Map Implementation Summary

or set

	unsorted list (array/linked)	sorted array	sorted linked list	hash table	balanced BST
contains	O(n)	O(log n)	O(n)	O(1) expected* O(n) worst	O(log n)
put / add	O(n)	O(log n) if key present O(n) worst	O(n)	O(1) expected* O(n) worst	O(log n)
remove	O(n)	O(n)	O(n)	O(1) expected* O(n) worst	O(log n)
iterate	O(n)	O(n)	O(n)	O(n)	O(n)
sorted iterate	O(n log n)	O(n)	O(n)	O(n log n)	O(n)

sequential search

move

* assuming $\alpha \leq c$ and hashes of keys uniform

keys : disjoint intervals of integers
values : whatever

(10, 20) → jrg 94
(30, 60) → kg 301
(61, 100) → jrg 94
(105, 170) → kg 301

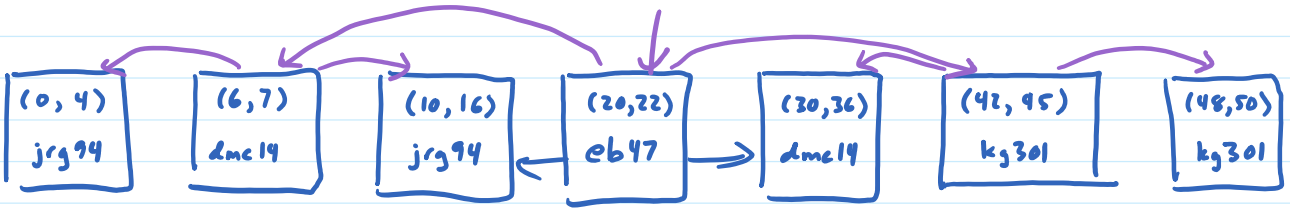
get : given int x, find value for interval containing x

get(36) = kg301

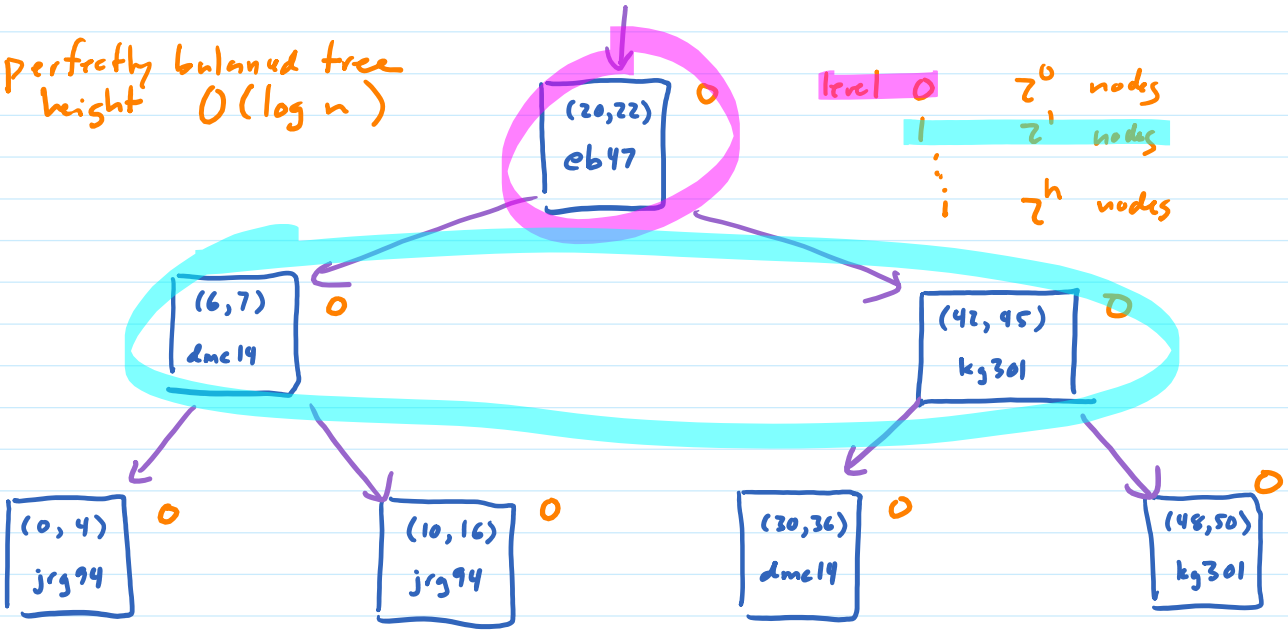
need keys in sorted order

Binary Search Trees

key = 32



perfectly balanced tree
height $O(\log n)$



level 0 2^0 nodes
1 2^1 nodes
i 2^h nodes

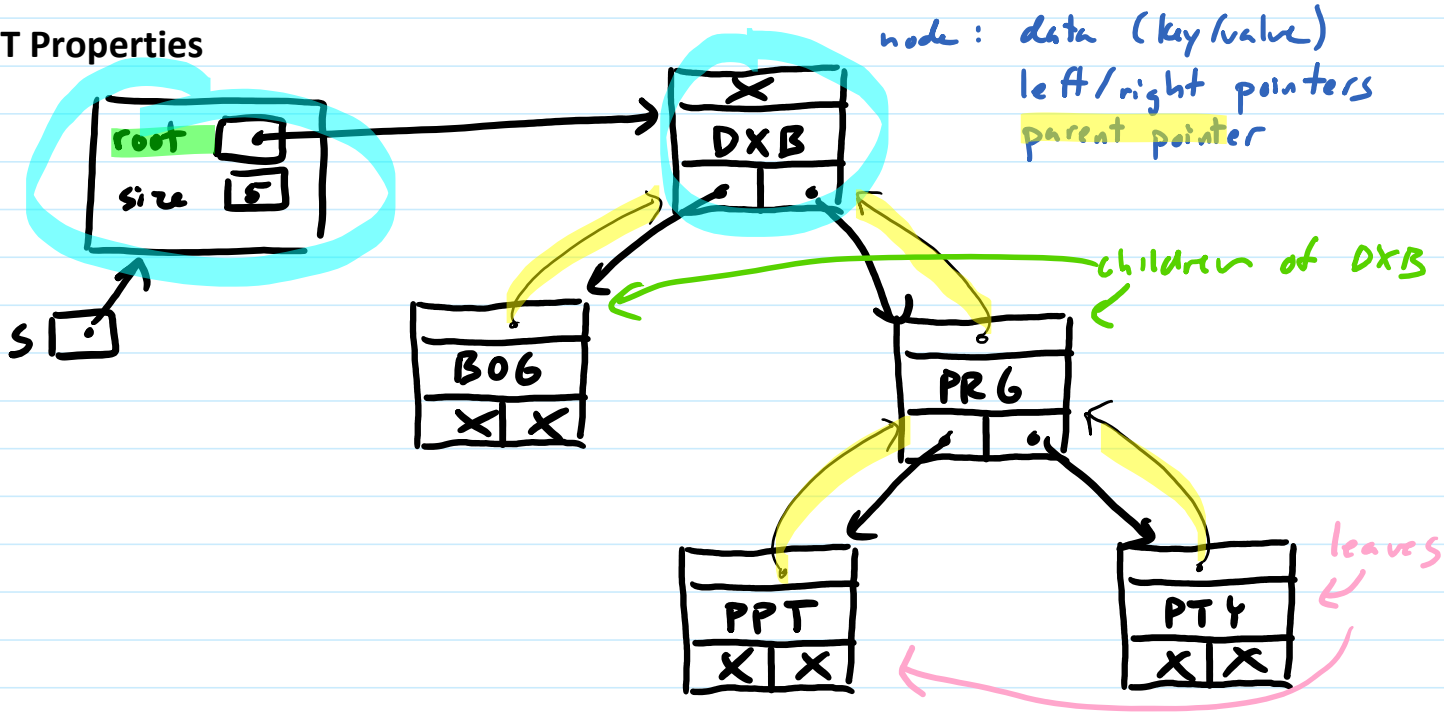
sum leaf 0...i $= 2^{h+1} - 1$

$2^{h+1} - 1 = n$

$h = \log_2(n+1) - 1$

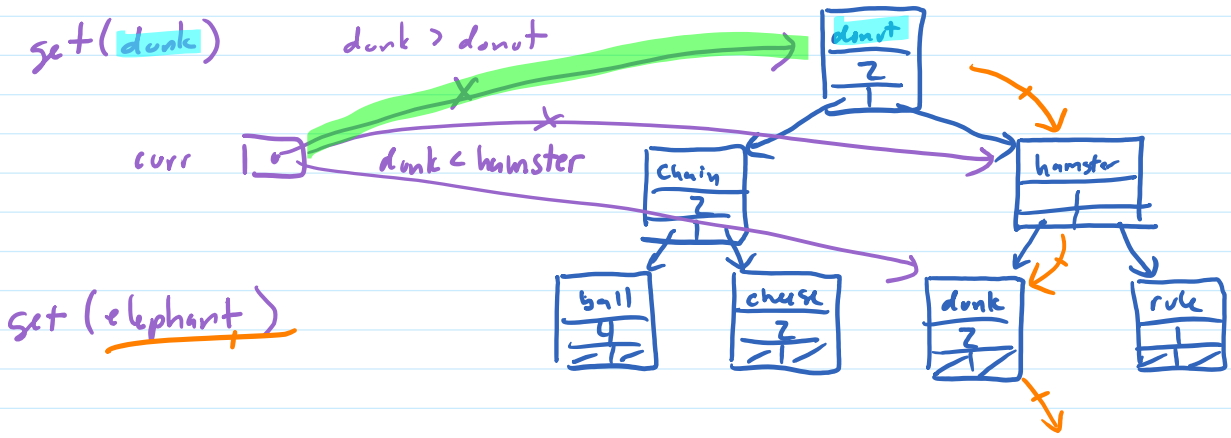
h is $O(\log n)$

BST Properties



Properties: everything in left subtree of node n has key < n's key
" right " " > "

Searching in a BST



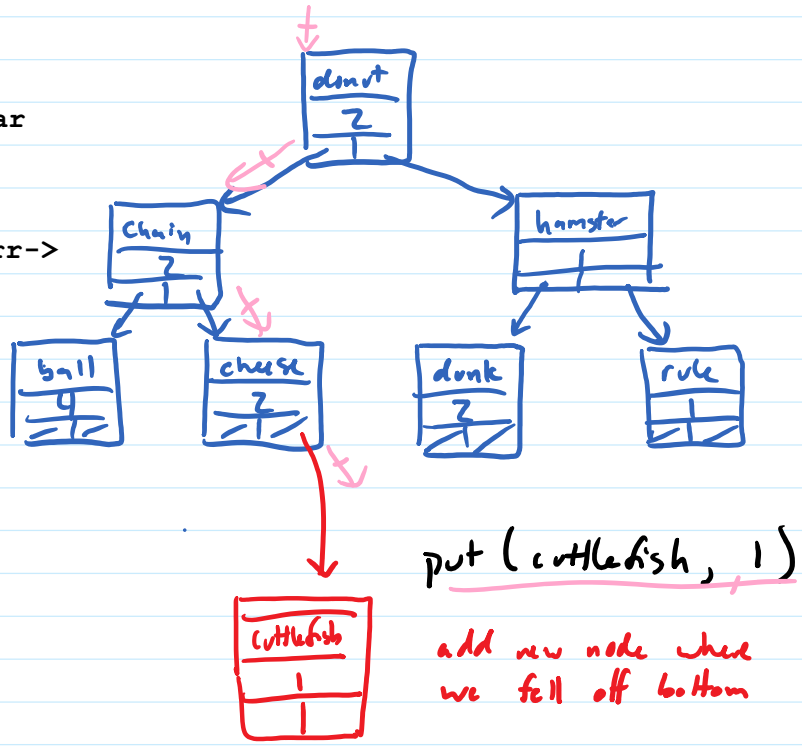

```
bool smap_contains_key(smap *m, const char *key)
{
   smap_node *curr = m->root;
   while (curr != NULL && strcmp(key, curr->key) != 0)
   {
      if (strcmp(key, curr->key) < 0)
      {
         curr = curr->left;
      }
      else
      {
         curr = curr->right;
      }
   }
   return (curr != NULL);
}
```

← not out of places to look and not at what we're looking for

why did we stop?

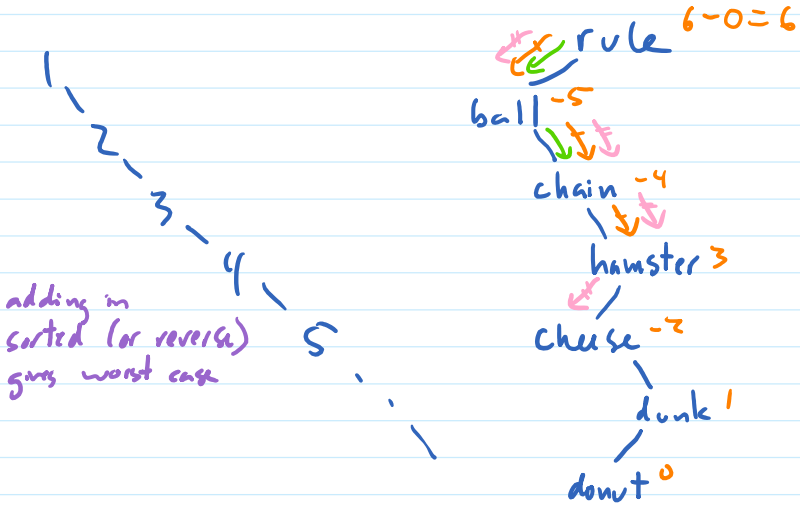
Adding to a BST

```
bool smap_contains_key(smap *m, const char
*key)
{
   smap_node *curr = m->root;
   while (curr != NULL && strcmp(key, curr->
   key) != 0)
   {
      if (strcmp(key, curr->key) < 0)
      {
         curr = curr->left;
      }
      else
      {
         curr = curr->right;
      }
   }
   return (curr != NULL);
}
```



Unshapely Trees

rule ball chain hamster cheese dunk donut

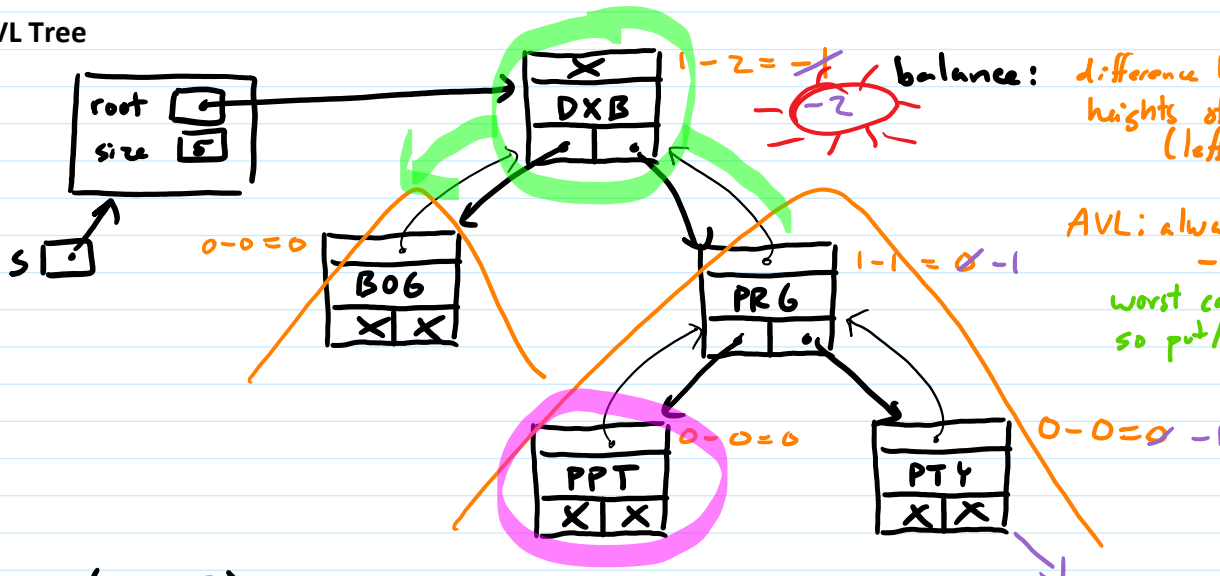


worst case height = n

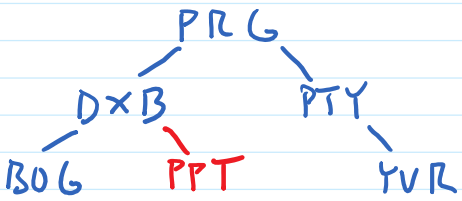
put/contains/get O(h) one iteration per level O(1) per iter

worst case O(n)

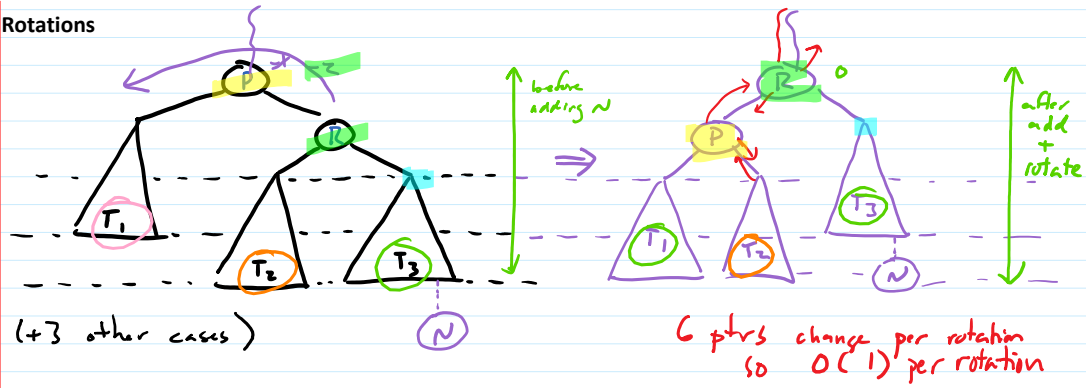
AVL Tree

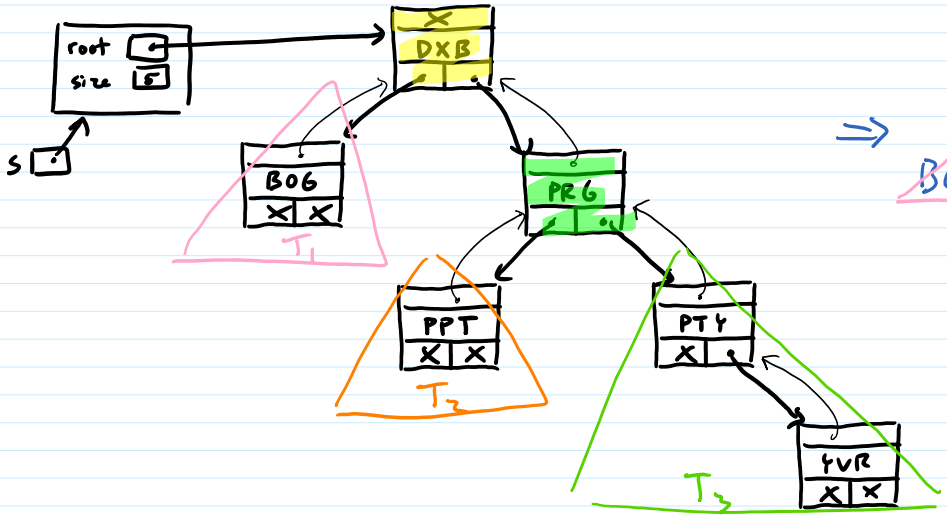


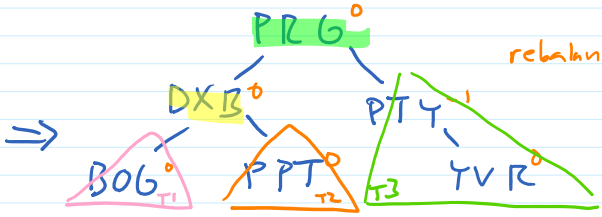
add(s, YVR)



Rotations


before adding N
after add + rotate
(+3 other cases)
6 ptrs change per rotation
so O(1) per rotation


root
size
S
DXB
BOG
PRG
PPT
PTY
YVR
T1
T2
T3


PRG
DXB
PTY
BOG
PPT
YVR
rebalanced!