Example 5: Skip Lists

Input: A set S of n distinct numbers

Store them in a data structure to answer

QUERY:

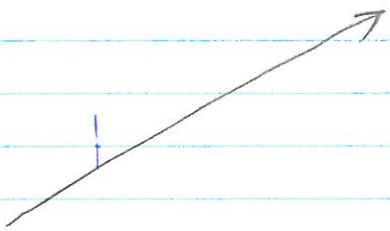
a) SEARCH(x): Return the largest element in S that is less than or equal to x.

b) INSERT(x): Insert x into S.

(for simplicity we will assume x is not in S)

c) DELETE(x): Remove x from S.

Classical Structures: { AVL-Tree
{ Red-Black Tree



COMP 3804
- Dynamic balanced binary trees

Searching is easy, but insertion/deletion are cumbersome due to rotation operations etc.

Here: Skip Lists — structure that is balanced in terms of expected value
- insertion/deletion are very simple.

Refinement of set S.

We define subsets of S via coin flips as follows.

1. Set $S_0 = S$;

2. Assume, for $i = 0, 1, 2, \cdots$, S_i has already been constructed.

3. If $S_i \neq \phi$ do

   [Set $S_{i+1} = \phi$;
   For each element $y \in S_i$, flip a fair coin, and if it is "heads", add y to S_{i+1}.

4. Terminate when the next set is empty (i.e $S_{i+1} = \phi$)

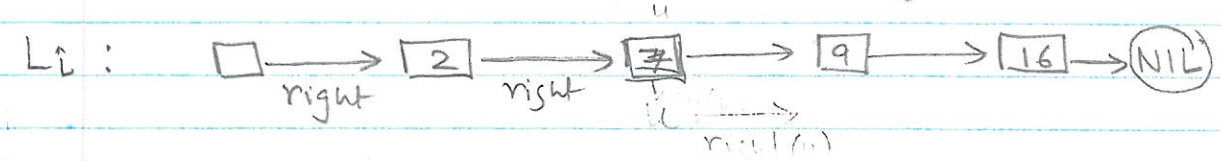
Define h = # non-empty sets constructed by this process.

Note that h, and each of the sets $S_1, S_2, \cdots, S_h$ are determined by a random process.

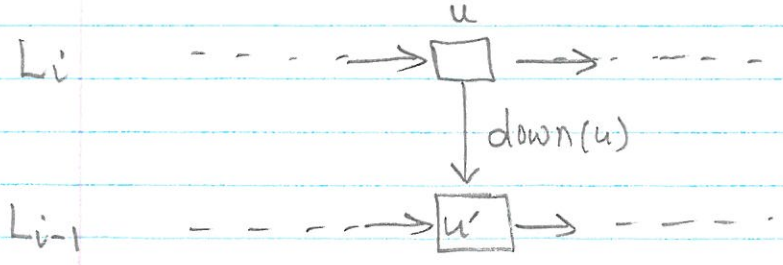
Construction of Skip List for S

1. For each S_i, $0 \le i \le h$, store the elements in S_i in sorted order in a linked list L_i

   - Each node u of L_i stores one element of set S_i and it is denoted by key(u).
   - Each node u of L_i stores a pointer to its successor node, denoted by right(u). (If u is rightmost node, right(u) = NIL)
   - Add a dummy node at the beginning of L_i. Its key is NIL, and its successor is the node of L_i storing the smallest element of set S_i.



2. For each i with $1 \le i \le h$, each node u of L_i stores a pointer to the node u' in L_{i-1} for which key(u) = key(u'). The node u' is denoted by down(u).



3. There is a pointer to the dummy node in L_h. Dummy node in L_h is called the "root"-node of Skip List and for each operation on Skip-List, we start with the root-node.

$h \equiv$ Height of Skip-List.

Here is an illustration of Skip List for $S = \{2, 5, 7, 9, 12, 16, 20\}$.

Here $S_1 = \{2, 7, 9, 16, 20\}$

$S_2 = \{2, 7, 9, 16\}$

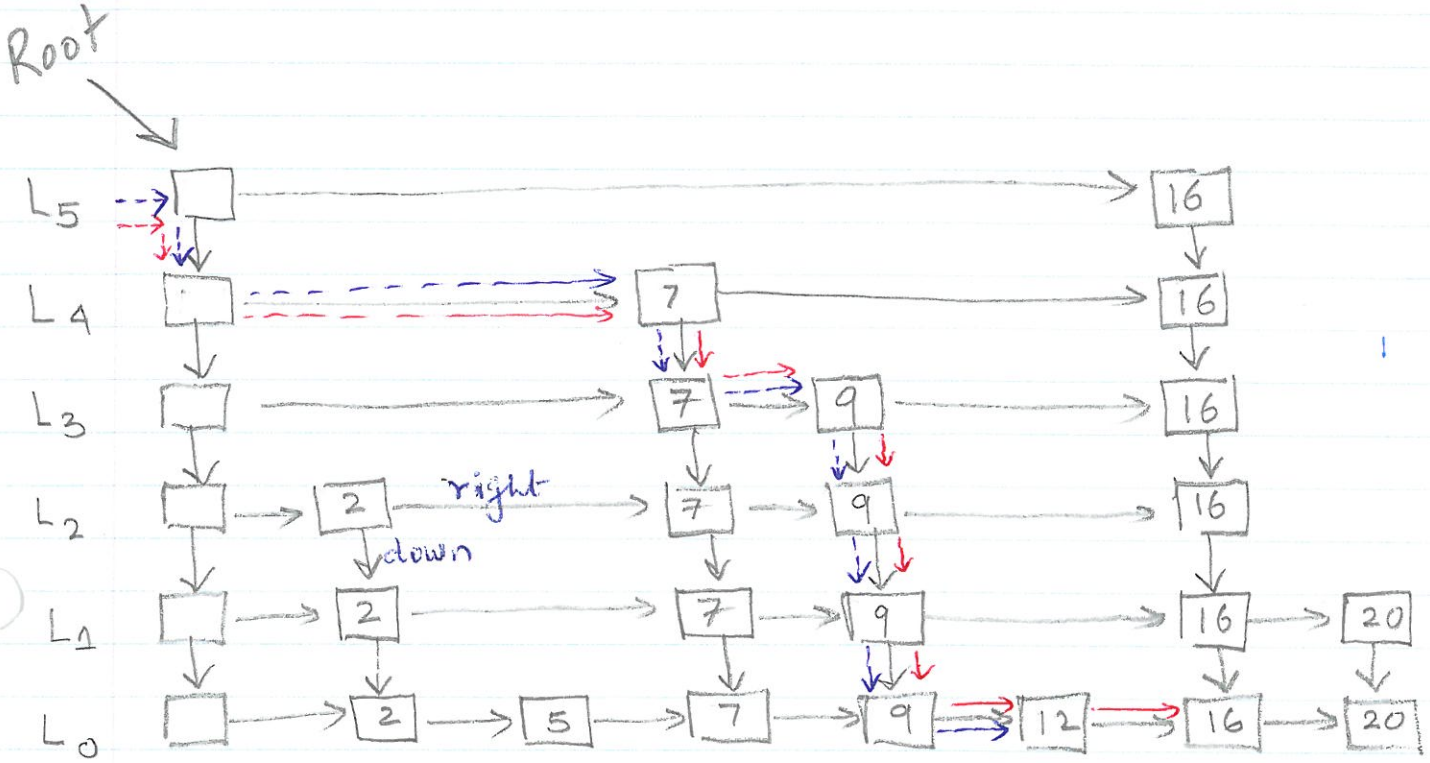
$S_3 = \{7, 9, 16\}$

$S_4 = \{7, 16\}$

$S_5 = \{16\}$.

$h = 5$ and corresponding lists

$L_0, L_1, L_2, L_3, L_4, L_5$ with all the keys, right, down, and root ptr are illustrated.



search(13) → Returns(12)

search(16) → Returns(16)

SEARCH(x)

Idea: To seach x, start at the root of the skip list and keep track of the current node u and the index i of List L_i that contains u.

Initially: $u = root$, $i = h$

Iterate:

```
If i ≥ 1 then
    if key(right(u)) < x
    then u = right(u)
    else [u = down(u);
          i := i - 1]
else (i.e. i = 0)
    [while key(right(u)) ≤ x do
        u := right(u);]
```

Return (u)

In our example search(13) and search(16) are illustrated.

Note that we test for key(right(u)) < x and not key(right(u)) ≤ x.

This is to simplify our Delete(x) procedure.

INSERT(x)

- We assume $x \notin S$.

- Suppose we execute SEARCH(x) and the procedure returns the node u in list L_0. Observe that
  - We know that x needs to inserted between u and right(u).
  - Moreover, we need to flip a fair coin and decide in how many lists (or sets) $L_1, L_2, \ldots$; x needs to be inserted. (This may lead to creating even new levels.)

Let k be the number of lists where x needs to be inserted. (i.e it needs to be added in $L_0, L_1, L_2, \ldots, L_{k-1}$)

If $k \geq h+2$, we add new lists $L_{h+1}, L_{h+2}, \ldots, L_{k-1}$ to the skip list, each containing a dummy node and a node storing x.

We set $h = k-1$.

- We modify search(x) so that insertion can be achieved easily.

Assume that we know k, and if $k \geq h+2$, we have already added extra levels $L_{h+1}, \ldots, L_{k-1}$.

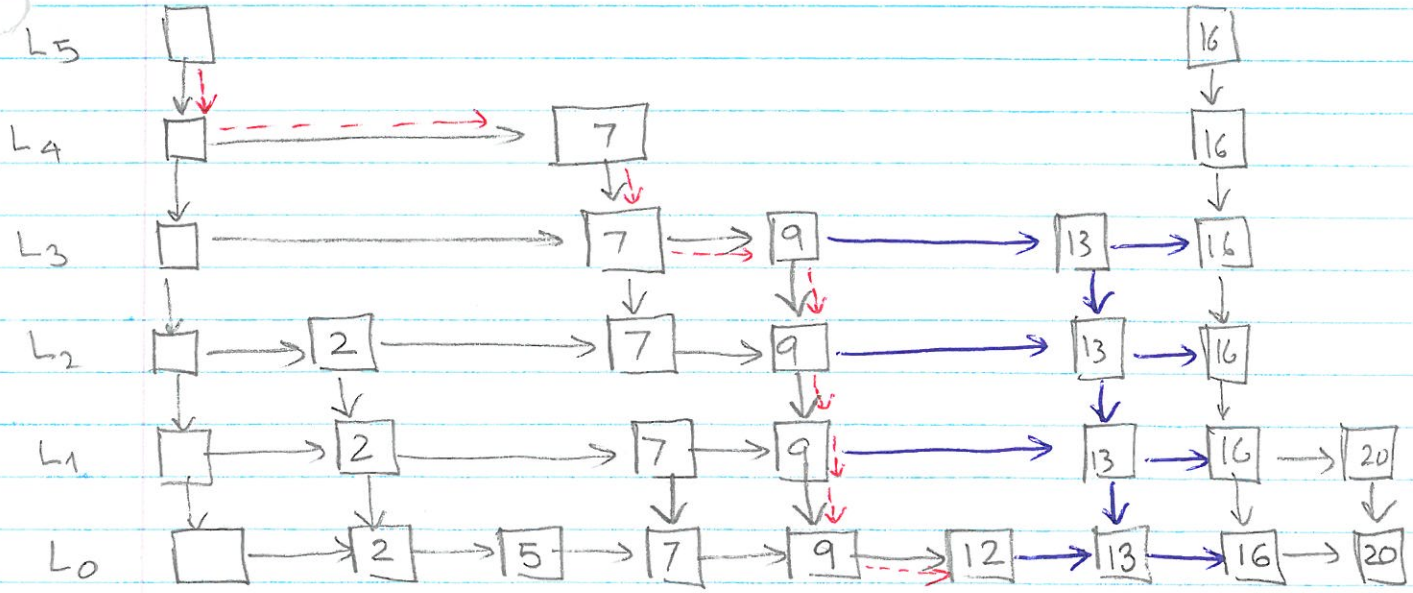
In search(x) procedure, if $i < k$ and the current node u moves down, we add the new element x to L_i between the nodes u and right(u).

Suppose in our example we need to INSERT (13).

Let us say we flip a coin and it shows "heads" 3 times, i.e, $k = 4$. So element 13 needs to be inserted in lists L_0, L_1, L_2, L_3.

Here is the new skip list – all the BLUE pointers are new.

Path for SEARCH(13) is shown in RED



DELETE(x)

Execute Search(x), but each time the current node u moves down, check if key(right(u)) = x. then

[delete right(u) by setting
right(u) := right(right(u))

If there are some lists $L_h, L_{h-1}, \ldots$ that are only left with dummy nodes, delete those lists and readjust h.

Next: Analysis of Skip Lists.

To show

1. $E(h(x)) = 1$

   $h(x) \equiv$ largest i such that $x \in L_i$.

2. $Pr(x \in L_i) = \frac{1}{2^i}$

3. $E(|L_i|) = \frac{n}{2^i}$

4. $E(X) = 2n$

   $X = \sum_{i=0}^{h} |L_i|$

   $|L_i| \equiv$ # elements in list L_i excluding the DUMMY ELEMENT

5. $E(h) \leq \log n + 1$

6. $E(N) \leq 2\log n + 5$

   $N =$ # nodes on the search path of x.

7. Search, Insert, Delete takes $O(\log n)$ time on a SKIP LIST of of set S consisting of n elements.

1. $E(h(x)) = 1$

Let $x \in S$, i.e $x \in L_0$

Define $h(x)$ = largest i such that $x \in L_i$

Note that for each element $x \in L_0$, we flip a fair coin and if it comes up heads, the element is placed in the next list (set) and the process is repeated till a tails shows up. We know that expected # coin flips required to obtain a tails is 2, thus $E(h(x)) = 1$.

2. $\Pr(x \in L_i) = \frac{1}{2^i}$

$x \in L_i$ iff first i coin flips for x (all results) in heads.

3. $E(|L_i|) = \frac{n}{2^i}$, $i \geq 0$

$|L_i|$ = # elements in list L_i excluding the dummy element.

Each element $x \in L_0$ is contained in L_i with probability $\frac{1}{2^i}$, independent of all other elements.

Size of L_i is a random variable with binomial dist. with parameter $1/2^i$ and n.

Thus $E(|L_i|) = \frac{1}{2^i} \cdot n = n/2^i$ ∎

4. $E(X) = 2n$

where $X = |L_0| + |L_1| + \cdots + |L_h|$

Proof: $E(X) = E\left(\sum_{i=0}^{\infty} |L_i|\right)$

↘ Linearity of expectation

$$= \sum_{i=0}^{\infty} E(|L_i|)$$

$$= \sum_{i=0}^{\infty} n/2^i = n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n.$$

Alternate Proof.

We know that each element $x \in S$ occurs in $h(x)+1$ lists. It occurs in lists $L_0, L_1, L_2, \ldots, L_{h(x)}$.

Thus $X = \sum_{x \in S} (h(x)+1)$

$$E(X) = E\left(\sum_{x \in S} (h(x)+1)\right)$$

↘ linearity of expectation

$$= \sum_{x \in S} E(h(x)+1)$$

$$= \sum_{x \in S} E(h(x)) + 1 = \sum_{x \in S} 2 = 2n.$$ ∎

5. $E(h) \leq \log n + 1$

Define indicator r.v. $X_1, X_2, \ldots$

such that $X_i = \begin{cases} 1, & \text{if the list } L_i \text{ is non-empty (has elements other than dummy elements)} \\ 0, & \text{otherwise} \end{cases}$

Thus $h = \sum_{i=1}^{\infty} X_i$

Note that $X_i \leq 1$ (being 0-1 indicator r.v)

Thus, $E(X_i) \leq 1$. — (A)

Also $X_i \leq |L_i|$

Thus $E(X_i) \leq E(|L_i|) = n/2^i$ — (B)

Then $E(h) = E\left(\sum_{i=1}^{\infty} X_i\right)$

$= \sum_{i=1}^{\infty} E(X_i)$ ← linearity of Expectation

$= \sum_{i=1}^{\log n} E(X_i) + \sum_{i=\log n+1}^{\infty} E(X_i)$

$\leq \sum_{i=1}^{\log n} 1 + \sum_{i=\log n+1}^{\infty} \frac{n}{2^i}$ (by (A) and by (B))

$= \log n + \sum_{i=\log n+1}^{\infty} \frac{n}{2^i}$

Thus

$$E(h) \le \log n + \sum_{j=0}^{\infty} \frac{n}{2^{\log n + 1 + j}}$$

$$= \log n + \sum_{j=0}^{\infty} \frac{n}{2^{\log n}\, 2^{1+j}}$$

$$= \log n + \sum_{j=0}^{\infty} \frac{n}{n \cdot 2^{1+j}}$$

$$= \log n + \frac{1}{2} \sum_{j=0}^{\infty} \frac{1}{2^j}$$

$$= \log n + \frac{1}{2} \cdot 2 = 1 + \log n. \quad \blacksquare$$

Note that the expected # of nodes including the dummy nodes in all the lists will be at most $\underbrace{2n}_{\text{by } (4)} + \underbrace{\log n + 2}_{\text{by } (5)}$.

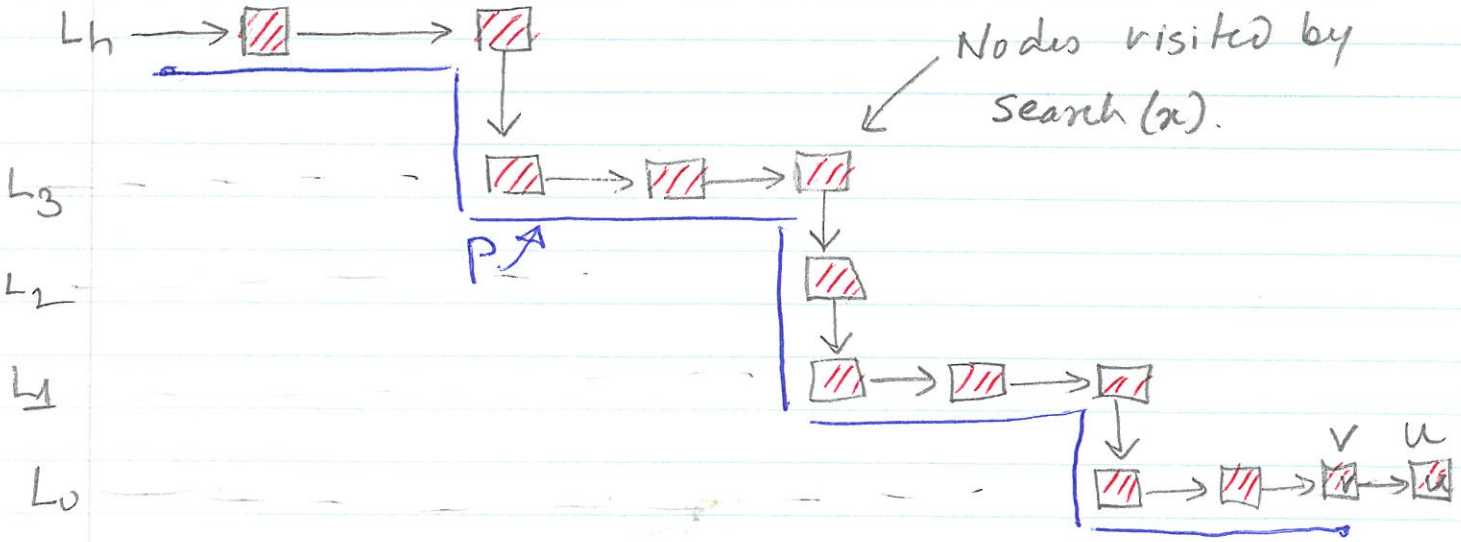
6. $E(N) \leq 2\log n + 5$

N = # nodes on the search path of algorithm SEARCH(x).

Let SEARCH(x) returns the node $u \in L_0$.

Search path starts at root-node and in each list L_i the path either moves towards right or moves to the list L_{i-1} till it arrives at node $u \in L_0$.

N = # nodes visited on this search path.



Let v be the node visited by search(x) just before reaching the destination node u.

Let P be the blue path, the subpath of search(x), terminating at node v.

Let M = # nodes on path P.

Note that $N = M+1$ and

$E(N) = E(M+1) = 1 + E(M)$

Thus if we show $E(M) \leq 2\log n + 4$ then we prove our claim.

We define a path P' — "Reverse of P" as follows:

- P' starts at v.
- At any node on P', the path P' goes up one level if this is possible, and goes left one node otherwise.

Note that $P \& P'$ are identical paths (except the reversal), thus P' has M nodes.

For $i \geq 0$, define r.v. M_i = # nodes on L_i for which P' goes left

thus $M = h + 1 + \sum_{i=0}^{h} M_i$

- h: accounts for P' moving up
- 1: root node
- $\sum_{i=0}^{h} M_i$: accounts for all other nodes in each of lists.

Thus,

$$E(M) = E\left(h + 1 + \sum_{i=0}^{h} M_i\right)$$

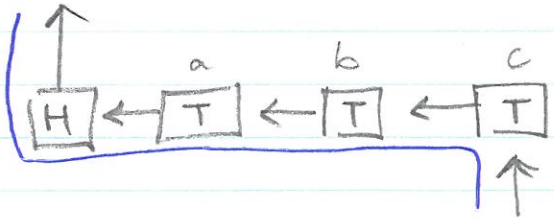
$$= E(h) + 1 + E\left(\sum_{i=0}^{h} M_i\right)$$

$E(h)$ — by 5 — $\log n + 1$

$E\left(\sum_{i=0}^{h} M_i\right)$ — ?

Consider M_i:

L_i:



At some point P' reaches L_i and it traverses left till either it - reaches the dummy node

- or reaches a node with "UP" pointer.

The reason it traverses left is that when we were deciding whether these elements will go to next level (L_{i+1}), the outcome of tosses for each of them were "Tails".

So M_i can be interpreted as # Tails obtained when flipping a fair coin until it comes up heads.

Infact $M_i \leq$ # Tails obtained when flipping a fair coin till it shows up heads.

(A) Thus $E(M_i) \leq 1$ (as E(#tosses to obtain Heads) = 2)

Also $M_i \leq$ # Nodes in list L_i

(B) Thus $E(M_i) \leq \frac{n}{2^i}$ by [3]

Now

$$E(M) = E(h) + 1 + \sum_{i=0}^{\log n} E(M_i) + \sum_{i=\log n+1}^{\infty} E(M_i)$$

(USE A) (USE B)

$$\leq \log n + 1 + 1 + \log n + 1 + \sum_{i=\log n+1}^{\infty} \frac{n}{2^i} \leq 2$$

$$= 2\log n + 3 + \frac{1}{2}\sum_{j=0}^{\infty} \frac{1}{2^j}$$

$$= 2\log n + 4.$$

Thus $E(M) \leq 2\log n + 4$ or $E(N) \leq 2\log n + 5$.