Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Einführung in die Informatik
Sommersemester 2011

Prof. Dr. Wolfram Burgard
PD Dr. Cyrill Stachniss
Dipl. Inf. Axel Rottmann
Dipl. Inf. Markus Kuderer

Übungsblatt 13

Zusatzblatt

Aufgabe 13.1

Die Exponentialfunktion e^x mit der Eulerschen Zahl e als Basis, kann durch folgende Reihe näherungsweise berechnet werden:

$$e^x = \sum_{n=0}^{N} \frac{x^n}{n!}.$$

Schreiben Sie ein Java-Programm, das den Benutzer zur Eingabe der gewünschten Anzahl an Iterationen N sowie einer Zahl x auffordert und anschließend den Wert von e^x ausgibt.

Aufgabe 13.2

1. Schreiben Sie eine rekursive Java-Methode, die die folgende Funktion $f : \mathbb{Z} \to \mathbb{Z}$ ($\mathbb{Z}$ sind die ganzen Zahlen) implementiert:

$$f(x) = \begin{cases} 0 & \text{falls } x = 0 \\ -f(-x) & \text{falls } x < 0 \\ x + f(x-1) & \text{sonst} \end{cases}$$

2. Zeichnen Sie die Activation Records für den Aufruf `f(-3)` zu dem Zeitpunkt, an dem die maximale Rekursionstiefe erreicht ist.

Aufgabe 13.3

Bei der Arbeit mit Robotern oder anderen Sensorplattformen ist es hilfreich, empfangene Daten in einem sogenannten Logfile zu speichern. Häufig kommen dabei menschenlesbare Textdateien zum Einsatz.

Gehen Sie davon aus, dass Sie einen Roboter haben, der seine aktuelle Position einerseits anhand der Umdrehungen seiner Räder messen kann (sog. Odometriemessungen) und zusätzlich einen GPS Empfänger besitzt. Des Weiteren ist der Roboter mit einem Lagesensor (sog. IMU) ausgestattet. Dieser Sensor besteht aus drei Beschleunigungssensoren, drei Drehratensensoren sowie drei Magnetfeldsensoren und dient zur Lagebestimmung des Roboters.

Jede Sensormessung, auch Nachricht genannt, besitzt eine eindeutige Zeichenkette (Bezeichner) zur Identifikation der Nachricht (z.B. ODOM, GPS, IMU) sowie einen Zeitstempel. Der Zeitstempel speichert dabei die vergangenen Sekunden seit Start des Systems.

Jede Nachricht hat das folgende Format: `Bezeichner Daten Zeit` wobei die `Daten` erst durch die Nachricht selbst spezifiziert werden. Jede Nachricht steht in einer neuen Zeile der Textdatei.

Es existieren die folgenden Nachrichten im angegebenen Format:

- `ODOM X Y Orientierung Zeitstempel`
- `GPS Latitude Longitude Altitude Zeitstempel`
- `IMU AccX AccY AccZ GyroX GyroY GyroZ MagX MagY MagZ Zeitstempel`

Beispiel eines Logfiles:

```
ODOM 0.000000 0.000000 0.000000 0.000000
IMU 0.356737 -0.459438 9.689002 -0.000768 0.714784 -1.577571 0.005957
-0.009101 0.021694 0.487513
GPS 48.013792 7.832673 258.100000 0.611695
ODOM 0.105000 -0.002000 0.000000 0.750657
IMU -0.318621 -0.585081 9.575044 0.006109 0.702133 -1.585088 0.001483
-0.032105 0.009289 1.327513
ODOM 0.226000 -0.003000 0.000000 1.520215
GPS 48.013797 7.832680 259.600000 1.635700
IMU -0.326542 -0.228753 9.680017 0.013160 0.709784 -1.582964 -0.005338
-0.003636 -0.021803 2.167513
```

Ihre Aufgabe ist es nun ein System zum Einlesen, Verarbeiten und Schreiben solcher Logfiles zu erstellen. Nutzen Sie dabei Techniken wie Klassenhierarchie, Vererbung, Polymorphismus, etc.

1. Geben Sie eine Klassenhierarchie für die Sensor-Nachrichten an, in dem Sie diese als Graphen visualisieren. Zeichnen Sie ein Rechteck für jede Klasse und einen Pfeil für jede Vererbung (jeweils von der Subklasse zur Superklasse).
   Hinweis: Verwenden Sie eine abstrakte Klasse `SensorReading`.

2. Setzen Sie Ihre Klassenhierarchie in Java um und implementieren Sie alle benötigten Funktionen zum Einlesen und Verarbeiten der Daten. Hinweis: Gehen Sie davon aus, dass jeweils eine komplette Nachricht (eine Zeile des Logfiles) als String übergeben wird.

   Beispiel: `public void readData(String str)`

3. Implementieren Sie das Interface `Comparable<Sensor>` für Ihre `SensorReading`-Klasse. Die Vergleichsmethode soll dazu verwendet werden, ein Logfile nach dem Zeitstempel zu sortieren.

4. Schreiben Sie eine Klasse `SortedSingleLinkedList`, die eine sortierte, einfach verkettete Liste von Sensor-Nachrichten darstellt. Beim Einfügen einer Nachricht soll die Position entsprechend der Ordnungsrelation bestimmt werden, die durch die `compareTo`-Methode gegeben ist.

5. Implementieren Sie für die `SortedSingleLinkedList`-Klasse das Interface `Iterator`.

6. Schreiben Sie eine Methode, die alle Nachrichten des Logfiles `logfile.log`[1] einliest und in einer `SortedSingleLinkedList` speichert.

7. Welche Distanz hat der Roboter, basierend auf den Odometriemessungen, zurückgelegt? Die Daten der Odometrie sind absolute Werte der Position bzgl. der Startposition in Metern.

8. Wo befand sich der Roboter während des Experiments und welche Trajektorie hat er zurückgelegt. Erzeugen Sie dazu eine `.kml` Datei bestehend aus den GPS Daten und visualisieren Sie diese in Google Earth[2] oder auf

   `http://gpsvisualizer.com/`

   Verwenden Sie dazu die Klasse `KML` von der Vorlesungshomepage.

Aufgabe 13.4

Betrachten Sie den folgenden Algorithmus:

```
public static void do_something(int[] x) {
   if (x.length < 2)
      return;

   boolean has_swapped = true;

   while(has_swapped) {
      has_swapped = false;

      for(int i = 0; i < x.length - 1; i++) {
         int current = x[i];
         int next = x[i+1];

         if (next < current) {
            swap(i, i+1, x);
            has_swapped = true;
         }
      }
   }
}

public static void swap(int i, int j, int[] x) {
   int xi = x[i];
   x[i] = x[j];
   x[j] = xi;
}
```

[1] siehe Vorlesungshomepage
[2] http://earth.google.de/

1. Geben Sie die Komplexität der Methode `swap` an und begründen Sie ihre Antwort.
2. Schätzen Sie den Aufwand der Methode `do_something` für der Best- und Worst-Case in Abhängigkeit der Länge der Eingabe ab. Begründen Sie Ihre Antwort.
3. Geben Sie für die Methode `do_something` jeweils eine Eingabe (Eingabelänge $n > 5$) für den Best- und Worst-Case an.

Aufgabe 13.5

Entscheiden Sie, ob die folgenden Aussagen richtig oder falsch sind.

	Richtig	Falsch
Mittels Induktion können beliebige Java-Programme auf Korrektheit überprüft werden.	☐	☐
Jedes Java-Programm ist ein Algorithmus.	☐	☐
Jeder Algorithmus ist partiell berechenbar.	☐	☐
Terminierende Algorithmen enden nach endlich vielen Schritten.	☐	☐
Die Menge der Algorithmen ist überabzählbar.	☐	☐
Es gibt Java-Programme, für die entscheidbar ist, ob sie für beliebige Eingaben terminieren.	☐	☐
Alle Probleme lassen sich mit einem Algorithmus lösen.	☐	☐