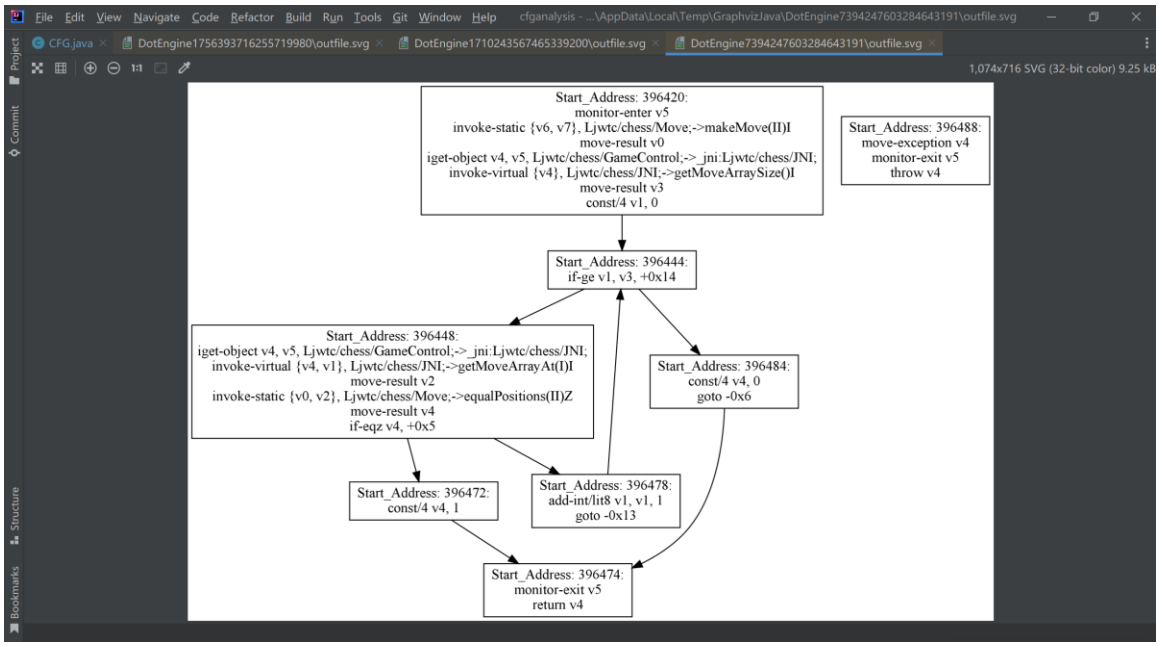
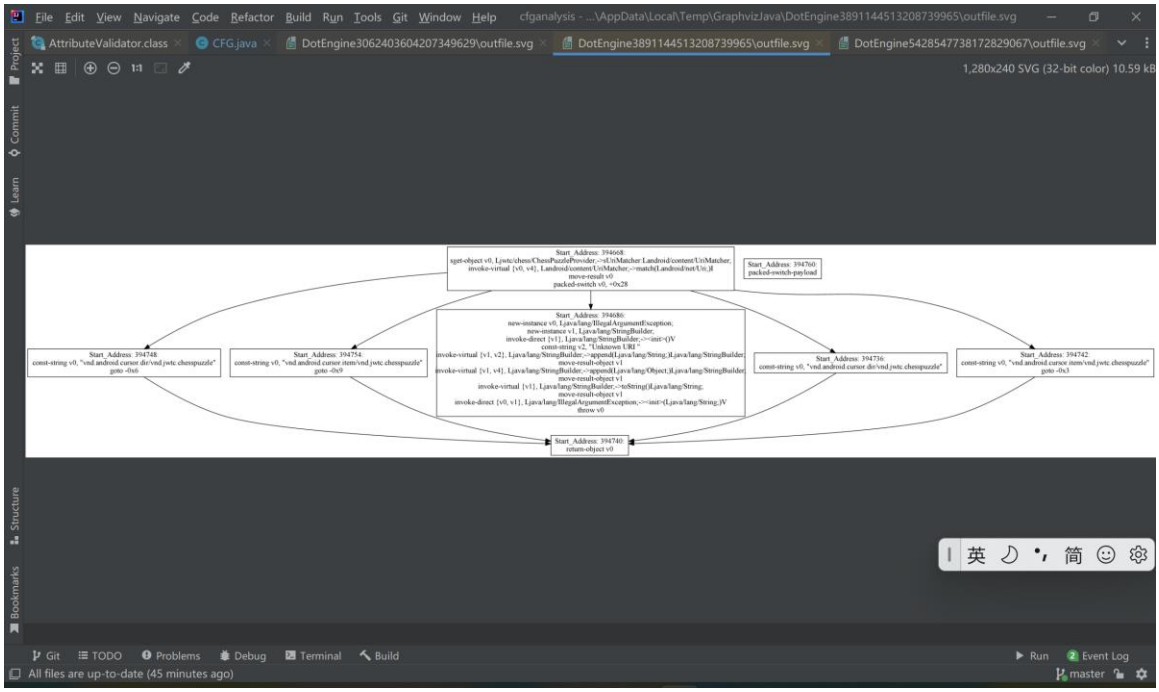
Lab 5. Building CFG

- **Task 1**

(1) Your Answer





(2) Writeup

[Record how you solve this task here.]

注释：本次我除了对 createCFG()进行修改，还在 CFG.class 中增加了一个 addBlock()函数，并在 createCFG 中调用。

1. 因为这次没看 ppt 就直接开始写作业，用的算法和 ppt 的两种算法都有所区别。将二者结合了起来，划分 basic block 和 connect basic block 同时进行——针对每一条指令不仅判断其所属的 basic block 同时针对 goto，if，switch 等语句都进行了 basic block 连接。（因此代码结构都在一个 for 循环里得到解决，但是每一步都需要判断，因此实际上结构不如全部划分基本快再连接来得清晰明了）。
2. 后来因为 delete 函数不能在 basicblock 中实现，因此对代码进行了优化，同时算法改成先识别出 bb，再统一连线的操作。下面讲述更改后的算法：
3. 首先确定 entryBB，再遍历了各个 instruction 之后，确定了第一条语句就是 entryBB 的 startAddress。因此在遍历之前先确定
4. 对于 instruction 确定 basic block 的算法：
   a) 基本思想：根据上一句语句的标志信息来判断下一句语句（本句语句）的 bb：
   b) 根据语句类型，定义了几个局部变量：

```
int switchAddress = 0; //便于记录原来的switch地址
int switchFlag = 0;
int isReturn=0;  //1:上一句是return语句
int offset = 0;  //!0:上一句是GOTO/IF/SWITCH语句
int notNew=0;    //1:对于上一句是GOTO/IF/SWITCH语句的情况，是否创建新的bb
int isThrow=0;
```

   c) 对于 switch-payload 和 exception 语句，throw 和 return 语句，无论当前 basic block 如何，下一个语句都会另起一个 bb。因此直接 new 一个新的 bb，加入指令，加入 cfg 中，同时赋值给 currentBB；

```
if(dbi.opcode == Opcode.SPARSE_SWITCH_PAYLOAD || dbi.opcode==Opcode.PACKED_SWITCH_PAYLOAD
        || dbi.opcode == Opcode.MOVE_EXCEPTION || isThrow==1 || isReturn==1){
    BasicBlock newBB = new BasicBlock(method, dbi.instructionStart);
    newBB.addInstruction(i);
    cfg.blocks.add(newBB);
    currentBB = newBB;
    isReturn=0;
    isThrow=0;
}
```

```
case THROW:
case THROW_VERIFICATION_ERROR:
    isThrow = 1;
    break;

case RETURN_OBJECT:
case RETURN:
case RETURN_VOID:
case RETURN_VOID_BARRIER:
case RETURN_VOID_NO_BARRIER:
case RETURN_WIDE:
    isReturn=1;
    break;
```

d) 对于上一句是 switch/if/goto 语句:因为下一个语句都会分到新的模块，但是 switch/if/goto 语句本身分别也会有 N/2/1 个分支；因为在 switch-case 的情况中该三句语句都要更新 offset，因此将 offset 作为**上一句是 switch/if/goto 语句**的标志。如果 offset！=0,则判断是否需要为 switch/if/goto 语句的下一句语句新建一个语句（根据是否存在以该语句的 instuctionStart 为 startAdress 的 bb，以 notNew 作为标志。

i. 此外，对于每个该类型语句，都要在 case 中 addBlock

```
} else {
    iterator = cfg.blocks.iterator();
    System.out.println("notNew="+notNew);
    while (iterator.hasNext()) {
        BasicBlock BB = iterator.next();
        if (BB.getStartAddress() == dbi.instructionStart) {
            currentBB = BB;
            notNew=1;
            break;
        }
    }
    if(notNew==0){
        BasicBlock newBB = new BasicBlock(method, dbi.instructionStart);
        if(switchFlag==1) {
            currentBB.addSuccessor(newBB);
            switchFlag = 0;
        }
        currentBB = newBB;
        cfg.blocks.add(currentBB);
    }
    currentBB.addInstruction(i);
}
```

e) 其他语句：基本上就是划分在上一个语句中的。Offset=0；无需新建 bb。

```
else if (offset == 0) {
    iterator = cfg.blocks.iterator();
    while (iterator.hasNext()) {
        BasicBlock BB = iterator.next();
        if (BB.getStartAddress() == dbi.instructionStart) {
            currentBB = BB;
            break;
        }
    }
    currentBB.addInstruction(i);
} else {
```

5. 对于每个分支语句（switch/goto/if 指令）之后的 bb 连接的算法：（写在新定义的 addBlock(cfg,currentBB,offset)中：

```
case GOTO:
    offset = ((DexBackedInstruction10t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
    System.out.println(dbi.getOpcode() + ", offset: " + offset);
    addBlock(cfg, currentBB, offset);
    break;
```

a) 对于已经有了分支 bb 的情况，直接接在后面。

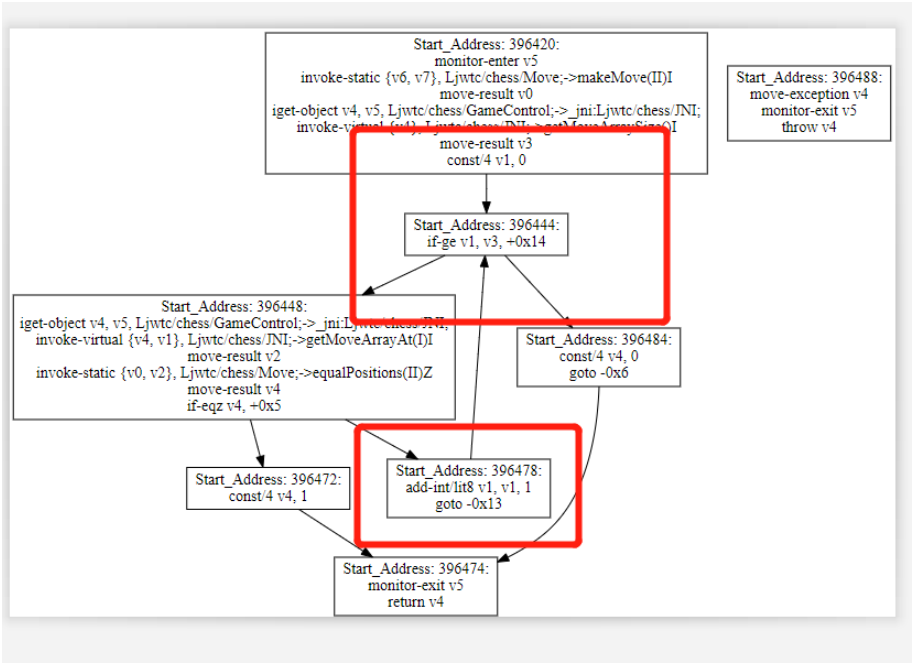
```
private static void addBlock(CFG cfg,BasicBlock currentBB,int offset){
    Iterator<BasicBlock> iterator=cfg.blocks.iterator();
    while (iterator.hasNext()) {
        BasicBlock BB =  iterator.next();
        if(BB.getStartAddress()==offset) {  //已经有了，不新创
              currentBB.addSuccessor(BB);
            return;
        }
    }
```

b) 对于不存在的情况，新建一个 bb——此步骤有两种情况：

i. 一种是所有现有的模块都确定不会无需再进行切割的情况：直接新建：

```
BasicBlock newBB = new BasicBlock(cfg.targetMethod, offset);
cfg.blocks.add(newBB);
```

ii. 一种是后面的模块会再次指向前面的模块中的非头语句；如 478bb 重新指向 444bb，而原来的 444 语句是在 420bb 中的，因此在第一种情况的基础之上需要再次切割，但要确保切割之后的原本的 bb 的后继连接上新生成的 bb。



```
    BasicBlock newBB = new BasicBlock(cfg.targetMethod, offset);
    cfg.blocks.add(newBB);
      currentBB.addSuccessor(newBB);
    iterator=cfg.blocks.iterator();
    while (iterator.hasNext()) {
        int tag=0;
        BasicBlock nextBB =  iterator.next();
        Iterator<Instruction> iteratorInstr=nextBB.getInstructions().iterator();
        while (iteratorInstr.hasNext()) {
            Instruction nextInstr=iteratorInstr.next();
            DexBackedInstruction dbi = (DexBackedInstruction)nextInstr;
            if(dbi.instructionStart==offset) {
                tag = 1;
            }
            if(tag==1 && dbi.instructionStart>=offset){
                newBB.addInstruction(nextInstr);
                nextBB.getInstructions().remove(nextInstr);
                iteratorInstr=nextBB.getInstructions().iterator();
            }
        }
        if(tag==1){
            nextBB.addSuccessor(newBB);
            return;
        }

    }
```

6. 对于 switch-payload 语句：
   a) 先确定 table 中每个分支的 startAddress 语句：
   b) 防止出现重新分割的情况，每个 offset 都要新建一个 bb（用 addBlock()）。

```
for (SwitchElement s : switchElements) {
    /*
     * !!! Important:
     * According to sparse-switch-payload Format :
     * The targets are relative to the address of the switch opcode,
     */

    offset = s.getOffset() * 2 + switchAddress;
    System.out.println(dbi.getOpcode() + ", offset: " + offset);
    addBlock(cfg, currentBB, offset);
    }
break;
}
```

7. 连接所有的 bb：
   a) 遍历 cfg 中的每个模块。从每个模块中遍历每条指令。如果对应于 goto/if 指令，就直接用 linkBlock 连接。

```
for (BasicBlock preBB : cfg.blocks) {
    ArrayList<Instruction> Instuctions = preBB.getInstructions();
    for (Instruction instruction : Instuctions) {
        DexBackedInstruction dbi = (DexBackedInstruction) instruction;
        switch (dbi.opcode) {
            case GOTO:
                offset = ((DexBackedInstruction10t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
                System.out.println(dbi.getOpcode() + ", offset: " + offset);
                linkBlock(cfg, preBB, offset);
                break;
            case GOTO_16:
                offset = ((DexBackedInstruction20t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
                System.out.println(dbi.getOpcode() + ", offset: " + offset);
                linkBlock(cfg, preBB, offset);
                break;
            case GOTO_32:
                offset = ((DexBackedInstruction30t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
```

   b) 对于 switch 指令，再次遍历 cfg.blocks，来寻找 switch-paylod 模，拿出其中的指令（亦可以直接遍历指令集，找到对应的 pay-load 语句），遍历得到其 offset，用 linkBlock 连接。

```
case PACKED_SWITCH:
case SPARSE_SWITCH:
    switchAddress = dbi.instructionStart;
    offset = ((DexBackedInstruction31t) dbi).getCodeOffset() * 2 + dbi.instructionStart;
    System.out.println(dbi.getOpcode() + ", switch payload offset: " + offset);
    iterator=cfg.blocks.iterator();
    while (iterator.hasNext()) {
        BasicBlock payloadBB = iterator.next();
        System.out.println(payloadBB.getStartAddress());
        if(payloadBB.getStartAddress()==offset){
            Iterator plIterator=payloadBB.getInstructions().iterator();
            Instruction plInstr=null;
            while (plIterator.hasNext()) {
                plInstr = (org.jf.dexlib2.iface.instruction.Instruction) plIterator.next();
                break;
            }
            DexBackedInstruction DBI=(DexBackedInstruction)plInstr;
            List<? extends SwitchElement> switchElements = null;
            if (DBI instanceof DexBackedPackedSwitchPayload)
                switchElements = ((DexBackedPackedSwitchPayload) DBI).getSwitchElements();
            else
                switchElements = ((DexBackedSparseSwitchPayload) DBI).getSwitchElements();

            for (SwitchElement s : switchElements) {

                offset = s.getOffset() * 2 + switchAddress;
                System.out.println(dbi.getOpcode() + ", offset: " + offset);
                linkBlock(cfg,preBB,offset);
```