CSC 321: Data Structures

Fall 2018

See online syllabus (also available through BlueLine):

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

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, Sudoku solver, path finder

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

example: dictionary lookup

- you are given a large dictionary of 100K+ words
- want to be able to store and lookup words

    1. store in an unsorted ArrayList, perform sequential search
    2. insert into a sorted ArrayList, perform binary search
    3. store in an unsorted ArrayList, sort before each sequence of binary searches
    4. store in a LinkedList or TreeSet (?) or HashSet (?)

- the efficiency of each approach depends not only on the size of the dictionary, but the pattern of operations
    - ✓ sequence of adds followed by sequence of searches?
    - ✓ mixture of adds and searches?

Another example: anagram finder

you are given a large dictionary of 100K+ 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 100K+ words?
  - ✓ how do you test your solution?

Possible implementations

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

2. compare against each word in the dictionary and test 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?

HW1: credit card numbers

HW1 is posted

- part1 is to be completed in 2-person teams, due in 1.5 weeks
  we will meet to go over the code, go over holes in your knowledge/skills
- part2 is to be completed individually, builds on part1 code

both parts involve verifying credit card numbers

- Visa, Mastercard & Discover use 16-digits (6 for issuer, 9 for user account, 1 check)
- American Express uses 15-digits (6 for issuer, 8 for user account, 1 check)
- as a security measure, the numbers must conform to the Luhn Formula

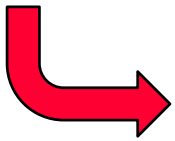
```
4289 0298 1524 0026

4 2 8 9 0 2 9 8 1 5 2 4 0 0 2 6
8  16   0  18   2   4   0   4
8+2+7+9+0+2+9+8+2+5+4+4+0+0+4+6 = 70
```

HW1 part 1: working in pairs

1. read in digit sequences from a file
   - file name should be specified by user
   - one sequence per line (varying lengths), ignore spaces
2. for each, determine and display if valid or invalid

```
4289 0298 7524 0023
4289 0298 7524 0026
313 4890 444 2000 120
42 89 01 44 32 58 99 4
42 89 01 44 32 58 99 40
4289 0144 3258 9941
1234-5678-9876-5432
```



```
4289 0298 7524 0023 VALID
4289 0298 7524 0026 INVALID
313 4890 444 2000 120 VALID
42 89 01 44 32 58 99 4 VALID
42 89 01 44 32 58 99 40 INVALID
4289 0144 3258 9941 INVALID
1234-5678-9876-5432 INVALID
```

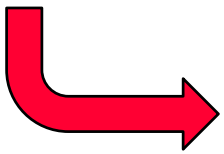
ADVICE: work with your partner – you both should understand everything in your program

be introspective – identify holes, help each other, come see me!

HW1 part 2: working individually

1. display valid/invalid sequences in groups
   within each group, display in numerical order (ignoring spaces)
2. in the case of a corrupted digit, determine the missing digit
   a corrupted digit specified as '?'

```
4289 0298 7524 0023
4289 0298 7524 0026
313 4890 444 2000 120
42 89 01 44 32 58 99 4
42 89 01 44 32 58 99 40
4289 0144 3258 9941
1234-5678-9876-5432
4289 0298 7524 002?
4289 0298 ?524 0026
4289 0298 ?524 002?
```



```
VALID
313 4890 444 2000 120
42 89 01 44 32 58 99 4
4289 0298 1524 0026
4289 0298 7524 0023
4289 0298 7524 0023

INVALID
1234-5678-9876-5432
42 89 01 44 32 58 99 40
4289 0144 3258 9941
4289 0298 7524 0026
4289 0298 ?524 002?
```

NOTE: this part must be completed individually, building upon your team's code

come see me A LOT!