UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephone: (306) 966-4886, Facimile: (306) 966-4884

CMPT 145
Spring 2018
Principles of Computer Science

Assignment 8

Binary trees

Date Due: July 27, 2018, 10pm **Total Marks: 45**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Resources for Assignment 8

On the Assignment 8 Moodle page you can find the following files:

- The TreeNode ADT in the file `treenode.py`
- Four traversal functions in the file `traversals.py`
- Example trees in the file `exampletrees.py`
- Some useful functions on trees in `treefunctions.py`, including:
  - `is_leaf(t)` Returns `True` if `t` is a treenode with no children; otherwise, `False`.
  - `to_string(t)` Returns a string that represents the tree. The string presents the tree with the root at the top left, and then levels are left to right, rather than top to bottom. In other words, the tree is rotated a quarter turn counter-clockwise.

Question 0 (5 points):

Purpose: To force the use of Version Control in Assignment 8

Degree of Difficulty: Easy

You are expected to practice using Version Control for Assignment 8. This is a tool that you need to be required to use, even when you don't need it, so that when you do need it, you are already familiar with it. Do the following steps.

1. Create a new PyCharm project for Assignment 8.
2. Use `Enable Version Control Integration...` to initialize Git for your project.
3. Download the Python and text files provided for you with the Assignment, and add them to your project.
4. Before you do any coding or start any other questions, make an initial commit.
5. As you work on each question, use Version Control frequently at various times when you have implemented an initial design, fixed a bug, completed a question, or want to try something different. Make your most professional attempt to use the software appropriately.
6. When you are finished your assignment, open the terminal in your Assignment 8 project folder, and enter the command: `git --no-pager log` (double dash before the word 'no'). The easiest way to do this is to use PyCharm, locate PyCharm's `Terminal` panel at the bottom of the PyCharm window, and type your command-line work there.

   Note: You might have trouble with this if you are using Windows. Hopefully you are using the department's network filesystem to store your files. If so, you can log into a non-Windows computer (Linux or Mac) and do this. Just open a command-line, `cd` to your A8 folder, and run `git --no-pager log` there. If you did all your work in this folder, git will be able to see it even if you did your work on Windows. Git's information is out of sight, but in your folder.

   Note: If you are working at home on Windows, Google for how to make git available on your command-line window. You basically have to tell the command-line app where the git app is.

You may need to work in the lab for this; Git is installed there.

What to Hand In

After completing and submitting your work for Questions 1-4, open a command-line window in your Assignment 8 project folder. Run the following command in the terminal: `git --no-pager log` (double dash before the word 'no'). Git will output the full contents of your interactions with Git in the console. Copy/paste this into a text file named `a8-git.log`.

If you are working on several different computers, you may copy/paste output from all of them, and submit them as a single file. It's not the way to use git, but it is the way students work on assignments.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 5 marks: The log file shows that you used Git as part of your work for Assignment 8. For full marks, your log file contains
  - Meaningful commit messages.
  - At least two commits per programming question for a total of at least 8 commits.

Question 1 (16 points):

Purpose: To practice recursion on binary trees.

Degree of Difficulty: Easy to Moderate to Tricky

You can find the `treenode` ADT on the assignment page. Using this ADT, implement the following functions:

1. `count_node_types(tnode)` Purpose: Returns a tuple containing the number of leaf nodes in the tree, and the number of non-leaf nodes in the tree. A leaf node is a node without any children. The `is_leaf()` function provided in `treefunctions.py` can be used to check if a node is a leaf node. Remember, you can use circle brackets to create a tuple:

```
a_tuple = ("a", "b")
print( a_tuple[0] )
# Prints out "a"
```

2. `copy(tnode)` Purpose: To create an exact copy of the given tree, with completely new treenodes, but exactly the same data values, in exactly the same places. If `tnode` is `None`, return `None`. If `tnode` is not `None`, return a reference to the new tree.

3. `collect_data_inorder(tnode)` Purpose: To collect all the data values in the given tree. Returns a list of all the data values, and the data values appear in the list according to the in-order sequence. Hint: Base this function on the `in_order()` traversal, which you can find in `traversals.py`.

4. `alter_subtrees(tnode, left_tree_offset, right_tree_offset, root_offset=0)` Purpose: Adds the given offsets to the data values of each subtree (and current root). Assume that the given tree has data values that are numbers only. Ex: `alter_subtrees(a_tree, -2, 2, 0)` will subtract 2 from each LEFT subtree data value, and add 2 to each RIGHT subtree data value. Since this is a recursive function, this function will be called MULTIPLE times, thus adding MULTIPLE offsets to each data value (or having the offset ACCUMULATE). The offset that should be applied to the current root is passed on as the `root_offset`.
An example of `alter_subtrees()` is with a tree containing only 0's below:

```
#           0
#      0          0
#   0     0    0     0
#  0 0   0 0  0 0   0 0
alter_subtrees(zero_tree, -1, 1, 0)
# Becomes:
#            0
#      -1          1
#   -2     0    0     2
#  -3 -1 -1 1 -1 1   1 3
```

On Moodle you will find a file called `exampletrees.py` which has a few example trees that you can use for demonstration and testing.

You'll also find some useful functions in the file `treefunctions.py`.

Note: Test your functions carefully. You may start with the trees in the Python file provided, but those are demonstrations. Your testing could and probably should be a bit more deliberate.

What to Hand In

- A file `a8q1.py` containing your 4 functions.
- A file `a8q1_testing.py` containing your testing for the 4 functions.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

Each function will be graded as follows:

- 1 mark: Your function has a good doc-string.
- 1 mark: Your function has a correct base case.
- 1 mark: Your function's recursive case is correct.
- 1 mark: You tested your function adequately.

Question 2 (8 points):

Purpose: To do more thinking about binary trees.

Degree of Difficulty: Moderate

During a trial a judge or jury must weigh the evidence presented by the defence and the prosecution. To help them with this matter, you must write a function called `weigh_tree(root)` that does the following:

1. Takes in as a parameter the root of a tree created by the Tree Node ADT.
2. Returns a tuple with 3 values: the summed total of the left subtree, the data value of the root node, and the summed total of the right subtree.

You may assume the passed-in tree only contains numbers.

```
#         2
#     7       5
#   11  6
```

The tree above should return: (24, 2, 5)

What to Hand In

- A file called `a8q2.py`, containing your function definition.
- A file called `a8q2_testing.py`, containing your testing.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of the file.

Evaluation

- 4 marks: Your function works.
- 4 marks: Your testing is adequate.

Question 3 (8 points):

Purpose: To do more thinking about binary trees; to practice the art of internal functions; to practice the art of tupling.

Degree of Difficulty: Moderate

Write a function `complete(tnode)` that returns `True` is the given tree is a complete binary tree, and `False` otherwise.

In class we defined a complete binary tree as follows:

> A *complete* binary tree is a binary tree that has exactly two children for every node, except for nodes at the maximum level, where the nodes are barren (i.e., leaf nodes).

Visually, complete binary trees are easy to detect. But if you're just given a root node, it will be harder.

Consider the function below:

```
1  def bad_complete(tnode):
2      def cmplt(tnode):
3          if tnode is None:
4              return 0
5          else:
6              ldepth = cmplt(tn.get_left(tnode))
7              rdepth = cmplt(tn.get_right(tnode))
8              if ldepth == rdepth:
9                  return rdepth+1
10             else:
11                 return -1
12     result = cmplt(tnode)
13     return result > 0
```

Notice the internal definition of `cmplt()`. This function is almost identical to our `height` function except for lines 8-11. The function uses the integer value -1 to report an incomplete tree. Whether this use of the return value is good or bad is a matter of opinion, but without an alternative, we make a virtue of necessity, which is not always good!

Rewrite the internal definition to return a tuple of 2 values, `flag, height`, where:

- `flag` is True if the subtree is complete, False otherwise
- `height` is the height of the subtree, if `flag` is True, `None` otherwise.

This technique is called "tupling." You'll also have to change the second-to-last line of the function, line 12, to account for the change to the internal function.

Hint: Your internal function will return a tuple with 2 values. Python allows tuple assignment, i.e., an assignment statement where tuples of the same length appear on both sides of the =. For example:

```
# tuple assignment
a,b = 3,5

# tuple assignment
a,b = b,a
```

What to Hand In

- The file `a8q3.py` containing the function definition, using tupling, and an internal function.
- The file `a8q3_testing.py` containing testing for your function.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 2 marks: you used an internal function to do most of the work
- 3 marks: your internal function correctly uses tupling
- 1 mark: your function calls the internal function correctly, and returns a Boolean value only.
- 2 marks: your testing is adequate.

Question 4 (8 points):

Purpose: To do more thinking about binary trees.

Degree of Difficulty: Tricky Tricky Tricky

We say that a binary tree is *ordered* if all of the following conditions are true:

1. The left subtree is ordered.
2. The right subtree is ordered.
3. If the left subtree is not empty, all of the data values in the left subtree are less than the data value at the root.
4. If the right subtree is not empty, all of the data values in the right subtree are greater than the data value at the root.

The empty tree is *ordered* by definition.

Write a function `ordered(tnode)` that returns `True` if the given tree is *ordered*, and `False` otherwise.

What to Hand In

- A file called `a8q4.py`, containing your function definition.
- A file called `a8q4_testing.py`, containing your testing.

Be sure to include your name, NSID, student number, course number and laboratory section at the top of the file.

Evaluation

- 4 marks: Your function works.
- 4 marks: Your testing is adequate.