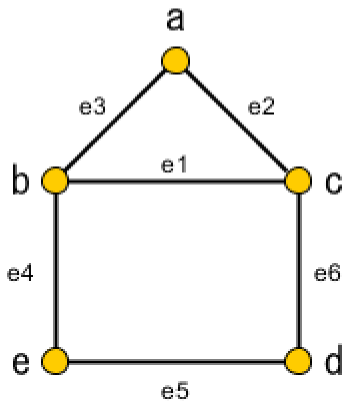
Introduction to Graphs

Graph

A graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ is a set V of vertices connected by an edge set E.

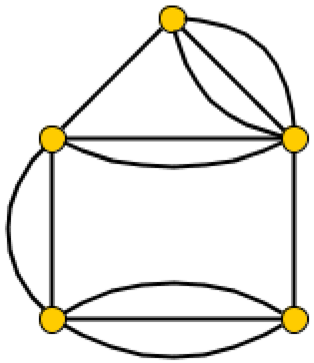


$V = \{a, b, c, d, e\}$

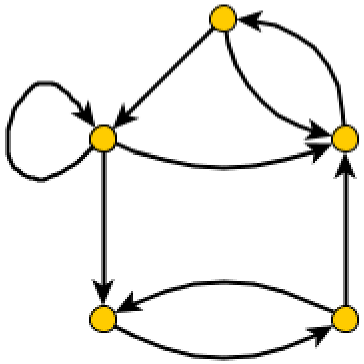
$E = \{e1, e2, e3, e4, e5, e6\} = \{(b, c), (c, a), (a, b), (b, e), (e, d), (d, c)\}$

Graph Variations

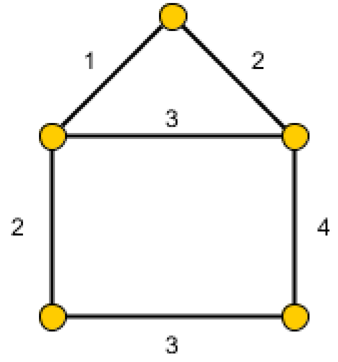
Multi-Graph: Multiple edges between two vertices.
Directed: Edges have a direction.
Weighted: Vertices and/or edges have weights.
Simple: No multiple edges, no loops.



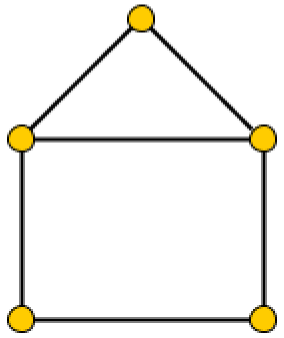
Multi-graph



Directed



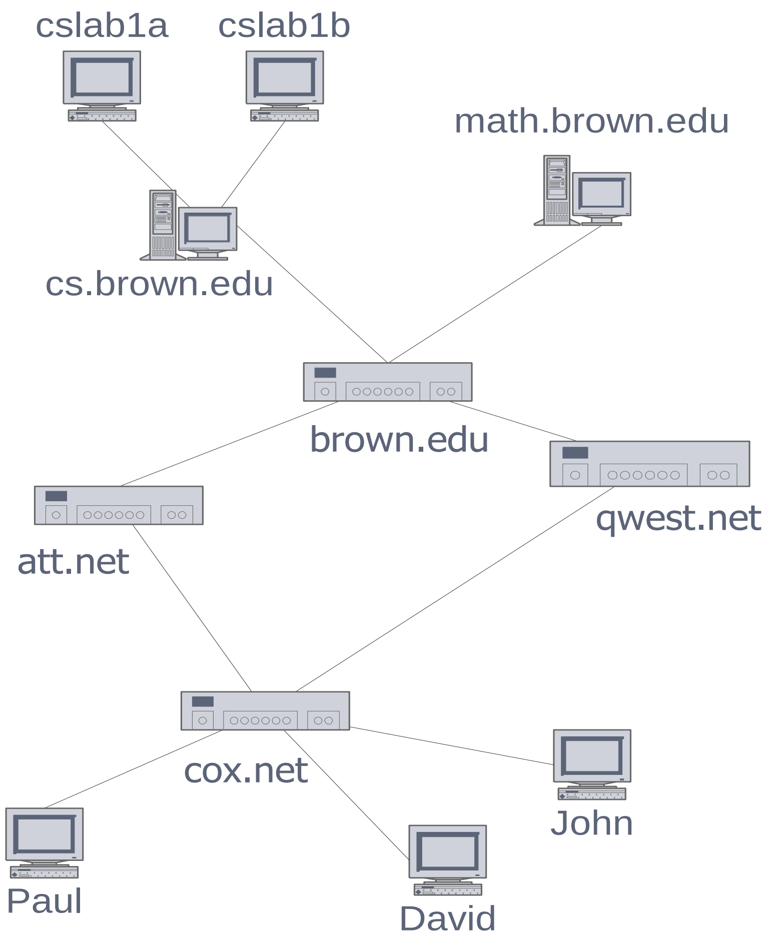
Weighted Undirected



Simple Undirected

Graph Applications

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



Examples

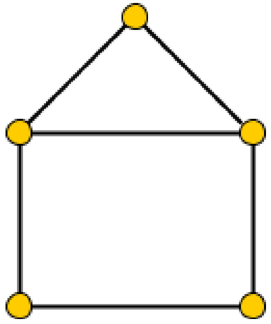
- Flight graph: A vertex represents an airport and an edge represents a flight route between two airports and stores the mileage of the route (edge).
- Social networks (like Facebook): A vertex is user, and two vertices are connected by an edge if and only they are friends.
- Road network: A vertex is a place(point, city), and an edge represent the road between two places.
- Collaboration graphs: Vertices in these graphs correspond to authors of papers, and they are connected by an edge whenever the corresponding authors co-authored an article.
- Web graphs: Vertices correspond to web pages and two vertices are connected by an edge if the web page corresponding to at least one of them has a hyperlink to the other.

Undirected and Directed graphs

Simple Undirected Graph

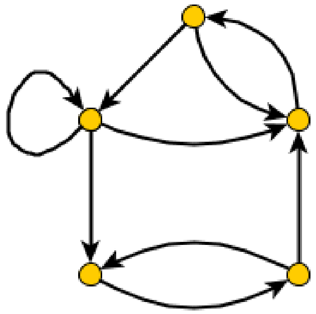
A Simple undirected graph is a set of vertices that are connected by the set of edges, where edges are an unordered pair of distinct vertices.

- In a simple undirected graph both multiple edges and loops are not allowed.



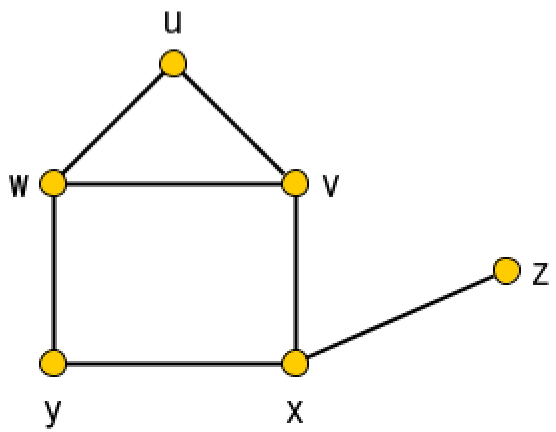
Directed Graph

A *directed graph* (or *digraph*) *(V ,E)* consists of a nonempty set of vertices *V* and a set of *directed edges* (or *arcs*) *E*. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (*u, v*) is said to *start* at *u* and *end* at *v*.



Simple Undirected Graphs

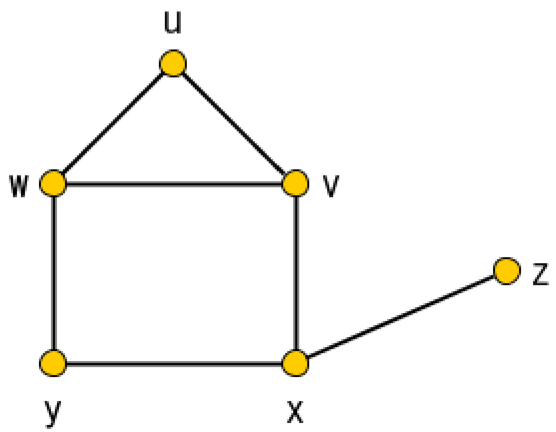
- Two vertices u and v are called *adjacent* (or *neighbors*) in undirected graph G if u and v are endpoints of an edge e of G. Such an edge e is called *incident with* the vertices u and v and e is said to *connect* u and v.



- w is adjacent with u, v and y but not with x and z.
- y is adjacent with x and w but not u, v and z.

Simple Undirected Graphs

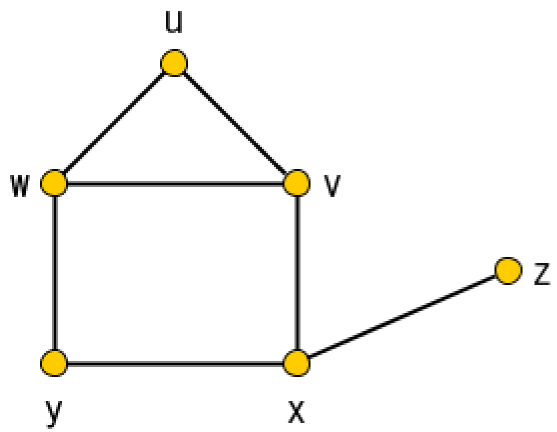
- Given a graph $G = (V, E)$
  - The open neighborhood $N(v) = \{ u \in V \mid u \neq v, uv \in E\}$ of a vertex v is the set of all vertices adjacent to v (not including v).
  - The closed neighborhood $N[v] = N(v) \cup \{v\}$ includes v.



- $N(w) = \{u, v, y\}$
- $N[w] = \{u, v, y, w\}$
- $N(y) = \{w, x\}$

Simple Undirected Graphs - Degree

- The **degree** of a vertex v is the number of incident edges, denoted by $\deg(v)$.



- $\deg(w) = 3$
- $\deg(z) = 1$
- $\deg(y) = 2$

A vertex of degree one is called **pendant**. Consequently, a pendant vertex is adjacent to exactly one vertex.

- Vertex **z** is pendant.

Degree of vertices

- Let $G = (V, E)$ be a simple undirected graph, Then:

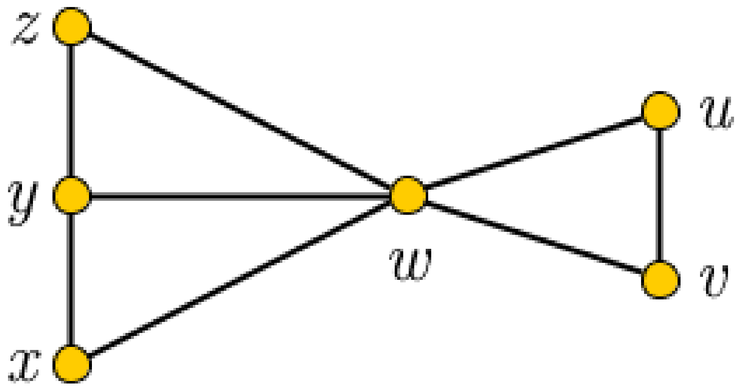
Lemma	$\sum_{v \in V} \deg(v) = 2\|E\|$

- If G is directed graph, Then:

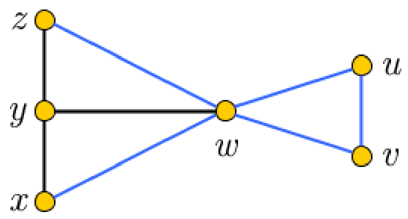
Lemma	$\sum_{v \in V} \text{indeg}(v) = \sum_{v \in V} \text{outdeg}(v) = \|E\|$

Path

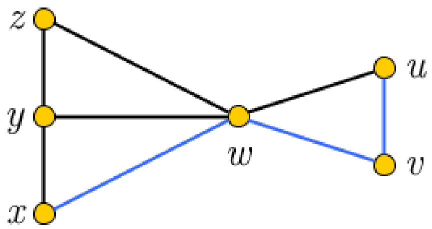
- **Path**: Is a sequence of adjacent vertices.
  - The **length** of a path from a vertex v to a vertex u is the **number of edges** in the path.
  - A path **P** of length l is sequence of $l + 1$ adjacent vertices.
- A path is **simple** if all vertices are different.



Path: (x,w,v,u,w,z)

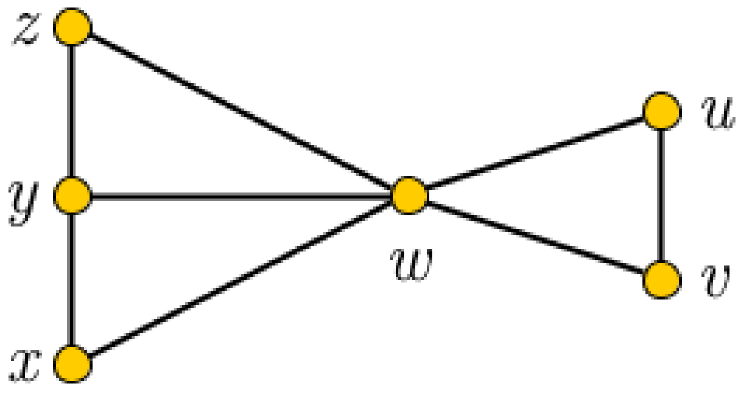


Simple Path: (x,w,v,u)

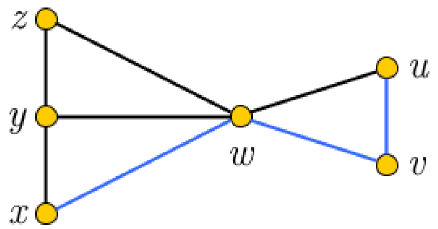


Shortest Paths

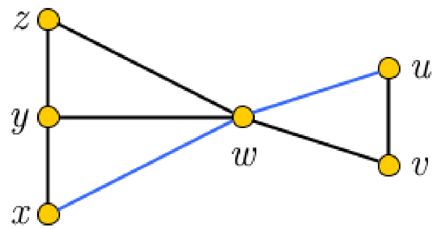
- A **Shortest path** between two vertices $u\ and\ v$ is path with the **minimum** number of edges.



A Path (x, w, v, u) is a simple path between $x\ and\ u$, but not shortest.

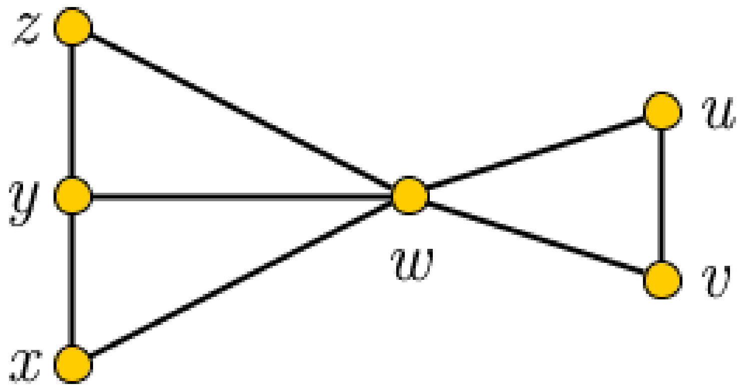


A Path (x, w, u) is a shortest path between $x\ and\ u$.



Distance

- **Distance**: The distance **d(u, v)** from a vertex u to a vertex v in a graph G is the shortest path (minimum number of edges) from u to v. It is a shortest path length from u to v.

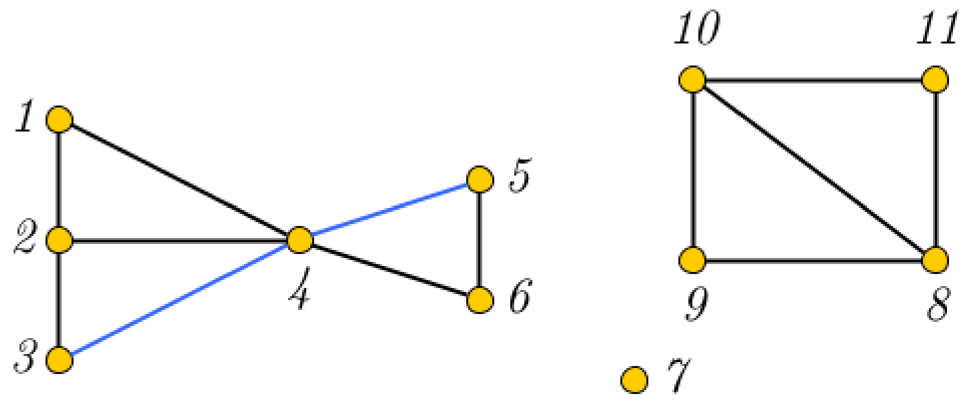


$$d(u,v) = 2$$
$$d(y,w) = 1$$

Connectedness

- Vertices v, w are connected **if and only if** there is a path starting at v and ending at w.
- A graph is **connected** iff every pair of vertices are connected. So a graph is connected if and only if it has only 1 connected component.

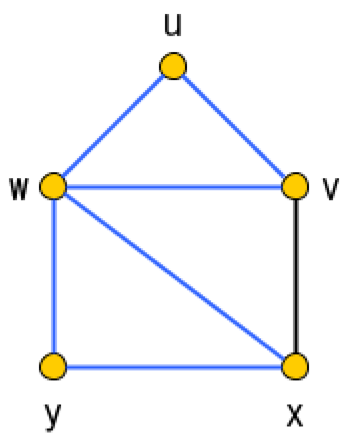
- Every graph consists of separate connected pieces called connected components



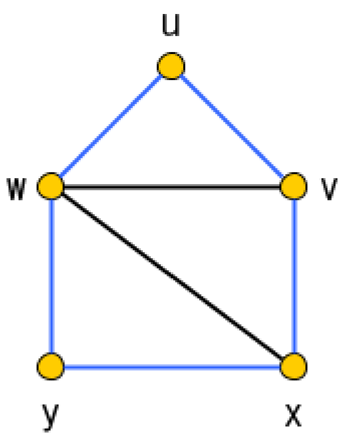
3 connected components

Cycle

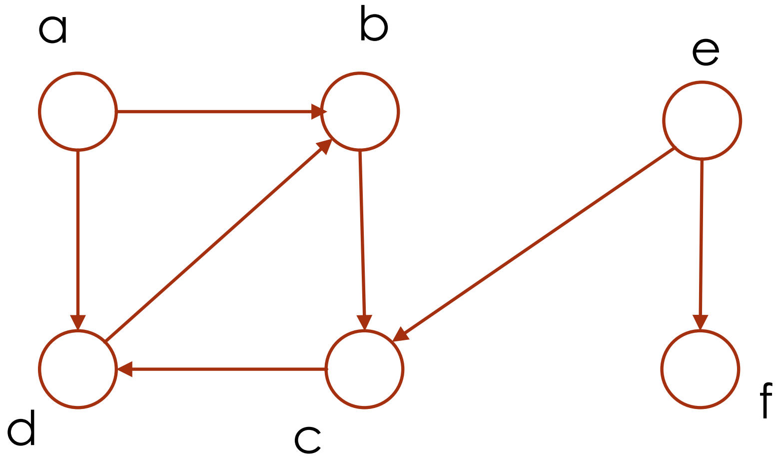
- A **cycle** is a path that begins and ends with the same vertex.
- A cycle is **simple**, if it doesn't cross itself.



Cycle
(u,v,w,x,y,w,u)



Simple Cycle
(u,v,x,y,w,u)



Simple Cycle
(b,c,d)

Properties

Property 1. In an undirected graph with no self-loops and no multiple edges

$$m \leq n\,(n-1)/2$$

Proof: each vertex has degree at most $(n-1)$.

Property 2. A tree with n vertices has $n-1$ edges.

- So, $n \leq m \leq \frac{n(n-1)}{2}$

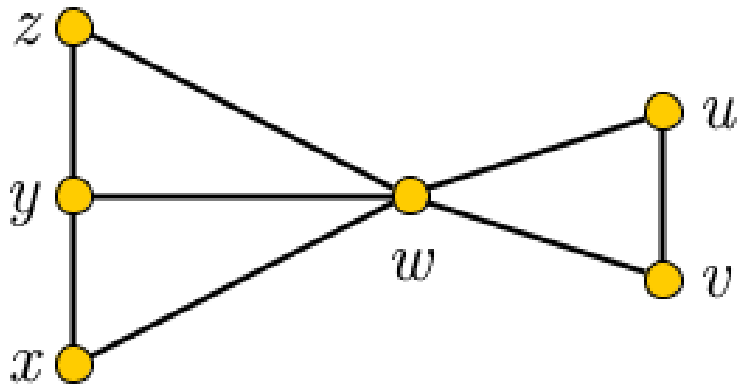
Data Structures for graphs (Graph Representation)

- Structures to represent a graph:

1. **Edge List**
2. **Adjacency List**
3. **Adjacency Matrix**

Edge List

- One simple way to represent a graph G=(V,E) is just a list, or array, of E edges, which we call an **edge list**.



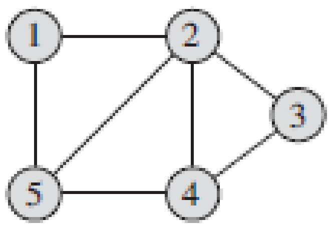
Edge List:

$\{(u,v),(u,w),(v,w),(w,z),(w,y),(w,x),(z,y),(y,x)\}$

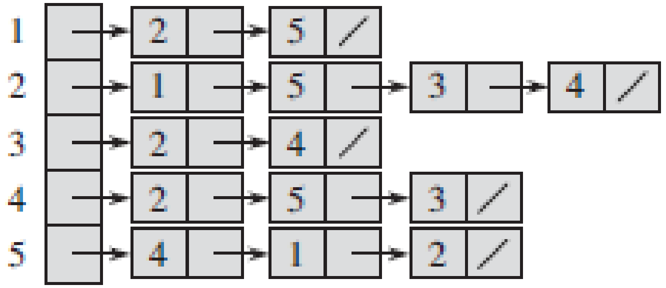
- Space Complexity: O(E)

Adjacency List Example

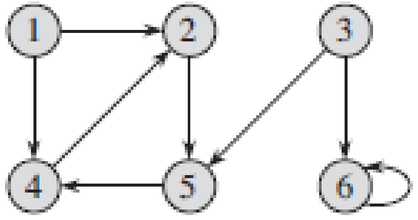
Undirected Graph



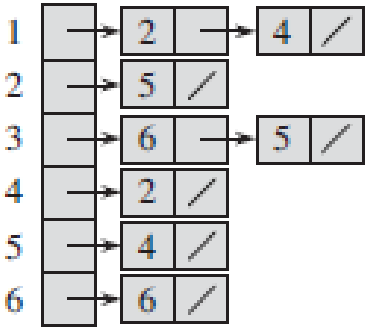
Adjacency list



directed Graph



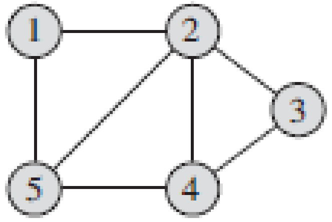
Adjacency list



- The ***adjacency-list representation*** of a graph $G = (V, E)$ consists of an array *Adj* of V lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$.
- **$Adj[u]$ consists of all the vertices adjacent to u in G.**
- **Space Complexity: O(V+E)**

Adjacency Matrix

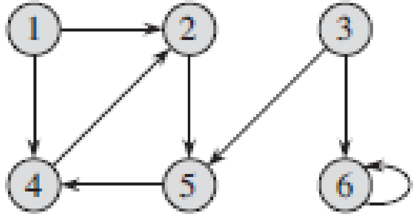
Undirected Graph



Adjacency Matrix

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

directed Graph



Adjacency Matrix

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

- The adjacency-matrix representation of a graph G=(V,E) consists of a VXV matrix (2-Dimensional array **Ar**) such that:
  - **Ar[i,j]=1 if (i,j) is an edge**
  - **Otherwise Ar[i,j]=0**
- **Space Complexity: O(V^2)**

Main Methods of the Graph ADT

Accessor methods

- aVertex()
- incidentEdges(v)
- endVertices(e)
- isDirected(e)
- origin(e)
- destination(e)
- opposite(v, e)
- areAdjacent(v, w)

Update methods

- insertVertex(o)
- insertEdge(v, w, o)
- insertDirectedEdge(v, w, o)
- removeVertex(v)
- removeEdge(e)

Generic methods

- numVertices()
- numEdges()
- vertices()
- edges()

Asymptotic Performance

◈ n vertices, m edges ◈ no parallel edges ◈ no self-loops ◈ Bounds are "big-Oh"	Edge List	Adjacency List	Adjacency Matrix
Space	$n + m$	$n + m$	n^2
incidentEdges(v)	m	deg(v)	n
areAdjacent (v, w)	m	min(deg(v), deg(w))	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	deg(v)	n^2
removeEdge(e)	1	1	1