Übungen Algorithmen und Programmierung 1

WS 2009/10

Mo 14-16, Mi 16-18, Do 10-12 im RTK

Andre Herbst
Jeremias Herrmann
Sabine Storandt

Inhalte

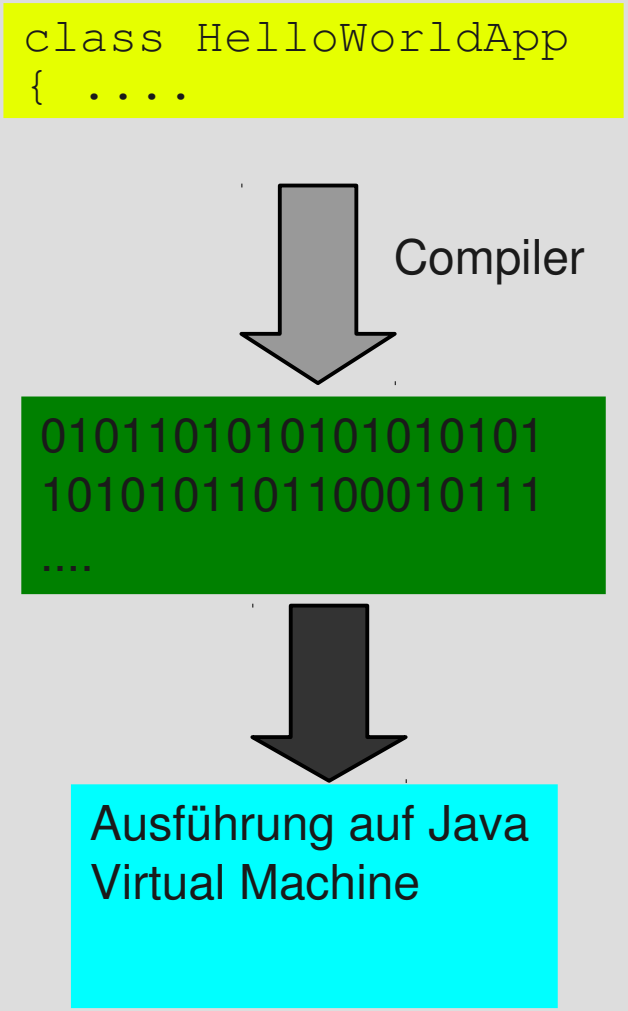
- Einführung in das Programmieren in Java (C++ wäre aber auch nicht viel anders)
- Implementierung einiger Algorithmen und Datenstrukturen, die in der Vorlesung vorgestellt wurden

Ein Beispiel für ein Java Programm

```
public class HelloWorldApp {
    public static void main(String[] args)
{
     System.out.println("Hello World!");
    }
}
```

Was geschieht mit einem solchen Programm?

- Der Rechner kann dieses Sourcefile nicht direkt ausführen
- Es muß zuerst vom Compiler übersetzt werden; im Falle von Java in den sogenannten Bytecode
- Dieser Bytecode wird dann von einer Java Virtual Machine ausgeführt



Beispiel anhand des OpenJDK6 unter Linux

Aufruf des Compilers:

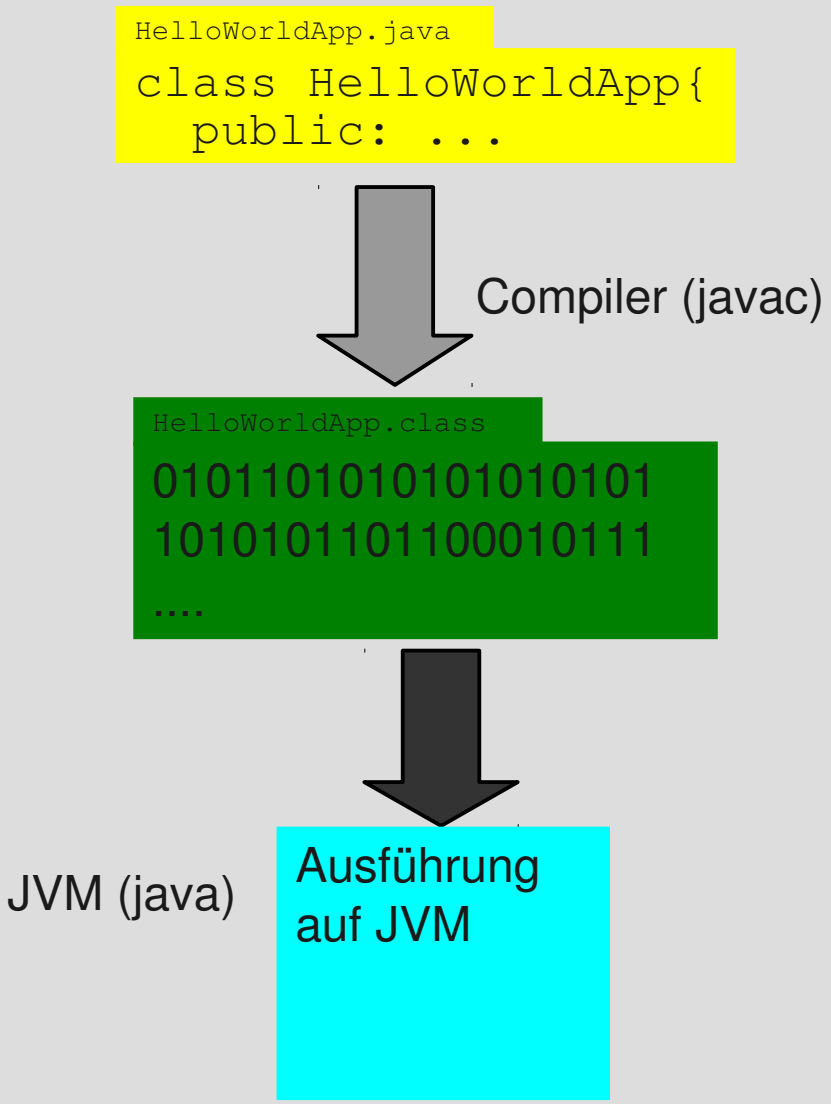
```
javac HelloWorldApp.java
```

erzeugt Java Bytecode

```
HelloWorldApp.class
```

Dieser JBC kann auf einer Java Virtual Machine ausgeführt werden mittels:

```
java HelloWorldApp
```



Vorteile von Java Bytecode

- C, C++, Pascal, … werden typischerweise in plattformspezifischen **Maschinencode** übersetzt, welcher nur auf dieser Architektur lauffähig ist (Beispiel: ein auf einem Intelsystem aus C++ erzeugtes Executable ist z.B. nicht auf einer ARM Plattform lauffähig)
- Java Bytecode ist auf jeder Plattform lauffähig, auf der die sog. **Java Virtual Machine** implementiert ist

Unsere Arbeitsumgebung: LINUX

- Freies UNIX Betriebssystem
- Enthält eine Vielzahl an Entwicklungsumgebungen für C/C++, Java, Pascal, ...
- Es gibt sowohl die Möglichkeit kommandozeilenorientiert zu arbeiten als auch eine IDE (Integrated Development Environment, z.B. Eclipse oder Netbeans, dazu später mehr) zu nutzen
- Multiuser-Betriebssystem: mehrere Benutzer können gleichzeitig auf einem Rechner arbeiten (Remote-Login)

Start von Linux auf Ihrem Rechner im RTK

- Rechner einschalten :-)
- im Bootmanager „Ubuntu Linux“ auswählen
- Einloggen durch Klick auf „UserXX“
- Shell/Terminal starten

Kommandos auf der Shell

ls	Verzeichnisinhalt ausgeben
mkdir	Verzeichnis erstellen
cd	in Verzeichnis wechseln (ohne Argument: in Home-Verz. wechseln)
rmdir	Verzeichnis löschen
rm	Datei löschen
mv	Dateien/Verzeichnisse ver-schieben/umbenennen
man	Hilfe zu Kommando aufrufen
gedit	Texteditor aufrufen

Aufgabe 1

- Erstellen Sie folgende Verzeichnisstruktur
  ```
  NameVorname/
     Trockenuebungen/
     MeinProjekt/
  ```

- Erzeugen Sie in `NameVorname/` eine Datei `Email.txt`, welche Ihre Email Adresse enthält
- Erzeugen Sie in `NameVorname/Trockenuebung/` eine Datei `Changelog.txt`, in welcher Sie dokumentieren, was sie gerade getan haben, z.B. so:
  `2009/10/17: Verzeichnisstruktur erstellt`

Ein minimales Java Programm

```
class HelloWorldApp {
    public static void main(String[]
args) {
     System.out.println("Hello World!");
    }
}
```

Speichern/Übersetzen des Programms

- Tippen Sie das Programm in einem Editor ein und speichern Sie es unter HelloWorldApp.java
- gehen Sie auf die Shell und kompilieren es:
  `javac HelloWorldApp.java`
- schauen Sie nach, was der Compiler produziert hat mit:
  `ls`
- führen Sie das Programm aus mit
  `java HelloWorldApp`

Erläuterungen

- Momentan interessieren uns nur die Zeilen
  `public static void main (String[] args)`
  und
  `System.out.println("Hello World!");`
- die erste Zeile bezeichnet dass der folgende, durch { … } eingeschlossene Codeblock als erstes bei Aufruf in der JVM ausgeführt wird
- die zweite Zeile gibt einfach nur „Hello World!“ auf der Konsole aus

Aufgabe

- Geben Sie anstatt „Hello World“ „Hallo Welt“ aus. Ändern Sie hierzu das Programm, speichern es erneut ab, übersetzen und starten Sie es auf der JVM.

Operationen auf Zahlen

- Schreiben Sie ein Javaprogramm, welches, anstatt „Hello World“ auszugeben, folgende Codesequenz ausführt:

```
System.out.println("Berechne die Summe 1+2+...+99+100");
int sum=0;
for (int i=1; i<=100; i++)
{
    sum=sum+i;
}
System.out.println(sum);
```

Erläuterungen

- Dieses Codebeispiel nutzt neben der Ausgabe folgende neue Konzepte:
  - **Variablen**: im Programm werden die Variablen `i` und `sum` benutzt. In unserem Fall sind sie vom Typ `int` und können eine Ganzzahl (integer) speichern. Variablen vom Typ `int` können zugewiesen, addiert, subtrahiert, ... und auch ausgegeben werden.
    Es gibt noch weitere Variablentypen, die andere Daten speichern können.

Erläuterungen

- **for-Schleife**: die for-Schleife hat die Form

  `for(<Startanweisung>; <Bedingung>; <Endschleifenanweisung>) {Rumpf}`

  - Was passiert dabei?
    - zunächst wird die Startanweisung ausgeführt
    - dann wird getestet, ob die Bedingung erfüllt ist;
      falls **ja**, wird der Rumpf ausgeführt, dann die Endschleifenanweisung und danach wieder die Bedingung getestet (und ggf. wieder der Rumpf …)
      falls **nein**, geht es zur nächsten Anweisung (hinter dem Rumpf); die Schleife wird verlassen

Aufgabe

- Schreiben Sie ein Programm, welches nur die *geraden* Zahlen zwischen 1 und 1000 addiert und die Summe ausgibt

Variablen und Konstanten in Java

- Java unterscheidet verschiedene Datentypen, u.a.:
  - `int` ganze Zahlen, z.B. 7
  - `double` Gleitkommazahlen, z.B. 4.6
  - `String` Zeichenketten

Aufgabe

- Ändern Sie Ihr Programm, sodass es das Produkt der Zahlen 1.5, 2.5, 3.5, 4.5, ..., 22.5 berechnet und ausgibt. Nutzen Sie hierzu das Konstrukt der for-Schleife.
- Versuchen Sie, statt Variablen vom Typ `double`, Variablen vom Typ `int` zu benutzen. Was passiert?

Theorieaufgaben

- Beweisen Sie, dass die Summe
  1+2+3+...+n = n(n+1)/2
  ist.
- Was gilt für die Summe
  1^2+2^2+3^2+4^2+5^2+ … + n^2 ?
  Also z.B. für n=4 haben wir 1+4+9+16=30.
  Beweisen Sie Ihre Vermutung durch Induktion.
- Beschreiben Sie, wie Sie die Zahlenfolge
  2,59,3,10,23,99,1,-20
  mittels des von Ihrer Kommilitonin vorgeschlagenen **Insertion Sort** sortieren würden. Generalisieren Sie Ihre Beschreibung, sodass Sie damit auch n Zahlen sortieren könnten (im Stile eines Kochrezeptes).

Aufgabe: Arbeiten mit Zufallszahlen

- Laden Sie von der Webseite die Datei ZufallsZahlen.java herunter, kompilieren Sie sie und führen Sie das entsprechende Executable aus
- versuchen Sie zu verstehen, was das Programm macht

Erläuterungen

- ein neues und spannendes Sprachkonstrukt in diesem Programm ist ein sogenanntes **Array**; in einem Array können z.B. eine große Anzahl an Zahlen gespeichert werden, ohne sehr viele separate Variablen zu definieren:

```
int [] zahlen;
zahlen=new int[20];
```

erzeugt ein Array von 20 Zahlen, die mit `zahlen[0], zahlen[1], … zahlen[19]` wie „normale“ Variablen angesprochen werden

Erläuterungen

- desweiteren wird zum ersten Mal die sog. if-Anweisung benutzt:
  ```
  if (meineZahlen[i]<threshold]
     countSmall++;
  ```
  testet ob der Wert in `meineZahlen[i]` kleiner dem Wert in `threshold` ist und erhöht ggf. den Wert von `countSmall` um 1. Generell gilt:
  ```
  if (Bedingung)
  { Body }
  ```
  falls die Bedingung erfüllt ist, werden die Anweisungen im Body ausgeführt. Falls der Body nur eine Anweisung enthält, können die geschweiften Klammern weggelassen werden.

Erläuterungen

- die Zeile
  `Random generator=new Random()`
  erzeugt ein „Zufallszahlengenerator-Objekt“;
  generator ist ähnlich einer Variablen vom Typ `int`,
  allerdings hat es eine andere Funktionalität; anstatt die Fähigkeit, eine ganze Zahl zu speichern, spuckt es durch Aufruf von
  `generator.nextInt(`*`limit`*`)`
  eine Zufallszahl zwischen 0 und *limit* aus

Aufgabe

- Ergänzen Sie das Programm durch
  - die naive Routine zur Bestimmung des Maximums (~n*n Vergleiche)
  - die verbesserte Routine zur Bestimmung des Maximums (~n Vergleiche)

  wie sie in der Vorlesung vorgestellt wurden
- Evaluieren Sie Ihre Implementierungen durch Testläufe mit 200, 2000, 20000, 200000, 2000000 Zahlen
- Hinweis: Kommentiere Sie vor der Evaluation die Ausgabe der Zahlen aus

Aufgabe

- Ergänzen Sie das Programm, sodass die Zahlen gemäß InsertionSort wie in der Vorlesung behandelt sortiert werden
- Evaluieren Sie wiederum Ihre Implementierung für unterschiedlich große Eingaben

Aufgabe

- Laden Sie das Programm `Reissverschluss.java` von der Webseite herunter, übersetzen Sie es und versuchen Sie zu verstehen, was es macht
- Fügen Sie dann entsprechenden Code hinzu, der die sortierten Folgen meineZahlenL und meineZahlenR in einem neuen Array „mischt“, so wie wir es im Reissverschlussschritt von MergeSort kennengelernt haben

while-Loop (eigentl. überflüssig)

- Syntax:
```
while (Bedingung)
   { ..<body>....}
```
- führt den Code in <body> aus, solange die Schleifenbedingung erfüllt ist
- kann ersetzt werden durch
```
for( ;Bedingung; )
   {..<body>....}
```
- umgekehrt können Sie eine for-Schleife auch durch einen while-Loop ersetzen (mit entsprechenden Variablen)

while-Loop (nicht ganz überflüssig)

- alternativer Syntax:

```
do
{...<body>...}
while (Bedingung);
```

- führt den Schleifenbody einmal aus (auf jeden Fall) und wiederholt, solange die Bedingung erfüllt ist

Aufgaben

- Schreiben Sie je einProgramm,
  - das die Zahlen von 1 bis 20 aufsteigend ausgibt
  - das die Zahlen von 20 bis 1 absteigend ausgibt
  - die Summe der ersten n (Eingabe) Zahlen berechnet
  - n! berechnet

  und dabei die while-Schleife benutzt

Aufgaben

- Schreiben Sie ein Programm, das zunächst 100 Zufallszahlen erzeugt und dann aus diesen die beiden Zahlen a und b bestimmt, deren Differenz |a-b| minimal ist.
- Analysieren Sie die Laufzeit Ihrer Implementierung.

Methoden/Funktionen

- Java (und auch andere Programmiersprachen) erlauben es, Codefragemente in Funktionen (oder in Java „Methoden“) auszulagern
- Beispiel:

```
public static void printMe(int zahl)
{
   System.out.println(zahl);
}
```

in der Main-Funktion könnte diese Methode aufgerufen werden durch

```
public static void main(...)
{ …
  for(int i=0; i<10; i++)
  {
    printMe(i);
  }
}
```

- Argumente können der Methode übergeben werden, müssen jedoch samt Typ in der ersten Zeile angegeben werden (hier: `int zahl`); hierbei ist die Reihenfolge wichtig...

Aufgabe

- Schreiben Sie Methoden, um
  - die Summe von zwei übergebenen Zahlen auszugeben
  - die Differenz von zwei übergebenen Zahlen auszugeben
  - das Produkt von zwei übergebenen Zahlen auszugeben
  - das Maximum von zwei übergebenen Zahlen auszugeben
- testen Sie diese Methoden anhand eines Beispielprogramms (wie auf der Seite zuvor)

Rekursive Aufrufe von Funktionen/Methoden

- Funktionen und Methoden können sich auch selbst aufrufen
- Manche Aufgaben lassen sich dadurch extrem elegant lösen
- Beispiel:

```
public static int numbersSum(int n)
{
   if (n==0) return 0;
   else return n+numbersSum(n-1);
}
```

- Diese Methode könnte man aufrufen mit

```
int sum=numberSum(5);
```

  sum enthält dann die Summe 0+1+2+3+4+5
- Anders als die Methode printMe von vorhin liefert diese Methode auch noch einen Rückgabewert
- der Typ (hier: int) des Rückgabewerts muss vor dem Methodennamen angegeben werden

Aufgabe

- Berechnen Sie n! rekursiv

Aufgabe

- Laden Sie von der Webseite den Quellcode von „MergeSort.java“ herunter, übersetzen ihn und führen ihn z.B. mit
  `java MergeSort 17`
  aus. Versuchen Sie zu verstehen, was darin passiert
- Ergänzen Sie die Methode/Funktion `mergeSort`, sodass sie in der Tat sortiert (wie in der Vorlesung erklärt)
- **Hinweis:** mergeSort(A, l, r) sollte die Zahlen A[l], A[l+1], ...,A[r] sortieren. Möglicherweise müssen Sie sich „Zwischenspeicher“ besorgen, in welchen sie die rekursiv sortierten Folgen A[l],...,A[m] und A[m+1],...A[r] zunächst „mischen“ (siehe vorherige Übung) und daraus dann in A[l],...,A[r] kopieren

Midterm-Übungsaufgabe

- Die Midterm-Übungsaufgabe muss bis zum 12. Dezember (Samstag) per EMail an den jeweiligen Übungsgruppenleiter sowie den Dozenten geschickt werden (Quellcode samt ein paar Zeilen Erläuterungen in einer EMail)
- In der Woche 14.-18. Dezember werden die Übungsgruppenleiter bzw. der Dozent stichprobenweise Fragen zu den eingereichten Lösungen stellen
- Es dürfen Teams von max. 2 Teilnehmern gebildet werden; jedes Teammitglied muss jedoch in der Lage sein, den Code selbstständig zu erklären
- Die Übungsaufgabe gilt als erfolgreich bearbeitet, wenn sich der Programmcode übersetzen+ausführen läßt, der in der Aufgabe gestellte Algorithmus implementiert wurde, und der Teilnehmer die Fragen zur Implementierung korrekt beantworten kann.

Midterm-Übungsaufgabe

- Implementieren Sie Heapsort, nutzen Sie hierzu das Gerüst, das auf der Webseite steht
- die notwendigen algorithmischen Grundlagen werden in der Woche vom 23. behandelt
- Falls Sie die vorhergehenden Aufgaben *selbstständig* lösen konnten, sollte die MTÜ kein Problem darstellen.

Endterm-Übungsaufgabe

- Ziel der „Endterm-Übungsaufgabe“ ist es, Dijkstra's Algorithmus, so wie er in der Vorlesung behandelt wurde, in Java zu implementieren
- Einige Komponenten wie z.B. Teile der Graphrepräsentation werden vorgegeben werden, Sie müssen also – wie bei der MTÜ – nur noch komplettieren :-)
- Formalitäten entsprechen denen der MTÜ; Abgabe/Kontrolle Ende Januar

Erläuterungen (1)

- Die grundlegende Datenstruktur des Routenplaners ist die Repräsentation des Straßennetzwerks
- Diese Repräsentation sollte so gewählt sein, dass sowohl der Zugriff auf die Daten zur Berechnung kürzester Wege so effizient wie möglich erfolgen kann
- desweiteren ist auf Platzeffizienz zu achten, da große Straßennetzwerke wie das der USA ca. 20 Mio Knoten und 60 Mio Kanten enthalten

Erläuterungen (2)

- Grobe Vorgehensweise
  - Entwurf einer Datenstruktur, welche die Straßendaten speichert (Anfang ist vorgegeben)
  - Implementierung eines Algorithmus zur Berechnung kürzester Wege in Graphen (Dijkstra's Algorithmus), zunächst naiv, ohne eine Heap-Datenstruktur
  - Nutzung einer entsprechenden Heap-Datenstruktur

Teilaufgaben Woche 4.1.-8.1.10

- Laden Sie das Skelett „Dijkstra.java“ von der Webseite herunter; diese Implementierung liest bereits einen Graph aus einer Datei ein und füllt insbesondere das Kantenarray. Eine Testdatei finden Sie ebenfalls auf der Webseite.
- 1. Aufgabe:
  Komplettieren Sie die Funktion `computeEdgeOffsets();`
  Ihre Tutoren werden Ihnen das Konzept der EdgeOffsets auch anhand von Beispielen erklären
- 2. Aufgabe:
  Nutzen Sie die berechneten EdgeOffsets, um effizient die ausgehenden Kanten eines Knotens auszugeben; komplettieren Sie hierzu die Methode `printAdjacentEdges()`

Neues Konzept „class“

- Im Skelett wird eine Klasse `class Edge` definiert, welche die Kanten eines Graphen repräsentiert
- Sie können mittels
  ```
  Edge meineKante;
  meineKante=new Edge();
  ```
  eine Instanz dieser Klasse erzeugen, genauso wie Sie eine Integervariable erzeugen mittels
  ```
  int meineZahl;
  ```
- Im Gegensatz zu einer Variablen vom Typ `int` (die nur eine Zahl trägt) , enthält eine Variable/Instanz vom Typ `Edge` jedoch mehr Informationen, nämlich Quell- und Zielknoten sowie Kantengewicht. Sie können auf diese Informationen durch `meineKante.src, meineKante.trg` und `meineKante.weight` zugreifen
- Selbstverständlich können Sie auch Arrays von Elementen vom Typ `Edge` erzeugen, wie es im Beispielcode auch passiert
- Die Definition einer Klasse bietet sich dann an, wenn man explizit Daten „gemeinsam“ verwalten möchte; wir hätten natürlich auch 3 Arrays anlegen können (eins für Quell-, eins für Zielknoten und eines für das Gewicht – alle vom Typ `int`)

Teilaufgabe Woche 11.01.-15.01.

- Erstellen Sie die Methode
  `public static void computeSP(int s)`
  welche:
  - ein Distanzarray dist anlegt (Typ int, Größe=#Knoten, zunächst auf eine sehr große Zahl gesetzt)
  - eine Knotenmenge U in einer Schlange verwaltet (d.h. Elemente hinten anfügen und vorne entnehmen; nutzen Sie dazu auch ein int-Array, ggf. mit zwei weiteren Variablen, die den aktuellen Stand von „Kopf“ und „Schwanz“ der Schlange repräsentieren)
  - zu Beginn ist nur Knoten s in U, dist[s] wird auf 0 gesetzt
  - solange U nicht leer ist:
    - entnehme das erste Element v von U, überprüfe alle ausgehenden Kanten; falls für Kante (v,w) gilt: dist[w]>dist[v]+c(v,w), ändere dist[w] und füge w der Schlange hinten hinzu (falls noch nicht enthalten – dazu könnte es vorteilhaft sein, ein weiteres Array zu erstellen, welches sich merkt ob der entsprechende Knoten schon in U ist ...)

Teilaufgabe Woche 11.01.-15.01.

- rufen Sie computeSP(...) aus der main Methode auf
- geben Sie danach die berechneten Distanzen aller Knoten aus
- Testen Sie Ihre erste naive Implementierung einer Kürzesten-Wege-Berechnung mit dem Testgraphen

Teilaufgabe Woche 18.01.-22.01.

- ersetzen Sie die Schlange durch einen Heap; hierbei ist folgendes zu berücksichtigen:
  - beim Heap, wie er in Heapsort genutzt wurde, waren die zu organisierenden Elemente (Zahlen) auch gleichzeitig die, welche die Ordnung bestimmten; jetzt wollen wir Knoten verwalten, diese jedoch gemäß Ihrer Distanz organisieren
  - Mögliche Lösung: zwei Arrays, eines für die Distanzen, eines für die KnotenIDs; das Distanzarray wird behandelt wie beim bereits implementierten Heap; wenn ein Tausch stattfindet, wird dieser jedoch auch im KnotenID-Array durchgeführt
- Erweitern Sie Ihre Implementierung, sodass nicht nur die Distanz, sondern auch der Pfad zum gewünschten Zielknoten ausgegeben wird

Teilaufgabe Woche 25.01.-29.01.

- Führen Sie den aktuellen Stand Ihrer Implementierung den Tutoren vor
- Falls Ihre Implementierung zu diesem Zeitpunkt noch nicht funktionsfähig ist, können Sie immer noch Anfang Februar in einem weiteren Termin Ihr Programm vorführen; **bei den BMI Studenten muß die Präsentation des aktuellen Stands in der Woche 25.-29.01. aber auf jeden Fall stattfinden!**