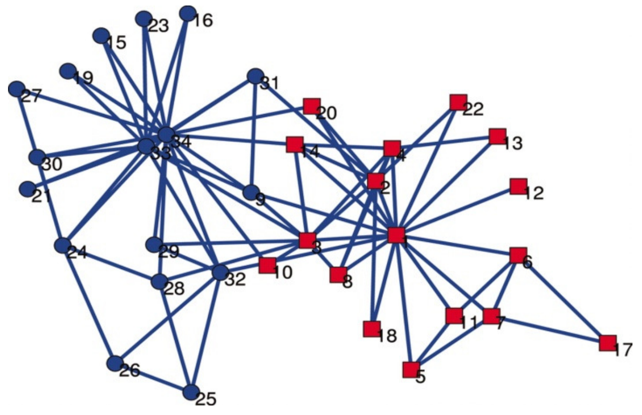
Community Detection Algorithms

Dr. Natarajan Meghanathan
Professor of Computer Science
Jackson State University, Jackson, MS
E-mail: natarajan.meghanathan@jsums.edu

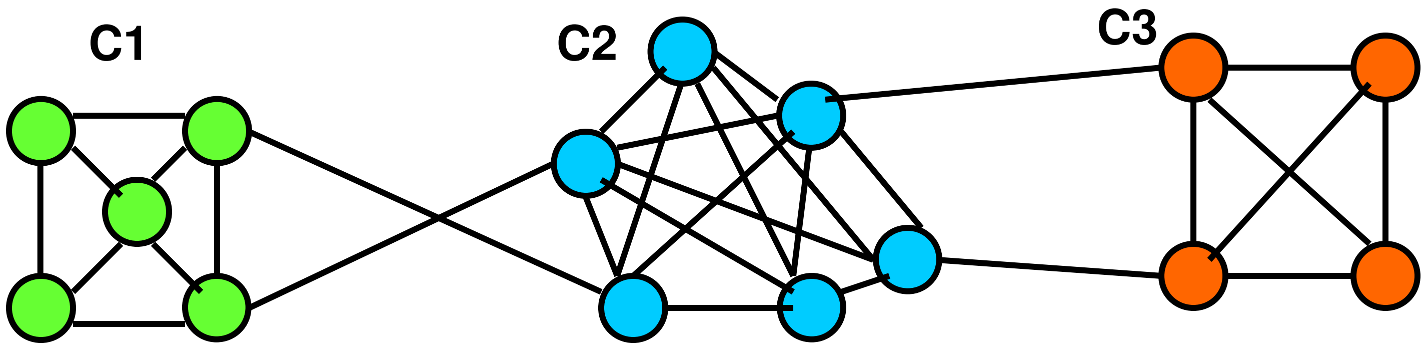
Community

- Community: It is formed by individuals such that those within a group <u>interact</u> with each other more frequently than with those outside the group.
- Community detection: discovering groups in a network where individuals' <u>group memberships</u> are not explicitly given.
  - <u>Interactions</u> (edges) between nodes can help determine communities
- Community structures are quite common in real networks. Social networks include community groups based on common location, interests, occupation, etc.
- Metabolic networks have communities based on functional groupings.
- Citation networks form communities by research topic.
- Identifying the community sub structures within a network can provide insight into how network function and topology affect each other.



There is most likely a path from one vertex to another vertex within a community through the vertices that are also part of the same community.
For the Karate Club network (to the left), the internal densities of the two communities are 0.26 and 0.24; the external densities are 0.035; the overall network density is 0.14.

Internal and External Community Densities



- Let C be a subset of nodes (V) that form a community.
- For every node i in C, let k_i^{int} and k_i^{ext} be the # links connecting node i to a node in C and outside C respectively.

$$\delta_{\text{int}}(C) = \frac{\sum_i k_i^{\text{int}}}{n_C(n_C - 1)}$$

$$\delta_{ext}(C) = \frac{\sum_i k_i^{ext}}{2n_C(n_C - 1)}$$

The internal density of every cluster is significantly larger than the external density as well as the total density of the network.

Internal Densities

C1	(4*3 + 1*4) / (5*4) = 0.8
C2	(6*5)/(6*5) = 1.0
C3	(4*3)/(4*3) = 1.0

External Densities

C1	(1 + 1) / (2*5*4) = 0.05
C2	(1 + 1 + 1 + 1)/(2*6*5) = 0.067
C3	(1 + 1)/(2*4*3) = 0.083

Schemes for Identifying Communities

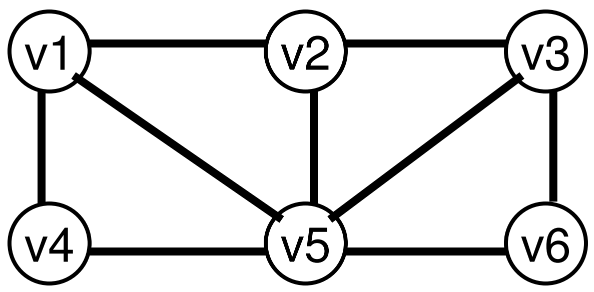
- The number of communities within a network is typically unknown and the communities are often of unequal size and/or density.
- Schemes:
  - Clique-based
  - Hierarchical Clustering
    - Bottom-up and Top-down
  - Neighborhood Overlap based
  - Homophily
  - Eigen Vector based
- Evaluation:
  - Modularity Maximization
  - Silhouette Index

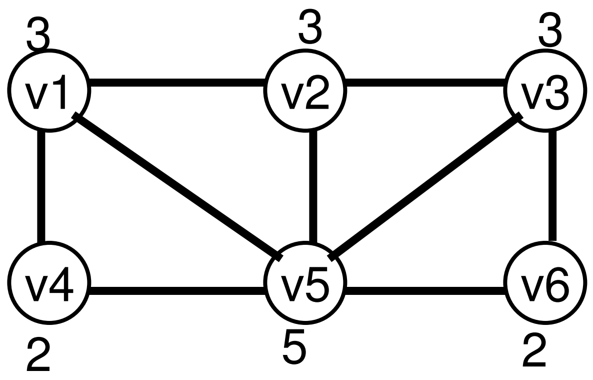
Clique-based Schemes

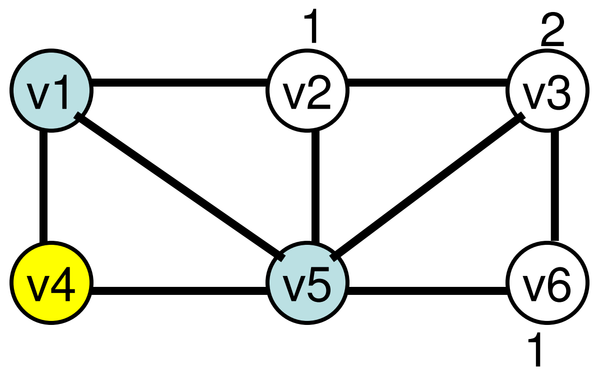
Clique (Complete Mutuality)

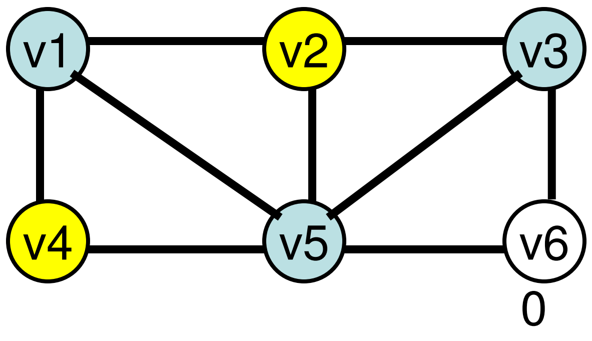
- A clique in a graph is a sub graph in which all the constituent nodes are directly reachable from one another.
- It is a NP-hard problem to find the maximum-sized clique in a graph.
  - Independent Set-based Minimum Neighbors Heuristic
    - Could find more than one clique of different sizes
  - Repeated Vertex and Edge Removal Heuristic
    - To find a clique of maximum size (depends on an underlying heuristic)
  - Clique Percolation methods (could find overlapping communities)
- We will use the notion of Independent Sets to find a clique in a graph
  - Independent Set: A subset of vertices such that there is no edge in the graph between any two vertices in the subset
  - For a given graph G, we will find a complement graph G*
    - The vertices in G and G* are the same.
    - If an edge (u, v) is in G, there is no edge (u, v) in G*
    - If an edge (u, v) is not in G, there is an edge (u, v) in G*
  - An independent set in the complement graph G* is a clique in the original graph G.

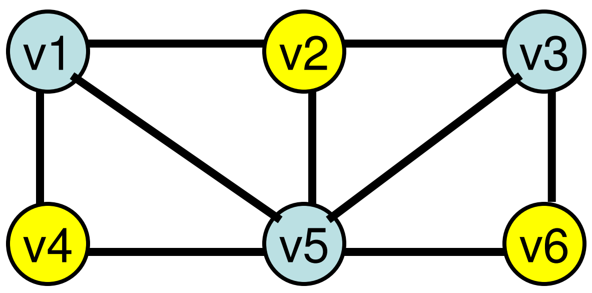
Example to Find Independent Set and Clique: Minimum Neighbors Heuristic









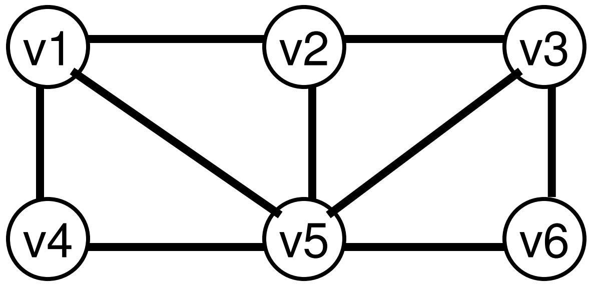


Idea: Give preference to vertices with minimal number of (uncovered) neighbors to be part of the Independent Set. A vertex is said to be covered if itself or any of its neighbors in the Independent Set.

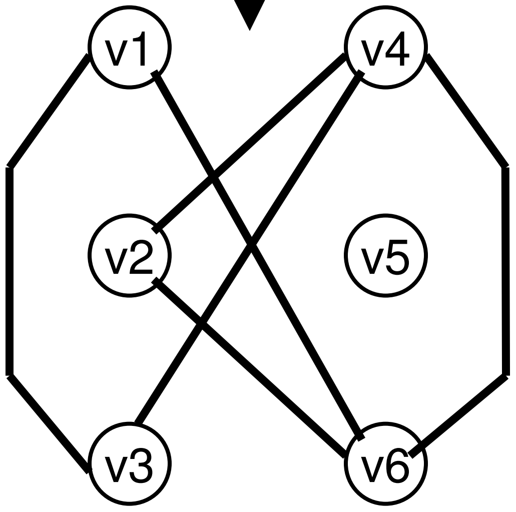
Independent Set for the above graph = {v2, v4, v6}

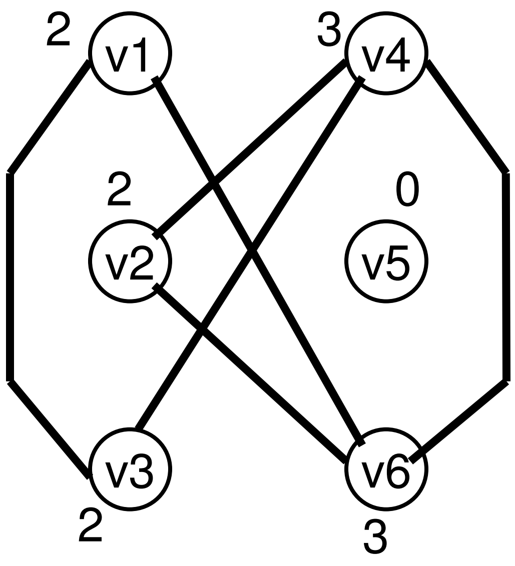
This is also the Maximal Independent Set (i.e., there exists no Independent Set of size 4 or more for the above graph). However, the heuristic is not guaranteed in general to give a maximal Independent set.

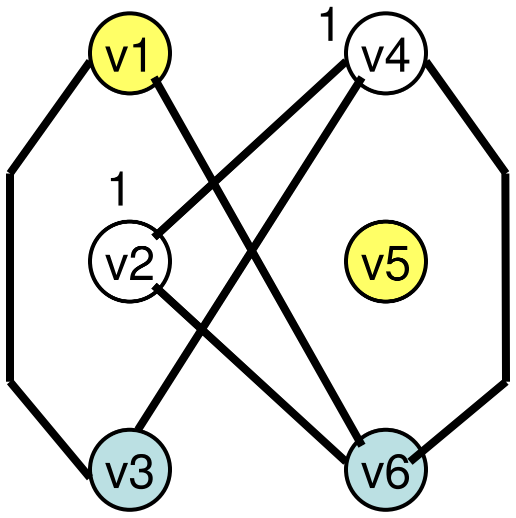
Given G ------->



Find G*, complement of G

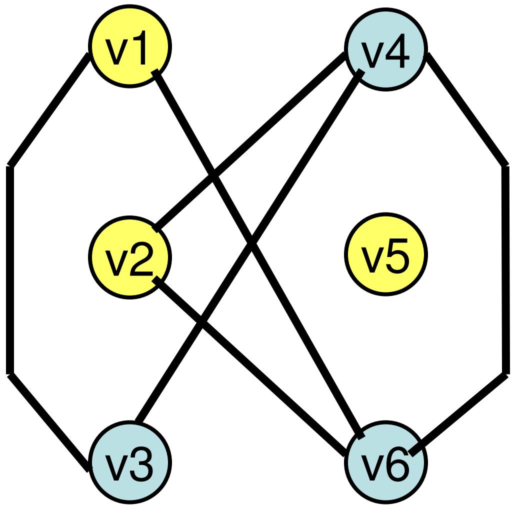




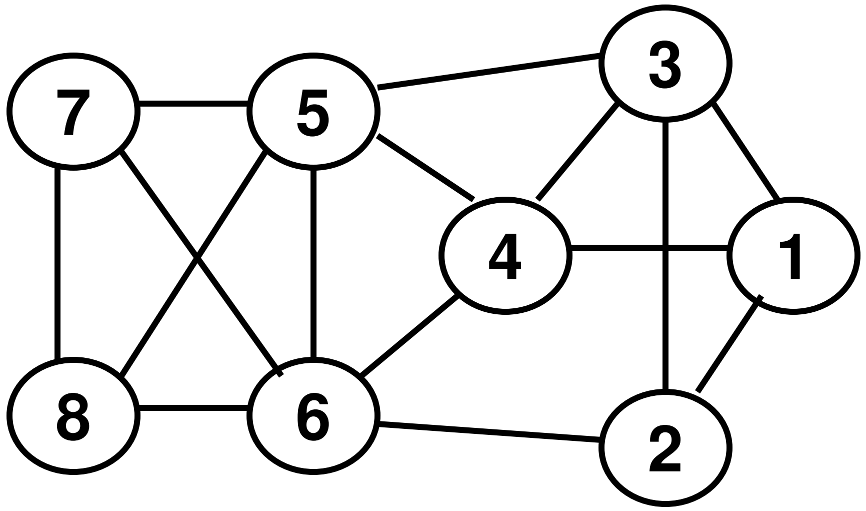


Example 1 to Determine a Clique Using the Minimum Neighbors Heuristic to Approximate an Independent Set

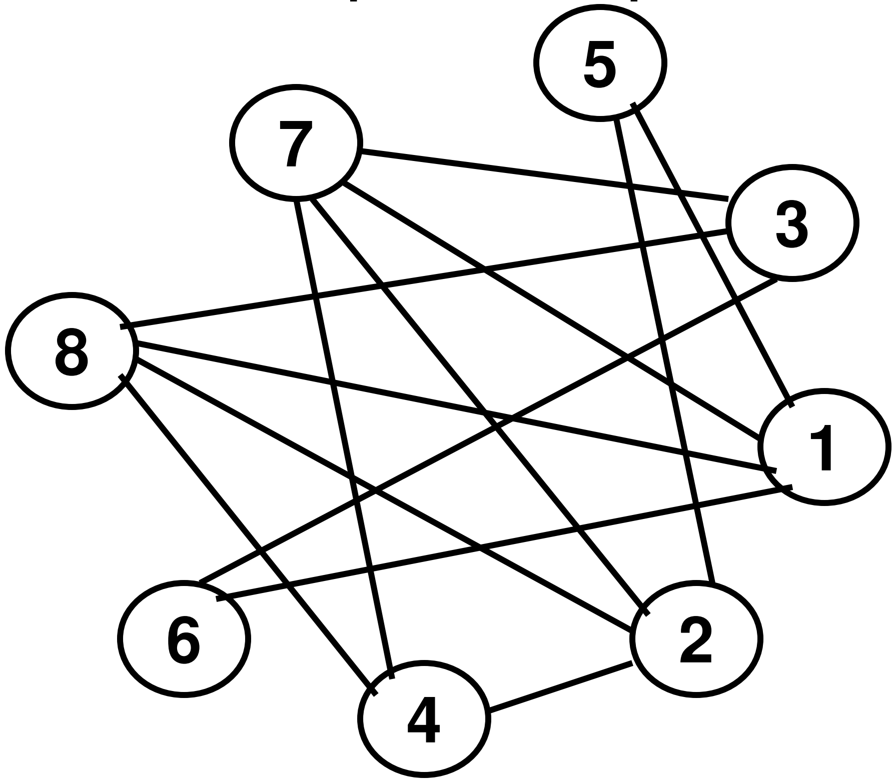
{v1, v2, v5} is an Independent Set in G* and it is a clique in G.



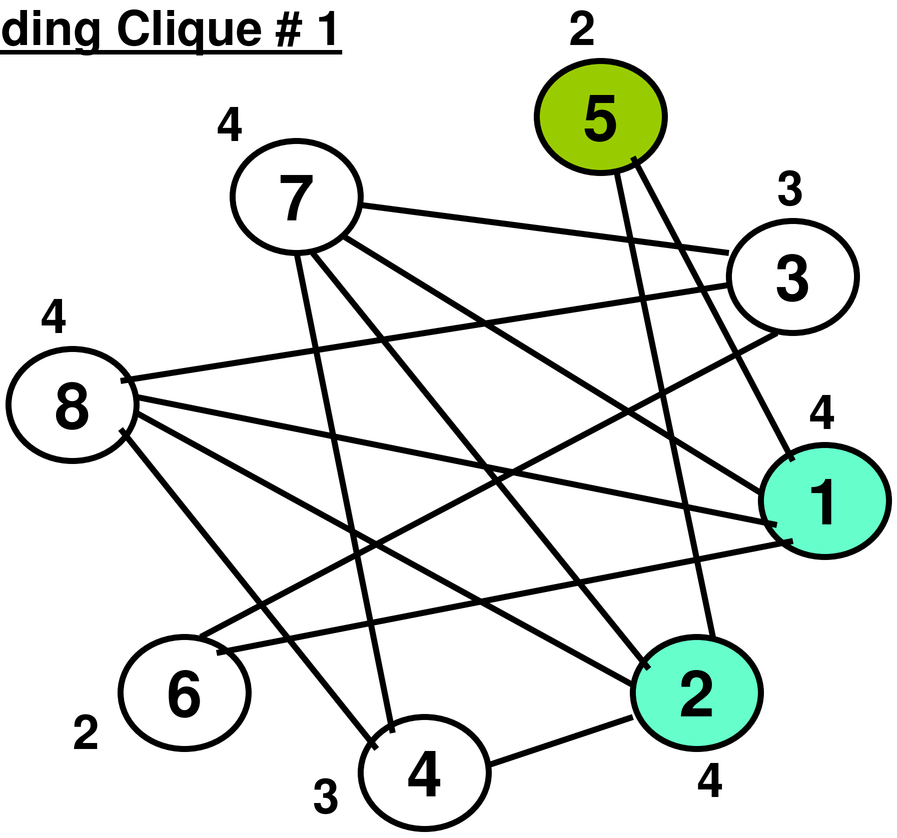
Example to Find Several Cliques In a Graph

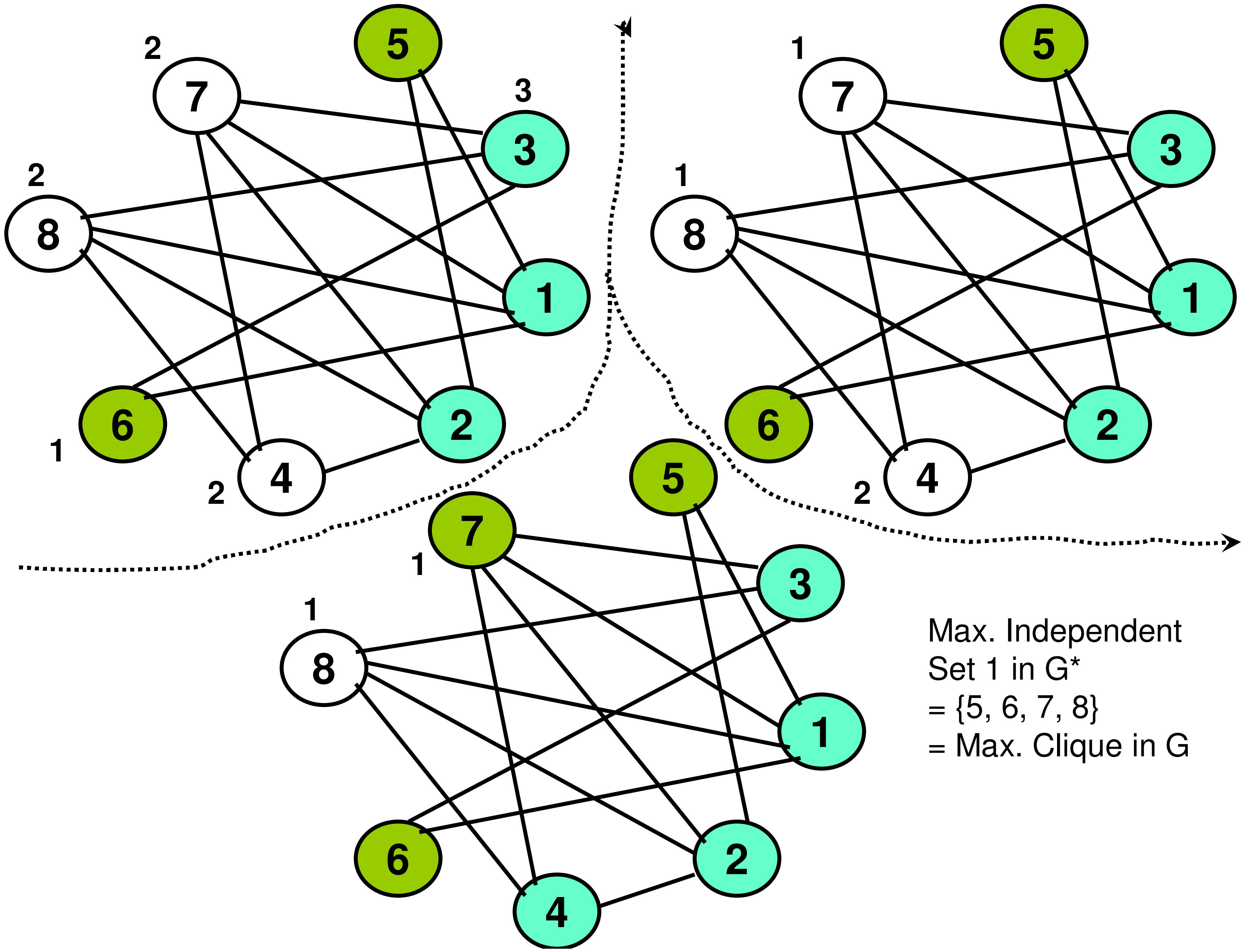


Find the Complement Graph G*



Finding Clique # 1





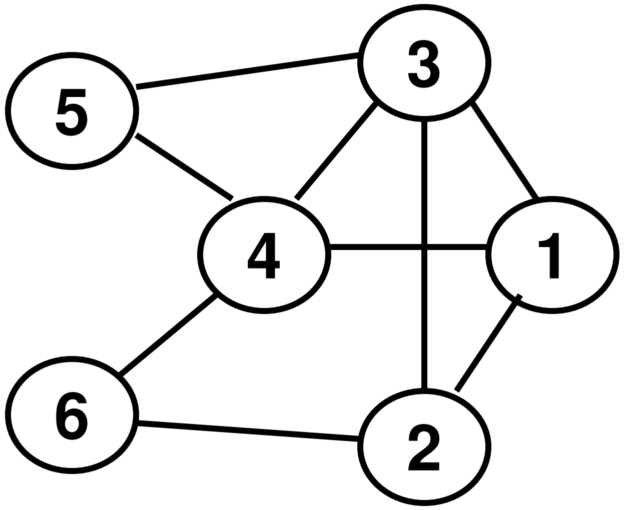
Max. Independent
Set 1 in G*
= {5, 6, 7, 8}
= Max. Clique in G

Remove the edges associated with 5, 6, 7, 8
from the original graph G and generate a new graph G'
Find a new complement Graph G** (for G')

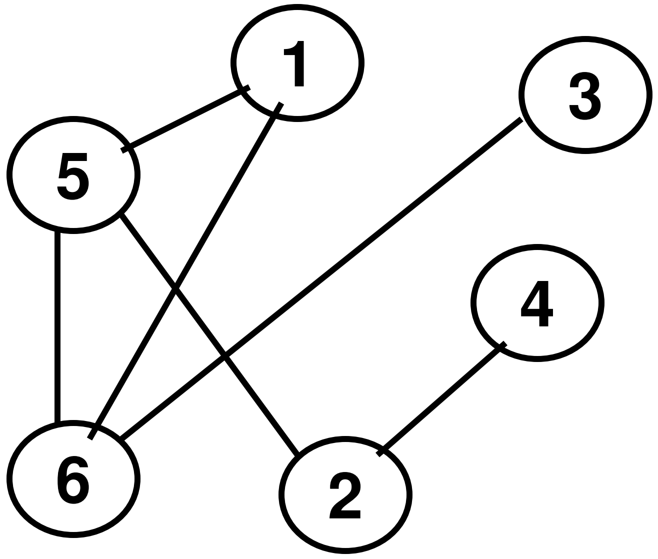
[5, 6, 7, 8] is Clique # 1

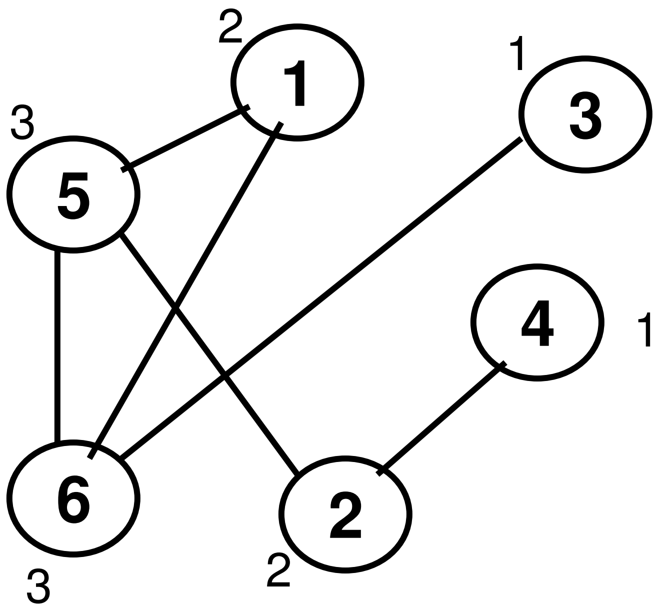
Remove stub nodes and isolated nodes
(nodes with only one edge or no edge)

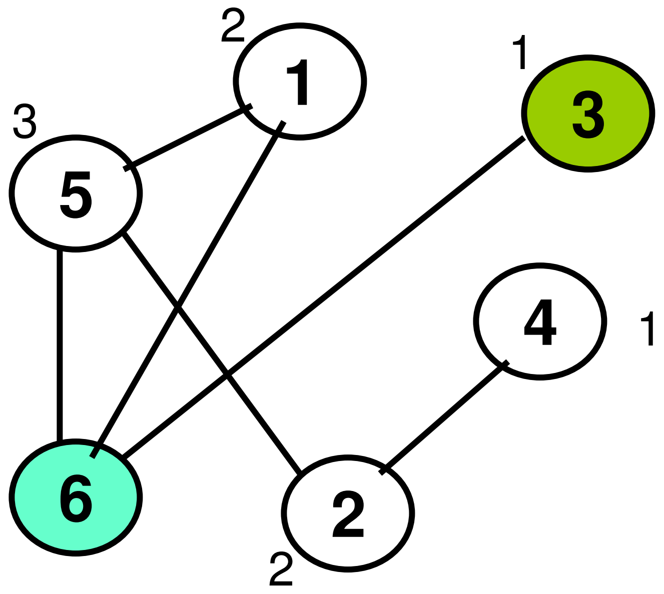
Reduced graph G'

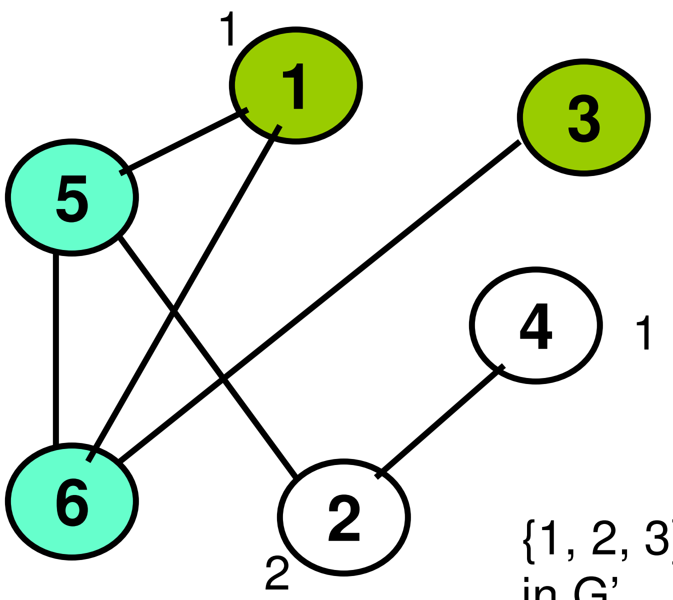


New Complement Graph G**

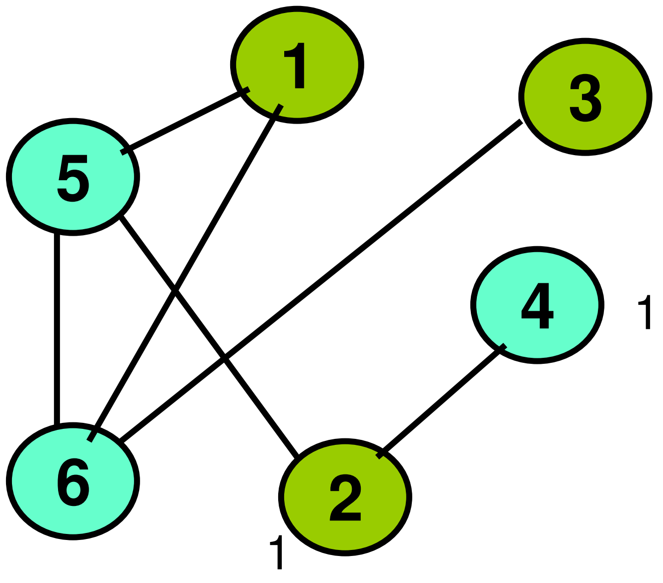


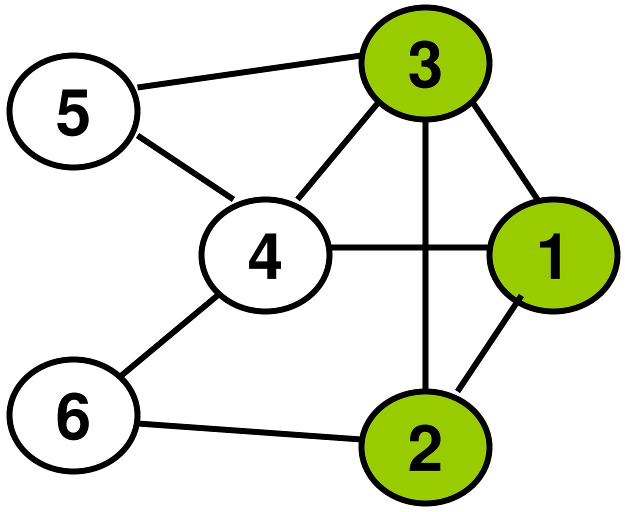






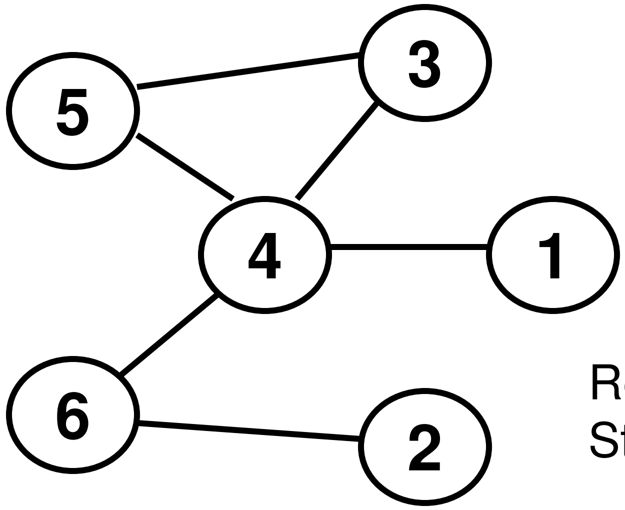
{1, 2, 3} is a clique in G’

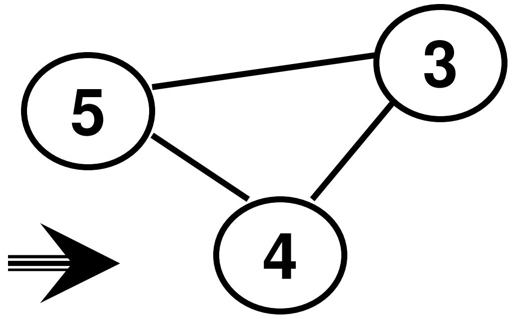




[1, 2, 3] is Clique # 2

Removing edges associated with 1, 2, 3

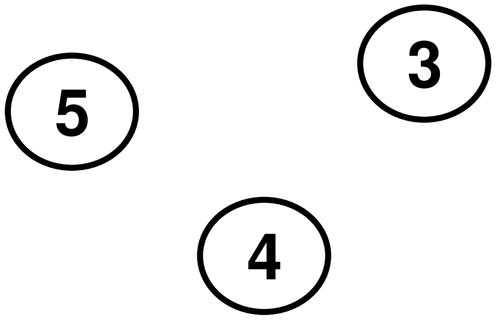




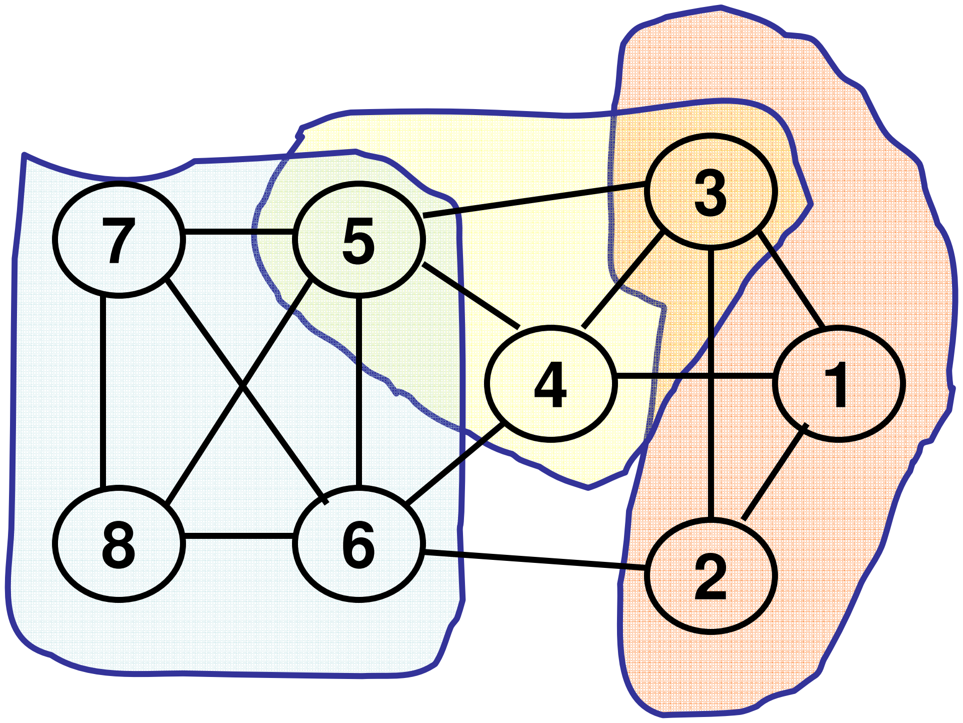
Graph G''

Removing all the Stub nodes (1, 2 and 6)

[3, 4, 5] is a clique in G''



Complement Graph of G''

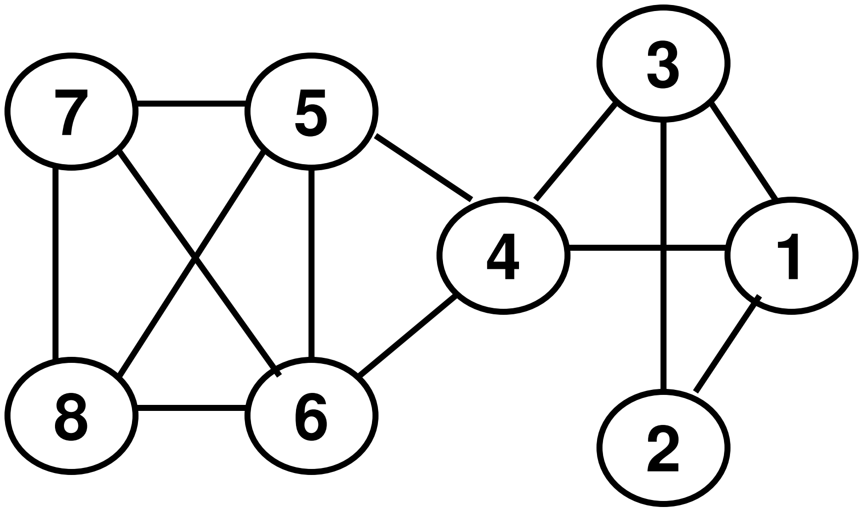


The three cliques

[5, 6, 7, 8]
[1, 2, 3]
[3, 4, 5]

Pruning Technique: Maximum Clique

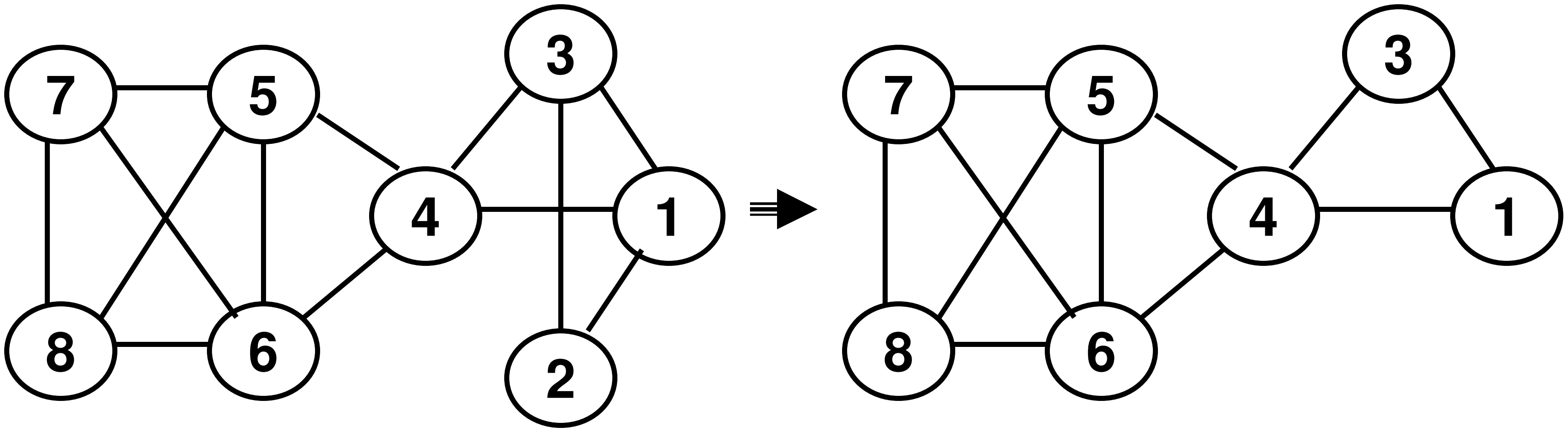
- Lets say, we are interested to find a clique of size k.
  - All vertices in such clique must have degree at least k-1.
    - Repeat the following until we only have vertices with degree k-1 or above in the graph
      - Step 1: Remove vertices that have degree less than k-1
      - Step 2: Because of the removal of the vertices in Step 1, the degree of some other vertices would have become less than k-1.
        - » If any such vertices exist, Go to Step 1.
        - » Otherwise, exit from the loop
    - If the reduced graph obtained from the above has one or more components in which each component has vertices with degree k-1 or above, run any heuristic to find clique (say, the Independent Set-based heuristic) on the reduced graph

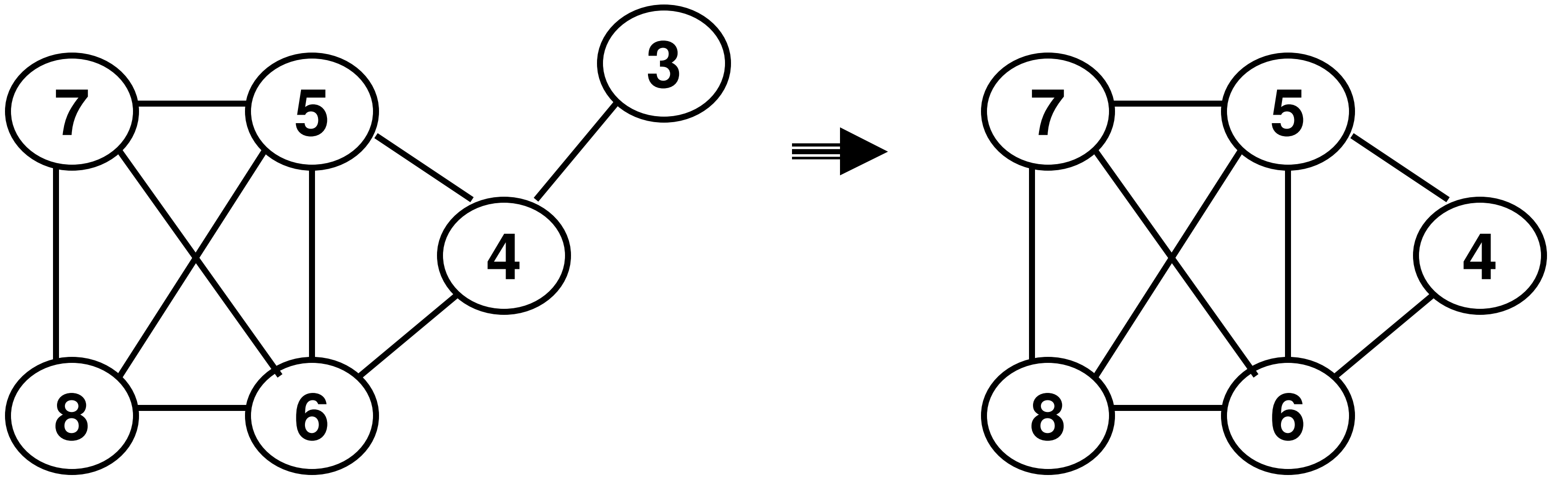


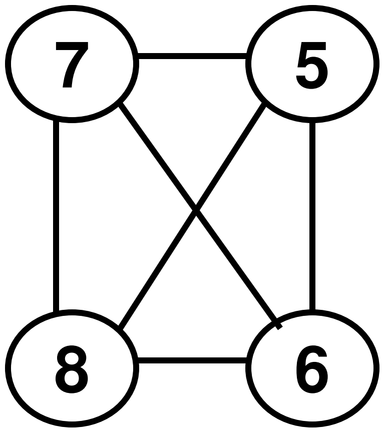
Example to Find The Maximum Clique Using Pruning Technique

<u>Finding the Maximum Clique</u>

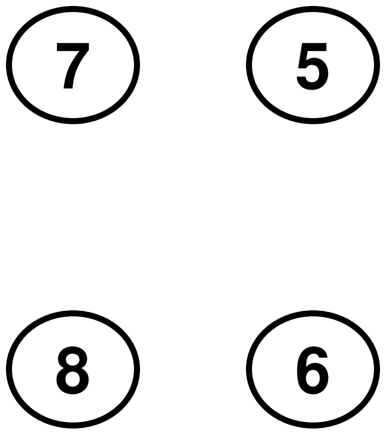
Anticipating a Clique of Size 4 (i.e., each vertex in the clique has degree 3)
Remove from the graph all vertices with degree less than 3.
Recursively remove all the vertices and associated edges until each vertex has degree 3 or above.







Apply the Independent Set Heuristic



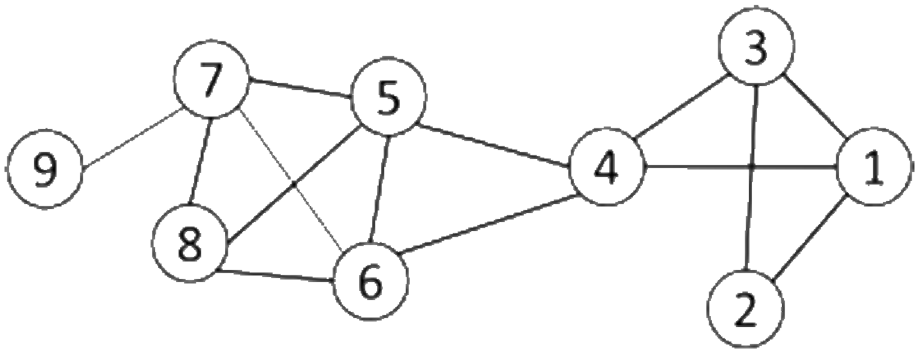
[5, 6, 7, 8] form an independent set in the complement graph

Hence [5, 6, 7, 8] form a clique in the original graph

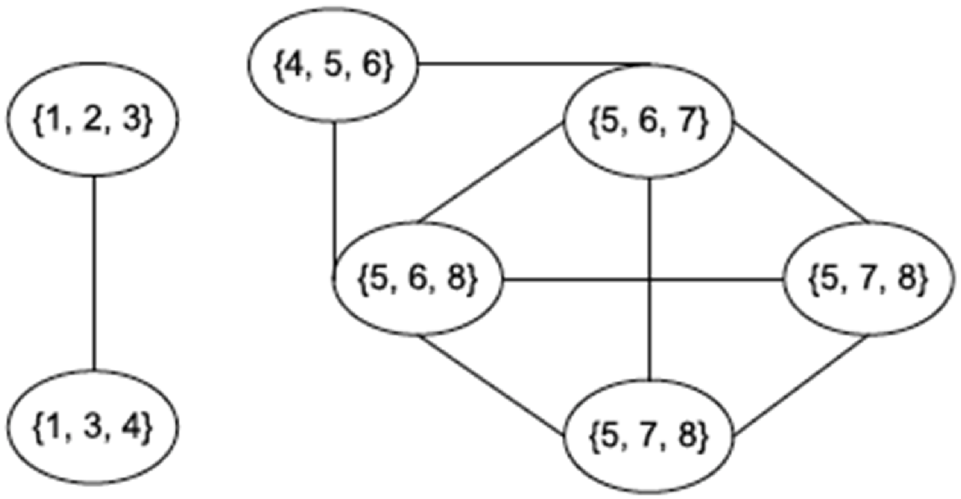
Clique Percolation Method

- Used to find overlapping communities
  - **Input**
    - A parameter k, and a network
  - **Procedure**
    - Find out all cliques of size k in a given network
    - Construct a clique graph. Two cliques are adjacent if they share k-1 nodes
    - Each connected component in the clique graph forms a community

Example 1: Clique Percolation Method

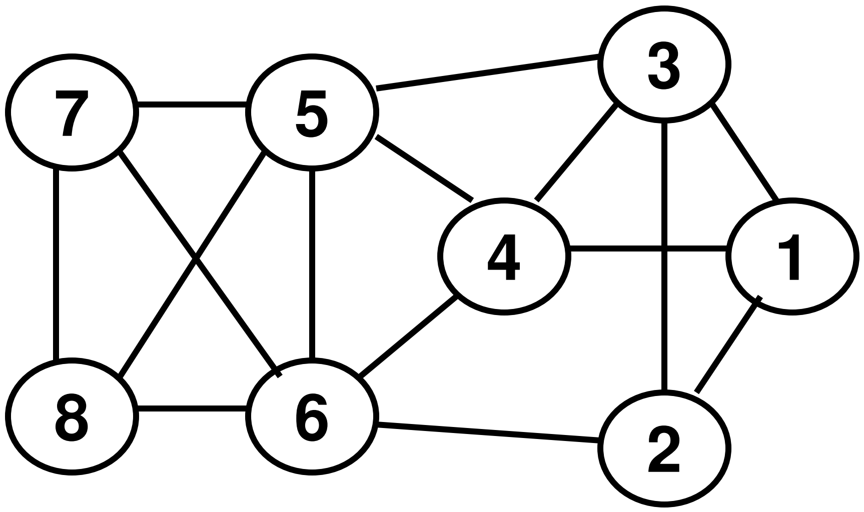


Cliques of size 3:
{1, 2, 3}, {1, 3, 4}, {4, 5, 6}, {5, 6, 7}, {5, 6, 8}, {5, 7, 8}, {6, 7, 8}



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

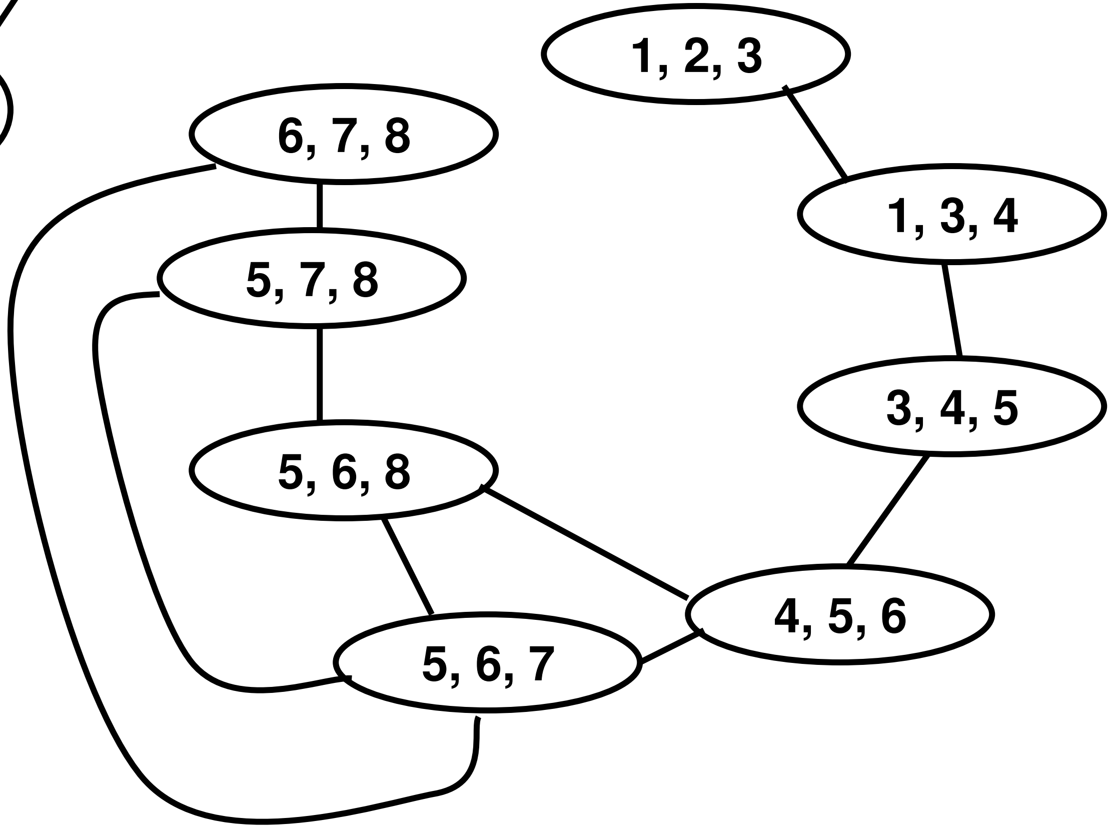
Example 2: Clique Percolation Method



The following is the clique graph. All the cliques of size 3 are connected. Hence, all the vertices in the given graph are said to be in one single community.

Cliques of Size 3

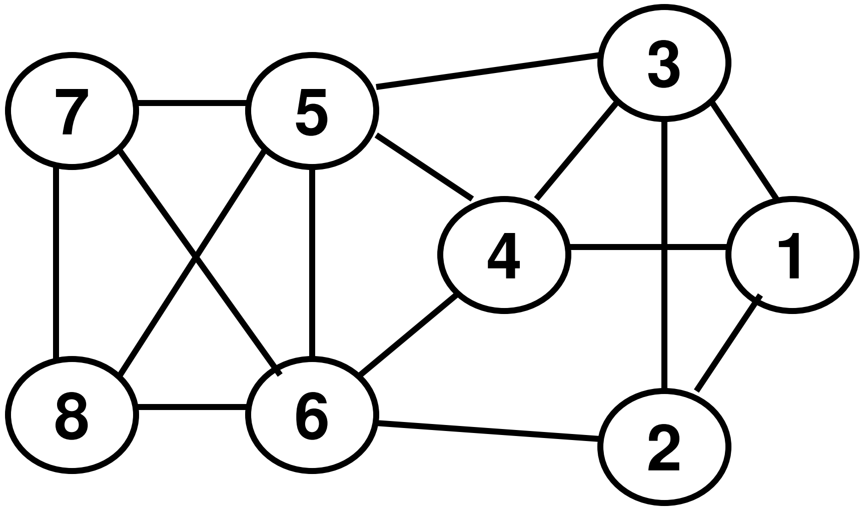
[1, 2, 3]
[1, 3, 4]
[3, 4, 5]
[4, 5, 6]
[5, 6, 7]
[5, 6, 8]
[5, 7, 8]
[6, 7, 8]



Modularity Maximization

Modularity Maximization

- Modularity measures the strength of a community partition by taking into account the degree distribution.
- Given a network with *m* edges, the expected number of edges between two nodes *i* and *j* with degrees d_i and d_j respectively is $d_i^* d_j$ / 2m.



Expected number of edges between nodes 1 and 2 is (3)(2) / (2*15) = 0.20

Strength of a Community, C

$$\boxed{\sum_{i\in C,\, j\in C} A_{i,j} - \frac{d_i d_j}{2m}}$$

For a network with *k* communities and a total of *m* edges

Modularity:

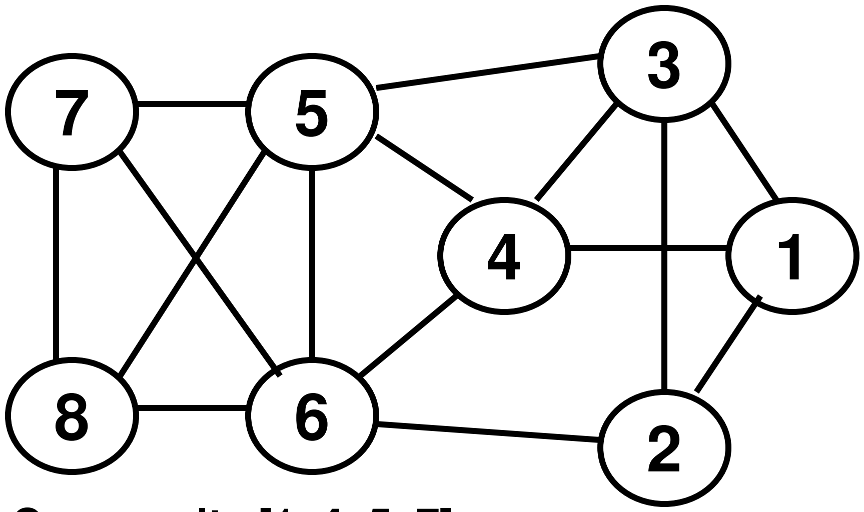
$$\boxed{Q = \sum_{l=1}^{k} \sum_{i\in C_l,\, j\in C_l} A_{i,j} - \frac{d_i d_j}{2m}}$$

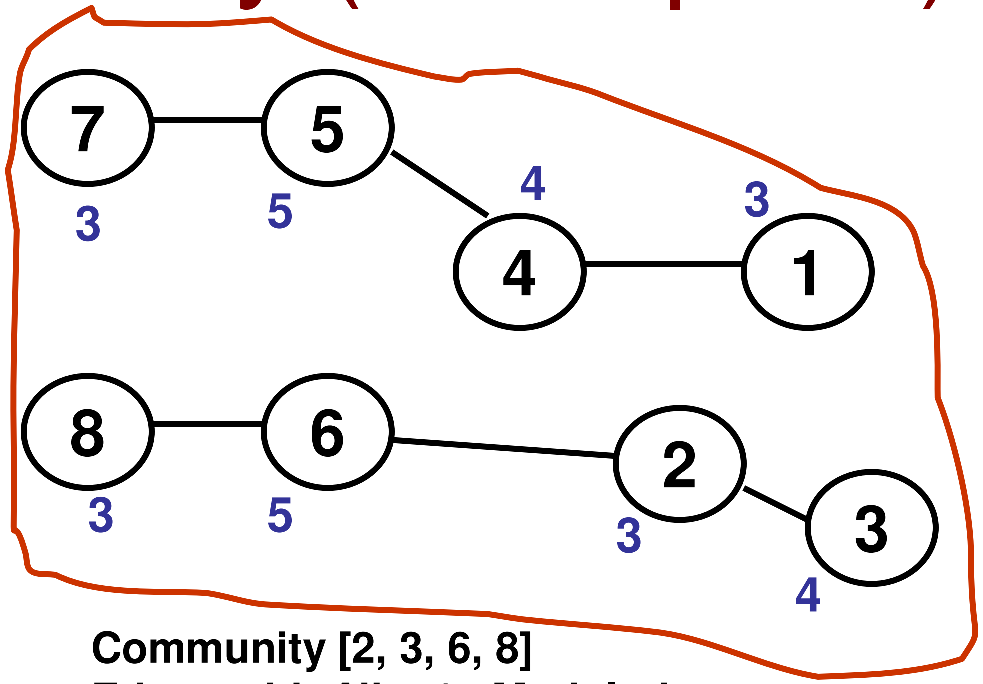
A larger value for Q indicates a good community structure

Modularity Maximization

- The intuition behind the idea of modularity is that a community is a structural element of a network that has been formed in a manner far from a random process.
- If we consider the actual density of links in a community, it should be significantly larger than the density we would expect if the links in the network were formed by a random process.
  - In other words, if two nodes i and j are end up being in the same community, there should be more likely a link between them (i.e., Aij = 1, leading to an overall high value for Q).
  - If i and j end up being in a community such that the chances of having a link between them is just as the same as between any two nodes in the network (i.e., a random network), then the value of Q is more likely to be low (because there could be some Aij = 0 that will bring down the value of Q).

Evaluating Modularity (Example 1)





Community [1, 4, 5, 7]

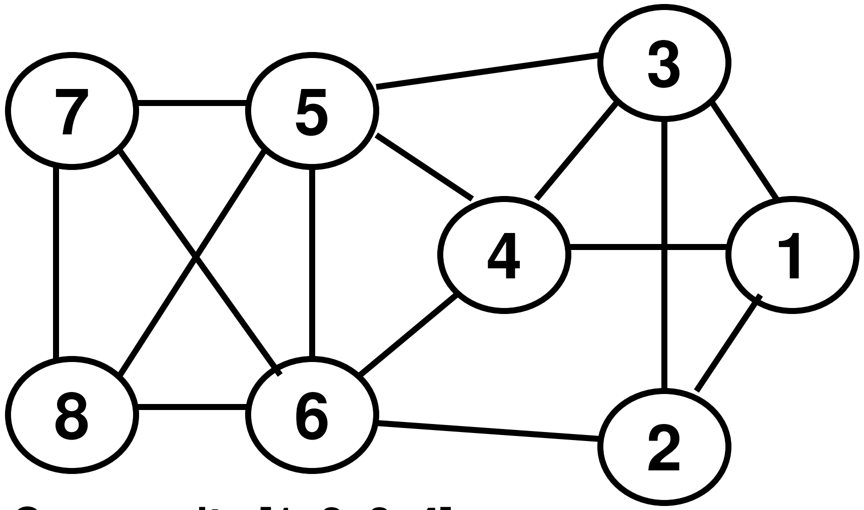
Edges with Aij = 1	**Modularity**
1 – 4	1 – (3)(4)/(2*15) = 0.60
4 – 5	1 – (4)(5)/(2*15) = 0.33
5 – 7	1 – (3)(5)/(2*15) = 0.50
Edges with Aij = 0	
1 – 5	0 – (3)(5)/(2*15) = -0.50
1 – 7	0 – (3)(3)/(2*15) = -0.30
4 – 7	0 – (4)(3)/(2*15) = -0.40
Total Modularity Score for Community [1, 4, 5, 7]	**0.23**

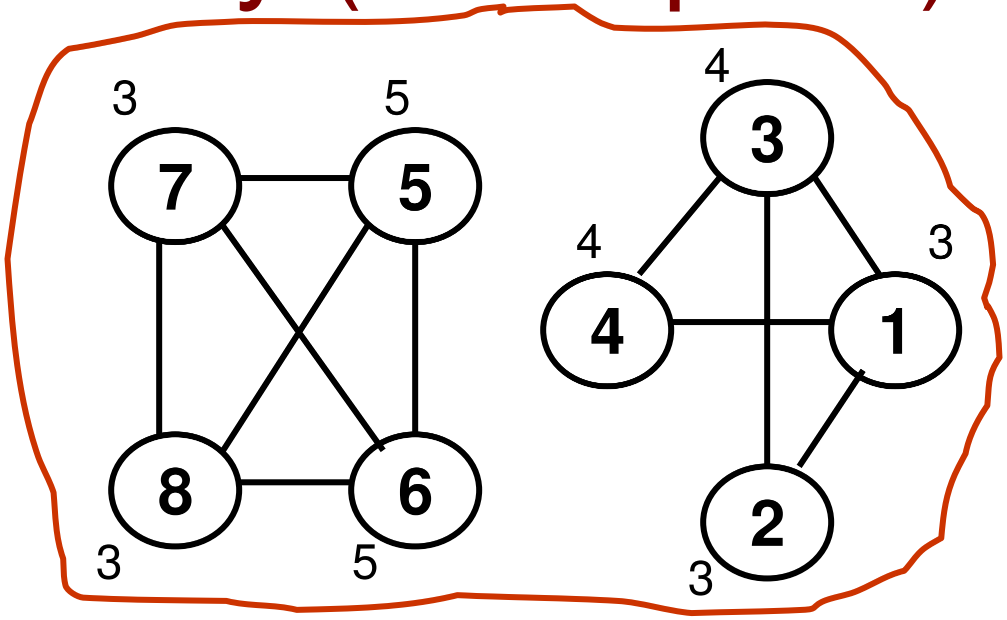
Community [2, 3, 6, 8]

Edges with Aij = 1	**Modularity**
2 – 3	1 – (3)(4)/(2*15) = 0.60
2 – 6	1 – (3)(5)/(2*15) = 0.50
6 – 8	1 – (3)(5)/(2*15) = 0.50
Edges with Aij = 0	
2 – 8	0 – (3)(3)/(2*15) = -0.30
3 – 6	0 – (4)(5)/(2*15) = -0.67
3 – 8	0 – (4)(3)/(2*15) = -0.40
Total Modularity Score for Community [2, 3, 6, 8]	**0. 23**

<u>Total Modularity for the two Communities: 0.23 + 0.23 = 0.46</u>

Evaluating Modularity (Example 2)





Community [1, 2, 3, 4]

Edges with Aij = 1	**Modularity**
1 – 2	**1 – (3)(3)/(2*15) = 0.70**
1 – 3	**1 – (3)(4)/(2*15) = 0.60**
1 – 4	**1 – (3)(4)/(2*15) = 0.60**
2 – 3	**1 – (3)(3)/(2*15) = 0.70**
3 – 4	**1 – (4)(4)/(2*15) = 0.47**
Edges with Aij = 0	
2 – 4	**0 – (3)(4)/(2*15) = -0.40**
Total Modularity Score for Community [1, 2, 3, 4]	**2.67**

Community [5, 6, 7, 8]

Edges with Aij = 1	**Modularity**
5 – 6	**1 – (5)(5)/(2*15) = 0.17**
5 – 7	**1 – (3)(5)/(2*15) = 0.50**
5 – 8	**1 – (3)(5)/(2*15) = 0.50**
6 – 7	**1 – (3)(5)/(2*15) = 0.50**
6 – 8	**1 – (3)(5)/(2*15) = 0.50**
7 – 8	**1 – (3)(3)/(2*15) = 0.70**
Total Modularity Score for Community [2, 3, 6, 8]	**2.87**

Total Modularity for the two Communities: 2.67 + 2.87 = 5.54

Hierarchical Clustering (Complete Linkage Clustering)

Bottom-Up Approach

(Agglomerative)

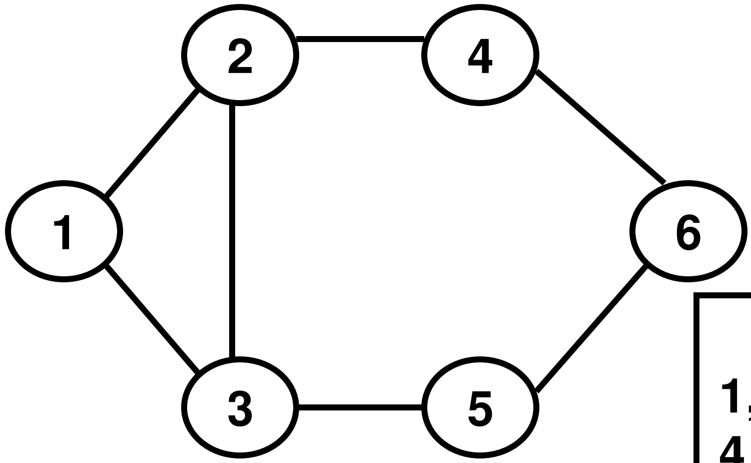
Complete Linkage Clustering

- Compute the “pair-wise” distance matrix P between any two vertices.
- Initially, start with each vertex in its own cluster.
- Merge the two “closest” vertices (clusters)
  - In case of a tie (between two cluster-cluster pairs or between two cluster-vertex pairs), choose the pair with the minimum value for the total pair-wise distance / sum of the two pair sizes
  - In case of a tie between a cluster-vertex pair and a vertex-vertex pair, choose the cluster-vertex pair
  - In case of a tie between two vertex-vertex pairs, break the tie arbitrarily.
- Remove the entries from P, for the two vertices (clusters) merged, and add an entry corresponding to the merged vertex (cluster).
  - Update this entry with the longest distance between any vertex in the merged cluster with the vertices in the other clusters in P.

$$\boxed{\text{Distance}(\text{C}_\text{i}, \text{C}_\text{j}) = Max\left[\underset{u\in C_i}{\forall}\ \underset{v\in C_j}{\forall}\ MinHops(u,v)\right]}$$

- Repeat the above step of merging and removing/adding entries to P until there is only one cluster.

Complete Linkage Clustering (Example 1)

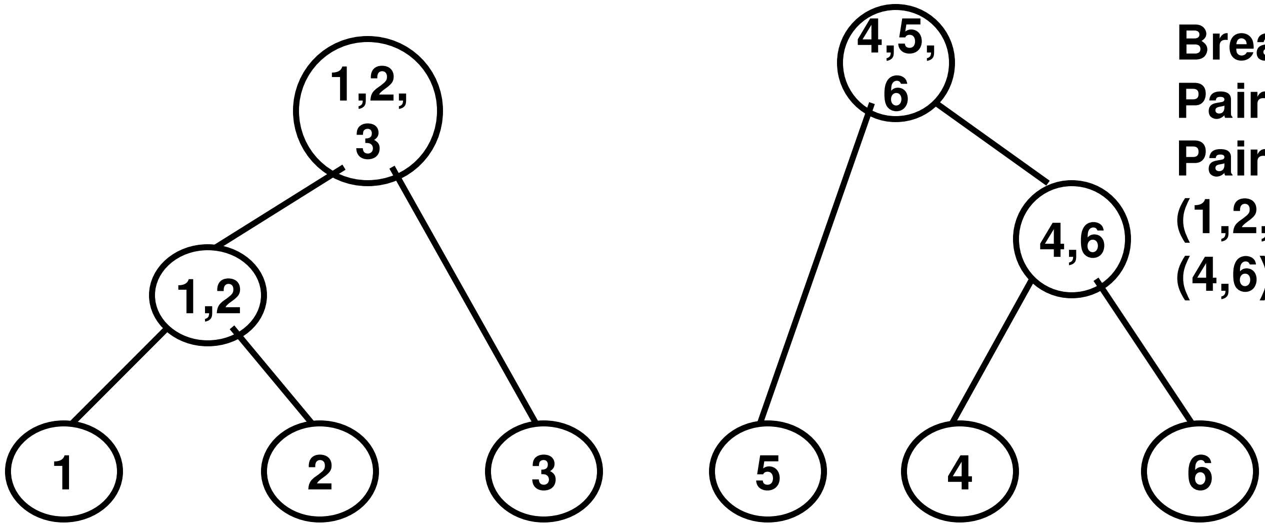


	1	2	3	4	5	6
1	0	1	1	2	2	3
2		0	1	1	2	2
3			0	2	1	2
4				0	2	1
5					0	1
6						0

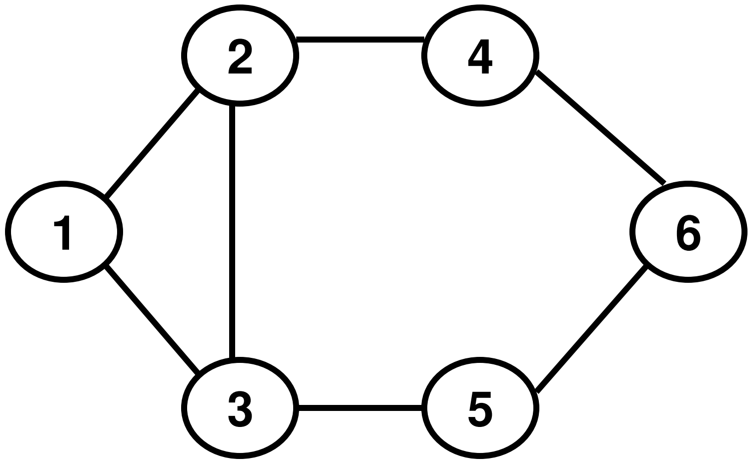
	1,2	3	4	5	6
1,2	0	1	2	2	3
3		0	2	1	2
4			0	2	1
5				0	1
6					0

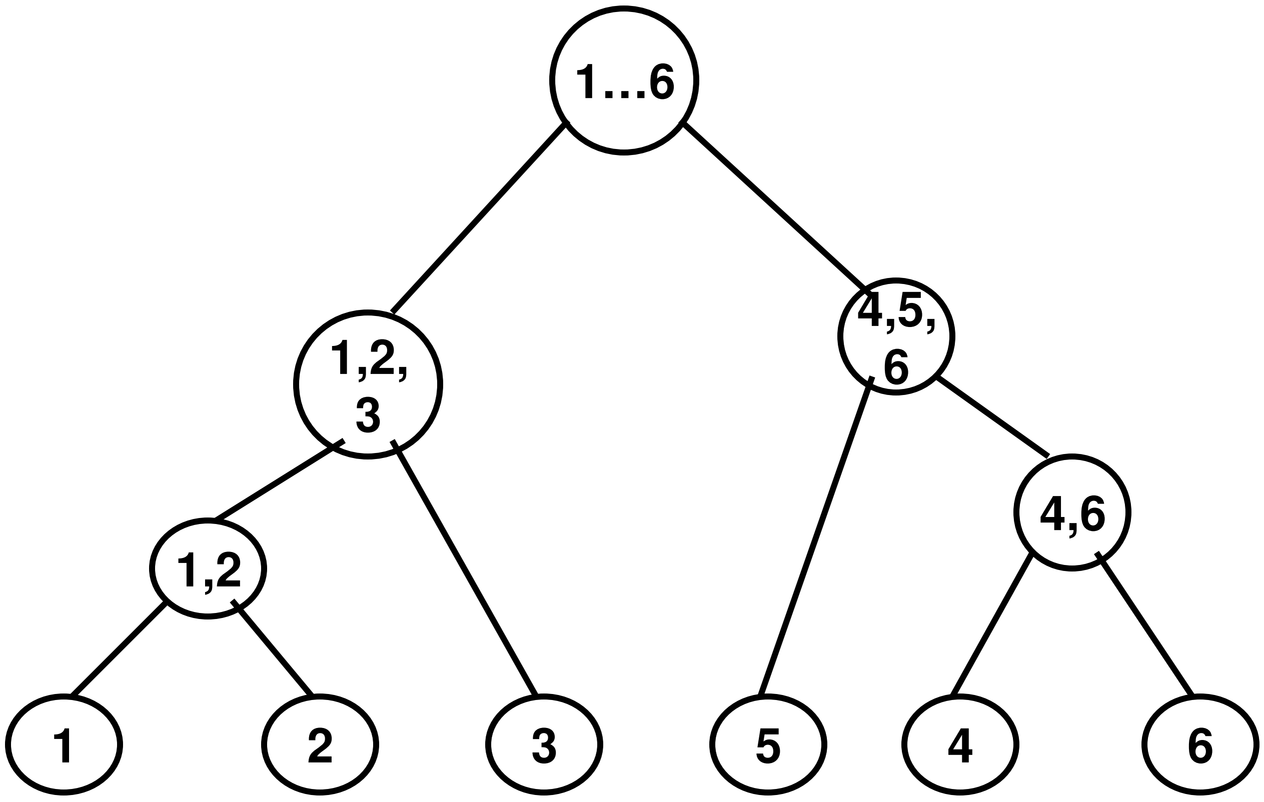
	1,2,3	4	5	6
1,2,3	0	2	2	3
4		0	2	1
5			0	1
6				0

	1,2,3	4,6	5
1,2,3	0	3	2 (5)
4,6		0	2 (3)
5			0 (0)

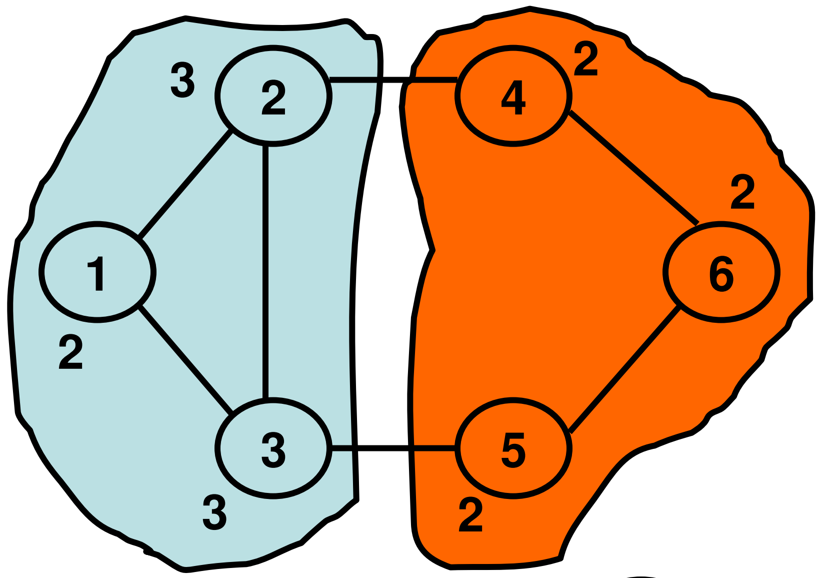


Break the tie by choosing the Pair with the minimum total Pair-wise distance / Pair size
(1,2,3) and (5): 5/4 = 1.25
(4,6) and (5): 3/3 = 1.0



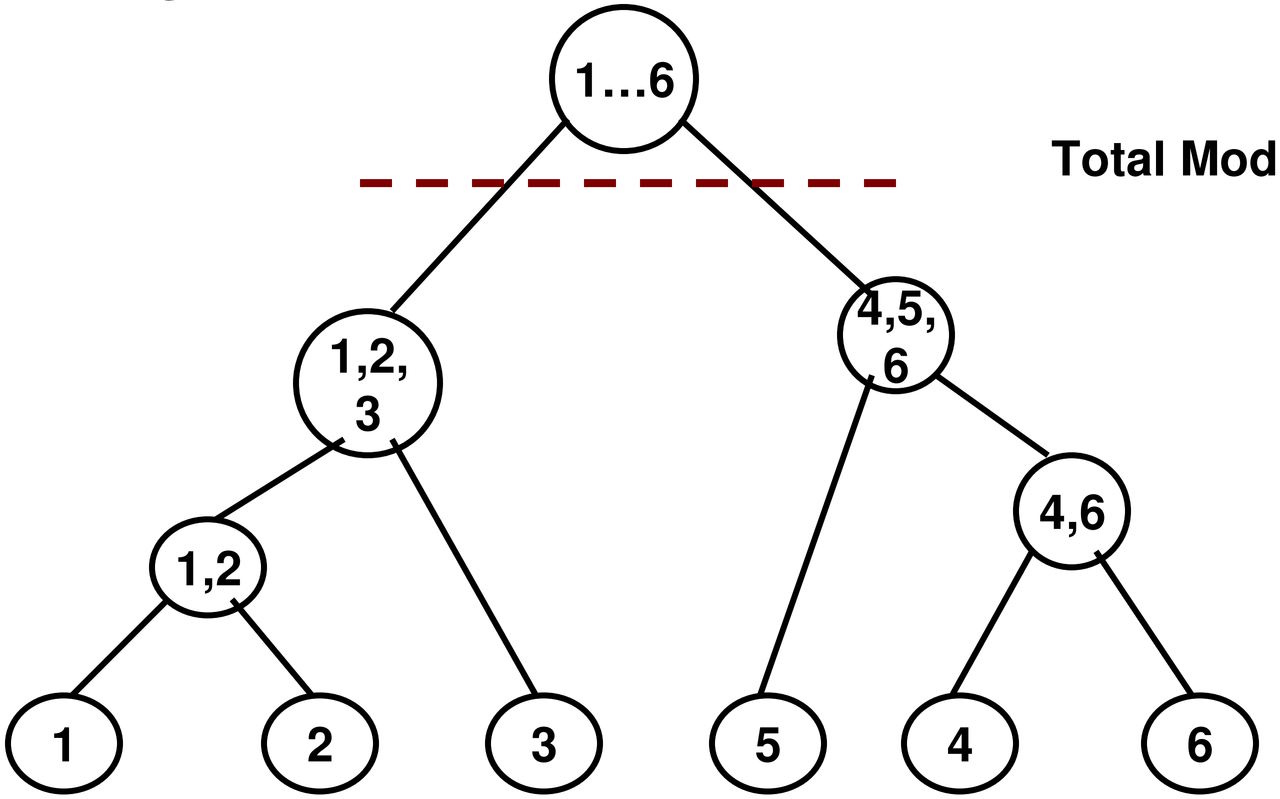


Complete Linkage
Clustering



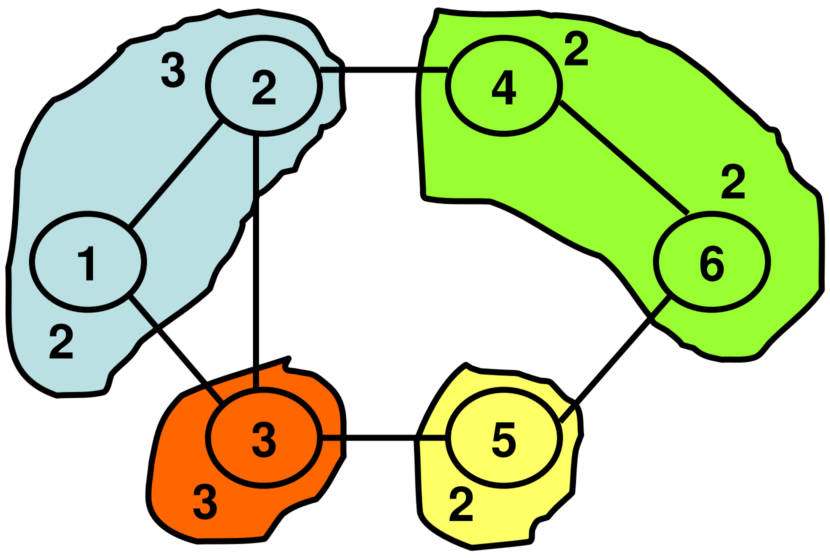
Modularity(1,2,3)
Mod(1,2) = 1 – (2*3)/(2*7) = 0.571
Mod(1,3) = 1 – (2*3)/(2*7) = 0.571
Mod(2,3) = 1 – (3*3)/(2*7) = 0.357

Modularity(4,5,6)
Mod(4,5) = 0 – (2*2)/(2*7) = -0.286
Mod(4,6) = 1 – (2*2)/(2*7) = 0.714
Mod(5,6) = 1 – (2*2)/(2*7) = 0.714

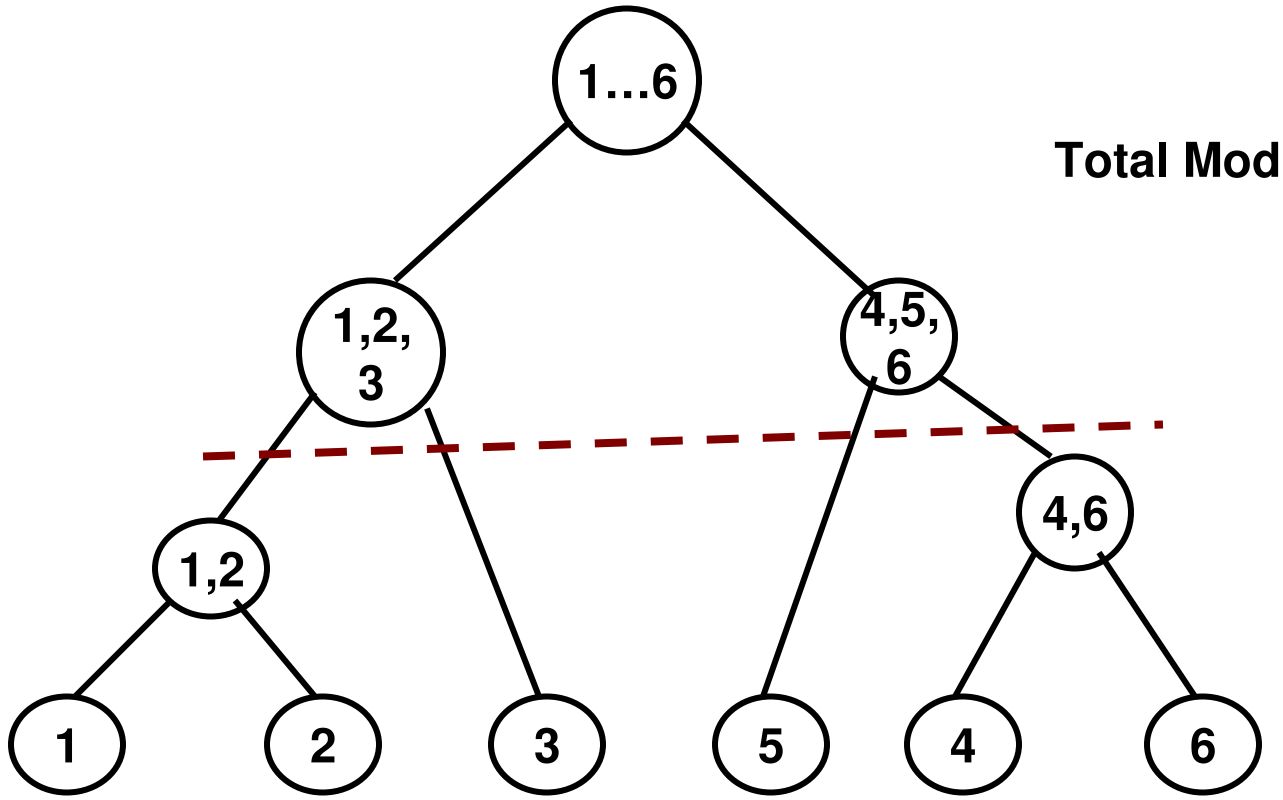


Total Modularity = 2.641

Complete Linkage Clustering



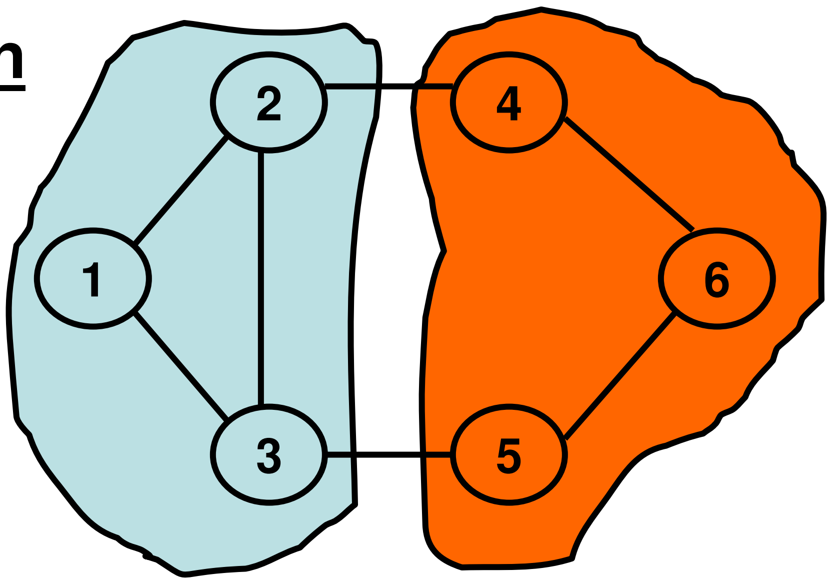
Mod(1,2) = 1 – (2*3)/(2*7) = 0.571
Mod(3) = 0
Mod(5) = 0
Mod(4,6) = 1 – (2*2)/(2*7) = 0.714



Total Modularity = 1.285

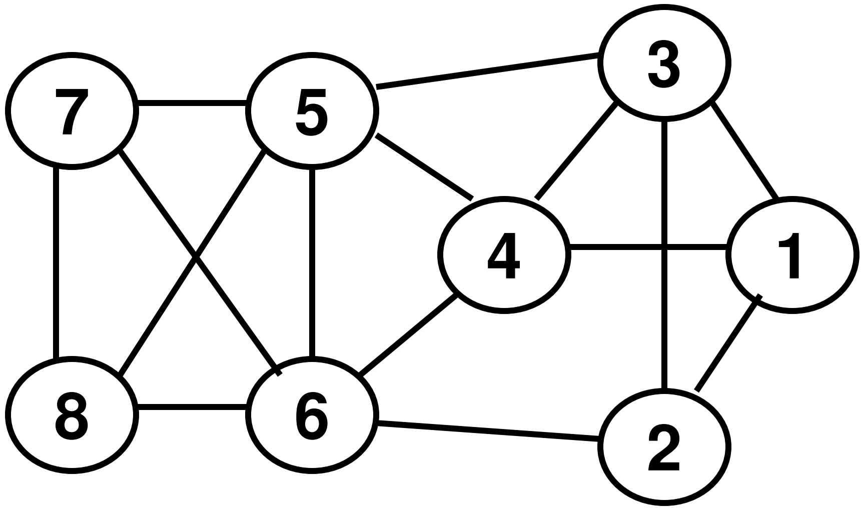
Complete Linkage Clustering

Final Partition



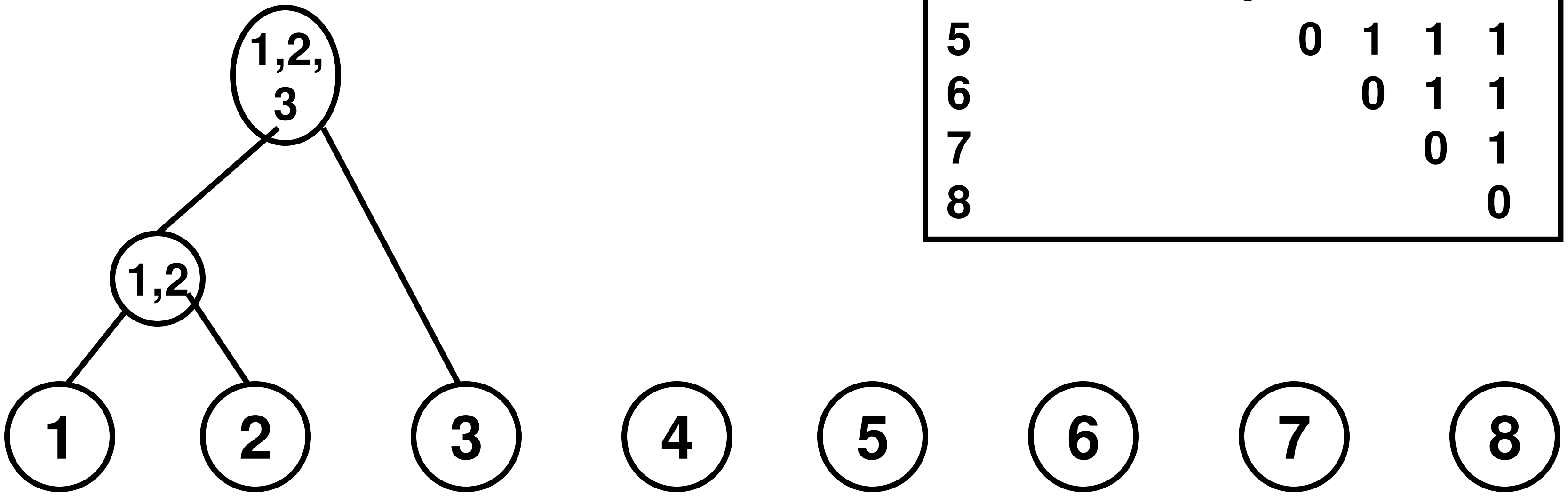
Total Modularity = 2.641

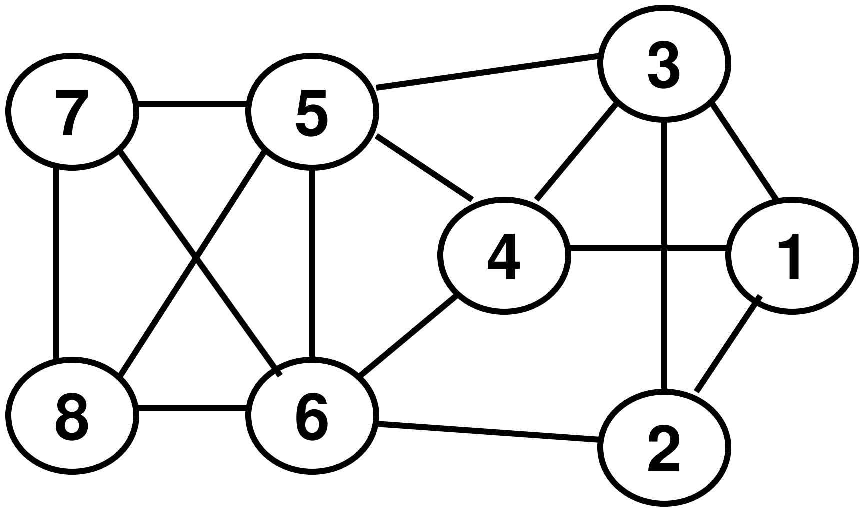
Complete Linkage Clustering (Example 2)



	1	2	3	4	5	6	7	8
1	0	1	1	1	2	2	3	3
2		0	1	2	2	1	2	2
3			0	1	1	2	2	3
4				0	1	1	2	2
5					0	1	1	1
6						0	1	1
7							0	1
8								0

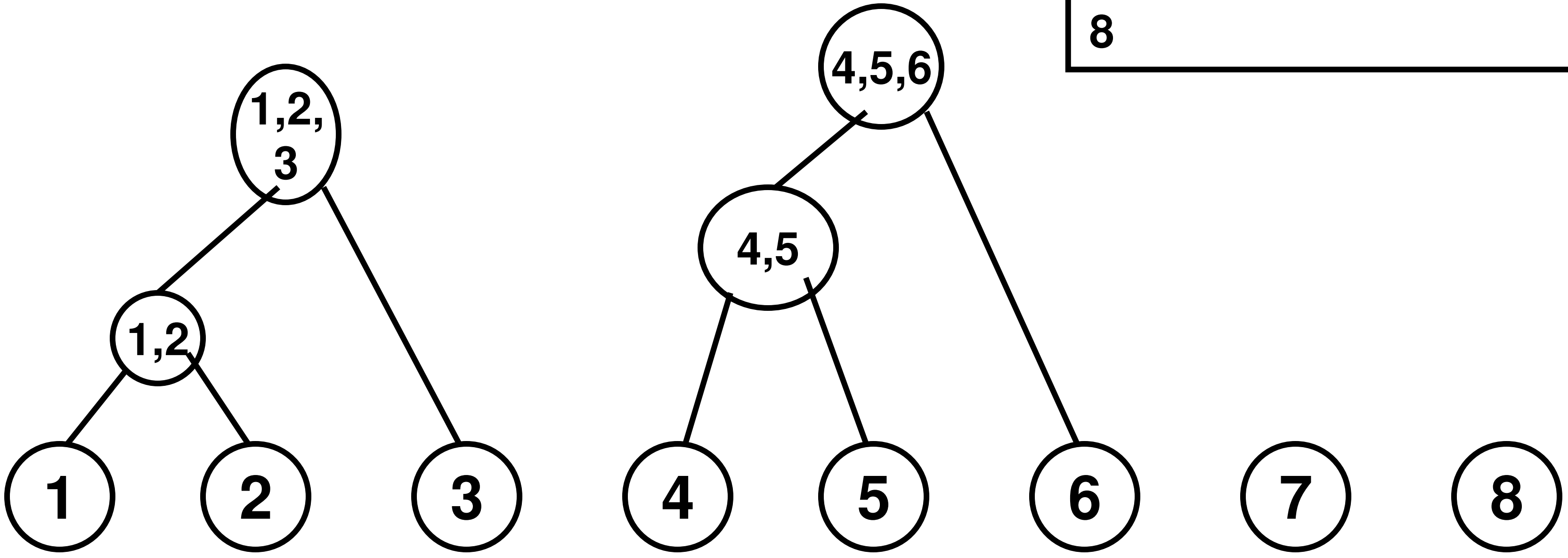
	1,2	3	4	5	6	7	8
1,2	0	1	2	2	2	3	3
3		0	1	1	2	2	3
4			0	1	1	2	2
5				0	1	1	1
6					0	1	1
7						0	1
8							0

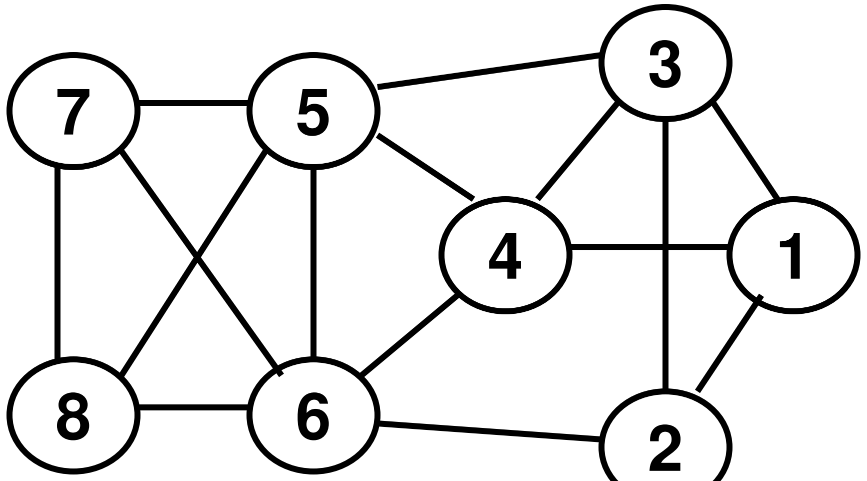




	1,2,3	4	5	6	7	8
1,2,3	0	2	2	2	3	3
4		0	1	1	2	2
5			0	1	1	1
6				0	1	1
7					0	1
8						0

	1,2,3	4,5	6	7	8
1,2,3	0	2	2	3	3
4,5		0	1	2	2
6			0	1	1
7				0	1
8					0

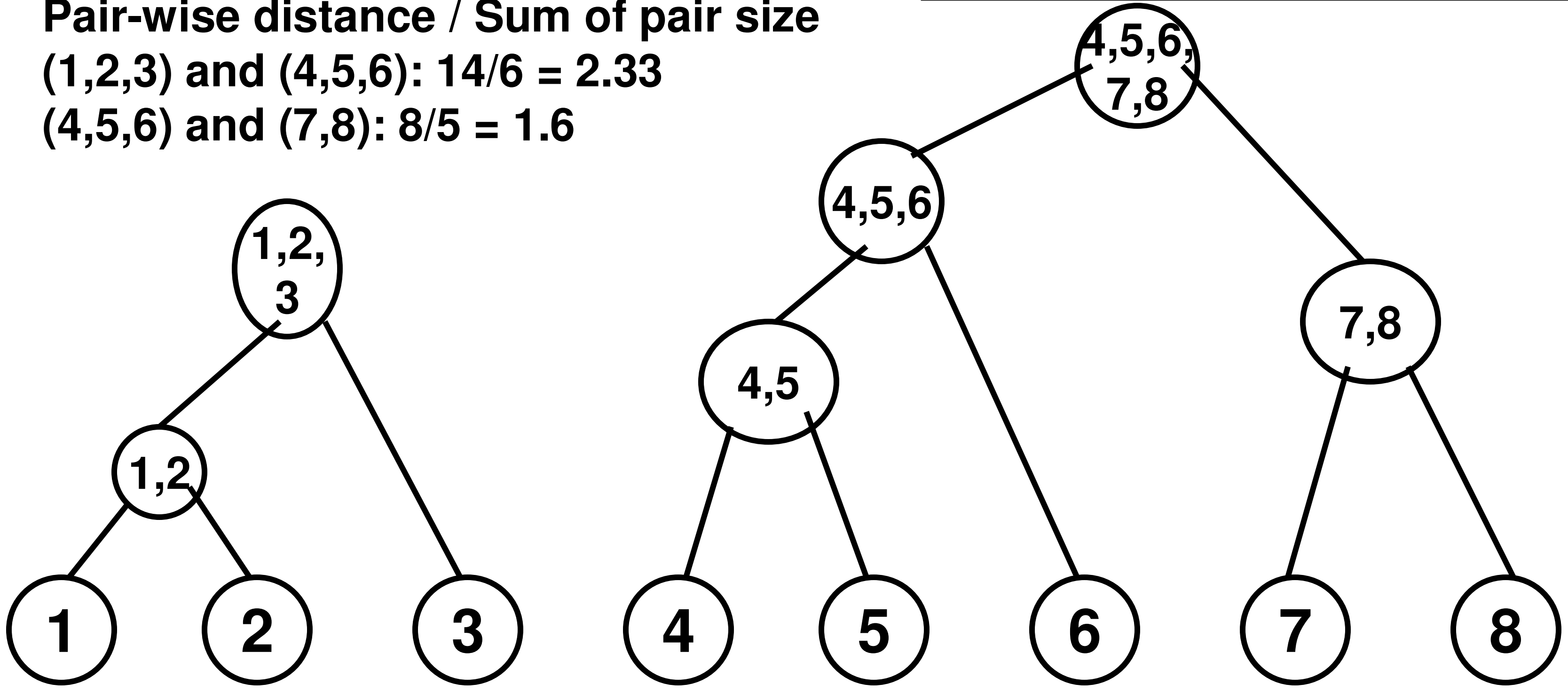


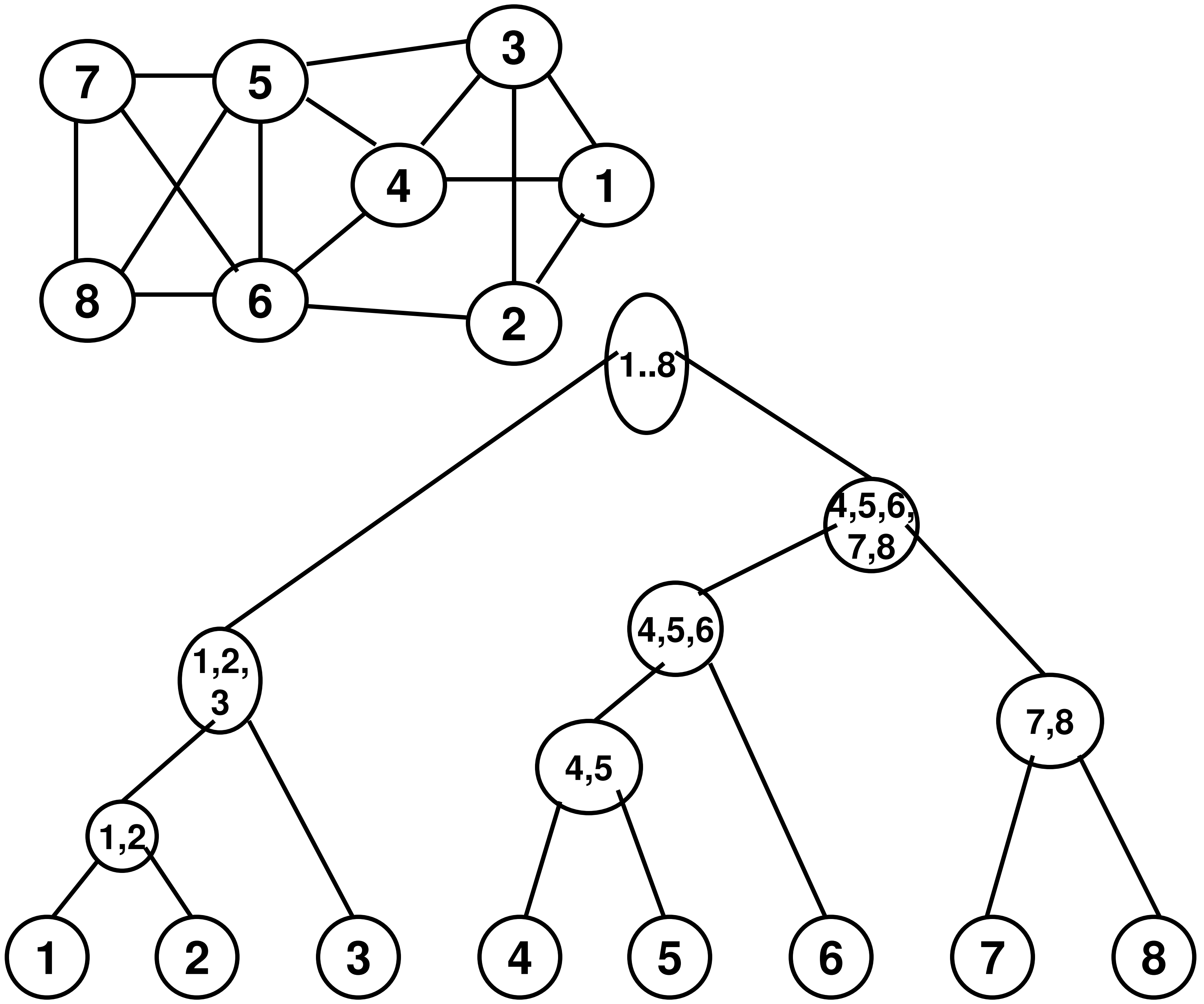


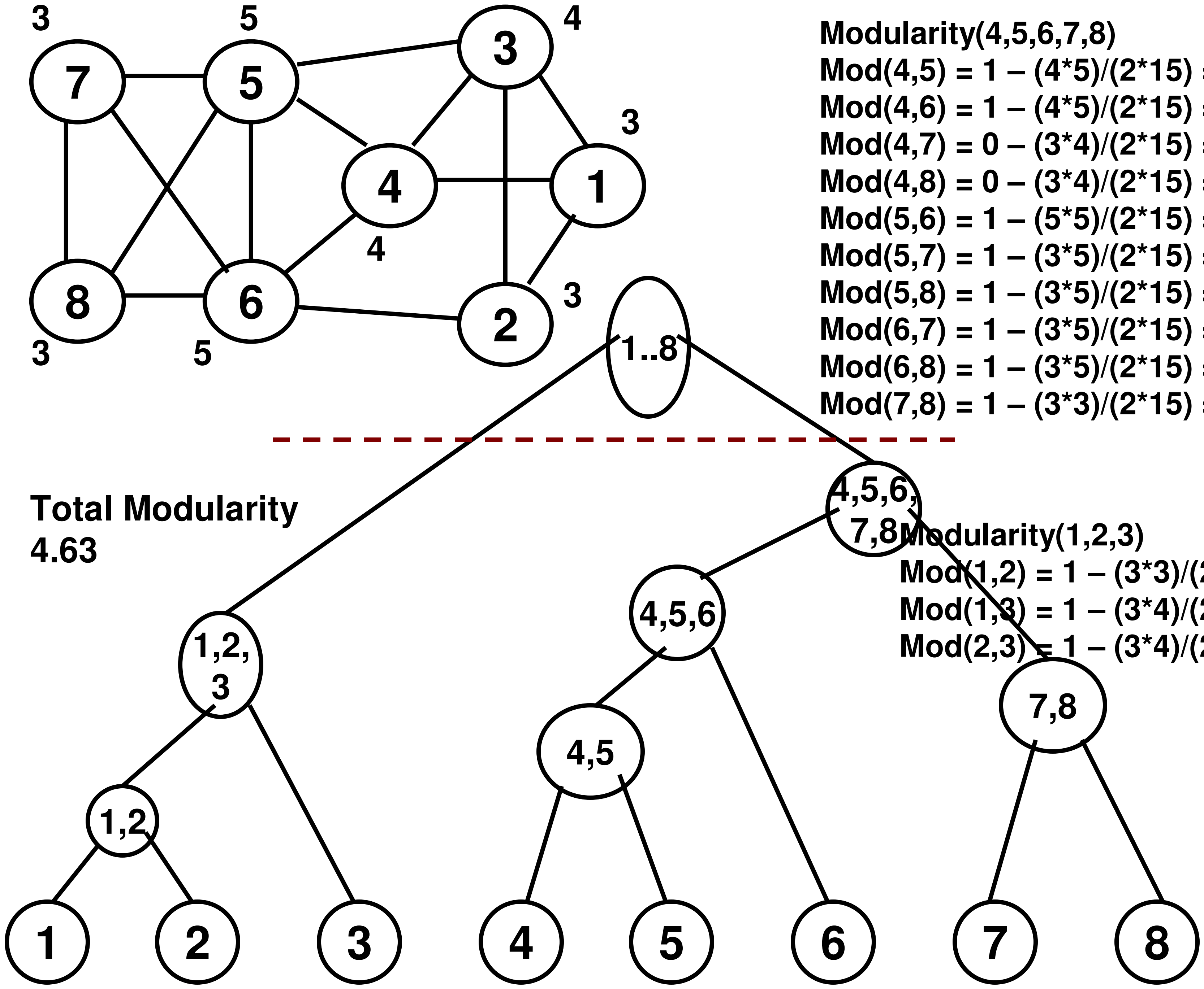
	1,2,3	4,5,6	7	8
1,2,3	0	2	3	3
4,5,6		0	2	2
7			0	1
8				0

	1,2,3	4,5,6	7,8
1,2,3	0	2 (14)	3
4,5,6		0	2 (8)
7,8			0

Break the tie by choosing the Pair with the minimum total Pair-wise distance / Sum of pair size
(1,2,3) and (4,5,6): 14/6 = 2.33
(4,5,6) and (7,8): 8/5 = 1.6




7
5
3
4
1
8
6
2
1..8
4,5,6,
7,8
1,2,
3
4,5,6
7,8
4,5
1,2
1
2
3
4
5
6
7
8

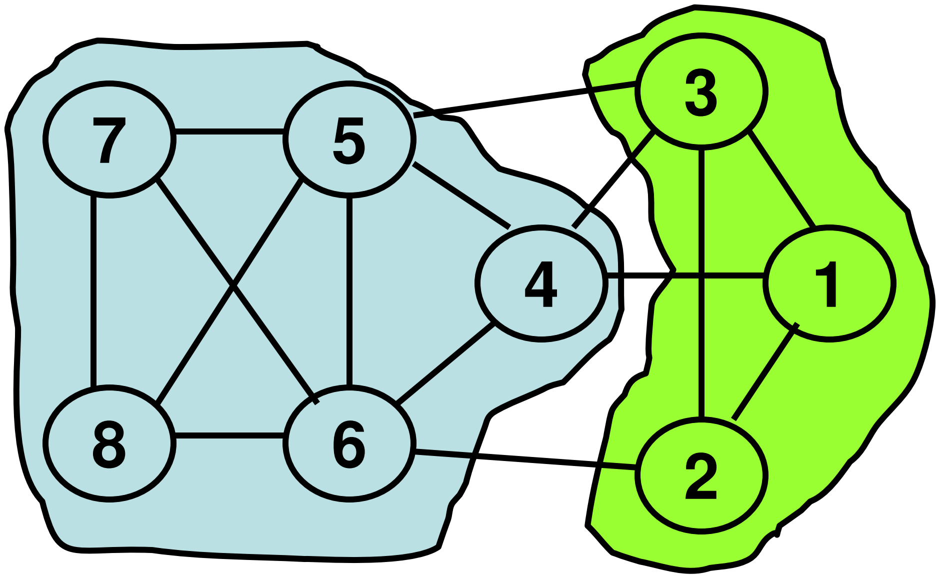


Modularity(4,5,6,7,8)
Mod(4,5) = 1 – (4*5)/(2*15) = 0.33
Mod(4,6) = 1 – (4*5)/(2*15) = 0.33
Mod(4,7) = 0 – (3*4)/(2*15) = -0.4
Mod(4,8) = 0 – (3*4)/(2*15) = -0.4
Mod(5,6) = 1 – (5*5)/(2*15) = 0.17
Mod(5,7) = 1 – (3*5)/(2*15) = 0.50
Mod(5,8) = 1 – (3*5)/(2*15) = 0.50
Mod(6,7) = 1 – (3*5)/(2*15) = 0.50
Mod(6,8) = 1 – (3*5)/(2*15) = 0.50
Mod(7,8) = 1 – (3*3)/(2*15) = 0.70

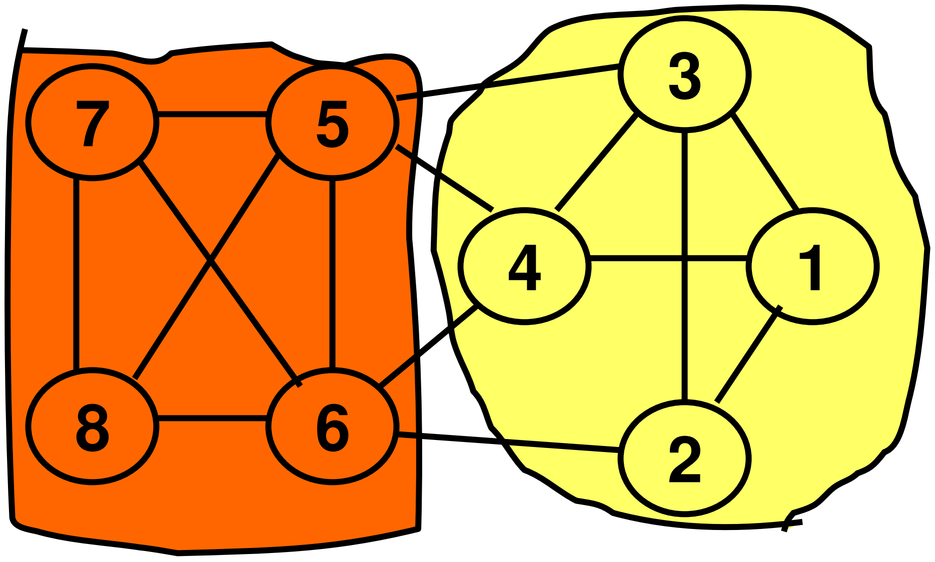
Total Modularity
4.63

Modularity(1,2,3)
Mod(1,2) = 1 – (3*3)/(2*15) = 0.70
Mod(1,3) = 1 – (3*4)/(2*15) = 0.60
Mod(2,3) = 1 – (3*4)/(2*15) = 0.60

Final Partition



From the previous slides,
We know that the optimal
Partitioning of the graph is:



Total Modularity = 4.63

Complete Linkage Clustering

Modularity (1, 2, 3, 4) +
Modularity (5, 6, 7, 8) = 5.54

Thus, complete linkage clustering need not always give the optimal solution.

Hierarchical Clustering Edge Betweenness

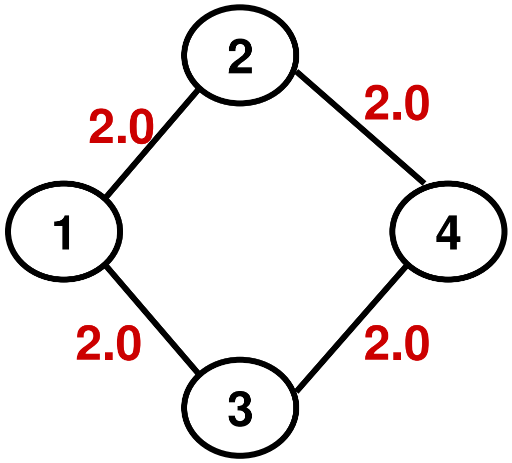
Top-Down Approach

(Divisional)

Edge Betweenness

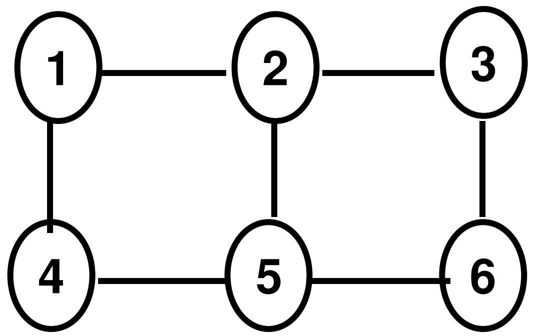
- Edge Betweenness (EdgeBW) is a measure of the number of shortest paths the edge is part of
  - To be exact, it is the sum of the fraction of the shortest paths going through the edge between any two vertices.

Example 1



Each fraction is **1/number of paths for the pair**

Pair	**Paths**	**Edges** 1-2	1-3	2-4	3-4
1, 2:	1-2	1/1			
1, 3:	1-3		1/1		
1, 4:	1-2-4, 1-3-4	1/2	1/2	1/2	1/2
2, 3:	2-1-3, 2-4-3	1/2	1/2	1/2	1/2
2, 4:	2-4			1/1	
3, 4:	3-4				1/1
EdgeBW (sum fractions)		**2.0**	**2.0**	**2.0**	**2.0**



Edge Betweenness

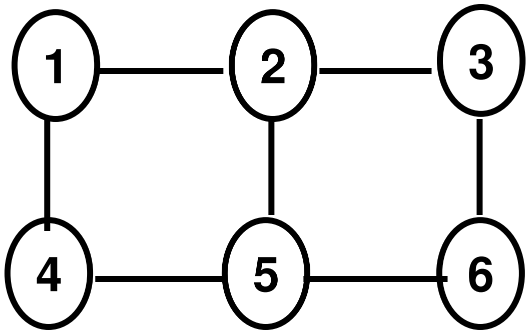
Example 2

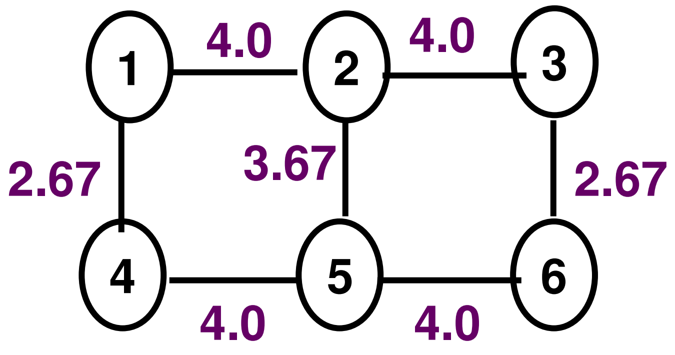
Each fraction is **1/number of paths for the pair**

Pair	Paths	1-2	1-4	2-3	2-5	3-6	4-5	5-6
1, 2	1-2	1/1						
1, 3	1-2-3	1/1		1/1				
1, 4	1-4		1/1					
1, 5	1-2-5	1/2			1/2			
	1-4-5		1/2				1/2	
1, 6	1-2-3-6	1/3		1/3		1/3		
	1-4-5-6		1/3				1/3	1/3
	1-2-5-6	1/3			1/3			1/3
2, 3	2-3			1/1				
2, 4	2-1-4	1/2	1/2					
	2-5-4				1/2		1/2	
2, 5	2-5				1/1			
2, 6	2-3-6			1/2		1/2		
	2-5-6				1/2			1/2
	Sum (partial)	3.67	2.33	2.83	2.83	0.83	1.33	1.17

Edge Betweenness

Example 2 (continued…)



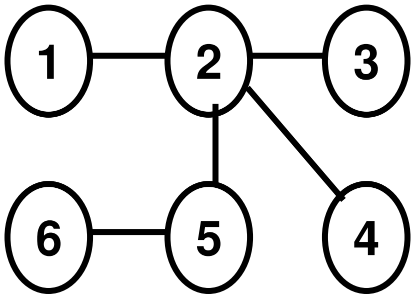


Pair	Paths	1-2	1-4	2-3	2-5	3-6	4-5	5-6
	Sum (partial)	3.67	2.33	2.83	2.83	0.83	1.33	1.17
3, 4	3-2-1-4	1/3	1/3	1/3				
	3-6-5-4					1/3	1/3	1/3
	3-2-5-4			1/3	1/3		1/3	
3, 5	3-2-5			1/2	1/2			
	3-6-5					1/2		1/2
3, 6	3-6					1/1		
4, 5	4-5						1/1	
4, 6	4-5-6						1/1	1/1
5, 6	5-6							1/1
EdgeBWC (Sum)		4.0	2.67	4.0	3.67	2.67	4.0	4.0

Edge Betweenness

- Edge Betweenness (EdgeBW) is a measure of the number of shortest paths the edge is part of: sum of the fraction of the shortest paths going through the edge between any two vertices.
- Note that in this graph below, there is only one shortest path between any two nodes. Hence, the EdgeBW is simply the # shortest paths through that edge

Example 3

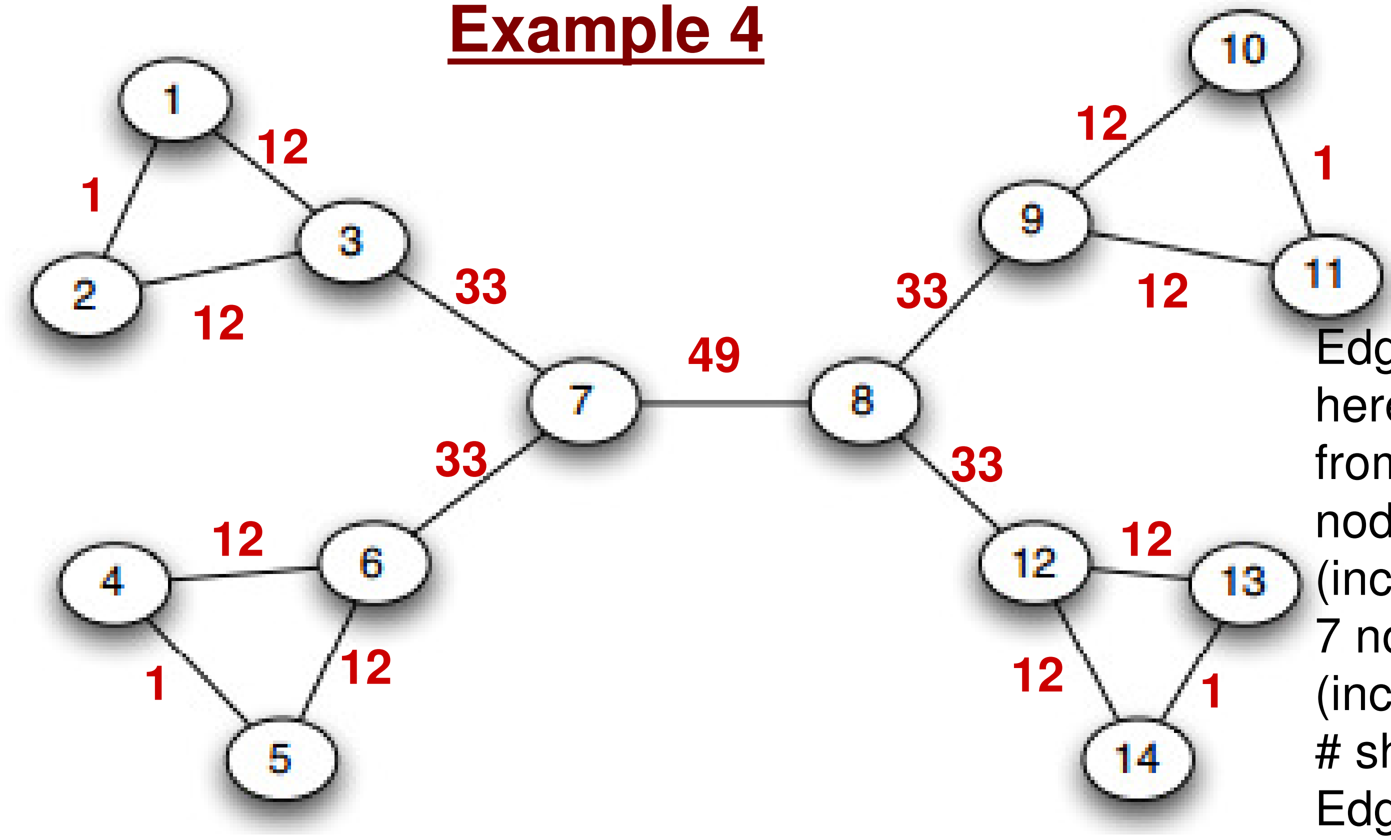


Pair	Paths	Edges				
		1-2	2-3	2-4	2-5	5-6
1, 2	1-2	1/1				
1, 3	1-2-3	1/1	1/1			
1, 4	1-2-4	1/1		1/1		
1, 5	1-2-5	1/1			1/1	
1, 6	1-2-5-6	1/1			1/1	1/1
2, 3	2-3		1/1			
2, 4	2-4			1/1		
2, 5	2-5				1/1	
2, 6	2-5-6				1/1	1/1
3, 4	3-2-4		1/1	1/1		
3, 5	3-2-5		1/1		1/1	
3, 6	3-2-5-6		1/1		1/1	1/1
4, 5	4-2-5			1/1	1/1	
4, 6	4-2-5-6			1/1	1/1	1/1
5, 6	5-6					1/1
EdgeBW	(Sum)	5.0	5.0	5.0	8.0	5.0

Edge Betweenness

- Edge Betweenness (EdgeBW) is a measure of the number of shortest paths the edge is part of: sum of the fraction of the shortest paths going through the edge between any two vertices.
- Note that in this graph below, there is only one shortest path between any two nodes. Hence, the EdgeBW is simply the # shortest paths through that edge

Example 4



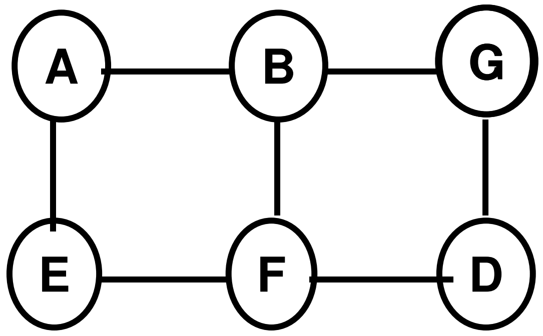
Edge 7-8 in the graph here carries info from each of the 7 nodes on the left (incl. node 7) and the 7 nodes on the right (incl. node 8)
# shortest paths through Edge 7-8 is 7*7 = 49

Finding the # Shortest Paths through an Edge

- For graphs in which there is more than one paths between one or more pair of vertices, the total betweenness of an edge is not equal to the total # shortest paths through the edge.

- We will now see an algorithm proposed by Girvan and Newman to determine the total betweenness of an edge.
- Repeat the following for every vertex
  - Perform a Breadth First Search (BFS) of the graph, starting from the first vertex, say A.
  - Determine the # shortest paths from A to each other node using the BFS levels of the nodes
  - Based on the above numbers of shortest paths, determine the amount of info from A to all the other vertices that uses each edge.

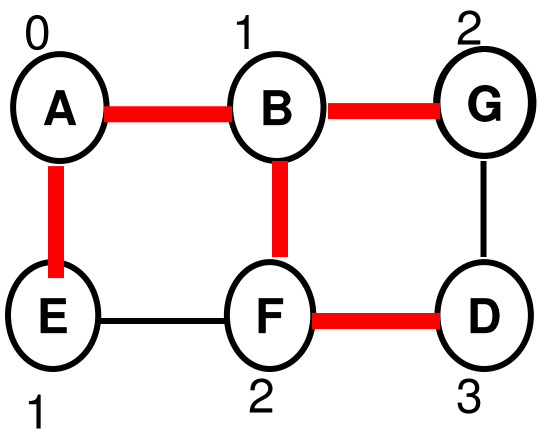
- The total betweenness through an edge is the sum (for directed graph) or half of the sum (for undirected graph) of the betweenness determined through that edge when BFS is run from every vertex in the graph.
  - For undirected graph, we divide by the total sum of the BWs by 2 because an edge is counted twice on the shortest path between any two vertices.
  - For example A – B – C; the edge A – B is counted twice (once on the shortest path from A to C and once on the shortest path from C to A



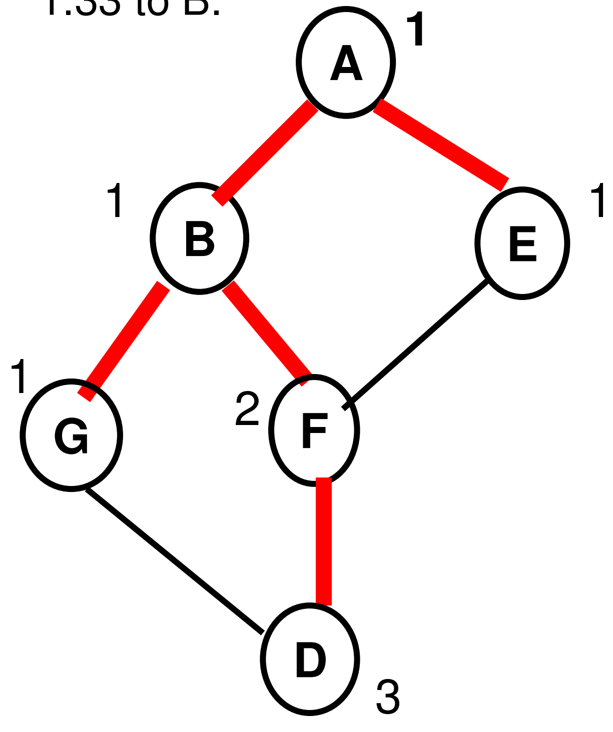
Computing the Fraction of Shortest Path Values

We assume one unit of info originates at each node. We start with the node at the bottom most level. Let 1 unit of info start from node D. Node D gets 2 of its Shortest paths to node A through F and 1 through G. So, node D sends 2/3 of the info to F and 1/3 of the info to G. Node F adds 2/3 info received to 1 unit of info originating at itself and splits the resulting 1.67 equally and sends 0.835 to each of B and E. G merely adds the 0.33 info units to the 1 units of info originating at itself and sends 1.33 to B.

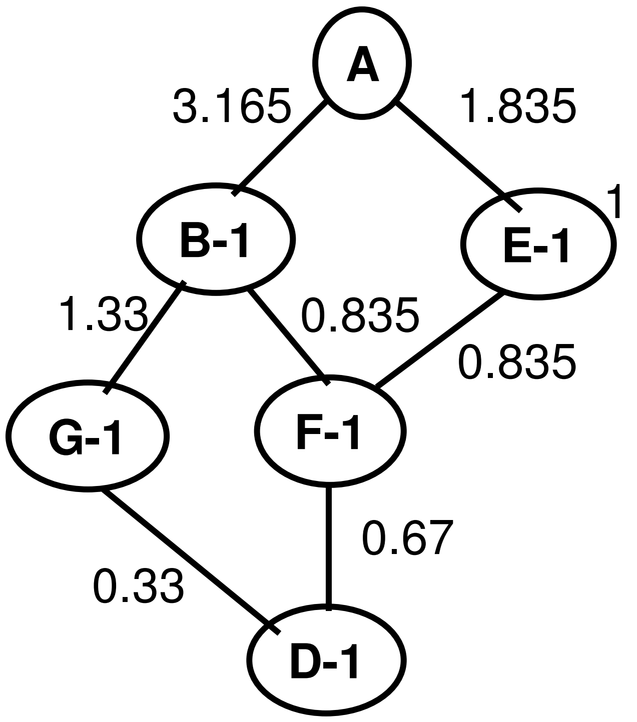
BFS run on A



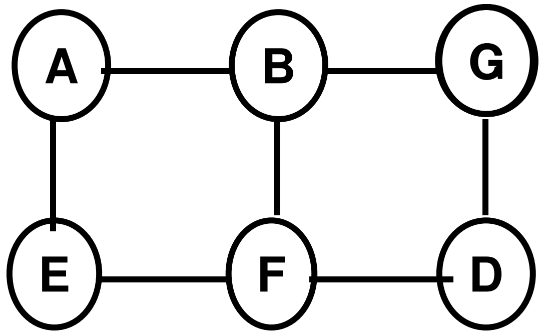
Node Levels

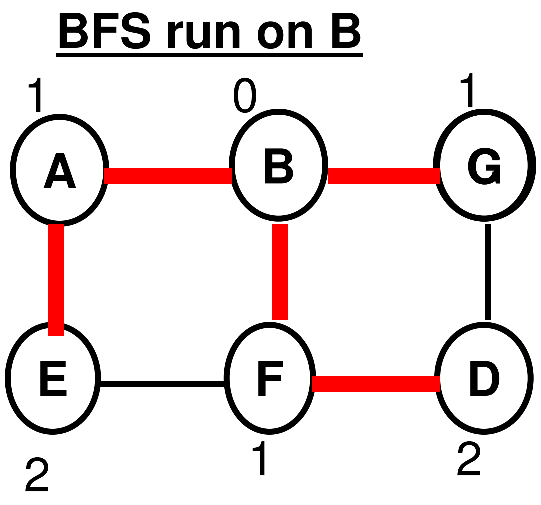


# Shortest Paths from Node A to every other Node

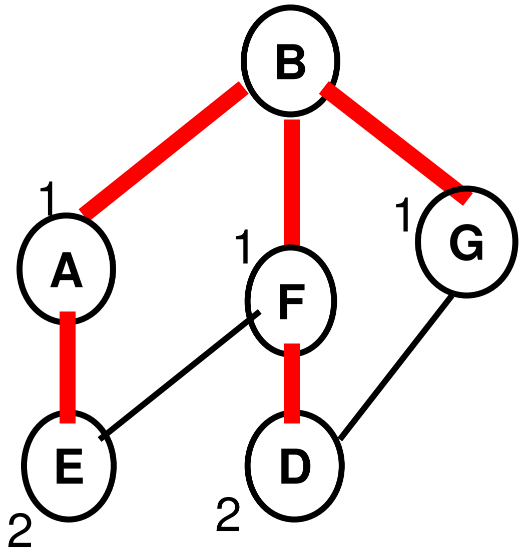


BW on each edge

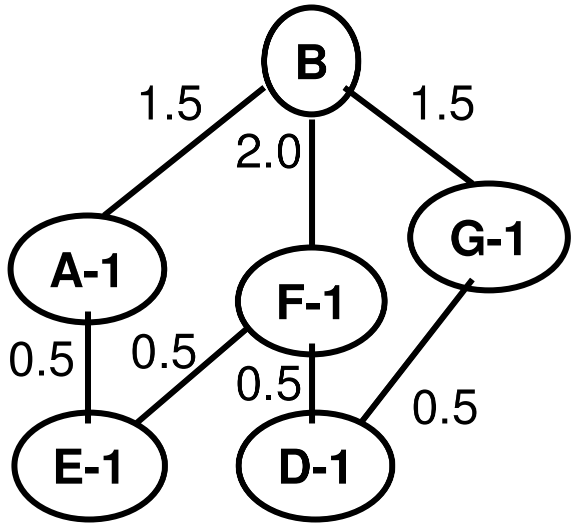




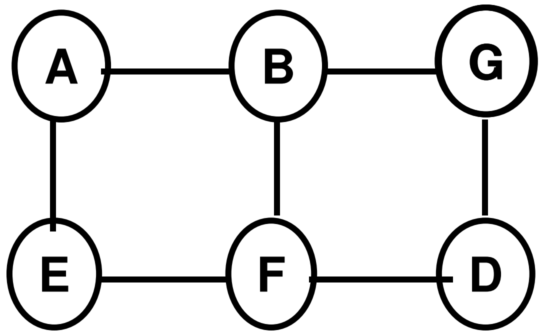
Node Levels



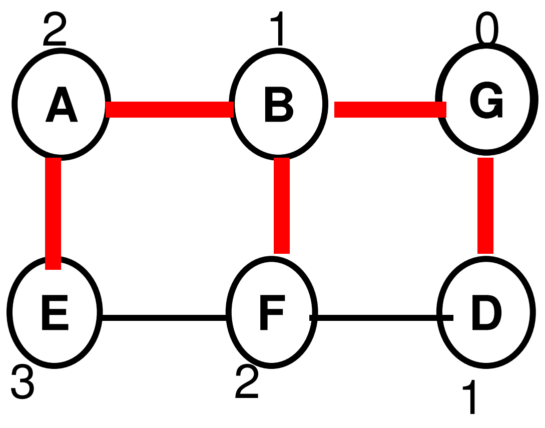
# Shortest Paths from Node B to every other Node



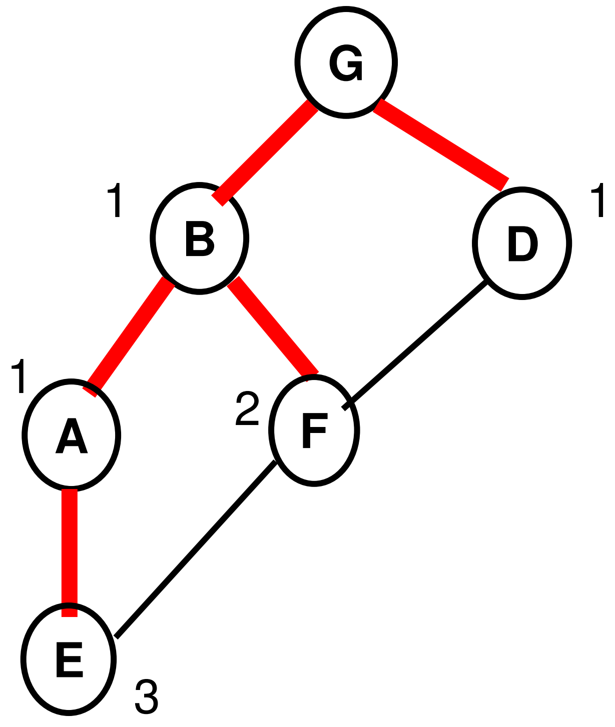
BW on each edge



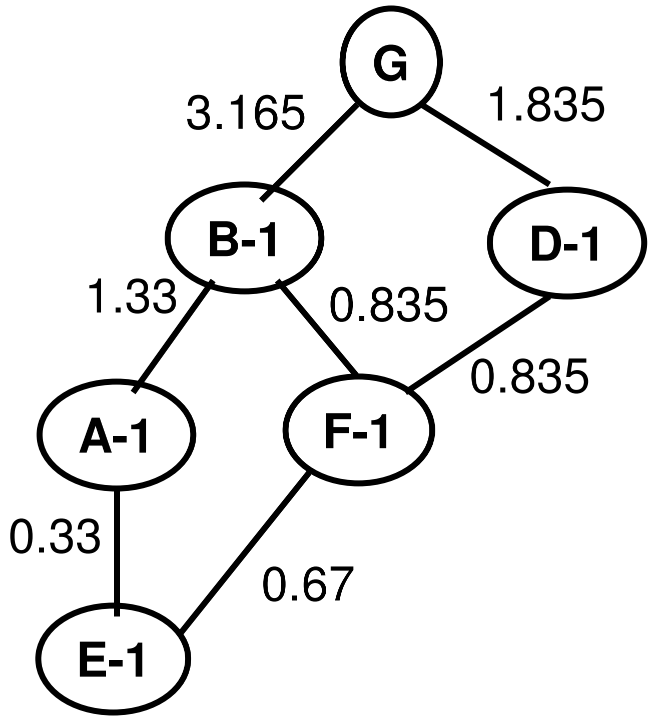
BFS run on G



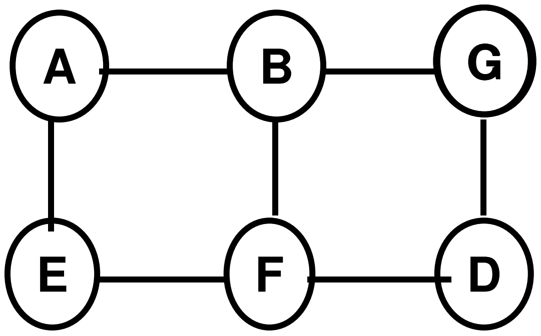
Node Levels



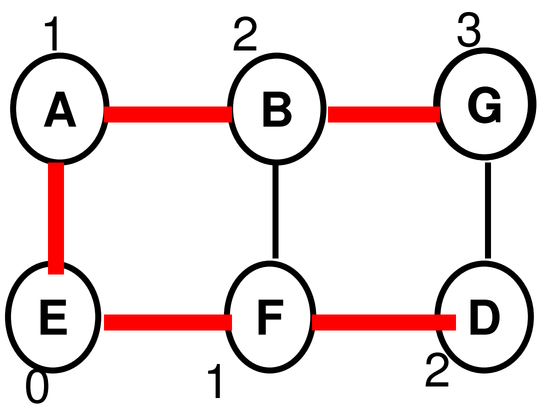
# Shortest Paths from Node G to every other Node



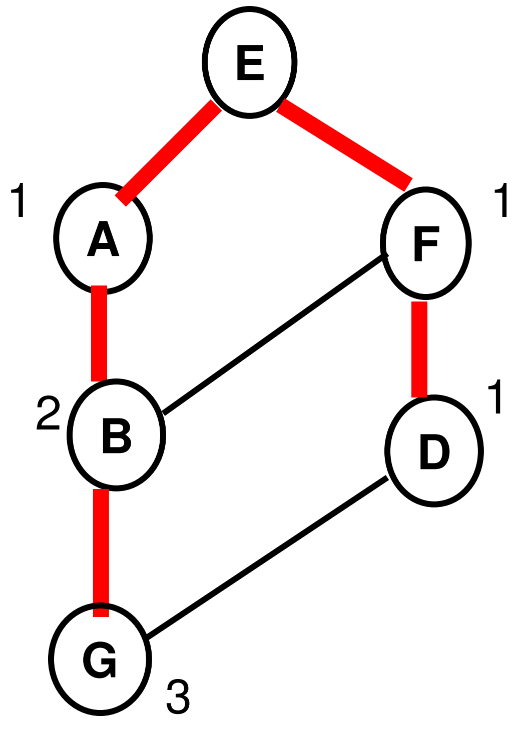
BW on each edge



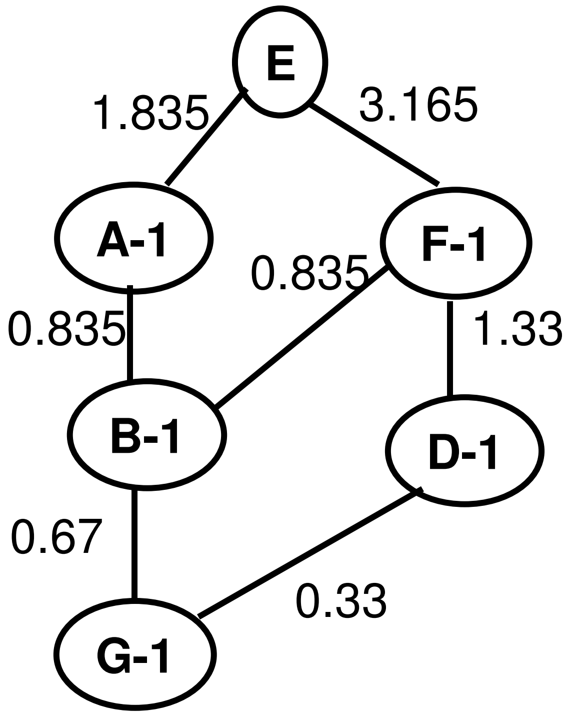
BFS run on E



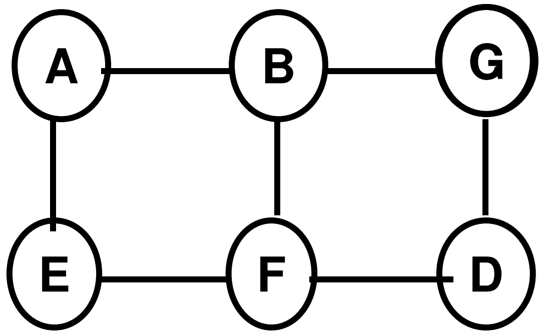
Node Levels



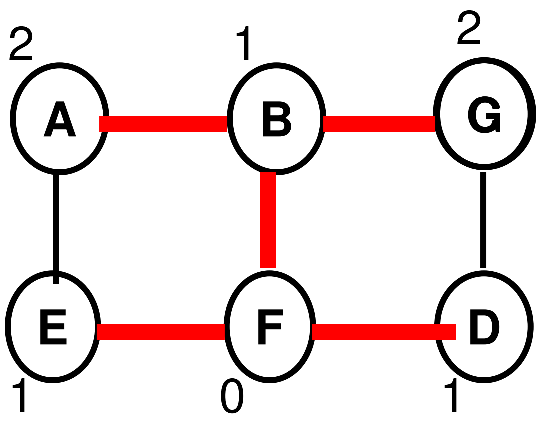
# Shortest Paths from Node E to every other Node



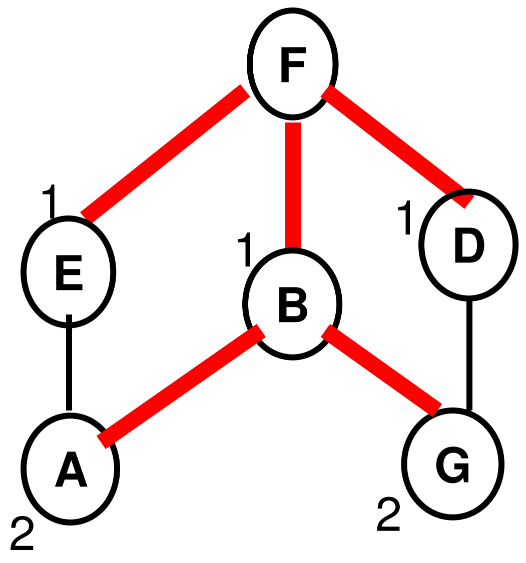
BW on each edge



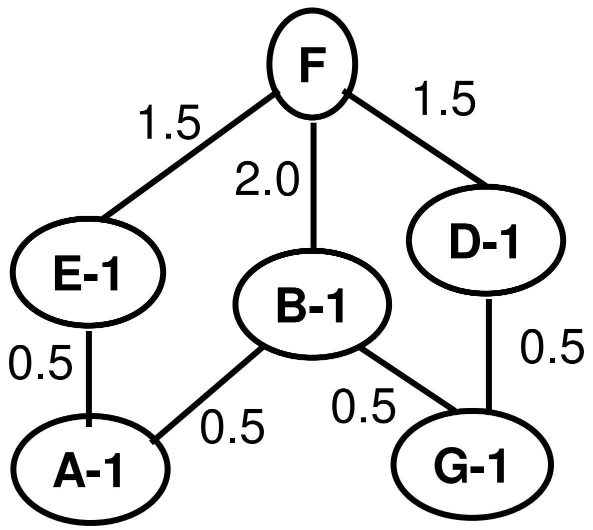
BFS run on F



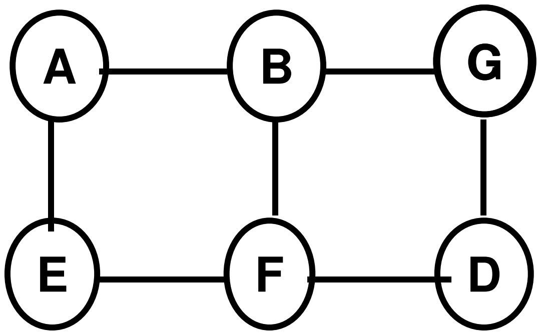
Node Levels



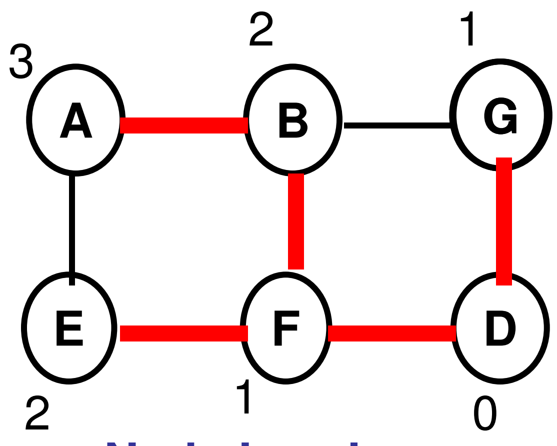
# Shortest Paths from Node F to every other Node



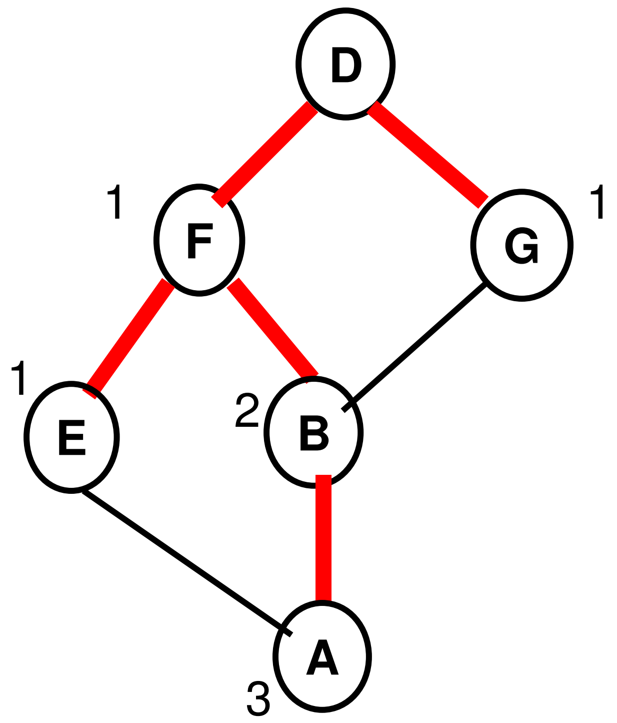
BW on each edge



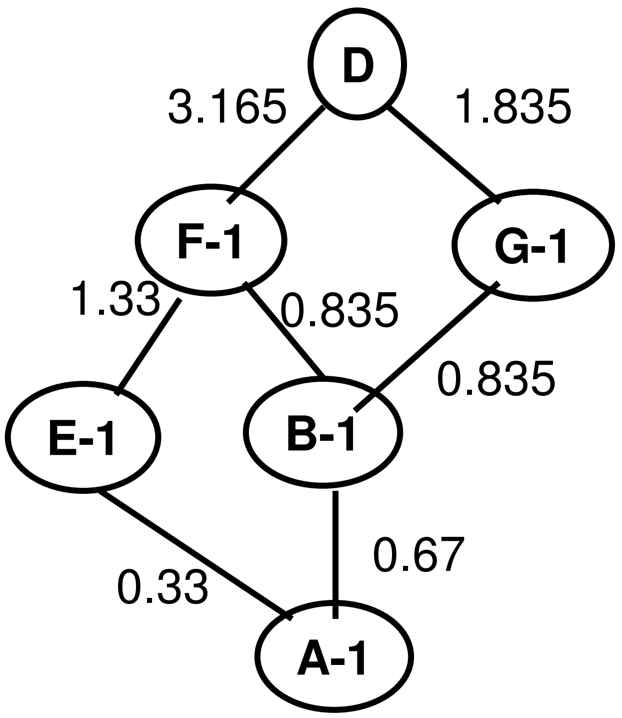
BFS run on D



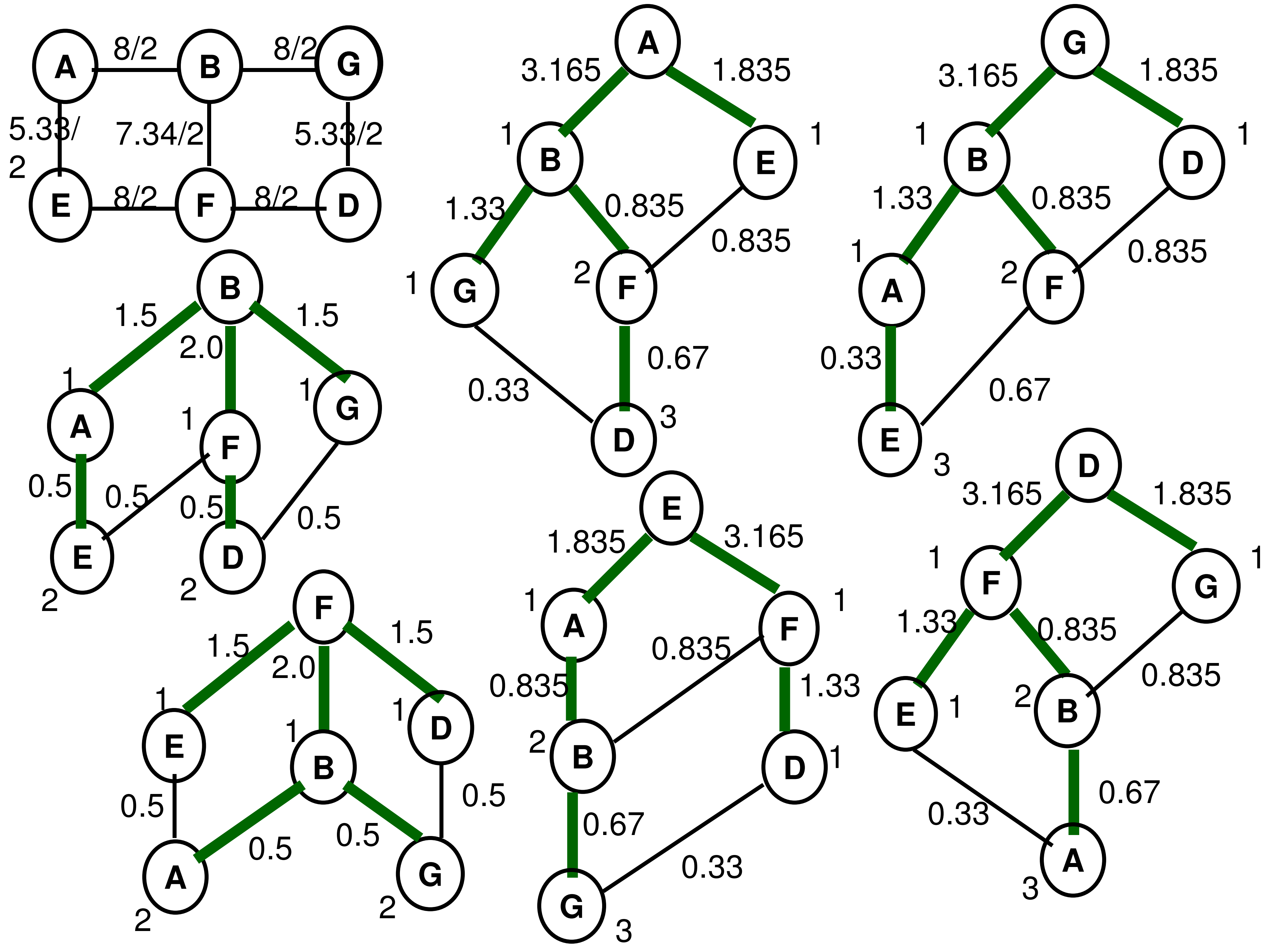
Node Levels



# Shortest Paths from Node D to every other Node



BW on each edge



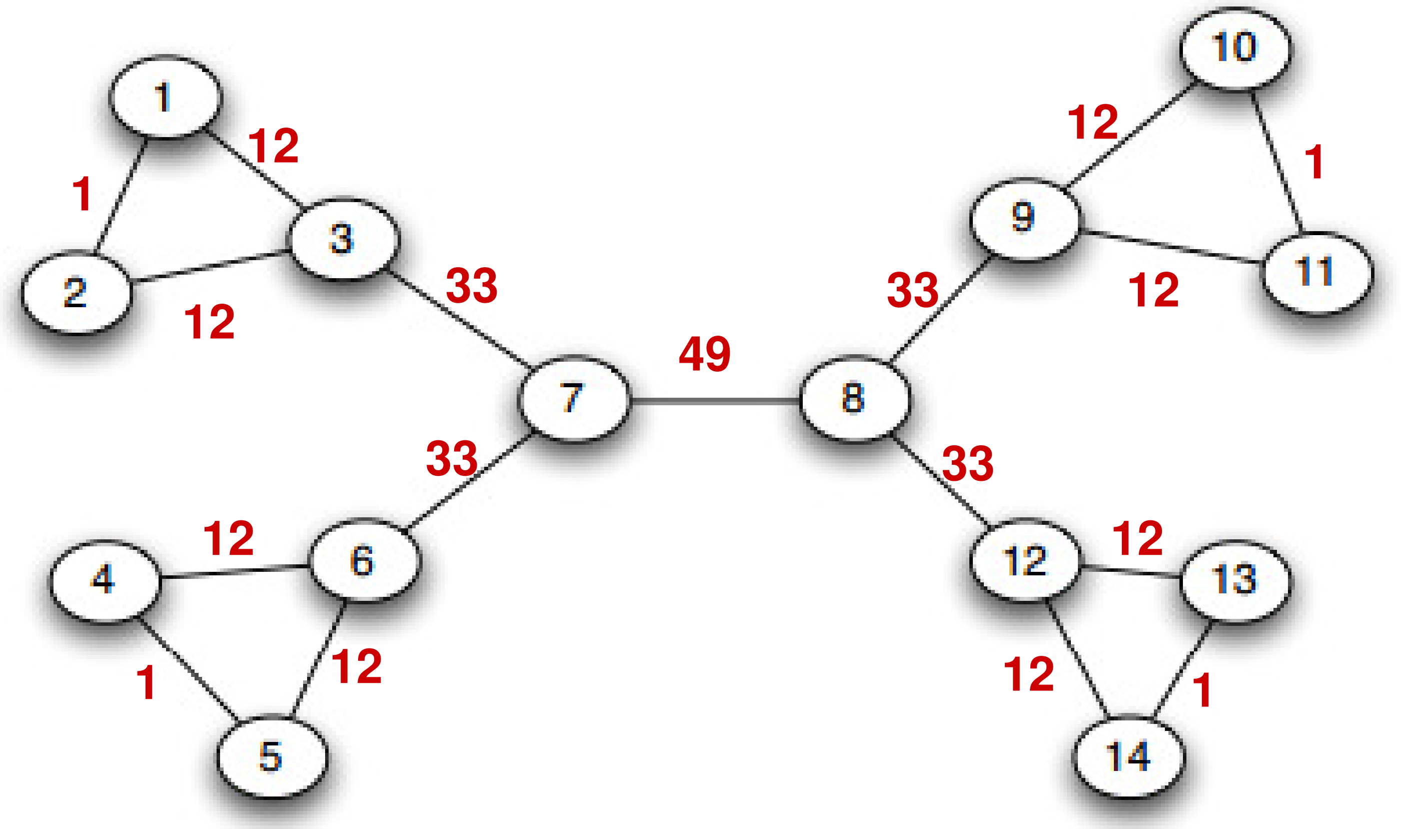
A
B
G
E
F
D
8/2
8/2
5.33/2
7.34/2
5.33/2
8/2
8/2
3.165
1.835
1.33
0.835
0.835
0.67
0.33
1.5
2.0
0.5
1.33
0.67
1
2
3

Girvan-Newman (GN) Algorithm

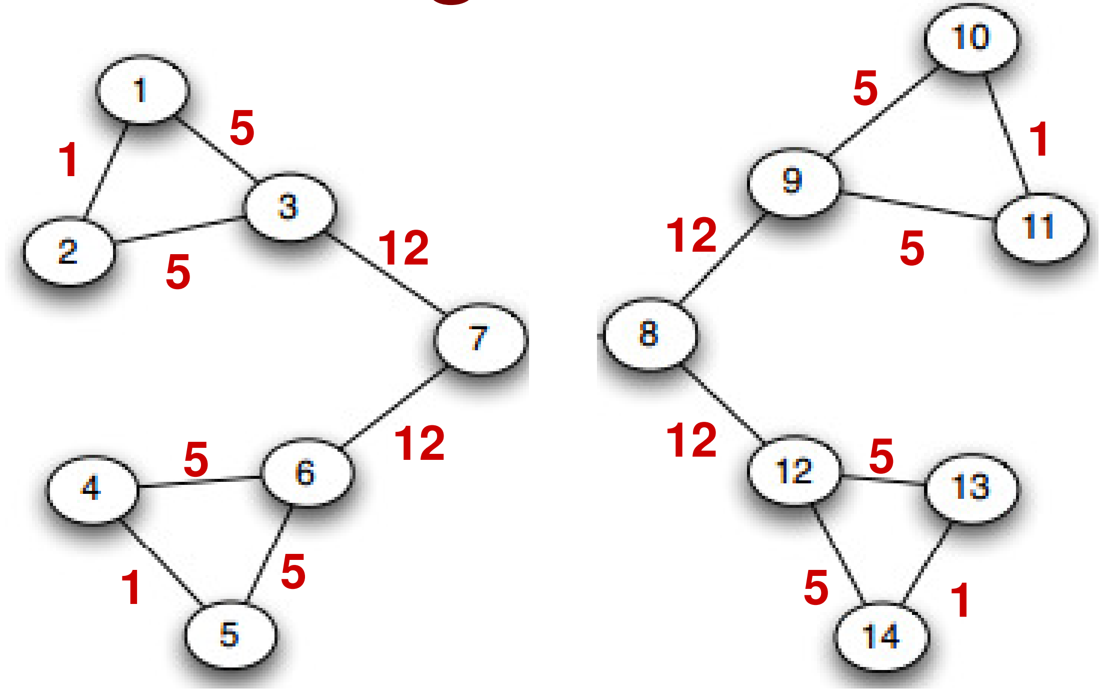
- Proceeds in iterations
- At the beginning of each iteration, we compute the Betweenness of the edges in the graph and remove the edge(s) with the largest betweenness.
  - If more than one edge has the largest betweenness, remove all such competing edges at the same time.
  - If the graph gets disconnected to two or more communities (components), we compute the total modularity score of the resulting communities
- Repeat the iterations until there are no more edges
- The partition (set of communities) with the total modularity score is the optimal partition.
- **We could stop dividing a community if the total modularity score of the undivided community is greater than the total modularity score of the divided community.**

- **You could use the Java program (EdgeBWC.java) given to you to compute the betweenness of the edges in each iteration.**
- **You could use the Java program (PairwiseModularity.java) given to you to compute the modularity scores of the pairs of vertices in the original graph and use these scores in your modularity calculations for each iteration.**

GN Algorithm: Example 1



GN Algorithm: Example 1 (It # 1)

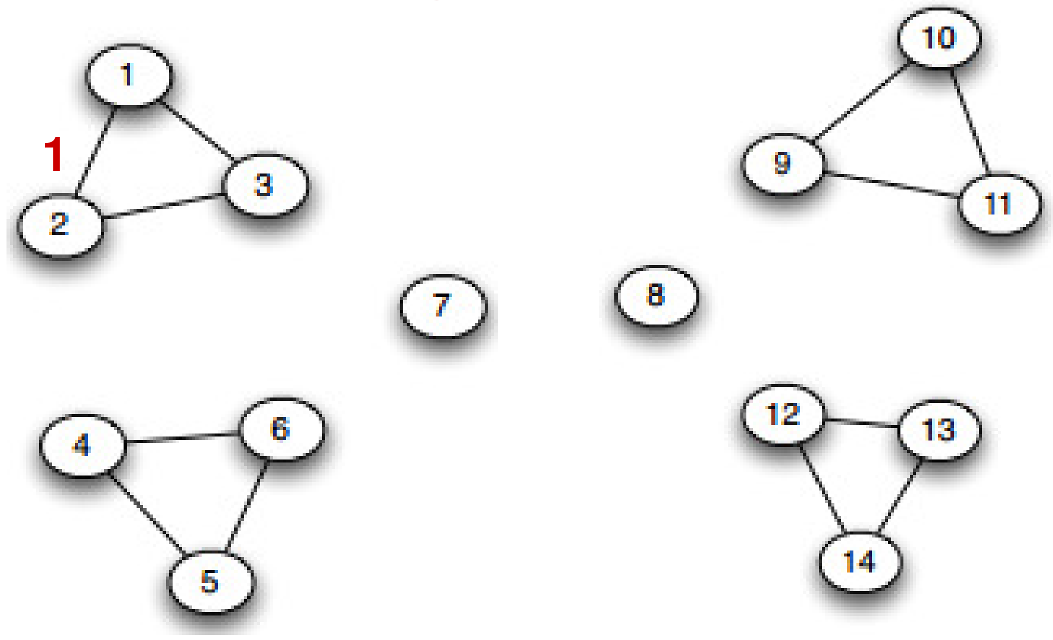


After removing edge 7-8 with the largest BW (49)

Modularity (1, 2, 3, …, 7) = 4.385
Modularity (8, 9, 10, …, 17) = 4.385
Total Modularity Score = 8.77

Modularity (1, 2, 3, 4 , 5, 6, 7)
Mod (1, 2) = 1 – (2*2)/(2*17) = 0.882
Mod (1, 3) = 1 – (2*3)/(2*17) = 0.824
Mod (1, 4) = 0 – (2*2)/(2*17) = -0.118
Mod (1, 5) = 0 – (2*2)/(2*17) = -0.118
Mod (1, 6) = 0 – (2*3)/(2*17) = -0.176
Mod (1, 7) = 0 – (2*3)/(2*17) = -0.176
Mod (2, 3) = 1 – (2*3)/(2*17) = 0.824
Mod (2, 4) = 0 – (2*2)/(2*17) = -0.118
Mod (2, 5) = 0 – (2*2)/(2*17) = -0.118
Mod (2, 6) = 0 – (2*3)/(2*17) = -0.176
Mod (2, 7) = 0 – (2*3)/(2*17) = -0.176
Mod (3, 4) = 0 – (2*3)/(2*17) = -0.176
Mod (3, 5) = 0 – (2*3)/(2*17) = -0.176
Mod (3, 6) = 0 – (3*3)/(2*17) = -0.265
Mod (3, 7) = 1 – (3*3)/(2*17) = 0.735
Mod (4, 5) = 1 – (2*2)/(2*17) = 0.882
Mod (4, 6) = 1 – (2*3)/(2*17) = 0.824
Mod (4, 7) = 0 – (2*3)/(2*17) = -0.176
Mod (5, 6) = 1 – (2*3)/(2*17) = 0.824
Mod (5, 7) = 0 – (2*3)/(2*17) = -0.176
Mod (6, 7) = 1 – (3*3)/(2*17) = 0.735

GN Algorithm: Example 1 (It # 2)

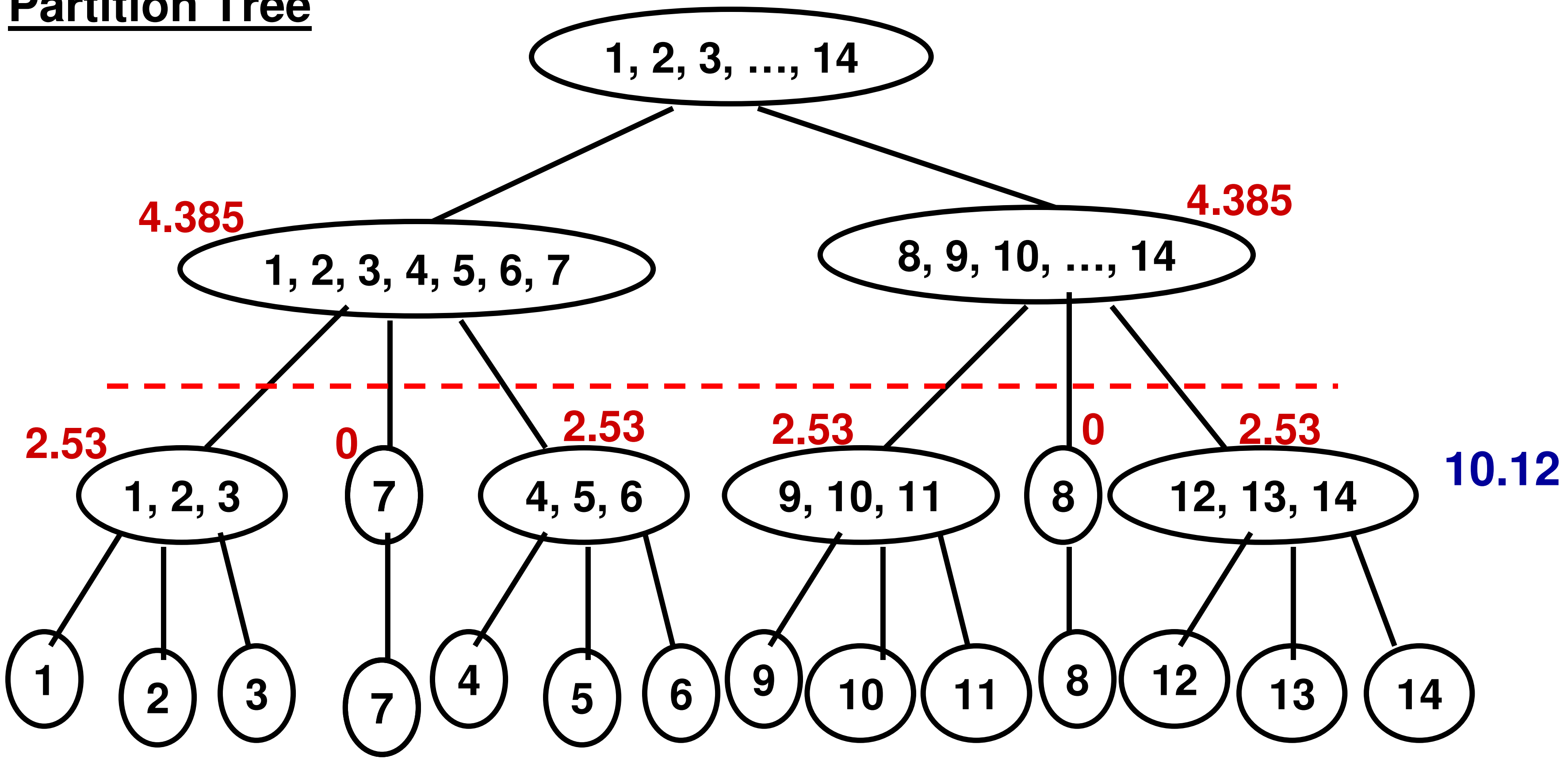


Modularity (1, 2, 3)
Mod (1, 2) = 1 – (2*2)/(2*17) = 0.882
Mod (1, 3) = 1 – (2*3)/(2*17) = 0.824
Mod (2, 3) = 1 – (2*3)/(2*17) = 0.824
Modularity (1, 2, 3) = 2.53

Similarly,
Modularity (4, 5, 6) = 2.53
Modularity (9, 10, 11) = 2.53
Modularity (12, 13, 14) = 2.53

Total Modularity Score = 10.12

Partition Tree



Optimal Partition

(1, 2, 3)

(7)

(4, 5, 6)

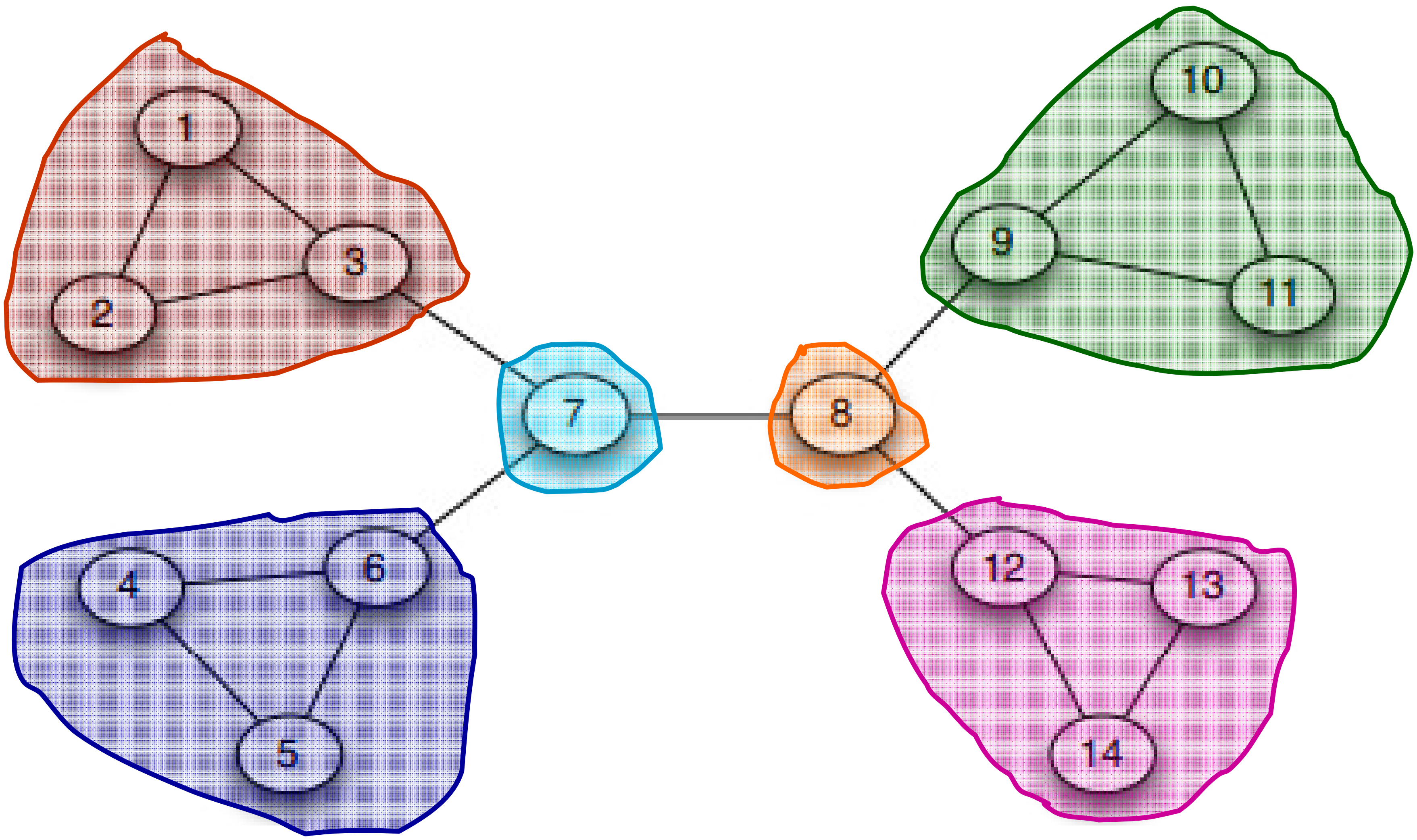
(9, 10, 11)

(8)

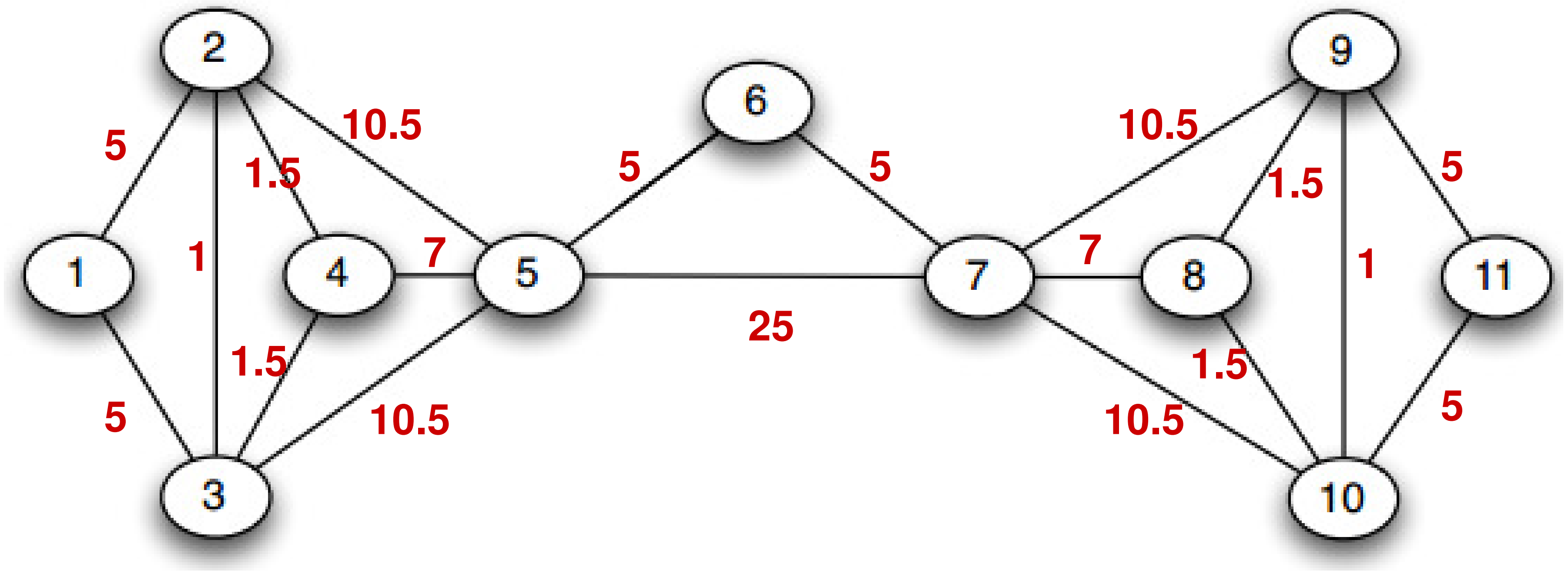
(12, 13, 14)

GN Algorithm Example 1

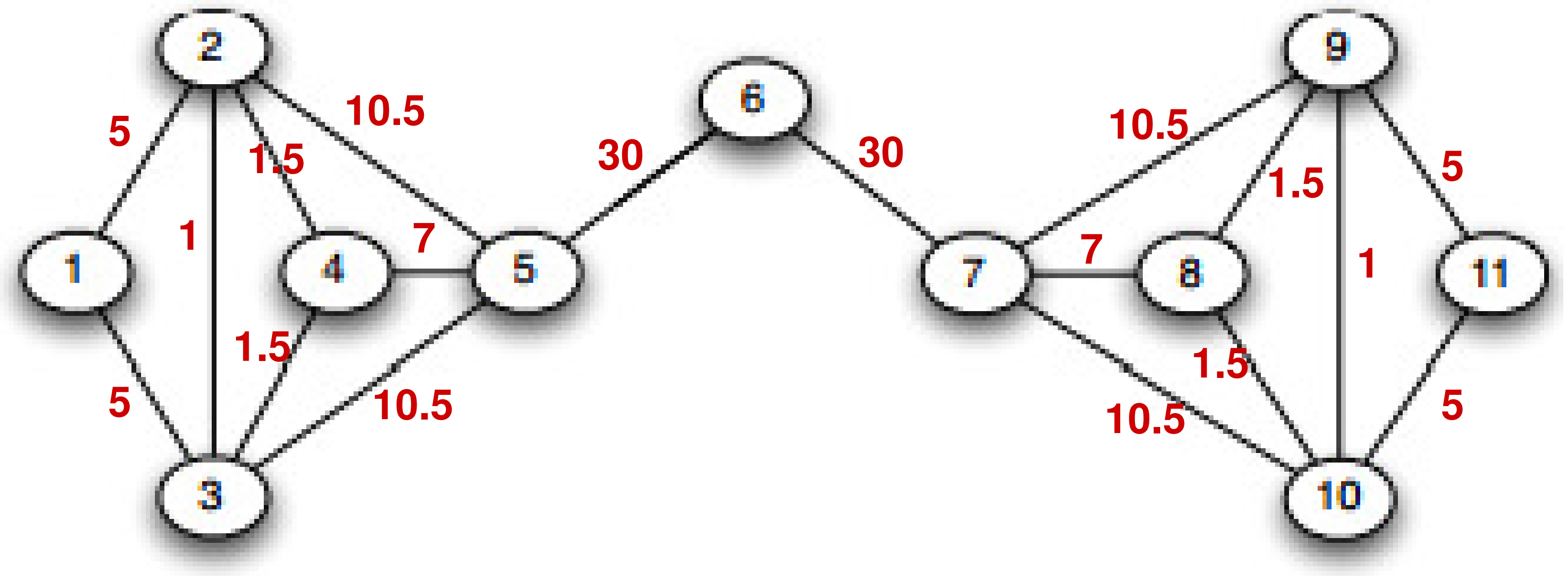
Final Partitioning into Communities



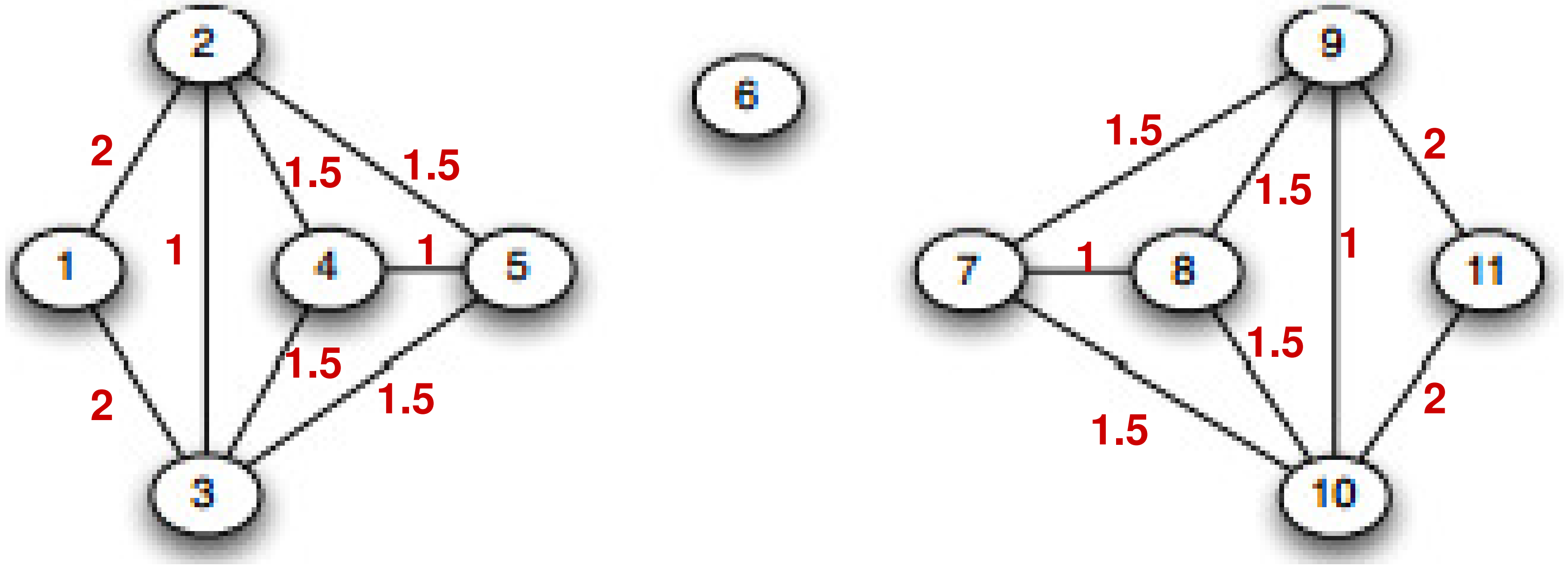
GN Algorithm: Example 2



GN Algorithm: Example 2 (It # 1)



GN Algorithm: Example 2 (It # 2)



<u>Modularity(1, 2, 3, 4, 5)</u>

Mod (1, 2) = 1 – (2*4)/(2*19) = 0.789
Mod (1, 3) = 1 – (2*4)/(2*19) = 0.789
Mod (1, 4) = 0 – (2*3)/(2*19) = -0.158
Mod (1, 5) = 0 – (2*5)/(2*19) = -0.263
Mod (2, 3) = 1 – (4*4)/(2*19) = 0.579
Mod (2, 4) = 1 – (4*3)/(2*19) = 0.684
Mod (2, 5) = 1 – (4*5)/(2*19) = 0.474
Mod (3, 4) = 1 – (3*4)/(2*19) = 0.684
Mod (3, 5) = 1 – (4*5)/(2*19) = 0.474
Mod (4, 5) = 1 – (3*5)/(2*19) = 0.605

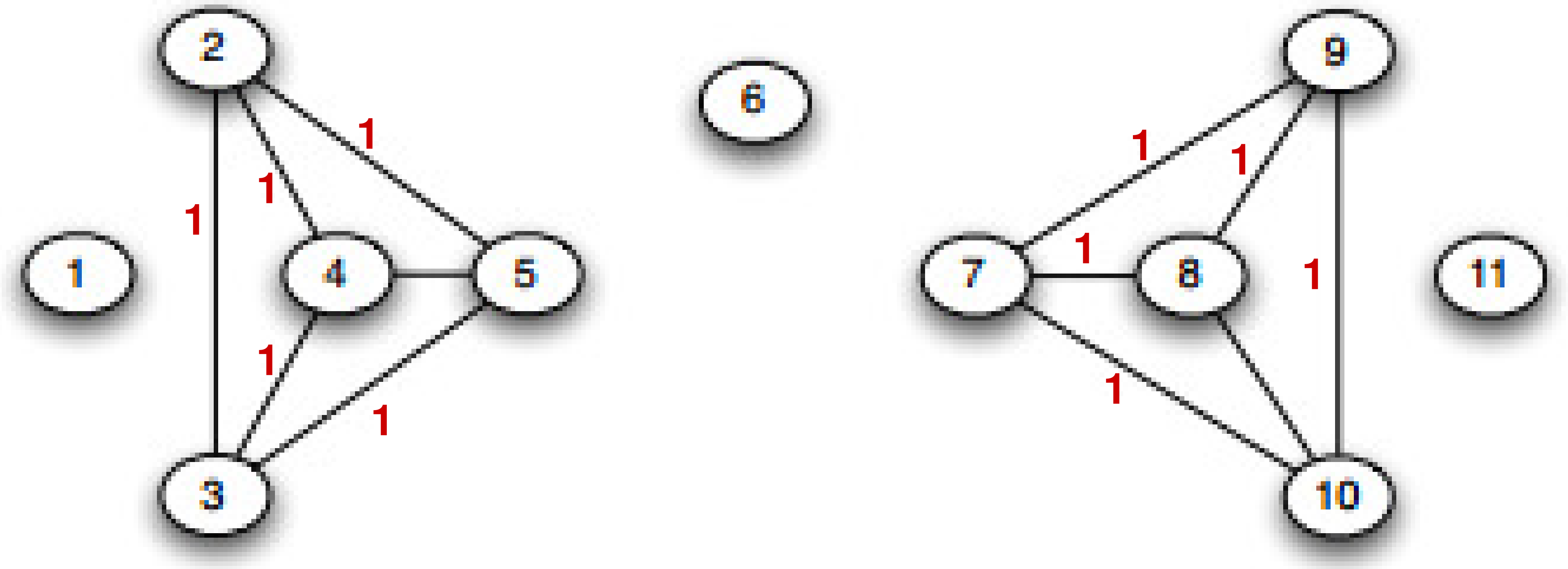
Modularity (1, 2, 3, 4, 5) = 4.657

Modularity (7, 8, 9, 10, 11) = 4.657

Modularity (6) = 0

Total Modularity Score = 9.314

GN Algorithm: Example 2 (It # 3)



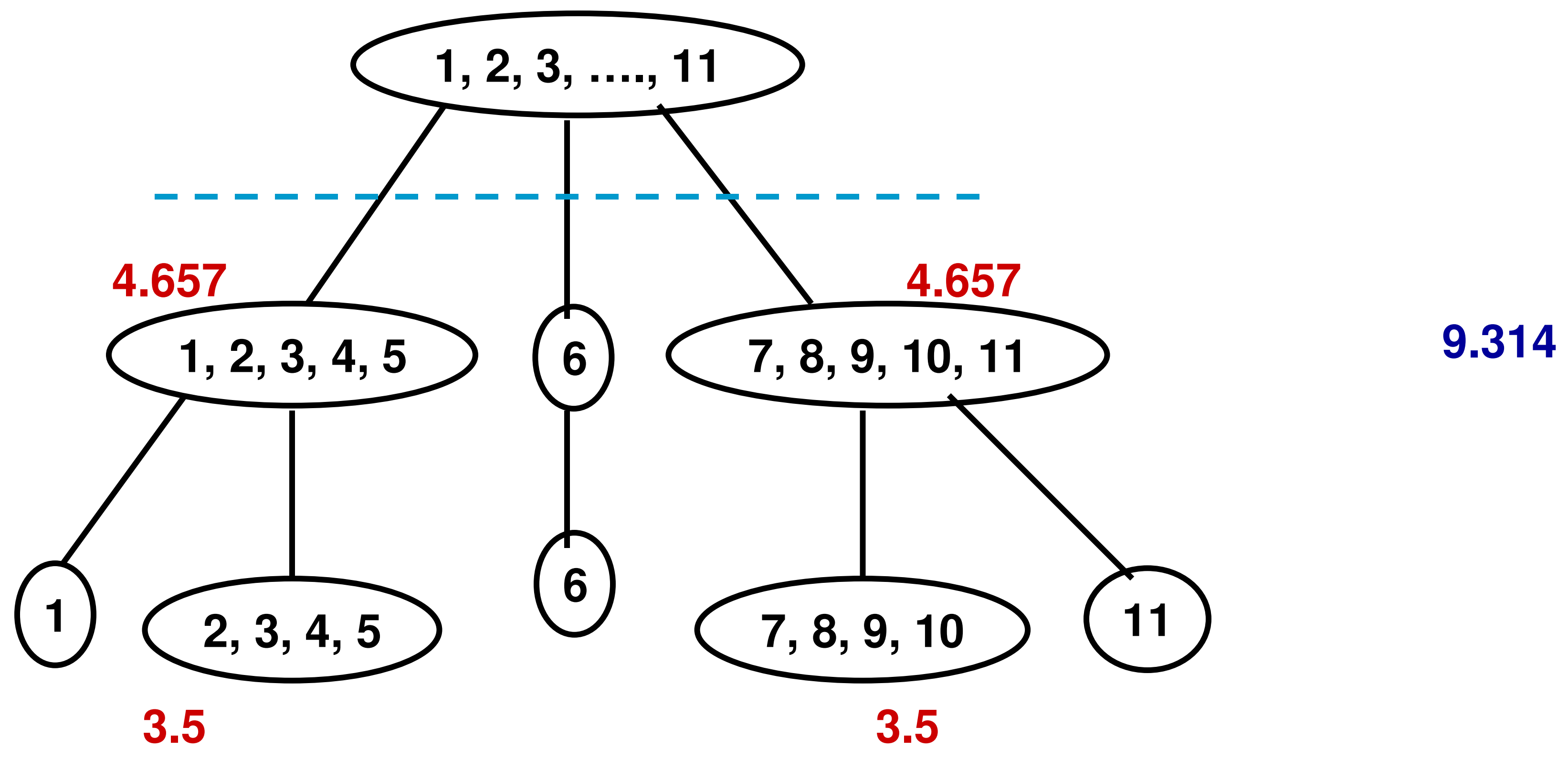
<u>Modularity(2, 3, 4, 5)</u>

Mod (2, 3) = 1 – (4*4)/(2*19) = 0.579
Mod (2, 4) = 1 – (4*3)/(2*19) = 0.684
Mod (2, 5) = 1 – (4*5)/(2*19) = 0.474
Mod (3, 4) = 1 – (3*4)/(2*19) = 0.684
Mod (3, 5) = 1 – (4*5)/(2*19) = 0.474
Mod (4, 5) = 1 – (3*5)/(2*19) = 0.605

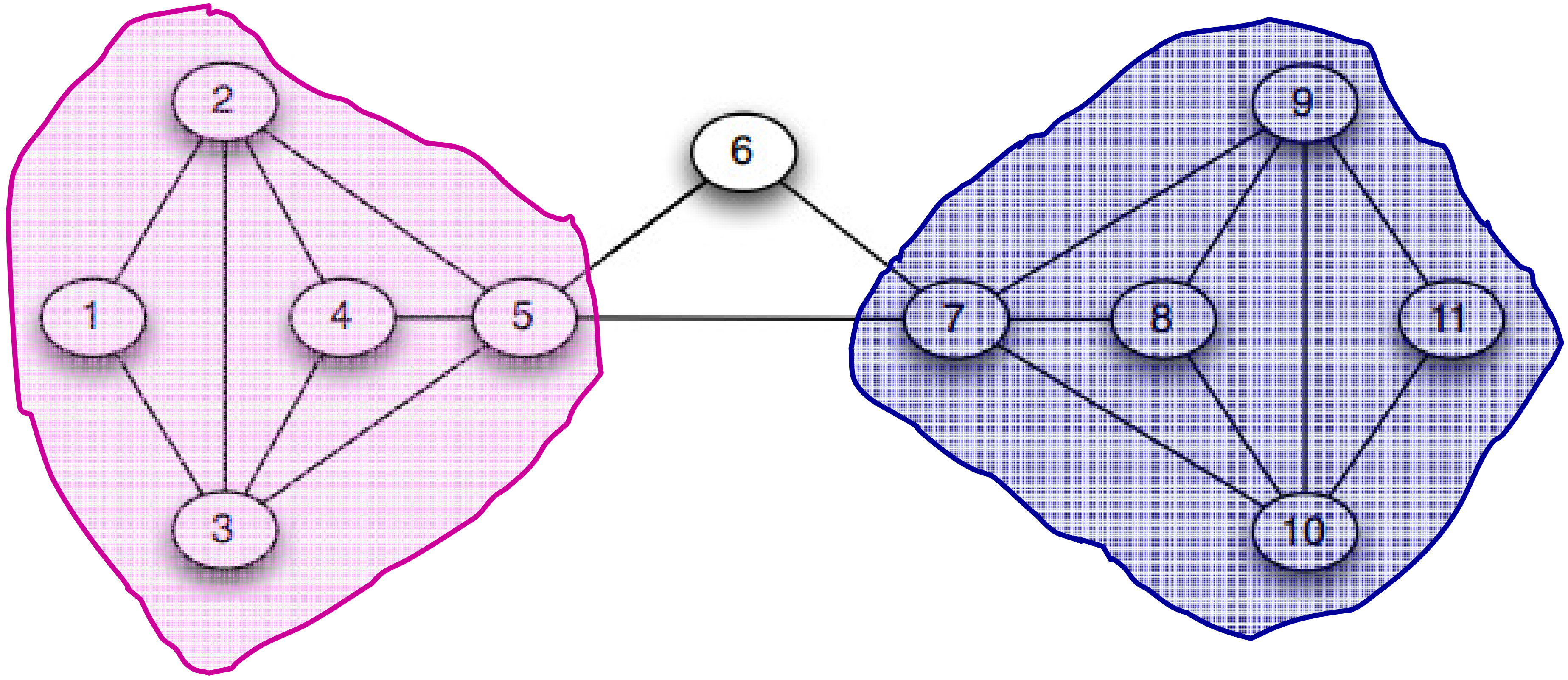
Modularity (2, 3, 4, 5) = 3.5
Modularity (7, 8, 9, 10) = 3.5

Total Modularity Score = 7.0

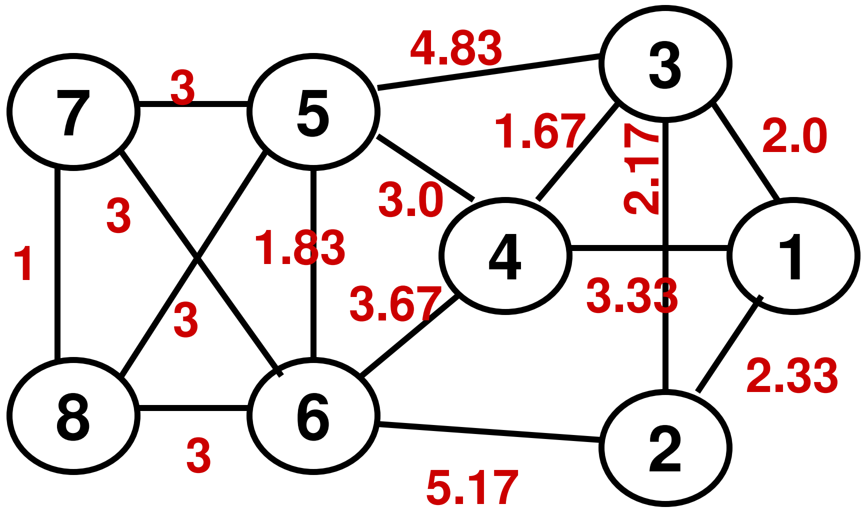
GN Algorithm: Example 2

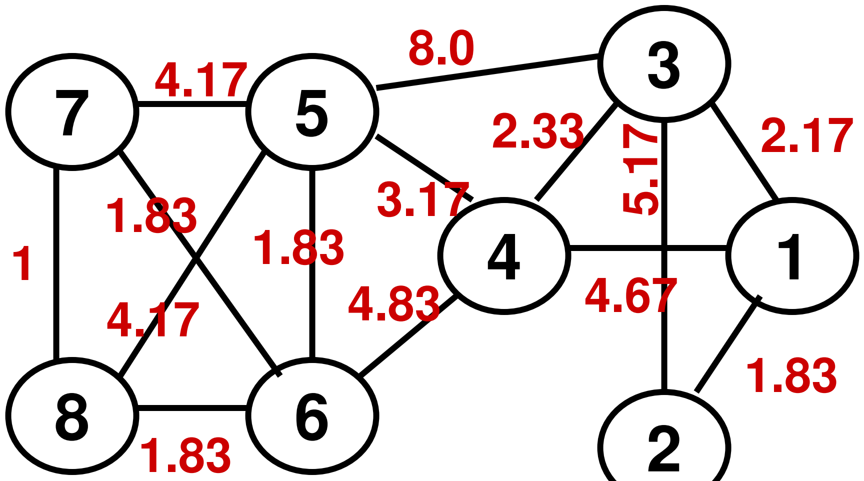


Final Partitioning

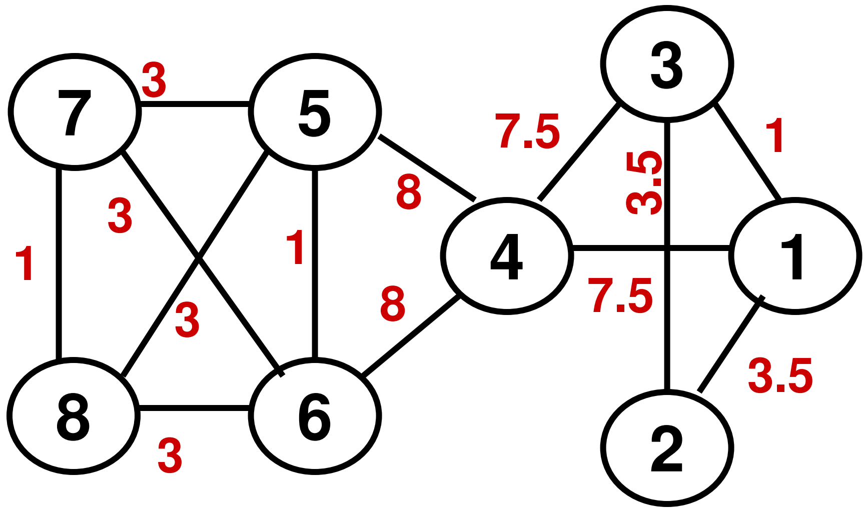


GN Algorithm: Example 3

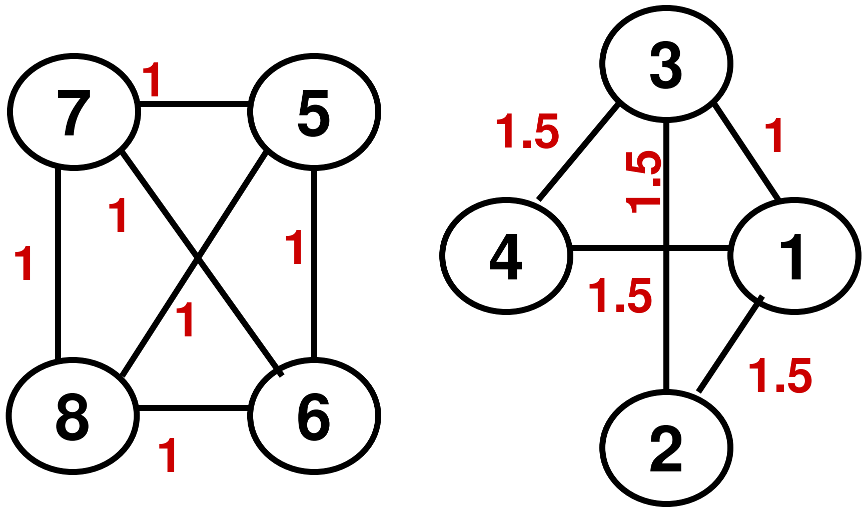




Iteration 1

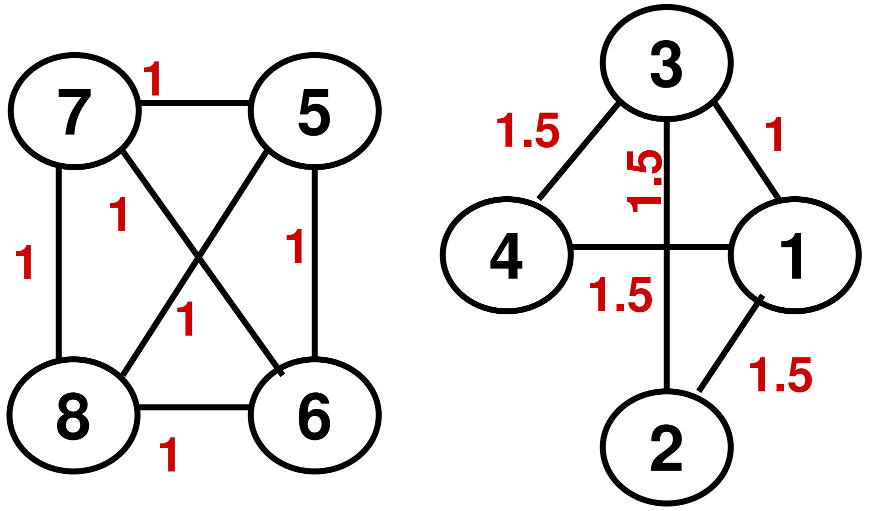


Iteration 2



Iteration 3

GN Algorithm: Example 3 (2)



Iteration 3: Modularity Analysis

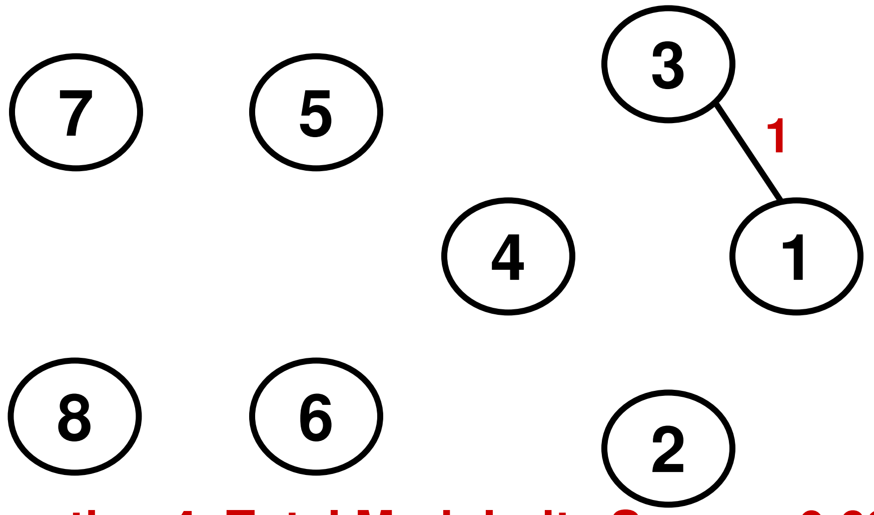
Modularity(1, 2, 3, 4)
Mod(1,2) = 1 – (3*3)/(2*15) = 0.70
Mod(1,3) = 1 – (3*4)/(2*15) = 0.60
Mod(1,4) = 1 – (3*4)/(2*15) = 0.60
Mod(2,3) = 1 – (3*4)/(2*15) = 0.60
Mod(2,4) = 1 – (3*4)/(2*15) = 0.60
Mod(3,4) = 1 – (4*4)/(2*15) = 0.47

Modularity (1, 2, 3, 4) = 3.57

Total Modularity Score = 6.44

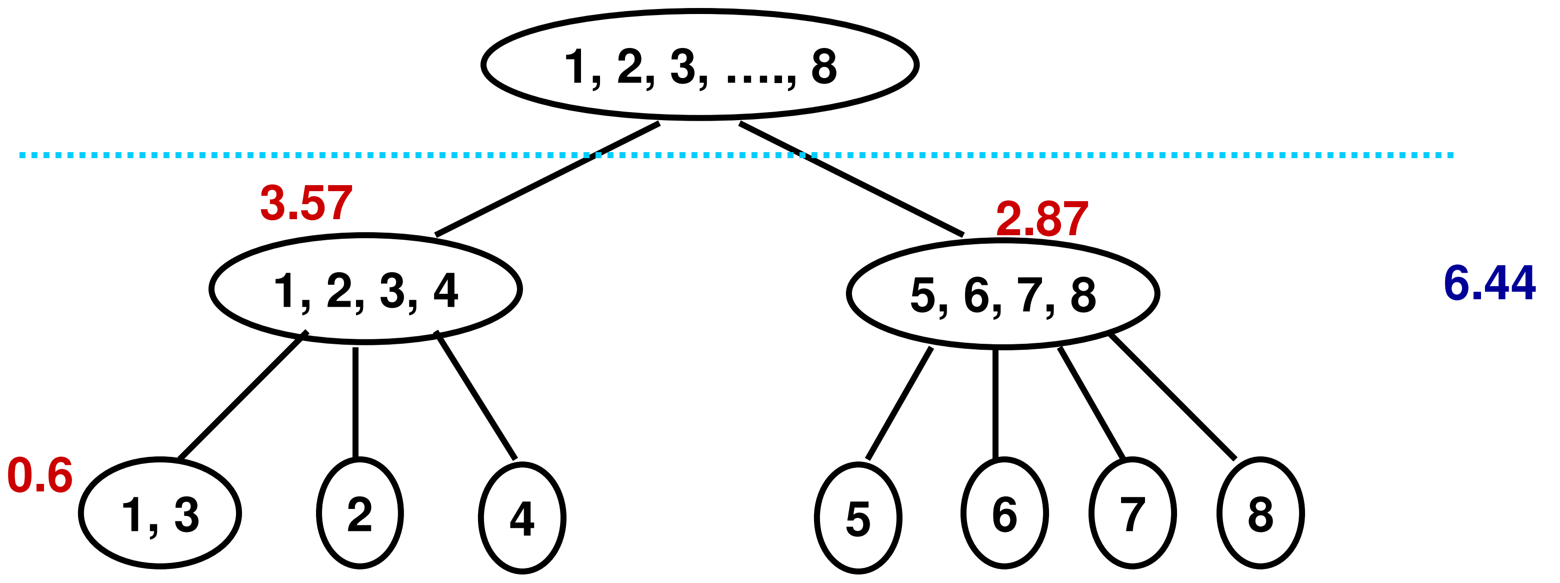
Modularity(5,6,7,8)
Mod(5,6) = 1 – (5*5)/(2*15) = 0.17
Mod(5,7) = 1 – (3*5)/(2*15) = 0.50
Mod(5,8) = 1 – (3*5)/(2*15) = 0.50
Mod(6,7) = 1 – (3*5)/(2*15) = 0.50
Mod(6,8) = 1 – (3*5)/(2*15) = 0.50
Mod(7,8) = 1 – (3*3)/(2*15) = 0.70

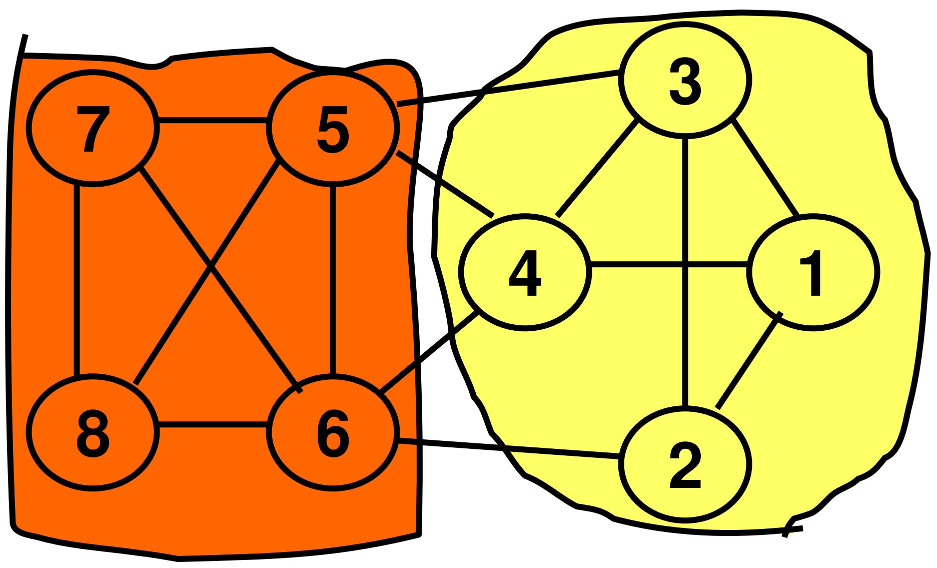
Modularity (5, 6, 7, 8) = 2.87



Iteration 4: Total Modularity Score = 0.60

GN Algorithm: Example 3 (3)

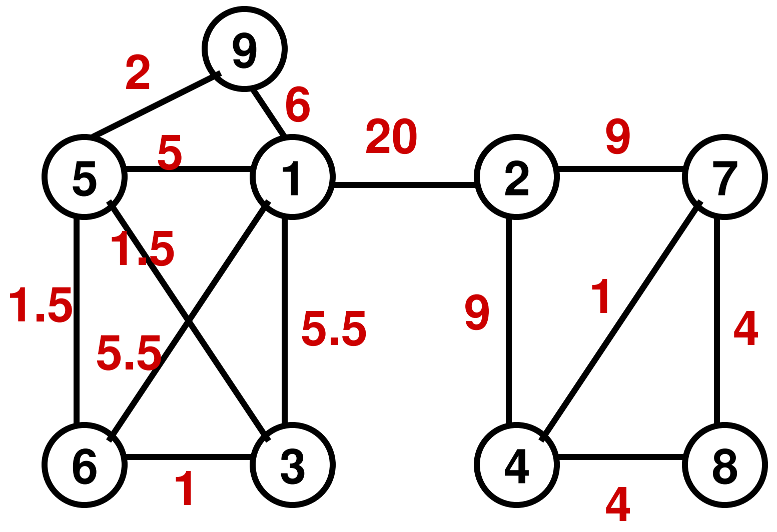


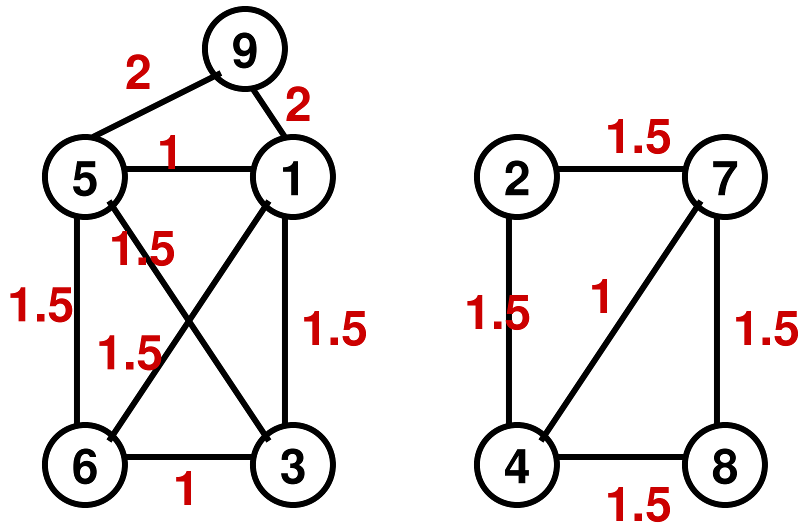


Optimal Partitioning Into Communities

Total Modularity Score 6.44

GN Algorithm: Example 4





Iteration 1

Modularity (1, 3, 5, 6, 9)

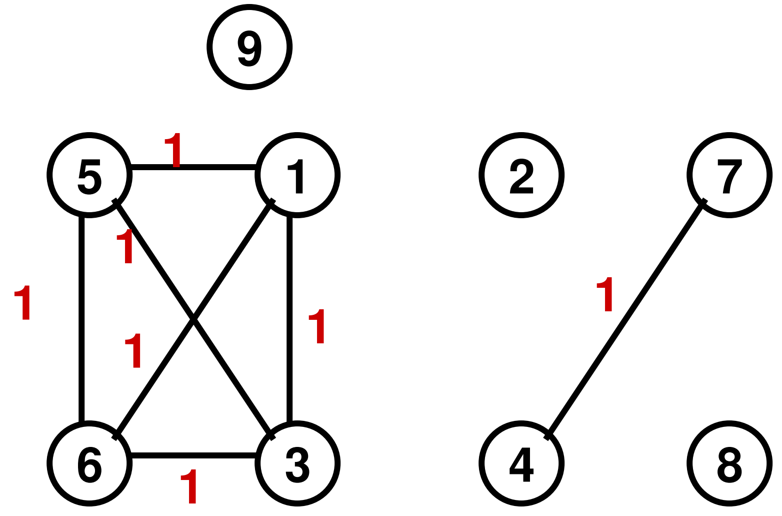
Mod (1, 3) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 5) = 1 – (5*4)/(2*14) = 0.28
Mod (1, 6) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 9) = 1 – (5*2)/(2*14) = 0.64
Mod (3, 5) = 1 – (3*4)/(2*14) = 0.57
Mod (3, 6) = 1 – (3*3)/(2*14) = 0.68
Mod (3, 9) = 0 – (3*2)/(2*14) = - 0.21
Mod (5, 6) = 1 – (4*3)/(2*14) = 0.57
Mod (5, 9) = 1 – (4*2)/(2*14) = 0.71
Mod (6, 9) = 0 – (3*2)/(2*14) = - 0.21
Modularity (1, 3, 5, 6, 9) = 3.95

Modularity (2, 4, 7, 8)

Mod (2, 4) = 1 – (3*3)/(2*14) = 0.68
Mod (2, 7) = 1 – (3*3)/(2*14) = 0.68
Mod (2, 8) = 0 – (3*2)/(2*14) = - 0.21
Mod (4, 7) = 1 – (3*3)/(2*14) = 0.68
Mod (4, 8) = 1 – (3*2)/(2*14) = 0.78
Mod (7, 8) = 1 – (3*2)/(2*14) = 0.78
Modularity (2, 4, 7, 8) = 3.39

Total Modularity Score = 7.34

GN Algorithm: Example 4 (1)



Iteration 2

Modularity (1, 3, 5, 6)

Mod (1, 3) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 5) = 1 – (5*4)/(2*14) = 0.28
Mod (1, 6) = 1 – (5*3)/(2*14) = 0.46
Mod (3, 5) = 1 – (3*4)/(2*14) = 0.57
Mod (3, 6) = 1 – (3*3)/(2*14) = 0.68
Mod (5, 6) = 1 – (4*3)/(2*14) = 0.57

Modularity (1, 3, 5, 6) = 3.02

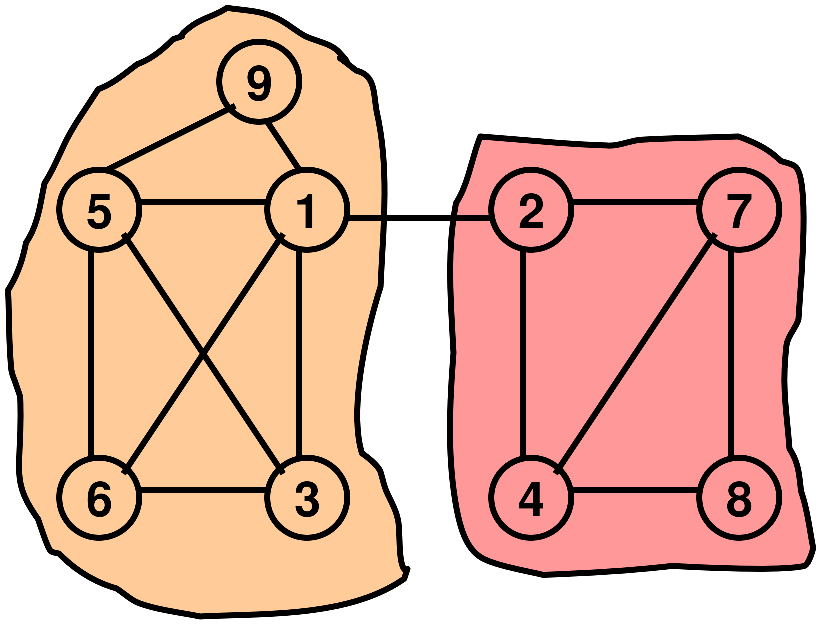
Modularity (4, 7)

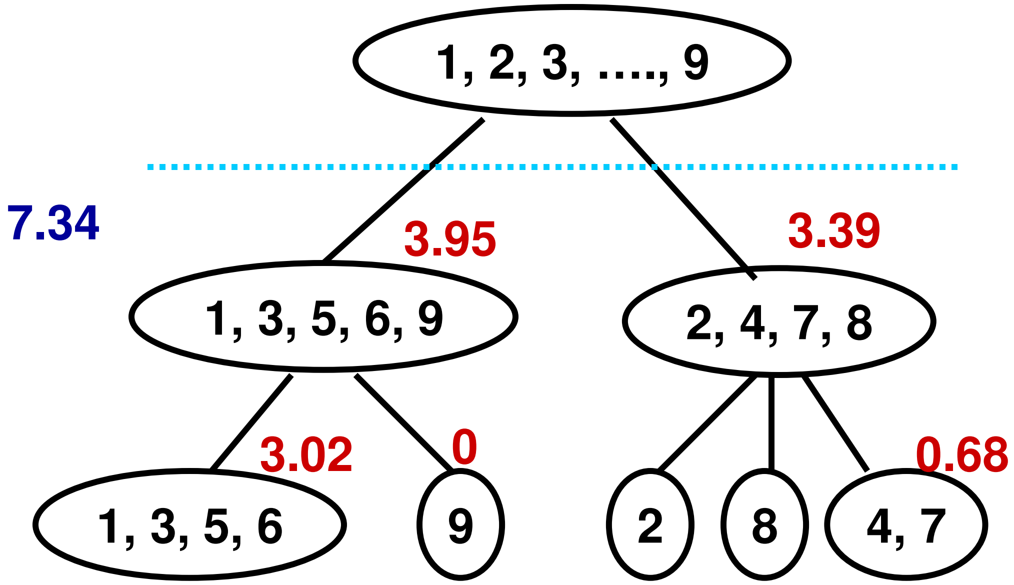
Mod (4, 7) = 1 – (3*3)/(2*14) = 0.68

Modularity (4, 7) = 0.68

Total Modularity Score = 3.70

GN Algorithm: Example 4 (2)

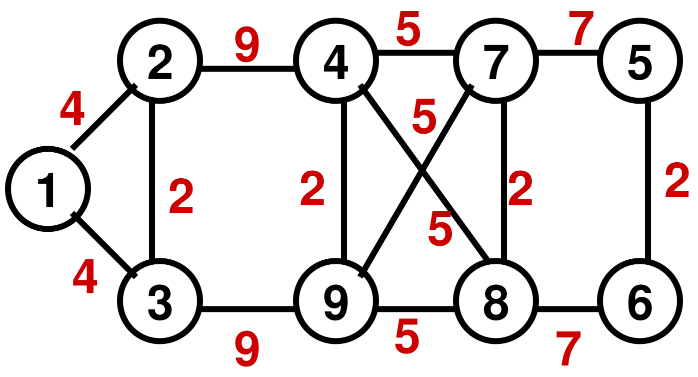




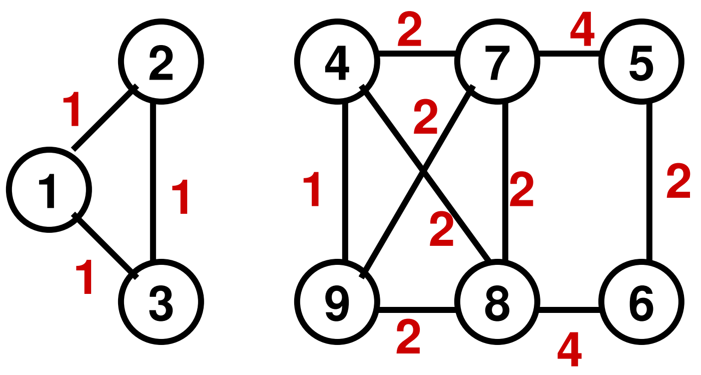
Analysis of: Girvan and Newman Algorithm

- After we find the betweenness of the edges, we remove the edge with the largest betweenness.
- We re-run BFS on each vertex and find the betweenness of every edge and remove the edge with the largest betweenness henceforth.
- We repeat this process until we divide the graph into individual vertices.
- We keep track of the communities that get generated with each edge removal and then decide on the level of partition (to stop the edge removal process) by evaluating the modularity scores of the community scores formed at different levels.
- The Girvan and Newman algorithm, though effective in delineating communities with high modularity scores, is very inefficient as it requires BFS (of time complexity $\Theta(E+V)$) to be run on each vertex for every edge removal.
  - For a graph with E edges and V vertices, the overall time complexity will be $\Theta(EV(E+V))$

GN Algorithm: Example 5



Given Graph with Edge BW Values



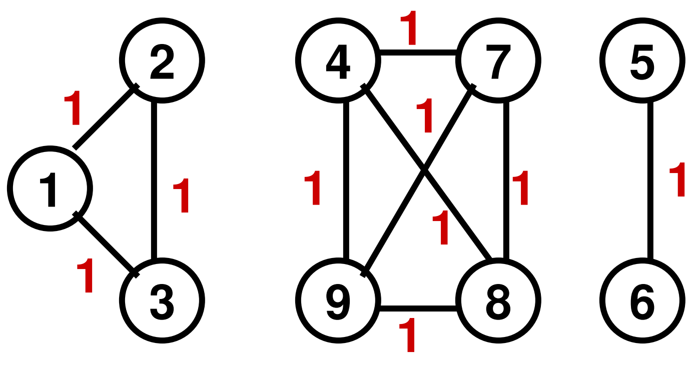
Iteration 1

Modularity (1, 2, 3)
Mod (1, 2) = 0.786
Mod (1, 3) = 0.786
Mod (2, 3) = 0.678
Modularity (1, 2, 3) = 2.25

Total Modularity
= 5.393

Modularity (4, 5, 6, 7, 8, 9)
Mod (4, 5) = - 0.286
Mod (4, 6) = - 0.286
Mod (4, 7) = 0.429
Mod (4, 8) = 0.429
Mod (4, 9) = 0.429
Mod (5, 6) = 0.857
Mod (5, 7) = 0.714
Mod (5, 8) = - 0.286
Mod (5, 9) = - 0.286
Mod (6, 7) = - 0.286
Mod (6, 8) = 0.714
Mod (6, 9) = - 0.286
Mod (7, 8) = 0.429
Mod (7, 9) = 0.429
Mod (8, 9) = 0.429
Modularity (4, 5, 6, 7, 8, 9) = 3.143

GN Algorithm: Example 5 (1)



Iteration 2

Modularity (4, 7, 8, 9)

Mod (4, 7) = 0.429	Mod (7, 8) = 0.429
Mod (4, 8) = 0.429	Mod (7, 9) = 0.429
Mod (4, 9) = 0.429	Mod (8, 9) = 0.429

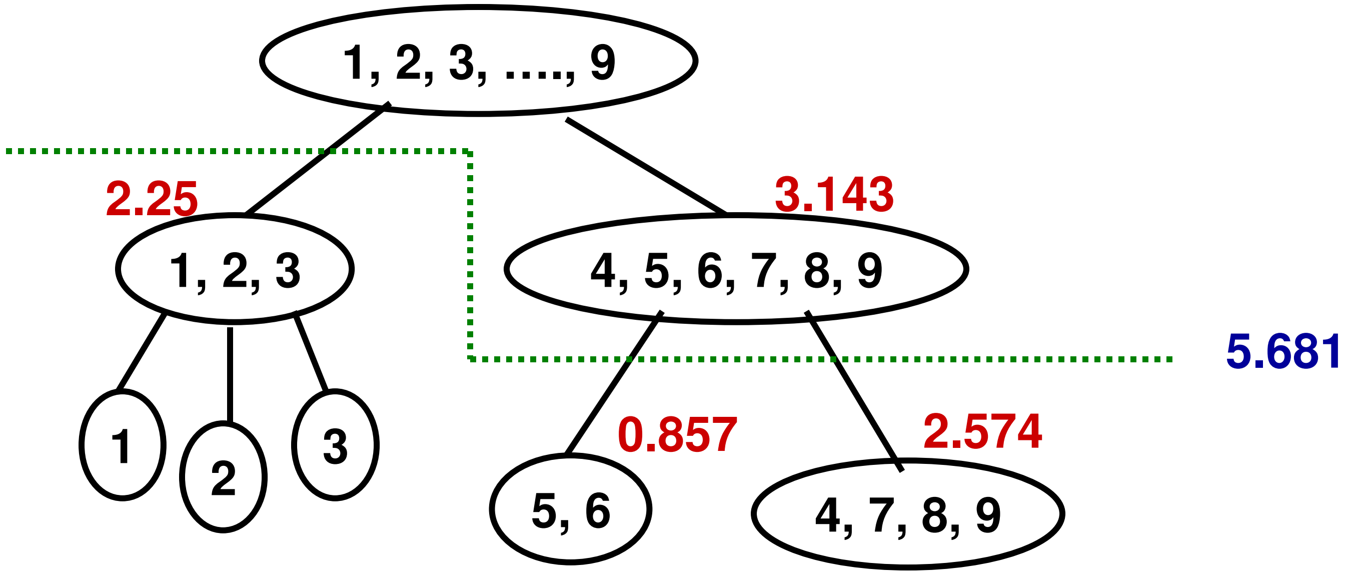
Modularity (4, 7, 8, 9) = 2.574

Modularity { (5, 6); (4, 7, 8, 9) } = 3.431

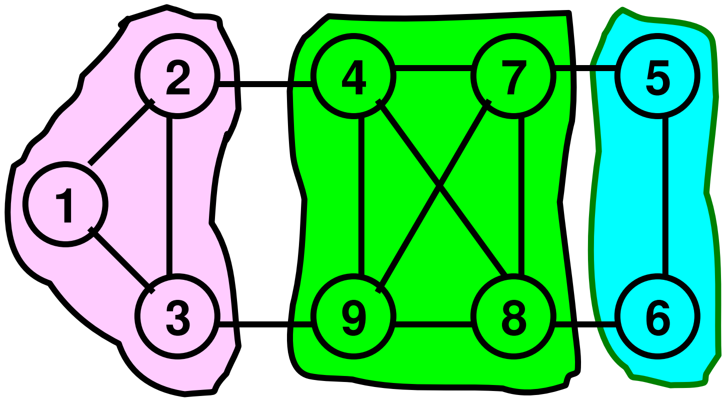
We do not need to partition (1, 2, 3) further as the Modularity of the partitioned Communities (1), (2), (3) will be just 0.

Modularity (5, 6) = 0.857

GN Algorithm: Example 5 (2)



Note: Edges with a larger BW are considered to facilitate Communication between two or more communities. Hence, removing such edges could lead to the identification of the Communities in a network.

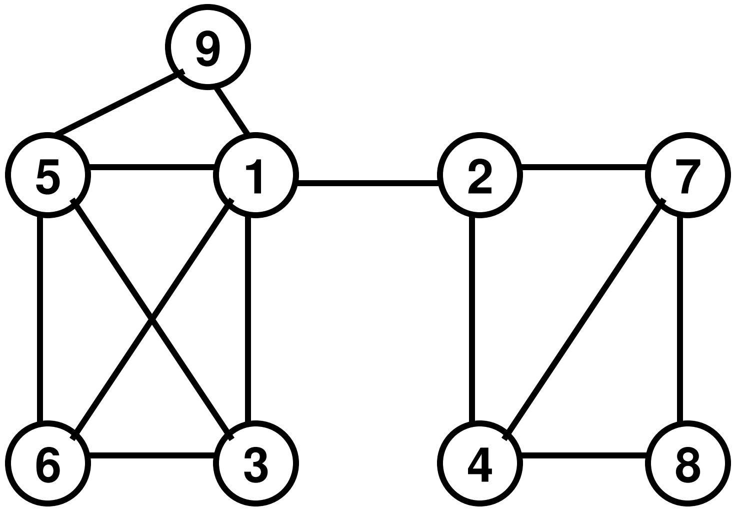


Neighborhood Overlap based Approach

Neighborhood Overlap (NOVER)

$$\text{Neighborhood Overlap} = \frac{\text{number of nodes who are neighbors of } \textit{both } A \text{ and } B}{\text{number of nodes who are neighbors of } \textit{at least one of } A \text{ or } B}$$

Note that one should not count neither A nor B as part of the neighbors in the denominator, and each node should be counted only once.



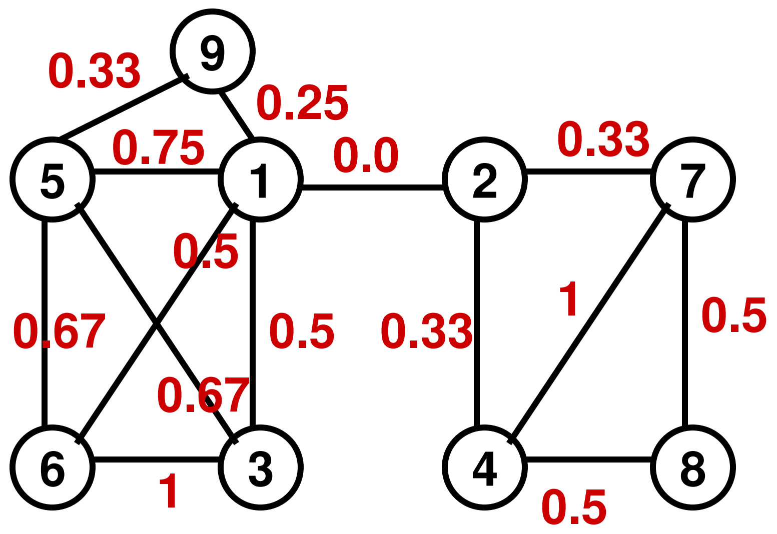
1 → {2, 3, 5, 6, 9}	7 → {2, 8}
2 → {1, 4, 7}	8 → {4, 7}
3 → {1, 5, 6}	9 → {1, 5}
4 → {2, 7, 8}	
5 → {1, 3, 6, 9}	
6 → {1, 3, 5}	

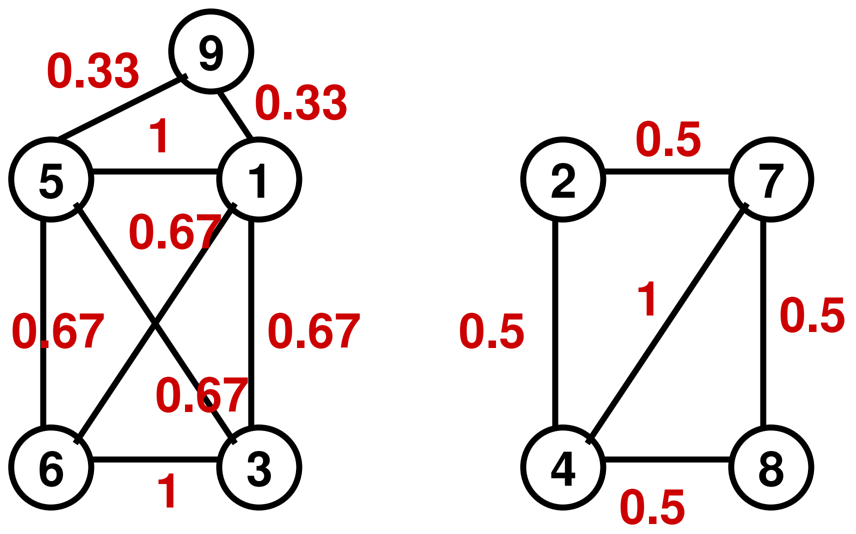
Edge	Union of Neighb.	Intersec.	NOVER
1 – 2	{3, 5, 6, 9, 4, 7}	{}	0/6 = 0.0
1 – 3	{2, 5, 6, 9}	{5, 6}	2/4 = 0.5
1 – 5	{2, 3, 6, 9}	{3, 6, 9}	3/4 = 0.75
1 – 6	{2, 3, 5, 9}	{3, 5}	2/4 = 0.5
1 – 9	{2, 3, 5, 6}	{5}	1/4 = 0.25
2 – 4	{1, 7, 8}	{7}	1/3 = 0.33
2 – 7	{1, 4, 8}	{4}	1/3 = 0.33
3 – 5	{1, 6, 9}	{1, 6}	2/3 = 0.67
3 – 6	{1, 5}	{1, 5}	1/1 = 1.0
4 – 7	{2, 8}	{2, 8}	2/2 = 1.0
4 – 8	{2, 7}	{7}	1/2 = 0.5
5 – 6	{1, 3, 9}	{1, 3}	2/3 = 0.67
5 – 9	{1, 3, 6}	{1}	1/3 = 0.33
7 – 8	{2, 4}	{4}	1/2 = 0.5

NOVER-based GN Algorithm

- At the beginning of each iteration, we compute the NOVER scores of the edges in the graph and remove the edge(s) with the smallest NOVER score(s).
  - If more than one edge has the smallest NOVER score, remove all such competing edges at the same time.
  - If the graph gets disconnected to two or more communities (components), we compute the total modularity score of the resulting communities
- Repeat the iterations until there are no more edges
- The partition (set of communities) with the total modularity score is the optimal partition.

NOVER-based GN Algorithm: Ex. 1





Iteration 1

Modularity (1, 3, 5, 6, 9)

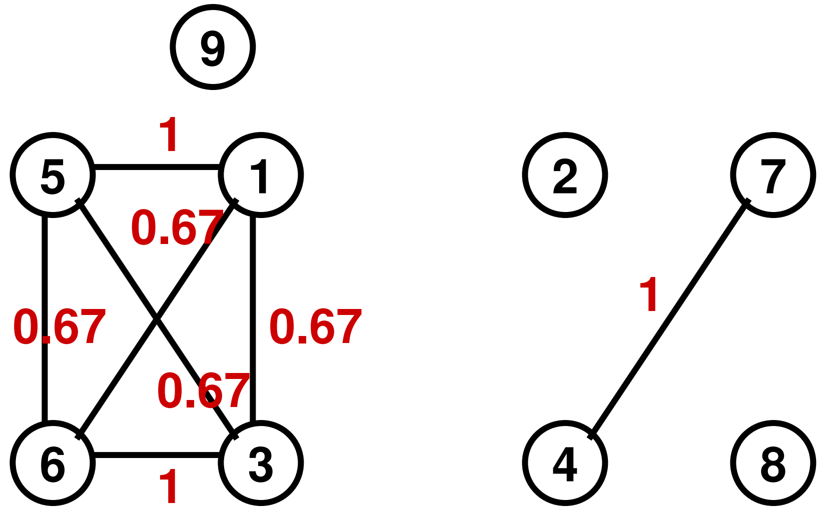
Mod (1, 3) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 5) = 1 – (5*4)/(2*14) = 0.28
Mod (1, 6) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 9) = 1 – (5*2)/(2*14) = 0.64
Mod (3, 5) = 1 – (3*4)/(2*14) = 0.57
Mod (3, 6) = 1 – (3*3)/(2*14) = 0.68
Mod (3, 9) = 0 – (3*2)/(2*14) = - 0.21
Mod (5, 6) = 1 – (4*3)/(2*14) = 0.57
Mod (5, 9) = 1 – (4*2)/(2*14) = 0.71
Mod (6, 9) = 0 – (3*2)/(2*14) = - 0.21
Modularity (1, 3, 5, 6, 9) = 3.95

Modularity (2, 4, 7, 8)

Mod (2, 4) = 1 – (3*3)/(2*14) = 0.68
Mod (2, 7) = 1 – (3*3)/(2*14) = 0.68
Mod (2, 8) = 0 – (3*2)/(2*14) = - 0.21
Mod (4, 7) = 1 – (3*3)/(2*14) = 0.68
Mod (4, 8) = 1 – (3*2)/(2*14) = 0.78
Mod (7, 8) = 1 – (3*2)/(2*14) = 0.78
Modularity (2, 4, 7, 8) = 3.39

Total Modularity Score = 7.34

NOVER-based GN Algorithm: Ex. 1(2)



Iteration 2

Modularity (1, 3, 5, 6)

Mod (1, 3) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 5) = 1 – (5*4)/(2*14) = 0.28
Mod (1, 6) = 1 – (5*3)/(2*14) = 0.46
Mod (3, 5) = 1 – (3*4)/(2*14) = 0.57
Mod (3, 6) = 1 – (3*3)/(2*14) = 0.68
Mod (5, 6) = 1 – (4*3)/(2*14) = 0.57

Modularity (1, 3, 5, 6) = 3.02

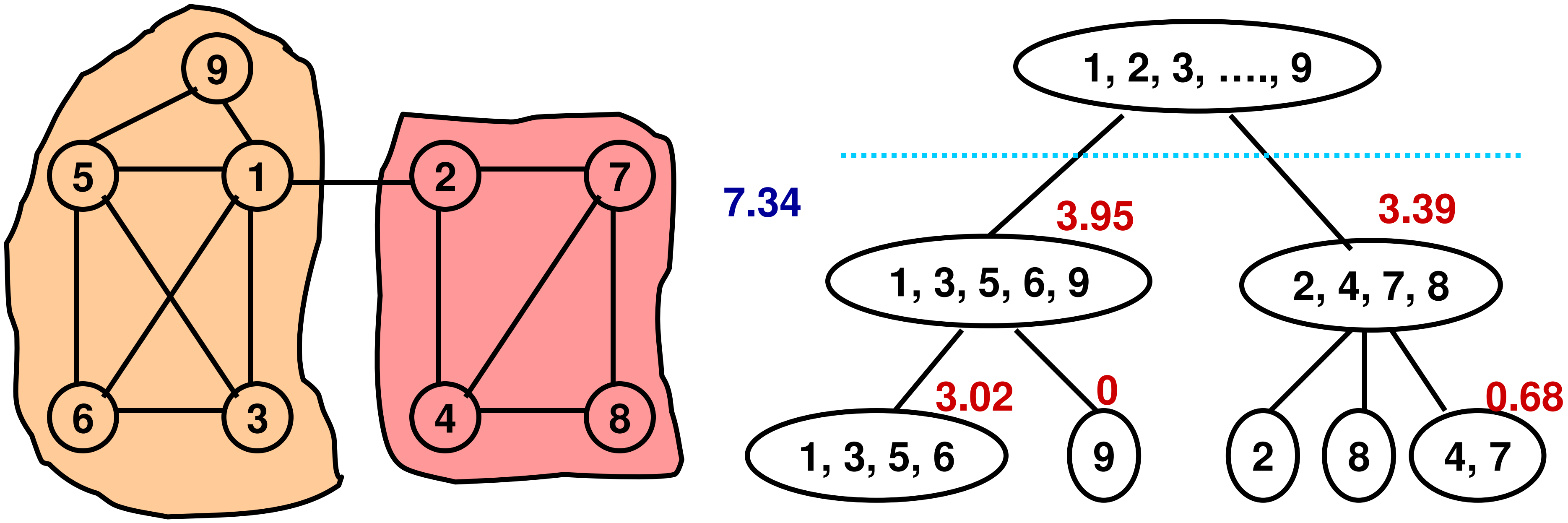
Modularity (4, 7)

Mod (4, 7) = 1 – (3*3)/(2*14) = 0.68

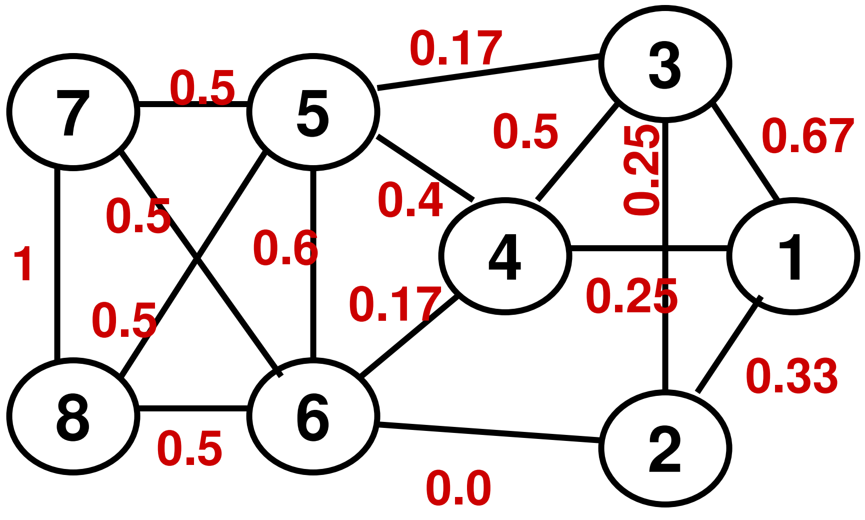
Modularity (4, 7) = 0.68

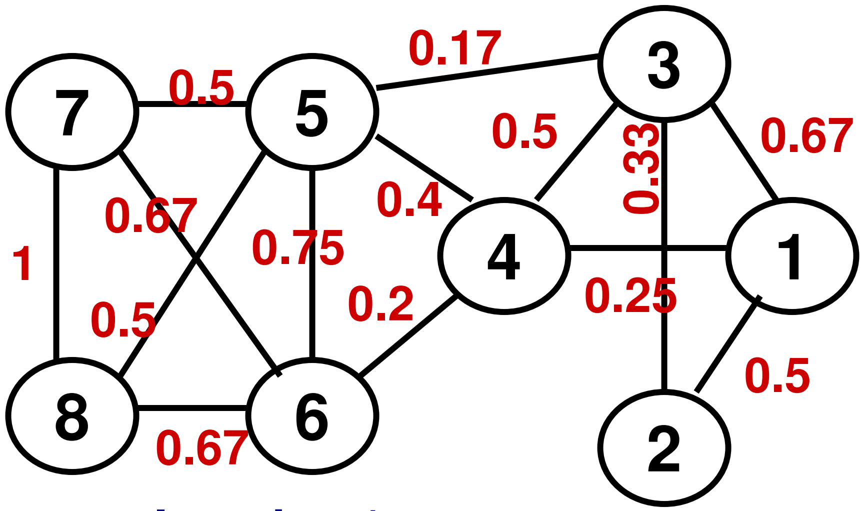
Total Modularity Score = 3.70

NOVER-based GN Alg. Ex-1(2)

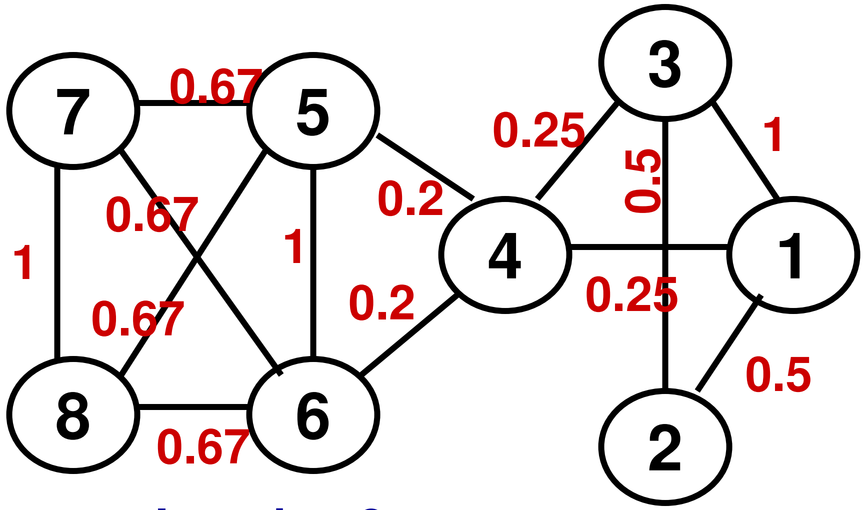


NOVER-based GN Alg. Ex-2

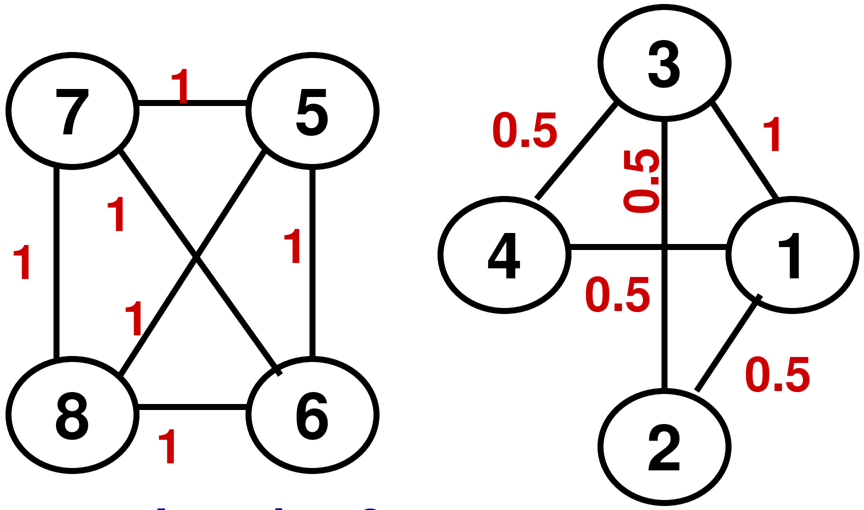




Iteration 1

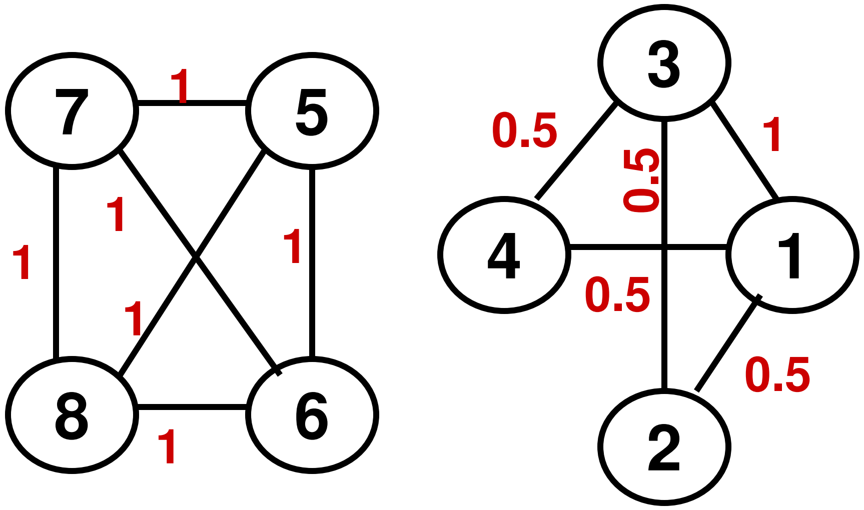


Iteration 2



Iteration 3

NOVER-based GN Alg. Ex-2(1)



Iteration 3: Modularity Analysis

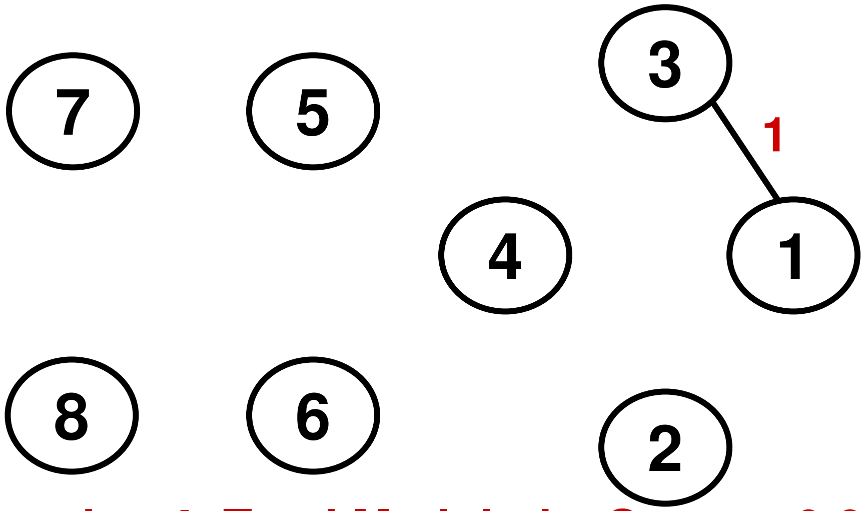
Modularity(1, 2, 3, 4)
Mod(1,2) = 1 – (3*3)/(2*15) = 0.70
Mod(1,3) = 1 – (3*4)/(2*15) = 0.60
Mod(1,4) = 1 – (3*4)/(2*15) = 0.60
Mod(2,3) = 1 – (3*4)/(2*15) = 0.60
Mod(2,4) = 1 – (3*4)/(2*15) = 0.60
Mod(3,4) = 1 – (4*4)/(2*15) = 0.47

Modularity (1, 2, 3, 4) = 3.57

Total Modularity Score = 6.44

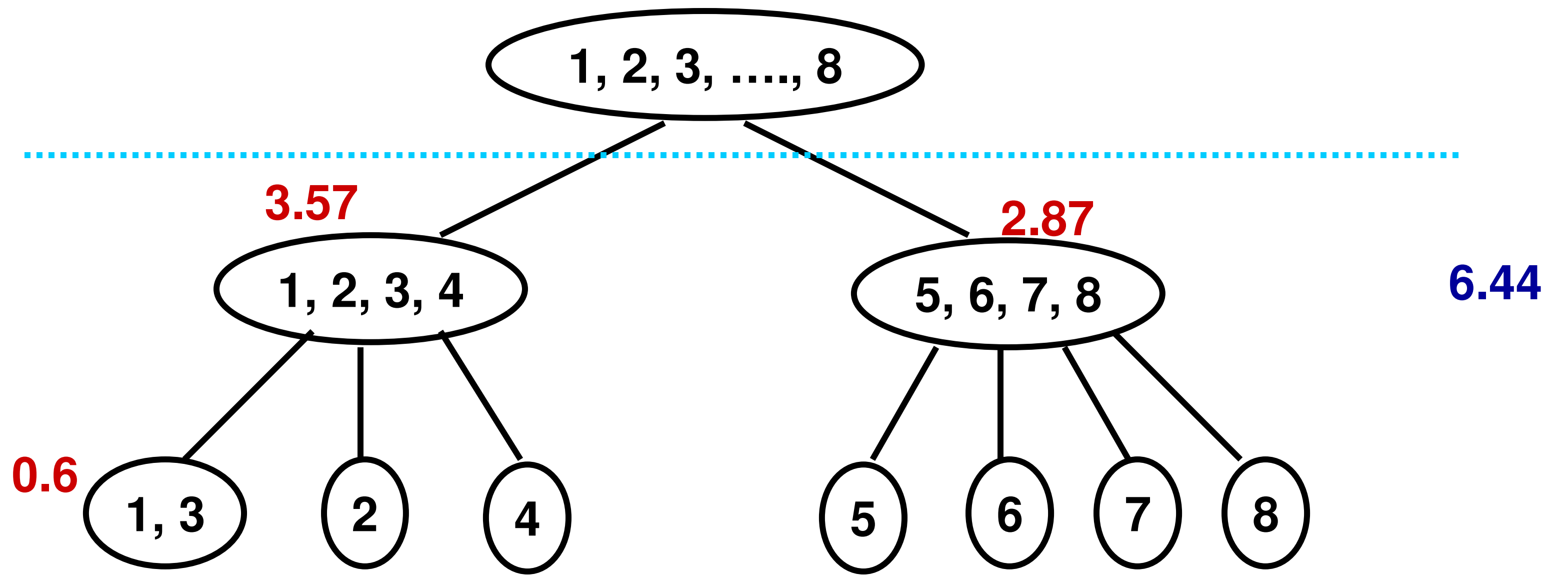
Modularity(5,6,7,8)
Mod(5,6) = 1 – (5*5)/(2*15) = 0.17
Mod(5,7) = 1 – (3*5)/(2*15) = 0.50
Mod(5,8) = 1 – (3*5)/(2*15) = 0.50
Mod(6,7) = 1 – (3*5)/(2*15) = 0.50
Mod(6,8) = 1 – (3*5)/(2*15) = 0.50
Mod(7,8) = 1 – (3*3)/(2*15) = 0.70

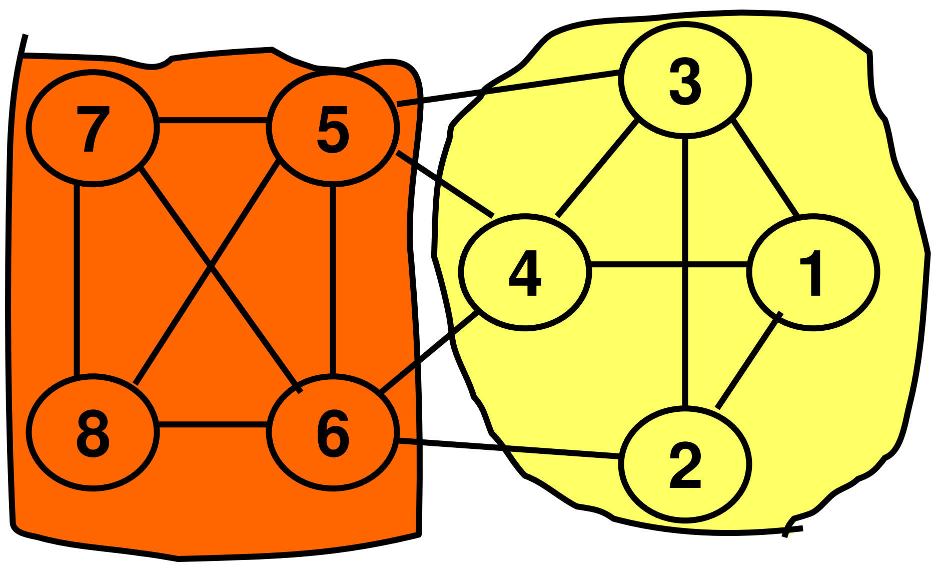
Modularity (5, 6, 7, 8) = 2.87



Iteration 4: Total Modularity Score = 0.60

NOVER-based GN Alg. Ex-2(1)

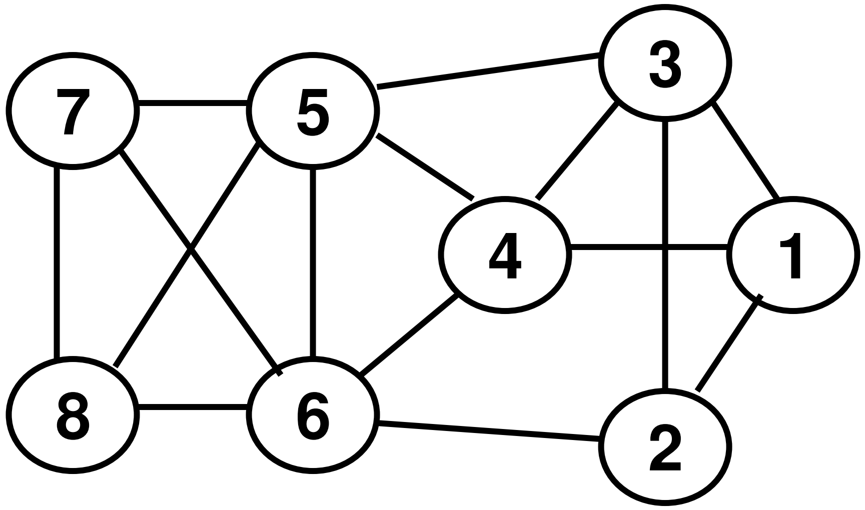




Optimal Partitioning Into Communities

Total Modularity Score 6.44

Rank-based Correlation: BW vs. NOVER



Rank-based Corr. Coeff

$$1 - \frac{6\sum_{i=1}^{n} d_i^{\,2}}{n(n^2 - 1)} = \underline{0.79}$$

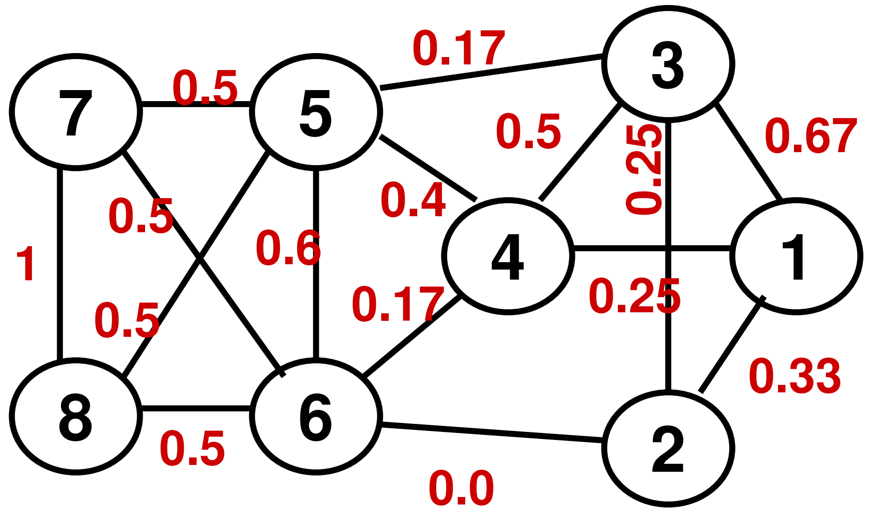
Sum Sq. Diff = 114.91
n = 15

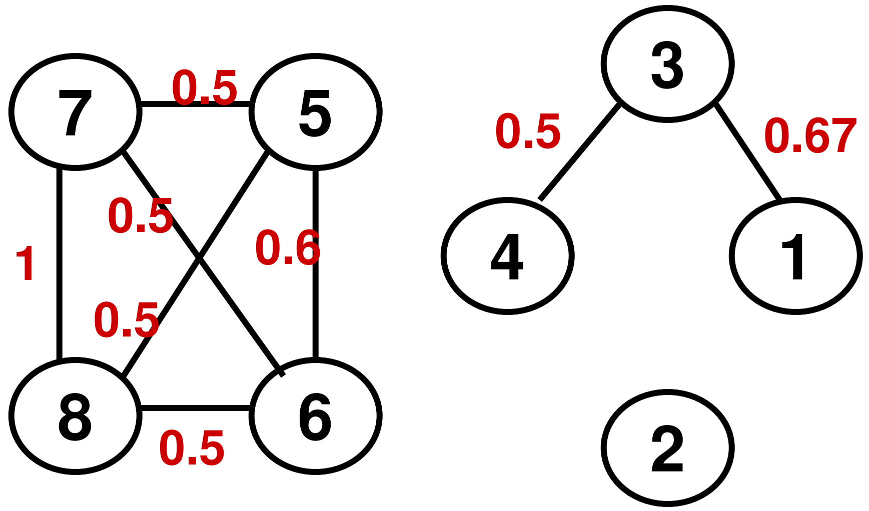
Edge	Edge ID	BW	BW-rank-ten	BW-rank-fin	NOVER	NOVER-rank-ten	NOVER-rank-fin	diff	sq. diff
1, 2	1	2.33	10	10	0.33	6	6	4	16
1, 3	2	2	12	12	0.67	14	14	-2	4
1, 4	3	3.33	4	4	0.25	4	4.4	-0.4	0.16
2, 3	4	2.17	11	11	0.25	5	4.5	6.5	42.25
2, 6	5	5.17	1	1	0	1	1	0	0
3, 4	6	1.67	14	14	0.5	8	10	4	16
3, 5	7	4.83	2	2	0.17	2	2.5	-0.5	0.25
4, 5	8	3	5	7	0.4	7	7	0	0
4, 6	9	3.67	3	3	0.17	3	2.5	0.5	0.25
5, 6	10	1.83	13	13	0.6	13	13	0	0
5, 7	11	3	6	7	0.5	9	10	-3	9
5, 8	12	3	7	7	0.5	10	10	-3	9
6, 7	13	3	8	7	0.5	11	10	-3	9
6, 8	14	3	9	7	0.5	12	10	-3	9
7, 8	15	1	15	15	1	15	15	0	0

Weak Ties/Strong Ties

- An edge could be classified either as a weak tie or strong tie based on its NOVER score.
  - An edge is a weak tie if its NOVER score is less than or equal to a threshold NOVER score; otherwise, the edge is classified as a strong tie.
- Edges with lower NOVER scores bridge two different communities (as the end vertices of these edges have few common neighbors)
- Edges with higher NOVER scores are more likely to connect vertices within a community as the end vertices of these edges have more common neighbors.
- We will now see a simple community detection algorithm based on this notion of weak ties and strong ties.
  - Given a threshold NOVER score (or an appropriate score determined using the **Strong Triadic Closure Property**), all edges with NOVER score less than or equal to the threshold score will be classified as weak ties and the rest as strong ties.
  - It is just a one-step algorithm. We will remove all the weak ties and the components resulting from these removals will constitute the different communities.

Weak Ties-based Detection: Ex-1





Let threshold NOVER score be 0.4

Modularity(5,6,7,8)
Mod(5,6) = 1 – (5*5)/(2*15) = 0.17
Mod(5,7) = 1 – (3*5)/(2*15) = 0.50
Mod(5,8) = 1 – (3*5)/(2*15) = 0.50
Mod(6,7) = 1 – (3*5)/(2*15) = 0.50
Mod(6,8) = 1 – (3*5)/(2*15) = 0.50
Mod(7,8) = 1 – (3*3)/(2*15) = 0.70

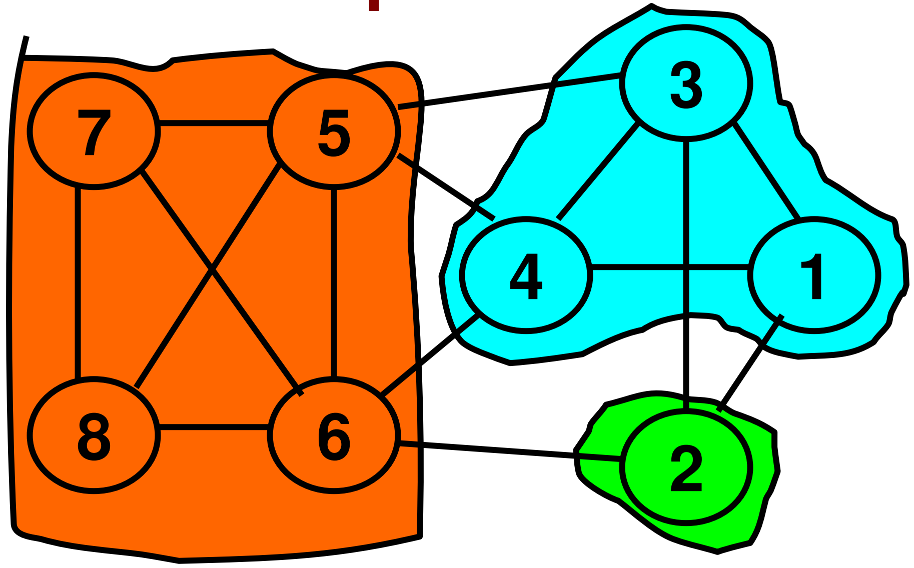
Modularity (5, 6, 7, 8) = 2.87

Modularity(1, 3, 4)
Mod(1,3) = 1 – (3*4)/(2*15) = 0.60
Mod(1,4) = 1 – (3*4)/(2*15) = 0.60
Mod(3,4) = 1 – (4*4)/(2*15) = 0.47

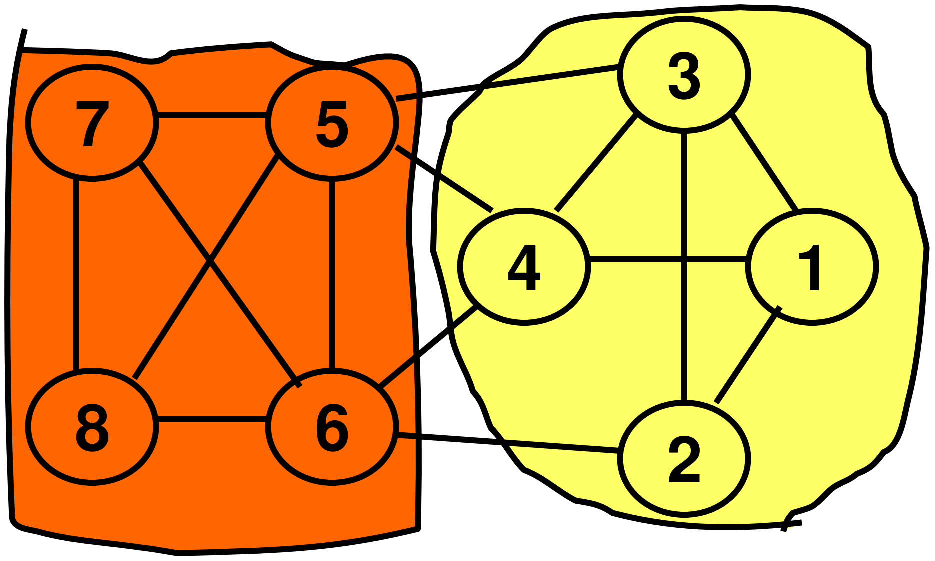
Modularity (1, 3, 4) = 1.67

Total Modularity Score = 4.54

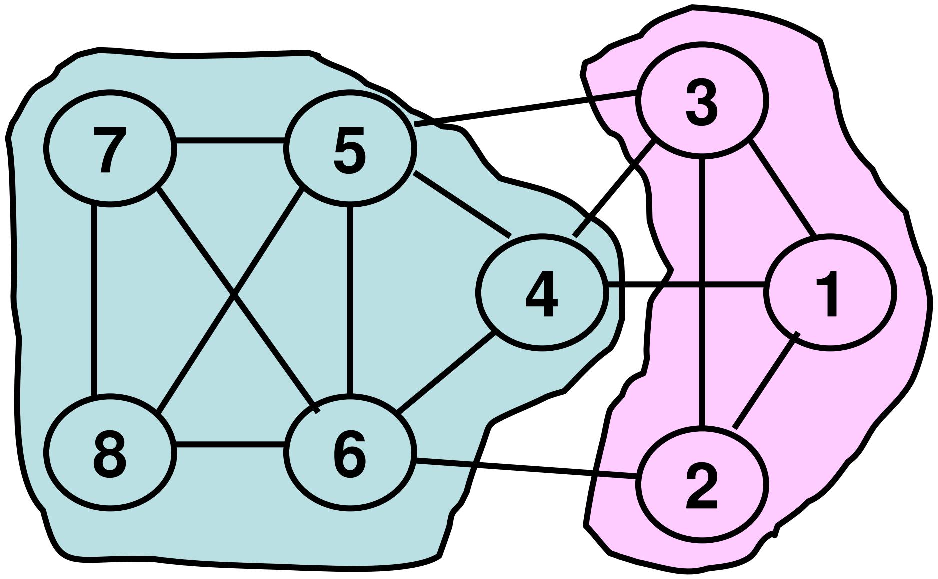
Comparison of Algorithms: Ex-1



Weak Ties-based Algorithm
Total Modularity Score = 4.54

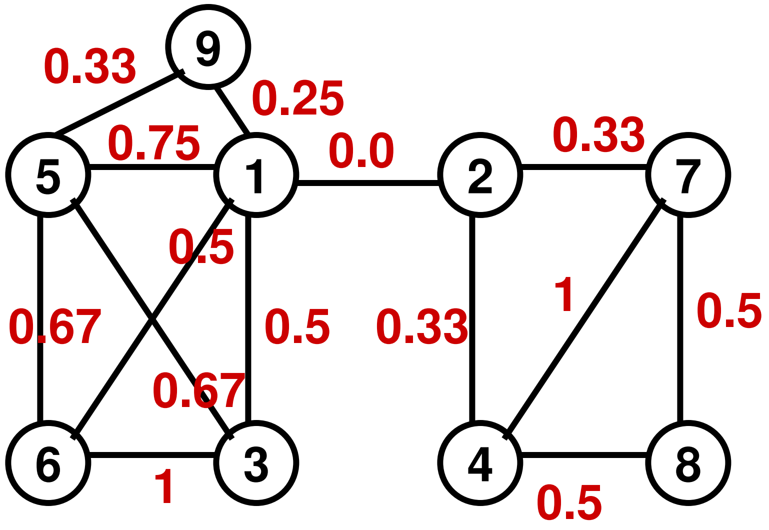


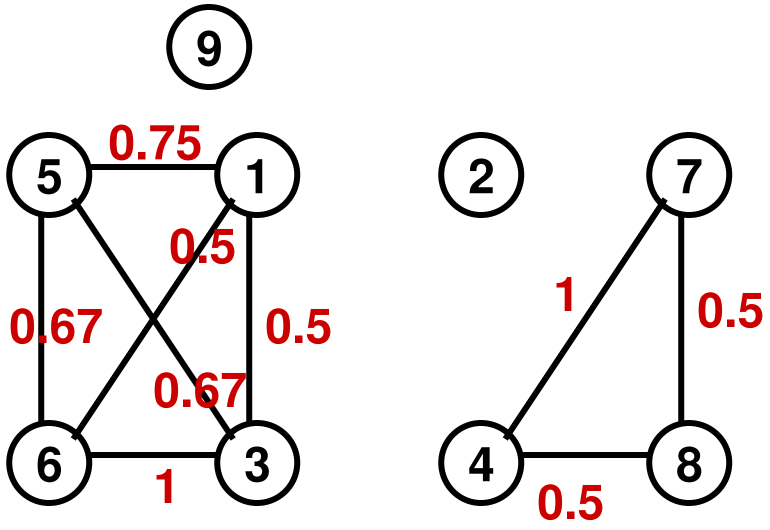
NOVER/BW-based Algorithm
Total Modularity Score = 6.44



Complete Linkage-
Based Algorithm
Total Modularity Score = 4.64

Weak Ties for Community Detection: Ex-2





Let threshold NOVER score be 0.4

Modularity (1, 3, 5, 6)

Mod (1, 3) = 1 – (5*3)/(2*14) = 0.46
Mod (1, 5) = 1 – (5*4)/(2*14) = 0.28
Mod (1, 6) = 1 – (5*3)/(2*14) = 0.46
Mod (3, 5) = 1 – (3*4)/(2*14) = 0.57
Mod (3, 6) = 1 – (3*3)/(2*14) = 0.68
Mod (5, 6) = 1 – (4*3)/(2*14) = 0.57

Modularity (1, 3, 5, 6) = 3.02

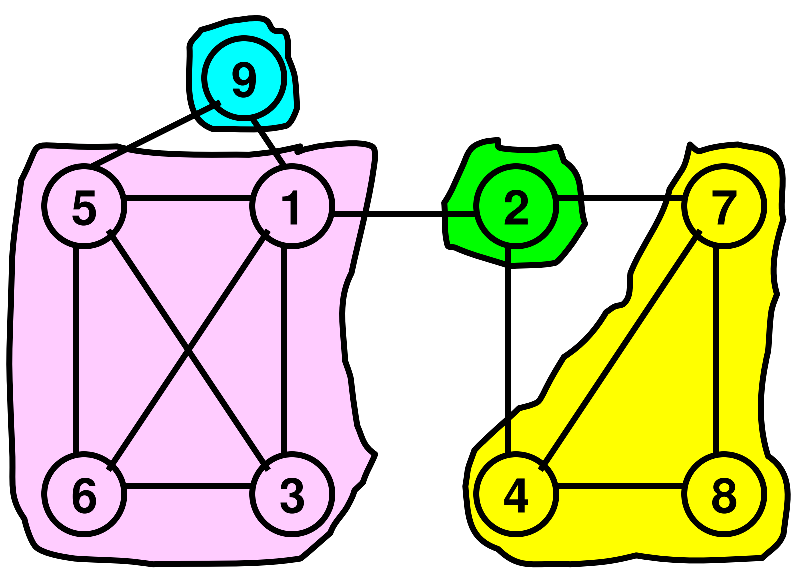
Modularity (4, 7, 8)

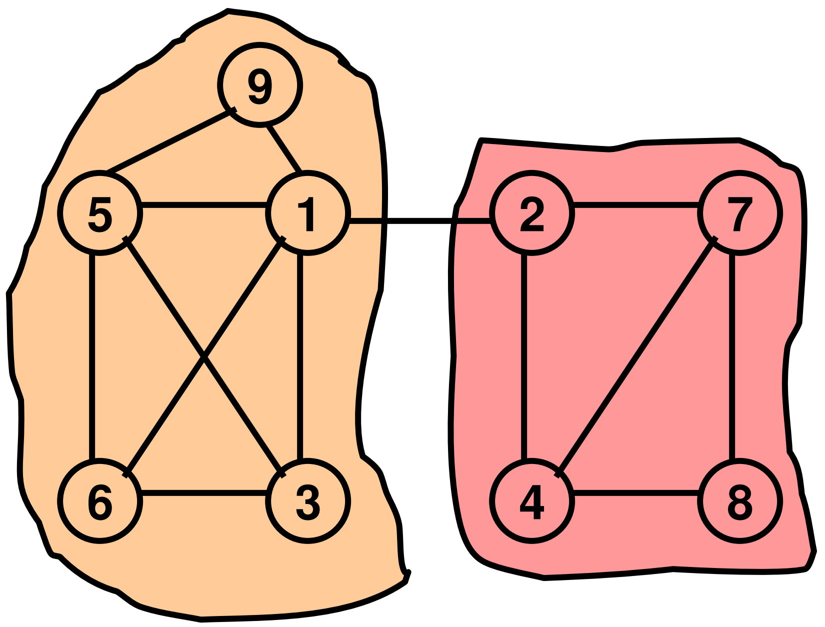
Mod (4, 7) = 1 – (3*3)/(2*14) = 0.68
Mod (4, 8) = 1 – (3*2)/(2*14) = 0.78
Mod (7, 8) = 1 – (3*2)/(2*14) = 0.78

Modularity (4, 7, 8) = 2.24

Total Modularity Score = 5.26

Comparison of Algorithms: Ex-2





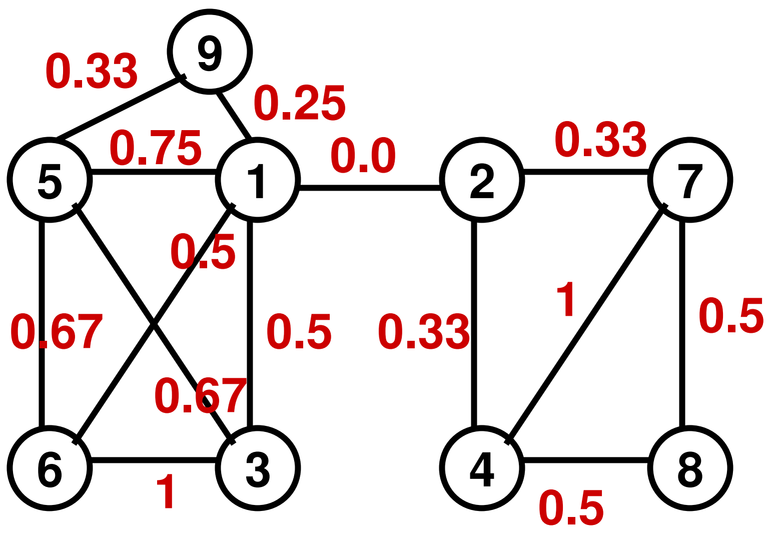
Weak Ties-based Algorithm
Total Modularity Score = 5.26

NOVER/BW-based Algorithm
Total Modularity Score =7.34

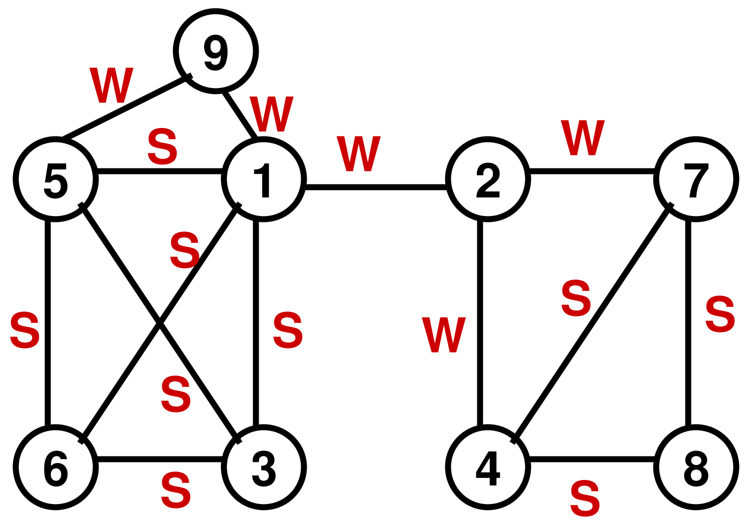
Strong Triadic Closure Property

- If a node A has strong ties to two nodes B and C, then B and C are expected to have at least a weak tie between them.
  - More relevant for social networks
- A node that satisfies the above property for any of its two neighbors with which it has a strong tie is said to exhibit the Strong Triadic Closure property; otherwise the node is said to VIOLATE the property.
- A threshold NOVER score is considered appropriate only if the strong triadic closure property is satisfied for every node.
  - Smaller the threshold NOVER score, larger the chances for the property not being satisfied and vice-versa.

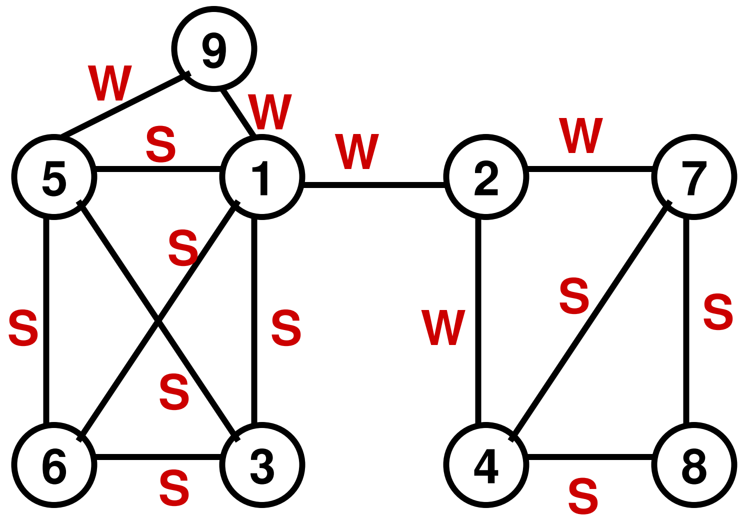
We will use the NOVER scores of the edges to pick a candidate threshold NOVER score



Let threshold NOVER score be 0.4



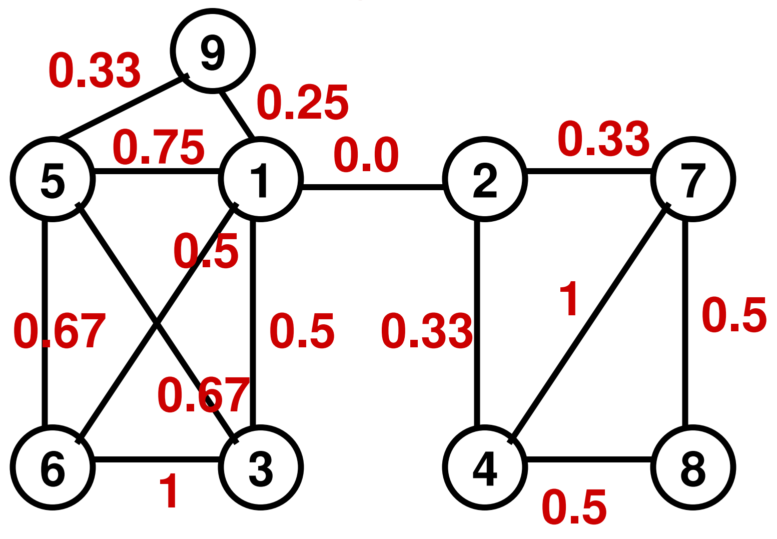
Strong Triadic Closure: Ex-1



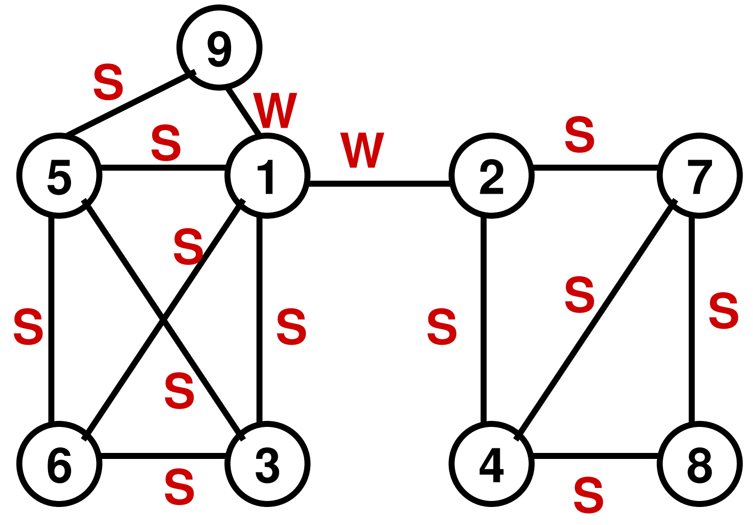
Let threshold NOVER score be 0.4

Node	Strong Tie Neighbors	Weak Tie Neighbors	Strong Triadic Closure Property
1	3, 5, 6	2, 9	YES
2	-	1, 4, 7	N/A
3	1, 5, 6	-	YES
4	7, 8	2	YES
5	1, 3, 6	9	YES
6	1, 3, 5	-	YES
7	4, 8	2	YES
8	4, 7	-	YES
9	-	1, 5	N/A

Strong Triadic Closure: Ex-1 (1)

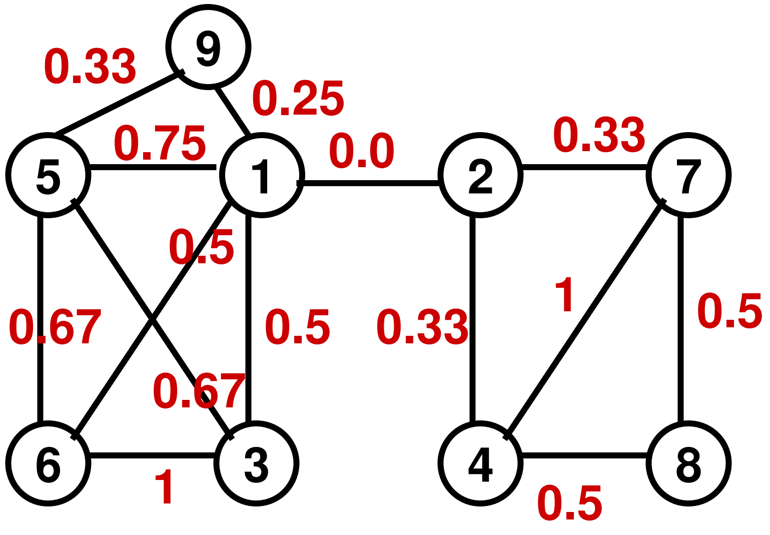


Let threshold NOVER score be 0.25

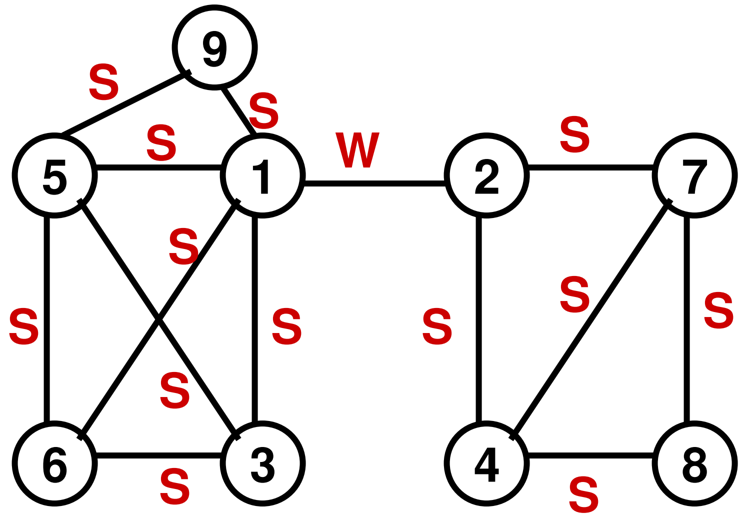


Node	Strong Tie Neighbors	Weak Tie Neighbors	Strong Triadic Closure Property
1	3, 5, 6	2, 9	YES
2	4, 7	1	YES
3	1, 5, 6	-	YES
4	2, 7, 8	-	VIOLATED (no edge between 2, 8)
5	1, 3, 6, 9	-	VIOLATED (no edge: 3, 9; 6, 9)
6	1, 3, 5	-	YES
7	2, 4, 8	-	VIOLATED (no edge between 2, 8)
8	4, 7	-	YES
9	5	1	N/A

Strong Triadic Closure: Ex-1 (2)

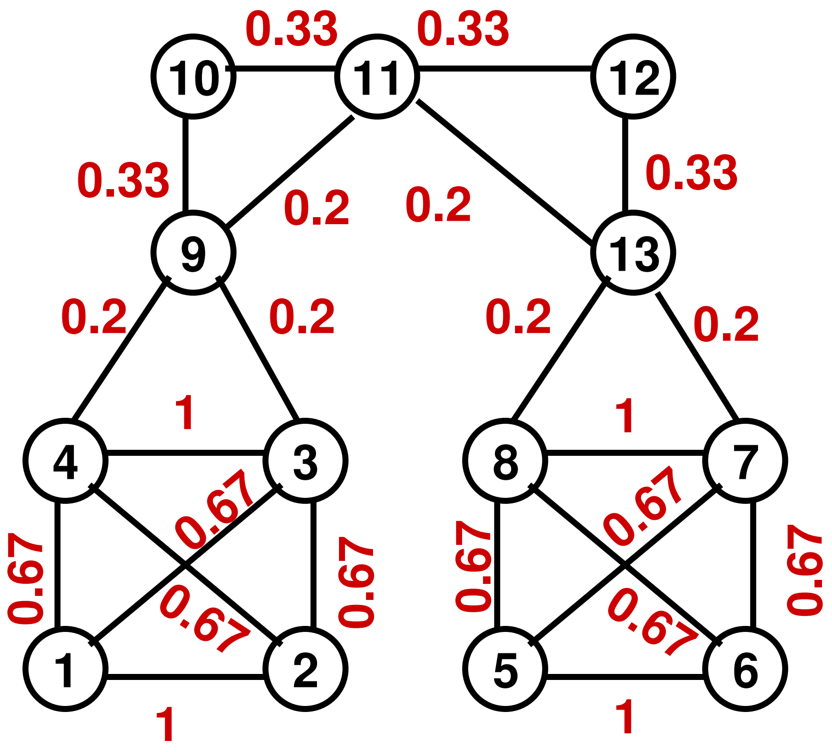


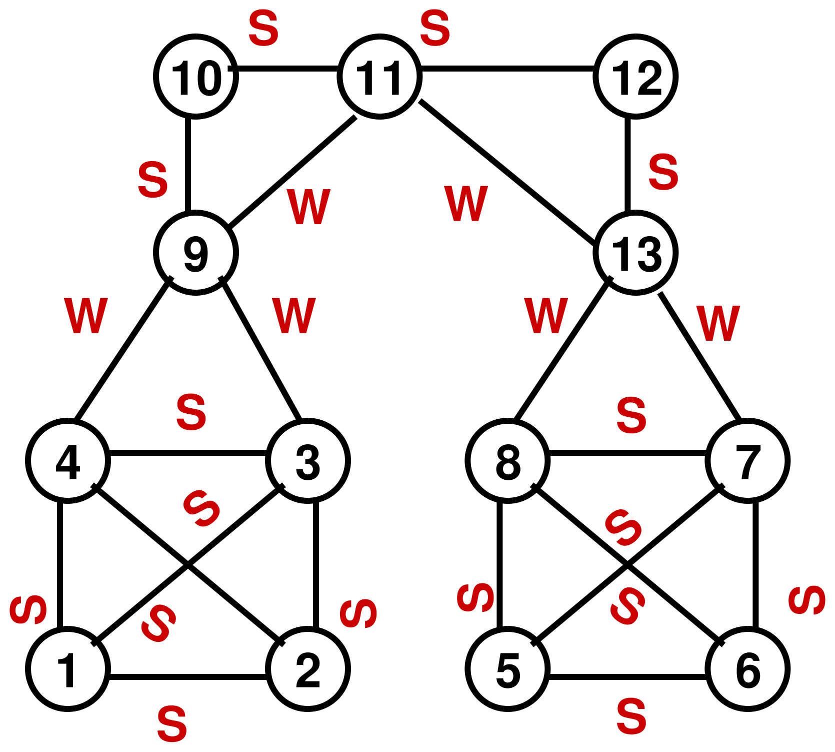
Let threshold NOVER score be 0.0



Node	Strong Tie Neighbors	Weak Tie Neighbors	Strong Triadic Closure Property
1	3, 5, 6, 2, 9	-	VIOLATED (no edge: 2, 3; 3, 9; 2, 5; etc)
2	4, 7, 1	-	VIOLATED (no edge: 4, 1; 7, 1)
3	1, 5, 6	-	YES
4	2, 7, 8	-	VIOLATED (no edge between 2, 8)
5	1, 3, 6, 9	-	VIOLATED (no edge between 3, 9)
6	1, 3, 5	-	YES
7	2, 4, 8	-	VIOLATED (no edge between 2, 8)
8	4, 7	-	YES
9	5, 1	-	YES

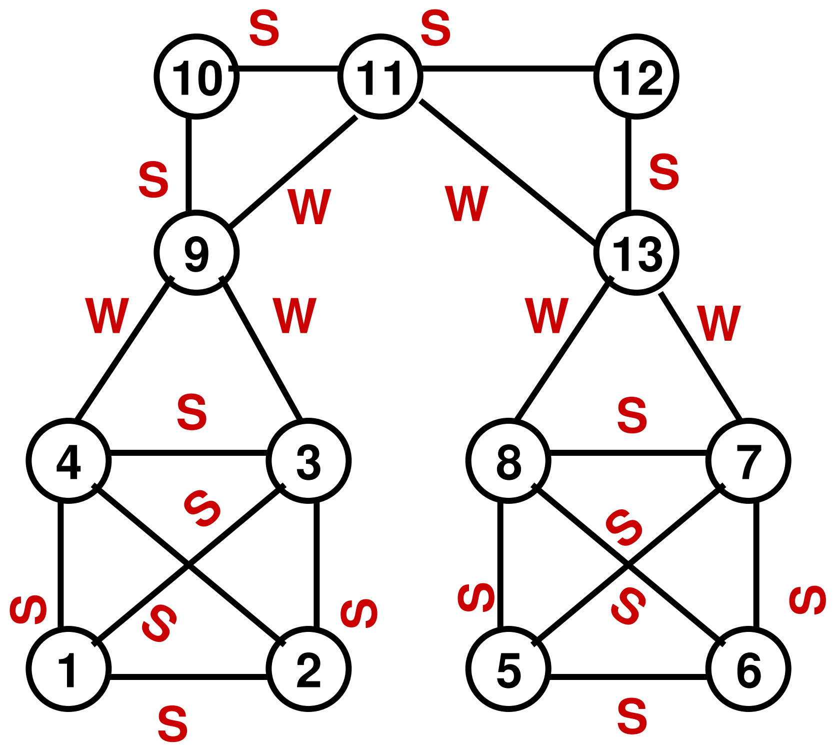
Strong Triadic Closure: Ex-2





Trial # 1
Threshold NOVER Score = 0.2

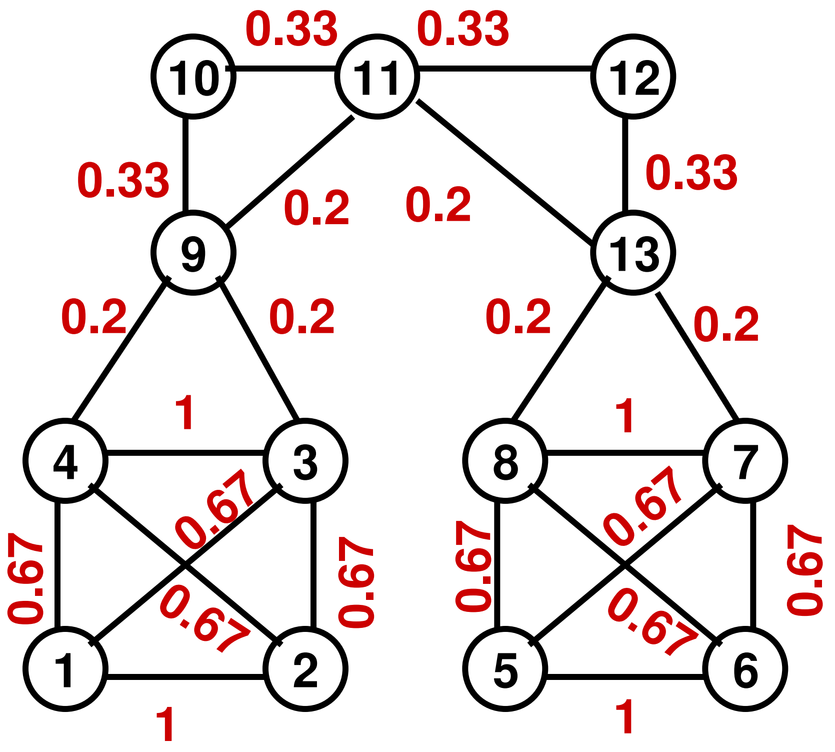
Strong Triadic Closure: Ex-2 (1)

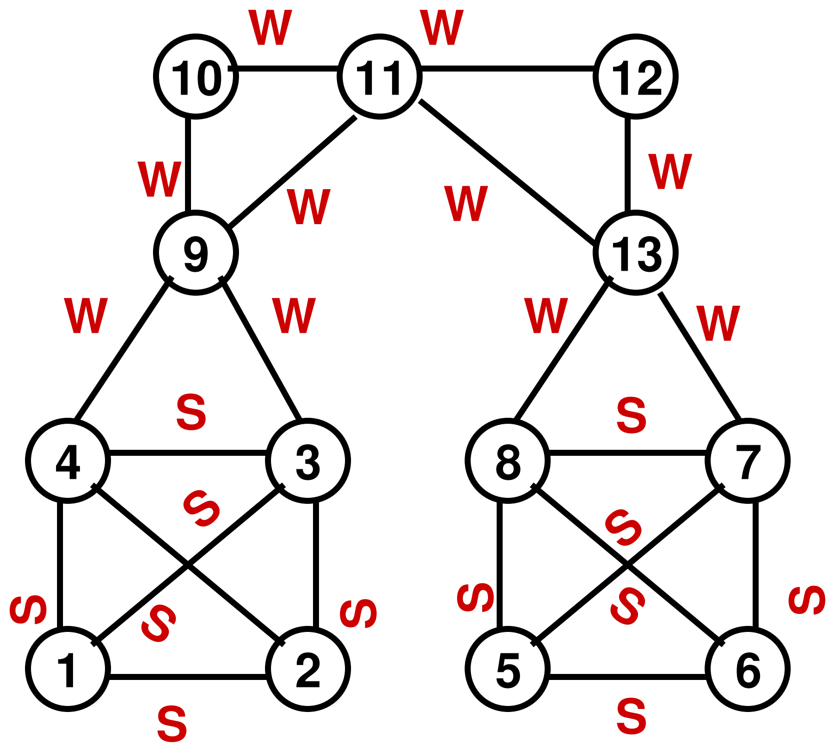


Trial # 1
Threshold NOVER Score = 0.2

Node	Strong Tie Neighbors	Weak Tie Neighbors	Strong Tri. Clos. Prop.
1	2, 3, 4	-	YES
2	1, 3, 4	-	YES
3	1, 2, 4	9	YES
4	1, 2, 3	9	YES
5	6, 7, 8	-	YES
6	5, 7, 8	-	YES
7	5, 6, 8	13	YES
8	5, 6, 7	13	YES
9	10	3, 4, 11	N/A
10	9, 11	-	YES
11	10, 12	9, 13	VIOLATED
12	11, 13	-	YES
13	12	11, 8, 7	N/A

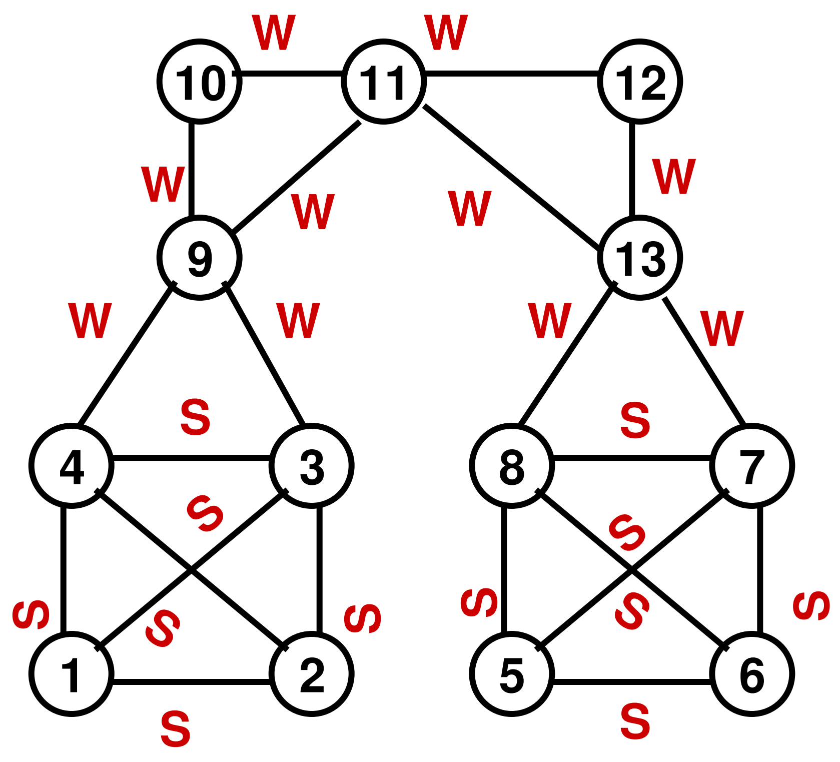
Strong Triadic Closure: Ex-2 (2)





Trial # 2
Threshold NOVER Score = 0.33

Strong Triadic Closure: Ex-2 (3)

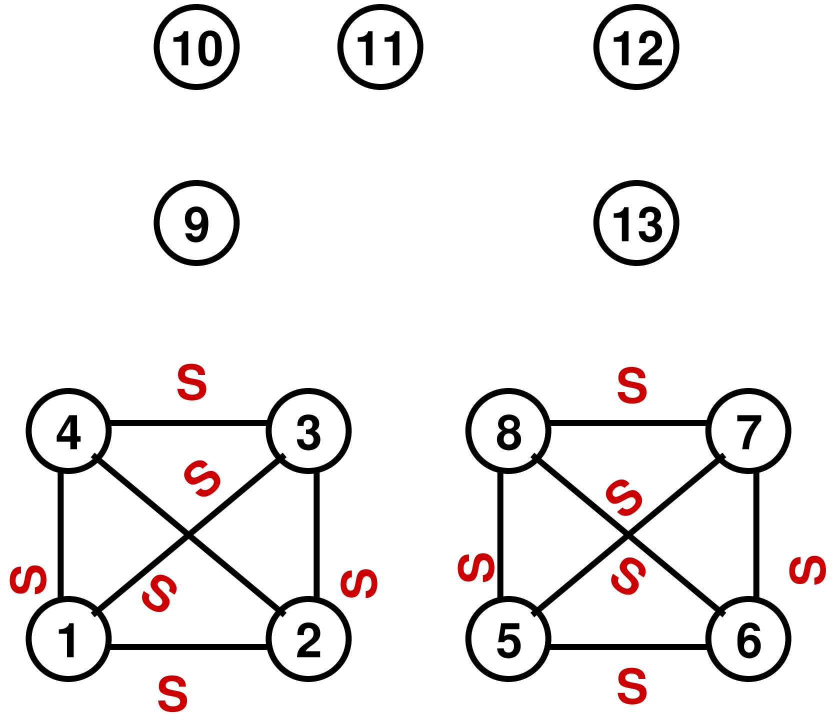


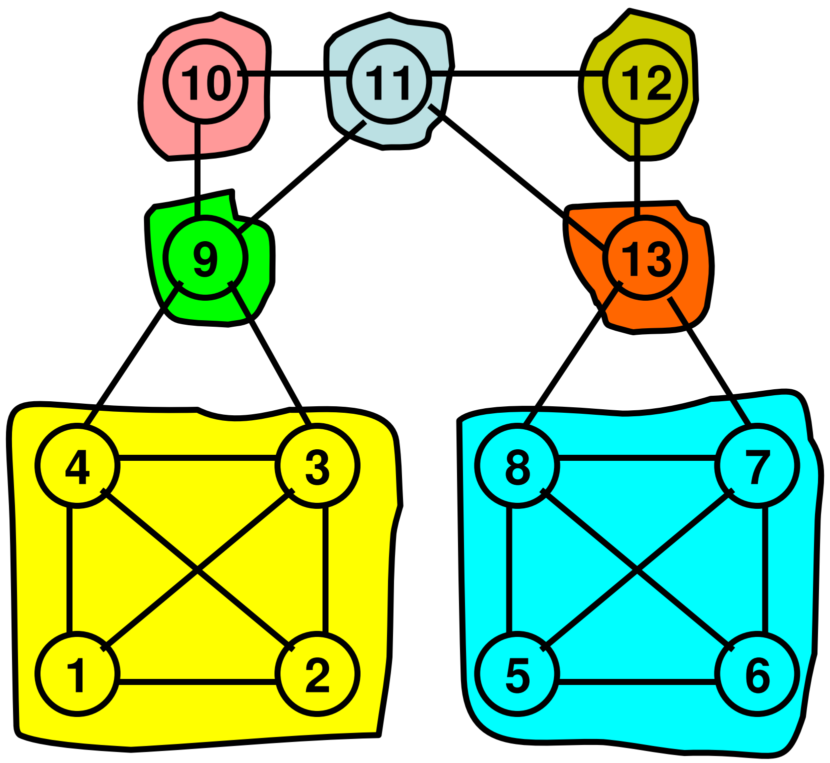
Trial # 2
Threshold NOVER Score = 0.33

Node	Strong Tie Neighbors	Weak Tie Neighbors	Strong Tri. Clos. Prop.
1	2, 3, 4	-	YES
2	1, 3, 4	-	YES
3	1, 2, 4	9	YES
4	1, 2, 3	9	YES
5	6, 7, 8	-	YES
6	5, 7, 8	-	YES
7	5, 6, 8	13	YES
8	5, 6, 7	13	YES
9	-	10, 3, 4, 11	N/A
10	-	9, 11	N/A
11	-	10, 12, 9, 13	N/A
12	-	11, 13	N/A
13	-	12, 11, 8, 7	N/A

Since the Strong Triadic Closure Property is NOT VIOLATED for any node, We will use the Threshold NOVER Score = 0.33.

Weak Ties-based Detection: Ex-2





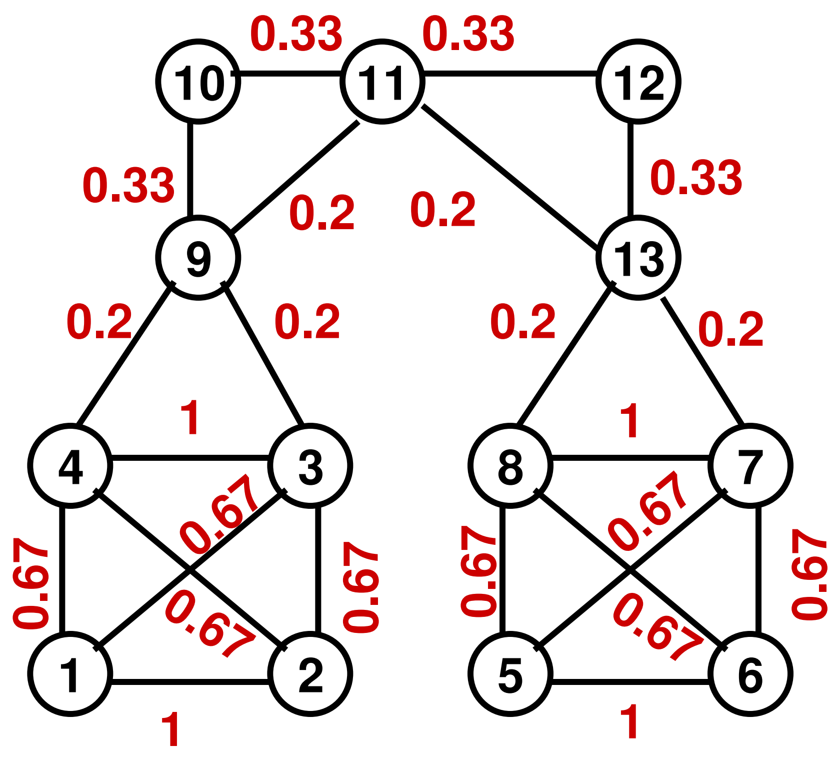
Modularity (1, 2, 3, 4)

Mod (1, 2) = 0.795
Mod (1, 3) = 0.727
Mod (1, 4) = 0.727
Mod (2, 3) = 0.727
Mod (2, 4) = 0.727
Mod (3, 4) = 0.636
Modularity (1, 2, 3, 4) = 4.339

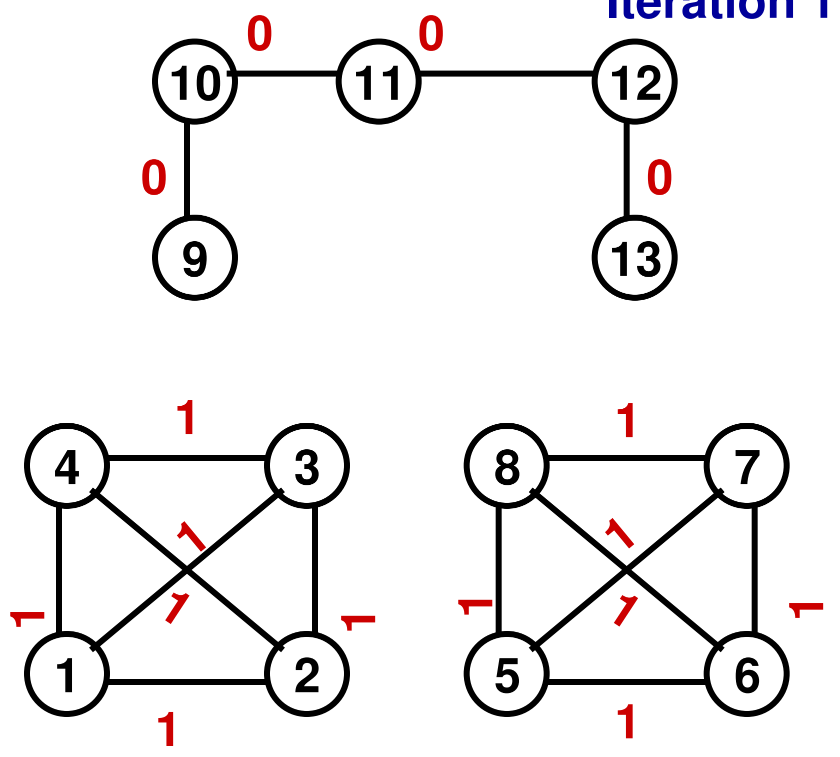
Modularity (5, 6, 7, 8) = 4.339

Total Modularity Score = 8.678

NOVER-based GN: Ex-2



Iteration 1



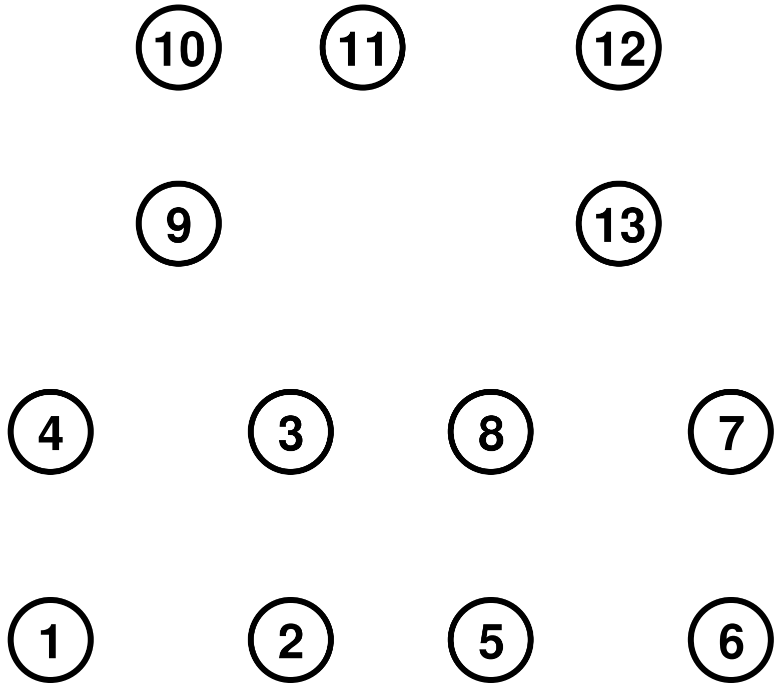
Modularity (1, 2, 3, 4) = 4.339
Modularity (5, 6, 7, 8) = 4.339

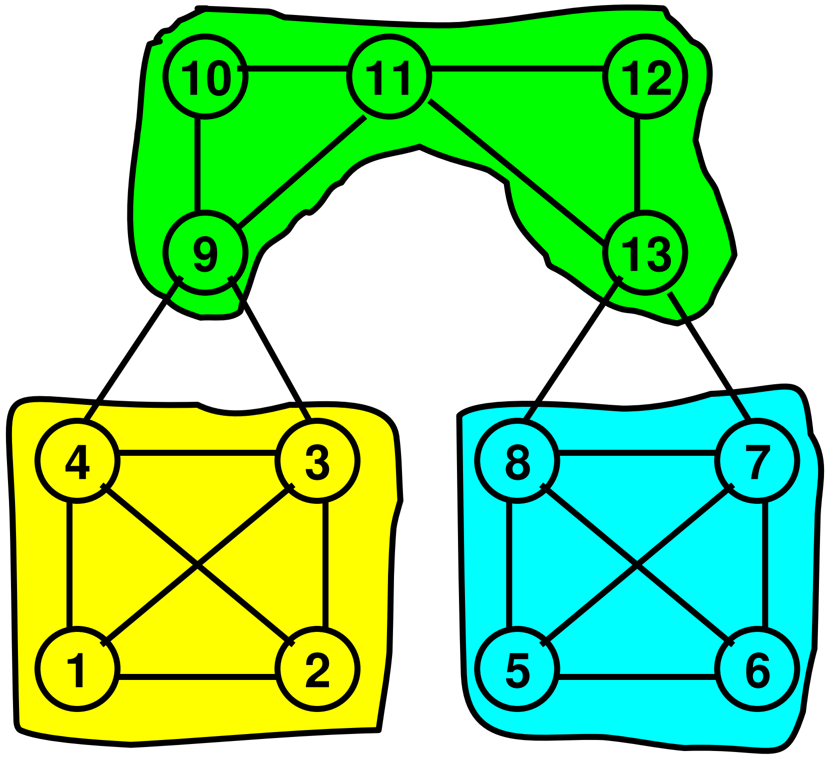
Modularity (9, 10, 11, 12, 13)

Mod (9, 10) = 0.818	Mod (11,1 2) = 0.818
Mod (9, 11) = 0.636	Mod (11, 13) = 0.636
Mod (9, 12) = -0.182	Mod (12, 13) = 0.818
Mod (9, 13) = -0.364	Modularity (9, …, 13)
Mod (10, 11) = 0.818	= 3.725
Mod (10, 12) = -0.091	Total Modularity
Mod (10, 13) = -0.182	= 12.403

NOVER-based GN: Ex-2 (1)

Iteration 2





Total Modularity = 0

Homophily-based Community Detection

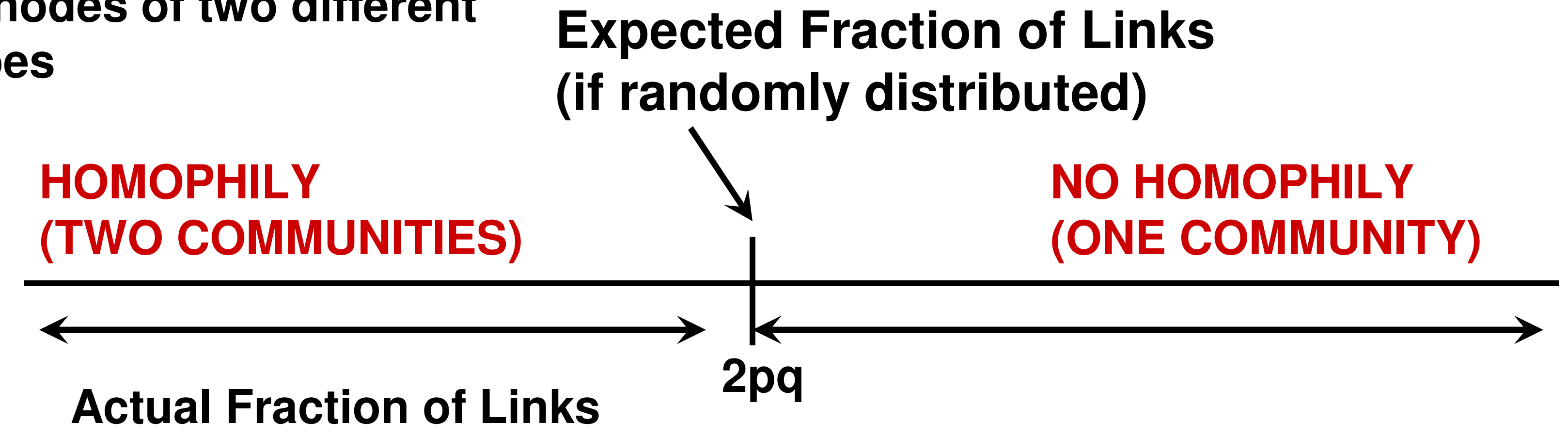
Homophily

- In social networks, people tend to associate more closer with people who are similar to each other with respect to race, ethnicity, job, home town, etc (offline characteristics).
  - Homophily
- The community detection algorithms we have seen until now do not take the offline characteristics of the nodes into consideration.
- We will now assume the offline info of the nodes are available and we make use of this info for community detection.

- If p and q are the fractions of nodes of certain type in a network, then if the links in the network are randomly distributed (without taking the type of the nodes), the fraction of the links expected to connect these two types of nodes is **2pq**.
- If the actual fraction of links in a network connecting nodes of two different types is less than 2pq, we say the network exhibits a homophily, and the nodes cannot be part of one single community.
- If the actual fraction of links in a network connecting nodes of two different types is greater than or equal to 2pq, then the network is not considered to exhibit homophily, and the nodes are considered to be part of just one single community.

Measuring Homophily

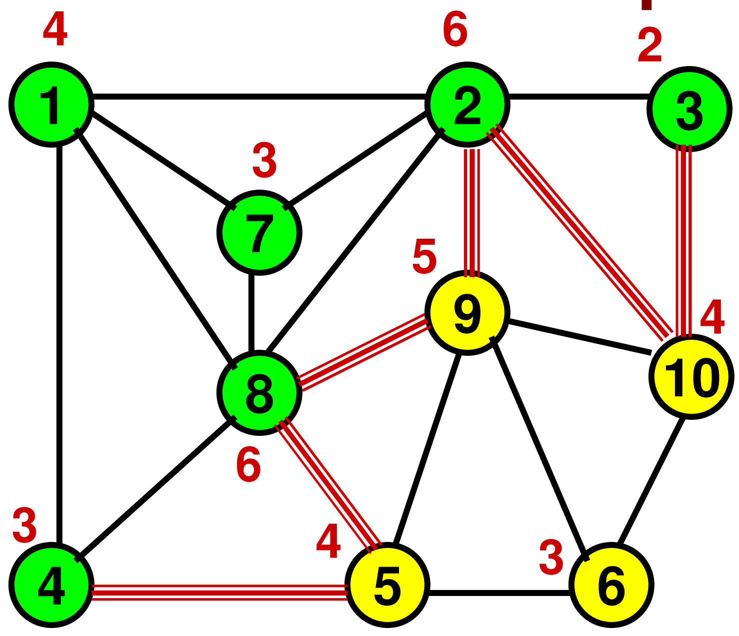
p and q are the fractions of nodes of two different types

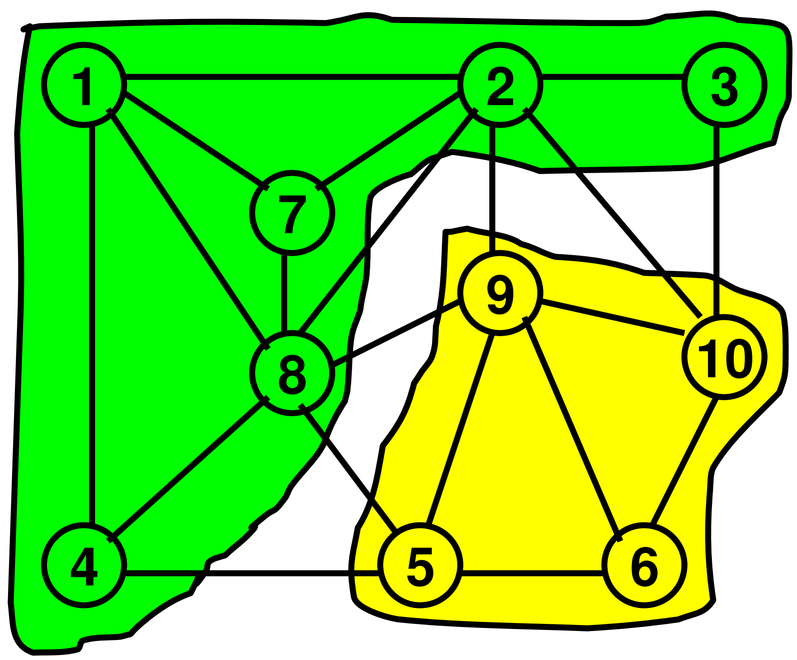


If two sets of nodes are to be of their own community (i.e., for HOMOPHILY to exist), we would expect a relatively lower number of cross-community links between them.

FOR HOMOPHILY TO EXIST,
The actual fraction of cross-community links between two different communities should be less than the expected fraction of links between two the sets of nodes if they were to exist as one community.

Homophily: Example-1





# Nodes = 10
# Links = Sum of Degrees / 2 = 20

Male Students: 1, 2, 3, 4, 7, 8

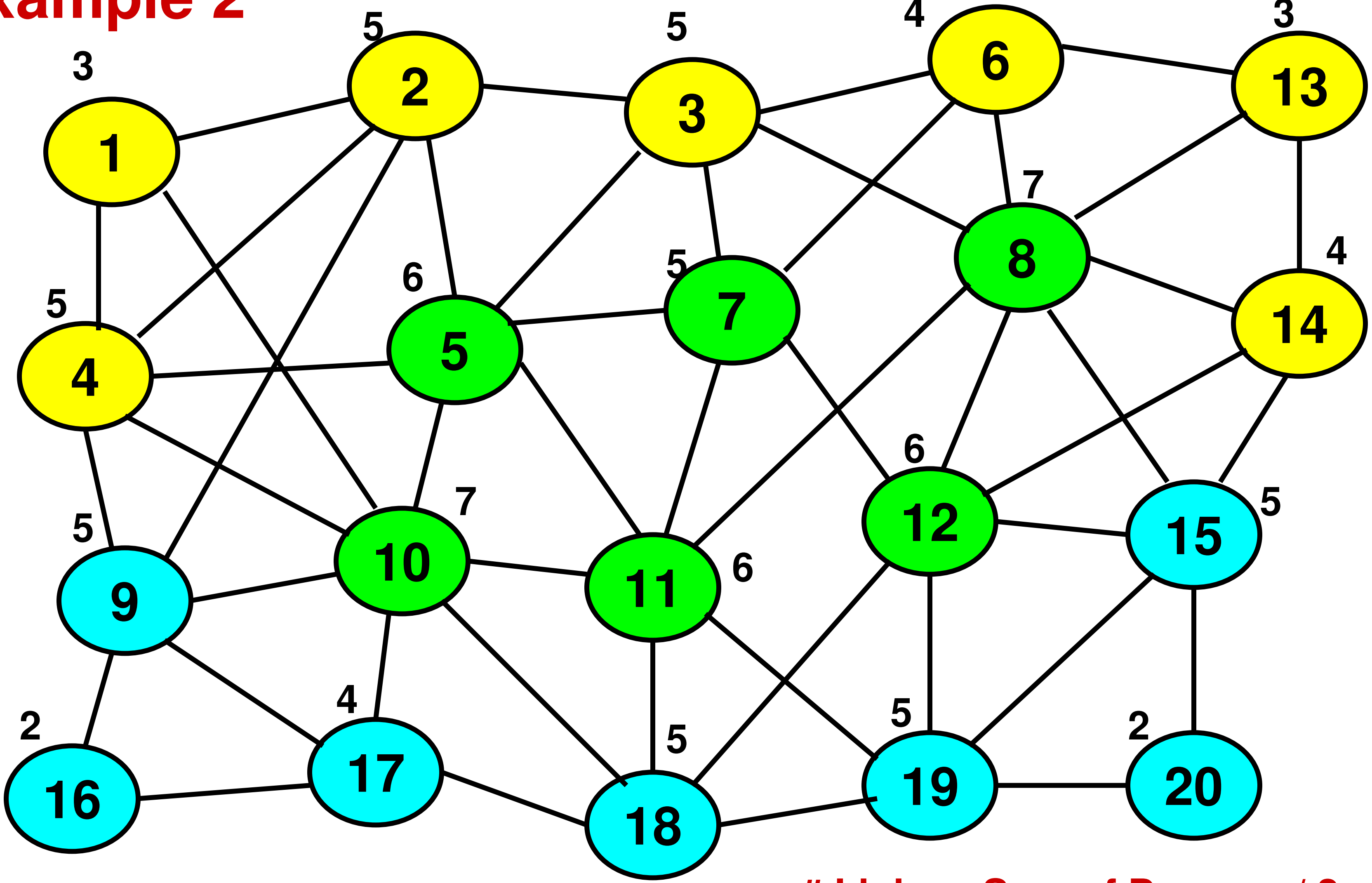
Female Students: 5, 6, 9, 10

Fraction of Male Nodes: 6/10 = 0.6
Fraction of Female Nodes: 4/10 = 0.4
Expected Fraction of Male-Female Links = 2*0.6*0.4 = 0.48

# Actual Links between Male and Female nodes: 6

Actual Fraction of Male-Female Links = 6/20 = 0.3 < 0.48
Hence, there is HOMOPHILY.

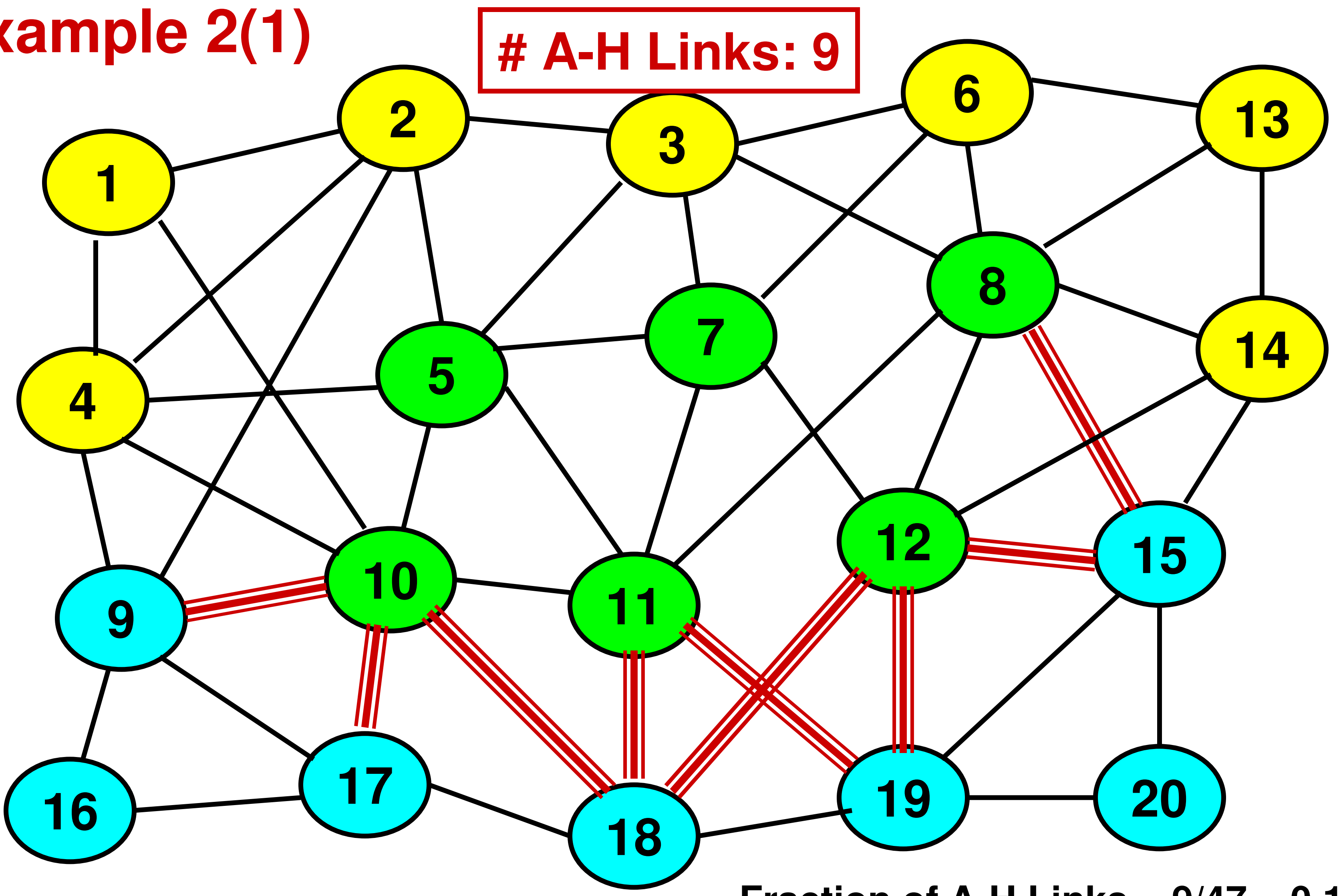
Example 2



# Links = Sum of Degrees / 2 = 47

Asian Origin Vertices:	5, 7, 8, 10, 11, 12	(A): 6/20 = 0.3
Caucasian Origin Vertices:	1, 2, 3, 4, 6, 13, 14	(C): 7/20 = 0.35
Hispanic Origin Vertices:	9, 15, 16, 17, 18, 19, 20	(H): 7/20 = 0.35

Example 2(1)

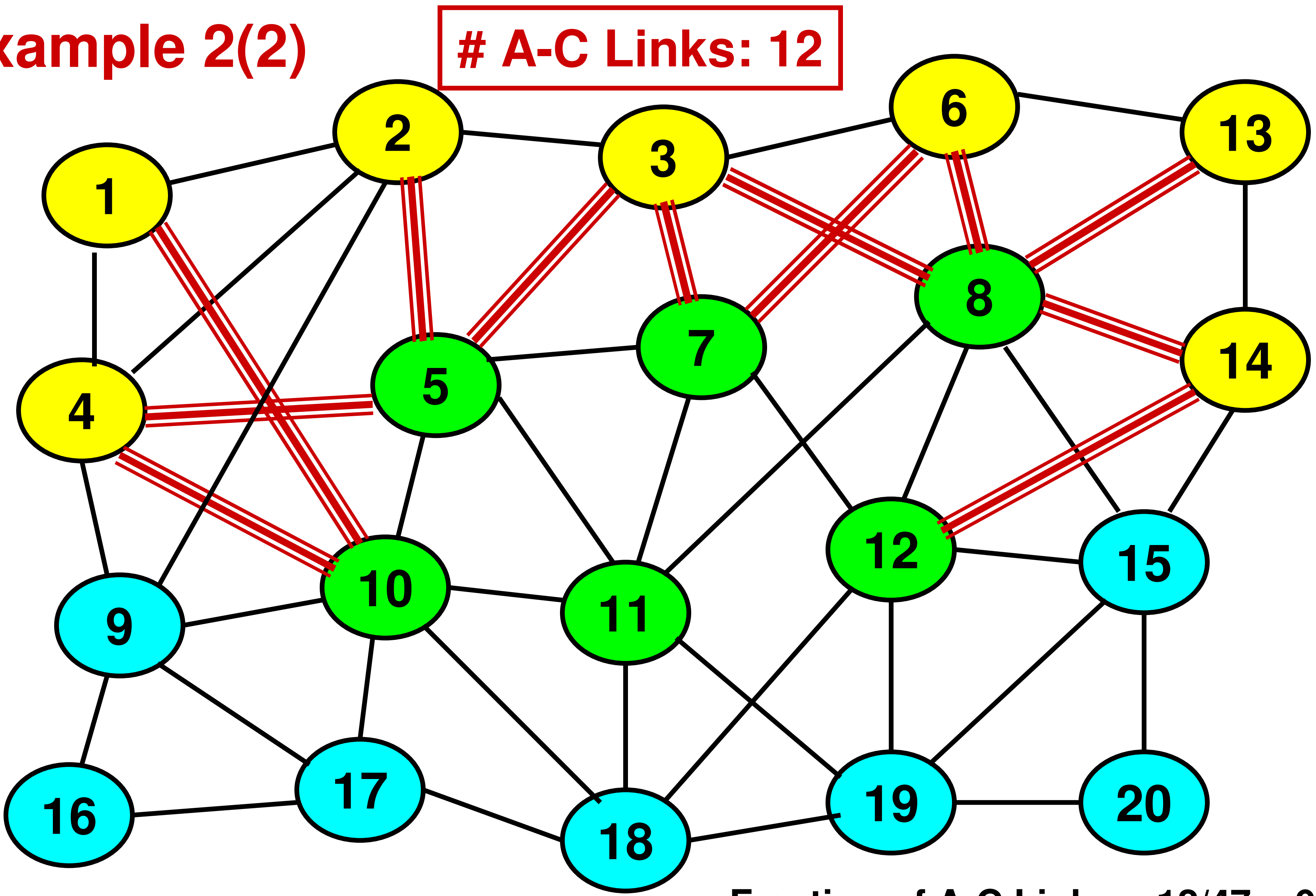


🟩 5, 7, 8, 10, 11, 12
🟨 1, 2, 3, 4, 6, 13, 14
🟦 9, 15, 16, 17, 18, 19, 20

(A): 6/20 = 0.3
(C): 7/20 = 0.35
(H): 7/20 = 0.35

Fraction of A-H Links = 9/47 = 0.19 < 2*A*H = 2*0.3*0.35 = 0.21
Hence, the Asians and Hispanics exist as two different communities and NOT together.

Example 2(2)

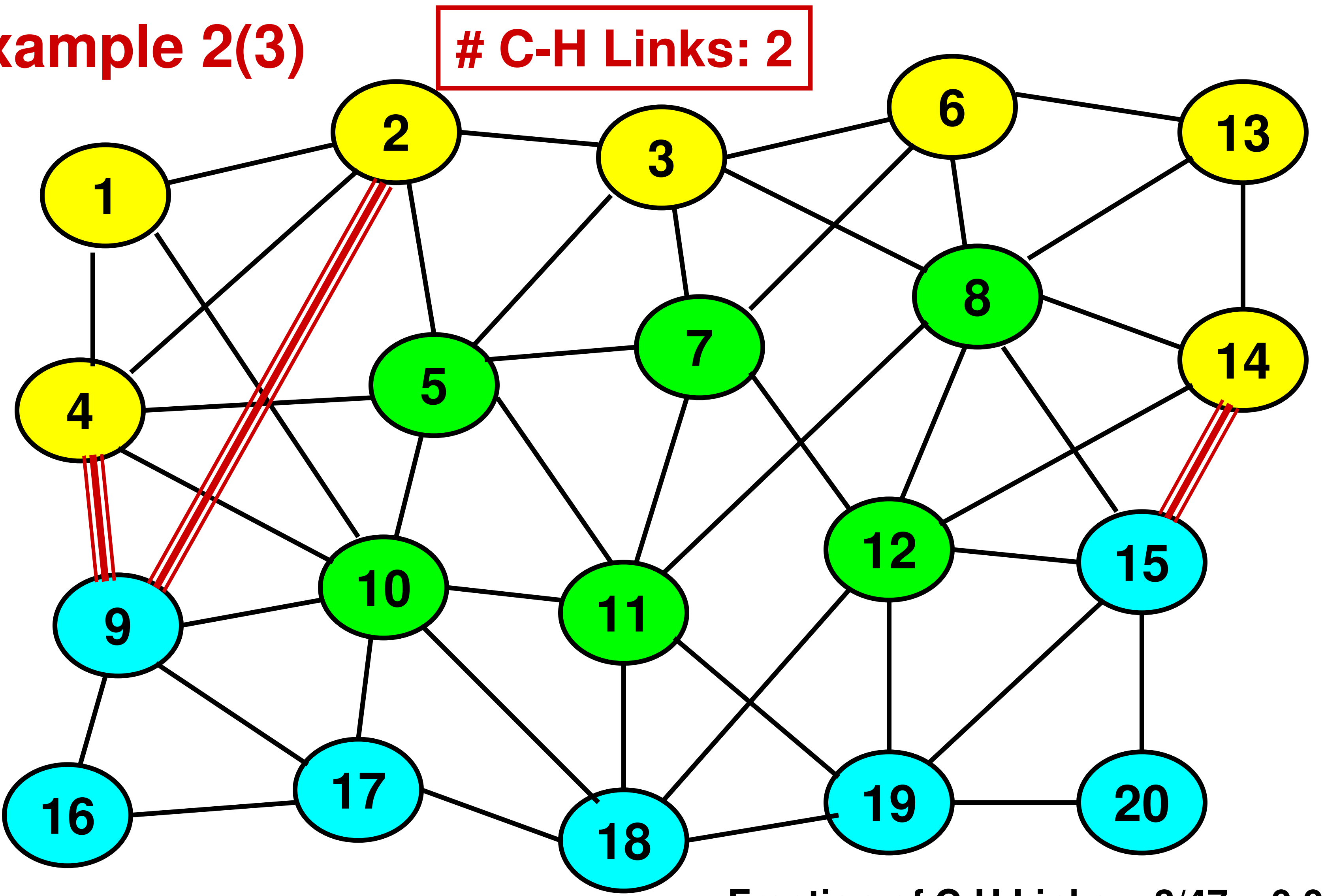


🟩 5, 7, 8, 10, 11, 12
🟨 1, 2, 3, 4, 6, 13, 14
🟦 9, 15, 16, 17, 18, 19, 20

(A): 6/20 = 0.3
(C): 7/20 = 0.35
(H): 7/20 = 0.35

Fraction of A-C Links = 12/47 = 0.26
> 2*A*C = 2*0.3*0.35 = 0.21
Hence, the Asians and Caucasians DO NOT exist as two different communities.

Example 2(3)

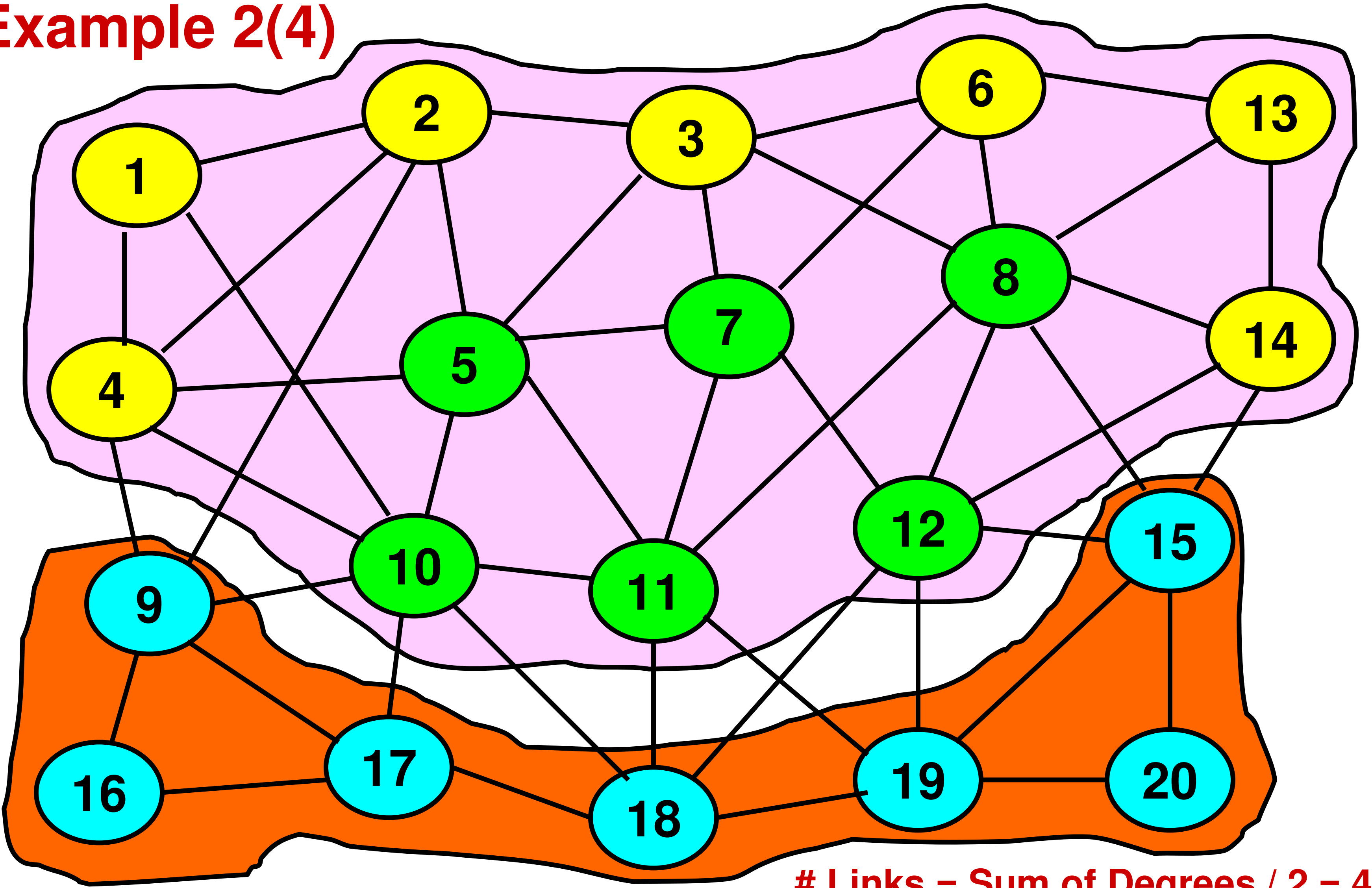


🟩 5, 7, 8, 10, 11, 12
🟨 1, 2, 3, 4, 6, 13, 14
🟦 9, 15, 16, 17, 18, 19, 20

(A): 6/20 = 0.3
(C): 7/20 = 0.35
(H): 7/20 = 0.35

Fraction of C-H Links = 3/47 = 0.064
< 2*C*H = 2*0.35*0.35 = 0.245
Hence, the Caucasians and Hispanics exist as two different communities and NOT together.

Example 2(4)



# Links = Sum of Degrees / 2 = 47

Asian Origin Vertices:	5, 7, 8, 10, 11, 12	(A): 6/20 = 0.3
Caucasian Origin Vertices:	1, 2, 3, 4, 6, 13, 14	(C): 7/20 = 0.35
Hispanic Origin Vertices:	9, 15, 16, 17, 18, 19, 20	(H): 7/20 = 0.35