CS2100

Intro to graphs

Recap

- HW due Wed.
- Worksheet: coming (today)
- Review: 1 week from today
- Sample final coming tomorrow
- No lab - lecture instead
- Wed next week: 8am final
- No office hours today

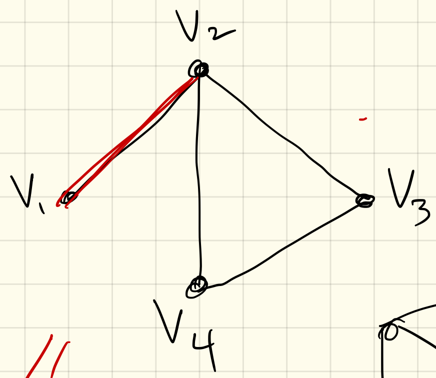
Graphs

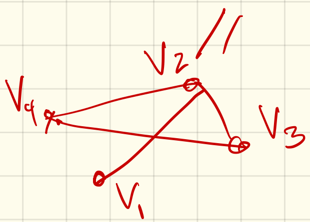
A graph $G=(V,E)$ is an ordered pair of 2 sets:

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

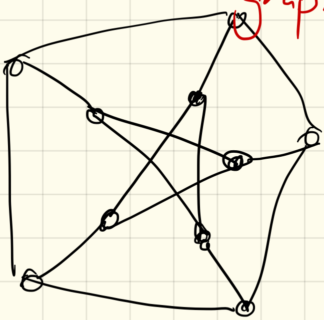
E = edges = $\{\{v_1,v_2\}, \{v_2,v_3\}, \{v_3,v_4\}, \{v_2,v_4\}\}$

View:





Peterson graph



Why?

They model everything!
(non-hierarchical, non-linear)

Examples

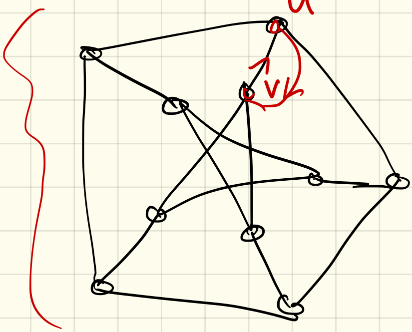
- connections on social media
- road networks
- internet

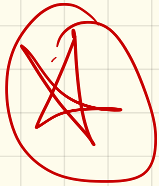
⋮

More dfns:

G is undirected if edges are unordered pairs

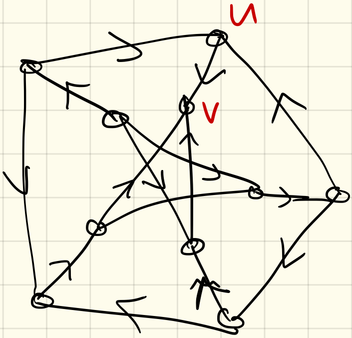
so $\{u,v\} = \{v,u\}$ ← endpoints



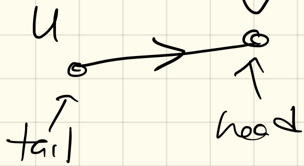


G is directed if edges are ordered pairs

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

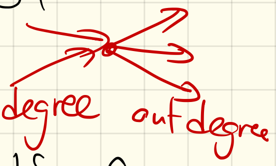


(u,v)



Dfns cont:

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

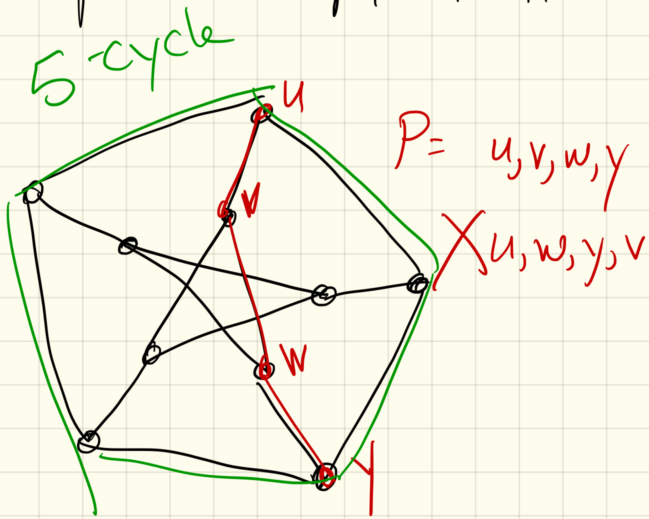


A path $P = v_1, \dots, v_k$ is a set of vertices with $\{v_i, v_{i+1}\} \in E$

(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 $v_1 = v_k$.



Lemma: (degree-sum formula)

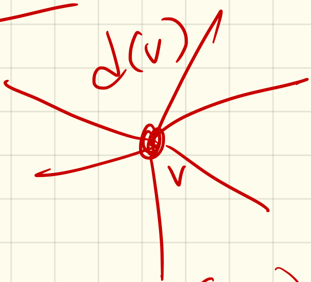
$$\sum_{v \in V} d(v) = 2|E|$$

handshaking lemma

pf:

sum over all vertices of deg of vertex $= d(v_1) + d(v_2) + \cdots + d(v_n)$

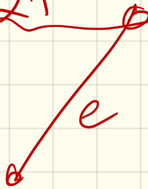
G



(sum over each vertex)

G

$$\underbrace{2+2+\cdots+2}_{\#\text{ of edges}}$$



(sum over each edge)

Size of G:

2 parameters:

$|V| = n$

$|E| = m$

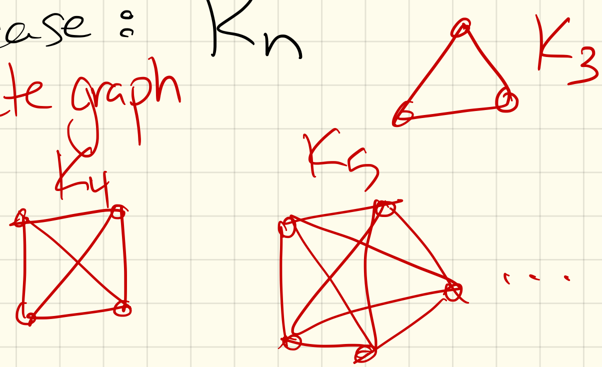
How big can m be in terms of n?

$$m \le \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

$$= O(n^2)$$

Worst case: K_n

Complete graph

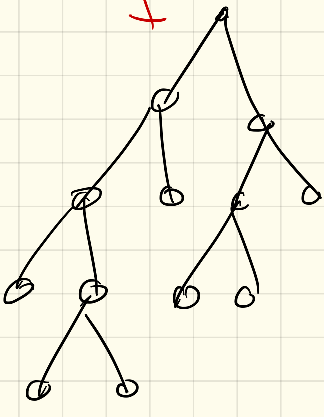


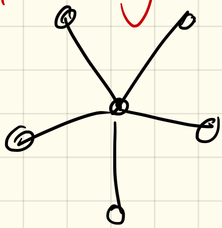
Tree:

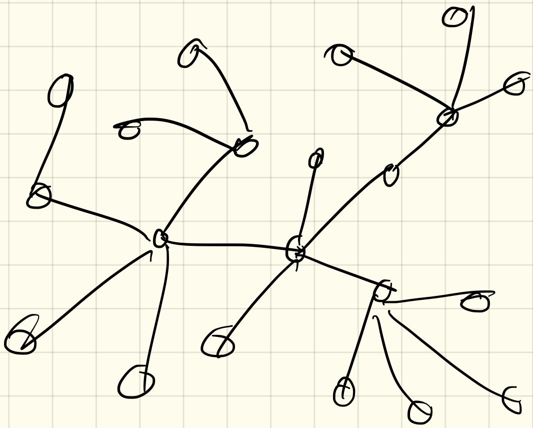
A connected graph with no cycles

(Note: no root here!)

→ $n-1$ edges $= m$



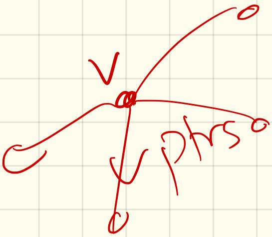




Representing graphs

How do we make this data structure?

- pointers!
  - ↳ list-like



Adjacency (or vertex) lists :

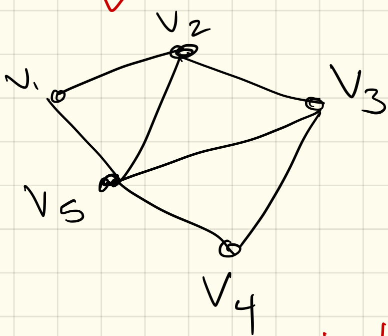
v_1 : v_2, v_5

v_2 : v_1, v_5, v_3

v_3 :

v_4 :

v_5 :



Size : n "lists", each ~~size~~ upper bnd $\leq n-1$

$O(n+m)$

Lookup : Time to check if v_i + v_j are nbrs :

$O(n)$

(or $O(\log n)$)

Implementation:

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

Options:

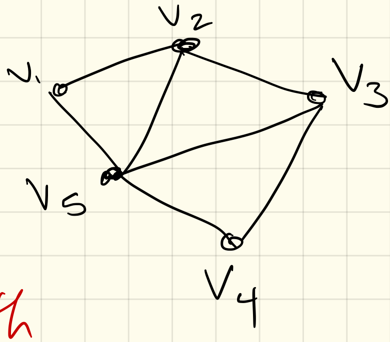
- lists
- vectors
- ⋮

Trade-offs:

insert/remove vs. lookup

Adjacency Matrix

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



directed: need both "halves" of matrix

space: $O(n^2)$

check nbr: $O(1)$

$A[i][j]$

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	$O(1)$	$O(\deg(u)+\deg(v)) = O(V)$	$O(1)^*$

$O(n^2)$ space

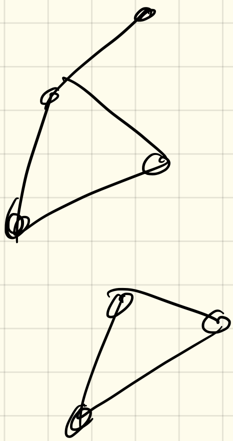
$O(n+m)$

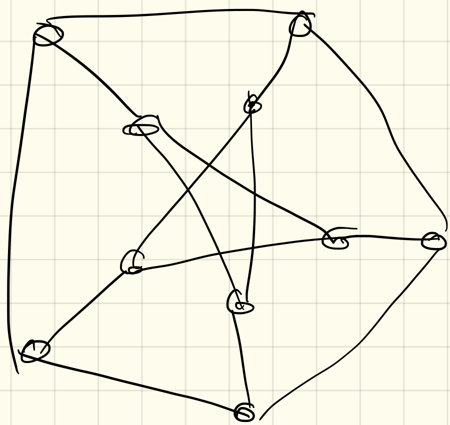
Dfn:

paths

- G is connected if $\forall u, v$, there $\exists$ path from u to v.

- The distance 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?

How to solve?

Suggestion:

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

What do you do?

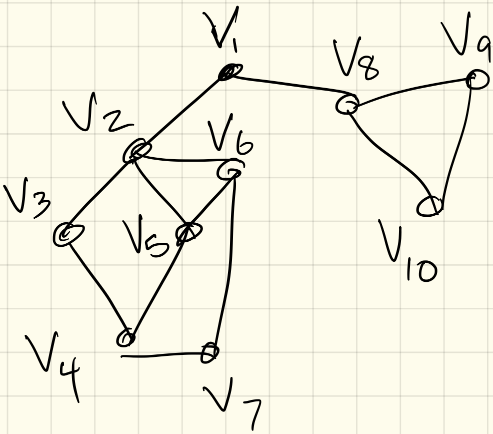
Pseudocode: two versions

```
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)
```

Really, building a "tree":

DFS tree:



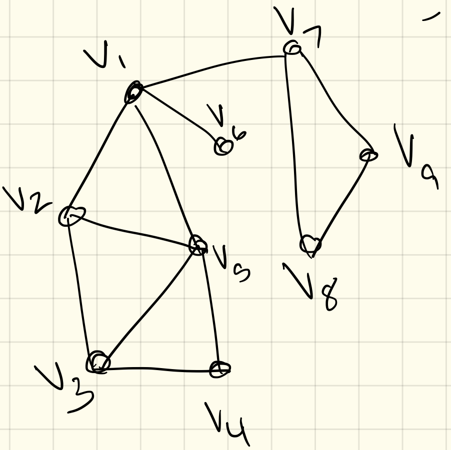
General traversal strategy:

```
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: