Graphs

CE-1103 Algorithms and Data Structures

Disclaimer / Descargo de Responsabilidad

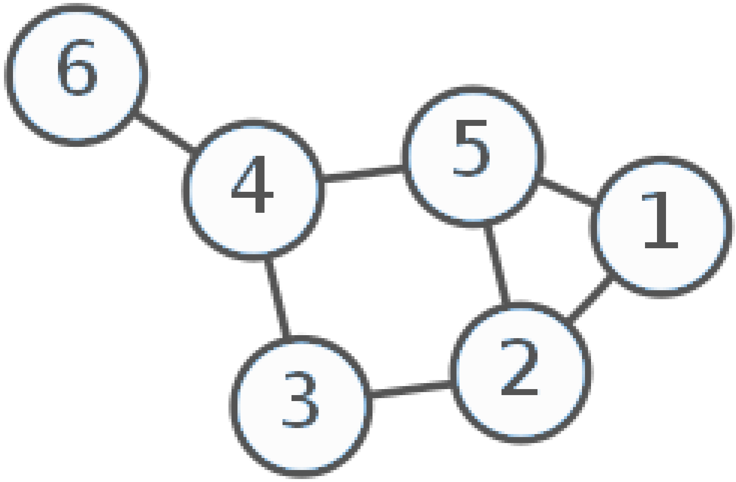
Esta presentación corresponde a una guía usada por el profesor durante las clases. La misma ha sido modificada para ser utilizado en el modelo de cursos asistidos por tecnología. No es una versión final, por lo que la misma podría requerir todavía hacer algunos ajustes. Para aspectos de evaluación esta presentación es solo una guía, por lo que el estudiante debe profundizar con el material de lectura asignado y lo discutido en clases para aspectos de evaluación.

This presentation corresponds to a guide material used by the professor during classes. It has been modified to be used in the model of technology-assisted courses. It is not a final version, so it may still require some adjustments. For evaluation aspects, this presentation is only a guide, so the student should delve with the assigned reading material and what has been discussed in class.

Introduction

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

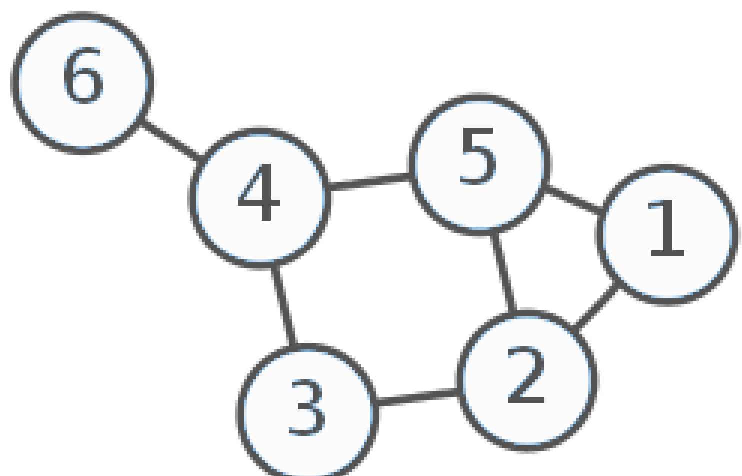
- Sociology
- Chemistry
- Geography
- Electrical engineering
- Industrial engineering



Introduction

- Graphs are general data structures that have a wide range … ns:
  - Sociology
  - Chemistry
  - Geography
  - Electrical engineering
  - Industrial engineering

i.e Identify criminal networks

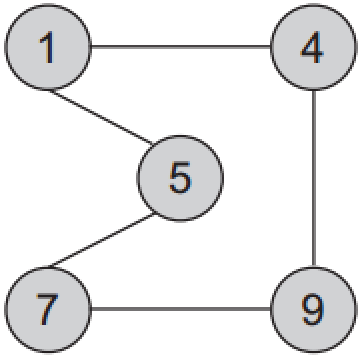


Concepts & definitions

➔ 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 $G = \{V, A\}$

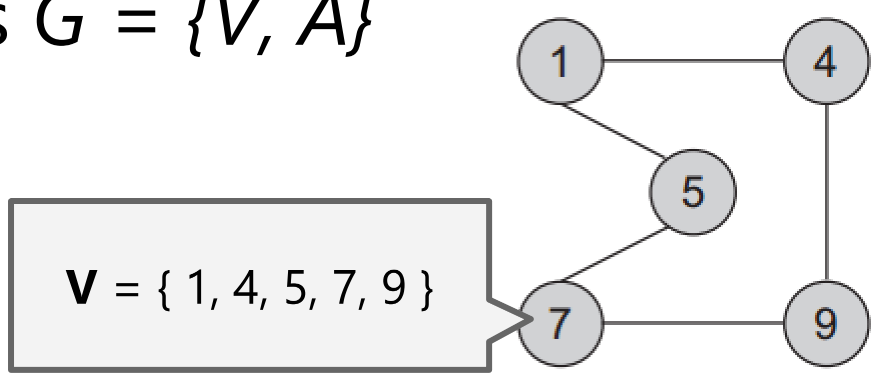


Concepts & definitions

➔ 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 $G = \{V, A\}$

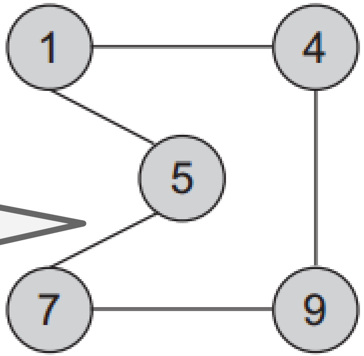


Concepts & definitions

- ➔ 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.

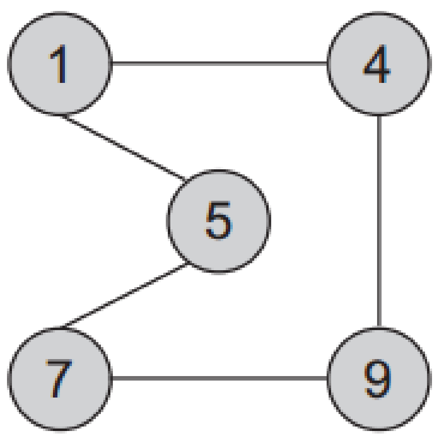
- ➔ A graph is denoted as $G = \{V, A\}$

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

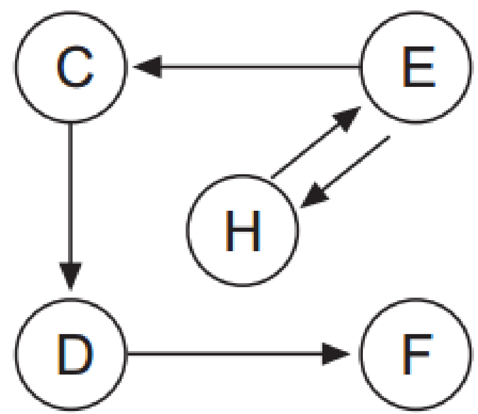


Concepts & definitions

➔A graph can be directed or undirected:



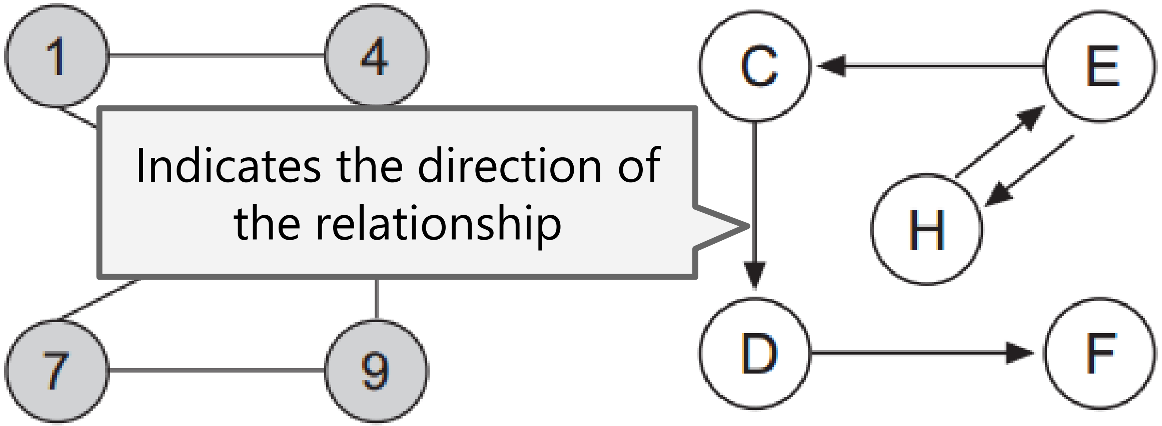
Undirected graph



Directed graph

Concepts & definitions

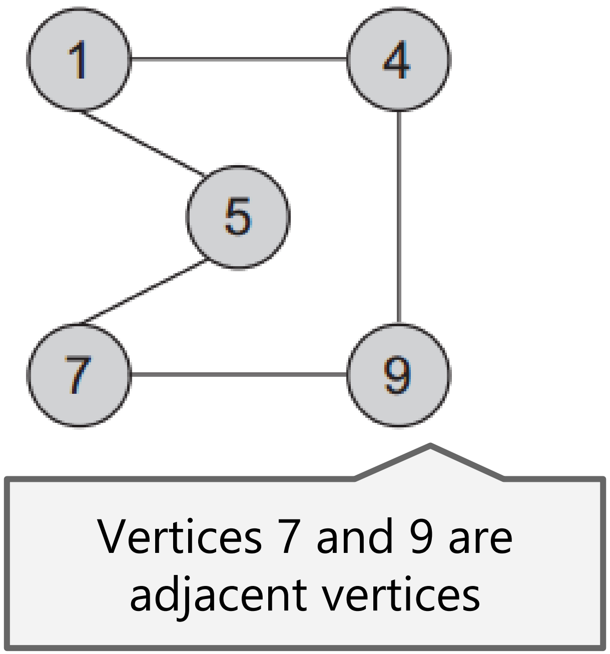
➔A graph can be directed or undirected:

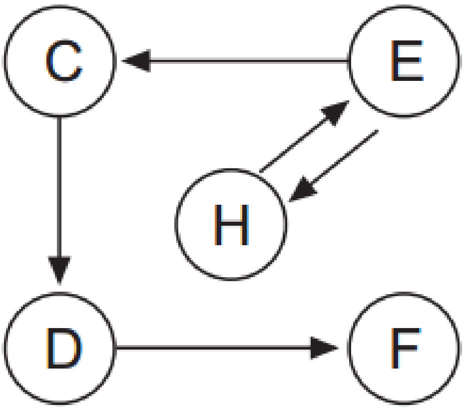


Undirected graph **Directed graph**

Concepts & definitions

➔A graph can be directed or undirected:

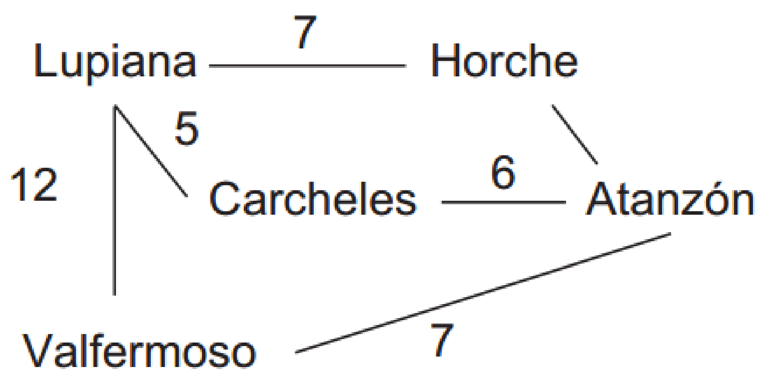




Directed graph

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*

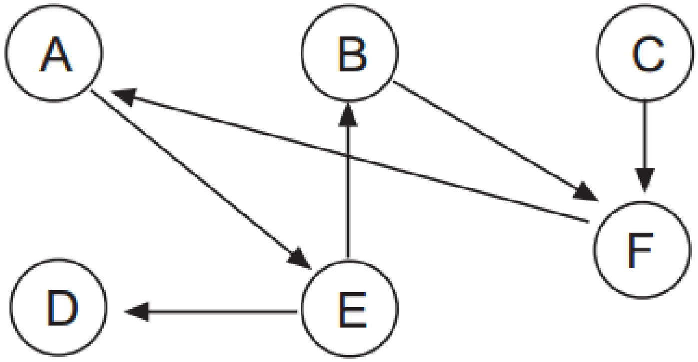
➔ In a directed graph the **indegree** is the number of tail ends adjacent to *v*. **Outdegree** is the number of head ends adjacent to *v*

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.

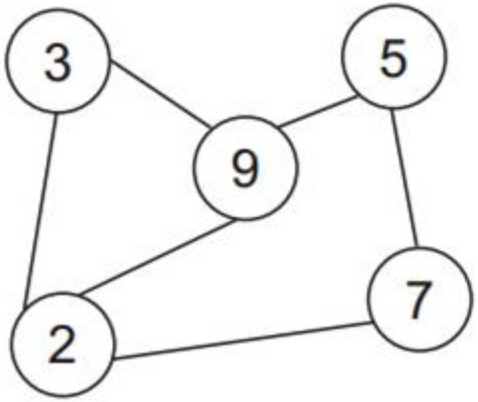


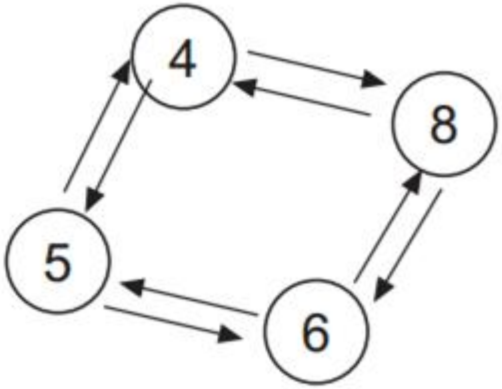
➔ A **DAG** is a directed acyclic graph, a graph where there are no cycles

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 … tation known as **adjacency list**

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

> If the graph is sparse, choose the linked list

- Choosing between one repre… the other depends of the type of the array and the operations that will be done

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:

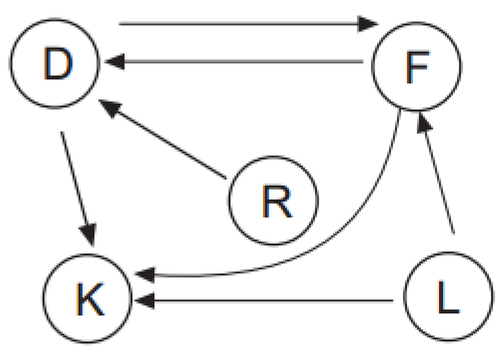
$$a_{ij} \begin{cases} \mathbf{0} \text{ if there is not an arc between } (v_i, v_j) \\ \mathbf{1} \text{ if there is an arc between } (v_i, v_j) \end{cases}$$

Graph representation

Adjacency Matrix

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

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

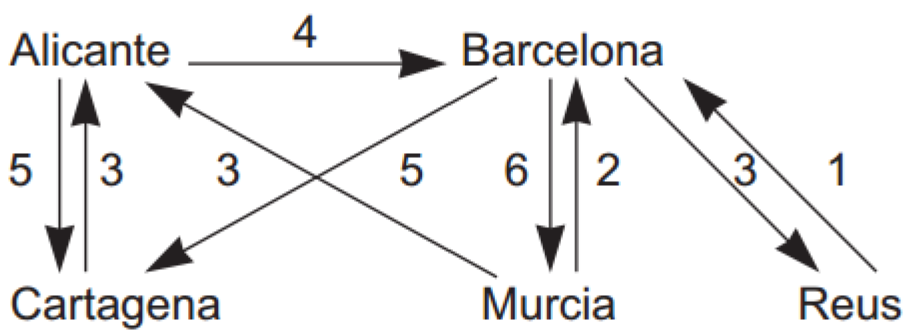


Graph representation

Adjacency Matrix

➔ If the graph is weighted:

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

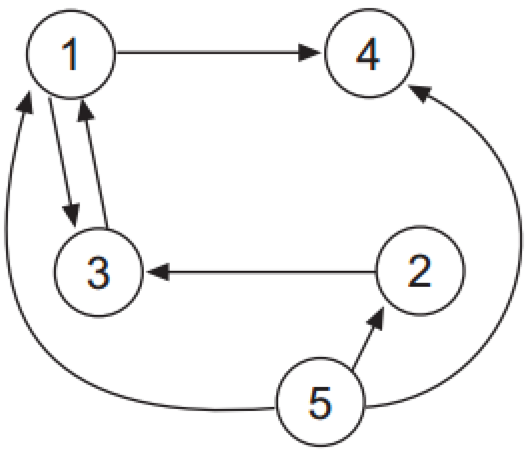


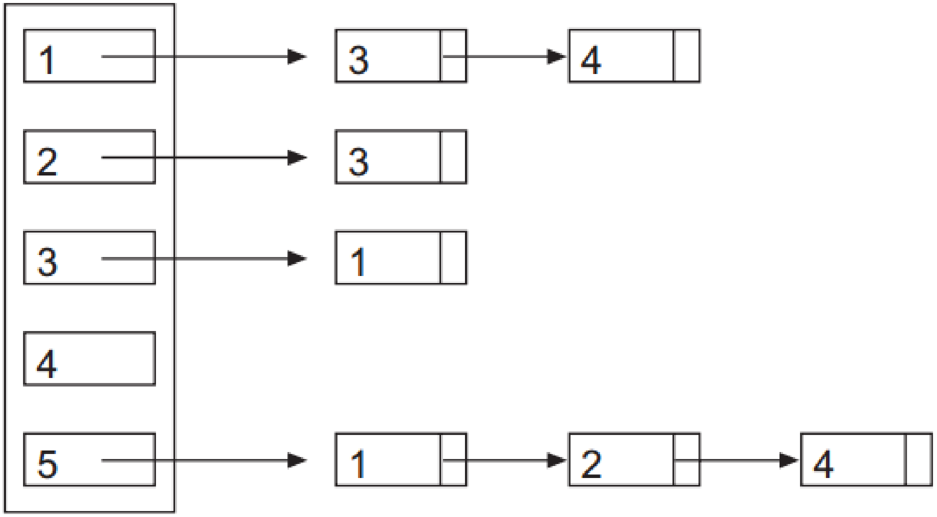
Graph representation

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

Graph traversals

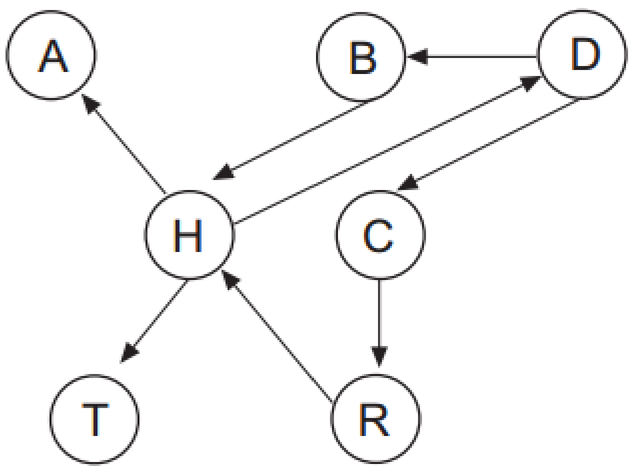
Breadth-First

➔ 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

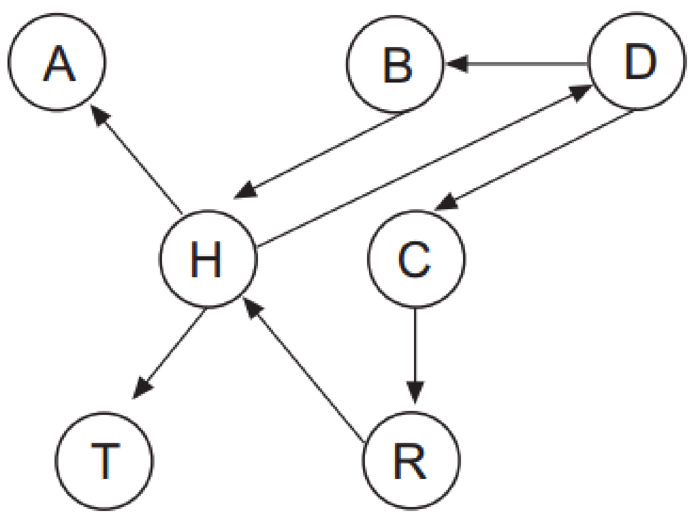
Depth-First

➔ 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**

Graph algorithms

Shortest-path: Dijkstra

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

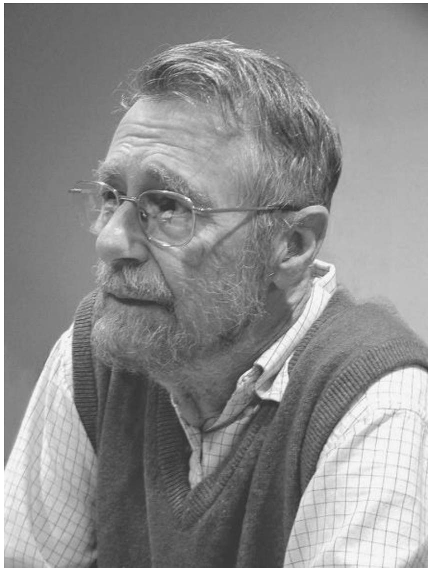
➔ For this kind of problem we consider a directed and weighted graph

➔ 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



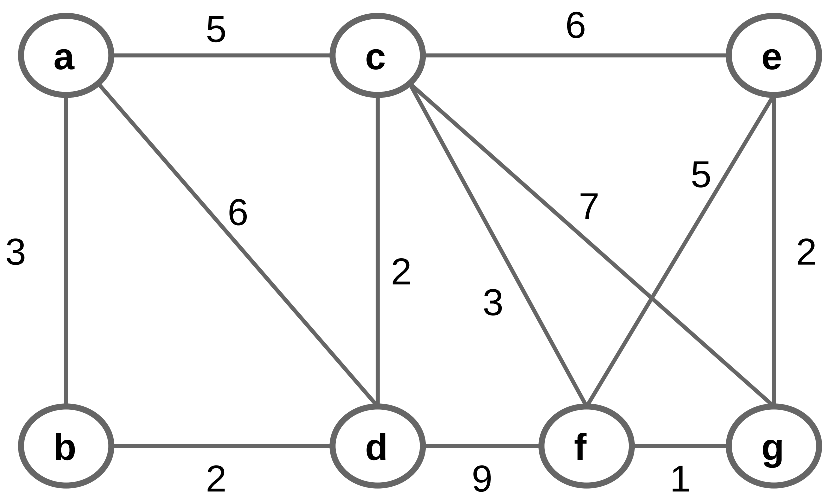
Shortest-path: Dijkstra

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

➔ Let's see how it works

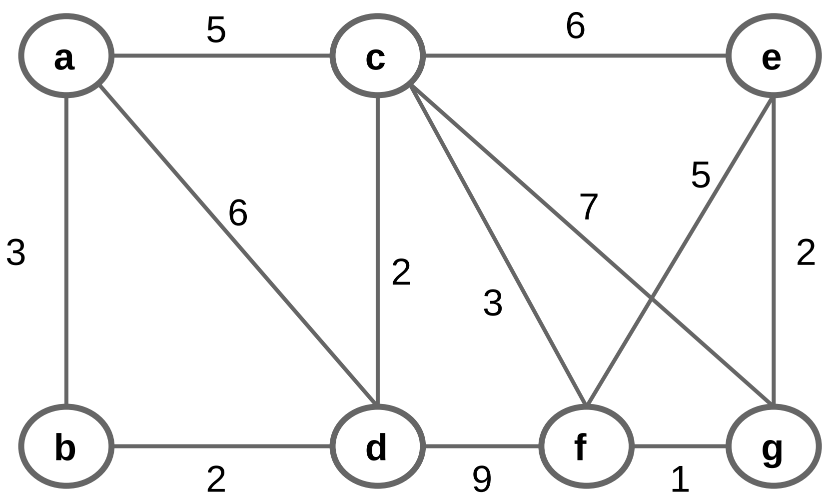
Graph algorithms

Shortest-path: Dijkstra



v	a	b	c	d	e	f	g

Shortest-path: Dijkstra

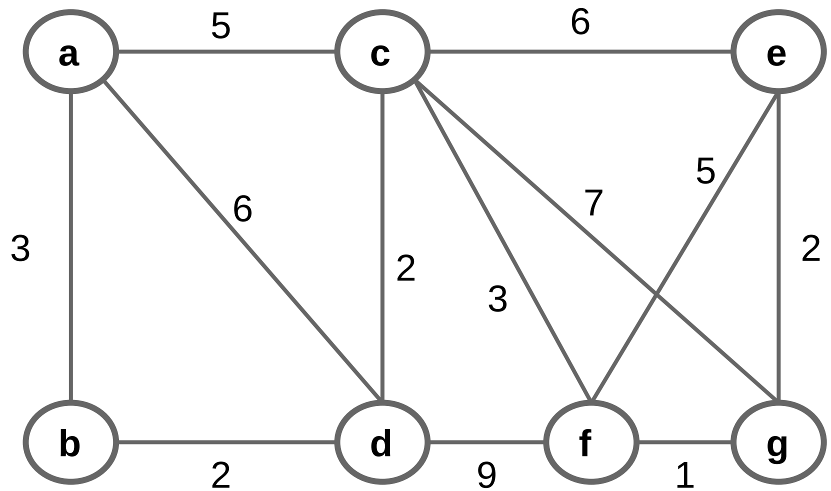


Let’s find the shortest path from a to every other vertex

v	a	b	c	d	e	f	g

Graph algorithms

Shortest-path: Dijkstra

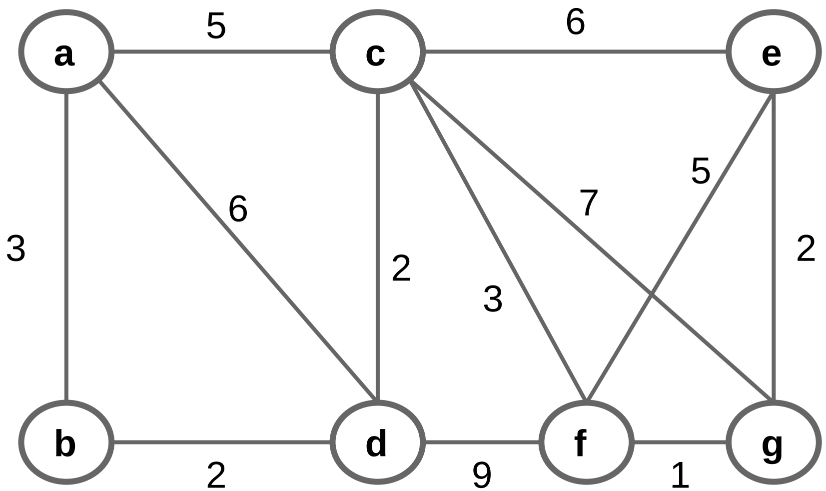


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-

Sub index indicates the **from** node

Graph algorithms

Shortest-path: Dijkstra

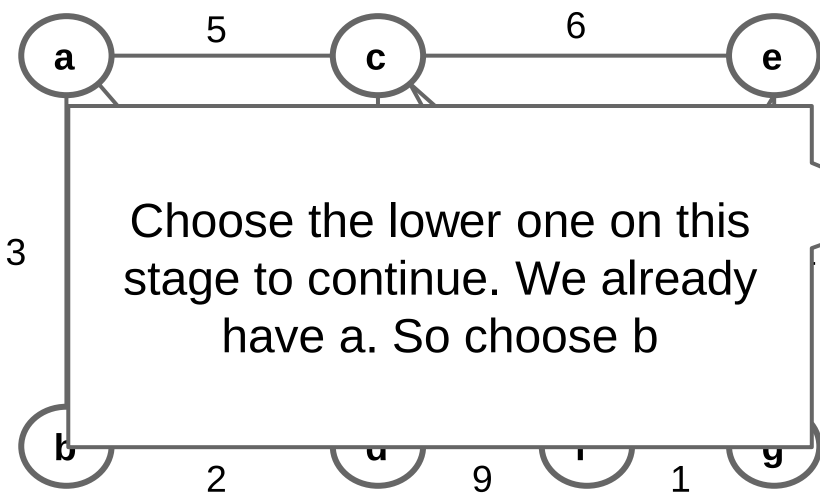


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-

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

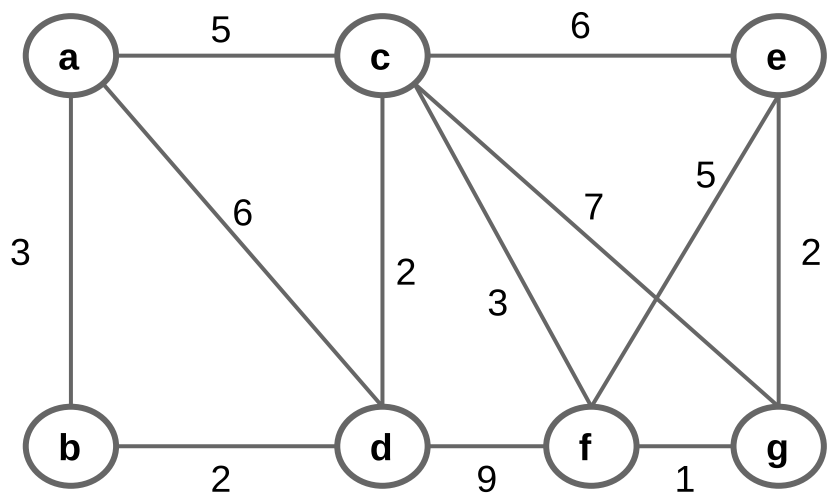


Choose the lower one on this stage to continue. We already have a. So choose b

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

Graph algorithms

Shortest-path: Dijkstra

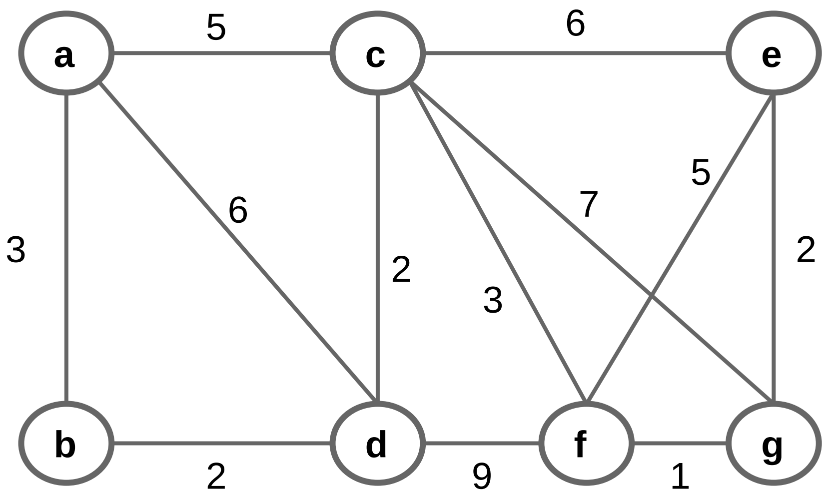


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a				-	-	-

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

Graph algorithms

Shortest-path: Dijkstra

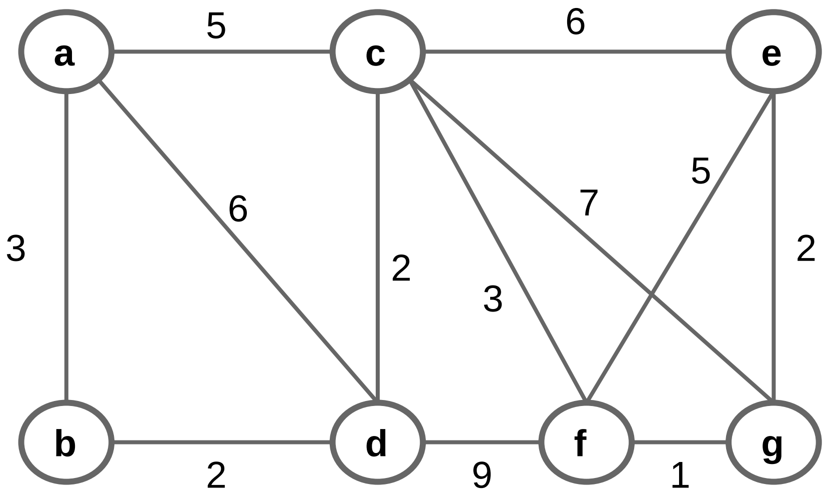


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a				-	-	-

The shortest path to b from node a is?

Graph algorithms

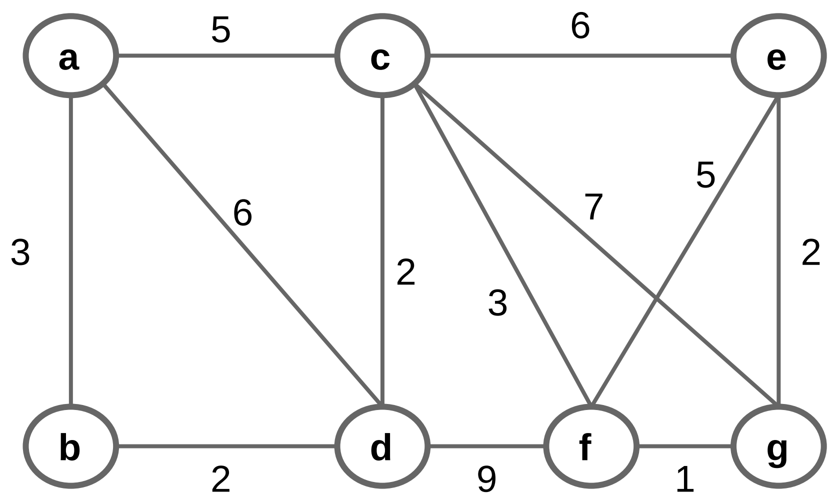
Shortest-path: Dijkstra



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

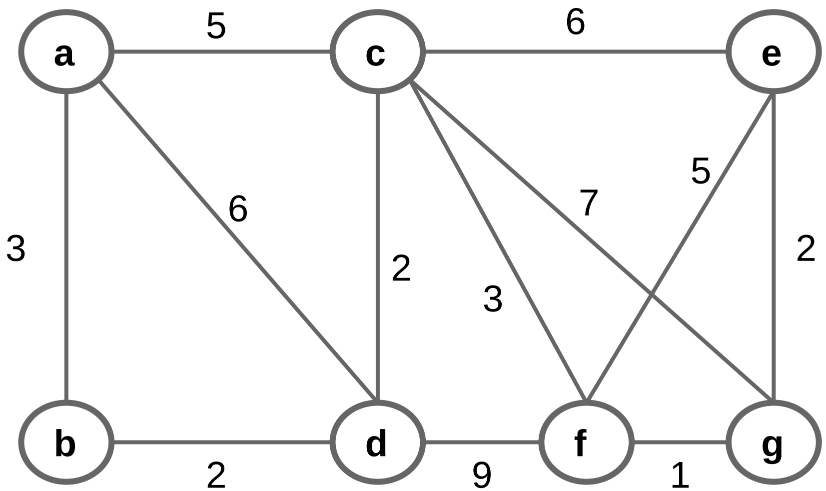


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-

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

Graph algorithms

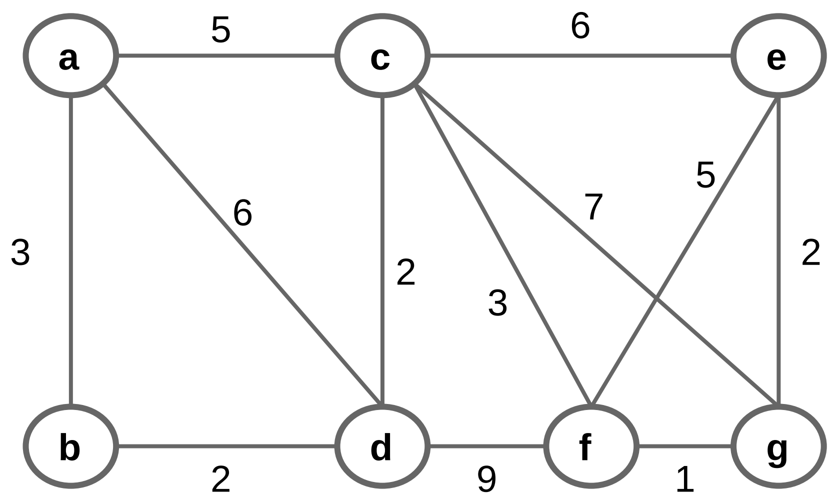
Shortest-path: Dijkstra



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

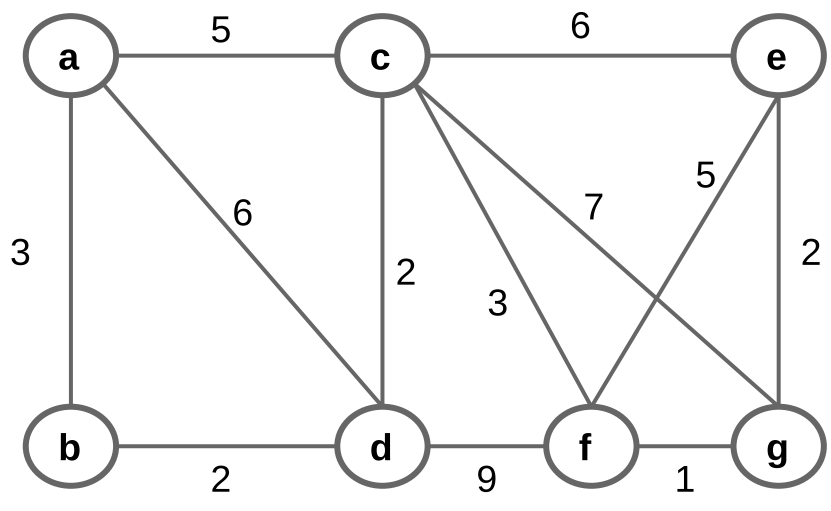
Shortest-path: Dijkstra



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

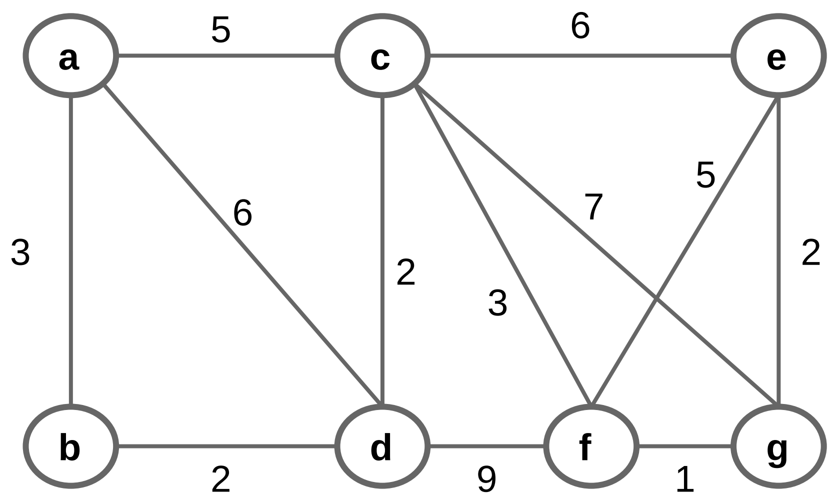


v	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	-	-	-
b	0_a	3_a	5_a	5_b	-	-	-
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f
g							

No further improvements...so we are done!

Graph algorithms

Shortest-path: Dijkstra

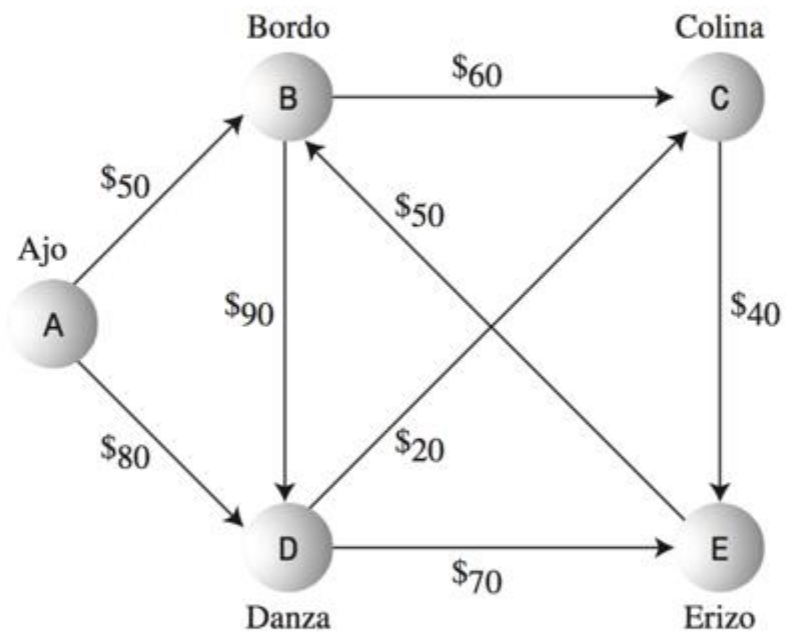


What is the shortest path from a to g? **a > c > f > g**

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

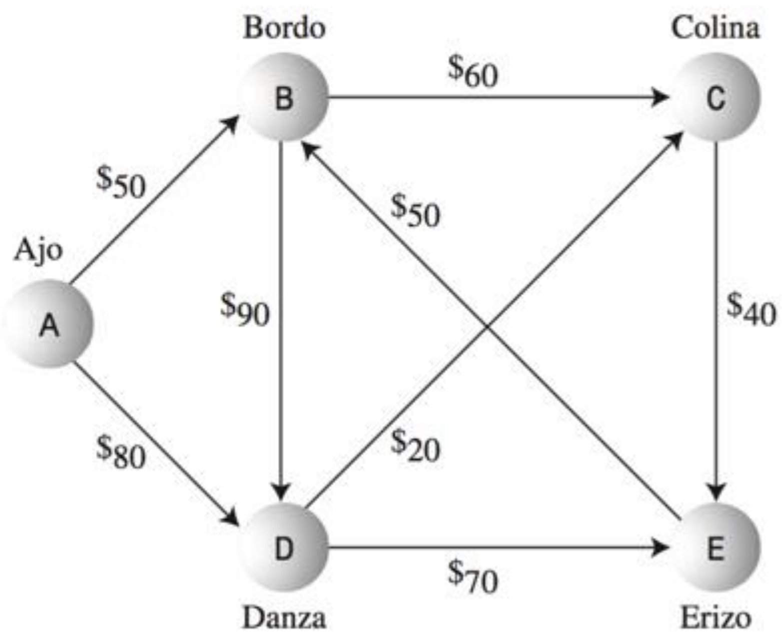
Shortest-path: Dijkstra



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

Graph algorithms

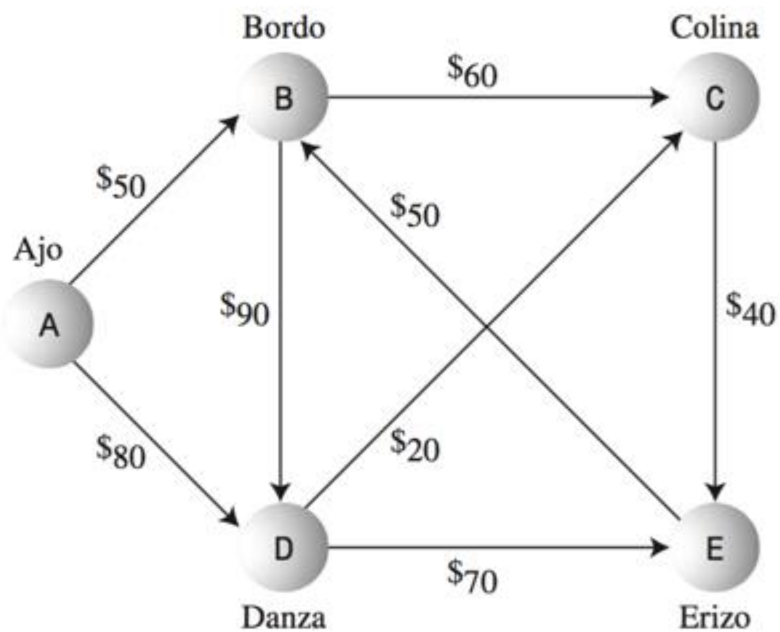
Shortest-path: Dijkstra



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

Graph algorithms

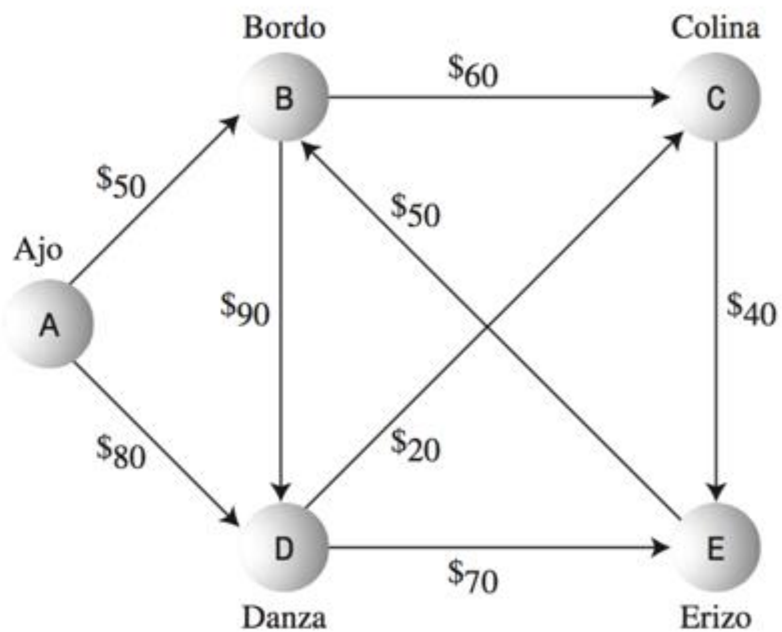
Shortest-path: Dijkstra



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

Graph algorithms

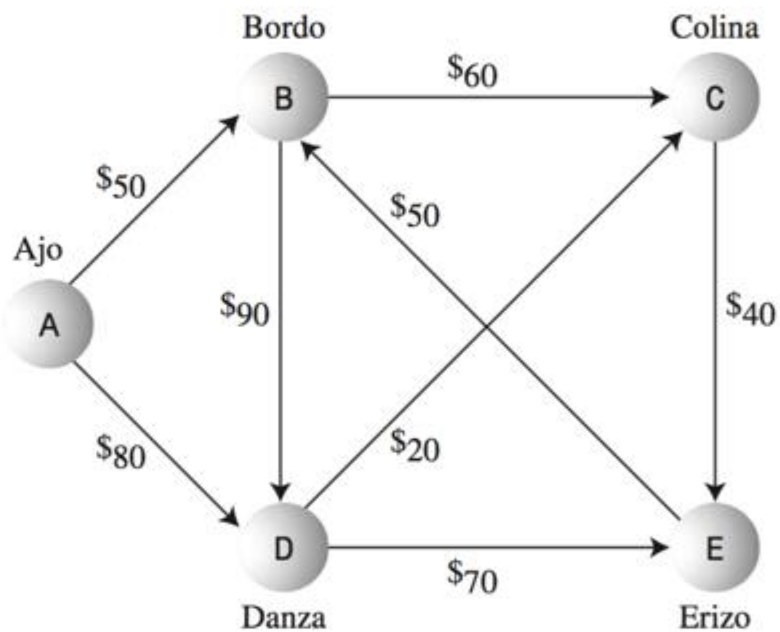
Shortest-path: Dijkstra



v	a	b	c	d	e
a	0_a	50_a	-	80_a	-
b	0_a	50_a	110_b	80_a	-
d	0_a	50_a	100_d	80_a	150_d
c	0_a	50_a	100_d	80_a	140_c

Graph algorithms

Shortest-path: Dijkstra

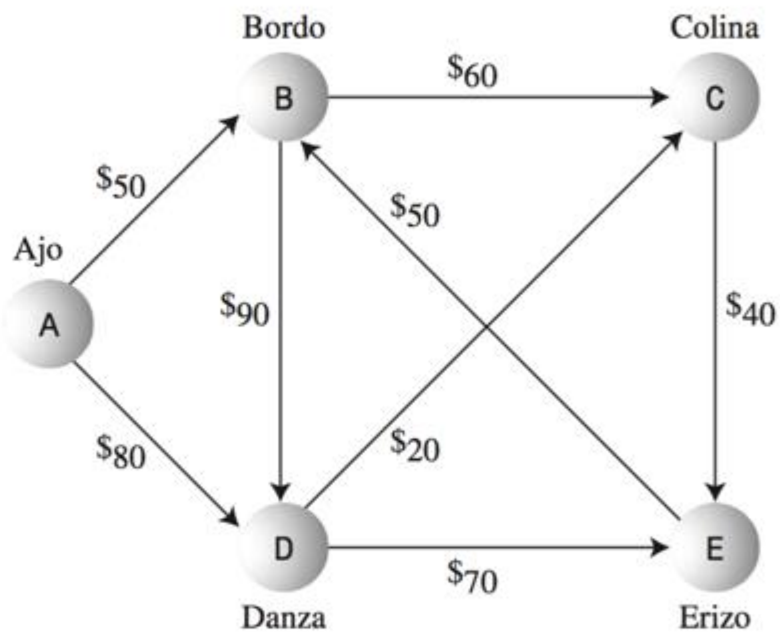


v	a	b	c	d	e
a	0_a	50_a	-	80_a	-
b	0_a	50_a	110_b	80_a	-
d	0_a	50_a	100_d	80_a	150_d
c	0_a	50_a	100_d	80_a	140_c

What is the cheapest way to get from a to e?

Graph algorithms

Shortest-path: Dijkstra



v	a	b	c	d	e
a	0_a	50_a	-	80_a	-
b	0_a	50_a	110_b	80_a	-
d	0_a	50_a	100_d	80_a	150_d
c	0_a	50_a	100_d	80_a	140_c

Graph algorithms
Shortest-path: Dijkstra

```
Foreach node set distance[node] = HIGH
SettledNodes = empty
UnSettledNodes = empty

Add sourceNode to UnSettledNodes
distance[sourceNode]= 0
while (UnSettledNodes is not empty) {
  evaluationNode = getNodeWithLowestDistance(UnSettledNodes)
  remove evaluationNode from UnSettledNodes
    add evaluationNode to SettledNodes
    evaluatedNeighbors(evaluationNode)
}
getNodeWithLowestDistance(UnSettledNodes){
  find the node with the lowest distance in UnSettledNodes and return it
}
evaluatedNeighbors(evaluationNode){
  Foreach destinationNode which can be reached via an edge from evaluationNode AND which is not in SettledNodes {
    edgeDistance = getDistance(edge(evaluationNode, destinationNode))
    newDistance = distance[evaluationNode] + edgeDistance
    if (distance[destinationNode]  > newDistance) {
      distance[destinationNode]  = newDistance
    evaluation.predecessor = evaluationNode
      add destinationNode to UnSettledNodes
    }
  }
}
```

Shortest-path: Dijkstra

Shortest-path: Floyd-Warshall

- ➔ How to calculate the shortest path from every node to every node?
  - ◆ Run Dijkstra from every node!
  - ◆ Yes but there is a more direct solution

- ➔ Floyd algorithm calculates using dynamic programming the shortest path from every node to every node

Shortest-path: Floyd

➔ Floyd algorithm represents the graph as a weighted matrix. Each arc (v_i, v_j) has a weight c_{ij}. If the arc does not exist, the value is ∞.

➔ The diagonal of the matrix is equal to zero.

Graph algorithms

Shortest-path: Floyd

➔ Floyd algorithm determines a new matrix D of $n_x n$ elements, where each D_{ij} is the minimum path from v_i to v_j

➔ A new matrix D_1, D_2, … , D_k, D_n is generated on each step from D_0. On each step a new vertex is included to determine if that vertex improves the paths to become shorter

Shortest-path: Floyd

$$D_0[i, j] = \begin{cases} c_{ij} \text{ weight of the arc from vertex}_i \text{ to vertex}_j \\ \infty \text{ if there is no arc} \end{cases}$$

$$D_1[i, j] = \text{minimum}(D_0[i,j], D_0[i,1]+D_0[1,j])$$

$$D_2[i, j] = \text{minimum}(D_1[i,j], D_1[i,2]+D_1[2,j])$$

$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j], D_{k-1}[i,k]+D_{k-1}[k,j])$$

Shortest-path: Floyd

- ➔ Another matrix Q_1, Q_2, … , Q_k, Q_n is generated on each step from Q_0. Q is the predecessor matrix.

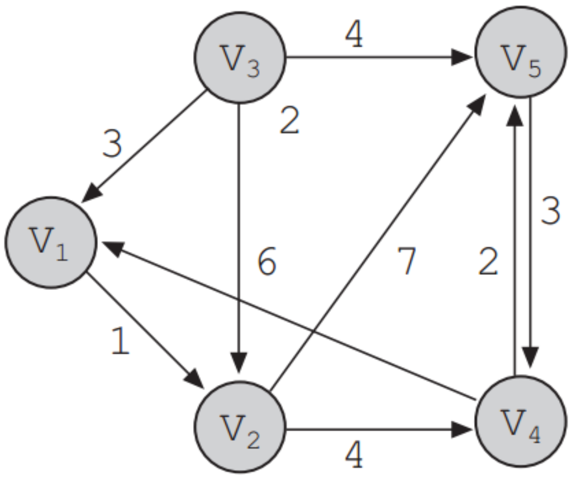
Shortest-path: Floyd

$$\mathbf{Q_0[i, j]} = \begin{cases} 0 \text{ if } i = j \text{ or } D_{ij} = \infty \\ i \text{ in all other cases} \end{cases}$$

$$\mathbf{Q_n[i, j]} = \begin{cases} Q_{n-1}[i, j] \text{ if } D_{n-1}[i,j] \leq D_{n-1}[i,n]+D_{n-1}[n,j] \\ Q_{n-1}[n, j] \text{ if } D_{n-1}[i,j] > D_{n-1}[i,n]+D_{n-1}[n,j] \end{cases}$$

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j])$$

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

i

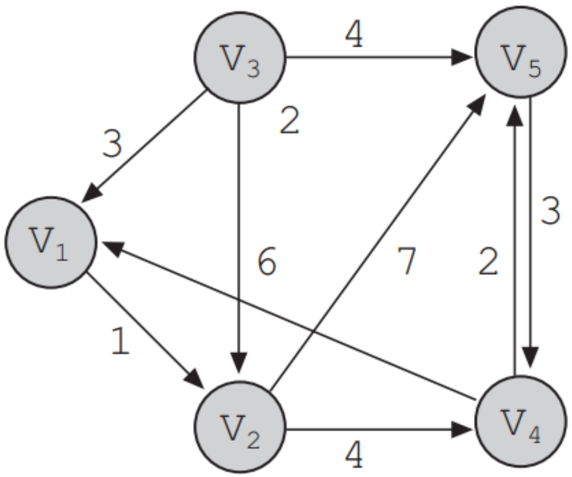
j

i

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

j

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

i

i

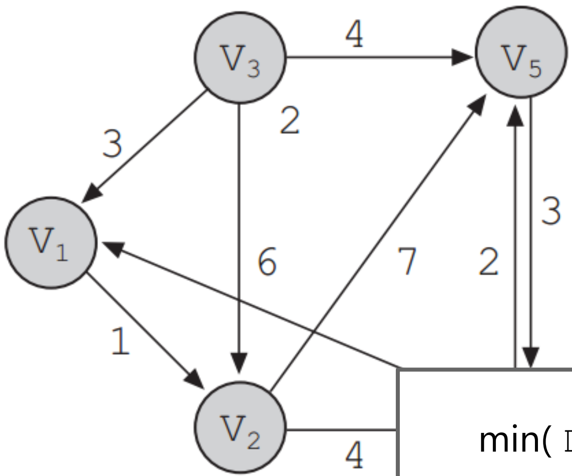
D_1	V_1	V_2	V_3	V_4	V_5
V_1	0				
V_2					
V_3					
V_4					
V_5					

i

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0				
V_2					
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
					0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

min($D_0[1,2]$, $D_0[1,1]+D_0[1,2]$)

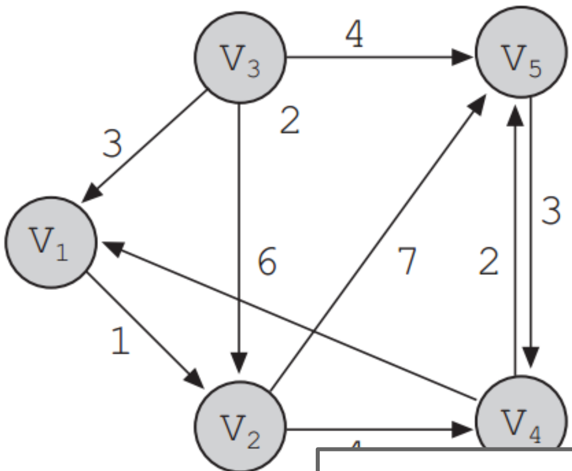
$D_k[i, j]$ = minimum($D_{k-1}[i,j]$,
$D_{k-1}[i,k]+$
$D_{k-1}[k,j]$)

D_1	V_1		V_3	V_4	V_5
V_1	0	1			
V_2					
V_3					
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2					
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$, $D_{k-1}[i,k]+D_{k-1}[k,j]$)

j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

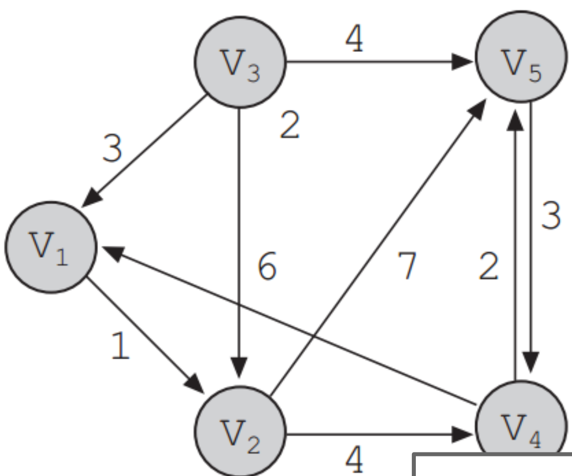
min($D_0[2,1]$, $D_0[2,1]+D_0[1,1]$)

				V_4	V_5
V_1		1	∞	∞	∞
V_2	∞				
V_3					
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0				
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

min($D_0[2,2], D_0[2,1]+D_0[1,2]$)

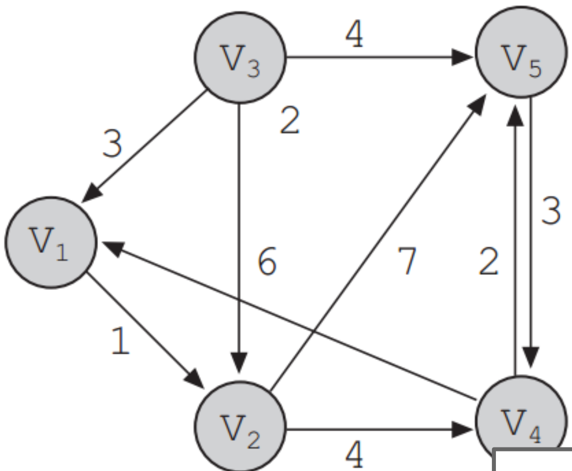
$D_k[i, j]$ = minimum($D_{k-1}[i,j]$,
$D_{k-1}[i,k]+$
$D_{k-1}[k,j]$)

					V_5
V_1	0		∞	∞	∞
V_2	∞	0			
V_3					
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0			
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

j

D₀	V₁	V₂	V₃	V₄	V₅
V₁	0	1	∞	∞	∞
V₂	∞	0	∞	4	7
V₃	3	2	0	∞	4
V₄	6	∞	∞	0	2
V₅	∞	∞	∞	3	0

Q₀	V₁	V₂	V₃	V₄	V₅
V₁	0	1	0	0	0
V₂	0	0	0	2	2
V₃	3	3	0	0	3
V₄	4	0	0	0	4
V₅	0	0	0	5	0

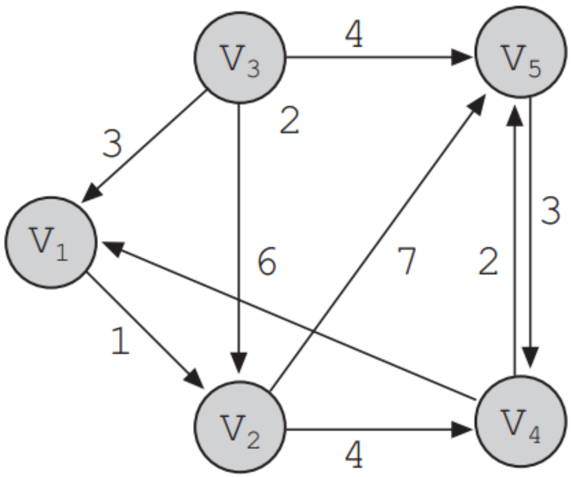
$\min(D_0[2,3], D_0[2,1]+D_0[1,3])$

V₁	0	1		∞	∞
V₂	∞	0	∞		
V₃					
V₄					
V₅					

Q₁	V₁	V₂	V₃	V₄	V₅
V₁	0	1	0	0	0
V₂	0	0	0		
V₃					
V₄					
V₅					

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$, $D_{k-1}[i,k]$ + $D_{k-1}[k,j]$)

j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

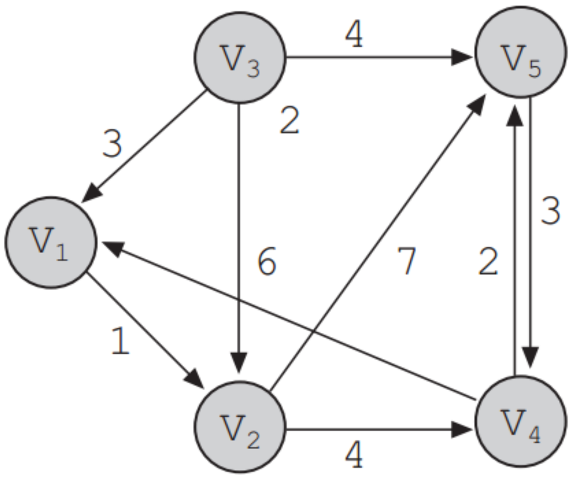
min($D_0[2,4]$, $D_0[2,1]$+$D_0[1,4]$)

V_1	0	1	∞	[illegible]	∞
V_2	∞	0	∞	4	
V_3					
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$,
$D_{k-1}[i,k]$ +
$D_{k-1}[k,j]$)

j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

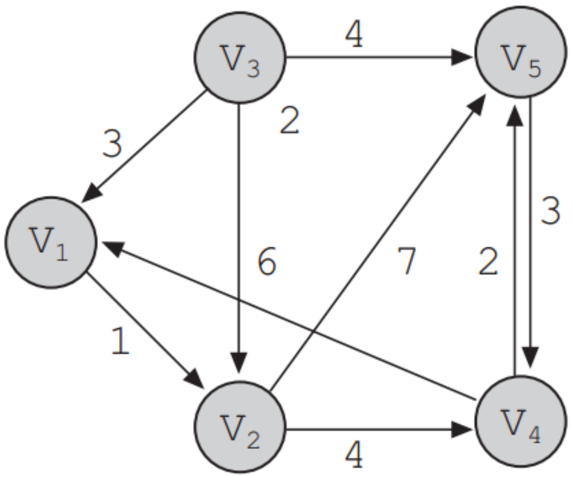
min($D_0[2,5]$, $D_0[2,1]$+$D_0[1,5]$)

V_1	0	1	∞	∞	
V_2	∞	0	∞	4	7
V_3					
V_4					
V_5					

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

i

$\mathbf{D_k[i, j]}$ = minimum(

min($D_0[3,1], D_0[3,1]+D_0[1,1]$)

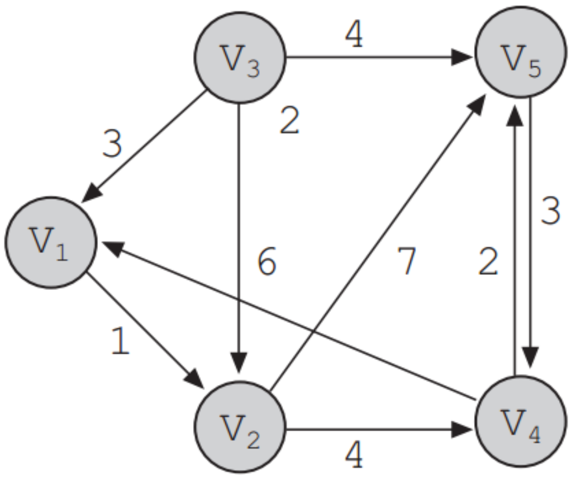
$D_{k-1}[k,j]$)

				V_4	V_5
				∞	∞
V_2		0	∞	4	7
V_3	3				
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3				
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

i

D_k[i, j] = minimum(D_{k-1}
D_{k-1} [
$D_{k-1}[k,j]$)

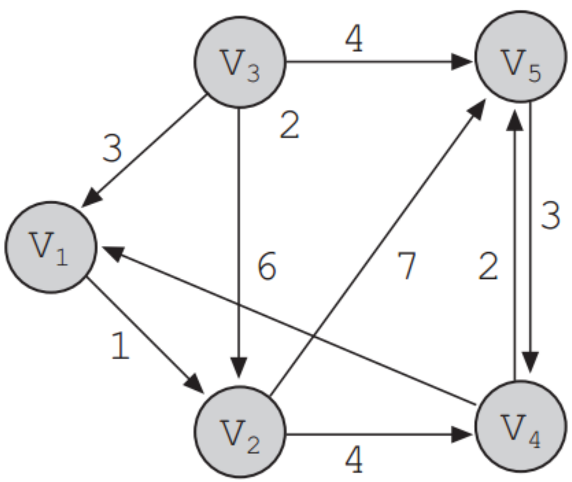
min($D_0[3,2], D_0[3,1]+D_0[1,2]$)

					V_5
					∞
V_2	∞		∞	4	7
V_3	3	2			
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3			
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

i

D_k**[i, j]** = minimum(D_{k-1}
D_{k-1} [
D_{k-1}[k,j])

min(D_0[3,2], D_0[3,1]+D_0[1,2])

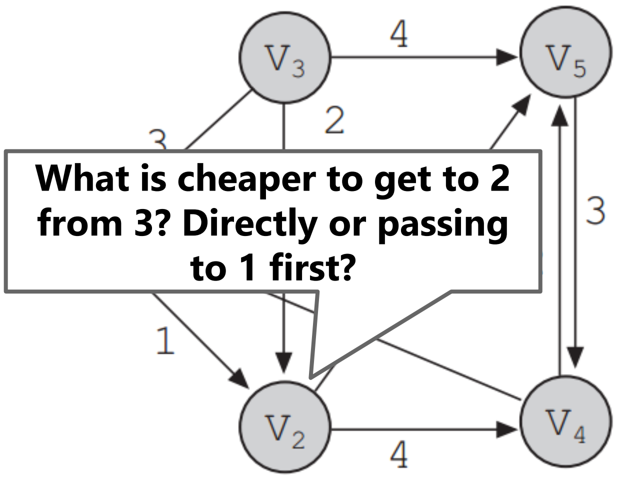
					V_5
					∞
V_2	∞		∞	4	7
V_3	3	2			
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3			
V_4					
V_5					

See the idea?

Graph algorithms

Shortest-path: Floyd



What is cheaper to get to 2 from 3? Directly or passing to 1 first?

$D_k[i, j]$ = minimum(D_{k-1}
D_{k-1} [
$D_{k-1}[k,j]$)

min($D_0[3,2]$, $D_0[3,1]+D_0[1,2]$)

j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

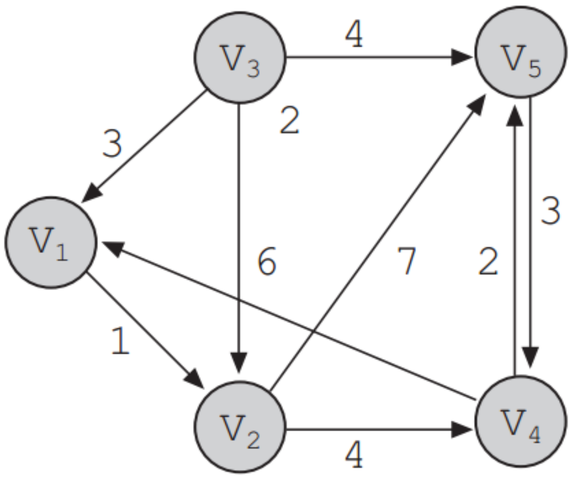
i

					V_5
					∞
V_2	∞		∞	4	7
V_3	3	2			
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3			
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

i

$D_k[i, j]$ = minimum($D_{k-1}[i,j$
$D_{k-1}[i,k$
$D_{k-1}[k,j])$

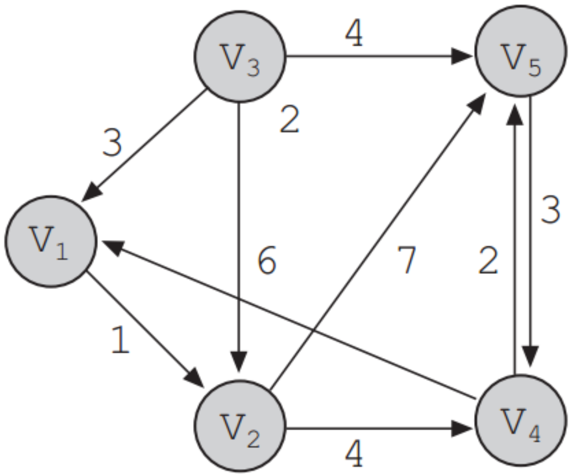
min($D_0[3,3], D_0[3,1]+D_0[1,3]$)

V_2	∞	0		4	7
V_3	3	2	0		
V_4					
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0		
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$,
$D_{k-1}[i,k]$ + $D_{k-1}[k,j]$)

4,1],D_0[4,1]+D_0[1,1])

j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

i

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

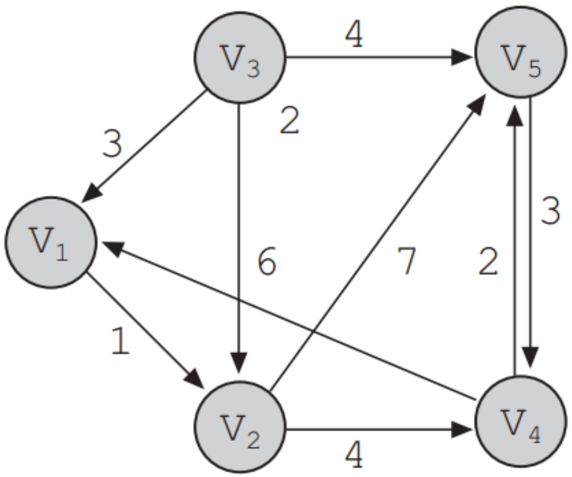
i

D_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6				
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4				
V_5					

Graph algorithms

Shortest-path: Floyd



j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

i

$D_k[i, j]$ = minimum(D_{k-1}
D_{k-1}
D_{k-1}

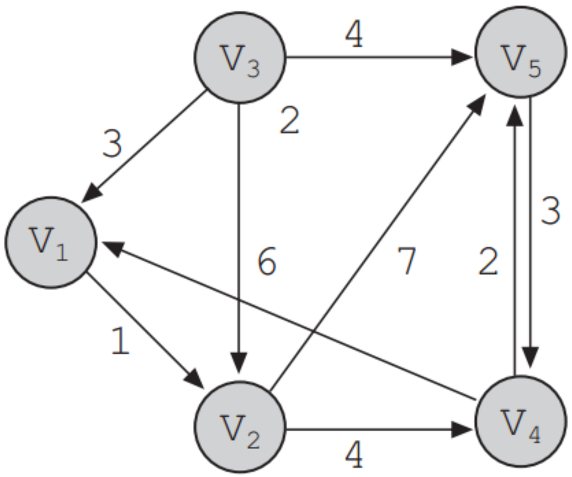
min($D_0[4,2]$, $D_0[4,1]+D_0[1,2]$)

D_1	V_1	V_2	V_3	V_4	V_5
V_1					∞
V_2					7
V_3	3		0	∞	4
V_4	6	7			
V_5					

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	1			
V_5					

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

j

D_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	∞	∞	0	2
V_5	∞	∞	∞	3	0

Q_0	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	0	0	0	4
V_5	0	0	0	5	0

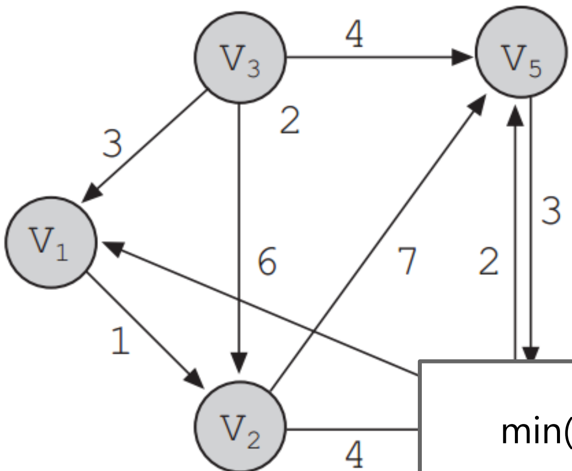
i

D_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	7	∞	0	2
V_5	∞	∞	∞	3	0

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	1	0	0	4
V_5	0	0	0	5	0

Graph algorithms

Shortest-path: Floyd



min($D_1[1,2], D_1[1,2]+D_1[2,2]$)

D_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	7	∞	0	2
V_5	[illegible]	[illegible]	[illegible]	[illegible]	0

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

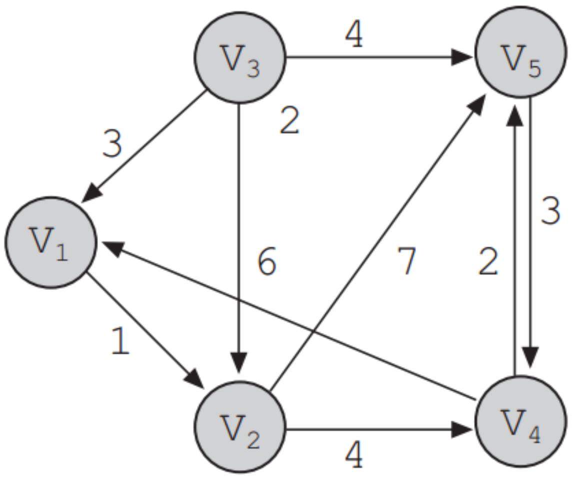
$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j], D_{k-1}[i,k]+ D_{k-1}[k,j])$$

D_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1			
V_2		0			
V_3			0		
V_4				0	
V_5					0

Q_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1			
V_2					
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$,
$D_{k-1}[i,k]+$
$D_{k-1}[k,j]$)

j

D_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	7	∞	0	2

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

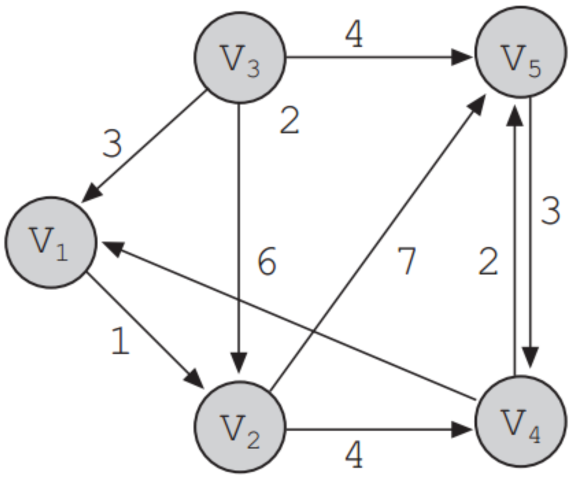
min($D_1[1,4], D_1[1,2]+D_1[2,4]$)

D_2	V_1	V_2	V_3		V_5
V_1	0	1	∞	5	
V_2		0			
V_3			0		
V_4				0	
V_5					0

Q_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	
V_2					
V_3					
V_4					
V_5					

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$, $D_{k-1}[i,k] + D_{k-1}[k,j]$)

D_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	∞	∞
V_2	∞	0	∞	4	7
V_3	3	2	0	∞	4
V_4	6	7	∞	0	2
V_5	∞	∞	∞	3	0

j

Q_1	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	0
V_2	0	0	0	2	2
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

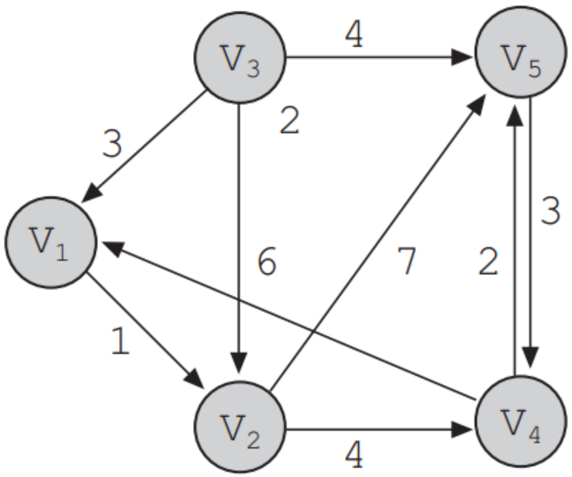
i

D_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	8
V_2	∞	0	∞	4	7
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	∞	∞	∞	3	0

Q_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	2
V_2	0	0	0	2	2
V_3	3	3	0	2	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

j

D_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	8
V_2	∞	0	∞	4	7
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	∞	∞	∞	3	0

Q_2	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	0
V_2	0	0	0	2	2
V_3	3	3	0	2	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

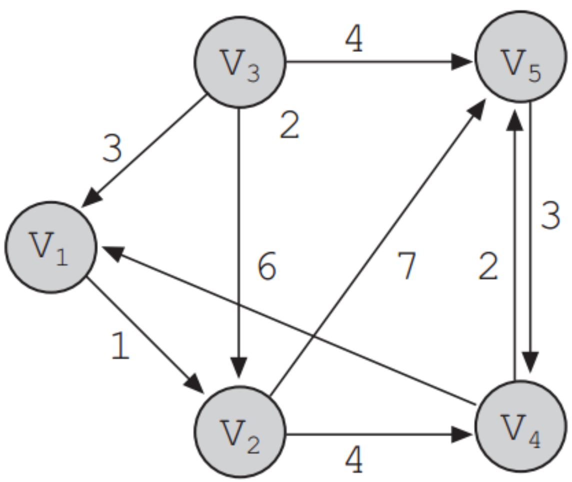
i

D_3	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	8
V_2	∞	0	∞	4	7
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	∞	∞	∞	3	0

Q_3	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	0
V_2	0	0	0	2	2
V_3	3	3	0	2	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

j

D_3	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	8
V_2	∞	0	∞	4	7
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	∞	∞	∞	3	0

Q_3	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	0
V_2	0	0	0	2	2
V_3	3	3	0	2	3
V_4	4	1	4	0	4
V_5	0	0	0	5	0

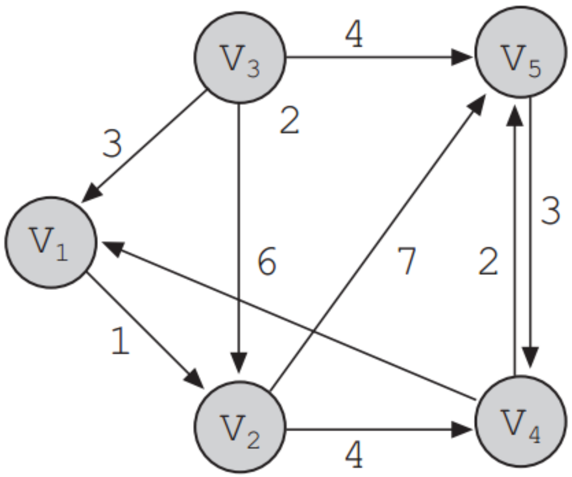
i

D_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	2	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

Graph algorithms

Shortest-path: Floyd



j

D_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

i

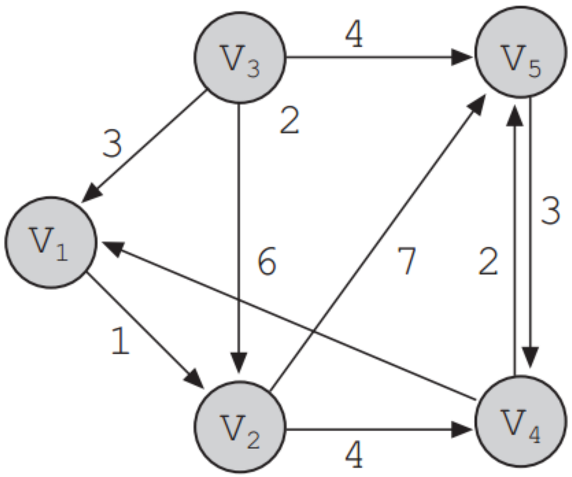
D_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	2	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

$$\mathbf{D_k[i, j]} = \text{minimum}(D_{k-1}[i,j],\; D_{k-1}[i,k] + D_{k-1}[k,j])$$

Graph algorithms

Shortest-path: Floyd



$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\ D_{k-1}[i,k] + D_{k-1}[k,j])$$

What's the shortest path from v_1 to v_4?

j

D_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

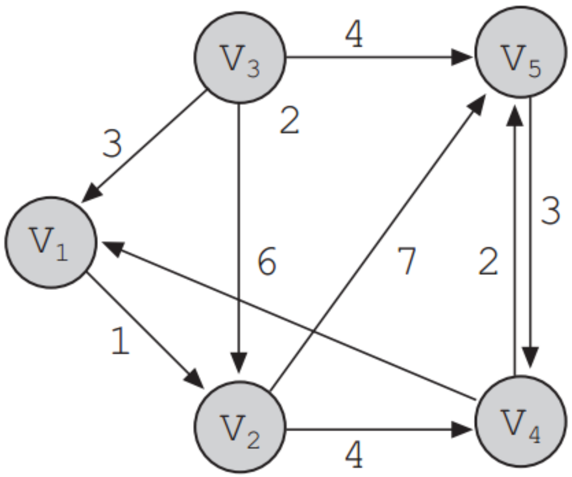
i

D_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

Graph algorithms

Shortest-path: Floyd



D_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

$$D_k[i, j] = \text{minimum}(D_{k-1}[i,j],\; D_{k-1}[i,k] + D_{k-1}[k,j])$$

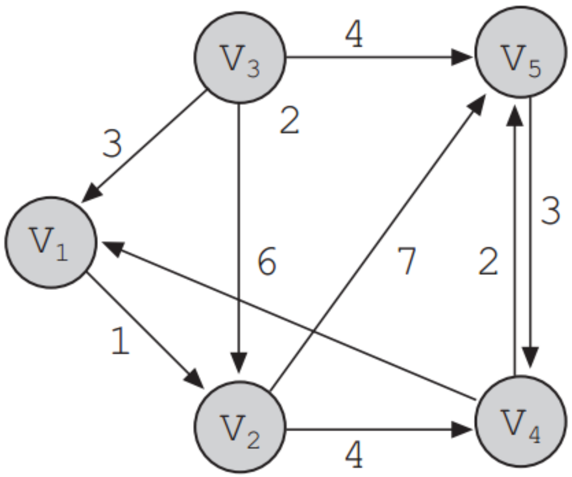
D_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

What's the shortest path from v_1 to v_4?

Graph algorithms

Shortest-path: Floyd



$D_k[i, j]$ = minimum($D_{k-1}[i,j]$, $D_{k-1}[i,k] + D_{k-1}[k,j]$)

What's the shortest path from v_1 to v_4?
R/$\{v_1, v_2, v_4\}$

D_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	7
V_3	3	2	0	6	6
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_4	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

D_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	∞	5	7
V_2	10	0	∞	4	6
V_3	3	2	0	6	4
V_4	6	7	∞	0	2
V_5	9	10	∞	3	0

Q_5	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	2	4
V_2	4	0	0	2	4
V_3	3	3	0	0	3
V_4	4	1	4	0	4
V_5	4	4	0	5	0

Graph algorithms

Warshall

- ➔ Calculates the **path matrix** *P* (also called transitive closure) of a graph *G* of n vertices, represented by its adjacency matrix *A*.
- ➔ Defines a sequence of matrices $n_x n$ P_0, P_1, P_2, P_3, … P_n
- ➔ Similar to Floyd algorithm

Graph algorithms

Warshall

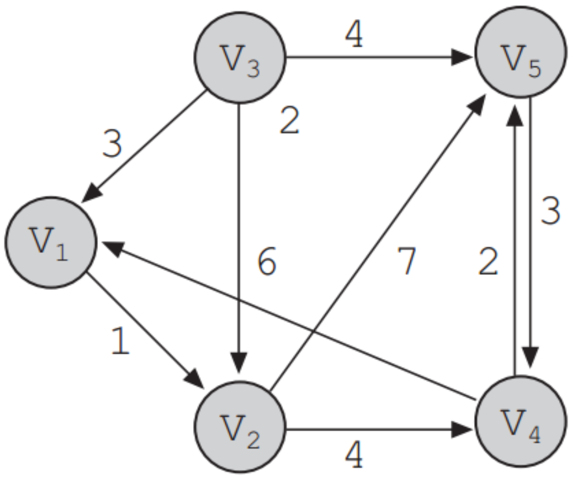
$$P_0[i, j] = \begin{cases} 1 \text{ if there is an arc from } v_i \text{ to } v_j \\ 0 \text{ if there is no arc} \end{cases}$$

$$P_1[i, j] = \begin{cases} 1 \text{ if there is an arc from } v_i \text{ to } v_j \text{ that does not go through} \\ \text{other vertex different of } v_1 \\ 0 \text{ if there is no arc} \end{cases}$$

$$P_2[i, j] = \begin{cases} 1 \text{ if there is an arc from } v_i \text{ to } v_j \text{ that does not go through} \\ \text{other vertex different of } v_1 \ldots v_2 \\ 0 \text{ if there is no arc} \end{cases}$$

$$P_n[i, j] = \begin{cases} 1 \text{ if there is an arc from } v_i \text{ to } v_j \text{ that does not go through} \\ \text{other vertex different of } v_1 \ldots v_n \\ 0 \text{ if there is no arc} \end{cases}$$

Graph algorithms
Warshall



j

i

P_0	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	0	0
V_2	0	1	0	1	1
V_3	1	1	1	0	1
V_4	1	0	0	1	1
V_5	0	0	0	1	1

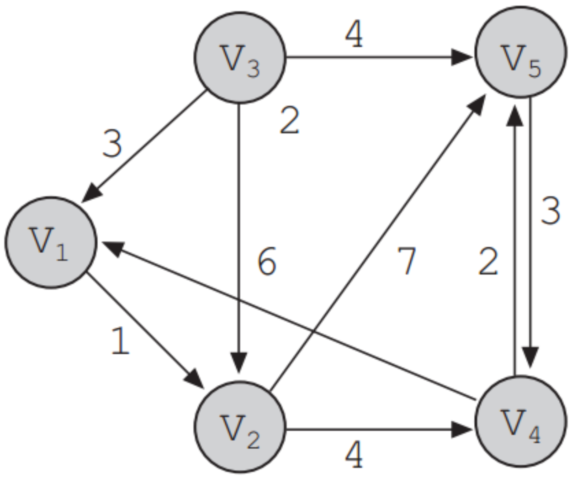
P_1	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	0	0
V_2	0	1	0	1	1
V_3	1	1	1	0	1
V_4	1	1	0	1	1
V_5	0	0	0	1	1

P_2	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	1
V_2	0	1	0	1	1
V_3	1	1	1	1	1
V_4	1	1	0	1	1
V_5	0	0	0	1	1

P_3	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	1
V_2	0	1	0	1	1
V_3	1	1	1	1	1
V_4	1	1	0	1	1
V_5	0	0	0	1	1

Graph algorithms

Warshall



j

i

P_2	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	1
V_2	0	1	0	1	1
V_3	1	1	1	1	1
V_4	1	1	0	1	1
V_5	0	0	0	1	1

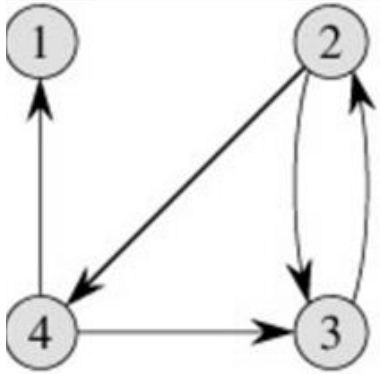
P_3	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	1
V_2	0	1	0	1	1
V_3	1	1	1	1	1
V_4	1	1	0	1	1
V_5	0	0	0	1	1

P_4	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	1
V_2	1	1	0	1	1
V_3	1	1	1	1	1
V_4	1	1	0	1	1
V_5	1	1	0	1	1

P_5	V_1	V_2	V_3	V_4	V_5
V_1	1	1	0	1	1
V_2	1	1	0	1	1
V_3	1	1	1	1	1
V_4	1	1	0	1	1
V_5	1	1	0	1	1

Graph algorithms

Warshall



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Minimal spanning tree

- ➔ A undirected graph is used to model symmetrical relations between vertices of the graph. Any arc (v,w) of an undirected graph is the same as the arc of (w,v).

- ➔ A common task is to determine if for any pair of vertices, there is a path that connects them, in other words, **if it is a connected graph**

Minimal spanning tree

- ➔ A minimal spanning tree is a subset of the graph that covers all vertices and which edges have a sum of the minimal weights
  - Applied on networks

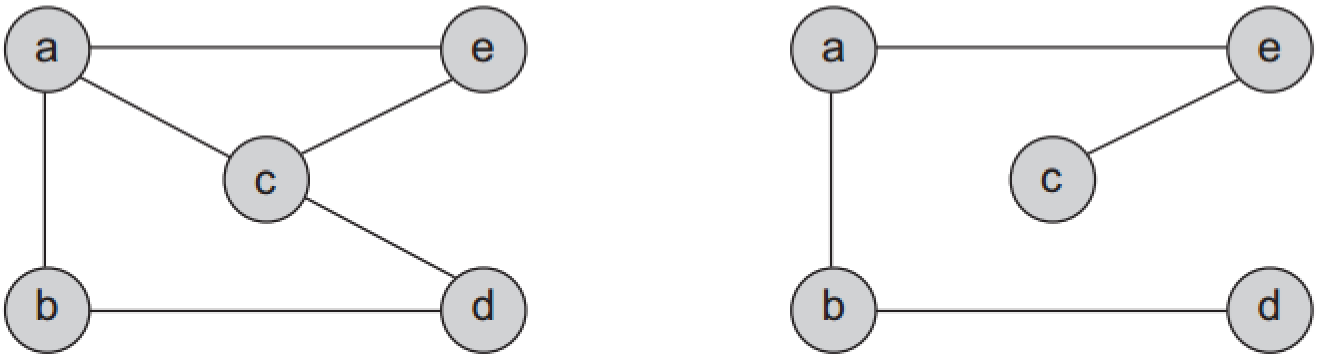
- ➔ A **tree** is a subset of the graph that is connected and has no cycles
  - If it has n vertices, then it has n-1 edges
  - There exists a unique path between any two vertices of a tree
  - If an edge is added it results in a cycle

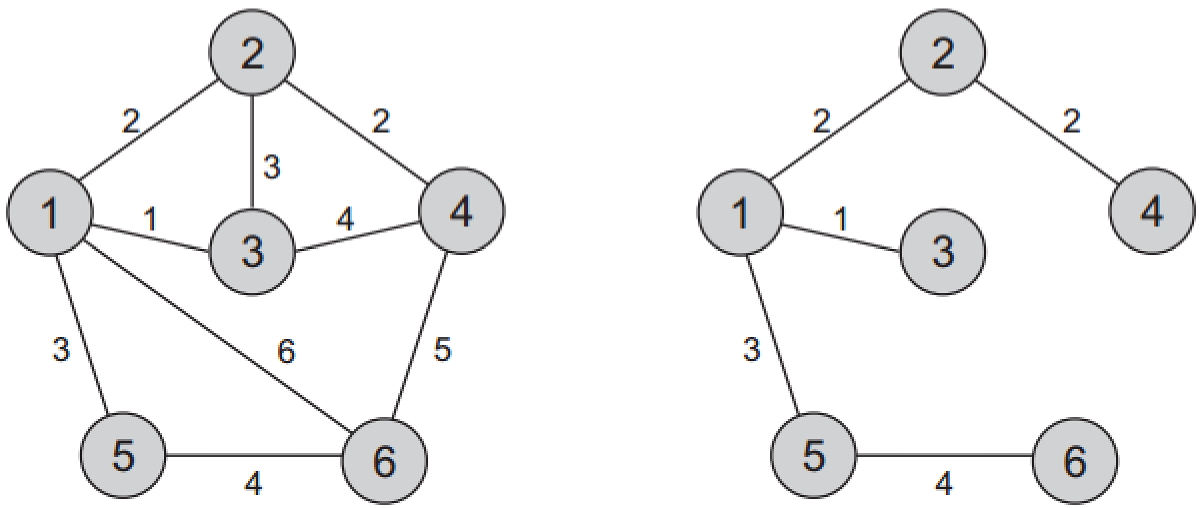
Minimal spanning tree

- ➔ If all vertices are on the tree, then it is a connected graph
- ➔ Given undirected graph, find the minimal spanning tree

Graph algorithms

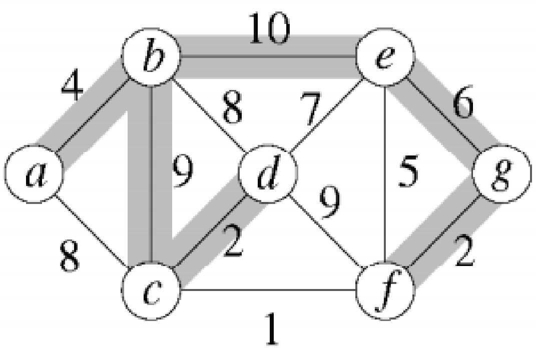
Minimal spanning tree



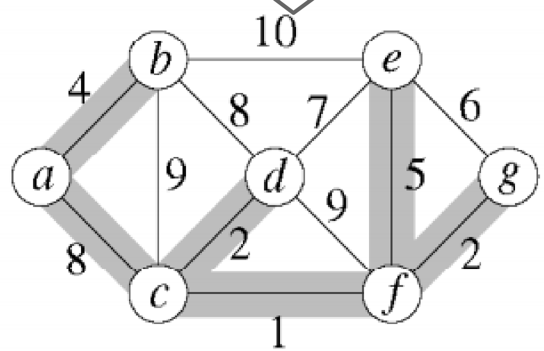


Minimal spanning tree

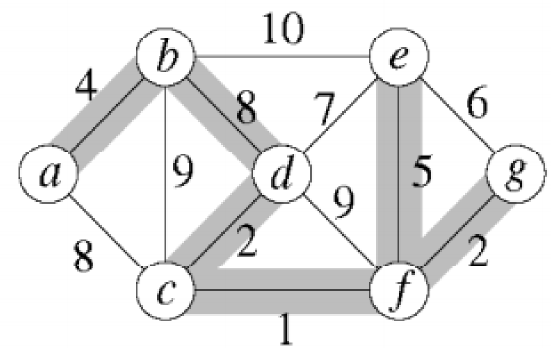
The same graph can have several spanning trees, but not all are the minimum



Cost = 33



Cost = 22



Cost = 22

Minimal spanning tree

➔ How to do this?

- Prim algorithm
- Kruskal algorithm

Prim

- ➔ Grows the tree in successive stages. In each stage, one node is picked as the root, and we add an edge and the associated vertex to the tree

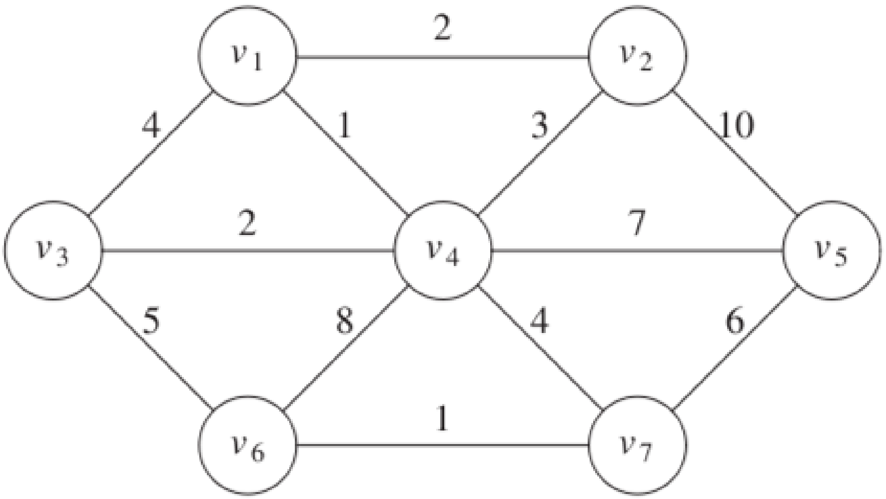
- ➔ At any point we have a set of vertices that have been already included in the tree.

Prim

- ➔ The algorithm finds a new vertex to add to the tree by choosing the edge (u,v) such as the cost of (u,v) is the smallest among all edges where u is in the tree and v not

Graph algorithms

Prim

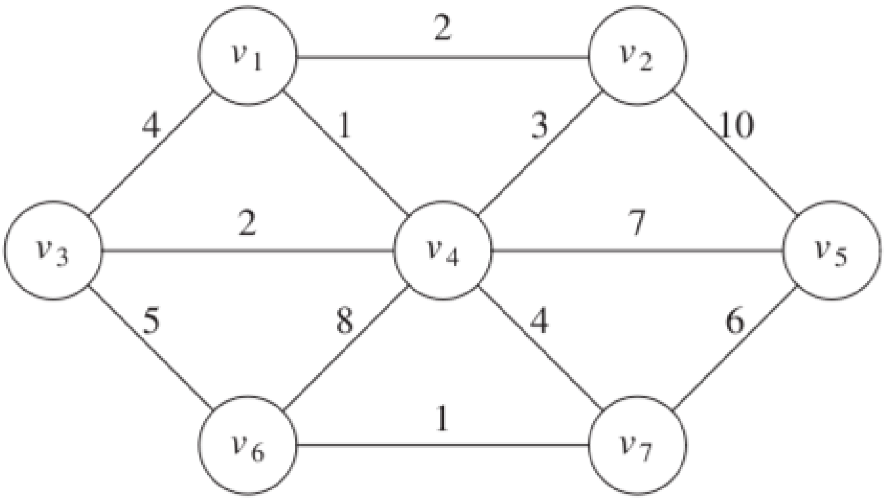


Minimum weight connecting to a known node

v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

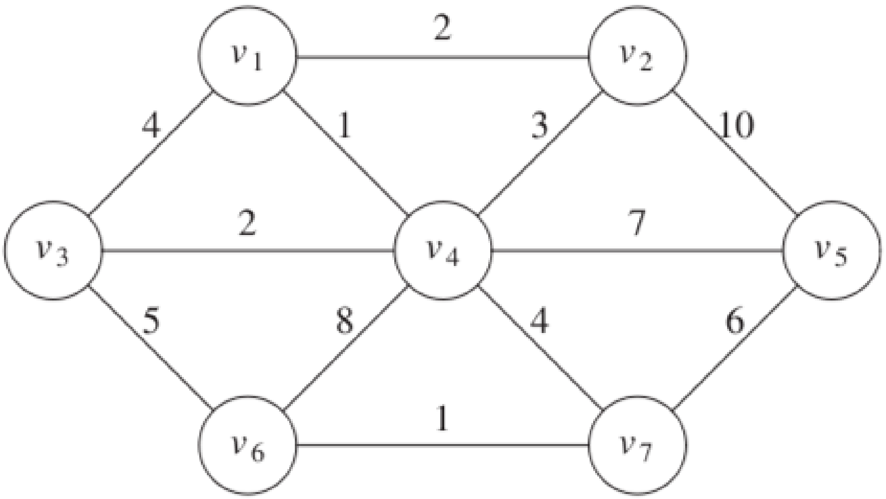
Graph algorithms

Prim



v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

Graph algorithms
Prim

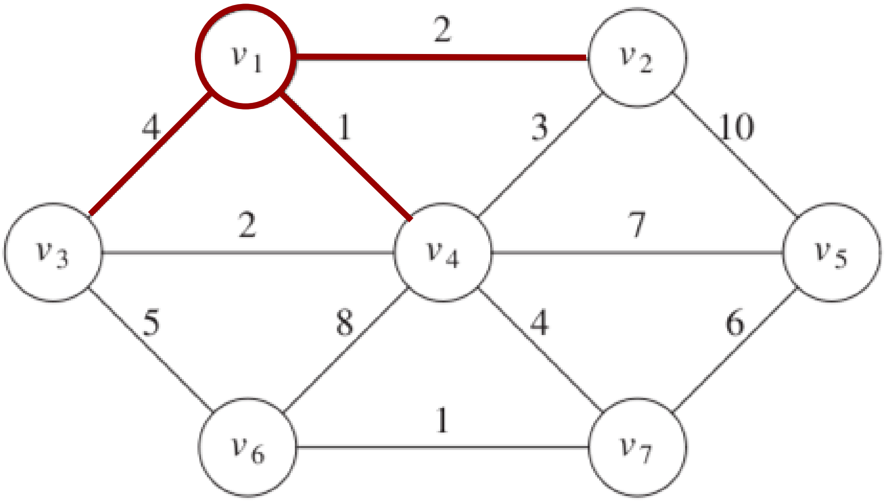


Last vertex that cause a change in d_v

v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

Graph algorithms

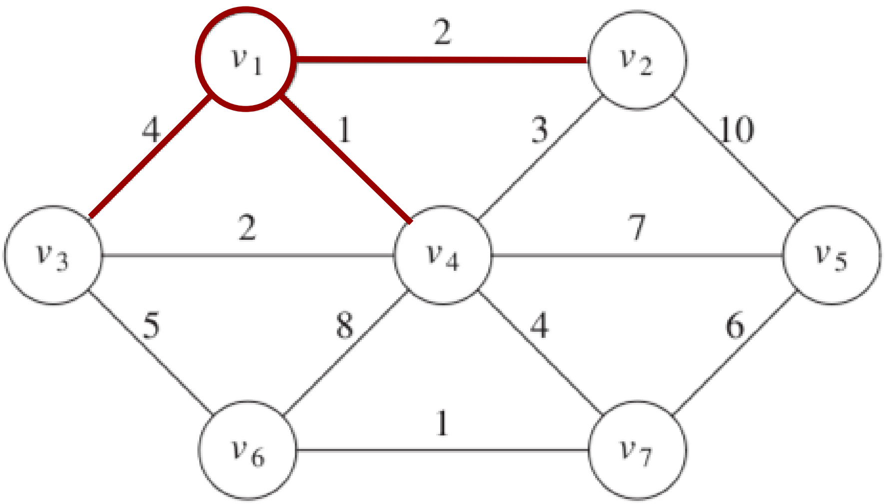
Prim



v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	4	v_1
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

Graph algorithms

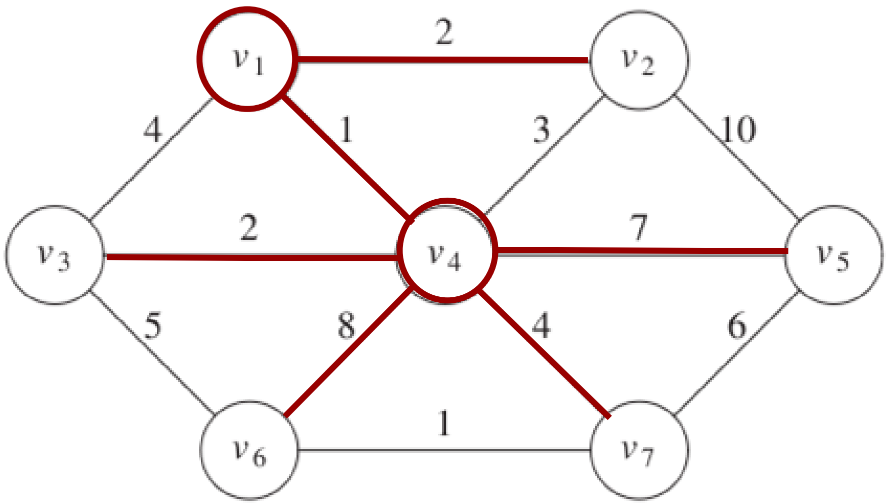
Prim



Next is the minimum

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	4	v_1
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

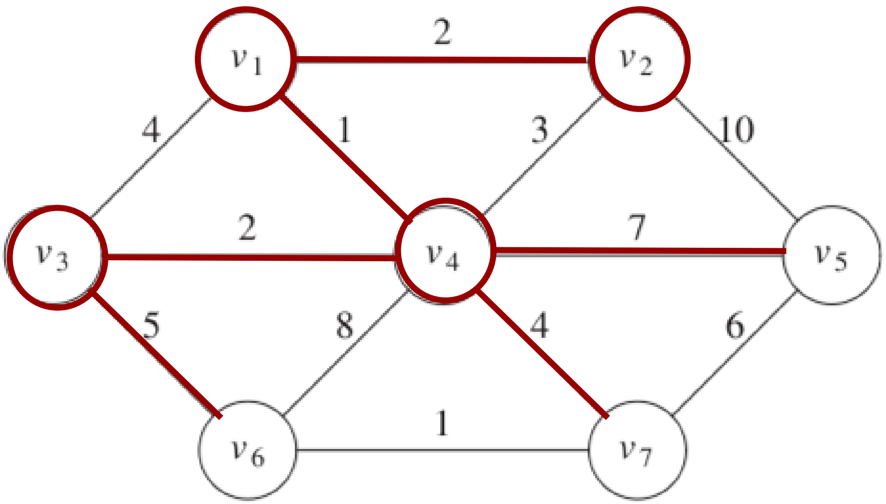
Graph algorithms
Prim



v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	2	v_4
v_4	T	1	v_1
v_5	F	7	v_4
v_6	F	8	v_4
v_7	F	4	v_4

Graph algorithms

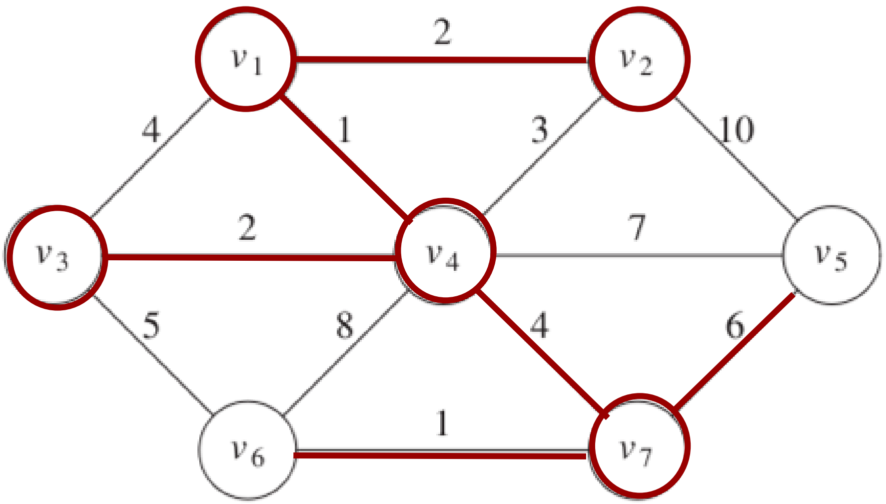
Prim



v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	F	7	v_4
v_6	F	5	v_3
v_7	F	4	v_4

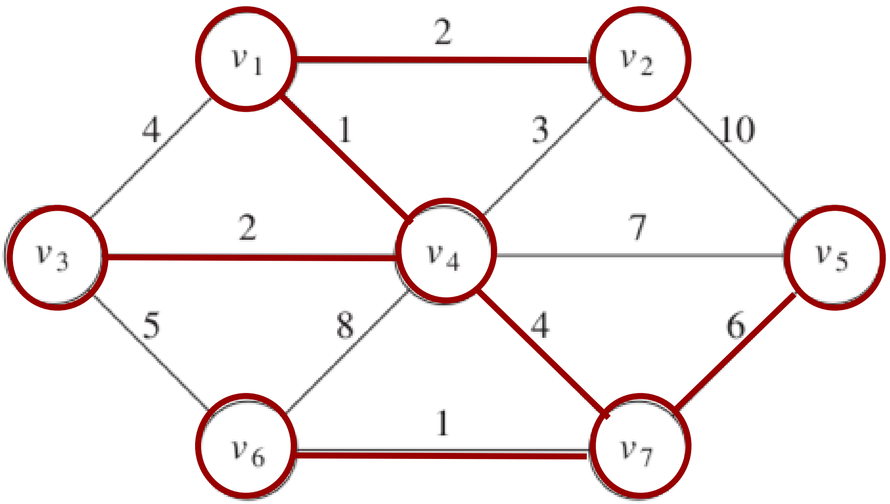
Graph algorithms

Prim



v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	F	6	v_7
v_6	F	1	v_7
v_7	T	4	v_4

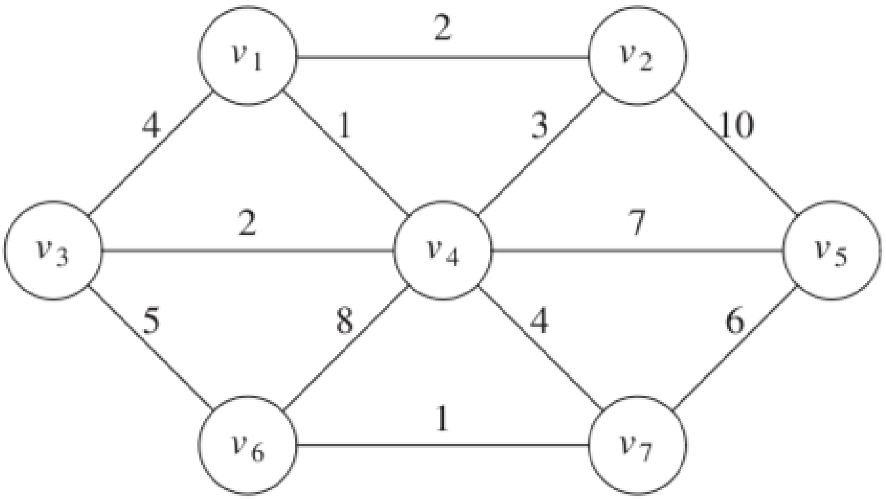
Graph algorithms
Prim

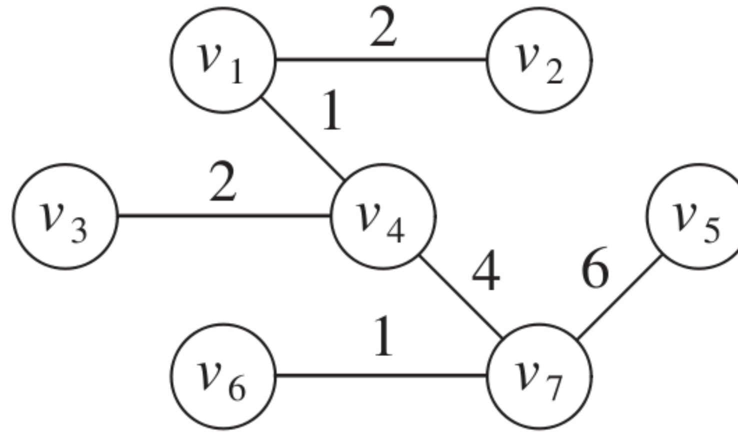


v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	2	v_4
v_4	T	1	v_1
v_5	T	6	v_7
v_6	T	1	v_7
v_7	T	4	v_4

Graph algorithms

Prim





Graph algorithms
Kruskal

➔ Selects edges if order of smallest weight and accept an edge if does not cause a cycle

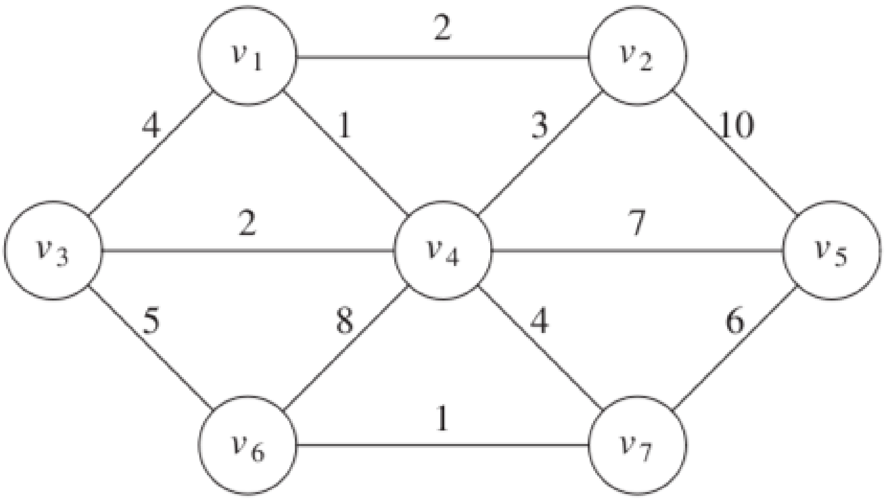
➔ Maintains a forest (collection of trees). Initially all are single node trees. Adding an edge merges two trees into one

➔ If u and v are in the same set the edge (u,v) is rejected, because adding it would cause a cycle

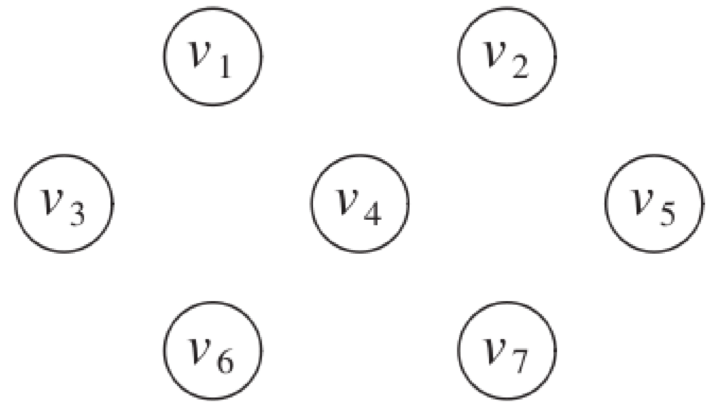
Kruskal

- ➔ u and v are in the same set if they are connected

Kruskal

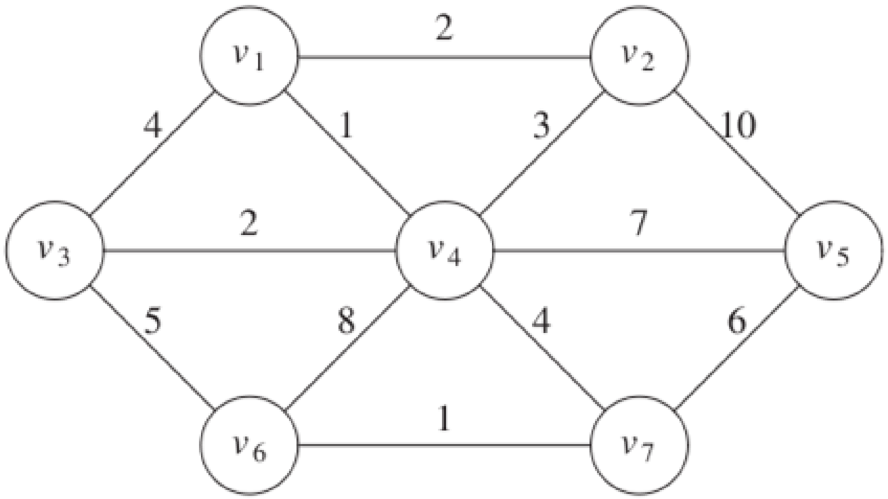


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

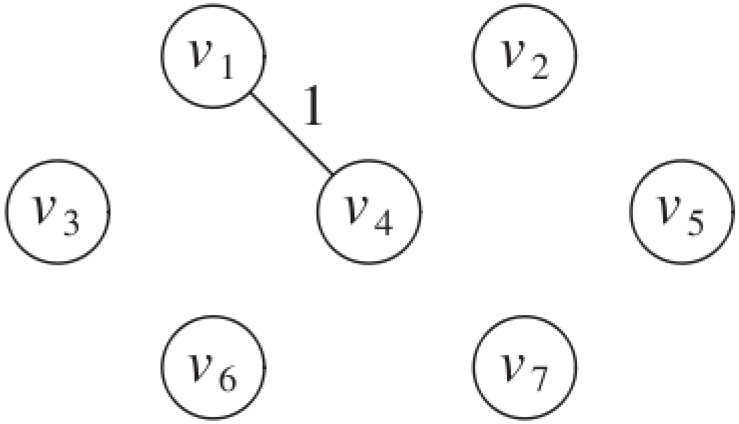


Graph algorithms

Kruskal

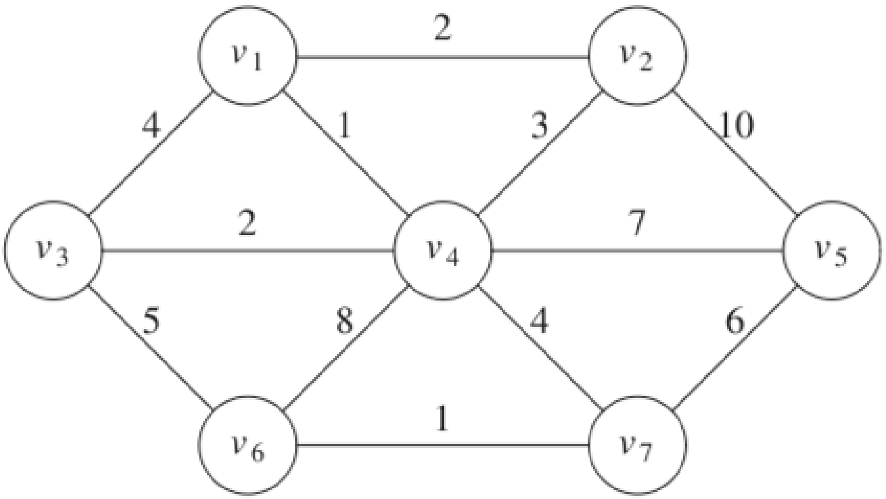


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

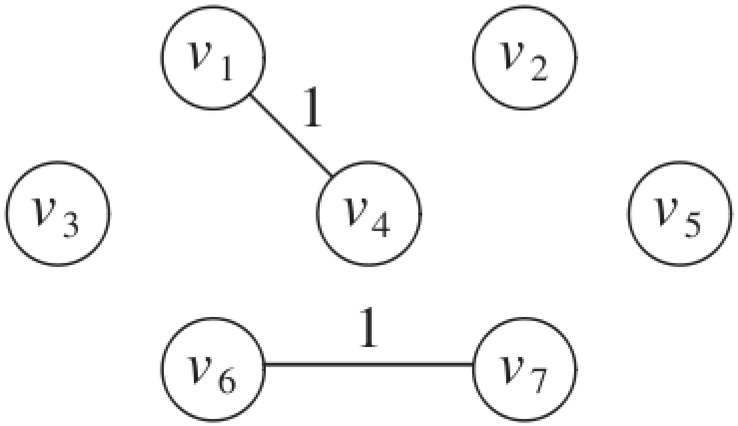


Graph algorithms

Kruskal

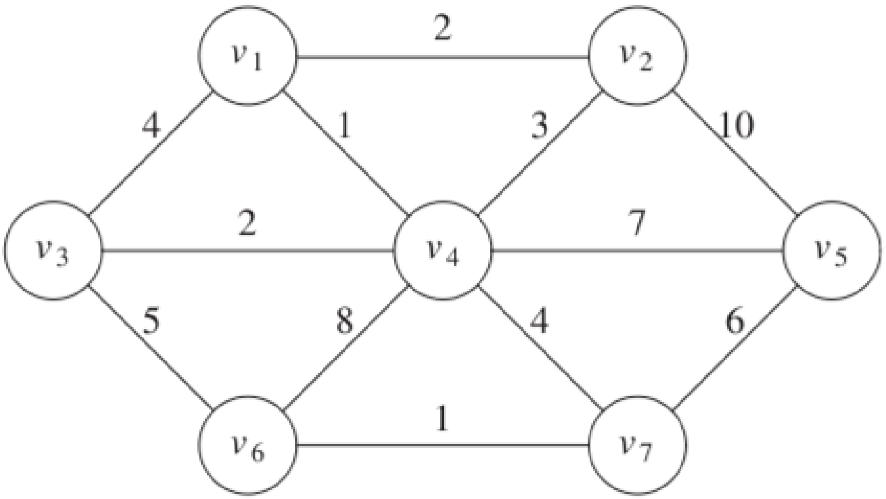


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

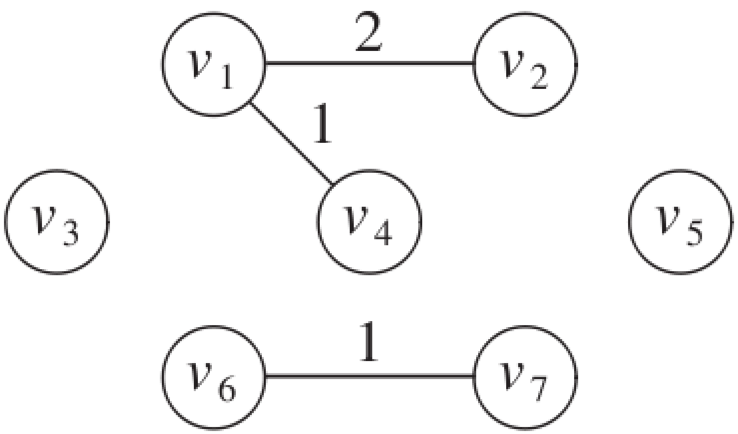


Graph algorithms

Kruskal

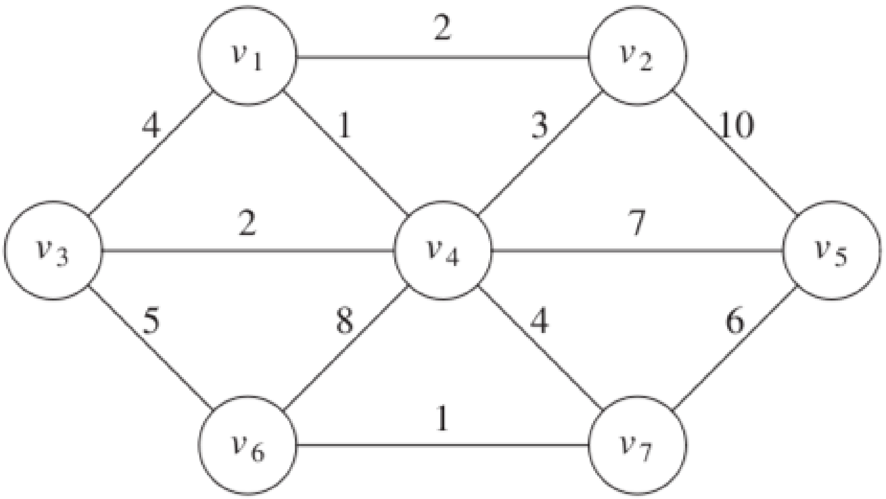


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

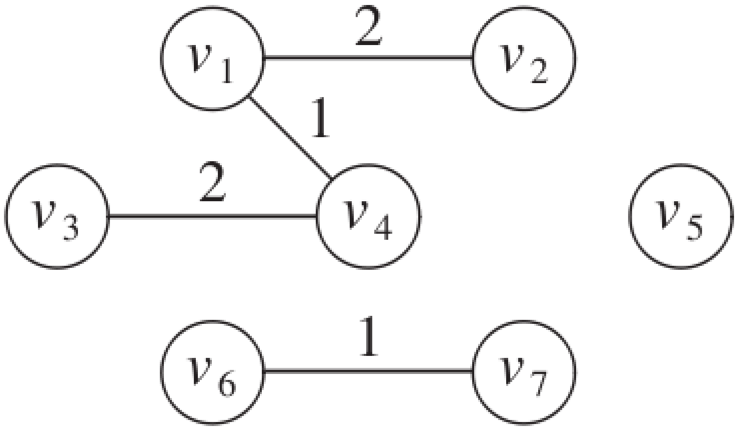


Graph algorithms

Kruskal

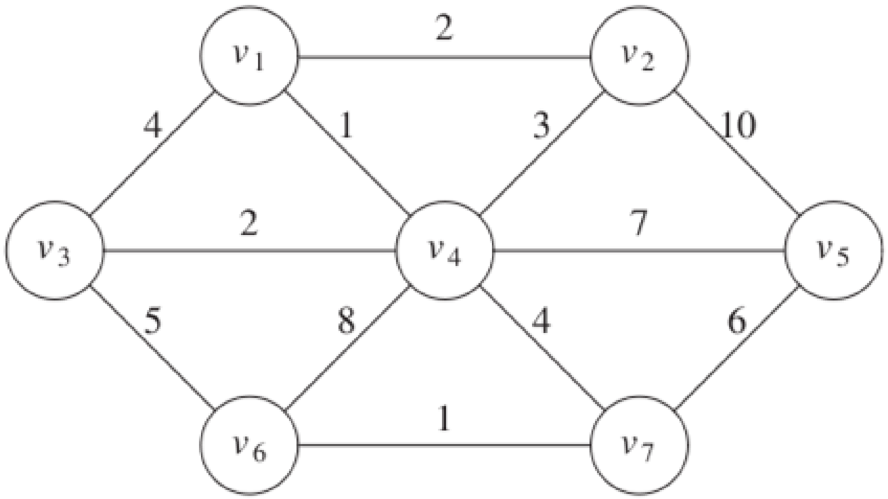


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

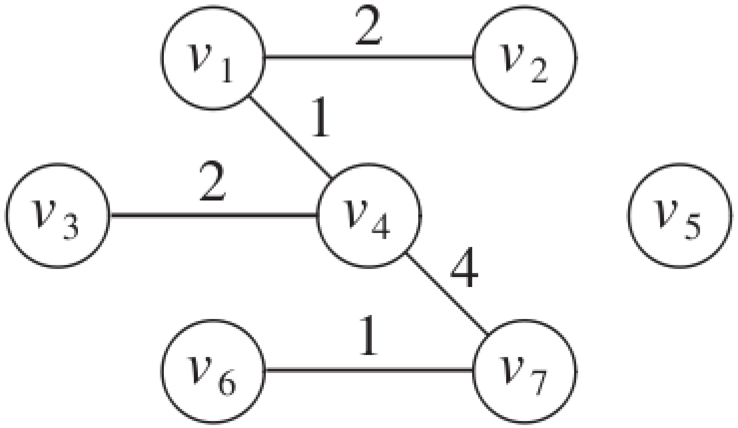


Graph algorithms

Kruskal

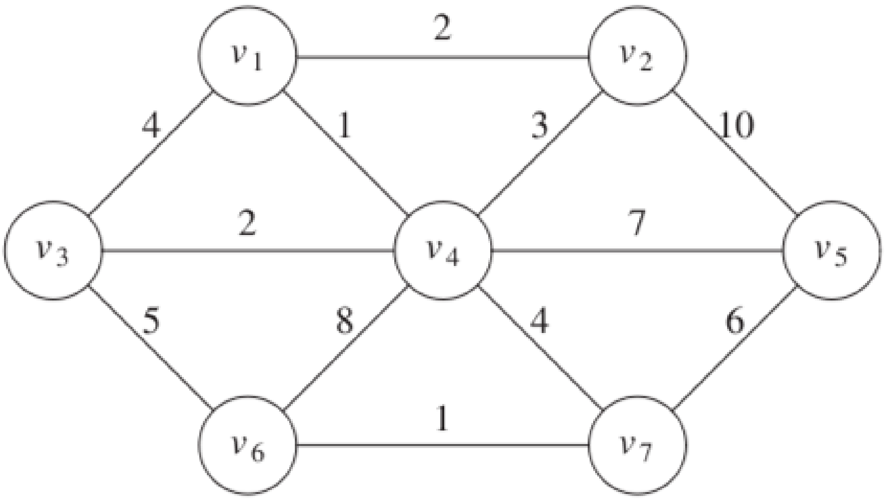


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

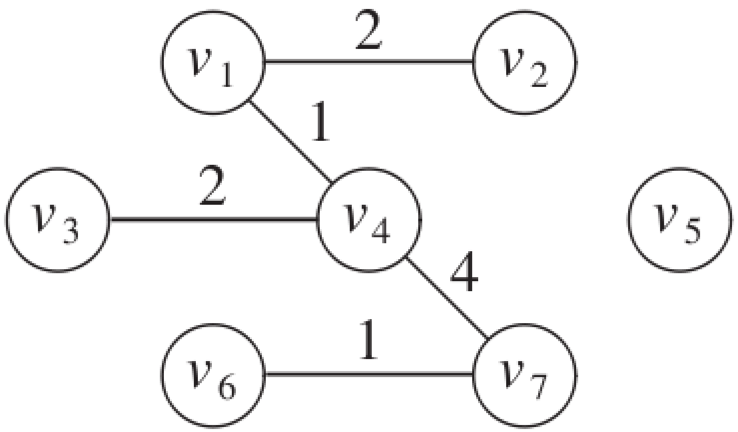


Graph algorithms

Kruskal

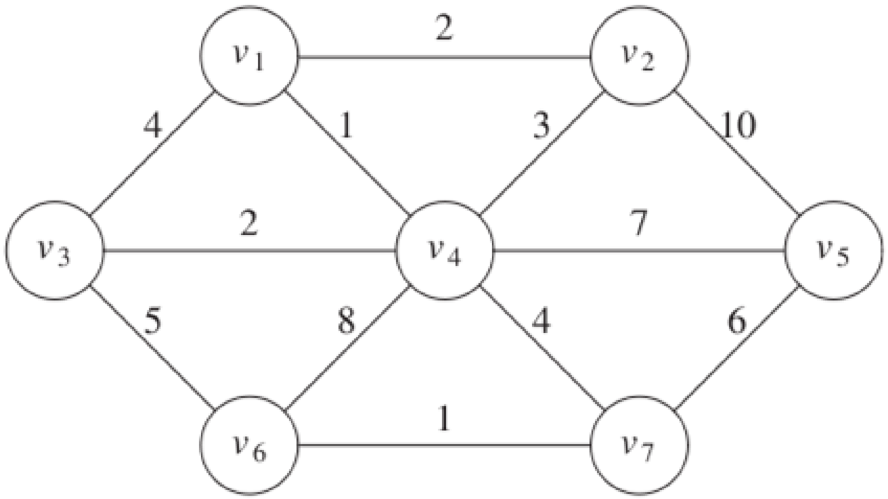


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

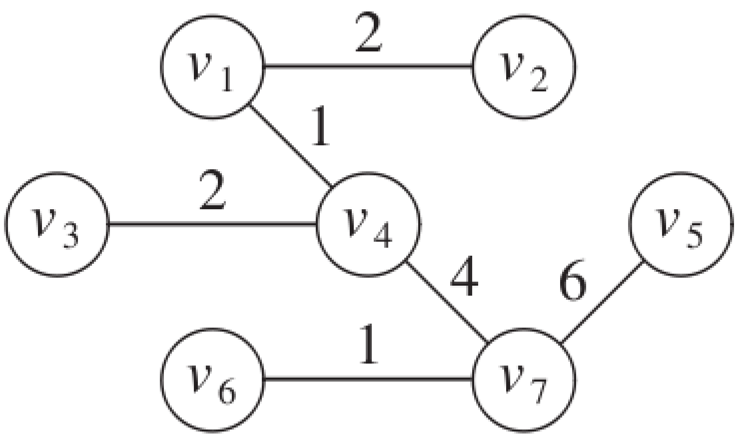


Graph algorithms

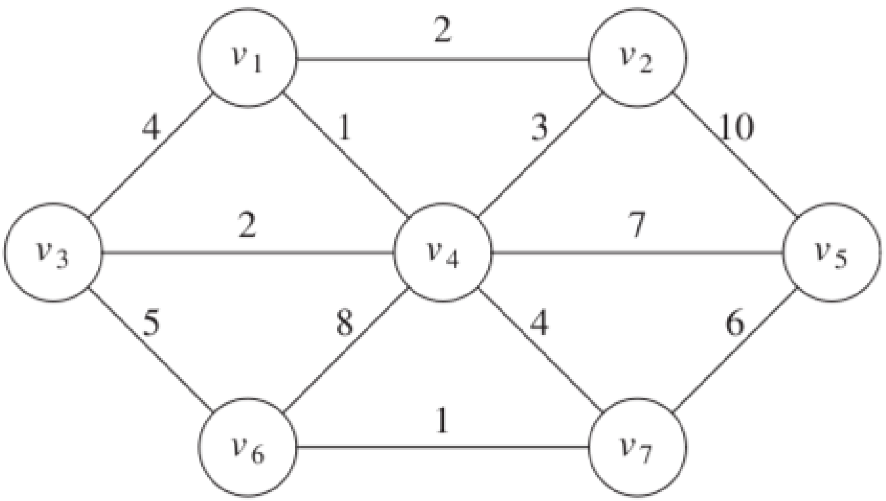
Kruskal



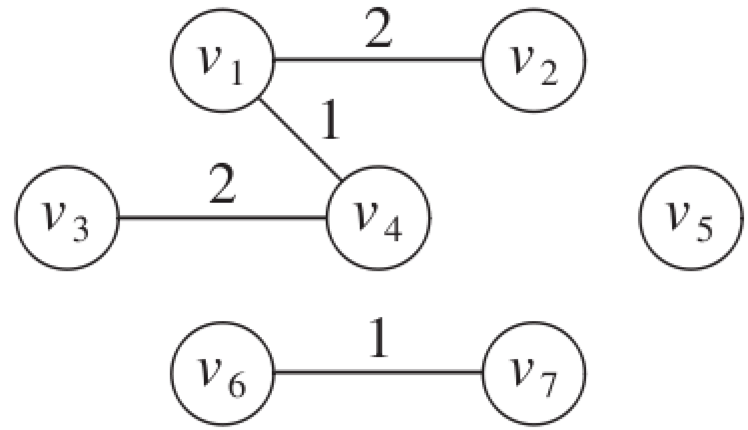
Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted



Graph algorithms
Kruskal

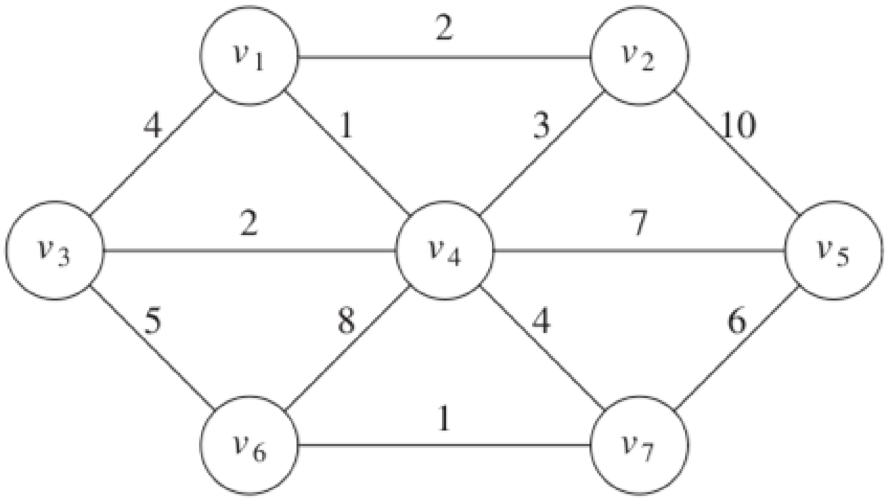


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

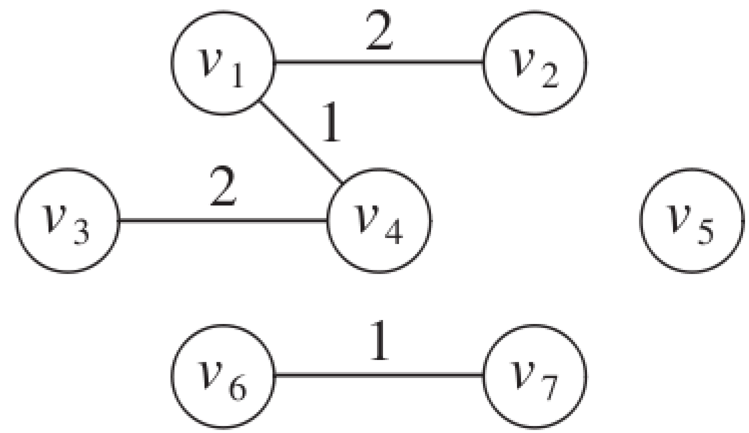


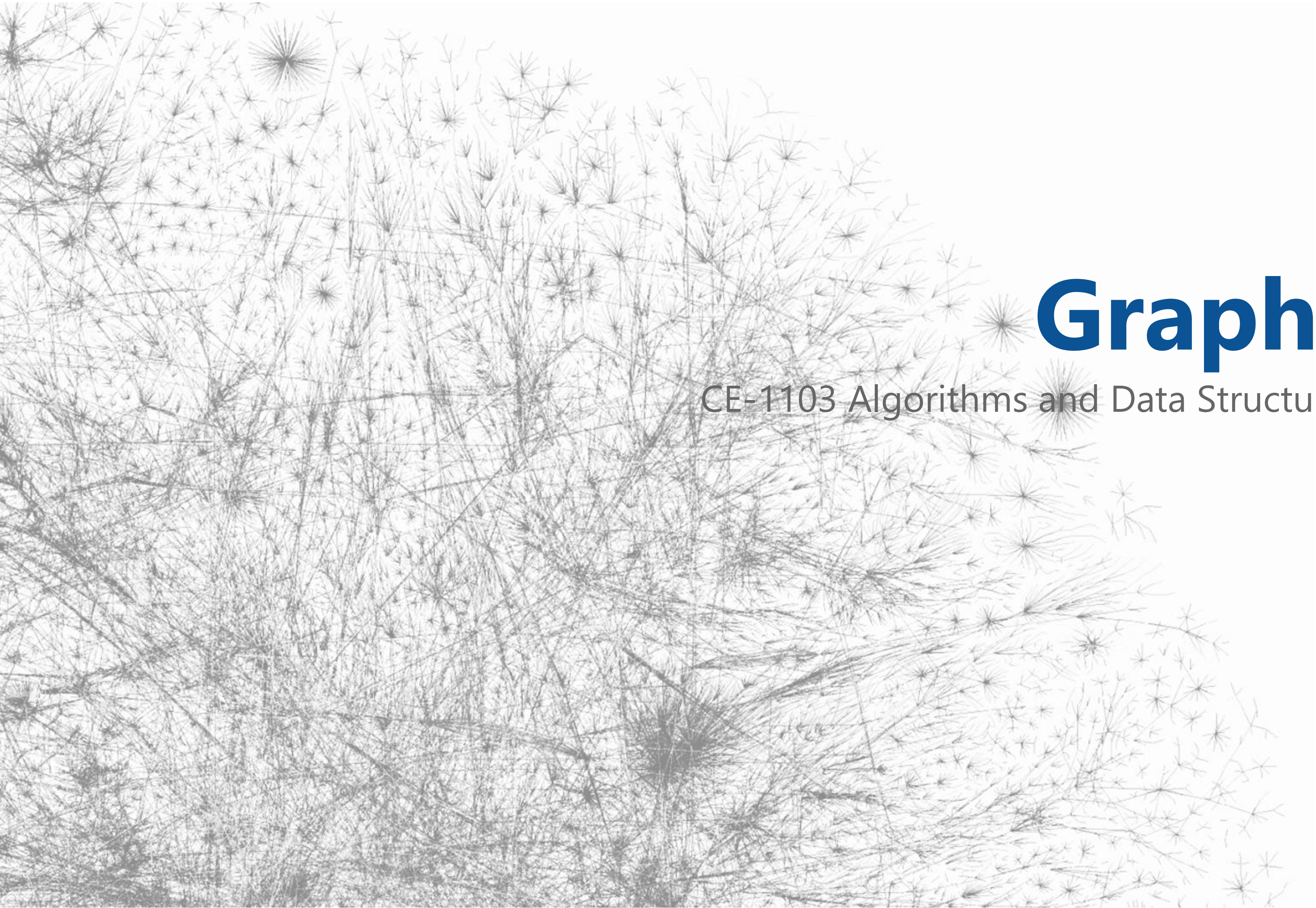
Graph algorithms

Kruskal



Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted





Graphs

CE-1103 Algorithms and Data Structures