BD_LAB

Integrazione e Test di Sistemi Software a.a. 2023-2024

Corso di laurea in Informatica e Tecnologie per la Produzione del Software

Realizzato da

Di Gennaro Pasquale 758195 ITPS
p.digennaro5@studenti.uniba.it

Balzano Nicola 754694 ITPS
n.balzano2@studenti.uniba.it

Dipartimento di Informatica - Università degli studi di Bari
Via Orabona 4 - 70125 - Bari
email: abcduniba@gmail.com
www.uniba.it/it/ricerca/dipartimenti/informatica

BD_lab
SOFTWARE DEVELOPMENT

HOMEWORK NUMERO 1

PRIMA FASE:

specification-based-testing(Black Box)

1. Comprendere i requisiti

Il programma deve concatenare le stringhe nella stessa posizione poste rispettivamente in due liste e restituire la lista delle stringhe concatenate

2. Comprendere cosa fa il programma con vari input e cosa restituisce in output

Input:

- Se una delle sue stringhe è nulla restituisce null.
- Se la lunghezza di una delle due liste in input è maggiore di N_MAX (scelto arbitrariamente) solleva l'eccezione IllegalArgumentException.
- Se una stringa nella lista di destra contiene spazi li cancella.
- Se una stringa nella lista di destra contiene la parola “no” la cancella.
- Se una stringa nella lista di sinistra contiene un numero attaccato ad un carattere non numerico (quindi senza lo spazio tra i due), lo deve inserire.
- Se una delle due liste è di dimensione differente, tratta le stringhe non presenti come “[empty]”.

Output:

Se la stringa risultante è più lunga di 20 caratteri deve eliminare gli spazi e riscriverla in camelCase se incontra uno spazio tra le stringhe concatenate.

3. Identificare le partizioni

CLASSI DI INPUT

	listOfLeftStrings parameter:
1	lista null
2	lista con string empty
3	lista empty
4	lista not empty
5	lista che contenga una stringa con un numero seguito subito da un carattere
6	lista che contenga una stringa con un numero preceduto subito da un carattere
7	lista con numero di stringhe > N_MAX

	listOfRightStrings parameter:
1	lista null
2	lista con string empty
3	lista empty
4	lista not empty
5	lista che contenga in una stringa la parol “no”
6	lista che contenga in una stringa il carattere "[space]"
7	lista con numero di stringhe > N_MAX

	Combinations of input
1	length (listOfLeftStrings) > length (listOfRightStrings)
2	length (listOfLeftStrings) < length (listOfRightStrings)
3	length (listOfLeftStrings) == length (listOfRightStrings)
4	length (listOfLeftStrings) > N_MAX and length (listOfRightStrings) < N_MAX
5	length (listOfLeftStrings) < N_MAX and length (listOfRightStrings) > N_MAX
6	1<=i<=N_MAX (length(listOfLeftStrings.get(i))+length(listOfRightStrings.get(i)) > 20

CLASSI DI OUTPUT

List **listOfConcatenatedStrings:**		**ogni stringa listOfConcatenatedStrings:**	
1	Null	1	-
2	lista con string empty	2	string empty (“”)
3	lista empty	3	-
4	lista not empty	4	stringa concatenata
5	IllegalArgumentException	5	-

4. Boundary cases

```
// Verifica se le liste hanno lunghezza maggiore di n
if (listLeft.size() > N_MAX || listRight.size() > N_MAX) {
    throw new IllegalArgumentException("Le liste non possono avere dimensione maggiore di " + N_MAX);
}
```

-

1. on-point: 15
2. off-point: 14

```
// Verifica se le liste hanno lunghezza maggiore di n
if (listLeft.size() > N_MAX || listRight.size() > N_MAX) {
    throw new IllegalArgumentException("Le liste non possono avere dimensione maggiore di " + N_MAX);
}
```

-

3. on-point: 15
4. off-point: 14

- length(listOfLeftStrings) == 0

- length(listOfRightStrings) == 0

```
String concatenatedString = leftString + rightString;

// Se la lunghezza della stringa risultante è superiore a 20 caratteri, trasformala in camel case
if (concatenatedString.length() > 20) {
```

-

5. on-point: 20
6. off-point: 21

```
// Verifica se una delle liste è nulla
if (listOfLeftStrings == null || listOfRightStrings == null) {
```

7. on-point: null
8. off-point: 0

```
// Verifica se una delle liste è nulla
if (listOfLeftStrings == null || listOfRightStrings == null) {
```

9. on-point: null
10. off-point: 0

```
if(sizeListRight>sizeListLeft) {
```

11. on-point:
    length (listOfLeftStrings) < length (listOfRightStrings)
12. off-point:
    length (listOfLeftStrings) == length (listOfRightStrings)

```
else if(sizeListRight<sizeListLeft){
```

13. on-point
    length (listOfLeftStrings) > length (listOfRightStrings)
14. off-point
    length (listOfLeftStrings) == length (listOfRightStrings)

5. Fornire casi di test

casi eccezionali

- **T1:** listOfRightStrings null
- **T2:** listOfLeftStrings null
- **T3:** stringa in listOfLeftStrings vuota
- **T4:** stringa in listOfRightStrings vuota
- **T5:** listOfRightStrings vuota
- **T6:** listOfLeftStrings vuota

liste di lunghezza 1

- **T7:** liste di lunghezza 1

casi speciali di listOfLeftStrings

- **T8:** listOfLeftStrings contenente numero che segue un carattere e contenente un numero che precede un carattere

casi speciali di listOfRightString

- **T9:** listOfRightStrings contenente stringhe rispettivamente con i caratteri “no” e “ “

liste di lunghezza differente

- **T10:** length(listOfLeftStrings)<length(listofRightStrings)
- **T11:** length(listOfLeftStrings)>length(listofRightStrings)

boundary cases

- **T12:** length(listOfLeftStrings)+length(listOfLeftStrings)=20
- **T13:** length(listOfLeftStrings)+length(listOfLeftStrings)=21
- **T14:** length(listOfLeftString)=14
- **T15:** length(listOfRightString)=14
- **T16:** length(listOfLeftString)=15
- **T17:** length(listOfRightString)=15

6. Esecuzione casi di test

BUG REPORTED

T8: da specifica nelle stringhe della lista di destra i numeri presenti dovrebbero essere staccati dai caratteri adiacenti. Ma in questo caso di test è stato trovato un bug: il programma non inserisce lo spazio per dividere un numero da un carattere adiacente se quest’ultimo segue il numero.

7. Aumentare la test suite con creatività ed esperienza

T18: listOfLeftStrings contenente stringhe rispettivamente con “-1”, “ 1-” e “-1-”.

T19: listOfLeftStrings contenente stringhe rispettivamente con “!1”, “ 1!” e “!1!”.

T20: listOfLeftStrings contenente stringhe rispettivamente con “@1”, “ 1@” e “@1@”.

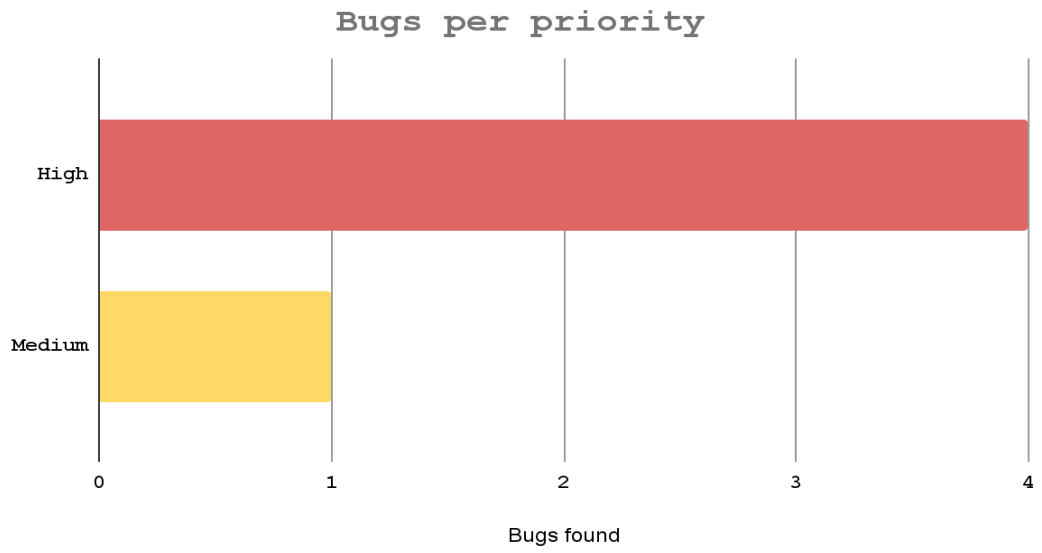
T21: listOfRightStrings contenente stringhe rispettivamente con “ no”, “no “, “no[stringa]” e “[stringa]no[stringa]”.

T22: listOfLeftStrings contenente una stringa con *Integer.MAX_VALUE* preceduto da un numero.

BUGS REPORTED

T18/T19/T20: Da specifica nella lista delle stringhe di sinistra se un qualsiasi carattere è adiacente ad un numero il programma dovrebbe inserire uno spazio. Ma in questo caso di test è stato trovato un bug: i caratteri speciali non vengono considerati dal metodo, quindi tutte in tutte le stringhe contenenti caratteri speciali adiacenti ad un numero il carattere “[space]” non viene inserito

T21: Da specifica, dato un qualsiasi numero contenuto in una stringa di listOfLeftStrings, questo dovrebbe essere distanziato dalla restante stringa con uno spazio. In questo caso di test è stato trovato un bug: se il numero è superiore al massimo intero rappresentabile secondo lo standard IEEE 754, tale numero non viene riconosciuto e quindi il metodo non inserisce lo spazio tra il numero e la restante stringa.



SECONDA FASE:

structural-based-testing(white Box)

METODO DI CODE COVERAGE UTILIZZATO

branch + condition coverage

TOOL DI CODE COVERAGE UTILIZZATO

JaCoCo

Report presente nella directory .\Reports\index.html

Dopo aver finito la fase di specification-based test sono stati eliminati i tre bug trovati nel codice, tutti derivanti dal metodo *addSpaceToNumberInString()*.
Che precedentemente conteneva questo codice.

```
private static String addSpaceToNumberInString(String inputString) {
    // Aggiunge uno spazio tra una parola e un numero attaccato senza spazio
    return inputString.replaceAll("([a-zA-Z])(\\d)", "$1 $2");
```

Successivamente è stata eseguita la code-coverage relativa alla classe di test ed è stato verificato il raggiungimento del 97% della copertura del codice secondo il tool di coverage scelto.

Il branch mancante riguardava questa riga di codice:

```
if (j<inputString.length()-1 && !(Character.isDigit(inputString.charAt(j + 1))) && inputString.charAt(j+1)!=' ') {
```

non veniva mai riscontrata la terza condizione come vera, cioè quella riguardante il controllo del carattere '[space]' nella posizione della successiva.Per questo è stato effettuato un altro test.

T23: listOfLeftStrings contenente una stringa con "1 ".

Successivamente è stata rieseguita la code-coverage.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
ConcatenateLists()		0%		n/a	1	1	1	1	1	1
addSpaceToNumberInString(String)		100%		100%	0	9	0	11	0	1
concatenateLists(List, List)		100%		100%	0	7	0	16	0	1
addEmptyStringsToMinorList(List, List)		100%		100%	0	6	0	17	0	1
toCamelCase(String)		100%		100%	0	2	0	5	0	1
removeSpacesAndNoFromRightString(String)		100%		n/a	0	1	0	1	0	1
Total	3 of 258	98%	0 of 40	100%	1	26	1	51	1	6

Nonostante il tool di code coverage segnalasse quasi il 100% di copertura (dovuto alla mancata copertura della sola riga del nome della classe che definiva il metodo principale), dopo aver letto attentamente il codice si è ritenuto necessario effettuare altri due test:

T24: camel case effettuato su un carattere speciale.
T25: camel case effettuato su un carattere numerico.

HOMEWORK NUMERO 2

1. Descrizione della classe da testare

Questo codice Java definisce una classe chiamata *VerifyOrdinatedString*, che fornisce un metodo statico chiamato *verifyOrdinatedString*. Il metodo accetta una stringa (`str`) e due indici (firstIndex e lastIndex) come input e restituisce una sottostringa della stringa di input in base agli indici forniti. La classe include anche tre metodi privati per verificare se una sottostringa è ordinata in ordine crescente, decrescente o se è un palindromo.

Il **metodo principale** *verifyOrdinatedString* esegue diverse operazioni in base ai valori degli indici forniti:

- Se *firstIndex è negativo*, viene estratta una sottostringa dalla posizione firstIndex alla posizione lastIndex. Se questa sottostringa è un palindromo, viene restituita.
- Se *firstIndex è positivo* (compreso lo 0 che non ha segno) *e pari*, viene estratta una sottostringa dalla posizione firstIndex alla posizione lastIndex. Se questa sottostringa è ordinata in ordine crescente, viene restituita.
- Se *firstIndex è positivo e dispari*, viene estratta una sottostringa dalla posizione firstIndex alla posizione lastIndex. Se questa sottostringa è ordinata in ordine decrescente, viene restituita.
- *altrimenti* se la sottostringa data in input non rispetta nessuno dei precedenti vincoli, qualsiasi sia il firstIndex, il metodo restituisce la stringa data in input.

  In caso di *input non valido* (stringa vuota, null, indici non validi), il metodo genera un'eccezione di tipo `IllegalArgumentException` con un messaggio appropriato.

2. Come abbiamo testato il codice

In questo caso la classe da testare VerifyOrdinatedString gestisce diversi scenari a seconda dei valori di firstIndex e lastIndex.
Secondo l' approccio property-based testing verrà generata automaticamente una vasta gamma di input, difficilmente generabili con seguendo l'approccio example based, vista la variabilità degli input in questo codice. Per generare i metodi di test è stato utilizzato il framework JQWIK per i property-based e JUNIT per gli example-based

3. Come abbiamo operato

L'approccio utilizzato è stato il seguente: è stato suddiviso il codice per partizioni di output ottenibili ed a seconda di queste sono stati creati i metodi della test-suite.

Le possibili partizioni di output sono:

- IllegalArgumentException;
- Sottostringa in ordine ASCII crescente;
- Sottostringa in ordine ASCII decrescente;
- Sottostringa palindroma.

4. Example based tests

Questo tipo di test è stato utilizzato solo per testare gli input che generassero un'eccezione di tipo IllegalArgumentException all'interno del codice.
Gli input con cui i tests sono stati eseguiti sono i seguenti:

T1. firstIndex == lastIndex
(per testare l'onPoint della condizione firstIndex >= lastIndex);

T2. firstIndex *(-1) == lastIndex
(per testare l'onPoint della condizione firstIndex >= lastIndex);

T3. lastIndex > length(inputString);

T4. inputString vuota;

T5. inputString null.

5. Property based tests

Questo tipo di test è stato utilizzato per testare la maggior parte del codice del metodo principale verifyOrdinatedString.

Per effettuare questo tipo di test è stata creata una classe statica ***Utils*** annidata all'interno della test suite. La classe in questione contiene due metodi e una classe:

- **InputObject**
  Questo oggetto è stato introdotto nella test suite per utilizzare il metodo Combinators.combine() utilizzato insieme al metodo .filter() per andare a generare i corretti input, quindi effettuare i test del metodo principale.

- **checkStringLength(firstIndex, stringLength)**
  Questo metodo prende in input firstIndex e la lunghezza della stringa arbitraryString generata in ogni Provide di ciascun metodo di test. Il metodo in questione viene utilizzato all'interno di un filtro per generare oggetti di tipo InputObject con una baseString di lunghezza maggiore di firstIndex.

- **checkNotOrderedString(inputString)**
  Questo metodo prende in input una stringa per verificare che non sia in ordine ASCII crescente o decrescente.

6. Descrizione dei tests property-based effettuati

Tutti i seguenti test hanno un proprio metodo Provide che genera i dati di input per la funzione da testare.

L'approccio nel generare gli input del metodo da testare è stato il seguente:

1) viene generata una serie di stringhe arbitraryString;
2) viene generata una serie di stringhe injectedString che nei vari metodi di test verranno ordinate in base alla determinata partizione da testare. Successivamente ogni injectedString verrà iniettata nella stringa arbitraryString a partire dalla posizione firstIndex;
3) viene generato l'intero firstIndex;
4) vengono combinati tramite il metodo Combinators.combine i dati precedenti nell'oggetto Utils.InputObject, applicando il metodo filter() per verificare che ogni firstIndex generato sia minore uguale della lunghezza della corrispettiva arbitraryString, più altri eventuali vincoli in base al test effettuato.

I PROPERTY-BASED TEST EFFETTUATI

T6. *Stringa restituita in ordine crescente*

Per questo test oltre agli altri arbitrary già definiti sopra, vengono generati firstIndex pari.

```
private Arbitrary<String> getArbitraryString(){
    return Arbitraries.strings().alpha().ofMinLength(0).ofMaxLength(20);
}

new *
private Arbitrary<String> getInjectedString(){
    return Arbitraries.strings().alpha().ofMinLength(1).ofMaxLength(20);
}
//T6
nicolabalzano *
@Provide
private Arbitrary<Utils.InputObject> evenIntegerProvider() {
    Arbitrary<String> arbitraryString = getArbitraryString();
    Arbitrary<String> injectedString = getInjectedString();
    Arbitrary<Integer> firstIndex = Arbitraries.integers().greaterOrEqual( min: 0).filter(n -> n % 2 == 0);
    return Combinators.combine(arbitraryString, injectedString, firstIndex).as(Utils.InputObject::new)
            .filter(o -> Utils.checkStringLength(o.getFirstIndex(), o.getBaseString().length()));
}
```

Il metodo Provide è il seguente:

Per costruire la stringa inputString per il metodo principale da testare abbiamo seguito il seguente approccio:

- aggiunti in inputString i caratteri dall' indice 0 all' indice firstIndex della stringa baseString;

- ordinamento di injectedString in ordine crescente secondo la conversione ASCII;
- aggiunta di injectedString in inputString;
- aggiunti in inputString i caratteri restanti della stringa baseString;
- calcolo di lastIndex.

T7. *Stringa restituita in ordine decrescente*

Per questo test oltre agli altri arbitrary già definiti, vengono generati firstIndex dispari.

Per costruire la stringa inputString per il metodo principale l'approccio utilizzato è stato identico al precedente, tranne per l'ordinamento della stringa injectedString seguendo l'ordine ASCII decrescente.

APPROCCIO GENERALE PER STRINGA PALINDROMA (T8/T9)

L'approccio inizialmente utilizzato per generare stringhe palindrome generava le injectedString tramite un filtro, verificava se la stringa fosse identica a se stessa invertita. Essendo questo un caso talmente raro da far sollevare eccezione a jqwik, in relazione al fatto che effettuava più di 10000 verifiche senza riscontro sul filtro applicato, la soluzione è stata quella di costruire manualmente una stringa palindroma concatenando se stessa con la medesima invertita. Durante lo sviluppo siamo però stati obbligati ad effettuare due test differenti in relazione al fatto che il risultato di una stringa di lunghezza casuale concatenata a se stessa abbia sempre lunghezza pari. Le due soluzioni sono state le seguenti.

T8. *Stringa restituita, palindroma e di lunghezza pari*

Per questo test oltre agli altri arbitrary già definiti, vengono generati firstIndex negativi.

Per costruire la stringa inputString per il metodo principale da testare abbiamo seguito il seguente approccio:

- firstIndex = firstindex *(-1);
- aggiunti in inputString i caratteri dall' indice 0 all' indice firstIndex della stringa baseString;
- modifica di injectedString uguale a se stessa più la stessa invertita;
- aggiunta injectedString in inputString;
- aggiunti in inputString i caratteri restanti della stringa baseString;
- calcolo di lastIndex.

T9. Stringa restituita palindroma e di lunghezza dispari

Per questo test oltre agli altri arbitrary già definiti, vengono generati firstIndex negativi.

Per costruire la stringa inputString per il metodo principale da testare abbiamo seguito il seguente approccio:

- firstIndex = firstindex *(-1);

- aggiunti in inputString i caratteri dall' indice 0 all' indice firstIndex della stringa baseString;
- calcolo della stringa invertita nella seconda metà di injectedString senza l'ultimo carattere
- modifica di injectedString uguale a se stessa più la stringa calcolata al punto precedente
- aggiunta injectedString in inputString;
- aggiunti in inputString i caratteri restanti della stringa baseString;
- calcolo di lastIndex.

T10. Restituita stringa data in input

Questo test è stato creato per andare a verificare tutti gli altri casi in cui la sottostringa non sia né ordinata né palindroma.

Per effettuare questo test non è stato necessario utilizzare una injectedString, per questo motivo la stringa inputString corrisponde esattamente alle arbitraryString con il vincolo di uniqueChars() per far sì che non venga mai generata una stringa palindroma. Inoltre in questo metodo per generare gli input viene utilizzato anche il metodo Utils.checkNotOrderedString() per far si che non venga mai generata una stringa in ordine ASCII crescente o decrescente.

Il metodo Provide è il seguente:

```
@Provide
private Arbitrary<Utils.InputObject> noSpecialSubStringProvider() {
    Arbitrary<String> arbitraryString = Arbitraries.strings().alpha().uniqueChars().ofMinLength(0).ofMaxLength(20);
    Arbitrary<String> injectedString = Arbitraries.strings().ofLength(0);
    Arbitrary<Integer> firstIndex = Arbitraries.integers();
    return Combinators.combine(arbitraryString, injectedString, firstIndex).as(Utils.InputObject::new)
            .filter(o ->
                    (Utils.checkStringLength(o.getFirstIndex(), o.getBaseString().length()) ||
                            Utils.checkStringLength( firstIndex: o.getFirstIndex()*-1, o.getBaseString().length())
                    ) &&
                            (Utils.checkNotOrderedString(o.getBaseString(), o.getFirstIndex()))
            );
}
```

7. Rilassamento vincoli

Dopo esserci assicurati che tutti i metodi di test generassero gli input corretti che andassero tutti a buon fine, abbiamo rilassati i vincoli riguardanti la generazione delle stringhe arbitraryString e injectedString nel modo seguente.

nicolabalzano
```
private Arbitrary<String> getArbitraryString(){
    return Arbitraries.strings().withCharRange('!', '~').ofMinLength(0).ofMaxLength(50);
}
```

nicolabalzano
```
private Arbitrary<String> getInjectedString(){
    return Arbitraries.strings().withCharRange('!', '~').ofMinLength(1).ofMaxLength(50);
}
```

8. Esecuzione code coverage

METODO DI CODE COVERAGE UTILIZZATO

branch + condition coverage

TOOL DI CODE COVERAGE UTILIZZATO

JaCoCo

Report presente nella directory .\Reports\index.html

Eseguendo la code-coverage è stato riscontrato quasi il 100% di copertura del codice della classe, dovuto alla mancata copertura della riga riguardante il nome della classe che definisce il metodo principale, mentre tutti i metodi in essa avevano ciascuno il 100% di copertura.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
VerifyOrdinatedString()		0%		n/a	1	1	1	1	1	1
verifyOrdinatedString(String, int, int)		100%		100%	0	11	0	15	0	1
isStringSorted(String, String)		100%		100%	0	3	0	4	0	1
isStringSortedReverse(String, String)		100%		100%	0	3	0	4	0	1
isPalindrome(String, String)		100%		n/a	0	1	0	1	0	1
Total	3 of 122	97%	0 of 28	100%	1	19	1	25	1	5

9. Generazione e commenti delle statistiche

Le statistiche coerenti generate riguardanti la test-suite che abbiamo individuato, riguardano i valori del parametro firstIndex. In relazione al fatto che jqwik effettua molti test relativi agli edge-cases ci aspettavamo di riscontare un maggior numero di test concentrati su quei casi. Nei vari test effettuati gli edge-cases riguardanti firstIndex sono i seguenti:

- Per quanto riguarda il test **T6** ci aspettavamo un numero di conteggio più elevato nel caso in cui firstIndex == 0 poiché è il primo numero accettato per quella

variabile. Inoltre ci aspettavamo anche che tutti i firstIndex fossero positivi e pari, ed il conteggio riguardante il numero contenuto nei firstIndex fosse decrescente. Quanto detto è ben visibile all'interno della statistica generata

#	label	count	
0	0	220	##
1	2	142	##
2	4	128	##
3	6	124	###
4	8	55	#####################
5	10	42	###############
6	12	47	##################
7	14	50	###################
8	16	42	###############
9	18	35	############
10	20	31	##########
11	22	28	##########
12	24	15	#####
13	26	9	###
14	28	3	#
15	30	5	#
16	32	2	
17	34	4	#
18	36	3	#
19	38	5	#
20	40	2	
21	42	2	
22	44	1	
23	46	2	
24	48	3	#

- Per quanto riguarda il test **T7** ci aspettavamo un numero di conteggio più elevato nel caso in cui firstIndex == 1 poiché è il primo numero accettato per quella variabile. Inoltre ci aspettavamo anche che tutti i firstIndex fossero positivi e dispari, ed il conteggio riguardante il numero contenuto nei firstIndex fosse decrescente. Quanto detto è ben visibile all'interno della statistica generata.

```
  # | label | count |
----|-------|-------|-------------------------------------------------------------------------------
  0 |     1 |   163 | ################################################################################
  1 |     3 |   146 | ########################################################################
  2 |     5 |   139 | ###################################################################
  3 |     7 |   164 | ################################################################################
  4 |     9 |    56 | ############################
  5 |    11 |    58 | #############################
  6 |    13 |    47 | ########################
  7 |    15 |    50 | #########################
  8 |    17 |    35 | ##################
  9 |    19 |    46 | #######################
 10 |    21 |    25 | #############
 11 |    23 |    26 | #############
 12 |    25 |    19 | #########
 13 |    27 |     3 | #
 14 |    29 |     2 |
 15 |    31 |     4 | #
 16 |    33 |     4 | #
 17 |    37 |     2 |
 18 |    39 |     4 | #
 19 |    41 |     3 | #
 20 |    43 |     4 | #
```

- Per i test **T8** e **T9** ci aspettavamo inveceun numero di conteggio più elevato nel caso in cui firstIndex == -1 poiché è il primo numero accettato per quella variabile. Inoltre ci aspettavamo che tutti i firstIndex fossero negativi, e il conteggio riguardante il numero contenuto nei firstIndex fosse decrescente. Avendo i test il medesimo metodo di Provide ci aspettavamo delle statistiche anch'esse quasi identiche. Quanto detto è ben visibile all'interno della statistica generata.

Statistica del test T8

#	label	count	
0	-44	2	##
1	-43	1	#
2	-42	1	#
3	-41	1	#
4	-40	1	#
5	-39	4	####
6	-38	1	#
7	-36	2	##
8	-35	3	###
9	-34	5	#####
10	-33	3	###
11	-32	5	#####
12	-30	3	###
13	-29	4	####
14	-28	2	##
15	-27	6	######
16	-26	15	###############
17	-25	7	#######
18	-24	13	#############
19	-23	15	###############
20	-22	17	#################
21	-21	20	####################
22	-20	18	##################
23	-19	20	####################
24	-18	22	######################
25	-17	24	########################
26	-16	35	###################################
27	-15	30	##############################
28	-14	23	#######################
29	-13	27	###########################
30	-12	25	#########################
31	-11	30	##############################
32	-10	26	##########################
33	-9	68	###
34	-8	42	##
35	-7	69	##
36	-6	67	##
37	-5	67	##
38	-4	60	##
39	-3	70	###
40	-2	73	##
41	-1	73	##

Statistica del test T9

#	label	count	
0	-48	2	##
1	-45	1	#
2	-43	1	#
3	-42	1	#
4	-41	1	#
5	-40	2	##
6	-39	1	#
7	-38	1	#
8	-36	2	##
9	-35	2	##
10	-34	1	#
11	-33	2	##
12	-32	1	#
13	-31	1	#
14	-30	9	##########
15	-29	8	#########
16	-28	5	#####
17	-27	4	####
18	-26	15	###############
19	-25	11	###########
20	-24	15	###############
21	-23	14	##############
22	-22	20	####################
23	-21	15	###############
24	-20	17	#################
25	-19	28	############################
26	-18	22	######################
27	-17	21	#####################
28	-16	26	##########################
29	-15	18	##################
30	-14	29	#############################
31	-13	19	###################
32	-12	29	#############################
33	-11	31	###############################
34	-10	37	#####################################
35	-9	62	##
36	-8	60	##
37	-7	64	##
38	-6	53	###
39	-5	69	###
40	-4	62	##
41	-3	70	###
42	-2	73	##
43	-1	75	##

- Per quanto riguarda il test T10 ci aspettavamo un numero di conteggio più elevato nel caso in cui firstIndex == 0, e che l'istogramma disegnasse una distribuzione gaussiana il cui apice fosse in un intorno del punto 0, questo è dovuto al fatto che in questo test vengono generati firstIndex sia positivi che negativi.

#	label	count	
0	-16	1	#
1	-15	3	###
2	-14	5	#####
3	-13	7	#######
4	-12	8	########
5	-11	15	###############
6	-10	13	#############
7	-9	20	####################
8	-8	53	##
9	-7	47	###
10	-6	47	###
11	-5	43	###
12	-4	50	##
13	-3	54	##
14	-2	51	##
15	-1	63	##
16	0	79	###
17	1	53	##
18	2	58	##
19	3	56	##
20	4	53	##
21	5	48	##
22	6	43	##
23	7	45	##
24	8	22	######################
25	9	13	#############
26	10	13	#############
27	11	9	#########
28	12	9	#########
29	13	6	######
30	14	7	#######
31	15	5	#####
32	17	1	#