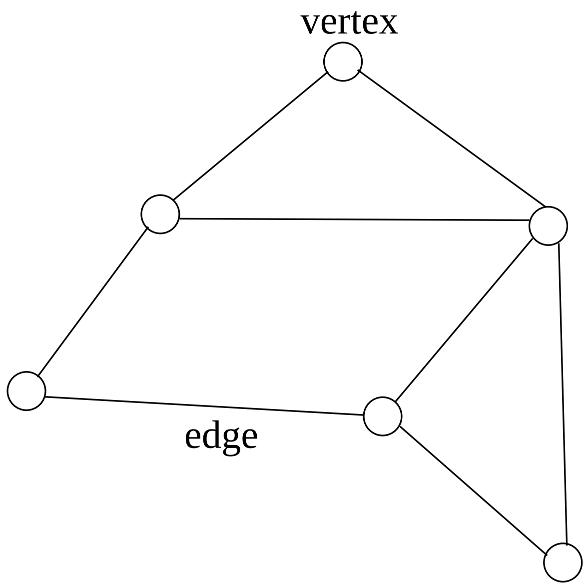
Graphs

- one of the MOST useful tool in modeling problems.

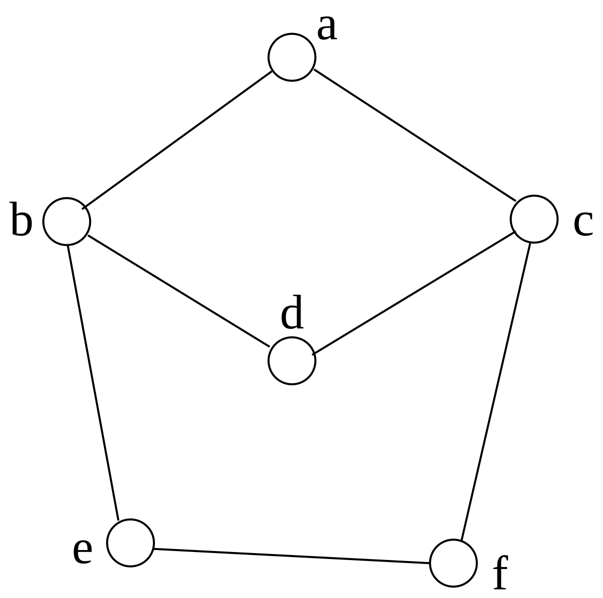


Examples of Applications:

- Computer network
  - edges are physical links
  - vertices are gateways/computers

Formal Definitions

A graph G is specified by an ordered pair (V, E), where V is the set of vertices and E is the set of edges.



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

$$E = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, d\}, \{b, e\}, \{c, f\}, \{e, f\}\}$$

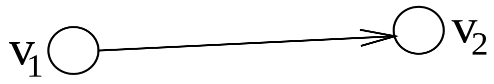
Terminologies

1. If v_1 & v_2 are connected, then they are said to be *adjacent* vertices.
   v_1 & v_2 are *neighbors* of each other.
   v_1 & v_2 are *endpoints* of the *edge* $\{v_1, v_2\}$.

2. If an edge e is connected to v, then v is said to be *incident* to e. The edge e is *incident* to v.

3. $\{v_1, v_2\} = \{v_2, v_1\}$.

4. If edge has direction, the it can be drawn as an arrow (called *arc*).

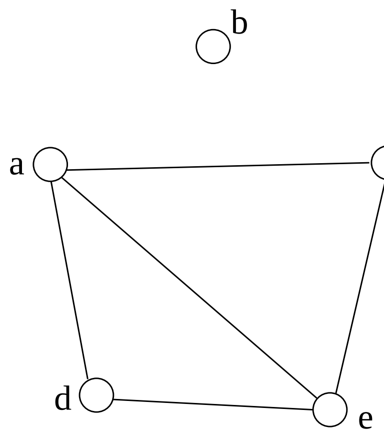


(v_1, v_2)

Graph Representation

1. Adjacency Matrix

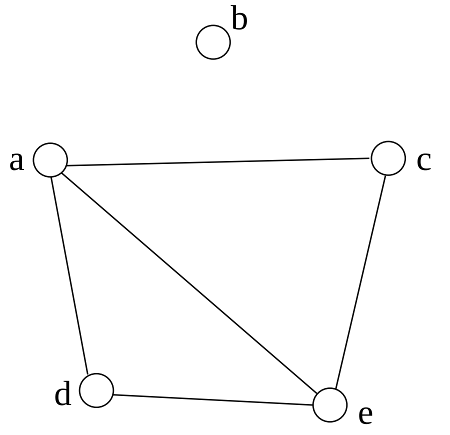
$$
\begin{array}{cc}
 & \text{vertices} \\
\text{vertices} & \begin{bmatrix} & & \\ & a_{ij} & \\ & & \end{bmatrix}
\end{array}
\longrightarrow \text{entry} = \begin{cases} 1 \text{ if connected} \\ 0 \text{ otherwise} \end{cases}
$$

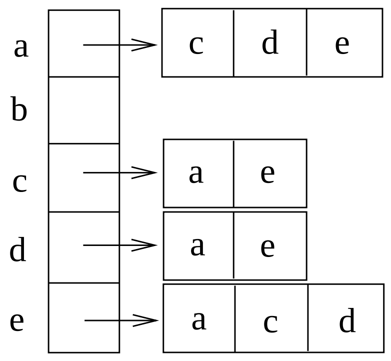


	a	b	c	d	e
a	0	0	1	1	1
b	0	0	0	0	0
c	1	0	0	0	1
d	1	0	0	0	1
e	1	0	1	1	0

$O(n^2)$ storage, where $n = |V|$.

2. *Adjacency List*

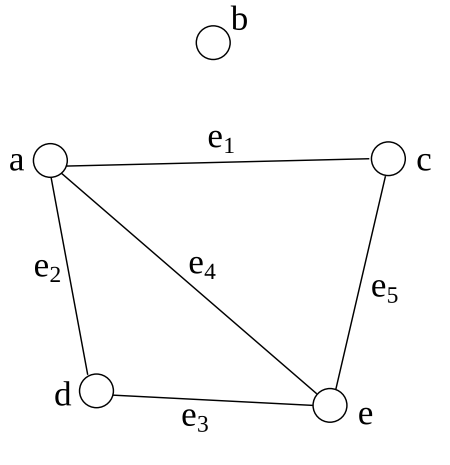




$O(n+m)$ storage, where $n = |V|$ & $m = |E|$.

3. *Incidence Matrix*

- each edge has a name.



	e_1	e_2	e_3	e_4	e_5
a	1	1	0	1	0
b	0	0	0	0	0
c	1	0	0	0	1
d	0	1	1	0	0
e	0	0	1	1	1

$O(nm)$ storage, where $n = |V|$ & $m = |E|$.

Path

- A *path* is a sequence of vertices $(v_0, v_1, v_2, \ldots, v_n)$ such that $\{v_i, v_{i+1}\}, 0 \leq i < n$, is an edge. (Its *length* $= n$.

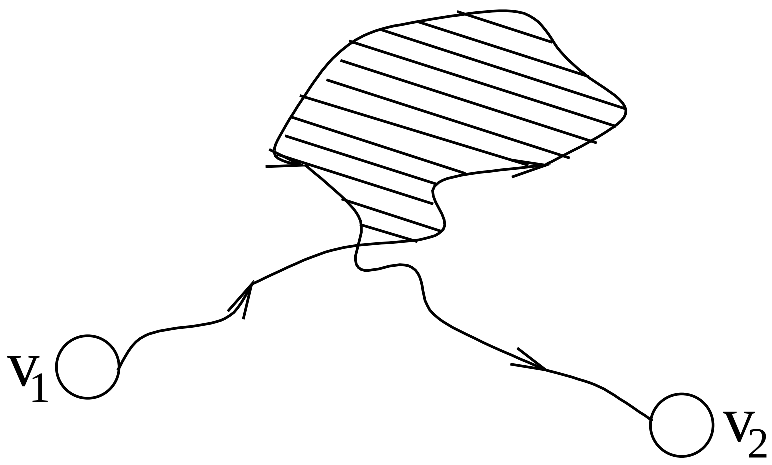
- A path is a *cycle* if and only if $v_0 = v_n$.

- A path is *simple* if and only if it does not contain the same vertex twice.

Connectivity

- A graph is *connected* if and only if there exists a path between every pair of distinct vertices.

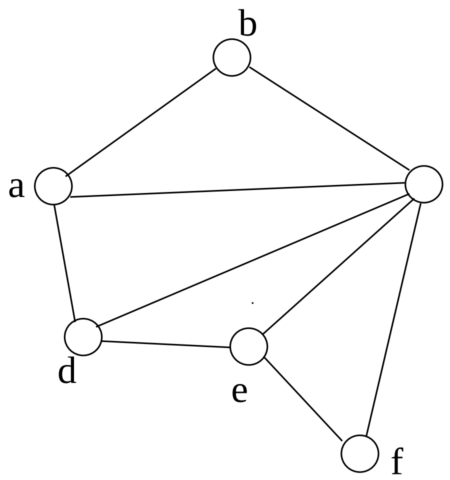
A simple property:
A graph is connected if and only if there exists a simple path between every pair of distinct vertices.



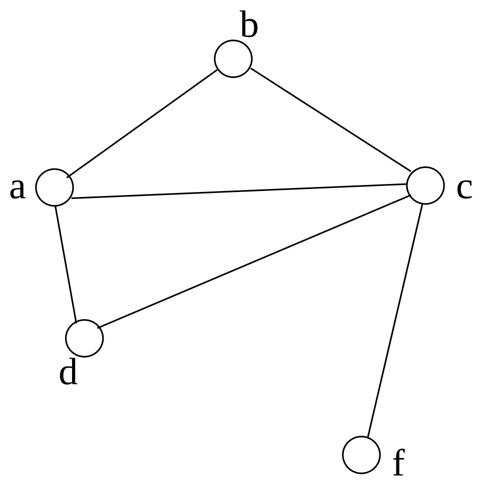
- remove one cycle at a time.

Subgraph

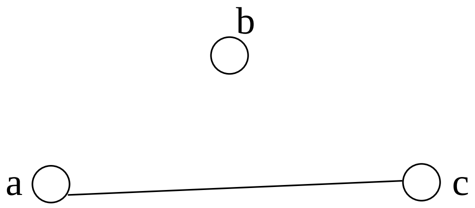
- A graph $H(V_H, E_H)$ is a *subgraph* of $G(V_G, E_G)$ if and only if $V_H \subset V_G$ and $E_H \subset E_G$.



graph G



subgraph H 1

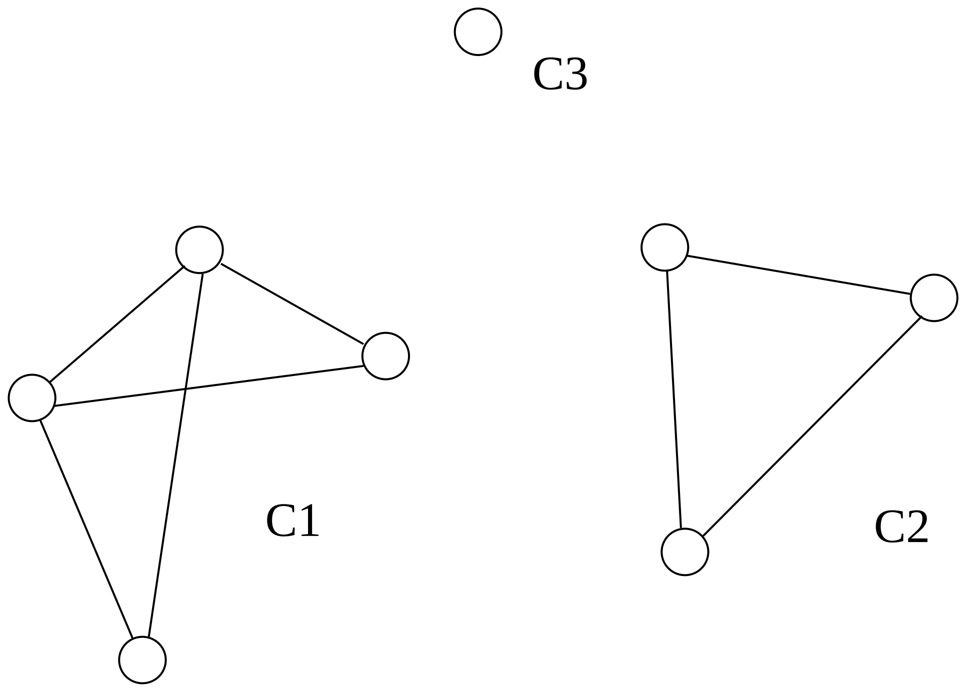


subgraph H 2

Connected Components

- A *connected component* is a maximal connected subgraph of a graph.

- The set of connected components is unique for a given graph.



3 components: C1, C2, and C3

Tree

- A graph G is a *tree* if and only if it is connected and acyclic.
  (i.e., it does not contain any cycle.)

The following statements are equivalent:

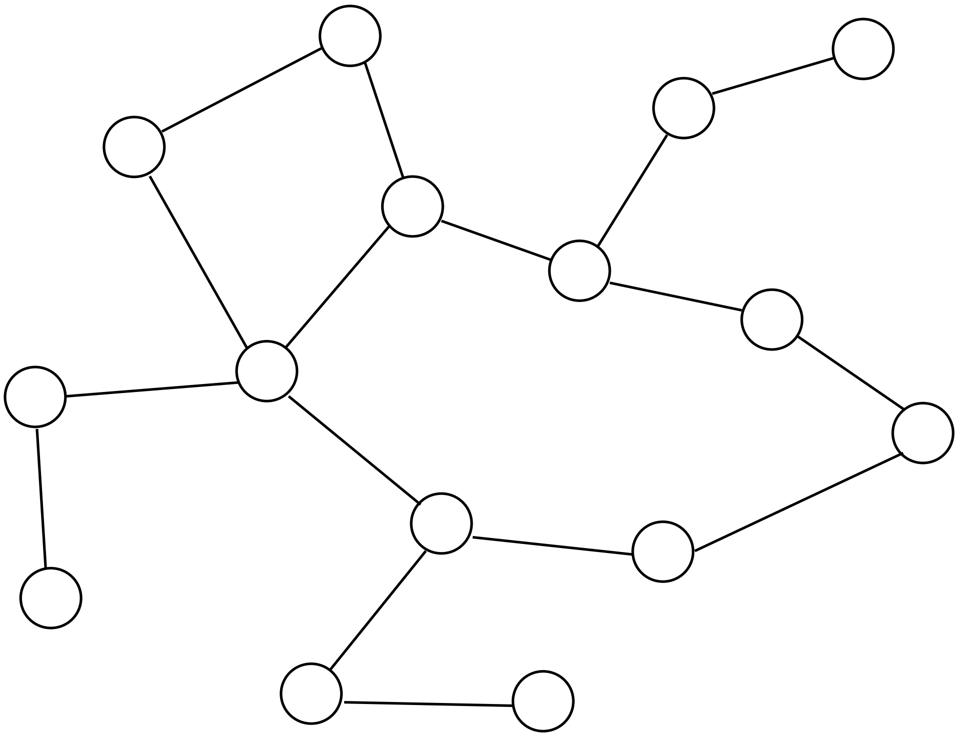
1. G is a tree.
2. G is connected and has exactly $n-1$ edges.
3. G is acyclic and has exactly $n-1$ edges.

Graph Searching

For the following, we assume that the search starts at a particular vertex v.

1. Breadth-First Search(BFS)

- visit the vertices at increasing distances from v.



Encountered(v) will be true after first visit, which is used to avoid repeated visits.

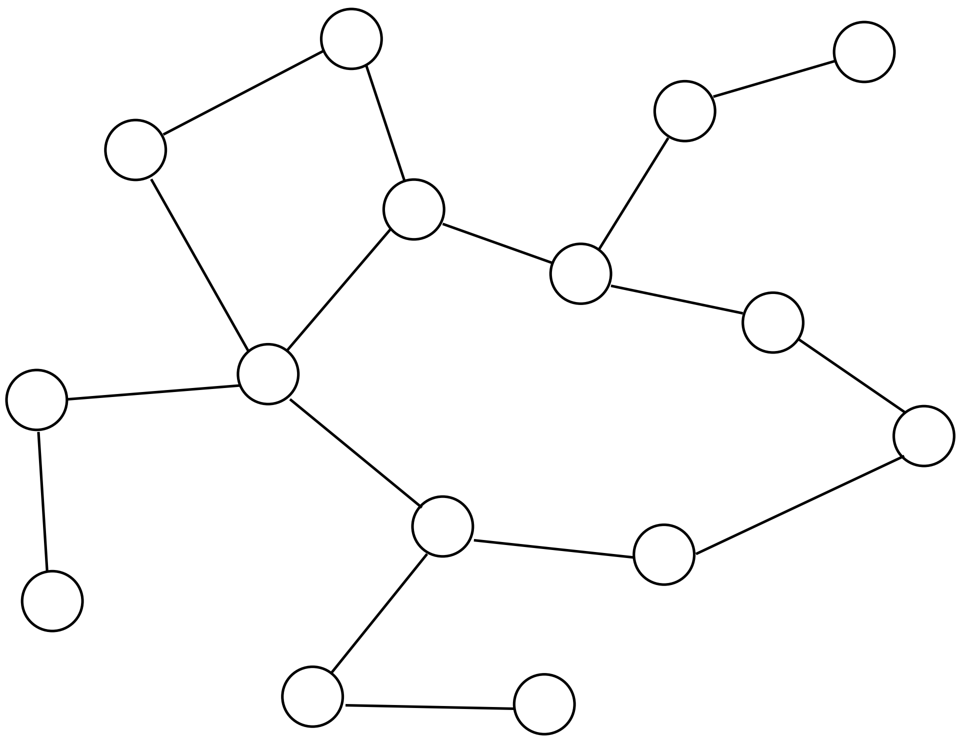
```
procedure BFS(v)
    for each vertex w
        Encountered(w) = false;

    Q = empty queue;

    Encountered(v) = true;
    Enqueue(v, Q);

    while Q is not empty do
        w = Dequeue(Q);
        visit(w);
        for each neighbour w' of w do
            if not Encountered(w') then
                Encountered(w') = true;
                Enqueue(w', Q);
```

1. It also defines a *breadth-first search(BFS) tree*, which is the subgraph consisting of the edges traversed.

2. If each edge has distance 1, then the BFS tree is the union of shortest paths from v to all other vertices.

2. Depth-First Search(DFS)

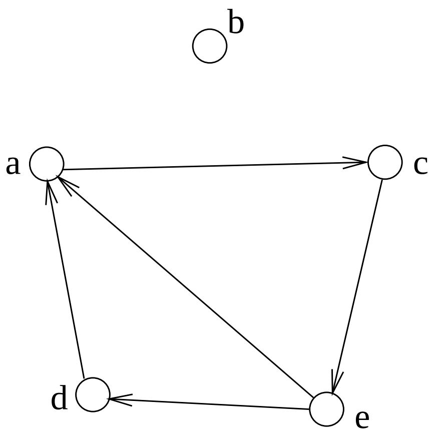
- visit a neighbour of the current node first.



```
procedure DFS(v)
    for each w in G do
        Encountered(w) = false;
    RDFS(v);

procedure RDFS(v)
    Encountered(v) = true;
    visit(v);
    for each neighbour w of v do
        if not Encountered(w) then
            RDFS(w);
```

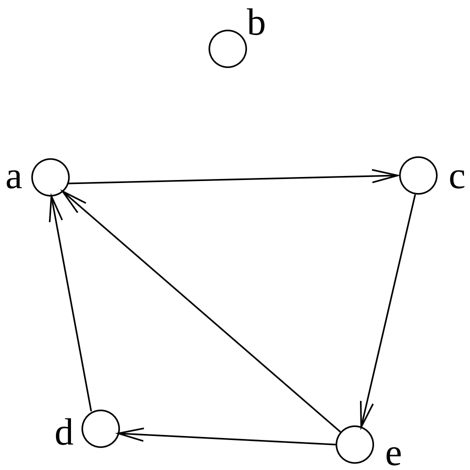
Directed Graph

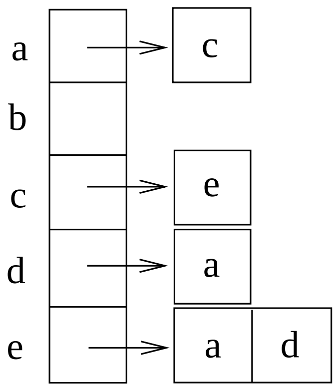
1. *Adjacency Matrix*



	a	b	c	d	e
a	0	0	1	0	0
b	0	0	0	0	0
c	0	0	0	0	1
d	1	0	0	0	0
e	1	0	0	1	0

2. *Adjacency List*





Directed Acyclic Graph

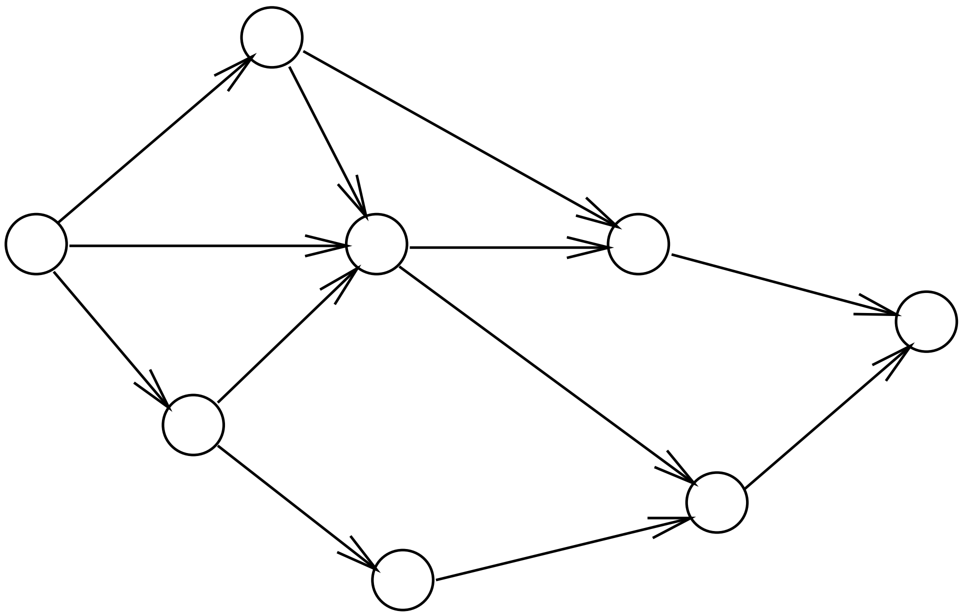
- *"Acyclic"* means that there is no directed cycle in the graph.

- A *directed cycle* is a sequence of vertices such that the interconnecting arcs have consistent directions.

Topological Sort

1. Start at a vertex with no incoming arc that has not been traversed (if there is any).

2. Traverse all the arcs outgoing from this vertex. GOTO Step 1.



```
procedure Topological-Sort(G)
    Initialize Q to be an empty queue;

    for i = 1 to n do
        if indegree(v_i) = 0 then
            Enqueue(v_i);

    while Q != empty do
        v = Dequeue(Q);
        visit(v);
        for all neighbours w of v do
            indegree(w) = indegree(w) - 1;
            if indegree(w) = 0 then
                Enqueue(w);
```