

Transition: DS+ Algorithms

Graph Data
Structures



Worksheet:



Graphs

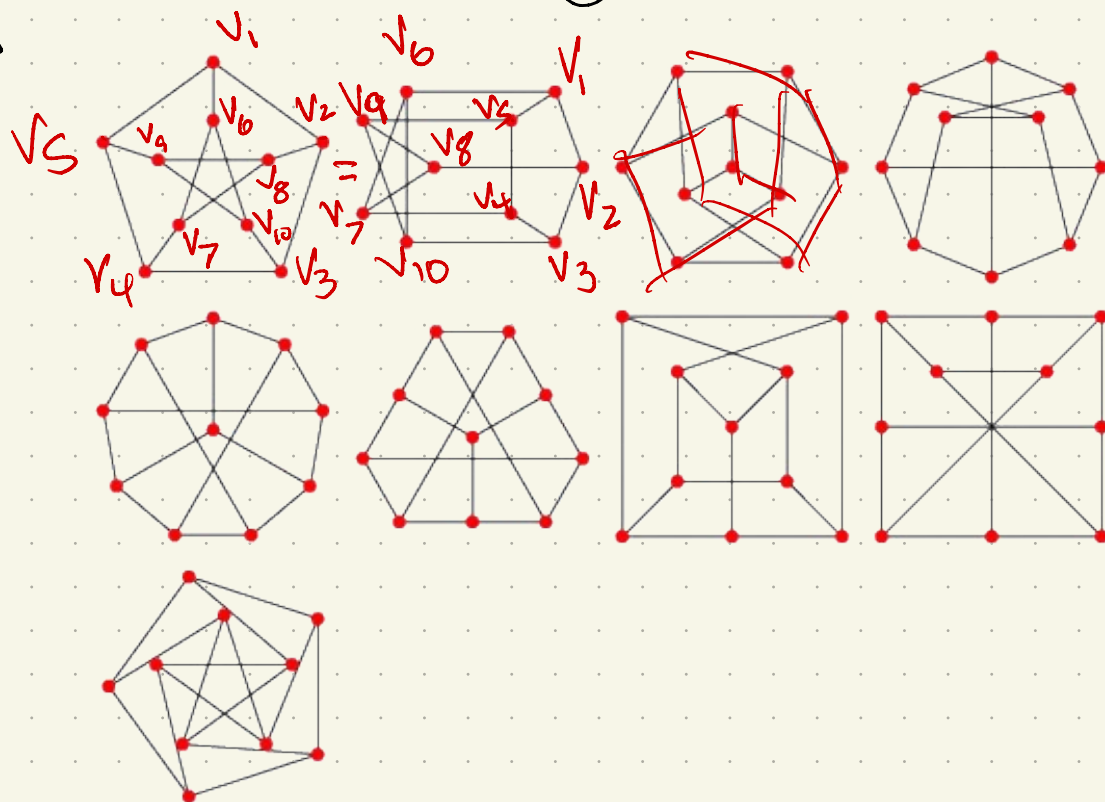
A graph $G = (V, E)$ is an ordered pair of 2 sets.

$V = \text{vertices} = \{v_1, v_2, \dots, v_n\}$

$n = V = |V|$

$E = \text{edges} = \{ \{v_i, v_j\}, \dots \}$

We often draw them, but they do not come with coordinates



Petersen graph

$V = \{v_1, v_2, \dots, v_{10}\}$

$E = \{ \{v_i, v_j\}, \dots \}$

Why?

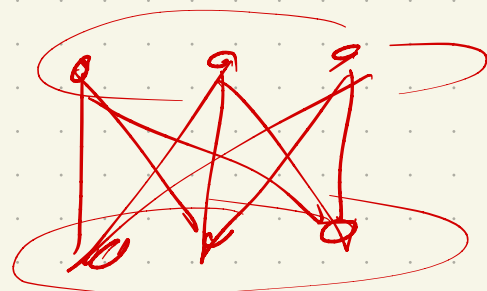
Internet

ML $\rightarrow$ graphs

Road

Biological network

no embedding



relationship

$\rightarrow$

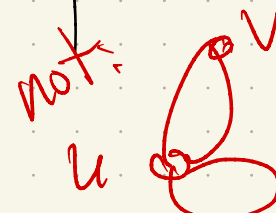
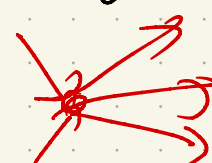
objects

of G :

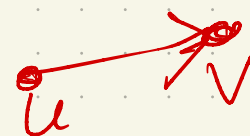
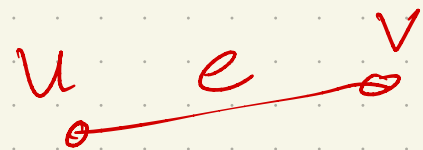
Drawing or embedding

$$m: V \rightarrow \mathbb{R}^2$$

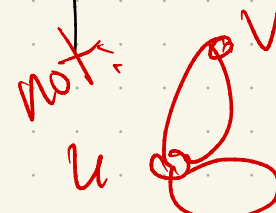
Definitions: See book!

- Vertices (nodes), V
- Edges, $E = \{ \{u, v\} \dots \}$
- endpoints of an edge $e \in E$
- head & tail: $u \rightarrow v$ or $(u, v) = e$
- Simple: no parallel edges or 1-edge loops
not:  $E = \{ \{u, v\}, \{u, v\}, \{u, u\} \}$
parallel loop
- adjacent: u & v are adj' if $uv \in E$
- degree(v): # of edges with v as an endpoint
- predecessor & successor $\rightarrow$ $\text{pred}(e) = u$
 $\text{succ}(e) = v$
- Indegree + outdegree:
of edges into of v 

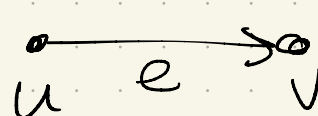
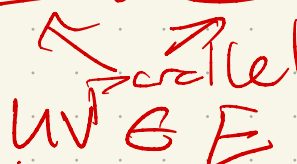
$$e = \{u, v\}$$



every pair uv is in E at most once



$$E = \{ \{u, v\}, \{u, v\}, \{u, u\} \}$$

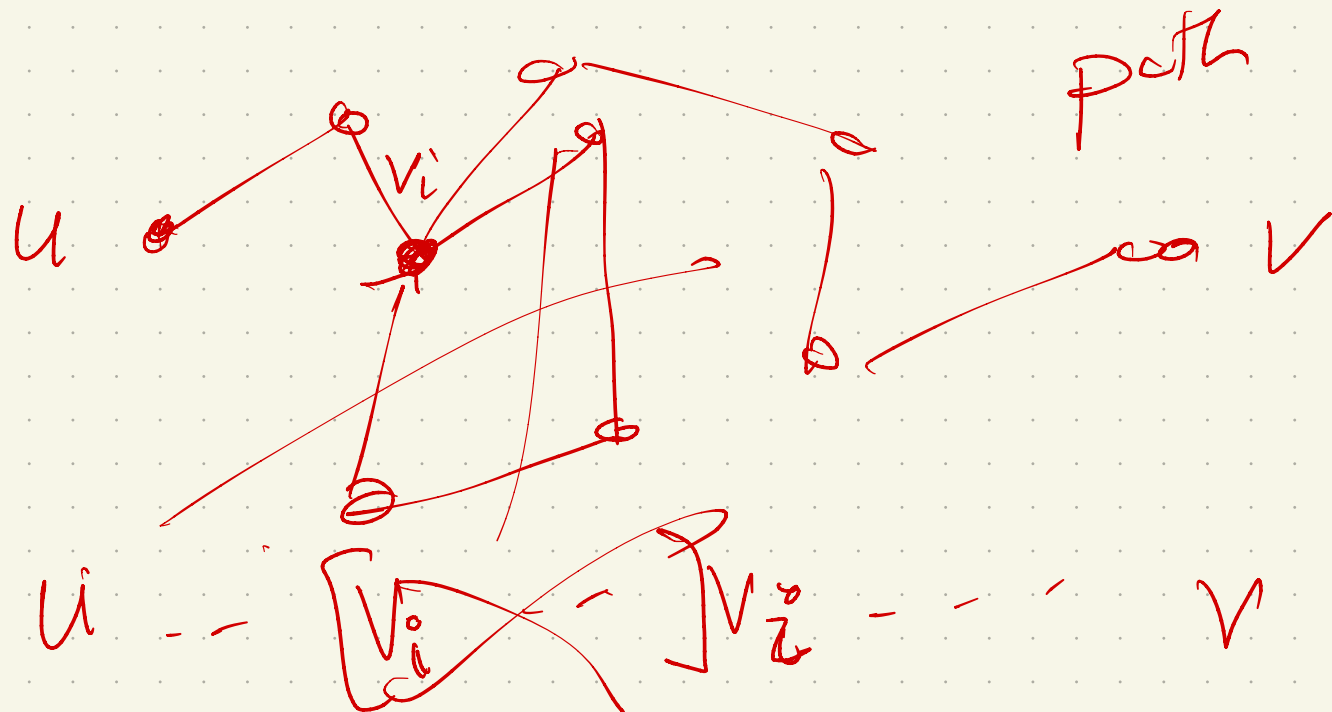


More!

- Sub graph: $G' \subseteq G = (V, E)$, $G' = (V', E')$
 $V' \subseteq V + E' \subseteq E$
- Walk: ordering $v_1, e_1, v_2, e_2, v_3, e_3, \dots$ using verts in V'
 $v_1, v_2 \in E$ $v_2, v_3 \in E$
- Path: no repeated vertices or edges

Note! If you have a walk
 $u \rightsquigarrow v$ can make a path.

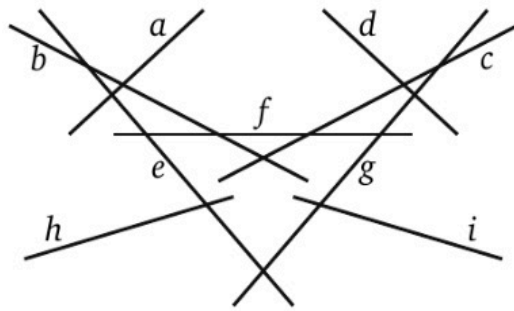
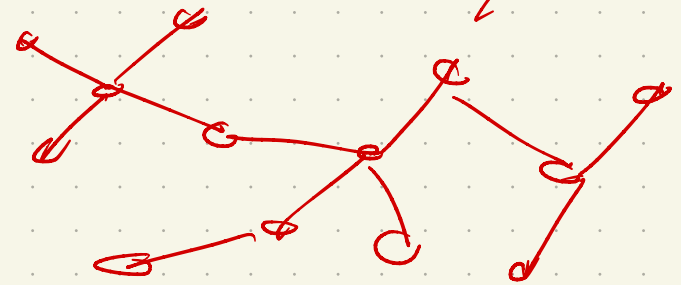
How?



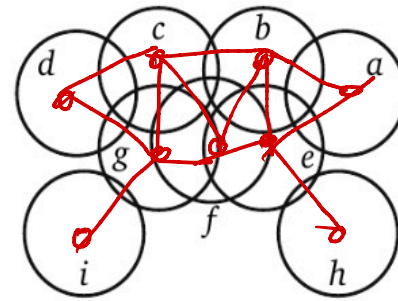
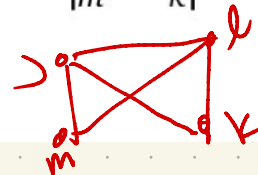
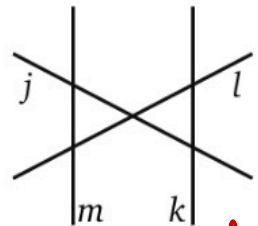
And more:

- Connected: ^{graph} every pair $u, v \in V$ has a path connecting
- Closed (walk): start vertex = end vertex
- cycle: closed path: start vert = end, path otherwise
- tree: connected graph with no cycles

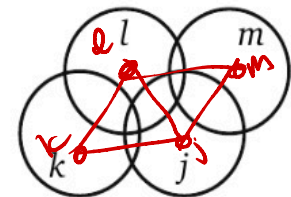
"Contact representations"



(a)



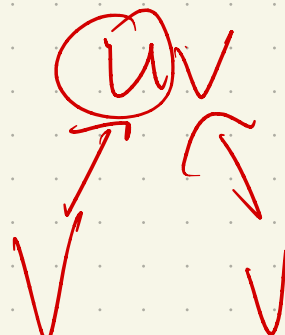
(b)

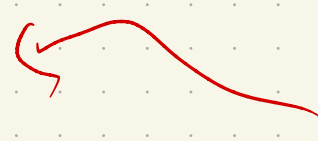


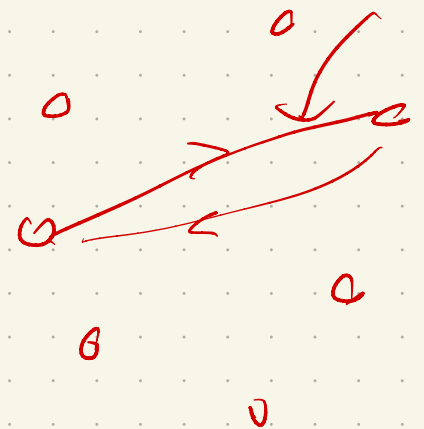
Lemma: $E \leq \frac{V(V-1)}{2} = \binom{V}{2} = \frac{n(n-1)}{2}$ if $n \neq 0$
 $= O(V^2)$

pf: each edge is a pair of vertices $e = \{v_i, v_j\}$

A simple: $v_i \neq v_j$
 $\& v_i \& v_j$ are paired ≤ 1 time

 $= \frac{V(V-1)}{2}$

$uv = vu$  b/c



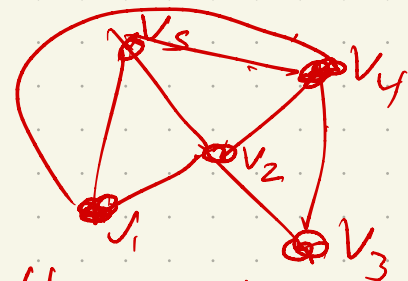
if G is directed: $E \leq V(V-1)$

"Handshaking lemma"

Lemma: $\sum_v d(v) = 2E$

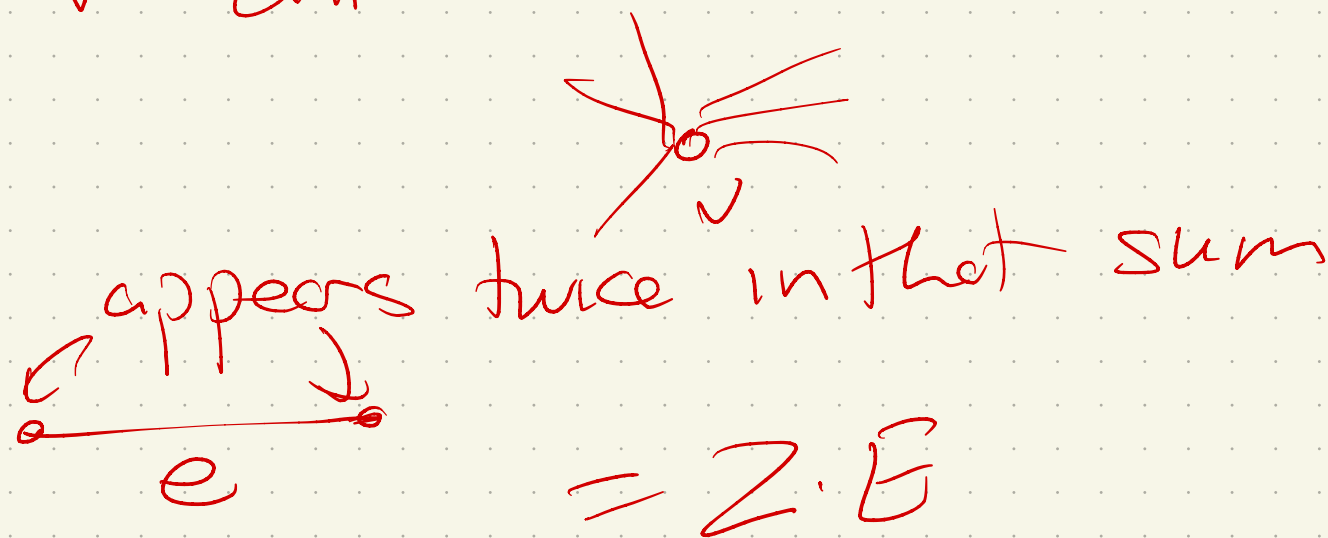
$\uparrow$
degree of v

pf



$$3 + 4 + 2 + 4 + 3 = 16$$

$\sum_{v \in V} d(v) =$ for each vertex, all incident edges



Why care? for each vertex v
for all edges vu

First question

Computers don't do well with images!
So pictures won't help them.

We need to store this info (somehow).

Ideas from data structures:

- lists
- arrays
- trees
- hashing

Adjacency (or vertex) lists:

$\rightarrow V_1$: V_2, V_5

V_2 : V_1, V_3, V_5

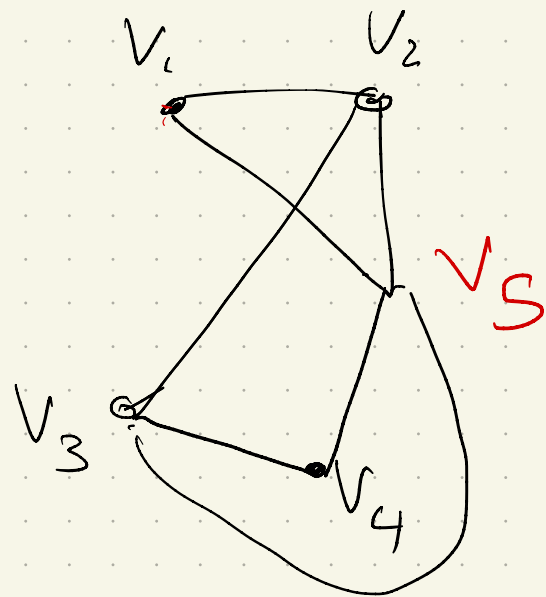
V_3 : V_2, V_4, V_5

V_4 : V_3, V_5

V_5 : V_1, V_2, V_3, V_4

Size: $O(V + \sum_v d(v)) = O(V + E)$

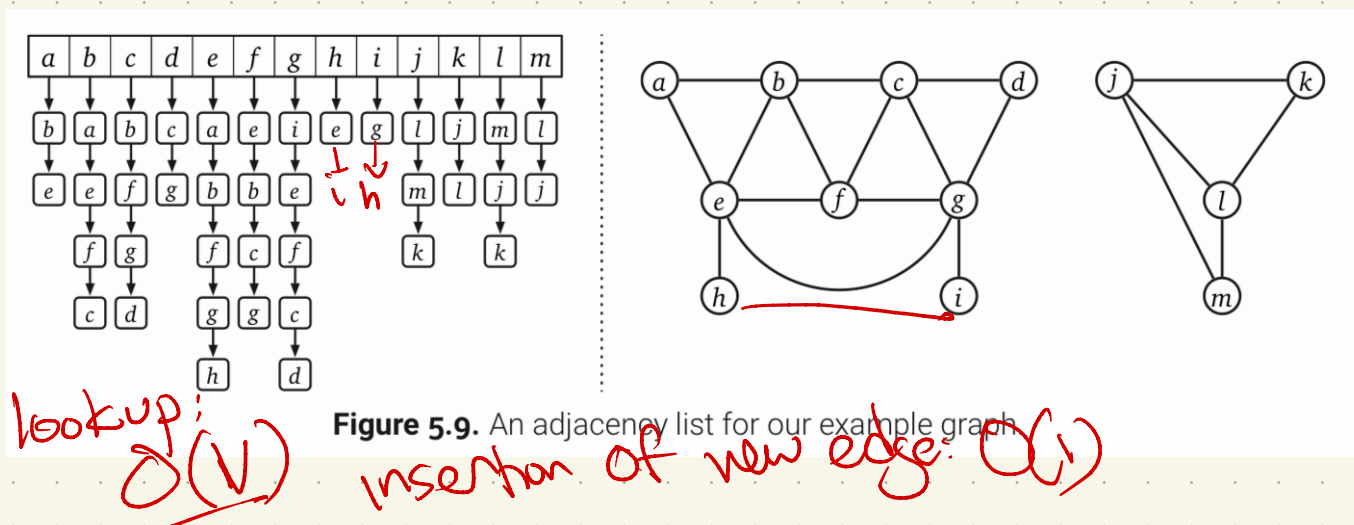
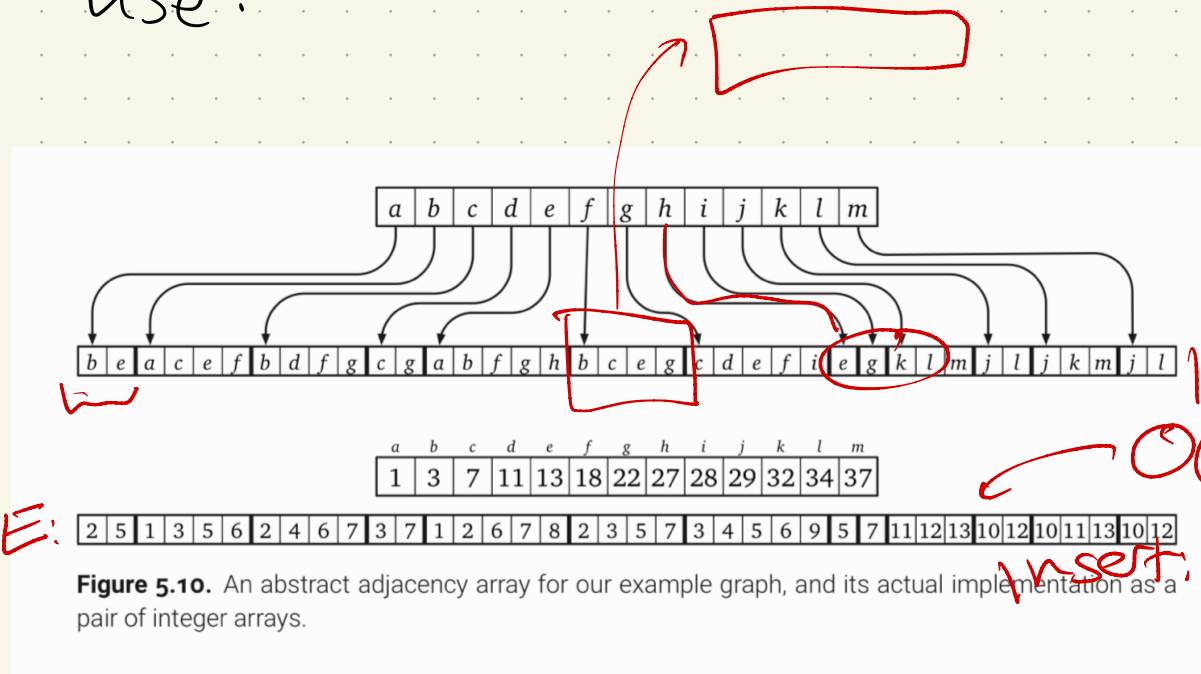
lookup: time to check if $u+v$ are
neighbors: time to lookup in
my list data structure



Implementation:

More buried data structures!

Could use:



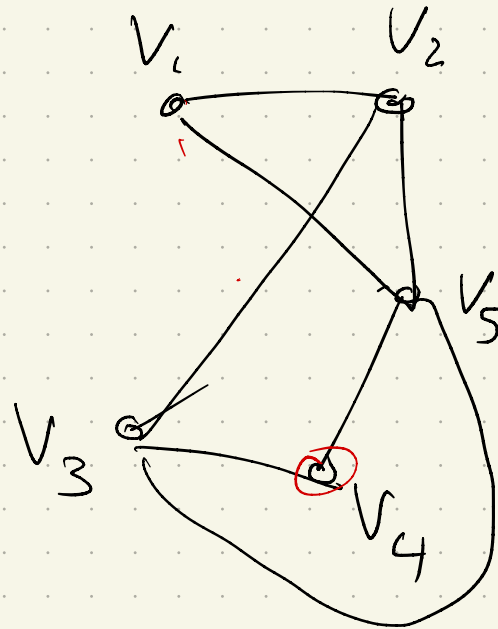
Adjacency Matrix

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	0	0	1
V_2	1	0	1	0	1
V_3	0	1	0	1	0
V_4	0	0	1	0	1
V_5	1	1	0	1	0

$E[V][V]$

Space : $O(V^2)$

check nbr : $O(V)$



$E \leq V^2$



Implementation

	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	1	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	1	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

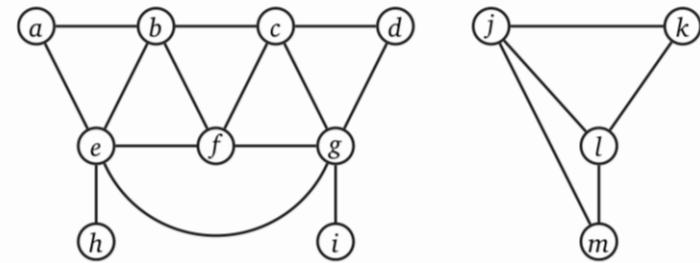


Figure 5.11. An adjacency matrix for our example graph.

Which is better?

Depends!

	Adjacency matrix	Standard adjacency list (linked lists)	Adjacency list (hash tables)
Space	$\Theta(V^2)$	$\Theta(V + E)$	$\Theta(V + E)$
Time to test if $uv \in E$	$O(1)$	$O(1 + \min\{\deg(u), \deg(v)\}) = O(V)$	$O(1)$
Time to test if $u \rightarrow v \in E$	$O(1)$	$O(1 + \deg(u)) = O(V)$	$O(1)$ 
Time to list the neighbors of v	$O(V)$	$O(1 + \deg(v))$	$O(1 + \deg(v))$
Time to list all edges	$\Theta(V^2)$	$\Theta(V + E)$	$\Theta(V + E)$
Time to add edge uv	$O(1)$	$O(1)$	$O(1)^*$
Time to delete edge uv	$O(1)$	$O(\deg(u) + \deg(v)) = O(V)$	$O(1)^*$

In the rest of this book, unless explicitly stated otherwise, all time bounds for graph algorithms assume that the input graph is represented by a standard adjacency list. Similarly, unless explicitly stated otherwise, when an exercise asks you to design and analyze a graph algorithm, you should assume that the input graph is represented in a standard adjacency list.

Really - might depend on input!

- size of graph

- freq. of changes

- representation: usually, some "word problem" is handed to you! You'll have to build the graph.

Ex: Given a set of overlapping circles, find the largest set where no 2 intersect.

Build G:



ind set ✓

Even more:

- Space available
- language used
- previous "legacy" code
- other developers.

⋮

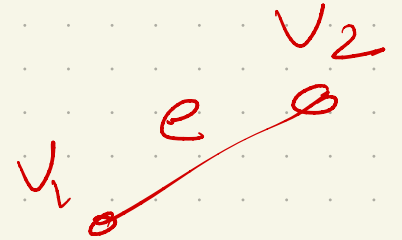
To repeat — too keep it simple here:

In the rest of this book, unless explicitly stated otherwise, all time bounds for graph algorithms assume that the input graph is represented by a standard adjacency list. Similarly, unless explicitly stated otherwise, when an exercise asks you to design and analyze a graph algorithm, you should assume that the input graph is represented in a standard adjacency list.

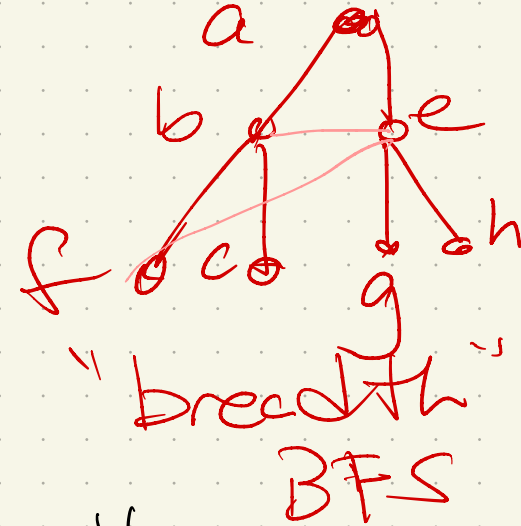
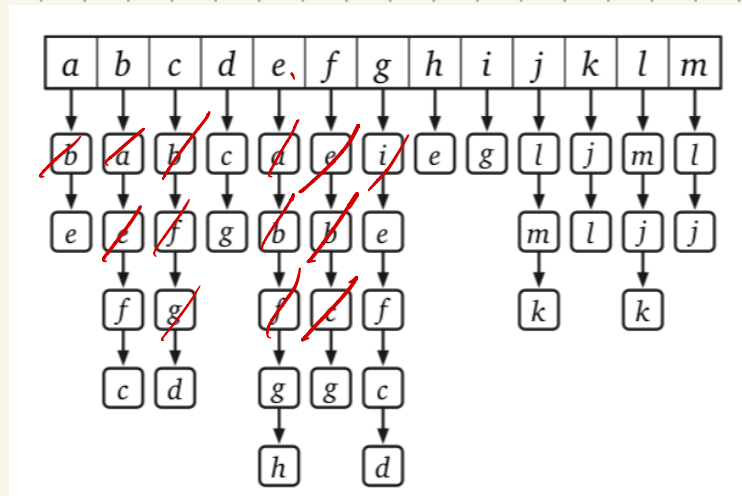
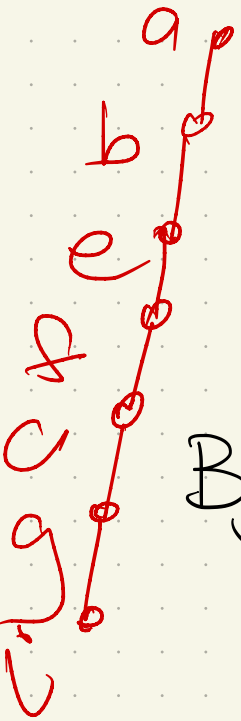
Graph Searching

How can we tell if 2 vertices are connected?

Remember the computer only has:



Depth!



Bigger question: Can we tell if all the vertices are in a single connected component?

Breadth First Search
+ Depth First Search