Lösungsvorschlag Arbeitsheft 2

1 Kern-Mengen in gerichteten Graphen

Eine Kernmenge $K \subseteq V$ eines gerichteten Graphen $G = (V, E)$ erfüllt folgende Eigenschaften:

$$K^2 \cap E = \varnothing \qquad \text{bzw.} \qquad \forall k \in K : Succ(k) \cap K = \varnothing \tag{1}$$

$$\forall v \in V \setminus K : Succ(v) \cap K \neq \varnothing \tag{2}$$

Dabei ist $Succ(v) := \{w \in V : (v, w) \in E\}$ die Menge Nachfolger eines Knotens.

a)

Sei also $n \in \mathbb{N}$, $V = [1, n], E = \{(i, j) \in V^2 \mid i \neq j\}$ und $G = (V, E)$ gerichtet.
Setze $K = \{1\}$. Es gilt:

$$(1, 1) \notin E \implies (1)$$

$$(V \setminus K) \times \{1\} \subset E \implies (2)$$

Folglich erfüllt K beide Eigenschaften einer Kernmenge.

b)

Im folgenden Schreiben wir $[a]_n$ anstatt $k \pmod n$ für Lesbarkeit.

Sei $C_n := ([1, n], \{(i, [i + 1]_n) \mid i \in [1, n]\})$ der gerichtete Kreis mit n Knoten.

Sei nun $n = 2\ell \geq 3$ für $\ell \in \mathbb{N}$. Setze $K := 2[1, \ell] = \{2, 4, 6, \cdots, 2\ell\} \subset V_{C_n}$. Dann gilt:

$$\forall k \in K : Succ(k) = \{(k, [k + 1]_n)\} \not\subset K^2 \implies (1)$$

$$\forall v \in V_{C_n} \setminus K : Succ(v) = \{(v, v + 1)\} \in V_{C_n} \times K \implies (2)$$

Damit ist K eine Kernmenge von C_n.

Sei nun $n = 2\ell + 1 \geq 3$ für $\ell \in \mathbb{N}$. Angenommen $K \subseteq V_{C_n}$ wäre eine Kernmenge.

Offensichtlich muss dann $K \neq \varnothing$. Sei nun $v \in V_{C_n}$. Dann:

$$v \in K \implies (1) \wedge Succ(v) = \{[v + 1]_n\} \implies [v + 1]_n \notin K$$

$$v \notin K \implies (2) \wedge Succ(v) = \{[v + 1]_n\} \implies [v + 1]_n \in K$$

Sei nun $k \in K$. Wenn man obige Resultate endlich oft iteriert anwendet, so erhält man

$$k \in K \implies [k + 1]_n \notin K \implies [k + 2]_n \in K \implies \cdots \implies [k + 2\ell]_n \in K$$

Jedoch ist $[k + 2\ell]_n = [k - 1]_n$. Ferner gilt $Succ([k - 1]_n) = \{k\}$, damit ist aber (1) verletzt, und K kann keine Kernmenge sein.

Insgesamt haben genau die gerichteten Kreise mit einer geraden Anzahl an Knoten, $C_{2\ell}, \ell \in \mathbb{N}$ eine Kernmenge.

c)

Wir beweisen dies durch Angabe eines Algorithmus und dessen Korrektheitsbeweis.

Der Algorithmus wird die Knoten des Baumes in V färben mit $c : V \to \{0, 1, 2\}$, wobei 0 für nicht gefärbt steht. $c(v) = 1$ soll dabei $v \in K$ bedeuten, und $c(v) = 2$ dann $v \notin K$, wobei K die gesuchte Kernmenge ist.

1. Solange $c^{-1}(0) \neq \varnothing$, es also ungefärbte Knoten gibt:

   (a) Setze $c(k) := 1$ für alle $k \in \{v \in c^{-1}(0) \mid Succ(v) = \varnothing\}$.
   Knoten ohne Nachfolger müssen in K liegen.

   (b) Setze $c(v) := 2$ für alle $v \in c^{-1}(0)$ mit $Succ(v) \cap c^{-1}(1) \neq \varnothing$.
   Knoten mit Nachfolgern in K können wegen (1) nicht in K sein.

   (c) Setze $c(k) := 1$ für $k \in \{v \in c^{-1}(0) \mid \forall w \in Succ(v) : c(w) = 2\}$.
   Knoten die nur Nachfolger haben, welche (schon festgelegt) nicht in K liegen, dürfen in K liegen.

Wir zeigen dass zu jeder Zeit $c^{-1}(1)$ eine Kernmenge von $c^{-1}(\{1, 2\})$ bildet. Ferner sagen wir, dass $\varnothing$ eine korrekte Kernmenge von $\varnothing$ ist.

Sei nun $c^{-1}(1)$ eine Korrekte Kernmenge von $c^{-1}(\{1, 2\})$.
Durch anwenden von (a) werden beide Eigenschaften (1) und (2) einer Kernmenge erhalten, denn:

- Für (1): Sei $k \in V$ einer der gerade gefärbten Knoten, also $c(k) = 1$ und $Succ(k) = \varnothing$. Wegen letzterem, gilt schonmal $Succ(k) \cap K = \varnothing$, also kann k nicht (1) verletzen.

  Betrachte nun $v \in \{w \in c^{-1}(\{1, 2\}) \mid k \in Succ(w)\}$ insofern nichtleer. Wäre $c(v) = 1$, so müsste v durch die Schritte (a) oder (c) des Algorithmus gefärbt worden sein. Wegen $Succ(v) \neq \varnothing$ kann es nicht durch (a) gewesen sein. Da k bis vor kurzem ungefärbt und nun mit 1, also insbesondere nie $c(k) = 2$ gegolten hat, kann dies auch nicht durch Schritt (c) passiert sein. Folglich muss $c(v) = 2$, und damit ist Bedingung (1) nicht verletzt.

- Für (2) spielt das 1-färben (hinzufügen von Knoten zur Kernmenge) keine Rolle.

Folglich ist nach anwenden von (a) $c^{-1}(1)$ immernoch eine korrekte Kernmenge von $c^{-1}(\{1, 2\})$.

Wir untersuchen nun Schritt (b). Sie wieder $c^{-1}(1)$ eine korrekte Kernmenge von $c^{-1}(\{1,2\})$. Es werden durch anwenden von (b) wieder die Bedingungen erhalten:

- Für (1) spielt das 2-färben (hinzufügen zu $V \setminus K$) keine Rolle.
- Für (2): Da jeder der gerade 2-gefärbten Knoten per Vorraussetzung von (b) einen Nachfolger in $c^{-1}(1)$ hat, bleibt die Bedingung (2) für $c^{-1}(\{1,2\})$ erhalten.

Folglich ist nach anwenden von (b) $c^{-1}(1)$ immernoch eine korrekte Kernmenge von $c^{-1}(\{1,2\})$.

Wir untersuchen nun Schritt (c). Sie wieder $c^{-1}(1)$ eine korrekte Kernmenge von $c^{-1}(\{1,2\})$. Es werden durch anwenden von (c) wieder die Bedingungen erhalten:

- Für (1): Sei $k \in V$ einer der gerade gefärbten Knoten, also $c(k) = 1$ und $\forall w \in Succ(k) : c(w) = 2$. Wegen letzterem kann k nicht selbst die Bedingung (1) verletzen.

  Betrachte nun $v \in \{w \in c^{-1}(\{1,2\}) \mid k \in Succ(w)\}$ insofern nichtleer. Wäre $c(v) = 1$, so müsste v durch die Schritte (a) oder (c) des Algorithmus gefärbt worden sein. Wegen $Succ(v) \neq \varnothing$ kann es nicht durch (a) gewesen sein. Da k bis vor kurzem ungefärbt und nun mit 1, also insbesondere nie $c(k) = 2$ gegolten hat, kann dies auch nicht durch Schritt (c) passiert sein. Folglich muss $c(v) = 2$, und damit ist Bedingung (1) nicht verletzt.
- Für (2) spielt das 1-färben (hinzufügen von Knoten zur Kernmenge) keine Rolle.

Folglich ist nach anwenden von (c) $c^{-1}(1)$ immernoch eine korrekte Kernmenge von $c^{-1}(\{1,2\})$.

Wir zeigen nun dass in jeder Iteration mindestens ein Knoten gefärbt werden kann:

Es ist klar, dass jeder (endliche) orientierte Baum mindestens einen Knoten besitzt, welcher keine Nachfolger hat. Andererseits hätte man einen Pfad, in welchem jeder Knoten einen Nachfolger hat. Da Bäume per Definition aber azyklisch sind, hätte man damit per trivialer Induktion einen unendlich langen Pfad in dem orientiertem Baum, was grundsätzlich nicht möglich ist.

Folglich können wir zu beginn mindestens einen Knoten 1-färben (zur Kernmenge hinzufügen).

Sei $c^{-1}(0) \neq \varnothing$, da wir sonst fertig sind, und $Succ(v) \neq \varnothing$ für $v \in c^{-1}(0)$, da Knoten ohne Nachfolger beim ersten (a) gefärbt werden.

Dann gibt es ein $v \in c^{-1}(0)$ mit $Succ(v) \subseteq c^{-1}(\{1,2\})$, also einen ungefärbten Knoten, welcher nur gefärbte Nachfolger hat, da sonst jeder Knoten einen ungefärbten Nachfolger hat und wir damit einen Zykel oder einen unendlichen Pfad in $c^{-1}(0)$ hätten.

Falls nun $Succ(v) \subseteq c^{-1}(2)$, so lässt sich (c) anwenden, andernfalls (b).

Insgesamt lässt sich mindestens ein Knoten in jeder Iteration färben. Da wir von endlichen Eingabebäumen ausgegangen sind, wird der Algorithmus also nach endlich vielen Schritten terminieren. Dann ist aber auch $V = c^{-1}(\{1,2\})$ und wir haben die Kernmenge $c^{-1}(1)$ von V.

d)

Das Zertifikat ist einfach eine Kodierung der Kernmenge selbst. Dies ist offensichtlich kürzer als die Eingabe da $K \subseteq V$. Ferner lässt sich dies in polynomieller Zeit verifizieren, indem man für alle Knoten alle Nachfolger betrachtet und je nach Knoten die Bedingung (1) oder (2) überprüft. Dies geht in quadratischer Zeit.

e) Nein, dies würde Bedingung (1) widersprechen.

f) Nein, dies würde Bedingung (2) verletzen, da $A(x_i)$ und $A(\overline{x_i})$ nur einen ausgehenden Pfeil zum jeweils anderen haben. Wenn beide nicht in der Kernmenge wären, hätten sie keinen Pfeil zu einem Knoten in der Kernmenge.

g) k.

h) Bei m Klauseln werden $3m$ der $B_j(c_i)$-Knoten eingeführt. Für n Variablen werden $2n$ Knoten eingeführt (1 pro Literal). Folglich ist $|V_{G(\Phi)}| = 3m + 2n \leq 3(m+n)$ also polynomiell in $m+n$.

i) Wir haben $2n$ ausgehende Pfeile der Literal-Knoten (1 pro Literal). Jeder der $B_j(c_i)$-Knoten hat pro Literal in Klausel c_i genau 1 ausgehenden Pfeil zum entsprechendem Literal, also 3 Pfeile pro $B_j(c_i)$-Knoten, von denen es $3m$ gibt, was $9m$ ausgehende Pfeile der $B_j(c_i)$-Knoten macht. Insgesamt haben wir $9m + 2n$ Pfeile, was polynomiell in $m+n$ beschränkt ist.

j) Die Anzahl der Kanten / Pfeile in einem Graphen $G = (V, E)$ ist beschränkt durch $|V|^2$. Wenn also $|V|$ schon polynomiell beschränkt in der Eingabe ist, so ist dies auch $|V|^2$.

k) Da jeder Literal-Knoten genau einen ausgehenden Pfeil hat, folgt mit e) und f) schon, dass die Bedingung (1) einer Kernmenge erfüllt ist, da diese nur durch die Knoten der Kernmenge (also hier die Literalknoten) verletzt werden kann.

Der jeweils andere Literalknoten, welcher nicht in der Kernmenge liegt, verletzt die (2) Bedingung nicht, da sein einziger ausgehender Pfeil eben auf sein entsprechendes gegenüber in der Kernmenge zeigt.

Alle Klauselknoten haben Pfeile zu den Literalen, welche die Klausel wahrmachen (und damit in der Kernmenge liegen). Insgesamt ist also Bedingung (2) auch im Graphen erfüllt. Daher bildet die gegebene Menge der Literalknoten eine Kernmenge für $G(\Phi)$.

l) Eine Wahrheitsbelegung weißt jeder Variable x einen Wert zu. Dann hat genau eines der Literale x und $\overline{x}$ den Wahrheitswert 1, sodass per Definition der Kernmenge genau einer der Literalknoten $A(x), A(\overline{x})$ in dieser liegt.

m) Die 3 Klauselknoten $B_1(c), B_2(c), B_3(c)$ einer Klausel c bilden einen C_3. Wären mindestens 2 dieser in der Kernmenge, also B_i, B_j mit $i \neq j$, so hätten wir wegen $B_1 \to B_2 \to B_3 \to B_1$ eine Verletzung der Bedingung (1) einer Kernmenge. Folglich darf höchstens einer der 3 Klauselknoten in K liegen.

n) Sei also $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ eine der Klausel. Angenommen $A(\ell_i) \notin K$ für $i = 1, 2, 3$. Nach m) existiert ein $j \in \{1, 2, 3\}$ sodass $B_j(c) \notin K$ keinen Pfeil auf einen Klauselknoten in K hat. Da $B_j(c)$ aber sonst nur ausgehende Pfeile zu $A(\ell_i)$ für $i = 1, 2, 3$ hat, wäre dann Bedingung (2) der Kernmenge verletzt. Folglich $\exists j \in \{1, 2, 3\} : A(\ell_j) \in K$.

o) Die Wahrheitsbelegung $\varphi : X \to \{0, 1\}$ ist $\varphi(x) := \begin{cases} 1 & , A(x) \in K \quad (\overset{\text{e}}{\iff} A(\overline{x}) \notin K) \\ 0 & , \text{sonst} \end{cases}$

2 Drei-Färbbarkeit von Graphen

Im folgenden betrachten wir ungerichtete Graphen $G = (V, E)$. Sei für $v \in V$ dann

$$Adj(v) := \{w \in V \mid (v, w) \in E \quad \vee \quad (w, v) \in E\}$$

a)

Wir betrachten C_n für $n \geq 3$. Wir schreiben wieder $[k]_n$ für k (mod n).
Entsprechend ist $V_{C_n} = [0, n-1]$.

Fall 1: $n = 2\ell$ für $\ell \in \mathbb{N}$:
Hier lässt sich C_n sogar 2-färben, indem man $c(1+2k) := 1$ für $k \in [0, \ell-1]$ setzt, also alle ungeraden Knoten in V_{C_n} mit 1 färbt. Alle anderen färbt man mit 2, also $c(2k) := 2$ für $k \in [0, \ell-1]$. Dann folgt für $k \in V_{C_n}$ sofort, dass $c([k-1]_n) \neq c(k) \neq c([k+1]_n)$, und da $Adj(k) = \{[k-1]_n, [k+1]_n\}$, damit c eine 2-Färbung von C_n ist.

Fall 2: $n = 2\ell + 1$ für $\ell \in \mathbb{N}$:
Wir setzen $c(0) := 3$. Dann analog zum oberen Fall färben wir die restlichen ungeraden Knoten mit 1, die geraden mit 2, also

$$c(1+2k) := 1 \quad \text{für } k \in [0, \ell-1] \qquad \text{sowie} \qquad c(2k) := 2 \quad \text{für } k \in [1, \ell-1]$$

Dann ist für $c(n) \neq c(0) \neq c(1)$ und $Adj(0) = \{n, 1\}$. Ferner gilt für $k \in [1, n]$, dass $c([k-1]_n) \neq c(k) \neq c([k+1]_n)$ und $Adj(k) = \{[k-1]_n, [k+1]_n\}$, also Insgesamt c eine 3-Färbung von C_n ist.

Folglich lässt sich C_n in jedem Fall 3-Färben.

b)

Wir betrachten einen endlichen Baum $T = (V, E)$ mit maximaler Tiefe $d \in \mathbb{N}$. Sei $D(v) \in [0, d]$ die Ebene bzw Tiefe eines Knotens $v \in V$. T lässt sich 2-färben, indem man

$$\forall v \in \{w \in V \mid D(v) \text{ gerade}\} : c(v) := 1 \quad \text{und} \quad \forall v \in \{w \in V \mid D(v) \text{ ungerade}\} : c(v) := 2$$

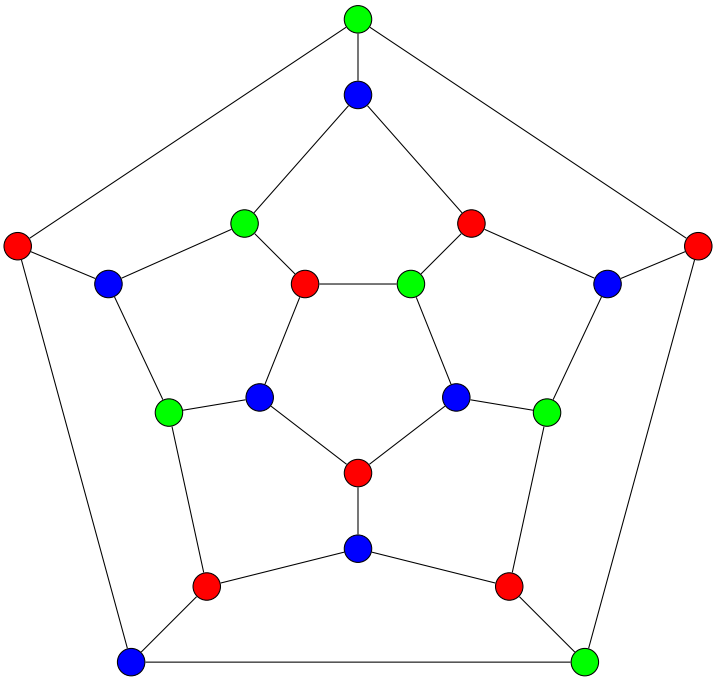
setzt. Dank der Struktur des Baumes gilt

$$\forall v \in V : \forall w \in Adj(v) : [D(w)]_2 \neq [D(v)]_2$$

also dass alle zu v adjazenten Knoten auf ebenen mit anderer Parität als v liegen.

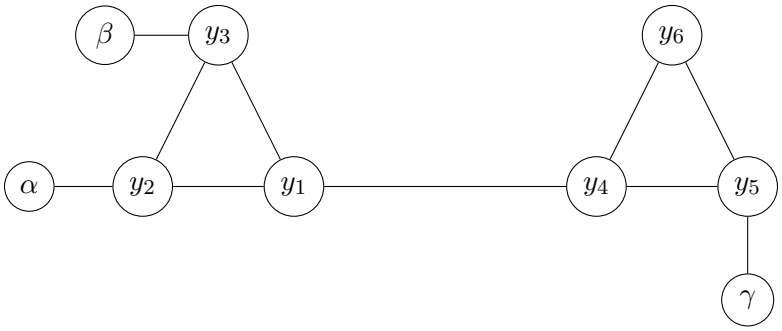
Seien nun $u, v \in V$. Wir fügen die Kante (u, v) zu E hinzu. Falls $c(u) \neq c(v)$, so muss nichts geändert werden. Falls jedoch $c(u) = c(v)$ so setzen wir $c(u) := 3$, und benutzen unsere dritte Farbe. Dann ist c eine korrekte 3-Färbung des veränderten T.

c)



d) Das Zertfikat ist einfach die 3-partition der Knoten hintereinandergeschrieben, welches offensichtlich als permutation der Knoteneingabe der Länge her polynomiell in der Eingabelänge beschränkt ist. Zuerst überprüft man das Zertifikat auf Vollständigkeit, dass auch wirklich jeder Knoten genau 1 mal vorkommt. Dann betrachtet man der Reihe nach die Knoten einer Farbe und überprüft für jeden, dass sie nicht adjazent zu einem der Knoten mit der selben Farbe sind. Dies dauert maximal $\mathcal{O}(|V|^2)$ Operationen.

e)



f)

Sei also $c(\alpha) = c(\beta) = c(\gamma) = 1$. Dann muss $c(y_1) = 1$, da $\{c(y_2), c(y_3)\} = \{2, 3\}$. Folglich ist $c(y_4) \in \{2, 3\}$, wodurch mit $c(\gamma) = 1$ dann auch $\{c(y_4), c(y_5)\} = \{2, 3\}$ folgt. Also bleibt nur die Wahl $c(y_6) = 1$.

g) Betrachte einfach alle Fälle:

α	β	γ	y_1	y_2	y_3	y_4	y_5	y_6
1	1	1	1	2	3	2	3	1
1	1	2	1	2	3	2	3	1
1	1	3	1	2	3	3	2	1
1	2	1	3	2	1	2	3	1
1	2	2	3	2	1	2	3	1
1	2	3	3	2	1	3	2	1
1	3	1	2	3	1	2	3	1
1	3	2	2	3	1	2	3	1
1	3	3	2	3	1	3	2	1
2	1	1	3	1	2	2	3	1

α	β	γ	y_1	y_2	y_3	y_4	y_5	y_6
2	1	2	3	1	2	2	3	1
2	1	3	2	1	3	3	2	1
2	2	1	2	1	3	3	2	1
2	3	1	1	3	2	2	3	1
3	1	1	2	1	3	3	2	1
3	1	2	2	1	3	2	3	1
3	1	3	2	1	3	3	2	1
3	2	1	1	2	3	2	3	1
3	3	1	3	1	2	2	3	1

h) Offensichtlich und auch schon öfters benutzt.

i) Folgt aus h), da $A(x_i), A(\overline{x_i}), DUMMY$ ein Dreieck bilden.

j) k.

k) Es gibt $2n$ Variablenknoten sowie m Gadgets mit jeweils 6 Knoten (α, β, γ werden verschmolzen). Dazu haben wir die 3 Knoten WAHR,FALSCH,DUMMY. Insgesamt gibt es also $2n + 6m + 3 \in \mathcal{O}(m + n)$ Knoten.

l) Es gibt pro Variable 3 Kanten; zwischen den entsprechenden Literalknoten und von beiden zu DUMMY. Ein Gadget hat 10 Kanten an sich. Ferner kommen pro Gadget 2 Kanten hinzu; von y_6 zu FALSCH und DUMMY. Damit haben wir insgesamt $3n + 12m \in \mathcal{O}(m + n)$ Kanten.

m) Sei also $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ eine Klausel aus Φ und φ die erfüllende Variablenbelegung.
Dann ex. $i \in \{1, 2, 3\} : \varphi(\ell_i) = 1$. Da dann $A(\ell_i)$ mit einem der α, β, γ Knoten aus $G_9(c)$ verschmolzen ist, folgt mit g), dass es eine 3-Färbung gibt in der y_6 die Farbe WAHR hat. Da verschiedene Gadgets abgesehen von den Literalknoten nicht miteinander verbunden sind, haben wir damit eine korrekte 3-Färbung von $G(\Phi)$.

n) Dies folgt aus h), da die Knoten y_6, DUMMY, FALSCH ein Dreieck bilden.

o) Da die Knoten α, β, γ eines Gadgets $G_9(c)$ mit den Literalknoten $A(\ell_1), A(\ell_2), A(\ell_3)$ verschmolzen sind, können diese nicht die Farbe DUMMY annehmen. Wenn also keiner der 3 WAHR ist, so müssen alle 3 FALSCH sein. Nach f) müsste dann aber y_6 ebenfalls FALSCH sein, Widerspruch. Folglich muss mindestens einer der 3 Literalknoten WAHR sein.

p) Wenn also $G(\Phi)$ eine 3-Färbung hat, so ist $G_9(c)$ für jede Klausel c der Eingabeformel korrekt gefärbt. Nach o) ist dann mindestens einer der 3 Knoten α, β, γ, welche mit den entsprechenden Literalknoten der Klausel c verschmolzen sind, WAHR gefärbt. Die Wahrheitsbelegung $\varphi : X \to \{0, 1\}$ ist dann

$$\varphi(x) := \begin{cases} 1 & , A(x) \text{ hat Farbe WAHR} \\ 0 & , \text{sonst} \end{cases}$$

Dann ist nämlich pro Klausel mindestens eines der vorkommenden Literale erfüllt und damit Φ erfüllt.

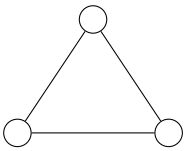
q) Wir gehen von endlichen Graphen aus. Man kann sich pro Zusammenhangskomponente des Graphen einen beliebigen "Startknoten" $v \in V$ wählen. Wir setzen o.B.d.A $c(v) := 0$. Sei nun $k \in V$ gefärbt. Dann setzen wir $c(\ell) := [c(k) + 1]_2$ für alle noch nicht gefärbten Knoten $\ell \in Adj(k)$. Nach endlich vielen Schritten werden alle Knoten gefärbt sein. Nun kann man in polynomieller Zeit testen, ob der Graph korrekt gefärbt ist. Da das vertauschen von Farben semantisch keine Rolle spielt, gibt es praktisch nur diese eine Möglichkeit den Graphen zu 2-färben. Damit lässt sich also in polynomieller Zeit entscheiden, ob ein endlicher Graph 2-färbbar ist.

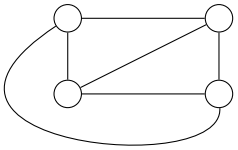
3 Drei-Färbbarktei von planaren Graphen

a) Mit n Knoten hat der Vollständige Graph K_n dann $\binom{n}{2} = \frac{n(n-1)}{2}$ Knoten. Für $n > 5$ ist dann

$$0 < (n-3)(n-4) = n^2 - 7n + 12 \quad \Longrightarrow \quad |E| = \frac{n(n-1)}{2} > 3n - 6$$

was dem Eulerschen Polyedersatz widerspricht, also K_n nicht planar sein kann. Die Fälle $n = 3, 4$ sind wie folgt:





b)

Man kann einen Baum mit maximalem fan-out von $\Delta := \max\{|Adj(v)| - 1 : v \in V\} \in \mathbb{N}$ in die Euklidische Ebene "quetschen", indem man die Kinder eines Knoten v in der n-ten Ebene des Baumes auf ein Liniensegment der Länge $\frac{1}{\Delta^n}$ eine Längeneinheit unter v gleichverteilt. Der Einfachheit halber nehmen wir $\Delta = 2k+1 \geq 3$ für ein $k \in \mathbb{N}$ an, da Plätze für nicht vorhandene Kinder kein Problem darstellen.

Wir geben der Wurzel $r \in V$ die Koordinaten $P(r) := (0, 0)$. Die Kinder eines Knotens $v \in V$ mit Koordinaten $P(v) = (x, n) \in \mathbb{Q} \times \mathbb{N}$ werden dann auf die Positionen

$$\mathcal{P}_v := [x - \frac{1}{\Delta^n}, x + \frac{1}{\Delta^n}, \frac{2}{\Delta^n(\Delta-1)}] \times \{n+1\} = \{(x - \frac{1}{\Delta^n}, n+1), \cdots, (x, n+1), \cdots, (x + \frac{1}{\Delta^n}, n+1)\}$$

verteilt, wobei $[a, b, c] := \{a + nc \mid n \in \mathbb{N}, a + nc \leq b\}$ die lineare Gleichverteilung zwischen a und b mit Distanz c darstellt. Die Kanten bleiben geraden von Knoten zu Knoten.

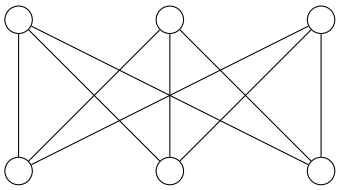
Seien $v, w \in V$ mit $P(v) = (x, n+1), P(w) = (y, n+1)$. Sei o.B.d.A. $x < y$. Dann gilt

$$|(y - \frac{1}{\Delta^{n+2}}) - (x + \frac{1}{\Delta^{n+2}})| = y - x \geq \frac{2}{\Delta^{n+1}(\Delta-1)} > 0$$

sodass die "Kinderbereiche" $\mathcal{P}_v, \mathcal{P}_w$ zweier Knoten der selben Ebene sich niemals überschneiden bzw. überlappen. Damit sind Bäume also planar. Man könnte dies auch mit dem Satz von Kuratowski zeigen, was aber weniger anschaulich ist.

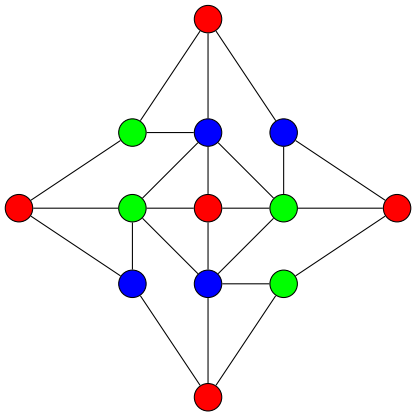
c)

Wir geben ein Gegenbeispiel. Folgender Graph hat 6 Knoten und 9 Kanten, also $|E| = 9 \leq 12 = 3|V| - 6$, ist jedoch ein bekanntes Beispiel für nicht-planare Graphen:

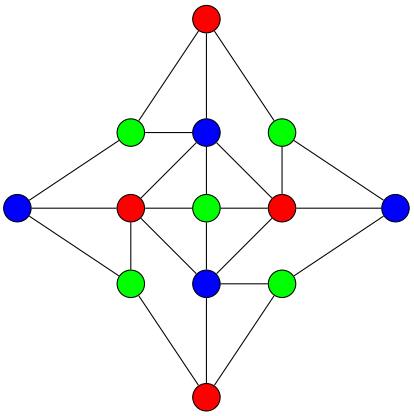


d) Auf offensichtliche weise gilt PLANAR-3-COLORING $\leq_p$ 3-COLORING $\in$ NP und damit auch PLANAR-3-COLORING $\in$ NP.

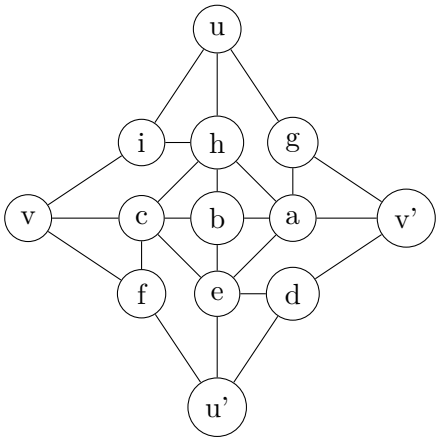
e)



f)



g) Sei die Benennung wie folgt:

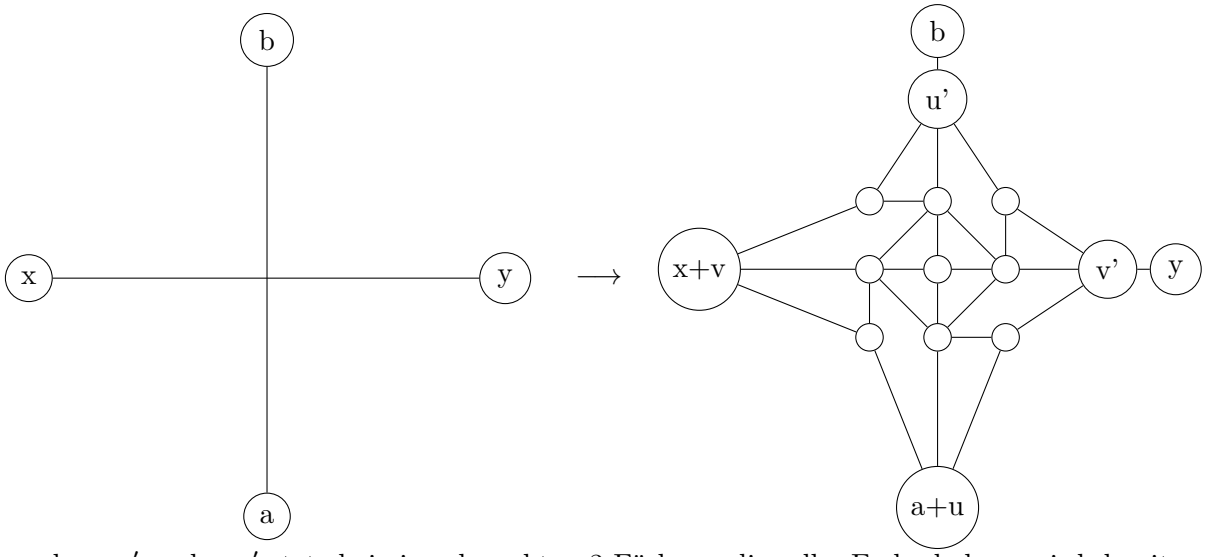


Sei also eine 3-Färbung c gegeben. Wir sagen o.B.d.A. $c(b) = 1, c(a) = c(c) = 2, c(e) = c(h) = 3$. Angenommen $c(u) \neq c(u')$. Wieder o.B.d.A. $c(u) = 1, c(u') = 2$. Es muss $c(d) = 1, c(g) = 3$. Nun haben wir also $c(g) = 3, c(a) = 2, c(d) = 1$ somit kann $c(v')$ nicht korrekt sein.

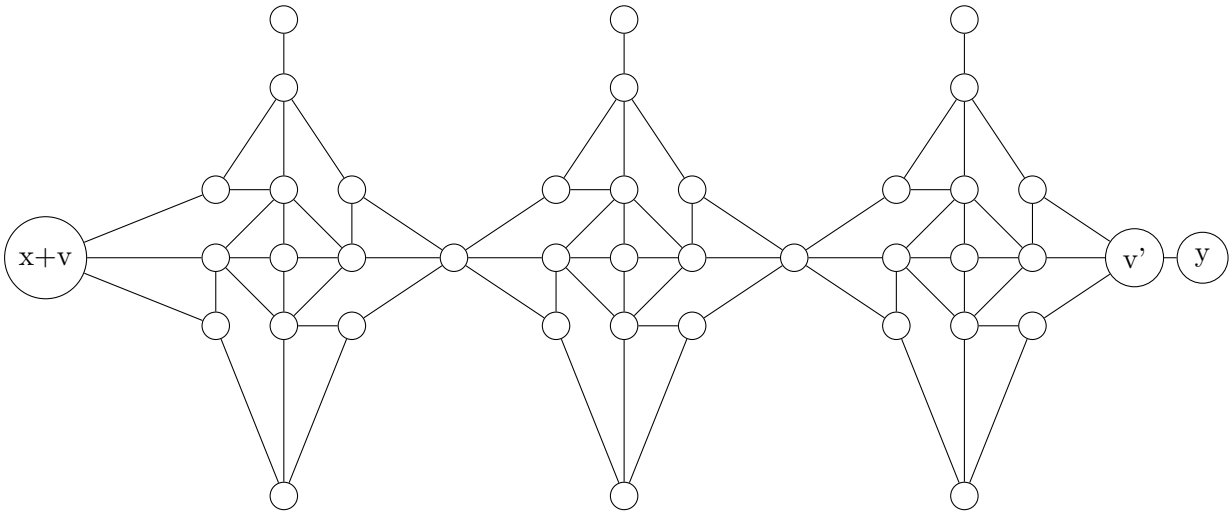
Folglich müssen u, u' in jeder 3-Färbung die selbe Farbe haben. Das selbe gilt für v, v', da der Graph mit vertauschten u's und v's der gleiche wäre.

h) Offensichtlich kann man die Strecken so wählen, dass 2 Strecken höchstens 1 Schnittpunkt haben. Schneidet sich eine Strecke mir mehreren anderen im selben Punkt, so können die Schnittpunkte um vielfache eines geeigneten $\varepsilon > 0$ verschoben werden sodass die Schnittpunkte disjunkt sind. Man hat prakitsch $|\mathbb{R}|$ mögliche Schnittpunkte für jede Kante, was reichen sollte.

i) Seien also S, S' zwei Streckenzüge mit Anfangs und Endknoten $S = (a, b), S' = (x, y)$ und einem Schnittpunkt $p \in \mathbb{R}^2$. Es lässt sich ein $\varepsilon > 0$ finden sodass $K_\varepsilon(p)$ keine weiteren Schnittpunkte enthält. Dieser Kreis wird nun "ausgestanzt" und wir fügen das Kreuzungsgadget hinein, wobei wir u mit a und v mit x verschmelzen, und die Kanten $(u', b), (v', y)$ hinzufügen, sodass sich keine neuen Schnittpunkte bilden:



Denn da u, u' und v, v' stets bei einer korrekten 3-Färbung die selbe Farbe haben, wird damit gesichert, dass a, b und x, y jeweils verschiedene Farben haben. Falls jedoch mehrere Kreuzungsgadgets auf einem Streckenzug von Knoten x zu Knoten y sind, wird das v des ersten Gadgets mit x verschmolzen, und das v' des letzten gadgets mit y verbunden, und alle v, v' dazwischen je paarweise miteinander verschmolzen, also das v' des i-ten Gadgets mit dem v des $i+1$-ten:



Dann müssen $x + v$ und v' immernoch die selbe Farbe haben bei einer korrekten 3-Färbung, und die oben diskutierten Eigenschaften bleiben erhalten.

j) Der resultierende Graph ist planar, da sich in dem ursprünglichen Graphen in jedem Punkt höchstens 2 Streckenzüge geschnitten haben, und die Schnittpunkte nun durch planare Kreuzungsgadgets ersetzt wurden. Damit schneiden sich keine 2 Streckenzüge des resultierenden Graphen.

k) Die Größe eines Kreuzungsgadgets ist konstant. Ferner können sich die Strecken des (sinnvollen) Eingabegraphen in höchstens $|E|^2$ Punkten schneiden (jede Kante jeder andere höchstens 1 mal) sodass wir höchstens $|E|^2$ der Kreuzungsgadgets hinzufügen müssen. Die Schnittpunkte selbst lassen sich auch in polynomieller Zeit, im Notfall mit numerischen Methoden bis auf Maschinengenauigkeit (wo wir dann das Gadget "einstanzen" können), finden.

l) Angenommen der Eingabegraph hat eine 3-Färbung. Die Knoten im planaren Graphen, welche aus dem Eingabegraphen stammen können ihre Farbe übernehmen. In einem Kreuzungsgadget übernehmen die 4 Knoten u, u', v, v' die Farbe des gegenüberliegende Knoten, insofern dieser gerade gefärbt wurde. Mit den Variablennamen der obigen Skizze: Damit hat der Knoten $x + v$ die Farbe von x im Eingabegraphen, und alle verschmolzenen Knoten auf dem Weg durch die 3 Gadgets übernehmen diese Farbe, inklusive dem letzten dieser Knoten, v'. Dies stellt kein Problem für die Kante (v', y) dar, da ja (x, y) eine Kante im Eingabegraphen war und damit $c(v') = c(x) \neq c(y)$. Also lässt sich nach e),f) die 3-Färbung auf die restlichen Knoten der Kreuzungsgadgets erweitern.

Nun angenommen wir haben eine 3-Färbung des resultierenden planaren Graphen G'. Da $V \subseteq V'$, also alle Knoten des Eingabegraphen in V' vorkommen, können wir einfach die Farben übernehmen. Für $x, y \in V$ mit $(x, y) \in E$ haben wir dann:

Fall 1: x und y sind durch eine Kante in G' verbunden, ohne Gadget (also $(x, y) \in E'$).
Dann folgt aber schon sofort $c(x) \neq c(y)$, da c ja eine korrekte 3-Färbung von G' ist.

Fall 2: x und y haben Kreuzungsgadget(s) zwischen sich (also $(x, y) \notin E$).
Dann haben wir einen Fall wie in der Skizze oben, und es folgt dass y zu einem Knoten v' adjazent ist, der per Kontruktion die selbe Farbe wie x haben muss. Folglich $c(y) \neq c(v') = c(x)$, da c eine Korrekte 3-Färbung von G' ist.

Damit folgt in beiden Fällen aber, dass $c(x) \neq c(y)$. Also haben wir eine korrekte 3-Färbung des Eingabegraphen G durch Einschränkung von c auf V.

4 Prozessplanung mit Deadlines

a)

Prozess P_i	Start	Finish	Strafkosten
1	0	1	0
2	1	3	0
3	3	6	0
5	6	11	0
6	11	17	0
4	17	21	1

Damit haben wir einen Prozessplan mit Strafkosten 1. Da alle Strafkoste $s_i \geq 1$ sind, und die Summe der Prozesszeiten $\sum_{i=1}^{6} P_i = 21$ die höchste Deadline $\max\{d_i \mid i \in 1..6\} = 19$ überschreitet, wird man immer Strafkosten $s \geq 0$ zahlen müssen. Folglich habe wir damit eine optimale Lösung.

b)

Sei also eine Instanz mit Prozesszeiten $t_1, \cdots, t_n$, Deadline $d_1, \cdots, d_n$ und Strafkosten $s_1, \cdots, s_n$ sowie eine Optimallösung $\pi \in S_n$ gegeben. Wir definieren

$$S_\pi(i) := \sum_{k \in [1,n], \pi(k) < \pi(i)} t_k \qquad V_\pi(i) :\iff S_\pi(i) + t_i > d_i \qquad K_\pi(i) := \begin{cases} s_i & , V_\pi(i) \\ 0 & , \neg V_\pi(i) \end{cases}$$

Dabei ist $S_\pi(i)$ die Startzeit von P_i in der Optimallösung. $V_\pi(i)$ ist genau dann wahr, wenn P_i in π verspätet ist, und $K_\pi(i)$ sind die zu zahlenden Strafkosten für P_i in der Lösung π.

Insofern $\forall i \in [1,n] : \neg V_\pi(i)$, also kein Prozess verspätet ist, ist nichts zu zeigen.

Es gelte also $\exists i \in [1,n] : V_\pi(i)$. Falls nun

$$\forall i \in [1,n] : \Big(V_\pi(i) \implies (\forall j \in [1,n], \pi(j) > \pi(i) : V_\pi(j)) \Big)$$

also alle verspäteten Prozesse am Ende der abgehandelt werden, ist ebenso nichts zu zeigen.

Also angenommen wir haben $i, j \in [1,n]$ mit $\pi(i) < \pi(j)$ sowie $V_\pi(i)$ aber $\neg V_\pi(j)$. Das heißt, P_j wird nach dem verspäteten Prozess P_i abgearbeitet, ist aber selbst nicht verspätet.

Da nun offensichtlich $S_\pi(i) < S_\pi(j)$, da $\pi(i) < \pi(j)$, folgt damit wegen $S_\pi(j) + t_j \leq d_j$ auch, dass $S_\pi(i) + t_j < S_\pi(j) + t_j \leq d_j$. Folglich gilt für $\sigma := (i,j)\pi \in S_n$ dann ebenfalls $S_\sigma(j) + t_j = S_\pi(i) + t_j \leq d_j$, also $\neg V_\sigma(j)$. Andererseits bleibt $V_\sigma(i)$, da $S_\sigma(i) + t_i = S_\pi(j) + t_i > S_\pi(i) + t_i > d_i$. Folglich, da V_- eine Invariante die Permutation der Prozesse P_i und P_j durch σ ist, gilt dann auch $\sum_{i=1}^{n} K_\pi(i) = \sum_{i=1}^{n} K_\sigma(i)$, also ist σ ebenfalls eine Optimallösung.

Da $n \in \mathbb{N}$ können wir durch endlich maliges anwenden von Permutation obiger Art die verspäteten Prozesse ans Ende der ursprünglichen Optimallösung π Permutieren, ohne die Optimalität zu zerstören.

Insgesamt gibt es immer eine Optimallösung, in der zuerst alle pünktlichen Prozesse und danach erst alle verspäteten Prozesse bearbeitet werden.

c)

Man benutzt die selbe Argumentation wie in b)

d)

Folgt aus c) mit endlich vielen Permutationen der Prozesse, sodass die Deadlines nachher aufsteigend sind.

e)

Entscheidungsvariante: DEADLINES (in Notation von b))

Eingabe: $n, S \in \mathbb{N}, t, d, s \in \mathbb{N}^n$.

Frage: Existiert $\pi \in S_n$ sodass $\sum_{i=1}^{n} K_\pi(i) \leq S$, also eine Reihenfolge für die Abbarbeitung der Prozesse, sodass maximal S Strafkosten gezahlt werden müssen?

f)

Es ist DEADLINE $\in \mathsf{NP}$, da wir als Zertifikat einfach die Permutation π nehmen können, so kodiert dass wir die Zahlen 1 bis n in der entsprechenden Reihenfolge auflisten. Das Zertifikat bestehs aus n Zahlen, welche je durch $\log(n)$ in ihrer Länge beschränkt sind. Also ist die Länge des Zertifikats polynomiell in der Eingabelänge.

Zuerst verifizieren wir, dass das Zertifikat wirklich eine Permutation von $[1, n]$ ist, was mit n Speicherzellen in linearer Zeit geht. Daraufhin haben wir Variablen K, T, die jeweils momentane Strafkosten und Zeit darstellen. Anfangs ist $T := 0$. Dann gehen wir der Reihe nach die Permutation durch. Für ein $i \in [1, n]$ setzen wir dann

$$T := T + t_i \qquad \text{sowie} \qquad K := \begin{cases} K + s_i & , T > d_i \\ K & , \text{sonst} \end{cases}$$

Sodass wir für jeden prozess überprüfen, ob er wenn er an der angegebenen Position startet dann auch vor seiner Deadline fertig wird. Falls dem nicht so ist erhöhen wir die Strafkostenvariable entsprechend. Dies geht in polynomieller Zeit für jede der Prozesse, also insgesamt auch in polynomieller Zeit. Zuletzt überprüfen wir noch ob $K \leq S$, und akzeptieren genau dann wenn dem so ist.

h) Es gilt also $\sum_{i=1}^{n} a_i = 2A$. Wir haben $t_i := a_i, s_i := a_i, d_i = A$. Da wir die a's ja eben in zwei Teilsummen mit jeweiliger Summe A aufteilen wollen, bietet es sich an die Strafkostenschranke $S := A$ zu definieren.

i) Also angenommen die PARTITION-Instanz ist lösbar, d.h. es existiert $I \subset [1, n]$ sodass $\sum_{i \in I} a_i = A$. Dann können wir eine Lösungspermutation konstruieren, indem wir als erstes die Prozesse aus I, und danach die Prozesse aus $[1, n] \setminus I$ bearbeiten. Denn dann werden ja alle Prozesse aus I unabhängig von ihrer Reihenfolge genau zum Zeitpunkt A fertig, da dies die Summe ihrer Laufzeiten ist. Die restlichen Prozesse sind dann zwangsläufig alle verspätet und haben dann in der Summe die Strafkosten $\sum_{i \in [1,n] \setminus I} a_i = A \leq S$. Damit haben wir eine korrekte Instanz von DEADLINE.

j) Nun angenommen die DEADLINE-Instanz besitzt einen Bearbeitungsplan $\pi \in S_n$, sodass $\sum_{i=1}^{n} K_\pi(i) \leq S = A$. Dann definiere $I := \{i \in [1,n] \mid V_\pi(i)\}$. Da die Instanz bezüglich der Schranke $S = A$ korrekt ist haben wir schonmal $\sum_{i \in I} K_\pi(i) \leq A$.

Ferner gilt nun, da ja $\forall i \in [1,n] : \Big(t_i = a_i = s_i\Big) \wedge \Big(V_\pi(i) \iff i \in I\Big)$, dass

$$A = S \geq \sum_{i=1}^{n} K_\pi(i) = \sum_{i \in I} K_\pi(i) = \sum_{i \in I} a_i$$

Sei $J := [1,n] \setminus I$. Angenommen es gilt $A > \sum_{i \in I} a_i$. Dann folgt $A < \sum_{j \in J} a_j$. Ferner gilt aber auch $\forall j \in J : \neg V_\pi(j)$, sodass also alle Prozesse P_j pünktlich sind. Ebenso haben wir aber wegen $t_j = a_j$, dass $\sum_{j \in J} t_j > A$. Dies ist ein Widerspruch, da es nicht möglich ist alle Prozesse sequentiell abzuarbeiten sodass alle vor ihrer Deadline A fertig werden, wenn die Summe ihrer Bearbeitungszeiten $\sum_{j \in J} t_j$ eben diese Schranke A überschreitet.

Folglich muss $A = \sum_{i \in I} a_i$. Dann folgt aber sofort auch $\sum_{j \in J} a_j = 2A$. Damit haben wir unsere gesuchte PARTITION-Instanz $\{I, J\}$.

k) Nein, da PARTITION nicht stark NP-schwer ist.

l) Nein, analog zu Hausaufgabe 12.1, wo wir PARTITION $\leq_p$ PARTITION-INTO-THREE-SETS gezeigt haben, musste man nochmal extra zeigen, dass letzteres in pseudopolynomieller Zeit gelöst werden kann, und damit nicht stark NP-schwer ist. Beispielsweise existiert auch eine Reduktion PARTITION $\leq_p$ THREE-PARTITION, wobei letzteres stark NP-schwer ist.