CS 314 - Graphs

Note Title 12/8/2011

Announcements

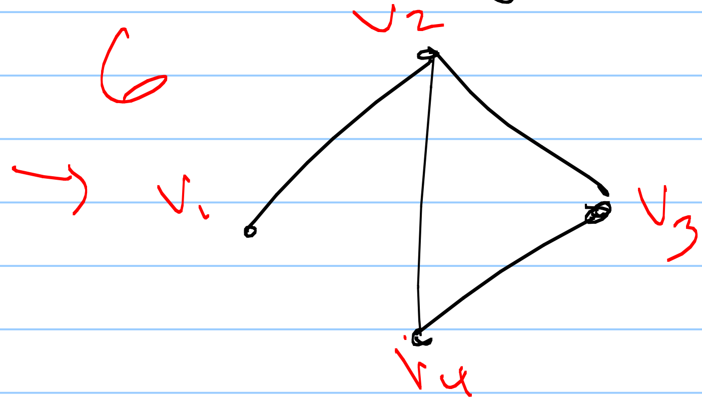
- HW3 due now
- HW4 posted - oral grading next Tuesday (sign up - next class)

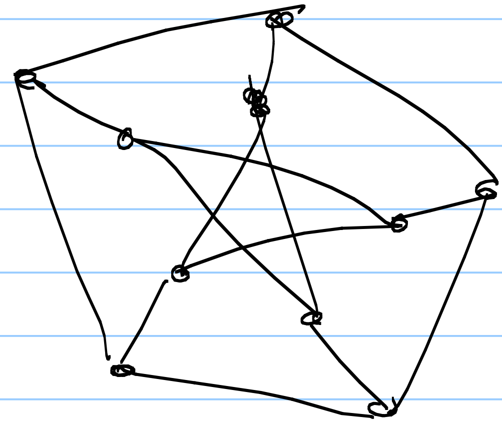
Graphs

A graph $G = (V, E)$ is a set of 2 sets V + E.

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

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





Why use graphs?

They can model anything!

Examples:

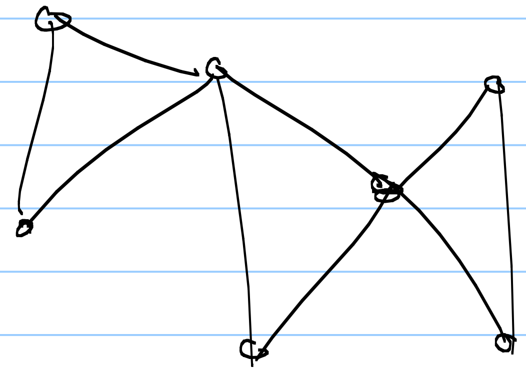
- social networks
- transportation network
- communication networks

⋮

Definitions

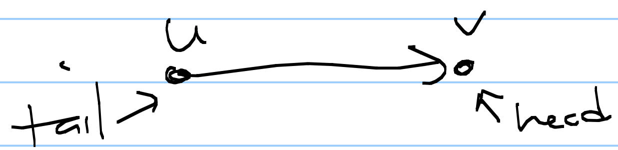
- G is <u>undirected</u> if every edge is an unordered pair

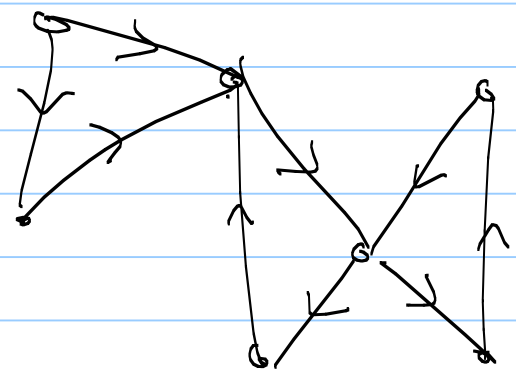
  so $\{u, v\} = \{v, u\}$



- G is directed is every edge is an ordered pair

  $e = (u, v) \neq (v, u)$





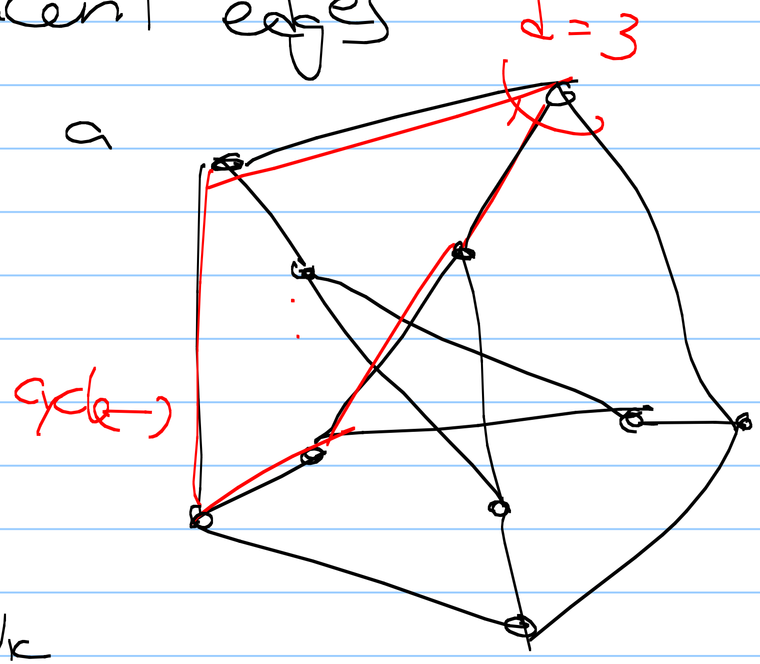
Dfns

- The <u>degree</u> of a vertex, $d(v)$, is the number of adjacent edges

- A <u>path</u> (walk) $P = v_1 \dots v_k$ is a set of vertices with $\{v_i, v_{i+1}\} \in E$

- A path is <u>simple</u> if all vertices are distinct

- A path is a <u>cycle</u> if it is simple except $v_1 = v_k$



$$\sum d(v) = 2m$$

Lemma: (degree-sum formula)

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

pf: Each edge has 2 adjacent vertices (or endpoints).

It adds +1 to $d(v)$ for 2 vertices.

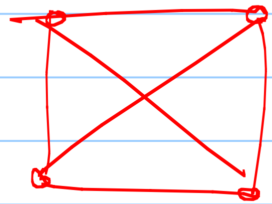
So sum on left has +2 for each edge, & hence $= 2 \cdot |E|$.

$\log m = O(\log n)$

Sizes of $|V|$ & $|E|$

We usually let $n = |V|$ and $m = |E|$.

How big can m be?



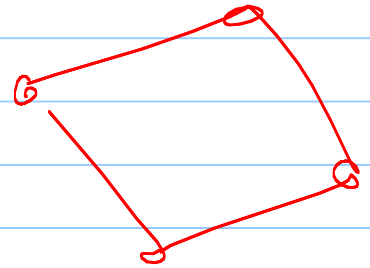
$$m \le \binom{n}{2} = \frac{n(n-1)}{2}$$

$$= O(n^2)$$

Trees: $n-1$ edges

Graphs on a computer

How can we construct this data structure?



(Adjacency lists)

Vertex Lists

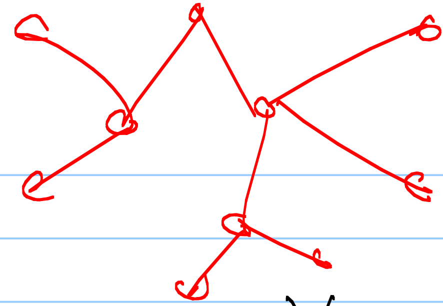
V_1 : V_2, V_5

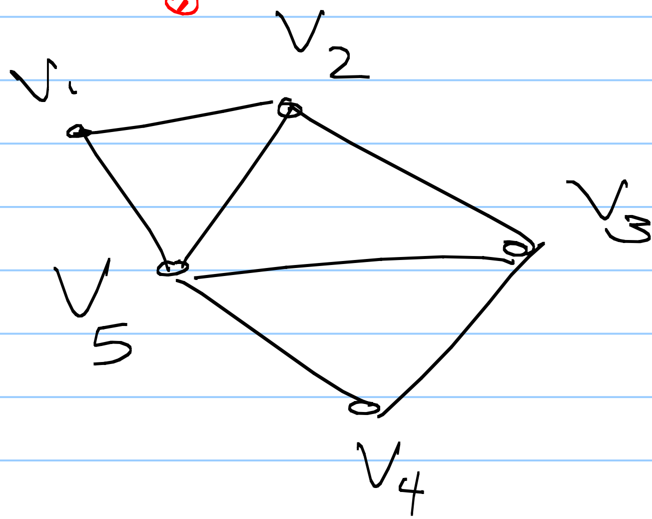
V_2 : V_1, V_3, V_5

V_3 :

V_4 :

V_5 :





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

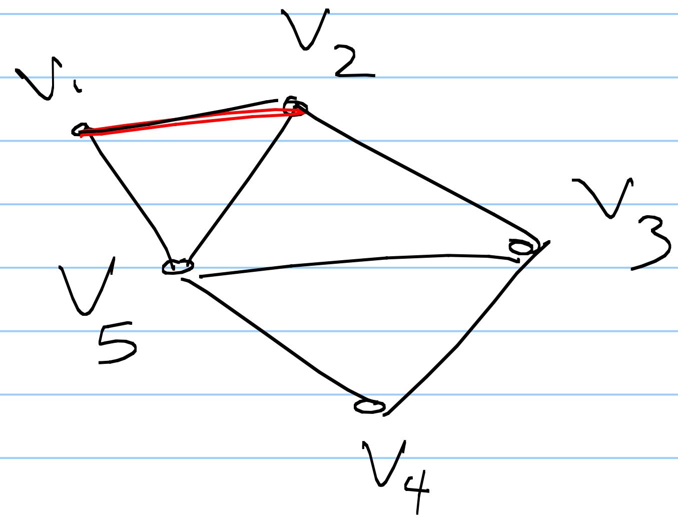
Check if v_i is neighbor of v_j : $O(d(v)) = O(n)$

Implementation

We call these vertex lists, but don't actually need lists.

Adjacency Matrix

$$
\begin{array}{c|ccccc}
 & v_1 & v_2 & v_3 & v_4 & v_5 \\
\hline
v_1 & 0 & 1 & 0 & 0 & 1 \\
v_2 & & 0 & 1 & 0 & 1 \\
v_3 & & & 0 & & \\
v_4 & & & & 0 & \\
v_5 & & & & & 0
\end{array}
$$



space: $O(n^2)$

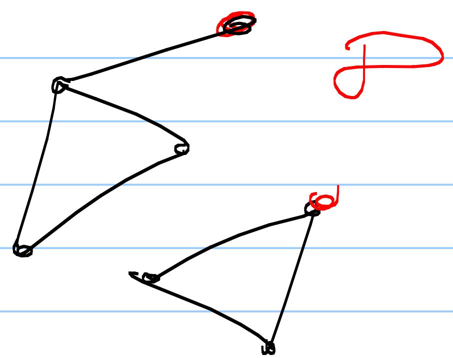
check neighbor: $O(1)$

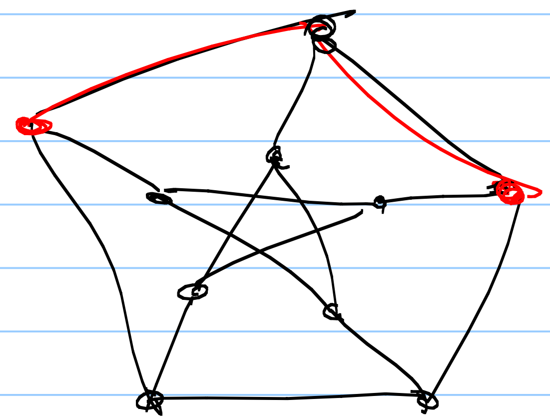
Which is best?

Just depends.

Dfns

- G is connected if for all u & v, there is a path from u to v.
- The distance from u to v, $d(u,v)$, is equal to the length of the minimum u,v-path.





Algorithms on Graphs

Basic Question: Given 2 vertices, are they connected?

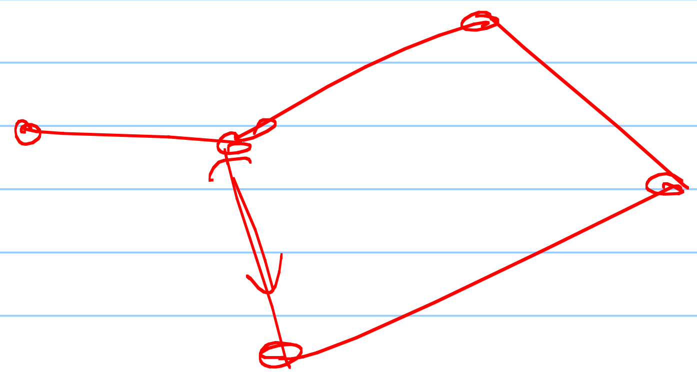
How to solve?

how far apart?

Suggetion:

- Suppose we're in a maze, searching for a treasure.

What do you do?



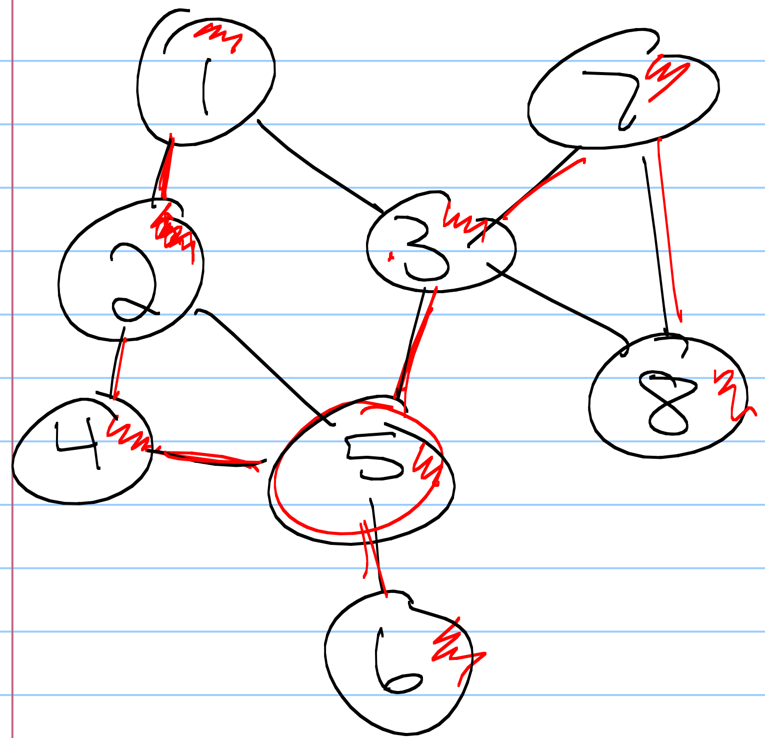
Pseudo code:

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

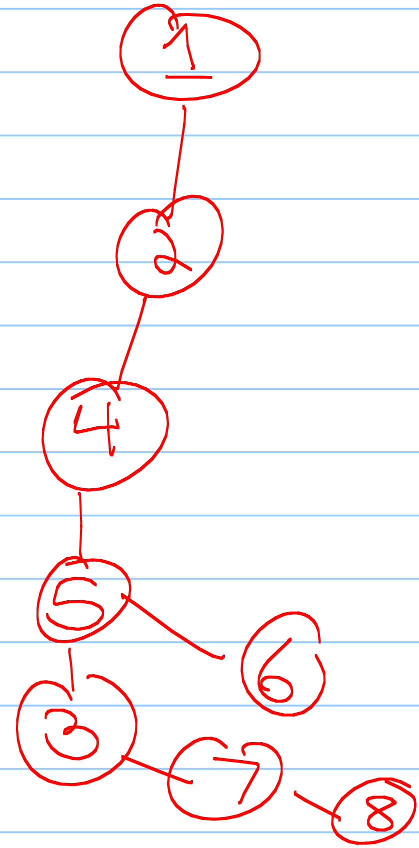
To check if s & t are connected,
call DFS(s).

At end, if t is marked, return true

DFS "tree":



DFS(1):



Iterative version:

```
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)
```

Runtime?   Each edge is added at most twice.

→ $O(m+n)$

Generalized traversal:

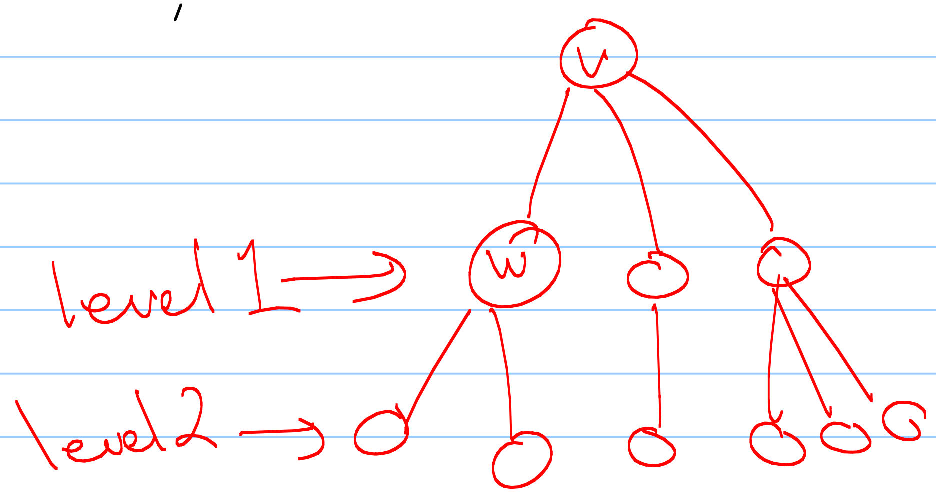
TRAVERSE(s):
```
put s in bag
while the bag is not empty
    take v from the bag
    if v is unmarked
        mark v
        for each edge vw
            put w into the bag
```

Q: What if we use something other than a stack?

queue

BFS: use a queue!

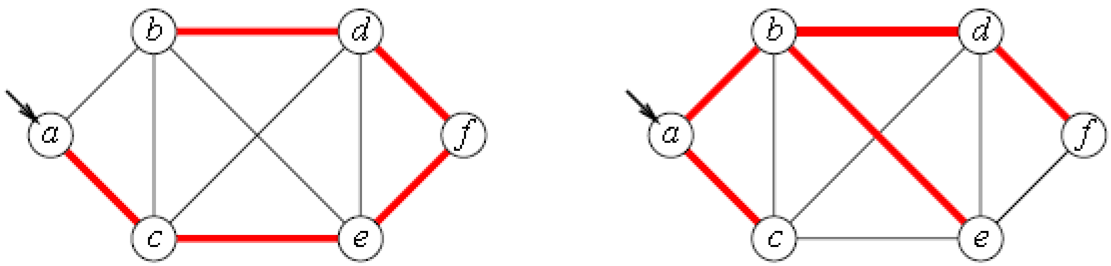
TRAVERSE(s):
put s in ~~bag~~ queue
while the bag is not empty
  take v from the bag
  if v is unmarked
    mark v
    for each edge vw
      put w into the bag



Runtime? $O(m+n)$

BFS versus DFS

- Both can tell if 2 vertices are connected
- Both can be used to detect cycles.
  How?
- Difference:



A depth-first spanning tree and a breadth-first spanning tree of one component of the example graph, with start vertex a.