CS180 - Graphs

Note Title 12/8/2011

Announcements

- 2 weeks left
  Final is Dec 17th at noon
  (Review last of class or Friday before)
- Checkpoint due tomorrow
- Lab on Thursday
- Instructor evals this week

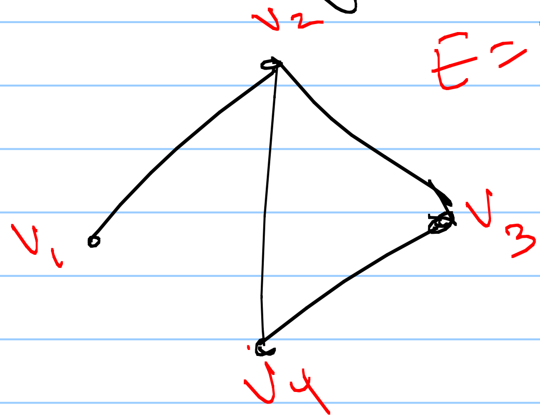
Graphs

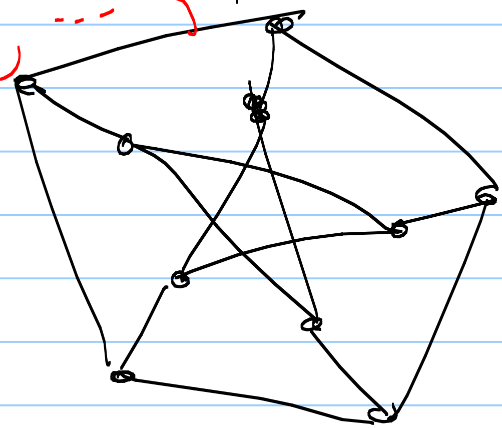
A graph $G=(V,E)$ is a set of 2 sets V + E.

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

E = edges (which are pairs of vertices)

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





Why use graphs?

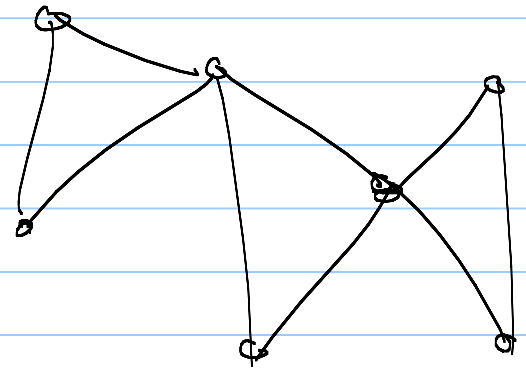
They can model anything!

Examples:

- Airport terminals
- Road networks
- Computer networks
- Social networks

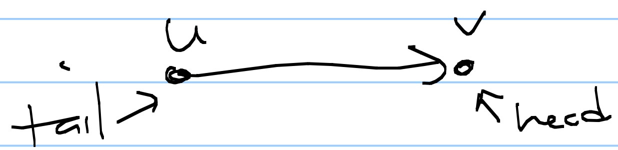
Definitions

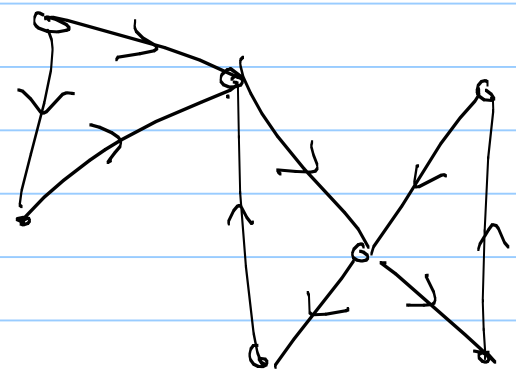
- G is undirected if every edge is an unordered pair
  so $\{u, v\} = \{v, u\}$



- G is directed is every edge is an ordered pair

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





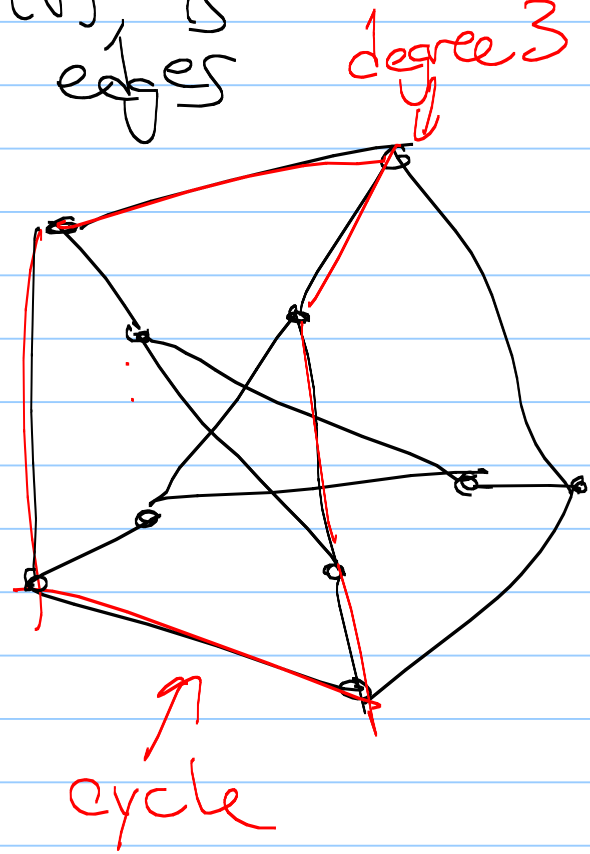
Dfns

- The 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 $\{v_i, v_{i+1}\} \in E$

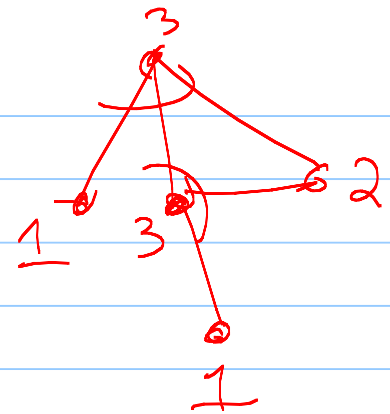
- A path is simple if all vertices are distinct

- A path is a cycle if it is simple except $v_1 = v_k$



Lemma: (degree-sum formula)

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



$3+3+1+1+2=10$

Why? LHS: counting all edges connected to each vertex.

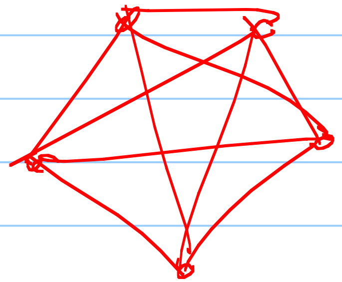
Every edge is connected to 2 vertices.

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

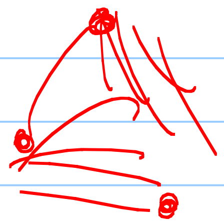
We usually let $n = |V|$ and $m = |E|$.

How big can m be? $m \le \frac{n(n-1)}{2}$, $m = O(n^2)$

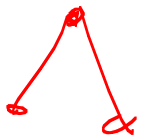
$\{1, 2, \dots, n\}$



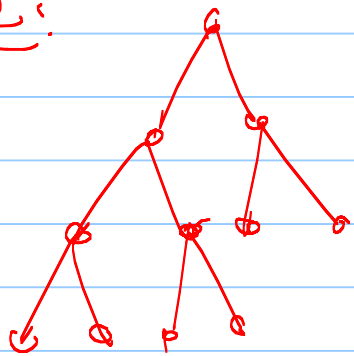
$$\binom{n}{2} = \frac{n!}{(n-2)!\,2!} = \frac{n(n-1)}{2}$$



$$(n-1) + (n-2) + (n-3) + \dots + 1 = \frac{n(n-1)}{2}$$



Tree:



n vertices in a tree

$m = n - 1$

Sparse graphs have $m \approx n$

Graphs on a computer

How can we construct this data structure?

Node - w/ pointers to other nodes

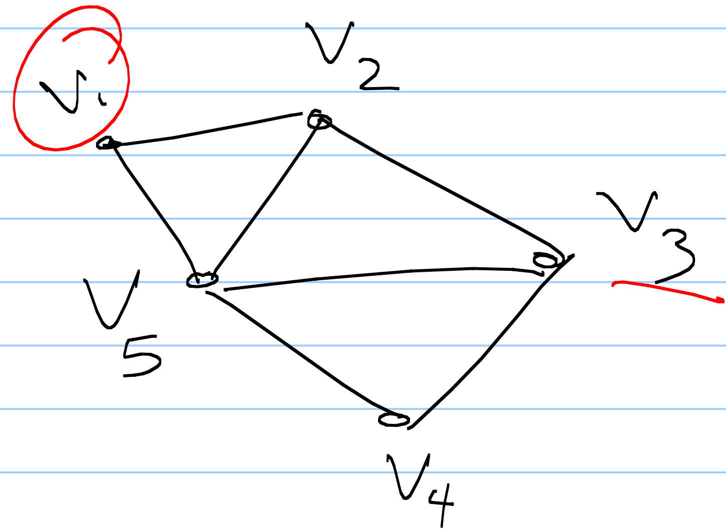
$\boxed{\text{city}}$

list of pointers
(or vector)

Vertex Lists (or vectors)

n vertices
m edges

v_1 : 2, 5
v_2 : 1, 3, 5
v_3 : 2, 4, 5
v_4 : ⋮
v_5 :



size : $2m$ (in list) $+ n = O(m+n)$

check if v_i is neighbor of v_j : $O(n)$
$O(d(v_i))$

Implementation

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

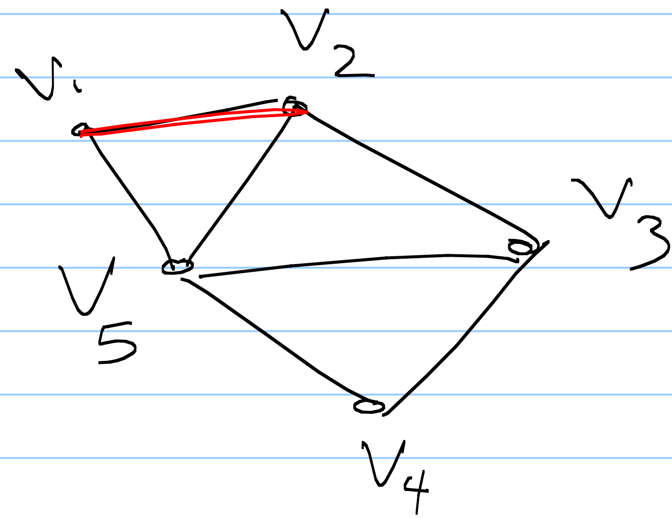
Can store in any auxiliary data structure.

Trade-offs: usual

- insert / delete
- keep sorted & have binary search

Adjacency Matrix

$$A \quad \begin{array}{c|ccccc} & V_1 & V_2 & V_3 & V_4 & V_5 \\ \hline V_1 & 0 & 1 & 0 & 0 & 1 \\ V_2 & 1 & 0 & 1 & 0 & 1 \\ V_3 & 0 & 1 & 0 & 1 & 1 \\ V_4 & 0 & 0 & 1 & 0 & 1 \\ V_5 & 1 & 1 & 1 & 1 & 0 \end{array}$$



space: $O(n^2)$ ← always worse

check neighbor: $A[i][j] == 1 \Rightarrow O(1)$

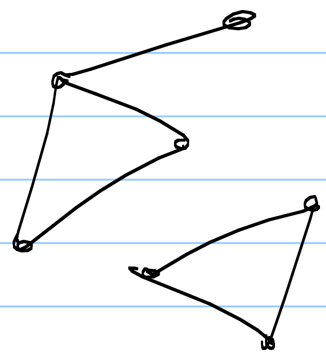
Which is best?

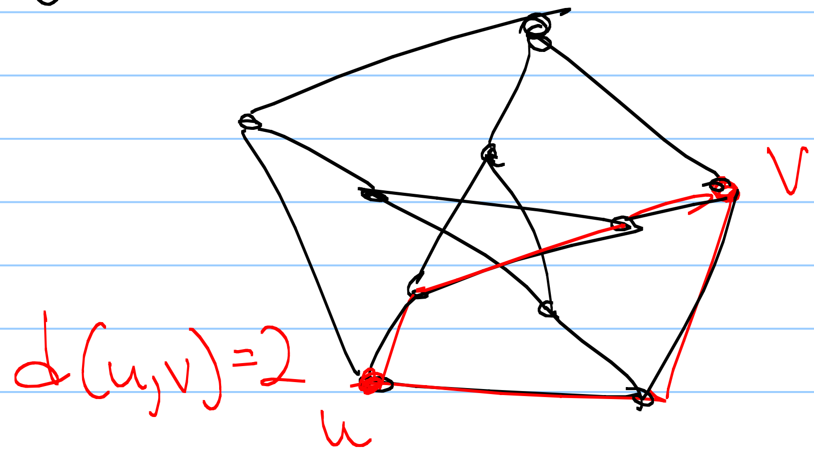
Just depends.

sparse graphs – use lists

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.

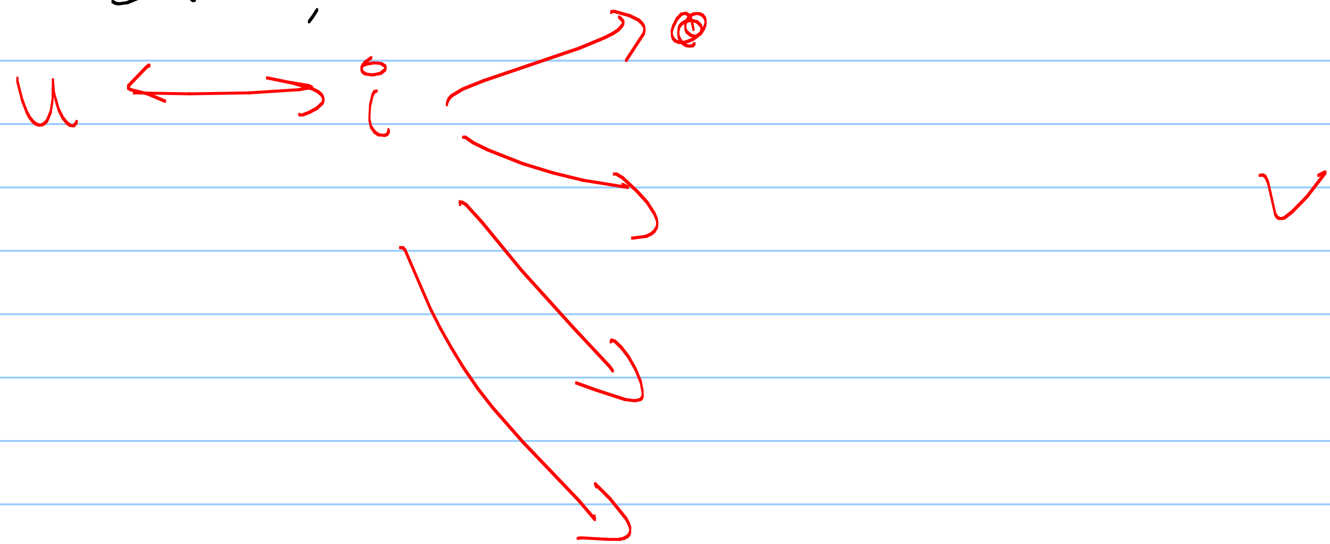




Algorithms on Graphs

Basic Question: Given 2 vertices, are they connected?

How to solve?



Suggetion:

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

What do you do?

RecursiveDFS(u):

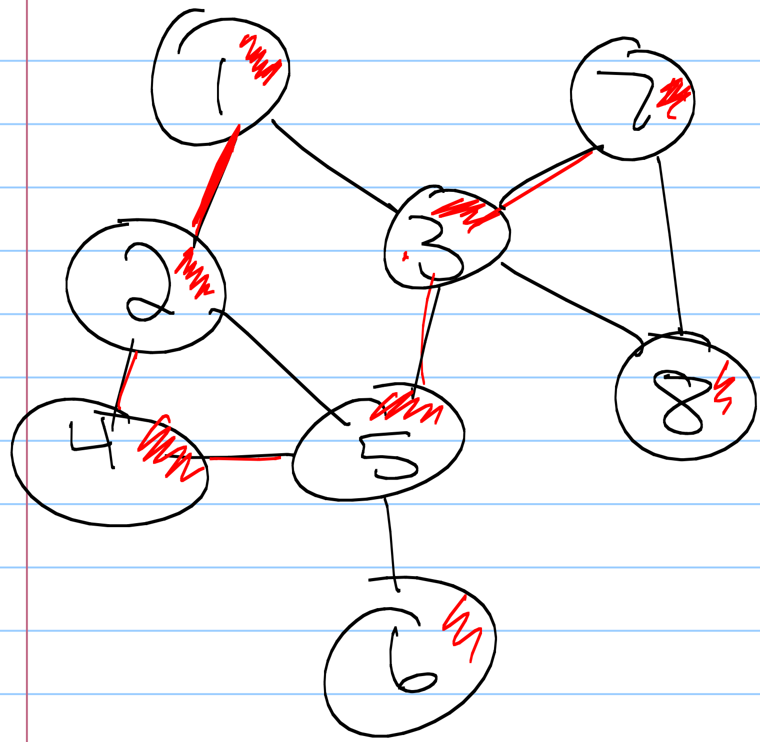
If u is unmarked: extra O(n) space
- mark u
- for each edge $\{u,v\} \in E$
  RecursiveDFS(v)

(depth-first search)

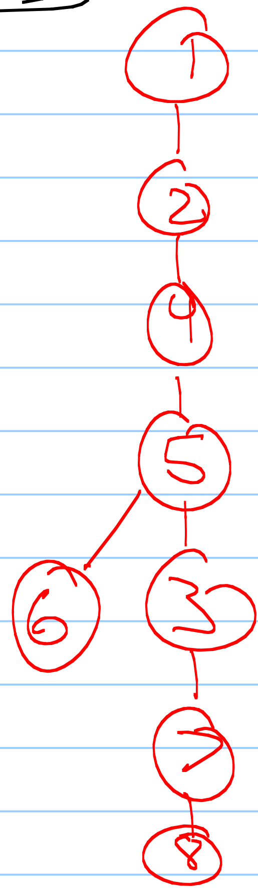
To check if s & t are connected,
call DFS(s).

At end, if t is marked, return true

DFS "tree":



DFS(1):



Another version of DFS

IterativeDFS(u):
  create empty stack S
  S.push(u)

  while S is not empty:
    v ← S.pop
    if v is not marked
      mark(v)
      for each edge vw
        S.push(w)

Iterative DFS (1):

Stack :