6.897 Advanced Data Structures (Spring'05)

Prof. Erik Demaine TA: Mihai Pătraşcu

Problem 7 – Solution

We can assume integers are distinct. Otherwise, we build a hash table, and see how many times each integer appears. This can be done in linear time (n lookups, and at most n inserts). Then, we sort the set without duplicates, and we can reconstruct the multiplicity at the end (by lookups in the hash table again).

To sort n integers of w bits, we proceed as follows. First, we break each integer from the set, $x \in S$, in two parts: $hi(x)$ and $lo(w)$. Let $hi(S) = \{hi(x) \mid x \in S\}$. We keep a hash table which represents $hi(S)$. Each entry $h \in hi(S)$ has as associated data $data(h) = \min\{x \in S \mid hi(x) = h\}$. This is easily constructed: we scan all x's; if $hi(x)$ is not in the hash table, we insert it with x as the minimum; otherwise, we see if the minimum for $hi(x)$ needs to be updated.

Now, we build a set S' of $\frac{w}{2}$-bit integers. The set $S' = hi(S) \cup \{lo(x) \mid x \in S, data(hi(x)) \neq x\}$. We can eliminate duplicates in S' through hashing. Observe that S' has at most n values, and it can be generated in linear time: for each element x, either $data(hi(x)) \neq x$, in which case we add $lo(x)$ to S', or $data(hi(x)) = x$, in which case we add $hi(x)$ to S'.

Now we sort S' recursively. After we have S' sorted, we reconstruct the sorted order for S in linear time. First, for all $h \in hi(S)$, we now want to store in the hash table an associated value which is a linked list of elements $\{x \in S \mid hi(x) = h\}$, sorted by x (which is equivalent to sorted by $lo(x)$). The first element is $data(h)$. To generate the other values, we first create a hash table for $lo(S)$, in which each l stores a linked list (in arbitrary order) of $\{x \in S \mid lo(x) = l, x \neq data(hi(x))\}$. Then, we traverse S' in order, and for each $t \in S'$ scan the list associated with t in the hash table for $lo(S)$. For each x in the list, append it to the list for $hi(x)$. For each list in $hi(S)$, the elements are being appended by $lo(\cdot)$, so each list ends up sorted. Then, we traverse the sorted S' again and we append the sorted lists corresponding to each high value, in order, which gives the sorted order of S.

The recursive sorting stops when $w = \lg n$ (this is the original n, not the current size of S, which may be smaller after removing duplicates). At that point, we can sort in linear time, by marking in an array of size n, which is an additive $O(n)$ in our running time. We do $\lg \frac{w}{\lg n}$ steps of the recursion, and we have $O(n)$ cost at each level, so we get the desired running time.