CS 234, Spring 2024 A. Jamshidpey

ASSIGNMENT 4

Due date: July 24, 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 9

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.

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.

W1. *[6 marks]* In this question, you will consider implementations of the ADT PriorityPlus, which supports all the operations of the ADT Priority Queue as well as the operation DELETE_MAX*(P)*. You can find the pseudocode interface of the operations of the ADT Priority Queue on the page **Reference material** > **Indices** > **ADTs**, and the new operation in Table 1.

For each implementation, **state and briefly justify** the **worst-case** running time of DELETE_MAX in Θ notation as a function of n, the number of (key, element) pairs stored in the priorityplus, and, for hashing, b, the number of buckets. In all implementations, the key (not the element) is used in ordering.

Name	Preconditions	Produces	Effects
DELETE_MAX*(P)*	*P* is an priorityplus that is not empty	a pair such that key has maximum key value	deletes returned pair from *P*

Table 1: Pseudocode interface for ADT PriorityPlus operation DELETE_MAX

(a) *[2 marks]* A heap

(b) *[2 marks]* An AVL tree

(c) *[2 marks]* Hashing with separate chaining

W2. *[8 marks]* Each subquestion specifies ADTs A and B (and possibly C) and operation X. Follow the instructions below for each subquestion:

- Provide a description of the implementation of ADT A using ADT B (and ADT C, if given), 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 (by mentioning their names), including specific inputs to the operations.

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

(a) *[2 marks]* A: Indexed Sequence, B: Dictionary; X: ADD*(I, 3, Data)*

(b) *[2 marks]* A: Ranking; B: Dictionary; X: MAX_RANKING*(R)*

(c) *[4 marks]* A: Dictionary; B: Ranking; C: Pair; X: ADD*(D, Key, Element)*

W3. *[6 marks]* In this question, you will consider implementations of a new ADT, the ADT Double Dictionary. This ADT is similar to an ADT Dictionary, except that each data item consists of a pair of keys, where it is possible to search based on either key. Keys are all distinct.

Read the pseudocode interface in Table 2 carefully to make sure that you understand the definitions of the operations.

Each implementation specifies how data items are organized with respect to the first key as well as how data items are organized with respect to the second key. If both methods of organization use linked memory, you should assume that each linked node stores both keys as well as pointers for both methods of organization. If one method uses contiguous memory and the other uses linked memory, you should assume that each array entry contains a linked node storing the two keys as well as the pointer or pointers for the linked method or organization. In addition, for each linked structure, the implementation will contain one or more variables storing pointers to linked nodes, such as to the root of a tree or the first linked node in a linked list.

Each subquestion specifies an implementation and an operation. Complete the following steps for each subquestion:

- Briefly explain the algorithm for the specified operation using the given implementation. You do not need to provide pseudocode. Do not change the implementation.
- Give the details of a **best-case** input for the operation.
- Provide the **best-case** running time of your algorithm on your input in Θ notation as a function of n, the total number of data items stored at the time of the operation, briefly justifying the running time you have provided.

(a) *[2 marks]* First key: ordered linked list; second key: ordered linked list; operation: DELETE_TWO

(b) *[2 marks]* First key: hashing using separate chaining; second key: unordered linked list; operation: ADD_ONE

(c) *[2 marks]* First key: AVL tree; second key: (2,3) tree; operation: ADD_TWO

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.

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

P1. *[20 marks]* In this question, you will write code for the ADT Double Dictionary, as described in W3, though for simplicity, the operations ADD_TWO, DELETE_ONE, and DELETE_TWO are not required.

Your implementation should adhere to the following requirements:

- For each data item, the first key is a string and the second key is an integer.
- Each data item is stored in a node, implemented in `doubledictnode.py`. A node contains the two keys as well as four pointers:
  - _parent points to the parent, if any, else None
  - _left points to the left child, if any, else None
  - _right points to the right child, if any, else None
  - _next points to the next node, if any, else None
- The first three pointers are used to form a BST based on the first key, and the last pointer forms an ordered linked list based on the second key. In the ordered linked list, the second keys are stored in increasing order.
- The ADT DoubleDict is implemented as two pointers, one to the node storing the root of the BST, and the other to the node that appears first in the ordered linked list.

You should form your code by modifying the interface file for the ADT, `doubledictinterface.py`, which (along with `doubledictnode.py`) is linked off of the page **Assessments** > **Assignments** > **Assignment 4**. The file includes headers for all the methods that are required. Each provides a `repr` and a `tree_repr` method and enough of the class definition to ensure that the two methods use the correct field names.

You may wish to create additional methods for your convenience, such as various helper functions used in the methods for multiple ADT operations.

Do not change the headers of the methods, and do not change the `repr` and `tree_repr` methods, as they will be used in the testing of your code.

Submit your work in a file with the name `doubledict.py`.

Name	Preconditions	Produces	Effects
CREATE*()*		a new empty Double Dictionary	creates a Double Dictionary
IS_EMPTY*(D)*	*D* is a Double Dictionary	*True* if empty, else *False*	
LOOK_UP_ONE*(D, Key_1)*	*D* is a Double Dictionary, *Key_1* is a string	*Key_2* in pair (*Key_1*, *Key_2*) if any, else *False*	
LOOK_UP_TWO*(D, Key_2)*	*D* is a Double Dictionary, *Key_2* is an integer	*Key_1* in pair (*Key_1*, *Key_2*) if any, else *False*	
ADD_ONE*(D, Key_1, Key_2)*	*D* is a Double Dictionary, *Key_1* is a string, *Key_2* is an integer		adds a pair (*Key_1*, *Key_2*) to *D*, replacing a pair with *Key_1* as first key, if any
ADD_TWO*(D, Key_1, Key_2)*	*D* is a Double Dictionary, *Key_1* is a string, *Key_2* is an integer		adds a pair (*Key_1*, *Key_2*) to *D*, replacing a pair with *Key_2* as second key, if any
DELETE_ONE*(D, Key_1)*	*D* is a Double Dictionary, *Key_1* is the first key of a data item in *D*		deletes a pair with *Key_1* as first key
DELETE_TWO*(D, Key_2)*	*D* is a Double Dictionary, *Key_2* is the second key of a data item in *D*		deletes a pair with *Key_2* as second key

Table 2: Pseudocode interface for ADT Double Dictionary