Fibonacci heaps

Recall that a heap data structure supports

- ins (x, v): insert key x with value v
- extractMin(): delete and return the key with min v
- decKey (x, v'): decrease the value of x to v'

Binary heap: $O(\log n)$.

Today: Fibonacci heap. ins, decKey in $O(1)$
extractMin in $O(\log n)$
amortized.

↑ n_i ins
n_d decKey
n_e - - - -
in $O(n_i + n_d + n_e \log n)$.

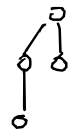
Binomial heaps.

Store keys in a forest.

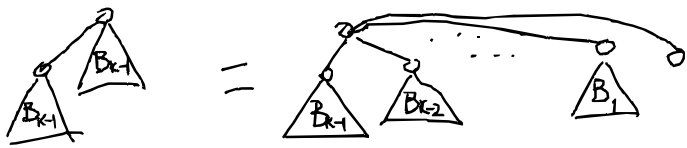
Each tree is a binomial tree B_k.

B_0: ○ (one node)

B_1: 

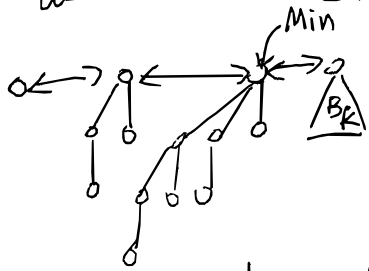
B_2: 

k is the rank
= root degree

B_k: 

Claim: B_k has 2^k nodes. Hence, $k \le \log n$.

A binomial heap maintains
- a set of binomial trees with distinct ranks
- each binomial tree stores 2^k keys, and satisfies the heap property (parent ≤ child)
- all roots are stored in a doubly linked list.

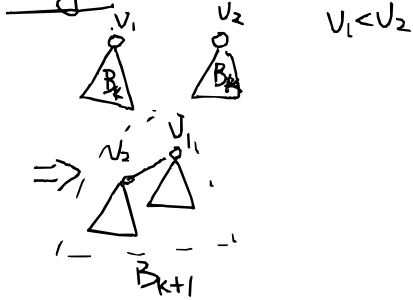


Claim: There can be at most $\log n$ trees.

Claim: The min must be in one of roots.

ins(x, v):
create a B_0 containg (x, v)
while ∃ two trees with same rank
merge them

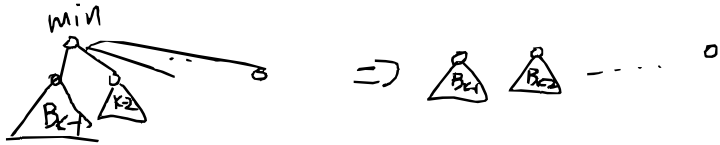
merge:



extractMin:

Suppose min = root of a B_K

remove this root and <u>break</u> B_K in $B_0, \ldots, B_{k-1}$



while ∃ two trees with same rank

<u>merge</u> them.

return min

decKey(x, V')

while V' < value of x's parent

swap x with x's parent.

all ops in O(log n) time.

Fib heap

maintain a binomial heap with "lazy" updates

it maintains a set of trees

- each tree is a binomial tree, possibly with some branches removed
- each tree satifies the heap property
- there may be more than one tree with the same rank.

ins(x, V):

create a B_0 with (x, V)

extractMin:

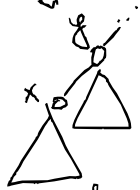
delete the min and break subtrees

while ∃ two trees with same root degree

merge

return min

decKey(x, V'): (each node y stores a bit mark[y] indicating if it has lost one branch)



if V' < value of y

- cut x from y, make  a separate tree
- while y is marked

  cut y from the parent, make  a separate tree

  unmark y

  y ← y's previous parent
- mark y if y ≠ root

Lemma: A subtree with root deg d must have ≥ F_{d+2} nodes

Fibonacci number

Proof. Let r be a node with deg d.

Suppose $y_1, \dots, y_d$ are its children, s.t. y_i became a child before y_{i+1} did.

Claim: y_i must have degree $\geq i-2$.

When y_i is merged with r, they have the same degree. deg r at the time $\geq i-1$.

Then at most one child of y_i may be removed due to decKey, deg of $y_i \geq i-2$.

Then prove by induction,

size of the subtree

$$\geq 1 + F_d + F_{d-1} + \cdots + F_1$$

$$= F_d + F_{d-1} + \cdots + F_2 + F_1 + F_2$$

$$= \cdots + F_3 + F_2 + F_3$$

$$= \cdots + F_4 + F_3 + F_4$$

$$\vdots$$

$$= F_{d+2}.$$

□

Potential argument.

define the potential function

$$\Phi = \#\text{trees} + 2 \cdot \#\text{marked nodes}$$

$\Phi = 0$ initially

$\Phi \geq 0$ always

We will prove:

for an operation,

$$(\text{actual time}) + \Phi_{new} - \Phi_{old} \leq \begin{cases} O(1) & \text{ins, decKey} \\ O(\log n) & \text{extractMin} \end{cases}$$

This implies the amortized time bound, since

$$\text{total time} = \sum_{\text{op}i} (\text{actual time of op}i)$$
$$\leq \sum_i \left((\text{actual time of op}i) + \Phi_i - \Phi_{i-1} \right)$$
$$\leq O(\#\text{ins} + \#\text{decKey}) + O(\#\text{extractMin} \cdot \log n)$$

$\Phi_0 = 0$

$\Phi_m \geq 0$

ins: actual = 1, #trees increases by 1

extractMin: let L = #trees,

actual = L, #trees decreases to $O(\log n)$

$\Phi_{new} - \Phi_{old} \leq -L + O(\log n)$.

decKey: let K = #subtrees cut iteratively

actual = K, #trees increases by K

#marked nodes decreases by $\geq K-2$.

$\Phi_{new} - \Phi_{old} \leq -K + O(1)$