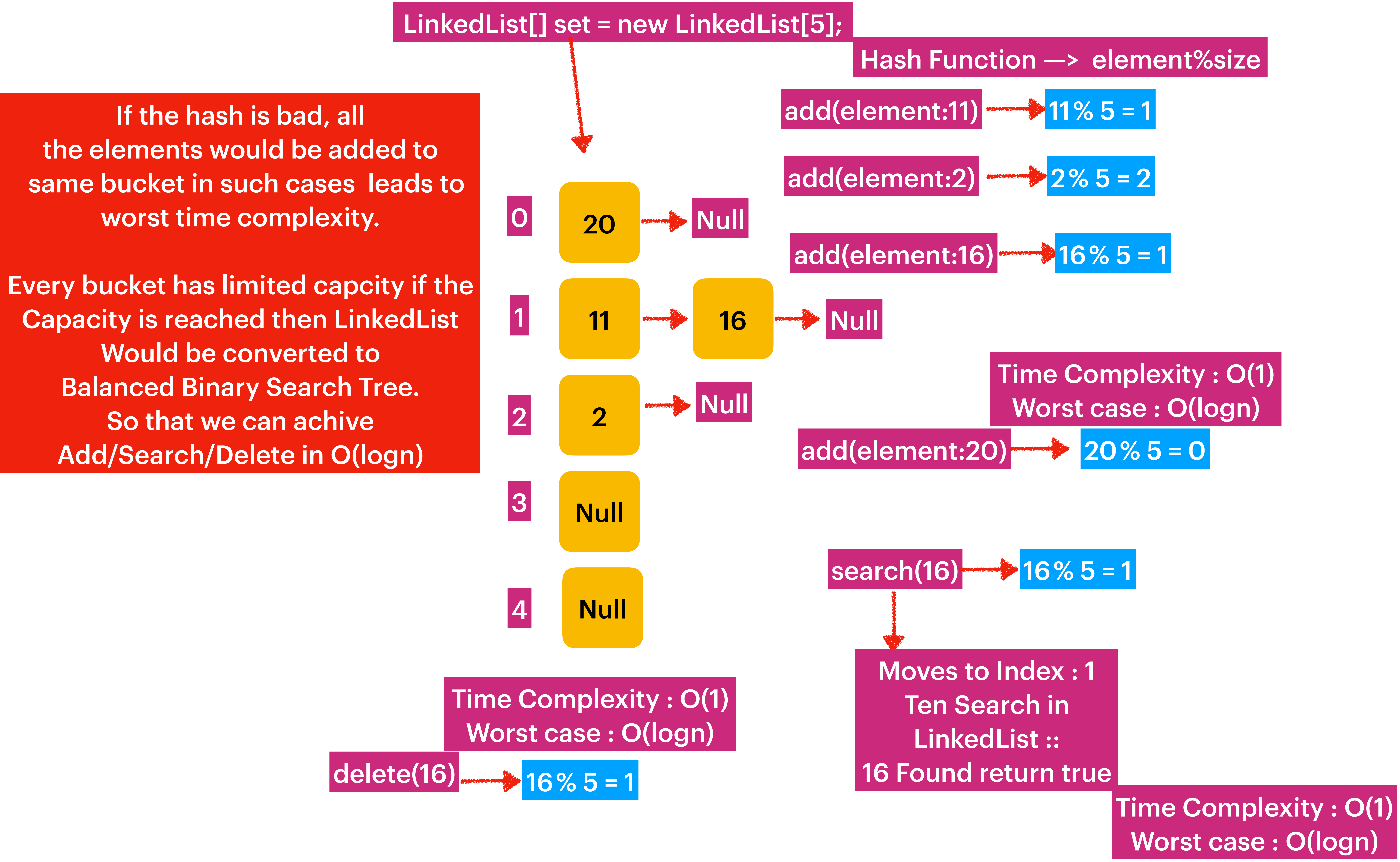
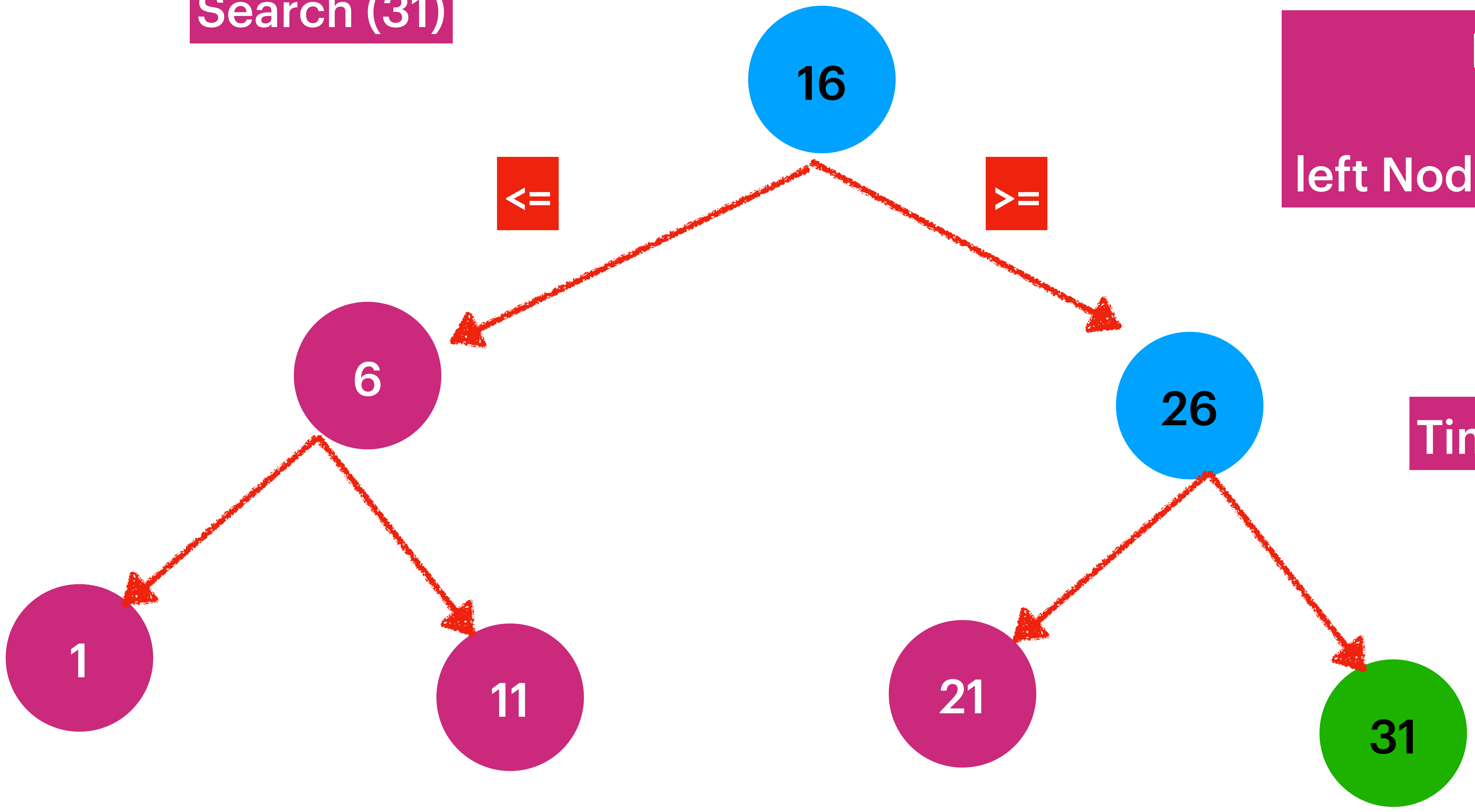

LinkedList[] set = new LinkedList[5];
Hash Function —> element%size
If the hash is bad, all the elements would be added to same bucket in such cases leads to worst time complexity.
Every bucket has limited capcity if the Capacity is reached then LinkedList Would be converted to Balanced Binary Search Tree. So that we can achive Add/Search/Delete in O(logn)
0
20
Null
1
11
16
Null
2
2
Null
3
Null
4
Null
add(element:11)
11 % 5 = 1
add(element:2)
2 % 5 = 2
add(element:16)
16 % 5 = 1
Time Complexity : O(1)
Worst case : O(logn)
add(element:20)
20 % 5 = 0
search(16)
16 % 5 = 1
Moves to Index : 1
Ten Search in
LinkedList ::
16 Found return true
Time Complexity : O(1)
Worst case : O(logn)
Time Complexity : O(1)
Worst case : O(logn)
delete(16)
16 % 5 = 1

1 -> 6 -> 11 -> 16 -> 21 -> 26 -> 31

When we have bad hash then there is possibility that
All the elements mapped same bucket which causes O(n) in search/delete operation,
To avoid this we will have hieight balanced binery search tree when the size >= capacity

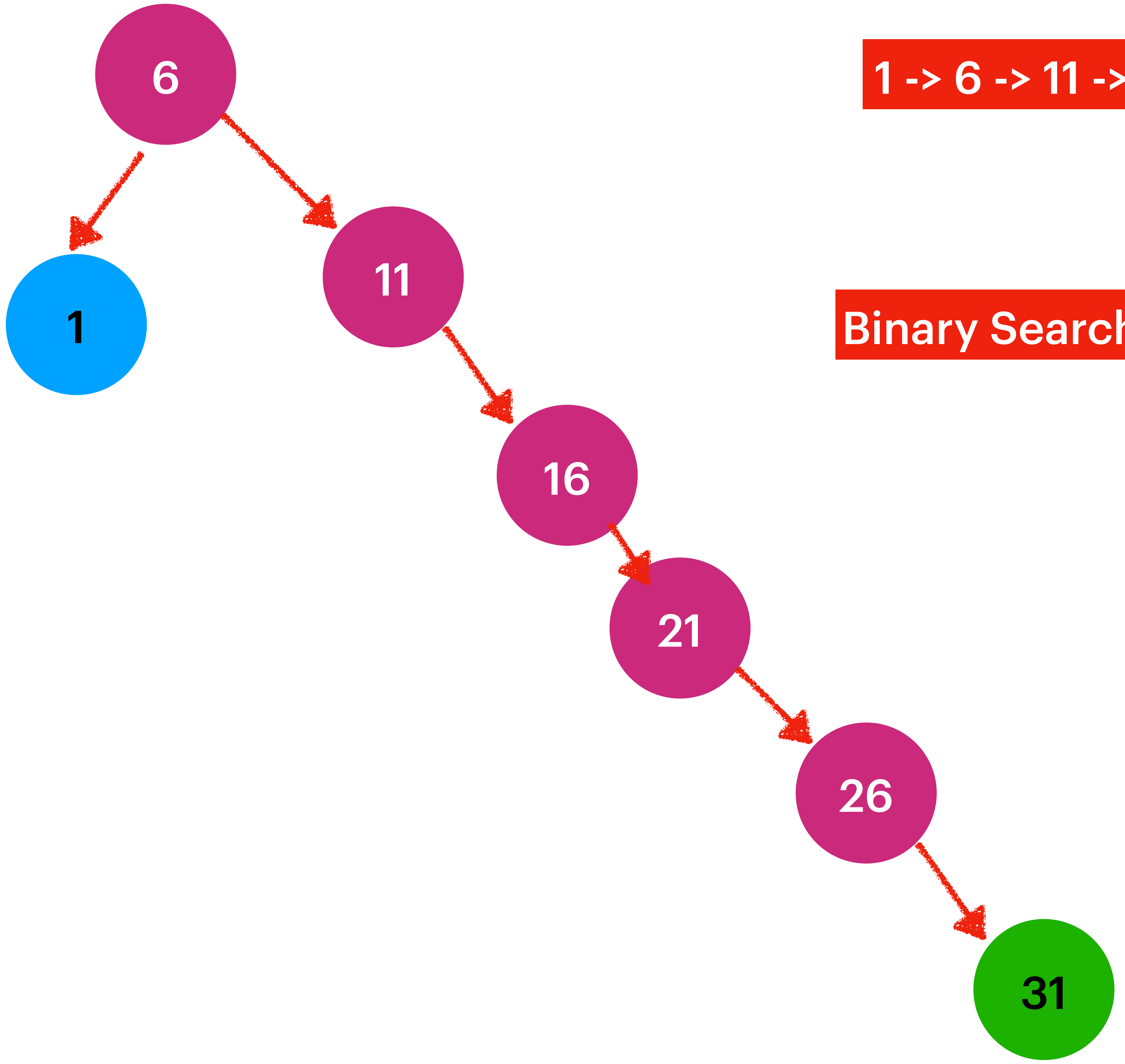
Search (31)



Binary Search Tree : Take any Node :
left Node value are <= & Right Node values >=

Time Complexity : O(logn)

Height Balanced Binary Search Tree :
Its a Binary Search Tree, the max height difference between left sub tree and right sub tree is 1.



1 -> 6 -> 11 -> 16 -> 21 -> 26 -> 31

Binary Search Tree but not height balanced.

Search(31)

In Worst case it becomes : O(n)

LinkedList[] set = new LinkedList[5];

Hash Function —> element%size

add(element:11) → 11% 5 = 1

add(element:2) → 2% 5 = 2

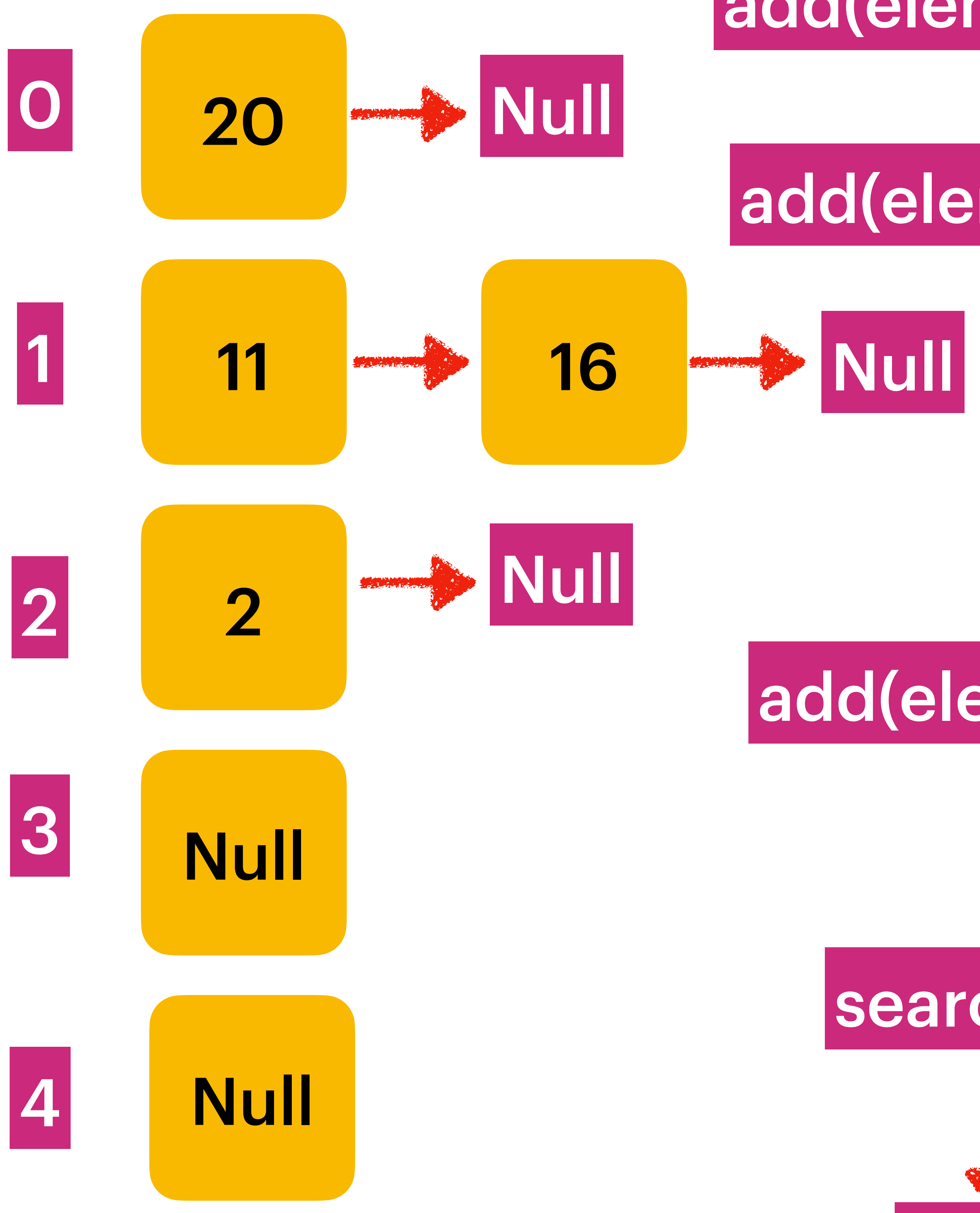
add(element:16) → 16% 5 = 1

add(element:20) → 20% 5 = 0

Time Complexity : O(1)
Worst case : O(logn)

If the hash is bad, all the elements would be added to same bucket in such cases leads to worst time complexity.

Every bucket has limited capcity if the Capacity is reached then LinkedList Would be converted to Balanced Binary Search Tree. So that we can achive Add/Search/Delete in O(logn)



add(2) → 2% 5 = 2

On bucket/index:2
value is already presets.
So that element won’t be added.

delete(16) → 16% 5 = 1

Time Complexity : O(1)
Worst case : O(logn)

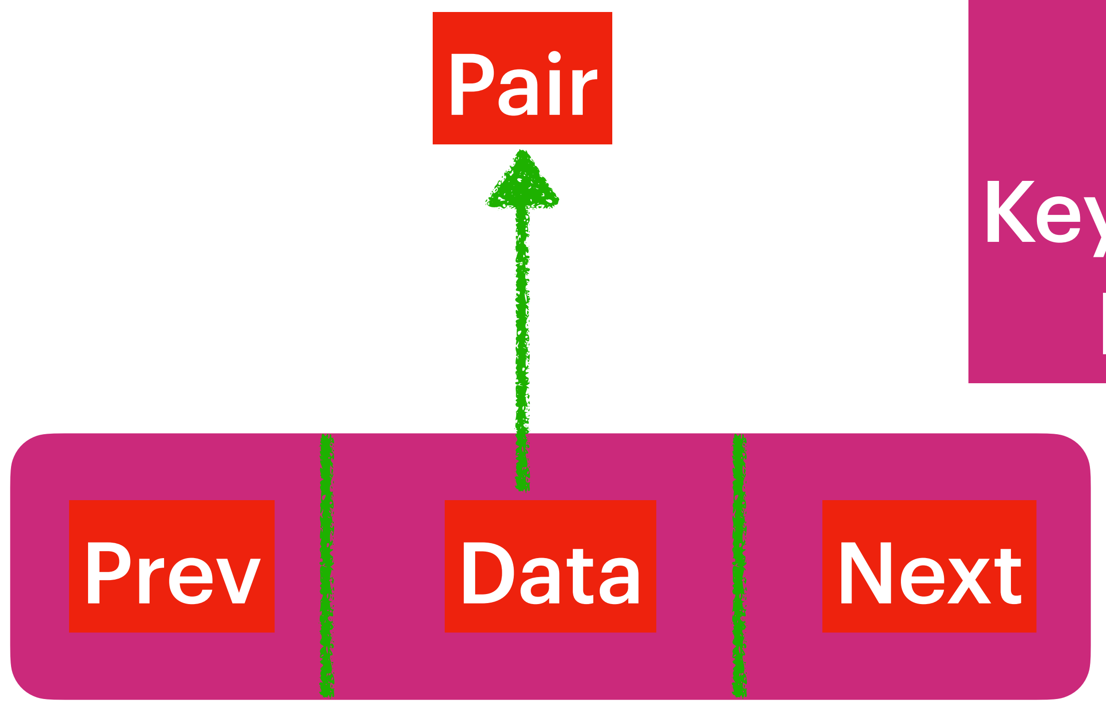
search(16) → 16% 5 = 1

Moves to Index : 1
Ten Search in
LinkedList ::
16 Found return true

Time Complexity : O(1)
Worst case : O(logn)

Design Map

Even in map as well each bucket Has threefold limit, once it reached Then converts to Height Balanced Binary Search Tree.



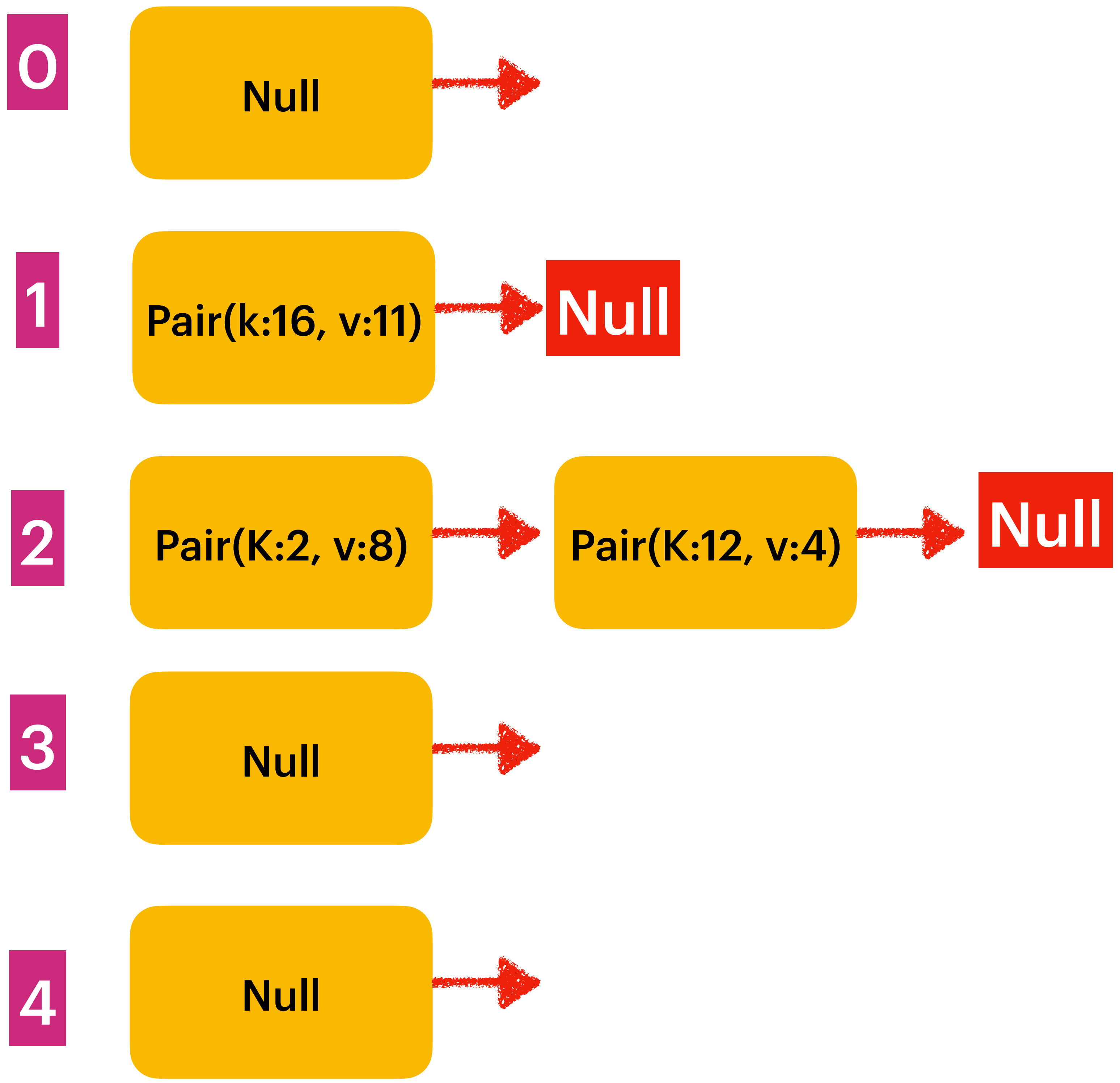
Map has key and values. We apply hashing on key.

It means what ?

Key can not be duplicate but value can be duplicate.
If the key is already exist we just replace value;

```
class Pair
{
int key;
int value;
public Pair(int k, int v)
{
key = k;
value = v;
}
}
```

LinkedList<Pair> [] set = new LinkedList[5];



Time Complexity : O(1)
Worst case : O(logn)

put(k:2, v:3) —>
Hash = 2%5 = 2

put(k:16, v:11) —>
Hash = 16%5 = 1

put(k:12, v:4) —>
Hash = 12%5 = 2

put(k:2, v:8) —>
Hash = 2%5 = 2

Key:2 already exist so replaces the value.

Time Complexity : O(1)
Worst case : O(logn)

search(k:12) —>
Hash = 12%5 = 2

On second bucket key 12 is exist so returns true.

Time Complexity : O(1)
Worst case : O(logn)

remove(k:2) —>
Hash = 2%5 = 2

Key:2 Present so removes the node.