6.851 Advanced Data Structures (Spring'14)

Prof. Erik Demaine TAs: Timothy Kaler, Aaron Sidford

Problem 6 *Sample solution*

Solution:

We maintain the nodes in a linked list and use the $O(1)$ list order maintenance data structure we saw in Lecture 8 to obtain labels for the elements. Recall that in amortized $O(1)$ time per insertion and deletion the $O(1)$ list order maintenance data structure allowed us to maintain the following

- A "top" summary structure consisting of $O(\log n)$ bit labels such that each "top" label corresponds to at most $O(\log n)$ nodes and such that each node has exactly one top label.
- A "bottom" label consisting of $O(\log n)$ bits for each node

In addition the data structure provides the following guarantees

- Concatenating the top label and the bottom label for each node yields a list labeling of the nodes.
- The amortized number of changes to the top and bottom labels per insertion or deletion is $O(\log n)$

In addition to the $O(1)$ list order maintenance data structure we maintain two van Emde Boas (vEB) structures for the top labels. One stores the top labels for which there is a node that has that label and is black and one stores the top labels for which there is a node that has that label and is red. Furthermore, for every top label we maintain two vEB trees, one storing the black nodes with that top label sorted by bottom label and one storing the red nodes with the top label sorted by bottom label.

To insert and delete we simply update the list order maintenance structure in $O(1)$ amortized time and then update the vEB structure to maintain the desired invariants. Since for every label change there are $O(1)$ updates to the vEB structures and since the vEB structures have keys which are $O(\log n)$ bits, the amortized cost of these vEB updates is $O(\log \log n)$.

To answer a `prev` or `next` query, we simply use the list labeling to find the label of the query node. Then we use the vEB on the top label to find the previous or next top label that has a node of the desired color and then use the vEB for that top label to find the previous or next node. Again, since the keys are all $O(\log n)$ bits this takes $O(\log \log n)$ time amortized.

If we use any of the modifications we saw in class to make the vEB structures use linear space, then since every node is stored in exactly one vEB for its top label and one vEB for its bottom label we see that the space used by the vEB is $O(n)$. Furthermore, since the space of using $O(1)$ list labeling is $O(n)$ we see that the total space usage is $O(n)$ as desired.

Common Mistakes:

On the following page we list some common mistakes observed in the submitted solutions:

- **Using the data values:** Note that the problem statement said that the data being inserted could be arbitrary. Consequently, one cannot assume that data can be compared or even hashed. Therefore to use vEB something else, e.g. list labeling, must be done to get keys.
- **Using ordered maintenance labels directly:** While the amortized cost of an insert or delete using $O(1)$ list labeling is $O(1)$ the amortized number of elements whose label change on an insert is $\theta(\log n)$. To see this, note that $O(\log n)$ top labels change every $O(\log n)$ insertions and every top label corresponds to $O(\log n)$ nodes. Therefore, it is possible that $O(\log^2 n)$ nodes have their label change every $O(\log n)$ insertions. Consequently, the labels from $O(1)$ list labeling cannot be used directly. This is why our solution has separate VEB for the top labels and the bottom labels.
- **Using list labeling directly:** Some students simply noted that if you can use labels of size 2^n to maintain labels in amortized $O(1)$ time. However, if you use vEB on these labels than the time for insertion, deletion, and query will all be $O(\log\log(2^n)) = O(\log n)$ which is prohibitively expensive.