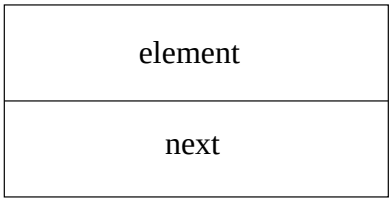
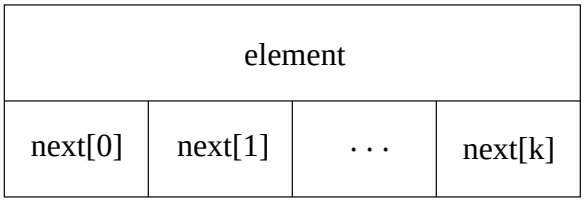
Skip Lists

Generalization of sorted linked lists for implementing Dictionary ADT (insert, delete, find, min, succ) in $O(logn)$ expected time per operation, where the n is the size of the dictionary. Skip lists compete with balanced search trees like AVL, Red-Black, and B-Trees.

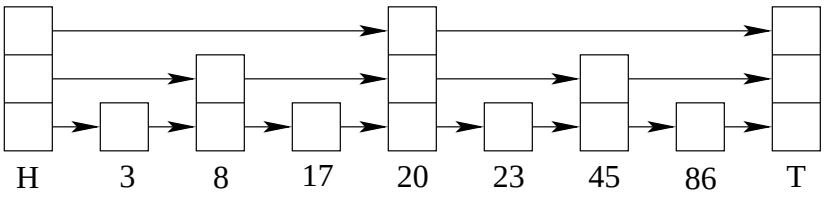


List Entry



Skip List Entry

The elements are stored in sorted order, in a linked list of nodes. Each skip list entry has an array of next pointers, where $next[i]$ points to an element that is roughly 2^i nodes away from it. The *next* array at each entry has random size between 1 and $maxLevel$, the maximum number of levels in the current skip list. Ideally, $maxLevel \approx \log n$. Each skip list has dummy head and tail nodes, both of $maxLevel$ height, storing sentinels $-\infty$ and $+\infty$, respectively. Iterating through the list using $next[0]$ will go through the nodes in sorted order. A reference to the previous element can also be stored by adding a *prev* field to Skip List Entry.



Search starts at $maxLevel$, goes as far as possible at each level, without going past target, descending one level at a time, until reaching the target node. Addition/Removal of nodes makes it difficult to maintain an ideal skip list, in which $next[i]$ of a node points to a node that is exactly 2^i away from it. Skip lists solve this problem by selecting the number of levels (size of $next[\,]$) of a new node probabilistically. When the size exceeds a threshold, elements are reorganized into an ideal skip list, with a new choice of $maxLevel$.

```
find(x): // Helper function
// return prev[0..maxLevel] of nodes at which search went down one level, looking for x
p ← head
for i ← maxLevel downto 0 do
    while p.next[i].element < x do
        p ← p.next[i]
    prev[i] ← p
return prev

contains(x):
prev ← find(x)
return prev[0].next[0].element = x ?

add(x):
prev ← find(x)
if prev[0].next[0] = x then
    prev[0].next[0].element ← x
else
    lev ← choice(maxLevel)
    n ← new SkipListEntry(x, lev)
    for i ← 0 to lev do
        n.next[i] ← prev[i].next[i]
        prev[i].next[i] ← n

choice(lev): // Prob(choosing level i) = 1/2 Prob(choosing level i − 1)
i ← 0
while i < maxLevel do
    b ← random.nextBoolean()
    if b then i++ else break
return i

remove(x):
prev ← find(x)
n ← prev[0].next[0]
if n.element ≠ x then
    return null
else
    for i ← 0 to maxLevel do
        if prev[i].next[i] = n then
            prev[i].next[i] ← n.next[i]
        else break
    return n
```