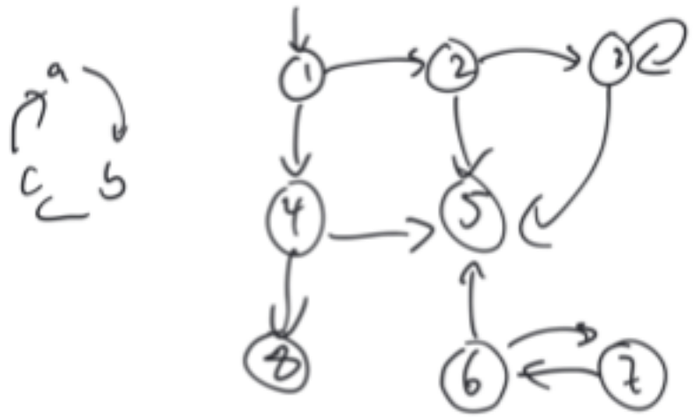
Lecture 19: Graph implementation + traversals

- Graph demo
- Depth first & breadth-first search

Announcements:

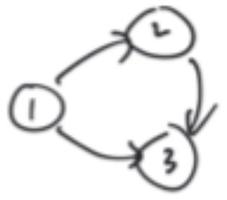
- visiting lecture today after class, Gates 114 (reception in 122)
- Project 5



explore every vertex reachable from a given vertex.

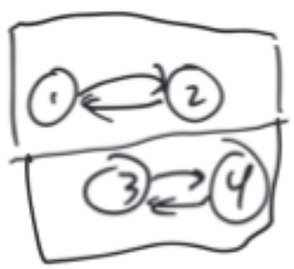
idea: make as much progress as possible (completely explore one path before backing out and exploring next)

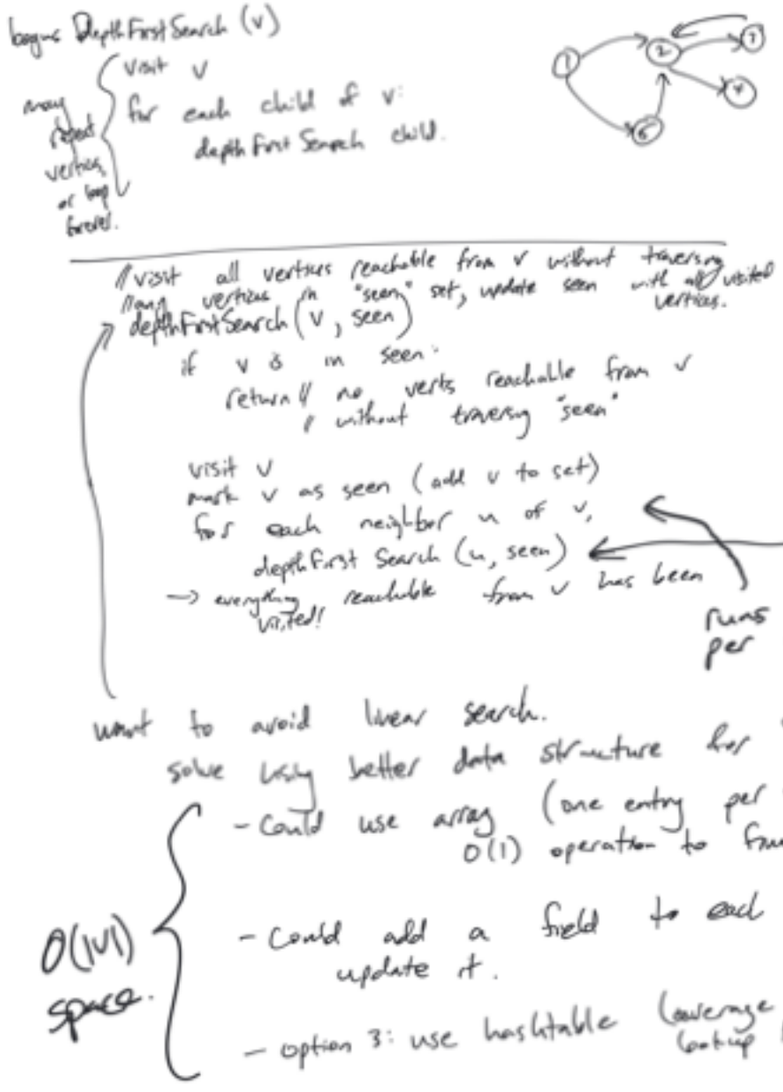
: Depth-first search.

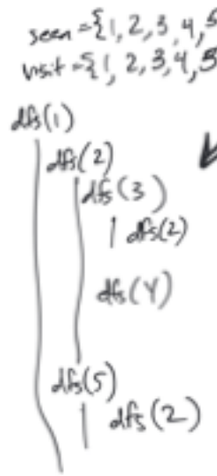


- find all vertices.
- detect cycles
- determine if a graph is connected: every vertex is reachable from every other vertex. } in an undirected graph

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







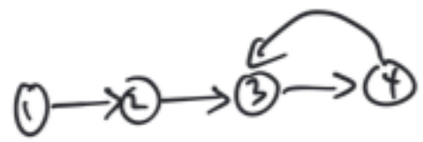
Size of call stack = # of verts on the path being considered.

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

run time:
- do some work for each vertex
- do some constant work for every edge

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, $|V|+1$



dfs(1)
dfs(2)
dfs(3)
dfs(4)
dfs(3) ← return immediately.

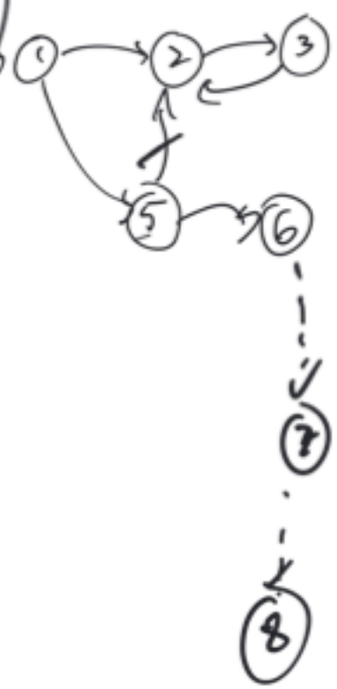
DFS iterative version.

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

use a stack!
- push, adds to end of stack
- pop, removes from end of stack.

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

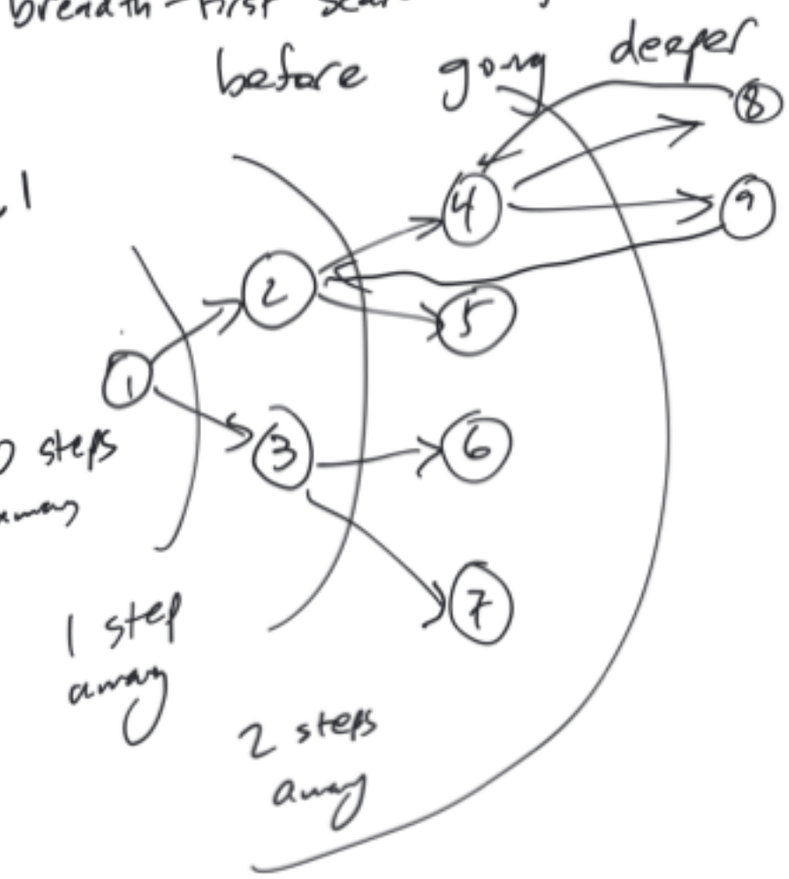
```
dfs(v):
  Collection<vertex> worklist;

  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.
```



worklist	visited
~~1~~	1
~~2~~	5
~~5~~	6
~~2~~	2
~~6~~	3
~~3~~	~~2~~
~~7~~	
~~8~~	

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

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



breadth first search:
- maintain a worklist of vertices to be visited
- repeatedly remove a vertex, (to get breadth first, want to remove vertices in order added.)
  - visit it,
  - add neighbors to WL. (only if we haven't seen vertex yet.)