Introduction to Graphs

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실 NETWORK SCIENCE LAB

가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

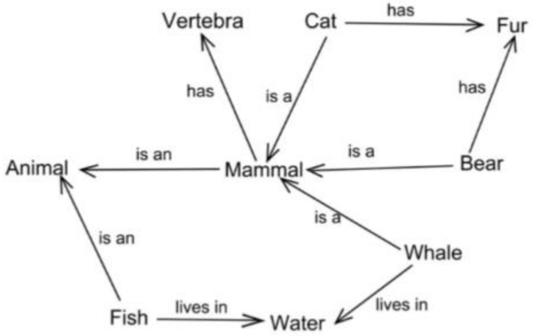
Contents

- The overview of Machine Learning on Graphs
- Graph Terminology
- Graph Characterization
  - Centrality measurements
  - Community
- Graph Kernel.

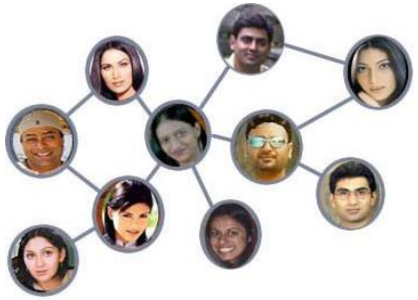
Graphs are everywhere

Networks are a general language for describing and modelling complex systems

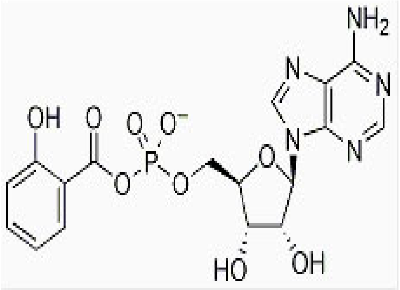
Street network



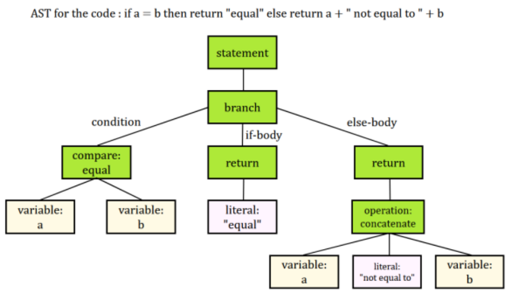
Ecological network



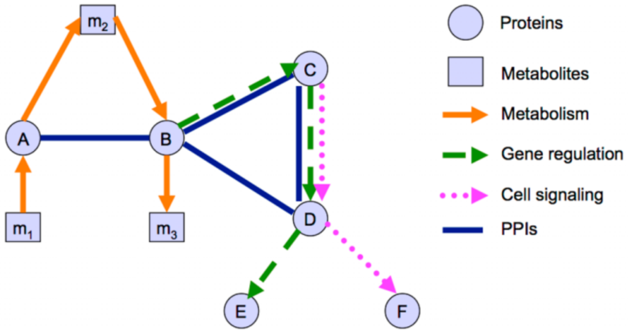
Social media



Chemical network



Program flow



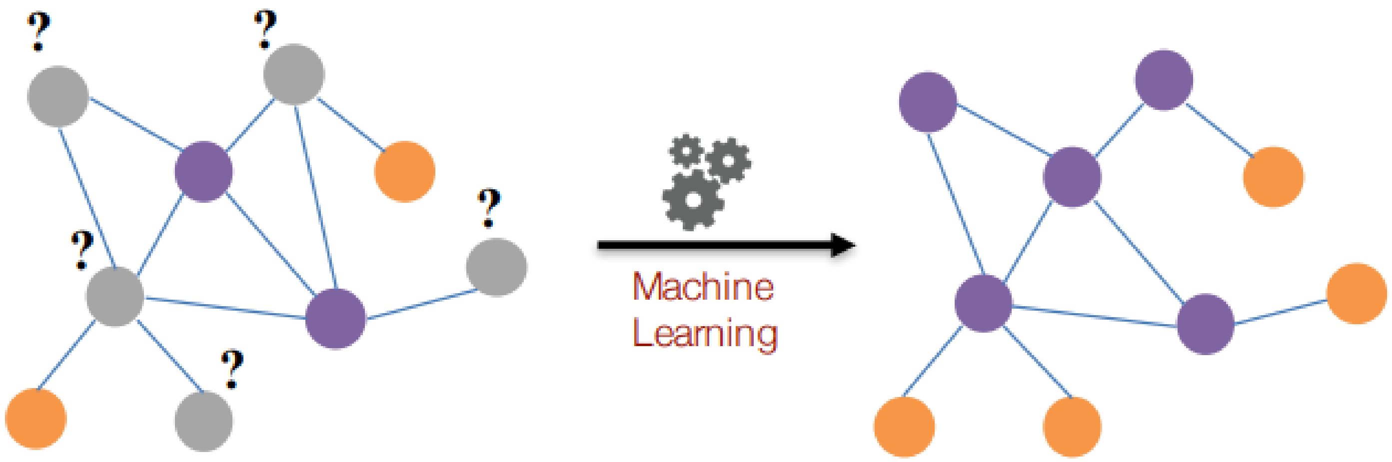
Biological network

Why Graph?

- Universal language for describing complex data
  - Chemical compounds (Cheminformatics)
  - Protein structures, biological pathways/networks (Bioinformactics)
  - Program control flow, traffic flow, and workflow analysis
- Data availability (+computational challenges)
  - Web/mobile, bio-health, and medical data
- Shared vocabulary between fields:
  - Computer science, Social science, Physics, Statistics, Biology
- Impact:
  - Social networking, social media, Drug design

Machine Learning tasks on Graphs

- Node classification
  - Predict a type of a given node

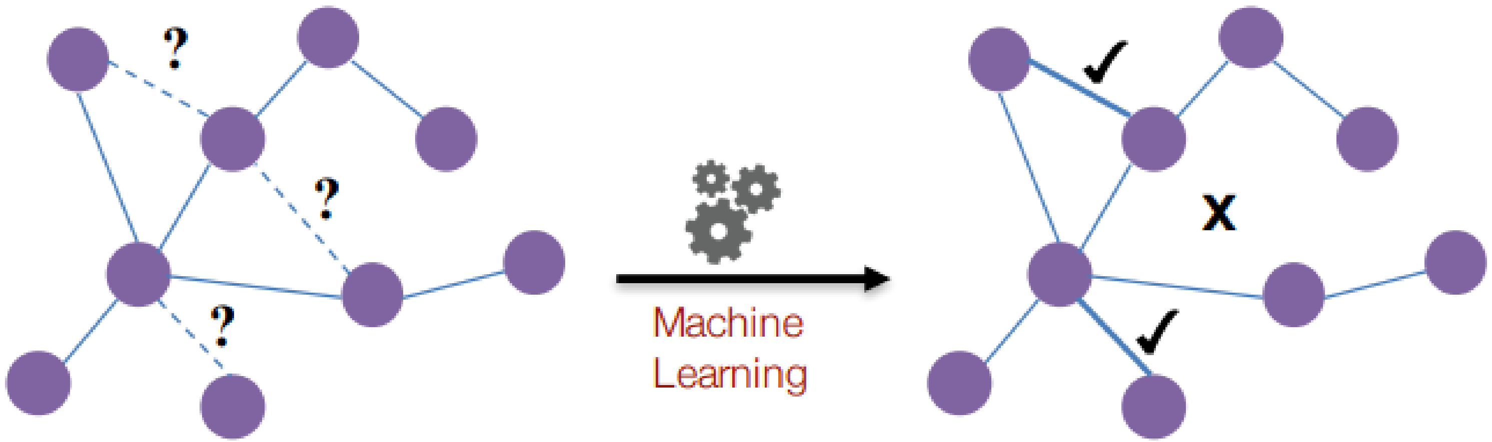


- Many possible ways to create node features:
  - Node degree, PageRank score, motifs

https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/node-representation-learning

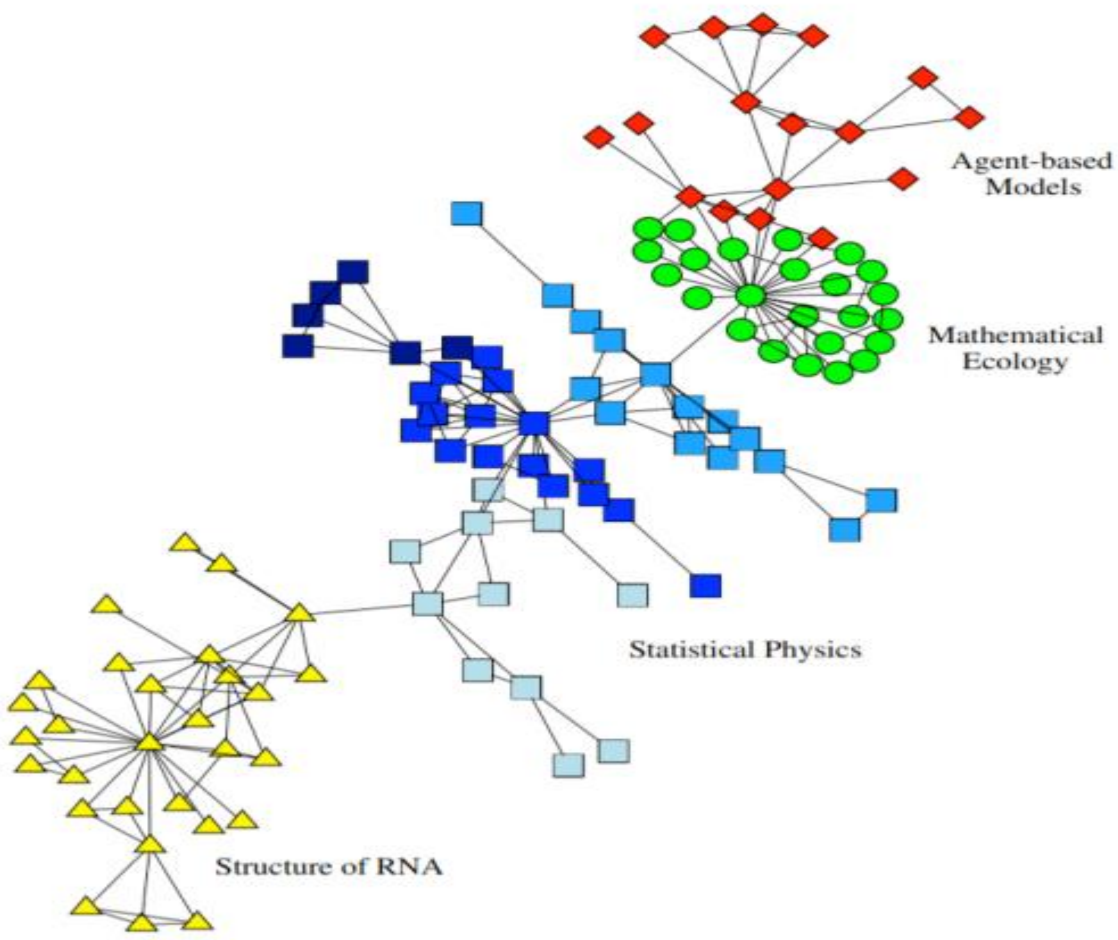
Machine Learning tasks on Graphs

- Link prediction
  - Predict whether two nodes are linked



Machine Learning tasks on Graphs

- Community detection
  - Identify densely linked clusters of nodes



https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications

- Given a graph, we can extract node, edge, graph-level features from the graph, then learn a model to map the features to the desired labels.

Input Graph → Structured Features → Learning Algorithm → Prediction

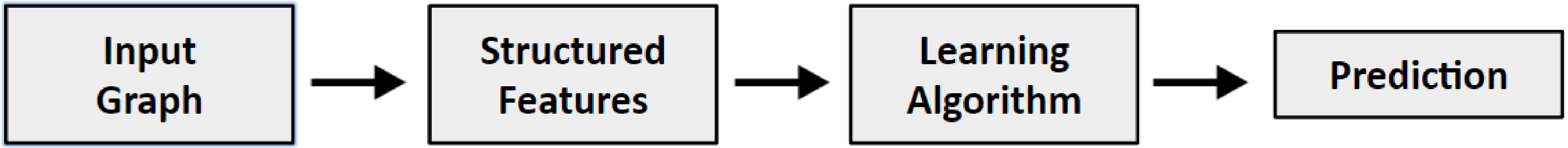
Feature Engineering

- Node feature
- Edge feature
- Graph feature

- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

- Graph Representation Learning aims to generate graph representation vectors that describe graph's structure.
- We don't need to do feature engineering every single time.



Feature Engineering

Representation Learning

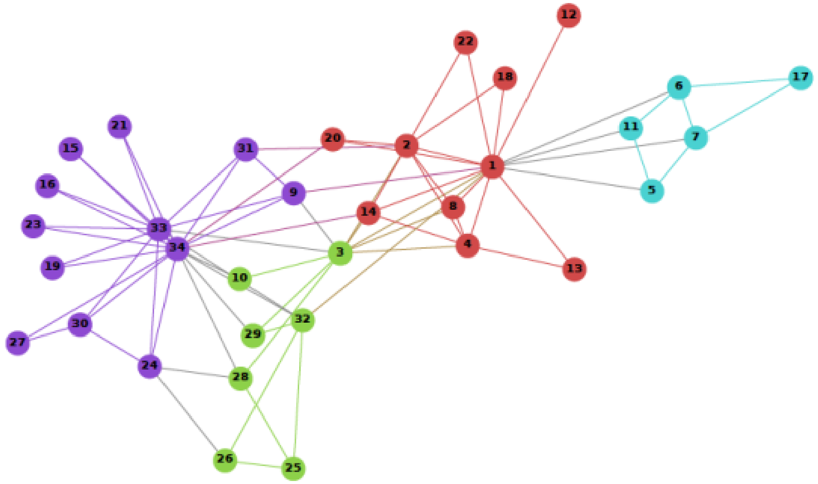
learn the features by itself

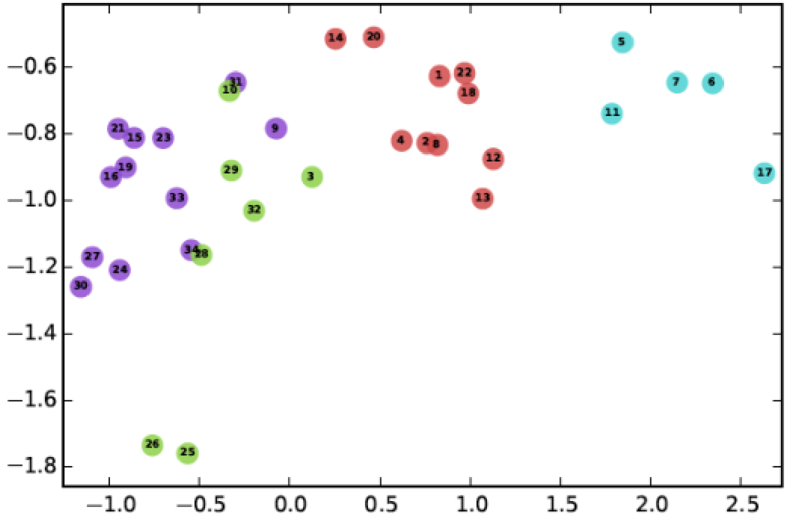
- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

Node Embedding

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.





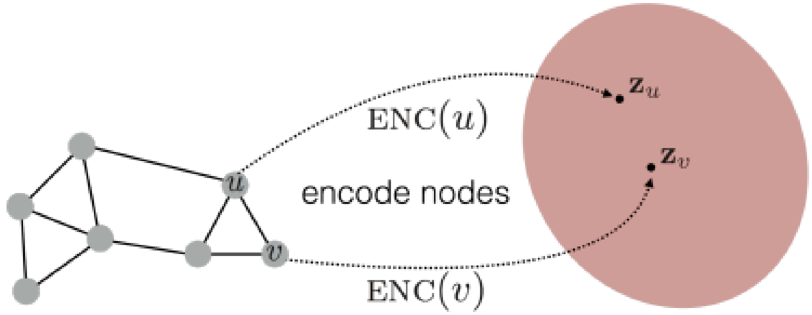
Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 701-710).

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.
- Let z_u be the embedding of node u.
- Goal is to find the encoder function f such that:

$$\text{similarity}(u, v) \approx z_u^T\, z_u$$

- Learning node embedding:
  - Define an encoder .
  - Define a node similarity function.
  - Optimize the parameters of the encoder so that:

$$\text{similarity}(u, v) \approx z_u^T\, z_u$$

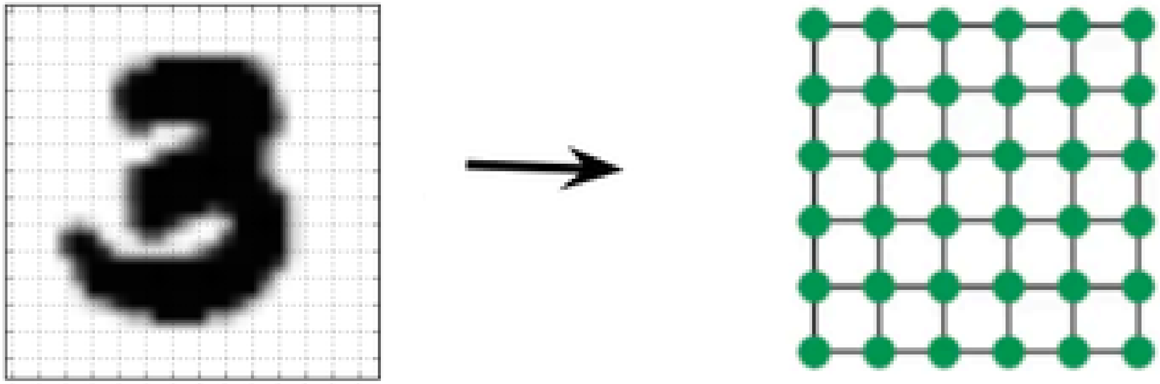


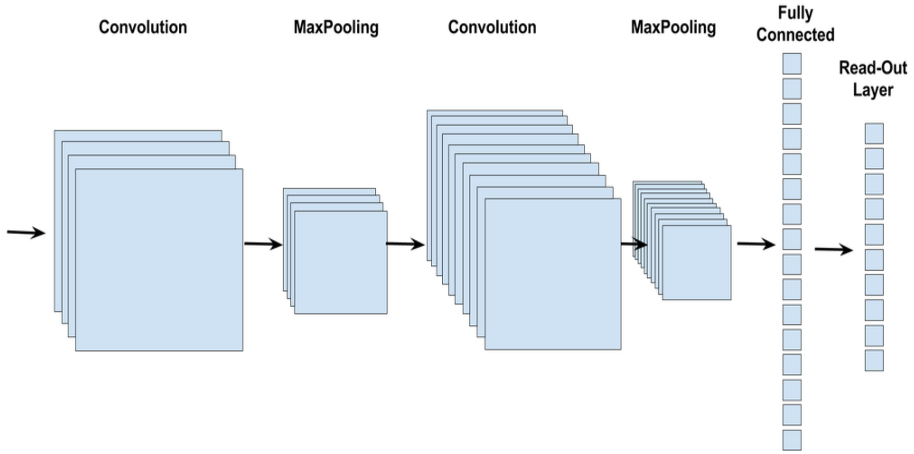
Why learn Embeddings?

- The goal is to map each node into a low-dimensional space
  - Distributed representation for nodes
  - Similarity between nodes indicates link strength
  - Encodes network information and generate node representation

Why is it hard to analyze a graph?

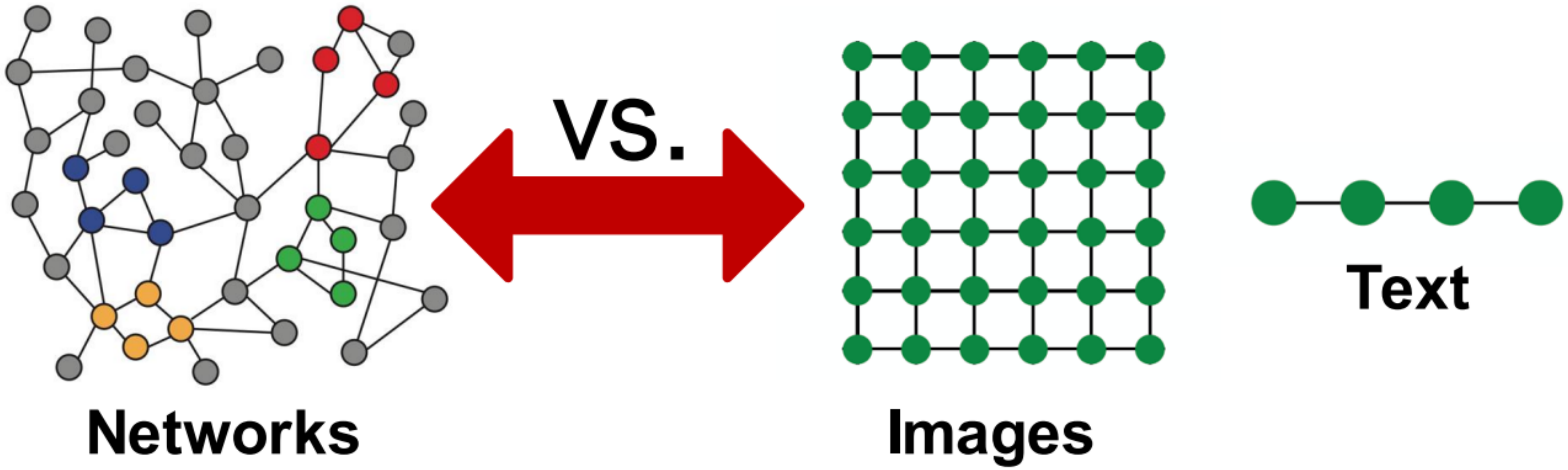
- Graph data is so complex that it's created a lot of challenges for existing machine learning algorithms.
- Images with the same structure and size can be considered as fixed-size grid graphs.
- Text and speech are sequences, so they can be considered as line graphs. (text and speech have linear 1D structure.





Why is it hard to analyze a graph?

- Graphs have arbitrary size and complex topological structure.
- In graphs, there is no fixed node ordering or reference point.
- Graphs are often dynamic and have multimodal features.



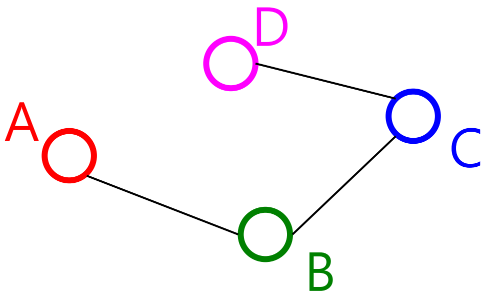
Our goal

- How can we develop neural networks that are much more broadly applicable?
- Feature learning for networks:
  - "Linearizing" the graphs:
    - Create a "sentence" for each node using random walks (node2vec).
      - or proximity: first-order and second-order (LINE).
  - Graph neural networks:
    - Propagate information between the nodes in graphs (message passing).

A graph is a pair: G = (V, E):

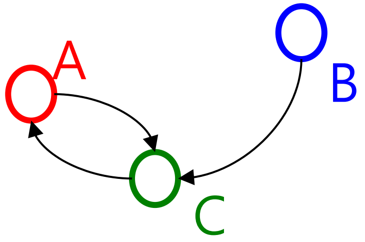
- A set of nodes, also known as nodes: $V = \{v_1, v_2, \ldots, v_n\}$
- A set of edges $E = \{e_1, e_2, \ldots, e_m\}$
  - Each edge e_i is a pair of nodes (v_j, vk).
  - An edge "connects" the nodes.

Graphs can be *directed* or *undirected.*



V = { A, B, C, D}
E = {(A, B), (B, A), (B, C), (C , B),
(C, D), (D, C)}

Undirected



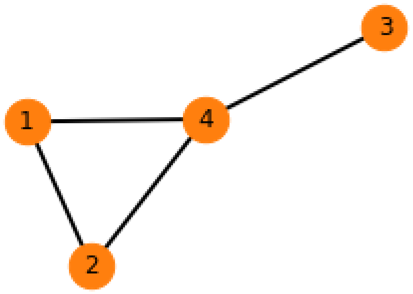
V = { A, B, C }
E = {(B ,C), (A, C), (C , A)}

Directed

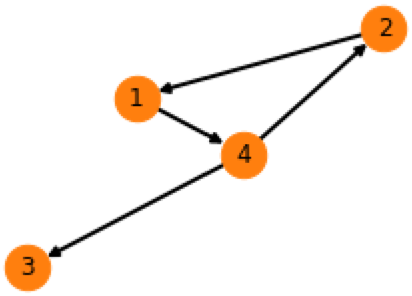
Graph Representations

- Graphs efficiently model relationships, perfect for addressing questions like "What's the lowest-cost path from A to B?”
- We need a data structure that represents graphs
- Determining the "Best" Data Structure can depend on:
  - Properties of the graph (dense vs. sparse)
  - Common queries
    - For example: "is (u ,v) an edge?" vs "what are the neighbors of node u?"
- There are two standard graph representations:
  - Adjacency Matrix.
  - Adjacency List.

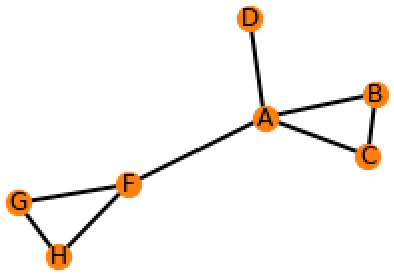
Graph Components and Types



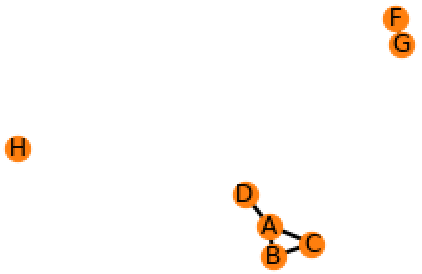
Undirected



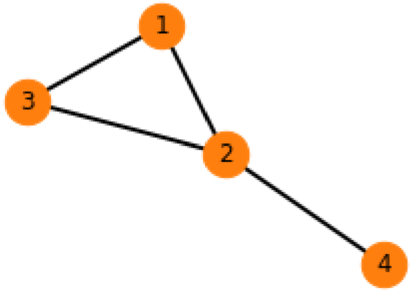
Directed



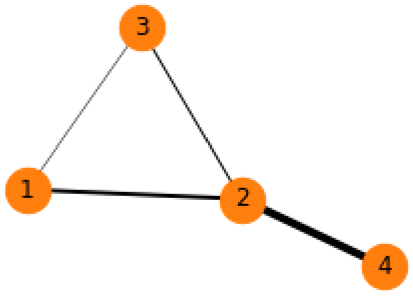
Connected



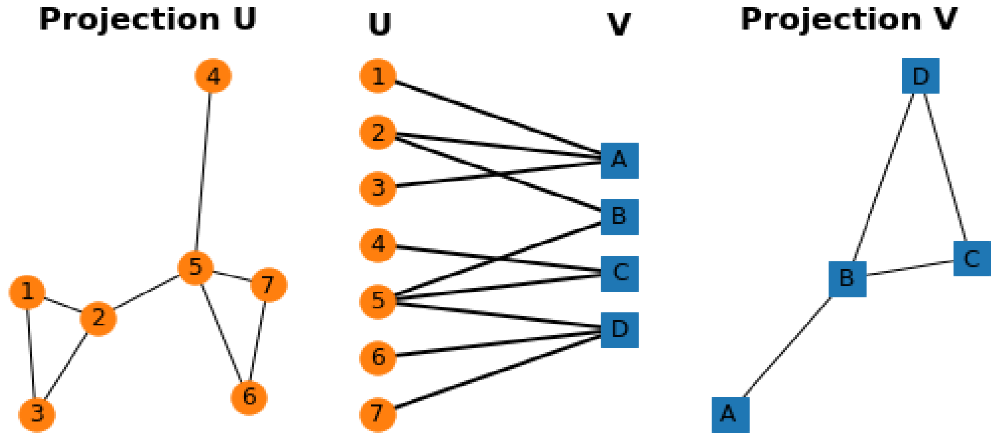
Disconnected



Unweighted



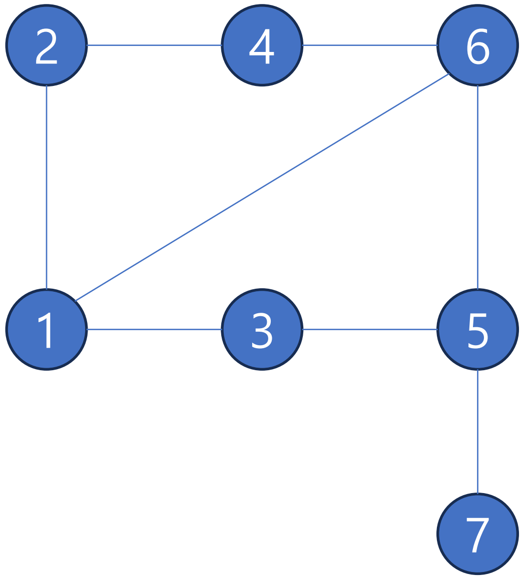
Weighted



Folded/Projected Bipartite Graphs

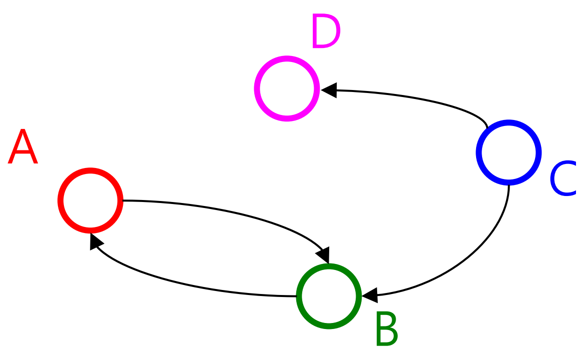
Diameter

- The diameter of a graph is the length of the shortest path between the most distanced nodes.
- d measures the extent of a graph and the topological length between two nodes.



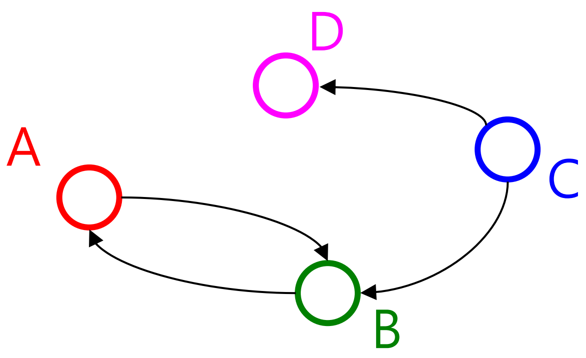
Diameter of this graph is 4

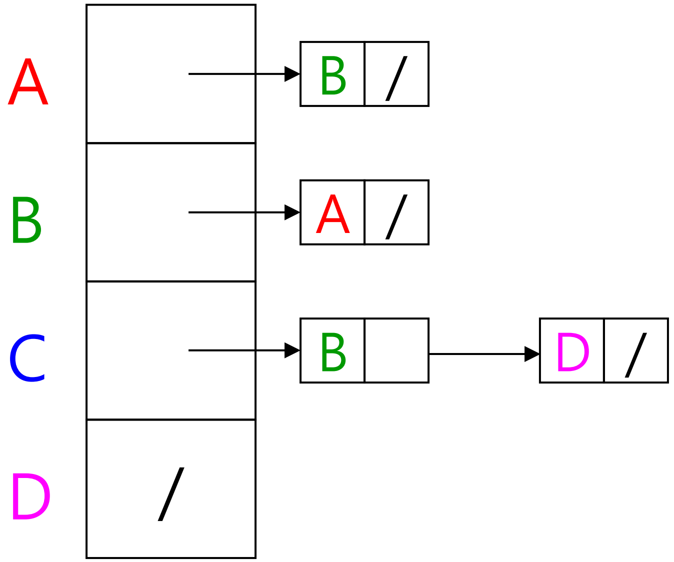
- Adjacency Matrix
  - Assign each node a number from 0 to |V|-1
  - A |V| x |V| matrix of Booleans (or 0 vs. 1)
    - Then M[u][v] == true means there is an edge from u to v.



	A	B	C	D
A	0	1	0	0
B	1	0	0	0
C	0	1	0	1
D	0	0	0	0

- Adjacency List





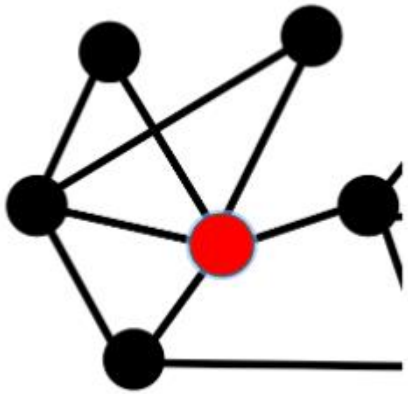
Adjacency List Properties

- Running time to:
  - Get a vertex's out-edges: O(*d*) where *d* is out-degree of vertex
  - Get a vertex's in-edges: O(|E|) (could keep a second adjacency list for this!)
  - Decide if some edge exists: O(*d*) where *d* is out-degree of source
  - Insert an edge: O(1) (unless you need to check if it's already there)
  - Delete an edge: O(*d*) where *d* is out-degree of source

- Space requirements: O(|V|+|E|)

- Best for sparse or dense graphs?
  - Sparse graphs.

- Knowing the network structure, we can calculate various useful quantities or measures that capture features of network topology
- Centrality measures represent the most important nodes in graphs:
  - The most influential person in a social network.
  - The most critical nodes in an infrastructure.
  - The highest spreaders of disease.
- Several common measurements:
  - Degree centrality
  - Betweenness centrality
  - Closeness centrality
  - Eigenvector centrality
  - PageRank



Degree Centrality Examples

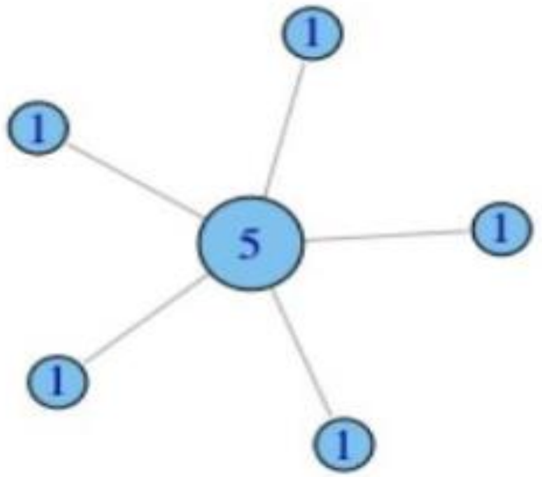
- Using Freeman's general formula for centralization (which ranges from 0 to 1):

$$C_D(G) = \frac{\sum_{i=1}^{n}\left[C_D(v^*) - C_D(v_i)\right]}{(\mathrm{n}-1)(\mathrm{n}-2)},$$

$$where:$$

$$v^*: \text{ the node with the highest degree in G}$$

$C_D = 0.167$



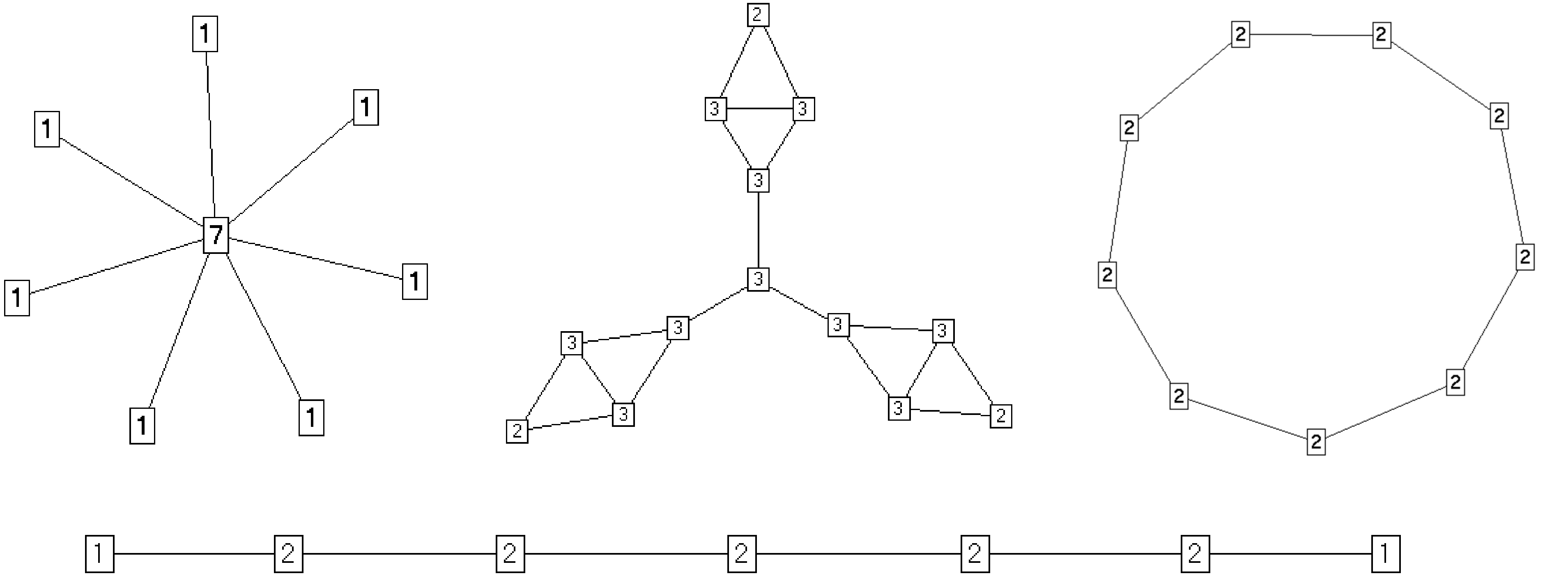
$C_D = 1.0$

$$C_D(G) = \frac{(2-1)+(2-0)+...+(2-1)}{(5-1)(5-2)} = 0.167$$

$$C_D(G) = \frac{(5-1)+...+(5-1)}{(6-1)(6-2)} = \frac{20}{20} = 1$$

Degree Centrality

- The most intuitive notion of centrality focuses on degree:
  - The actor with the most ties is the most important:



$$C_D(v_i) = d(v_i) = \sum_{j=1}^{n} A_{ij}$$

- Betweenness Centrality of node v_i:
  - A node is important if it **lies on many shortest paths** between other nodes.

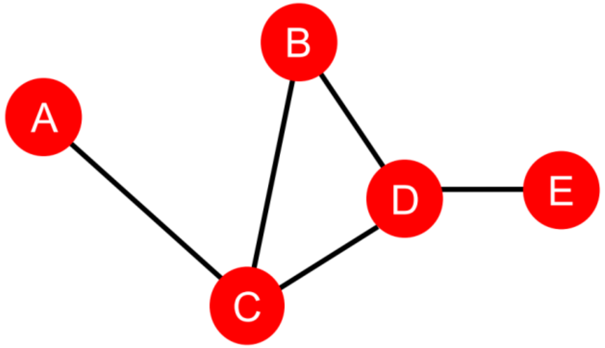
$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths betwen } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

  - Usually normalized by:

No. of nodes in the graph

$$\bar{B}(v_i) = B(v_i) / [(n-1)(n-2)/2]$$

  - For example:



- $C_A = C_B = C_E = 0$
- $C_C = 3$ (A-C-B, A-C-D, A-C-D-E)
- $C_D = 3$ (A-C-D-E, B-D-E, C-D-E)

Betweenness Centrality

- Vertices with high betweenness centrality have influence in the network by virtue of their control over information passing between others.
  - They get to see the messages as they pass through
  - They could get paid for passing the message along
- Thus, they get a lot of power: their removal would disrupt communication

Closeness Centrality

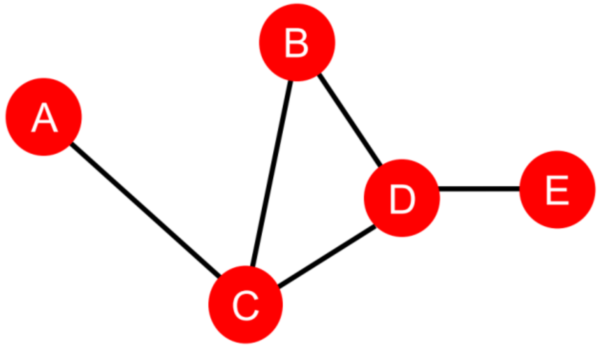
- **Definition:**
  - A node is important if it has **small shortest path lengths** to all other nodes.

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

- Usually normalized by:

$$\bar{C}(v_i) = \frac{\mathrm{n} - 1}{\sum_{j=1}^{n} \left| \mathrm{d}(v_i, v_j) \right|}$$

- For example:



- $C_A = 1/(2 + 1 + 2 + 3) = 1/8$

(A-C-B, A-C, A-C-D, A-C-D-E)

- $C_D = 1/(2 + 1 + 1 + 1) = 1/5$

(D-C-A, D-B, D-C, D-E)

Eigenvector centrality

- Define the centrality x_i' of i recursively in terms of the centrality of its neighbors:

$$x_i' = \sum_{v_j \in N(v_i)} A_{ij} x_j \qquad \text{with the initial node centrality } x_j = 1, \forall j$$

- That is equivalent to:

$$x_i(t) = \sum_{v_j \in N(v_i)} A_{ij} x_j(t-1) \qquad \text{with the centrality at time t=0 being } x_j(0) = 1, \forall j$$

The centrality of nodes x_i and x_j at time *t* and *(t-1)*, respectively.

Katz centrality

- Katz centrality computes the centrality for a node based on the centrality of its neighbours. It is a generalization of the eigenvector centrality.
- The Katz centrality for node v_i is:

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where:

α is a constant called damping factor, and β is a bias constant,

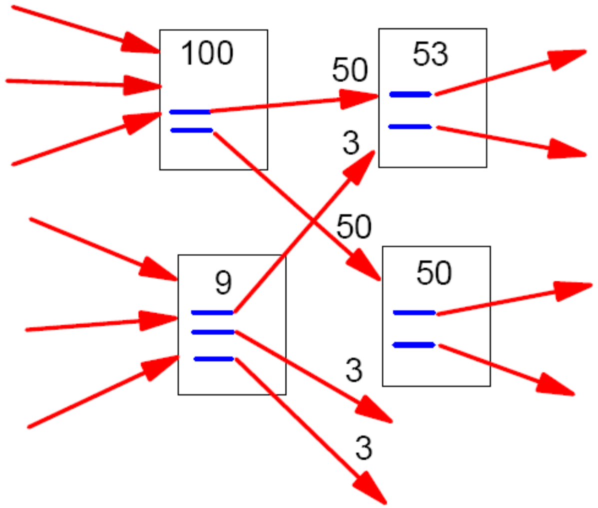
A is the adjacency matrix.

- When $\alpha = 1/\lambda_{\max}, \beta = 0$, Katz centrality is the same as eigenvector centrality.

- PageRank is a numeric value that represents how important a page is on the web.
- Webpage importance
  - One page links to another page = A vote for the other page A link from page A to page B is a vote on A to B.
  - If page A is more important itself, then the vote of A to B should carry more weight.
  - More votes = More important the page must be.
- How can we model this importance?

Simplified PageRank: A simplified version from Google

- Importance Computation
  - The importance of a page is distributed to pages that it points to.
  - The importance of a page is the aggregation of the importance shares of the pages that points to it.
    - If a page has 5 outlinks, the importance of the page is divided into 5 and each link receives one fifth share of the importance.

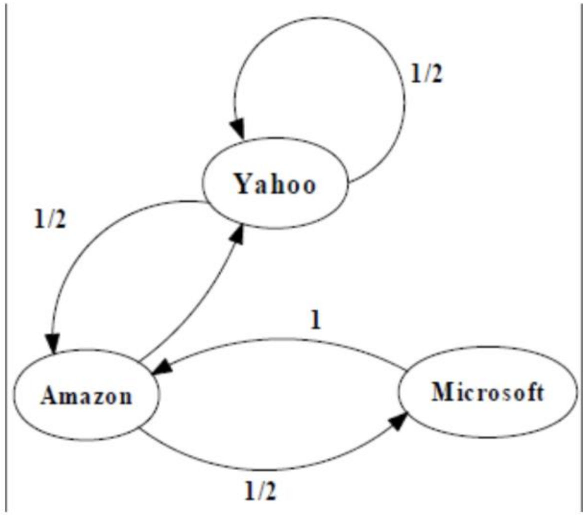


A Simple Version of PageRank

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

- u: a web page.
- B_u: the set of u's backlinks.
- N_v: the number of forward links of page v.
- c: the normalization factor to make $\|R\|_{L1} = 1 (\|R\|_{L1} = |R_1 + \cdots + R_n|)$.

➢ For example:

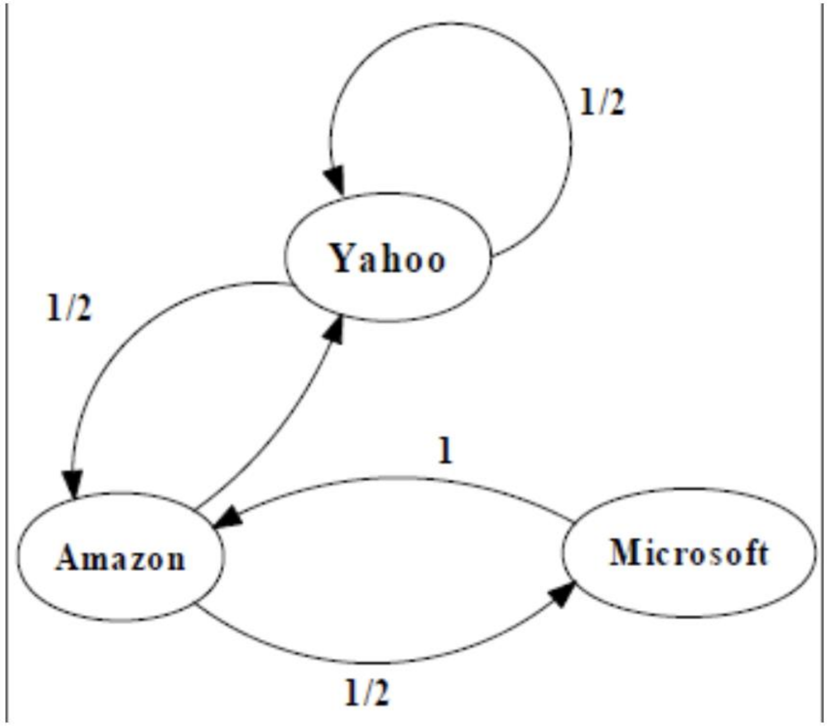


$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

A Simple Version of PageRank



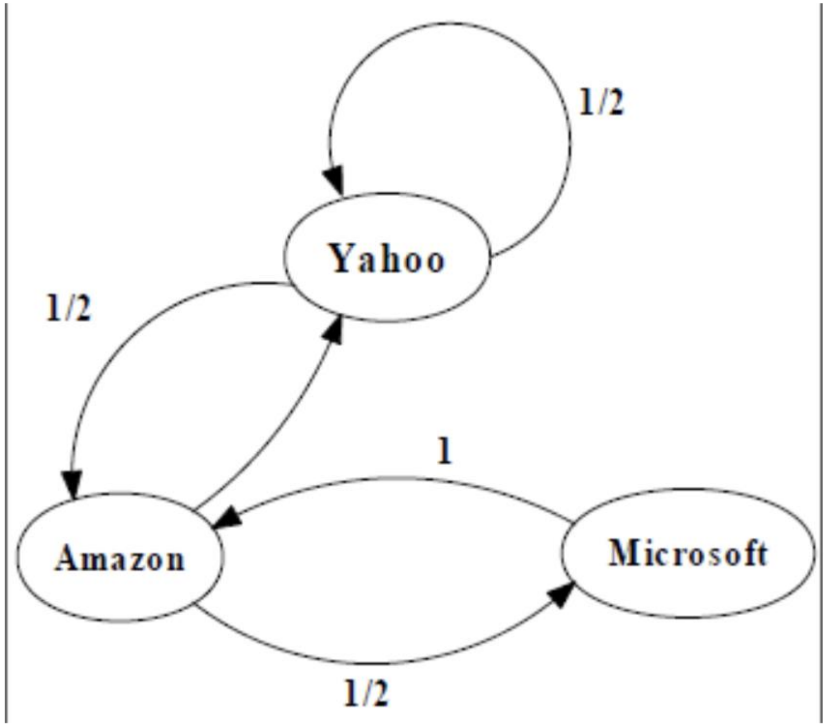
$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 5/12 \\ 1/3 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$$

PageRank Calculation: second iteration

A Simple Version of PageRank



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 3/8 \\ 11/24 \\ 1/6 \end{bmatrix} \begin{bmatrix} 5/12 \\ 17/48 \\ 11/48 \end{bmatrix} \ldots \begin{bmatrix} 2/5 \\ 2/5 \\ 1/5 \end{bmatrix}$$

Convergence after some iterations

Taxonomy of Community Criteria

- Criteria vary depending on the tasks.
- Roughly, community detection methods can be divided into 4 categories (not exclusive):
  - 1. Node-Centric Community:
    - Each node in a group satisfies certain properties.
  - 2. Group-Centric Community:
    - Consider the connections within a group as a whole. The group must satisfy certain properties without zooming into node-level.
  - 3. Network-Centric Community:
    - Partition the whole network into several disjoint sets.
  - 4. Hierarchy-Centric Community:
    - Construct a hierarchical structure of communities.

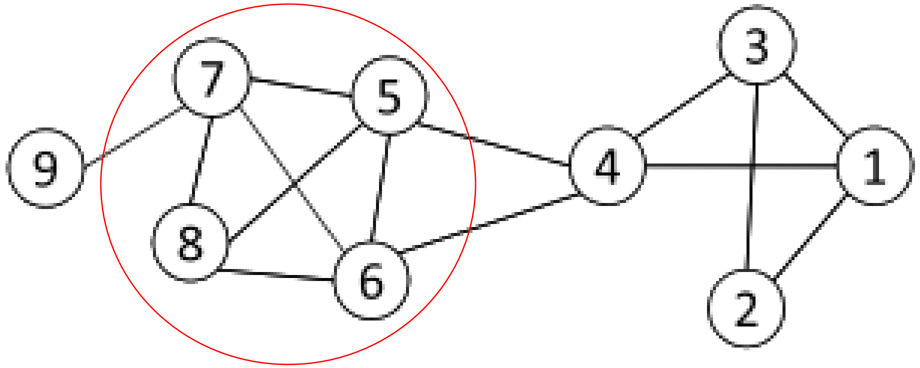
Node-Centric Community Detection

- Nodes satisfy different properties
  - Complete Mutuality
    - cliques
  - Reachability of members
    - k-clique, k-clan, k-club
  - Nodal degrees
    - k-plex, k-core
  - Relative frequency of Within-Outside Ties
    - LS sets, Lambda sets
- Commonly used in traditional social network analysis

We discuss some representative ones

Complete Mutuality: Cliques

- Clique: a maximum complete subgraph in which all nodes are adjacent to each other



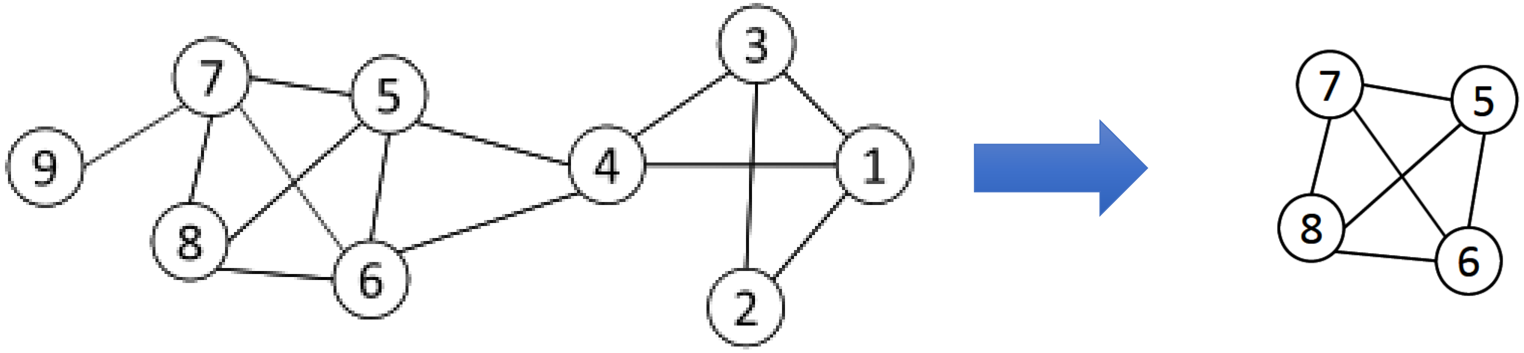
Nodes 5, 6, 7 and 8 form a clique

- NP-hard to find the maximum clique in a network
- Straightforward implementation to find cliques is very expensive in time complexity

Finding the Maximum Clique

- In a clique of size k, each node maintains degree $\geq k - 1$.
- Nodes with degree $< k - 1$ will not be included in the maximum clique.
- Recursively apply the following pruning procedure:
  - Sample a sub-network from the given network, and find a clique in the sub-network, say, by a greedy approach.
  - Suppose the clique above is size k.
  - To find out a larger clique, all nodes with degree $\leq k - 1$ should be removed.
- Repeat until the network is small enough.
- Many nodes will be pruned as social media networks follow a power law distribution for node degrees.

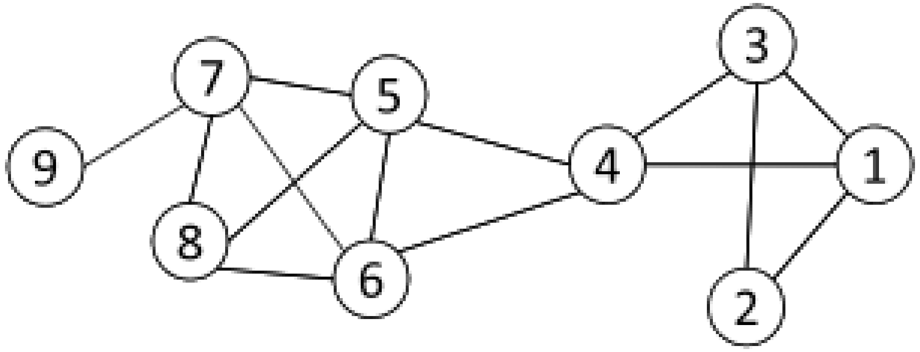
- Suppose we sample a sub-network with nodes $\{1 - 5\}$ and find a clique $\{1, 2, 3\}$ of size 3
- To find a clique > 3, remove all nodes with degree $\leq\ 3 - 1\ =\ 2$
  - Remove nodes 2 and 9
  - Remove nodes 1 and 3
  - Remove node 4



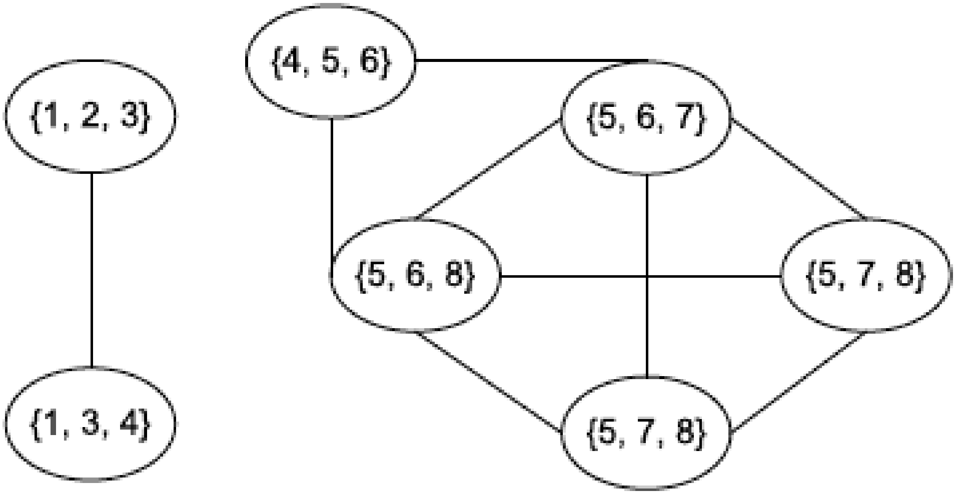
Clique Percolation Method (CPM)

- Clique is a very strict definition, unstable.
- Normally use cliques as a core or a seed to find larger communities.

- CPM is such a method to find overlapping communities.
- Input:
  - Parameter k and a network.
- Procedure
  - Given a network, find out all cliques of size k.
  - Construct a clique graph. Two cliques are adjacent if they share $k-1$ nodes.
  - Each connected components in the clique graph form a community.



Cliques of size 3:
{1, 2, 3}, {1, 3, 4}, {4, 5, 6}, {5, 6, 7}, {5, 6, 8}, {5, 7, 8}, {6, 7, 8}.



Communities:
+ {1, 2, 3, 4}.
+ {4, 5, 6, 7, 8}.

Clustering Coefficient

- The clustering coefficient measures how connected a node's neighbors are to one another.
- The range is from 0 to 1 (from non-neighbor are connected to each other to all neighbors are fully connected).
- There are three types of clustering coefficient:
  - Local clustering coefficient.
  - Average clustering coefficient.
  - Global clustering coefficient.

Local Clustering Coefficient

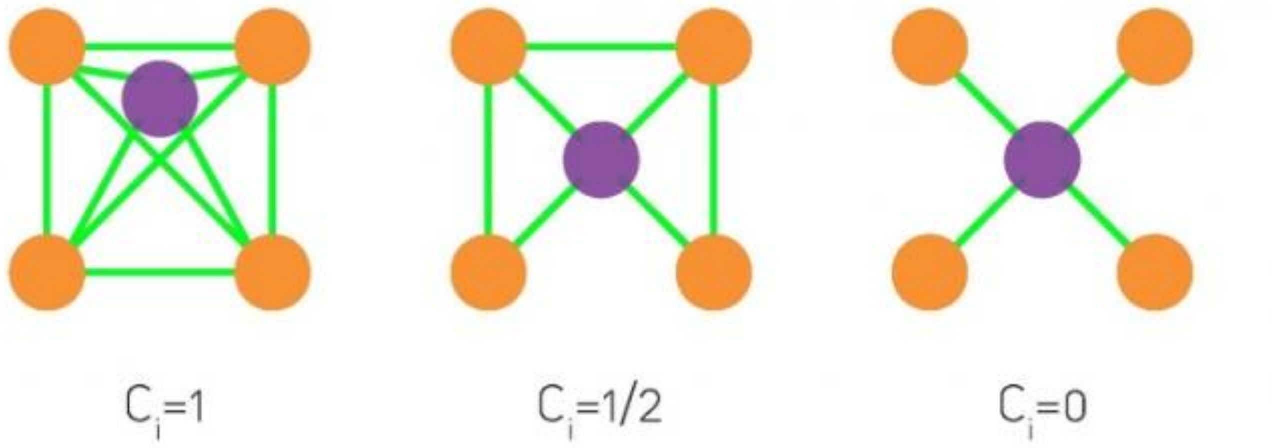
- How close its neighbours are to being a clique (complete graph).
- For a node i with degree d_i and L_i represents the number of edges between neighbors of node i.

The local clustering coefficient C_i for a node i is defined as:

$$C_i = \frac{2L_i}{d_i(d_i - 1)}$$

50% chance that two neighbors of a node i are linked

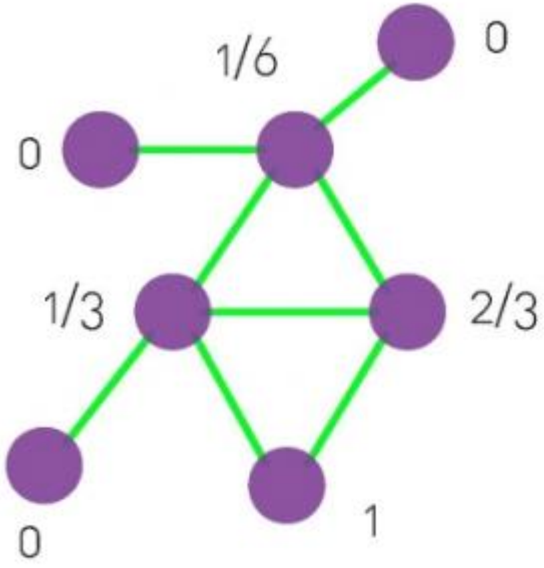
neighbors of node i form a complete graph



None of neighbors of node i link to each other

- The degree of clustering of a whole network is captured by the average clustering coefficient, namely $\langle C \rangle$, representing the average of all the local clustering coefficient C_i over all nodes $i = 1, \dots, N$.

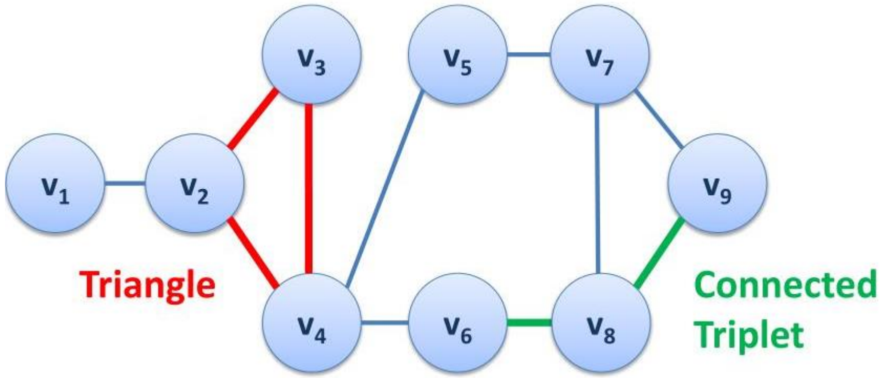
$$\langle C \rangle = \frac{1}{N} \sum_{i=0}^{N} C_i$$



$$\langle C \rangle = \frac{1}{7} * \left(0 + \frac{1}{6} + \frac{1}{3} + \frac{2}{3} + 1 + 0 + 0 \right) = 0.333$$

Global Clustering Coefficient

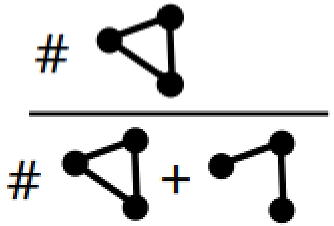
- The global clustering coefficient is based on triplets of nodes.
- A triplet consists of three connected nodes. A triangle therefore includes three closed triplets, one centered on each of the nodes.



- The global clustering coefficient is the number of closed triplets over the total number of triplets (both open and closed)

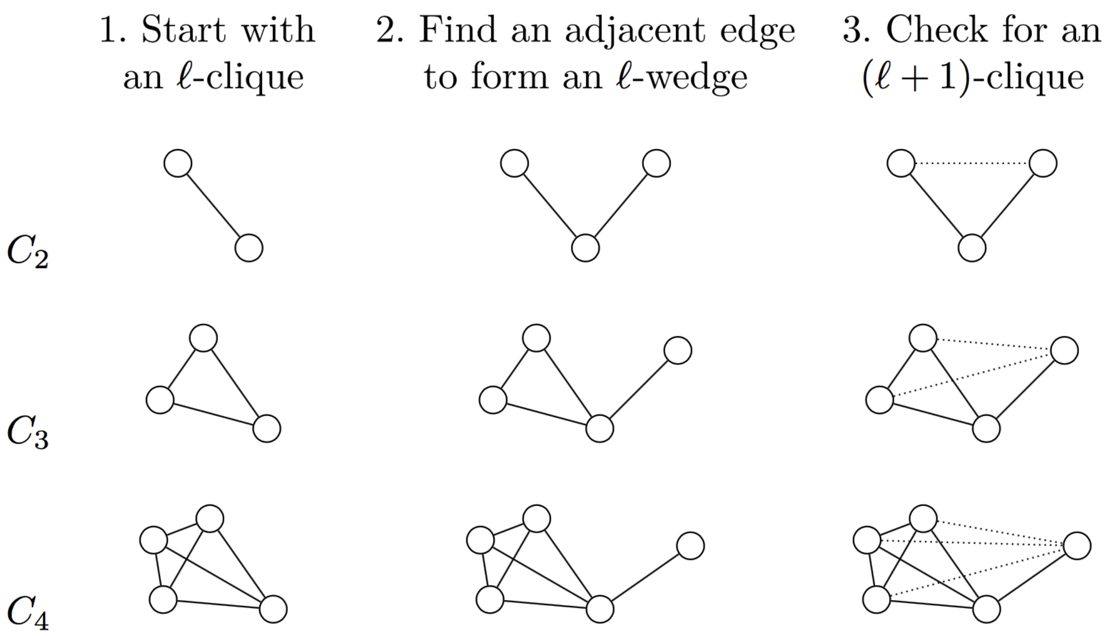
Closed triplets: $(\mathrm{v}_2, \mathrm{v}_3, \mathrm{v}_4)$, $(\mathrm{v}_7, \mathrm{v}_8, \mathrm{v}_9)$
Connected triplets: $(\mathrm{v}_6, \mathrm{v}_8, \mathrm{v}_9), \cdots$

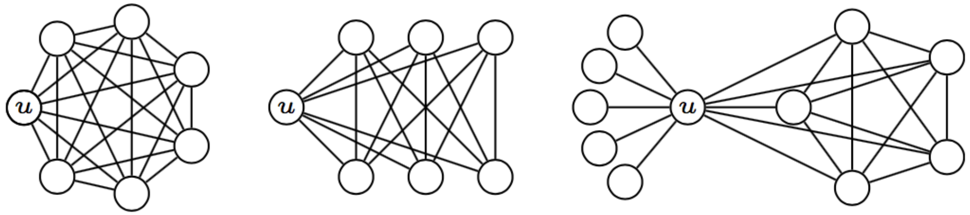
$$C(G) = \frac{\#of\ closed\ triplets}{\#\ of\ connected\ triplets}$$



Higher-order Clustering Coefficient

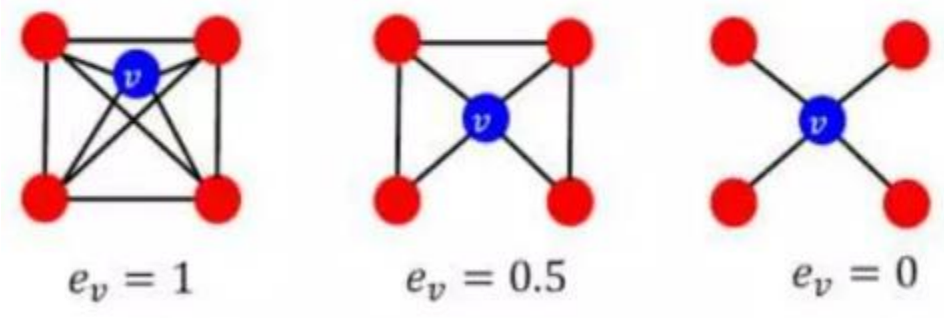
- The clustering coefficient can be extended to higher order structures with k-cliques.



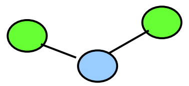
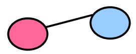
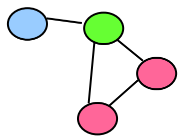


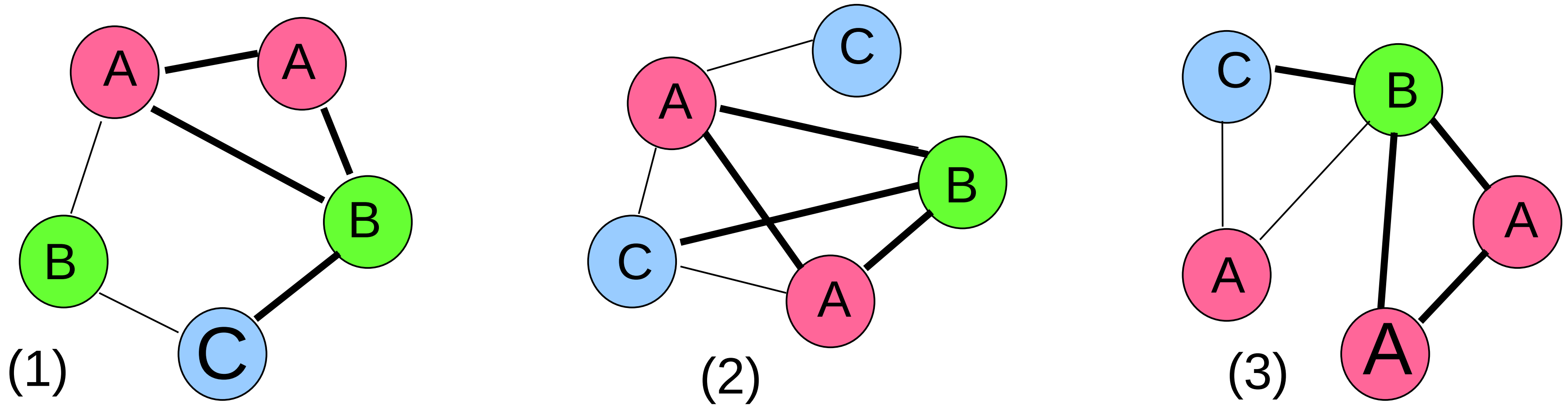
$C_2(u)$	1	$\frac{d}{2(d-1)} \approx \frac{1}{2}$	$\frac{d-2}{4d-4} \approx \frac{1}{4}$
$C_3(u)$	1	0	$\frac{d-4}{2d-4} \approx \frac{1}{2}$
$C_4(u)$	1	0	$\frac{d-6}{2d-6} \approx \frac{1}{2}$

- Degree of nodes: in-degree, out-degree, and total degree.
- Node centrality measurement: betweenness, closeness, eigenvector, katz, etc.
- Clustering Coefficient
  - Measures how connected neighboring nodes are
  - E.g., The number of edges among neighboring nodes



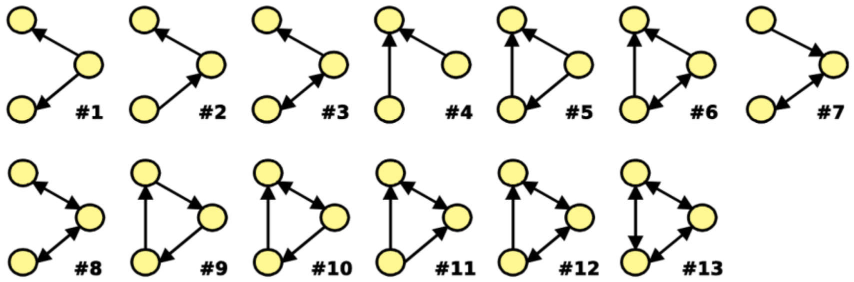
- Frequent subgraphs
  - A (sub)graph is frequent if its support (occurrence frequency) in a given dataset is no less than a minimum support threshold
  - Suppose t = 2, the frequent subgraphs are (only edge labels)
    - a, b, c
    - a-a, a-c, b-c, c-c
    - a-c-a …

Support	1	3	3
Subgraph			



Graph Features: Motifs

- Given a network, most of the time, some subgraphs are “overrepresented”.
- A connected graph that has many occurrences in a network is called a motif of the network.
- Assume set of occurrences G' in G is $occ_G(H)$.
  - cardinality of $occ_G(H)$ in G is frequent.
  - How to know if G' is frequent in G?



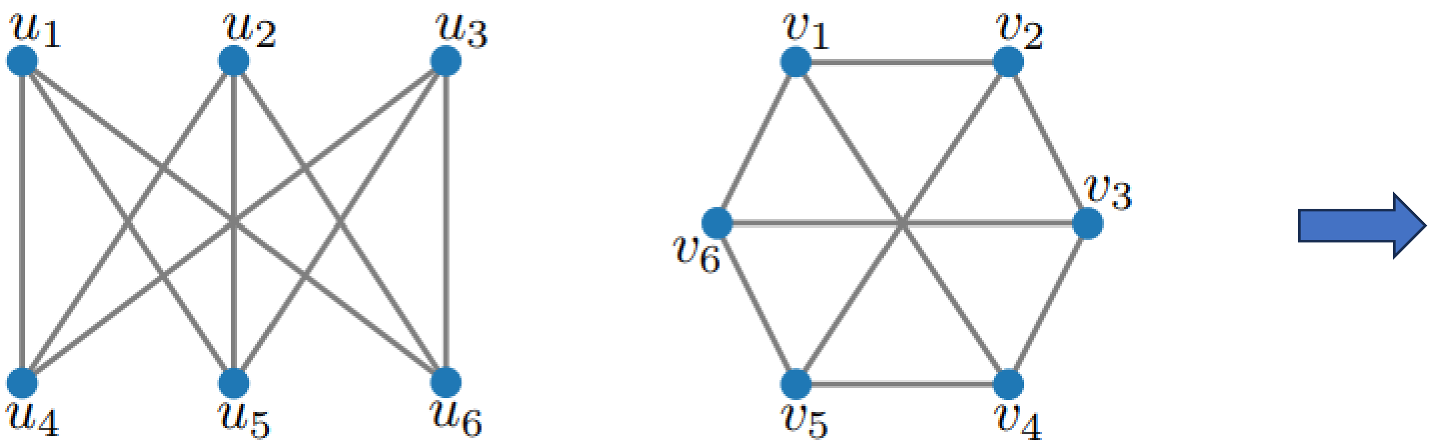
Compute the probability that $occ_N(G') \geq occ_G(G')$ for a random network N.

G' is said to be frequent in G if this probability is small enough.

To compute this probability, we need to have a distribution over networks.

Graph Isomorphism

- Graph isomorphism:
  - Find a mapping f of the vertices of G_1 to the vertices of G_2 such that G_1 and G_2 are identical;
  - i.e. (u_i, u_j) is an edge of G_1 iff $(f(u_i), f(u_j))$ is an edge of G_2. Then f is an isomorphism, and G_1 and G_2 are called isomorphic.
  - No polynomial-time algorithm is known for graph isomorphism.
  - Neither is it known to be NP-complete.

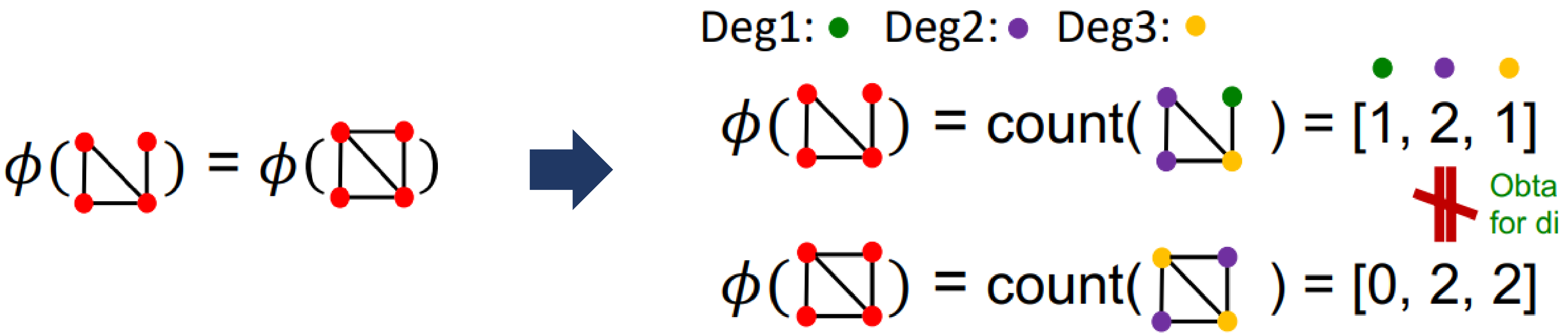


$$
\begin{array}{ccc}
 & V^{G_1} & V^{G_2} \\
 & v_1 \to & u_1 \\
 & v_2 \to & u_4 \\
\psi : & v_3 \to & u_2 \\
 & v_4 \to & u_5 \\
 & v_5 \to & u_3 \\
 & v_6 \to & u_6
\end{array}
$$

Graph Features: What is a Kernel? (Schölkopf ,1997)

- Kernel is a type of measures of similarity.
- Mapping two objects x and x' via mapping $\boxed{\phi}$ into feature space *H.*
- Measure their similarity in *H* as $\langle \phi(x), \phi(x') \rangle$.
- **Kernel Trick**: Compute inner product in *H* as kernel in input space

$$\boxed{k(x, x') = \langle \phi(x), \phi(x') \rangle.}$$



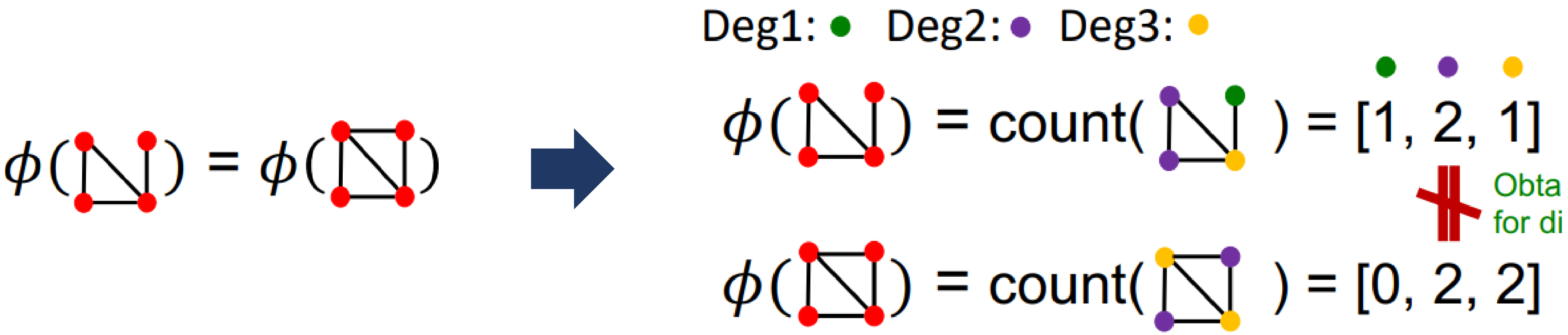
- Instance of R-convolution kernels by Haussler (1999):
  - R-convolution kernels compare decompositions of two structured objects.

$$k_{convolution}(x, x') = \sum_{(x_d, x) \in R} \sum_{(x'_d, x') \in R} k_{parts}(x_d, x'_d)$$

  - Decompose graphs into their substructures and add up the pairwise similarities between these substructures.
- Concept:
  - Kernel function to measure the similarity of pairs of graphs by computing an inner product on graphs.
  - Compare substructures of graphs that are computable in polynomial time.

Graph Features: Graph Kernel

- Graph kernels based on bags of patterns:
  - Extraction of a set of patterns from graphs
  - Comparison between patterns
  - Comparison between bags of patterns

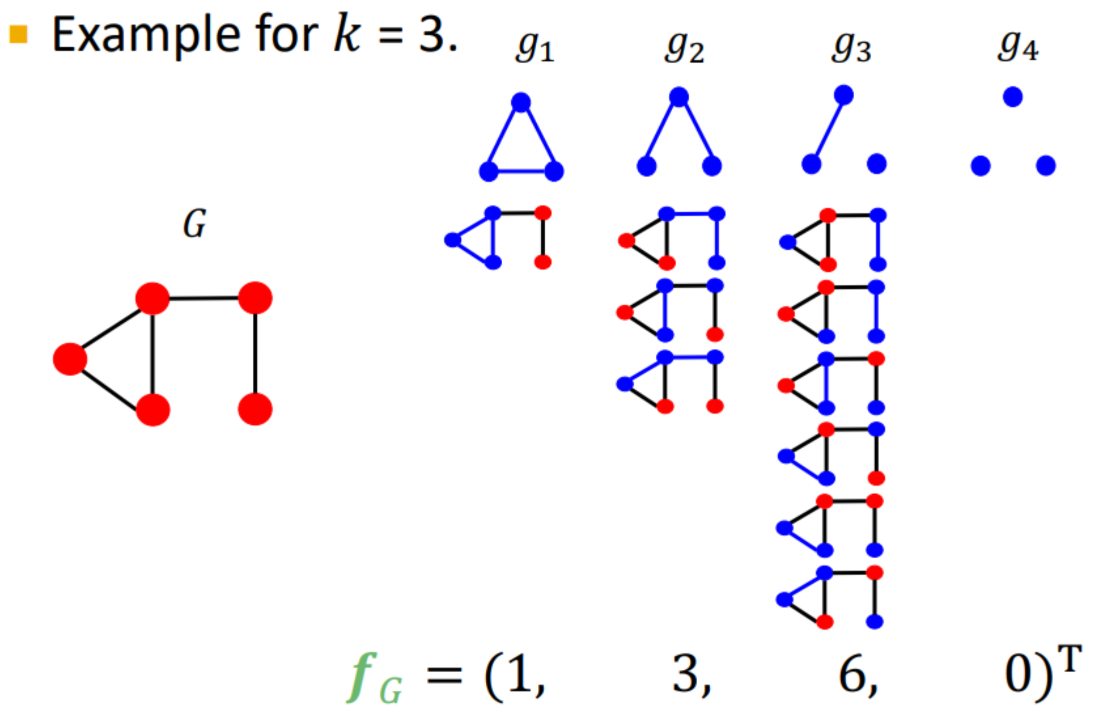


Graph Features: Why Graph Kernels?

- Graph kernels are one of the most recent approaches to graph comparison.
- Interestingly, graph kernels employ concepts from all three traditional branches of graph comparison:
  - Measure similarity in terms of isomorphic substructures of graphs.
  - Allow for inexact matching of nodes, edges, and labels.
  - Treat graphs as vectors in a Hilbert space of graph features.

Graph Features: Graphlet

- Graphlet Kernel (B., Petri, et al., MLG 2007)
- Count subgraphs of limited size 3:



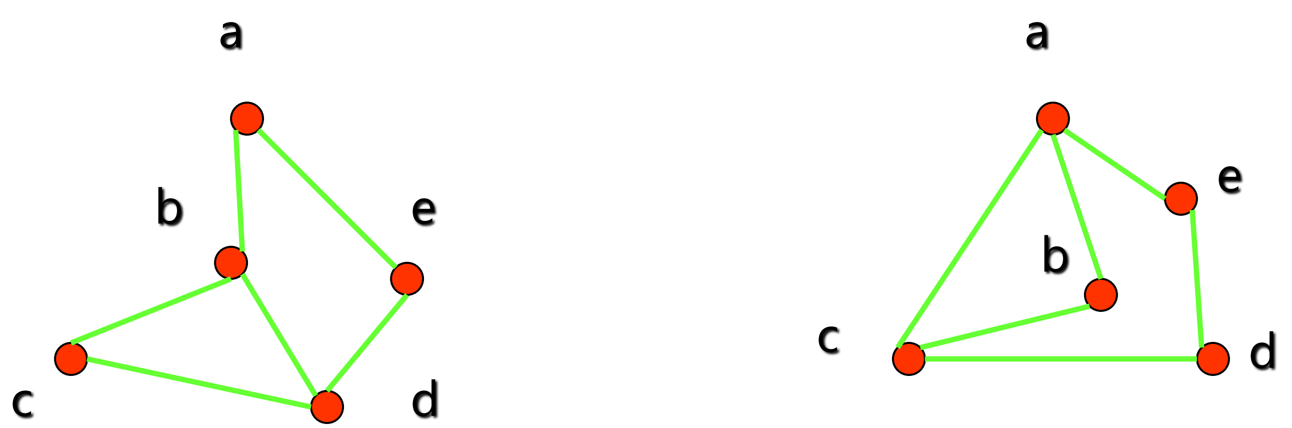
Definition:

- The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection (a one-to-one and onto function) f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2, for all a and b in V_1.
- Such a function f is called an isomorphism.
- In other words, G_1 and G_2 are isomorphic if their vertices can be ordered in such a way that the adjacency matrices $M(G_1)$ and $M(G_2)$ are identical.

- For this purpose, we can check invariants, that is, properties that two isomorphic simple graphs must both have.
- For example, they must have
  - The same number of nodes,
  - the same number of edges,
  - And the same degrees of nodes.
- Note that two graphs that differ in any of these invariants are not isomorphic, but two graphs that match in all of them are not necessarily isomorphic.

Isomorphism of Graphs: Example 1

- Are the following two graphs isomorphic?



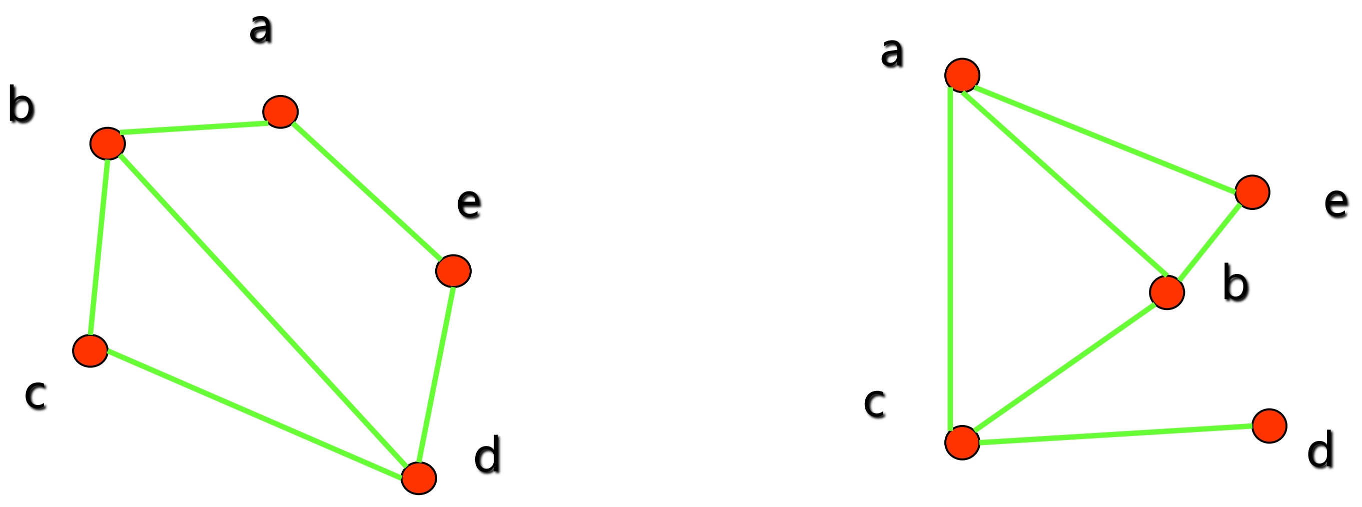
- **Solution**: Yes, they are isomorphic, because they can be arranged to look identical.

- You can see this if in the right graph you move vertex b to the left of the edge {a, c}. Then the isomorphism f from the left to the right graph is

f(a) = e, f(b) = a, f(c) = b, f(d) = c, f(e) = d.

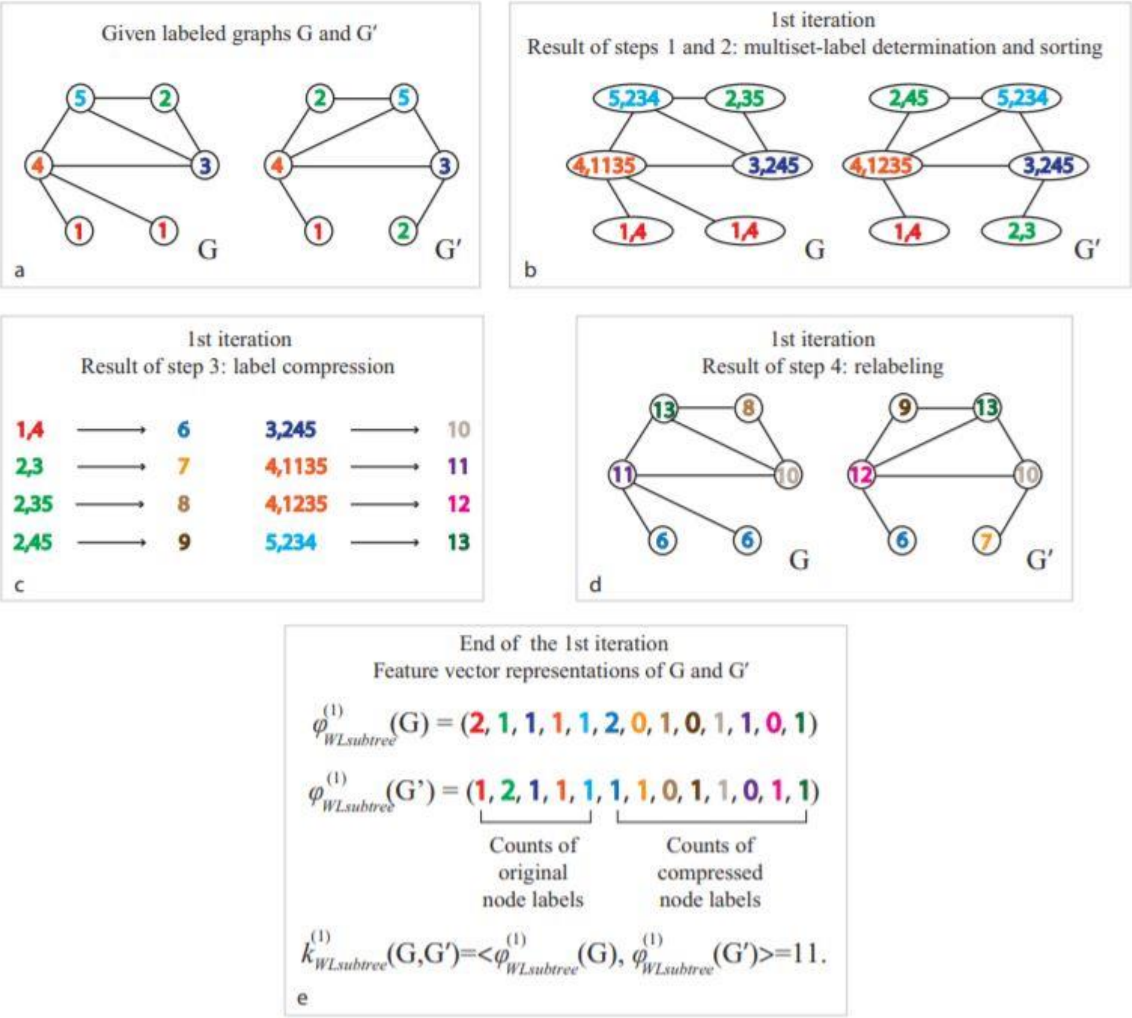
Isomorphism of Graphs: Example 2

- Are the following two graphs isomorphic?



- **Solution**: No, they are not isomorphic, because they differ in the degrees of their vertices.
- Vertex d in right graph is of degree one, but there is no such vertex in the left graph.

- Weisfeiler-Lehman Isomorphism Testing:



Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$

t $\leftarrow 0$;

repeat

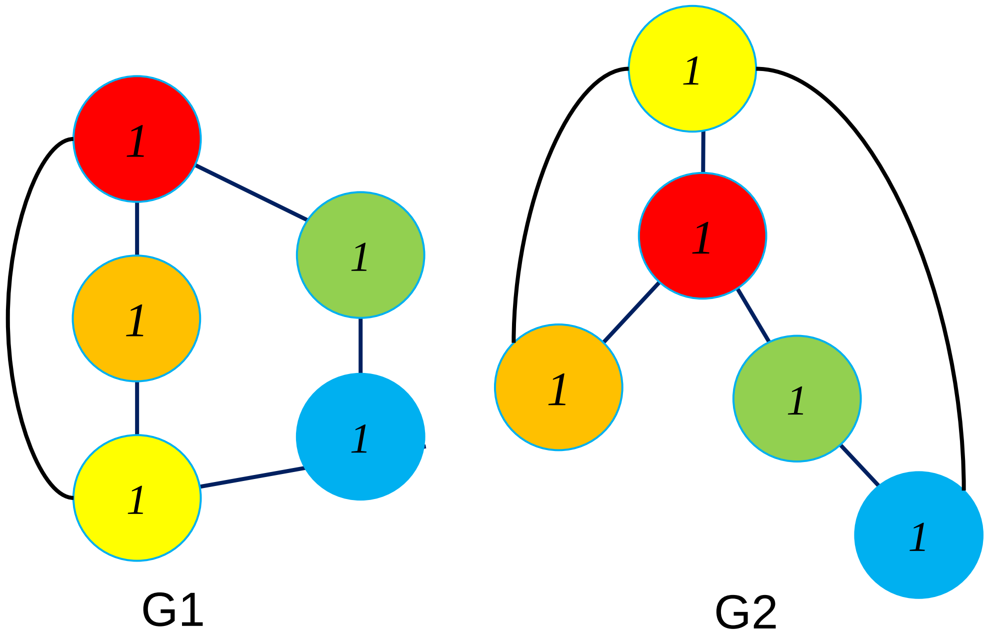
  for $v_i \in \mathcal{V}$ **do**

    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;

  $t \leftarrow t + 1$;

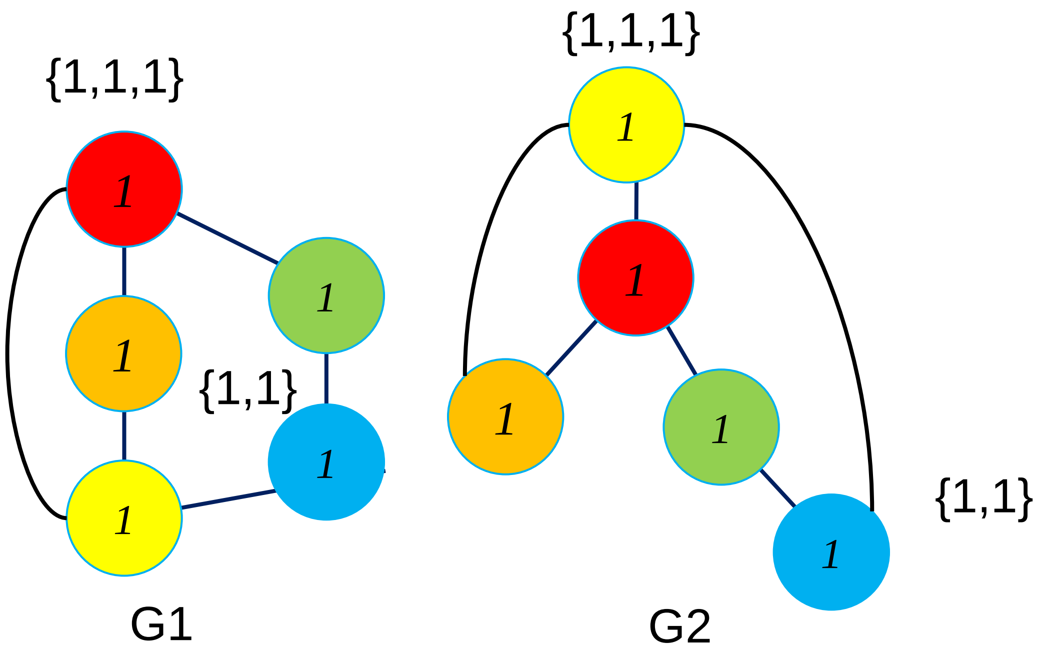
until *stable node coloring is reached*;

- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a m eans of illustrating the test.
- Step 1: Set node label =1 for all nodes

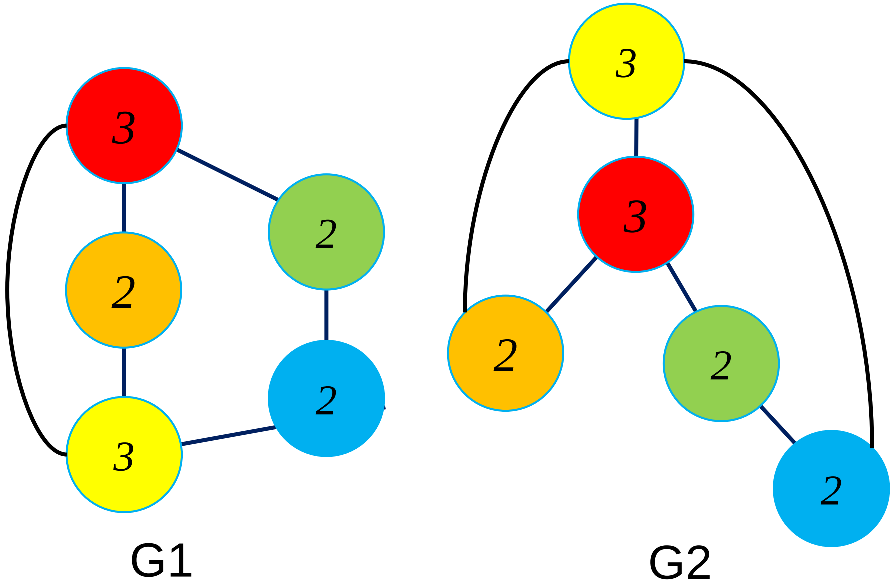


The Weisfeiler-Lehman Isomorphism Test

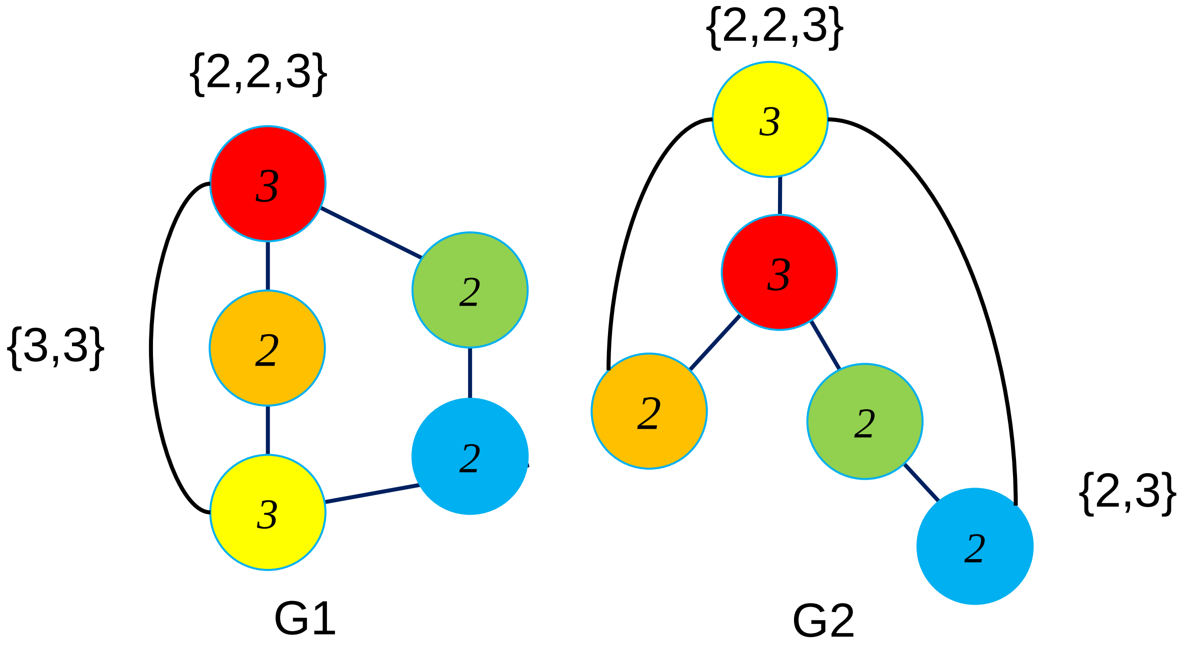
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test.
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels.



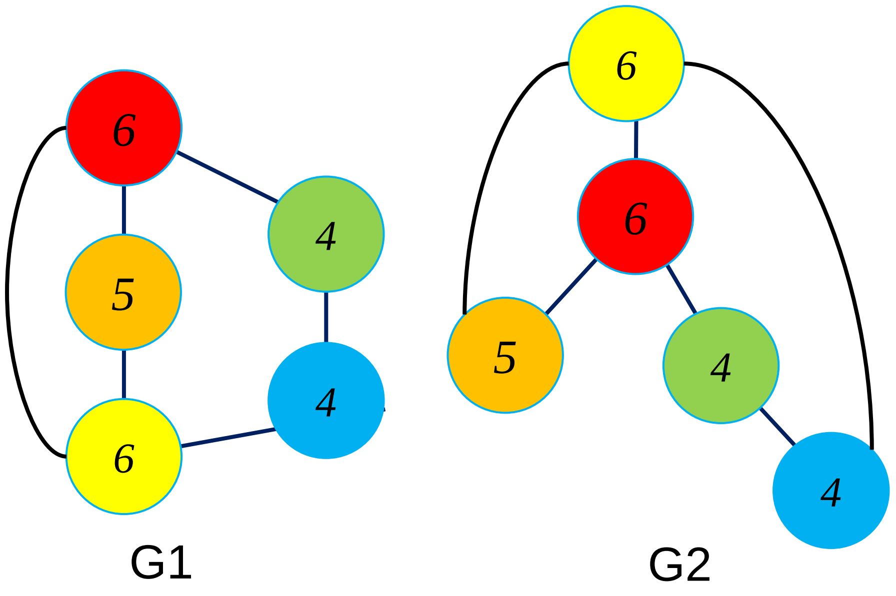
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test.
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels.



- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test.
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels.
- Step 3: Continuous.

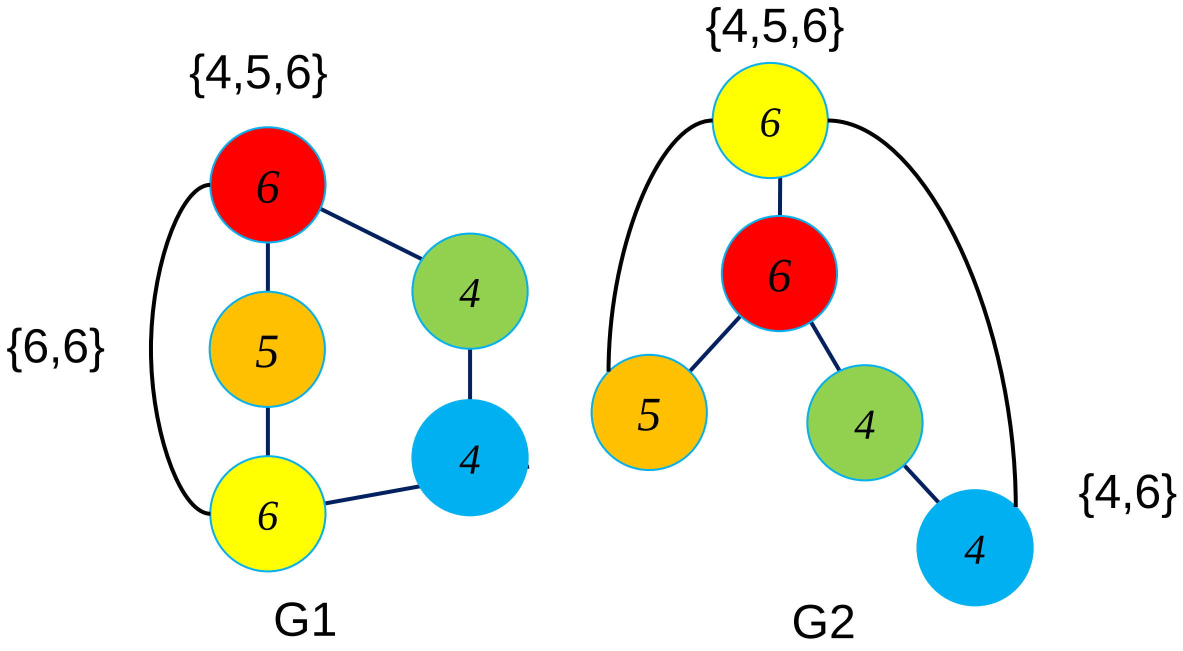


- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a me ans of illustrating the test.
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels.
- Step 3: Continuous….

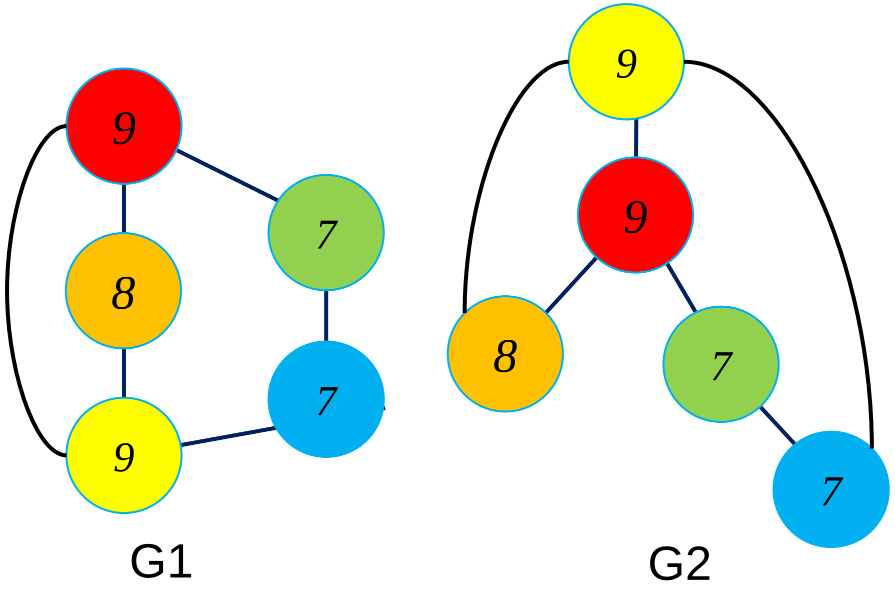


The Weisfeiler-Lehman Isomorphism Test

- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a mea ns of illustrating the test.
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels.
- Step 3: Continuous…



- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a mea ns of illustrating the test.
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels.
- Step 3: Continuous….
- Step 4: Since the partition of nodes by compressed label has not changed, we m ay terminate the algorithm here



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA