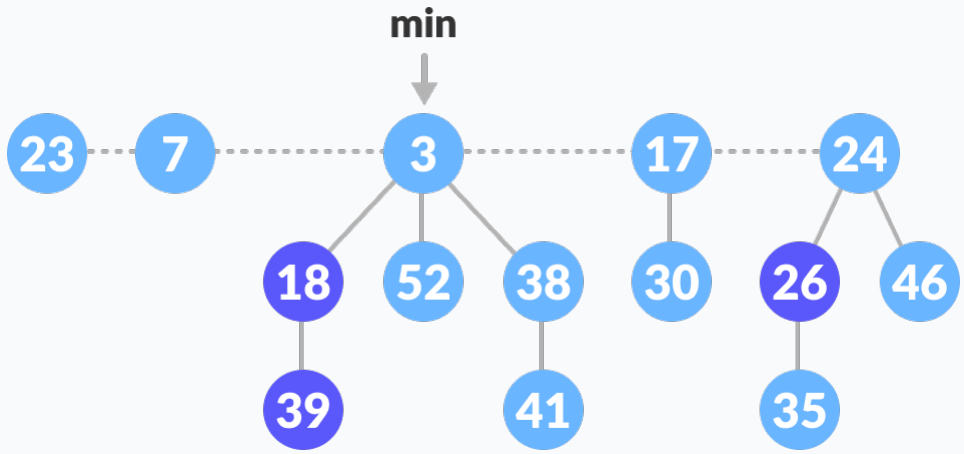
Fibonacci Heap

A fibonacci heap is a data structure that consists of a collection of trees which follow min heap or max heap property. We have already discussed **min heap** and **max heap property** in the Heap Data Structure article. These two properties are the characteristics of the trees present on a fibonacci heap.
In a fibonacci heap, a node can have more than two children or no children at all. Also, it has more efficient heap operations than that supported by the binomial and binary heaps.

The fibonacci heap is called a **fibonacci** heap because the trees are constructed in a way such that a tree of order n has at least F_{n+2} nodes in it, where F_{n+2} is the $(n + 2)^{th}$ Fibonacci number.



Fibonacci Heap

Properties of a Fibonacci Heap

Important properties of a Fibonacci heap are:

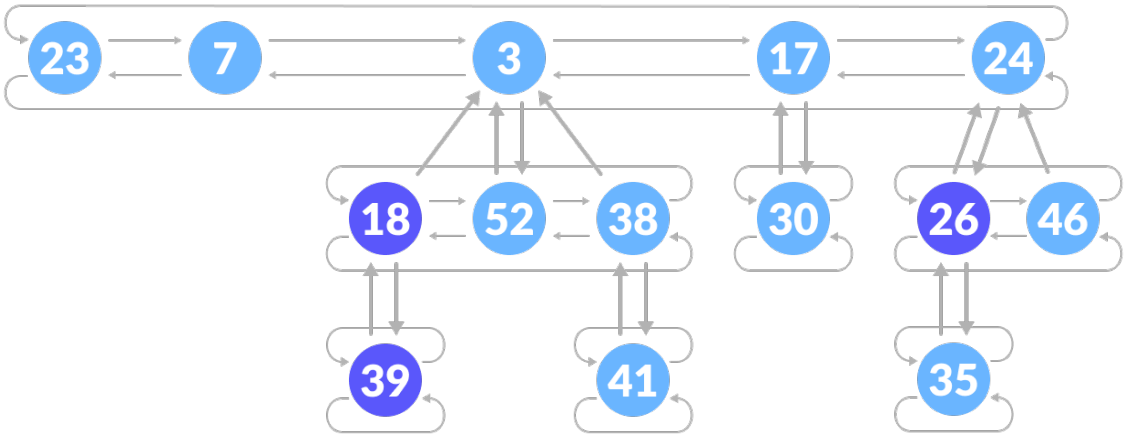
1. It is a set of **min heap-ordered** trees. (i.e. The parent is always smaller than the children.)
2. A pointer is maintained at the minimum element node.
3. It consists of a set of marked nodes. (Decrease key operation)
4. The trees within a Fibonacci heap are unordered but rooted.

Memory Representation of the Nodes in a Fibonacci Heap

The roots of all the trees are linked together for faster access. The child nodes of a parent node are connected to each other through a circular doubly linked list as shown below.

There are two main advantages of using a circular doubly linked list.

1. Deleting a node from the tree takes `O(1)` time.
2. The concatenation of two such lists takes `O(1)` time.



Fibonacci Heap Structure

Operations on a Fibonacci Heap

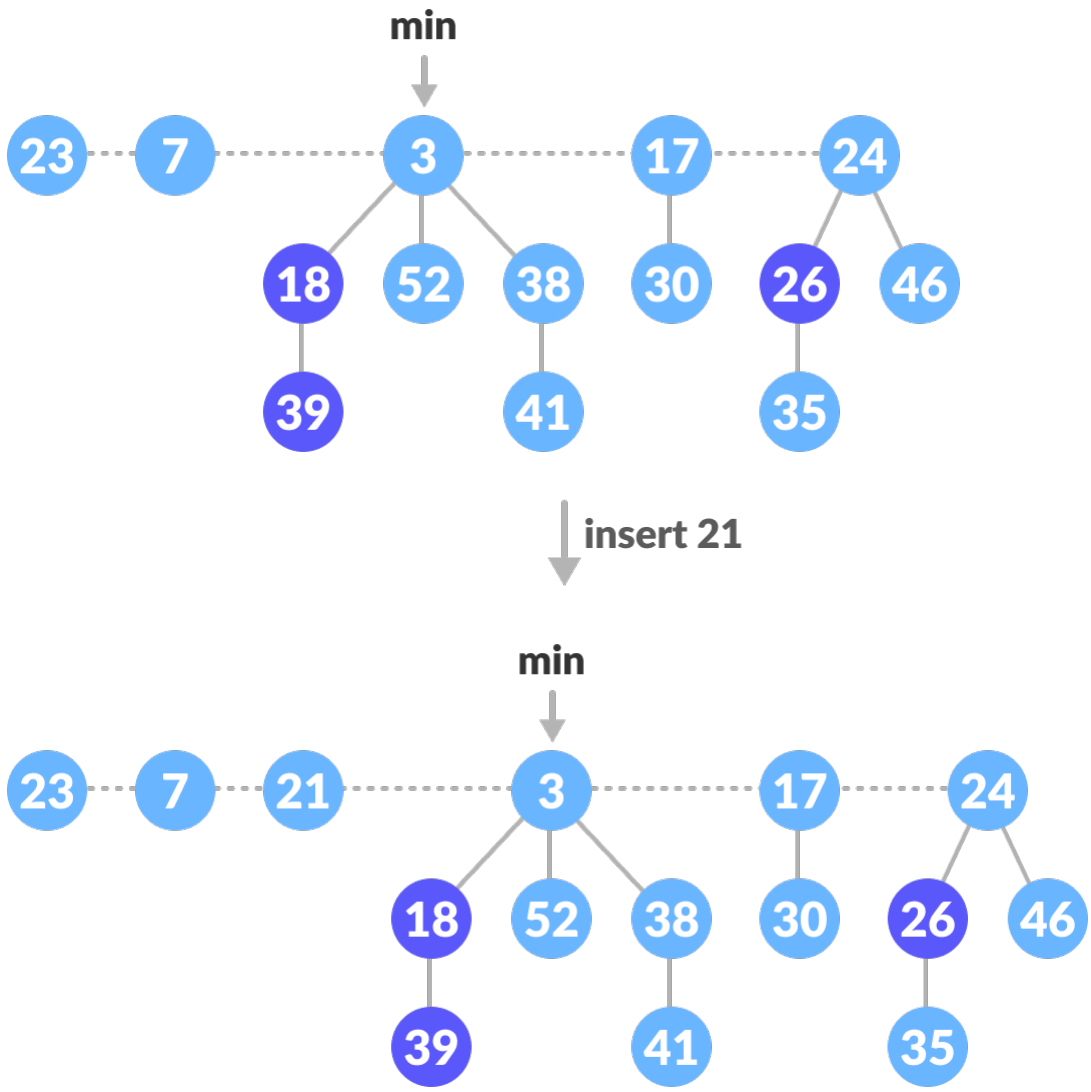
Insertion

Algorithm

```
insert(H, x)

    degree[x] = 0

    p[x] = NIL

    child[x] = NIL

    left[x] = x

    right[x] = x

    mark[x] = FALSE

    concatenate the root list containing x with root list H

    if min[H] == NIL or key[x] < key[min[H]]

        then min[H] = x

    n[H] = n[H] + 1
```

Inserting a node into an already existing heap follows the steps below.

1. Create a new node for the element.
2. Check if the heap is empty.
3. If the heap is empty, set the new node as a root node and mark it `min`.
4. Else, insert the node into the root list and update `min`.



In

sertion Example

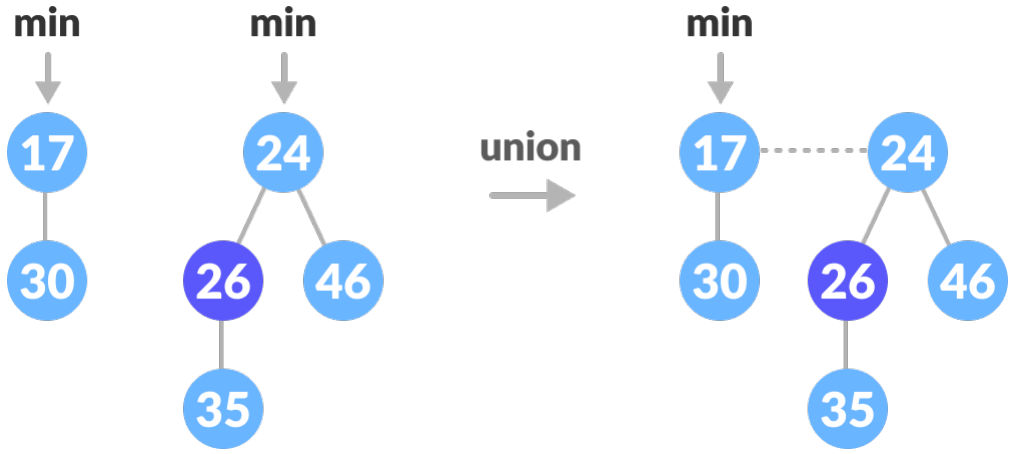
Find Min

The minimum element is always given by the `min` pointer.

Union

Union of two fibonacci heaps consists of following steps.

1. Concatenate the roots of both the heaps.
2. Update `min` by selecting a minimum key from the new root lists.



Union of two heaps

Extract Min

It is the most important operation on a fibonacci heap. In this operation, the node with minimum value is removed from the heap and the tree is re-adjusted.

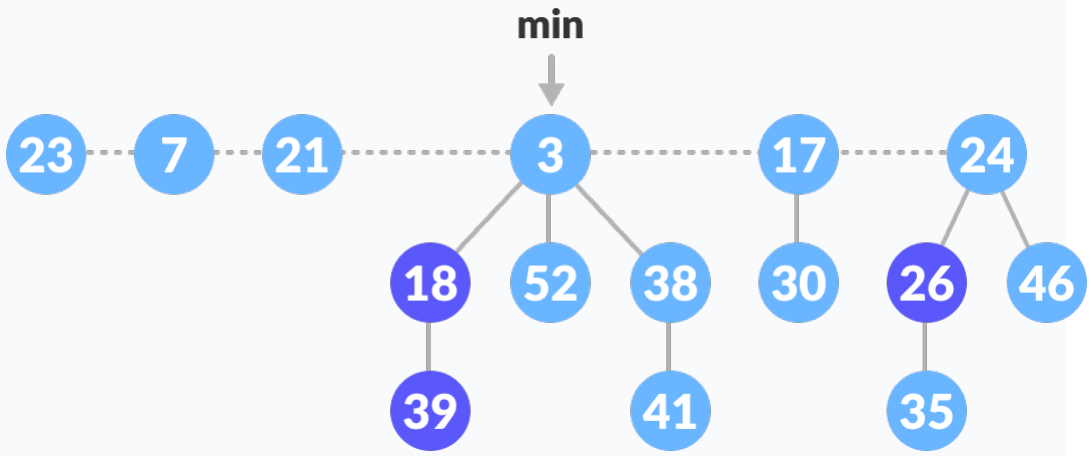
The following steps are followed:

1. Delete the min node.
2. Set the min-pointer to the next root in the root list.
3. Create an array of size equal to the maximum degree of the trees in the heap before deletion.
4. Do the following (steps 5-7) until there are no multiple roots with the same degree.
5. Map the degree of current root (min-pointer) to the degree in the array.
6. Map the degree of next root to the degree in array.

7. If there are more than two mappings for the same degree, then apply union operation to those roots such that the min-heap property is maintained (i.e. the minimum is at the root).

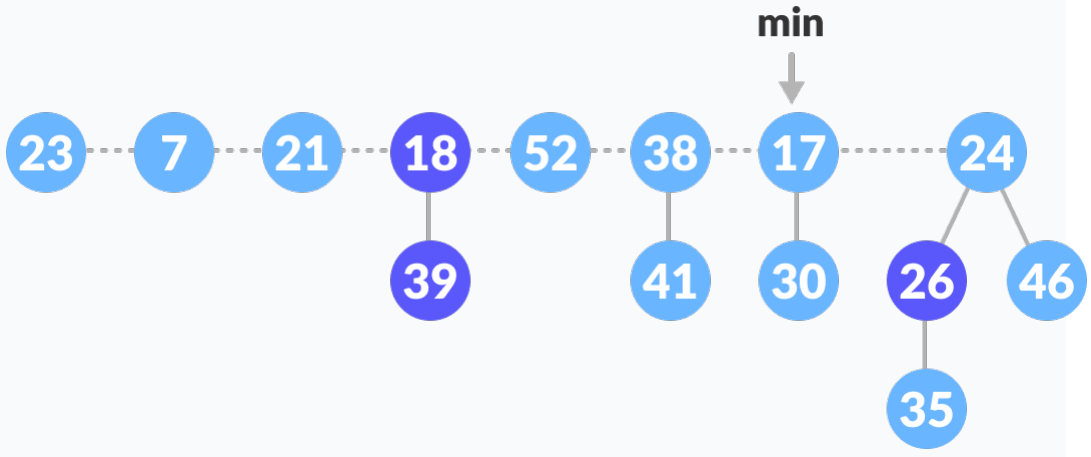
An implementation of the above steps can be understood in the example below.

1. We will perform an extract-min operation on the heap below.



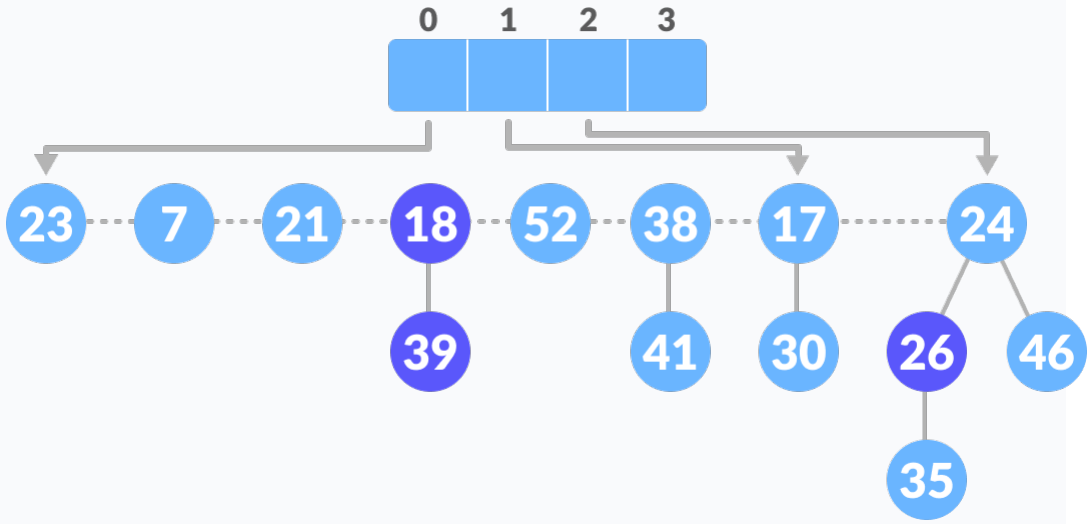
Fibonacci Heap

2. Delete the min node, add all its child nodes to the root list and set the min-pointer to the next root in the root list.



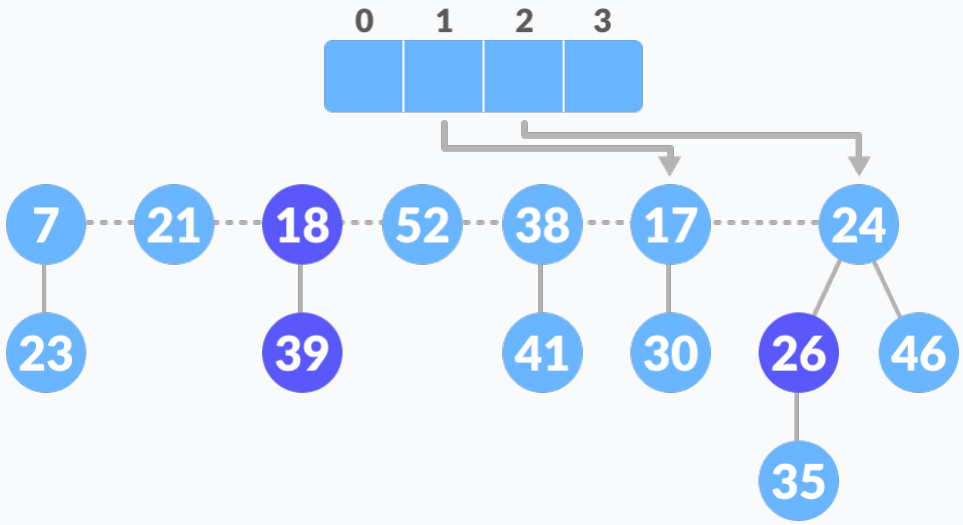
Delete the min node

3. The maximum degree in the tree is 3. Create an array of size 4 and map degree of the next roots with the array.



Create an array

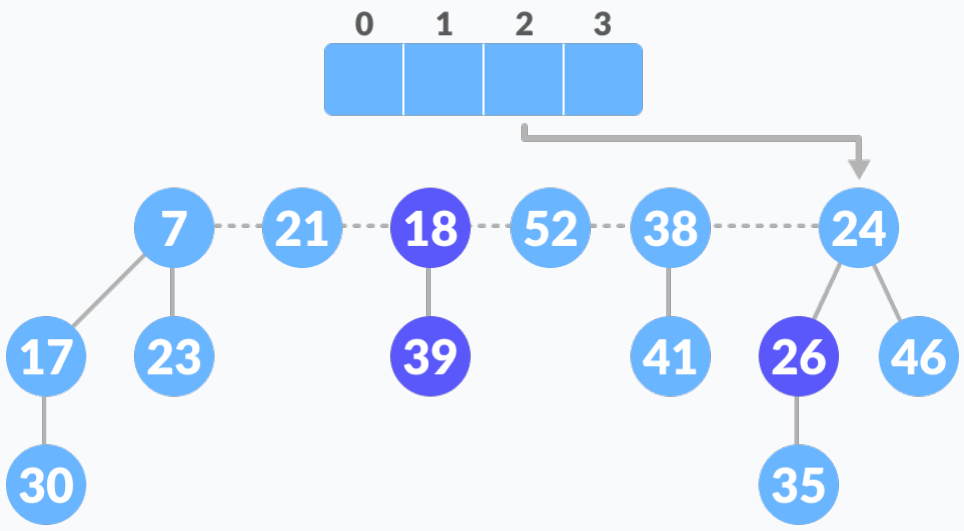
4. Here, 23 and 7 have the same degrees, so unite them.



Uni

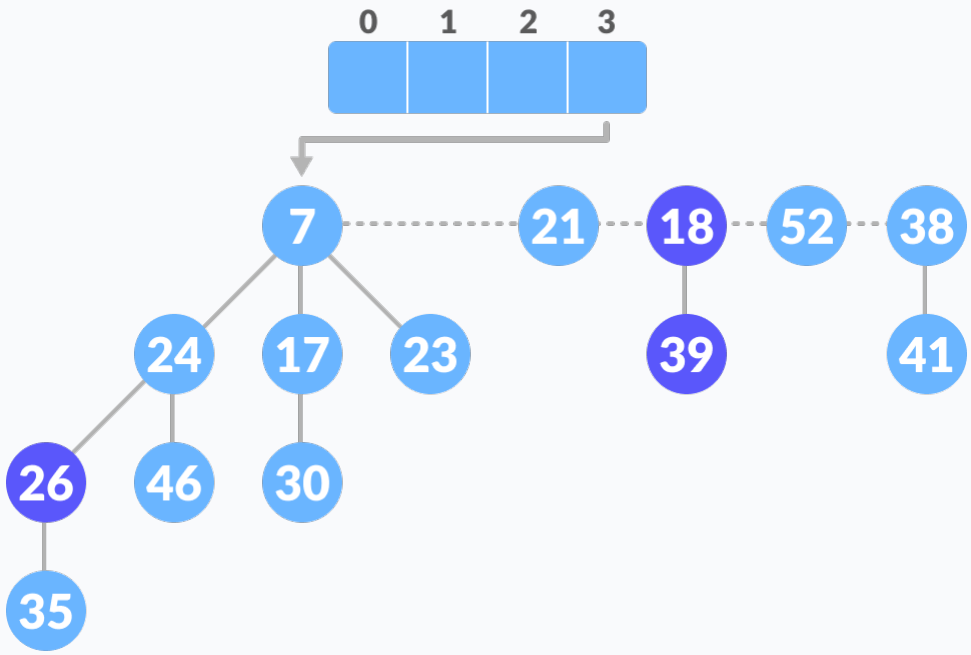
te those having the same degrees

5. Again, 7 and 17 have the same degrees, so unite them as well.



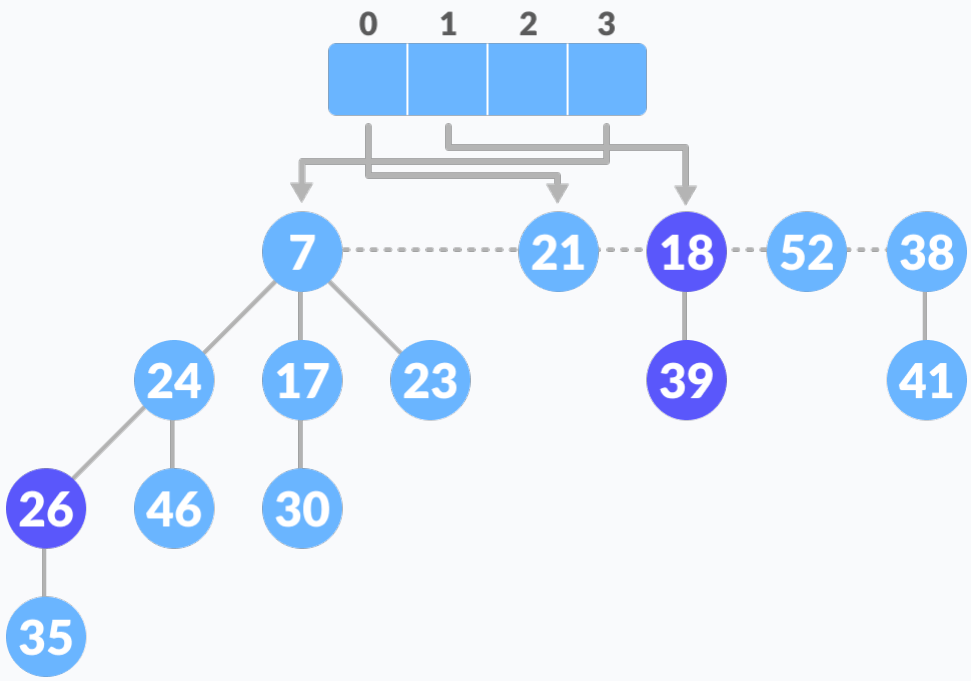
Unite those having the same degrees

6. Again 7 and 24 have the same degree, so unite them.



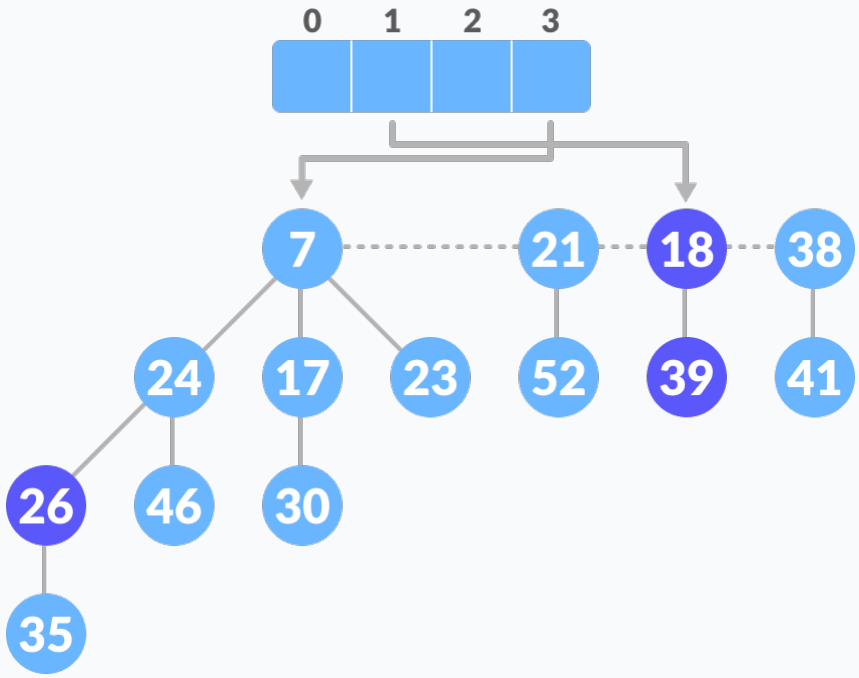
Unite those having the same degrees

7. Map the next nodes.



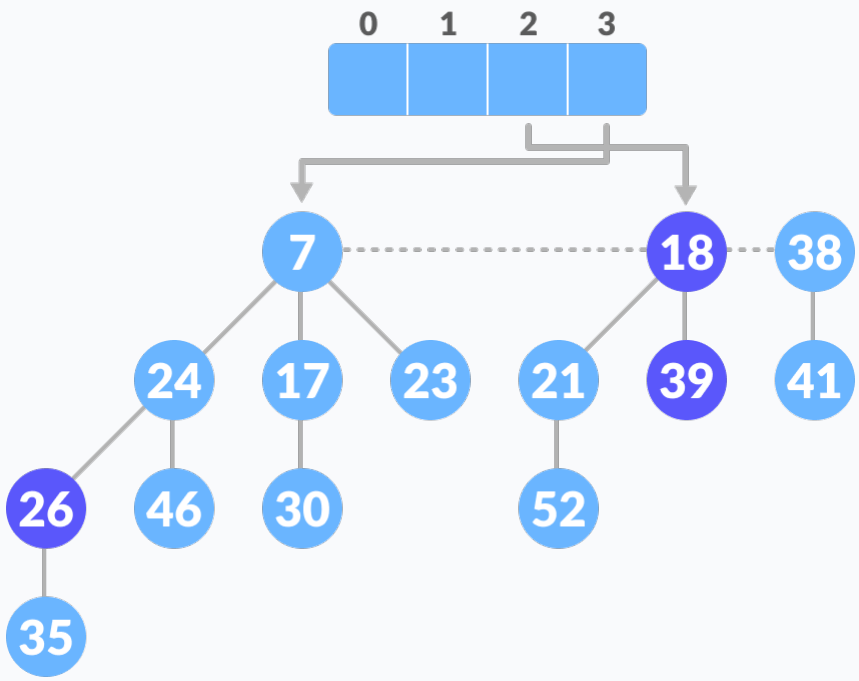
Map the remaining nodes

8. Again, 52 and 21 have the same degree, so unite them



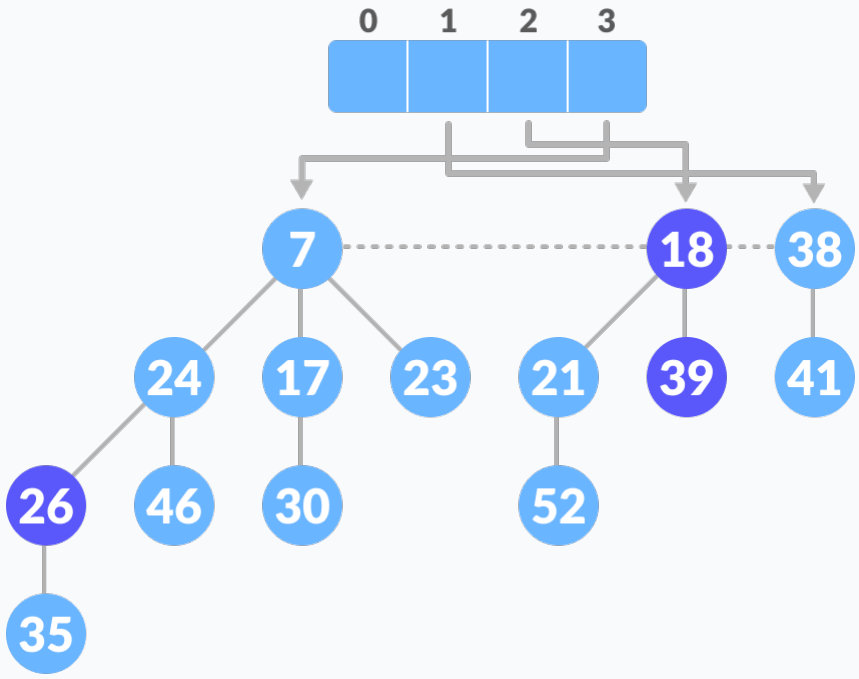
Unite those having the same degrees

9. Similarly, unite 21 and 18.



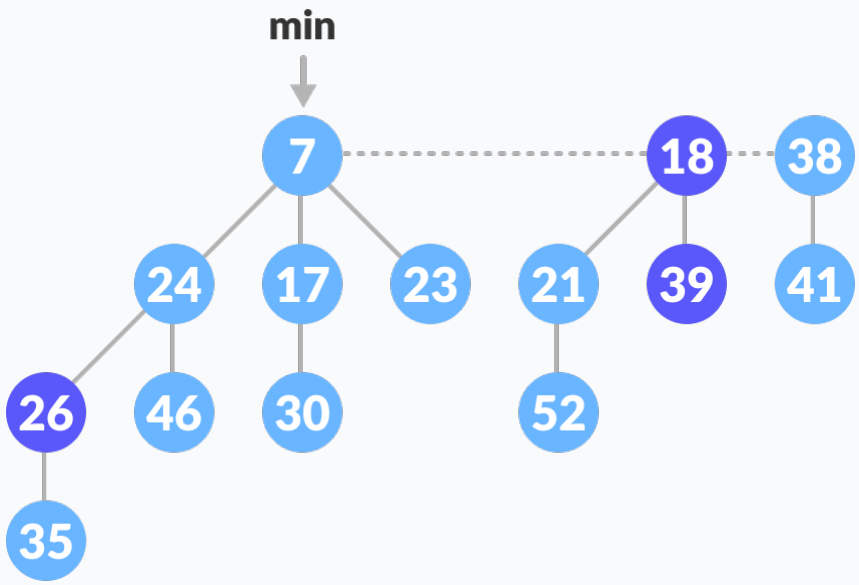
Unite those having the same degrees

10. Map the remaining root.



Map the remaining nodes

11. The final heap is.



Final fibonacci heap

Decreasing a Key and Deleting a Node

ollowing functions are used for decreasing the key.

Decrease-Key

1. Select the node to be decreased, `x`, and change its value to the new value `k`.
2. If the parent of `x`, `y`, is not null and the key of parent is greater than that of the `k` then call `Cut(x)` and `Cascading-Cut(y)` subsequently.
3. If the key of `x` is smaller than the key of min, then mark `x` as min.

Cut

1. Remove `x` from the current position and add it to the root list.
2. If `x` is marked, then mark it as false.

Cascading-Cut

1. If the parent of `y` is not null then follow the following steps.

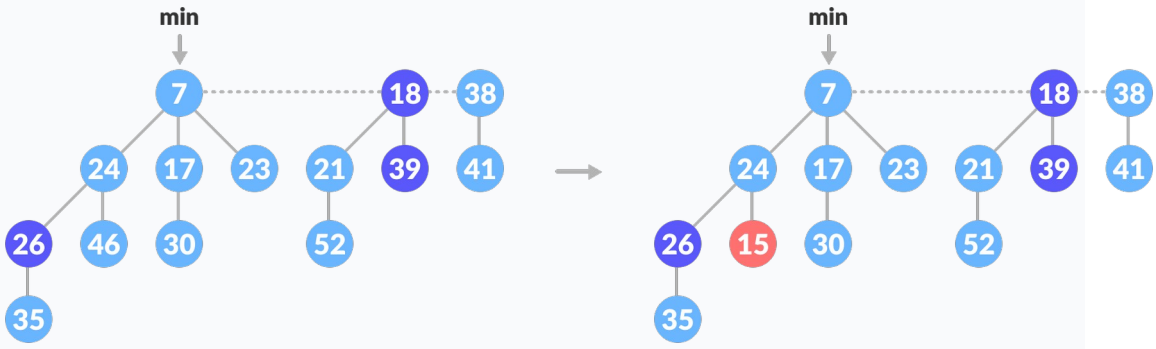
2. If `y` is unmarked, then mark `y`.
3. Else, call `Cut(y)` and `Cascading-Cut(parent of y)`.

Decrease Key Example

The above operations can be understood in the examples below.

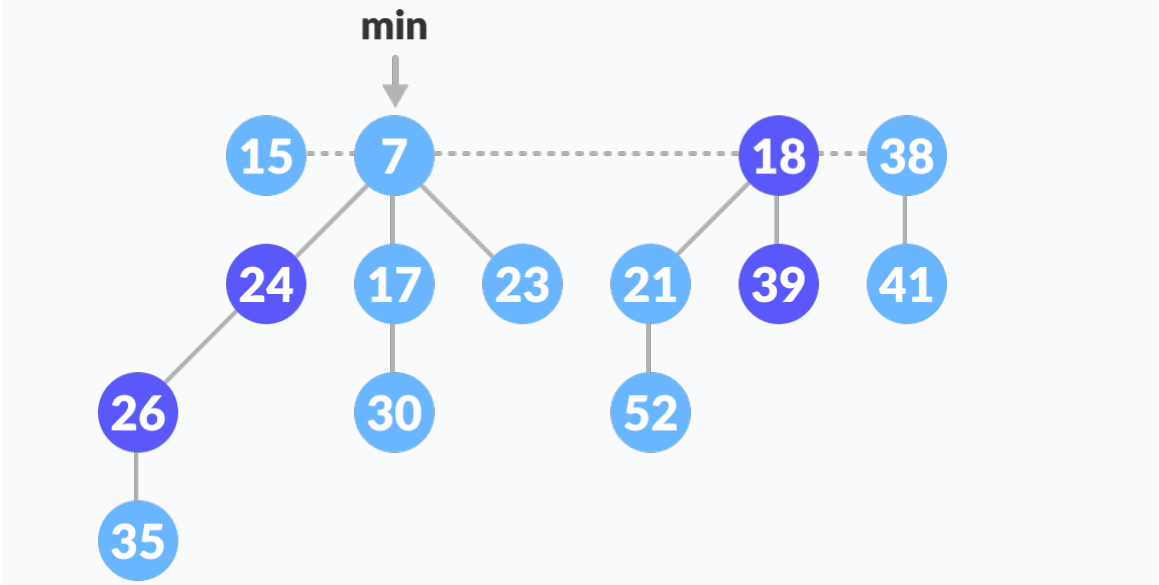
Example: Decreasing 46 to 15.

1. Decrease the value 46 to 15.



   Decrease 46 to 15

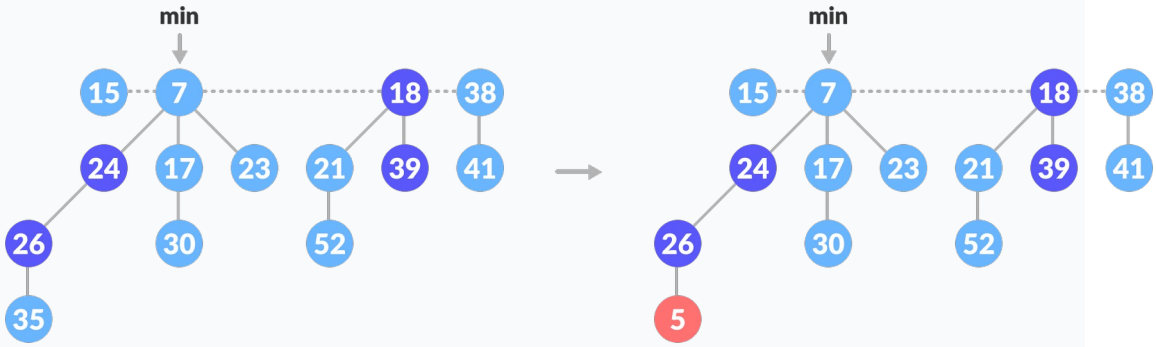
2. **Cut part:** Since `24 ≠ nill` and `15 < its parent`, cut it and add it to the root list. **Cascading-Cut part:** mark 24.



Add 15 to root list and mark 24

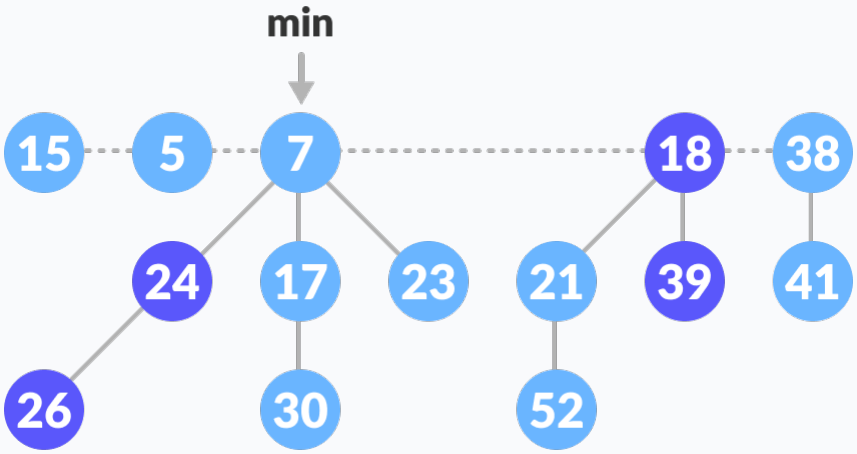
Example: Decreasing 35 to 5

1. Decrease the value 35 to 5.



Decrease 35 to 5

2. Cut part: Since `26 ≠ nill` and `5<its parent`, cut it and add it to the

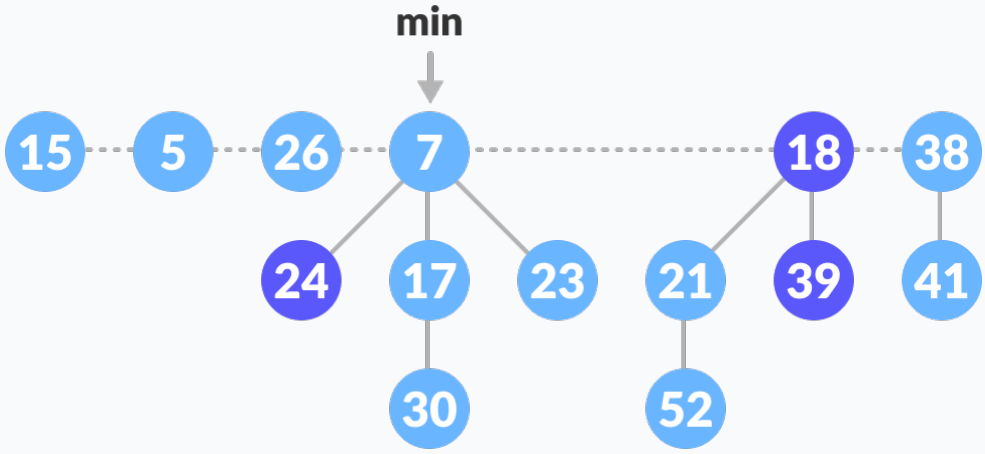


root list.

Cut 5 and add it to root list

3. Cascading-Cut part: Since 26 is marked, the flow goes to `Cut` and `Cascading-Cut`.

Cut(26): Cut 26 and add it to the root list and mark it as false.

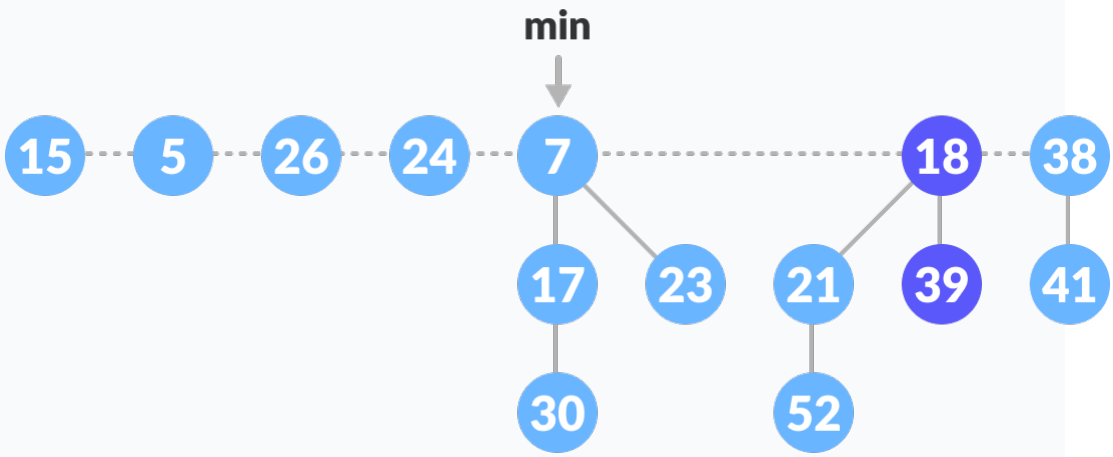


C

ut 26 and add it to root list

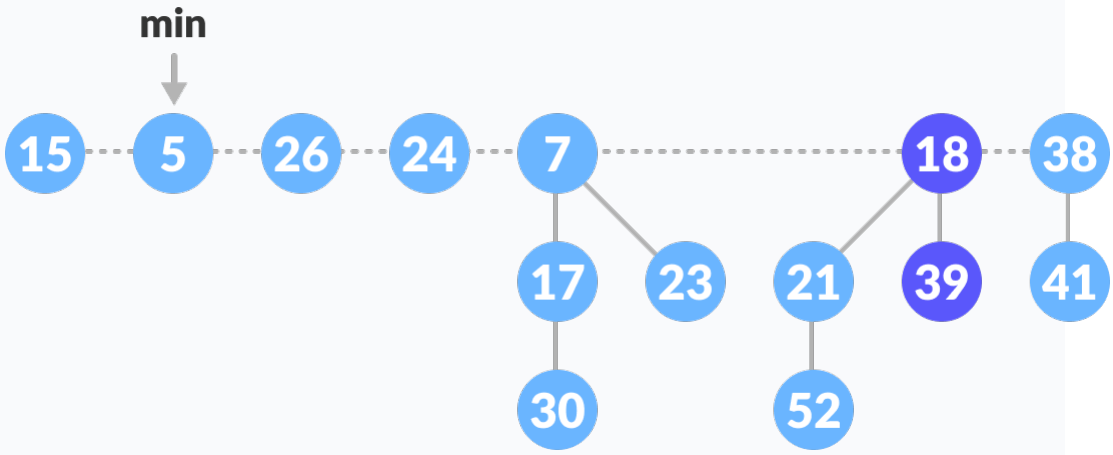
Cascading-Cut(24):

Since the 24 is also marked, again call `Cut(24)` and `Cascading-Cut(7)`. These operations result in the tree below.



Cut 24 and add it to root list

4. Since `5 < 7`, mark 5 as min.



Mark 5 as min

Deleting a Node

This process makes use of decrease-key and extract-min operations. The following steps are followed for deleting a node.

1. Let `k` be the node to be deleted.

2. Apply decrease-key operation to decrease the value of `k` to the lowest possible value (i.e. -∞).
3. Apply extract-min operation to remove this node.