1/19/16

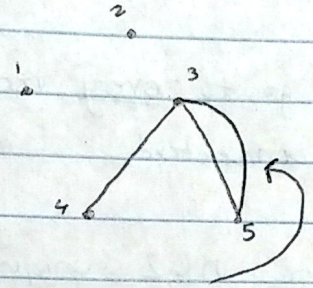
Ch 3 - Graphs

used frequently for industry + research

ex: ~~G(V, E)~~ G = (V, E)

V: the set of vertices (may be called nodes)

E: the set of edges (pairs of vertices)



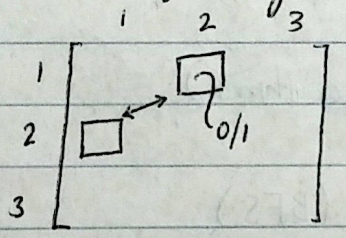
typically vertices don't have any weight on them
otherwise → weighted graph

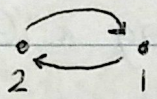
parallel edge: double edges
typically not present

How do you represent a graph?

1) adjacency matrix

- always has n^2 entries
- easy to check for specific edge between vertices





undirected graph: edge has no direction
→ matrix is symmetrical

- assuming simple graphs!

if weighted, you wouldn't put 0/1

2) Linked List

- entries are according to # of edges

1 → 2 → 5
2 → 1
3

- ordering doesn't matter only indicates direct links or edges, NOT paths

Path: sequence of edges that takes you from one vertex to another

usually mean simple paths -- no repetitions in edges/vertices

cycle: a path in which beginning and end vertex are the same

connected: if you are able to go to every vertex from one vertex -- the graph is connected

w/ n vertices, there are $nC2$ unique edges

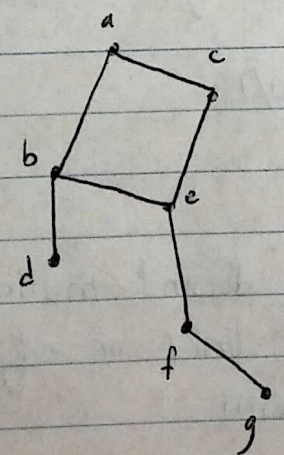
maximum $nC_2 \sim n^2$ (dense graph)

minimum $n-1$ (sparse graph)

algorithm may differ depending on density
b/c of redundancy, space requirement

ex: Exploring a Graph (Algorithms)

ex #1: Breadth First Search (BFS)
"look next door" -- distance of 1 then 2 then 3...



pick arbitrary starting point (a)
look at all neighbors of a (b, c)
explore b's neighors (e, d)
then go the c vertex b/c it came first (e)
pick d or e arbitrarily (d)
then e (f)
then f (e, g)
then g
done

# of edges in a graph w/ one cycle → n edges

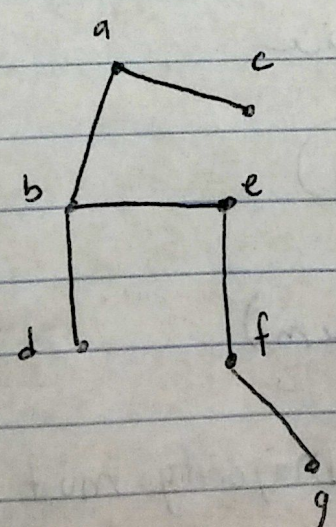
length of path: number of edges in a path
determines "close" or "far"
always look @ closest path

FIFO: determines order of vertices in BFS
g f e d c b
front of queue →

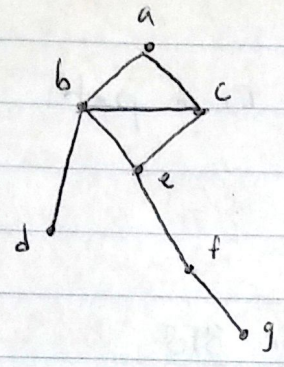
time complexity of BFS?
$O(|E|)$ or $O(e)$ where $e = |E|$ } for connected graph
$|E|$ number of edges

if graph is NOT connected, you jump to the next component and repeat the BFS algorithm
↓ this increases ~~time~~ time complexity
$O(|E| + |V|)$ ~ linear time in E

BFS tree: info about distances from starting point
→ shortest path from a
(only for unweighted)



if c was explored before b, then edge (c, e) would be highlighted instead of (b, e) but never both



Depth First Search (DFS)

"go all the way down the hallway"

from a, I go to b

b → d, stuck so back up once

b → e, e → f, e → g (stuck)

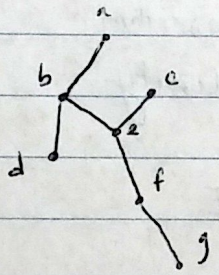
back up to f

back up to e

e → c (everything has been explored)

back up to initial starting point

DFS: tree: the path formed visiting the vertices the first time



DFS: info about cycles

backward edges: edges used to visit a vertex for the n^{th} time ($n > 1$)

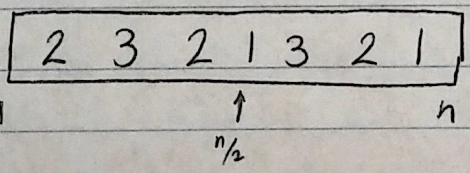
→ indicates cycles

every edge is looked @ twice

$O(e + n) \equiv O(|E| + |V|)$

ex: election (majority problem)

counting is $O(n)$



majority: must have more than $n/2$ votes

→ only one majority

possible techniques:

- sort the votes and check halfway line ($n/2$)
  - if crossed → candidate for majority!
    - then check # of votes

$O(n + n \log n)$

linear algorithm?

can only check once

implies we have to remove a ~~candidate~~ vote (but removal can't be arbitrary)

=> remove 2 arbitrary ^different votes (majority does not change) -- constant time

must double check the candidate to see if actual majority

if remainder of the removal process is 2 different votes

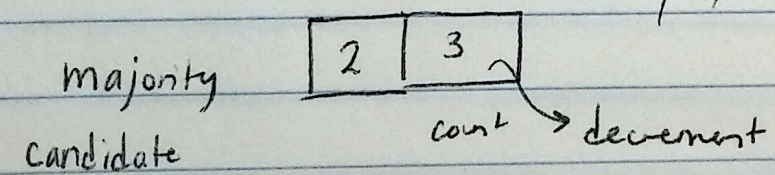
=> no majority

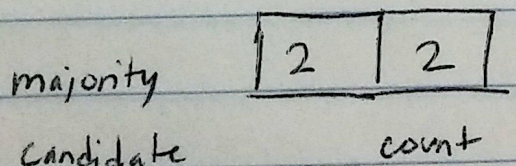
otherwise, remaining is a majority candidate

must specify how to choose 2 different votes!!

pick first entry

~~2 3 2 2 2 1~~





if same, increment the count

if different, decrement the count

if 0, remove as candidate and take the next as candidate