Set 10: Ordered Dictionary Abstract Data Types: Skip Lists

CS240: Data Structures and Data Management

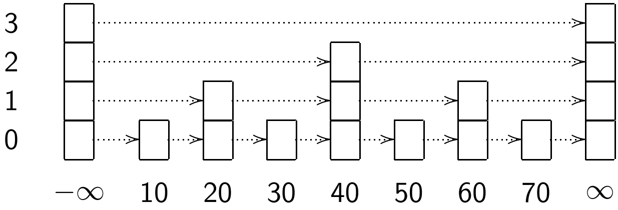
Jérémy Barbay

Outline

Skip Lists
Motivation

- A binary search tree built with random insertions has $\Theta(\log n)$ expected height. But we may get non-random insertion!
- Can we devise a data structure with randomness built-in?
  - Different runs on the same insert sequence yield a different structure
  - Not one particular bad input, just bad random sequences
- We wish to perform the binary search on linked list structure

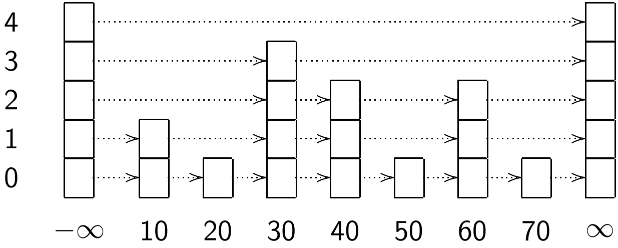
Skip List
Definition



A skip list of height h storing a dictionary/set, D, as a series of sequences $S_0, \ldots, S_h$ such that

- each S_i stores a subset of D in increasing order by key
- $-\infty$ and ∞ are in each S_i
- all of D is in S_0
- none of D is in S_h
- S_i contains a random subset of items in S_{i-1}

Comments



- Nodes in S_i form level i.
- Each key has a tower associated with it.
- Root is the top node of tower $-\infty$.
- Need to navigate down and right.
- Randomization is based on coins flipped by the computer.

Searching Algorithm

Find(K)

Start at the root node

while *not at level 0* **do**

Drop down a level

while *next key on level* $< K$ **do**

Move right on level

end while

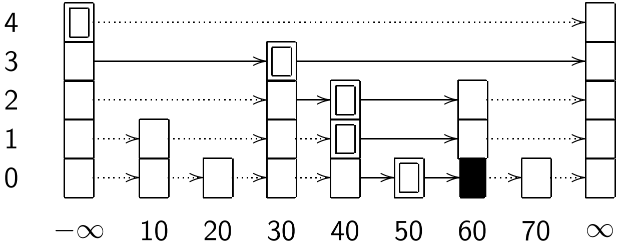
end while

return largest node $\leq K$

Searching Algorithm

Example

- `Find( 60 )`



- Framed nodes indicate where we would "drop down"
- Solid pointers indicate key comparisons made during search

Insertion Algorithm

Insert(K)

  FIND(K) *yielding node p*

  Insert tower base after node p

  while *flipped coin yields Heads* **do**

    Add a new level to tower

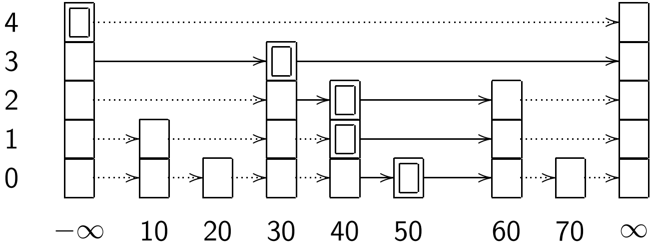
    Add pointer based on “drop down” for this level

  end while

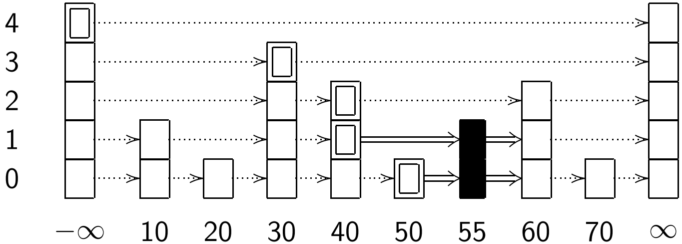
Insertion Algorithm

Example

- Start from 10-4 and `Insert( 55 )`



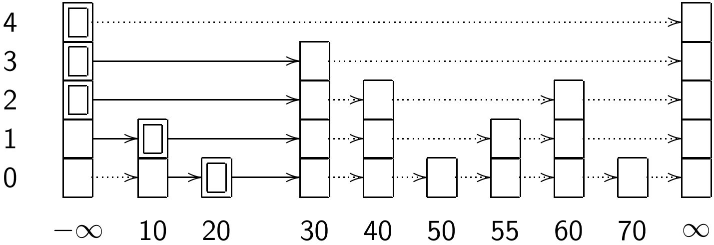
- Coin flip: `H`, `T`



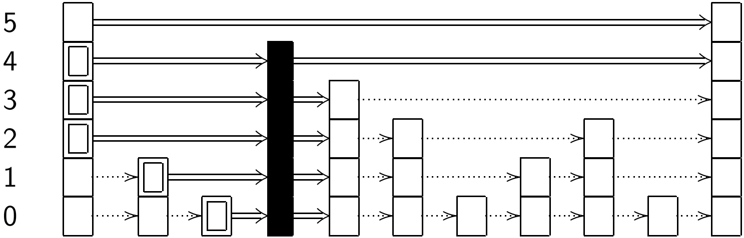
Insertion Algorithm

Example (cont)

- `Insert( 25 )`



- Coin flip: `H, H, H, H, T`



Deletion Algorithm

Delete(K)

FIND(K) *yielding node* p.

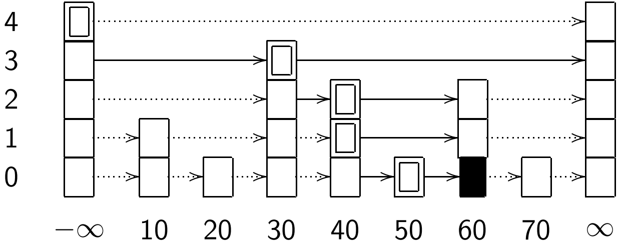
Climb the tower from p.

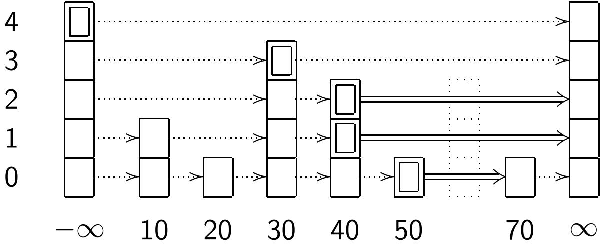
Adjust the "drop down" pointer for each level.

Deletion Algorithm

Example

- Start from 10-4 and `Delete( 60 )`

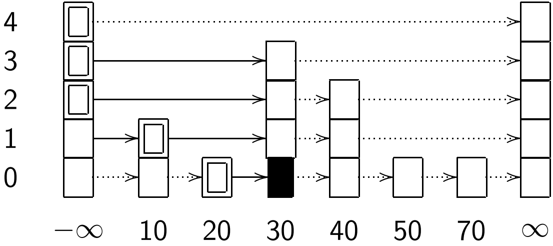


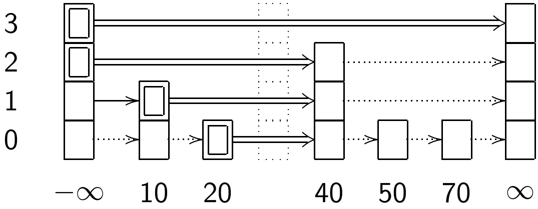


Deletion Algorithm

Example (cont)

- `Delete( 30 )`





General Remarks

- The Find operator must also return the drop down nodes
  Stack is appropriate
- If we use DLLs the drop down nodes can be omitted
  - Ability to navigate Up and Left through structure
  - Comes at a cost of extra space
- Data associated with keys stored only in S_0
- What should we do if a tower is "too high"?
  - force $h \leq c$, c a constant - problems as $n \to \infty$
  - force $h \leq f(n)$, $f(n) = \lceil 3 \log n \rceil$ is good
  - allow to grow infinitely .. not likely to get too high

Space Analysis

What is the expected number of nodes in the skip list?

- First, what are the expected number of keys, $|S_i|$, at level i?

$$\sum_{j=1}^{n} 1 \cdot \frac{1}{2^i} = \frac{n}{2^i}$$

- Summing over all levels yields:

$$\sum_{i=0}^{h} |S_i| = \sum_{i=1}^{h} \frac{n}{2^i} \approx 2n \in O(n)$$

Time Analysis

- Focus on expected cost of the Find operation, $T(n)$, with n elements in the skip list
  - Insert and Delete: $T(n) + h$
- What is a quick upper bound for $T(n)$ in terms of n and h?

$$O(n+h)$$

Theorem

The expected height of a skip list is $\Theta(\log n)$.

Proof.

At any level h, $E(|S_h|) = \frac{n}{2^h}$. Hence $E(h) \in O(\lg n)$. $\square$

Time Analysis (cont')

Theorem

The expected cost for a skip list search is $\Theta(\log n)$.

Proof:

- Cost of "dropping down" or "moving right" is 1
- Imagine the situation backwards: What is the length of the path from the node back to the root?
  Note $C(k)$ the expected path lenght that rises k levels on its backward trajectory.

$$
\begin{aligned}
C(k) &= \frac{1}{2}((\text{cost of going back one pointer}) + C(k)) \\
&\quad + \frac{1}{2}((\text{cost from level } i \text{ to level } i+1) + C(k-1)) \\
&= \frac{1}{2}(1 + C(k)) + \frac{1}{2}(1 + C(k-1)).
\end{aligned}
$$

Summary SkipLists

- Binary search in chained list is complicated but possible.
- Randomisation gives log time search and dynamicity.

References:

- Goodrich and Tamassia: pp. 195-202
- Cormen, Leisersen, Rivest, Stein: Not covered.