Problem Set 4 Solutions

Due: Wednesday, October 11, 2017 at noon

Problem 4.1 [Dynamic Dictionary with Working-Set Property].

A binary search tree has the \textit{worst-case working-set property} if every access $x_i$ costs $O(\log t_i)$ worst-case time, where $t_i$ is the number of distinct keys accessed since the last access to key $x_i$.

Describe and analyze a dynamic dictionary (not necessarily a BST) that has the working-set property. Your data structure should:

(a) use $O(n)$ space, where $n$ is the current number of items in the dictionary;

(b) support searching for key $x_i$ in $O(\log t_i)$ worst-case time, where $t_i$ is the number of distinct keys accessed since the insertion or last access to the key $x_i$; and

(c) support insertions and deletions in $O(\log n)$ amortized time.

\textit{Hint}: Consider representing your dictionary as a list of binary search trees of increasing size.

\textit{Solution}: Maintain a list of $\log \log n$ balanced binary search trees with the $i$th tree containing $2^{2^i}$ keys. The largest tree will contain all keys in the dictionary and the smaller trees act as “caches” that contain recently accessed keys.

Insertions and deletions are performed by inserting/deleting the key in each of the $\log \log n$ trees. The time to perform an insert/delete operation on the $i$th tree is $O(\log 2^{2^i}) = 2^i$. The total time to perform an insert/delete operation, therefore, is $\sum_{i=0}^{\log \log n} 2^i$ which is $O(\log n)$. The size constraints on each of the trees are maintained via a least-recently used eviction policy. Each tree has an associated BST which contains its keys sorted by their last access time. When inserting a key into a full tree the least-recently accessed key in that tree is found and deleted.

To search for a key $x_i$, we query each of the $\log \log n$ trees in order of increasing size until the key is found or it has been determined that the key is not in any of the trees. If the $j$th tree was the first tree in the list containing $x_i$, then we insert $x_i$ into the $j-1$ smaller trees. The cost of searching the $j$th tree for $x_i$ dominates the cost of inserting $x_i$ into the $j-1$ smaller trees. The total time required to lookup the key $x_i$, therefore, is $\sum_{i=0}^{j} 2^i$ which is $O(2^j)$.

Suppose that $t_i$ elements were accessed since the last search for key $x_i$. These elements plus $x_i$ will be contained in the $\lceil \log \log (t_i + 1) \rceil$th tree which contains the $t_i + 1$ most recently accessed keys. The total time to search for the key $x_i$, by our previous argument, is bounded by the time to search the $\lceil \log \log (t_i + 1) \rceil$th tree which is $O(\log t_i)$.

Finally, we note that we can grow the data structure in linear time by appending a new tree to the list whose contents are copied from the old largest tree. Shrinking the data structure consists of just deleting the largest tree, provided all keys are already present in the second largest tree. Various resize criteria work without effecting the amortized bounds — for example the dictionary can grow whenever its capacity $N < n^2$ and can shrink whenever $N^4 > n$. This gives $O(\log n)$ amortized time bounds for the insert and delete operations. It is also possible to deamortize, but this was not required of student solutions.