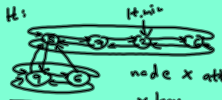
Fibonacci Heaps

	Binary Heap	Fib Heap (amort)
Insert	$\Theta(\lg n)$	1
Find min	1	1
Extract min	$\lg n$	$\lg n$
MakeHeap	1	1
Merge	n	1
DecreaseKey	$\lg n$	1
Delete	$\lg n$	$\lg n$

Fib heap H: is a collection of trees, each tree in min-heap order ($x.key \geq x.p.key$, p = parent)

linked data struct



node x attrs:

- $x.key$
- $x.next$, $x.prev$ } sibs
- $x.p$ (parent)
- $x.child$ } one of x's children
- $x.degree$ = # of children of x
- $x.mark$ — Boolean: true = "x is loser", false = "x is unmarked"

$H.min$

$H.n$ = size of H

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

$D(n)$ = upper bound on the degree of any node in a heap of size n

$O(\lg n)$

$m(H)$ = # of marked nodes

Define potential of heap H:

$\Phi(H) = c(t(H) + 2m(H))$

where c is a constant that is large enough to cover cost of all $O(1)$-time ops described below. [suppress: $(c := 1)$]

MakeHeap(H): $H.n := 0$

amort cost = $O(1)$. $H.min :=$ nil

FindMin(H) — return $H.min$

$\hat{c} = O(1)$

Insert(H, x) — make add x (1-node tree) to root list of H

$\hat{c} = 1$

($\Delta\Phi = 1$)

(x is unmarked)

[adjust $H.min$ if necessary, $H.n$ ++]

Union(H_1, H_2)

splice the two root lists together; new heap H

- $H.n := H_1.n + H_2.n$
- $H.min$ points to smaller (key) of $H_1.min$ & $H_2.min$.

$\Delta\Phi = 0 \Rightarrow \hat{c} = O(1)$.

ExtractMin(H):

$x := H.min$

remove x from root list of H

for each child of x:

- $x.p := nil$
- insert x into root list of H

$x.degree$ updates → (update $H.min$ as nec.)

$H.n$ --

actual cost $c = O(x.degree)$

$\Delta\Phi = x.degree$

$\hat{c} = O(x.degree)$

Fact: $x.degree = O(\lg n)$

$\hat{c} = O(\lg n)$ so far

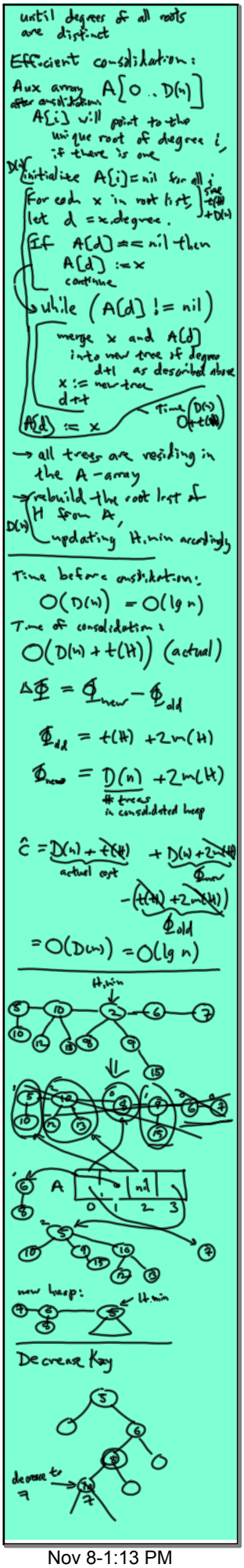
Now consolidate roots

repeat:

if x, y are roots with same degree, make into one tree:

if $x.key < y.key$ then make y a new child of x else make x a new child of y increment degree of surviving root.

Nov 8-12:29 PM


until degrees of all roots are distinct
Efficient consolidation:
Aux array A[0..D(n)]
after consolidation A[i] will point to the unique root of degree i, if there is one
initialize A[i]=nil for all i
for each x in root list
let d = x.degree.
if A[d] == nil then
A[d] := x
continue
while (A[d] != nil)
merge x and A[d] into new tree of degree d+1 as described above
x := new tree
d++
A[d] := x
→ all trees are residing in the A-array
→ rebuild the root list of H from A, updating H.min accordingly
Time before consolidation:
O(D(n)) = O(lg n)
Time of consolidation:
O(D(n) + t(H)) (actual)
ΔΦ = Φnew − Φold
Φold = t(H) + 2m(H)
Φnew = D(n) + 2m(H)
trees in consolidated heap
ĉ = actual cost + Φnew − Φold
= O(D(n)) = O(lg n)
H.min
A
nil
0 1 2 3
new heap:
H.min
Decrease Key
decrease to 7

Nov 8-1:13 PM