CSC 321: Data Structures

Fall 2013

See online syllabus (also available through BlueLine2):
http://dave-reed.com/csc321

Course goals:

- To understand fundamental data structures (lists, stacks, queues, sets, maps, and linked structures) and be able to implement software solutions to problems using these data structures.
- To achieve a working knowledge of various mathematical structures essential for the field of computer science, including graphs, trees, and networks.
- To develop analytical techniques for evaluating the efficiency of data structures and programs, including counting, asymptotics, and recurrence relations.
- To be able to design and implement a program to model a real-world system, selecting and implementing appropriate data structures.

221 vs. 222 vs. 321

221: intro to programming via scripting

- focused on the design & analysis of small scripts (in Python)
- introduced fundamental programming concepts
  - ✓ variables, assignments, expressions, I/O
  - ✓ control structures (if, if-else, while, for), lists
  - ✓ functions, parameters, intro to OO

222: object-oriented programming

- focused on the design & analysis of more complex programs (in Java)
- utilized OO approach & techniques for code reuse
  - ✓ classes, fields, methods, objects
  - ✓ interfaces, inheritance, polymorphism, object composition
  - ✓ searching & sorting, Big-Oh efficiency, recursion, GUIs

you should be familiar with these concepts (we will do some review next week, but you should review your own notes & text)

321: data-driven programming & analysis

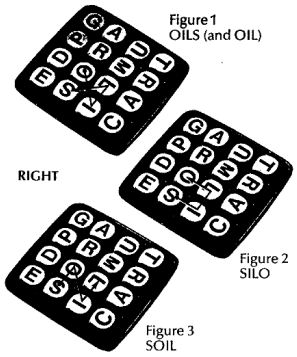
- focus on problems that involve storing & manipulating large amounts of data
- focus on understanding/analyzing/selecting appropriate structures for problems
  - ✓ standard collections (lists, stacks, queues, trees, sets, maps)
  - ✓ mathematical structures (trees, graphs, networks)
  - ✓ analysis techniques (counting, asymptotics, recurrence relations)

When problems start to get complex…

…choosing the right algorithm and data structures are important

- e.g., phone book lookup, checkerboard puzzle

- must develop problem-solving approaches (e.g., brute force, backtracking)
- be able to identify appropriate data structures (e.g., lists, trees, sets, maps)



EXAMPLE: suppose you want to write a program for playing Boggle (Parker Bros.)

- need to be able to represent the board
- need to be able to store and access the dictionary
- need to allow user to enter words
- need to verify user words for scoring
- perhaps show user words they missed

3

Possible implementations

1. for each user word entered, search the Boggle board for that word
   - But how do you list all remaining words at the end?

2. build a list of all dictionary words on the Boggle board by:
   - searching for each word in the dictionary, add to list if on the board.

   For each user word entered, search the list to see if stored (and mark as used).

   At the end, display all words in the list not marked as used.

3. build a list of all dictionary words on the Boggle board by:
   - exhaustively searching the board, checking letter sequences to see if in the dictionary.

   For each user word entered, search the list to see if stored (and mark as used).

   At the end, display all words in the list not marked as used.

4

Another example: anagram finder

you are given a large dictionary of 117,663 words

repeatedly given a word, must find all anagrams of that word

- pale → leap pale peal plea
- steal→ least setal slate stale steal stela taels tales teals tesla
- banana → banana

there are many choices to be made & many "reasonable" decisions

- how do you determine if two words are anagrams?
- should you store the dictionary words internally? if so, how?
- should you preprocess the words? if so, how?
- is a simplistic approach going to be efficient enough to handle 117K words?
- how do you test your solution?

5

Possible implementations

1. generate every permutation of the letters, check to see if a word
   - how many permutations are there?
   - will this scale?

2. for each word, compare against every other word to see if an anagram
   - how costly to determine if two words are anagrams?
   - how many comparisons will be needed?
   - will this scale?

3. preprocess all words in the dictionary and index by their sorted form
   - e.g., store "least" and "steal" together, indexed by "aelst"
   - how much work is required to preprocess the entire dictionary?
   - how much easier is the task now?

6

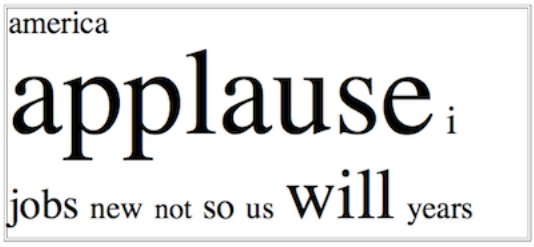
HW1: tag clouds

HW1 is posted

- due in 2+ weeks, so lots of time
- designed to reengage you with CSC222 material
- you will works in two-person teams (assigned by instructor)
- work together (and with me) to locate and fill holes in your knowledge/skills

you will design & implement two programs that generate tag clouds

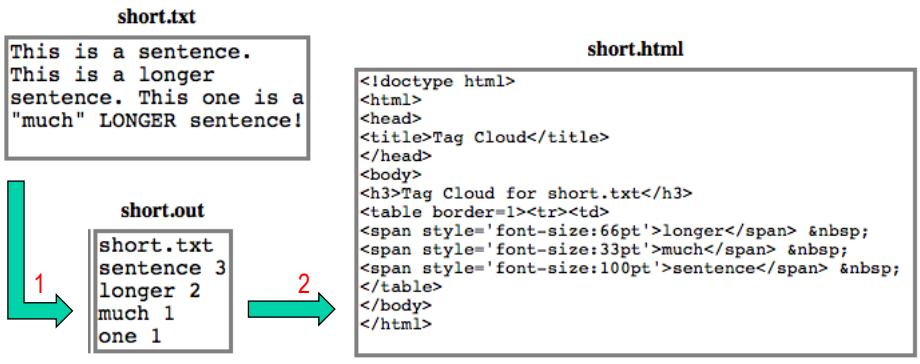
e.g., tag cloud of 20 most frequent words in 2013 State of the Union Speech



7

HW1: in 2 parts

1. WordFrequencies.java
   - takes a text file and generates a stats file, with words & frequencies
2. TagCloud.java
   - takes a stats file & a number, and generates a Web page with tag cloud



8