6.897 Advanced Data Structures (Spring'05)

Prof. Erik Demaine TA: Mihai Pătraşcu

Problem 5 *Due: Monday, Mar. 7*

Timing: This problem depends on Thursday's lecture.

Consider the predecessor problem. Let t_q be the amortized time for one predecessor query, t_u be the amortized time for one update (insert or delete), and S be the space of the data structure.

In class, we discussed the idea of bucketing elements, and using a balanced binary search tree inside each bucket. This idea shows that we can convert a solution with parameters

$$\Big[t_q = O(\lg w),\ t_u = O(\text{poly}(w)),\ S = O(n \cdot \text{poly}(w))\Big]$$

into a solution with parameters:

$$\Big[t_q = O(\lg w),\ t_u = O(\lg w),\ S = O(n)\Big]$$

This problem achieves an even more impressive result when all bounds are in terms of n.

Prove the following: Given a data structure for the predecessor problem with parameters

$$\Big[t_q = O(T),\ t_u = O(\text{poly}(n)),\ S = O(\text{poly}(n))\Big]$$

one can construct a data structure with parameters

$$\Big[t_q = O(T \cdot \lg\lg n),\ t_u = O(T \cdot \lg\lg n),\ S = O(n)\Big]$$

Furthermore, if $T = \lg^{\alpha} n$, where $\alpha > 0$ is a constant, the new data structure has parameters

$$\Big[t_q = O(T),\ t_u = O(T),\ S = O(n)\Big]$$

Note: You need only discuss insertions. Deletions can be handled, but they are a bit trickier.

Thus, for any polynomial space and update times, we can reduce the space to linear, and the update to match the query time, while only blowing up the query complexity by $O(\lg\lg n)$. If the query complexity is not too low, i.e. it is $\lg^{\Omega(1)} n$, we don't even change the query complexity at all! So for bounds in terms of n, space and update times essentially don't matter.

Hint: Consider a tree with branching factor n^{ε}.