CSCI 3100

Today: Graphs

Announcements

- Boeing scholarships
- HW due Monday
- HW0 - back + posted
  (+ I think I fixed blackboard...)

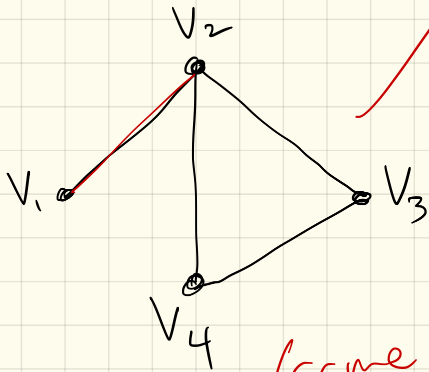
Graphs

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

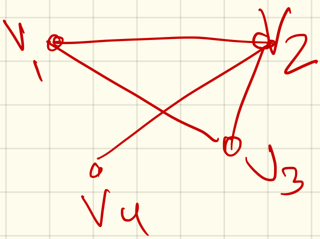
V = vertices $= \{v_1, v_2, v_3, v_4\}$

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

View:



(same!)



Why?

(my favorite!)

They model everything!

Examples

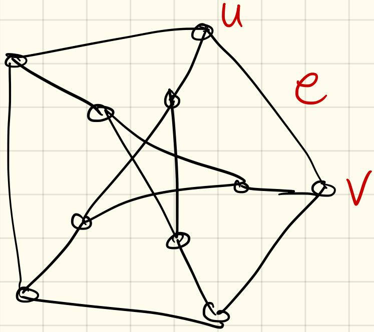
- social network
- roads
- connectivity
- sensor nework
- communication

⋮

More dfns:

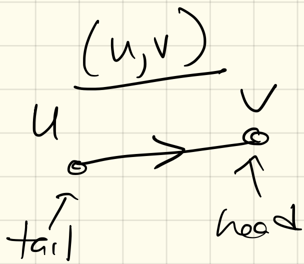
G is undirected if edges are unordered pairs

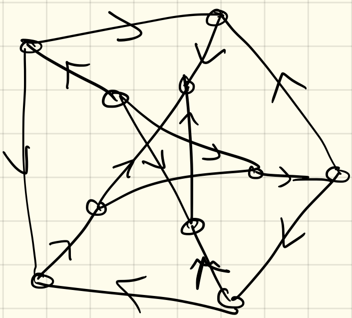
so $\{u,v\} = \{v,u\}$



G is directed if edges are ordered pairs

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





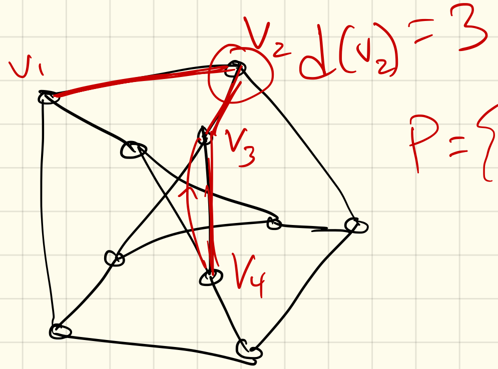
Dfns cont:

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

A **path** $P = v_1, \dots, v_k$ 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 $v_1 = v_k$.



$P = \{v_1, v_2, v_3, v_4, v_3\}$

__Lemma:__ (degree-sum formula)

$$\underbrace{\sum_{v \in V} d(v)}_{\text{sum degrees of all vertices}} = 2|E|$$

__pf:__

Consider 1 edge:

has 2 vertices in is connected to

→ each edge contributes +2 to sum on left side

→ $= 2|E|$ ▢

__Why?__

Size of G:

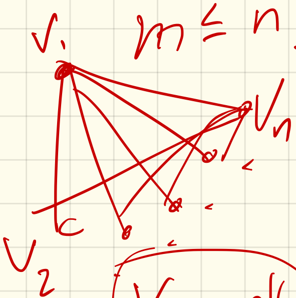
2 parameters:

$$|V| = n$$

$$|E| = m$$ ← # of pairs

How big can m be in terms of n?

edges in a graph with n vertices:



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

$$= \sum_{i=n-1}^{\text{down to } v_1} i = O(n^2)$$

K_n – all edges

<u>trees</u>: acyclic graph, connected

↳ how many edges?

$$m = n-1$$

Representing graphs

How do we make this data structure?

- arrays or lists
- matrix

more options.. –

Adjacency (or vertex) lists:

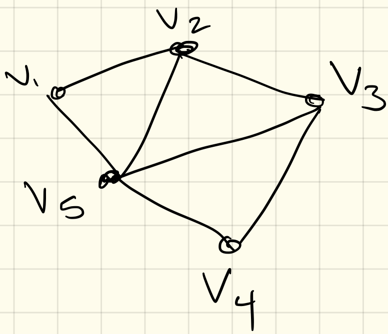
v_1: v_2, v_5

v_2: v_1, v_3, v_5

v_3: v_2, v_4, v_5

v_4:

v_5:



Size: $\cancel{n^2}$ $O(n+m)$

Lookup: Time to check if $v_i + v_j$ are nbrs: $O(n)$

Implementation:

More buried data structures!

Could use:

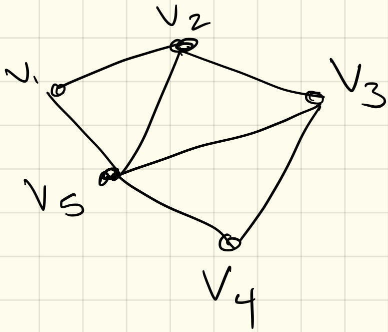
linked ← assume

array

issues w/ insertion, sorting, ...

Adjacency Matrix

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



directed:
use whole matrix

space: $O(n^2)$
check nbr: $O(1)$

Which is better?

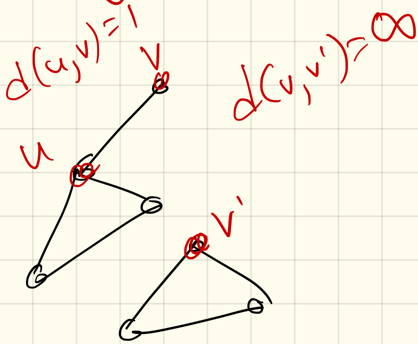
Depends!

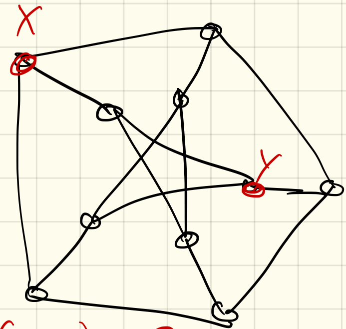
	Adjacency matrix	Standard adjacency list (linked lists)	Adjacency list (hash tables)
Space	$\Theta(V^2)$	$\Theta(V+E)$	$\Theta(V+E)$
Time to test if $uv \in E$	$O(1)$	$O(1+\min\{\deg(u),\deg(v)\}) = O(V)$	$O(1)$
Time to test if $u{\to}v \in E$	$O(1)$	$O(1+\deg(u)) = O(V)$	$O(1)$
Time to list the neighbors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

(graphs are unweighted)





$d(x,y) = 2$

Algorithms on graphs

Basic 1st question:

Given any 2 vertices, are they connected?

Also: what is their distance?

How to solve?

BFS

DFS

Suggestion:

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

What do you do?

Depth F S:

go left until revisit

back up & try next leftmost

⋮

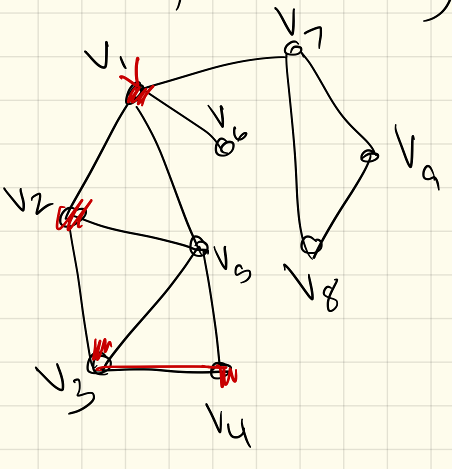
Pseudocode : two versions

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

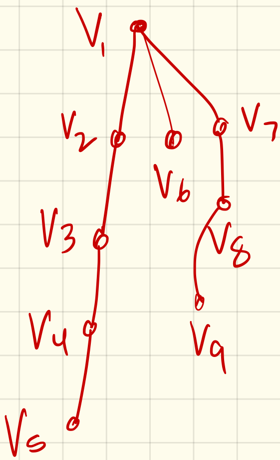
```
ITERATIVEDFS(s):
    PUSH(s)                         O(1)
    while the stack is not empty
        v ← POP                     O(1)
        if v is unmarked
            mark v
            for each edge vw
                PUSH(w)             O(1)
```

$O(m+n)$ total

Really, building a "tree":



DFS tree:



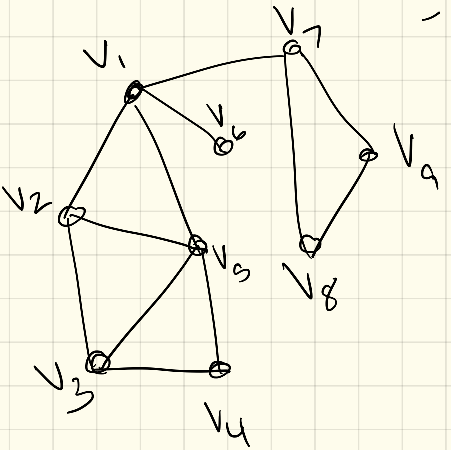
General traversal strategy:

```
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: