

Algorithmen und Datenstrukturen

— Übungsblatt 6 —

Betreuer: Melf Johansen

Hauke Stieler
(4stieler@inf)

Philipp Quach
(4quach@inf)

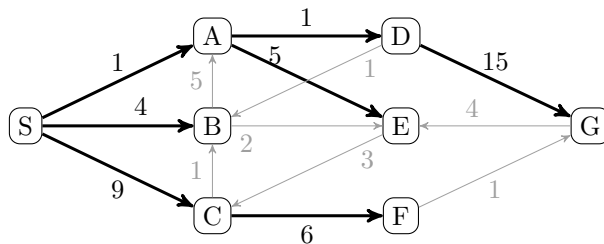
Nils Heine
(4heine@inf)

Abgabe am: 4. Januar 2016

Aufgabe 6.1

Aufgabe 6.1.1

Ein Breitensuchbaum:

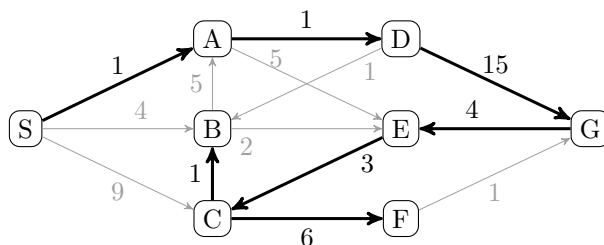


Aufgabe 6.1.2

Das Ergebnis ist nicht eindeutig, da der entstehende Baum davon abhängt, in welcher Reihenfolge die, direkt mit dem aktuellen Knoten verbundenen, Knoten abgearbeitet werden. Wenn zum Beispiel Knoten C vor Knoten A betrachtet wird, würde ein anderer Graph entstehen als bei 6.1.1 angegeben.

Aufgabe 6.1.3

Der Tiefensuchbaum:



Die Zeiten:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(S	(A	(D	(G	(E	(C	(B	(B	(F	F)	C)	E)	G)	D)	A)	S)

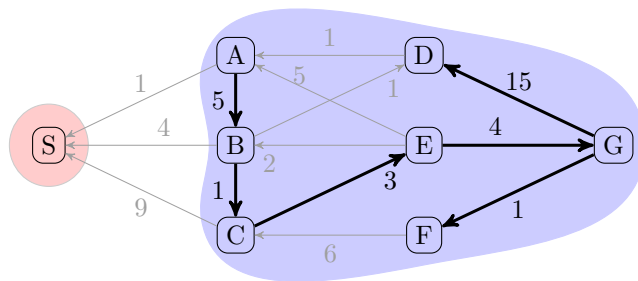
Mit $(X=d(u)$ und $X)=f(u)$.

Aufgabe 6.1.4

G ist zyklisch, dadurch gibt es keine topologische Sortierung. Für eine topologische Sortierung muss der Graph azyklisch sein. Ein Kreis in G ist z.B. die Teilmenge $\{E, G, F, C\}$ von V .

Aufgabe 6.1.5DFS(G)

siehe 6.1.3

DFS(G^T)**Kanten**

$(a, b), (b, c), (c, e), (e, g), (g, f), (g, d)$

Zeiten

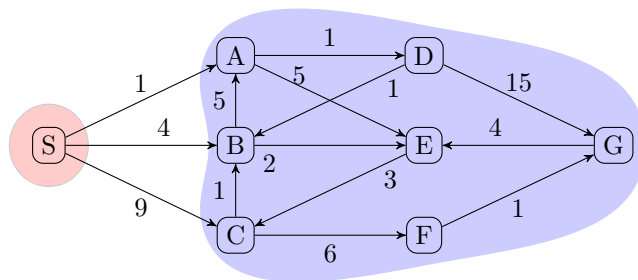
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(S	S)	(A	(B	(C	(E	(G	(F	F)	(D	D)	G)	E)	C)	B)	A)

Starke Zusammenhangskomponenten

1. S
2. A D G E C F B (Alle anderen Knoten)

Aufgabe 6.1.6

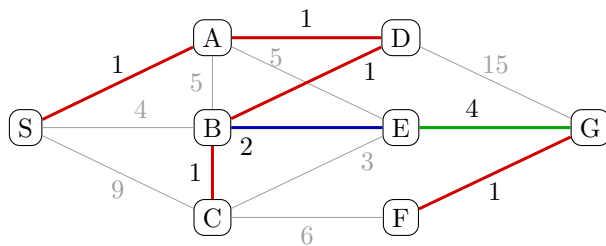
Folgt aus 6.1.5:

 $(S) \rightarrow (ADGECFB)$ 

Aufgabe 6.1.7

Kanten werden in folgender Reihenfolge hinzugefügt:

1. $(s, a), (b, c), (a, d), (b, d), (f, g)$
2. (b, e)
3. (e, g)

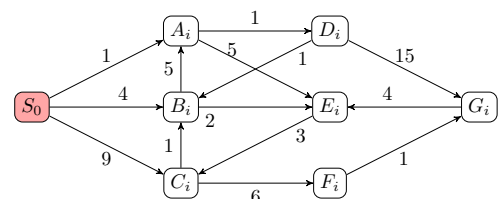
**Aufgabe 6.1.8**

Bei $d(v)$ stehen oben die Knoten, direkt darunter die Werte. Das i steht in diesem Fall für unendlich. Außerdem sind klein geschriebene Knoten bereits fertig abgearbeitet, großgeschriebene Knoten noch nicht.

1. start

$d(v)$:

	S	A	B	C	D	E	F	G
0	i	i	i	i	i	i	i	i

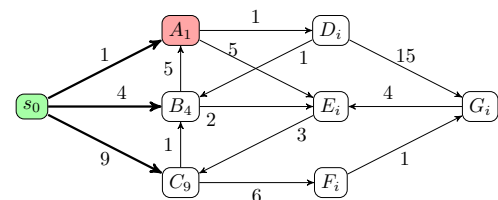
**2. s**

$d(v)$:

	s	A	B	C	D	E	F	G
0	1	4	9	i	i	i	i	

Kanten:

$(s, a), (s, b), (s, c)$

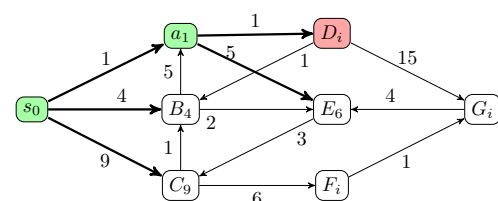
**3. a**

$d(v)$:

	s	a	B	C	D	E	F	G
0	1	4	9	2	6	i	i	

Kanten:

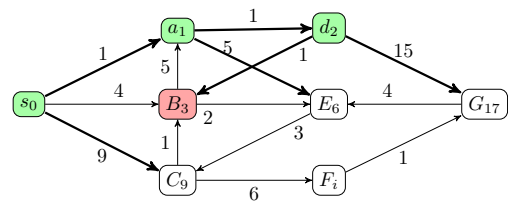
$(s, a), (s, b), (s, c), (a, d), (a, e)$



4. d $d(v)$:

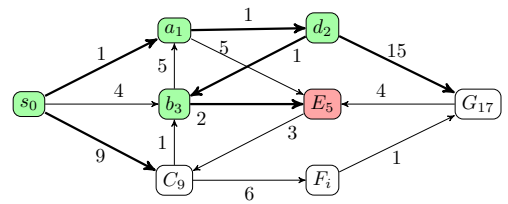
s	a	B	C	d	E	F	G
0	1	3	9	2	6	i	17

Kanten:

 $(s, a), (s, c), (a, d), (a, e), (d, b), (d, g)$ **5. b** $d(v)$:

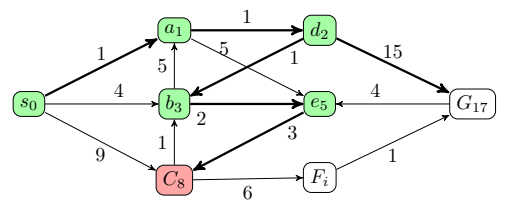
s	a	b	C	d	E	F	G
0	1	3	9	2	5	i	17

Kanten:

 $(s, a), (s, c), (a, d), (d, b), (d, g), (b, e)$ **6. e** $d(v)$:

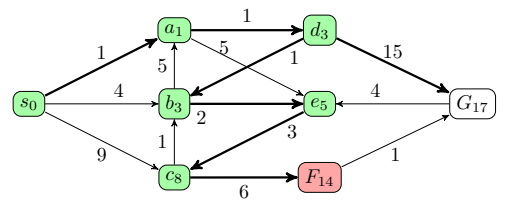
s	a	b	C	d	e	F	G
0	1	3	8	2	5	i	17

Kanten:

 $(s, a), (a, d), (d, b), (d, g), (b, e), (e, c)$ **7. c** $d(v)$:

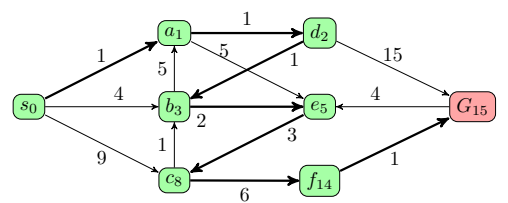
s	a	b	c	d	e	F	G
0	1	3	8	2	5	14	17

Kanten:

 $(s, a), (a, d), (d, b), (d, g), (b, e), (e, c), (c, f)$ **8. f** $d(v)$:

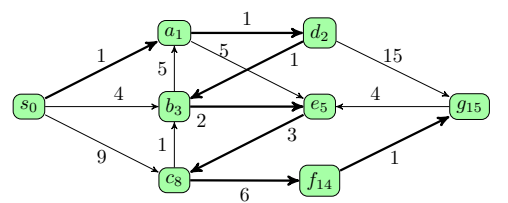
s	a	b	c	d	e	f	G
0	1	3	8	2	5	14	15

Kanten:

 $(s, a), (a, d), (d, b), (b, e), (e, c), (c, f), (f, g)$ **9. g** $d(v)$:

s	a	b	c	d	e	f	g
0	1	3	8	2	5	14	15

Kanten:

 $(s, a), (a, d), (d, b), (b, e), (e, c), (c, f), (f, g)$ 

Fertig

Kürzester Pfad, siehe Schritt 9.

Aufgabe 6.1.9

Der wesentliche Unterschied ist die Entfernung, welche in den Knoten gespeichert wird. Bei **PrimMST** ist das die Entfernung zum Vorgänger Knoten und bei **Dijkstra** – durch das **Relax** (if-Abfrage in der **for**-Schleife) – die Entfernung zum Ursprungsknoten.

Aufgabe 6.2**Aufgabe 6.2.1 Zeigen, dass 8-col NPC ist****Aufgabe 6.2.1.a Zeigen, dass 8-col in NP liegt**

Gegeben sei ein Zertifikat $Z = \langle V, E, f \rangle$ (Graph inklusive Färbungsfunktion). Man kann nun in polynomieller Zeit prüfen ob die gegebene Färbung in Form der Funktion f korrekt ist. Dazu muss man die Funktion durchgehen und nach dem Fall $f(x) = f(y)$ prüfen, was mindestens in $\mathcal{O}(n^2)$ geht. Da dieser Verifikationsalgorithmus in polynomieller Zeit läuft ist 8-col in *NP*.

Aufgabe 6.2.1.b Problem für die Reduktion

Für die Reduktion wird 3-col verwendet, da wir aus der Aufgabenstellung wissen, dass dies ein *NPC* Problem ist.

Aufgabe 6.2.1.c Algorithmus für die Reduktion

Um nun 3-col auf 8-col zu reduzieren fügt man dem gegebenen Graphen G einfach fünf neue Knoten hinzu und von jedem Knoten geht eine Kante zu jedem anderen Knoten des Graphen aus. Dadurch erhalten wir den Graphen G_1 mit den neuen Knoten $v_{n_1}, v_{n_2}, \dots, v_{n_5}$.

Allen Knoten v_{n_i} gibt man nun eine eindeutige Farbe aus $[k]$, wobei $|[k]| = 8$ gilt, da wir ja 8-col behandeln, sprich es existiert keine Kante $\{v_{n_i}, v_{n_j}\} \in E$ mit $i, j \in \mathbb{N}$ wobei $i \neq j$ und $0 < i, j \leq 5$ gilt, sodass $f(v_{n_i}) = f(v_{n_j})$ gilt (Bedingung einer korrekten Färbung zwischen den neuen Knoten). Die restlichen Farben, welche noch nicht verwendet wurden sind $[k_r]$, wobei $|[k_r]| = 3$ gilt (vorher waren 8 Farben da, 5 wurden verwendet, 3 bleiben übrig).

Diese Schritte liegen in polynomieller Zeit, da nur 5 Knoten ausgewählt, 5 Farben vergeben und $5 \cdot |V|$ Verbindungen (zwischen G und den neuen Knoten) gebildet wurden.

Man hat nun einen Graphen G_1 , der fünf gefärbte Knoten besitzt, und eine Funktion $f : \{v_{n_1}, v_{n_2}, \dots, v_{n_5}\} \rightarrow [k]$ für die korrekte Färbung der neuen Knoten. Man muss nun eine Funktion $g : V \rightarrow [k_r]$ finden, welche dann eine korrekte 3-Färbung für G angibt.

Um die Funktion g für den Originalgraphen G zu finden kann man einen Algorithmus für 3-col mit der Eingabe $\langle V, E, [k_r] \rangle$ benutzen.

Die Färbungsfunktion $g : V \rightarrow [k]$ mit dem Graphen G gibt nun eine korrekte 3-Färbung an.

Aufgabe 6.2.1.d

Da 8-col ein *NP* Problem ist und die Reduktion in polynomieller Zeit (s. 3. Absatz) möglich ist, ist 8-col sogar ein *NPC* Problem. □

Aufgabe 6.2.2

Das in der Aufgabe gegebene Problem, ob das Spiel "gewinnbar" ist oder nicht, wird der Einfachheit halber im folgenden als G (für Gewinnbar) bezeichnet.

Aufgabe 6.2.2.a Zeigen, dass $G \in NP$ gilt

Zunächst zeigen wir, dass das Problem in NP liegt indem wir einen deterministischen, in Polynomialzeit laufenden Verifizierungsalgorithmus aufstellen, der uns sagt ob eine vorgegebene Lösung korrekt ist.

Wir überprüfen einfach für alle Zeilen und Spalten ob die entsprechende Bedingung für die Spalte/Zeile erfüllt ist:

1. Wir überprüfen ob es in jeder Zeile mindestens einen Stein gibt und brechen ab wenn einer gefunden wurde. Das hat für jede Zeile im worst-case eine Laufzeit von $\mathcal{O}(n)$, wenn der Stein auf dem letzten abgefragten Feld der Zeile liegt. Für alle Zeilen ergibt das im worst-case eine Laufzeit von $\mathcal{O}(n^2)$, da wir uns dann jedes Feld angeschaut haben.
2. Wir gehen jede einzelne Spalte einmal durch und schauen ob es Steine 2 verschiedener Farben gibt. Dafür schauen wir uns jedes Feld einer Spalte an, da wir sobald ein erster Stein gefunden wurde, für jedes übrige Feld überprüfen müssen, dass kein Stein einer zweiten Farbe drauf liegt. Auch hier ist die Laufzeit $\mathcal{O}(n^2)$.

Zusammen ergibt das eine Laufzeit von $\mathcal{O}(2 \cdot n^2)$, wohl nicht optimal aber immernoch polynomiell. Dies zeigt $P \in NP$.

Aufgabe 6.2.2.b $SAT \leq_p G$ zeigen

Um G in eine Aussagenlogische Formel für einen SAT-Algorithmus umzuwandeln benötigt man zwei Teilformeln. Die erste beschreibt die Korrektheit der Zeilen (keine leeren Zeilen) und die zweite die Korrektheit der Spalten (nur Spalten mit der gleichen Farbe).

In beiden Formeln werden Literale weggelassen für die kein Stein auf dem Spielfeld existiert. Bsp.: Existiert kein Stein auf dem Feld (2,3), so wird das Literal x_{23} weggelassen und kommt in keiner Formel vor.

Zeilenformel

Man bildet eine KNF wobei nur Literale für vorhandene Steine genommen werden, sprich wenn auf dem Feld x_{ij} ein Stein liegt (Farbe ist dabei vernachlässigbar), so wird x_{ij} als Literal in die Klausel mit aufgenommen. Man erhält so eine KNF folgender Form:

$$(x_{1j_1} \vee x_{1j_2} \vee \dots \vee x_{1n}) \wedge (x_{2j_1} \vee x_{2j_2} \vee \dots \vee x_{2n}) \wedge \dots \wedge (x_{nj_1} \vee x_{nj_2} \vee \dots \vee x_{nn})$$

Durch die Konjunktionen wird sichergestellt, dass eine Zeile Korrekt ist, sobald ein Stein beliebiger Farbe vorhanden ist.

Spaltenformel

Bei den Spalten wird ebenfalls eine KNF gebildet, welche pro Spalte zwei Fälle abdeckt: Es sind nur weiße Steine vorhanden *oder* es sind nur schwarze Steine vorhanden.

Dabei wird das vorhanden sein eines weißen Steines durch ein positives Literal, z.B. x_{12} , und das eines schwarzen durch ein negatives Literal, z.B. $\neg x_{23}$, beschrieben.

Es gibt zudem einen Sonderfall: Wenn in einer Spalte entweder nur weiße oder nur schwarze Steine sind, braucht man die Negation nicht mit aufnehmen, da durch das weglassen dieser Steine (also das aufnehmen der negierten Formel) Zeilen leer werden könnten.

Pro Spalte gibt es also zwei konjugierte DNFs, was eine KNF folgender Form ergibt:

$$\begin{aligned} & ((x_{1j_1} \wedge x_{1j_2} \wedge \dots \wedge x_{1n}) \vee (\neg x_{1j_1} \wedge \neg x_{1j_2} \wedge \dots \wedge \neg x_{1n})) \wedge \\ & ((x_{2j_1} \wedge x_{2j_2} \wedge \dots \wedge x_{2n}) \vee (\neg x_{2j_1} \wedge \neg x_{2j_2} \wedge \dots \wedge \neg x_{2n})) \wedge \\ & \vdots \\ & ((x_{nj_1} \wedge x_{nj_2} \wedge \dots \wedge x_{nn}) \vee (\neg x_{nj_1} \wedge \neg x_{nj_2} \wedge \dots \wedge \neg x_{nn})) \wedge \end{aligned}$$

Beide KNFs nun zusammengefügt ergeben die Formel, welche man für den SAT-Algorithmus nehmen kann.

Polynomialzeit

Die Reduktion verläuft in Polynomialzeit, da man einmal zeilenweise über das Feld iteriert und die Zeilen-KNF erstellt (mit einer Laufzeit von $\mathcal{O}(n^2)$) und einmal spaltenweise iteriert und die Spalten-KNF erstellt. Hierbei wird jedoch jede Spalte zwei mal behandelt, somit hat man hier eine Laufzeit von $\mathcal{O}(2 \cdot n^2)$. Zusammen mit der Zeilen-KNF ergibt das eine Laufzeit von $\mathcal{O}(3 \cdot n^2)$, was polynomiell ist.

Auswertung

Gibt nun der SAT-Algorithmus *wahr* zurück, so ist an jedem Feld, welches ebenfalls *wahr* ist ein Stein (Beispiel: Wurde x_{12} zu *wahr* ausgewertet, so befindet sich in der Lösung des Spiels an der Stelle (1,2) ein Stein). Alle anderen Steine können entfernt werden.

Gibt der Algorithmus *falsch* zurück, ist das Spiel mit der gegebenen Anfangsbelegung nicht lösbar.

Dadurch und durch die Tatsache, dass $G \in NP$, ist $G \in NPC$.

□

Aufgabe 6.2.2.c Beispiele zur Erstellung der Formeln

Zum leichteren Verständnis hier noch zwei Beispiele bezüglich der Erstellung der Formeln. Dabei werden schwarze Steine durch $\bullet$ und weiße Steine durch $\circ$ repräsentiert.

Beispiel 1 (lösbares Spiel)

	1	2	3
1	$\bullet$	$\circ$	
2	$\bullet$		$\bullet$
3	$\circ$	$\circ$	$\circ$

Die KNF sieht dann so aus:

$$\begin{aligned}
 & (x_{11} \vee x_{12}) \wedge (x_{21} \vee x_{23}) \wedge (x_{31} \vee x_{32} \vee x_{33}) \\
 & \wedge ((\neg x_{11} \wedge \neg x_{21} \wedge x_{31}) \vee (x_{11} \wedge x_{21} \wedge \neg x_{31})) \\
 & \wedge ((x_{12}) \vee (x_{32})) \\
 & \wedge ((\neg x_{23} \wedge x_{33}) \vee (x_{23} \wedge \neg x_{33}))
 \end{aligned}$$

Eine mögliche erfüllende Belegung wäre somit $x_{11} = 1, x_{12} = 1, x_{21} = 1, x_{23} = 1, x_{31} = 0, x_{32} = 1, x_{33} = 0$. Somit sieht das gelöste Spielfeld wie folgt aus:

	1	2	3
1	$\bullet$	$\circ$	
2	$\bullet$		$\bullet$
3		$\circ$	

Beispiel 2 (nicht lösbares Spiel)

	1	2	3
1		$\bullet$	
2	$\bullet$		
3	$\circ$		

Die KNF sieht dann so aus:

$$(x_{12}) \wedge (x_{21}) \wedge (x_{31}) \wedge ((\neg x_{21} \wedge x_{31}) \vee (x_{21} \wedge \neg x_{31}))$$

Für eine erfüllende Belegung müssten alle Variablen *wahr* sein, damit die Zeilen-KNF (erste Zeile) erfüllt ist. Da jedoch in der Spalten-KNF ein Literal (entweder x_{21} oder x_{31}) *falsch* sein muss damit die Klausel zu *wahr* ausgewertet wird, kann es keine erfüllende Belegung geben.

In anderen Worten:

Das Spiel ist nicht lösbar, da man den Stein (2, 1) oder (3, 1) wegnehmen muss, damit die erste Spalte korrekt ist. Dadurch wird jedoch die Korrektheit der zweiten oder dritten Zeile verletzt, was das Spiel nicht lösbar macht.