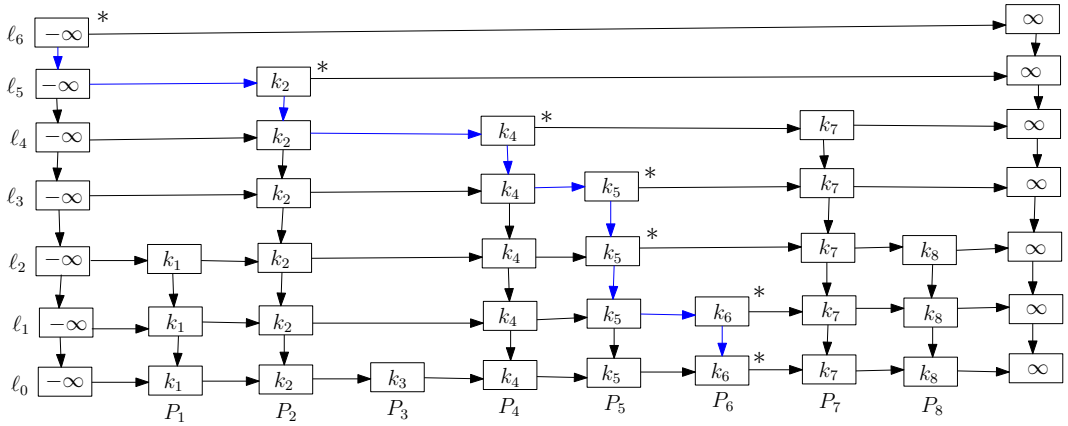
- Objective: Efficiently searching in a singly linked list while supporting dictionary operations.
- Let S_0 be an ordered set comprising input keys $k_1, k_2, \ldots, k_n$, such that $k_i < k_{i+1}$ for $1 \leq i \leq n-1$. The algorithm computes a hierarchy of ordered sets: $S_0 \supseteq S_1 \supseteq \ldots \supseteq S_r$. For each k_i, the *level* ℓ_i of k_i is the number of coin flips before obtaining a tail for the first time. In this note, we assume the algorithm uses an unbiased coin. The key k_i is included in every set in $S_0, S_1, \ldots, S_{\ell_i}$. We also say the level of S_j is ℓ_j, for every j.

  The below figure shows these sets stacked one above the other, and the elements of each ordered set S_j is stored in a singly linked list: nodes of each row corresponds to a set in the hierarchy, and the collection of all the nodes in any column is called a *pile*. For every k_i, the pile P_i of k_i has $\ell_i + 1$ nodes, and each node of that pile stores k_i. We define $r = 1 + \max\{\ell_i : 1 \leq i \leq n\}$. The skip list also has two sentinel piles: every node of one such pile $P_{-\infty}$ has key $-\infty$ and every node of the other pile P_{∞} has key ∞, and the number of nodes in either of these two piles is $r+1$. The $-\infty$-pile precedes P_1 and the ∞-pile succeeds P_n, and these are placed as shown in the figure. Naturally, $S_r = \{-\infty, \infty\}$.

  Any node x not belonging to $-\infty$-pile, ∞-pile, and S_0, has a horizontal pointer and a vertical downward pointer. Suppose x belongs to set S_j and to pile P_i. Then, the horizontal pointer from x refers to node having the smallest key in S_j that is larger than k_i, whereas the vertical pointer refers to the node in $S_{j-1} \cap P_i$.



$n = 8, r = 6$, and the level of k_2 is 5

$S_2 = \{k_1, k_2, k_4, k_5, k_7, k_8\}$

keys in ℓ_2, in sorted order, are $-\infty, k_1, k_2, k_4, k_5, k_7, k_8, \infty$

search path for key 27 is shown in blue, assuming $k_6 = 20, k_7 = 30$

the $*$ marked nodes are the nodes pushed to stack in the algorithm to insert key 27 between k_6 and k_7

also, the $*$ marked nodes in levels 0 to 4 help in deleting key $k_7 = 30$ (the entire pile P_7)

- With high probability, the number of levels is $O(\lg n)$.

  * The probability r is $\geq k$ is at most $n \cdot pr(\ell_i \geq k)$[1] $= n(\frac{1}{2})^k$.

[1] from Boole's inequality: $pr(\cup_{i=1}^{n} E_i) \leq \sum_{i=1}^{n} pr(E_i)$

Hence, probability the number of levels r is $\geq c \lg n$ is at most $\frac{n}{2^{c \lg n}} = \frac{1}{n^{c-1}}$, that is, polynomially small.

The number of levels is $O(\lg n)$ in expectation.

* $E(r)$
  $= \sum_{i=1}^{\infty} ipr(r = i)$
  $= \sum_{i=1}^{\infty} i(pr(r \geq i) - pr(r \geq i+1))$
  $= \sum_{i=1}^{\infty} pr(r \geq i)$
  $= \sum_{i=1}^{c \lg n} pr(r \geq i) + \sum_{i > c \lg n} pr(r \geq i)$
  $\leq \sum_{i=1}^{c \lg n} 1 + \sum_{i > c \lg n} \frac{n}{2^i}$
  $= c \lg n + \frac{n}{2^{c \lg n + 1}}(1 + \frac{1}{2} + \frac{1}{2^2} + \ldots)$
  $= c \lg n + \frac{1}{n^{c-1}}$.

The space needed to store n keys in a skip list is $O(n)$ with high probability.

* Since a fair coin is used, if the number of nodes in ℓ_i is n_i, the number of nodes in ℓ_i that survive to reach level ℓ_{i+1} is $\frac{n_i}{2}$. Since the number of levels is $O(\lg n)$ with high probability, the space including the space for sentinel piles, with high probability, is
  $O(\lg n) + (n + \frac{n}{2} + \frac{n}{4} + \ldots + (O(\lg n) \text{ terms}))$
  $\leq O(\lg n) + (n + \frac{n}{2} + \frac{n}{4} + \ldots)$
  $= O(n)$.
  As shown above, probability level ℓ_r, for $r = O(\lg n)$, to have more than two nodes is polynomially small. Hence, the space is $O(n)$ with high probability.

The space needed to store n keys in a skip list is $O(n)$ in expectation.

* The probability a key k_i belongs to set S_j is $\frac{1}{2^j}$. Hence, $E[|S_j|] = \frac{n}{2^j}$.
  Therefore, $E[\sum_{j=1}^{r}(|S_j|)] = \sum_{j=1}^{r} E[|S_j|] \leq \sum_{j=0}^{\infty} \frac{n}{2^j} = 2n$.
  Since $E[r] = O(\lg n)$, including nodes in two sentinel piles and the nodes in $|S_0|$, the space in expectation is $O(n)$.

- Algorithm to search for a key k in the skip list:

The search starts at node x with key $-\infty$ in level ℓ_r. Consider any node v, not belonging to level ℓ_0 and not belonging to either of the sentinel piles, on the search path. At v, algorithm checks whether the horizontal pointer at v is to a node with key $> k$. If this is the case, then the search proceeds to node located below v using the vertical pointer stored at v. Otherwise, the search algorithm proceeds to node referred by horizontal pointer stored at v. The search terminates upon reaching to any node y in level ℓ_0. Then the algorithm checks whether the key stored at y is equal to k.

Observation 1: When the search path for key k ends at a node y of ℓ_0, the key stored at y is equal to k iff k is in the skip list.

Observation 2: Any search path P enters any pile at the top node of that pile.

The algorithm to search for any key k takes $O(\lg n)$ time with high probability.

* As part of backward analysis, consider path P^r obtained by reversing P, so that P^r goes from $y \in \ell_0$ to $x \in \ell_r$. From Observation 2, P^r moves up till it reaches the top node of a pile and then goes left, and this repeats up till it reaches x. Specifically, at any node v, the node above v belongs to P^r with probability equal to the probability of obtaining heads with a coin flip and the node left of v belongs to P^r with probability equal to the probability of obaining tail with a coin flip.

  Since we know the expected number of vertical moves in P^r is $O(\lg n)$, the path length of P^r is essentially the total number of moves needed to obtain $O(\lg n)$ vertical moves. This is equivalent to number of coin flips needed until $O(\lg n)$ heads are obtained.

  For $c' > 1$, probability that there are exactly $c \lg n$ heads in $c'c \lg n$ coin flips is
  $= \binom{c'c \lg n}{c \lg n} \cdot (\frac{1}{2})^{(c'-1)c \lg n} (\frac{1}{2})^{c \lg n}$.

  Hence, for $c' > 1$, probability that there are at most $c \lg n$ heads in $c'c \lg n$ coin flips is
  $\leq \binom{c'c \lg n}{c \lg n} \cdot (\frac{1}{2})^{(c'-1)c \lg n}$ (ignoring the term due to heads)
  $\leq (e\frac{c'c \lg n}{c \lg n})^{c \lg n} \cdot (\frac{1}{2})^{(c'-1)c \lg n}$ (since $\binom{a}{b} \leq (e\frac{a}{b})^b$)
  $= (c'e)^{c \lg n} 2^{-(c'-1)c \lg n}$
  $= 2^{\lg ((c'e)^{c \lg n})} 2^{-(c'-1)c \lg n}$
  $= 2^{(\lg (c'e) - (c'-1))c \lg n}$
  $= \frac{1}{n^{((c'-1) - \lg ((c'e))c}}$.

  For $c' = 10$, this is already polynomially small. That is, in $10c \lg n$ coin flips, the probability of having $> c \lg n$ heads is large. Hence, the path length is $\leq 10c \lg n$ with high probability.

The algorithm to search for any key k takes $O(\lg n)$ time in expectation.

* Let $C(j')$ be the expected number of steps to walk up j' top levels to reach x. Then,

  $C(j') =$ cost of visiting current node $+$
  (probability of not moving up $* C[j']) +$
  (probability of moving up $* C[j'-1]$) for $j' > 0$, and $C(0) = 0$.

  Since $C(j')$ is again appearing on the right-hand side in this recurrence, it appears the expected number of moves is not changing when the path stays in the same level. However, multiplying $C(j')$ with the probability of not moving up is mitigating the fact $C(j')$ not changing.

  Rearranging terms of this recurrence, $C(j') = 1 + \frac{1}{2} \cdot C(j') + \frac{1}{2} \cdot C(j'-1)$
  $\Rightarrow C(j') = 2 + C(j'-1)$.

  Unraveling this recurrence leads to $C(j') = 2j'$, since $C(0) = 0$.

  Since the maximum value of j' is r, $C(r) = 2r$.

  Since r is $O(\lg n)$ in expectation, $C(r)$ is $O(\lg n)$. This is precisely the number of moves to walk up r levels, the search path length.

- Algorithm to insert a key k_i into the skip list:

  On the search path for k_i, if k_i is ever encountered, then this algorithm exits. Hence, to describe this algorithm, it is reasonable to assume k_i is not in the skip list. When the search algorithm is invoked from the algorithm to insert any key k_i, in every level ℓ from r to 0, a pointer to node containing the largest key in ℓ that is less than k_i is pushed to stack. Note that, at every node v' pushed to stack, search path moves downwards at v'. A subset of these nodes on stack help in inserting key k into singly linked lists in levels to which k_i belongs to. Suppose tossing a coin yielded ℓ_i number of heads before obtaining a tail. Then, a pile P_i of length $\ell_i + 1$ is initiated by setting key of each node in P_i to k_i and by initializing appropriate vertical downward pointers of nodes in P_i. Then, one node of P_i is introduced in each of the levels in $[0, \ell_i]$: for each node $v'' \in S_j$ of the first $\ell_i + 1$ nodes that are popped from the stack, a node $v''' \in P_i \cap S_j$ with $v'''.key = k_i$ is inserted after v'' in the linked list to which v'' belongs. Of course, if it is known that the insertion algorithm is invoked only on keys that are not in the skip list, then the stack is not required since k_i can be inserted into every level $\ell \in [0, \ell_i]$, just after the largest key in ℓ that is less than k_i (similar to deletion algorithm mentioned below). The degenerate case of $\ell_i > r$ is handled by including more nodes into sentinel piles.

  Algorithm to delete key k_i:

  To delete k_i, the algorithm searches for the predecessor k_i' of k_i. In every level ℓ from r to 0 on the search path of k_i', for the node v containing the largest key in ℓ that is less than k_i, if the horizontal pointer from v is to a node v' with key k_i, then delete v' from the linked list in ℓ. Again, if this deletion leads to number of levels reducing, then the sentinel piles are cleaned up accordingly.

  Since the time complexity of insertion (resp. deletion) algorithm is dominated by the length of the search path it computes, the insertion (resp. deletion) of a key into (resp. from) skip list takes $O(\lg n)$ time in expectation, and is accomplished in $O(\lg n)$ time with high probability.

References:

- Randomized Algorithms by R. Motwani and P. Raghavan. Cambridge University Press. 1995.

- Skip Lists: A Probabilistic Alternative to Balanced Trees. William Pugh. Communications of the ACM. 33(6): 668–676, 1990.

- Plus, the folklore.