- How did I test tree.h ?

  1) Using google tests. I have add google tests as submodule and added version random valued tests.

```
TEST_F(RandomizedTreeTest, RandomOperations) {
  tree_t btree(3);
  std::map<int, int> referenceMap;
  const int TOTAL_OPERATIONS = 1000;
  std::set<int> keysInserted;

  for (int opIndex = 0; opIndex < TOTAL_OPERATIONS; ++opIndex) {
    int randomKey = getRandomKey();
    int randomValue = getRandomValue();
    int operationType = getRandomKey() % 3; // 0, 1, or 2

    try {
      switch (operationType) {
        case 0: // Set
          btree.set(randomKey, randomValue);
          referenceMap[randomKey] = randomValue;
          keysInserted.insert(randomKey);
          logOperation("set", randomKey, randomValue);
          break;
        case 1: // Get
          if (referenceMap.find(randomKey) != referenceMap.end()) {
            ASSERT_EQ(btree.get(randomKey), referenceMap[randomKey]) << log.str();
            logOperation("get", randomKey, randomValue);
          } else {
            ASSERT_THROW(btree.get(randomKey), std::invalid_argument) << log.str();
            logOperation("get", randomKey);
```

  2) Manual tests with 100 value set, remove and get.

```
16  void testComplexScenario(tree_t& tree) {
17      // Insert a series of values
18      for (int i = 0; i < 100; i++) {
19          tree.set(i, i * 10);
20      }
21
22      // Check if all values are inserted correctly
23      bool insertionTestPassed = true;
24      for (int i = 0; i < 100; i++) {
25          if (tree.get(i) != i * 10) {
26              std::cout << "Insertion test failed at key " << i << std::endl;
27              insertionTestPassed = false;
28              break;
29          }
30      }
31      if (insertionTestPassed) {
32          std::cout << "All values inserted correctly." << std::endl;
33      }
34
35      // Remove a subset of values
36      for (int i = 0; i < 100; i += 2) {
37          tree.remove(i);
38      }
39
40      // Check if removals were successful
41      bool removalTestPassed = true;
42      for (int i = 0; i < 100; i += 2) {
43          try {
44              tree.get(i);
```

Bugs)

1.

```
while (!node->is_leaf) { // fix1: !
  node = node->get_child(key);
}
return node;
```

Type: Logical bug: get child if not leaf

How found: Read the code and found immediately

How Fixed: add ! (not)

2.

```
~node_t() {
  // for (node_t *child : down) {
  //   delete child;
  // } // fix2
}
```

Type: null pointer access. Deleted children who are referanced from other place
How found: My remove gave Segmentation Error, cause all values of children were null
How fixed: Commented . (Later need to check for memory leak).

3.

```
std::tuple<int, node_t *, node_t *> split_leaf() {
  node_t *left_ = new node_t(up, true, left, this); // fix3: left -> left_
  int mid = key_list.size() / 2;

  left_->key_list = std::vector<int>(key_list.begin(), key_list.begin() + mid);
  left_->value_list =
      std::vector<int>(value_list.begin(), value_list.begin() + mid);

  key_list.erase(key_list.begin(), key_list.begin() + mid);
  value_list.erase(value_list.begin(), value_list.begin() + mid);

  return std::tuple<int, node_t*, node_t*>(key_list[0], left_, this);
}
};
```

Type: Wrong variable name
How found: node has left attribute and this will overwrite it
How fixed: changed the name to left_

4.

```
317     }
318     // delete node; fix4
```

Type: null pointer access
How found: later referencing deleted node
How fixed: commented (need to check for memory leak)