7 Das Menger-Problem

Das Menger-Problem ist ein „Kernproblem“ vieler Anwendungsprobleme, etwa aus dem Bereich „Verkehrsplanung“, „Schaltkreisentwurf“ oder „Kommunikationsnetzwerke“. Wir werden Linearzeitalgorithmen zur Lösung des kantendisjunkten bzw. knotendisjunkten Menger-Problems in planaren Graphen angeben. Beide Algorithmen beruhen im wesentlichen auf einer Right-First-Tiefensuche.

Eine Menge $S \subset V$ heißt *Separator* von $G = (V, E)$, falls der durch $V \backslash S$ induzierte Subgraph von G unzusammenhängend ist. S trennt die Knoten $u, v \in V \backslash S$, falls u und v in dem durch $V \backslash S$ induzierten Subgraph (bezeichnet mit $G - S$) in verschiedenen Zusammenhangskomponenten liegen. Siehe Abb. 2.3. Wir definieren den *Knotenzusammenhang* $\kappa_G(u, v)$ zweier Knoten u und v bzw. den *Knotenzusammenhang* $\kappa(G)$ des Graphen G wie folgt.

$$\kappa_G(u,v) := \begin{cases} |V| - 1, \text{ falls } \{u,v\} \in E \\ \min\limits_{\substack{S \subset V, \\ S \text{ trennt } u \text{ und } v}} |S|, \text{ sonst.} \end{cases}$$

$$\kappa(G) := \min_{\substack{S \subset V, \\ S \text{ Separator von } G}} \{|S|, |V| - 1\} = \min_{u,v \in V} \kappa_G(u,v)$$

Eine Menge $S \subseteq E$ heißt *Schnitt* von $G = (V, E)$, falls der durch $E \backslash S$ induzierte Subgraph von G unzusammenhängend ist, d.h. in Graphen $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ zerfällt, mit $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$, $E_1 \cup E_2 = E \backslash S$, $E_1 \cap E_2 = \emptyset$, wobei alle Kanten aus S einen Endknoten in V_1 und einen Endknoten in V_2 haben. S *trennt* die Knoten $u, v \in V$, falls u und v in dem durch $E \backslash S$ induzierten Subgraph (bezeichnet mit $G - S$) in verschiedenen Zusammenhangskomponenten liegen. Entsprechend definieren wir den *Kantenzusammenhang* $\lambda_G(u, v)$ zweier Knoten u und v, bzw. den *Kantenzusammenhang* $\lambda(G)$ des Graphen G wie folgt.

$$\lambda_G(u,v) := \min_{\substack{S \subseteq E, \\ S \text{ trennt } u \text{ und } v}} |S|$$

$$\lambda(G) := \min_{\substack{S \subseteq E, \\ S \text{ Schnitt von } G}} |S| = \min_{u,v \in V} \lambda_G(u,v)$$

G heißt *k-fach knoten- bzw. kantenzusammenhängend*, falls $k \leq \kappa(G)$ bzw. $k \leq \lambda(G)$. Zwei Wege in einem Graphen G heißen *(intern) knotendisjunkt*, wenn sie (außer den Endknoten) keine gemeinsamen Knoten enthalten und *kantendisjunkt*, wenn sie keine gemeinsame Kante enthalten.

Satz 7.1. Satz von Menger (1927)
Seien s *und* t *zwei Knoten eines Graphen* G*,* s *und* t *nicht adjazent bei der knotendisjunkten Version.*

- $\kappa_G(s,t) \geq k$ *genau dann, wenn es* k *paarweise intern knotendisjunkte Wege zwischen* s *und* t *in* G *gibt.*
- $\lambda_G(s,t) \geq k$ *genau dann, wenn es* k *paarweise kantendisjunkte Wege zwischen* s *und* t *in* G *gibt.*

MENGER-PROBLEM
Gegeben sei ein Graph $G = (V, E)$ und Knoten s, $t \in V$. Finde eine maximale Anzahl paarweise kanten- bzw. intern knotendisjunkter Wege, die s und t verbinden.

7.1 Das kantendisjunkte Menger-Problem in planaren Graphen

Der im folgenden ausgeführte Algorithmus zur Lösung des kantendisjunkten Menger-Problems in planaren Graphen wurde 1994 von K. Weihe veröffentlicht. Eine Variante des Algorithmus wurde 1997 von Coupry vorgestellt. Beide Algorithmen haben lineare Laufzeit. Betrachte im folgenden einen planaren Graphen $G = (V, E)$ mit einer planaren Einbettung, bei der t auf der äußeren Facette liegt.

Kantendisjunkter Menger-Algorithmus

Schritt 1: Ersetze in Linearzeit $G = (V, E)$ durch den *gerichteten Graphen* $\overrightarrow{G} = (V, \overrightarrow{E})$, der entsteht, indem jede Kante $\{u, v\} \in E$ ersetzt wird durch die beiden *gerichteten Kanten* (u, v), $(v, u) \in \overrightarrow{E}$.

Schritt 2: Berechne in Linearzeit eine Menge geeigneter einfacher kantendisjunkter gerichteter Kreise $\overrightarrow{C_1}, \ldots, \overrightarrow{C_l}$, und betrachte den Graph $\overrightarrow{G_C}$, der aus $\overrightarrow{G}$ entsteht, indem alle Kanten, die auf einem $\overrightarrow{C_i}$ liegen, umgedreht werden.

Schritt 3: Berechne in Linearzeit eine maximale Anzahl von kantendisjunkten gerichteten s-t-Wegen in $\overrightarrow{G_C}$.

Schritt 4: Berechne in Linearzeit aus der Menge der kantendisjunkten s-t-Wege in $\overrightarrow{G_C}$ eine Menge kantendisjunkter s-t-Wege in G gleicher Kardinalität.

Ausführung von Schritt 1: Konstruktion von $\overrightarrow{G} = (V, \overrightarrow{E})$

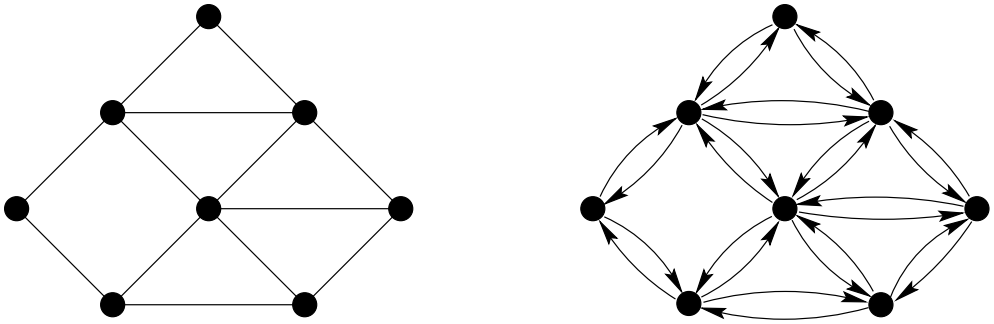


Abbildung 7.1: Ersetzung von $G = (V, E)$ durch $\overrightarrow{G} = (V, \overrightarrow{E})$.

Ersetze $G = (V, E)$ durch den *gerichteten Graphen* $\overrightarrow{G} = (V, \overrightarrow{E})$, der entsteht, indem jede Kante $\{u, v\} \in E$ ersetzt wird durch die beiden *gerichteten Kanten* (u, v), $(v, u) \in \overrightarrow{E}$. Die Begriffe Weg, s-t-Weg und Kreis werden kanonisch übertragen zu den Begriffen *gerichteter Weg, gerichteter* s-t-*Weg* und *gerichteter Kreis.*

Lemma 7.2. *Seien $p_1, \ldots, p_r$ kantendisjunkte, gerichtete* s-t-*Wege in $\overrightarrow{G}$. Dann enthält die Menge $P \subseteq E$, $P := \{\{u, v\}$: genau eine der beiden gerichteten Kanten (u, v) oder (v, u) gehört zu einem der Wege $p_i, 1 \leq i \leq r\}$ gerade r kantendisjunkte Wege* s-t-*Wege in G.*

Beweis.

1. Wenn p_i beide Kanten (u, v) und (v, u) enthält, so ist p_i nicht einfach. Es gibt zu p_i einen einfachen gerichteten s-t-Weg, der weder (u, v) noch (v, u) enthält.
2. Seien nun p_i, p_j gerichtete s-t-Wege, p_i enthalte die Kante (u, v) und p_j enthalte die Kante (v, u). Dann gibt es zwei gerichtete kantendisjunkte s-t-Wege, die alle Kanten aus p_i und p_j enthalten, außer (u, v) und (v, u).

Damit folgt die Behauptung (siehe Abb. 7.2). □

Folgerung 7.3. *Eine maximale Anzahl kantendisjunkter gerichteter* s-t-*Wege $p_1, \ldots, p_k$ in $\overrightarrow{G}$ induziert eine maximale Anzahl kantendisjunkter* s-t-*Wege in G. Diese können offensichtlich in Linearzeit aus $p_1, \ldots, p_k$ konstruiert werden.*

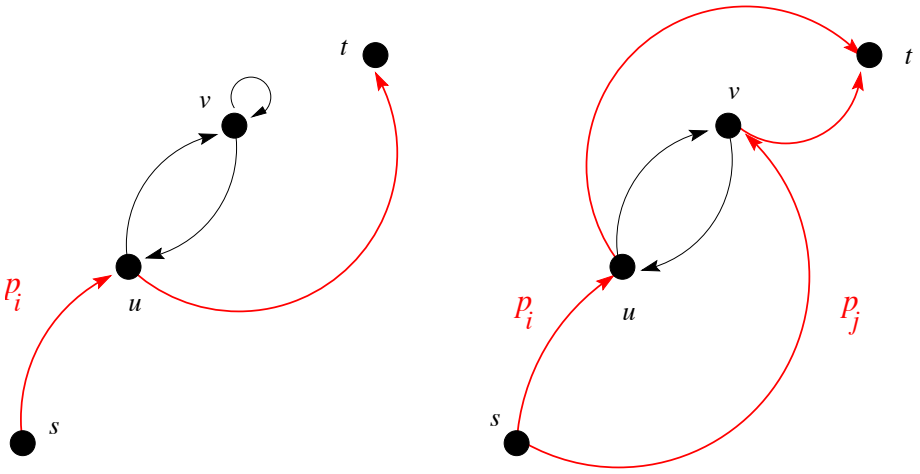


Abbildung 7.2: Illustration von Lemma 7.2

Ausführung von Schritt 2: Konstruktion von $\overrightarrow{G_C}$

Berechne in Linearzeit eine Menge von einfachen kantendisjunkten gerichteten Kreisen $\overrightarrow{C_1}, \ldots, \overrightarrow{C_l}$, so dass der Graph $\overrightarrow{G_C}$, der aus $\overrightarrow{G}$ entsteht, durch Ersetzen aller Kanten (u, v), die auf einem der $\overrightarrow{C_i}$ liegen, durch die Kante $(v, u)'$ folgende Eigenschaften hat.

Eigenschaft 1 $\overrightarrow{G_C}$ enthält keinen Rechtskreis, d.h. keinen einfach gerichteten Kreis, dessen „Inneres“ ($\widehat{=}$ Menge der vom Kreis umschlossenen Facetten, die nicht die äußere Facette enthält) rechts vom Kreis liegt.

Eigenschaft 2 Bilde $\overrightarrow{P_C} \subseteq \overrightarrow{E_C}$ Menge von kantendisjunkten gerichteten s-t-Wegen in $\overrightarrow{G_C}$ und $\overrightarrow{P} \subseteq \overrightarrow{E}$ sei definiert als

$$\begin{aligned}\overrightarrow{P} := (&\overrightarrow{P_C} \cap \overrightarrow{E}) \\ &\cup \{(u, v) \in \overrightarrow{E} : (u, v) \text{ auf einem der } \overrightarrow{C_i} \text{ und } (v, u)' \notin \overrightarrow{P_C}\}.\end{aligned}$$

Dann soll $\overrightarrow{P}$ genau dann eine maximale Menge kantendisjunkter gerichteter s-t-Wege in $\overrightarrow{G}$ sein, wenn $\overrightarrow{P_C}$ eine maximale Menge kantendisjunkter gerichteter s-t-Wege in $\overrightarrow{G_C}$ ist.

Bemerkung.

1. $\overrightarrow{G_C}$ *kann doppelte Kanten* (v, u) *und* $(v, u)'$ *enthalten.*
2. $\overrightarrow{P}$ *entsteht aus* $\overrightarrow{P_C}$*, indem genau die Kanten „herausgenommen“ werden, die in* $\overrightarrow{P_C}$ *liegen und „umgedreht“ wurden, und die Kanten „hinzugenommen“ werden, die nicht in* $\overrightarrow{P_C}$ *liegen und „umgedreht“ wurden.*

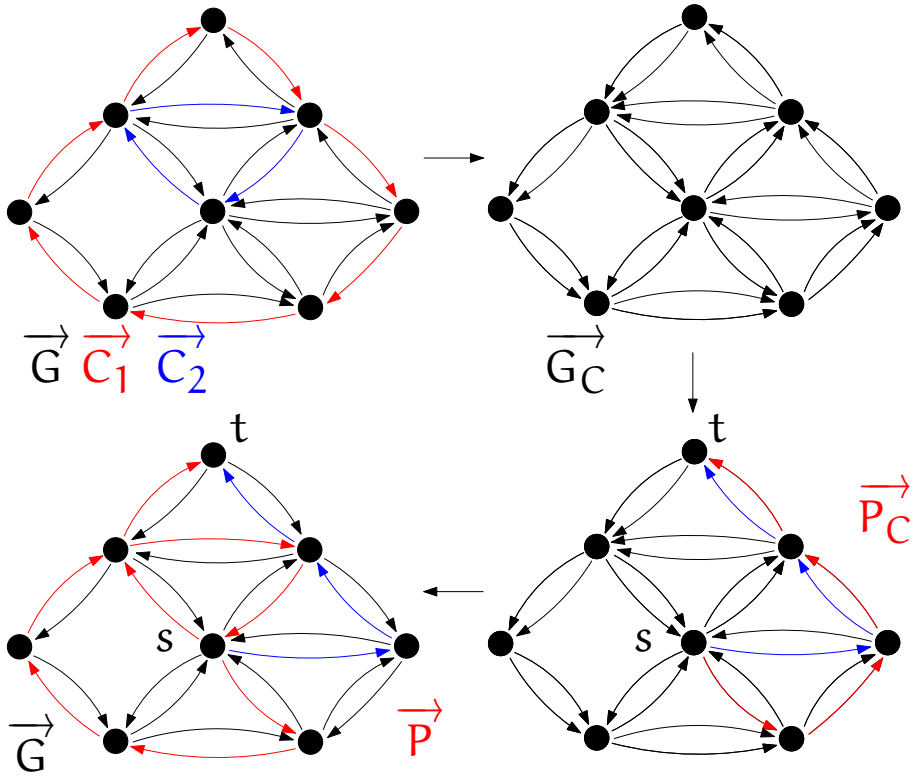


Abbildung 7.3: Illustration von Schritt 2.

Konstruktion der $\overrightarrow{C_1}, \ldots, \overrightarrow{C_l}$

Sei F Menge der Facetten von G. Definiere den *Abstand* einer Facette $f \in F$ von der äußeren Facette f_0, d.h.

$$\text{dist}(f) := \text{Länge eines kürzesten Weges von dem Dualknoten zu } f \text{ zum Dualknoten der äußeren Facette } f_0 \text{ in } G^*.$$

Sei $l := \max_{f \in F} \text{dist}(f)$ und für $1 \leq i \leq l$ sei C_i Vereinigung der einfachen Kreise c_i in G, für die alle Facetten $f \in \text{Inneres}(c_i)$ erfüllen $\text{dist}(f) \geq i$, und alle Facetten $f \in \text{Äußeres}(c_i)$ erfüllen $\text{dist}(f) < i$. Dann seien $\overrightarrow{C_1}, \ldots, \overrightarrow{C_l}$ die $C_1, \ldots, C_l$ entsprechenden Rechtskreise in $\overrightarrow{G}$.

Wir beweisen, dass der durch Umkehren der Kreise $\overrightarrow{C_1}, \ldots, \overrightarrow{C_l}$ in $\overrightarrow{G}$ induzierte Graph $\overrightarrow{G_C}$ die gewünschten Eigenschaften hat.

Zu Eigenschaft 1: Offensichtlich enthält $\overrightarrow{G_C}$ keine Rechtskreise, da von jedem Rechtskreis aus $\overrightarrow{G}$ mindestens eine Kante auf einem der $\overrightarrow{C_i}$ liegen muss.

Zu Eigenschaft 2: Wir benutzen folgendes Lemma über kantendisjunkte s-t-Wege in gerichteten Graphen.

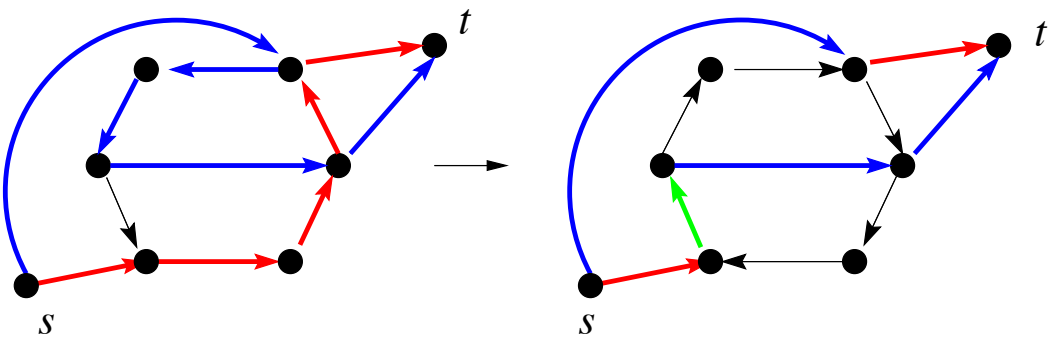


Abbildung 7.4: Illustration von Eigenschaft 2.

Lemma 7.4. *Sei* $\overrightarrow{H} = (V(\overrightarrow{H}), E(\overrightarrow{H}))$ *ein gerichteter zusammenhängender Graph,* $s, t \in V(\overrightarrow{H})$*. Also besteht* $\overrightarrow{H}$ *aus genau* k *kantendisjunkten gerichteten* s-t*-Wegen genau dann, wenn gilt:*

(1) Für alle $v \in V(\overrightarrow{H})\backslash\{s,t\}$ *ist* $d^{\rightarrow}(v) = d^{\leftarrow}(v)$*,*

wobei $d^{\rightarrow}(v)$ *:= # Kanten, die* v *verlassen, und*
$d^{\leftarrow}(v)$ *:= # Kanten, die in* v *hereinführen*

(2) $k = d^{\rightarrow}(s) - d^{\leftarrow}(s) = d^{\leftarrow}(t) - d^{\rightarrow}(t)$

Beweis. „$\Longrightarrow$“ Da jeder s-t-Weg für jeden Knoten v dieselbe Anzahl Kanten zu $d^{\rightarrow}(v)$ wie zu $d^{\leftarrow}(v)$ beiträgt, gilt (1). Jeder s-t-Weg trägt genau eine Kante zu $d^{\rightarrow}(s) - d^{\leftarrow}(s)$, und genau eine Kante zu $d^{\leftarrow}(t) - d^{\rightarrow}(t)$. Also gilt auch (2).

„$\Longleftarrow$“ Wenn jeder Knoten aus $V(\overrightarrow{H})\backslash\{s,t\}$ Bedingung 1 erfüllt, und $d^{\rightarrow}(s) - d^{\leftarrow}(s) = d^{\leftarrow}(t) - d^{\rightarrow}(t)$, so gibt es dabei gerade k s-t-Wege. Damit besteht $\overrightarrow{H}$ also aus k nicht notwendig einfachen, kantendisjunkten s-t-Wegen. □

Sei nun $\overrightarrow{P_C}$ Menge kantendisjunkter gerichteter s-t-Wege in $\overrightarrow{G_C}$. Beim Übergang von $\overrightarrow{P_C}$ zu $\overrightarrow{P}$ ändern sich $d^{\rightarrow}(v)$ und $d^{\leftarrow}(v)$ für jeden Knoten $v \in V$ um den gleichen Betrag. Nur für v auf einem C_i ändert sich $d^{\rightarrow}(v)$ bzw. $d^{\leftarrow}(v)$ und zwar für (v,u), (w,v) aus C_i.

$$(u,v)', (v,w)' \in \overrightarrow{P_C} \quad \text{g.d.w.} \quad (v,u), (w,v) \notin \overrightarrow{P} \tag{1}$$
$$(u,v)' \in \overrightarrow{P_C}, (v,w)' \notin \overrightarrow{P_C} \quad \text{g.d.w.} \quad (v,u) \notin \overrightarrow{P}, (w,v) \in \overrightarrow{P} \text{ bzw.} \tag{2}$$
$$(u,v)' \notin \overrightarrow{P_C}, (v,w)' \in \overrightarrow{P_C} \quad \text{g.d.w.} \quad (v,u) \in \overrightarrow{P}, (w,v) \notin \overrightarrow{P}$$
$$(u,v)', (v,w)' \notin \overrightarrow{P_C} \quad \text{g.d.w.} \quad (v,u)\,(w,v) \in \overrightarrow{P} \tag{3}$$

In $\overrightarrow{P_C}$ gilt

$$k = d^{\rightarrow}(s) - d^{\leftarrow}(s) = d^{\leftarrow}(t) - d^{\rightarrow}(t)$$

g.d.w. in $\overrightarrow{P}$ gilt

$$k = d^{\rightarrow}(s) - d^{\leftarrow}(s) = d^{\leftarrow}(t) - d^{\rightarrow}(t) \ .$$

Folgerung 7.5. *Aus* k *kantendisjunkten* s-t-*Wegen in* $\overrightarrow{G_C}$ *können in Linearzeit* k *kantendisjunkte* s-t-*Wege in* $\overrightarrow{G}$ *berechnet werden.*

Die $\overrightarrow{C_1}, \ldots, \overrightarrow{C_l}$ *können in Linearzeit konstruiert werden (Übung).*

Ausführung von Schritt 3: Berechnung einer maximalen Anzahl von s-t-Wegen in $\overrightarrow{G_C}$

Zur Berechnung einer maximalen Anzahl von kantendisjunkten gerichteten s-t-Wegen in $\overrightarrow{G_C}$ in Linearzeit geben wir eine Prozedur an, die auf einer Right-First-Tiefensuche basiert.

RIGHT-FIRST PROZEDUR ($\overrightarrow{G_C}$, $\overrightarrow{P_C}$)

1: Seien $e_1, \ldots, e_r$ Kanten aus $\overrightarrow{G_C}$, die aus s herauslaufen; $\overrightarrow{P_C} \leftarrow \emptyset$.
2: **Für** $i := 1$ bis r **führe aus**
3: $\quad e \leftarrow e_i$
4: $\quad p_i \leftarrow \{e_i\}$
5: $\quad$ **Solange** Einlaufknoten von e nicht s oder t **führe aus**
6: $\quad\quad \nu \leftarrow$ Einlaufknoten von e
7: $\quad\quad e' \leftarrow$ rechteste freie auslaufende Kante von ν bzgl. „Referenzkante“ von ν
8: $\quad\quad p_i \leftarrow p_i \cup \{e'\}$; $e \leftarrow e'$
9: $\quad\quad$ **Falls** p_i s-t-Weg **dann**
10: $\quad\quad\quad$ setze $\overrightarrow{P_C} \leftarrow \overrightarrow{P_C} \cup \{p_i\}$
11: $\quad\quad$ **Ende** „**Falls**“
12: $\quad$ **Ende** „**Solange**“
13: **Ende** „**Für**“

Version von Weihe: Referenzkante von ν ist jeweils die aktuelle in ν einlaufende Kante e.

Version von Coupry: Referenzkante von ν ist die allererste Kante über die der Knoten ν besucht wird. Diese Kante muss also beim ersten Besuch an ν abgespeichert werden.

Laufzeit: Die Version von Coupry kann direkt in Linearzeit realisiert werden. Die rechteste freie auslaufende Kante bzgl. der Referenzkante ist in diesem Fall immer die

nächste auslaufende Kante nach der zuletzt besetzten in der Adjazenzliste des Knoten, falls diese im Gegenuhrzeigersinn angeordnet ist. Die Gesamtlaufzeit ist also linear in der Anzahl der Kanten in $\overrightarrow{G_C}$ amortisiert über alle Durchläufe $i = 1$ bis r von Schritt 2.

Um die Version von Weihe in Linearzeit zu realisieren, werden die Suchschritte an jedem Knoten als Folge von UNION- und FIND-Operationen ausgeführt. Die hier verwendete Version von UNION-FIND ist linear.

Korrektheit: Wir beweisen die Korrektheit für die Version von Weihe. Die Schleife 5. der RIGHT-FIRST-PROZEDUR endet immer in s oder t, da für jeden Knoten v in $\overrightarrow{G_C}$ per Konstruktion $d^{\rightarrow}(v) = d^{\leftarrow}(v)$. Wir beweisen, dass am Ende $\overrightarrow{P_C}$ eine maximale Anzahl kantendisjunkter gerichteter s-t-Wege enthält. Dazu benutzen wir die gerichtete, kantendisjunkte Version des *Satz von Menger*.

Satz 7.6. *Die Maximalzahl gerichteter kantendisjunkter* s-t-*Wege in einem gerichteten Graphen ist gleich der minimalen Kardinalität eines* s-t-*Schnitts. Dabei ist* $A \subseteq \overrightarrow{E}$ *ein* s-t-*Schnitt in einem (beliebigen) gerichteten Graphen* $\overrightarrow{G} = (V, \overrightarrow{E})$*, wenn* $\overrightarrow{G} - A$ *keinen gerichteten* s-t-*Weg enthält.*

Wir konstruieren basierend auf den in Schleife 5 berechneten $p_1, \ldots, p_r$ (d.h. $\overrightarrow{P_C}$ zusammen mit den p_i, die in s geendet haben) einen gerichteten Kreis $\overrightarrow{K}$ in $\overrightarrow{G_C}$ mit folgenden Eigenschaften:

i) s liegt in Inneres($\overrightarrow{K}$) oder auf $\overrightarrow{K}$,

ii) t liegt in Äußeres($\overrightarrow{K}$),

iii) die Anzahl der Kanten, die von $\overrightarrow{K}$ aus in Äußeres($\overrightarrow{K}$) zeigen, ist gleich der Anzahl der s-t-Wege in $\overrightarrow{P_C}$.

Wenn i – iii gelten, so induziert $\overrightarrow{K}$ einen s-t-Schnitt mit der gewünschten Kardinalität.

Konstruktion von $\overrightarrow{K}$

Seien $p_1, \ldots, p_r$ die Linkskreise und s-t-Wege, die von der RIGHT-FIRST-PROZEDUR berechnet werden. $\overrightarrow{K}$ wird ausgehend von einer Kante (v, s), die von einem der $p_1, \ldots, p_r$ besetzt ist, mittels einer rückwärts gerichteten Suche mit Left-First-Auswahlregel konstruiert. Falls es keine solche Kante (v, s) gibt, so setze $\{s\} := \overrightarrow{K}$. Ansonsten ist $\overrightarrow{K}$ eine Folge von Kanten $(v_r, v_{r-1}) \ldots (v_1, v_0)$ mit $(v_r, v_{r-1}) = (v, s)$. Die Kante (v_{i+1}, v_i) ist die im Uhrzeigersinn nächste Kante nach (v_i, v_{i-1}) in der Adjazenzliste von v_i, die von einem der $p_1, \ldots, p_r$ besetzt ist. (v_1, v_0) ist entweder die erste Kante nach (v_r, v_{r-1}) mit

$v_0 = s$ oder die erste Kante in der Folge, für die die nächste zu wählende Kante bereits zu $\overrightarrow{K}$ gehört.

Nach Konstruktion von $\overrightarrow{K}$ gelten offensichtlich i und ii.

Lemma 7.7. *Betrachte* $\overrightarrow{G_C} = (V, \overrightarrow{E_C})$ *und* $\overrightarrow{K}$. *Jede Kante* $(u, v) \in \overrightarrow{E_C}$ *mit* u *auf* $\overrightarrow{K}$ *und* $v \in$ Äußeres($\overrightarrow{K}$) *gehört zu einem der* s-t-*Wege auf* $\overrightarrow{P_C}$

Beweis. Nach Konstruktion von $\overrightarrow{K}$ zeigt keine Kante, die von einem der Wege und Linkskreise $p_1, \ldots, p_r$ besetzt ist, von Äußeres($\overrightarrow{K}$) auf $\overrightarrow{K}$. Dann kann jedoch auch keine Kante $(u, v) \in \overrightarrow{E_C}$ mit u auf $\overrightarrow{K}$ und $v \in$ Äußeres($\overrightarrow{K}$) von einem der Linkskreise aus $p_1, \ldots, p_r$ besetzt sein.

Zu einem Knoten u auf $\overrightarrow{K}$ betrachte die Kante (u, w) auf $\overrightarrow{K}$. Die Referenzkante von u zeigt von Inneres($\overrightarrow{K}$) auf u oder liegt auf $\overrightarrow{K}$, und deshalb liegt (u, v), mit $v \in$ Äußeres($\overrightarrow{K}$) rechts von (u, w) bzgl. der Referenzkante von u. Dann muß aber (u, v) durch einen s-t-Weg aus $\overrightarrow{P_C}$ besetzt sein, ansonsten wäre vor (u, w) die Kante (u, v) im Algorithmus durch ein p_i besetzt worden. □

Aus Lemma 7.7 folgt dann direkt das folgende Lemma und damit die Korrektheit von Schritt 3.

Lemma 7.8. *Die Menge* $A := \{(u, v) \in \overrightarrow{E_C} : u$ *auf* $\overrightarrow{K}, v \in$ Äußeres($\overrightarrow{K}$)$\}$ *ist ein* s-t-*Schnitt von* $\overrightarrow{G_C}$ *und* $|A| = |\overrightarrow{P_C}|$.

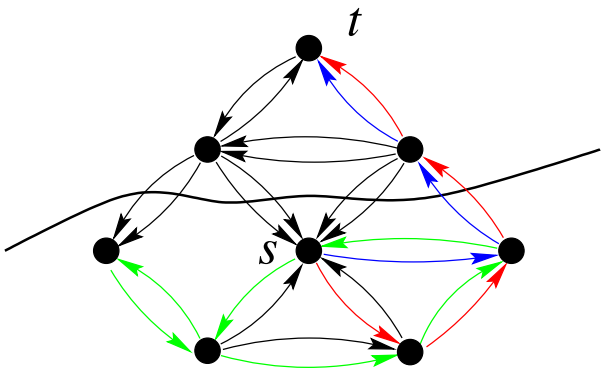


Abbildung 7.5: Wege p_1, p_2 (rot und blau) und der Kreis p_3 in $\overrightarrow{G_C}$. $\overrightarrow{K}$ besteht aus der in s hereinführenden Kante von p_3, gefolgt von der zweiten Kante aus p_1 und vier weiteren Kanten aus p_3, und der durch $\overrightarrow{K}$ induzierte Schnitt A.

Folgerung 7.9. *Der Algorithmus zu Schritt 3 berechnet eine maximale Anzahl kantendisjunkter* s-t-*Wege in* $\overrightarrow{G_C}$. *Damit berechnet der Algorithmus insgesamt in* $\mathcal{O}(n)$ *eine maximale Anzahl kantendisjunkter* s-t- *Wege im planaren Graphen* G.

Ausführung von Schritt 4: Konstruktion einer maximalen Anzahl kantendisjunkter s-t-Wege in G

Berechne in Linearzeit aus der Menge der kantendisjunkten s-t-Wege in $\overrightarrow{G_C}$ eine Menge kantendisjunkter s-t-Wege in G gleicher Kardinalität. Dies geht entsprechend Lemma 7.2 und Folgerung 7.5.

7.2 Das knotendisjunkte Menger-Problem

Betrachte wieder einen planaren Graphen $G = (V, E)$ mit einer planaren Einbettung, bei der t auf der äußeren Facette liegt. Wir behandeln einen Linearzeitalgorithmus zur Lösung des knotendisjunkten Menger Problems.

Knotendisjunkter Menger-Algorithmus

Schritt 1: Ersetze $G = (V, E)$ durch den gerichteten Graphen $\overrightarrow{G} := (V, \overrightarrow{E})$, der entsteht, indem jede Kante $\{u, v\} \in E$ ersetzt wird durch die beiden gerichteten Kanten (u, v), $(v, u) \in \overrightarrow{E}$, falls sie nicht mit s oder t inzident ist; Kanten $\{u, s\} \in E$ nur durch (s, u) und Kanten $\{u, t\} \in E$ nur durch (u, t). Wenn $\overrightarrow{p_1}, \ldots, \overrightarrow{p_l}$ knotendisjunkte s-t-Wege in in $\overrightarrow{G}$ sind, so induzieren diese direkt knotendisjunkte s-t-Wege $p_1, \ldots, p_l$ in G.

Schritt 2: Seien $e_1, \ldots, e_r \in \overrightarrow{E}$ die Kanten aus $\overrightarrow{G}$, die aus s herauslaufen. In einer Schleife über $e_1, \ldots, e_r$ werden knotendisjunkte, gerichtete s-t-Wege mittels einer Suche mit „Right-First“ Auswahlregel konstruiert. Dabei werden „Konflikte“ zwischen bereits besetzten Knoten und dem aktuellen Suchweg geeignet „aufgelöst“.

Ein besetzter Knoten v ist mit genau einer besetzten einlaufenden Kante (u, v) und genau einer besetzten auslaufenden Kante (v, w) inzident. Der Suchweg kann also von links oder von rechts in Bezug auf (u, v), (v, w) auf v treffen.

Behandlung von Konflikten zwischen Suchweg und besetztem Knoten v.

1. Konflikt von links: Der aktuelle Suchweg trifft von links auf einen besetzten Knoten v. Dann wird ein *Backtrack-Remove-Schritt* ausgeführt, d.h. die letzte Kante des Suchwegs vom Suchweg und aus $\overrightarrow{G}$ entfernt.

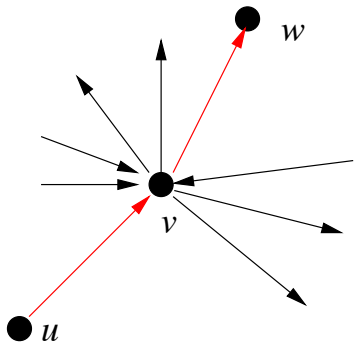


Abbildung 7.6: Eindeutig in v einlaufende und auslaufende besetzte Kanten (u, v) und (v, w).

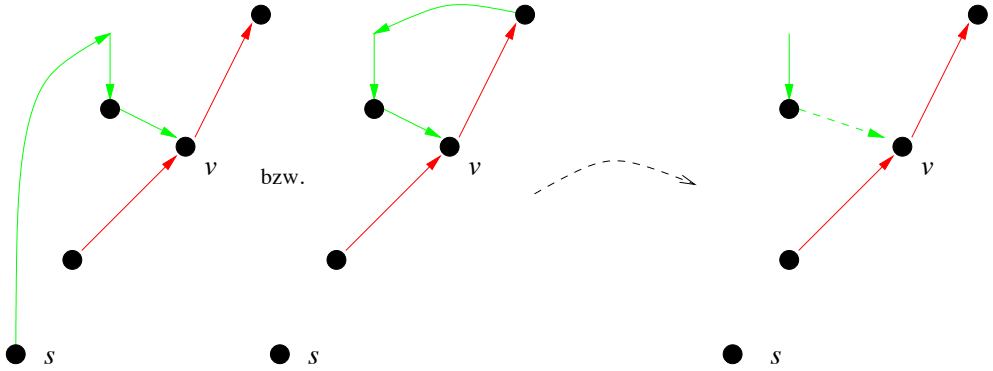


Abbildung 7.7: Konflikt von links.

2. Konflikt von rechts: Der aktuelle Suchweg trifft von rechts auf einen besetzten Knoten v. p sei der Teilweg von s nach v, zu dem die besetzte in v einlaufende Kante gehört, q der Teilweg von v nach t, zu dem die besetzte aus v auslaufende Kante gehört, und r der aktuelle Suchweg. Die Teilwege r und q werden zu einem s-t-Weg zusammengesetzt, und p als aktueller Suchweg betrachtet. Dies kann als eine „Umorganisation von Wegen“ angesehen werden. Danach trifft der aktuelle Suchweg von links auf den besetzten Knoten, d.h. es tritt ein Konflikt von links ein. Dies wird wie gehabt durch einen *Backtrack-Remove-Schritt* aufgelöst. Siehe Abb. 7.8.

Voraussetzung dafür, dass diese „Umorganisation von Wegen“ sinnvoll ist, ist die Bedingung, dass die zu v inzidenten besetzten Kanten nicht zu dem aktuellen Suchweg gehören. Daher ist der Algorithmus so angelegt, dass diese Situation erst gar nicht auftritt, d.h. der Suchweg keinen Rechtskreis durchläuft. Beachte, dass $\overrightarrow{G}$ selbst Rechtskreise enthält.

Trick 1: Man kann beweisen, dass für eine Kante (v, w), über die der Suchweg einen Rechtskreis mit Konflikt in v durchlaufen würde, bereits zu einem früheren Zeitpunkt des Algorithmus die umgekehrte Kante (w, v) in einem Linkskreis mit Konflikt in v durchlaufen wurde. Daher wird eine Kante (v, w) entfernt, für die gilt:

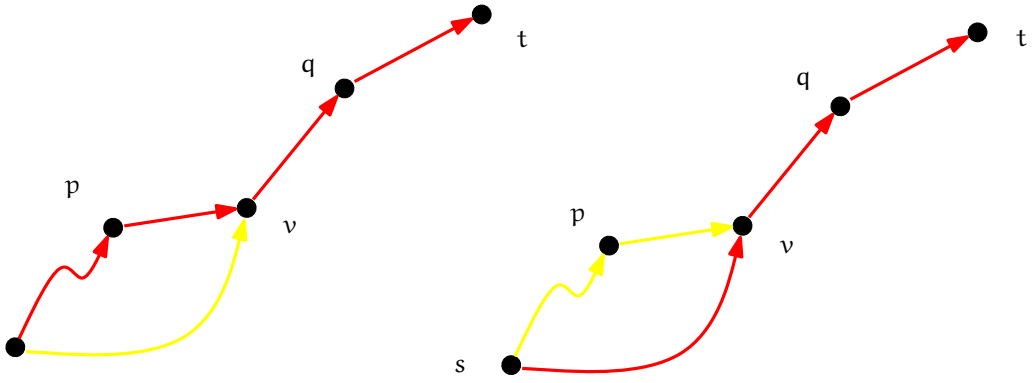


Abbildung 7.8: Umorganisation der Teilwege $\mathtt{p}$, $\mathtt{q}$ (rot) und dem aktuellen Suchweg $\mathtt{r}$ (grün).

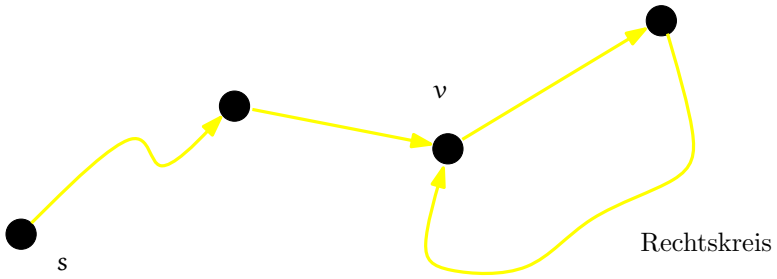
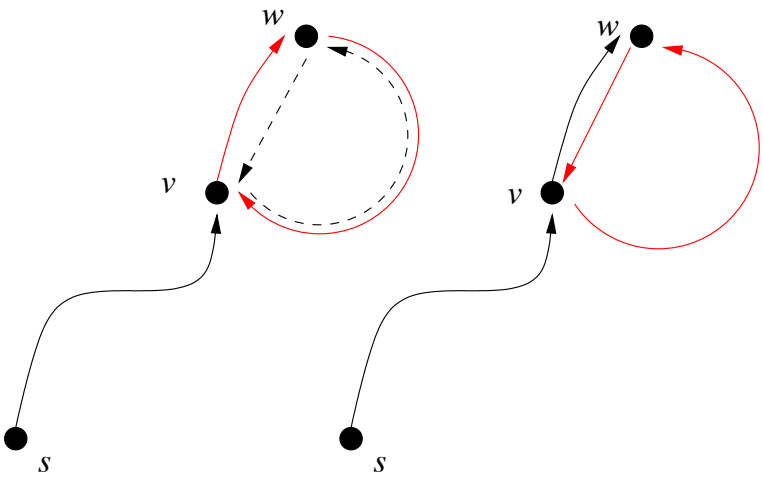


Abbildung 7.9: Konflikt von rechts an einem Knoten v durch einen Rechtskreis im aktuellen Suchweg.



Durchlauf eines Rechtskreis vorher Durchlauf von entsprechendem Linkskreis

Abbildung 7.10: Illustration zu Trick 1.

> (v, w) würde nach Right-First-Auswahlregel als nächste Kante von dem aktuellen Suchweg belegt, und (w, v) ist zuvor einmal von dem aktuellen Suchweg belegt worden.

Um diese Bedingung abzutesten, muss für eine Kante bekannt sein, ob sie bereits von dem aktuellen Suchweg belegt wurde. Es müsste also zu jeder belegten Kante gespeichert werden, von welchem Weg sie belegt ist.

Problem: Belegte Kanten ändern im Laufe des Algorithmus ihre Zugehörigkeit zu Wegen, da die Wege selbst im Laufe des Algorithmus jeweils bei einem Konflikt von rechts umorganisiert werden. Ein Update für alle betroffenen Kanten bei jeder Umorganisation würde jedoch zu quadratischer Laufzeit führen.

Trick 2: Es wird ein „globaler Zähler“ eingeführt, der immer dann erhöht wird, wenn entweder von s aus ein neuer Suchweg startet, oder wegen eines Konflikts von rechts der Suchweg umorganisiert wird. Jeder Knoten erhält einen „lokalen Zeitstempel“. Wird ein Knoten zum führenden Knoten des aktuellen Suchwegs, so wird sein lokaler Zeitstempel auf den aktuellen Wert des globalen Zählers gesetzt.

Es gilt dann: Falls Kante (w, v) im Algorithmus bereits zuvor von dem aktuellen Suchweg belegt wurde, so sind bei Betrachtung der Kante (v, w) als eventuelle nächste vom Suchweg zu belegende Kante, die lokalen Zeitstempel von v und w identisch.

RIGHT-FIRST-PROZEDUR zu Schritt 2

Insgesamt besteht nun Schritt 2 aus einer RIGHT-FIRST-PROZEDUR (Algorithmus 1). Sie beginnt jeweils bei einer aus s herausführenden Kante, und endet entweder bei t oder wieder bei s. Im Laufe des Verfahrens werden knotendisjunkte s-t-Wege konstruiert. Wegen der wiederholt durchgeführten Umorganisation von Wegen, tritt ein einmal konstruierter s-t-Weg nicht unbedingt als solcher in der endgültigen Lösung auf.

Laufzeit: Die Gesamtzahl der Durchläufe von 6. ist linear in der Anzahl der Kanten von $\overrightarrow{G}$, denn jede einzelne Kante von $\overrightarrow{G}$ ist zunächst unbesetzt, wird im Laufe des Algorithmus höchstens einmal besetzt, und höchstens einmal nach Besetzen („ansonsten“ in 14.) bzw. direkt bei erster Betrachtung (in 18.) entfernt. Die Anzahl der Operationen, die für jede Kante ausgeführt wird, ist konstant. Beachte dazu, dass die Auswahl der rechtesten freien herausführenden Kante bzgl. der führenden Kante des aktuellen Suchwegs in 16. in konstanter Zeit ausgeführt werden kann. Die gesuchte Kante ist immer die nächste freie herausführende Kante in der Adjazenzliste. Denn falls für einen Knoten v dieser Auswahlschritt mehrfach ausgeführt wird, so ist die „Referenzkante“ immer dieselbe. Damit ist die Gesamtlaufzeit des Algorithmus in $\mathcal{O}(n)$.

Korrektheit: Wir führen hier keinen ausführlichen Korrektheitsbeweis, da dieser relativ aufwendig ist. Die Korrektheit des Algorithmus kann bewiesen werden, indem die folgenden beiden „Invarianten“ für den Algorithmus bewiesen werden.

Algorithmus 1 RIGHT-FIRST-PROZEDUR zu Schritt 2

1: Seien $e_1, \ldots, e_r$ die Kanten von $\overrightarrow{G} = (V, \overrightarrow{E})$, die aus s herausführen. Die Reihenfolge ist beliebig.
2: Setze Zähler $\leftarrow 0$
3: **Für** $i := 1$ bis r **führe aus**
4: Aktueller Suchweg bestehe aus Kante $e_i := (s, v)$.
5: Setze Zähler $\leftarrow$ Zähler $+ 1$.
6: **Solange** $v \notin \{s, t\}$ **führe aus**
7: Setze Zeitstempel(v) $\leftarrow$ Zähler
8: **Falls** der aktuelle Suchweg in v einen Konflikt von links hat **dann**
9: führe „Backtrack-Remove“ aus.
10: **sonst**
11: **Falls** der aktuelle Suchweg in v einen Konflikt von rechts hat **dann**
12: führe „Umorganisation“ aus.
13: Setze Zähler $\leftarrow$ Zähler $+ 1$.
14: **sonst**
15: **Falls** eine unbesetzte aus v herausführende Kante (verschieden von der Gegenkante zur gerade führenden Kante des aktuellen Suchweges) existiert **dann**
16: wähle die rechteste freie herausführende Kante bzgl. der führenden Kante des aktuellen Suchweges. (v, w) sei diese Kante.
17: **Falls** Falls Zeitstempel(v) = Zeitstempel(w) **dann**
18: entferne (v, w).
19: **sonst**
20: füge (v, w) zum aktuellen Suchweg hinzu.
21: **Ende** „**Falls**“
22: **sonst**
23: führe „Backtrack-Remove“ mit der führenden Kante des aktuellen Suchwegs aus.
24: **Ende** „**Falls**“
25: **Ende** „**Falls**“
26: **Ende** „**Falls**“
27: Setze $v \leftarrow$ führender Knoten des aktuellen Suchwegs.
28: **Ende** „**Solange**“
29: **Ende** „**Für**“

Invariante 1: Der Suchweg formt zu keinem Zeitpunkt des Algorithmus einen Rechtszykel. (Dies bedeutet, dass Trick 1 und 2 greifen.)

Invariante 2: Die maximale Anzahl knotendisjunkter s-t-Wege in $\overrightarrow{G}$ sei k. $\{a_1, \ldots, a_m\}$ seien die Kanten aus $\overrightarrow{G}$, die im Laufe des Algorithmus (in dieser Reihenfolge) entfernt werden. Dann gilt: Der Graph $\overrightarrow{G_i} := (V, \overrightarrow{E} \backslash \{a_1, \ldots, a_i\}), 1 \leq i \leq m$, enthält ebenfalls k knotendisjunkte s-t-Wege.

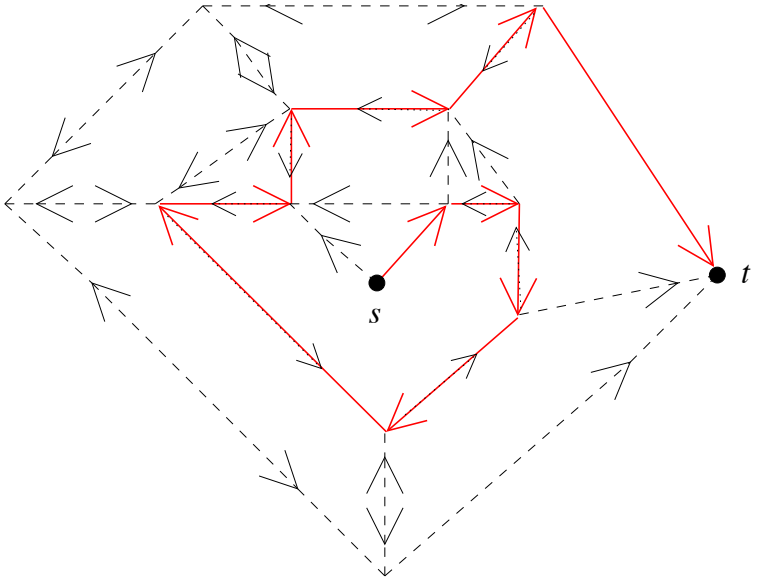


Abbildung 7.11: Die Situation nach dem ersten Durchlauf von 3. mit dem in diesem Durchlauf berechneten s-t-Weg (rot). Zwei gegenläufige Pfeile geben an, dass beide gerichteten Kanten noch im Graph existieren. Beachte, dass rechts des ersten Weges keine auslaufende Kante mehr existiert.

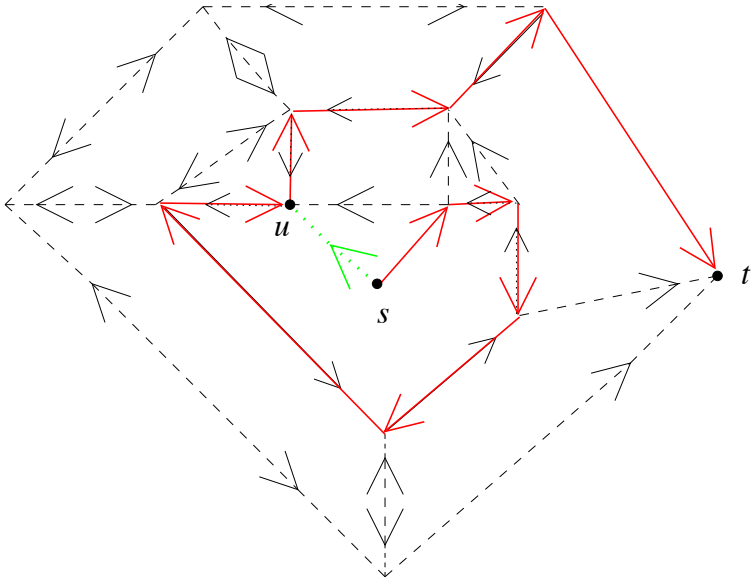


Abbildung 7.12: Die Situation nach dem ersten Suchschritt des zweiten Durchlauf von 3. Der Suchweg (grün gestrichelt), d.h. $s \to u$ trifft im Knoten u von rechts den s-t-Weg, der im ersten Durchlauf konstruiert wurde.

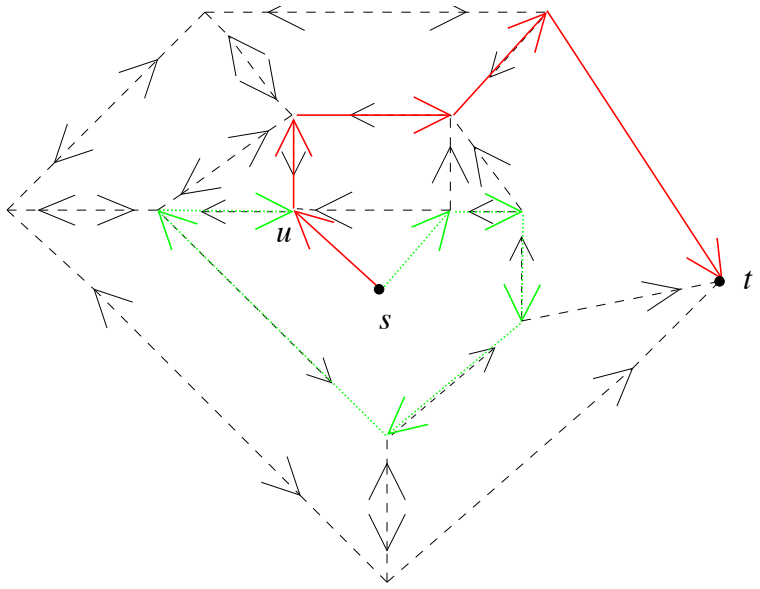


Abbildung 7.13: Der Teilwegvon $s \to u$ des ersten s-t-Weges wird zum neuen Suchweg (grün gestrichelt), und der vorherige Suchweg wird mit dem Teilweg von $u \to t$ des ersten s-t-Weges zu einem s-t-Weg (rot) zusammengesetzt.

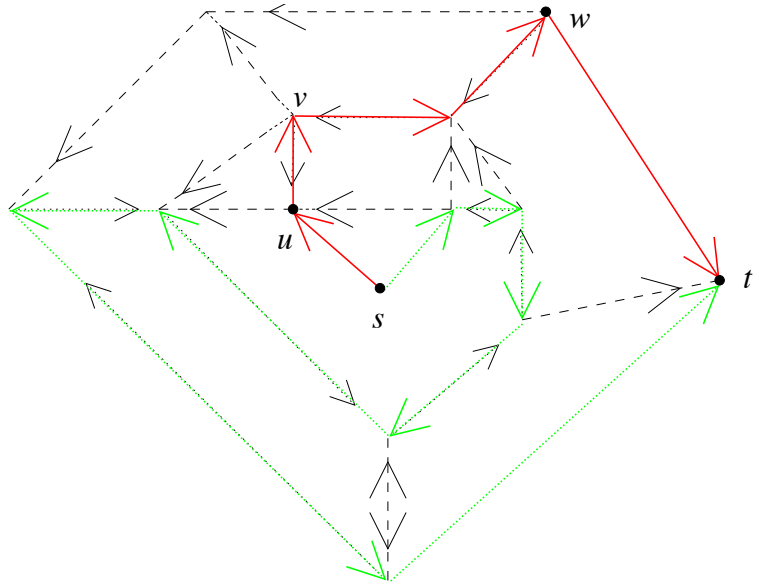


Abbildung 7.14: Alle weiteren Konflikte im zweiten Durchlauf sind Konflikte, bei denen der Suchweg (grün gestrichelt) den konstruierten s-t-Weg (rot) von links trifft, zweimal in v und später einmal in w.

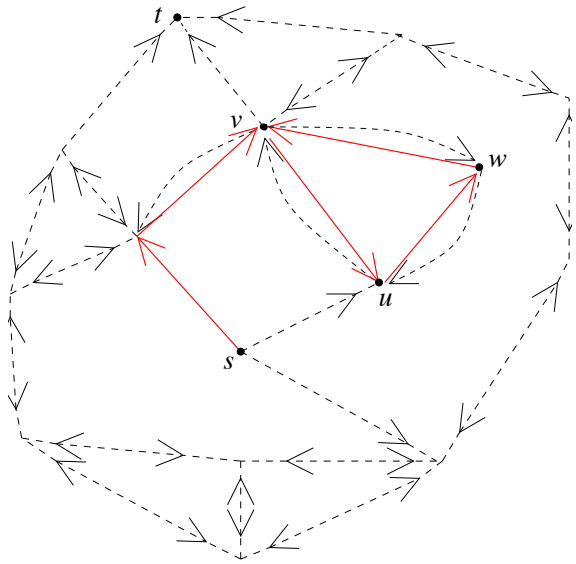


Abbildung 7.15: Ein Beispiel, in dem ein Konflikt auftritt, der den vermeintlichen Durchlauf eines Rechtskreises „ankündigt“.

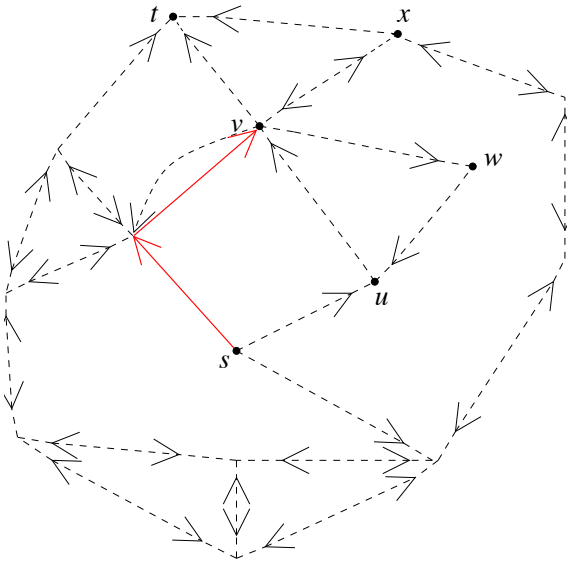


Abbildung 7.16: Nach drei Backtrack-Remove-Schritten. Nun ist (v, w) die nächste Kante, die der Suchweg betrachtet. Die Zeitstempel von v und w sind zu diesem Zeitpunkt identisch, also wird (v, w) nicht gewählt, sondern entfernt und stattdessen (v, x) gewählt. Würde der Algorithmus tatsächlich (v, w) wählen, so würde ein Rechtkreis $v \to w \to u \to v$ durchlaufen. Auflösung des entsprechenden Konflikts von rechts durch Entfernen von (u, v) würde alle optimalen Lösungen zerstören!