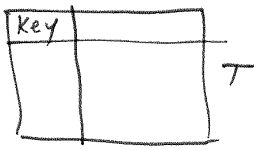
Dictionary ADT (Abstract Data Type)



Operations: Insert/Add $(x : T)$

Condition: At most one element with a given key value.

class compareTo is based on key value.

Find / contains $(x : T)$ — Look for an object with given key value.

Delete / Remove $(x : T)$ — Remove object x from dictionary.

Min() / Max() — Return object with min/max value of key.

Range Search (k_1, k_2) — Return a set of objects whose key is between k_1 and k_2.

Successor / Predecessor (x) — Find object whose key is next to key of x in sorted order of keys.

Common implementation: TreeMap : Key ⟶ T
↑ Red-Black trees.

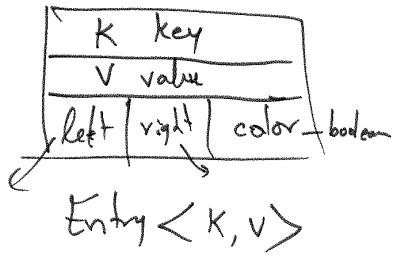
Red-Black Trees — a binary search tree.

Size of dictionary = n

Height of RB tree = O(log n)

Max height ≤ 2·log n.

Constants
Red / Black
true / false



- Instead of having null pointers, if you set left/right = NullEntry, then many edge cases can be avoided.
↑ colored black.

Rules for a valid RB tree

1. Root is black.
2. Two red nodes cannot be adjacent to each other.
   ⇒ a red node always has a black parent
3. From any node of the tree, every path to a descendant leaf has the same number of black nodes — black height of a node.

---

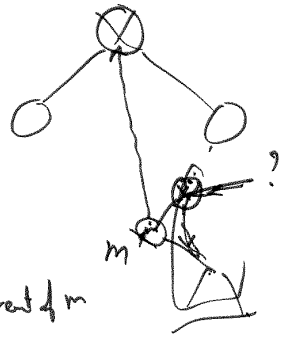
black height of a tree with n nodes ≤ log n.

find / contains — same as BST.

add(x): add x to tree just like BST.
color new node red.
If x and its parent are both colored red, then
call Repair(x).

remove(x): same as bst.

m is node that is removed.
if color(m) = Red — done
if color(m) = ~~Red~~ Black — call Fix(?). ← parent of m



Precondition for Fix(x): subtree at x
is deficient by 1 in its black height.

RT of Red-Black tree operations: $O(\log n)$ per operation
where n = # of elements stored
(size of dictionary)

Q: What can we do to make some ops run in $O(1)$ time?
e.g. min, max, succ, pred. given "Entry" of element x.

Ans: threaded BST.

↳ requires rewriting add/remove.

element	. .
left	right
succ	pred

Entry

Challenges: (i) Write algorithms for BST class
succ(x), pred(x)
(ii) Write an iterator for BST class.

Multidimensional Search

class Invoice: Ino – Invoice number – long

Itemno — key of class Item – long

Qty – int

Price – Price per unit $$$.¢¢

Other fields, – T.

Normal mode: Given Ino: add, remove, contains, find ..

Ans: HashMap, TreeMap: Ino → Invoice

Additional queries:

Given an item no, find the set of invoices with that item.

Build additional access structures as needed:

TreeMap <Item → Set <Invoice>>

Given an Ino, find top K items of value in that invoice.

TreeMap <Ino → ? >.