6.851 Advanced Data Structures (Spring'10)

Prof. Erik Demaine Dr. André Schulz TA: Aleksandar Zlateski

Problem 9 *Due: Thursday, Apr. 15*

Be sure to read the instructions on the assignments section of the class web page.

Link-cut tree analysis.

1. For the bound of the number of preferred child changes we used the heavy-light edge decomposition of the forest. We showed that the preferred child changes during the access operations happen only $O(\log n)$ time per operation (amortized). In the analysis we counted all light preferred edge creations ($O(\log n)$ per access). Then we argued that the total number of heavy preferred edge creations is smaller than the total number of light preferred edge creations plus n. Check if the link/cut operations interfere with this counting scheme. In particular, we might create to many light preferred edges during a link or cut. Can we charge these costs to the link/cut operations?

2. In the final analysis we used the access theorem for splay trees, with

   $$\text{size}(v) = \text{ number of nodes in } v\text{'s subtree,}$$

   as weights. This allows us to analyze all splay costs for the access operations, since in this case we only change weights in one splay tree. However, link/cut operations might affect the weights more globally. In order to apply the access theorem we have to show the the splay potential is not getting too large. Give upper and lower bounds for the splay potential $\Phi = \sum_v \log(\text{size}(v))$ during the execution of the algorithm. Show how we can charge the costs of the $\Phi_{max} - \Phi_{min}$ to the link/cut operations.

Dynamic partition of $[n]$ into intervals. Construct a data structure that dynamically maintains a partition of the set $\{1, \ldots, n\} := [n]$ into intervals. This means that every set of the partition consists of consecutive numbers $\{a, a+1, \ldots\}$. For every interval we pick one of its elements as its *name*. The following operations have to be supported by the data structure:

- `name(`x`)`: returns the name of the interval containing x.
- `merge(`x`)`: merges the interval containing x with immediately following interval.
- `divide(`x`)`: subdivide the interval I containing x, into $\{y \in I \mid y \leq x\}$ and $\{y \in I \mid y > x\}$.

All operations should work in $O(\log \log n)$ time.