Karlsruher Institut für Technologie

Algorithmen für Planare Graphen

19. Juni 2018, Übung 5

Lars Gottesbüren, Michael Hamann

INSTITUT FÜR THEORETISCHE INFORMATIK

Prüfungstermine

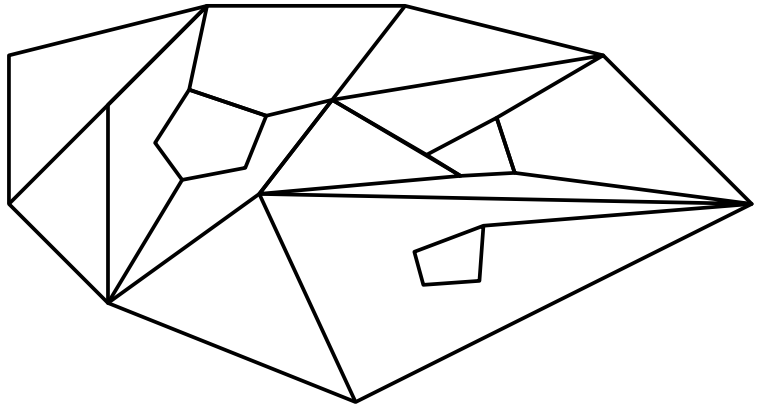
- 11. Juli - AUSGEBUCHT
- 31. Juli - AUSGEBUCHT
- 28. August - AUSGEBUCHT
- 29. August - AUSGEBUCHT
- 27. September
- 4. Oktober
- 15. Oktober - AUSGEBUCHT

1 – Entfernung von Kreisen

Sei F die Menge der Facetten und f_0 die äußere Facette von G. Bezeichne weiter $\text{dist}(f)$ die Länge eines kürzesten Weges vom der Facette f entsprechenden Dualknoten zum f_0 entsprechenden Dualknoten, und $l := \max_{f \in F} \text{dist}(f)$. Für $1 \leq i \leq l$ sei C_i die Vereinigung der einfachen Kreise in G so, dass $\text{dist}(f) \geq i$ für alle Facetten f im Inneren und $\text{dist}(f) < i$ für alle Facetten f im Äußeren eines Kreises aus C_i gilt.

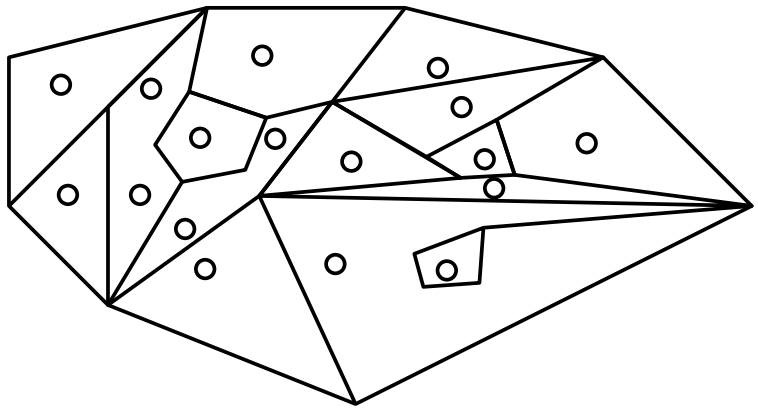
Aufgabe: Geben Sie einen Algorithmus mit linearer Laufzeit an, der zu einem gegebenen Graphen G mit fester Einbettung die Kantenmengen $C_1, \ldots, C_l$ bestimmt.

1 – Entfernung von Kreisen



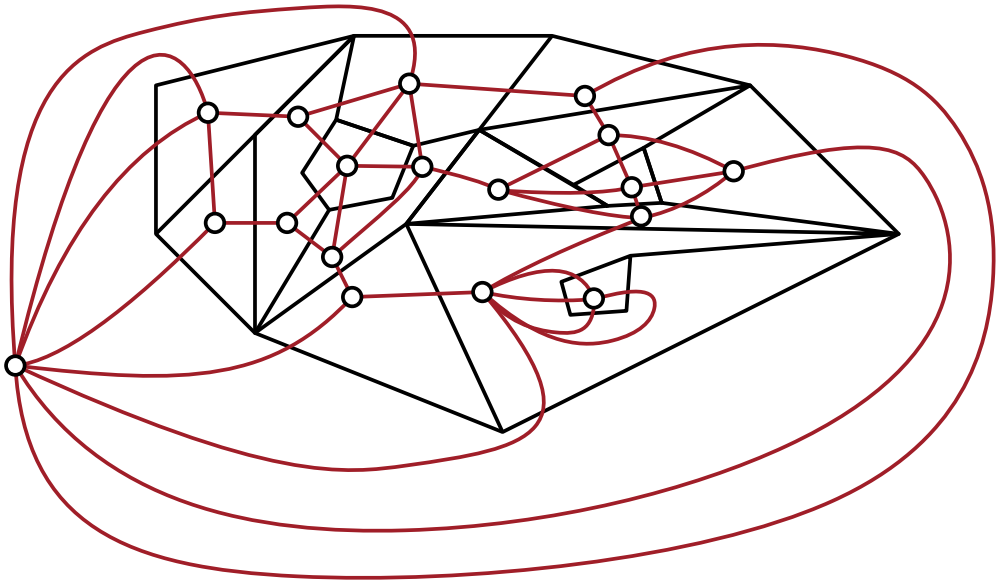
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



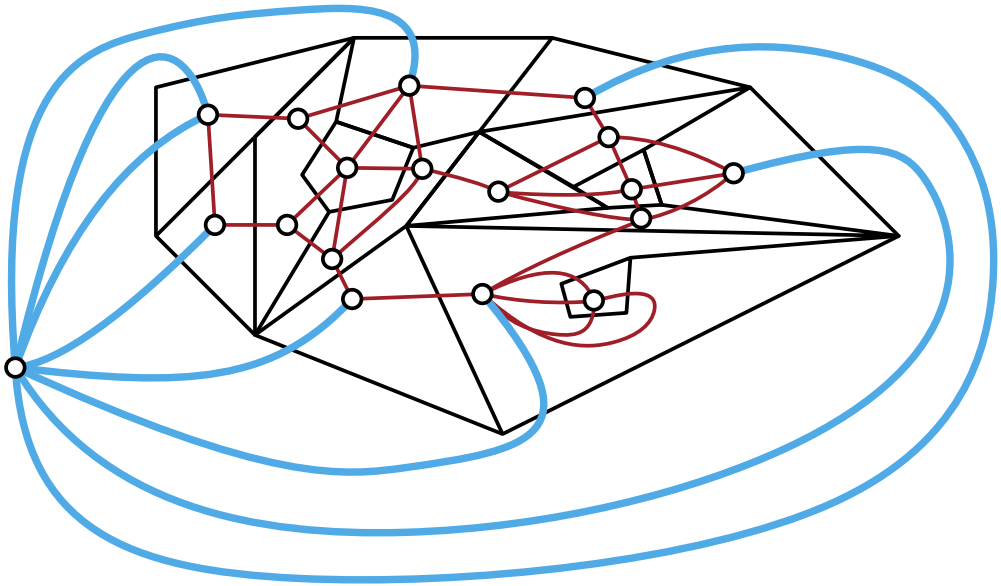
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



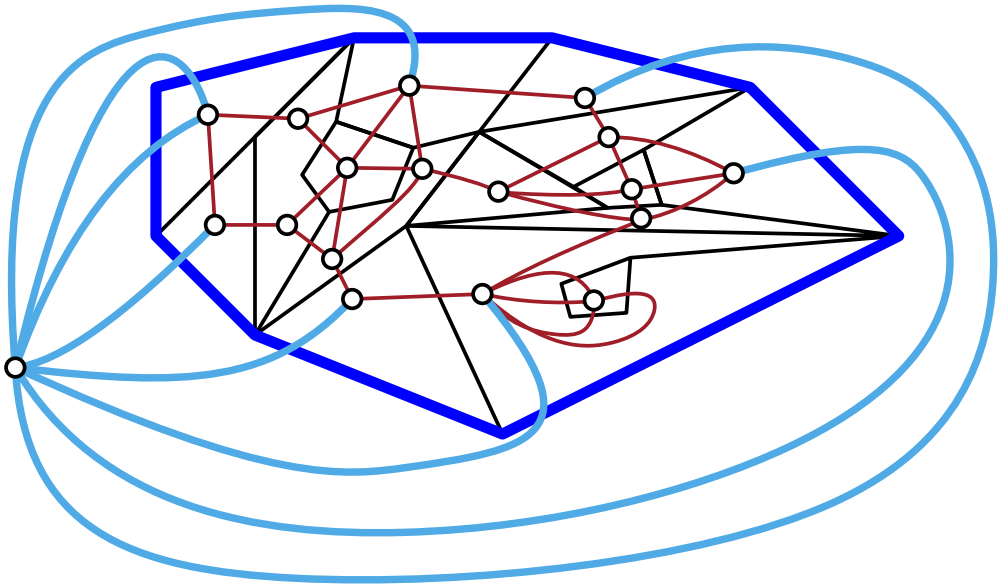
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



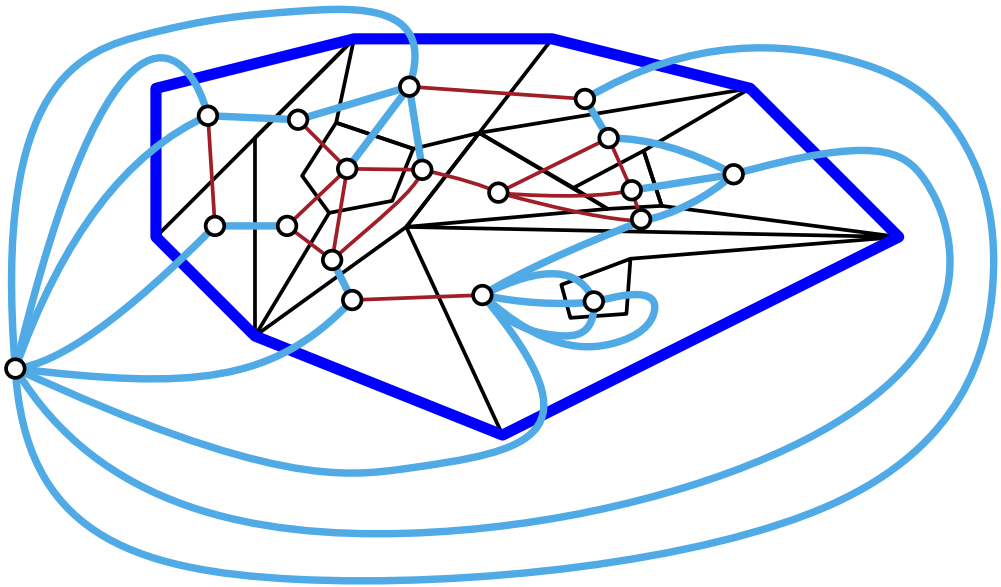
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



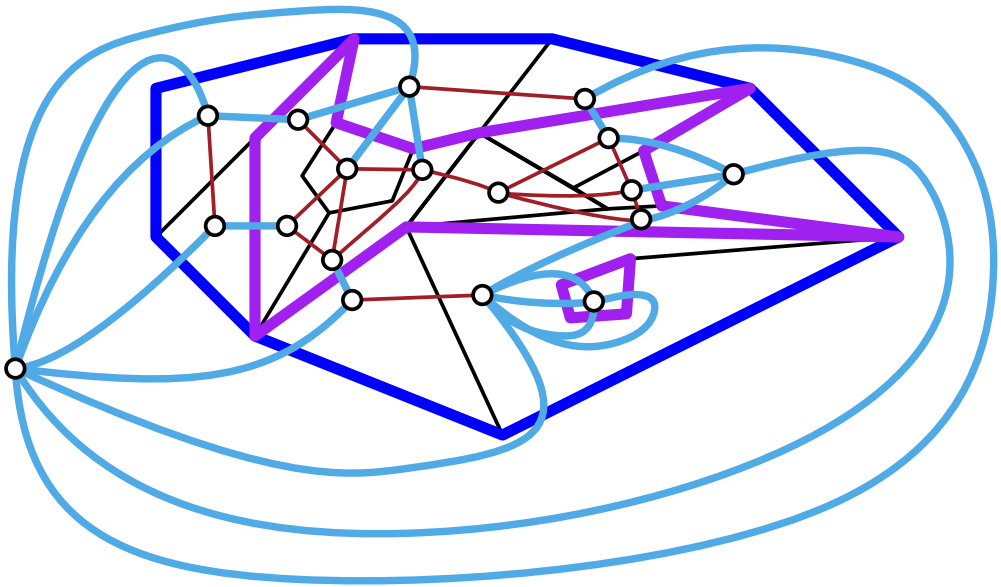
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



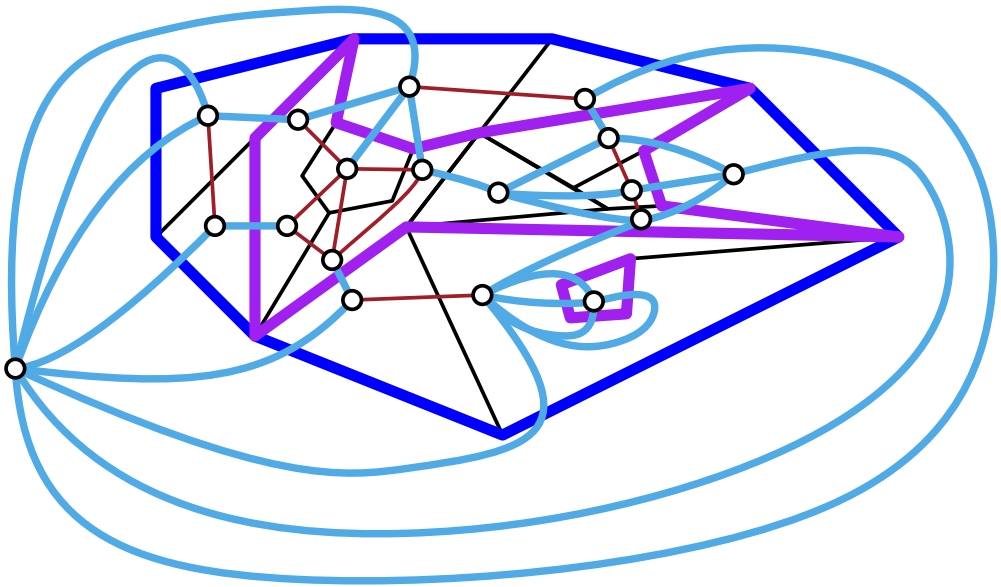
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



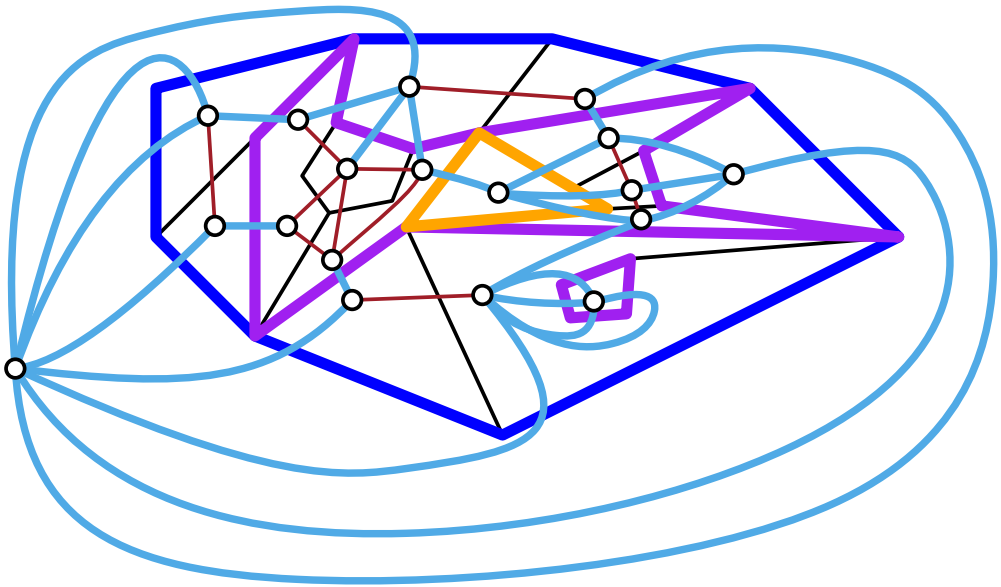
- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen



- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

1 – Entfernung von Kreisen

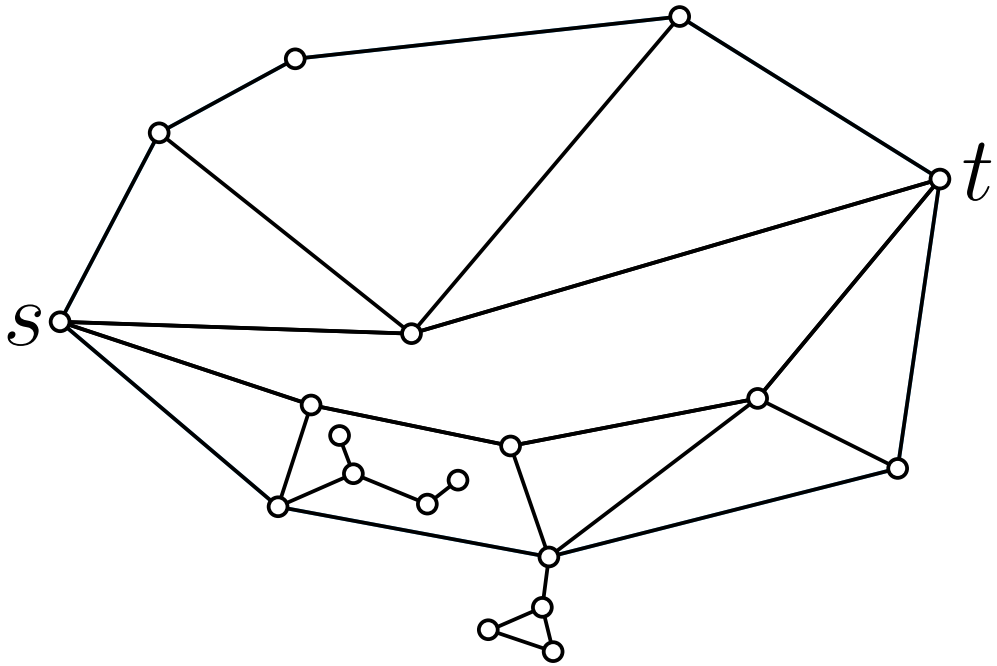


- Breitensuche ausgehend von f_0-Dualknoten.
- Level i entspricht Facetten f mit $dist(f) = i$.
- Dualkanten der Baumkanten + Nicht-Baumkanten die von Level $i - 1$ in Level i führen, bilden C_i.

2 – Spezialfall von s-t-Wegen

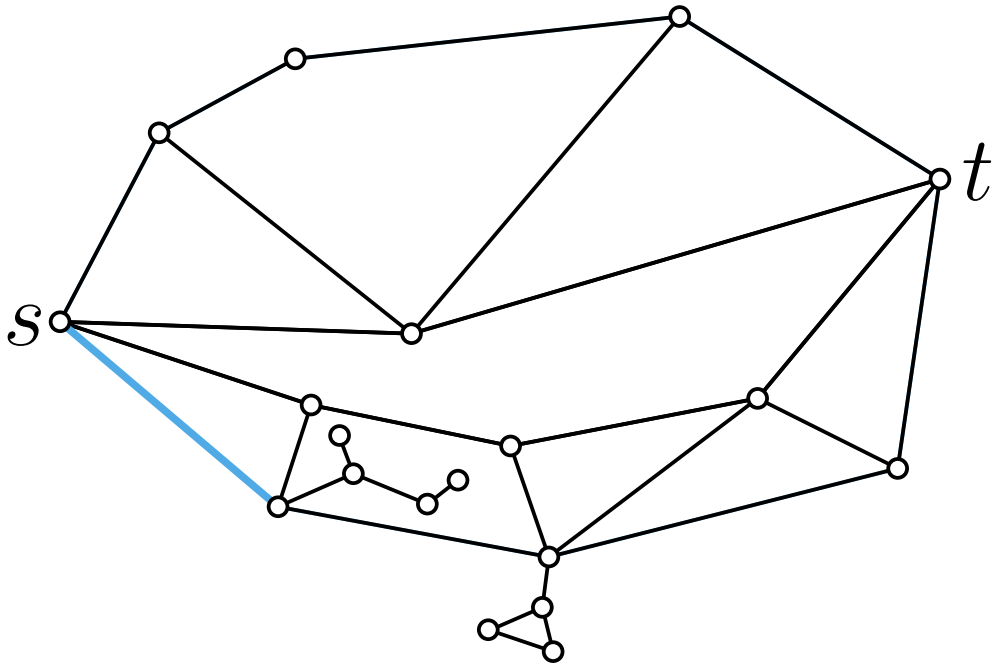
Geben Sie einen einfachen Algorithmus an, der folgendes Problem in Linearzeit löst:
Gegeben ein planarer Graph G mit fester Einbettung und ausgezeichneten Knoten s und t, die an der äußeren Facette liegen, bestimme eine maximale Anzahl von paarweise kantendisjunkten s-t-Wegen in G.

2 – Spezialfall von s-t-Wegen



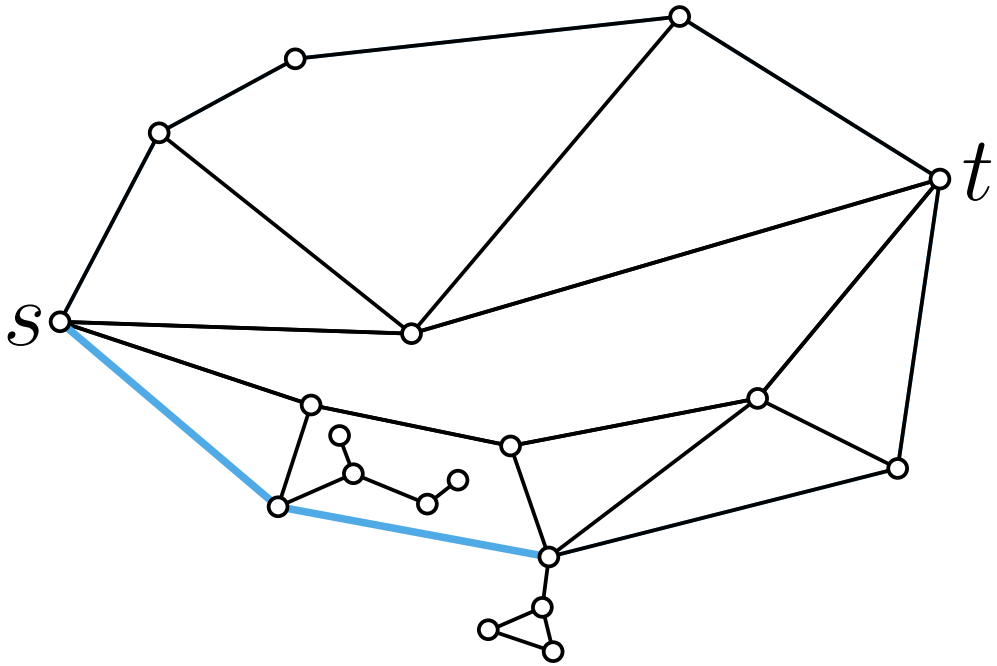
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



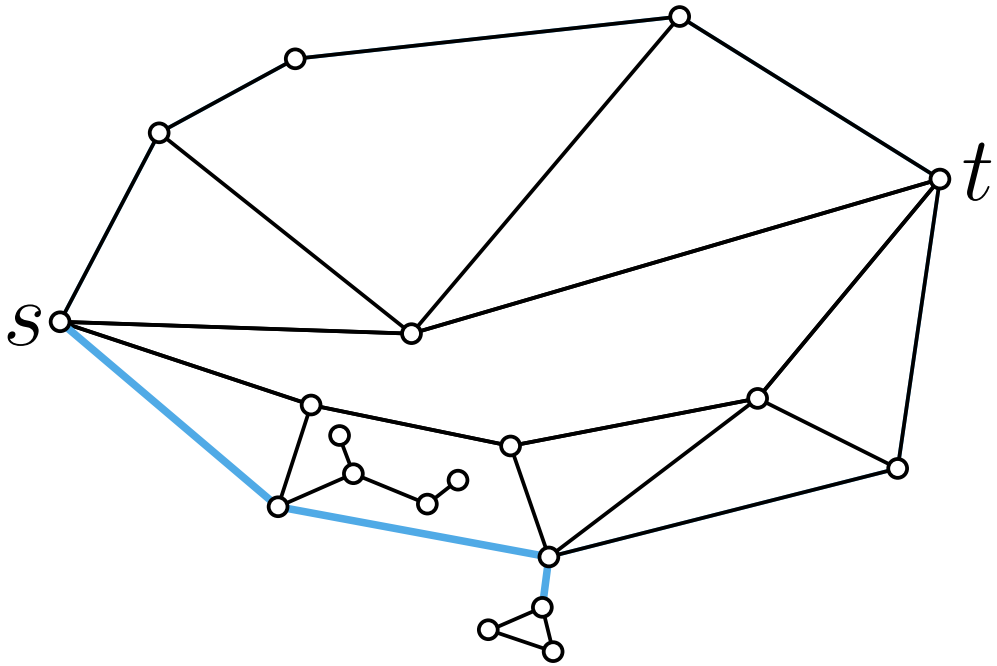
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



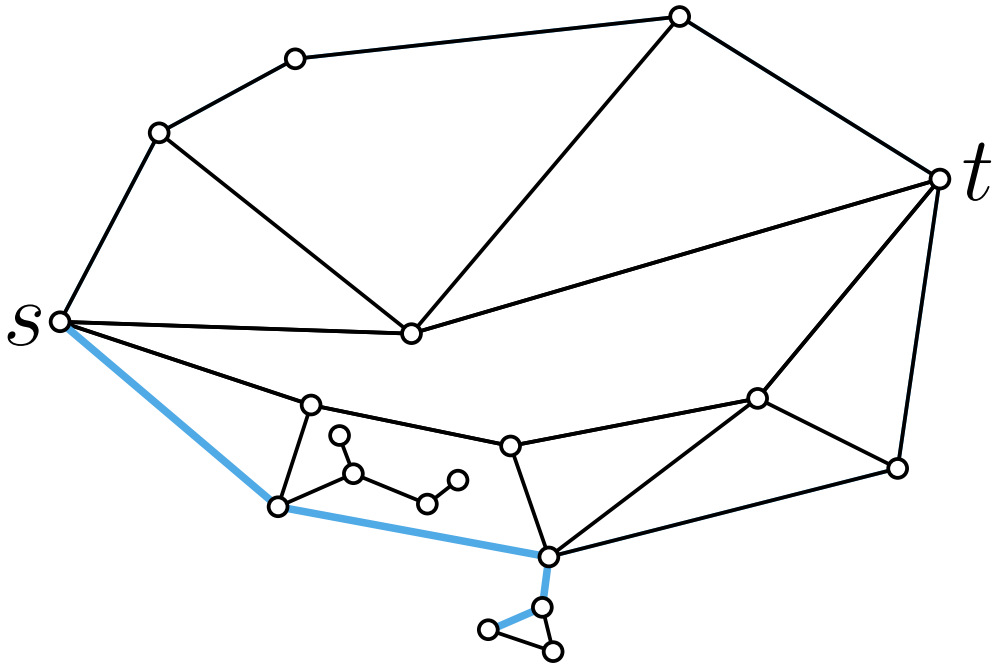
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



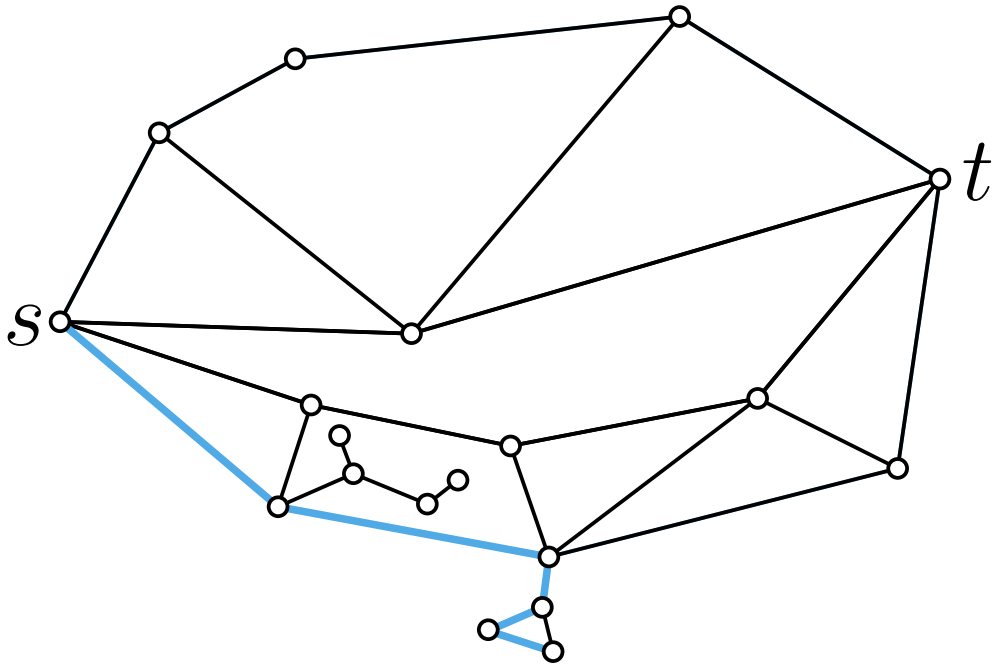
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



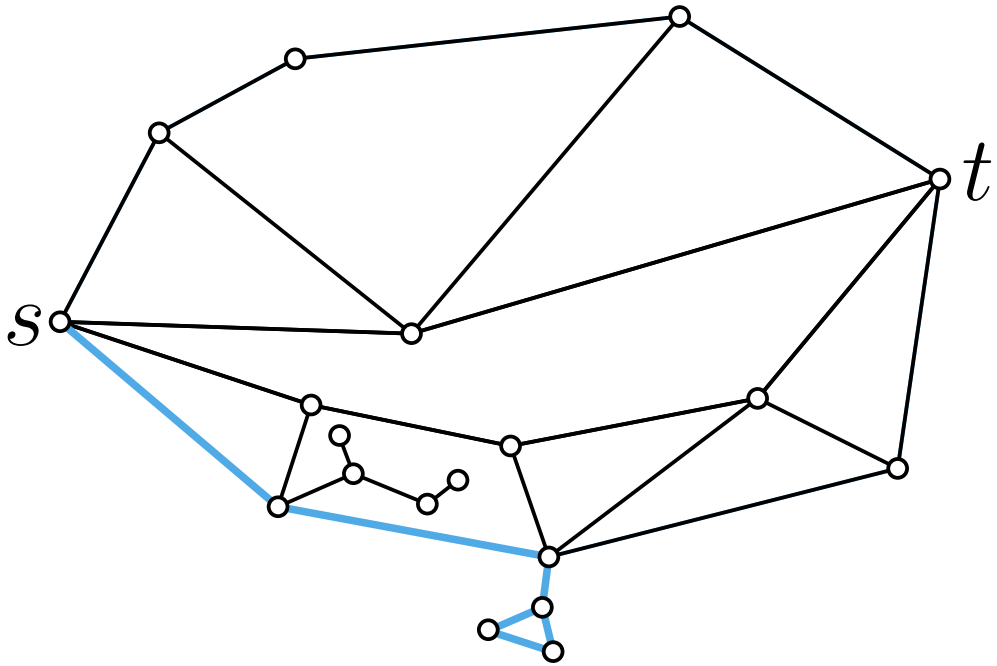
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



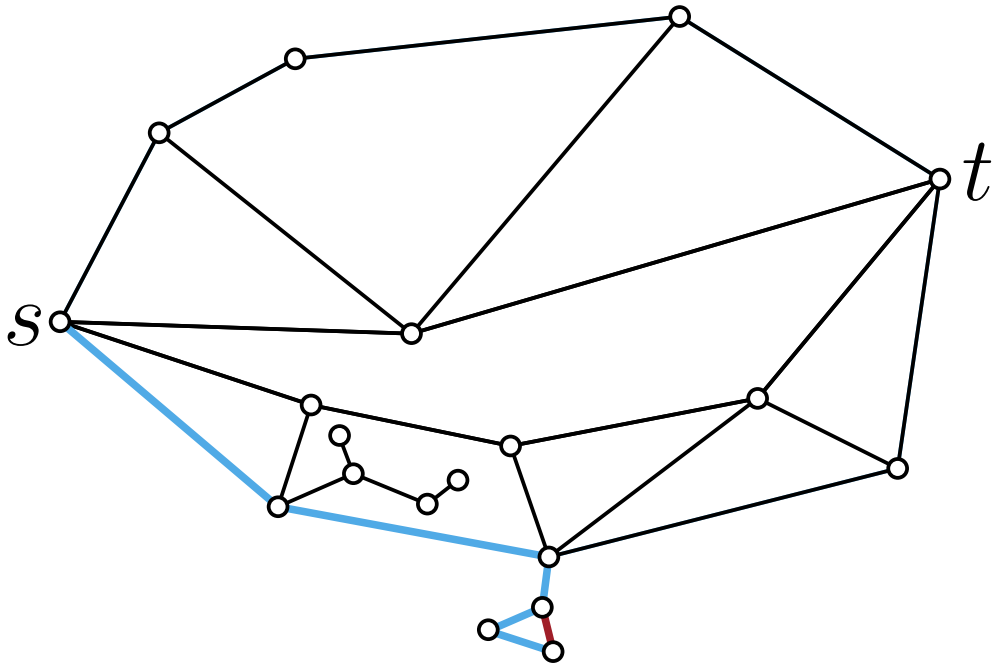
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



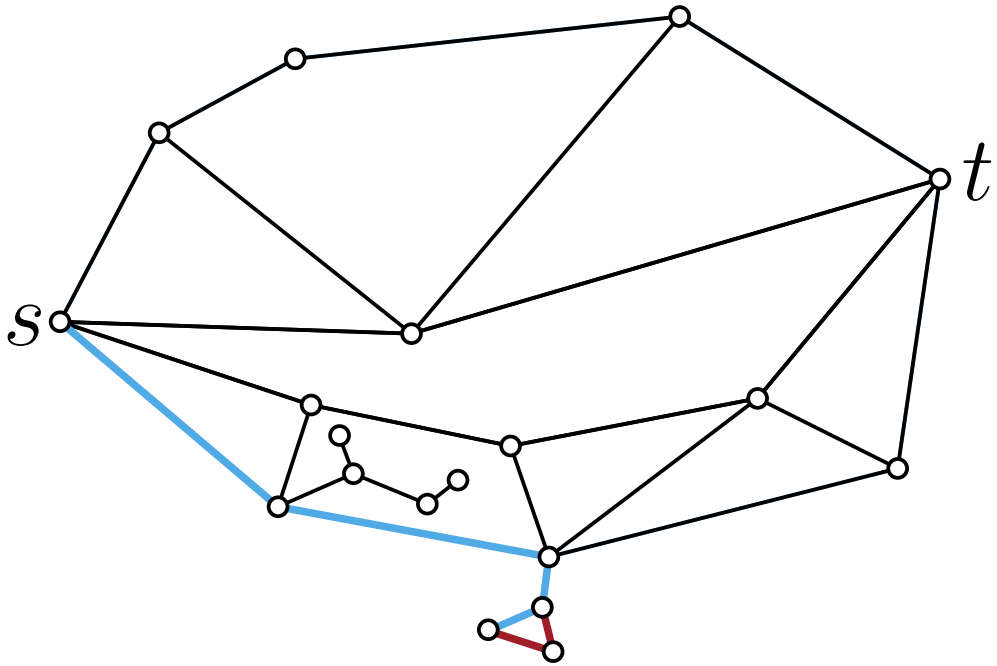
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



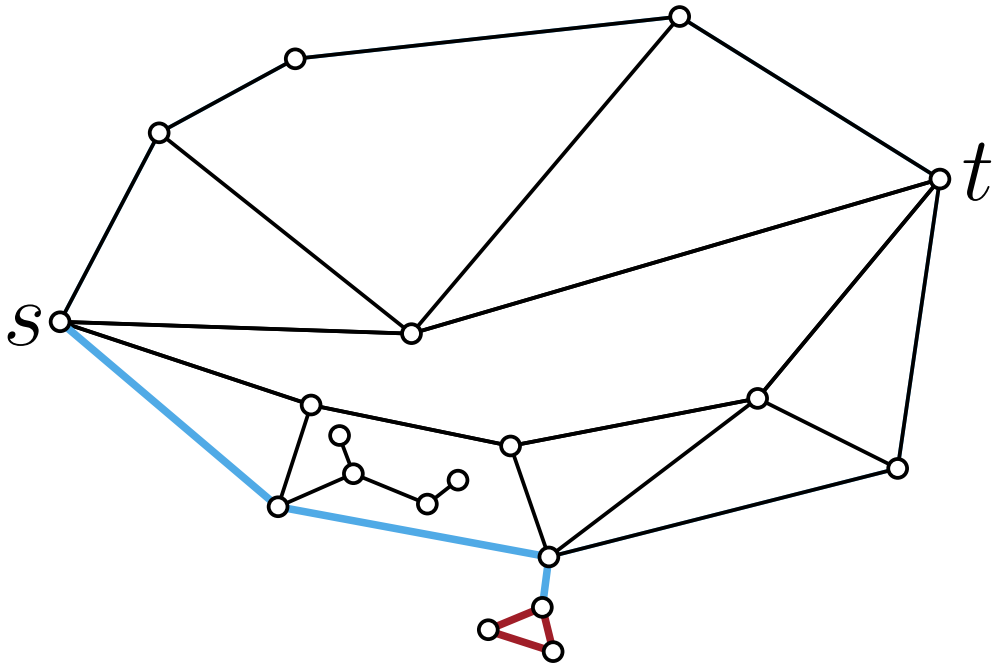
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



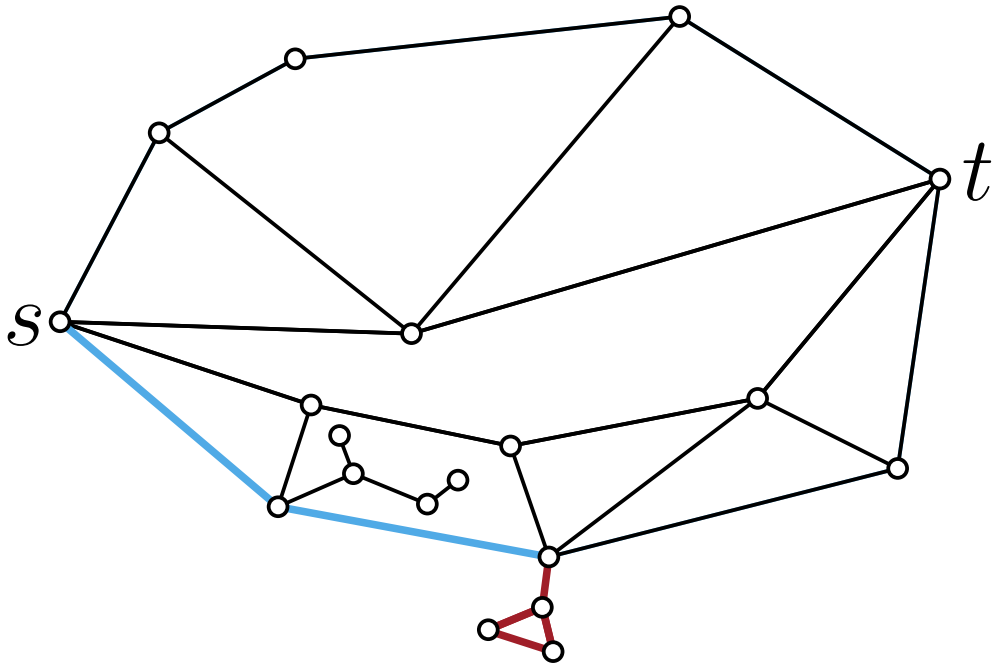
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



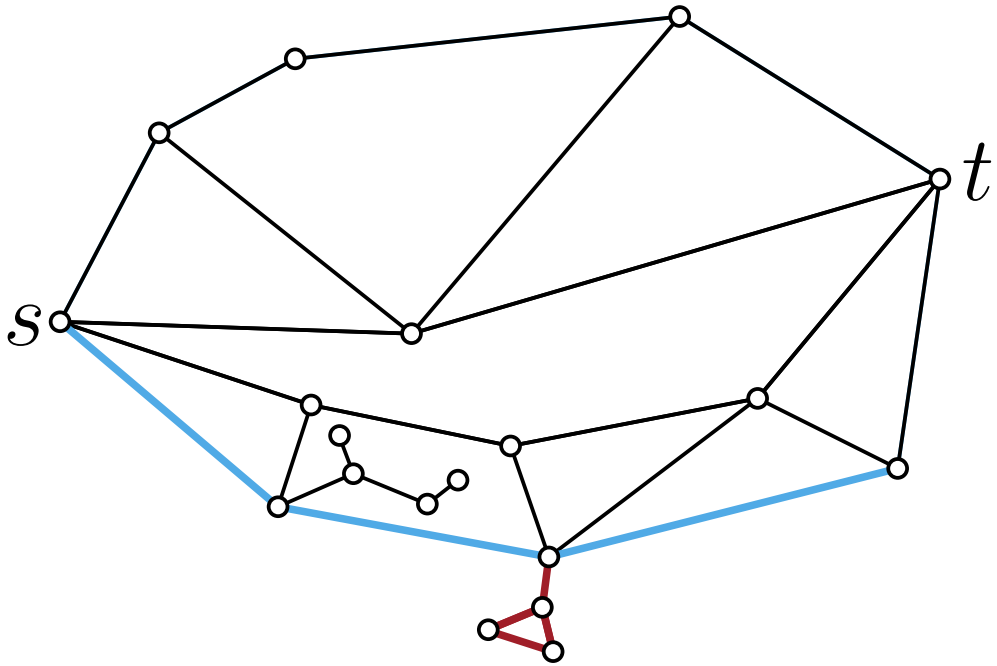
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



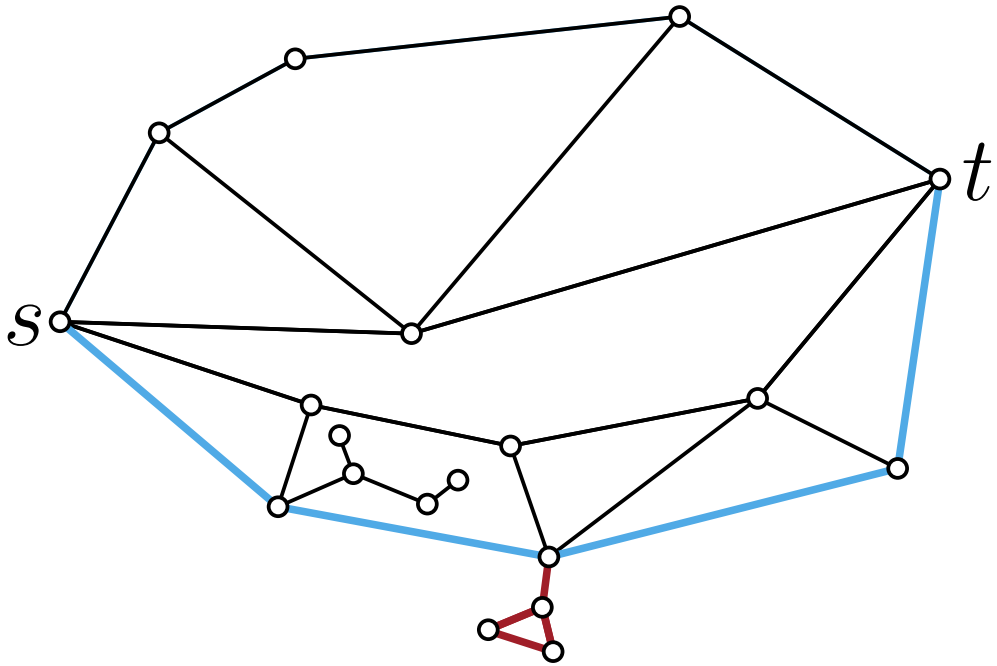
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



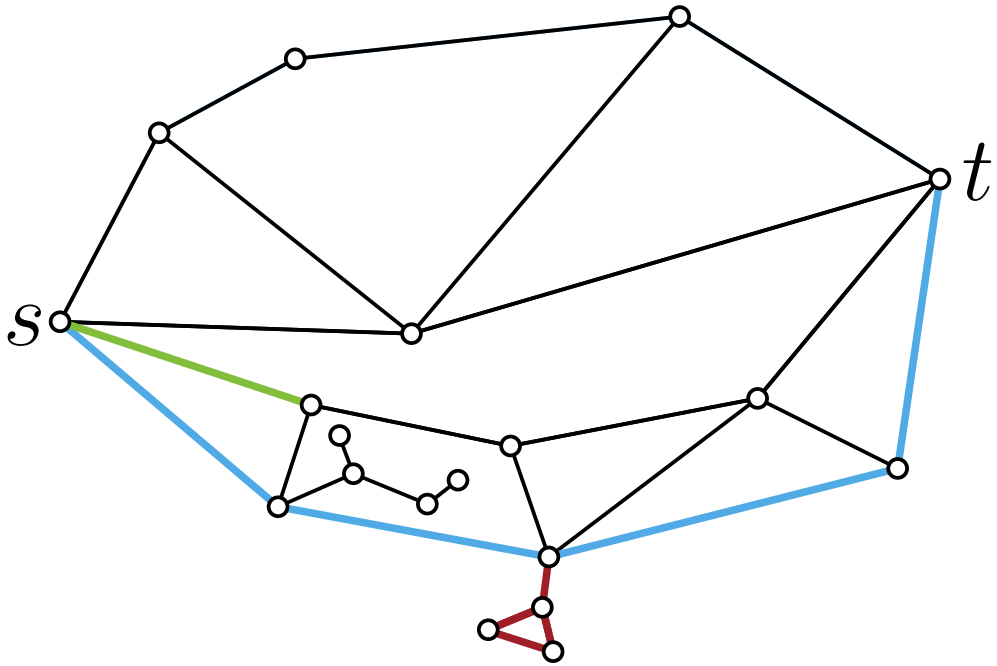
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



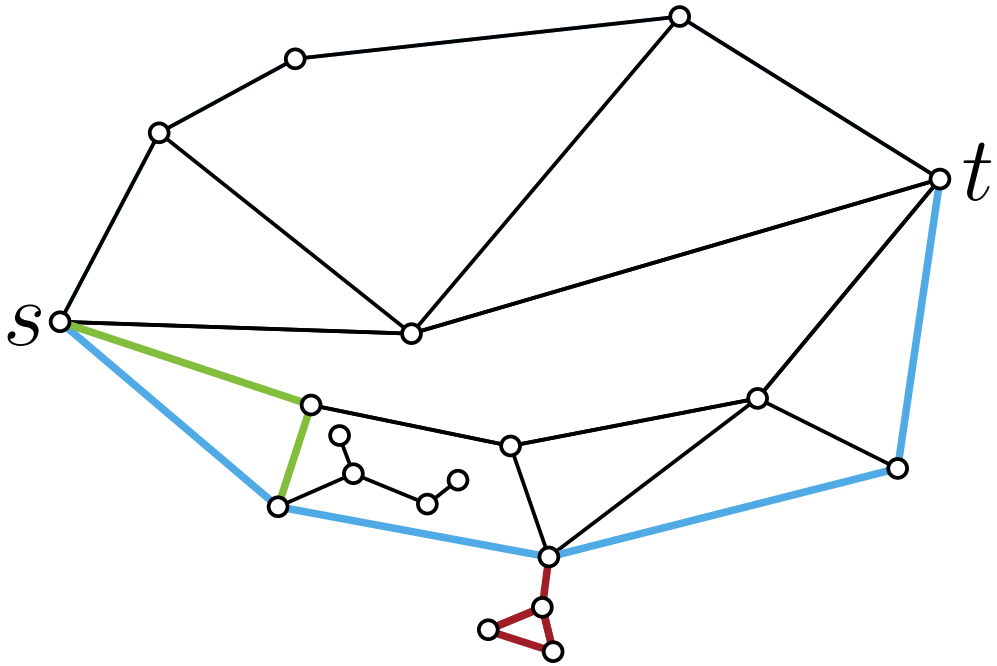
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



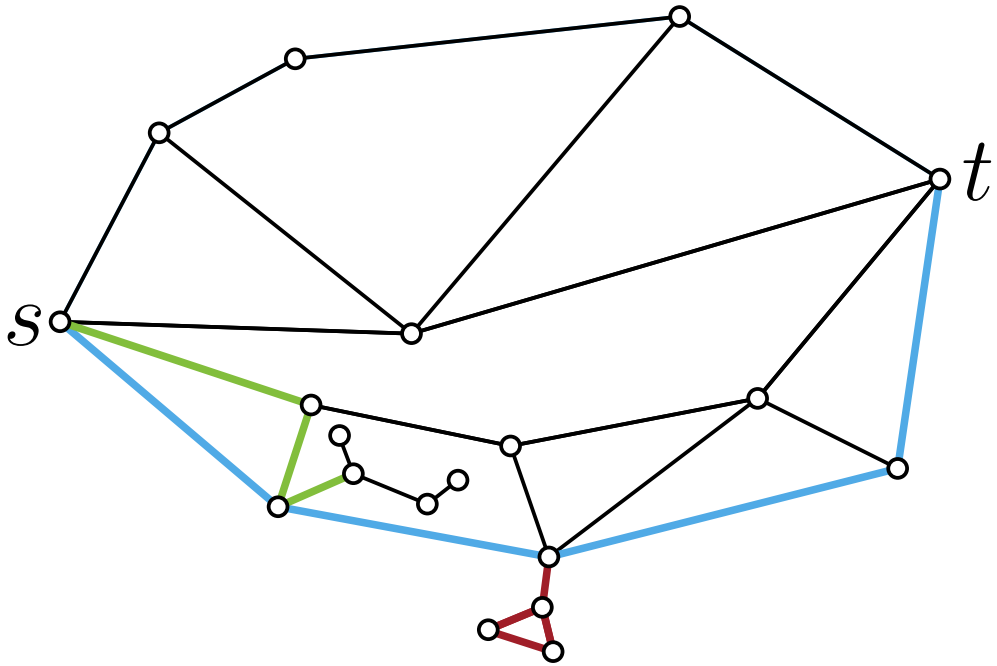
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



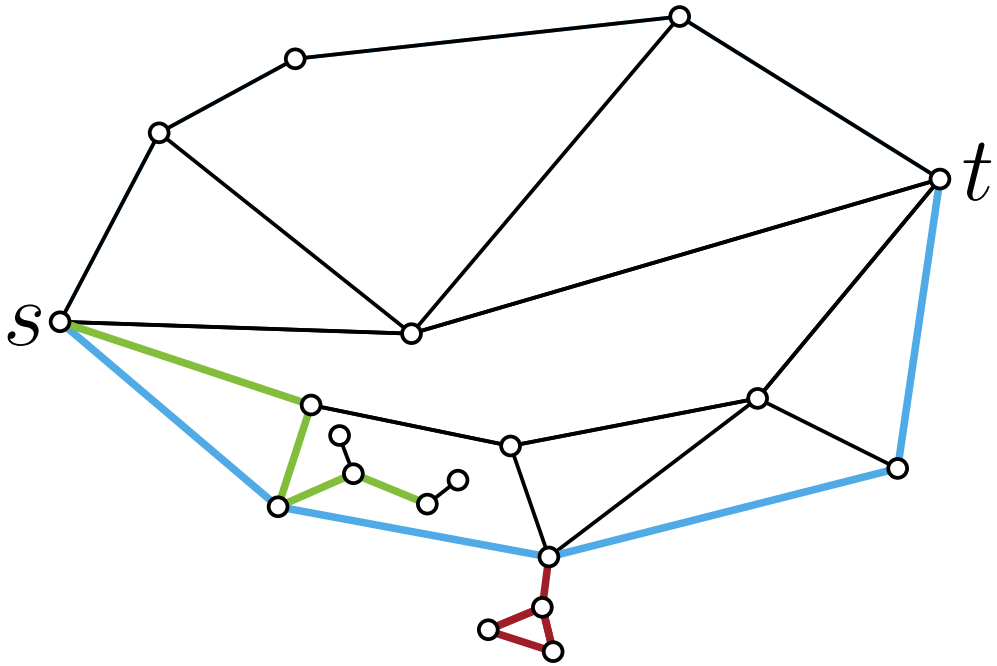
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



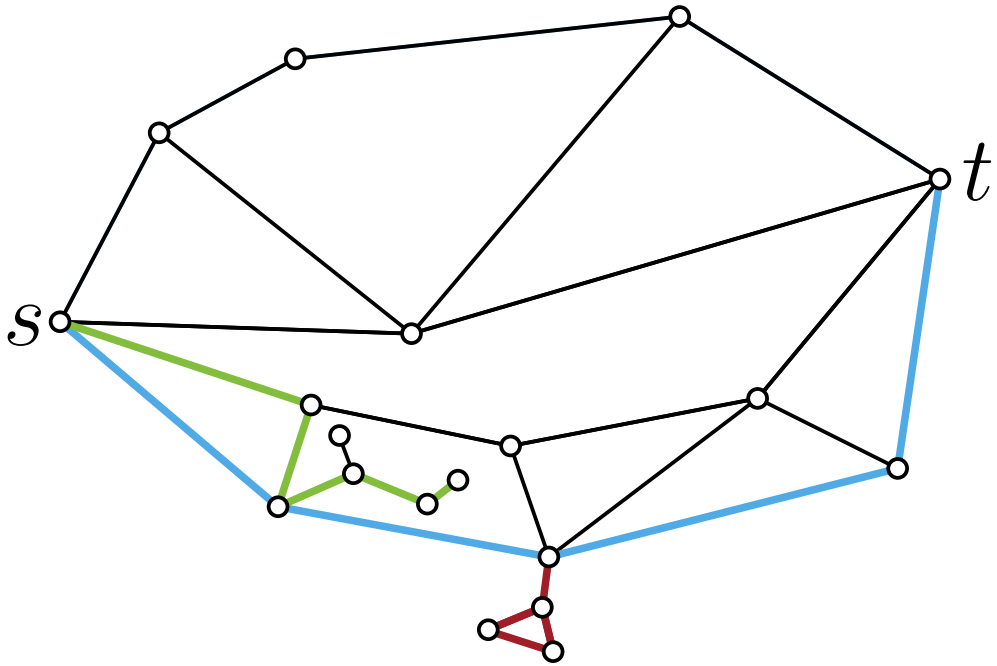
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



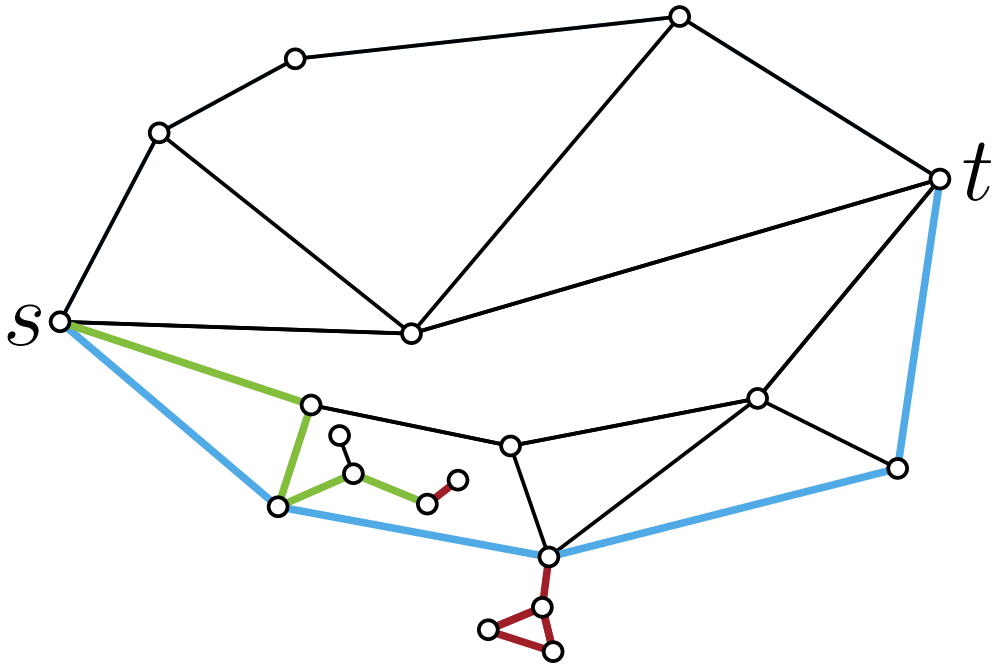
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



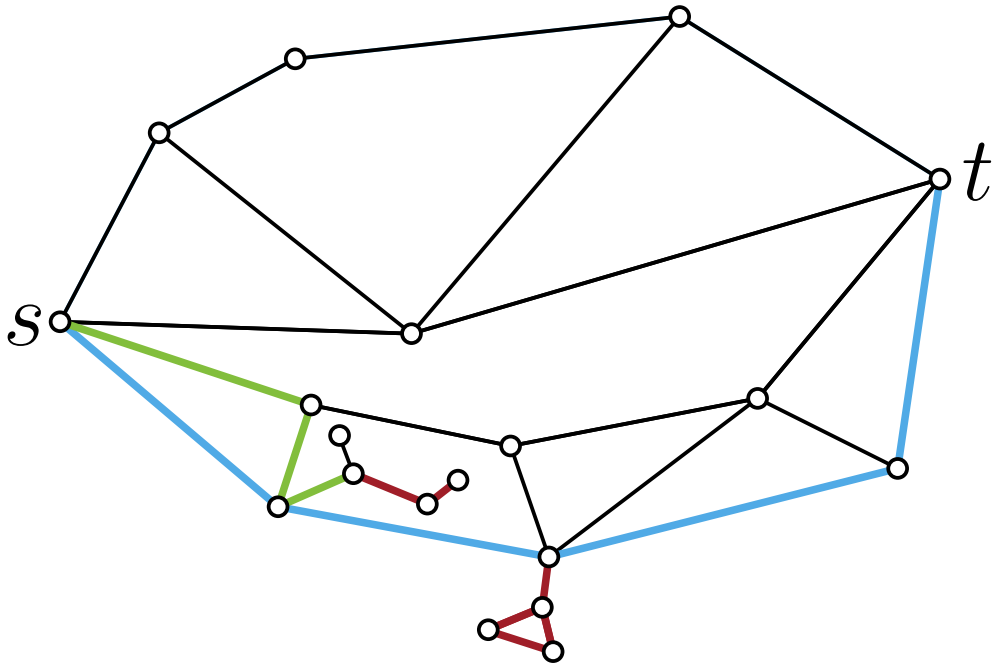
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



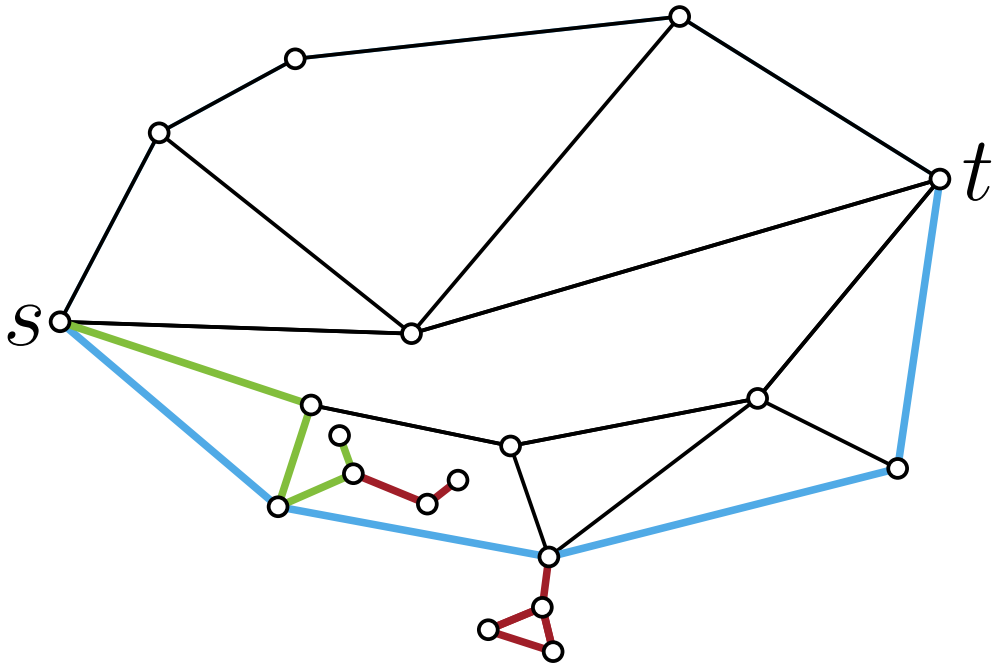
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



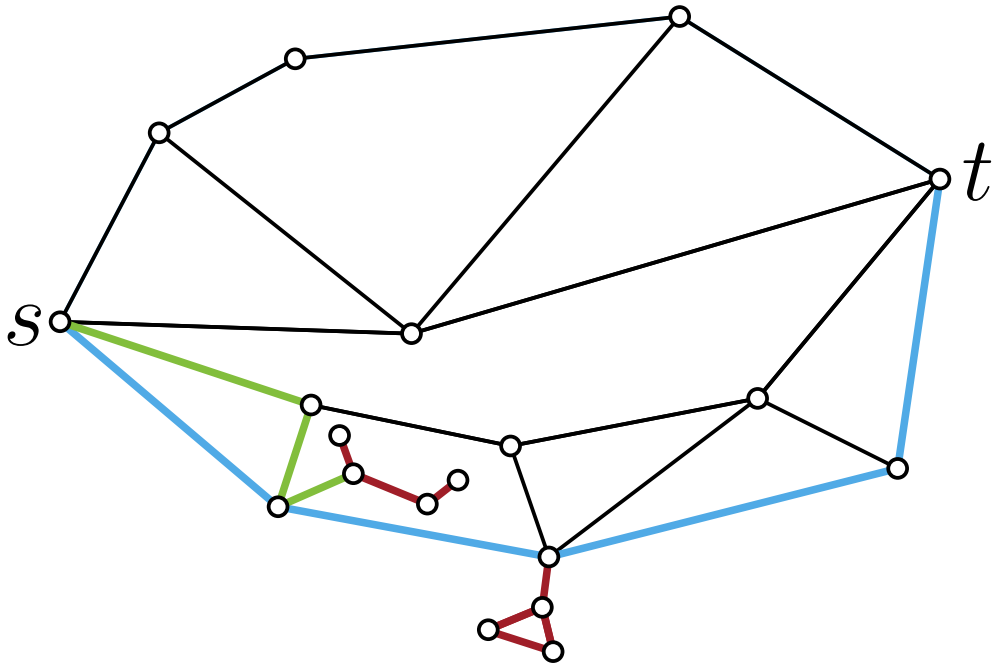
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



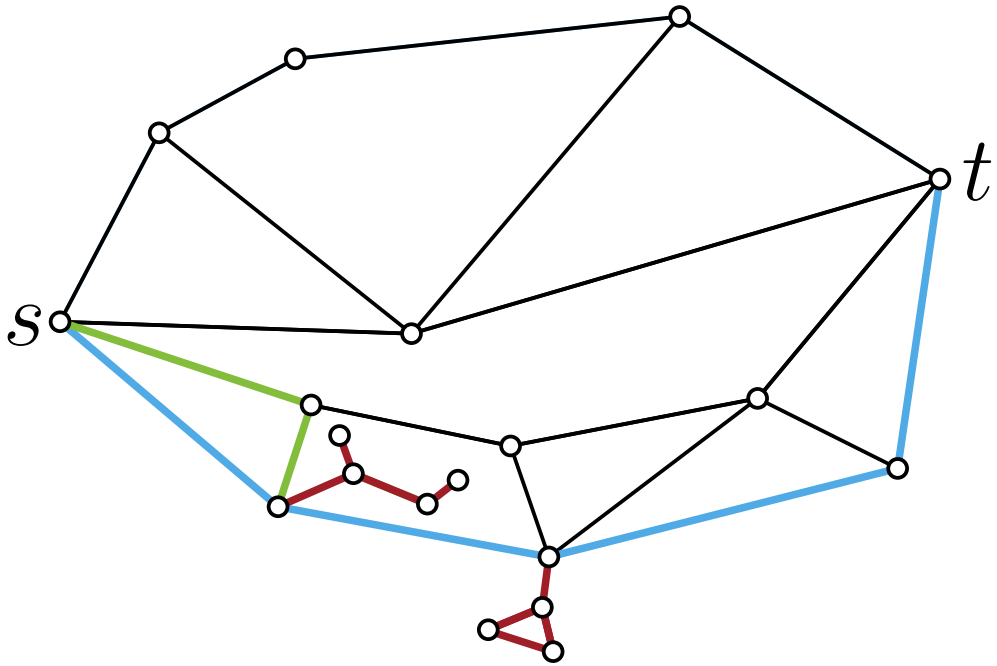
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



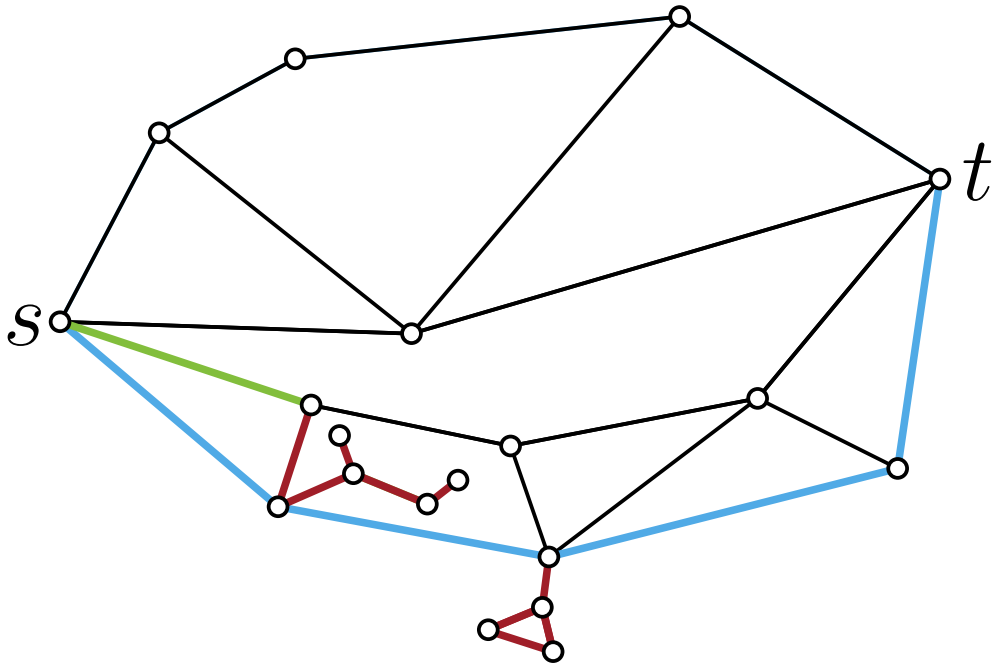
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



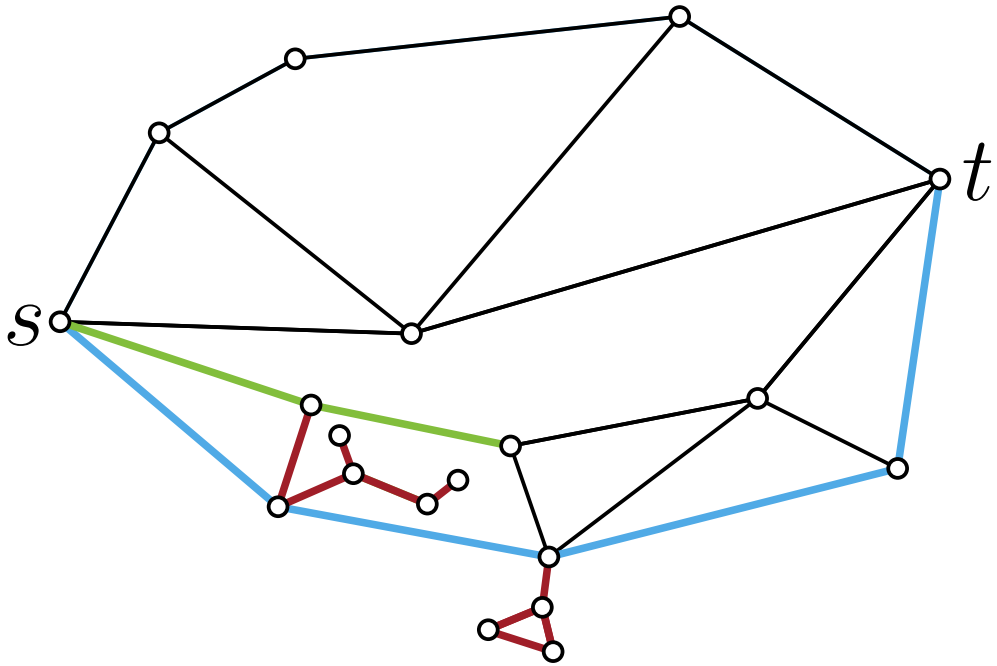
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



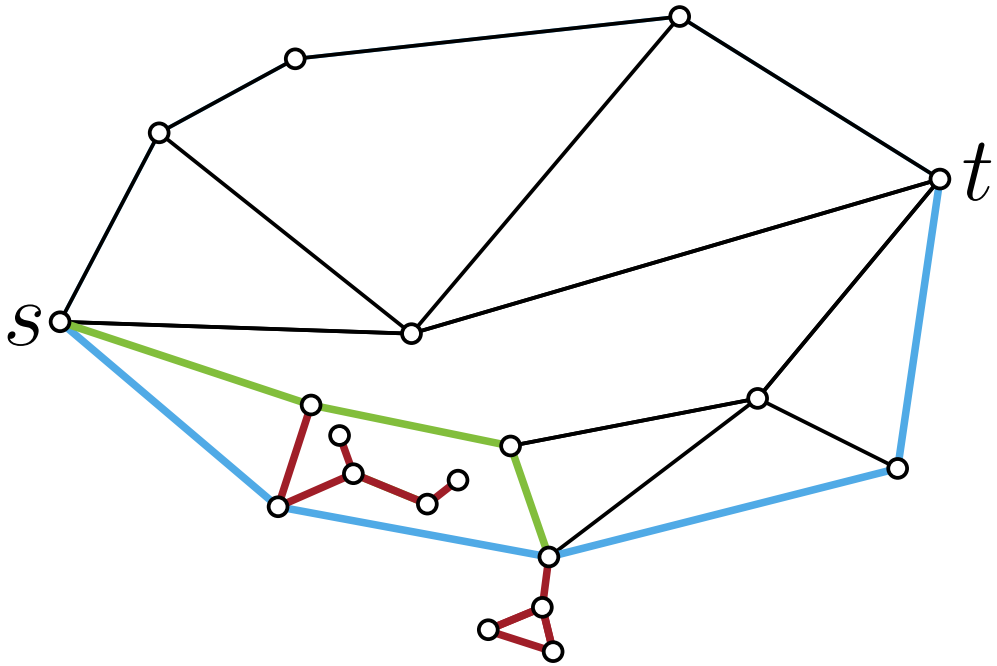
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



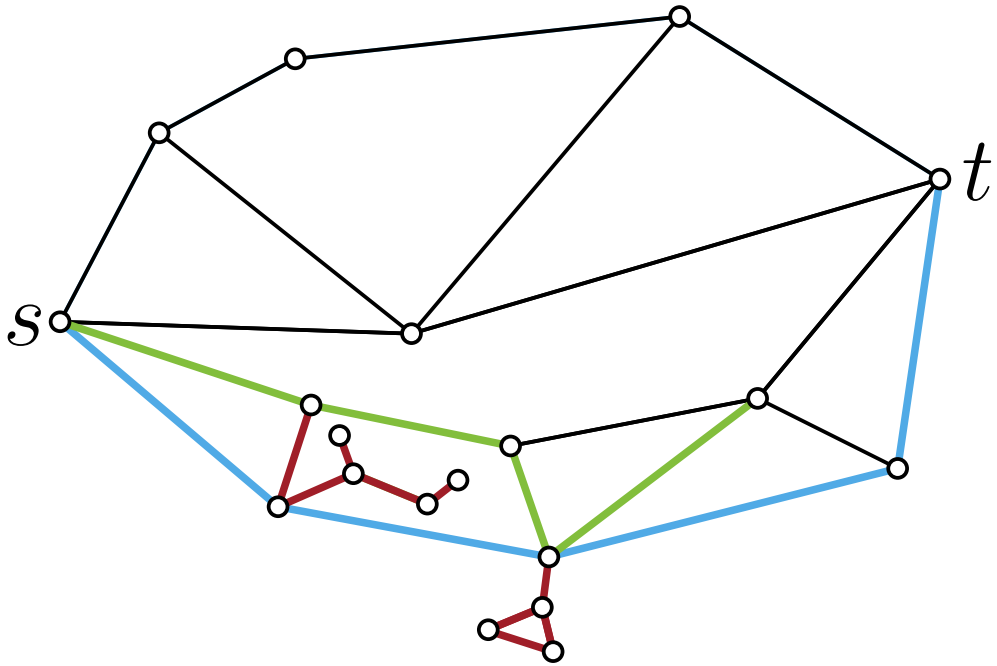
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



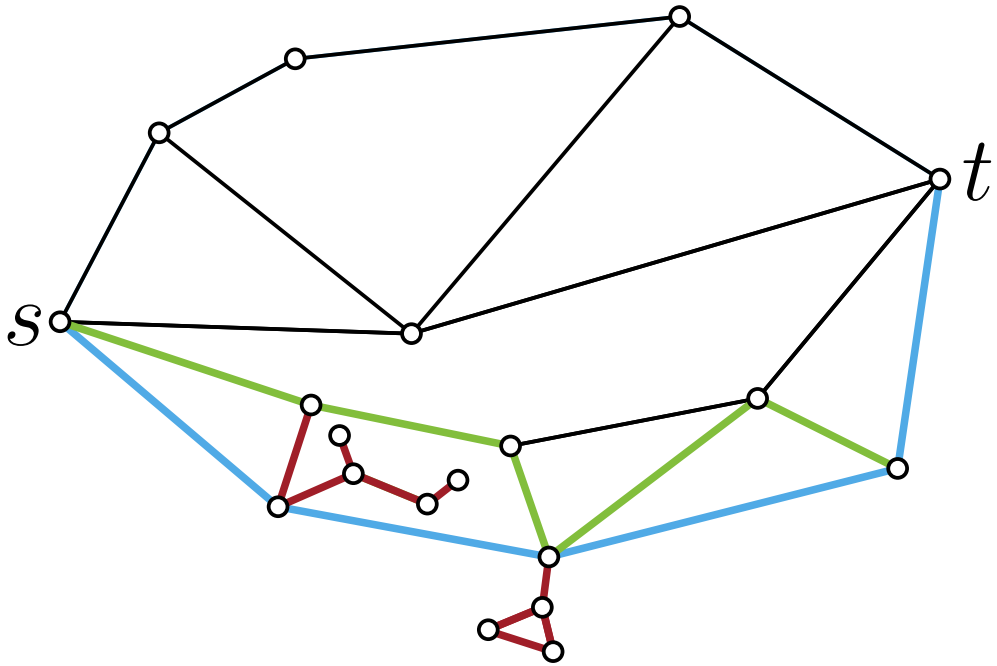
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



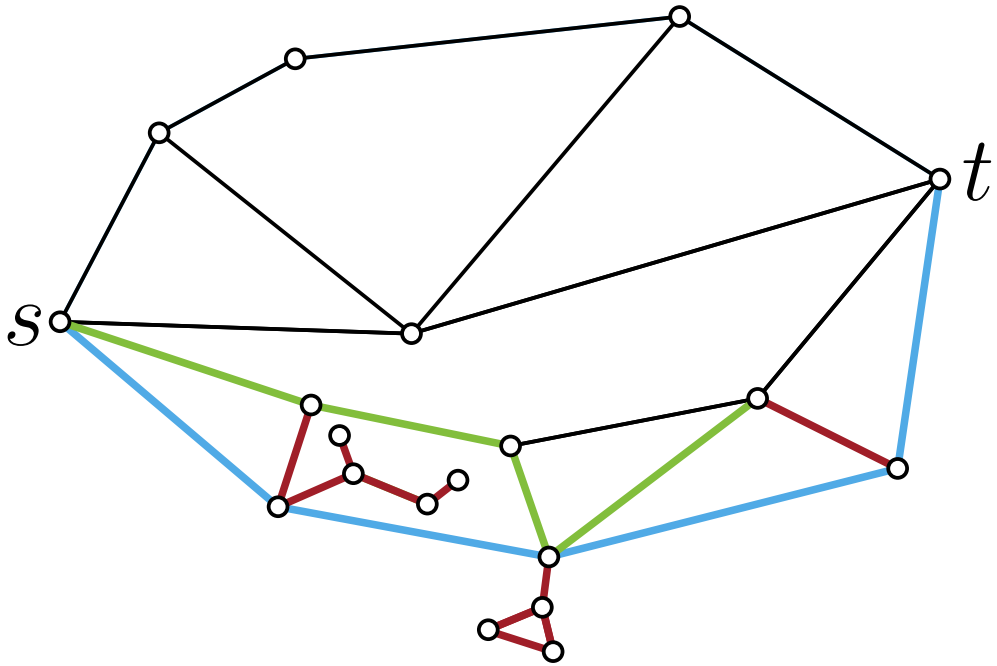
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



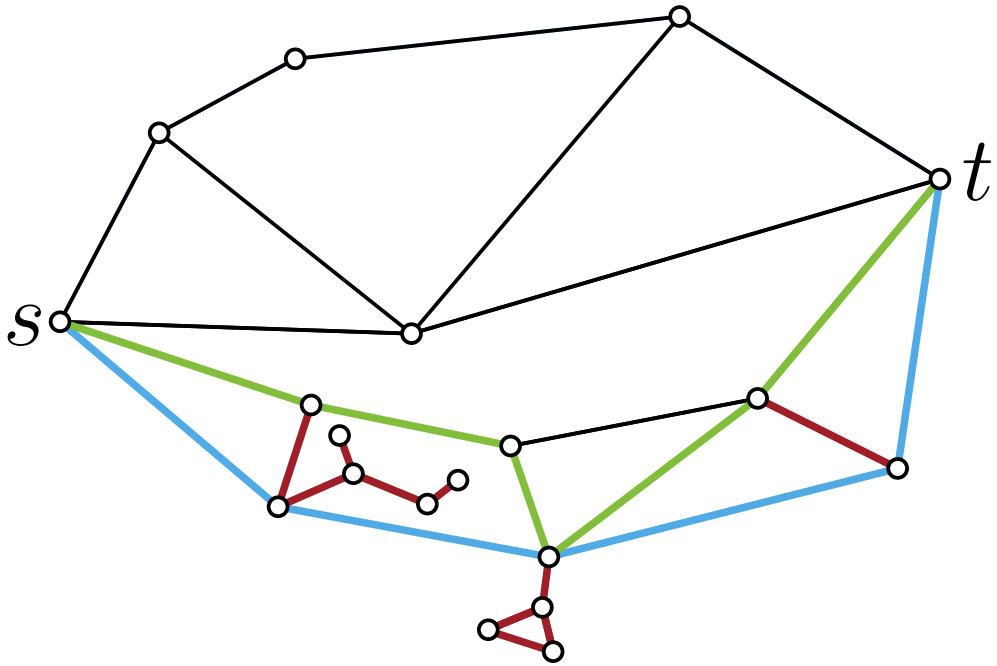
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



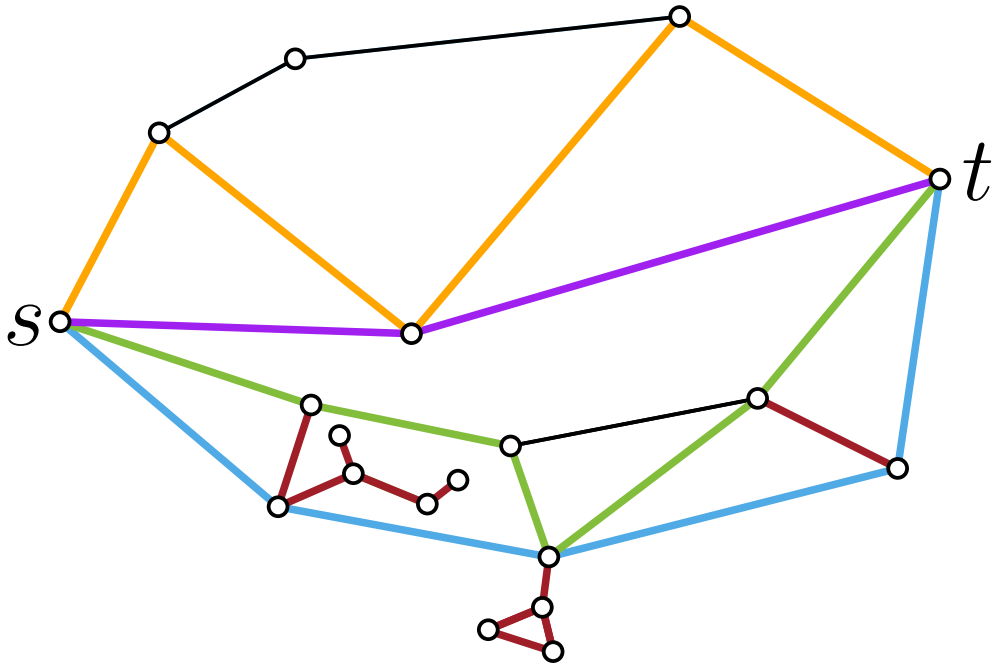
- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen



- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

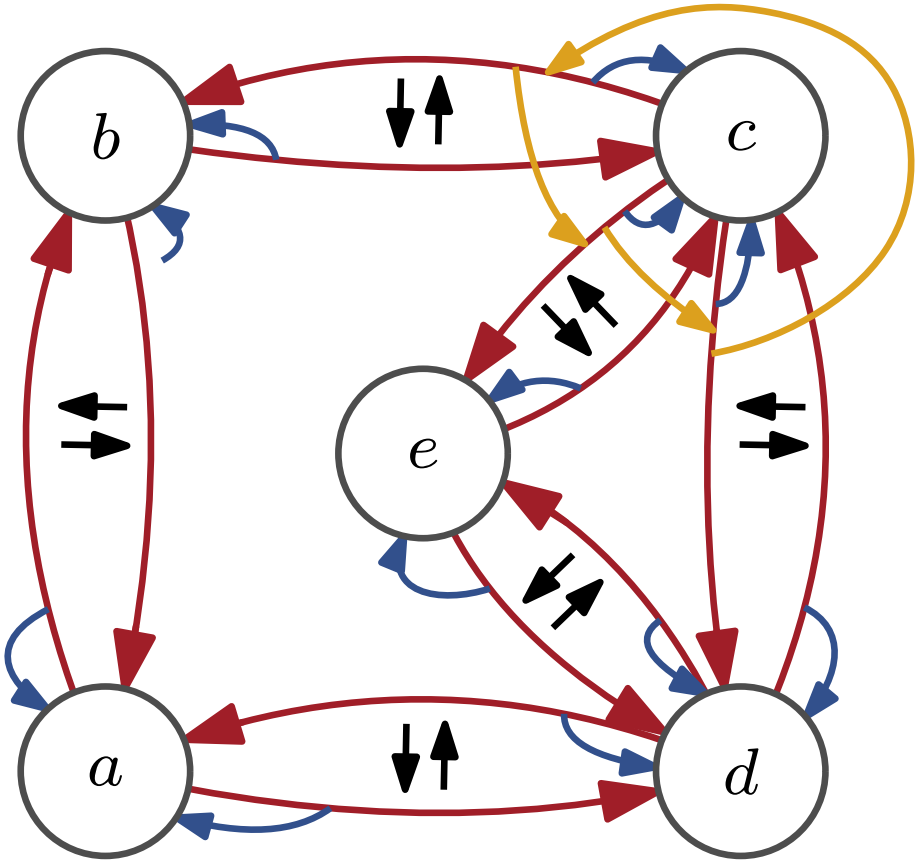
2 – Spezialfall von s-t-Wegen



- Right-First-DFS ausgehend von s.
- Besuchte Kanten aus G löschen.
- Bei Erreichen von t ist ein neuer Pfad gefunden.
- Wiederhole.

2 – Spezialfall von s-t-Wegen

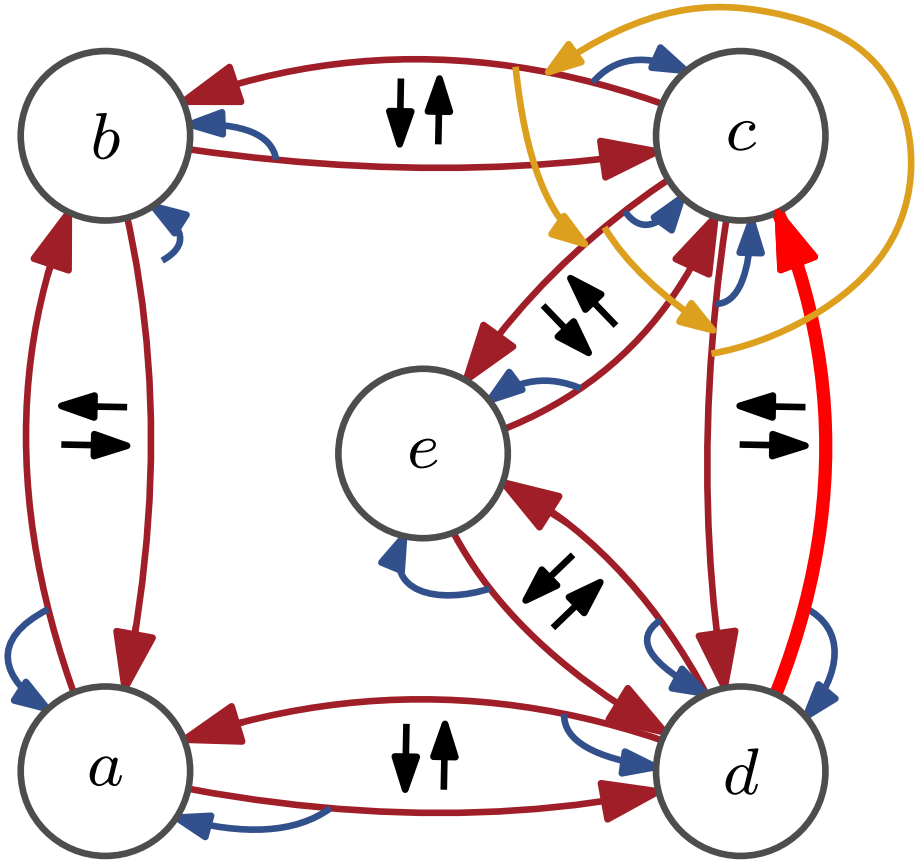
Kanten löschen



- Liste gerichteter Kanten entgegen dem Uhrzeigersinn
- $\rightsquigarrow$ Liste gerichteter Kanten im Uhrzeigersinn
- Beide Richtungen der Kante aus der doppelt verketteten Liste löschen.
- $\mathcal{O}(1)$

2 – Spezialfall von s-t-Wegen

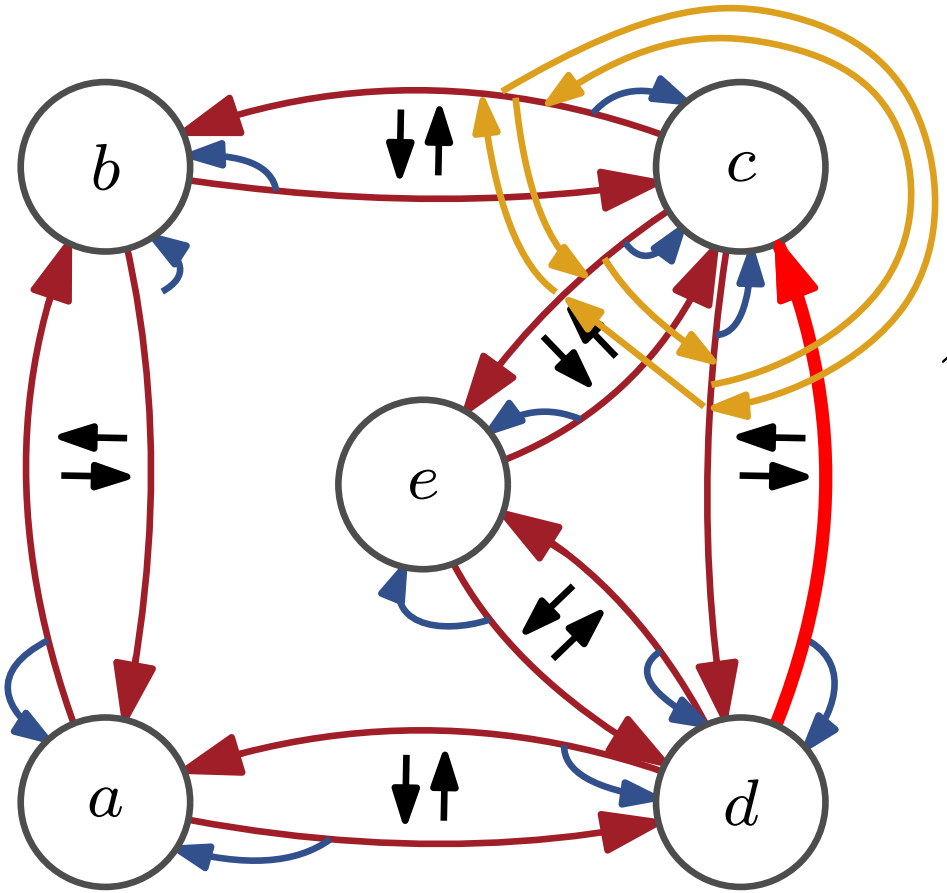
Kanten löschen



- Liste gerichteter Kanten entgegen dem Uhrzeigersinn
- $\rightsquigarrow$ Liste gerichteter Kanten im Uhrzeigersinn
- Beide Richtungen der Kante aus der doppelt verketteten Liste löschen.
- $\mathcal{O}(1)$

2 – Spezialfall von s-t-Wegen

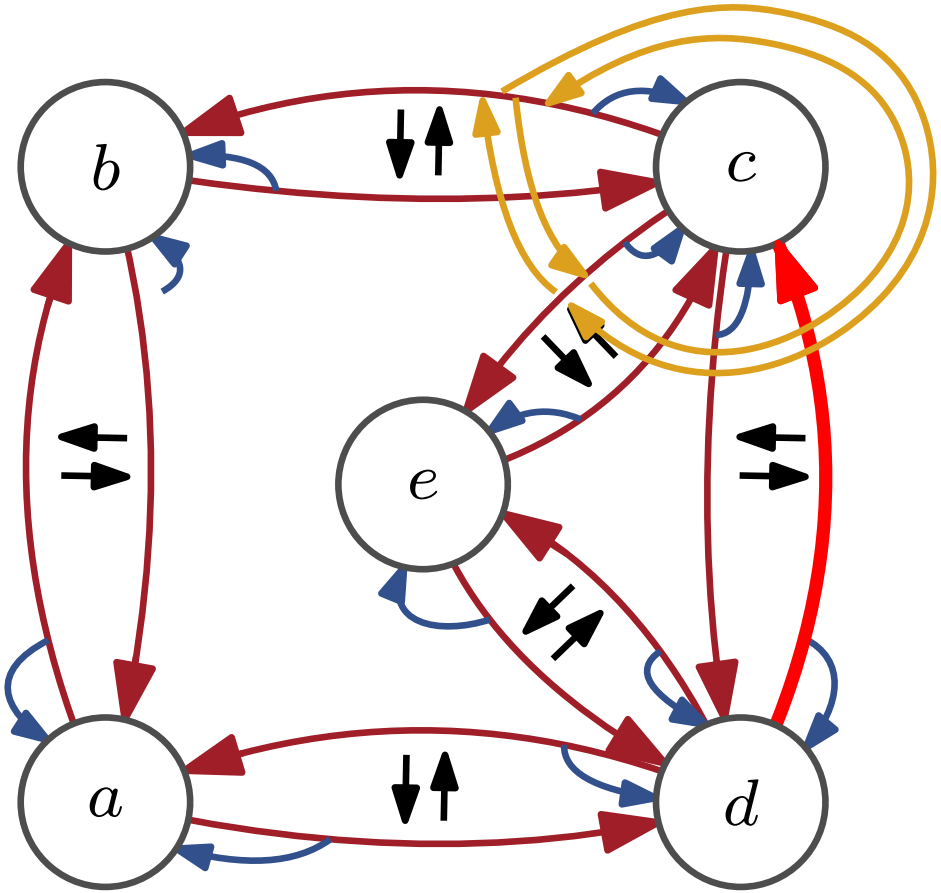
Kanten löschen



- Liste gerichteter Kanten entgegen dem Uhrzeigersinn
- $\rightsquigarrow$ Liste gerichteter Kanten im Uhrzeigersinn
- Beide Richtungen der Kante aus der doppelt verketteten Liste löschen.
- $\mathcal{O}(1)$

2 – Spezialfall von s-t-Wegen

Kanten löschen



- Liste gerichteter Kanten entgegen dem Uhrzeigersinn
- $\rightsquigarrow$ Liste gerichteter Kanten im Uhrzeigersinn
- Beide Richtungen der Kante aus der doppelt verketteten Liste löschen.
- $\mathcal{O}(1)$

3 – Dreiecke zählen

- Sei G ein Graph mit n Knoten und m Kanten.
- Der k-Core von G ist der maximale Subgraph in dem jeder Knoten mindestens Grad k hat.
- Die Core-Zerlegung weist jedem Knoten das maximale k zu für welches er im k-Core liegt.
- Die Entartetheit $D(G)$ ist das größte k für welches G einen nicht-leeren k-Core hat.
- $\mathcal{N}(v)$: Nachbarschaft von v in G.

3.1 – k-Core

- Geben Sie einen Algorithmus an, der die Core-Zerlegung in $\mathcal{O}(n+m)$ berechnet.

3.2 – k-Core

Algorithm CORE DECOMPOSITION

$\delta(v) \leftarrow$ Grad von Knoten v

$\Delta \leftarrow max\{\delta(v) \mid v \in V\}$

Sortiere die Knoten nach $\delta(v)$ in *Buckets* $b_0, \ldots, b_\Delta$

for $b_k \in (b_0, \ldots, b_\Delta)$ **do**

  while $b_k \neq \emptyset$ **do**

    $v \leftarrow b_k.pop()$

    $\text{CORE}[v] \leftarrow k$

    for $u \in N(v)$ **do**

      $b_j \leftarrow$ Bucket in dem u liegt

      if $j > k$ **then**

        Verschiebe v von b_j nach b_{j-1}

$D(G) \leftarrow max\{\text{CORE}[v] \mid v \in V\}$

3.2 – Dreiecke zählen

Aufgabe

Geben Sie einen Algorithmus an der die Anzahl Dreicke in G in $\mathcal{O}((n+m) \cdot D(G))$ berechnet.
Hinweis: Modifizieren Sie den Algorithmus von Übungsblatt 4.

- Dreiecke im gesamten Graphen, nicht pro Knoten
- $\mathcal{N}(v)$ Nachbarschaft von v
- $\mathcal{N}^+(v)$ Nachbarn von v über orientierte Kanten (v, u)
- $\widetilde{\mathcal{N}}(v)$ Nachbarschaft von v über noch nicht orientierte Kanten

3.2 – Dreiecke zählen

Aufgabe

Geben Sie einen Algorithmus an der die Anzahl Dreicke in G in $\mathcal{O}((n+m) \cdot D(G))$ berechnet.
Hinweis: Modifizieren Sie den Algorithmus von Übungsblatt 4.

- Dreiecke im gesamten Graphen, nicht pro Knoten
- $\mathcal{N}(v)$ Nachbarschaft von v
- $\mathcal{N}^+(v)$ Nachbarn von v über orientierte Kanten (v, u)
- $\widetilde{\mathcal{N}}(v)$ Nachbarschaft von v über noch nicht orientierte Kanten

3.2 – Dreiecke zählen

Algorithm KANTEN-ORIENTIERUNG

Sei $v_1, \ldots, v_n$ die Knotenreihenfolge in der sie bei der Core-Zerlegung gelöscht wurden

for $\{v_i, v_j\} \in E$ **do**

  if $j > i$ **then**

    Orientiere $\{v_i, v_j\} \to (v_i, v_j)$

  else

    Orientiere $\{v_i, v_j\} \to (v_j, v_i)$

Achtung: Ergänzung

Die obige, korrekte Version stimmt nicht mit der in der Übung vorgestellten Variante überein, da man innerhalb eines k-Cores nicht beliebig orientieren kann (Beispiel Stern).

$\Rightarrow \forall v \in V : |\mathcal{N}^+(v)| \leq D(G)$

3.2 – Dreiecke zählen

Algorithm KANTEN-ORIENTIERUNG

Sei $v_1, \ldots, v_n$ die Knotenreihenfolge in der sie bei der Core-Zerlegung gelöscht wurden

for $\{v_i, v_j\} \in E$ **do**

  if $j > i$ **then**

    Orientiere $\{v_i, v_j\} \to (v_i, v_j)$

  else

    Orientiere $\{v_i, v_j\} \to (v_j, v_i)$

Achtung: Ergänzung

Die obige, korrekte Version stimmt nicht mit der in der Übung vorgestellten Variante überein, da man innerhalb eines k-Cores nicht beliebig orientieren kann (Beispiel Stern).

$$\Rightarrow \forall v \in V : |\mathcal{N}^+(v)| \leq D(G)$$

3.2 – Dreiecke zählen

Algorithm DREIECKE

KANTEN-ORIENTIERUNG()

$D = 0$

for $v \in V$ **do**

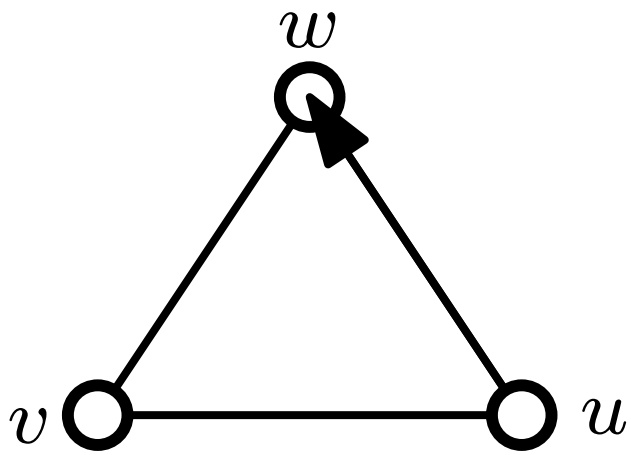
  for $u \in \mathcal{N}(v)$ **do**

    for $w \in \mathcal{N}^+(u)$ **do**

      if ADJAZENT(v, w) **then**

        $D = D + 1$

return $\frac{D}{3}$



3.2 – Dreiecke zählen

INIT_ADJAZENZ()

- Initialisiere Array

SET_ADJAZENZ(v)

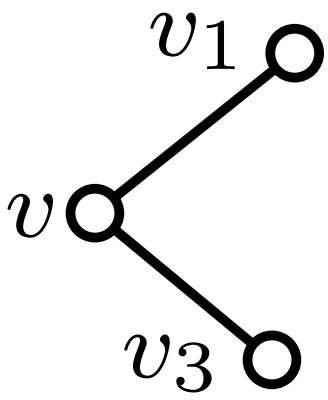
- Setze Bits in $\mathcal{N}(v)$

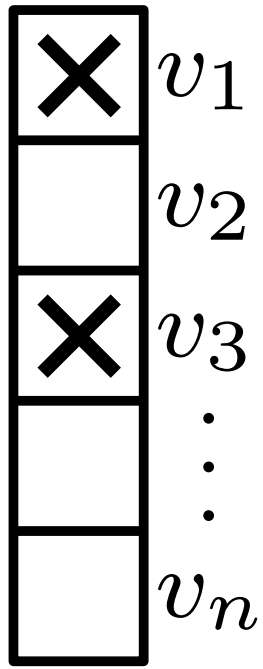
UNSET_ADJAZENZ(v)

- Entferne Bits in $\mathcal{N}(v)$

ADJAZENT(v, w)

- Ist Bit an Stelle w gesetzt?





3.2 – Dreiecke zählen

Algorithm DREIECKE

KANTEN-ORIENTIERUNG()

INIT_ADJAZENZ()

$D = 0$

for $v \in V$ **do**

SET_ADJAZENZ(v)

for $u \in \mathcal{N}(v)$ **do**

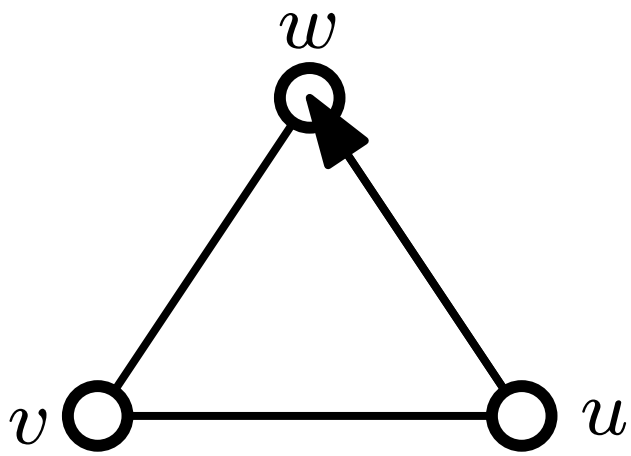
for $w \in \mathcal{N}^+(u)$ **do**

if ADJAZENT(v, w) **then**

$D = D + 1$

UNSET_ADJAZENZ(v)

return $\frac{D}{3}$



3.3 – Entartetheit von planaren Graphen

Zeigen Sie, dass für planare Graphen G $D(G) \leq 5$ gilt.

- In planaren Graphen gibt es mindestens einen Knoten v mit $\delta(v) \leq 5$.
- $G - v$ ist wieder planar.

3.3 – Entartetheit von planaren Graphen

Zeigen Sie, dass für planare Graphen G $D(G) \leq 5$ gilt.

- In planaren Graphen gibt es mindestens einen Knoten v mit $\delta(v) \leq 5$.
- $G - v$ ist wieder planar.