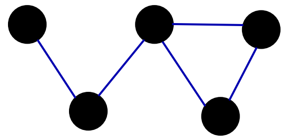
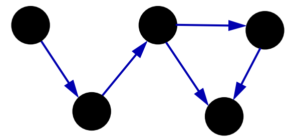
Graph Theory Definitions



Undirected Graph

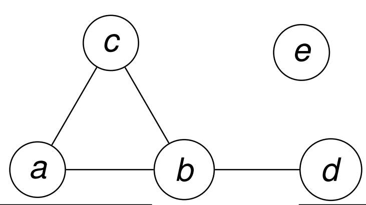


Directed Graph

- A *graph* $G = (V, E)$ consists of a set of *vertices* (or *nodes*) V and
- A set of *edges* E.
- Let $n = |V|$, the number of nodes, and $m = |E|$, the number of edges.
- In an *undirected* graph, each edge is a set of two vertices $\{u, v\}$ (so (u, v) and (v, u) are the same), and self-loops are *not allowed*. When it's clear from the context, we will use (u, v) for $\{u, v\}$.
- In a *directed* graph, each edge is an *ordered pair* of nodes (u, v) (so (u, v) is considered *different* from (v, u)); also, self-loops (edges of the form (u, u)) are allowed.

Terminology: Adjacent

Two vertices are *adjacent* iff there is an edge between them.

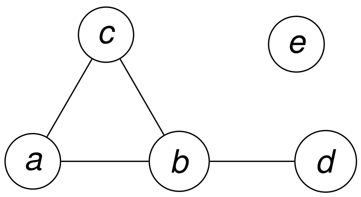


	a	b	c	d	e
a		✓	✓		
b	✓		✓	✓	
c	✓	✓			
d		✓			
e					

	is adjacent to
a	b, c
b	a, c, d
c	a, b
d	b
e	

This brings us to two nice ways to store a graph...

Storing a Graph: Adjacency Matrix

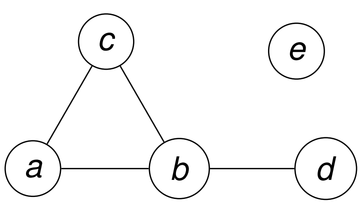


	a	b	c	d	e
a		✓	✓		
b	✓		✓	✓	
c	✓	✓			
d		✓			
e					

Adjacency matrix = store this in a 2D array.

- space: $\Theta(n^2)$
- “who are *adjacent* to v?”: $\Theta(n)$ time
- “are v and w adjacent?”: $\Theta(1)$ time
- convenient for some other operations/queries
- This matrix is for an *undirected graph*. What would change for a *directed graph*?

Storing a Graph: Adjacency Lists



	is adjacent to
a	b, c
b	a, c, d
c	a, b
d	b
e	

Adjacency lists = store *vertices* in a 1D *array* or *dictionary*.
Use a list or a set for each entry on the right.

- At entry $A[i]$, we store the *neighbours* of v_i
- If the graph is *directed*, we store only the *out-neighbours*.
- space: $\Theta(n+m)$
- “who are *adjacent* to v?”: $\Theta(deg(v))$ time, ie, length of adjacency list
- “are v and w adjacent?”: $\Theta(deg(v))$ time if a *list*. Can we make this *faster*?
- *optimal* for graph searches
- How will the adjacency list look different for a *directed graph*?

Operations on Graphs

Some standard operations on graphs are:

- **Add/Remove** a vertex/edge.
- **Edge Query**: given two vertices u, v, find out if the edge (u, v) (if the graph is directed) or the edge $\{u, v\}$ is in E.
- **Neighborhood**: given a vertex u in an *undirected graph*, get the set of vertices $\{v \mid \{u, v\} \in E\}$.
- **In-neighborhood, out-neighborhood**: given a vertex u in a directed graph, get the set of vertices $\{v \mid (v, u) \in E\}$ (in) or $\{v \mid (u, v) \in E\}$ (out).
- **Degree, in-degree, out-degree**: compute the *size* of the *neighborhood*, *in-neighborhood*, or *out-neighborhood*, respectively.