University of Toronto
CSC263 Summer 2020, Project-Part3: Analysis for Skip-List

Yu-Ting Wang

August 2020

1 State THE BEST AND WORST TIME COMPLEXITY for each method without justification

Operation	Best Time Complexity	Worst Time Complexity
$Search$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
$Insert$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
$Delete$	$\mathcal{O}(1)$	$\mathcal{O}(n)$

2 Analyze THE SPACE COMPLEXITY FOR THE ENTIRE ADT

W.T.S. The list with n elements has $\mathcal{O}(\log n)$ expected number of levels
In part 1 we have stated that if we flip a fair coin and get a head, then we promote the node.
Define the probability that an element get PROMOTED to the next level as $p = \frac{1}{2}$
THEN:
The probability that an element in this list get up to level i (get i promotions) is $P_i = \frac{1}{p^i} = \frac{1}{2^i}$

Define the probability that the list has more levels than $c \cdot \log n$ times where $c \in \mathbb{R}^+$ to be F
THEN:
F = P(at least one element x get promoted more than $c \cdot \log n$ times)
$\leq n\cdot$ P(x get promoted at least $c \cdot \log n$ times) [by Boole's Ineqaulity]
= n $\cdot (\frac{1}{p})^{c \cdot \log n}$
= n $\cdot (\frac{1}{n^c})$
= $(\frac{1}{n^{c-1}})$; let $\alpha = c - 1$
$\implies F \leq \frac{1}{n^\alpha}$
By choosing wisely, we can make α arbitrarily large hence F arbitrarily small.
$\implies$ The list has $\mathcal{O}(\log n)$ expected number of levels *w.h.p.*

So we have the total space in the list as:
$n \cdot \Sigma_{i=0}^{\mathcal{O}(\log n)} \frac{1}{2^i} \leq n \cdot \Sigma_{i=0}^{\infty} \frac{1}{2^i} = n \cdot 2 = \mathcal{O}(n)$

Hence this ADT has an expected space complexity of $\mathcal{O}(n)$
The worst case happens when all $\mathcal{O}(\log n)$ levels has n elements, which make the worst space complexity as $\mathcal{O}(n \log n)$

3 Analyze the expected and worst time complexity for SEARCH (involving randomization)

EXPECTED CASE: $\mathcal{O}(\log n)$
Analyze it backwards. $\implies$ from the bottom to the top.
Search starts[ends] at a leaf(bottom level)

At each node visited:
if node wasn't promoted higher, then we go [CAME FROM] left. get TAILS here
if node was promoted higher, then we go [CAME FROM] up. get HEADS here
Search stops [STARTS] at the root(top level) or the dummy header. $\implies$ our movements are either UP or LEFT

Define the expected number of steps required to go up i levels = $C(i)$; $C(0) = \mathcal{O}(1)$

C(i)= Make one step, then make either $\begin{cases} C(i-1)\text{steps if this step went up; } p = \frac{1}{2}. \text{ get a HEAD} \\ C(i) \text{ steps if this step went left; } 1-p = \frac{1}{2}. \text{ get a TAIL} \end{cases}$

$\implies C(i) = 1 + \frac{1}{2}C(i-1) + \frac{1}{2}C(i)$
$\implies 2C(i) = 2 + C(i-1) + C(i)$
$\implies C(i) = 2 + C(i-1)$
$\implies$ Expected number of steps required at one level is 2.
$\implies C(i) = 2i$
and the expected number of levels of this ADT is $\mathcal{O}(\log n)$, hence the expected time for SEARCH is $2 \cdot \mathcal{O}(\log n) = \mathcal{O}(\log n) Q.E.D.$

WORST CASE: $\mathcal{O}(n)$
No one get promoted, so this list is just like a normal sorted linked list.
Therefore the worst case is simply traversing the whole list, which is $\mathcal{O}(n)$.

4 Analyze the expected and worst time complexity for DELETE (using amortization)

EXPECTED CASE: $\mathcal{O}(\log n)$

Consider m series of delete(x_i) where $x_i = x_1, ..., x_m$.
Each delete(x_i) takes an expected time $\mathcal{O}(\log n)$ time to search for x_i, [proven in 3],
aside: this ADT is a probabilistic data structure, we can only calculate the expected cost for one operation, it is not meaningful to consider the actual cost, so I will use expected time for one operation here.
and disconnecting takes constant time, so each one takes $\mathcal{O}(\log n) + \mathcal{O}(1) = \mathcal{O}(\log n)$ and m series of delete(x_i) will take $m \cdot \mathcal{O}(\log n)$
$WCSC(m) = m \cdot \mathcal{O}(\log n)$;
Hence the expected time complexity of DELETE is $\mathcal{O}(\log n)$

WORST CASE: $\mathcal{O}(n)$
No one get promoted, so this list is just like a normal sorted linked list.
Therefore we have to traverse the whole list to find x and then disconnect the node.

Time(traverse) + time (disconnect) = $\mathcal{O}(n) + \mathcal{O}(1) = \mathcal{O}(n)$.

references:
William Pugh . July, 1989 . A Skip List Cookbook , University of Maryland
https://www.cs.cmu.edu/ ckingsf/bioinfo-lectures/skiplists.pdf