Skip Lists I

Ideal Skip List:

- Organize list in levels
- Level 0: Everything
  - 1: Every other
  - 2: Every fourth
  - ⋮
  - i: Every 2^i

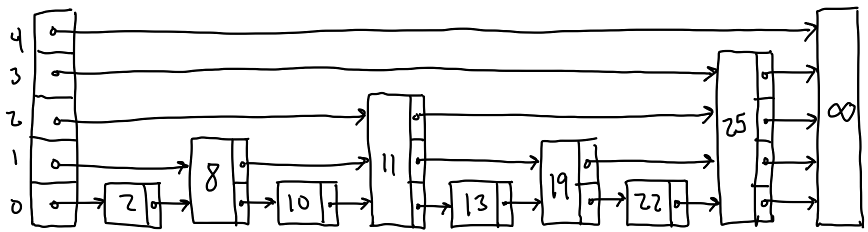
Sorted linked lists:

- Easy to code
- Easy to insert/delete
- Slow to search… $O(n)$

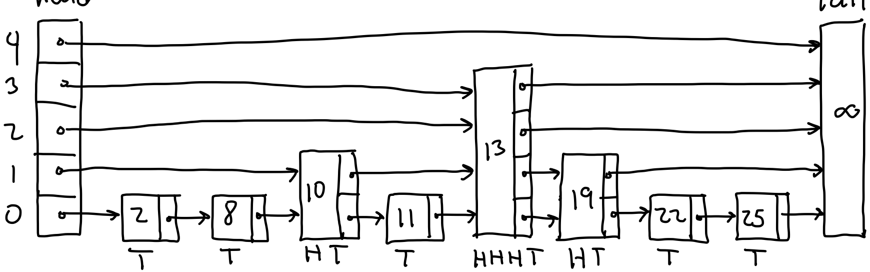
Idea: Add extra links to skip

How to generalize?

Example:



Too rigid → **Randomize!** To determine level – toss a coin + count no. of consec. heads:



Node Structure: (Variable sized)

```
class SkipNode {
    Key key
    Value value
    SkipNode[] next
}
```

In constructor, set size (height)

```
Value find(Key x) {
    i = topmost Level
    SkipNode p = head
    while (i ≥ 0) {
        if (p.next[i].key ≤ x) p = p.next[i]
        else i--
    }
    if (p.key == x) return p.value
        else return null
}
```

current node

until we hit base level

advance horizontal

drop down a level

we are at base level

Skip Lists II

Thm: A skip list with n nodes has $O(\log n)$ levels in expectation

Proof: Will show that probability of exceeding $c \cdot \lg n$ is $\le 1/n^{c-1}$.

→ Prob that any given node's level exceeds ℓ is $1/2^\ell$ [ℓ consecutive heads]

→ Prob that any of n node's level exceeds ℓ is $\le n/2^\ell$ [n trials with prob $1/2^\ell$]

→ Let $\ell = c \cdot \lg n$ ($\lg \equiv \log_2$)

Prob that max level exceeds $c \cdot \lg n$ is:

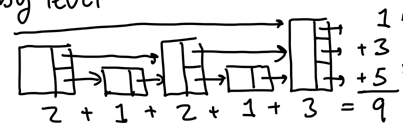
$$\le n/2^\ell = n/2^{(c \cdot \lg n)} = n/(2^{\lg n})^c = n/n^c = 1/n^{c-1}$$

$\square$

Obs: Prob. level exceeds $3 \cdot \lg n$ is $\le 1/n^2$. (If $n \ge 1{,}000$, chances are less than 1 in million!)

Thm: Total space for n-node skip list is $O(n)$ expected.

Proof: Rather than count node by node, we count level by level:



- Let n_i = no. of nodes that contrib. to level i.
- Prob that node at level $\ge i$ is $1/2^i$
- Expected no. of nodes that contrib. to level i = $n/2^i$

$\Rightarrow E(n_i) = n/2^i$

Total space (expected) is:

$$E\left(\sum_{i=0}^{\infty} n_i\right) = \sum_{i=0}^{\infty} E(n_i) = \sum_{i=0}^{\infty} n/2^i = n \sum_{i=0}^{\infty} 1/2^i = 2n$$

$\square$

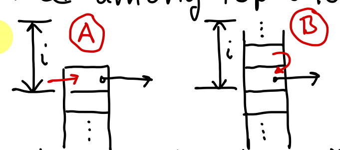
Thm: Expected search time is $O(\log n)$

Proof:
- We have seen no. levels is $O(\log n)$
- We'll show that we visit 2 nodes per level on average

Obs – Whenever search arrives first time to a node, it's at top level. (Can you see why?)

Def: $E(i)$ = Expect. num. nodes visited among top i levels.

Cases:



$$E(i) = 1 + (\text{Prob}(A))E(i) + (\text{Prob}(B))E(i-1)$$

current node ↑ same level ↑ from prior level

$$= 1 + \tfrac{1}{2}E(i) + \tfrac{1}{2}E(i-1)$$

$\Rightarrow E(i)(1 - \tfrac{1}{2}) = 1 + \tfrac{1}{2}E(i-1)$

$\Rightarrow E(i) = [1 + \tfrac{1}{2}E(i-1)]2 = 2 + E(i-1)$

Basis: $E(0) = 0 \Rightarrow E(i) = 2 \cdot i$

Let ℓ = max level. Total visited $= E(\ell) = 2 \cdot \ell$

$\Rightarrow$ We visit 2 nodes per level on average.

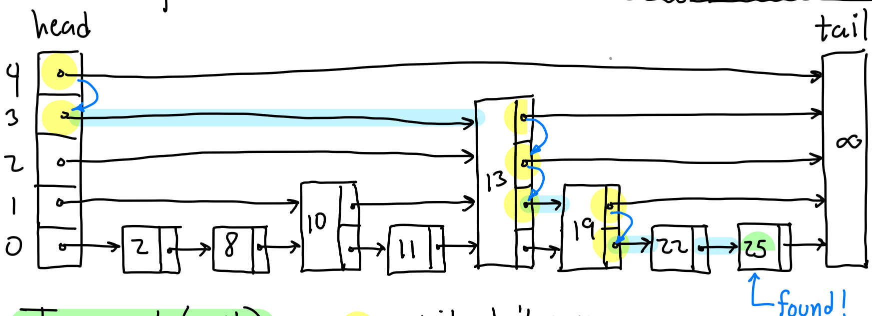
$\square$

Skip Lists III

Delete:
- Start at top
- Search each level saving last node < key
- On reaching node at level 0, remove it and unlink from saved pointers

Insert: (Similar to linked lists)
- Start at top level
- At each level:
  - Advance to last node ≤ key
  - Save node + drop level
- At level 0:
  - Create new node (flip coins to determine height)
  - Link into each saved node

Example: find(25)



Insert(24)

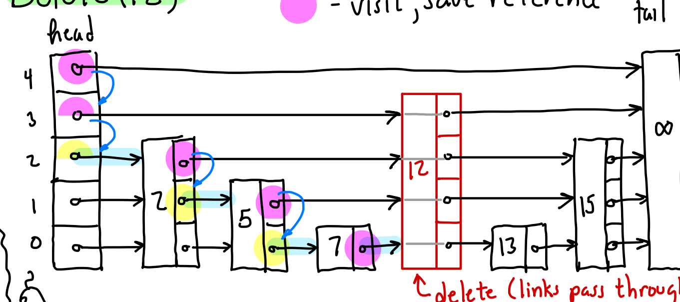
- visit, don't save

- visit, save reference

- visit, don't save

- visit, save reference

Delete(12)



Analysis: All operations run in time ~ find ⇒ $O(\log n)$ expected

Note: Variation in running times due to randomness only - not sequence
⇒ User cannot force poor performance.