CS2100

Graphs- pt1

Recap

- Lab 10 - extended until tonight
- HW 9 - next Wed.
- no class Thurs, Fri, & Mon.
- HW10 over graphs
  ↳ up next week, due last Fri. of classes
- Review session on final day
- Final exam: Wed. of finals at 2pm
  - Conflicts or accomodations, let me know ASAP!

Graphs

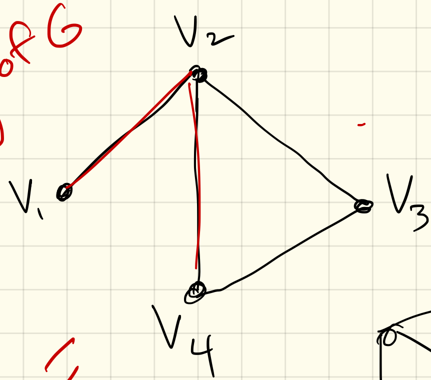
A graph $G = (V, E)$ is an ordered pair of 2 sets:

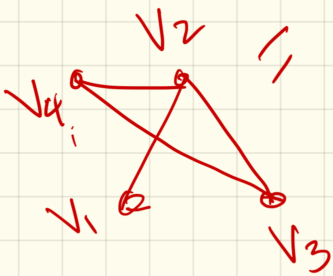
V = vertices = $\{v_1, v_2, v_3, v_4\}$

E = edges = $\{\{v_1, v_2\}, \{v_2, v_4\}, \dots\}$

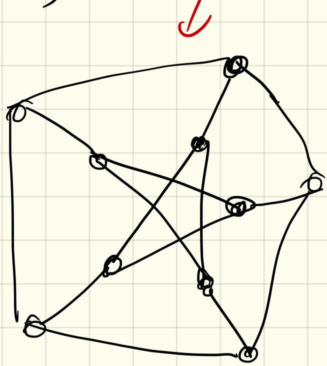
View:

drawing of G





=



Why?

They model everything!

non-heirachical,
non-linear

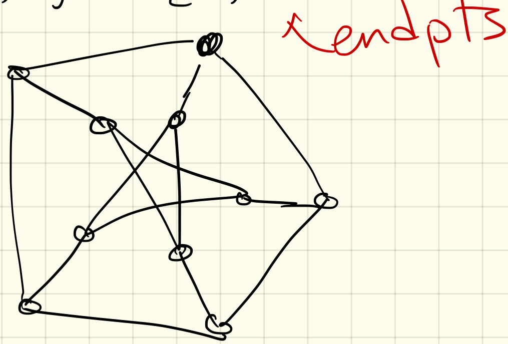
Examples

road networks
social connection
internet
⋮

More dfns:

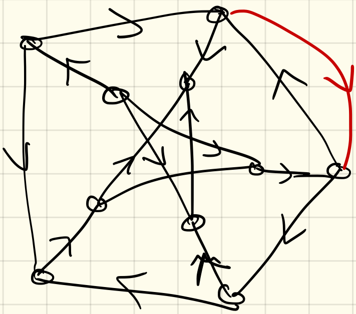
G is <u>undirected</u> if edges are unordered pairs

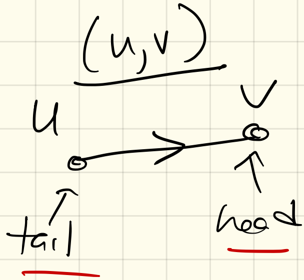
so $\{u,v\} = \{v,u\}$



G is <u>directed</u> if edges are ordered pairs

so $(u,v) \neq (v,u)$





Dfns cont:

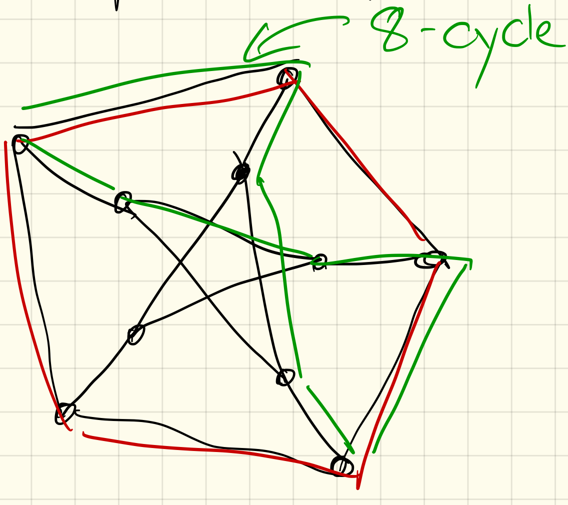
The **degree** of a vertex, $d(v)$, is the number of adjacent edges.

A **path** $P = v_1, \ldots, v_k$ is a set of vertices with $\{v_i, v_{i+1}\} \in E$

(or $(v_i, v_{i+1}) \in E$ if directed)

A path is **simple** if all vertices are distinct

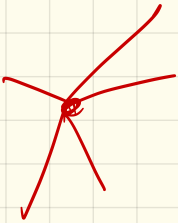
A cycle is a path which is simple except $v_1 = v_k$.

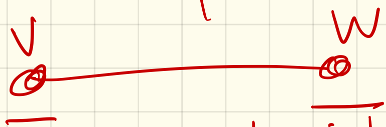


<u>Lemma:</u> (degree-sum formula)

$$\sum_{v\in V} d(v) = 2|E|$$

<u>pf:</u>





2 endpoints

every edge adds +1
so sum on left
exactly twice.

Size of G:

2 parameters:

$|V| = \underline{n}$

$|E| = \underline{m}$

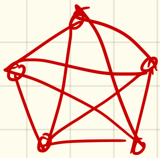
How big can m be in terms of n?

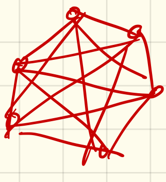
For every 2 vertices could have 1 edge.

$$\binom{n}{2} = \frac{n(n+1)}{2} = O(n^2)$$

Worst case: $\underline{K_n}$

K_5

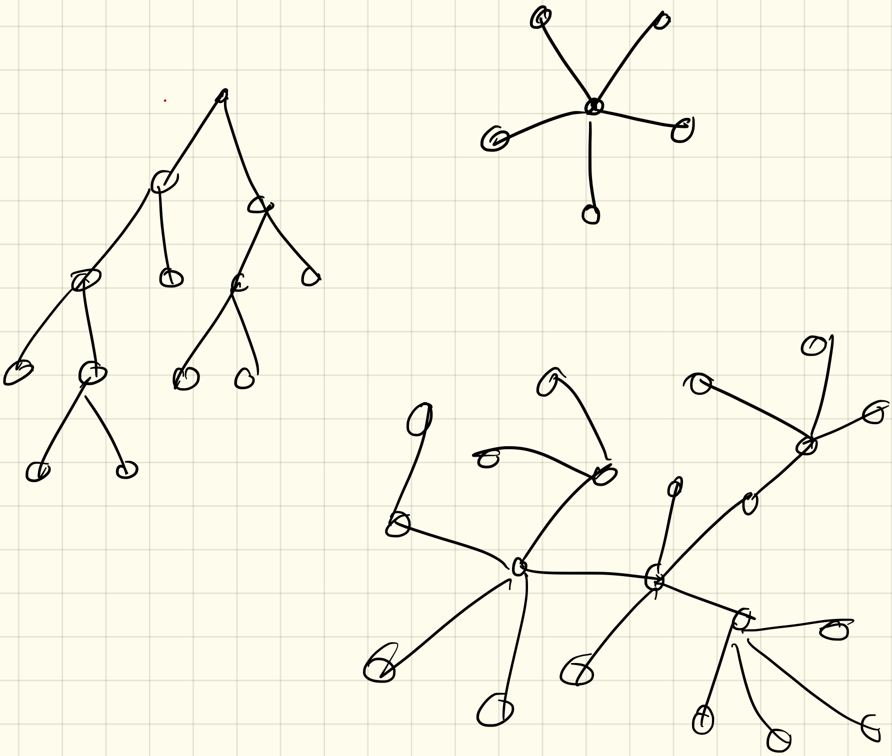




Tree:

A connected graph with no cycles

(Note: no root here!)



Representing graphs

How do we make this data structure?

Build it from ones we've seen already.

Adjacency (or vertex) lists:
↳ or vector

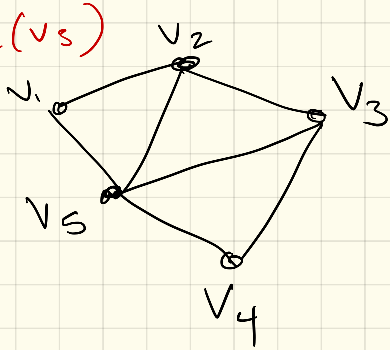
V_1 : $\underbrace{V_2, V_5}_{d(v_1)}$

V_2 : V_1, V_3 ← $d(v_2)$

V_3 : V_2, V_4, V_5 ← $d(v_3)$

V_4 :

V_5 :



in terms of n & m

1 per vertex (list)

Size: $n + 2m = O(m+n)$

Lookup: Time to check if V_i + V_j are nbrs:

$O(n)$ → $d(v_i)$ or $d(v_j)$

but – tricky! list or vector.

Implementation:

We call these vertex lists,
but don't have to
use lists

Options:

- vector
- list
- BST (?)

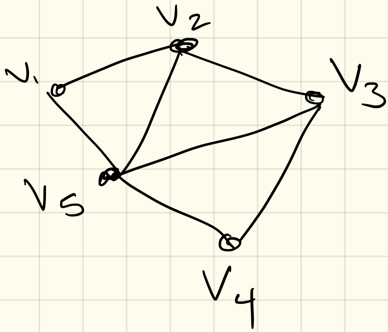
Trade-offs:

Binary search

O(1) - insertion

Adjacency Matrix

	v_1	v_2	v_3	v_4	v_5
v_1	X	1	0	0	1
v_2		X	1	0	1
v_3			X	1	1
v_4				X	1
v_5					X



space: $O(n^2)$

check nbr: $O(1)$

$A[i][j] == 1$?

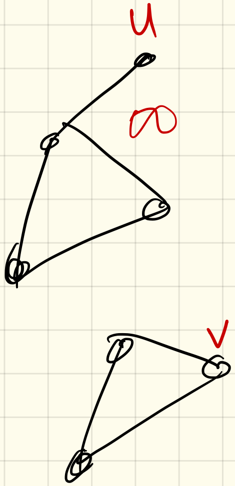
Which is better?

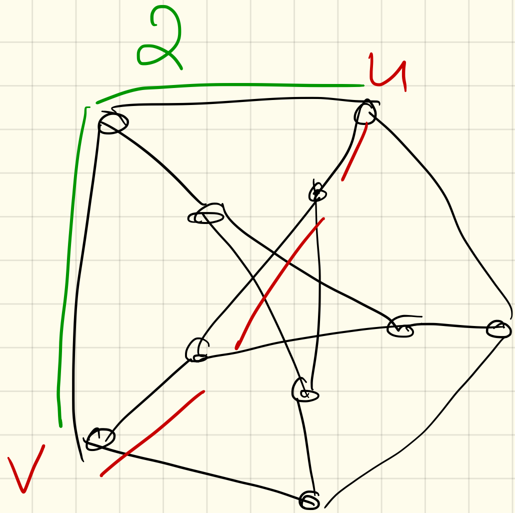
Depends!

	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 $u{\to}v \in E$	$O(1)$	$O(1+\deg(u)) = O(V)$	$O(1)$
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	$O(1)$	$O(\deg(u)+\deg(v)) = O(V)$	$O(1)^*$

Dfn:

- G is <u>connected</u> if $\forall u, v$, there $\exists$ path from u to v.
- The <u>distance</u> from u to v, $d(u,v)$, is equal to the # of edges on the minimum u,v-path





Algorithms on graphs

Basic 1st question:

Given any 2 vertices, are they connected?

Also: what is their distance?

↳ minimum path

How to solve?

Suggestion:

Suppose we're in a maze, seaching for something.

What do you do?

right hand rule – search until reach previously seen room

↳ depth first search

Pseudocode: two versions

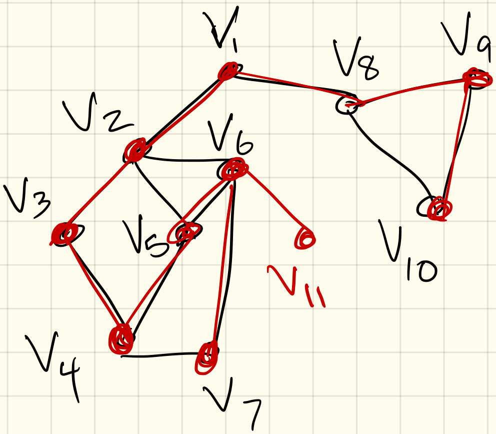
```
RECURSIVEDFS(v):
    if v is unmarked
        mark v
        for each edge vw
            RECURSIVEDFS(w)
```

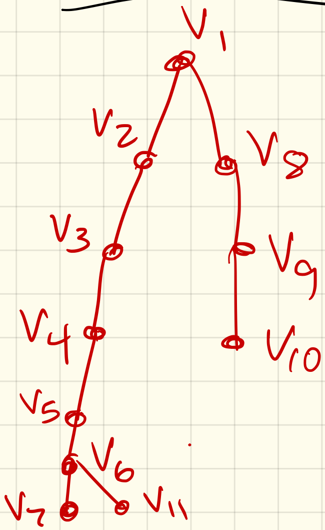
```
ITERATIVEDFS(s):
    PUSH(s)
    while the stack is not empty
        v ← POP
        if v is unmarked
            mark v
            for each edge vw
                PUSH(w)
```

runtime!

Really, building a "tree":

DFS tree:





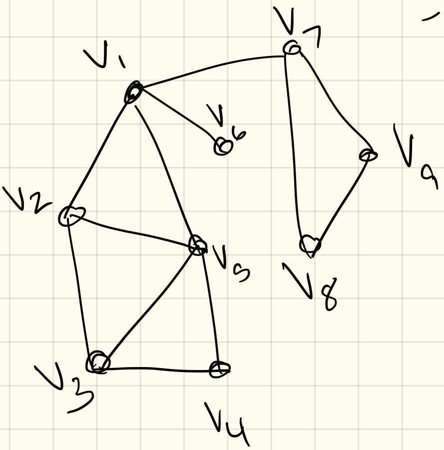
General traversal strategy:

```
TRAVERSE(s):
  put s into the bag
  while the bag is not empty
      take v from the bag
      if v is unmarked
          mark v
          for each edge vw
              put w into the bag
```

Q: Can we use a different "bag"?

BFS: use a queue

```
TRAVERSE(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```



BFS vs. DFS: