Graph theory reharsal

Davide Mottin

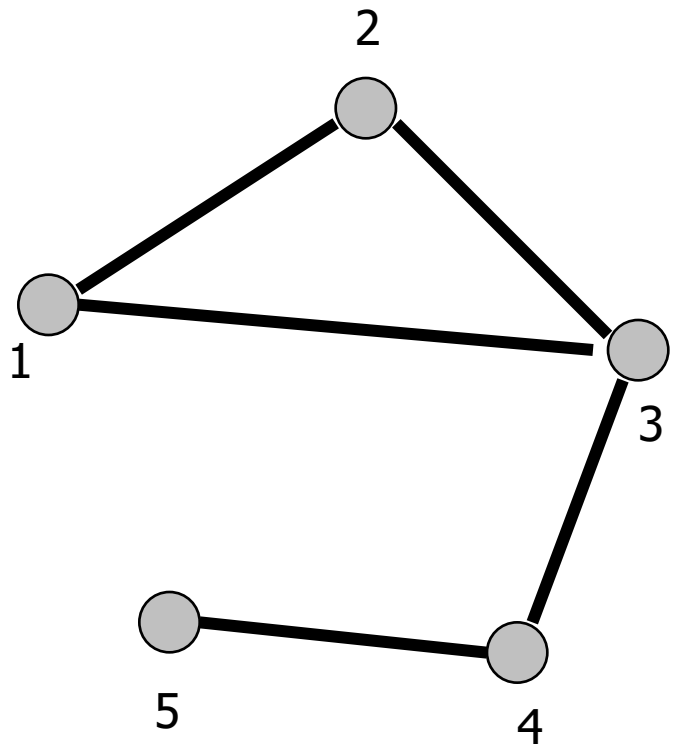
Hasso Plattner Institute

Graph Mining course Winter Semester 2016

(slides adapted from: http://www.cs.uoi.gr/~tsap/teaching/cs-l14/)

Undirected graph

- Graph G=(V,E)
  - V = set of vertices (nodes)
  - E = set of edges



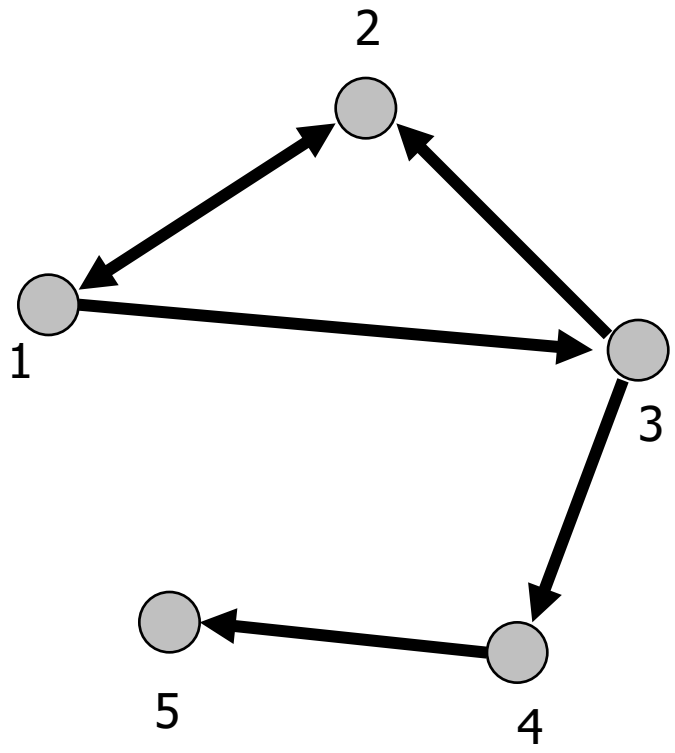
undirected graph

V = {1, 2, 3, 4, 5}

E={(1,2),(1,3),(2,3),(3,4),(4,5)}

Directed graph

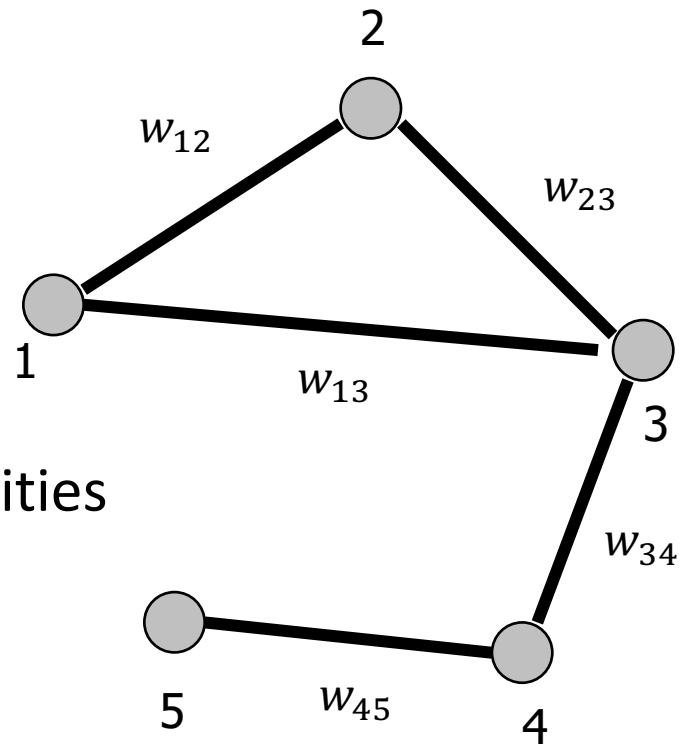
- Graph G=(V,E)
  - V = set of vertices (nodes)
  - E = set of edges



directed graph
V = {1, 2, 3, 4, 5}
E={‹1,2›, ‹2,1› ‹1,3›, ‹3,2›, ‹3,4›, ‹4,5›}

Weighted graph

- Graph G=(V,E)
  - V = set of vertices (nodes)
  - E = set of edges and their weights



Weights can be either distances or similarities

Weighted graph

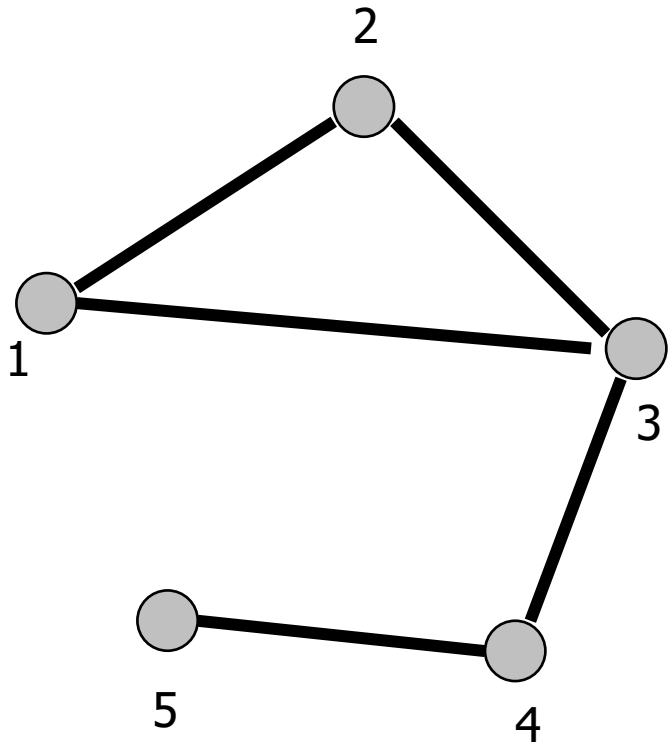
V = {1, 2, 3, 4, 5}

E={(1,2,w_{12}),(1,3, w_{12}),(2,3, w_{12}),(3,4, w_{12}),(4,5, w_{12})}

Graph Representation

- Adjacency Matrix
  - symmetric matrix for undirected graphs

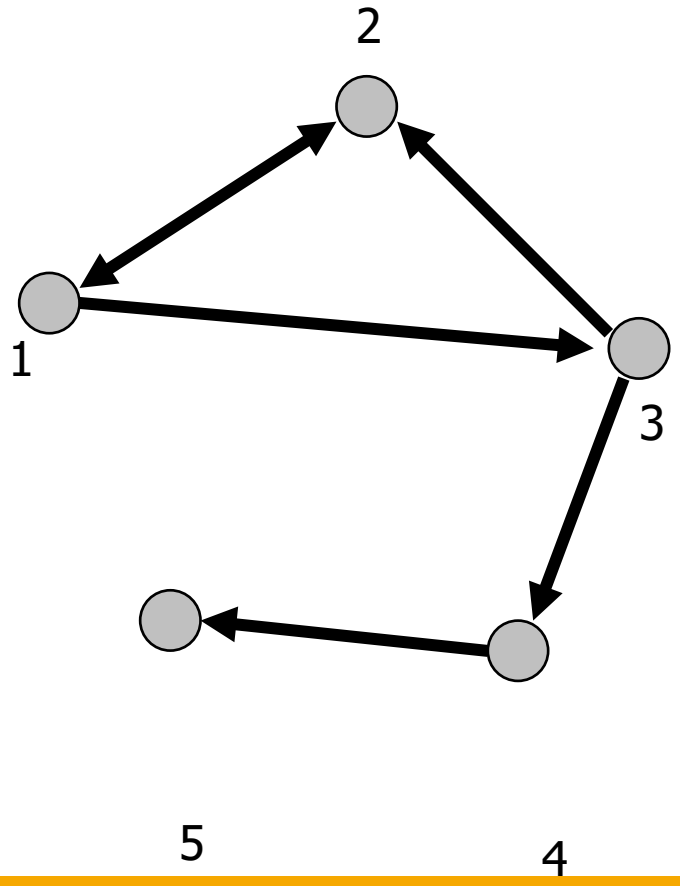
$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Graph Representation

- Adjacency Matrix
  - unsymmetric matrix for undirected graphs

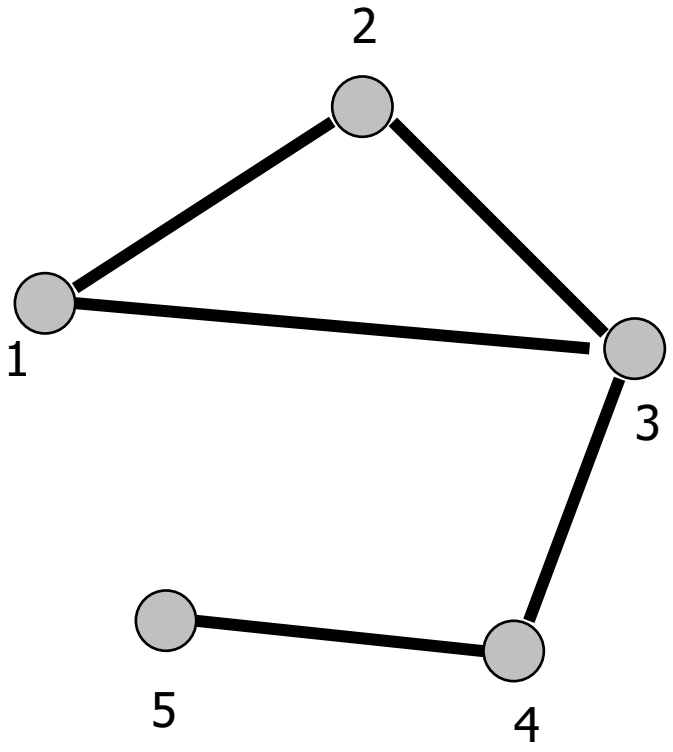
$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Graph Representation

- Adjacency List
  - For each node keep a list with neighboring nodes

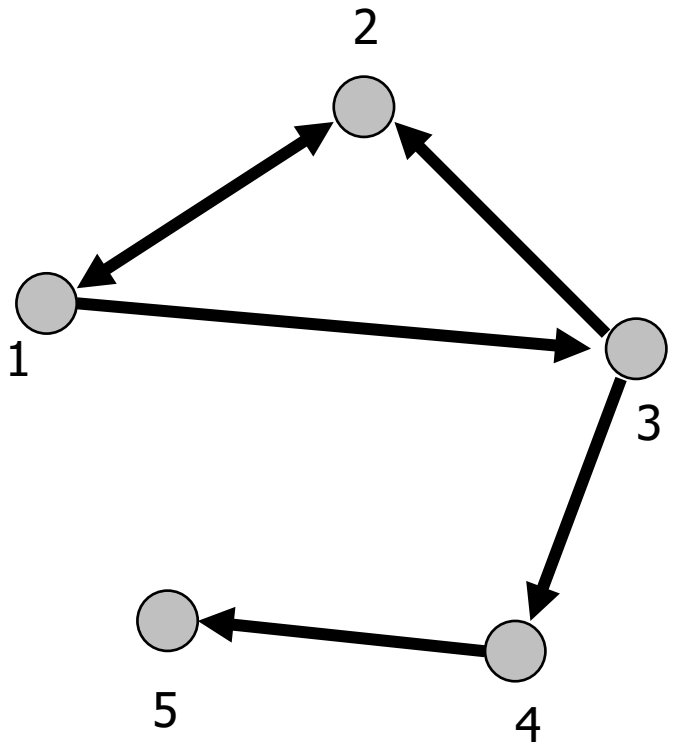
1: [2, 3]
2: [1, 3]
3: [1, 2, 4]
4: [3, 5]
5: [4]



Graph Representation

- Adjacency List
  - For each node keep a list of the nodes it points to

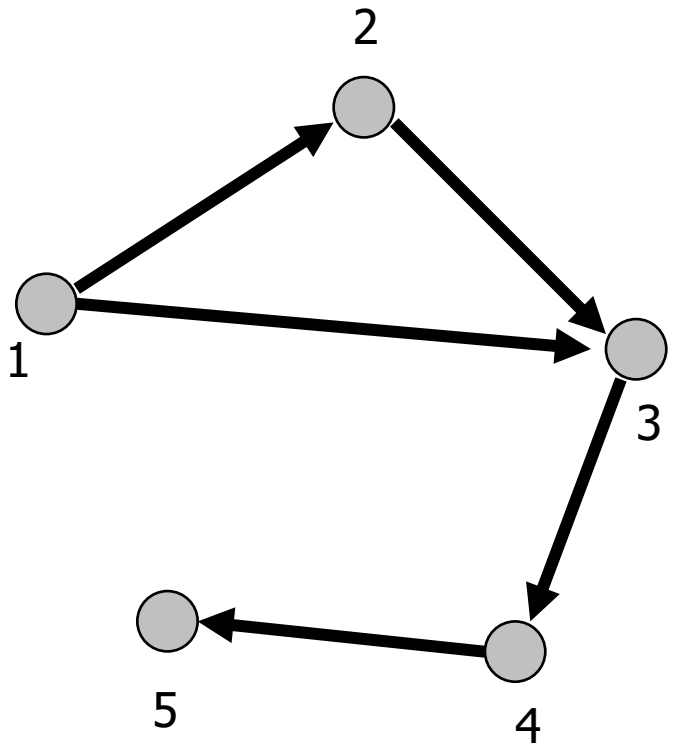
1: [2, 3]
2: [1]
3: [2, 4]
4: [5]
5: []



Graph Representation

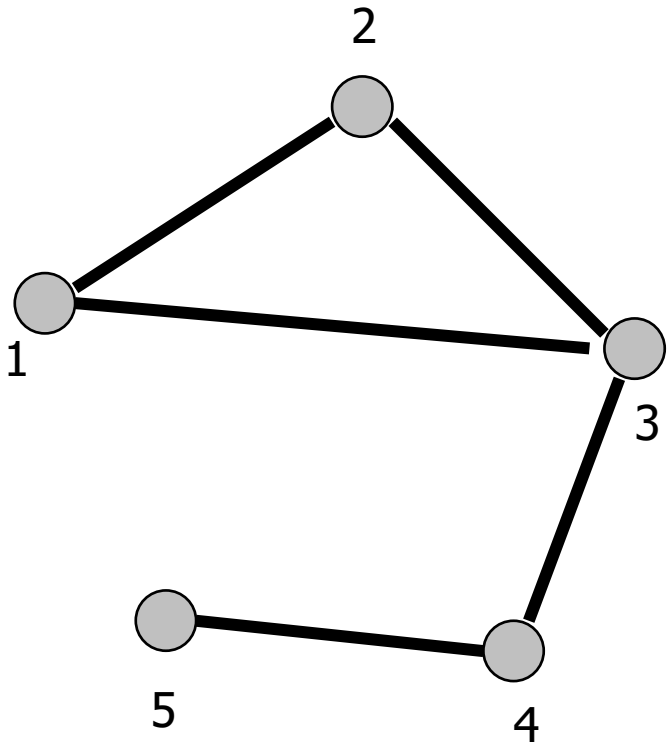
- List of edges
  - Keep a list of all (directed) the edges in the graph

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



Neighborhood and degree

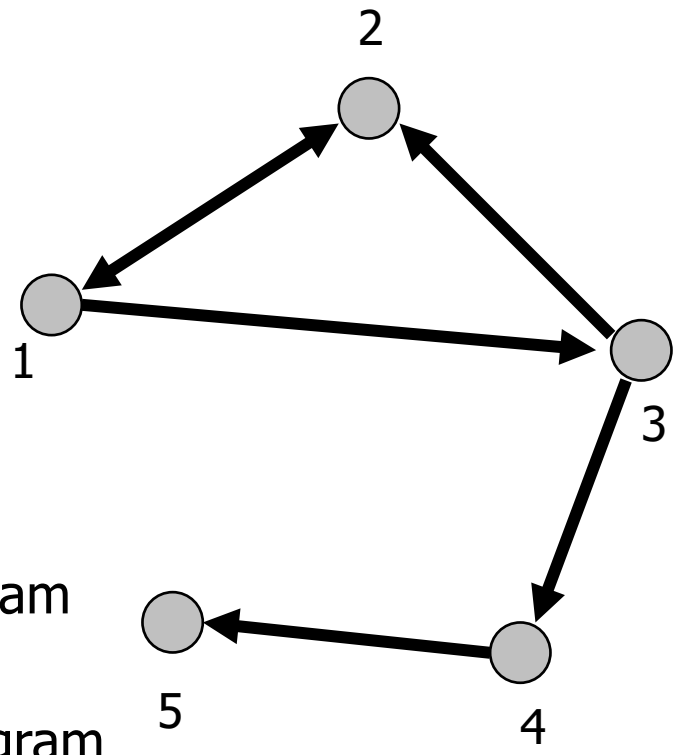
- Neighborhood N(v) of node v
  - Set of nodes adjacent to v
- degree d(v) of node v
  - Size of N(v), i.e. |N(v)|
  - number of edges incident on v
- degree sequence $\langle d(i), i = 1..5 \rangle$
  - $\langle 2,2,3,2,1 \rangle$
- degree histogram $\langle (1,1), (2,3), (3,1) \rangle$



Degree distribution= Values divided by the sum of the degrees

Degree (directed graphs)

- in-degree $d_{in}(i)$ of node i
  - number of edges incoming to node i
- out-degree $d_{out}(i)$ of node i
  - number of edges leaving node i



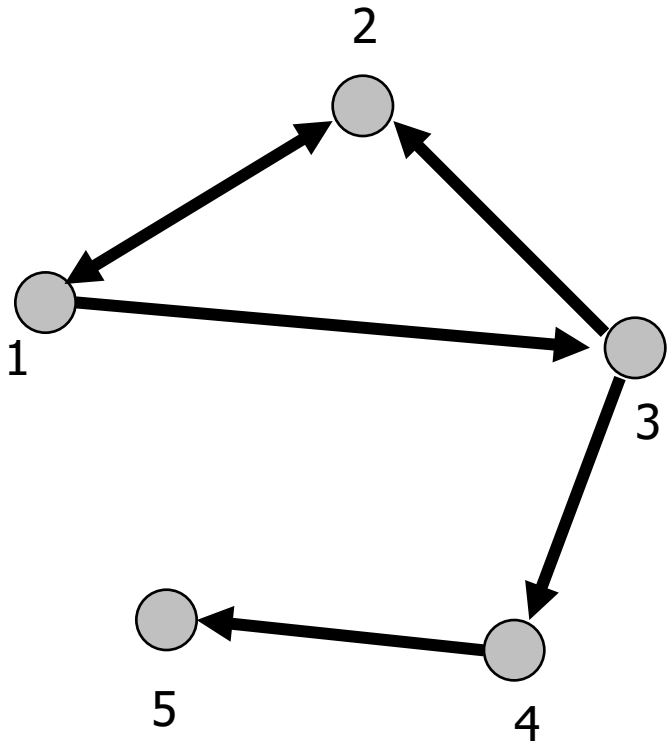
- in-degree sequence
  - [1,2,1,1,1]
- out-degree sequence
  - [2,1,2,1,0]
- in-degree histogram
  - [(1:3),(2:1)]
- out-degree histogram
  - [(0:1),(1:2),(2:2)]

Graph Traversals or Visits

- **Traversal:** strategy of visiting the nodes in a graph

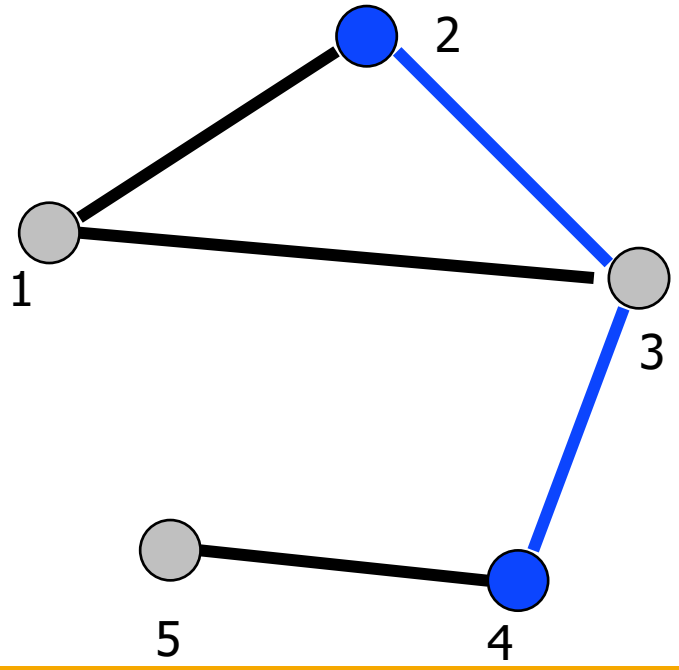
Breadth First Search (BFS)
1. Pick a node v
2. Visit (print) all the neighbors $v_1 \in N(V)$
3. Repeat 2 for all $v_1 \in N(V)$ until no node can be visited

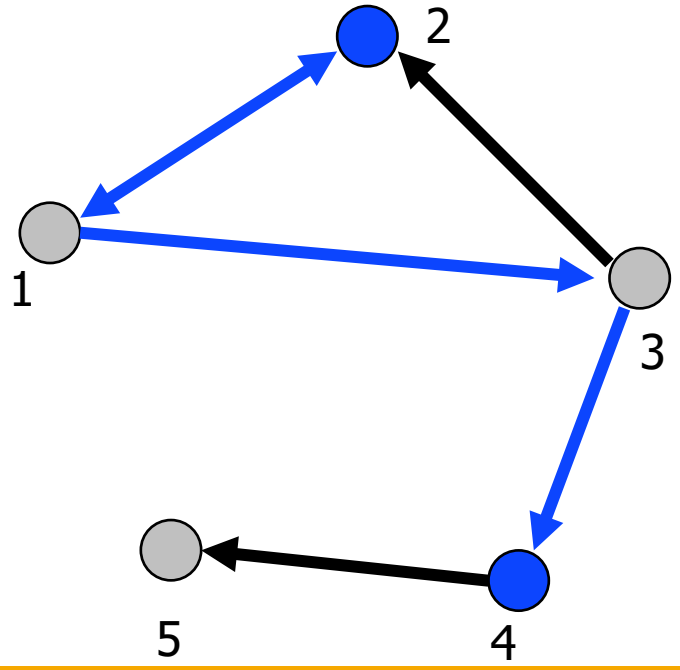
Depth First Search (DFS)
1. Pick a node v
2. Visit (print) **one** neighbors $v_1 \in N(V)$ before visiting the other neighbors
3. Repeat 2 for all $v_1 \in N(V)$ until no



Paths

- Path from node u to node v: a sequence of edges (directed or undirected) from node u to node v
  - path length: number of edges on the path
  - nodes u and v are connected
- Shortest path between u and v: path with minimum length among all the paths between u and v
- **Cycle:** a path that starts and ends at the same node



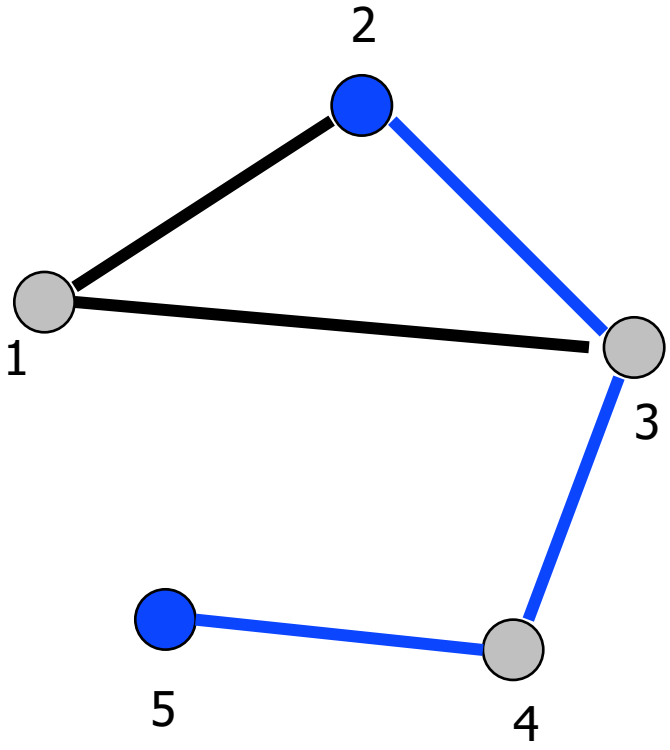


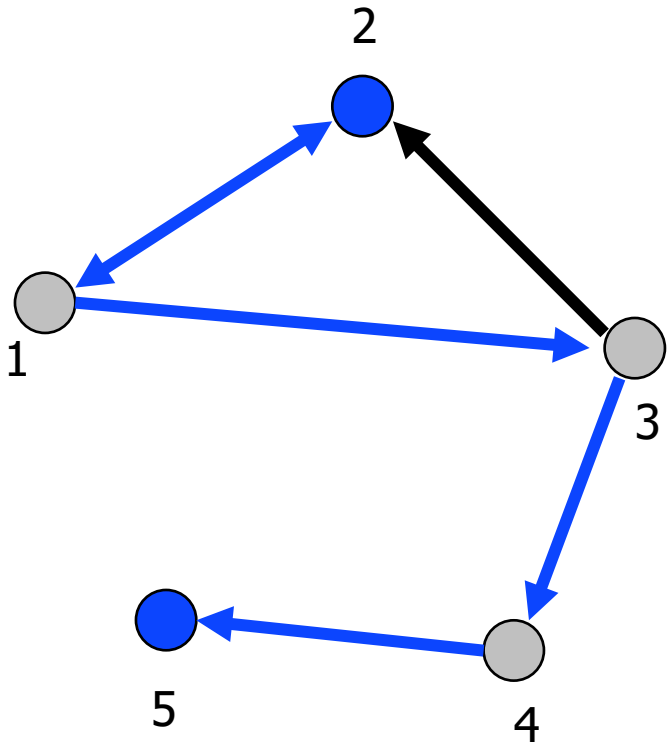
Shortest paths on weighted graphs

- Shortest paths on weighted graphs are harder to construct
  - There are several well known algorithms for finding single-source, or all-pairs shortest paths
  - For example: Dijkstra's Algorithm

Diameter

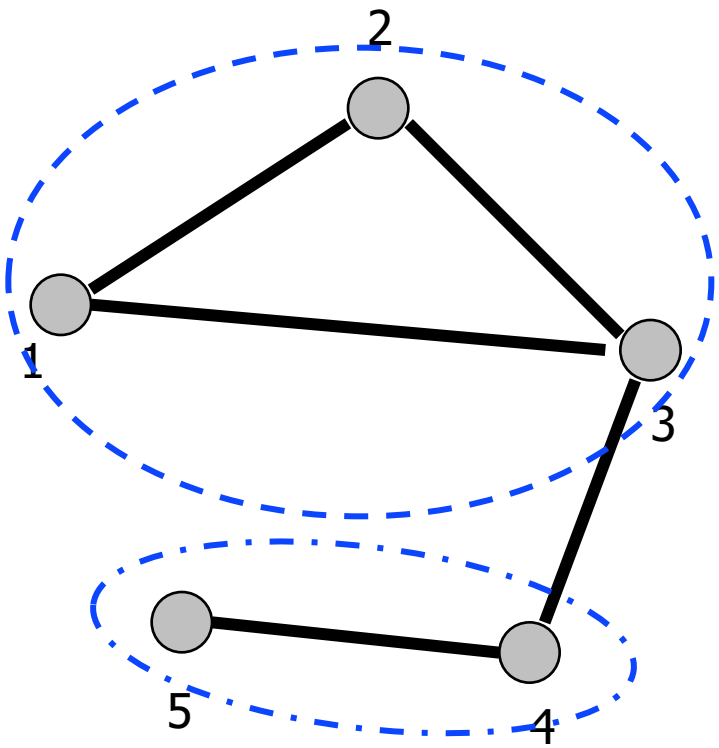
- The longest shortest path in the graph





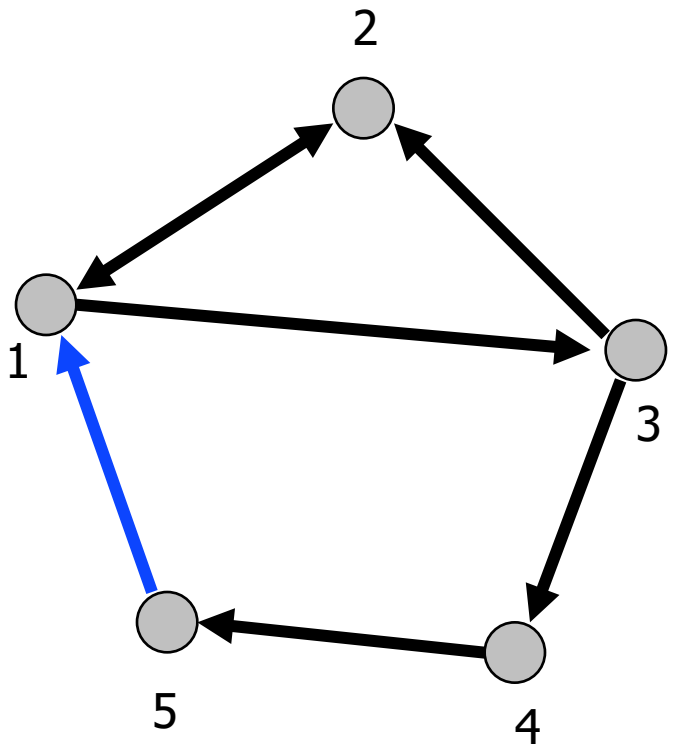
Connected graph

- Connected graph: a graph where there every pair of nodes is connected.
  - i.e., there is a path between any pair of node
- Disconnected graph: a graph that is not connected
- Connected Components: subsets of vertices that are connected



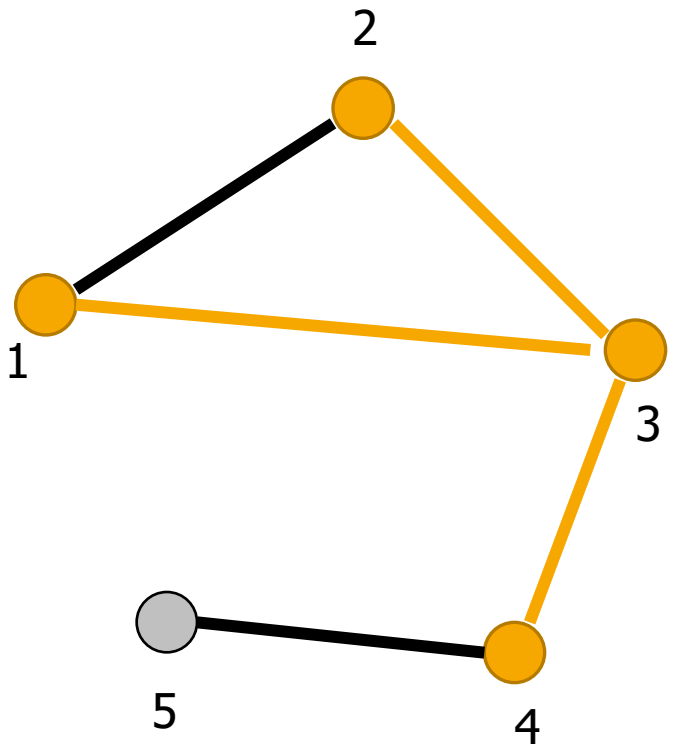
Connected Graph (directed)

- Strongly connected graph: there exists a path from every i to every j
- Weakly connected graph: If edges are made to be undirected the graph is connected



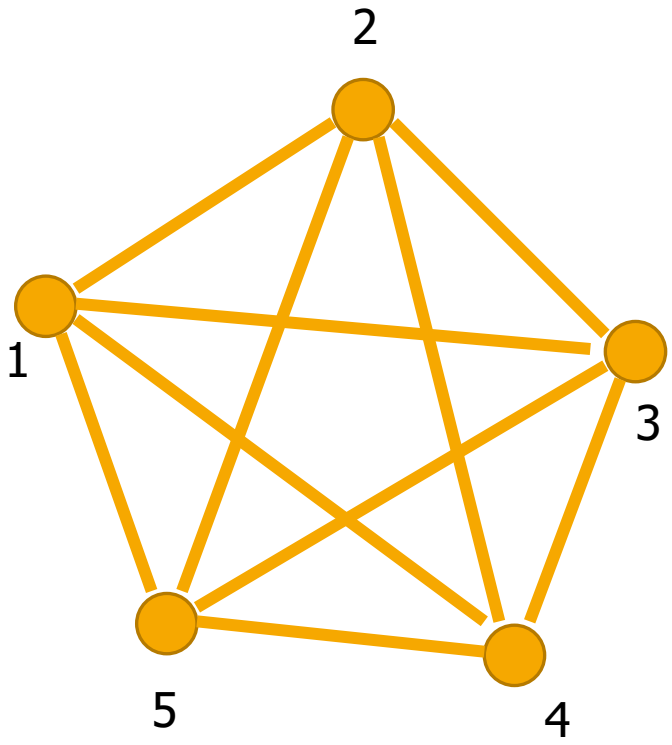
Subgraphs

- Subgraph: Given $V' \subseteq V$, and $E' \subseteq E$, the graph $G'=(V',E')$ is a subgraph of G.
- Induced subgraph: Given $V' \subseteq V$, let $E' \subseteq E$ is the set of all edges between the nodes in V'. The graph $G'=(V',E')$, is an induced subgraph of G



Fully connected (sub)-graphs (cliques)

- Clique K_n
- A graph that has all possible $\binom{n}{2} = \frac{n(n-1)}{2}$ edges



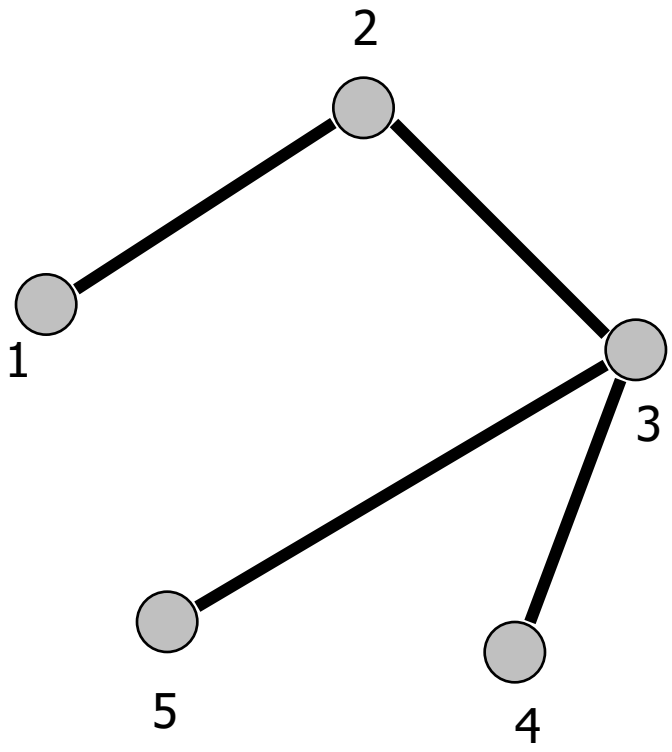
K_5 clique

K_4 clique

...

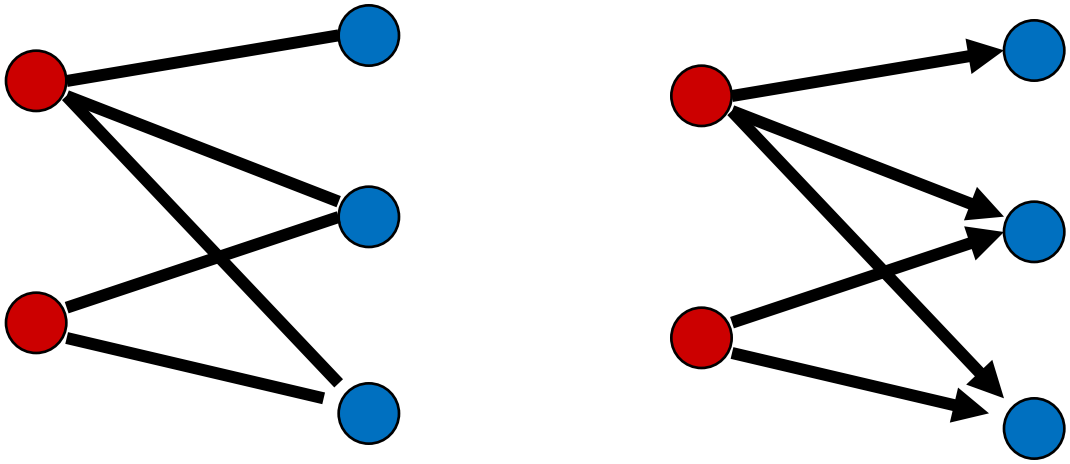
Trees

- Connected Undirected graphs without cycles



Bipartite graphs

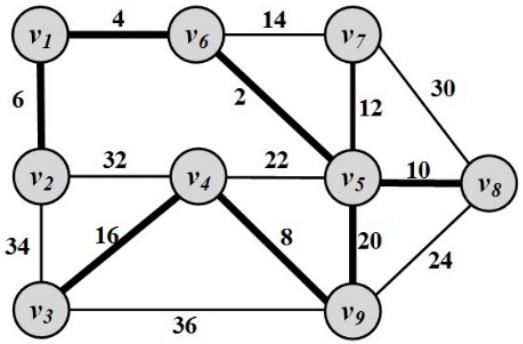
- Graphs where the set of nodes V can be partitioned into two sets L and R, such that there are edges only between nodes in L and R, and there is no edge within L or R



Spanning Tree

- For any connected graph, the spanning tree is a subgraph and a tree that includes all the nodes of the graph
- There may exist multiple spanning trees for a graph.
- For a weighted graph and one of its spanning tree, the weight of that spanning tree is the summation of the edge weights in the tree.
- Among the many spanning trees found for a weighted graph, **the one with the minimum weight** is called the

minimum spanning tree (MST)



P and NP

- **P**: the class of problems that can be solved in polynomial time
- **NP**: the class of problems that can be verified in polynomial time, but there is no known solution in polynomial time
- **NP**-hard: problems that are at least as hard as any problem in **NP**
- **NP**-complete: a problem is NP-complete iff
  - Is NP-hard
  - Is in NP (i.e., can be verified in polynomial time)

- Other complexity classes we might hear of
  - #P-complete (counting problems), PSPACE (polynomial space Turing Machine), APX-hard (problems hard to approximate) …


ANY
QUESTIONS
?