CS 180 - Graphs

Note Title 1/22/2010

Announcements

- I'll post room for review session next Friday.
- Program due Sunday by 11:59pm.

Recap:

Topics

- Basic C++ & run-times
- Stacks
- Queues
- Lists
- Vectors
- Sorting
- Binary Trees
- BST
- AVL trees
- Huffman Trees
- Hashing
- Graphs
- Heaps

represented on worksheet

Graphs

$|V| = n$

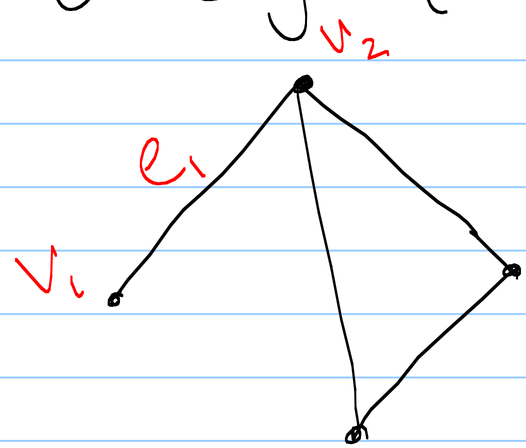
$|E| = m$

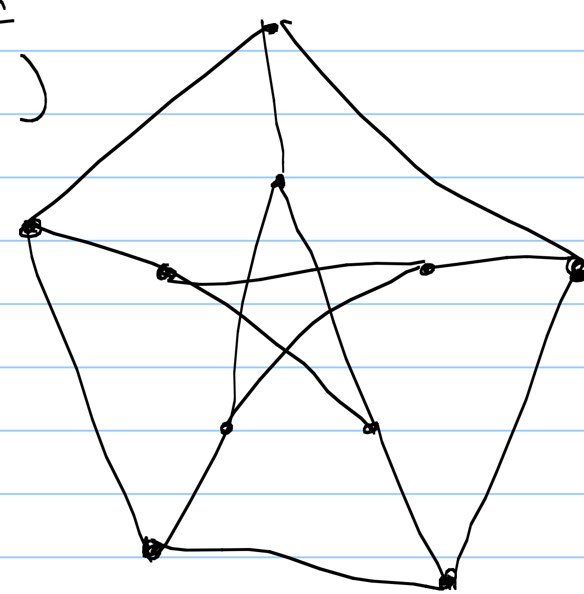
vertex edge

A graph $G = (V, E)$ is a set containing 2 other sets V & E

V = vertices

E = edges (or pairs of vertices)





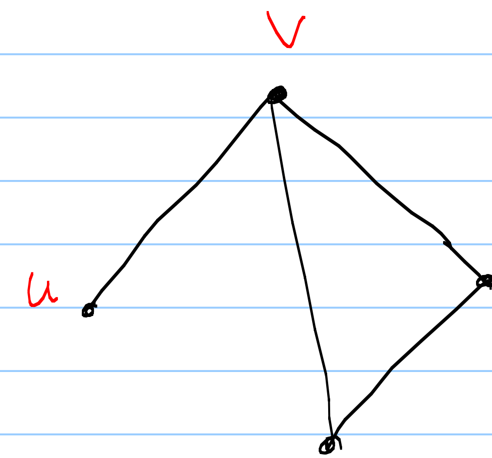
$e_1 = \{v_1, v_2\}$

Examples:

- routes – road networks
  ↳ specialize
- facebook
- games
  (3-dim meshes)
- internet
- collaboration

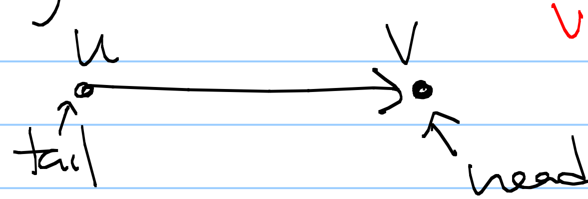
Definitions:

- G is <u>undirected</u> if every edge is an unordered pair, so $\{u,v\} = \{v,u\}$

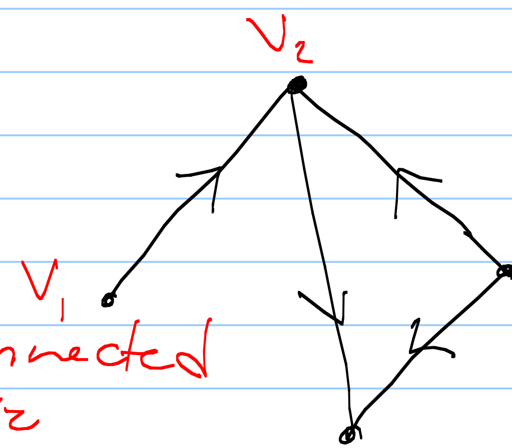


- G is <u>directed</u> if every edge is an ordered pair

$e=(u,v)$

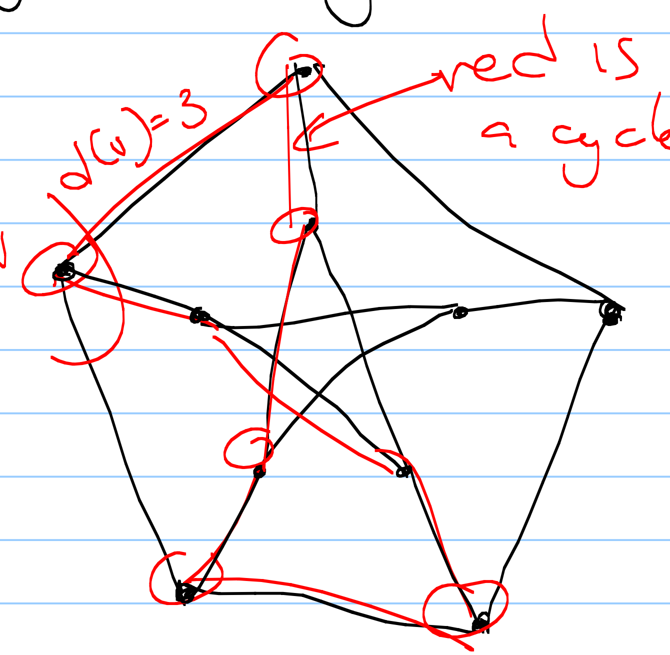


v_1 is connected to v_2



Definitions:

- The degree of a vertex v, $d(v)$, is the number of adjacent edges
- A path $P = v_1, v_2, \ldots, v_k$ is a set of vertices such that $\{v_i, v_{i+1}\} \in E$ (usually no repeated vertices) V
- A path is simple if all vertices are distinct
- A path is a cycle if it is simple except for $v_1 = v_k$



(relates m & n)

Lemma (degree-sum formula):

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

counting every edge-vertex incidence

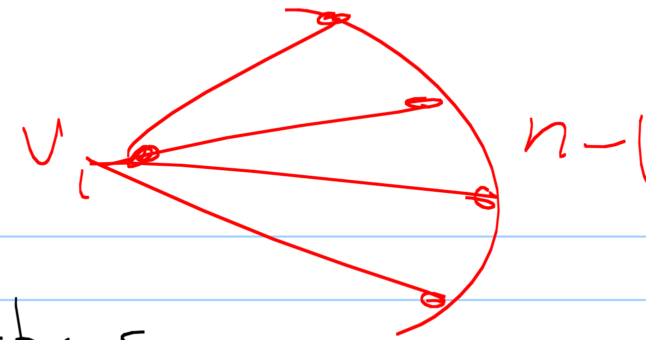
proof:

How to analyze:

Let G have n vertices.

How big can m be?

$$m = O(n^2)$$



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

Each vertex connects to $\le n-1$ other vertices

$\Rightarrow d(v) \le n-1$

$$\overbrace{2m = \sum_{v\in V} d(v)}^{\text{deg-sum}} \le \sum_{v\in V} (n-1) = n(n-1)$$

$$2m \le n(n-1)$$

So – how to store these?

→ list for each vertex
(of edges or vertices it connects to)

list of vertices
list of edge
} each could point to other

n, m

Vertex Lists:

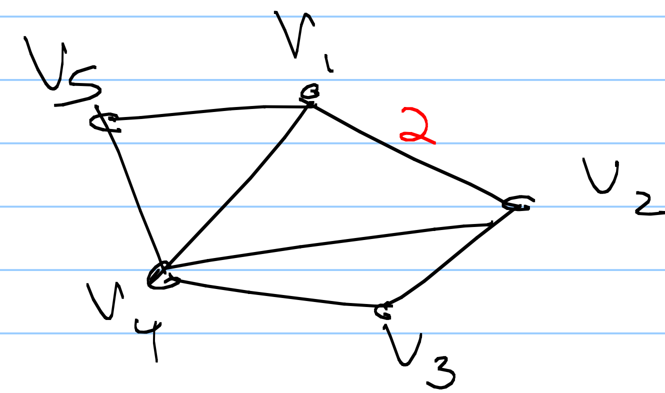
v_1 : $v_2 \to v_5 \to v_4$

v_2 : $v_1 \to v_3 - v_5$

v_3 : ⋮

v_4 :

v_5 :



$= \underbrace{O(n+m)}$ ← $\le O(n^2)$

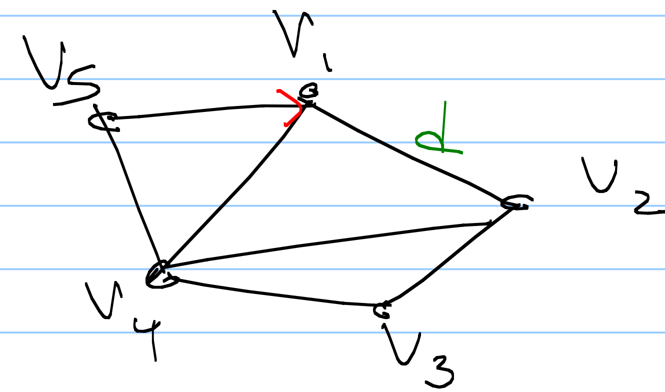
$\boxed{O(n)}$

if we can "sort",
$O(\log n)$

Suppose G is weighted

Adjacency matrices

	v_1	v_2	v_3	v_4	v_5
v_1	0	~~1~~ d		1	1
v_2	~~1~~ d	0	1		
v_3		1	0		
v_4	1			0	
v_5	1				0



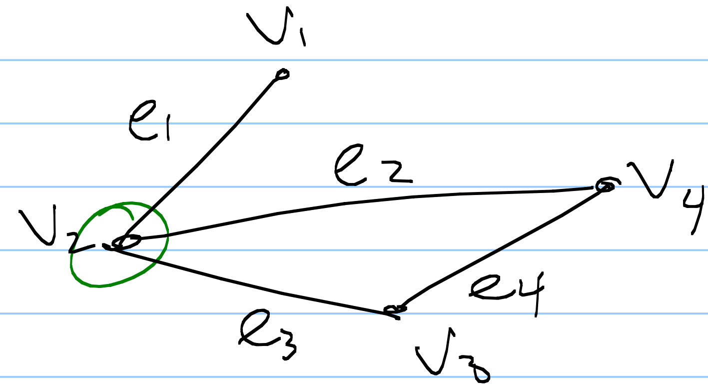
space: $O(n^2)$

Nice things: symmetric (if undirected G)

Worse space

Incidence Matrices

	e_1	e_2	e_3	e_4
v_1	1	0	0	0
v_2	1	1	1	0
v_3				
v_4				



space: $O(nm) \leq O(n^3)$

Which are best?

In terms of space – vertex lists

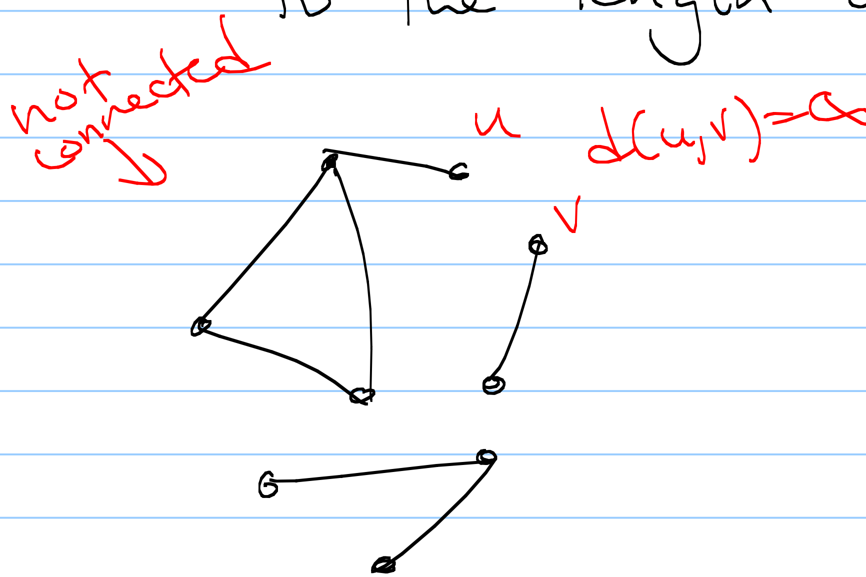
Sometimes use adjacency matrices.

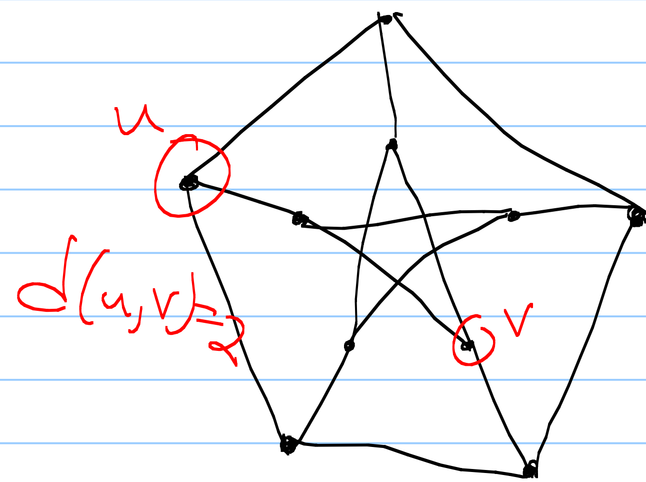
Dfns:

for all in there exists

- G is <u>connected</u> if $\forall u,v \in V$, $\exists$ a path from u to v

- The distance from u to v, $d(u,v)$, is equal to the length of a minimum u,v path





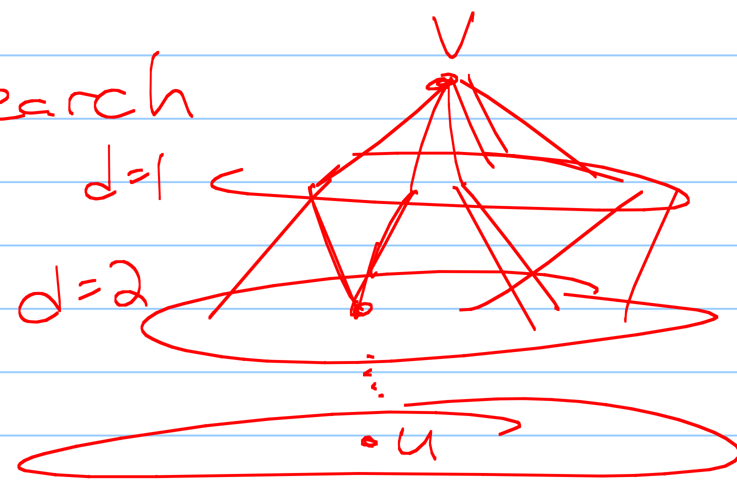
Algorithms on graphs

Basic question: given 2 nodes, are they connected?

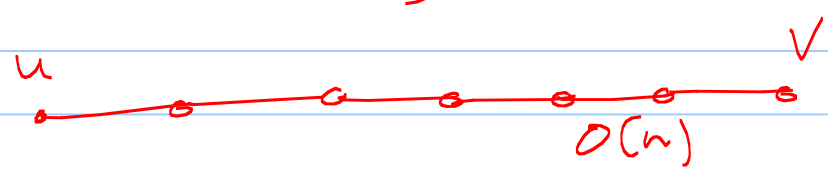
u + v

How to solve?

Breadth-First Search



$O(n+m)$



Suggestion:

Pretend we are in a maze searching for a treasure.

How do you proceed?

Depth First Search

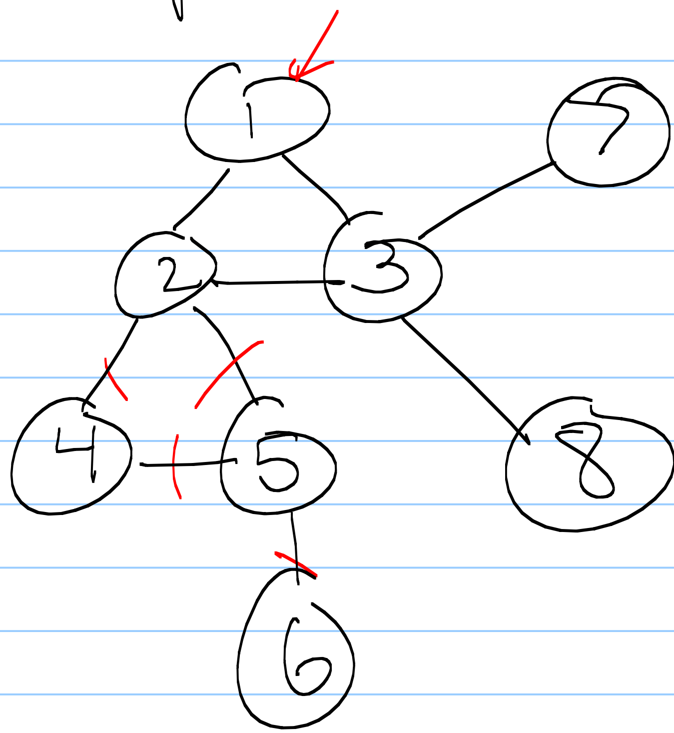
RecursiveDFS(u):

```
if u is unmarked
    mark u
    for each edge {u,v} ∈ E(G)
        RecursiveDFS(v)
    endfor
end if
```

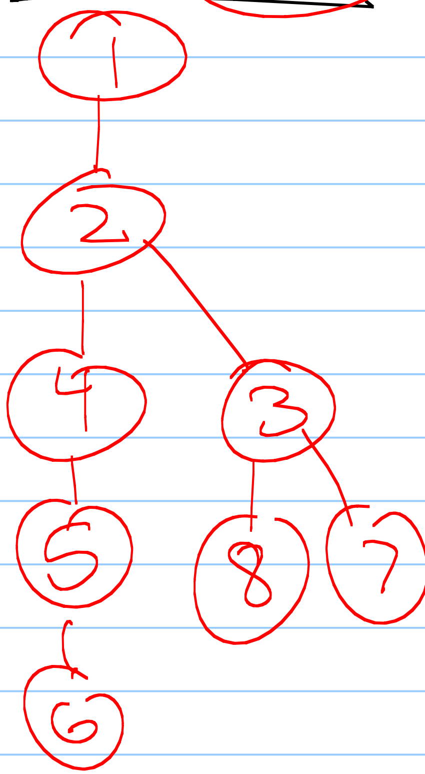
$O(m+n)$

For s-t connectivity, call DFS(s), & if t is ever marked then they're connected.

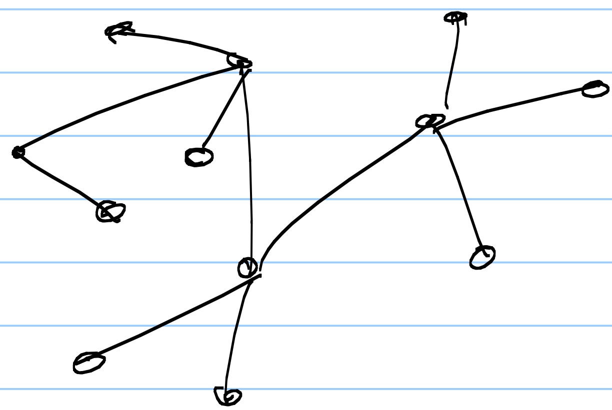
Example



DFS tree:



Dfn: A **tree** is a connected, acyclic graph.



A **leaf** in a tree is a vertex v with $d(v) = 1$.

Running time of DFS?