```
}

void
tree_avl_insert(AVLTree *avl, int key) {

   if (avl->elements == avl->capacity) {
      tree_avl_resize(avl);
   }


   /* For an empty tree, set the new node as root. */
   if (avl->elements == 0) {
      avl->tree_root = _avl_insert_recursive(avl, IDX_INVALID, key);
   } else {
      avl->tree_root = _avl_insert_recursive(avl, avl->tree_root, key);
   }
}
void
tree_avl_in_order(AVLTree *avl) {

   void _traverse(AVLNode *nodes, idx_t i) {
      if (i == IDX_INVALID) return;
      AVLNode no = nodes[i];
      _traverse(nodes, no.left);
      printf("%d ", no.key);
      _traverse(nodes, no.right);
   }

   _traverse(avl->nodes, avl->tree_root);
   puts("");
}

void
tree_avl_in_order_non(AVLTree *avl) {

   AVLNode current_node;
   idx_t current_index = avl->tree_root;

   while (current_index != IDX_INVALID) {
      current_node = avl->nodes[avl->tree_root];

      /* Traverse left sub-tree until leaf */
      if (current_node.left != IDX_INVALID) {
         current_index = current_node.left;
         continue;
      }
      printf("%d ", current_node.key);
   }
}


AVLNode*
tree_avl_search(AVLTree *avl, int key) {
   idx_t current_index = avl->tree_root;
   while (current_index != IDX_INVALID) {
      if (avl->nodes[current_index].key == key)
         return &avl->nodes[current_index];

      else if (key < avl->nodes[current_index].key)
         current_index = avl->nodes[current_index].left;
```