Graphentheoretische Konzepte und Algorithmen / Skizze 02

Team: <2>, <Dmitrij Schaefer, Flah Ahmad>

Aufgabenaufteilung:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>,
   <Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>,
   <Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden>

Quellenangaben:
<
www.erlang.org,
stackoverflow.com/questions/1459152/erlang-listsindex-of-function,
http://de.wikipedia.org/wiki/Bellman-Ford-Algorithmus
http://fuzzy.cs.uni-magdeburg.de/studium/graph/txt/duvigneau.pdf
GRBuch: Seite 53 - 56
>

Begründung für Codeübernahme:
<
Eine Hilfsmethode um Index für ein bestimmtes Element in einer Liste zu finden,
hätten es selbst implementieren können, jedoch aus Zeitgründen die Methode von
stackoverflow.com/questions/1459152/erlang-listsindex-of-function einfach kopiert, Methode spielt für die beiden Algorithmen selbst keine Rolle!

Investierte Zeit für diesen Vorgang, ca. 2 Minuten, wobei das selbst implementieren vermutlich das 5 oder 10 fache an Zeit gekostet hätte.
>

Bearbeitungszeitraum: <

Bellman_Ford:
09.04.14 → 3std
10.04.14 → 3std
13.04.14 → 3std
14.04.14 → 3std
15.04.14 → 3std
12 Stunden benötigt

Floyd-Warshall:
16.04.2014 → 3std.
19.04.2014 → 3std.
20.04.2014 → 3std.
21.04.2014 → 2std.
11 Stunden benötigt

Gesamtzeit = 23 Stunden
>
Aktueller Stand: <fertig>

Graphentheoretische Konzepte und Algorithmen / Skizze 02

***Skizze**:*

<

Unser erster Schritt den wir machen wollen ist, die Algorithmen ganz zu verstehen.
Nach dem die Algorithmen verinnerlicht wurden, haben wir uns Gedanken gemacht wie wir den Algorithmen implementieren wollen, Es bietet sich an den Algorithmen in einzelne Teile auf zu teilen, ähnlich wie der Algorithmen selbst beschrieben wird.

Somit haben wir eine geringe Kopplung und man könnte die Methoden weiter verwenden, unabhängig von Algorithmus wenn man es möchte.

Z.B. Eine Methode die wir Initialisierungsphase nennen,
Wo der erste Teil von Bellman und Ford verarbeitet wird:

für jedes v aus V
Distanz(v) := unendlich, Vorgänger(v) := kein
Distanz(s) := 0

Dann könnte man das Herz des Algorithmus in eine extra Methode verpacken, die man z.B. Heart nennt:

wiederhole n - 1 mal
für jedes (u,v) aus E
wenn Distanz(u) + Gewicht(u,v) < Distanz(v)
dann
Distanz(v) := Distanz(u) + Gewicht(u,v)
Vorgänger(v) := u

Zur aller letzt könnte man noch eine Methode bauen, die den negativen Zyklus finden, die könnte man z.B. checkNegativeZyklus nennen.

für jedes (u,v) aus E
wenn Distanz(u) + Gewicht(u,v) < Distanz(v) dann
STOP mit Ausgabe "Zyklus negativer Länge gefunden"
Ausgabe Distanz

Und genau so möchten wir es auch für Floyd & Warshall implementieren, sie in Teile zu unterteilen und mit einer Super Methode wie z.B. StartAlgo die ganzen Teile vereinen.

Somit schaffen wir auch eine bessere Übersichtlichkeit und können besser und schneller Fehler entdecken, falls welche Auftreten.

Nach der Implementierung:

Wir haben das beschriebe Konzept von Seite 2 konsequent eingehalten und sind auf keine Probleme in der Implementierung gestoßen.
Ungewöhnlich war es, Elemente in eine Matrix Funktional hinzuzufügen, denn so einfach wie in OO funktioniert es nicht, denn eine Zuweisung > 2 auf genau eine Variable ist nicht möglich.
Somit ermöglicht das auch Erlang die Variablen im Prozessor zu halten und somit schneller zu sein, weil nicht auf den Arbeitsspeicher zugegriffen werden muss, da keine Referenzen vorhanden sind und jede Variable eindeutig ist.
Auch was vermutlich man in Java (OO) alles iterativ machen würde, da der Algorithmus selbst auch iterativ beschrieben wird, mussten wir in Erlang Rekursiv arbeiten, was jedoch zu keinen Problemen geführt hat. Den Jede iterative Schleife kann man in Rekursion überführen.

>