

Teste Estrutural

Teste Estrutural

- Criação de casos de teste com base na estrutura do código
- Uma vez que o teste baseado em especificações é feito, o próximo passo é aumentar o conjunto de testes com a ajuda do código-fonte.

Porque Teste Estrutural

- Podemos ter esquecido uma partição ou duas ao analisar os requisitos, e podemos notar isso ao olhar para o código-fonte
- Ao implementar o código, usamos construções de linguagem, algoritmos e estruturas de dados que não estão explícitos na documentação. Detalhes específicos da implementação também devem ser exercitados para melhorar os testes.

Teste Estrutural

- Refletir sistematicamente sobre o código-fonte, ver o que está sendo exercitado pelo conjunto de testes que derivamos com a ajuda da especificação e o que resta a ser testado.

Cobertura de Código

- Compreender as técnicas de teste estrutural significa compreender os critérios de cobertura de código.

Cobertura de Código

Exemplo

Dada uma frase, o programa deve contar o número de palavras que terminam com “s” ou “r”. Uma palavra termina quando aparece um caractere que não é uma letra. O programa retorna o número de palavras.

```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' '  
  
        for (int i = 0; i < str.length(); i++) {  
            if (!Character.isLetter(str.charAt(i)) &&  
                (last == 's' || last == 'r')) {  
                words++;  
            }  
            last = str.charAt(i);  
        }  
  
        if (last == 'r' || last == 's') {  
            words++;  
        }  
  
        return words;  
    }  
}
```

```
@Test
```

```
void twoWordsEndingWithS() {  
    int words = new CountWords().count("dogs cats");  
    assertEquals(2, words);  
}
```

```
@Test
```

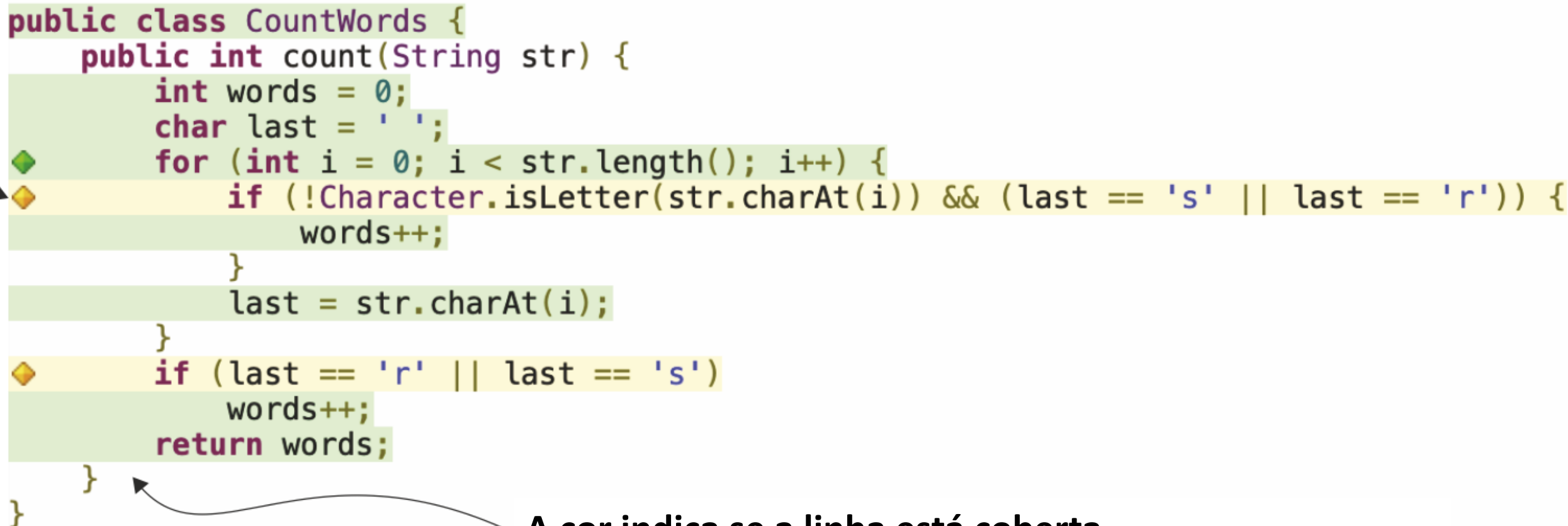
```
void noWordsAtAll() {  
    int words = new CountWords().count("dog cat");  
    assertEquals(0, words);  
}
```


Cobertura de Código

- Identificar quais partes do código nossos testes exercitam
- Existem muitas ferramentas de cobertura de código para todas as linguagens e ambientes de programação.
- Por exemplo, a JaCoCo

Resultado da JaCoCo

Losangos indicam que essa é uma instrução de desvio e pode haver vários casos para cobrir.



```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' '  
        for (int i = 0; i < str.length(); i++) {  
            if (!Character.isLetter(str.charAt(i)) && (last == 's' || last == 'r')) {  
                words++;  
            }  
            last = str.charAt(i);  
        }  
        if (last == 'r' || last == 's')  
            words++;  
        return words;  
    }  
}
```

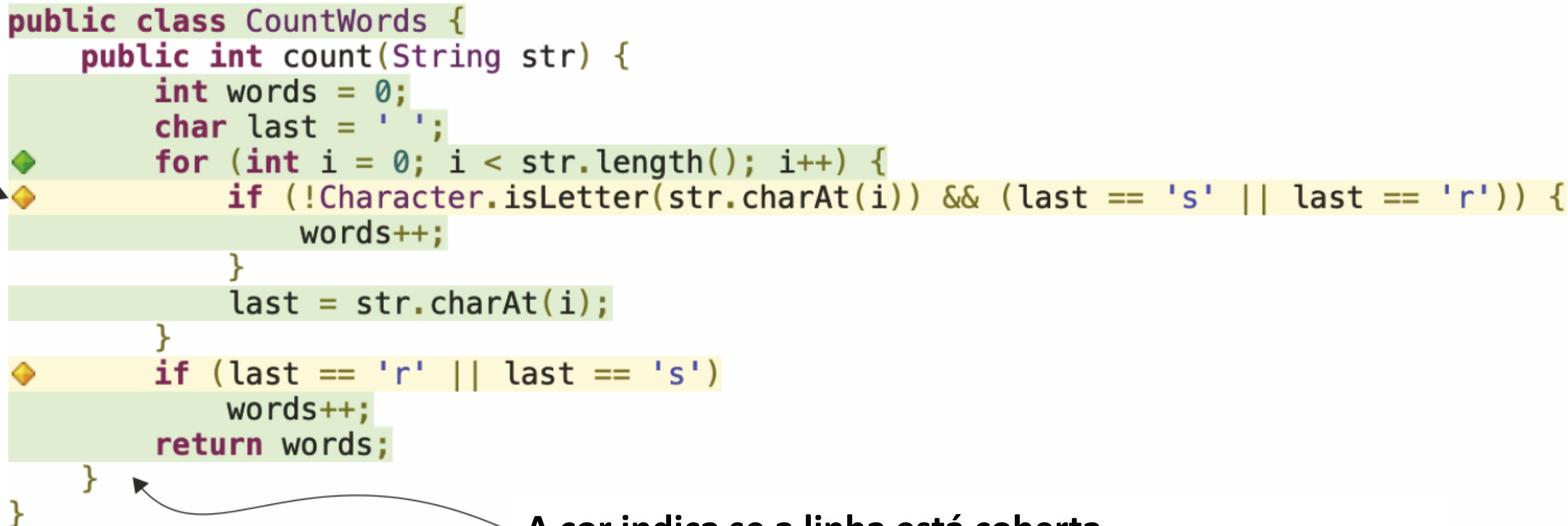
A cor indica se a linha está coberta.

JaCoCo

- Fundo verde indica que uma linha está completamente coberta
- Fundo amarelo significa que a linha está parcialmente coberta
- Fundo vermelho significa que a linha não está coberta
- Linhas sem cor de fundo (como }) são linhas que a ferramenta de cobertura não vê.
- Um losango identifica uma linha que pode ramificar o programa

Resultado da JaCoCo

Losangos indicam que essa é uma instrução de desvio e pode haver vários casos para cobrir.



```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' ';  
        for (int i = 0; i < str.length(); i++) {  
            if (!Character.isLetter(str.charAt(i)) && (last == 's' || last == 'r')) {  
                words++;  
            }  
            last = str.charAt(i);  
        }  
        if (last == 'r' || last == 's')  
            words++;  
        return words;  
    }  
}
```

A cor indica se a linha está coberta.


```
@Test
void twoWordsEndingWithS() {
    int words = new CountWords().count("dogs cats");
    assertEquals(2, words);
}
```

```
@Test
void noWordsAtAll() {
    int words = new CountWords().count("dog cat");
    assertEquals(0, words);
}
```

```
@Test
void wordsThatEndInR() {
    int words = new CountWords().count("car bar");
    assertEquals(2, words);
}
```

Resultado da JaCoCo

Todas as linhas estão verdes, o que significa que todas linhas e ramificações estão cobertas por pelo menos um caso de teste.



A curved arrow on the left side of the slide points from the text above to the code block below, indicating that the green lines in the code represent 100% coverage.

```
public class CountWords {  
    public int count(String str) {  
        int words = 0;  
        char last = ' ';  
        for (int i = 0; i < str.length(); i++) {  
            if (!Character.isLetter(str.charAt(i)) && (last == 's' || last == 'r')) {  
                words++;  
            }  
            last = str.charAt(i);  
        }  
        if (last == 'r' || last == 's')  
            words++;  
        return words;  
    }  
}
```

Teste Estrutural

- Execute testes baseados em especificações.
- Leia a implementação e entenda as principais decisões de codificação.
- Execute o conjunto de testes desenvolvido com uma ferramenta de cobertura de código.
- Para cada trecho de código não coberto:
 - Entenda por que esse pedaço de código não foi testado. Por que você não viu este caso de teste durante o teste baseado em especificações? Consulte o engenheiro de requisitos se precisar de mais explicações.
 - Decida se o pedaço de código merece um teste.
 - Se for necessário um teste, implemente um caso de teste.
- Volte ao código-fonte e procure outros testes interessantes que você possa criar com base no código.

Teste Estrutural

Testes estruturais complementam o conjunto de testes previamente elaborado por meio de testes baseados em especificações.

Critérios de Cobertura de Código

Sempre que identificamos uma linha de código que não é coberta, temos que decidir quão completos (ou rigorosos) queremos ser ao cobrir essa linha.

Critérios de Cobertura de Código

Revisitando o *if* do exemplo da classe CountWords:

```
if (!Character.isLetter(str.charAt(i)) &&  
    (last == 's' || last == 'r'))
```

Critérios de Cobertura de Código

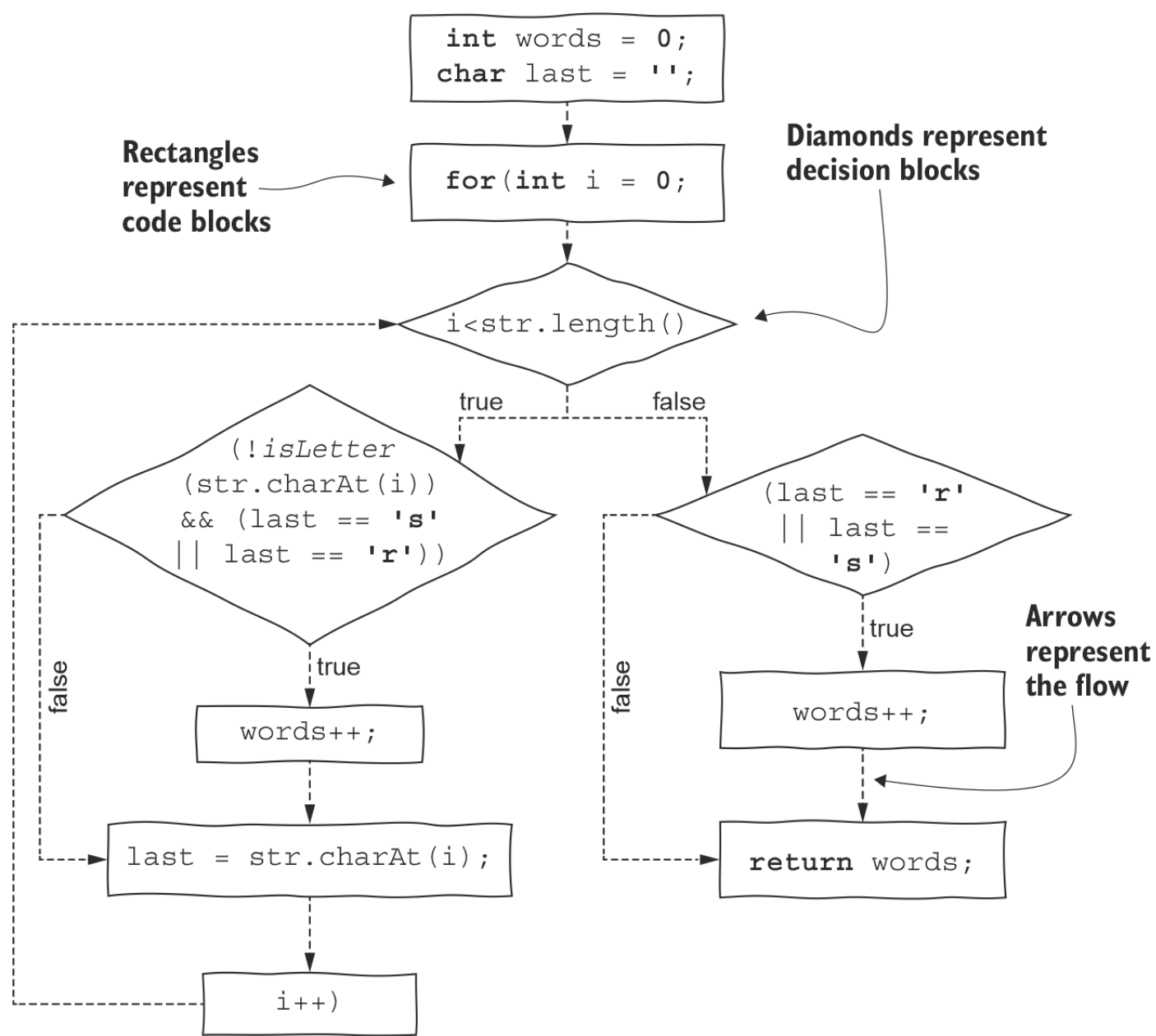
- Podemos cobrir apenas a linha. Basta um caso de teste.
- Podemos cobrir o *if* sendo avaliado como verdadeiro e falso. Dois casos de teste são necessários.
- Podemos explorar cada condição na instrução *if*. Se temos três condições que exigem pelo menos dois testes cada, precisaremos de seis testes.
- Podemos cobrir todos os caminhos de execução possíveis desta instrução. Dado que tem três condições diferentes, isso requer $2 \times 2 \times 2 = 8$ casos de teste.

Critério de Cobertura de Linha

Quando desejamos pelo menos um caso de teste que cubra a linha em teste. Não importando se essa linha contém uma instrução *if* complexa cheia de condições. Se um teste tocar essa linha de alguma forma, podemos contar a linha como coberta.

Critério de Cobertura de Ramificação

- Considera que instruções de desvio (ifs, fors, whiles etc) fazem o programa se comportar de maneiras diferentes, dependendo de como a instrução é avaliada.
- Para uma instrução simples, por exemplo *if (a && b)*, ter um caso de teste que torna o *if* verdadeiro e outro caso de teste que torna o *if* falso é suficiente para considerar toda ramificação coberta.



Cobrir todas as arestas do grafo significa atingir 100% de cobertura de ramificação.

Critério de Cobertura de Condição + Ramificação

Considera não apenas ramificações possíveis, mas também cada condição de cada declaração de ramificação.

Critério de Cobertura de Condição + Ramificação

Por exemplo, o primeiro *if* de `CountWords` contém três condições: `!Character.isLetter(str.charAt(i))`, `last == 's'` e `last == 'r'`.

Devemos criar um conjunto de testes que exercite cada uma dessas condições individuais como verdadeiras e falsas pelo menos uma vez e toda a instrução sendo verdadeira e falsa pelo menos uma vez.

Critério de Cobertura de Condição + Ramificação

- Olhar apenas para as condições (e ignorar como elas são combinadas) pode resultar em conjuntos de testes que não cobrem tudo.
- Imagine um simples *if (A || B)*.
- E dois testes: T1 que torna A verdadeiro e B falso e T2 que torna A falso e B verdadeiro.
- Os dois testes cobrem as duas condições, pois cada condição é exercitada como verdadeira e falsa.
- Mas não cobrem totalmente a ramificação, pois em ambos os testes, a avaliação de toda a instrução *if* é sempre verdadeira.

Critério de Cobertura de Caminhos

- Por esse critério, se cobre todos os caminhos possíveis de execução do programa.
- Embora idealmente este seja o critério mais forte, muitas vezes é impossível ou muito caro de alcançar.
- Em um programa com três condições, teríamos $2^3 = 8$ caminhos para cobrir.
- Em um programa com 10 condições, teríamos $2^{10} = 1024$.

Critério de Cobertura de Caminhos

Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

Condições complexas e o critério de cobertura MC/DC

Cobertura de condição/decisão modificada (MC/DC) é uma boa solução para minimizarmos o número de testes para instruções *if* complexas e longas.

Condições complexas e o critério de cobertura MC/DC

- O critério MC/DC analisa combinações de condições.
- No entanto, em vez de testar todas as combinações possíveis, identificamos as combinações importantes que precisam ser testadas.
- A ideia é que os testes exercitem cada uma das condições de forma que cada uma delas, independentemente das outras condições, afetem o resultado de toda a decisão.
- Cada condição deve influenciar o resultado pelo menos uma vez.

Condições complexas e o critério de cobertura MC/DC

If (A && (B || C)), onde A, B, e C são booleanos.

- Para a condição A:
 - Um caso de teste em que A = verdadeiro (digamos, T1).
 - Outro caso de teste em que A = falso (digamos, T2).
 - T1 e T2 devem levar a resultados diferentes (por exemplo, T1 faz toda a decisão ser verdadeira e T2 faz toda a decisão ser falsa).
 - B e C devem ter os mesmos valores de verdade em T1 e T2.

Condições complexas e o critério de cobertura MC/DC

If (A && (B || C)), onde A, B, e C são booleanos.

- Para a condição B:
 - Um caso de teste em que B = verdadeiro (digamos, T3).
 - Outro caso de teste em que B = falso (digamos, T4).
 - T3 e T4 devem levar a resultados diferentes.
 - A e C em T3 devem ser equivalentes a A e C em T4, ou seja, ter os mesmos valores verdade nos dois testes.

Condições complexas e o critério de cobertura MC/DC

If (A && (B || C)), onde A, B, e C são booleanos.

- Para a condição C:
 - Um caso de teste em que C = verdadeiro (digamos, T5).
 - Outro caso de teste em que C = falso (digamos, T6).
 - T5 e T6 levam a resultados diferentes.
 - As variáveis A e B em T5 devem ser equivalentes a A e B em T6.

Condições complexas e o critério de cobertura MC/DC

- Se as condições assumem apenas resultados binários (por exemplo, verdadeiro ou falso), o número de testes necessários para atingir 100% de cobertura MC/DC é $N + 1$, onde N é o número de condições na decisão.
- $N + 1$ é menor que o número total de combinações possíveis (2^N).
- Assim, para termos 100% MC/DC, devemos criar $N + 1$ casos de teste que, quando combinados, exercitem todas as combinações independentemente das demais.

Condições complexas e o critério de cobertura MC/DC

Como selecionar (mecanicamente) tais casos de teste?

```
if (!Character.isLetter(str.charAt(i)) &&  
    (last == 's' || last == 'r'))
```



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
 T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

!isLetter: {1, 5},



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

!isLetter: {1, 5},



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

!isLetter: {1, 5}, {2, 6},



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

!isLetter: {1, 5}, {2, 6},



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

!isLetter: {1, 5}, {2, 6}, {3, 7}



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

!isLetter: {1, 5}, {2, 6}, {3, 7}

	Caso de Teste	!isLetter	last == s	last == r	Decisão
	T1	true	true	true	true
	T2	true	true	false	true
	T3	true	false	true	true
➡	T4	true	false	false	false
	T5	false	true	true	false
	T6	false	true	false	false
	T7	false	false	true	false
➡	T8	false	false	false	false

!isLetter: {1, 5}, {2, 6}, {3, 7}



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == s:



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == s:



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == s:



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == s: {2, 4}




Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == s: {2, 4}

	Caso de Teste	!isLetter	last == s	last == r	Decisão
	T1	true	true	true	true
	T2	true	true	false	true
	T3	true	false	true	true
	T4	true	false	false	false
➡	T5	false	true	true	false
	T6	false	true	false	false
➡	T7	false	false	true	false
	T8	false	false	false	false

last == s: {2, 4}




Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false


last == s: {2, 4}

	Caso de Teste	!isLetter	last == s	last == r	Decisão
	T1	true	true	true	true
	T2	true	true	false	true
	T3	true	false	true	true
	T4	true	false	false	false
	T5	false	true	true	false
➡	T6	false	true	false	false
	T7	false	false	true	false
➡	T8	false	false	false	false

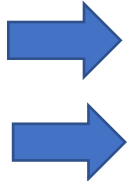
last == s: {2, 4}



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false



last == r:



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == r:




Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == r:



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == r:



Caso de Teste	!isLetter	last == s	last == r	Decisão
T1	true	true	true	true
T2	true	true	false	true
T3	true	false	true	true
T4	true	false	false	false
T5	false	true	true	false
T6	false	true	false	false
T7	false	false	true	false
T8	false	false	false	false

last == r: {3, 4}

Condições complexas e o critério de cobertura MC/DC

!isLetter: {1, 5}, {2, 6}, {3, 7}

last == s: {2, 4}

last == r: {3, 4}

Basta termos um par de casos de testes para cada condição.
Sabemos que necessitamos apenas de $N+1$ casos de testes.

Condições complexas e o critério de cobertura MC/DC

!isLetter: {1, 5}, {2, 6}, {3, 7}

last == s: {2, 4}

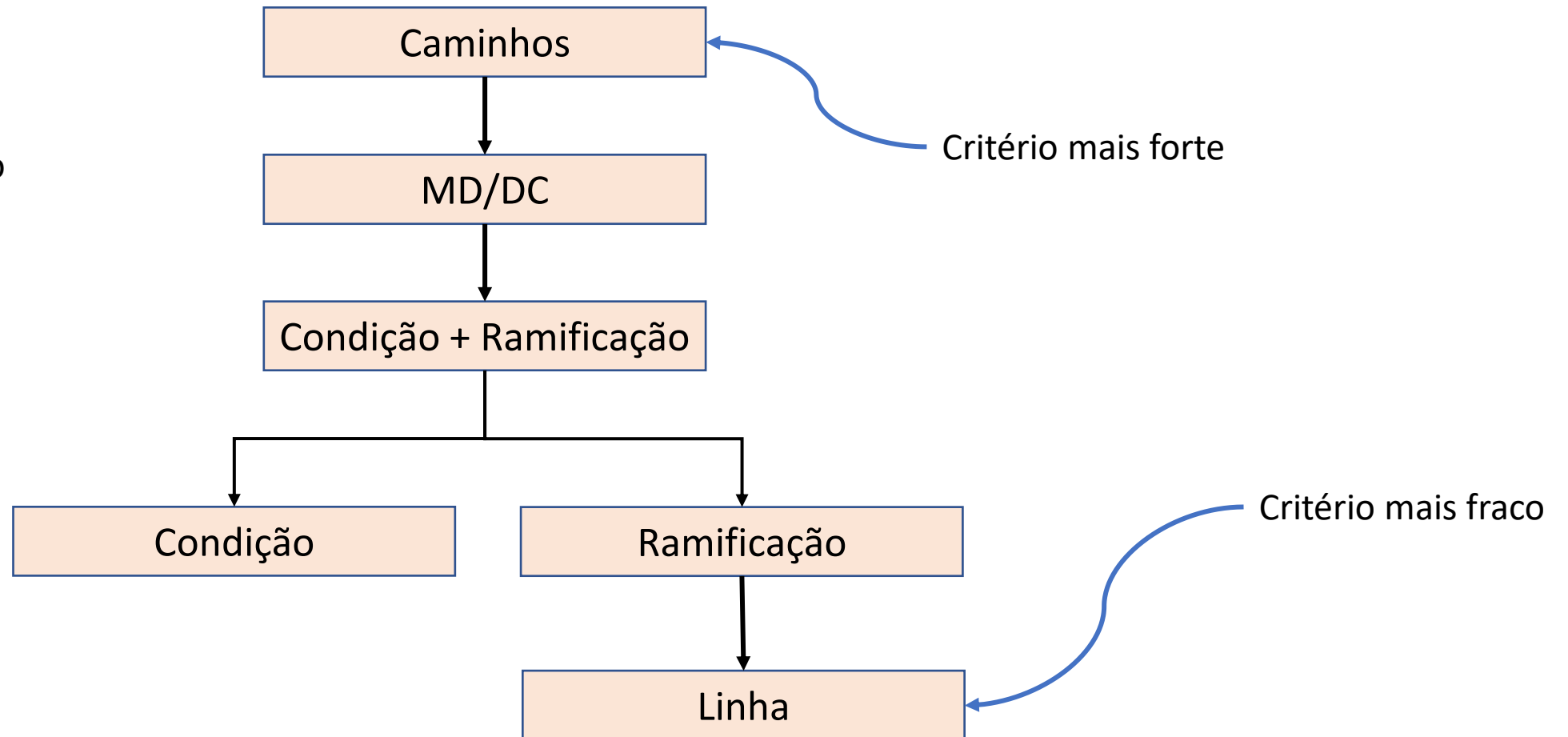
last == r: {3, 4}

Basta termos um par de casos de testes para cada condição.
Sabemos que necessitamos apenas de $N+1$ casos de testes.

Portanto, os testes que precisamos para termos 100% de cobertura MC/DC são T2, T3, T4 e T6.

Relação entre os Critérios de Cobertura

A seta indica que
usando um critério
você atinge o outro



Lidando com Loops

- Quantas vezes o bloco de código dentro de um loop deve ser executado nos testes?
- Imagine um loop `while(true)`.
 - Para sermos rigorosos, deveríamos ter um teste que executasse o código dentro do `while` uma vez, outro que o executasse duas vezes, três vezes e assim por diante. Impossível.
- Um `for(i = 0; i < 10; i++)` com `break` dentro.
 - Dez testes?
- E um loop que não sabemos exatamente a quantidade de iterações?

Lidando com Loops

- Dada a impossibilidade de teste exaustivo, os testadores geralmente se baseiam no “critério de adequação de fronteira de loop”.
- Uma suíte de teste satisfaz esse critério se e somente se para cada loop:
 - Há um caso de teste que exercita o loop zero vezes.
 - Há um caso de teste que exercita o loop uma vez.
 - Há um caso de teste que exercita o loop várias vezes.

Lidando com Loops

- O mais difícil é decidir quantas vezes o loop deve ser executado pelo caso de teste que o exercita várias vezes.
- Essa decisão requer um bom entendimento do programa e dos requisitos.
- Não hesite em criar dois ou mais casos de testes que exercitam o loop várias vezes.