CS151 Intro to Data Structures

Final Exam Review

Announcements

HW8 Due Sunday Dec 15th

Lab11 today - extra credit

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**

True / False

1. Given a Probe HashMap H, an element x will always be placed in slot hash(x), if H's load factor is below the threshold.

2. After removing the minimum element x from a Min Heap H, and re-inserting it, H.getRoot() will return x.

3. In an AVL tree, finding all prime numbers less than the root of a subtree x, results in O(n) runtime complexity, where n is the number of elements of the subtree.

4. A breadth-first traversal of a max heap prints the tree in descending order with O(n) time complexity where n is the number of elements in the heap.

Selecting the right data structure

You are building an online ticketing system for an event venue. The system needs to efficiently manage the following operations:

1. Sell a ticket: Sell a ticket to a customer and add their name to the system.
2. Cancel a ticket: Remove a customer from the system if they cancel their ticket.
3. Find the next customer to check in: Retrieve the name of the customer who bought their ticket first, but do not remove them.
4. Check in a customer: Mark the customer who bought their ticket first as checked in and remove them from the system.
5. Check if there are tickets sold: Determine if there are any customers left to check in.

Select a data structure to efficiently handle these operations. You can use any structure we have learned in this class. Justify your choice and explain the time complexity for each operation.

Working with Data Structures

Show the state of the underlying data structure after each step:

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

2. remove: 10, 17

Show this for the following data structures:

1. BST - implemented as an array

2. AVL Tree

3. Probe Hash Map with an underlying array of size 17 and h(x) = x % 17. Collisions should be handled with a quadratic probe.

4. Min Heap

What was the runtime complexity of insert and remove for each structure?

Perform the following operations - extra examples

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)

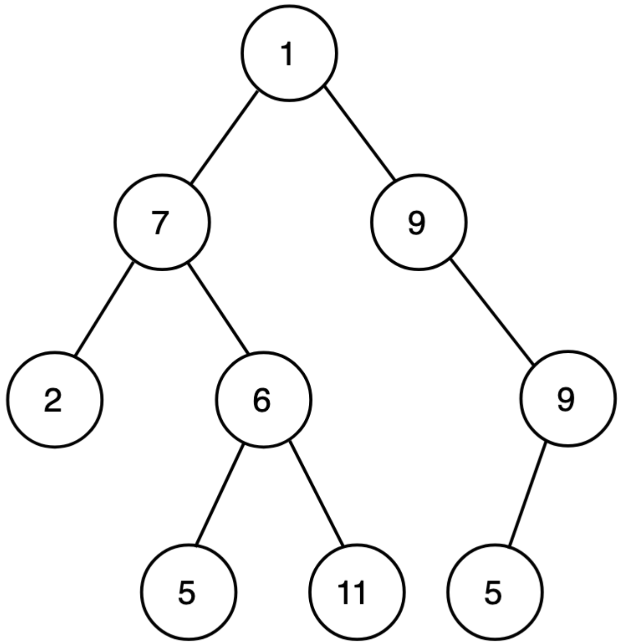
insert: 42, 17, 89, 5, 63, 28, 77

remove: 5, 17

what was the runtime complexity?

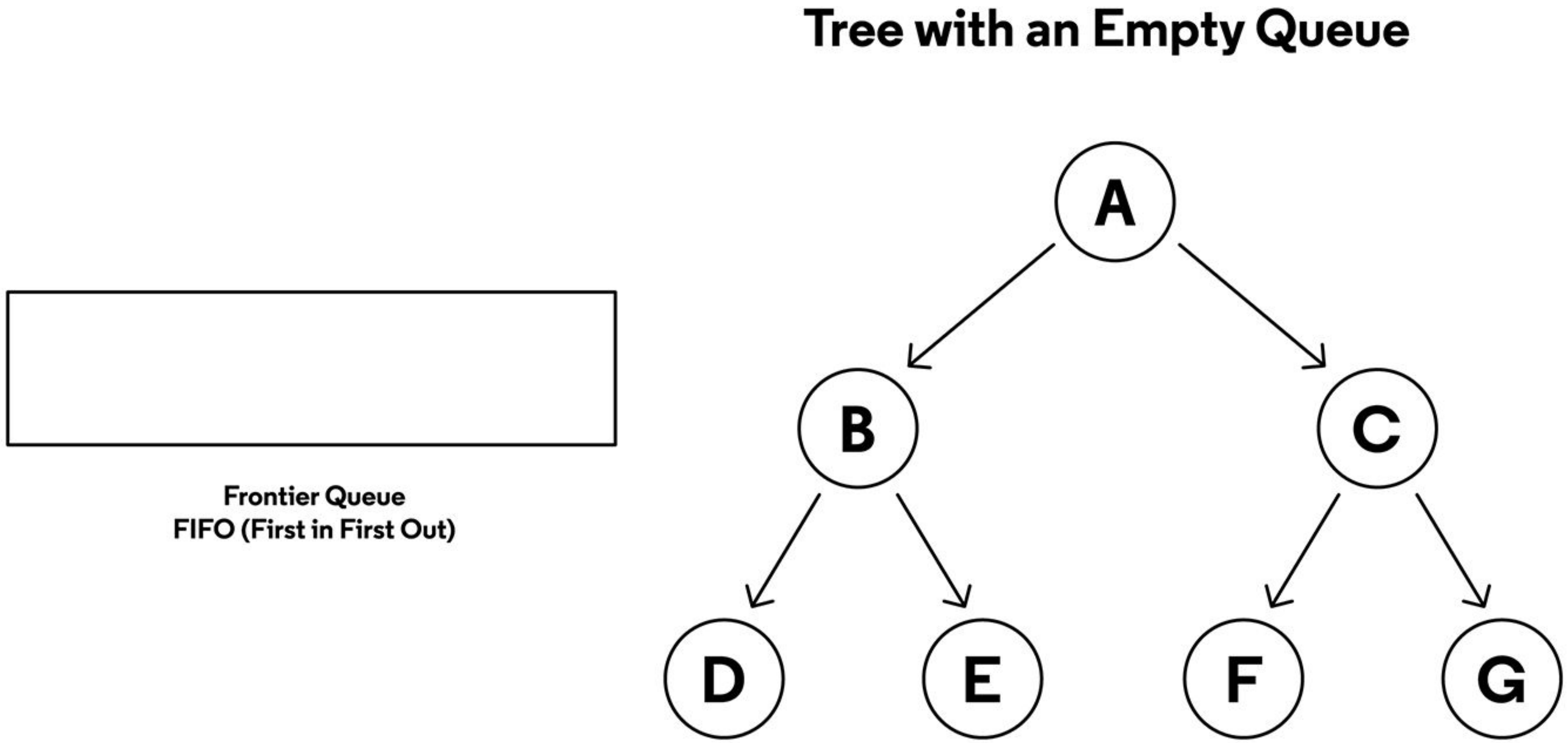
Breadth-First Traversal

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

Runtime Complexity

Sort these from fastest to slowest:

- O(n)
- O(n^2)
- O(logn)
- 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
   b. runtime complexity?
      i. **O(n^2)**
2. Heap Sort
   a. space complexity?
      i. **O(n) or O(1)** we did both in place and with an additional heap in class
   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

Complexity

For the given operation, sort the data structures from lowest to highest worst case run time complexity. Include a short 1 sentence explanation if necessary.

Removing an element x:

1. Unsorted array
2. Balanced binary search tree
3. Collision resistant probe hash map
4. Queue
5. Sorted Doubly Linked List

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?