Fibonacci Heap

A Fibonacci Heap is a collection of trees that satisfies the min-heap property.

→ A fibonacci Heap is a specialized, flexible collection of Heap ordered trees used for priority queue operations.

Insert operation in Fibonacci Heap :-

Algorithm :-

* create a new Fibonacci Heap H containing only X.
* set x.left = x and x.right = x (circular doubly linked list)
* if H is Empty, set H.min = x.
* otherwise, insert x into the root list of H and update H.min if x.key < H.min.key
* Increase the total node count of H.

Ex:- Insert elements [23, 7, 3, 17, 24, 18, 52, 38, 30, 26, 46, 39, 41, 35] in Fibonacci Heap.

solution :-

step-1 :- insert 23

H is empty.

H.min = 23

min = 23 updated

step-2 insert 7

7 < 23 → H.min = 7

min = 7 updated

step-3 :- insert 3

3 < 7

↓

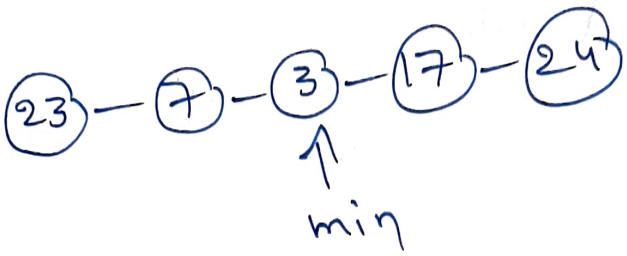
H.min = 3

min = 3 updated

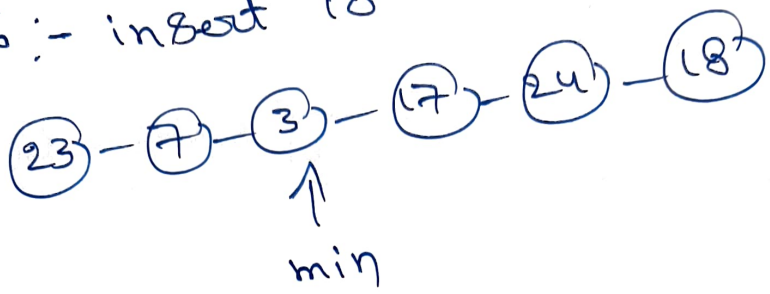
step 4 :- insert 17

min

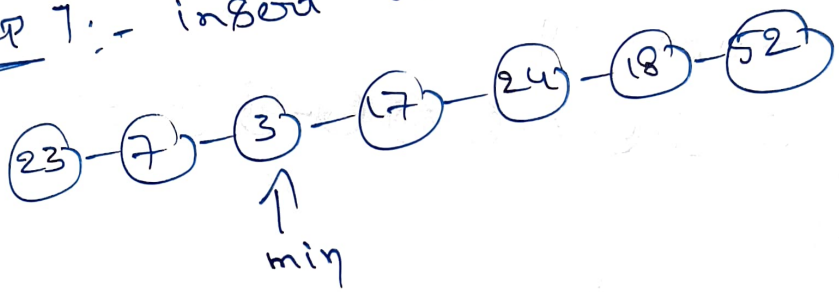
step 5:- insert 24



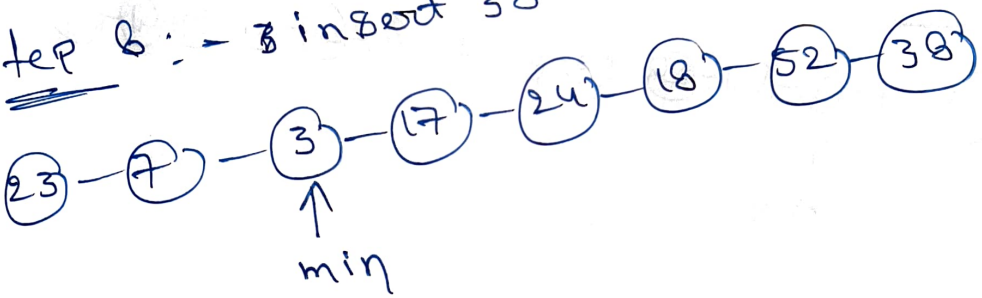
step 6:- insert 18



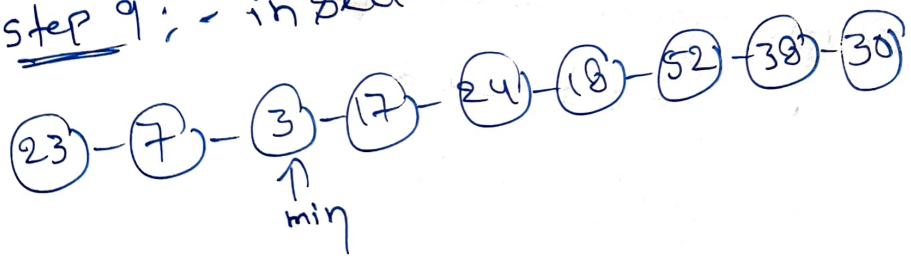
step 7:- insert 52



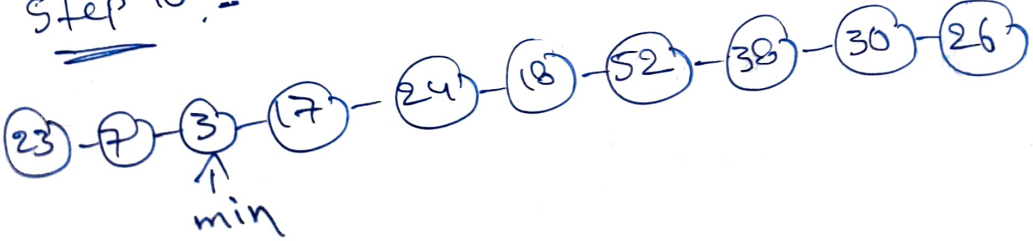
step 8:- insert 38



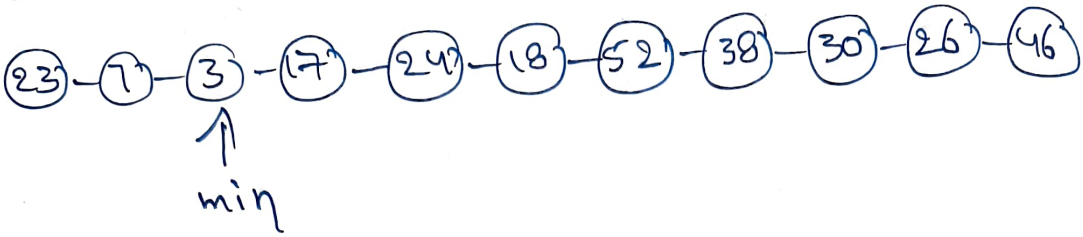
step 9:- insert 30



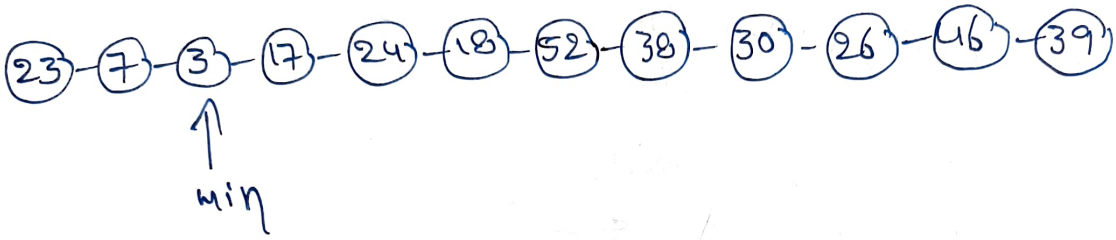
step 10:- insert 26



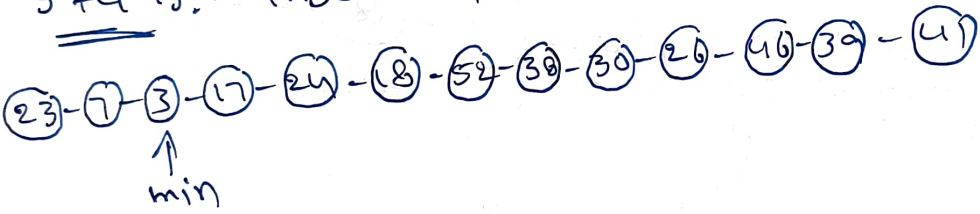
Step 11:- insert 46



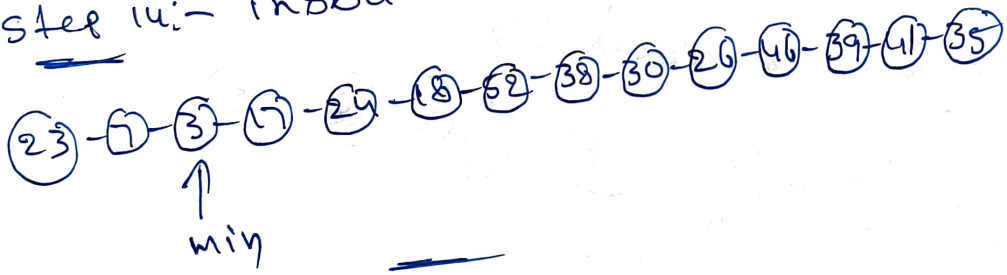
Step 12:- insert 39.



Step 13:- insert 41



Step 14:- insert 35



NOTE:

→ All nodes are single-node trees

→ No parent-child links yet.

→ Fibonacci heap does not link nodes during insertion

→ Each insert takes O(1) time

→ Trees form only during Extract-min

2 Extract-Min operation in Fibonacci Heap

It removes the node containing the minimum key from the Heap H.

Rules :-

* Locate the minimum key node in the Heap.
* if the node Heap is empty, stop.
* Move all children of the minimum node to the root list and remove parent links.
* Delete the minimum node from the root list.
* if the Heap becomes empty, set the minimum pointer to all.
* otherwise, select a temporary minimum from remaining roots.
* perform consolidation to merge trees of Equal degree.
* update the minimum pointer after Consolidation
* Decrease the total node count by one
* Return the removed minimum node.

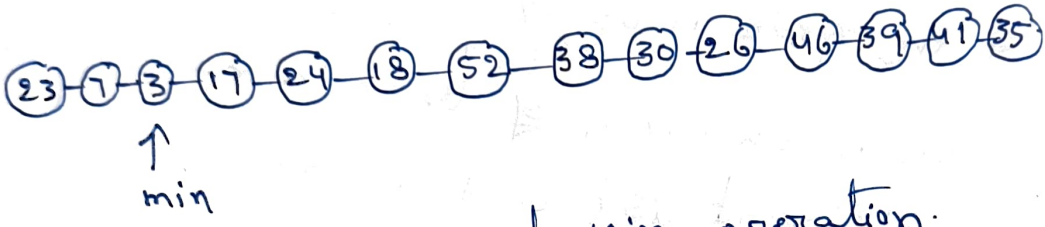
Ex:- Implement Extract-min of the following Elements

[23, 7, 3, 17, 24, 18, 52, 38, 30, 26, 46, 39, 41, 35]

in Fibonacci Heap.

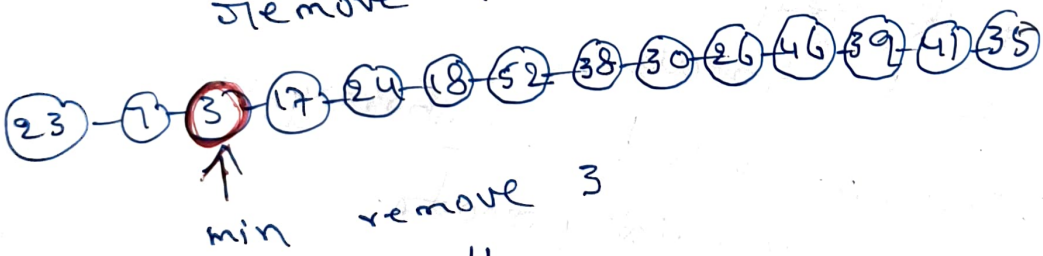
Solution:-

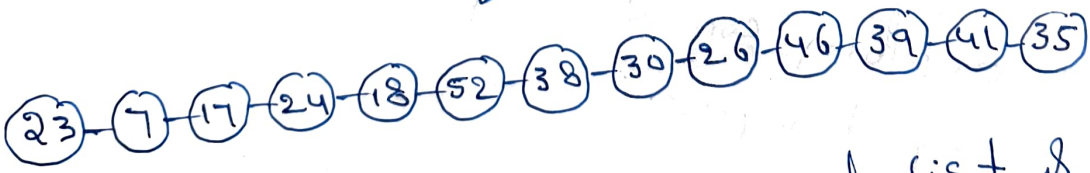
After Insation



Step-1 :- Apply Extract-min operation.

→ Identify the minimum node and remove it.





→ Add children of 3 to root list.

At this moment, node 3 has no children because no Extract-min operation was done earlier.

→ So nothing is added to the root list.

→ child nodes Appear only after consolidation:—

Step-2 start consolidation

↳ No Two Trees in the root list should have the same degree.

↳ Right now all nodes have degree 0.

Step-3 Link (23) and (7) (degree 0)

Step-4 Link (17) and (24)

Step-5 Link (18) and (52)

Step 6 Link (38) and (30)

Step 7:- Link (26) and (46)

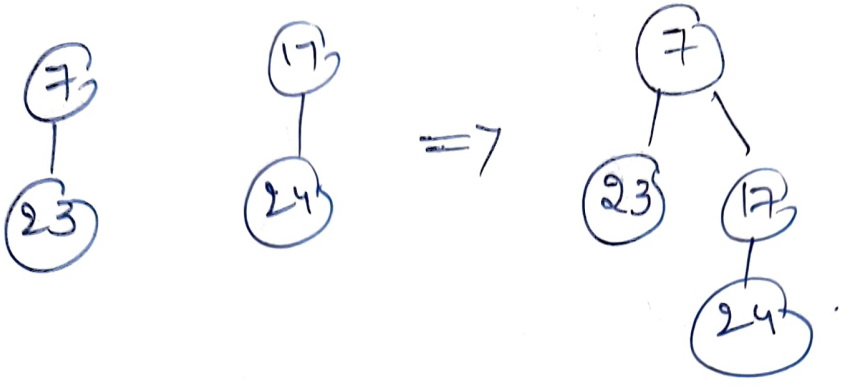
Step-8:- Link (39) and (41)

Step-9:- Node 35 remaing alone

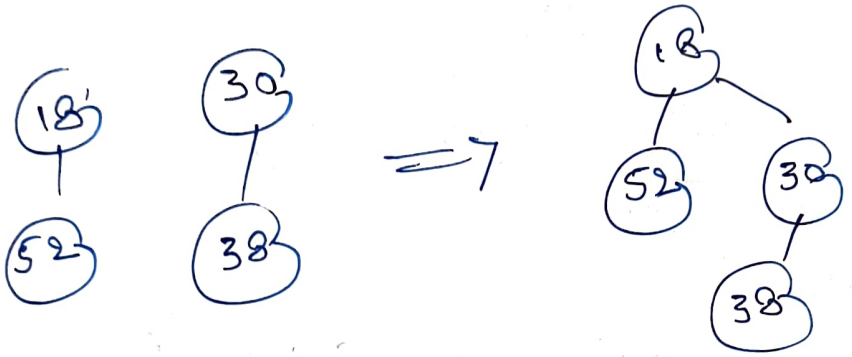
Step-10 Link Trees with same degree

Again

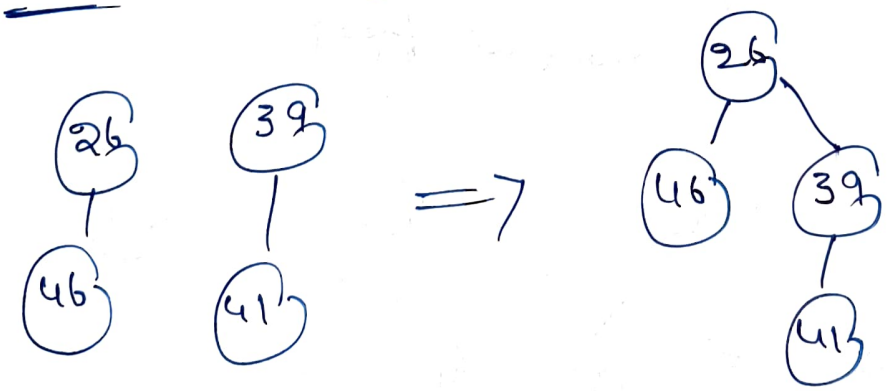
Link (7-23) and (17-24) i.e



Step-11 Link (18-52) and (30-38)



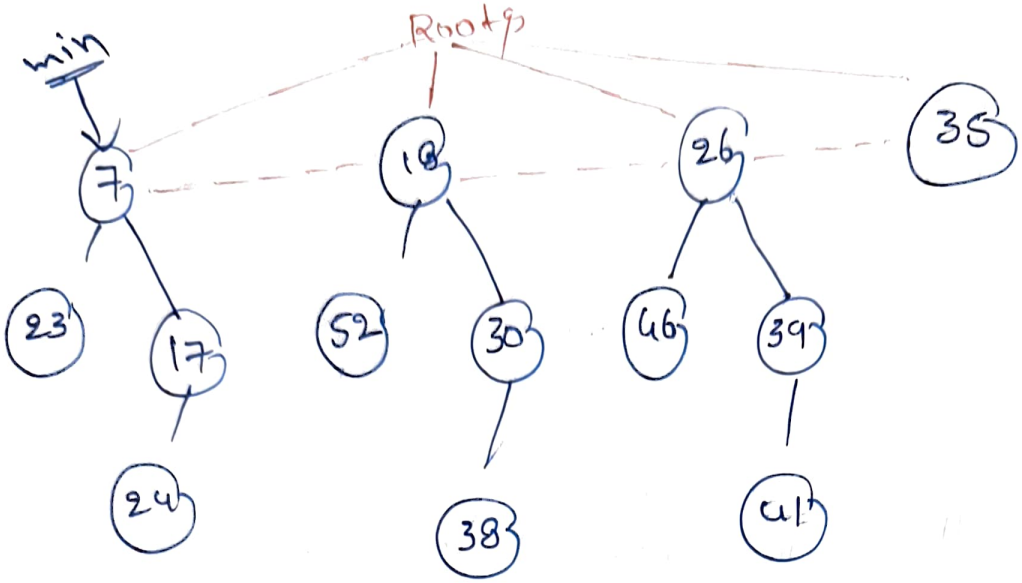
Step-12 Link (26-46) and (39-41)



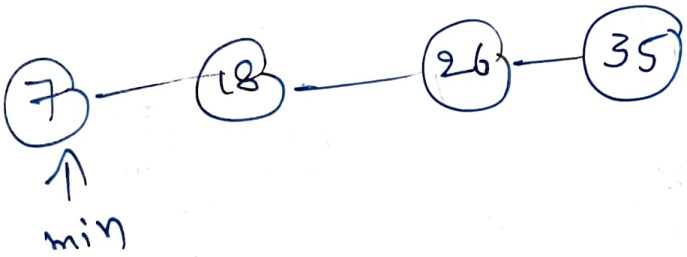
Step-13 : check degrees again.

if no two root have same degree

Stop it.

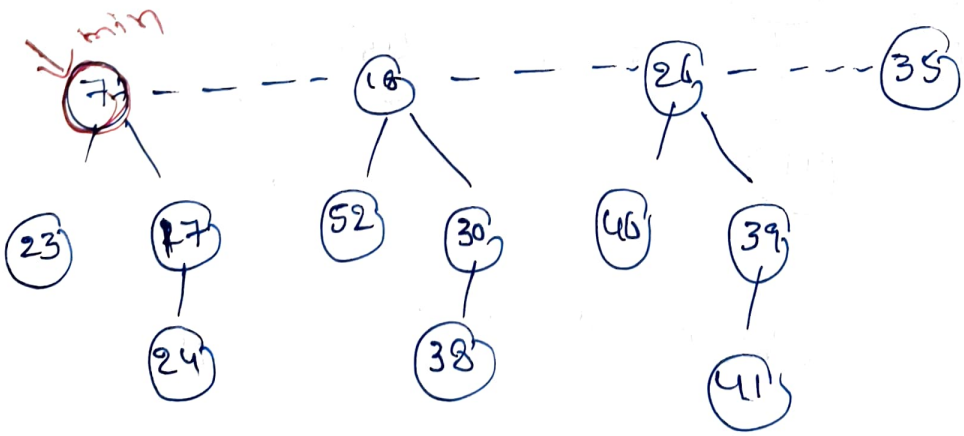


or



Again apply Extract-min operation

↳ Remove smallest node in a fibonacci heap



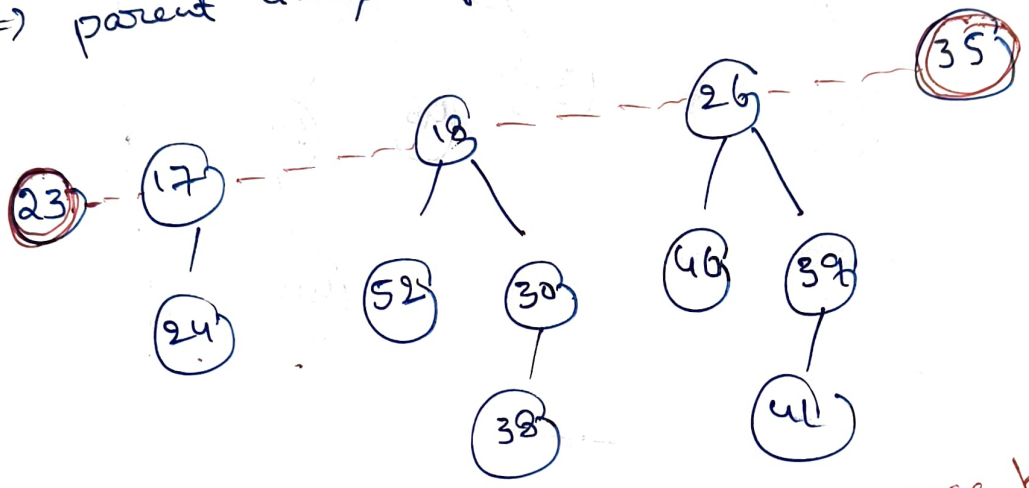
⇓ ⇒ Remove 7

↓

=> Node 7 is Removed

=> it's children 23 and 17 are moved to root list

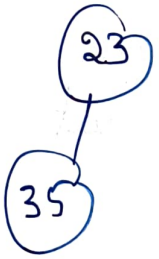
=> parent links of 23 and 17 are removed.



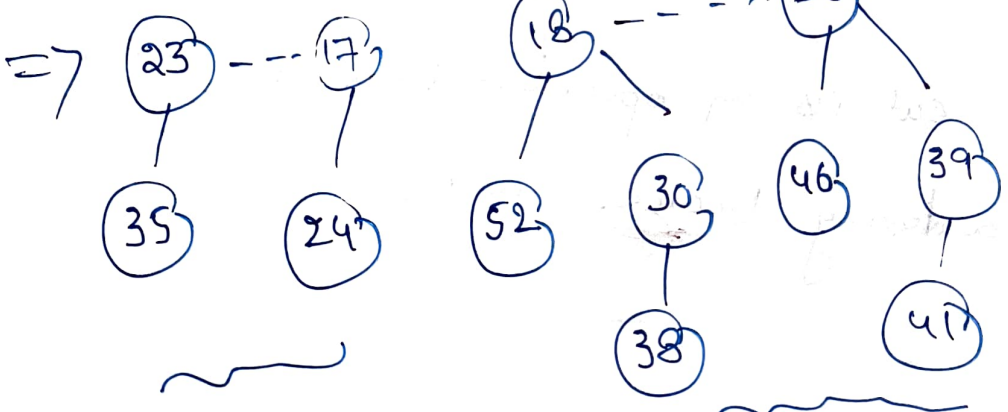
!! Now merge Tree having the same degree

=> Nodes 23 and 35 have degree 0

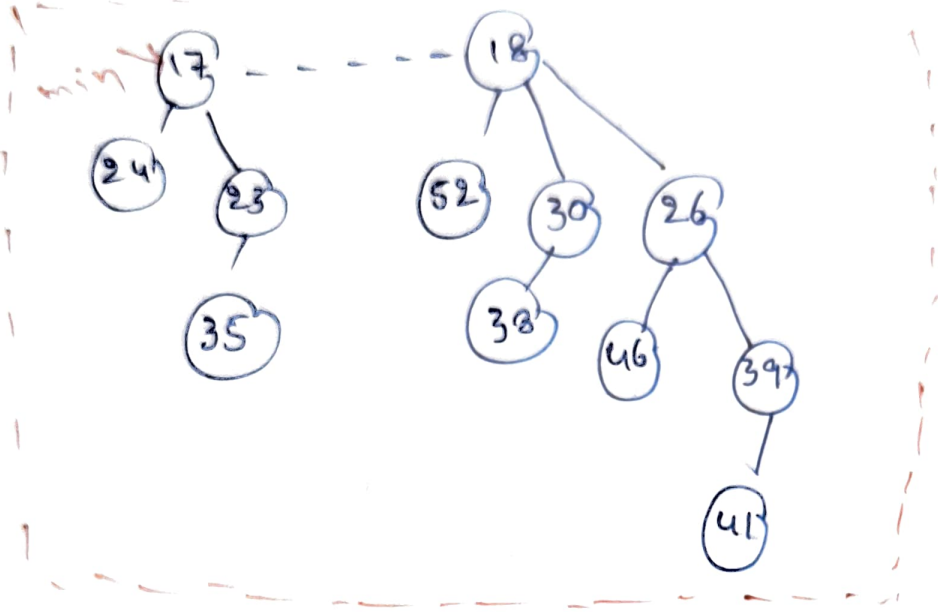
=> smaller key becomes parent -> 23.



=>



Final Fibonacci Heap :-



3 Decrease-key :-

↳ The value of a key is decreased to lower value. If you want perform Decrease-key operation, we need to perform -

* cut
* cascading cut

cut :-

cut is an operation in a fibonacci heap used during Decrease-key.

cut(x): removes a node x from it's parent when it's key becomes smaller than it's parent key and adds x to the root list of the heap.

→ The mark of x is reset to False.

→ why cut operation means that

* Because heap-order property is violated
* Child becomes smaller than parent.

Cascading cut :—

Cascading cut(y) is the repeated cutting of parent nodes when a node loes more than one child. if a parent is already marked and loses another child, it is cut and moved to the root list and the process continues upward.

→ why cascading cut operation means that to prevent trees from becomming deep.

NOTE :-

* cut removes a node whose decreased key violates heap order and moves it to the root list.

* cascading cut repeatedly cuts marked parent nodes until an unmarked node (or) root is reached.

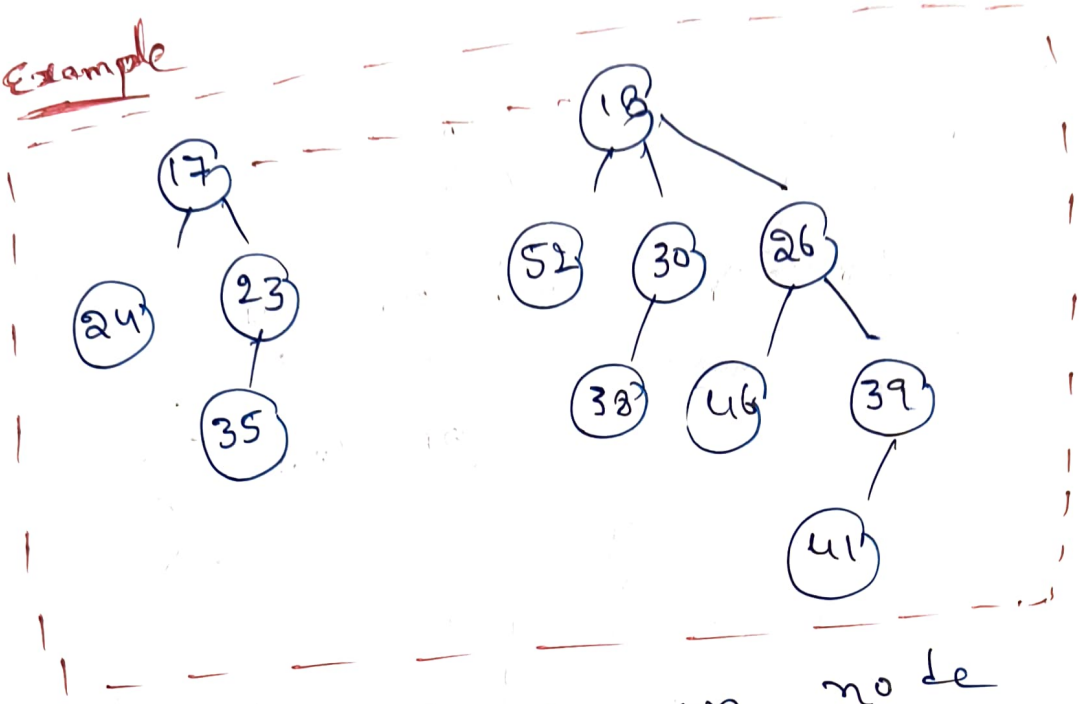
* A marked node in a fibonacci Heap is a node that has already lost one child since that last time it became a child of Another node.
  - ↳ mark = warning signal
  - ↳ if I lose one more child, I will be cut.

---

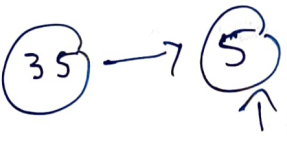
* An un-marked node is a node that has not lost any child, or has lost only one child and has not yet been marked before.

* Lost 0 children — unmarked
  Lost 1 child — marked
  Lost 2 children — cut

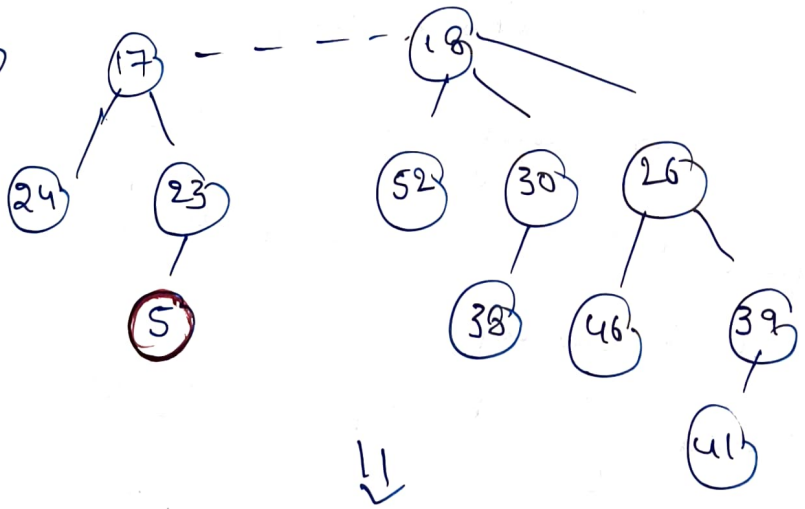
Example



→ Apply Decrease-key on node



⇒



!!

5 < 23 (Violated)

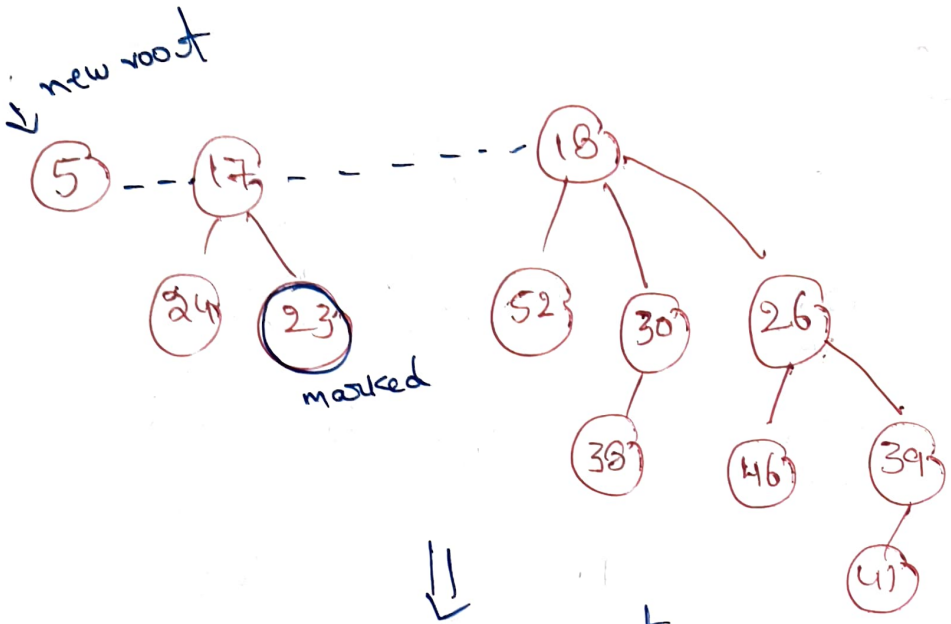
→ cut operation required. So, cut 5

means

↳ Remove 5 From children of 23

↳ Add 5 to the root list.

↳ mark parent 23



Now check cascading cut

→ Now look at parent of 23, which is 17

→ 23 already marked

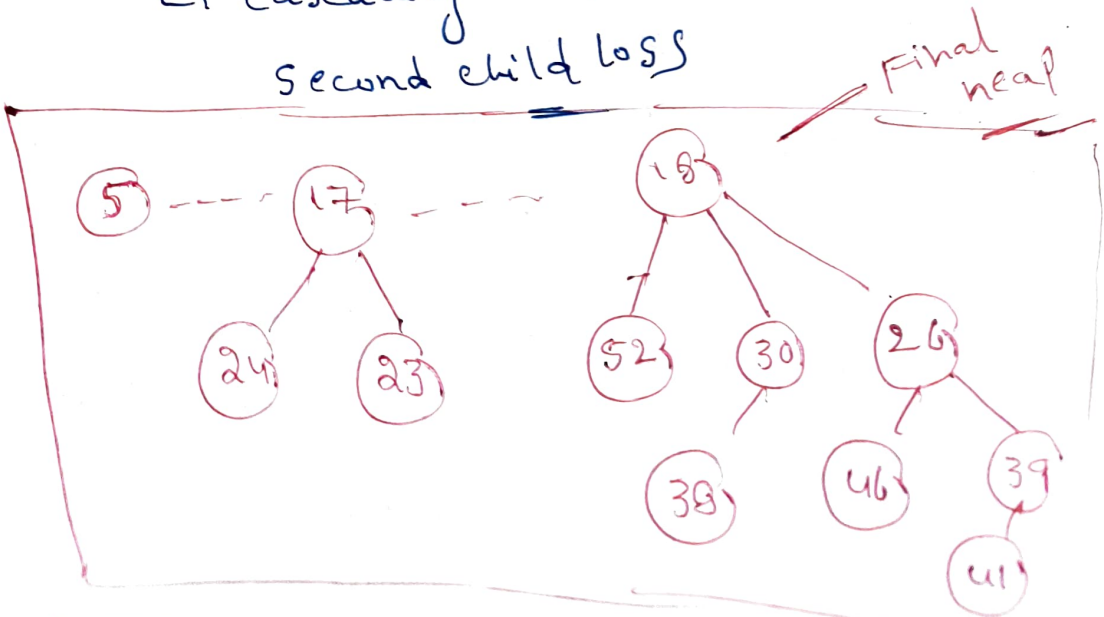
→ Stop cascading.

→ No further cuts happen

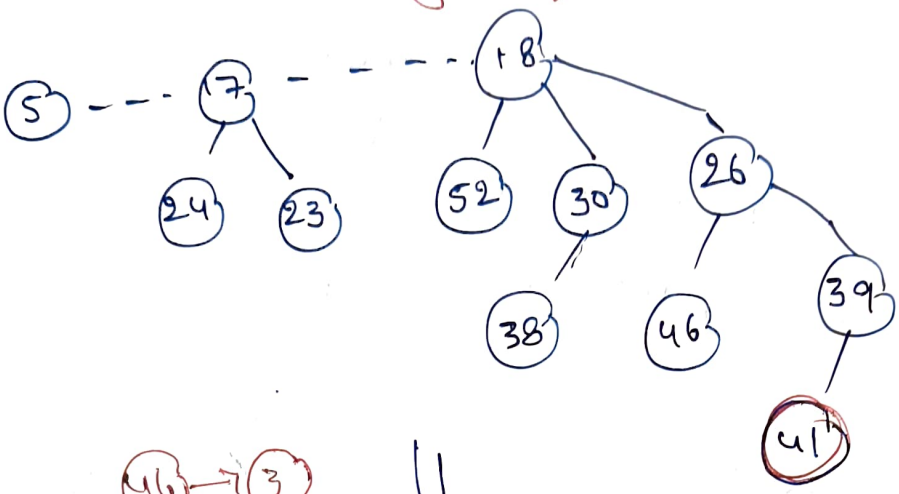
→ why no cascading cut here

↳ parent (17) had not lost a child earlier

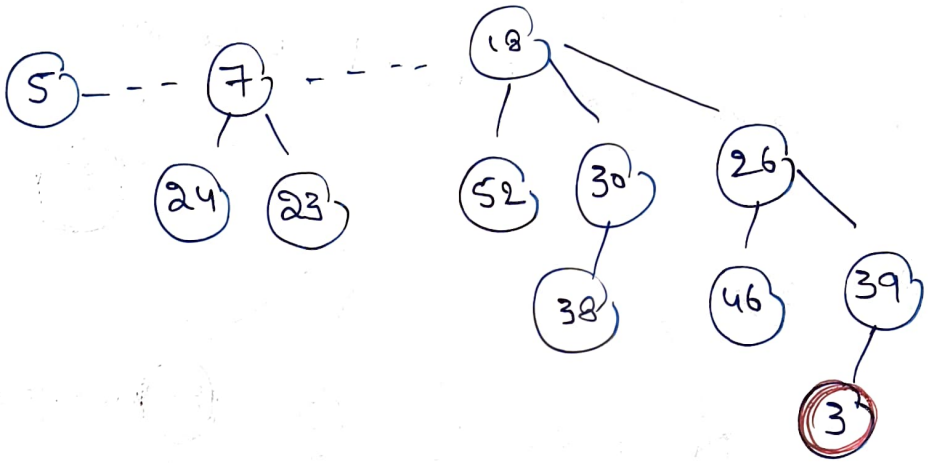
↳ cascading cuts start only after second child loss



EX:- Decrease-key on node 41 → 3



46 → 3



⇒ 3 9 7 3 violated

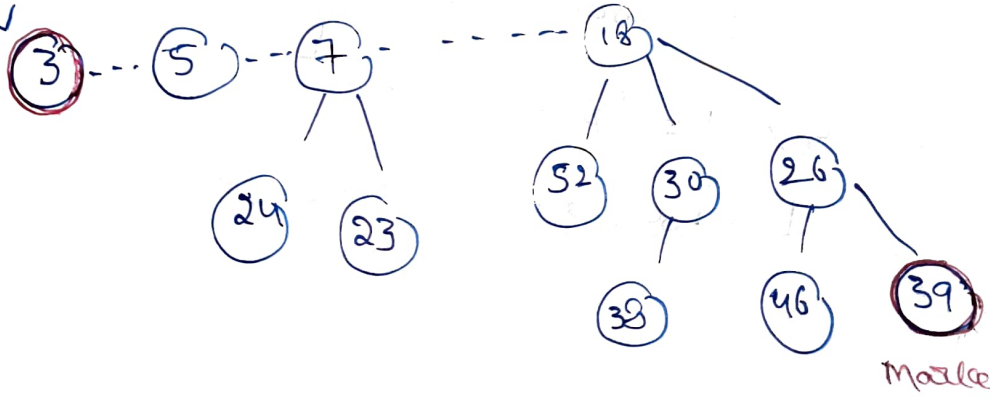
↳ so cut operation required.

↳ remove 3 from 39 node

↳ add node 3 to root list

↳ mark 39 (first child loss)

newroot



Now check cascading cut 39

↳ 39 is now marked

↳ 26 is not root

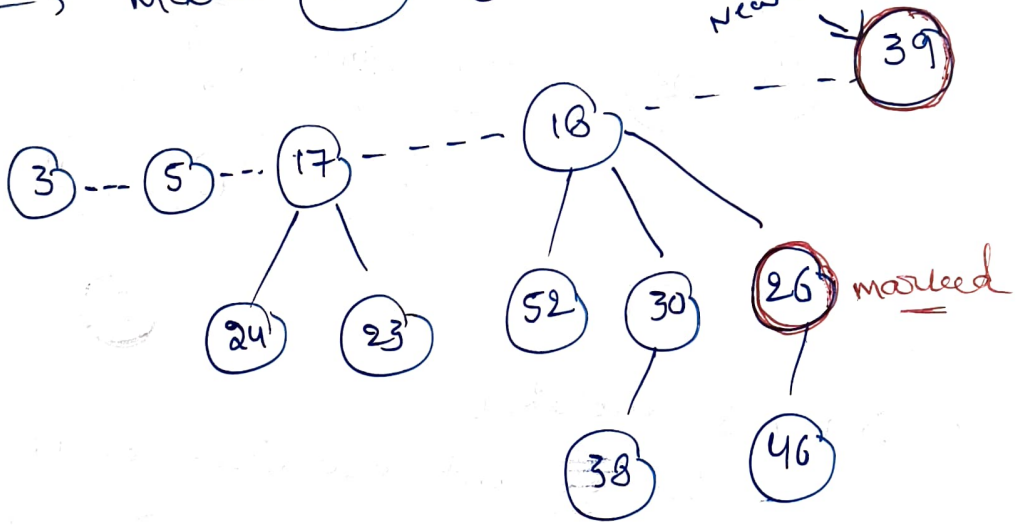
↳ 26 loses a child

↳ so cut 39

→ Remove (39) from (26)

→ Add (39) node to root list

→ Mark (26) { First child loss }



Now check cascading cut 26

↳ 26 was just marked

↳ 18 is not root

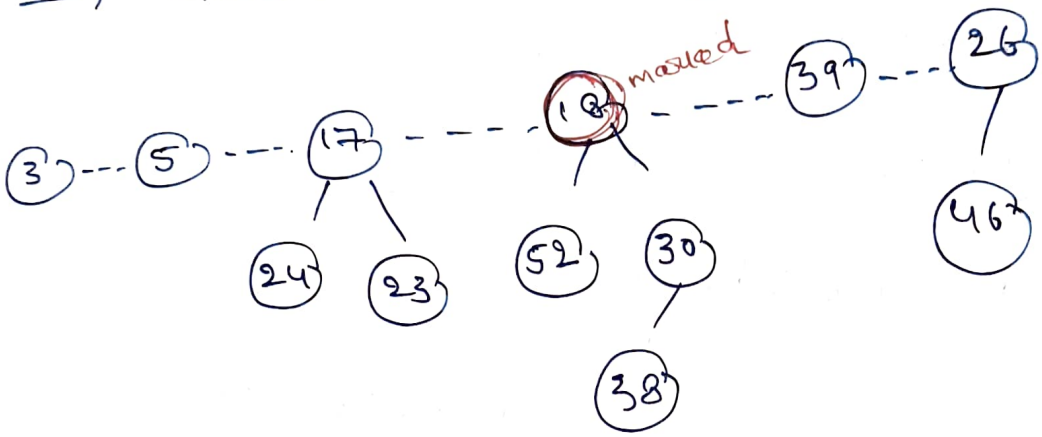
↳ Lossing a marked child

↳ so cut 26

→ Remove (26) from children of (18)

→ Add (26) node to root list.

→ mark node (18) (first child loss)

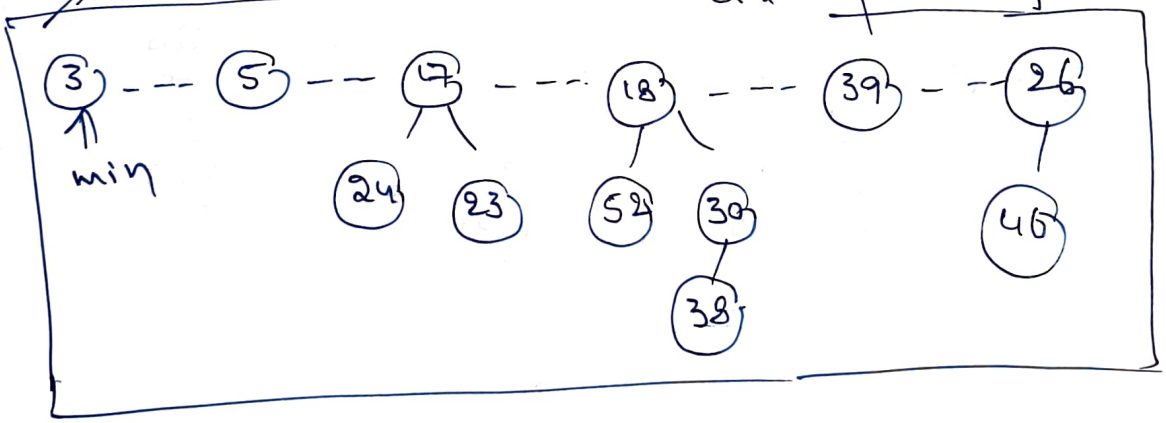


Now check cascading cut

parent of 18 = NULL (root)

so stop cascading cut operation.

Final heap



NOTE:-

Node cut	Reason
3 →	violated heap order (3<39)
39 →	was marked and not child
26 →	was marked "
stop at 18 →	Root nodes are never cut.

4 Delete operation in Fibonacci Heap :

Delete(x) in a fibonacci Heap is performed in two main phases.

(i) Decrease-key $(x, -\infty)$

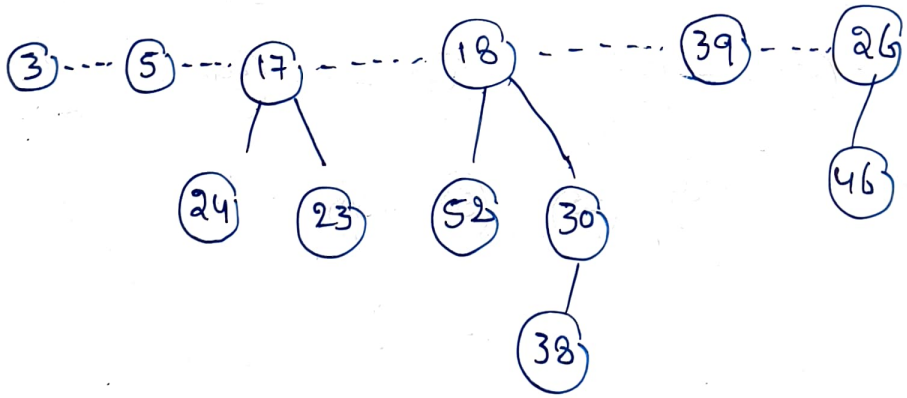
(ii) Extract-min

↳ Remove smallest key from the Heap

$\Rightarrow$

Delete = Decrease-key + Extract-min

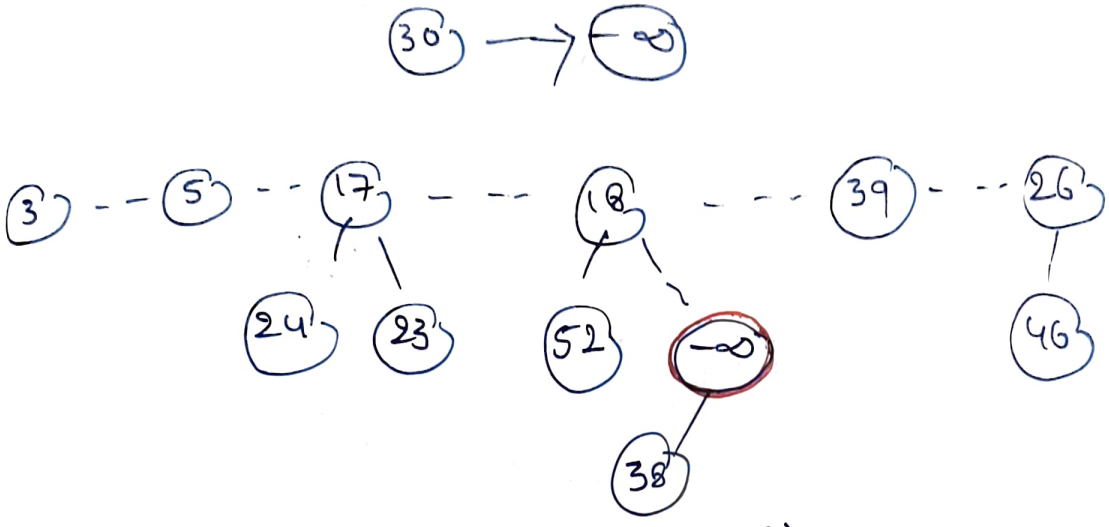
Ex :-



Delete Node (30)

Solution

Step-1 :- Decrease-key operation



-> Now $-\infty$ < it parent (18)

-> cut 30 is performed and

-> (30) is removed from node (18)

-> (30) is added to the root list

└> $(-\infty)$ added to root list.

-> Now check cascading cut operation

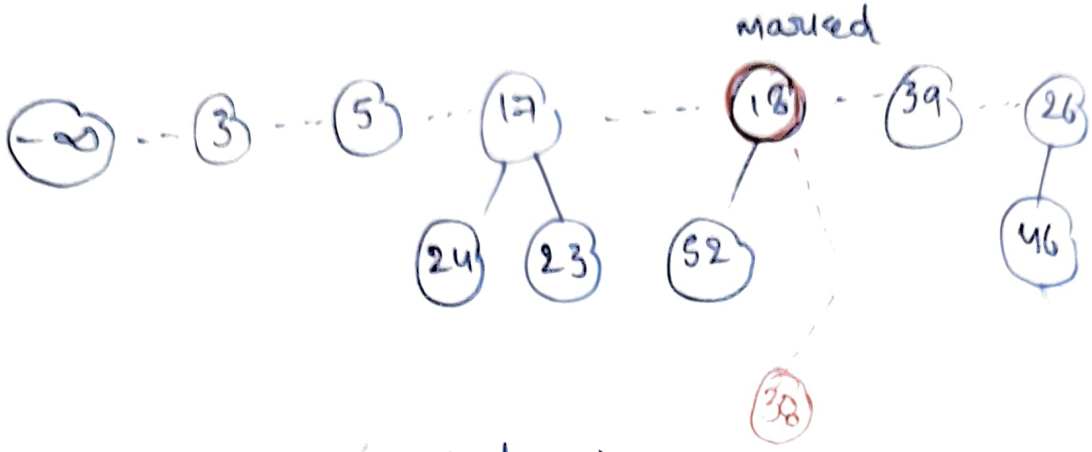
└> parent 18 was allready marked.

└> so (18) loses a second child.

└> (18) is cut and moved to root list.

└> cascading cut stop because 18 is now root.

After Decrease-key



Step-2 Apply Extract-min

↳ Remove minmum node in a heap. i.e $-\infty$.

→ Remove $-\infty$ (30) from the root list

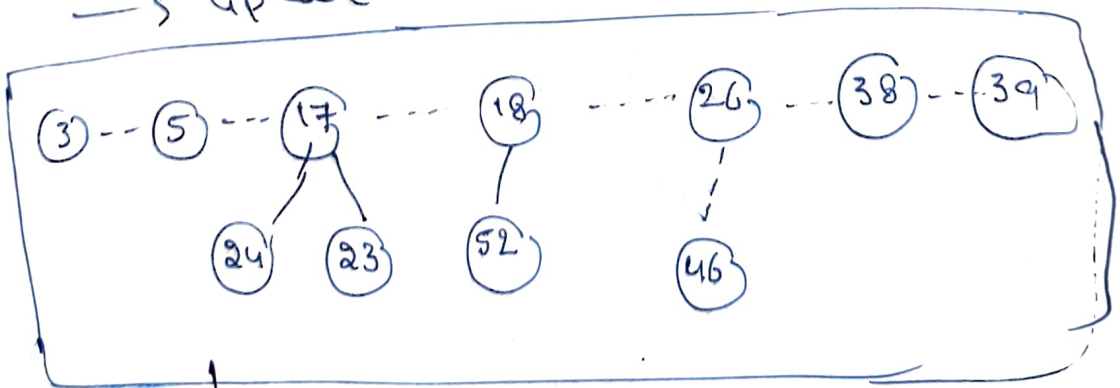
→ Add it's childre to root list

↳ i.e 30 has child 38

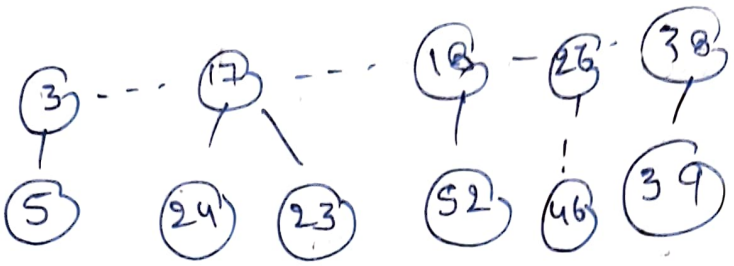
↳ 38 becomes a root

→ perform Consolidation

→ update minmum pointer.

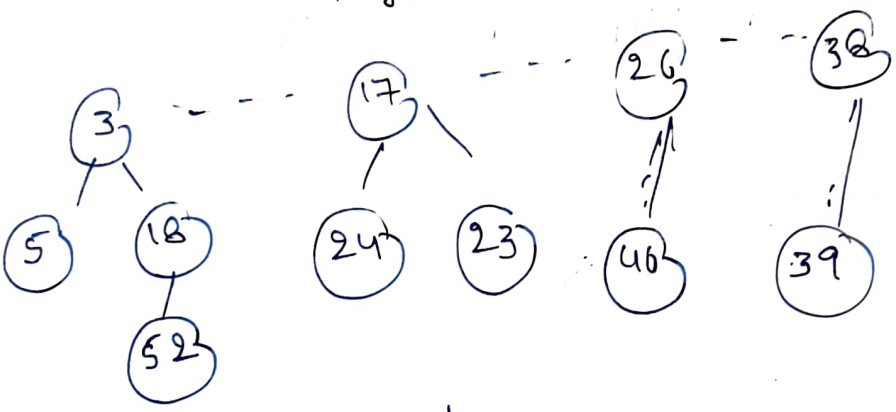


↳ Apply Consolidation to reduce the no. of trees.



degree same

merge root 3 and root 18



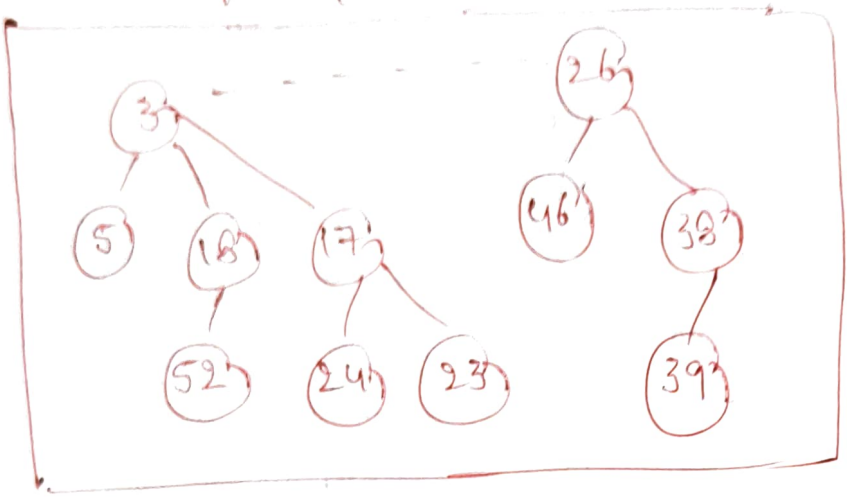
merge root 26 and root 38

↳ degree same

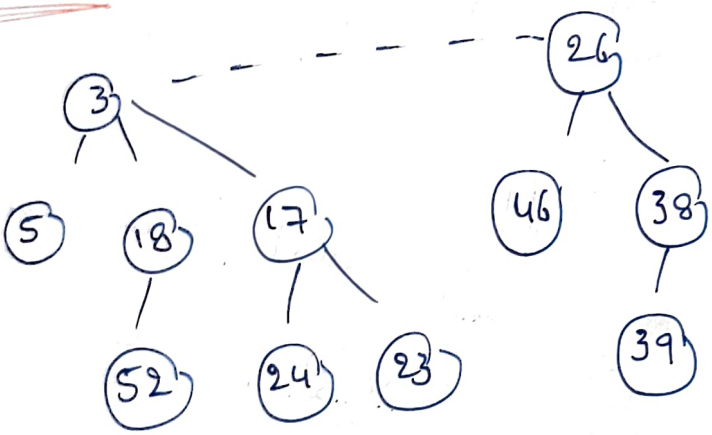
(degree-same) merge root 3 and root ~~17~~

Final heap.

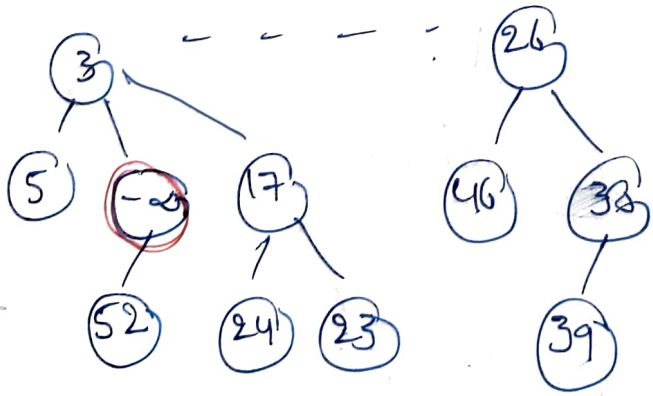
Ex-2:- Delete node 18 of following Fibonacci heap.



Solution :-



Delete 18 so
Apply Decrease-key i.e
18 -> -∞



-∞ < 3
↳ cut happens.

→ Node 18 becomes the smallest

→ Compare with parent 3

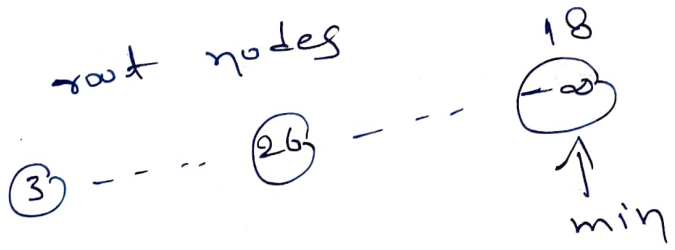
→ $-\infty < 3$

↳ cut operation happen.

→ Remove (18) from children of (3)

→ Add (18) to root list

→ un mark (18) (6) $(-\infty)$



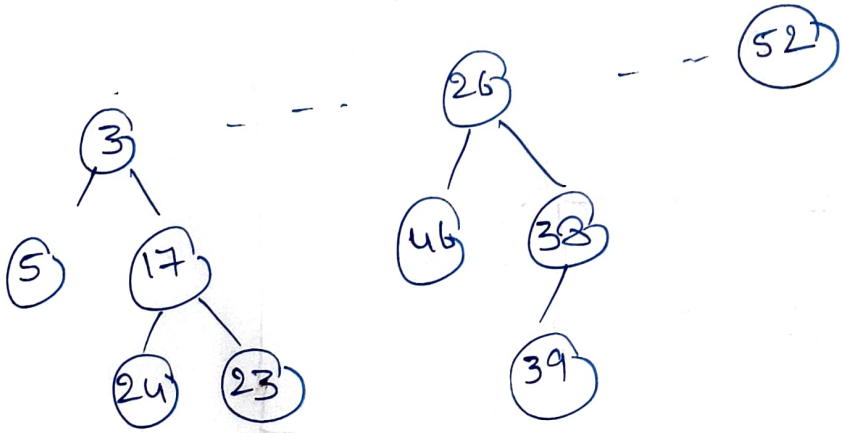
→ Apply Extract min

↳ remove minimum node

$-\infty$ i.e (18) node

↳ children (18) have (52)

↳ add (52) to root list.



→ Apply Consolidating.

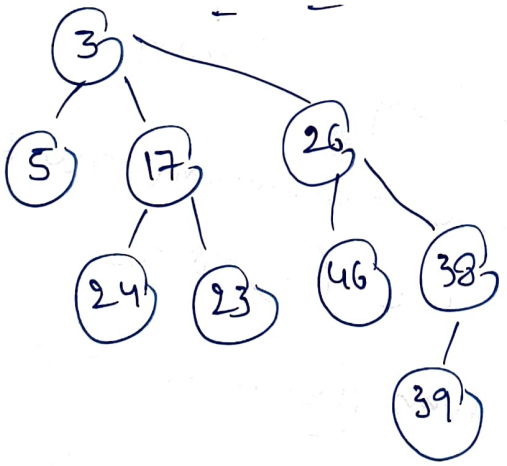
↳ calculate degrees.

root	Degree
3	2 (5,17)
26	2 (46,38)
52	0

→ merge same degree Trees.

↳ merge (3) and (26)

- - - - - (52)



Final heap after delete (18) is