A **graph** $G=(V,E)$ is a set of **vertices** V & the set of **edges** E.

any set

If G is **undirected**, $E \subseteq$ pairs of vertices.

unordered

uv

o.w. G is directed & $E \subseteq V \times V$.

$u \to v$ ← successor/head

predecessor/tail of $u \to v$

If $uv \in E$, u & v are **neighbors** or **adjacent**

For $u \in V$,

Degree of u is # neighbors.

If G is directed,

in-degree: # edges $x \to u$

out-degree: # edges $u \to y$

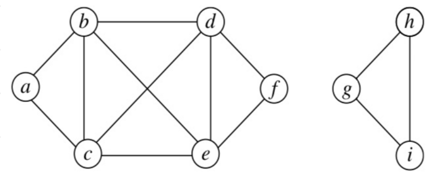
For graph algorithms,

V or E might mean $|V|$ or $|E|$.

i.e. $O(V+E)$

Representations

Adjacency matrix.



$|V| \times |V|$ 2D array/matrix

$A[i,j] = 1$ if edge $i,j \in E$.

$A[i,j] = 0$ o.w.

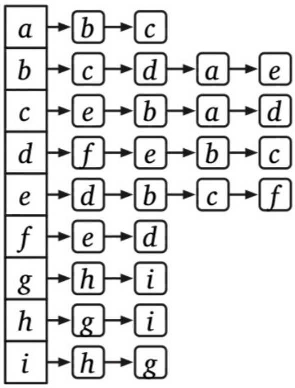
$\Theta(1)$ time to check if an edge in E

	a	b	c	d	e	f	g	h	i
a	0	1	1	0	0	0	0	0	0
b	1	0	1	1	1	0	0	0	0
c	1	1	0	1	1	0	0	0	0
d	0	1	1	0	1	1	0	0	0
e	0	1	1	1	0	1	0	0	0
f	0	0	0	1	1	0	0	0	0
g	0	0	0	0	0	0	0	1	0
h	0	0	0	0	0	0	1	0	1
i	0	0	0	0	0	0	1	1	0

$\Theta(V^2)$ space always

$\Theta(V)$ time to find all neighbors of a vertex

Adjacency list:



An array of length $|V|$.

Each entry points to a list of adjacent vertices of the entry's vertex

If G is undirected, each edge appears twice.

uv is v in u's list & u in v's list

If G is directed $u \to v$ appears as v in u's list only.

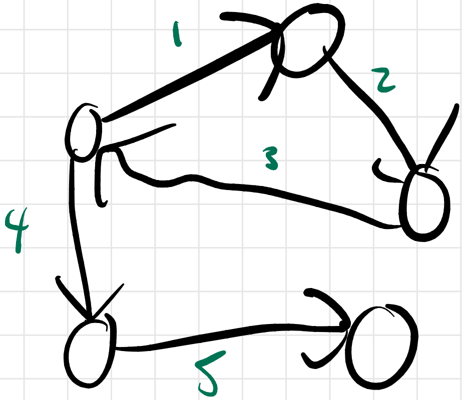
$\Theta(V+E)$ space

$\Theta(\text{degree}(u))$ time to list neighbors of u

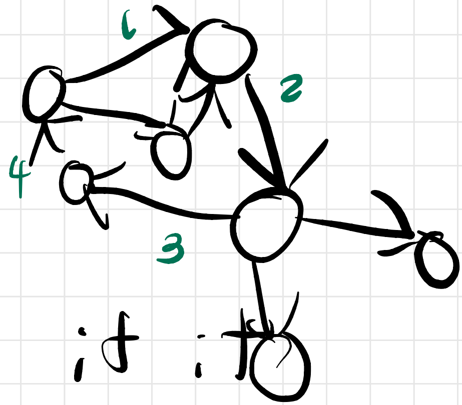
$\Theta(\min\{\text{degree}(u), \text{degree}(v)\})$ to check if uv exists

Assume adjacency list unless told otherwise.

A <u>walk</u> is a sequence of edges s.t. each successive pair share a vertex



← not a path



It is a <u>path</u> if it repeats no vertices

A <u>cycle</u> is a path <u>except</u> we do repeat exactly the first & last vertex

A undirected graph is **connected** if there is a path from every vertex to every other vertex.

Problem: Given graph G + a vertex s. Also given v, is v **reachible** from s. i.e. does there exist a path from s to v?

Breadth-first search (BFS)

```
BFS(s):
  put (∅, s) in a queue
  while queue is not empty
    take (p, v) from queue
    if v is unmarked
      mark v
      parent(v) ← p
      for each edge vw
        put (v, w) in queue
```

Facts: 1) Marks every vertex reachable from s exactly once.

2) Edges of the form parent(v) v form a spanning tree on the <u>component</u> of G containing s.

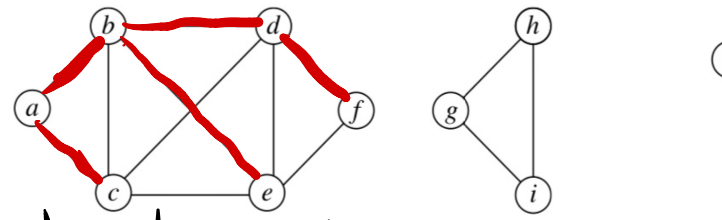
subgraph H of G: Has a subset of G's vertices & edges

A <u>component</u> of G is a maximal connected subgraph

A <u>spanning tree</u> is a connected acyclic subgraph containing every vertex

no cycles

BFS(a)



3) Tree contains shortest path from s to every reachible vertex

min # edges

Running time: $O(V+E)$

(Faster if s's component is small.)

(one case of Erickson's Whatever First Search)

Please use BFS for shortest paths with unit (1) edge weights.

Depth-First Search (DFS):

```
DFS(v):
    mark v
    PREVISIT(v)
    for each edge vw
        if w is unmarked
            parent(w) ← v
            DFS(w)
    POSTVISIT(v)
```

```
DFSALL(G):
    PREPROCESS(G)
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            DFS(v)
```

$O(V+E)$ time

```
DFSAll(G):
    clock ← 0
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            clock ← DFS(v, clock)
```

```
DFS(v, clock):
    mark v
    clock ← clock + 1;  v.pre ← clock
    for each edge v→w
        if w is unmarked
            w.parent ← v
            clock ← DFS(w, clock)
    clock ← clock + 1;  v.post ← clock
    return clock
```

Imagine we pass around a "clock" to time events...

v.pre: starting time of v

v.post: finishing time

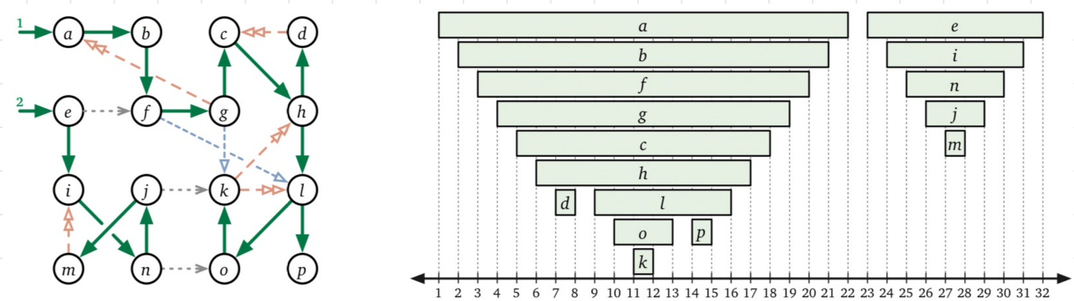
[v.pre, v.post]: active interval of v

Either two active intervals are disjoint or one contains the other.

$[v.pre, v.post] \subset [u.pre, u.post]$

iff DFS(u) (indirectly) calls DFS(v)

implies u can reach v.



Sort by x.pre to get a <u>preorder</u>

Sort by x.post to get a <u>postorder</u>.

Say we run DFSAll...

Fix a vertex v & its (future) $v.pre$ & $v.post$ values & consider any moment in the algorithm.

v is <u>new</u> if $clock < v.pre$

<u>active</u> if $v.pre \le clock < v.post$

<u>finished</u> if $v.post \le clock$

Consider an edge $u \to v$ at the moment DFS(u) begins.

If v is new, a recursive call will mark v.

$u.pre < v.pre < v.post < u.post$

$u \to v$ is a <u>tree edge</u> if DFS(u) calls DFS(v) directly

$u \to v$ is a <u>forward edge</u> o.w.

If v is active,

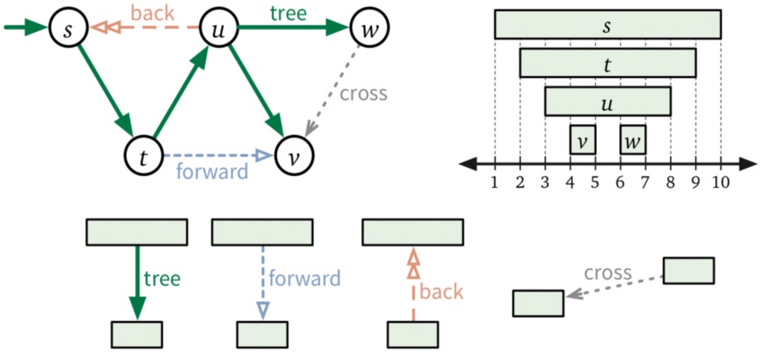
$v.pre < u.pre < u.post < v.post$

$u \to v$ is a <u>back edge</u>

If v is <u>finished</u>,

$v.post < u.pre < u.post$

$u \to v$ a <u>cross edge</u>



Thm (for next time):

Graph G has a directed cycle iff DFSAll(G) yield at least one <u>back edge</u>.