Skip List

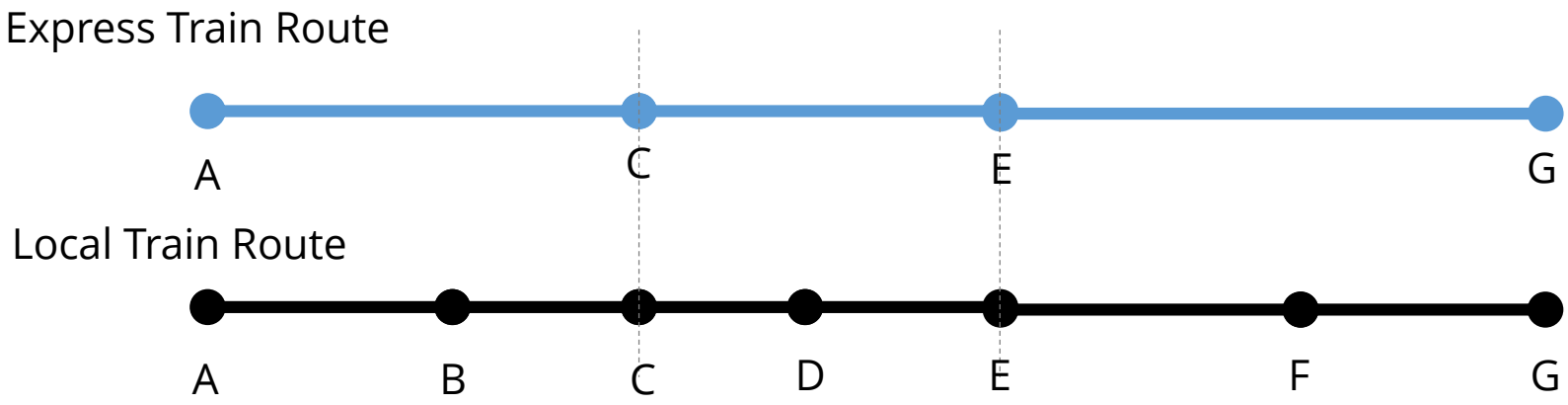
“A probabilistic data structure”

Prerequisite: Linked List, Binary Search

Reference: Data Structures & Algorithm Analysis in C++, Chapter 10.4.2

Md. Saidul Hoque Anik
onix.hoque.mist@gmail.com

Local Train vs Express Train



A Sorted Linked List

What is the complexity for searching?

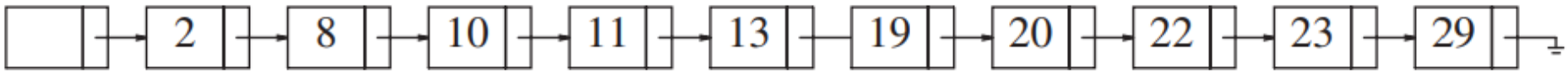


Figure 10.57 Simple linked list

A Sorted Linked List

What is the complexity for searching?

O(N), because we can’t perform binary search in it like array.

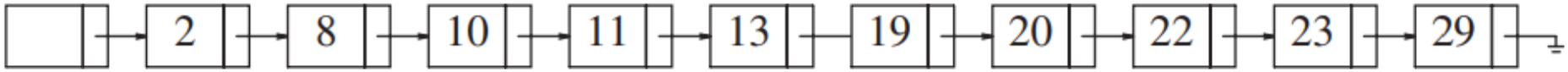


Figure 10.57 Simple linked list

Linked List with forward nodes

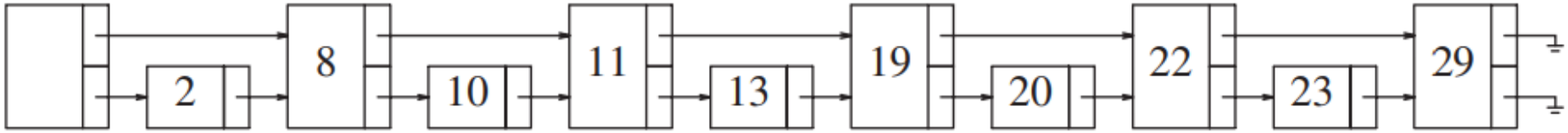


Figure 10.58 Linked list with links to two cells ahead

Linked List with forward nodes

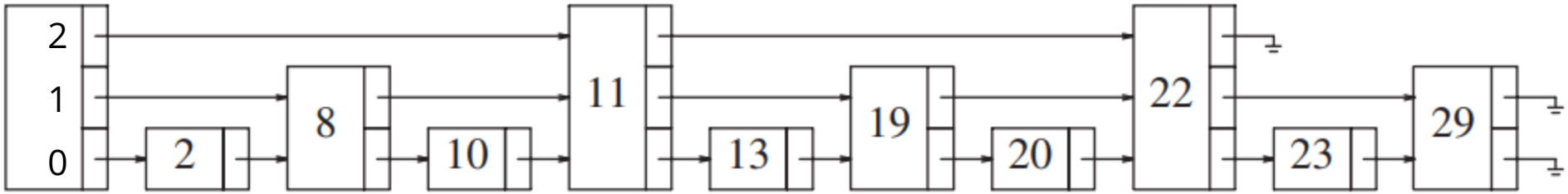


Figure 10.59 Linked list with links to four cells ahead

A Perfect Skip-List

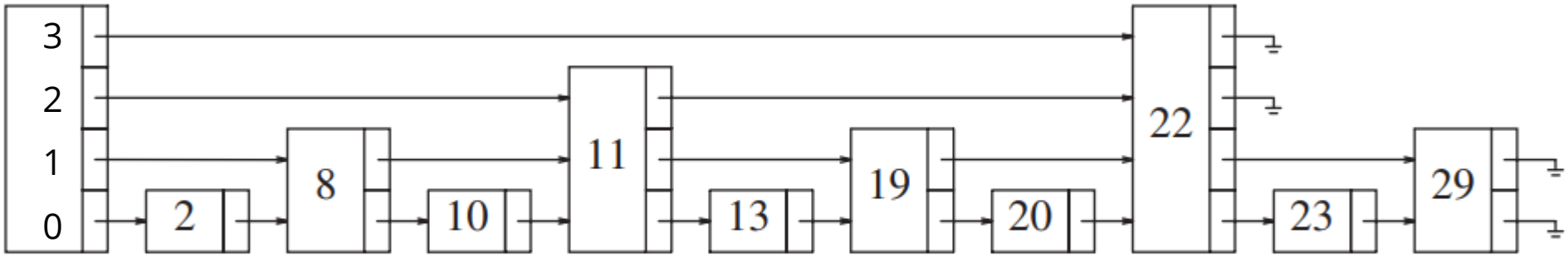


Figure 10.60 Linked list with links to 2^i cells ahead

Task

Create a perfect skip list for the following linked list.

How many levels should we need?

	4	8	11	14	23	35	41	49	53	58

Perfect Skip Lists are too Perfect!

It's difficult to maintain the 2^i property after insertion or deletion. Because we are deciding the link based on the **relative position** of the nodes.

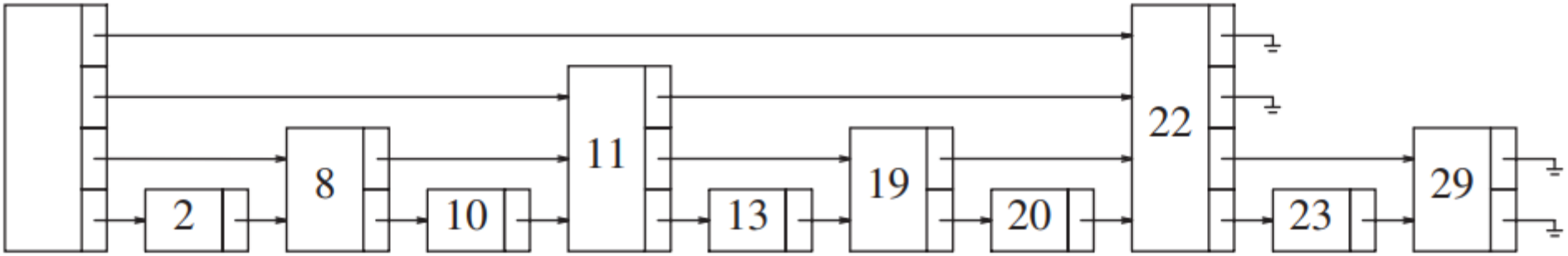
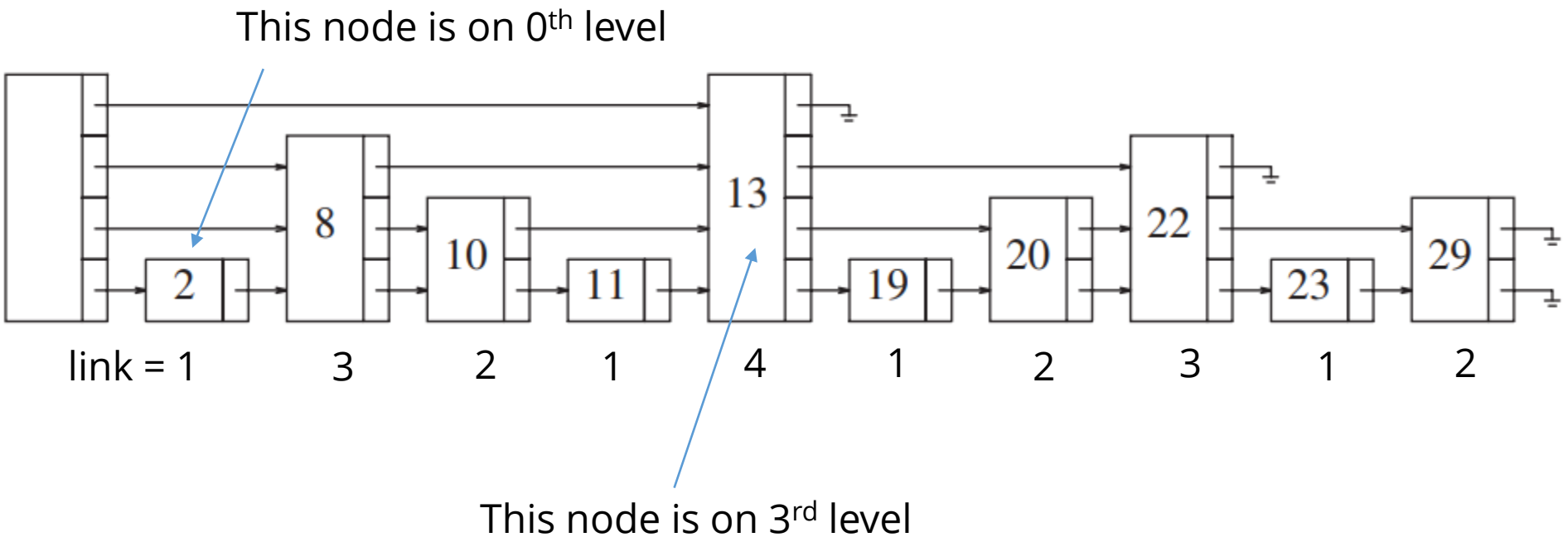


Figure 10.60 Linked list with links to 2^i cells ahead

A Randomized approach

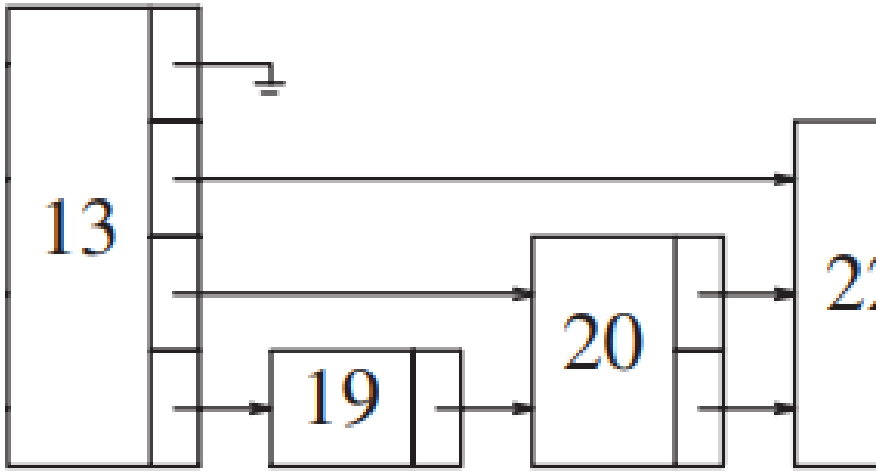
- A level-k node will be a node that has k+1 links
- The level of the node will be decided during the creation of the node. **The value will be random**.
- The i^{th} link in any level k node ($k \geq i$) links to the next node with at least i levels



Implementation

A node will have the following elements:

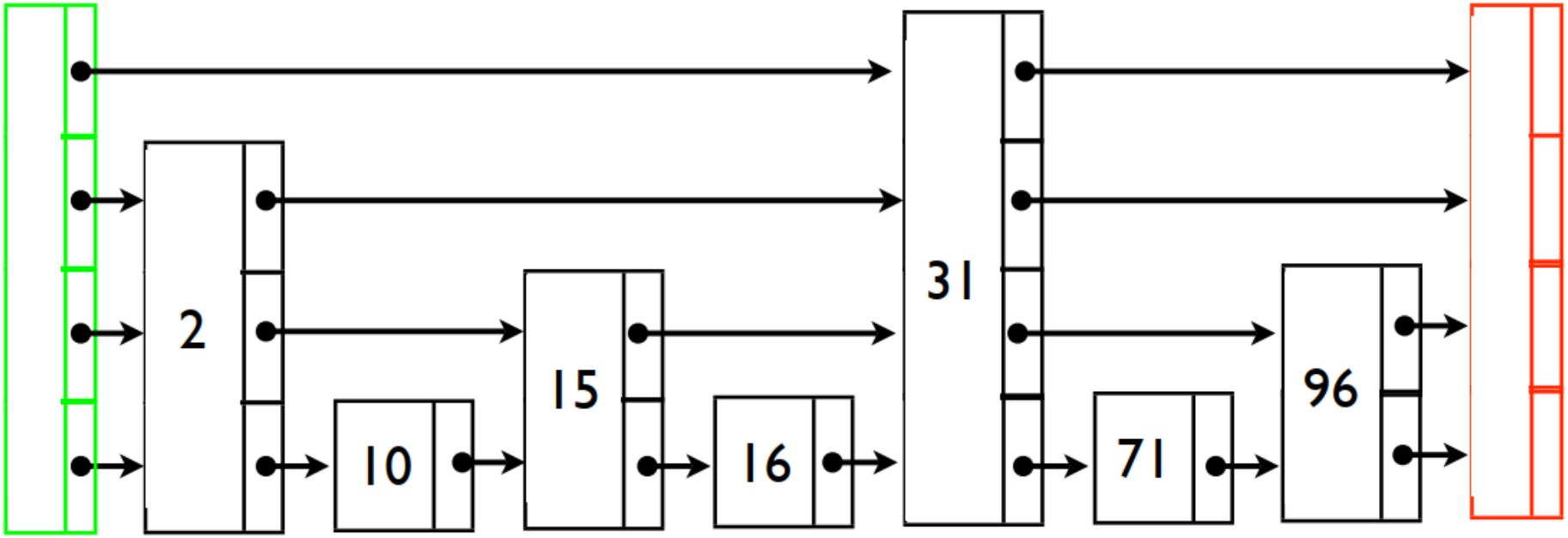
- key x 1
- forward pointer x k



Note:

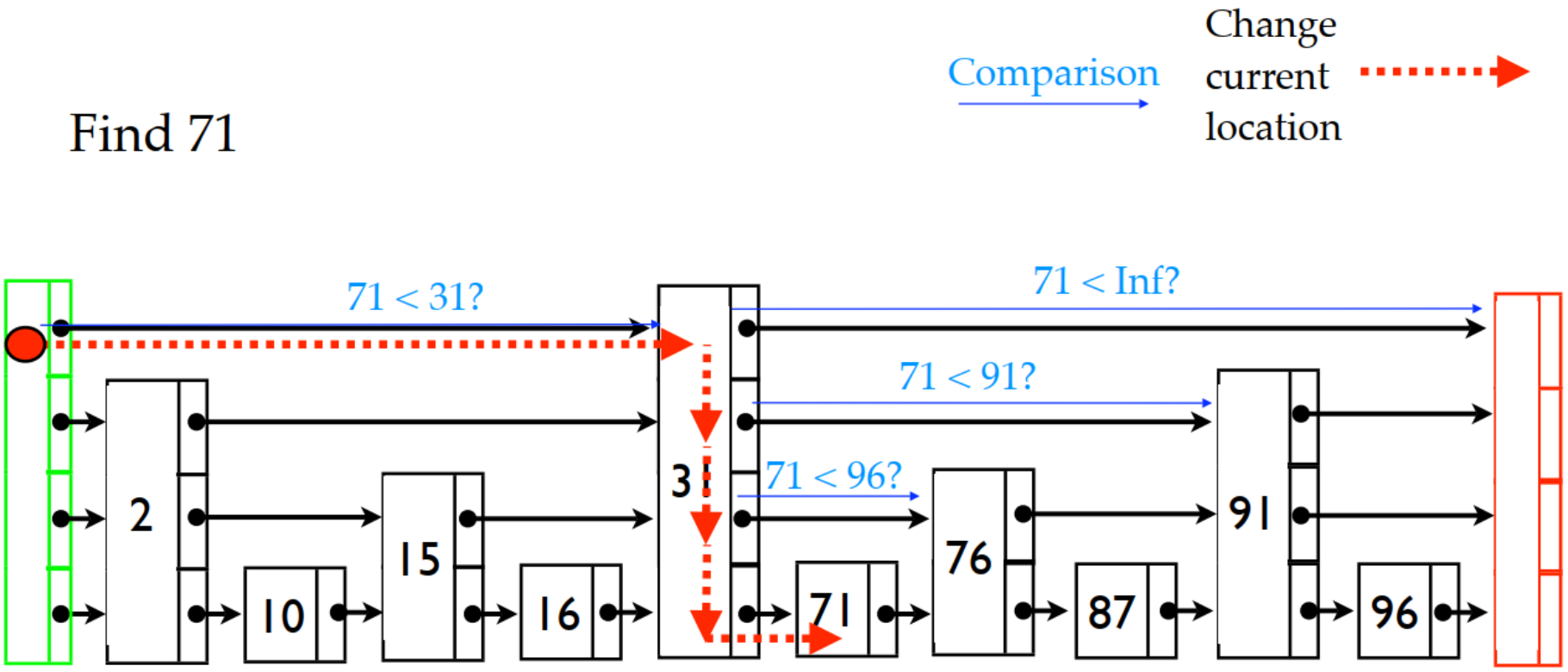
- Node structures are of variable size
- But once a node is created, its size won't change
- It's often convenient to assume that you know the maximum number of levels in advance (but this is not a requirement).

Another Example



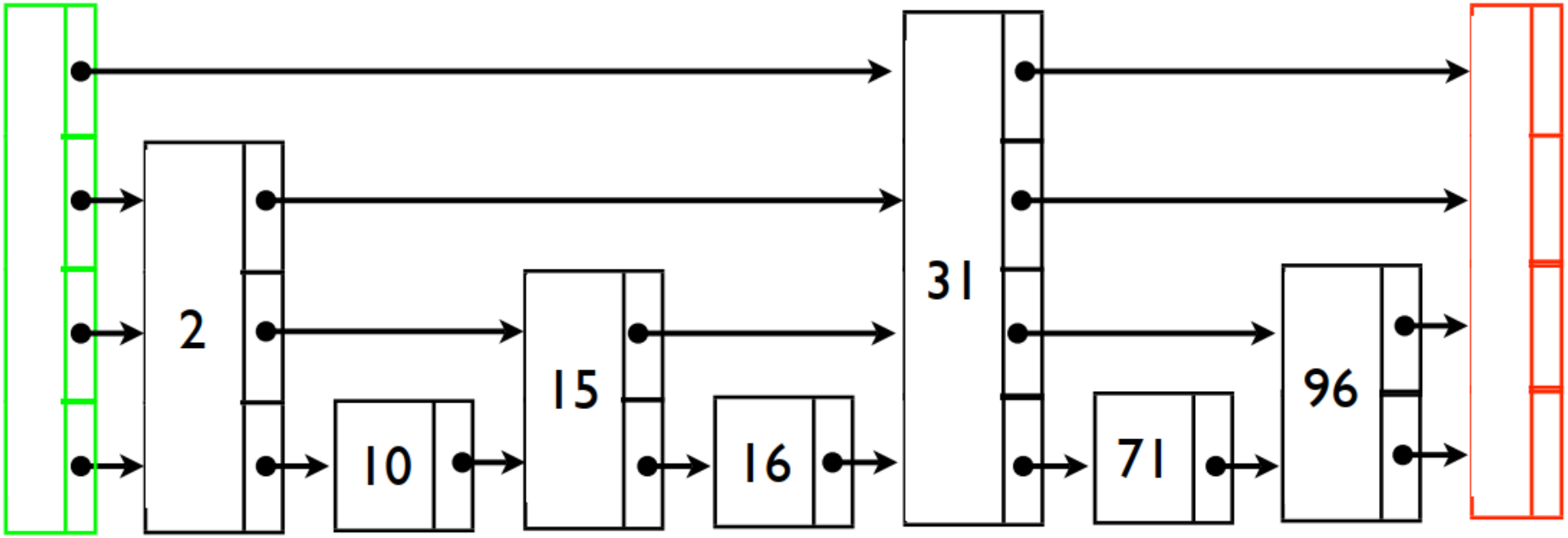
Searching in Skip List

Find 71



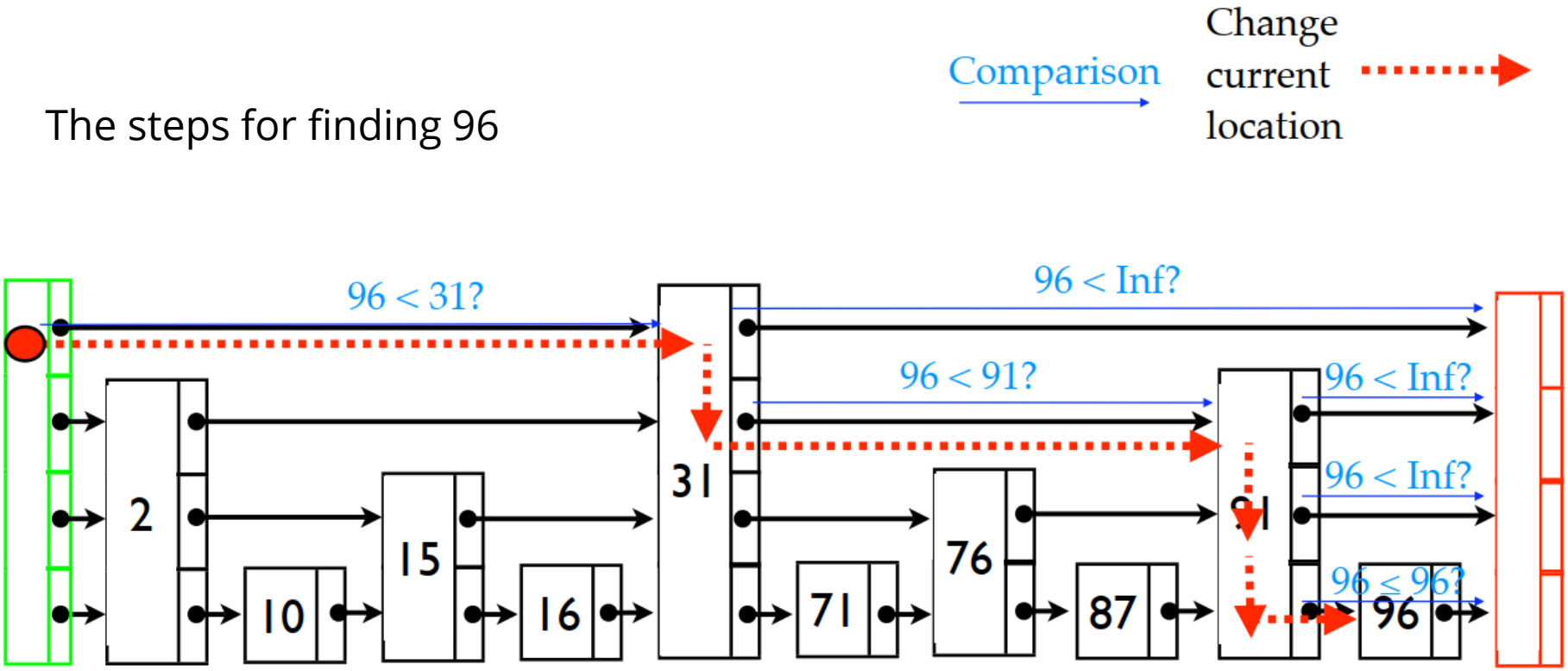
Task

Show the steps for finding 96



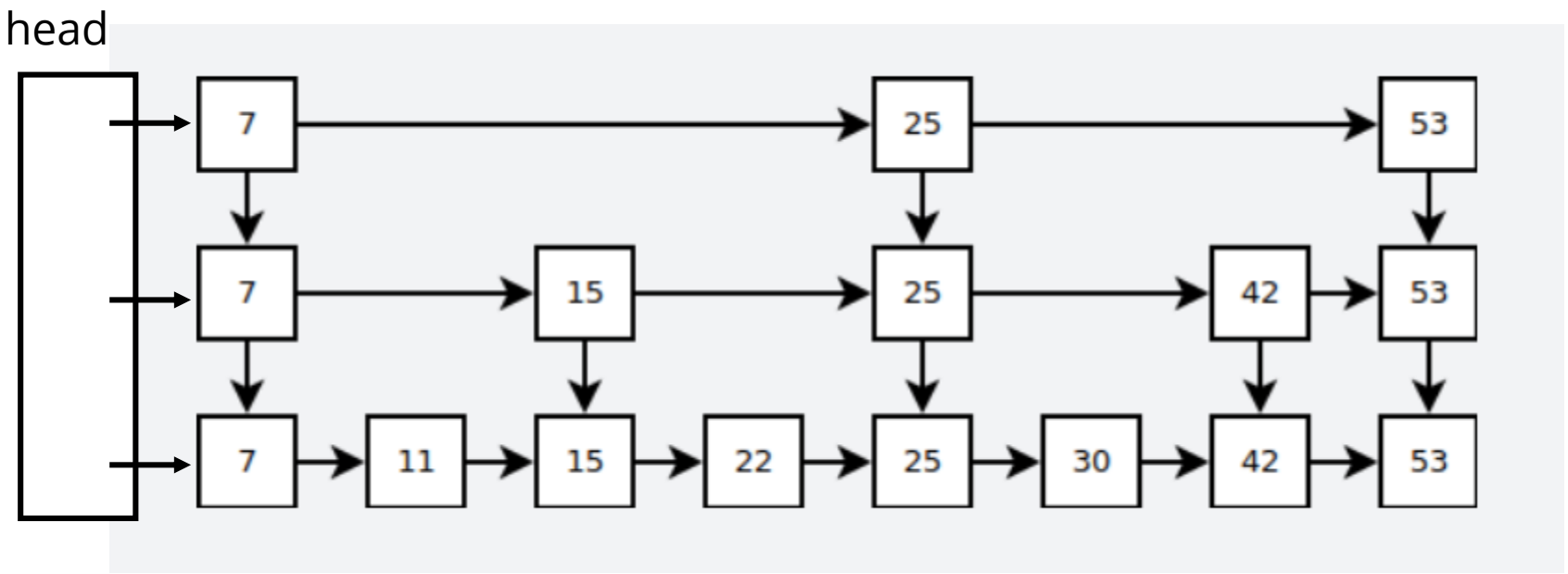
Solution

The steps for finding 96

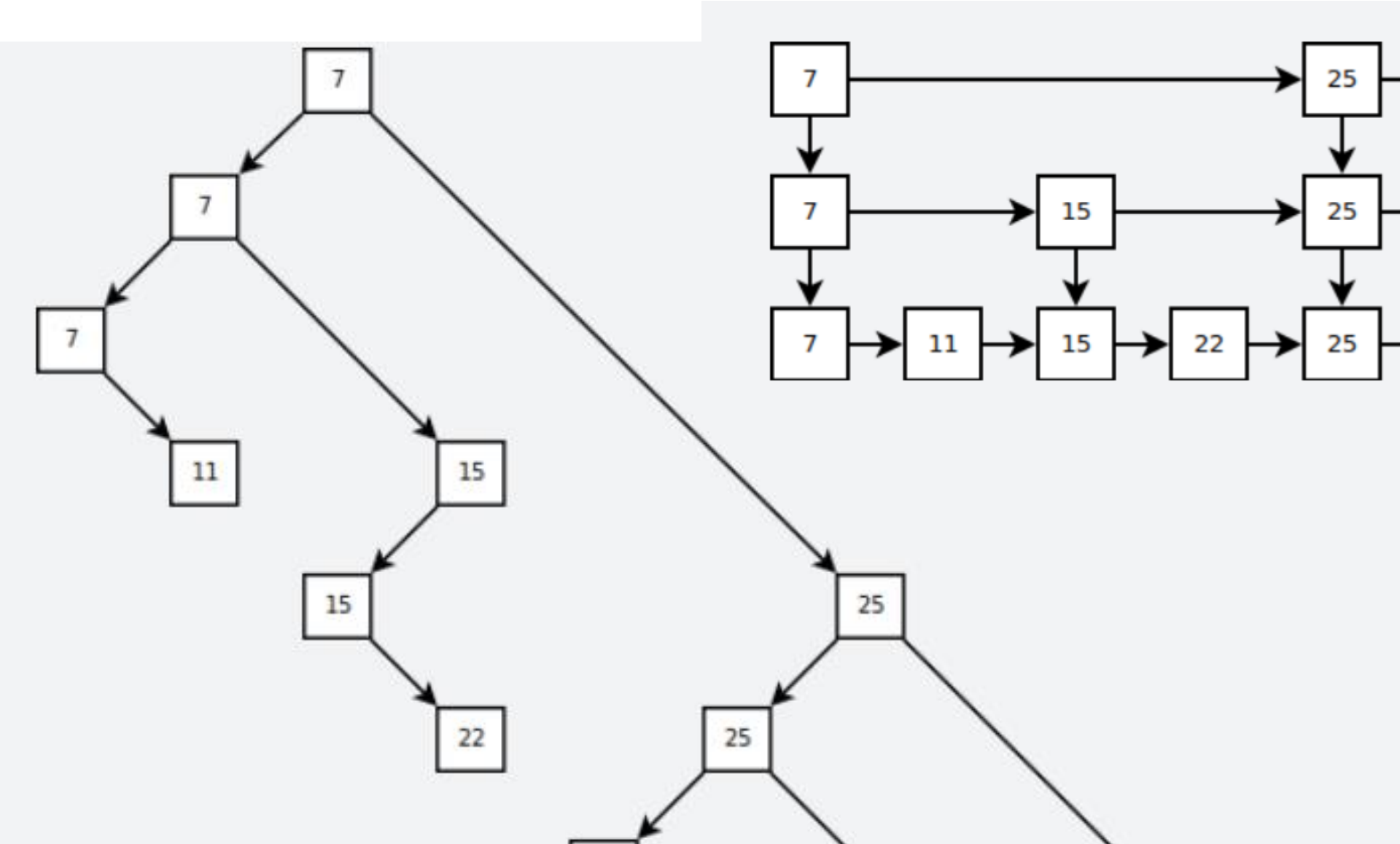


Task

Show the steps for finding 30



Skip Lists are practically BSTs



Implementation

```cpp
#include <bits/stdc++.h>
using namespace std;

class Node
{
public:
    int key;

    /// Array to hold pointers to node of different level
    Node **forward;

    Node(int key, int level)
    {
        this->key = key;
        /// Allocate memory to forward
        forward = new Node*[level+1];
        /// Fill forward array with 0(NULL)
        memset(forward, 0, sizeof(Node*)*(level+1));
    }
};
```

Implementation

```
class SkipList
{
    /// Maximum level for this skip list
    int MAXLVL;
    /// P is the fraction of the nodes with level
    /// i pointers also having level i+1 pointers
    float P;

    /// current level of skip list
    int level;

    /// pointer to header node
    Node *header;
public:
    SkipList(int, float);
    int randomLevel();
    Node* createNode(int, int);
    void insertElement(int);
    void displayList();
};
```

SkipList Constructor

```
SkipList::SkipList(int MAXLVL, float P)
{
    this->MAXLVL = MAXLVL;
    this->P = P;
    level = 0;

    // create header node and initialize key to -1
    header = new Node(-1, MAXLVL);
};
```

Getting the level number for new node

```
randomLevel()
    lvl := 1
    -- random() that returns a random value in [0...1)
    while random() < p and lvl < MaxLevel do
        lvl := lvl + 1
    return lvl
```

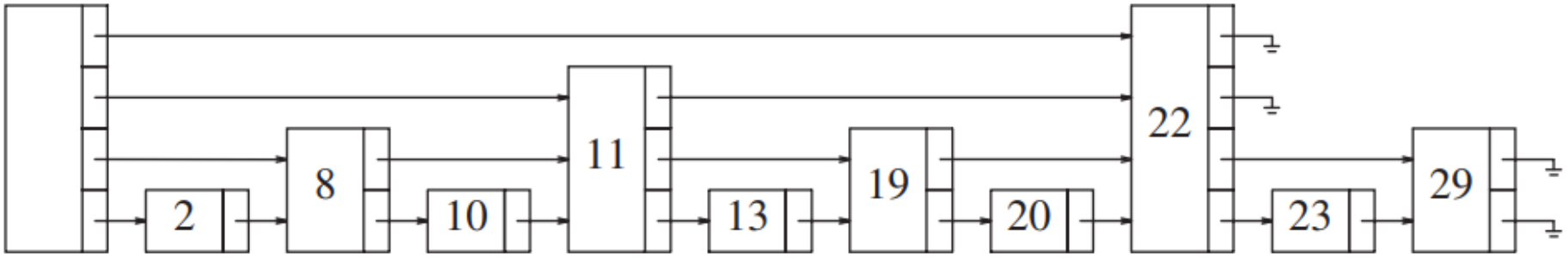


Figure 10.60 Linked list with links to 2^i cells ahead

Getting the level number for new node

```
// create random level for node
int SkipList::randomLevel()
{
    float r = (float)rand()/RAND_MAX;
    int lvl = 0;
    while (r < P && lvl < MAXLVL)
    {
        lvl++;
        r = (float)rand()/RAND_MAX;
    }
    return lvl;
};
```

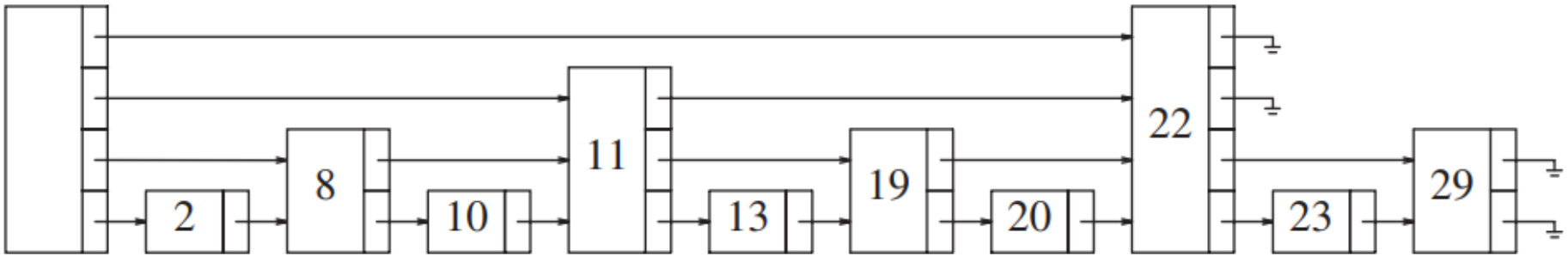
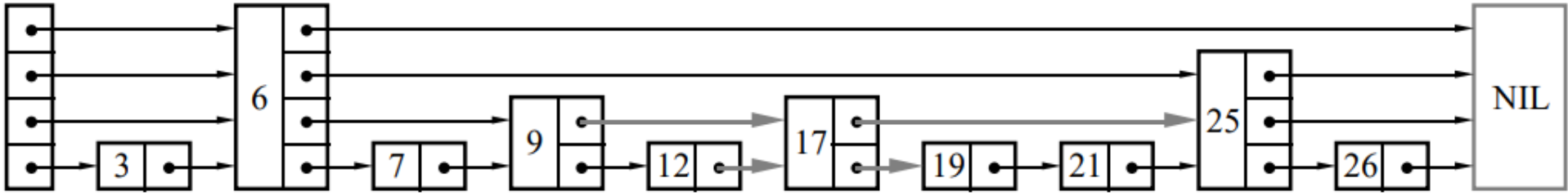


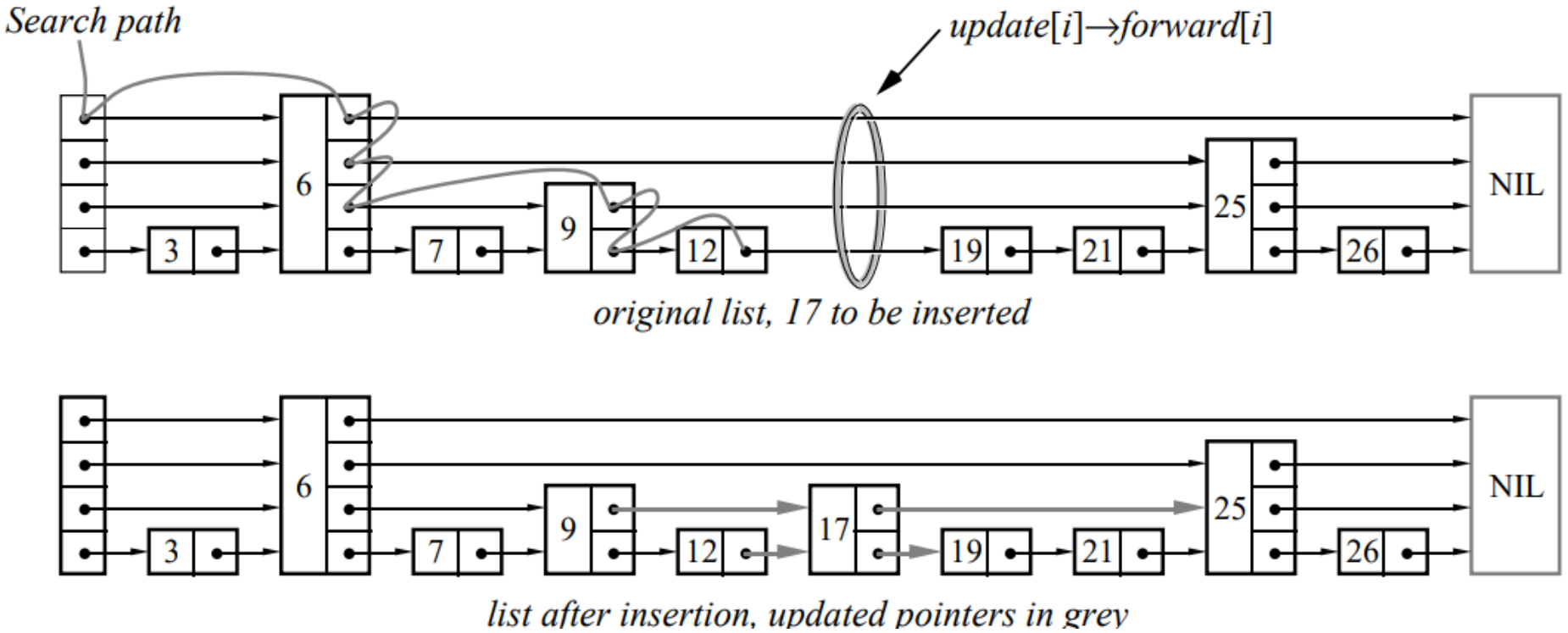
Figure 10.60 Linked list with links to 2^i cells ahead

Search Pseudocode

```
Search(list, searchKey)
    x := list→header
    -- loop invariant: x→key < searchKey
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]
    -- x→key < searchKey ≤ x→forward[1]→key
    x := x→forward[1]
    if x→key = searchKey then return x→value
        else return failure
```



Insertion Operation



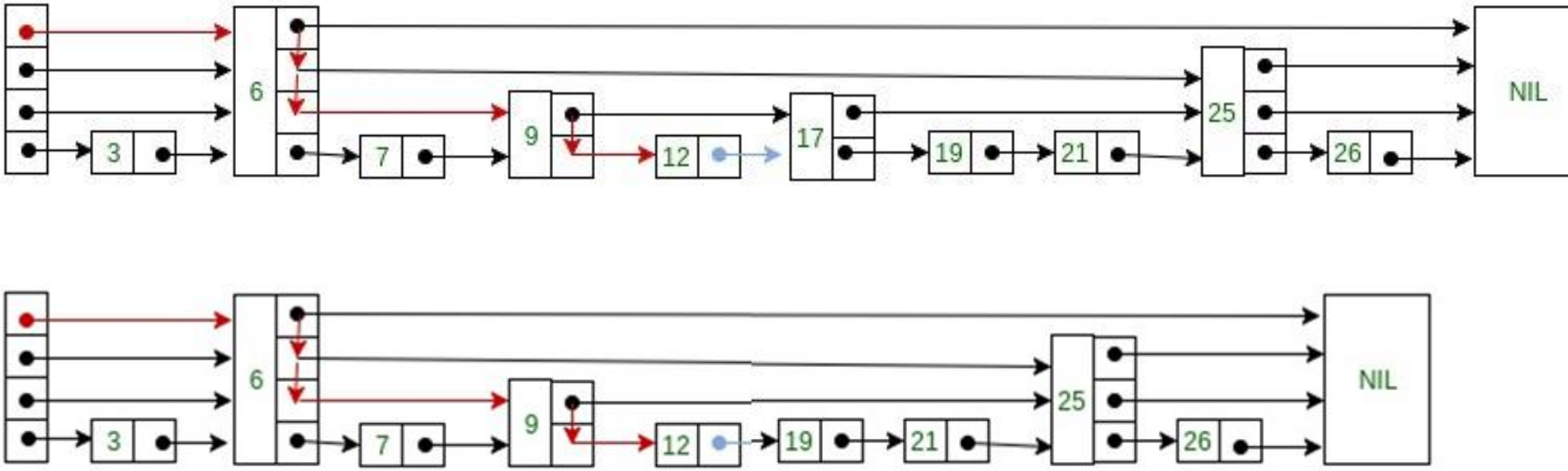
Insertion Pseudocode

```
Insert(list, searchKey, newValue)
    local update[1..MaxLevel]
    x := list→header
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]

        update[i] := x
    x := x→forward[1]
    if x→key = searchKey then x→value := newValue
    else
        lvl := randomLevel()
        if lvl > list→level then
            for i := list→level + 1 to lvl do
                update[i] := list→header
            list→level := lvl
        x := makeNode(lvl, searchKey, value)
        for i := 1 to level do
            x→forward[i] := update[i]→forward[i]
            update[i]→forward[i] := x
```

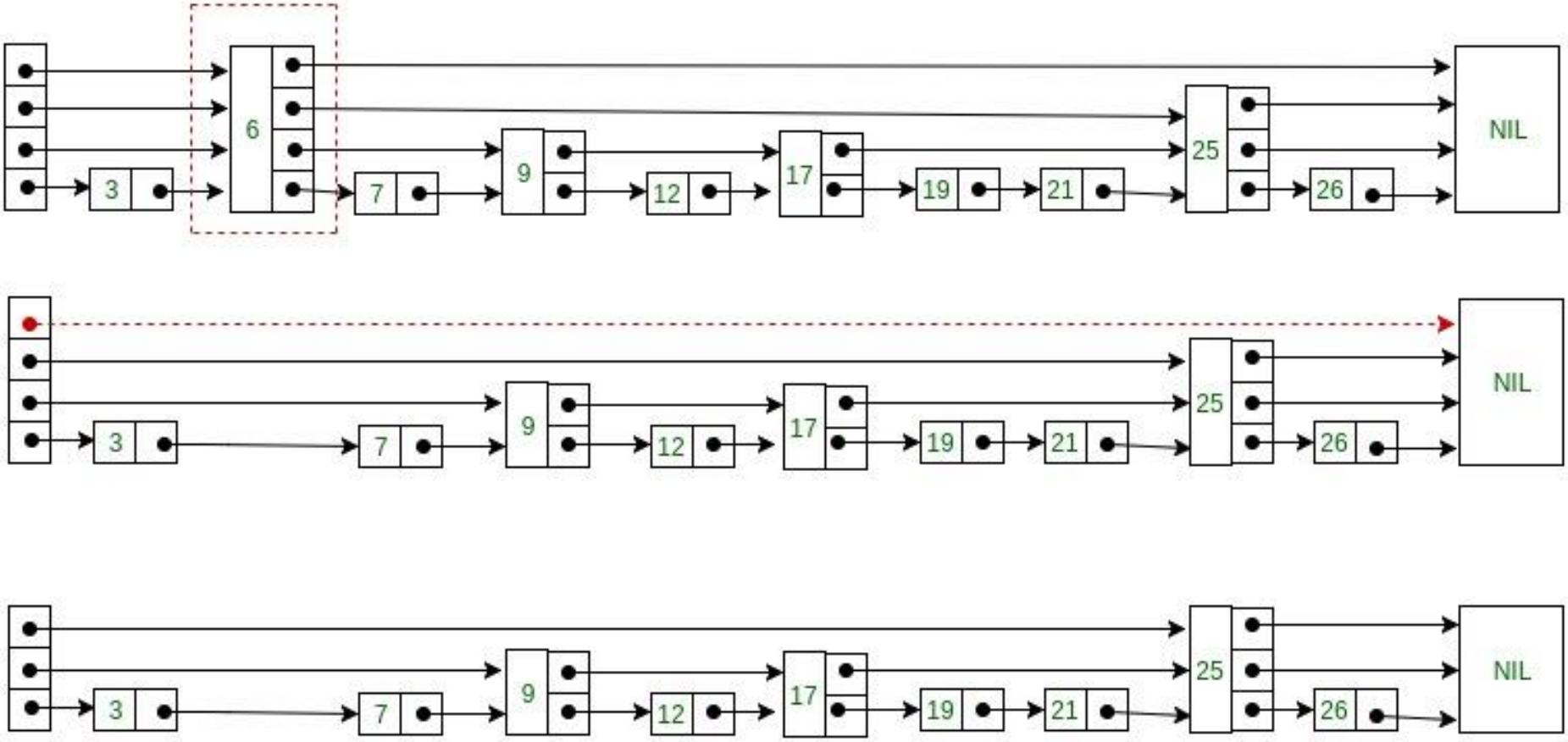
Deletion Operation

Deletion of 17



Deletion Operation

Deletion of 6



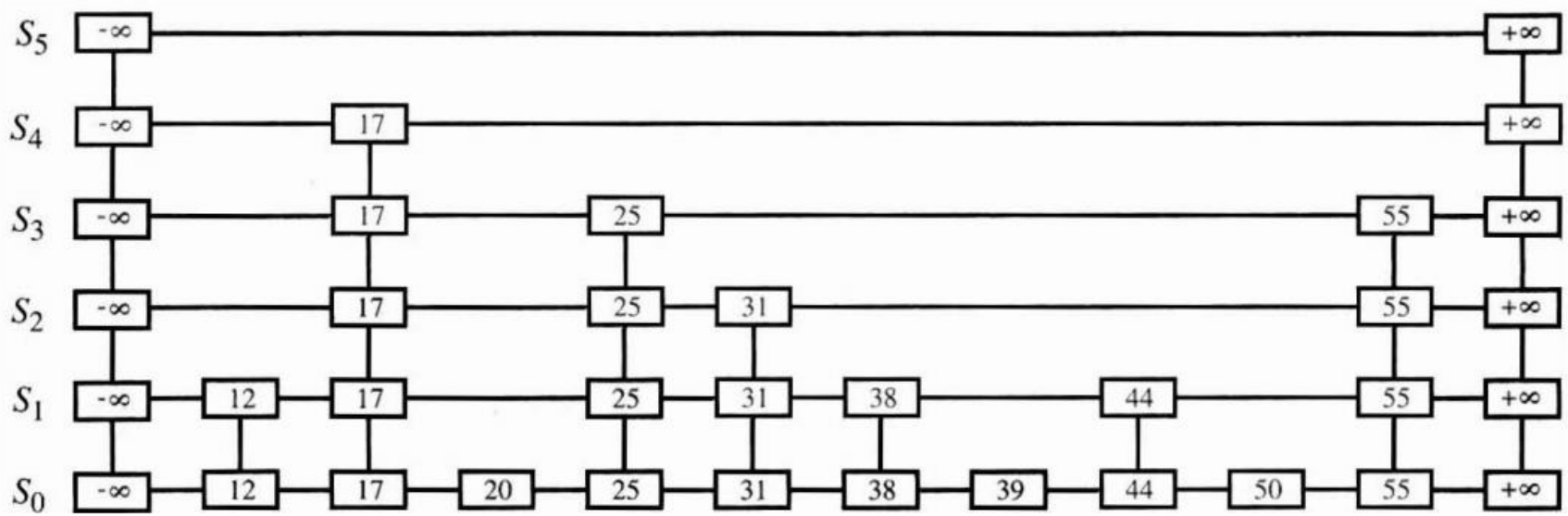
Deletion Pseudocode

```
Delete(list, searchKey)
    local update[1..MaxLevel]
    x := list→header
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]
        update[i] := x
    x := x→forward[1]
    if x→key = searchKey then
        for i := 1 to list→level do
            if update[i]→forward[i] ≠ x then break
            update[i]→forward[i] := x→forward[i]
        free(x)
        while list→level > 1 and
                list→header→forward[list→level] = NIL do
            list→level := list→level – 1
```

Run Time Analysis for Search Operation

What is the average running time to find an element in a skip list?

Note: It will also determine the running time for Insertion and Deletion



Average Runtime for Search

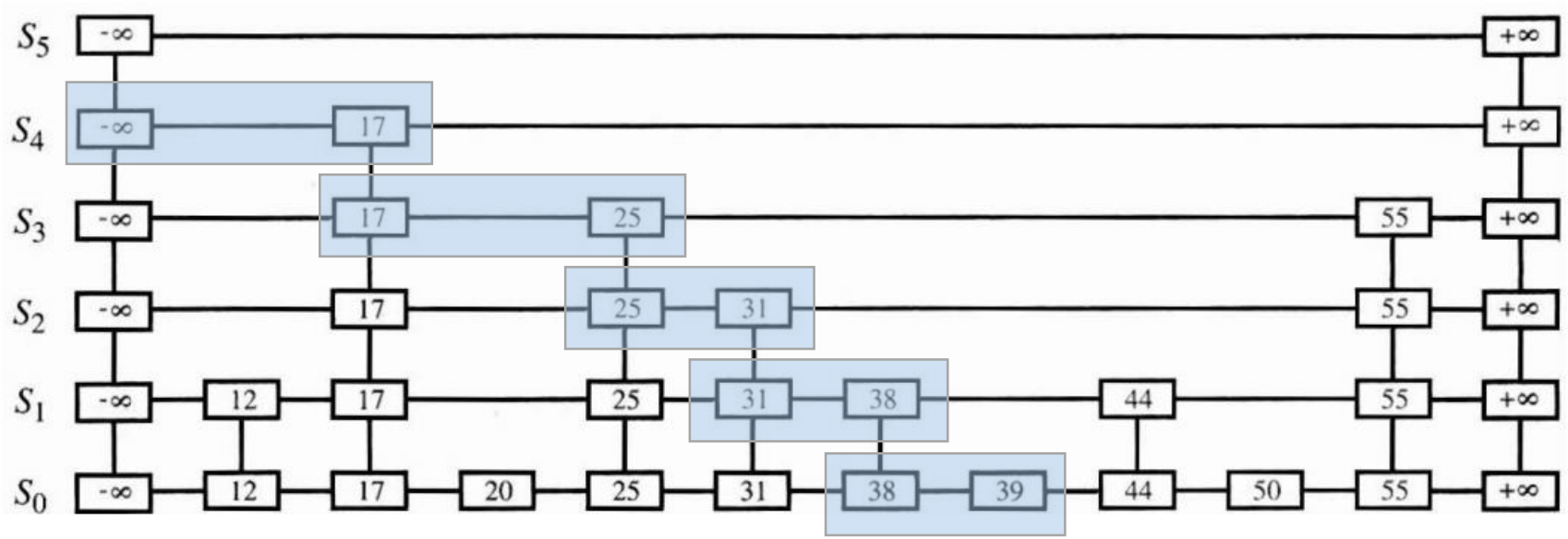
Number of entries visited =

entries visited in level h-1

+ # entries visited in level h-2

+ ....

+ # entries visited in level 0



Average Runtime for Search

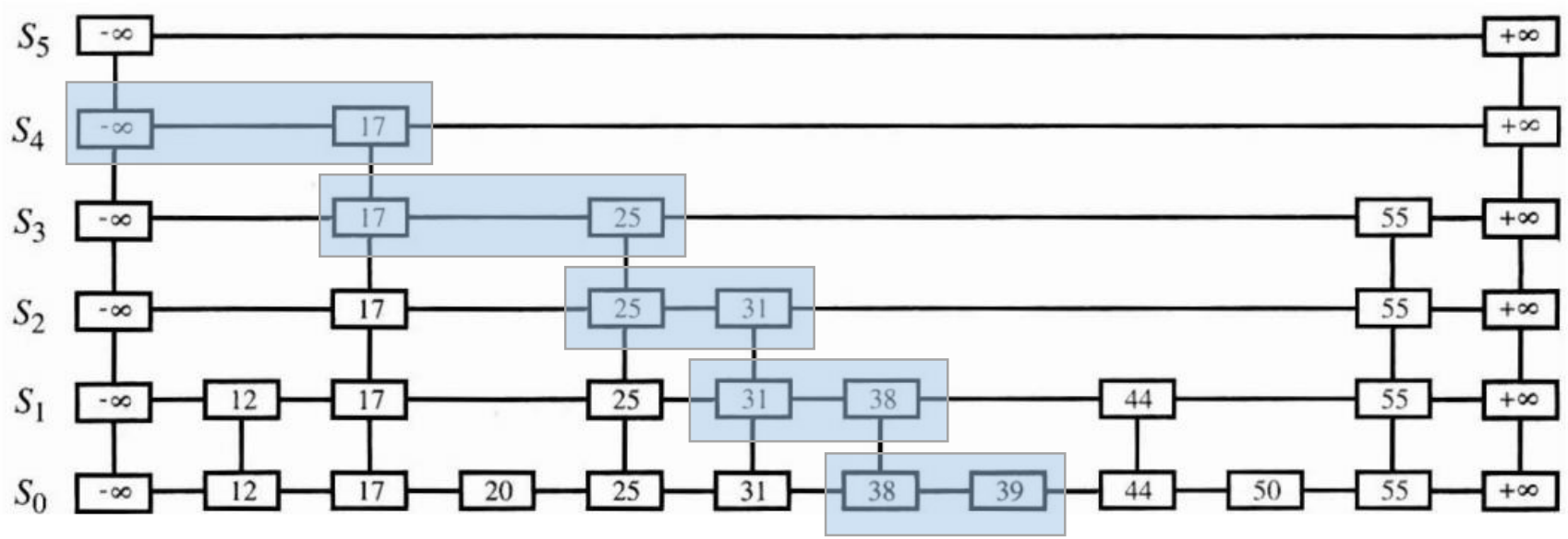
Number of entries visited =

1 + # right traversals on level h-1

+1 + # right traversals on level h-2

+ ....

+1 + # right traversals on level 0



Average Runtime for Search

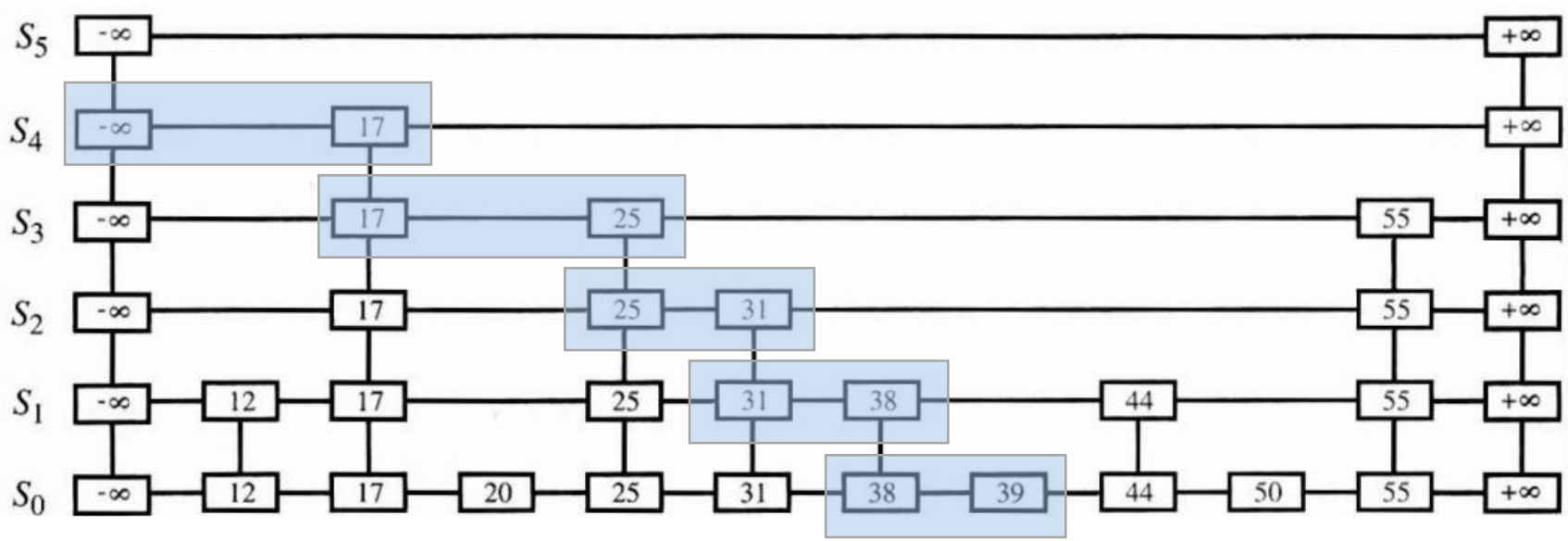
Number of entries visited =

h + # right traversals on level h-1

+ # right traversals on level h-2

+ ....

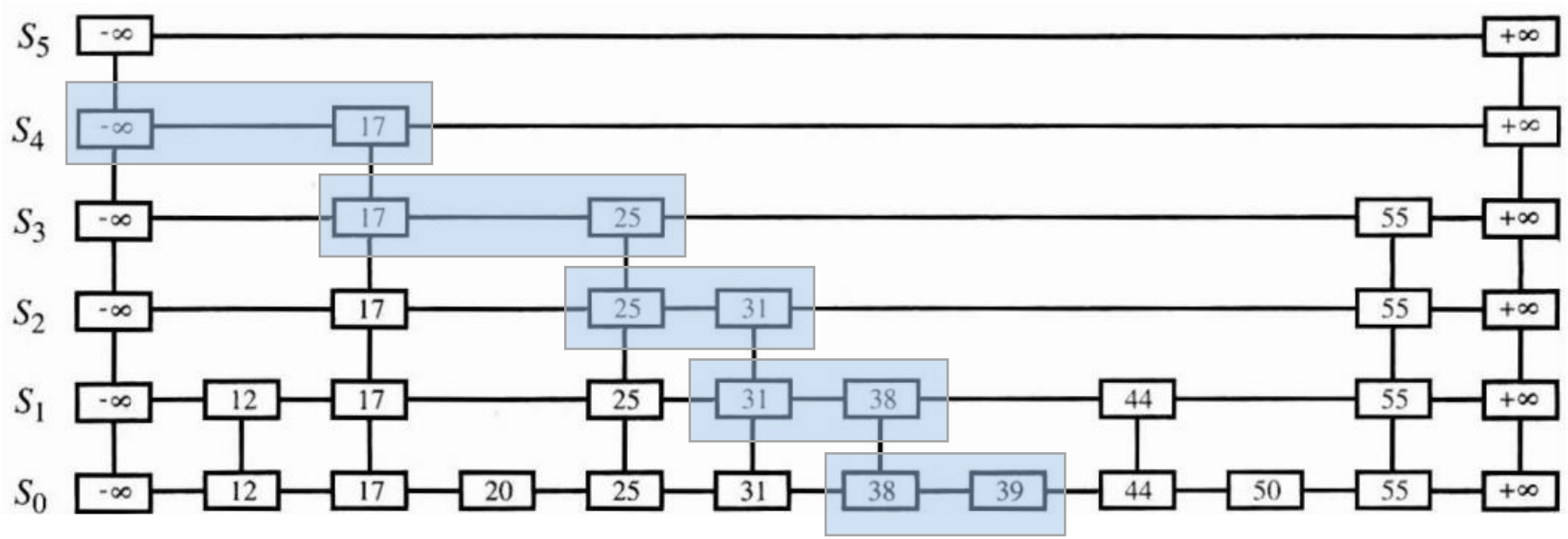
+ # right traversals on level 0



Average Runtime for Search

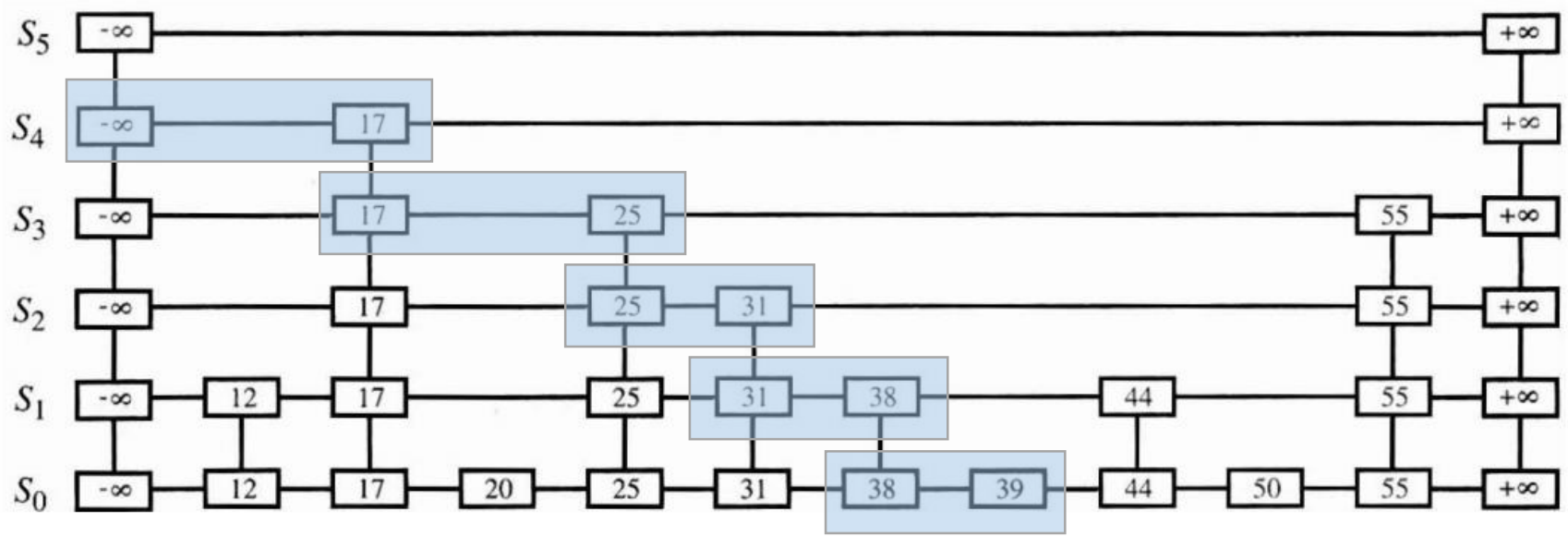
Number of entries visited =

h + h x (average right traversals on one level)



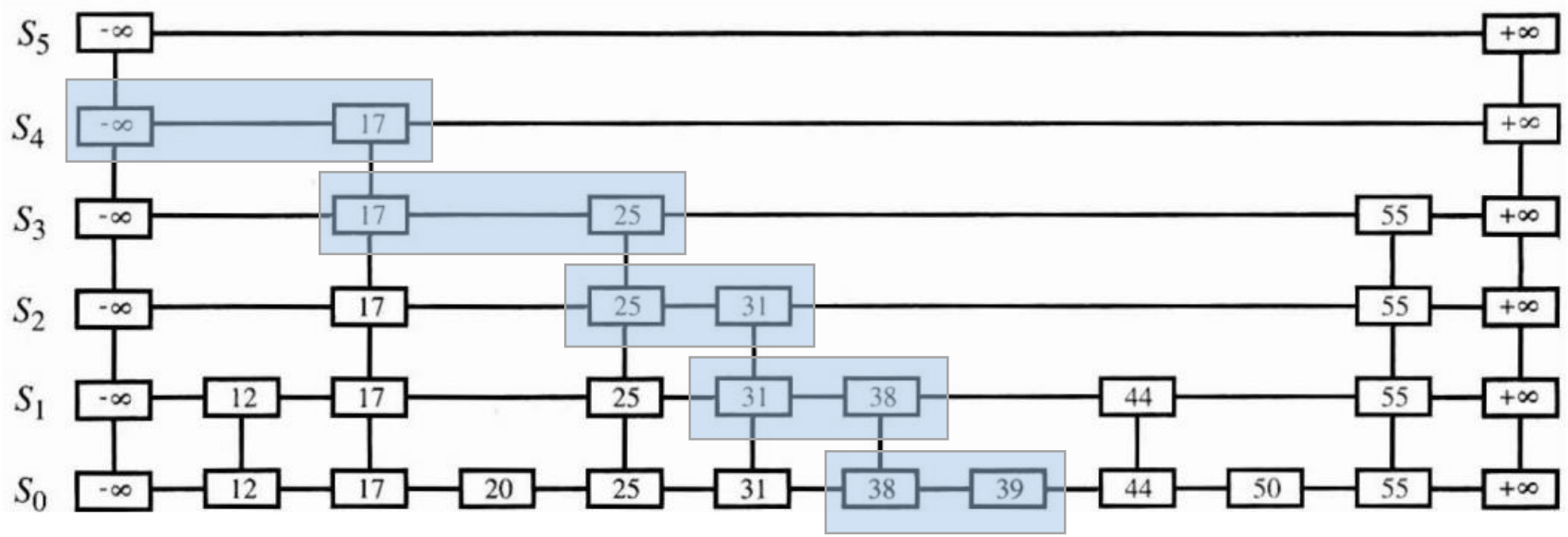
Average Runtime for Search

Average Runtime = avg(h) + avg(h) x (average right traversals on one level)



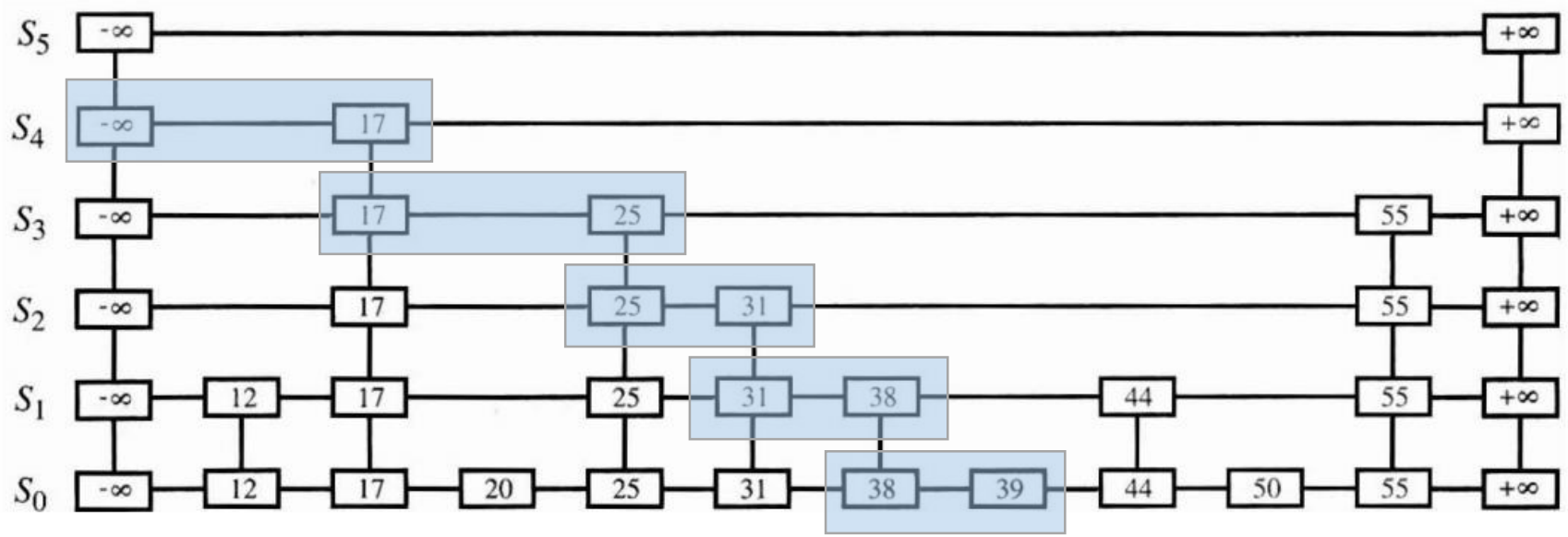
Finding the average height : avg(h)

Let’s find how likely a skip list has height h.



Finding the average height : avg(h)

Probability that a tower has height 0 = 1



For a single tower

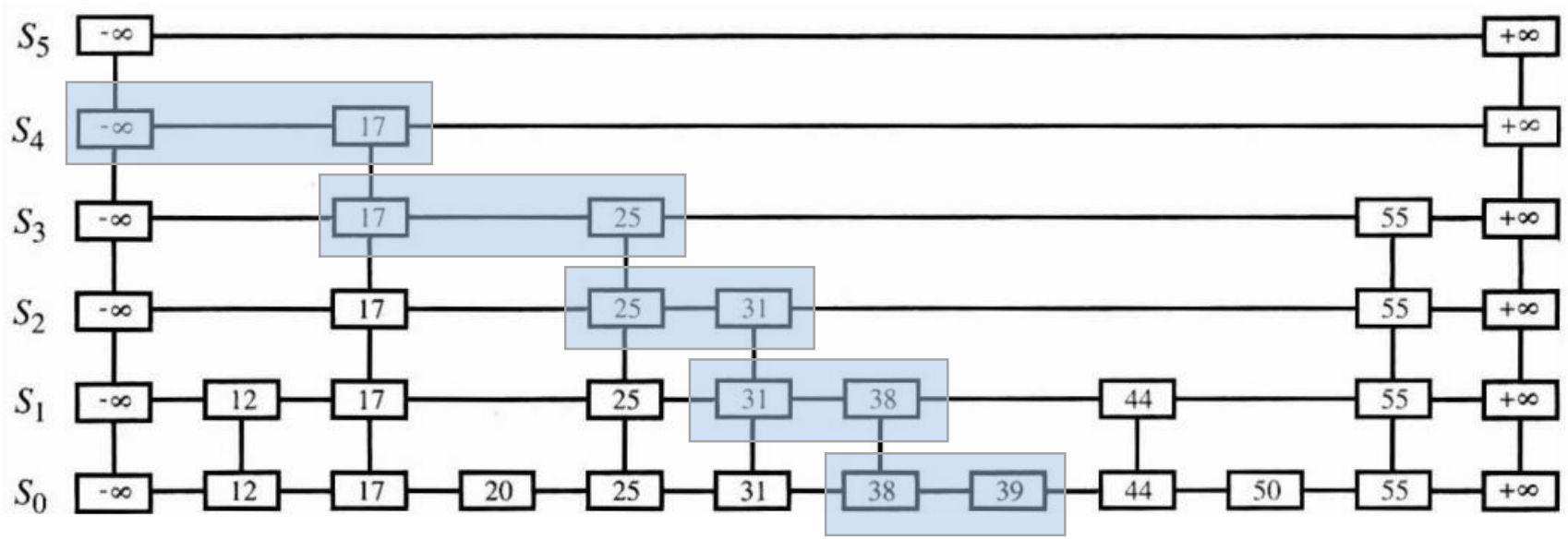
For p = 0.5 = ½,

P[height = 1] =$1/2^1$

P[height = 2] =$1/2^2$

...

P[height = i] =$1/2^i$



Finding the average height : avg(h)

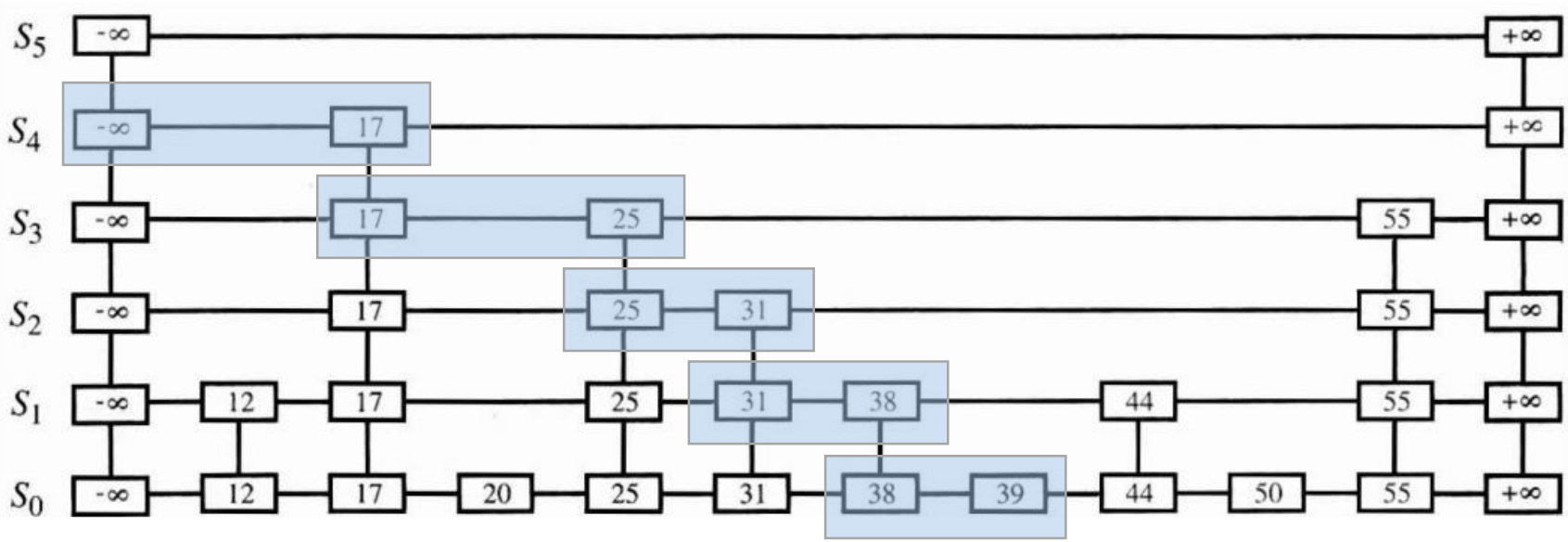
Skip list has height h = (height of tower 1 = h)

OR (height of tower 2 = h)

OR (height of tower 3 = h)

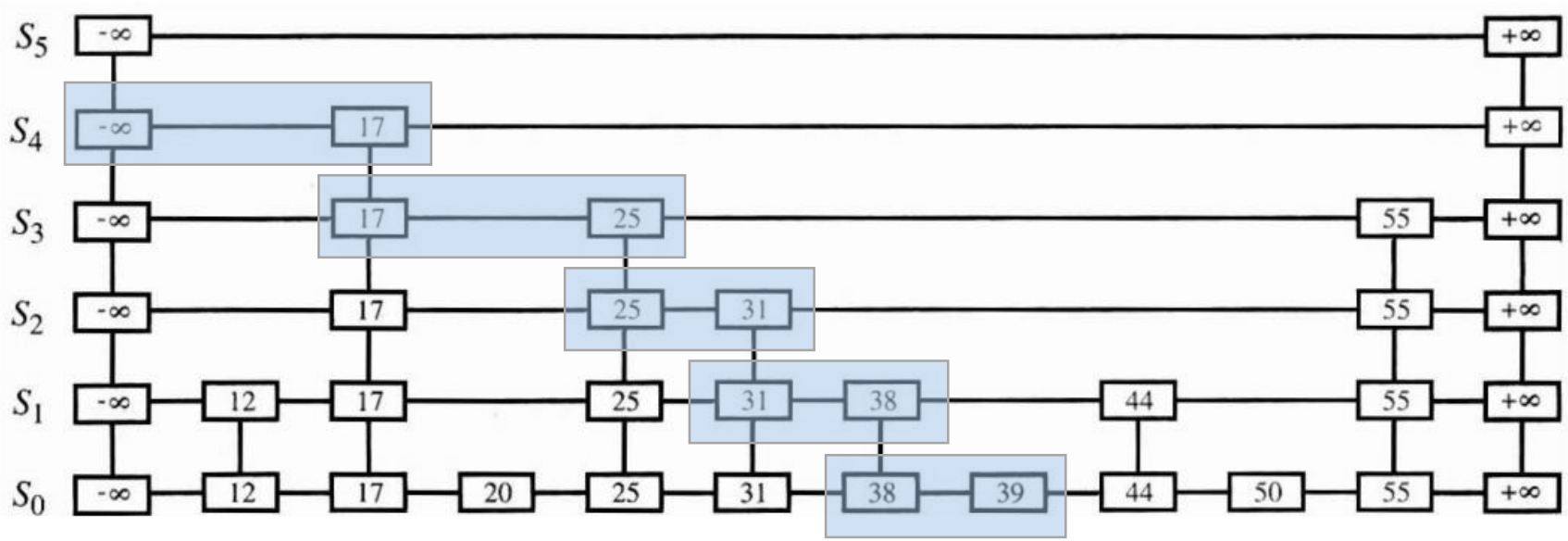
...

OR (height of tower n = h)



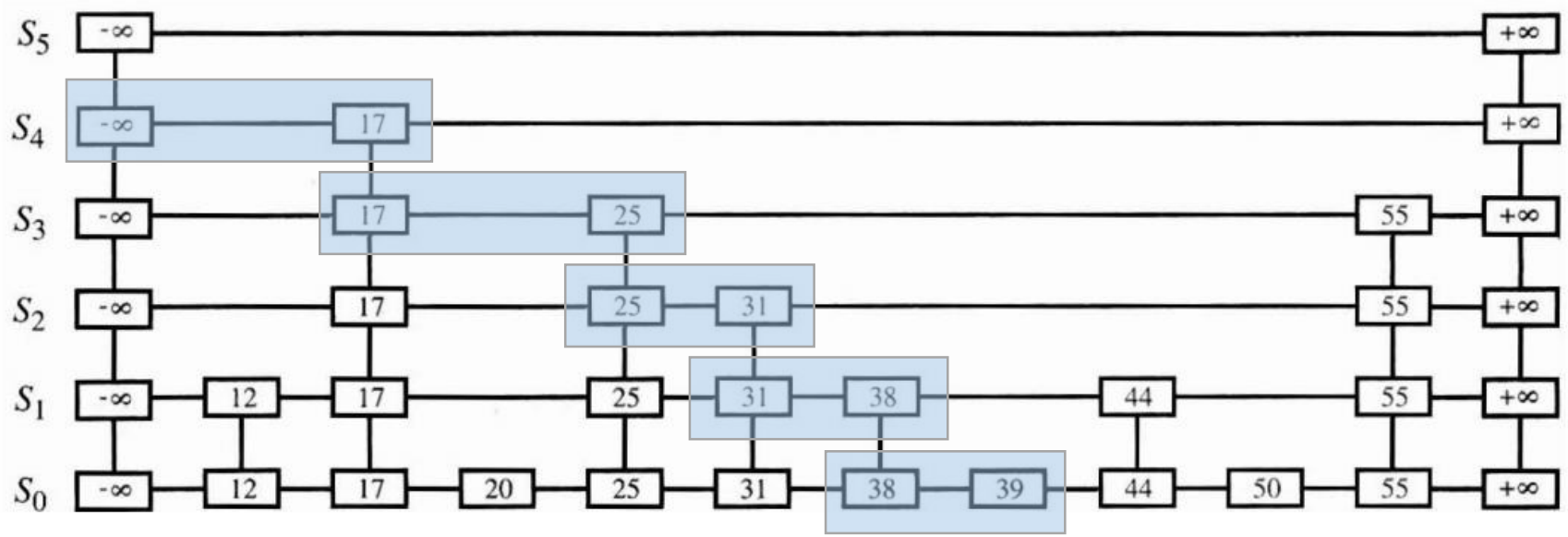
Finding the average height : avg(h)

```
P[ Skip list has height h ] = P[    (height of tower 1 = h)
                                    OR (height of tower 2 = h)
                                    OR (height of tower 3 = h)
                                 ...
                                    OR (height of tower n = h) ]
```



Finding the average height : avg(h)

$$\mathbb{P}[\text{ Skip list has height } h\ ] \;\tilde{=}\; \mathbb{P}[\text{ (height of tower 1 = } h)\ ] + \mathbb{P}[\text{ (height of tower 2 = } h)\ ] + \mathbb{P}[\text{ (height of tower 3 = } h)\ ] + \ldots + \mathbb{P}[\text{ (height of tower n = } h)\ ]$$



Finding the average height : avg(h)

$$P[\text{ Skip list has height } h\text{ }] = \frac{1}{2^h} + \frac{1}{2^h} + \ldots + \frac{1}{2^h}$$

$$= \frac{n}{2^h}$$

Finding the average height : avg(h)

$$P[\text{ Skip list has height } h\text{ }] = \frac{1}{2^h} + \frac{1}{2^h} + \ldots + \frac{1}{2^h}$$

$$= \frac{n}{2^h}$$

What is P[Skip list has height 3log(n)] ?

Finding the average height : avg(h)

$$P[\text{ Skip list has height } h\text{ }] = \frac{1}{2^h} + \frac{1}{2^h} + \ldots + \frac{1}{2^h}$$

$$= \frac{n}{2^h}$$

What is P[Skip list has height 2log(n)] ?

Finding the average height : avg(h)

$$P[\text{ Skip list has height } h\text{ }] = \frac{1}{2^h} + \frac{1}{2^h} + \ldots + \frac{1}{2^h}$$

$$= \frac{n}{2^h}$$

What is P[Skip list has height log(n)] ?

Finding the average height : avg(h)

$$P[\text{ Skip list has height } h\text{ }] = \frac{1}{2^h} + \frac{1}{2^h} + \ldots + \frac{1}{2^h}$$

$$= \frac{n}{2^h}$$

$$P[\text{ Skip list has height } \log(n)\text{ }] = 1$$

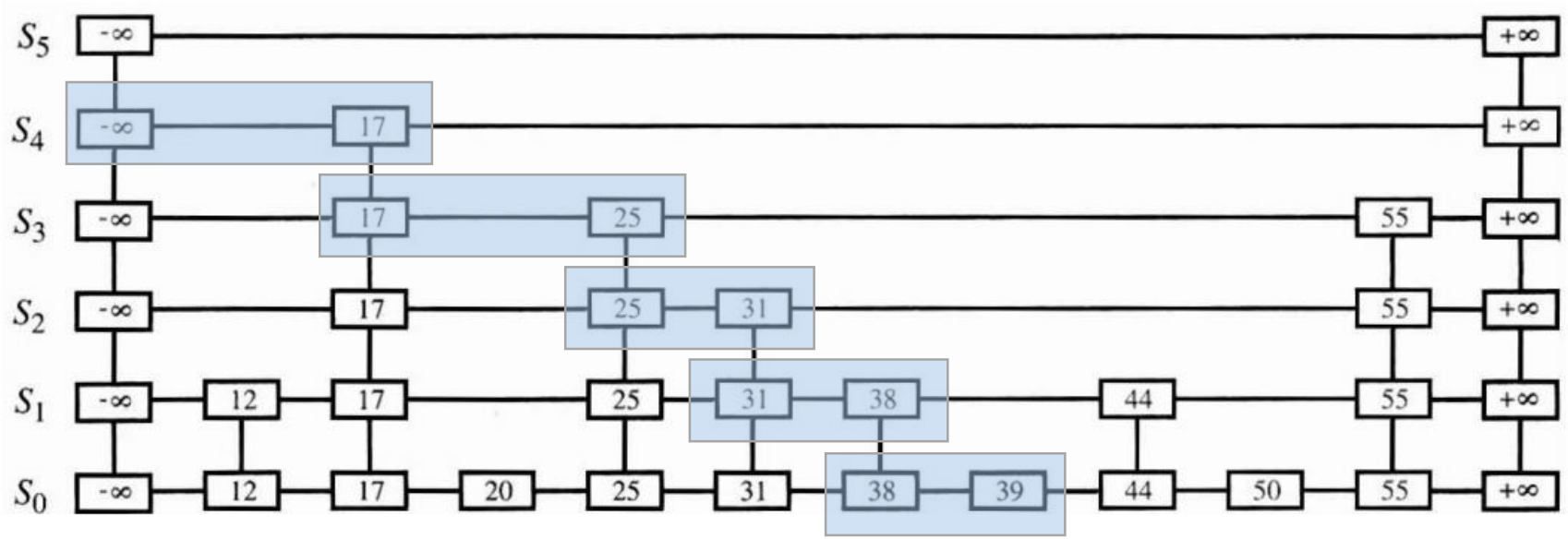
Finding the average height : avg(h)

Avg(h) of a Skip list with n entries = log(n)

Average Runtime for Search

Average Runtime = avg(h) + avg(h) x (average right traversals on one level)

avg(h) of a Skip list with n entries = log(n)

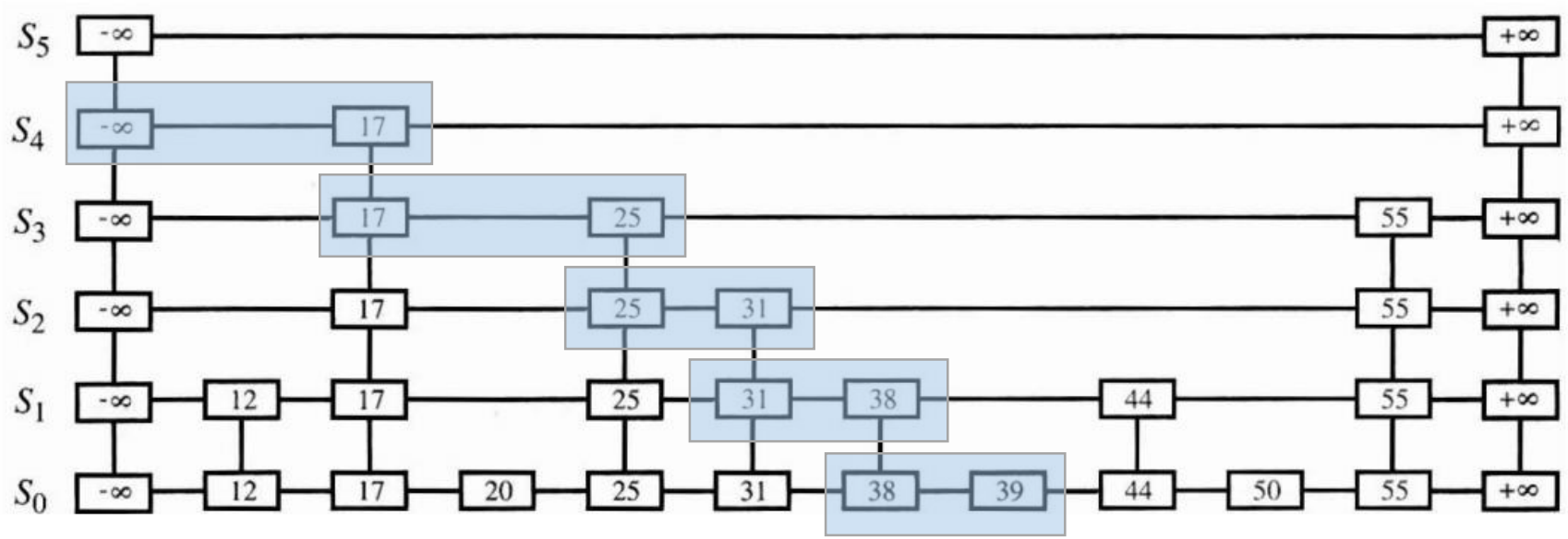


Average Runtime for Search

Average Runtime = avg(h) + avg(h) x (average right traversals on one level)

avg(h) of a Skip list with n entries = log(n)

Avg # right moves = ?

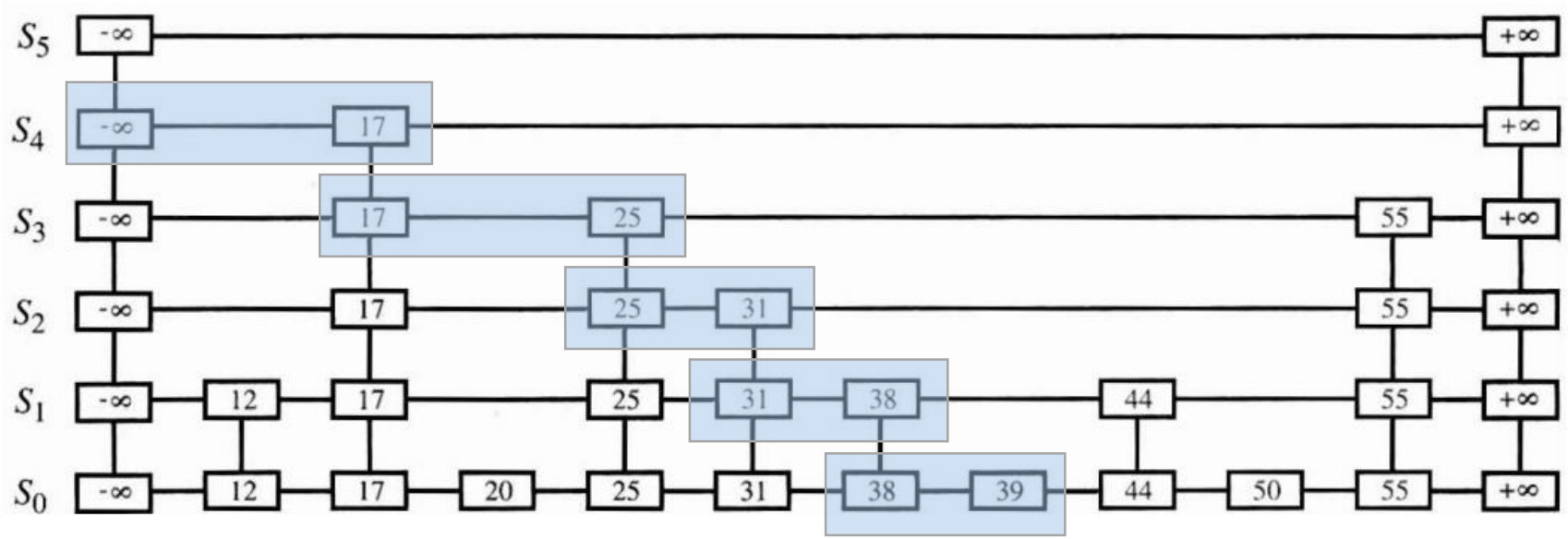


Average Runtime for Search

Average Runtime = avg(h) + avg(h) x (average right traversals on one level)

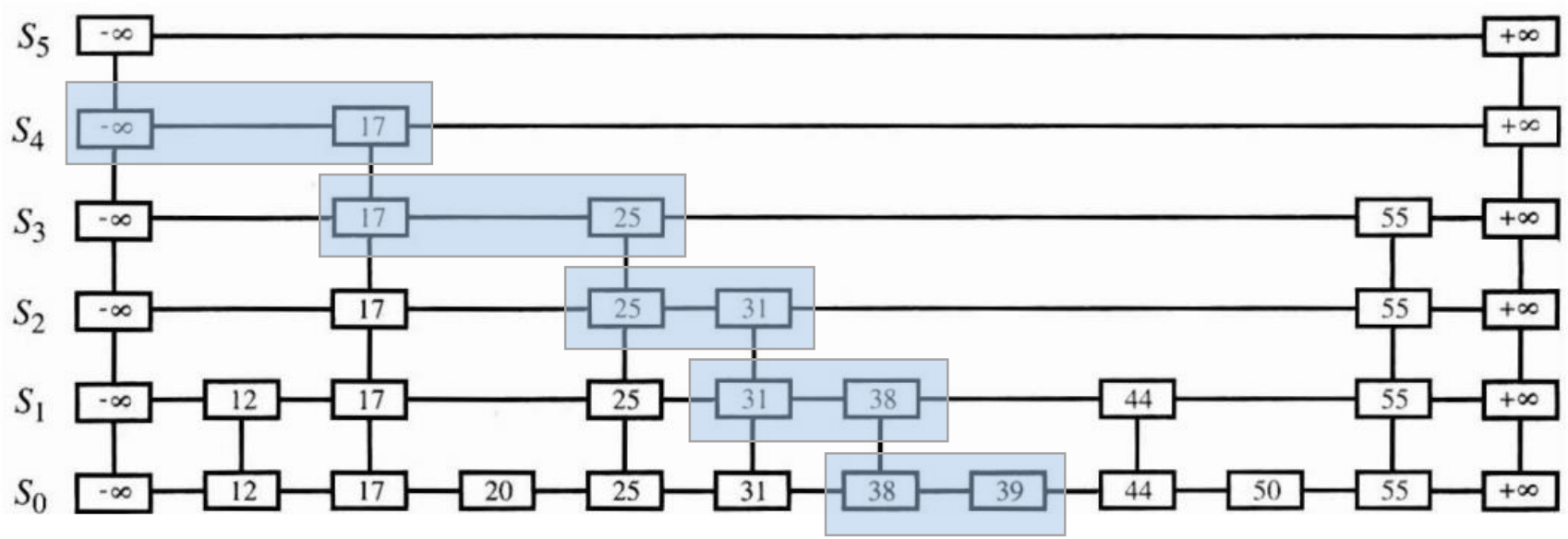
avg(h) of a Skip list with n entries = log(n)

Avg # right moves = 0 * (0.5) + 1*(0.5)2 + 2*(0.5)3 ... = 1



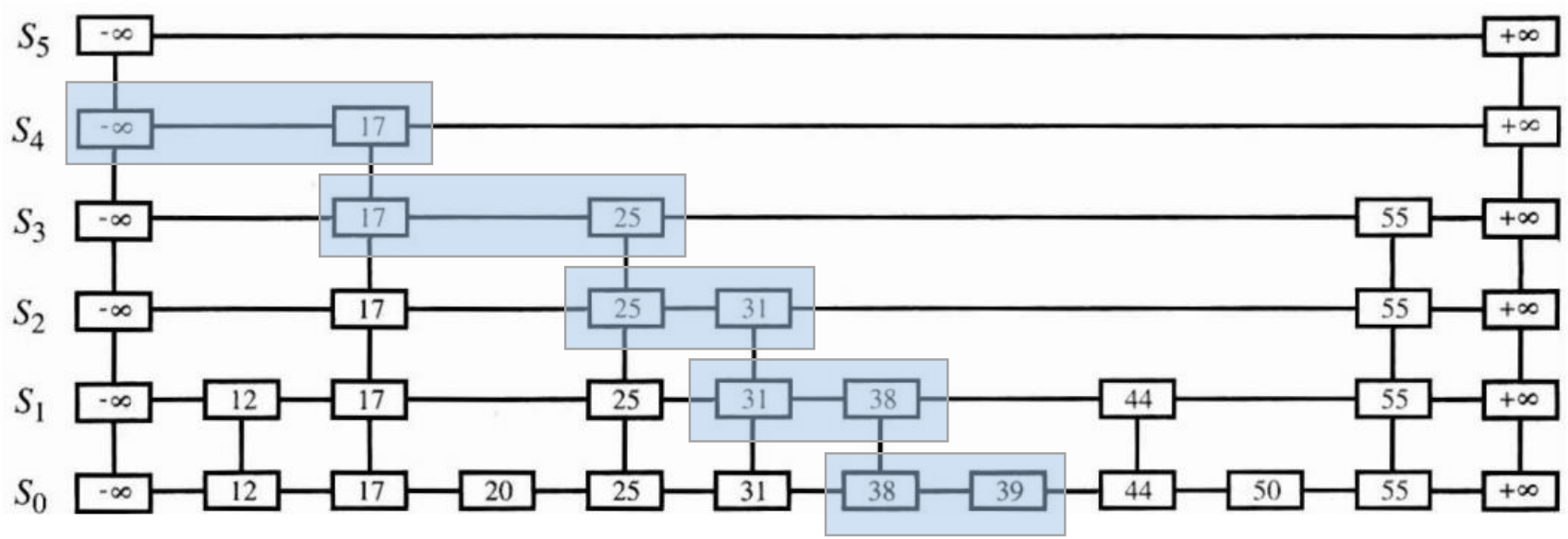
Average Runtime for Search

Average Runtime = 2×log(n)



Finding the average height : avg(h)

Average cost = $f_1 x C_1 + f_2 x C_2 + \ldots \ldots \ldots + f_n x C_n$



Reference

1. https://www.geeksforgeeks.org/skip-list-set-2-insertion/
2. https://www.geeksforgeeks.org/skip-list-set-3-searching-deletion/
3. http://ticki.github.io/blog/skip-lists-done-right/
4. ftp://ftp.cs.umd.edu/pub/skipLists/skiplists.pdf
5. http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/Map/skip-list-perf.html (Runtime Analysis)