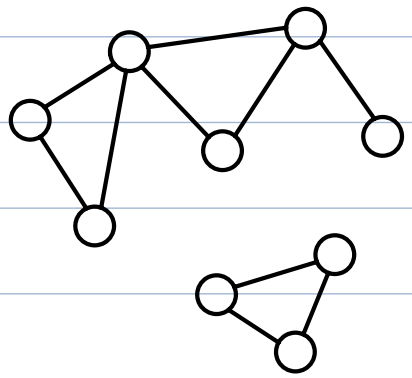
CMSC 451 - Algorithm Design
Lecture 2 - Graph Basics

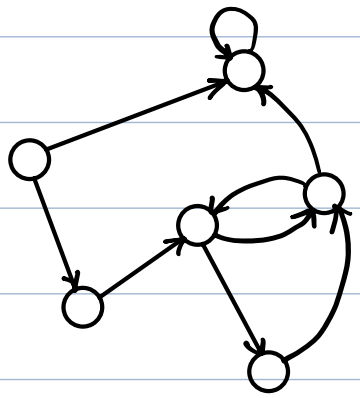
Graph - A discrete structure representing nodes (vertices) joined by links (edges)

Edges may either be:

undirected - or - directed



(Undirected) Graph



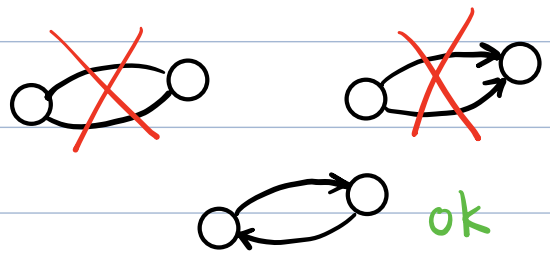
Directed Graph (Digraph)

Definition:

(Undirected) Graph: $G=(V,E)$ is a finite set V of vertices and a set E of unordered pairs of vertices, called edges

Directed Graph (or digraph) $G=(V,E)$ is a finite set V of vertices and a set E of ordered pairs of vertices, called edges

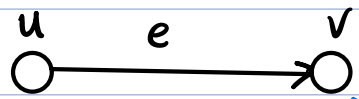
This definition rules out:



- multiple instances of the same edge

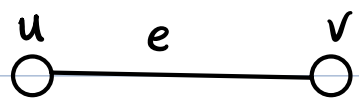
- self loop edges for undirected graphs

Terminology:



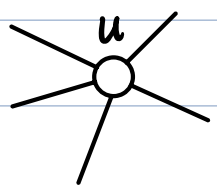
e's origin

e's destination



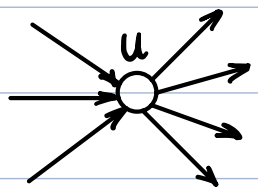
- e is incident on u & v
- u is adjacent to v

Graph degree:



degree = 5

Digraph degree:



out-degree = 4

in-degree = 3

Identities:

$$\sum_{u \in V} \text{in-deg}(u) = \sum_{u \in V} \text{out-deg}(u) = |E|$$

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

How many edges? $n = |V|$ $m = |E|$

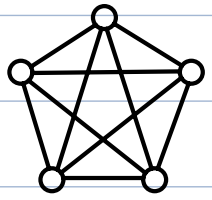
Digraph: $0 \le m \le n^2$

Graph: $0 \le m \le \binom{n}{2} = \frac{n(n-1)}{2}$

A graph/digraph is **sparse** if $m = O(n)$

$\Rightarrow$ **average vertex degree** is $O(1)$

Complete graph $m = \binom{n}{2}$



K_5

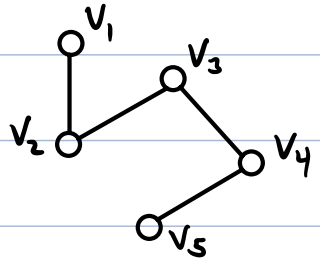
Paths & Connectivity:

Path - Sequence $v_1, v_2, \ldots, v_k$, s.t. $(v_i, v_{i+1}) \in E$

Length = num. of edges = k-1

Cycle - $v_1 = v_k$

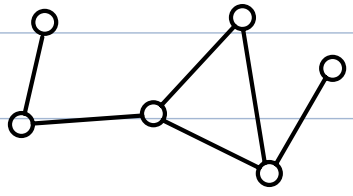
Simple - No vertex is repeated



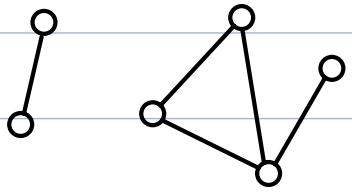
u is **reachable** from v if

there is a path $u \rightsquigarrow v$

Graph is **connected** if all vertex pairs reachable

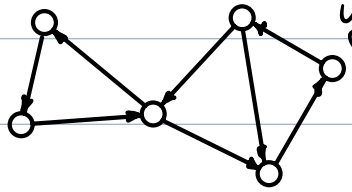


yes

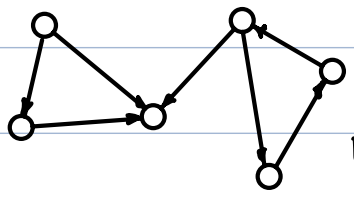


no

Digraph is **strongly connected** if $\forall u,v \in V$ $u \rightsquigarrow v$ + $v \rightsquigarrow u$



yes



no

Q: What is the **min. number of edges** in a **connected graph** on n vertices?

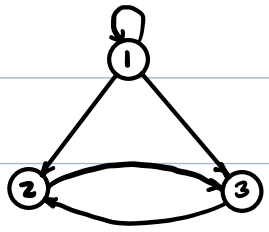
Q: What is the **min. number of edges** in a **strongly-connected digraph** on n vertices?

Representations: Let $G = (V, E)$ be a graph/digraph

$n = |V|$ and $m = |E|$ $\quad V = \{1, 2, \ldots, n\}$

Adjacency Matrix: An **n×n matrix** A

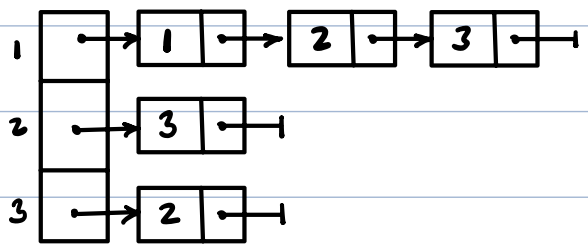
$$A[v,w] = \begin{cases} 1 & \text{if } (v,w) \in E \\ 0 & \text{o.w.} \end{cases}$$



A:

	1	2	3
1	1	1	1
2	0	0	1
3	0	1	0

Adj:



Adjacency List: Array Adj[n] where Adj[v] is **head of a linked list** of v's **neighbors**

Depth-First Search:

- A systematic process for visiting the vertices and edges of a graph.
- Induces a tree structure on the graph
- Intuition - Go vertex to vertex, backtracking only when all neighbors have been visited
- Additional information to guide search:

  for $u \in V$:

  mark[u] - has u been visited?

  d[u] - when was u first discovered?

  f[u] - when was u finished?

  p[u] - the predecessor, the vertex that discovered u

```
DFSVisit(Vertex u)
    mark[u] ← visited      // u discovered
    d[u] ← ++time
    for each (v ∈ Adj[u])  // check neighbors
        if (mark[v] = undiscovered)
            pred[v] ← u    // if unvisited
            DFSVisit(v)    // ... then visit
    mark[u] ← finished     // done with u
    f[u] ← ++time
```

DFS(G): time ← 0

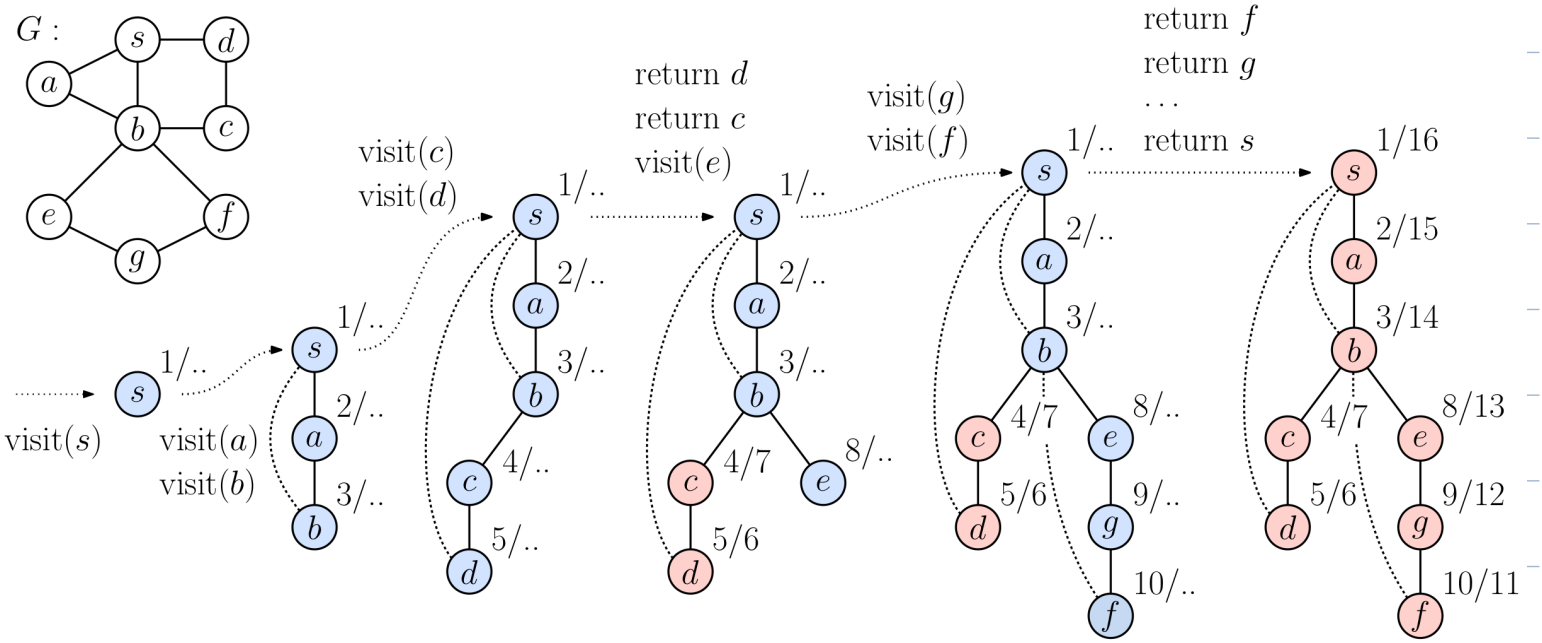
mark all vertices undiscovered

as long as there is an undiscovered vertex v

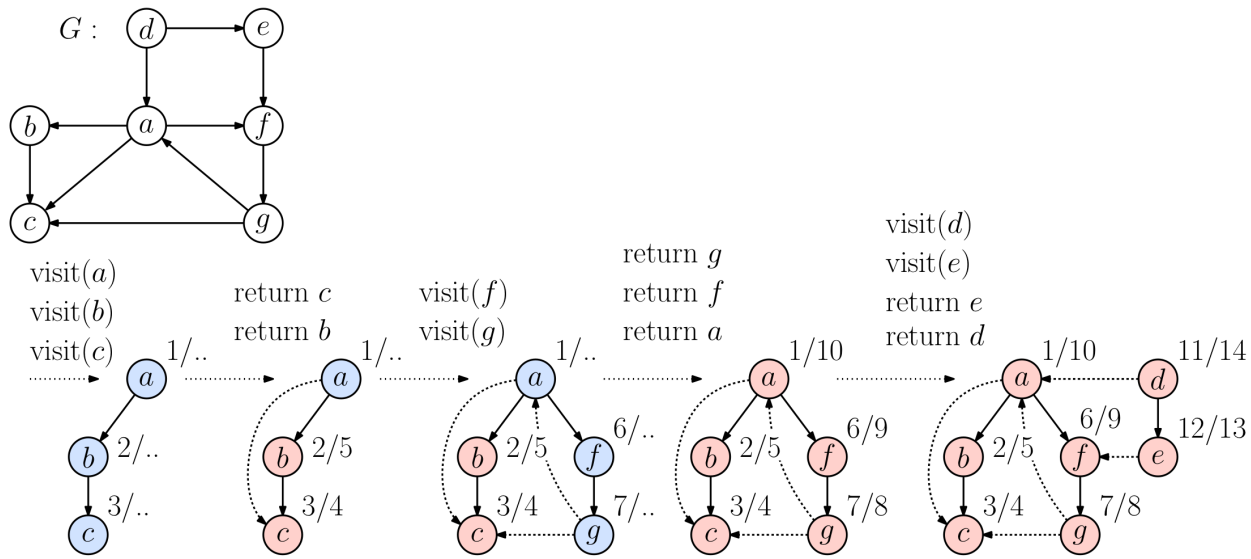
DFSvisit(v)

Example: (Undirected)

○ discov/finish



Example (Directed)



Analysis:

DFS runs in time $O(n+m)$ $[n=|V|, m=|E|]$

- We hit each vertex once: $O(n)$
- We visit each neighboring vertex: $O(\deg(u))$
  - Total:

$$\sum_{u \in V} \deg(u) = 2|E| = O(m)$$

- Total: $O(n+m)$

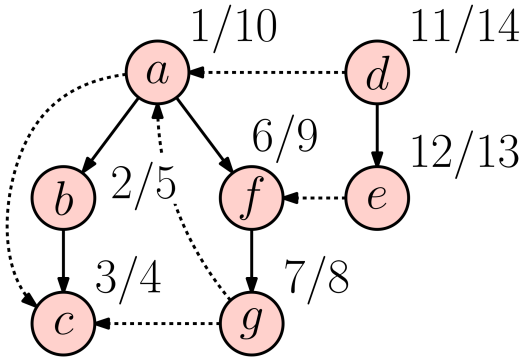
Hierarchical Structure:

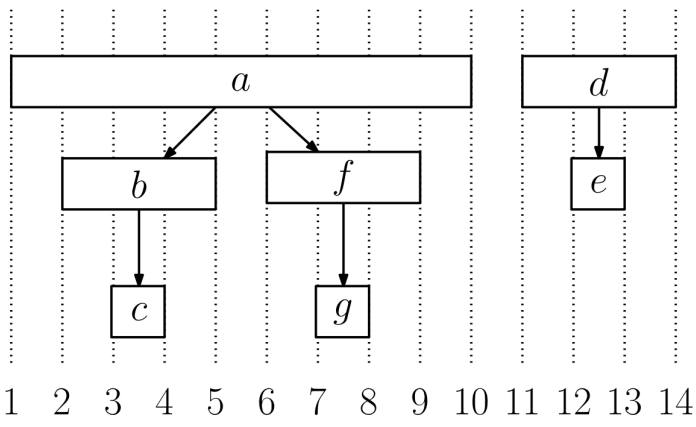
$d[u]$ = discovery time
$f[u]$ = finish time

Parenthesis Lemma:

For all $u, v \in V$ in DFS tree:

- u is descendant of $v \Leftrightarrow [d[u], f[u]] \subseteq [d[v], f[v]]$
- " " ancestor " " " " " $\supseteq$ "
- otherwise $[d[u], f[u]]$ is disjoint from $[d[v], f[v]]$



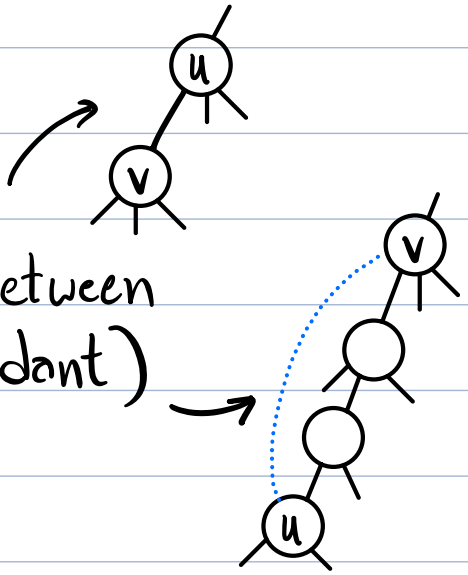


Edge Classification: Tree structure induces distinct edge types. Edge (u, v)

Undirected Graph:

Tree edge: u discovers v

Back edge: Anything else (between ancestor + descendant)



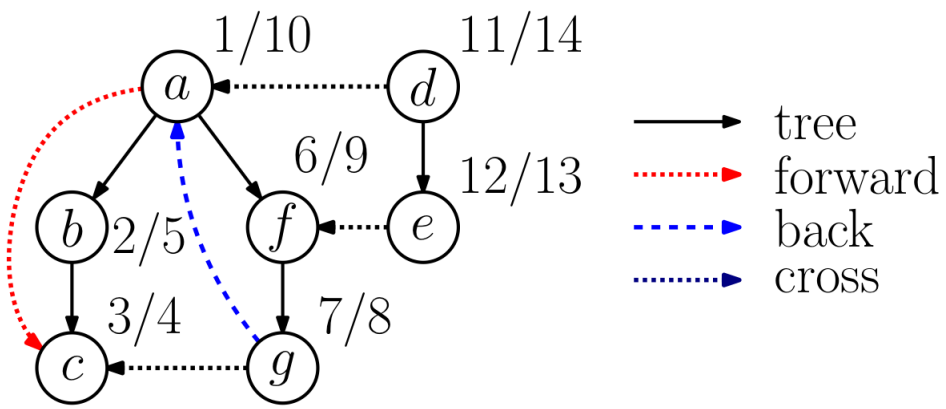
Directed Graph:

Tree edge: u discovers v

Back edge: v is an ancestor of u (includes self loops)

Forward edge: v is a non-child descendant of u

Cross edge: Anything else



Summary:

- Basic graph concepts, definitions, props.
- Depth-First Search
  - Discovery / Finish times
  - Edge classification