代码逻辑

实现了基于图着色的寄存器分配算法，核心流程包括简化（simplify）、合并（coalesce）、冻结（freeze）、溢出（spill）、选择（select）五个阶段

---

1. simplify()

功能：
在干扰图中查找邻居数小于寄存器数 k 的结点，并将其压入简化栈（simplifiedNodes），同时从图中删除该结点
细节：

- 遍历所有结点，跳过被保护（即在 movePairs 中）的结点
- 找到满足条件的结点后，压栈并删除，返回 true
- 若无可简化结点，返回 false

---

2. coalesce()

功能：
尝试合并属于同一 move 对的两个结点，以减少 move 指令
细节：

- 遍历 movePairs，取出一对 (dst, src)
- 若两结点间有干扰边，则不能合并
- 使用 Briggs 策略：合并后新结点的高阶邻居数（度数≥k）小于 k 才允许合并
- 若 src 是机器寄存器，交换 dst 和 src，保证 dst 为机器寄存器
- 合并后，删除 src 结点，将其邻居与合并集添加到 dst，并更新 movePairs 中所有 src 为 dst
- 合并成功返回 true，否则继续尝试，若无可合并的，返回 false

---

3. freeze()

功能：
当无法简化或合并时，解除某一 move 对的保护关系，使相关结点可以被简化
细节：

- 直接移除 movePairs 中的一个 move 对（及其相关的对），返回 true
- 若 movePairs 为空，返回 false

---

4. spill()

功能：
选择一个高干扰（度数最大）的结点作为溢出结点，压入简化栈，并从图中删除
细节：

- 记录度数最大的结点，执行压栈和删除操作，返回 true
- 若无可溢出结点，返回 false

---

5. select()

功能：
从简化栈中依次弹出结点分配颜色，若无法分配则标记为溢出
细节：

- 先为机器寄存器分配固定颜色
- 对每个弹出的结点，统计其邻居已用颜色，分配未被占用的颜色
- 若无可用颜色，则将该结点及其合并集标记为溢出
- 最后调用 checkColoring() 检查着色合法性

Git Graph