

Mining Graph Data

# GRAPH THEORY REVIEW

Teacher: Le Ngoc Thanh

Email: [lnthanh@fit.hcmus.edu.vn](mailto:lnthanh@fit.hcmus.edu.vn)

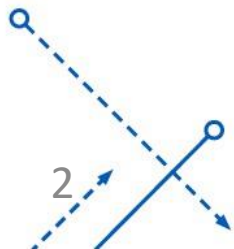


**fit@hcmus**

# Contents

---

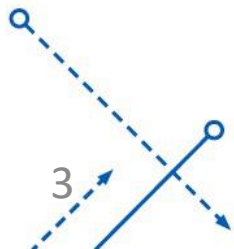
- Graph concepts and properties
- Types of graphs
- Graph storage
- Some basic algorithms on graphs
  - Graph Traversal Algorithm
  - Topological Sorting Algorithm
  - Spanning Tree Algorithm
  - Shortest Path Algorithm
- Slices and Threads
- Graph Match



# Review questions

---

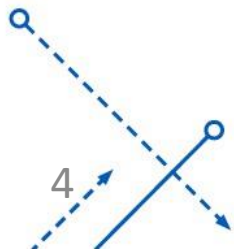
- Question 1: What components does graph  $G$  consist of?
- Question 2: What is a subgraph?
- Question 3: When are two vertices of a graph called adjacent?
- Question 4: What is a path between two vertices?
- Question 5: What is a simple path?
- Question 6: What is a cycle?
- Question 7: What is a simple cycle?



# Review questions

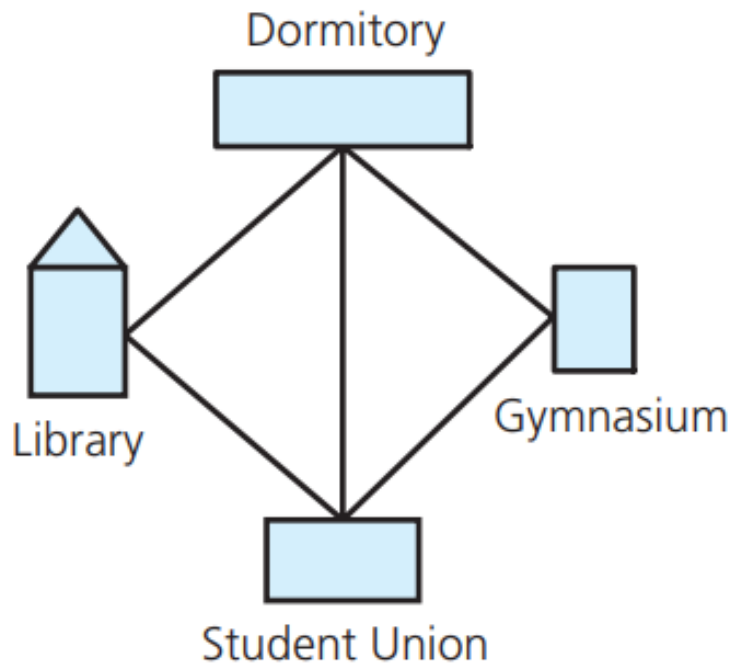
---

- Q: A vertex of a graph is called even or odd depending upon?
  - A) Total number of edges in a graph is even or odd
  - B) Total number of vertices in a graph is even or odd
  - C) Its degree is even or odd
  - D) D None of these



# Graph

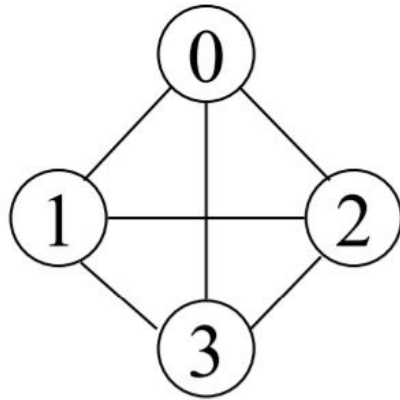
- A graph  $G$  consists of two sets: a set  $V$  containing vertices (vertices, nodes), and a set  $E$  containing edges connecting vertices.



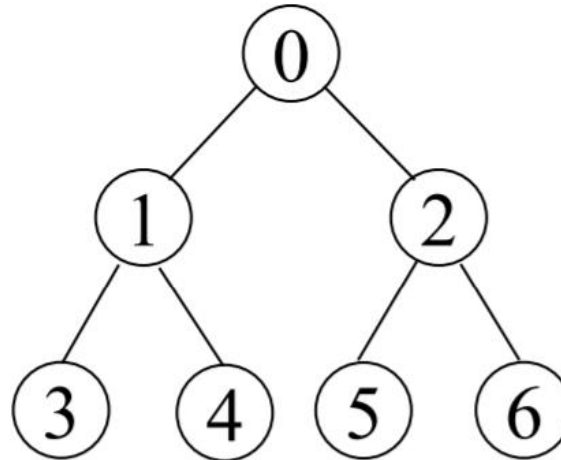
For example, a map of a University is a graph with vertices being buildings and edges being paths between buildings.



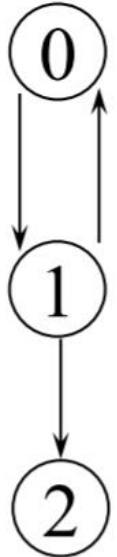
# Example for a graph



$G_1$



$G_2$



$G_3$

$$V(G_1) = \{0, 1, 2, 3\}$$

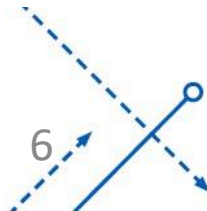
$$V(G_2) = \{0, 1, 2, 3, 4, 5, 6\}$$

$$V(G_3) = \{0, 1, 2\}$$

$$E(G_1) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)\}$$

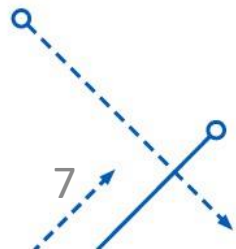
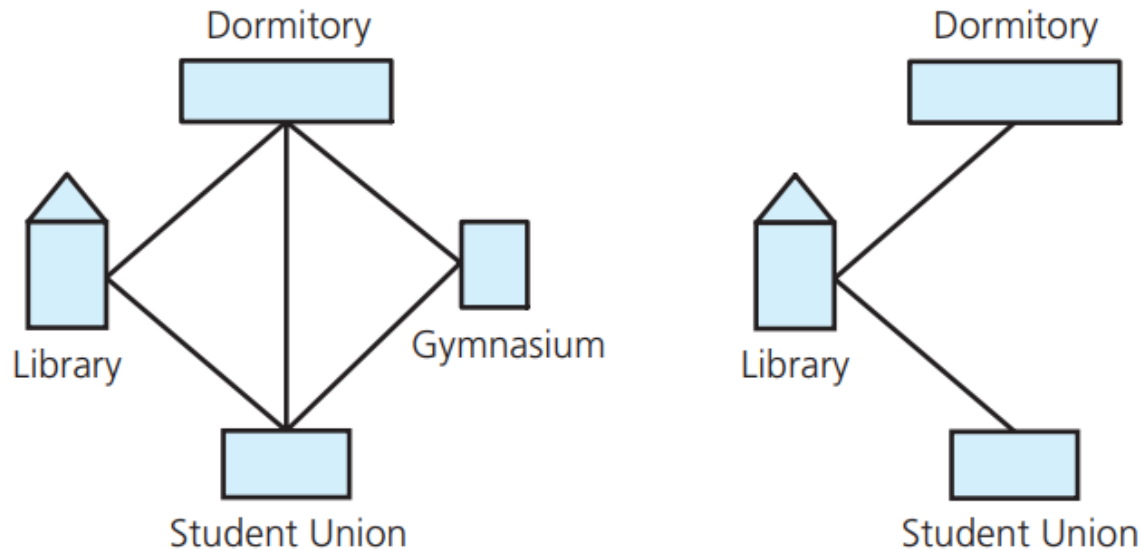
$$E(G_2) = \{(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)\}$$

$$E(G_3) = \{<0, 1>, <1, 0>, <1, 2>\}$$



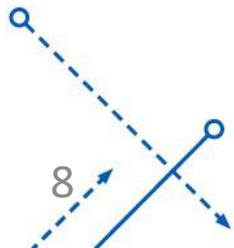
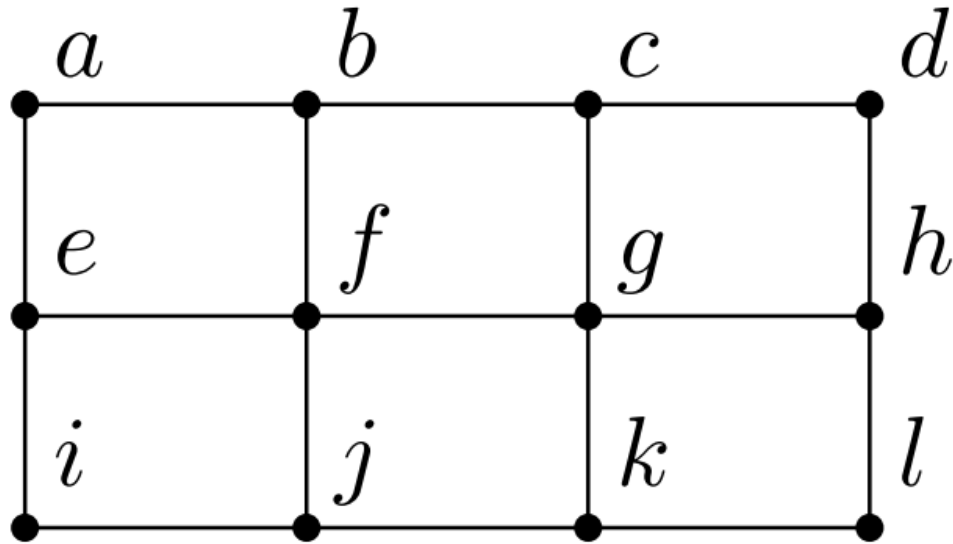
# Subgraph

- The **subgraph** consists of a subset of the vertices and a subset of the edges of the graph.



# Questions

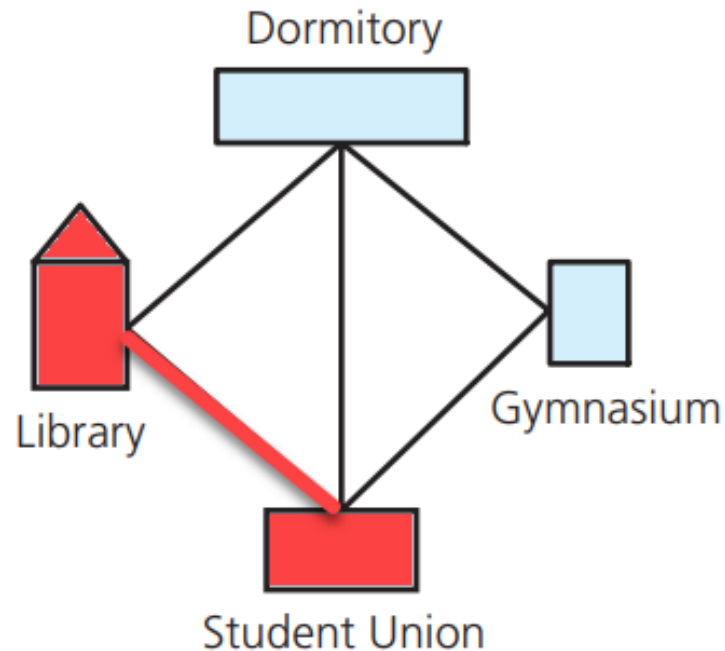
- Q1: What are the degrees of the vertices of a star?
- Q2: Let  $G$  be the graph represented in figure and  $X$  be the set  $\{a, b, f, e, g, l\}$ . Draw a figure of  $G[X]$ .



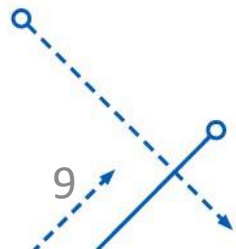


# Adjacent

- Two vertices of a graph are adjacent if connected by an edge.

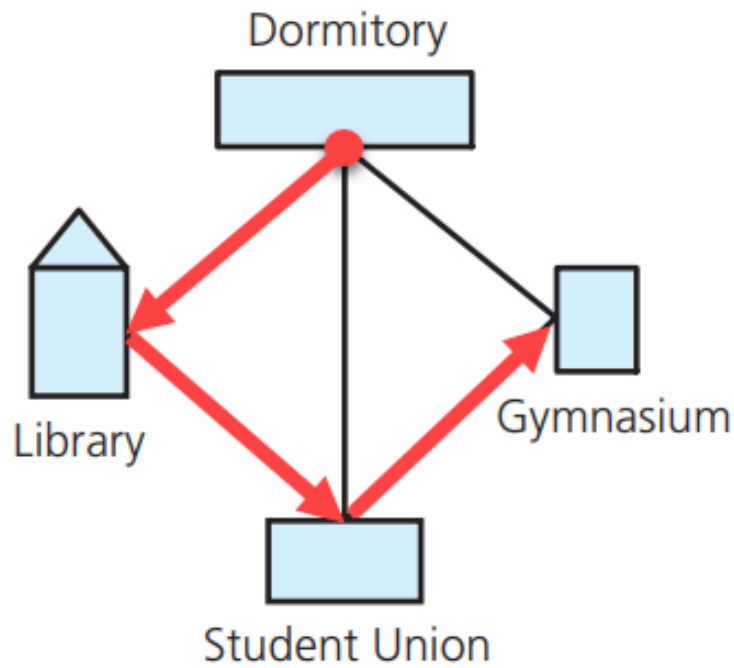


The Library và the Student Union là liền kề.

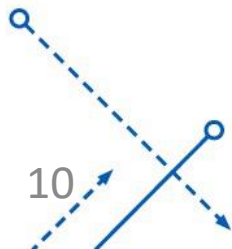


# Path

- A **path** between two vertices is a sequence of edges that begins at one vertex and ends at another.

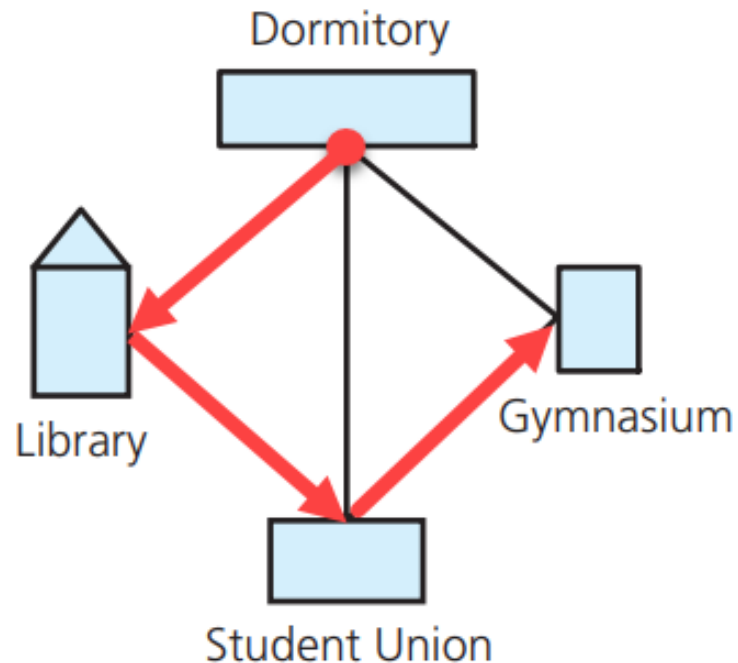


There is a route that starts from Dormitory, through the Library, to the Student Union, and finally to the Gymnasium,



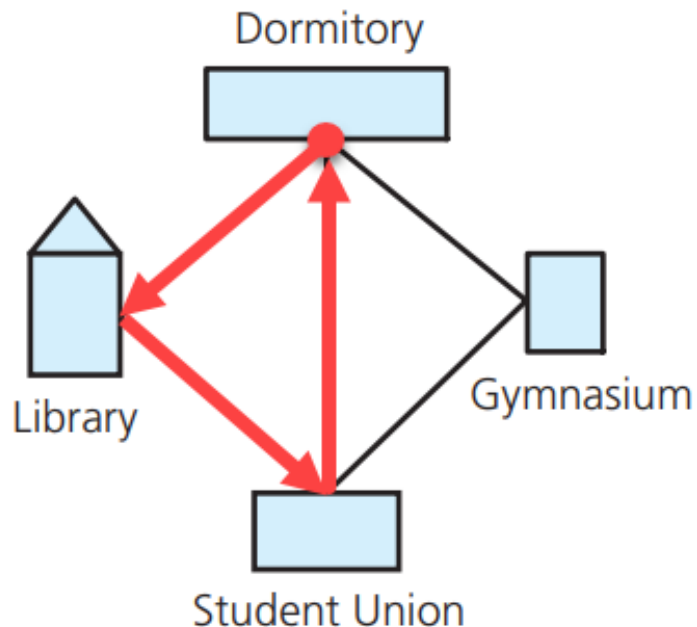
# Simple Path

- A **simple path** is a path that does not pass through the same vertex more than once.

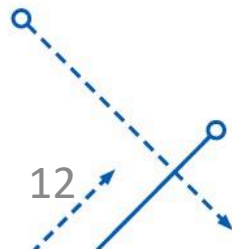


# Cycle

- A **cycle** is a path that begins and ends at the same vertex; A single cycle is a cycle that does not pass through the other vertices more than once.



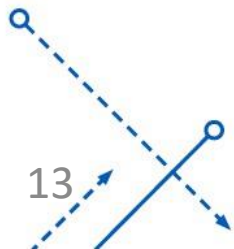
Library–Student Union–  
Gymnasium–Dormitory–Library  
is a single cycle



# Questions

---

- Draw a figure of a path of length 0, of a path of length 1 and of a path of length 2.
- Draw a figure of a circuit of length 3 and of a circuit of length 4.
- Why does the definition of a circuit require “ $n \geq 3$ ”?



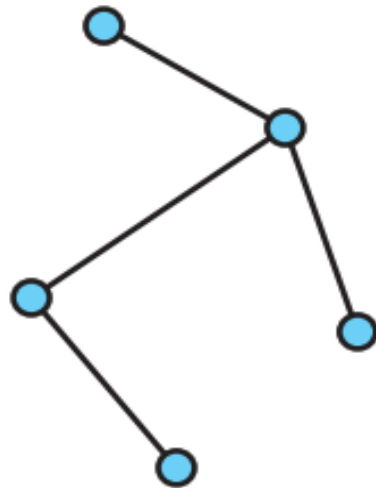
# Review questions

---

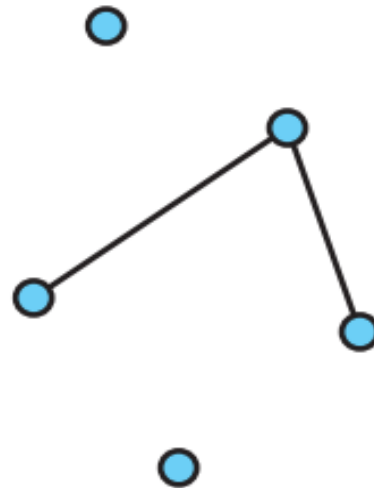
- Question 1: When is a graph called connected?
- Question 2: What is a complete graph?
- Question 3: Is a complete graph necessarily connected?
- Question 4: Can a graph have duplicate edges between vertices?
- Question 6: Is a multigraph a graph?
- Question 7: Can the edges of a graph start and end at the same vertex?
- Question 8: Can you label the edges of the graph?

# Connected and Unconnected Graphs

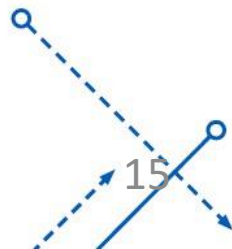
- A graph is **connected** if each pair of distinct vertices has a path between them.
- That is, in a connected graph you can go from any vertex to any other vertex by following a path.



Connected Graph

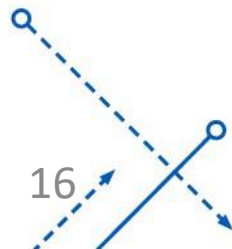
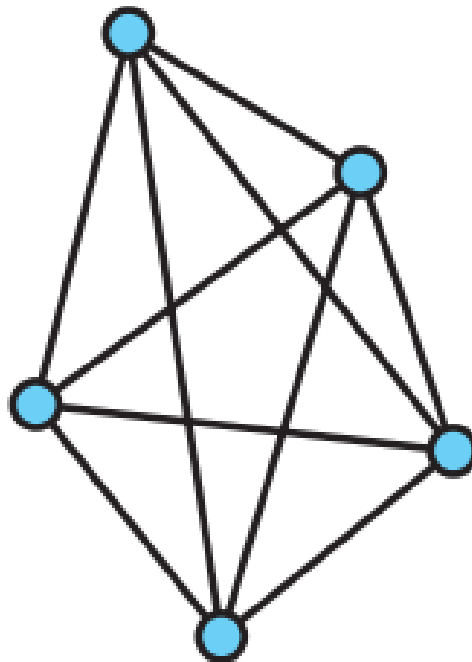


Disconnected Graph



# Complete graph

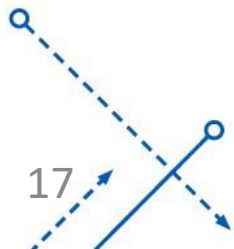
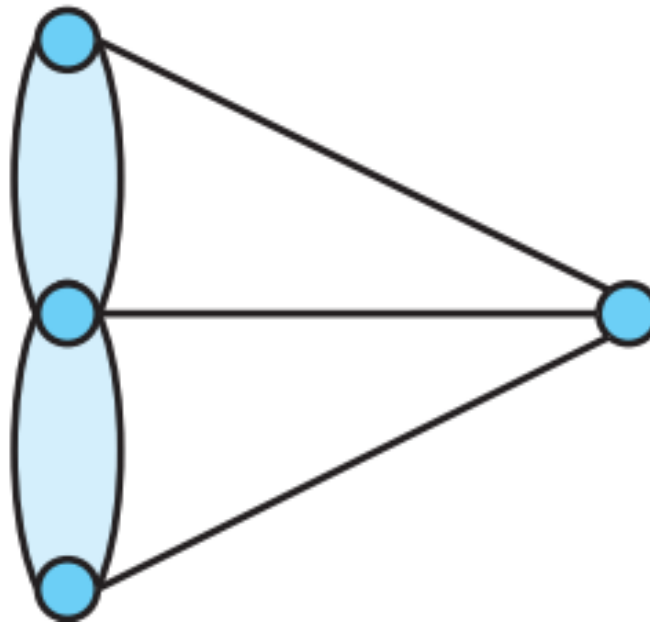
- In a **complete graph**, each pair of distinct vertices has an edge between them.
- A complete graph is also connected, but a connected graph is not necessarily complete.





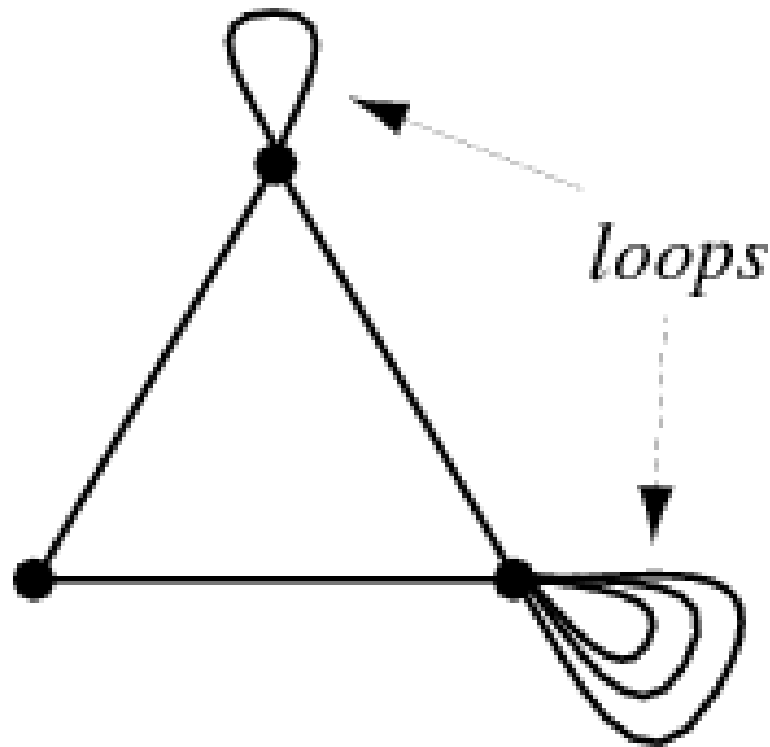
# Multigraph

- Because a graph has a set of edges, a graph cannot have duplicate edges between vertices.
- However, a **multigraph** allows multiple edges.
  - Therefore, a multigraph is not a graph. The edges of a graph can't start and end at the same vertex.



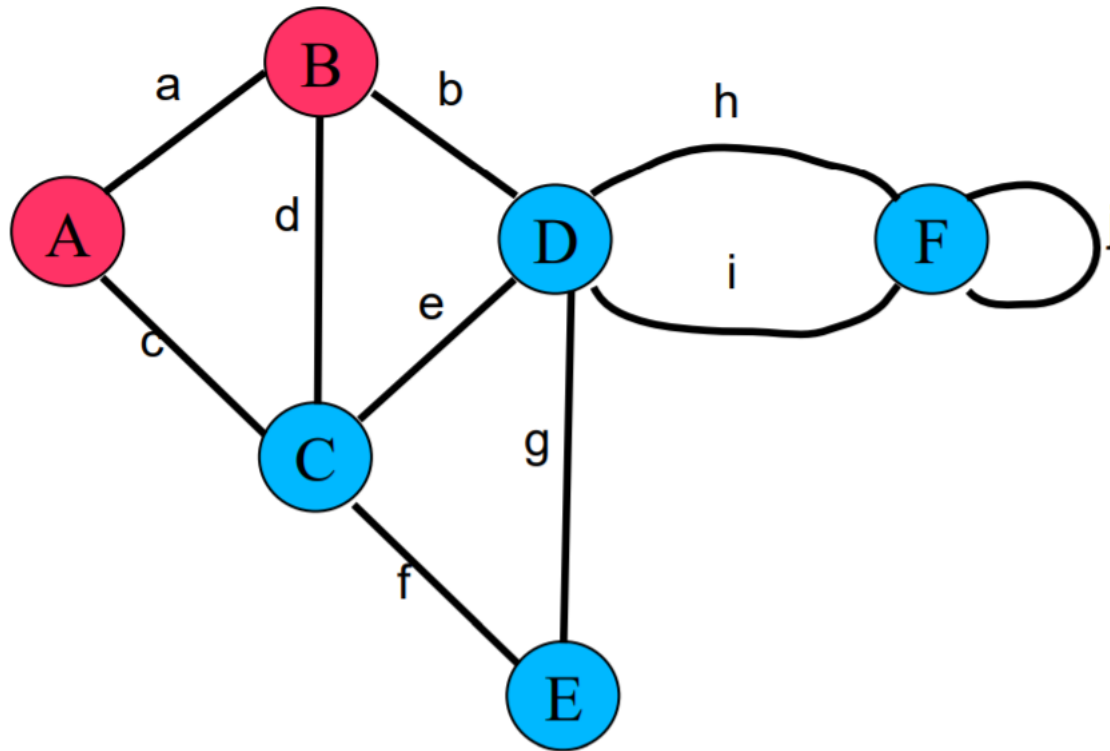
# Self edge

- The edges of a graph that start and end at the same vertex are called **self edges** or loops.



# Label

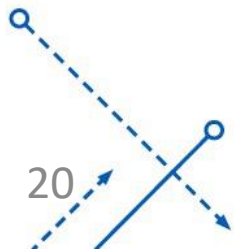
- You can **label** the edges of the graph.



# Content

---

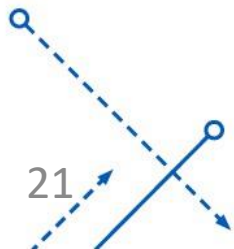
- Graph concepts and properties
- **Types of graphs**
- Graph storage
- Some basic algorithms on graphs
  - Graph Traversal Algorithm
  - Topological Sorting Algorithm
  - Spanning Tree Algorithm
  - Shortest Path Algorithm
- Slices and Threads
- Graph Match



# Review questions

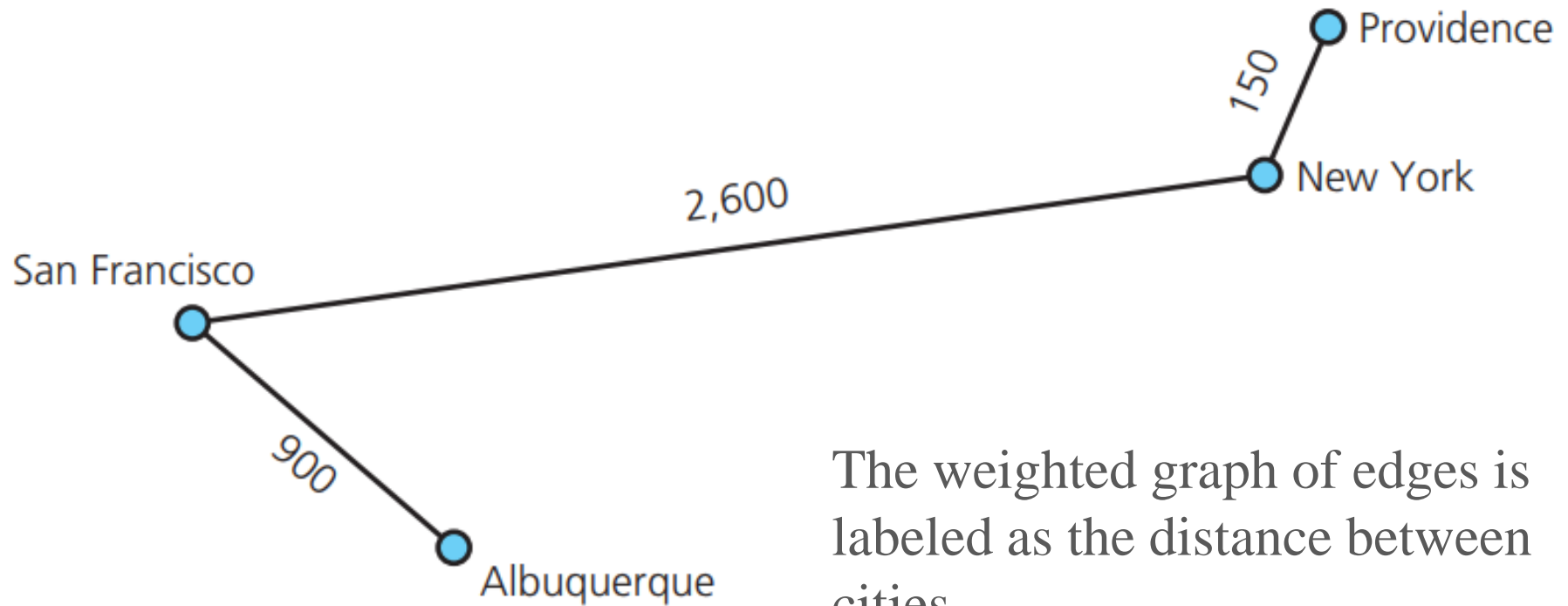
---

- Q1: What is a weighted graph?
- Q2: What is the difference between an undirected graph and a directed graph?
- Q3: What is a bipartite graph?
- Q4: What is a regular graph?
- Q5: Are all complete graphs regular graphs?
- Q6: What characterizes a small-world graph?
- Q7: What is a stochastic graph?



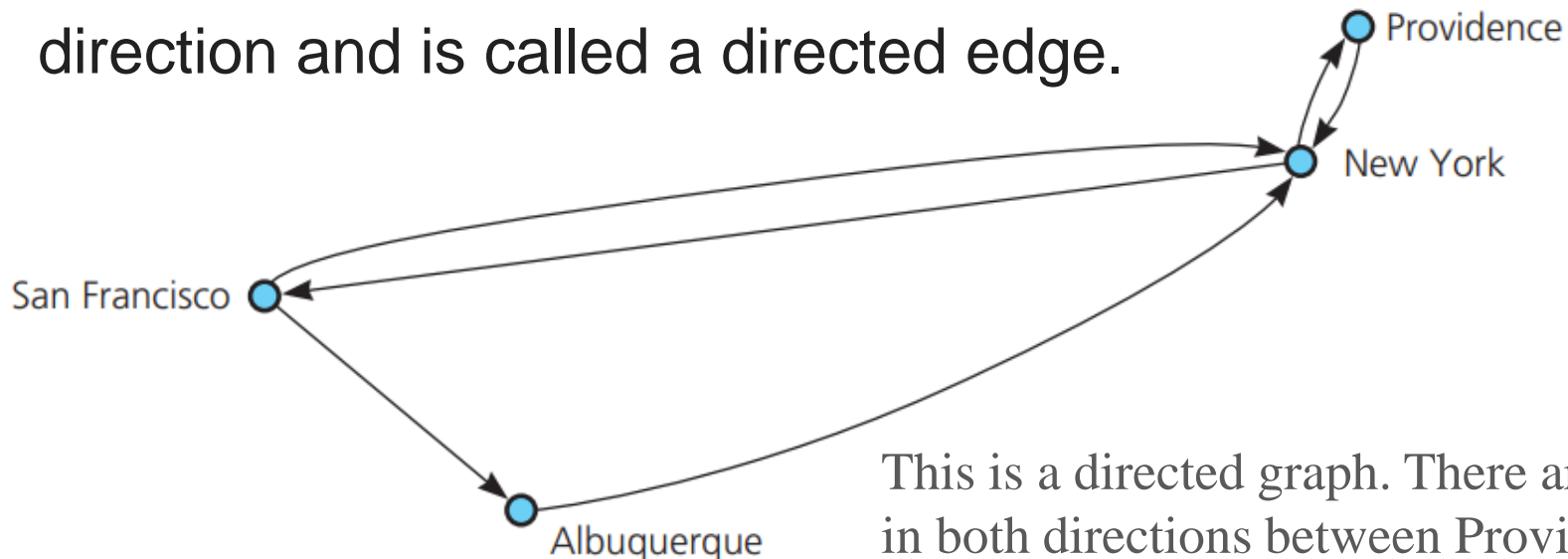
# Weighted graph

- When the labels represent numeric values, the graph is called a **weighted graph**.



# Directed and Undirected Graphs

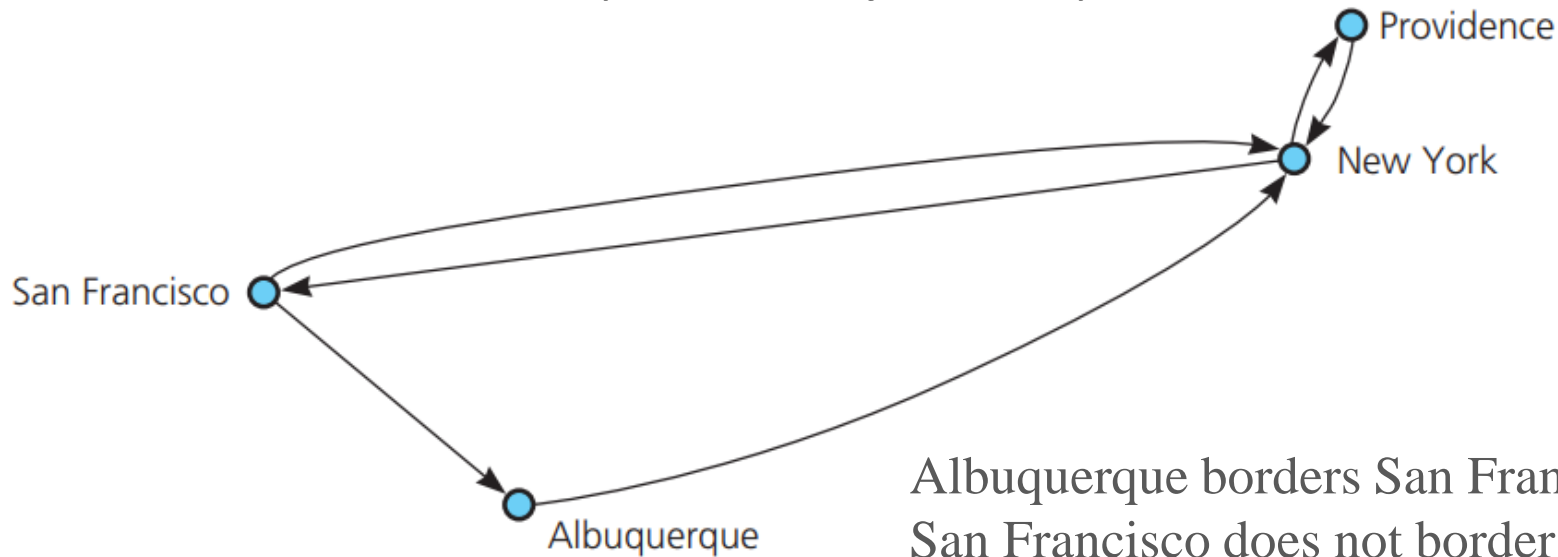
- In an **undirected graph**, edges have no direction. That is, we can move in either direction along the edges between vertices in an undirected graph.
- Conversely, each edge in a **directed graph** will have a direction and is called a directed edge.



This is a directed graph. There are flights in both directions between Providence and New York, but although there are flights from San Francisco to Albuquerque, there are no flights from Albuquerque to San Francisco.

# Directed and Undirected Graphs

- The definition of adjacent vertices is not exactly the same in both types of graphs.
- In the digraph:
  - If there is an edge directed from vertex  $x$  to vertex  $y$ , then  $y$  is adjacent to  $x$ . (In other words,  $y$  is a descendant of  $x$  and  $x$  is an ancestor of  $y$ .)And it's not necessary that  $x$  is adjacent to  $y$ .



Albuquerque borders San Francisco, but San Francisco does not border Albuquerque.



# Exercises

---

Which of the following statements is/are TRUE for undirected graphs?

P: Number of odd degree vertices is even.

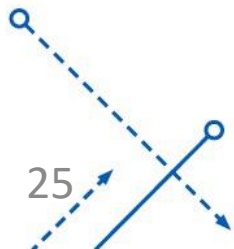
Q: Sum of degrees of all vertices is even.

(A) Neither P nor Q

(B) Both P and Q

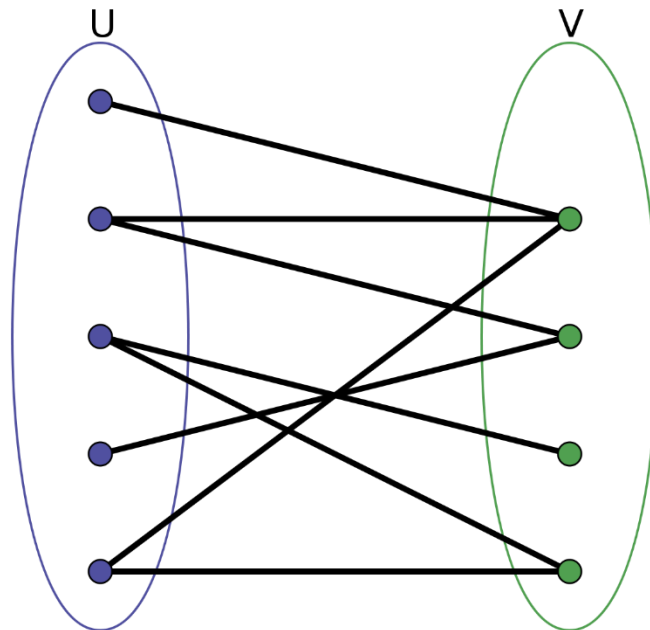
(C) Q only

(D) P only



# Bipartite graph

- A **bipartite graph** is a graph in which vertices can be divided into two sets that do not communicate with each other, and all edges in the graph connect only vertices on different sets.

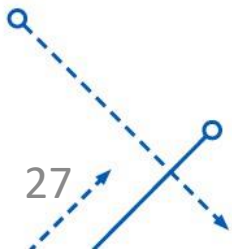


# Exercises

---

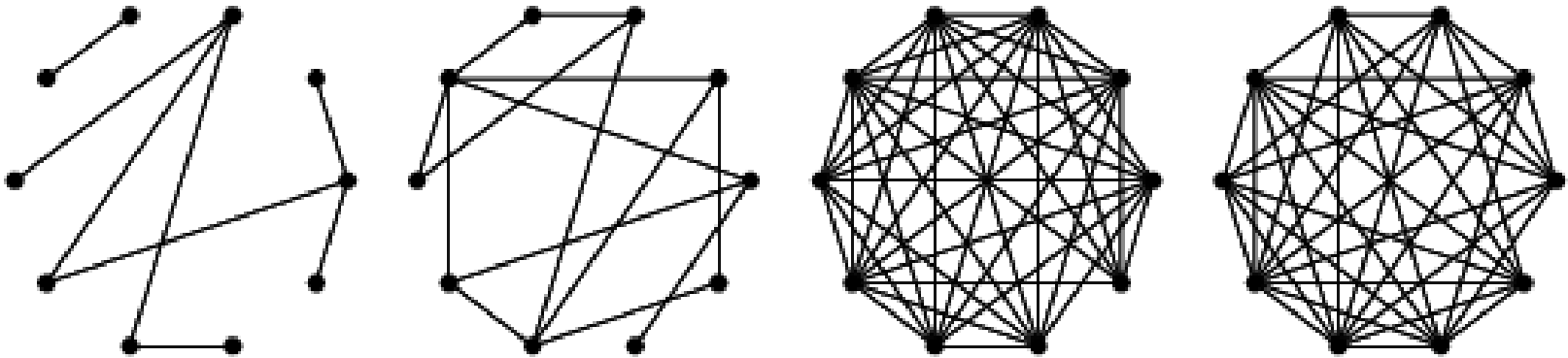
The maximum number of edges in a bipartite graph on 12 vertices is \_\_\_\_\_.

- A) 12
- B) 24
- C) 36
- D) 48



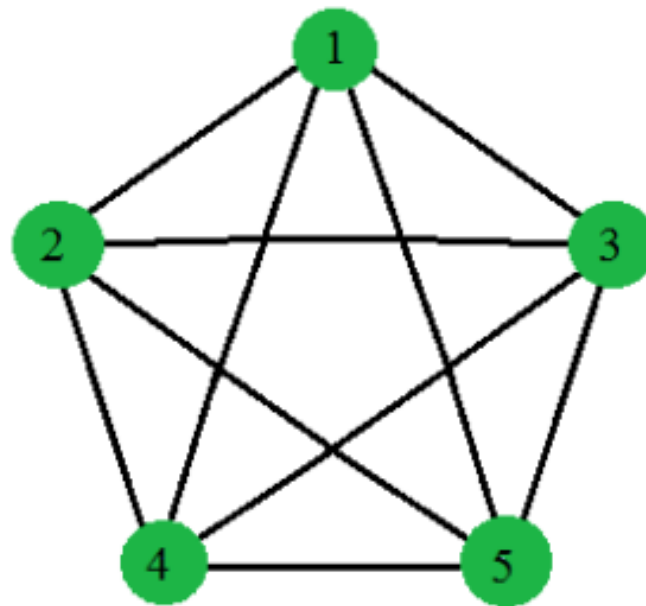
# Random graph

- **Random graph** is the graph where the edges are randomly generated.



# Regular graph

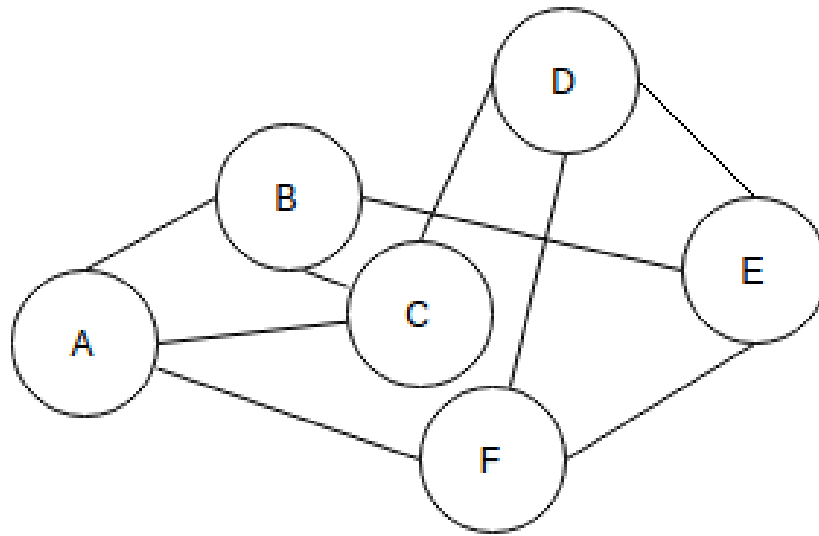
- A **regular graph** is a graph where all vertices have the same degree.
- From there, it can be seen that all complete graphs are regular graphs



4 Regular

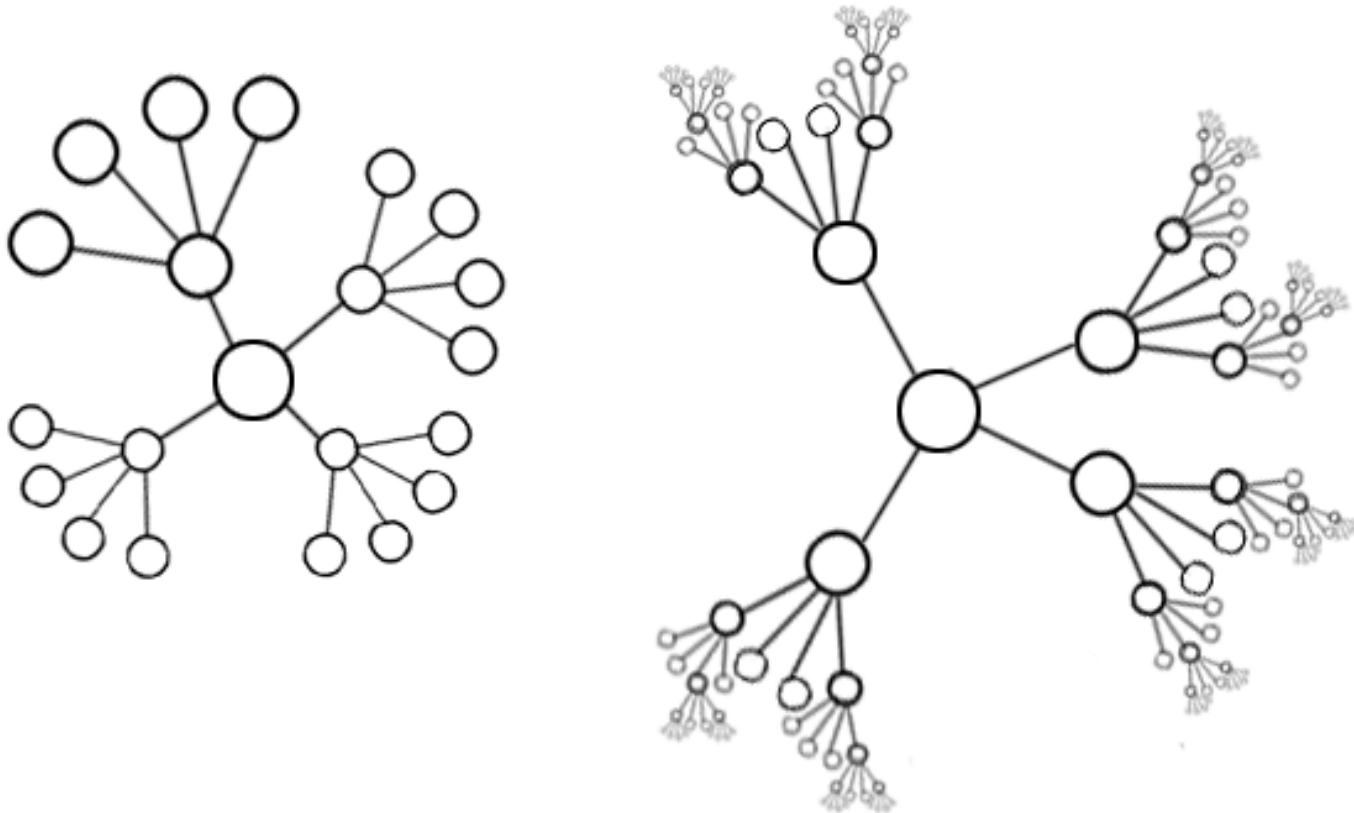
# Exercises

- The given Graph is regular?



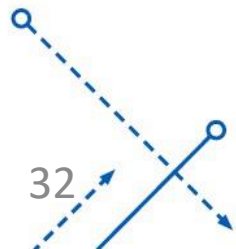
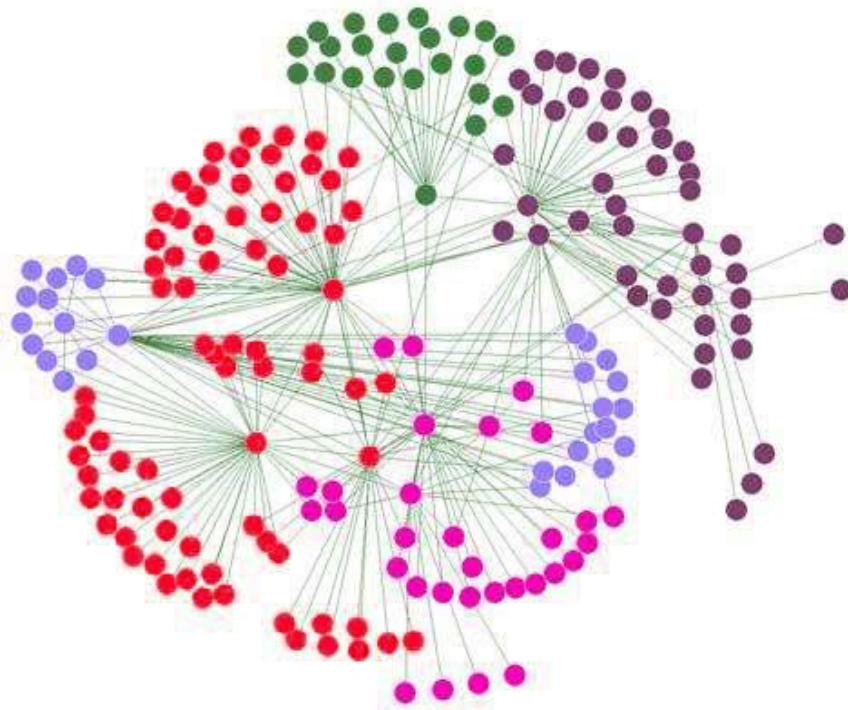
# Scale-free graph

- A graph is said to be **scale-free** if its features are independent of the size of the graph (the number of vertices). That is, as the graph expands, the underlying structure remains the same.



# Small-world graph

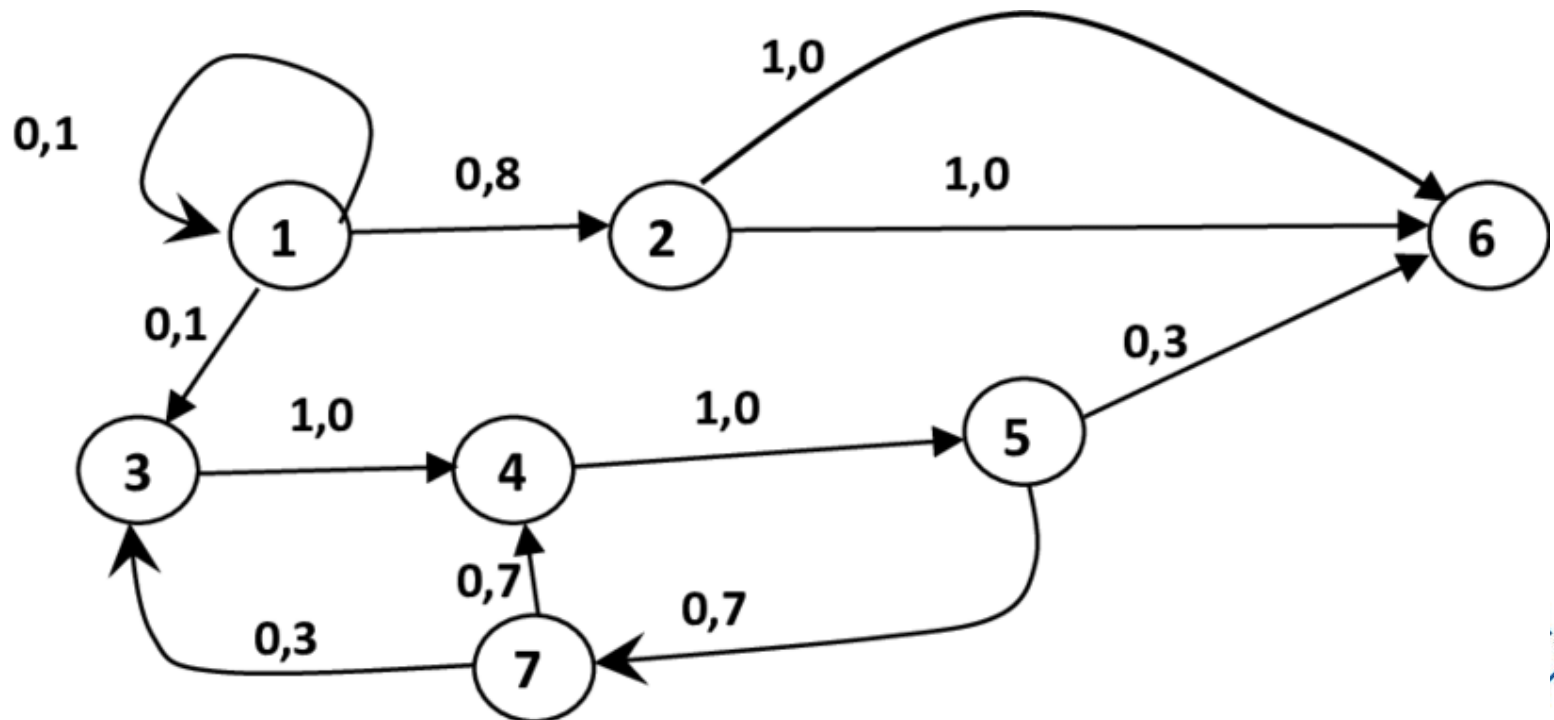
- **Small-world** is a graph where the average path between all vertices is small. That is, a vertex can reach any vertex through only a few edges.





# Stochastic graph

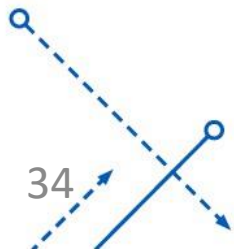
- A **stochastic graph** is a graph where the weights of each edge are represented with a probability value representing the probability of moving between two determinists. The sum of the weights of the edges at each vertex is 1.



# Contents

---

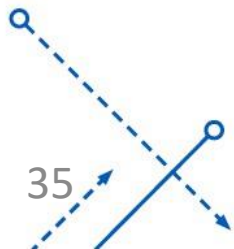
- Graph concepts and properties
- Types of graphs
- **Graph storage**
- Some basic algorithms on graphs
  - Graph Traversal Algorithm
  - Topological Sorting Algorithm
  - Spanning Tree Algorithm
  - Shortest Path Algorithm
- Slices and Threads
- Graph Match



# Review questions

---

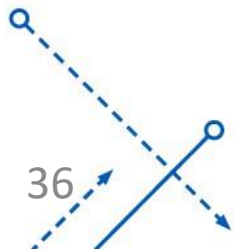
- Q1: What are the two most common implementations of graphs in computer systems?
- Q2: How does an adjacency matrix represent the existence of edges between vertices?
- Q3: What changes in an adjacency matrix when the graph is weighted?
- Q4: How does an adjacency list represent edges in a graph?
- Q5: Which graph implementation is better, adjacency matrices, or adjacency lists?
- Q6: How do space requirements differ between adjacency matrices and adjacency lists?



# Graph storage

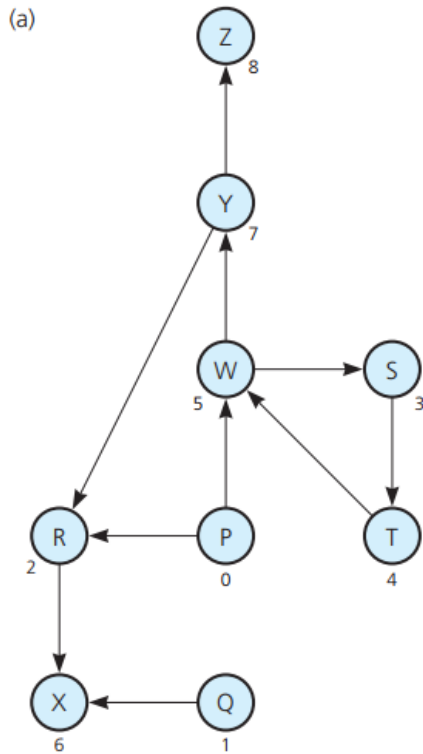
---

- There are different ways to store graphs in computer systems.
- The data structure used depends on both the graph structure and the algorithm used to manipulate the graph.
- The two most common implementations of graphs are adjacency matrices and adjacency lists.



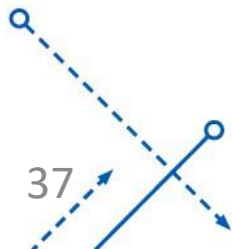
# Adjacent Matrix

- The **adjacent matrix** for a graph with  $n$  vertices numbered  $0, 1, \dots, n - 1$  is:
  - An array matrix  $n \times n$  such that the matrix  $[i][j]$  is 1 if there is an edge from vertex  $i$  to vertex  $j$  and 0 if there is none.



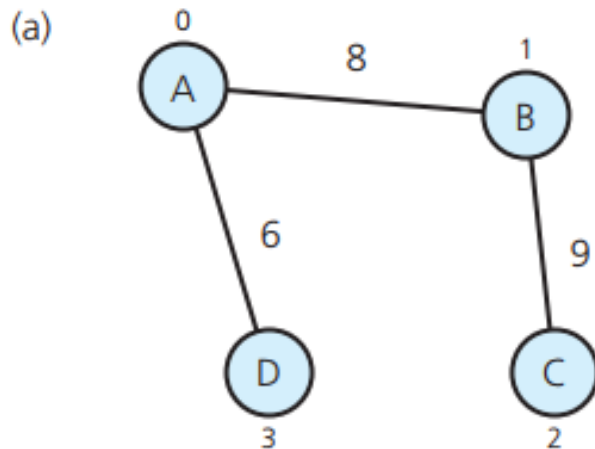
(b)

		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0



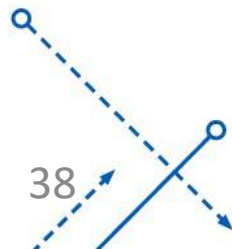
# Adjacent matrix

- When the graph is weighted:
  - Let the matrix  $[i][j]$  be the side-labeled weighting from vertex  $i$  to vertex  $j$ , instead of simply 1
  - Let the matrix  $[i][j]$  be equal to  $\infty$  instead of 0 when there are no sides from vertex  $i$  to vertex  $j$ .



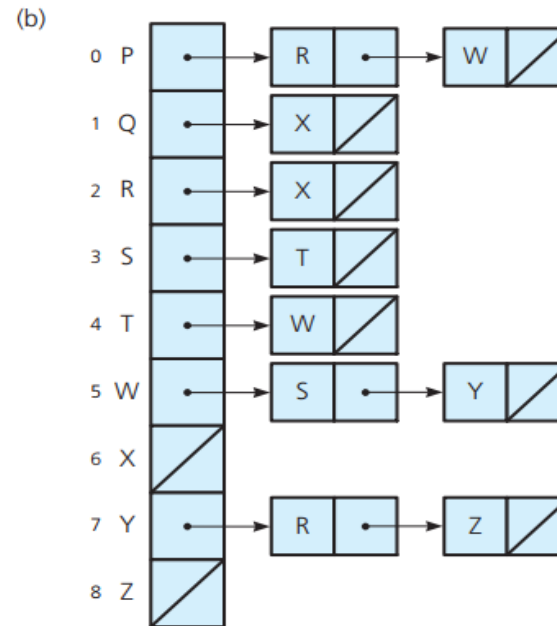
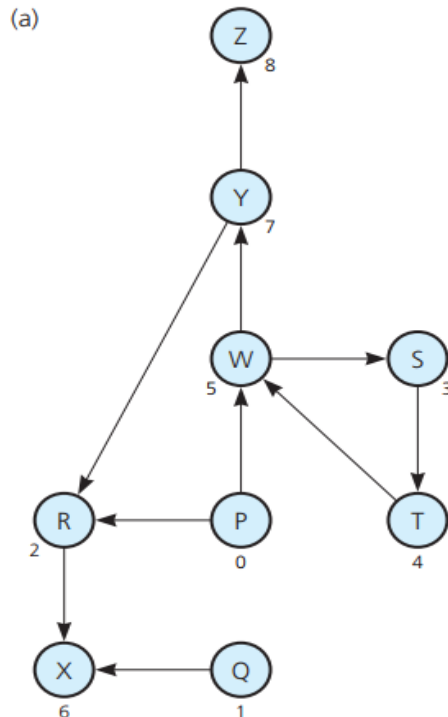
(b)

		0	1	2	3
		A	B	C	D
0	A	$\infty$	8	$\infty$	6
1	B	8	$\infty$	9	$\infty$
2	C	$\infty$	9	$\infty$	$\infty$
3	D	6	$\infty$	$\infty$	$\infty$



# Adjacent lists

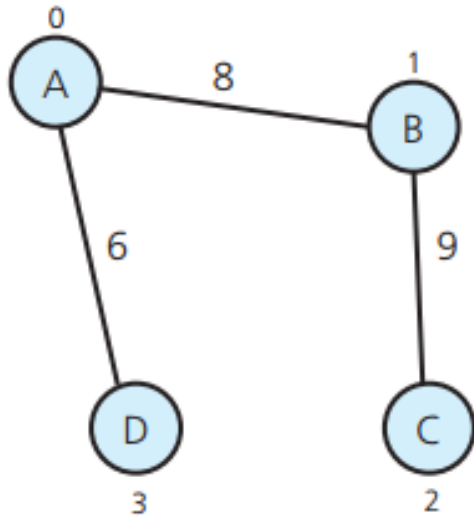
- The **adjacent list** for a graph with  $n$  vertices numbered  $0, 1, \dots, n - 1$  consists of  $n$  association sequences.
  - The  $i$ -link series has a node for vertex  $j$  if and only if the graph contains an edge from vertex  $i$  to vertex  $j$ . This node may contain the value of vertex  $j$ , if any.



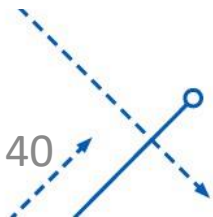
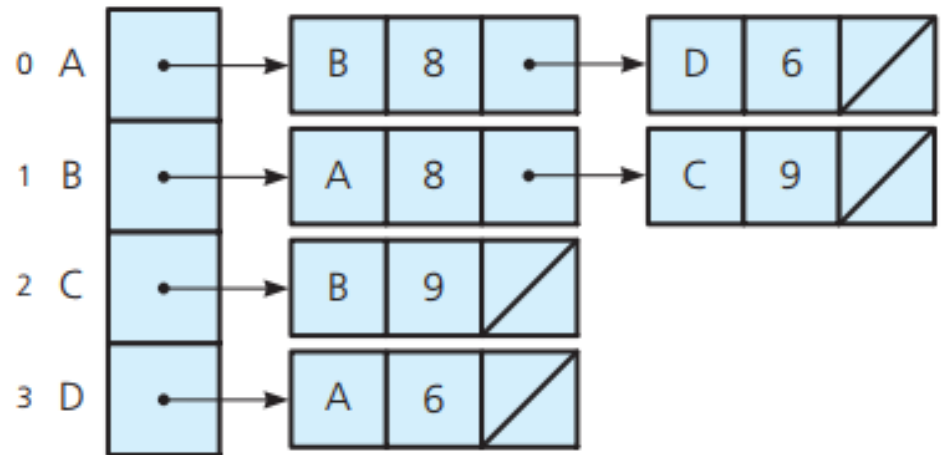
# Adjacent lists

The adjacent list for a scalar graph treats each edge as if it were two edges with opposite directions.

(a)



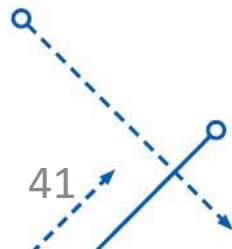
(b)





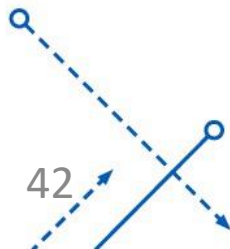
# Which way is better?

- Which of these two graph implementations—adjacent matrices or adjacent lists—is better?
  - The answer depends on how your particular application uses graphs.
- Example:
  - Determine if there is an edge from vertex  $i$  to vertex  $j$  -> adjacent matrix
  - Find all vertices adjacent to a given  $i$ -vertex -> the adjacent list



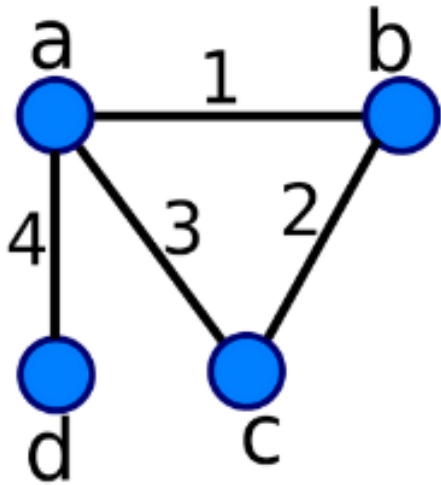
# Which way is better?

- Consider the space requirements of the two deployments.
  - On the surface, adjacent matrices require less memory than adjacent lists, because each entry in the matrix is simply an integer, while each list node contains both values to define vertices and pointers.
  - However, the adjacent matrix always has  $n^2$  entry, while the number of nodes in the adjacent list is equal to the number of edges in a directed graph or double that number for a scalar graph. Although the adjacent list also has  $n$  leading pointers, it usually requires less storage than the adjacent matrix

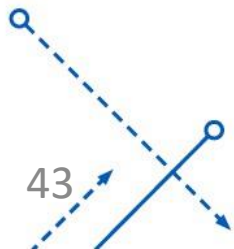
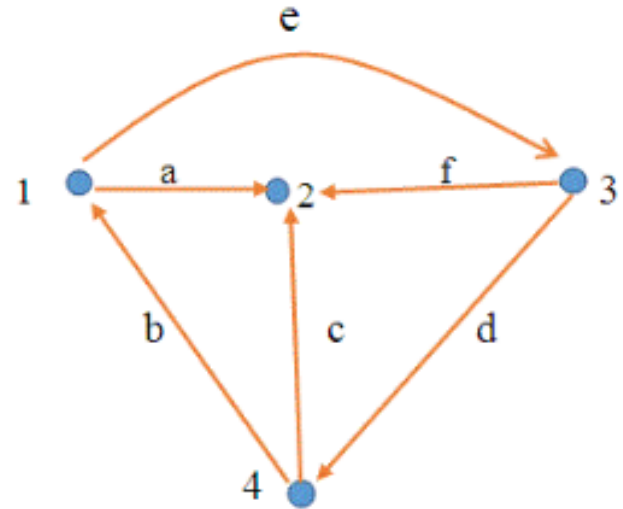


# Question

- Do you know incidence matrix?



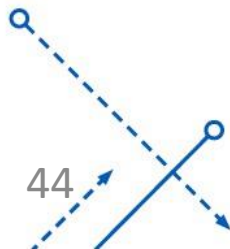
	1	2	3	4
a	1	0	1	1
b	1	1	0	0
c	0	1	1	0
d	0	0	0	1



# Contents

---

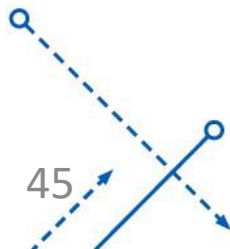
- Graph concepts and properties
- Types of graphs
- Graph storage
- Some basic algorithms on graphs
  - Graph Traversal Algorithm
  - Topological Sorting Algorithm
  - Spanning Tree Algorithm
  - Shortest Path Algorithm
- Slices and Threads
- Graph Match



# Review questions

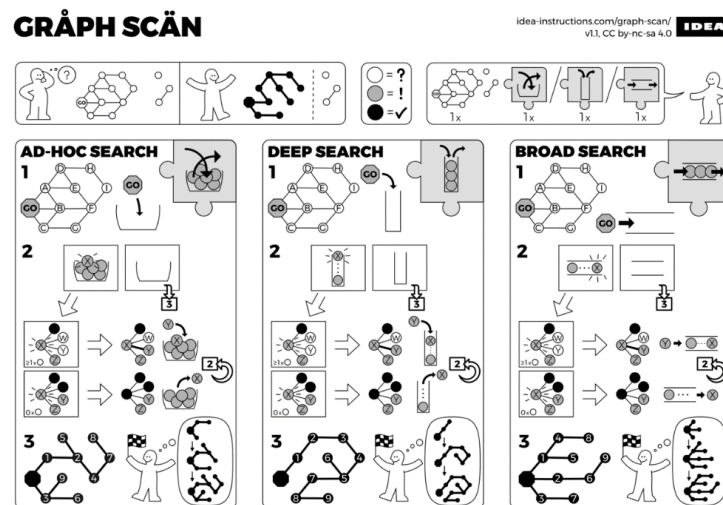
---

- Q1: How can you determine the connected components of a graph?
- Q2: Describe the difference between Depth-First Search (DFS) and Breadth-First Search (BFS) in graph traversal.
- Q3: Define topological sorting in a directed acyclic graph.
- Q4: Which algorithm is used to find the shortest paths between a given origin point and all other vertices in a graph?
- Q5: Describe the graph coloring problem and common variants.



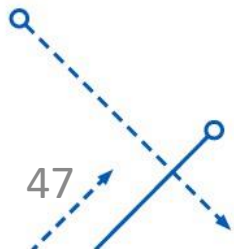
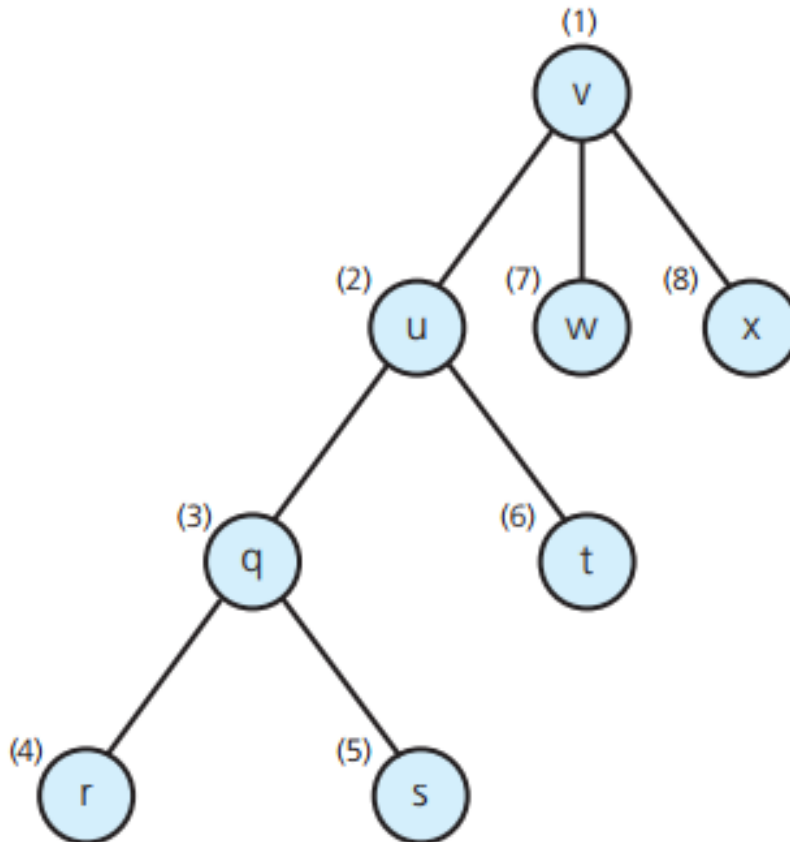
# Graph Traversal

- A graph traversal starting at vertex  $v$  will visit all vertices  $w$  that have a path between  $v$  and  $w$ .
  - If a graph is not connected, a path traversing the graph starting at vertex  $v$  will only access a subset of the vertices of the graph. This subset is called the connection component containing  $v$ .
  - If a graph contains a cycle, then a graph traversal algorithm can iterate infinitely. To avoid that unfortunate scenario, the algorithm must mark each vertex in one hit and never visit a vertex more than once.



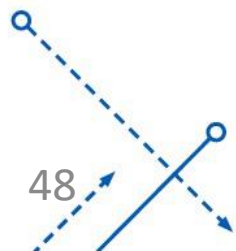
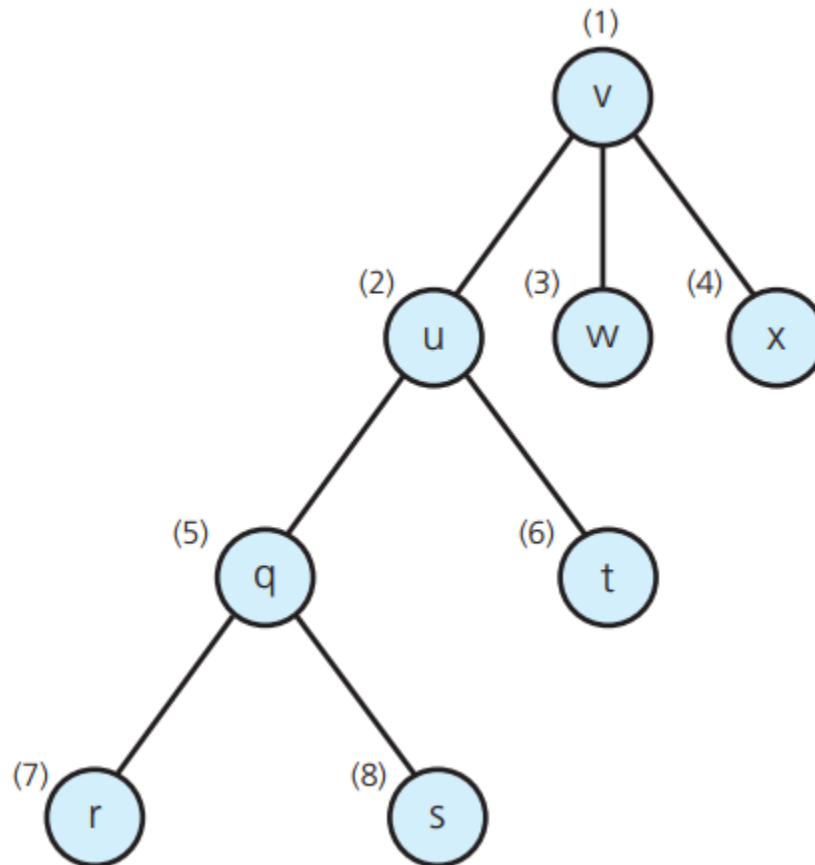
# Depth-First Search

- From a given vertex  $v$ , the depth-first search (DFS) strategy of the graph traversal method will proceed along the path from  $v$  as far into the graph as possible before backing up.



# Breadth-first search

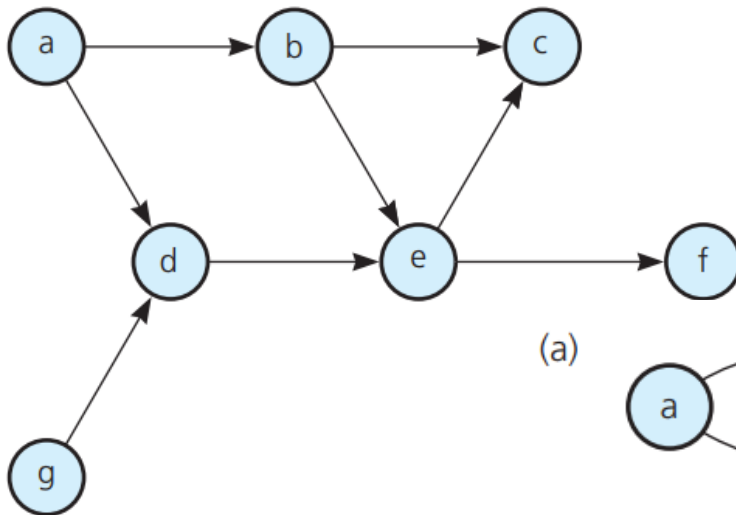
- **Breadth-first search** (BFS): the graph traversal strategy visits every vertex adjacent to  $v$  that it can before accessing any other vertices.



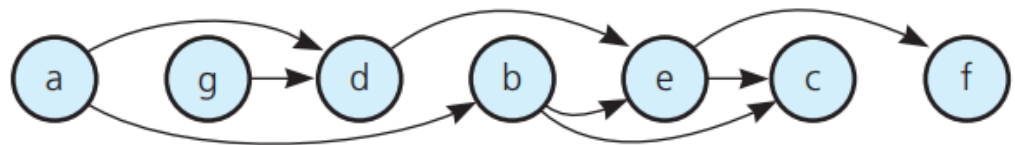


# Topological sort

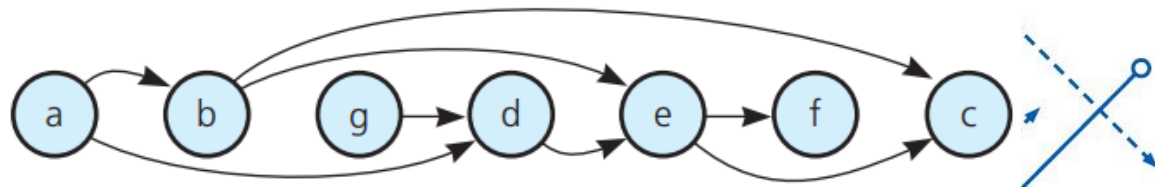
- A directed graph with no cycles has a natural order. It is a linear order, called a topological order.
- The arrangement of vertices into a topological order is called topological sorting.



(a)

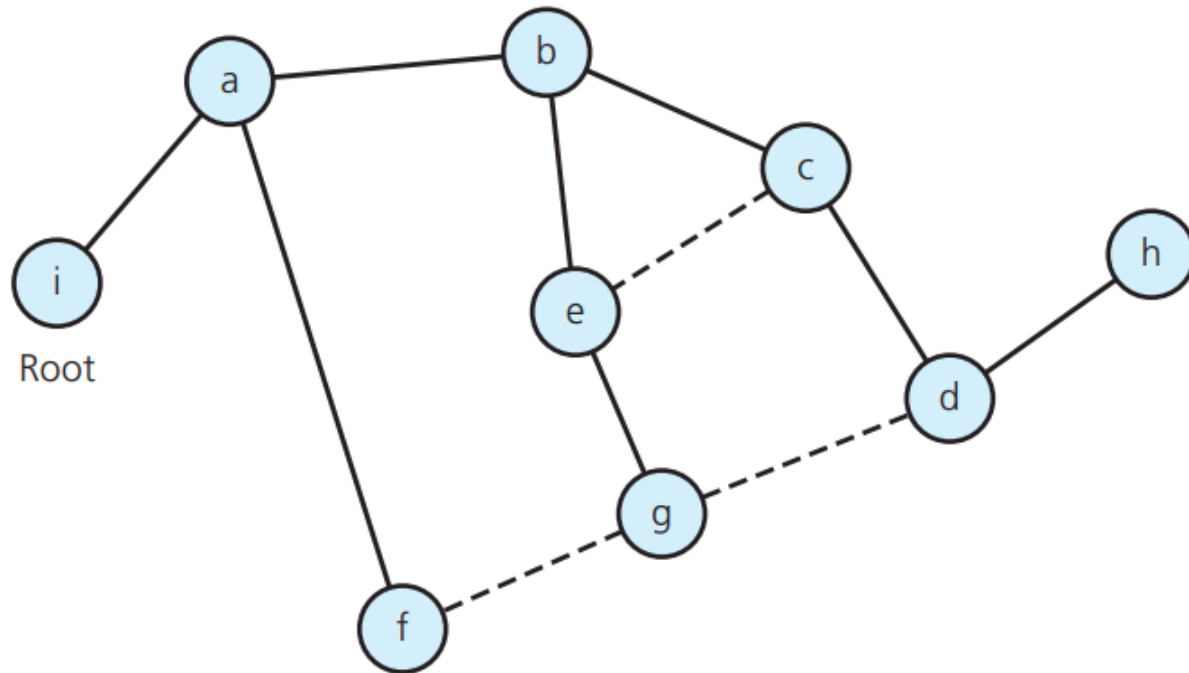


(b)



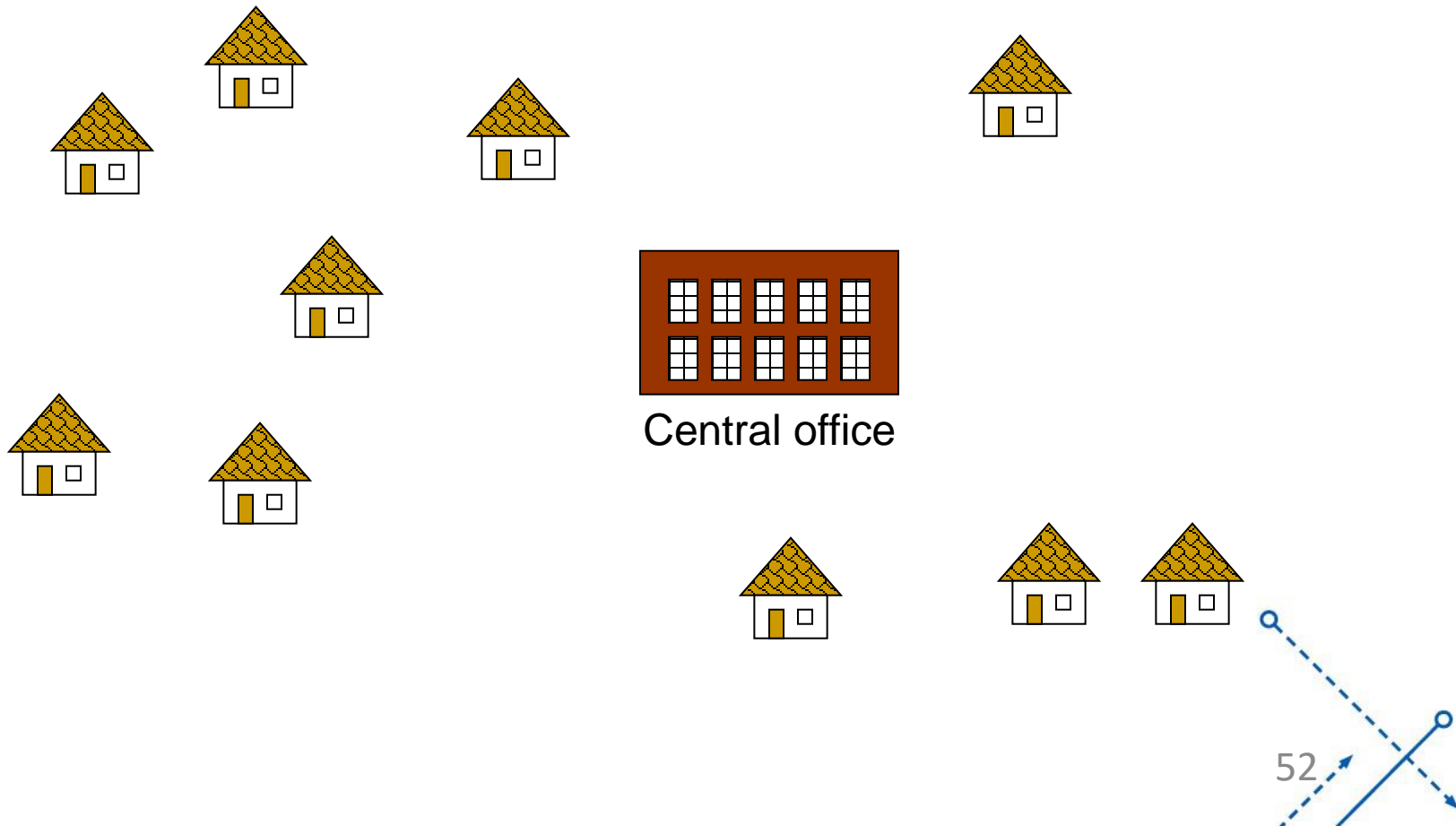
# Spanning tree

- The **spanning tree** of a connected undirected graph  $G$  is a subgraph of  $G$  containing all of  $G$ 's vertices and enough of its edges to form a tree.



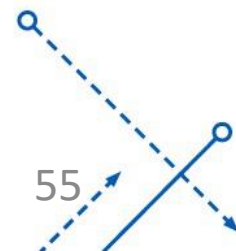
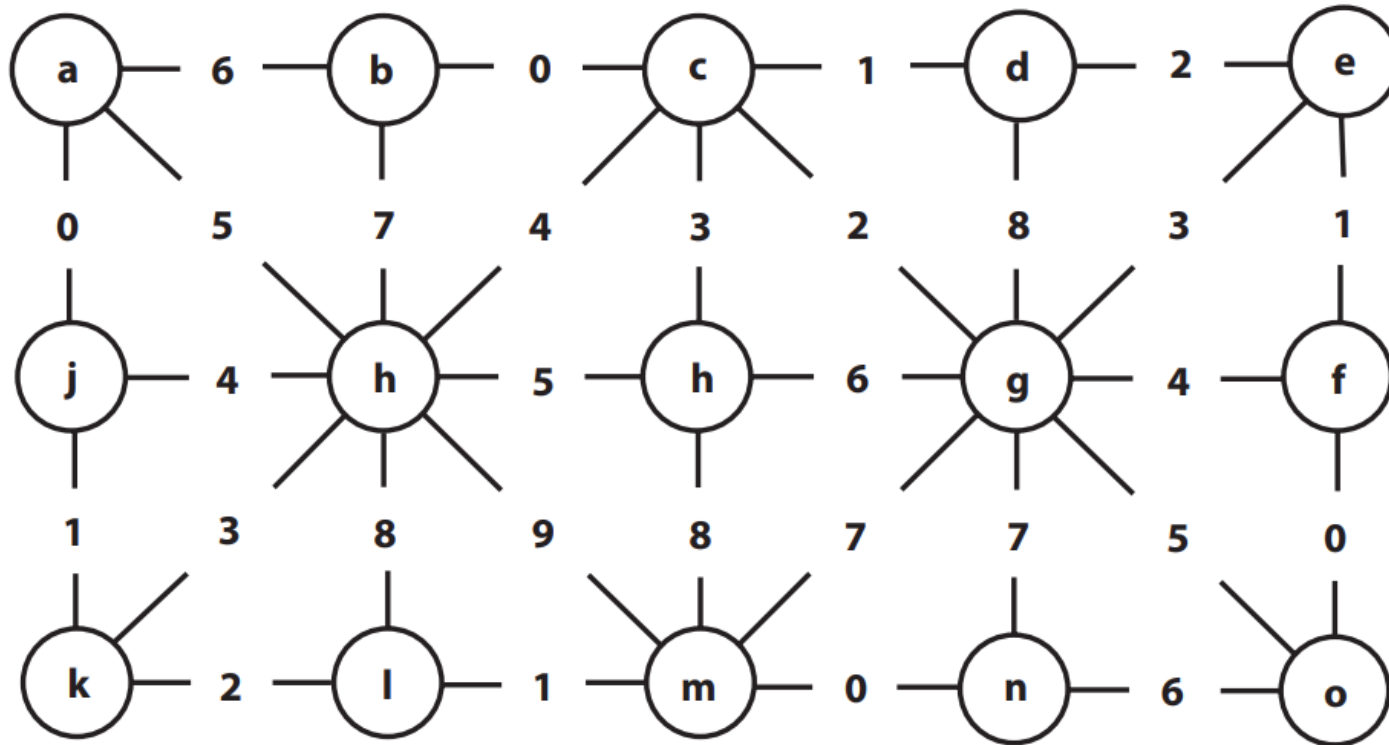
# Minimum spanning tree

- Imagine that a developing country hires you to design their phone system so that all cities in the country can call each other.



# Exercises

- Draw the minimum spanning tree of



# The shortest path

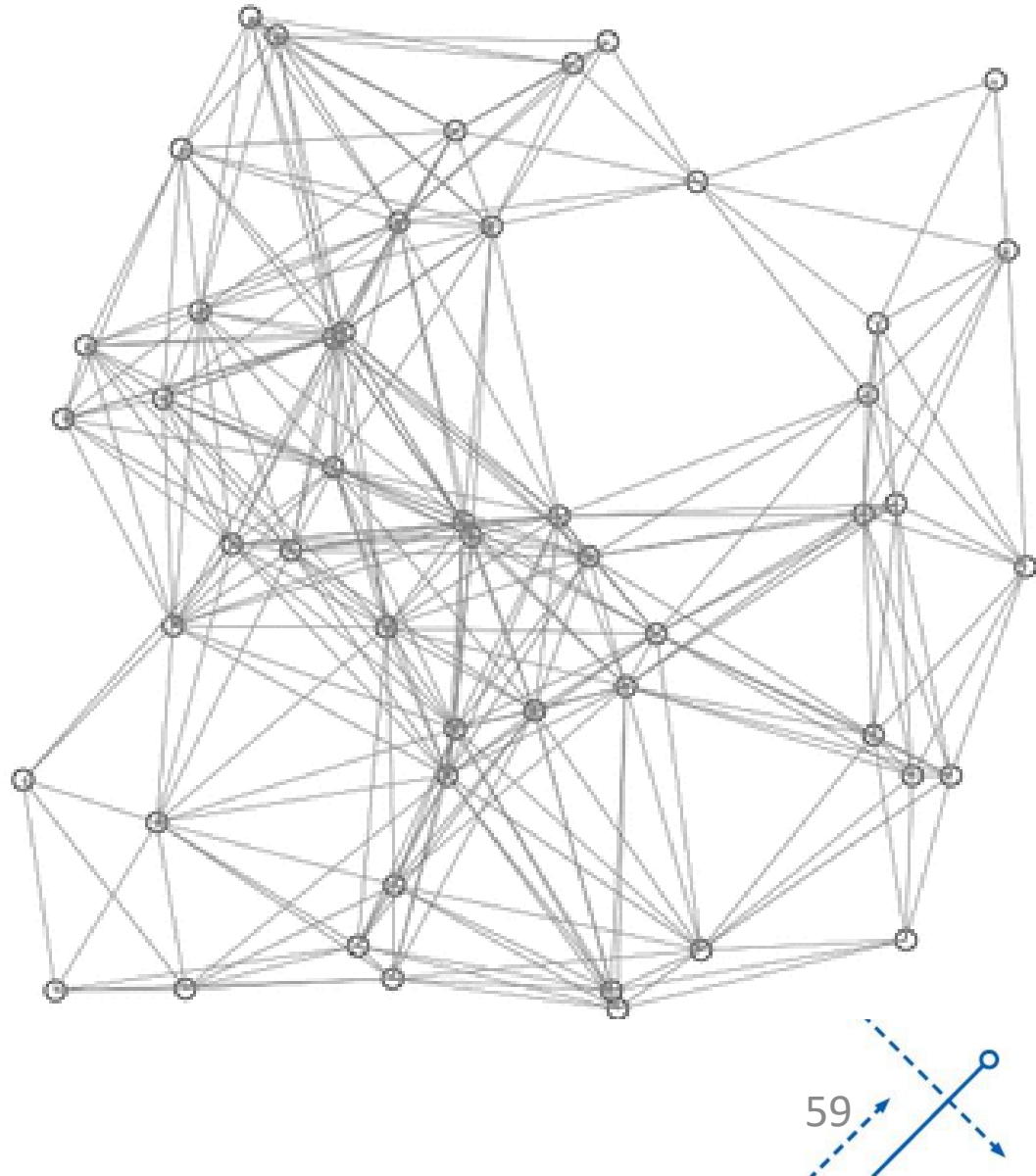
- Find the shortest path between two specific peaks in the city map.



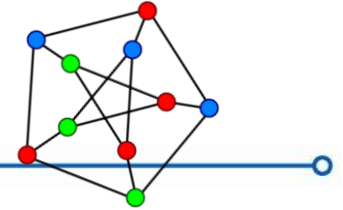
Small section of Williamsburg in Brooklyn, NY

# Dijkstra algorithm

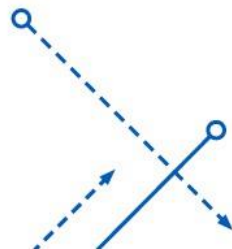
- The Dijkstra algorithm determines the shortest paths between a given origin point and all other vertices.
- The algorithm uses a set of vertices. The set of selected vertices and an array weight, where the weight  $[v]$  is the weight of the shortest path from vertex 0 to vertex  $v$ .



# Graph coloring

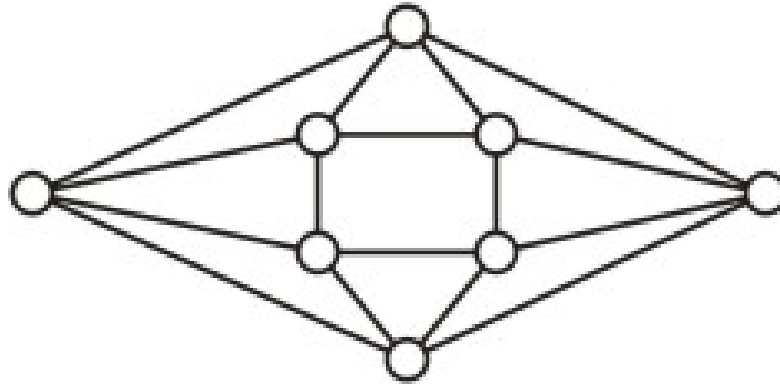


- Graph coloring is an assignment of "colors" to the elements of a graph that satisfies given constraints.
- Some common coloring problems are:
  - Vertex coloring, where adjacent vertices must not share the same color.
  - Edge coloring, where adjacent edges must not share the same color.
  - Face coloring of a planar graph, where each face or region is colored such that no two faces sharing a boundary have the same color.



# Exercises

- The minimum number of colours required to colour the following graph, such that no two adjacent vertices are assigned the same colour, is



- A) 2
- B) 3
- C) 4
- D) 5



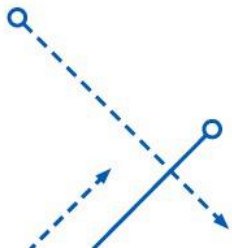
# Exercises

---

There are 6 football teams: A, B, C, D, E, F competing in a round-robin (one round) tournament. The following matches have already taken place:

- Team A has played against teams B and E.
- Team B has played against teams A and F.
- Team C has played against teams D and F.

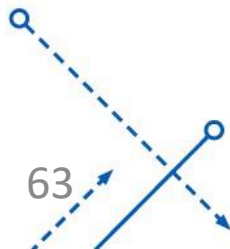
Each team can only play one match per week. Arrange the matches into weeks so that the total number of weeks is minimized.



# Content

---

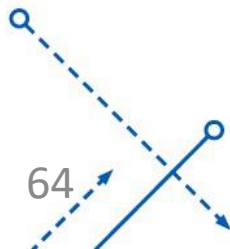
- Graph concepts and properties
- Types of graphs
- Graph storage
- Some basic algorithms on graphs
  - Graph Traversal Algorithm
  - Topological Sorting Algorithm
  - Spanning Tree Algorithm
  - Shortest Path Algorithm
- Slices and Threads
- Graph Match



# Review questions

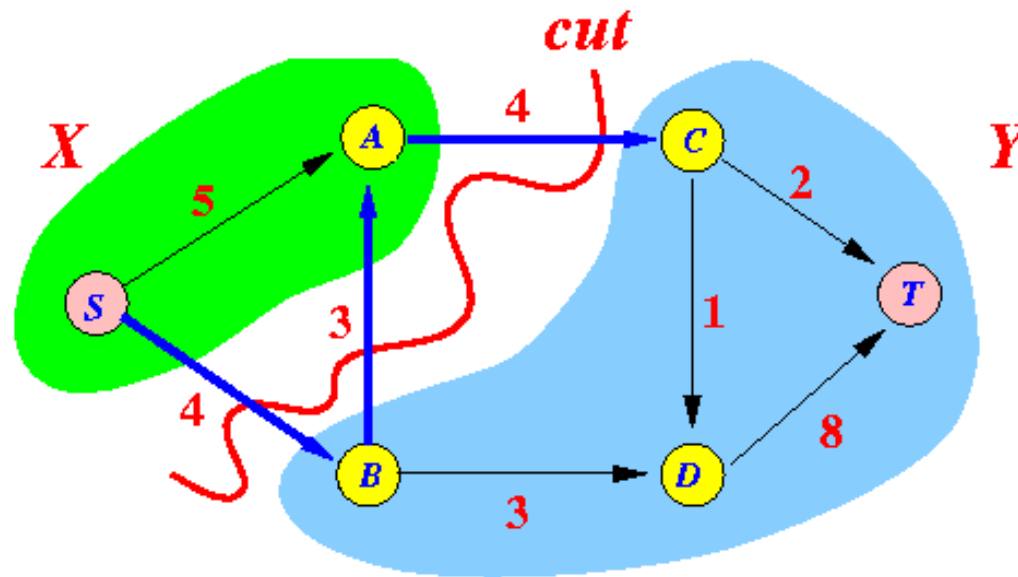
---

- Q1: What is a cut in a graph?
- Q2: What is a cut-set in a graph?
- Q3: How is the size/weight of a smallest/largest slice defined?
- Q4: What does flow represent in a graph?
- Q5: Define maximum flow in a graph.
- Q6: What is the relationship between maximum flow and minimum slice between two vertices  $s$  and  $t$ ?



# Cut

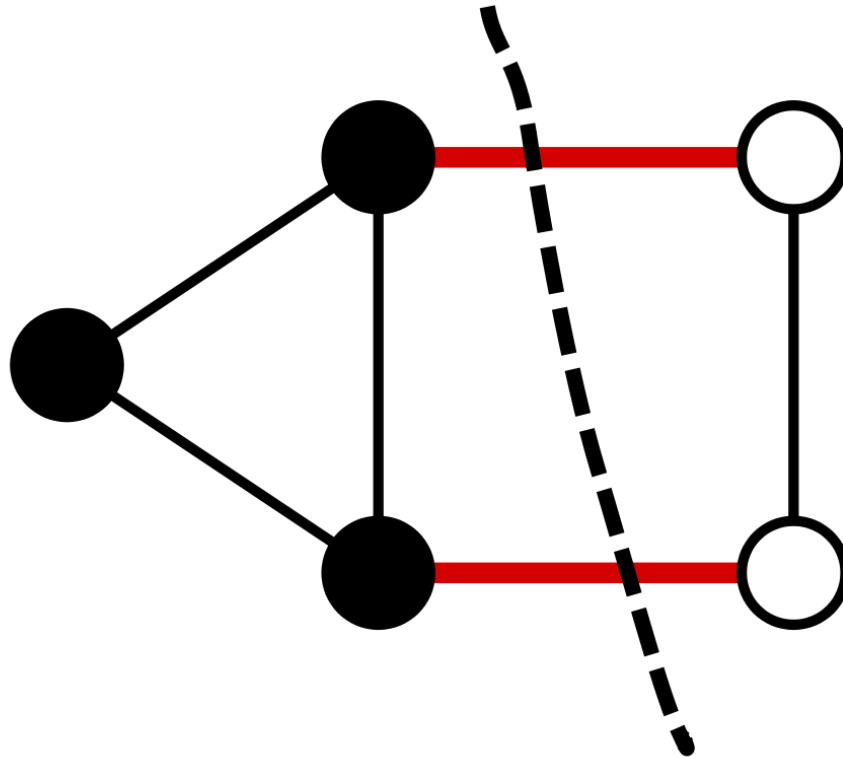
- A **cut** is a way of dividing the set of vertices of a graph into 2 subsets that do not intersect.
- A **cut-set** is a set of edges that each end is in a subset after cutting.



**Cut-set =  $\{ (S,B), (B,A), (A,C) \}$**

# Smallest slice

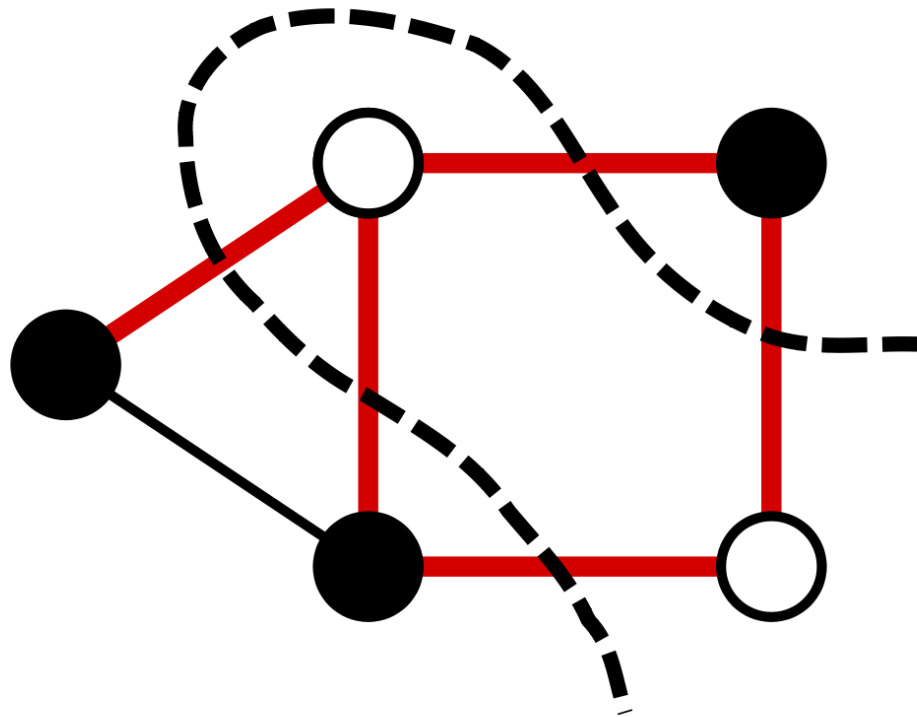
- A slice is smallest when its size/weight is not greater than any other slice.



The smallest size is 2

# The largest slice

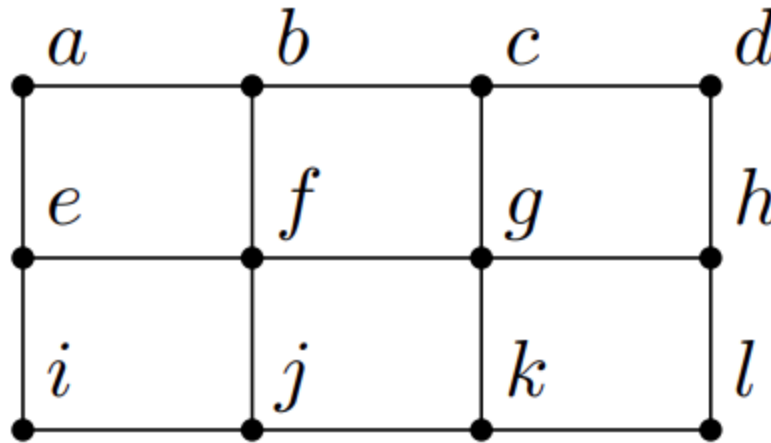
- A slice is largest when its size/weight is not less than any other slice.



The largest size is 4

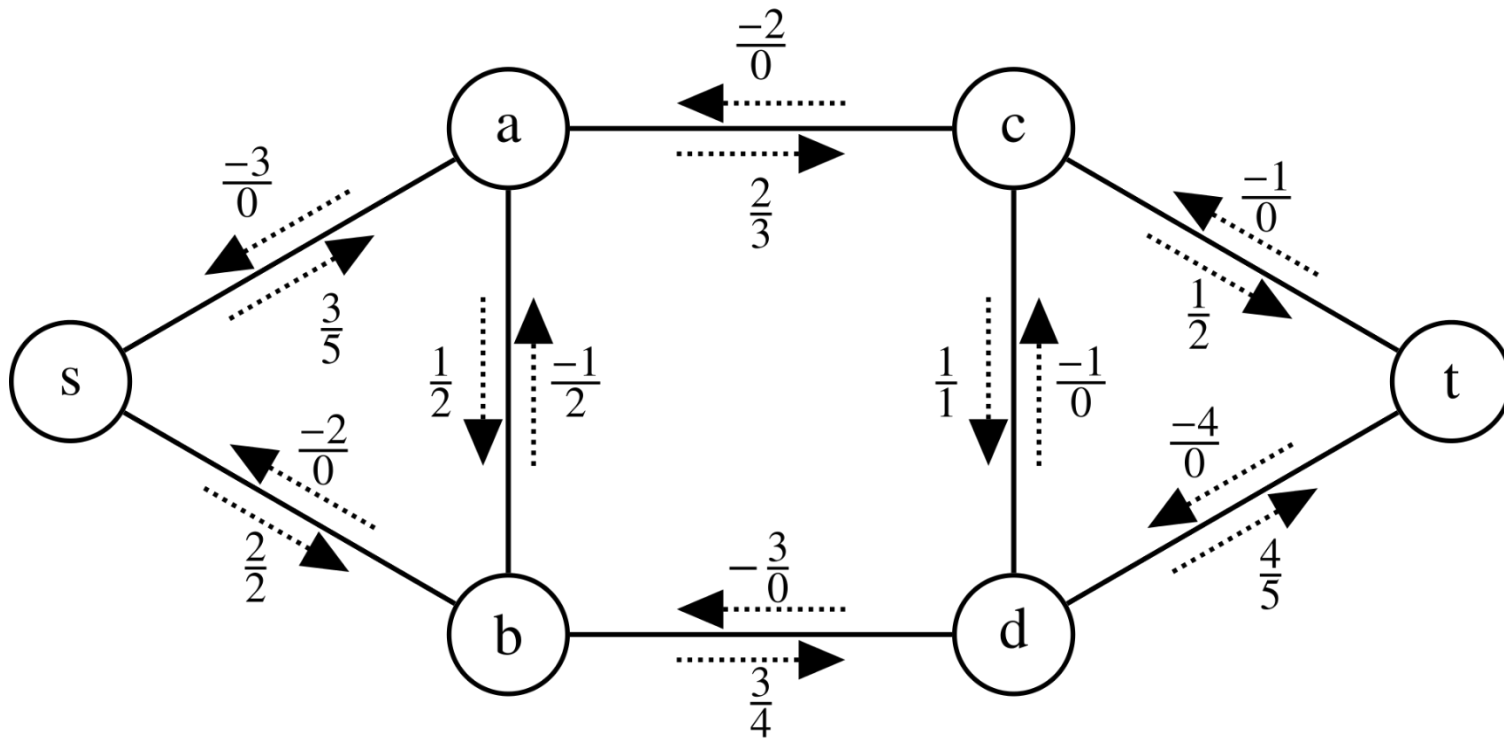
# Exercies

- Q1: Let  $G$  be the graph represented in figure. Is it true that the set  $\{ae, ef, fj, jk, cd, dh\}$  is a cut?



# Flow

- **Flow** represents the load capacity of an edge in a graph.

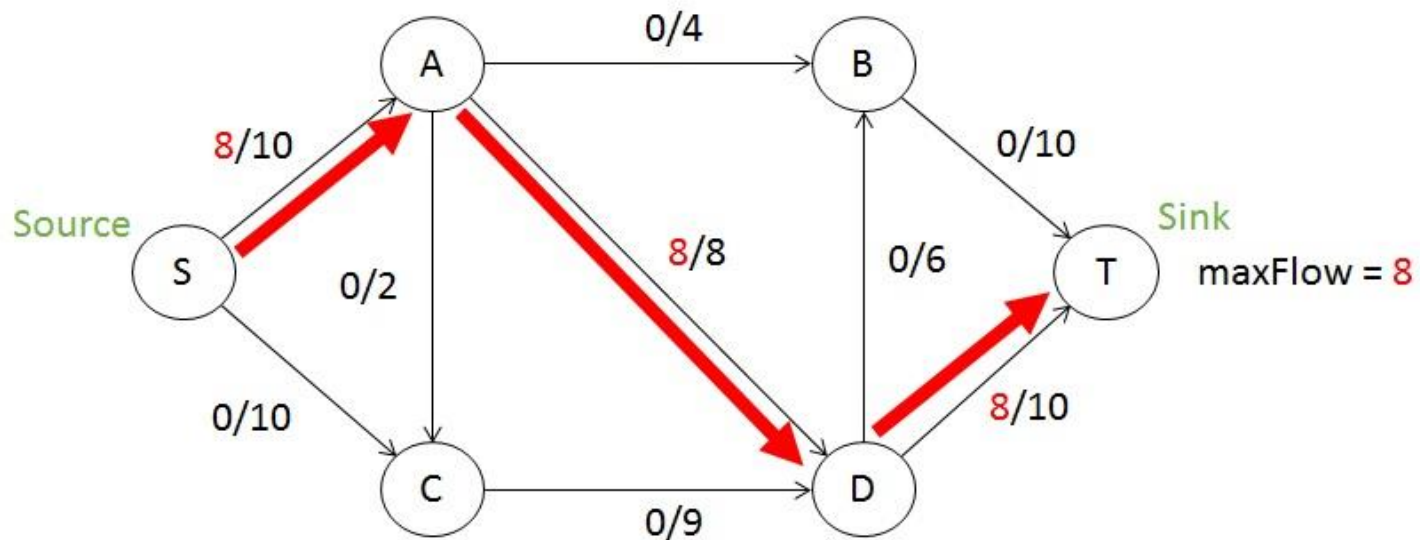


The flow from  $s$  to  $t$  with a total volume of 5.

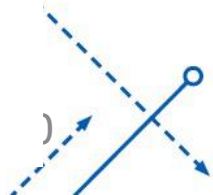


# Maximum flow

- The **maximum flow** is the flow that can achieve a maximum load transfer rate.
- The maximum value of a flow from s to t is equal to the minimum slice between s and t.



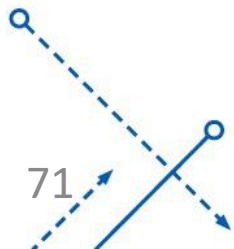
$$S \rightarrow A \rightarrow D \rightarrow T = 8$$



# Contents

---

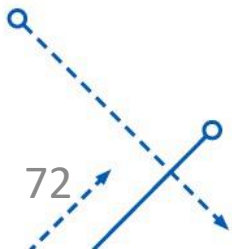
- Graph concepts and properties
- Types of graphs
- Graph storage
- Some basic algorithms on graphs
  - Graph Traversal Algorithm
  - Topological Sorting Algorithm
  - Spanning Tree Algorithm
  - Shortest Path Algorithm
- Slices and Threads
- Graph Match



# Review Questions

---

- Q1: Define graph isomorphism.
- Q2: What is graph edit distance?

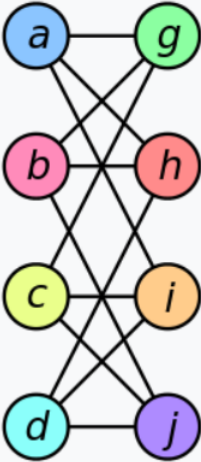
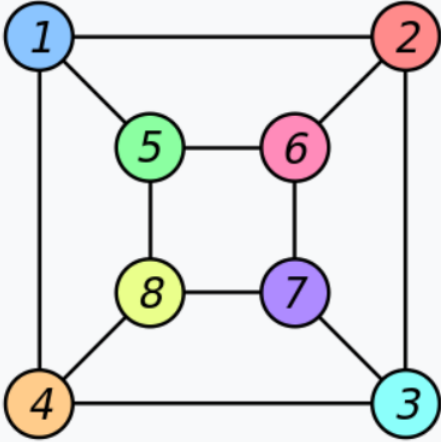


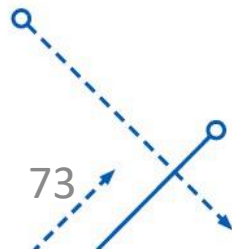
# Match graphs - isomorphic

- Two graphs G and H are called **isomorphic** if there exists a parallelism:

$$f: V(G) \rightarrow V(H)$$

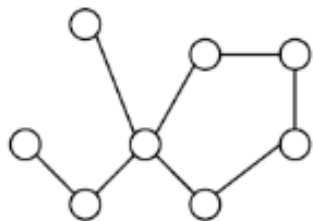
and if there is an edge between two vertices  $u$  and  $v$  in G, there will also be an edge between the vertices  $f(u)$  and  $f(v)$  in H.

Graph G	Graph H	An isomorphism between G and H
		$\begin{aligned} f(a) &= 1 \\ f(b) &= 6 \\ f(c) &= 8 \\ f(d) &= 3 \\ f(g) &= 5 \\ f(h) &= 2 \\ f(i) &= 4 \\ f(j) &= 7 \end{aligned}$

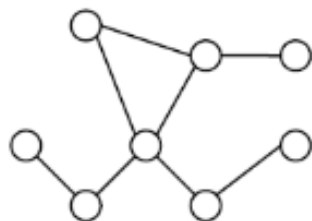


# Exercises

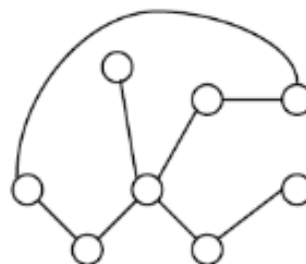
- Which of the following graphs is isomorphic to:



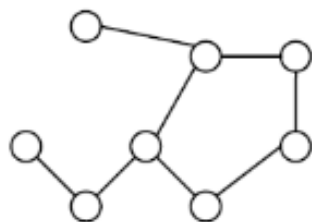
(A)



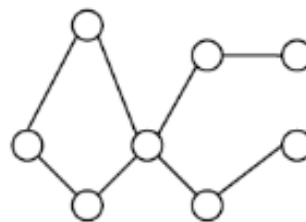
(B)



(C)

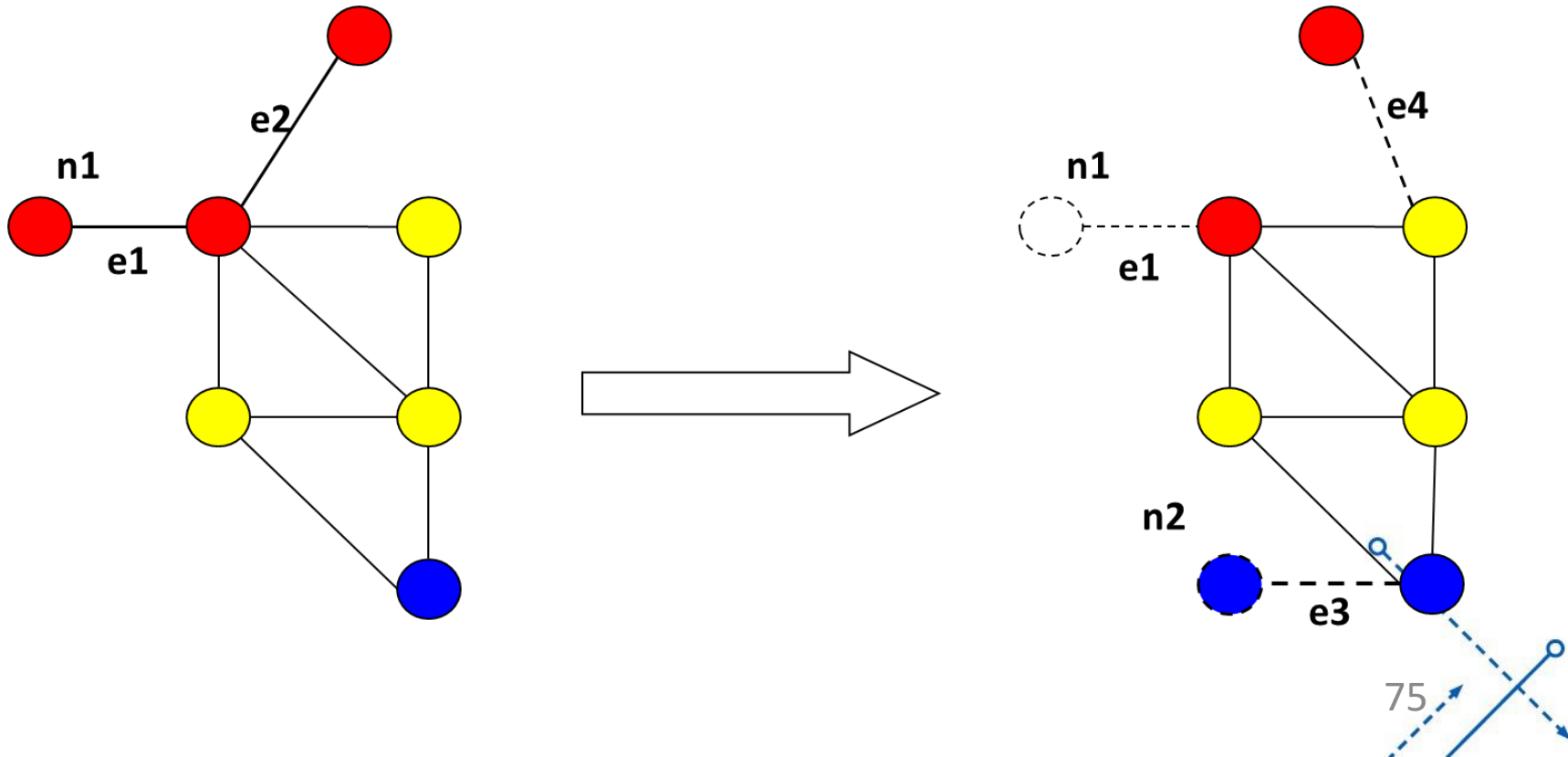


(D)



# Match graphs - adjustable distance

- **Graph edit distance** is the number of changes required to convert one graph into the other.



# Reference

---

- Mihalcea, Rada, and Dragomir Radev. *Graph-based natural language processing and information retrieval*. Cambridge university press, 2011.
- Frank M. Carrano, Data Abstraction and Problem Solving with C++: Walls and Mirrors, Frank M. Carrano, 6<sup>th</sup> Edition
- Minimum Spanning Tree in Graph - Week 13. 2 Problem: Laying Telephone Wire Central office.

