In this lab, we will sketch out the Girvan-Newman algorithm for community detection on undirected, weighted graphs. This algorithm comes from their 2002 PNAS paper:

Girvan and Newman. Community structure in social and biological networks. PNAS 2002.
Available on arXiv: `https://arxiv.org/pdf/cond-mat/0112110v1.pdf`.

Your task in HW6 is to implement this algorithm.
The steps are:

1. Calculate the *edge betweenness* of all edges in the network. Recall that the edge betweenness is defined as

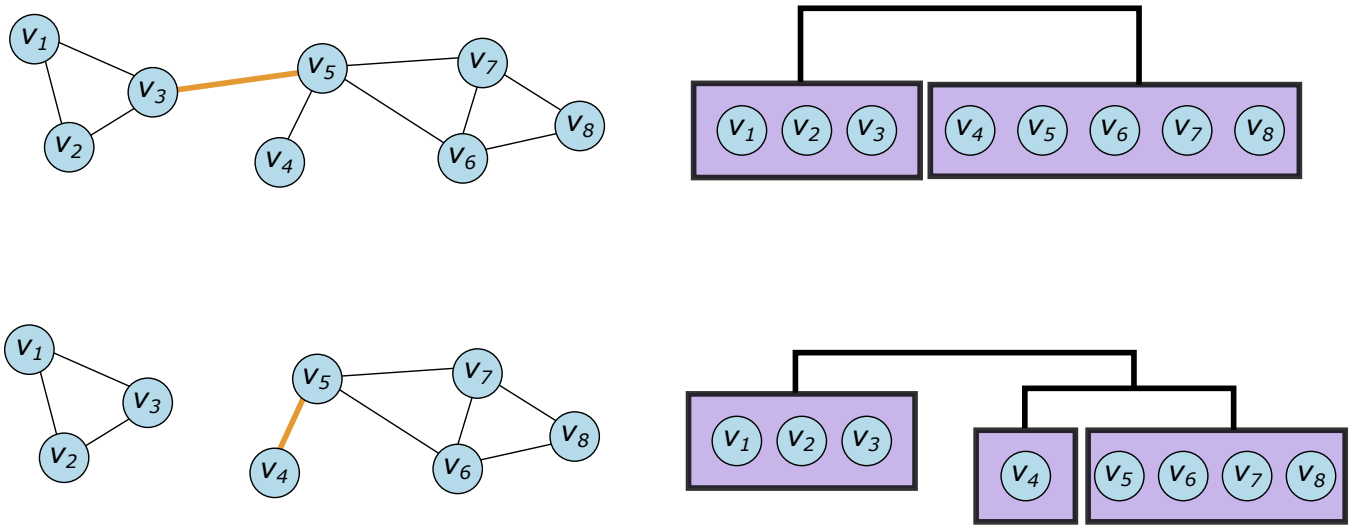
$$C_B(u,v) = \sum_{s,t \in V} \frac{\sigma(s,t|(u,v))}{\sigma(s,t)}, \tag{1}$$

   where $C_B((u,v))$ is the edge betweenness centrality of edge $(u,v) \in E$, $\sigma(s,t)$ is the number of shortest paths between $s$ and $t$, and $\sigma(s,t|(u,v))$ is the number of shortest paths between $s$ and $t$ that pass through $e$.

2. Remove the edge with the highest betweenness.

3. Recalculate betweenness for all edges affected by the removal.

4. Repeat from Step 2 until no edges remain.




The method should output a partition/grouping that increases the number of groups by one each time, e.g.,

```
[['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8']]
[['v1', 'v2', 'v3'], ['v4', 'v5', 'v6', 'v7', 'v8']]
[['v1', 'v2', 'v3'], ['v4'], ['v5', 'v6', 'v7', 'v8']]
[['v1', 'v2', 'v3'], ['v4'], ['v5'], ['v6', 'v7', 'v8']]
[['v1', 'v2'], ['v3'], ['v4'], ['v5'], ['v6', 'v7', 'v8']]
[['v1'], ['v2'], ['v3'], ['v4'], ['v5'], ['v6', 'v7', 'v8']]
[['v1'], ['v2'], ['v3'], ['v4'], ['v5'], ['v6'], ['v7', 'v8']]
[['v1'], ['v2'], ['v3'], ['v4'], ['v5'], ['v6'], ['v7'], ['v8']]
```

# 1 Pseudocode in Groups

In small groups, work through the following functions. We are intentionally starting at the "big picture" level and will work our way down to important functions and subroutines. Within each function, denote which parts need more detail.

`girvan_newman()`: *Identify communities using the Girvan-Newman algorithm.*
**Inputs:** Weighted, undirected graph $G = (V, E)$
**Returns:** The communities (groups of nodes) from $G$ for $k = 1, 2, \ldots, n$.

`edge_betweenness()`: *Write a function to calculate edge betweenness for an edge in the graph.*
**Inputs:** Undirected, weighted graph $G = (V, E)$.
**Returns:** The edge betweenness of every edge $(u, v) \in E$ as shown in Eq. (1).

`dijkstra_all()`: *Get all shortest paths from $s$ to all nodes.*
**Inputs:** Weighted, undirected graph $G = (V, E)$ and a source node $s \in V$
**Returns:** A dictionary $D$ of distances and $\pi$ of predecessors that capture all shortest paths.

The algorithm below is from HW5 - modify it to return all shortest paths.

---

`dijkstra`$(G = (V, E),\ s \in V)$
  $D[v] = \infty$ for all $v \in V$ # *Dictionary of distances*
  $D[s] = 0$
  $\pi[v] = \emptyset$ for all $v \in V$ # *Dictionary of predecessors*
  **for** $v \in V$ **do**
    Add $v$ to $Q$ with priority $D[v]$
  **end for**
  **while** $Q$ is nonempty **do**
    Remove $w$ from $Q$ with the minimum value
    **for** each $x \in N_w$ **do**
      **if** $D[x] > D[w] + c(w, x)$ **then**
        $D[x] = D[w] + c(w, x)$
        $\pi[x] = w$
        Update value $D[x]$ for node $x$ in $Q$
      **end if**
    **end for**
  **end while**
  **return** $D$,$\pi$

---

`get_paths()`: *Return all shortest paths between two nodes. (This is hard! Think recursively...)*
**Inputs:** Predecessor dictionary $\pi$ and target node $t \in V$
**Returns:** All paths that connect $s$ and $t$.

**Hand In**: *There is no handin - use these ideas to begin on HW6.*