18. Dezember, AD: de Bruyn Kops (6591853), Voroshylov (6590822), Eggert (5690653)

Aufgabe 1

Damit der Bellman-Ford-Algorithmus nach $m + 1$ Durchläufen der äußeren Schleife abbricht, müsste geprüft werden, ob es noch zu Veränderungen von $v.dist$ (Distanz von s nach v) kommt. Sobald es keine Veränderungen mehr gibt, ist $v.dist$ für alle v minimal, und der Algorithmus hat $m + 1$ Iterationen durchlaufen. Um festzuhalten, dass $v.dist$ verändert wurde, müsste man zuerst an die Funktion Relax folgende Zeile anhängen:

active = true

Danach würde man den Bellman-Ford-Algorithmus wie folgt erweitern:

```
InitializeSingleSource(G, s)
active = true
while active == true do                    ▷ Prüfung ob v.dist noch verändert wurde
    active = false
    for all edges (u, v) ∈ E do
        RELAX(u,v)
    end for
end while
```

Durch die Änderung bricht der Algorithmus nun ab, sobald m+1 Iterationen durchlaufen sind, da die Variable active dann nicht mehr auf true gestzt wird.

Aufgabe 4

a)
Der Durchmesser eines Baumes mit nicht-negativen Kantengewichten könnte in Zeit O(|V|) bestimmt werden wenn jeder Knoten des Baumes maximal ein Kind hat, zB:

```
O
|
O
|
O
|
O
```

Dann um den Durchmesser herauszufinden würde ein Algorithmus nur von der Wurzel nach dem Blatt einmal durch das Baum laufen. Damit würde die Laufzeit O(|V|) werden.

b)
Algorithmus: alle kürzeste Pfade finden (zB mit Dijkstra Algorithm) und davon der längste ausgeben. Laufzeit: $O(|V|^2|E|)$

Pseudocode:

```
maxpath = 0                              /* maximal length of shortest path */
for vertex = 1, ..., |V|                 /* Find shortest path starting at each vertex */
S = {s}                                  /* S set of explored vertices */
    d(s) = 0
    while S ≠ V
        U := {u ∉ S | u neighbor of a vertex ∈ S }          /*candidates */
        for all u ∈ U
            for all pre(u) ∈ S that are predecessors of u
                d′(u, pre(u)) := d(pre(u)) + w (pre(u), u)         /* candidate distances */
                u′ := argmin {d′ (u,pre(u)), u ∈ U }            /* choose best candidate */
                d(u′) = d′(u′)
                S = S ⋃ u′
            if d(u′) > maxpath
                maxpath = d(u′)
return maxpath                           /* maxpath = diameter of the graph */
```

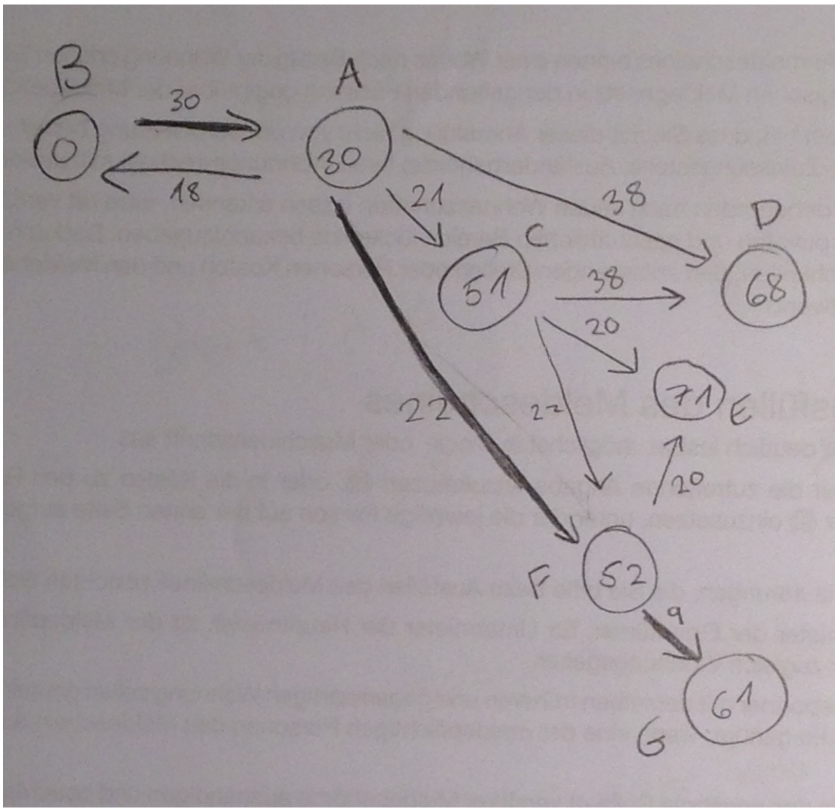
Aufgabe 5

(a) Um die Möglichkeit einer Währungsarbitrage aus der gegebenen Matrix zu ermitteln, wäre es sinnvoll einen gerichteten und gewichteten Graphen G zu erstellen. Hierbei würden die Währungen die Knoten v darstellen, die gerichteten Kanten E hätten als Gewicht w die Logarithmen der Umtauschkurse r_{ij}. Eine Währungsarbitrage besteht dann wenn ein Zyklus existiert, welcher vom Startpunkt s zurück zu s führt, und die Summe von w dabei positiv ausfällt. Die letztere Bedingung entsteht, da die Logarithmen der Wechselkurse nur dann positiv sind, falls $r_{ij} > 1$ ist.

(b) Ja, es können negative Zyklen auftreten. Negative Zyklen stehen für ein Verlustgeschäft, da die Summe von w entlang der Pfade des Zyklus negativ ist, und somit auch die Wechselkurse zu einer Abnahme gegenüber der Startposition führen.

Aufgabe 6

Da jeder Taxifahrer für den jeweiligen Zeitraum entweder ganz oder gar nicht gebuchen werden kann, können wir den folgenden Graphen herstellen:



Durch Beobachtung der Zeiträume sehen wir, dass wir ein kürzeste Pfad finden müssen, der mit entweder A oder B anfängt (und dass B für minimalen Kosten nicht in einem kürzestem Pfad eingeschlossen werden könnte) und mit entweder D, E, oder G endet. Das Bellman-Ford Algorithm ergibt die obrigen Nummerierung von Kanten. Dadurch sehen wir, dass der Pfad, der mit G endet, der kürzeste Pfad ist. Also der kürzeste Pfad ist AFG. Das heißt, dass für die jeweilige Zeiträume, A, F, und G ganz im Einsatz sind und alle anderen Taxifahrer gar nicht im Einsatz sind.