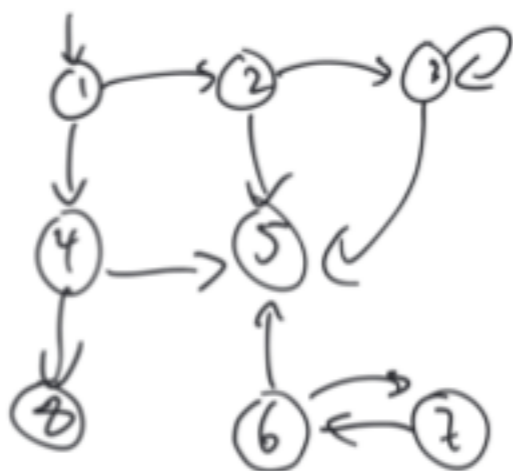


## Lecture 19: Graph implementation + traversals

- Graph demo
- Depth first & breadth-first search

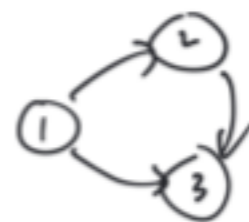
### Announcements:

- Visiting lecture today after class, Gates 214 (reception in 122)
- Project 5



explore every vertex  
reachable from a  
given vertex.

idea: make as much progress as possible  
(completely explore one path before backing  
out and exploring next)  
: Depth-first search.



- find all vertices.

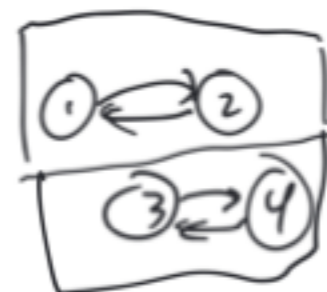
- detect cycles

- determine if a graph is connected:  
every vertex is reachable from  
every other vertex.

} it is an  
undirected  
graph

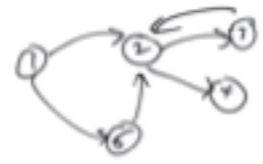
• break a graph up into  
components, many algorithms  
connected graphs.

connected  
only work on

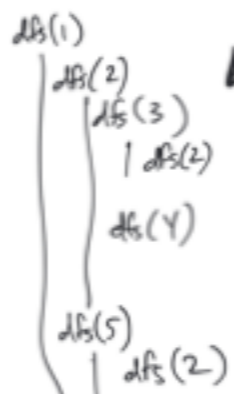


begin Depth-First Search (v)

visit v  
 for each child of v:  
 depth first search child.



seen = {1, 2, 3, 4, 5}  
 visit = {1, 2, 3, 4, 5}



Size of call stack  
 = # of verts on the path being considered.

visit all vertices reachable from v without traversing any vertices in "seen" set, update seen with all visited vertices.  
 depthFirstSearch(v, seen)  
 if v is in seen:  
 return // no verts reachable from v without traversing "seen"  
 visit v as seen (add v to set)  
 mark v as seen (add v to set)  
 for each neighbor u of v:  
 depthFirstSearch(u, seen)  
 → everything reachable from v has been visited!

runs once per vertex  
 once per edge

run time:

- do some work for each vertex
- do some constant work for every edge

(assuming visiting is const. time)  
 $O(|V| + |E|)$

total storage is constant for each stack frame.

longest possible path we would consider? } worst case  
 - paths never go through same vertex twice.

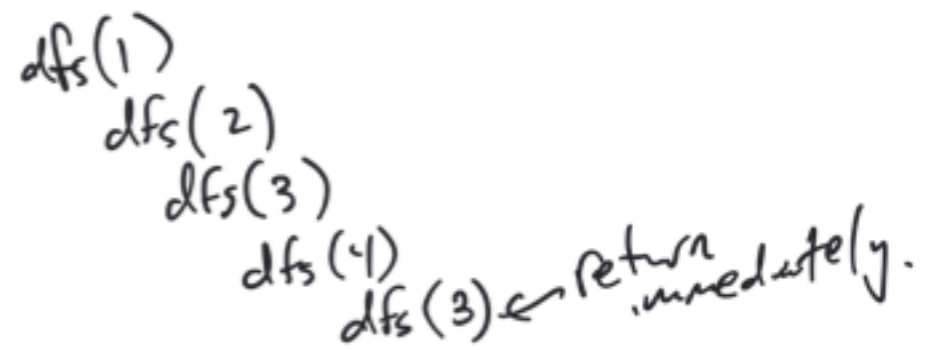
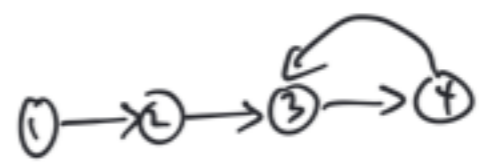
- longest path  $O(|V|)$   
 in fact,  $|V| + 1$

want to avoid linear search.

some better data structure for "seen" set.

$O(|V|)$  space.

- could use array (one entry per vertex)  $O(1)$  operation to find entry
- could add a field to each vertex, update it.
- option 3: use hashtable (average  $O(1)$  lookup time)



DFS iterative version.

we'll maintain a worklist of vertices still need to visit.

dfs(v):

Collection<vertex> worklist;

vertices u with v-u

invariant: not all children of elements of worklist have been visited, all parents have.

//init: visit(v);

for each child of v, add child to worklist.

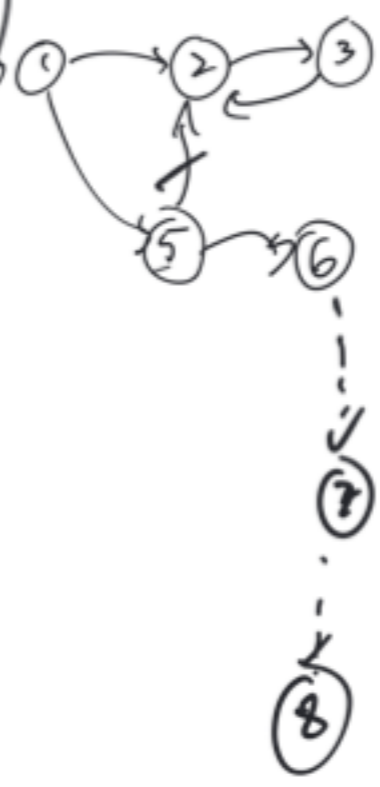
//term: worklist is empty (everything has been visited)

//progress: take something out of worklist, visit it, add all children to worklist.

want the last entry put in to be first that we process

use a stack!

push, adds to end of stack  
pop, removes from end of stack.

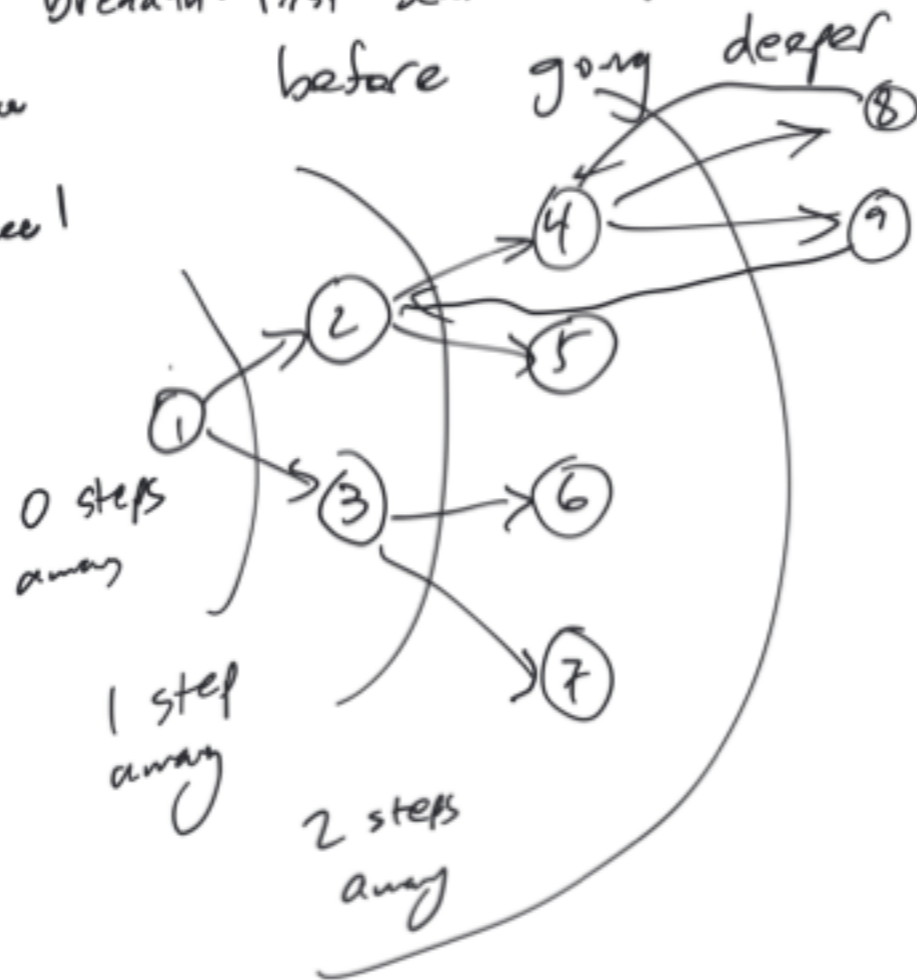


worklist	visited
1	1
2	5
5	6
2	2
5	3
2	
2	

worklist  
 +  
 ⇒ 2  
 3  
 4  
 5  
 6  
 7  
 8  
 9  
 10  
 11

visited  
 1 } distance 0  
 2 } distance 1  
 3 }  
 4 }  
 5 } dist. 2  
 6 }  
 7 }  
 8 } dist. 3  
 9 }

breadth-first search: go as wide as possible before going deeper



as possible

breadth first search:

- maintain a worklist of vertices to be visited

- repeatedly remove a vertex,

only if we haven't seen vertex yet.

visit it, add neighbors to WL.

to get breadth first, want to remove vertices in order added.