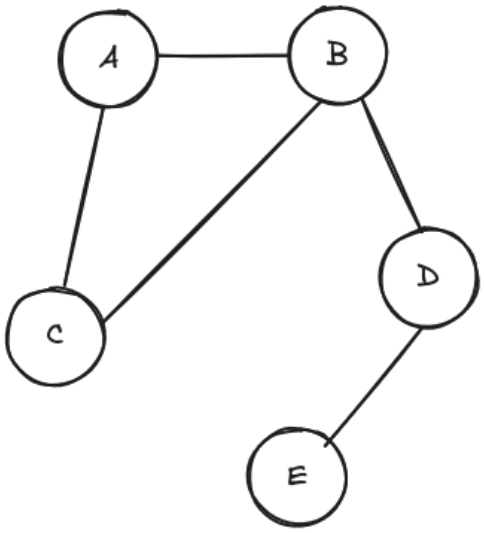
HW01 Graphs and Social Networks

Sam Ly

September 14, 2025

Q1 (5 pts)

1. Draw the graph.



2. Compute the degree of each node.

$$\deg(A) = 2$$
$$\deg(B) = 3$$
$$\deg(C) = 2$$
$$\deg(D) = 2$$
$$\deg(E) = 1$$

3. Verify the **Handshake Theorem**.

$$\sum_{v \in V} \deg(v) = 2|E|$$

$$2 + 3 + 2 + 2 + 1 = 10 = 2|E|$$

4. Explain briefly why this must always hold for undirected graphs.

   This must always hold true for undirected graphs because each edge always connects two nodes. This means those two nodes increase their degree by one when connected by one edge. By extension, each edge increases the sum of all node degrees.

   Therefore, the sum of all degrees of nodes is equal to two times the number of edges.

Q2 (5 pts)

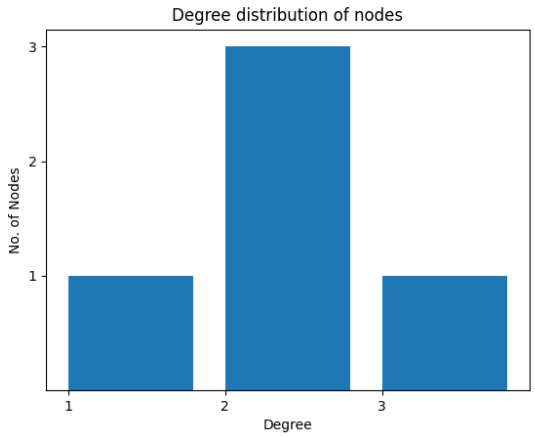
1. Write down the **degree distribution**: how many nodes have degree 1, 2, 3, etc.

Degree	No. of Nodes
1	1
2	3
3	1

2. Write a short **Python snippet** that:
   - Stores the graph as a dictionary of neighbors.
   - Loops over nodes to compute degrees.
   - Prints the degree distribution.

```python
from typing import Counter

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

graph = {v : [] for v in V}
for e in E:
    a, b = e
    graph[a].append(b)
    graph[b].append(a)

degrees = Counter(len(edges) for edges in graph.values())

for degree, count in degrees.items():
    print(f"{count} nodes have degree {degree}")

# 3 nodes have degree 2
# 1 nodes have degree 3
# 1 nodes have degree 1
```

3. Make a **bar chart** of the distribution (x = degree, y = count). Label axes and add a title.

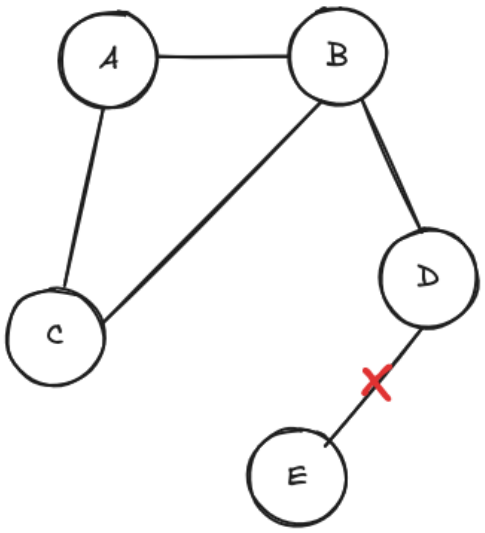


Q3 (5 pts)

1. List its connected components.

   All nodes are in one connected component, thus the only connected component is $\{A, B, C, D, E\}$

2. Suppose the edge **D**–**E** is removed. How many components now? Which nodes are in each?



There are now 2 connected components:

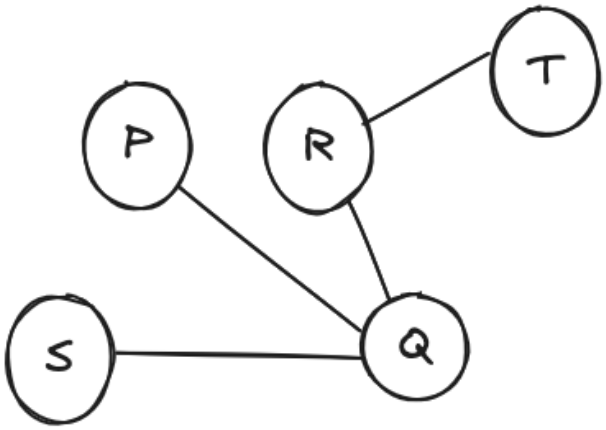
- $\{A, B, C, D\}$, 4 nodes.
- $\{E\}$, 1 node.

3. Is **D-E** a **bridge** (an edge whose removal increases the number of components)? Justify in one sentence.

D-E is a bridge because it is the only edge that connects node **E** to the rest of the graph, thus removing it makes node **E** a new component.

Q4 (5 pts)

Consider this undirected graph:
Nodes = $\{P, Q, R, S, T\}$
Edges = $\{P - Q, Q - R, Q - S, R - T\}$



1. Which node has the highest **degree centrality**?

Node Q has the highest degree centrality because it has the most edges connected to it, meaning it has the highest degree.

2. Which node likely has the highest **betweenness centrality**? Explain briefly.

   Node Q has the highest **betweenness centrality** because it is on the most shortest paths.

```python
from collections import Counter, defaultdict, deque


V = ["P", "Q", "R", "S", "T"]
E = {("P", "Q"), ("Q", "R"), ("Q", "S"), ("R", "T")}

graph = {v : [] for v in V}
for e in E:
    a, b = e
    graph[a].append(b)
    graph[b].append(a)

degrees = Counter(len(edges) for edges in graph.values())

def bfs(start, end) -> list[str]:
    frontier = deque([start])
    prev = {start: None}
    while frontier:
        curr = frontier.popleft()
        if curr == end:
            break

        for n in graph[curr]:
            if n in prev:
                continue
            prev[n] = curr
            frontier.append(n)

    if end not in prev:
        return []

    out = [end]
    curr = prev[end]
    while curr:
        out.append(curr)
        curr = prev.get(curr)

    out.reverse
    return out

shortest_paths = []
for i in range(len(V)-1):
    for j in range(i+1, len(V)):
        shortest_paths.extend(bfs(V[i], V[j]))

centrality = Counter(shortest_paths)
print(centrality)
# Counter({'Q': 9, 'R': 7, 'P': 4, 'S': 4, 'T': 4})
```

3. Which node is closest to all others (high **closeness centrality**)? Explain briefly.

   Node Q has the highest **closeness centrality** because it has the lowest "total distance" from the other nodes.

```python
distance = defaultdict(int)
for start in V:
    for end in V:
        if start == end:
            continue
        distance[start] += (len(bfs(start, end)) - 1)
print(distance)
print(sorted([(d, v) for v, d in distance.items()]))
# defaultdict(<class 'int'>, {'P': 8, 'Q': 5, 'R': 6, 'S': 8, 'T': 9})
# [(5, 'Q'), (6, 'R'), (8, 'P'), (8, 'S'), (9, 'T')]
```