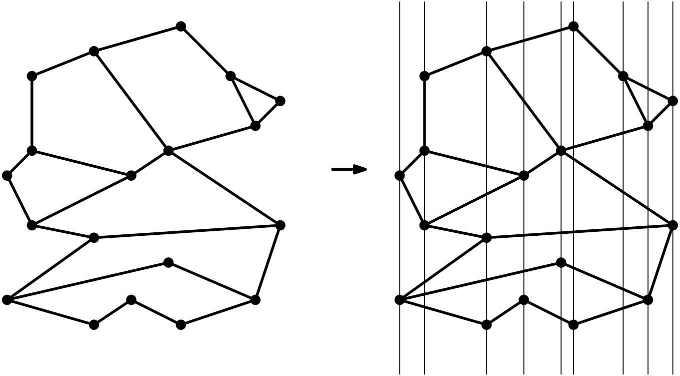
Point Location Using a Persistent Search Tree

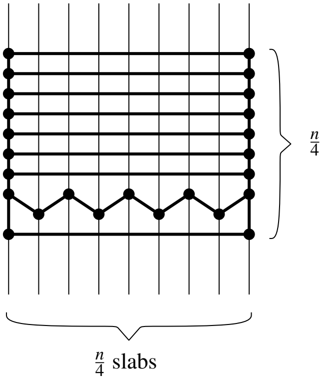
Elisha Sacks

Slab Decomposition Revisited



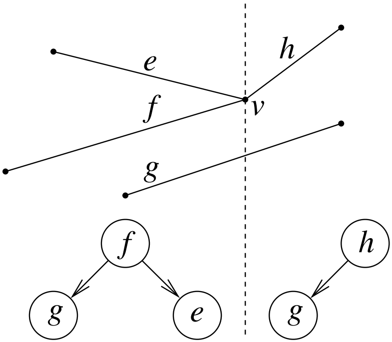
- Sort the subdivision vertices along the x axis.
- Build a binary search tree for the edges in each slab.
- Find the face that contains a point p.
  1. Find the slab of p by binary search on the vertex x coordinates.
  2. Search its binary tree for the edge directly above or below p.

Complexity Revisited



- Subdivision has n edges.
- Good point location time: $O(\log n)$.
- Bad space complexity: $O(n^2)$.
- Bad preprocessing time: $O(n^2 \log n)$.

Complexity Reconsidered



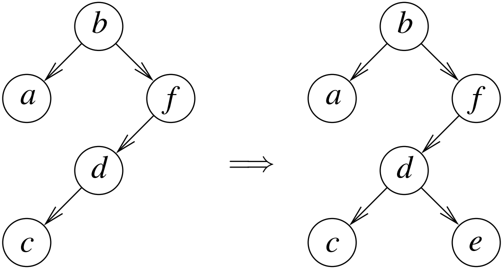
- The binary tree for the slab with left vertex v is obtained from the tree with right vertex v by deleting the edges with right vertex v then inserting the edges with left vertex v.
- The total number of insertions and deletions is at most $2n$.
- A persistent binary tree encodes the trees of all the slabs in $O(n)$ space and supports search in any tree in $O(\log n)$ time.

Persistent Binary Trees

- Insertion creates one vertex and changes one edge.
- Deletion removes one vertex and changes at most four edges.
- Storing these changes takes a constant amount of space.
- Search can use the stored data with constant time overhead.

Planar Point Location Using Persistent Search Trees, Sarnak and Tarjan CACM 29(7):669–679, July 1986

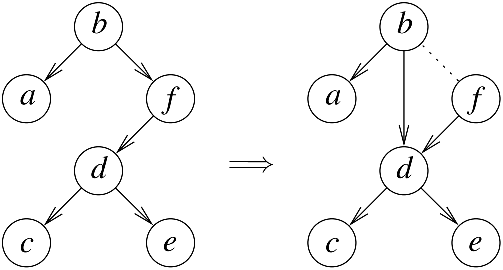
Insertion



Insert e into a binary tree

1. search for e ending at a vertex d
2. make e a child of d

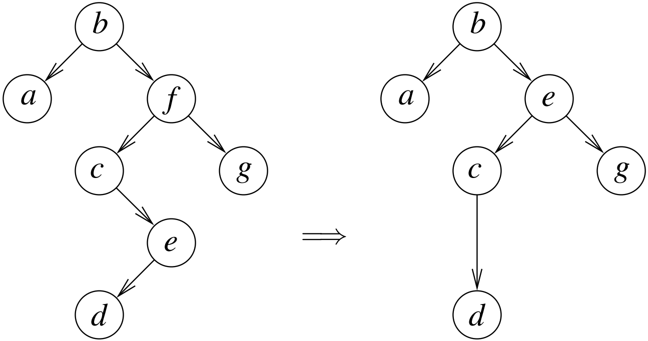
Deletion



Delete f from a binary tree

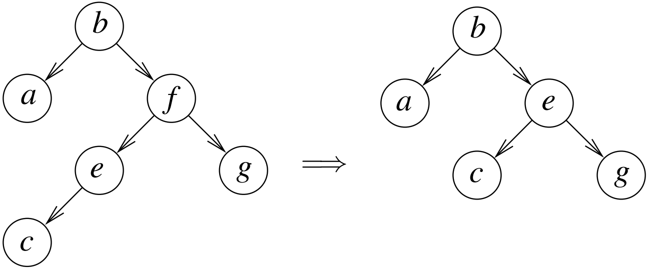
1. find the f vertex in the tree
2. if f has a null child
   2.1 if f is the root
         make its other child the root
       else redirect the parent of f to the other child of f
   2.2 return

Binary Tree Deletion



3. find the node e that precedes f (e has no right child)
4. redirect the parent of e to the left child of e
5. if f is the root
   make e the root
   else redirect the parent of f to e
6. set the children of e to the children of f

Special Case

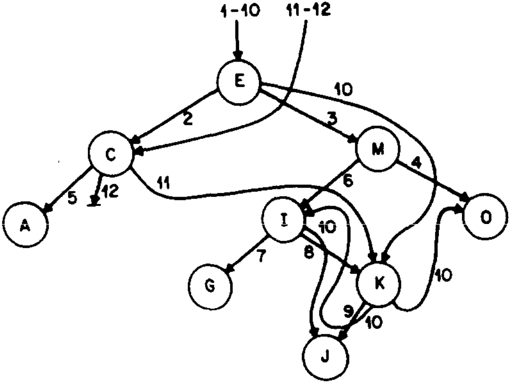


6. if e is the left child of f
   set the right child of e to the right child of f.

Fat Node Method

- Insertions, deletions, and queries have time stamps.
- Insertions and deletions occur in increasing time order.
- Queries occur at arbitrary times.
- A node has a fixed value and a list of time stamped children.
- The tree has a list of roots with time stamps.
- Insertion and deletion append nodes to these lists.
- Memory cost for n insertions and deletions is $O(n)$.
- Tree traversal at time t uses the root and nodes with the largest stamp $s \leq t$; the result is null if there is no $s \leq t$.
- Finding the root or the successor takes $O(\log n)$ time.
- Insertion, deletion, and query take $O(\log^2 n)$ time.

Example

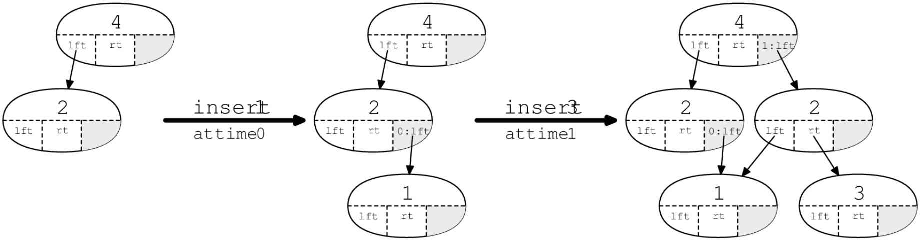


Insertion of E at time 1, C at time 2, M at time 3, 0, A, I, G, K, and J then deletion of M at time 10, E at time 11, and A at time 12. Pointers are labeled with their time stamps.

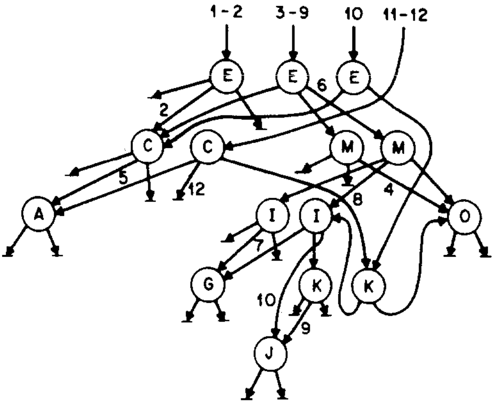
Vertex Copy Method

- A vertex stores a value, two children, and a change box.
- A change box stores a time stamp and a left or right child.
- The change box child overrides the vertex child when its time step is less than that of the traversal.
- The first change to a vertex is stored in its change box.
- The next change is processed as follows.
  1. Copy the vertex using the change box to set one child.
  2. Perform the change directly on the copied vertex. The change box of the copied vertex is empty.
  3. Change the parent of the vertex recursively.
  4. If the root is copied, add the new root to a list of roots.
- Memory cost is $O(n)$ and operation time is $O(\log n)$.
- Red/black tree balancing is ok because only the latest colors are needed.

Example



Larger Example



Insertion of E, C, M, 0, A, I, G, K, and J at times 1–9 then deletion of M, E, and A at times 10–12.

Change box links are labeled with the time step where they are created.