COMP2402
Abstract Data Types and Algorithms

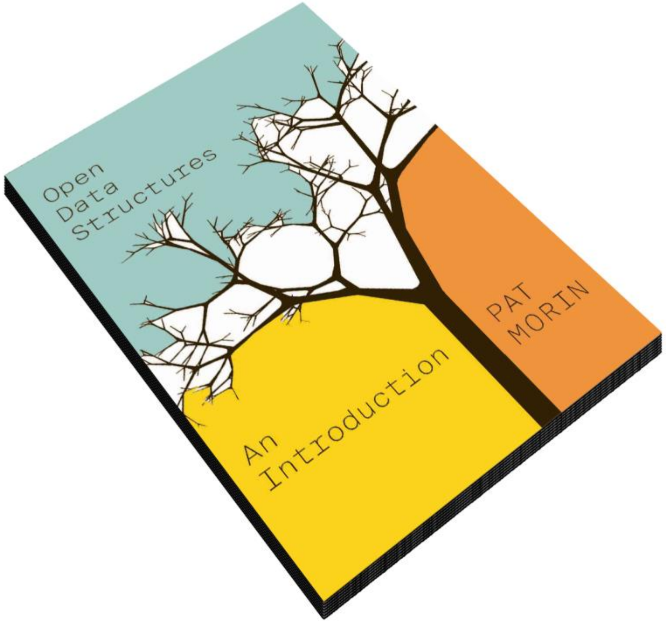
Skiplists

Reading Assignment

Open Data Structures in Java

by Pat Morin

Chapter 4



Contrasting Array and Linked List Performances

Get/Set near the Ends

Arrays:

Linked Lists:

Get/Set near the Middle

Arrays:

Linked Lists:

Add/Remove near the Ends

Arrays:

Linked Lists:

Add/Remove near the Middle

Arrays:

Linked Lists:

Contrasting Array and Linked List Performances

Get/Set near the Ends

Arrays: Fast

Linked Lists: Fast

Get/Set near the Middle

Arrays: Fast

Linked Lists: Slow

Add/Remove near the Ends

Arrays: Fast

Linked Lists: Fast

Add/Remove near the Middle

Arrays: Slow

Linked Lists: Fast

Introducing the Skiplist

the Skiplist is an Implementation of the List Interface

the Skiplist Supports:

- `set(i,x)` with Expected Time Complexity O(log n)
- `get(i)` with Expected Time Complexity O(log n)
- `add(i,x)` with Expected Time Complexity O(log n)
- `remove(i)` with Expected Time Complexity O(log n)

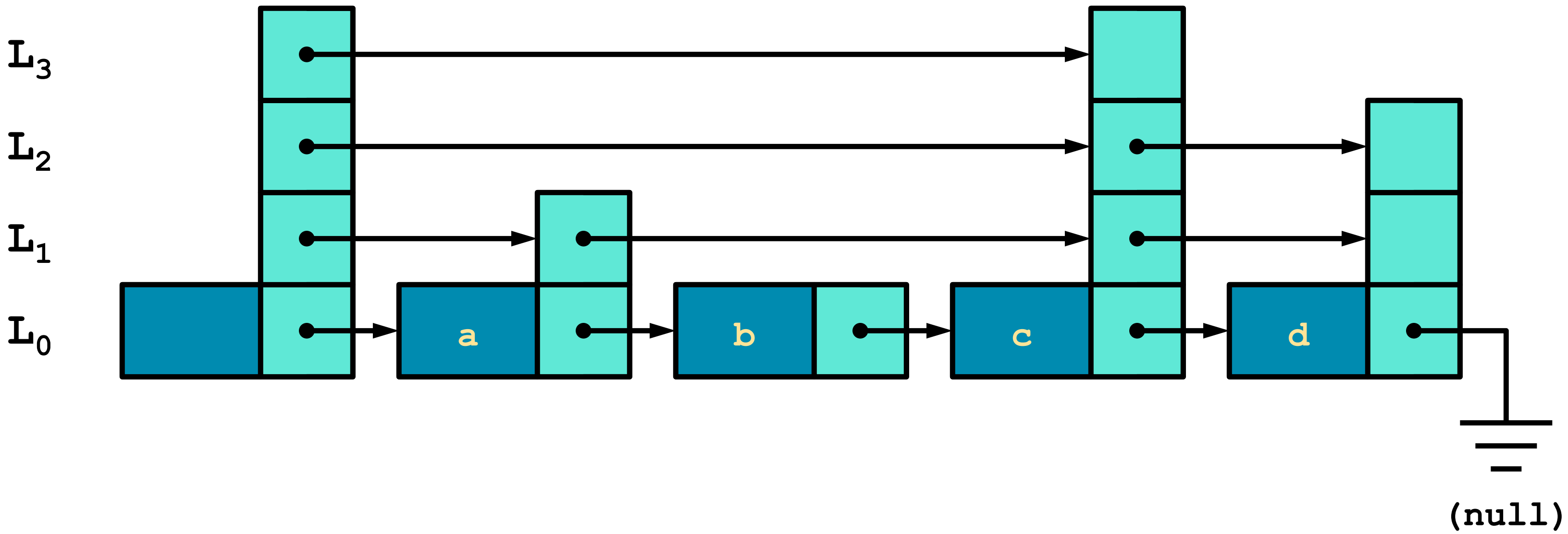
these are Expected Time Complexities because Skiplists are Built Stochastically (i.e., using Randomness)

Sample Skiplist

the Skiplist is a Sequence of Singly-Linked Lists

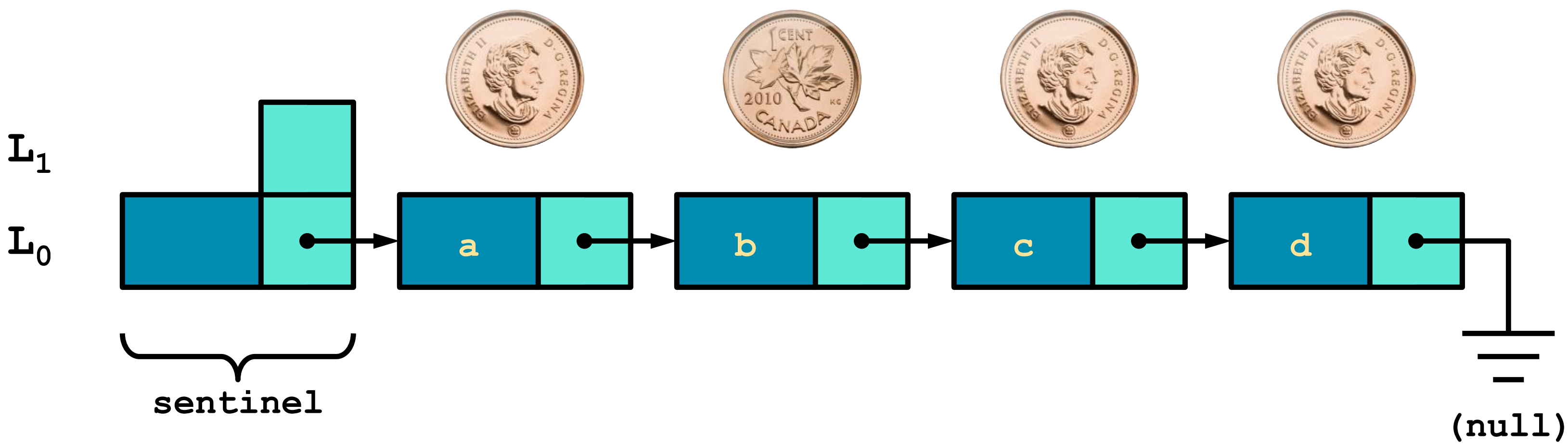
L_0, L_1, ..., L_h (where h is the Height of the Skiplist)

each List L_r Contains a Subset of the Elements of L_{r-1}



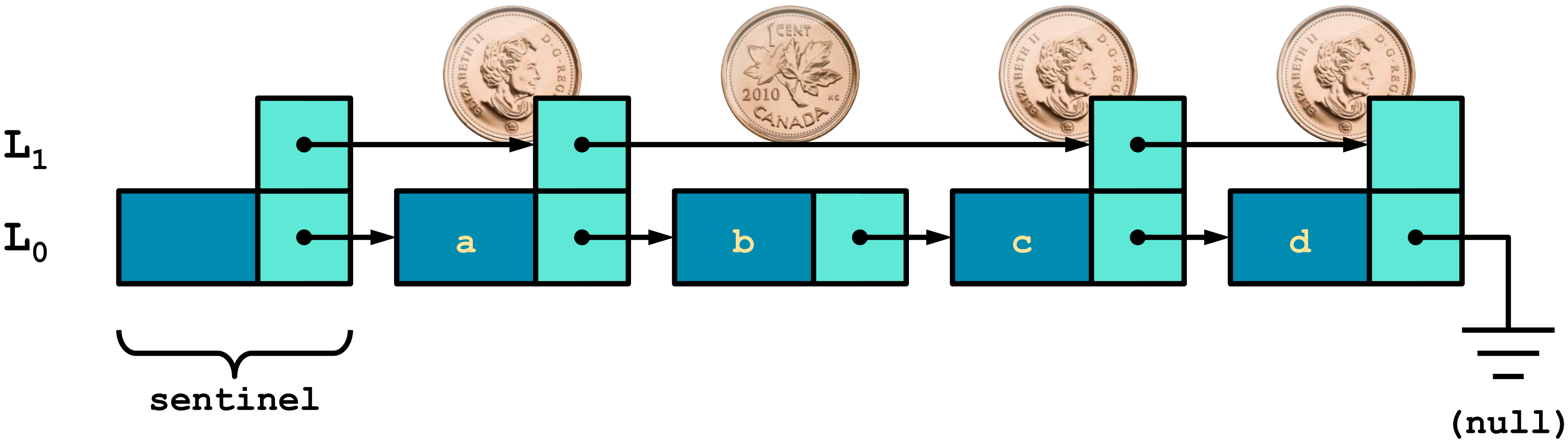
Constructing a Skiplist

from Singly-Linked List L_0 (of n items) Construct L_1 by Traversing L_0 and Generating a Uniformly-Distributed (Pseudo)Random Value for Every Node *(i.e., a "Coin-Flip")*



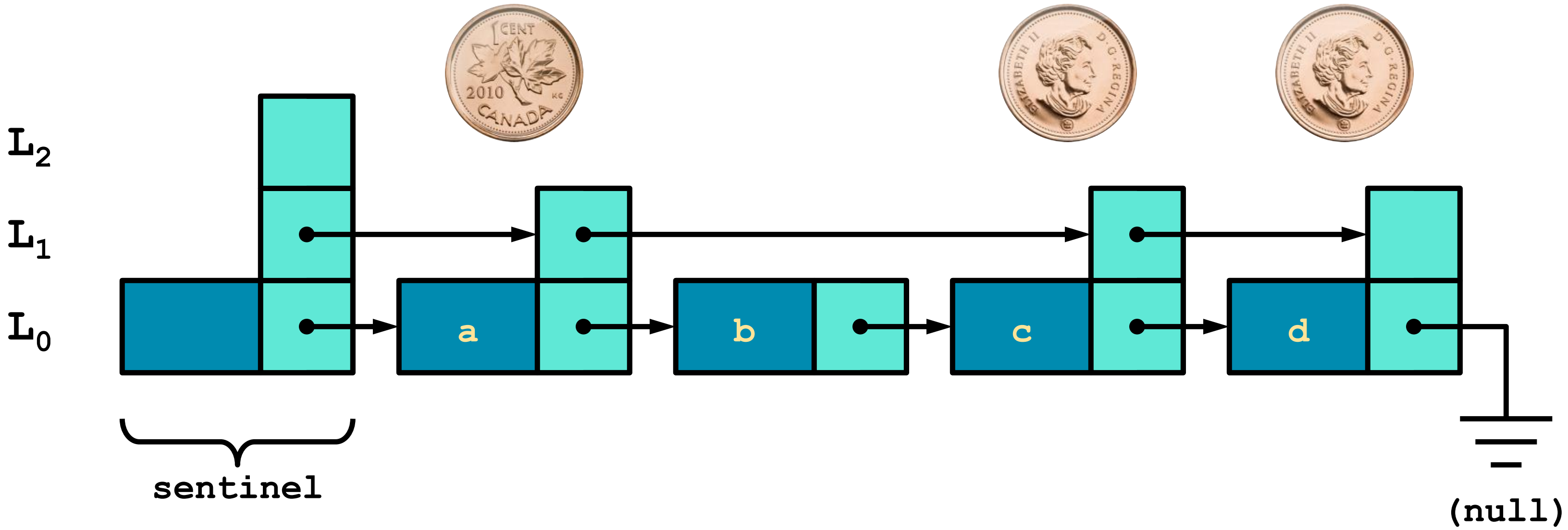
Constructing a Skiplist

from **Singly-Linked List** L_0 (of n items) **Construct** L_1 by **Traversing** L_0 and **Generating** a **Uniformly-Distributed (Pseudo)Random Value** for **Every Node** *(i.e., a "Coin-Flip")*



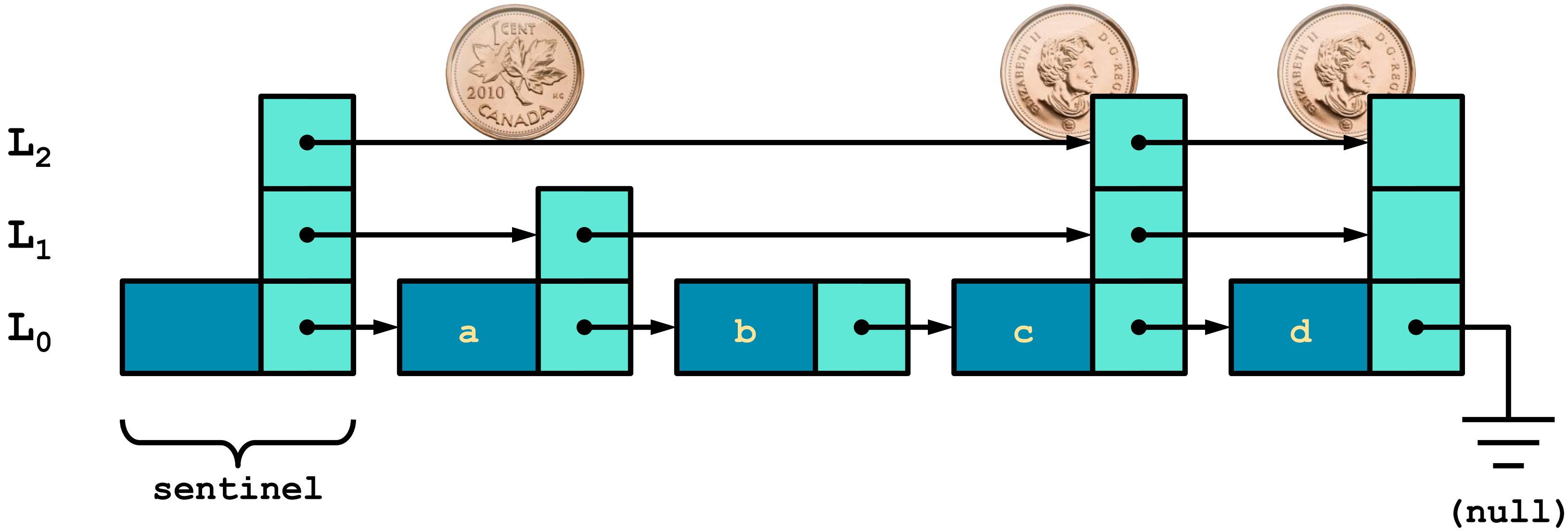
Constructing a Skiplist

from Singly-Linked List L_0 (of n items) Construct L_1 by Traversing L_0 and Generating a Uniformly-Distributed (Pseudo)Random Value for Every Node *(i.e., a "Coin-Flip")*



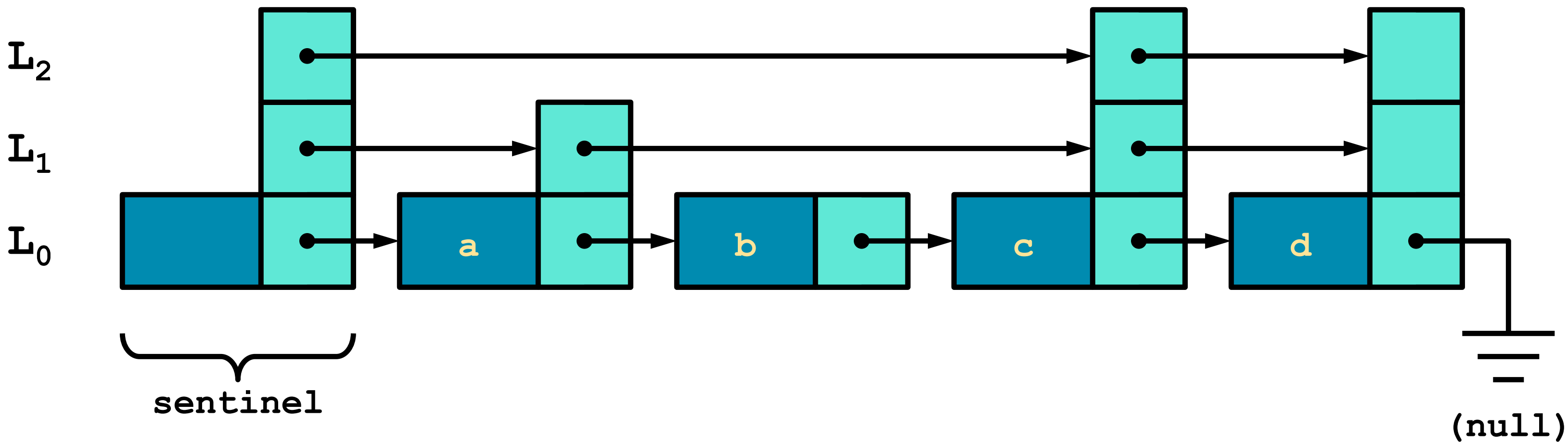
Constructing a Skiplist

from Singly-Linked List L_0 (of n items) Construct L_1 by Traversing L_0 and Generating a Uniformly-Distributed (Pseudo)Random Value for Every Node *(i.e., a "Coin-Flip")*



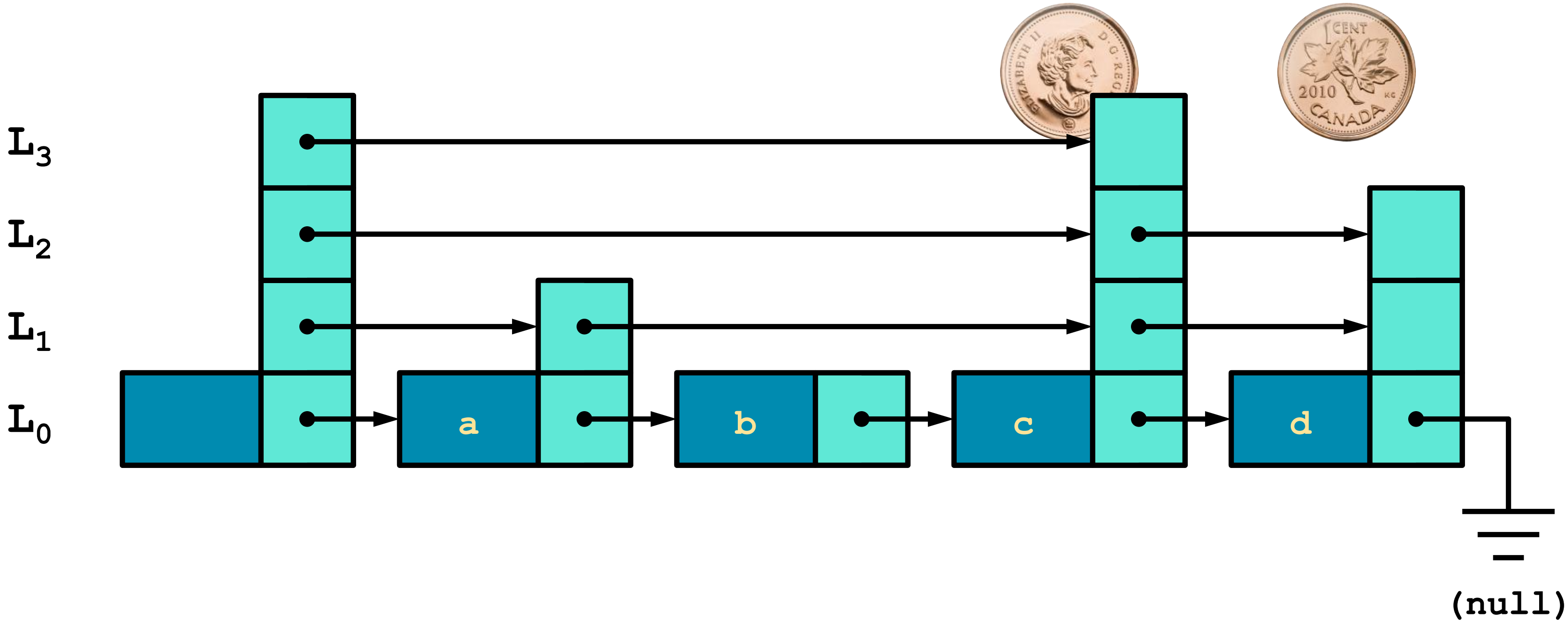
Constructing a Skiplist

from Singly-Linked List L_0 (of n items) Construct L_1 by Traversing L_0 and Generating a Uniformly-Distributed (Pseudo)Random Value for Every Node *(i.e., a "Coin-Flip")*

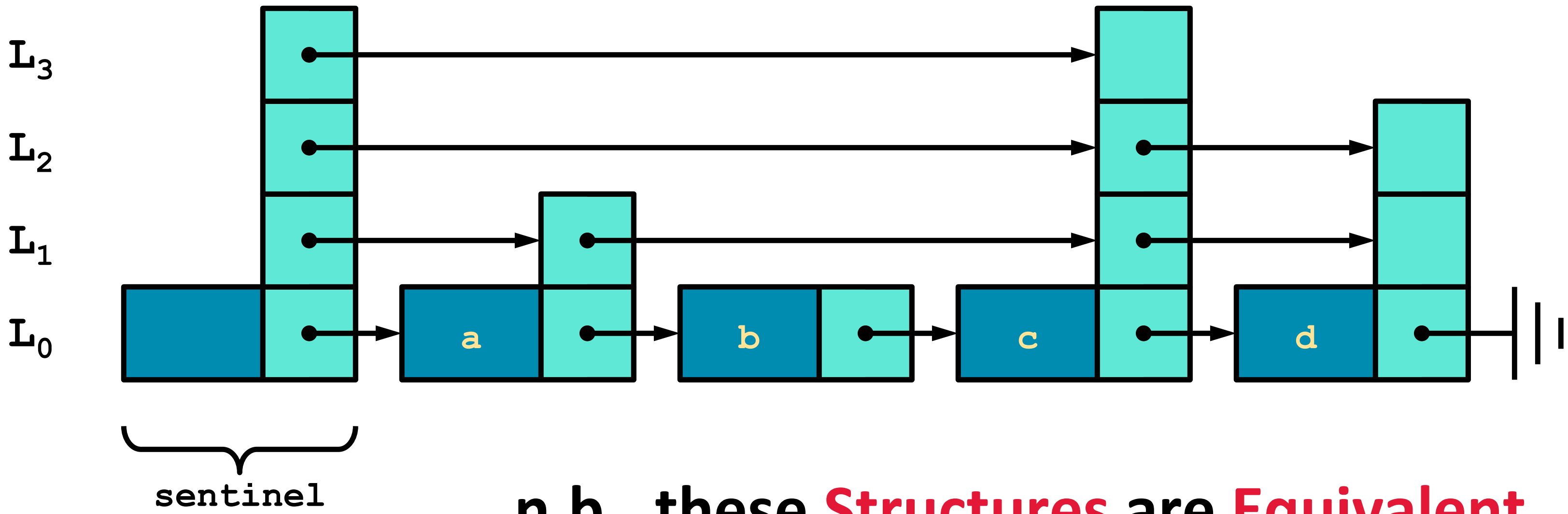


Constructing a Skiplist

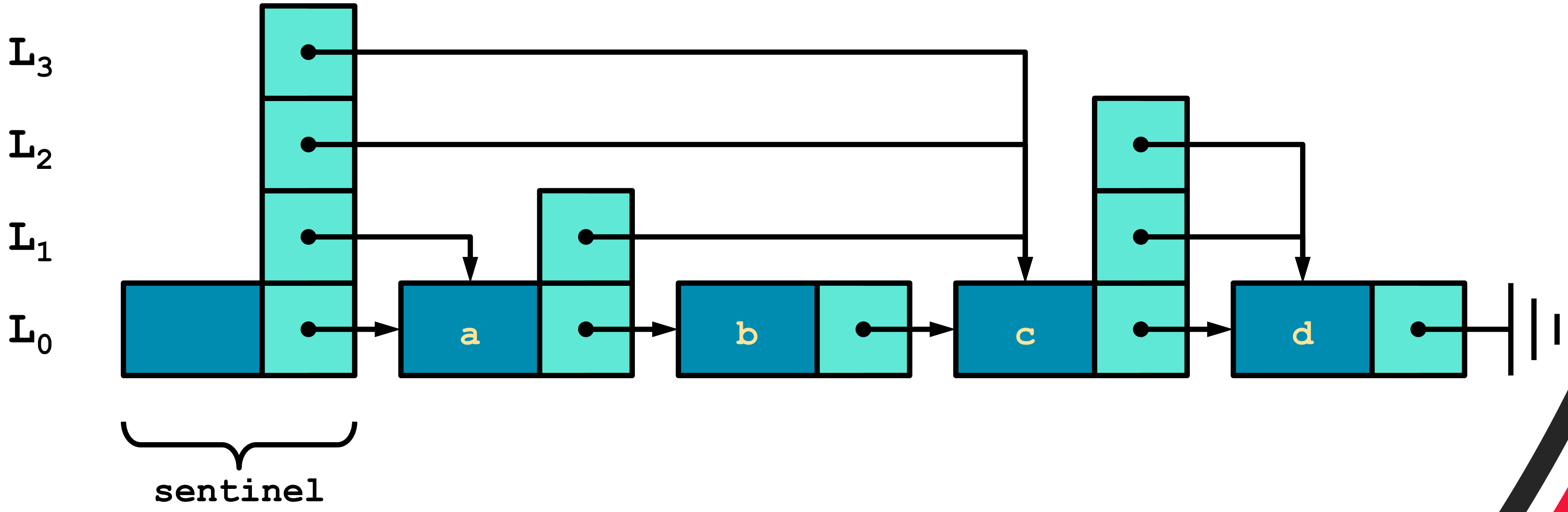
from Singly-Linked List L_0 (of n items) Construct L_1 by Traversing L_0 and Generating a Uniformly-Distributed (Pseudo)Random Value for Every Node *(i.e., a "Coin-Flip")*



Equivalent Structure, Alternative Representation



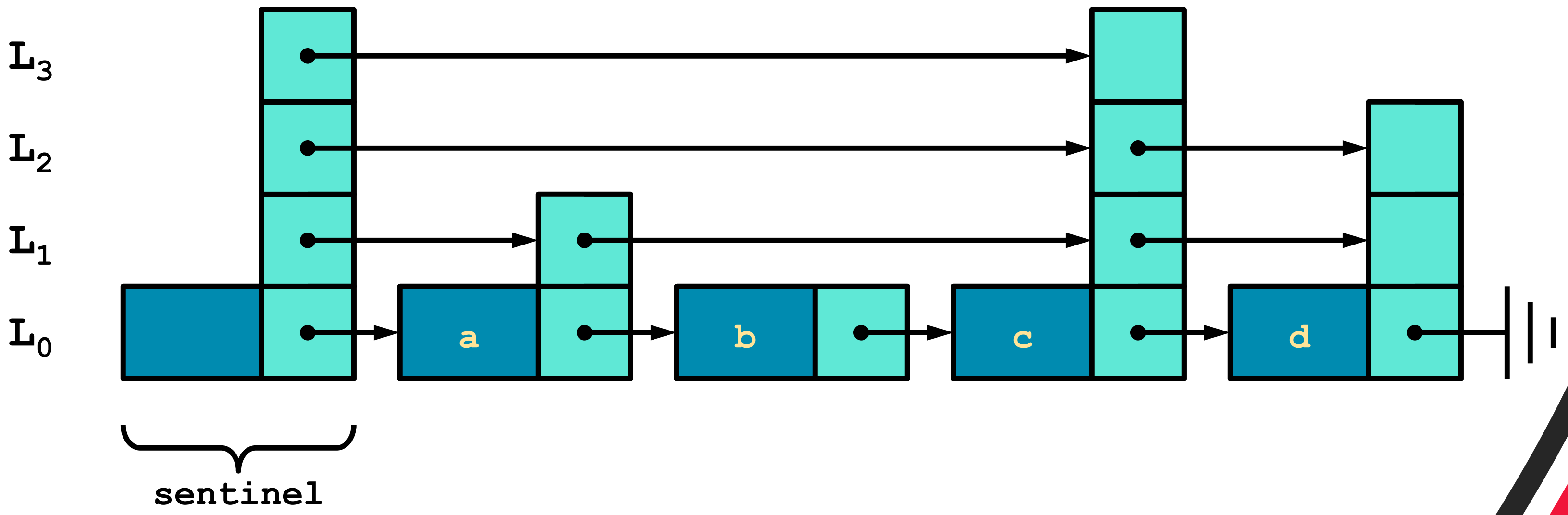
n.b., these Structures are Equivalent



Find Search Paths in Skiplists

there is a Short path, called the Search Path, from the `sentinel` (i.e., Dummy) Node to Any Other Node:

1. Start from the Top-Left (i.e., L_h, at `sentinel`)
2. If Moving Right would Overshoot Target, Move Down
3. Otherwise, Move Right

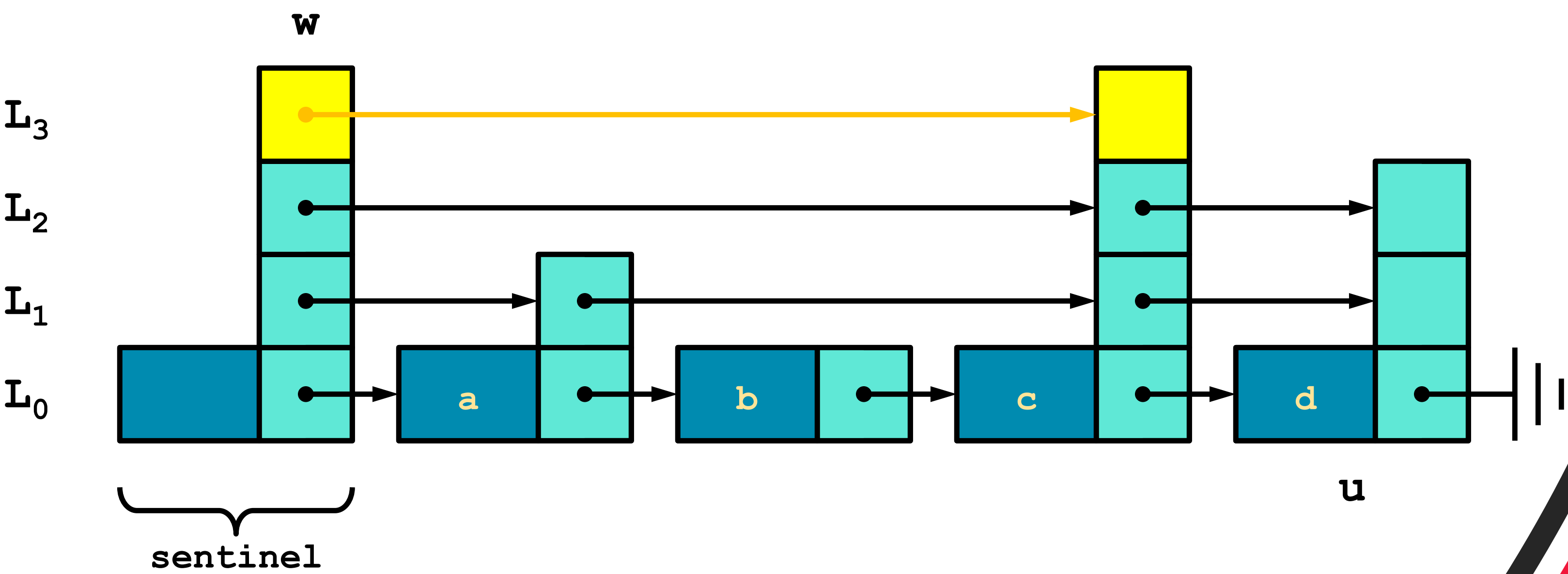


Find Search Paths in Skiplists

Find the Search Path for Node with Index `d`

(n.b., `u`/`w` denote Target/Pointer, respectively, and `h` = 3)

1. in L_3, `w.next` > `u` = False → `w` = `w.next`

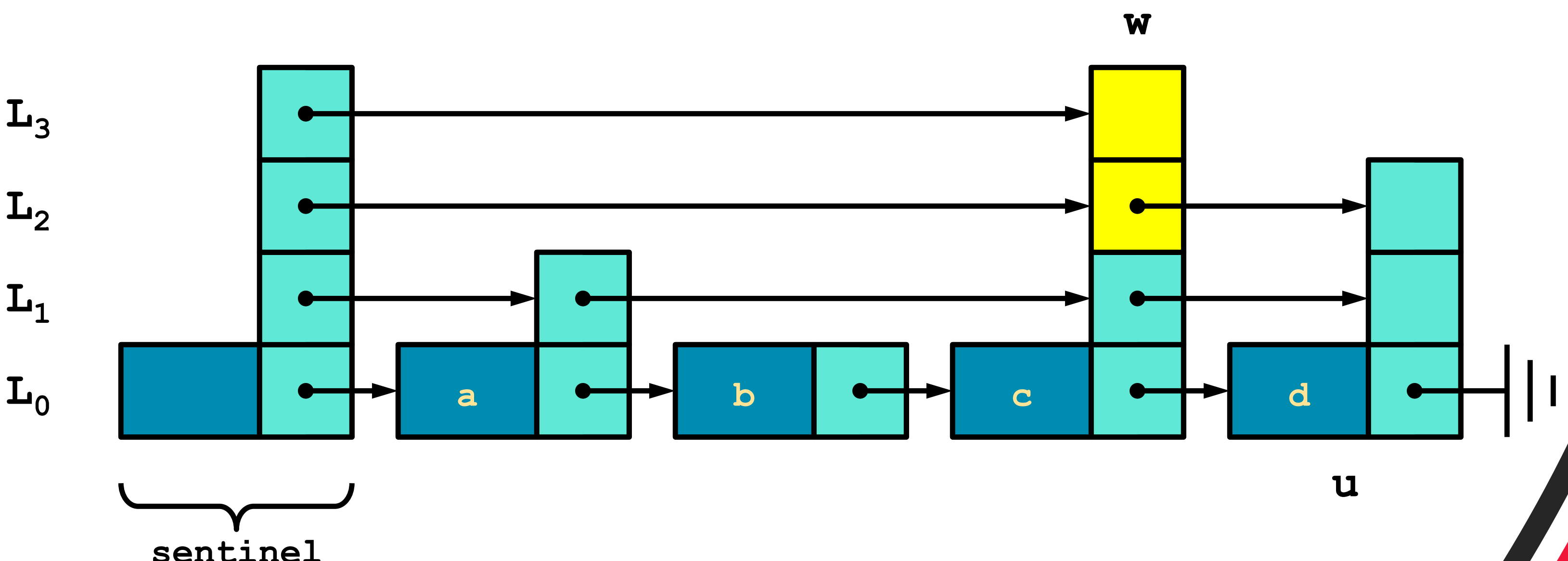


Find Search Paths in Skiplists

Find the Search Path for Node with Index `d`

(n.b., `u`/`w` denote Target/Pointer, respectively, and `h` = 3)

1. in L_3, `w.next` > `u` = False → `w` = `w.next`
2. in L_3, `w.next` is null, so move to L_2

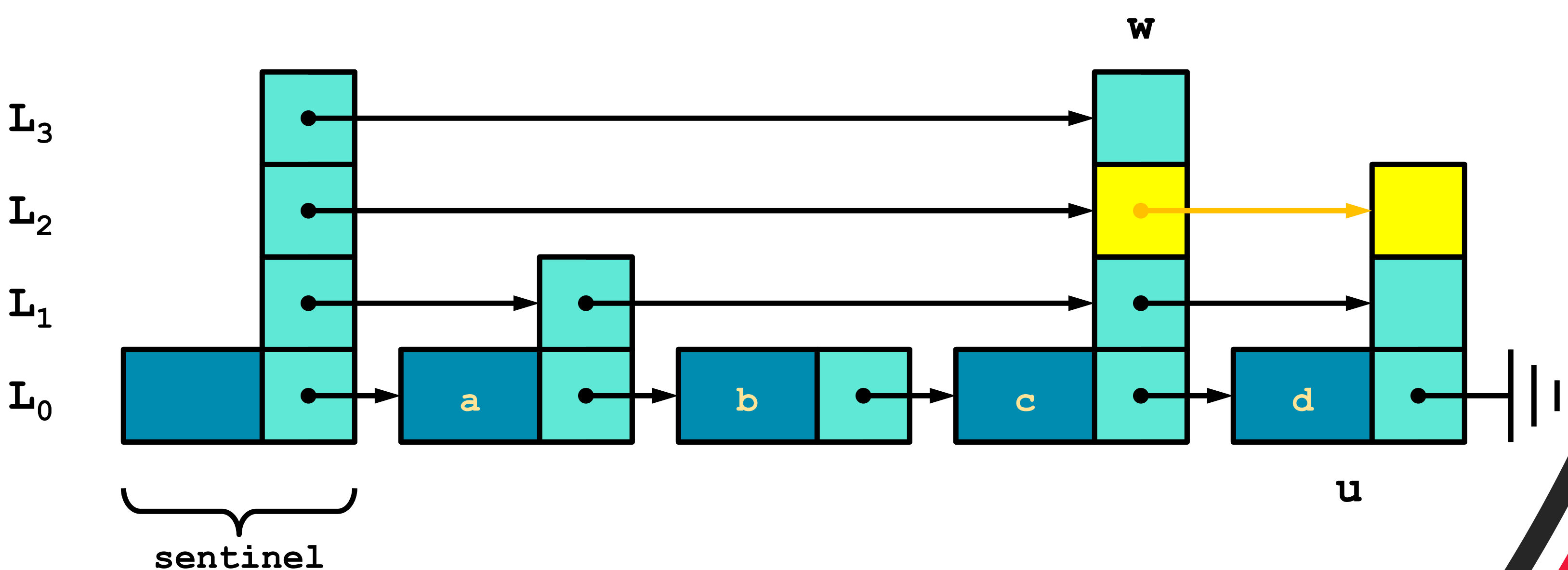


Find Search Paths in Skiplists

Find the Search Path for Node with Index `d`

(n.b., `u`/`w` denote Target/Pointer, respectively, and `h` = 3)

1. in L_3, `w.next` > `u` = False → `w` = `w.next`
2. in L_3, `w.next` is null, so move to L_2
3. in L_2, `w.next` > `u` = False → `w` = `w.next`



First Skiplist Lemma

Start Flipping a Coin and Don't Stop Until a Heads occurs

If T is the Number of Flips (including Last Flip to Heads)

then the Expected Value* of T (i.e., $E[T]$) is 2.

(i.e., the product of a value and the likelihood of observing that value)*

First Skiplist Lemma

E[Number of Flips Before Observing Heads on Attempt 1] is:

(p(Observing Heads Immediately) · 1) +

p(Not Observing Heads Immediately; i.e., Observing Tails) ·

1 + E[Number of Flips Before Observing Heads on Attempt 2]

First Skiplist Lemma

E[Number of Flips Before Observing Heads on Attempt 1] is:

(p(Observing Heads Immediately) · 1) +

(p(Not Observing Heads Immediately; i.e., Observing Tails) ·

1 + E[Number of Flips Before Observing Heads on Attempt 2])

$$\begin{aligned} \boldsymbol{E[T]} &= \boldsymbol{p \cdot 1 + (1-p) \cdot (1 + E[T])} \\ &= \boldsymbol{p + 1 + E[T] - p - p \cdot E[T]} \end{aligned}$$

$$\boldsymbol{p \cdot E[T] = 1}$$

$$\boldsymbol{E[T] = \frac{1}{p}}$$

First Skiplist Lemma

E[Number of Flips Before Observing Heads on Attempt 1] is:

(p(Observing Heads Immediately) · 1) +

p(Not Observing Heads Immediately; i.e., Observing Tails) ·

1 + E[Number of Flips Before Observing Heads on Attempt 2]

$$
\begin{aligned}
E[T] &= p \cdot 1 + (1-p) \cdot (1 + E[T]) \\
&= p + 1 + E[T] - p - p \cdot E[T] \\
p \cdot E[T] &= 1 \\
E[T] &= \frac{1}{p}
\end{aligned}
$$

...and if the p is $^{1}/_{2}$...

$$\frac{1}{p} = \frac{1}{^{1}/_{2}} = 2$$

Second Skiplist Lemma

the Number of Nodes in a Skiplist that Contains n Elements is Determined by a Stochastic Process*...

*** (i.e., it is random)**

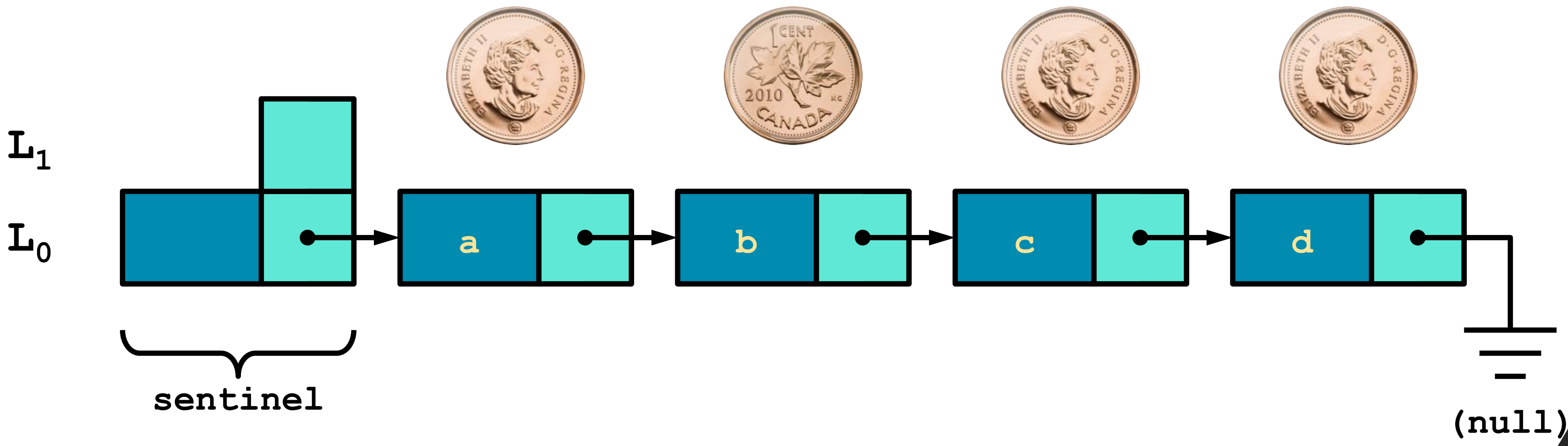
the Expected Number of Nodes in a Skiplist that Contains n Elements, not including the sentinel, is 2n

Second Skiplist Lemma

the **Expected Number of Nodes** in L_0 is n

since the **Subset** of L_0 that **Becomes** L_1 is **Determined** by n **Coin Flips**, the **Expected Number of Nodes** in L_1 is $^n/_2$

since the **Subset** of L_1 that **Becomes** L_2 is **Determined** by $^n/_2$ **Coin Flips**, the **Expected Number of Nodes** in L_2 is $^{^n/_2}/_2$



Second Skiplist Lemma

the Expected Number of Nodes in a Skiplist that

Contains n Elements, not including the sentinel, is

$$n + {}^{n}/_{2} + {}^{{}^{n}/_{2}}/_{2} + \cdots$$

$$= {}^{n}/_{2^0} + {}^{n}/_{2^1} + {}^{n}/_{2^2} + \cdots$$

$$= \sum_{r=0}^{\infty} {}^{n}/_{2^r}$$

Second Skiplist Lemma

the Expected Number of Nodes in a Skiplist that Contains n Elements, not including the sentinel, is

$$n + {}^{n}/_{2} + {}^{{}^{n}/_{2}}/_{2} + \cdots$$

$$= {}^{n}/_{2^0} + {}^{n}/_{2^1} + {}^{n}/_{2^2} + \cdots$$

$$= \sum_{r=0}^{\infty} {}^{n}/_{2^r}$$

$$= n \sum_{r=0}^{\infty} {}^{1}/_{2^r}$$

$$= 2n$$

Third Skiplist Lemma

Suppose we "Count" a List L_r Using an Indicator Variable I_r

(i.e., I_r has a value that is 1 if the list L_r exists, and 0 otherwise)

the Height of the Skiplist would then be the Sum Over All I_r

(i.e., a sum of 1's for every list L_r that exists)

Third Skiplist Lemma

if L_r Exists it has At Least One Node and I_r = 1

and if L_r Does Not Exist it has Zero Nodes and I_r = 0

Since I_r is Never More than the Length of the List L_r ...

$$E[I_r] \leq E[L_r.size] = {}^{n}/_{2^r}$$

Third Skiplist Lemma

if the Height h of the Skiplist is $h = \sum_{r=1}^{\infty} I_r$

$$E[h] = E\left[\sum_{r=1}^{\infty} I_r\right]$$

$$= \sum_{r=1}^{\infty} E[I_r]$$

$$= E[I_1] + E[I_2] + E[I_3] + \cdots + E[I_\infty]$$

Third Skiplist Lemma

we can (always) Divide this Sum at an Arbitrary Point*...

*(*so we choose a point – in this case log(n) – that will let us cancel out n)*

$$= E[I_1] + E[I_2] + E[I_3] + \cdots + E[I_\infty]$$

$$= E[I_1] + \cdots + E\left[I_{\lfloor \log(n) \rfloor}\right] + E\left[I_{\lfloor \log(n) \rfloor + 1}\right] + \cdots + E[I_\infty]$$

n.b., $\lfloor \log(n) \rfloor$ **times**

$$\leq 1 + 1 + \cdots + 1 + E\left[I_{\lfloor \log(n) \rfloor + 1}\right] + \cdots + E[I_\infty]$$

$$= \sum_{r=1}^{\lfloor \log(n) \rfloor} 1 + E\left[I_{\lfloor \log(n) \rfloor + 1}\right] + \cdots + E[I_\infty]$$

$$= \log(n) + E\left[I_{\lfloor \log(n) \rfloor + 1}\right] + \cdots + E[I_\infty]$$

Third Skiplist Lemma

$$= \log(n) + \sum_{r=\lfloor \log(n) \rfloor + 1}^{\infty} E[I_r]$$

$$\leq \log(n) + \frac{n}{2^{\lfloor \log(n) \rfloor + 1}} + \frac{n}{2^{\lfloor \log(n) \rfloor + 2}} + \frac{n}{2^{\lfloor \log(n) \rfloor + 3}} \dots$$

$$= \log(n) + \frac{n}{2^{\lfloor \log(n) \rfloor} \cdot 2^1} + \frac{n}{2^{\lfloor \log(n) \rfloor} \cdot 2^2} + \frac{n}{2^{\lfloor \log(n) \rfloor} \cdot 2^3} \dots$$

$$= \log(n) + \frac{n}{n \cdot 2^1} + \frac{n}{n \cdot 2^2} + \frac{n}{n \cdot 2^3} \dots$$

$$= \log(n) + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} \dots$$

Third Skiplist Lemma

$$= \log(n) + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} \ldots$$

$$= \log(n) + \sum_{r=1}^{\infty} \frac{1}{2^r}$$

$$\leq \log(n) + \sum_{r=0}^{\infty} \frac{1}{2^r}$$

$$= \log(n) + 2$$

Q.E.D.

Fourth Skiplist Lemma (4.6 in the Text)

the Expected Length of a Search Path is

the Distance from Top to Bottom (i.e., the Height)

+

the Expected Number of "Steps" through the Linked Lists

Since the Number of Steps through List $\mathrm{L_r}$

Cannot possibly Be Longer than $\mathrm{L_r}$

$$E[\boldsymbol{S}_r] \leq E[|\boldsymbol{L}_r|] = \frac{\boldsymbol{n}}{\mathbf{2}^r}$$

Fourth Skiplist Lemma (4.6 in the Text)

$$E[S] = E\left[h + \sum_{r=0}^{\infty} S_r\right]$$

$$= E[h] + \sum_{r=0}^{\infty} E[S_r]$$

$$= E[h] + \sum_{r=0}^{\lfloor \log(n) \rfloor} E[S_r] + \sum_{r=\lfloor \log(n) \rfloor + 1}^{\infty} E[S_r]$$

Fourth Skiplist Lemma (4.6 in the Text)

$$\leq E[h] + \sum_{r=0}^{\lfloor \log(n) \rfloor} 1 + \sum_{r=\lfloor \log(n) \rfloor + 1}^{\infty} \frac{n}{2^r}$$

$$\leq E[h] + \sum_{r=0}^{\lfloor \log(n) \rfloor} 1 + \sum_{r=0}^{\infty} \frac{1}{2^r}$$

$$= E[h] + (1 + \log(n)) + \sum_{r=0}^{\infty} \frac{1}{2^r}$$

$$\leq E[h] + (1 + \log(n)) + 2$$

$$\leq (\log(n) + 2) + (1 + \log(n)) + 2$$

$$= 2\log(n) + 5$$

Fourth Skiplist Lemma (4.6 in the Text)

the **Expected Size** of a **Skiplist** that **Contains** n **Elements** is

$$O(n)$$

the **Expected Length** of a **Search Path** for an **Arbitrary Element** is

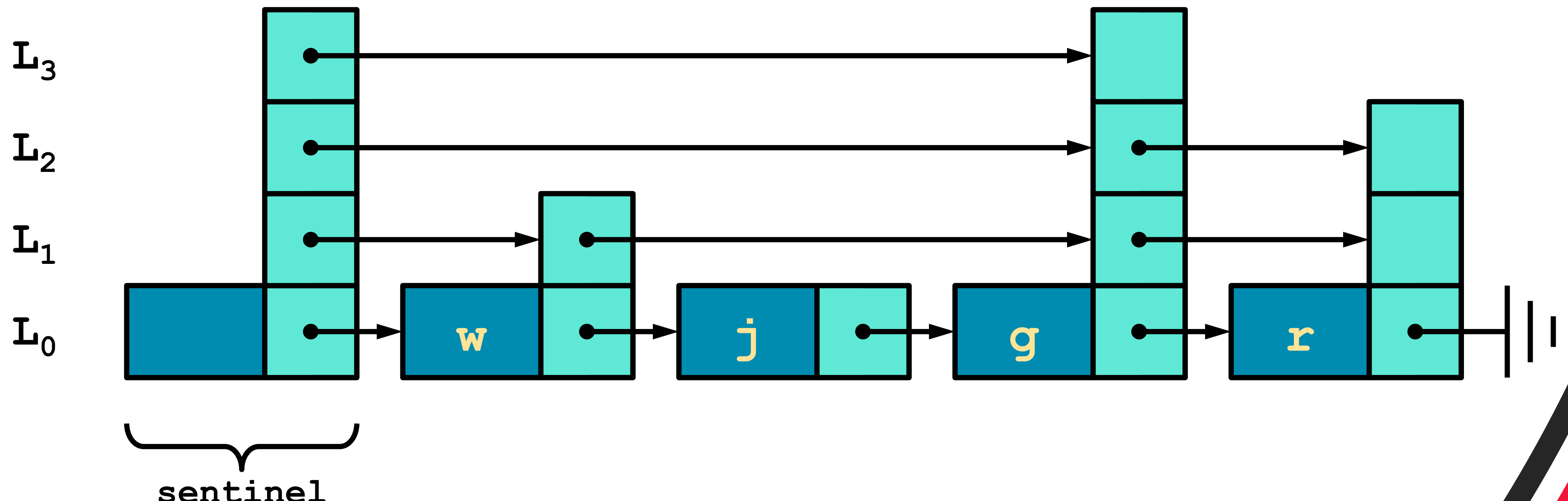
$$2\log(n) + O(1)$$

Search Paths in Skiplists, Revisited

the Search Path from the `sentinel` to Any Other Node can be Found with the Following Algorithm...

1. Start from the Top-Left (i.e., L_h , at `sentinel`)
2. If Moving Right would Overshoot Target, Move Down
3. Otherwise, Move Right

...If and Only If you can Detect having Overshot!



Applications of Skiplists

"SkiplistSSet" Implements the Sorted Set Interface and:

- `add(x)` with Time Complexity O(log n)
- `remove(x)` with Time Complexity O(log n)
- `find(x)` with Time Complexity O(log n)

"SkiplistList" Implements the List Interface and:

- `get(i)` with Time Complexity O(log n)
- `set(i, x)` with Time Complexity O(log n)
- `add(i, x)` with Time Complexity O(log n)
- `remove(i)` with Time Complexity O(log n)

Finding a Node in a Sorted Set

```
T find(T x) {
    Node<T> u = findPredNode(x);
    if (u.next[0] == null) {
        return null;
    } else {
        return u.next[0].x;
    }
}

Node<T> findPredNode(T x) {
    Node<T> u = sentinel;
    int r = h;                      // i.e., the topmost list
    while (r >= 0) {
        while (u.next[r] != null
                && compare(u.next[r].x, x) < 0) {
            u = u.next[r];          // go right
        }
        r--;                        // go down
    }
    return u;
}
```

Fourth Skiplist Lemma, Revisited

the Expected Length of a Search Path is

the Distance from Top to Bottom (i.e., the Height)

+

the Expected Number of "Steps" through the Linked Lists

$$2\log(n) + O(1)$$

$$\downarrow$$

$$O(\log(n))$$

this is also the Expected Running Time for `find(x)`

Locating a Node in a Random-Access List

If the Elements of the Collection are Not Sorted,
How Could one Detect if the Target Node was Overshot?

Traversing a Linked List to Find the ith Node is Easy
Because Every `next` Refers to a Node that is 1 Unit Away*
**(n.b., this isn't true in a skiplist)*

to Implement an Efficient Random-Access* List
it is Necessary to Define the Length of an Edge in List L_r
**(n.b., contrast with sequential access)*

Random Access List Nodes (in a Skiplist)

```
class Node {

    T x;
    Node[] next;
    int[] length;

    Node(T ix, int h) {
        x = ix;
        next = (Node[])Array.newInstance(Node.class, h+1);
        length = new int[h+1];
    }

    int height() {
        return next.length - 1;
    }

}
```

Locating a Node in a Random-Access List

if we are Currently at a Node that is at Position j in L_0 Following an Edge of Length m, leads to a Node whose Position (relative to list L_0) is $j+m$

thus, it is Possible to Track the Current Position While Following a Search Path

Update the Search Algorithm to Consider that If we are Currently at Node u in L_r

If the j + Length of Edge $\texttt{u.next[r]} < \texttt{i}$, Go Right Otherwise, Go Down (to List L_{r-1})

Adding an Element in a SkiplistList

when Adding a New Node to a Skiplist, it is Necessary to Determine How Many Times it will be Promoted

as an Alternative to Flipping Coins, Generate a Random Integer and Count the Number of Trailing Ones* that appear in the Binary Representation of that integer

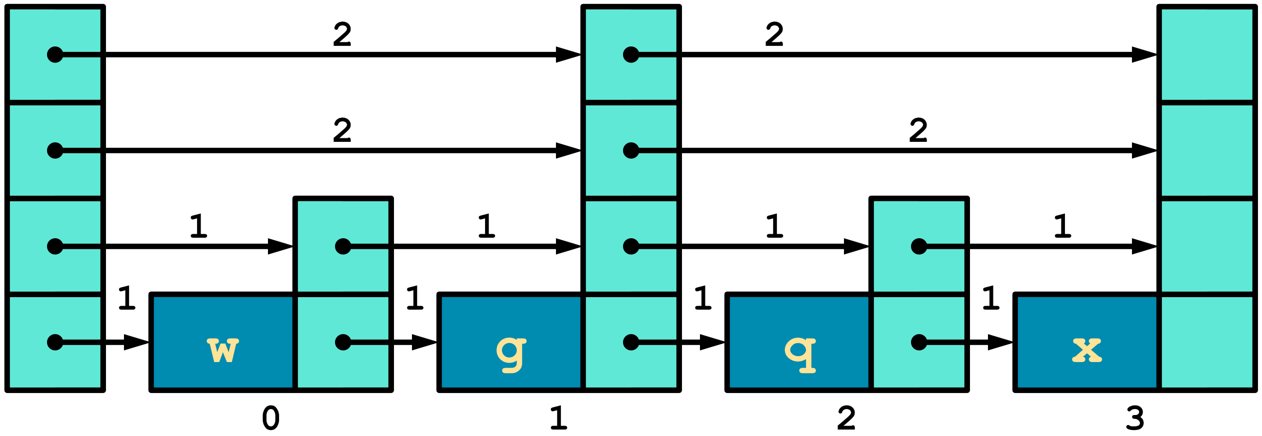
**(i.e., the number of consecutive 1's from right to left)*

the textbook refers to this* as `pickHeight`

**(i.e., determining how often a new node will be promoted)*

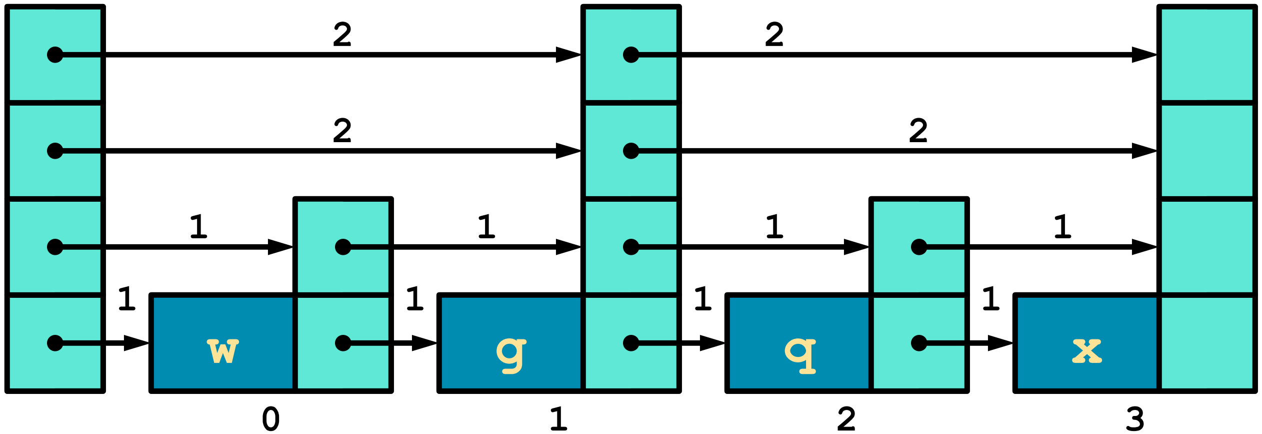
Adding an Element in a SkiplistList

when Adding it is also Necessary to Update Edge Lengths



Adding an Element in a SkiplistList

when Adding it is also Necessary to Update Edge Lengths



`insert(2, 'r')`