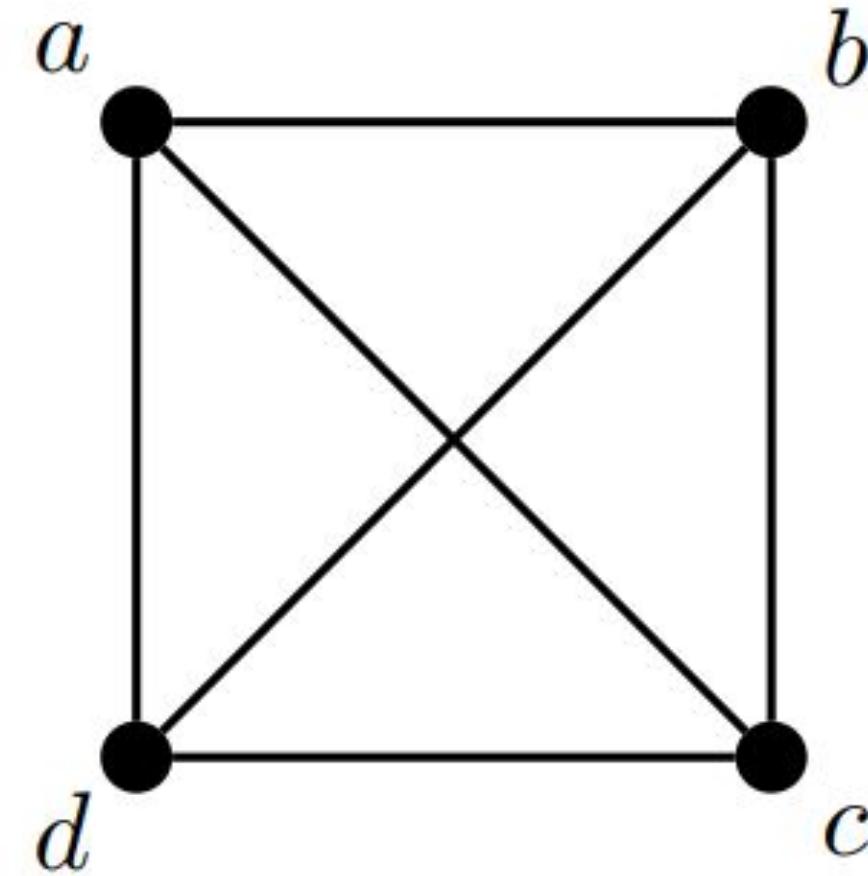
Proposition 6.28 For any simple graph G , $ch(G) \leq \Delta(G) + 1$.

EX #: 6.5

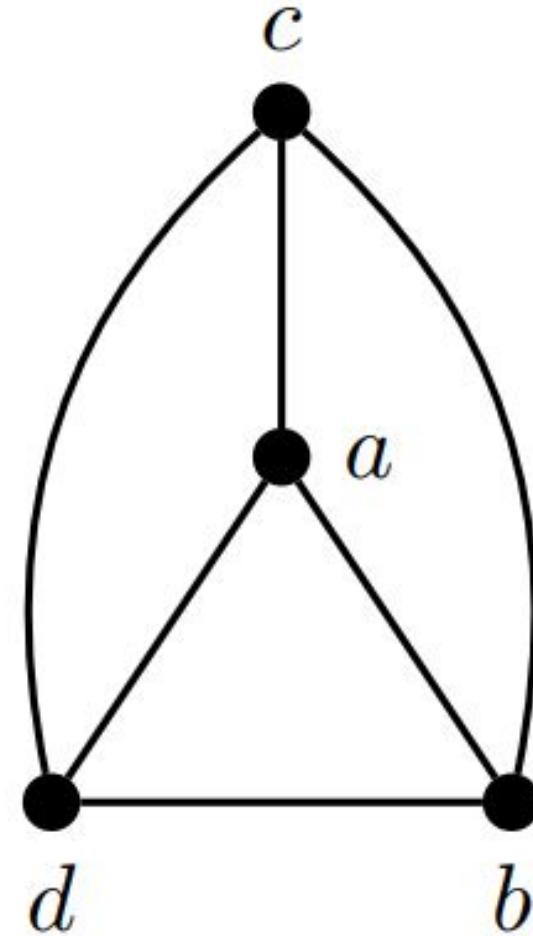
Problems: 6.1-6.9, 6.12, 6.13, 6.14, 6.19.

Planarity

Definition 7.1 A graph G is *planar* if and only if the vertices can be arranged on the page so that edges do not cross (or touch) at any point other than at a vertex.

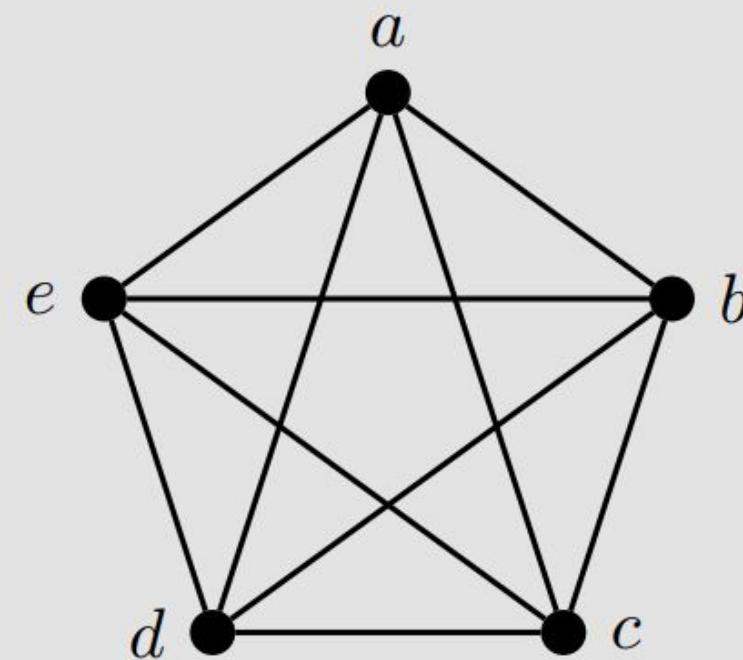


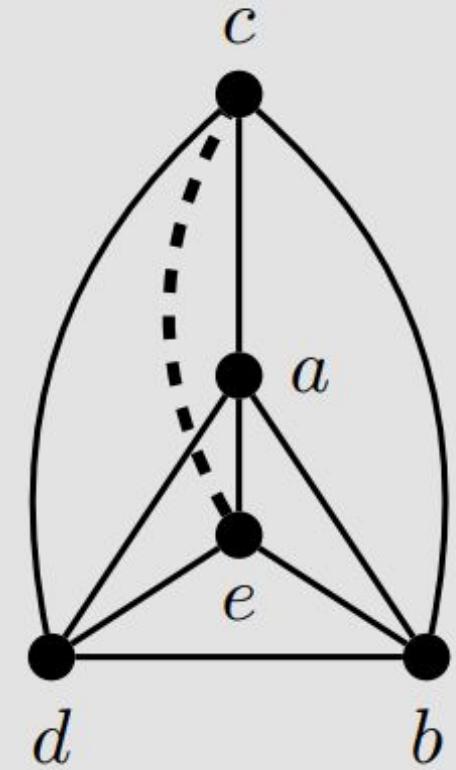
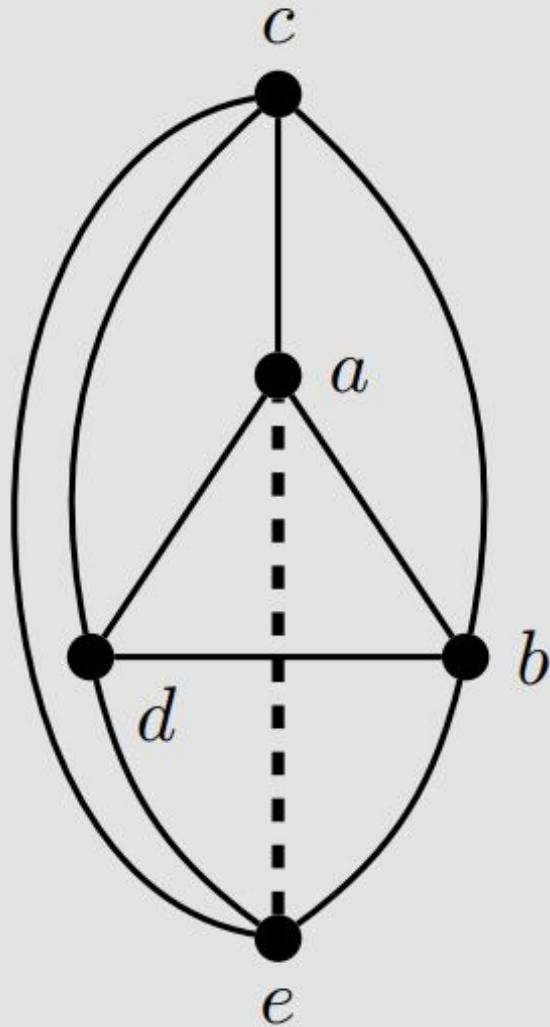
K_4



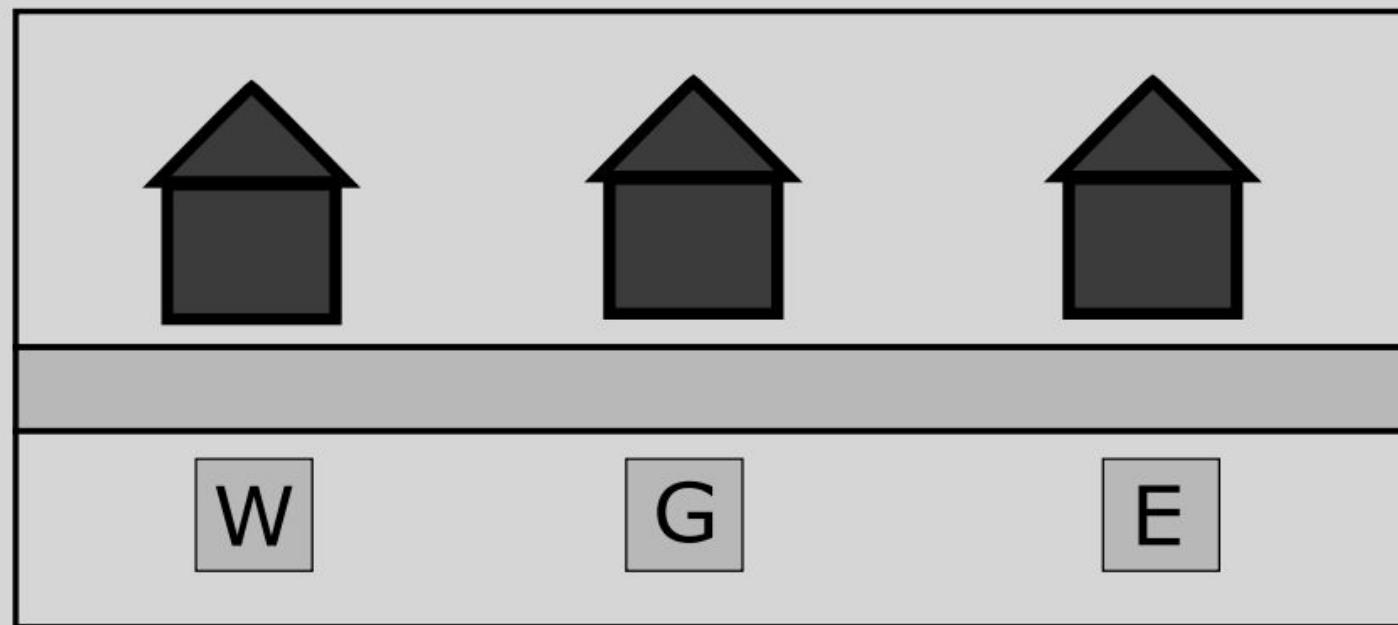
planar drawing of K_4

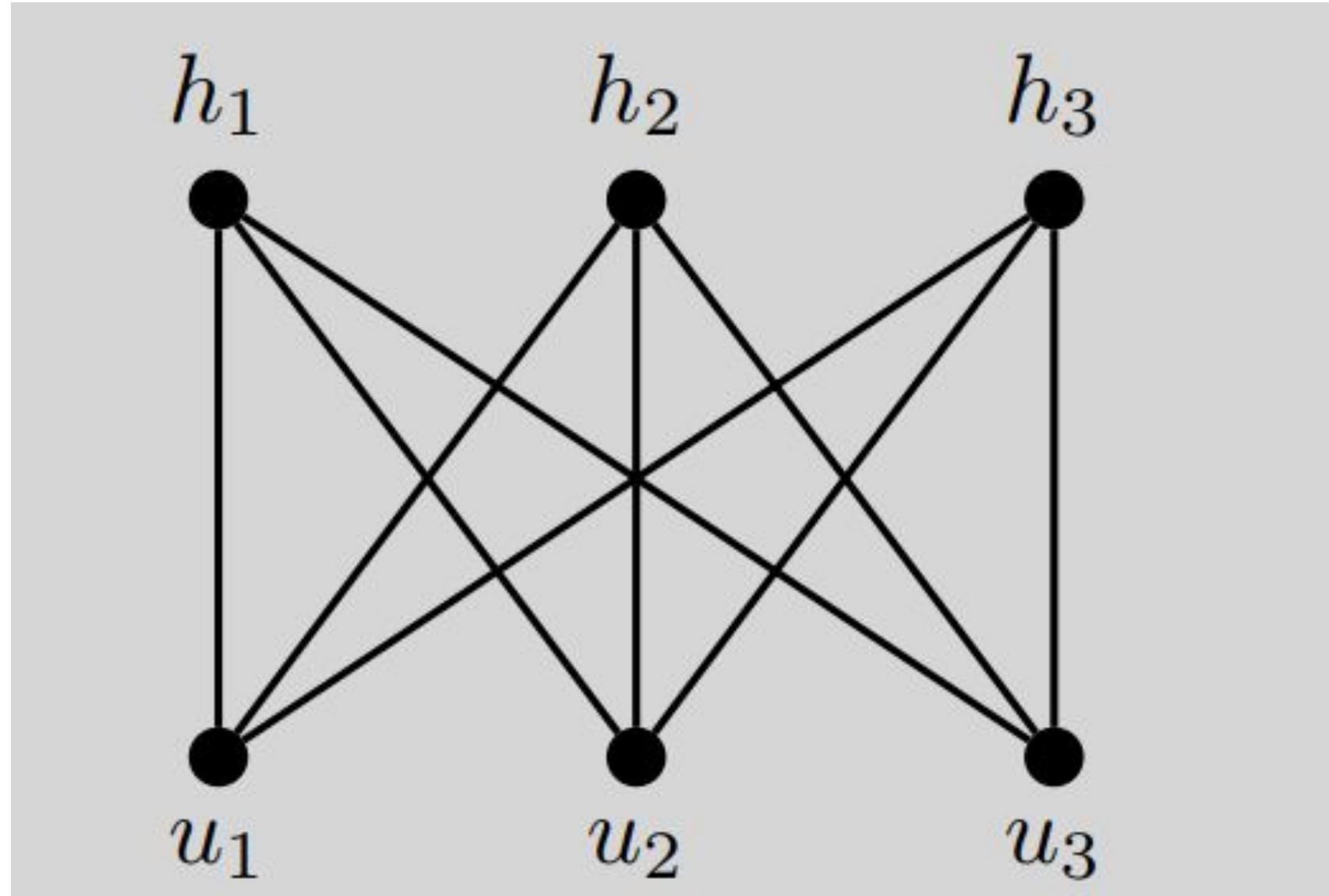
Example 7.2 Determine if K_5 is planar. If so, give a planar drawing; if not, explain why not.

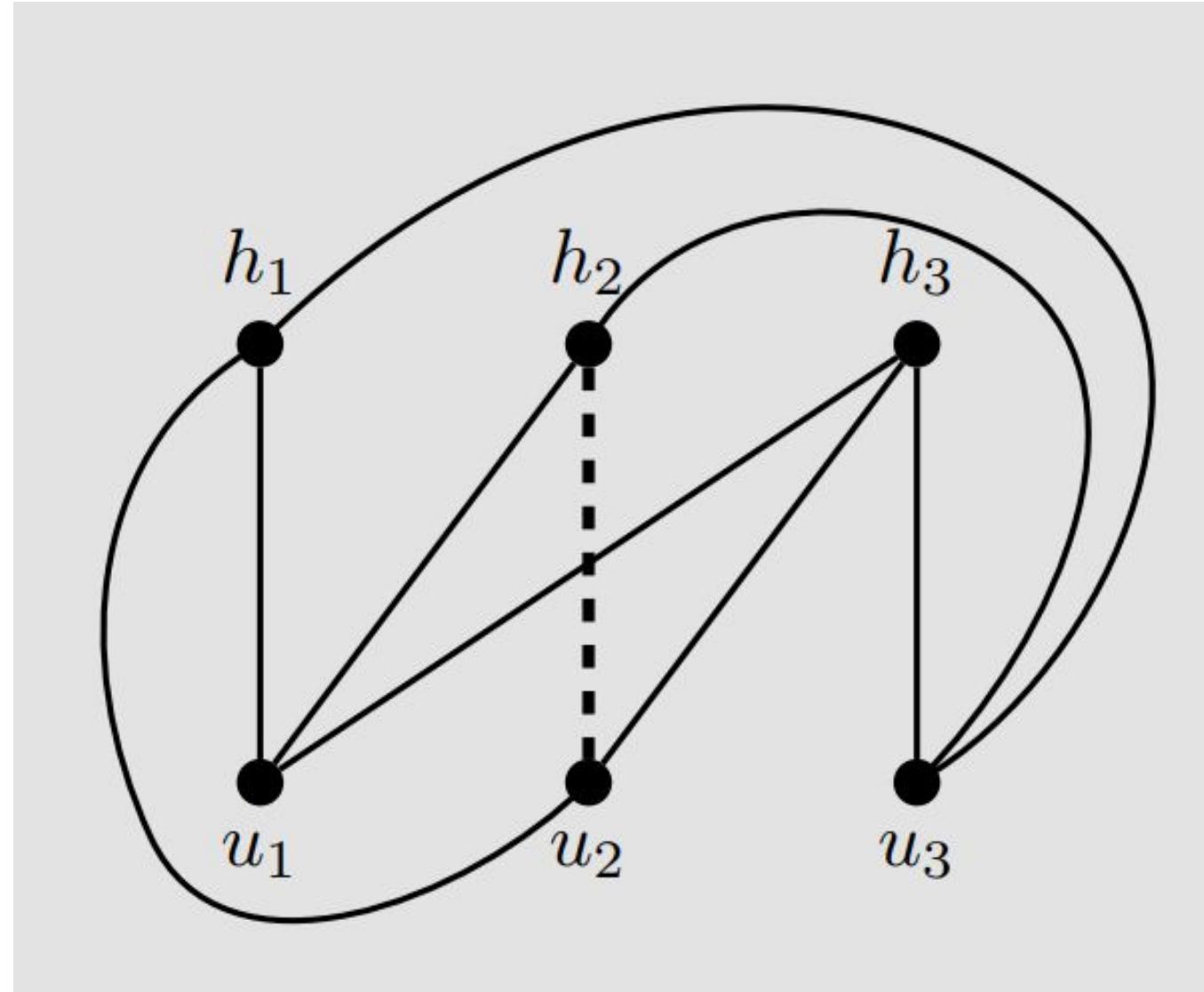




Example 7.1 Three houses are set to be built along a new city block; across the street lie access points to the three main utilities each house needs (water, electricity, and gas). Is it possible to run the lines and pipes underground without any of them crossing?

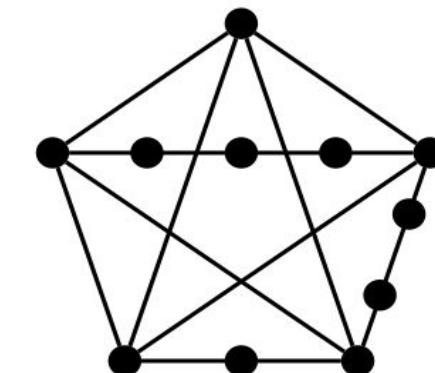
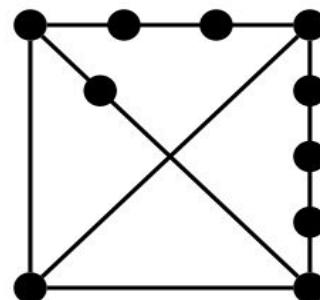
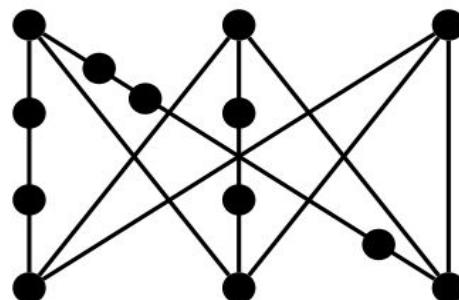






Definition 7.2 A *subdivision* of an edge xy consists of inserting vertices so that the edge xy is replaced by a path from x to y . The subdivision of a graph G is obtained by subdividing edges in G .

Note that a subdivision of a graph can be obtained by subdividing one, two, or even all of its edges. However, the new vertices placed on the edges from G cannot appear in more than one subdivided edge. Below are three examples of subdivided graphs. The graph on the left shows a subdivision of $K_{3,3}$, the graph in the middle shows a subdivision of K_4 , the graph on the right shows a subdivision of K_5 .



Theorem 7.3 (Kuratowski's Theorem) A graph G is planar if and only if it does not contain a subdivision of $K_{3,3}$ or K_5 .

Definition 7.4 Given a planar drawing of a graph G , a *region* is a portion of the plane completely bounded by the edges of the graph.

Theorem 7.5 (Euler's Formula) If G is a connected planar graph with n vertices, m edges, and r regions then $n - m + r = 2$.

Definition 7.6 A graph G is *maximally planar* if $G + e$ is nonplanar for any edge $e = xy$ for any two nonadjacent vertices $x, y \in V(G)$.

Theorem 7.5 (Euler's Formula) If G is a connected planar graph with n vertices, m edges, and r regions then $n - m + r = 2$.

Proof: Argue by induction on m , the number of edges in the graph. If $m = 1$ then since G is connected it is either a tree with one edge and so $n = 2$ and $r = 1$, or G is a graph with a loop, and so $n = 1$ and $r = 2$. In either case, Euler's Formula holds.

Now suppose Euler's Formula holds for all graphs with $m \geq 1$ edges and consider a graph G with $m + 1$ edges, n vertices, and r regions. We will consider the graph that is formed with the removal of an edge from G .

First, if $G' = G - e$ is not connected for any edge e in G then we know e must be a bridge of G . Thus G must be a tree, and so $n = m + 1$ and $r = 1$. Therefore $n - m + r = m + 1 - m + 1 = 2$.

Next, if $G' = G - e$ is connected for some edge e of G , then e must be a part of some cycle in G . Then there must be two different regions on the two sides of e in G , but these regions join into just one with the removal of e . Thus G' has n vertices, $r - 1$ regions, and $m - 1$ edges. By the induction hypothesis applied to G' , we know $n - (m - 1) + (r - 1) = 2$, which simplifies to $n - m + r = 2$.

Thus by induction we know Euler's Formula holds for all connected planar graphs.

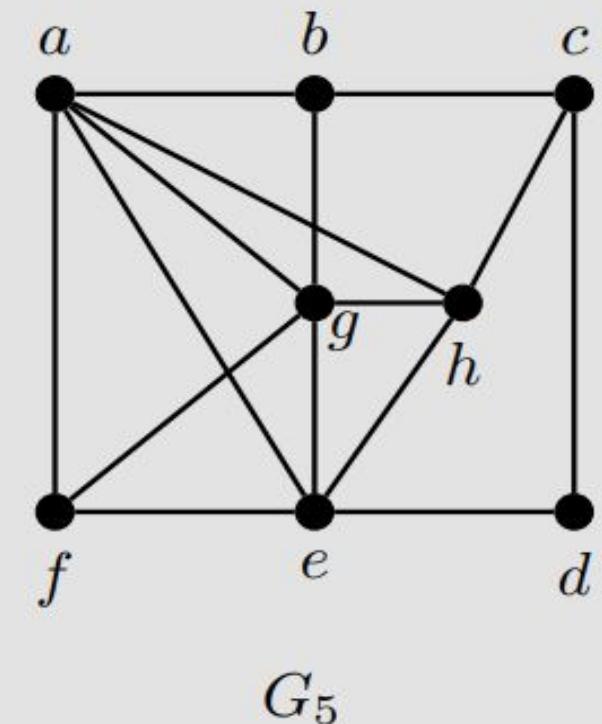
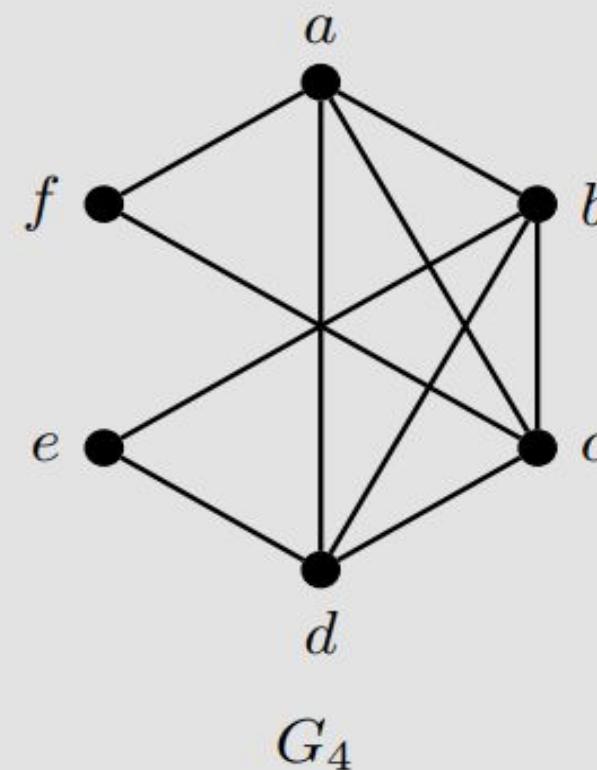
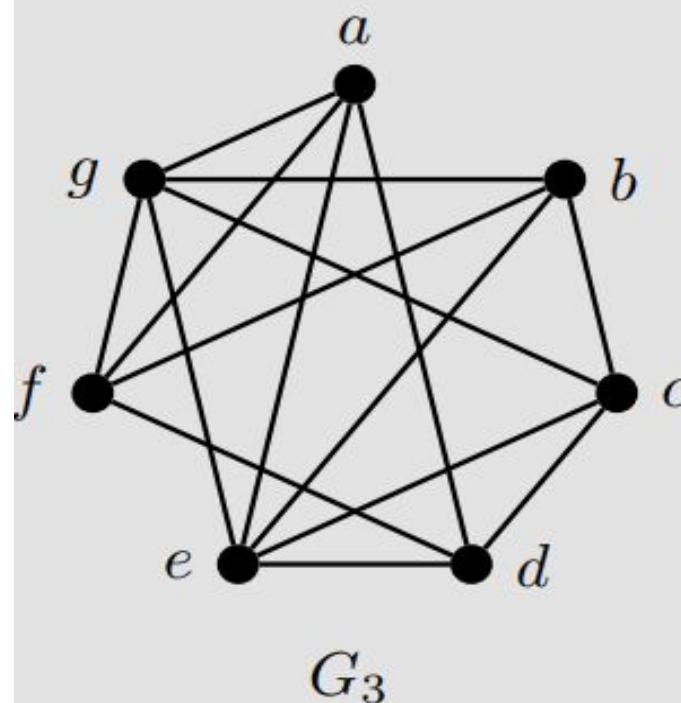
Theorem 7.7 If G is a maximally planar simple graph with $n \geq 3$ vertices and m edges, then $m = 3n - 6$.

Theorem 7.8 If $G = (V, E)$ is a simple planar graph with m edges and $n \geq 3$ vertices, then $m \leq 3n - 6$.

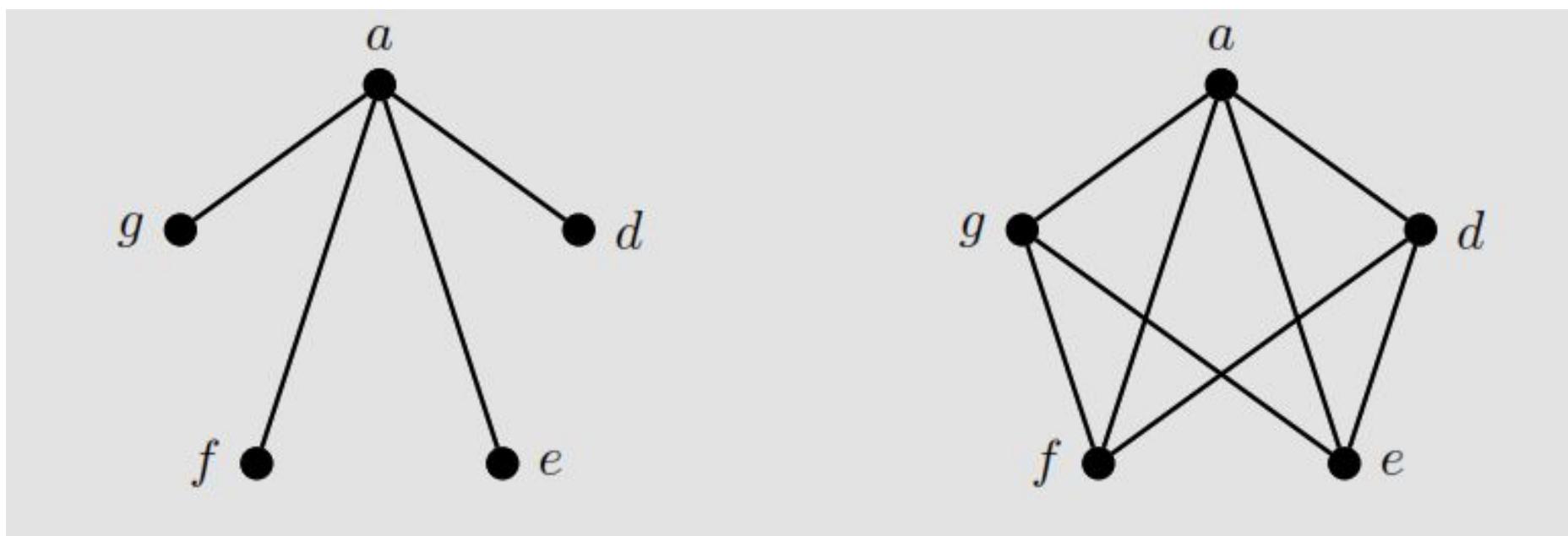
Theorem 7.9 If $G = (V, E)$ is a simple planar graph with m edges and $n \geq 3$ and no cycles of length 3, then $m \leq 2n - 4$.

These theorems are less about verifying the relationship between edges and vertices when a graph is known to be planar, but rather in determining if a graph satisfies this inequality. If a graph does not satisfy the inequality, then it must be nonplanar; however, if the graph satisfies the inequality, then it may or may not be planar and further investigations are needed to determine planarity. The proof of Theorem 7.9 appears in Exercise 7.10.

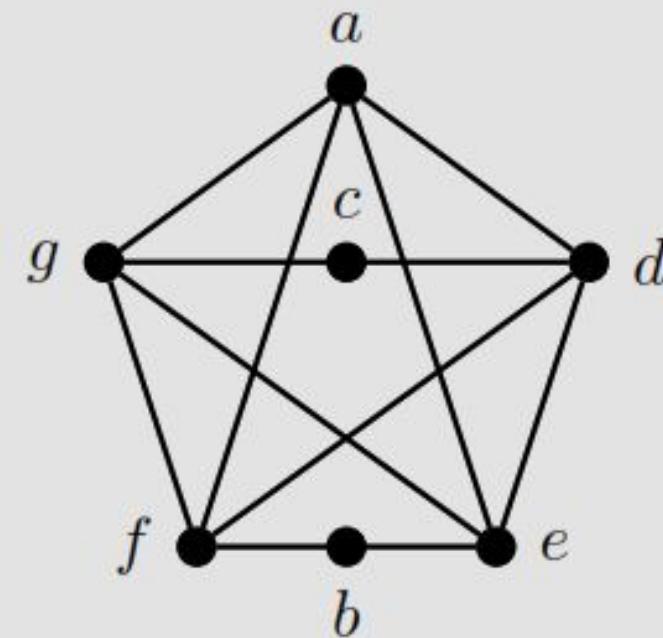
Example 7.3 Determine which of the following graphs are planar. If planar, give a drawing with no edge crossings. If nonplanar, find a $K_{3,3}$ or K_5 subdivision.



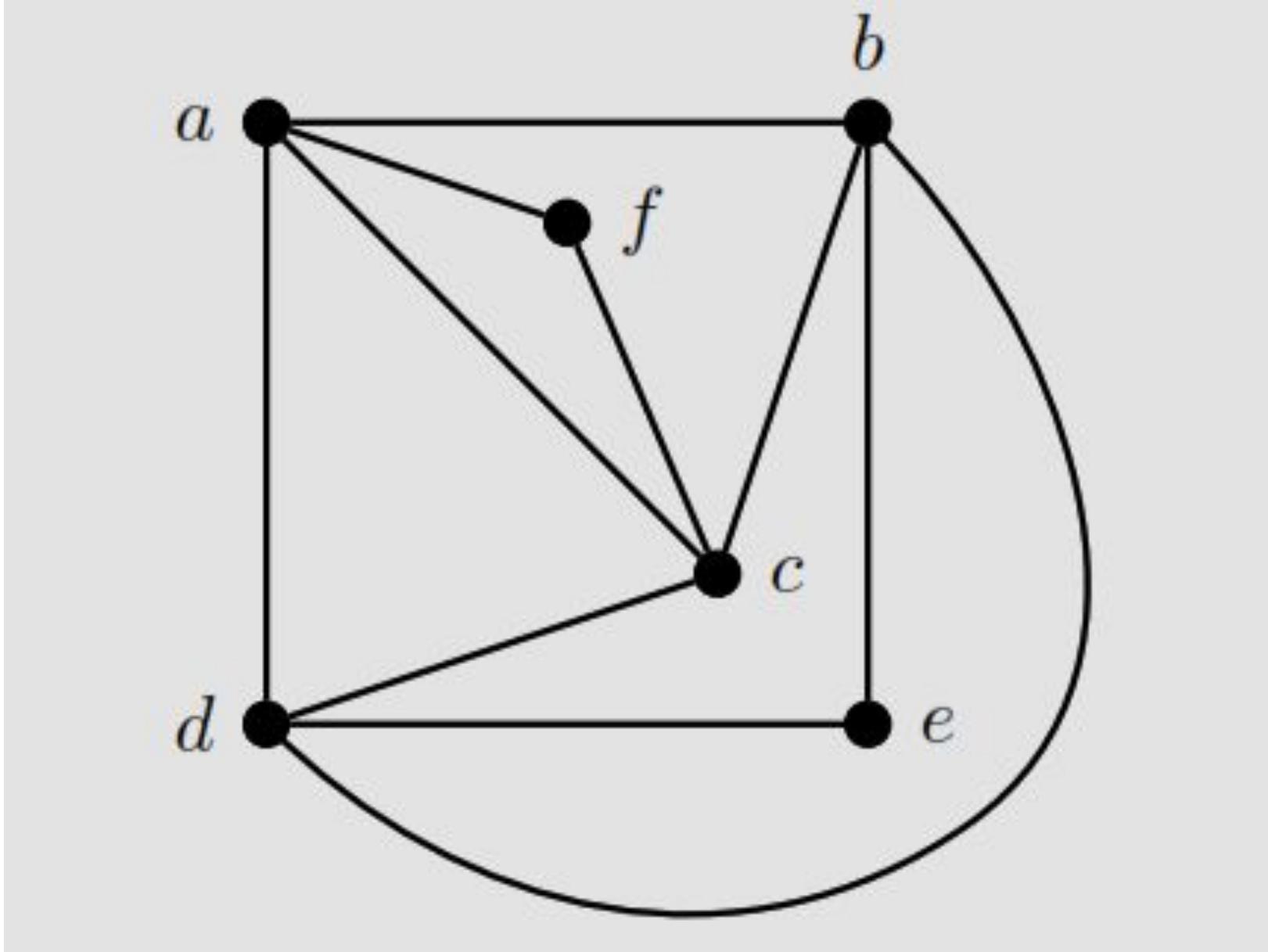
For G_3 , we have $|E| = 15$ and $|V| = 7$, giving us the inequality from Theorem 7.8 of $15 \leq 3 \cdot 7 - 6 = 15$. Thus the number of edges is as high as possible for a graph with 7 vertices. Although this does not guarantee the graph is nonplanar, it provides good evidence that we should search for a K_5 or $K_{3,3}$ subdivision. Also notice that every vertex has degree at least 4, so we will begin by looking for a K_5 subdivision.



At this point, we are only missing two edges in forming K_5 , namely dg and ef . We have two vertices available to use for paths between these nonadjacent vertices, and using them we find a K_5 subdivision. Thus G_3 is nonplanar.



For G_4 , we have $|E| = 11$ and $|V| = 6$, giving us the inequality from Theorem 7.8 of $11 \leq 3 \cdot 6 - 6 = 12$. We cannot deduce from this result that the graph is nonplanar. However, notice that two vertices have degree 2 and the remaining 4 vertices have degree 4. There cannot be a K_5 subdivision since there are not enough vertices of degree at least 4 and there cannot be a $K_{3,3}$ subdivision since there are not enough vertices of degree at least 3. Thus we can conclude that G_4 is in fact planar. A planar drawing is shown below.

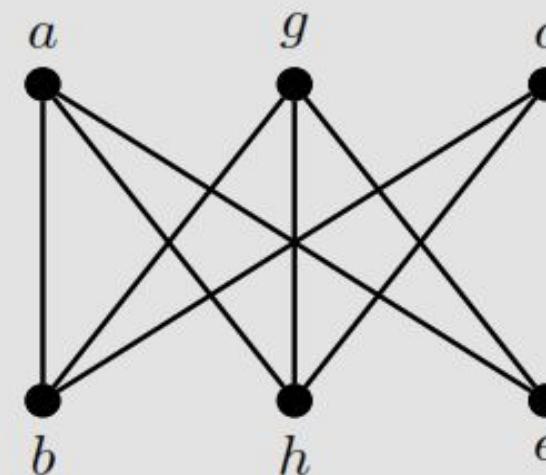
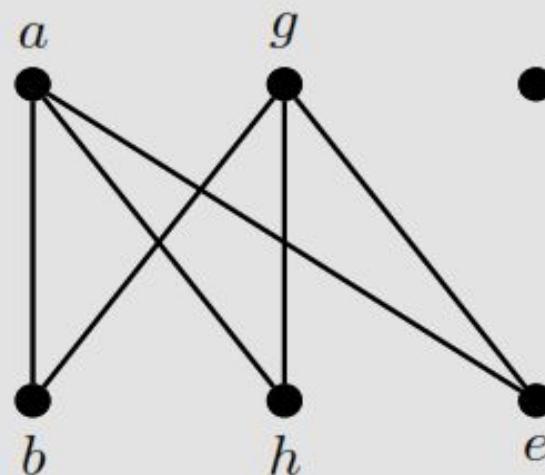


For G_5 , we have $|E| = 15$ and $|V| = 8$, giving us the inequality from Theorem 7.8 of $15 \leq 3 \cdot 8 - 6 = 18$. As in the previous examples, we cannot deduce that the graph is nonplanar from Euler's Theorem but the high number of edges relative to the number of vertices should give us some suspicion that the graph may not be planar.

When inspecting the vertex degrees, we see only four vertices of degree at least 4 (namely a, e, g, h), indicating the graph cannot contain a K_5 subdivision. However, there are another three vertices of degree 3, allowing for a possibility of a $K_{3,3}$ subdivision.

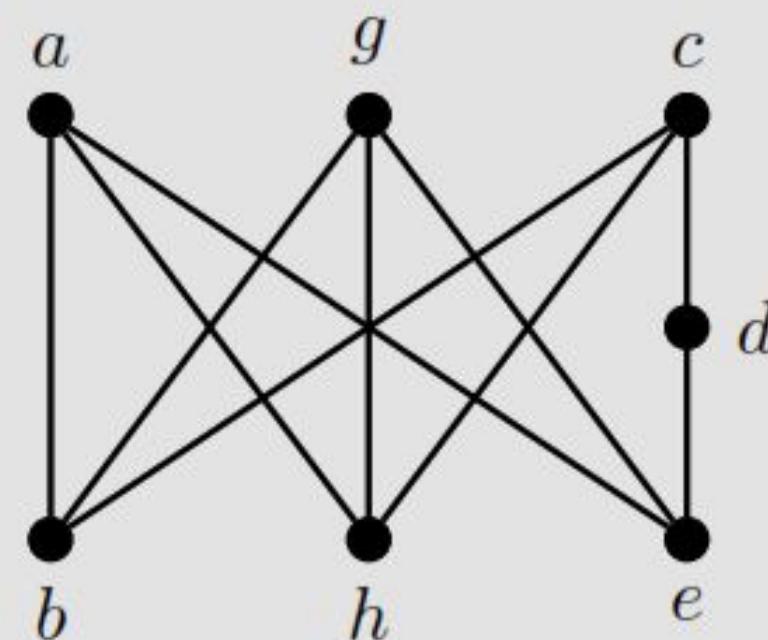
Again, we start by selecting vertex a to be one of the main vertices of a possible $K_{3,3}$ subdivision. At the same time, we will look for another vertex that is adjacent to three of the neighbors of a . We see that a and g are both adjacent to b, h, e , and f . Let us begin with b, h , and e for the vertices on the other side of the $K_{3,3}$, as shown below on the left.

We now search among the remaining vertices (f, d, c) for one that is adjacent to as many of b, h , and e as possible and find c is adjacent to both b and h , as shown below on the right.

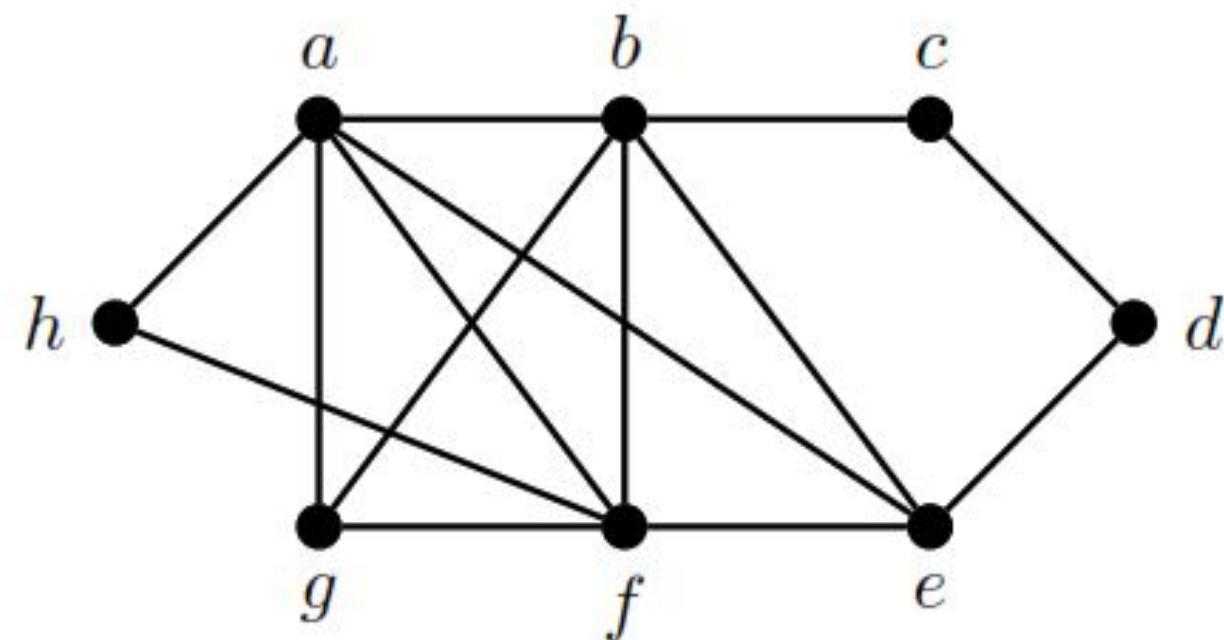


At this point we are only missing one edge to form a $K_{3,3}$ subdivision, namely ce . Luckily we can form a path from c to e using the available

vertex d . Thus we have found a $K_{3,3}$ subdivision and proven that G_5 is nonplanar.

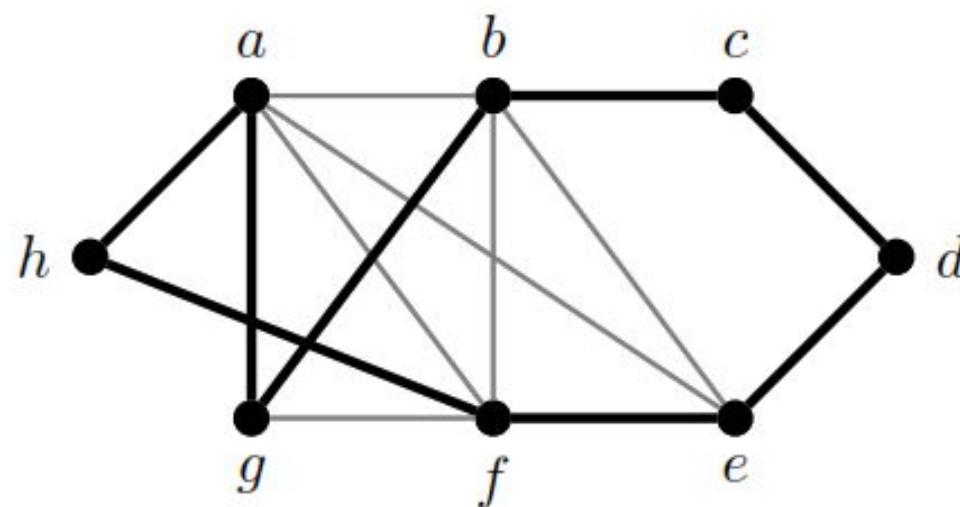


Cycle-Chord Method

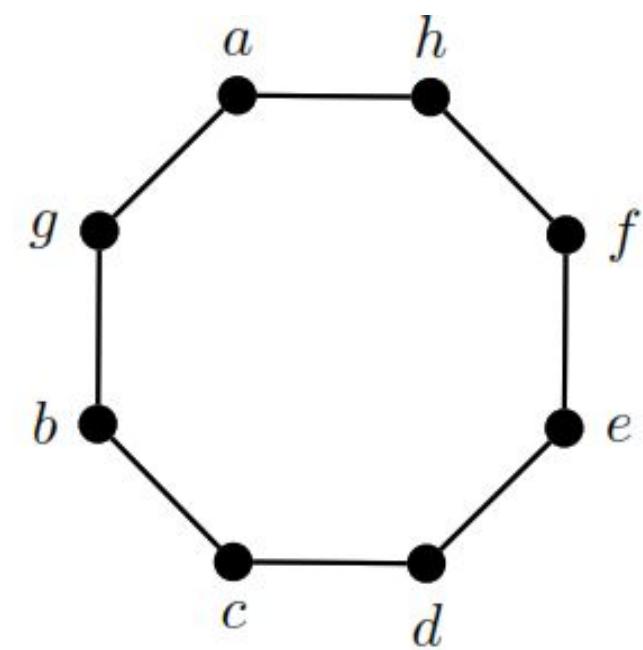


G_6

To begin, put the vertices in a **circular pattern**, but with some care in their arrangement. We want to find a **spanning cycle** (also called a **hamiltonian cycle**) or something approximating a spanning cycle, when placing the vertices. **The edges in bold on the left represent those that are currently being placed in the planar drawing; the gray edges are ones not yet placed.**

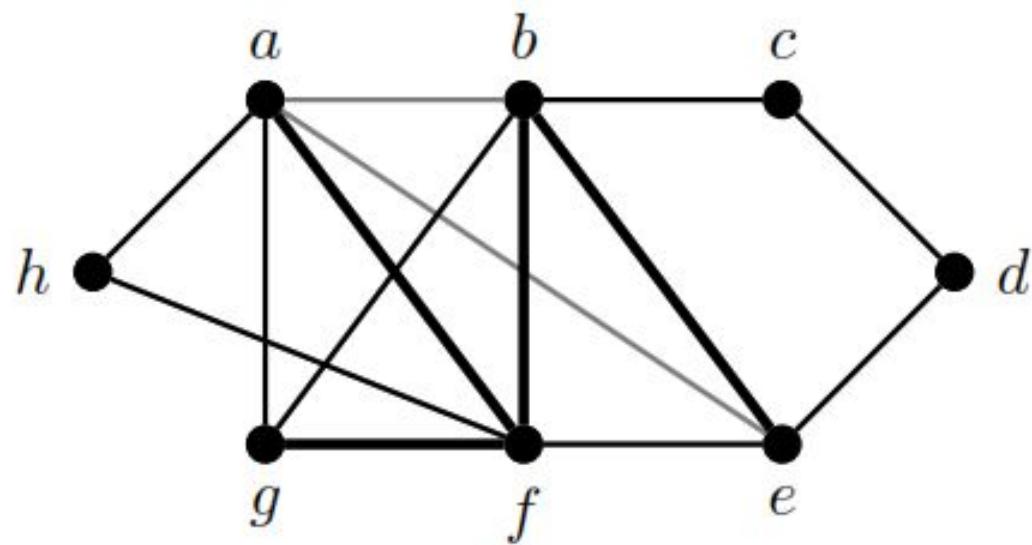


highlighted portion of G_6

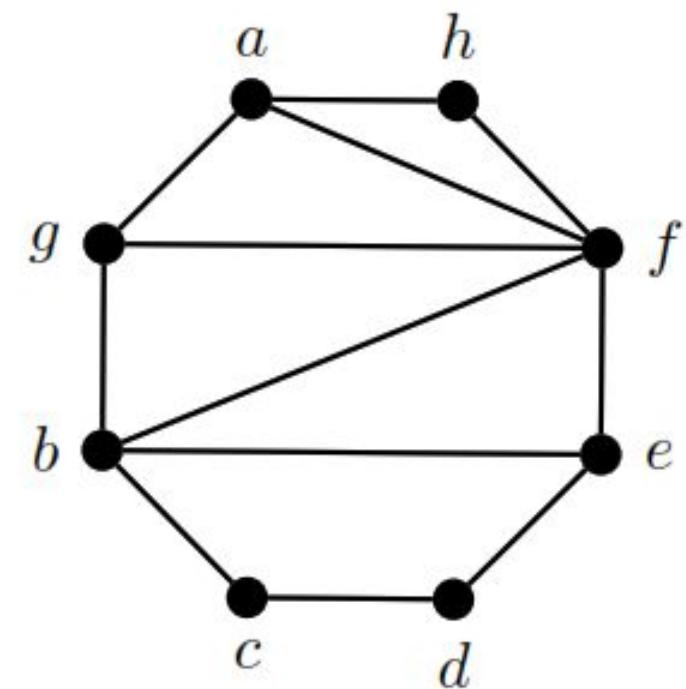


planar drawing

Once we have created the spanning cycle, we attempt to place as many edges in the interior of the cycle as possible so that they do not cross. This should resemble chords of a circle. In the example G_6 below, we are making most of the interior regions of the cycle into triangulations.

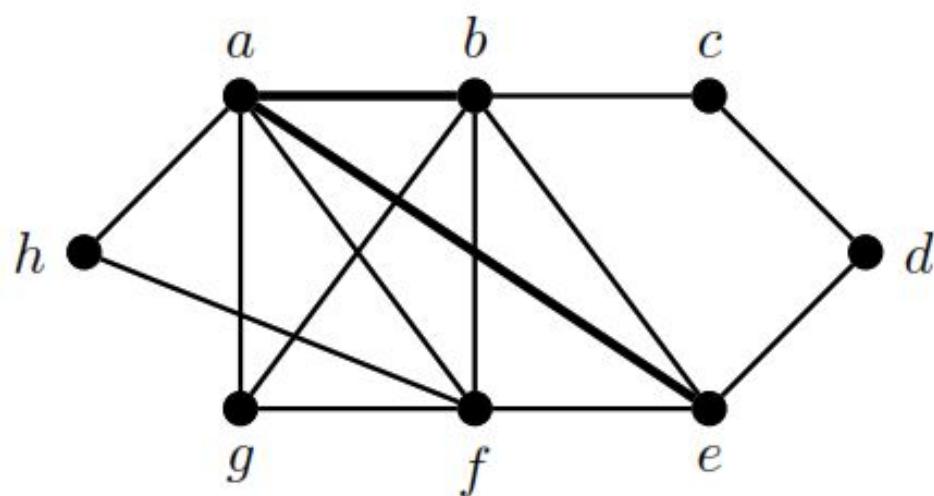


highlighted portion of G_6

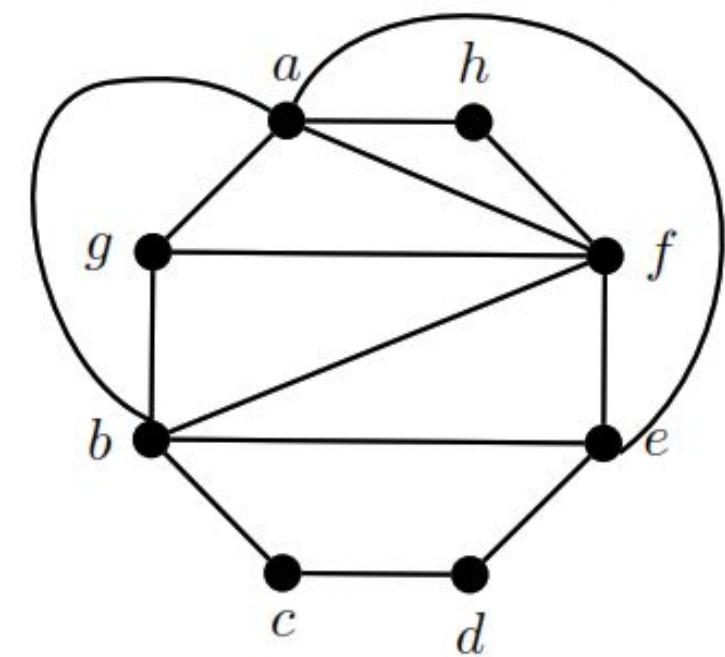


planar drawing

When placing these chords, we should take care to notice any vertices that have incident edges remaining, as those will need to be placed as curves along the outside of the cycle. For example, in G_6 only two edges remain to be placed outside the spanning cycle, namely ab and ae . Having multiple edges from the same vertex left to place is often a benefit since they can be drawn in the same or opposite directions of the cycle without creating edge crossings.



highlighted portion of G_6



planar drawing

This procedure is not perfect, and we may discover that we have made a poor choice as to which edges should be going through the interior of the cycle versus along the outside. When this happens, we simply try again using the knowledge gained from our first attempt.

Obviously this method is not written as an efficient algorithm and would be unwieldy for graphs of large size. Luckily, for the purposes of this book we will only focus on graphs of small size when finding planar drawings.

Lemma 7.10 Let H be a subgraph of G . Then G is nonplanar if H is nonplanar.

Lemma 7.11 Let G' be a subdivision of G . Then G is planar if and only if G' is planar.

Lemma 7.12 A graph G with at least 3 vertices is 2-connected if and only if for every pair of vertices x and y there exists a cycle through x and y .

Home Task

Theorem 7.3 (Kuratowski's Theorem) A graph G is nonplanar if and only if it contains a subdivision of $K_{3,3}$ or K_5 .

Proof: First suppose G contains a subdivision of $K_{3,3}$ or K_5 . By our discussions above we know $K_{3,3}$ and K_5 are nonplanar. By Lemma 7.11, this means any subdivision of $K_{3,3}$ and K_5 must also be nonplanar, and so by Lemma 7.10 we know G is also nonplanar.

Conversely, suppose for a contradiction that there exists a nonplanar graph G that does not contain a subdivision of $K_{3,3}$ or K_5 . Choose G to be a minimal such graph; that is, any graph with fewer vertices or edges that does not contain a subdivision of $K_{3,3}$ or K_5 must be planar. Note that since G is nonplanar then it must contain a nonplanar block B . If B is a proper subgraph of G , then G would not be minimal. Thus we know that G must itself be 2-connected.

Claim 1: Every vertex of G have degree at least 3.

Proof: Suppose for a contradiction that there exists some vertex v that does not have degree 3. We know $\deg(v) \geq 2$ since G is 2-connected, and so $\deg(v) = 2$. Let x and y be the two distinct neighbors of v . If x and y are adjacent, then $G' = G - v$ must be planar by the minimality of G . But then there must be a region for which the edge xy is part of its boundary, and the path $xv y$ can be inserted into this region, making G planar, which is a contradiction.

Thus x and y cannot be adjacent. Define $G' = G - v + xy$; that is, G' is the graph obtained from G by removing vertex v and adding the edge xy . Since G' has fewer vertices than G , we know it must be planar. But then we can obtain a planar drawing of G from that of G' by subdividing the edge xy with the vertex v .

This is also a contradiction, so every vertex of G must have degree at least 3.

Claim 2: There exists an edge $e = xy$ such that $G' = G - e$ is 2-connected.

Proof: Since every vertex of G has degree at least 3, we know G cannot simply be a cycle. Moreover, since G is 2-connected we know G has an ear decomposition by Theorem 4.24. Then last path added to the decomposition must be a singular edge, as otherwise any internal vertex of the path would have degree 2 in G . Thus the graph obtained by removing this edge will remain 2-connected as it still has an ear decomposition.

Let x, y be chosen so that $G' = G - e$ is 2-connected. Then by the minimality of G we know G' must be planar. By Corollary 4.18, there exists a cycle C in G' that contains both x and y . Consider the planar drawing of G' with C chosen so that:

- (i) it contains x and y ,
- (ii) the number of regions inside C is maximal among all planar drawings of G' ,
- (iii) given any other cycle C' containing x and y , no planar drawing of C' has more regions inside of it than that of C .

Let $C = v_0 v_1 \cdots v_k v_{k+1} \cdots v_l v_0$, where $v_0 = x$ and $v_k = y$. Since x and y are not adjacent in G' , we know $k \geq 2$.

Edge-Crossing

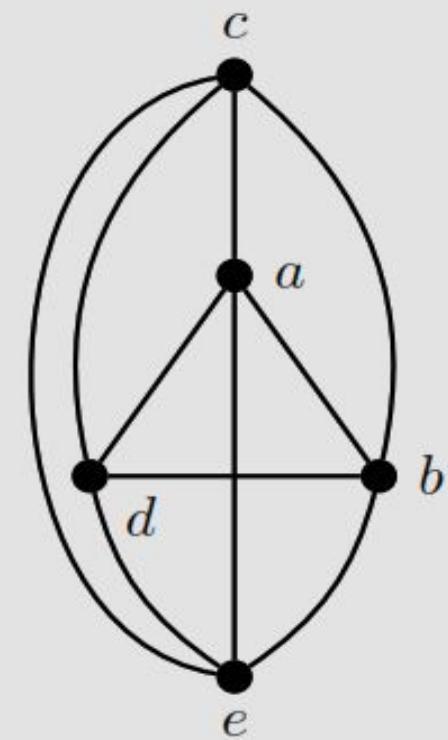
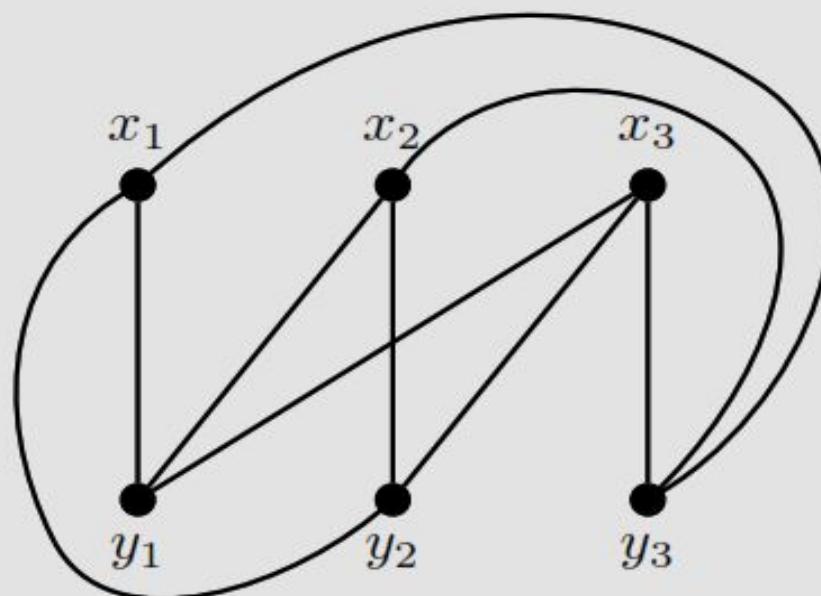
Definition 7.18 For any simple graph G the *crossing number* of G , denoted $cr(G)$, is the minimum number of edge crossings in any drawing of G satisfying the conditions below:

- (i) no edge crosses another more than once, and
- (ii) at most two edges cross at a given point.

It should be clear that $cr(G) = 0$ if and only if G is planar. But what about K_5 and $K_{3,3}$, the two graphs that are instrumental in determining planarity?

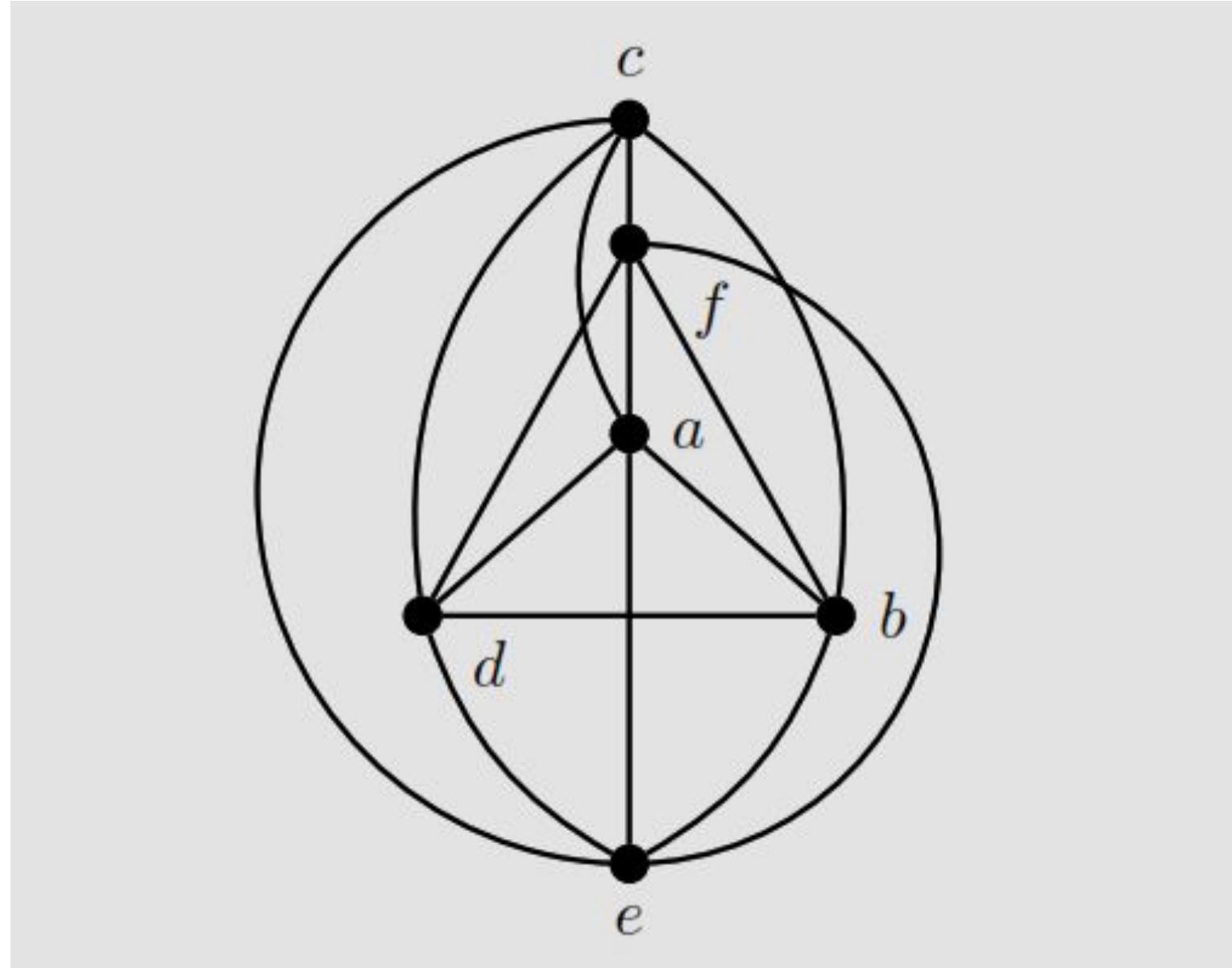
Example 7.4 Determine the crossing numbers for K_5 and $K_{3,3}$.

Solution: Since we know K_5 and $K_{3,3}$ are not planar, $cr(K_5), cr(K_{3,3}) \geq 1$. A drawing of each of these adhering to the criteria above is shown below, proving they each have crossing number 1.

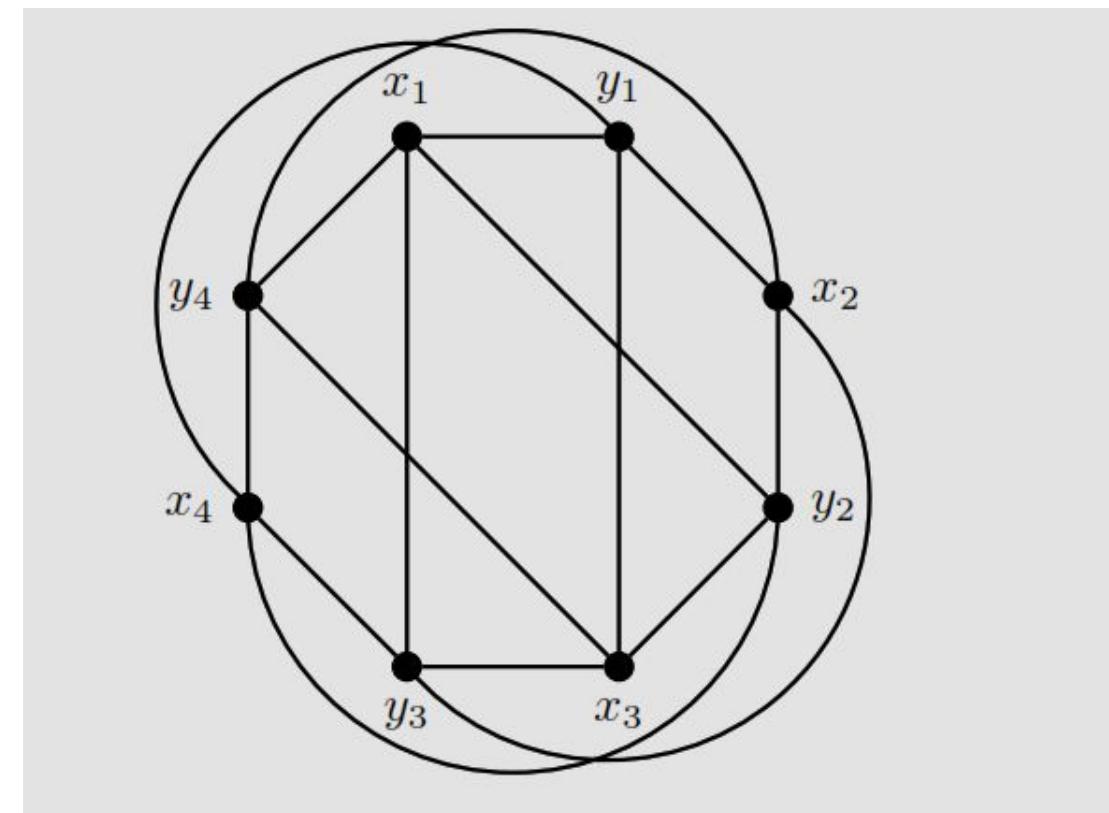


Theorem 7.19 Let G be a simple graph with m edges and n vertices. Then $cr(G) \geq m - 3n + 6$. Moreover, if G is bipartite then $cr(G) \geq m - 2n + 4$.

Example 7.5 Find the crossing number for K_6 and $K_{4,4}$.



Next, we see that $K_{4,4}$ has 16 edges and so by Theorem 7.7, we know $cr(K_{4,4}) \geq 16 - 2 * 8 + 4 = 4$. Since the drawing below of $K_{4,4}$ has 4 edge crossings, we know $cr(K_{4,4}) = 4$.



Unfortunately, there is no clear formula for the crossing number of a graph, even for complete graphs! The two results below give upper bounds for K_n and $K_{m,n}$ in terms of the floor function (see Appendix B).

Theorem 7.20

$$cr(K_n) \leq \frac{1}{4} \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor$$

The Polish mathematician Kazimierz Zarankiewicz proved a similar upper bound for bipartite graphs (shown below) in 1954, though he conjectured this quantity should also be the exact formula for the crossing number of complete bipartite graphs [87].

Theorem 7.21

$$cr(K_{m,n}) \leq \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{m-1}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor$$

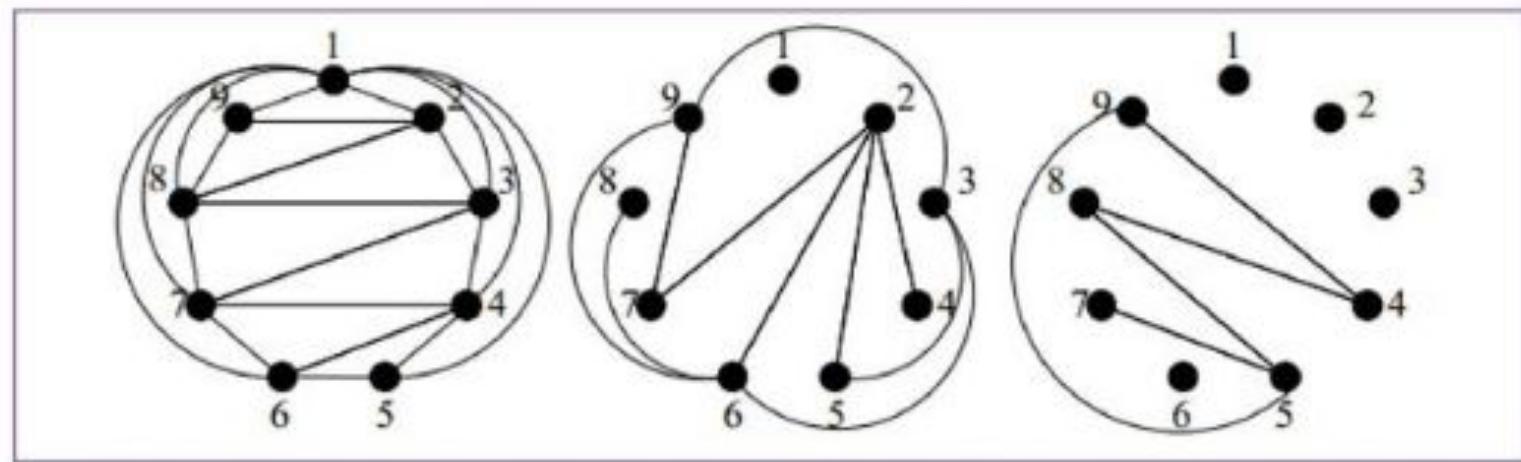
Thickness

Definition 7.22 Let $T = \{H_1, H_2, \dots, H_t\}$ be a set of spanning subgraphs of G so that each H_i is planar and every edge of G appears in exactly one graph from T . The ***thickness*** of a graph G , denoted $\theta(G)$, is the minimum size of T among all possible such collections.

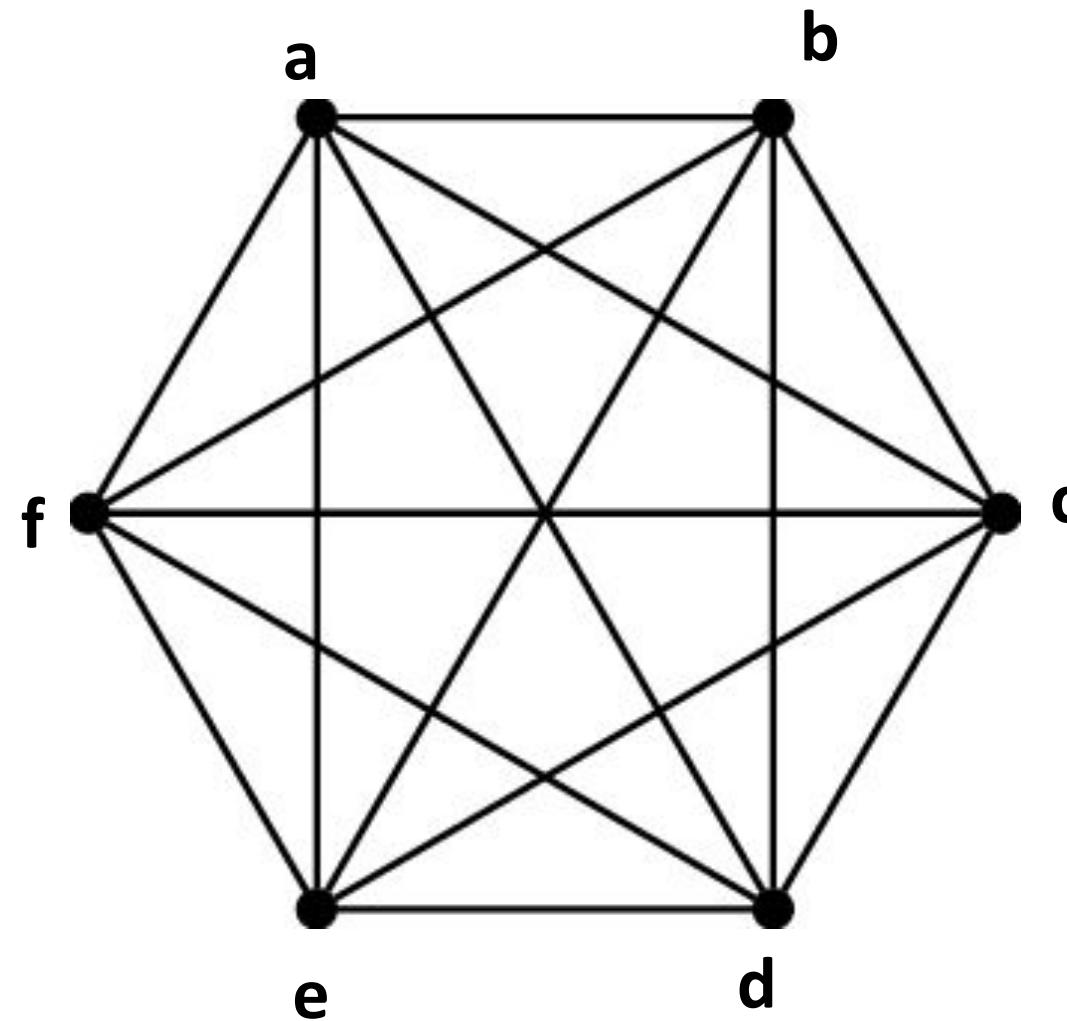
Thickness:

The smallest integer k such that G can be decomposed into k planar graphs.

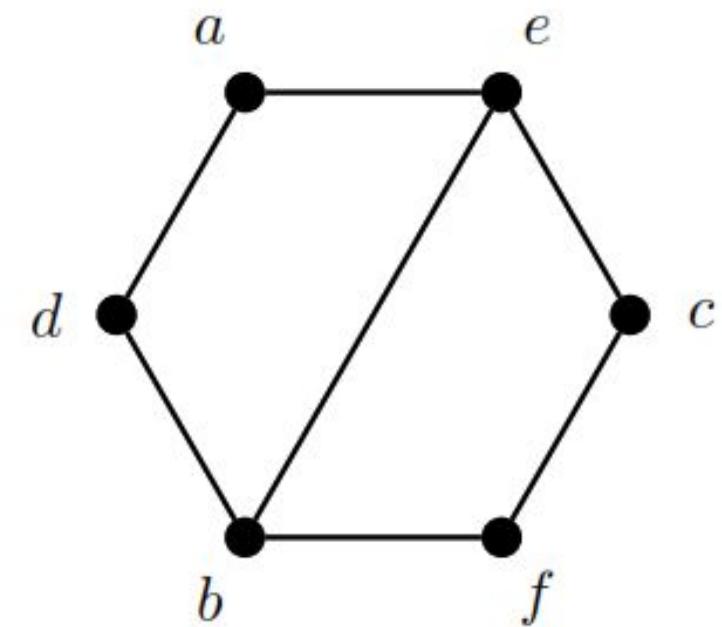
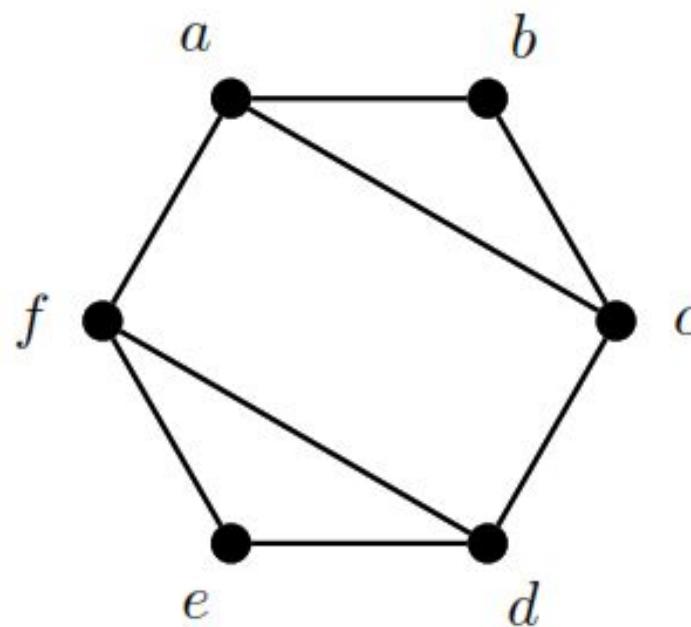
K_9 is a thickness-3 graph



Thickness



Clearly $\theta(G) = 1$ if and only if G is planar, since T would contain only G itself. Below is a decomposition of K_6 into two planar spanning subgraphs and since we know that K_6 is not planar we have shown $\theta(K_6) = 2$.



Corollary 7.23 Let G be a connected simple graph with n vertices and m edges. Then

$$\theta(G) \geq \left\lceil \frac{m}{3n - 6} \right\rceil$$

Corollary 7.24 Let G be a connected simple bipartite graph with n vertices and m edges. Then

$$\theta(G) \geq \left\lceil \frac{m}{2n - 4} \right\rceil$$

While a general formula is not known for the thickness of a graph, the theorem below does establish the thickness for a complete graph (and so could serve as an upper bound for any graph on n vertices). This result is based on the work from [1],[3],[4], and [81].

Theorem 7.25

$$\theta(K_n) = \begin{cases} \left\lfloor \frac{n+7}{6} \right\rfloor & n \neq 9, 10 \\ 3 & n = 9, 10 \end{cases}$$

نگاه بلند سخن دلنواز جان پُرسوز
یہی ہے رختِ سفر میر کاروان کیائے