Datenstrukturen und Algorithmen, SS 2019
Prof. Dr. P. Rossmanith
S. Dollase, J. Dreier, H. Lotze

Datum: 30.5.2019

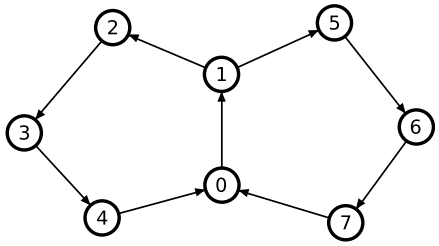
Übungsblatt mit Lösungen 08

Aufgabe T27

Beweisen oder widerlegen Sie: Ergibt eine Tiefensuche in einem gerichteten Graphen genau eine Rückwärtskante, so liefert jede Tiefensuche in diesem Graphen genau eine Rückwärtskante.

Lösungsvorschlag

Die Aussage ist falsch, wie man sich an folgendem Gegenbeispiel überlegen kann.



Beginnen wir die Tiefensuche am Knoten 1, so findet man genau eine Rückwärtskante, nämlich $(0, 1)$ (egal welchen Kreis man zuerst besucht). Eine Tiefensuche, die am Knoten 0 startet, wird jedoch die Rückwärtskanten $(4, 0)$ und $(7, 0)$ finden!

Aufgabe T28

Gegeben sei ein gerichteter Graph G und zwei seiner Knoten s und t.

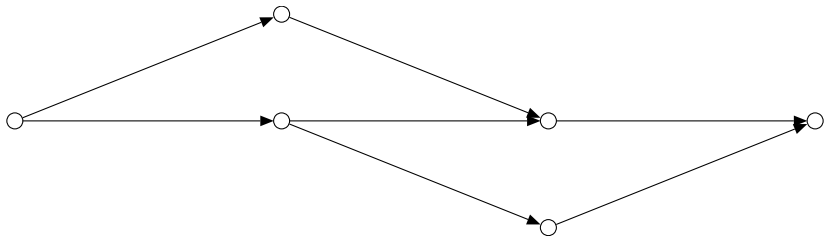
Es soll festgestellt werden, ob es zwischen s und t mindestens zwei kantendisjunkte Pfade gibt. Überlegen Sie, was das genau bedeutet.

Nun entwirft ein genialer Erfinder folgenden Algorithmus, um dieses Problem zu lösen:

> *Nun, zuerst suchen wir nach irgendeinem Pfad zwischen s und t. Wenn es keinen gibt, dann gibt es natürlich auch keine zwei kantendisjunkte Pfade. Das kann ich mit Tiefensuche in linearer Zeit bewerkstelligen.*
>
> *Falls ich einen solchen Pfad finde, dann lösche ich einfach alle Kanten aus G, die auf dem Pfad liegen. Dadurch kann ich keinesfalls eine Kante aus Versehen zweimal verwenden.*
>
> *Jetzt suche ich einfach wieder nach einem Pfad von s nach t. Finde ich einen, dann gibt es zwei kantendisjunkte Pfade zwischen diesen Knoten. Wenn nicht, dann eben nicht. Die Gesamtlaufzeit ist jedenfalls linear.*

Was sagen Sie dazu?

Lösungsvorschlag
Leider funktioniert dieses Verfahren nicht, wie folgendes Beispiel zeigt. Wird zunächst der „mittlere“ Pfad von links nach rechts gewählt und gelöscht, kann dann kein zweiter Pfad mehr gefunden werden.



Aufgabe T29
Der Algorithmus von Bellman und Ford kann die Abstände eines Knotens s zu allen anderen Knoten finden, falls es keinen Kreis mit negativem Gewicht gibt.
Entwerfen Sie einen Algorithmus, der nicht nur feststellen kann, ob es einen Kreis mit negativem Gewicht in einem gerichteten Graphen mit Kantengewichten gibt, sondern einen solchen auch finden und ausgeben kann.

Lösungsvorschlag
Lassen wir den Algorithmus von Bellman und Ford zunächst einmal laufen und feststellen, dass es einen Kreis mit negativem Gewicht gibt. Die erste Schleife lassen wir aber nicht $n-1$ sondern n-mal durchlaufen.
Nun müssen wir nur von jedem Knoten aus die Vorgängerkanten zurückverfolgen und sehen, ob wir wieder den Ursprungsknoten erreichen. Wenn das passiert, haben wir einen gewünschten Kreis gefunden.

Aufgabe T30
Professor Rossmanith hat in seiner Spezialvorlesung *Kreative Fragestellungen für wehrlose Informatiker* fünf Prüflinge $\{A, B, C, D, E\}$, welche eine mündliche Prüfung bei ihm ablegen müssen. Da Professor Rossmanith keine Lust hat, sich für jeden Prüfling neue Fragen auszudenken, schickt er seine Assistenten, um mehr über die Prüflinge zu erfahren.
Nach monatelanger Recherche berichten die Assistenten ihm, dass es Zwist zwischen einzelnen Prüflingen gibt und diese sich nicht gegenseitig helfen wollen. Der Sachverhalt ist der folgende, dabei bedeutet die Relation $X < Y$, dass Y das eigene Wissen an X weitergeben würde: $A < C, B < A, D < A, B < E, D < B, E < C$. Beachten Sie, dass diese Relation nicht symmetrisch ist. Gibt es eine Prüfungsreihenfolge, in der keine Absprachen zwischen den Prüflingen geschehen? Wenn ja, wie lautet diese?

Lösungsvorschlag
D, B, A, E, C

Aufgabe H22 (3+3+4 Punkte)
In einem ungerichteten Graph ist jede Rückwärtskante auch eine Vorwärtskante sowie jede Vorwärtskante auch eine Rückwärtskante. Beweisen Sie die folgenden Aussagen:

a) Ein ungerichteter Baum mit $n \geq 1$ Knoten hat genau $n - 1$ Kanten.

b) Eine Tiefensuche in einem ungerichteten Graphen liefert keine Querkanten.

c) Ergibt eine Tiefensuche in einem ungerichteten Graphen genau eine Rückwärtskante, so liefert jede Tiefensuche in diesem Graphen genau eine Rückwärtskante.

Lösungsvorschlag

a) Dies kann man sich einfach veranschaulichen, indem man in einem Baum willkürlich eine Wurzel festlegt und alle Kanten von der Wurzel weg orientiert—auf diese Weise zeigt nun genau eine Kante auf jeden Knoten, abgesehen von der Wurzel, auf welche keine Kante zeigt. Alternativ kann man auch Induktion benutzen: Ein Baum mit einem Knoten hat null Kanten. Ein neu eingefügtes Blatt erhöht die Kantenzahl um eins.

b) Angenommen eine Tiefensuche würde eine Querkante zischen zwei Knoten u und v liefern. Das heißt v ist kein Nachfahre von u und u ist ein Nachfahre von v. Sei u der Knoten mit der geringeren discovery Zeit. Während u grau markiert ist, ist v also noch weiß markiert. Eine Tiefensuche würde die Kante von u nach v als Baumkante aufnehmen. Dies ist ein Widerspruch. Somit kann es keine Querkanten geben.

c) Stellen wir zunächst fest, dass wir uns auf zusammenhängende Graphen beschränken können: Sollte der Graph mehrere Komponenten besitzen, so kann natürlich nur in einer eine Rückwärtskante auftauchen.

   Diese Komponente besteht also aus den Baumkanten, einer Rückwärtskante und laut b) keiner Querkante. Angenommen die Komponente hat n Knoten. Laut a) gibt besteht der Tiefensuchbaum aus $n-1$ Kanten. Insgesamt gibt es also n Kanten.

   Jeder Tiefensuchbaum muss, da wir ihn als Spannbaum unseres Graphen auffassen können, $n-1$ Baumkanten besitzen. Da der Graph n Kanten besitzt, muss die verbleibende Kante eine Rückwärtskante sein.

Aufgabe H23 (10 Punkte)
Ein *deterministischer Büchi-Automat* (DBA) ist ein Automat, welcher auf einem endlosen Eingabewort läuft und dieses genau dann akzeptiert, wenn während dieses Laufs ein Endzustand des Automaten unendlich oft besucht wird.
Formal ist dieser definiert durch $\mathfrak{A} = (Q, \Sigma, q_0, \delta, F)$, mit der gleichen Bedeutung der Symbole wie bei einem DFA. Für jedes unendliche Wort $\alpha \in \Sigma^\omega$ gibt es einen eindeutigen Lauf ρ von $\mathfrak{A}$ auf α, definiert als $\rho(0) = q_0, \rho(i+1) = \delta(\rho(i), \alpha(i))$. Das Akzeptanzkriterium lautet dann: $\exists^\omega i : \rho(i) \in F$, wobei $\exists^\omega i$ zu lesen ist als *es existieren unendlich viele i*. Unter dieser Aufgabe finden Sie zwei Beispiele für solche Automaten.
Gegeben sei nun ein deterministischer Büchi-Automat $\mathfrak{A}$. Geben Sie einen Algorithmus an, mit dem Sie in Polynomialzeit feststellen können, ob die Sprache, die $\mathfrak{A}$ erkennt, leer ist.

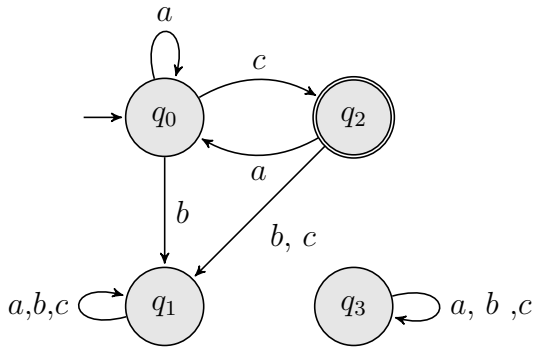


Abb. 1: Ein DBA für die Sprache $L = a^*(ca^+)^\omega$

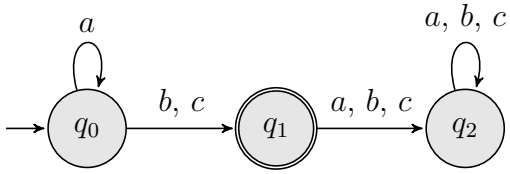


Abb. 2: Ein DBA für die Sprache $L = \emptyset$:

Lösungsvorschlag
Wir können mit einer Tiefensuche vom Startzustand aus alle starken Zusammenhangskomponenten des Automaten finden. Anschließend reicht es aus zu prüfen, ob einer dieser Zusammenenhangskomponenten mindestens eine Kante (inklusive Selfloops auf Endzuständen) und einen Endzustand enthält.

Aufgabe H24 (10 Punkte)

Stewie möchte aus dem Jahr 2019 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 in einer vernünftigen Programmiersprache erledigen.

Reichen Sie Ihren ausführbaren Quellcode als Beleg für ihr Ergebnis mit ein.

Lösungsvorschlag

Der folgende Code löst das Problem:

```
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Queue;

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 = 2019;
        int target = 1889;
        q.add(start);
        while(true) {
            int x = q.remove();
            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());
    }
}
```

Die Ausgabe dieses Programms (in umgekehrter Reihenfolge) lautet:

2019 / 3 = 673
673 + 7 = 680
680 + 7 = 687
687 / 3 = 229
229 + 7 = 236
236 + 7 = 243
243 / 3 = 81
81 + 7 = 88

$88 * 2 = 176$
$176 * 2 = 352$
$352 * 2 = 704$
$704 * 2 = 1408$
$1408 + 7 = 1415$
$1415 * 2 = 2830$
$2830 * 2 = 5660$
$5660 + 7 = 5667$
$5667 / 3 = 1889$