

# 软件测试与质量保证

## 4.1 覆盖率测试

张宇霞 副研究员

# 目录

CONTENTS

01

白盒测试

02

语句覆盖

03

判定覆盖

04

条件覆盖

05

条件/判定覆盖

06

条件组合覆盖

07

路径覆盖

08

小结



# 目录

CONTENTS

01

白盒测试

02

语句覆盖

03

判定覆盖

04

条件覆盖

05

条件/判定覆盖

06

条件组合覆盖

07

路径覆盖

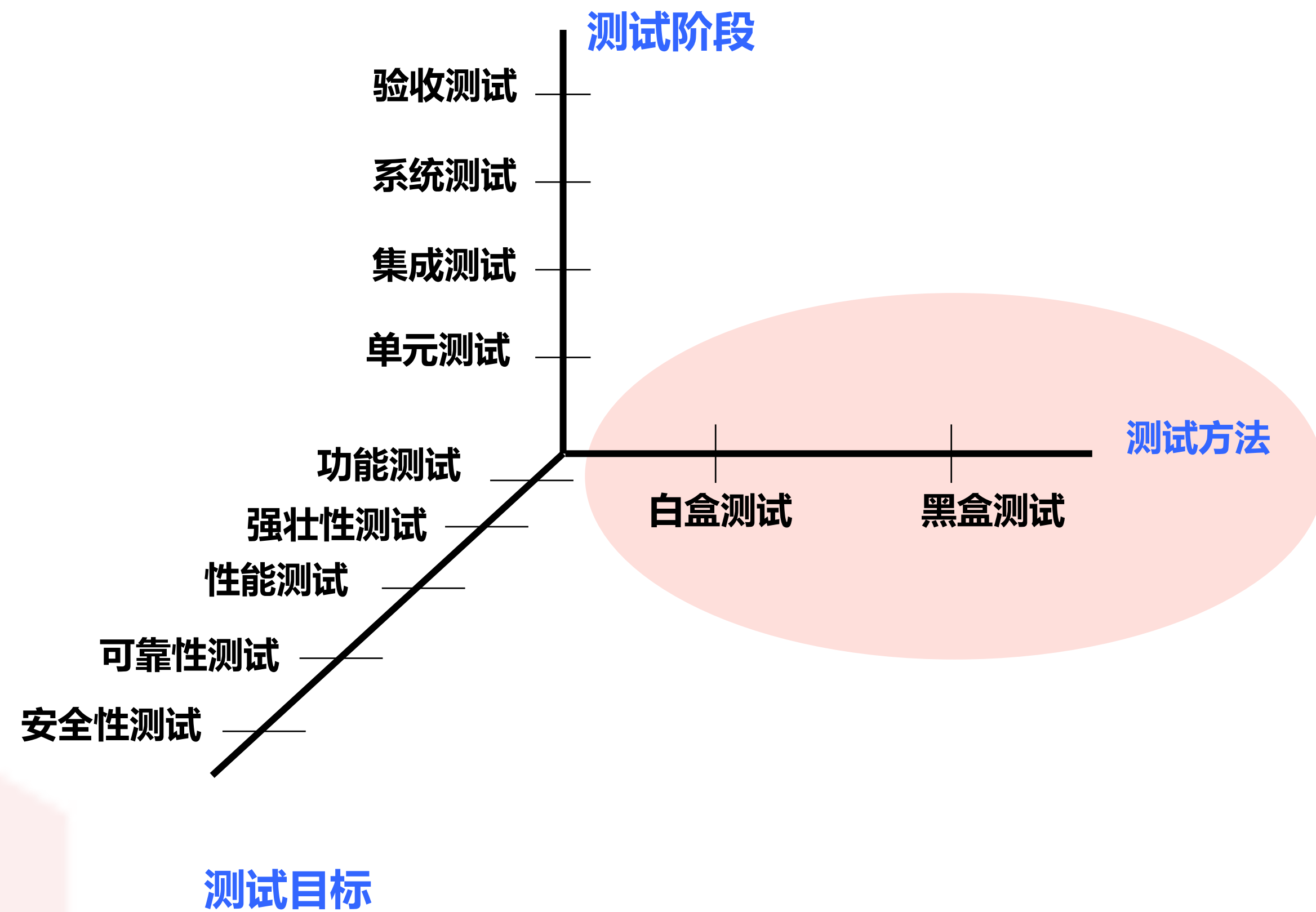
08

小结



01

# 白盒测试



多维度分类:

测试方法

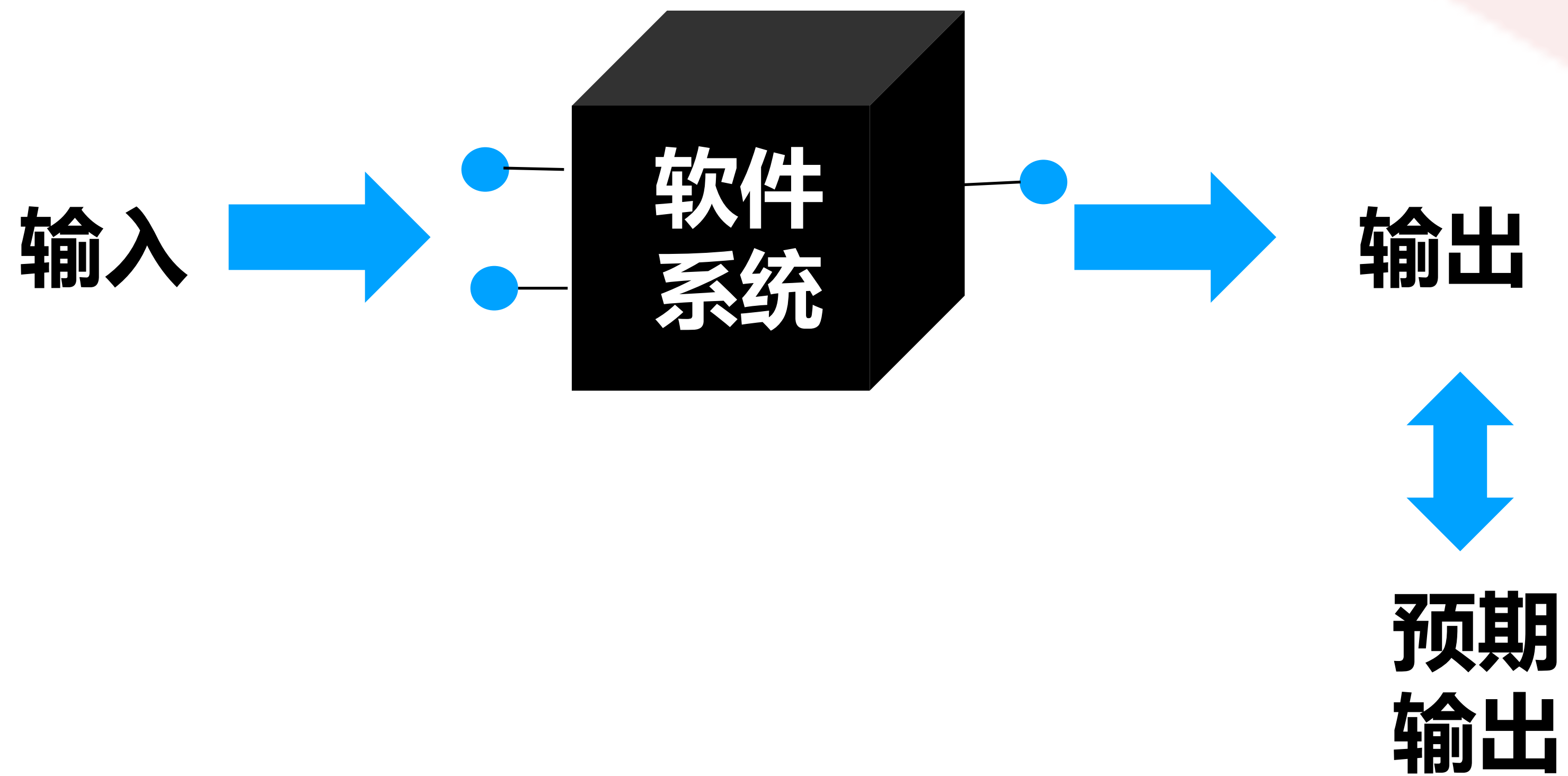
测试阶段

测试目标



01

# 白盒测试

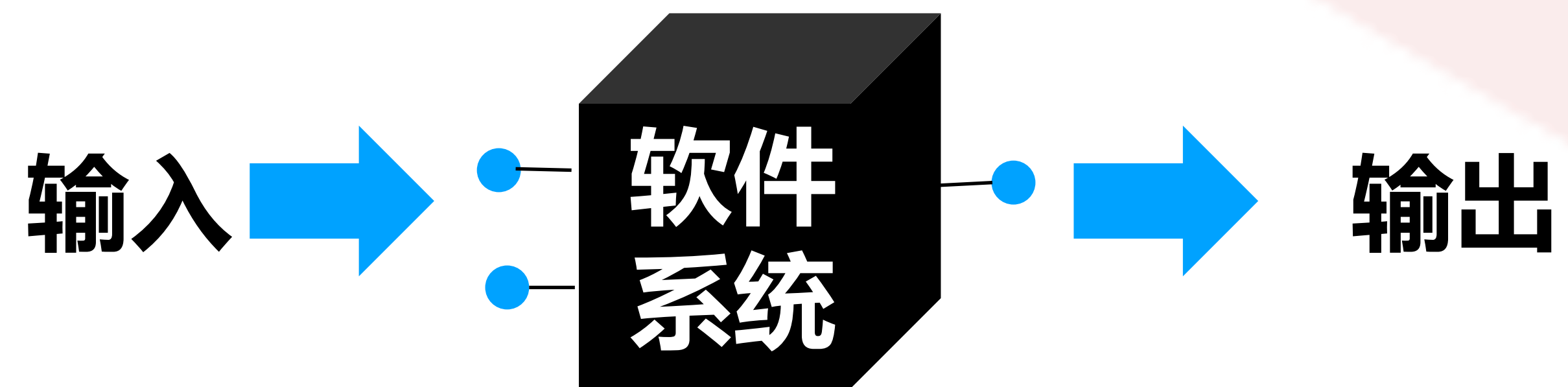


## ■ 黑盒测试（数据驱动测试）

- 它是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。

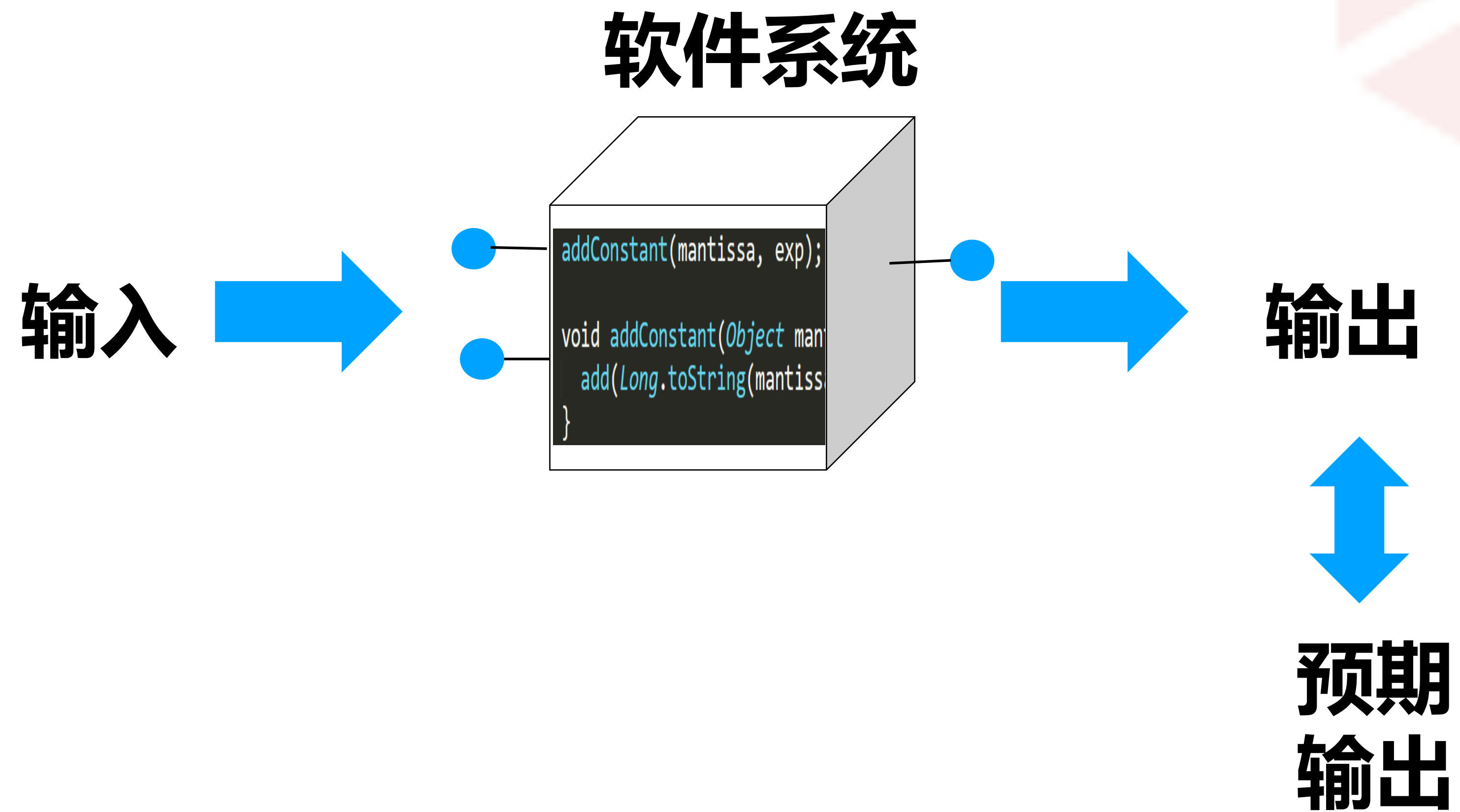
## ■ 黑盒测试技术

- 等价类划分
- 边界值分析
- 因果图
- 输入组合法
- 基于状态测试



01

# 白盒测试

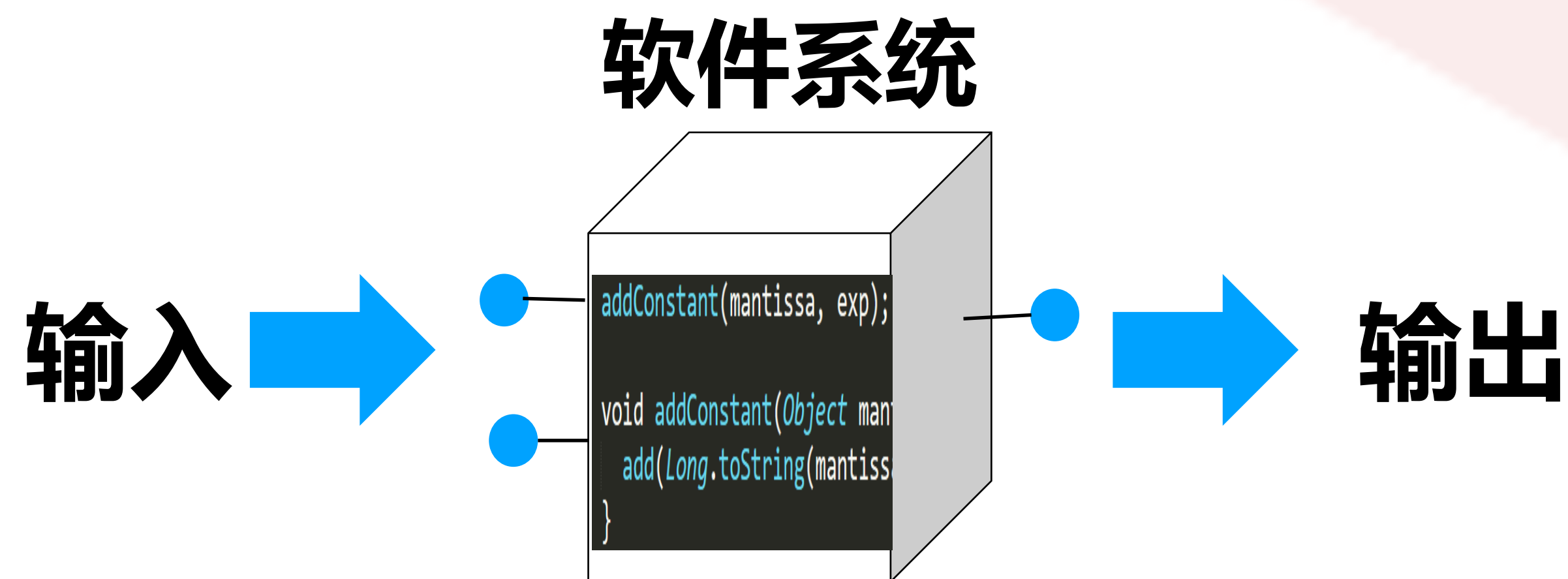




- 白盒测试把测试对象看做一个透明的盒子，所以又称**玻璃盒测试**。
- 测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。因此白盒测试又称为**结构测试**或**逻辑驱动测试**。
- 通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。

## ■ 白盒测试技术

- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定/条件覆盖
- 条件组合覆盖
- 路径覆盖



## ■ 白盒测试的优点

- 迫使测试人员去仔细思考软件的实现。
- 揭示隐藏在代码中的错误。
- 对代码的测试比较彻底。
- 优化代码。

## ■ 缺点

- 昂贵（人力/开发周期）。

## ■ 黑盒测试的优点

- 对于较大的代码单元来说，黑盒测试比白盒测试效率要高。
- 测试人员不需要了解实现的细节。
- 测试人员和编码人员是相对独立的。
- 从用户的视角进行测试，很容易被理解和接受。
- 有助于暴露任何规格不一致或有歧义的问题。
- 测试用例可以在规格完成之后马上进行。

## ■ 黑盒测试的缺点

- 只有一小部分可能的输入被测试到，要测试每个可能的输入几乎是不可能的。
- 没有清晰的和简明的规格，测试用例是很难设计的。

**白盒测试利用程序内部的逻辑结构及有关信息，设计或选择测试用例**

**白盒测试与黑盒测试各有优缺点**

**白盒测试的优点是测试比较彻底，缺点是比较昂贵**

# 目录

CONTENTS

01

白盒测试

02

**语句覆盖**

03

判定覆盖

04

条件覆盖

05

条件/判定覆盖

06

条件组合覆盖

07

路径覆盖

08

小结



## ■ 白盒测试技术


- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定/条件覆盖
- 条件组合覆盖
- 路径覆盖

覆盖率测试！



## ■ 语句覆盖

- 选取足够多的测试数据，使被测试程序中每个语句至少执行一次。
- 可执行的语句，不包括注释、标记等。



```
41 public CompilationUnit unit=null;
42
43
44
45 @Override
46 public boolean visit(ConstructorInvocation node) {
47     System.out.println("ConstructorInvocation: ");
48     IMethodBinding im= node.resolveConstructorBinding();
49     //info for debugging
50     //System.out.println("Invoked Method"+node.getName());
51     IMethodBinding im=(IMethodBinding)node.resolveMethodBinding();
52     if(im==null)
53         return false;
54     String ClassName=im.getDeclaringClass().getName();
55     String MethodName=im.getName();
56     // if (ClassName.contains("HashMap"))
57     {
58         System.out.println("ConstructorInvocation: "+MethodName);
59         // System.out.println("Invoked Method from CLASS "+ClassName+" "+ node.getName());
60         //getEnclosingMethodClass(node);
61     }
62
63     return true;
64
65 }
```

```
float Compute(float A, float B, float X)
{
    if (A>1) && (B==0) X=X/A;

    if (A==2) || (X>1) X=X+1;

    return X;
}
```

## ■ 语句覆盖

➤ 选取足够多的测试数据，使被测试程序中每个语句至少执行一次。

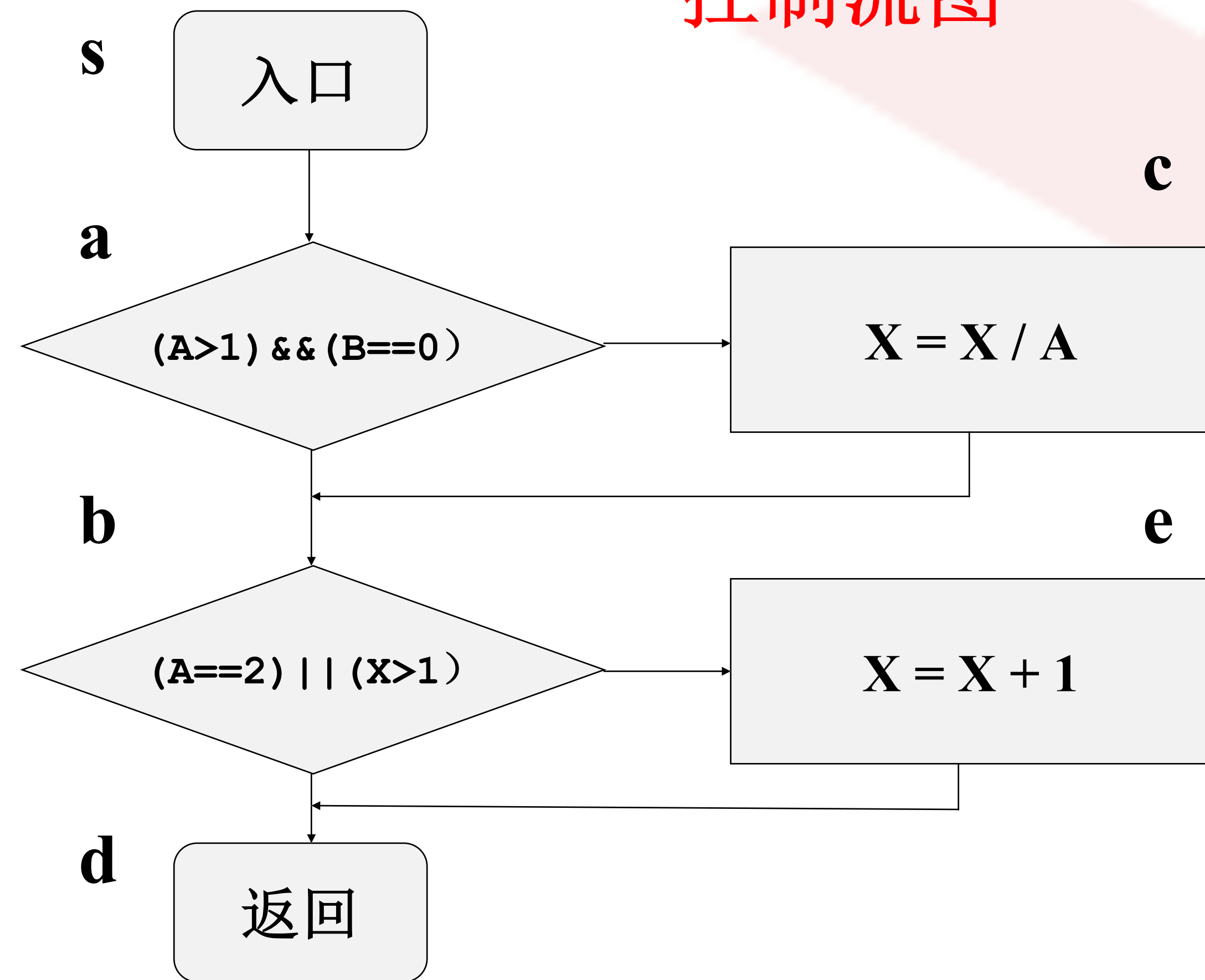
```
float Compute(float A, float B, float X)
{
    if (A>1) && (B==0) X=X/A;

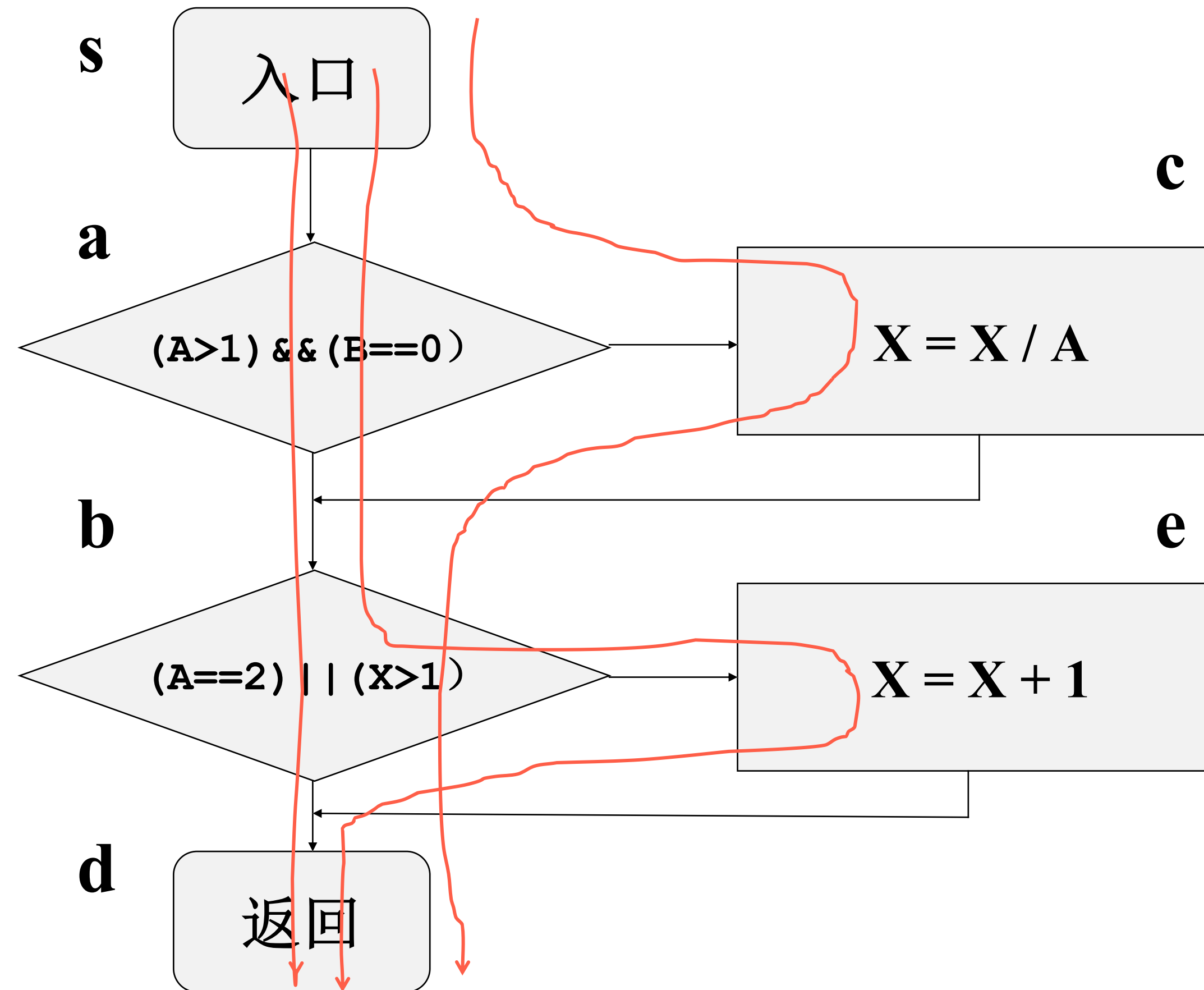
    if (A==2) || (X>1) X=X+1;

    return X;
}
```

```
float Compute(float A, float B, float X)
{
    if (A > 1) && (B == 0) X = X / A;
    if (A == 2) || (X > 1) X = X + 1;
    return X;
}
```

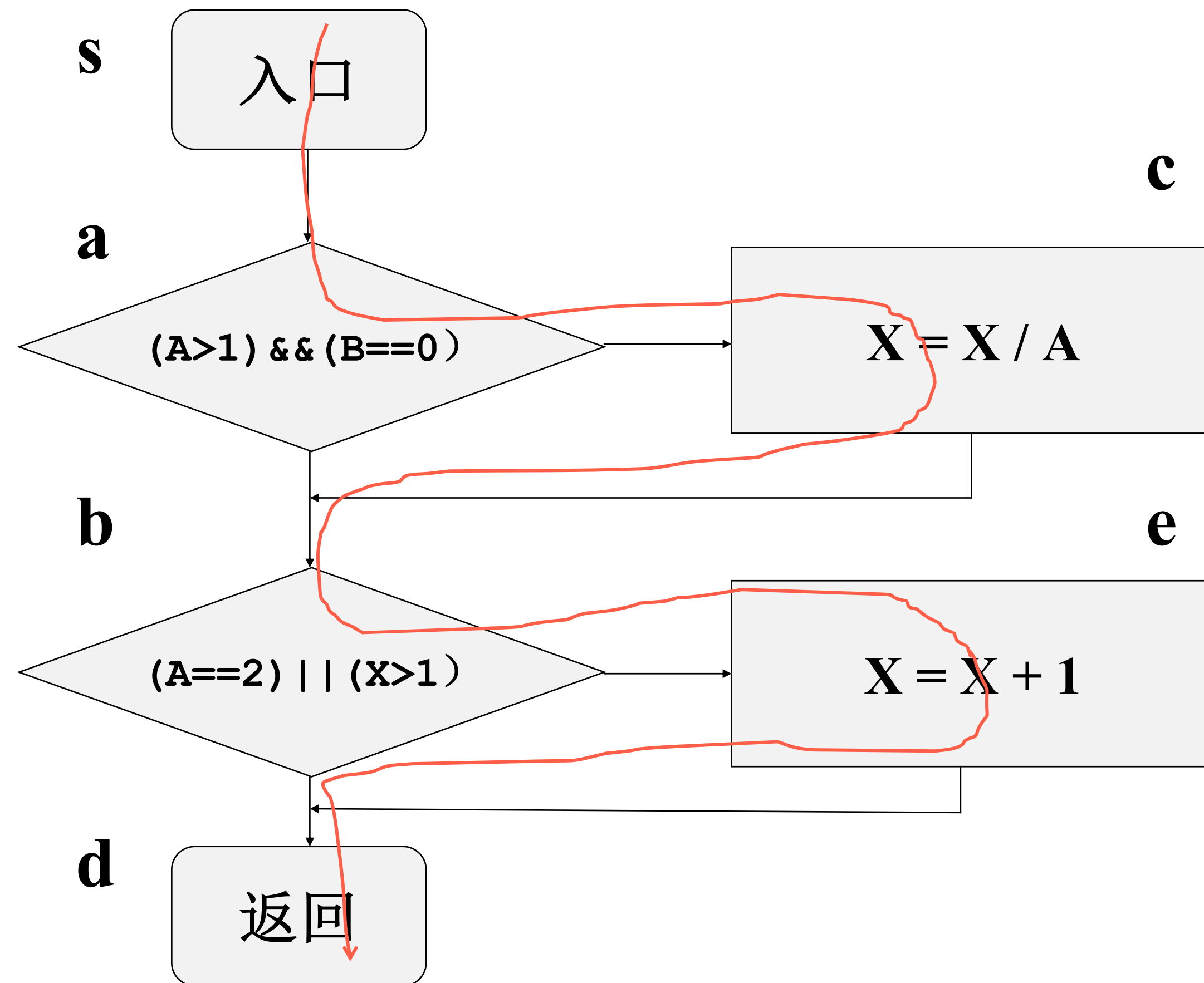
控制流图





## ■ 语句覆盖

- 选取足够多的测试数据，使被测试程序中每个语句至少执行一次。
- 使用尽可能少的测试用例！



## ■ 测试用例

ID	A	B	X
1	2	0	4

```
float Compute(float A,float B,float X)
```

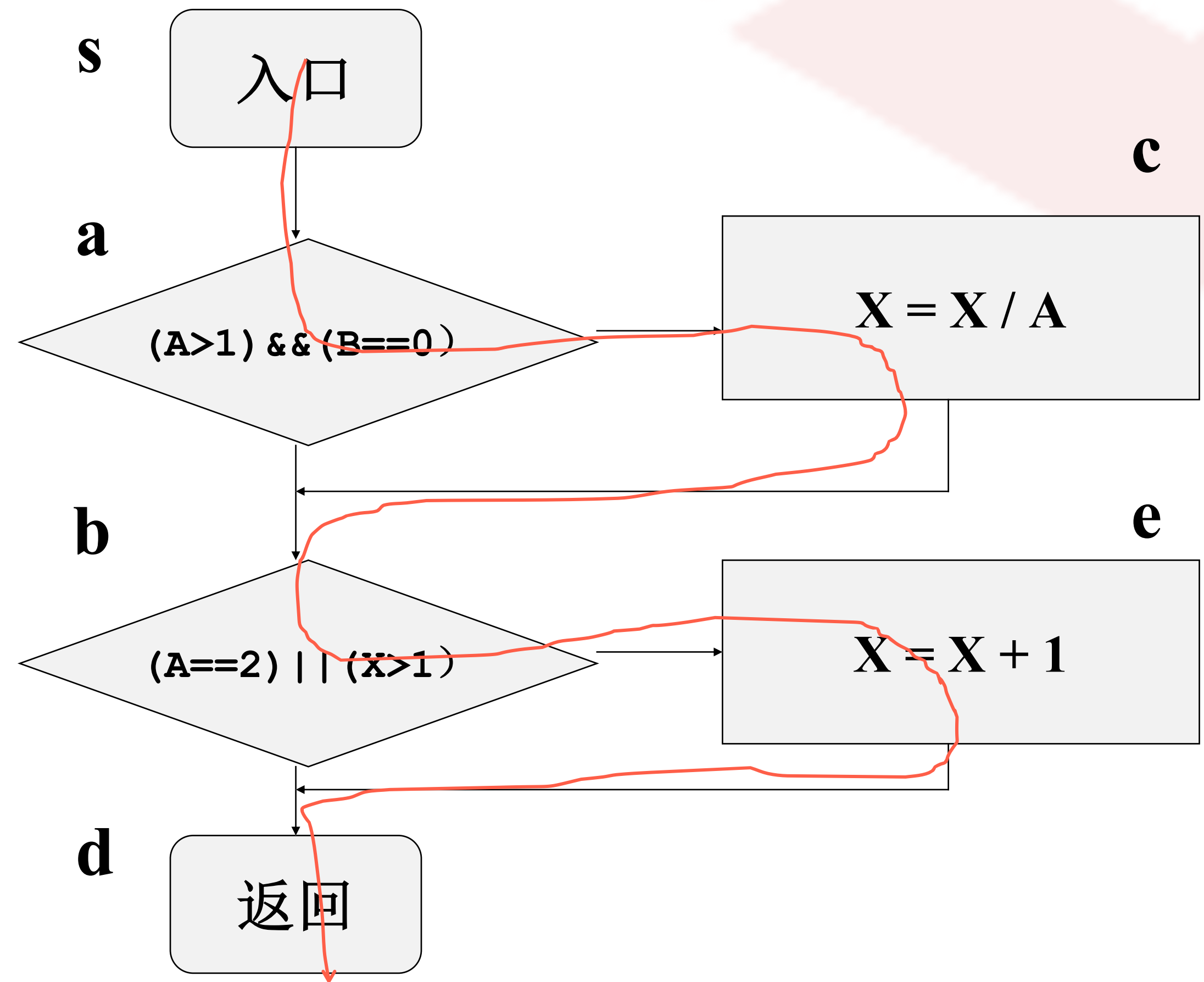
```
{
```

```
    if (A>1) && (B==0) X=X/A;
```

```
    if (A==2) || (X>1) X=X+1;
```

```
    return X;
```

```
}
```

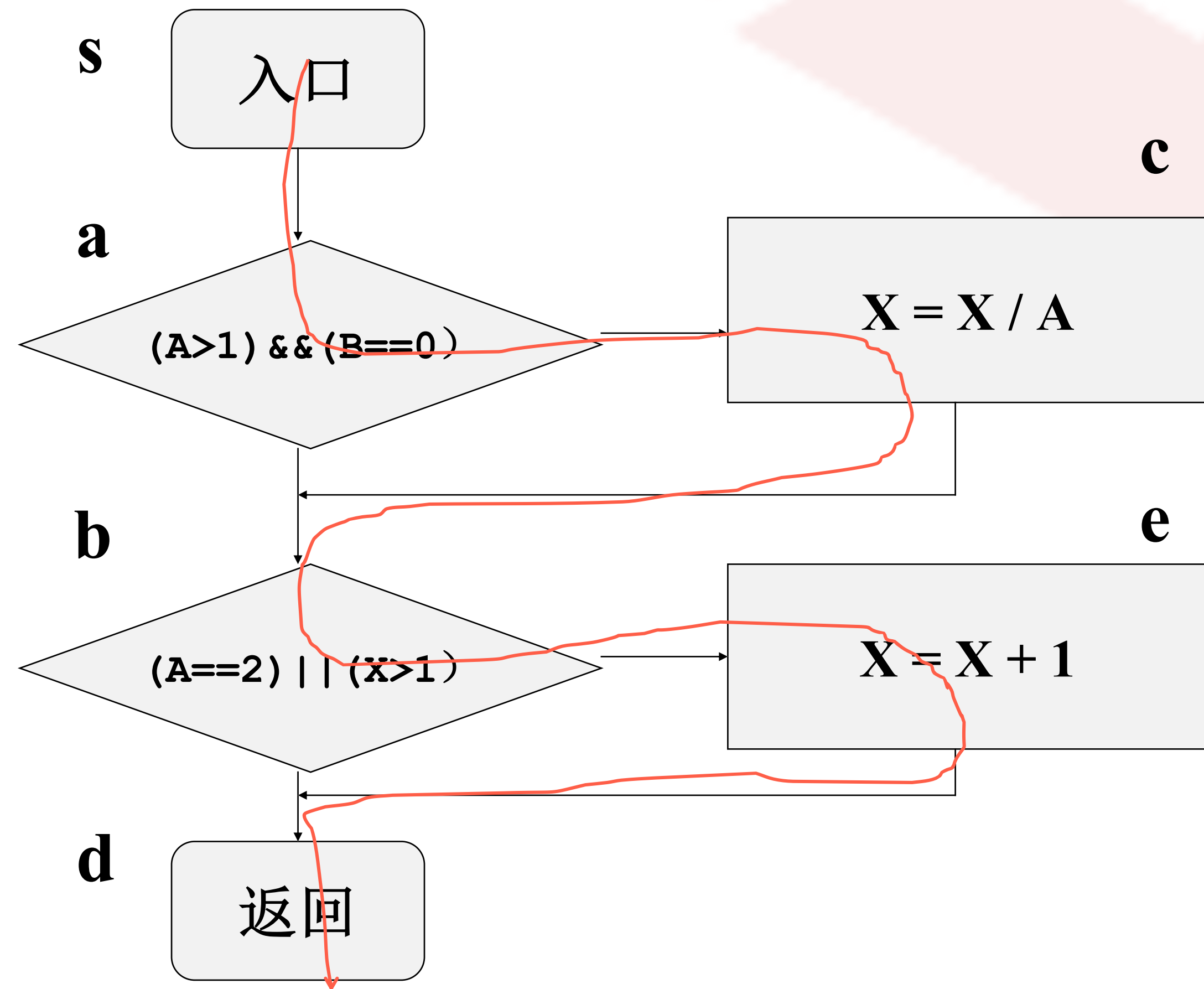




## ■ 测试用例

ID	A	B	X
1	2	0	4

## ■ 测试用例执行路径

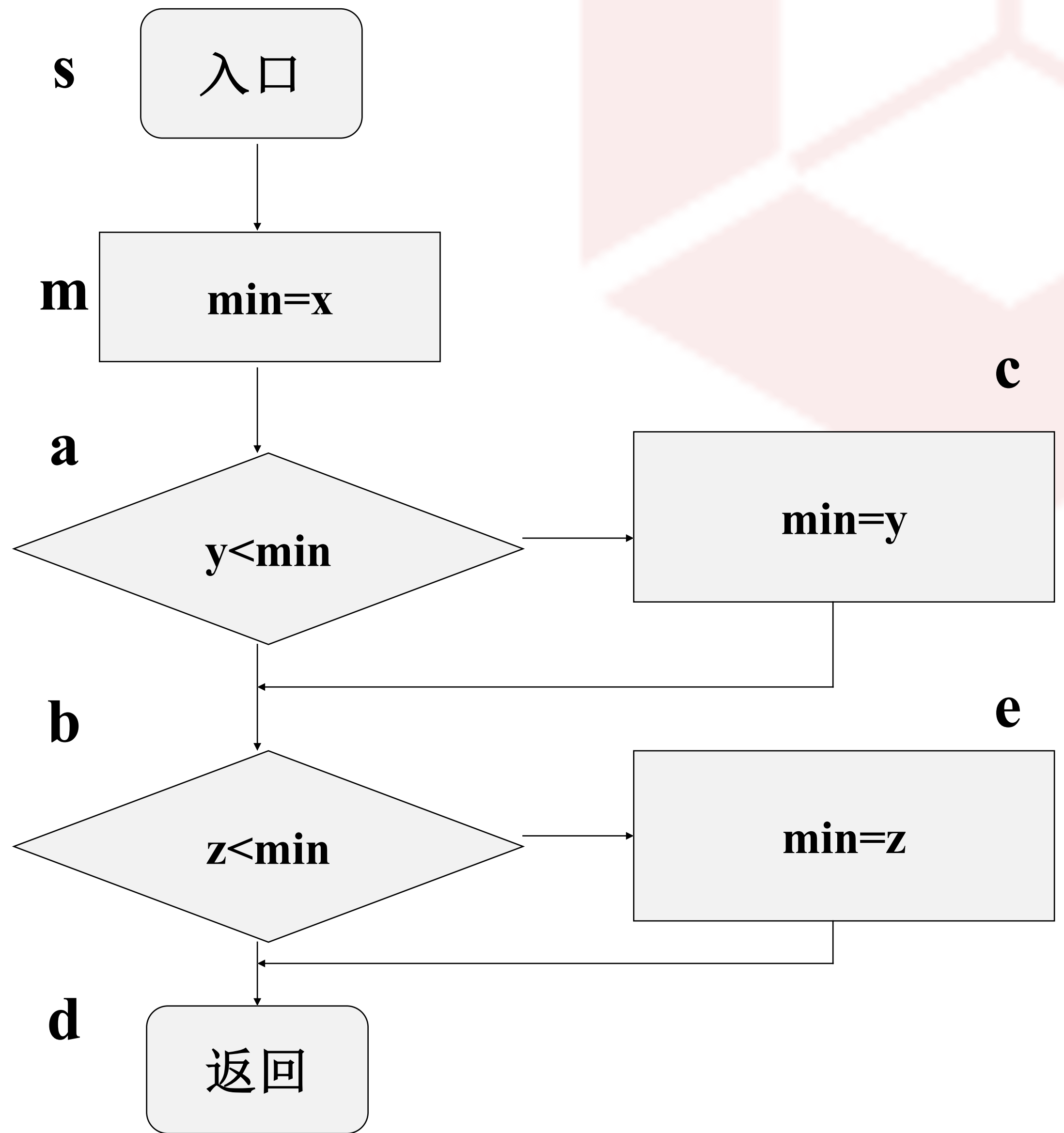
➤ **sacbed**

## ■ 语句覆盖

- 选取足够多的测试数据，使被测试程序中每个语句至少执行一次。
- 使用尽可能少的测试用例！

```
1  package dubo;  
2  
3  public class Comparator {  
4  
5      public int min(int x, int y, int z) {  
6          int min = x;  
7          if (y < min)  
8              min = y;  
9          if (z < min)  
10             min = z;  
11             return min;  
12         }  
13     }  
14
```

```
1 package dubo;  
2  
3 public class Comparator {  
4  
5     public int min(int x, int y, int z) {  
6         int min = x;  
7         if (y < min)  
8             min = y;  
9         if (z < min)  
10            min = z;  
11         return min;  
12     }  
13 }  
14
```

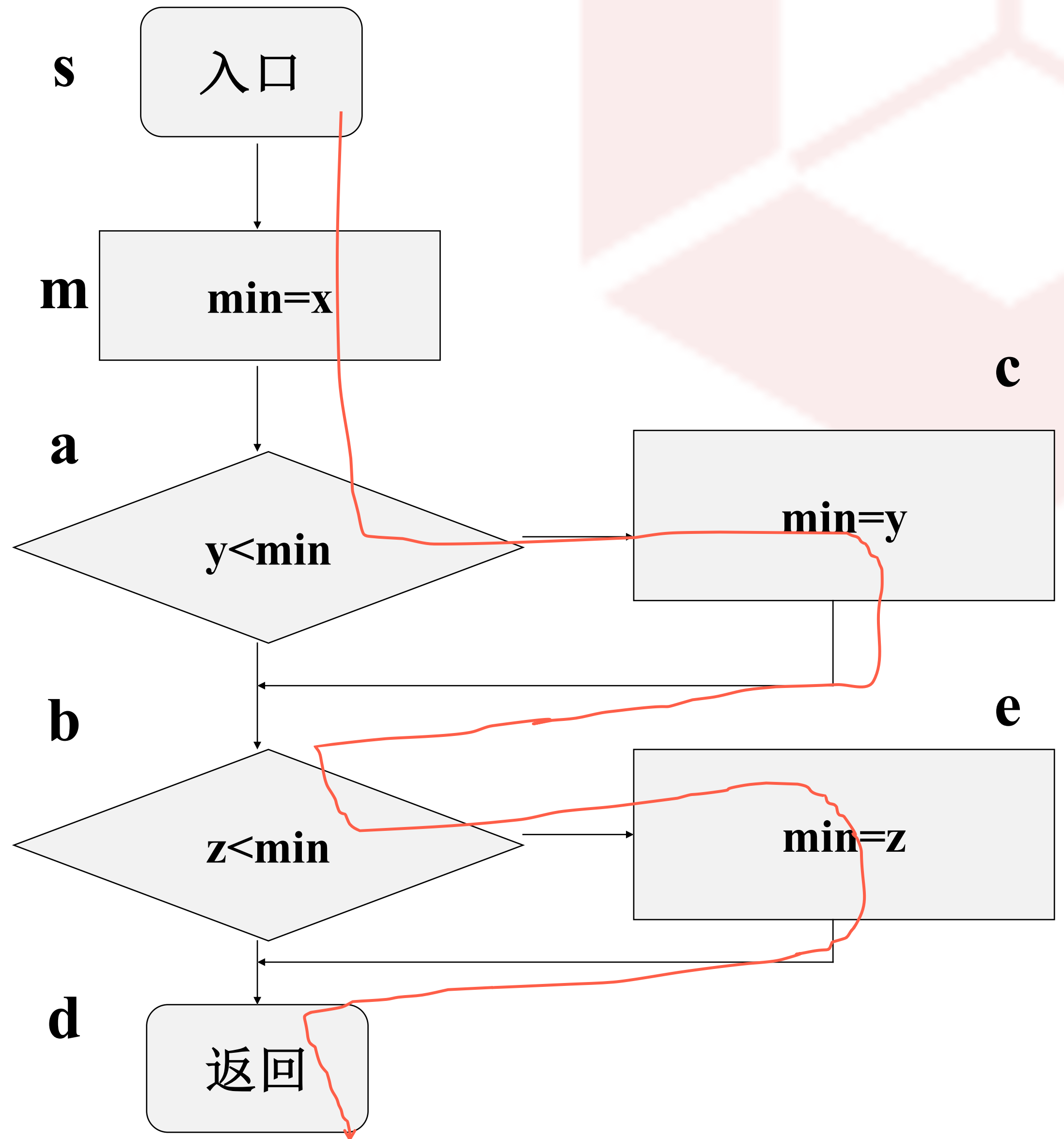


## ■ 测试用例

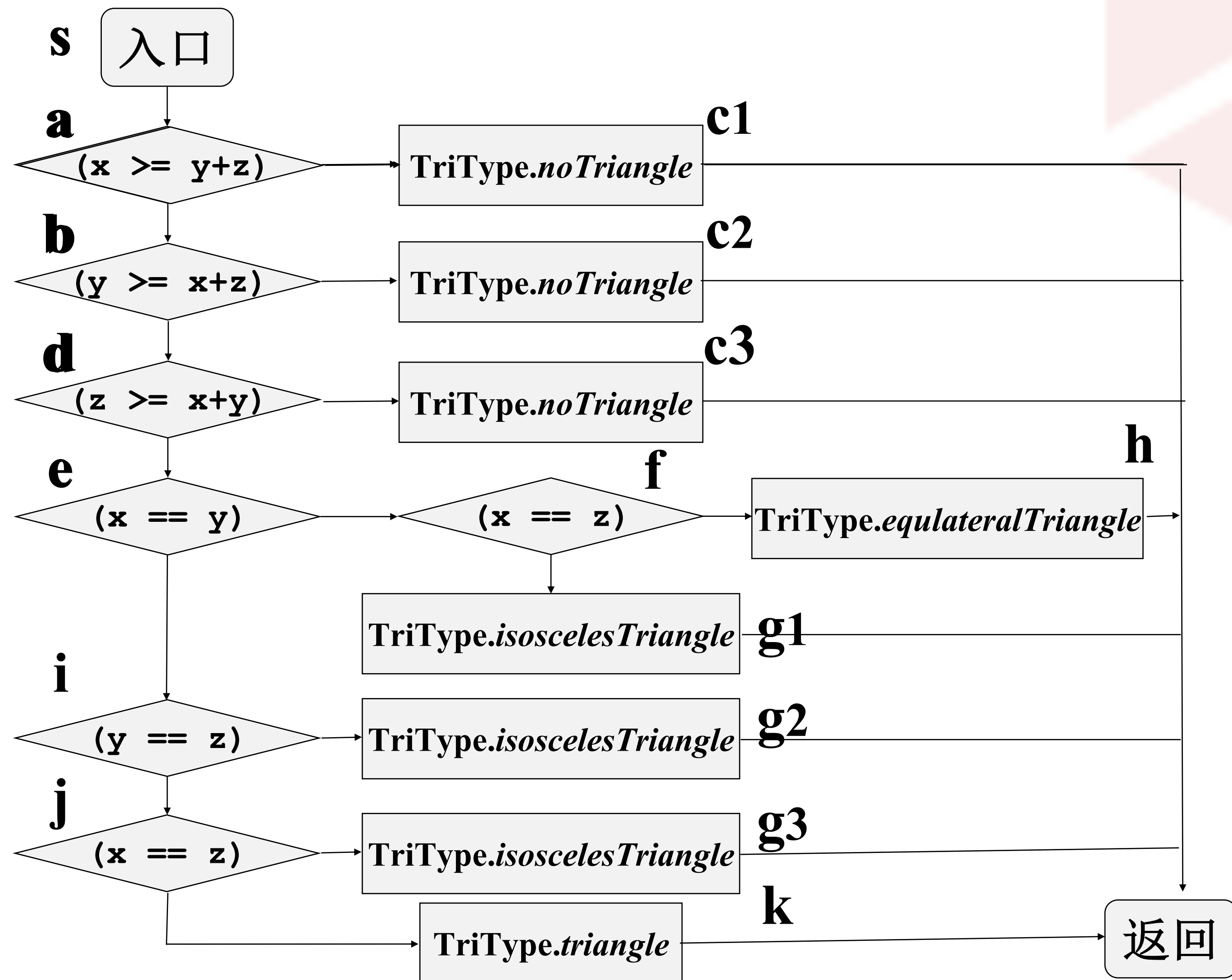
ID	x	y	z
1	9	8	7

## ■ 测试用例执行路径

➤ smacbed



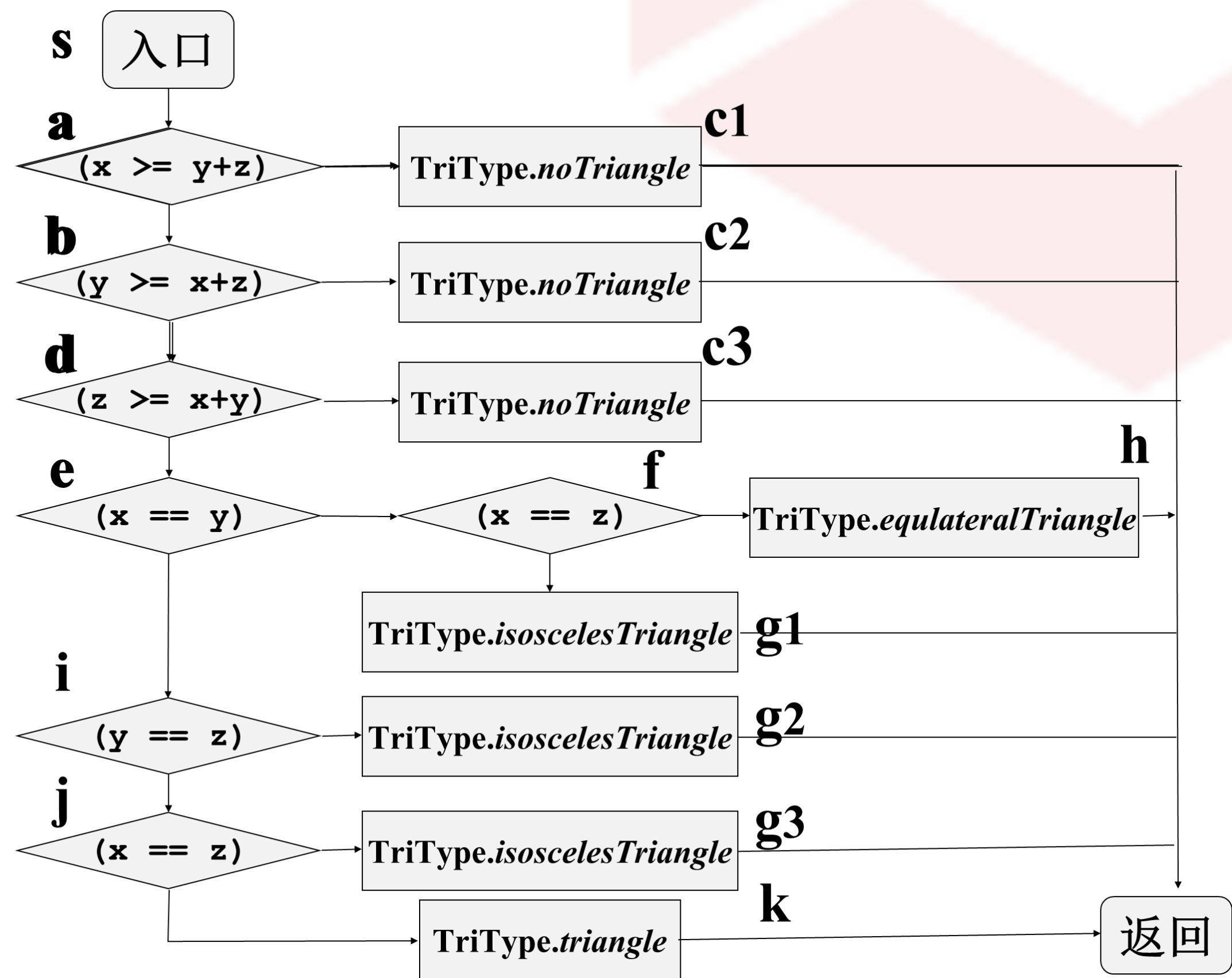
```
1 package dubo;
2 public class Triangle {
3     public TriType ReturnTriangleType(int x, int y, int z) {
4         if (x >= y + z)
5             return TriType.noTriangle;
6         if (y >= x + z)
7             return TriType.noTriangle;
8         if (z >= x + y)
9             return TriType.noTriangle;
10        if (x == y)
11            if (x == z)
12                return TriType.equilateralTriangle;
13            else
14                return TriType.isoscelesTriangle;
15        else if (y == z)
16            return TriType.isoscelesTriangle;
17        else if (x == z)
18            return TriType.isoscelesTriangle;
19        else
20            return TriType.triangle;
21    }
22 }
23 enum TriType {
24     equilateralTriangle, isoscelesTriangle, triangle, noTriangle
25 }
26 }
```





## ■ 测试用例

ID	x	y	z
1	9	1	1
2	1	9	1
3	1	1	9
4	1	1	1
5	2	2	3
6	3	2	2
7	2	3	2
8	2	3	4



# 目录

CONTENTS

01

白盒测试

02

语句覆盖

03

**判定覆盖**

04

条件覆盖

05

条件/判定覆盖

06

条件组合覆盖

07

路径覆盖

08

小结





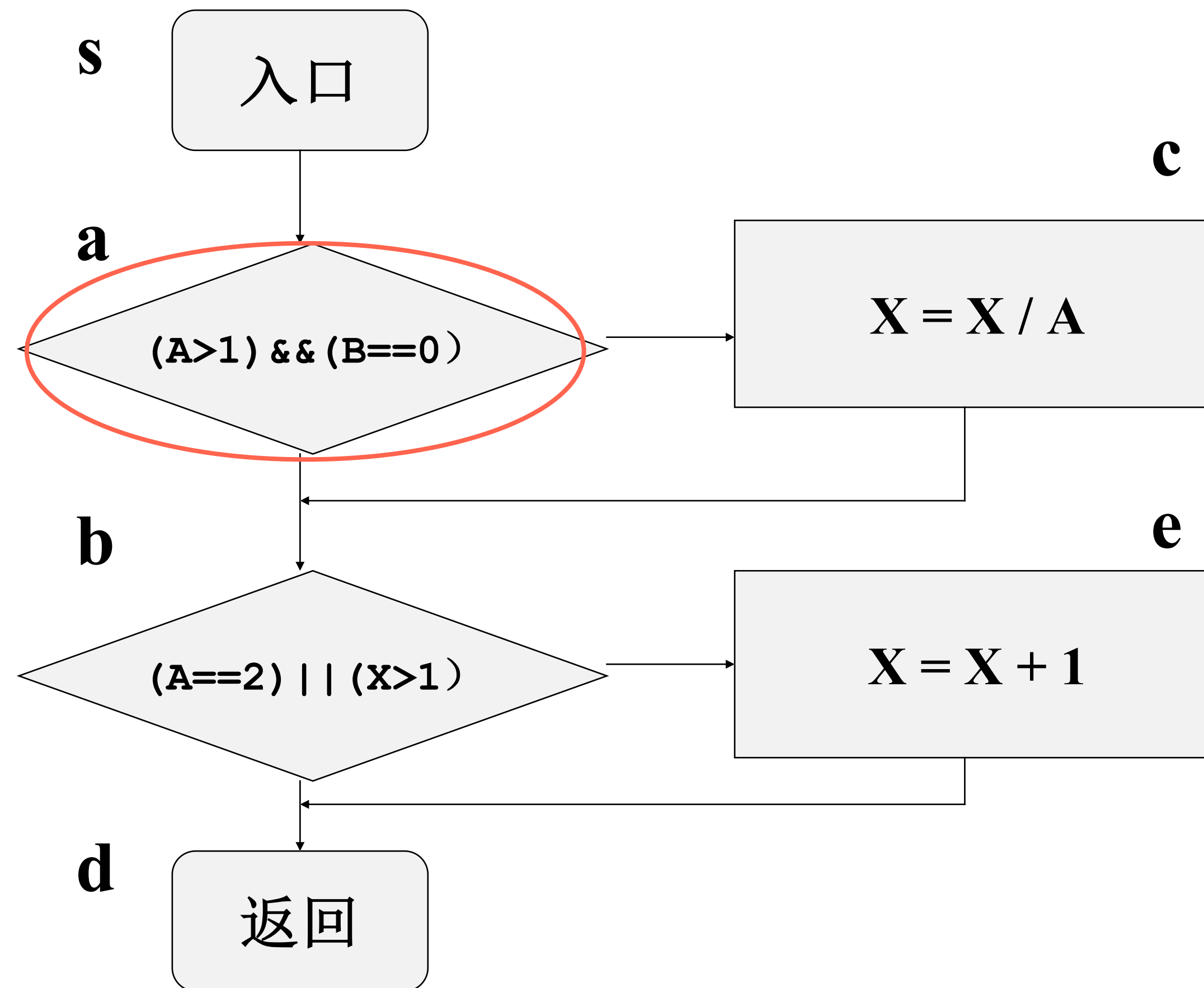
## ■ 分支/判定覆盖

- 选取足够多的测试数据，使被测试程序中不仅每个语句至少执行一次，而且每个判定的每种可能的结果都至少执行一次。

```
float Compute(float A, float B, float X)
{
    if (A>1) && (B==0) X=X/A;

    if (A==2) || (X>1) X=X+1;

    return X;
}
```



```
float Compute(float A, float B, float X)
```

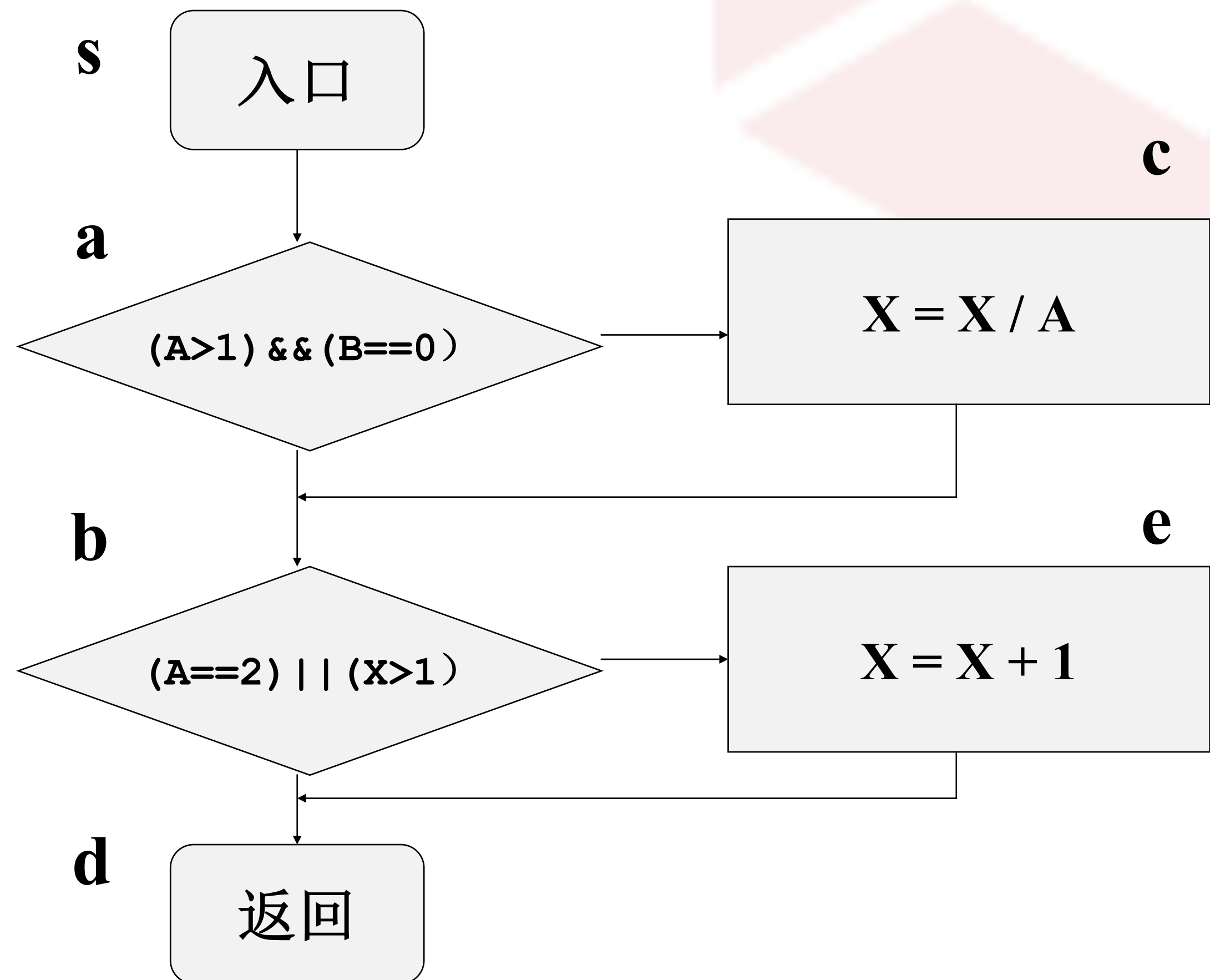
```
{
```

```
    if (A>1) && (B==0) X=X/A;
```

```
    if (A==2) || (X>1) X=X+1;
```

```
    return X;
```

```
}
```



## ■ 测试用例

ID	A	B	X
1	2	0	4
2	0	1	1

```
float Compute(float A, float B, float X)
```

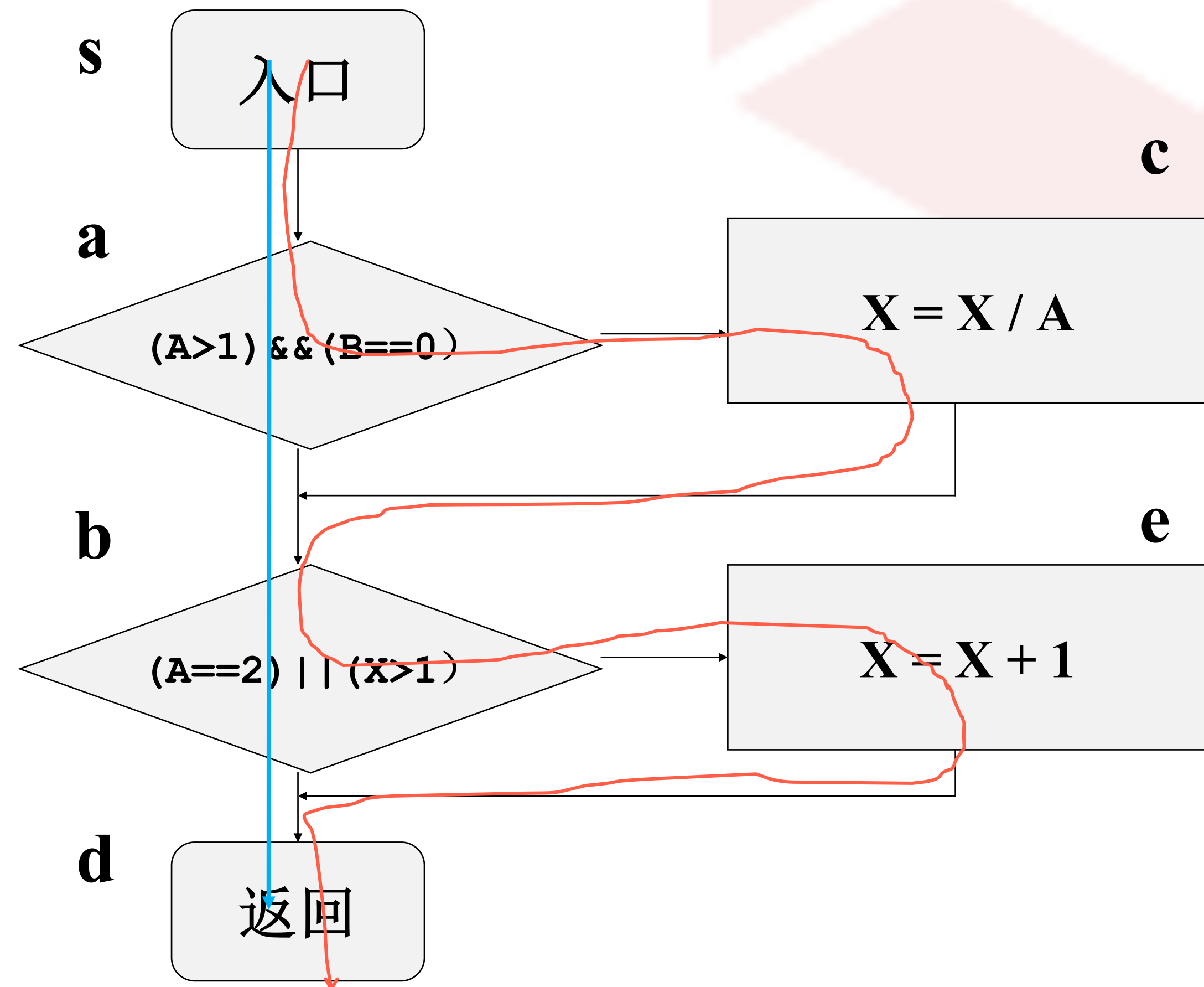
```
{
```

```
    if (A>1) && (B==0) X=X/A;
```

```
    if (A==2) || (X>1) X=X+1;
```

```
    return X;
```

```
}
```



# 分支/判定覆盖

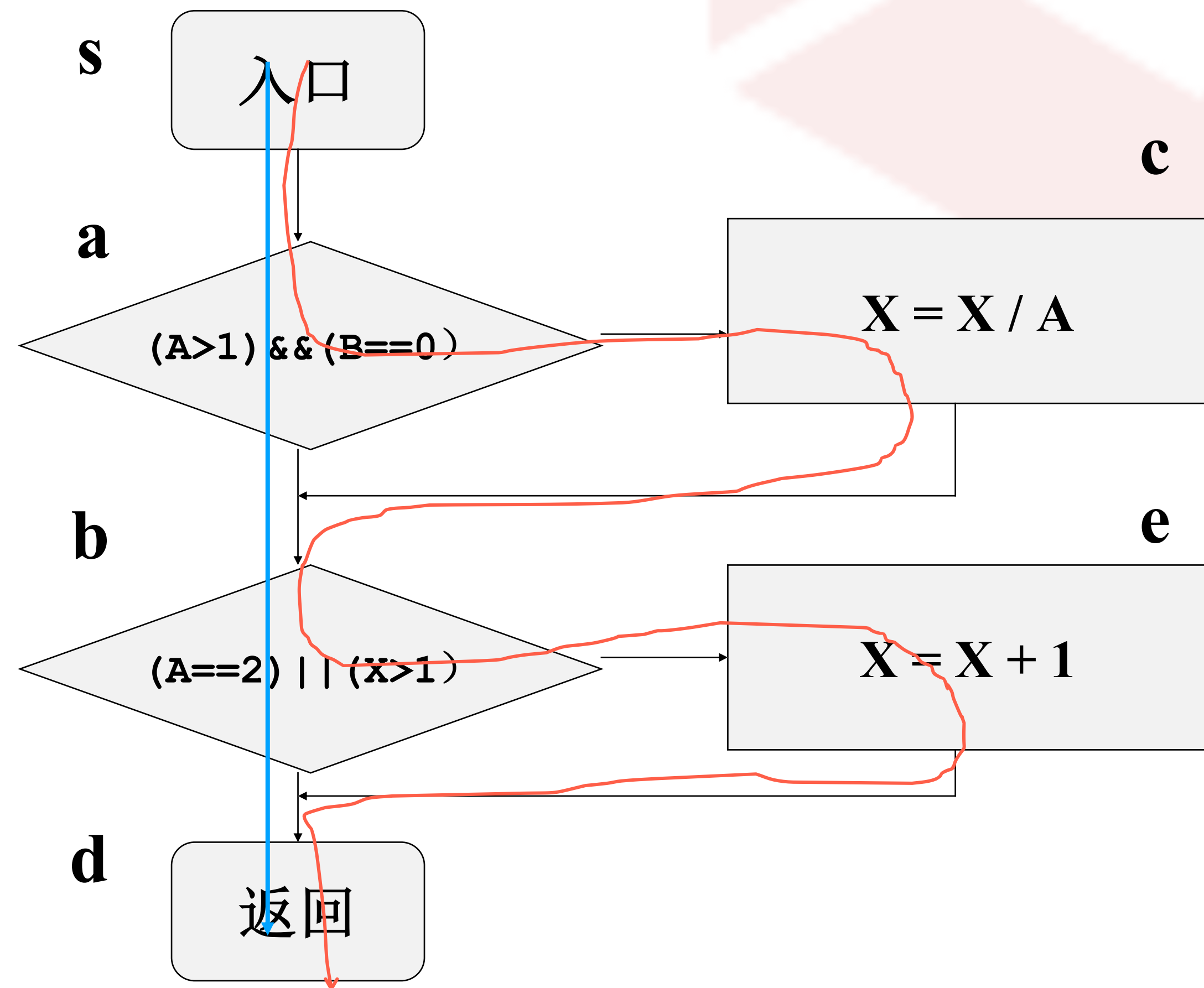
## ■ 测试用例

ID	A	B	X
1	2	0	4
2	0	1	1

## ■ 测试用例执行路径

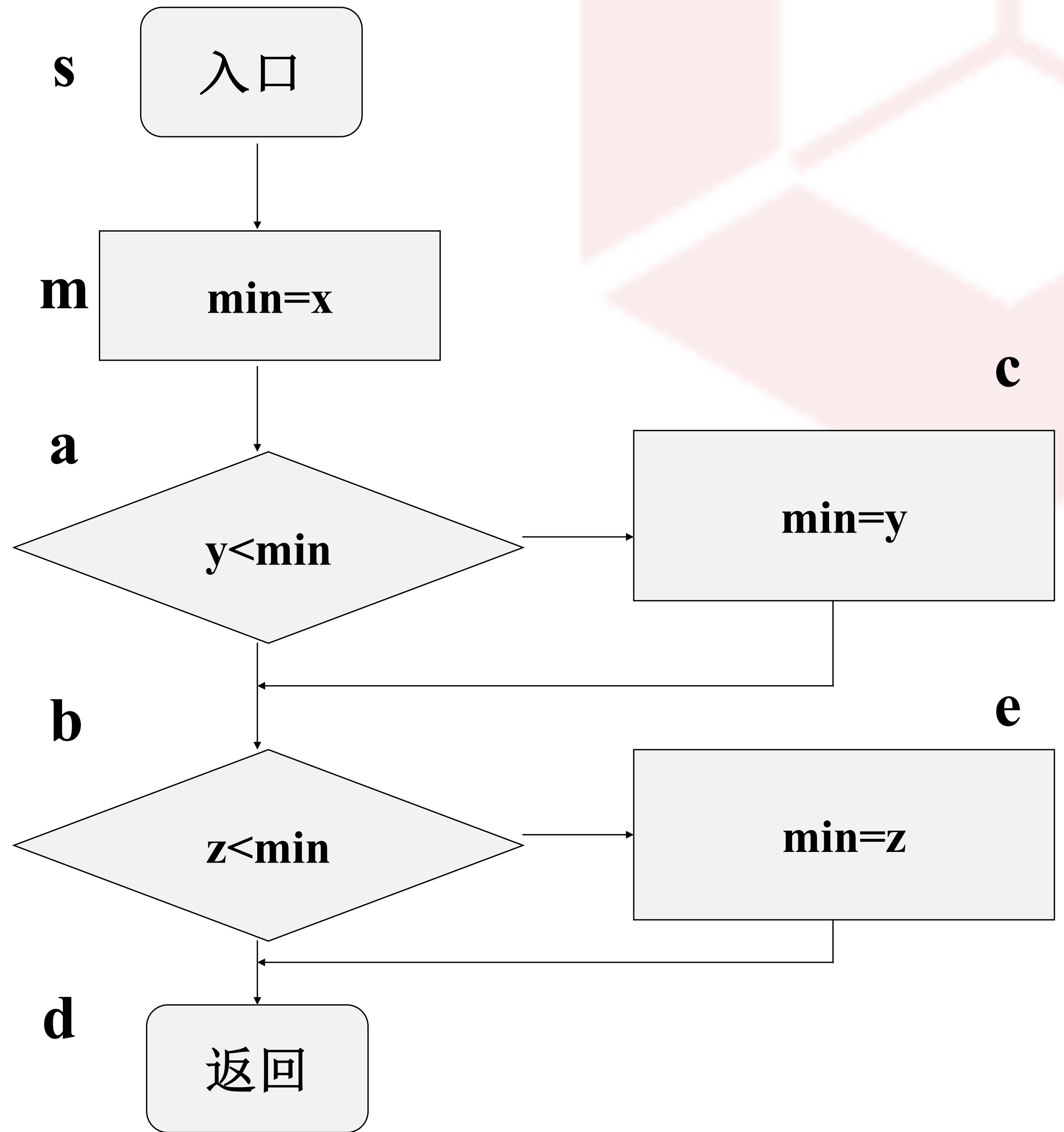
➤ sacbed

➤ sabd



```
1  package dubo;
2
3  public class Comparator {
4
5      public int min(int x, int y, int z) {
6          int min = x;
7          if (y < min)
8              min = y;
9          if (z < min)
10             min = z;
11         return min;
12     }
13 }
14
```

```
1 package dubo;  
2  
3 public class Comparator {  
4  
5     public int min(int x, int y, int z) {  
6         int min = x;  
7         if (y < min)  
8             min = y;  
9         if (z < min)  
10            min = z;  
11         return min;  
12     }  
13 }  
14
```





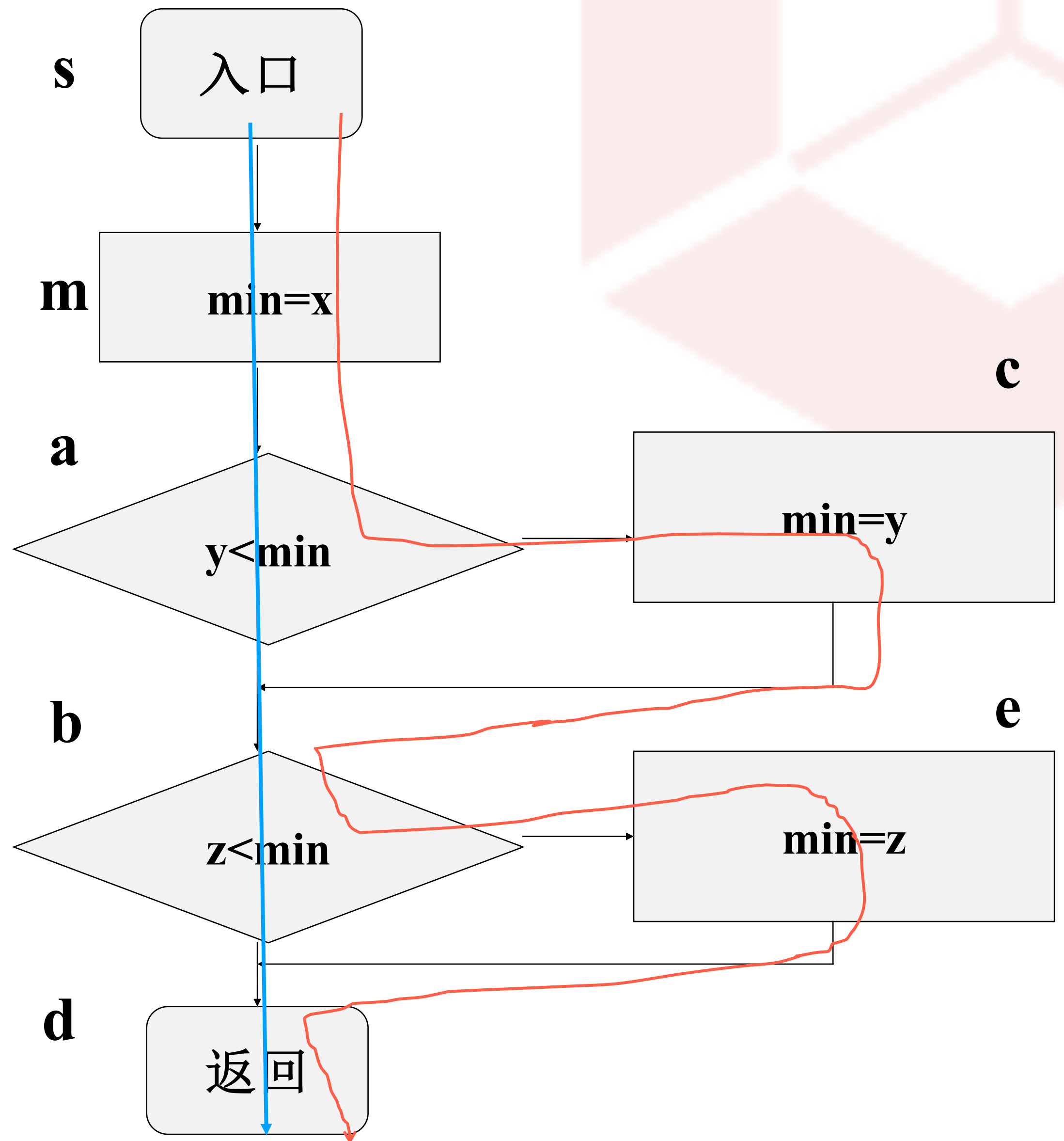
## ■ 测试用例

ID	x	y	z
1	9	8	7
2	7	8	9

## ■ 测试用例执行路径

➤ smacbed

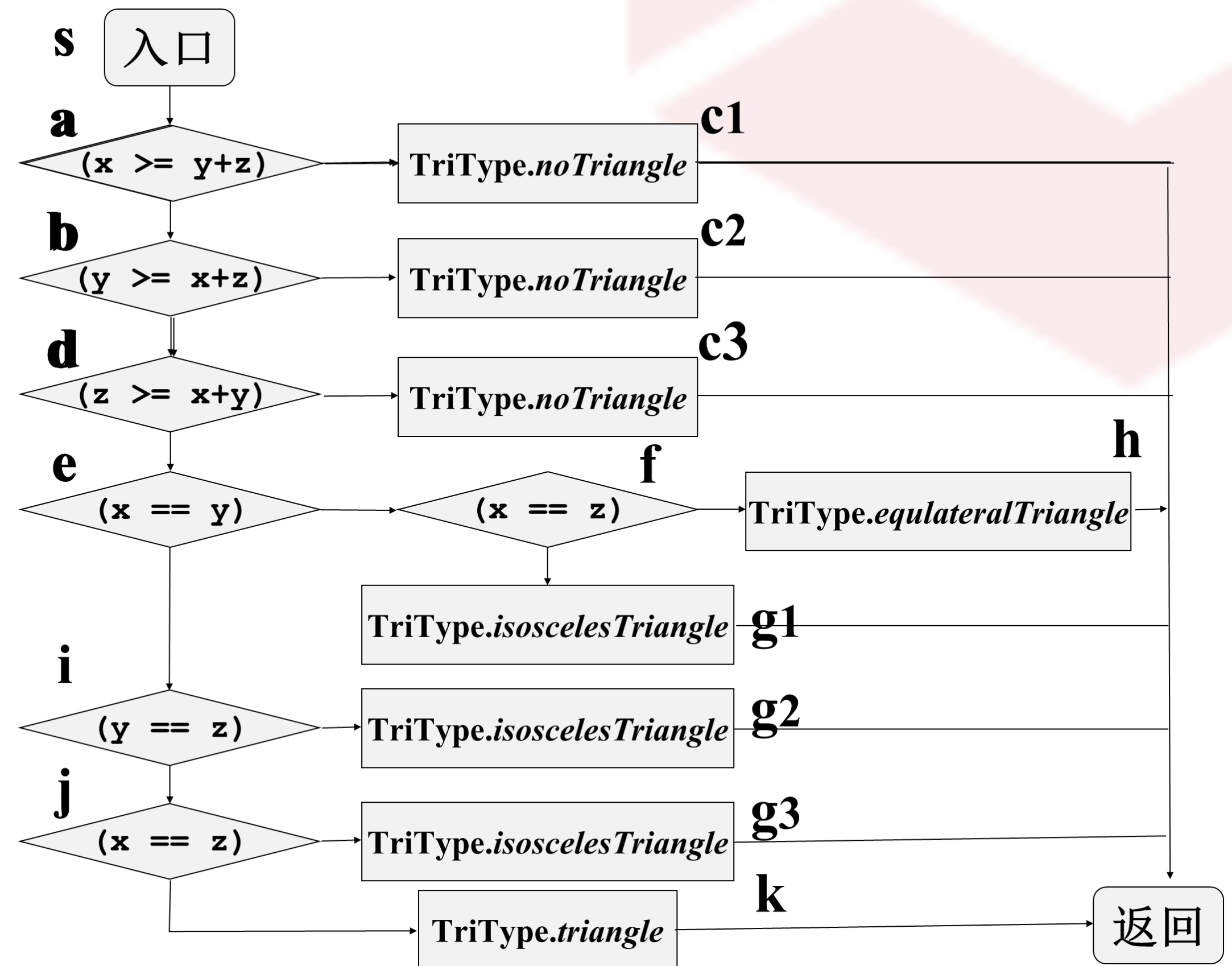
➤ smabd



```
1 package dubo;
2 public class Triangle {
3     public TriType ReturnTriangleType(int x, int y, int z) {
4         if (x >= y + z)
5             return TriType.noTriangle;
6         if (y >= x + z)
7             return TriType.noTriangle;
8         if (z >= x + y)
9             return TriType.noTriangle;
10        if (x == y)
11            if (x == z)
12                return TriType.equilateralTriangle;
13            else
14                return TriType.isoscelesTriangle;
15        else if (y == z)
16            return TriType.isoscelesTriangle;
17        else if (x == z)
18            return TriType.isoscelesTriangle;
19        else
20            return TriType.triangle;
21    }
22 }
23 enum TriType {
24     equilateralTriangle, isoscelesTriangle, triangle, noTriangle
25 }
26 }
```

## ■ 测试用例

ID	x	y	z
1	9	1	1
2	1	9	1
3	1	1	9
4	1	1	1
5	2	2	3
6	3	2	2
7	2	3	2
8	2	3	4



**谢谢**

