technische universität dortmund

Fakultät für Informatik
Lehrstühle 2, 7, 11, 13

Prof. Dr. C. Sohler, D. Köppl
R. Kalkreuth, D. Siedhoff, I. Kamehkhosh, M. Ludewig
`http://tiny.cc/dap2praktikum`

Sommersemester 2016

DAP2 Praktikum – Blatt 12

Ausgabe: 22. Juni — Abgabe: 5.–8. Juli

Hinweis zu den Langaufgaben

Für diese Aufgaben stehen Ihnen ca. zwei Wochen Bearbeitungszeit zur Verfügung. In den Praktika dieser Woche können (und sollen) Sie mit der Bearbeitung beginnen. Den Rest müssen Sie außerhalb des Praktikums erledigen. Neben den Lernräumen können Sie dafür auch die Poolräume (z.B. in einer Freistunde) nutzen, wenn dort gerade keine Veranstaltungen stattfinden. Im Praktikum in zwei Wochen präsentieren Sie Ihrem Tutor dann Ihre Lösungen auf Ihrem Praktikumsrechner im Pool und bekommen Punkte dafür (nach Absprache mit dem Tutor ist es auch möglich, die Lösung auf Ihrem eigenen Rechner zu präsentieren). **Wichtig:** Sie müssen die Aufgaben **vor** Beginn des Praktikums, in dem Sie die Lösungen präsentieren, fertig haben. **Sie dürfen und sollen die Langaufgaben in Gruppen von bis zu drei Studierenden bearbeiten und präsentieren!**

Wichtig: Der Quellcode ist natürlich mit sinnvollen Kommentaren zu versehen. Überlegen Sie außerdem in welchen Bereichen Invarianten gelten müssen und überprüfen Sie diese ggf. an sinnvollen Stellen mit *Assertions* (siehe Hinweis auf Blatt 2).

Langaufgabe 12.1 (8 Punkte)

Lernziel: Experimentelle Analyse (KSP-Algorithmus)

Untersuchen Sie den aus der Vorlesung bekannten Algorithmus von Karp, Shenker und Papadimitriou experimentell. Überlegen Sie sich hierzu verschiedenen sinnvolle Szenarien und Kriterien zur Bewertung des Verfahrens. Unter anderem könnten folgende Aspekte von Interesse sein:

- Die Länge des Datenstroms und die Wahl des Parameters ε,
- die Häufigkeitsverteilung der Elemente im Datenstrom,
- die Datenstruktur, mit der die Menge `K` und das Array `count` implementiert wurden.

Gehen Sie zur Analyse wie folgt vor:

(1) Implementieren Sie den KSP-Algorithmus bzw. greifen Sie auf Ihre vorhandenen Implementierung für Blatt 9 zurück und passen Sie diese entsprechend an.

(2) Erzeugen Sie geeigneten Eingabedaten für Ihre Szenarien.

(3) Evaluieren Sie das Verhalten Ihrer Implementierung für unterschiedliche Szenarien.

Dokumentieren Sie Ihre Untersuchungen schriftlich, bereiten Sie die experimentellen Ergebnisse angemessen auf (z. B. mittels Diagrammen) und werten Sie die Resultate für die einzelnen Szenarien aus. Decken sich die Ergebnisse mit Ihren Erwartungen?

Die Hinweise und Tipps auf der folgenden Seite könnten nützlich sein.

Hinweise und Tipps

12.1 Messung von Laufzeiten

Um Laufzeiten in Programmen zu messen, kann man die Methode `System.currentTimeMillis()` benutzen. Diese gibt die Systemzeit in Millisekunden zurück, und hat den Rückgabetyp `long` Ein Beispielcode zur Laufzeitmessung sieht dann wie folgt aus:

```
long tStart = System.currentTimeMillis();        // Beginn der Messung
// Hier wird der Code ausgeführt, dessen Laufzeit gemessen werden soll
long tEnd = System.currentTimeMillis();          // Ende der Messung
long msecs = tEnd - tStart;                      // Vergangene Zeit
```

Beachten Sie, dass der Garbage-Collector einen Einfluss auf die Laufzeit haben kann. Bei einzelnen Messungen kann dies zu recht großen Laufzeitunterschieden führen. Außerdem kann es aufgrund verschiedener Effekte (wechselnde CPU-Auslastung der Poolrechner, wechselnde CPU-Taktungen bei Laptops und anderen Rechnern) zu Schwankungen in der Laufzeit kommen.

Daher ist es empfehlenswert, jede Messung mehrmals zu wiederholen und den Mittelwert der Laufzeiten zu bestimmen. Dem Garbage-Collector kann auch zwischen den Messungen mittels `System.gc()` empfohlen werden den Speicher aufzuräumen. Vorher müssen natürlich die betreffenden Referenzen auf `null` gesetzt sein, damit der Speicher auch wirklich freigegeben werden kann.

12.2 Generieren von Zufallszahlen

In Java steht ein Pseudozufallszahlengenerator zur Verfügung. Die Klasse `java.util.Random` stellt den Konstuktor für einen Pseudozufallszahlengenerator, sowie die Methoden `nextInt()` und `nextDouble()` zur Verfügung.

Siehe auch: `http://docs.oracle.com/javase/7/docs/api/java/util/Random.html`
Für eine andere Implementierung eines Pseudozufallszahlengenerators siehe:
`http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/random/MersenneTwister.html`

12.3 Generieren von Zufallsfolgen

Die folgende Funktion generiert zufällige Folgen der Länge n:

```
String randStr(int n, Random r) {
  String alphabet ="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
  StringBuilder res = new StringBuilder(n);
  while (--n>=0) {
    res.append(alphabet.charAt(r.nextInt(alphabet.length())));
  }
  return res.toString();
}
```