<u>CS201c: Programming Evaluation 1 Solution Outline</u>

Broad Structure of the solution:

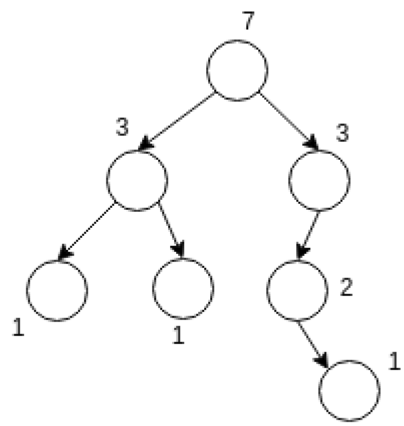
We store a single float *curr*, which denotes the current time.
The solution comprises of 3 BSTs.

- The first BST T1 stores the cars with their registration id as the key. Further, for each registration id r in T1, a pointer to the unique node corresponding to car r in either tree T2 or T3 is also stored.
- The second BST T2 stores information about all the cars traveling in the left direction.
- The third BST T3 stores information about all the cars traveling in the right direction.
- Each node of trees T2 and T3 stores:
    a. Registration id r of the car (as auxiliary data).
    b. The pair (x,t) as the key, which denotes that the position of car r was x, when it was inserted into the data structure at some (past) time t.
    c. Pointer to the parent node.
    d. Pointer to left child
    e. Pointer to right child
- For trees T2 and T3, the comparison between key (x,t) [the key to be inserted] and any other key (x',t') (here t >= t') is as follows:

    (i) (x,t) < (x',t') if and only if (x'+(t-t')) > x,
    (ii) (x,t) > (x',t') if and only if (x'+(t-t')) < x,
    (iii) (x,t) == (x',t') if and only if (x'+(t-t')) = x

    The main point is that if a car was at coordinate x' at time t', then its coordinate x_{new} at (future) time t, is given by the following equation: $x_{new} = x' + (t' - t)$.

Further, we augment both trees T2 and T3 with extra information. With each node v of these two trees, we store the count of all nodes present in the subtree with node v as the root (see figure below):



(covered in class. Also, for reference, see https://en.wikipedia.org/wiki/Order_statistic_tree)

Key point: The ordering of cars moving in the same direction does not change (uniform speed) except at insertion or deletion.

Notes. We use an augmented, balanced BST (such as AVL trees) to implement trees T1, T2, and T3. To avoid repetition of code, make a single balanced BST class template with Z as the name of data class. Instantiate this class template three times with Z equal to relevant classes to obtain trees T1, T2, and T3 (see Practice Lab 2). You will have to appropriately overload comparison operators for your classes.

Implementation of individual functions:

1) *int insert(int r, float x, float t, int d)*:

   To insert a car on the highway with registration id r, with location x at time t in direction d, we perform the following steps:

   (i) We search for r in tree T1. If r is found, insert is unsuccessful, and we stop.0..
   (ii) If r is not found, we insert car r into tree T1. Let the new node created be w.
   (iii) We next check ‘d’ to determine the tree Ti to which the car belongs:

   - If d = 0 -> the vehicle is traveling from right to left and hence will be inserted in the tree T2 i.e., i=2.
   - Else if d = 1 -> the vehicle is traveling from left to right and hence will be inserted in tree T3 i.e., i=3.

(iv) We insert ((x,t), r) in balanced BST Ti (where i = 2 or 3). Let v be the new node created as a result of this insert operation.
(v) We set the pointer field in node w of tree T1 to node v. Also, set *curr* to t.

2) *int delete(int r, float t)*:

(i) Search for registration number r in tree T1. If r is not found, delete is unsuccessful, stop. Otherwise, let w be the node found.
(ii) Follow the pointer field of node w to find corresponding node v in either tree T2 or T3.
(iii) Delete node v from its balanced BST.
(iv) Delete entry for registration number r from tree T1.
(v) Set *curr* to t.

3) *int find_immediate_left(int r, int t):* Using the first BST, we obtain the pointer to node v corresponding to car r in either tree T2 or T3. We next find the inorder predecessor of v in O(log_2(n)) time.

4) *int find_immediate_right(int r, int t):* Using the first BST, we obtain the pointer to node v corresponding to car r in either tree T2 or T3. We next find its inorder successor in O(log_2(n)) time.

5) *int count_left(int r, int t):*

(i) To count_left, we use first BST to find node v corresponding to r in tree T2 or T3.
(ii) We then use the augmented data structure to find the number of nodes less than or equal to v in the tree in O(log_2(n)) time.

6) *int count_right(int r, int t):* Same as count_left, the only difference being that we now find the number of nodes greater than or equal to v.

7) *int number_of_crossings(int r, int t):*
(i) Use the first tree T1 to find node v corresponding to car r in either tree T2 or T3.
(ii) Without loss of generality, assume v is in T2. (The other case is symmetric).
(iii) The only cars that can cross v are traveling in opposite direction i.e., are in tree T3.
(iv) Calculate the current location x_c of car r using the formula x'+(*curr*-t'), where (x',t') is the key at node v, and *curr* is the current time.
(v) Now use the concept of relative velocity. Assume car r is stationary, then the cars in tree T3 are coming at speed 2 towards it. Thus, exactly those cars in T3 which occupy

coordinates in the interval (x_c, x_c+2*(t-*curr*)) will cross car r by time t.

(iv) Since we have augmented tree T3 (and T2) with number of nodes field, the number of keys in a given interval can be obtained in O(log_2(n)) time.