The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Winter 2024

Lab 8: Binary Trees, AVL Trees, and Tree Traversals

Instructions:
Work on this document with your group, then enter the answers on the canvas quiz. **This assignment is due on April 8th, at 11:59 PM.**

Note:
Be prepared before you meet with your lab group, and read this document so that you know what you must submit for full credit. You can even start it ahead of time and then ask questions during any lab section for help completing it.

You MUST include the following assignment identifier at the top of every file you submit to the autograder as a comment. This includes all source files, header files, and your Makefile (if there is one).

Project Identifier: EAA16B5C3724FBFD78F132136AABBBBA4952E261

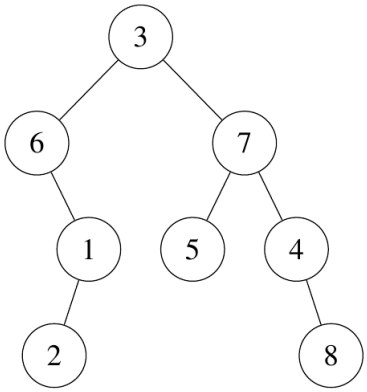
---

1 Logistics

1. When are lab 7's autograder and quiz due?
   A. 04/04/2024
   B. 04/02/2024
   C. 04/01/2024
   D. 04/08/2024
2. When are lab 8's autograder and quiz due?
   A. 04/10/2024
   B. 04/12/2024
   C. 04/04/2024
   D. 04/08/2024
3. What is the last day you can submit lab 8's handwritten problem to an instructor in lab?
   A. 04/04/2024
   B. 04/08/2024
   C. 04/01/2024
   D. 04/02/2024
4. When is project 3 due?
   A. 04/08/2024
   B. 04/02/2024
   C. 04/01/2024
   D. 04/04/2024
5. What is the theme of lab 8's autograder assignment?
   A. AVL Trees
   B. Dynamic Programming
   C. Proving P=NP
   D. Euchre

2 Trees

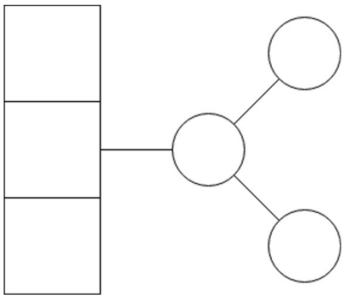
For questions 6-8, consider the following tree:



Acceptable answer formats for ALL tree traversal questions in this lab:

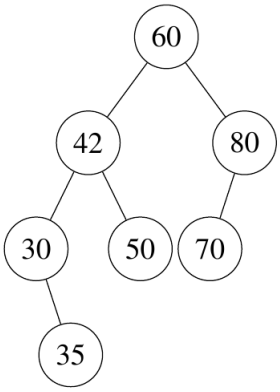
- a comma-separated list with spaces: 1, 2, 3, 4, 5, 6, 7, 8, 9
- a comma-separated list without spaces: 1,2,3,4,5,6,7,8,9
- a list separated by only spaces: 1 2 3 4 5 6 7 8 9

6. What is the in-order traversal of the tree above?
7. What is the pre-order traversal of the tree above?
8. What is the post-order traversal of the tree above?
9. Given the following pre-order and in-order traversals of a binary tree, what is its post-order traversal?
   Pre-order: 3, 4, 0, 1, 5, 2, 6, 7
   In-order: 4, 3, 1, 5, 0, 2, 7, 6



10. You have a hash table that uses the separate chaining collision resolution method. However, instead of chaining elements using a linked list, this hash table chains elements using an AVL tree (see diagram). What is the worst-case time complexity of searching for an element in this hash table, if it contains n elements? Assume the hash function runs in $\Theta(1)$ time.
    A. $\Theta(1)$
    B. $\Theta(\log n)$
    C. $\Theta(n)$
    D. $\Theta(n \log n)$
    E. $\Theta(n^2)$

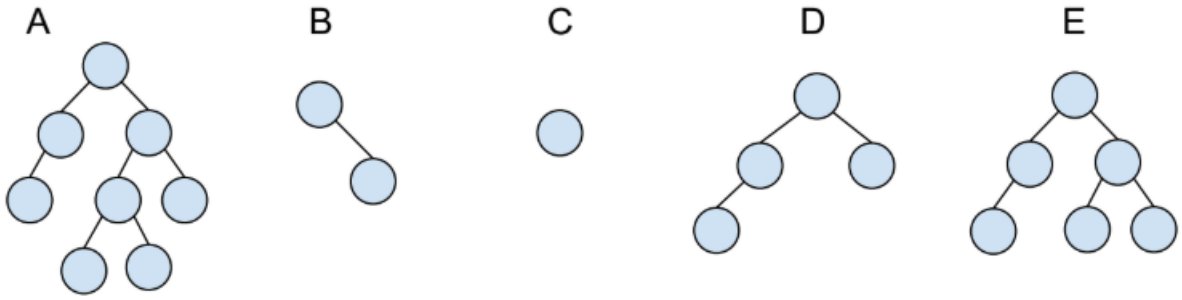
11. Consider the following AVL tree:



How many tree rotations (with double rotations counting as 2) would occur if the following elements were inserted in the given order: 75, 68, 65, 40?

A. 2
B. 3
C. 4
D. 5
E. more than 5

for questions 12-13, consider the following binary trees:



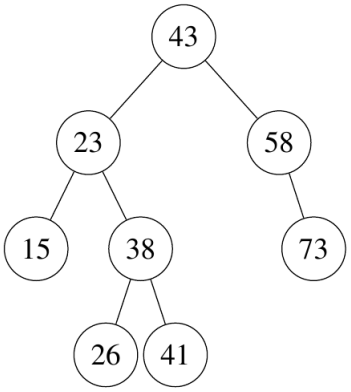
12. Which of the above trees (A, B, C, D, or E) is/are complete binary trees? Select all that apply.

A. A
B. B
C. C
D. D
E. E

13. Which of the above trees (A, B, C, D, or E) is/are proper (full) binary trees? Select all that apply.

A. A
B. B
C. C
D. D
E. E

For questions 14-15, consider the following tree:



14. Suppose we insert the elements 95, 71, and 82 into this AVL tree (in the order given). What is the pre-order traversal of the resulting tree after all operations have been completed? Remember to rebalance after each step.

15. Suppose we delete the element 43 from the original AVL tree (without the insertions in the previous question). What is the post-order traversal of the resulting tree after the deletion? Remember to rebalance, if necessary. You may replace the deleted node with either the inorder successor or predecessor; both answers will be accepted.

16. Which of the following statements is/are TRUE? Select all that apply.
    A. finding an arbitrary element in an AVL binary search tree has $\Theta(n)$ worst-case time complexity
    B. traversing a binary tree can be done in $\Theta(\log n)$ time
    C. inserting an element into a binary search tree has $\Theta(n)$ worst-case time complexity
    D. an in-order traversal of any BST produces sorted output, even if it is unbalanced
    E. none of the above statements are true

17. Consider the following snippet of code:

```
int mysteryFunction(struct Node *root) {
    if (root == nullptr) {
        return 0;
    } else if (root->left == nullptr && root->right == nullptr) {
        return 1;
    } else {
        return 1 + mysteryFunction(root->left) + mysteryFunction(root->right);
    }
}
```

    What does the mysteryFunction do if root is the root of a binary tree?
    A. it returns the number of leaf nodes
    B. it returns the number of internal nodes
    C. it returns the number of nodes in the tree
    D. it returns the height of the tree
    E. it returns the depth of the tree
    F. none of the above

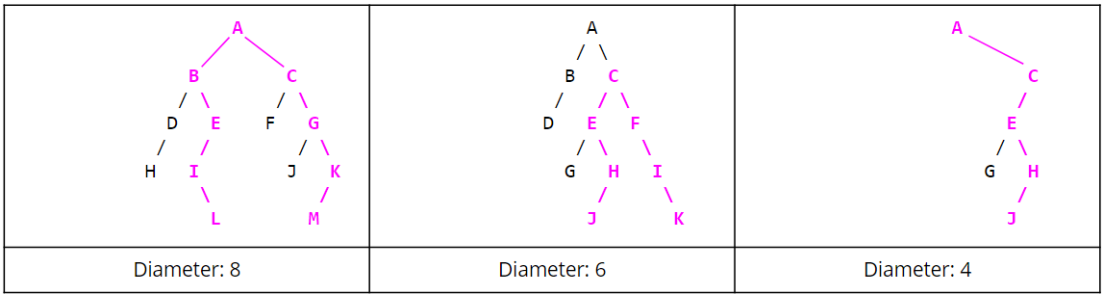
18. Consider the Tree Sort algorithm in which sorting is achieved by placing all elements to be sorted into a binary search tree and then taking the in-order traversal of the tree. Another sorting algorithm, AVL Tree Sort, places all elements into an AVL binary search tree and then takes the in-order traversal of the tree. Which of the following statements is/are TRUE about these sorts? Select all that apply.
    A. both Tree Sort and AVL Tree Sort perform well on input that is already in nearly sorted order
    B. the best-case time complexity of Tree Sort is $\Theta(n \log n)$
    C. the worst-case time complexity of AVL Tree Sort is $\Theta(n^2)$
    D. the best-case space complexity of Tree Sort is $\Theta(n)$
    E. the best-case space complexity of AVL Tree Sort is $\Theta(\log n)$

3 Handwritten Problem

This problem is to be submitted independently. We recommend trying it on your own, checking your answer with a neighbor or a group and discussing solutions, and then submitting it to your lab instructor. These will be graded on effort, not by correctness. We primarily want to see that you were thinking about the problem. Please implement your solution on the handwritten template or some other 8.5"x11" sheet of blank, unlined paper. Starter files may be found on Canvas if you wish to write and test this function further on your own.

Diameter of a Binary Tree:

Background
Let's say the diameter of a tree is the maximum number of edges on any path connecting two nodes of the tree. For example, here are three sample trees and their diameters. In each case the longest path is bolded and shown in purple. Note that there can be more than one longest path.



Your task: Implement the function `diameter` that computes the diameter of a binary tree represented by a pointer to an object of type BinaryTreeNode. Assume that `nullptr` represents an empty tree or a missing child. Do not modify the definition of the BinaryTreeNode class. You may write one or more helper functions if you need.

Implement diameter in $O(n^2)$ or better time (it can be done in O(n)).

See the definition of BinaryTreeNode below.

```
class BinaryTreeNode {
public:
    BinaryTreeNode* left;
    BinaryTreeNode* right;
    int value;
    BinaryTreeNode(int n)
     : value(n), left(nullptr),
       right(nullptr) {}
};
```

4 Coding Assignment

You can work on these problems by yourself or with your group, but a solution must be submitted to the autograder for each individual.

Ro-Tater Tots
For this lab, finish the implementation of an AVL tree class.
In the average case, unbalanced binary search trees attain $\Theta(\log n)$ time operations, but their worst case is $\Theta(n)$. For this problem, you are tasked with completing the implementation for an AVL tree, which attains $\Theta(\log n)$ time in the worst case. Searching has been completed for you (it's exactly the same as in an ordinary BST).

You're given an AVL tree class with the following Node struct type (in the `avl_lab.h file`):

```
struct Node {
   int datum;
   int height;
   Node* left;
   Node* right;
   int left_height() {
       return left ? left->height : 0;
   }
   int right_height() {
       return right ? right->height : 0;
   }
   int balance() {
       return left_height() - right_height();
   }
   // when the height of its children change, call this function
   // to recalculate the height of this node, the parent
   void fix_height() {
       height = 1 + max(left_height(), right_height());
   }
};
```

Note that this is really just a node with a datum, height, left, and right; it also has some helper functions to handle height and balance. For this problem, you will implement three functions:

```
// insert_node returns the new root of this subtree after inserting datum.
// if datum already exists in the tree, the copy is inserted to its right.
AVL::Node* AVL::insert_node(AVL::Node* node, int datum);

// rotate_left performs a left rotation;
// it returns the new 'root' of the rotated subtree
// (remember to update the heights of nodes!)
// you may assume that it has a right child
AVL::Node* AVL::rotate_left(AVL::Node* node);

// rotate_right performs a right rotation;
// it returns the new 'root' of the rotated subtree
// (remember to update the heights of nodes!)
// you may assume that it has a left child
AVL::Node* AVL::rotate_right(AVL::Node* node);
```

You may define any helpers functions or methods that you like, although you can complete the lab without any.

To help you debug your implementation, a tree-printing utility has been implemented for you. main() will construct 3 different AVL trees using your code and print them. The correct tree structures for these 3 trees are given as comments in main(). If the diagrams your program prints look different, there's a bug in your code!

Submission

Add you and (if you have one) your partner's names as a comment near the top of the `avl_lab.h` file. To submit to the autograder, run `make fullsubmit` and submit the `fullsubmit.tar.gz` file that is created.

If you are working with a partner, both partners must submit to the autograder. Only students who submit code to the autograder will receive points. It's fine if both of you submit the same code for this assignment.

Your program will be judged on 10 different test cases, similar to the ones provided to you in the `avl_lab.cpp` file. These test cases will involve searching for a number in a large AVL tree. You can infer the following from the autograder test case names:

- a test case beginning with **L** means that the number is located to the left of the root
- a test case beginning with **R** means that the number is located to the right of the root
- a test case with the letter **M** indicates that the number cannot be found in the tree
- a test case with the letter **X** indicates that the number can be found in the tree

