```
      found |= ((ptr_back - 3)->data == key);

      ptr_front += 4;
      ptr_back -= 4;

      if (found) return 0;
   }

   /* elementos restante */
   while (ptr_front <= ptr_back) {
      found |= (ptr_front->data == key) | (ptr_back->data == key);
      ptr_front++;
      ptr_back--;
   }

   return found ? 0 : IDX_INVALID;
}

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

   BinTree btree;
   clock_t start = 0, end = 0;
   clock_t total = 0;

   for (int i = 0; i < g_average; i++) {
      start = clock();
      btree = tree_binary_create(g_treesize);
      tree_binary_insert_arr(&btree, arr, g_treesize);
      end = clock();

      total += (end-start);
      tree_binary_destroy(btree);
   }

   double total_time = ((double) total*1000) / CLOCKS_PER_SEC;
   fprintf(fptr, "Binary Tree = %0.4lfms\t(0 rotations)\n", total_time/g_average);
}

AVLTree
tree_avl_create(idx_t inicial_capacity) {
   assert(inicial_capacity > 0);

   AVLTree avl = {NULL, 0, 0, inicial_capacity};
   avl.nodes = (AVLNode*) malloc( sizeof(AVLNode) * inicial_capacity);

   if (avl.nodes == NULL) {
      perror("Couldn't allocate AVL tree.");
      exit(EXIT_FAILURE);
   }

   for (idx_t i = 0; i < inicial_capacity; i++) {
      avl.nodes[i] = (AVLNode) {IDX_INVALID, IDX_INVALID, 0, 1};
   }

   return avl;
}

void
tree_avl_destroy(AVLTree* avl) {
   assert(avl);
```