Simulação e Teste de Software (CC8550)

Aula 03 – Técnicas caixa-preta: particionamento e análise de valor limite

Prof. Luciano Rossi

Ciência da Computação
Centro Universitário FEI

1º Semestre de 2026

Tópico 3: Técnicas de Caixa-Preta

Particionamento e Análise de Valor Limite

Recordando do Tópico 1:

- Caixa-preta: teste baseado na **especificação**
- Não conhecemos o código interno
- Perspectiva do usuário/cliente

Neste tópico vamos aprender:

- Como selecionar casos de teste **sistematicamente**
- Particionamento em classes de equivalência
- Análise de valores limite
- Tabelas de decisão
- Property-based testing com Hypothesis

Objetivo: Testar eficientemente sem conhecer o código!

centro universitário FEI

O Problema Chave

O Problema da Exaustividade

Cenário: Função que recebe 3 parâmetros inteiros (32 bits)

Quantos casos de teste existem?

$$2^{32} \times 2^{32} \times 2^{32} = 2^{96} \approx 7.9 \times 10^{28}$$

Se cada teste leva 1 microssegundo:

$\approx$ 2.5 **quadrilhões** de anos para testar tudo!

Conclusão

Teste **exaustivo é impossível.** Precisamos de técnicas para selecionar casos representativos!

Teste Caixa-Preta

Teste Caixa-Preta

Características e quando usar

Características:

- Baseado em **especificação/requisitos**
- Sem acesso ao código-fonte
- Foca no **comportamento externo**
- Valida se sistema faz o que deveria

Quando usar:

- Teste de requisitos funcionais
- Validação de APIs e interfaces
- Teste de aceitação
- Complementar testes caixa-branca

Analogia

Como testar um carro sem abrir o capô: aceleração, freios, direção funcionam?

Classes de Equivalência

Classes de Equivalência

Conceito fundamental

Ideia: Dividir domínio de entrada em **partições** onde todos elementos se comportam similarmente

Classes de Equivalência:

- Elementos da mesma classe produzem resultado equivalente
- Se um elemento da classe passa, todos devem passar
- **Testar um representante de cada classe**

Tipos de classes:

- **Válidas:** Entrada aceita, comportamento esperado
- **Inválidas:** Entrada rejeitada, erro tratado

Benefício

Reduz drasticamente número de testes mantendo boa cobertura!

Exemplo 01

Desconto por Idade

Especificação:

- 0–17 anos: sem desconto
- 18–64 anos: 10% desconto
- 65+ anos: 20% desconto
- Idade negativa ou não-numérica: erro

Classes de Equivalência:

Classe	Intervalo	Tipo
CE1	idade < 0	Inválida
CE2	$0 \leq$ idade ≤ 17	Válida (0%)
CE3	$18 \leq$ idade ≤ 64	Válida (10%)
CE4	idade ≥ 65	Válida (20%)
CE5	não-numérico	Inválida

Casos de teste: um representante por classe (ex: -5, 10, 30, 70, "abc")

Implementando Classes de Equivalência

```python
@pytest.mark.parametrize("idade,desconto_esperado", [
    (-5, None),         # CE1: inválida
    (10, 0),            # CE2: 0-17
    (30, 0.10),         # CE3: 18-64
    (70, 0.20),         # CE4: 65+
], ids=["negativa", "crianca", "adulto", "idoso"])

def test_calcular_desconto_idade(idade, desconto_esperado):
    if desconto_esperado is None:
        with pytest.raises(ValueError):
            calcular_desconto(idade)
    else:
        resultado = calcular_desconto(idade)
    assert resultado == desconto_esperado

```

centro universitário FEI

Critérios para Identificação de Classes de Equivalência

Identificando Classes de Equivalência

Critérios práticos

Para intervalos numéricos:

- Uma classe para cada intervalo válido
- Classes inválidas: abaixo e acima dos limites

Para conjuntos/enumerações:

- Uma classe para cada valor válido
- Uma classe para valores inválidos

Para condições booleanas:

- Uma classe verdadeira
- Uma classe falsa

Para strings:

- Vazia, tamanho normal, tamanho limite, muito longa
- Caracteres válidos/inválidos

centro universitário FEI

Exemplo 02

Validação de Senha

Exercício

Especificação: Um sistema de cadastro deve validar a senha informada pelo usuário. A senha é considerada válida quando possui entre 8 e 20 caracteres, contém pelo menos uma letra maiúscula e pelo menos um dígito numérico, e não apresenta espaços. Senhas fora do intervalo de comprimento, sem maiúscula, sem dígito ou com espaço devem ser rejeitadas. Caso a entrada não seja do tipo textual, o sistema deve lançar um erro.

Tarefa

Identifique as classes de equivalência, monte a tabela e escreva os casos de teste com pytest.

centro universitário FEI

Validação de Senha

Resolução - Classes de Equivalência

Classe	Descrição	Tipo
CE1	comprimento < 8 caracteres	Inválida
CE2	comprimento > 20 caracteres	Inválida
CE3	sem letra maiúscula	Inválida
CE4	sem dígito numérico	Inválida
CE5	contém espaço	Inválida
CE6	entrada não-textual	Inválida
CE7	8–20 chars, maiúscula e dígito	Válida

Representantes:

- CE1: `"Ab1x"` CE2: `"Ab1x"* 6`
- CE3: `"abcdefg1"` CE4: `"Abcdefgh"`
- CE5: `"Ab1 cdef"` CE6: `12345678`
- CE7: `"Senha123"`

centro universitário FEI

Validação de Senha

Resolução - Casos de Teste

```python
@pytest.mark.parametrize("senha,esperado", [
("Ab1x",          False),  # CE1: muito curta
("Ab1x" * 6,      False),  # CE2: muito longa
("abcdefg1",      False),  # CE3: sem maiuscula
("Abcdefgh",      False),  # CE4: sem digito
("Ab1 cdef",      False),  # CE5: tem espaco
("Senha123",      True),   # CE7: valida
], ids=["curta","longa","sem_mai","sem_dig","espaco","valida"])
def test_validar_senha(senha, esperado):
    assert validar_senha(senha) == esperado

def test_validar_senha_tipo_invalido():
    with pytest.raises(TypeError):  # CE6
        validar_senha(12345678)

```

centro universitário FEI

Validação de Senha

Resolução - Implementação

```python
def validar_senha(senha):
    if not isinstance(senha, str):
        raise TypeError("Senha deve ser texto")
    if len(senha) < 8 or len(senha) > 20:
        return False
    if not any(c.isupper() for c in senha):
        return False
    if not any(c.isdigit() for c in senha):
        return False
    if " " in senha:
        return False
    return True

```

Observação

7 classes identificadas → 7 casos de teste. Cobertura completa sem testes redundantes!

Análise de Valor Limite

Análise de Valor Limite

Bugs nas fronteiras

Observação empírica:
Defeitos concentram-se nas **fronteiras** das classes de equivalência

Por quê?

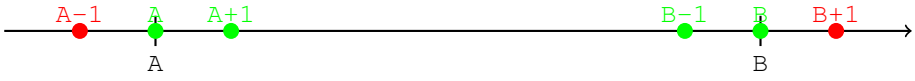
- Erros de comparação (< vs <=)
- Erros off-by-one
- Limites mal definidos
- Casos extremos não considerados

Princípio

Além de testar representantes, teste os **valores limite** de cada classe!

Regra dos Valores Limite

Para um intervalo **[A, B]** válido, testar:



6 valores: A−1, A, A+1, B−1, B, B+1

Exemplo 03

Sistema de Notas

Especificação: Nota entre 0.0 e 10.0 (inclusive)

Classes de Equivalência:

- CE1: nota < 0.0 (inválida)
- CE2: $0.0 \leq$ nota ≤ 10.0 (válida)
- CE3: nota > 10.0 (inválida)

Valores Limite:

Valor	Classe	Status
−0.1	CE1	Inválido
0.0	CE2	**Válido (mínimo)**
0.1	CE2	Válido
9.9	CE2	Válido
10.0	CE2	**Válido (máximo)**
10.1	CE3	Inválido

centro universitário FEI

Implementando Valores Limite

```python
@pytest.mark.parametrize("nota,valida", [
    (-0.1, False),    # Abaixo do mínimo
    (0.0, True),      # Mínimo
    (0.1, True),      # Logo acima mínimo
    (5.0, True),      # Meio do intervalo
    (9.9, True),      # Logo abaixo máximo
    (10.0, True),     # Máximo
    (10.1, False),    # Acima do máximo
], ids=[
"abaixo_min", "min", "acima_min", "meio", "abaixo_max", "max", "acima_max"
])

def test_validar_nota(nota, valida):
    if valida:
        assert validar_nota(nota) == True
    else:
        with pytest.raises(ValueError):
            validar_nota(nota)

```

centro universitário FEI

Exemplo 04

Desconto por Quantidade

Exercício

Especificação:

Um sistema de vendas calcula descontos com base na quantidade de itens do pedido. Pedidos com 1 a 5 itens não recebem desconto. Pedidos com 6 a 20 itens recebem 10% de desconto. Pedidos com 21 ou mais itens recebem 20% de desconto. Quantidade zero ou negativa deve lançar um erro, assim como quantidade não-inteira.

Tarefa

Identifique as classes de equivalência, liste os valores limite de cada fronteira e escreva os casos de teste com pytest.

Desconto por Quantidade

Resolução - Classes de Equivalência

Classe	Descrição	Tipo
CE1	quantidade ≤ 0	Inválida
CE2	não-inteiro	Inválida
CE3	$1 \leq$ quantidade ≤ 5	Válida (0%)
CE4	$6 \leq$ quantidade ≤ 20	Válida (10%)
CE5	quantidade ≥ 21	Válida (20%)

Fronteiras identificadas:

- Fronteira inferior: entre CE1 e CE3 → valores 0, 1, 2
- Fronteira interna 1: entre CE3 e CE4 → valores 4, 5, 6, 7
- Fronteira interna 2: entre CE4 e CE5 → valores 19, 20, 21, 22

centro universitário FEI

Desconto por Quantidade

Resolução - Tabela de Valores Limite

Valor	Classe	Status	Desconto
0	CE1	Inválido	erro
1	CE3	**Mínimo CE3**	0%
2	CE3	Válido	0%
4	CE3	Válido	0%
5	CE3	**Máximo CE3**	0%
6	CE4	**Mínimo CE4**	10%
7	CE4	Válido	10%
19	CE4	Válido	10%
20	CE4	**Máximo CE4**	10%
21	CE5	**Mínimo CE5**	20%
22	CE5	Válido	20%

centro universitário FEI

Desconto por Quantidade

Resolução - Casos de Teste (1/2)

```
@pytest.mark.parametrize("qtd,esperado", [
(0,    None),  # CE1: invalido
(1,    0.00),  # minimo CE3
(2,    0.00),  # interior CE3
(4,    0.00),  # proximo ao limite CE3
(5,    0.00),  # maximo CE3
(6,    0.10),  # minimo CE4
(7,    0.10),  # interior CE4
(19,   0.10),  # proximo ao limite CE4
(20,   0.10),  # maximo CE4
(21,   0.20),  # minimo CE5
(22,   0.20),  # interior CE5
])

```

centro universitário FEI

Desconto por Quantidade

Resolução - Casos de Teste (2/2)

```python
def test_calcular_desconto(qtd, esperado):
    if esperado is None:
        with pytest.raises(ValueError):
            calcular_desconto(qtd)
    else:
        assert calcular_desconto(qtd) == esperado

def test_desconto_tipo_invalido():
    with pytest.raises(TypeError):  # CE2
        calcular_desconto(3.5)

```

centro universitário FEI

Desconto por Quantidade

Resolução - Implementação

```python
def calcular_desconto(qtd):
    if not isinstance(qtd, int):
        raise TypeError("Quantidade deve ser inteira")
    if qtd <= 0:
        raise ValueError("Quantidade deve ser positiva")
    if qtd <= 5:
        return 0.00
    if qtd <= 20:
        return 0.10
    return 0.20

```

Observação

Duas fronteiras internas (5→6 e 20→21) exigem atenção redobrada aos operadores < vs <= -- erros clássicos de off-by-one!

Combinando Classes e Limites

Estratégia completa:

1. **Identificar classes de equivalência**

   Válidas e inválidas

2. **Para cada classe válida:**
   - Testar um valor representativo (meio)
   - Testar valores limite (fronteiras)

3. **Para cada classe inválida:**
   - Testar valores que violam restrições
   - Testar valores limite adjacentes

Resultado

Cobertura abrangente com número reduzido de testes!

centro universitário FEI

Tabela de Decisão

Tabela de Decisão

Lógica condicional complexa

Quando usar:

- Múltiplas condições independentes
- Lógica de negócio com várias regras
- Comportamento depende de combinações

Estrutura:

- **Condições:** Entradas/critérios (S/N ou V/F)
- **Ações:** Resultados/comportamentos esperados
- **Regras:** Combinações de condições → ações

Benefício

Garante que todas combinações relevantes foram consideradas!

Exemplo 05

Aprovação de Empréstimo

Especificação:

- Aprovar SE: idade $\geq$ 18 E renda $\geq$ R$ 3000 E score $\geq$ 600
- Rejeitar caso contrário

Tabela de Decisão:

Condição	R1	R2	R3	R4	R5	R6	R7	R8
Idade $\geq$ 18?	S	S	S	S	N	N	N	N
Renda $\geq$ 3000?	S	S	N	N	S	S	N	N
Score $\geq$ 600?	S	N	S	N	S	N	S	N
Ação								
Aprovar	X							
Rejeitar		X	X	X	X	X	X	X

8 regras = 2^3 combinações (todas relevantes)

Implementando Tabela de Decisão

```python
@pytest.mark.parametrize("idade,renda,score,aprovado", [
    (25, 4000, 700, True),   # R1: SSS -> Aprovar
    (25, 4000, 500, False),  # R2: SSN -> Rejeitar
    (25, 2500, 700, False),  # R3: SNS -> Rejeitar
    (25, 2500, 500, False),  # R4: SNN -> Rejeitar
    (17, 4000, 700, False),  # R5: NSS -> Rejeitar
    (17, 4000, 500, False),  # R6: NSN -> Rejeitar
    (17, 2500, 700, False),  # R7: NNS -> Rejeitar
    (17, 2500, 500, False),  # R8: NNN -> Rejeitar
])

def test_aprovar_emprestimo(idade, renda, score, aprovado):
    resultado = aprovar_emprestimo(idade, renda, score)
    assert resultado == aprovado

```

centro universitário FEI

Redução de Tabelas

Don't Care (X)

Situação: Algumas condições não importam para certas ações

Exemplo Empréstimo Simplificado:

Condição	R1	R2	R3	R4
Idade $\geq$ 18?	S	S	S	N
Renda $\geq$ 3000?	S	S	N	X
Score $\geq$ 600?	S	N	X	X
Aprovar	X			
Rejeitar		X	X	X

R4: Se idade < 18, renda e score não importam (X = don't care)

R3: Se renda < 3000, score não importa

Redução: 8 regras $\rightarrow$ 4 regras

Pairwise Testing

Teste Combinatorial

O problema da explosão combinatória

Cenário: Sistema com 10 parâmetros booleanos (S/N)

Combinações totais: $2^{10} = 1024$ casos de teste

Observação empírica:

- Maioria dos bugs envolve interação de **2 parâmetros**
- Poucos bugs envolvem 3+ parâmetros

Pairwise Testing (Teste aos Pares)

Garante que **todas combinações 2-a-2** são testadas, reduzindo drasticamente o número de casos.

centro universitário FEI

Exemplo 06

Pairwise Testing

Parâmetros:

- SO: Windows, Linux, Mac
- Browser: Chrome, Firefox
- Idioma: PT, EN

Combinações totais: $3 \times 2 \times 2 = 12$

Pairwise (exemplo):

SO	Browser	Idioma
Windows	Chrome	PT
Windows	Firefox	EN
Linux	Chrome	EN
Linux	Firefox	PT
Mac	Chrome	PT
Mac	Firefox	EN

6 casos cobrem todos os pares! (50% redução)

centro universitário FEI

Property-Based Testing

Property-Based Testing

Nova abordagem

Testes tradicionais:

- Casos específicos: "2 + 3 = 5"
- Você define entradas e saídas
- Limitado aos casos que você pensou

Property-Based Testing:

- Define **propriedades** que devem sempre ser verdadeiras
- Ferramenta **gera casos automaticamente**
- Testa centenas/milhares de combinações
- Framework Python: **Hypothesis**

Vantagem

Descobre casos que você nunca pensaria em testar manualmente!

centro universitário FEI

Propriedades vs Exemplos

Teste tradicional (exemplo):

- `assert somar(2, 3) == 5`
- `assert somar(0, 5) == 5`
- `assert somar(-2, 2) == 0`

Property-based (propriedade):

- **Comutatividade:** `somar(a, b) == somar(b, a)` para qualquer `a, b`
- **Identidade:** `somar(x, 0) == x` para qualquer `x`
- **Associatividade:** `somar(somar(a,b), c) == somar(a, somar(b,c))`

Importante

Propriedades são **universais**, exemplos são **específicos**!

Hypothesis: Instalação e Uso Básico

Instalação:

```
$ pip install hypothesis
```

Exemplo simples:

```python
from hypothesis import given
import hypothesis.strategies as st

@given(st.integers())
def test_reverter_duas_vezes(x):
    # Reverter uma lista duas vezes = original
    lista = [x]
    assert lista == list(reversed(list(reversed(lista))))

```

Hypothesis gera automaticamente vários valores de x e testa!

centro universitário FEI

Strategies: Gerando Dados

Hypothesis oferece **strategies** para gerar diversos tipos:

```python
import hypothesis.strategies as st

st.integers()               # Inteiros
st.floats()                 # Floats
st.text()                   # Strings
st.booleans()               # Booleanos
st.lists(st.integers())     # Listas de inteiros
st.tuples(st.text(), st.integers())  # Tuplas

# Com restrições
st.integers(min_value=0, max_value=100)
st.text(min_size=1, max_size=10)
st.lists(st.integers(), min_size=1)

```

centro universitário FEI

Exemplo 07

Propriedade de Ordenação

```python
from hypothesis import given
import hypothesis.strategies as st

@given(st.lists(st.integers()))
def test_ordenar_propriedades(lista):
    ordenada = sorted(lista)

    # Propriedade 1: tamanho não muda
    assert len(ordenada) == len(lista)

    # Propriedade 2: todos elementos presentes
    assert set(ordenada) == set(lista)

    # Propriedade 3: está ordenada
    for i in range(len(ordenada) - 1):
        assert ordenada[i] <= ordenada[i+1]

```

Hypothesis testa com: listas vazias, grandes, duplicados, negativos...

centro universitário FEI

Exemplo 08

Descobrindo Bugs

Função com bug sutil:

```python
def calcular_media(numeros):
    return sum(numeros) / len(numeros)

```

Property-based test:

```python
@given(st.lists(st.floats(), min_size=1))
def test_media_propriedades(numeros):
    media = calcular_media(numeros)

    # Média deve estar entre min e max
    assert min(numeros) <= media <= max(numeros)

```

Hypothesis descobre:

Falha com lista vazia! Erro: ZeroDivisionError

centro universitário FEI

Simulação e Teste de Software (CC8550)

Aula 03 – Técnicas caixa-preta: particionamento e análise de valor limite

Prof. Luciano Rossi

Ciência da Computação
Centro Universitário FEI

1º Semestre de 2026