```
_treap_rotate_right(Treap *treap, idx_t no_idx) {
   g_rotation_count++;

   TreapNode *nodes = treap->nodes;
   idx_t pivot_idx = nodes[no_idx].left;

   /* à esquerda agora fica a subtree do pivot */
   nodes[no_idx].left = nodes[pivot_idx].right;
   /* à direita do pivot fica o nó atual */
   nodes[pivot_idx].right = no_idx;

   /* o pivot não muda de sitio mas pode passar a ser a nova raiz */
   return pivot_idx;
}

static idx_t
_treap_rotate_left(Treap *treap, idx_t no_idx) {
   g_rotation_count++;

   TreapNode *nodes = treap->nodes;
   idx_t pivot_idx = nodes[no_idx].right;

   /* subtree do pivot */
   nodes[no_idx].right = nodes[pivot_idx].left;
   /* o pivot agora leva ao nó */
   nodes[pivot_idx].left = no_idx;

   /* o pivot pode passar a ser a nova raiz */
   return pivot_idx;
}

static idx_t
_treap_insert_recursive(Treap *treap, idx_t idx, key_t key) {
   TreapNode *nodes = treap->nodes;

   /* Criamos um novo nó quando quando o idx é inválido,
    * ou seja, quando o BST tenta inserir numa folha
    * depois devolvemos o novo indice à chamada anterior desta função*/
   if (idx == IDX_INVALID) {
      idx_t new_index = treap->elements;
      treap->elements++;
      nodes[new_index] = (TreapNode){
         .key = key,
         .priority = (idx_t)rand_idx(1, IDX_INVALID - 1),
         .left = IDX_INVALID,
         .right = IDX_INVALID
      };
      return new_index;
   }

   /* inserir tipo binary search tree */
   if (key < nodes[idx].key) {
      nodes[idx].left = _treap_insert_recursive(treap, nodes[idx].left, key);

      /* manter max heap */
      if (nodes[nodes[idx].left].priority > nodes[idx].priority) {
         idx = _treap_rotate_right(treap, idx);
      }
   } else if (key > nodes[idx].key) {
      nodes[idx].right = _treap_insert_recursive(treap, nodes[idx].right, key);
```