6.897 Advanced Data Structures (Spring'05)

Prof. Erik Demaine TA: Mihai Pătraşcu

Problem 5 – Solution

Choose a constant c such that $\max\{t_u, S\} = O(n^{c-1})$. We consider $\Theta(n^{1/c})$ "summary" elements, which are spaced relatively evenly, i.e. there are $\Theta(n^{1-1/c})$ elements of the set between two consecutive elements of the summary. We construct a data structure for the summary elements, and then we recurse for the intervals between every two summary elements, as well as the two extremes (before the min and max in the summary). When we're down to $O(1)$ elements we use some brute-force solution. Note that elements in the summary do not appear in lower levels – we recurse strictly between summary elements.

The space is given by $S'(n) = S(n^{1/c}) + n^{1/c} \cdot S'(n^{1-1/c}) = n^{(c-1)/c} + n^{1/c} \cdot S'(n^{1-1/c})$. In the recursion tree, the bottom level dominates, and it has cost just $O(n)$, so $S'(n) = O(n)$. Given a set of n elements in order, we can construct our recursive structure in time $O(n)$ – the recursion for the construction time is the same, with t_u instead of S.

A query can query the data structure in the root node, and then recurse in the appropriate interval. We keep the closest predecessor seen in the summaries from above, so that we can return that value if we find that our seach key is smaller than every element in the recursive set. The running time is $t'_q(n) = t'_q(n^{1/c}) + t'_q(n^{1-1/c})$, because we only recurse in one subtree. We have $O(\lg \lg n)$ levels, because $\lg n$ decreases by a factor of $1 - \frac{1}{c}$ each time, so $t'_q(n) = T \cdot O(\lg \lg n)$. If $T = \lg^\alpha n$, we actually have

$$t'_q(n) = \left(\frac{\lg n}{c}\right)^\alpha + t'_q(n^{1-1/c}) = \left(\frac{\lg n}{c}\right)^\alpha + \left(\left(1 - \frac{1}{c}\right)\frac{\lg n}{c}\right)^\alpha + \left(\left(1 - \frac{1}{c}\right)^2 \frac{\lg n}{c}\right)^\alpha + \dots$$

So the query time is given by a geometric with rate $(1 - \frac{1}{c})^\alpha$, and therefore $t_q(n) = \Theta(\lg^\alpha n)$.

Now we need to discuss how to maintain the structure dynamically. Insertions are propagated recursively. However, we need to maintain that the elements of the summary are relatively evenly spread. Say the number of elements between two summary values (in some arbitrary node, not necessarily the root) was initially $X = \Theta(n^{1-1/c})$. If after a number of insertions, it reaches $2X$, we take the median, and insert it into the summary. This requires rebuilding the summary, in time $O(n^{(c-1)/c}) = O(n^{1-1/c})$. Also, we rebuild the two new intervals recursively. As shown above, this takes time linear in the size of the intervals, i.e. $O(n^{1-1/c})$. Thus, a split costs us $O(n^{1-1/c})$, and this can be amortized because X doubled. The cost of an insertion is given by the cost to search for that element (so that we know where to insert it), plus the increase in potential that covers future rebuildings. An element increments some X on every level, so the total amortized cost is $O(\lg \lg n)$, which is less than the search cost.