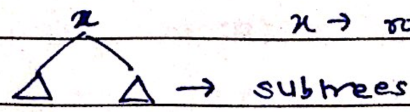
1. Suppose x is a tree having two subtrees as follows;



- We need to rebuild the subtree if a new node is added / deleted and tree turns out to be imbalanced
- For rebuilding of subtree, we can use divide and conquer algorithm.
- Divide and conquer algorithm puts the middle element from the list of nodes at the root and constructs left and right subtrees recursively
- This results in hight difference of tree at most 1 between two existing subtrees.
- Imbalance of subtrees can be measured by;

$$I(x) = \max \{ 0, |\text{size}(\text{left}(x)) - \text{size}(\text{right}(x))| - 1 \}$$

- As the imbalance of the rebuilt subtree turns out to be zero and also the remainder of entire tree has 0 imbalance before rebuilding, the entire new tree has imbalance of 0

2. $I(x) = \{ |\text{size}(\text{left}(x)) - \text{size}(\text{right}(x))| \}$

For deletion :

① If no rebuilding is needed :

Actual cost = $O(\log \text{size}(T))$ = $O(\text{height}(T))$

$\Delta\phi = O(1)$ → as no change in imbalance

∴ Amortized cost = $\Theta(\log \text{size}(T))$

② If rebuilding is needed :

Actual cost = $O(\log n) + \Theta(n)$

Imbalance after rebuilding could be at most $\Theta(n)$ instead of zero as original

$$\Delta\phi \leq -\frac{\hat{c}\ \text{size}(T)}{2} + \Theta(n)$$

∴ After scaling $\hat{c}$;

Amortized cost of deletion = $O(\log \text{size}(T))$

$= O(\log n)$

For -Insertion :

① If No rebuilding is needed:

Inbalance change of path from root → new node would atmost height $(T) \leq \beta \log n$

$\Delta\phi = O(\beta \log n)$

∴ Amortized cost = Actual cost + $\Delta\phi$

$= O(\log n)$

$= O(\log \text{size}(T))$

$= O(\log n)$ P.T.O.

2. (2) If Rebuilding is required:

$$\text{Actual cost} = O(\log n) + \Theta(n)$$
$$= \Theta(n) = \Theta(\text{size}(T))$$

- Imbalance needs to be calculated before insertion

- Suppose insertion takes place in left subtree of x without loss of generality where x is the root

$$\therefore \text{height}(\text{left}(x)) \geq \text{height}(\text{right}(x))$$
i.e. $\text{height}(x) = 1 + \text{height}(\text{left}(x))$

- As tree was balanced before insertion

$$\therefore \text{height}(\text{left}(x)) \leq \beta \log \text{size}(\text{left}(x))$$

- After insertion, the tree is unbalanced;

$$\therefore \beta \log \text{size}(x) \leq \text{height}(x) \quad \text{—— (1)}$$

- As x is lowest point of imbalance, we have

$$\text{height}(x) = \text{height}(\text{left}(x)) + 1 \leq \beta \log \text{size}(\text{left}(x)) + 1 \quad \text{—— (2)}$$

Exponenting (1) & (2) we get,

$$2^{\beta \log \text{size}(x)} \leq 2^{\beta \log \text{size}(\text{left}(x)) + 1}$$

$\therefore$ After insertion

$$\text{size}(x) < 2^{1/\beta} \text{size}(\text{left}(x))$$

P.T.O.

- Before Insertion

$$size(x) + 1 < 2^{1/\beta}(size(left(x)) + 1)$$

$$size(x) < 2^{1/\beta}\, size(left(x)) + (2^{1/\beta} - 1)$$

- ~~Before~~

We know, $size(x) = size(left(x)) + size(right(x)) + 1$

$$size(right(x)) = size(left(x)) - size(left(x)) - 1$$

$$size(right(x)) < (2^{1/\beta} - 1)\, size(left(x)) + 2^{1/\beta} - 2$$

$$size(right(x)) \le size(left(x)) \quad \left[\text{As } 2^{1/\beta} - 2 \text{ is negative}\right]$$

Before insertion;

$\therefore$ Imbalance $I(x) \ge size(left(x)) - size(right(x))$

$$I(x) \ge \frac{size(x)}{2^{1/\beta}} - \left[\left(1 - \frac{1}{2^{1/\beta}}\right) size(x) - 1\right]$$

$$I(x) = \left(2^{-1/\beta} - 1\right) size(x) + 1$$

$$I(x) > \left(2^{1-1/\beta} - 1\right) size(x)$$

After insertion

$$I(x) = O(size(x))$$

$$\Delta\phi = \phi_{final} - \phi_{before}$$

$$= O(size(x)) - \left[2^{1-1/\beta} - 1\right] size(x)$$

2. (2) $\Delta\phi = O(size(x)) = O(n)$

$\therefore$ Amortized Cost $=$ Actual cost $+ \Delta\phi$

Amortized Cost $= O(n)$

4. According to Professor Pinocchino ;

- height of Fibonacci heap with n number of nodes is O(log n)
- To prove that professor's claim is wrong, we need to create an algorithm of Fibonacci heap which has only one tree with (n-1) nodes with empty Fibonacci heap (FH). Later one node is being inserted into the heap
- Create Fibonacci Heap of height 1 with the root key as R. Add elements as;

(R-1) - value less than current root's value

(R+1) - value greater than current root's value

(R-2) - value ~~less~~ twice less than current roots value which is to be deleted value

(R-2) - remove node with key R-2

Pseudocode :

Assume number of nodes in the tree is greater than 2

```
Linear-heap (FH, R, n)          // empty FH
Linear- heap (FH, R+1, n-1)
Insert (FH, min (FH) + 1)
Insert (FH, min (FH) - 1)
Insert (FH, min(FH) - 2)
Deletemin (FH)                  // delete the node with min key
A = min (FH). secondchild
Decreasekey (A, min (FH) - 2)   // decrease node's value by 2
Deletemin (FH)
return
```

P.T.O

④ # Correctness :

- Base Case :

Hypothesis is correct for $n=1$ as Fibonacci heap contains 1 tree with a linear chain of n nodes

Inductive Process :

- Assume that the above statement is is true for $n=k$
- For $n=k+1$, the algorithm first creates a linear chain of 'k' nodes (inductive hypothesis) and then it adds 2 new elements say x and y
  - x - key which is less than minimum key
  - y - key which is ~~to~~ greater than minimum key
- Finally it adds an element R whose value is smaller than minimum key value
- When R is removed, then their remains the chain of k nodes containing x and y.
- As degree of x and y turns to zero, they are being collaborated. Now x becomes the root with the degree 1
- Thus, the chain of height 2 and chain of height k are consolidated.
- By deleting y, we get linear chain with n nodes.
- Hence, Fibonacci heap containing one tree with linear chain of n nodes can be created from a sequence of Fibonacci -heap operations.

6. (a) FIB-HEAP-CHANGE-KEY (H, x, k)

– this operation changes the key of node x to value k

1. Case 1 : $k < x.key$

- When value of k is less than value of $x.key$ then in that case we don't need to change anything ~~and rebuild the subtree~~
- We can just perform DECREASE-KEY operation
- Therefore, we just need to call FIB-HEAP-DECREASE-KEY (H, x, k)
- The amortized cost $\hat{c} = O(1)$

P.T.O.

2. case 2 : $k = x.key$

- No need to do any operation when values are equal

$\therefore$ Amortized cost $\hat{c} = c \rightarrow O(1)$

3. case 3 : $k > x.key$

- When value of k is greater than x.key value then we need to update the value of x.key to value k First

- After updating value of x.key, we need to put down the value of x.key (original) into the subtree until minheap property is achieved

- Basically, we first need to delete the node x and insert new node with the value equivalent to 'k'

$\therefore$ Amortized cost $= O(\log n) + O(1)$
$= O(\log n)$

→ FIB-HEAP-CHANGE-KEY (H, x, k)

```
IF k ≤ x.key then
    FIB-HEAP-DECREASE-KEY (H,x,k)
Else
    FIB-HEAP-DELETE-KEY (H, x)
    FIB-HEAP-INSERT-KEY (n, k)        // n = new node
```

6. (b) FIB_HEAP_PRUNE (H, r)

- We can delete nodes from leaves, so no rearrange-ments are necessary, each single node deletion are $O(1)$
- Ammortized Analysis :

Let $(D_i) = t(H) + m(H)$

$S = \min(r, n[H])$ // number of nodes deleted

so,

$(D_i) - (D_{i-1}) = -s$

$\therefore$ Amortized cost $= c_i + (D_i) - (D_{i-1})$

$= s * O(1) - s$

$= \underline{\underline{O(s)}}$