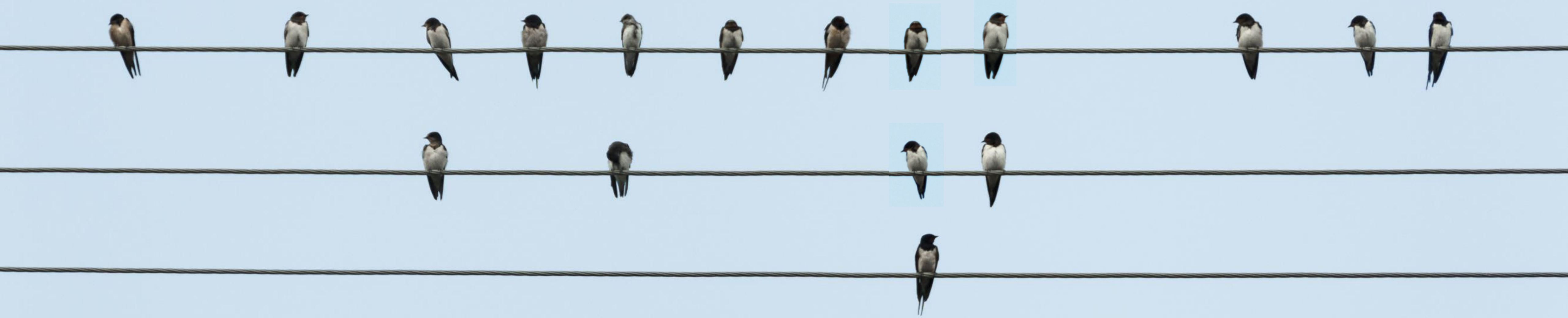
Carleton University



COMP 2402

SkipLists

Alina Shaikhet

SkipList data structure

SkipList is a beautiful **randomized** data structure.

It uses **random coin tosses** to determine the **height** of the newly added element.

Using a **SkipList** we can implement:

- a **List** that has $\boldsymbol{O(\log n)}$ **expected** runtime implementations of

  get($\boldsymbol{i}$), set($\boldsymbol{i, x}$), add($\boldsymbol{i, x}$), remove($\boldsymbol{i}$).

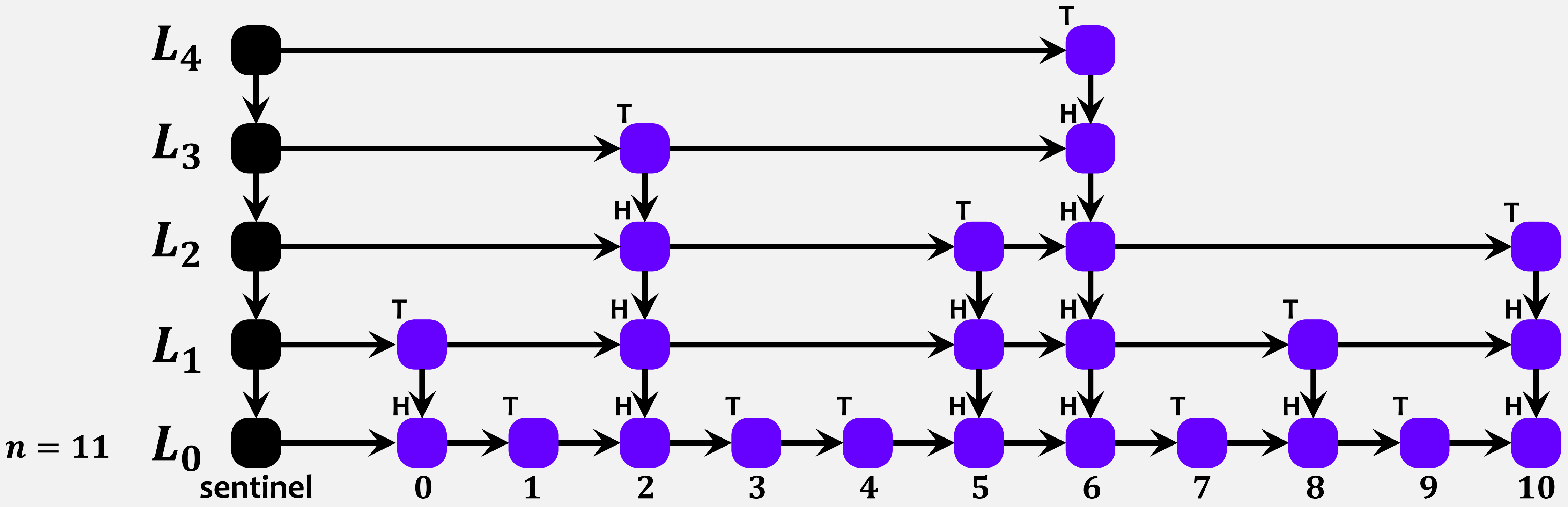
  (not only near ends of the list but also near its middle)

- an **SSet** (Sorted Set) in which all operations run in $\boldsymbol{O(\log n)}$ **expected** time.

SkipList

A SkipList is a sequence of singly-linked lists $L_0, L_1, \ldots, L_h$.
Each list L_r contains a **subset** of the items in L_{r-1}.

- we start with the input list L_0 that contains n items;
- we construct L_1 from L_0; L_2 from L_1, and so on;
- the items in L_r are obtained by **tossing a coin** for each element, x, in L_{r-1} and including x in L_r if the coin turns up as **heads.**
- this process ends when we create a list L_r that is **empty.**

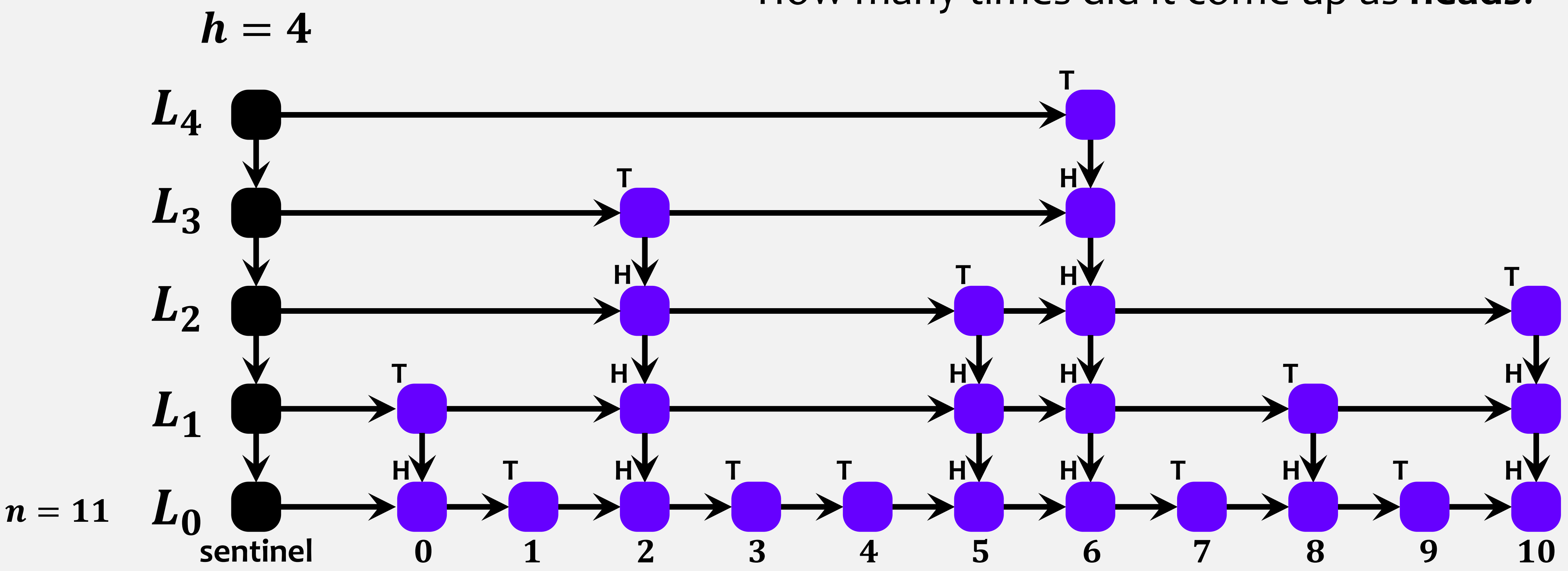


SkipList

The **height** of an element x in a **SkipList** is the largest value r such that x appears in L_r.

The **height** (h) of a **SkipList** is the height of its tallest node.

Toss a coin repeatedly until it comes up as **tails.**
How many times did it come up as **heads**?

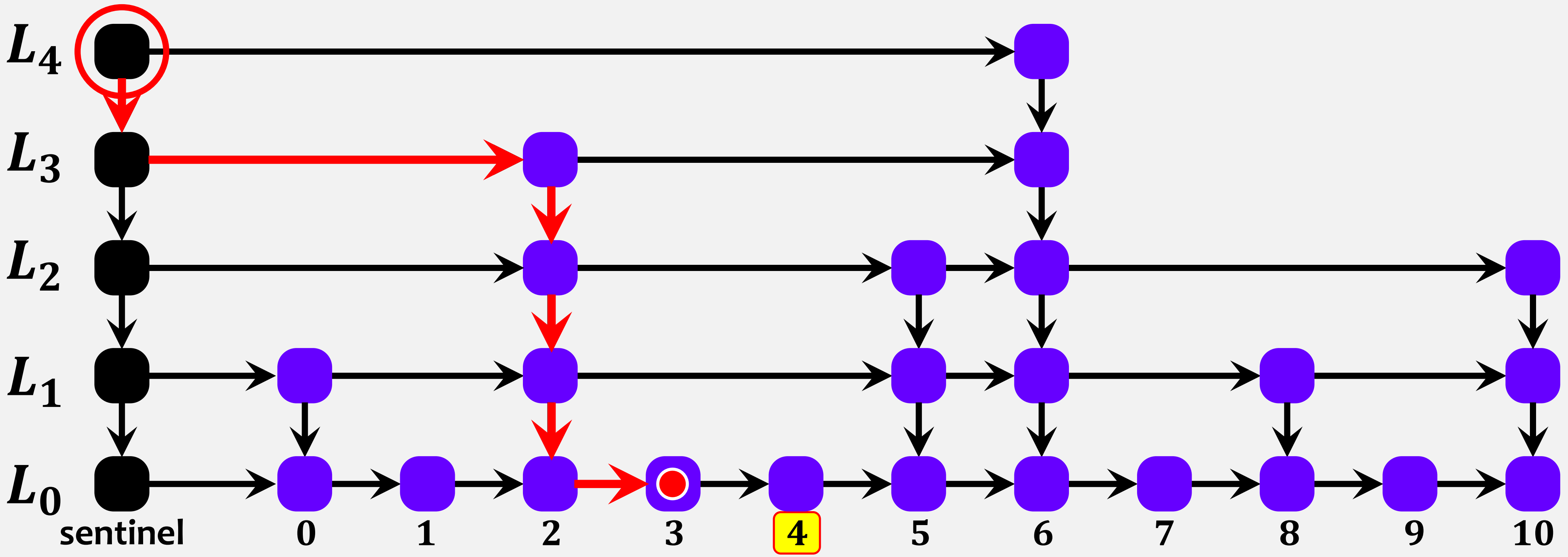


SkipList

There is a short path, called the **search path**, from the sentinel in L_h to every node in L_0.

To construct a search path for a node u:
Start at the top left corner of the SkipList (the sentinel in L_h) and always go **right** unless that would overshoot u, in which case you should take a step **down** into the list below.

a search path for a node u stops **before** u.

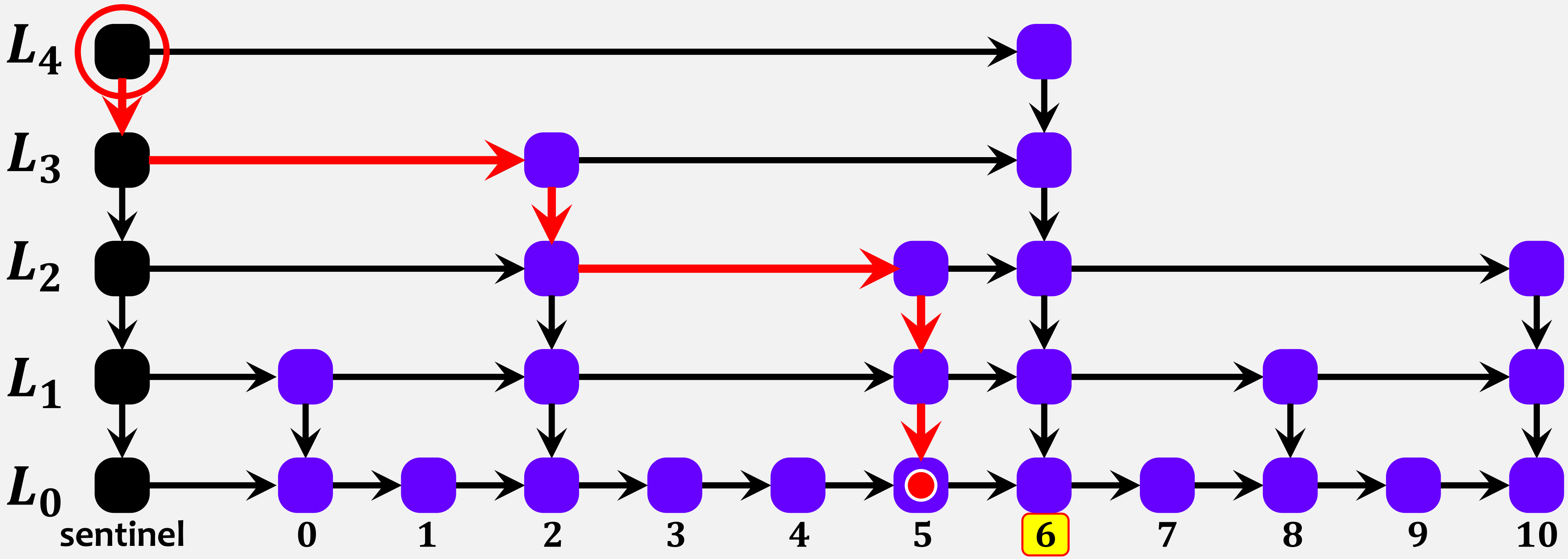


SkipList

There is a short path, called the **search path**, from the sentinel in L_h to every node in L_0.

To construct a search path for a node u:
Start at the top left corner of the SkipList (the sentinel in L_h) and always go **right** unless that would overshoot u, in which case you should take a step **down** into the list below.

a search path for a node u stops **before** u.

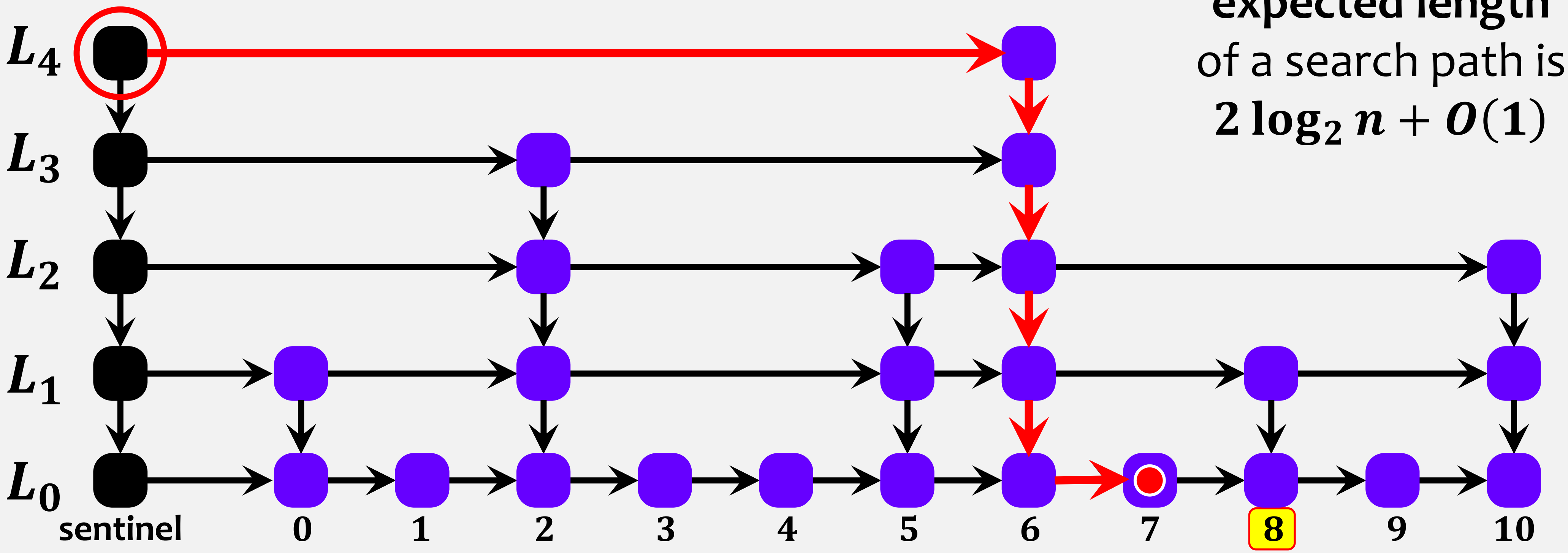


SkipList

There is a short path, called the **search path**, from the sentinel in L_h to every node in L_0.

To construct a search path for a node u:
Start at the top left corner of the SkipList (the sentinel in L_h) and always go **right** unless that would overshoot u, in which case you should take a step **down** into the list below.

a search path for a node u stops **before** u.



expected length of a search path is $2\log_2 n + O(1)$

SkipList

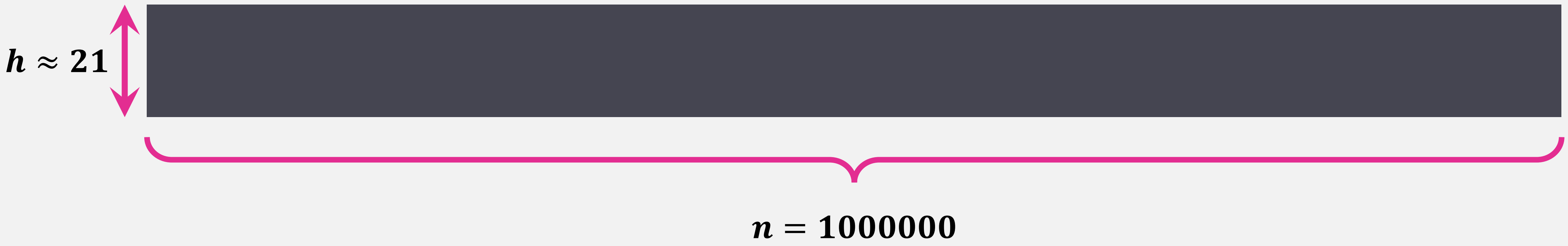
SkipLists are very long and skinny.

$$y = \log_b n$$

$$b^y = b^{\log_b n} = n$$

expected length of a search path is $2\log_2 n + O(1)$

$$2^{20} = 1,048,576$$



Lemma 4.1

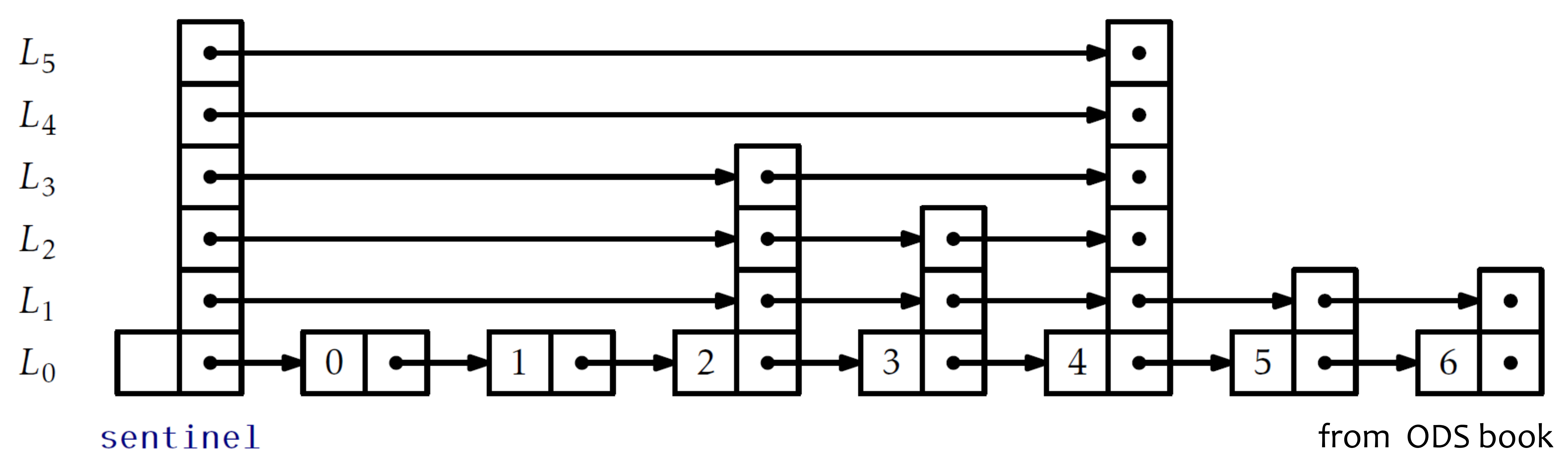
For any SkipList $\boldsymbol{L}$ containing $\boldsymbol{n}$ elements in $\boldsymbol{L_0}$, and any node $\boldsymbol{u}$ in $\boldsymbol{L_0}$, the **expected length** of the search path for $\boldsymbol{u}$ is at most $\underbrace{\mathbf{2\log_2 \boldsymbol{n} + \boldsymbol{O}(1)}}_{=\,\boldsymbol{O}(\log \boldsymbol{n})}$.

SkipList – space-efficient implementation

A node $\boldsymbol{u}$ consists of

- a data value $\boldsymbol{x}$, and
- an array **next** of pointers.

$\boldsymbol{u}$.**next**$[\boldsymbol{i}]$ points to $\boldsymbol{u}$'s successor in the list $\boldsymbol{L_i}$.
In this way, the data $\boldsymbol{x}$ in a node is referenced only once, even though $\boldsymbol{x}$ may appear in several lists.



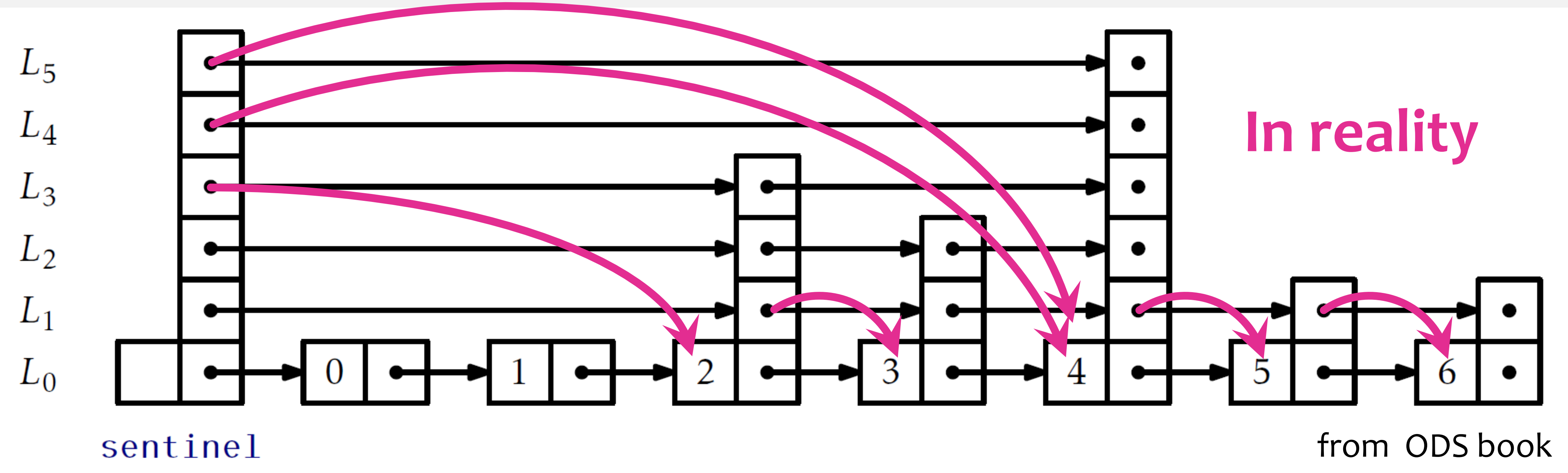
from ODS book

SkipList – space-efficient implementation

A node u consists of

- a data value x, and
- an array **next** of pointers.

$\boldsymbol{u}$**.next**$[\boldsymbol{i}]$ points to $\boldsymbol{u}$'s successor in the list $\boldsymbol{L_i}$.
In this way, the data $\boldsymbol{x}$ in a node is referenced only once, even though $\boldsymbol{x}$ may appear in several lists.



Alina Shaikhet – COMP2402 – Carleton University

Sorted Set (SSet)

- add($\boldsymbol{x}$),
- remove($\boldsymbol{x}$),
- find($\boldsymbol{x}$) – find the **smallest** value that is $\geq \boldsymbol{x}$.

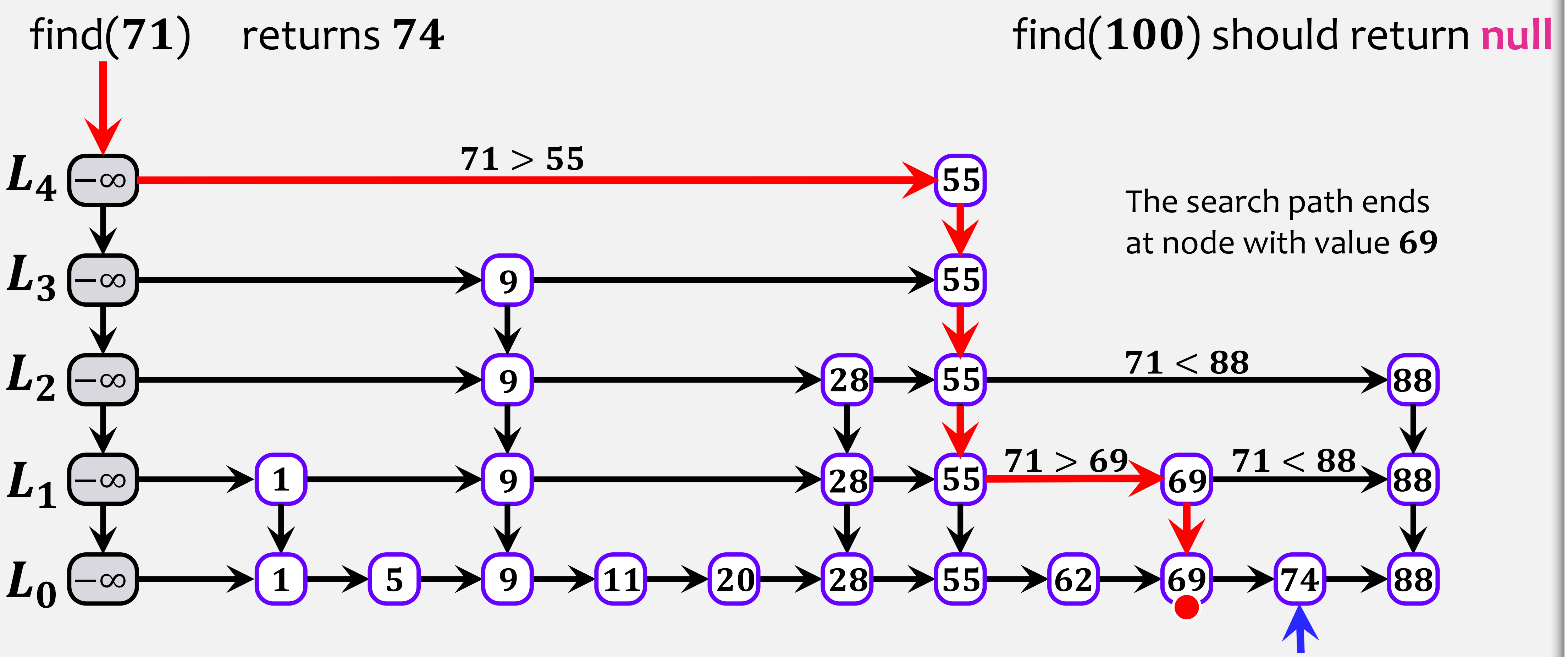
A **SkiplistSSet** uses a skiplist structure to implement the **SSet** interface

The list $\boldsymbol{L_0}$ stores the elements of the **SSet** in sorted order.

A **SkiplistSSet** supports the operations add($\boldsymbol{x}$), remove($\boldsymbol{x}$), and find($\boldsymbol{x}$) in $\boldsymbol{O(\log n)}$ expected time per operation.

SSet – find(x)

The find(x) method works by following the search path for the smallest value y such that $y \geq x$.



Alina Shaikhet – COMP2402 – Carleton University

SSet – find(x)

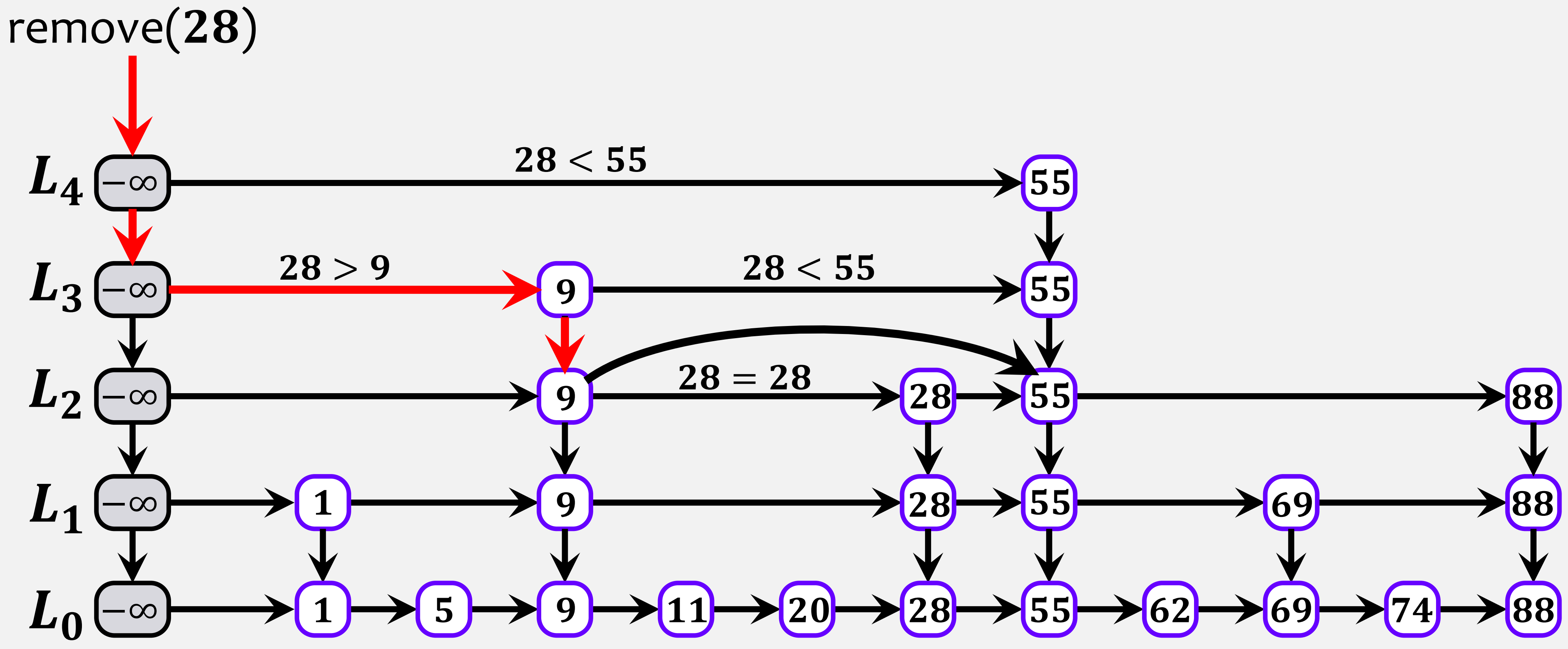
The expected running time is $\boldsymbol{O}(\log \boldsymbol{n})$.

SkiplistSSet

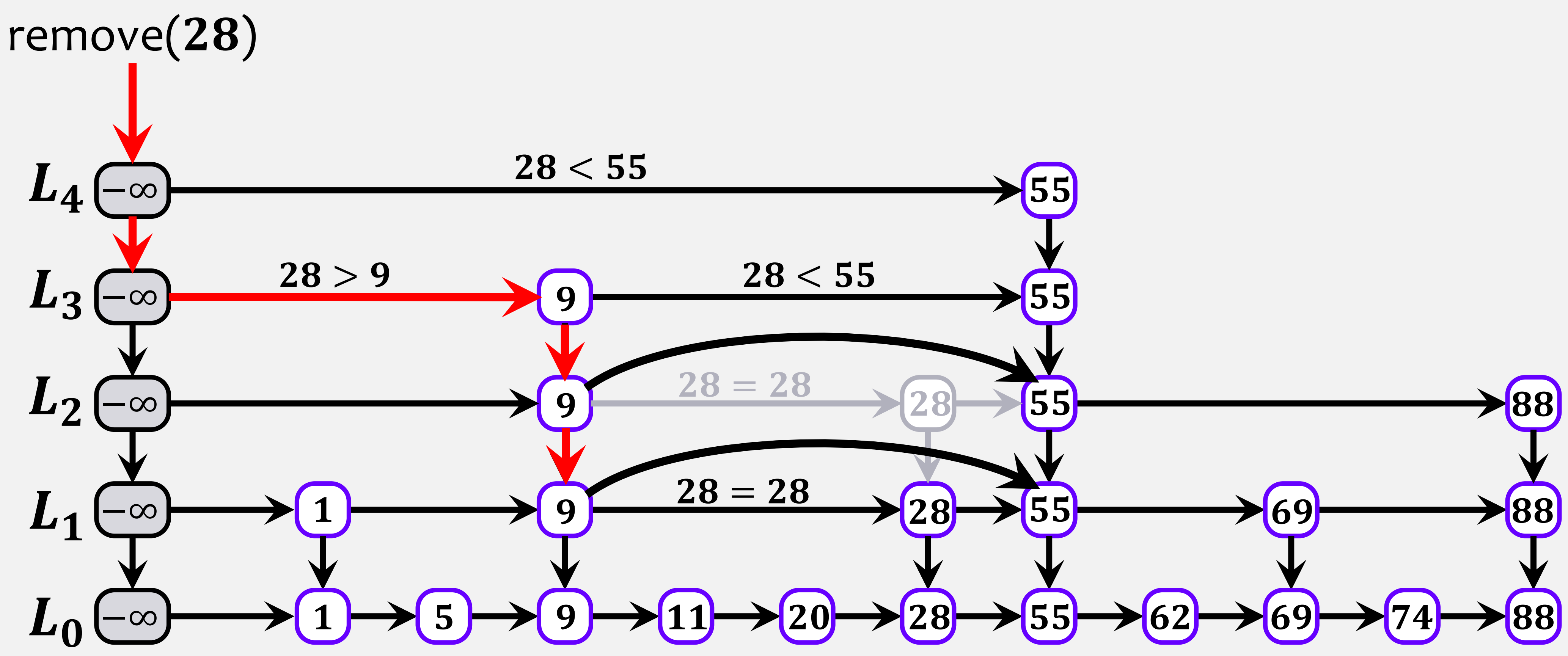
```
Node<T> findPredNode(T x) {
  Node<T> u = sentinel;
  int r = h;
  while (r >= 0) {
    while (u.next[r] != null && compare(u.next[r].x,x) < 0)
      u = u.next[r];   // go right in list r
    r--;               // go down into list r-1
  }
  return u;
}
T find(T x) {
  Node<T> u = findPredNode(x);
  return u.next[0] == null ? null : u.next[0].x;
}
```

from ODS book

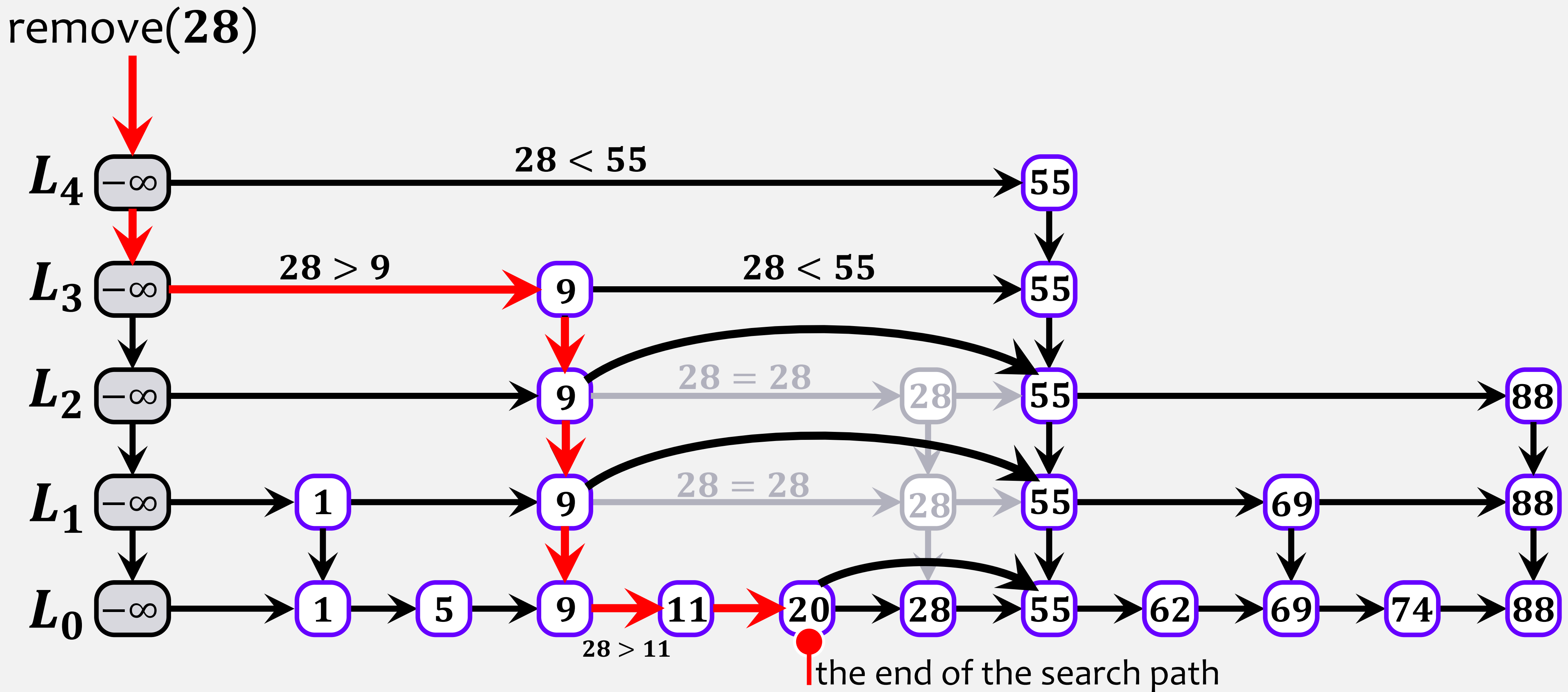
SSet – remove(x)



SSet – remove($\boldsymbol{x}$)



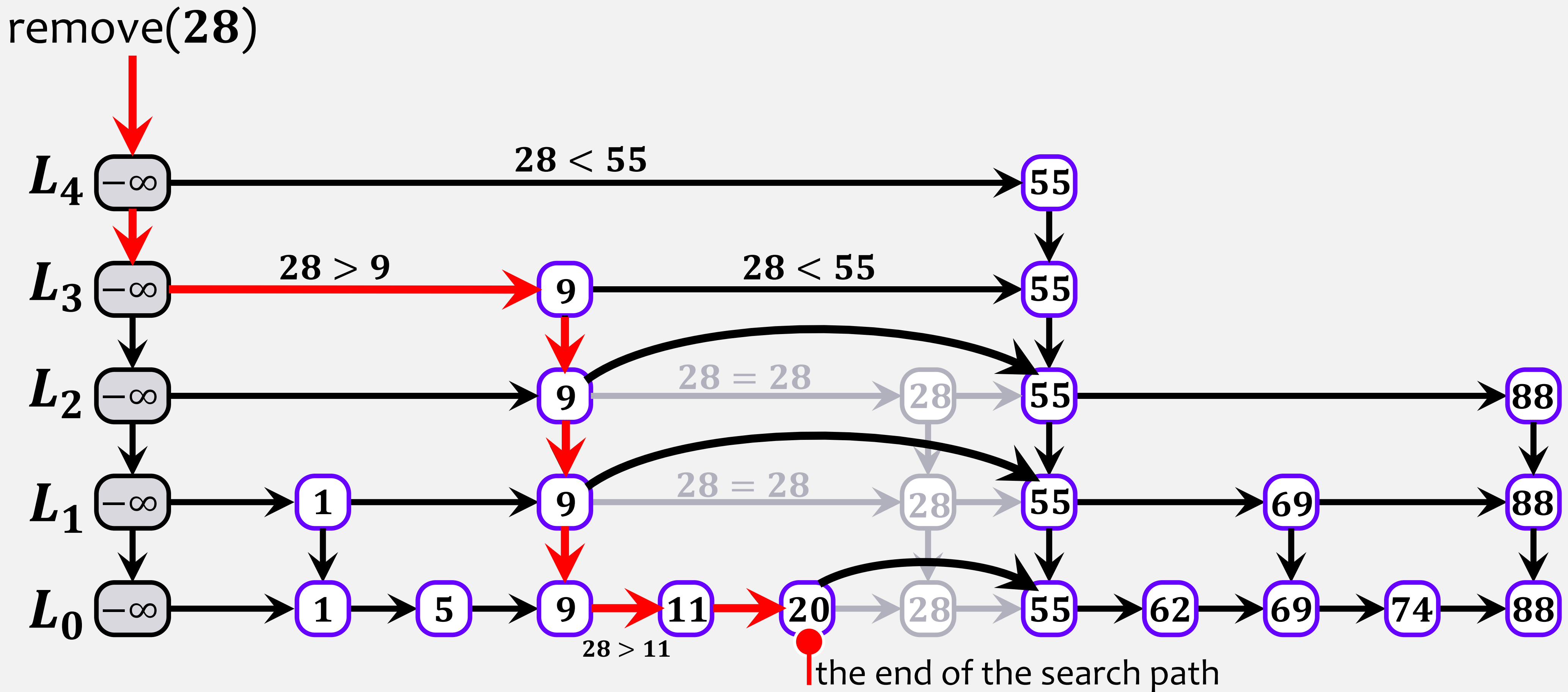
SSet – remove(x)



Alina Shaikhet – COMP2402 – Carleton University

SSet – remove($\boldsymbol{x}$)

The expected running time is $\boldsymbol{O}(\mathbf{log}\,\boldsymbol{n})$.



SSet – remove(x)

The expected running time is $O(\log n)$.

SkiplistSSet

```
boolean remove(T x) {
  boolean removed = false;
  Node<T> u = sentinel;
  int r = h;
  int comp = 0;
  while (r >= 0) {
    while (u.next[r] != null
           && (comp = compare(u.next[r].x, x)) < 0) {
      u = u.next[r];
    }
    if (u.next[r] != null && comp == 0) {
      removed = true;
      u.next[r] = u.next[r].next[r];
      if (u == sentinel && u.next[r] == null)
        h--;  // height has gone down
    }
    r--;
  }
  if (removed) n--;
  return removed;
}
```

going right

removal

going down

from ODS book

SSet – add($\boldsymbol{x}$)

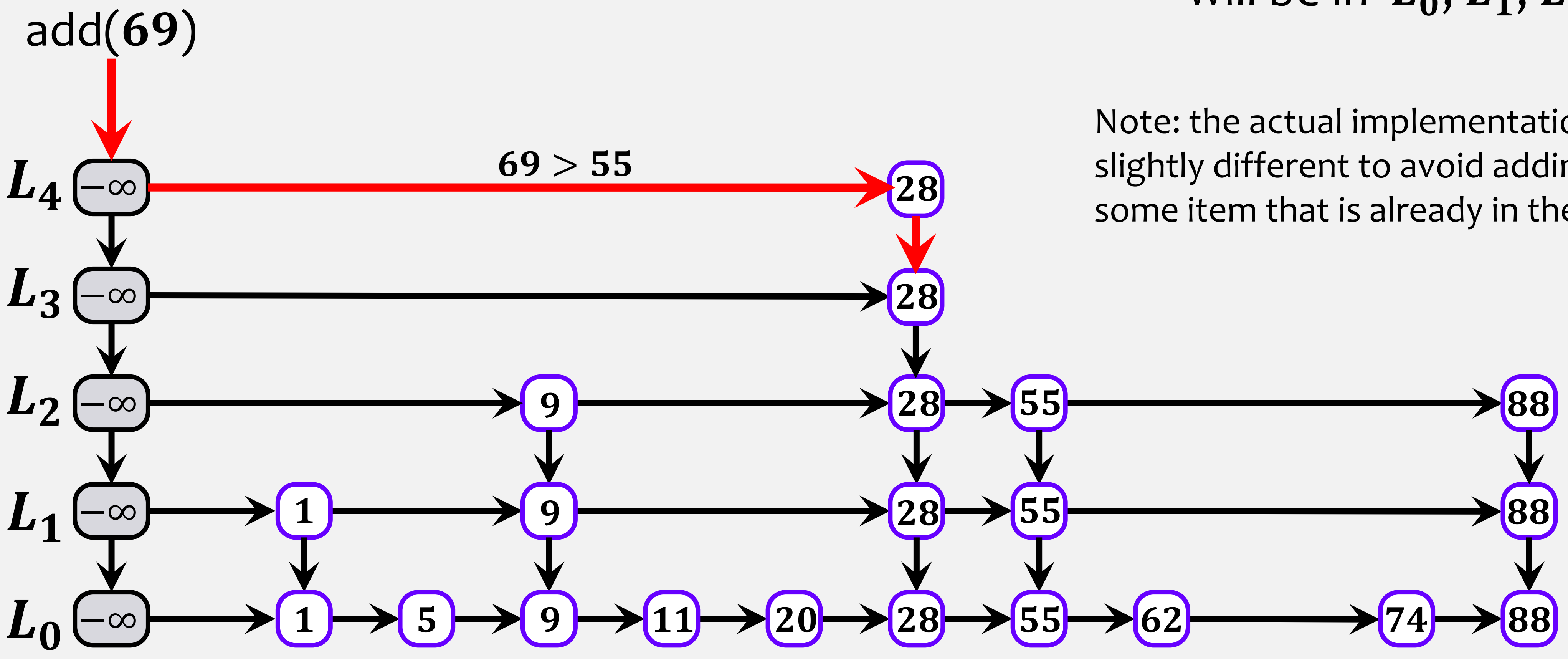
The expected running time is $\boldsymbol{O(\log n)}$.

- determine the height $\boldsymbol{k}$ of a new node by tossing a coin.
- follow the search path and modify the lists $\boldsymbol{L_k, L_{k-1}, \dots, L_0}$ by adding the new node to them.

For example: H, H, H, T

The node with value $\mathbf{69}$ will be in $\boldsymbol{L_0, L_1, L_2, L_3}$.

Note: the actual implementation is slightly different to avoid adding some item that is already in the set.



SSet – add($\boldsymbol{x}$)

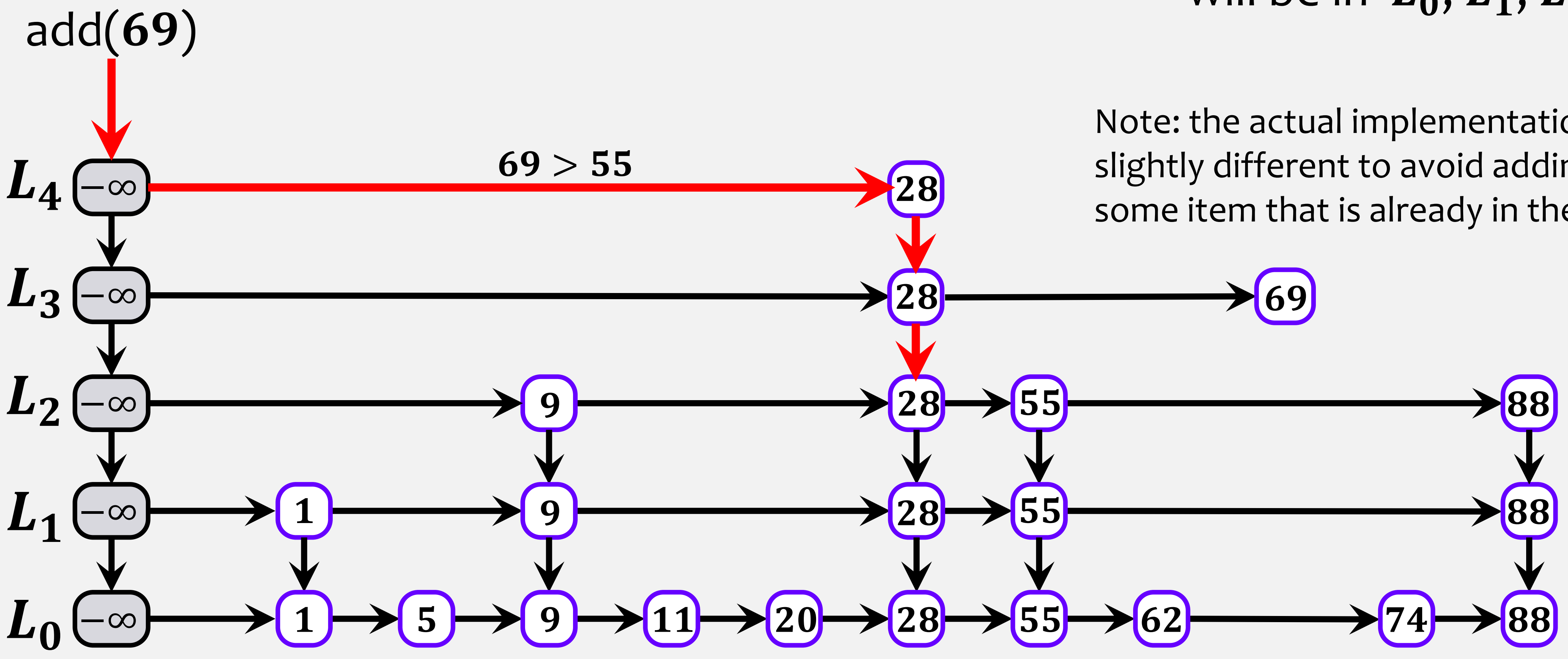
The expected running time is $\boldsymbol{O(\log n)}$.

- determine the height $\boldsymbol{k}$ of a new node by tossing a coin.
- follow the search path and modify the lists $\boldsymbol{L_k, L_{k-1}, \dots, L_0}$ by adding the new node to them.

For example: H, H, H, T

The node with value $\mathbf{69}$ will be in $\boldsymbol{L_0, L_1, L_2, L_3}$.

Note: the actual implementation is slightly different to avoid adding some item that is already in the set.



Alina Shaikhet – COMP2402 – Carleton University

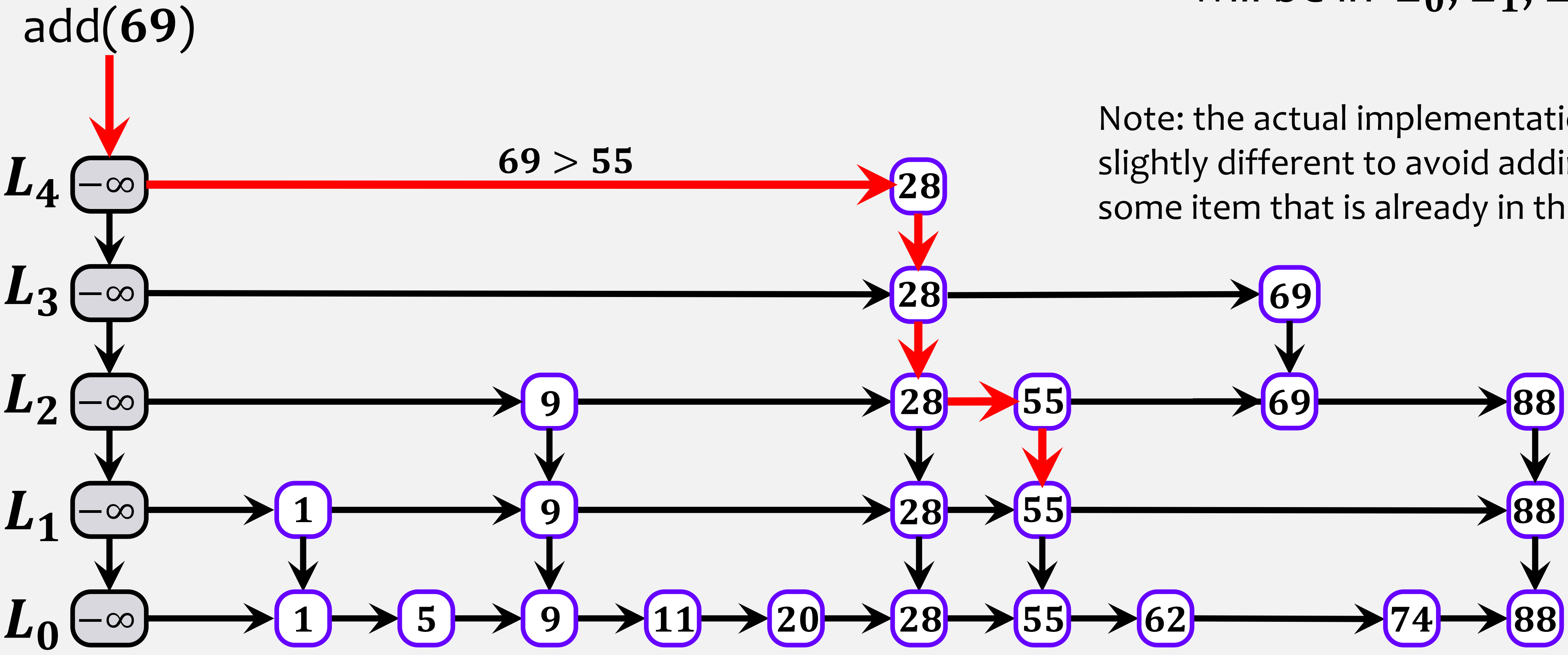
SSet – add(x)

The expected running time is $O(\log n)$.

- determine the height k of a new node by tossing a coin.
- follow the search path and modify the lists $L_k, L_{k-1}, \dots, L_0$ by adding the new node to them.

For example: H, H, H, T

The node with value $\mathbf{69}$ will be in L_0, L_1, L_2, L_3.



Note: the actual implementation is slightly different to avoid adding some item that is already in the set.

Alina Shaikhet – COMP2402 – Carleton University

SSet – add($\boldsymbol{x}$)

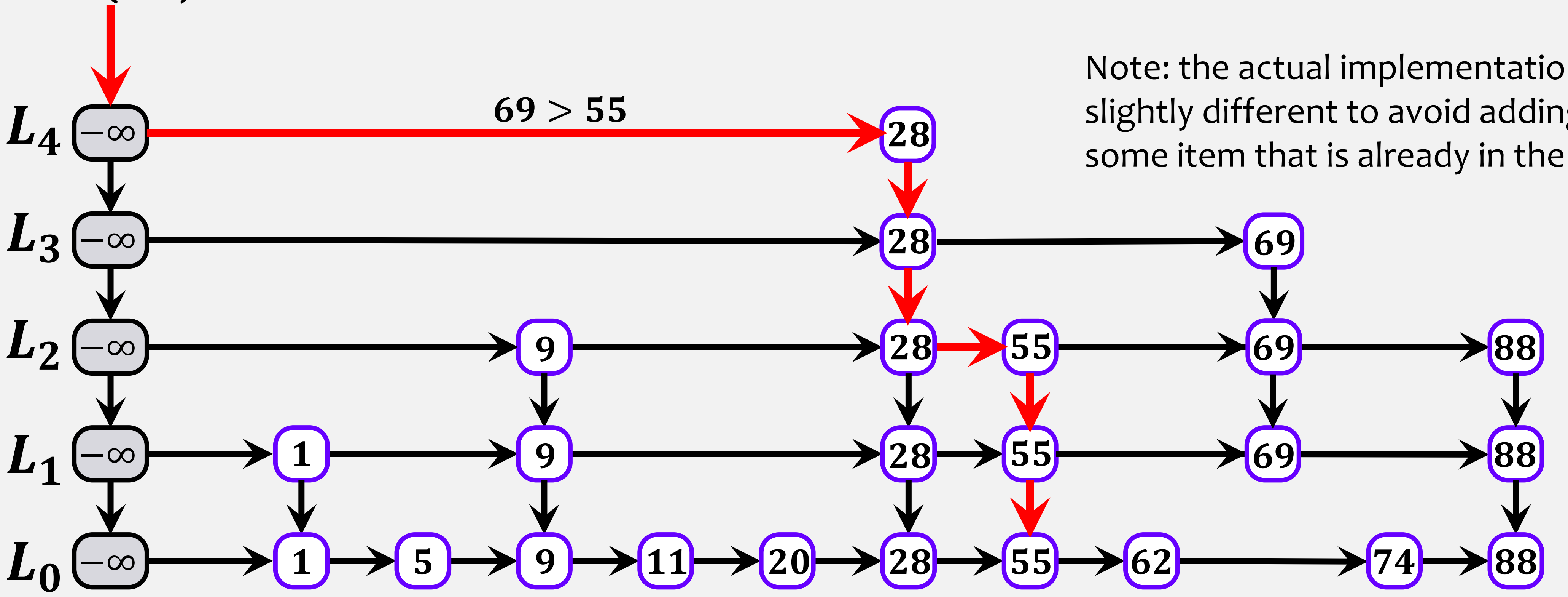
The expected running time is $\boldsymbol{O(\log n)}$.

- determine the height $\boldsymbol{k}$ of a new node by tossing a coin.
- follow the search path and modify the lists $\boldsymbol{L_k, L_{k-1}, \ldots , L_0}$ by adding the new node to them.

For example: H, H, H, T

The node with value **69** will be in $\boldsymbol{L_0, L_1, L_2, L_3}$.

add(**69**)



Note: the actual implementation is slightly different to avoid adding some item that is already in the set.

SSet – add(x)

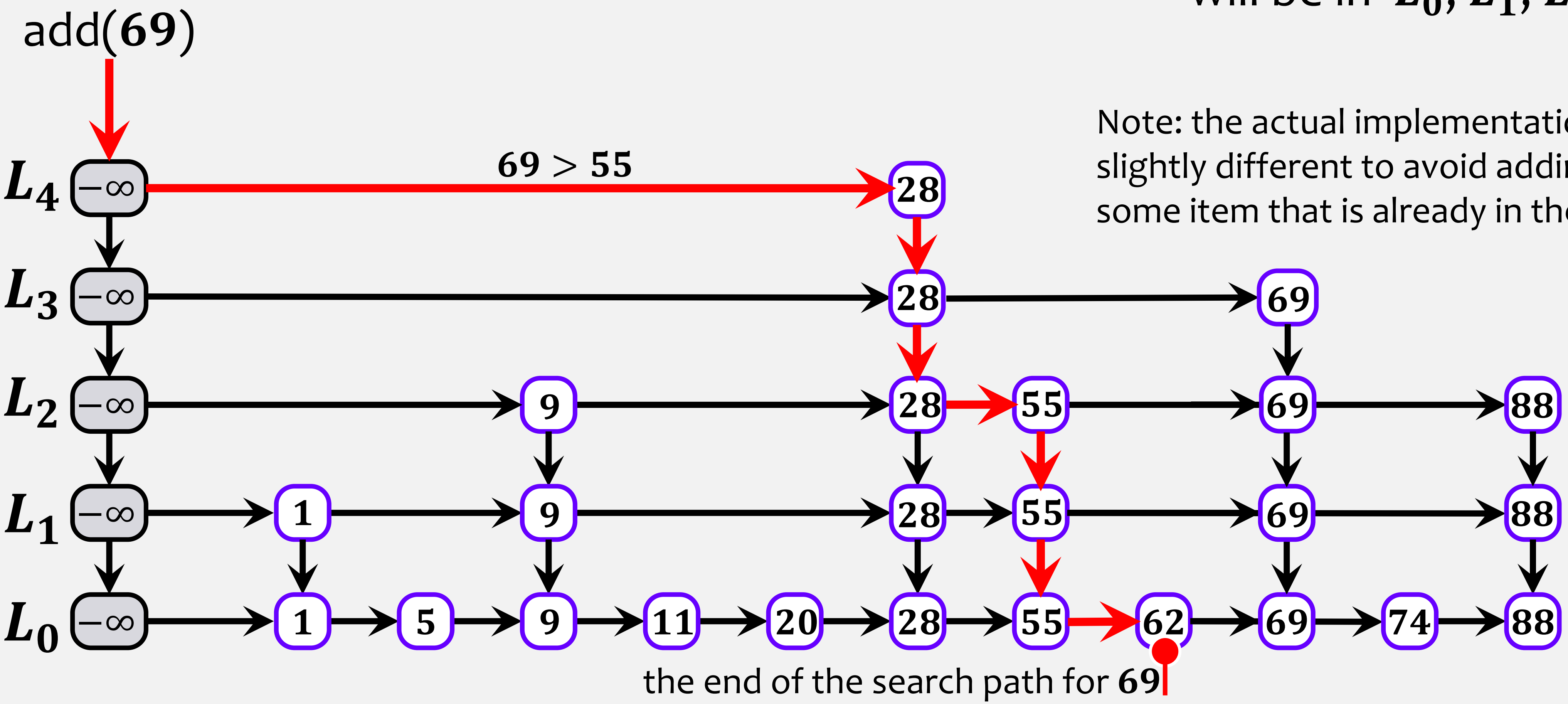
The expected running time is $O(\log n)$.

- determine the height k of a new node by tossing a coin.
- follow the search path and modify the lists $L_k, L_{k-1}, \dots, L_0$ by adding the new node to them.

For example: H, H, H, T

The node with value **69** will be in L_0, L_1, L_2, L_3.

Note: the actual implementation is slightly different to avoid adding some item that is already in the set.



List

- get($\boldsymbol{i}$),
- set($\boldsymbol{i}, \boldsymbol{x}$),
- add($\boldsymbol{i}, \boldsymbol{x}$), and
- remove($\boldsymbol{i}$)

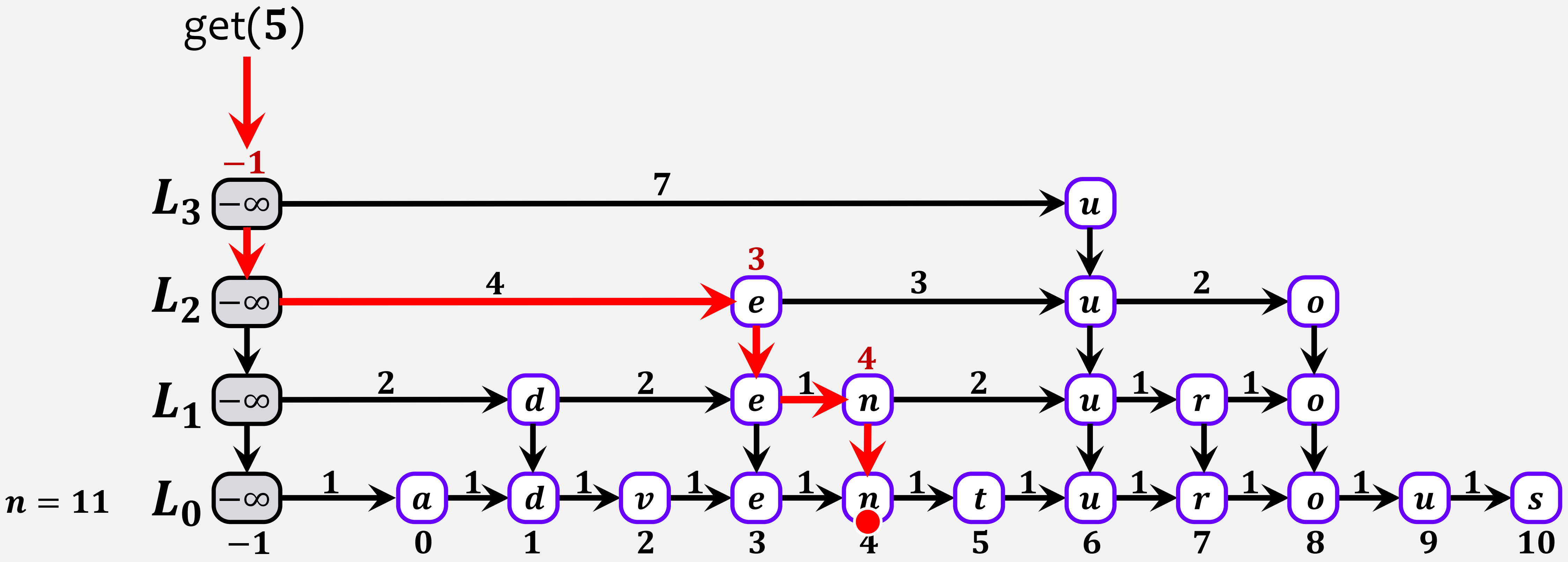
A **SkiplistList** implements the **List** interface:

- $\boldsymbol{L_0}$ contains the elements of the list in the order in which they appear in the list.
- elements can be added, removed, and accessed in $\boldsymbol{O(\log n)}$ **expected** time per operation.

List – get($\boldsymbol{i}$)/set($\boldsymbol{i}, \boldsymbol{x}$)

Every edge has a “length” associated with it (we store this information).

We do not store indices of the nodes. We only keep track of the node we are at.



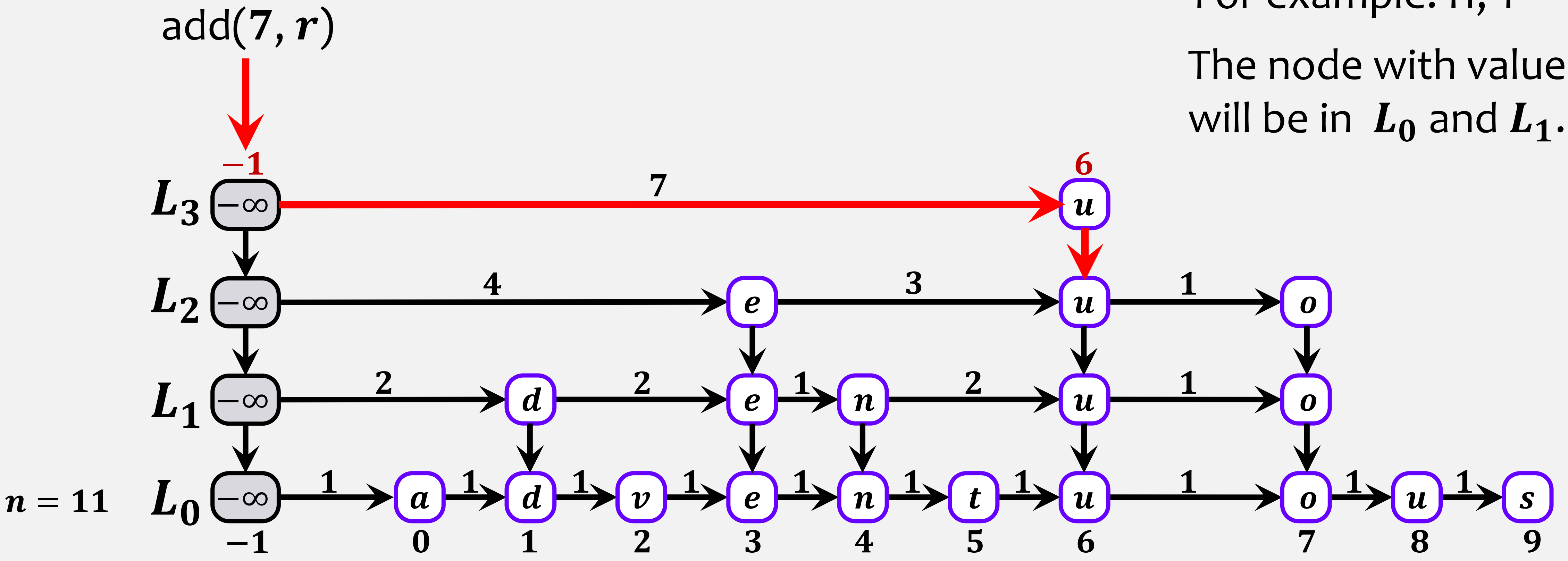
Alina Shaikhet – COMP2402 – Carleton University

List – add(i, x)

- create a new node and determine its height k in the skiplist by tossing a coin.
- follow the search path and modify the lists $L_k, L_{k-1}, \ldots, L_0$ by adding the new node to them.

For example: H, T

The node with value r will be in L_0 and L_1.

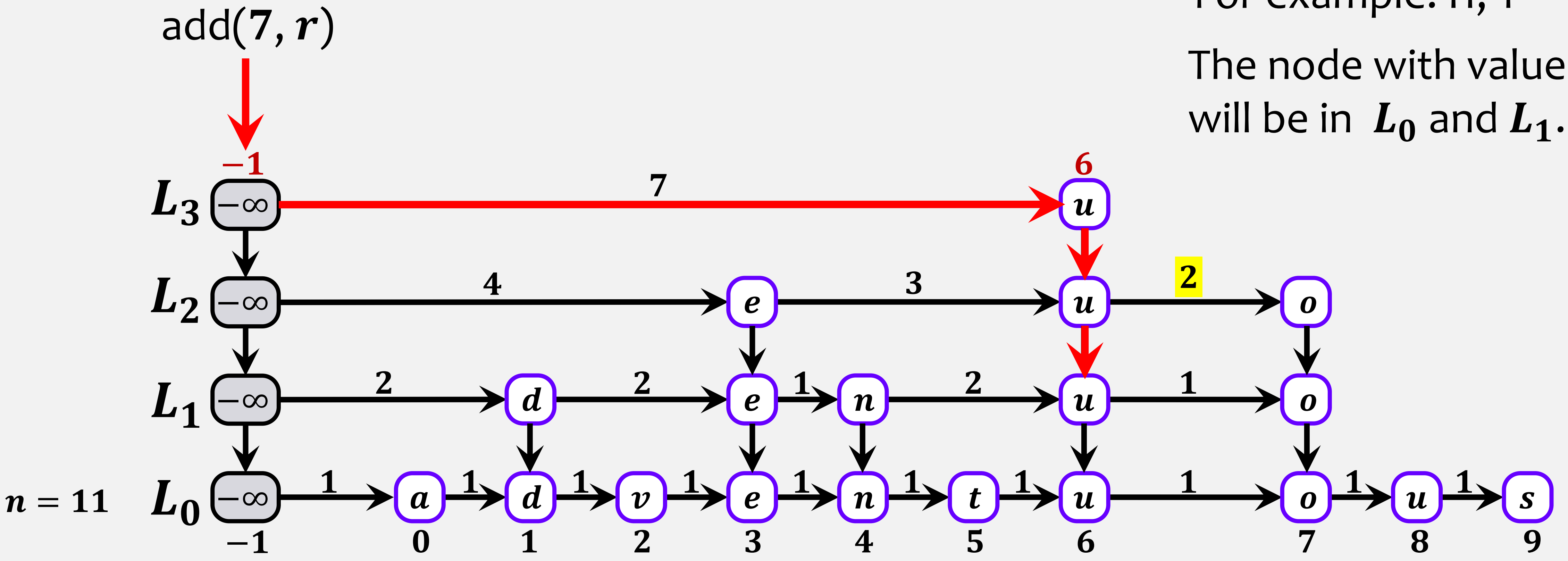


List – add($\boldsymbol{i}, \boldsymbol{x}$)

- create a new node and determine its height $\boldsymbol{k}$ in the skiplist by tossing a coin.
- follow the search path and modify the lists $\boldsymbol{L_k}, \boldsymbol{L_{k-1}}, \dots, \boldsymbol{L_0}$ by adding the new node to them.

For example: H, T

The node with value $\boldsymbol{r}$ will be in $\boldsymbol{L_0}$ and $\boldsymbol{L_1}$.



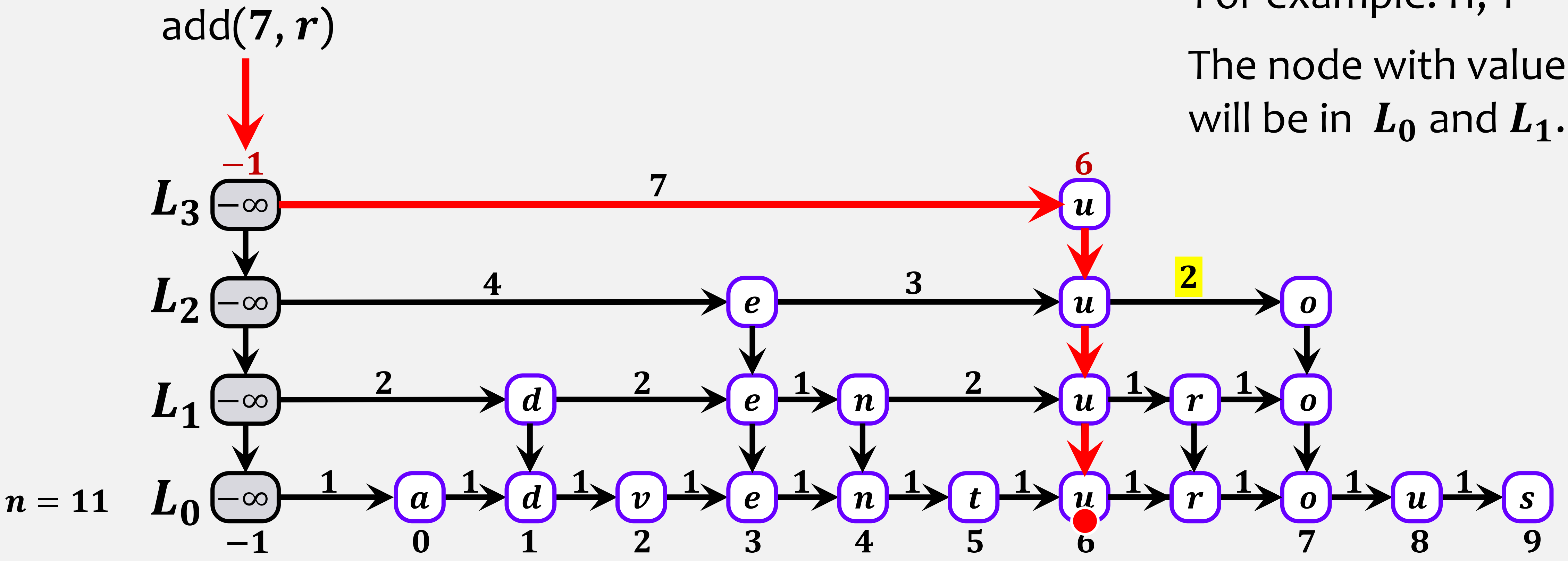
Alina Shaikhet – COMP2402 – Carleton University

List – add($\boldsymbol{i}, \boldsymbol{x}$)

- create a new node and determine its height $\boldsymbol{k}$ in the skiplist by tossing a coin.
- follow the search path and modify the lists $\boldsymbol{L_k}, \boldsymbol{L_{k-1}}, \dots, \boldsymbol{L_0}$ by adding the new node to them.

For example: H, T

The node with value $\boldsymbol{r}$ will be in $\boldsymbol{L_0}$ and $\boldsymbol{L_1}$.



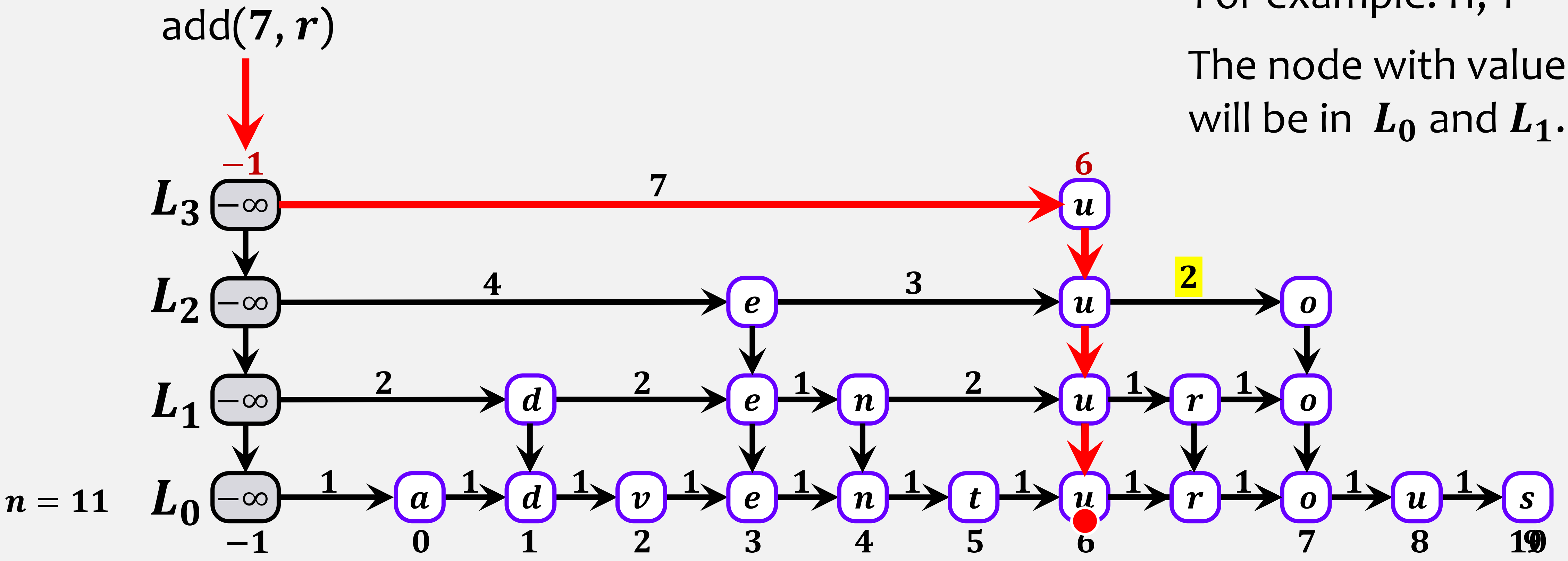
Alina Shaikhet – COMP2402 – Carleton University

List – add($\boldsymbol{i}, \boldsymbol{x}$)

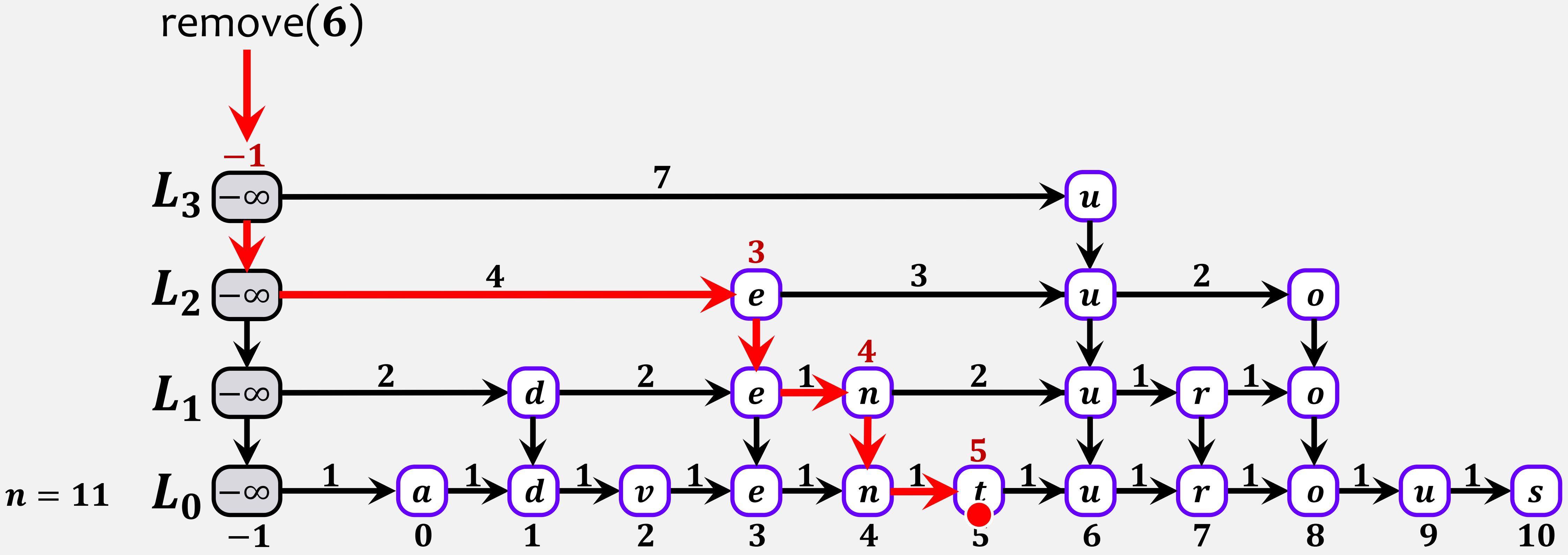
- create a new node and determine its height $\boldsymbol{k}$ in the skiplist by tossing a coin.
- follow the search path and modify the lists $\boldsymbol{L_k}, \boldsymbol{L_{k-1}}, \dots, \boldsymbol{L_0}$ by adding the new node to them.

For example: H, T

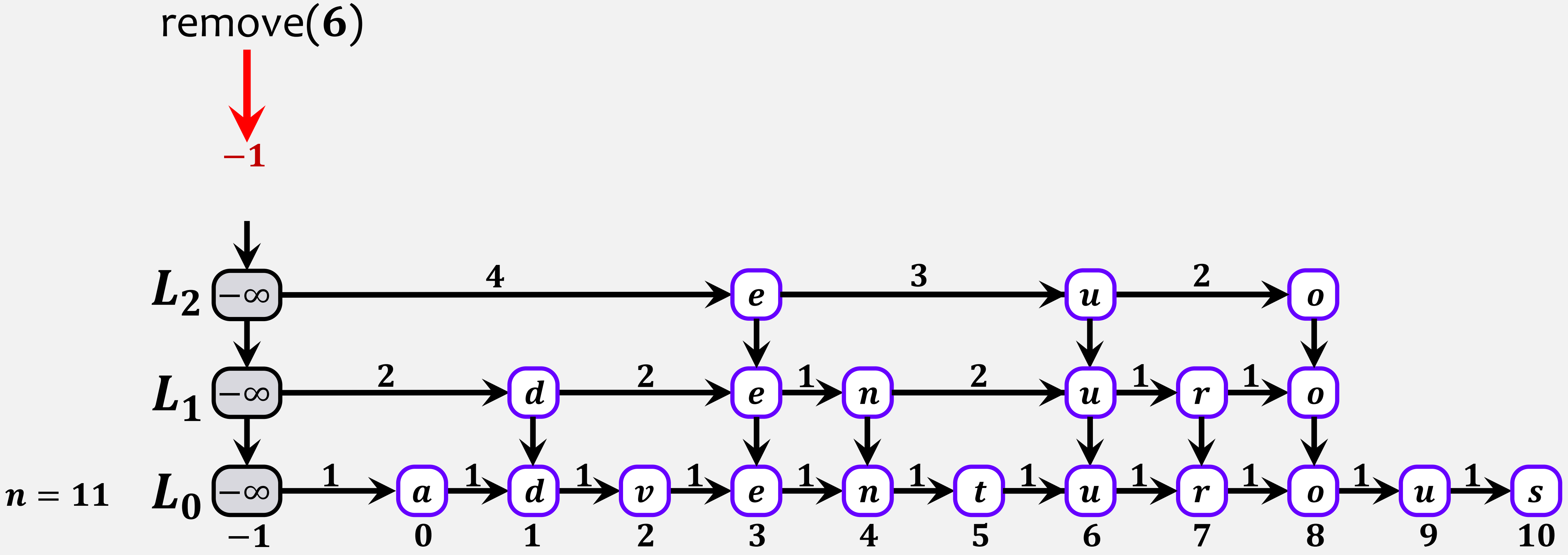
The node with value $\boldsymbol{r}$ will be in $\boldsymbol{L_0}$ and $\boldsymbol{L_1}$.



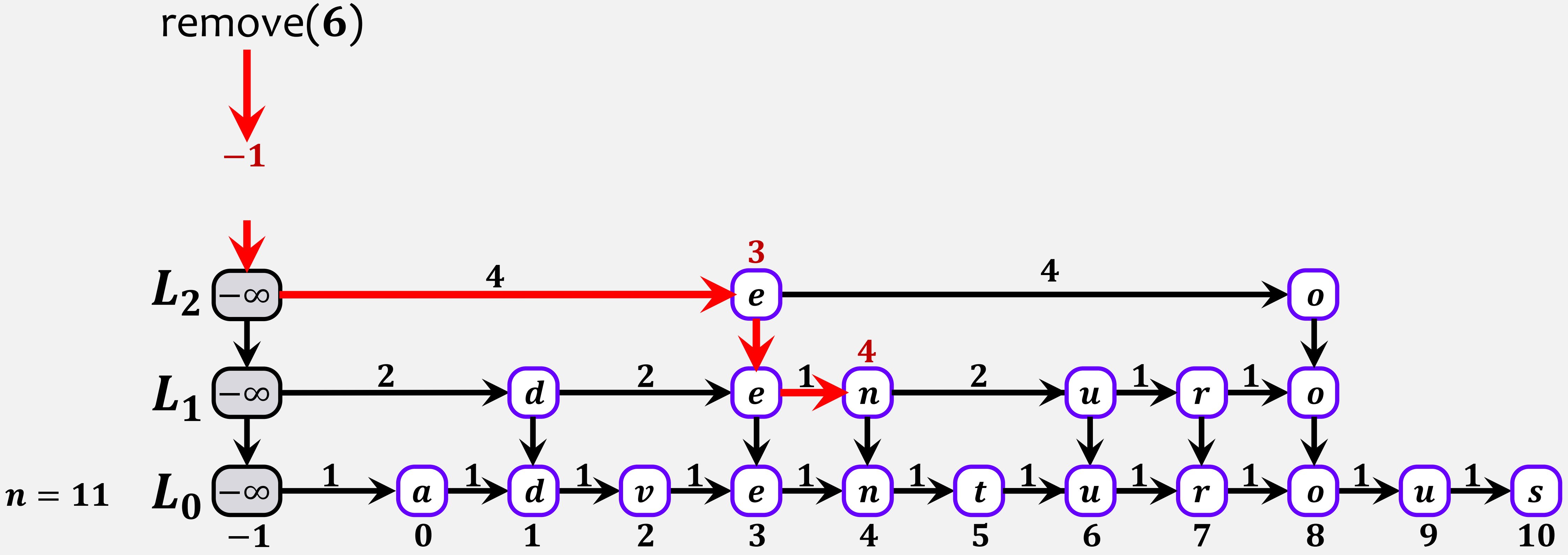
List – remove($\boldsymbol{i}$)



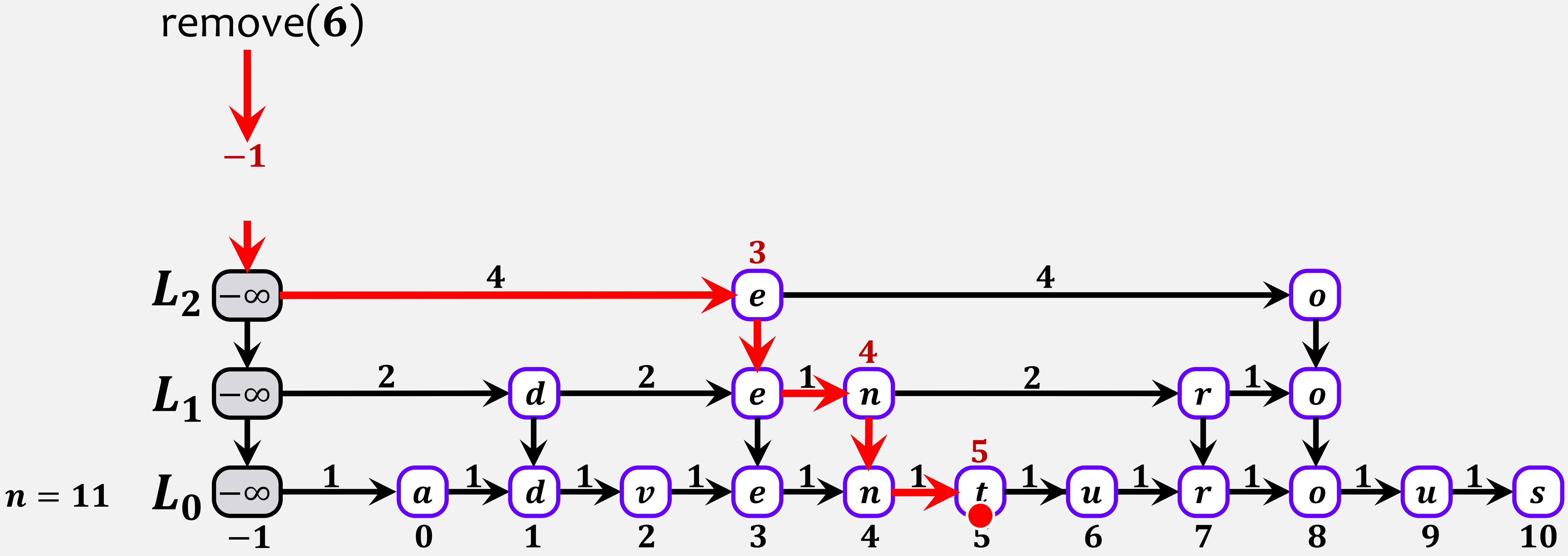
List – remove(i)



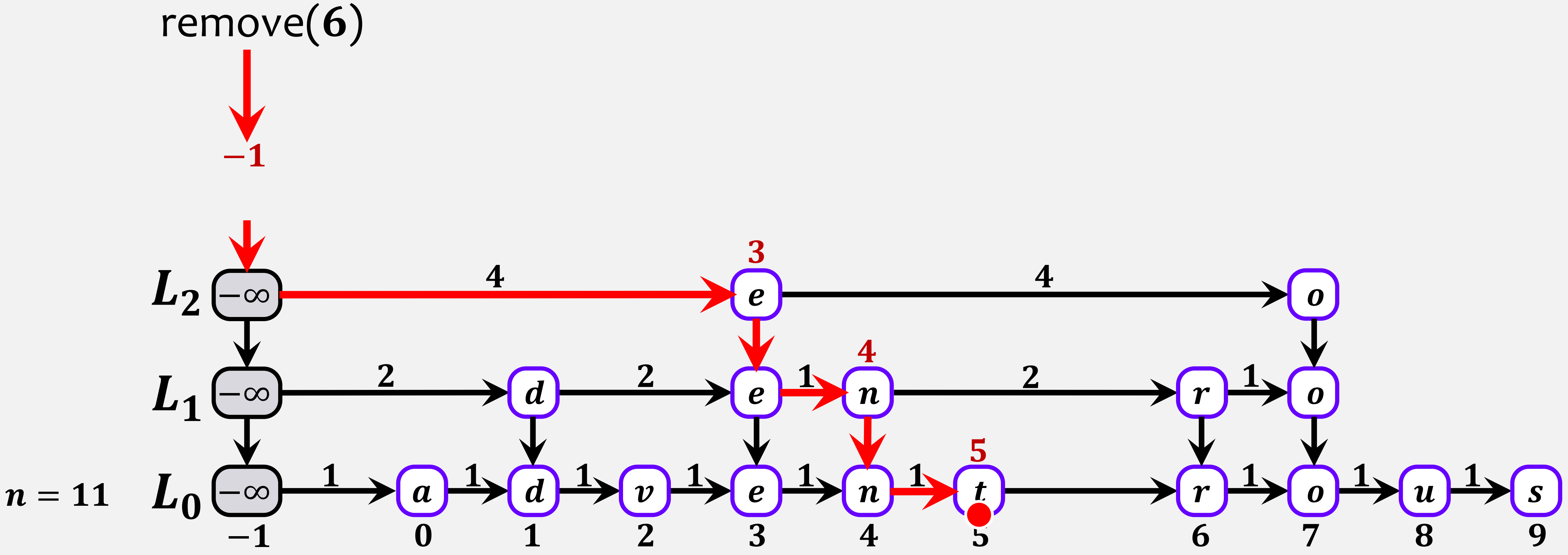
List – remove(i)



List – remove($\boldsymbol{i}$)



List – remove($\boldsymbol{i}$)



Basic Probability – Random Variables

Examples of random variables:

- fair coin $\boldsymbol{Pr(H) = Pr(T) = \frac{1}{2}}$
- random bit $\boldsymbol{Pr(0) = Pr(1) = \frac{1}{2}}$
- 6-sided die $\boldsymbol{Pr(1) = Pr(2) = \cdots = Pr(6) = \frac{1}{6}}$

The running time of an operation on a randomized data structure is a random variable, and we want to study its expected value.

Basic Probability – Expectation

For a discrete random variable $\boldsymbol{X}$ taking on values in some countable universe $\boldsymbol{U}$, the expected value of $\boldsymbol{X}$, denoted by $\boldsymbol{E}[\boldsymbol{X}]$, is given by the formula

$$\boldsymbol{E}[\boldsymbol{X}] = \sum_{\boldsymbol{x} \in \boldsymbol{U}} \boldsymbol{x} \cdot \boldsymbol{Pr}(\boldsymbol{X} = \boldsymbol{x})$$

The expected value of $\boldsymbol{X}$ is the value of $\boldsymbol{X}$ that we observe on **average**.

$$\boldsymbol{E}[\text{random bit}] = \mathbf{0} \cdot \boldsymbol{Pr}(\text{bit} = \mathbf{0}) + \mathbf{1} \cdot \boldsymbol{Pr}(\text{bit} = \mathbf{1}) = \mathbf{0} \cdot \frac{\mathbf{1}}{\mathbf{2}} + \mathbf{1} \cdot \frac{\mathbf{1}}{\mathbf{2}} = \frac{\mathbf{1}}{\mathbf{2}}$$

$$\boldsymbol{E}[\text{die}] = \mathbf{1} \cdot \boldsymbol{Pr}(\text{die} = \mathbf{1}) + \mathbf{2} \cdot \boldsymbol{Pr}(\text{die} = \mathbf{2}) + \cdots + \mathbf{6} \cdot \boldsymbol{Pr}(\text{die} = \mathbf{6}) =$$

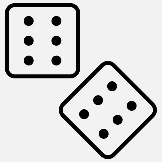
$$= \mathbf{1} \cdot \frac{\mathbf{1}}{\mathbf{6}} + \mathbf{2} \cdot \frac{\mathbf{1}}{\mathbf{6}} + \cdots + \mathbf{6} \cdot \frac{\mathbf{1}}{\mathbf{6}} = \frac{\mathbf{1}}{\mathbf{6}} + \frac{\mathbf{2}}{\mathbf{6}} + \frac{\mathbf{3}}{\mathbf{6}} + \frac{\mathbf{4}}{\mathbf{6}} + \frac{\mathbf{5}}{\mathbf{6}} + \frac{\mathbf{6}}{\mathbf{6}} = \frac{\mathbf{21}}{\mathbf{6}} = \mathbf{3.5}$$

Basic Probability – Linearity of Expectation

The Linearity of Expectation is one of the most useful properties of expected values.

It tells us how to obtain the expected value of a random variable $\boldsymbol{Z = X + Y}$ from the expected values of $\boldsymbol{X}$ and $\boldsymbol{Y}$.

For any two random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$: $\boldsymbol{E[X+Y] = E[X] + E[Y]}$



For example, we roll two fair and independent dice, one being red and the other being blue.

More generally, for any random variables $\boldsymbol{X_1, X_2, \dots, X_m}$:

$$\boldsymbol{E\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} E[X_i]}$$

Coin Tosses

Define a random variable $\boldsymbol{X}$: $\boldsymbol{X} =$ "number of times we toss a coin up to and including the first time the coin comes up heads".

(we stop tossing the coin the first time it comes up heads)

$$
\begin{array}{rll}
\boldsymbol{H} & \boldsymbol{X = 1} & \boldsymbol{X_1 = 1, X_2 = 0, X_3 = 0, X_4 = 0, \dots} \\
\boldsymbol{T, H} & \boldsymbol{X = 2} & \boldsymbol{X_1 = 1, X_2 = 1, X_3 = 0, X_4 = 0, \dots} \\
\boldsymbol{T, T, H} & \boldsymbol{X = 3} & \boldsymbol{X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 0, \dots} \\
\boldsymbol{T, T, T, H} & \boldsymbol{X = 4} & \boldsymbol{X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1, X_5 = 0, \dots}
\end{array}
$$

For each integer $\boldsymbol{i \geq 1}$ we define the **indicator random variable**:

$$\boldsymbol{X_i} = \begin{cases} \boldsymbol{1}, & \text{if we have to make the } \boldsymbol{i}\text{-th coin toss} \\ \boldsymbol{0}, & \text{otherwise} \end{cases}$$

Observe, that $\boldsymbol{X} = \sum_{\boldsymbol{i=1}}^{\infty} \boldsymbol{X_i}$

Coin Tosses

Linearity of expectation:

$$E\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} E[X_i]$$

$$X_i = \begin{cases} 1, & \text{if we have to make the } i\text{-th coin toss} \\ 0, & \text{otherwise} \end{cases}$$

Note, that $X_i = 1$ if and only if the first $i-1$ coin tosses are tails

$$X = \sum_{i=1}^{\infty} X_i$$

by linearity of expectation

$$E[X] = E\left[\sum_{i=1}^{\infty} X_i\right] = \sum_{i=1}^{\infty} E[X_i] = \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots = 2$$

$$\begin{aligned} E[X_i] &= 0 \cdot Pr(X_i = 0) + 1 \cdot Pr(X_i = 1) \\ &= Pr(X_i = 1) \\ &= Pr(\text{previous } i-1 \text{ coin tosses all came up tails}) \\ &= \frac{1}{2^{i-1}} \end{aligned}$$

Lemma 4.2:
$E[X] = 2$

Definition of expectation:

$$E[X] = \sum_{x \in U} x \cdot Pr(X = x)$$

Lemma 4.3

> The expected number of nodes in a SkipList containing n elements, not including occurrences of the sentinel, is $2n$.

Proof:

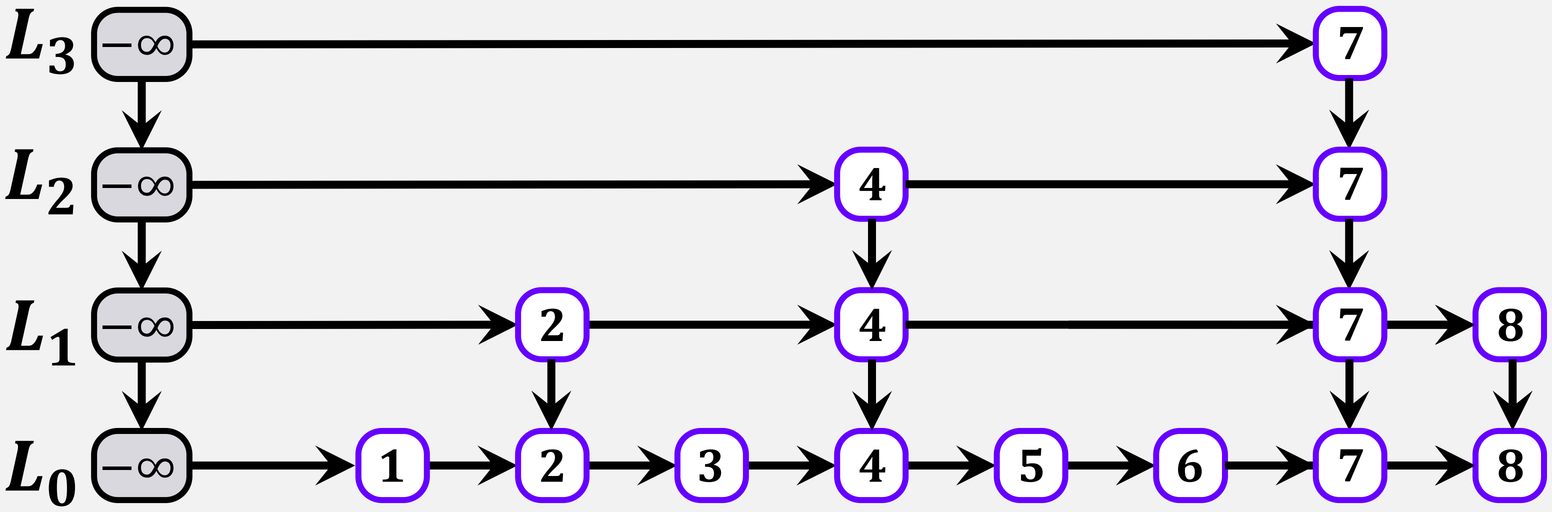
For each $i \geq 1$, we define a random variable $|L_i|$ = the size of L_i (not including the sentinel)

$$E[|\text{SkipList}|] = E\left[\sum_{i=0}^{\infty} |L_i|\right] = \sum_{i=0}^{\infty} E[|L_i|] = \sum_{i=0}^{\infty} \frac{n}{2^i} = n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

$$X_{i,j} = \begin{cases} 1, & \text{if node } j \in \{1, \dots, n\} \text{ is in } L_i \\ 0, & \text{otherwise} \end{cases} \qquad |L_i| = \sum_{j=1}^{n} X_{i,j} \qquad E[|L_i|] = E\left[\sum_{j=1}^{n} X_{i,j}\right] = \sum_{j=1}^{n} E[X_{i,j}]$$

$$= \sum_{j=1}^{n} Pr(X_{i,j} = 1)$$

$$= \sum_{j=1}^{n} \frac{1}{2^i} = \frac{n}{2^i}$$



Lemma 4.4

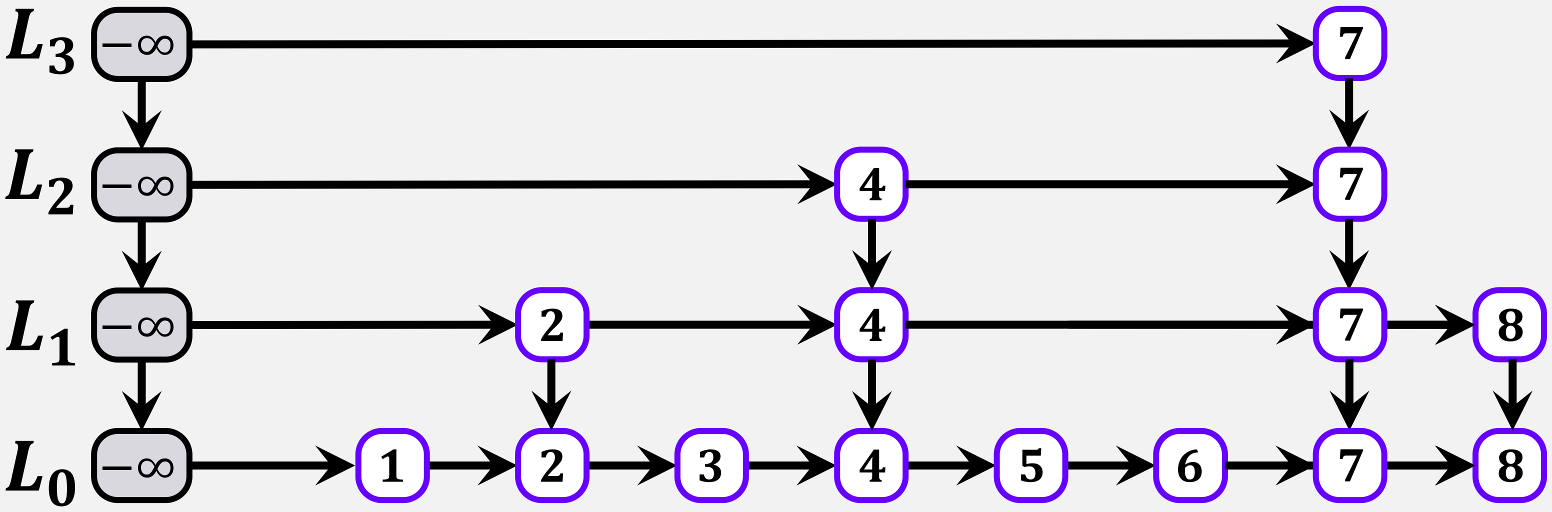
The expected height of a SkipList containing $\boldsymbol{n}$ elements is at most $\log n + 2$.

the largest value $\boldsymbol{i}$ such that $\boldsymbol{L_i}$ is not empty

Proof:

For each $i \geq 1$, we define an indicator random variable $\quad I_i = \begin{cases} 0, & \text{if } L_i \text{ is empty} \\ 1, & \text{otherwise} \end{cases}$

$$\text{height} = \sum_{i=1}^{\infty} I_i \qquad E[\text{height}] = E\left[\sum_{i=1}^{\infty} I_i\right] = \sum_{i=1}^{\infty} E[I_i] = \sum_{i=1}^{\infty} Pr(I_i = 1)$$



Lemma 4.4

The expected height of a SkipList containing n elements is at most $\log n + 2$.

the largest value i such that L_i is not empty

Proof:

For each $i \geq 1$, we define an indicator random variable $I_i = \begin{cases} 0, & \text{if } L_i \text{ is empty} \\ 1, & \text{otherwise} \end{cases}$

$$\text{height} = \sum_{i=1}^{\infty} I_i$$

$$I_i \leq 1$$

$$E[I_i] \leq E[1] = 1$$

$$I_i \leq |L_i|$$

$$E[I_i] \leq E[|L_i|] = \frac{n}{2^i}$$

$$E[\text{height}] = E\left[\sum_{i=1}^{\infty} I_i\right] = \sum_{i=1}^{\infty} E[I_i] = \sum_{i=1}^{\infty} Pr(I_i = 1)$$

$$= \sum_{i=1}^{\infty} E[I_i] = \sum_{i=1}^{\lfloor \log n \rfloor} E[I_i] + \sum_{i=\lfloor \log n \rfloor + 1}^{\infty} E[I_i]$$

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

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

$2^i = n$ when $i = \log n$
$2^i > n$ when $i > \log n$

$$\sum_{i=\lfloor \log n \rfloor + 1}^{\infty} \frac{n}{2^i} \leq \frac{n}{n} + \frac{n}{2n} + \frac{n}{4n} + \cdots$$

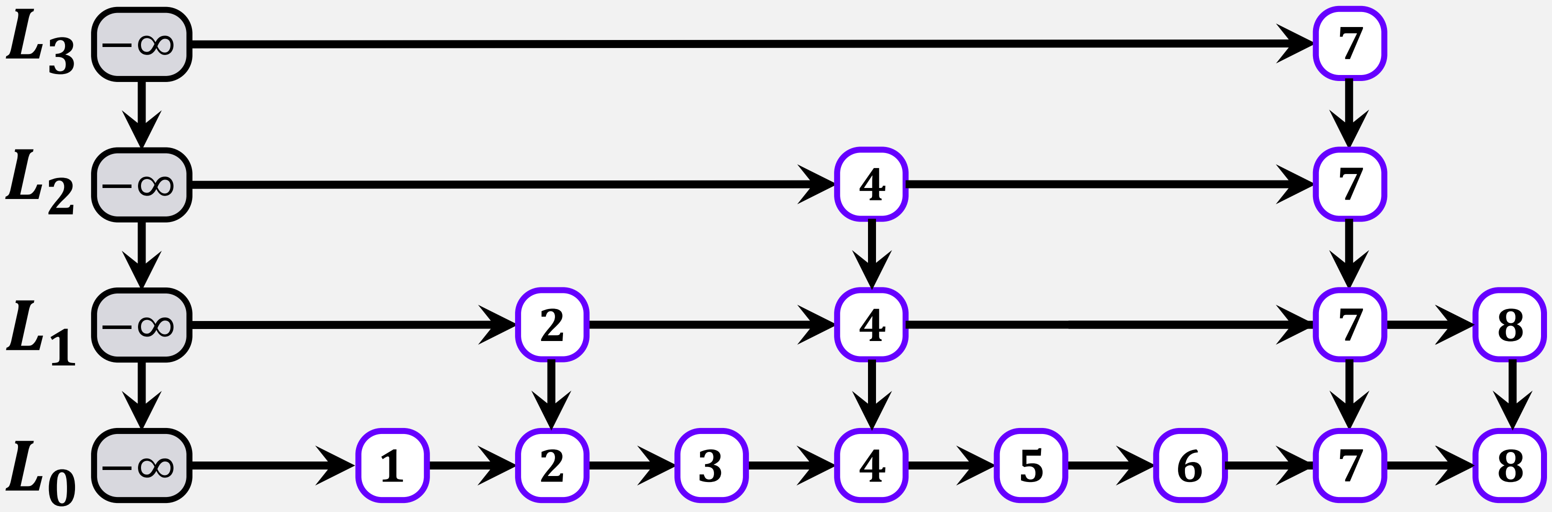
$$\frac{n}{2^{\log n + 1}} = \frac{n}{2^{\log n} \cdot 2}$$

Lemma 4.5

> The expected number of nodes in a SkipList containing n elements, including all occurrences of the sentinel, is $2n + O(\log n)$.

Proof:

- By Lemma 4.3, the expected number of nodes, not including the sentinel, is $2n$.
- The number of occurrences of the sentinel is equal to the height of the SkipList.
- By Lemma 4.4 the expected number of occurrences of the sentinel is at most $\log n + 2 = O(\log n)$.



Lemma 4.6

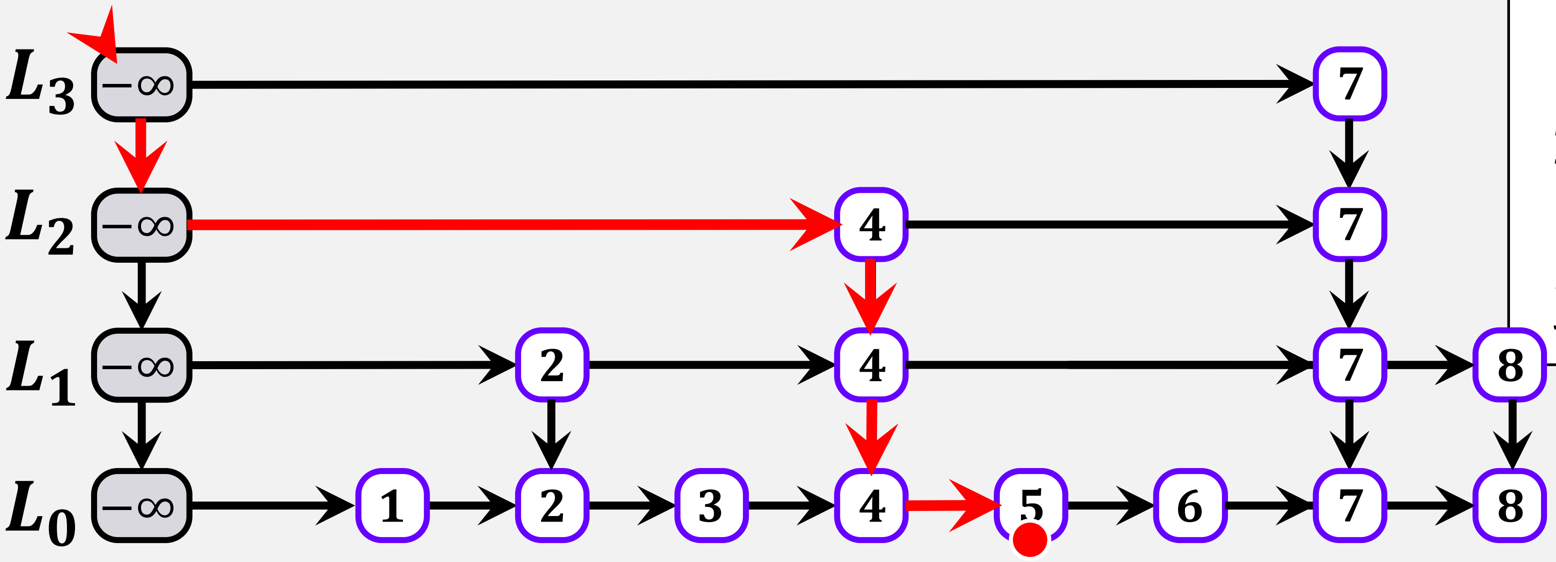
The expected length of a search path in a SkipList containing $\boldsymbol{n}$ elements is at most $\mathbf{2}\log \boldsymbol{n} + \boldsymbol{O}(\mathbf{1})$.

Proof:

$$\text{length of a search path} = \#\text{ steps down} + \#\text{ steps right}$$

$$\boldsymbol{E}[\text{length of a search path}] = \boldsymbol{E}[\#\text{ steps down}] + \boldsymbol{E}[\#\text{ steps right}]$$

$$= \underbrace{\boldsymbol{E}[\text{height}]}_{\leq\ \log \boldsymbol{n} + \mathbf{2}} + \boldsymbol{E}[\#\text{ steps right}]$$

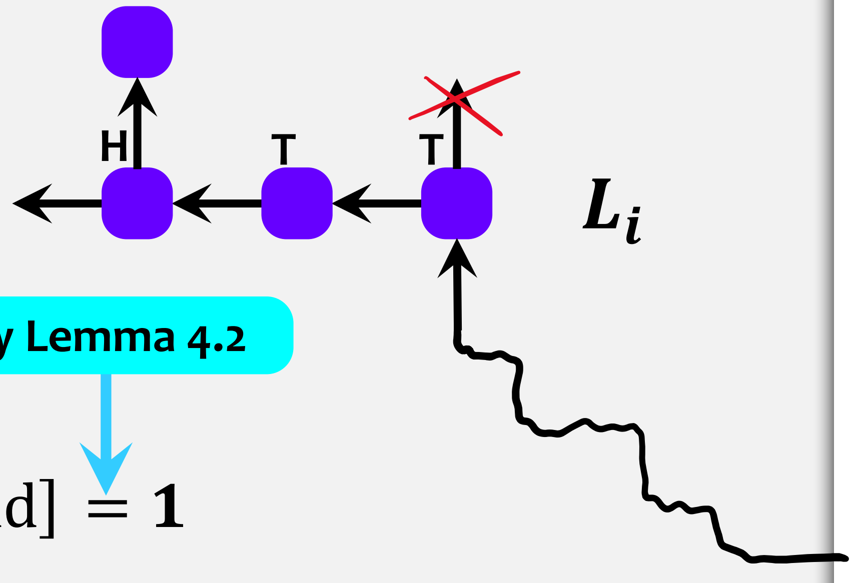


reverse search path:

1. starts at the predecessor of $\boldsymbol{x}$ in $\boldsymbol{L_0}$
2. if the path can go up a level, then go **up.**
3. otherwise, go to the **left.**

Lemma 4.6

The number of coin tosses before the heads represents the number of steps to the left that a reverse search path takes at a particular level i.



by Lemma 4.2

$$E[\#\text{ steps right in } L_i] \le E[\#\text{ coin tosses before the first head}] = 1$$

$$E[\#\text{ steps right in } L_i] \le E[|L_i|] = \frac{n}{2^i}$$

proof of Lemma 4.3

$$\begin{aligned}
E[\text{length of a search path}] &= E[\text{height}] + E[\#\text{ steps right}] \\
&= E[\text{height}] + E\left[\sum_{i=0}^{\infty} \#\text{ steps right in } L_i\right] \\
&= E[\text{height}] + \sum_{i=0}^{\lfloor \log n \rfloor} E[\#\text{ steps right in } L_i] + \sum_{i=\lfloor \log n \rfloor + 1}^{\infty} E[\#\text{ steps right in } L_i] \\
&\le E[\text{height}] + \sum_{i=0}^{\lfloor \log n \rfloor} 1 + \sum_{i=\lfloor \log n \rfloor + 1}^{\infty} \frac{n}{2^i}
\end{aligned}$$

by linearity of expectation

Lemma 4.6

$$\boldsymbol{E}[\text{length of a search path}] \le \underbrace{\boldsymbol{E}[\text{height}]}_{\le \log \boldsymbol{n} + \boldsymbol{2}} + \overbrace{\sum_{\boldsymbol{i}=\boldsymbol{0}}^{\lfloor \log \boldsymbol{n} \rfloor} \boldsymbol{1}}^{\le \log \boldsymbol{n} + \boldsymbol{1}} + \underbrace{\sum_{\boldsymbol{i}=\lfloor \log \boldsymbol{n} \rfloor + \boldsymbol{1}}^{\infty} \frac{\boldsymbol{n}}{\boldsymbol{2}^{\boldsymbol{i}}}}_{\le \boldsymbol{2}} \le \boldsymbol{2} \log \boldsymbol{n} + \boldsymbol{5}$$

Theorem 4.3

A SkipList containing $\boldsymbol{n}$ elements has expected size $\boldsymbol{O(n)}$ and the expected length of the search path for any particular element is at most $\mathbf{2\log_2}\,\boldsymbol{n} + \boldsymbol{O}(\mathbf{1})$.