CS 225: Exam 3

Review Session

Overview

- We will be referencing Practice Exam 3
- Covering some MCQ problems as well as an FRQ
- Coding questions tips
- Questions/Coding question walkthrough

Example Questions

BST Operations

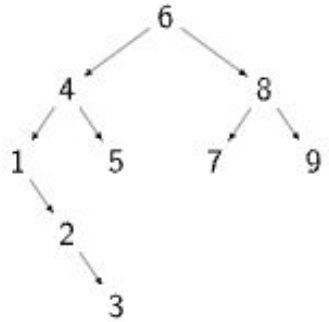
Consider a BST with n nodes. What is the minimum and maximum number of comparisons that you need to make in order to insert a new node?

- (a) $\log n$ and n
- (b) 1 and n
- (c) 1 and $\log n$
- (d) We do not have enough information to answer the question.
- (e) $\log n$ and $\log n$

Save & Grade | Save only | New variant

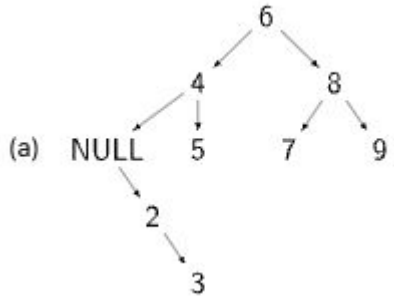
Binary Search Tree

Consider the following Binary Search Tree.

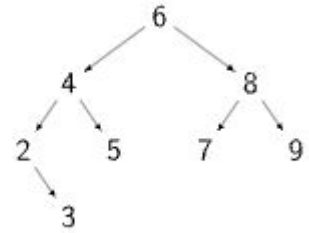


If we delete the node with the key 1, which is the correct tree?

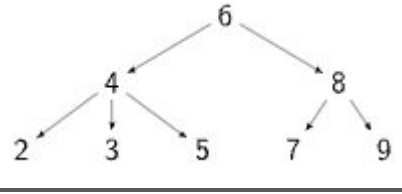
(a)



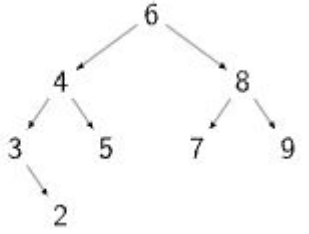
(b)



(c)



(d)



Sample Exam Question (**PE3**)

Data Structure Selection

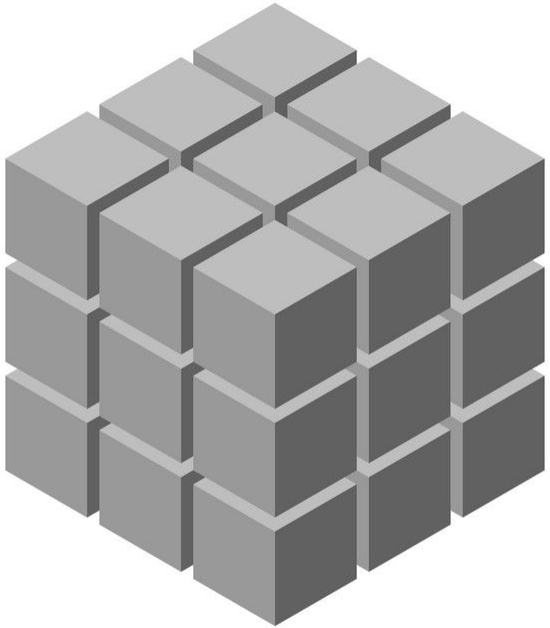
View...

You are tasked with developing a system to track the current contents of a very large (but fixed size) warehouse. The warehouse itself is represented as a three-dimensional grid, where items can be found using an (x, y, z) coordinate. Only one item may exist in any coordinate space at one time and an empty space has its own unique value indicating it is empty. Your solution must allow users to place, remove, or look up items when given a specific coordinate and you are expected to be able to handle a full warehouse. **You should optimize each operation equally.**

Explain your answer in the form of English sentences. As a guideline for your answer, be sure to:

1. Explicitly state the data structure you would implement to solve this problem.
2. Explicitly define variables for data size in the context of Big O. (For example, to say something is O(N), you must explicitly define N.)
3. Describe the interface (ADT) of the data structure as well as how you would use this interface to construct your data structure and solve the problem. **No implementation details about the functions are needed, just how you would use them.**
4. For each function described in (3), explicitly state the Big O of each function.
5. Justify why your solution is the optimal solution by highlighting what is good about your choice.

Where do I start?



FRQ Essentials (Rubric)

Note: Even if the wrong structure is selected, students can still earn up to 70%!

Selecting the Optimal Structure	30%
Understanding the structure	30%
Explanation (Big O Defined)	10%
Solving the problem	30%

Sample Exam Response (**PE3**)

I would solve this problem using a 3-dimensional vector pre-allocated to the maximum size needed. The key variables in this problem are N, the number of items being stored at one time, and M, the maximum size of the warehouse. (Note: Neither is actually relevant to Big O!) The 3D vector has three key operations, insert, remove, and find and all of them can be performed in O(1) time using array access. Insert() looks up the (x, y, z) coordinate and updates the value stored there. Remove() looks up the (x, y, z) coordinate and updates the value to 'empty' Find() looks up and returns the item stored at (x, y, z)

Sample Exam Response Cont. (**PE3**)

To solve this problem, every time a new new is added, a call to insert will add it to my 3D vector. Every time I need to access an item, a call to find will find and return the item of interest. Every time I need to remove an item, a call to remove will delete the transaction by setting it to some null value. Since the vector array has random access and every one of my operations requires only a single lookup and modification, we have a total runtime of O(1) for each interaction. We cannot do better than this, assuming we have the space needed for the 3D matrix. If I need to insert or remove or lookup N items, this has a runtime of O(N). An optimal solution here must be O(1) for all three operations and include all three as part of the interface. The overall runtime should be either O(1) or linear depending on what they set their key variable to.

Coding Tips

The coding question will be related to trees:

- Recursion is always helpful
- Make helper functions to assist with recursion

General tips

- Spin up your coding workspace while reading the question
- Skim through the header file
- Be careful with the function declaration (use spiral rule)
- #include <bits/stdc++.h>
- Run local tests/memcheck in main.cpp