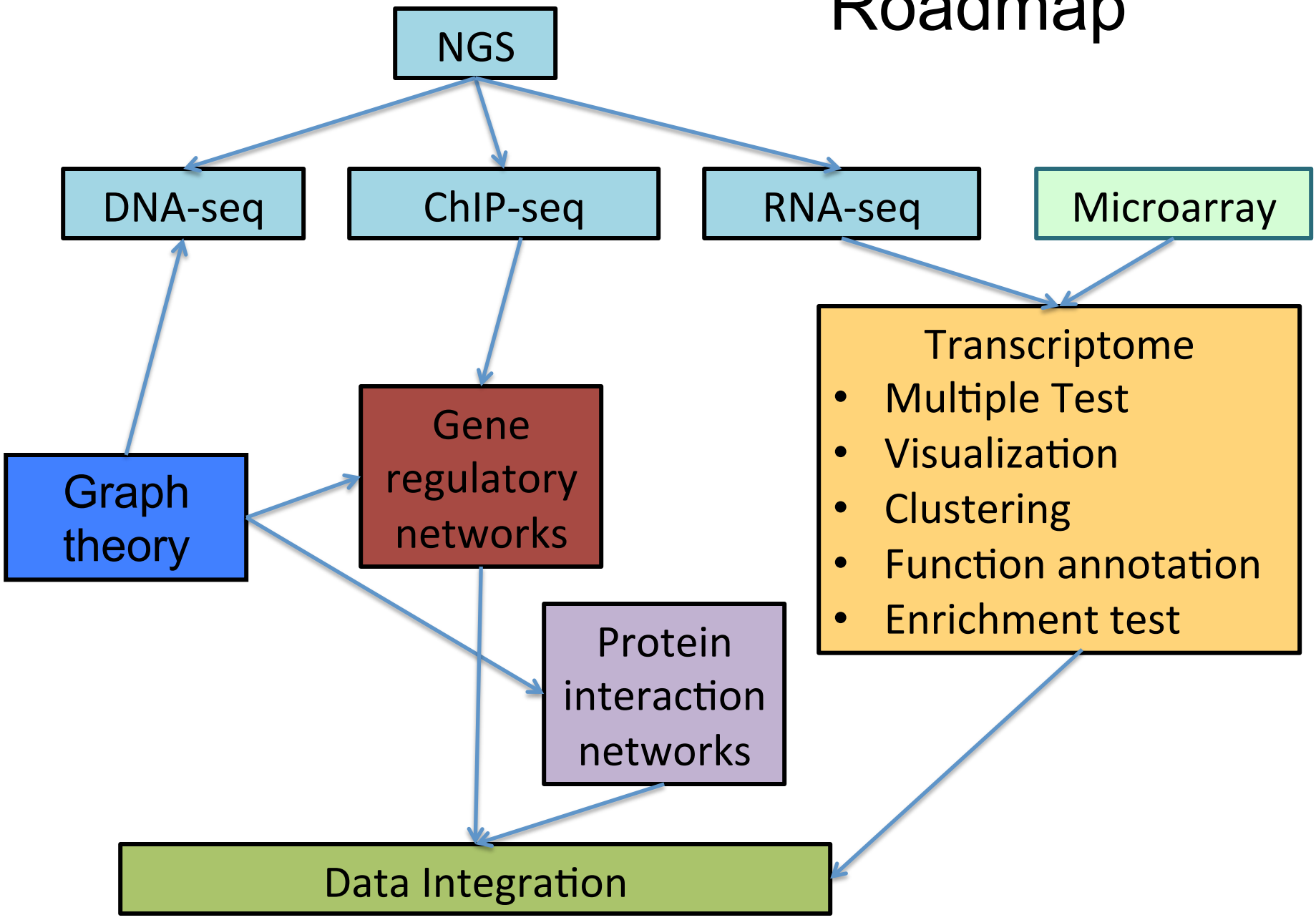
Graph Theory

Lecture 1

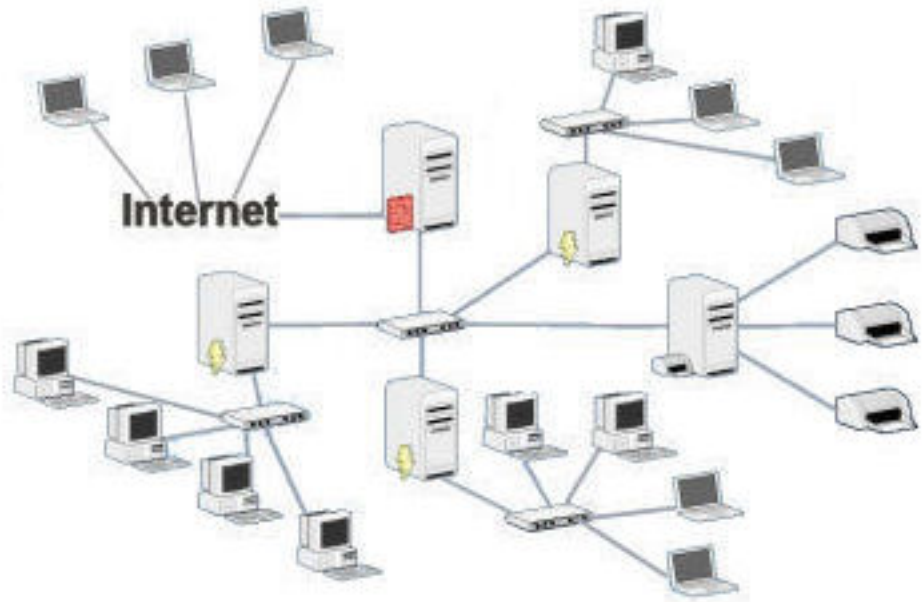
Roadmap

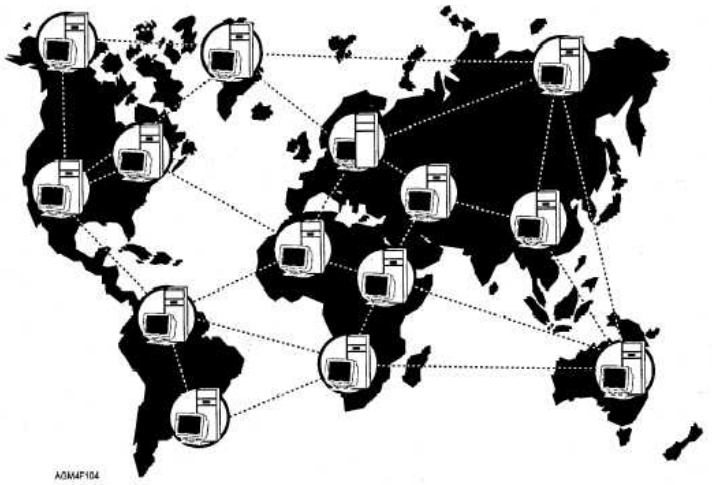

NGS
DNA-seq
ChIP-seq
RNA-seq
Microarray
Transcriptome
• Multiple Test
• Visualization
• Clustering
• Function annotation
• Enrichment test
Graph theory
Gene regulatory networks
Protein interaction networks
Data Integration

Many complicate systems have an underlying network topology

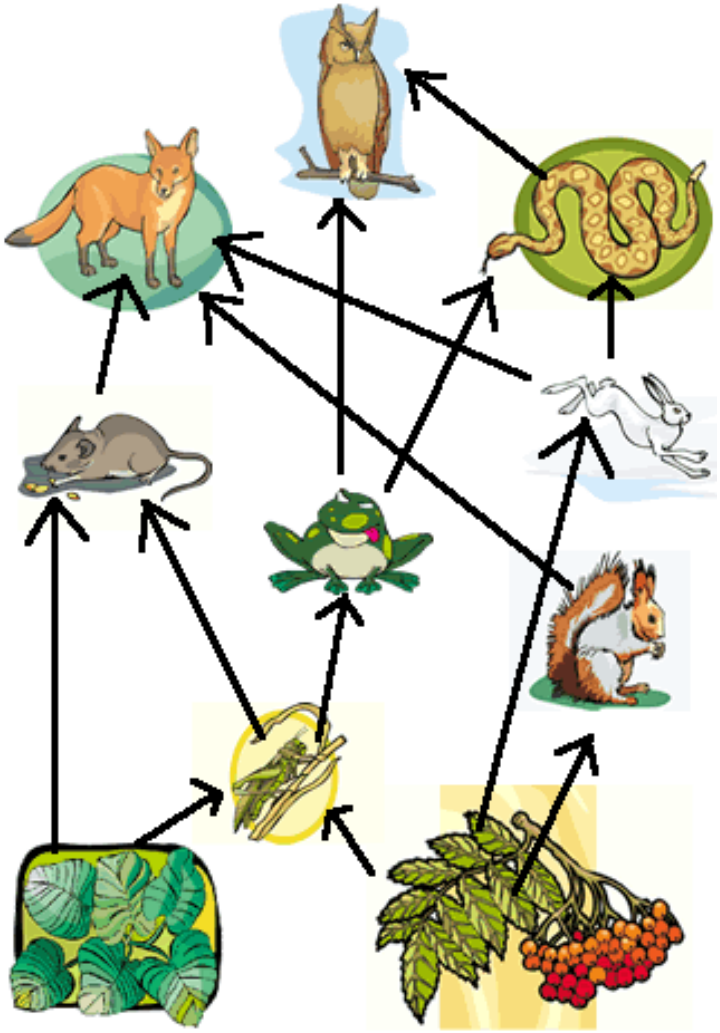
- Computer networks
- Social networks
- Biological networks
  - Food webs (chains)
  - Gene networks
  - Protein interaction networks
  - Signal transduction networks

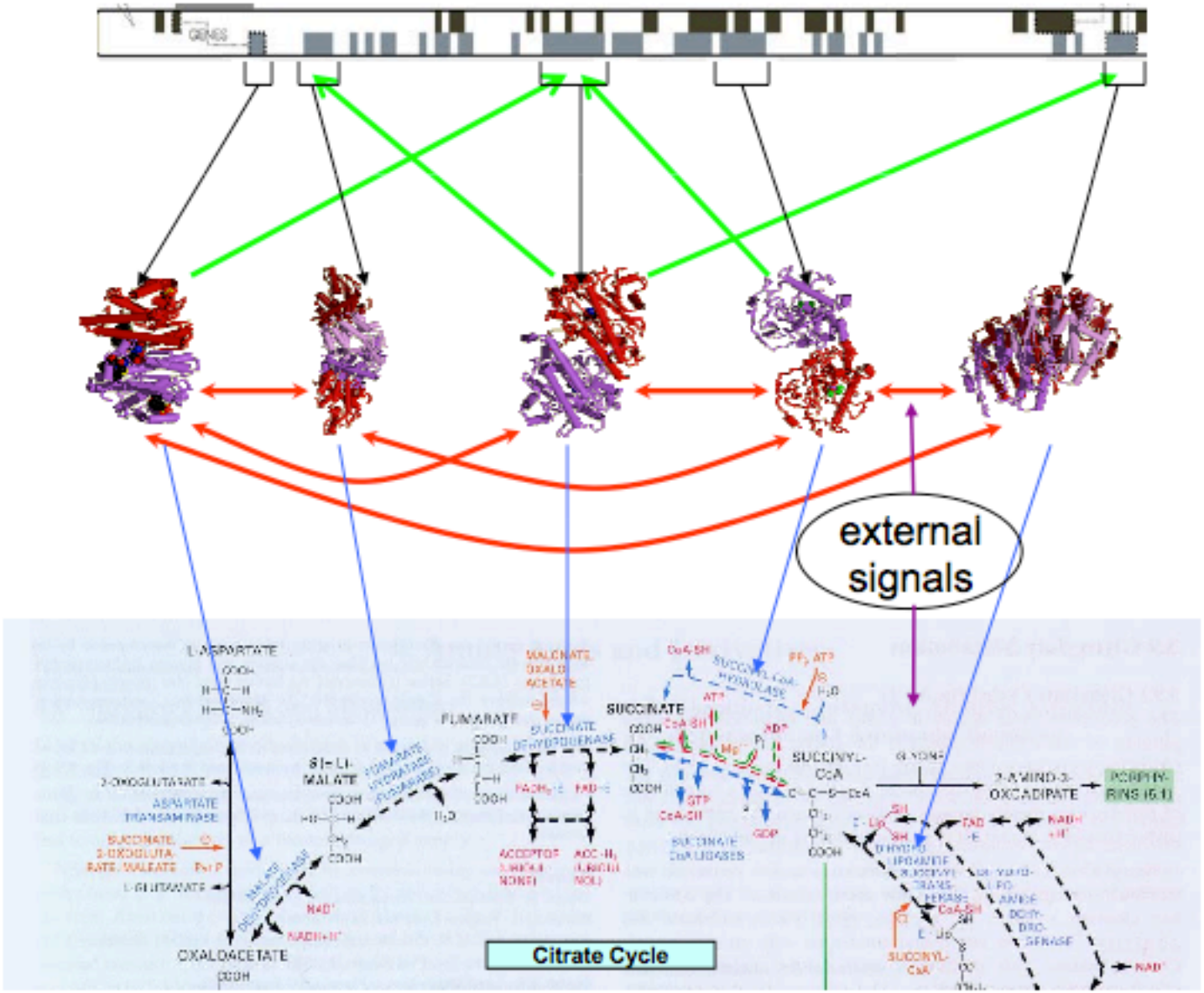
Computer networks





Food web (Nutrition flow in species)





GENOME

gene regulation

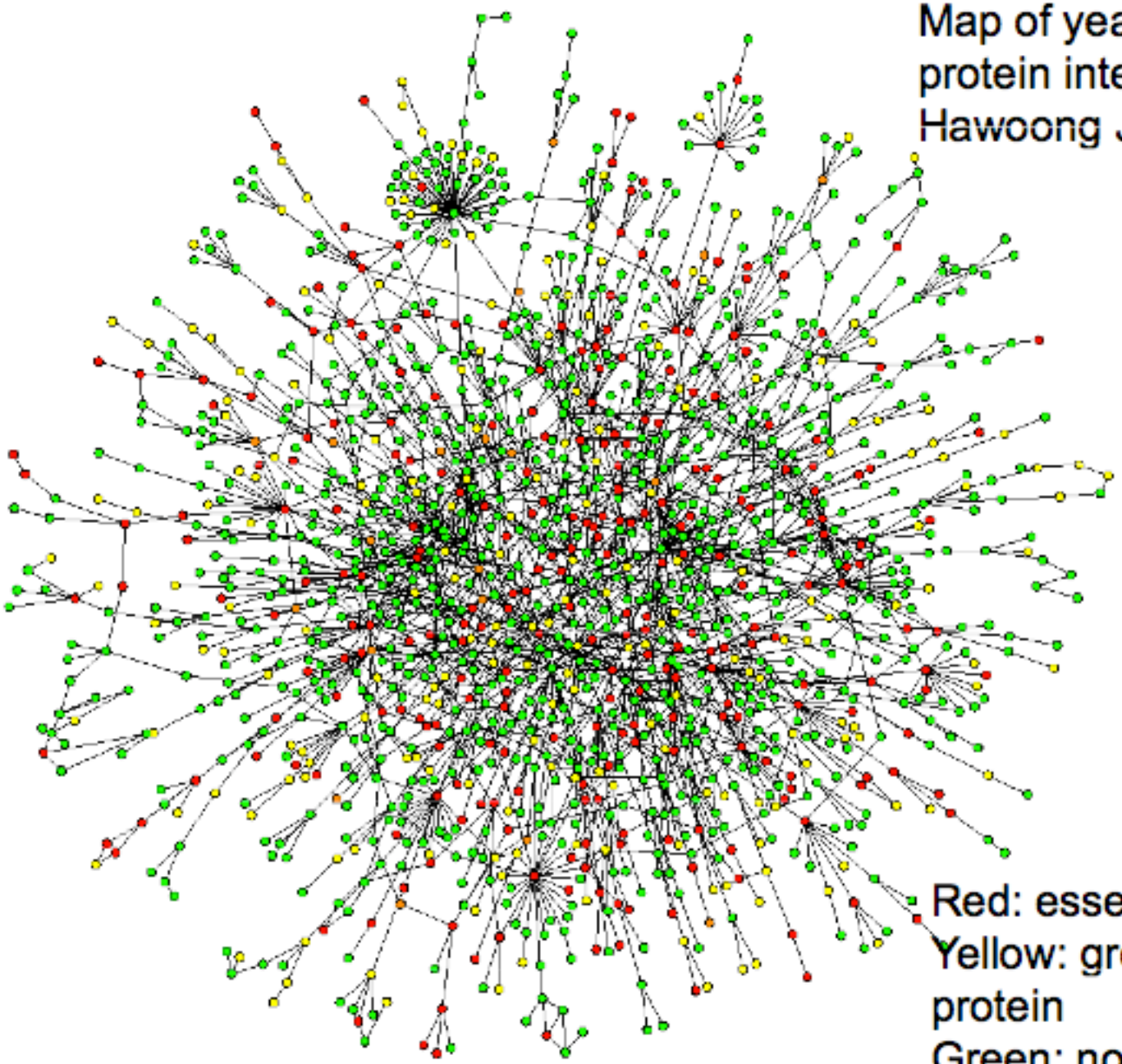
PROTEOME

protein-protein interactions

signal transduction

METABOLISM

Bio-chemical reactions



Map of yeast protein-protein interactions, by Hawoong Jeong

Red: essential protein
Yellow: growth- affecting protein
Green: non-essential protein

Why study networks?

- It is increasingly recognized that complex systems cannot be described in a reductionist view.
- Understanding the behavior of such systems starts with understanding the topology of the corresponding network.
- Topological information is fundamental in constructing realistic models for the function of the network.

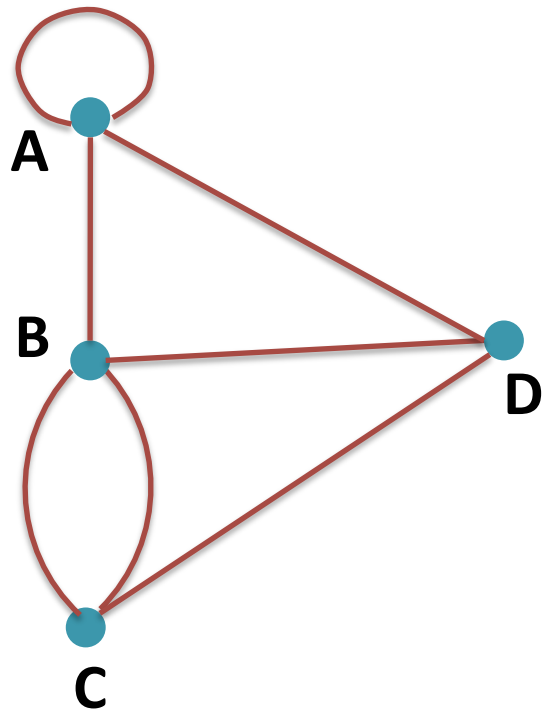
Network related questions

- How do we determine or infer network topology ? How do we build a network?
- How can we quantitatively describe large networks?
- How did networks get to be the way they are?
- What are the consequences of a specific network organization?

Graph Theory

- Some basic concepts of Graph theory
- Some examples of Special graphs
- Graph operations
- Graph paths and cycles
- Graph connectivity
- Tree and Bipartite graph
- Network motifs

Graph concepts



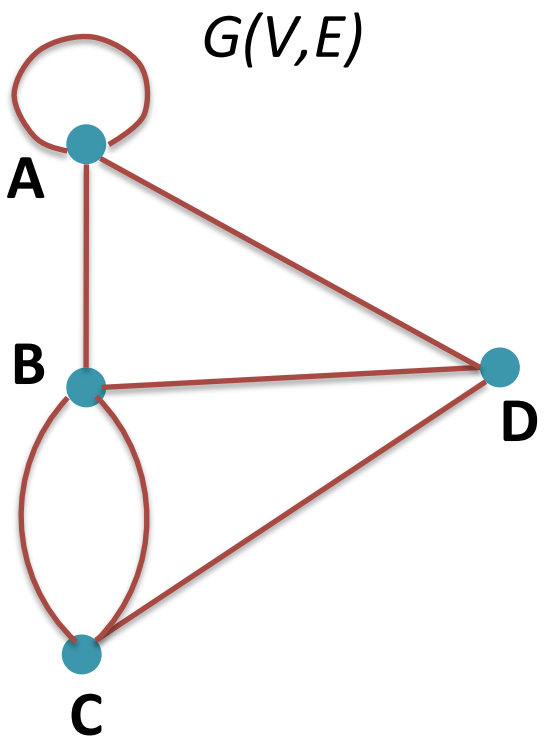
- Graphs are made up by vertices (nodes) and edges (links).
- An edge connects two vertices, or a vertex with itself.

$G=(V,E)$

V: a finite set of vertices.

E: edges of the graph

Graph concepts: some terms



- **Order** of graph *G*: the number of all vertices,

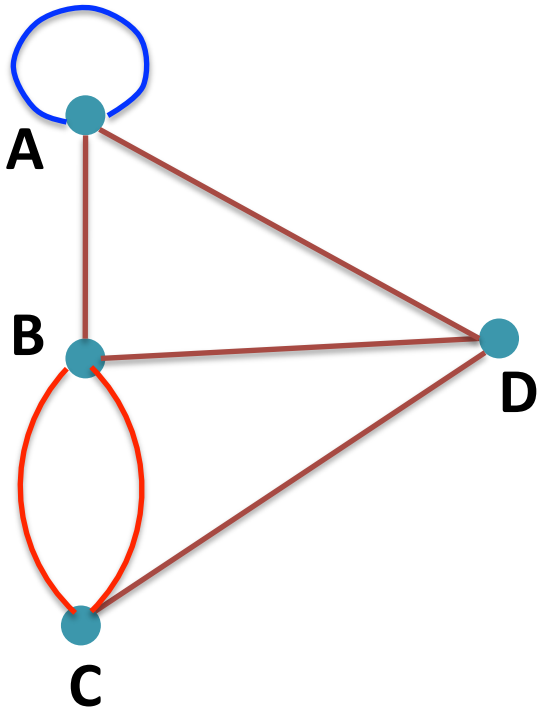
  $v_G = |V|$

- **Size** of graph: the number of all edges,

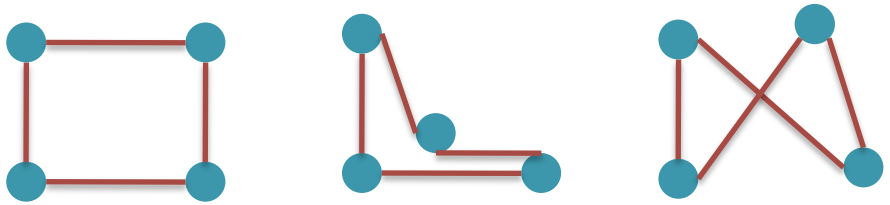
  $e_G = |E|$

- For an edge $e=uv$, the vertices *u* and *v* are the **Ends** of this edge; *u* and *v* are **neighbors**.

Graph concepts

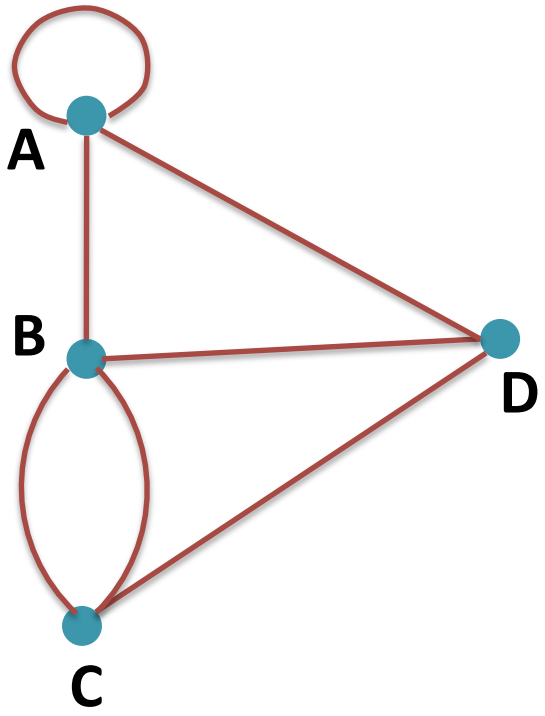


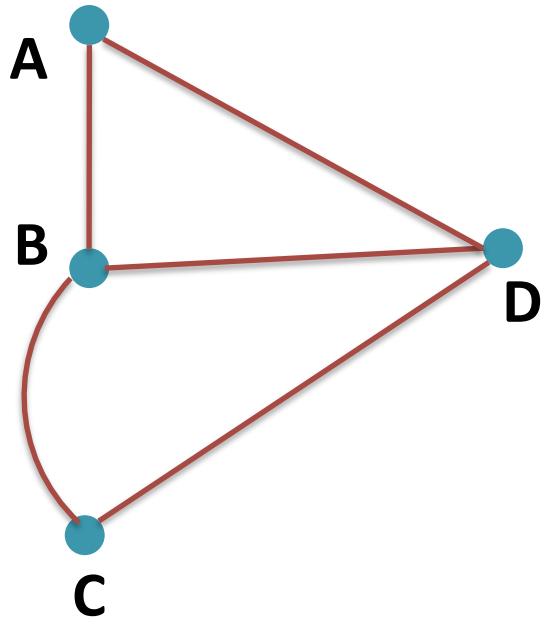
- Edges between B and C- multiple edges
- AA – loop
- The shape of the graph does not matter; only the way the nodes are connected to each other.



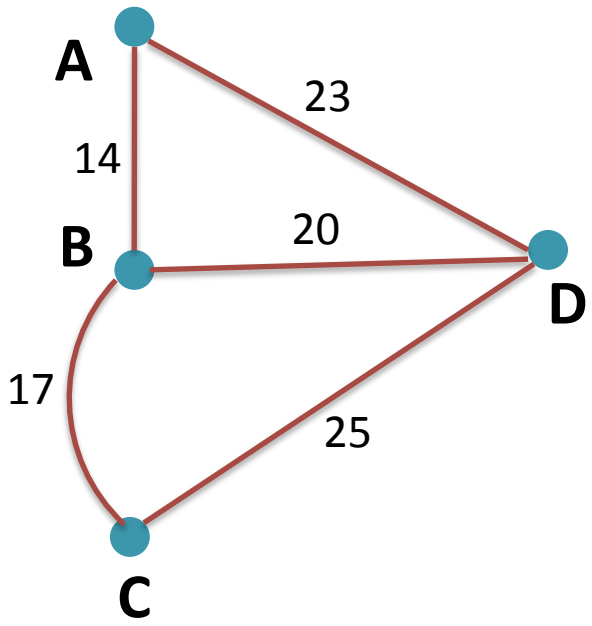
Simple graph

- A simple graph does not have loops (self edges) and multiple identical edges.



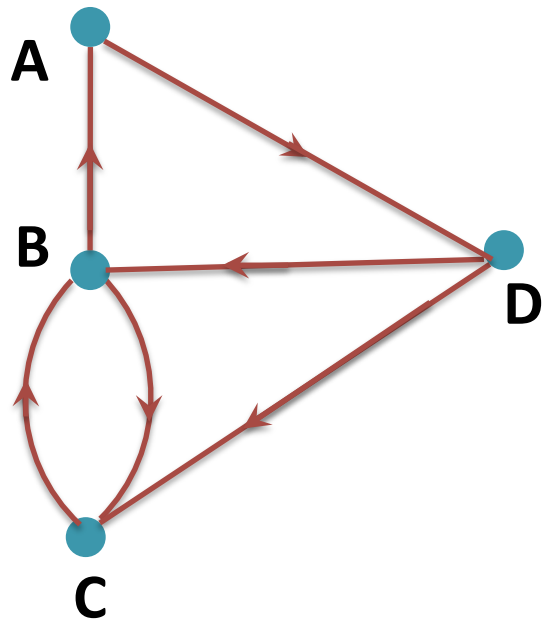


Weighted graph



- A edge has a weight value.
- In some applications, the weights, e.g., correspond to travel costs or geographical distances.

Directed Graph (Digraph)

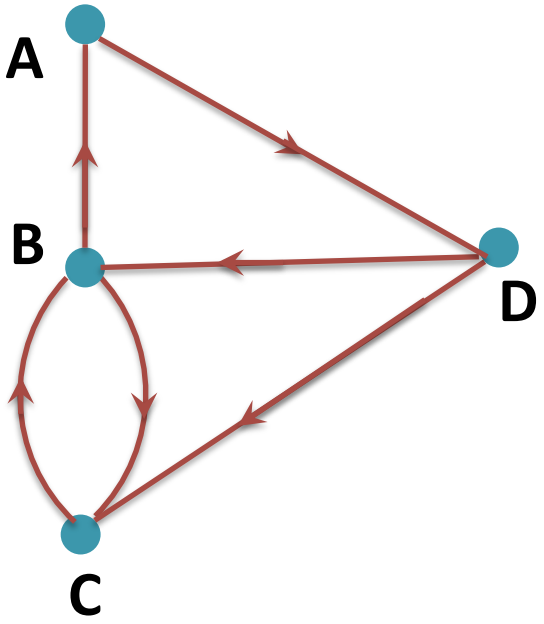


- Edges have directions, where the edges are drawn as arrows.
- The edges in the digraph are also called "arcs".
- A digraph can contain edges BC and CB of opposite directions.

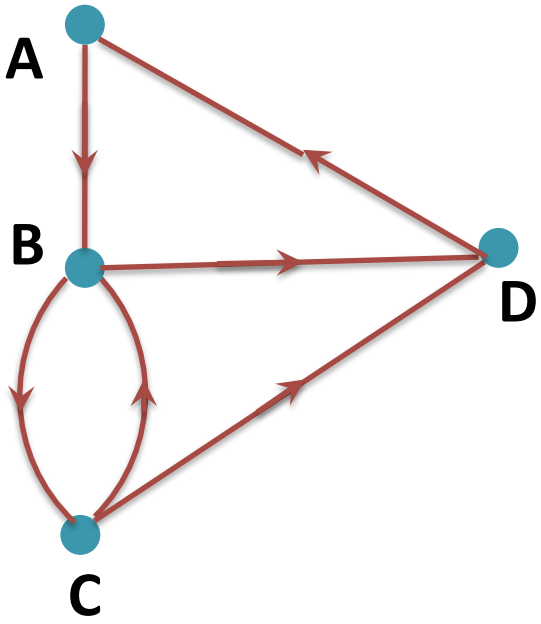
Question: Is this a simple graph?

Opposite of a Digraph

G(E,V)



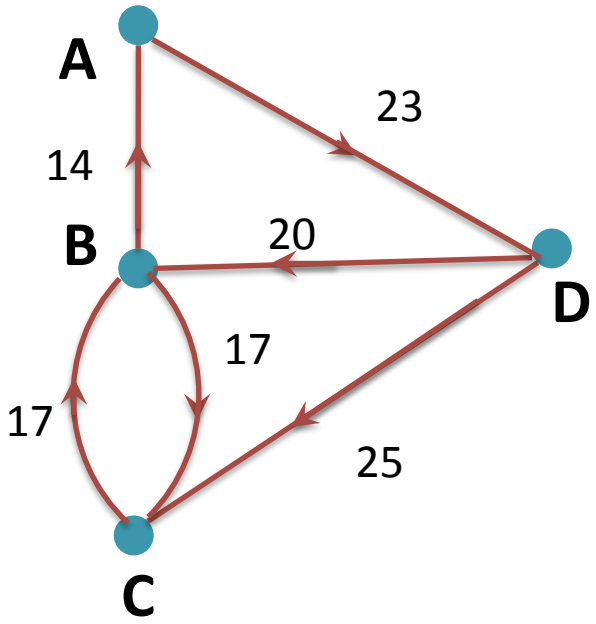
$G^{op}(E,V)$



All vertices are same, but the arrows reversed.

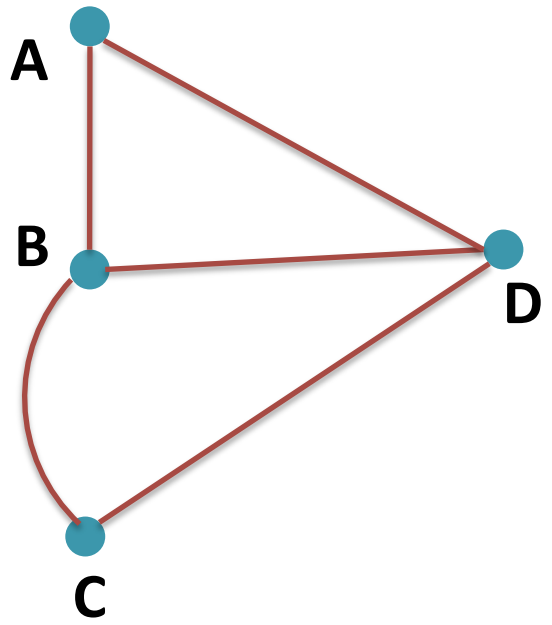
Weighted Digraph

- Edges have both weights and directions.
-



Representations of a graph

- Plane figures.
- List of edges.
- Adjacency matrix.



Representations: List of edges

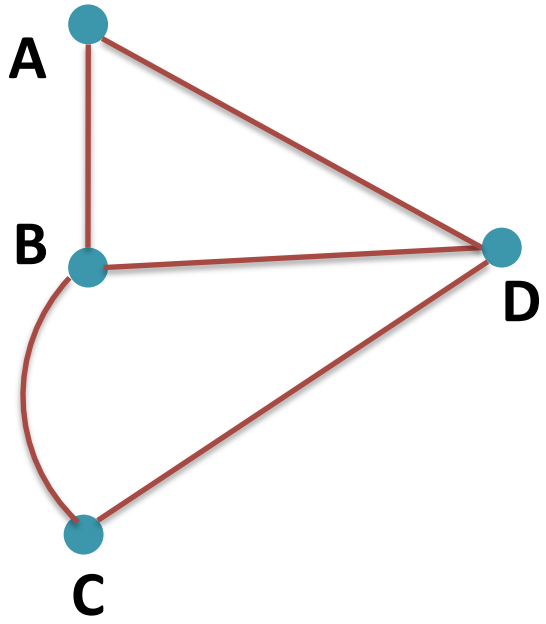
A – B

A – D

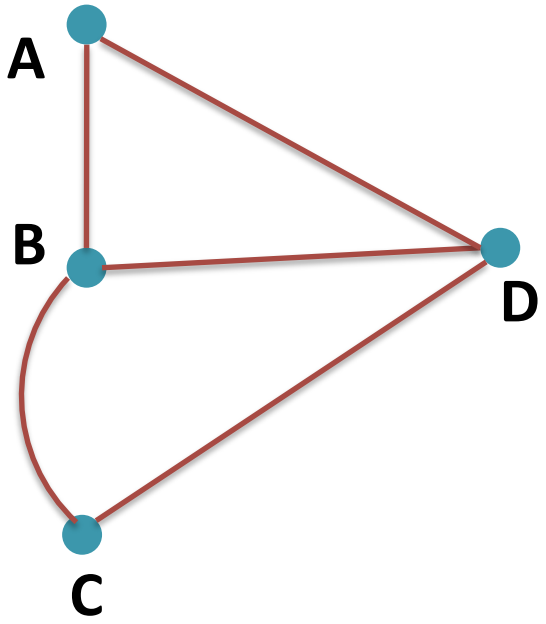
B – C

B – D

C – D



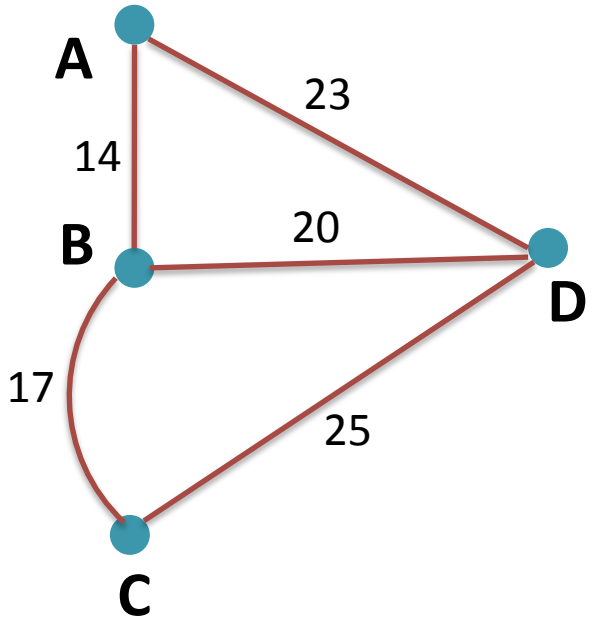
Representations: adjacency matrix



	A	B	C	D
A		1		1
B	1		1	1
C		1		1
D	1	1	1	

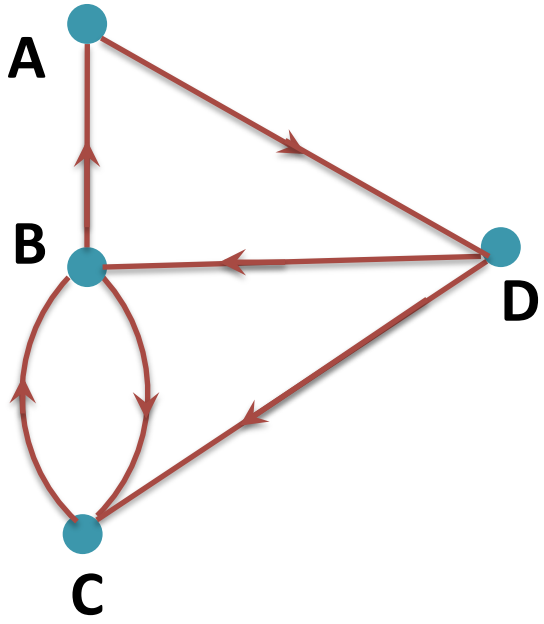
(1) Symmetric matrix.
(2) What does it mean if there is a number for a diagonal entry?

Representations: adjacency matrix for a weighted graph



	A	B	C	D
A		14		23
B	14		17	20
C		17		25
D	23	20	25	

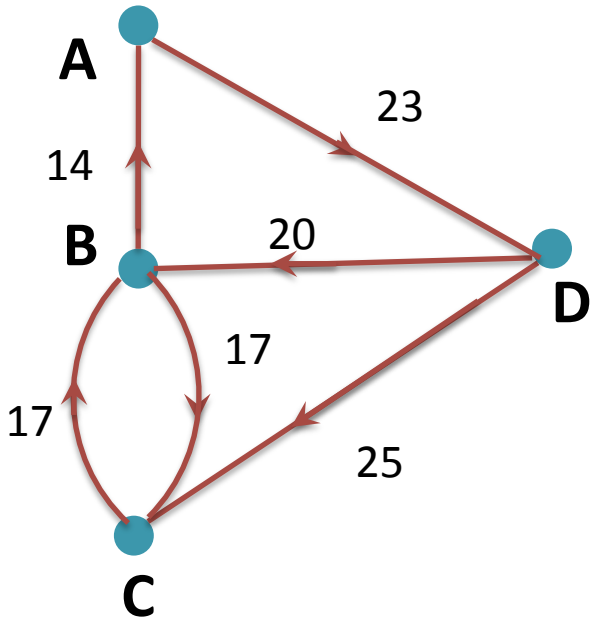
Representations: adjacency matrix for a digraph



	A	B	C	D
A				A->D 1
B	B->A 1		B->C 1	
C		C->B 1		
D		D->B 1	D->C 1	

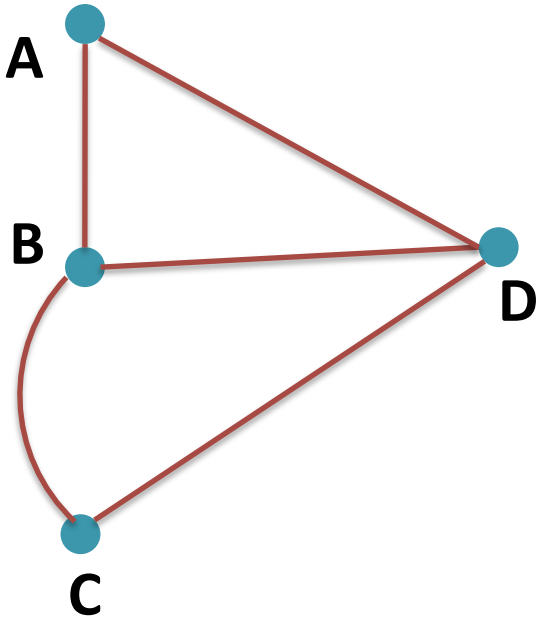
This matrix is not necessary to be symmetric.

Representations: adjacency matrix for a weighted digraph



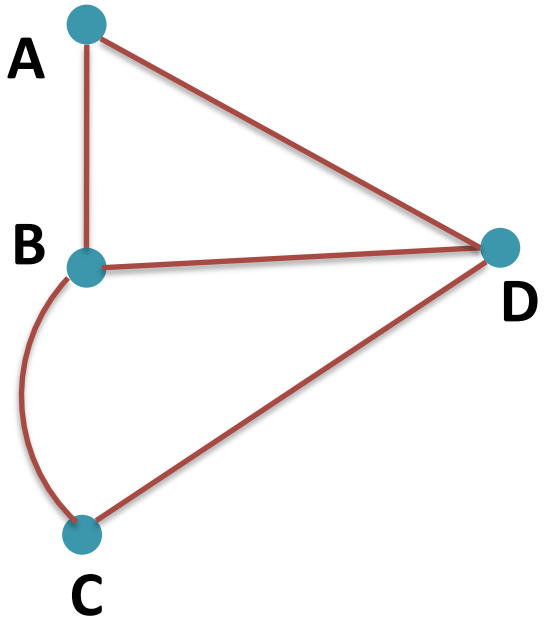
	A	B	C	D
A				A->D 23
B	B->A 14		B->C 17	
C		C->B 17		
D		D->B 20	D->C 25	

Node degrees



- Neighborhood: all neighbors of a node.
- Degree: the number of edges connected to the nodes; the number of neighbors of a node (vertex).
- Maximum degree and minimum degree. In a graph, the largest degree and the smallest degree.

Degrees in the adjacency matrix



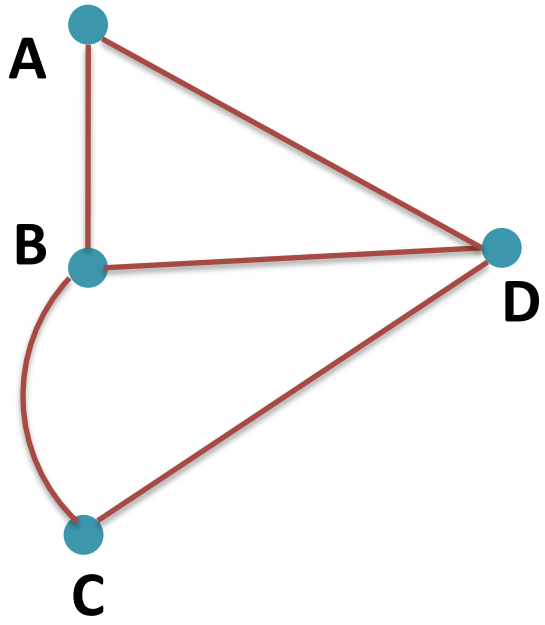
	A	B	C	D
A		1		1
B	1		1	1
C		1		1
D	1	1	1	

Degrees: 2 3 2 3

Handshaking Lemma

handshaking lemma (Euler, 1736): every finite undirected graph has an even number of vertices with odd degree.

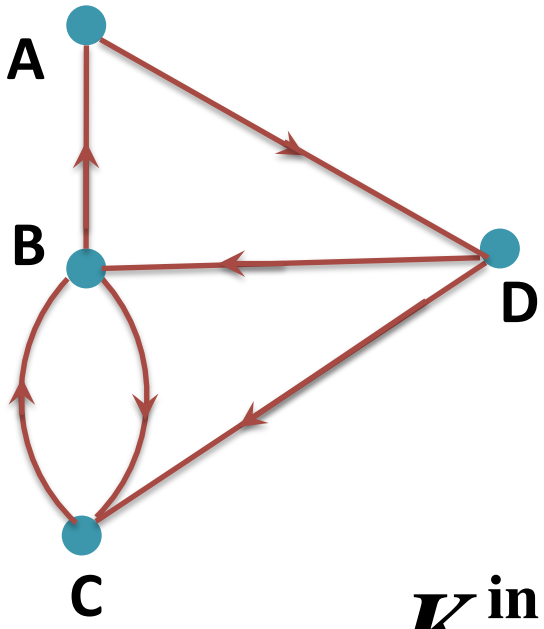
eg. B and D



What is the sum of degrees of all vertices in a graph?

Answer: 2|E|

Degrees for Digraphs



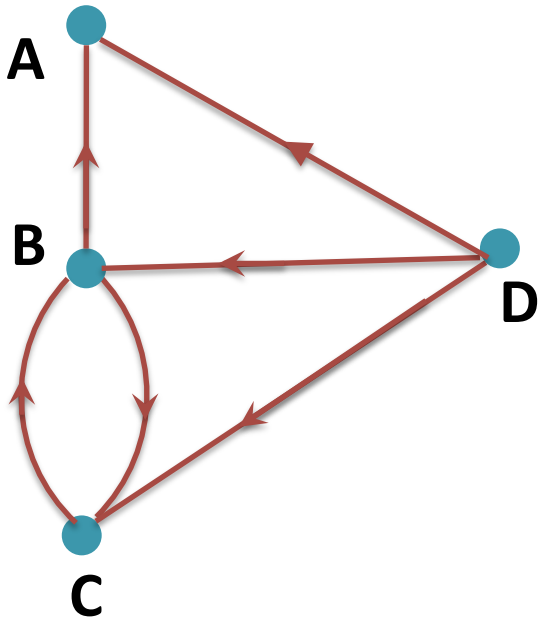
- In digraphs, we can define an in-degree and out-degree for a given node.
- The (total) degree is the sum of in- and out-degree.

$$K_C^{\text{in}} = 2$$

$$K_C^{\text{out}} = 1$$

$$K_C = 3$$

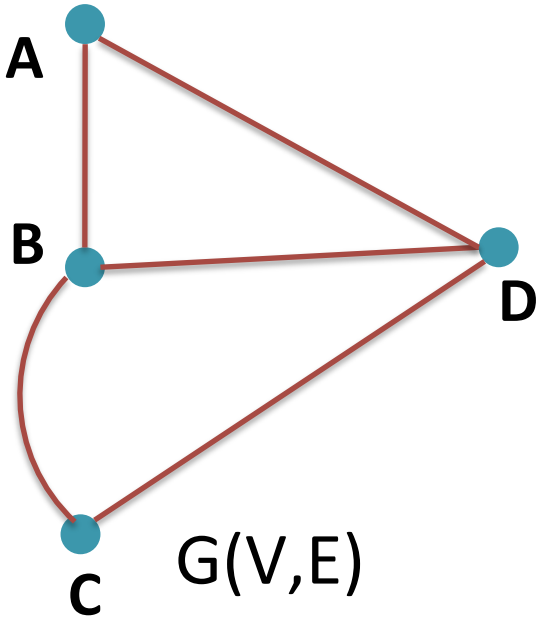
Degrees for Digraph – source or sink



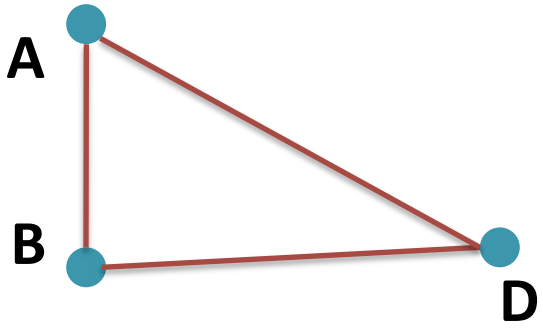
- Source: a node with in-degree=0.
- Sink: a node with out-degree = 0.
- eg. D is a source.
- eg. A is a sink.

Subgraph

- If *H* is a subgraph of *G*, V(H) $\subseteq$ V(G) and E(H) $\subseteq$ E(G).



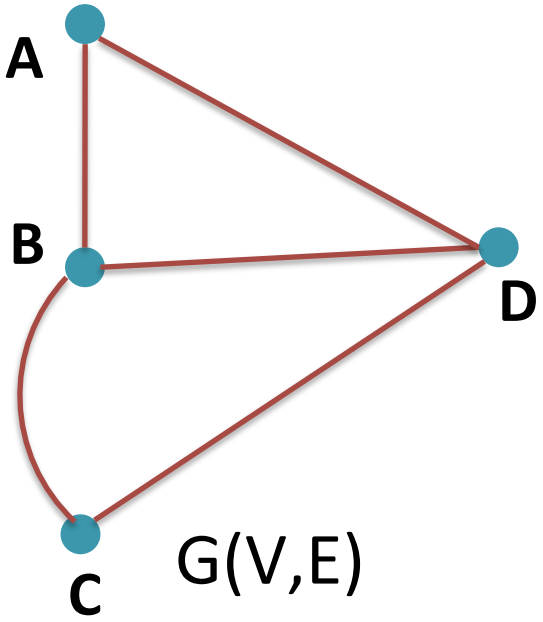
G(V,E)



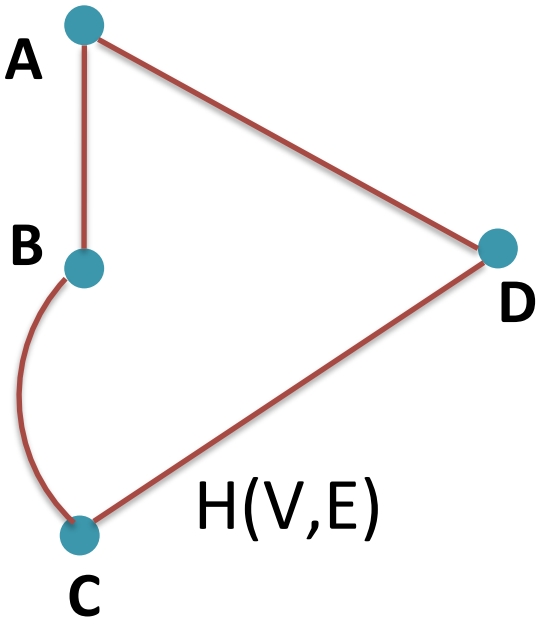
H(V,E)

Spanning Subgraph

- If *H* is a spanning subgraph of *G*, V(H)=V(G) and E(H) $\subseteq$ E(G).



G(V,E)



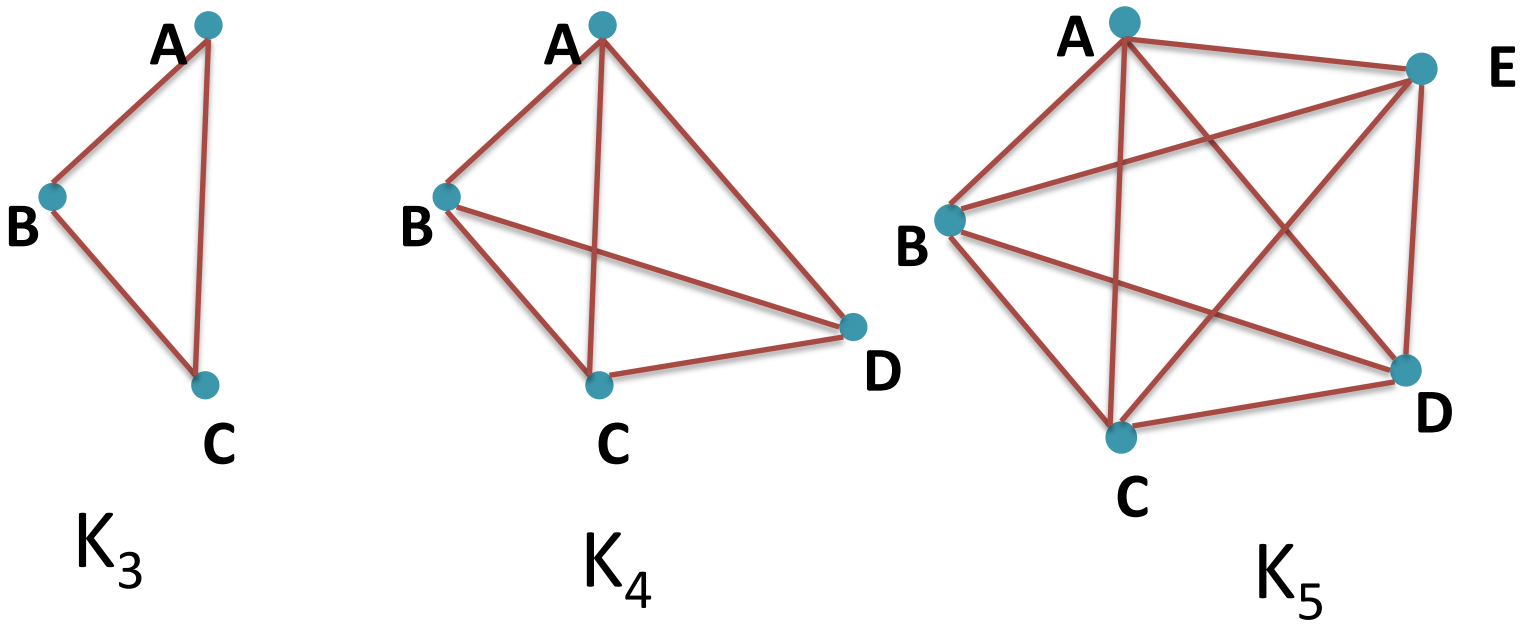
H(V,E)

Special graphs

- Trivial graph
- Complete graph
- Connected and disconnected graph
- Trees
- Bipartite graph
- Regular graph
- Line graph

Trivial graph

Complete graph (clique)



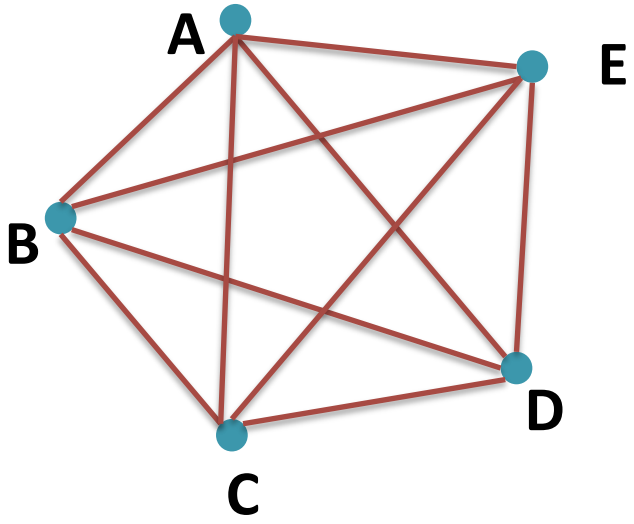
K_3 K_4 K_5

What the total number of edges in a complete graph?

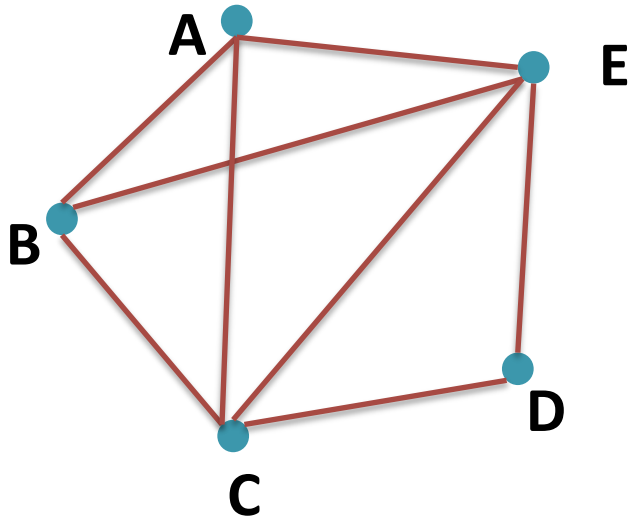
$$|E| = \binom{n}{2} = \frac{n(n-1)}{2}$$

Connection Density

$$Q = \frac{\#\text{of Connections}}{\text{Max.}\#\text{of possible Connections}} = \frac{E}{V(V-1)/2}$$



Q=1

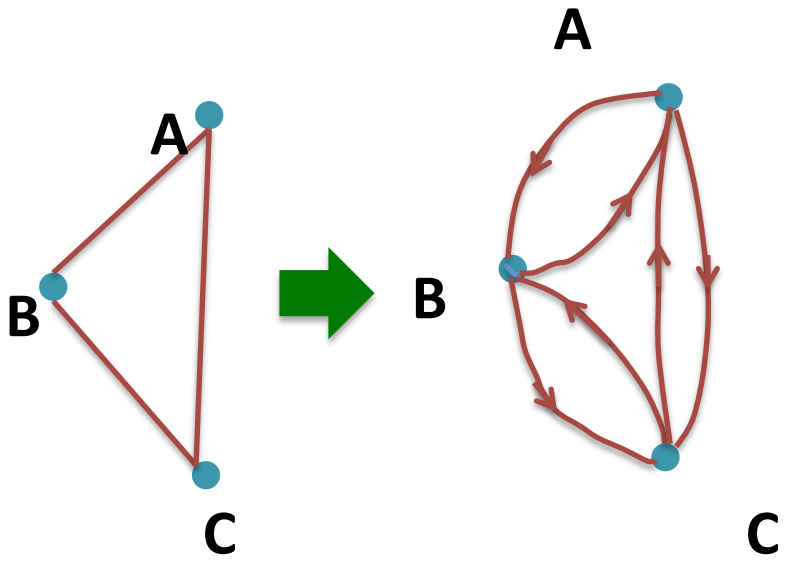


Q=8/10

Complete digraph

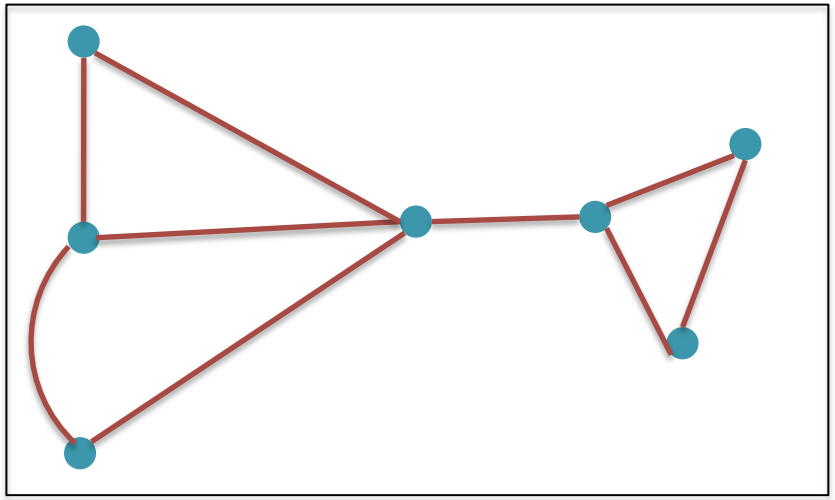
What is the largest number of arcs that a simple digraph with *N* nodes can have?

$$|E| = 2 \times \binom{n}{2} = 2 \times \frac{n(n-1)}{2}$$

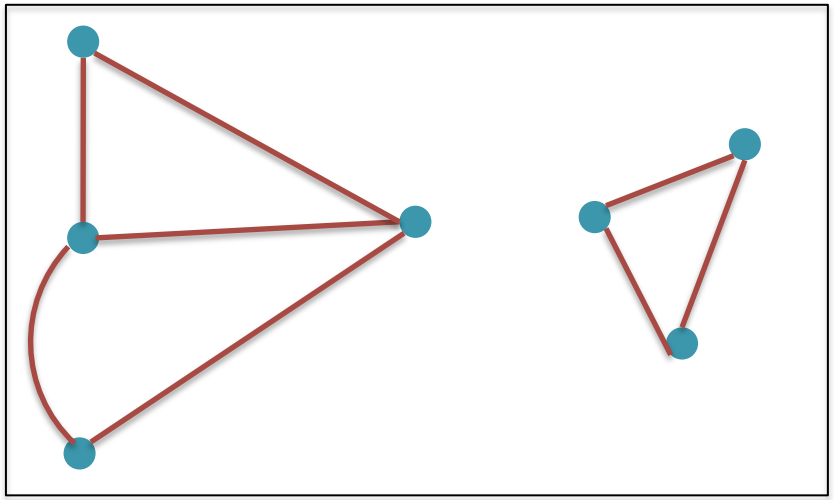


Connected and disconnected

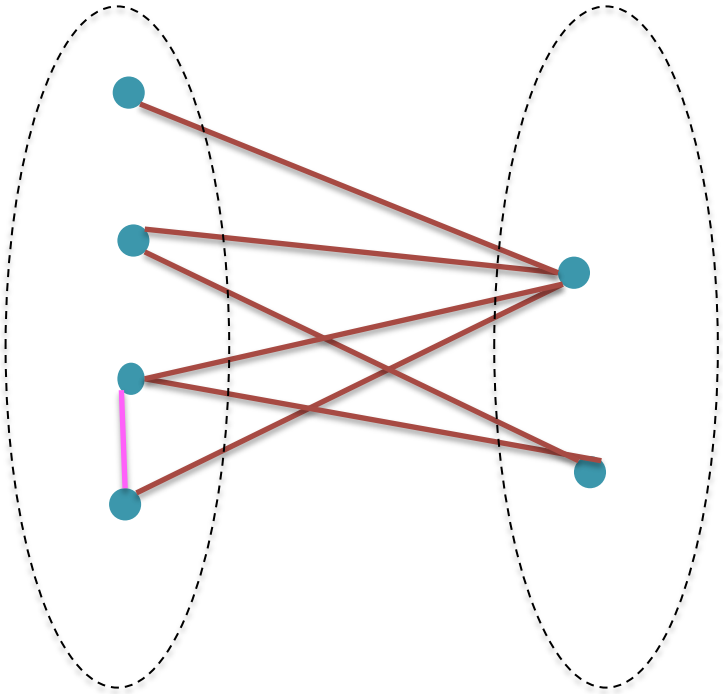
G1



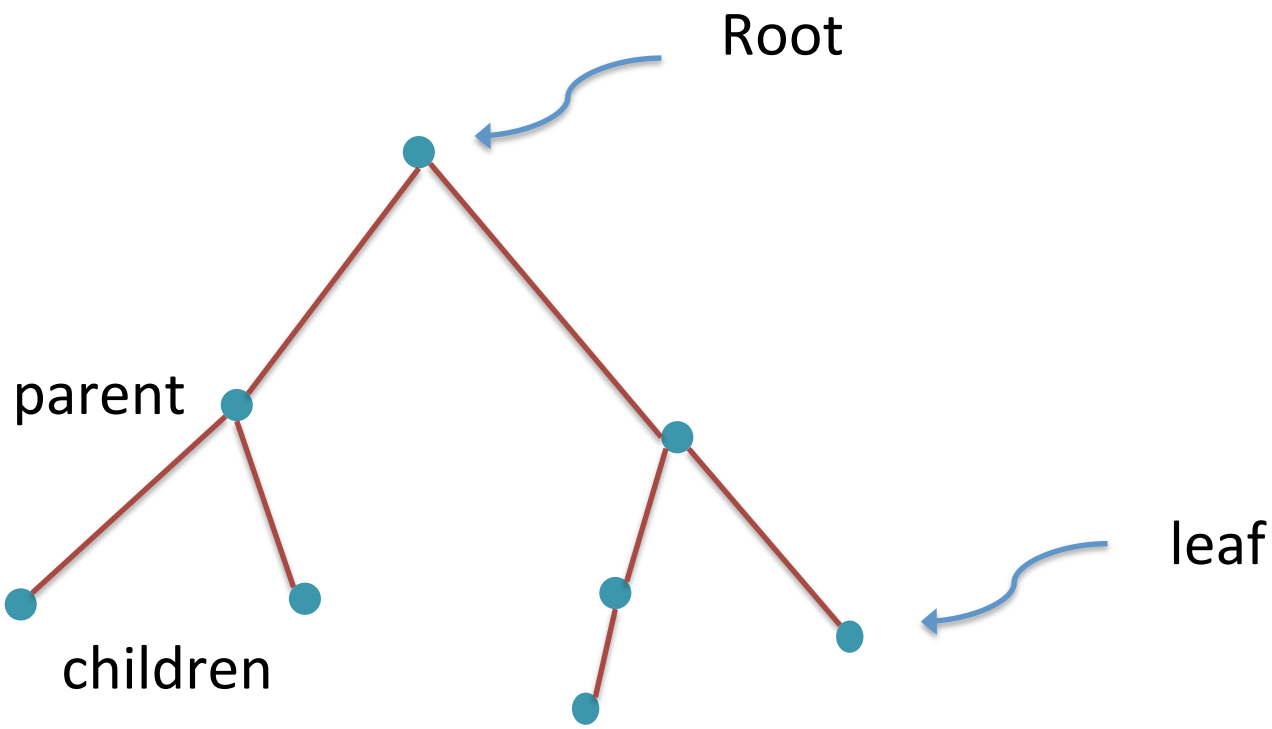
G2



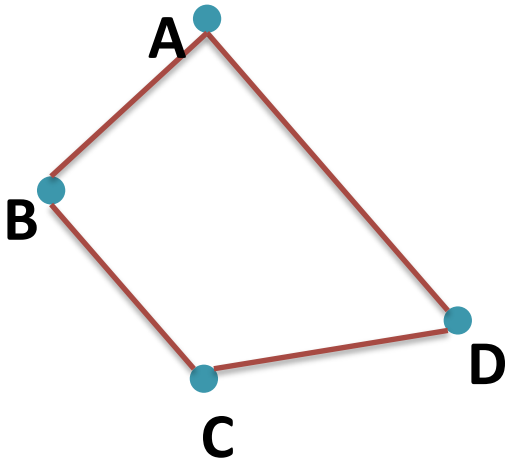
Bipartite graph



Trees



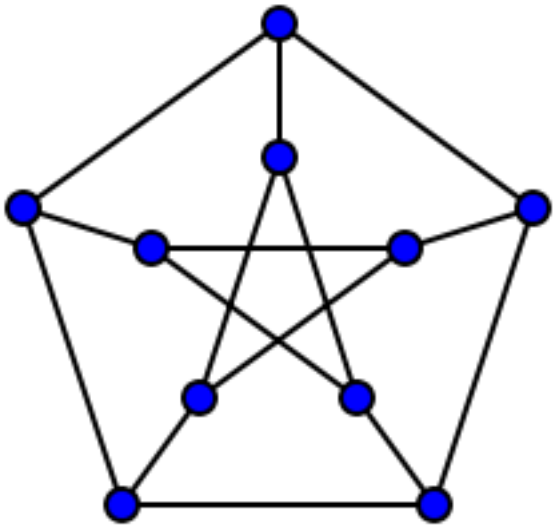
Regular graph



- A graph *G* is said to be regular, if every vertex of *G* has the same degree.
- If the degree is *r*, then *G* is *r*-regular.

Is the complete graph a regular graph?

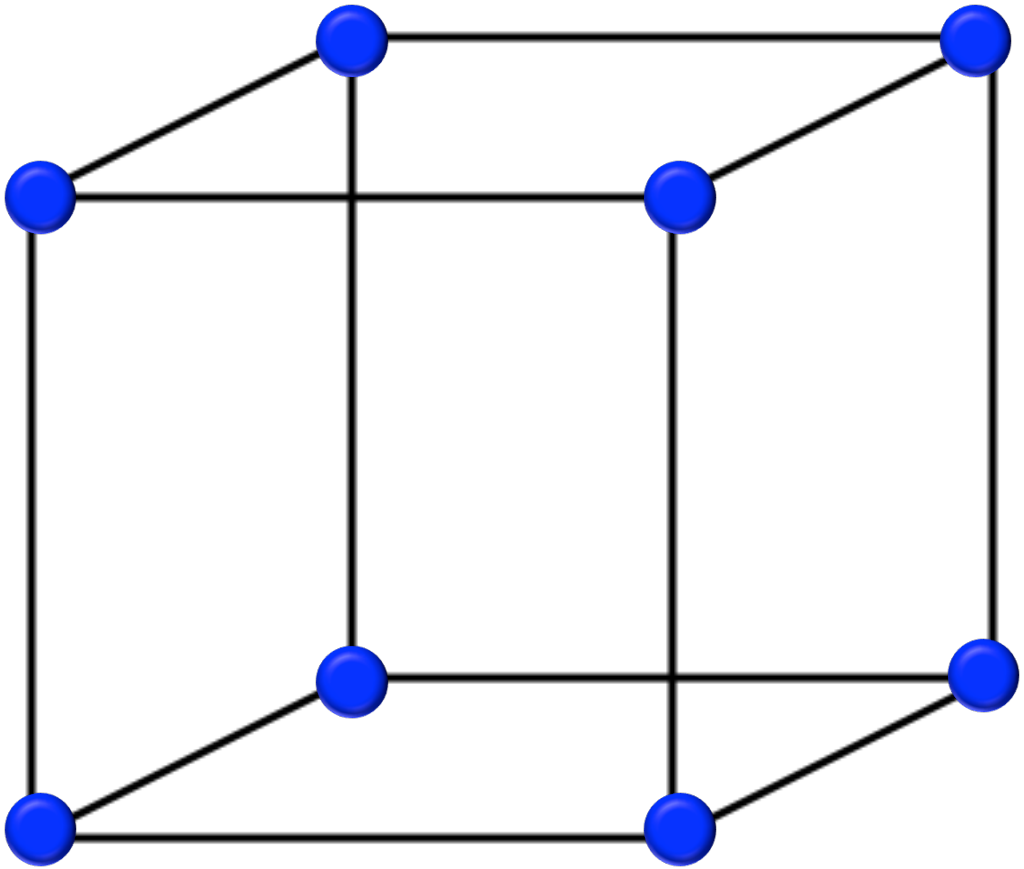
Regular graph: Petersen Graph



- Order = 10.
- Size = 15.
- 3-regular graph

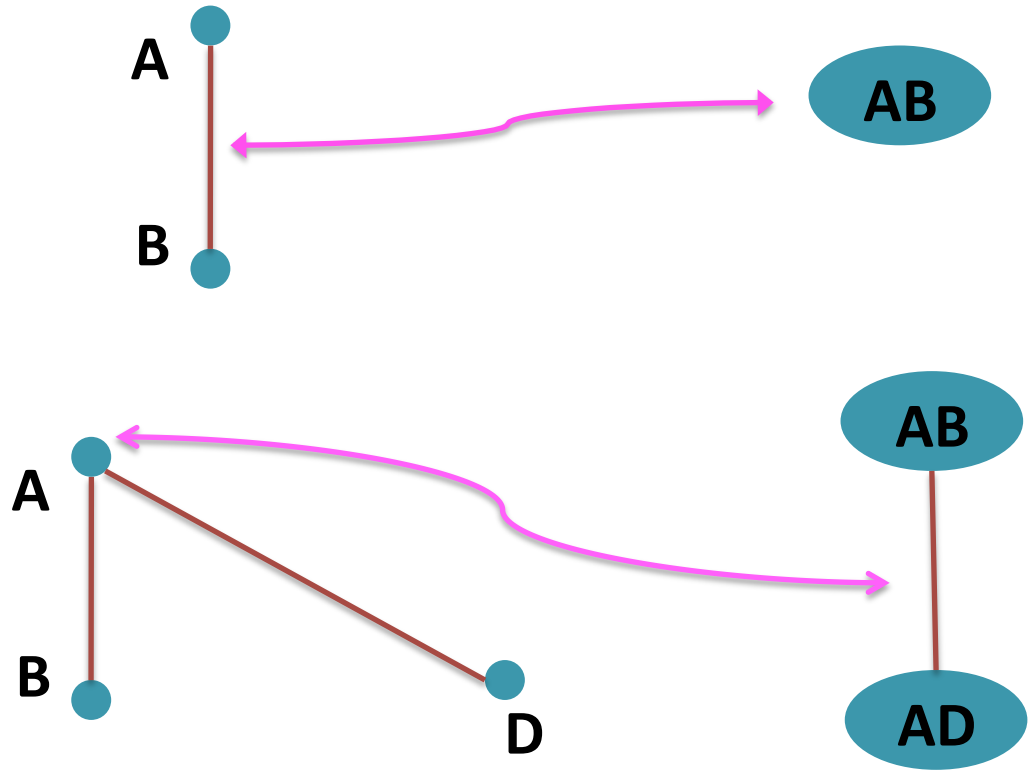
Regular graph

Can you find a 3-regular graph whose order is 8?

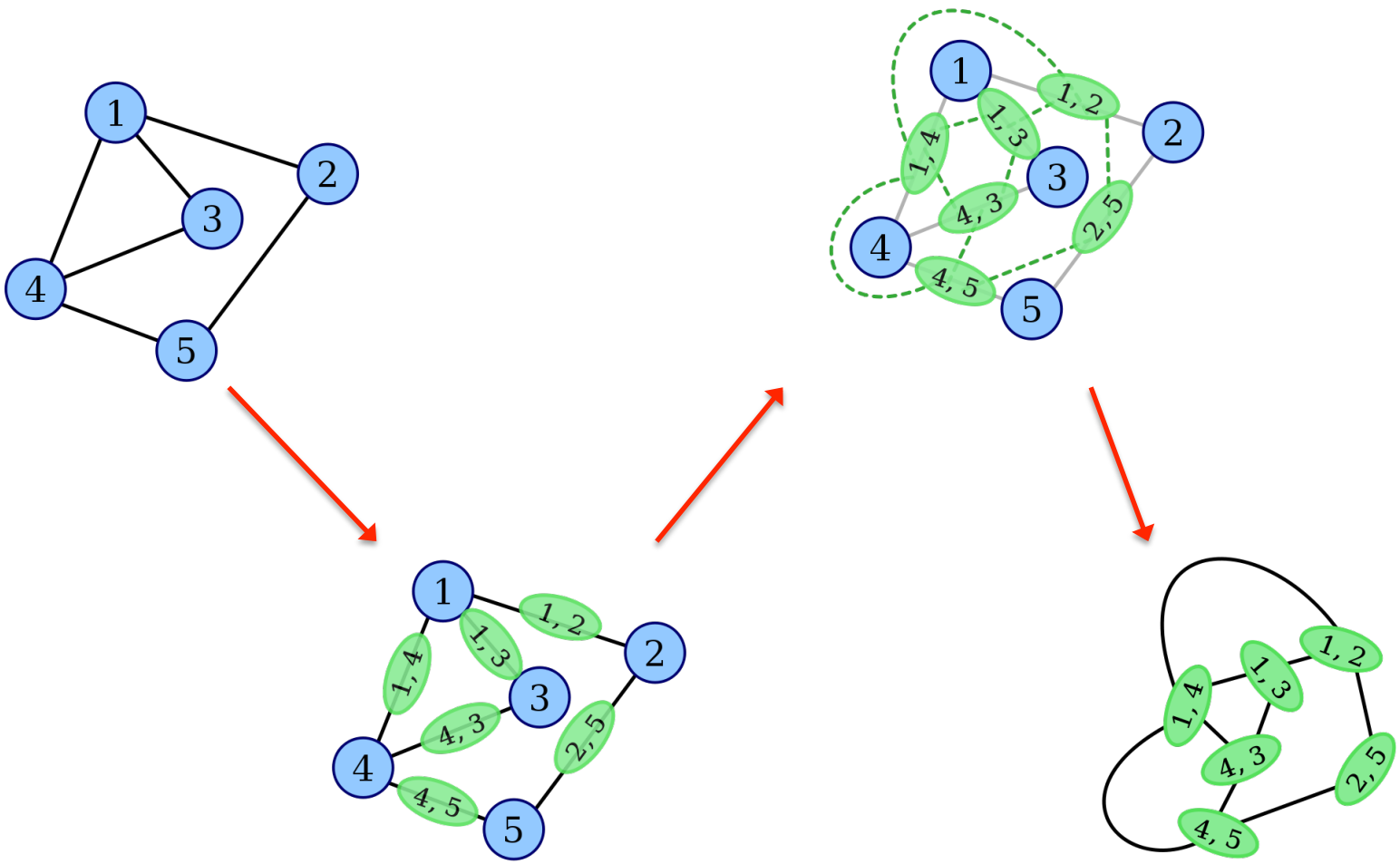


Line graph

G(V,E) → G’(E,V)



Line graph



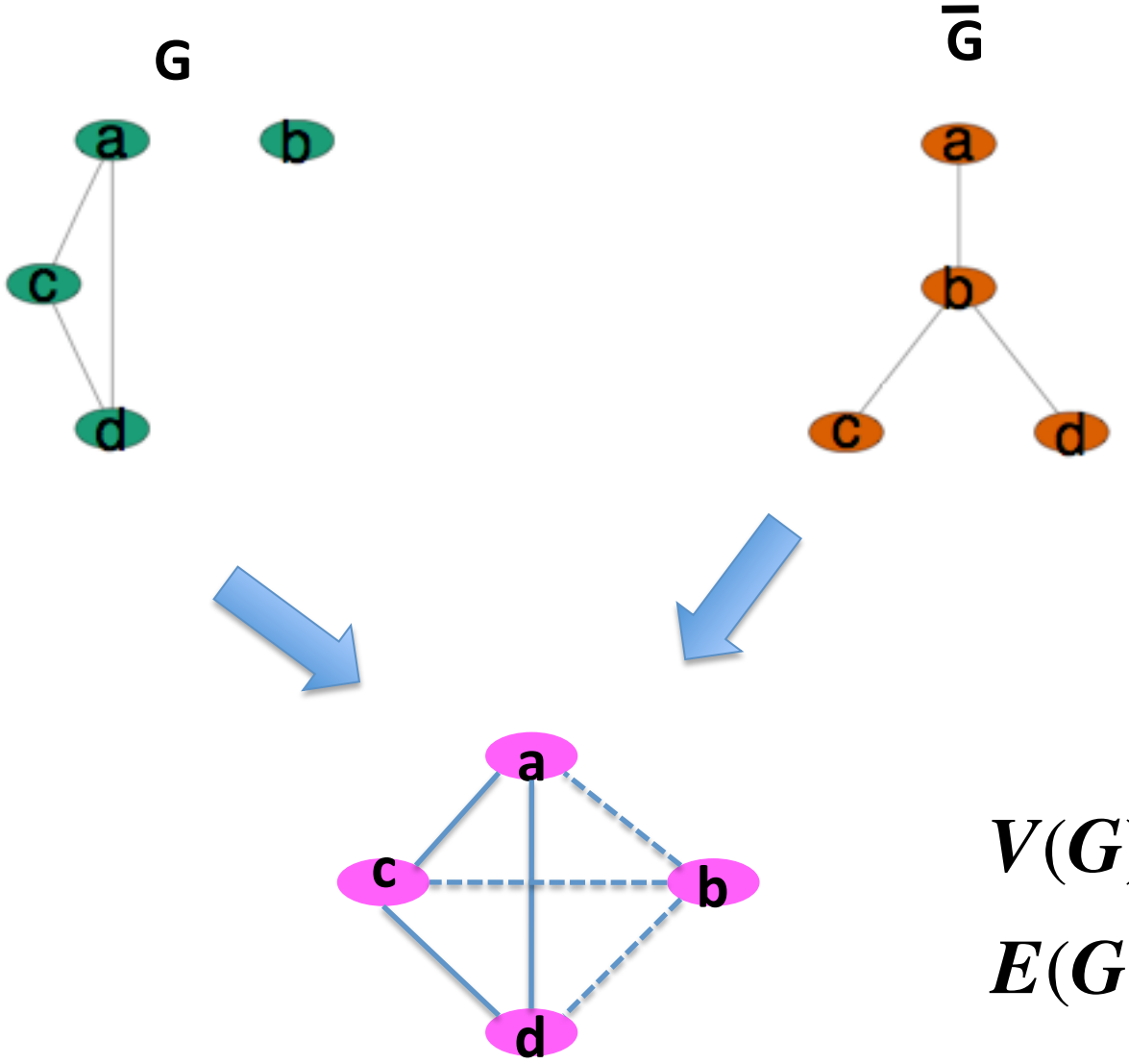
Graph Theory

- Some basic concepts of Graph theory
- Some examples of Special graphs
- Graph operations
- Graph paths and cycles
- Graph connectivity
- Tree and Bipartite graph
- Network motifs

Graph Operations

- Complement
- Union
- Intersection
- Rooted product of graphs

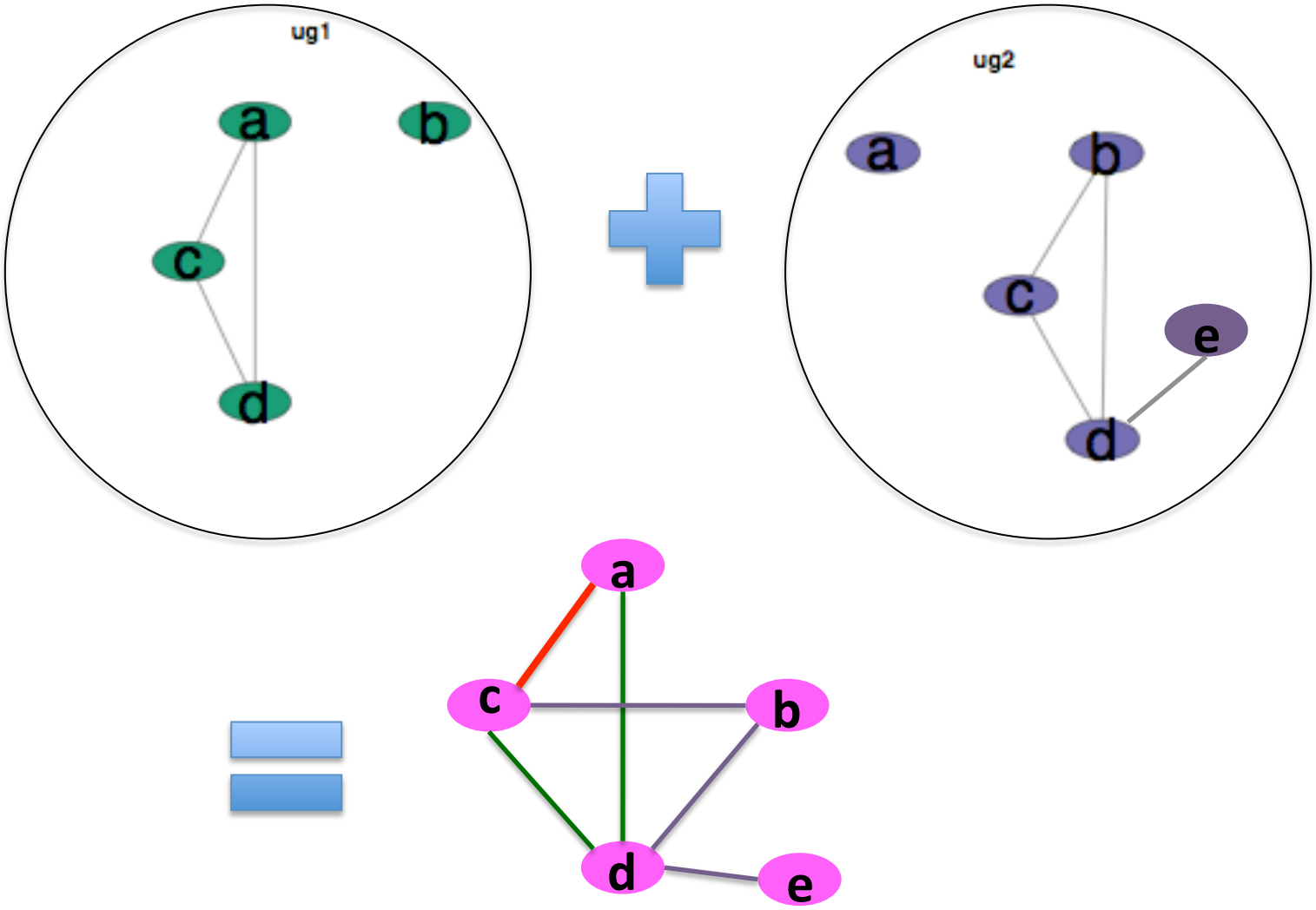
Graph Operations: complement



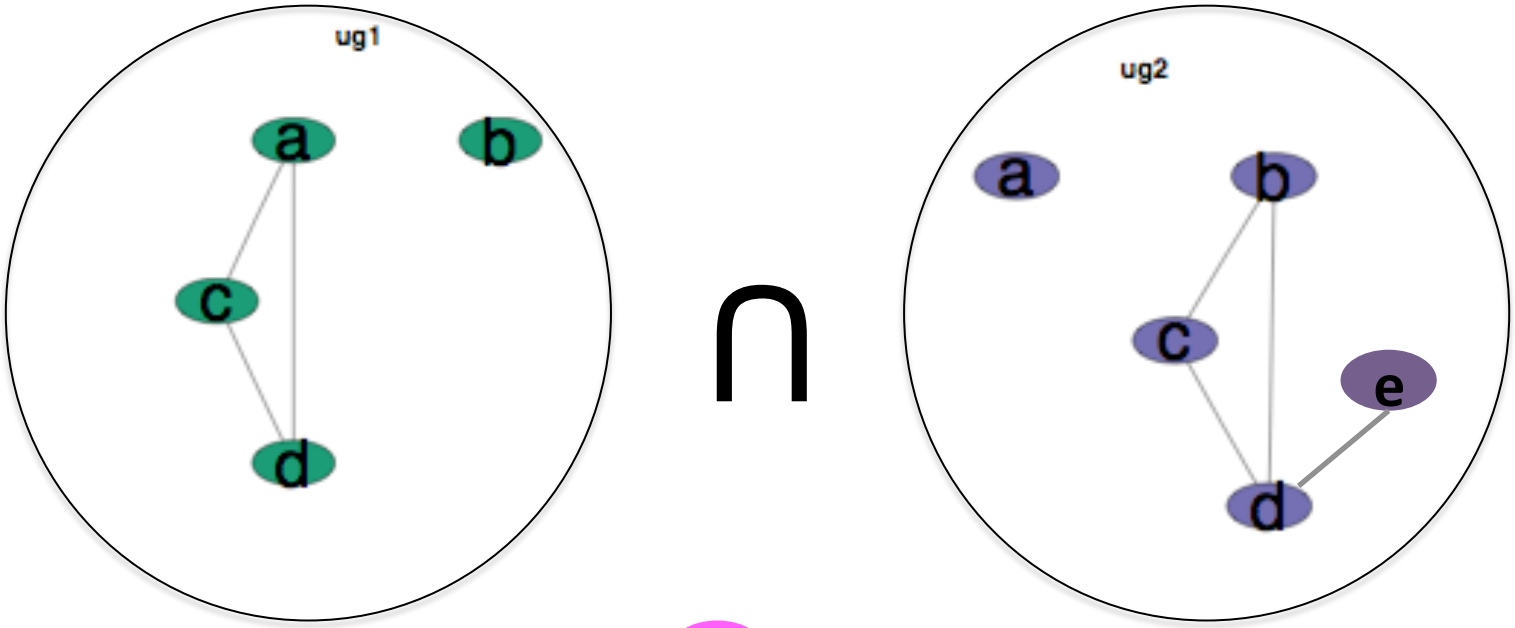
$$V(G) = V(\overline{G})$$

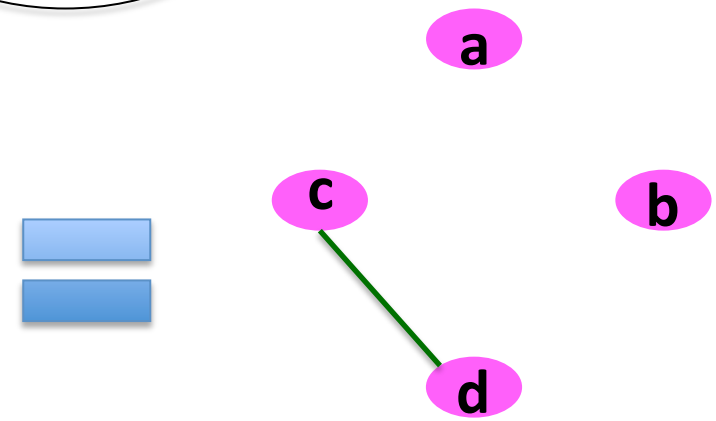
$$E(G) \neq E(\overline{G})$$

Graph Operations: union

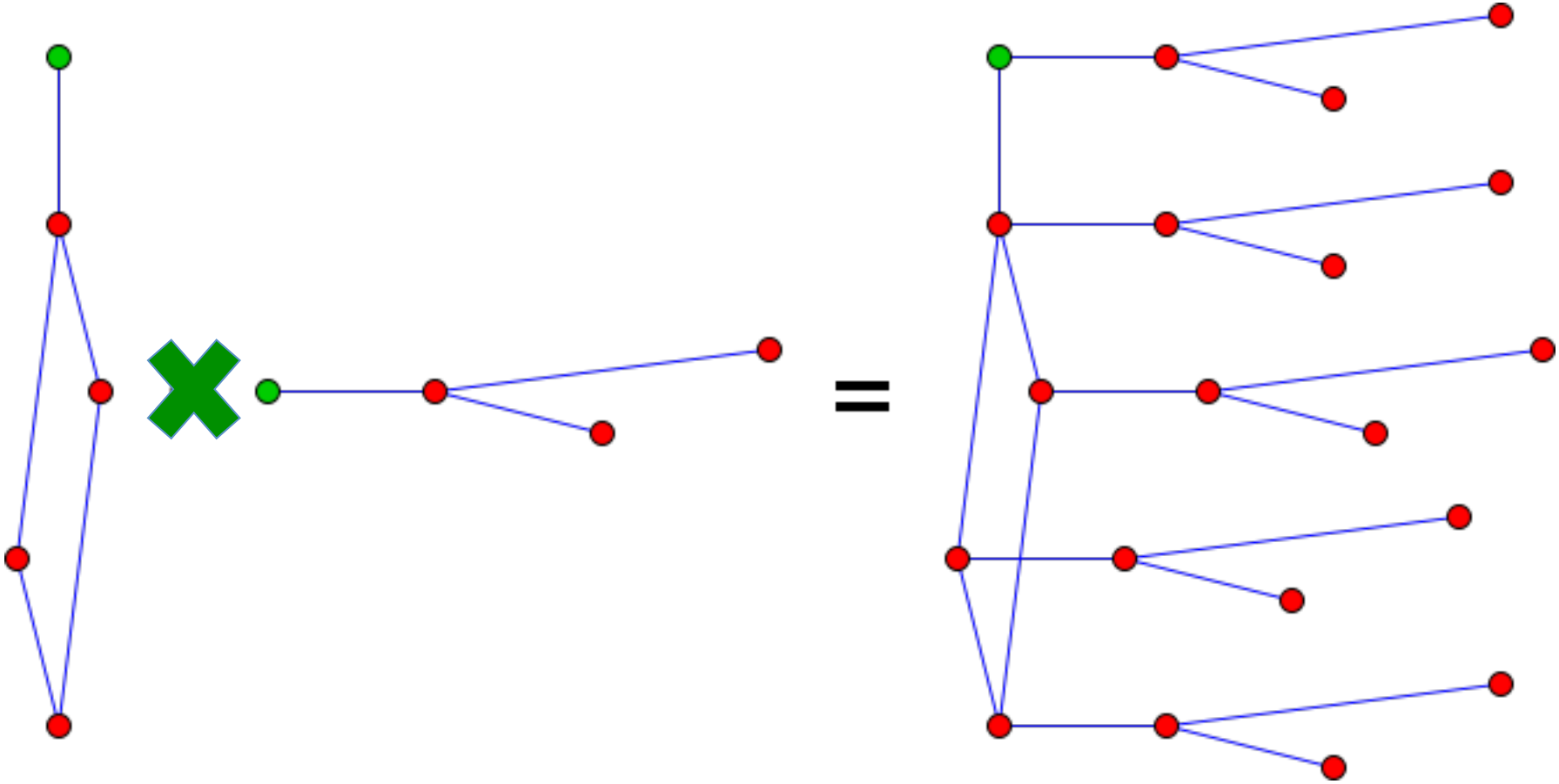


Graph Operations: intersection





Graph Operations: root production



Graph Theory

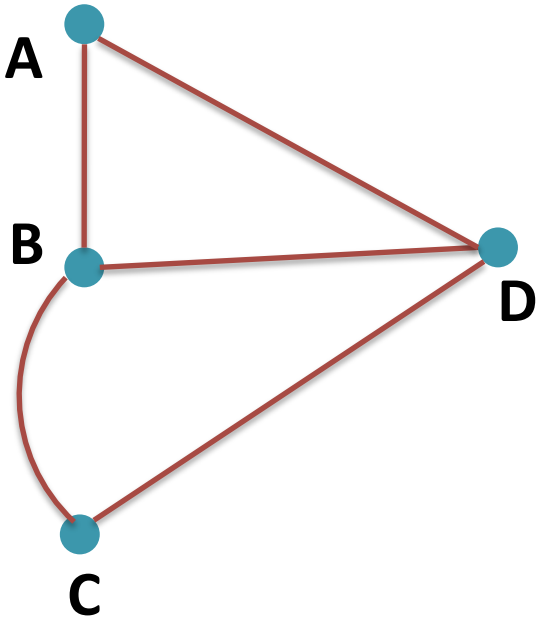
- Some basic concepts of Graph theory
- Some examples of Special graphs
- Graph operations
- Graph paths and cycles
- Graph connectivity
- Tree and Bipartite graph
- Network motifs

Subgraphs again, special

- Walks
- Paths
- Circuits
- Cycles

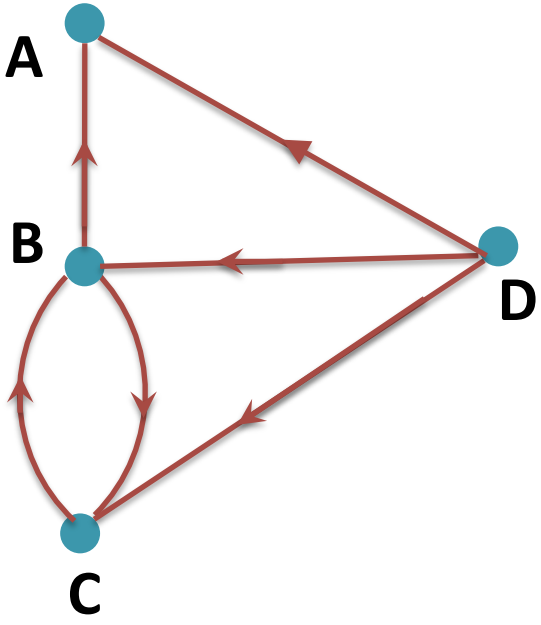
Walks

- Walk: a sequence of nodes in which each node is adjacent to the next one.
- In the digraph, a walk needs to follow the direction of edges.



A-B-D-C

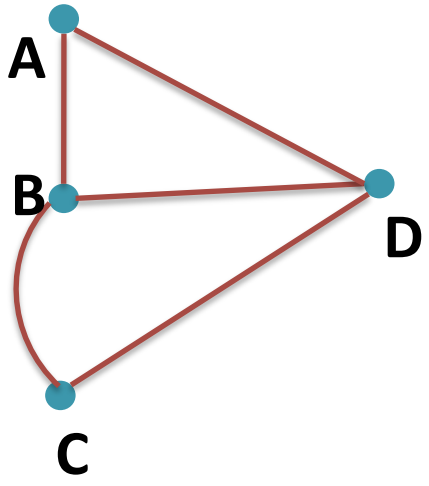
A-B-D-B-C-D



D→ C→ B→A

Paths

- Path: a sequence of nodes in which each node is adjacent to the next one, and edges can be part of a path only once.



A-B-D-C

~~**A-B-D-B-C-D**~~

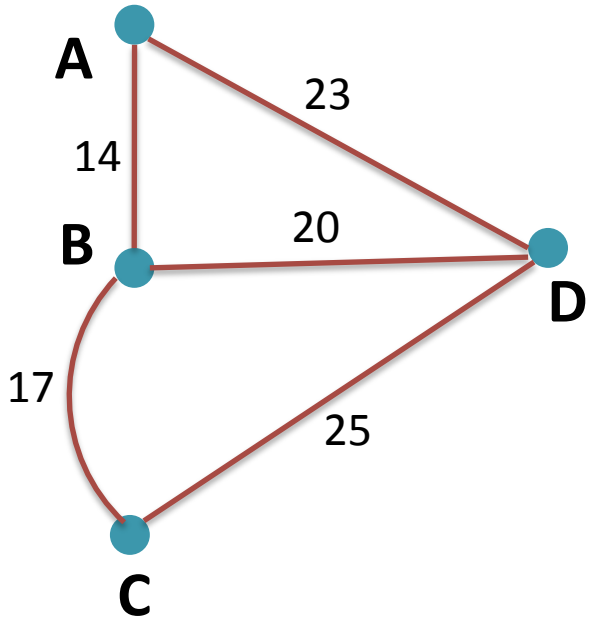
A path having *k* vertices, is denoted by P_k . The length of this path is *k-1*.

The longest Path

For a non-weighted complete graph with *n* nodes, what is the length for the longest path in this graph?

$$|E| - 1 = \binom{n}{2} - 1 = \frac{n(n-1)}{2} - 1$$

Paths in Weighted graph

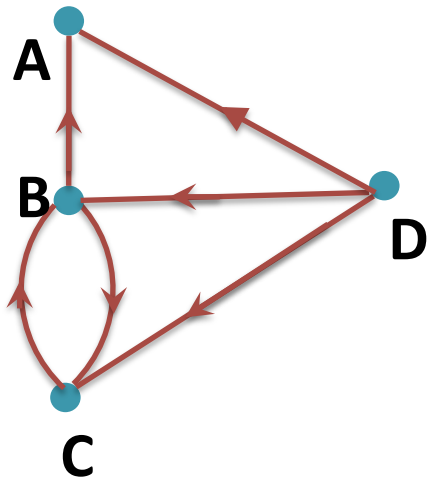


- The length of a path is the sum of all edge weights in the path.

Length of (A –B–C –D) = 14+17+25

Paths for digraph

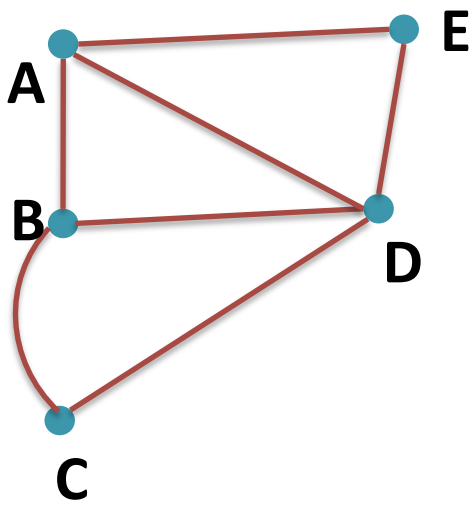
- In a digraph, a path needs to follow the direction of edges.
- In a digraph framework, a symmetrical edge can be used once in one direction and once in the opposite direction.



C→ B→C

Disjoint Paths

- Path P and Q are disjoint, if they do not share any vertex.
- Path P and Q are independent, if they at most share their ends.

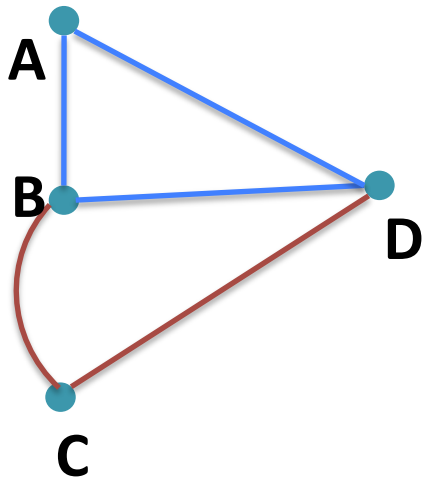


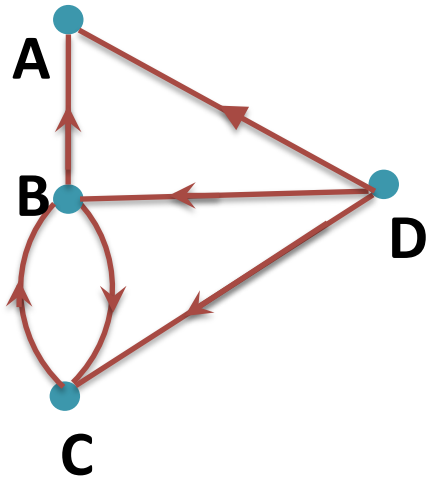
A-E-D and B-C are disjointed

A-E-D and A-B-C-D are independent

Circuits

- Circuit: a walk that starts and ends at the same vertex.



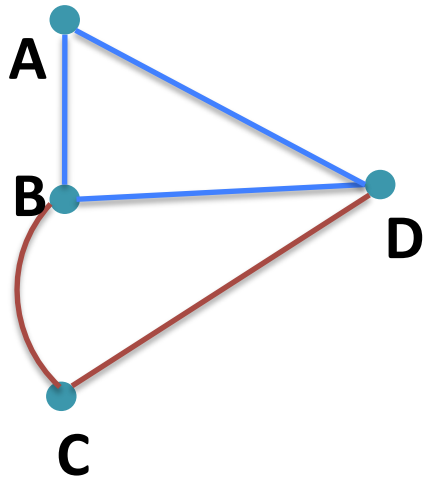


A - B – D - A
A - B - D - C - B - A

C→ B→C

Cycles

- Cycle: a circuit that does not revisit any nodes.

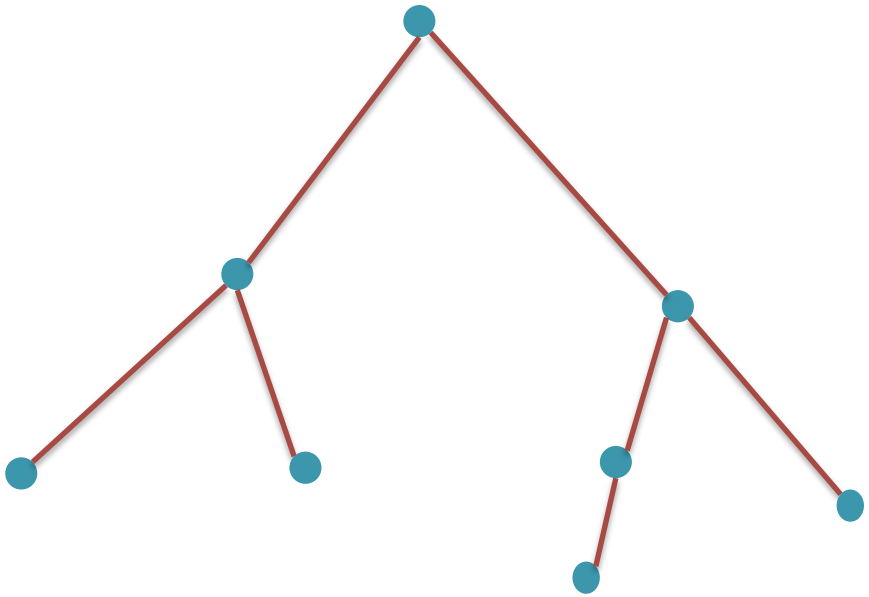


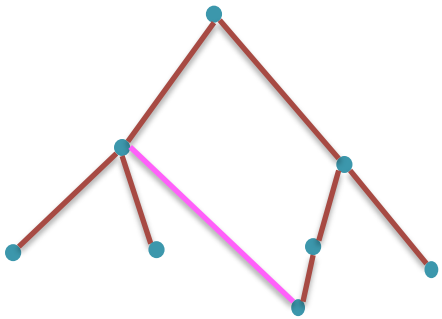
A cycle having *k* vertices, is denoted by C_k . The length of this cycle is also *k*.

A - B – D - A

~~**A - B - D - C - B - A**~~

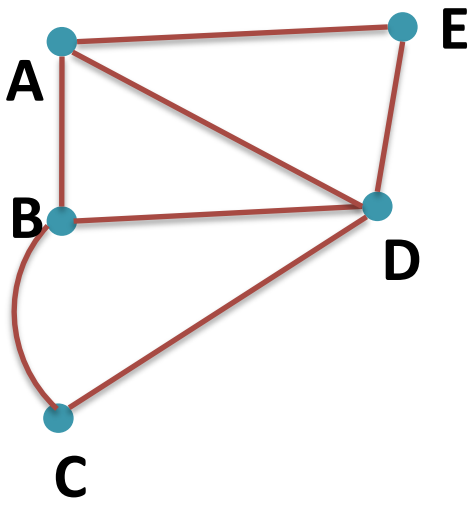
A tree does not have any cycle



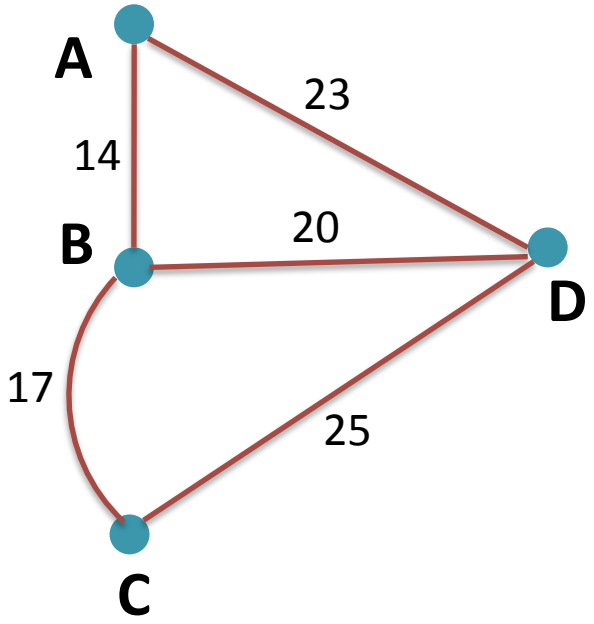


Shortest path

Between two nodes, there are multiple paths. The path having the shortest length is called the shortest path.



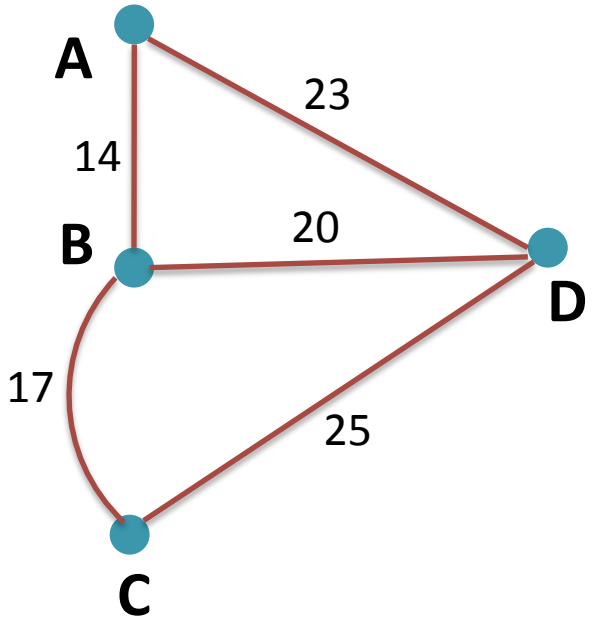
A – B – C
A – E – D – C

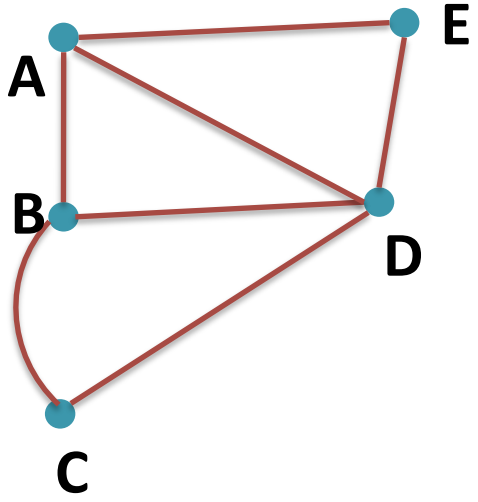


A – B – C : 14+7
A – D – C : 23+25

Looking for the Shortest path

Dijkstra's algorithm

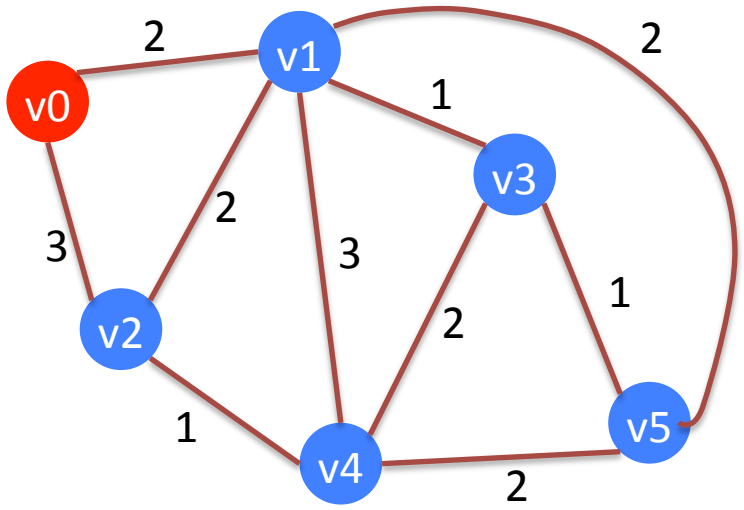




Example of Dijkstra's algorithm

Find the shortest paths start from node "v0".

Insert "v0" into the visited set.



Initialization

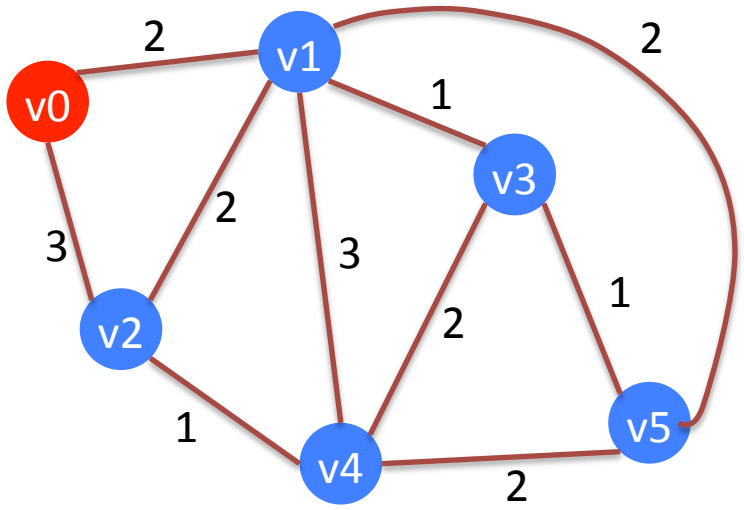
v1	∞	∞	∞	∞	∞
v2	∞	∞	∞	∞	∞
v3	∞	∞	∞	∞	∞
v4	∞	∞	∞	∞	∞
v5	∞	∞	∞	∞	∞

Example of Dijkstra's algorithm

Go through all un-visited neighbors of "v0", and calculate distance between the current value saved in the distance matrix and the sum of edge-weight and the matrix value of the visiting node.

T(v1) = min(T(v1), T(v0)+e01) = min(∞, 2) = 2

T(v2) = min(T(v2), T(v0)+e02) = (∞, 3) = 3



Step 1

v1	2	∞	∞	∞	∞
v2	3	∞	∞	∞	∞
v3	∞	∞	∞	∞	∞
v4	∞	∞	∞	∞	∞
v5	∞	∞	∞	∞	∞

Example of Dijkstra's algorithm

Go through all un-visited neighbors of "v1", and calculate distance between the current value saved in the distance matrix and the sum of edge-weight and the matrix value of the visiting node.

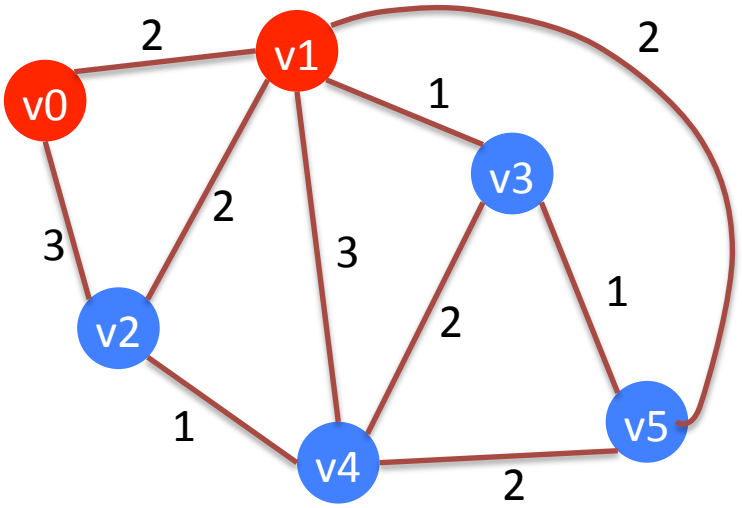
~~T(v0)~~

T(v2) = min(T(v2), T(v1)+e12) = min(3, 2+2) = 3

T(v3) = min(T(v3), T(v1)+e13) = min(∞, 2+1) = 3

T(v4) = min(T(v4), T(v1)+e14) = min(∞, 2+3)=5

T(v5) = min(T(v5), T(v1)+e15) = min(∞, 2+2)=4

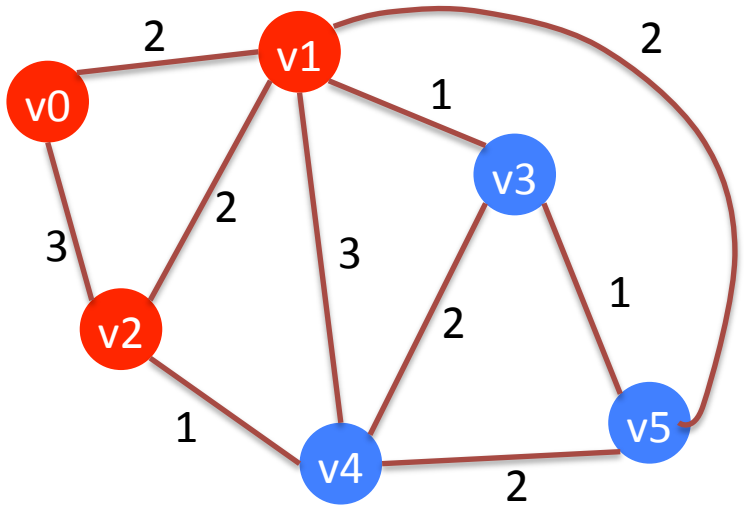


Step 2

v1	**2**	—	—	—	—
v2	3	3	∞	∞	∞
v3	∞	3	∞	∞	∞
v4	∞	5	∞	∞	∞
v5	∞	4	∞	∞	∞

Example of Dijkstra's algorithm

~~T(v0)~~
~~T(v1)~~
T(v4) = min(T(v4), T(v2)+e24) = min(∞, 3+1)=4

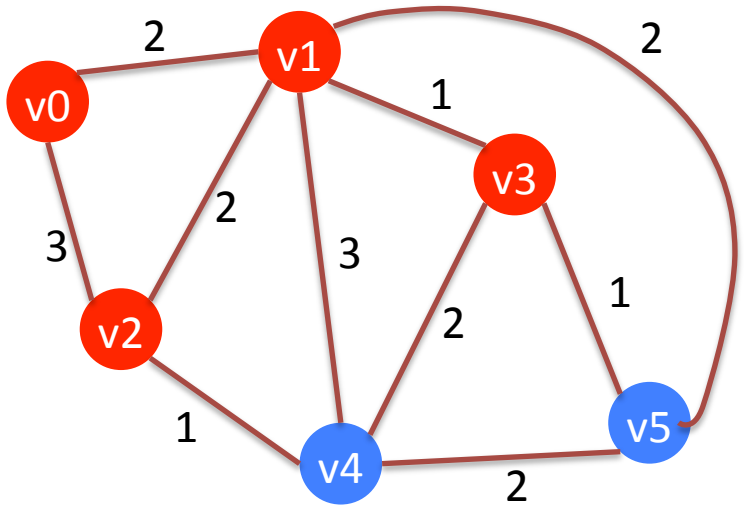


Step 3

v1	2	—	—	—	—
v2	3	**3**	—	—	—
v3	∞	3	3	∞	∞
v4	∞	5	4	∞	∞
v5	∞	4	4	∞	∞

Example of Dijkstra's algorithm

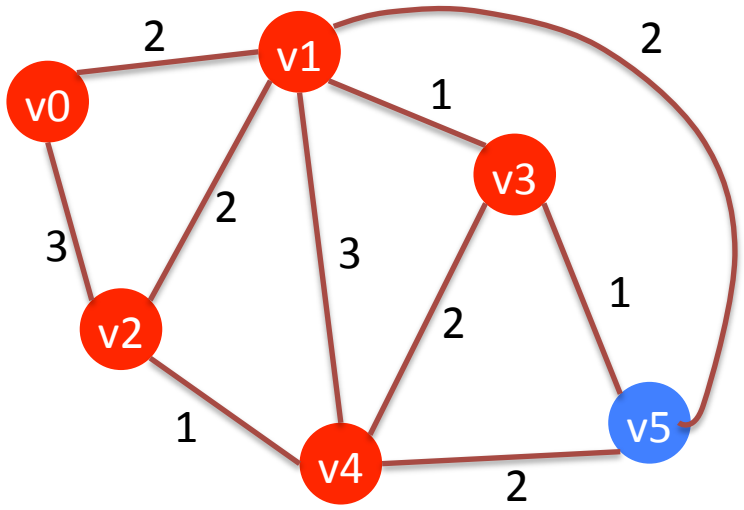
~~T(v1)~~
T(v4) = min(T(v4), T(v3)+e34) = min(4, 3+2) = 4
T(v5) = min(T(v5), T(v3)+e35) = min(4, 3+1) = 4



			Step 4		
v1	2	—	—	—	—
v2	3	3	—	—	—
v3	∞	3	**3**	—	—
v4	∞	5	4	4	∞
v5	∞	4	4	4	∞

Example of Dijkstra's algorithm

~~T(v1)~~
~~T(v2)~~
~~T(v3)~~
T(v5) = min(4, 5+2) = 4



Step 5

v1	2	—	—	—	—
v2	3	3	—	—	—
v3	∞	3	3	—	—
v4	∞	5	4	**4**	—
v5	∞	4	4	4	4