Fibonacci Heap

- This is an advanced data structure. It is used when we want all operations (except deletion) to be very fast, eg. $O(1)$ time!

- For eg. in Dijkstra's algorithm we do n Extract-min & m ($\gg n$) Decrease-key operations. Trees take $O(m \lg n)$ time while Fibonacci heaps will take only $O(m + n \lg n)$.

- The heap is a collection of (rooted) trees with relaxed conditions; but each node stores a lot of pointers!

• Each node x contains a pointer $p[x]$ to its parent & to a single child$[x]$.

- All the children of x are linked in a circular doubly linked list (cdll):
  Each child y (of x) has pointers $left[y]$ & $right[y]$. If y is the only child then these pointers equal y.

- Number of children is stored in $deg[x]$.
- A special flag $mark[x]$ indicates whether x lost a child since x was made a child of $p[x]$.
  (False)
  Newly created nodes have mark = F.
- A Fibonacci heap H is accessed by the pointer $min[H]$ that points to the root of a tree with min. key.
- The number of nodes in H is $n[H]$.
- The roots are also in a cdll.
- Key of $x \leq$ Key of any descendant of x.

- Let $t[H] :=$ #trees, $m[H] :=$ #marked nodes & $d(H) :=$ max deg of a node in H.

- Amortized analysis of heap operations:

Define $\varphi(H) := t[H] + 2 \cdot m[H]$.

When a sequence of operations change the heap as $\phi =: H_0 \to H_1 \to \cdots \to H_\ell$ with c_i being the actual cost in the i-th step, we define the amortized cost as

$$\hat{c}_i := c_i + \varphi(H_i) - \varphi(H_{i-1}).$$

overestimates / underestimates

(This may be negative for some i.)

▷ Total amortized cost $= \sum_{i=1}^{\ell} \hat{c}_i$

$$= \left(\sum_i c_i\right) + \varphi(H_\ell) - \varphi(H_0).$$

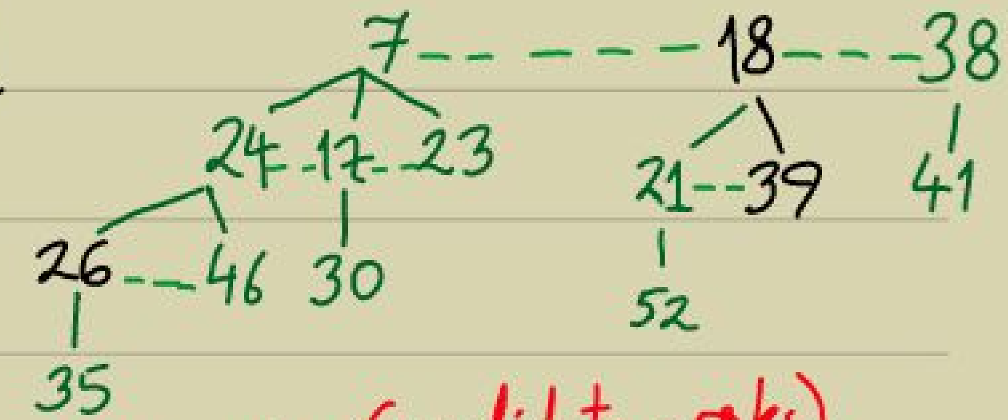
$$\geq \sum_i c_i = \text{actual cost.} \quad \left[\because \varphi(H_0) = 0 \;\&\; \varphi(H_i) \geq 0\right]$$

▷ Thus, (upper) bounding the amortized cost also bounds the actual cost.

Idea: There is not much structure in H after the insert & decrease-key operations. This keeps them $O(1)$ time.

However, after extract-min an expensive step is done to consolidate the heap — in each row of siblings the degrees are distinct (almost!).

The amortized cost in this case is $O(d(H))$.



(the way Consolidate works)

<u>Lemma</u>: Assuming the above, $d(H) = O(\lg n)$, where $n = n[H]$.

<u>Proof</u>:

- Let x be a node in H with degree $d[x] =: k$.
- Let $y_1, \dots, y_k$ be the children of x in the order in which they were linked. (by Consolidate)
- $\deg[y_1] \geq 0$. For $i \geq 2$, when y_i was linked to x, $\deg[x] = \deg[y_i]$, so we must have had $\deg[y_i] = i-1$; if y_i lost a child its $\deg[y_i] = i-2$. Thus, $\deg[y_i] \geq i-2$, $\forall i \geq 2$.

- Let s_k denote the min. possible size of the tree rooted at x with $deg[x] = k$.

$\Rightarrow s_k \geq 2 + s_0 + \dots + s_{k-2}$

- It can be shown by induction that

$s_k \geq F_{k+2}$ [Fibonacci numbers $F_0 := 0$, $F_1 := 1$, $F_{i+2} = F_{i+1} + F_i$.]

[It can be shown that $F_{k+2} \geq \phi^k$, where $\phi := (1+\sqrt{5})/2$.]

$\Rightarrow n[H] \geq s_k \geq \phi^k$.

$\Rightarrow k = O(\lg n)$. □

– Let us now sketch the operations of Insert, Decrease-Key & Extract-Min.

(Fredman & Tarjan 1985)

```
Insert (H, x) {
    Create a node with key x;
    Add it in the root list;
    Update min[H] if required;
}
```

▷ Amortized cost $= O(1) + (t+1+2m) - (\underset{t(H)}{t} + 2\underset{m(H)}{m}) = O(1)$.

- Suppose in H we want to decrease the key of x to k. ($key[x] > k$)

Let y = parent of x.

- If $key[y] > k$ then x is cut & added to the root list. [Cut(H, x, y).]

- This requires a special process on the ancestors, called Cascading-cut:
  - If y is unmarked then mark it. Else cut y Cut$(H, y, z = p[y])$ & Cascade on z.

▷ Suppose Cascade-Cut is called c times. Then the amortized cost of Decrease-key is: $O(c) + \phi(H') - \phi(H)$

justifies the choice of ϕ

$$= O(c) + (t+c) + 2(m-(c-1)+1) - [t+2m]$$

$$= O(c) + c + 2(2-c) = O(c) - c$$

By scaling up ϕ we make the above $O(1)$.

→ Cost gets reduced by the negative potential change!

- Finally, we describe the most complicated operation — Extract-Min(H).
  It is here that the structure of the Fibonacci heap is secured.

- The min[H] root is removed & its children are added to the root list.

- Next, Consolidate(H) is called to link the roots that have the same degree.
  At the end, each sibling (in a row) has a distinct degree.

```
Consolidate(H) {
    for i = 0 to d(H)
        A[i] ← null;
    for w in the root list of H {
        x ← w; d ← deg[x];
        while A[d] ≠ null {
            y ← A[d]; // x & y have equal degree
```

x
/ ... $(d+1)$
y

```
      if key[x] > key[y] then swap x,y;
      Make y a child of x; Increment deg[x];
      Mark[y] ← False;
      A[d] ← null;   d ← d+1;
    } //end while
    A[d] ← x;
  } //end for
  Update min[H];
}
```

- Amortized cost of Extract-Min(H):

$d(H) = O(\lg n[H])$ → as other operations preserve it.

The roots are $d(H)+t(H)$ many to begin with. [& in the end they're $\leq d+1$ many]

Each while-iteration links two of them. Thus, the actual cost is $O(d(H)+t(H))$.

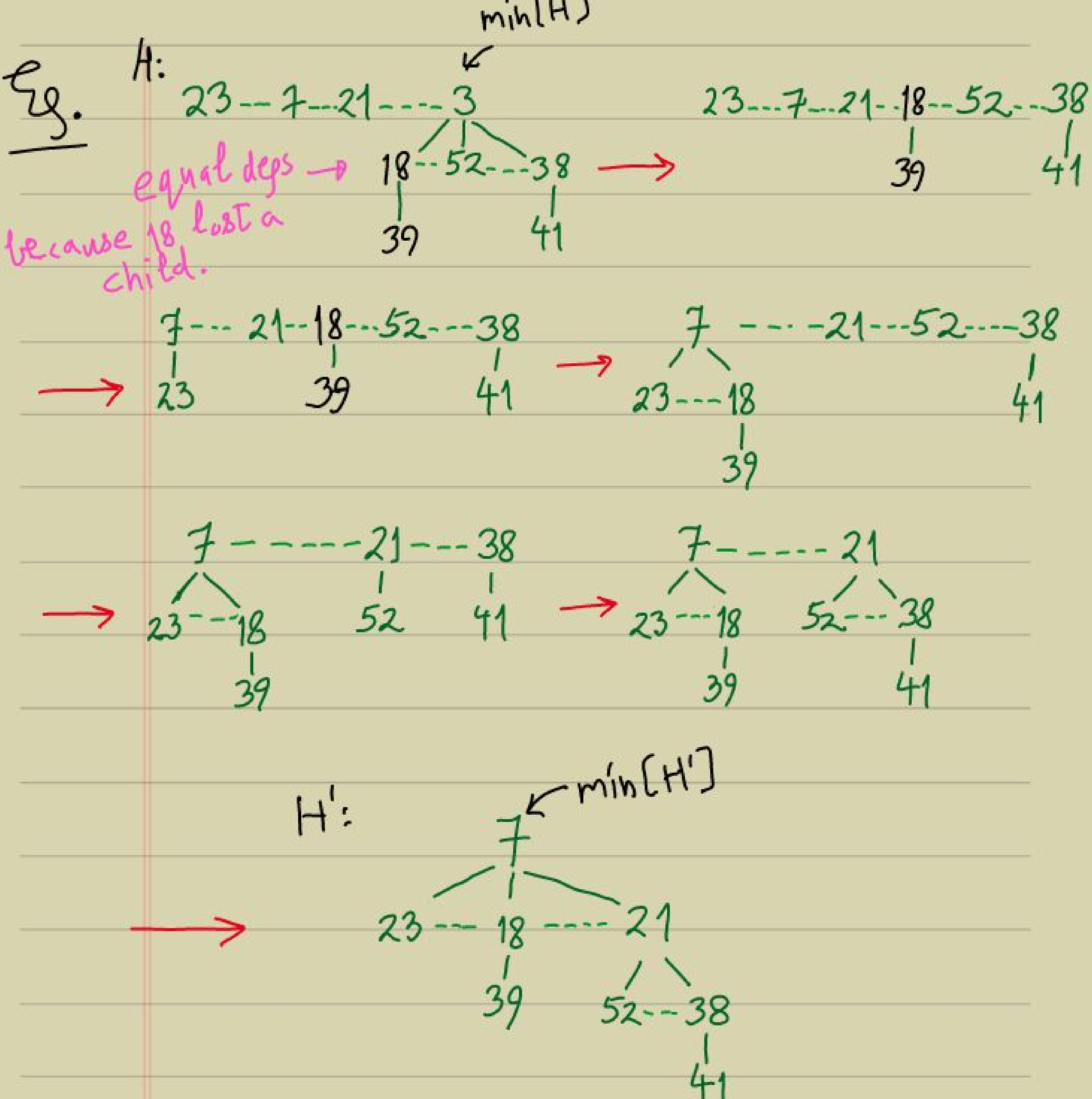
The amortized one is $O(d+t) + \phi(H') - \phi(H)$

$$= O(d(H)+t(H)) + [(d(H)+1) + 2m(H)] - [t(H) + 2m(H)] = O(d(H))$$

(By scaling up $\phi(H)$ function)

$$= O(\lg n[H]).$$

Eg.



Exercise: In each set of siblings (in the end), the degrees are distinct among unmarked.
(Hint: Insert & Decrease-Key don't affect the deg of unmarked.)