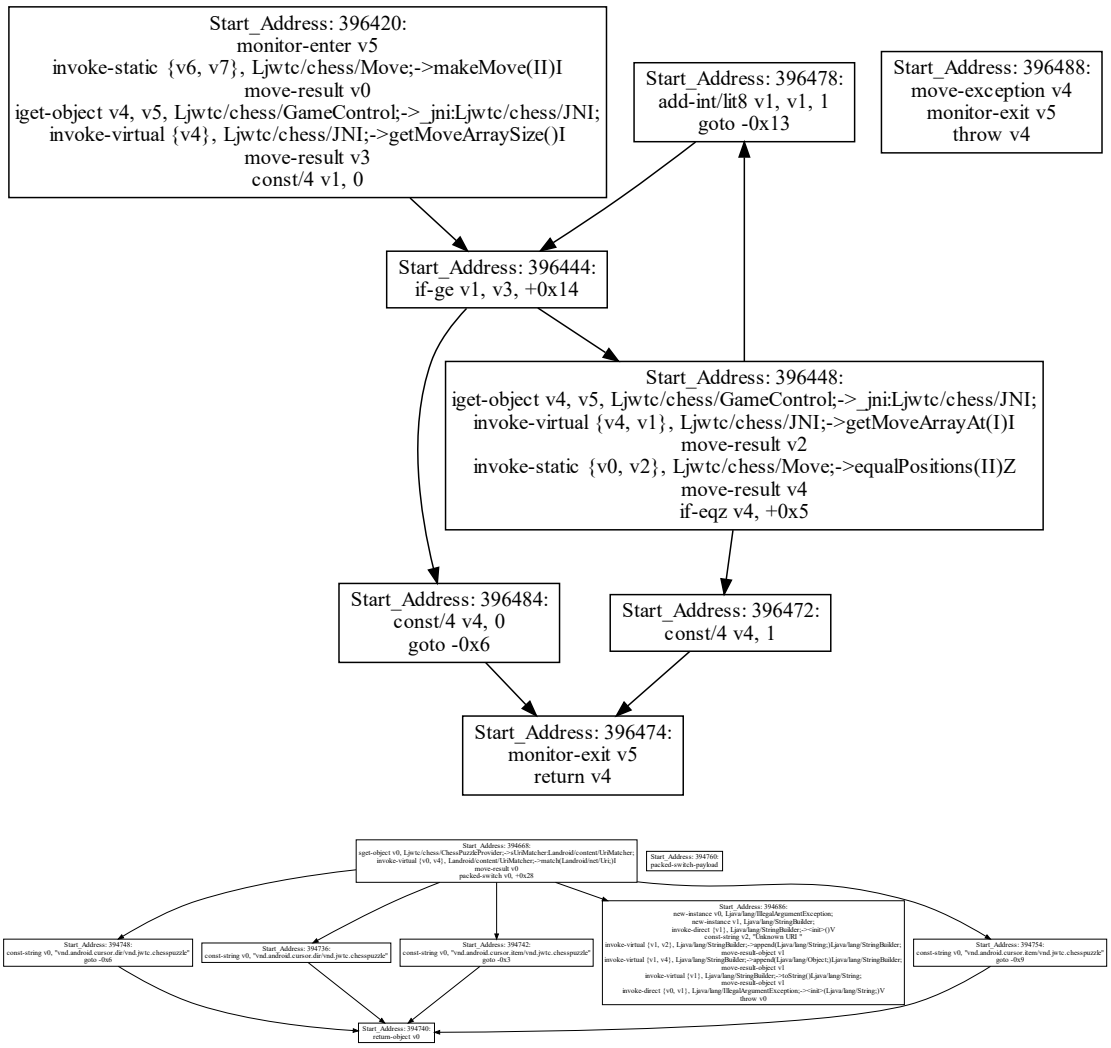
Lab 6. Building CFG

- **Task**

结果展示：



关于完成此任务的简要说明：

createCFG 里面主要分成了三个循环：

第一个是找到 leader instruction 的地址，并把它们记录下来，在下面构造 basic block 的时候用以分界。找 leader instruction 的条件应该有两个，一个是它是跳转指令的下一条指令，一个是跳转到的地址。我用 offsets 向量存放所有 leader instruction（除首条指以外）的地址，然后因为 switch 语句的所有对应位置不是用 PACKED_SWITCH 或者 SPARSE_SWITCH 直接能算的，而是要用 payload，所以 switch_des 向量就是储存所有的目标值。switch_origin 储存 switch 语句地址，用来在 payload 的时候计算进入地址。

```
int switch_origin = 0;
Iterable<? extends Instruction> instructions = cfg.targetMethod.getImplementation().getInstructions();
Vector offsets = new Vector();
Vector switch_des = new Vector();

for (Instruction i : instructions) {
    DexBackedInstruction dbi = (DexBackedInstruction) i;
    if (flp == 0) {
        entry = dbi.instructionStart;
        flp = 1;
    }
    int offset = 0;
    switch (dbi.opcode) {
        case GOTO:
            offset = ((DexBackedInstruction10t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            System.out.println(dbi.getOpcode() + ", offset: " + offset);
            offsets.add(offset);
            break;
        case GOTO_16:
            offset = ((DexBackedInstruction20t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            System.out.println(dbi.getOpcode() + ", offset: " + offset);
            offsets.add(offset);
            break;
        case GOTO_32:
            offset = ((DexBackedInstruction30t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            System.out.println(dbi.getOpcode() + ", offset: " + offset);
```

第二个循环是进行 basicblock 的分割操作，因为我之前没有对于开始的第一部指令进行标记，只记了个地址，所以需要标记判断是不是第一个 basicblock，如果是第一个，把这个 block 赋值给 entyBB，剩下的，如果命令是 Switch 意味着下面无论是什么都要分割（因为 packed_switch 需要后面的参数参与命令地址，但我不知道用什么函数/(ㄒo ㄒ)/~~，我先用这个方法直接替代），如果指令是 leader instruction，那么就把当前模块加到 hashtable 里面，重新加一个新模块，如果不是，那么就在当前 bb 后面添加指令就行。

```
BasicBlock nowblock = new BasicBlock(method, entry);
flp = 0;
for (Instruction i : instructions) {
    DexBackedInstruction dbi = (DexBackedInstruction) i;
    if (dbi.getOpcode() == Opcode.PACKED_SWITCH || dbi.getOpcode() == Opcode.SPARSE_SWITCH) { //正常的代码switch之后不会没有指令了，所以不会出错
        nowblock.addInstruction(i);
        cfg.blocks.add(nowblock);
        nowblock = new BasicBlock(method, startAddress: dbi.instructionStart + dbi.getOpcode().format.size);
    }
    else if (offsets.contains(dbi.instructionStart)) {
        if (flp == 0) {
            cfg.entryBB = nowblock;
            flp = 1;
        }
        if (nowblock.getInstructions() == null){
            nowblock.addInstruction(i);
        }
        else {
            cfg.blocks.add(nowblock);
            nowblock = new BasicBlock(method, dbi.instructionStart);
            nowblock.addInstruction(i);
        }
    }
    else
```

第三段，对于 hashtable 里面的每个 table，块的连接对于每个块的最后一条指令，不同指令不同；如果是跳转指令，它需要连接到跳转的位置；如果是 switch，就跳转到刚才保存的几个位置；如果是 return，throw 等结束函数的指令，就不再往下连；如果是普通指令，就只用往它后面连。

```
for (BasicBlock i : cfg.blocks) {
    Instruction m = i.getInstructions().get(i.getInstructions().size() - 1);
    DexBackedInstruction dbi = (DexBackedInstruction) m;
    int offset = 0;

    offset = dbi.instructionStart + dbi.getOpcode().format.size;
    switch (dbi.opcode) {
        case GOTO:
            offset = ((DexBackedInstruction10t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            linkBlock(cfg,i,offset);
            break;
        case GOTO_16:
            offset = ((DexBackedInstruction20t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            linkBlock(cfg,i,offset);
            break;
        case GOTO_32:
            offset = ((DexBackedInstruction30t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            linkBlock(cfg,i,offset);
            break;
        case IF_EQ:
        case IF_NE:
        case IF_LT:
        case IF_GE:
        case IF_GT:
        case IF_LE:
            linkBlock(cfg,i,offset);
            offset = ((DexBackedInstruction22t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
            linkBlock(cfg,i,offset);
```