摘要

本次编程作业主要重写了 `BST.h` 中的 `remove` 函数，实现了二叉搜索树里元素的删除，避免了使用递归的方式。

1 BST.h 的设计思路

1. 首先新增了函数 `BinaryNode *detachMin(BinaryNode *&t)`，主要作用是为了找到以 `t` 为 root 的二叉搜索树中最小元素的节点,并返回该节点。由于最小元素一定在 tree 的最左边,只要检测 `if(current->left!=nullptr)` 即可。同时对节点t做出以下修改：

```
//如果最小元素的节点不是根节点t
if(parent!=nullptr){
        parent->left=current->right;
}
//如果是根节点
else{
        t=current->right;
}
```

这里的 `parent` `current` 分别指的是最小元素所在节点的父节点和最小元素所在的节点。

2. 修改remove函数。这里分为五种情况：

(a) 最简单的情况：二叉树为空，直接 `return`

(b) 若二叉树不为空，首先要找到需要删除的元素所在的节点，因此要追踪当前节点和父节点。

```
BinaryNode *parent=nullptr;
BinaryNode *current=t;
```

若要删除的元素不在 tree 里面，直接 `return`

(c) 如果要删除的元素只有一个子节点，这时候直接删除该元素，用该元素的子节点取代当前节点。

(d) 如果要删除的元素有两个子节点，而且要删除的元素就是根节点：此时根节点由右子节点中最小元素所在的节点取代，根节点的左子节点不变，右子节点的结构由 `BinaryNode *detachMin(current->right)` 之后的结果给出。

```
if(current->left!=nullptr && current->right!=nullptr){
BinaryNode *newroot=detachMin(current->right);
newroot->right=current->right;
//此时要删除的节点就是root，current指向root
if(parent==nullptr){
        t=newroot;
}
```

(e) 若要删除的元素有两个子节点，且不是根节点，这时候与 (c) 相比需要额外将新的根节点接到原来的父节点上。

```
//要删除的节点还有父节点，这时候要把newroot接到父节点上
else if(x<parent->element){
        parent->left=newroot;
}
else{
        parent->right=newroot;
}
```

2 test_BST.cpp 测试思路

1. 首先新建空的 BST，删除元素 1, 测试情况 (a)。插入 10 5 15 3 7 12 18 20 1 4 到空的二叉搜索树中，打印检测是否插入成功。之后删除元素 50，测试情况 (b)。之后删除元素 5 测试情况 (e)，删除元素 7 测试情况 (c), 删除元素 15 测试情况 (e), 删除元素 10 测试情况 (d)。每次都打印观察结果。

3 测试的结果

测试结果一切正常。

最终在终端输出的结果如下：

```
g++ -o test test_BST.cpp
Running test:
./test
after removing 1
Empty tree
Initial Tree:
1
3
4
5
7
10
12
15
18
20
Minimum element: 1
Maximum element: 20
Contains 7? Yes
Contains 20? Yes
Tree after removing 50:
1
3
4
5
7
10
12
15
18
20
Tree after removing 5:
1
3
4
7
10
12
15
18
20
Tree after removing 7:
1
3
4
10
12
```

```
15
18
20
Tree after removing 15:
1
3
4
10
12
18
20
Tree after removing 10:
1
3
4
12
18
20
```