Car Dealership Manager

Noam Chen and Shaked Zameret

May 28, 2021

In this section we will describe our data structure and Algorithms. Our data structure contains 5 AVL trees s.t each node has a key field and a data field. The tree contains a pointer to its root, a pointer to its left most object and a pointer to its right most object. These fields varies between each tree and most likely contain objects which will be discussed in the following sections. A diagram of the structure will be attached at the bottom of this document.

- $tree_1$
  - **key:** type ID
  - **value:** a pointer to a car_type object
- $tree_2$
  - **key:** sales amount of each type's most sold model
  - **value:** a pointer to a car_type object
- $tree_3$
  - **key:** Tree3_Key object
  - **value:** a pointer to a car_model object s.t its rate is no 0.
- $tree_4$
  - **key:** Tree4_Key object
  - **value:** a pointer to another AVL tree which contains only models with a rate that is equal to 0.
- $tree_5$ (explicitly defined as a sub tree of each node in $tree_4$).
  - **key:** model ID
  - **value:** a pointer to a car_type object
- num_of_models (unsigned integer).

Data Types

- **carType:** represents a car type. contains the following fields:
  - type ID
  - nuber of models under this type
  - a pointer to a car model object which represents the most sold model under this type
  - a pointer to an array of car models s.t each of these car models belongs to this car type

- **carModel:** represents a car model under a certain type. contains the following fields:
  - model ID
  - rate (can be negative)
  - sales amount
  - an associative type ID
- **Tree3_Key**
  - rate (can be negative)
  - an associative type ID
  - model ID
- **Tree4_Key**
  - rate (which is always 0 in this tree)
  - type ID
- **Tree4_Data**
  - a pointer to a car_type object
  - a pointer to a sub tree (tree_5 as was told before)

Space Complexity

- tree_1 has $n$nodes s.t each node has an interger field that represents the typeID and k_l elements in the array it points to. By definition:
$$car\,models = m = \Sigma_{l=1}^{n} k_l$$
- tree_2 has $n$elements s.t each element has one element with $O(1)$ space complexity
- tree_3 has $q_1 \leq m$ elements when q_1 is the number of car models with non-zero rate.
- tree_4 has $q_2 \leq m$ elements when q_2 is the number of car types that have registered models with zero rate. The tree contains overall $q_3 \leq m$ when q_3 is the number of nodes in each of tree_4 node's sub tree. each of these nodes has $O(1)$ space complexity (it contyains only pointers). Overall tree_4 contains $q_2 + q_3 \leq m + m = 2m$ nodes
- there are overall $n$car_type objects in the structure.
- there are overall m car_model object in the structure.

  The total number of variabl-es in the data structure is:
$$\#fields(tree_1) + \#fields(tree_2) + \#fields(tree_3) + \#fields(tree_4) + \#(cartype) + \#(carmodel) =$$
  So:
$$= n + n + q_2 + q_3 + n + m \leq n + n + 2m + n + m = 3 \cdot (n + m) = O(n + m)$$
  when $c = 3$. Hence the space complexity for the structure is $O(m + n)$ as requested •

Algorithms

- **void* Init() -** The function creates 4 empty AVL trees (assigning NULL to their fields). Evantually it returns a pointer to the structure. **number of operations:** 5 (creating 4 avl trees and one unsigned integer) $\rightarrow$ **Time Complexity:** $O(1)$,
  number of allocations: $4 \cdot 3(3\,fields\,per\,tree) + 1 = 13 \rightarrow$ **Space Complexity:** $O(1)$.

- **StatusType AddCarType(void *DS, int typeID, int numOfModels) -** At first we will add a node which represents the typeID we got as a parameter into tree_1 (As we saw in the lecture node insertion into an AVL tree lasts $log(n)$ operations at most), then we will initialize an array of m car_model objects with NULL value (that also takes m operations) and 0 value in the sales number for each of one of them. Evantually we will add the node that represents the car type into tree_2(since it has n elements the number of operations is boundedby $log(n)$ operations). We will assign 0 value sales amount in the node we've just added in tree_2 and 0 for the model with maximum sales. We will also add a node that represent the car type into tree_4. tree_5 will be initialized with special algorithm s.t the initialization will perform under $O(m)$ time since m iterations are made. The space complexity for this algorithm will be $O(h)$ when h is the height of the stack which is being created during the recursion process. **number of operations:** $log(n)+log(n)+log(n)+m = 3 \cdot log(n)+m = O(log(n)+m) \rightarrow$ **Time Complexity:** $O(log(b)+m)$**. number of allocations:** m allocations for m nodes $\rightarrow$ **Space Complexity:** $O(m)$.

- **StatusType RemoveCarType(void *DS, int typeID) -** We will search the node in tree_1 that matches the typeID that matches the typeID we got as a parameter (it is bounded by $log(n)$ operations in order to find it), right after we will check what is the max selling model under the same type and we will remove its node from $tree_2$. We will delete all the models under the same type under the following procedure:

  - each model that has a non-zero rate is represented by a node in tree_3, hence we will remove it from tree_3. it is bounded by $log(q_1)$ operations to find the right node in the tree and an additional $log(q_1)$ operations to remove it. In overall it will be bounded by $2 \cdot log(q_1)$ operations when $q \leq M$ is the number of model nodes in tree_3 with a non-zero rate. It will not require any space allocations.
  - if a model that we would like to delete is not located in tree_3 then the model is located in a sub tree of one of tree_4 nodes. we will remove the node from the subtree. it is bounded by $log(q_2)$ operations to find the right model node in the sub tree and by another $log(q_2)$ operations to delete it. after we delete all the nodes from the sub tree of the type node in $tree_4$ we will additionally delete this node from tree_4 (this is bounded by $log(q_3)$ operations when $q_3 \leq n$ is the number of nodes in tree_4).

    In overall the deletion process's operations number is bounded by: (k is the numebr of models with non-zero rate and l is the number of models with zero rate

    $$log(n) + k \cdot log(q_1) + l \cdot log(q_2) + log(q_3) \underbrace{\leq}_{k+l \leq m} log(n) + m \cdot (log(q_1) + log(q_2)) + log(n)$$

    so

    $$log(n) + m \cdot (log(q_1) + log(q_2)) + log(n) \underbrace{\leq}_{(1)} 2 \cdot log(n) + 2m \cdot log(M) = O(log(n) + m \cdot log(M))$$

    (1) $q_1 \leq M$, $q_2 \leq M \underbrace{\rightarrow}_{\text{log is a monotonic and continuous function}} log(q_1) \leq log(M)$, $log(q_2) \leq log(M)$
  - hence **Time Complexity:** $O(log(n) + mlog(M))$**, Total Space Complexity:** $O(1)$ since no allocs were made.

- **StatusType SellCar(void *DS, int typeID, int modelID) -** The functions is used in order to sell a model under the provided model ID which is under the type ID. We start by finding the type node in $tree_1$ (bounded by $log(n)$ operations), then we will refer to the array of models and throughout the array we will find the specific model by its ID (which is also is index). this is bounded by a constant number of atomic operations. We will proceed with comparison in order to check if the current model is the most sold model after the sale, if it is necessarry we will update the pointer in the car_type object to point on it. In this case we will also remove and insert its node from $tree_2$(bounded by $2 \cdot log(M)$ operations). We will proceed and divide the handling into the following cases:
  - the model's rate before the sale was 0 and hence after the sale its rate will be 10, therfore we will find the node that represent its type in $tree_4$ (bounded by $log(n)$ operations) and then we will find the node of the car model (bounded by $log(M)$ operations). We will remove it from the sub tree (bounded by $log(M)$ operations) and then insert it into $tree_3$ (bounded by $log(M) + log(n)$ operations in case that after the deletion the sub tree stays empty which will require to remove the type node from $tree_4$). In overall if we sum all of the operations we get: $log(n) + log(M) + log(n) + log(M) + log(M) = 2 \cdot log(n) + 3 \cdot log(M)$.
  - the model's rate isn't 0:
    - if the new rate is 0 (since the old rate can be negative) it requires to remove the car model node from $tree_3$ and insert it into the subtree of the car_type node (bounded by $log(M)$ for the deletion, $log(n) + log(M)$ for the insertion). In overall we get that the number of operations is bounded by $log(n) + log(M)$ in this case.
    - if the new rate is not 0 then we only need to remove the car model node from $tree_3$ and insert the updated node to $tree_3$ again. In overall we get that the operations number is bounded by $2 \cdot log(M)$.

  In the worst case (in time complexity aspect) we get $2 \cdot log(n) + 3 \cdot log(M)$ operations. hence **Time Complexity:** $O(log(n) + log(M))$ **Space Complexity:** $O(1)$ **(**since no allocations were made).
- **StatusType MakeComplaint(void *DS,int typeID, int modelID ,int t) -** The function is used to create a complaint about a certain car model. The process is very similar to the selling process only that here we do not make any comparison. hence the time complexity stays the same like above. No memory allocations were made so we get in overall: **Time complexity:** $O(log(n) + log(M))$, **Space Complexity:** $O(1)$.
- **StatusType GetBestSellerModelByType(void *DS, int typeID, int *modelID) -** The function returns the following value:
  - if $typeID = 0$ then it returns the most sold model in all the system. Since $tree_2$ contains nodes that each of it represent a most sold car model under a certain type we can find the most sold model in the system by finding the most sold model in the tree (since $tree_2$ is an AVL tree then the right most node in the tree will be that node). We then directly access this node by the right most pointer of $tree_2$. hence **Time Complexity:** $O(1)$, **Space Complexity:** $O(1)$ **(**since no memory allocations were made).
  - if $typeID \neq 0$ then we will find the node that represent the right type_id in $tree_1$ (bounded by $log(n)$) and then access its most sold model field. hence in the worst case: **Time Complexity:** $O(log(n))$**, Space Complexity:** $O(1)$ **(**since no allocations were made).
- **StatusType GetWorstModels(void *DS, int numOfModels, int *types, int *models) -** In this function we will find the m worst models by rate in the system by using a modified InOrder traversal on $tree_3$ and $tree_4$ since a regular InOrder traversal is bounded by n operations in the worst case. We will start traversing from the left most node $tree_3$ (in case it contains models with negative rate) and not from the root like a regular InOrder traversal. Once we get to a node with a positive rate we will move to $tree_4$ to continue iterating the nodes with rate zero (we will start from the left most node and the comparison between the nodes is made by their model_id). In case we have iterated all the nodes and sub trees in $tree_4$ and we got more nodes to iterate we will move back iterating on $tree_3$ by our modified InOrder starting from the last node we have visited in $tree_3$. In overall we have made m iterations, hence **Time Complexity:** $O(m)$, **Space Complexity:** $O(h)$ when h is the maximal height of the stack which was created during the recursion process, $h \leq m$.

- **void Quit(void **DS) -** This function releases the structure's memory. It releases each node of a car model and each node of a car type in the structure (when there is n nodes of car types and m nodes of car models). Therfore **Time Complexity** is $O(m+n)$. **Space Complexity:** $O(1)$ (since no allocations were made).