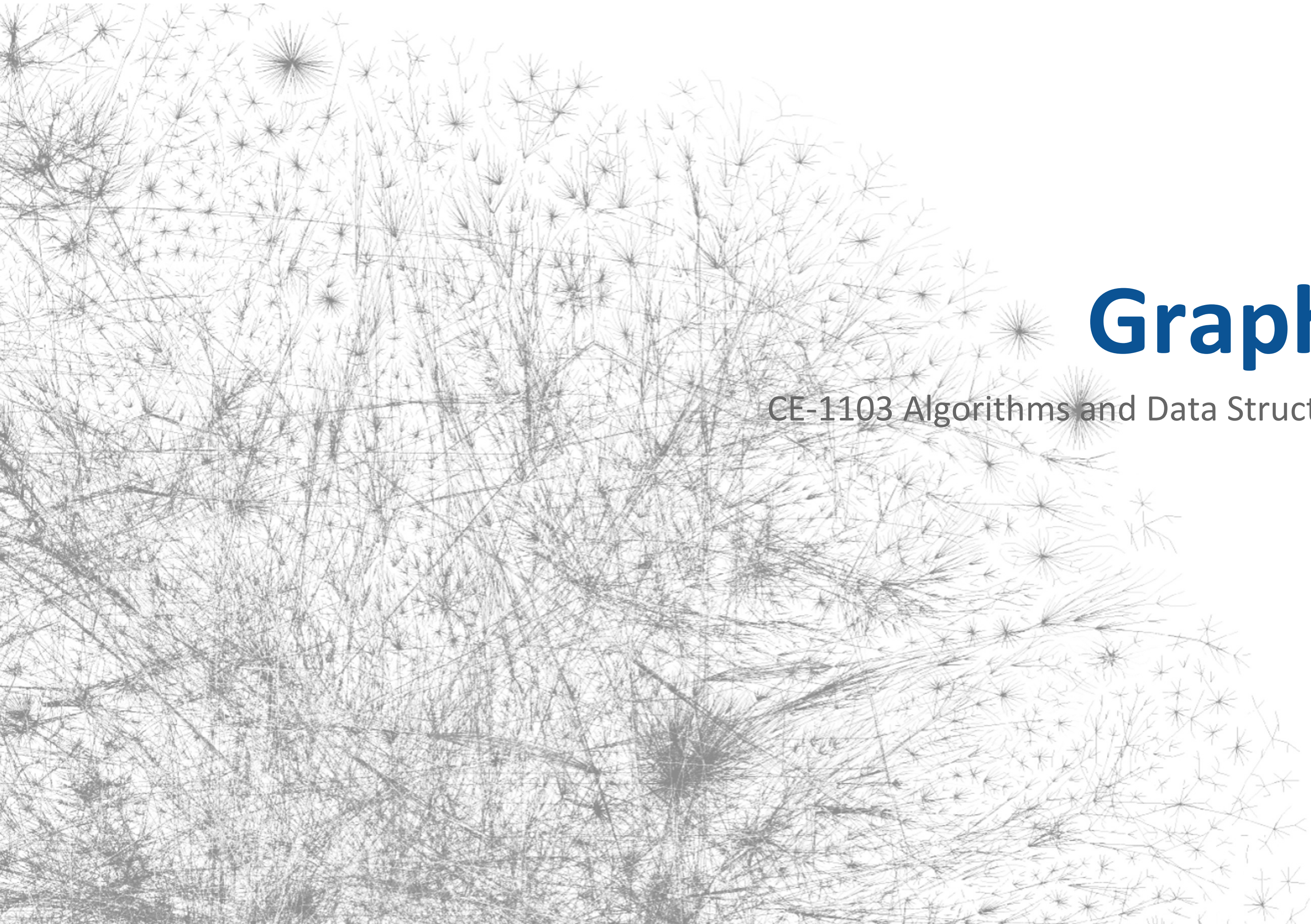


Graphs

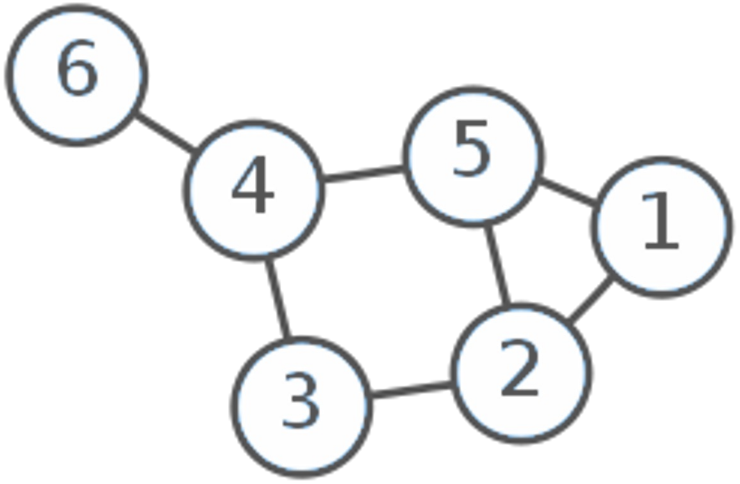
CE-1103 Algorithms and Data Structures

Disclaimer / Descargo de Responsabilidad

Introduction

➔ Graphs are general data structures that have a wide range of applications:

- ◆ Sociology
- ◆ Chemistry
- ◆ Geography
- ◆ Electrical engineering
- ◆ Industrial engineering

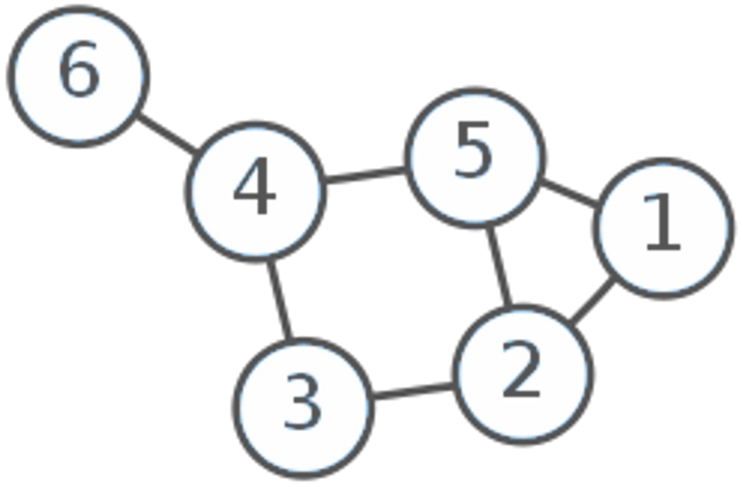


Introduction

➔ Graphs are general data structures that have a wide range o

i.e Identify criminal networks

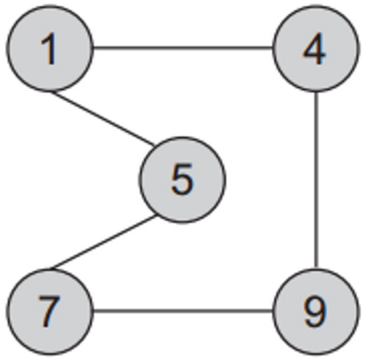
- ◆ Sociology
- ◆ Chemistry
- ◆ Geography
- ◆ Electrical engineering
- ◆ Industrial engineering



Concepts & definitions

- ➔ A graph *G* groups together physical or conceptual entities. A graph is composed of:
  - ◆ Vertices, nodes or points that represents each of the entities
  - ◆ Edges, arcs or lines that represents relationships between nodes.

- ➔ A graph is denoted as *G = {V, A}*

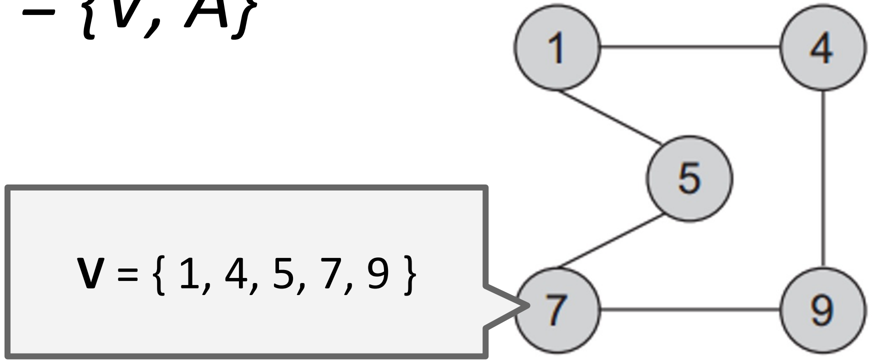


Concepts & definitions

➔ A graph *G* groups together physical or conceptual entities. A graph is composed of:

- ◆ Vertices, nodes or points that represents each of the entities
- ◆ Edges, arcs or lines that represents relationships between nodes.

➔ A graph is denoted as *G = {V, A}*



V = { 1, 4, 5, 7, 9 }

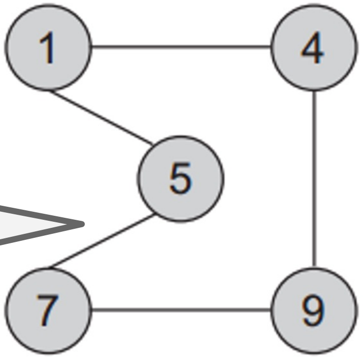
Concepts & definitions

➔ A graph *G* groups together physical or conceptual entities. A graph is composed of:

- **Vertices**, nodes or points that represents each of the entities
- **Edges**, arcs or lines that represents relationships between nodes.

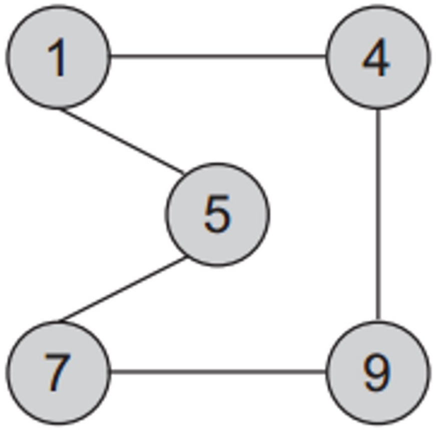
➔ A graph is denoted as *G = {V, A}*

A = { (1,4), (4,1), (1,5), (5,1), (4,9), (9,4), (5,7),(7,5) (7,9),(9,7) }

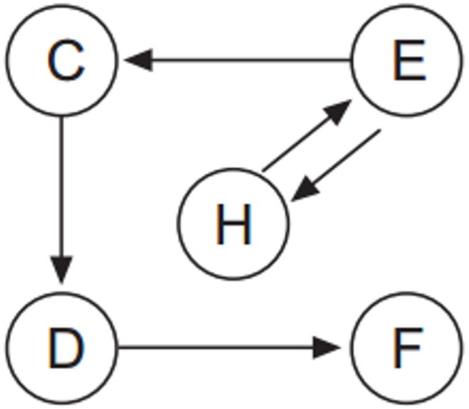


Concepts & definitions

➔A graph can be directed or undirected:



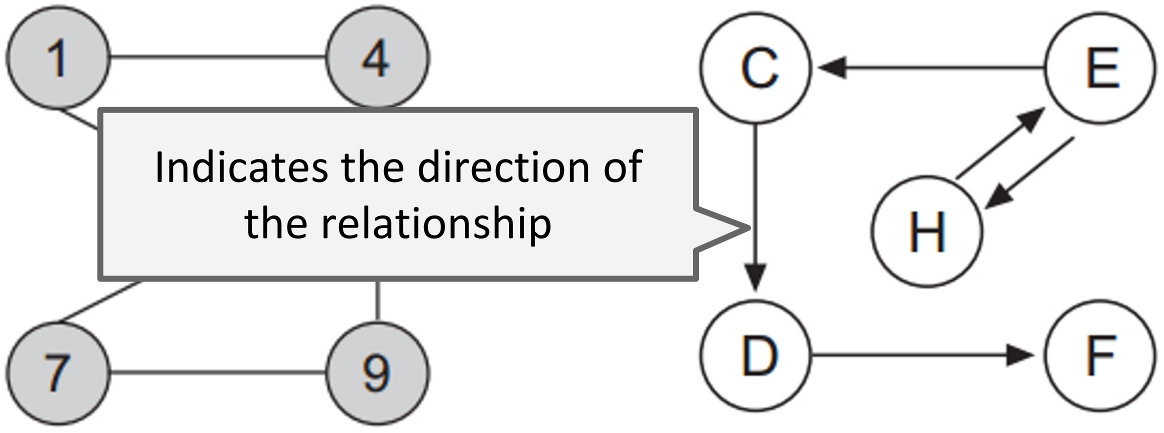
Undirected graph



Directed graph

Concepts & definitions

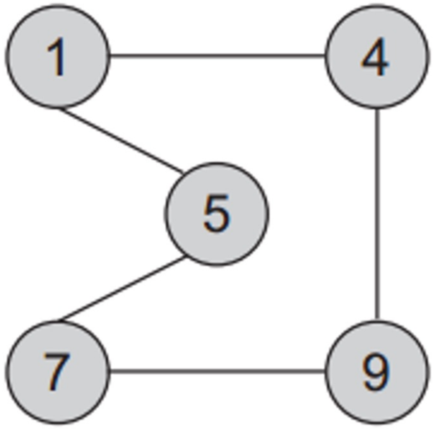
➔A graph can be directed or undirected:



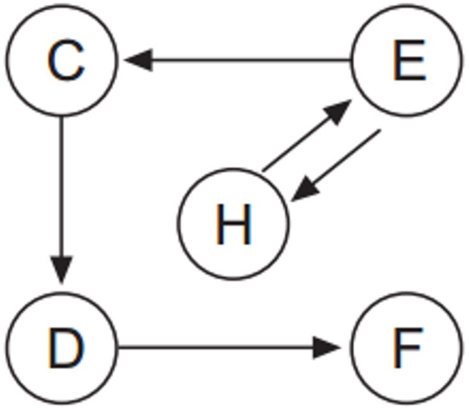
Undirected graph **Directed graph**

Concepts & definitions

➔ A graph can be directed or undirected:



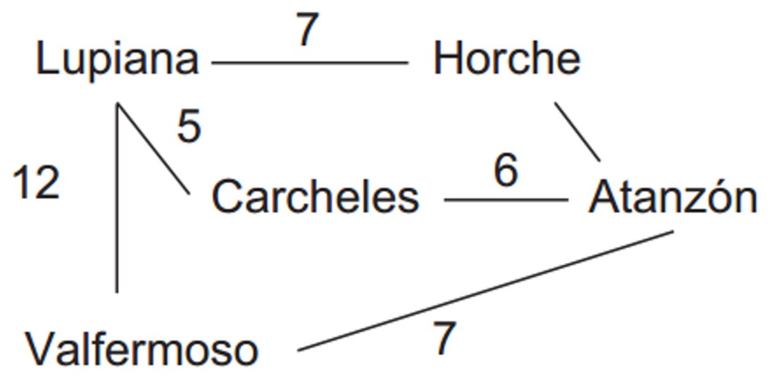
Vertices 7 and 9 are adjacent vertices



Directed graph

Concepts & definitions

- ➔ An edge can have a weight associated denoting a magnitude associated with the relationship
- ➔ Graphs with weight set on the edges are called **weighted graphs**



Weighted graph

Concepts & definitions

- ➔ The degree of *v* is a quality of a node of a graph. In an **undirected graph** it is the number of edges that contains *v*

- ➔ In a directed graph the **indegree** is the number of tail ends adjacent to *v*. **Outdegree** is the number of head ends adjacent to *v*

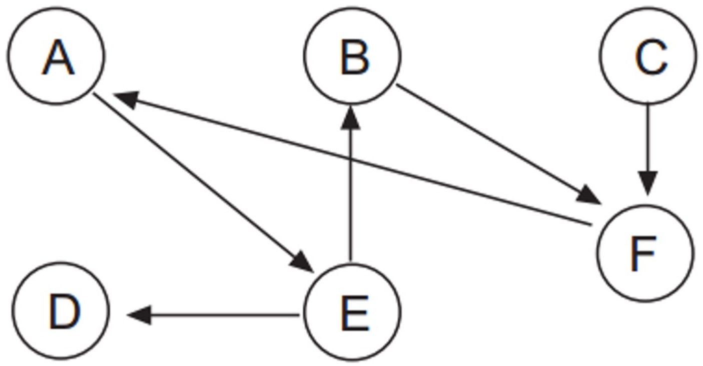
Concepts & definitions

➔ A path P = (v_0, v_1, v_2, … , v_n) is a set of vertices that form the path from v_0 to v_n. v_0 and v_n can be the same.

➔ It the vertices between v_0 to v_n are different, the path is called **simple path**

Concepts & definitions

→ A **cycle** is a simple path that begins and ends in the same node.

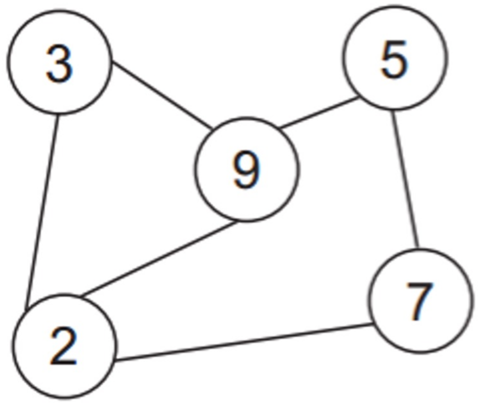


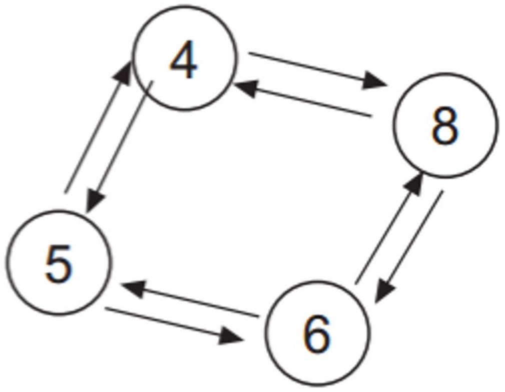
→ A **DAG** is a directed acyclic graph, a graph where there are no cycles

Concepts & definitions

➔ A graph is **connected** if there is a path between any pair of nodes that compose it

➔ A graph is **strongly connected** if the graph is connected and is a digraph





Graph representation

- ➔ Graphs can be represented using two different approaches:
  - ◆ Using a bidimensional array known as **adjacency matrix**
  - ◆ Using a dynamic representation known as **adjacency list**

- ➔ Choosing between one representation or the other depends of the type of the array and the operations that will be done

Graph representation

- Graphs can be represented using two different approaches:
  - Using a bidimensional array known as **adjacency matrix**
  - Using a dynamic representation known as **adjacency list**

If the graph is dense (lots of arcs), is better to chose a matrix

- Choosing epres other depends of t type of th operations that will be done

If the graph is sparse, choose the linked list

Graph representation

Adjacency Matrix

➔ Let G = {V, A} where V = $\{v_0, v_1, v_2, \ldots, v_{n-1}\}$ and A = {(vi, vj)}. Nodes can be represented by matrix A of nxn known as adjacency matrix. Each element of a_{ij} can take one of the following values:

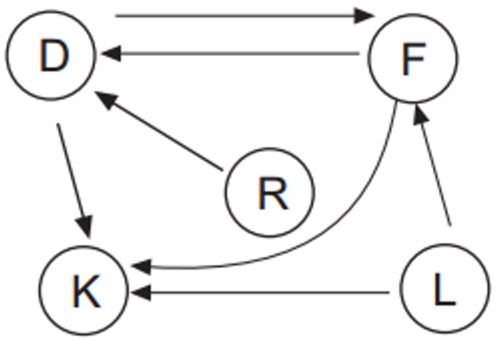
$$a_{ij} \left\{ \begin{array}{l} \textbf{0} \text{ if there is not an arc between } (v_i, v_j) \\ \\ \textbf{1} \text{ if there is an arc between } (v_i, v_j) \end{array} \right\}$$

Graph representation

Adjacency Matrix

➔ For example, let's say the nodes are {D, F, K, L, R} the matrix will be:

$$A = \begin{vmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

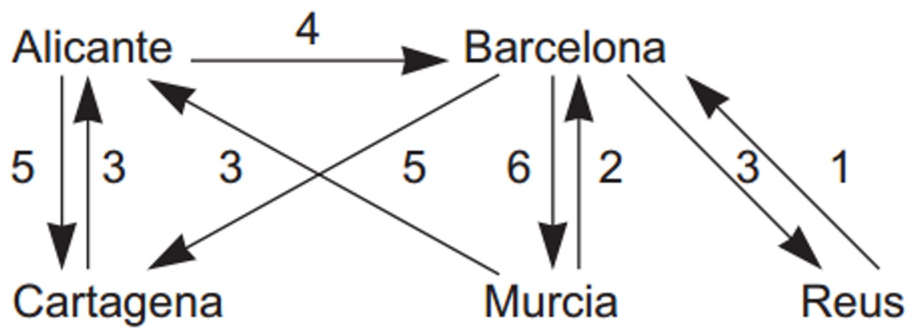


Graph representation

Adjacency Matrix

➔ If the graph is weighted:

$$P = \begin{vmatrix} 0 & 4 & 5 & 0 & 0 \\ 0 & 0 & 3 & 6 & 3 \\ 3 & 0 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{vmatrix}$$



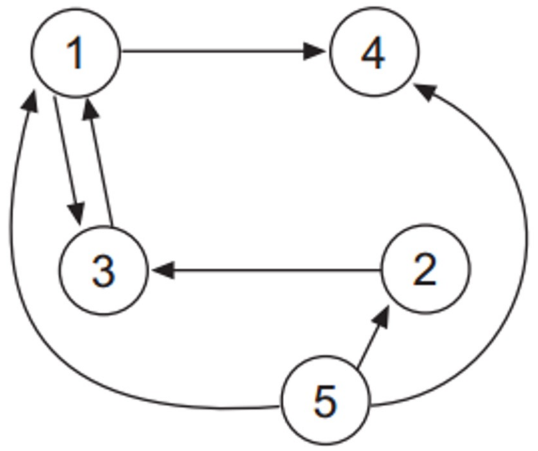
Graph representation

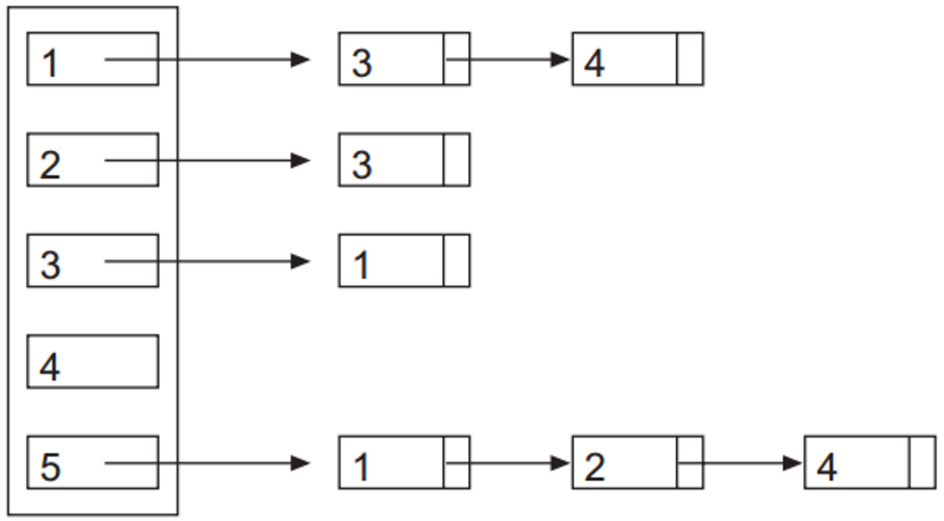
Adjacency List

- ➔ An adjacency list is a linked list where each element represents a node of the graph. Each element contains a list of relationships with other nodes, being the element node, the origin

Graph representation

Adjacency List





Graph traversals

- ➔ Traversing a graph involves **visiting** all reachable nodes starting from an specific node
- ➔ Basic traversing algorithm:
  - ◆ Let V be the set of vertices of the graph
  - ◆ Let W be the set of not visited nodes. Initially it only contains the initial node v
  - ◆ Let Y be the set of visited nodes.
  - ◆ On each pass of the algorithm, removes a node w, get processed and for each edge of w, if not visited, will be added to w
  - ◆ Algorithm ends when W is empty

Graph traversals

Breadth-First

- ➔ Uses a queue that keeps marked vertices

- ➔ FIFO achieves that from *v*, all adjacent vertices are processed first, then all the adjacents of the adjacents of *v*...and so on

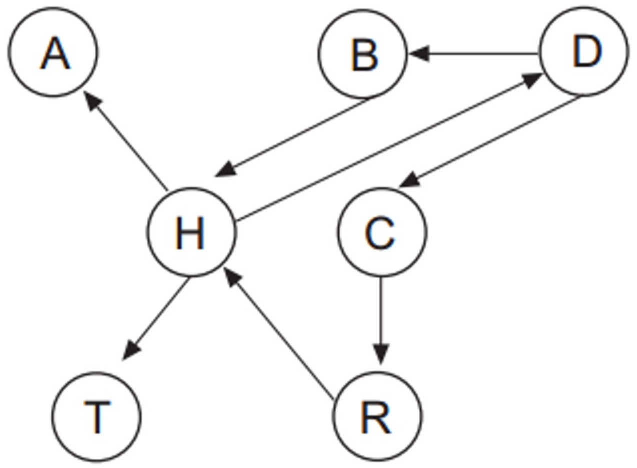
Breadth-First

➔ Algorithm:

1. Mark the start node v
2. Enqueue the start node v
3. Repeat step 4 and 5 until queue is empty
4. Dequeue node w from the queue, process w
5. Enqueue all adjacents nodes to w that are not marked, marked queued nodes
6. End

Graph traversals

Breadth-First



Queue	Visited
D	
B C	**D**
C *H*	**B**
H *R*	**C**
R *A T*	**H**
A T	**R**
T	**A**
Empty Queue	**T**

Graph traversals

Depth-First

➔ We saw that in the Breadth-First traversal, adjacents nodes **are processed in a FIFO** approach

➔ In Depth-First, the processing order is given by a **LIFO approach**

Graph traversals

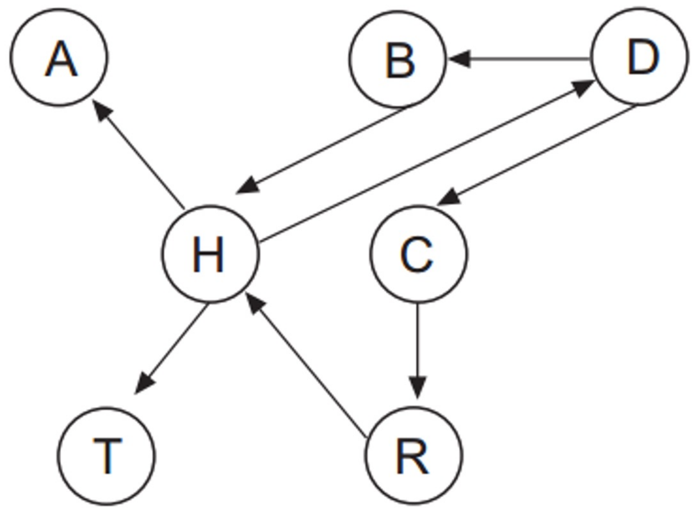
Depth-First

➔ Traverse beings with a node v. v is marked as visited and pushed to the stack. Top of the stack is popped. Each unvisited adjacent node of v is pushed to the stack.

➔ Continue until there are not more elements in the stack

Graph traversals

Depth-First



Stack	Visited nodes
D	
B C	**D**
B *R*	**C**
B *H*	**R**
B *A T*	**H**
B A	**T**
B	**A**
Empty Stack	**B**

Shortest-path: Dijkstra

➔ One of the most common problems is to determine the shortest path between a pair of nodes

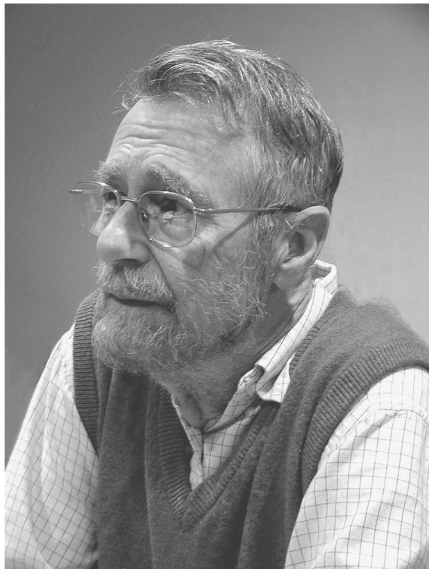
➔ For this kind of problem we consider a directed and weighted graph

➔ The length of the shortest path is the sum of the weight of each edge

Shortest-path: Dijkstra

➔ **Dijkstra algorithm** finds the shortest path from an origin node to all other nodes in a graph with positive weights

➔ Edsger Dijkstra (1930 - 2002) was a Dutch computer scientist that shaped computer programming as a recognized science



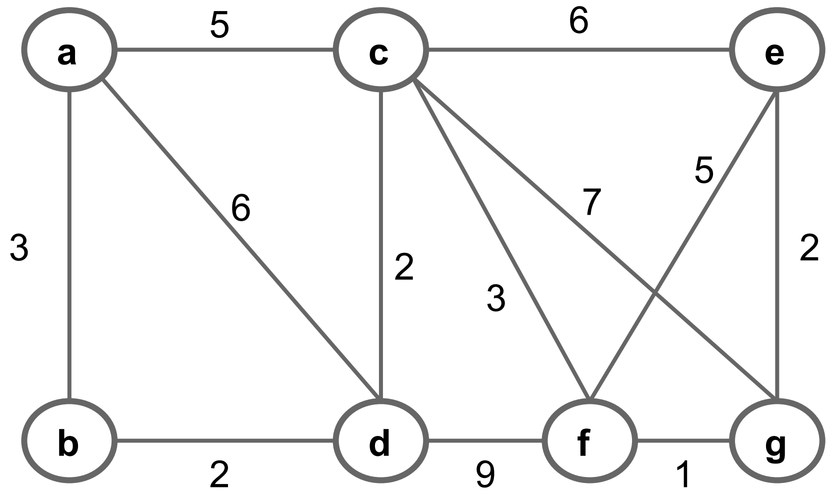
Shortest-path: Dijkstra

→ **Dijkstra algorithm** is an avid algorithm that selects the best solution on each step

→ Let’s see how it works

Graph algorithms

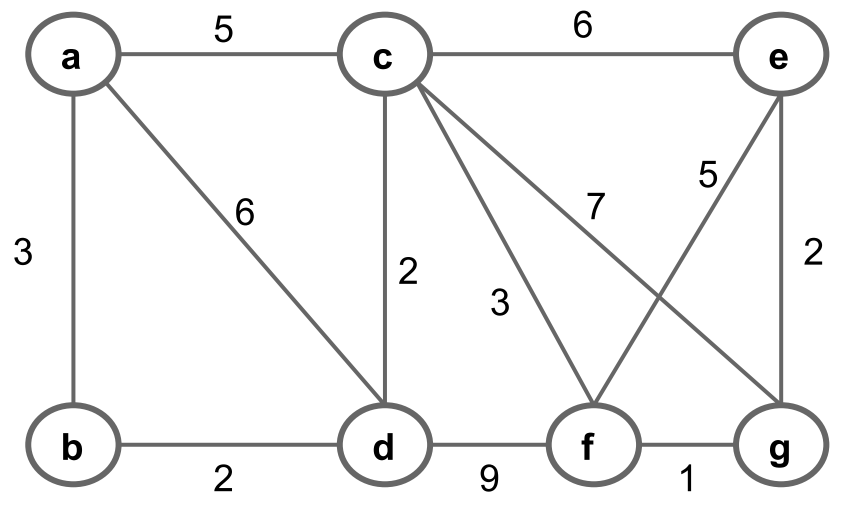
Shortest-path: Dijkstra



v	a	b	c	d	e	f	g

Graph algorithms

Shortest-path: Dijkstra

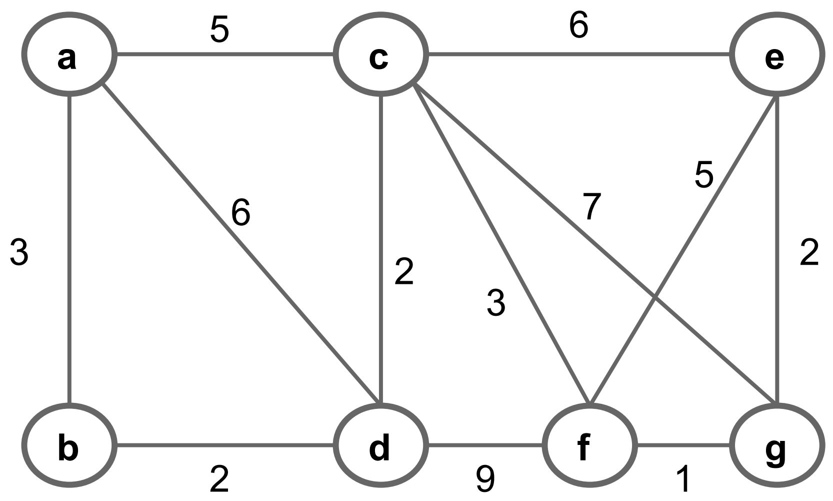


Let’s find the shortest path from a to every other vertex

v	a	b	c	d	e	f	g

Graph algorithms

Shortest-path: Dijkstra

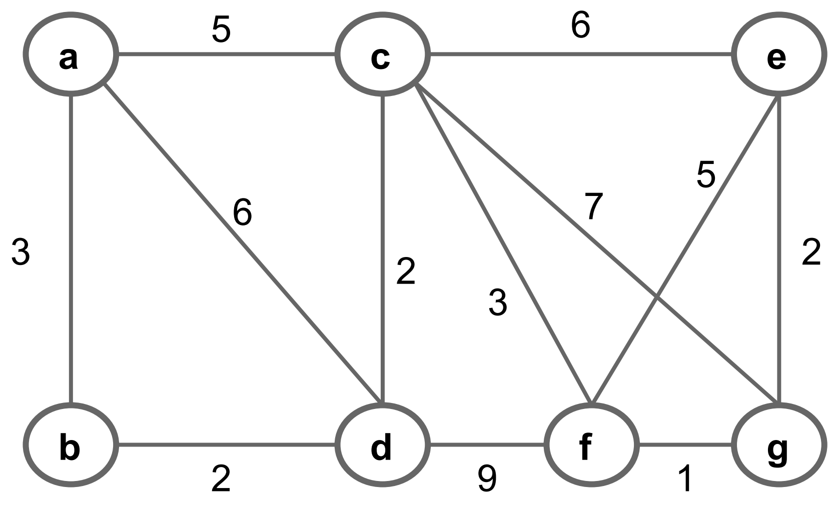


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-

Sub index indicates the **from** node

Graph algorithms

Shortest-path: Dijkstra

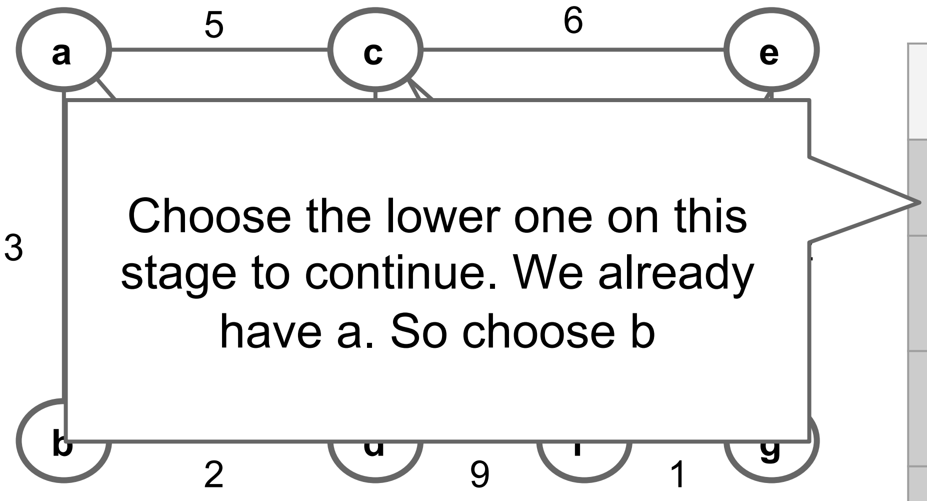


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-

Blue means: this is the shortest path. For example, from a to a, there won't be other path shorter

Graph algorithms

Shortest-path: Dijkstra

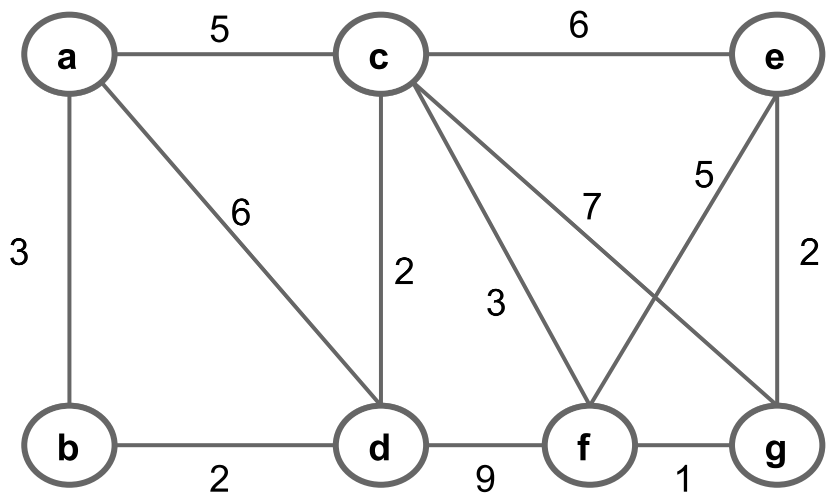


Choose the lower one on this stage to continue. We already have a. So choose b

v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-

Graph algorithms

Shortest-path: Dijkstra

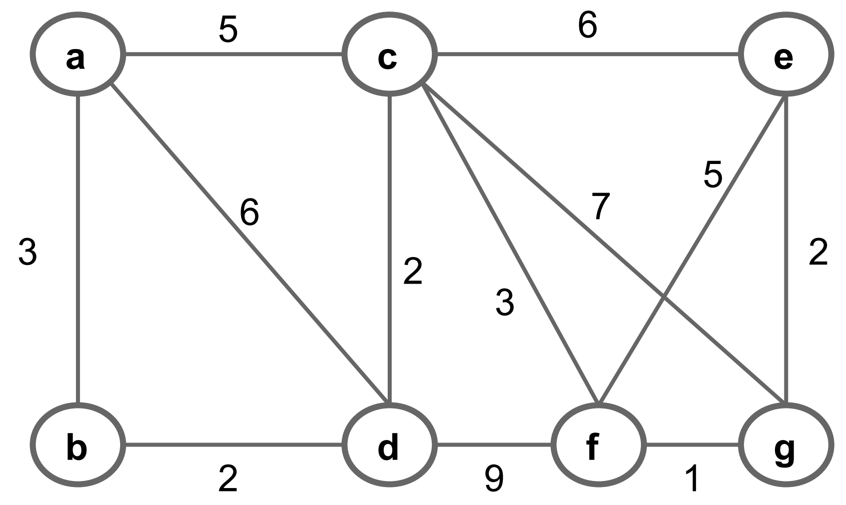


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a				-	-	-

We already knew that the shortest path to a was from a itself

Graph algorithms

Shortest-path: Dijkstra

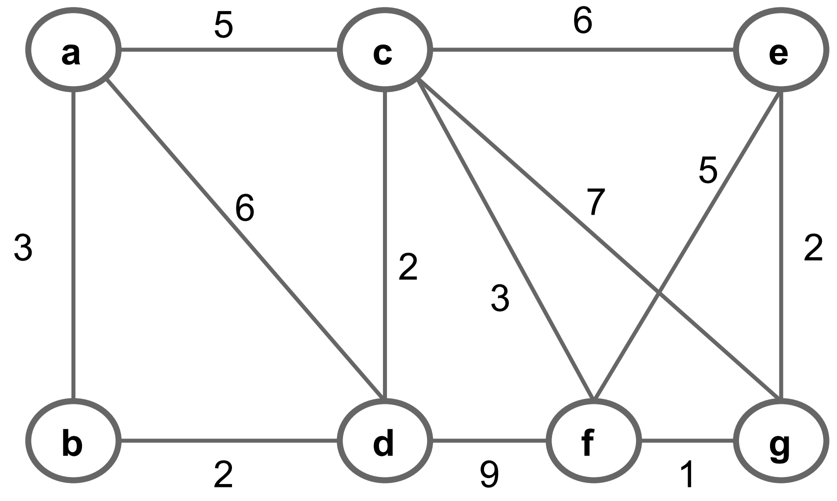


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a				-	-	-

The shortest path to b from node a is?

Graph algorithms

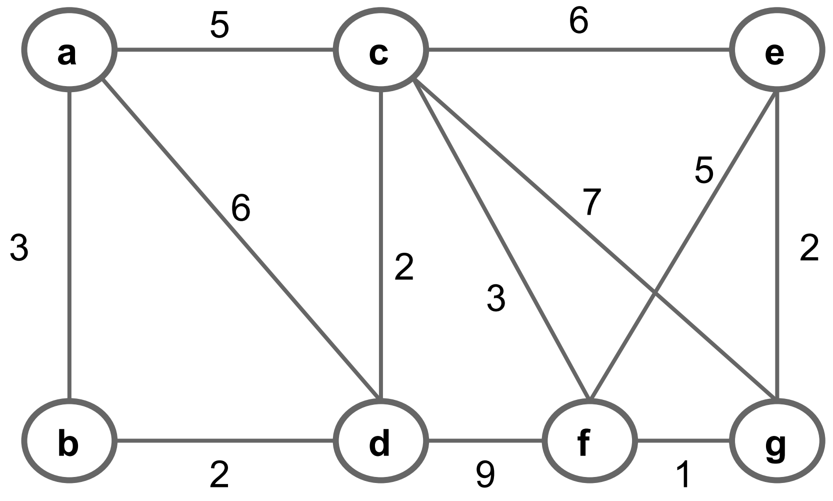
Shortest-path: Dijkstra



v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a			-	-	-

Graph algorithms

Shortest-path: Dijkstra

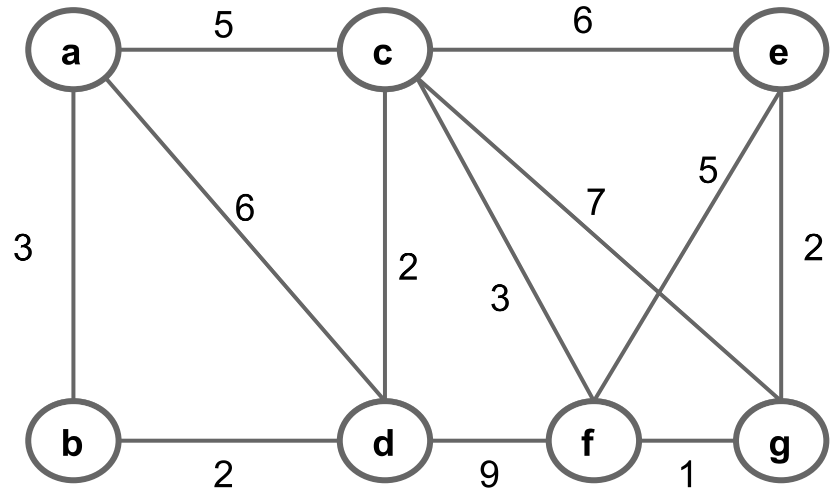


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-

To get to d from a we have two paths. Choose the lower one indicating the source node

Graph algorithms

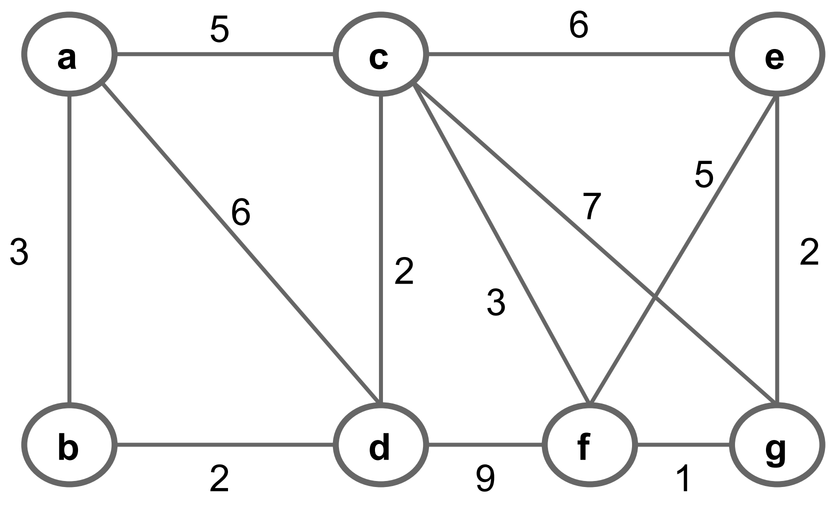
Shortest-path: Dijkstra



v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-
c	0_a	3_a	5_a				

Graph algorithms

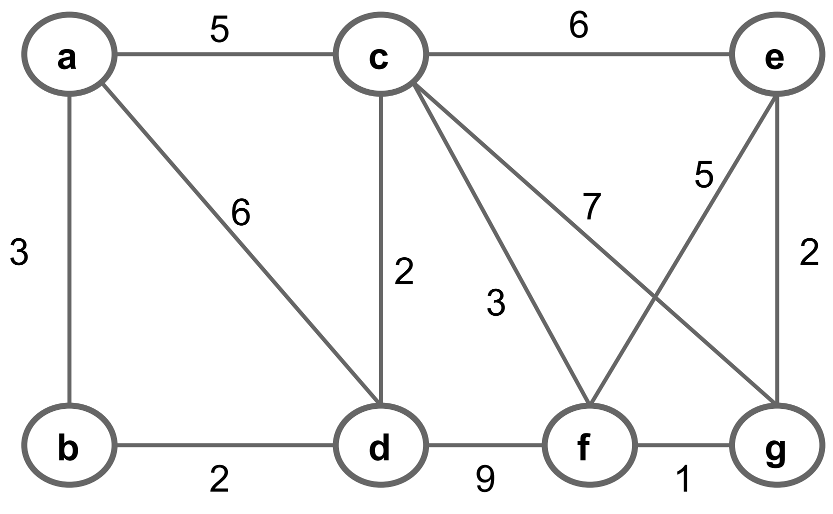
Shortest-path: Dijkstra



v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c

Graph algorithms

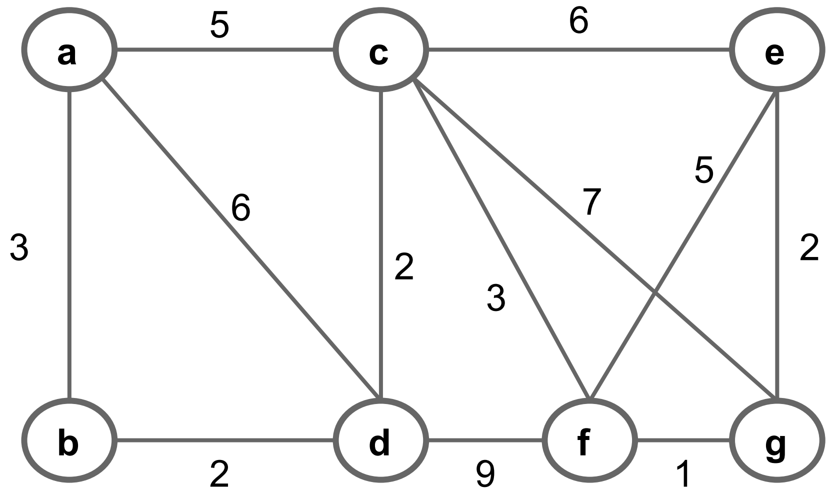
Shortest-path: Dijkstra



v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f
g							

No further improvements...so we are done!

Shortest-path: Dijkstra

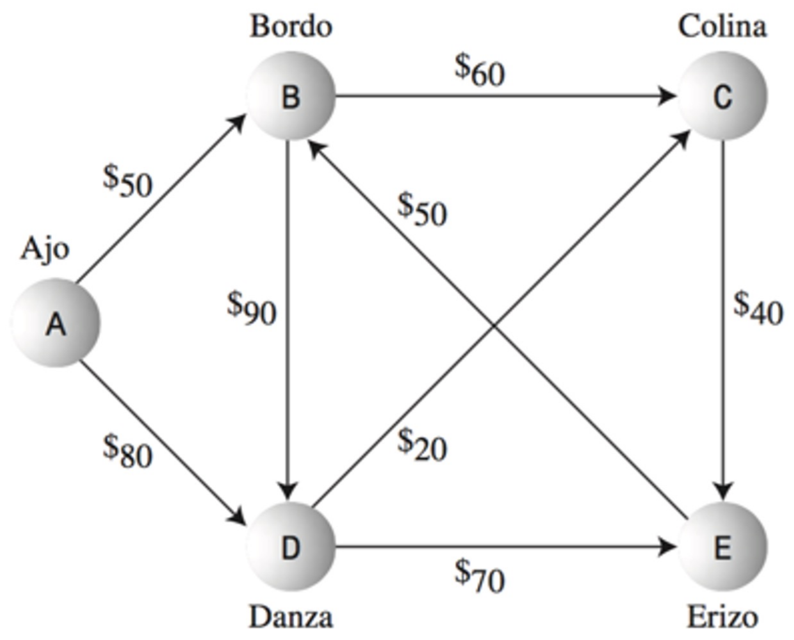


What is the shortest path from a to g? **a > c > f > g**

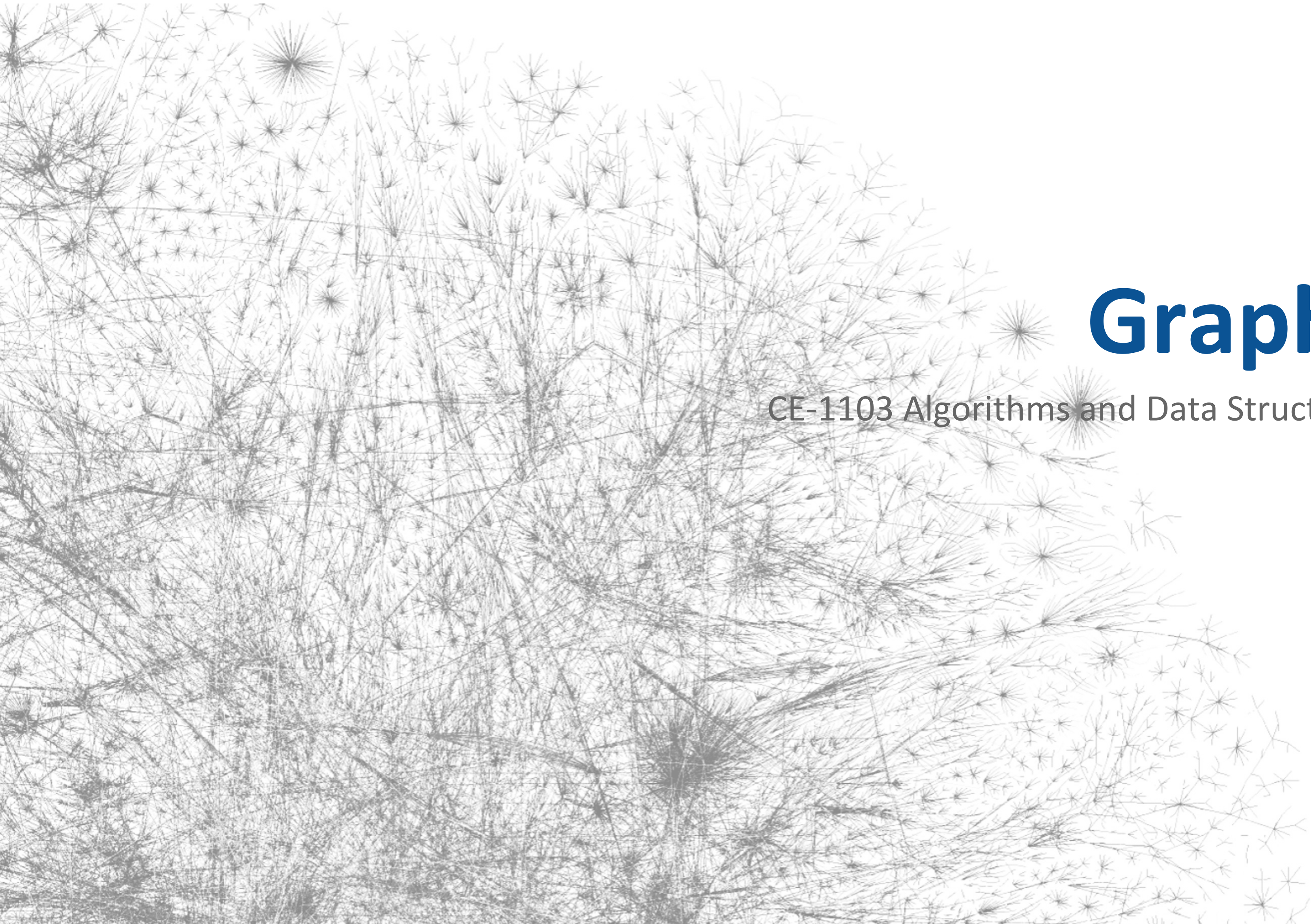
v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f

Graph algorithms

Shortest-path: Dijkstra



v	a	b	c	d	e
a	0_a	50_a	-	80_a	-



Graphs

CE-1103 Algorithms and Data Structures