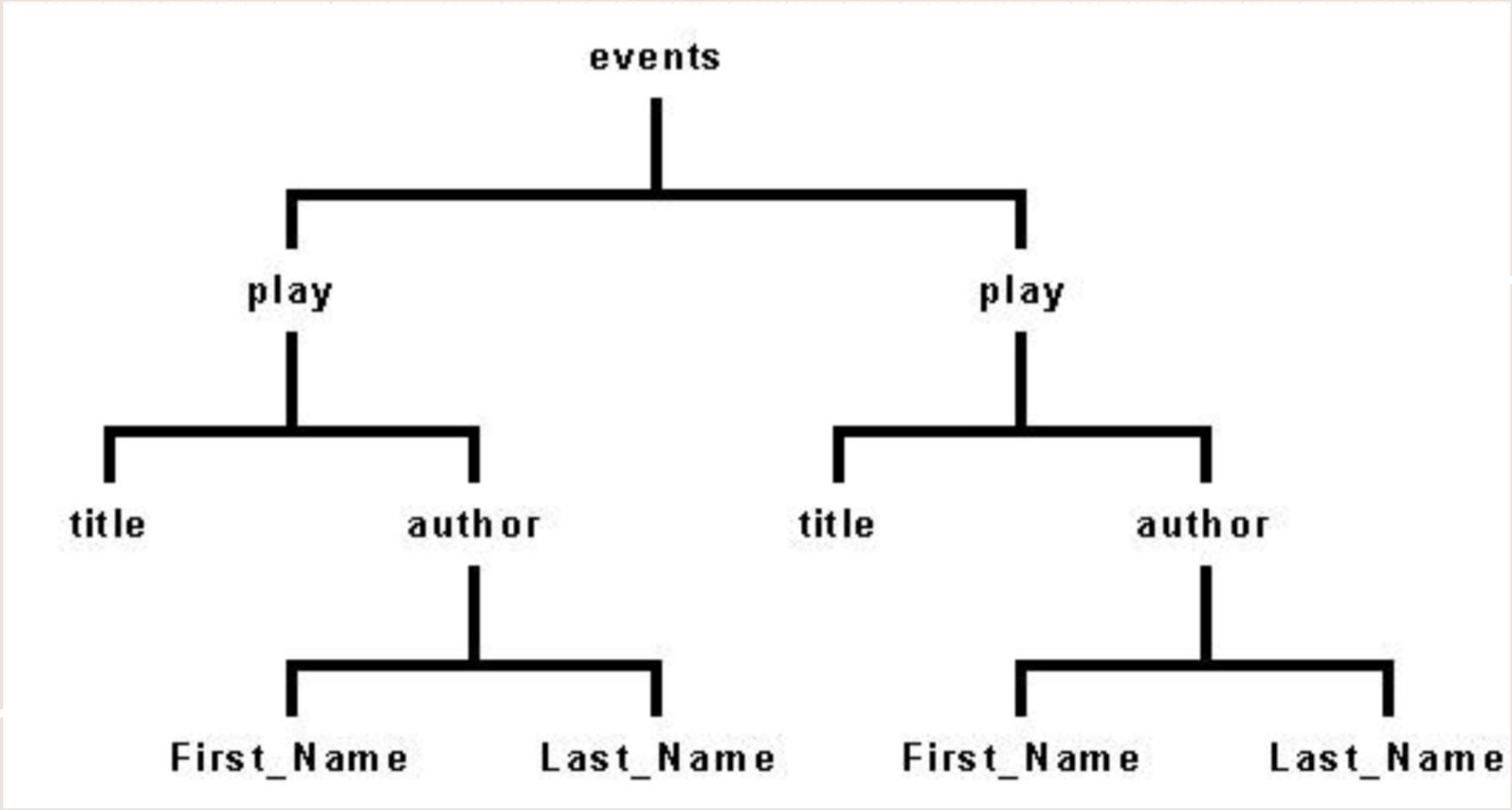
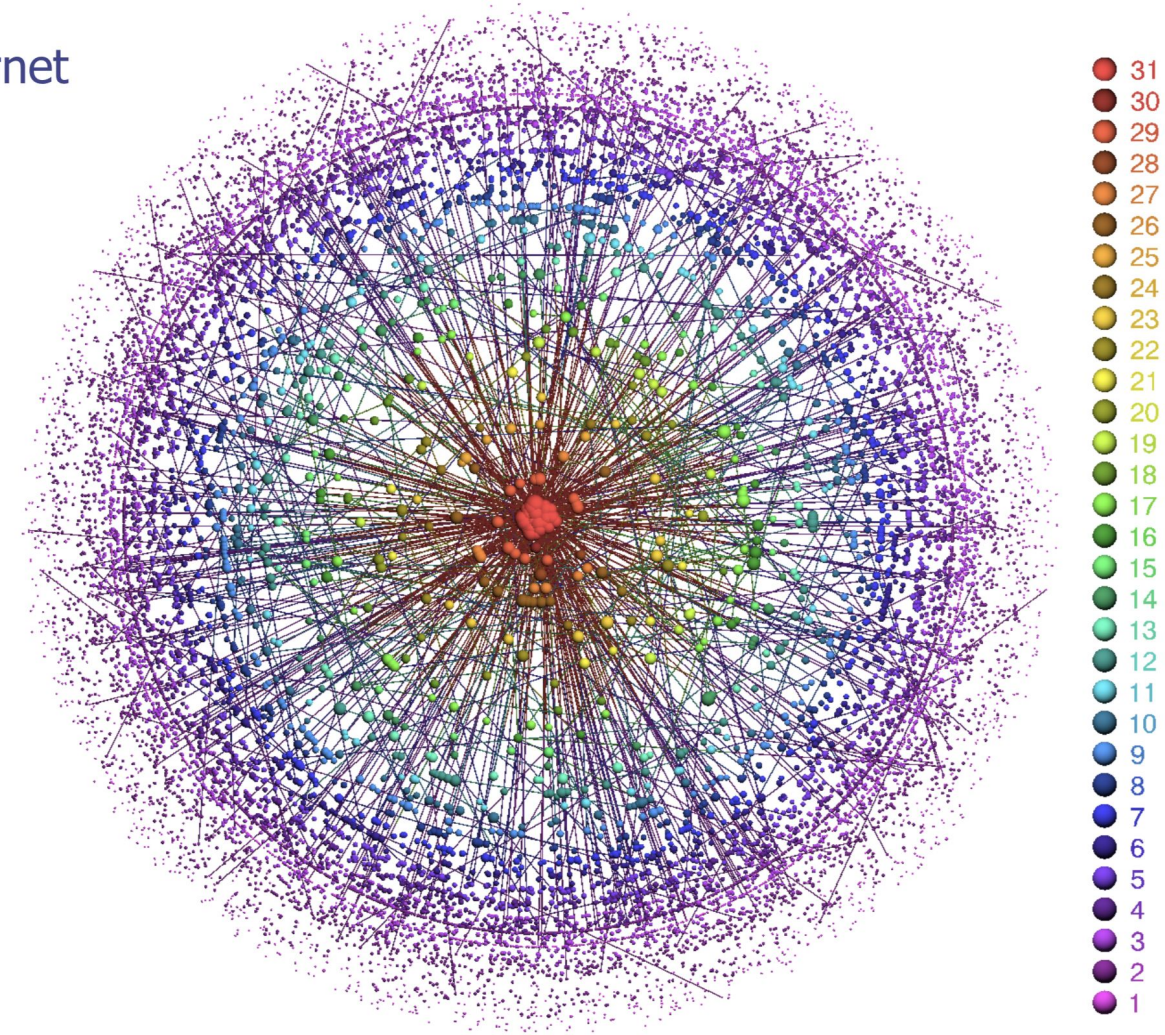
Introduction to Graphs

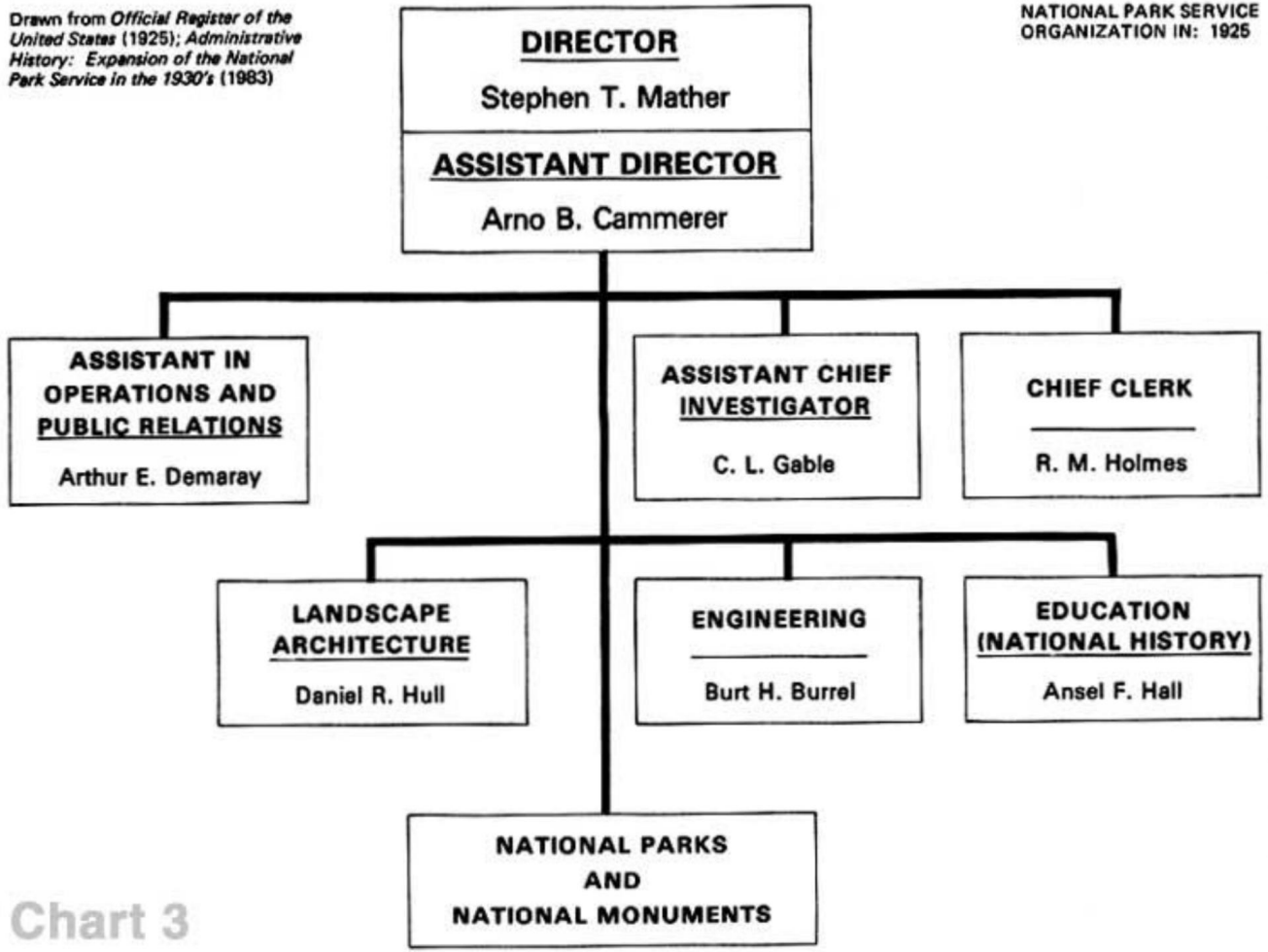
- Applications
- Terminology
- Graph API
- Implementations


events
play
play
title
author
title
author
First_Name
Last_Name
First_Name
Last_Name

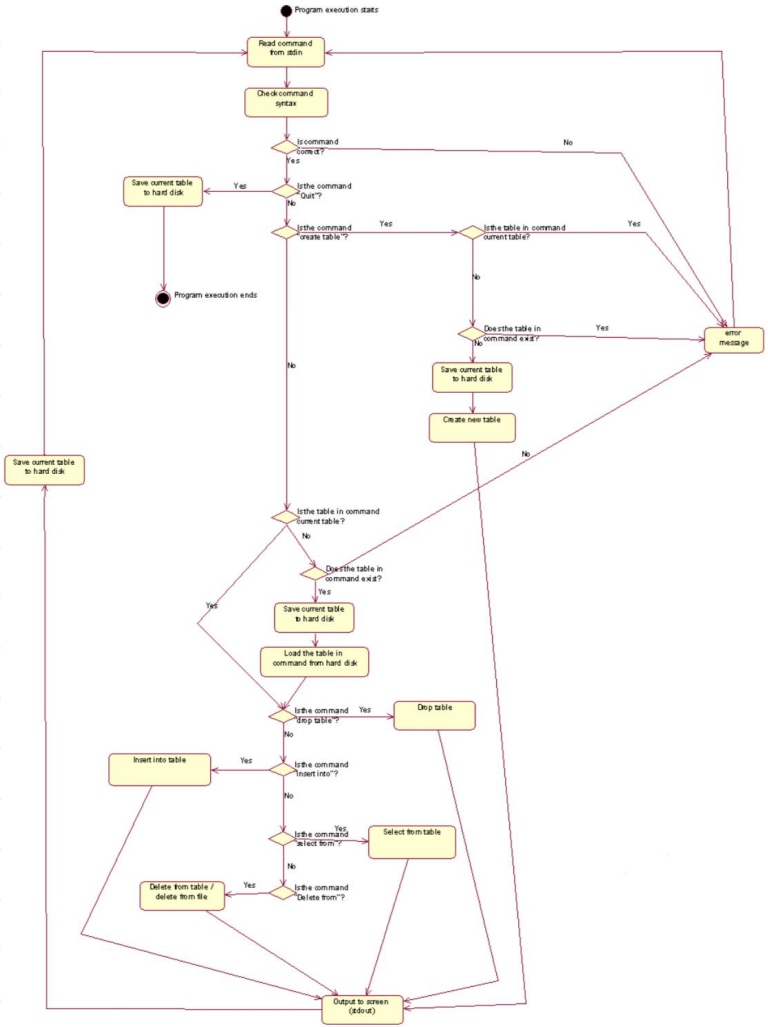
Internet


31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1

Organizational chart

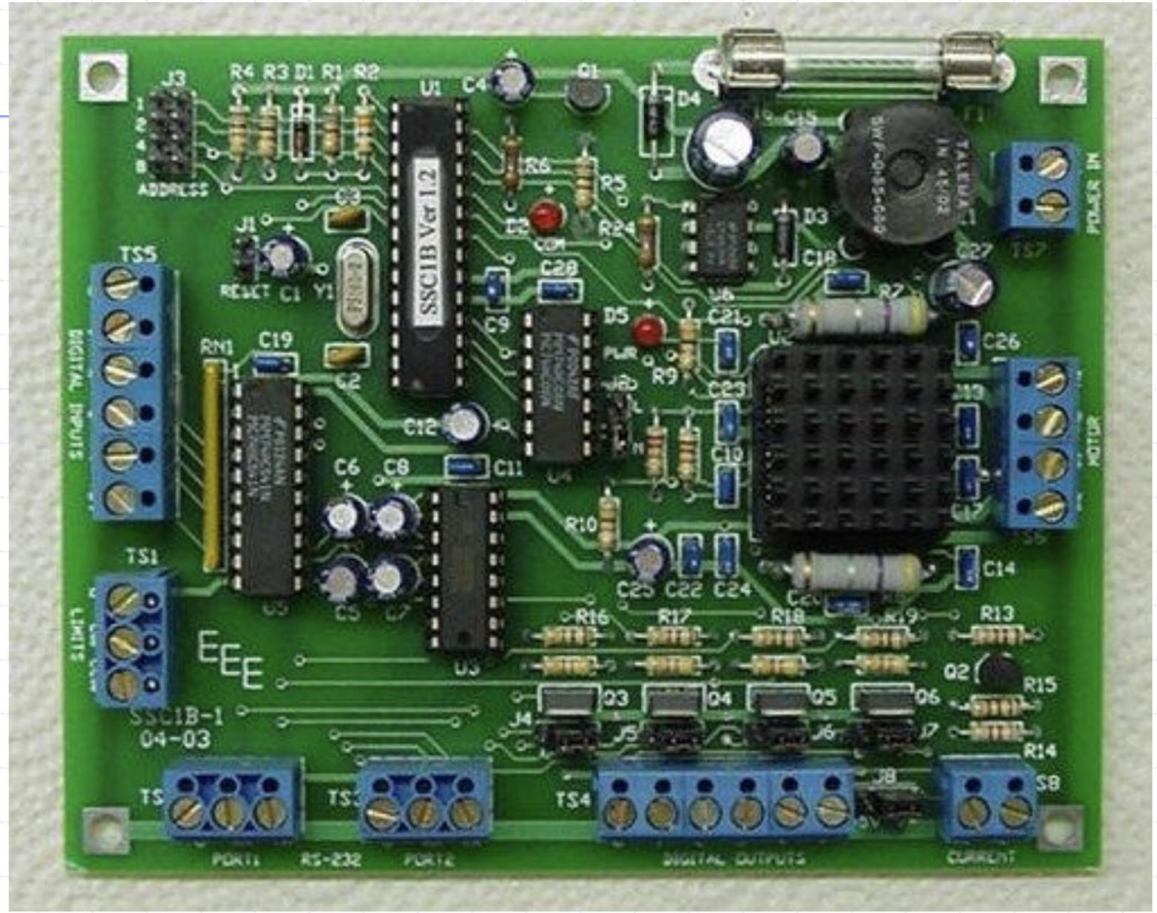


Program Flow Chart

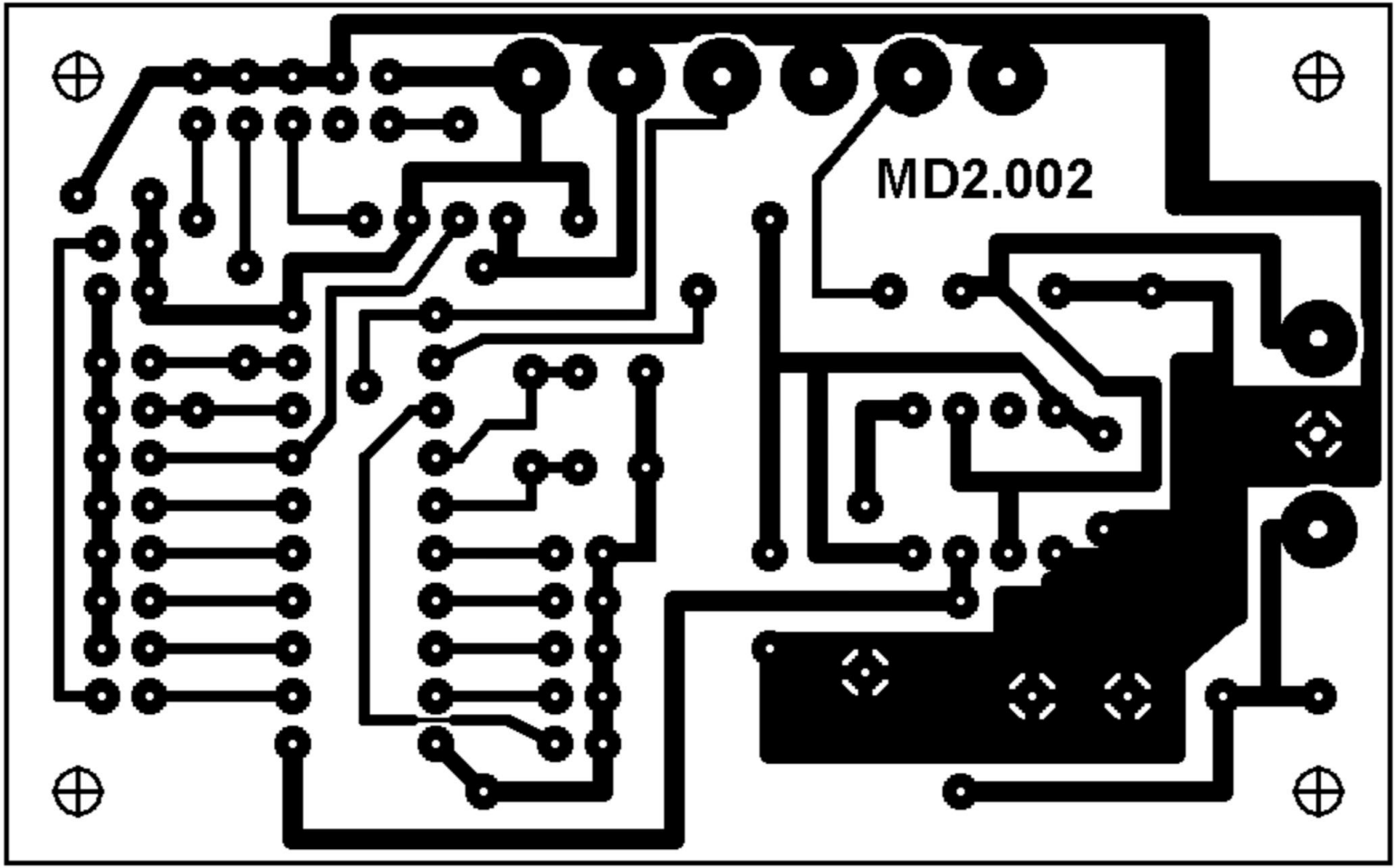


Flight Routes

Electronic Circuit Board



Electronic Circuit Board



What do these all have in common?

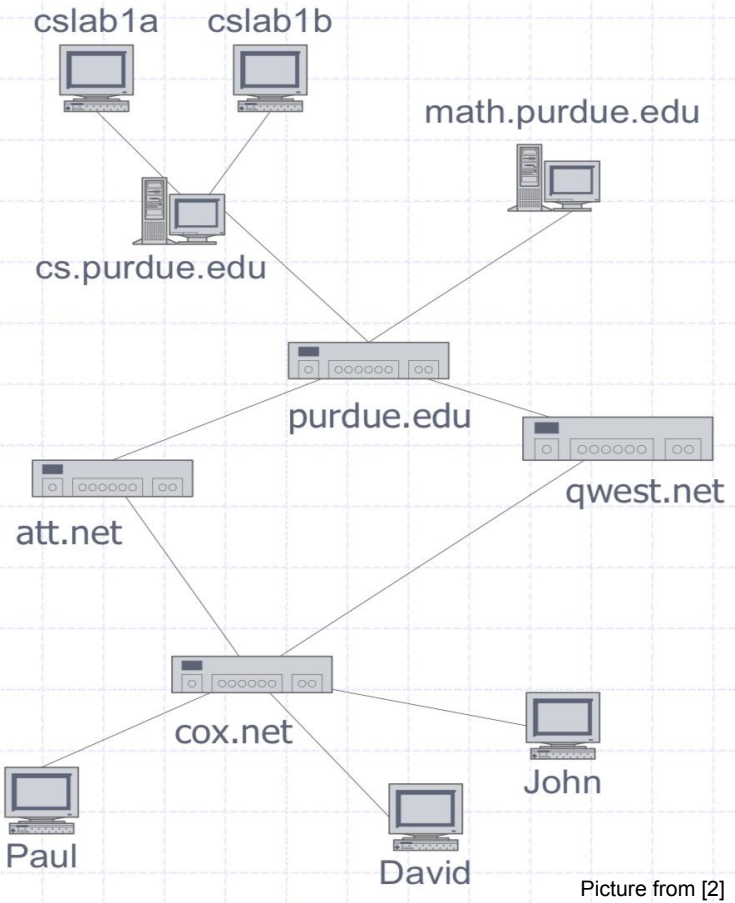
They can all be represented as **graphs**.

Common questions we might answer using graphs:

- What is the **shortest path** between point A and point B?
- Is it possible to reach another location from point A? **(reachability)**
- What is the most efficient way to visit all locations exactly once?
- What is the best way to connect all the nodes while minimizing other characteristics of the graph? **(MST)**
- Is it possible to put the nodes in order so that all the edges are pointing in the same direction? **(topological sorting)**

Applications

- Computer networks
  - Local area network
  - Internet
  - Web
- Electronic circuits
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  - Highway network
  - Flight network
- Databases
  - Entity-relationship diagram

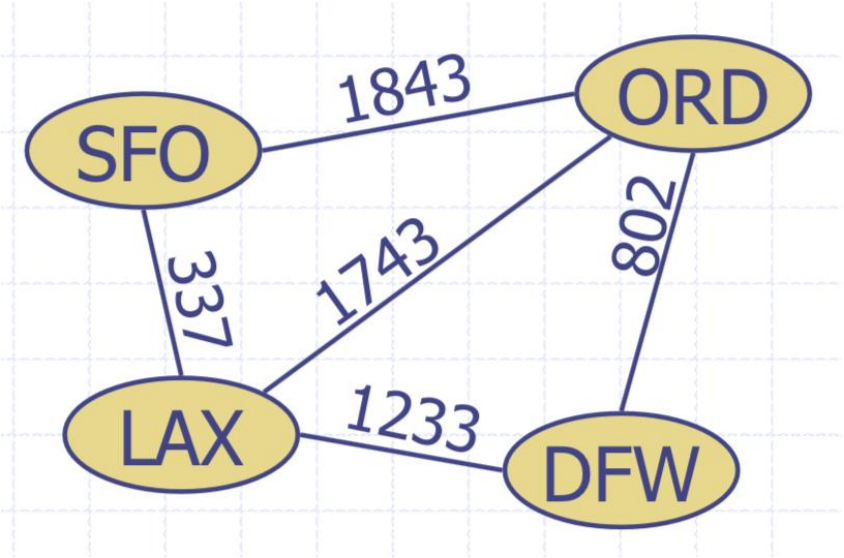


Picture from [2]

What is a graph?

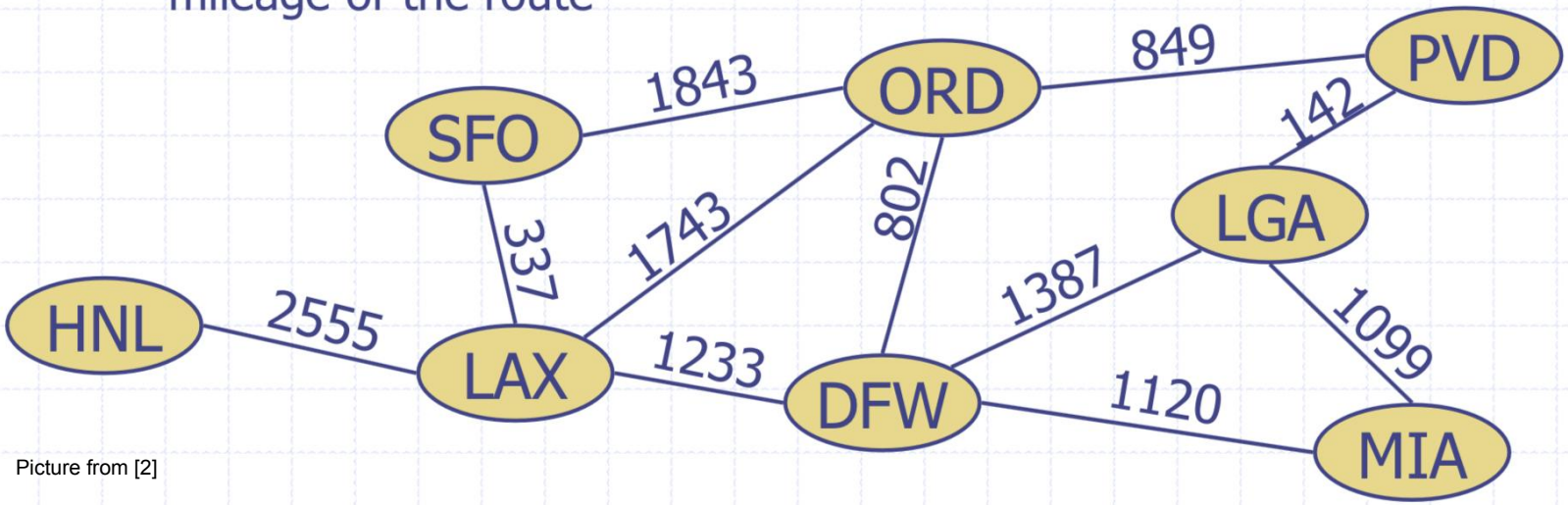
A ***graph*** is a set of ***vertices*** and a collection of ***edges*** that connect a pair of vertices.

(You've already seen quite a few graphs in this class…)



Picture from [2]

- A graph is a pair (V, E), where
  - V is a set of nodes, called vertices
  - E is a collection of pairs of vertices, called edges
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route



Picture from [2]

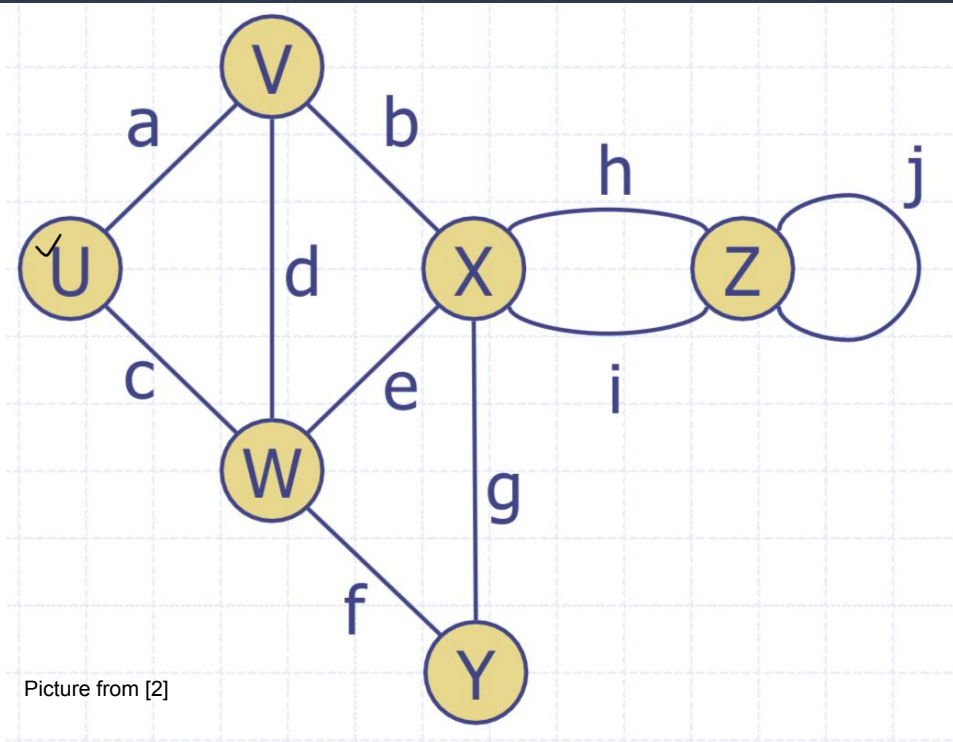
Edge Types

- Directed edge
  - ordered pair of vertices (u,v)
  - first vertex u is the origin
  - second vertex v is the destination
  - e.g., a flight
- Undirected edge
  - unordered pair of vertices (u,v)
  - e.g., a flight route
- Directed graph
  - all the edges are directed
  - e.g., route network
- Undirected graph
  - all the edges are undirected
  - e.g., flight network

Picture from [2]

Terminology

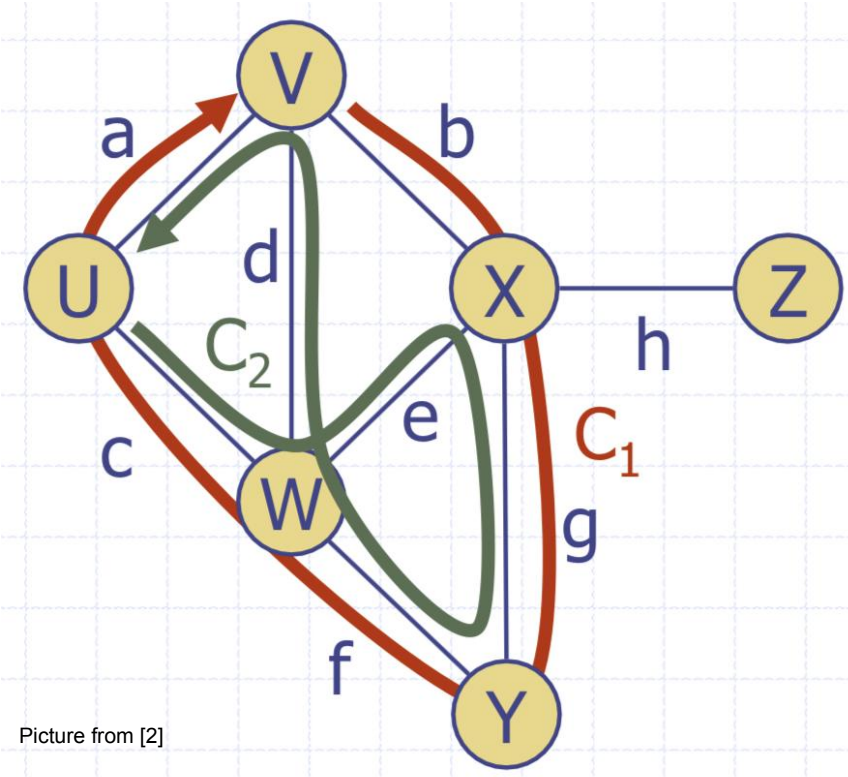
- **endpoints** (of an edge): the vertices connected by that edge (e.g. *V* and *X* are the endpoints of *b*)
- **incident (on)**: when an edge touches a vertex (e.g. *a* is incident on *V*)
- **adjacent (to)**: when two vertices are connected by a single edge (e.g. *V* and *X* are adjacent)
- **degree** (of a vertex): the number of edges incident on that vertex (e.g. the degree of *V* is 3)
- **parallel edges**: edges with the same endpoints (e.g. *h* and *i*)
- **self-loop**: an edge that starts and ends at the same vertex (e.g. *j*)
- **simple graph**: a graph with no parallel edges or self loops (vs. **multigraph)**



Picture from [2]

More Terminology

- **path**: sequence of vertices connected by edges
- **simple path**: a path with no repeated edges or vertices
- **cycle**: a circular sequence of vertices connected by edges
- **simple cycle**: a cycle that has no repeated edges and no repeated vertices (except the first and last, making it a cycle)
- **length** (of a path or cycle): the number of edges included in the path or cycle
- **subgraph**: a subset of a graph's edges (and associated vertices)
- **connected graph**: a graph where there exists a path connecting any two pairs of vertices
- **tree**: an acyclic connected graph



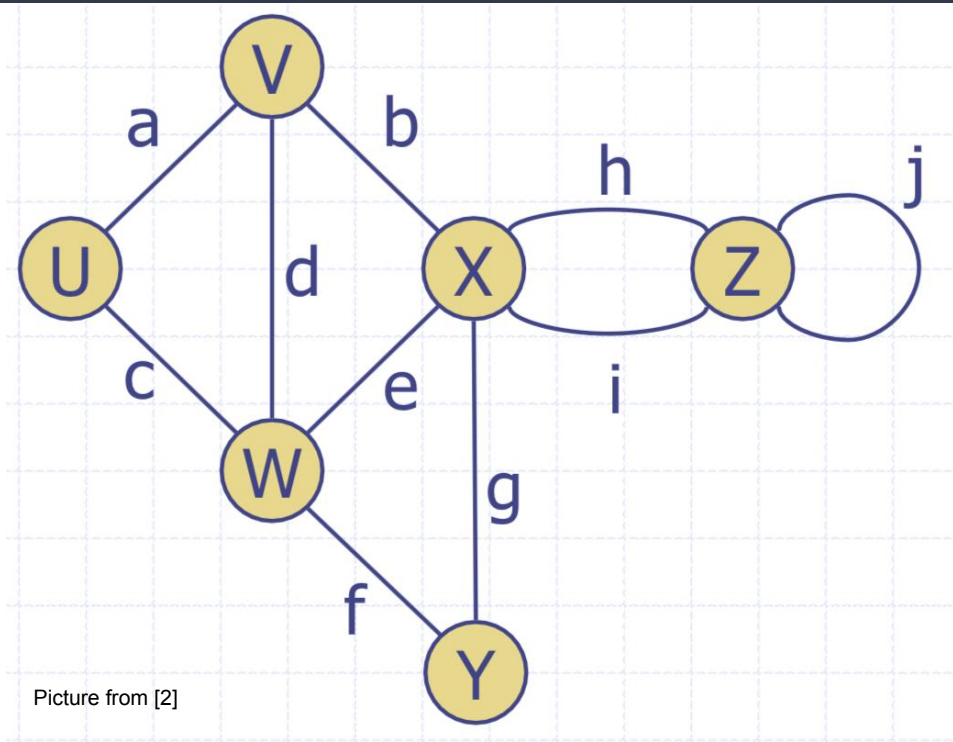
Picture from [2]

Properties of Graphs

Property 1: Degree Sum

Given a graph with **V** vertices and **E** edges, what is the relationship between the sum of the degrees of all the vertices and the number of edges?

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



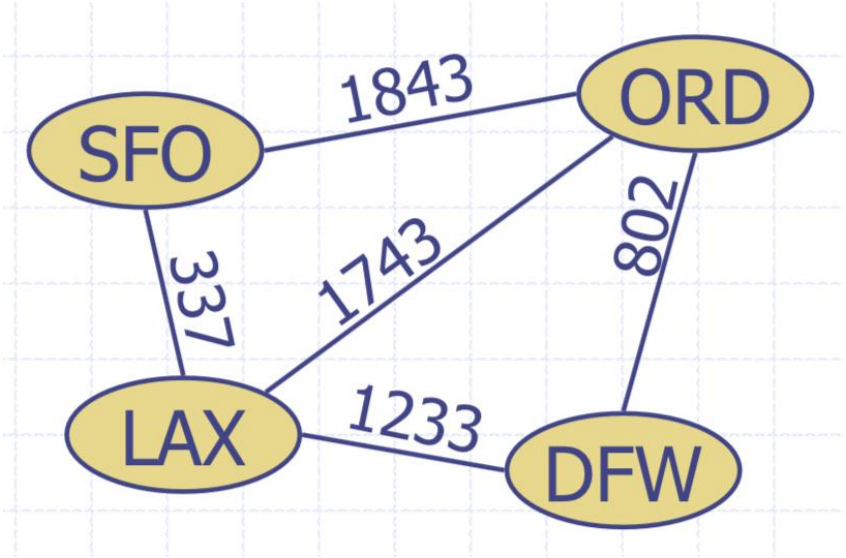
Picture from [2]

Properties of Graphs

Property 2: Maximum Degree

Given a simple, undirected graph with **V** vertices and **E** edges, what is the maximum possible degree of any vertex?

$V - 1$



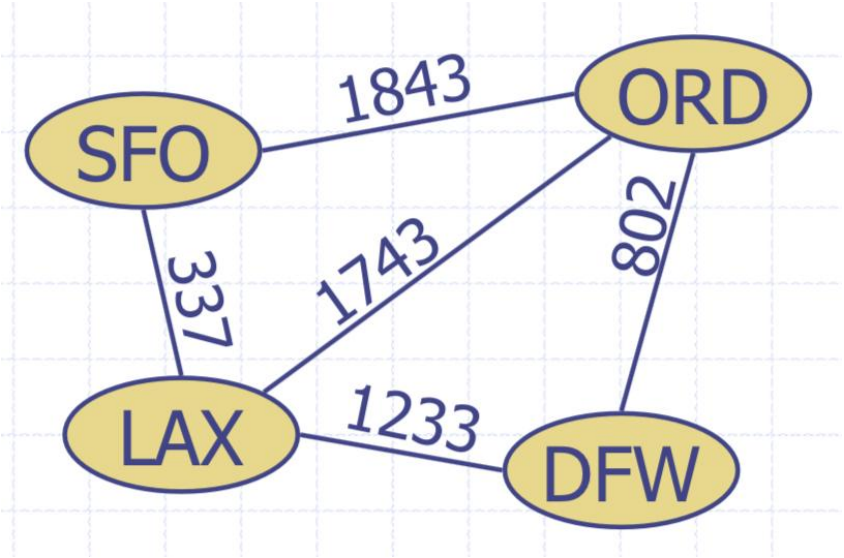
Pictures from [2]

Properties of Graphs

Property 3: Edge Count

Given a simple, undirected graph with **V** vertices and **E** edges, what is the maximum possible number of edges in terms of **V**?

$$\frac{V(V-1)}{2}$$



Pictures from [2]

Undirected Graph API

public class	Graph	
	Graph(int V)	*create a V-vertex graph with no edges*
	Graph(In in)	*read a graph from input stream* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge* v-w *to this graph*
Iterable<Integer>	adj(int v)	*vertices adjacent to* v
String	toString()	*string representation*

API for an undirected graph

Picture from [3]

Graph Implementation

- What do we care about when implementing a graph?

  Space

  time

- How can we implement a graph?

Graph Implementation

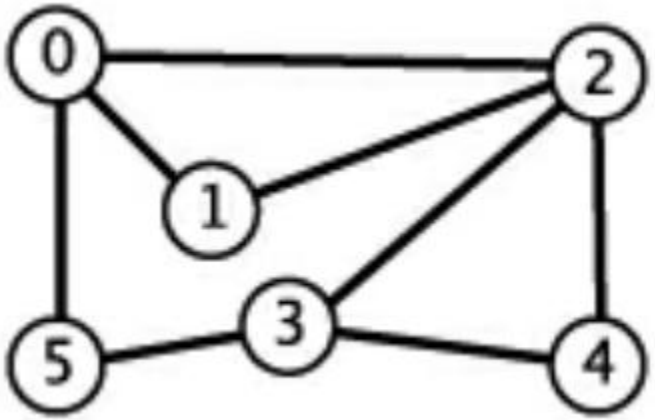
- What do we care about when implementing a graph?
  - Space (efficient usage of memory)
  - Time (efficient implementation of the operations)

- How can we implement a graph?
  - Edge List
  - Adjacency Lists
  - Adjacency Matrix

Edge Lists: m edges, n vertices

- Represent the graph as a list of edges denoted by their endpoints

- EXAMPLE: [[0,1], [0,2], [0,5], [1,2], [2,3],[2,4], [3,4], [3,5]]



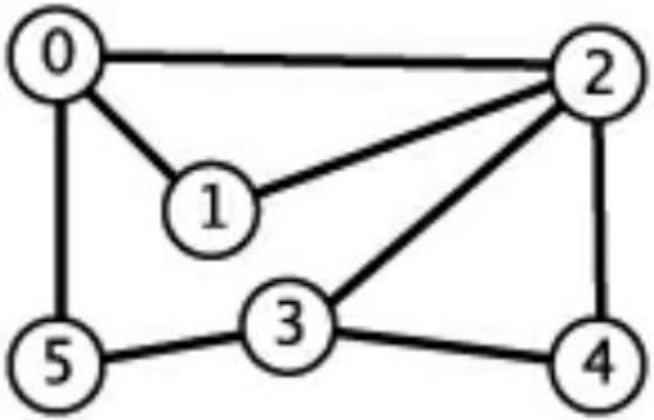
- Space: $O(E) = O(m)$
- Add edge: $O(1)$
- Check if two vertices are adjacent: $O(m) = O(E)$
- Iterate through the vertices adjacent to a vertex: $O(m) = O(E)$

Edge Lists: m edges, n vertices

- Represent the graph as a list of edges denoted by its endpoints

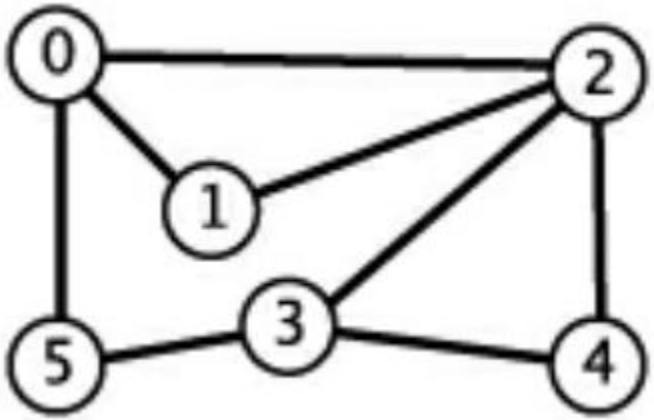
- EXAMPLE: [[0,1], [0,2], [0,5], [1,2], [2,3],[2,4], [3,4], [3,5]]

- Space: **O(m)**
- Add edge: **O(1)**
- Check if two vertices are adjacent: **O(m)**
- Iterate through the vertices adjacent to a vertex: **O(m)**



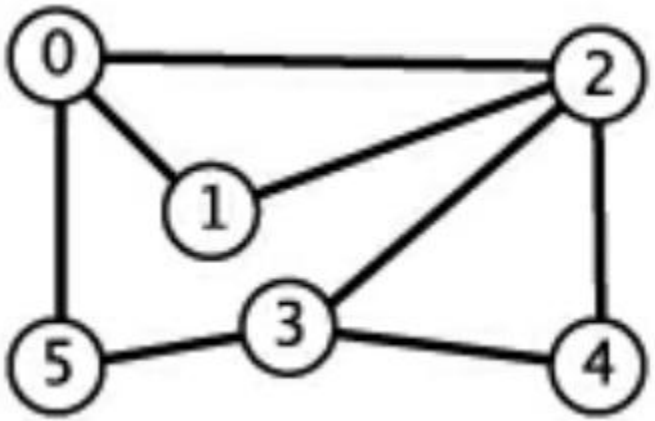
Adjacency Matrix: m edges, n vertices

Represent the graph as a matrix where the columns are vertices and the rows are vertices, and the elements are 0's or 1's depending on whether or not there is an edge between the vertices.



Adjacency Matrix: *m* edges, *n* vertices

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



Adjacency Matrix: m edges, n vertices

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

- Space: $O(n^2)$
- Add edge: $O(1)$
- Check if two vertices are adjacent: $O(1)$
- Iterate through the vertices adjacent to a vertex: $O(n)$

Adjacency Matrix: m edges, n vertices

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

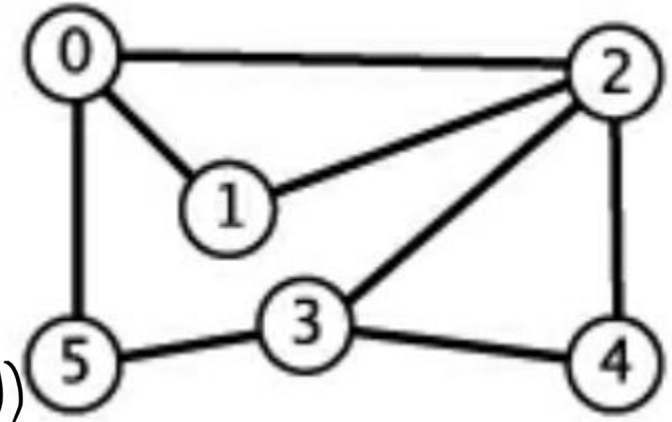
- Space: **$O(n^2)$**
- Add edge: **O(1)**
- Check if two vertices are adjacent: **O(1)**
- Iterate through the vertices adjacent to a vertex: **O(n)**

Adjacency Lists: m edges, n vertices

Represent the graph as an array of adjacency lists, where each list indicates the vertices adjacent to one of the **V** vertices.

0	1, 2, 5
1	0, 2
2	0, 1, 3, 4
3	2, 4, 5
4	2, 3
5	0, 3

- Space: $n + 2m$
- Add edge: $O(1)$
- Check if a vertex is adjacent to another vertex: $O(deg(v))$
- Iterate through the vertices adjacent to a vertex: $O(deg(v))$

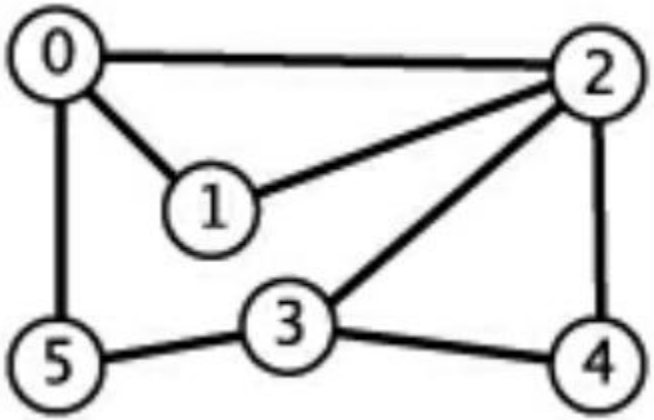


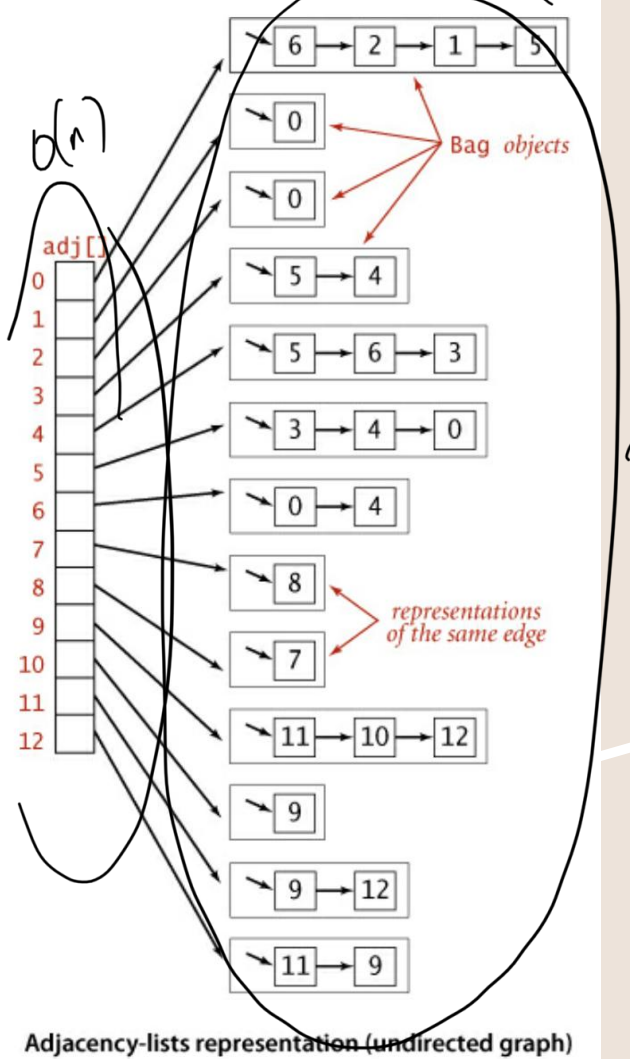
Adjacency Lists: *m* edges, *n* vertices

Represent the graph as an array of adjacency lists, where each list indicates the vertices adjacent to one of the **V** vertices.

0	1, 2, 5
1	0, 2
2	0, 1, 3, 4
3	2, 4, 5
4	2, 3
5	0, 3

- Space: **O(n+m)**
- Add edge: **O(1)**
- Check if a vertex **v** is adjacent to another vertex: **deg(v)**
- Iterate through the vertices adjacent to a vertex **v**: **deg(v)**





Adjacency-lists representation (undirected graph)

What does this graph look like?

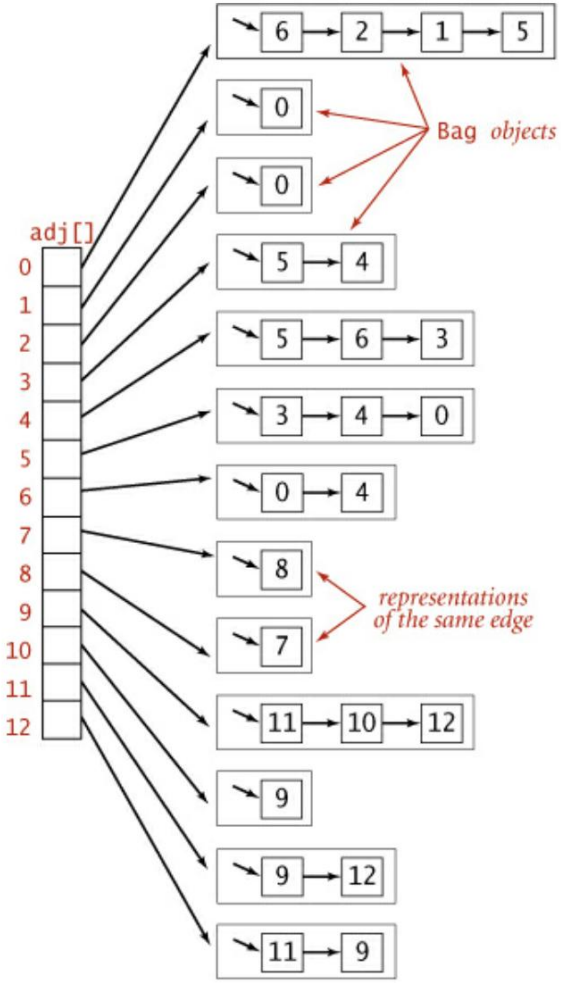
n^2

$\sim 2m$

$O(n+m)$

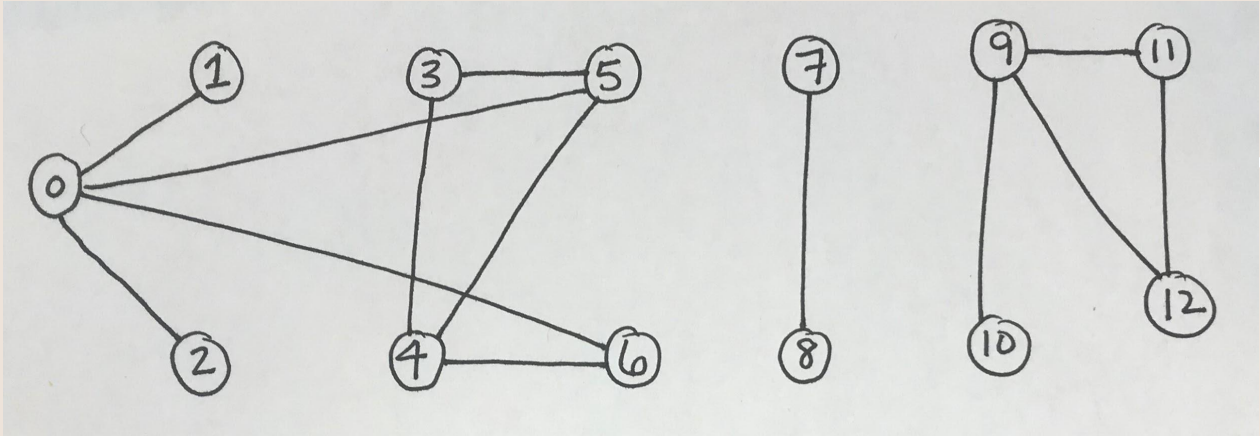
$\frac{n(n-1)}{2}$

$m \leq \frac{n(n-1)}{2}$ $O(n^2)$



Adjacency-lists representation (undirected graph)

What does this graph look like?

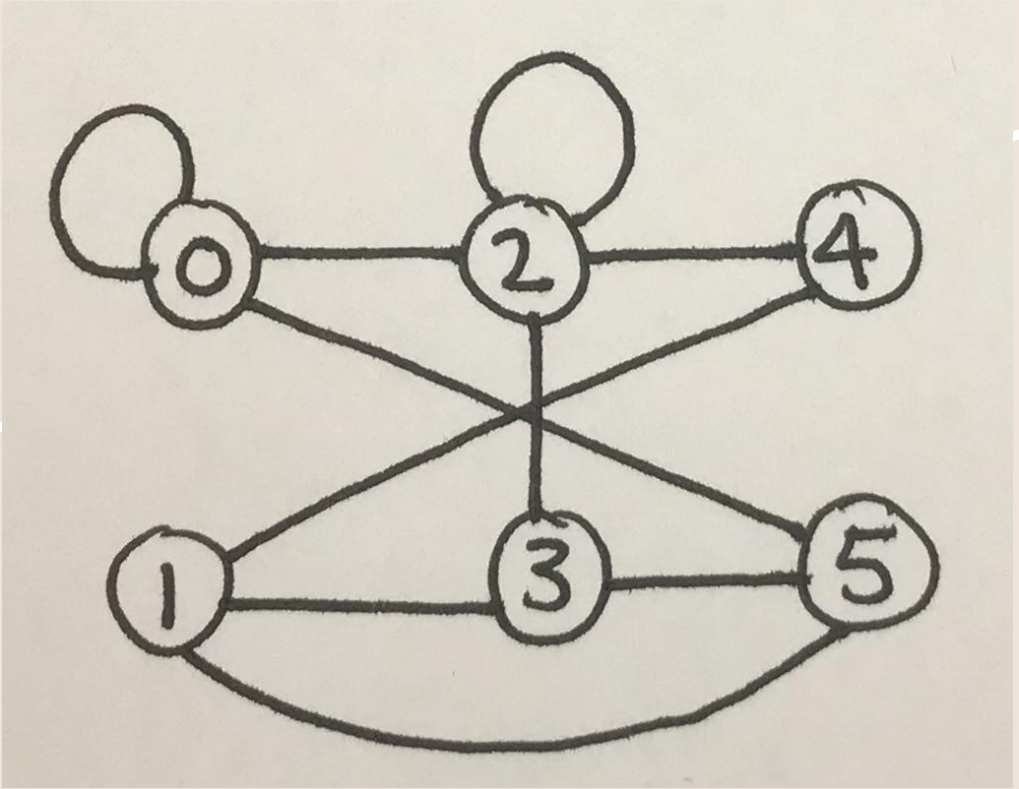


What does this graph look like?

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

What does this graph look like?

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

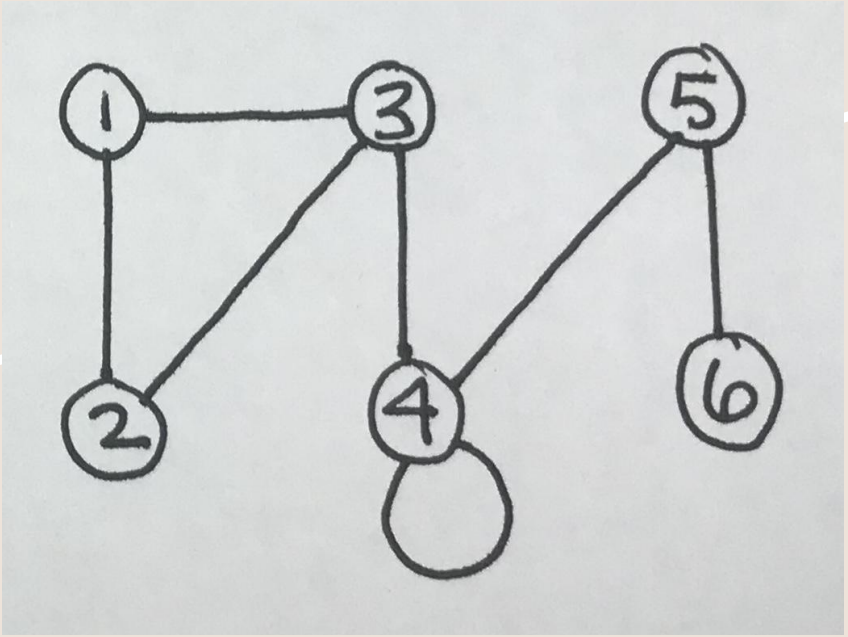


What does this graph look like?

[[3, 4], [6, 5], [1, 2], [4, 4], [3, 2], [1, 3], [4, 5]]

What does this graph look like?

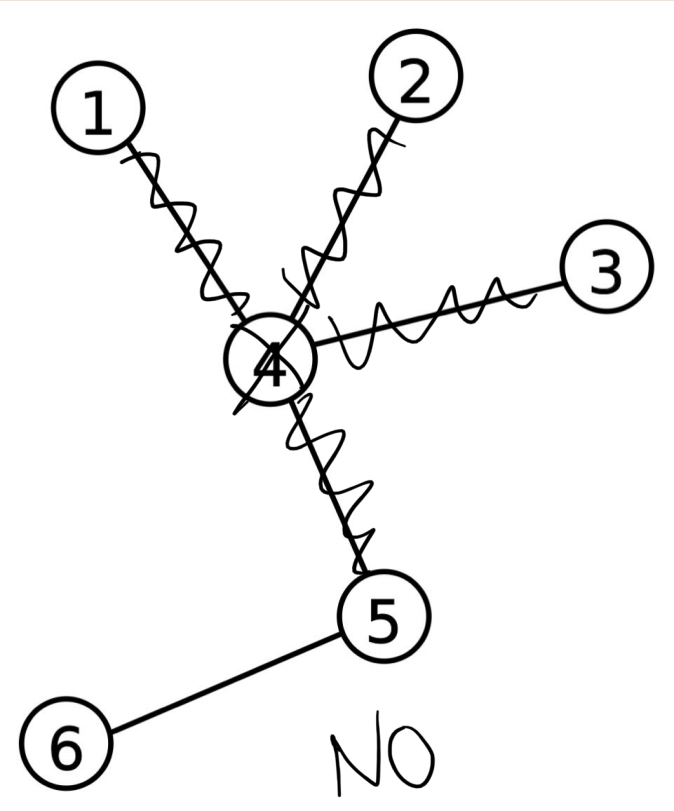
[[3, 4], [6, 5], [1, 2], [4, 4], [3, 2], [1, 3], [4, 5]]

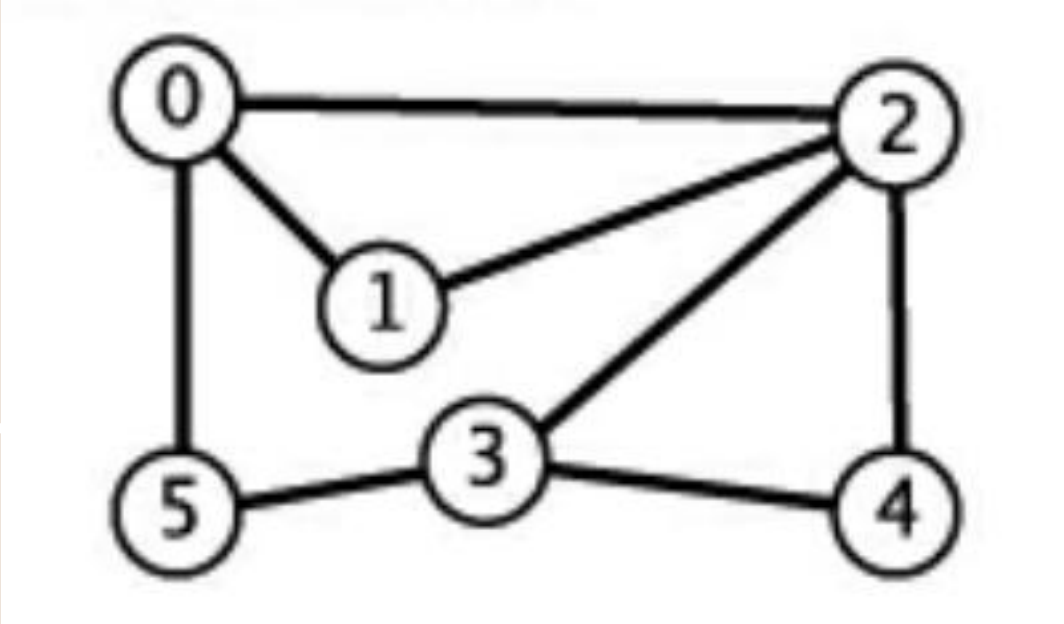


Biconnectivity

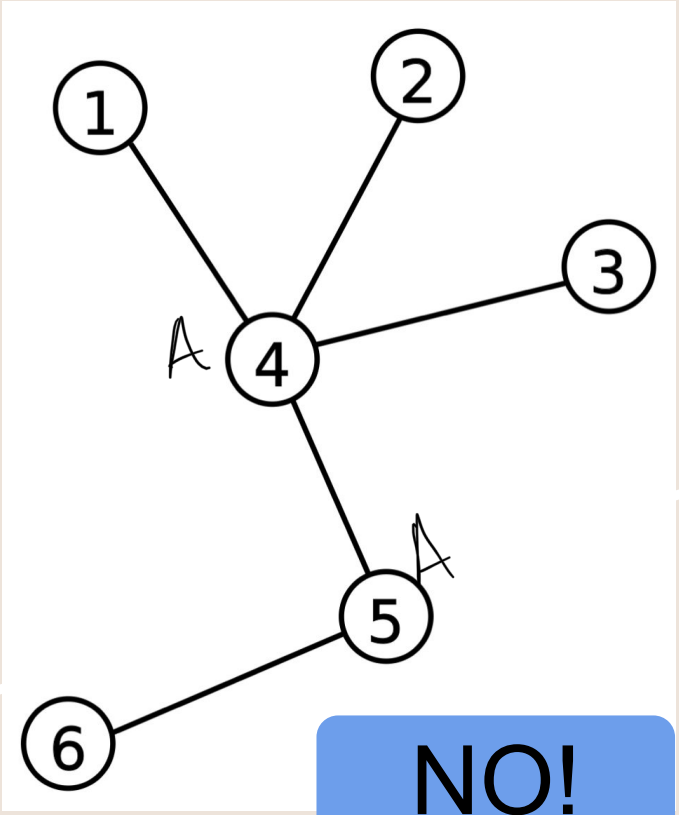
- A *biconnected* graph is one that is not separable. In other words, removing a single vertex (and associated edges) will not make the graph unconnected.
- Another way of putting this is that there are no *articulation vertices* (i.e. vertices that, if removed, will disconnect the graph).

Are these graphs biconnected?

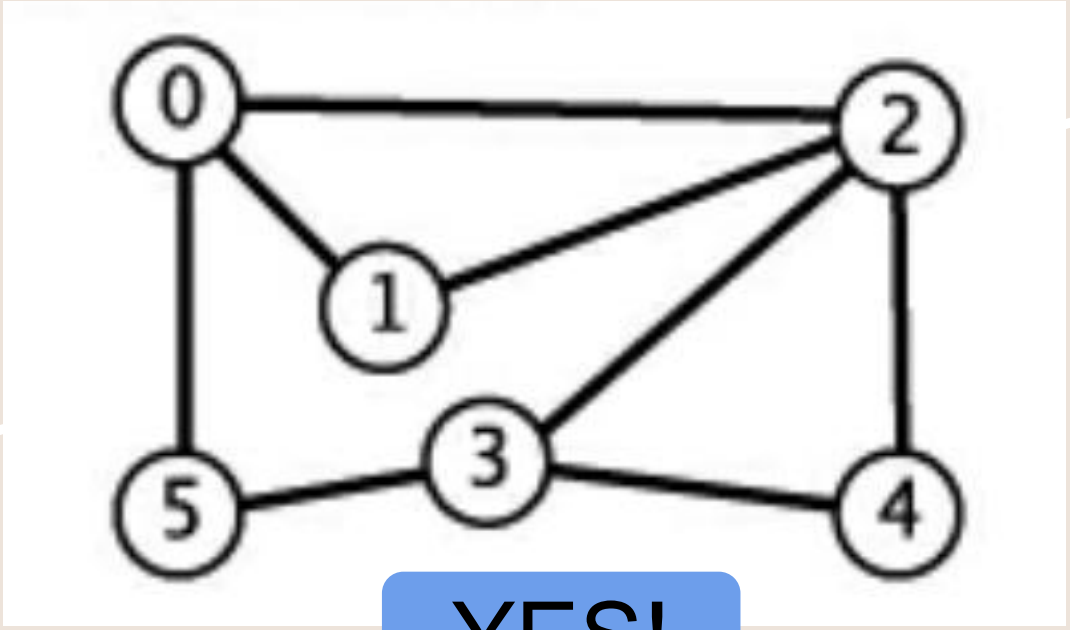




Are these graphs biconnected?



NO!



YES!

References

[1] https://optimization.mccormick.northwestern.edu/index.php/Traveling_salesman_problems
[2] Tamassia and Goodrich
[3] Sedgewick and Wayne