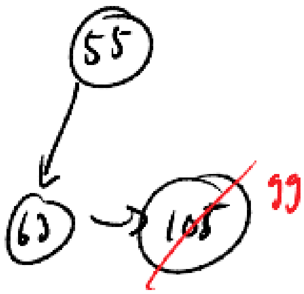
Lecture 5

Monday, January 21, 2013 1:06 PM

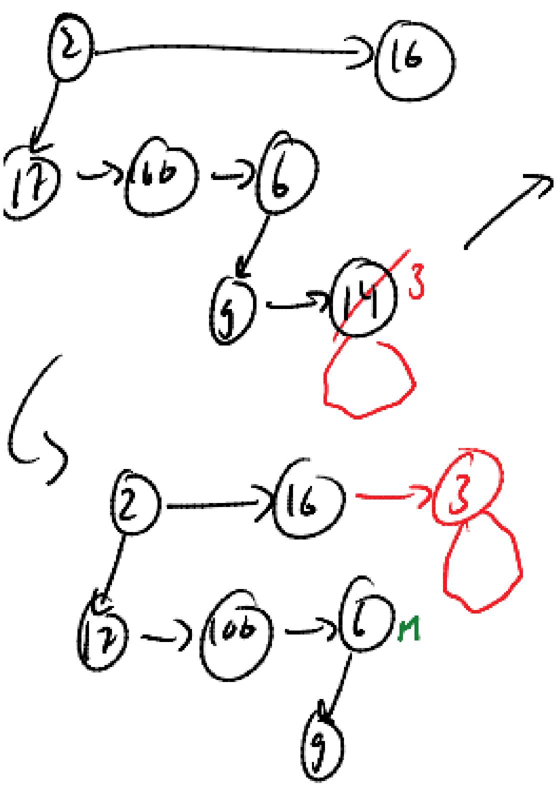
suite de lecture 4...

nb: d(H) is the maximum degree of any node
 d(h) E O(log(n))

decrease-key:

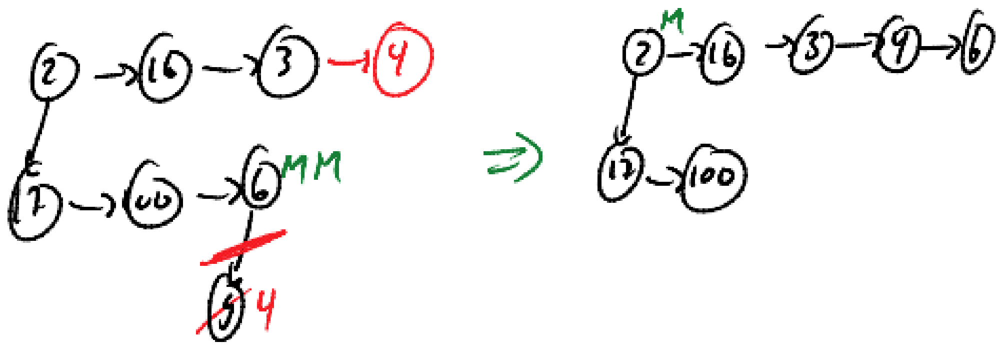


→ if we don't violate the heap property
 done!



if w

condition: no node is marked more than one
if we mark a node twice
 cut it out → move to the root list



repeat this as much as neccesary → until you read the root list
→ full ammortization potential function
ϕ(H) = t(H) + 2m(h) i.e 2 marked nodes
(not shown!) amortized cost of decrease key: O(1)

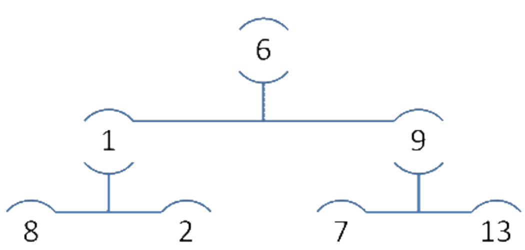
Delete:
→ decrease key and to smth smaller than min } O(log n) amortized
→ delete min }

op	Binary Heap	Fip heap(Amortized)
make-heap	θ(1)	θ(1)
insert	θ(log n)	θ(1)
find-min	θ(1)	θ(1)
delete-min	θ(log n)	θ(log n)
union	θ(n)	θ(1)
decrease-key	θ(log n)	θ(1)
delete	θ(log n)	θ(log n)

nb: asymptotically Fib heap wins
in pratice it depends Fib is more complex and constant is larger

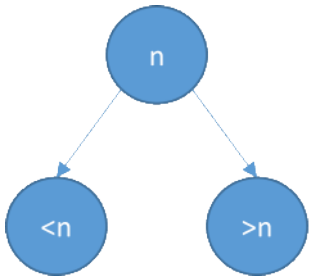
B-Trees

→ Familiar with a binary search tree.



Fully balanced O(log n)

Binary tree property



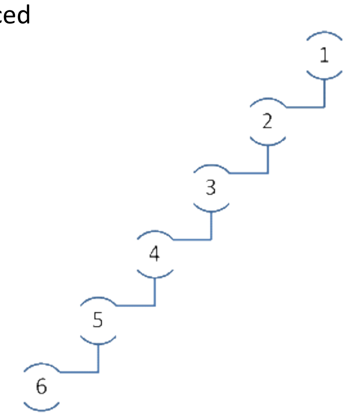
→ easy to search

  find 7 → compare at each node

  → found it

  → go down left/right branch

→ if we construct one naively! : 1,2,3,4,5,6
unbalanced



→ *O(N)*

self balancing binary search trees
B-Trees

  Guaranteed O(log n) cost for operation

    insert

    search

    delete

  → basic idea we use

  tableau + tree   keys are ordered $n_i < n_{i+1}$

  to search: binary search to find the key inside a node

    → find it

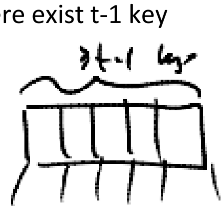
    → don't → recursive search the child where it would be

  insert → hard operation!

B tree: choose a t ≥2

  each node will have at least   keys and at most   keys

  there exist t-1 key



internal node have

- *≥ t children,*
- *≤ 2t children*

(on example: root can have fewer but always has at least 1)

most have at least t as a branching function
O(log t n) height
→ all lines have the same depth

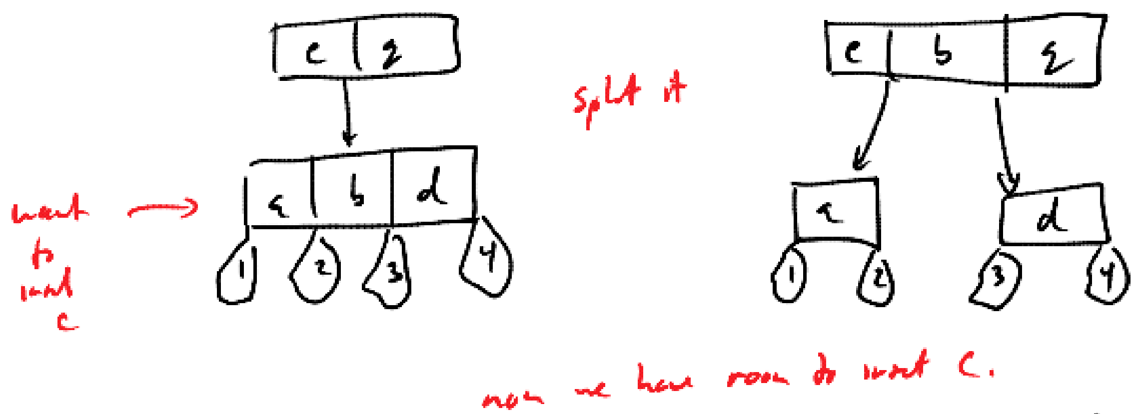
notice: t = 2, # of keys : 1, 2 or 3 keys in a node
eg:
  2-4 trees, 2-3-4 tree } same
  (nb: "2-3" tree → different)

Insert

Insert c

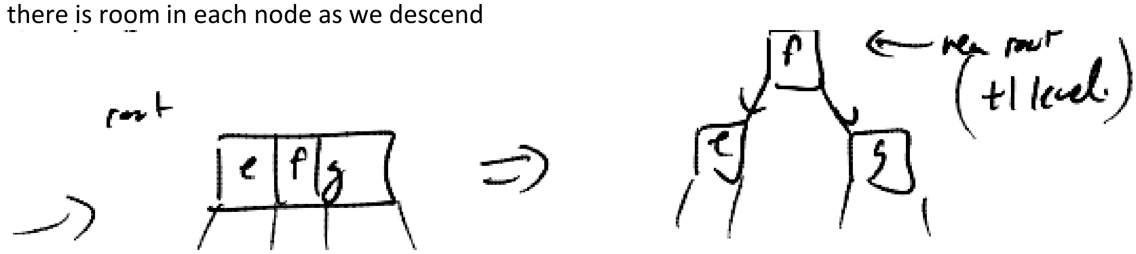
a	b	c	a	c	b	c	b	a

i.e if there is some room, just insert it while maintaining the key order
What if the node is full?



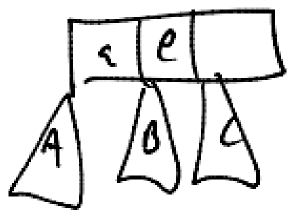
split it now we have room to insert c

trick: ensure there is room in each node as we descend

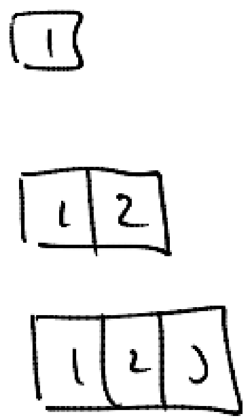


→ recursively descend, making room, assuming that there's room in the parent node
→ if a leaf:
  insert it
→ if not a leaf:

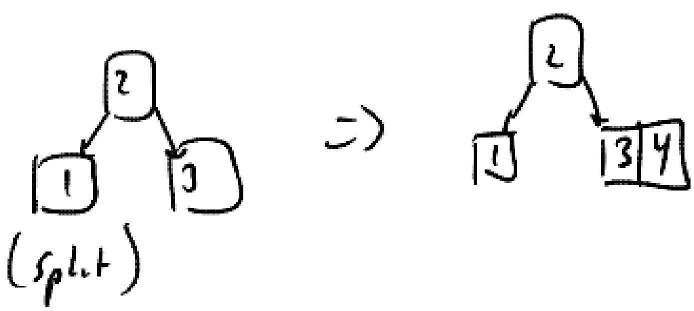
→ find which child it should go in
suppose it is B
if root B is full → split it
recursively descend



- insert: 1,2,3,4,5,6

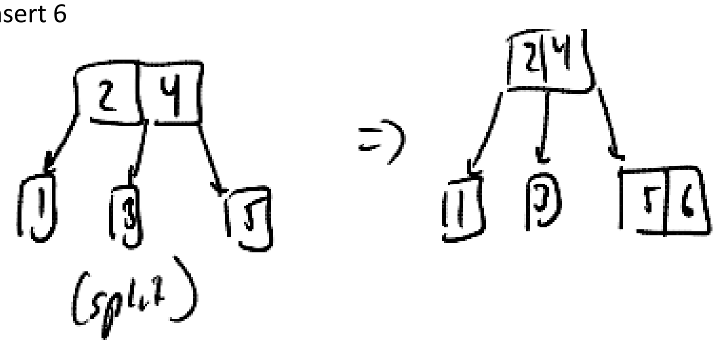


- insert 4:



- insert 5:

- insert 6



exercise: 2,4,3,1,5,6 → similar tree
→ same height

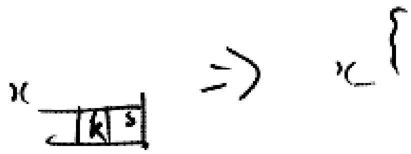
->nb: originally designed to help with efficient data I/O

Delete:

naïve delete → violate our B-tree properties
idea: restructure the tree as we move down to the node being deleted
ensure as we descend that we have at least 1 more key than the minimum re… (i.e ≥+keys)

Delete(k, x)

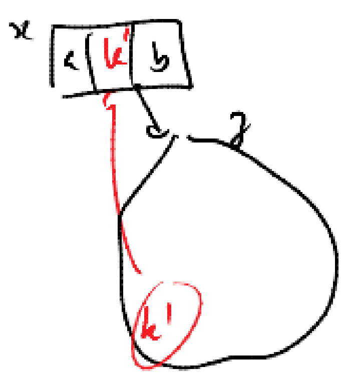
1. If k is in x, and x is a leaf node
   i. delete k from x



2. If k is in x and x is an node

if $|y| \geq t$

- If $|y| \geq t$, recursively delete k in x



- If $|z| \geq t$, symmetric if we do [?] child
- If neither y nor z has at least t keys I.e they both have t-1 keys