Module 4 - Dictionaries and Balanced Search Trees

Graham Cooper

June 2nd, 2015

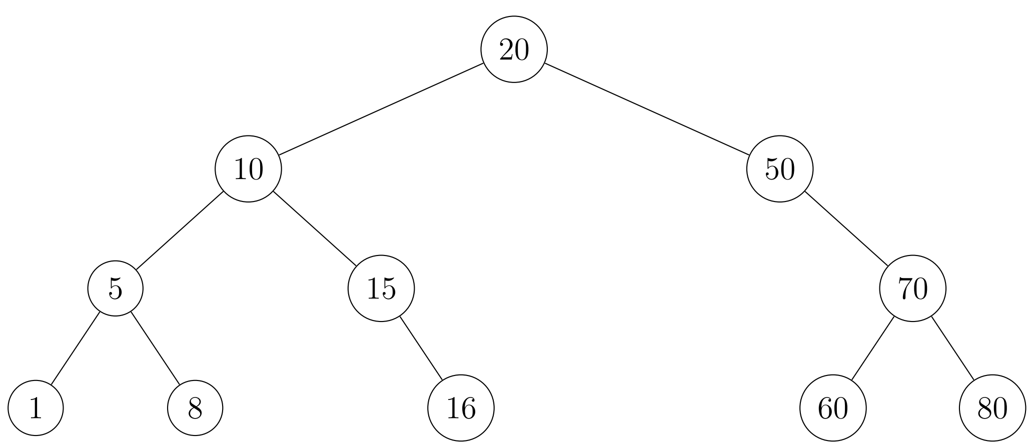
Dictionaries

- An ADT
- Data (key, value) pairs
- operations: search, insert, delete

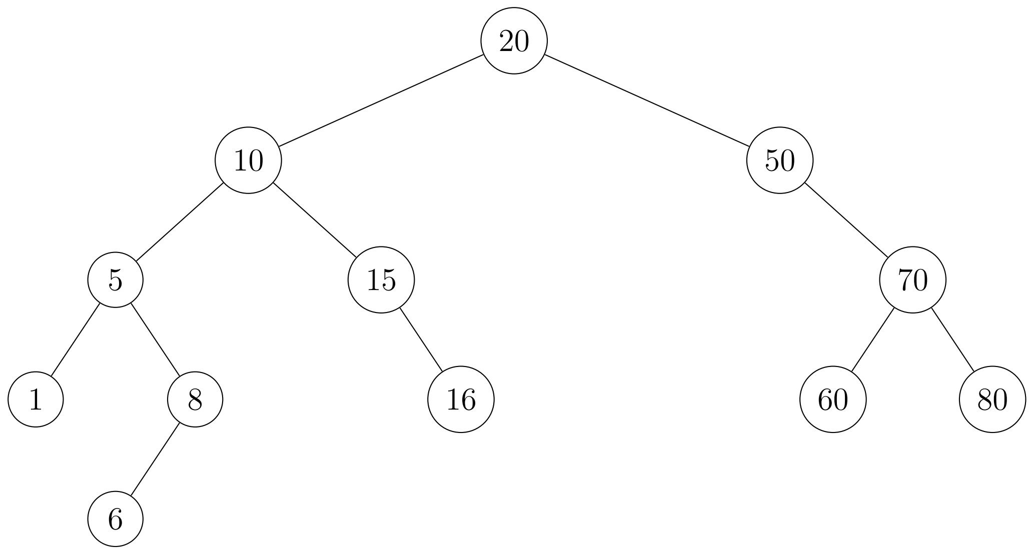
Data Structures for Dictionaries:

- unsorted array or linked list
  - search: O(n)
  - insert: O(1)
  - Delete: O(n)
- sorted array
  - search - binary search O(logn)
  - insert O(n)
  - delete O(n)

0.1 BST



Insert 6



Delete in a BST

- if n is a leaf - just delete it
- if n is a node with one child, replace it with its child
- if n has two children, replace with the predecessor (rightmost on the left) or sucessor (left most)

Fun with AVL trees (control)

insert(y)
- insert as a leaf like usual bst
—-Move up, update balance factors
—— if x.balance factor ∈ {-2, 2}
——— fin(x)
Fin is called at most once after that bf, are all fixed (no need to update higher levels)

fin(x)

if x.bf = -2 (too heavy on the left)
{ – if x.left.bf = 1 then
—- x.left → rotate(left)
– n → rotate Right }

if x.bf = +2 (too heavy on right)
{
– if x.right.bf = -1
—- x.right → rotateRight
– x → rotateLeft()
}

Delete(j)

– as usual BST, replcae with successor/predecessor
– move from location of seccessor, predecessor
– – move up
– – – if x.bf ← {-2,2}
– – – – fin(x)

fin may be called log(n) times because the height changes.

insertion

– Insert as a usual BST
– – O(height)
– move up check balance factor, apply fin() if neccessary
– time for fin → O(1)
In total, time for insert: Θ(height)

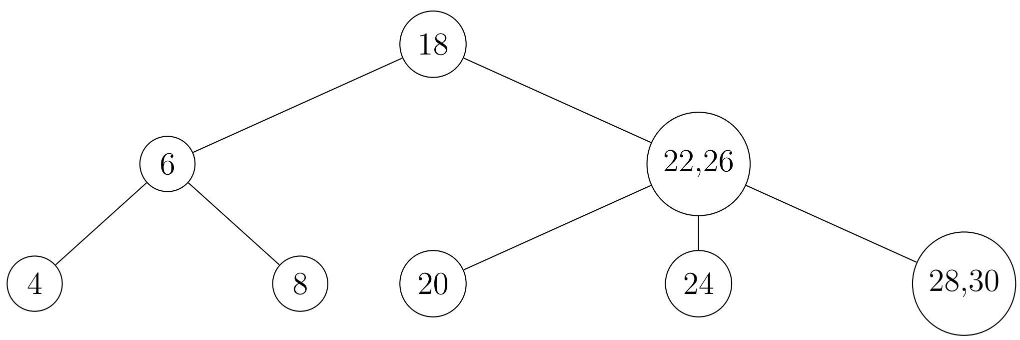
Height of AVL:

let N(h) denote the minumum number of nods in an AVL tree with height h.
N(h) =
0 if h = -1
1 if h = 0
N(h-1) + N(h-2) + 1 else

N(h) = fionacci(h+3) - 1

$= \text{roof}(\frac{p^{h+3}}{5}) - 1$
where p = $\frac{1+\sqrt{5}}{2}$

B-Tree (beautiful tree)



An (a,b) tree B-Tree

1. An ordered tree
2. Each internal node has at least a, and at most b children, root has at least 2, at most b children
3. A node with k children → k-1 key value pairs
   – An (roof($\frac{u}{2}, u$)) B-tree is order u B-tree, eg u = 2→ order b-tree → A(2,3)-tree

Insertion
–Insert at a leaf – overfilled noes send the middle key to the parent and split
Deletion
– As BST, the removed key is replaced by successor/predeccesor (which is a leaf)
– if a node becomes underloaded
– – if ∃ a sibling with an extra key (more than 'a' keys)
– – – take the key from parent and parent gets a key from the sibling.

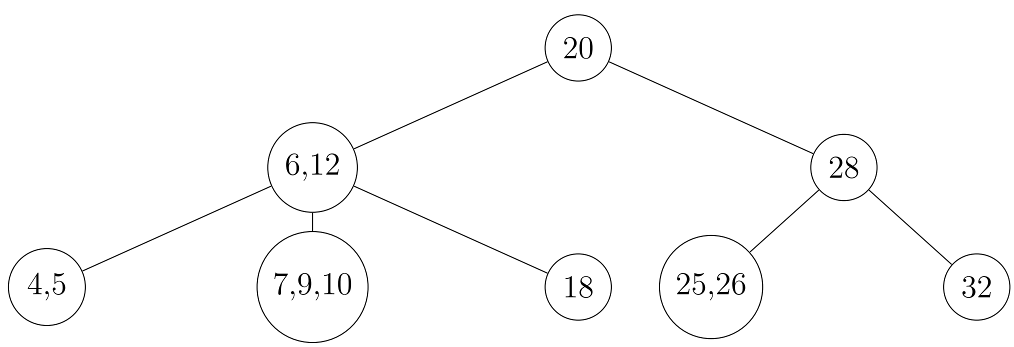
(a-b) B-tree

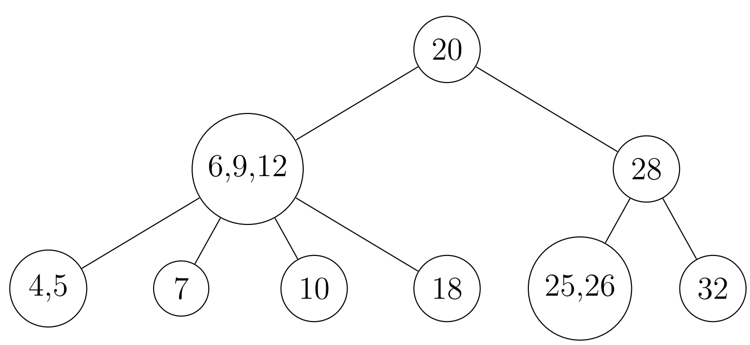
- each internal node has at least a and at most b children
- the root has at least 2 and at most b children
- minimum number of key value pairs in a node is at least a-1 and at most b-1 except root which has at least 1 key value pair.

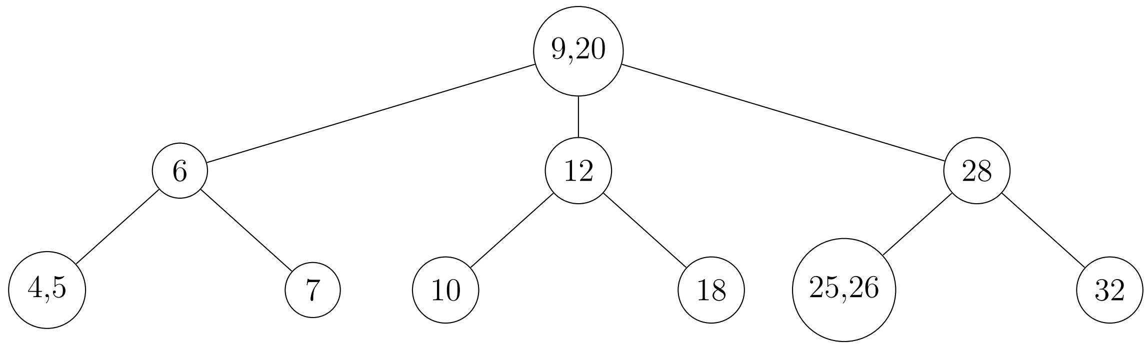
if $a = roof(\frac{M}{2})$ $b = M$ - order M B-tree
M = S $\implies$ a = 2 and b = 3
at least 1 kvp and at most 2 kvp
This particular B-tree is a 2-3 tree

a = 2 and b = 2
at least 1 kvp/node and at most 1 kvp/node
This is just a regular bst

Insertion in a B-tree








```
insert as BST
if node is overloaded
-- split send midkey to parent
```

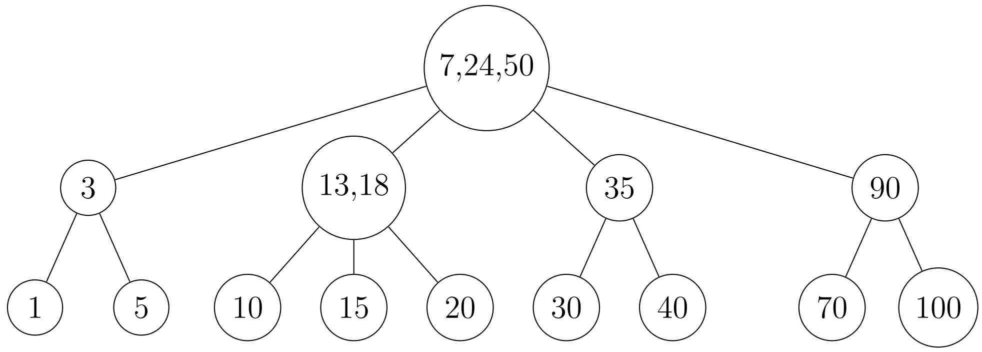
b-max number of kvps in a node
– following the right link take log(b)
– insertion takes O(hxlog(b))

Deletion from a B-tree

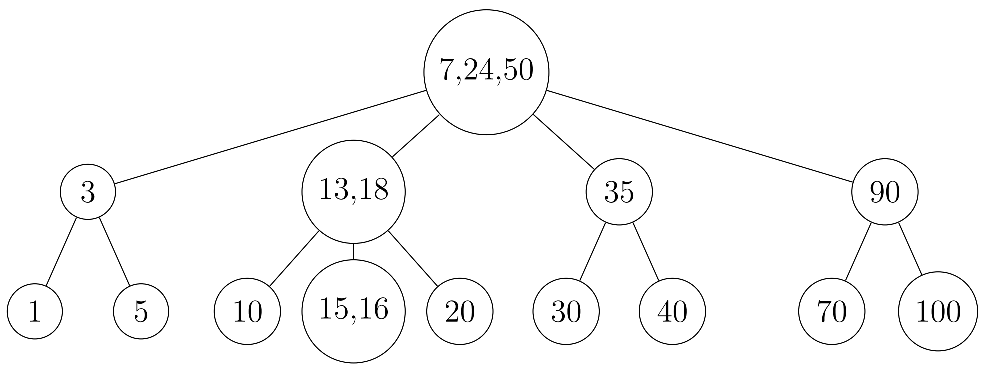
```
delete as usual from BST
if there exists an underloaded node x
-- if there exists a direct sibling of x with extra keys
-- -- borrow a key through parent
-- else (when direct siblings have no extra keys)
-- -- merge them
-- -- take parent down
```

Missed the example :(

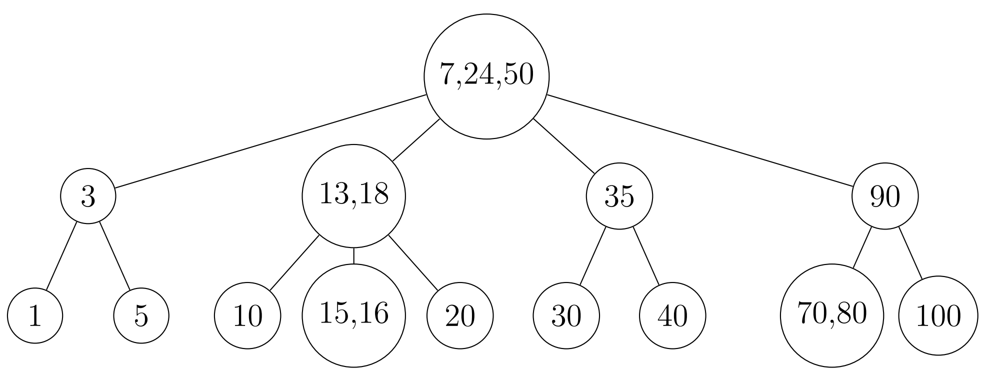
$a = 2$, $b = 4$



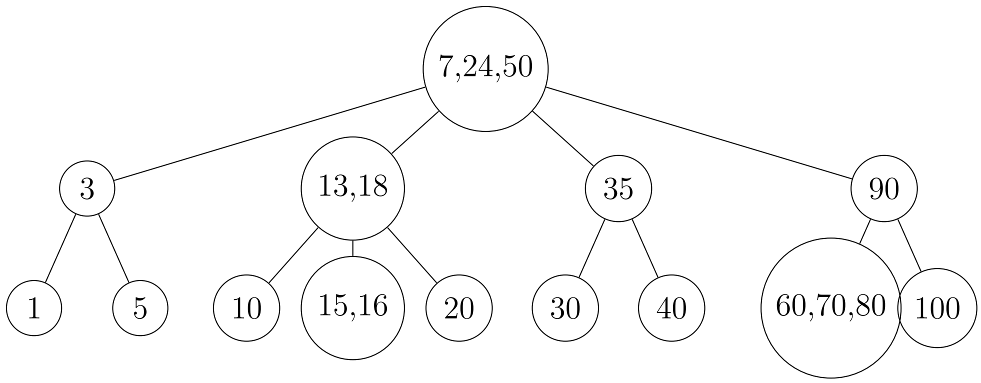
insert(16)



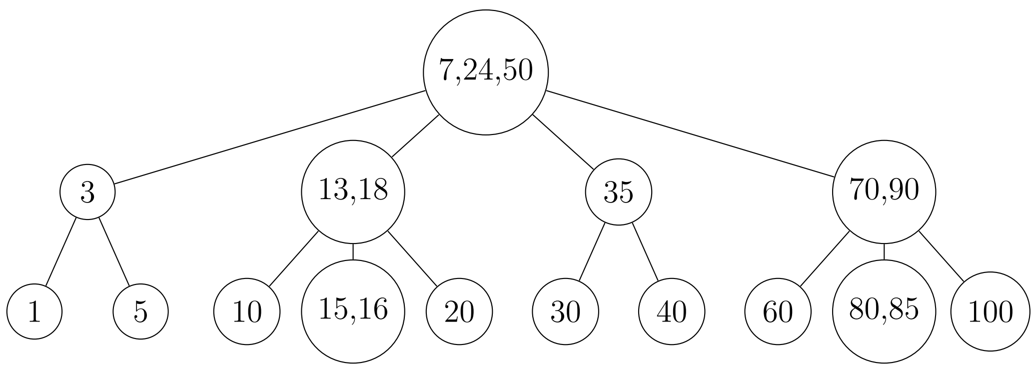
insert(80)



insert(60)



insert(85)



Height of a B-tree of order n

level	nodes per level	links/node	kvp/node	kvp on level
0	1	≥ 2	≥ 1	≥ 1
1	2	$roof(\frac{n}{2})$	$roof(\frac{n}{2}) - 1$	$2(roof(\frac{n}{2}) - 1)$
2	$2(roof(\frac{n}{2}))$	$roof(\frac{n}{2})$	$roof(\frac{n}{2}) - 1$	$2\frac{n}{2}(\frac{n}{2} - 1)$

$$n \geq 1 + 2(\frac{m}{2} - 1)\sum_{i=0}^{h-1}(\frac{m}{2})^i$$

$$= 1 + 2(\frac{m}{2} - 1)(\frac{(\frac{m}{2})^h - 1}{(\frac{m}{2} - 1)})$$

$$= (\frac{m^h}{2} - 1)$$

$$n \geq (\frac{m}{2})^h - 1$$

$$log(h+1) \geq hlog(\frac{n}{2})$$

$$\implies h \in O(\frac{logn}{logm})$$

insertion, selection, search
O(logb * h) = O(logM * h)
= O(logm * $\frac{logn}{logm}$) $= O(logn)$

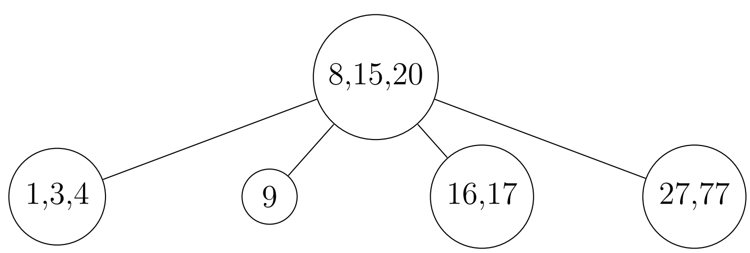
In RAM → number of primitive operations → O(logn) search/insert/delete
if data is huge
→ does not fit in RAM
→ need disk access
→ we need to minimize number of disk accessses

In AVL trees
→ might needl ogn disk access (height of the tree) → if all M items in cache
node fit in a block → $fraclognlogm$ disk accesses height of he tree

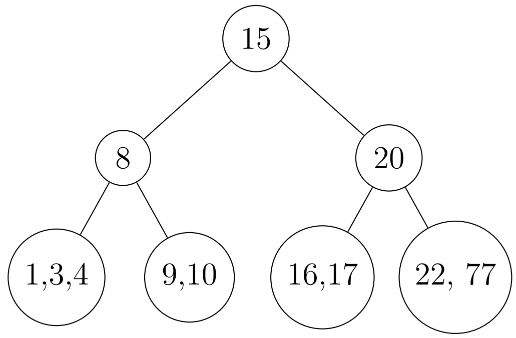
pre-emptive splitting:

- when inserting, split any node with main number of key value pairs

(2,4)



insert(10)



Red/Black Trees

- less structures than AVL
- Less "rotation"
- if insert/delete and search are equaly likely, red/black tree is better
- if search is more likely, AVL has a better time