1 测试程序的设计思路

根据评分标准中要求，我们在 BinaryNode 结构体添加了 height 来表示树的高度。因为子树高度是对数进行 AVL 修改的关键判断因素，所以随后我们随后通过一系列函数来进行计算子树的高度。子树高度主要是取左右子树的最大值再 +1 计算而来，我们可以设计一个简单的递归程序来实现从末端树枝叠加至目标节点。随后设计函数计算 VAL 的判据，即左边分支高度与右边分支高度之差。若差值大于 1，则明显左边过长，差值小于-1，则右边过长。

```
378     int height(BinaryNode *&t) {
379         return t == nullptr ? 0 : t ->height;
380     }
381
382     void calculate_height(BinaryNode *&t) {
383         if (t != nullptr)
384             t -> height = std::max(height(t -> right), height(t -> left)) + 1;
385     }
386
387     int get_balancefactor (BinaryNode *&t) {
388         int balancefactor = height(t->left) - height(t->right);
389         return balancefactor;
390     }
```

图 1: function about height

随后我们设计 VAL 更改树结构的函数。右旋转为例进行说明。首先右旋转一次即需要创建一个新的结构体指针指向当前结点的左节点，我们称其为新节点。随后我们需要把将新节点的新右子节点连接将当前节点上，所以我们首先要将曾经的右子节点连接到当前节点的左节点上以保证树结构的完整性。进行完这两步操作之后，可以对节点高度进行一次迭代，随后将父节点指向当前节点的指针修改为指向新节点的指针，完成一次右旋转。

左旋转与右旋转逻辑类似，只是中间左右节点相反。而其余两种情况则是需要考虑子节点的 VAL 判据。选择决定是否需要再进行左（右）旋转前先对右（左）子节点先进行一次右（左）旋转。

```
392     void balance (BinaryNode *&t) {
393         if (t != nullptr) {
394             if (get_balancefactor(t) > 1) {
395                 if (get_balancefactor(t -> left) <= 0) {
396                     BinaryNode *node1 = t -> left -> right;
397                     t -> left -> right = node1 -> left;
398                     node1 -> left = t -> left;
399                     calculate_height(t -> left);
400                     calculate_height(node1);
401                     t-> left = node1;
402 
403                     BinaryNode *node2 = t -> left;
404                     t -> left = node2 -> right;
405                     node2 -> right = t;
406                     calculate_height(t);
407                     calculate_height(node2);
408                     t = node2;
409 
410                 } else {
411                     BinaryNode *node = t -> left;
412                     t -> left = node -> right;
413                     node -> right = t;
414                     calculate_height(t);
415                     calculate_height(node);
416                     t = node;
417                 }
418 
419             }
420             else if (get_balancefactor(t) < -1) {
421                 if (get_balancefactor(t -> right) >= 0) {
422                     BinaryNode *node1 = t -> right -> left;
423                     t -> right -> left = node1 -> right;
424                     node1 -> right = t -> right;
425                     calculate_height(t -> right);
426                     calculate_height(node1);
427                     t -> right = node1;
428 
429                     BinaryNode *node2 =  t -> right;
430                     t ->right = node2 -> left;
431                     node2 -> left = t;
432                     calculate_height(t);
433                     calculate_height(node2);
434                     t = node2;
435 
436                 } else {
437                     BinaryNode *node = t -> right;
438                     t -> right = node -> left;
439                     node -> left = t;
440                     calculate_height(t);
441                     calculate_height(node);
442                     t = node;
443                 }
444 
445             }
446         }
447         else if (t ==nullptr) {
448             return;
449         }
450     }
```

图 2: VAL_balance

最后是对 remove 函数的修改，只需要在 remove 节点之后调用上述函数即可。

```
459     void remove(const Comparable &x, BinaryNode *&t) {
460         if (t == nullptr) {
461             return;
462         }
463         if (x < t->element) {
464             remove(x, t->left);
465         } else if (x > t->element)
466         {
467             remove(x, t->right);
468         }
469         else if (t->left != nullptr && t->right != nullptr)
470         {
471             BinaryNode *oldNode = t;
472             BinaryNode *newNode = detachMin(t->right);
473             t = newNode;
474             t -> right = oldNode -> right;
475             t -> left = oldNode -> left;
476             delete oldNode;
477         } else {
478             BinaryNode *oldNode = t;
479             t = (t->left != nullptr) ? t->left : t->right;
480             delete oldNode;
481         }
482
483         if (t != nullptr) {
484             calculate_height(t);
485             balance(t);
486         }
487     }
```

图 3: remove

2 测试的结果

测试结果一切正常。

输出为:

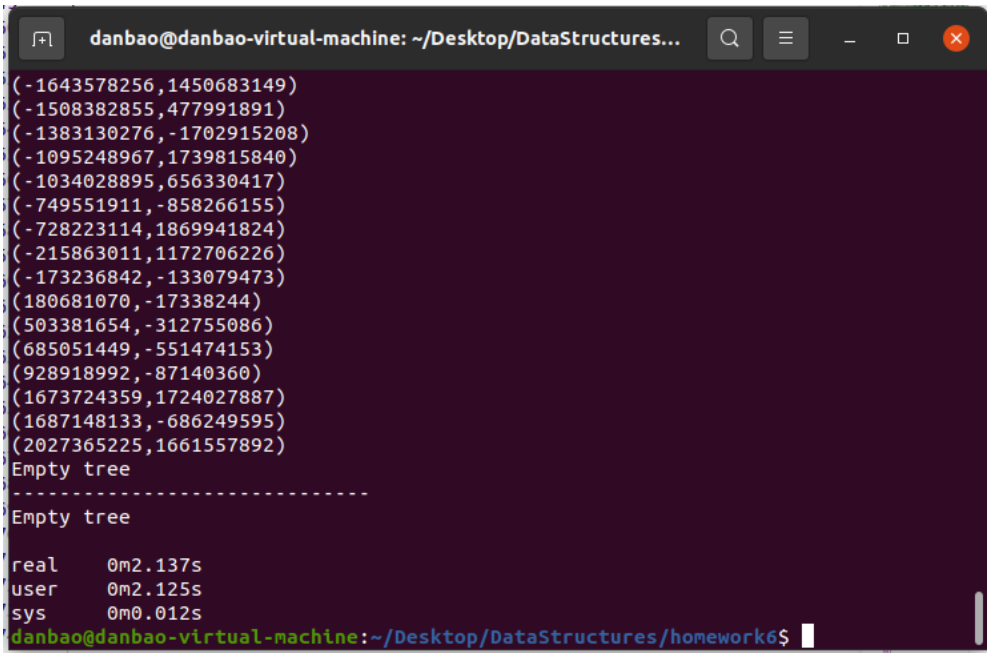

图 4: output

3 bug 报告

一切正常。