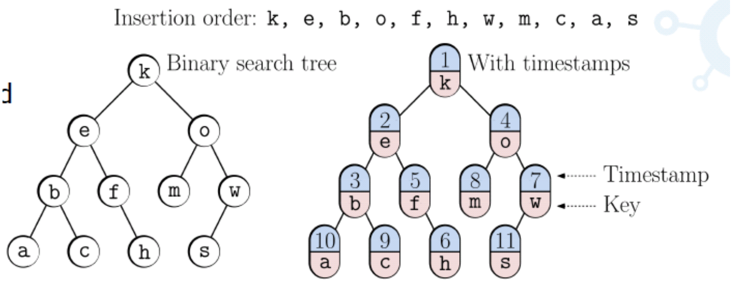
Treaps

Average vs. Expected Time

- We have seen many data structures with good average case performance on random inputs, but bad behavior on particular inputs. E.g. Binary Search Trees.
- Instead of **randomizing** the input (since we cannot!), consider **randomizing** the data structure
  - No bad inputs, just unlucky random numbers
  - Expected case good behavior on any input
- Deterministic with good average time
  - If your application happens to always (or often) use the "bad" case, you are in big trouble!
- Randomized with good expected time
  - Once in a while you will have an expensive operation, but no inputs can make this happen all the time

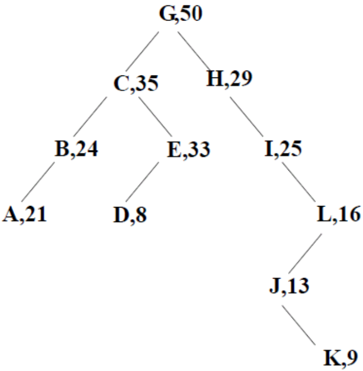
Treaps

- Treap - A binary tree that behaves "as if" keys were inserted in random order
- Invented by R. Seidel and C. Aragon in 1989.
  "Treap = Tree + Heap "
- Nodes in a treap contain both a key, and a priority (timestamp)
- A treap has the BST ordering property with respect to its keys, and the heap ordering property with respect to its priorities



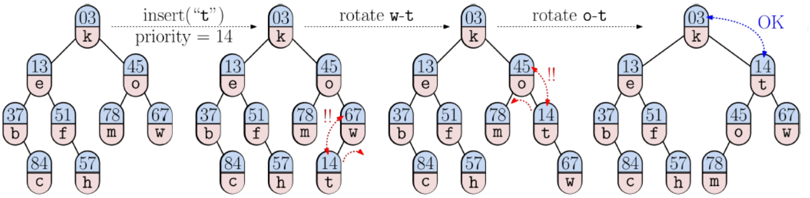
Treaps

- If the priority values as well as the key values are unique, the treap containing the (key,priority) pairs is unique.
- For example, what is the treap containing these pairs: (G,50),(C,35),(E,33),(H,29),(I,25),(B,24),(A,21),(L,16),(J,13),(K,9),(D,8)?
- In this example, larger the number, higher the priority



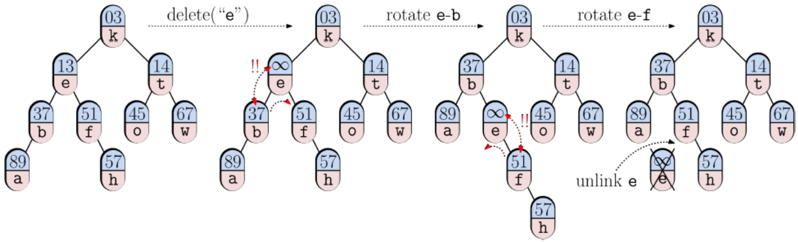
Treap Insertion

- Apply the standard insertion process - create node where we fall out of tree
- Assign a random priority value to the new node
- Apply rotations up the tree until it is in proper heap order



Treap Deletion

- Find the node to be deleted
- Set its priority value to ∞
- Rotate it down to the leaf level and unlink

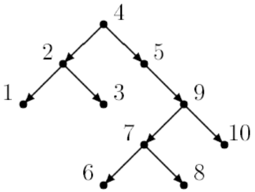


Treap Performance

- Implements Dictionary ADT
  - insert in expected $O(\log n)$ time
  - delete in expected $O(\log n)$ time
  - find in expected $O(\log n)$ time
  - but worst case $O(n)$
- Memory use $O(1)$ per node about the cost of AVL trees
- Very simple to implement little overhead – less than AVL trees

Analysis

- For key set = $\{1, 2, ..., n\}$, priority assignment can be thought of as a permutation of $\{1, 2, ..., n\}$
- Let $m_{\leq} = \{1, 2, ..., m\}$ and $m_{\geq} = \{m, m+1, ..., n\}$
- Let A be the set of ancestors of m, including m itself. Let random variable X = length of the path from the root down to m = $|m_{\leq} \cap A| + |m_{\geq} \cap A| - 2$.
- E.g., $n = 10$, $m = 8$, $\sigma = (4, 5, 9, 2, 1, 7, 3, 10, 8, 6)$.



- W.r.t. $m_{\leq}$, scanning from left to right and checking only k that are $>$ to the left of k, we have $(4, 5, 7, 8)$. Likewise, W.r.t. $m_{\geq}$, we have $(9, 8)$.

Analysis

- H_m, the number of checks obtained when scanning a random permutation σ of $\{1, 2, ...m\}$ from left to right and checking every element that is greater than anything to its left.
- Claim:

$$E(H_m) = \sum_{k=1}^{m} \frac{1}{k}$$

- Observation:
  - Key 1 is checked iff it occurs first in σ, which has prob = $\frac{1}{m}$
  - Let σ' is σ without 1. The number of "checks" on keys other than 1 in σ and σ' are identical.
  - E.g. $\sigma = (4, 5, 9, 2, 1, 7, 3, 10, 8, 6)$ and $\sigma' = (4, 5, 9, 2, 7, 3, 10, 8, 6)$
  - $$E(H_m) = E(H_{m-1}) + \frac{1}{m}$$
- Hence, $E(H_m) = O(\log m)$