BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
Second Semester 2014-2015
CS F211 Data Structures and Algorithms

Lab Sheet – 8

Problem

In this lab, you have to write an application that keeps track of traffic fines.
All violations arenoted by a traffic policeman in a file as a record <license number of driver, fine amount>. At the end of each day, files from all traffic policemen are collated.
If a driver had been charged with more than three violations so far, then he has to be booked for further legal action. All such license numbers are to be output in a file called "violators.txt".
Also, the police department provides additional bonus to those policemen who have brought in large fine earnings. All policemen who have collected more than 90% of the maximum fine amount collected by any individual so far, shall be awarded the bonus. The list of policemen eligible for bonus must be output in a file called "bonus.txt".

For this problem, it was decided to use hash tables for keeping track of drivers (and their violations), and an inorder-tagged binary search tree for keeping track of policemen (and their bookings). Note that an inorder tag BST is a binary search tree, where every node will also store its inorder successor, (as was discussed in previous lab.)

Data structures to be used:
DriverHash: A separately chained hash table indexed by license numbers where each entry is of the form < license number, number of violations>. A simple hash function h(x) = x mod M, where M is the size of hash table can be used for this.
PoliceTree: This is an inorder tagged BST of entries of the form <police ID, total fine amount>

Functions used by the application:
The ADT for hashing related operations are provided in DriverHash.h and DriverHashOps.h

1) void initializeHash(DriverHash dh)
This function creates an empty hash table.

2) insertHash(DriverHash dh, License lic):
This function inserts the license number lic to the hash table: if a driver's license number is already present, only the number of violations need to be updated else a new entry has to be created.

3) printViolators(DriverHash dh):
This function prints the serious violators by looking through all hash table entries and printing the license numbers of the drivers who have more than 3 violations onto the file "violators.txt".

4) destroyHash(DriverHash dh):
This function destroys all the entries inside hash table. This is a clean up code.
The ADT for tree related operations are provided in PoliceTree.h and PoliceTreeOps.h

5) PoliceTree insertByPoliceId(PoliceTree root, PoliceId id, int amount):
This function inserts an entry <id, amount> into the police tree ordered by police id. If the PoliceId id is already found in the tree, then this function updates the total amount to be stored against him. This function returns the updated tree.

6) PoliceTree reorderByFineAmount(PoliceTree root):

This function reorders the inorder tagged BST on the basis of fine amount, instead of police id. This function removes the nodes from the original PoliceTree, and puts it in a new tree ordered by fine amount. Note that if the fine amount is equal to the amount in node j, then they will be inserted to the left of the node j. This function returns the root node of the new tree.

7) printBonusPolicemen(PoliceTree root):

This function prints the policemen who have earned more than 90% of largest fine amount collected by an individual on to a file called "bonus.txt".

8) destroyPoliceTree (PoliceTree root):

This function is a clean up function that destroys all the nodes in the police tree.

9) printPoliceTree (PoliceTree root):

This function is meant for debugging purposes. This function prints the contents of the PoliceTree in-order.

Steps to perform:

Note that each function is implemented in a c file by the same name as function. For eg, initializeHash function is implemented in the file named initializeHash.c. The driver file is already been implemented in fine.c.

Implement the functions

1) insertHash in insertHash.c
2) printViolators in printViolators.c
3) insertByPoliceId in insertByPoliceId.c
4) reorderByFineAmount in reorderByFineAmount.c
5) printBonusPolicemen in printBonusPolicemen.c

File formats to be used:

1) input file:

The following entry will be present in each line of the input file.
<police id>,<license number>,<fine amount>

2) bonus.txt:

The following entry will be present in each line of this file.
<police id>

3) violators.txt:

The following entry will be present in each line of this file.
<license number>

Support files provided: History.h, PoliceTree.h, PoliceTreeOps.h, DriverHash.h, DriverHashOps.h, initializeHash.c, destroyHash.c, destroyPoliceTree, printPoliceTree, fine.c.
Deliverables: reorderByFineAmount.c, printBonusPolicemen.c,

Sample Solution is already given for almost all the functions (except 2)

Take Home:

1)

Instead of using an In-order tagged BST, use BST. Is there any difference in tiem while identifying and printing the policemen who should be given bonus?

2)

Instead of using BST, use AVL tree. Is there any difference in execution of various functions.