Skip Lists - Description: (1)

- Data structure for efficient realization of the *Ordered Dictionary*.
- Makes random choices in arranging the items
  - *Average* Search and Update times: $O(\log n)$
  - Doesn’t depend on the probability distribution of the keys!

Skip Lists - Description: (2)

- Random Number Generator for Insertions:
  - Help decide where to place a new item.
  - Data structures and algorithms utilizing *randomization* are usually simple and efficient!

Skip Lists – Random Numbers (1)

- Extensive use for Random Numbers:
  - Cryptography, computer simulations, computer games…
  - Usually, not really random! But rather, *pseudo-random*.
    - Generates random-like numbers.
    - Good enough for most situations!
Skip Lists – Random Numbers (2)
- Random Numbers with Java:
  - Math.random()
  - Java.util.Random()

Skip Lists – Definition: (1)
- Skip List $S$ for Dictionary $D$:
  - Consists of a series of lists $\{S_0, S_1, S_2, \ldots, S_h\}$.
  - Each list $S_i$ stores a subset of the items of $D$:
    - Sorted by non-decreasing key
    - Items with two special keys: $+\infty$ and $-\infty$
    - $-\infty \rightarrow$ less than every possible key $k$ to be inserted in $D$.

Skip Lists – Definition: (2)
- $+\infty \rightarrow$ greater than every possible key $k$ to be inserted in $D$.
- Each $S_i$ also satisfies the following:
  1. List $S_0$ contains every item of $D$ and $-\infty$, $+\infty$
  2. For $i = 1 \ldots h$, $S_i$ contains a random generated subset of items in List $S_{i-1}$ & $-\infty$, $+\infty$
  3. List $S_h$ contains only $-\infty$, $+\infty$

Skip Lists – Definition: (3)
- 2D collection of positions arranged horizontally into levels and vertically into towers.
Skip Lists – Definition: (4)

- Explanation:
  - $S_{i+1}$ contains approx. every other item in $S_i$.
  - Items in $S_{i+1}$ are chosen randomly from $S_i$.
    - Probability of $\frac{1}{2}$ for each item.
  - In general: $S_i$ contains approx. $\frac{n}{2^i}$ items.
    - Height $h \rightarrow \log n$

Skip Lists – ADT

- ADT Specialized Methods:
  - $after(p)$: Return position after $p$ on same level.
  - $before(p)$: Return position before $p$ on same level.
  - $below(p)$: Return position below $p$ on same tower.
  - $above(p)$: Return position above $p$ on same tower.

Return null if no position!

Skip Lists – Searching: (1)

- $SkipSearch$ Algorithm:

<table>
<thead>
<tr>
<th>Input: Search key $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Position $p$ in $S$ s.t item at $p$ has largest key $\leq k$</td>
</tr>
</tbody>
</table>

  Let $p$ be top most, left position of $S$:
  
  ```
  while below(p) ≠ null do
    p ← below(p) {drop down}
  while key (after(p)) ≤ k do
    p ← after(p) {scan forward}
  return p
  ```

Skip Lists – Searching: (2)

- $SkipSearch$ Algorithm:
  - Takes key $k$ and finds item in $S$ with largest key $\leq k$ (possibly $-\infty$).
  - Begin at top most, left position in $S$ ($-\infty$) call it $p$:

  1. If $S$.below($p$) = null then done – located largest key $\leq k$ in $S$. otherwise drop down a level: $p ← below(p)$. 

Skip Lists – Searching: (2)

2. Otherwise, if $S.after(p) \leq k$ then $p \leftarrow after(p)$. When after(p) > key, go back to step 1.

Skip Lists – Insertion: (1)

- Insertion:
  - Randomization to decide how many references to new item (k, e) to add.
  - Perform SkipSearch(k)
    - Gives position $p$ of bottom level item with key $\leq k$.
  - Insert new item right after $p$ on same level.

Skip Lists – Insertion: (2)

- Call method random - returns number between 0 and 1.
  - If return is $\leq \frac{1}{2}$ add copy of new item one level up.
  - If return > $\frac{1}{2}$ stop – do not insert any more copies!
  - This process creates the tower.

Skip Lists – Insertion: (3)

Algorithm SkipInsert(k, e)

Input: Item(k, e)
Output: None

$p \leftarrow$ SkipSearch(k)
$q \leftarrow$ insertAfterAbove(p, null, (k,e))

while random() < $\frac{1}{2}$ do
  while above(p) = null do
    $p \leftarrow$ before(p)
  $p \leftarrow$ above(p)
  $q \leftarrow$ insertAfterAbove(p, q, (k,e))
Skip Lists – Removal: (1)

Algorithm SkipRemove(k)

Input: Key k

Output: Position (element with key k) if found otherwise NO_SUCH_KEY

\( p \leftarrow \text{SkipSearch}(k) \)

if key(p) \( \neq k \)
return NO_SUCH_KEY

else
use above(p) to go to top most level of position p in tower
remove all positions in tower starting from top

Skip Lists – Notes: (1)

- Insert / Removal: \( O(\log n) \)
- Improvements:
  - No need for references to the items at levels above 0 – only keys are needed.
  - No need for above and before methods.
  * Perform insertion and removal in a top-down scan-forward approach thus saving space for “up”, “prev” references

Skip Lists – Notes: (2)

- Improvements wont affect running time by more than a constant!
- Experiments indicate optimized Skip Lists in practice are faster than AVL trees and other balanced search trees!
- Keep reference to top most left position in S.
  - Can’t insert beyond top level of S.