CS151 Intro to Data Structures

Final Exam Review

Announcements

HW7 Due May 9th

Extra office hours this week - vote on piazza

Lab today is extra credit on the final exam

Exam Format

- Cumulative but heavily focused on second half of content
- Tested on knowledge of DS (how they work and their pros and cons), programming skills, and problem solving
- 180min
- 2 8.5/11in cheat sheets allowed (front and back)
- Format: 125 total points
  - 5 points T/F questions
  - 10 points reading and understanding code
  - 33 points programming
  - 77 points short answer

Topics

Data Structures

- Arrays
- Expandable Arrays
- Stacks
- Queues
- Linked Lists
- Binary Trees
- **Binary Search Trees**
- **Heaps**
- **Hash Tables**
- **AVL Trees**
- **Splay Trees**
- **Graphs**

Other concepts:

- Generics
- Iterators
- **Big-O analysis**
- OOP & Inheritance
- Interfaces
- **Sorting**
    - **Selection Sort**
    - **Heap Sort**
    - **Merge Sort**
    - **Quick Sort**

Data Structures

Expandable Arrays

search

- How do we implement?
- Best case?
- Worst case?

insertion

- How do we implement?
- Best case?
- Worst case?

removal

- How do we implement?
- Best Case?
- Worst Case?

LinkedList

search

- How do we implement?
- Best case?
- Worst case?

insertion

- How do we implement?
- Best case?
- Worst case?

removal

- How do we implement?
- Best Case?
- Worst Case?

Binary Trees

Search?

- How do we implement?
- Best Case?
- Worst Case?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Balanced Binary Search Tree (AVL)

Search?

- How do we implement?
- Best Case?
- Worst Case?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Stacks - LinkedList implementation

Search?

- How do we implement?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Stacks - Array implementation

Search?

- How do we implement?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Queues - LinkedList implementation

Search?

- How do we implement?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Queues - Array implementation

Search?

- How do we implement?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Heaps

Search?

- How do we implement?
- Best Case?
- Worst Case?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Poll?

- How do we implement?
- Best Case?
- Worst Case?

Hash Tables (with probing)

Search?

- How do we implement?
- Best Case?
- Worst Case?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Hash Tables (with chaining)

Search?

- How do we implement?
- Best Case?
- Worst Case?

Insertion?

- How do we implement?
- Best Case?
- Worst Case?

Removal?

- How do we implement?
- Best Case?
- Worst Case?

Graphs - Adjacency List

add vertex?

- How do we implement?
- Best Case?
- Worst Case?

remove vertex?

- How do we implement?
- Best Case?
- Worst Case?

add edge?

- How do we implement?
- Best Case?
- Worst Case?

remove edge?

- How do we implement?
- Best Case?
- Worst Case?

Graphs - Adjacency Matrix

add vertex?

- How do we implement?
- Best Case?
- Worst Case?

remove vertex?

- How do we implement?
- Best Case?
- Worst Case?

add edge?

- How do we implement?
- Best Case?
- Worst Case?

remove edge?

- How do we implement?
- Best Case?
- Worst Case?

AVL & Splay Trees

How many rotations required for insertion?

- AVL Tree:
  - 1 (a single or double rotation)
- Splay Tree:
  - n

Perform the following operations

For the following data structures:

1. bst
2. avl tree
3. splay tree
4. min heap

insert: 42, 17, 89, 5, 63, 28, 10, 15, 77, 33, 50

remove: 10, 17

what was the runtime complexity?

Perform the following operations

For the following hash tables of size 7 with h(x) = x % 7

1. Linear probing
2. Quadratic probing
3. Double probing (h(x) + f(i * h2(x))
   a. with h2(x) = 11 - (x % 11)

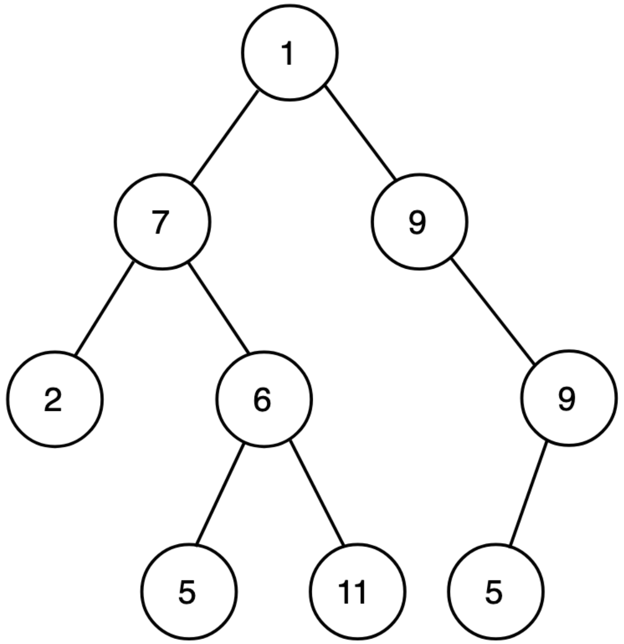
insert: 42, 17, 89, 5, 63, 28, 77

remove: 5, 17

what was the runtime complexity?

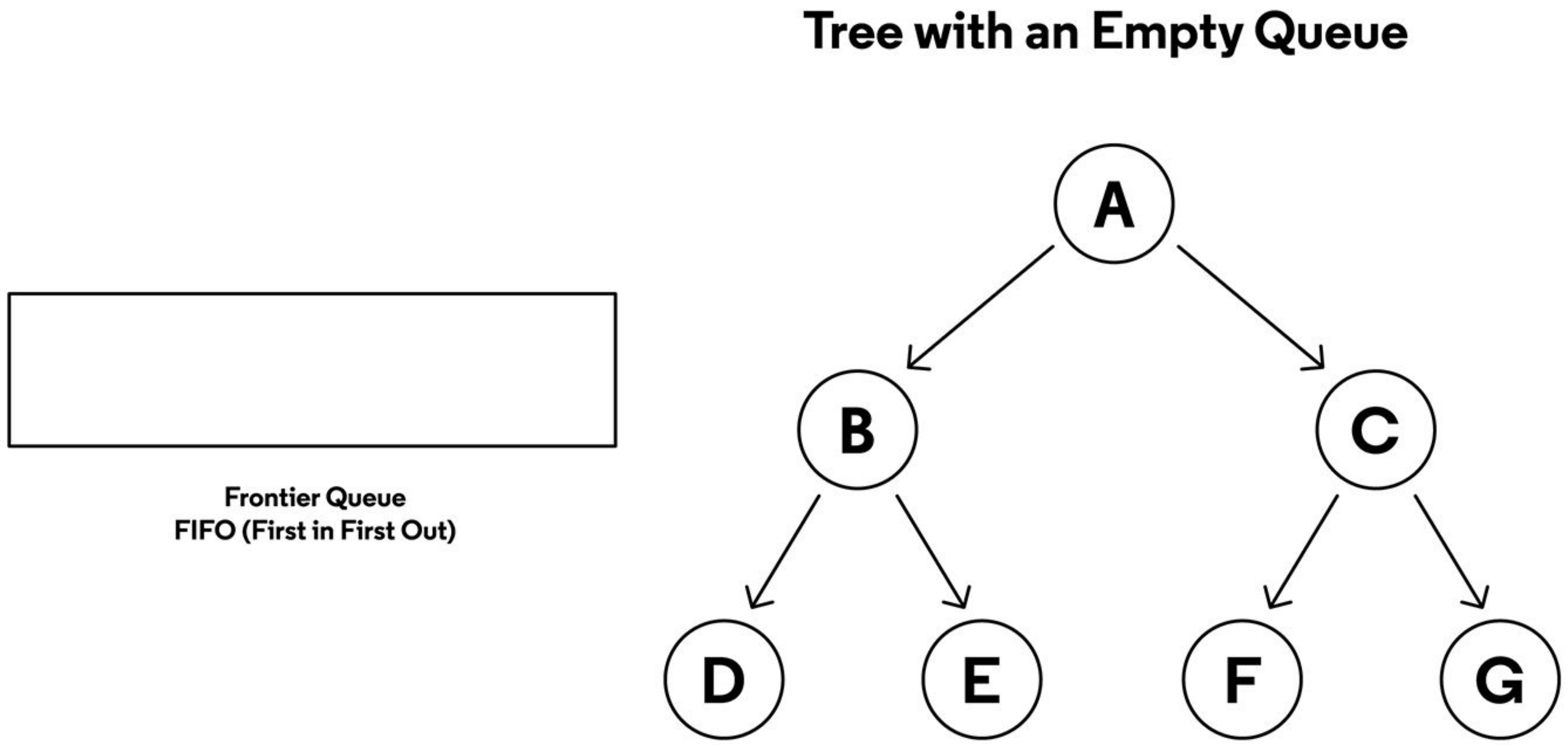
Breadth-First Traversal

what is the breadth first traversal output of this tree?



1 7 9 2 6 9 5 11 5

Breadth First Search (BFS)



https://www.codecademy.com/article/tree-traversal

Breadth-First Traversal

Let’s code it

Other Concepts

Runtime Complexity

Sort these from fastest to slowest:

- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(1)$
- $O(2^n)$

Sorting

Sort [**5, 18, 42, 67, 29, 10, 56, 83**] using the following algorithms. Show your work at each step

1. Selection Sort
2. Heap Sort
3. Merge Sort
4. Quick Sort - use the following pivots: 29,10,56

Sorting

Discuss runtime and space complexity of each algorithm

1. Selection Sort
    a. space complexity?
        i. **O(1)** it is in place
        ii. **O(n)** also accepted if you explain that you are counting the original array
    b. runtime complexity?
        i. O(n^2)
2. Heap Sort
    a. space complexity?
        i. **O(n)** because we make a heap
    b. runtime complexity?
        i. **O(nlogn)** ... each insert is O(logn) and we do n inserts. Each poll is O(logn) and we do n polls = O(nlogn + nlogn) = O(nlogn)
3. Merge Sort
    a. space complexity?
        i. O(n) because create smaller arrays which are then merged
    b. runtime complexity?
        i. O(nlogn) ... runtime of merge is O(n) and we do log n merges
4. Quick Sort
    a. space complexity?
        i. O(1) in place
    b. runtime complexity?
        i. O(nlogn) with a good pivot
        ii. O(n^2) with a bad pivot

Data Structure Design Selection

You are designing a database system for a large e-commerce platform. The system needs to efficiently manage customer orders, allowing for quick retrieval and modification of orders. The main operations required are:

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Which data structure would you choose to implement the order management system, and why? Provide an explanation of your choice, considering factors such as time and space complexity

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

ExpandableArray?

1. Insertion: O(n)
2. Removal: O(n)
3. Updates: O(n) ... we need to find the order
4. Expansion: O(n)

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Doubly LinkedList?

1. Insertion: O(1)
2. Removal: O(1) since we're removing the first order that was put in (head)
3. Updates: O(n) since we need to search
4. Expansion: O(1) we can keep adding things to tail as they come in

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Stack (FILO)?

1. Insertion: O(1)
2. Removal: O(n) a stack is FILO but we want to remove from the first ordered...
    a. we'll need to copy over to another stack! Extra memory!
3. Updates: O(n) we'll need to copy over to another stack! Extra memory!
4. Expansion: O(1)

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Queue (FIFO)?

1. Insertion: O(1)
2. Removal: O(1)
3. Updates: O(n) we'll need to copy over to another stack! Extra memory!
4. Expansion: O(1)

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Balanced BST / AVL / Splay ?

1. Insertion: O(logn)
2. Removal: O(logn)
3. Updates: O(logn)
4. Expansion: O(1)

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Heap?

1. Insertion: O(logn)
2. Removal: O(logn) poll
3. Updates: O(logn)
4. Expansion: O(1)

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Hash Table?

1. Insertion: O(1)
2. Removal: O(1)
3. Updates: O(1)
4. Expansion: O(n)

Data Structure Design Selection

1. Rapid insertion of new orders into the system.
2. Efficient removal of orders based on time they were ordered.
3. Supporting fast updates or cancellations of orders.
4. Fast expansion to support a rapid increase in orders

Which data structure would you select?

DLL - only drawback is updates since we have to search

Hash Table - only drawback is expansion since its array based

Heap / Tree - logn for insert, remove, and update, but expansion is constant time.

Programming Questions

ChainHashMap - numElements

Add a method `int numElements()` to count the number of elements in the hash table. It should be a method within the `ChainHashMap` class. If needed, you may add additional methods to that class as well.

`ChainHashMap.java`

First Unique Character

Given a string `s`, find the first non-repeating character in it and return its index. If it does not exist, return `-1`. **You may use an additional data structure.** Discuss the runtime and space complexity. Your solution should have a complexity of **O(n)** for full credit.

Example 1:

Input: s = "leetcode"

Output: 0

Example 2:

Input: s = "loveleetcode"

Output: 2

Example 3:

Input: s = "aabb"

Output: -1

Ideas?

- for each char.. loop over the rest of the string to see if it exists again. O(n^2)

- What data structure has fast insertion and lookups?