算法设计：

1. 预处理：
    a. 在算法运行前使用SOOT提供的优化选项，从Java源代码生成Jimple中间代码。这个过程中会消除一部分不可能被执行的控制流，从而提高了算法的精确性并降低了运行时间。
    b. 使用VASCO[1][2]框架提供的组件产生一个更加精确的控制流图来指导数据流分析的分析过程。
2. 主要算法：
   为了达到更高的精确度，我们实现了多种不同的指针分析算法，并在算法运行结束后对不同的算法产生的结果取交集。由于各个算法都是安全的，这样的处理不会产生不安全的结果。同时不同的算法在不同的数据集上有不同的性能表现，可以通过不同算法的运行来迅速获得一个可以接受的解并逐渐提升其精确度。
    a. 以论文[1]中的算法为基础，利用其提供的VASCO过程间数据流分析框架实现了Anderson指针分析算法。
        i. 上下文敏感性：VASCO框架提供了一定的上下文敏感性，并且提供了较SOOT更为精确的控制流图。
        ii. 域敏感性：我们实现了精确的域敏感分析，我们将每个内存位置展开为内存位置，及其所对应的对象中包含的域的引用。对于数组，我们将其展开为数组本身及其内容，数组的所有元素看作是同一对象，实现了性能和精确度的平衡。
        iii. 流敏感性：同一般的数据流分析相同。
        iv. 性能：我们的算法允许域在被访问到的时候才会创建，避免产生大量的无用的域，并且在过程间分析的时候只传递实际需要的集合，降低了内存消耗。
    b. 根据 CFL Reachability 算法实现了域精确的指针分析。
        i. 上下文敏感性：用克隆的方法展开了两层。
        ii. 域敏感性：使用 CFL Reachability 达到了精确。
        iii. 流敏感性：流不敏感。
    c. 根据 CFL Reachability 实现了上下文精确的指针分析。
        i. 上下文敏感性：使用 CFL Reachability 达到了精确。
        ii. 域敏感性：尝试通过展开一层的方法来达到域敏感。但是课上提到的域敏感算法并不正确，这儿改成了所有约束都是等价关系。因此相当程度上牺牲了精度。
        iii. 流敏感性：流不敏感。
        iv. 性能：因为关系都是等价关系，因此最终可达性图中的边数非常多。算法中使用了等于关系来代替原有的子集关系，因此采用了类似Steensgaard算法的方法，用并查集来加速求解。

测试数据集生成：

对于各种不同的Java特性的支持，我们采用手工编写的较小的测试样例进行测试。对于整个分析的性能，鲁棒性及可扩展性的测试，我们实现了一个随机化的测试框架，框架可以根据一些给定的参数生成样例代码及其真值，然后进行批量测试。

随机测试框架产生的样例基于一个有范围的整数输入，并且会在输入不同的时候执行不同的控制流。通过穷举参数空间即可得到样例所有的真值。在测试样例中会有循环，递归和

分支等多种控制流及大量的赋值、函数调用等操作，可以覆盖最常见的Java程序语法，并且允许产生任意大的程序以满足测试需求。

测试结果

在我们生成的45组测试数据上，我们的几个算法总的结果如下：

算法	上下文不敏感CFL	复制两层的CFL	Anderson算法	三种算法的交集	真实值
结果中变量数总和	2988	2960	2609	2602	2104
百分比(%)	142.0	140.7	124.0	123.7	100

可以看到，由于Anderson是流敏感的，它相比CFL有比较大的提升。而在流不敏感的情况下，在我们产生的数据集上复制两层对性能的提升不大。

运行方法：

（注意须先将类编译成class文件）

```
java -jar pta.jar [classPath] [className]
```

示例：`java -jar pta.jar code test.Hello`

参考文献：

1. Padhye R, Khedker U P. Interprocedural Data Flow Analysis in Soot using Value Contexts[J]. 2013.
2. VASCO. https://github.com/rohanpadhye/vasco. 2018-11-27