Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 5: Software-Entwicklung 1 (WS 2017/18)

Ausgabe: 20.11.17
Abgabe: 27.11.17

Aufgabe 1 Spezifikationen umsetzen (9 Punkte)

Implementieren Sie die folgenden Prozeduren gemäß ihrer Spezifikation. Verwenden Sie die JUnit Tests aus der Datei `IntersectionTests.java` um Ihre Implementierung zu testen.

Freiwillige Zusatzaufgabe: Implementieren Sie die Prozeduren `subset` *und* `intersection` *so, dass sie mit möglichst wenig Berechnungsschritten auskommen.*

```
public class Intersection {
    /*
    requires a != null
    modifies \nothing
    ensures \result == (es existiert ein int i in [0,a.length-1], so dass a[i] == x)
    */
    public static boolean contains(int[] a, int x) {
        return false;
    }

    /*
    requires a != null
    modifies \nothing
    ensures \result == (fuer alle int i in [0,a.length-2]: a[i] < a[i+1])
    */
    public static boolean increasing(int[] a) {
        return false;
    }

    /*
    requires a != null
          && b != null
          && increasing(a)
          && increasing(b)
    modifies \nothing
    ensures \result == (fuer alle int x: !contains(a, x) || contains(b,x))
    */
    public static boolean subset(int[] a, int[] b) {
        return false;
      }

    /*
    requires a != null
          && b != null
          && increasing(a)
          && increasing(b)
    modifies \nothing
    ensures \result != null
         && (fuer alle int x: (contains(a, x) && contains(b, x)) == contains(\result, x))
         && increasing(\result)
    */
    public static int[] intersection(int[] a, int[] b) {
        return null;
    }
}
```

Aufgabe 2 Spezifizieren und Testen (12 Punkte)

Geben Sie für die folgenden Prozeduren jeweils eine sinnvolle Spezifikation mit Vorbedingungen, Nachbedingungen und gegebenenfalls einer Variablenliste für veränderten Zustand an. Achten Sie darauf, die Vorbedingung so zu wählen, dass keine Exceptions auftreten und die Prozedur terminiert, wenn die Vorbedingung eingehalten wird.

Schreiben Sie außerdem für jede der Prozeduren zwei oder mehr JUnit Testfälle.

Geben Sie eine Datei `Procs.java` ab, in der die Prozeduren mit den Spezifikationen als Kommentaren stehen und eine Datei `ProcsTest.java` in der sich die JUnit Tests befinden.

Hinweis: Im Abschnitt "Testen von Seiteneffekten" finden Sie ein Beispiel, das zeigt, wie Tests auf Implementierungen in anderen Dateien zugreifen können.

```
public class Procs {

    public static int[] take(int[] ar, int n) {
        int[] res = new int[n];
        for (int i=0; i<n; i++) {
            res[i] = ar[i];
        }
        return res;
    }

    public static void reverse(int[] input) {
        for (int i=0; i<input.length/2; i++) {
            int temp = input[i];
            input[i] = input[input.length - 1 - i];
            input[input.length - 1 - i] = temp;
        }
    }

    public static int[] repeat(int[] snippet, int len) {
        int[] res = new int[len];
        int resPos = 0;
        while (resPos < len) {
            for (int i=0; i < snippet.length && resPos < len; i++) {
                res[resPos] = snippet[i];
                resPos = resPos + 1;
            }
        }
        return res;
    }
}
```

Aufgabe 3 Rekursion und Terminierung (16 Punkte)

Geben Sie für die folgenden Prozeduren jeweils eine sinnvolle Vorbedingung an, so dass die Prozedur immer ohne Fehler terminiert, wenn die Vorbedingung erfüllt ist.

Beweisen Sie dann die Terminierung der Prozedur für alle erlaubten Werte. Verwenden Sie dazu das Verfahren aus der Vorlesung.

Hinweise: Sie können den Java Visualizer[1] verwenden, um das Verhalten der Prozeduren zu beobachten. Sollten Sie Probleme haben, eine passende Abstiegsfunktion zu finden, wenden Sie sich an Ihren Tutor.

a)

```
1   static void partition(int[] a, int pivot, int left, int right) {
2       if (left > right) {
3           return;
4       }
5       if (a[left] < pivot) {
6           partition(a, pivot, left+1, right);
7       } else if (a[right] >= pivot) {
8           partition(a, pivot, left, right-1);
9       } else {
10          int t = a[left];
11          a[left] = a[right];
12          a[right] = t;
13          partition(a, pivot, left+1, right-1);
14      }
15  }
```

b)

```
1   static int sum(int[][] a, int i, int j, int n, int m) {
2       if (j >= m || i > n) {
3           return 0;
4       } else if (i == n) {
5           return sum(a, 0, j+1, n, m);
6       } else {
7           return a[i][j] + sum(a, i+1, j, n, m);
8       }
9   }
```

Aufgabe 4 Prozedurabstraktion (11 Punkte)

Laden Sie sich das unten abgedruckte Programm `Histograms.java` herunter.

Das Programm nimmt eine Anzahl von Segmenten als Programm-Parameter und liest dann eine Tabelle mit Bezeichnern und **`int`**-Werten über die Standard-Eingabe ein. Das Programm berechnet dann die Verteilung der Werte und gibt ein Histogramm auf der Konsole aus. Als Beispiel können Sie die Datei `klausur.txt` herunterladen und das Programm wie folgt aufrufen:

```
java Histograms 20 < klausur.txt
```

Das Programm besteht aus einer einzelnen `main`-Prozedur. Daher ist es schwer JUnit Tests für das Programm zu schreiben.

a) Teilen Sie das Programm in mehrere kleinere Prozeduren auf. Beachten Sie dabei die Hinweise zur Abstraktion durch Prozeduren aus der Vorlesung. Geben Sie für diese Teilaufgabe die verbesserte Datei `Histograms.java` ab.

[1] Online unter http://cscircles.cemc.uwaterloo.ca/java_visualize/ oder lokal mit Docker (Befehl `docker run --rm --name java_visualzie --privileged -p 80:80 mweber/java_visualize:v1`)

b) Schreiben Sie mindestens 2 JUnit-Testfälle für jede Prozedur aus Teil a), welche keinen Effekt auf `System.out` hat. Schreiben Sie insgesamt mindestens 10 Testfälle. Geben Sie für die Tests eine zweite Datei namens `HistogramsTest.java` ab.

```
public class Histograms {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Geben Sie die Anzahl der Segmente "
                + "im Histogramm als Programmparameter an.");
            return;
        }

        // Werte von Eingabe lesen (immer Bezeichnung mit Wert)
        String[] input = ReadStrings.readStrings();
        if (input.length < 2) {
            System.out.println("Keine Werte eingegeben.");
            return;
        }

        // Werte aus Eingaben extrahieren (jeder zweite Wert)
        int[] values = new int[input.length / 2];
        for (int i = 0; i < values.length; i++) {
            values[i] = Integer.parseInt(input[i * 2 + 1]);
        }

        // Kleinsten Wert berechnen:
        int min = values[0];
        for (int i = 1; i < values.length; i++) {
            if (values[i] < min) {
                min = values[i];
            }
        }
        // Größten Wert berechnen:
        int max = values[0];
        for (int i = 1; i < values.length; i++) {
            if (values[i] > max) {
                max = values[i];
            }
        }

        // Die Anzahl der Werte für jedes Segment berechnen:
        int[] counts = new int[Integer.parseInt(args[0])];
        // Abstand zwischen min und max:
        int range = (max - min);
        for (int i2 = 0; i2 < values.length; i2++) {
            int section = (values[i2] - min) * Integer.parseInt(args[0]) / (range+1);
            counts[section] = counts[section] + 1;
        }

        // Werte so normalisieren, dass 50 der Maximalwert ist:
        int[] normalizedCounts = new int[counts.length];
        int res = counts[0];
        for (int i = 1; i < counts.length; i++) {
            if (counts[i] > res) {
                res = counts[i];
            }
        }
        int maxCount = res;
        for (int i1 = 0; i1 < counts.length; i1++) {
            normalizedCounts[i1] = counts[i1] * 50 / maxCount;
        }

        // Histogramm als Balken auf der Konsole ausgeben:
        for (int i = 0; i < normalizedCounts.length; i++) {
            int n = normalizedCounts[i];
            for (int j = 0; j < n; j++) {
                System.out.print("|");
            }
            System.out.println();
        }
    }
}
```

Freiwillige Zusatzaufgabe (0 Punkte)

Die folgende Aufgabe können Sie selbstständig bearbeiten und abgeben. Dazu finden Sie im Exclaim eine zusätzliche Übung "SE1WS17Z", wo Sie Ihre Lösung als Programm `Superpowers.java` hochladen können. Die Abgaben werden automatisch getestet, aber nicht bewertet.

Bei der Lösung der Aufgaben kommt es auch auf die Effizienz des Algorithmus an. Mit einer langsamen Lösung werden Sie wahrscheinlich für große Eingaben einen Timeout erhalten.

Superpowers

Science has done it! We unlocked the secret of superpowers! In fact, it is rather easy. Using the patented DNA-Resequencer, we can rewrite the DNA of one lucky person so that he will develop either super strength or the ability to telepathically talk to tomatoes (depending on prior genetic makeup).

To find out which one will happen, the scientists have to count the number of times a specific sequence (which differs for each person) occurs in the subjects DNA. Can you help them with that?

Input

The first line of the input contains an integer t. t test cases follow.

Each test case consists of two lines. The first line contains w, the DNA of a potential subject. The second line contains p, the specific sequence that influences the superpower.

Output

For each test case, print a line containing "Case #i: x" where i is its number, starting at 1 and x is the amount of times p occurs in w. Each line of the output should end with a line break.

Constraints

- $1 \leq t \leq 20$
- $1 \leq |p| \leq |w| \leq 50000$
- Both p and w consist only of the letters "A", "C", "G" and "T"