Persistent data structures

The first lecture is about “persistence” (which corresponds to the “branching universe” model of time travel). On the one hand, we'd like to remember all past versions of our data structure (“partial persistence”). On the other hand, we'd like to be able to modify past versions of our data structure, forking off a new version (“full persistence”). Surprisingly, both of these goals are achievable with only a constant-factor overhead, in a fairly general model called the “bounded-degree pointer machine.” A bigger challenge is the ability to merge different versions into one version, which can e.g. allow you to double your data structure's size in one operation (“confluent persistence”). It's still unsolved whether this is possible with small overhead, but I'll describe what's known.

[Details and references]

This lecture overviews the nine subjects of the course: integer and string data structures, persistent and dynamic data structures, data structures that takes memory-hierarchy into account and data structures that uses a minimal amount of space, the problem of whether there exists an optimal binary search tree and the studying of hashing, and geometric data structures. The first lecture covers persistent data structures. First we introduce the pointer machine model, and four levels of persistencies: partial, full, confluent, and functional. Then we show that any pointer-machine DS with no more than $O(1)$ pointers to any nodes (in any version) can be made partially persistent with $O(1)$ amortized multiplicative overhead & $O(1)$ space per change [Driscoll, Sarnak, Sleator, Tarjan - JCSS 1989]. Then we show that this applies to full persistence as well [Driscoll, Sarnak, Sleator, Tarjan - JCSS 1989] with the help of an order file maintenance data structure [Dietz & Sleator - STOC, 1987]. We briefly mention a deamortization technique for partial persistence [Brodal - NJC 1996] with its full persistence-version remaining open. For conflict persistence, in the general transformation scenario, given a version v in the version DAG, we define $e(v)$ = 1+ log(#paths from root to v), which measures the version DAG's deviation from a tree. Confluent persistence with $O(e(v))$ per operation is possible [Fiat & Kaplan - J.Alg 2003]. A data structure with $O(\lg n)$ or lower cost per op remains open. For disjoin transformation, $O(\lg n)$ overhead is possible [Callette, Iacono, Langerman - SODA 2012]. For functional persistence, we show a data structure for balanced BST with $O(\lg n)$ per op [Okasaki-book 2003], a data structure for link-cut tree with the same bound [Demaine, Langerman, Price], one for deques with concatenation in $O(1)$ per op [Kaplan,Okasaki, Tarjan - SICOMP 2000] and update and search in $O(\lg n)$ per op [Brodal, Makris, Tsichlas, - ESA 2006], and one for tries with local navigation and subtree copy/delete and $O(1)$ fingers maintained to

"present" [Demaine, Langerman, Price]. What's beyond the scope of this lecture includes a log factor separation for functional persistence from pointer machine [Pippinger - TPLS 1997], the open problem of proving bigger separation for both functional and confluent persistence, general transformations for functional and confluent persistence, lists with split and concatenate, and arrays with copy and paste.

6.851 Advanced Data Structures

Spring 2012