CS1 Lecture 35 Apr. 14, 2017

- HW6 scores have been posted
- HW 8 due Wed.
  - Q2 based on today's material (will be posted by this afternoon)
- Exam 2: April 20, 6:30-8:00pm
- Main topics: dictionaries (Ch 11), classes and objects (Ch 15-18), algorithm analysis (Ch 21), sorting and searching, graphs randomization. Details on Mon. and Wed.

An interesting video

https://strangeparts.com/how-i-made-my-own-iphone-in-china/

Last time

- Reviewed graph representation and bfs traversal
- Discuss HW8 word ladder problem
- Short depth-first search overview

Today

- Some problems involving randomization and simulation

HW8 Q1 Word Ladder problem

Several parts, none complicated. (AGAIN: Implement and test parts – don't just write all the code at once and hope it works!)

1. Read word list and create graph
   - One node per word
   - Edge between each pair of words differing by one letter
2. Execute breadth-first search starting from start word's node.
3. "extract" ladder from graph (after bfs) by following a parent path (you will add 'parent' property to Node class and set during bfs) from end word's node back to start node
4. Interactive loop requesting input from user: request two words, call findLadder (which should return a representation of the ladder), print result nicely

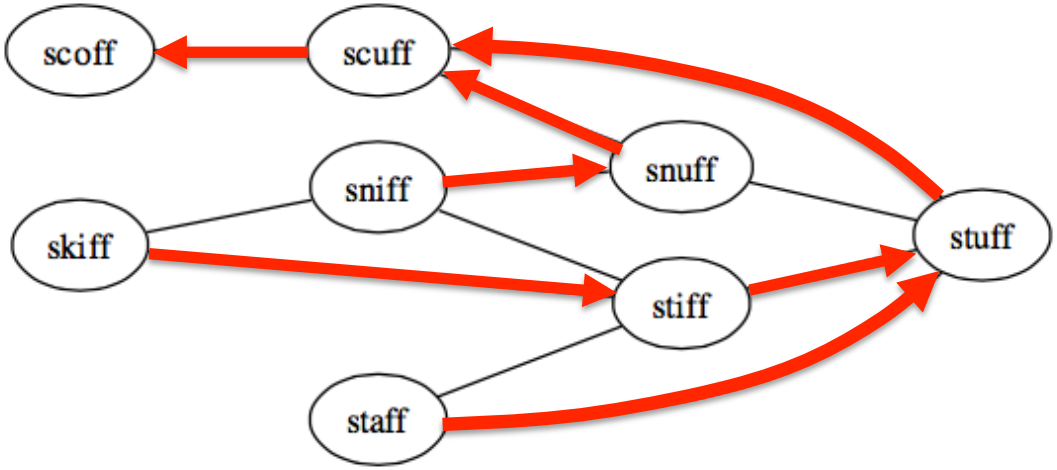
wordladderStart.py (see HW8 assignment) has stubs for all the functions you need.

NOTES:

- it's okay to use simple O(n^2) process to create the graph
  - If you are interested, there is a faster way: http://interactivepython.org/courselib/static/pythonds/Graphs/BuildingtheWordLadderGraph.html
- Create a tiny word file to test on. Choose the words, draw the graph by hand, and check program's results. See example on next slide.
- findWordLadder should do both:
  1. bfs – modified to keep track of 'parent'. When a neighbor node is marked 'seen', set parent to current node. This is similar to what you do in disc. section this week!
  2. ladder can be constructed/printed by following parent path from end word to start word

  (NOTE: with interactive loop repeatedly asking for two new words, are there cases where step1, the bfs, of findWordLadder is not necessary? Don't worry about this – it's no problem to always do the bfs. It's fast!)

Parent property, BFS, and extractWordLadder



- 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 parant to 'snuff'
    - dist 3 stiff, staff (seen from 'stuff') — set parent of each to 'stuff'
    - dist 4 skiff — set parent to 'stiff'
- extractWordLadder(startNode, 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?

Again - modifying Node class and bfs for wordLadder

1. **add parent property** and **getParent** and **setParent** methods to Node class

2. modify bfs:

Mark all nodes 'unseen'
Set all nodes' parents to None
Mark start node 'seen' and put it on queue

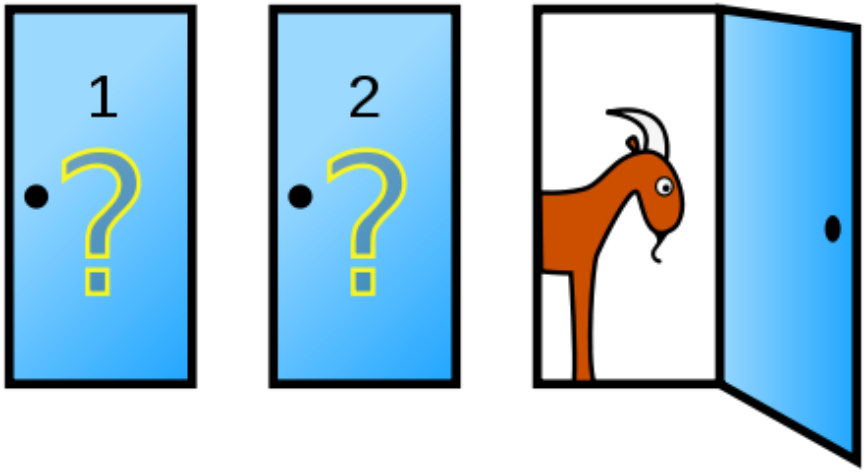
Until queue empty do:

- 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**, and put it on the queue.
- Mark the current node 'processed'

This is very similar to what you did with distance in disc. section

Monty Hall problem

http://en.wikipedia.org/wiki/Monty_Hall_problem



Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

- Parade magazine, 1990, Marilyn vos Savant's "Ask Marilyn column"

Monty Hall problem

- M. vos Savant (whose column generated huge public awareness) quotes psychologist M. Piattelli-Palmarini: "... no other statistical puzzle comes so close to fooling all the people all the time." Herbranson and Schroeder: Pigeons repeatedly exposed to the problem show that they rapidly learn always to do the right thing, unlike humans. ☺
- Demonstrate simulation using randomization to help "see" that switching is the right thing to do.
  montyhall.py

A little Monte Carlo simulation

Roughly speaking, Monte Carlo simulation is a fancy name for using repeated random sampling of a problem space to determine results

lec35coins.py is a very simple Monte Carlo simulation of coin flipping.

- doNCoinFlips(numFlips): return number of heads and number of tails resulting from specified number of flips
- doCoinFlipTrials(numTrials, numCoins): doing *one* set of flips is not usually a good way to do an experiment. Better to do several "trials" flipping the same number of coins. E.g flip 100 coins 10 times

  Compute, print, and return statistical info:
  - average head/tail ratio (averaged over the set of trials)
  - std deviation of head/tail ratios

- doCoinFlipExperiment(minCoins, maxCoins, factor)
  - collect doCoinFlipTrials data for different numbers of coins, so we can see trend of statistics as number of flips grows:
    - minNum, minNum*factor, minNum*factor*factor, …
- plotResults
  - use pylab to create graphs of stats gathered by doCoinFlipExperiment

HW8 Q2

Suppose we flip 15 coins and the result is

H, T, T, H, H, H, T, H, H, T, H, T, T, T, T

The length of the longest consecutive "run" of heads is 3.

If you flip 100 coins, what do expect will be the length of the longest consecutive string of heads?

How can we figure this out?

Flip 100 coins once? No reason to believe result is close to expected value if we do it again

Better: flip 100 coins several times and take average.

How about 200 coins? 1000 coins?

As number of coins grows, how do expect long run length grow?

HW8 Q2: modify lec35coins.py

to try to answer this problem

Using randomization to mix up (shuffle) a list of numbers

```
def mixup(L):
    newL = L[:]
    length = len(L)
    for i in range(length):
        newIndex = random.randint(0,  length-1)
        newL[newIndex], newL[i] = newL[i], newL[newIndex]
    return(newL)
```

- What do you think?
- Test on a few lists. mixupB.py

```
>>> mixup([1,2,3,4,5])
[1, 3, 4, 5, 2]
```

- Run testMixup(100000). What do you expect as result?
- Hmm…
- See: http://blog.codinghorror.com/the-danger-of-naivete/ and https://www.cigital.com/papers/download/developer_gambling.php

An problem to think about for fun

You want to sell your phone.

A large line of people assembles seeking to buy it. Each has a random price offer (bid) in mind. You want to maximize price BUT you must consider the bids ONE AT A TIME, in the order received, and REJECT OR ACCEPT EACH ONE IMMEDIATELY.

Is there are strategy that can make it likely you get a good price?

E.g. “always take first bid” – chance of getting best price is then 1/n. Not so good….

Next time

One or two more randomization examples. E.g. calculating pi via randomization: https://learntofish.wordpress.com/2010/10/13/calculating-pi-with-the-monte-carlo-method/

Introduction to graphical user interface (GUI) programming in Python