COMP9312 Cohesive Subgraph Mining and Node Feature Engineering

Outline

- K-core
- K-truss
- Other cohesive subgraph
- Node features

K-core

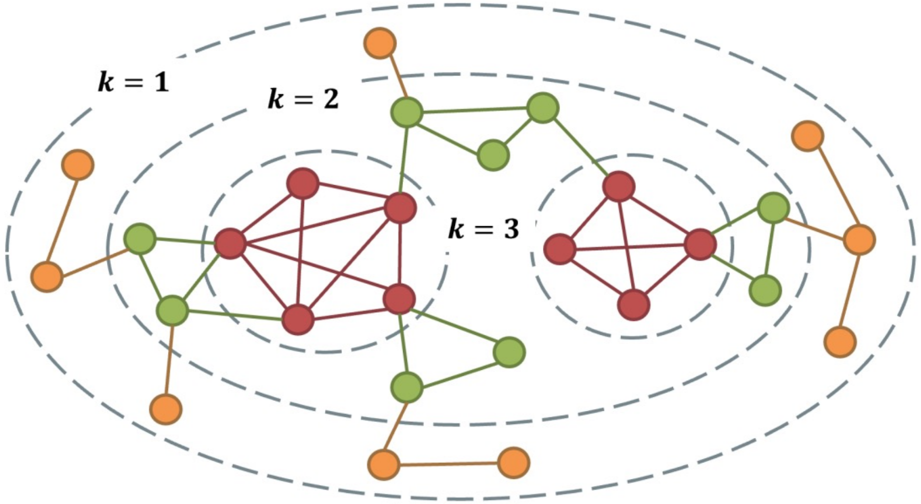
Definition

- K-core is a maximal connected subgraph in which each vertex has at least k neighbors in the subgraph

K-core

- **Core decomposition**

For each unvisited vertex u with the lowest degree in G
  assign core(u) as degree(u)
  mark u as visited
  decrease the degree of its unvisited neighbors with higher degree than u by 1



K-core

- **Time complexity analysis**
  - Iterate over all the vertices takes $O(n)$
  - Get the vertex with min degree in each iteration takes $O(n)$
  - Decrease degree of unvisited neighbors takes $O(m)$
  - Overall time complexity: $O(n^2 + m) = O(n^2)$

Different way to get vertex with min degree in each iteration:

- Using heap: $O(m * \log(n))$
- Using Fabonacci heap: $O(m + n * \log(n))$

K-core

- **Core decomposition using Flat array**

Algorithm : CoreDecomposition

Input : $G = (V, E)$: a graph
Output : $\{cn(u) \mid u \in V\}$: core number of every vertex in G
1 $d(u) \leftarrow deg(u, G)$ for every $u \in V$;
2 order the vertices in V in increasing order of their degrees;
3 **for each** $u \in V$ in the order **do**
4     $cn(u) \leftarrow d(u)$;
5   **for each** $v \in N(u)$ with $d(v) > d(u)$ **do**
6         $d(v) \leftarrow d(v) - 1$;
7   reorder V accordingly;
8 **return** $cn(u)$ of every $u \in V$

- If the degree of neighbor vertex u greater than degree of v, decrease the degree of u by 1.
- Line 7: swap the positions of u and the first vertex with same degree as u's original degree.
- Because we use the bin sort, the time complexity of reorder the array is O(n).
- The total time complexity for core decomposition is O(m).

K-core

- **Core decomposition using Flat array**

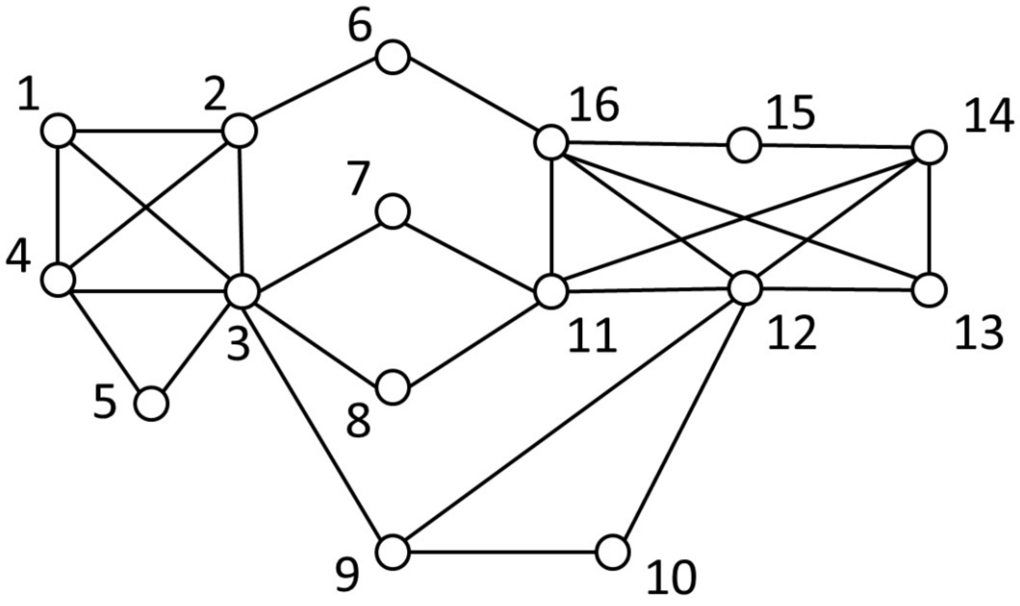
```
function K-CORES(Graph G)
    initialize(d, b, D, p, G)
    for all i ← 1 to n do
        v ← D[i]
        for all u ∈ N_G(v)  do
            if d[u] > d[v] then
                du ← d[u], pu ← p[u]
                pw ← b[du], w ← D[pw]
                if u ≠ w then
                    D[pu] ← w, D[pw] ← u
                    p[u] ← pw, p[w] ← pu
                end if
                b[du]++, d[u]−−
            end if
        end for
    end for
    return d
end function
```

In the implementation, we need

- **D** <- An array to sort vertices in non-decreasing order of degree
- **b** <- An array to locate the start position for each degree
- **p** <- An array to get the position of each vertex id
- **d** <- An array to maintain the degree of each vertex

K-core

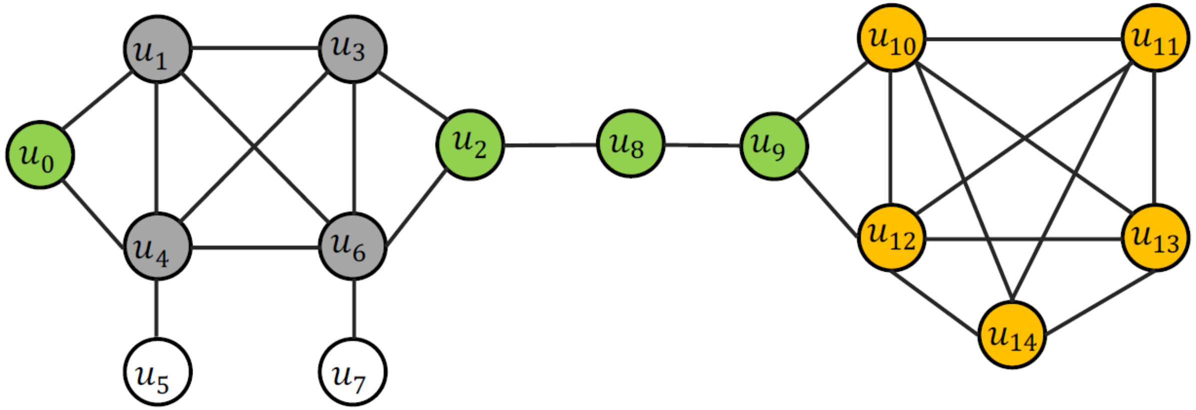
- **Core decomposition using Flat array**



index	**d**	**b**	**D**	**p**
1	3	0	5	7
2	4	1	6	10
3	7	7	7	16
4	4	10	8	11
5	2	13	10	1
6	2	15	15	2
7	2	16	1	3
8	2		9	4
9	3		13	8
10	2		2	5
11	5		4	13
12	6		14	15
13	3		11	9
14	4		16	12
15	2		12	6
16	5		3	14

K-core

- **Exercise:**
  - Implement KcoreDecomposition in tutorial_7.py
  - Find the k-core for 1<=k<=3



Now, you have 15 minutes to do Ex1.4.

K-truss

- **Definition**
  A maximal subgraph where each edge is contained in at least k-2 triangles in the subgraph, i.e., each edge has a support of at least k-2 in the subgraph.

  Each k-truss of G is a subgraph of a (k-1)-core of G.

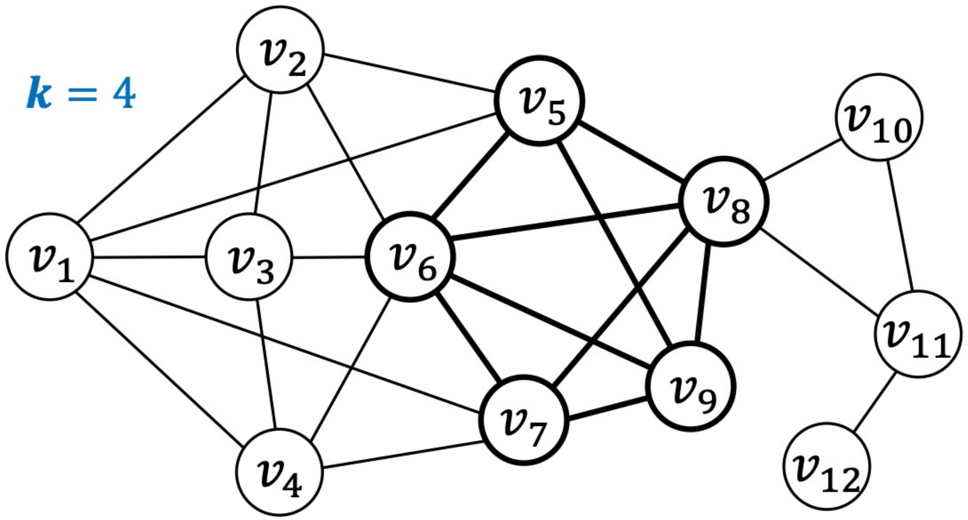
K-truss

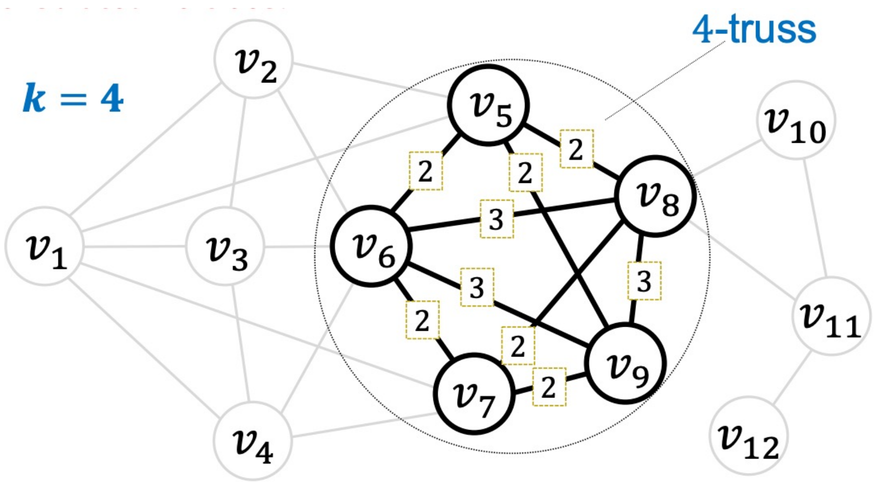
- **K-truss Computation**

1. Compute the (k-1)-core
2. Compute the support of each edge
3. Recursively delete each edge with support of less than k-2
4. Delete the isolated vertices

K-truss

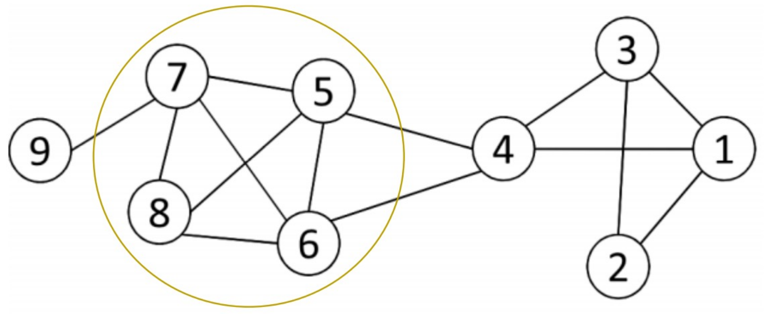
- **K-truss Computation**

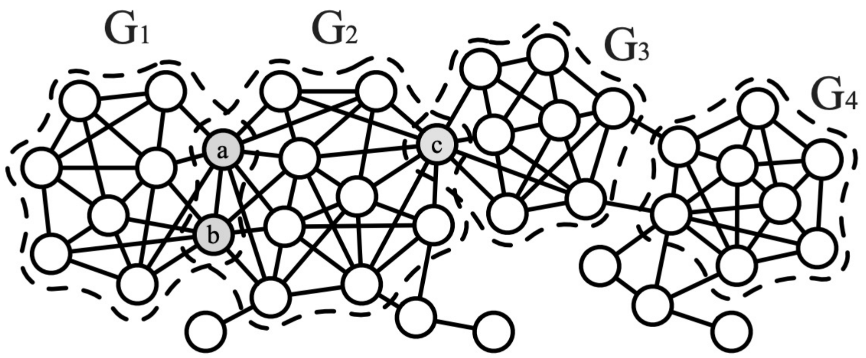




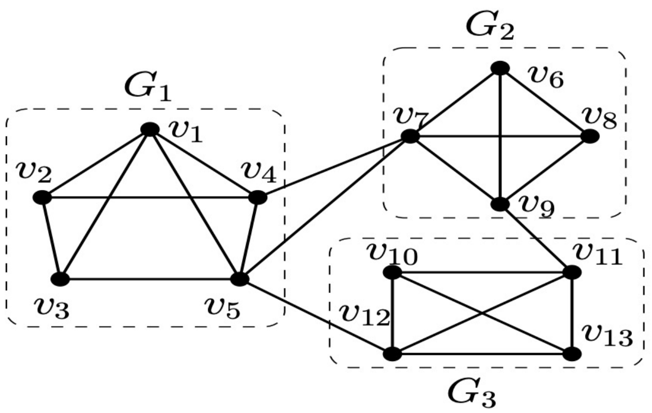
Other Cohesive Subgraph

- K-edge Connected Components
- K-vertex Connected Components
- Clique
- ….





4-Core: {G1∪G2∪G3∪G4}   4-ECC: {G1∪G2∪G3, G4}   4-VCC: {G1, G2, G3, G4}



3-ECCs

Node Feature Engineering

- **Importance based features**
  - Node degree
  - Different node centrality measures

- **Structure-based features**
  - Node degree
  - Clustering coefficient
  - Graphlet count vector

Node Feature Engineering

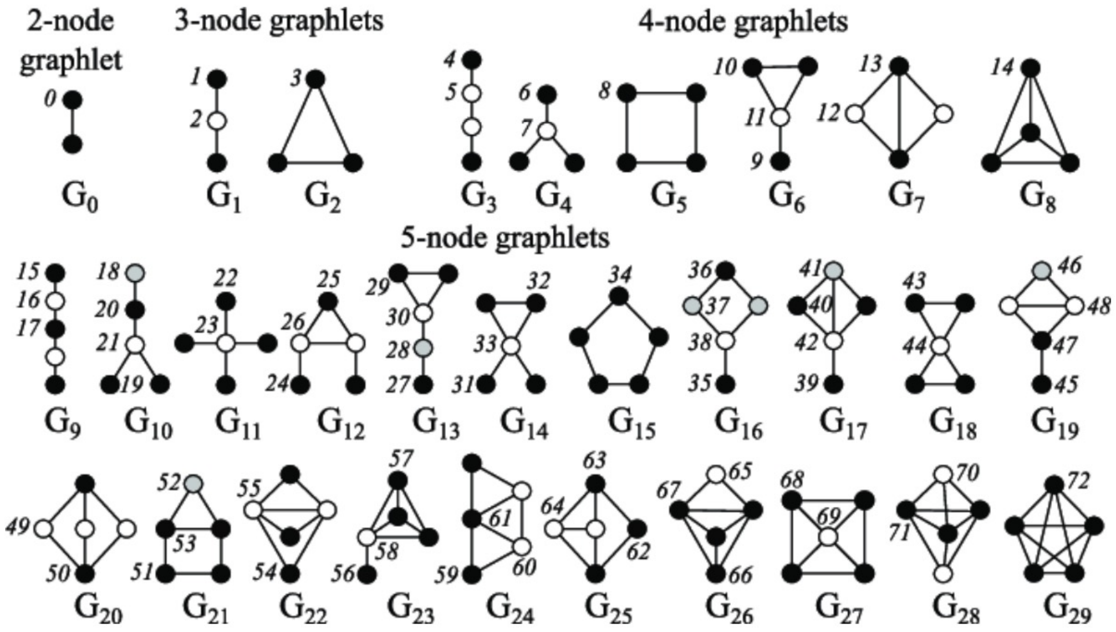
- **Node Centrality: Clustering coefficient**
  measures how connected v's neighboring nodes are

$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\binom{k_v}{2}} \in [0,1]$$

Can be also understand as #triangles/#possible triangles

Node Feature Engineering

- **Graphlet Degree Vector**
  describe network structure around the node based on graphlets



Node Feature Engineering

- **Node Centrality: Eigenvector**
  Motivation: A node is important if surrounded by important neighbors

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$

λ is normalization constant (it will turn out to be the largest eigenvalue of A)

Node Feature Engineering

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

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

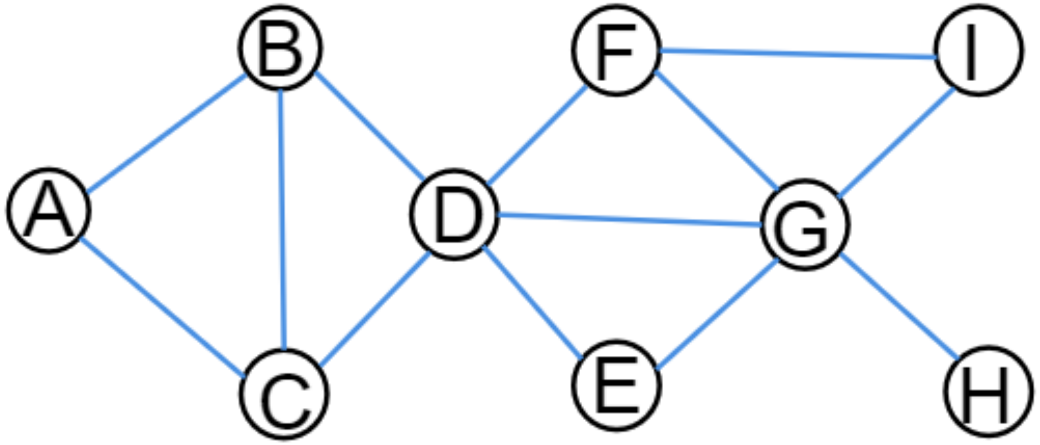
Node Feature Engineering

- **Node Centrality: Closeness**
  Motivation: A node is important if it has small shortest path lengths to all other nodes

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

Node Feature Engineering

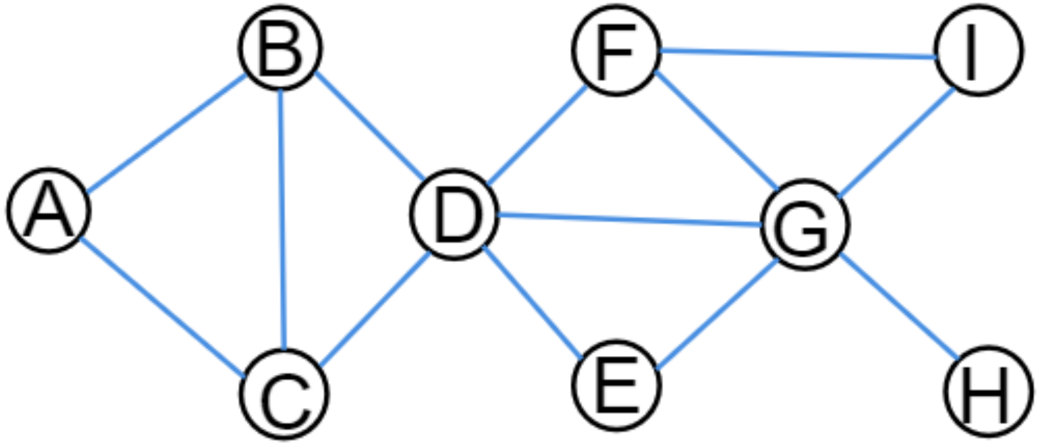
- **Exercise**
  - compute the clustering coefficients for nodes D and F



Now, you have 3 minutes to do Ex2.2.

Node Feature Engineering

- **Exercise**
  - compute the clustering coefficients for nodes D and F



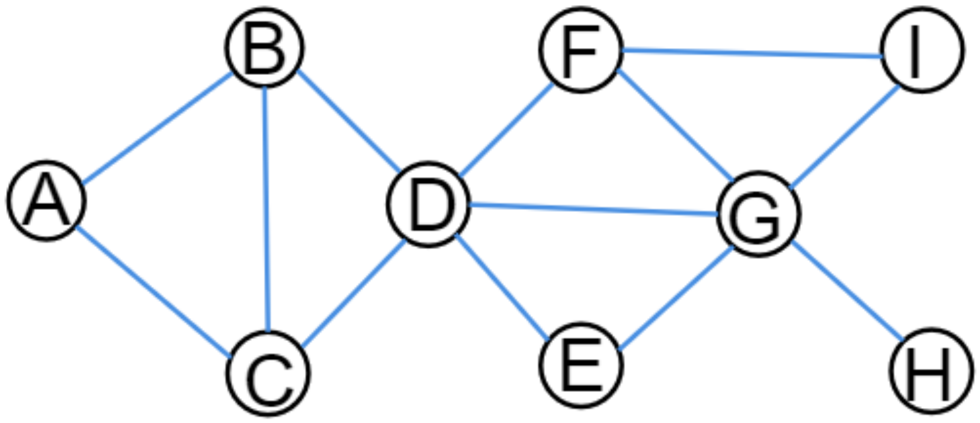
Answer:

D: 0.3

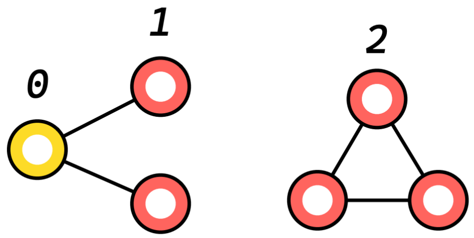
F: 2/3

Node Feature Engineering

- **Exercise**
  - compute the graphlet degree vector for nodes B and G.

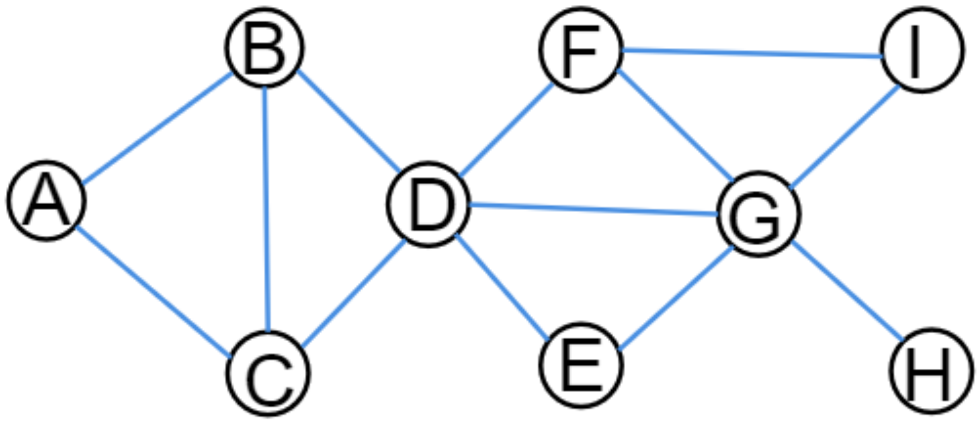


Now, you have 3 minutes to do Ex2.3.



Node Feature Engineering

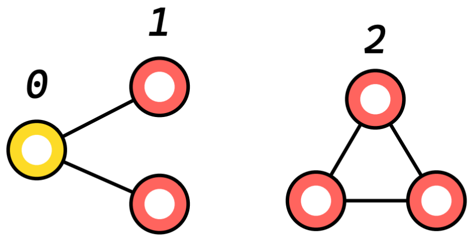
- **Exercise**
  - compute the graphlet degree vector for nodes B and G.



Answer:

B:[1,3,2]

G:[7,2,3]



Q & A