```
      else
         current_index = avl->nodes[current_index].right;
   }
   return NULL;  // Key not found.
}


void
tree_avl_insert_arr(AVLTree *avl, key_t* arr, size_t size) {
   assert(avl && arr);
   for (key_t* endptr = arr+size; arr != endptr && arr != NULL; arr++) {
      tree_avl_insert(avl, *arr);
   }
}


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

   AVLTree avl;
   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();
      avl = tree_avl_create(10);
      tree_avl_insert_arr(&avl, arr, g_treesize);
      end = clock();

      total += (end-start);
      tree_avl_destroy(&avl);
   }

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

/* Red Black Tree Implementation */
/* Criar arvore */
RBTree
tree_rb_create(uint32_t initial_capacity) {
   RBTree tree;
   tree.nodes = malloc(initial_capacity * sizeof(RBNode));
   assert(tree.nodes != NULL);

   tree.elements = 0;
   tree.capacity = initial_capacity;

   // a raiz da arvore é invalida inicialmente
   // para que seja pintada correctamente de preto (caso especial)
   // e inserida imediatamente
   tree.tree_root = IDX_INVALID;

   RBNode *endptr = tree.nodes+tree.capacity;
   for (RBNode *ptr = tree.nodes; ptr != endptr; ptr++) {
      ptr->key = 0;
      ptr->color = -1;
```