Übung zur Vorlesung Algorithmen und Datenstrukturen

Aufgabe T22

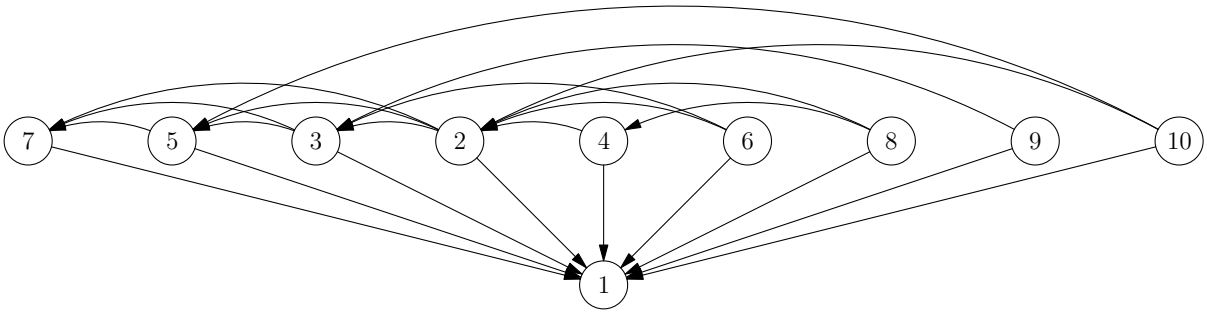
Folgende Relation sei auf den natürlichen Zahlen definiert:
$n \prec' m$ gilt genau dann, wenn n ein Teiler von m ist oder wenn $n > m$ und sowohl n als auch m Primzahlen sind.
Die Halbordnung $\prec$ ist die transitive Hülle von $\prec'$.

a) Skizzieren Sie den gerichteten Graphen, der die Relation $\prec'$ (und damit indirekt auch $\prec$) auf der Teilmenge $\{1, 2, 3, \ldots, 10\}$ darstellt.

b) Bestimmmen Sie eine topologische Sortierung dieser Menge, indem Sie den Algorithmus aus der Vorlesung verwenden.

Lösungsvorschlag:

a)



b) Eine gültige Sortierung ist z.B. $1, 7, 5, 3, 2, 4, 6, 8, 9, 10$.

Aufgabe T23

Wo steckt der Gedankenfehler in folgendem „Beweis“?

> Wenn ich n Dinge sortieren will, dann sortiere ich sie einfach topologisch, anstatt normal. Danach sind sie sortiert und ich habe es in linearer Zeit geschafft. Quicksort braucht aber ja schon mehr als lineare Zeit und daher habe ich Quicksort geschlagen.

Lösungsvorschlag:

Quicksort benötigt $O(n \log n)$ Zeit, wobei n die Anzahl der Schlüssel ist. Topologisches Sortieren benötigt nur $O(n + m)$ Zeit, wobei n die Anzahl der Knoten und m die Anzahl der Kanten im zugrundeliegenden gerichteten Graphen sind. Wenn wir topologisches Sortieren in der vorgeschlagenen naiven Weise verwenden wollen, müßten wir zuerst diesen Graphen erstellen. Bei n „Dingen“ hat dieser Graph aber $\Theta(n^2)$ Kanten, so daß die Gesamtlaufzeit dann nur $\Theta(n + m) = \Theta(n^2)$ ist.

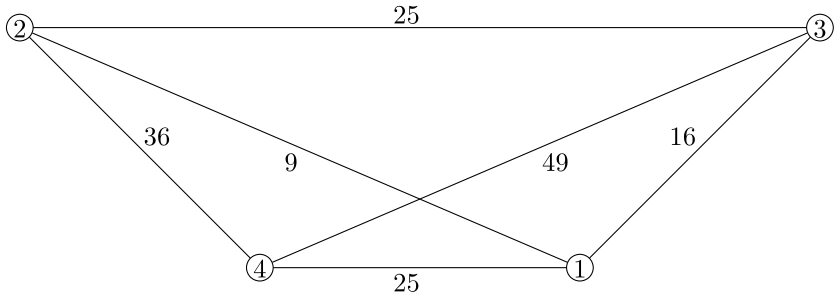
Aufgabe H22 (5 Punkte)

Wir betrachten folgenden ungerichteten Graphen: Die Knoten sind die Zahlen $\{1, 2, 3, 4\}$ und es gibt Kanten zwischen allen Knotenpaaren. Darüber hinaus hat eine Kante zwischen den Knoten i und j das Gewicht $(i + j)^2$.

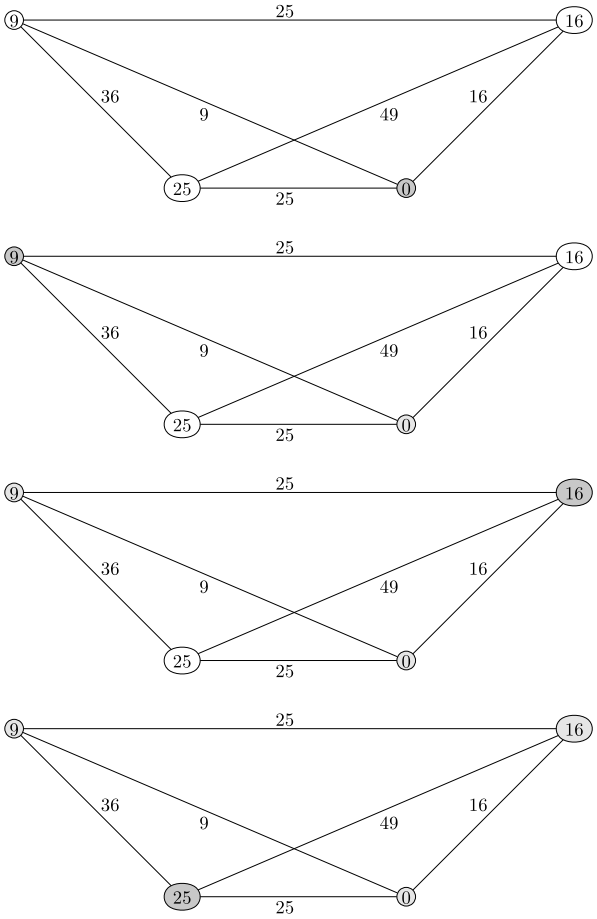
Führen Sie den Algorithmus von Dijkstra auf diesem gewichteten Graphen aus, wobei Sie 1 als Startknoten verwenden. Geben Sie alle Zwischenschritte an.

Lösungsvorschlag:

Der Graph sieht anfangs so aus:



Die vier einzelnen Relaxationsschritte liefern folgende Zwischenergebnisse:



Zuletzt sind alle Entfernungen richtig. Aber ab der zweiten Iteration wurde nichts mehr verändert, also bestehen alle kürzesten Wege aus je nur einer Kante.

Aufgabe H23 (10 Punkte)

Stewie möchte aus dem Jahr 2018 mit seiner Zeitmaschine in das Jahr 1889 zurück reisen um die Geschichte zu ändern. Leider funktioniert seine Zeitmaschine momentan nur eingeschränkt. Sie hat drei Optionen: Falls man momentan im Jahr x ist, kann man in das Jahr $x + 7$, $2x$ oder (falls x durch drei teilbar ist) $x/3$ reisen. Geben Sie eine kürzestmögliche Folge von Zeitsprüngen an, um in das Jahr 1889 zu reisen.

Hinweis: Verwenden sie Breitensuche und lassen Sie die meiste Arbeit von einem Computer erledigen.

Reichen Sie Ihren Quellcode ebenfalls mit ein.

Lösungsvorschlag:

2018 + 7 = 2025
2025 / 3 = 675
675 / 3 = 225
225 / 3 = 75
75 / 3 = 25
25 + 7 = 32
32 + 7 = 39
39 / 3 = 13
13 + 7 = 20
20 + 7 = 27
27 * 2 = 54
54 * 2 = 108
108 + 7 = 115
115 * 2 = 230
230 * 2 = 460
460 + 7 = 467
467 * 2 = 934
934 + 7 = 941
941 * 2 = 1882
1882 + 7 = 1889

```
public class Zeitreise {
    public static void main(String[] args) {
        Map<Integer, String> reached = new HashMap<Integer, String>();
        Map<Integer, Integer> predecessor = new HashMap<Integer, Integer>();
        Queue<Integer> q = new LinkedList<Integer>();
        int start = 2018;
        int target = 1889;
        q.add(start);
        System.out.println("Start");
        while(true) {
            int x = q.remove();
            System.out.println(x);
            if(x == target) {
                break;
            }
            if(!reached.containsKey(x + 7)) {
                q.add(x + 7);
                predecessor.put(x + 7, x);
                reached.put(x + 7, "" + x + " + 7 = " + (x + 7));
            }
            if(x%3 == 0 && !reached.containsKey(x/3)) {
                q.add(x/3);
                predecessor.put(x/3, x);
                reached.put(x/3, "" + x + " / 3 = " + (x/3));
            }
            if(!reached.containsKey(x * 2)) {
                q.add(x * 2);
                predecessor.put(x * 2, x);
                reached.put(x * 2, "" + x + " * 2 = " + (x * 2));
            }

        }
        int x = target;
        while(x ≠ start) {
            System.out.println(reached.get(x));
            x = predecessor.get(x);
        }
        System.out.println(reached.keySet().size());
    }
}
```