

Professor Phil McMinn

5.3 Symbolic Execution

dynamic

(execute the program under test)

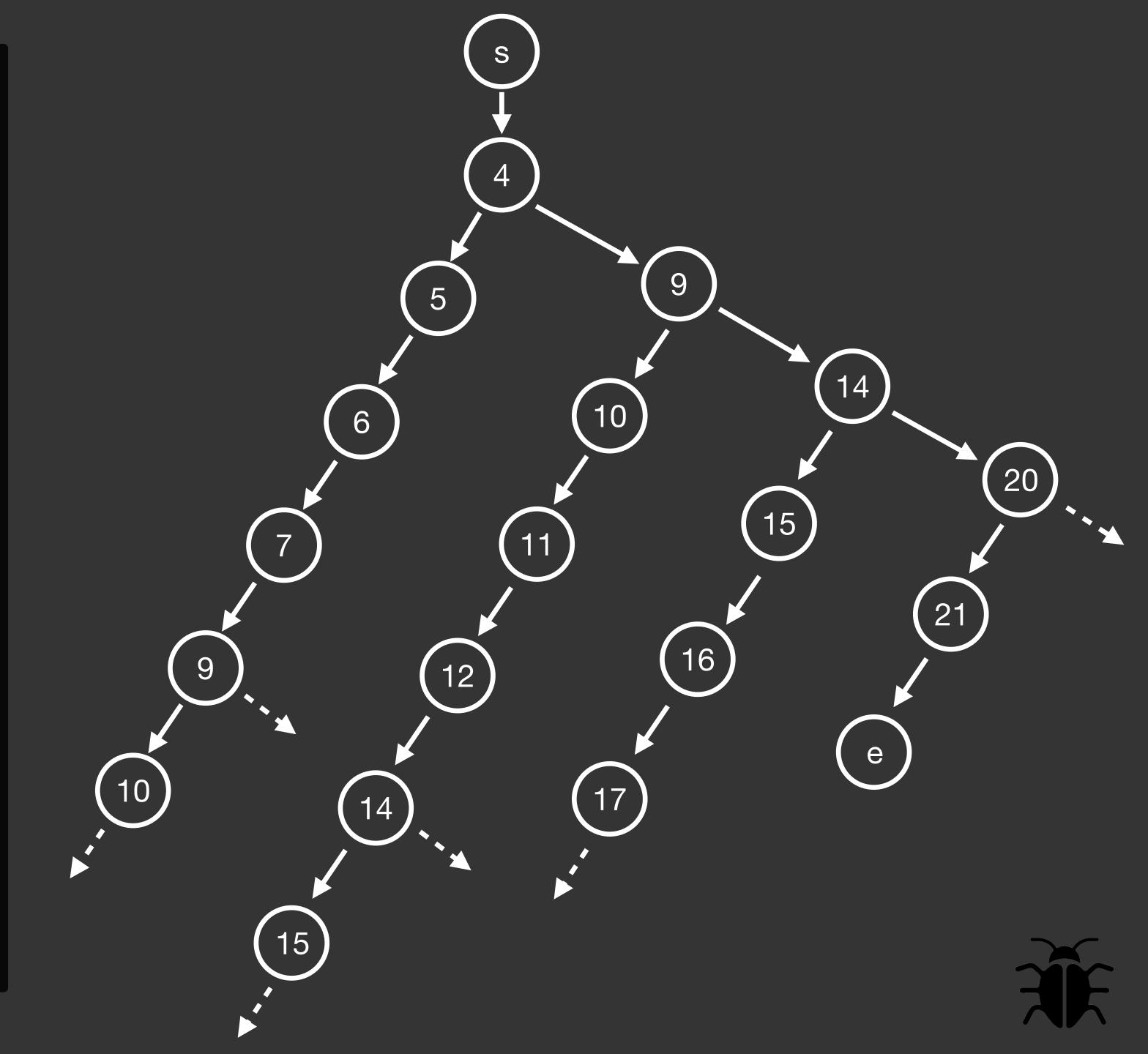
static

(examine the program without fully executing it)

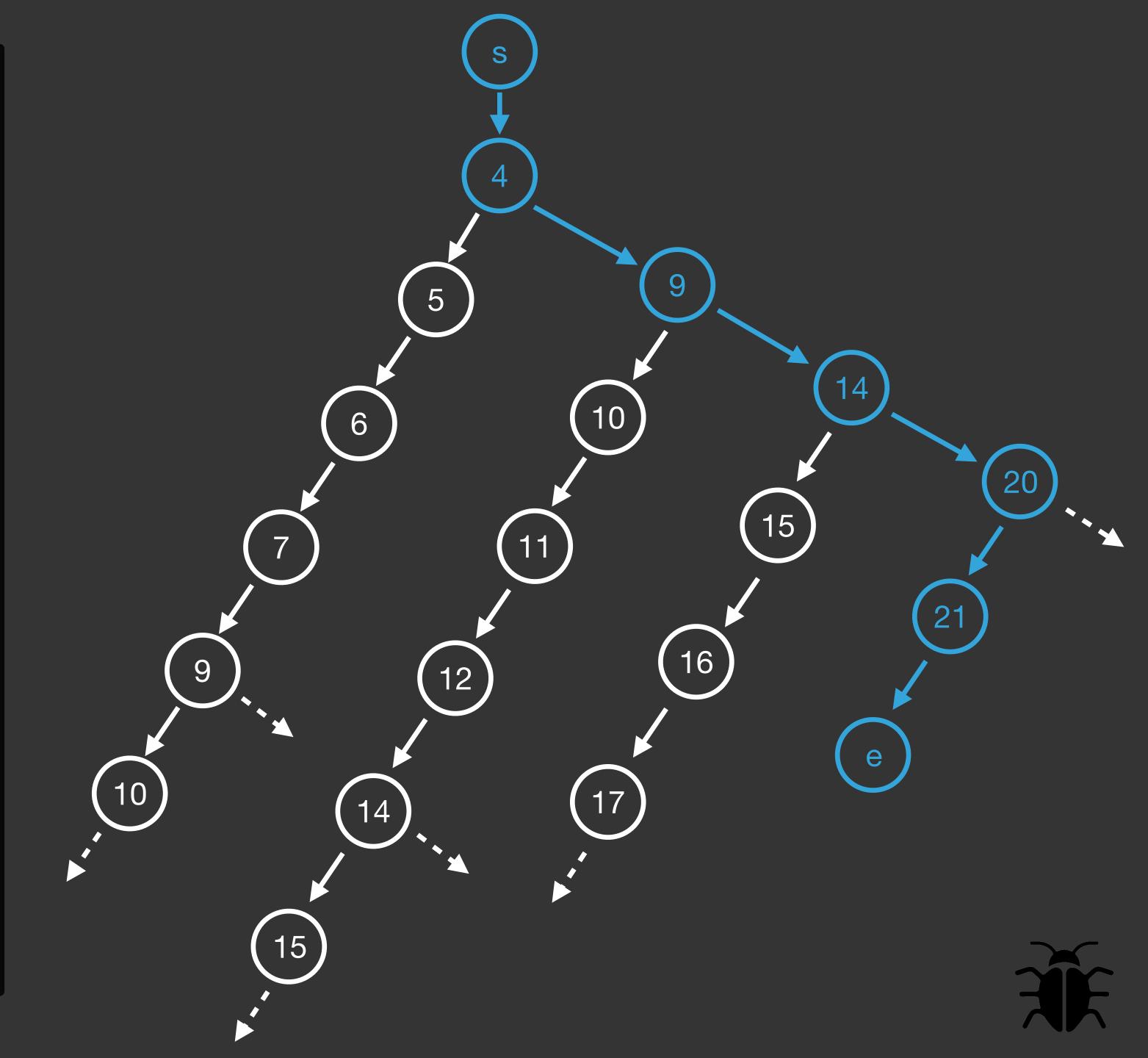
symbolic execution



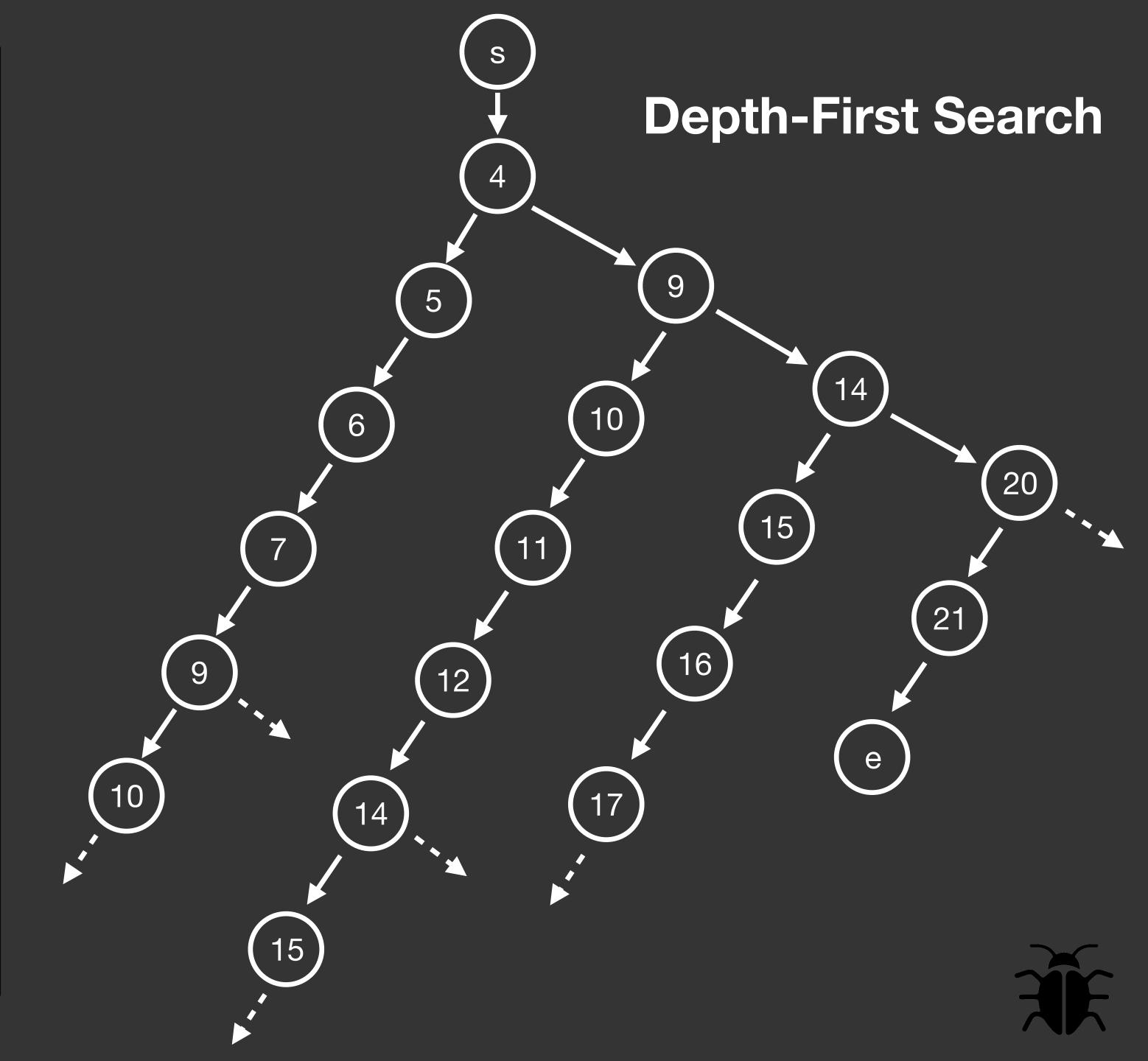
```
1 public static Type classify(int side1, int side2, int side3) {
      Type type;
 3
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
 9
10
           int temp = side1;
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
15
          int temp = side2;
16
          side2 = side3;
          side3 = temp;
17
18
19
20
      if (side1 + side2 <= side3) {</pre>
21
          type = Type.INVALID;
22
      } else {
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
                  type = Type.EQUILATERAL;
26
27
          } else {
28
29
              if (side2 == side3) {
                  type = Type.ISOSCELES;
30
31
33
      return type;
35 }
```



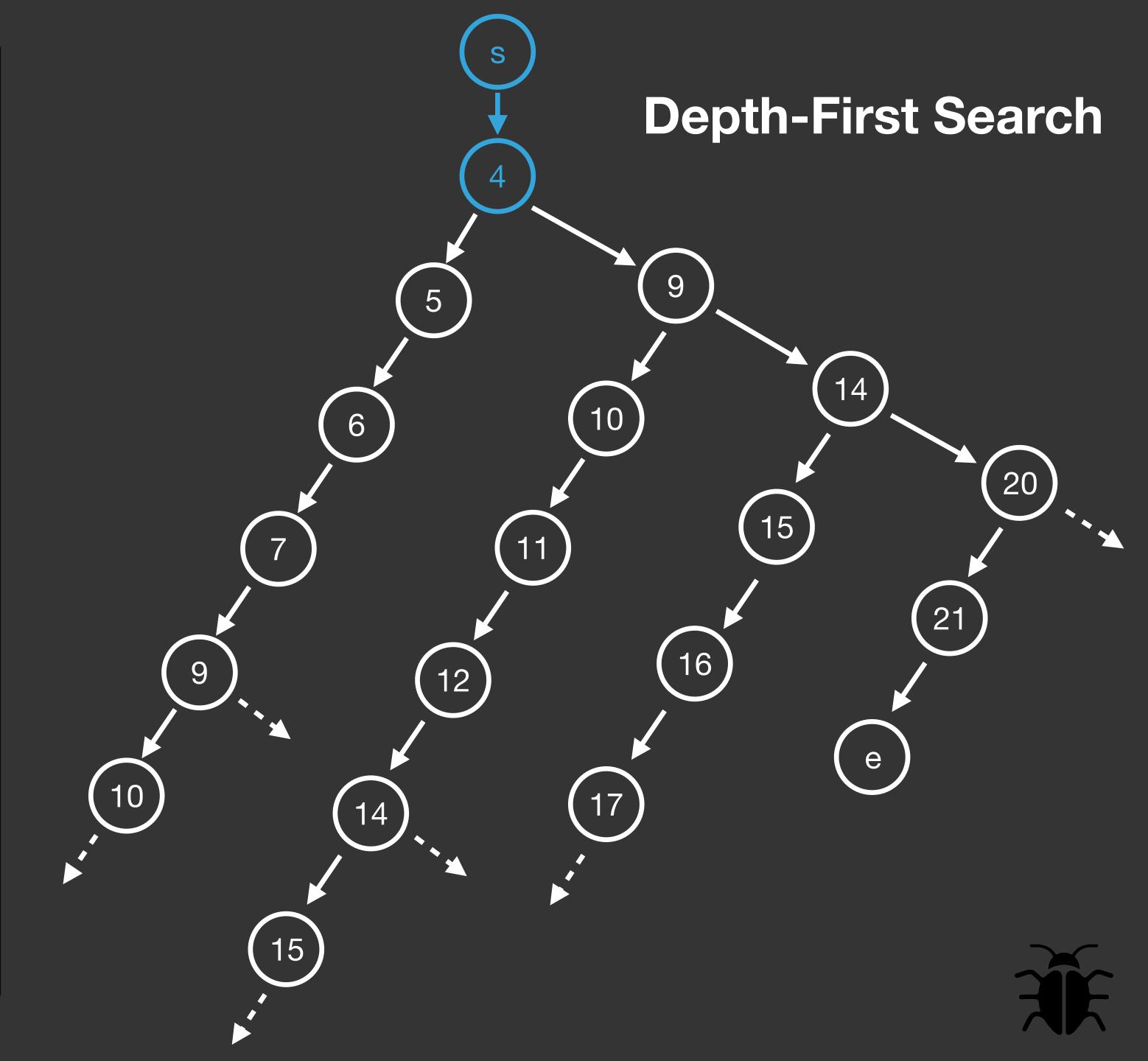
```
1 public static Type classify(int side1, int side2, int side3) {
      Type type;
 3
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
 9
10
           int temp = side1;
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
15
          int temp = side2;
16
          side2 = side3;
17
          side3 = temp;
18
19
20
      if (side1 + side2 <= side3) {</pre>
21
          type = Type.INVALID;
22
      } else {
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
                  type = Type.EQUILATERAL;
26
27
          } else {
28
29
              if (side2 == side3) {
                  type = Type.ISOSCELES;
30
31
33
      return type;
35 }
```



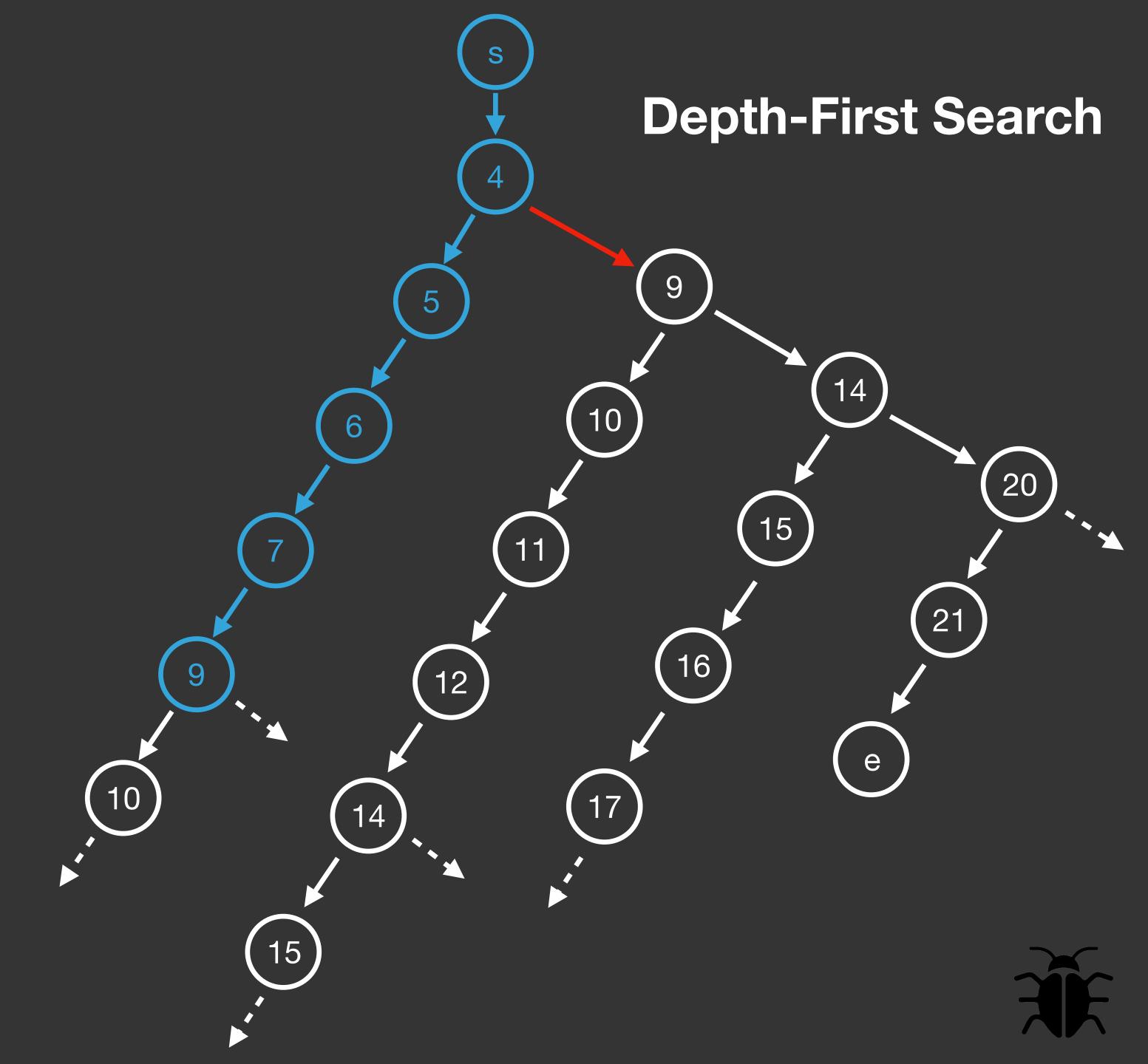
```
1 public static Type classify(int side1, int side2, int side3) {
       Type type;
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
 9
           int temp = side1;
10
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
          int temp = side2;
15
16
          side2 = side3;
17
          side3 = temp;
18
19
20
      if (side1 + side2 <= side3) {</pre>
21
          type = Type.INVALID;
22
      } else {
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
26
                   type = Type.EQUILATERAL;
27
          } else {
28
              if (side2 == side3) {
29
30
                   type = Type.ISOSCELES;
31
33
      return type;
35 }
```



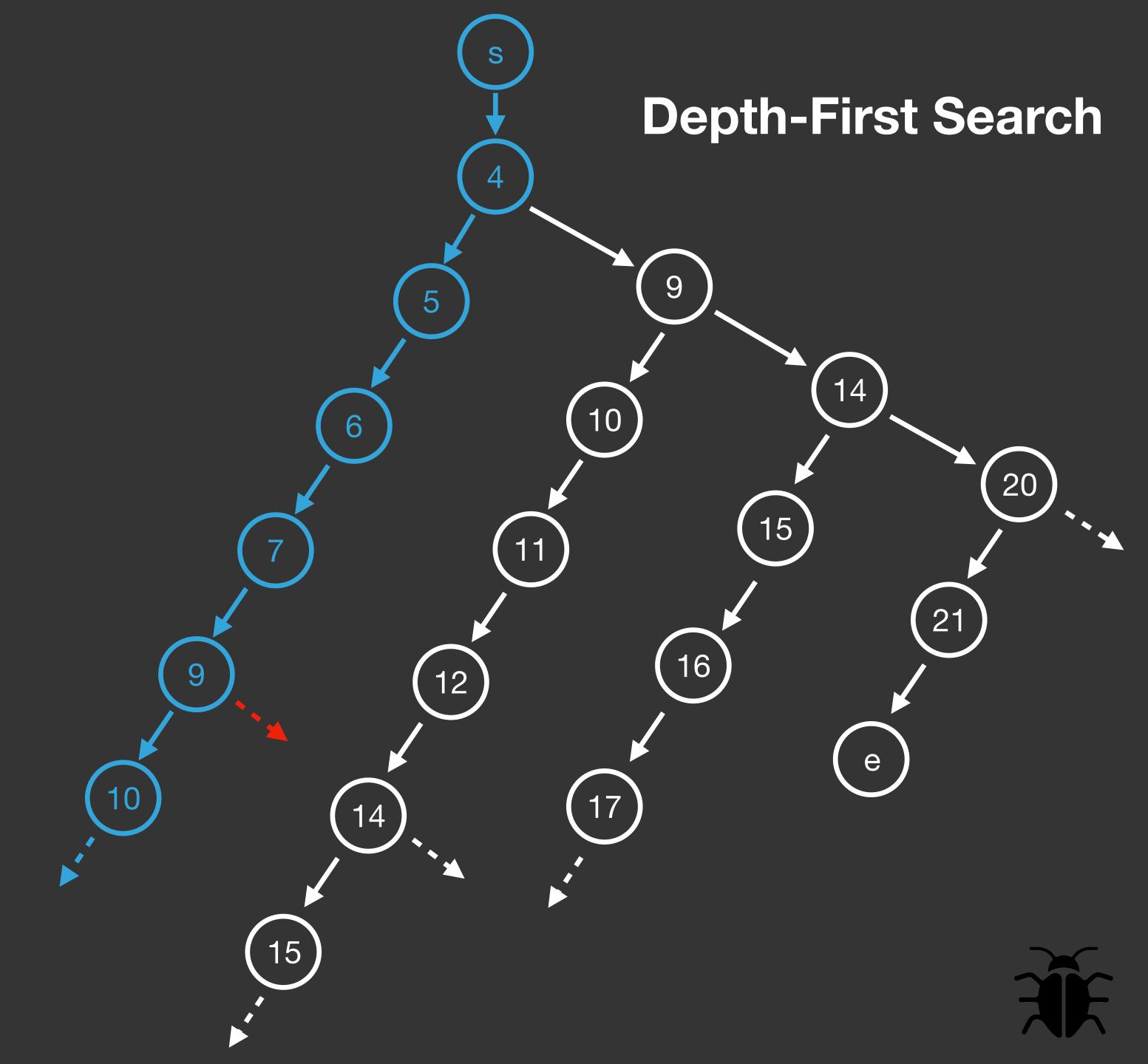
```
1 public static Type classify(int side1, int side2, int side3) {
       Type type;
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
 9
           int temp = side1;
10
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
15
          int temp = side2;
16
          side2 = side3;
17
          side3 = temp;
18
19
20
      if (side1 + side2 <= side3) {</pre>
21
          type = Type.INVALID;
22
      } else {
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
26
                   type = Type.EQUILATERAL;
27
          } else {
28
              if (side2 == side3) {
29
30
                   type = Type.ISOSCELES;
31
33
      return type;
35 }
```



```
1 public static Type classify(int side1, int side2, int side3) {
       Type type;
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
 9
           int temp = side1;
10
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
15
          int temp = side2;
16
          side2 = side3;
17
          side3 = temp;
18
19
      if (side1 + side2 <= side3) {</pre>
20
21
          type = Type.INVALID;
22
      } else {
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
26
                   type = Type.EQUILATERAL;
27
          } else {
28
              if (side2 == side3) {
29
30
                   type = Type.ISOSCELES;
31
33
      return type;
35 }
```



```
1 public static Type classify(int side1, int side2, int side3) {
       Type type;
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
 9
           int temp = side1;
10
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
15
          int temp = side2;
16
          side2 = side3;
17
          side3 = temp;
18
19
      if (side1 + side2 <= side3) {</pre>
20
21
          type = Type.INVALID;
22
      } else {
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
26
                   type = Type.EQUILATERAL;
27
          } else {
28
              if (side2 == side3) {
29
30
                   type = Type.ISOSCELES;
31
33
      return type;
35 }
```



```
CFG
                                                    Symbolic State
                                                     side2 side3 temp Path Condition
Node Code
                                              side1
      if (side1 > side2) as TRUE
      int temp = side1
     side1 = side2
      side2 = temp
      if (side1 > side3) as TRUE
  10
      int temp = side1
      side1 = side3
  12 \quad \text{side3} = \text{temp}
  14 if (side2 > side3) as TRUE
  15
      int temp = side2
     side2 = side3
  17
      side3 = temp
  20
      if (side1 + side2 <= side3) as TRUE
  21
```

CFG		Symbolic State				
Node	Code	side1	side2	side3	temp	Path Condition
S		α	β	γ	_	true
4	if (side1 > side2) <i>as TRUE</i>					
5	int temp = side1					
6	side1 = side2					
7	side2 = temp					
9	if (side1 > side3) <i>as TRUE</i>					
10	int temp = side1					
11	side1 = side3					
12	side3 = temp					
14	if (side2 > side3) <i>as TRUE</i>					
15	int temp = side2					
16	side2 = side3					
17	side3 = temp					
20	if (side1 + side2 <= side3) as TRUE					
21						
е						

CFG	Symbolic State					
Node	Code	side1	side2	side3	temp	Path Condition
S		α	β	γ	_	true
4	if (side1 > side2) <i>as TRUE</i>	lpha	eta	γ	lpha	$\alpha > \beta$
5	int temp = side1	lpha	eta	γ	lpha	$\alpha > \beta$
6	side1 = side2	eta	eta	γ	lpha	$\alpha > \beta$
7	side2 = temp	eta	lpha	γ	lpha	$\alpha > \beta$
9	if (side1 > side3) <i>as TRUE</i>					
10	int temp = side1					
11	side1 = side3					
12	side3 = temp					
14	if (side2 > side3) <i>as TRUE</i>					
15	int temp = side2					
16	side2 = side3					
17	side3 = temp					
20	if (side1 + side2 <= side3) as TRUE					
21						
e	• • •					

CFG			Symboli	c State		
Node	Code	side1	side2	side3	temp	Path Condition
S		α	β	γ	_	true
4	if (side1 > side2) <i>as TRUE</i>	lpha	eta	γ	lpha	$\alpha > \beta$
5	int temp = side1	lpha	eta	γ	lpha	$\alpha > \beta$
6	side1 = side2	eta	eta	γ	lpha	$\alpha > \beta$
7	side2 = temp	eta	lpha	γ	lpha	$\alpha > \beta$
9	if (side1 > side3) <i>as TRUE</i>	eta	lpha	γ	-	$\alpha > \beta \wedge \beta > \gamma$
10	int temp = side1	eta	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$
11	side1 = side3	γ	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$
12	side3 = temp	γ	lpha	eta	eta	$\alpha > \beta \wedge \beta > \gamma$
14	if (side2 > side3) <i>as TRUE</i>					
15	int temp = side2					
16	side2 = side3					
17	side3 = temp					
20	if (side1 + side2 <= side3) as TRUE					
21						
e						

CFG			Symboli	c State		
Node	Code	side1	side2	side3	temp	Path Condition
S		α	β	γ	-	true
4	if (side1 > side2) <i>as TRUE</i>	lpha	eta	γ	lpha	$\alpha > \beta$
5	int temp = side1	lpha	eta	γ	lpha	$\alpha > \beta$
6	side1 = side2	eta	eta	γ	lpha	$\alpha > \beta$
7	side2 = temp	eta	lpha	γ	lpha	$\alpha > \beta$
9	if (side1 > side3) <i>as TRUE</i>	eta	lpha	γ	-	$\alpha > \beta \wedge \beta > \gamma$
10	int temp = side1	eta	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$
11	side1 = side3	γ	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$
12	side3 = temp	γ	lpha	eta	eta	$\alpha > \beta \wedge \beta > \gamma$
14	if (side2 > side3) <i>as TRUE</i>	γ	lpha	eta	-	$lpha>eta\wedgeeta>\gamma$ (no change)
15	int temp = side2	γ	lpha	eta	lpha	$\alpha > \beta \wedge \beta > \gamma$
16	side2 = side3	γ	eta	eta	lpha	$\alpha > \beta \wedge \beta > \gamma$
17	side3 = temp	γ	eta	lpha	lpha	$\alpha > \beta \wedge \beta > \gamma$
20	if (side1 + side2 <= side3) as TRUE					
21						
е						

CFG			Symboli	c State		
Node	Code	side1	side2	side3	temp	Path Condition
S		α	β	γ	_	true
4	if (side1 > side2) <i>as TRUE</i>	lpha	eta	γ	lpha	$\alpha > \beta$
5	int temp = side1	lpha	eta	γ	lpha	$\alpha > \beta$
6	side1 = side2	eta	eta	γ	lpha	$\alpha > \beta$
7	side2 = temp	eta	lpha	γ	lpha	$\alpha > \beta$
9	if (side1 > side3) <i>as TRUE</i>	eta	lpha	γ	-	$\alpha > \beta \wedge \beta > \gamma$
10	int temp = side1	eta	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$
11	side1 = side3	γ	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$
12	side3 = temp	γ	lpha	eta	eta	$\alpha > \beta \wedge \beta > \gamma$
14	if (side2 > side3) as TRUE	γ	lpha	eta	-	$lpha>eta\wedgeeta>\gamma$ (no change)
15	int temp = side2	γ	lpha	eta	lpha	$\alpha > \beta \wedge \beta > \gamma$
16	side2 = side3	γ	eta	eta	lpha	$\alpha > \beta \wedge \beta > \gamma$
17	side3 = temp	γ	eta	lpha	lpha	$\alpha > \beta \wedge \beta > \gamma$
20	if (side1 + side2 <= side3) as TRUE	γ	eta	lpha	lpha	$\alpha > \beta \wedge \beta > \gamma \wedge \gamma + \beta \leq \alpha$
21		•				
е						

CFG							
Node	Code	side1	side2	side3	temp	Path Condition	
S		α	β	γ	_	true	
4	if (side1 > side2) <i>as TRUE</i>	lpha	eta	γ	lpha	$\alpha > \beta$	
5	int temp = side1	lpha	eta	γ	lpha	$\alpha > \beta$	
6	side1 = side2	eta	eta	γ	lpha	$\alpha > \beta$	
7	side2 = temp	eta	lpha	γ	lpha	$\alpha > \beta$	
9	if (side1 > side3) <i>as TRUE</i>	eta	lpha	γ	-	$\alpha > \beta \wedge \beta > \gamma$	
10	int temp = side1	eta	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$	
11	side1 = side3	γ	lpha	γ	eta	$\alpha > \beta \wedge \beta > \gamma$	
12	side3 = temp	γ	lpha	eta	eta	$\alpha > \beta \wedge \beta > \gamma$	
14	if (side2 > side3) <i>as TRUE</i>	γ	lpha	eta	-	$\alpha>\beta\wedge\beta>\gamma$ (no change)	
15	int temp = side2	γ	lpha	eta	lpha	$\alpha > \beta \wedge \beta > \gamma$	
16	side2 = side3	γ	eta	eta	lpha	$\alpha > \beta \wedge \beta > \gamma$	
17	side3 = temp	γ	eta	lpha	lpha	$\alpha > \beta \wedge \beta > \gamma$	
20	if (side1 + side2 <= side3) as TRUE	γ	eta	lpha	lpha	$\alpha > \beta \wedge \beta > \gamma \wedge \gamma + \beta \leq \alpha$	
21							
е	$lpha$ =side 1, eta =side2, and γ =side3						

 $\label{eq:side2} \mbox{side2} > \mbox{side3} \ \wedge \mbox{side3} + \mbox{side2} \leq \mbox{side1}$ e.g. $\mbox{side1} = 3, \mbox{side2} = 2, \mbox{and side3} = 1$

Dynamic Symbolic Execution

Concolic Execution



Dynamic Symbolic Execution

Concrete Symbolic Execution



Dynamic Symbolic Execution

DSE

Concrete Symbolic Execution



```
1 public static Type classify(int side1, int side2, int side3) {
      Type type;
 3
      if (side1 > side2) {
          int temp = side1;
          side1 = side2;
          side2 = temp;
 8
      if (side1 > side3) {
          int temp = side1;
10
11
          side1 = side3;
12
          side3 = temp;
13
14
      if (side2 > side3) {
          int temp = side2;
15
16
          side2 = side3;
17
          side3 = temp;
18
19
      if (side1 + side2 <= side3) {</pre>
20
21
          type = Type.INVALID;
      } else {
22
23
          type = Type.SCALENE;
24
          if (side1 == side2) {
25
              if (side2 == side3) {
26
                  type = Type.EQUILATERAL;
27
28
          } else {
29
              if (side2 == side3) {
30
                  type = Type.ISOSCELES;
31
33
      return type;
35 }
```

side1 = 4 side2 = 2 side3 = 3



```
1 public static Type classify(int side1, int side2, int side3) {
       Type type;
       if (side1 > side2) {
           CITE COMP
           side1 = side2;
 6
           side2 = temp;
 8
         (side1 > side3) {
 9
10
           the temp - states,
           side1 = side3;
11
12
           side3 = temp;
13
       if (side2 > side3) {
14
           int temp = side2;
15
           side2 = side3;
16
           side3 = temp;
17
18
19
20
       if (side1 + side2 <= side3) {</pre>
21
           type = Type.INVALID;
22
       } else {
23
           type = Type.SCALENE;
           if (side1 == side2) {
24
```

```
side1 = 4
side2 = 2
side3 = 3
```

DSE makes Symbolic Execution faster and more efficient



Dealing with Path Explosion





Toos



EvoSuite (Java) https://www.evosuite.org

KLEE (C) https://klee.github.io



