东南大学国家示范性软件学院
College of Software Engineering
Southeast University

软件测试基础与实践

实验报告

实验名称：黑盒测试实验二

实验地点：计算机楼 268

实验日期：2018.11.22

学生姓名：杨昱昊

学生学号：71116216

东南大学 软件学院 制

一、实验目的

(1)能根据待测软件的特点，选择合适的方法对软件进行黑盒测试（功能测试）；

(2)了解随机测试，巩固白盒测试和黑河测试方法。

(3)了解 JUnit 测试开发框架及其应用

(4)能对一些特定的程序进行蜕变测试

二、实验内容

（一）题目 1：随机测试 VS 黑盒测试 VS 白盒测试

1. 题目内容

在游戏引擎开发中,检测物体碰撞是一项重要的基础功能,比如 DOTA 和王者荣耀等游戏中的各种华丽大招的伤害波及范围计算等。为简单起见,我们这里只考虑二维平面空间的情况,并用 RectManager 程序判断平面上任意两矩形的相交关系(A:不相交,B:相交:B1:相交为一个区域,B12:包含,B13:完全重合,B2:交点为 1 个点,B3:交点为 1 条线段),如果相交,则同时给出相交部分的面积。这里的二维平面限定为 iphone4 屏幕(640*960 分辨率),且所有矩形的边都与坐标轴平行。计算机图形学中,通常用左上角和右下角的坐标来表示一个矩形。

（1）请编写一简单程序,随机生成两个矩形的数据作为测试用例,请用这些测试用例对 RectManager 进行测试。

（2）请用黑盒测试方法，设计相应的测试用例开测试程序

（3）请分析 RectManager 的实现源代码，利用基本路径测试方法对程序进行白盒测试。只要求针对 solve()方法进行测试（只给出基本路径，不用设计具体测试用例）

（4）在上述实验的基础上分析三中测试方法发现缺陷的能力上有何差别

实验过程注意事项:

a. 面积计算问题:计算机屏幕上，1 个像素点也是有面积的,面积为 1 ；1 条线段也是有面积的，1 × 2 的线段的面积为 2 。这点和我们学数学上抽象的面积不同,因为线段或点都实实在在占用了设备的面积。

b. 通过实验理解随机测试的不足:即便生成很多测试用例,依然很难覆盖各种需要测试的情形,对于一些特殊的测试情形很难覆盖。

c. 随机的测试用例生成中,随机生成的矩形的区域应该比显示区域稍微大一点。

d. 黑盒测试的思路较多,但等价类划分和边界值分析还是基础。

2. 题目解答

（1）编写程序如下

```
package weeke;
import java.util.Random;
public class RectManagerTester{
    /*
    对RectManager 进行随机测试，实现目标：
        随机成城测试用例
        对6 种相交情况进行统计，同时记录不合法输入情况
     */
    public static void main(String[] args){
        // 实例化两个四边形
        Rect testA = new Rect();
        Rect testB = new Rect();
        // 6 中相交情况的数量记录
        int case0Num = 0, case1Num = 0, case2Num = 0,
                case3Num = 0, case4Num = 0, case5Num = 0;
        // 随机测试中非法用例的数量
        int errorInputNum = 0;
        // 实例化一个测试者
        RectManager manager = new RectManager();
        int usecaseNum = 100000;
        for (int i = 0; i < usecaseNum; i++) {
```

```
            // 随机生成参数
            testA.left = new Random().nextInt(660)-10;
            testA.right = new Random().nextInt(660)-10;
            testA.top = new Random().nextInt(980)-10;
            testA.bottom = new Random().nextInt(980)-10;

            testB.left = new Random().nextInt(660)-10;
            testB.right = new Random().nextInt(660)-10;
            testB.top = new Random().nextInt(980)-10;
            testB.bottom = new Random().nextInt(980)-10;

            //检查异常输入
            if (!(testA.left>=0 && testA.right<640)|| !(testA.top>=0 && testA.bottom<96
0)
                    ||!(testA.right>=testA.left) || !(testA.bottom>=testA.top)){
                errorInputNum++;
            }else if (!(testB.left>=0 && testB.right<640)|| !(testB.top>=0 &&   testB.b
ottom<960)||!(testB.right>=testB.left)|| !(testB.bottom>=testB.top)){
                errorInputNum++;
            }else if (testA.left<0||testA.right>639||testA.top<0||testA.bottom>950
            ||testB.left<0||testB.right>639||testB.top<0||testB.bottom>950){
                errorInputNum++;
            }
            // 统计随机测试结果
            manager.solve(testA,testB);
            switch (manager.nFlag){
                case 0:
                    case0Num++;
                    break;
                case 1:
                    case1Num++;
                    break;
                case 2:
                    case2Num++;
                    break;
                case 3:
                    case3Num++;
                    break;
                case 4:
                    case4Num++;
                    break;
                case 5:
                    case5Num++;
                    break;
```

```
            }
        }
        System.out.print("测试用例共: "+usecaseNum+" 个\n"
            +"矩形不相交: "+case0Num+" 个\n"
            +"矩形相交于一个区域: "+case1Num+" 个\n"
            +"矩形相交于一个区域且为包含关系: "+case2Num+" 个\n"
            +"矩形相交于一个区域且正好重合: "+case3Num+" 个\n"
            +"矩形相交于一个区域且交点为 1 个点: "+case4Num+" 个\n"
            +"矩形相交于一个区域且交点为一条线段: "+case5Num+" 个\n"
            +"非法用例: "+errorInputNum+" 个\n"
        );
    }
}
```

运行结果截图如下：

```
/usr/lib/jvm/java-10-openjdk/bin/java -javaagent:/opt/intellij-idea-u
测试用例共: 100000 个
矩形不相交: 97138 个
矩形相交于一个区域: 2470 个
矩形相交于一个区域且为包含关系: 372 个
矩形相交于一个区域且正好重合: 0 个
矩形相交于一个区域且交点为1个点: 0 个
矩形相交于一个区域且交点为一条线段: 20 个
非法用例: 94952 个

Process finished with exit code 0
```

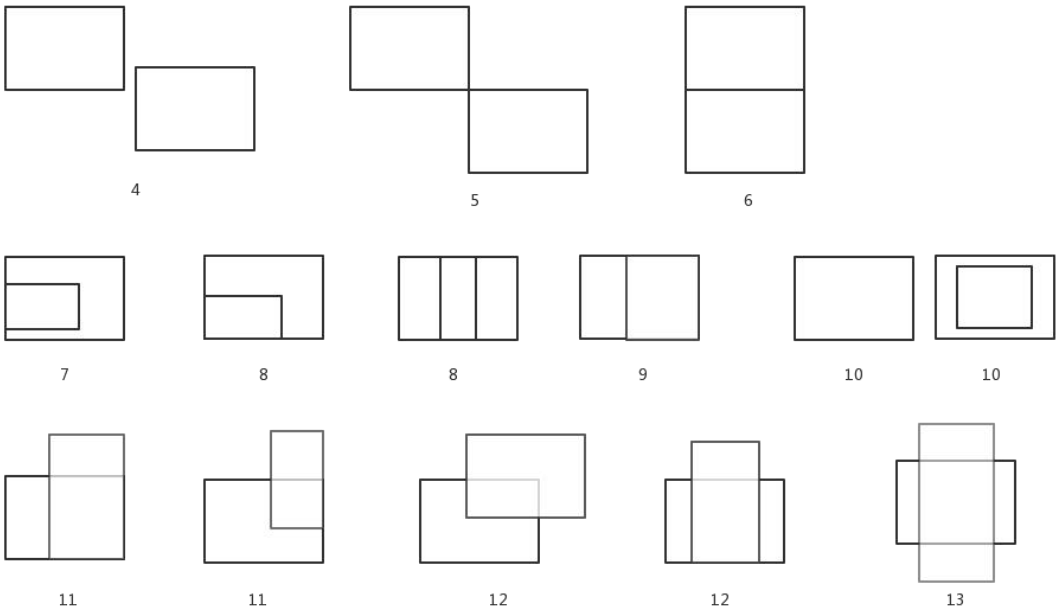
（2）黑盒测试

划分等价类：

编号	类型	等价类
1	无效等价类	输入超出屏幕边界
2	无效等价类	输入 top>bottom||left>right
3	无效等价类	输入参数为非整数和字符
4	有效等价类	不重叠

5	有效等价类	重叠且共有一点
6	有效等价类	重叠且共有一边
7	有效等价类	包含且有一边重叠
8	有效等价类	包含且有两边重叠
9	有效等价类	包含且有三边重叠
10	有效等价类	完全重叠
11	有效等价类	重叠且边相交于一点
12	有效等价类	重叠且边相交于两点
13	有效等价类	重叠且边相交于四点

为直观描述，作图如下：



设计测试用例如下

编号	测试用例		等价类	期望输出	实际输出
	A	B			

1	-1,1,3,3	2,2,5,5	1	Input error in Rectangle A	Input error in Rectangle A
2	1,1,3,3	-1,2,4,4	1	Input error in Rectangle B	Input error in Rectangle B
3	630,100,650,200	620,100,630,200	1	Input error in Rectangle A	Input error in Rectangle A
4	3,2,2,3	1,1,2,2	2	Input error in Rectangle A	Input error in Rectangle A
5	2,2,3,1	1,1,2,2	2	Input error in Rectangle A	Input error in Rectangle A
6	a,b,c,d	1,1,2,2	3	Input error in Rectangle A	java.util.InputMismatchException
7	0.1,1,2,2	1,1,2,2	3	Input error in Rectangle A	java.util.InputMismatchException
8	1,1,2,2	3,3,4,4	4	矩形不相交！	矩形不相交！
9	2,2,3,3	2,5,3,7	4	矩形不相交！	矩形不相交！
10	3,3,4,4	1,1,2,2	4	矩形不相交！	矩形不相交！
11	1,1,2,2	2,2,3,3	5	矩形相交于一个区域且交点仅为 1 个点！相交面积：1.0	矩形相交于一个区域且交点仅为 1 个点！相交面积：1.0
12	2,2,3,3	3,1,4,2	5	矩形相交于一个区域且交点仅为 1 个点！相交面积：1.0	矩形相交于一个区域且交点仅为 1 个点！相交面积：1.0
13	2,2,3,3	1,1,2,2	5	矩形相交于一个区域且交点仅为 1 个点！相交面积：1.0	矩形相交于一个区域且交点仅为 1 个点！相交面积：1.0
14	1,1,3,3	1,3,3,4	6	矩形相交于一个区域且交点为 1 条线段！	矩形相交于一个区域且交点为 1 条线段！

				相交面积：3.0	相交面积：3.0
15	1,3,3,4	1,1,3,3	6	矩形相交于一个区域且交点为 1 条线段！相交面积：3.0	矩形相交于一个区域且交点为 1 条线段！相交面积：3.0
16	1,1,3,3	3,1,4,2	6	矩形相交于一个区域且交点为一条线段！相交面积 3.0	矩形相交于一个区域且交点为一条线段！相交面积 3.0
17	3,1,4,2	1,1,3,3	6	矩形相交于一个区域且交点为一条线段！相交面积 3.0	矩形相交于一个区域且交点为一条线段！相交面积 3.0
18	1,1,4,4	3,2,4,3	7	矩形相交于一个区域且为包含关系！相交面积：4.0	矩形相交于一个区域且为包含关系！相交面积：4.0
19	1,1,4,4	1,2,2,3	7	矩形相交于一个区域且为包含关系！相交面积：4.0	矩形相交于一个区域且为包含关系！相交面积：4.0
20	1,1,3,3	1,1,4,4	8	矩形相交于一个区域且为包含关系！相交面积：9.0	矩形相交于一个区域且为包含关系！相交面积：9.0
21	2,2,4,4	1,1,4,4	8	矩形相交于一个区域且为包含关系！	矩形相交于一个区域且为包含关系！

				相交面积：9.0	相交面积：9.0
22	1,1,4,4,	2,2,4,4	8	矩形相交于一个区域且为包含关系！ 相交面积：9.0	矩形相交于一个区域且为包含关系！ 相交面积：9.0
23	1,2,3,3	1,1,3,3	9	矩形相交于一个区域且为包含关系！ 相交面积：6.0	矩形相交于一个区域且为包含关系！ 相交面积：6.0
24	1,1,3,3	2,1,3,3	9	矩形相交于一个区域且为包含关系！ 相交面积：6.0	矩形相交于一个区域且为包含关系！ 相交面积：6.0
25	1,1,3,3	1,1,3,3	10	矩形相交于一个区域且正好重合！ 相交面积：9.0	矩形相交于一个区域且正好重合！ 相交面积：9.0
26	1,1,4,4	1,2,5,4	11	矩形相交于一个区域！ 相交面积：12.0	矩形相交于一个区域！ 相交面积：12.0
27	1,1,4,4	2,2,5,5	12	矩形相交于一个区域！ 相交面积：9.0	矩形相交于一个区域！ 相交面积：9.0
28	1,1,4,4	2,0,3,5	13	矩形相交于一个区域！ 相交面积：8.0	矩形相交于一个区域！ 相交面积：8.0

边界条件如下：

矩形 A：

编号	left	top	right	bottom
1	0	0-959	0-639	0-959
2	639	0-959	0-639	0-959
3	0-639	0	0-639	0-959
4	0-639	959	0-639	0-959
5	0-639	0-959	0	0-959
6	0-639	0-959	639	0-959
7	0-639	0-959	0-639	0
8	0-639	0-959	0-639	959

测试用例：

编号	测试用例		边 界 条件	期望输出	实际输出
	A	B			
1	0 1 3 3	1 1 4 4	1	矩形相交于一个区域！ 相交面积：9.0	矩形相交于一个区域！ 相交面积：9.0
2	-1 1 3 3	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A
3	1 1 3 3	1 1 4 4		矩形相交于一个区域且为包含关系！ 相交面积：9.0	矩形相交于一个区域且为包含关系！ 相交面积：9.0
4	639 1 639 3	1 1 4 4	2	矩形不相交！ 相交面积：0.0	矩形不相交！ 相交面积：0.0
5	638 1 639 3	1 1 4 4		矩形不相交！ 相交面积：0.0	矩形不相交！ 相交面积：0.0
6	640 1 639 3	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A
7	1 0 3 3	1 1 4 4	3	矩形相交于一个区域！ 相交面积：9.0	矩形相交于一个区域！ 相交面积：9.0
8	1 -1 3 3	1 1 4 4		Input error in Recta	Input error in Recta

				ngle A	ngle A
9	1 1 3 3	1 1 4 4		矩形相交于一个区域且 为包含关系！ 相交面积：9.0	矩形相交于一个区域且 为包含关系！ 相交面积：9.0
10	1 959 3 959	1 1 4 4	4	矩形不相交！ 相交面积：0.0	矩形不相交！ 相交面积：0.0
11	1 958 3 959	1 1 4 4		矩形不相交！ 相交面积：0.0	矩形不相交！ 相交面积：0.0
12	1 961 3 959	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A
13	0 1 0 3	1 1 4 4	5	矩形不相交！ 相交面积：0.0	矩形不相交！ 相交面积：0.0
14	0 1 -1 3	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A
15	0 1 1 3	1 1 4 4		矩形相交于一个区域且 交点为 1 条线段！ 相交面积：3.0	矩形相交于一个区域且 交点为 1 条线段！ 相交面积：3.0
16	1 1 639 3	1 1 4 4	6	矩形相交于一个区域！ 相交面积：12.0	矩形相交于一个区域！ 相交面积：12.0
17	1 1 638 3	1 1 4 4		矩形相交于一个区域！ 相交面积：12.0	矩形相交于一个区域！ 相交面积：12.0
18	1 1 640 3	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A
19	1 0 3 0	1 1 4 4	7	矩形不相交！ 相交面积：0.0	矩形不相交！ 相交面积：0.0

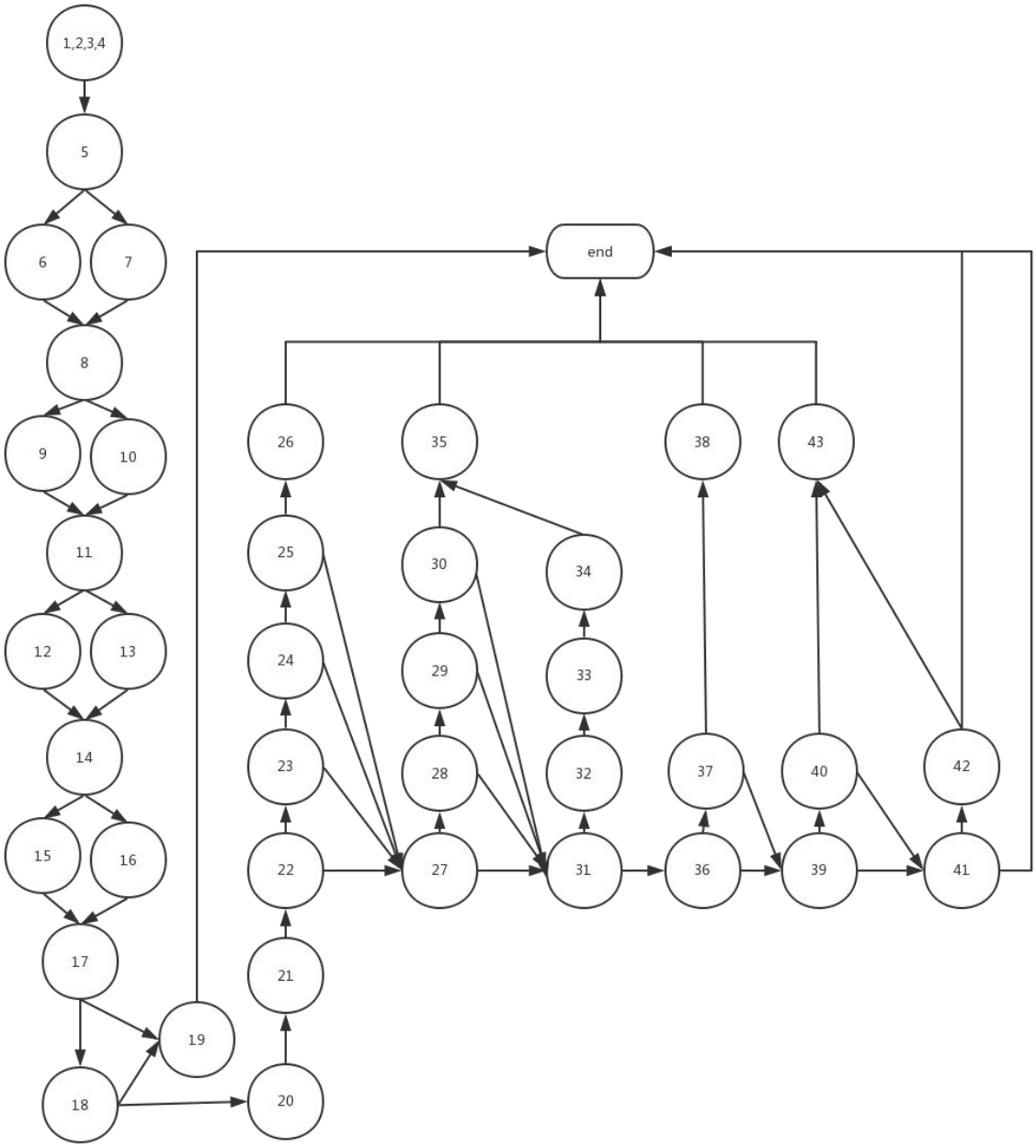
20	1 0 3 1	1 1 4 4		矩形相交于一个区域且 交点为 1 条线段！ 相交面积：3.0	矩形相交于一个区域且 交点为 1 条线段！ 相交面积：3.0
21	1 0 3 -1	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A
22	1 1 3 959	1 1 4 4	8	矩形相交于一个区域！ 相交面积：12.0	矩形相交于一个区域！ 相交面积：12.0
23	1 1 3 958	1 1 4 4		矩形相交于一个区域！ 相交面积：12.0	矩形相交于一个区域！ 相交面积：12.0
24	1 1 3 960	1 1 4 4		Input error in Rectangle A	Input error in Rectangle A

（3）基本路径测试

路径标号：

```
    public void solve(Rect A, Rect B){
1       int nMaxLeft = 0;
2       int nMaxTop = 0;
3       int nMinRight = 0;
4       int nMinBottom = 0;
5       if (A.left >= B.left) {
6          nMaxLeft = A.left;
        }
        else {
7          nMaxLeft = B.left;
        }
8       if (A.top >= B.top) {
9          nMaxTop = A.top;
        }
        else {
10         nMaxTop = B.top;
        }
11      if (A.right <= B.right) {
12         nMinRight = A.right;
        }
        else {
13         nMinRight = B.right;
        }
14      if (A.bottom <= B.bottom) {
15         nMinBottom = A.bottom;
        }
        else {
16         nMinBottom = B.bottom;
        }
17      if ((nMaxLeft > nMinRight) ||
18         (nMaxTop > nMinBottom)) {
19         nFlag=0;
        }
        else {
20         nFlag = 1;
21         area = (nMinRight - nMaxLeft   + 1 ) * (nMinBottom -
22  nMaxTop + 1);
23         if ((B.left==A.left) &&
24            (B.right==A.right) &&
25            (B.top==A.top) &&
26            (B.bottom==A.bottom)){
               nFlag = 3;
           }
27         else if (((nMaxLeft==A.left)
28                && (nMinRight==A.right)
29                && (nMaxTop==A.top)
30                && (nMinBottom==A.bottom))
31          ||((nMaxLeft==B.left)
32                && (nMinRight==B.right)
33                && (nMaxTop==B.top)
34                && (nMinBottom==B.bottom))){
35            nFlag = 2;
           }
36         else if ((nMaxLeft==nMinRight)
37                && (nMaxTop == nMinBottom)){
38            nFlag = 4;
           }
39         else if (((nMaxLeft==nMinRight)
40                && (nMaxTop < nMinBottom))
41             || ((nMaxLeft<nMinRight)
42                && (nMaxTop == nMinBottom))){
43            nFlag = 5;
           }
        }
    }
```

程序流图：



基本路径：

1-2-3-4-5-6-8-9-11-12-14-15-17-19-end
1-2-3-4-5-7-8-9-11-12-14-15-17-19-end
1-2-3-4-5-6-8-10-11-12-14-15-17-19-end
1-2-3-4-5-6-8-9-11-13-14-15-17-19-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-19-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-24-25-26-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-29-30-35-end

1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-27-28-29-30-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-24-27-28-29-30-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-23-24-25-27-28-29-30-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-32-33-34-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-31-32-33-34-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-29-31-32-33-34-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-28-29-30-31-32-33-34-35-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-37-38-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-43-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-37-39-40-43-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-43-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-41-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-41-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-41-42-end
1-2-3-4-5-6-8-9-11-12-14-15-17-18-20-21-22-27-31-36-39-40-41-42-43-end

（4）三种测试方法对比

随机测试能够一次性产生大量的测试用例，比如我使用了 100000 个测试用例，生成测试用例的过程非常快速。但是由于随机生成测试用例是不可控的，所以有很多情况是测试不到的，比如在我测试的过程中，正好重合和恰好有一个点重叠这两种情况并没有产生。同时还产生了大量的非法用例。

黑盒测试能够有针对性的对可能出现的软件缺陷进行测试，等价类划分可以减少测试用例数目，而边界值分析可以迅速定位到容易出现 bug 的地方。同时黑盒测试不注重代码的实现过程，所以发现了缺陷后并不能做到精确的定位修改。

白盒测试面向代码，这次实验中使用基本路径测试，通过这样的方法能够对合法输入情况下的各种情况达到很好的测试效果。同时白盒测试方法也是三种测试方法中成本最高的，需要梳理程序代码，理清程序执行的具体过程。

综上：随机测试速度快，同时效率相对较低；黑盒测试相对复杂，容易发现缺陷，同时无法定位具体错误位置：白盒测试最繁琐，可以精确定位错误。

（二）题目 2：蜕变测试问题

1. 题目内容

Sin.exe(该执行程序见 FTP 服务器)是一个用某种数值计算方法求解数学函数 f(x)=sin(x)的程序。请设计测试用例,实现对该程序的黑盒测试。

要求:

(1) 给出每个测试用例设计的理由;

(2) 这个实验中展示的测试用例输出值无法预测的情形还可能出现在哪些实际应用中。

提示:

a) 查找蜕变测试资料,了解测试 Oracle 问题和蜕变测试简单原理;

b) 理解蜕变关系,并知道某些特定测试问题的蜕变关系。

c) 测试过程中,应在 cmd 命令行终端环境下执行,避免程序闪退看不到结果。

2. 题目解答

Oracle 问题导致测试人员只能对一些可以预期结果的特殊测试用例进行测试，而不能进行完整的测试。

蜕变测试就是一种依据北侧软件的领域知识和软件的实现方法建立蜕变关系，利用蜕变关系从源测试用例生成新的测试用例并进行黑盒测试的一种特殊的黑盒测试方法。

对 sin.exe 的测试借助下面的蜕变关系：

sin(x)=sin(x+2pi)
sin(x)=-sin(x+pi)
sin(x)=-sin(-x)
sin(x)=sin(-x+pi)
sin(x)=-sin(-x+2pi)
sin(x)=sin(x+2pi)
sin2(x)+sin2(pi/2-x)=1

设计测试用例如下：

编号	原测试用例	新测试用例	预期结果	实际结果	蜕变关系
1	30		0.5	0.5	
2		210	-0.5	-0.5	sin(x)=-sin(x+pi)
3		390	0.5	0.5	sin(x)=sin(x+2pi)
4		150	0.5	0.5	sin(x)=sin(-x+pi)
5		-30	-0.5	-0.523599	sin(x)=-sin(-x)
6		-330	0.5	-5.75959	sin(x)=-sin(-x+2pi)

三、实验体会

(1)经过一系列的实验,请谈谈你所感受的软件测试中存在哪些挑战性的难题。

黑盒测试中划分等价类和边界值分析是最重要的两项技能，如果没有划分良好的等价类或者边界条件不明确就会导致冗余的测试用例。白盒测试的难点在于选择不同的覆盖标准来对代码进行覆盖，同时要求分析代码的执行顺序逻辑。在进行测试的时候，由于两种测试方法的特点不同，如何结合两种测试方法也是难点之一。

(2)随机测试中,对很多同学的随机数生成范围来说,矩形不相交的统计概率不是 0.5 ,你的实验是否存在这种现象?尝试用概率知识对结果进行分析一下。

在我的实验过程中，矩形不相交的概率远高于相交概率。