CS 234, Spring 2024 A. Jamshidpey

ASSIGNMENT 3

Due date: July 10, 2024 16:00 (Waterloo time)

WARNING: To receive credit for this assignment, you checked "I agree" to the academic integrity declaration. Please keep all the rules in mind as you complete your work.

Coverage: Through Module 6

This assignment consists of a written component and a programming component. Please read the instructions on the reference page for assignments carefully to ensure that you submit each component correctly.

Please check the pinned FAQ in the discussion forum for corrections and clarifications.

In this assignment, you will be reasoning about trees and graphs. You might find it helpful to draw pictures as you work through the various problems.

Written component

For full marks, you are expected to provide a brief justification of any answer you provide. For asymptotic notation, informal arguments are sufficient (that is, using formal definitions is not required).

Read each question carefully to make sure you know whether you are playing the role of the user or the role of the provider. You do not need to state the role explicitly, but you should be aware that your answers will be used to assess your understanding of and adherence to the appropriate role.

For any written question that involves an operation that produces multiple values, you can assume that the operation produces an ADT Indexed Sequence of capacity equal to the number of values produced. The index numbers of items will provide additional information in the natural way. For example, for the operation CHILDREN in an ADT Ordered Tree, the indices of the values will correspond to the order of the children. For running time calculations, use the best implementation of the ADT Indexed Sequence found in the course material.

W1. *[10 marks]* We define two nodes in a tree to be *cousins* if their parents are siblings. For example, consider the tree illustrated on page Module 5a under the heading "Tree Terminology". The nodes with labels 9 and 20 are cousins, because their parents (the nodes with labels 8 and 12, respectively), are siblings. (Remember that a node is **not** its own sibling.)

In this question, you will write functions to determine whether a pair of nodes in an ADT Ordered Tree are cousins.

(a) *[3 marks]* Using the ADT Ordered Tree, write pseudocode for the function ARE_COUSINS that consumes an ordered tree O and two distinct nodes *One* and *Two* in O and produces *True* if *One* and *Two* are cousins and *False* otherwise. Notice that since *One* and *Two* are required to be distinct nodes in O, O is guaranteed to have at least two nodes.

In this question, you are the user of the ADT Ordered Tree.

(b) *[3 marks]* Suppose that the ADT Ordered Tree is implemented using linked nodes with pointers *Prev*, *First*, and *Next*, as discussed in the lecture content as Ordered Tree linked implementation 1.

Complete the following steps to determine the worst-case cost of your algorithm from part (a), using two different sets of variables:

- State and briefly justify the worst-case cost in Θ notation as a function of n (the number of nodes in the tree).
- State and briefly justify the worst-case cost in Θ notation as a function of c (the maximum number of children of each node) and h (the height of the tree).

(c) *[4 marks]* In this question, you are considering the provider point of view. Suppose you wished to augment the ADT Ordered Tree with a new operation ARE_COUSINS*(O, One, Two)* that produces *True* if the distinct nodes *One* and *Two* in ordered tree O are cousins and *False* otherwise. Your goal is to ensure that, even if the worst-case running time your algorithm is not better than the running time in terms of n from part (b), there exists at least one "fast" instance for which the running time for (b) is non-constant but the running time for your augmented operation is constant. **The operation should produce *True* on the "fast" instance.**

Briefly describe your algorithm and the "fast" instance. Explain why the running time is non-constant for part (b) for the "fast" instance and why the running time is constant for the augmented operation for the "fast" instance.

Hint: The purpose of this question is for you to see the difference between the user and provider points of view. Make sure you are using the implementation directly, not just trying to copy what you submitted for part (a).

W2. *[12 marks]* Each subquestion specifies ADTs A and B and operation X. Follow the instructions below for each subquestion:

- Provide a description of the implementation of ADT A using ADT B, including any extra variables that are used and the meanings of their values. Do not add any additional ADTs to the implementation.
- For the implementation you provided, give an algorithm for operation X of the ADT A. You do not need to provide pseudocode, but you should refer to specific operations of ADT B that are needed, including specific inputs to the operations.

You can find the details of the ADT operations on the page **Reference material > Indices > ADTs**.

(a) *[3 marks]* A: Bucket; B: Unordered Tree; X: ADD_TO_BUCKET*(B, Data)*

(b) *[3 marks]* A: Pair; B: Binary Tree; X: IS_IN*(P, Data)*

(c) *[3 marks]* A: Indexed Sequence; B: Binary Tree; X: ADD*(I, 5, Data)*

(d) *[3 marks]* A: Unordered Tree ; B: Undirected graph; X: PARENT*(U, Node)*

W3. *[3 marks]* In this question, you will demonstrate your understanding of BFS and DFS by considering the behaviour of one or both algorithms on particular graphs. For each subquestion:

- Describe a graph on n vertices and a starting vertex v for which the statement is true **no matter in what order the neighbours of each vertex are stored**.

- Justify your answer by explaining the behaviour of the algorithm on the input.

For full marks, make sure that you are not choosing a particular value of n (that is, you are describing a family of graphs, for unbounded values of n).

(a) *[1 mark]* At some point during BFS from v, $\Theta(n)$ nodes are gray.

(b) *[1 mark]* At some point during DFS from v, $\Theta(n)$ nodes are gray.

(c) *[1 mark]* At some point during DFS from v, $\Theta(n)$ nodes are white while $\Theta(n)$ nodes are black.

Programming component

Please read the information on assignments and Python carefully to ensure that you are using the correct version of Python and the correct style. For full marks, you are required not only to have a correct solution, but also to adhere to the requirements of the assignment question and the style guide, including aspects of the design recipe.

Although submitting tests is not required, it is highly recommended that you test your code. For each assignment question, create a testing file that imports your submission and tests the code. Do not submit your testing file.

You may import the files `treenode.py` and `collection.py`, described below.

Be sure to read the instructions on the Python requirements page to ensure that you have imported the files as required.

For full marks, make sure to remember when you are playing the role of user and when you are playing the role of provider.

P1. *[25 marks]* In this question, you will write code for a modification of the ADT Unordered Tree. In addition to all of the operations of the ADT Unordered Tree, the ADT HeightTree supports the operation *Height*, which consumes a tree and a node *Node*, and produces the height of *Node* in the tree.

In your implementation, each node in the tree will be implemented using the class `TreeNode`, which has been provided for you in the file `treenode.py`. A `TreeNode` stores a value and a height as well as three pointers. The pointers will be familiar to you from the Ordered Linked implementation 1 of the ADT Ordered Tree, pointing to the previous sibling or parent, next sibling, and first child.

The operation CHILDREN has been defined to return the nodes in an ADT Collection, as defined in Assignment 2.

Both `treenode.py` and `collection.py` linked off of the page **Assessments** > **Assignments** > **Assignment 3**. Remember that you are the user of the classes `TreeNode` and `Collection`; make sure that your code makes use of the provided methods and does not access any of the fields directly.

An ADT HeightTree will be implemented as a variable storing a pointer to the `TreeNode` containing the root.

You should form your code by modifying the interface file for the ADT, `heighttreeinterface.py`, which is linked off of the page **Assessments** > **Assignments** > **Assignment 3**. The file includes headers for all the methods that are required, as well as the method for `height`. Do not change the method for height: the intention of this assignment is to ensure that after the execution of each operation, all height fields have been updated appropriately.

You may wish to create additional methods for your convenience, such as `__repr__` for testing, or various helper functions used in the methods for multiple ADT operations. For this question you are allowed to use operations in the methods for other operations (this is an exception to the general rule stated in the section "Roles in programming questions" on the Python requirements page).

Make sure that in your methods for each of the operations, all of the fields and pointers in each `TreeNode`, including the height field, are updated appropriately.

Submit your work in a file `heighttree.py`.