1. -- Implementation of deletion for red black trees by Matt Might
3. -- Original available from:
4. -- http://matt.might.net/articles/red-black-delete/code/RedBlackSet.hs
5. -- Slides:
6. -- http://matt.might.net/papers/might2014redblack-talk.pdf
7. -- Draft paper:
8. -- http://matt.might.net/tmp/red-black-pearl.pdf
10. {-# LANGUAGE TypeApplications #-}
11. {-@ LIQUID "--no-termination" @-}
12. {-@ LIQUID "--no-totality" @-}
13. {-@ LIQUID "--exact-data" @-}
15. module MightRedBlack where
17. import Prelude hiding (max)
18. import Control.Monad
19. import Test.QuickCheck hiding (elements)
20. import Data.List(nub,sort)
22. data Color =
23. R -- red
24. | B -- black
25. | BB -- double black
26. | NB -- negative black
27. deriving (Show, Eq)
29. data RBSet a =
30. E -- black leaf
31. | EE -- double black leaf
32. | T Color (RBSet a) a (RBSet a)
33. deriving (Show, Eq)
35. -- Private auxiliary functions --
37. {-@ redden :: {x:CT a | color’ x == B}
38. -> {v:IM a | blackHeightL v == (blackHeightL x - 1) } @-}
39. redden :: RBSet a -> RBSet a
40. redden (T \_ a x b) = T R a x b
42. {-@ blacken’ :: IM a -> RT a @-}
43. -- blacken for insert
44. -- never a leaf, could be red or black
45. blacken’ :: RBSet a -> RBSet a
46. blacken’ (T R a x b) = T B a x b
47. blacken’ (T B a x b) = T B a x b
49. -- blacken for delete
50. -- root is never red, could be double black
51. {-@ blacken :: IM a -> RT a @-}
52. blacken :: RBSet a -> RBSet a
53. blacken (T B a x b) = T B a x b
54. blacken (T BB a x b) = T B a x b
55. blacken E = E
56. blacken EE = E
58. {-@ isBB :: rb : RBSet a -> { b : Bool | b <=> isBB’ rb } @-}
59. isBB :: RBSet a -> Bool
60. isBB EE = True
61. isBB (T BB \_ \_ \_) = True
62. isBB \_ = False
64. {-@ blacker :: {x:Color | not tooBlack x}
65. -> {v:Color | colorValue v == (colorValue x + 1)} @-}
66. blacker :: Color -> Color
67. blacker NB = R
68. blacker R = B
69. blacker B = BB
70. blacker BB = error "too black"
72. {-@ redder :: {x:Color | not tooRed x}
73. -> {v:Color | (colorValue v == (colorValue x - 1))
74. && ((x == BB) => (v == B)) } @-}
75. redder :: Color -> Color
76. redder NB = error "not black enough"
77. redder R = NB
78. redder B = R
79. redder BB = B
81. {-@ blacker’ :: {x:RBSet a | canBeBlacker x} -> RBSet a @-}
82. blacker’ :: RBSet a -> RBSet a
83. blacker’ E = EE
84. blacker’ (T c l x r) = T (blacker c) l x r
86. {-@ redder’ :: {x:RBSet a | (prop\_IM x && isBB’ x) || (prop\_CT x)}
87. -> {v:RBSet a | ((prop\_IM x && isBB’ x && prop\_CT v) ||
88. (prop\_CT x && prop\_IM v))
89. && (blackHeightL v == (blackHeightL x - 1)) } @-}
90. redder’ :: RBSet a -> RBSet a
91. redder’ EE = E
92. redder’ (T c l x r) = T (redder c) l x r
94. -- `balance` rotates away coloring conflicts:
95. {-@ balance :: c:Color
96. -> {l:RBSet a | (prop\_IM l) || (prop\_CT l) }
97. -> x:a
98. -> {r:RBSet a | ((prop\_IM l && prop\_CT r) ||
99. (prop\_CT l && prop\_IM r))
100. && (blackHeightL l == blackHeightL r)}
101. -> {v:RBSet a | ((c /= B && prop\_IM v) || prop\_CT v)
102. && (blackHeightL v == (blackHeightL l + colorValue c))}
103. @-}
104. balance :: Color -> RBSet a -> a -> RBSet a -> RBSet a
106. -- Okasaki’s original cases:
107. balance B (T R (T R a x b) y c) z d = T R (T B a x b) y (T B c z d)
108. balance B (T R a x (T R b y c)) z d = T R (T B a x b) y (T B c z d)
109. balance B a x (T R (T R b y c) z d) = T R (T B a x b) y (T B c z d)
110. balance B a x (T R b y (T R c z d)) = T R (T B a x b) y (T B c z d)
112. -- Six cases for deletion:
113. balance BB (T R (T R a x b) y c) z d = T B (T B a x b) y (T B c z d)
114. balance BB (T R a x (T R b y c)) z d = T B (T B a x b) y (T B c z d)
115. balance BB a x (T R (T R b y c) z d) = T B (T B a x b) y (T B c z d)
116. balance BB a x (T R b y (T R c z d)) = T B (T B a x b) y (T B c z d)
118. balance BB a x (T NB (T B b y c) z d@(T B \_ \_ \_))
119. = T B (T B a x b) y (balance B c z (redden d))
120. balance BB (T NB a@(T B \_ \_ \_) x (T B b y c)) z d
121. = T B (balance B (redden a) x b) y (T B c z d)
123. balance color a x b = T color a x b
125. -- `bubble` "bubbles" double-blackness upward:
126. {-@ bubble :: {c:Color | not (tooBlack c)}
127. -> {l:RBSet a | prop\_CT l || prop\_IM l}
128. -> x:a
129. -> {r:RBSet a | ((prop\_CT l && prop\_IM r) ||
130. (prop\_IM l && prop\_CT r))
131. && (blackHeightL l == blackHeightL r) }
132. -> {v:IM a | blackHeightL v == blackHeightL l + (colorValue c) }
133. @-}
134. bubble :: Color -> RBSet a -> a -> RBSet a -> RBSet a
135. bubble color l x r
136. | isBB(l) || isBB(r) = balance (blacker color) (redder’ l) x (redder’ r)
137. | otherwise = balance color l x r



142. -- Public operations --
144. {-@ empty :: {v:RT a | normalLeaf v} @-}
145. empty :: RBSet a
146. empty = E

149. {-@ member :: (Ord a) => a -> CT a -> Bool @-}
150. member :: (Ord a) => a -> RBSet a -> Bool
151. member x E = False
152. member x (T \_ l y r) | x < y = member x l
153. | x > y = member x r
154. | otherwise = True
156. {-@ insert :: (Ord a) => a -> RT a -> RT a @-}
157. {-@ ins :: (Ord a) => x:CT a -> {v:IM a | blackHeightL v == blackHeightL x} @-}
158. insert :: (Ord a) => a -> RBSet a -> RBSet a
159. insert x s = blacken’ (ins s)
160. where ins E = T R E x E
161. ins s@(T color a y b) | x < y = balance color (ins a) y b
162. | x > y = balance color a y (ins b)
163. | otherwise = s


167. {-@ max :: {x:CT a | not normalLeaf x} -> a @-}
168. max :: RBSet a -> a
169. max E = error "no largest element"
170. max (T \_ \_ x E) = x
171. max (T \_ \_ x r) = max r
173. -- Remove this node: it might leave behind a double black node
174. {-@ remove :: {x:CT a | not normalLeaf x}
175. -> {v:IM a | blackHeightL v == blackHeightL x} @-}
176. remove :: RBSet a -> RBSet a
177. -- remove E = E -- impossible!
178. -- ; Leaves are easiest to kill:
179. remove (T R E \_ E) = E
180. remove (T B E \_ E) = EE
181. -- ; Killing a node with one child;
182. -- ; parent or child is red:
183. -- remove (T R E \_ child) = child
184. -- remove (T R child \_ E) = child
185. remove (T B E \_ (T R a x b)) = T B a x b
186. remove (T B (T R a x b) \_ E) = T B a x b
187. -- ; Killing a black node with one black child:
188. -- remove (T B E \_ child@(T B \_ \_ \_)) = blacker’ child
189. -- remove (T B child@(T B \_ \_ \_) \_ E) = blacker’ child
190. -- ; Killing a node with two sub-trees:
191. remove (T color l y r) = bubble color l’ mx r
192. where mx = max l
193. l’ = removeMax l
195. {-@ removeMax :: {x:CT a | not normalLeaf x}
196. -> {v:IM a | blackHeightL v == blackHeightL x} @-}
197. removeMax :: RBSet a -> RBSet a
198. removeMax E = error "no maximum to remove"
199. removeMax s@(T \_ \_ \_ E) = remove s
200. removeMax s@(T color l x r) = bubble color l x (removeMax r)
202. {-@ delete :: (Ord a) => a -> x:RT a -> v:RT a @-}
203. delete :: (Ord a) => a -> RBSet a -> RBSet a
204. delete x s = blacken (del x s)
206. {-@ del :: Ord a => a
207. -> x:CT a
208. -> {v:IM a | blackHeightL v == blackHeightL x} @-}
209. del x E = E
210. del x s@(T color a’ y b’) | x < y = bubble color (del x a’) y b’
211. | x > y = bubble color a’ y (del x b’)
212. | otherwise = remove s
214. {-@ prop\_del :: Int -> RT Int -> Bool @-}
215. prop\_del :: Int -> RBSet Int -> Bool
216. prop\_del x s = color (del x s) `elem` [B, BB]

219. --- Testing code
221. {-@ elements :: Ord a => RT a -> [a] @-}
222. elements :: Ord a => RBSet a -> [a]
223. elements t = aux t [] where
224. aux E acc = acc
225. aux (T \_ a x b) acc = aux a (x : aux b acc)
227. instance (Ord a, Arbitrary a) => Arbitrary (RBSet a) where
228. arbitrary = liftM (foldr @[] insert empty) arbitrary
230. {-@ prop\_BST :: x:RT Int -> {v:Bool | v <=> prop\_BST’ x} @-}
231. prop\_BST :: RBSet Int -> Bool
232. prop\_BST t = isSortedNoDups (elements t)
234. {-@ color :: x:RBSet a -> {c:Color | c = color’ x} @-}
235. color :: RBSet a -> Color
236. color (T c \_ \_ \_) = c
237. color E = B
238. color EE = BB
240. {-@ prop\_Rb2 :: x:RT Int -> {v:Bool | v <=> blackRoot x} @-}
241. prop\_Rb2 :: RBSet Int -> Bool
242. prop\_Rb2 t = color t == B
244. {-@ prop\_Rb3 :: x:RT Int -> {v:Bool | v <=> validBlackHeight x} @-}
245. prop\_Rb3 :: RBSet Int -> Bool
246. prop\_Rb3 t = fst (aux t) where
247. aux E = (True, 0)
248. aux (T c a x b) = (h1 == h2 && b1 && b2, if c == B then h1 + 1 else h1) where
249. (b1 , h1) = aux a
250. (b2 , h2) = aux b
252. {-@ prop\_Rb4 :: x:RBSet Int -> {v:Bool | v <=> redChildIsBlack x} @-}
253. prop\_Rb4 :: RBSet Int -> Bool
254. prop\_Rb4 E = True
255. prop\_Rb4 (T R a x b) = color a == B && color b == B && prop\_Rb4 a && prop\_Rb4 b
256. prop\_Rb4 (T B a x b) = prop\_Rb4 a && prop\_Rb4 b

259. isSortedNoDups :: Ord a => [a] -> Bool
260. isSortedNoDups x = nub (sort x) == x

263. {-@ prop\_delete\_spec1 :: RT Int -> Bool @-}
264. prop\_delete\_spec1 :: RBSet Int -> Bool
265. prop\_delete\_spec1 t = all (\x -> not (member x (delete x t))) (elements t)
267. {-@ prop\_delete\_spec2 :: RT Int -> Bool @-}
268. prop\_delete\_spec2 :: RBSet Int -> Bool
269. prop\_delete\_spec2 t = all (\(x,y) -> x == y || (member y (delete x t))) allpairs where
270. allpairs = [ (x,y) | x <- elements t, y <- elements t ]
272. {-@ prop\_delete\_spec3 :: RT Int -> Int -> Property @-}
273. prop\_delete\_spec3 :: RBSet Int -> Int -> Property
274. prop\_delete\_spec3 t x = not (x `elem` elements t) ==> (delete x t == t)
276. {-@ prop\_delete\_bst :: RT Int -> Bool @-}
277. prop\_delete\_bst :: RBSet Int -> Bool
278. prop\_delete\_bst t = all (\x -> prop\_BST (delete x t)) (elements t)
280. {-@ prop\_delete2 :: RT Int -> Bool @-}
281. prop\_delete2 :: RBSet Int -> Bool
282. prop\_delete2 t = all (\x -> prop\_Rb2 (delete x t)) (elements t)
284. {-@ prop\_delete3 :: RT Int -> Bool @-}
285. prop\_delete3 :: RBSet Int -> Bool
286. prop\_delete3 t = all (\x -> prop\_Rb3 (delete x t)) (elements t)
288. {-@ prop\_delete4 :: RT Int -> Bool @-}
289. prop\_delete4 :: RBSet Int -> Bool
290. prop\_delete4 t = all (\x -> prop\_Rb4 (delete x t)) (elements t)
292. check\_insert = do
293. putStrLn "BST property"
294. quickCheck prop\_BST
295. putStrLn "Root is black"
296. quickCheck prop\_Rb2
297. putStrLn "Black height the same"
298. quickCheck prop\_Rb3
299. putStrLn "Red nodes have black children"
300. quickCheck prop\_Rb4
302. check\_delete = do
303. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete\_spec1
304. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete\_spec2
305. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete\_spec3
306. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete2
307. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete3
308. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete4
309. quickCheckWith (stdArgs {maxSuccess=100}) prop\_delete\_bst

312. main :: IO ()
313. main =
314. do
315. return $! ()
317. -- Helper functions for verification
319. {-@ measure color’ @-}
320. color’ :: RBSet a -> Color
321. color’ (T c \_ \_ \_) = c
322. color’ E = B
323. color’ EE = BB
325. {-@ measure normalLeaf @-}
326. normalLeaf :: RBSet a -> Bool
327. normalLeaf E = True
328. normalLeaf \_ = False
330. {-@ inline blackRoot @-}
331. blackRoot :: RBSet a -> Bool
332. blackRoot t = color’ t == B
334. {-@ inline noSpecialColor @-}
335. {-@ noSpecialColor :: x:RBSet a -> {v:Bool | (v => noSpecialChild x)
336. && (v => not (isBB’ x)) }
337. @-}
338. noSpecialColor :: RBSet a -> Bool
339. noSpecialColor t = (color’ t /= BB)
340. && (color’ t /= NB)
341. && noSpecialChild t
343. {-@ measure noSpecialChild @-}
344. noSpecialChild :: RBSet a -> Bool
345. noSpecialChild (T \_ l \_ r) = noSpecialColor l && noSpecialColor r
346. noSpecialChild \_ = True
348. {-@ measure redChildIsBlack @-}
349. {-@ redChildIsBlack :: x:RBSet a -> {v:Bool | v => redChildIsBlackNT x} @-}
350. redChildIsBlack :: RBSet a -> Bool
351. redChildIsBlack E = True
352. redChildIsBlack EE = True
353. redChildIsBlack (T R a x b) = color’ a == B &&
354. color’ b == B &&
355. redChildIsBlack a && redChildIsBlack b
356. redChildIsBlack (T \_ a x b) = redChildIsBlack a && redChildIsBlack b
358. {-@ measure redChildIsBlackNT @-}
359. redChildIsBlackNT :: RBSet a -> Bool
360. redChildIsBlackNT (T \_ l \_ r) = redChildIsBlack l && redChildIsBlack r
361. redChildIsBlackNT \_ = True
363. {-@ measure colorValue @-}
364. colorValue :: Color -> Int
365. colorValue NB = -1
366. colorValue R = 0
367. colorValue B = 1
368. colorValue BB = 2
370. {-@ measure blackHeightL @-}
371. {-@ blackHeightL :: {x:RBSet a | noSpecialColor x} -> { i : Int | i >= 1} @-}
372. blackHeightL :: RBSet a -> Int
373. blackHeightL (T c l \_ \_) = blackHeightL l + colorValue c
374. blackHeightL t = colorValue $ color’ t
376. {-@ measure validBlackHeight @-}
377. {-@ validBlackHeight :: {x:RBSet a | noSpecialChild x} -> Bool @-}
378. validBlackHeight :: RBSet a -> Bool
379. validBlackHeight E = True
380. validBlackHeight EE = True
381. validBlackHeight (T \_ l \_ r) = validBlackHeight l && validBlackHeight r
382. && blackHeightL l == blackHeightL r
384. {-@ inline prop\_CT @-}
385. {-@ prop\_CT :: {x:RBSet a | noSpecialColor x}
386. -> {v:Bool | v => prop\_IM x} @-}
387. {-@ invariant {t:RBSet a | prop\_CT t => prop\_IM t} @-}
388. prop\_CT :: (Ord a) => RBSet a -> Bool
389. prop\_CT t = noSpecialColor t
390. && redChildIsBlack t
391. && validBlackHeight t
392. -- && prop\_BST’ t
393. {-@ type CT a = {v:RBSet a | prop\_CT v} @-}
395. {-@ inline prop\_RBSet @-}
396. {-@ prop\_RBSet :: {x:RBSet a | noSpecialColor x}
397. -> {v:Bool | v => prop\_CT x} @-}
398. prop\_RBSet :: (Ord a) => RBSet a -> Bool
399. prop\_RBSet t = prop\_CT t
400. && blackRoot t
401. {-@ type RT a = {v:RBSet a | prop\_RBSet v} @-}
403. {-@ inline prop\_IM @-}
404. {-@ prop\_IM :: {x:RBSet a | noSpecialChild x} -> Bool @-}
405. prop\_IM t = noSpecialChild t
406. && redChildIsBlackNT t
407. && validBlackHeight t
408. -- && prop\_BST’ t
409. {-@ type IM a = {v:RBSet a | prop\_IM v} @-}
411. {-@ measure tooBlack @-}
412. tooBlack :: Color -> Bool
413. tooBlack BB = True
414. tooBlack \_ = False
416. {-@ measure tooRed @-}
417. tooRed :: Color -> Bool
418. tooRed NB = True
419. tooRed \_ = False
421. {-@ inline canBeBlacker @-}
422. canBeBlacker x = color’ x /= BB
424. {-@ inline isBB’ @-}
425. isBB’ t = color’ t == BB
427. {-@ measure prop\_BST’ @-}
428. prop\_BST’ :: (Ord a) => RBSet a -> Bool
429. prop\_BST’ (T \_ l@(T \_ \_ xl \_) x
430. r@(T \_ \_ xr \_)) = (xl < x) && (x < xr)
431. && prop\_BST’ l
432. && prop\_BST’ r
433. prop\_BST’ (T \_ l@(T \_ \_ xl \_) x r) = (xl < x)
434. && prop\_BST’ l
435. && prop\_BST’ r
436. prop\_BST’ (T \_ l x r@(T \_ \_ xr \_)) = (x < xr)
437. && prop\_BST’ l
438. && prop\_BST’ r
439. prop\_BST’ (T \_ l x r) = prop\_BST’ l && prop\_BST’ r --Why do I need these
440. prop\_BST’ \_ = True