1 detachMin() 函数

1.1 功能

查找以 t 为根的子树中的最小节点，返回这个节点，并从原子树中删除这个节点。

1.2 参数

指向一个子树 root 的指针，并且为引用，因此函数内部修改 t 时，直接修改实参。

1.3 返回值

指向被删除（子树中的最小节点）的节点的指针。

1.4 功能实现

一共分两种情况。

1. t 所指的子树只有一个根，或根只有右节点：

   此时最小节点就是根。指向 t 的指针修改为指向`t->right`即可。

2. t 所指的根有左节点：

   不断向左节点寻找子树中的最小节点，最终得到指向最小节点的指针，此时删除这个最小节点就是 remove() 中“只有一个孩子”的情况，只需简单修改指针指向即可。

注：根据 remove() 中调用 detachMin() 的逻辑，t 一定不会是空树。

2 remove() 函数

2.1 功能

删除指定节点。若找不到该节点，则输出错误提示。

2.2 参数

外部：要删除的节点内容。内部：要删除的节点内容，以及指向当前节点的指针 t。同样 t 为引用，修改实参。

2.3 返回值

无

2.4 功能实现

先执行寻找过程，若要找的节点小于当前节点，则向左节点寻找，反之向右节点寻找。若最终当前节点为 nullptr，则说明找不到该节点，输出错误提示。若此时找到了该节点，则分多种情况讨论。设此时 t 指向的节点为 currentNode，注意此时 t 为 currentNode 的父节点指向 currentNode 的指针，由于 t 是引用，修改 t 直接修改父节点的 left 或 right 的指针。

1. 要删除的节点只有一个孩子，或没有孩子:

   指向 currentNode 的节点改为指向 currentNode 的孩子，然后删除 currentNode。

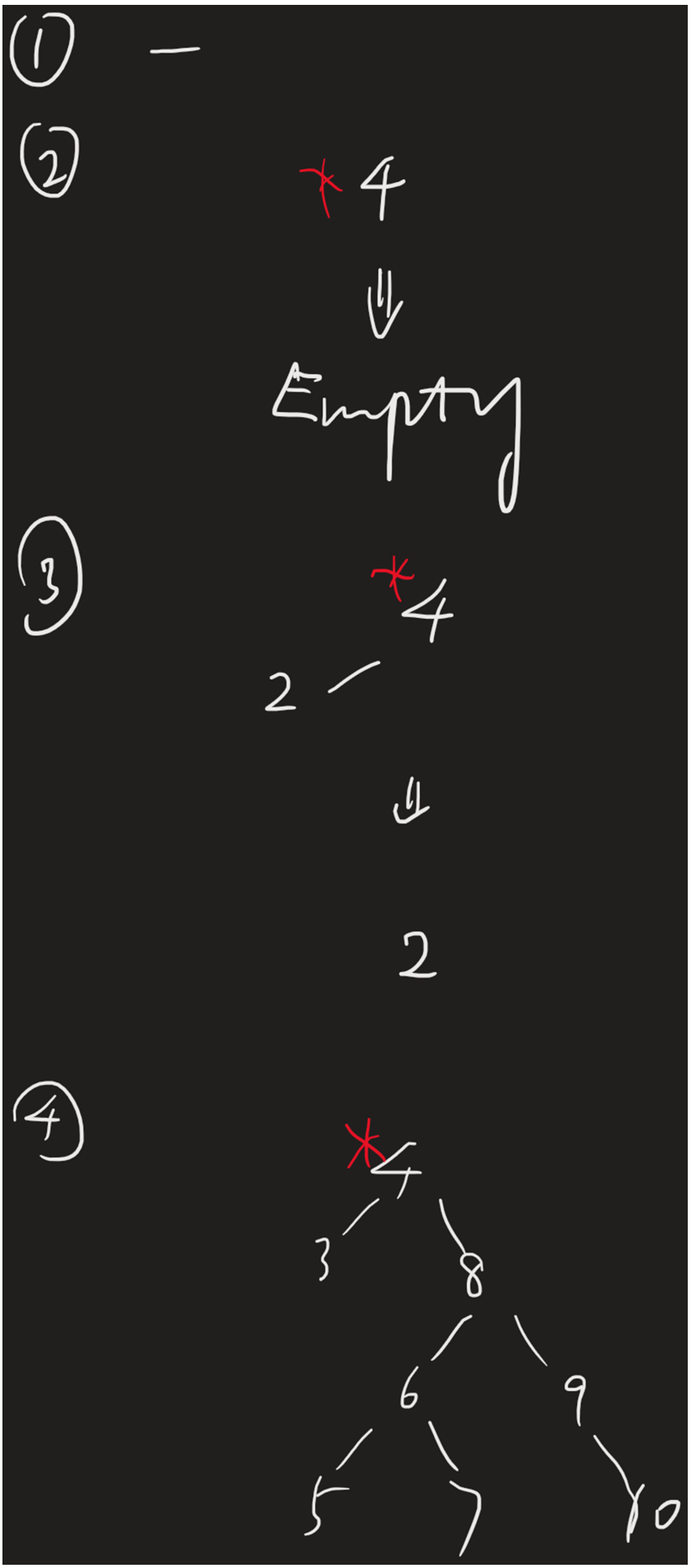
2. 要删除的节点有两个孩子:

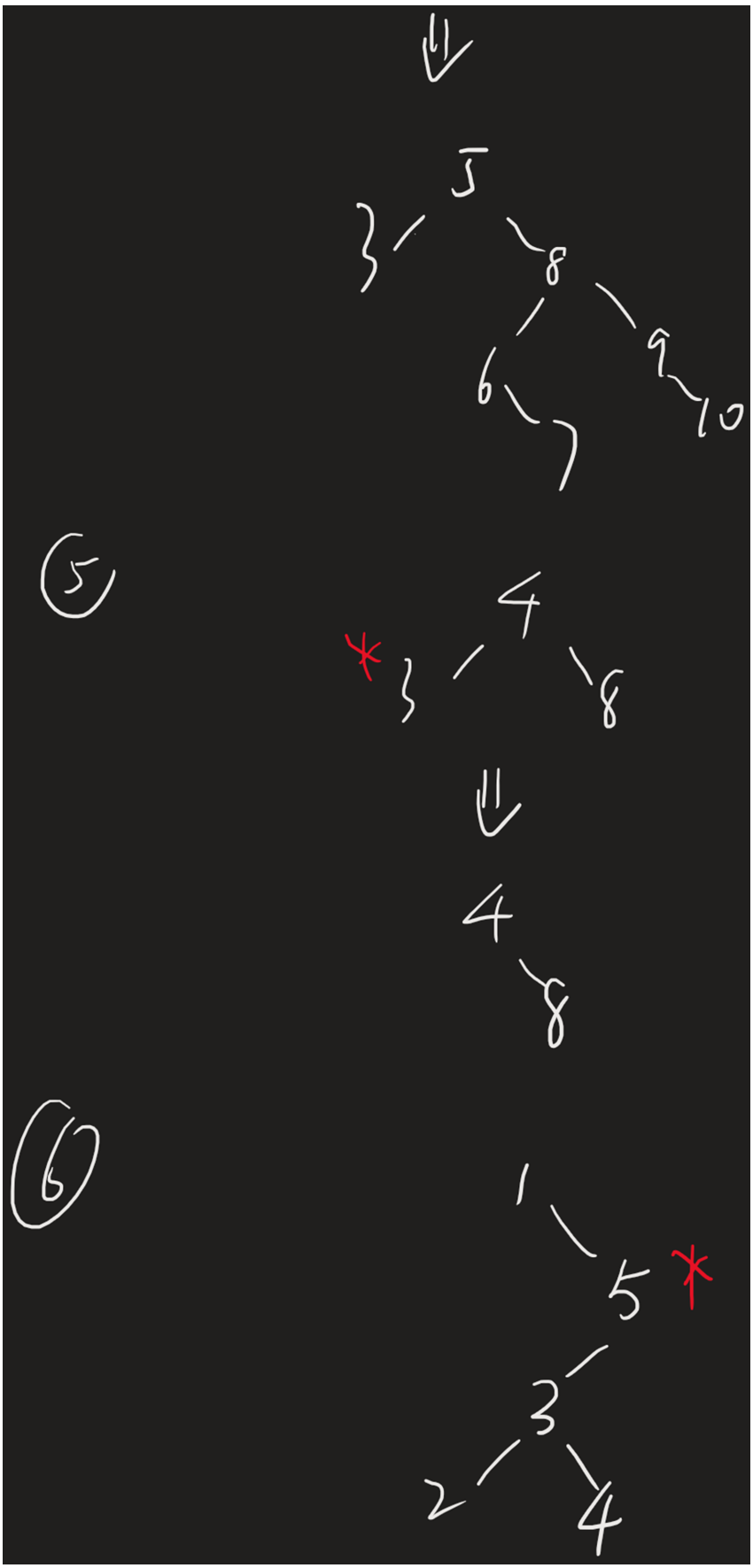
   使用 detachMin() 函数找到 currentNode 右节点的最小节点，使它从树中脱离出来，通过修改若干指针的方法替换 currentNode 位置，再将 currentNode 删除。

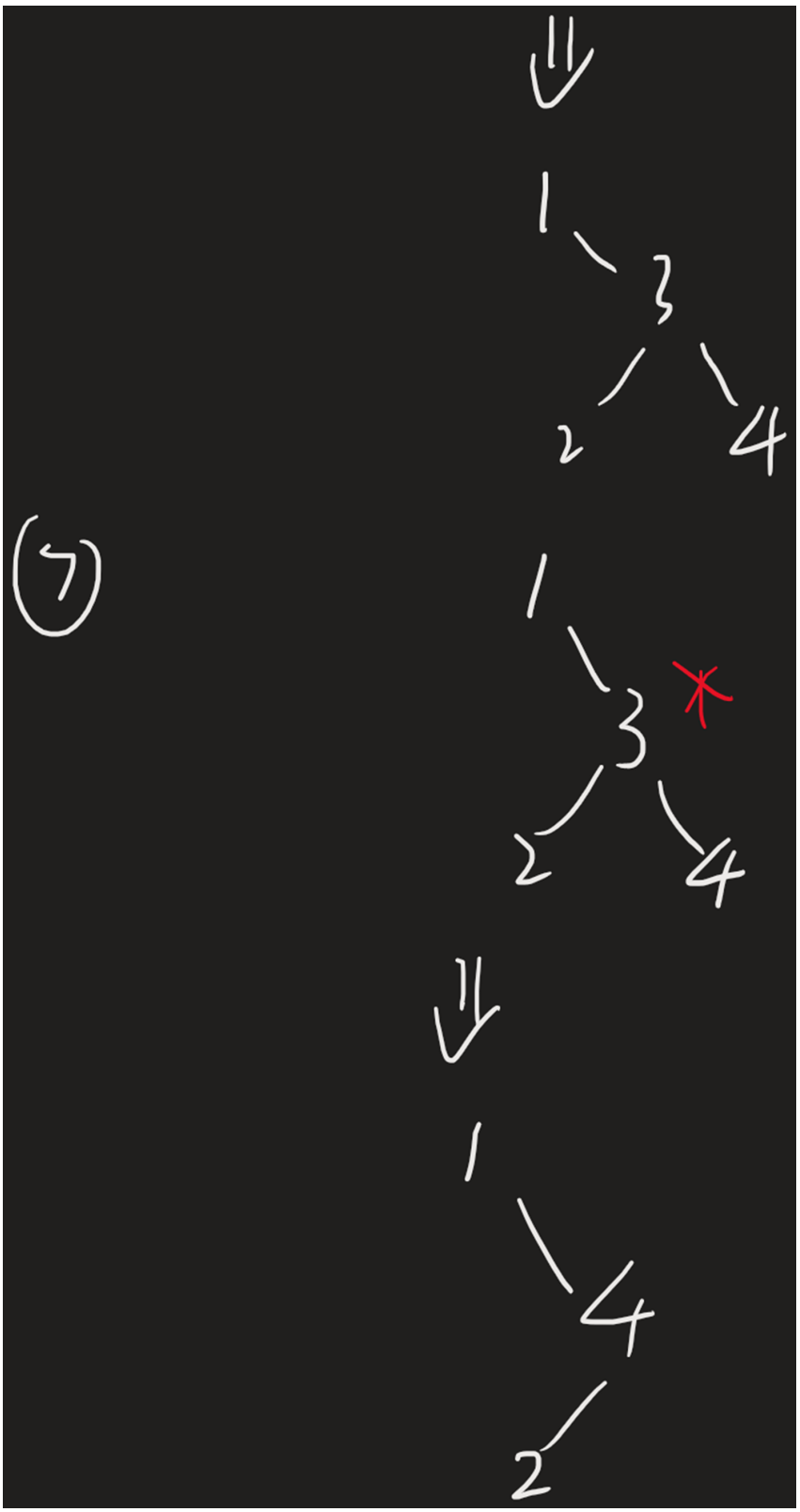
3 测试结果与分析

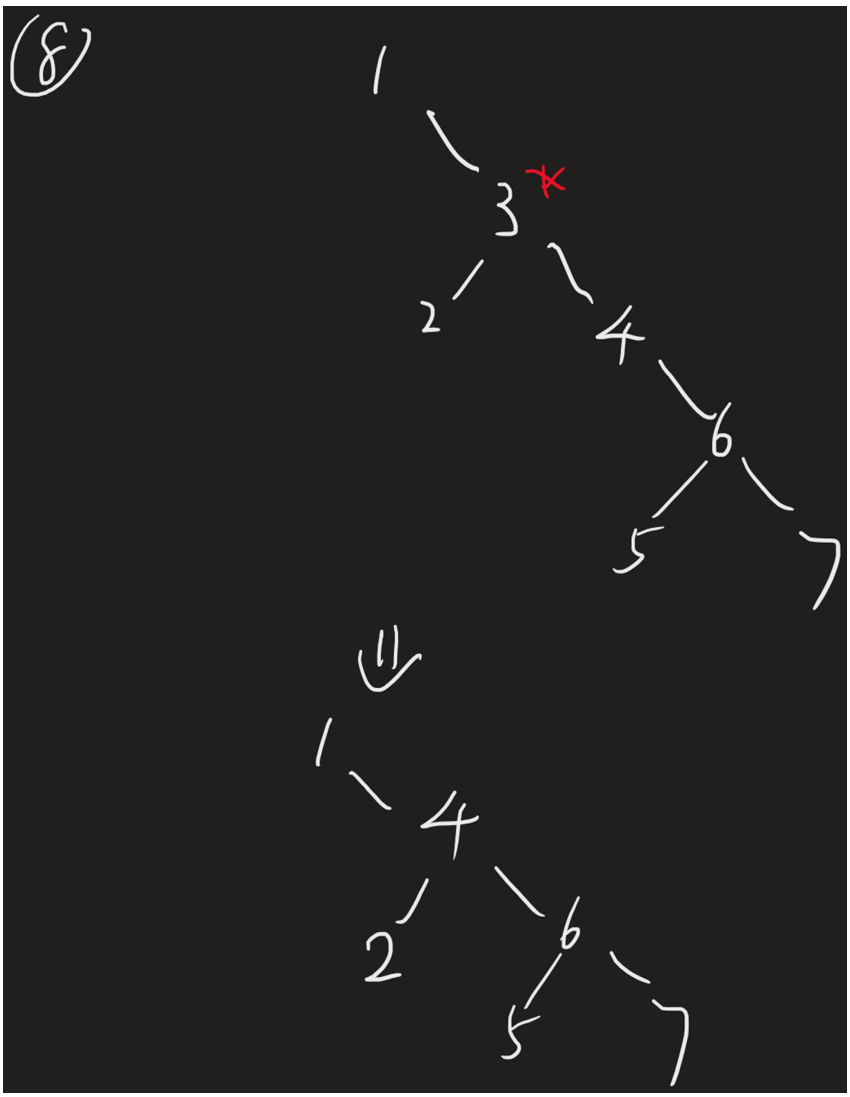
有几个要点需要考虑：要删除的元素是否找得到？currentNode 是否为根节点？currentNode 有几个孩子？currentNode 的右节点有几个孩子？据此设计测试程序（详细测试方案见输出内容），最后再进行空树的异常测试。

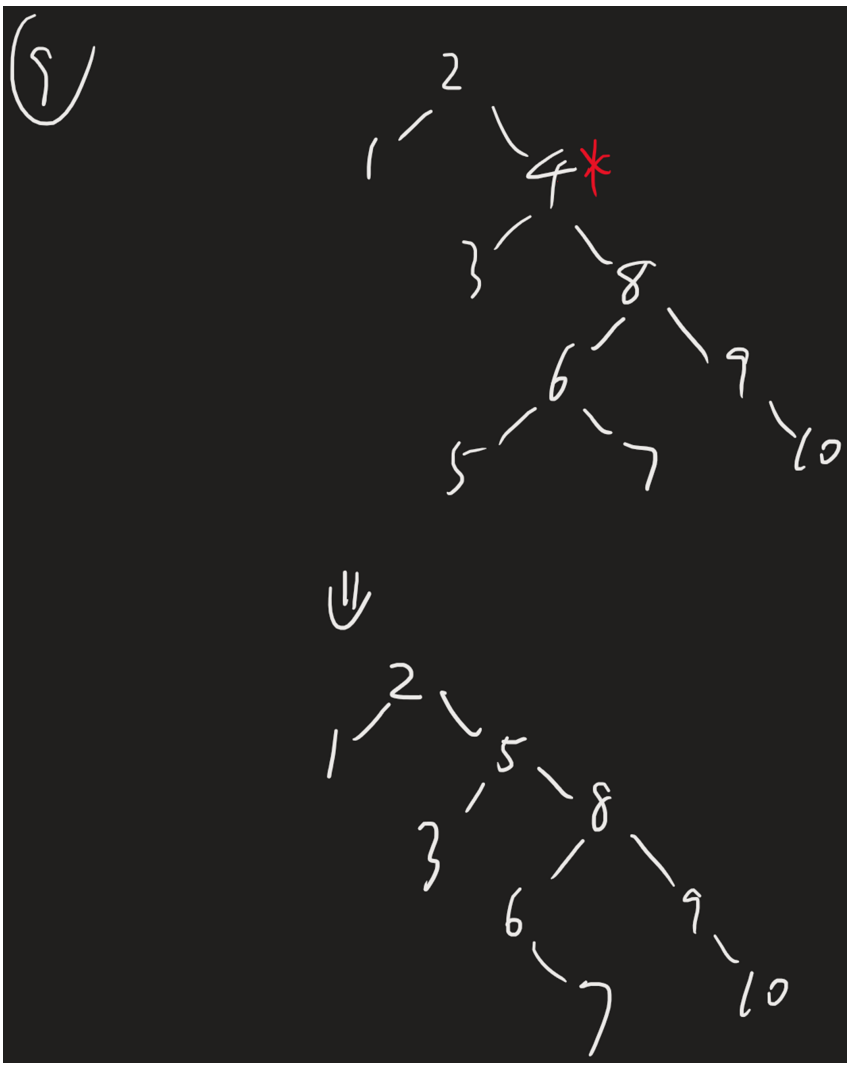
附上测试对应的分析图：


①
—
②
4
Empty
③
4
2
2
④
4
3
8
6
9
5
7
10


(5)
(6)


1
3
2
4
(7)
1
3
2
4
1
4
2


(8)
1
3
2
4
6
5
7
(11)
1
4
2
6
5
7



程序输出内容:

```
1. 找不到要删除的节点:
        删除前的树:
        中序遍历        2       3       5       7       8
        前序遍历        5       3       2       7       8
删除 6
6 is not found.
        删除后的树:
        中序遍历        2       3       5       7       8
        前序遍历        5       3       2       7       8
2. 要删除的节点为根节点, 且该节点没有孩子:
        删除前的树:
        中序遍历        4
        前序遍历        4
删除 4
        删除后的树:
Empty tree
```

```
Empty tree
```

3. 要删除的节点为根节点，且该节点只有一个孩子:

```
        删除前的树:
        中序遍历        2       4
        前序遍历        4       2
删除 4
        删除后的树:
        中序遍历        2
        前序遍历        2
```

4. 要删除的节点为根节点，且该节点有两个孩子:

```
        删除前的树:
        中序遍历        3       4       5       6       7       8       9       10
        前序遍历        4       3       8       6       5       7       9       10
删除 4
        删除后的树:
        中序遍历        3       5       6       7       8       9       10
        前序遍历        5       3       8       6       7       9       10
```

5. 要删除的节点不是根节点，且该节点没有孩子:

```
        删除前的树:
        中序遍历        3       4       8
        前序遍历        4       3       8
删除 3
        删除后的树:
        中序遍历        4       8
        前序遍历        4       8
```

6. 要删除的节点不是根节点，且该节点只有一个孩子:

```
        删除前的树:
        中序遍历        1       2       3       4       5
        前序遍历        1       5       3       2       4
删除 5
        删除后的树:
        中序遍历        1       2       3       4
        前序遍历        1       3       2       4
```

7. 要删除的节点不是根节点，该节点有两个孩子，且右节点没有孩子:

```
        删除前的树:
        中序遍历        1       2       3       4
        前序遍历        1       3       2       4
删除 3
        删除后的树:
        中序遍历        1       2       4
        前序遍历        1       4       2
```

8. 要删除的节点不是根节点，该节点有两个孩子，且右节点只有一个右孩子:

```
        删除前的树:
```

```
        中序遍历        1       2       3       4       5       6       7
        前序遍历        1       3       2       4       6       5       7
删除 3
        删除后的树:
        中序遍历        1       2       4       5       6       7
        前序遍历        1       4       2       6       5       7
9. 要删除的节点不是根节点，该节点有两个孩子，且右节点有两个孩子:
        删除前的树:
        中序遍历        1       2       3       4       5       6       7       8       9       10
        前序遍历        2       1       4       3       8       6       5       7       9       10
删除 4
        删除后的树:
        中序遍历        1       2       3       5       6       7       8       9       10
        前序遍历        2       1       5       3       8       6       7       9       10
10. 对空树执行remove操作
        删除前的树:
Empty tree

Empty tree

terminate called after throwing an instance of 'UnderflowException'
make: *** [Makefile:23: run] 已放弃 (核心已转储)
```

测试结果一切正常。运行结果：在 main 函数结束时调用析构函数。

我用 valgrind 进行测试，发现没有发生内存泄露。

```
All heap blocks were freed -- no leaks are possible
ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

---

4 （可选）bug 报告

我发现了一个 bug，触发条件如下：

1. main 函数中，一旦一个名为 name1 的 List 被创建，那么它在 main 结束前永远无法全死掉，最多只能 clear()，这不太符合常识（当然，我们不可能在 main 中使用析构函数）。比如说，我创建了一个名为 temp 的 List，用完之后 clear，然后又想用`List(std::initializer_list<Object> il)`的方法初始化这个 temp，但是这是 redeclaration，编译器会报错。要么用循环 push，要么换一个变量名，但是都不太舒服。注：因为复现 bug 会导致编译错误，所以我没有在 main.cpp 中复现这个 bug。
2. 其他 bug 可能算作异常处理，所以并没有详细阐述。例如，pop、erase 时需要判断是否为空指针，find 参数前后顺序相反时的处理等。