```
6.  remove z from the rootlist of H
7.  if z == z.right
8.      H.min = NIL
9.  else H.min = z.right
10.     CONSOLIDATE(H)
11. H.n = H.n - 1
12. return
```

CONSOLIDATE(H)

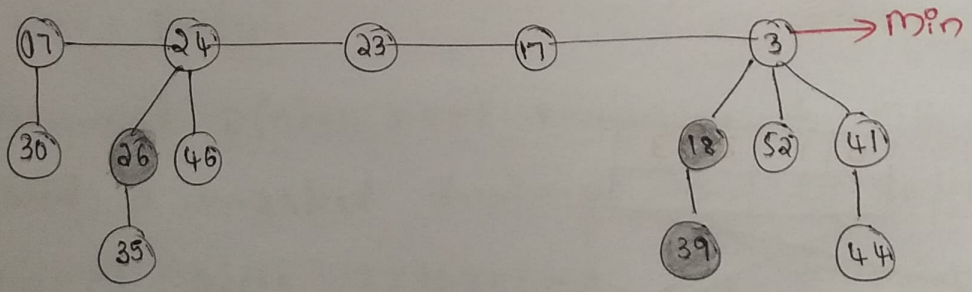
```
1.  let A[0..D(H.n)] be a new array
2.  for i = 0 to D(H.n)
3.      A[i] = NIL
4.  for each node w in the root list of H
5.      x = w
6.      d = x.degree
7.      while A[d] ≠ NIL
8.          y = A[d]        // another node with same degree as x
9.          if x.key > y.key
10.             exchange x with y
11.         FIB-HEAP-LINK(H, y, x)
12.         A[d] = NIL
13.         d = d + 1
14.     A[d] = x
15. H.min = NIL
16. for i = 0 to D(H.n)
```

```
17.     if A[i] ≠ NIL
18.         if H.min == NIL
19.             Create a rootlist for H containing just A[i]
20.             H.min = A[i]
21.         else insert A[i] into H's root list
22.             if A[i].key < H.min.key
23.                 H.min = A[i]
```

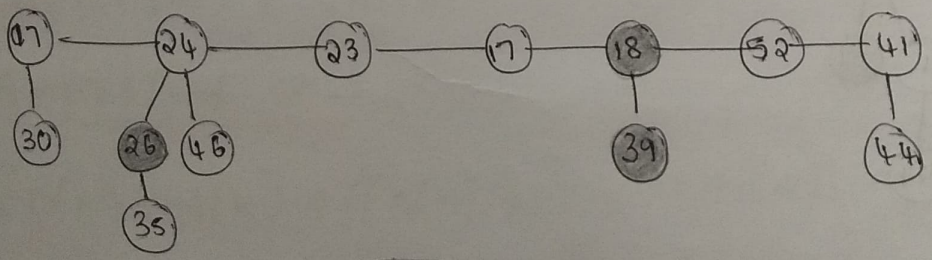
FIB-HEAP-LINK(H, y, x)

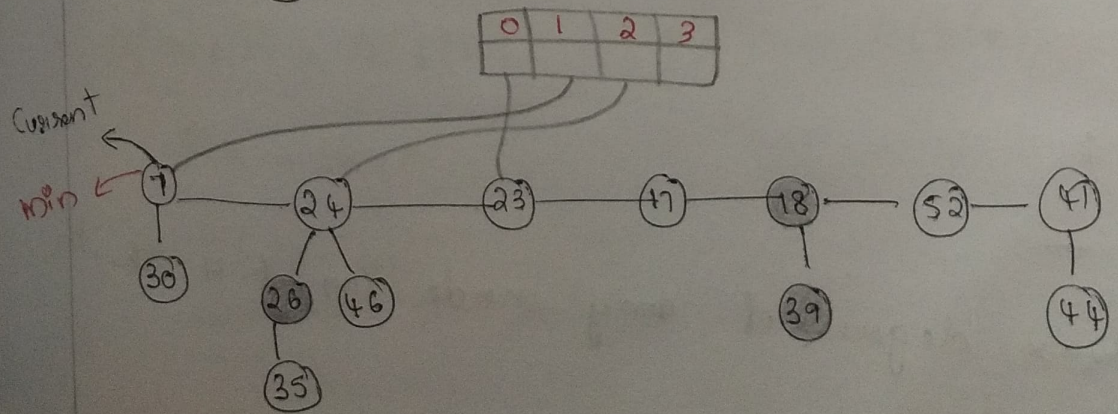
1. Remove y from the root list of H
2. Mark y a child of x, incrementing x, degree
3. y.mark = FALSE

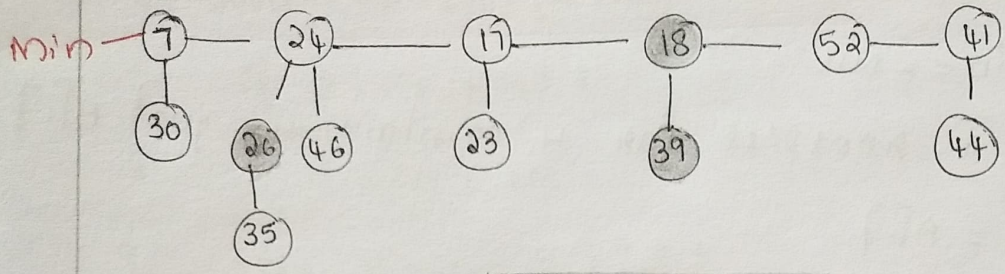
Eg:

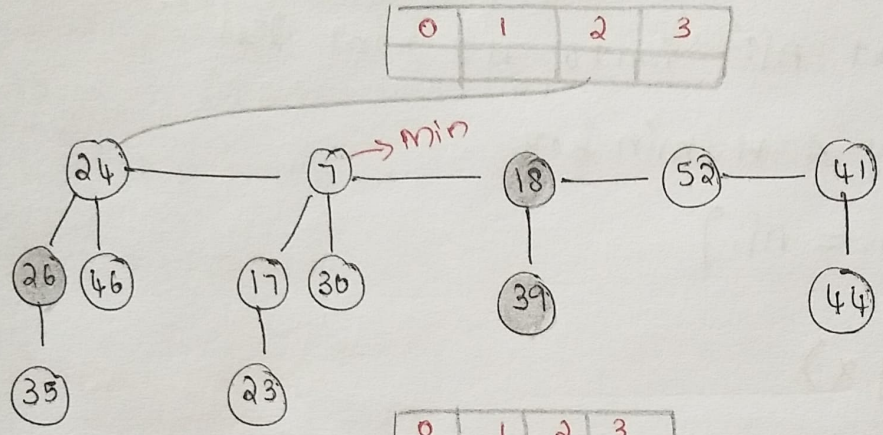


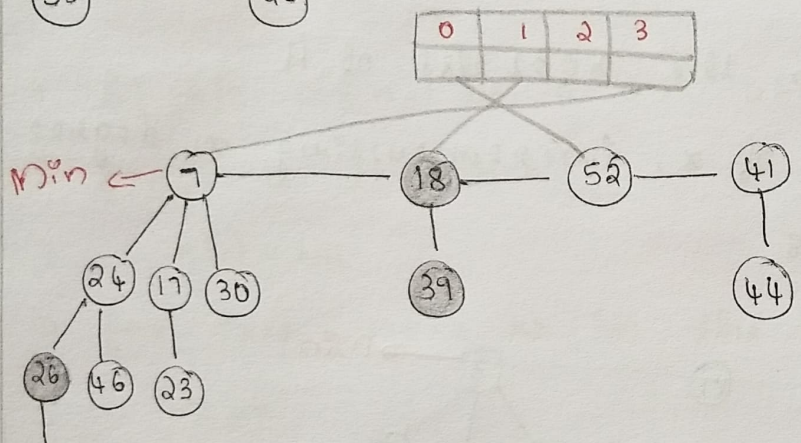
Remove = 3

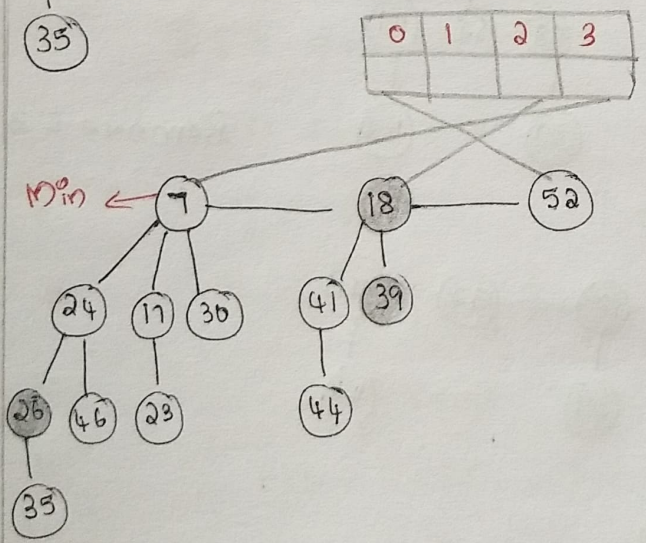












Analysis :

Notation —

D(n) = max degree of any node in F-H with n nodes.

$t(H)$ = # trees in heap H

$\phi(H) = t(H) + 2(m(H))$

- Actual cost : $O(D(n) + t(H))$

- $O(D(n))$ work adding min's children into root list & updating min.

  * atmost $D(n)$ children of min node.

- $O(D(n)) + t(H)$ work consolidating trees

  * work is proportional to size of root list since number of roots decrease by one after each merging.

  * $\leq D(n) + t(H) - 1$ root node at begining of consolidation.

→ Potential before extracting the minimum node is $t(H) + 2m(H)$

→ Atmost $D(n)+1$ root remain & no nodes become ~~marked~~ marked during the operation ⇒ the potential after extracting the minimum ~~no~~ node is $\leq (D(n)+1) + 2m(H)$

→ Amortized cost =

$$P(D(n) + t(H) + ((D(n)+1) + 2m(H)) - (t(H) + 2m(H))$$

$$= O(D(n)) + O(t(H)) - t(H) = O(D(n))$$

$Dn = \log n$

$\Rightarrow O(\lg n)$

5) UNION OF FIBONACCI HEAP

FIB-HEAP-UNION (H_1, H_2)

1) Consolidate the root list of H_1 & H_2 into new root list H.

2) Set the minimum node of H

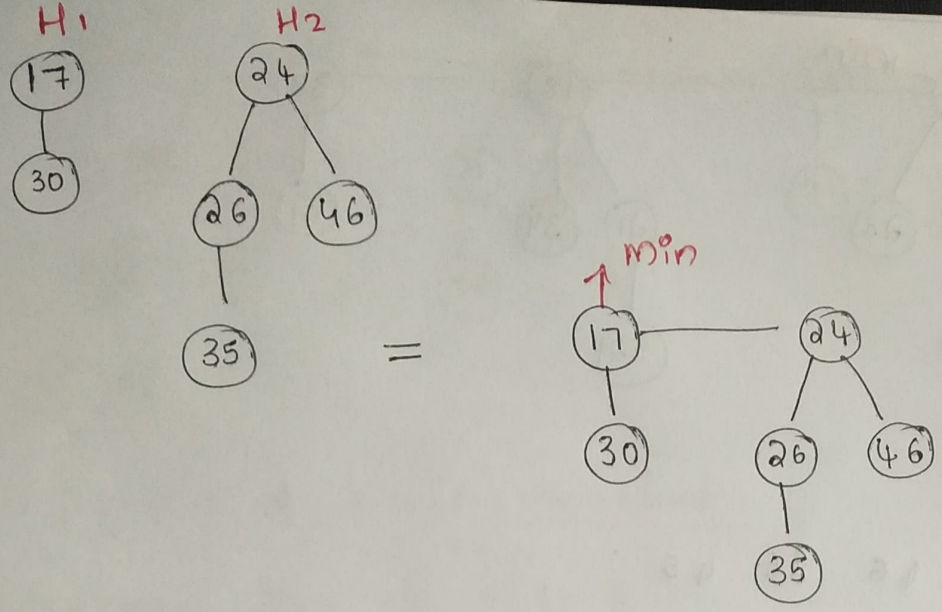
3) Set n[H] to total number of nodes

It simply concatenates the root lists H_1 & H_2, ~~destroy~~ and then determining the new minimum node. Then objects representing H_1 & H_2 will never be used again.

FIB-HEAP-UNION (H_1, H_2)

1. H = MAKE-FIB-HEAP()
2. Hmin = H_1.min
3. Concatenate the root list of H_2 with the root list of H
4. If (H_1.min == NIL) or (H_2.min ≠ NIL and H_2.min.key < H_1.min.key)
5. H.min = H_2.min
6. H.n = H_1.n + H_2.n
7. return H

Eg:



Analysis:

Change in potential is

$\phi(H) - (\phi(H_1) + \phi(H_2))$

$= (t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + ((t(H_2) + 2m(H_2))$

$= 0$ (since $t(H) = t(H_1) + t(H_2)$ and $m(H) = m(H_1) + m(H_2)$)

$\Rightarrow$ Amortised cost = Actual cost = $O(1)$

10/2/2021

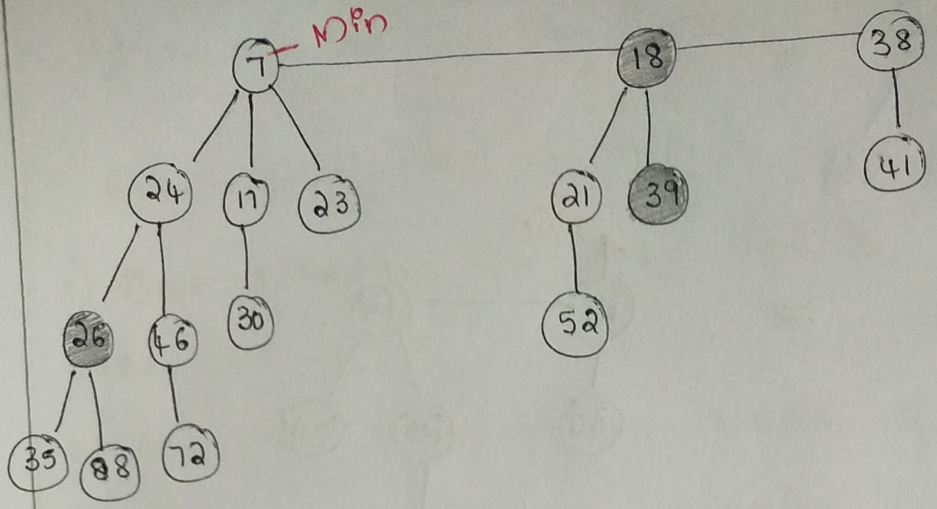
6) Fib heap decrease

To decrease the value of any element in the heap, we follow following algorithm.

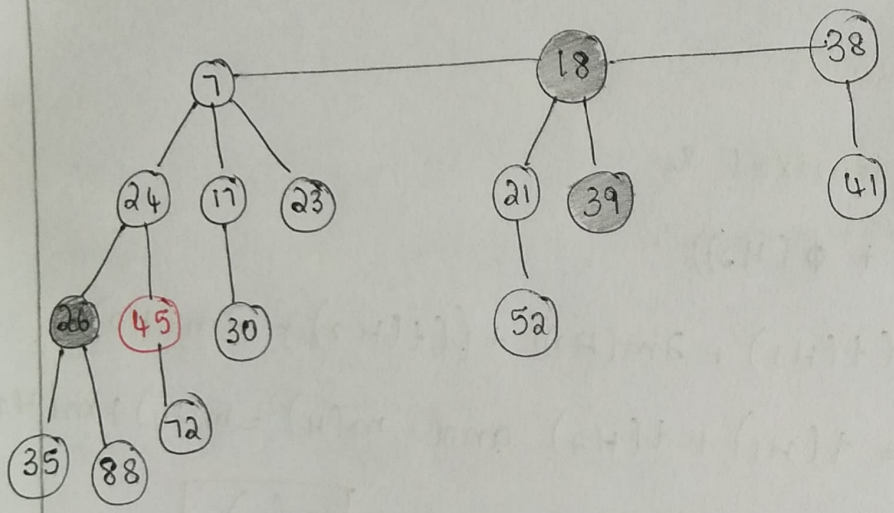
* Decrease the value of the node 'x' to the new chosen value.

CASE 1

1) If min-heap property is not violated,
   * update min-pointer if necessary
   * decrease key of x to k



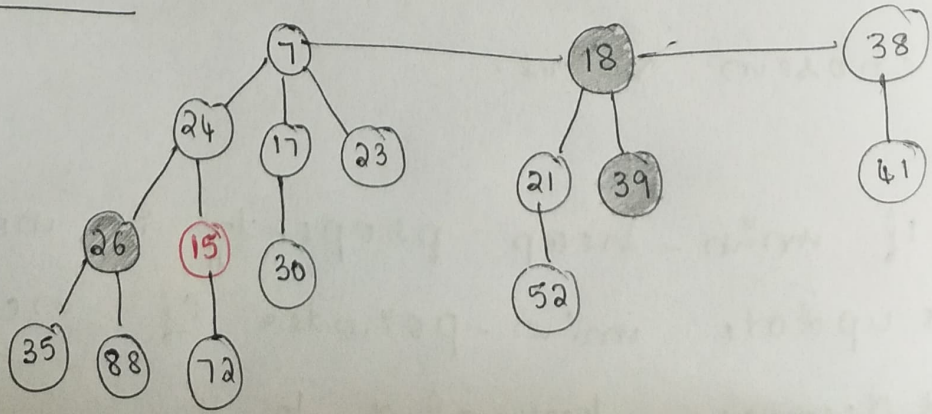
Decrease 46 to 45 :

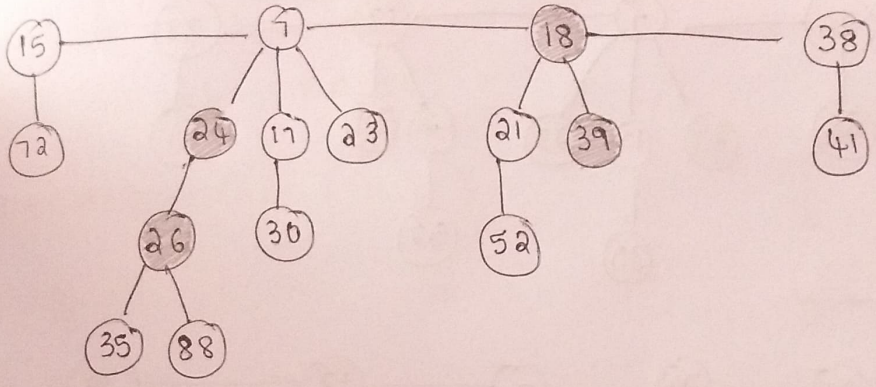


Case 2 : Parent of x is unmarked

- decrease key of x to k
- cut off link between
- mark parent
- add tree rooted at x to root list, updating heap min pointer.

Decrease 45 to 15

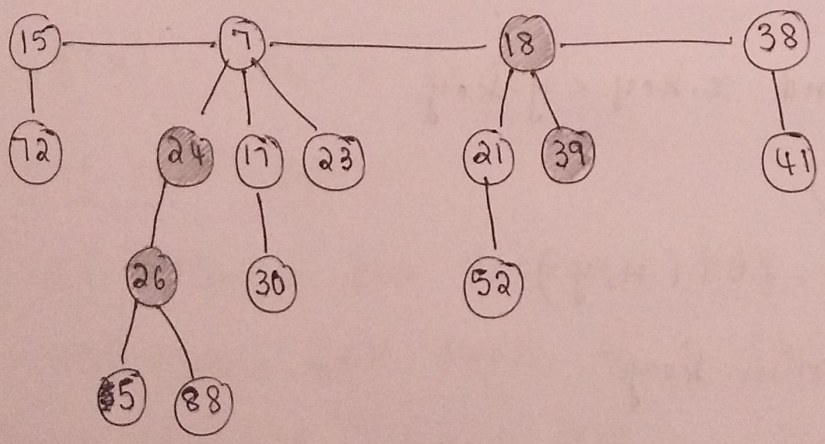


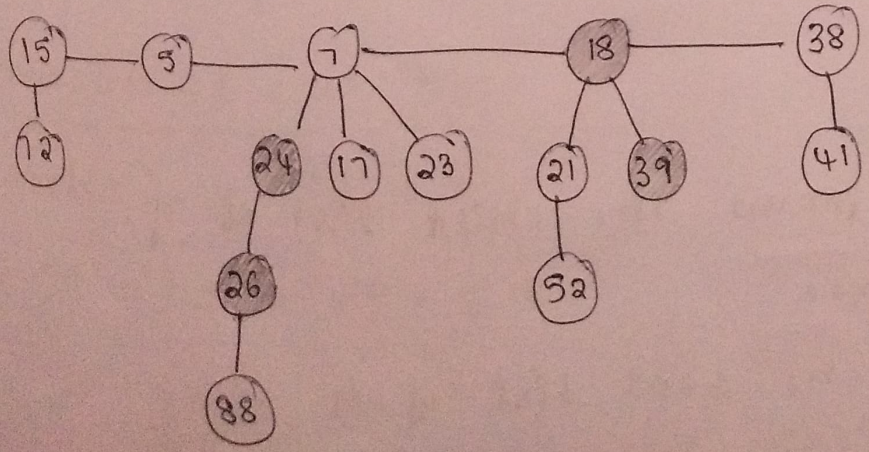


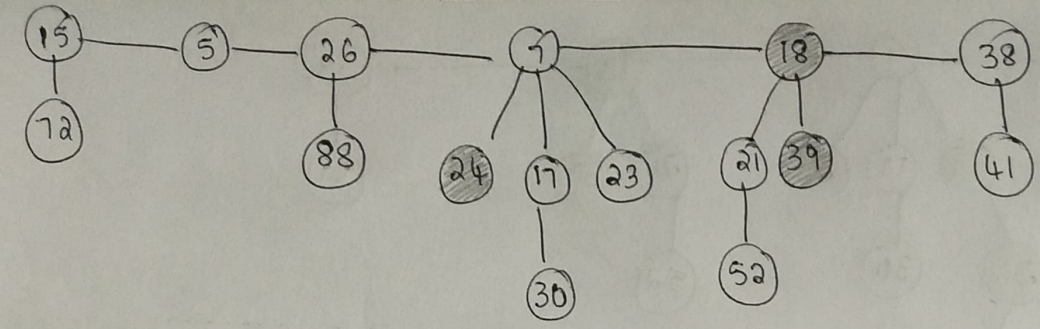
case 3 : parent of x is marked

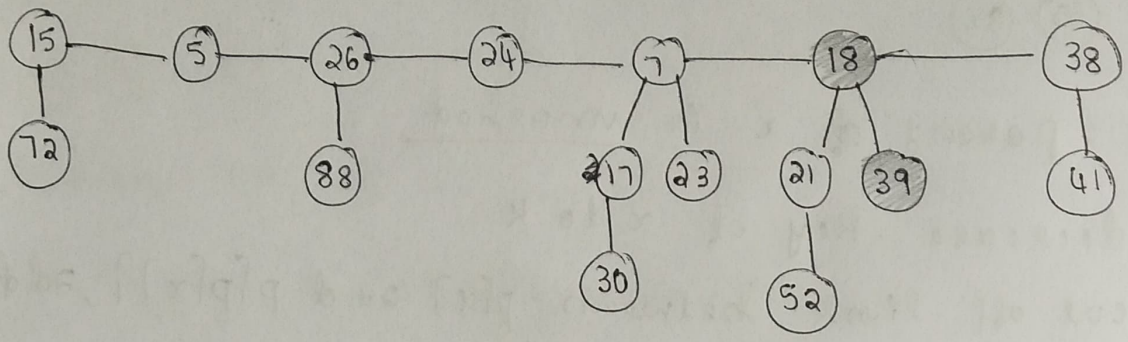
- decrease key of x to k
- cut off link between p[x] and p[p[x]], add p[x] to root list.
  - * if p[p[x]] unmarked, then mark it
  - * if p[p[x]] marked, cut of p[p[x]], unmark & repeat.

decrease 35 to 5









~~Algorithm~~

FIB-HEAP-DECREASE-KEY (H, x, k)

1. if k > x.key
2. error "New key is greater than current key"
3. x.key = k
4. y = x.p
5. if y ≠ NIL and x.key < y.key
6. CUT(H, x, y)
7. CASCADING-CUT(H, y)
8. if x.key < H.min.key
9. H.min = x

CUT (H, x, y)

1. remove x from the child list of y, decrementing y.degree
2. add x to the root list of H

3. x.p = NIL

4. x.mark = FALSE

CASCADING-CUT (H, y)

x = element

y = Parent

z = Grand Parent

1. z = y.p
2. if z ≠ NIL
3. if y.mark == FALSE
4. y.mark == TRUE
5. else CUT(H, y, z)
6. CASCADING-CUT(H, z)

Analysis :

Notation :

t(H) = # trees in heap H

m(H) = # marked nodes in heap H

$\phi(H) = t(H) + 2m(H)$

Actual cost : O(c) :

O(1) time for decrease key

O(1) time for each of c cascading cuts, plus reinserting in root list.

Amortised cost : O(1)

$t(H') = t(H) + c$

$m(H') \leq m(H) - c + 2$

- each cascading cut, unmark a node
- last cascading cut could potentially mark

a node.

$$\Delta\phi \leq c + a(-c+a) = 4 - c$$

7) Delete a key

The following pseudocode deletes a node from an n-node F.H in $O(D(n))$ amortised time. We assume that there is no key value of $-\infty$ currently in FH.

FIB-HEAP-DELETE(H, x)

1. FIB-HEAP-DECREASE-KEY(H, x, $-\infty$)
2. FIB-HEAP-EXTRACT-MIN(H)

• FIB-HEAP-DELETE makes x become the minimum node by giving $-\infty$.

• FIB-HEAP-EXTRACT-MIN procedure then removes node x from F.H

• The amortised time of FIB-HEAP-DELETE is the sum of $O(1)$ amortised time of FIB-HEAP-DECREASE KEY & $O(D(n))$ amortised time of FIB-HEAP-EXTRACT-MIN.

$\therefore D(n) = O(\lg n)$