Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger
Frühjahrssemester 2018

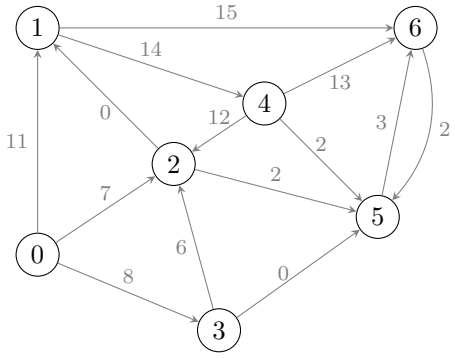
Universität Basel
Fachbereich Informatik

Übungsblatt 10

Abgabe: 18. Mai 2018

Aufgabe 10.1 (Dijkstras Algorithmus, 2 Punkte)

Betrachten Sie folgenden gewichteten Digraphen:



Simulieren Sie den Algorithmus von Dijkstra auf dem Graphen, um die kürzesten Pfade von 0 zu jedem anderen Knoten zu bestimmen. Geben Sie hierbei für jeden Aufruf von `relax` an, welcher Knoten relaxiert wird, welche Distanzen jeweils wie geupdated werden und welche Knoten am Ende des Aufrufes in der Priority-Queue sind.
Welchen Minimale-Pfade-Baum haben Sie berechnet?

Aufgabe 10.2 (Bellman-Ford-Algorithmus, 2.5+2.5 Punkte)

Implementieren Sie den Bellman-Ford-Algorithmus zum Finden von kürzesten Pfaden in Graphen mit beliebigen Kantengewichten.

(a) Vervollständigen Sie hierzu zunächst die Klasse `CycleDetector`, die Zykel in gewichteten Digraphen finden soll. Hierzu können Sie den Algorithmus `DirectedCycle` von Foliensatz C2 (Folie 42-44 in Druckversion) für die Repräsentation von `EdgeWeightedDigraph` anpassen. In Python kann das zum Beispiel folgendermassen aussehen:

```python
class CycleDetector:
    def __init__(self, graph):
        self.edge_to = [None] * graph.no_nodes()
        self.marked = [False] * graph.no_nodes()
        self.on_current_path = [False] * graph.no_nodes()
        self.cycle = None
        for node in range(graph.no_nodes()):
            if not self.marked[node]:
                self.dfs(graph, node)

    def dfs(self, graph, node):
        self.on_current_path[node] = True
        self.marked[node] = True
        for edge in graph.adjacent_edges(node):
            if self.has_cycle():
                return
            to = edge.to_node()
            if not self.marked[to]:
                self.edge_to[to] = edge
```

```
20                 self.dfs(graph, to)
21             elif self.on_current_path[to]:
22                 self.cycle = deque()
23                 prev_edge = edge
24                 while prev_edge.from_node() != to:
25                     self.cycle.appendleft(prev_edge)
26                     prev_edge = self.edge_to[prev_edge.from_node()]
27                 self.cycle.appendleft(prev_edge)
28                 return
29         self.on_current_path[node] = False
30
31     def get_cycle(self):
32         return self.cycle
```

(b) Vervollständigen Sie dann die Klasse `BellmanFord`.

Die Main-Methode (in `Main.java`) liest einen gewichteten Graphen vom Standardinput ein und führt den Bellman-Ford-Algorithmus für Knoten 0 aus. Sie können Ihre Implementierung mit den Dateien `tinyEWDn.txt` und `tinyEWDnc.txt` testen, z.B. mit `gradlew run < tinyEWDn.txt`. Beide Dateien gehören zu dem Lehrbuch von Sedgewick & Wayne; die erste enthält keinen negativen Zyklus, die zweite schon.

Aufgabe 10.3 (Anwendung: Arbitragemöglichkeiten, 2+2 Bonuspunkte)

Bei dieser Aufgabe handelt es sich um eine Bonusaufgabe, d.h. Sie können Punkte erlangen, aber die Aufgabe erhöht nicht die Anzahl der Punkte, die zur Klausurzulassung notwendig sind.
Diese Aufgabe baut auf Ihre Lösung von Aufgabe 10.2 auf, sie müssen diese also zuerst bearbeiten. Können Sie Aufgabe 10.2 nicht lösen, können Sie bei dieser Aufgabe immer noch 2 Punkte für die Beschreibung und Erklärung des Lösungswegs (Teilaufgabe a) erhalten.
In der Datei `rates.txt` finden Sie Wechselkurse für fünf verschiedene Währungen. Für einen USD erhalten Sie zum Beispiel 0.741 EUR oder 0.657 GBP.
Kann man sein Vermögen durch geschickten Umtausch vermehren, spricht man von Arbitragemöglichkeiten. Im Rahmen dieser Aufgabe sollen Sie die Wechselkurse auf solche Arbitragemöglichkeiten testen.
Man kann die einzelnen Währungen als Knoten in einem Graphen auffassen. Ein Pfad $X \to Y \to Z$ soll dann einen Umtausch von Währung X in Währung Z mit einem Zwischenschritt über Y repräsentieren. Um entsprechende Umrechungswerte zu erhalten, müsste man dabei die Umrechnungskurse multiplizieren. Findet man einen Zyklus mit (multipliziertem) Wert > 1, kann man Gewinn machen, indem man die Währungen einmal „im Kreis“ tauscht.
Bei der Suche nach kürzesten Pfaden und Zykeln addieren wir jedoch die Gewichte. Nutzen Sie daher bei der Lösung dieser Aufgabe den Logarithmus (Produktregel).

(a) Wie können Sie die Wechselkurse so in einem Graphen repräsentieren, dass Arbitragemöglichkeiten Zykeln mit negativem Gewicht entsprechen? Begründen Sie Ihre Antwort.

(b) Nutzen Sie Ihre Implementierung von Aufgabe 9.2, um festzustellen, ob es einen solchen Zyklus gibt. In diesem Fall geben Sie bitte den entsprechenden Zyklus aus. Wieviel Gewinn machen Sie, wenn sie mit 100 000 Einheiten der Ausgangswährung einmal im Kreis tauschen? Vervollständigen Sie `computeArbitrage()` in `Arbitrage.java` mit der notwendigen Berechnung und Ausgabe. Sie können den Code mit `gradlew runArbitrage < rates.txt` ausführen.

Die Übungsblätter dürfen in Gruppen von zwei Studierenden bearbeitet werden. Bitte schreiben Sie beide Namen auf Ihre Lösung.