313E Programming Assignment 11
Binary Trees

Setup

This assignment may be completed individually or with a pair programming partner. Be sure to include your and your partner's name and EID in the Python file header. If you are working alone, you may delete one of them.

Complete these steps to prepare for the Assignment.

Check	Description
☐	Download tree.py and tree.in. You will be working on the tree.py file. You may not change the file names. Otherwise, the grading script will not work.
☐	Place both files in the same folder/directory.

If you would like to turn on autosave in VSCode, click on **File -> Autosave.**

Problem Description

In this assignment, you will be finishing the implementation of Binary Tree classes Node and Tree. There are several short methods that you will have to write.

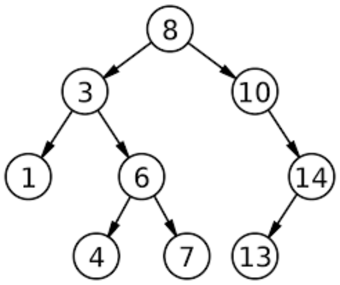
Requirements

For this programming assignment, we have defined the **Node** class for you (which defines each node in the binary search tree) along with the **Stack** and **Queue** class, which you may choose to use. Based on this, you will be working on the **Tree** class. You will finish implementing seven binary search tree methods. You will need to write helper methods to finish implementing these methods. You may include as many helper functions as needed.

There are a few restrictions you must follow:

1. For this assignment, you will be submitting your code to Gradescope to test your code. You may write your own code to test your file in main(), and you may use **tree.in** to construct a tree. You can use input redirection to read from this file.
2. **You may not change the names or parameters of the functions listed. They must have the functionality as given in the specifications. You can always add more functions than those listed.**
3. **You may not import any additional external libraries in your solution besides the given deque import from collections and sys.**

You will modify the following functions:



is_similar(self, tree)
Compares to another tree. If the trees have the same nodes with the same data in the same positions in the tree (if a node is a left vs right child matters!), return True. Otherwise, return False. If an identical tree was compared to the example tree above, the method would return **True.** If one or more compared values at the same position were not identical, the method would return **False.**

get_level(self, level)
Returns a list of all nodes at the specified ***level*** from left to right. If that level does not exist for that tree, return an empty list. Assume that the root is at level **0**. Using the example tree above, if **level** is **2,** the list of nodes would be **[1, 6, 14]**.

get_height(self)
Returns the height of the Tree. Recall that the height of a binary search tree is the longest path length from the root to a leaf. The height of the example tree above is of height **3**.

num_nodes(self)
Returns the total number of nodes in the Tree. This function must be implemented recursively. Do **not** create a num_nodes parameter in the Tree class.
Hint: The total number of nodes in the tree is the sum of the number of nodes in the left subtree and the number of nodes in the right subtree and the root itself. For the example tree tree above, the total number of nodes is **9**.

range(self)
Return the range of values stored in the Tree. For the example tree above, the range of values is **13.**

left_side_view(self)
Return the list of the nodes that you see from the left side. The order of the output should be in descending order, from the top of the Tree to the bottom of the Tree. In the example tree above, the returned list would be **[8, 3, 1, 4]**.

sum_leaf_nodes(self)
Return the sum of the values in all of the leaves. For the example tree above, the sum of the lead nodes would be **25**.

def main()
This method will not be graded. You may write code in this method to test your own program, reading from the provided tree.in file (or a .txt file of your own).

Input
For this program, you will submit your code to Gradescope to test your code. You may write your own code to test your file in main(), and you may use tree.in or your own .txt file as input. You can use input redirection to read from this file. **tree.in** will contain data for 3 binary trees, each on its own line.

```
50 30 70 10 40 60 80 7 25 38 47 58 65 77 96
50 30 70 10 40 60 80 7 25 38 47 58 65 77 96
58 77 65 30 38 50 7 25 47 96 80 10 60 70 40
```

The code can assume that the file is structured properly.

Output

There will be no specific code output for this program.

Running Your Program

You can run the program by typing the following command into your terminal:

```
python3 tree.py < tree.in
```

Alternatively, if you are using a Windows computer and are having trouble with input redirection <, you can run:

```
cat tree.in | python3 tree.py
```

Grading

Visible Test Cases - 75/100 points
The program output exactly matches the expected output for each function. To receive full credit, the program must produce the correct output based on all requirements in this document and pass all the test cases.

Hidden Test Cases - 15/100 points
The program output exactly matches the expected output for each function. To receive full credit, the program must produce the correct output based on all requirements in this document and pass all the test cases.

Style Grading - 10/100 points
The tree.py file must comply with the PEP 8 Style Guide.

If you are confused about how to comply with the style guide, paste the error or the error ID (e.g. C0116 for missing function or method docstring) in the search bar here in the Pylint Documentation, and you should find an example for how to fix your problem.

The TAs will complete a manual code review for each assignment to confirm that you have followed the requirements and directions on this document. Deductions will occur on each test case that fails to follow requirements.

Submission

Follow these steps for submission.

Check	Description
▢	Verify that you have no debugging statements left in your code.
▢	Submit **only** tree.py (the starter code file with your updates) to the given assignment in Gradescope. This will cause the grading scripts to run.
▢	If you have a partner, make sure you are submitting it for you **and** your partner on Gradescope. See this Gradescope documentation for more details.

When the grading scripts are complete, check the results. If there are errors, evaluate the script feedback, work on the fix, test the fix, and then re-submit the file to Gradescope. You can submit as many times as necessary before the due date.

NOTE: By default, only the most recent submission will be considered for grading. If you want to use a previous submission for your final grade, you must activate it from your submission history before the due date.

Academic Integrity

Please review the Academic Integrity section of the syllabus. We will be using plagiarism checkers, and we will cross-reference your solution with AI-generated solutions to check for similarities. Remember the goal in this class is not to write the perfect solution; we already have many of those! The goal is to learn how to problem solve, so:

- Don't hesitate to ask for guidance from the instructional staff.
- Discuss with peers about **only** bugs, their potential fixes, **high-level** design decisions, and project clarifying questions. Do not look at, verbalize, or copy another student's code when doing so.

Attribution

Thanks to Dr. Carol Ramsey and Dr. Shyamal Mitra for the instructions template and this assignment.