```
(require 2htdp/image)
(require spd/tags)

;; =================
;; Constants:

(define TEXT-SIZE  14)
(define TEXT-COLOR "BLACK")

(define KEY-VAL-SEPARATOR ":")

(define MTTREE (rectangle 20 1 "solid" "white"))



;; =================
;; Data definitions:

(@htdd BST)
(define-struct node (key val l r))
;; BST (Binary Search Tree) is one of:
;;  - false
;;  - (make-node Integer String BST BST)
;; interp. false means no BST, or empty BST
;;         key is the node key
;;         val is the node val
;;         l and r are left and right subtrees
;; INVARIANT: for a given node:
;;     key is > all keys in its l(eft)  child
;;     key is < all keys in its r(ight) child
;;     the same key never appears twice in the tree
(define BST0 false)
(define BST1 (make-node 1 "abc" false false))
(define BST7 (make-node 7 "ruf" false false))
(define BST4 (make-node 4 "dcj" false (make-node 7 "ruf" false false)))
(define BST3 (make-node 3 "ilk" BST1 BST4))
(define BST42
  (make-node 42 "ily"
             (make-node 27 "wit" (make-node 14 "olp" false false) false)
             (make-node 50 "dug" false false)))
(define BST10
  (make-node 10 "why" BST3 BST42))
(define BST100
  (make-node 100 "large" BST10 false))

(define (fn-for-bst t)
  (cond [(false? t) (...)]
        [else
         (... (node-key t)    ;Integer
              (node-val t)    ;String
              (fn-for-bst (node-l t))
              (fn-for-bst (node-r t)))]))



;; =================
;; Functions:

;; PROBLEM FROM: bsts-p6
;;
;; Design a function that consumes a bst and produces a SIMPLE
;; rendering of that bst including lines between nodes and their
;; subnodes.
;; Here is a link to an image of BST10 for reference:
;;
;; https://cs110.students.cs.ubc.ca/bank/bst10.png
;;
```

A

B

```
;; Here is a link to a sketch of a helper you can use to draw the lines:
;;
;; https://cs110.students.cs.ubc.ca/bank/bst-helper.png
;;
;; where lw means width of left subtree image and
;;       rw means width of right subtree image

(@htdf render-bst)
(@signature BST -> Image)
;; produce SIMPLE rendering of bst
;; ASSUME BST is relatively well balanced
(check-expect (render-bst false) MTTREE)
(check-expect (render-bst BST1)
              (above (render-key-val 1 "abc")
                     (lines (image-width (render-bst false))
                            (image-width (render-bst false)))
                     (beside (render-bst false)
                             (render-bst false))))

(@template-origin BST)

(define (render-bst bst)
  (cond [(false? bst) MTTREE]
        [else
         (local [(define li (render-bst (node-l bst)))
                 (define ri (render-bst (node-r bst)))
                 (define lw (image-width li))
                 (define rw (image-width ri))]
           (above (render-key-val (node-key bst) (node-val bst))
                  (lines lw rw)
                  (beside li ri)))]))


(@htdf render-key-val)
(@signature Integer String -> Image)
;; render key and value to form the body of a node
(check-expect (render-key-val 99 "foo")
              (text (string-append "99" KEY-VAL-SEPARATOR "foo")
                    TEXT-SIZE
                    TEXT-COLOR))

(@template-origin Integer)

(define (render-key-val k v)
  (text (string-append (number->string k)
                       KEY-VAL-SEPARATOR
                       v)
        TEXT-SIZE
        TEXT-COLOR))


(@htdf lines)
(@signature Natural Natural -> Image)
;; produce lines to l/r subtrees based on width of those subtrees
(check-expect (lines 60 130)
              (add-line (add-line (rectangle (+ 60 130)
                                             (/ 190 4)
                                             "solid"
                                             "white")
                                  (/ (+ 60 130) 2) 0
                                  (/ 60 2)         (/ 190 4)
                                  "black")
                        (/ (+ 60 130) 2) 0
                        (+ 60 (/ 130 2)) (/ 190 4)
                        "black"))

(@template-origin Natural)
```

```
(define (lines lw rw)
  (local [(define tw (+ lw rw))         ;total width
          (define th (/ (+ lw rw) 4))   ;total height
          (define cx (/ tw 2))          ;center x
          (define lx (/ lw 2))          ;left line, lower x
          (define rx (+ lw (/ rw 2)))] ;right line, lower x

    (add-line (add-line (rectangle tw th "solid" "white")
                        cx  0 lx th "black")
              cx 0 rx th "black")))
```