Skip Lists

[Notes for ESTR2102]

Yufei Tao

CSE Dept
CUHK

The AVL-tree is able to solve the dynamic predecessor problem with $O(\log n)$ time per operation. Today, we will see another structure called the skip list to achieve the same time, but in expectation. The main advantage of the skip list is that, it is extremely simple to implement, and involves nothing but several linked lists! The structure serves as another example demonstrating the power of randomization.

Recall:

Dynamic Predecessor Search

Let S be a set of integers. We want to store S in a data structure to support the following operations:

- A predecessor query: give an integer q, find its predecessor in S, which is the largest integer in S that does not exceed q;
- Insertion: adds a new integer to S;
- Deletion: removes an integer from S.

Skip List

For each element $e \in S$, compute its level $l(e)$ as follows:

1. Initialize $l(e) = 1$.
2. Toss a coin with head probability $1/2$.
3. If the coin heads, increment $l(e)$ by 1 and repeat from Step 2.
4. Otherwise, return $l(e)$.

Note that the value of $l(e)$ is a random variable.

Skip List

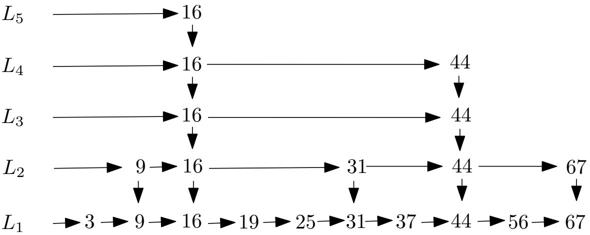
For each level $i \geq 1$, define L_i as $\{e \in S \mid l(e) \geq i\}$, namely, the set of elements in S whose levels are at least i.

Sort each L_i in ascending order, and chain up all the elements by the sorted order with a linked list.

For each element $e \in L_i$ ($i \geq 2$), store a downward pointer to its copy in L_{i-1}.

The height $h = \max_{e \in S} l(e)$. Note that h can be ∞ in the worst case!

Example



$l(16) = 5, l(44) = 4, l(9) = l(31) = l(67) = 2$, and the levels of all other elements equal 1. Here, $h = 5$.

Predecessor Query

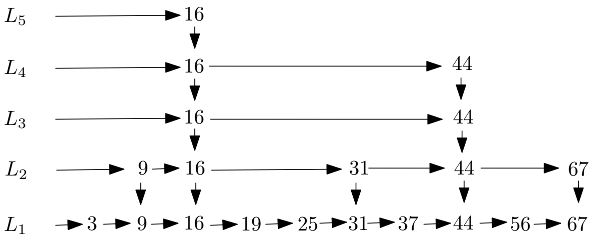
Let q be the search value of the query.

If $h > 2\log_2 n$, we simply scan L_1 in full to answer the query.

Otherwise, we start from the highest level, and descend via the predecessor at this level:

1. $i = h$, and define $x = -\infty$
2. Find the predecessor y of q in L_i by walking from x towards right.
3. If $i = 1$, then return y.
4. Otherwise, descend to the copy of y in L_{i-1}.
5. Decrease i by 1, set x to y, and repeat from Line 2.

Example



Consider $q = 38$. At L_5, we stop at 16. Then, descend to the 16 at L_4, but stop there. Same story at L_3. Descend to the 16 at L_3, and stop at 31 at that level. Descend to the 31 at L_1, and stop at 37, which is the predecessor.

Worst case query time? Actually $O(n)$ (why not $O(n \log n)$?)! Nevertheless, next we will prove that the expected query time is only $O(\log n)$.

Analysis

Let us start with the space consumption to gain some intuition. In the worst case, the space consumption can be ∞! But the probability of that happening ought to be extremely low.

Indeed, it is easy to verify that each element has probability 1 to be L_1, $1/2$ to be in L_2, $1/4$ to be in L_3, and so on.

So L_i ($i \geq 1$) has $n/2^{i-1}$ elements in expectation. The total number of elements of all the linked lists is in expectation:

$$\sum_{i=1}^{\infty} \frac{n}{2^{i-1}} = O(n).$$

Analysis

By the same reasoning, how large is the height h on average? Intuitively, it should be $O(\log n)$ because we are losing half of the elements each level up! The following theorem confirms this intuition with a stronger fact:

Theorem: $\boldsymbol{Pr}[h \geq 1 + 2 \cdot \log_2 n] \leq \frac{1}{n}$.

> Essentially says that $h \leq 2 \log_2 n$ with "high probability" at least $1 - 1/n$.

Analysis

Proof: Let l_i denote the level of the i-th element of S. We know that for any $x \geq 2$

$$\textbf{Pr}[l_i \geq x] = 1/2^{x-1}.$$

Setting $x = 1 + 2\log_2 n$ gives

$$\textbf{Pr}[l_i \geq 1 + 2\log_2 n] = 1/2^{2\log_2 n} = 1/n^2.$$

Therefore:

$$\textbf{Pr}[h \geq 1 + 2\log_2 n] \leq \sum_{i=1}^{n} \textbf{Pr}[l_i \geq 1 + 2\log_2 n] = n/n^2 = 1/n.$$

$\square$

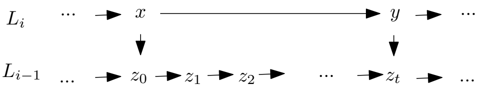
Analysis

So we know there are $O(\log n)$ levels with high probability. But how much time do we spend at each level? Note that if we are not lucky, we may need to spend $O(n)$ time at a level (think: why?).

Nevertheless, we will show that in expectation we spend only $O(1)$ time per level.

Analysis

Without loss of generality, let us consider an arbitrary level $i \geq 2$. Let x be the predecessor of q in L_i, and y be the element that succeeds x in L_i. Let $z_0, z_1, ..., z_t$ be the elements in L_{i-1} that are in the range $[x, y]$—notice that $z_0 = x$ and $z_t = y$.



If we are not lucky, at level $i - 1$, we may spend $O(t)$ time. But we will prove:

Lemma: The expectation of t is $O(1)$.

Analysis

Proof: Event $t = x$ happens only if $z_1, z_2, ..., z_{x-1}$ do not make it to level i, the probability of which is $1/2^{x-1}$.

Hence:

$$\boldsymbol{E}[t] = \sum_{i=1}^{n} \frac{x}{2^{x-1}} = O(1).$$

□

Analysis

So now it remains to prove that the overall query time is $O(\log n)$ in expectation. Note that it is not correct to simply multiply the expected height and the expected cost per level—because the two corresponding random variables are not independent!

But a little trick will do the job. We already know that with probability at least $1 - 1/n$, the height is at most $2 \log_2 n$, in which case the expected query cost is $O(\log n)$. In the event of the remaining $1/n$ probability, the query cost is no more than $O(n)$. Hence, the overall query cost is at most

$$O(\log n)(1 - 1/n) + O(n) \cdot (1/n) \quad = \quad O(\log n)$$

in expectation.

We have not elaborated on the insertion and deletion algorithms yet, which at this moment should have become simple exercises for you. Try them out. Your algorithms must finish in $O(\log n)$ expected time.