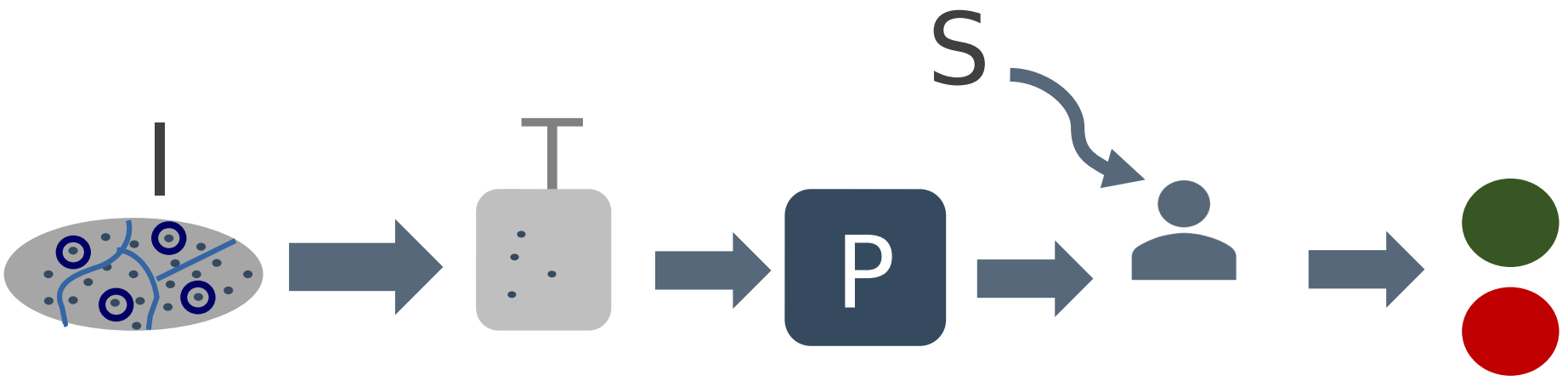
V V & T

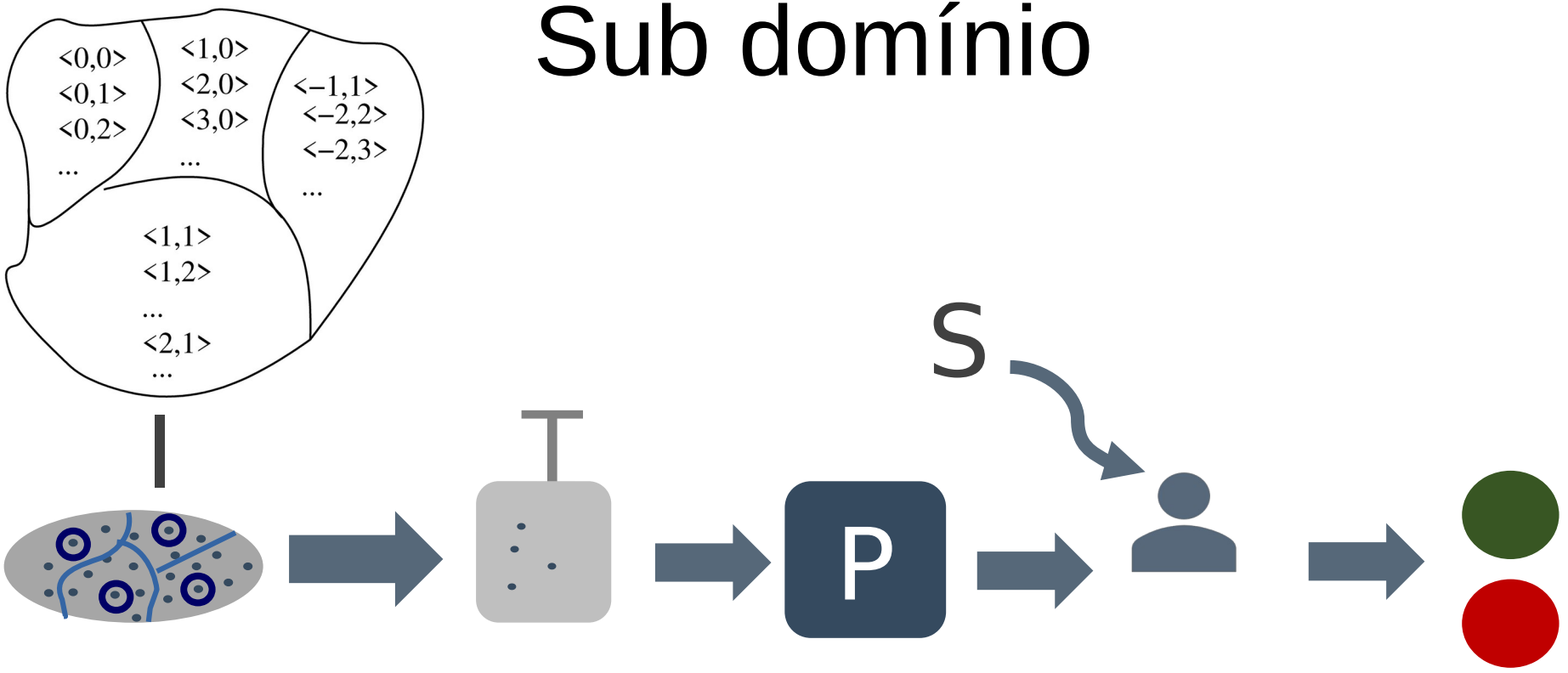
Teste funcional

Marcio Delamaro

Sub domínio



Sub domínio



Teste funcional

- Teste baseado na especificação
- Identificar as entradas
  - Tipos e valores (domínio)
- Identificar classes de equivalência
  - Válidas e inválidas
- Identificar valores para exercitar as classes
  - Limites
- Criar os casos de teste

Teste funcional

- Teste baseado na especificação
- Identificar as entradas e saídas
  - Tipos e valores (domínio)
- Identificar classes de equivalência
  - Válidas e inválidas
- Identificar valores para exercitar as classes
  - Limites
- Criar os casos de teste

Teste funcional

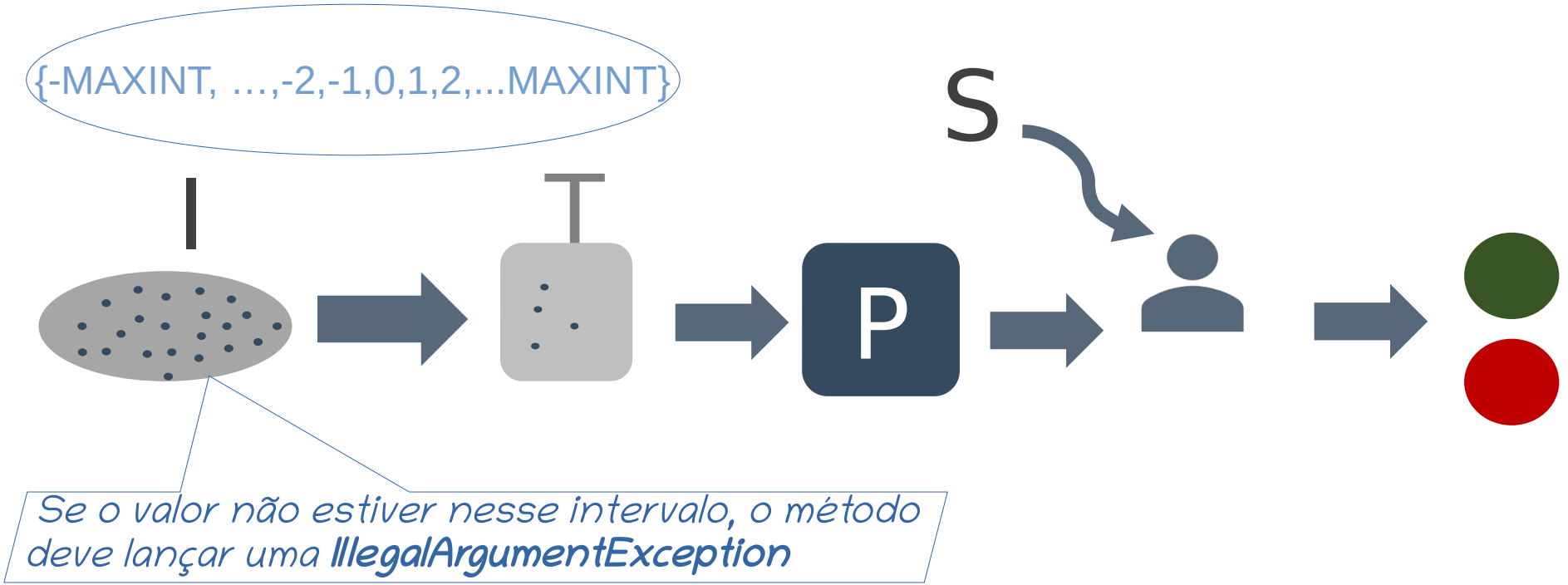
- Teste baseado na especificação
- Identificar as entradas e saídas
  - Tipos e valores (domínio)
- Identificar classes de equivalência
  - ~~Válidas e inválidas~~ Normais e excepcionais
- Identificar valores para exercitar as classes
  - Limites
- Criar os casos de teste

Exemplo

Escreva um método para verificar se um determinado ano é bissexto.

- A entrada deve ser um número inteiro entre 1 e 9999.
- Se o valor não estiver nesse intervalo, o método deve lançar uma ***IllegalArgumentException***
- A saída deve ser true ou false.

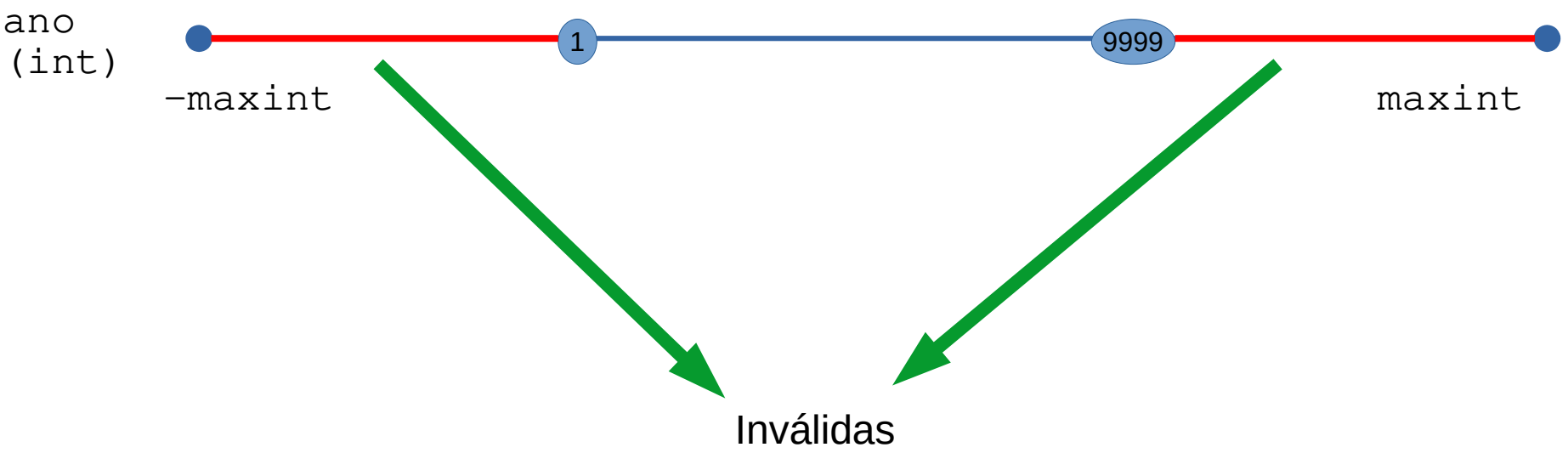
Identificar entradas



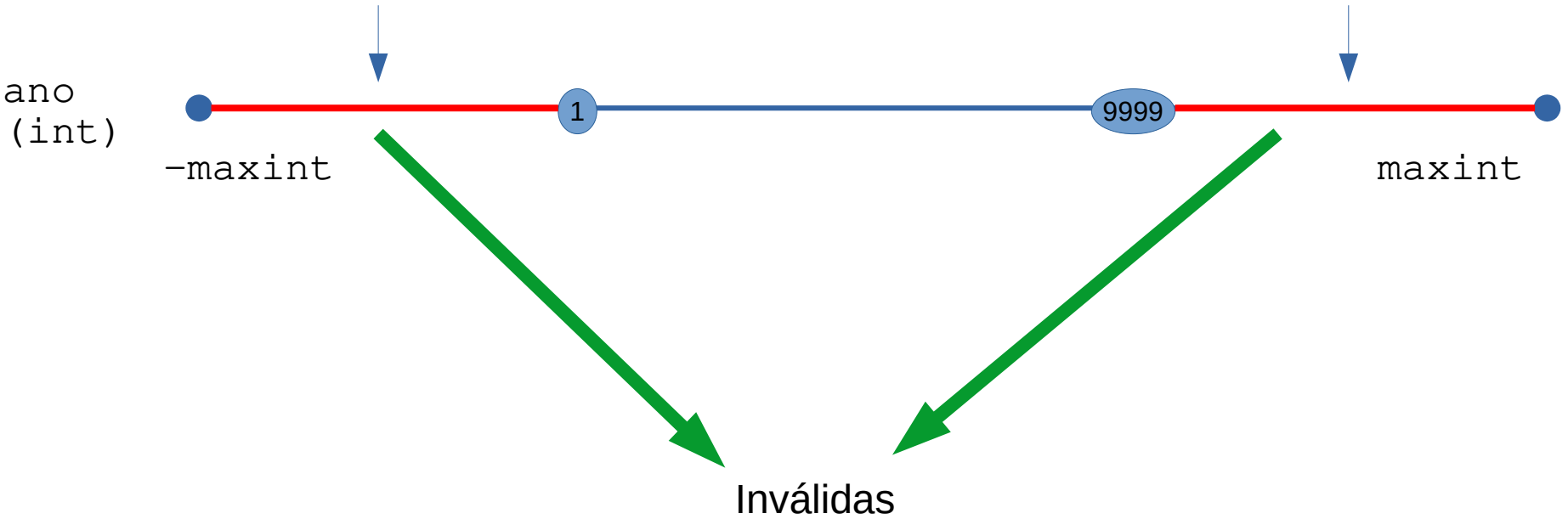
Classes de equivalência

Classes de equivalência

Classes de equivalência

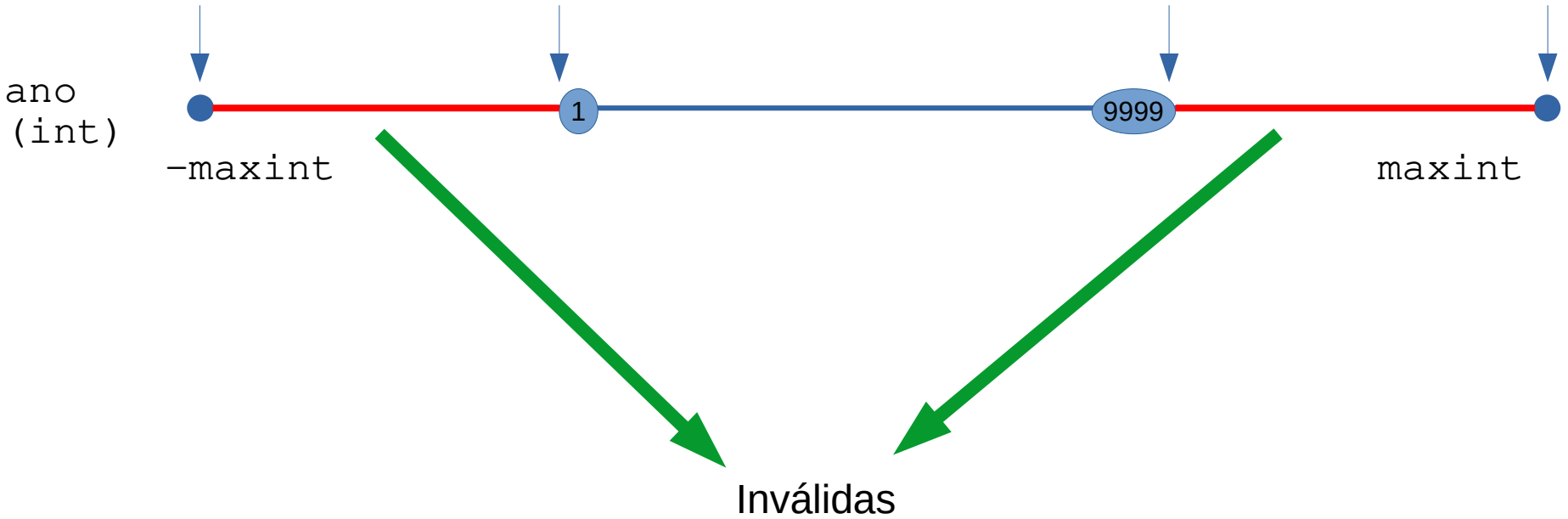


Selecionar valores



Selecionar valores

{-maxint, 0, 10.000, maxint}



Selecionar valores

{-maxint, 0, 10.000, maxint, 1, 9999}

Selecionar valores (saída)

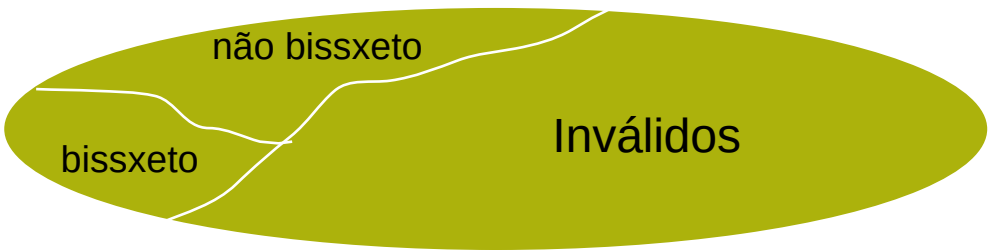
{-maxint, 0, 10.000, maxint, 1, 9999}

Selecionar valores (saída)

{-maxint, 0, 10.000, maxint, 1, 9999}

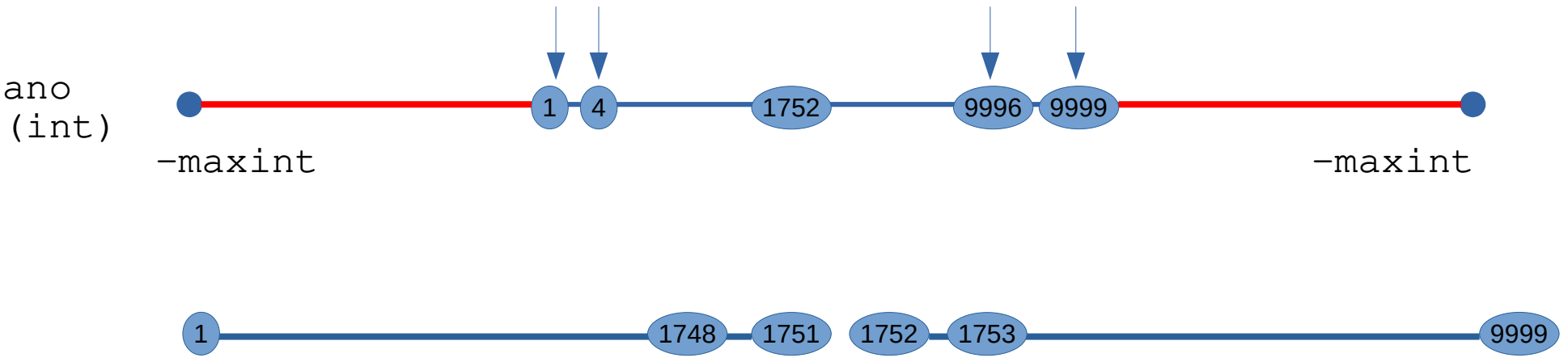
Selecionar valores (saída)

{-maxint, 0, 10.000, maxint, 1, 9999, 4, 9996}



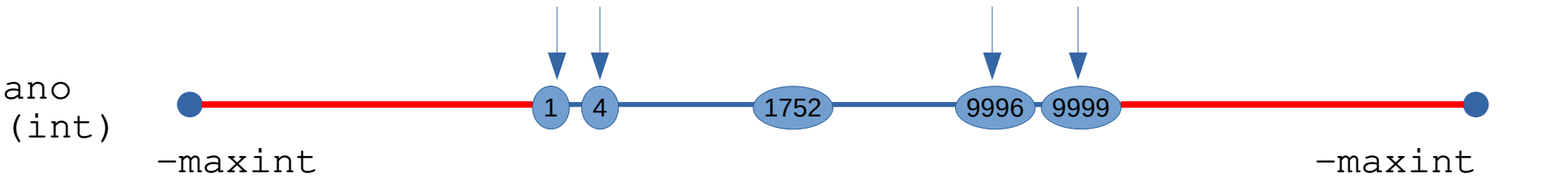
Dá pra melhorar

{-maxint, 0, 10.000, maxint, 1, 9999, 4, 9996, 1751, 1748, 1752, 1753}



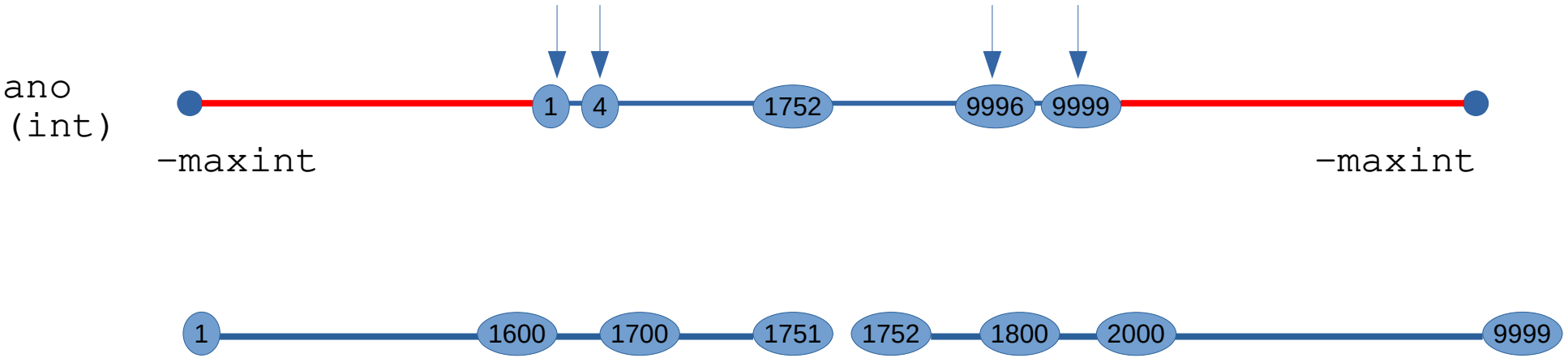
Dá pra melhorar

{-maxint, 0, 10.000, maxint, 1, 9999, 4, 9996, 1751, 1748, 1752, 1753}



E os 100 e os 400?

{-maxint, 0, 10.000, maxint, 1, 9999, 4, 9996, 1751, 1748, 1752, 1753, 1600, 1700, 1800, 2000}



E o código?

- Preciso do código para gerar os casos de teste?
  - Não preciso, basta a especificação
- Posso usar o código ao gerar os casos de teste?
  - Não, não pode

E o código?

- Preciso do código para gerar os casos de teste?
  - Não preciso, basta a especificação
- Posso usar o código ao gerar os casos de teste?
  - ~~Não, não pode~~

Alguma dica?

É difícil apresentar um "livro de receitas", pois [o teste funcional] requer um grau de criatividade e uma certa especialização em relação ao problema em questão. (Portanto, como muitos outros aspectos do teste, é mais um estado de espírito do que qualquer outra coisa.)

Alguma dica?

- In addition, use your ingenuity to search for other boundary conditions.
- Null, array tamanho 0, tamanho 1
- Ordem dos elementos
- Combinação de valores

Entendendo os requisitos, entradas e saídas

- Independentemente de como seus requisitos são escritos (ou mesmo se estiverem apenas em sua mente), eles incluem três partes.
- O que o programa/método deve fazer: suas regras de negócio.
- O programa recebe dados como entradas. Elas são parte fundamental do nosso raciocínio, pois é através deles que podemos testar os diferentes casos.
- O raciocínio sobre a saída nos ajudará a entender melhor o que o programa faz e como as entradas são convertidas na saída esperada.

Exemplo 2

Método: substringsBetween()

Pesquisa uma string por substrings delimitadas por uma tag inicial e final, retornando todas as substrings correspondentes em uma matriz.

str – A string contendo as substrings. Nulo retorna nulo; uma string vazia retorna outra string vazia.

open – A string que identifica o início da substring. Uma string vazia retorna null.

close – A string que identifica o final da substring. Uma string vazia retorna null.

O programa retorna um array de strings de substrings, ou null se não houver correspondência.

Exemplo: se str = "axcaycazc", open = "a" e close = "c", a saída será um array contendo ["x", "y", "z"]. Este é o caso porque a substring “a<something>c” aparece três vezes na string original: a primeira contém "x" no meio, a segunda "y" e a última "z".

Explore o que o programa faz para várias entradas

- Casos simples para entender o que o programa faz (TDD?)
- Por exemplo
  - `substringsBetween("abcd", "a", "d") → { “bc”}`
  - `substringsBetween("abcdabcdab", "a", "d") → {“bc”, “bc”}`
  - `substringsBetween("aabcddaabfddaab", "aa", "dd") → { "bc", "bf" }`

Identificar classes de equivalência

- Explorar classes de entradas individualmente
- Combinação dos valores de entrada
  - Que combinações posso tentar entre as tags de abertura e fechamento?
- As diferentes classes de saída esperadas deste programa
  - Ele retorna arrays?
  - Ele pode retornar um array vazio?
  - Ele pode retornar `nulls`?

Parâmetros individuais

- `str`:
  - null, vazia, tamanho 1, tamanho > 1
- `open`:
  - null, vazia, tamanho 1, tamanho > 1
- `close`:
  - null, vazia, tamanho 1, tamanho > 1

Combinação de valores

- `str` não contém nem a tag de abertura nem a tag de fechamento.
- `str` contém a tag de abertura, mas não a tag de fechamento.
- `str` contém a tag de fechamento, mas não a tag de abertura.
- `str` contém as tags de abertura e fechamento.
- `str` contém as tags de abertura e fechamento várias vezes.

Combinação de valores

- `str` não contém nem a tag de abertura nem a tag de fechamento.
- `str` contém a tag ão a tag de fechamento.
- `str` contém a tag s não a tag de abertura.
- `str` contém as tags echamento.
- `str` contém as tags de abertura e fechamento várias vezes.

Todos esses casos de teste não são diretamente absorvidos da especificação.
Eles vêm, particularmente, da sua experiência.

Saídas

- O array pode ser:
  - null, vazio, um elemento, vários elementos
  - `null, {}, {“abc”}, {“abc”, “d”}`

- Cada elemento pode ser:
  - vazio, tamanho 1, tamanho > 1
  - `{“abc”, “d”, “”}`

Limites

- Nesse caso, os limites se referem, principalmente ao tamanho dos elementos identificados
- Para cada entrada/ saída transição de vazio/não vazio
- Para a combinação str-open-close
  - str contém tags de abertura e fechamento, sem caracteres entre elas.
  - str contém tags de abertura e fechamento, com caracteres entre elas.

Elaborar casos de teste

- Escolher valores para entradas de forma a exercitar as classes de equivalência
  - ter uma noção do que é classe “válida” ou “inválida”
  - `substringsBetween(null, null, null)`
- Um teste para cada caso excepcional
- Casos normais combinados de forma racional

Casos excepcionais

- T1: str é nulo. `substringsBetween(null, "a", "b")`
- T2: str está vazio. `substringsBetween("", "a", "b"));`
- T3: open é nulo. `substringsBetween("abc", null, "b")`
- T4: open é vazio. `substringsBetween("abc", "", "b")`
- T5: close é nulo. `substringsBetween("abc", "a", null)`
- T6: close é vazio. `substringsBetween("abc", "a", "")`

Condições normais

- tamanho str = 1

```
    assertNull(substringsBetween("a", "a", "b"));
    assertNull(substringsBetween("a", "b", "a"));
    assertNull(substringsBetween("a", "b", "b"));
    assertNull(substringsBetween("a", "a", "a"));
```

- tamanho open e close = 1

```
  assertNull(substringsBetween("abc", "x", "y"));
  assertNull(substringsBetween("abc", "a", "y"));
  assertNull(substringsBetween("abc", "x", "c"));
  assertArrayEquals(new String[] {"b"},substringsBetween("abc", "a", "c"));
  assertArrayEquals(new String[] {"b", "b"},substringsBetween("abcabc", "a", "c"));
```

Condiçoes normais (2)

- tamanho str, close e open > 1

```
assertNull(substringsBetween("aabcc", "xx", "yy"));
assertNull(substringsBetween("aabcc", "aa", "yy"));
assertNull(substringsBetween("aabcc", "xx", "cc"));
assertArrayEquals(new String[] {"bb"},
        substringsBetween("aabbcc", "aa", "cc"));
assertArrayEquals(new String[] {"bb", "ee"},
        substringsBetween("aabbccaaeecc", "aa", "cc"));
```

- limite: saídas com tamanho 0

```
assertArrayEquals(new String[] {""},
        substringsBetween("aabb", "aa", "bb"));
```

Use sua criatividade

- Caracteres estranhos atrapalham?

```
assertArrayEquals(new String[] {"b", "byt byr"},
        substringsBetween("abcabyt byrc", "a", "c"));
```

```
assertArrayEquals(new String[] {"bb dd"},
        substringsBetween("a abb ddc ca abbcc", "a a", "c c"));
```

Código

```java
public static String[] substringsBetween(final String str,
 final String open, final String close) {

  if (str == null || open == null || close == null) {
    return null;
  }
  if (open.equals("") || close.equals("")) {
    return null;
  }

  int strLen = str.length();
  if (strLen == 0) {
    return EMPTY_STRING_ARRAY;
  }

  int closeLen = close.length();
  int openLen = open.length();
  List<String> list = new ArrayList<>();
  int pos = 0;

  while (pos < strLen - closeLen) {
    int start = str.indexOf(open, pos);

    if (start < 0) {
      break;
    }

    start += openLen;
    int end = str.indexOf(close, start);
    if (end < 0) {
      break;
    }

    list.add(str.substring(start, end));
    pos = end + closeLen;

  }

  if (list.isEmpty()) {
    return null;
  }

  return list.toArray(EMPTY_STRING_ARRAY);
}
```

Esse é o melhor conjunto de teste que você vai desenvolver

Paradoxo do pesticida

- Para matar as pragas de uma plantação, você uma um pesticida
- Ele vai matando as pragas até que uma hora você precisa trocar o veneno

Treinando

O método recebe dois números, `left` e `right` (cada um representado como uma lista de dígitos), adiciona-os e retorna o resultado como uma lista de dígitos.

Cada elemento nas listas de dígitos da esquerda e da direita deve ser um número de [0-9]. Uma `IllegalArgumentException` será lançada se essa pré-condição não for válida.

left – Uma lista contendo o número esquerdo. Nulo retorna nulo; vazio significa 0.

right – Uma lista contendo o número da direita. Nulo retorna nulo; vazio significa 0.

O programa retorna a soma de esquerda e direita como uma lista de dígitos.

Crie uma implementação e um conjunto de teste funcional para esse método.