AD [HA] zum 18. 12. 2013

Arne Struck, Lars Thoms

17. Dezember 2013

1.:

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

Die Anpassung durch nochanges bewirkt eine Terminierung einen Durchlauf nachdem alle kürzesten Kantenpfade gefunden sind. Dies geschieht, da nach m Durchläufen alle kürzesten Kantenpfade entdeckt sind. Die Schleife wird noch einmal durchlaufen und hier wird festgestellt, dass insgesamt keine Änderungen an einem der Pfadgewichte vorgenommen wurde. Also wird der Algorithmus darauf Terminieren. Weitere Terminierungen sollten nicht sinnvoll sein, da sie durch den spezifizierten Input (kein negativen Zyklen) obsolet geworden sind. Der endgültige return kann auch weggelassen werden, da ein anderer return auf jeden Fall erreicht wird.

2.:

```
DAG-Shortest-Path(G,s):
    sort G.V topologically
    InitializeSingleSource(G,s)
    for each u in G.V % now in topological order
        for each v in Adj(u)
            Relax(u,v)
```

3.:

Das Problem mit dem Dijkstra-Algorithmus mit negativem Kantengewicht ist, dass sie eventuell nicht berücksichtigt werden, da der Algorithmus nicht "in die Zukunft sehen" kann.
Da wenn einer der Nachfolgeknoten von S auf dem kürzesten Pfad liegt, muss nur noch gezeigt werden, dass Dijkstra den Knoten korrekt findet.
Angefangen wird immer mit der kleinsten Kantengewichtung zu einem der Nachfolgeknoten

von S (im folgenden A). Sollte kein Pfad von A zu einem der anderen Nachfolger von B existieren, wird auch hier die negative Kante genommen. Sollte nun ein Pfad von A zu einem der anderen Nachfolger existieren, wird Dijkstra korrekt vergleichen, welcher der beiden Pfade kürzer ist.
Damit ist gezeigt, dass der Algorithmus funktioniert.

4.: a)

Genutzt wird die DFS (worst-case-Laufzeit $\mathcal{O}(|V|)$). Mit dieser wird der am weitest entfernte Knoten vom Root gefunden und der Pfad zwischengespeichert. Dies wird wiederholt mit der Bedingung, dass die beiden Endknoten nicht gleich sein dürfen.
Damit ist der Längste Pfad zwischen 2 Knoten in einem Baum durch aneinanderhängen des einen Zwischenergebnis und dem Reversen des anderen bekannt. Da die zusätzlichen Operationen als Konstanten sind, bleibt die Laufzeit $\mathcal{O}(|V|)$.

b)

TODO

5.: a)

Die $n \times n$-Matrix enthält die Wechselkurse der Währungen zueinander. Dies kann man mehr oder weniger direkt in eine Adjazenzmatrix umwandeln und es als vollständigen, gerichteten Graphen darstellen.

Dabei entstehen natürlich auch negativ-gewichtete Kanten, wenn eine Währung einen niedrigeren Wert gegenüber einer anderen hat.

Der Algorithmus zur Erkennung von Währungsarbitrage ist im Grunde ein BellmanFord-Algorithmus. Er beginnt mit irgendeinem Knoten (Währung) und terminiert mit einem false, wenn er negative Zyklen findet. Dabei muss die Reflexivität der Währungen ignoriert werden.

b)

Ja, da der Algorithmus mit einem false terminiert, wenn er einen negativen Zyklus findet. Dadurch kann man ableiten, dass es einen Pfad gibt, der unendlich viel Geld beschert.

Angenommen man würde Euros in Dollars, dann in Yen und wieder in Euros umtauschen und die Wechselkurse Euro-Yen > Euro-Dollar-Yen wären, dann entstünde ein negativer Zyklus.

6.:

TODO