Universidade de Lisboa
Faculdade de Ciências
2016/2017

Verificação e Validação de Software

Relatório - Projecto 1

Mestrado em Informática

Grupo 3
Tiago Moucho, 43536
Inês de Matos, 43538

Índice

1. Faltas **2**

2. Testes Baseados na Partição do Espaço de Entrada **5**

2.1 Parâmetros e Estados Abstractos 5

2.2 Características e Partições 6

2.3 Blocos das Características 7

2.4 Representações de Blocos 7

2.5 Escolha Básica Múltipla 9

2.6 Tabela Resumo 12

2.7 Requisitos Inviáveis 13

2.8 Testes em JUnit 13

2.9 Correção de Faltas 13

3. Cobertura por Fluxo de Controle **15**

3.1 Grafo e Caminhos 15

3.2 Tabela de Resumo 18

3.3 Requisitos Inviáveis 20

4. Cobertura por Fluxo de Dados **21**

4.1 DU-Pairs e DU-Path 21

4.2 Casos de Teste 25

4.3 Programação de Testes 26

4.4 Requisitos Inviáveis 26

1. Faltas

Nesta secção são identificadas as faltas relativas à classe BoundedQueue. Primeiro é indicado a falta respetiva e de seguida são respondidas às questões:

a) Quando é que ocorre uma falta e não um erro
b) Quando é que ocorre uma falta e um erro, mas não existe falha
c) Quando é que ocorre uma falha

- **toString**

  A StringBuilder *sb* é iniciada com "[" mas não é finalizada com o respectivo carácter ("]").

  a) Não existe nenhuma situação em que seja falta e não erro.
  b) Atinge a falta e erro quando se usa o toString para imprimir a fila. Como o StringBuilder é iniciado com "[", a representação da fila fica algo como [1,2,3, e o esperado seria [1,2,3].
  c) Caso se percorra o output até ao carácter "]" para obter o conteúdo que está entre "[ ]" , este vai originar uma falha visto que o carácter "]" não existe.

- **size**

  O retorno deste método, deveria ser o próprio *size* e não *size + 1*, induzindo em erro o verdadeiro valor.

  a) Não existe nenhuma situação em que seja falta e não erro.
  b) Numa situação em que ocorre erro, por exemplo quando esta é criada o size é iniciado a zero, mas a função size() retorna sempre +1 em relação à variável, o que não é correto.
  c) Caso a fila esteja cheia, por exemplo com 4 elementos, o *size()* vai dizer que o tamanho da fila é 5. Ao usar o resultado deste método para aplicar noutras situações, como por exemplo um *for* para percorrer os elementos da fila, no index = 4 o programa vai tentar buscar um elemento inexistente (o array terá os index [0,1,2,3]). Visto que a variável *size* é private, o uso deste método é necessário para obter este valor.

- **enqueue**
  O valor *end* é atualizado antes de ser inserir o elemento na fila. Isto é, é feito o *nextNumModLen* antes do *elem.set,* o que leva a inserir o elemento uma posição à frente do que deveria ser.
  - **a)** Caso a capacidade da fila for 1, a inserção do elemento será feita na única posição que existe, sendo que tanto a *head* como a *tail* estão na mesma posição.
  - **b)** Caso a capacidade seja > 1, os elementos vão ser sempre inseridos numa posição à frente da suposta.
  - **c)** Ocorre falha se tentar aceder ao index 0 para efetuar uma operação sobre o elemento(após ter inserido pelo menos um elemento (por exemplo, 100)). O valor dado ser null em vez do suposto primeiro elemento (100) o que irá provocar uma falha na operação.

- **dequeue**
  O size não é decrementado quando um elemento é introduzido.
  - **a)** Não existe nenhuma situação considerada que seja falta e não erro.
  - **b)** Caso a capacidade da fila for 1, a eliminação do elemento da fila/movimento da *head* será na única posição existente (que corresponde, também, à posição da *tail*). Como não existe um set/remoção aos elementos, sempre que o toString é chamado irá sempre imprimir o elemento que supostamente deveria ter sido removido. Adicionalmente, como o size não é decrementado, a pré-condição de fila fazia não é satisfeita e assim irá permitir aceder à cabeça da fila, o que não seria suposto acontecer.
  - **c)** Como o size não é decrementado, caso se queira aceder a uma posição da fila que não existe (por exemplo o *size* ser 6 e na fila estarem 3 elementos, as posições inexistentes seriam), irá originar uma falha de leitura da fila.

- **addAll**
  Excepção não implementada.
  A condição está ao contrário. A condição deveria ser *>=* e não *<=* ou deveria se trocar os blocos de implementação entre o *if* e o *else*
  - **a)** Não existe nenhuma situação considerada que seja falta e não erro.
  - **b)** Basta o size da coleção ser menor ou igual que o espaço disponível para dar um erro, ou seja, seria suposto introduzir os elementos e não mandar uma exceção.
  - **c)** Não existe nenhuma situação considerada por nós que dê falha.

- **head**
  O start é atualizado após obter a cabeça da fila, o que não deveria acontecer, visto que este método apenas fornece a cabeça atual, o que leva a saltar uma posição na fila. Além disso, esta variável já é devidamente atualizada quando se pretende remover o elemento na cabeça.

a) Caso o tamanho da fila for 1 a posição da cabeça irá se manter, não saltando nenhuma posição.
b) Caso o tamanho da fila for maior que 1, na primeira chamada a cabeça será correcta, mas na segunda vez consecutiva já não. Se a fila for [1,3,3], na primeira chamada a *head* será "1" e na segunda será "3", embora esta em si nunca tenha mudado, o que causa um estado da variável incorrecto de acordo com o que era suposto.
c) Não existe nenhuma situação considerada por nós que dê falha.

- **nextNumModLen**

Não existe verificação nem pré-condição de que o *n* tem de ser menor *que elems.size()* (capacidade dos elementos) nem garantias de que não possa ser negativo.

a) Se o n for -1, um número negativo, existe a falta mencionada acima. No entanto, o valor retornado será 0, sendo este um valor válido.
b) Caso o *n* seja -5, a condição *n+1 == elems.size()* vai dar false e irá retornar a posição -4 que por sua vez não existe.
c) Usando o caso em **b)**, este iria causar uma falha pela utilização de um index inválido.

2. Testes Baseados na Partição do Espaço de Entrada

2.1 Parâmetros e Estados Abstractos

- **BoundedQueue**
  - int capacity
  - Estado das variáveis elems e size
- **Enqueue**
  - T elem
  - Estado das variáveis elems, end e size
- **Dequeue**
  - Estado da variável start e size
- **addAll**
  - Collection<? extends T> collection
  - Estado das variáveis elems e size
- **Head**
  - Estado das variáveis elems e start
- **isFull**
  - Estado das variáveis elems e size
- **isEmpty**
  - Estado da variável size
- **nextNumModLen**
  - int n
  - Estado da variável elems
- **Size**
  - Estado da variável size
- **toString**
  - Estado da variável start, size e elems

2.2 Características e Partições

Para a listagem de características, usámos uma abordagem baseada em interface.

- **BoundedQueue**
  - Capacidade ser maior que zero
- **Enqueue**
  - Elem ser null
  - Relação de end com -1
  - Lista estar cheia
  - Size ser negativo
- **Dequeue**
  - Relação de start com -1
  - Lista estar vazia
  - Size ser negativo
- **addAll**
  - Collection ser null
  - Relação do size da collection com zero(menor, igual e maior)
  - Size ser negativo
  - Elem ser null
- **Head**
  - Relação de start com -1
  - Result ser null
  - Lista estar vazia
- **isFull**
  - Capacidade ser maior que zero
  - Lista estar cheia
- **isEmpty**
  - Lista estar vazia
- **Size**
  - Size ser negativo
- **toString**
  - Relação de start com -1
  - Size ser negativo
  - Elem ser null

2.3 Blocos das Características

Com as características definidas no ponto anterior, os nosso blocos recaem em apenas dois tipos:

- **Booleano** - sendo o bloco 1 **Falso** e o bloco 2 **Verdadeiro**.
    - Capacidade ser maior que 0;
    - Elem ser null;
    - Collection ser null;
    - Result ser null;
    - Lista estar cheia;
    - Lista estar vazia;
    - Size ser negativo.

- **Numérico** - tendo três estados: **menor**, **igual** e **maior** que um dado valor.
    - Relação de end com -1;
    - Relação de start com -1;
    - Relação do size da collection com 0;

Nos casos em que relacionámos variáveis com valores, considerámos -1 por existirem casos onde pode, ou não, originar erro. Existem situações em que os métodos avançam uma casa extra, o que significa que caso o start ou o end fossem -1, o valor final iria dar 0 (sendo um valor válido).

2.4 Representações de Blocos

As bases foram definidas de acordo com a garantia de que o respectivo bloco originaria um erro. Outras dos blocos escolhidos para serem base foram o case de a lista ser vazia ou cheia, visto que este é um pré-requisito na maioria dos métodos. Definimos valores para os parâmetros de acordo com as características e estados da lista.

- **C1** - Capacidade ser maior que zero
    - **Bloco 1** - True
        - capacidade = 3
        - Não existe estado da lista
    - **Bloco 2** - False (**base**)
        - capacidade = -1
        - Não existe estado da lista
- **C2** - Elem ser null
    - **Bloco 1** - True (**base**)
        - elem = null
        - lista = [1, 2, null]
    - **Bloco 2** - False
        - elem = 4
        - lista = [1, 2, null]
- **C3** - Relação de end com -1
    - **Bloco 1** - Menor que -1 (**base**)
        - end = -5

        - lista = [1, 2, 4]
    - **Bloco 2** - Igual a -1
        - end = -1
        - lista = [1, 2, 4]
    - **Bloco 3** - Maior que -1
        - end = 0
        - lista = [1, 2, 4]
- **C4** - Lista estar cheia
    - **Bloco 1** - True (**base**)
        - lista = [1, 2, 4]
    - **Bloco 2** - False (**base**)
        - lista = [1, 2, null]

- **C5** - Relação de start com -1
    - **Bloco 1** - Menor que -1 (**base**)
        - start = -5
        - lista = [1, 2, 4]
    - **Bloco 2** - Igual a -1
        - start = -1
        - lista = [1, 2, 4]
    - **Bloco 3** - Maior que -1
        - start = 0
        - lista = [1, 2, 4]
- **C6** - Lista estar vazia
    - **Bloco 1** - True (**base**)
        - lista = [null, null, null]
    - **Bloco 2** - False (**base**)
        - lista = [1, 2, null]
- **C7** - Collection ser null
    - **Bloco 1** - True
        - collection = null
        - lista = [1, null, null]
    - **Bloco 2** - False (**base**)
        - collection = [5, 6, 7]
        - lista = [1, null, null]
- **C8** - Relação do size da collection com 0
    - **Bloco 1** - Menor que 0 (**base**)
        - collection.size = -1
        - lista = [1, null, null]
    - **Bloco 2** - Igual a 0
        - collection.size = 0
        - lista = [1, null, null]
    - **Bloco 3** - Maior que 0
        - collection.size = 5
        - lista = [1, null, null]
- **C9** - Result ser null
    - **Bloco 1** - True (**base**)
        - result = null
        - lista = [null, 1, 2]
    - **Bloco 2** - False
        - result = 1
        - ta = [1, 2, 4]
- **C11** - Size ser negativo
    - **Bloco 1** - True (**base**)
        - size = -1
        - lista = [1, 2, 4]
    - **Bloco 2** - Falso
        - size = 3
        - lista = [1, 2, 4]

2.5 Escolha Básica Múltipla

Para cada característica foram definidos blocos base já marcados anteriormente. Para duas destas caracteristicas foi definida uma segunda base, também já marcados acima. De seguida apresentamos a cobertura de escolha básica múltipla para cada método. Representamos as bases como os blocos que estão a **negrito** e os testes inviáveis como os que estão a vermelho.

- BoundedQueue (Q = C1)
  C1B1 (sem base)
  C1B2 (o pré-requisito requer que a fila não esteja cheia que a capacidade seja maior que zero)

- Enqueue (Q = C2,C3,C4,C11)
  C2B1-C3B1-C4B1-C11B1 (o pré-requisito requer que a fila não esteja cheia e o size não pode ser negativo)
  C2B1-**C3B1**-**C4B1**-C11B2 (o pré-requisito requer que a fila não esteja cheia)
  C2B1-**C3B1**-**C4B2**-**C11B1** (o size não pode ser negativo)
  C2B1-C3B2-**C4B1**-**C11B1** (o pré-requisito requer que a fila não esteja cheia)
  C2B1-**C3B1**-**C4B2**-C11B2 (sendo o end um index, nenhum index poderá ser negativo)
  C2B1-C3B2-**C4B1**-C11B2 (o pré-requisito requer que a fila não esteja cheia)
  C2B1-C3B2-**C4B2**-C11B2 (sendo o end um index, nenhum index poderá ser negativo)
  C2B1-C3B3-**C4B1**-C11B2 (o pré-requisito requer que a fila não esteja cheia)
  C2B1-C3B3-**C4B2**-**C11B1** (o size não pode ser negativo)
  C2B1-C3B3-**C4B1**-**C11B1** (o pré-requisito requer que a fila não esteja cheia e o size não pode ser negativo)
  C2B1-C3B3-**C4B2**-C11B2

  C2B2-**C3B1**-**C4B1**-**C11B1** (o pré-requisito requer que a fila não esteja cheia)
  C2B2-**C3B1**-**C4B1**-C11B2 (o pré-requisito requer que a fila não esteja cheia)
  C2B2-**C3B1**-**C4B2**-**C11B1** (o size não pode ser negativo)
  C2B2-C3B2-**C4B1**-**C11B1** (o pré-requisito requer que a fila não esteja cheia)
  C2B2-**C3B1**-**C4B2**-C11B2
  C2B2-C3B2-**C4B1**-C11B2 (o pré-requisito requer que a fila não esteja cheia)
  C2B2-C3B2-**C4B2**-C11B2
  C2B2-C3B3-**C4B1**-C11B2 (o pré-requisito requer que a fila não esteja cheia)
  C2B2-C3B3-**C4B2**-**C11B1** (o size não pode ser negativo)
  C2B2-C3B3-**C4B1**-**C11B1** (o pré-requisito requer que a fila não esteja cheia)
  C2B2-C3B3-**C4B2**-C11B2 (viável de certeza)

- Dequeue (Q = C5,C6)

C5B1-C6B1-C11B1 (o pré-requisito requer que a fila não esteja vazia e o size não pode ser negativo**)**

C5B1-C6B1-C11B2 **(**o pré-requisito requer que a fila não esteja vazia**)**

C5B1-C6B2-C11B1 (sendo o start um index, nenhum index poderá ser negativo e o size não pode ser negativo)

C5B1-C6B2-C11B2

C5B2-**C6B1-**C11B1 **(**o pré-requisito requer que a fila não esteja vazia e o size não pode ser negativo**)**

C5B2-**C6B1-C11B2 (**o pré-requisito requer que a fila não esteja vazia**)**

C5B2-**C6B2-**C11B1 (sendo o start um index, nenhum index poderá ser negativo)

C5B2-**C6B2-C11B2** (sendo o start um index, nenhum index poderá ser negativo)

C5B3-**C6B1-**C11B1 (o pré-requisito requer que a fila não esteja vazia e o size não pode ser negativo)

C5B3-**C6B1-C11B2 (**o pré-requisito requer que a fila não esteja vazia)

C5B3-**C6B2-**C11B1 (o size não pode ser negativo)

C5B3-**C6B2-C11B2**

- addAll (Q = C1,C7,C8,C11)

C1B1-C7B1-**C8B1-C11B1** (collection size não pode ser negativo e o size não pode ser negativo)

C1B1-C7B1-**C8B1-**C11B2 (collection size não pode ser negativo)

C1B1-C7B1-C8B2-**C11B1** (o size não pode ser negativo)

C1B1-**C7B2-C8B1-C11B1** (collection size não pode ser negativo e o size não pode ser negativo)

C1B1-C7B1-**C8B1-C11B1** (collection size não pode ser negativo e o size não pode ser negativo)

C1B1-C7B1-C8B2-C11B2 (sem base)

C1B1-**C7B2-C8B1-**C11B2 (collection size não pode ser negativo)

C1B1-**C7B2-**C8B2-C11B2

C1B1-C7B1-C8B3-C11B2 (sem base)

C1B1-C7B1-C8B3-**C11B1** (o size não pode ser negativo)

C1B1-**C7B2-**C8B3-**C11B1** (o size não pode ser negativo)

C1B1-C7B1-C8B3-**C11B1** (o size não pode ser negativo)

C1B1-**C7B2-**C8B3-C11B2

C1B1-C7B1-C8B3-C11B2 (sem base)

C1B2-C7B1-**C8B1-C11B1** (a capacidade não pode ser negativa)

C1B2-C7B1-**C8B1-**C11B2 (“)

C1B2-C7B1-C8B2-**C11B1** (“)

C1B2-C7B2-C8B1-C11B1 (“)

C1B2-C7B1-**C8B1-C11B1** (“)

C1B2--C7B1-C8B2-C11B2 (“)

C1B2-**C7B2**-**C8B1**-C11B2 (“)
C1B2-C7B1-**C8B1**-C11B2 (“)
C1B2-**C7B2**-C8B2-C11B2 (“)
C1B2-**C7B2**-C8B2-C11B2 (“)
C1B2-C7B1-C8B3-C11B2 (“)
C1B2-C7B1-C8B3-**C11B1** (“)
C1B2-**C7B2**-C8B3-**C11B1** (“)
C1B2-C7B1-C8B3-**C11B1** (“)
C1B2-C7B1-C8B3-C11B2 (“)
C1B2-**C7B2**-C8B3-C11B2 (“)
C1B2-C7B1-C8B3-C11B2 (“)
C1B2-**C7B2**-C8B3-C11B2 (“)

- Head (Q = C9,C5,C6)

C9B1-**C5B1**-**C6B1** **(**o pré-requisito requer que a fila não esteja vazia e sendo o start um index, nenhum index poderá ser negativo)
C9B1-**C5B1**-**C6B2**
C9B1-C5B2-**C6B1** **(**o pré-requisito requer que a fila não esteja vazia e sendo o start um index, nenhum index poderá ser negativo)
C9B1-C5B2-**C6B2** **(**sendo o start um index, nenhum index poderá ser negativo)
C9B1-C5B3-**C6B1** **(**o pré-requisito requer que a fila não esteja vazia)
C9B1-C5B3-**C6B2**

C9B2-**C5B1**-**C6B1** **(**o pré-requisito requer que a fila não esteja vazia e sendo o start um index, nenhum index poderá ser negativo)
C9B2-**C5B1**-**C6B2** **(**sendo o start um index, nenhum index poderá ser negativo)
C9B2-C5B2-**C6B1****(**o pré-requisito requer que a fila não esteja vazia e sendo o start um index, nenhum index poderá ser negativo)
C9B2-C5B2-**C6B2** **(**sendo o start um index, nenhum index poderá ser negativo)
C9B2-C5B3-**C6B1****(**o pré-requisito requer que a fila não esteja vazia)
C9B2-C5B3-**C6B2**

- isFull (Q = C1,C11)

C1B1-**C11B1** (o size não pode ser negativo)
C1B1-C11B2 (sem base)
C1B2-**C11B1** (a capacidade tem de ser maior que zero e o size não pode ser negativo)
C1B2-C11B2 (a capacidade tem de ser maior que zero)

- isEmpty (Q = C11)

C11B1 (o size não pode ser negativo)
C11B2 (sem base)

- Size (Q = C11)

  C11B1 (o size não pode ser negativo)
  C11B2 (sem base)

- toString (Q = C2,C5,C11)
  C2B1-C5B1-C11B1 (size não pode ser negativo)
  C2B1-C5B1-C11B2 (size não pode ser negativo)
  C2B1-C5B2-**C11B1** (size não pode ser negativo)
  C2B1-C5B2-C11B2 (size não pode ser negativo)
  C2B1-C5B3-**C11B1** (size não pode ser negativo)
  C2B1-C5B3-C11B2

  C2B2-C5B1-C11B1 (size não pode ser negativo)
  C2B2-C5B1-C11B2 (size não pode ser negativo)
  C2B2-C5B2-**C11B1** (size não pode ser negativo)
  C2B2-C5B2-C11B2 (sem base)
  C2B2-C5B3-**C11B1** (size não pode ser negativo)
  C2B2-C5B3-C11B2 (sem base)

2.6 Tabela Resumo

Depois de definidas as características, blocos e seus representantes, apresentamos em baixo uma tabela com um resumo geral.

	Caracteristicas		Blocos			Representantes		
	#C	Designação	B1	B2	B3	B1	B2	B3
BOOLEANO	C1	Capacidade ser maior que zero	True ser maior que zero	False ser menor que zero		capacidade = 4 não existe estado da lista	capacidade = -1 não existe estado da lista	
	C2	Elem ser null	True ser null	False ser não null		elem = null lista = [1,2,null]	elem = 4 lista = [1,2,null]	
	C7	Collection ser null	True ser null	False ser não null		collection = null lista = [1,null,null]	collection = [5,6,7] lista = [1,null,null]	
	C9	Result ser null	True ser null	False ser não null		result = null lista = [null,1,2]	result = 1 lista = [1,2,4]	
	C4	Lista estar cheia	True estar cheia	False não estar cheia		lista = [1,2,4]	lista = [1,2,null]	
	C6	Lista estar vazia	True estar vazia	False não estar vazia		lista = [null, null, null]	lista = [1,2,null]	
	C11	Size ser negativo	True ser negativo	False não ser negativo		size = -1 lista = [1,2,4]	size = 3 lista = [1,2,4]	
NUMÉRICO	C3	Relação de end com -1	menor que -1	igual a -1	maior que -1	end = -5, lista = [1,2,4]	end = -1, lista = [1,2,4]	end = 0, lista = [1,2,4]
	C5	Relação de start com -1	menor que -1	igual a -1	maior que -1	start = -5, lista = [1,2,4]	start = -1, lista = [1,2,4]	start = 0, lista = [1,2,4]
	C8	Relação do size da collection com 0	menor que 0	igual a 0	maior que 0	collection.size = -1 lista= [1,null,null]	collection.size = 0 lista= [1,null,null]	collection.size = 5 lista= [1,null,null]

Métodos	Testes
BoundedQueue	
Enqueue	C2B1->C3B3->C4B2->C11B2 C2B2->C3B3->C4B2->C11B2
Dequeue	C5B3->C6B2
Addall	C1B1->C7B2->C8B2->C11B2 C1B1->C7B2->C8B3->C11B2
head	C9B1->C5B3->C6B2 C9B2->C5B3->C6B2
isFull	
isEmpty	
size	
toString	C2B1->C5B2->C11B2 C2B1->C5B3->C11B2

2.7 Requisitos Inviáveis

Para requisitos inviáveis considerámos que todos os blocos das características que envolvessem valores negativos, isto é, o size ser menor que um ou o start e end, fossem requisitos inviáveis. Para além disto, todos os que violassem os pré-requisitos também foram considerados como inviáveis. A marcação dos requisitos como inviáveis foi feita no ponto **4.5**, estando estes a vermelho.

2.8 Testes em JUnit

Os testes em JUnit foram feitos no Eclipse, pelo que podem ser consultados no ficheiro vvs003.zip, classe tests/adts/BoundedQueueTest.java. Dado que existe uma exceção em falta, implementou-se a classe src/adts/BoundedQueueException.java, de forma a não limitar/deturpar os testes dos métodos que não dão uso a esta exceção.

2.9 Correção de Faltas

Nesta secção, é apresentada uma lista de todas as falhas dadas nos testes e as correções adequadas respectivas. As correções poderão ser encontradas no ficheiro src/adts/BoundedQueueCorrected.java

- **Enqueue**
  - **Falha 1** :java.lang.Exception: Unexpected exception, expected <java. lang.IllegalArgumentException> but was<java.lang.AssertionError>
  - **Correção 1**: acrescentou-se uma condição para que só receba **elem != null**
  - **Falha 2**: java.lang.AssertionError: Elem nao null (primeira posicao) expected:<4> but was:<1>

    - **Correção 2**: trocou-se a ordem entre **elems.set(end, elem)** e **end = nextNumModLen(end)**
- **Dequeue**
    - **Falha**: java.lang.AssertionError: # Elementos na lista expected:<2> but was:<3>
    - **Correção**: acrescentou-se **size--** após a execução do método nextNumModLen()
- **addAll**
    - **Falha**: BoundedQueueException
    - **Correção**: na condição do if alterou-se o <= para **>** e acrescentou-se na mesma **condição collection.size() == 0**
- **Head**
    - **Falha**: java.lang.AssertionError: Start mantém expected:<0> but was:<1>
    - **Correção**: **remoção** da atribuição do resultado **nextNumModLen()** ao start
- **Size**
    - **Falha**: java.lang.AssertionError: Número de elementos da fila expected:<3> but was:<4>
    - **Correção**: **remoção do +1** no return
- **toString**
    - **Falha**: org.junit.ComparisonFailure: Representacao da fila expected:<[null, 2[]]> but was:<[null, 2[]>
    - **Correção**: acrescentou-se **sb.append("]")** depois do ciclo

3. Cobertura por Fluxo de Controle

Todos os testes aqui realizados foram desenhados de acordo com a classe BoundedQueue corrigida (BoundedQueueaCorrected.java).

3.1 Grafo e Caminhos

Caminhos Primos

Caminos de comprimento 1 a 3:

[0,1]
[3,4,3]
[4,3,4]
[8,9,8]
[9,8,9]
[11,12,11]
[12,11,12]
[14,15,14]
[15,14,15]
[18,17,18]
[17,18,17]
[21,20,21]
[20,21,20]
[0,2,3,4]

Começa em 0:

[0,2,3,5,6,23]
[0,2,3,5,6,7,8,9]
[0,2,3,5,6,7,8,10,11,12]
[0,2,3,5,6,7,8,10,11,13,14,15]
[0,2,3,5,6,7,8,10,11,13,14,16,17,18]
[0,2,3,5,6,7,8,10,11,13,14,16,17,19,20,21]
[0,2,3,5,6,7,8,10,11,13,14,16,17,19,20,22]

Começa em 4:

[4,3,5,6,23]
[4,3,5,6,7,8,9]
[4,3,5,6,7,8,10,11,12]
[4,3,5,6,7,8,10,11,13,14,15]
[4,3,5,6,7,8,10,11,13,14,16,17,18]
[4,3,5,6,7,8,10,11,13,14,16,17,19,20,21]
[4,3,5,6,7,8,10,11,13,14,16,17,19,20,22]

Começa em 6:

[6,7,8,9]
[6,7,8,10,11,12]
[6,7,8,10,11,13,14,15]
[6,7,8,10,11,13,14,16,17,18]
[6,7,8,10,11,13,14,16,17,19,20,21]
[6,7,8,10,11,13,14,16,17,19,20,22,6]

Começa em 7:

[7,8,10,11,13,14,16,17,19,20,22,6,23]
[7,8,10,11,13,14,16,17,19,20,22,6,7]

Começa em 8:

[8,10,11,13,14,16,17,19,20,22,6,7,8]

Começa em 9:

[9,8,10,11,12]
[9,8,10,11,13,14,15]
[9,8,10,11,13,14,16,17,18]
[9,8,10,11,13,14,16,17,19,20,21]
[9,8,10,11,13,14,16,17,19,20,22,6,7]
[9,8,10,11,13,14,16,17,19,20,22,6,23]

Começa em 10:

[10,11,13,14,16,17,19,20,22,6,7,8,10]

Começa em 11:

[11,13,14,16,17,19,20,22,6,7,8,10,11]

Começa em 12:

[12,11,13,14,15]
[12,11,13,14,16,17,18]

[12,11,13,14,16,17,19,20,21]
[12,11,13,14,16,17,19,20,22,6,23]
[12,11,13,14,16,17,19,20,22,6,7,8,9]
[12,11,13,14,16,17,19,20,22,6,7,8,10]

Começa em 13:
[13,14,16,17,19,20,22,6,7,8,10,11,13]

Começa em 14:
[14,16,17,19,20,22,6,7,8,10,11,13,14]

Começa em 15:
[15,14,16,17,18]
[15,14,16,17,19,20,21]
[15,14,16,17,19,20,22,6,23]
[15,14,16,17,19,20,22,6,7,8,9]
[15,14,16,17,19,20,22,6,7,8,10,11,12]
[15,14,16,17,19,20,22,6,7,8,10,11,13]

Começa em 16:
[16,17,19,20,22,6,7,8,10,11,13,14,16]

Começa em 17:
[17,19,20,22,6,7,8,10,11,13,14,16,17]

Começa em 18:
[18,17,19,20,21]
[18,17,19,20,22,6,23]
[18,17,19,20,22,6,7,8,9]
[18,17,19,20,22,6,7,8,10,11,12]
[18,17,19,20,22,6,7,8,10,11,13,14,15]
[18,17,19,20,22,6,7,8,10,11,13,14,16]

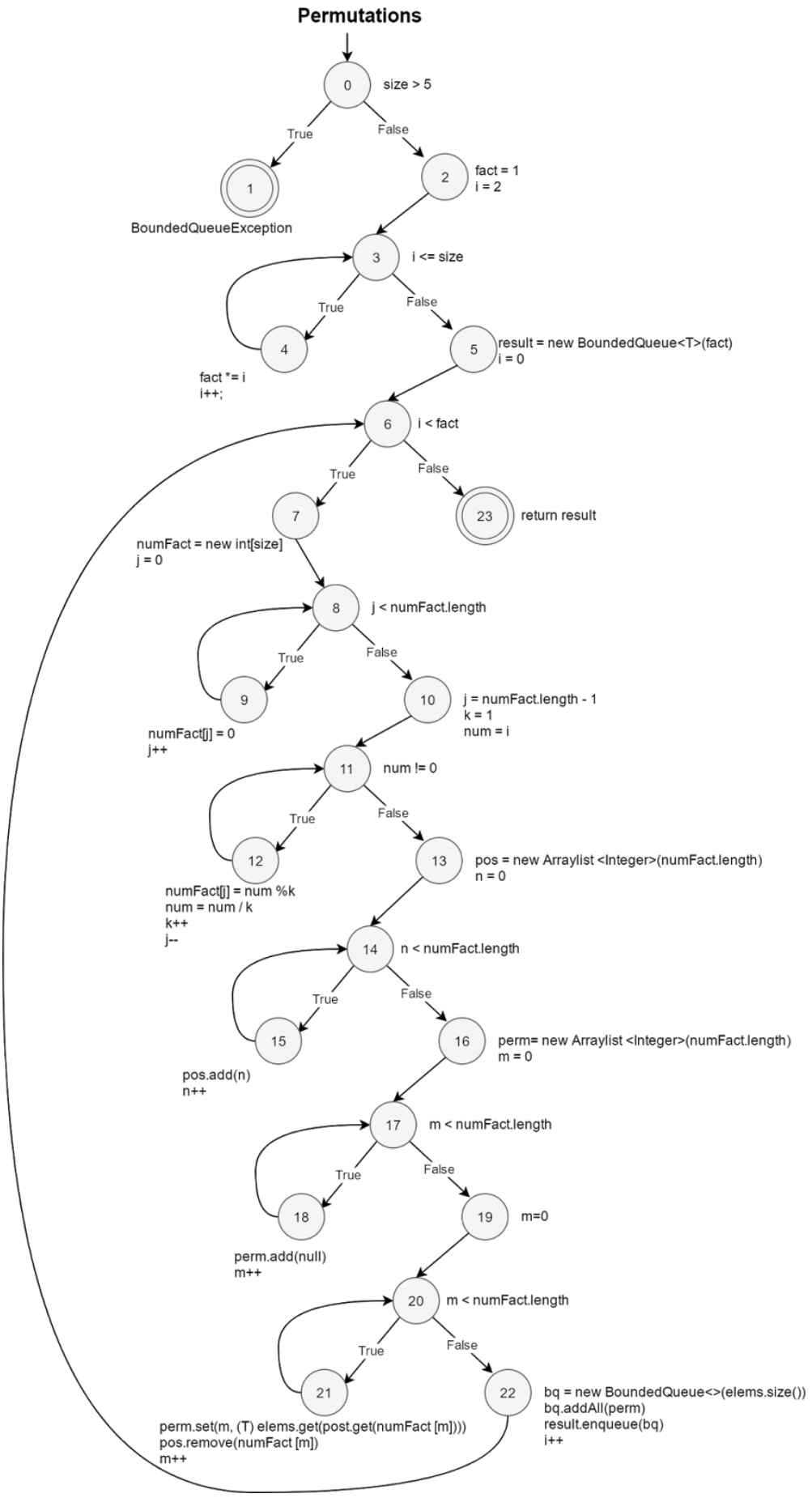
Começa em 19:
[19,20,22,6,7,8,10,11,13,14,16,17,19]

Começa em 20:
[20,22,6,7,8,10,11,13,14,16,17,19,20]

Começa em 21:
[21,20,22,6,23]
[21,20,22,6,7,8,9]
[21,20,22,6,7,8,10,11,12]
[21,20,22,6,7,8,10,11,13,14,15]
[21,20,22,6,7,8,10,11,13,14,16,17,18]
[21,20,22,6,7,8,10,11,13,14,16,17,19]

Começa em 22:
[22,6,7,8,10,11,13,14,16,17,19,20,22]

Total de 75 caminhos primos


Permutations
0
size > 5
True
False
1
BoundedQueueException
2
fact = 1
i = 2
3
i <= size
True
False
4
fact *= i
i++;
5
result = new BoundedQueue<T>(fact)
i = 0
6
i < fact
True
False
7
numFact = new int[size]
j = 0
23
return result
8
j < numFact.length
True
False
9
numFact[j] = 0
j++
10
j = numFact.length - 1
k = 1
num = i
11
num != 0
True
False
12
numFact[j] = num %k
num = num / k
k++
j--
13
pos = new Arraylist <Integer>(numFact.length)
n = 0
14
n < numFact.length
True
False
15
pos.add(n)
n++
16
perm= new Arraylist <Integer>(numFact.length)
m = 0
17
m < numFact.length
True
False
18
perm.add(null)
m++
19
m=0
20
m < numFact.length
True
False
21
perm.set(m, (T) elems.get(post.get(numFact [m])))
pos.remove(numFact [m])
m++
22
bq = new BoundedQueue<>(elems.size())
bq.addAll(perm)
result.enqueue(bq)
i++

w+

3.2 Tabela de Resumo

Variáveis (inputs)

- Estado das variáveis size e elems.

Características

- C1 - Relação de size com o intervalo [0,5]
    - Bloco 1 - size igual a 0 (**base**)
        - size = 0;
        - elems = [null, null, null ,null ,null, null]
    - Bloco 2 - size pertence a]0,5]
        - size = 3
        - elems = [1, 2, 3 ,null ,null, null]
    - Bloco 3 - size maior que 5
        - size = 6
        - elems = [1, 2, 3 ,4 ,5, 6]
- C2 - Elementos da lista são todos iguais
    - Bloco 1 - True
        - elems = [2, 2, 2, null, null, null]
    - Bloco 2 - False (**base**)
        - elems = [1,2,3, null, null ,null]
- C3 - Elem.size()(capacidade) é maior que 0:
    - Bloco 1 - True (**base**)
        - elems.size() = 6
    - Bloco 2 - False
        - elems.size() = 0

Combinações

C1B1-C2B1-**C3B1** (elem não pode ser null)
C1B1-C2B1-C3B2 (elem não pode ser null e capacidade não pode ser igual ou menor que 0(BoundedQueue requer que capacidade seja > 0))
C1B1-**C2B2**-**C3B1**
C1B1-**C2B2**-C3B2 (capacidade não pode ser igual ou menor que 0(BoundedQueue requer que capacidade seja > 0))
C1B2-C2B1-**C3B1** (elem não pode ser null)
C1B2-C2B1-C3B2 (sem base)
C1B2-**C2B2**-**C3B1**
C1B2-**C2B2**-C3B2 (capacidade não pode ser igual ou menor que 0(BoundedQueue requer que capacidade seja > 0))
C1B3-C2B1-**C3B1** (elem não pode ser null)

C1B3-C2B1-C3B2 (sem base e capacidade não pode ser igual ou menor que 0 (BoundedQueue)
C1B3-**C2B2**-**C3B1**
C1B3-**C2B2**-C3B2 (capacidade não pode ser igual ou menor que 0 (BoundedQueue requer que capacidade seja > 0))

Combinações Viáveis:

- **C1B1**-**C2B2**-**C3B1**
- C1B2-**C2B2**-**C3B1**
- C1B3-**C2B2**-**C3B1**

Para além das características considerámos outros inputs igualmente interessantes, relativamente ao valor do size:

Inputs a testar

- Size maior que 5:
  - size = 6
- Size igual a 0:
  - size = 0
- Size pertence a]0,5]:
  - size = 1 || 2 || 3 || 4 || 5

Caminhos a testar

a) Size = 0
   i) [0,2,3,5,6,7,8,10,11,13,14,16,17,19,20,22]
b) Size = 1
   i) [0,2,3,5,6,7,8,9,8,10,11,13,14,15,14,16,17,18,17,19,20,21,20,22,6,23]
c) Size = 2
   i) [0,2,3,4,3,5,6,7,8,9,8,9,8,10,11,13,14,15,14,15,14,16,17,18,17,18,17,19,20,21,20,21,20,22,6,7,8,9,8,9,8,10,11,12,11,12,11,13,14,15,14,15,14,16,17,18,17,18,17,19,20,21,20,21,20,22,6,23]
d) Size = 3
   i) Idêntico ao caminho do size = 2 com mais iterações nos ciclos
e) Size = 4
   i) Idêntico ao caminho do size = 2 com mais iterações nos ciclos
f) Size = 5
   i) Idêntico ao caminho do size = 2 com mais iterações nos ciclos
g) Size = 6
   i) [0,1]

Os testes desenhados focam-se em saber o número de permutações feitas, o número de elementos de cada elemento permutado, a distinção dos elementos a permutar e a distinção das permutações.

Método	Inputs	Valor esperado	Resultado	Caminho Percorrido	Caminhos Primos Cobertos
permutationsTestSize0	lista = [null, null ,null, null, null ,null] size = 0 elems.size() = 6	BoundedQueueException	BoundedQueueException	[0,2,3,5,6,7,8,10,11,13,14,16,17,19,20,22]	[0,2,3,5,6,7,8,10,11,13,14,16,17,19,20,22]
permutationTestSize1	lista = [1, null ,null, null, null ,null] size = 1 elems.size() = 6	[[1]]	[[1]]	[0,2,3,5,6,7,8,9,8,10,11,13,14,15,14,16,17,18,17,19,20,21,20,22,6,23]	[8,9,8], [14,15,14], [17,18,17], [20,21,20], [6,7,8,9],[0,2,3,5,6,7,8,9],[9,8,10,11,13,14,15], [15,14,16,17,18], [18,17,19,20,21],[21,20,22,6,23]
permutationTestSize2	lista = [1, 2 ,null, null, null ,null] size = 2 elems.size() = 6	[[1,2],[2,1]]	[[1,2],[2,1]]	[0,2,3,4,3,5,6,7,8,9,8,9,8,10,11,13,14,15,14,15,14,16,17,18,17,18,17,19,20,21,20,21,20,22,6,7,8,9,8,9,8,10,11,12,11,12,11,13,14,15,14,15,14,16,17,18,17,18,17,19,20,21,20,21,20,22,6,23]	[3,4,3], [8,9,8], [9,8,9], [14,15,14], [15,14,15], [17,18,17], [18,17,18], [20,21,20] [21,20,21], [0,2,3,4], [4,3,5,6,7,8,9], [6,7,8,9], [9,8,10,11,12], [9,8,10,11,13,14,15], [12,11,13,14,15], [15,14,16,17,18], [18,17,19,20,21], [21,20,22,6,23], [21,20,22,6,7,8,9]
permutationTestSize6	lista = [1, 2 ,3, 4, 5, 6] size = 6 elems.size() = 6	BoundedQueueException	BoundedQueueException	[0,1]	[0,1]

Decidimos não incluir na tabela os testes para size = 3 ,size = 4 e size = 5 porque a partir do size = 2, apesar dos caminhos serem diferentes, a cobertura de caminhos primos não vai diferir. Por isso, apesar de não estarem incluídos estes estão todos implementados no tests/adts/PermutationsTest.java.

3.3 Requisitos Inviáveis

Com base no terceiro teste, encontrámos um padrão nos restantes caminhos primos nos quais os ciclos são sempre acedidos, o que leva a que os caminhos percorridos deixem de ser primos. Concluindo assim que os únicos caminhos primos viáveis são:

- [0,1]
- [3,4,3]
- [8,9,8]
- [9,8,9]
- [14,15,14]
- [15,14,15]
- [17,18,17]
- [18,17,18]
- [20,21,20]
- [21,20,21]
- [0,2,3,4]
- [6,7,8,9]
- [9,8,10,11,12]
- [12,11,13,14,15]
- [15,14,16,17,18]
- [18,17,19,20,21]
- [21,20,22,6,23]
- [4,3,5,6,7,8,9]
- [9,8,10,11,13,14,15]
- [21,20,22,6,7,8,9]
- [0,2,3,5,6,7,8,9]
- [0,2,3,5,6,7,8,10,11,13,14,16,17,19,20,22]

Todos os outros caminhos são inviáveis, visto que não existe nenhum caminho de teste que os cubra.

Com base nos testes e correções feitos no ponto 3, encontrámos uma falta neste método relativamente ao **size** das permutações. Existe uma excepção para size > 5 mas não

para size = 0. Ao ter um size = 0, existe uma exceção no **addall()**, pelo que uma collection não pode ter tamanho 0.

4. Cobertura por Fluxo de Dados

4.1 DU-Pairs e DU-Path

Variáveis	DU-PAIRS	Caminhos-DU-ALL	Caminhos-DU-ALL Distinct
size	(0,(0,2))	[0,2]	[0,2]
	(0,(0,1))	[0,1]	[0,1]
	(0,(3,4))	[0,2,3,4]	[0,2,3,4]
	(0,(3,5))	[0,2,3,5]	[0,2,3,5]
	(0,7)	[0,2,3,5,6,7]	[0,2,3,5,6,7]
fact	(2,4)	[2,3,4]	[2,3,4]
	(2,(6,7))	[2,3,5,6,7]	[2,3,5,6,7]
	(2,(6,23))	[2,3,5,6,7,8,10,11,13,14,16,17,19,20,22,6,23]	[2,3,5,6,7,8,10,11,13,14,16,17,19,20,22,6,23]
	(2,5)	[2,3,5]	[2,3,5]
	(4,5)	[4,3,5]	[4,3,5]
	(4,4)	[4,3,4]	[4,3,4]
	(4,(6,7))	[4,3,5,6,7]	[4,3,5,6,7]
	(4,(6,23))	[4,3,5,6,23]	[4,3,5,6,23]
i	(2,(3,4))	[2,3,4]	
	(2,(3,5))	[2,3,5]	
	(2,4)	[2,3,4]	
	(4,(3,4))	[4,3,4]	
	(4,(3,5))	[4,3,5]	
	(4,4)	no path needed	
	(5,(6,7))	[5,6,7]	[5,6,7]

	(5,(6,23))	[5,6,23]	[5,6,23]
	(5,10)	[5,6,7,8,10]	[5,6,7,8,10]
	(5,22)	[5,6,7,8,10,11,13,14,16,17,19,20,22]	[5,6,7,8,10,11,13,14,16,17,19,20,22]
	(22,(6,7))	[22,6,7]	[22,6,7]
	(22,(6,23))	[22,6,23]	[22,6,23]
	(22,10)	[22,6,7,8,10]	[22,6,7,8,10]
	(22,22)	no path needed	
result	(5,23)	[5,6,23]	
	(22,23)	[22,6,23]	
numFact	(7,(8,9))	[7,8,9]	[7,8,9]
	(7,(8,10))	[7,8,10]	[7,8,10]
	(7,10)	[7,8,10]	
	(7,(14,15))	[7,8,10,11,13,14,15]	[7,8,10,11,13,14,15]
	(7,(14,16))	[7,8,10,11,13,14,16]	
	(7,16)	[7,8,10,11,13,14,16]	[7,8,10,11,13,14,16]
	(7,(17,18))	[7,8,10,11,13,14,16,17,18]	[7,8,10,11,13,14,16,17,18]
	(7,(17,19))	[7,8,10,11,13,14,16,17,19]	[7,8,10,11,13,14,16,17,19]
	(7,(20,21))	[7,8,10,11,13,14,16,17,19,20,21]	[7,8,10,11,13,14,16,17,19,20,21]
	(7,21)	[7,8,10,11,13,14,16,17,19,20,21]	
	(7,(20,22))	[7,8,10,11,13,14,16,17,19,20,22]	[7,8,10,11,13,14,16,17,19,20,22]
	(9,10)	[9,8,10]	[9,8,10]
	(9,(14,15))	[9,8,10,11,13,14,15]	[9,8,10,11,13,14,15]
	(9,(14,16))	[9,8,10,11,13,14,16]	[9,8,10,11,13,14,16]
	(9,16)	[9,8,10,11,13,14,16]	
	(9,(17,18))	[9,8,10,11,13,14,16,17,18]	[9,8,10,11,13,14,16,17,18]
	(9,(17,19))	[9,8,10,11,13,14,16,17,19]	[9,8,10,11,13,14,16,17,19]

	(9,(20,21))	[9,8,10,11,13,14,16,17,19,20,21]	[9,8,10,11,13,14,16,17,19,20,21]
	(9,(20,22))	[9,8,10,11,13,14,16,17,19,20,22]	[9,8,10,11,13,14,16,17,19,20,22]
	(9,21)	[9,8,10,11,13,14,16,17,19,20,21]	
	(12,(14,15))	[12,11,13,14,15]	[12,11,13,14,15]
	(12,(14,16))	[12,11,13,14,16]	[12,11,13,14,16]
	(12,16)	[12,11,13,14,15,14,16]	[12,11,13,14,15,14,16]
	(12,(17,18))	[12,11,13,14,16,17,18]	[12,11,13,14,16,17,18]
	(12,(17,19))	[12,11,13,14,16,17,19]	[12,11,13,14,16,17,19]
	(12,(20,21))	[12,11,13,14,16,17,19,20,21]	[12,11,13,14,16,17,19,20,21]
	(12,(20,22))	[12,11,13,14,16,17,19,20,22]	[12,11,13,14,16,17,19,20,22]
	(12,21)	[12,11,13,14,16,17,19,20,21]	
j	(7,(8,9))	[7,8,9]	
	(7,(8,10))	[7,8,10]	
	(7,9)	[7,8,9]	
	(9,9)	[9,8,9]	[9,8,9]
	(9,(8,9))	[9,8,9]	
	(9,(8,10))	[9,8,10]	
	(10,12)	[10,11,12]	[10,11,12]
	(12,12)	no path needed	
k	(10,12)	[10,11,12]	
	(12,12)	no path needed	
num	(10,(11,12))	[10,11,12]	
	(10,(11,13))	[10,11,13]	[10,11,13]
	(10,12)	[10,11,12]	
	(12,(11,12))	[12,11,12]	

	(12,(11,13))	[12,11,13]	[12,11,13]
	(12,12)	no path needed	
pos	(21,21)	no path needed	
	(13,21)	[13,14,16,17,19,20,21]	[13,14,16,17,19,20,21]
	(15,21)	[15,14,16,17,19,20,21]	[15,14,16,17,19,20,21]
n	(13,(14,15))	[13,14,15]	[13,14,15]
	(13,15)	[13,14,15]	
	(13,16)	[13,14,16]	[13,14,16]
	(15,15)	no path needed	
perm	(18,22)	[18,17,19,20,22]	[18,17,19,20,22]
	(21,22)	[21,20,22]	[21,20,22]
	(16,22)	[16,17,19,20,22]	[16,17,19,20,22]
m	(16,(17,18))	[16,17,18]	[16,17,18]
	(16,(17,19))	[16,17,19]	[16,17,19]
	(18,(17,18))	[18,17,18]	[18,17,18]
	(18,(17,19))	[18,17,19]	[18,17,19]
	(18,18)	no path needed	
	(19,(20,21))	[19,20,21]	[19,20,21]
	(19,(20,22))	[19,20,22]	[19,20,22]
	(21,(20,21))	[21,20,21]	[21,20,21]
	(21,(20,22))	[21,20,22]	[21,20,22]
	(19,21)	[19,20,21]	
	(21,21)	no path needed	
bq	(22,22)	no path needed	

4.2 Casos de Teste

Variáveis	ADUPC	AUC	ADC
size	[0,2] [0,1] [0,2,3] [0,2,3,4] [0,2,3,5] [0,2,3,5,6,7] [0,2,3,4,3,5,6,7]	[0,2] [0,1] [0,2,3,4] [0,2,3,5] [0,2,3,5,6,7]	[0,2,3,4]
fact	[2,3,4] [2,3,5] [2,3,5,6,7] [2,3,5,6,23] [4,3,5] [4,3,4] [4,3,5,6,7] [4,3,5,6,23]	[2,3,4] [2,3,5,6,7] [4,3,4] [4,3,5,6,7] [4,3,5,6,23]	[2,3,5,6,7]
i	[2,3,4] [2,3,5] [4,3,4] [4,3,5] [5,6,7] [5,6,23] [5,6,7,8,10] [5,6,7,8,9,8,10] [5,6,7,8,10,11,13,14,16,17,19,20,22] [22,6,7] [22,6,23] [22,6,7,8,10] [22,6,7,8,9,8,10]	[2,3,4] [2,3,5] [4,3,4] [4,3,5] [5,6,7] [5,6,23] [5,6,7,8,10] [5,6,7,8,10,11,13,14,16,17,19,20,22] [22,6,7] [22,6,23] [22,6,7,8,10]	[2,3,4]

	[5,6,7,8,9,8,10,11,13,14,15,14,16,17,18,17,19,20,21,20,22] [5,6,7,8,9,8,10,11,12,11,13,14,15,14,16,17,18,17,19,20,21,20,22] [22,6,7,8,9,8,10,11,13,14,15,14,16,17,18,17,19,20,21,20,22] [22,6,7,8,9,8,10,11,12,11,13,14,15,14,16,17,18,17,19,20,21,20,22]		
result	[5,6,23] [22,6,23]	[5,6,23] [22,6,23]	[5,6,23]

Não fizemos para o resto das variáveis porque o processo seria idêntico, para além das imensas combinações possíveis.

4.3 Programação de Testes

Como há sempre caminhos por testar, não podemos garantir que não nos tenha passado algum teste que não tenhamos feito. Logo, nunca se consegue programar todos os testes porque há sempre testes para serem programados.

4.4 Requisitos Inviáveis

Todos os caminhos em que as variáveis são definidas dentro dos ciclos, mas que o caminho não acede a outros ciclos posteriores, são considerados caminhos inviáveis. Isto acontece pelo facto de, se começarmos num ciclo, o numFact.length() será >0 e assim obriga a entrar nos restantes ciclos, excepto o que tem a condição com num.

Para além da situação mencionada, todos estes caminhos que não se encontram nos caminhos primos cobertos, são considerados inviáveis.