Arbeitsblatt 2 - GKA – Klauck - Abgabe
Team: 08, Matthias Nitsche, Swaneet Sahoo

Aufgabenaufteilung:
Swaneet:
OptionalTest, BellmanFord, FloydWarshall

Matthias:
OptionalTest, BellmanFord, FloydWarshall, Constants

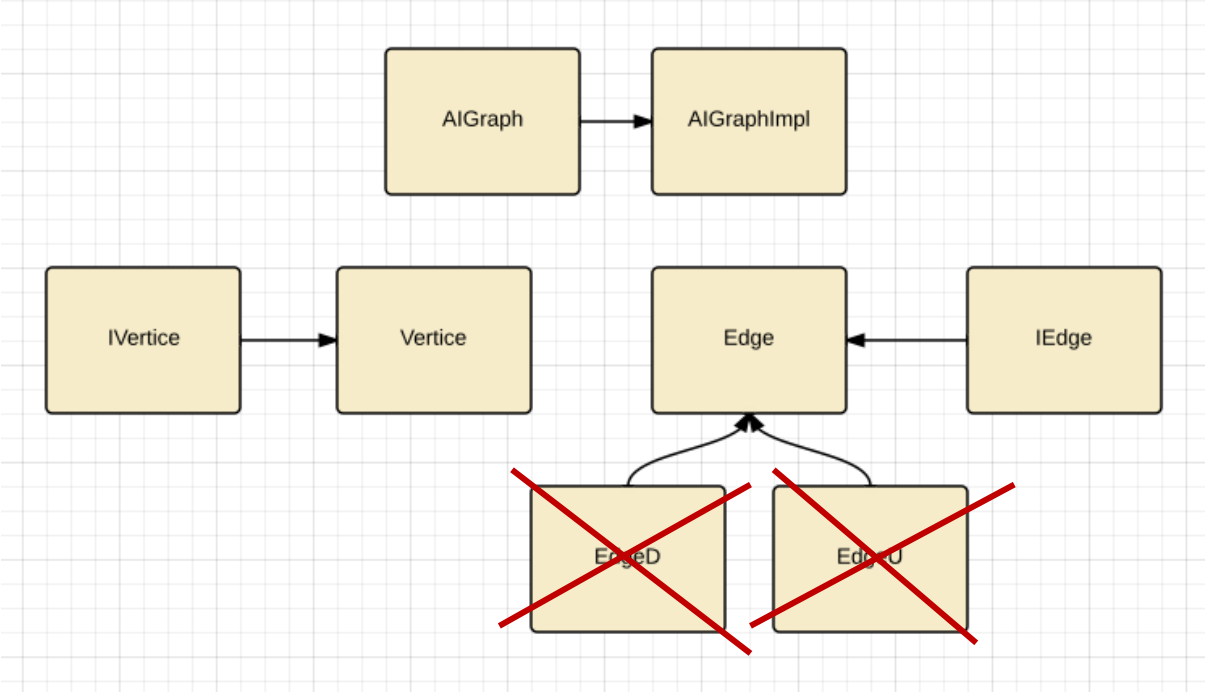
Sehr ausgeglichene Zeiteinteilung. Immer Pair Programming.

Bearbeitungszeitraum:
Swaneet: 1 Stunden
Matthias: 2Stunden
Zusammen: 6 Stunden
Insgesamt: 9 Stunden

Aktueller Stand:
Bellman Ford und Floyd Warshall implementiert
Test Abdeckung 95%
GraphParser abgeschlossen
Refactor von den ADTs -> EdgeD und EdgeU entfernt

Skizze:

Es gab ein paar kleine Änderungen in der Graph ADT. Da wir im Folgenden nur mit gerichteten oder ungerichteten Graphen arbeiten werden, ist der Umweg über die Kanten kein intuitiver Mechanismus. Es ist aus einer Objekt Orientierten Sicht nicht sehr sinnvoll eine einzige Kante zu fragen ob diese gerichtet ist. Der Graph sollte diese Information tragen. Es muss im Graph Konstruktor ein Boolean übergeben werden, der bei True annimmt, dass der Graph gerichtet ist und False das der Graph ungerichtet ist. Im Folgenden daher die überarbeitete Skizze.



Bellman Ford und Floyd Warshall Algorithmen für den kürzesten Pfad innerhalb eines Graphen mit nicht negativ gewichteten Kanten waren relativ einfach zu implementieren. Der vorgegebene Pseudocode war nicht sehr schwierig zu verstehen. Interessant war, dass uns die Datentypen zur Erzeugung der benötigten Informationen sehr viel mehr Schwierigkeiten bereit hat.

Für den Floyd Warshall haben wir uns entschieden unsere eigene Matrix Implementation zu benutzen um näher an den beschriebenen Algorithmus aus unserem Lehrbuch zu kommen und zu trainieren Graphen als Matrix darzustellen. Das initialisieren der Matrix von unserer Graphenstruktur hat also am Ende die meiste Zeit gekostet.
Der Pseudocode ist im Lehrbuch aus der GKA Vorlesung.

Beim Bellman Ford haben wir uns entschieden eine Version zu benutzen die auf Graphenstrukturen arbeitet. Das heißt Mengen von Kanten und Ecken. Es gab keine sonderlich große Intialisierungsphase, es war aber nicht ganz trivial genau zu verstehen wo hier der Vergleichsschritt zwischen i, j und k zu i, k stattgefunden hat. Für das Null Element in Bellman Ford haben wir eine Konstante „NULL_LONG“ mit dem Wert -1L genommen.

```
procedure BellmanFord(list vertices, list
edges, vertex source)
   // This implementation takes in a graph,
represented as lists of vertices and edges,
   // and fills two arrays (distance and
predecessor) with shortest-path information

   // Step 1: initialize graph
   for each vertex v in vertices:
       if v is source then distance[v] := 0
       else distance[v] := infinity
       predecessor[v] := null

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge (u, v) with weight w in
edges:
           if distance[u] + w < distance[v]:
               distance[v] := distance[u] + w
               predecessor[v] := u

   // Step 3: check for negative-weight cycles
   for each edge (u, v) with weight w in
edges:
       if distance[u] + w < distance[v]:
           error "Graph contains a negative-
weight cycle"
```

Quelle: http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

Unser Algorithmus sieht ähnlich aus, hat aber den unterschied das man einfach nur einen Graphen übergibt und intern alle Methoden aufgerufen warden.

Wir haben uns entschieden eine Graphen Instanz an unsere Algorithmen zu übergeben und in der jeweiligen Algorithmen Klasse die benötigte Datenstruktur zu initialisieren.
Das heißt wir erschaffen uns eine Instanz von dem Algorithmus und übergeben den Graphen und einen String welches das Kanten Attribut

darstellt. Beim Bellman Ford muss man außerdem einen Startknoten bei der Instanziierung übergeben. Die Algorithmen selbst sind Mutable, da sämtliche Argumente überschrieben werden können und neu gesetzt. Dies wurde allerdings für die Nutzbarkeit beim Testen eingefügt. Das start() keyword führt den ganzen Algorithmus aus, dass getPath() kann anschließend den kürzesten Weg von src nach dest, rekonstruieren.

Ein weiteres großes Problem war letztendlich, wie wir aus dem Algorithmus den aktuellen kürzesten Pfad denn nun herausbekommen. Beim Floyd Warshall haben wir zwei neue Matrizen gebaut die diese Information enthalten, allerdings schwer lesbar. Beim Bellman Ford haben wir zwei neue HashMaps gebaut. Wie bekommt man nun aber diesen kürzesten Pfad zu einem gegebenen Knoten? Hierfür haben wir zwei Methoden gebaut die rekursiv/endrekursiv die Matrizen/HashMaps ablaufen und den Weg, falls es einen gibt als Liste oder String zurückgeben.
Leere Liste heißt „Kein Pfad“.

Die Fehlerfälle handhaben wir so, dass ein Error geworfen wird wenn der entsprechende Knoten (Quelle oder Ziel ist hierbei egal) nicht im Graph ist oder wenn ein Cycle gefunden wurde. Die Testfälle haben wir aus dem GKA Lehrbuch von S. 54. Es spielt bei uns keine Rolle ob der Graph gerichtet oder ungerichtet is. was uns Flexibilität erschafft.

Quellen:
- Theoretische Annahmen und Grundlagen aus den Klauck Vorlesungsfolien
- Bellman Ford Pseudocode von http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
- AD Praktikum ADT für eine Matrix Implementation die wir selber gebaut haben

Begründung für Codeübernahme:
Die Matrix ADT von uns haben wir benutzt, um eine Distanz und Translationsmatrix für den Floyd Warshall bauen zu können.