CSC 321: Data Structures

Fall 2017

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?

3

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?

4

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?

5

HW1: robot navigation

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 a grid with rectangular obstacles

- the grid is specified by a file that contains # rows & # cols on first line
- subsequent lines contain row/col of top-left and bottom-right corners of an obstacle

```
8 12
2 2 3 4
5 5 5 5
7 7 8 12
1 10 4 10
```

6

HW1 part 1: represent and access the grid

1. read (and store) the grid info
2. display the grid, with a frame and * for obstacles
3. repeatedly prompt for and determine if positions are in obstacles

```
Please enter the grid file name: robo1.txt

+------------+
|          * |
|  ***     * |
|  ***     * |
|          * |
|    *       |
|            |
|      ******|
|      ******|
+------------+

Enter a row and column to check (zeros to exit): 10 8
    (10, 8) IS OUTSIDE THE GRID
Enter a row and column to check (zeros to exit): 3 3
    (3, 3) IS WITHIN AN OBSTACLE
Enter a row and column to check (zeros to exit): 4 3
    (4, 3) IS VACANT IN THE GRID
Enter a row and column to check (zeros to exit): 0 0
    DONE
```

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

be introspective – identify holes, work with your partner, come see me!

7

HW1 part 2: navigate the robot

1. display row & column labels (last digit only)
2. repeatedly process robot movements given:
   - initial row & column
   - sequence of moves using >, <, ^ and v

```
Please enter the grid file name: robo1.txt

  123456789012
 +------------+
1|          * |
2|  ***     * |
3|  ***     * |
4|          * |
5|    *       |
6|            |
7|      ******|
8|      ******|
 +------------+

Enter the start row and col (zeros to end) and moves: 1 3 >>>vv<v
    ROBOT ENDS UP AT (4,5)
Enter the start row and col (zeros to end) and moves: 7 6 >v>^^
    START POSITION IS WITHIN AN OBSTACLE
Enter the start row and col (zeros to end) and moves: 6 6 >v>^^
    ROBOT CRASHES INTO AN OBSTACLE AT (7,7)
Enter the start row and col (zeros to end) and moves: 6 6 >^^<vv
    ROBOT ENDS UP AT (6,6)
Enter the start row and col (zeros to end) and moves: 6 2 <<<<
    ROBOT LEAVES THE GRID AT (6,0)
Enter the start row and col (zeros to end) and moves: 0 0
    DONE
```

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

come see me A LOT!

8