Red Black Tree (RBT)

A Recitation of CLRS3 Chapter 13

Wei Peng

IUPUI

07 October 2013

quick review

red-black tree

red-black tree (RBT)

=

binary-search tree (BST)

+

red-black properties (RBP)

red-black properties

- node either red or black
- **root is black**
- every leaf is black
- **red node has black children**
- black height the same

logarithmic height

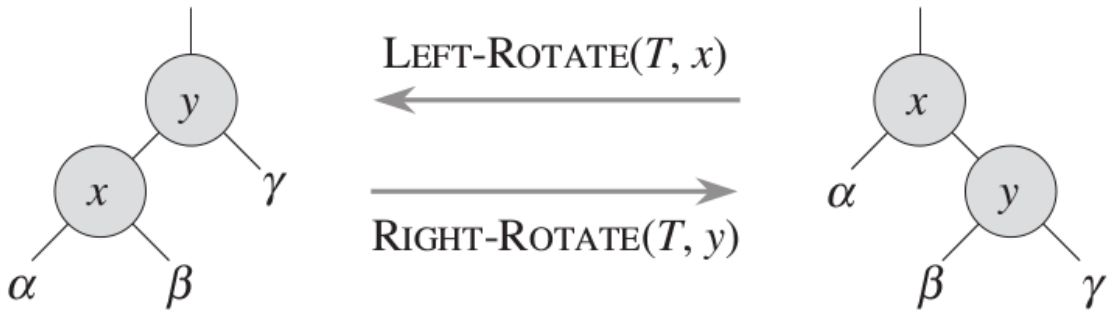
why?

- **well-defined black height** forces a shape of a tree rather than a chain
- **abundance of blacks** force the tree height is as much as twice of black height

this gives good worst-case complexity

the key is to **maintain RBP**

rotation



CLRS3 page 313, Figure 13.2

key: **fix pointers**

rotation

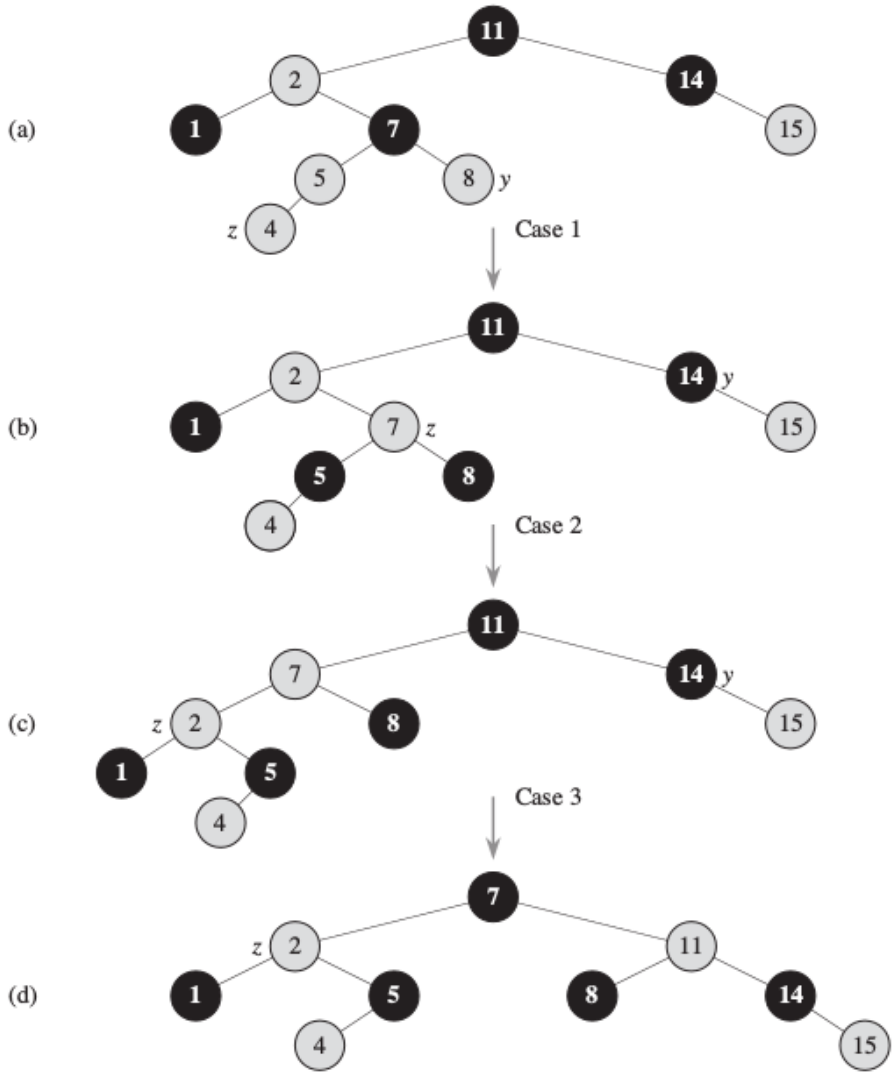
```
LEFT-ROTATE(T, x)
 1  y = x.right              // set y
 2  x.right = y.left         // turn y's left subtree into x's right subtree
 3  if y.left ≠ T.nil
 4       y.left.p = x
 5  y.p = x.p                // link x's parent to y
 6  if x.p == T.nil
 7       T.root = y
 8  elseif x == x.p.left
 9       x.p.left = y
10  else x.p.right = y
11  y.left = x               // put x on y's left
12  x.p = y
```

CLRS3 page 313

node insertion

- first a BST insertion
- mark the new node **red**
- only "red-parent-black-children" and "black root" may be violated
- **fix them!**
- remember to mark the root black

node insertion



CLRS3 page 317, Figure 13.4

key: **fix up by cases**

node insertion

```
RB-INSERT(T, z)
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4        y = x
 5        if z.key < x.key
 6              x = x.left
 7        else x = x.right
 8  z.p = y
 9  if y == T.nil
10        T.root = z
11  elseif z.key < y.key
12        y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

CLRS3 page 315

node insertion

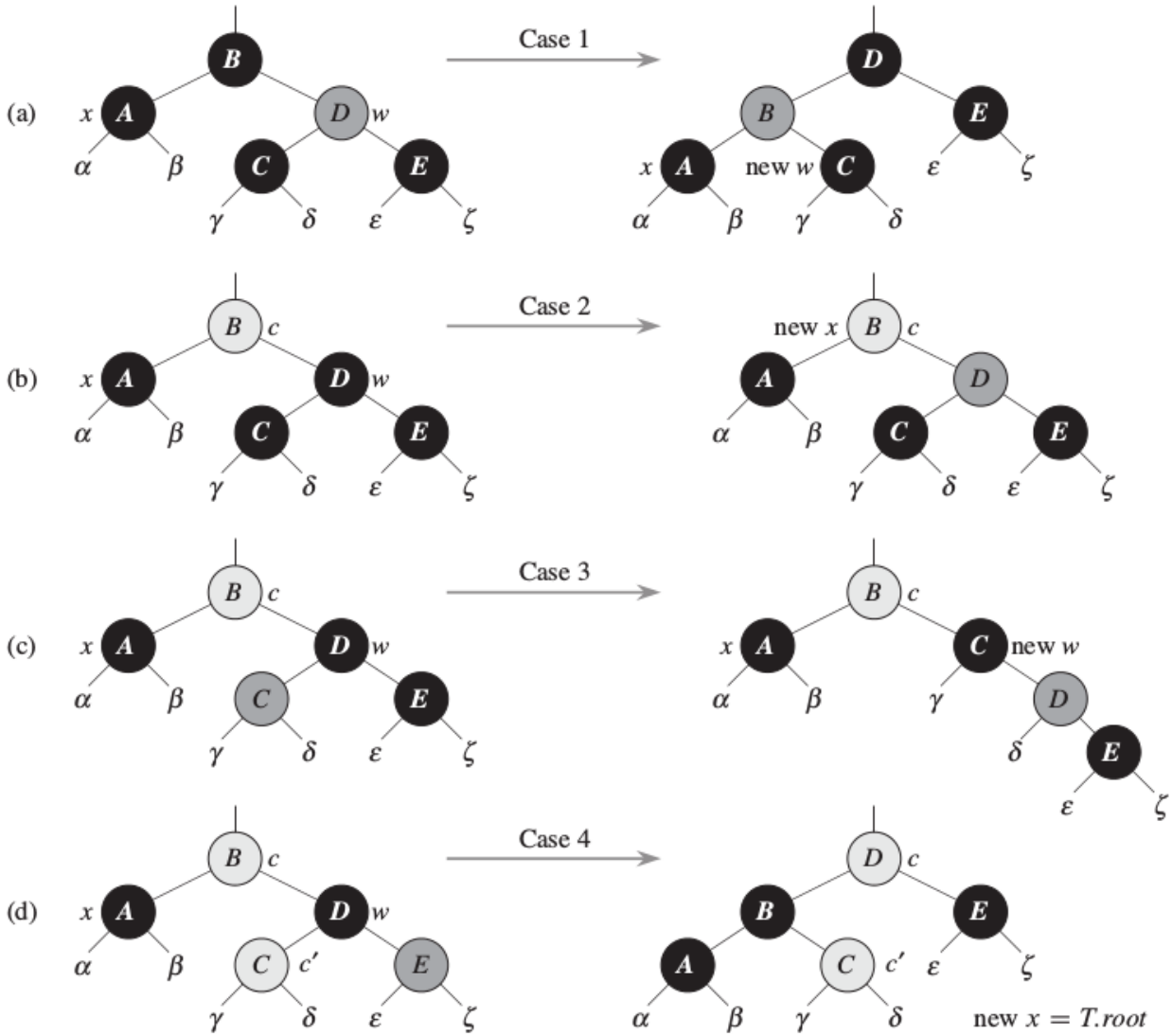
```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
 5              z.p.color = BLACK                    // case 1
 6              y.color = BLACK                      // case 1
 7              z.p.p.color = RED                    // case 1
 8              z = z.p.p                            // case 1
 9          else if z == z.p.right
10                  z = z.p                          // case 2
11                  LEFT-ROTATE(T, z)                // case 2
12              z.p.color = BLACK                    // case 3
13              z.p.p.color = RED                    // case 3
14              RIGHT-ROTATE(T, z.p.p)               // case 3
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```

CLRS3 page 316

node deletion

- a BST deletion of node z, recall the cases:
  - 2 "easy": z has less than 2 children, y is z
  - 1 "intermediate": successor y is z's immediate right child
  - 1 "difficult": successor y is *not* z's immediate right child
- keep track of y and x (the node that moves into y's original place)
  - beware if y is z's immediate right child
- for "non-easy," color y by z's color
- convince yourself why there is no violation if y was red
- possible violations if y was black
  - for "easy," y/z was root, and a red child has replaced it
  - if x and new $x.p$ (y for "intermediate" and old $y.p$ for "difficult") are both red
  - for "difficult," move y might disturb balanced black height
- fix violations

node deletion



CLRS3 page 329, Figure 13.7

key: **fix up by cases**

node deletion

RB-TRANSPLANT(T, u, v)

```
1  if u.p == T.nil
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```

CLRS3 page 323

node deletion

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

CLRS3 page 324

node deletion

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 4          if w.color == RED
 5              w.color = BLACK                          // case 1
 6              x.p.color = RED                          // case 1
 7              LEFT-ROTATE(T, x.p)                      // case 1
 8              w = x.p.right                            // case 1
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED                            // case 2
11              x = x.p                                  // case 2
12          else if w.right.color == BLACK
13                  w.left.color = BLACK                 // case 3
14                  w.color = RED                        // case 3
15                  RIGHT-ROTATE(T, w)                   // case 3
16                  w = x.p.right                        // case 3
17              w.color = x.p.color                      // case 4
18              x.p.color = BLACK                        // case 4
19              w.right.color = BLACK                    // case 4
20              LEFT-ROTATE(T, x.p)                      // case 4
21              x = T.root                               // case 4
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```

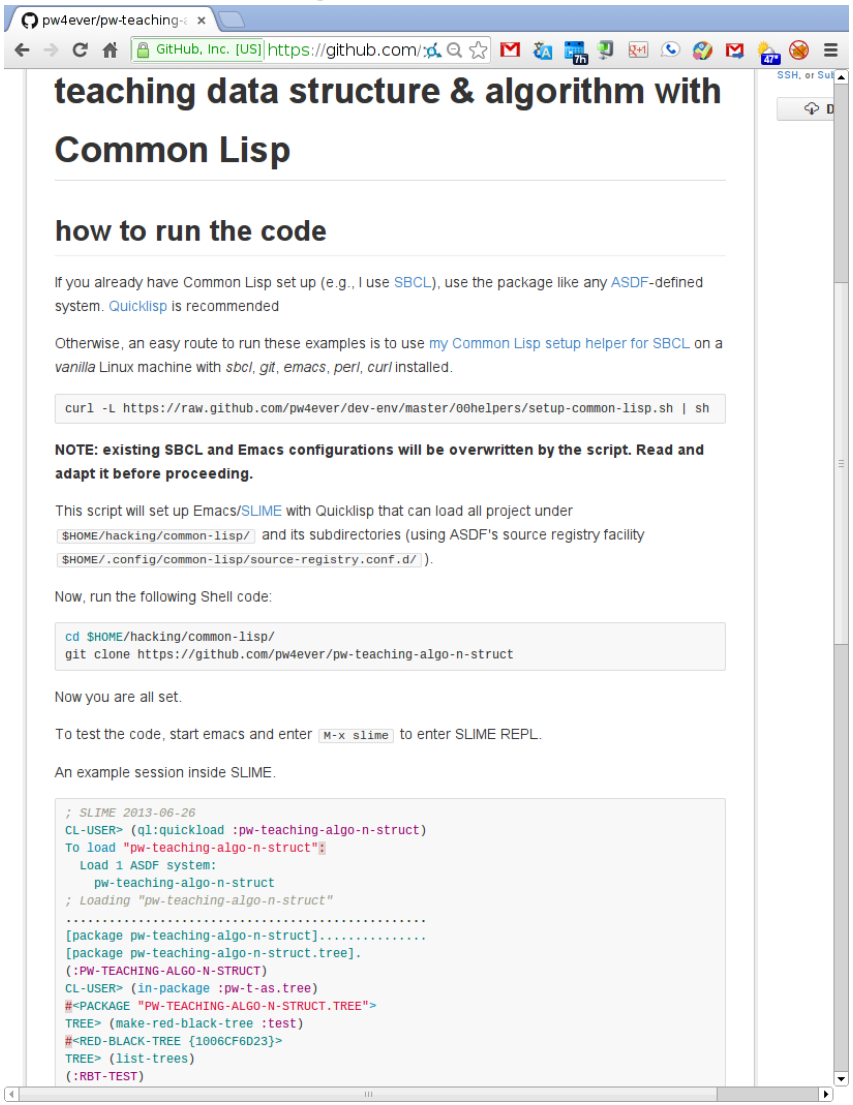
CLRS3 page 326

so much for the review

let us do some exercises

we can use some help in the exercises

https://github.com/pw4ever/pw-teaching-algo-n-struct



pw4ever/pw-teaching-algo-n-struct

teaching data structure & algorithm with Common Lisp

how to run the code

If you already have Common Lisp set up (e.g., I use SBCL), use the package like any ASDF-defined system. Quicklisp is recommended

Otherwise, an easy route to run these examples is to use my Common Lisp setup helper for SBCL on a *vanilla* Linux machine with *sbcl*, *git*, *emacs*, *perl*, *curl* installed.

```
curl -L https://raw.github.com/pw4ever/dev-env/master/00helpers/setup-common-lisp.sh | sh
```

NOTE: existing SBCL and Emacs configurations will be overwritten by the script. Read and adapt it before proceeding.

This script will set up Emacs/SLIME with Quicklisp that can load all project under `$HOME/hacking/common-lisp/` and its subdirectories (using ASDF's source registry facility `$HOME/.config/common-lisp/source-registry.conf.d/`).

Now, run the following Shell code:

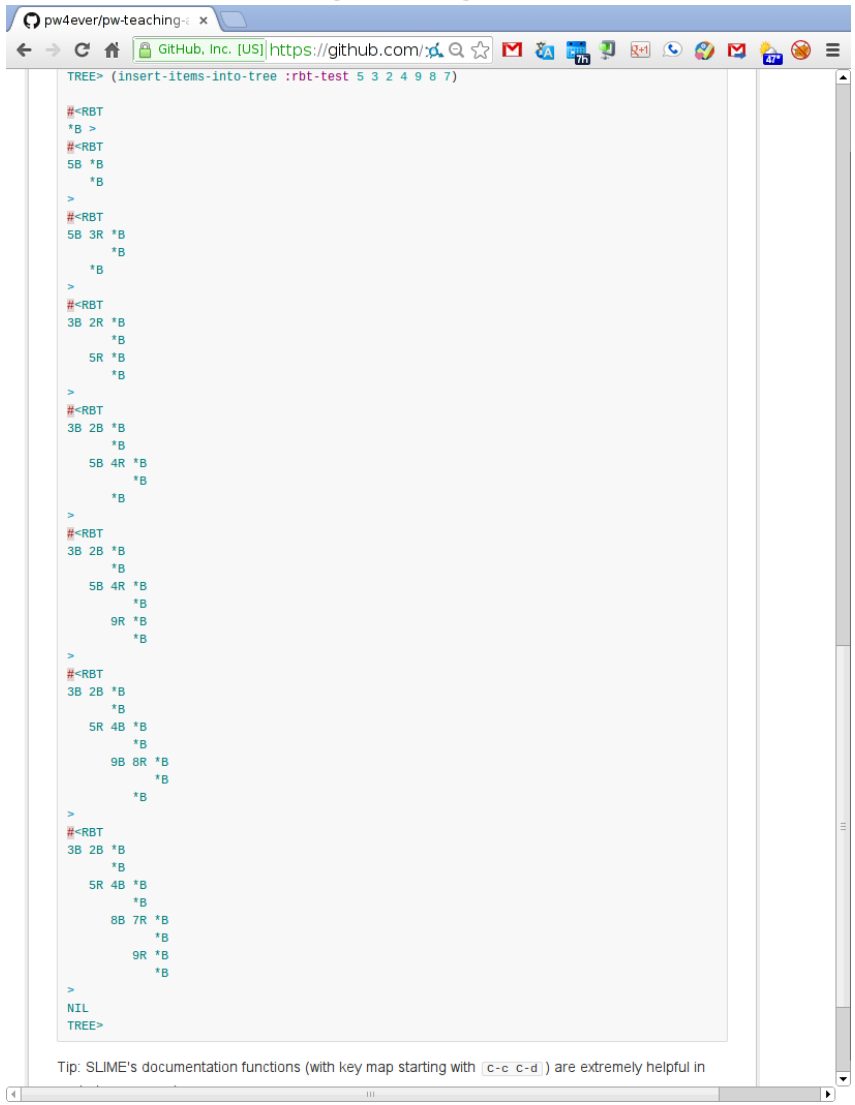
```
cd $HOME/hacking/common-lisp/
git clone https://github.com/pw4ever/pw-teaching-algo-n-struct
```

Now you are all set.

To test the code, start emacs and enter `M-x slime` to enter SLIME REPL.

An example session inside SLIME.

```
; SLIME 2013-06-26
CL-USER> (ql:quickload :pw-teaching-algo-n-struct)
To load "pw-teaching-algo-n-struct":
  Load 1 ASDF system:
    pw-teaching-algo-n-struct
; Loading "pw-teaching-algo-n-struct"
..................................................
[package pw-teaching-algo-n-struct].............
[package pw-teaching-algo-n-struct.tree].
(:PW-TEACHING-ALGO-N-STRUCT)
CL-USER> (in-package :pw-t-as.tree)
#<PACKAGE "PW-TEACHING-ALGO-N-STRUCT.TREE">
TREE> (make-red-black-tree :test)
#<RED-BLACK-TREE {1006CF6D23}>
TREE> (list-trees)
(:RBT-TEST)
```



pw4ever/pw-teaching-algo-n-struct

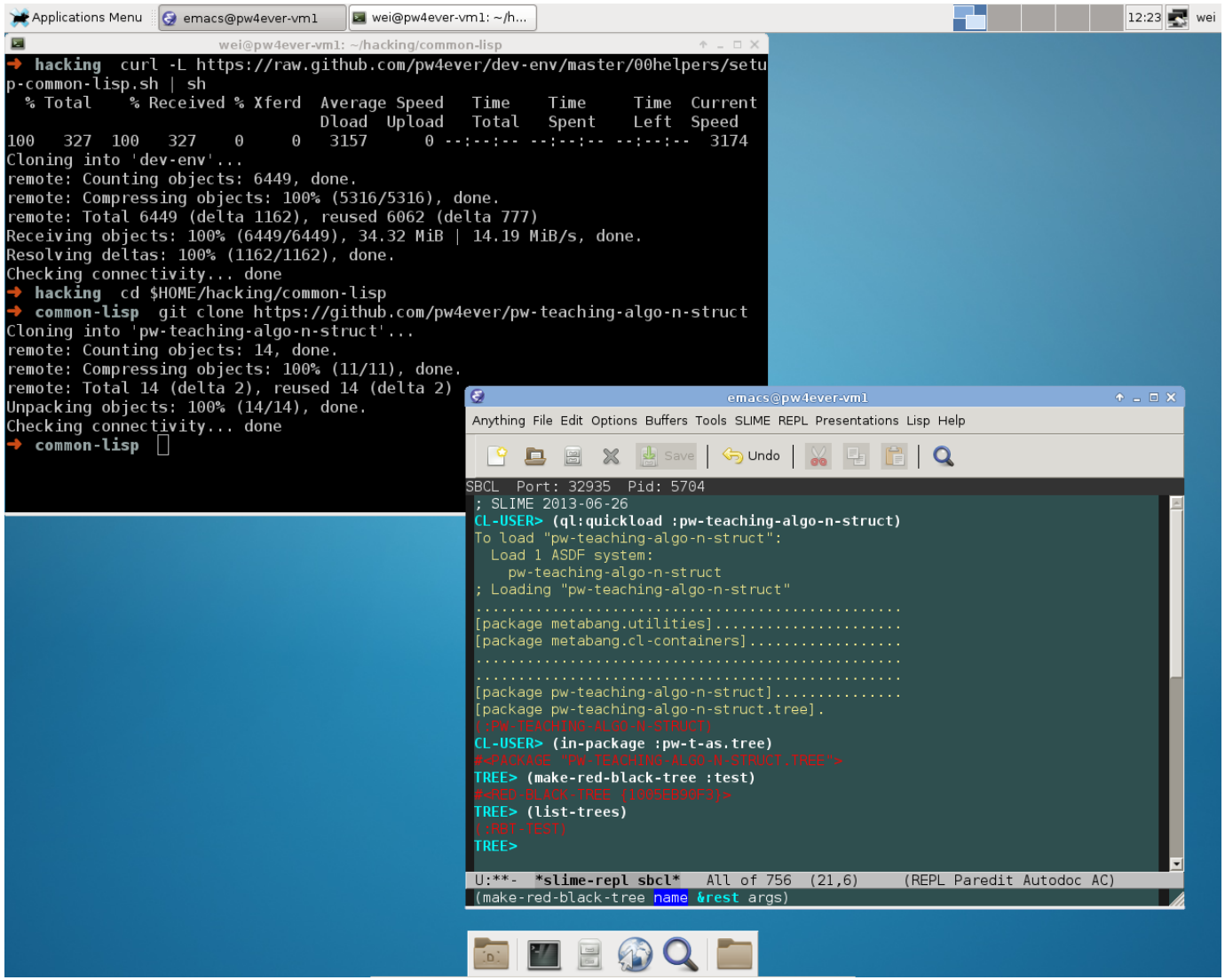
```
TREE> (insert-items-into-tree :rbt-test 5 3 2 4 9 8 7)

#<RBT
*B >
#<RBT
5B *B
   *B
>
#<RBT
5B 3R *B
     *B
   *B
>
#<RBT
3B 2R *B
     *B
   5R *B
     *B
>
#<RBT
3B 2B *B
     *B
   5B 4R *B
        *B
     *B
>
#<RBT
3B 2B *B
     *B
   5B 4R *B
        *B
     9R *B
        *B
>
#<RBT
3B 2B *B
     *B
   5R 4B *B
        *B
     9B 8R *B
           *B
        *B
>
#<RBT
3B 2B *B
     *B
   5R 4B *B
        *B
     8B 7R *B
           *B
        9R *B
           *B
>
NIL
TREE>
```

Tip: SLIME's documentation functions (with key map starting with `C-c C-d`) are extremely helpful in

we can use some help in the exercises

the setup is quite simple on a Linux VM



get your paper and pen

- start with an empty RBT T
- shuffle 1 to 18 as an array $A[0..17]$
- for i from 0 to 2
  - insert $A[6 \times i..6 \times i + 5]$ into T
  - let us pick a node on T and delete it

trace the whole process, see what is left

create T and pick A

```
; SLIME 2013-06-26
CL-USER> (ql:quickload :pw-teaching-algo-n-struct)
To load "pw-teaching-algo-n-struct":
  Load 1 ASDF system:
    pw-teaching-algo-n-struct
; Loading "pw-teaching-algo-n-struct"
................................................
[package pw-teaching-algo-n-struct].............
[package pw-teaching-algo-n-struct.tree].
(:PW-TEACHING-ALGO-N-STRUCT)
CL-USER> (in-package :pw-t-as.tree)
#<PACKAGE "PW-TEACHING-ALGO-N-STRUCT.TREE">
TREE> (make-red-black-tree :test)
#<RED-BLACK-TREE {1006C778A3}>
TREE> (defparameter *seq* (shuffle (loop for i from 1 to 18 collect i)))
*SEQ*
TREE> *seq*
(15 14 18 9 6 16 3 11 2 4 13 12 17 1 8 10 7 5)
TREE>
```

1st round

insert 15, 14, 18, 9, 6, and 16

```
TREE> *seq*
(15 14 18 9 6 16 3 11 2 4 13 12 17 1 8 10 7 5)
TREE> (apply #'insert-items-into-tree :rbt-test (subseq *seq* 0 6))

#<RBT
*B >
#<RBT
15B   *B
      *B
>
#<RBT
15B   14R   *B
            *B
      *B
>
#<RBT
15B   14R   *B
            *B
      18R   *B
            *B
>
#<RBT
15B   14B   9R *B
               *B
            *B
      18B   *B
            *B
>
#<RBT
15B   9B 6R *B
            *B
         14R   *B
               *B
      18B   *B
            *B
>
#<RBT
15B   9B 6R *B
            *B
         14R   *B
               *B
      18B   16R   *B
                  *B
            *B
>
NIL
TREE>
```

1st round

delete 18

```
TREE> (delete-items-from-tree :rbt-test 18)

#<RBT
15B   9B 6R *B
          *B
        14R   *B
              *B
      18B   16R   *B
                  *B
            *B
>
#<RBT
15B   9B 6R *B
          *B
        14R   *B
              *B
      16B   *B
            *B
>
NIL
TREE>
```

2nd round

insert 3, 11, 2, 4, 13, and 12

```
TREE> *seq*
(15 14 18 9 6 16 3 11 2 4 13 12 17 1 8 10 7 5)
TREE> (apply #'insert-items-into-tree :rbt-test (subseq *seq* 6 12))

#<RBT
15B   9B 6R *B
            *B
         14R   *B
               *B
      16B   *B
            *B
>
#<RBT
15B   9R 6B 3R *B
               *B
            *B
         14B   *B
               *B
      16B   *B
            *B
>
#<RBT
15B   9R 6B 3R *B
               *B
            *B
         14B   11R   *B
                     *B
               *B
      16B   *B
            *B
>
#<RBT
15B   9R 3B 2R *B
               *B
            6R *B
               *B
         14B   11R   *B
                     *B
               *B
      16B   *B
            *B
>
```

2nd round

insert 3, 11, 2, 4, 13, and 12

```
#<RBT
9B 3R 2B *B
         *B
      6B 4R *B
            *B
         *B
   15R   14B   11R   *B
                     *B
               *B
         16B   *B
               *B
>
#<RBT
9B 3R 2B *B
         *B
      6B 4R *B
            *B
         *B
   15R   13B   11R   *B
                     *B
               14R   *B
                     *B
         16B   *B
               *B
>
#<RBT
9B 3B 2B *B
         *B
      6B 4R *B
            *B
         *B
   15B   13R   11B   *B
                     12R   *B
                           *B
               14B   *B
                     *B
         16B   *B
               *B
>
```

2nd round

delete 15

```
TREE> (delete-items-from-tree :rbt-test 15)

#<RBT
9B 3B 2B *B
         *B
      6B 4R *B
            *B
         *B
   15B   13R   11B   *B
                     12R   *B
                           *B
               14B   *B
                     *B
         16B   *B
               *B
>
#<RBT
9B 3B 2B *B
         *B
      6B 4R *B
            *B
         *B
   13B   11B   *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               *B
>
NIL
```

3rd round

insert 17, 1, 8, 10, 7, and 5

```
TREE> (subseq *seq* 12 18)
(17 1 8 10 7 5)
TREE> (apply #'insert-items-into-tree
             :rbt-test
             (subseq *seq* 12 18))

#<RBT
9B 3B 2B *B
         *B
      6B 4R *B
            *B
         *B
   13B   11B   *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               *B
>
#<RBT
9B 3B 2B *B
         *B
      6B 4R *B
            *B
         *B
   13B   11B   *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               17R   *B
                     *B
>
#<RBT
9B 3B 2B 1R *B
            *B
         *B
      6B 4R *B
            *B
         *B
   13B   11B   *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               17R   *B
                     *B
>
```

3rd round

insert 17, 1, 8, 10, 7, and 5

```
#<RBT
9B 3B 2B 1R *B
            *B
         *B
      6B 4R *B
            *B
         8R *B
            *B
   13B   11B   *B
                  12R   *B
                        *B
         16B      14R   *B
                        *B
                  17R   *B
                        *B
>
#<RBT
9B 3B 2B 1R *B
            *B
         *B
      6B 4R *B
            *B
         8R *B
            *B
   13B   11B   10R   *B
                     *B
                  12R   *B
                        *B
         16B      14R   *B
                        *B
                  17R   *B
                        *B
>
```

3rd round

insert 17, 1, 8, 10, 7, and 5

```
#<RBT
9B 3B 2B 1R *B
           *B
         *B
      6R 4B *B
           *B
         8B 7R *B
              *B
           *B
   13B   11B   10R   *B
                     *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               17R   *B
                     *B
>
#<RBT
9B 3B 2B 1R *B
           *B
         *B
      6R 4B *B
           5R *B
              *B
         8B 7R *B
              *B
           *B
   13B   11B   10R   *B
                     *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               17R   *B
                     *B
>
```

3rd round

delete 9

```
TREE> (delete-items-from-tree :rbt-test 9)

#<RBT
9B 3B 2B 1R *B
            *B
         *B
      6R 4B *B
            5R *B
               *B
         8B 7R *B
               *B
            *B
   13B   11B   10R   *B
                     *B
               12R   *B
                     *B
         16B   14R   *B
                     *B
               17R   *B
                     *B
>
#<RBT
10B   3B 2B 1R *B
               *B
            *B
         6R 4B *B
               5R *B
                  *B
            8B 7R *B
                  *B
               *B
      13B   11B   *B
                  12R   *B
                        *B
            16B   14R   *B
                        *B
                  17R   *B
                        *B
>
```

Q & A