Lehrstuhl für Software Engineering
RWTH Aachen University
Prof. Bernhard Rumpe
Mathias Pfeiffer, M. Sc.
Hendrik Kausch, M. Sc.
Dipl.-Inform. Deni Raco

Softwaretechnik
Übung
WS 2022/23

Aufgabenblatt 9

Abgabe: 21.12.2022 18:30 Uhr

Aufgabe 9.1 (5 Punkte)

Gegeben ist die folgende Methode zur Berechnung des größten gemeinsamen Teilers zweier ganzer Zahlen:

```
    public int gcd(int a, int b) {
01    if (a == 0) {
02      return Math.abs(b);
      }

03    while (b != 0) {
04      int h = a % b;
        a = b;
        b = h;
      }
05    return Math.abs(a);
    }
```

Teilaufgabe a) Kontrollflussgraph erstellen (1 Punkt)

Konstruieren Sie einen Kontrollflussgraphen für die Methode `gcd(int a, int b)`. Benutzen Sie die links vom Methodenrumpf angegebenen Nummern zur Beschriftung der zugehörigen Knoten im Kontrollflussgraphen.

Teilaufgabe b) Anweisungsüberdeckungstest erstellen (1 Punkt)

Nennen Sie eine repräsentative Eingabemenge für einen Anweisungsüberdeckungstest der Methode `gcd(int a, int b)` und geben Sie für jede Eingabe die Reihenfolge der besuchten Knoten im Kontrollflussgraphen an.

Teilaufgabe c) Zweigüberdeckungstest (1 Punkt)

Ist jede repräsentative Eingabemenge für einen Anweisungsüberdeckungstest der Methode `gcd(int a, int b)` auch eine repräsentative Eingabemenge für einen Zweigüberdeckungstest der Methode? Begründen Sie Ihre Antwort.

Teilaufgabe d) Anweisungs- vs. Zweig- Überdeckungstests (1 Punkt)

Warum sind Zweigüberdeckungstests für Kontrollflussgraphen, in denen alle Knoten vom Startknoten aus erreichbar sind, mindestens genau so stark wie Anweisungsüberdeckungstests für diese Kontrollflussgraphen?

Teilaufgabe e) Zweig- vs. Pfad- Überdeckungstests (1 Punkt)

In welchen Fällen ist ein Pfadüberdeckungstest stärker als ein Zweigüberdeckungstest?

Aufgabe 9.2 (5 Punkte)

Die Methode `solveKnapsack(int[] weights, int[] values, int bound)` löst ein Knappsackproblem mit den Gewichten `weights`, Werten `values` und Rucksackkapazität `bound`. Der Wert `weights[i]` repräsentiert das Gewicht des Gegenstands `i`. Analog repräsentiert der Wert `values[i]` den Wert des Gegenstands `i`.

```
   Optional<Integer> solveKnapsack(int[] weights, int[] values, int bound) {
01   if(weights.length != values.length || bound <= 0) {
02     return Optional.empty();
     }
03   int objects = weights.length;
     int[][] r = new int[objects + 1][bound + 1];
04   for(int i = objects - 1; i >= 0; i--) {
05     for(int j = 1; j <= bound; j++) {
06       if(weights[i] <= j) {
07         int valWithI = values[i] + r[i+1][j-weights[i]];
           int valWithoutI = r[i+1][j];
08         if(valWithI > valWithoutI) {
09           r[i][j] = valWithI;
           } else {
10           r[i][j] = valWithoutI;
           }
         } else {
11         r[i][j] = r[i+1][j];
         }
       }
     }
12   return Optional.of(r[0][bound]);
   }
```

Teilaufgabe a) Kontrollflussgraph erstellen (3 Punkte)

Konstruieren Sie einen Kontrollflussgraphen für die Methode `solveKnapsack`. Benutzen Sie die links vom Methodenrumpf angegebenen Nummern zur Beschriftung der zugehörigen Knoten im Kontrollflussgraphen. Nutzen Sie zur Konstruktion des Kontrollflussgraphen die nächste Seite.

Teilaufgabe b) Anweisungsüberdeckungstest (2 Punkte)

Nennen Sie eine repräsentative Eingabemenge mit höchstens drei verschiedenen Eingaben für einen Anweisungsüberdeckungstest der Methode `solveKnapsack` und geben Sie für

jede Eingabe die Reihenfolge der besuchten Knoten im Kontrollflussgraphen an. Für die Angabe der Array-Eingabewerte können sie die übliche Tupelschreibweise nutzen. Beispielsweise repräsentiert das Tupel (1, 2, 3) ein Array A mit `A[0]=1`, `A[1]=2`, `A[2]=3`. Das Tupel () repräsentiert ein leeres Array. Die angegebenen Arrays dürfen jeweils nicht mehr als zwei Elemente enthalten. Sie können zur Angabe der Eingaben folgendes Schema verwenden:

1. Eingabe

   weights =

   values =

   bound =

   Reihenfolge der besuchten Knoten =

2. Eingabe

   weights =

   values =

   bound =

   Reihenfolge der besuchten Knoten =

3. Eingabe

   weights =

   values =

   bound =

   Reihenfolge der besuchten Knoten =