```
static idx_t
_rb_rotate_right(RBTree *tree, idx_t h) {
   g_rotation_count++;
   idx_t pivot = tree->nodes[h].left;
   tree->nodes[h].left = tree->nodes[pivot].right;
   tree->nodes[pivot].right = h;
   tree->nodes[pivot].color = tree->nodes[h].color;
   tree->nodes[h].color = RED;
   return pivot;
}

/* Inverter cores */
static void
_rb_flip_colors(RBTree *tree, idx_t h) {
   tree->nodes[h].color = !tree->nodes[h].color;
   idx_t left = tree->nodes[h].left;
   idx_t right = tree->nodes[h].right;
   if (left != IDX_INVALID)
      tree->nodes[left].color = !tree->nodes[left].color;
   if (right != IDX_INVALID)
      tree->nodes[right].color = !tree->nodes[right].color;
}

/* Resolve comflictos */
static idx_t
_rb_fix_up(RBTree *tree, idx_t h) {
   /* Caso 1: direita vermelha e esquerda preta -> rotação à esquerda */
   if (_rb_is_red(tree, tree->nodes[h].right) && !_rb_is_red(tree, tree->nodes[h].left))
      h = _rb_rotate_left(tree, h);
   /* Caso 2: filho esquerdo vermelho e neto esquerdo vermelho -> rotação à direita */
   if (_rb_is_red(tree, tree->nodes[h].left) && _rb_is_red(tree, tree->nodes[tree->nodes[h].left].left))
      h = _rb_rotate_right(tree, h);
   /* Caso 3: ambos os filhos são vermelhos */
   if (_rb_is_red(tree, tree->nodes[h].left) && _rb_is_red(tree, tree->nodes[h].right))
      _rb_flip_colors(tree, h);
   return h;
}

/* Inserção recursiva: Devolve o novo indice da raiz se inserir */
static idx_t
_rb_insert_recursive(RBTree *tree, idx_t h, key_t key) {

   /* Inserir após encontrar nova folha
    * (chamada anterior para filho que não existe) */
   if (h == IDX_INVALID) {
      idx_t new_index = tree->elements;
      tree->nodes[new_index].key = key;
      tree->nodes[new_index].left = IDX_INVALID;
      tree->nodes[new_index].right = IDX_INVALID;
      tree->nodes[new_index].color = RED;  // sempre vermelho
      tree->elements++;
      return new_index;
   }

   /* Recursão equivalente a binary search tree */
   if (key < tree->nodes[h].key) {
      tree->nodes[h].left = _rb_insert_recursive(tree, tree->nodes[h].left, key);
   } else if (key > tree->nodes[h].key) {
      tree->nodes[h].right = _rb_insert_recursive(tree, tree->nodes[h].right, key);
   }
```