Mergeable Heaps: Fibonacci heaps

A fibonacci heap implements the Mergeable heap operations with amortized times as follows:

Insert	— $O(1)$
Merge	— $O(1)$
DecreaseKey	— $O(1)$
DeleteMin (ExtractMin)	— $O(\lg n)$
FindMin	— $O(1)$
Delete	— $O(\lg n)$

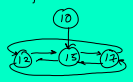
Implementation

Doubly linked circular list of trees (tree roots), each tree in minheap order.

Each node x has these fields:

- $x.key$ (includes satellite data)
- $x.parent$
- $x.left$ — left sibling of x
- $x.right$ — right sibling of x
- $x.degree$ — # children of x
- $x.child$ — points to any child of x
- $x.mark$ — boolean (data might only change after certain operations)

Children of a node are kept on a doubly linked circular list (left & right fields are the links)



The heap H has two attributes:

- $H.min$ — points to the node with min key (a root)
- $H.n$ — # items in the heap.

FindMin(H) — return $H.min$

H=Merge(H_1, H_2) — merge the two root lists into a single list (set $H.min$ accordingly)

$O(1)$

Insert(H, x) — create a new fib heap with one element x, merge with H.

Potential Function for fib heap H.

$\Phi(H) :=$ (# of trees) + 2 (# marked nodes)

Given a number of heaps $H_1, \ldots, H_n$ which could be merged,

Define $\hat{\Phi}(H_{1\ldots n}) = \sum_{i=1}^{n} \Phi(H_i)$

Check that everything above has amortized time $O(1)$.

FindMin:

$\hat{c}_i = c_i + \Phi(H_{after}) - \Phi(H_{before})$ (same)

$= c_i = O(1)$

H=Merge(H_1, H_2)

$\Phi(H_1, H_2) =$ (# trees in H_1) + 2 (# marked nodes in H_1) + (same in H_2)

$\Phi(H) =$ (# trees in H_1) + (# trees in H_2) + 2 (# marked in H_1) + 2 (# marked nodes in H_2)

so $\Phi(H) = \Phi(H_1, H_2)$

$\therefore \hat{c}_i = c_i + 0 = c_i = O(1)$

Insert(H, x)

$\hat{c}_i = c_i + \Delta\Phi = O(1)$

$O(1) + 1 + 0$

(1: added a tree; 0: change in # marked nodes)

DecreaseKey(H, x, k) [$k < x.key$]

$x.key := k$

If $k \geq x.parent.key$ then return (nothing to do)

else let p be x's parent

remove x from its sibling list and make it the root of a new tree in H's tree list

if $p.mark$ then

$p.mark :=$ true

// mark the parent

else, cut p out of the list of siblings and promote it to a new tree root.

"cascading cut"

and so on ...

Amortized analysis of DecreaseKey:

let r be the number of cuts.

assume each cut take time 1.

#trees increases by r

#marked nodes decreases by at least $r-2$

So $\hat{c}_i := \underbrace{O(1) + r}_{c_i} + \Delta(\#\text{trees}) + 2\Delta(\#\text{marked nodes})$

changing key, comparing with parent

$\leq O(1) + r + r - 2(r-2)$

$= O(1) + 4 = O(1)$

Delete(H, x)

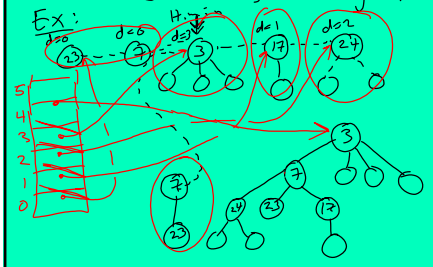
	amortized time
DecreaseKey(x, $-\infty$)	$O(1)$
DeleteMin(H)	$O(\lg n)$

DeleteMin(H)

$O(1)$
- Remove & save (for return) H.min from the root list
- promote H.min's children to the root list (merging the 2 lists)

Consolidate: Traverse the root list, looking for 2 trees of the same degree (d, say) when found, combine into a single tree of degree $d+1$ (as with a binomial heap) by making one root a child of the other root. Continue until no two trees have the same degree. Store degrees of trees in an array index by the degree.

Ex:



When done traversing/combining read off the non-null entries in the array to form a list of trees with distinct degrees.

Next time: Let $D(n)$ be the max possible degree of any tree in a heap with n items

Prove next time that $D(n) = O(\lg n)$

Amortized cost:

~~$O(\lg n)$ traverse/combine steps~~

single combine:

$\Delta(\#\text{trees}) = -1$

$\Delta(\#\text{marked nodes}) = 0$