Persistent Data Structures – Correctness and Time/Space Analysis Notes

Partial Persistence

read

Start with the access pointer corresponding to version *v*. The access pointer points to the root node containing mod with smaller version closest to *v*. Read the fields of the root node by applying all its mods with version ≤ *v*, sequentially in increasing order. Follow the pointers in the fields to the next nodes and repeat the reading procedure.

=> given a version *v*, the partially persistent structure is read with O(1) overhead per node

write

A mod is added to a node. If the node has free mod space, add the mod. Else, we make a new node:

1) *New node*. Make an empty node. Read the old full node by applying all mods in the old full node to its fields and enter the resulting values in the fields of the new node. Apply the current mod to a field in the new node. The mod space of the new node remains empty. The old node remains full and unchanged.

2) *Backpointers of new node.* Copy backpointers from the old full node to the new node. Follow copied backpointers. Add a mod to each reached node. The mod includes the new version, the index of the field that previously had the pointer to the old full node, and a pointer to the new node. A reached node may be full. The procedure is recursive upto the root node that may also be "copied".

=> all pointers leading to the old full node from access pointers corresponding to versions in the old full node, are preserved

3) *Field pointers of new node*. Follow each pointer of the new node and change the backpointer of each reached node. The backpointer now points to the new node instead of the old full node. No recursion necessary.

=> old full node is locked from recursive updates in the future

If indegree (recursive updates through backpointers) is bounded by a constant in the ephemeral structure, then the ephemeral structure is made partially persistent by the above construction, such that each update in the partially persistent structure only requires amortized O(1) factor overhead and O(1) space (both amortized).

Full Persistence

read

In partial persistence, versions are linearly ordered. A node stores mods corresponding to a subset of the global set of versions. To read the fields of a node, we apply its mods in increasing version order (version numbers may not be consecutive).

In full persistence, versions form a tree. To read the fields of a node at version *v,* we determine which mods in the mod space of the node have versions that are ancestors of *v* in the global tree of versions. Because the timestamp of ancestor versions monotonically increases along the path to *v* in the global version tree, mods with ancestor versions are applied to the fields of the node in increasing version order.

Order maintenance data structure, provides O(1) insertion for maintaining the global version tree and O(1) relative order query for determining version ancestors of *v* among versions in a node.

=> given a version *v*, the fully persistent structure is read with O(1) overhead per node

write