Carleton University
Department of Systems and Computer Engineering
SYSC 2100 - Algorithms and Data Structures - Winter 2021

Lab 10 - Binary Trees

Submitting Lab Work for Grading

Remember, you don't have to finish the lab by the end of your lab period. The deadlines for submitting solutions to cuLearn for grading are listed in the *Wrap Up* section at the end of this handout. Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the deadline.

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

References

Problem Solving with Algorithms and Data Structures Using Python, Chapter 7, *Trees and Tree Algorithms*, Sections 7.1 to 7.7 (*Tree Traversals*).

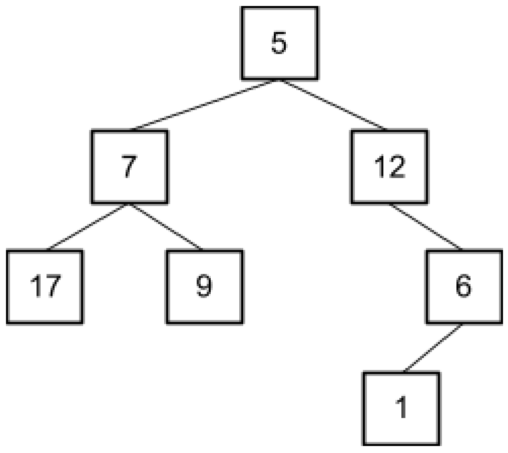
Lecture slides: *Binary Trees*

Getting Started

Download binary_tree.py from the *Lab Materials* section of the main cuLearn course page.

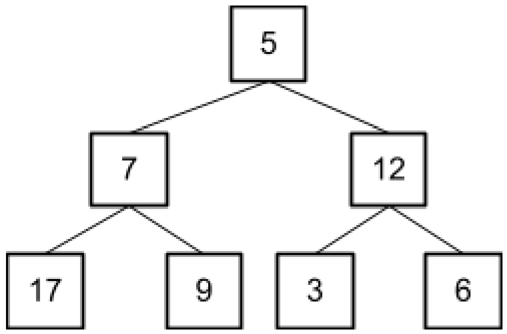
Class `BTNode` defines the nodes in a binary tree. Methods `insert_left` and `insert_right` insert a new node as the left or right child of a `BTNode` (see Section 7.5 of the textbook).

Exercise 1: Replace the `raise` statement in `build_binary_tree` with the code that creates this tree. The function must call `insert_left` and `insert_right` to insert the nodes beyond the root node.



Use the shell or write a short script to test the function. If you write a script, please put it in a separate module (file). We don't want test scripts in the `binary_tree.py` file you submit to cuLearn.

Exercise 2: Replace the `raise` statement in `build_full_binary_tree` with the code that creates this tree. The function must call `insert_left` and `insert_right` to insert the nodes beyond the root node.



Use the shell or write a short script to test the function.

Exercise 3: Function `preorder_print`, which is described in the lecture slides, prints the contents of a binary tree using a preorder traversal.

Predict what will be printed when `preorder_print` is passed the trees created by `build_binary_tree` and `build_full_binary_tree`. Now call `preorder_print` to print the trees. Were your predictions correct?

Exercise 4: Predict what would be printed by an inorder traversal of the trees created by `build_binary_tree` and `build_full_binary_tree`.

Read the docstring for function `inorder_print`. Replace the `raise` statement with a recursive implementation of the function. Your function <u>cannot</u> have any loops. Inorder traversals are described in the lecture slides and in Section 7.7 of the textbook.

Use the shell to test your function. Were your predictions correct?

Exercise 5: Predict what would be printed by a postorder traversal of the trees created by `build_binary_tree` and `build_full_binary_tree`.

Read the docstring for function `postorder_print`. Replace the `raise` statement with a recursive implementation of the function. Your function <u>cannot</u> have any loops. Postorder traversals are described in the lecture slides and in Section 7.7 of the textbook.

Exercise 6: Function `size` recursively calculates the number of nodes in a binary tree. Convert this function into a recursive method in class `BTNode`. A stub implementation is provided. (Section 7.7 in the textbook, Listings 2 and 3, describes how to convert a recursive function into a recursive method.)

Test the `size` method. For a given binary tree, does it return the same value as the `size` function?

Exercise 7: Read the docstring for method `count`. Replace the `raise` statement with a recursive implementation of the method.

Programming Style - Code Formatting

Before submitting your code, please run it through Wing 101's source reformatter, as described in following paragraphs.

Wing 101 can easily be reconfigured to reformat your code so that it adheres to some of the formatting conventions described in the "official" Python coding conventions document, PEP 8 - *Style Guide for Python Code*.[1]

To configure reformatting, follow the instructions in Section 3.2 of *Installing Python 3.9.1 and Wing 101 7.2.7*, which is posted on cuLearn. (The instructions apply to both the Windows 10 and macOS versions of Wing 101.)

If you prefer, you manually reformat your files even if you've disabled automatic reformatting. This is described in Section 3.3. Note that you still have to open the Auto-formatting window to configure which PEP 8 conventions are followed, as described in Section 3.2.

Manually reformatting an open file is easy:

- From the menu bar, select Source > Reformatting
- From the pop-up menu, select Reformat File for PEP 8

Wing 101 7.2.8 fixes a bug in release 7.2.7: the Reformat Selection for PEP 8 command now works.

Programming Style - Writing Readable Code

Focus on writing code that is readable and maintainable, and not just code "that works". Using a code reformatter can improve the layout of your code, but it can't enforce every coding convention. For example, a reformatter won't change your variable names if they aren't descriptive or follow the "official" Python convention (that is, variables names are lowercase, with words separated by underscores).

[1] https://www.python.org/dev/peps/pep-0008/

Here are some things to consider:[2]

"The ease by which other *people* can read and understand a program (often called "readability" in software engineering) is perhaps the most important quality of a program. Readable programs are used and extended by others, sometimes for decades. For this reason, we often say that programs are written to be read by humans, and only incidentally to be interpreted by computers.

A program is composed well if it is concise, well-named, understandable, and easy to follow.

Excellent composition does not mean adhering strictly to prescribed style conventions. There are many ways to program well, just as there are many styles of effective communication. However, the following guiding principles universally lead to better composition of programs:

- **Names**. To a computer, names are arbitrary symbols: "xegyawebpi" and "foo" are just as meaningful as "tally" and "denominator". To humans, comprehensible names aid immensely in comprehending programs. Choose names for your functions and variables that indicate their use, purpose, and meaning.

- **Functions**. Functions are our primary mechanism for abstraction, and so each function should ideally have a single job that can be used throughout a program. When given the choice between calling a function or copying and pasting its body, strive to call the function and maintain abstraction in your program.

- **Purpose**. Each line of code in a program should have a purpose. Statements should be removed if they no longer have any effect (perhaps because they were useful for a previous version of the program, but are no longer needed). Large blocks of unused code, even when turned into comments, are confusing to readers. Feel free to keep your old implementations in a separate file for your own use, but don't turn them in as your finished product.

- **Brevity**. An idea expressed in four lines of code is often clearer than the same idea expressed in forty. You do not need to try to minimize the length of your program, but look for opportunities to reduce the size of your program substantially by reusing functions you have already defined."

[2] Adapted from a style guide used by students in an introductory programming course at the University of California, Berkeley.

Wrap Up

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

The submission deadline for this lab is Sunday, March 28, 23:55 (Ottawa time), for all lab sections.

To submit your lab work, go to the cuLearn page **for your lab section** (not the main course page). Submit binary_tree.py. Ensure you submit the version of the file that contains your solutions, and not the unmodified file you downloaded from cuLearn! You are permitted to make changes to your solutions and resubmit the file as many times as you want, up to the deadline. Only the most recent submission is saved by cuLearn.

Last edited: March 23, 2021