CS1210 Lecture 35 Nov. 12, 2021

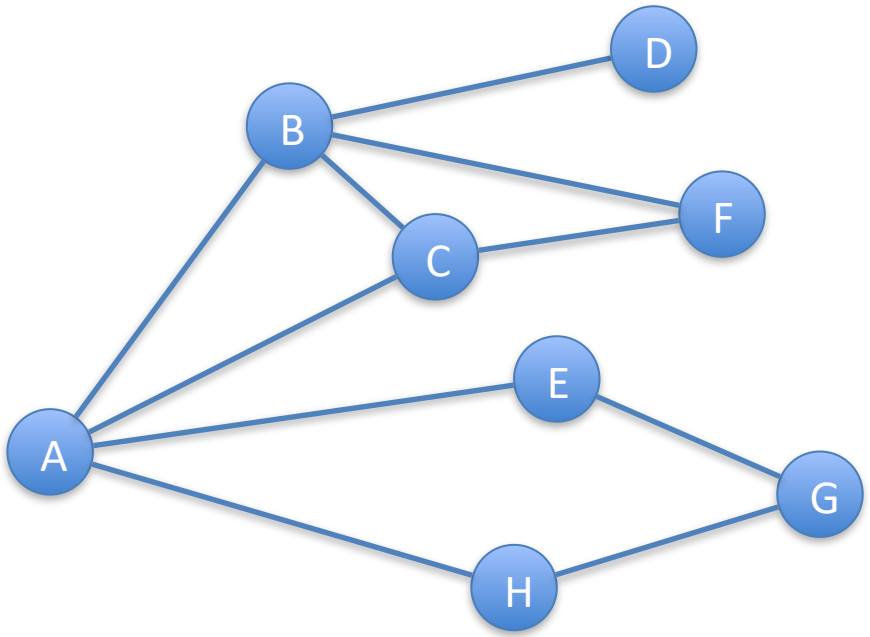
- HW 8 due Thursday
- Some extra slides and code called "DS9x" have been posted in the DS section of the course website. No points for doing it BUT
  - It will help a lot with the modify-bfs step of HW8. It walks you step by step through *exactly* what you need to do.
- Quiz 4 next Friday

Last time

- graph traversal: breadth first search
- the word ladder problem and HW 8

Today

- finish graphs -
  - HW8
    - modifying bfs
    - extractWordLadder
  - depth first search
- introduce GUIs (Ch 15)



KEY	VALUE
A	[B,C,E,H]
B	[A,C,D,F]
C	[A,B,F]
D	[B]
E	[A,G]
F	[B,C]
G	[E,H]
H	[A,G]

From last time: many real-world problems can be represented as problems involving graphs. The algorithms to solve those problems often involve **graph traversals**, organized exploration or "walkthroughs" of the graph.

Two famous ones are: depth-first search and breadth-first search. I will present breadth-first search.

You will not be responsible for knowing the details of breadth-first search (for exam purposes) but you need to understand it well enough to *use and extend* it in HW8.

Classic breadth-first search (bfs)

The goal of classic bfs is simply to travel to/explore, in an efficient and organized fashion, all nodes reachable from some given startNode. (The general goal of the other classic traversal, depth-first-search, is the same,. It just explores in a different order.)

First, we'll add a property, **status**, to nodes. Legal values will be 'unseen', 'seen', and 'processed'. The traversal process will use the values as it encounters nodes to keep from revisiting/re-processing already-processed nodes.

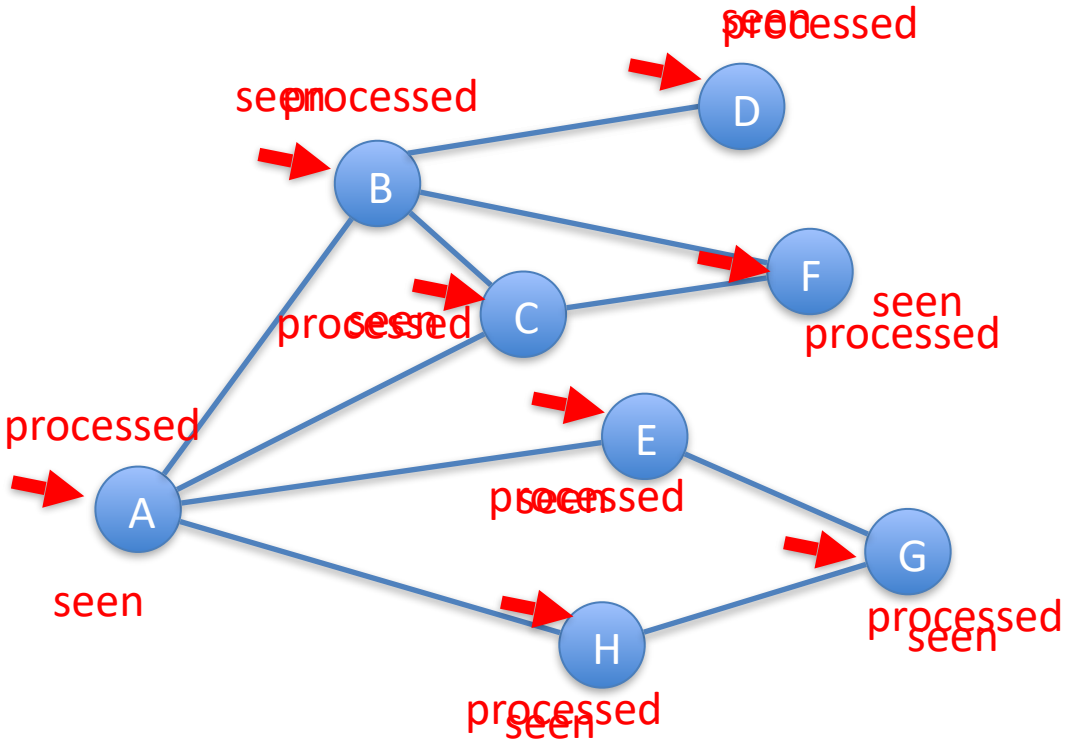
The algorithm in words:

1. Mark all nodes as 'unseen'
2. Initialize an empty queue (you implemented a queue class in DS6. "First-in first-out")
3. Mark the starting node as 'seen' and place it in the queue.
4. Remove the front node of the queue and call it the current node
5. Consider each neighbor of the current node. If its status is 'unseen', mark it as seen and put it on the queue.
6. Mark the current node 'processed' (*note: this step can be left out*).
7. If queue not empty go back to step 4.

This will explore all node reachable from the startNode in a breadth-first manner.

- Suppose startNode has neighbors n1 and n2. "Breadth first manner" means it travels start to n1 and then and then from start to n2 before exploring "beyond n1" – the process moves "broadly" out from the start a single step at a time.
- This enables a very simple modification of bfs to compute shortest (unweighted) distances from start to all other nodes in the graph

BFS starting at node A



Mark all nodes ‘unseen’
Mark A ‘seen’ and put A on queue Q
Until queue empty do:

- Remove the front node of the queue and call it the current node
- Consider each neighbor of the current node. If its status is ‘unseen’, mark as ‘seen’ and put it on the queue.
- Mark the current node ‘processed’

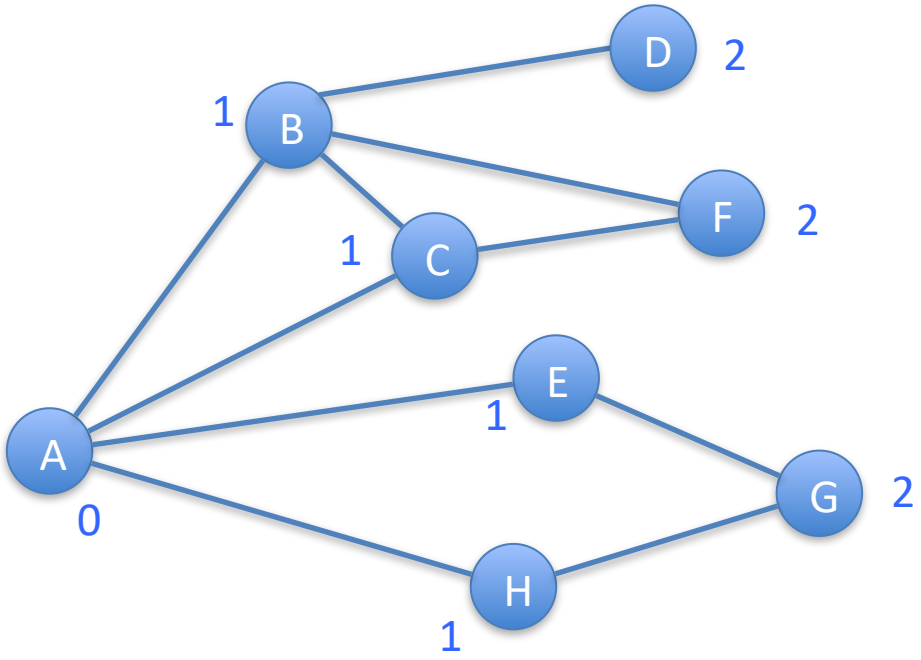
	Q: A
curr: A	Q:
	Q: B, C, E, H
curr: B	Q: C, E, H
	Q: C, E, H, D, F
curr: C	Q: E, H, D, F
	Q: E, H, D, F
curr: E	Q: H, D, F
	Q: H, D, F, G
curr: H	Q: D, F, G
	Q: D, F, G
curr: D	Q: F, G
	Q: F, G
curr: F	Q: G
	Q: G
curr: G	Q:
	Q: DONE

BFS

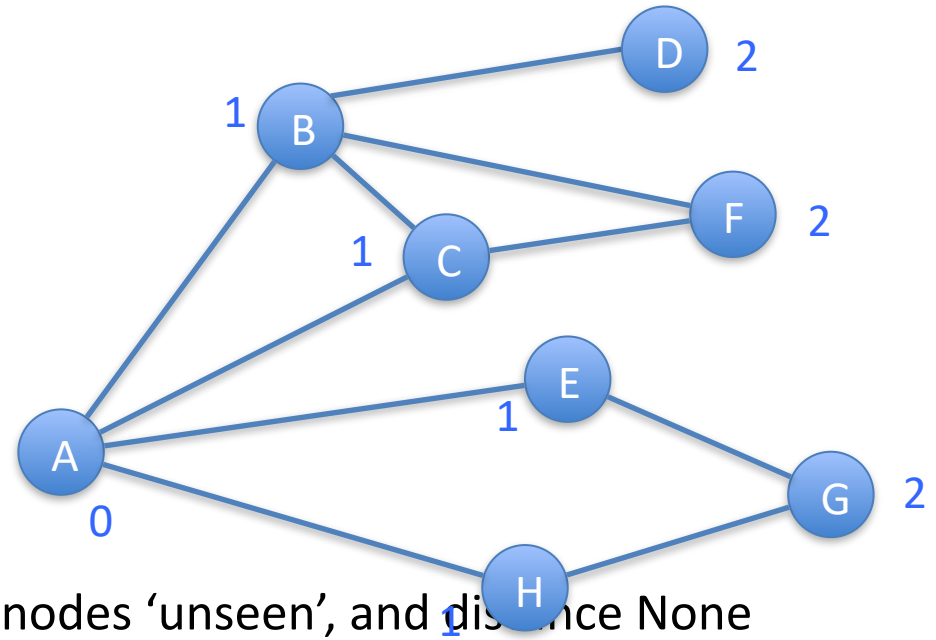
If we don't consider a node to be fully explored until its status value becomes "processed", then BFS explores the graph in "levels"

- first level 0 – items distance 0 from start
- then level 1 – items distance 1 from start
- then level 2
- Etc.
- This is very useful.

We can make a very slight change to bfs to record distance of each node from start!



BFS starting at node A



Mark all nodes 'unseen'

Mark A 'seen' and put A on queue Q

Until queue empty do:

- Remove the front node of the queue and call it the current node
- Consider each neighbor of the current node. If its status is 'unseen', mark as 'seen' and put it on the queue.
- Mark the current node 'processed'

Add a distance property to Node

Update node distance when first seen

Mark all nodes 'unseen', and distance None

Mark A 'seen', set A's distance to 0, and put A on queue Q

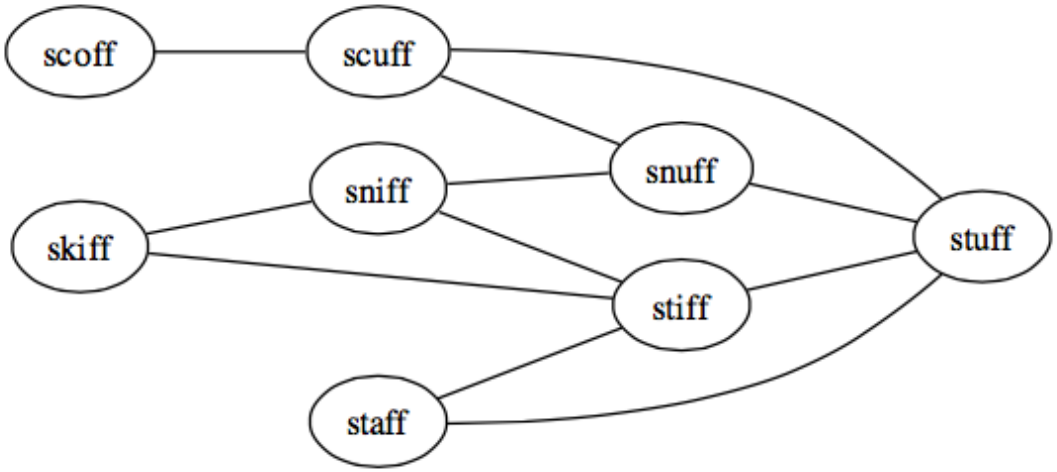
Until queue empty do:

- Remove the front node of the queue and call it the current node
- Consider each neighbor of the current node. If its status is 'unseen', mark as 'seen', set its distance to one more than current node's distance and put it on the queue.
- Mark the current node 'processed'

Breadth first search

- For unweighted graphs, bfs efficiently finds shortest path from start node to every other node!
  - "Three and a half degrees of separation" https://research.fb.com/three-and-a-half-degrees-of-separation/
  - Wikipedia game: given two topics (with Wikipedia pages), race to get from one to the other clicking only on Wikipedia page links. https://en.wikipedia.org/wiki/Wikipedia:Six_degrees_of_Wikipedia https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia
  - DS8x/HW8 word ladders! wikiGame.py demo
- Links
  - http://en.wikipedia.org/wiki/Breadth-first_search
  - http://interactivepython.org/courselib/static/pythonds/Graphs/ImplementingBreadthFirstSearch.html
  - animations:
    - https://www.cs.usfca.edu/~galles/visualization/BFS.html
    - https://visualgo.net/en/dfsbfs?slide=1
    - http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/search.html

wordsTest.txt



- Breadth first search from ‘scoff’:
  - order in which nodes are finished/processed:
    - dist 0 scoff
    - dist 1 scuff
    - dist 2 snuff, stuff
    - dist 3 sniff, stiff, staff
    - dist 4 skiff

DS9x: Slides and code posted in DS section of course website. Walks you precisely through modifying bfs to compute shortest distances. NO POINTS for completing this but shows you exactly how to do part of the modification of bfs.py needed for HW8

Given a start state and end state, we want to calculate the minimum number of states that must be entered (including the end state but not the start) to get from one to the other.

So, if start == Colorado and end == Montana, the answer is 2 (Colorado->Wyoming->Montana)
We will use a small modification of breadth-first-search to compute these distances.
Can you quickly see the min. for California -> Maine? Might be easier to let bfs do the work …

HW8 Word Ladder problem

Start from wordladderStart.py (see HW8 assignment) with stubs for all the functions you need.

Each part is pretty simple. Do them in order and do not go to next step until you've tested current step thoroughly!

1. Complete function "shouldHaveEdge" so that it correctly returns True when two length-5 words differ at exactly one character position. *THIS IS DS9 – USE IT!*
2. Complete function "buildWordGraph" to create and return a graph with one node for each word and an edge for each pair (w1, w2) of words where shouldHaveEdge(w1, w2) is True. *THIS IS DS9 – USE IT!*
3. Modify the Node class in basicgraph.py to include distance and parent properties and getDistance, setDistance, getParent, and setParent methods. *PART OF THIS IS DS9x – USE IT!*
4. Modify function bfs in bfs.py to correctly initialize the distance and parent properties and update them appropriately during the bread-first search. *PART OF THIS IS DS9x – USE IT!*
5. Complete function "extractWordLadder" to return a list of words representing a shortest path between the start and end words. See detailed comment on the provided "extractWordLadder" stub function.

(Steps 3 and 4) modifying Node class and bfs for wordLadder for HW8 to calculate distance and node ”parents” - DS9x shows you *exactly* how to do the distance part

0. add distance property and getDistance, setDistance methods.
1. add **parent property** and **getParent** and **setParent** methods to Node class

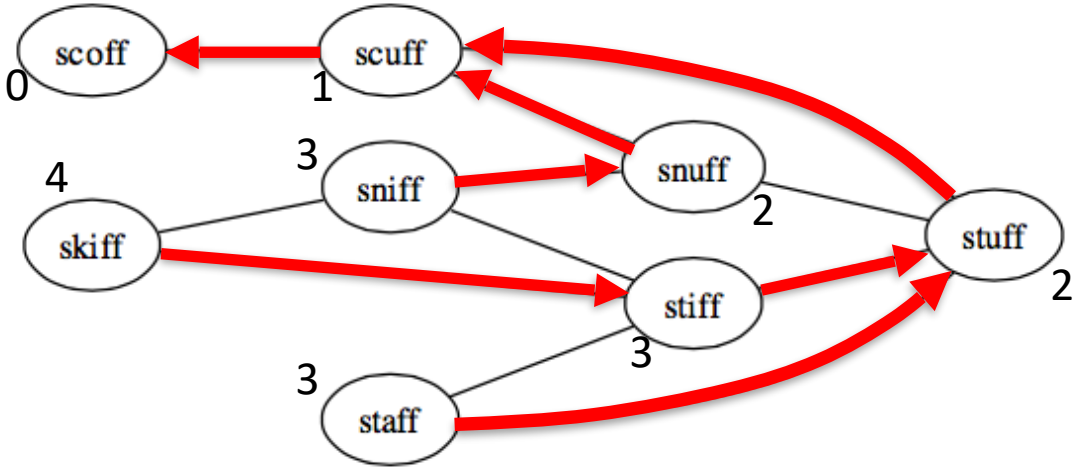
2. modify bfs:

Mark all nodes ‘unseen’
Set all nodes’ distances to None
Set all nodes’ **parents** to None
Mark start node ‘seen’ , give it distance 0, and put it on queue

Until queue empty do:

- Remove the front node of the queue and call it the current node
- Consider each neighbor of the current node. If its status is ‘unseen’, mark as ‘seen’, **set its parent to current node**, set its distance to 1 more than current node’s distance, and put it on the queue.
- Mark the current node ‘processed’

Parent property, BFS, and extractWordLadder (step 5)



- Breadth first search from 'scoff':
  - order in which nodes are finished/processed:
    - dist 0 scoff — set parent to None
    - dist 1 scuff — set parent to 'scoff' (Node for 'scoff')
    - dist 2 snuff, stuff — set parent of each to 'scuff'
    - dist 3 sniff (first seen from 'snuff') — set parent to 'snuff'
    - dist 3 stiff, staff (seen from 'stuff') — set parent of each to 'stuff'
    - dist 4 skiff — set parent to 'stiff'
- extractWordLadder(endNode)
  - can use a simple loop:
    - Initialize variable, e.g. currentNode, to endNode
    - Add currentNode to a result list
    - Update currentNode with currentNode's parent

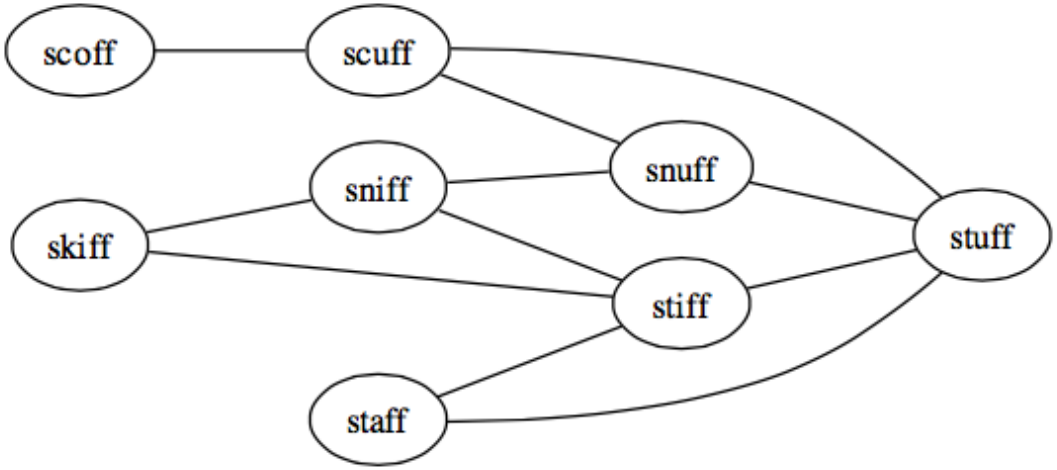
what is loop stopping condition?

HW8 – what happens when user enters one word instead of two

- If user provides just one word, a start word, what's the end word?
  - Code executes bfs.
  - All nodes reached will be marked with a distance from start word. Code chooses as end word any that is maximal distance
  - You then extract word ladder between your entered start word and that maximally distant word

Note: that is start-end ladder is a longest *shortest* ladder from start word. It is *not* (necessarily) the longest ladder from start word. Subtle but important difference

What if we wanted to find a longer path?



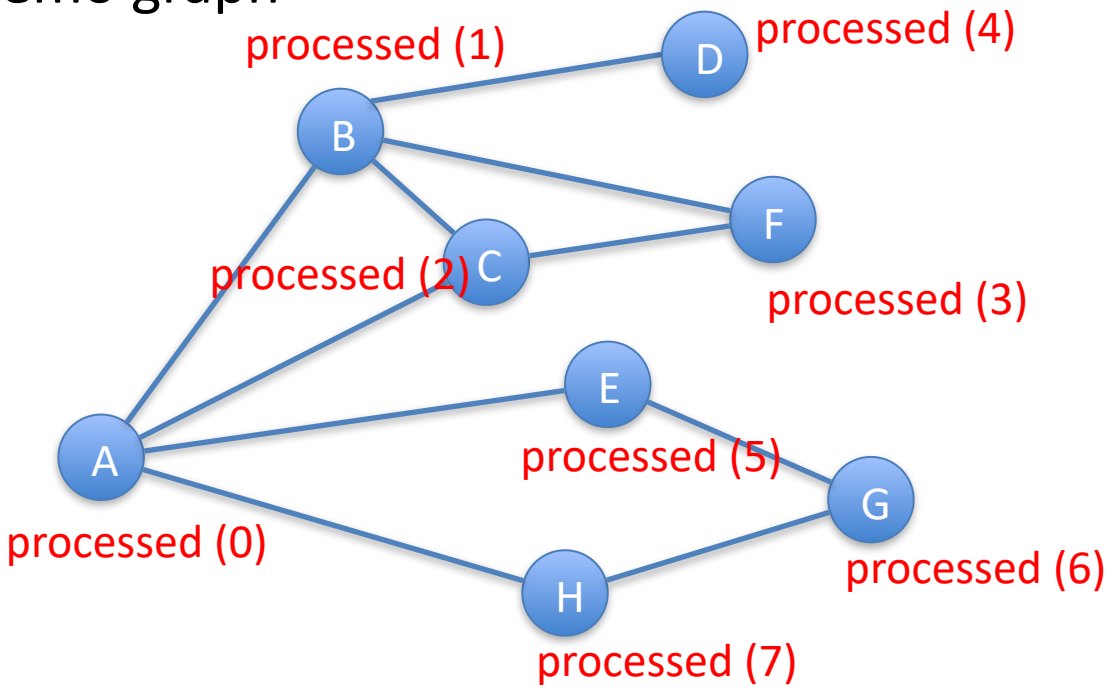
There is another classic graph traversal method called **depth first search** (https://en.wikipedia.org/wiki/Depth-first_search).

The basic idea of it is to explore deeply before exploring broadly. You will study the algorithm in later courses but it is quite concise:

```
DFS(node):
    mark node 'processed'
    for each 'unseen' neighbor of node:
        mark neighbor seen
        DFS(neighbor)
```

dfs.py

DFS starting at node A (on original demo graph



DFS(A)

DFS(B)

DFS(C)

DFS(F)

DFC(D)

DFS(E)

DFS(G)

DFS(H)

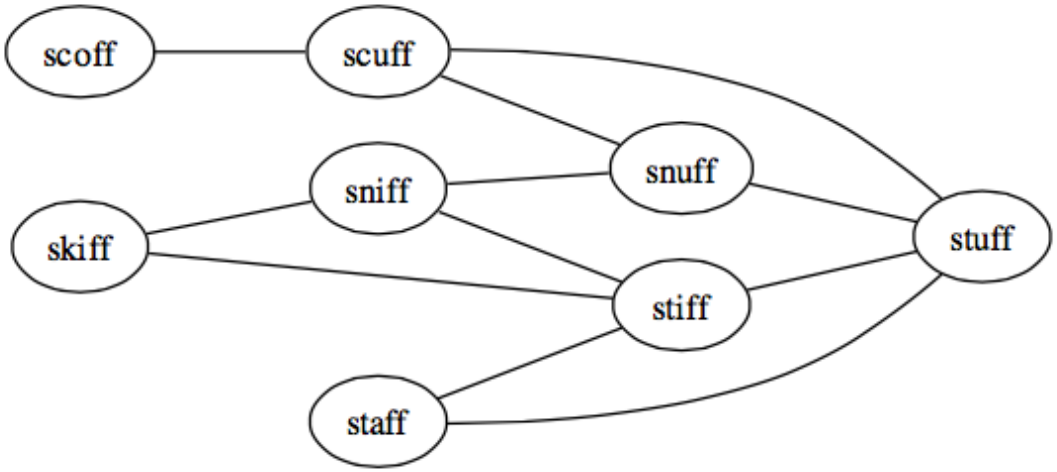
DONE

Mark all nodes ‘unseen’
Call DFS on desired start node

DFS(node):

- mark node ‘processed’
- for each ‘unseen’ neighbor of node:
  - mark neighbor seen
  - DFS(neighbor)

Note: number in parentheses corresponds to order in which nodes were marked processed



- Depth first search from ‘scoff’:
  - order in which nodes are finished/processed:
    - scoff, scuff, snuff, sniff, skiff, stiff, staff, stuff
- and wordLadder gives:
  - scoff -> scuff -> snuff -> sniff -> skiff -> stiff -> staff -> stuff for scoff-to-stuff ladder
- for black->white in full words5.text file?
  - DFS quickly finds ladder 192 long
  - Is that the longest possible? NO! There is one at least 648 long (I don’t know if there’s a longer one or not.)
  - Depth first search might find long paths, but not necessarily longest. In fact there is no known efficient general algorithm for finding longest path in a graph!

Next topic: Graphical User Interfaces in Python

- Chapter 15 of interactive textbook
- tkinter is a commonly used Python layer on top of a standard GUI toolkit TCL/Tk
- GUIs built using *widgets:* basic building blocks including buttons, labels, text, entry fields, scroll bars, etc.
- Steps:
  1. Define widgets and layout
  2. Specify how to handle “events” (button presses, mouse clicks, etc.)
  3. Start GUI/”event loop”

Monday and Wednesday

- Graphical user interfaces