```
      ptr->left = IDX_INVALID;
      ptr->right = IDX_INVALID;
   }
   return tree;
}

/* Destruir árvore */
void
tree_rb_destroy(RBTree *rb) {
   free(rb->nodes);
}

/* Aumentar capacidade */
void
tree_rb_resize(RBTree *tree) {
   assert(tree != NULL);
   uint32_t new_capacity = tree->capacity * RESIZE_FACTOR;

   RBNode *new_nodes = realloc(tree->nodes, new_capacity * sizeof(RBNode));
   if (new_nodes == NULL) {
      free(tree->nodes);
      perror("Failed to allocate more nodes.");
      exit(EXIT_FAILURE);
   }

   // inicializar novos nós
   RBNode *endptr = new_nodes + new_capacity;
   for (RBNode *ptr = new_nodes + tree->capacity; ptr != endptr; ptr++) {
      ptr->key = 0;
      ptr->color = -1;
      ptr->left = IDX_INVALID;
      ptr->right = IDX_INVALID;
   }

   tree->nodes = new_nodes;
   tree->capacity = new_capacity;
}


/* 1 (verdadeiro) se o nó for vermelho */
static int
_rb_is_red(RBTree *tree, idx_t i) {
   // a raiz da arvore é invalida inicialmente
   // isto significa que vai ser pintada correctamente de preto
   if (i == IDX_INVALID) return 0;
   return (tree->nodes[i].color == RED);
}

/* rotação à esquerda */
static idx_t
_rb_rotate_left(RBTree *tree, idx_t h) {
   g_rotation_count++;
   idx_t pivot = tree->nodes[h].right;
   tree->nodes[h].right = tree->nodes[pivot].left;
   tree->nodes[pivot].left = h;
   tree->nodes[pivot].color = tree->nodes[h].color;
   tree->nodes[h].color = RED;
   return pivot;
}

/* rotação à direita */
```