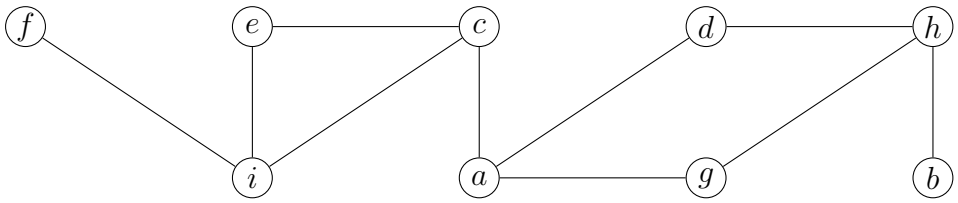
DAP2 – Präsenzübung 11

Besprechung: 05.07.2017 — 07.07.2017

Präsenzaufgabe 11.1: Breitensuche

Gegeben sei der folgende, ungerichtete Graph:



Führen Sie, beginnend bei Knoten a, eine Breitensuche auf diesem Graphen aus. Geben Sie dabei den Inhalt der Warteschlange und die Farbe und Distanz jeden Knotens nach der Initialisierung sowie nach jeder Iteration der **while**-Schleife an. In jeder Iteration sollen neue Knoten gemäß ihrer Reihenfolge im Alphabet an die Warteschlange angestellt werden.

Lösung:

Wir geben die Lösung in einer Tabelle an. Die Knoten für die die Färbung in einer Iteration wechselt, färben wir rot ein.

Iteration	Wert	a	b	c	d	e	f	g	h	i	Warteschlange
0	Farbe	g	w	w	w	w	w	w	w	w	
	Distanz	0	∞	∞	∞	∞	∞	∞	∞	∞	a
1	Farbe	b	w	g	g	w	w	g	w	w	
	Distanz	0	∞	1	1	∞	∞	1	∞	∞	c, d, g
2	Farbe	b	w	b	g	g	w	g	w	g	
	Distanz	0	∞	1	1	2	∞	1	∞	2	d, g, e, i
3	Farbe	b	w	b	b	g	w	g	g	g	
	Distanz	0	∞	1	1	2	∞	1	2	2	g, e, i, h
4	Farbe	b	w	b	b	g	w	b	g	g	
	Distanz	0	∞	1	1	2	∞	1	2	2	e, i, h
5	Farbe	b	w	b	b	b	w	b	g	g	
	Distanz	0	∞	1	1	2	∞	1	2	2	i, h
6	Farbe	b	w	b	b	b	g	b	g	b	
	Distanz	0	∞	1	1	2	3	1	2	2	h, f
7	Farbe	b	g	b	b	b	g	b	b	b	
	Distanz	0	3	1	1	2	3	1	2	2	f, b

Von hier an ändern sich die Distanzwerte nicht mehr und es wird lediglich erst der Knoten f und dann der Knoten b schwarz gefärbt.

Präsenzaufgabe 11.2: (Graphenalgorithmen)

Gegeben sei ein ungerichteter zusammenhängender Graphen $G = (V, E)$, der in *Adjazenzlistendarstellung* gespeichert ist. Ein *Rechteck* in diesem Graph ist dann gegeben, wenn es vier Knoten $v_1, \ldots, v_4 \in V$ gibt, sodass die Kanten $(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1) \in E$ sind.

a) Entwerfen Sie einen Algorithmus, der bei Eingabe eines Graphen $G = (V, E)$ den Wert `TRUE` ausgibt, wenn es mindestens ein Rechteck in G existiert, und `FALSE` sonst. Beschreiben Sie den Algorithmus zunächst mit eigenen Worten. Setzen Sie den Algorithmus dann in Pseudocode um. Für die volle Punktzahl wird ein Algorithmus erwartet, dessen Laufzeit durch $\mathcal{O}(|V|^3)$ beschränkt ist.

b) Analysieren Sie die Laufzeit Ihres Algorithmus.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.

Lösung:

a) Es ist möglich, zunächst einen Algorithmus zu entwickeln, der für einen Knoten $r \in V$ testet, ob dieser auf einem Rechteck liegt. Dieser Algorithmus muss dann nur noch für jeden Knoten des Graphen in geeigneter Weise aufgerufen werden.

Wir folgen der Idee und geben zunächst an, wie man feststellt, ob ein Knoten v auf einem Rechteck liegt. Dazu betrachten wir die Nachbarschaft $N(r) := \{v \in V \mid (r, v) \in E\}$ von r, also alle Knoten, die adjazent zu r sind. Sei dazu $v \in N(r)$.

Betrachten wir die Nachbarschaft von v ohne r, dann erhalten wir die Knoten, $w \in N(v) \setminus \{r\}$, für die es einen Pfad der Länge 2 von r nach w gibt (und zwar den Pfad $(r, v), (v, w)$). All diese Knoten färben wir schwarz, nachdem in einem Initialisierungsschritt alle Knoten weiß gefärbt wurden. Wollen wir aber einen Knoten w schwarz färben, obwohl dieser bereits schwarz gefärbt ist, gibt es zwei unterschiedliche Pfade von r nach w und somit einen Kreis, der r umfasst.

Diese Idee wird vom folgenden Algorithmus ausgenutzt:

TestKnotenRechteck(Graph $G = (V, E)$, Knoten r):

```
1 for v ∈ V do
2     color[v] ← weiß
3 for v ∈ Adj[r] do
4     for w ∈ Adj[v] do
5         if color[w] = schwarz ∧ w ≠ r then
6             return True
7         color[w] = schwarz
8 return False
```

Dieser Algorithmus muss nun lediglich für jeden Knoten des Graphen aufgerufen werden:

```
TestRechteck(Graph G = (V, E)):
1 for r ∈ V do
2     if TestKnotenRechteck(G, r) then
3         return True
4 return False
```

b) Der Algorithmus `TestKnotenRechteck` für einen Knoten r benötigt $\mathcal{O}(|V|)$ Laufzeit für die Initialisierung in den Zeilen 1 und 2, $\mathcal{O}(|V|^2)$ für die verschachtelten for-Schleifen in den Zeilen 3 bis 7 (da man in jeder Iteration die konstante Zeit braucht), und konstante Zeit für die Rückgabe in der Zeile 8. Insgesamt braucht dieser Algorithmus $\mathcal{O}(|V|^2)$ Laufzeit.

Da der Algorithmus `TestKnotenRechteck` für jeden Knoten $r \in V$ aufgerufen werden muss (durch den Algorithmus `TestRechteck`), ist die gesamte Laufzeit $\mathcal{O}(|V|^2) \cdot \mathcal{O}(|V|) = \mathcal{O}(|V|^3)$.

c) Wir müssen zeigen, dass der Algorithmus `TestRechteck` genau dann `TRUE` ausgibt, wenn es in $G = (V, E)$ ein Rechteck gibt (d.h. in zwei Beweisrichtungen).

`FALSE` Nehmen wir an, es gibt kein Rechteck in G, aber unser Algorithmus gibt `TRUE` zurück. Dann müsste eine $r \in V$ existieren, sodass `TestKnotenRechteck`(G, r) `TRUE` zurückgibt. Dann in Zeile 6. müssen es zwei Knoten $w, v_1 \in V$ geben, mit $w \neq r$, $color[w] =$ schwarz, $w \in Adj[v_1]$ und $v_1 \in Adj[r]$. Da $color[w] =$ schwarz, müsste dieser Knoten schon in der Zeile 7. getroffen werden, und zwar nicht durch die Kante (v_1, w), sondern durch eine andere Kante $(v_2, w) \in E$, wobei $v_2 \in Adj[r]$. Dann hätten wir die Kanten (r, v_2), (v_2, w), (w, v_1) und (v_1, r) in E, was ein Rechteck ist. Widerspruch.

`TRUE` Nehmen wir an, es gibt ein Rechteck in G mit den Kanten (r, v_2), (v_2, w), (w, v_1) und (v_1, r), aber es wird `FALSE` zurückgegeben. Wenn`TestKnotenRechteck` mit dem Knoten r als Parameter aufgerufen wird, werden in $Adj[r]$ die Knoten v_1 und v_2 schwarz gefärbt (durch die Zeile 7). Da $w \in Adj[v_1]$ und $w \in Adj[v_2]$, wird zuerst die Menge $Adj[v_1]$ besucht und w in schwarz gefärbt. Danach wird $Adj[v_2]$ besucht und da $w \neq r$ ist, wird in der Zeile 6 `TRUE` zurückgegeben. Widerspruch.

Damit ist die Korrektheit des Algorithmus `TestRechteck` bewiesen.