AD-Übung zum 3. Dezember

Arne Beer, MN 6489196
Merve Yilmaz, MN 6414978
Sascha Schulz, MN 6434677

17. Dezember 2013

1.
```
Relax(u,v):
    if v.dist > u.dist + w(u, v)
        v.dist = u.dist + w (u, v)
        v.pre = u


BellmanFord_modified(G,s):
    InitializeSingleSource(G,s)
    for i = 1, ..., |V| - 1
        stabilized = true
        for all edges(u, v) in E
            distBeforeRelax = v.dist
            Relax(u,v)
            if distBeforeRelax > v.dist
                stabilized = false
        if stabilized = false
            return true
    return true
```

Anmerkung: Der Test auf negative Zyklen ist auf Grund der Nebenbedingung der Aufgabe entfernt.

Terminierung:

Der Bellmann-Ford-Algorithmus nutzt zur Überprüfung auf negative Zyklen aus, dass sofern es keine negativen Zyklen gibt ein kürzester Pfad maximal alle Knoten durchläuft, also eine Länge von |V|-1 hat. Da die äußere for Schleife endlich ist und der Alorithmus danach returned, ist eine Termination stets gegeben.

Korrektheit:

Das bisherige Verfahren wurde lediglich um ein Abbruchkriteritum erweitert, welches bis zum Zeitpunkt des Abbruchs keinen Datenverändernden Einfluss hat, weshalb bis auf die Korrektheit des Abbruchs der Rest als korrekt angesehen wird.

Lauf Aufgabe ist m die maximale Länge eines Pfades aus der Menge aller kürzesten Pfade zur gegebenen Source in G und es soll nach m+1 Durchläufen abgebrochen werden, da zu diesem Zeitpunkt alle kürzesten Pfade bekant sind. Dies ist genau dann der Fall, wenn innerhalb einer Relax-Operation keine neue Distanz zum Source-Knoten eingetragen wird. Wenn der längste kürzeste Pfad die länge m besitzt, findet folglich im m+1 Durchlauf keine Veränderung der dist-Werte mehr statt. Die Modifikation prüft eben dies, in dem überprüft wird, ob der Wert mit der Relax-Operation verringert wurde und bricht sonst eben in genau dem Durchlauf m+1 ab, da keine Veränderung mehr vorliegt.

2.
```
SSSP(G,s)
    do topologische Sortierung von G
    InitializeSingleSource(G,s)
    forall Knoten u in topologischer Sortierung
```

```
        forall Knoten v \in Adj[u]
            Relax(u,v)
```

3. Der Dijkstra's Algorithmus funktioniert normalerweise mit negativen Kantengewichten nicht, da die Knoten, die bereits erreicht wurden, nicht erneut besucht werden, auch nicht ueber eine negative Kante.

Da negative Kanten alle vom Startknoten wegfuehren, wird es nicht vorkommen, dass ein Knoten bearbeitet wird, der eine negative Kante zu einem bereits abgeschlossenen Knoten besitzt. Dies erklaert sich dadurch, dass alle negativen Kanten des Startknotens direkt am Anfang bearbeitet werden, und dann keine negativen Kanten mehr vorhanden sind, die den Fehler provozieren würden.

4. (a) Wir wenden auf den Baum einen modifizierten BFS an, welcher beim Ausführen der Breitensuche den Abstand des Elternknotens plus das aktuelle Kantengewicht im entsprechenden Kindknoten speichert (sowie losgelöst davon den Knoten mit dem maximalen Wert), erhalten wir das Ergebnis in $O(|V| + |E|)$, da dies die Laufzzeit für BFS ist.

Da wir uns in einem Baum befinden, ist jeder Knoten mit genau einem Elternknoten verbunden, mit außnahme des Wurzelelements. Folglich ist die Anzahl der Kanten um 1 kleiner als die Anzahl der Knoten, somit ist für die Landau-Notation die Anzahl der Kanten nicht weiter von Bedeutung und das Verfahren liegt in $O(|V|)$.

(b)
```
Durchmesser(G):
    durchmesser = 0
    for all v in V
        m = BellmanFord_maxlengthSSSP(G, v)
        if durchmesser < m
            durchmesser = m
    return durchmesser


BellmanFord_maxlengthSSSP(G,s):
    InitializeSingleSource(G, s)
    for i = 1, ..., |V| - 1
        stabilized = true
        for all edges(u, v) in E
            distBeforeRelax = v.dist
            Relax(u,v)
            if distBeforeRelax > v.dist
                stabilized = false
        if stabilized = false
            return i-1
    return i
```

Terminierung: Knoten- und Kanzenmenge sind endlich und werden innerhalb des Algorithmus nicht verändert, folglich terminieren sämtliche Schleifen und somit der Algorithmus.

Korrektheit: Der Durchmesser ist die maximale Länge eines Pfades in der Menge aller möglichen kürzesten Pfade. In Aufgabe 1 wurde gezeigt, dass die Abwandlung des Abbruchkriteritums nach m+1 Schritten korrekt ist. Dies wurde nun weiter modifiziert, in dem m zurück gegeben wird. Was im Fall des Abbruchs eben der vorherige Durchlauf war, da in m+1 abgebrochen wird.

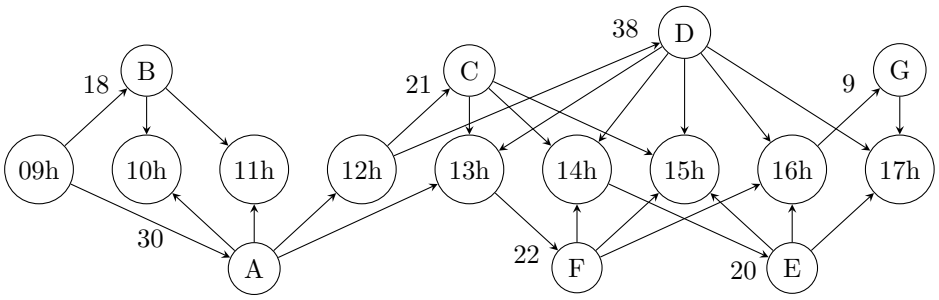
Laufzeit: $O(|V|^2 + |V||E|)$

5. a) Konstruiere aus der Währungs-Adjazenzmatrix einen vollständigen Graphen, indem man die Währungen Knoten repräsentieren lässt und die Werte der Matrix als Kantengewichte einträgt. Aus diesem kann man überprüfen, ob für einen Zyklus die Multiplizierten Kantengewichte nicht 1 ergeben. In diesem Fall würde das bedeuten, dass es Währungsarbitrage gibt. Falls dies für keinen Zyklus zutrifft, enthält die Währungs-Adjazenzmatrix keine Währungsarbitragen.

b) Nein, es sei denn jemand würde negative Werte in die Adjazenzmatrix schreiben, was bedeuten würde, dass jemand sein umzutauschendes Geld komplett verliert und stattdessen sogar Strafe zahlen muss.

6. Scheduling mittels kürzester Pfade

Für den folgenden Graphen ist anzunehmen, dass jedes Kantengewicht gleich 0 ist, sofern nicht anderweitig angegeben.



Da keine negativen Kantengewichte enthalten sind, kann der Algorithmus von Dijkstra angewendet werden, um den kürzesten Pfad von 09h zu 17h zu finden. Da nur eine eizige Point-To-Point Verbindung gefunden werden soll, terminieren wir, sobal der Vorgänger für den Zielknoten feststeht, um unnötige Arbeit zu sparen.

Es sei im folgenden die Entwicklung der hinzugefügten Knoten und die dabei entstehende Vorgänger-Tabelle angegeben.

$S_{01} = \{09h\}$
$S_{02} = \{09h, B\}$
$S_{03} = \{09h, B, 10h\}$
$S_{04} = \{09h, B, 10h, 11h\}$
$S_{05} = \{09h, B, 10h, 11h, A\}$
$S_{06} = \{09h, B, 10h, 11h, A, 12h\}$
$S_{07} = \{09h, B, 10h, 11h, A, 12h, 13h\}$
$S_{08} = \{09h, B, 10h, 11h, A, 12h, 13h, C\}$
$S_{09} = \{09h, B, 10h, 11h, A, 12h, 13h, C, 14h\}$
$S_{10} = \{09h, B, 10h, 11h, A, 12h, 13h, C, 14h, 15h\}$
$S_{11} = \{09h, B, 10h, 11h, A, 12h, 13h, C, 14h, 15h, E\}$
$S_{12} = \{09h, B, 10h, 11h, A, 12h, 13h, C, 14h, 15h, E, 16h\}$
$S_{13} = \{09h, B, 10h, 11h, A, 12h, 13h, C, 14h, 15h, E, 16h, 17h\}$

	A	B	C	D	E	F	G	09h	10h	11h	12h	13h	14h	15h	16h	17h
09h	09h	09h	12h	-	14h	-	-	-	B	B	A	A	C	C	E	E

Der kürzeste Pfad ist somit $17h \cdot E \cdot 14h \cdot C \cdot 12h \cdot B \cdot 09h$ und kostet $18 + 21 + 18 = 57$.