Homework 5: Red Black Tree V.S. SkipList V.S. AVL Tree

红黑树是对平衡二叉搜索树改进后的一种数据结构，与 AVL 树严格的维护平衡条件不同，红黑树通过维护红黑规则来达到 2-3-4 树（4阶 B 树）的效果，在不太损失查找性能的同时，提升了增删操作的性能。对于红黑树的更多细节和红黑规则的定义可以参考 slides 和其他资料，本实验中仅考察红黑树的查询和插入操作。

本次实验的目的是对比红黑树、跳表和 AVL 树的插入、查询开销。

Part 1: 红黑树的实现

红黑树的主体代码已经在文件 `rbtree.h` 中给出，但关键部分操作被略去，你需要理解代码逻辑后完善给出的代码。需要完善的部分（在源代码中用 **TODO** 标记出的部分）包括：

- `findNode` 函数（`rbtree.h` 文件第 **65** 行）：查找并返回某个节点
- `fixInsertion` 函数（`rbtree.h` 文件第 **91** 行，第 **97** 行）：插入节点后发生双红的修复函数

Tips：红黑树的实现中提供了打印函数，可供你调试使用

Part 2: 性能测试

将完善后的红黑树与之前作业中你的跳表（增长率取最优值）、AVL 树进行性能对比，具体的步骤如下：

1. 设计至少 **5** 组不同大小的输入集，并分别使用**顺序**和**乱序**两种输入方式（建议保存为文件而不是运行时再生成）；
2. 测试在红黑树、SkipList、AVL 树下，每组输入集使用**顺序**和**乱序**两种方式插入时的**耗时**和红黑树、AVL 树的**旋转操作次数**，并将结果制作为表格；
3. 设计至少 **5** 组不同大小的**查找集**（查找的 key 最好是已经存在的），并分别使用**顺序**和**乱序**两种输入方式；
4. 测试在红黑树、SkipList、 AVL 树下，每组输入集使用顺序和乱序两种方式查找时的**耗时**，并将结果制作为表格；
5. 分析所测红黑树、SkipList、 AVL 树的插入和查询操作开销是否符合理论（至少要对比在顺序和乱序输入场景下三种结构的插入操作的耗时和两种树的旋转次数的对比以及查找操作的耗时对比），并说明自己对数据结果差异的理解；

Part 3: 提交要求

你提交的内容应该包括：

- 你完善后的红黑树代码，SkipList 代码和 AVL 树代码；
- 你的测试参数的选取情况以及测试数据集；
- 你的**实验结果数据**以及对实验结果的**简单分析**；

Part 4: 注意事项

- 请将相关的代码和实验报告打包上传 Canvas，命名使用“学号+姓名+hw5”，如“521123456789+张三+hw5.zip”。

- **请勿抄袭**！课后作业采用倒扣分制，如果有遗漏或者得分不足会在最终成绩酌情减分，同时课后作业的内容会体现在期末试卷中，对同学们也是一种练习。
- 本次作业的截止时间是 **2023年4月2日23:59**，迟交将会酌情扣分。
- 有任何作业相关的问题可以询问 江玥璠、熊天磊 助教。