Übungsblatt 10 – Ausgabe am 29.06.2020

Abgabe bis 06.07.2020, 12 Uhr, per Mail

Aufgabe 10.1: Kontrollflussbasierte Testverfahren

Gegeben sei die folgende Methode `matrixSum`:

```
/**
 * Berechnet die Summe einer fest gegebenen 4x4 Matrix bis zu der übergebenen
 * Position. Zusätzlich wird ein Summenlimit übergeben, welches nicht über-
 * schritten werden darf.
 * Das Parsen der Matrix wird beim letzten (4,4) Element der Matrix begonnen.
 *
 * @param rowMax
 *            Zeilenindex der Matrix, bis wohin aufsummiert wird [1..4]
 * @param colMax
 *            Spaltenindex der Matrix, bis wohin aufsummiert wird [1..4]
 * @param maxSum
 *            Obergrenze der zu berechnenden Summe
 * @return Summe aller enthaltenen Matrixelemente
 */
55 public int matrixSum(int rowMax, int colMax, int maxSum) {
56     // 4x4 Matrix mit Werten wird fest definiert
57     final int[][] fixedMatrix = { { 1, 2, 5, 6 }, { 3, 4, 7, 8 },
58                   { 9, 10, 13, 14 }, { 11, 12, 15, 16 } };
59     boolean stopp = false;
60     int row = rowMax - 1, column = colMax - 1;
61     int sum = 0;
62     // Durchlauf der Spalten
63     while ((row >= 0) && (row <= 3) && (stopp == false)) {
64           // Durchlauf der Zeilen
65           while (!((column < 0) || (column >= 4) || (stopp == true))) {
66                 sum += fixedMatrix[row][column];
67                 // Ist bereits das Summenlimit überschritten?
68                 if (sum > maxSum) {
69                       sum -= fixedMatrix[row][column];
70                       stopp = true; // Schleife wird gestoppt
71                 }
72                 column--;
73           }
74           row--;
75           // Zurücksetzen des Spaltenindex für nächste Iteration
76           column = colMax - 1;
77     }
78     return sum;
79 }
```

Geben Sie, *sofern möglich*, jeweils eine minimale Menge[1] von Testfällen an, so dass

a) Anweisungsüberdeckung (C_0-Test)
b) Zweigüberdeckung (C_1-Test)
c) Atomare Bedingungsüberdeckung
d) Minimale Mehrfachbedingungsüberdeckung
e) Boundary Interior Test
f) Modifizierter Boundary Interior Test

[1] Möglichst wenig Testfälle

erfüllt sind, oder begründen Sie andernfalls, warum das Testkriterium nicht erfüllt werden kann. Geben Sie zu jeder Teilaufgabe die vom Testfall besuchten Pfade an. Hierfür können Sie den gegebenen Kontrollflussgraphen aus *Abbildung 1* verwenden. Sollte ein Kriterium unzumutbar viele Testfälle erfordern, brauchen Sie diese nicht aufzuführen; begründen Sie dies mit einer ausführlichen Berechnung der Anzahl der Testfälle.

Hinweis: Geben Sie für jeden Testfall den Methodenaufruf mit entsprechenden Parameterwerten und Erwartungswert der Rückgabe an: `matrixSum(0,0,0) = 0;`

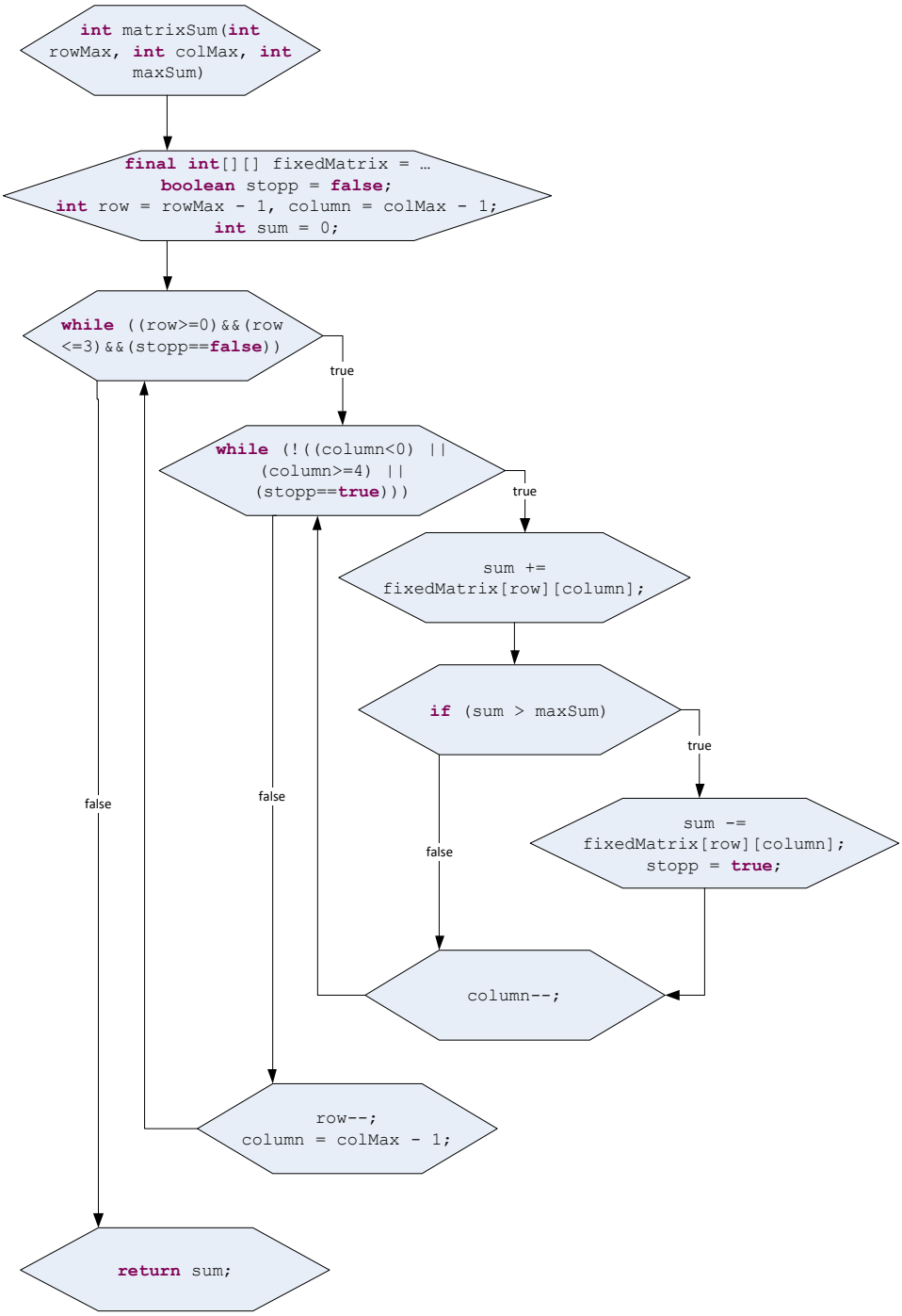

Abbildung 1 Kontrollfluss `positionsssuche`

Aufgabe 10.2: Datenflussbasierte Testverfahren

Gegeben sei die folgende Methode `littleGauss`:

```
01 /** berechne die Gaußsche Summe brute force*/
02 public int littleGauss(int e, int s) {
03       int r=0;
04       // are the numbers positive?
05       if (e > 0) {
06             // calculate
07             while (e != s) {
08                   r += s + e;
09                   e--;
10             }
11             System.out.println("Der Gaußsche Summe ist " + r);
12       } else {
13             r = 0;
14       }
15       return r;
16 }
```

Geben Sie jeweils eine minimale Menge von Testfällen an, so dass die folgenden Kriterien möglichst vollständig erfüllt werden. Geben Sie eine Begründung an, wenn ein Kriterium nur unvollständig erfüllt werden kann. Geben Sie anhand der Kontroll- und Datenflussgraphen in Abbildung 2 und 3 die durch Ihre Testfälle getesteten Datenflusspfade ein.

a) All-defs-Kriterium
b) All-c-uses-Kriterium
c) All-p-uses-Kriterium
d) All-p-some-c-uses Kriterium
e) All-uses-Kriterium

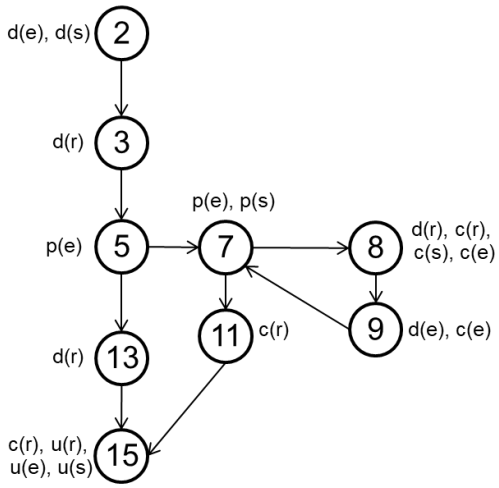

Abbildung 2: Kontrollfluss littleGauss

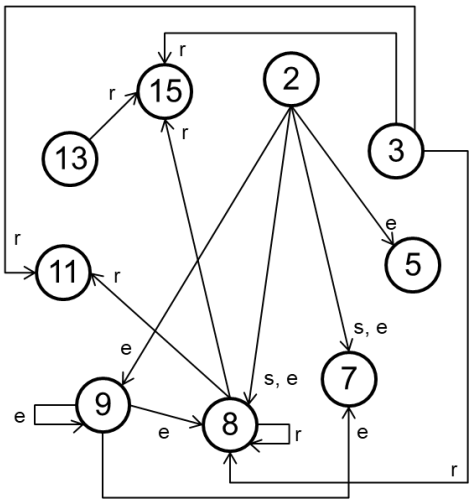

Abbildung 3: Datenfluss littleGauss

Hinweis: Geben Sie für jeden Testfall den Methodenaufruf mit entsprechenden Parameterwerten und Erwartungswert der Rückgabe an: `littleGauss(16,89) = 5037`.