Randomized Linked Lists

Skip Lists

The Skip List Motivation

Data structures are often needed to provide organization for large sets of data.

Traditional approaches offer a tradeoff in performance between the insertion, deletion and the search operations:

- For contiguous storage (e.g., a sorted array list):
  - Worst / average search costs $O(\log N)$, where N is number of data elements
  - Insert / delete costs $O(N)$
- For linear linked storage (e.g., a linked list):
  - Worst / average search costs $O(N)$
  - Insert / delete costs $O(1)$
- For balanced binary trees (e.g., an AVL tree):
  - Worst / average search costs $O(\log N)$
  - Insert / delete costs $O(\log N)$

Unfortunately, balanced binary tree implementations are quite complicated.
We'd like similar performance with less complexity…

The Skip List Concept

Linear linked structures are relatively simple to implement and well-understood.

We can improve search costs by including some additional pointers to selected nodes to allow "skipping" over nodes that can safely be ignored, called, Skip lists.

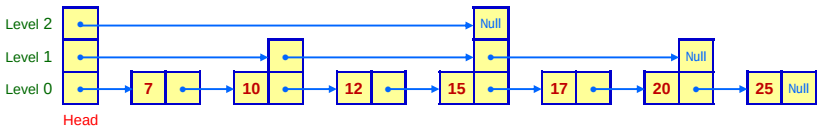
"Skip lists: a probabilistic alternative to balanced trees", CACM, W. Pugh, 1990.

Example:

Consider the following ordered list:

7, **10**, **12**, **15**, **17**, **20**, **25**

Stored as the shown Skip List, search for the values: **17** and **32**



How would the extra forward pointers be used?

How many comparisons would be required?

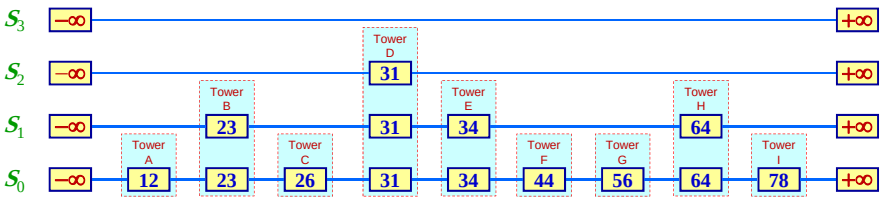
The Skip List Abstract Definition

A skip list for a set S of distinct (key, value) entries is a series of parallel linked lists, $S_0, S_1, \ldots, S_h$ such that:

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains all the keys of S **sorted** in non-decreasing order
- Each list is a subsequence of the previous one, i.e.,

$$S_0 \supseteq S_1 \supseteq \ldots \supseteq S_h$$

- List S_h contains only the two special keys.



The Skip List Operations

Remember that a skip list is an ordered linked list, where the order is defined by the key field of its entries, a current position is maintained. A node positional order is defined by its key, so:

if $key_i < key_j$ ➔ $position_i < position_j$, for all $i < j$

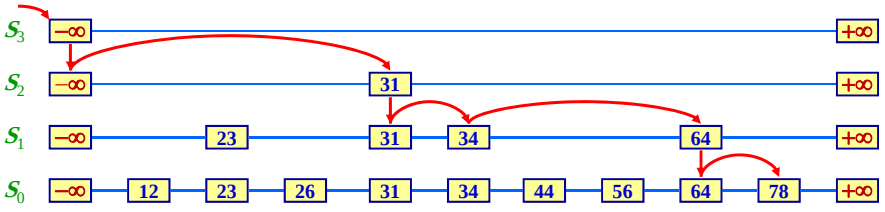
Operations:

constructor()	get()	clear()
first()	getPosition()	isElement()
last()	skipSearch(key)	isEmpty()
next()	skipAdd(key, value)	height()
prior()	skipRemove()	size()

The Skip List Search Algorithm

- To search for an entry(k, v) with key k in a skip list, proceed as follows:
  - Start at the first position of the top list
  - At any current position p, compare k with the key y, of the next element in the list:
    - if $k = y$, return the entry in the element next to p.
    - if $k > y$, scan forward in the same list, (i.e. next element becomes current).
    - if $k < y$, drop down to the next list, (i.e. corresponding element becomes current).
  - Repeat the previous step until dropping down past the bottom list, then return null.

Example: Search for key 78



The Skip List
Randomized Algorithm

- A randomized algorithm performs coin tosses, (i.e., uses random bits, 0 or 1) to control its execution. It contains statements of the type:

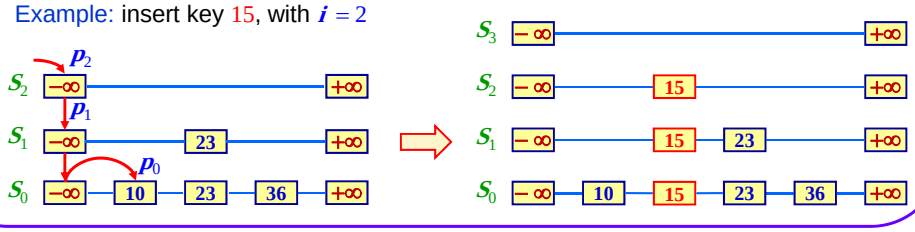
```
b ← random()
if b = 0
  do A …
else { b = 1}
  do B …
```

- Its running time depends on the outcomes of the coin tosses.
- The expected running time of a randomized algorithm is analyzed assuming:
  - The coins are unbiased, and
  - The coin tosses are independent
- The worst-case running time of a randomized algorithm is often large but has very low probability (e.g., it occurs when all the coin tosses give “heads”)
- A randomized algorithm is used to insert items into a skip list.

The Skip List
Insertion Algorithm

- To insert an entry(k, v) into a skip list, use the following randomized algorithm:
  - Repeatedly, toss a coin until you get tails; let i be the number of consecutive times the coin came up heads.
  - If $i \geq h$, add to the skip list new lists: S_{h+1}, … , S_{i+1}, each containing only the two special keys.
  - Search for k in the skip list to find the positions: p_0, p_1 , … , p_i of the items with the largest key less than k in each respective list: S_0, S_1, … , S_i
  - For $j \leftarrow 0$, … , i, insert entry(k, v) into list S_j after position p_j

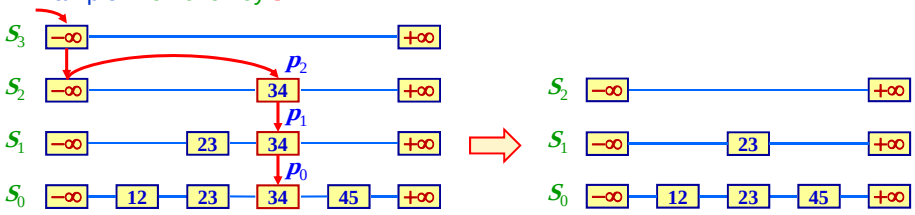
Example: insert key 15, with $i = 2$



The Skip List
Deletion Algorithm

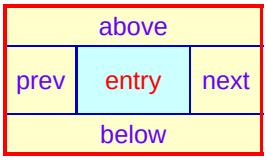
- To remove an entry(k, v) from a skip list, proceed as follows:
  - Search for key, k in the skip list and find the positions: p_0, p_1 , … , p_i of the items with key k, where position p_j is in list S_j
  - Remove the entries at positions p_0, p_1 , … , p_i from the lists S_0, S_1, … , S_i
  - Remove all but one list containing only the two special keys.

Example: Remove key 34



Skip List Implementation
Using Doubly Linked List Structure

- Use a set of Doubly Linked Lists structure.
- Each node in the lists is an instance of a QNode class that contains the following private fields:
  - entry — where the node information (key, value) are stored
  - next — which is a forward link pointer
  - prev — which is a backward link pointer
  - above — which is an up link pointer
  - below — which is a down link pointer



- Use four external pointers into the skip list:
  - start — points to the first node of the level h list
  - head — points to the first node of the level 0 list
  - tail — points to the last node of the level 0 list
  - current — points to the current node of the level 0 list

Skip List Implementation
Using Doubly Linked List Structure

- Use two integer variables:
  - height stores the number of lists currently in the structure -1.
  - count stores the number of nodes currently in the level 0 list.
- Also, define the three constants:
  - MAX_H Maximum allowable height of the skip list.
  - PLUS_INF Special key: For highest possible key (+ ∞).
  - MINUS_INF Special key: For lowest possible key (- ∞).

  and modify the key comparator to handle them properly.
- An empty list is initialized by having two empty lists at levels 0 & 1:
  - head = $P(-\infty)_0$, tail = $P(+\infty)_0$
  - start = $P(-\infty)_1$, current = head
  - height = 1, count = 0.

  ← The Constructor

Skip List Implementation
The Entry Class Structure

- Used as a generic class with variable data type parameters, K, as the type of the key, and V, as the type of the value associated with it, to be stored in each entry.
- The location field is used to make the entry aware of its location in the list.
- The class has a constructor to initialize its data fields
- Functions for accessing the class fields are also members of the Entry<K , V> class

```
public class Entry <K,V>
  private K key;              // K is a key object type
  private V value;            // V is a value object type
  private int location = 0;
public:
  Entry (     K initKey, V initValue);

  void setKey(K newKey);
  void setValue(V newValue);
  void setLocation(int newLocation);

  K  getKey();
  V  getValue();
  int getLocation();
```

Skip List Implementation
The QNode Class Structure

- Used as a generic class with a variable data type parameter, E, as the type of information to be stored in each node
- The link fields are pointers to a QNode<E> object type
- The class has a constructor to initialize its data fields
- Functions for accessing the class fields are also members of the QNode<E> class

```
public class QNode <E>
  private E entry;                // E is class Entry<K,V>
  private QNode<E> next, prev;
  private QNode<E> above, below;
public:
  QNode(E initEntry,
        QNode<E> initNext,
        QNode<E> initPrev,
        QNode<E> initAbove,
        QNode<E> initBelow);
  void setEntry(E newEntry);
  void setNext(QNode<E> newNext);
  void setPrev(QNode<E> newPrev);
  void setAbove(QNode<E> newAbove);
  void setBelow(QNode<E> newBelow);
  E getEntry();
  QNode<E> getNext();
  QNode<E> getPrev();
  QNode<E> getAbove();
  QNode<E> getBelow();
```

Skip List Implementation
The SkipQLinkList Class Structure

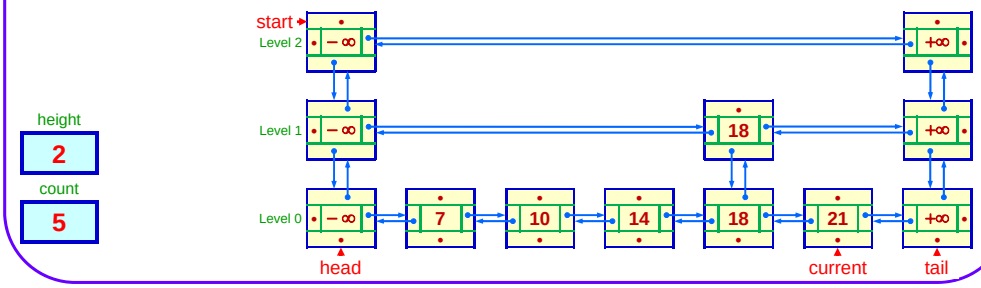
- Used as a generic class with the variable data type param's, K and V; where an entry type must have a key field of type K and a value field of type V as the information stored in each entry.
- The head, tail, start and current fields are external pointers to the list structure.
- The class has a constructor to initialize its data fields.
- Some functions that implement the Skip List operations are also members of the this class

```
public class SkipQLinkList<K,V>
  private QNode<Entry> start;
  private QNode<Entry> head;
  private QNode<Entry> tail;
  private QNode<Entry> current;
  private int height, count;
  public SkipQLinkList();
  public QNode<E> first(),   last();
  public QNode<E> next(),  prior();
  public QNode<E> skipSearch(K key);
  public QNode<E> skipAdd(K key, V value);
  public QNode<E> skipRemove();
  public boolean isElement();
  public boolean isEmpty();
  public void clear();
  public int height();
  public int size();
```

Mapping Operation: first() ... *O*(1)

- **Postcondition**:
  - Current is put at the first node of the level 0 list, this has the entry with minimum key.
  - If the list was empty, then there is no change.
- **Code**:

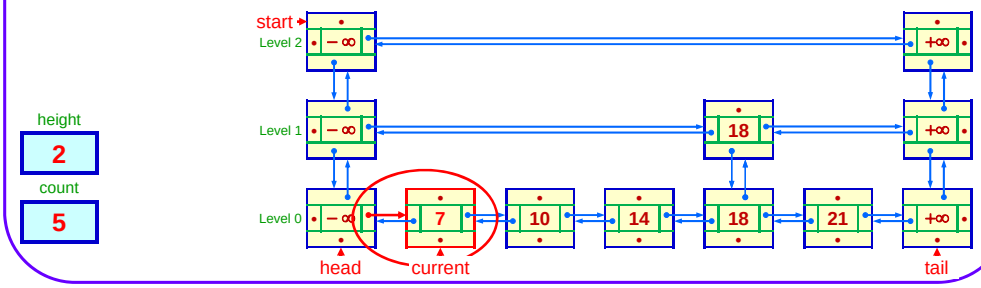
```
if (count > 0)
    current = head.getNext();
```



Mapping Operation: first() ... *O*(1)

- **Postcondition**:
  - Current is put at the first node of the level 0 list, this has the entry with minimum key.
  - If the list was empty, then there is no change.
- **Code**:

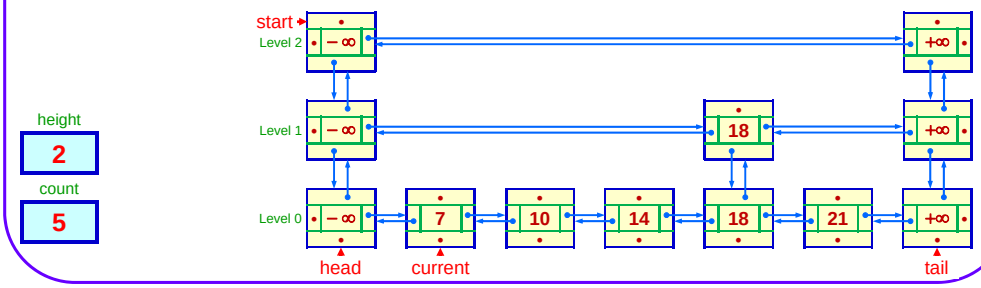
```
if (count > 0)
    current = head.getNext();
```



Mapping Operation: last() ... *O*(1)

- **Postcondition**:
  - Current is put at the last node of the level 0 list, this has the entry with maximum key.
  - If the list was empty, then there is no change.
- **Code**:

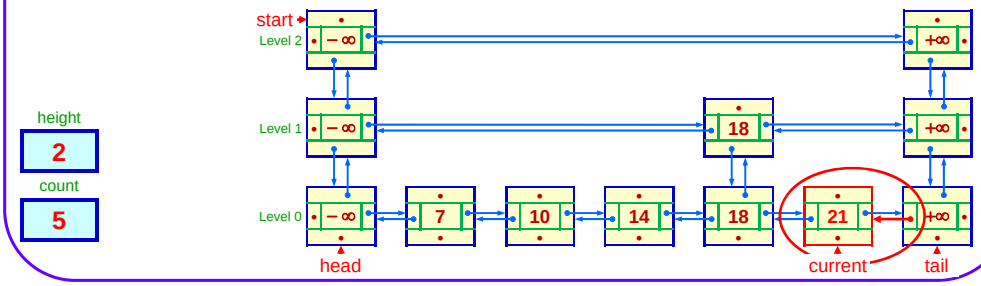
```
if (count > 0)
    current = tail.getPrev();
```



Mapping Operation: last() ... *O*(1)

- **Postcondition**:
  - Current is put at the last node of the level 0 list, this has the entry with maximum key.
  - If the list was empty, then there is no change.
- **Code**:

```
if (count > 0)
    current = tail.getPrev();
```

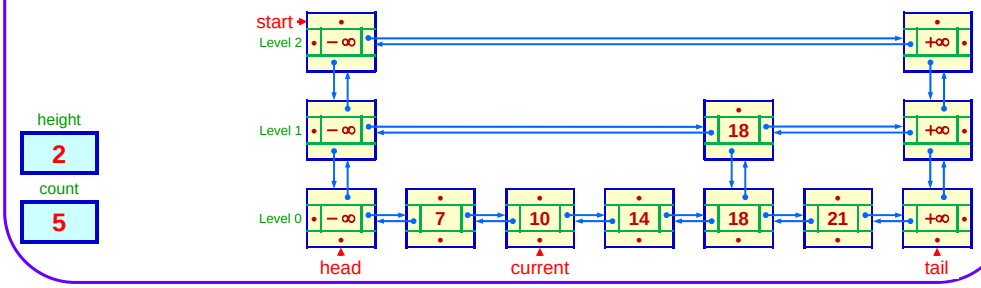



Mapping Operation: next() ... *O*(1)

- **Precondition**:
  - **isElement()** returns true.
- **Postcondition**:
  - If the current node is not the last node at level 0 of the skip list, then current is put at its successor node. Otherwise, current is not valid.

- **Code**:

```
if (isElement())
    current = current.getNext();
```

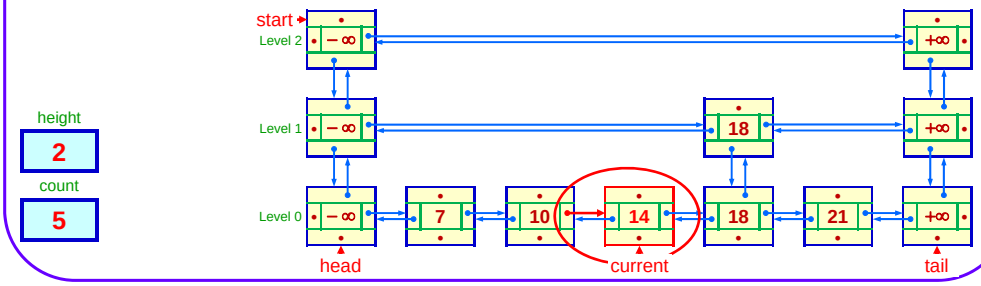


Mapping Operation: next() ... *O*(1)

- **Precondition**:
  - **isElement()** returns true.
- **Postcondition**:
  - If the current node is not the last node at level 0 of the skip list, then current is put at its successor node. Otherwise, current is not valid.

- **Code**:

```
if (isElement())
    current = current.getNext();
```

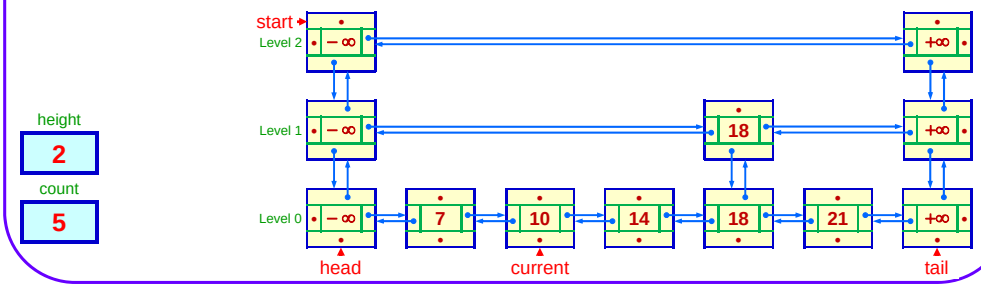



Mapping Operation: prior() ... *O*(1)

- **Precondition**:
  - **isElement()** returns true.
- **Postcondition**:
  - If the current node is not the first node at level 0 of the skip list, then current is put at its predecessor node. Otherwise, current is not valid.

- **Code**:

```
if (isElement())
    current = current.getPrev();
```

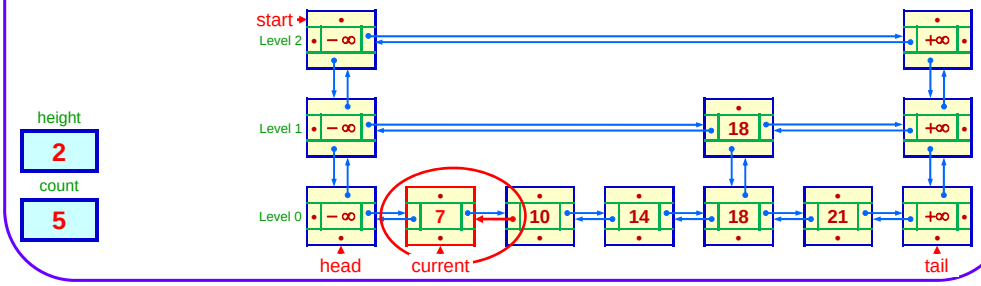


Mapping Operation: prior() ... *O*(1)

- **Precondition**:
  - **isElement()** returns true.
- **Postcondition**:
  - If the current node is not the first node at level 0 of the skip list, then current is put at its predecessor node. Otherwise, current is not valid.

- **Code**:

```
if (isElement())
    current = current.getPrev();
```

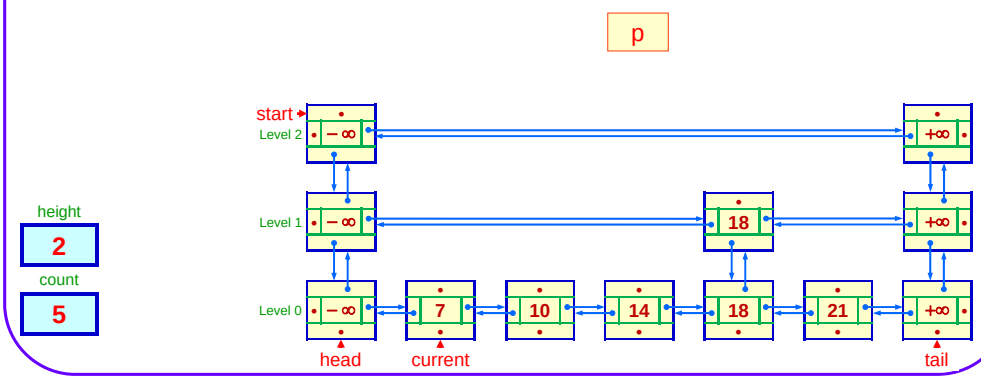
Mapping Operation: skipSearch(key) ... O("height")

- **Postcondition**:
  - The skip list has been searched for the given key. If the key was found, then current is at the node in level 0 list, with the entry that has the oldest occurrence of the key.
  - If it was not found then current is at the nearest node with maximum key less than the given key.
  - If there was no such a node, then current is not valid.
- **Code**:

```
QNode p;

p = start;
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}

current = p;
```

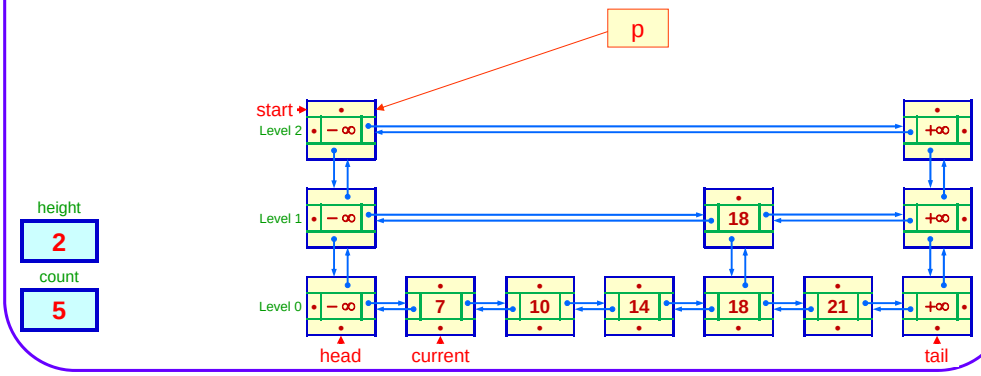
Mapping Operation: skipSearch(key) Example: search(21);

QNode p;
p = start;



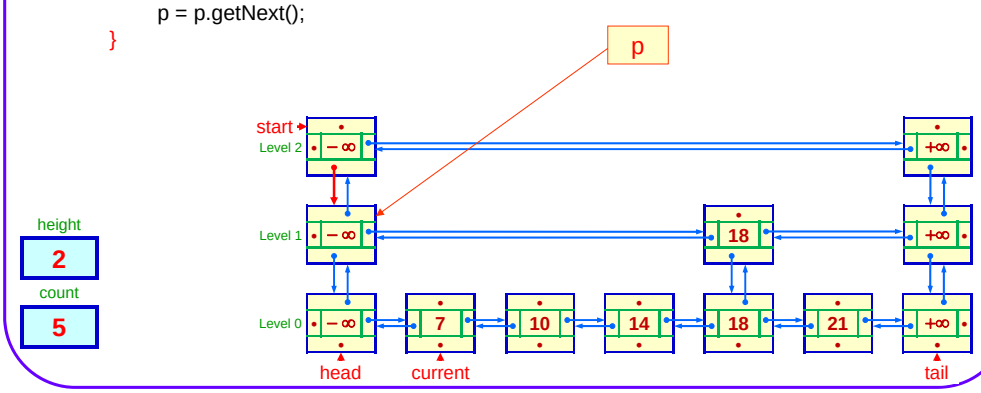
Mapping Operation: skipSearch(key)
Example: search(21);

```
QNode p;
p = start;
```



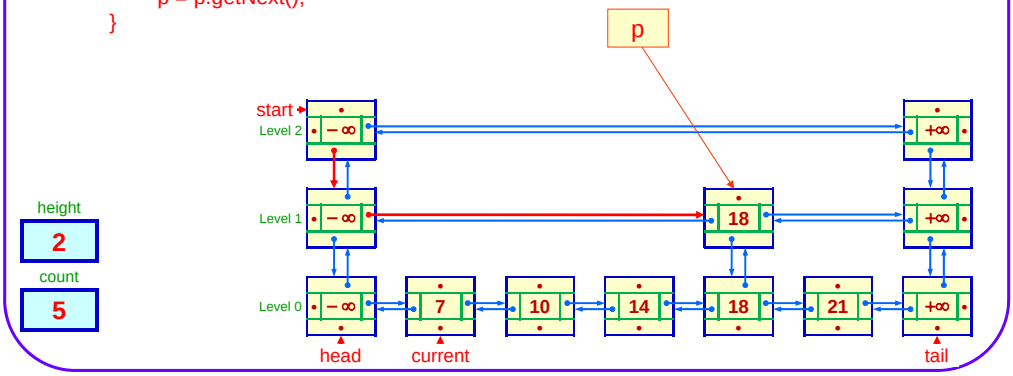
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```



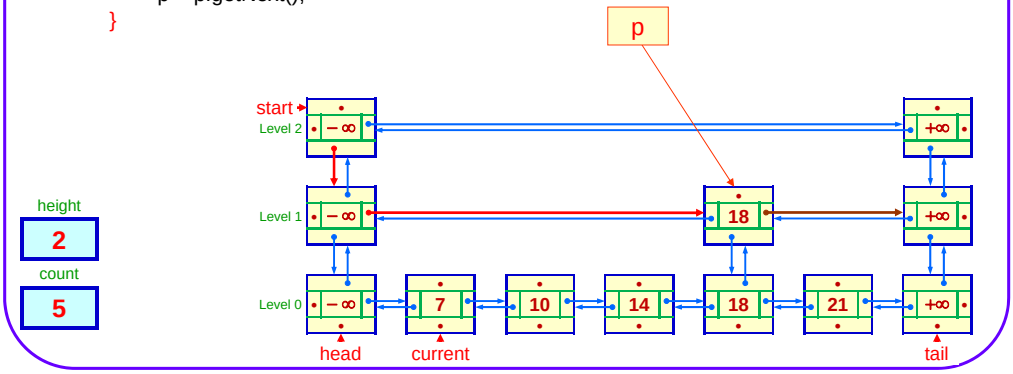
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```



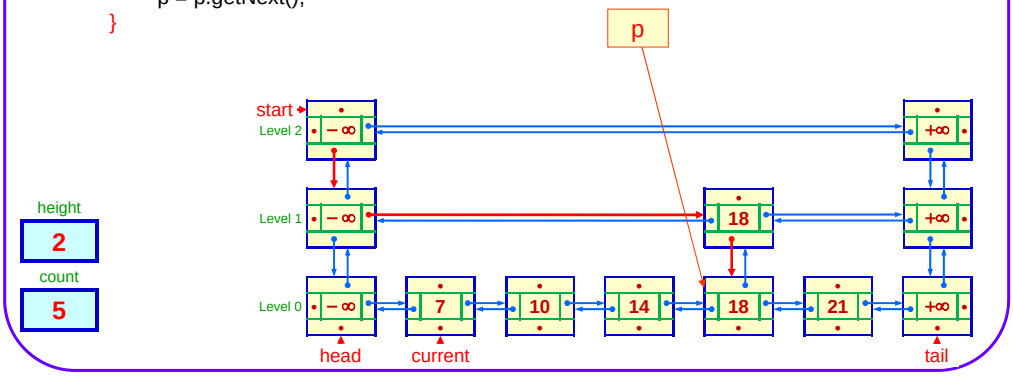
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```



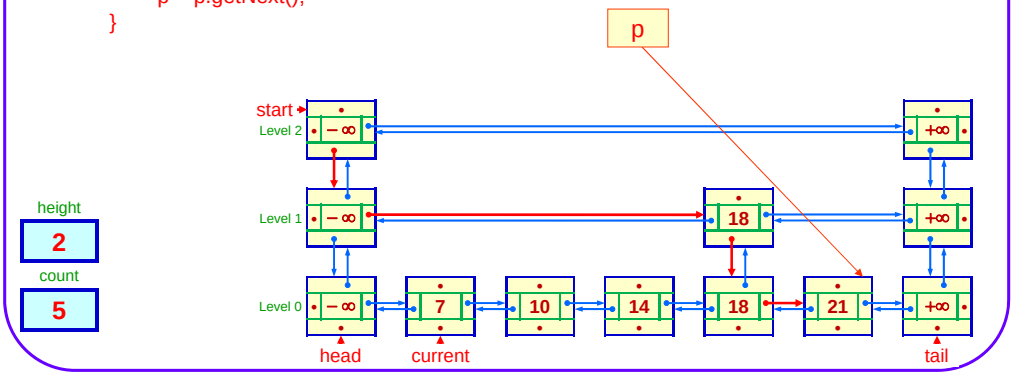
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```



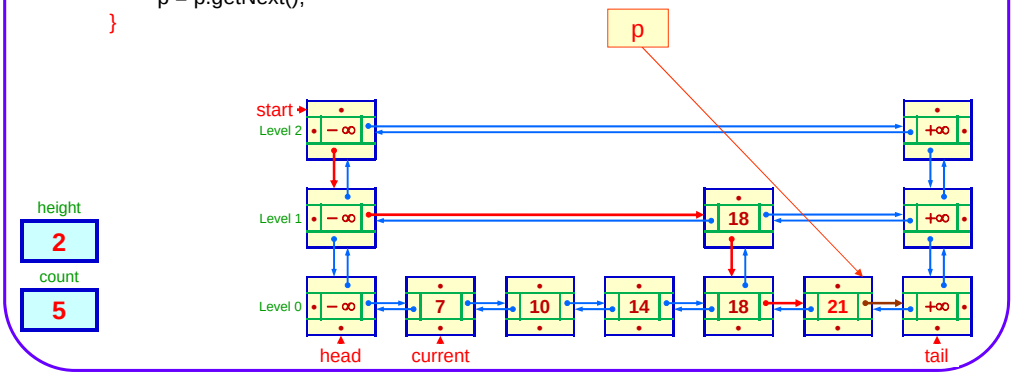
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```



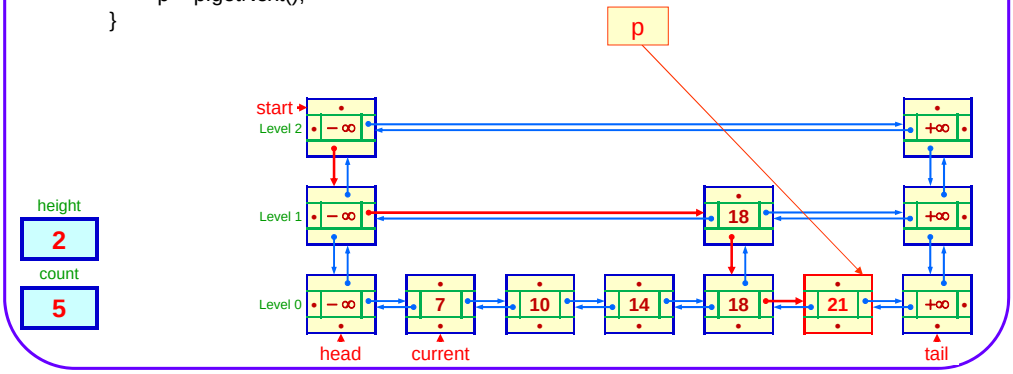
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```



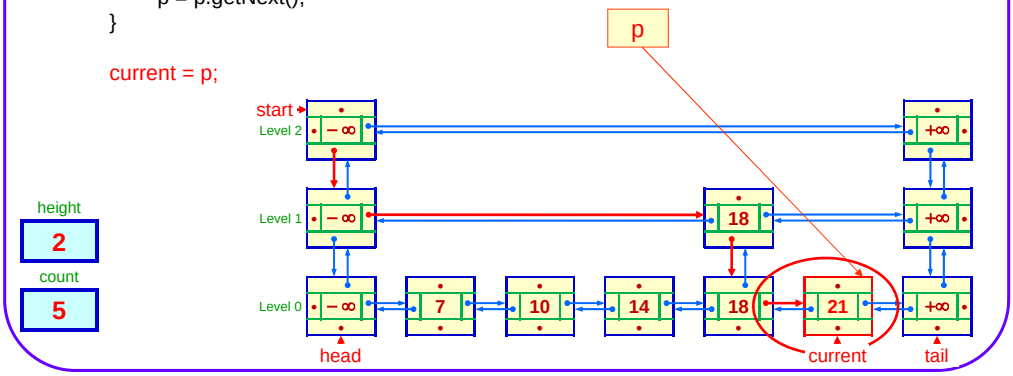
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}
```

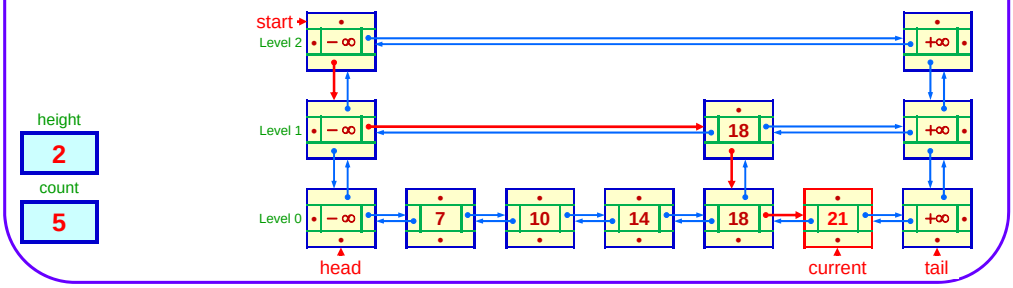
Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}

current = p;
```



Mapping Operation: skipSearch(key)
Example: search(21);

```
while (p.getBelow() != null) {
   p = p.getBelow();
   while (key >= p.getNext().getEntry().getKey())
      p = p.getNext();
}

current = p;
```



Mapping Operation: skipAdd(key, value) ... O("height")

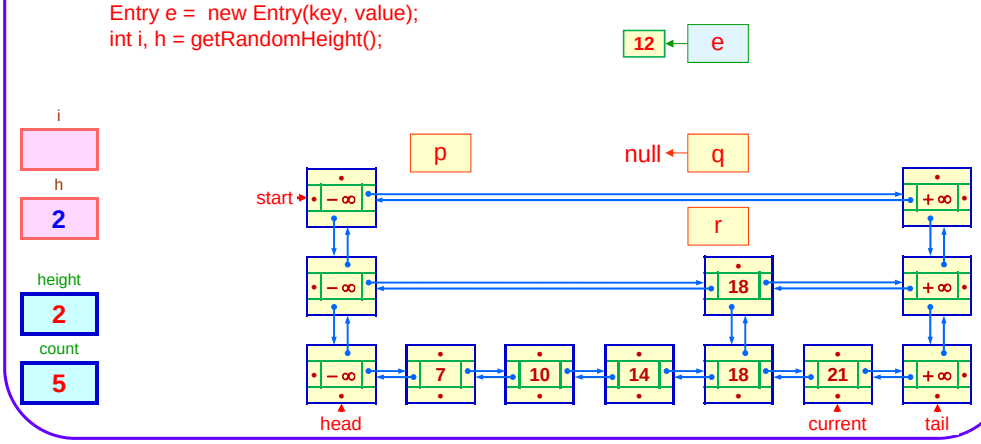
- **Postcondition**:
  - New node(s) that reference one entry of the specified key and value are created and inserted in their proper locations at all levels in the skip list, according to the new node's determined height.
  - The new node's height is given by a randomization algorithm.
  - After insertion, the list remains sorted, its height may have increased, limited by MAX_H, and its integrity is maintained.
  - If the given key was already in the list, the new node is inserted before the existing nodes of similar keys.
  - Current is put at the newly added node at level 0 list, and its entry is returned.

- **Code**:

```
QNode r, p, q = null;
Entry e =  new Entry(key, value);
int i, h = getRandomHeight();
if (h >= height)
   for (i = (h-height);  i >= 0;  --i)
      start = addOneLevel();   // add empty levels
p = start.getBelow();
for (i = height-1;  p != null;  --i) {
   while (key > p.getNext().getEntry().getKey()) )
      p = p.getNext();
   if (i <= h) {
      r = new QNode ((e, p.getNext(), p, q, null);
      p.getNext().setPrev(r);
      p.setNext(r);
      if (q != null)     q.setBelow(r);
      q = r;
   }
   p = p.getBelow();
}
count++;
current = q;
```

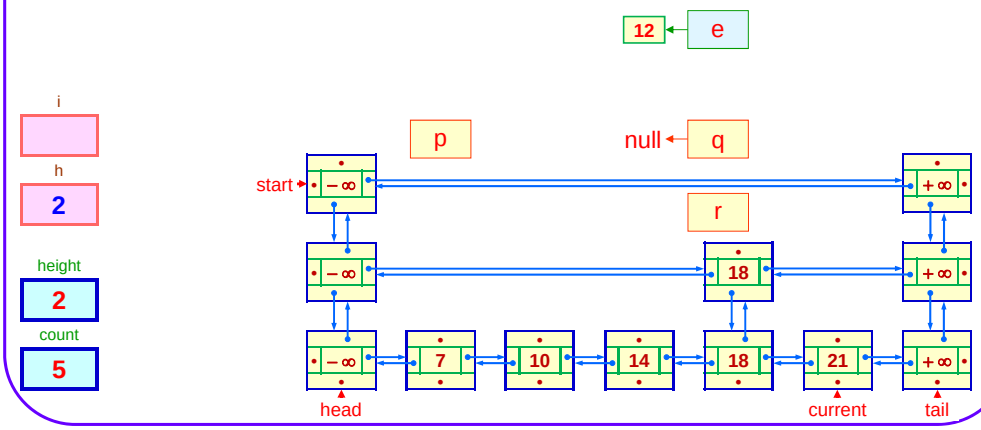
Mapping Operation: skipAdd(key, value) Example: skipAdd(12, null); at height=2

```
QNode r, p, q = null;
Entry e =  new Entry(key, value);
int i, h = getRandomHeight();
```



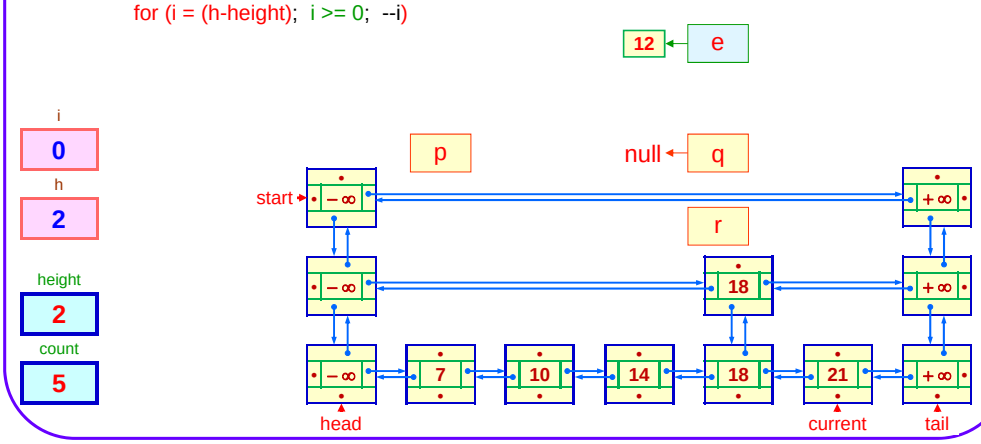
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (h >= height)
```



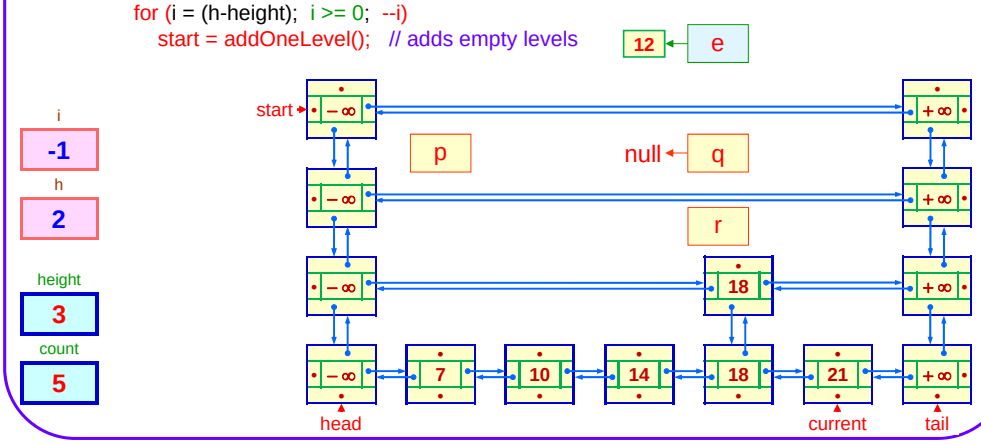
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (h >= height)
   for (i = (h-height);  i >= 0;  --i)
```



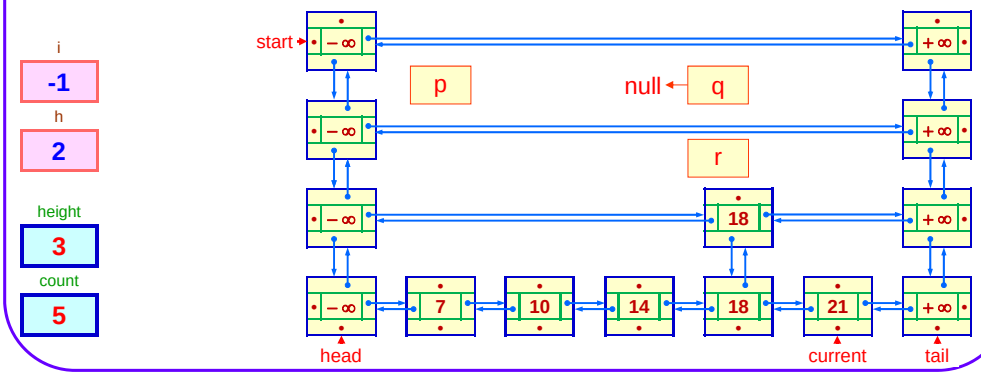
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (h >= height)
   for (i = (h-height);  i >= 0;  --i)
      start = addOneLevel();   // adds empty levels
```



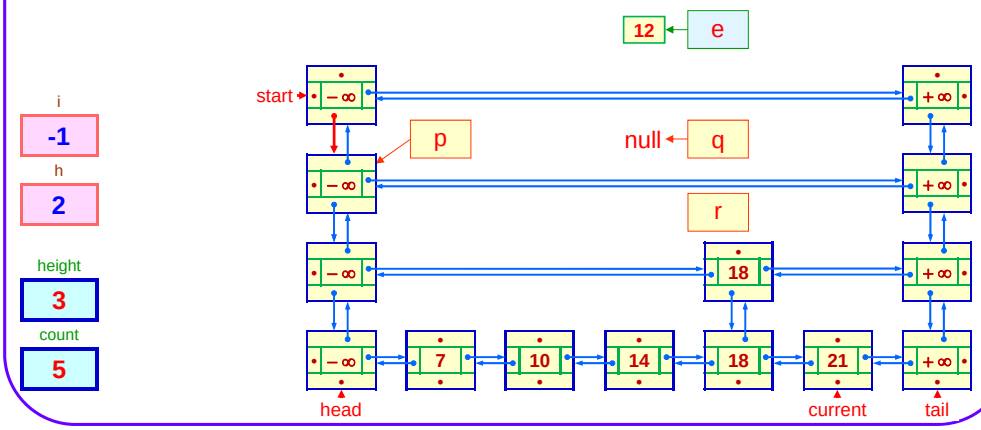
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (h >= height)
   for (i = (h-height);  i >= 0;  --i)
      start = addOneLevel();   // adds empty levels
```

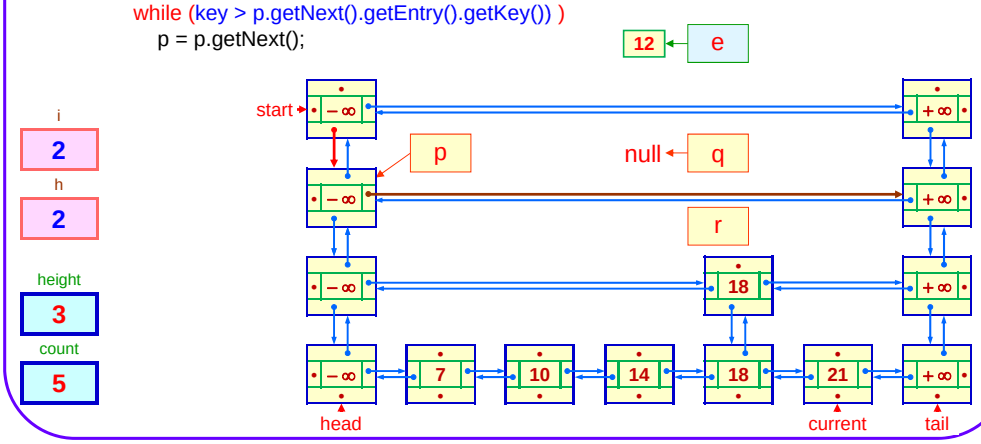
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
p = start.getBelow();
```



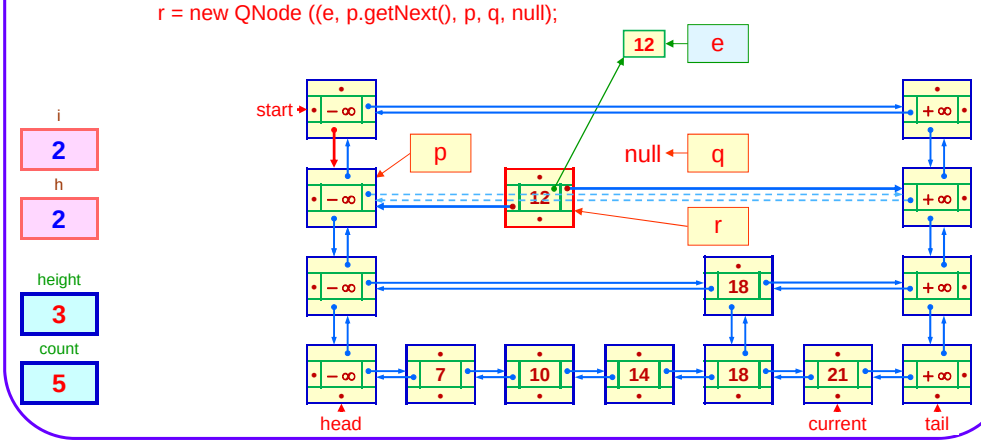
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
for (i = height-1;  p != null;  --i) {
   while (key > p.getNext().getEntry().getKey()) )
      p = p.getNext();
```



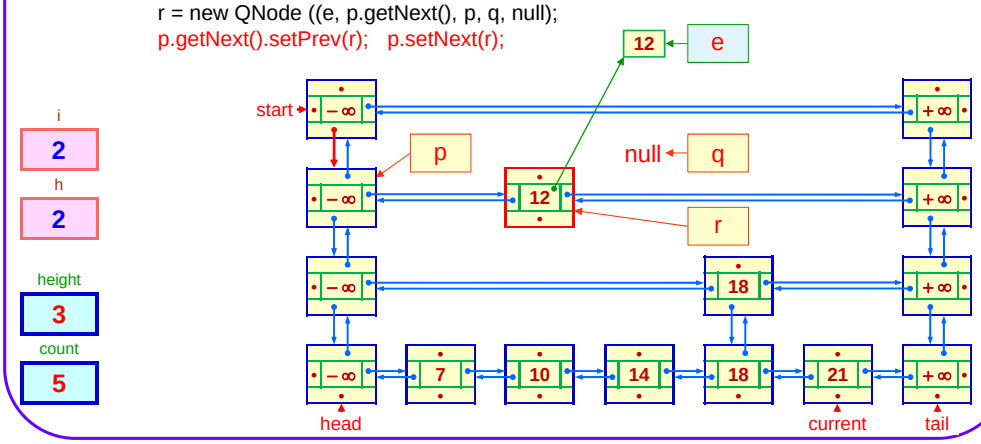
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
   r = new QNode ((e, p.getNext(), p, q, null);
```



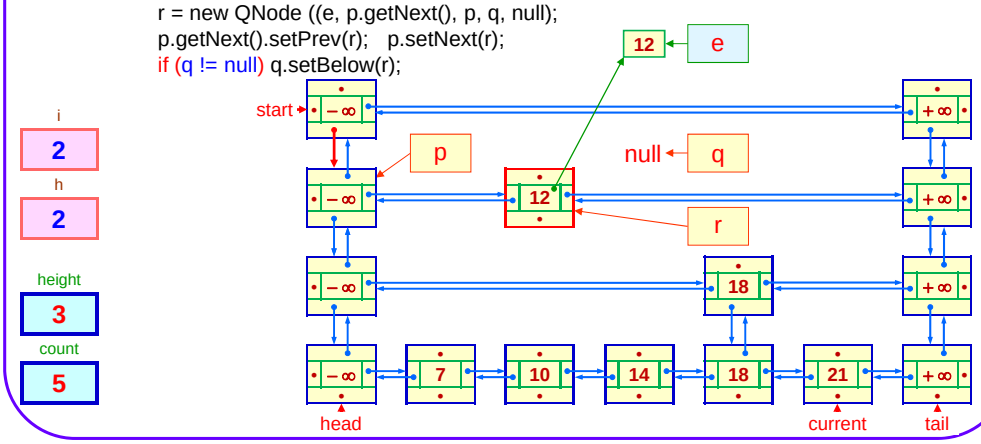
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
   r = new QNode ((e, p.getNext(), p, q, null);
   p.getNext().setPrev(r);   p.setNext(r);
```



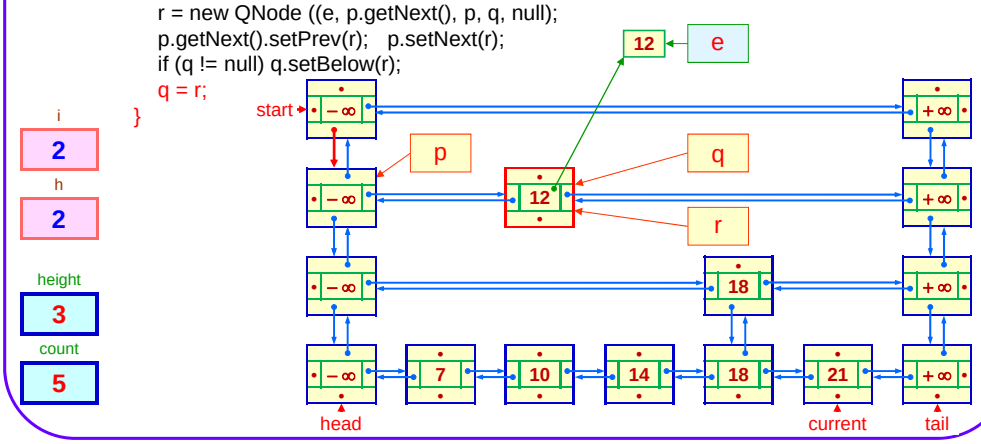
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
    r = new QNode ((e, p.getNext(), p, q, null);
    p.getNext().setPrev(r);   p.setNext(r);
    if (q != null) q.setBelow(r);
```



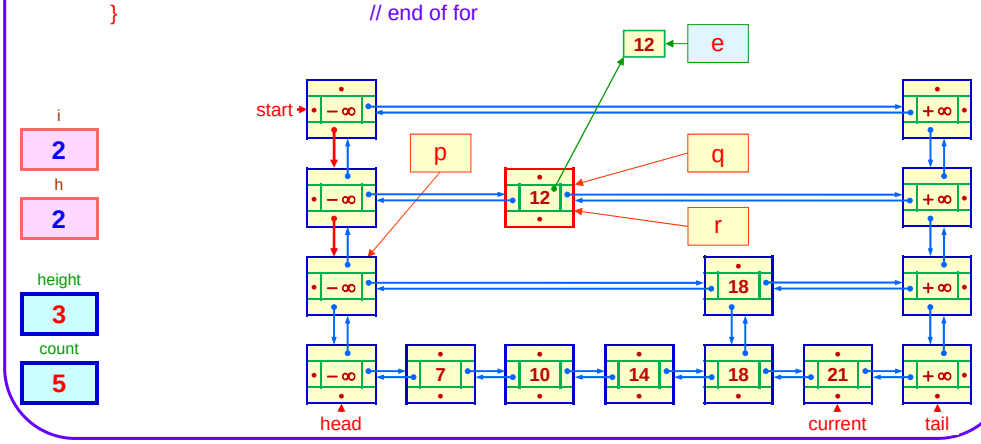
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
    r = new QNode ((e, p.getNext(), p, q, null);
    p.getNext().setPrev(r);   p.setNext(r);
    if (q != null) q.setBelow(r);
    q = r;
}
```



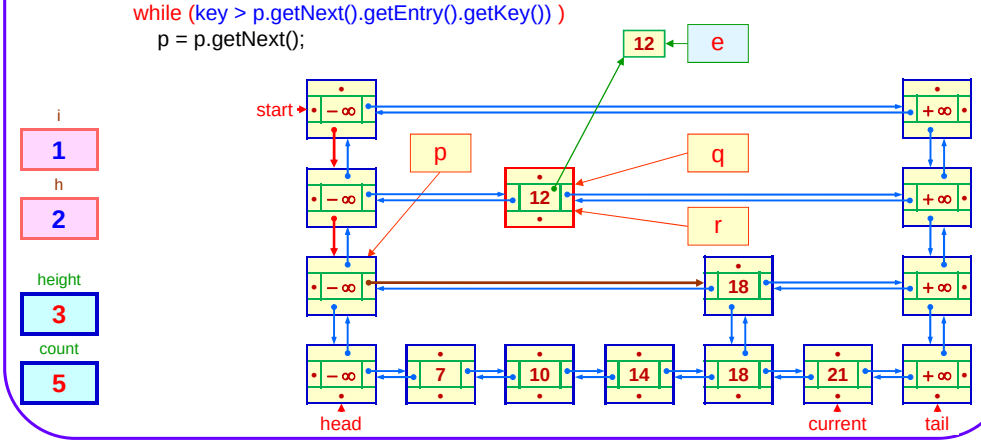
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
    p = p.getBelow();
}                          // end of for
```



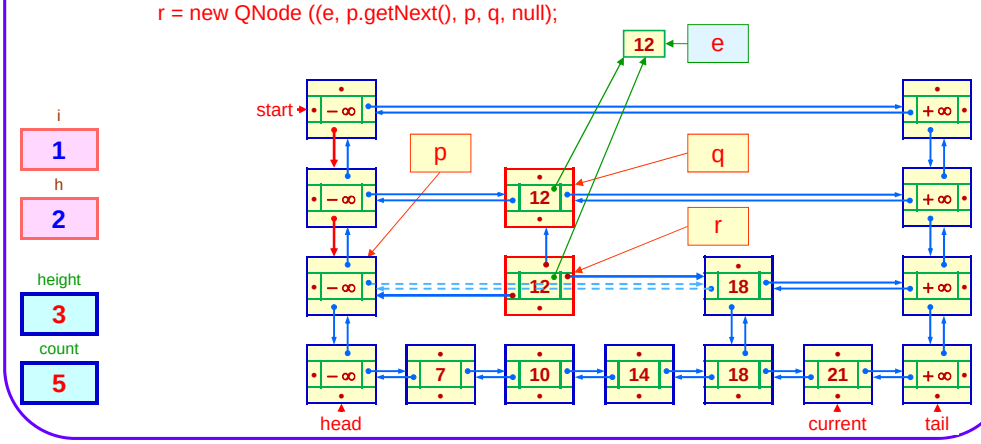
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
for (i = height-1;  p != null;  --i) {
   while (key > p.getNext().getEntry().getKey()) )
      p = p.getNext();
```



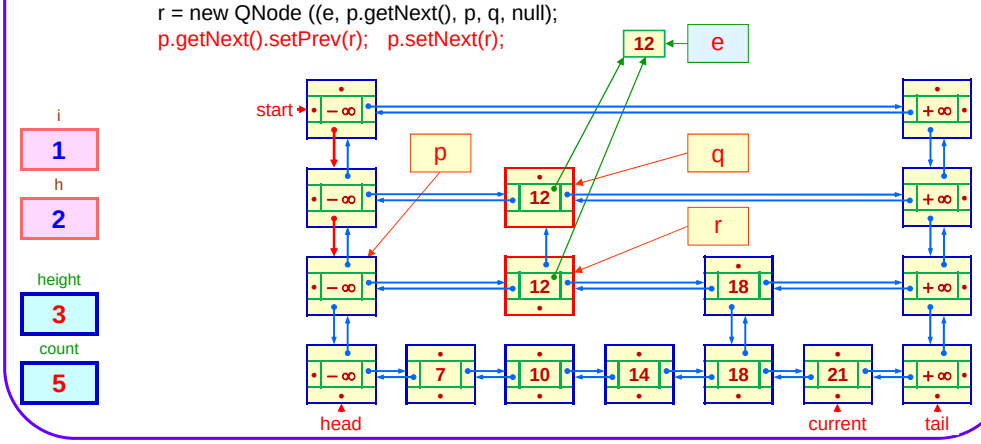
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
   r = new QNode ((e, p.getNext(), p, q, null);
```



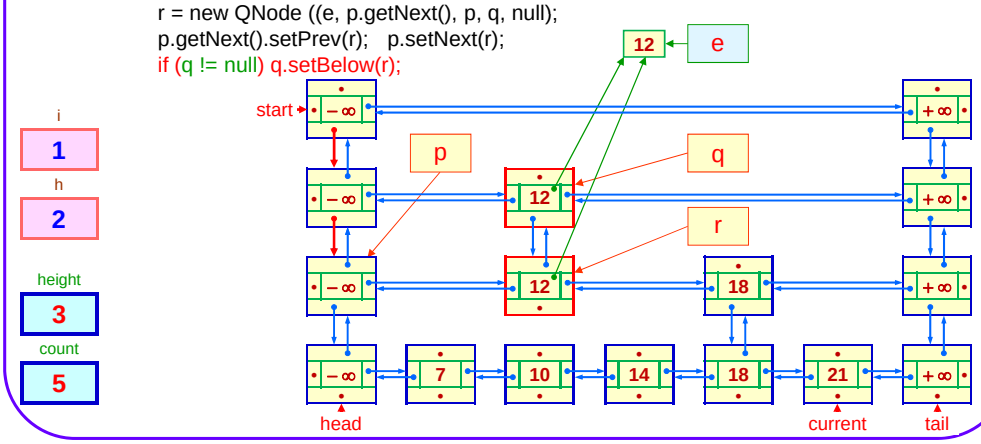
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
   r = new QNode ((e, p.getNext(), p, q, null);
   p.getNext().setPrev(r);   p.setNext(r);
```



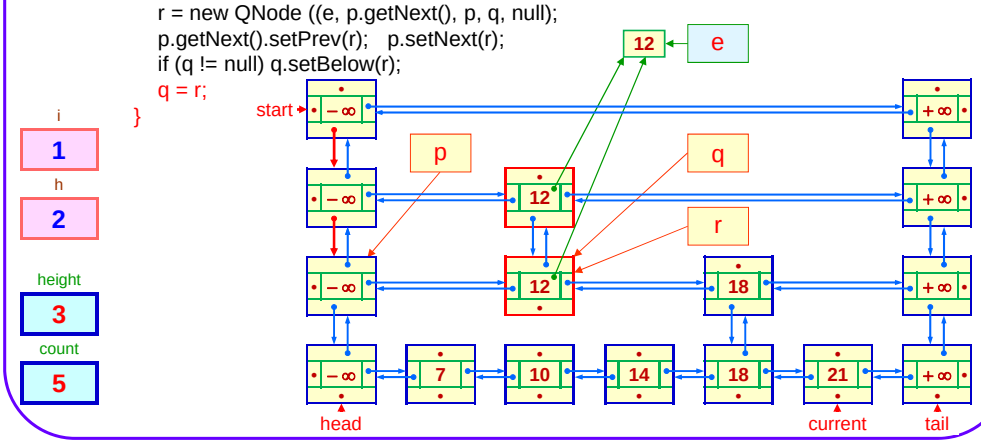
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
    r = new QNode ((e, p.getNext(), p, q, null);
    p.getNext().setPrev(r);   p.setNext(r);
    if (q != null) q.setBelow(r);
```



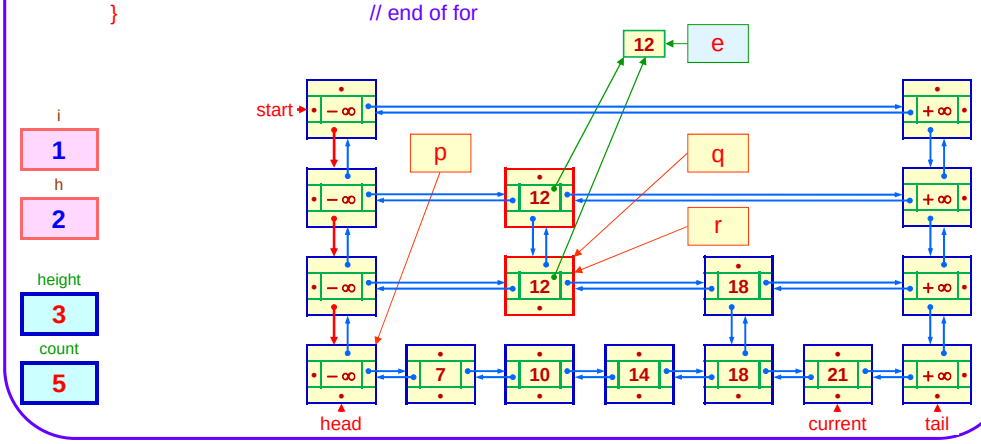
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
    r = new QNode ((e, p.getNext(), p, q, null);
    p.getNext().setPrev(r);   p.setNext(r);
    if (q != null) q.setBelow(r);
    q = r;
}
```



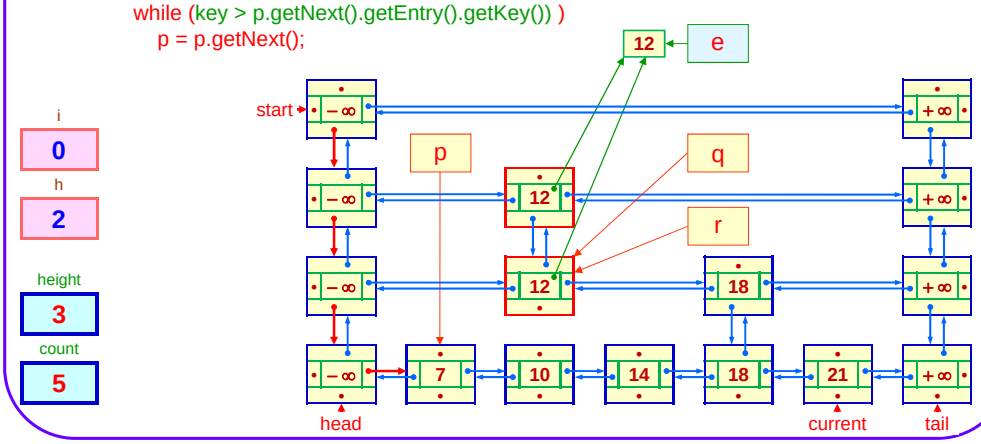
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
    p = p.getBelow();
}                          // end of for
```



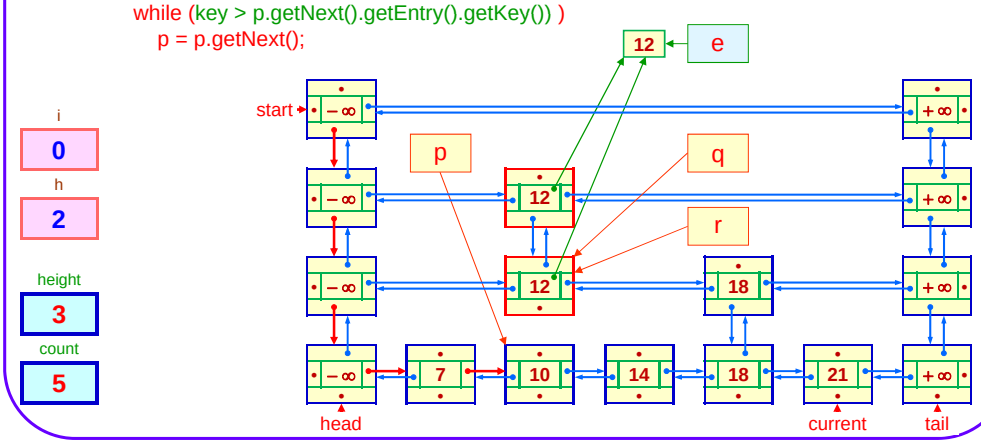
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
for (i = height-1;  p != null;  --i) {
   while (key > p.getNext().getEntry().getKey()) )
      p = p.getNext();
```



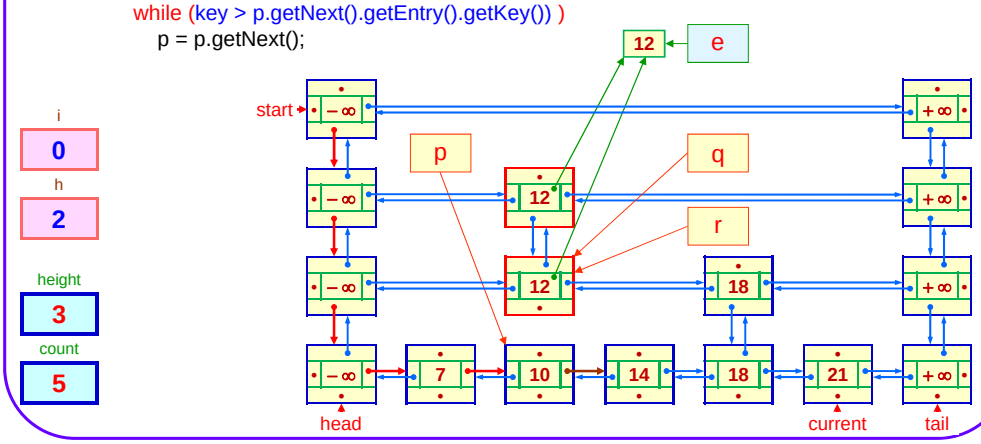
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
for (i = height-1;  p != null;  --i) {
   while (key > p.getNext().getEntry().getKey()) )
      p = p.getNext();
```



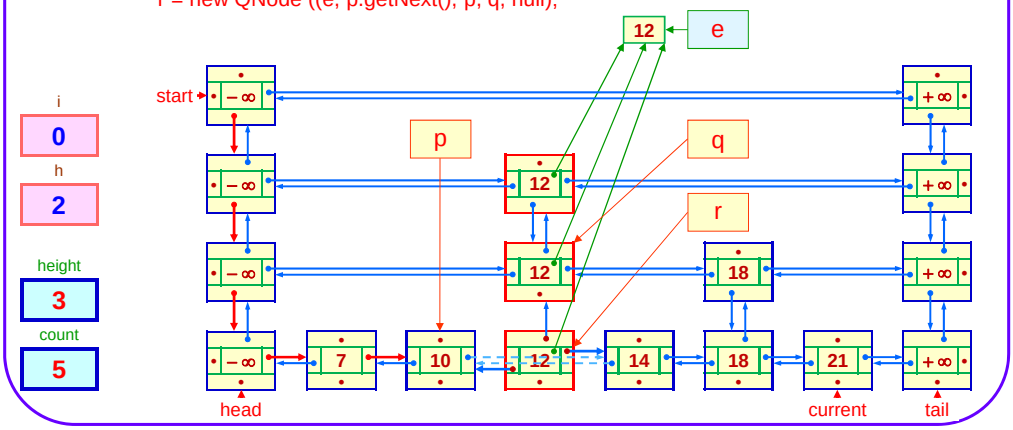
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
for (i = height-1;  p != null;  --i) {
   while (key > p.getNext().getEntry().getKey()) )
      p = p.getNext();
```

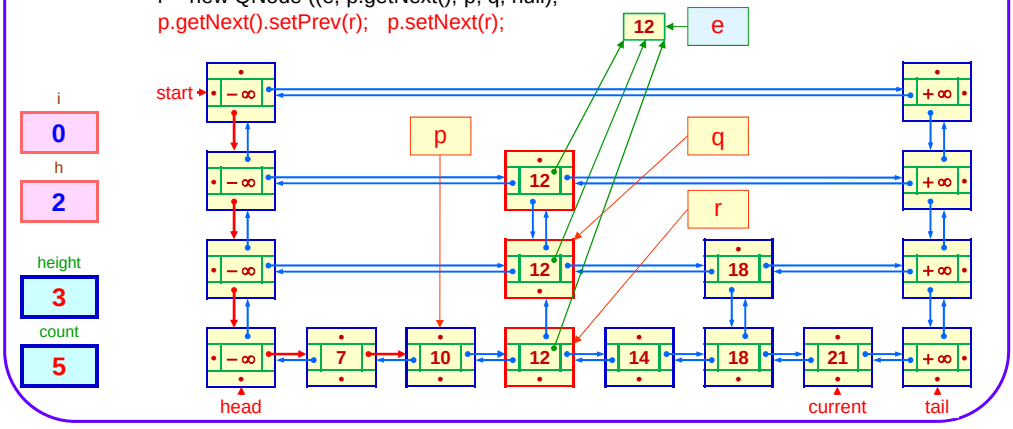
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
   r = new QNode ((e, p.getNext(), p, q, null);
```



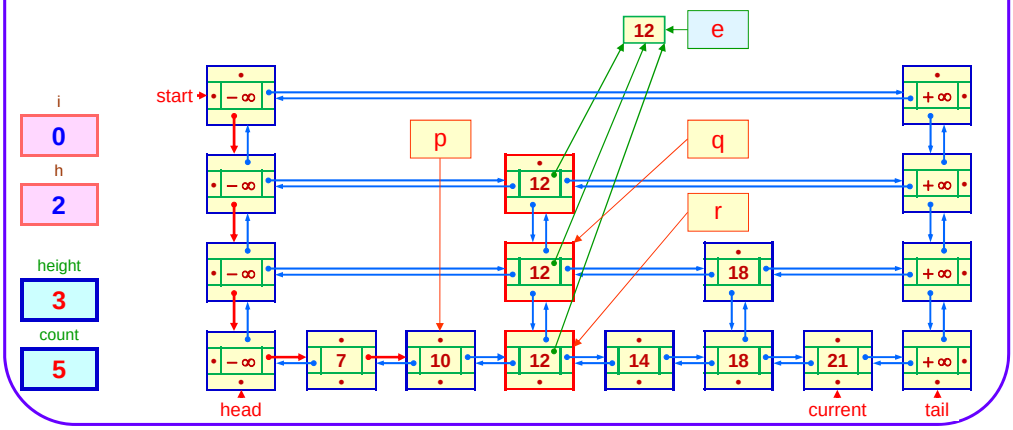
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (i <= h) {
   r = new QNode ((e, p.getNext(), p, q, null);
   p.getNext().setPrev(r);   p.setNext(r);
```



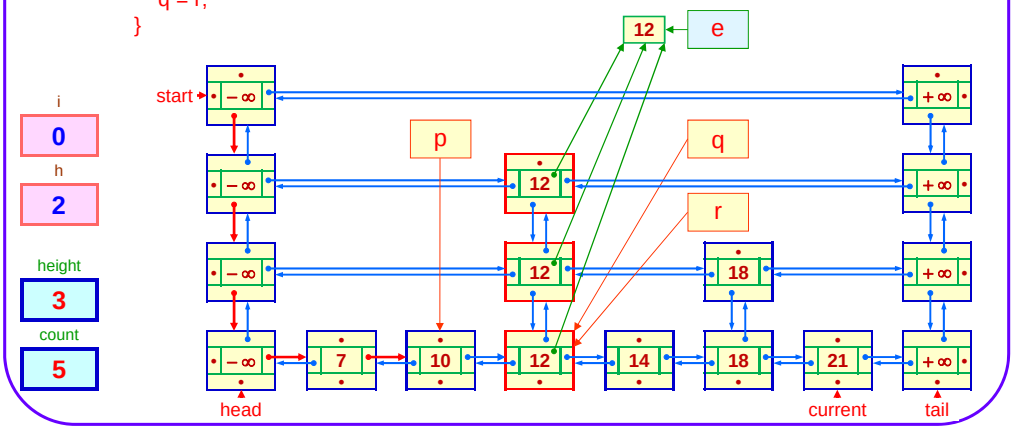
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (q != null) q.setBelow(r);
```



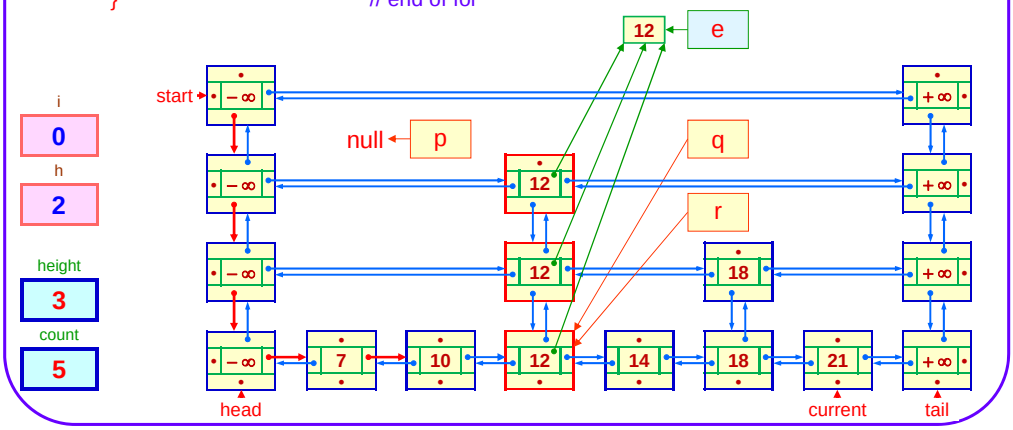
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
if (q != null) q.setBelow(r);
q = r;
}
```



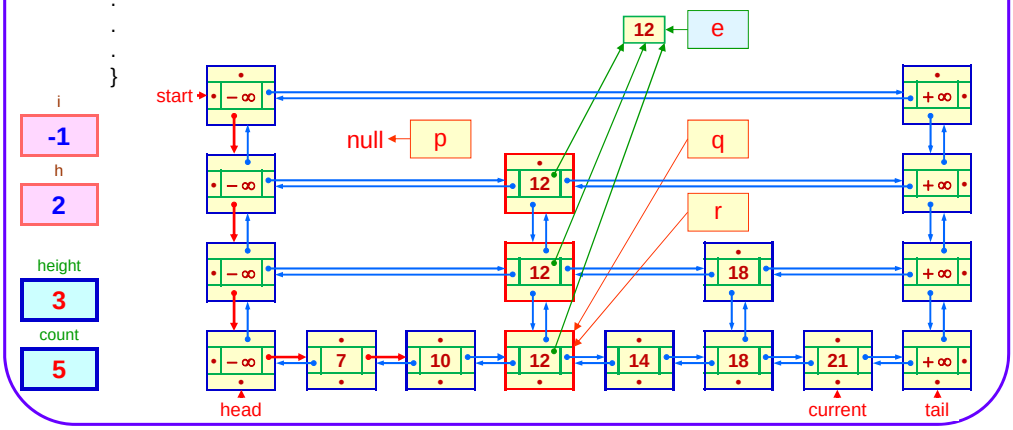
Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

```
    p = p.getBelow();
}                          // end of for
```

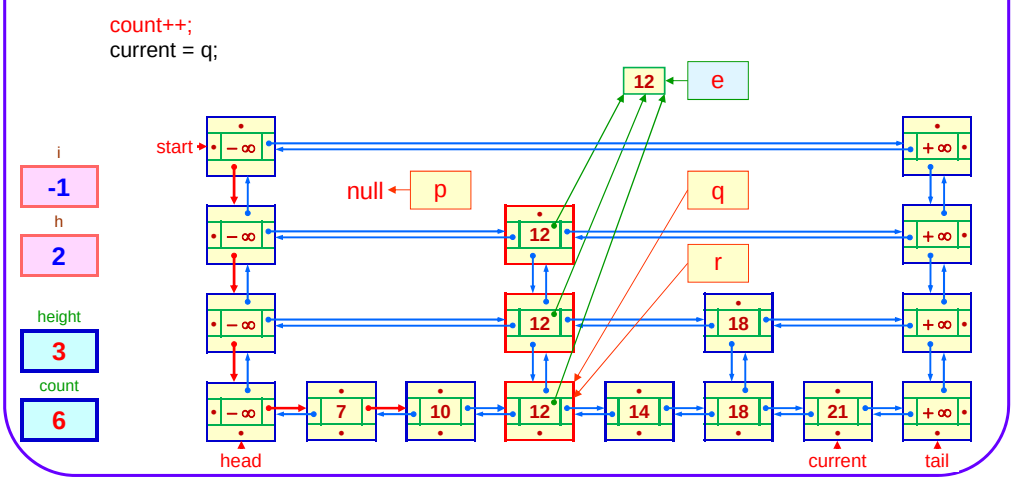


Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

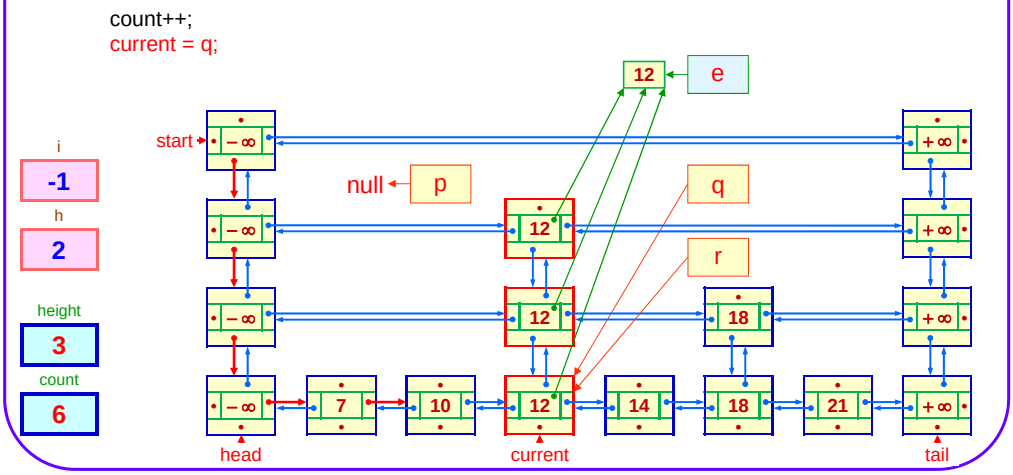
```
for (i = height-1;  p != null;  --i) {
.
.
.
}
```



Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

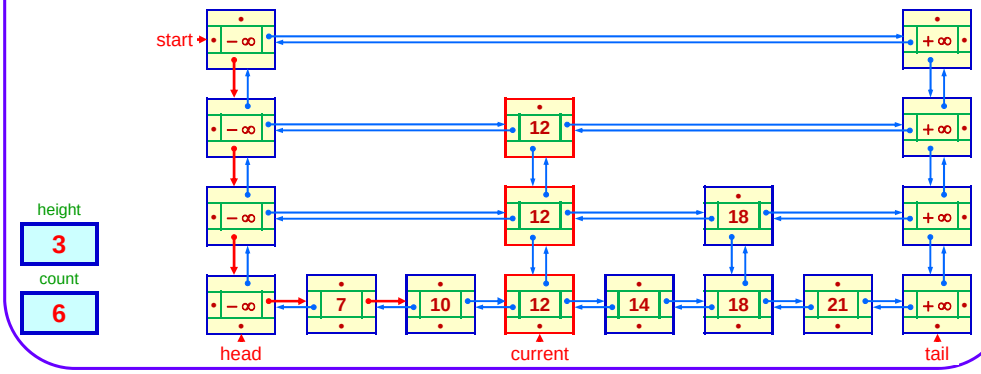


Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2



Mapping Operation: skipAdd(key, value)
Example: skipAdd(12, null); at height=2

count++;
current = q;

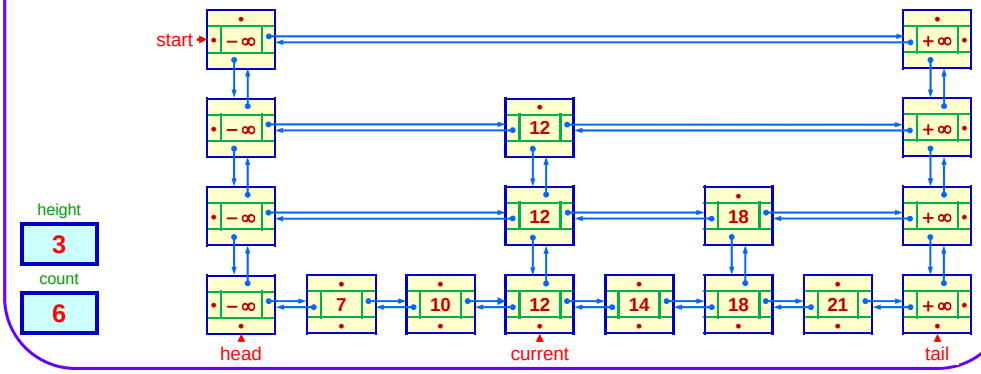


Mapping Operation: skipRemove() … *O("height")*

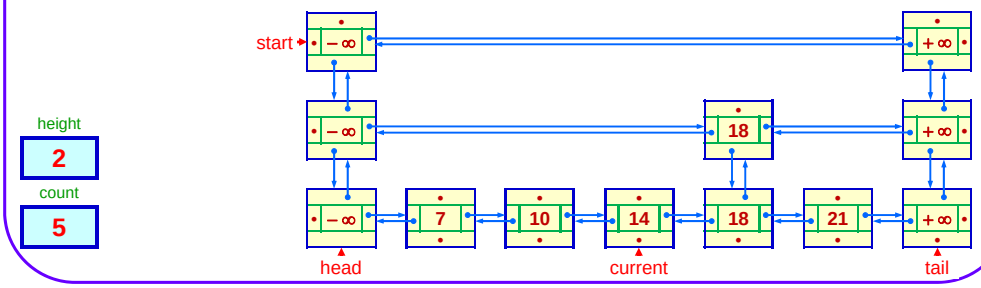
- **Precondition**:
  - **isElement()** returns true.
- **Postcondition**:
  - The current node at level 0 list is removed.
  - All nodes having the same entry as the current node and are at the same tower, are also removed from all levels of the list.
  - All empty levels of the skip list, but one are also removed.
  - Current is put at the level 0 successor of the removed node.
  - If the removed node have no successor, then current is not valid
- **Code**:

Withheld

Mapping Operation: skipRemove()
Example: skipRemove();



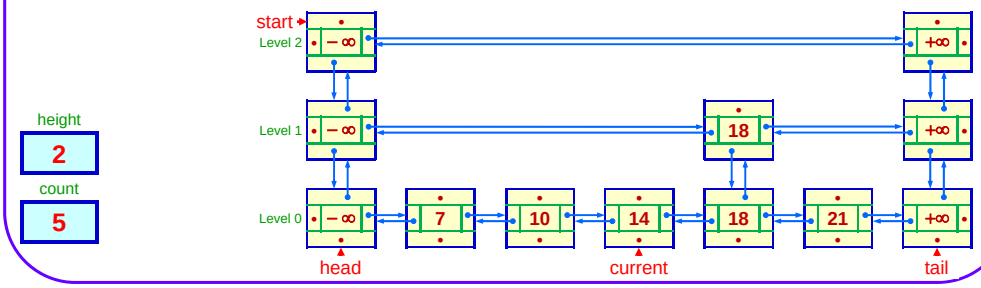
Mapping Operation: skipRemove()
Example: skipRemove();



Mapping Operation: isElement() ... *O*(1)

- **Postcondition**:
  - The return value is true if current is at a valid node, otherwise it returns false.
- **Code**:

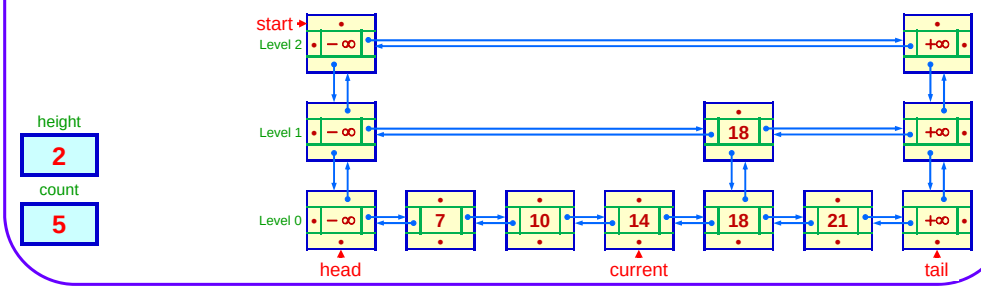
```
return ((current != head) &&
        (current != tail));
```



Mapping Operation: isEmpty() ... *O*(1)

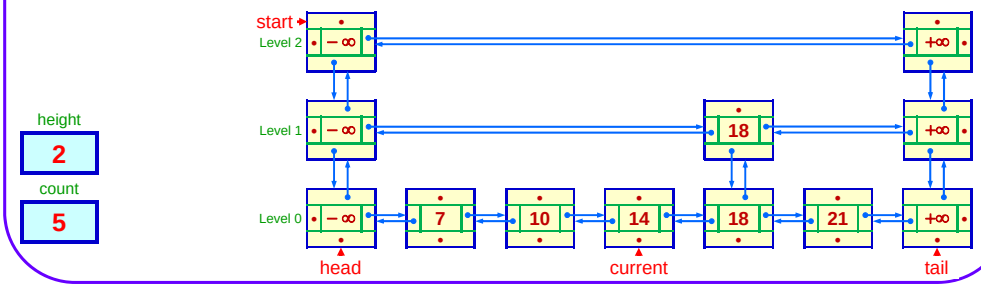
- **Postcondition**:
  - The return value is true if the list has no elements, otherwise, it is false.
- **Code**:

```
return (count == 0);
```



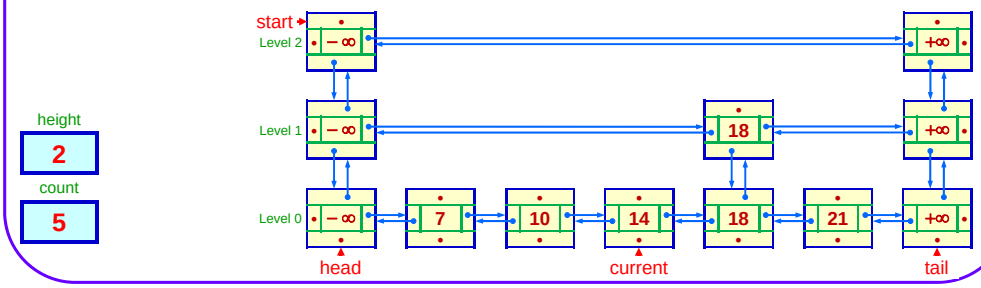
Mapping Operation: height() ... *O*(1)

- **Postcondition**:
  - The return value is the number of levels in the list, starting from zero.
- **Code**:
  return height;



Mapping Operation: size() ... *O*(1)

- **Postcondition**:
  - The return value is the number of nodes in the level zero list,.
- **Code**:
  return count;



Skip List Performance
Space Usage

- The space used by a skip list depends on the random bits used by each invocation of the insertion algorithm
- Use the following two basic probabilistic facts:

  Fact 1: The probability of getting i consecutive heads when flipping a coin is $1/2^i$.

  Fact 2: If each of n entries is present in a set with probability p, the expected size of the set is np.

- Consider a skip list with n entries:
  - By Fact 1, an entry is inserted in list S_i with probability $1/2^i$
  - By Fact 2, the expected size of list S_i is $n/2^i$
- The expected number of nodes used by the skip list is

$$\sum_{i=0}^{h} \frac{n}{2^i} = n\sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

- Thus, the expected space usage of a skip list with n items is $O(n)$.

Skip List Performance
List Height

- The running time of the search an insertion algorithms depends on the height h of the skip list.
- Show that with high probability, a skip list of n items has height $O(\log n)$
- Use the following probabilistic fact:

  Fact 3: If each of n events has probability p, then, the probability that at least one event occurs is at most np.

- Consider a skip list with n entries:
  - By Fact 1, we insert an entry in list S_i with probability $1/2^i$
  - By Fact 3, the probability that list S_i has at least one item is at most $n/2^i$
- Let, $h_1 = \log n$ and $h_2 = 3\log n$, then the probabilities that S_{h1} and S_{h2} have at least one entry is at most:

  $n/2^{\log n} = n/n = 1$ for h_1 (very high)

  and at most:

  $n/2^{3\log n} = n/n^3 = 1/n^2$ for h_2 (very low)
- Thus the probability that a skip list with n entries has a height $\geq 3\log n$ is at most $1/n^2$.

Skip List Performance
Search and Update Times

- The search time in a skip list is proportional to:
  - the number of drop-down steps, plus
  - the number of scan-left steps.
- The drop-down steps are bounded by the height of the skip list and thus are $O(\log n)$ with high probability.
- To analyze the scan-left steps, use another probabilistic fact:

  Fact 4: The expected number of coin tosses required in order to get tails is 2.

- When we scan left in a list, the destination key was not found in a higher list,
  - A scan-left step is associated with a former coin toss that gave tails.
- By Fact 4, in each list the expected number of scan-left steps is 2.
- Thus, the expected number of all scan-left steps is $O(\log n)$
- So, a search in a skip list takes $O(\log n)$ expected time.
- The analysis of insertion and deletion gives similar results.

The Skip List
Summary

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm
- In a skip list with n entries,
  - Expected space complexity is $O(n)$
  - Expected search, insertion and deletion time complexity is $O(\log n)$
- Using a more complex probabilistic analysis, one can show that these performance bounds also hold with high probability
- **Skip lists are fast and simple to implement in practice.**