Concurrent skip list algorithm

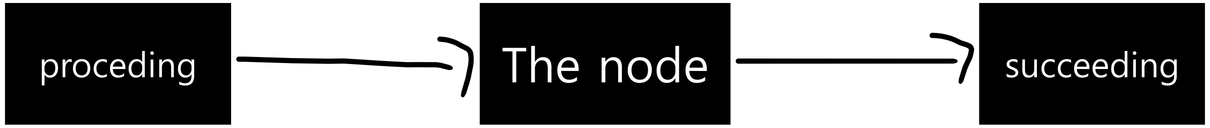
1. Create
2. Search
3. Insert
4. Delete

Presenter: Kim Kyung Min

Reference: https://github.com/shreyas-gopalakrishna/Concurrent-Skip-list

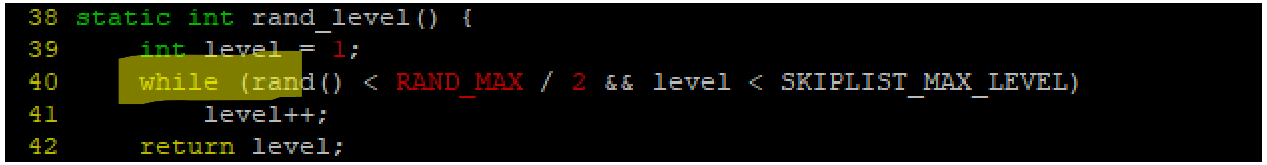
Marked: A variable that determines whether the node is being deleted
marked = 0 : The node is being deleted
marked = 1 : The node is not deleted

Fully-linked: A variable that determines whether the node is connected to the proceding or succeeding node
fully-linked = 0 : No connection with preceding and subsequent nodes
fully-linked = 1 : connection with preceding and subsequent nodes



Create

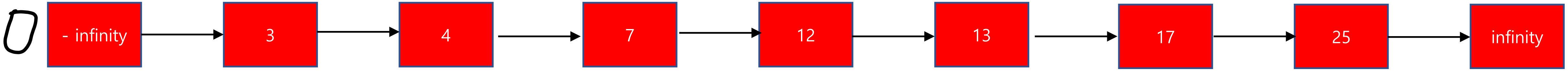
- The entire list is sorted at the bottom
- The leading and trailing values are -infinity, values corresponding to infinity.
- For every level you go up, there is a 1/2 chance that each element can move upstairs.



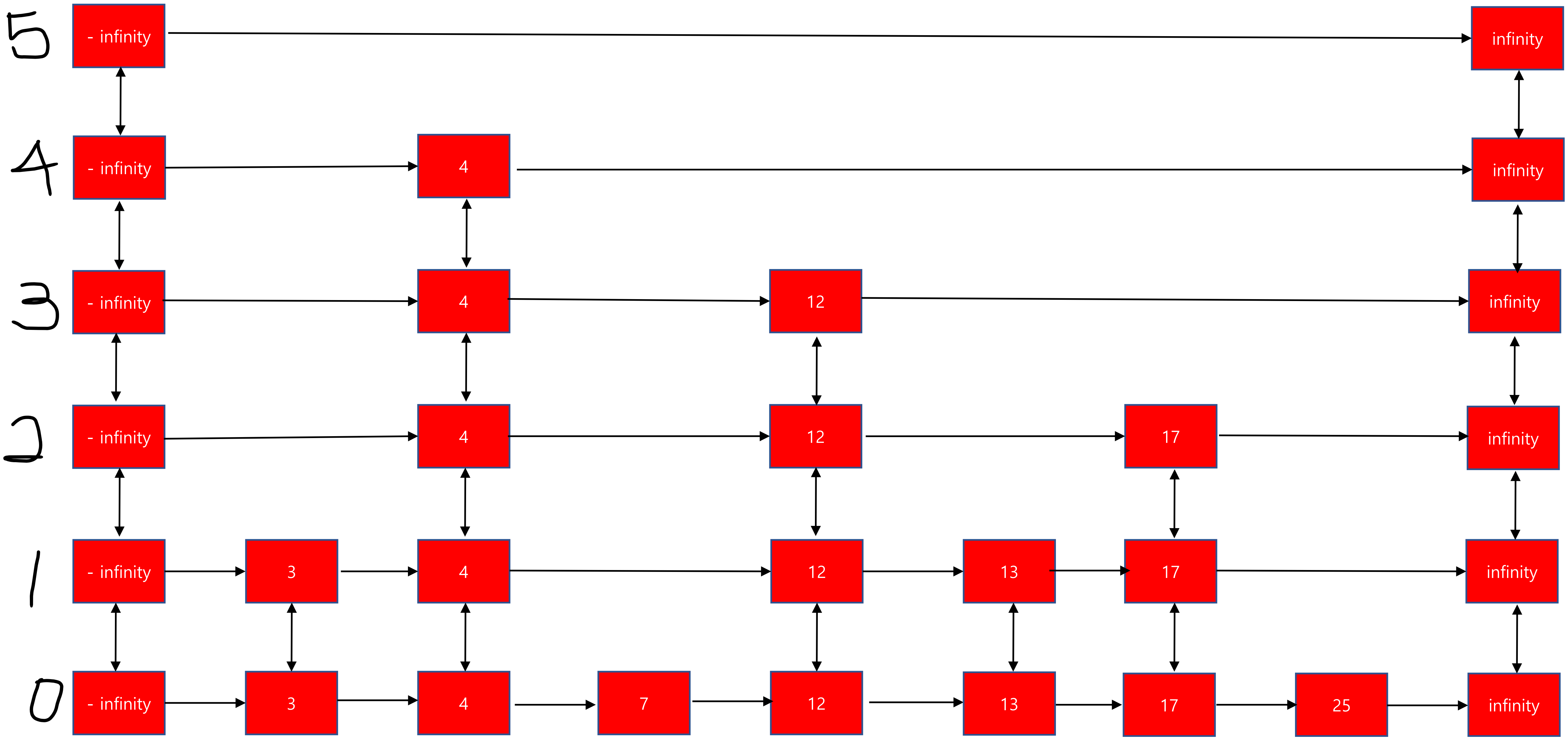
```
38 static int rand_level() {
39     int level = 1;
40     while (rand() < RAND_MAX / 2 && level < SKIPLIST_MAX_LEVEL)
41         level++;
42     return level;
```

Line 40: level must be less than the max level and the value called with rand() must be less than RAND_MAX /2

The shape of the skip list can be different each time it is made??

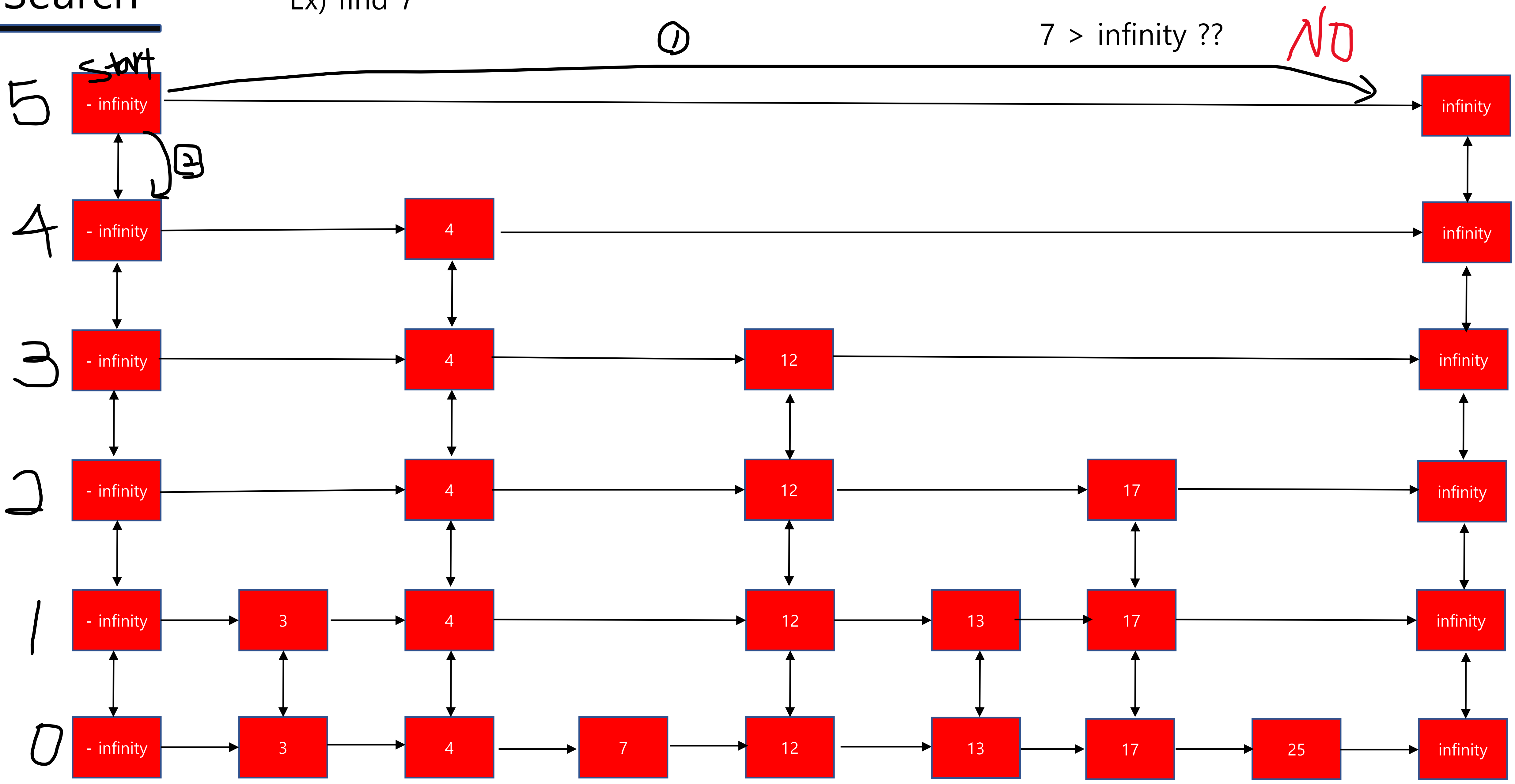


Create


5
- infinity
infinity
4
- infinity
4
infinity
3
- infinity
4
12
infinity
2
- infinity
4
12
17
infinity
1
- infinity
3
4
12
13
17
infinity
0
- infinity
3
4
7
12
13
17
25
infinity

Search

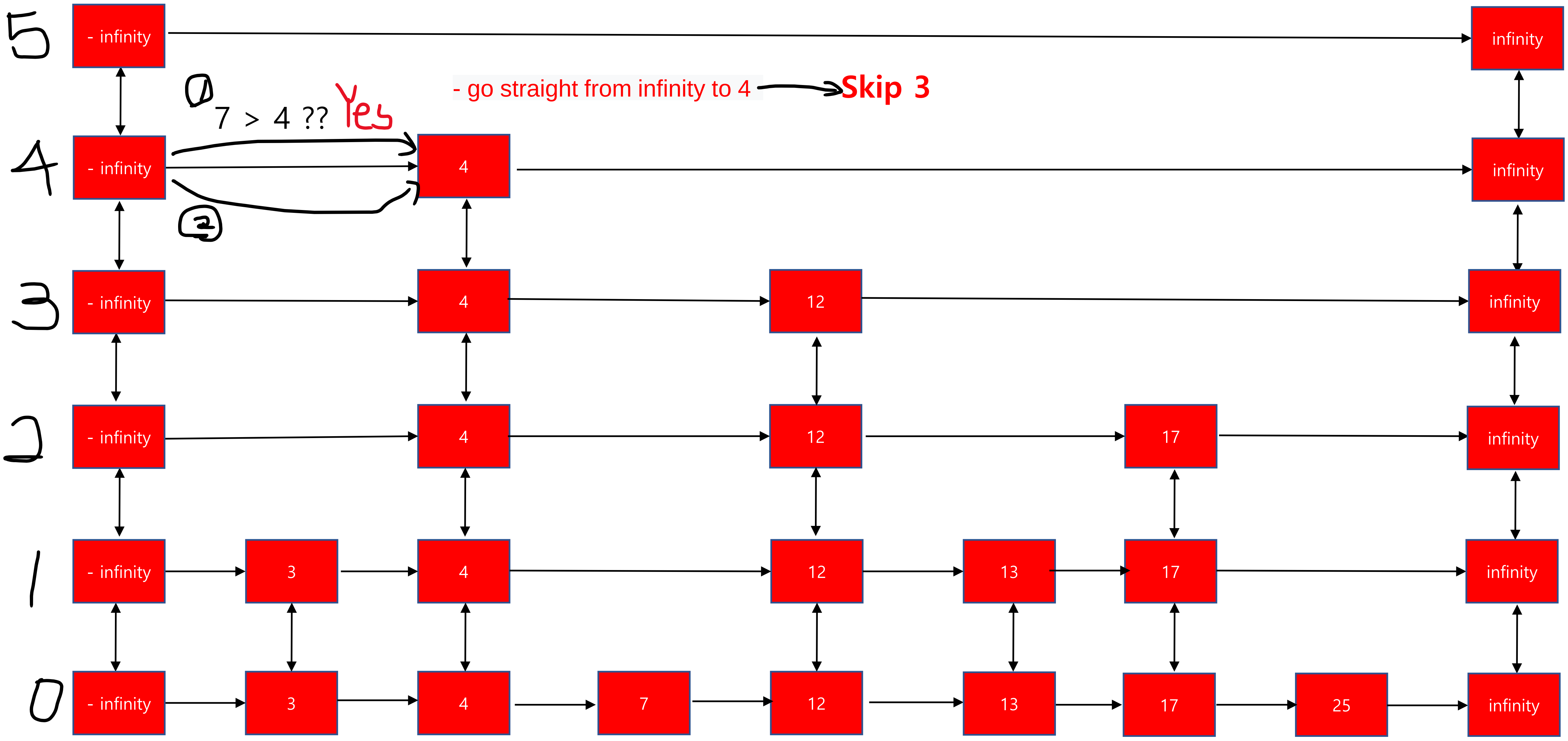
Ex) find 7



7 > infinity ??

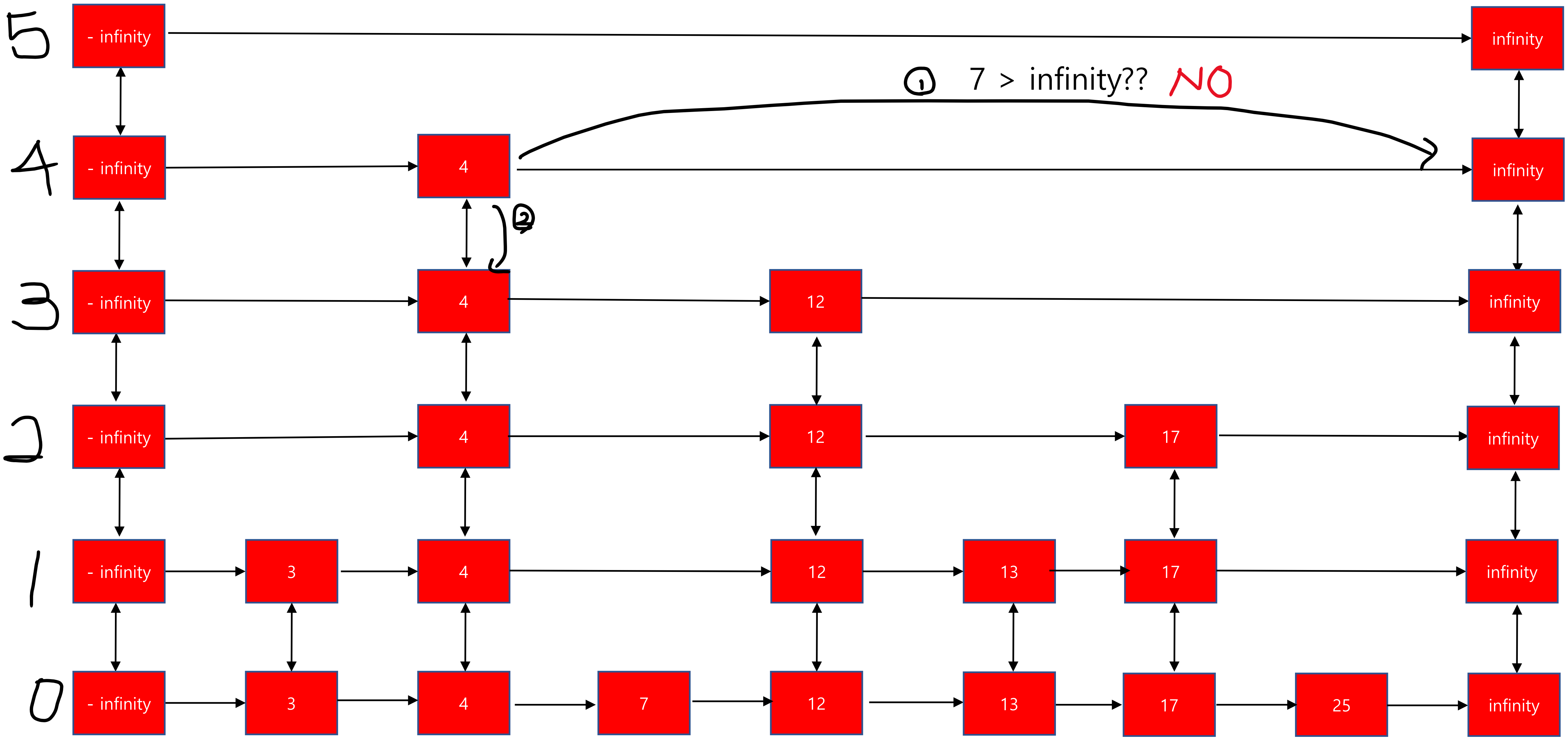
NO

Search

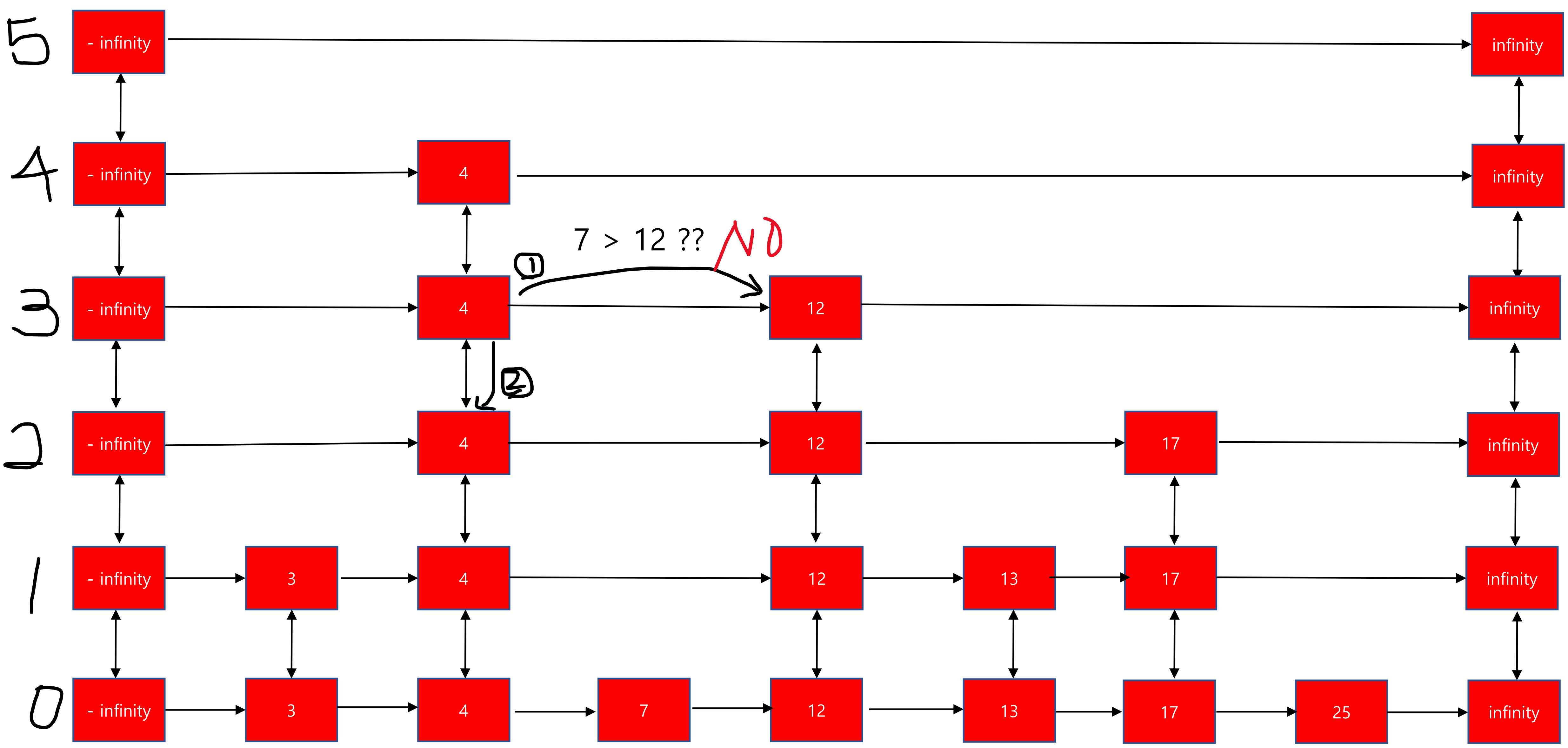


5
4
3
2
1
0
- infinity
infinity
4
12
17
3
13
7
25
①
7 > 4 ?? Yes
②
- go straight from infinity to 4
Skip 3

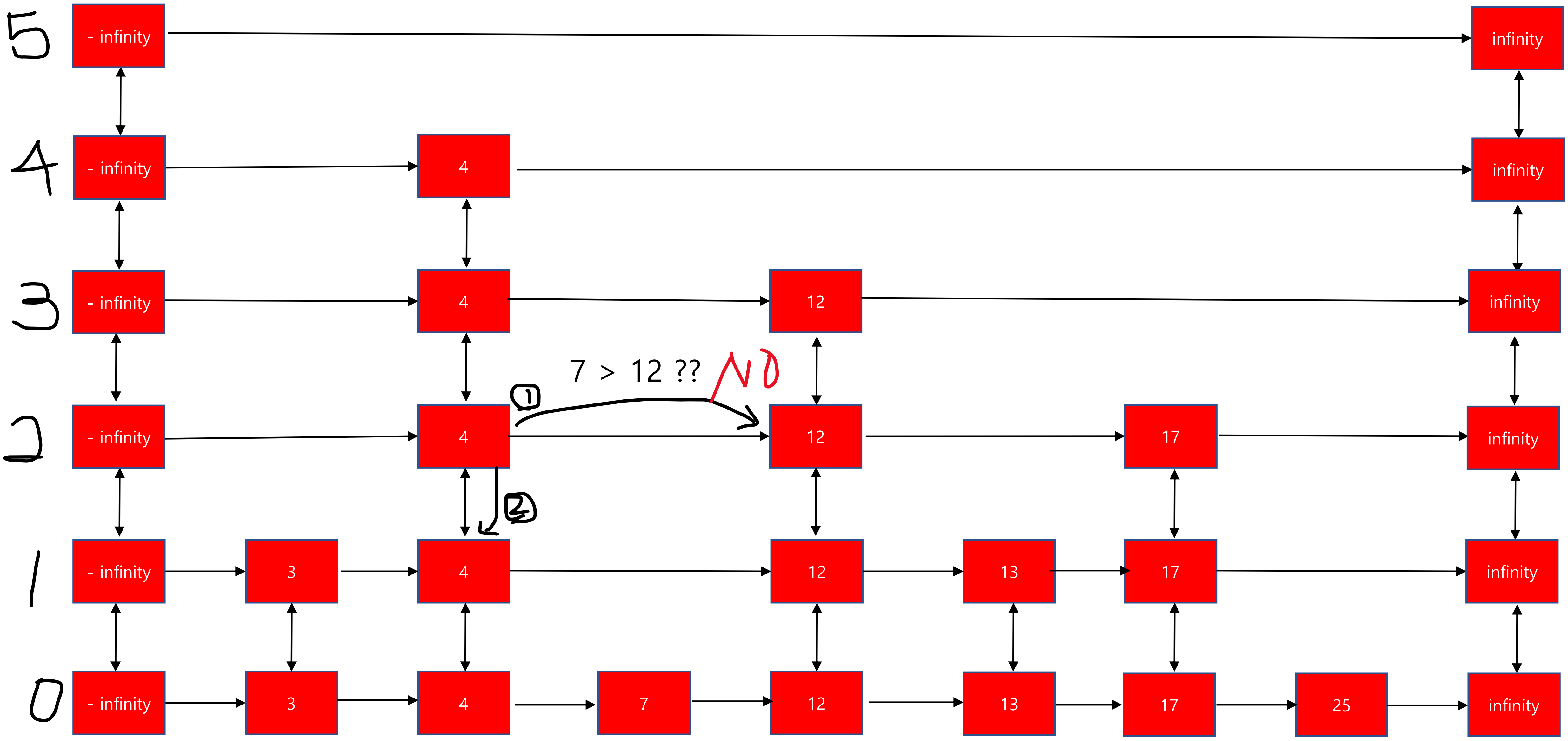
Search


5
4
3
2
1
0
- infinity
- infinity
- infinity
- infinity
- infinity
- infinity
infinity
infinity
infinity
infinity
infinity
infinity
① 7 > infinity?? NO
②
4
4
4
4
4
3
3
7
12
12
12
12
13
13
17
17
17
25

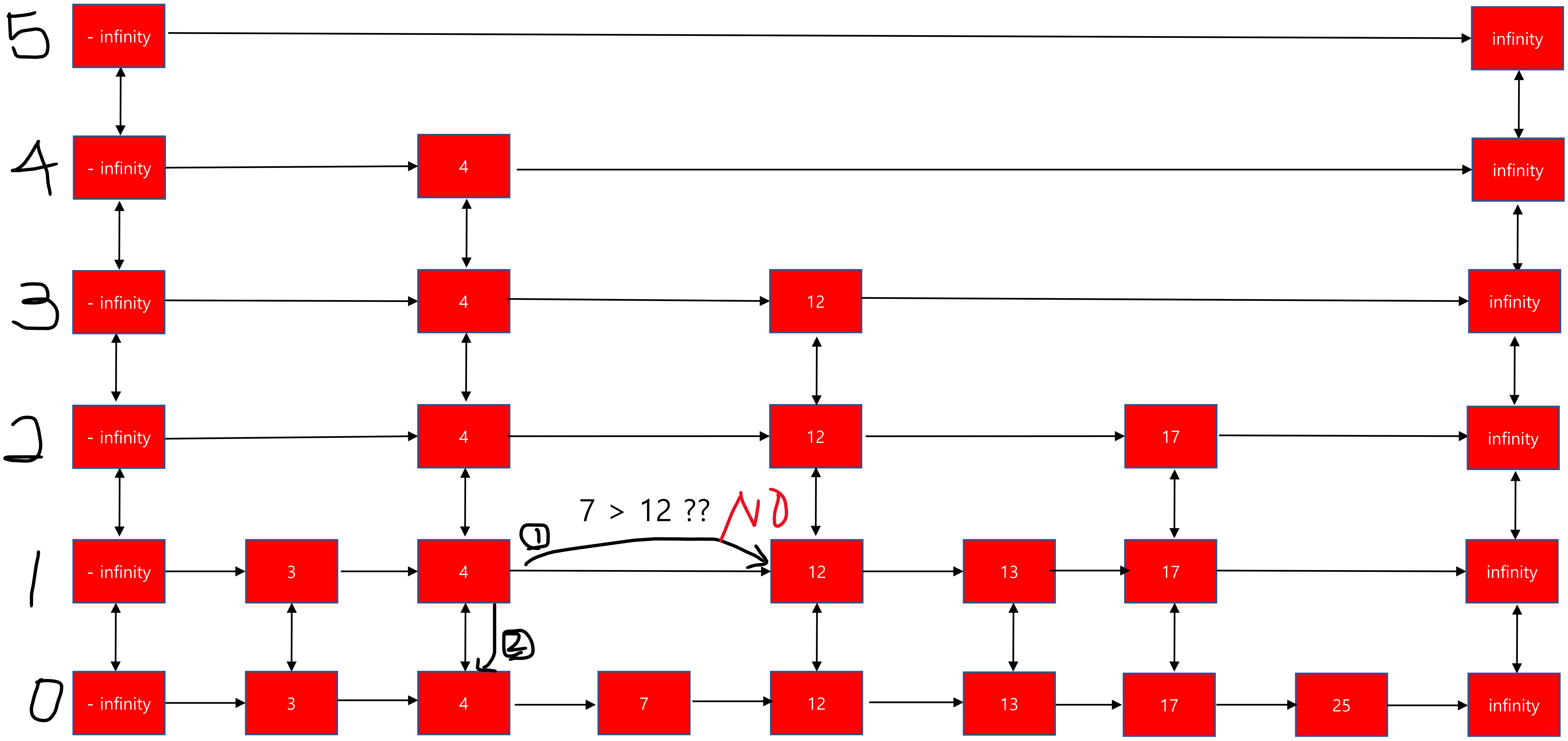
Search


5
4
3
2
1
0
- infinity
- infinity
- infinity
- infinity
- infinity
- infinity
4
4
4
4
4
3
3
7
12
12
12
12
13
13
17
17
17
25
infinity
infinity
infinity
infinity
infinity
infinity
7 > 12 ??
NO
1
2

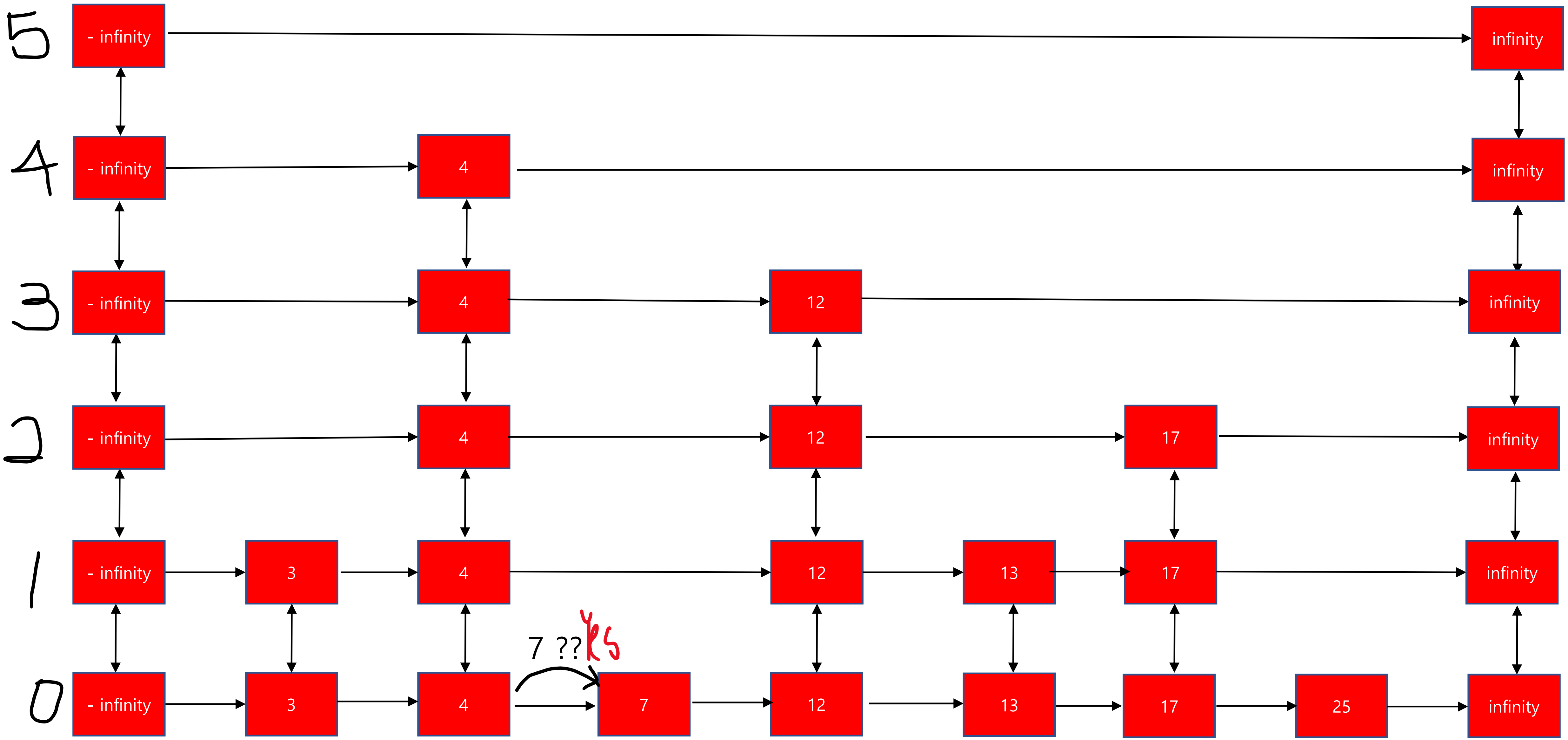
Search


5
- infinity
infinity
4
- infinity
4
infinity
3
- infinity
4
12
infinity
7 > 12 ?? NO
①
2
- infinity
4
12
17
infinity
②
1
- infinity
3
4
12
13
17
infinity
0
- infinity
3
4
7
12
13
17
25
infinity

Search


5
- infinity
infinity
4
- infinity
4
infinity
3
- infinity
4
12
infinity
2
- infinity
4
12
17
infinity
1
- infinity
3
4
12
13
17
infinity
0
- infinity
3
4
7
12
13
17
25
infinity
① 7 > 12 ?? NO
②

Search



Find 7

1. As it gets closer to the search key, it drops to the ‘upper->lower’ level.

2. When the search key is found (when the desired node is found)

```
If(Marked = 0 and fully-linked = 1)
        Return value

Else
        false
```

3. Allows read requests to run in parallel (lock free, non blocking)

While a specific thread is inserting or deleting, another thread searches -> the consistency of content is broken

While a specific thread is searching, other threads are also searched -> There doesn't seem to be any problem.

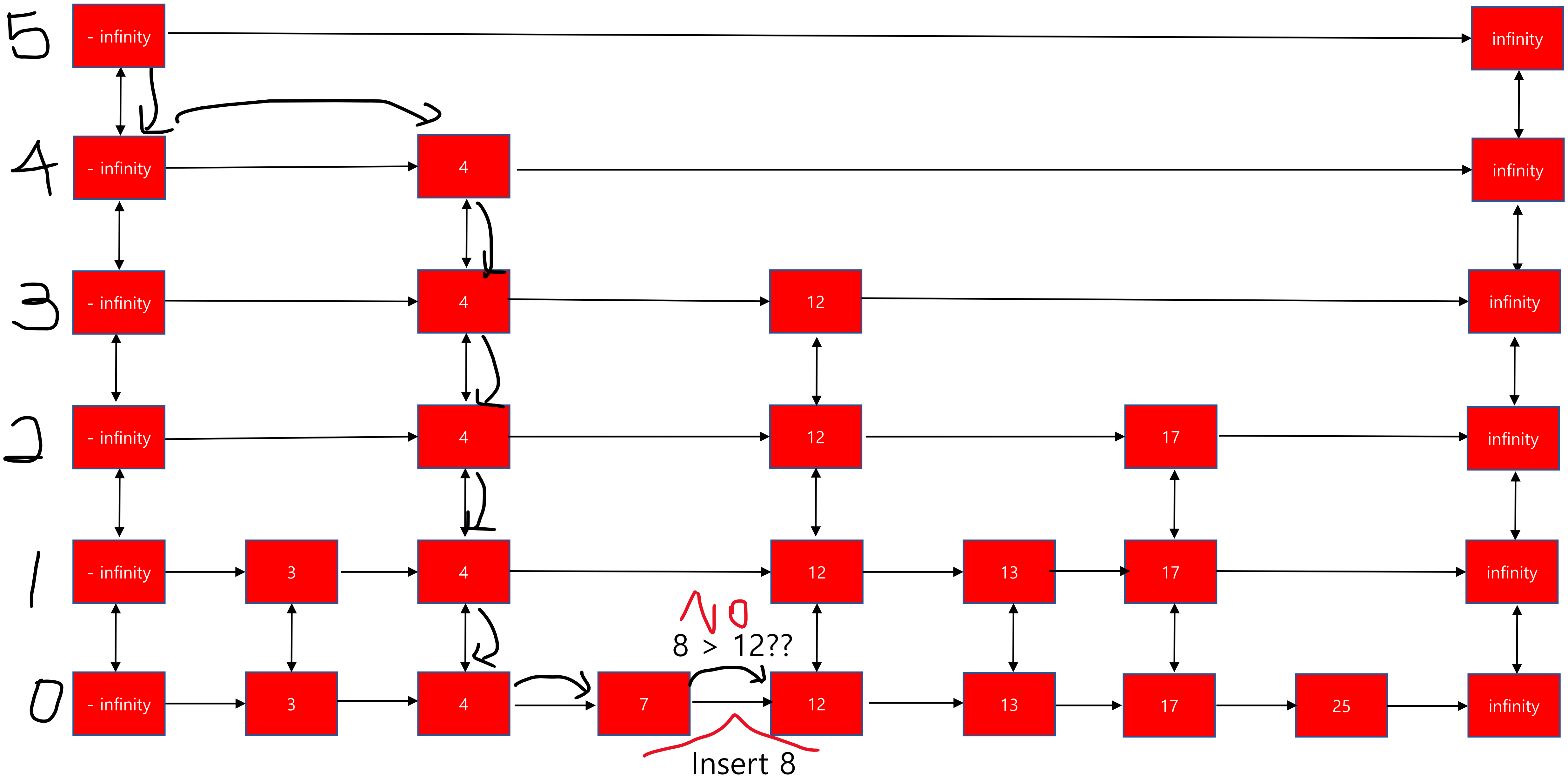
If multiple threads can search simultaneously, locks can be used efficiently.

However, it must be guaranteed that there is no insert or delete of another thread during the search of a specific thread.

Insert

Ex) insert 8

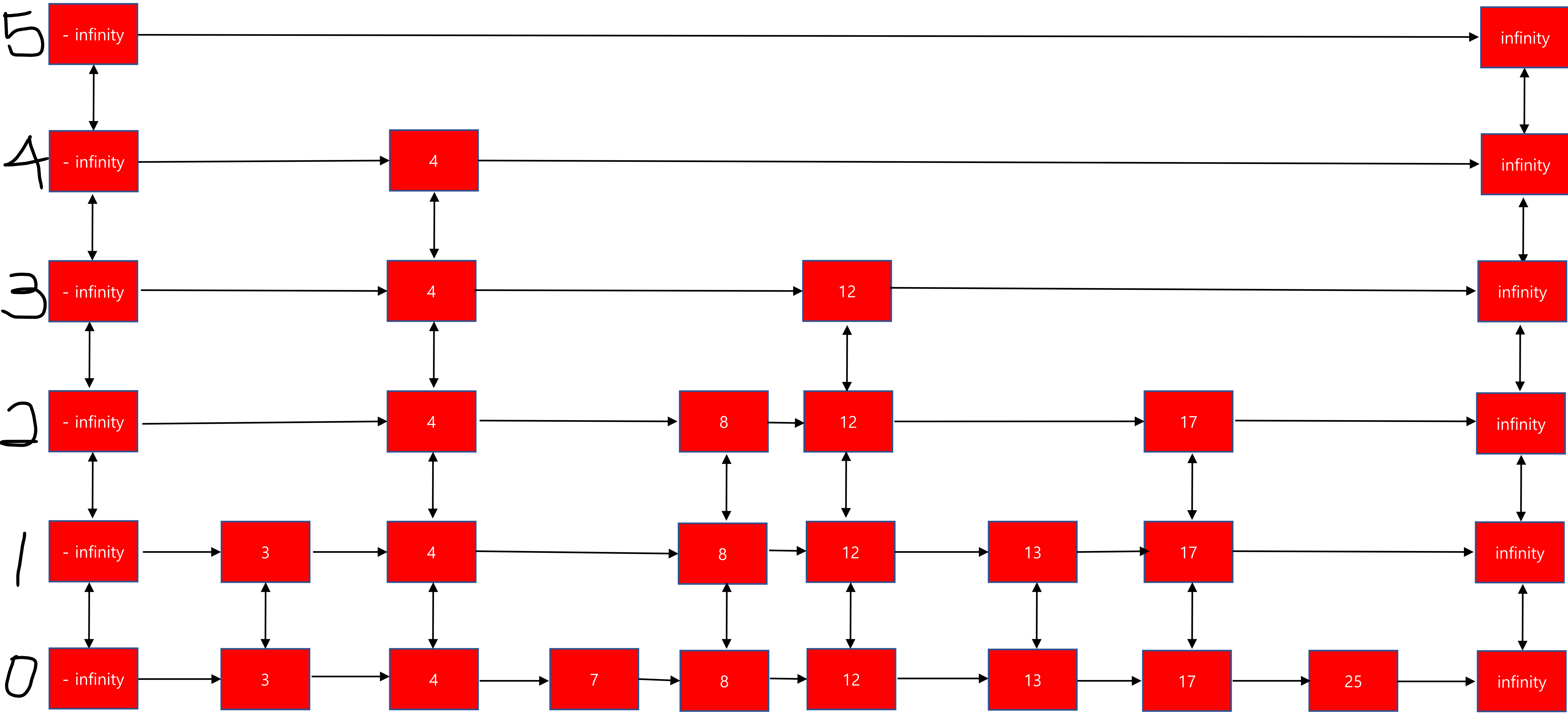
1. **Find location**
2. Insert(Flip & level)



Insert

Ex) insert 8

1. Find location
2. **Insert(Flip & level)**



(Check the presence of elements before insertion and check mark and fully-linked variables)

1. element already exists but node not showing -> insert x as already in skiplist

2. If element is present and not fully connected -> wait until fully connected

3. element exists but node is visible -> node is being deleted, wait and later Retry the entire insert algorithm

1. Find references to the preceding and following of the node insertion point.

2. Neither the preceding nor the succeeding nodes are marked, and it is necessary to check whether the next of the preceding nodes will succeed in each level.

3. 1, 2 not satisfied -> Wait and retry the entire insertion algorithm

1. When all of the above conditions are met, it starts to hold the locks of the preceding nodes at the same time (locks on the subsequent nodes are required x)

2. Before inserting a node, the level of a new node is randomly determined

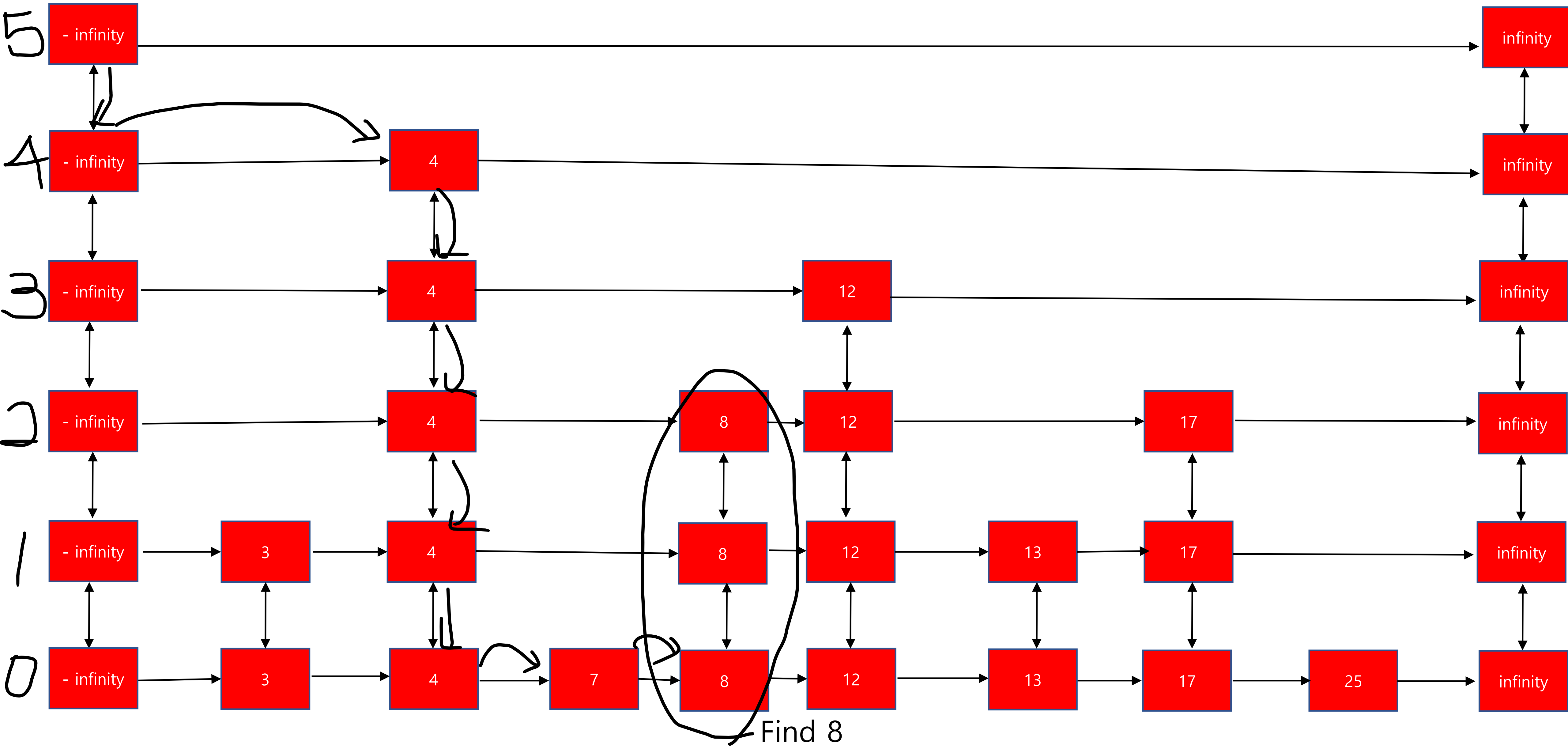
3. Concatenate preceding, new, and subsequent nodes

4. Fully-linked = 1 -> Release all locks -> Simultaneous insertion complete

Delete

Ex) delete 8

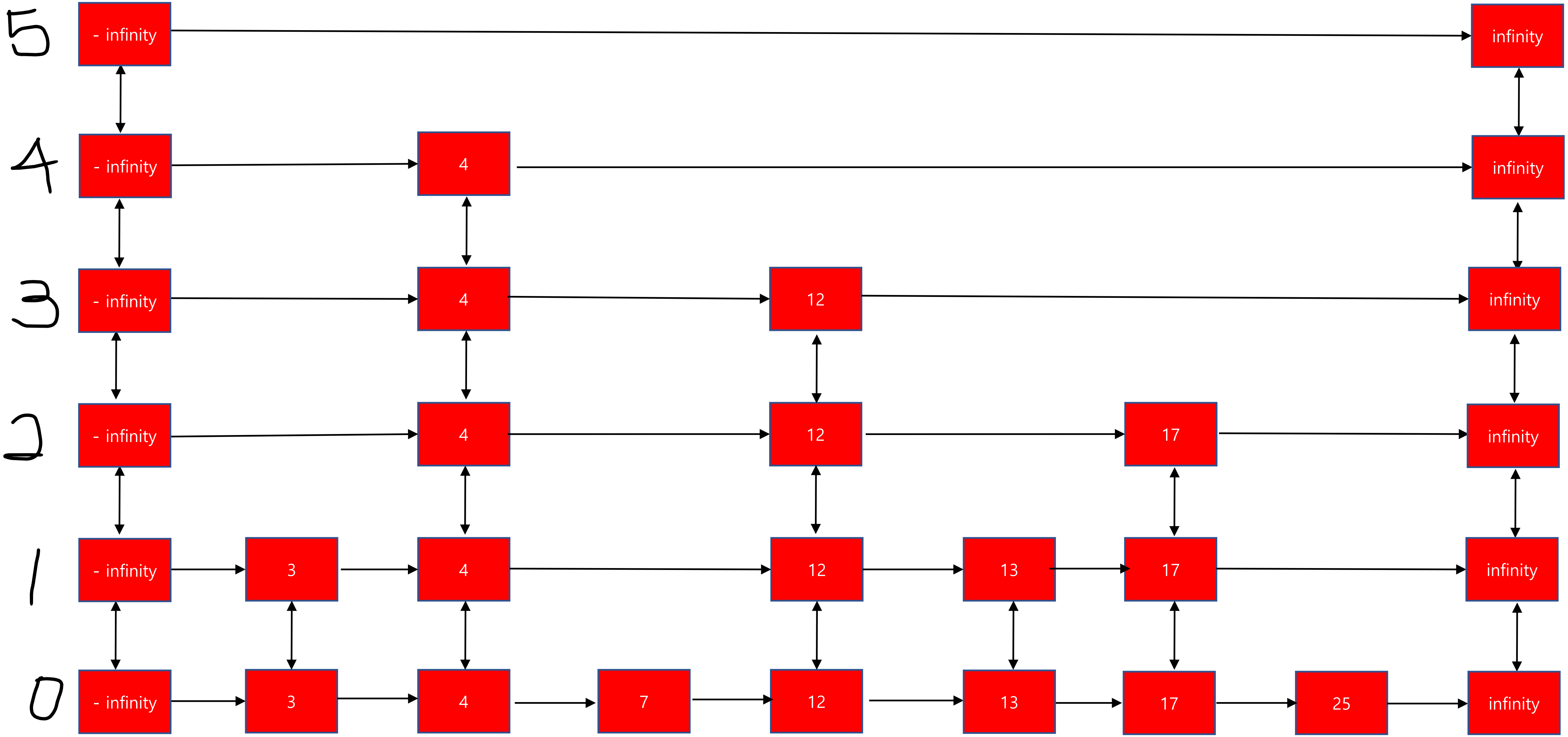
1. **Find location**
2. Delete node



Delete

Ex) delete 8

1. Find location
2. **Delete node**



1. Check if the node you want to clear exists -> false if not

2. Check if Fully-linked = 1, marked = 0 -> Retry deletion algorithm if condition is not met

1. Find references to preceding and subsequent nodes

2. Acquire the lock of the node to be deleted -> Acquire the lock of the preceding node

3. Check that the preceding node is marked =0, ‘preceding node -> next = delete node‘

4. If not satisfied 3 times, release all locks -> Retry the deletion algorithm

1. When all of the above conditions are met,
   the node is deleted and the preceding node is connected to the succeeding node.

2. Simultaneous deletion is completed when lock is released