Amortization :

- Suppose our data structure supports operations A, B, C

- we say the "amortized costs" of these operations are t_A, t_B, t_C ~~at any se~~ respectively, if any sequence n_A of A operations, n_B of B operations, n_C of C operations . .

$$\text{takes time} \le n_A t_A + n_B t_B + n_C t_C$$

- Common way to prove amortized bounds ~~on~~ is via potential function method.

Define potential fn. • $\Phi \left\{ \begin{array}{c} \text{state (space) of the} \\ \text{data structure} \end{array} \right\} \to \mathbb{R}^+$

$$\bullet\ \Phi\{\text{empty structure}\} = 0$$

- let us say ~~that~~ we perform k operations with actual times $t_1, t_2, \dots t_k$,

States of the data structure are
$S_0, S_1, S_2, \dots, S_i, \dots, S_k$

S_0: before any operation; S_i: after ith operations

- amortized cost of ith ~~an~~ operation is ~~defined to~~ bounded by $t_i + \Phi(S_i) - \Phi(S_{i-1})$

- The total amortized cost is bounded by

$$\sum t_i + \Delta\Phi = \sum t_i + \Phi(S_K) - \Phi(S_0)$$
$$= \left(\sum t_i + \Phi(S_K)\right) \ge \sum t_i$$

Stack operations

push, pop, Multipop

Define potential fun ϕ on a stack to be the number of object in the stack.

i.e., $\phi\{$state of the stack$\} \rightarrow$ number of object in the stack.

- $\phi(S_0) = 0$ [$\because S_0$ is the empty stack]
- If the ith operation on a stack containing s object is a PUSH operation, then the potential difference is $\Delta\phi = \phi(S_i) - \phi(S_{i-1}) = (s+1) - s$
$= 1$

$\therefore$ The amortized cost of push operation is equal to

$$\underbrace{c_i}_{\text{actual cost of push operation}} + \Delta\phi = 1 + 1 = 2$$

- Suppose the ith operation on the stack is Multipop(S, k), which causes $k' = \min(k, s)$ object to be ~~popped~~ popped off the stack, The actual cost of the operation is k', and the potential difference $\Delta\phi = \phi(S_i) - \phi(S_{i-1}) = -k'$

$\therefore$ The amortized cost of multipop operation is equal to $c_i + \Delta\phi = k' - k' = 0$

Similarly, the amortized cost of an ordinary pop operation is 0.

Amortization

Incrementing a binary counter (k bits)

It is a k bits counter

$\Phi\{\text{State of the counter}\} \rightarrow$ # of 1s in the counter.

Let b_i be the number of 1s after ith increment operation.

Suppose that the ith increment operation resets t_i bits. from 1 to 0

$\therefore$ Actual cost of the ith operation is $t_i + 1$

- If $b_i = 0$, then ith operation resets all k values, and so $b_{i-1} = t_i = k$
- If $b_i > 0$, then $b_i = b_{i-1} - t_i + 1$.

In either case, $b_i \le b_{i-1} - t_i + 1$, and the potential difference

$$\Delta\Phi = \Phi(S_i) - \Phi(S_{i-1}) \le (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$$

$\therefore$ The amortized cost

$$= \text{actual cost} + \text{potential difference}$$

$$= (t_i + 1) + (1 - t_i)$$

$$= 2$$

- Heap (H) (abstract data structure)
  - supports
    - insert (x)
    - deleteMin(H) → return min $x \in H$, then delete it
    - ~~decreasekey~~
    - deckey (decrease key value of an item

Dijkstra run time
$O(n.(t_I + t_{delmin}) + m\, t_{deckey})$

Implementation of heap

- Binary heaps (Williams, 1964)
  - Insertion (t_I) = deletemin (t_{dm}) = deckey (t_{dk}) = $O(\log n)$ worst case

- ~~Bipnomial~~ Binomial heaps (Vuillemin, 1978)
  - amortize $t_I = O(1)$ $t_{dm} = O(\log n)$
  - worst case $t_{dm} = t_{dk} = O(\log n)$
    worst case $t_I = O(\log n)$

- Fibonacci heaps (Fredman, Tarjan, JACM '87)
  - amortized $t_I = t_{deckey} = O(1)$, $t_{deletemin} = O(\log n)$

[• Brodal SODA '96
– gets all the same bound as Fibonacci heap but worst case

• Brodal, Lagogiannis, Tarjan, STOC '12]
– same as afore, just using ~~pointer machine~~ pointer ....

• Kaplan, Tarjan, Zwick arXiv '14

<u>Binomial Heap</u>

① • Each item is a node, and we maintain these nodes in a forest.

② • If $x = \text{parent}(y)$, then $\text{key}(x) \le \text{key}(y)$

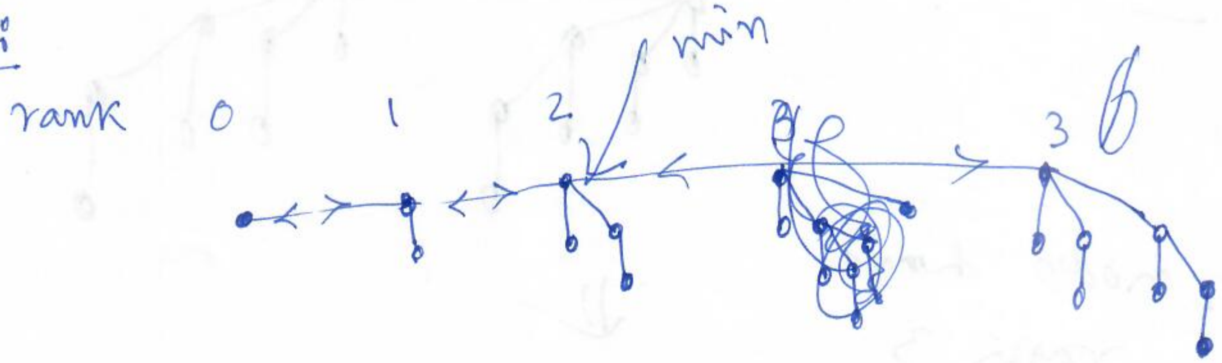
③ • For a tree in a heap, its "rank" is the degree of its root

④ • A tree with rank k has 2^k nodes in it

⑤ • For each k, we will have at most 1 tree of rank k.

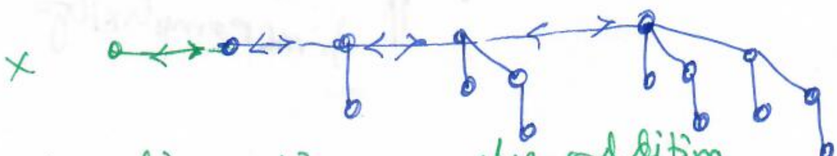
⑥ • root of rank k has k subtrees, these are trees of rank $0, 1, \dots, k-1$

<u>Example:</u>



• <u>Deckey</u> : change the key, then keep swapping upwards as necessary.

• Insert(x) : Add a new singleton tree to forest, then repeatedly merge trees of equal rank.

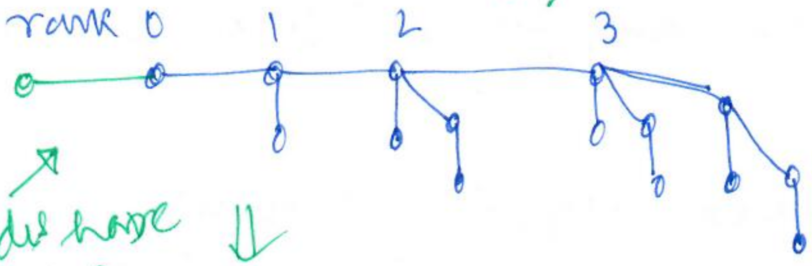


After adding new node it is violating condⁿ ⑤

It look like a binary number addition

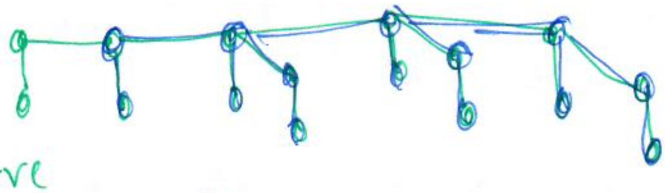
$$\begin{array}{r} 1111 \\ \hline 10000 \end{array}$$

→ ~~st~~ Kind structure containing only ~~at~~ 1 tree with rank 4 (by merging)



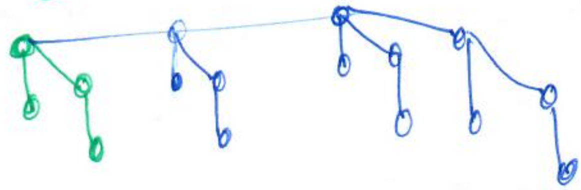
2 nodes have rank 0

⇓



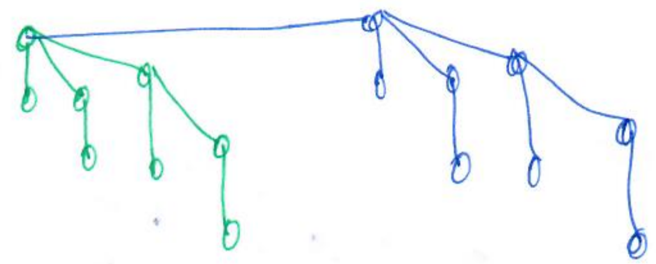
2 nodes have rank 1

⇓



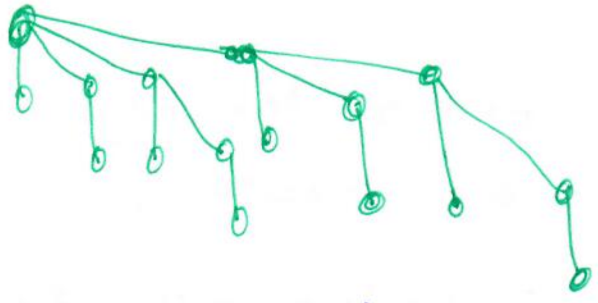
2 nodes have rank 2

⇓



2 nodes have rank 3

⇓



- worst-case running time $O(\log n)$
- amortized time $O(1)$.

binary counter in amortized time complexity

Binomial heap

$$\begin{array}{r} 1111 \\ 1 \\ \hline 10000 \end{array}$$

Insertion cost:

worst case: all operation takes $O(\log n)$ time
[like binary]

potential fn :

- Φ (data structure state) = # trees

- Actual cost of an insertion is

$O(T - t + 1)$

old number of trees — T
new number of trees — t

amortized cost of insertion

actual cost + potential difference

$= \underbrace{\text{actual cost}}_{T-t+1} + \underbrace{\Delta\Phi}_{t-T}$

$= 1$

$\Rightarrow$ amortized cost of insertion is $O(1)$.

Deletemin: ~~delete~~ remove root of some tree (mainly minimum), and insert all its childrens as root in the main forest. Now merge equal rank trees.

- worst case running time $O(\log n)$
- Amortized time $O(\log n)$. ~~binary counter amortized time complexity.~~

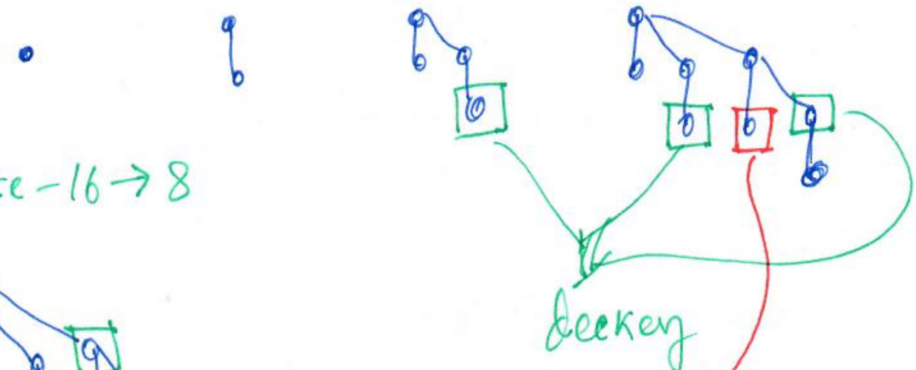
- Fibonacci Heap   Deckey: amortized $O(1)$
  - No reason for insertions to spend time ~~consoli~~ consolidating. Do all consolidation during deleteMin.

- Really, lazy deckey

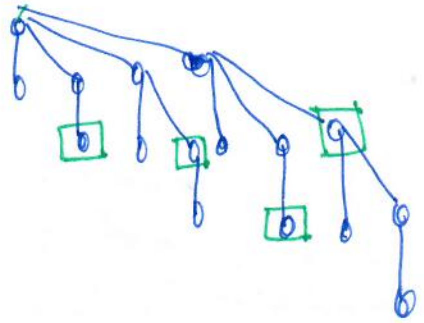
  dec x's key, and if x's new key is smaller than parent, cut x's tree out and place as a top level tree.

- If node p losses one child due to deckey, no problem. If p losses a second child, we cut p out of its tree too, and make p's tree a top level tree.

size	1	2	4→3	8→5
rank	0	1	2	3



rank 4 size-16→8



can't afford deckey as it parent lose two childs, so it will be place on ~~on~~ top level tree ⇒ ~~it will~~ tree ~~will loose~~ will loose it rank 3.

Claim: amortized $t_I, t_{dk} = O(1)$, $t_{dm} = O(\log n)$.

Insertion: amortized $O(1)$

deletemin: " $O(\log n)$

deckey: amortized $O(1)$ → How?

~~atea~~ let $\text{mark}(x) = \begin{cases} 1 & \text{if } x \text{ has lost child} \\ 0 & \text{otherwise} \end{cases}$

data structure

$$\Phi(\text{state}) = \underbrace{\#\text{ trees}}_{T(H)} + 2(\underbrace{\#\text{ marked items}}_{m(H)})$$

It is coming from the fact that after the operation deckey the changes happen on (i) # trees, and (ii) # marked items

Insertion:

$$\underbrace{\text{actual cost}}_{O(1)} + \underbrace{\Delta\Phi}_{1} = O(1)$$

deleteMin: $T + \underbrace{\Delta T(H)}_{t-T} + \underbrace{\Delta m(H)}_{\leq 0}$

$= O(t) = O(\log n)$

deckey:

case 1: x ~~dont~~ does not cut out

$$\underbrace{\text{actual cost}}_{O(1)} + \underbrace{\Delta\Phi}_{0} = O(1)$$

case 2: we do some number $c > 0$ of cascaded cut

$$\underbrace{\text{actual cost}}_{c} + \underbrace{\Delta T(H)}_{c} + \underbrace{2\Delta m(H)}_{-2c+2}$$

$= O(1)$

5/04/2023

200123055
210123006
210123041
222123014
200106055
200106019
210123062
200123010
200123025
200123045
222123029

13/02/2024

200106055
200106019
236123006
210123030
210123029
210123006
222123015
222123043
200123021
200123045
222123045
222123001
210123014
210123041