CS2110 Lecture 34 Apr. 12, 2021

- HW 8 due Thursday
- An optional DS assignment, DS8x, has been posted
  - It will help with the modify-bfs step of HW8
  - If you complete it and submit it, you will can get up to 3 points making up for any DS point missed on earlier DS assignments.
- No class Wednesday - University "instructional break"

Last time

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

Today

- finish graphs
- introduce GUIs (Ch 15)

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

Word ladder puzzles

CAT	CAT	CAT
???	COT	COT
???	???	DOT
DOG	DOG	DOG

Find 3-letter English words for ??? Positions. Each must differ from previous and next word in only one location

This problem is easily representable and solvable using graphs! DS8/HW8

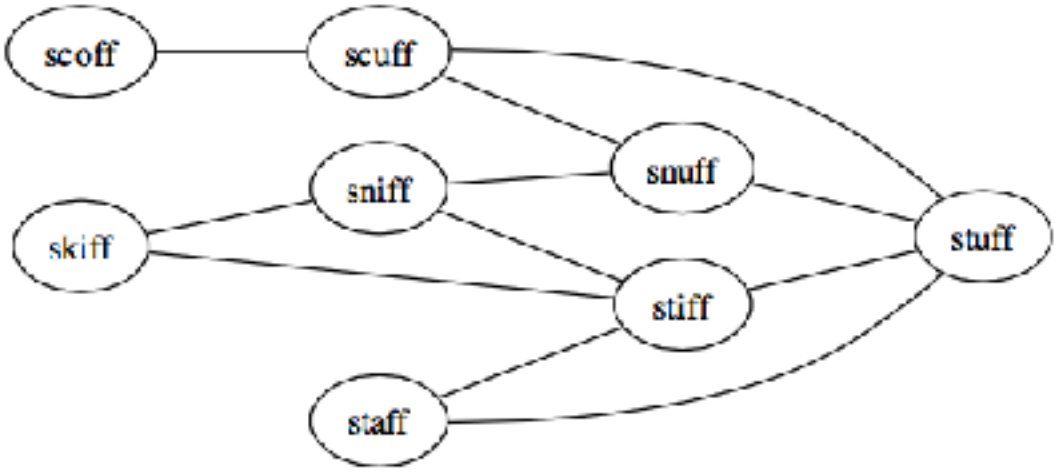
HW8 Word Ladder problem

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 DS8 – 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 DS8 – 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 DS8x – 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 DS8x – 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.

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

(Steps 3 and 4) modifying Node class and bfs for wordLadder for HW8 to calculate distance and node ”parents”

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

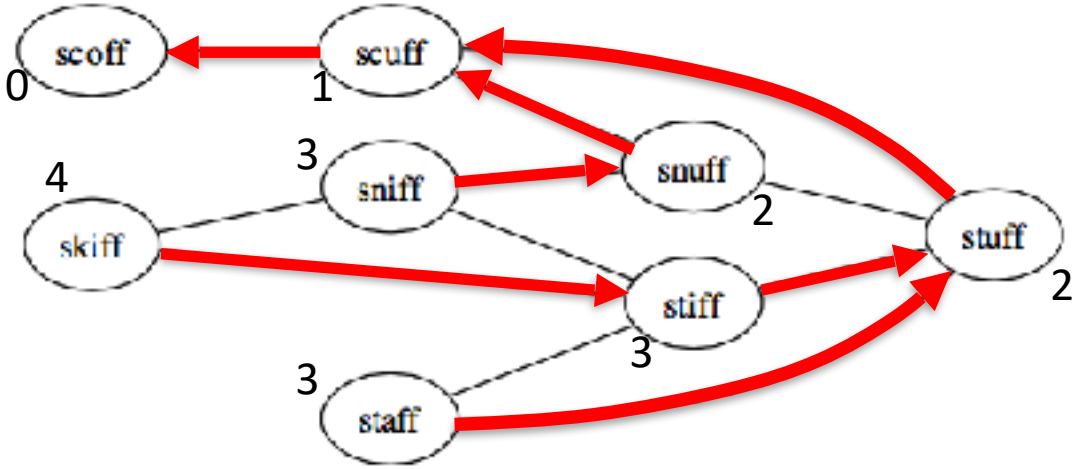
2. modify bfs:

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

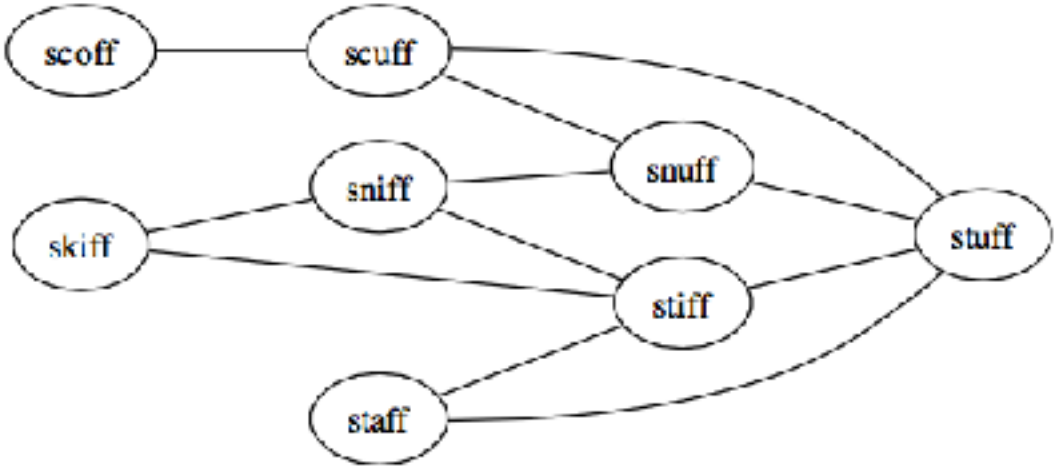
what is loop stopping condition?

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?



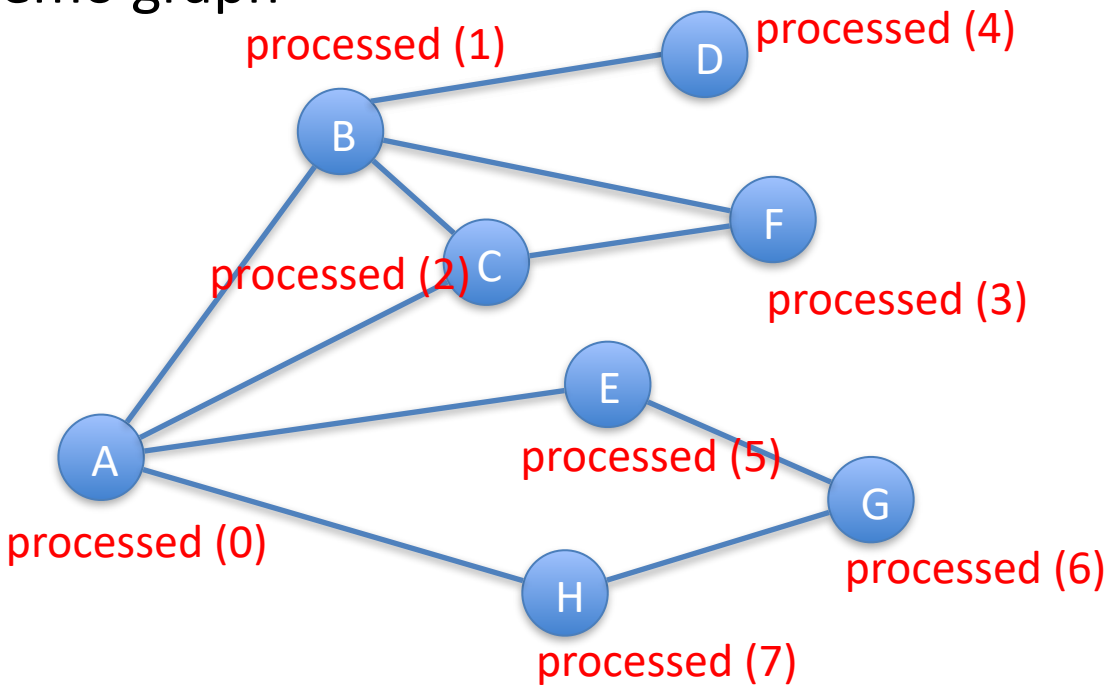
There is another classic graph traversal method called **depth first search** (https://en.wikipedia.org/wiki/Depth-first_search).

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)

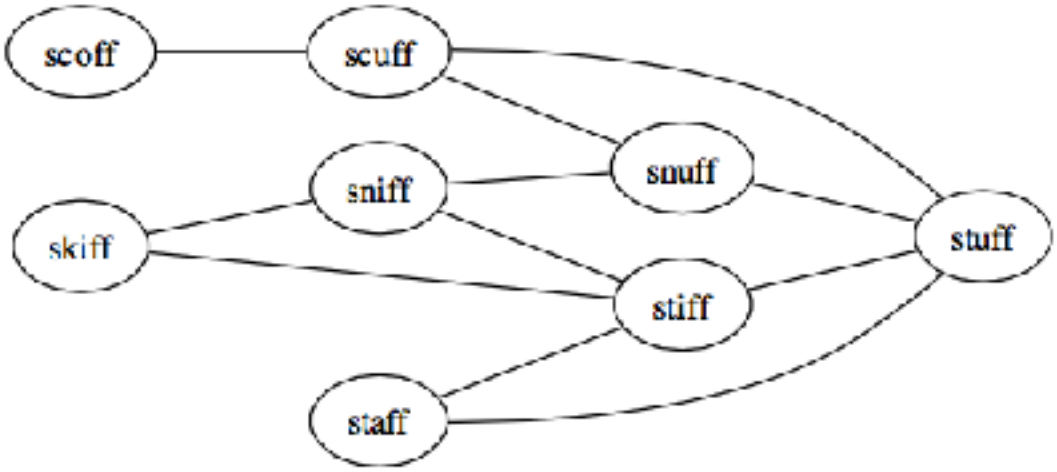
DONE

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!

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”

Getting started with tkinter

1. import tkinter
   - ***NOTE***: module was called Tkinter in Python 2. Much web documentation still calls it Tkinter. Functions, etc.. are the same but make sure to use 'tkinter' not 'Tkinter' when importing module, calling functions, etc.
2. Before using any other tkinter functions, you must call Tk() to create root/ main window and initialize Tkinter
   - e.g myMainWindow = Tk()
3. Next, define other GUI widgets – buttons, labels, entries, and their placement (via pack/grid function calls)
4. Specify how to handle event (e.g. button presses), typically by defining "callback functions" that are called by the GUI system when it detects events
5. Finally, when ready to start execution of the GUI, call mainloop()
   - e.g. myMainWindow.mainloop()

Note: mainloop() gives control of execution to Python. It won't return direct control to you until after you close/kill the main window. So, no commands/ function calls should follow it in your code. After you call it, you only get an opportunity to change things when *events* occur – i.e. when you click the mouse on buttons, type text in entries, etc.

- tkinter is big – there are long chapters and even whole books about it. You will use Tkinter in HW9 but not a lot of the features.
- You just need to understand the basics of a few widget types (Label, Entry, Button, Frame) and how to respond to events like button presses.

Links/resources for learning tkinter

- Chapter 15 of the interactive text has a LOT of info – more extensive that most of the other chapters.
- Tkinter info on the official Python site:
  https://docs.python.org/3.7/library/tkinter.html
- This tutorial - http://www.tutorialspoint.com/python/python_gui_programming.htm - does a good job of explaining and demonstrating the basics and includes several good small examples. It seems like a good place to start.
- If you want a "real textbook chapter", Chapter 6 of Kent Lee's book "Python Programming Fundamentals" is pretty good. The book is electronically available to UI students through the UI library (you need to be on the campus network to access it). http://link.springer.com/chapter/10.1007%2F978-1-84996-537-8_6
- This site can also very helpful. It's usually the first hit when I do Google search (always very useful to me, though I'm not certain all info is up-to-date): http://effbot.org/tkinterbook/tkinter-index.htm
- Some of the explanations here can be helpful (e.g. explains "pack" better than many others): http://thinkingtkinter.sourceforge.net

Friday (not Wed!)

- More GUI / tkinter