```
void
tree_binary_resize(BinTree *btree) {
   uint32_t new_capacity = btree->capacity*RESIZE_FACTOR;
   /* In case of overflow */
   if (new_capacity < btree->capacity) return;

   BinTreeNode* new_root = NULL;
   uint64_t tries = 0;
   while (new_root == NULL && tries < 1000) {
      new_root = (BinTreeNode*) realloc(btree->root, sizeof(BinTreeNode)*new_capacity);
      tries++;
   }

   if (new_root == NULL) {
      puts("Failed to allocate enough memory for tree resize.\n");
      exit(EXIT_FAILURE);
   }

   btree->root = new_root;
   btree->capacity = new_capacity;
}

void
tree_binary_insert(BinTree *btree, key_t key) {

   /* NA OPERAÇÃO DE INSERÇÃO QUANDO UMA CHAVE JÁ EXISTIR, NÃO É CRIADA NOVA CHAVE */
   if (IDX_INVALID != tree_binary_search_key_level(*btree, key)) {
      return;
   };

   if (btree->elements == btree-> capacity) {
      /*puts("capacity exceeded");*/
      /*printf("capacity = %d\n", btree->capacity);*/
      tree_binary_resize(btree);
      /*printf("new capacity = %d\n", btree->capacity);*/
   }

   BinTreeNode* root = btree->root;
   BinTreeNode* node = NULL;
   uint32_t inicial_elements = btree->elements;

   /* Está vazia */
   if (inicial_elements == 0) {
      node = btree->root;
   } else {
      /* Devido às propriedades das àrvores binárias implicitas,
       * o indice corresponde ao número de elementos-1,
       * como é o próximo indice, -1+1 = 0 */
      node = &root[inicial_elements];
      /* O pai do NOVO filho está em (Nº Elementos+1) / 2 arredondado para baixo -1 para o indice
       * O que pode ser simplificado para simplesmente o numero de elements>>1
       * O módulo diz-nos se é para a esquerda ou direita */
      BinTreeNode* parent_node = &root[inicial_elements>>1];
      if (inicial_elements % 2 == 0) {
         parent_node->idx_left = inicial_elements;
      } else {
         parent_node->idx_right = inicial_elements;
      }
   }

   /* Inserir nova chave  */
```