Assignment 3

Advanced Data Structures and Algorithms (EECE 4040) – Spring 2022

Due Friday, March 4 by 11:59PM EST

The leader of your group should submit three files: (1) a scan of the written part; (2) source code for your program; and (3) output from a sample run of your program. For ease of grading, all the C++ code for your program should be included in a ***single*** *file and designed using the C++ Visual Studio Platform. It should be well-commented and the output user-friendly.*

REMINDER. Test 1 will be held on Wednesday, February 23.

Written Part (30 pts)

1. [15pts] In many practical applications such as Prim's algorithm for computing a minimum spanning tree and Dijkstra's algorithm for computing shortest paths, both of which we will see later in this course, the priority queue ADT needs to be expanded to include the operation of changing the priority of an element. Design and give pseudocode as well as analyze the algorithm *ChangePriority*(*Q*,*x*,*v*), which changes the priority value of an element *x* in the priority queue *Q* to *v* when *Q* is implemented as a **min-heap**. *ChangePriority* should have worst-case complexity *O(*log *n*), where *n* is the number of elements in the min-heap.

   Hint. Consider two cases, the priority is decreased and the priority is increased and correct a min-heap property violation in a similar way to what we did for insertion and deletion into a min-heap**.**

2. [15pts] Suppose we have the following parent implementation of a forest representing a partition of a set of 17 elements:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Parent[*i*]	7	–3	8	7	14	0	1	8	–13	3	14	4	–1	3	3	0	1

   a. Sketch the trees in *F*.
   b. Show the state of *Parent*[0:16] after a call to *Union*(*Parent*[0:16],1,8) and sketch the trees in *F*.
   c. Given the state of *Parent*[0:16] in part (b), show the state of *Parent*[0:16] after an invocation of *Find2*(*Parent*[0:16],4) and sketch the trees in *F*.

Programming Project (70pts)

Binary Search Trees and Implementing an Electronic Phone Book

Write a program that provides a way for you to store and retrieve telephone numbers. Design a user interface that provides the following operations:

Add: Adds a person's name (first and last) and phone number to the phone book.

Delete: Deletes a given person's phone number, given only the name.

Find: Locates a person's phone number, given only the person's name.

Change: Changes a person's phone number given the person's name and new phone number.

Display: Displays (dumps) entire phone book in alphabetical order.

Quit: Quits the application, after first saving the phone book in a text file.

You can proceed as follows:

- Design and implement the class `Person`, which represents the last name (first and last) and phone number of a person. You will store instances of this class in the phone book.
- Design and implement the class `Book`, which represents the phone book. The class should contain a binary search tree as a data member, where the key is the person's name (when comparing keys first compare last names; then if last names are the same compare first names. You may assume that no two people have the same first and last names). This tree contains the people in the phone book.
- Output names in alphabetical order (of last names and then first names if last names are the same). **Hint:** Use inorder traversal. Display table with names (first and last) and phone numbers with appropriate title, headers, and formatting.
- Add member functions that use a text file to save and restore tree
- Design and implement the class `UserInterface`, which provides the program's user interface.

Suggestion. Design your program first without saving and restoring from a text file, i.e., start with an empty phone book and perform operations in one session/run (worth 65 points). Once you have this working do the full assignment involving a text file (for the full 70 points).