```
   /* É necessário corrigir erros causados pela inserção */
   h = _rb_fix_up(tree, h);
   return h;
}

/* Inserir nó */
void
tree_rb_insert(RBTree *tree, key_t key) {

   /* Aumentar capacidade  se for necessário */
   if (tree->capacity == tree->elements)
      tree_rb_resize(tree);

   tree->tree_root = _rb_insert_recursive(tree, tree->tree_root, key);
   tree->nodes[tree->tree_root].color = BLACK;
}

/* Pesquisa */
int
tree_rb_search(RBTree *tree, int key) {
   RBNode *nodes = tree->nodes;
   idx_t current = tree->tree_root;

   /* binary search tree search */
   while (current != IDX_INVALID) {
      printf("current = %d\n", current);
      if (key < nodes[current].key)
         current = nodes[current].left;
      else if (key > nodes[current].key)
         current = nodes[current].right;
      else
         return current;
   }

   return -1;
}


void
rb_test_and_log(key_t* arr, FILE *fptr) {

   RBTree vp;
   clock_t start = 0, end = 0;
   clock_t total = 0;

   /* Reset global rotation counter */
   g_rotation_count = 0;

   for (int i = 0; i < g_average; i++) { start = clock();
      vp = tree_rb_create(g_treesize);
      for (idx_t idx = 0; idx < g_treesize; idx++)
         tree_rb_insert(&vp, arr[idx]);
      end = clock();

      total += (end-start);
      tree_rb_destroy(&vp);
   }

   double total_time = ((double) total*1000) / CLOCKS_PER_SEC;
   fprintf(fptr, "RB Tree = %0.4lfms\t(%d rotations)\n", total_time/g_average,
g_rotation_count/g_average);
```