我是如何测试 tree.h 的

我用了多种方法调试 tree.h,包括：

1.编写随机生成的测试用例进行调试；

2.在 tree.h 中编写了一个简单的遍历函数，使得可以在控制台中输出这棵多叉树的结构与键值；

3.在 main.cpp 中编写一个简单的过程，使得可以在控制台中对树进行手动的插入删除；

4.仔细阅读 tree.h 中各函数的声明，理解多叉树是如何实现的，并尝试在代码中找到逻辑错误等 bug；

5.使用各种工具，例如在 linux 虚拟机环境下使用了 valgrind 工具来对程序进行内存泄漏的检测，以及使用 IDE 的 gdb 调试工具进行逐过程的调试，检查局部变量的数值是否有问题。

我发现了哪些 bug

1.通过原来给出的测试用例以及在 ide 查看调用堆栈，我发现在 tree 类的函数 find_leaf 中有一个逻辑错误 bug：while 循环的条件显然应该是 node 不是叶子节点，因此在前面加上一个 !；

```
node_t *find_leaf(int key) {
  node_t *node = root;
  while (node->is_leaf) {
    node = node->get_child(key);
  }
  return node;
}
```

```
node_t *find_leaf(int key) {
  node_t *node = root;
  while (!node->is_leaf) {//1.原来少一个！ 显然逻辑错误
    node = node->get_child(key);
  }
  return node;
}
```

2.通过原来的测试用例以及在 IDE 中查看调用堆栈，我发现在 node 类的函数 split_leaf 中，第一行的初始化参数的第三个 left 前应加上 this->，否则 left 节点会指向自己，为逻辑错误；

```
std::tuple<int, node_t *, node_t *> split_leaf() {
  node_t *left = new node_t(up, true, left, this);
  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 {key_list[0], left, this};
}
};
```

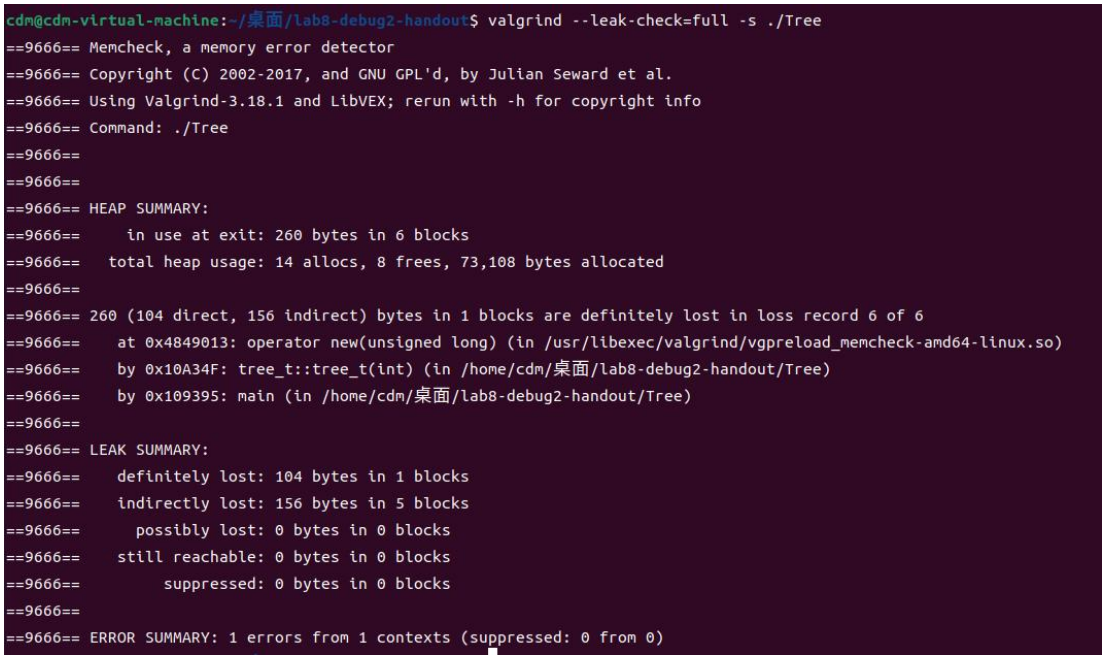
```
std::tuple<int, node_t *, node_t *> split_leaf() {
  node_t *left = new node_t(up, true, this->left, this);//2.left本来没有this
  int mid = key_list.size() / 2;

  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 {key_list[0], left, this};
}
};
```

3.通过自己编写的测试用例，我发现在试图删除一个叶节点上的键值时，若叶节点只有这一个键值，就会报错。经过调试我发现在 tree 类中的 remove_from_leaf 函数中存在错误，根据我的理解，若只剩一个键值，叶节点的父节点中的索引应该尝试修改为叶节点的右邻节点的第一个键值；若叶节点不存在右邻节点，这个分支情况下该如何操作我没有想出来，所以只能先空着了；

```
void remove_from_leaf(int key, node_t *node) {
  int index = node->index_of_key(key);
  if (index == -1) {
    throw std::invalid_argument("key not found");
  }
  node->key_list.erase(node->key_list.begin() + index);
  node->value_list.erase(node->value_list.begin() + index);
  if (node->up) {
    int index_in_parent = node->up->index_of_child(key);
    if (index_in_parent)
      node->up->key_list[index_in_parent - 1] = node->key_list.front();
  }
}
```

```
void remove_from_leaf(int key, node_t *node) {
  int index = node->index_of_key(key);
  if (index == -1) {
    throw std::invalid_argument("key not found");
  }
  node->key_list.erase(node->key_list.begin() + index);
  node->value_list.erase(node->value_list.begin() + index);
  if (node->up) {
    int index_in_parent = node->up->index_of_child(key);
    if (index_in_parent) {//
      if(node->key_list.size()) node->up->key_list[index_in_parent - 1] = node->key_list.front();
      else {
        if (node->right) node->up->key_list[index_in_parent - 1] = node->right->key_list.front();
        else {}
      }
    }
  }
}
```

4.在 linux 环境下使用 valgrind 工具调试时，发现程序有严重的内存泄露。经过测试，我大致确定了内存泄露应该是由 remove 函数中的过程引起的，大概率是其对节点的 down 进行了修改，使得在析构函数通过 down 来递归删除节点时有很多节点没有办法遍历到，最终产生了内存泄漏。由于我能力有限，并没有找到这个问题的解决办法；



5.在调试中还发现某些节点的 left 和 right 是存在问题的，会导致合并节点时产生错误，但由于能力有限，未能确定 bug 的根源在哪，也无法解决此问题。