Weekplan: Fully Persistent Data Structures

Inge Li Gørtz

References and Reading

[1] Topics in Data Structures, section 5.5, G. F. Italiano and R. Raman.

We recommend reading [1] in detail before the lecture (especially the proof of Thm 5.17).

The lecture will be about fully persistent data structures and the tree color problem.

Exercises

1 Fat node method Show the data structures used when we implement full persistence using the fat node method for the following example. We have an array of length 4. Initially in version a, all values in the array are 0. Update(v, i, k) updates version v, index i to value k. Consider the following updates:

Update(a,0,5), Update(a,2,4), Update(b,1,3), Update(d,0,8), Update(e,3,2), Update(b,1,3), Update(d,2,1), Update(h,3,7), Update(c,0,7), Update(j,3,7), Update(h,2,7), Update(c,1,7),

1.1 Draw the version tree and indicate for each node which index that was updated (you can give each index a color).

1.2 Construct the version list

1.3 Construct the color lists for each index/color and indicate which elements that are inserted in the predecessor data structure for that list.

1.4 Show how to perform the queries: access(h,0) and access(h,3).

1.5 Show how the version tree, version list, and color lists look after the performing the update Update(h,0,3).

2 Fully persistent heaps Describe an implementation of a partially persistent binary heap. The heap should support the operations find-min, extract-min and insert.

2.1 A binary heap can be represented either with pointers as a tree or as an array. Describe how to make it fully persistent in each case.

2.2 What are the time and space complexities for each of the operations?

2.3 Suppose you perform n insert operations on an initially empty fully persistent binary heap. What is the total time and space usage?

3 Making amortized data structures persistent Why does the slowdowns for fully persistent data structures not hold when the update and query times of the ephemeral data structure are amortized?

4 Lastcolor In this problem you are given a tree with n nodes, where each node has a color from a set of colors C. We want a data structure that supports the following query:

- LastColor(u,v, c): Return the highest node with color c on the path from u to v, where we always assume that u is an ancestor of v.

Give a data structure that can answer LastColor queries in $O(\log\log n)$ time and $O(n)$ space.