Graph $G = (V, E)$

V: vertices: an <u>arbitrary</u> finite set of anything

E: edges: pairs of elements of V

(i.e. vertices)

$u \to v$ (directed edge)

uv (undirected edge)

If edge e contains vertices u & v:

u & v are <u>adjacent</u>

e is <u>incident</u> to u & v

u is a <u>neighbor</u> of v (& vice versa)

degree of u is
neighbors (assume
no parallel edges)

If $u \to v$ is a directed
edge, u is the tail &
v is the head.

u is a predecessor of v
v is a successor of u.

in-degree: # predecessors
out-degree: # successors

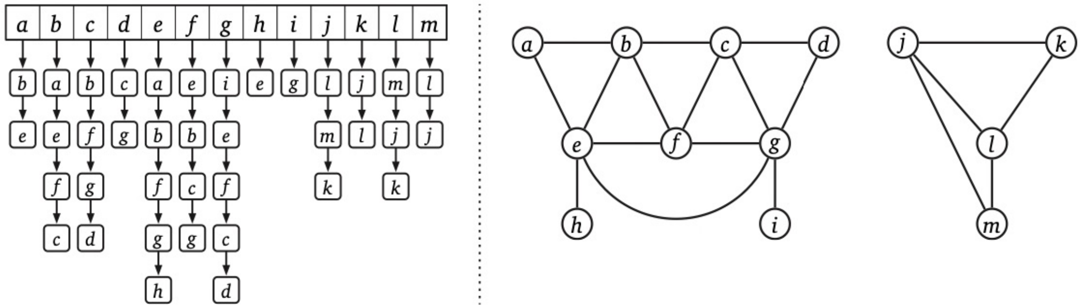
Sometimes V is # vertices
& E is # edges

e.g. "an algo runs in
$O(V+E)$ time"

Data Structures

adjacency list:

an array indexed by vertices or their label
elements are lists of adjacent vertices
(successors only if G is directed)



usually uses singly linked lists for adjacent vertices

each edge uv appears twice if undirected

Space: $\Theta(V+E)$

Learn neighbors of u in optimal $O(\deg(u))$ time.

degree of u (# neighbors)

Have to check u's whole list to know if $u \to v$ exists!

Could use hash table lists...

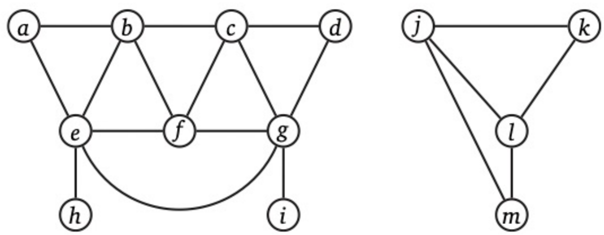
Adjacency matrix:

$|V| \times |V|$ matrix of 0s + 1s stored as 2D array $A[1..|V|, 1..|V|]$

undirected: $A[u,v] = 1$ iff $uv \in E$

directed: $A[u,v] = 1$ iff $u \to v \in E$

	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0	1	0	0	1	0	0	0	0	0	0	0	0
b	1	0	1	0	1	1	0	0	0	0	0	0	0
c	0	1	0	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	1	0	0	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	0	0	0
f	0	1	1	0	1	0	1	0	0	0	0	0	0
g	0	0	1	1	1	1	0	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	1	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	1	1	1
k	0	0	0	0	0	0	0	0	0	1	0	1	0
l	0	0	0	0	0	0	0	0	0	1	1	0	1
m	0	0	0	0	0	0	0	0	0	1	0	1	0



$\Theta(V^2)$ space.

Neighbors of u in $\Theta(V)$ time.

Check if uv exists in $\Theta(1)$ time.

	Standard adjacency list (linked lists)	Fast adjacency list (hash tables)	Adjacency matrix
Space	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(V^2)$
Test if $uv \in E$	$O(1+\min\{\deg(u),\deg(v)\}) = O(V)$	$O(1)$	$O(1)$
Test if $u \to v \in E$	$O(1+\deg(u)) = O(V)$	$O(1)$	$O(1)$
List v's (out-)neighbors	$\Theta(1+\deg(v)) = O(V)$	$\Theta(1+\deg(v)) = O(V)$	$\Theta(V)$
List all edges	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(V^2)$
Insert edge uv	$O(1)$	$O(1)^*$	$O(1)$
Delete edge uv	$O(\deg(u)+\deg(v)) = O(V)$	$O(1)^*$	$O(1)$

in expectation

Assume we're using an adjacency list.

Graph $G' = (V', E')$ is a <u>subgraph</u> of $G = (V, E)$ if $V' \subseteq V$ + $E' \subseteq E$.

walk: a sequence of edges where successive edges share a common vertex

path: a walk with no repeated vertices

G is <u>connected</u> if there is a walk between any pair of vertices

Components: maximal connected subgraphs of G

Given vertex s, a vertex u is reachible from s, if there is an s,u-walk.

Given s, what is reachible?

Whatever-first search:

uses a "bag" data structure

- supports adding objects
- removing added objects

(where we came from)

```
WHATEVERFIRSTSEARCH(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag                (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge vw                    (†)
              put (v, w) into the bag         (⋆⋆)
```

cycle: a walk that repeats only its first/last vertex

tree: a connected graph that has no cycles

spanning tree of G:
a subgraph of G that is a tree & contains every vertex

Lemma: WhateverFirstSearch(s) marks exactly the vertices reachible from s.

The set of pairs $(v, \text{parent}(v))$ where $\text{parent}(v) \neq \emptyset$ form a spanning tree of the component containing s.

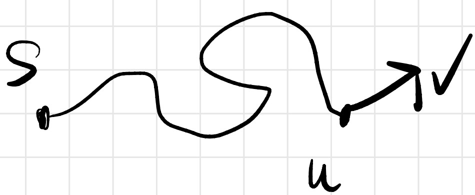
Proof: Each vertex marked at most once.

Show each reachible vertex is marked by induction on shortest path length from s.

s is marked right away.

If $v \neq s$ is reachible.

Let $s \to \cdots \to u \to v$ be shortest path to v.



u is reachible & closer to s than v.

IH implies u is marked.

So we add (u,v) to bag.
Guaranteed v is marked after (u,v) is removed.

We only mark reachible vertices.

s is marked + reachible.

If we mark $v \neq s$.

Pair $(\text{parent}(v), v)$ is an edge.

So we marked parent(v) first.

By induction on order we mark vertices, parent(v) is reachible +

there is a walk

$s \to \dots \to \text{parent}(v) \to v$

Finally the (parent(v), v)

edges spanning the component

of s.

All marked vertices except

s has a parent, so one

fewer edge than # vertices

in component.

$\Rightarrow$ the edges make a

tree

```
WhateverFirstSearch(s):
    put (∅, s) in bag
    while the bag is not empty
        take (p, v) from the bag                 (⋆)
        if v is unmarked
            mark v
            parent(v) ← p
            for each edge vw                     (†)
                put (v, w) into the bag          (⋆⋆)
```

$\le 2|E|+1$ times

$O(|V|)$ times

$\le 2|E|$ time

T: time to add or remove from bag

$O(V + ET)$ time

unmark vertices

put & pull from bag

Which bag?

stack: depth-first spanning tree

- long & skinny

$O(V+E)$

queue: breadth-first spanning tree

(unweighted) shortest paths!

$O(V+E)$

priority queue: depending on priorities:

- Prim's algo for MST
- Dijkstra
- "widest" paths

with min-heap:

$$O(V + E \log E)$$
$$= O(V + E \log V)$$