Report

Jason Young 522030990010

Bugs:

1. Line 146 : while (node -> is_leaf) (Runtime Error)
   - while (node -> is_leaf) change to while (!node -> is_leaf). If the node is not a leaf node, the value will be false, and triggers the loop to check if the child nodes of the current node (this) are leaf nodes. If it's not changed, a null node will be accessed and cause segFault.

2. Line 114: split_leaf in node class (Logic Error)
   - Changing the whole structure of the function to maintain the structure of B+ tree, specifically the double linked list of the leaf nodes.
   - Other than that, I added additional logic to manage the neighbor nodes of the newly created node (n_left). When (n_left) is created, it checks to see if there is left neighbor, if so, it sets that neighbor's (right) pointer to point back to (n_left). This is to ensure the double linked list is not broken.
   - The original code doesn't properly maintain (left) and (right) pointers, that could lead to breakdown of the structure and could result in incorrect behavior during insert, get / auto, and remove.
   - By managing the leaf nodes' links, it ensures the tree is balanced.

3. Line 24 : ~node_t() (use-after-free Error)
   - Changing the destructor of the node is to prevent double deletion. The nodes are already being managed and deleted elsewhere in the code.
   - Memory management conflicts also occur when the destructor is not changed, multiple associations for nodes(more than parent-child) may become problematic when managing the memory within the destructor.
   - This is also an approach to make the tree management more flexible, given how I already made changes to the function split_leaf in node class.
   - Having a destructor could also disrupt processes like remove, merging or balancing operations within the code.
   - To conclude, this version provides a safer default by not engaging in direct deletions in the destructor.