Lab 1：Scapegoat Tree

一、 实验介绍

Scapegoat Tree（替罪羊树）是一种可以自平衡的二叉搜索树。相比于基于旋转更改节点间关系的 AVL 树、红黑树、Spray 树等，Scapegoat 树的平衡方法相当简单而暴力，它不强制保证所有子树完全平衡，允许一定程度的不平衡存在，但有一个平衡底线。当某次插入或删除操作导致一棵子树超出这个平衡底线，不会去想办法调整平衡，而是直接对不平衡的子树进行重构来使其平衡。这个被重构的不平衡子树的根节点，替插入或删除操作的节点承担了不平衡的过错，所以把它称作 Scapegoat（替罪羊）。Scapegoat 树在插入、查找、删除操作上均能达到 O(log n) 的平摊时间复杂度。

下面简要介绍 Scapegoat 树中所使用到的概念。

一个节点 x 是 **α 权重平衡**（α-weight-balanced）的，当其满足

$$size(left[x]) \leq \alpha * size(x)$$
$$size(right[x]) \leq \alpha * size(x)$$

其中 **α** (0.5 < α < 1)称为平衡因子，表明了节点的左右子树的大小不会超过整个子树的 α 倍，所以简而言之 Scapegoat 树的平衡就是确保左右子树不会有一边过重。 易知，当 α 值越高，平衡要求就越小，为 1 的时候连单链表都能够满足，为 0.5 的时候则只有几乎完整的二叉树能够达到。一般的 Scapegoat 树的 α 取值在 0.6~0.8 之间。

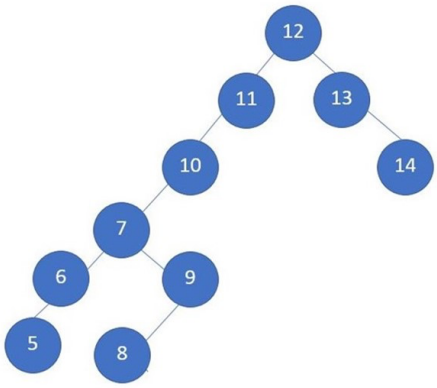
一个树 T 是 **α 高度平衡**（α-height-balanced）的，当其满足

$$h(T) \leq \left\lfloor \log_{1/\alpha}(size(T)) \right\rfloor$$

$h(T)$ 是树的高度，为从根节点到叶子节点的最长路径，直观上理解上式，就是指一棵树的高度不大于与它相同大小的最高的 α 权重平衡树的高度。反之，如果超出了这个大小，那么一定说明存在不能满足 α 权重平衡的子树。

因此如果插入一个过深的节点违背了上述高度平衡要求，就会触发一次 Scapegoat 树的重构操作，重构的方法也很简单，就是从该节点向上找到 Scapegoat，然后将整棵子树拍平然

后递归方式重构为完全二叉树。



α = 2/3

It's a balanced Scapegoat Tree in our definition because with height = 5, n = 10, there is 5 ⩽ floor[$\log_{3/2}(10)$] = floor(5.68) = 5 although there exists α-weight-unbalanced nodes

事实上，Scapegoat 树的平衡要求和实现方法也有很多变种，但基本思想大致相同，一些实现可能省去了高度平衡的相关部分，直接保证所有节点的权重平衡，也有一些删除实现采用“惰性删除”而非直接删除，这些都是可以的。本次 lab 为了统一，基于早期提出 Scapegoat 树的论文（随 lab 发布），更详细的内容可以参考论文，以按要求实现为主，有兴趣的同学可以自行研究其理论依据和其他 Scapegoat 树的相关实现。

二、 实验要求

本次 Lab 中，我们要求使用 C++实现 Scapegoat 树的插入、查找、删除功能。实现部分的 α 统一采用 0.6。

1. Insert

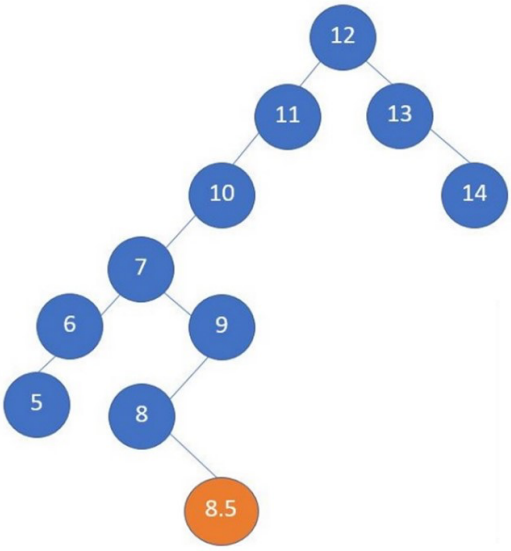
Scapegoat 树的插入操作首先找到插入点，然后判断其是否违背了 α 高度平衡的条件，因此需要记录新插入节点的深度（注：新节点的深度不一定等于树的高度，请使用深度判断高度平衡要求）。如果不违背，则说明树仍然平衡，返回即可。违背则需要找到 Scapegoat 进行重构。找 Scapegoat 的方式有多种，比如论文里提到我们可以从插入节点向根节点往上爬找到第一个不满足 α 权重平衡的节点，或者可以找到一个最深的祖先节点，它的子树高度不能满足 α 高度平衡。这里为了实现统一，我们采用第一种方式，即找到第一个不满足 α 权重平衡的节点作为 Scapegoat。找到 Scapegoat 之后，对以其为根节点的子树进行重构。重构时子树大小

如果为偶数，在选取中间节点时选择较小的那个。

如果插入的节点已存在，则不插入，本 lab 的 Scapegoat 树不对同值节点做要求。

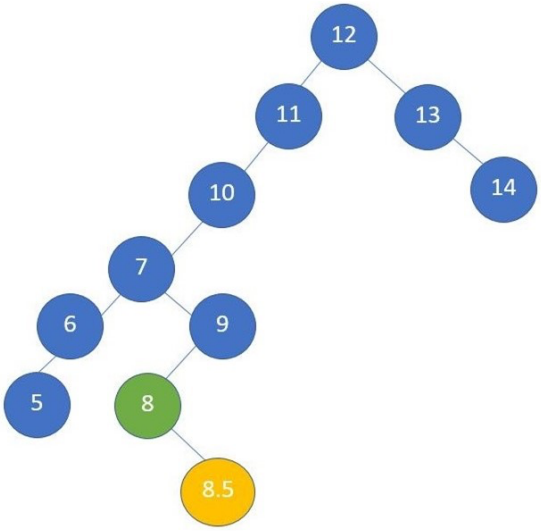
简而言之，插入过程的重构的核心在于，用是否违背高度平衡判断是否重构，用是否违背权重平衡寻找 Scapegoat。

An Example：



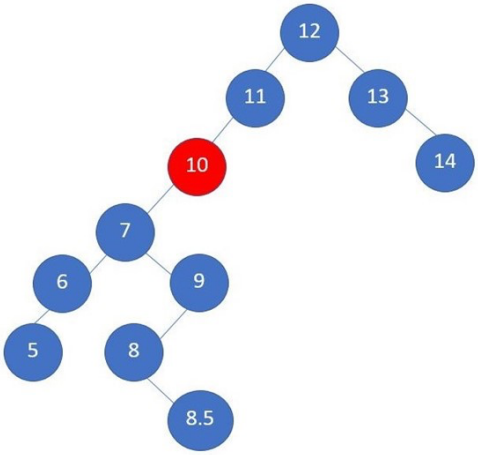
α = 2/3

Insert a new node 8.5, then height = 6 > floor[$\log_{3/2}(11)$] = floor(5.9) = 5, so we need a rebuild operation



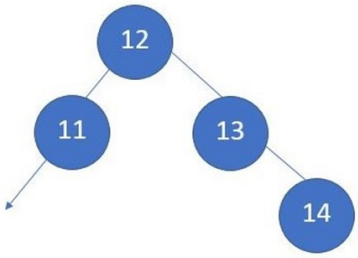
α = 2/3

We move up along the path to the root, check node 8.
For node 8, size(child) = max{size(left),size(right)} = 1, size(node) = 2, so size(child)/size(node) = ½ < α, then node 8 is α-weight-balanced and not is scapegoat.
Similarly, for node 9, there is 2/3 ⩽ α, not scapegoat
For node 7, there is 3/6 = ½ < α, not scapegoat

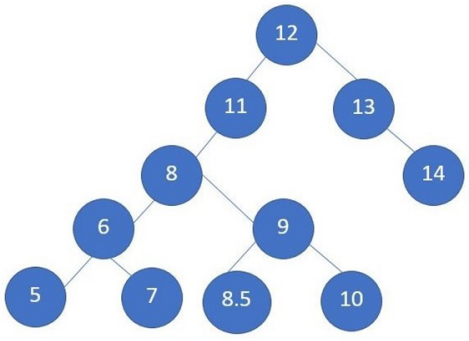


α = 2/3

For node 10, there is 6/7 > α, so node 10 is scapegoat



Flat with inorder traversal



Find the middle node to be the new root, and recursively do this in two sides to rebuild a complete binary tree

2. Search

Scapegoat 树的查找操作与一般的 BST 无异，不做赘述。

3. Delete

Scapegoat 树的删除操作相对于插入操作简单许多，首先像一般的 BST 一样直接删除对应节点，当删除节点左右子树都存在时，用找右子树的最小值方法，除此之外，还需要存储一个 max_size 表示曾经达到的最大节点数(注：因此需要在 Insert 操作时把这个值设为树实际 size 和 max_size 当中的最大值)，在任意一次删除操作完成后，如果有

$$size(T) < \alpha * \max\_size(T)$$

则重构整棵树（根节点为 Scapegoat），然后把 max_size 设为 size。

如果删除的节点不存在，则直接返回即可。

4. 输入输出

(1) 输入部分

本次实验的输入部分与 lab0 相同，通过文件给出指令序列，如下图所示：

```
Insert(1)
Insert(3)
Insert(6)
Search(3)
Search(5)
Search(6)
```

(2) 输出部分

本次实验的输出部分有两个：

（1） 所有的 Search 操作成功找到返回当前节点深度，没有找到返回“Not Found”，每个 Search 对应一行输出。

（2） 每个输入文件，在所有的指令执行完毕后，输出整个过程中的重构次数。

三、 实验评分

1. 代码运行

我们提供了 Makefile 工具，在主目录下执行 **make** 指令生成可执行程序 main。你可以通过下述指令查看自己的实现情况：

```
./main d 输入文件的路径
```

例如

```
./main d ./input/sgt_input1
```

将使用 input 文件夹下的 sgt_input1 文件中的指令序列用你的实现生成一个 Scapegoat 树，并将输出打印到标准输出中。你可以据此定义自己的指令序列用于 debug。

2. 代码评分

你可以在主目录下执行 **make grade** 指令（或者形如./main g 的指令）查看自己实现是否正确，如果不正确，会打印出你的输出与答案的差异，请检查是否正确实现以及按要求输出了，如果正确实现，你应该看到如下输出：

```
<<<<<<<< grade test >>>>>>>>>
  TEST1:    OK
  TEST2:    OK
  TEST3:    OK
  TEST4:    OK
  grade: 100/100
<<<<<<<< grade test over >>>>>>>>>
```

通过它们你将拿到该实验内容的正确性分数。

3. 性能分析书面报告

在上述实验过程中，我们可以看出 Scapegoat 树有一个可调节的平衡因子 α，理论上越大的 α 对应的树平衡要求越低，插入所需的平衡开销更小，但对应地，查找和删除等操作会更慢。本次实验的性能分析部分，需要你通过调整 α 的大小，分析 Scapegoat 树的插入、删

除、查找操作的平均时间随数据规模的变化，你的实验数据 α 应包括至少 5 个均匀分布在 0.6~0.8 范围内的点（如 0.6,0.65,0.7,0.75,0.8），数据规模应能反映出理论上三个操作的对数时间复杂度。表现形式图表和表格均可。

除此之外，你还需要与你在课上学过的至少一个基于旋转的平衡树的上述三个操作的性能作比较，并给出一些简单的建议或结论，比较开放，有数据支撑结论即可。

请将上述内容生成一个 PDF 文件。

你可能需要修改代码添加输出来收集实验数据，由于我们会使用输出判断实现的正确性，请注意保存一份能够通过先前 grade test 的版本。

4. 实验上交

你可以随意修改 ScapegoatTree.cpp 和 ScapegoatTree.h，必须提供 ScapegoatTree.h 中初始定义的 4 个接口函数。**提交实验请将 ScapegoatTree.cpp 和 ScapegoatTree.h 打包上传 Canvas，命名使用“学号+姓名+lab1”，如“520123456789+张三+lab1.zip”。**

性能分析书面报告同样上传到 Canvas 平台（会有单独的作业用于提交），要求 PDF 格式，命名无要求。

代码请提交能够通过 grade test 的版本，不要求提交性能分析的代码。

Lab1 截止日期为 2022.04.28 21:59PM。

四、 注意事项

1. 切勿**抄袭**！如若发现明显雷同现象，该实验评分将以 0 分计。
2. 注意实验截止日期，迟交或不交会视情况扣分，因疫情防控原因无法按时上交的请及时与助教联系说明原因，会酌情适当放宽期限。
3. 请注意按照实验要求完成，避免因为自动脚本评分造成的判分失误。
4. 有问题随时通过邮件或微信群联系负责 Lab 的助教姬浩迪、陈沛东。

五、参考资料

[1] Scapegoat tree - Wikipedia https://en.wikipedia.org/wiki/Scapegoat_tree

[2] Scapegoat Trees — Domicia Herring http://www.domiciaherring.com/scapegoat-trees