Skiplists

Guilherme P. Gonçalves

Jul/2014

Skiplist

Randomized data structure providing $O(logn)$ search, insertion and deletion

Simply put: hierarchy of sorted linked lists

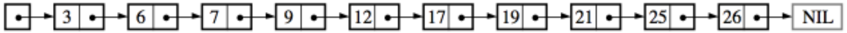
An element in the list at *level* i appears at level $i + 1$ with probability p

Search, insertion and deletion algorithms are direct extensions of linked list algorithms

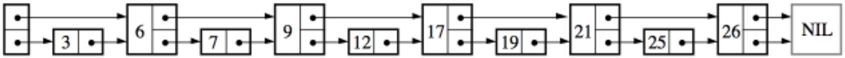
Deriving from a linked list

Consider a linked list of n nodes stored in sorted order.

A simple, singly-linked list would have search time $O(n)$.



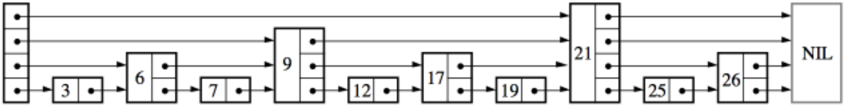
Adding pointers two nodes ahead every other node cuts this down to $O\left(\lceil \frac{n}{2} \rceil + 1\right)$:



Analogously, pointers four nodes ahead every fourth node enables search with $O\left(\lceil \frac{n}{4} \rceil + 2\right)$ comparisons

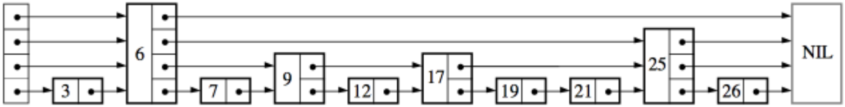
Deriving from a linked list

So how about pointing 2^i nodes ahead every 2^i nodes, $i \leq n$?



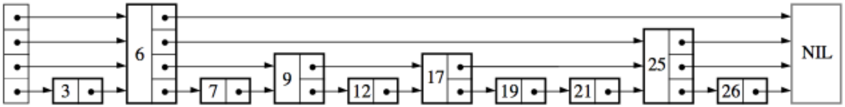
Then we have $O(\log n)$ search time, but modifications are hard!

The solution is to randomize levels such that they follow the same proportion as the rigid, deterministic data structure:



Skiplist - search

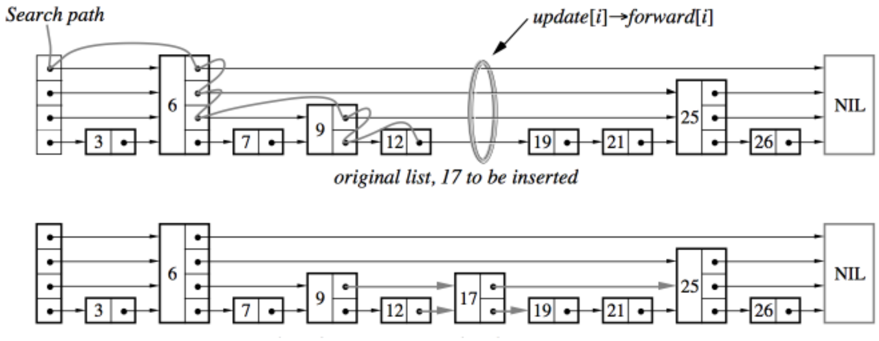
Descend through the levels, finding the rightmost element in each list that is smaller than the one we are searching for.



Example: a search for 21 visits nodes 6, 25, 9, 17, 25, 19, then finds 21.

Skiplist - insertion

The `update` array keeps track of the preceding elements in each list as we descend, much like a search. The level of the new node is chosen upon insertion and never modified.



The removal algorithm is also analogous to its linked list counterpart.

Analysis

Note that the time taken for each operation is dominated by the search time, so this analysis will focus on that.

For the deterministic version, the size of each list halved as the level went up, and we had $O(\log_2 n)$ time. With $p = \frac{1}{2}$, how close to that can we get?

In general, define $L(n) = \log_{\frac{1}{p}} n$. Informally, it can be proven that the search time is not larger than $L(n)$, with high probability.

Bounding the maximum level

Lemma: A skiplist with n elements has level $O(L(n))$ with probability $1 - o(\frac{1}{n^\alpha})$, for a constant α.

Proof: Let L_i, $i = 1, 2, ..., n$ be the level of the i-th element, and $L = max(L_i)$ be the level of the skiplist. Note that $L_i \sim G(1-p)$, so $Pr\left[L_i > k\right] = p^k$. Thus:

$$\begin{aligned} Pr\left[L > cL(n)\right] &= Pr\left[\bigcup_{i=1}^{n} L_i > cL(n)\right] && (1) \\ &\leq \sum_{i=1}^{n} Pr\left[L_i > cL(n)\right] && (2) \\ &= \sum_{i=1}^{n} p^{cL(n)} = np^{cL(n)} && (3) \\ &= np^{clog_{\frac{1}{p}} n} = \frac{1}{n^{c-1}} = \frac{1}{n^\alpha} && (4) \end{aligned}$$

Bounding the search time: backwards analysis

Idea: consider the search path for an element backwards, starting at the element itself and ending at the top left of the skiplist.

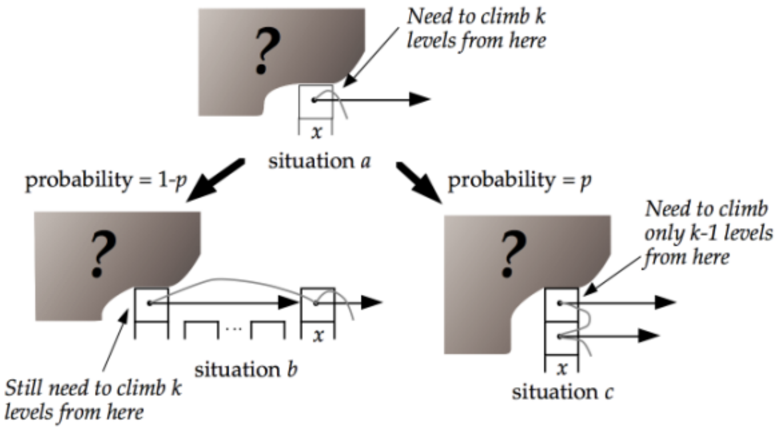

?
Need to climb k levels from here
x
situation a
probability = 1-p
probability = p
?
?
Need to climb only k-1 levels from here
x
x
situation b
situation c
Still need to climb k levels from here

Figure : One step in the backwards walk through the search path

Bounding the search time

Theorem: The search time in a skiplist of n elements is $O(\log n)$ with probability $1 - o(\frac{1}{n^\alpha})$.

Proof: Let $C(k)$ be the cost of climbing up to level k in the backwards search, with $C(0) = 0$.

$$\begin{aligned} C(k) &= p\left[C(k-1)+1\right] + (1-p)\left[C(k)+1\right] && (5) \\ &= \frac{1}{p} + C(k-1) = \frac{k}{p} && (6) \end{aligned}$$

But since the search starts at the topmost level, the total cost must be:

$$C(L) = \frac{L}{p} = O(logn) \tag{7}$$

with probability $1 - o(\frac{1}{n^\alpha})$

Implementation - initialization

```python
class SkiplistNode(object):
    def __init__(self, key, next):
        self.key = key
        self.next = next

class Skiplist(object):
    def __init__(self, p, maxlevel = 16):
        self.p = p
        self.level = 1
        self.maxlevel = maxlevel
        self.sentinel = SkiplistNode(float('inf'), [])
        self.header = SkiplistNode(
            None, [self.sentinel] * self.maxlevel)
```

Implementation - search

```
def search(self, key):
    x = self.header
    for i in range(self.level - 1, -1, -1):
        while x.next[i].key < key:
            x = x.next[i]
    x = x.next[0]
    return x.key == key
```

Implementation - insertion

```
def insert(self, key):
    rightmost = [self.header] * self.maxlevel
    x = self.header
    for i in range(self.level - 1, -1, -1):
        while x.next[i].key < key:
            x = x.next[i]
        rightmost[i] = x
    if x.next[0].key == key: return

    lvl = self._draw_random_level()
    self.level = max(self.level, lvl)
    xnext = [rightmost[i].next[i]
        for i in range(self.maxlevel)]
    x = SkiplistNode(key, xnext)
    for i in range(lvl):
        rightmost[i].next[i] = x
```

Implementation - drawing levels

```python
def _draw_random_level(self):
    lvl = 1
    while random.random() < self.p and
          lvl < self.maxlevel:
        lvl += 1
    return lvl
```

References

1. Skip Lists: A Probabilistic Alternative to Balanced Trees
2. Lec 12 | MIT 6.046J / 18.410J Introduction to Algorithms (SMA 5503), Fall 2005