Graphs: an introduction

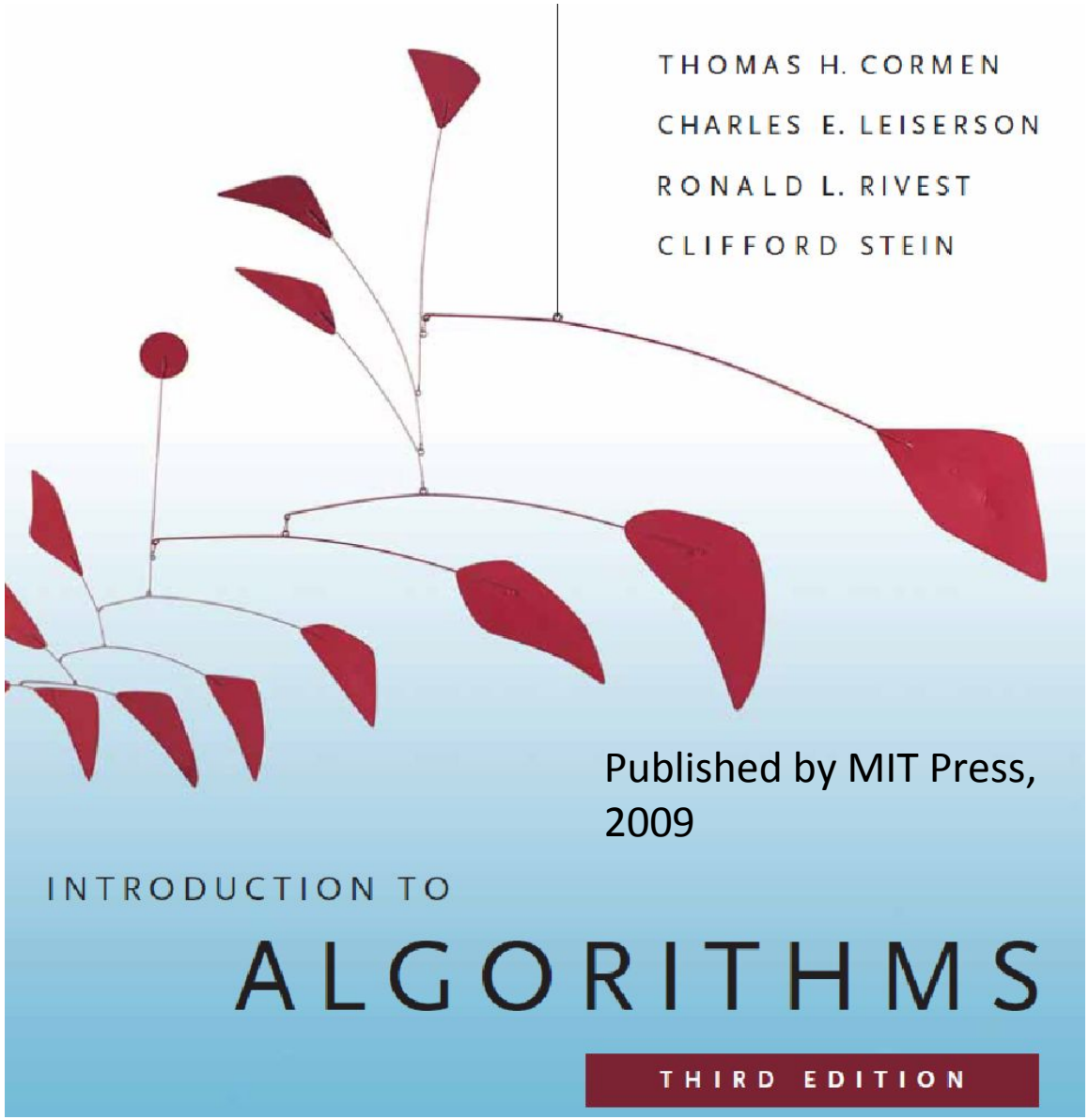
Bijendra Nath Jain

bnjain@iiitd.ac.in

Outline of next 9 lectures

- Graphs:
  - Undirected graphs
  - Directed graphs
  - (Directed) acyclic graphs (or DAGs)
  - Sparse graphs
  - Weighted graphs
- Graph applications
- Representation of graphs:
  - Adjacency matrix
  - Linked lists
- Algorithms:
  - Traversal algorithms:
    - BFS
    - DFS
  - Topological sort
  - Minimum spanning trees
  - Dijkstra's Shortest path
    - One-to-one
    - One-to-many
    - Many-to-many

Text to be used for next 9 lectures

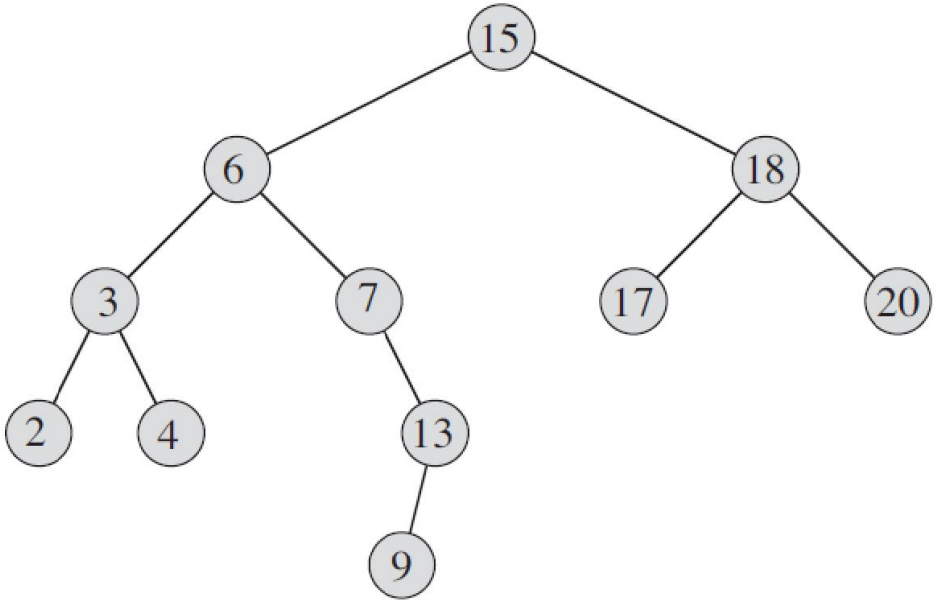


Outline

- Graphs:
  - Undirected graphs
  - Directed graphs
  - (Directed) acyclic graphs (or DAGs)
  - Sparse graphs
  - Weighted graphs
- Graph applications
- Representation of graphs:
  - Adjacency matrix
  - Linked lists
- Algorithms:
  - Traversal algorithms:
    - BFS
    - DFS
  - Topological sort
  - Minimum spanning trees
  - Dijkstra's Shortest path
    - One-to-one
    - One-to-many
    - Many-to-many

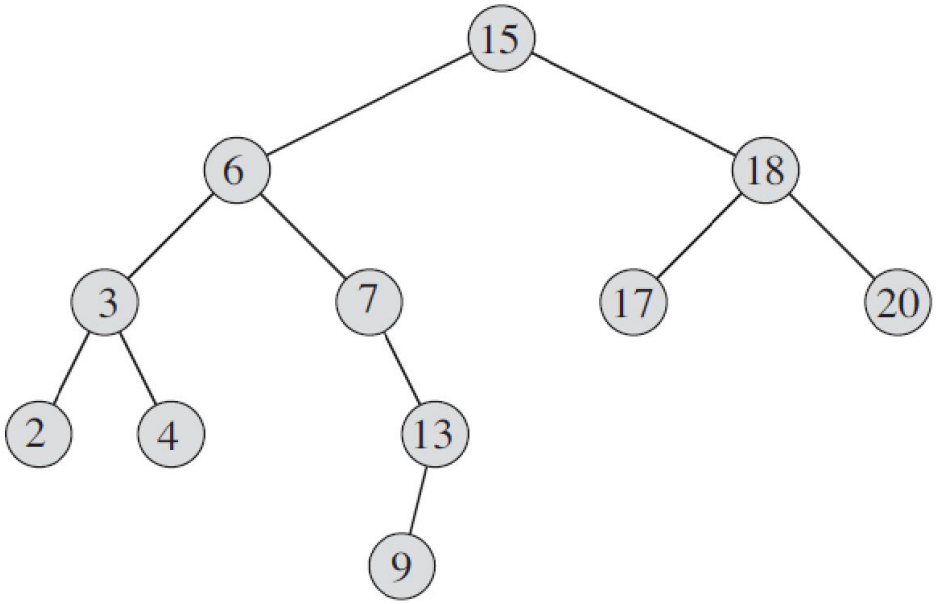
Some graph applications

- You are already familiar with trees, binary trees, and binary search trees
  - They are rooted
  - They are also graphs
- When looked upon as a graph, each edge is ‘directed’. That is, each edge identifies a pair of <parent node, child node>
- These graphs are special their representation is special, and algorithms are specific to search



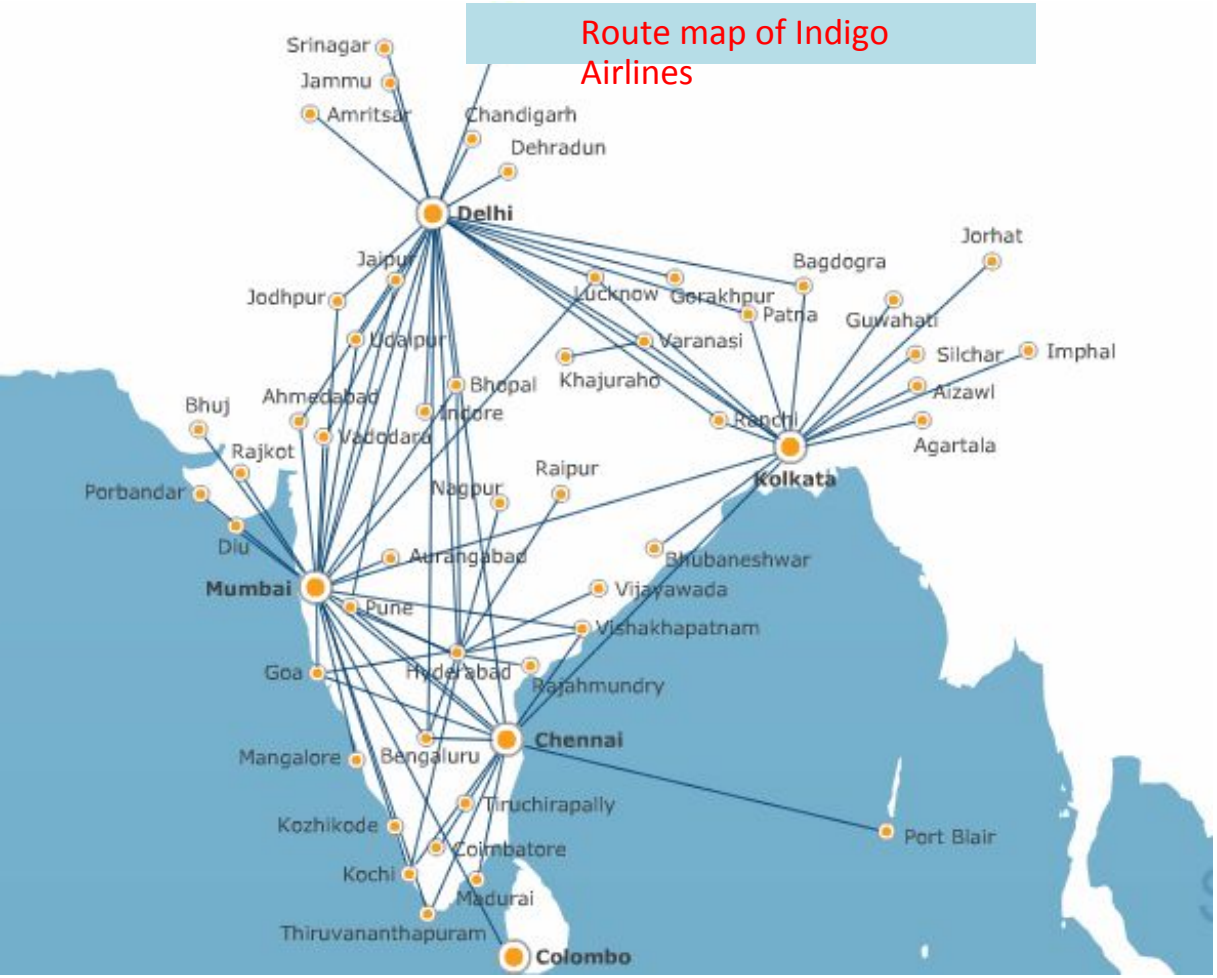
Some graph applications

- You are already familiar with trees, binary trees, and binary search trees
  - They are rooted
  - They are also graphs
- When looked upon as a graph, each edge is ‘directed’. That is, each edge identifies a pair of <parent node, child node>
- These graphs are special their representation is special, and algorithms are specific to search



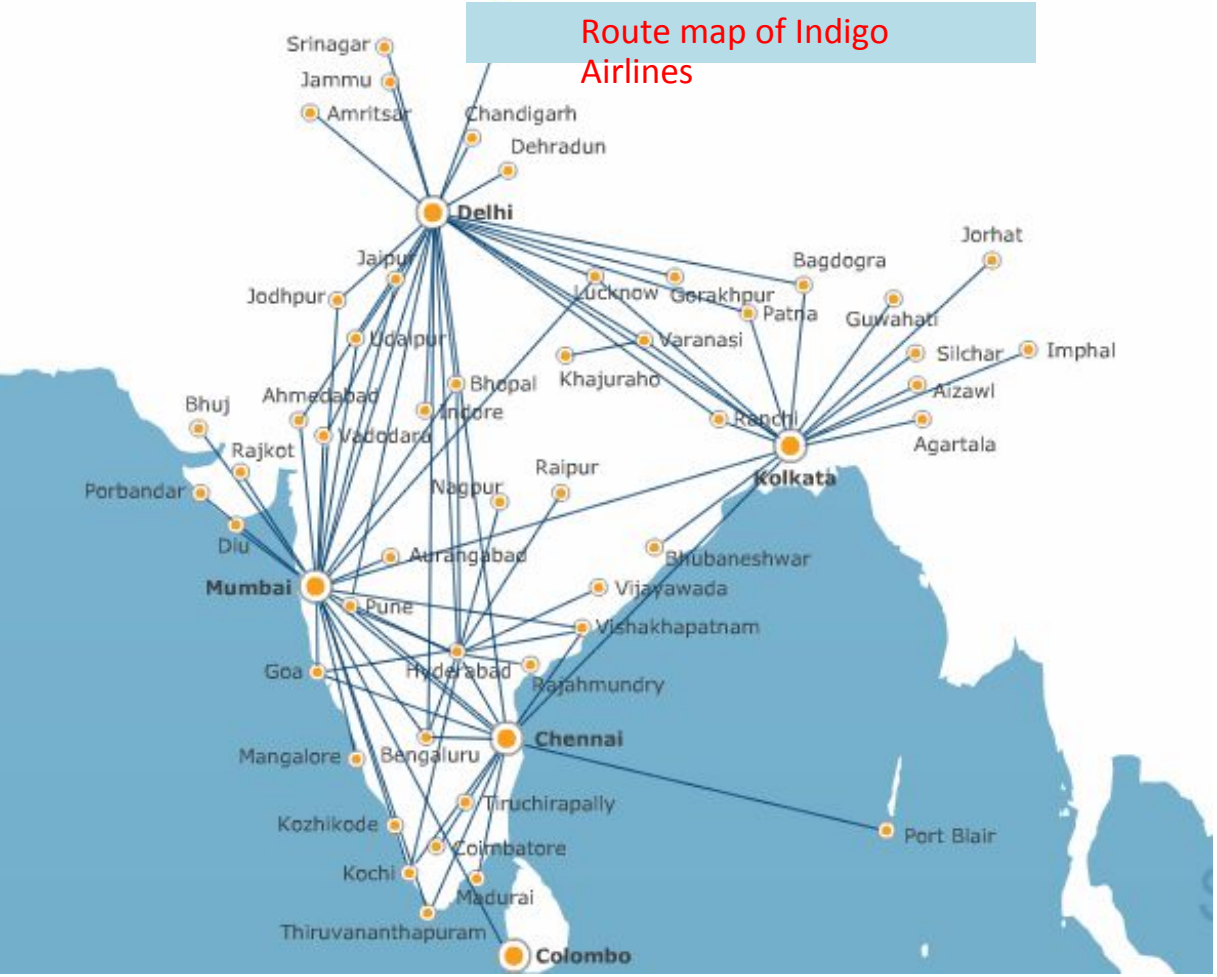
Some graph applications

- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990



Some graph applications

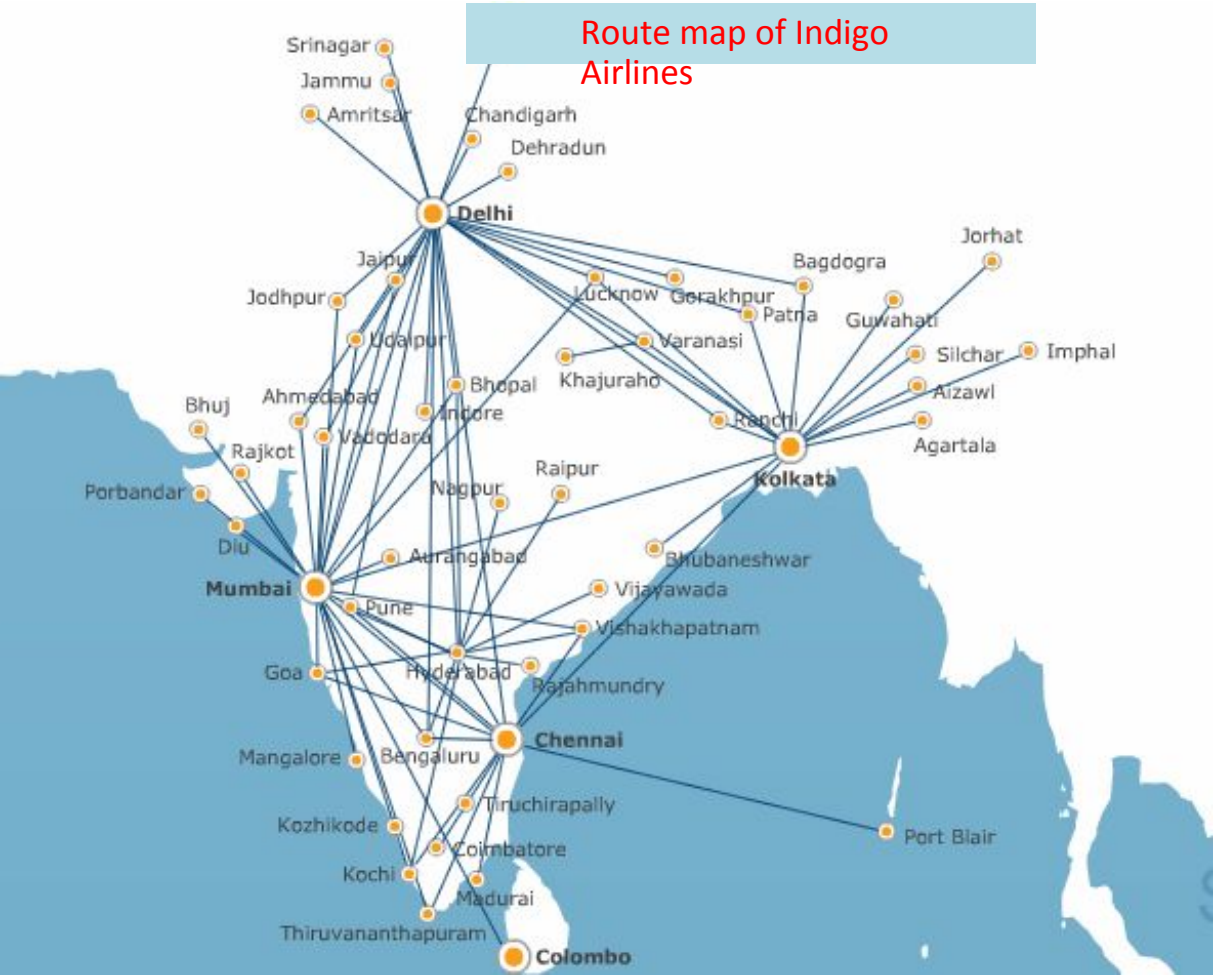
- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990



Reflect upon what problems would an airline attempt to solve

Some graph applications

- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990



Reflect upon what problems would an airline attempt to solve

Describe at least one problem that an airline may wish to solve, but do so in some detail

Some graph applications

- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990

Some graph applications

- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990

Reflect upon what problems would an highway planner attempt to solve

Some graph applications

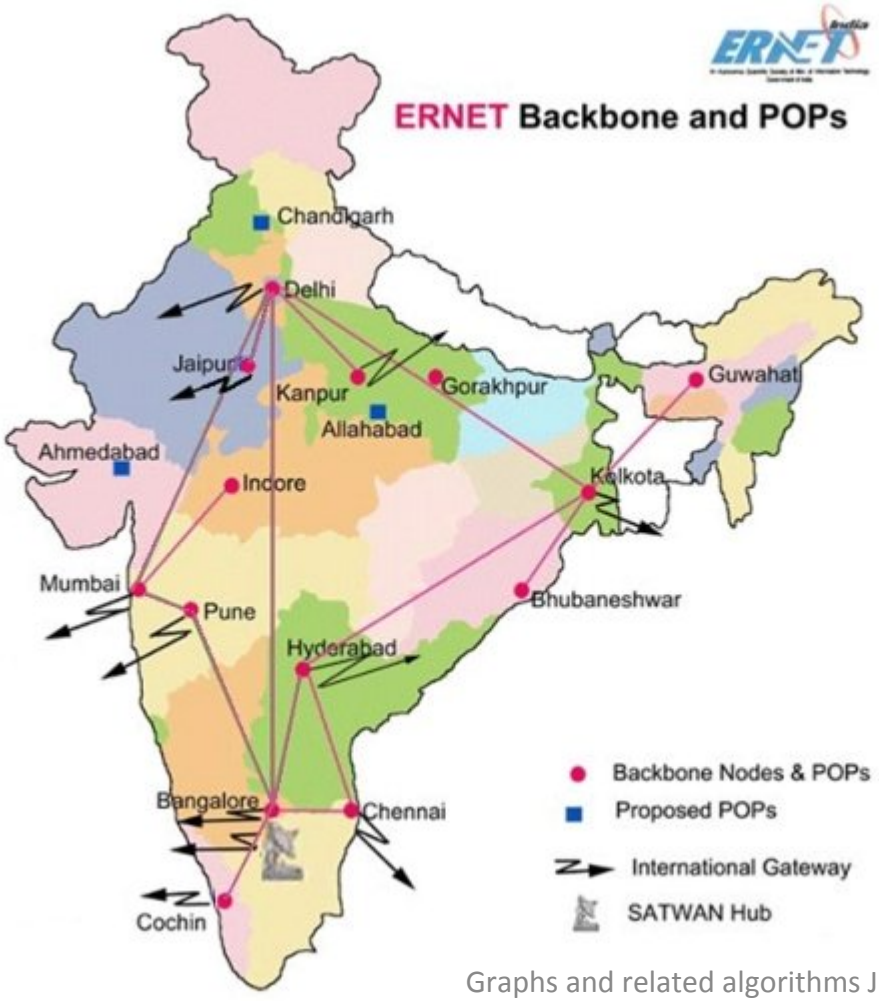
- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990

Reflect upon what problems would an highway planner attempt to solve

Describe at least one problem that an a highway authority may wish to solve, but do so in some detail

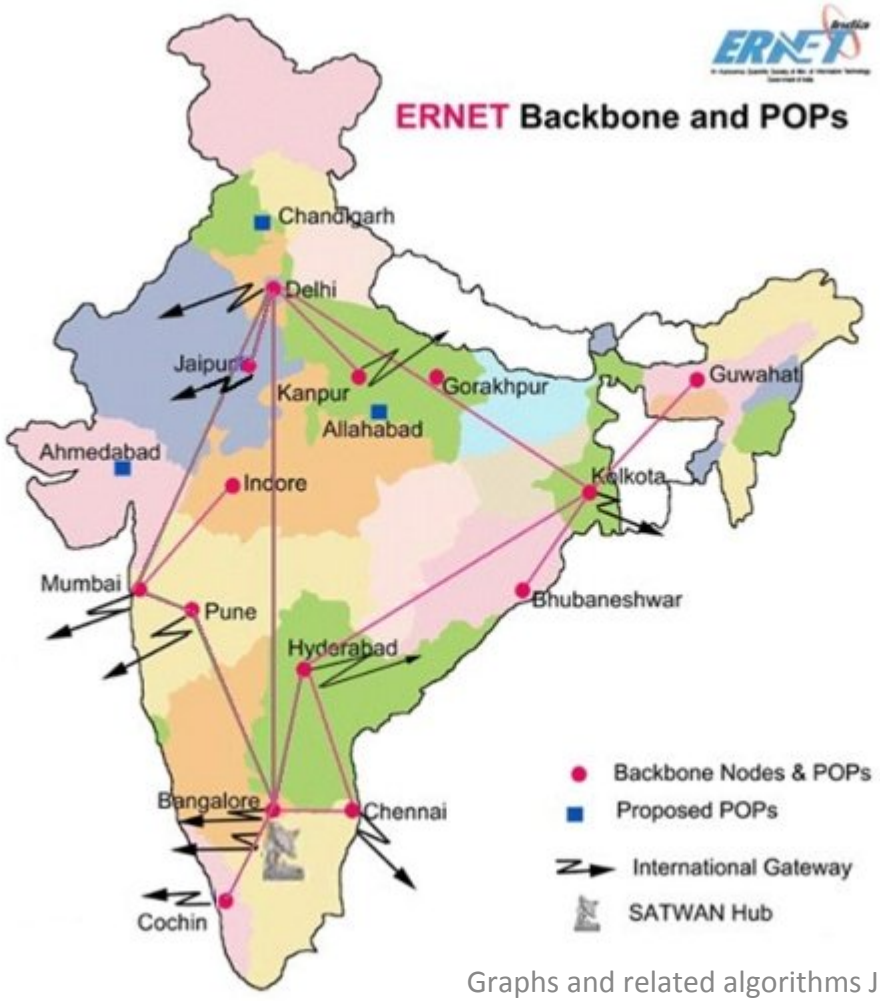
Some graph applications

- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990



Some graph applications

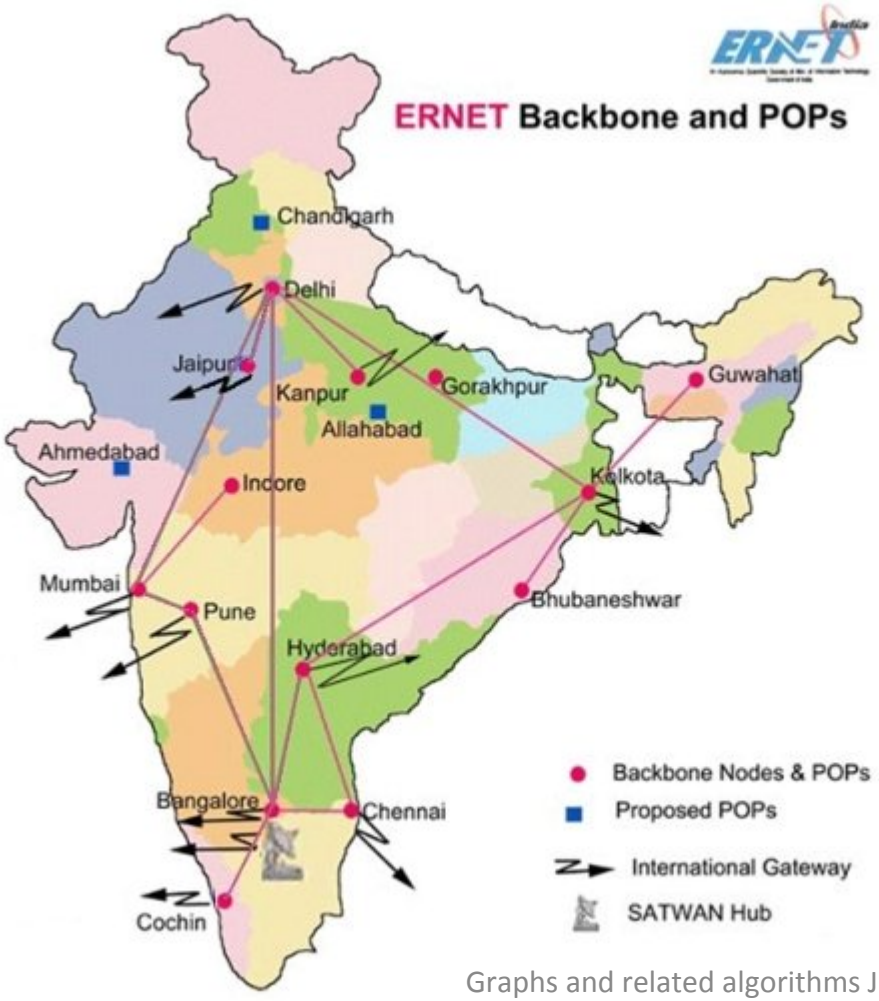
- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990



Reflect upon as to how network topology impacts throughput and latency in data networks

Some graph applications

- Maps:
  - Route map of an airline
  - Maps of highways in India
  - Internet: ERNet, in 1990

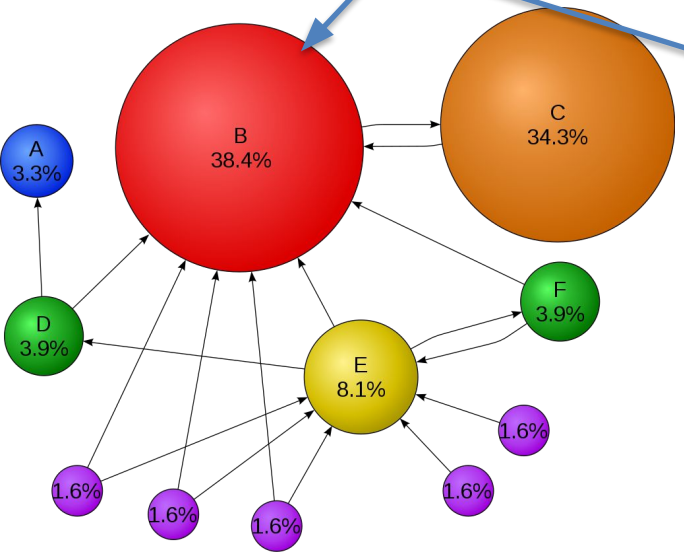


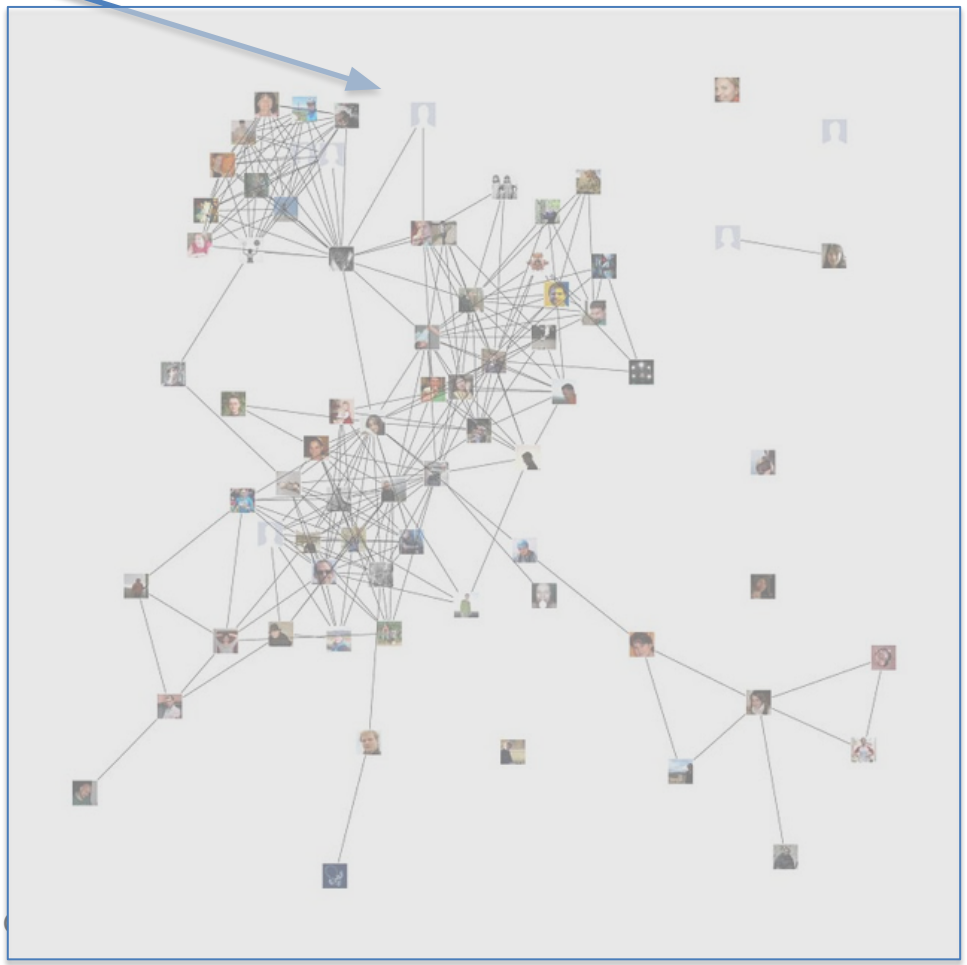
Reflect upon as to how network topology impacts throughput and latency in data networks

Describe at least one problem that an a network service provider may wish to solve, but do so in some detail

Some more graph applications

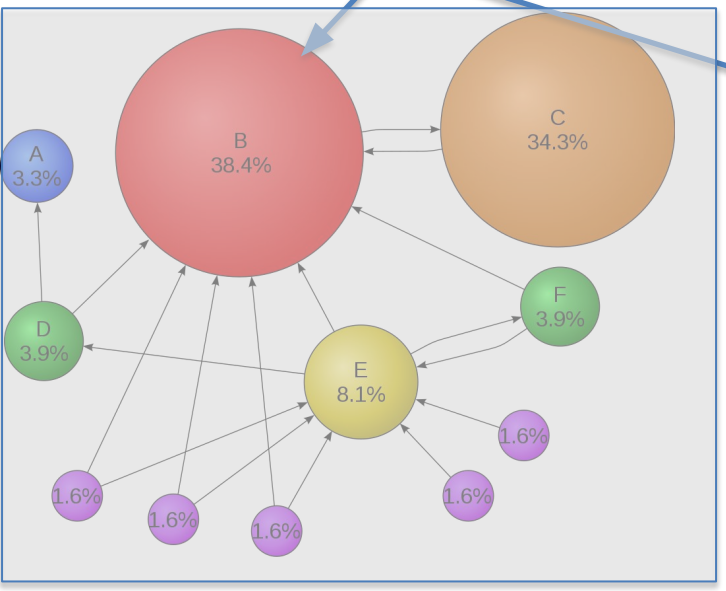
- More examples:
  - Web pages, and hyperlinks (see also https://en.wikipedia.org/wiki/PageRank)
  - Friends on social media
  - Etc.

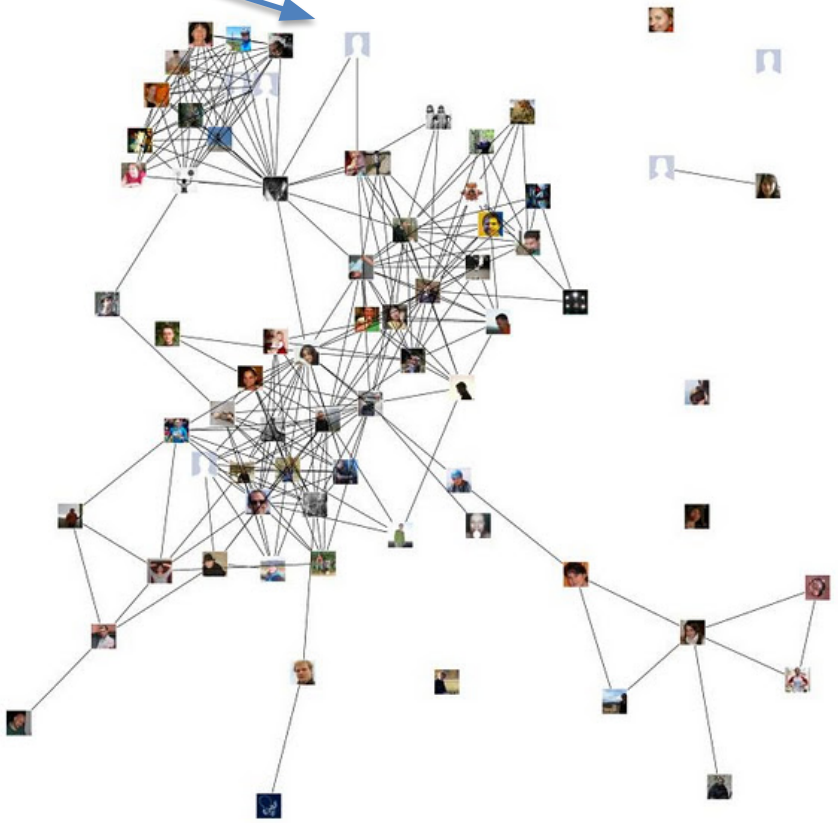




Some more graph applications

- More examples:
  - Web pages, and hyperlinks (see also https://en.wikipedia.org/wiki/PageRank)
  - Friends on social media
  - Etc.





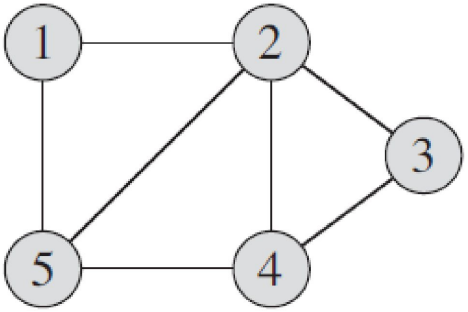
Some more graph applications

- Write a paragraph or two (with images) for 4 other graph applications, while identifying at least one problem that is IMP enough to be solved

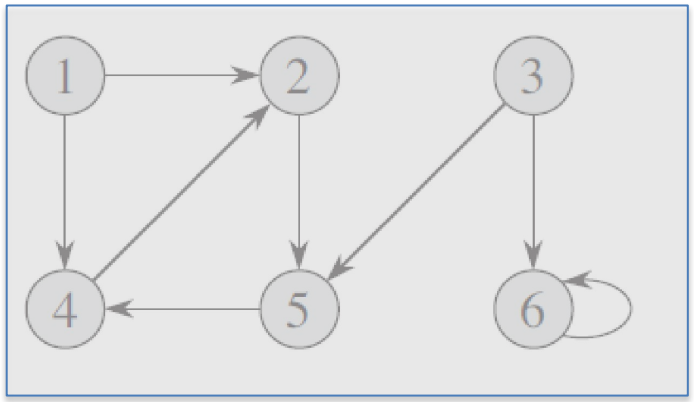
What is a graph?

- A graph, G = (V, E), consists of:
  - Set of vertices, V
  - Set of edges, E, where edges are either ‘undirected’ or ‘directed’

- Undirected graph:



- Directed graph:

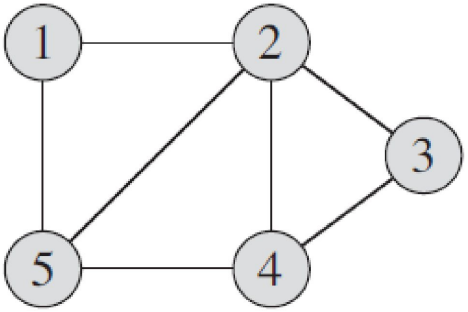


What is a graph?

- A graph, G = (V, E), consists of:
  - Set of vertices, V
  - Set of edges, E, where edges are either 'undirected' or 'directed'

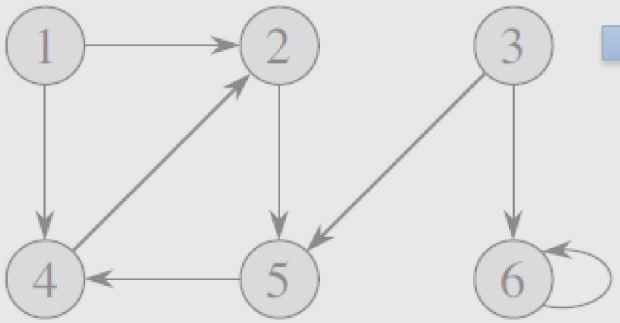
Note: the different notation (u, v) vs. <u, v>

- Undirected graph:



- G1 = (V1, E1), where
  V1 = {1, 2, 3, 4, 5},
  E1 = {(1, 2), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (4, 5)}
  Note: the order (u, v) may be changed

- Directed graph:

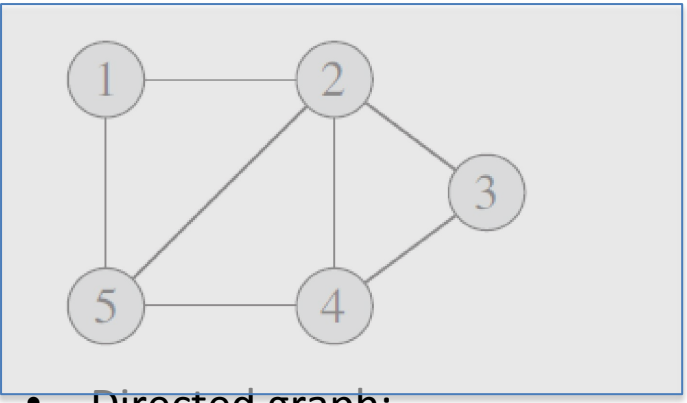


- G2 = (V2, E2), where
  V2 = {1, 2, 3, 4, 5, 6},
  E2 = {<1, 2>, <1, 4>, <2, 5>, <3, 5>, <3, 6>, <4, 2>, <5, 4>, <6, 6>}
  Note: the order <u, v> cannot be changed

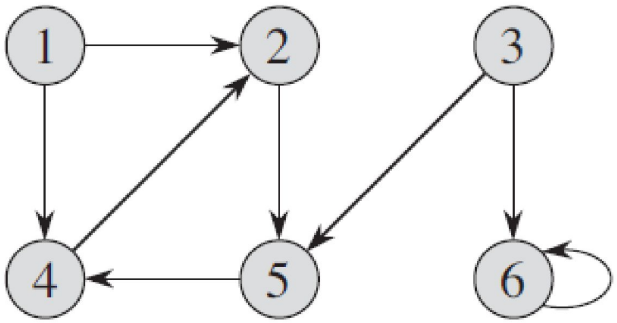
What is a graph?

- A graph, G = (V, E), consists of:
  - Set of vertices, V
  - Set of edges, E, where edges are either ‘undirected’ or ‘directed’

- Undirected graph:



- Directed graph:

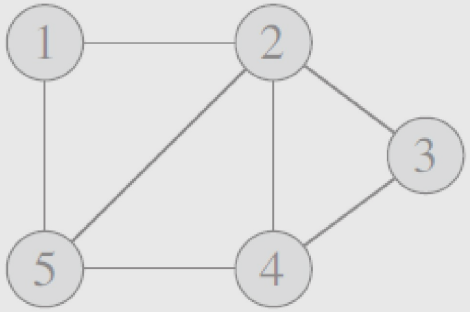


What is a graph?

- A graph, G = (V, E), consists of:
  - Set of vertices, V
  - Set of edges, E, where edges are either 'undirected' or 'directed'

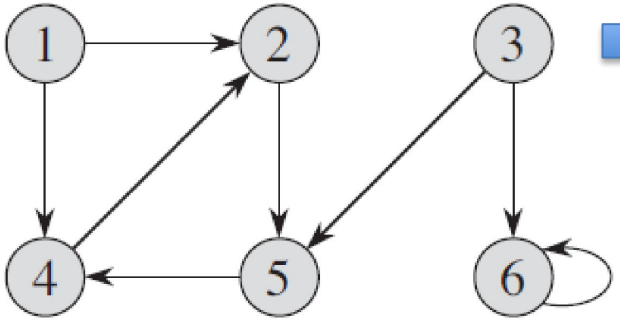
Note: the different notation (u, v) vs. <u, v>

- Undirected graph:



- G1 = (V1, E1), where
  V1 = {1, 2, 3, 4, 5},
  E1 = {(1, 2), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (4, 5)}
  Note: the order (u, v) may be changed

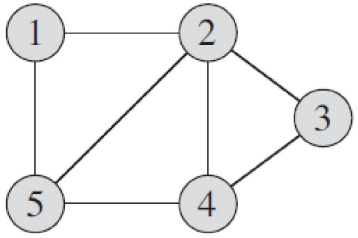
- Directed graph:



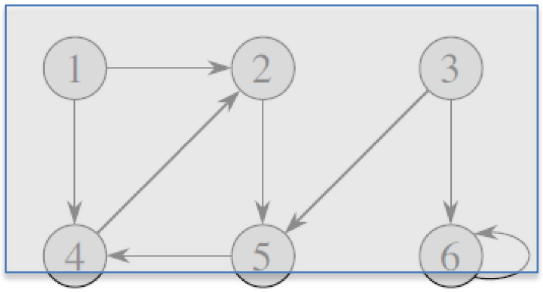
- G2 = (V2, E2), where
  V2 = {1, 2, 3, 4, 5, 6},
  E2 = {<1, 2>, <1, 4>, <2, 5>, <3, 5>, <3, 6>, <4, 2>, <5, 4>, <6, 6>}
  Note: the order <u, v> cannot be changed

Some terminology: adjacency

- In an undirected graph:
  - Two vertices u and v are "adjacent" if (u,v) or (v, u) is an edge in E
  - Also edge e = (u,v) is incident to vertex u and to vertex v

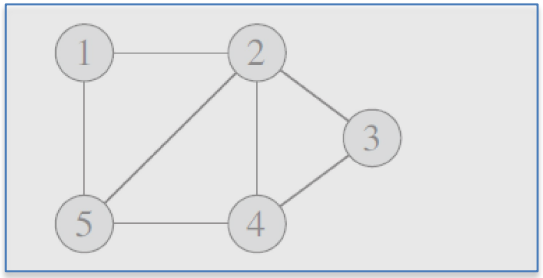


- In a directed graph:
  - Vertex u is adjacent **to** vertex **v** if <u, v> is an edge in E
    - vertex u is the initial vertex of <u,v>
  - Vertex v is adjacent **from** vertex **u** if <u, v> is an edge in E
    - vertex v is the terminal, or end, vertex of <u,v>



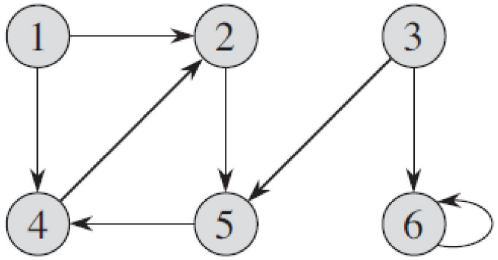
Some terminology: adjacency

- In an undirected graph:
  - Two vertices u and v are "adjacent" if (u,v) or (v, u) is an edge in E
  - Also edge e = (u,v) is incident to vertex u and to vertex v



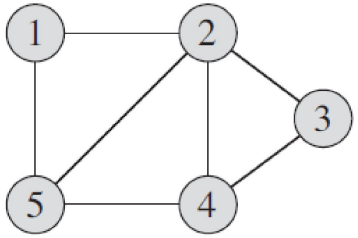
- In a directed graph:
  - Vertex u is adjacent **to** vertex **v** if <u, v> is an edge in E
    - vertex u is the initial vertex of <u,v>
  - Vertex v is adjacent **from** vertex **u** if <u, v> is an edge in E
    - vertex v is the terminal, or end, vertex of <u,v>

Note: the difference in "adjacent from" and "adjacent to"

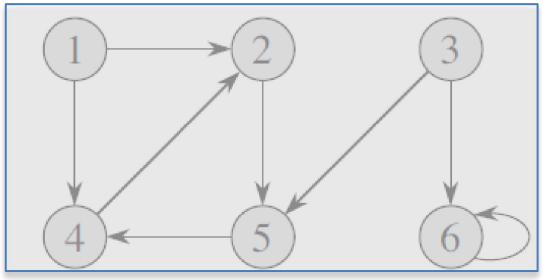


Some terminology: degree

- In an undirected graph the "degree" of vertex u is simply the number of edges that are incident **with** vertex u
  - a self-loop counts twice (both ends count)
  - The degree is denoted with deg(v)
  - If e=|E|, then always the case: $2e = \sum_{v \in V} \deg(v)$

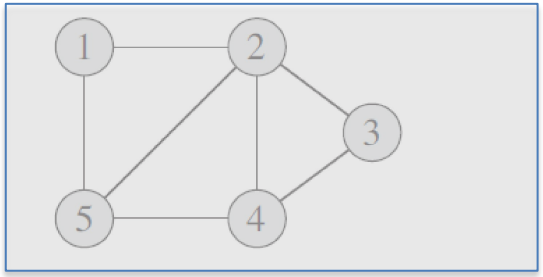


- In a directed graph:
  - In-degree of vertex u is number of edges adjacent to vertex u, viz. terminate at vertex u
    - The In-degree is denoted as in-deg(u)
  - Out-degree of vertex u is number of edges adjacent from vertex u, viz. where vertex u is the initial vertex
    - The Out-degree is denoted as out-deg(u)

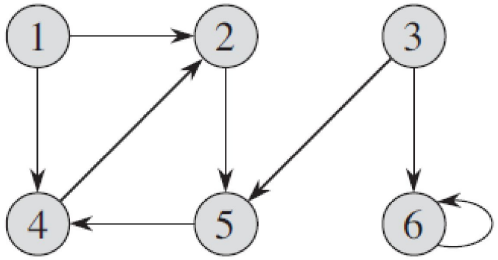


Some terminology: degree

- In an undirected graph the "degree" of vertex u is simply the number of edges that are incident **with** vertex u
  - a self-loop counts twice (both ends count)
  - The degree is denoted with deg(v)
  - If e=|E|, then always the case: $2e = \sum_{v \in V} \deg(v)$



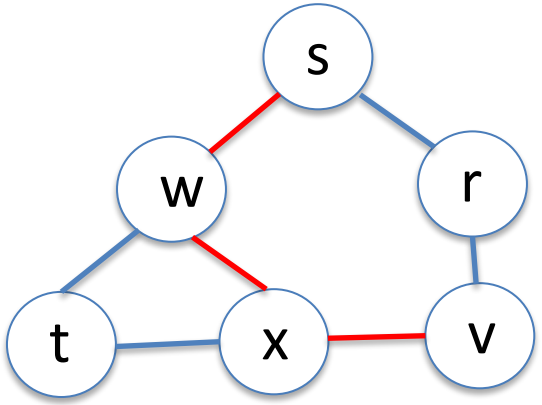
- In a directed graph:
  - In-degree of vertex v is number of edges adjacent **to** vertex v, viz. terminate at vertex v
    - The In-degree is denoted as in-deg(v)
  - Out-degree of vertex u is number of edges adjacent **from** vertex u, viz. where vertex u is the initial vertex
    - The Out-degree is denoted as out-deg(u)



Some more terminology: paths in an undirected graph

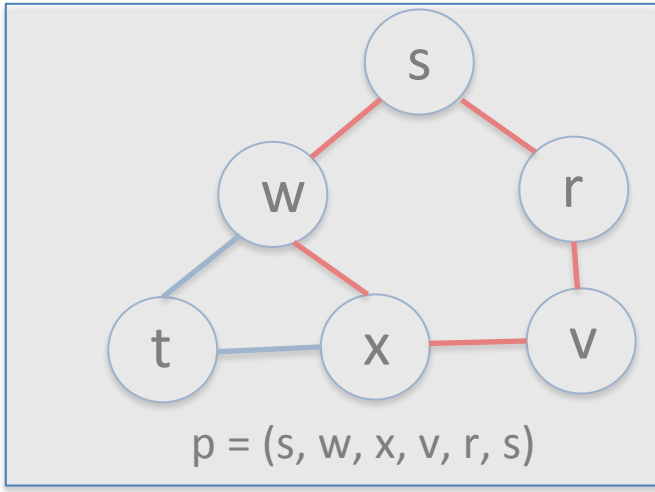
Let undirected graph, G =(V, E)

- A path, p = (u_1, u_2, u_3, …, u_n) is said to be a simple path if:
  - Edges (u_i, u_{i+1}) are in E, and
  - Vertices u_1, u_2, u_3, …, u_n are all distinct



p = (s, w, x, v)

- A path, p = (u_1, u_2, u_3, …, u_n) is said to be a cycle if:
  - Edges (u_i, u_{i+1}) are in E, and
  - Vertices u_2, u_3, …, u_n are all distinct, and
  - $u_1 = u_n$

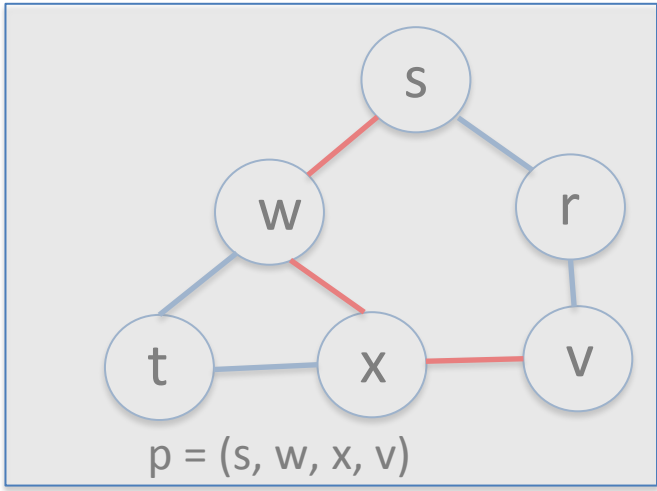


p = (s, w, x, v, r, s)

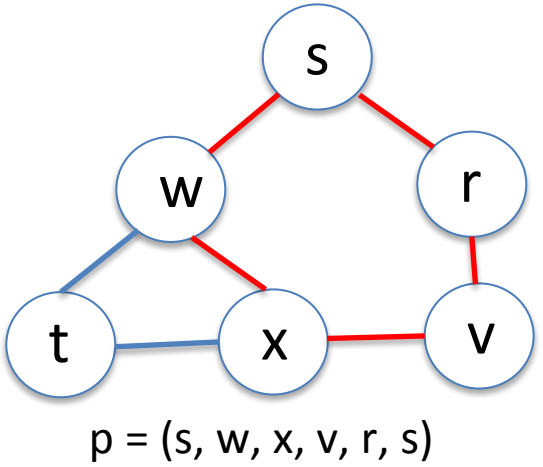
Some more terminology: paths in an undirected graph

Let undirected graph, G =(V, E)

- A path, p = (u_1, u_2, u_3, …, u_n) is said to be a simple path if:
  - Edges (u_i, u_{i+1}) are in E, and
  - Vertices u_1, u_2, u_3, …, u_n are all distinct



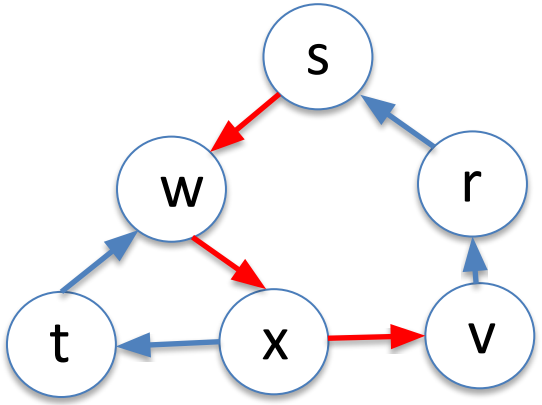
- A path, p = (u_1, u_2, u_3, …, u_n) is said to be a cycle if:
  - Edges (u_i, u_{i+1}) are in E, and
  - Vertices u_2, u_3, …, u_n are all distinct, and
  - $u_1 = u_n$



Some more terminology: paths in an directed graph

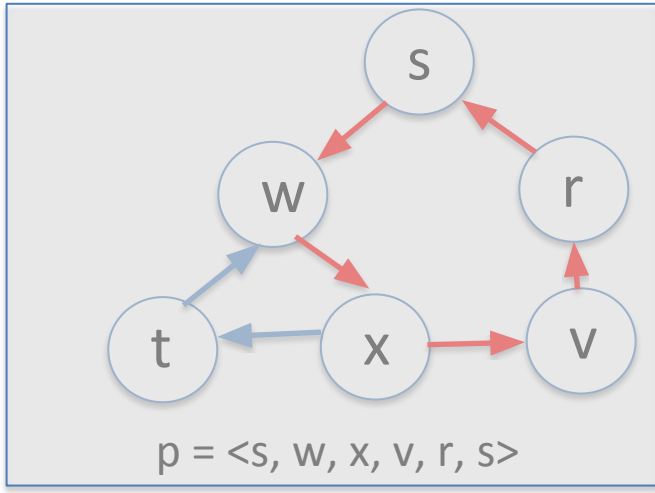
Let directed graph, G =(V, E)

- A path, p = <u_1, u_2, u_3, …, u_n> is said to be a simple path if:
  - Edges <u_i, u_{i+1}> are in E, and
  - Vertices u_1, u_2, u_3, …, u_n are all distinct



p = <s, w, x, v>

- A path, p = <u_1, u_2, u_3, …, u_n> is said to be a cycle if:
  - Edges <u_i, u_{i+1}> are in E, and
  - Vertices u_2, u_3, …, u_n are all distinct, and
  - $u_1 = u_n$

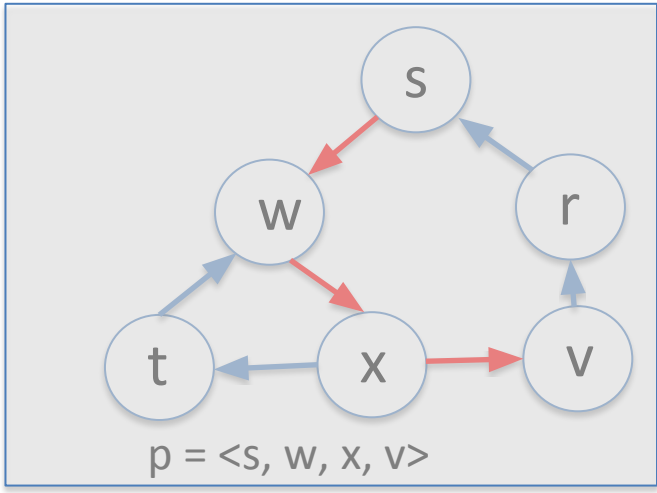


p = <s, w, x, v, r, s>

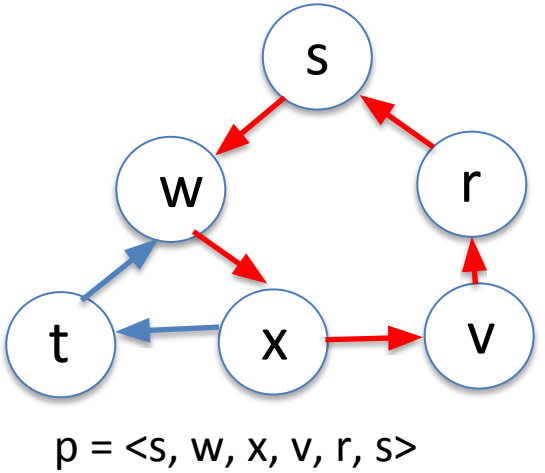
Some more terminology: paths in an directed graph

Let directed graph, G =(V, E)

- A path, p = <u_1, u_2, u_3, …, u_n> is said to be a simple path if:
  - Edges <u_i, u_{i+1}> are in E, and
  - Vertices u_1, u_2, u_3, …, u_n are all distinct



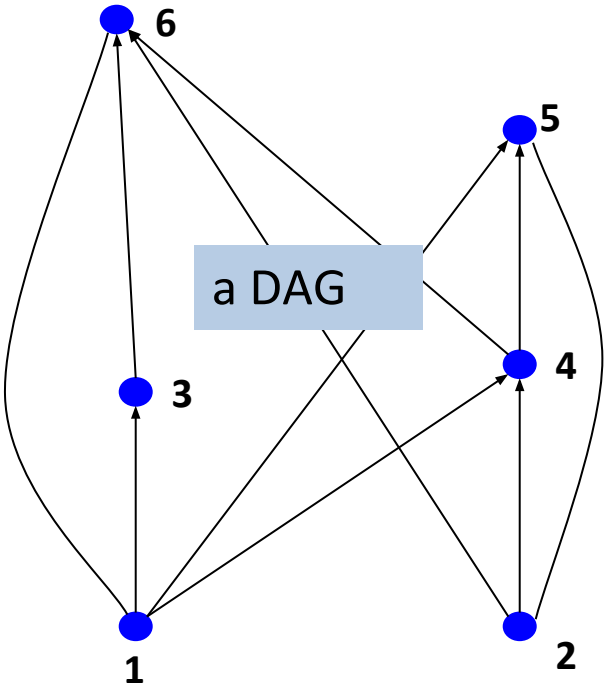
- A path, p = <u_1, u_2, u_3, …, u_n> is said to be a cycle if:
  - Edges <u_i, u_{i+1}> are in E, and
  - Vertices u_2, u_3, …, u_n are all distinct, and
  - $u_1 = u_n$

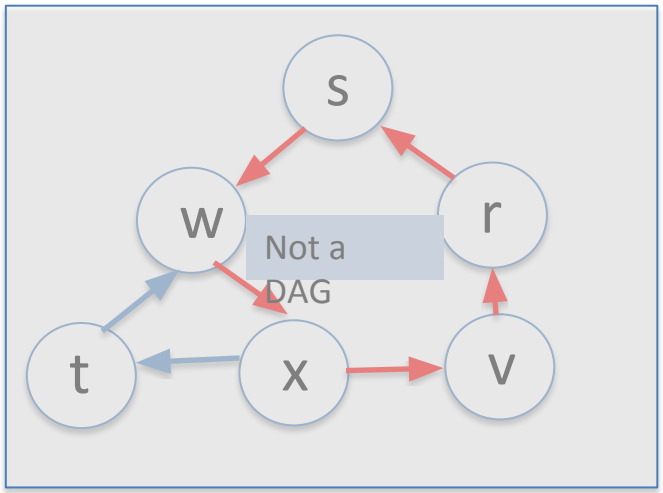


Some more terminology: directed acyclic graphs (DAGs)

Let directed graph, G =(V, E)

- Graph G is said to be a directed acyclic graph (or DAG) if:
  - There are no cycles in G

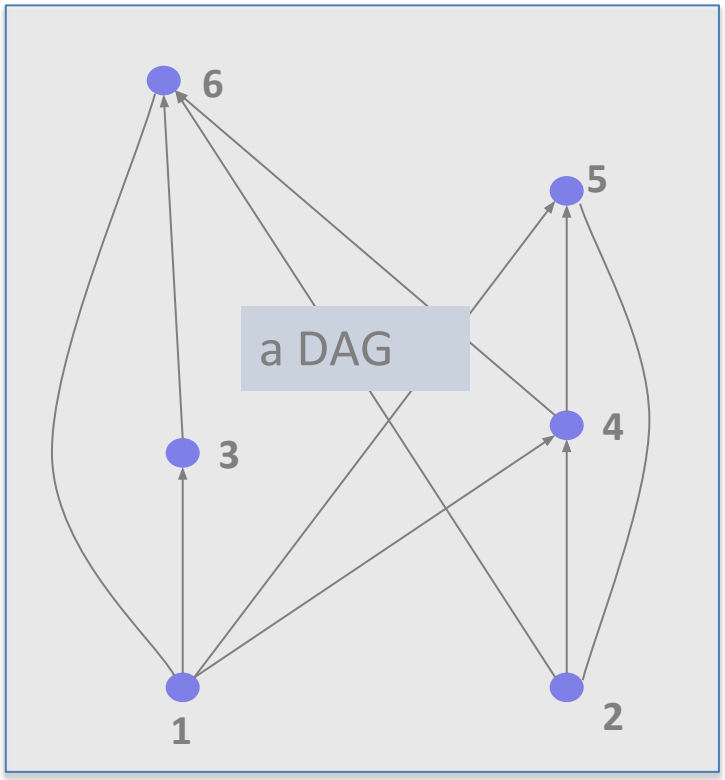


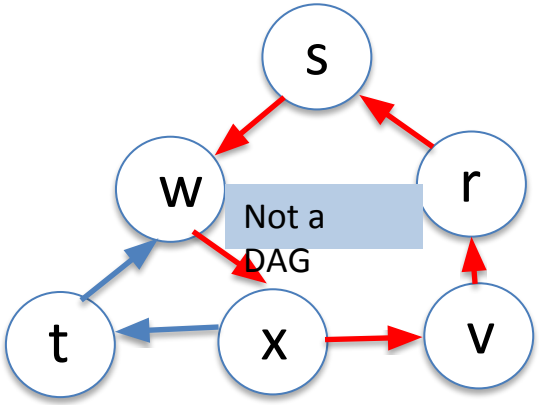


Some more terminology: directed acyclic graphs (DAGs)

Let directed graph, G =(V, E)

- Graph G is said to be a directed acyclic graph (or DAG) if:
  - There are no cycles in G

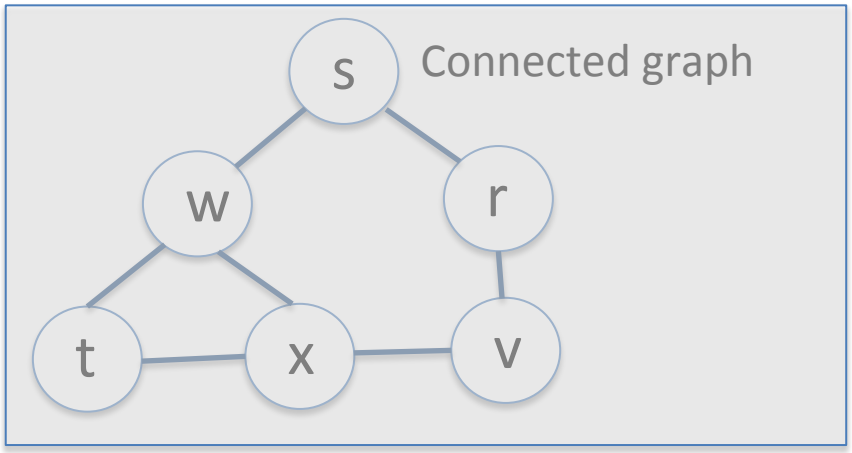


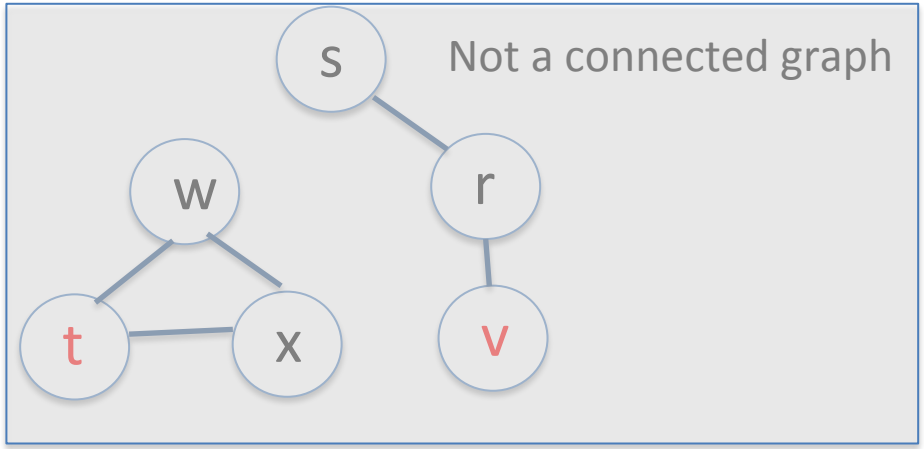


Some more terminology: connectedness of undirected graphs

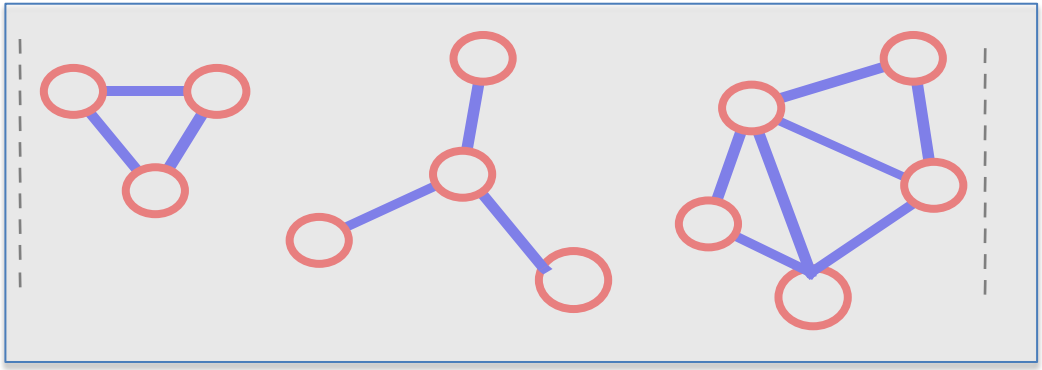
- Subgraph: subset of vertices, V_1 c V, and subset of edges. E_1 c E such that they form a graph

- A undirected graph, G =(V, E), is said to be connected if there is a path from u to v, for any and every given pair of vertices u and v, in V





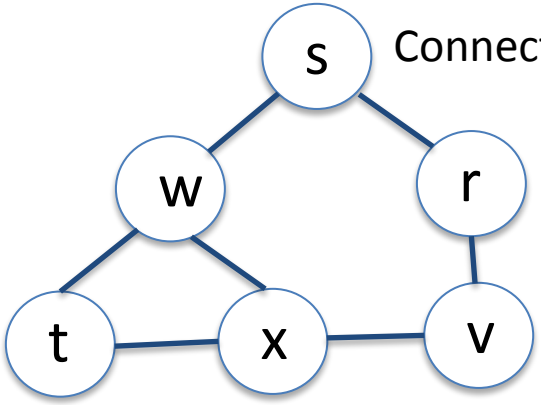
- Connected components: maximal connected subgraph. E.g., the graph below has 3 connected components



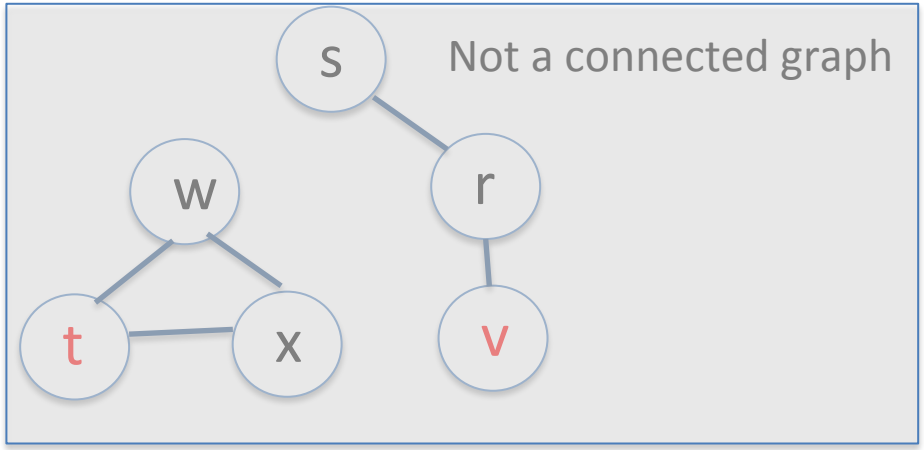
Some more terminology: connectedness of undirected graphs

- Subgraph: subset of vertices, V_1 c V, and subset of edges. E_1 c E such that they form a graph

- A undirected graph, G =(V, E), is said to be connected if there is a path from u to v, for any and every given pair of vertices u and v, in V

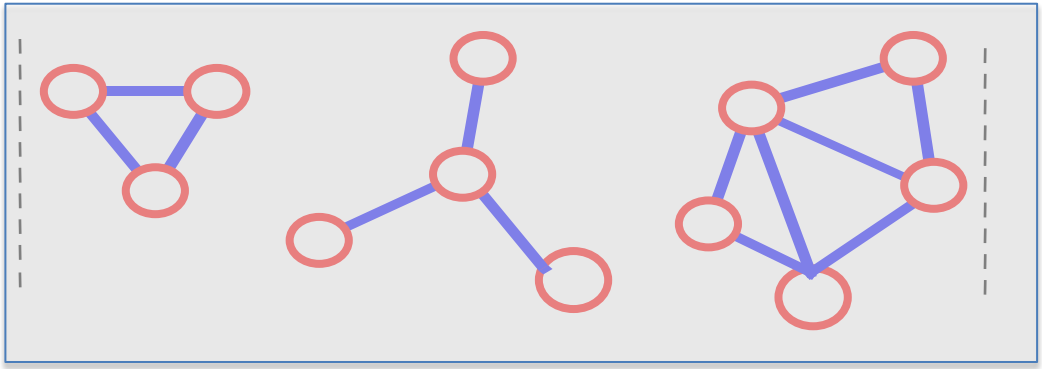


Connected graph



Not a connected graph

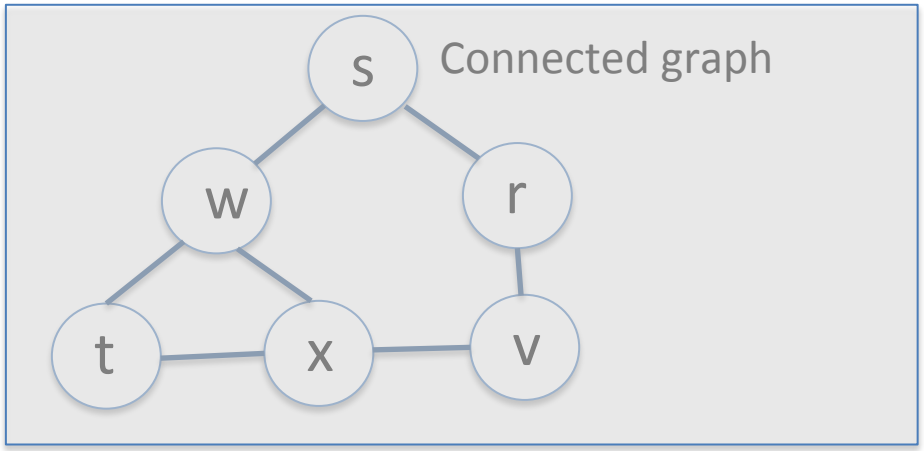
- Connected components: maximal connected subgraph. E.g., the graph below has 3 connected components

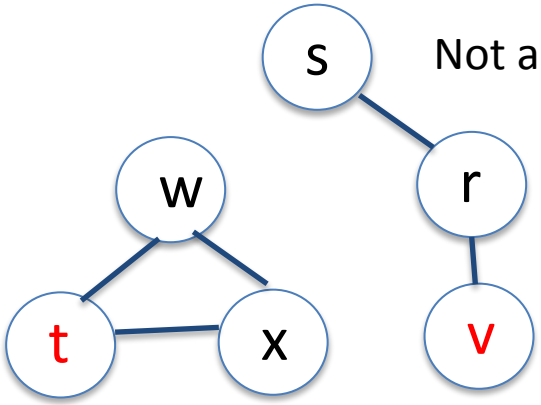


Some more terminology: connectedness of undirected graphs

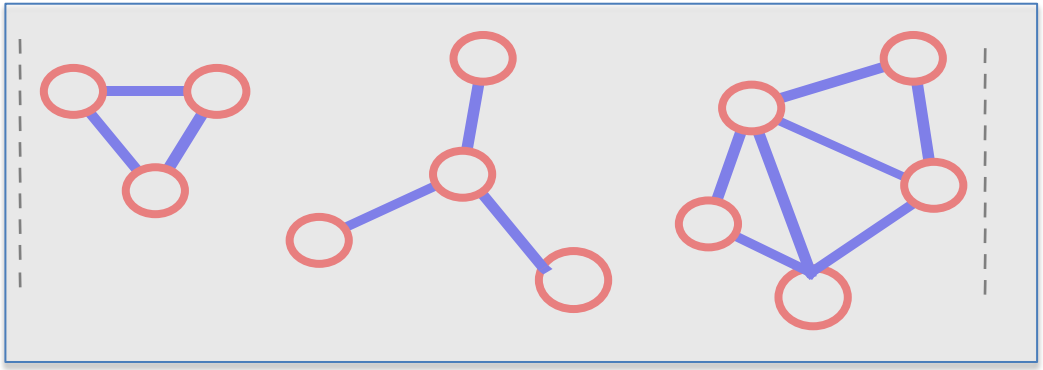
- Subgraph: subset of vertices, V_1 c V, and subset of edges. E_1 c E such that they form a graph

- A undirected graph, G =(V, E), is said to be connected if there is a path from u to v, for any and every given pair of vertices u and v, in V





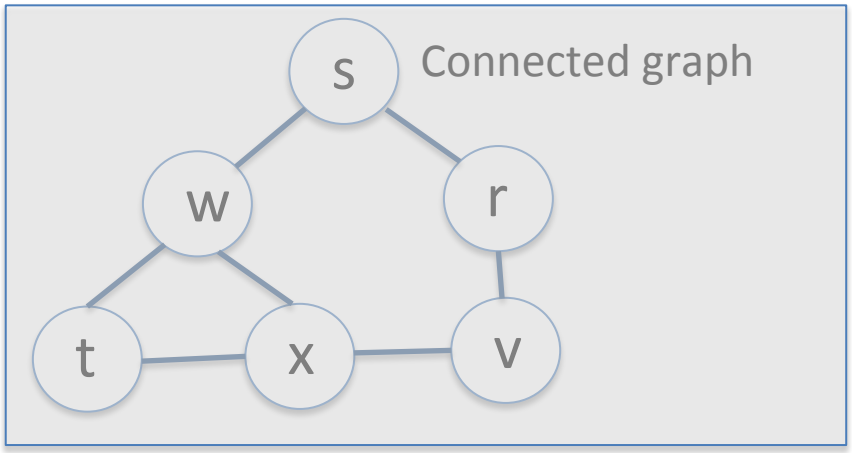
- Connected components: maximal connected subgraph. E.g., the graph below has 3 connected components

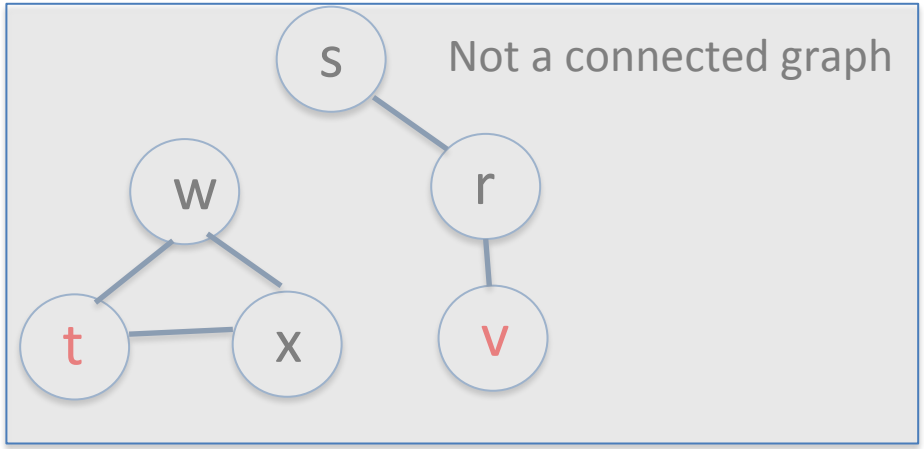


Some more terminology: connectedness of undirected graphs

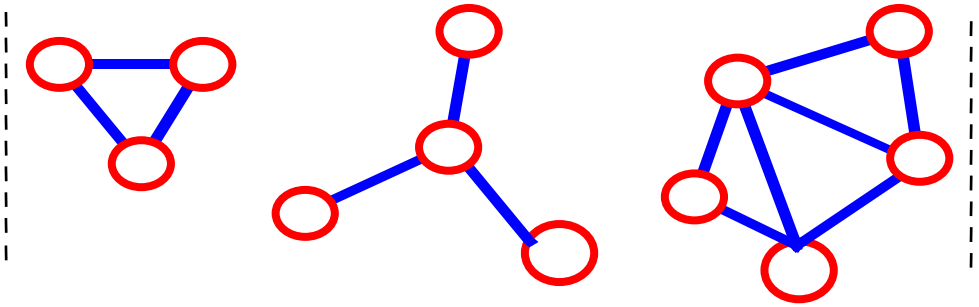
- Subgraph: subset of vertices, V_1 c V, and subset of edges. E_1 c E such that they form a graph

- A undirected graph, G =(V, E), is said to be connected if there is a path from u to v, for any and every given pair of vertices u and v, in V



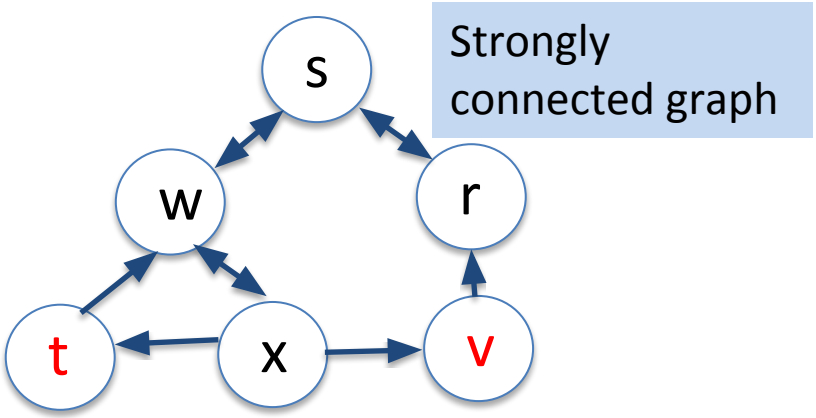


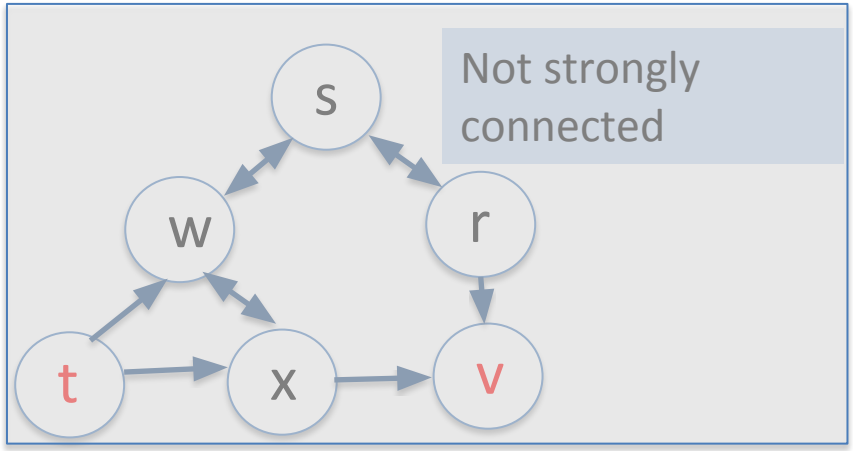
- Connected components: maximal connected subgraph. E.g., the graph below has 3 connected components



Some more terminology: connectedness of directed graphs

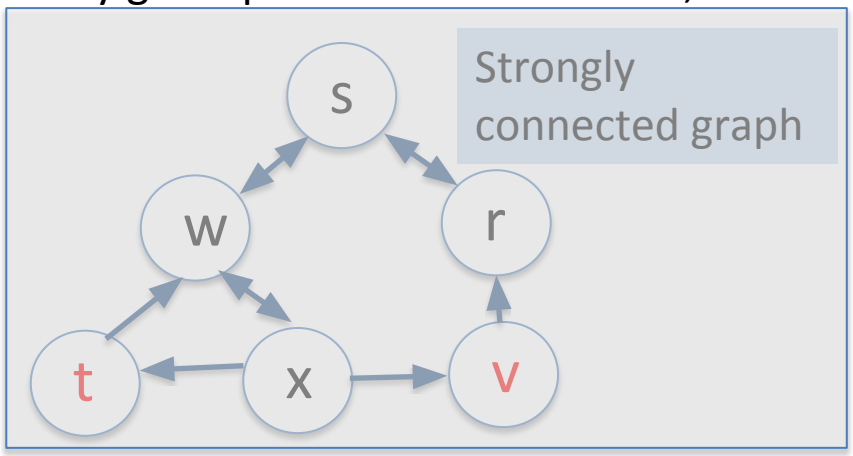
- A graph, G =(V, E), is strongly connected if there is a path from u to v, and from v to u, for any given pair of vertices u and v, in V

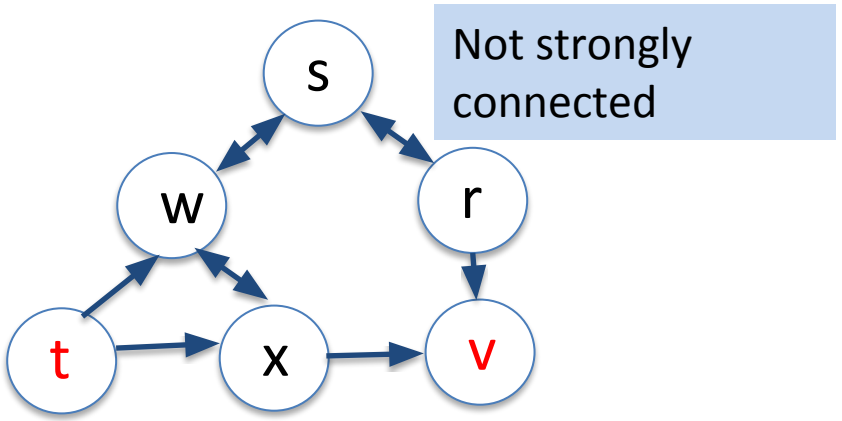




Some more terminology: connectedness of directed graphs

- A graph, G =(V, E), is strongly connected if there is a path from u to v, and from v to u, for any given pair of vertices u and v, in V





Representation of graphs using adjacency matrices

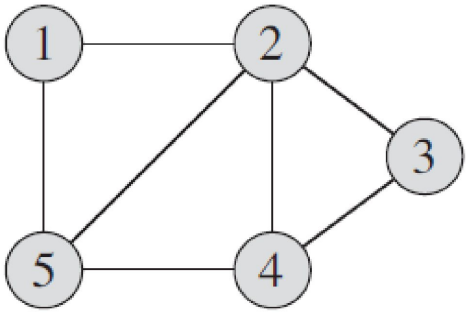
- A graph G = (V, E) may be represented by an n x n 'adjacency matrix', A = $\{a_{ij}\}$, where n = |V|
- For an 'undirected' graph, G = (V, E)

  $a_{ij} = a_{ji} = 1$, if there is an edge between verticies i and j

  $= 0$, otherwise
- For a 'directed' graph, G = (V, E)

  $a_{ij} = 1$, if there is an edge from vertex i to vertex j

  $= 0$, otherwise



A =

					i=
0	1	0	0	1	1
1	0	1	1	1	2
0	1	0	1	0	3
0	1	1	0	1	4
1	1	0	1	0	5

j=

1	2	3	4	5

Representation of graphs using adjacency matrices

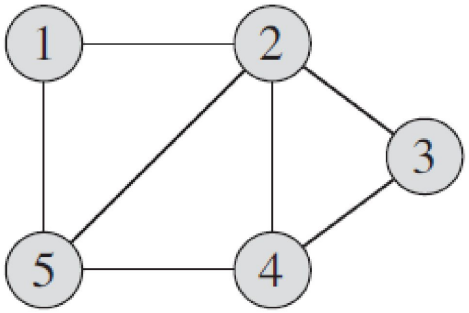
- A graph G = (V, E) may be represented by an n x n 'adjacency matrix', A = $\{a_{ij}\}$, where n = |V|
- For an 'undirected' graph, G = (V, E)

  $a_{ij} = a_{ji} = 1$, if there is an edge between verticies i and j

  = 0, otherwise
- For a 'directed' graph, G = (V, E)

  a_{ij} = 1, if there is an edge from vertex i to vertex j

  = 0, otherwise



A =

					i=
0	1	0	0	1	1
1	0	1	1	1	2
0	1	0	1	0	3
0	1	1	0	1	4
1	1	0	1	0	5

j=

1	2	3	4	5

Representation of graphs using adjacency matrices

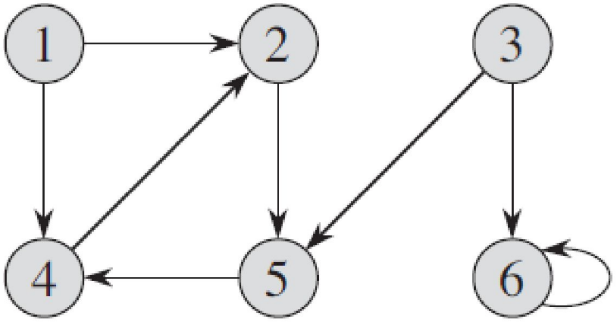
- A graph G = (V, E) may be represented by an n x n 'adjacency matrix', A = $\{a_{ij}\}$, where n = |V|
- For an 'undirected' graph, G = (V, E)

  $a_{ij} = a_{ji} = 1$, if there is an edge between verticies i and j

  $= 0$, otherwise
- For a 'directed' graph, G = (V, E)

  $a_{ij} = 1$, if there is an edge from vertex i to vertex j

  $= 0$, otherwise



A =

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

Representation of graphs using adjacency matrices

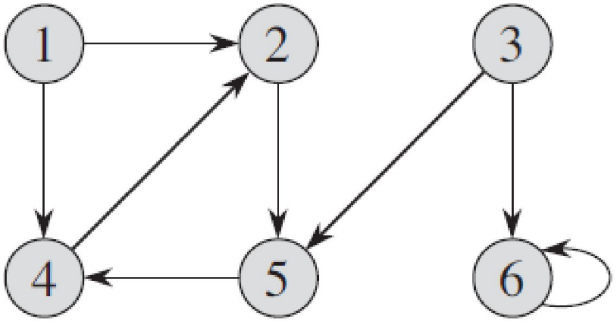
- A graph G = (V, E) may be represented by an n x n 'adjacency matrix', A = $\{a_{ij}\}$, where n = |V|
- For an 'undirected' graph, G = (V, E)

  $a_{ij} = a_{ji} = 1$, if there is an edge between verticies i and j

  $= 0$, otherewise
- For a 'directed' graph, G = (V, E)

  $a_{ij} = 1$, if there is an edge from vertex i to vertex j

  $= 0$, otherwise



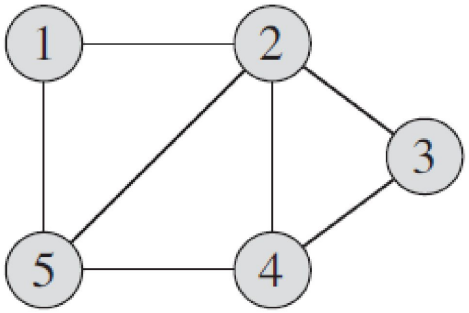
A =

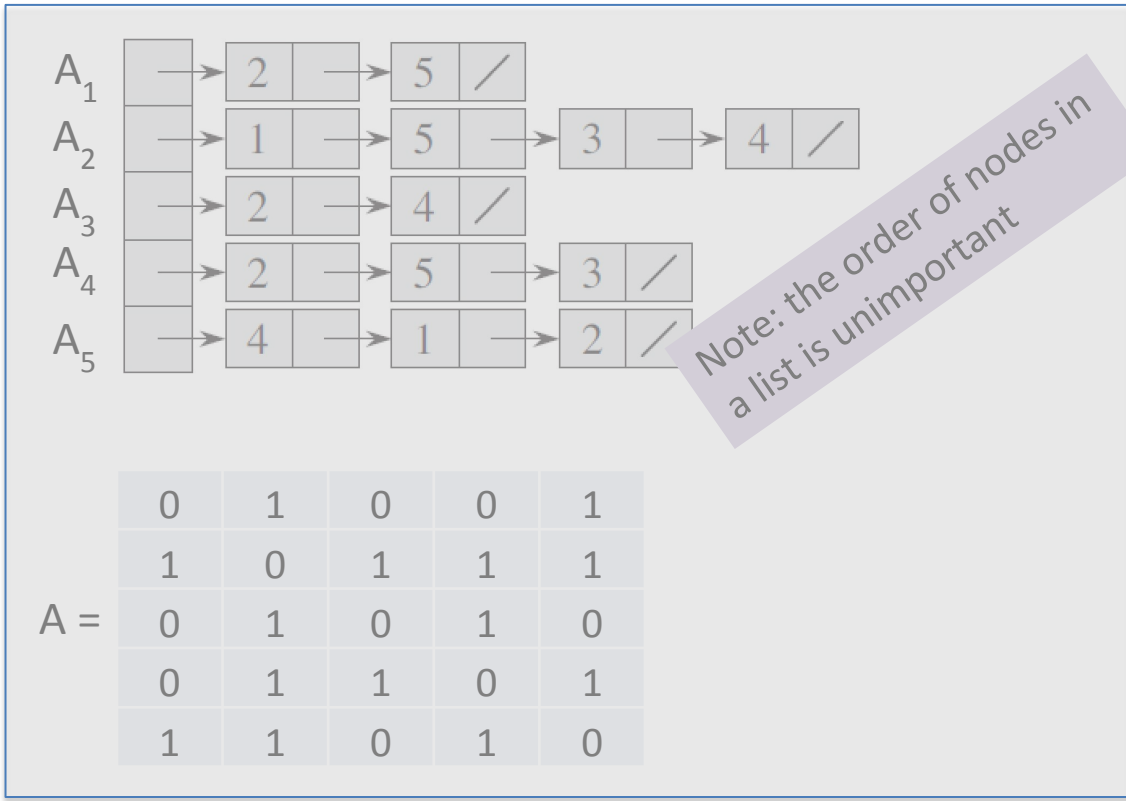
0	1	0	1	0	0	1
0	0	0	0	1	0	2
0	0	0	0	1	1	3
0	1	0	0	0	0	4
0	0	0	1	0	0	5
0	0	0	0	0	1	6
1	2	3	4	5	6	

Representation of graphs using adjacency lists

- A graph G = (V, E) may instead be represented by an array of n 'adjacency lists' A_i
- For an 'undirected' graph, G = (V, E)

  list A_i = [V_j: there is an edge between verticies i and j]

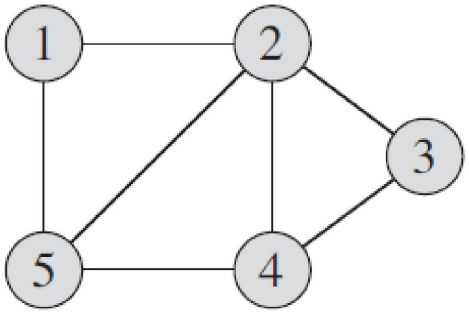


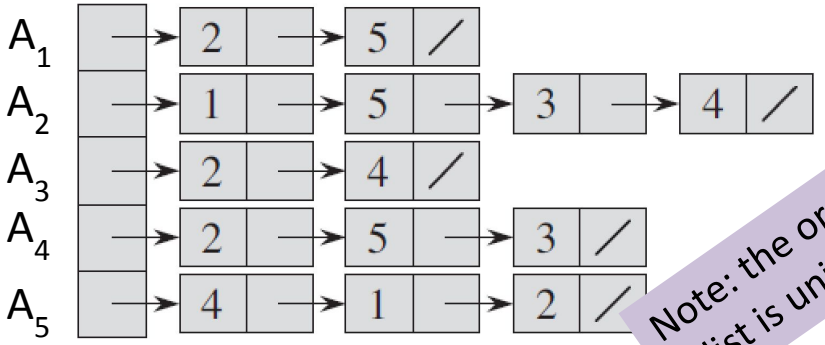


Representation of graphs using adjacency lists

- A graph G = (V, E) may instead be represented by an array of n 'adjacency lists' A_i
- For an 'undirected' graph, G = (V, E)

  list A_i = [V_j: there is an edge between verticies i and j]





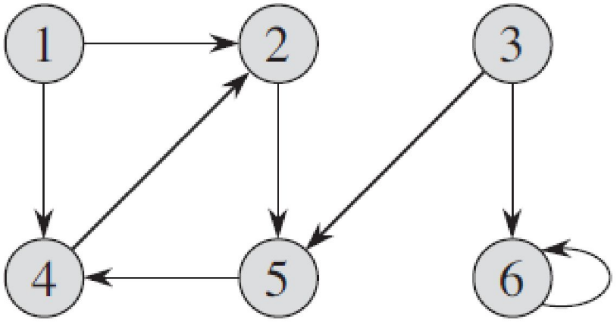
Note: the order of nodes in a list is unimportant

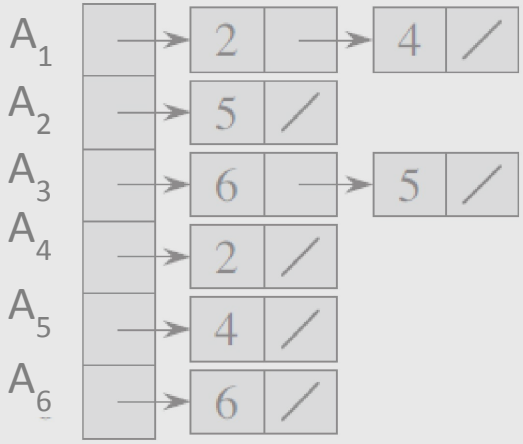
A =

0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0

Representation of graphs using adjacency lists

- A graph G = (V, E) may instead be represented by an array of n 'adjacency lists' A_i
- For an 'undirected' graph, G = (V, E)
  list A_i = [V_j: there is an edge between verticies i and j]
- For an 'directed' graph, G = (V, E)
  list A_i = [V_j: there is an edge from vertex i to j]



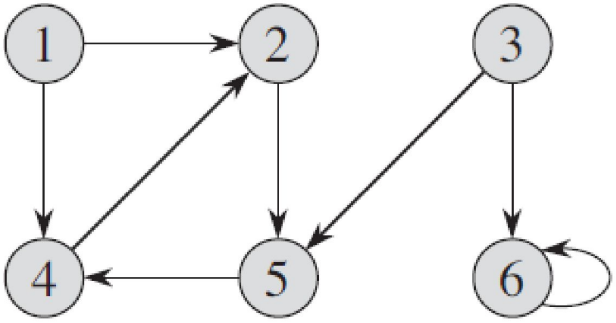


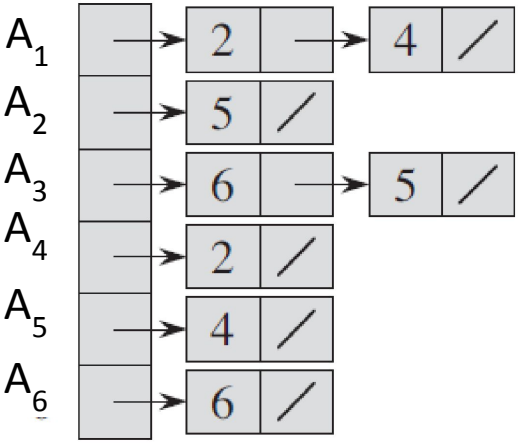
A =

0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Representation of graphs using adjacency lists

- A graph G = (V, E) may instead be represented by an array of n 'adjacency lists' A_i
- For an 'undirected' graph, G = (V, E)
  list A_i = [V_j: there is an edge between verticies i and j]
- For an 'directed' graph, G = (V, E)
  list A_i = [V_j: there is an edge from vertex i to j]





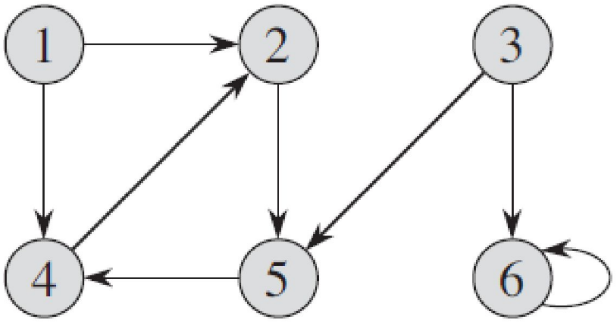
A =

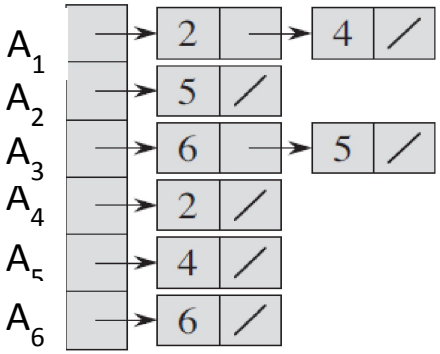
0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Representation of graphs: adjacency matrices vs. adjacency lists

- Truly, this is a direct consequence of fact that an n x n matrix can be represented by a collection of n lists
- Memory requirements:
  - adjacency matrix: $O(|V|^2)$ bits
  - adjacency lists: $O(|V| + |E|)$ integers/addresses
- Consider the two representations when working with 'sparse' graphs, where $|E| << |V|^2$
- Graph algorithms simpler to visualize using adjacency matric representation
  - BUT, run more efficiently on adjacency list representations
  - E.g. Visiting all edges, or vertices, requires:
    - $O(|V|^2)$ operations using an adjacency matrix
    - $O(|V| + |E|)$ operations when using adjacency lists

Note: time or space complexity related to graphs is invariably f(|V|, |E|)





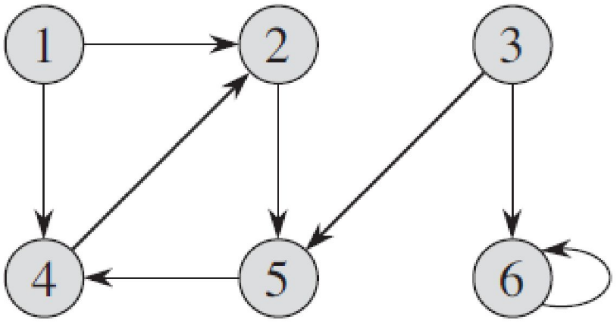
A =

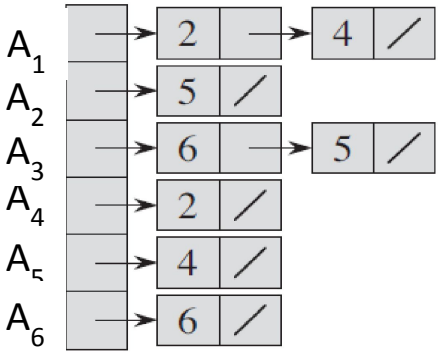
0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Representation of graphs: adjacency matrices vs. adjacency lists

- Truly, this is a direct consequence of fact that an n x n matrix can be represented by a collection of n lists
- Memory requirements:
  - adjacency matrix: $O(|V|^2)$ bits
  - adjacency lists: $O(|V| + |E|)$ integers/addresses
- Consider the two representations when working with 'sparse' graphs, where $|E| << |V|^2$
- Graph algorithms simpler to visualize using adjacency matric representation
  - BUT, run more efficiently on adjacency list representations
  - E.g. Visiting all edges, or vertices, requires:
    - $O(|V|^2)$ operations using an adjacency matrix
    - $O(|V| + |E|)$ operations when using adjacency lists

Note: time or space complexity related to graphs is invariably f(|V|, |E|)





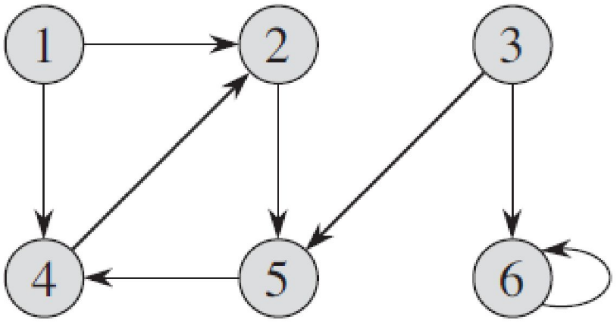
A =

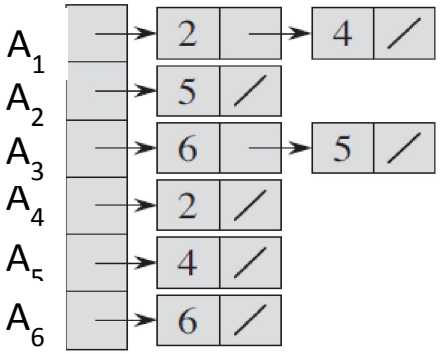
0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Representation of graphs: adjacency matrices vs. adjacency lists

- Truly, this is a direct consequence of fact that an n x n matrix can be represented by a collection of n lists
- Memory requirements:
  - adjacency matrix: $O(|V|^2)$ bits
  - adjacency lists: $O(|V| + |E|)$ integers/addresses
- Evaluate the two representations when working with 'sparse' graphs, where $|E| << |V|^2$
- Graph algorithms simpler to visualize using adjacency matric representation
  - BUT, run more efficiently on adjacency list representations
  - E.g. Visiting all edges, or vertices, requires:
    - $O(|V|^2)$ operations using an adjacency matrix
    - $O(|V| + |E|)$ operations when using adjacency lists

Note: time or space complexity related to graphs is invariably f(|V|, |E|)





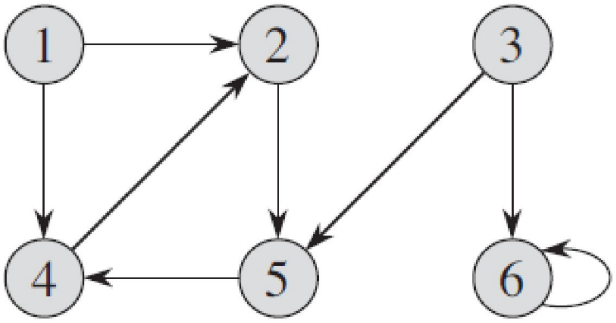
A =

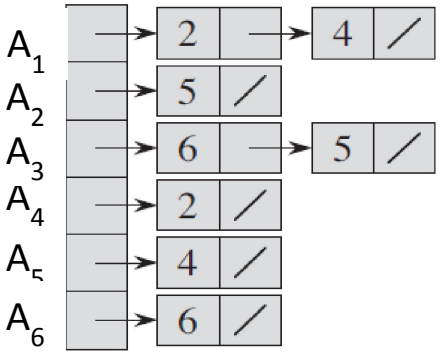
0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Representation of graphs: adjacency matrices vs. adjacency lists

- Truly, this is a direct consequence of fact that an n x n matrix can be represented by a collection of n lists
- Memory requirements:
  - adjacency matrix: $O(|V|^2)$ bits
  - adjacency lists: $O(|V| + |E|)$ integers/addresses
- Consider the two representations when working with 'sparse' graphs, where $|E| << |V|^2$
- Graph algorithms simpler to visualize using adjacency matric representation
  - BUT, run more efficiently on adjacency list representations
  - E.g. Visiting all edges, or vertices, requires:
    - $O(|V|^2)$ operations using an adjacency matrix
    - $O(|V| + |E|)$ operations when using adjacency lists

Note: time or space complexity related to graphs is invariably f(|V|, |E|)



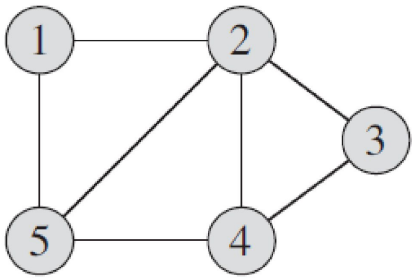


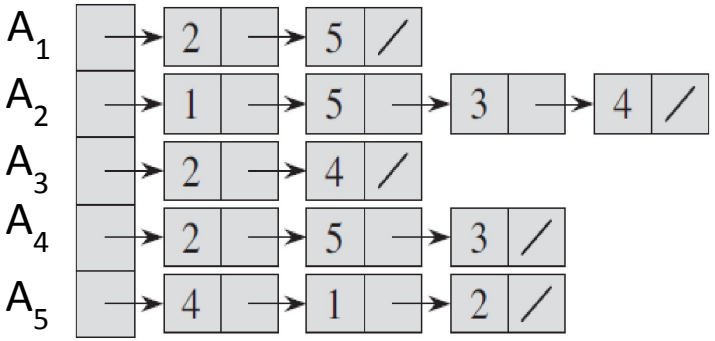
A =

0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Representation of graphs: adjacency matrices vs. adjacency lists

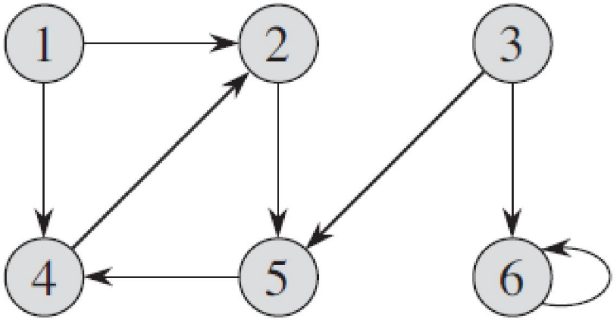
- Here are some problems for which you must write algorithms assuming that the graph is represented using an adjacency matrix. Re-do this using adjacency lists
  - Count the number of edges in an undirected graph
  - Count the number of edges in a directed graph
  - What is the OUT-degree of each vertex in a directed graph
  - What is the IN-degree of each vertex in a directed graph

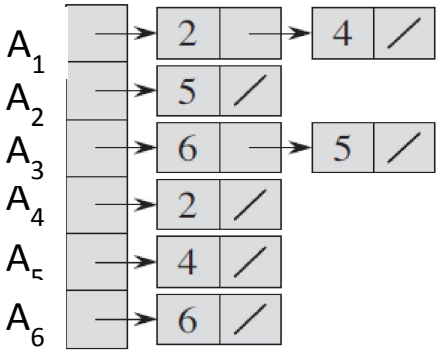




A =

0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0





A =

0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

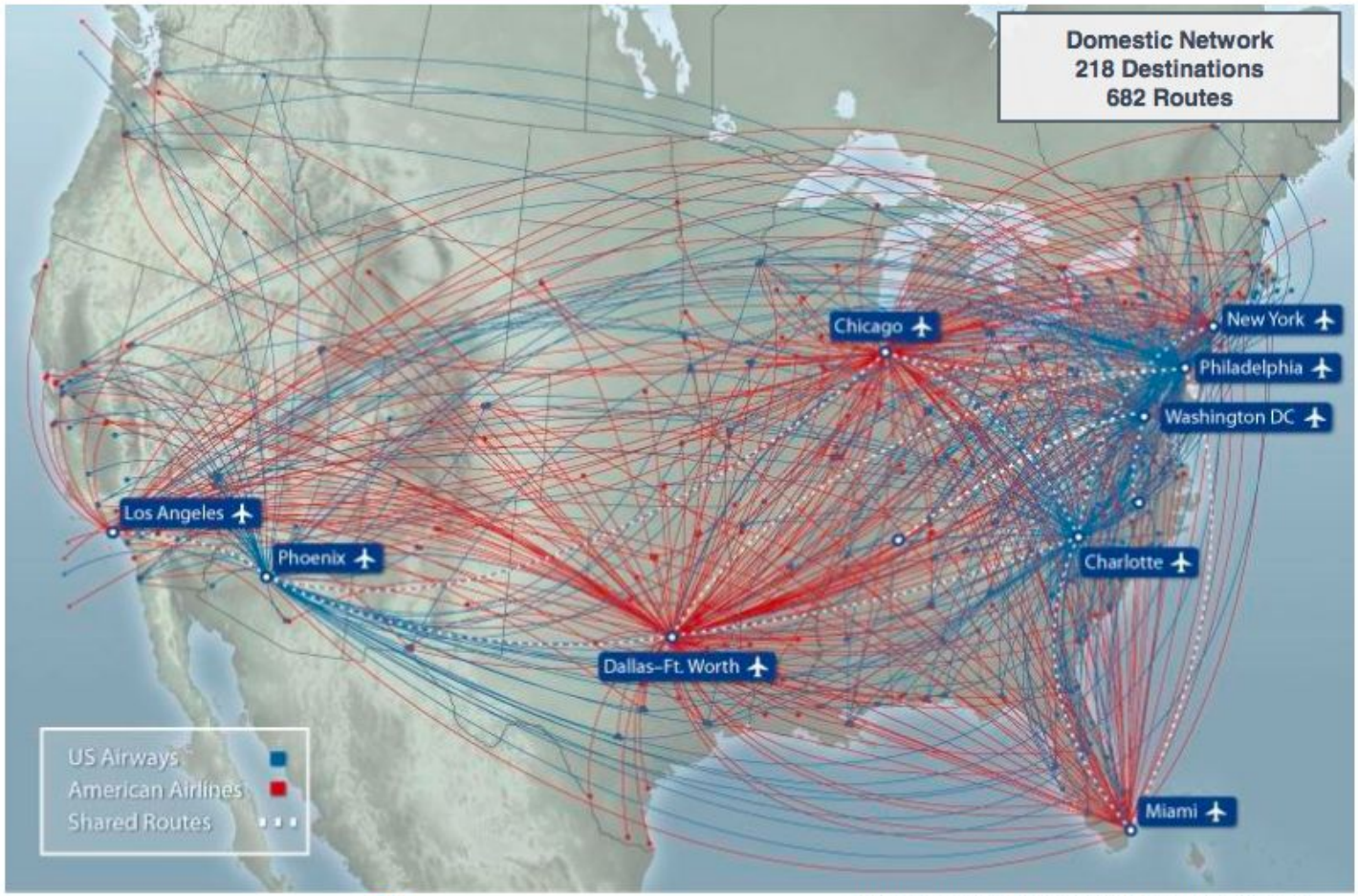
More on graphs

Sparse graphs

- Sparse graphs, where $|E| << |V|^2$
- Domestic network of American airlines:
  - $|V| = 218$ destinations, but only $|E| = 682$ routes $<< 23{,}653 = |V|^2$
- More efficient to work with adjacency list representation (218 lists, 682 list nodes)

Sparse graphs

- Sparse graphs, where $|E| << |V|^2$
- Domestic network of American airlines:
  - $|V| = 218$ destinations, but only $|E| = 682$ routes $<< 23{,}653 = |V|^2$
- More efficient to work with adjacency list representation (218 lists, 682 list nodes)



Diio 2013 published schedules as of January 25, 2013

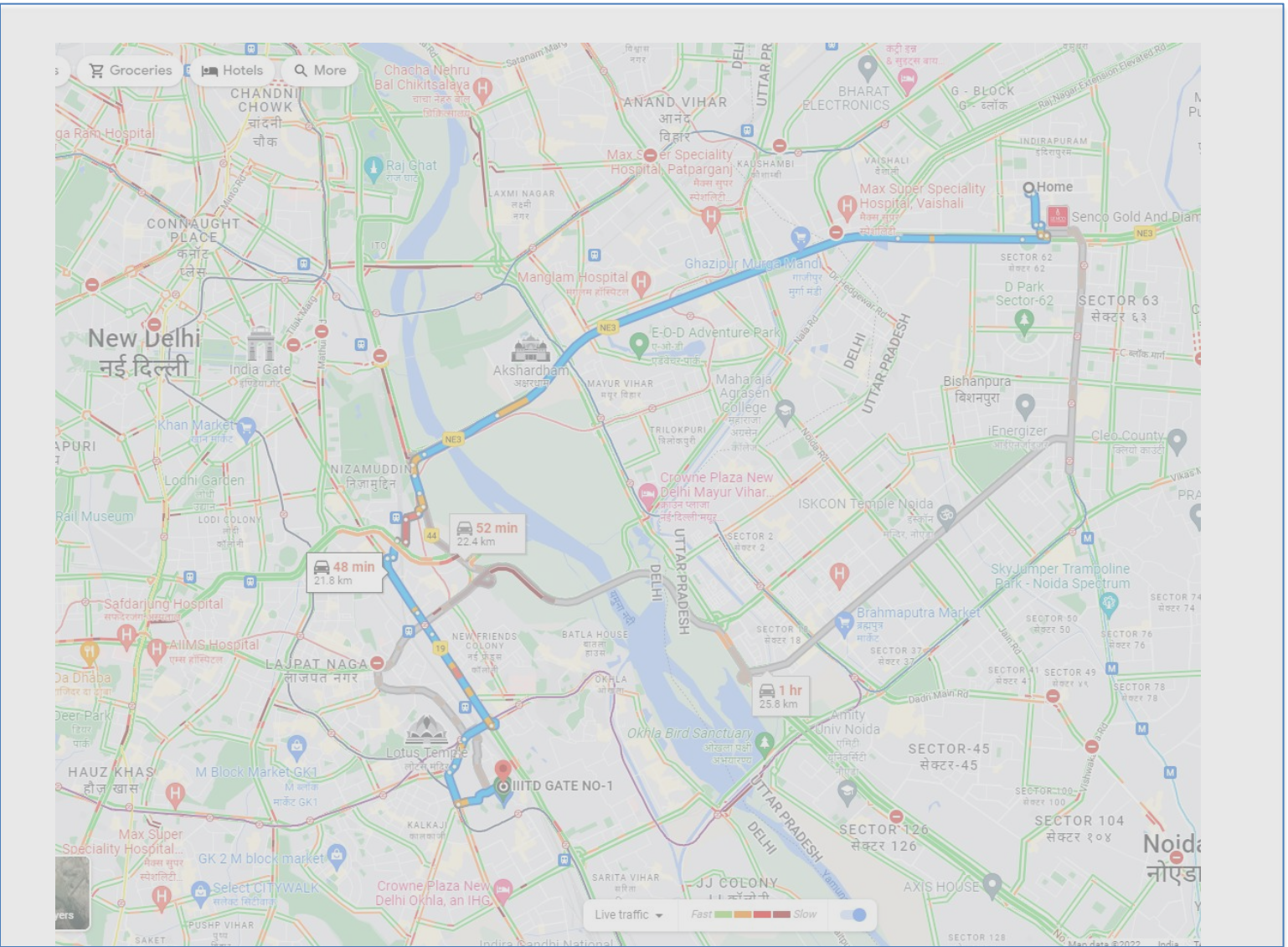
American Airlines

8

U·S AIRWAYS

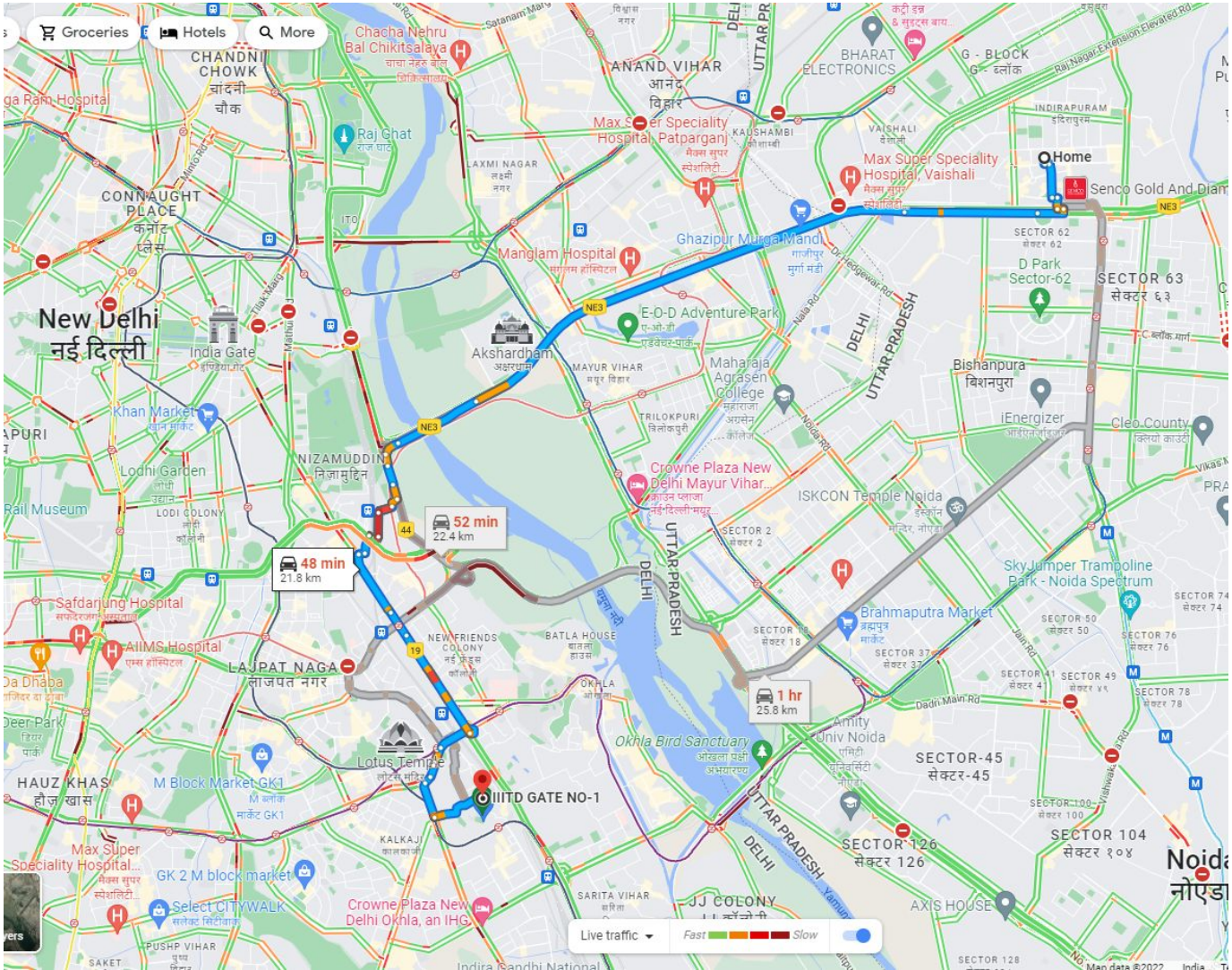
Weighted graphs

- Graphs where a ”weight” is associated with every edge
- Weight may reflect ”distance”, ”delay”, ”cost”, ”congestion”, etc.
  - Helps evaluate ”shortest paths”



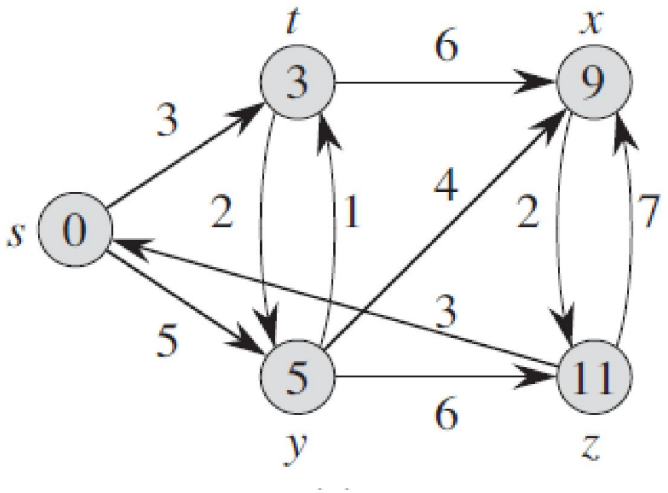
Weighted graphs

- Graphs where a ”weight” is associated with every edge
- Weight may reflect ”distance”, ”delay”, ”cost”, ”congestion”, etc.
  - Helps evaluate ”shortest paths”



Weighted graphs, and their representation

- Graphs where a ”weight” is associated with every edge



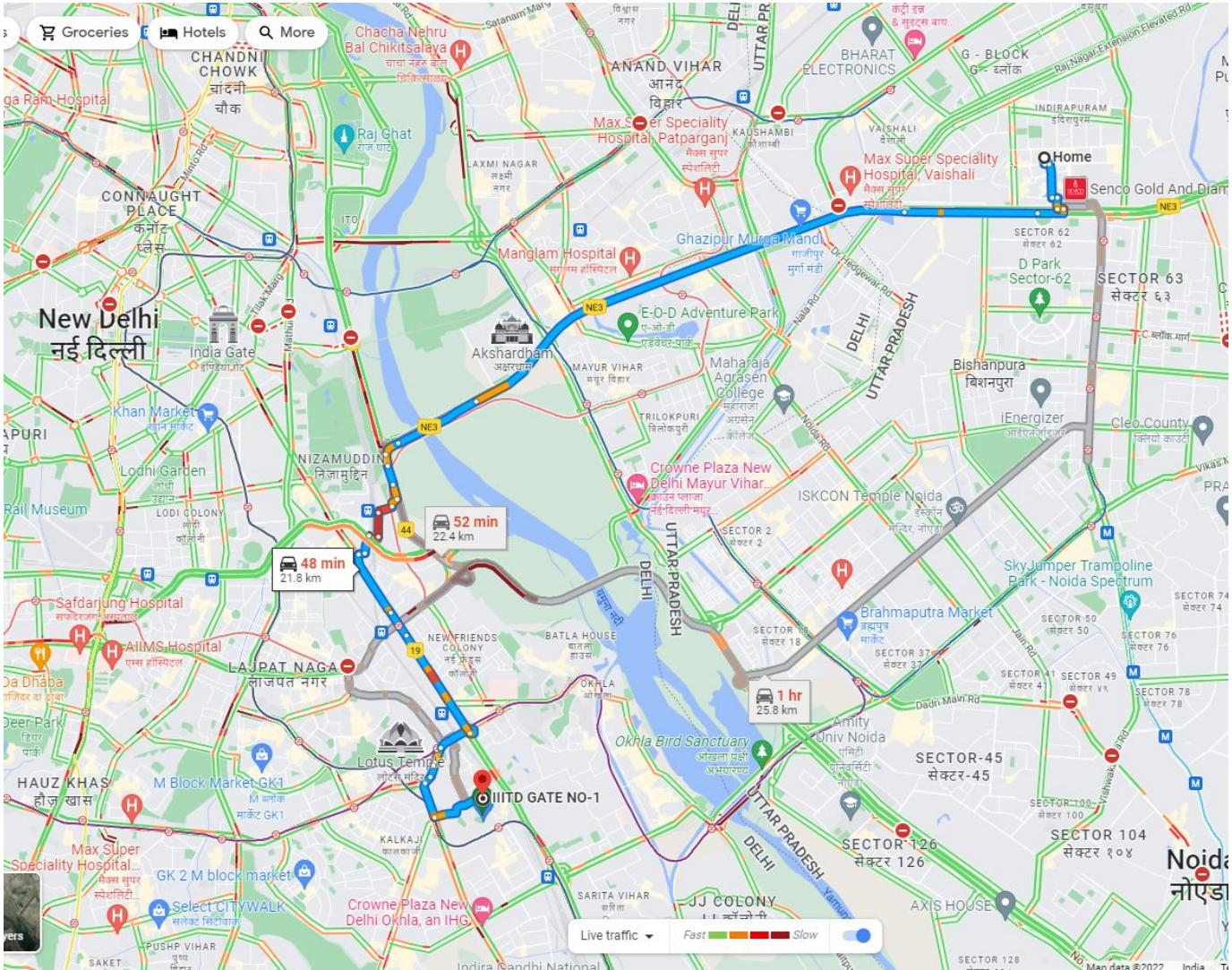
A =

	s	t	x	y	z
s	-	3	-	5	-
t	-	-	6	2	-
x	-	-	-	-	11
y	-	1	4	-	6
z	3	-	7	-	-

Some problems on graphs

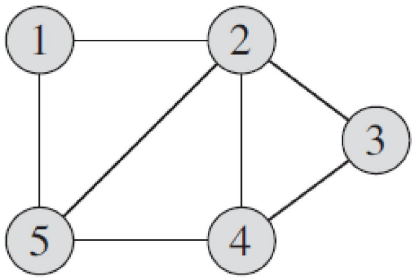
Shortest path algorithms

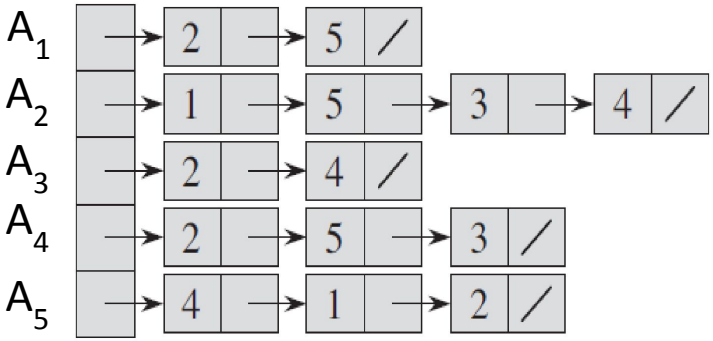
- One-to-one
- One-to-many
- Many-to-many



Traversal of graphs

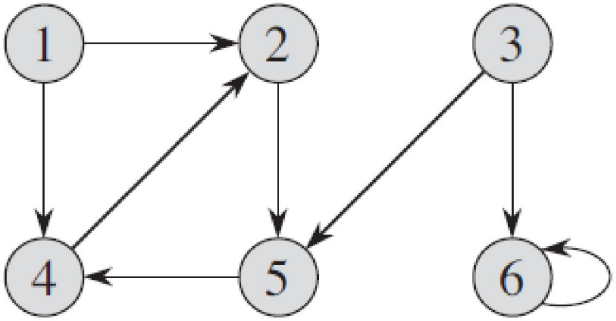
- Viz. search for a vertex, or a path that satisfies a certain property, or is an “optimum”
  - Through an undirected graph or through a directed graph
- Different search techniques:
  - BFS
  - DFS
  - others

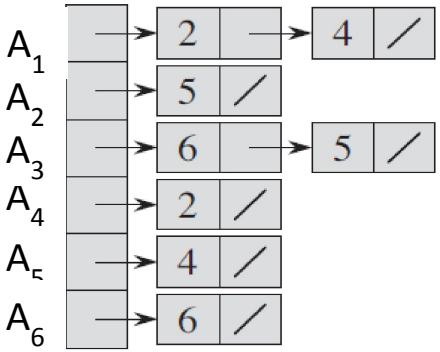




A =

0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0





A =

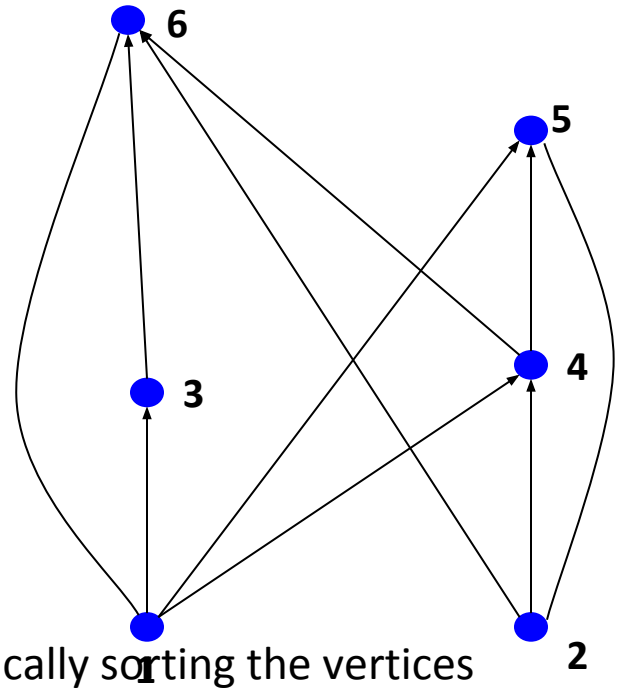
0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

Topological sort

- Consider the following executable statements:

- S_1: a=0;
- S_2: b=1;
- S_3: c=a+1
- S_4: d=b+a;
- S_5: e=d+1;
- S_6: e=c+d;

Nodes = statements
Edges = precedence requirements

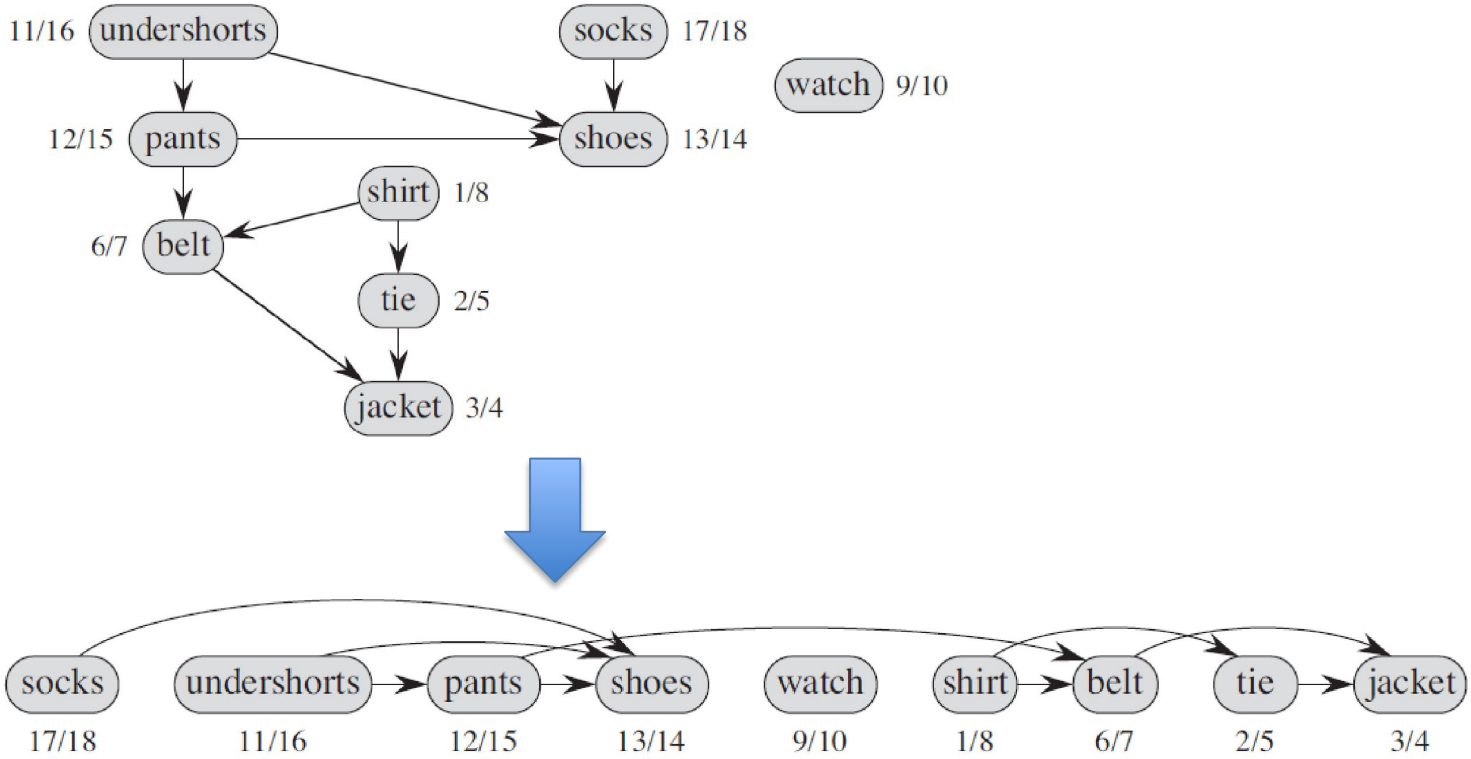


- Questions:

1. Which statements must execute before, for example, S_6?
2. Which statements can be executed in parallel?

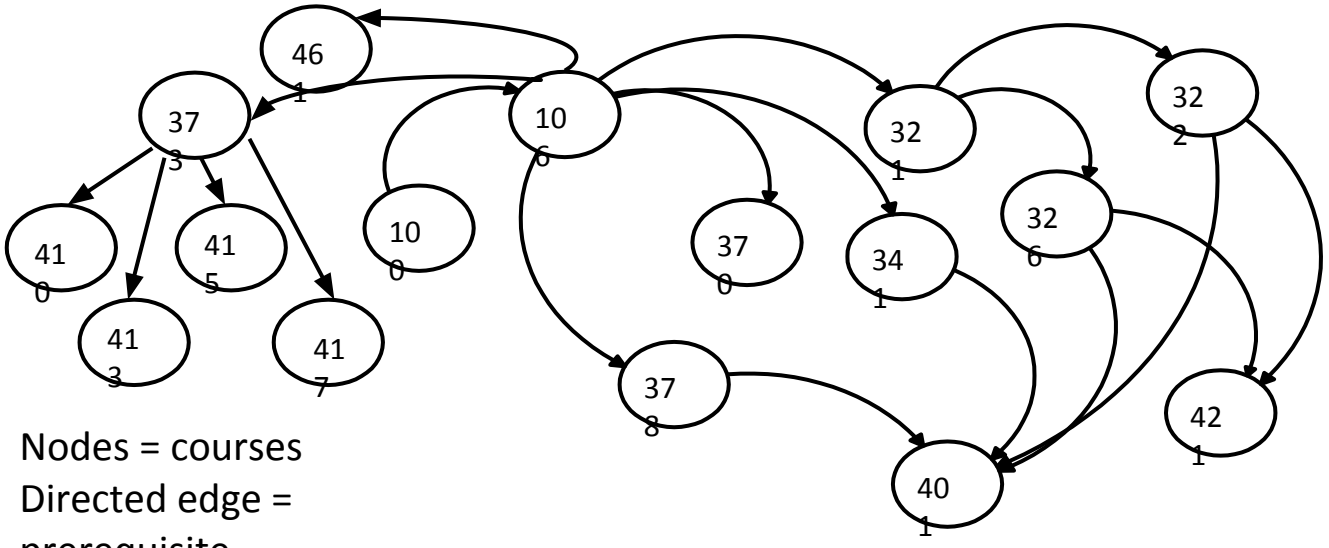
- Answers lie in studying the adjoining graph, G, and topologically sorting the vertices

Topological sort



Topological sort

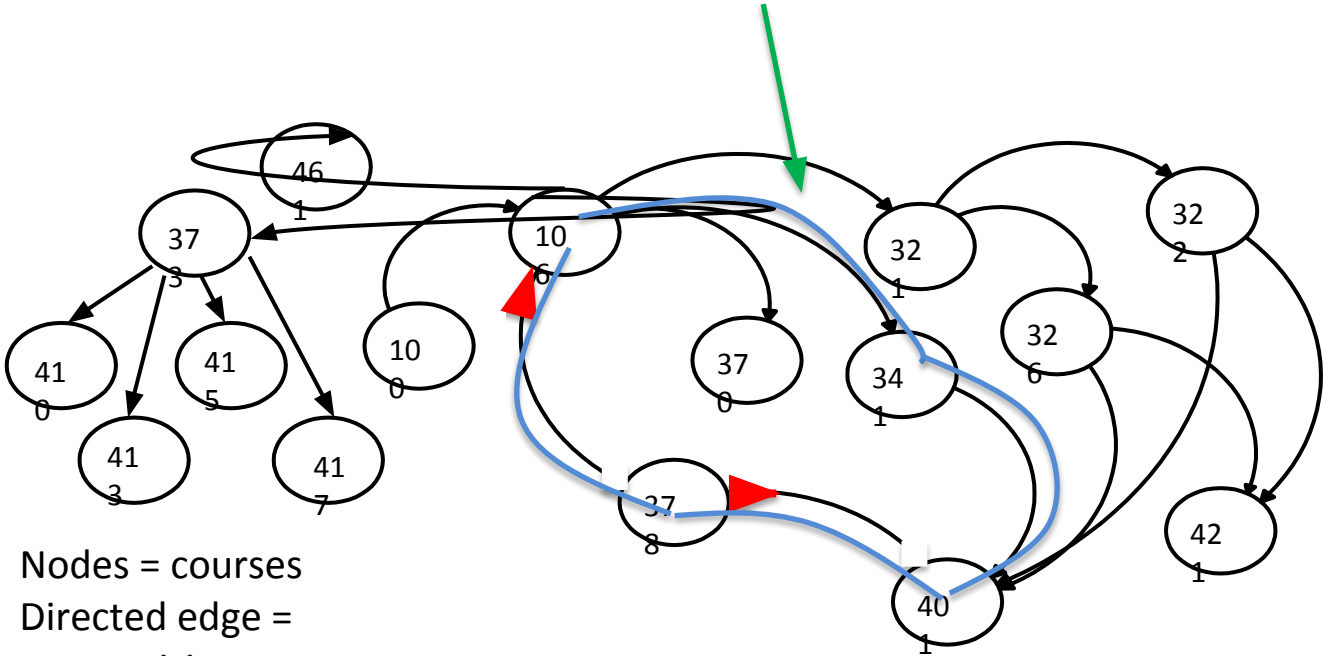
- In what possible sequence should one complete the courses so as to complete the program (consisting of these 16 courses)



Nodes = courses
Directed edge = prerequisite

Directed acyclic graphs,or DAGs

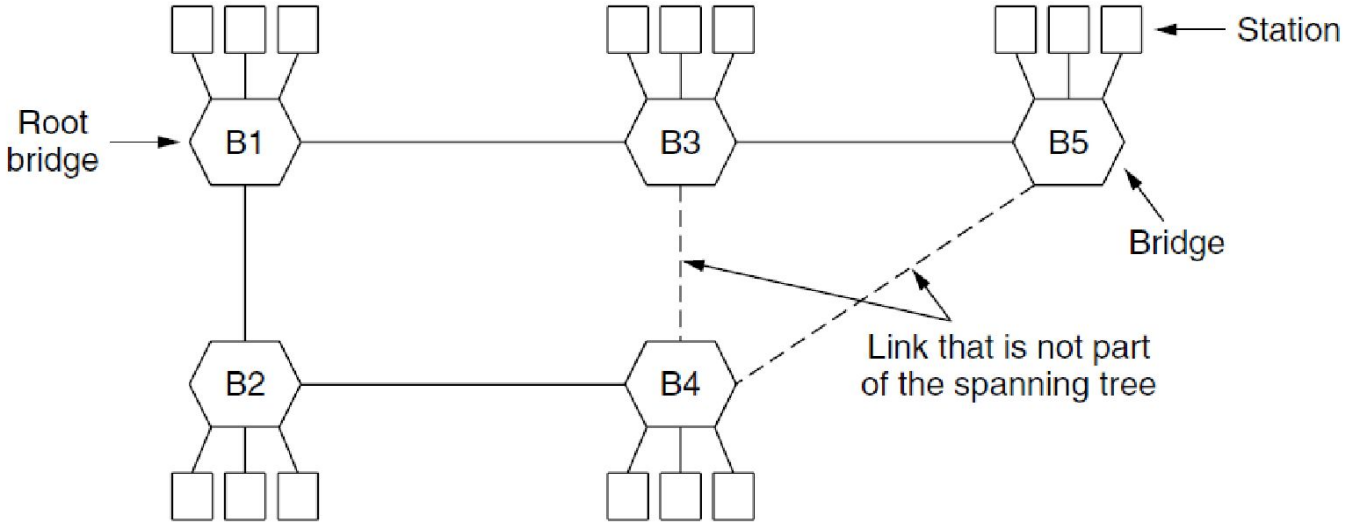
- Consider an 'directed' graph, G = (V, E), which has "cycles", as in diagram below showing pre-requisites
  - How does one complete the coursework
  - Vertices in a DAG (directed acyclic graph) only can be sorted topologically



Nodes = courses
Directed edge = prerequisite

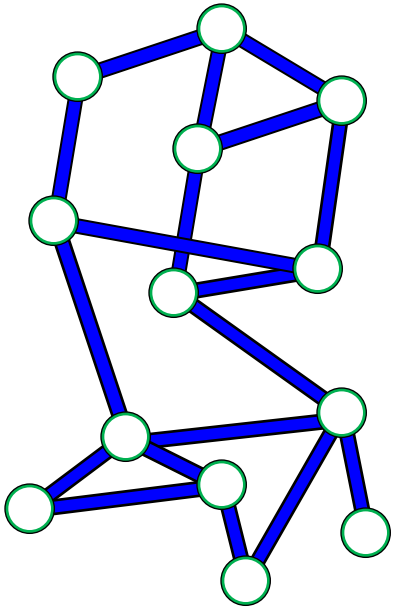
Other problems

- Minimum spanning tree
  - Spanning Tree algorithm applied to network of routers or bridges discovered by Radia Perlman in late 1970s/early 1980s

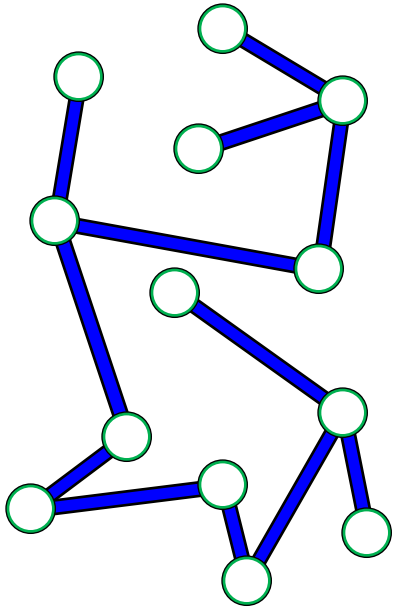


Other problems

- Minimum spanning tree
  - Another example
  - What can you say about fault-tolerance of a spanning tree



Graph G



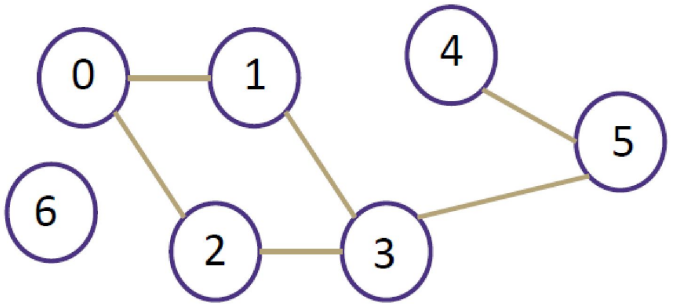
Spanning tree for Graph G

Q&A

Check these examples

- Courtesy https://courses.cs.washington.edu/courses/cse373/22sp/

Adjacency matrix or lists of directed/undirected graphs



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