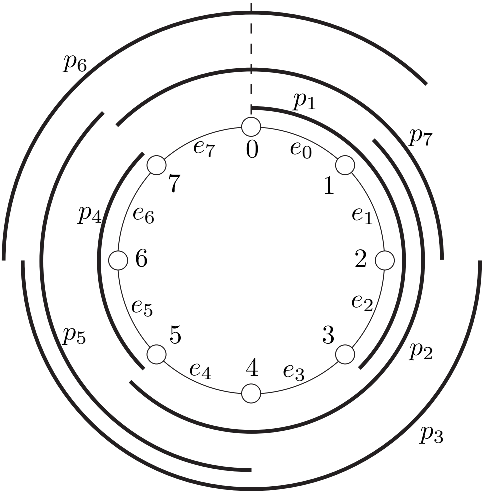
Call-Control in Ringnetzwerken

Seminar „Algorithmen und Datenstrukturen“

Universität Augsburg

Michael Markl



p_6
p_1
p_7
0
e_7
e_0
7
1
p_4
e_6
e_1
6
2
e_5
e_2
p_5
5
3
p_2
e_4
4
e_3
p_3

Gliederung

Problemdefinition

Problem in allgemeinen Graphen

Definition (Netzwerk)

Sei (V, E) ein ungerichteter Graph mit Knoten V und Kanten E, und $c : E \to \mathbb{N}$ eine Kapazitätsfunktion. Das Tupel (V, E, c) heißt (ungerichtetes) Netzwerk.

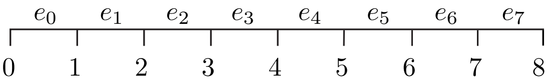
Definition (Call-Control)

Seien (V, E, c) ein ungerichtetes Netzwerk und P eine (Multi-)Menge von $m \in \mathbb{N}$ Pfaden in (V, E, c).
$Q \subseteq P$ heißt *zulässig*, falls für alle $e \in E$ die Anzahl aller Pfade in Q, die e enthalten, höchstens $c(e)$ ist.
Call-Control besteht darin, eine zulässige Menge Q maximaler Mächtigkeit zu finden.

Call-Control in Ketten

Definition (Kette)

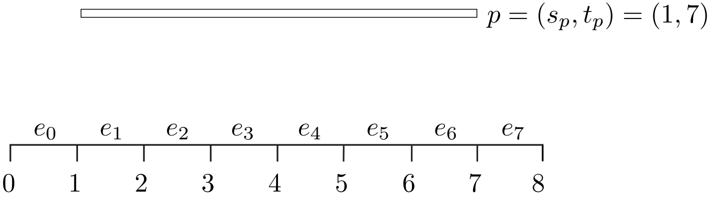
Eine *Kette* (V, E) ist ein Weg mit den Kanten
$E = \{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$ mit $v_i \neq v_j$ für $i \neq j$.



Call-Control in Ketten

Definition (Kette)

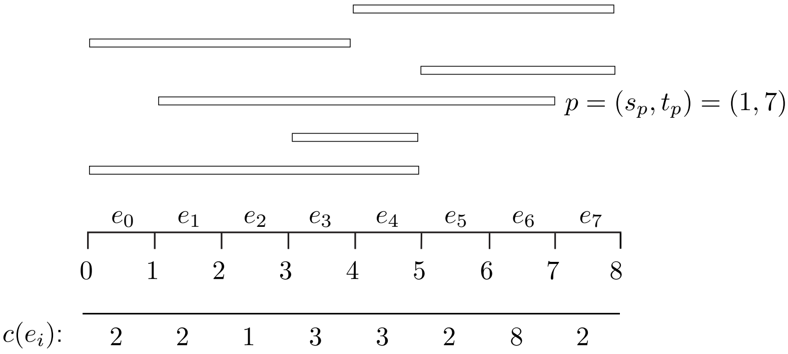
Eine *Kette* (V, E) ist ein Weg mit den Kanten
$E = \{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$ mit $v_i \neq v_j$ für $i \neq j$.



Call-Control in Ketten

Definition (Kette)

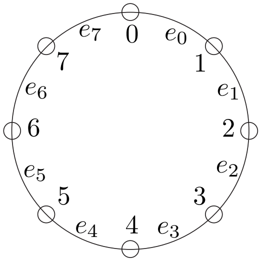
Eine *Kette* (V, E) ist ein Weg mit den Kanten
$E = \{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$ mit $v_i \neq v_j$ für $i \neq j$.



Call-Control in Ringen

Definition (Ring)

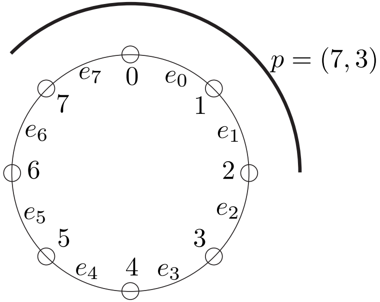
Ein *Ring* (V, E) ist ein Weg mit den Kanten $E = \{(v_0, v_1), \ldots, (v_{n-1}, v_n)\}$ mit $v_0 = v_n$ und $v_i \neq v_j$ für alle anderen $i \neq j$.



Call-Control in Ringen

Definition (Ring)

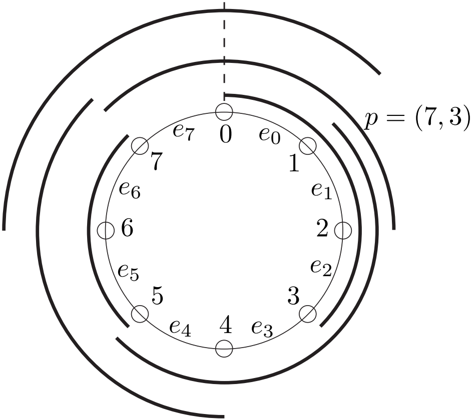
Ein *Ring* (V, E) ist ein Weg mit den Kanten $E = \{(v_0, v_1), \ldots, (v_{n-1}, v_n)\}$ mit $v_0 = v_n$ und $v_i \neq v_j$ für alle anderen $i \neq j$.



Call-Control in Ringen

Definition (Ring)

Ein *Ring* (V, E) ist ein Weg mit den Kanten $E = \{(v_0, v_1), \ldots, (v_{n-1}, v_n)\}$ mit $v_0 = v_n$ und $v_i \neq v_j$ für alle anderen $i \neq j$.

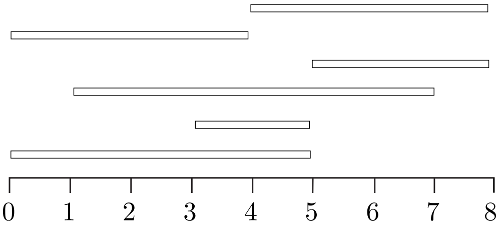


Call-Control in Ketten

Gierige Ordnung

Definition (Gierige Ordnung)

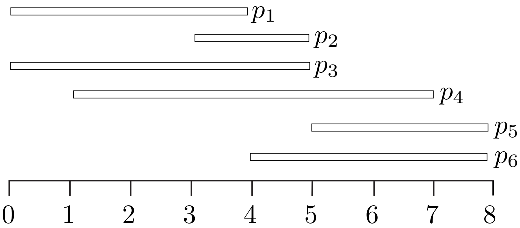
Auf einer Menge P von Pfaden in einer Kette nennen wir eine Totalordnung $\leq_G$ mit zugehöriger strenger Totalordnung $<_G$ *gierig*, falls $\forall p, q \in P\colon t_p < t_q \Rightarrow p <_G q$.



Gierige Ordnung

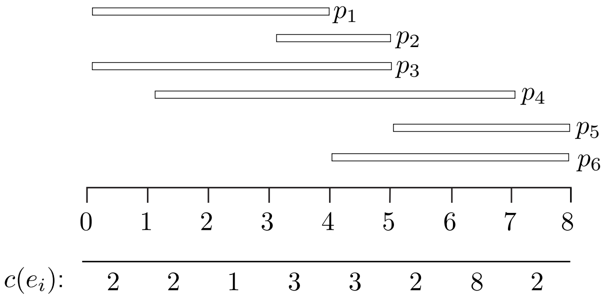
Definition (Gierige Ordnung)

Auf einer Menge P von Pfaden in einer Kette nennen wir eine Totalordnung $\leq_G$ mit zugehöriger strenger Totalordnung $<_G$ *gierig*, falls $\forall p, q \in P\colon t_p < t_q \Rightarrow p <_G q$.



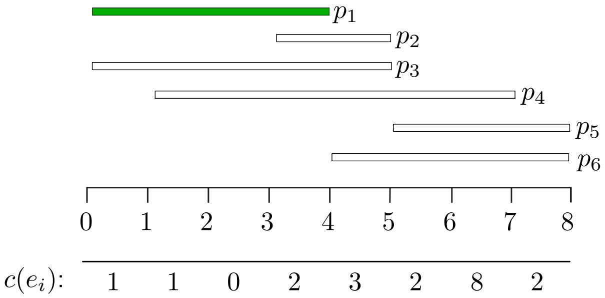
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



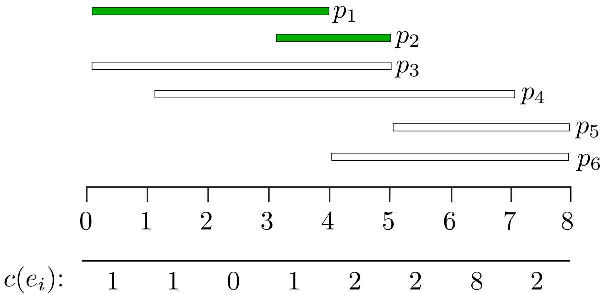
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



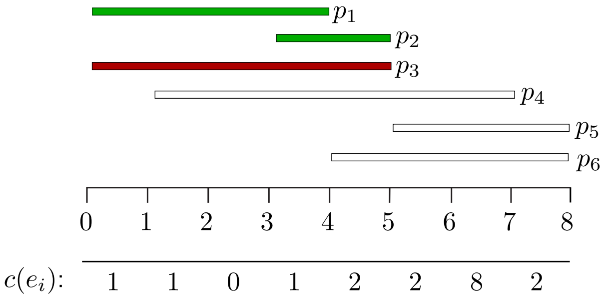
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



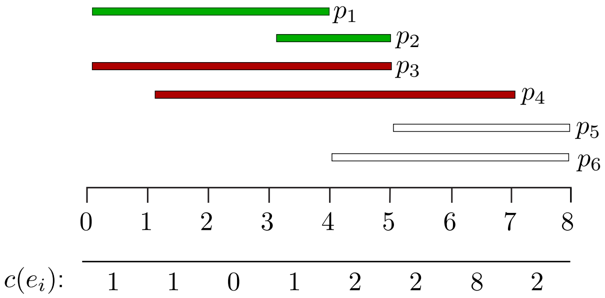
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



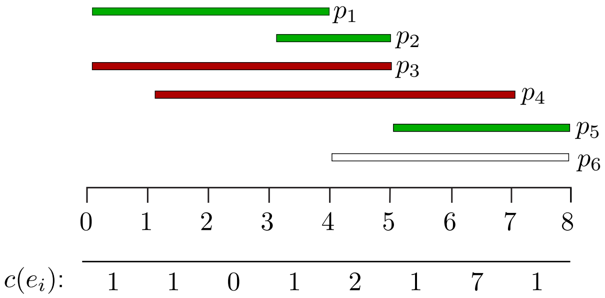
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



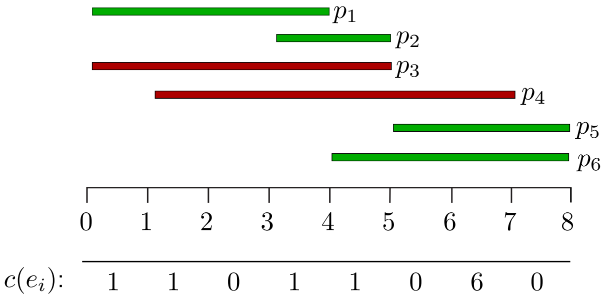
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



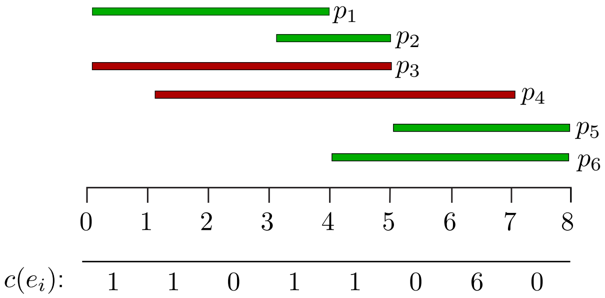
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



Das gierige Verfahren

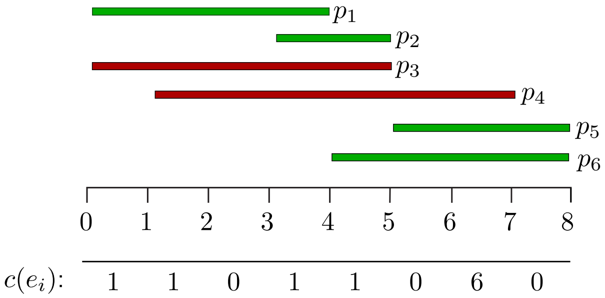
Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



- Menge der akzeptierten Pfade ist optimale Lösung.

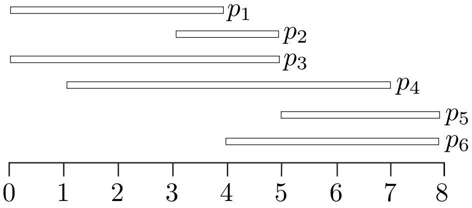
Das gierige Verfahren

Eine gierige Ordnung $\leq_G$ ist bereits gegeben.



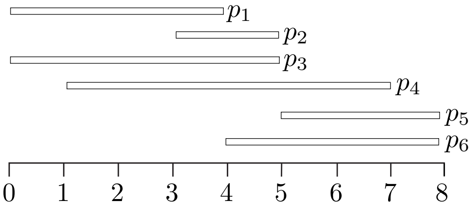
- Menge der akzeptierten Pfade ist optimale Lösung.
- Einfache Implementierung in $\mathcal{O}(m \cdot n)$ Zeit möglich (m Anzahl Pfade, n Anzahl Knoten).

Korrektheit des gierigen Verfahrens – I



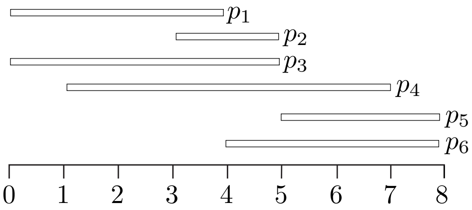
- Seien $A = \{a_1, \ldots, a_k\}$ und $B = \{b_1, \ldots, b_k\}$ Teilmengen der Pfade P mit $a_1 \leq_G \cdots \leq_G a_k$ und $b_1 \leq_G \cdots \leq_G b_k$. Wir schreiben $A \leq_G B$, falls $\forall i \leq k\colon a_k \leq_G b_k$.

Korrektheit des gierigen Verfahrens – I



- Seien $A = \{a_1, \ldots, a_k\}$ und $B = \{b_1, \ldots, b_k\}$ Teilmengen der Pfade P mit $a_1 \leq_G \cdots \leq_G a_k$ und $b_1 \leq_G \cdots \leq_G b_k$. Wir schreiben $A \leq_G B$, falls $\forall i \leq k \colon a_k \leq_G b_k$.
- Bsp.: $\{p_1, p_3, p_6\} \leq_G \{p_1, p_4, p_6\}$.

Korrektheit des gierigen Verfahrens – I



- Seien $A = \{a_1, \ldots, a_k\}$ und $B = \{b_1, \ldots, b_k\}$ Teilmengen der Pfade P mit $a_1 \leq_G \cdots \leq_G a_k$ und $b_1 \leq_G \cdots \leq_G b_k$. Wir schreiben $A \leq_G B$, falls $\forall i \leq k \colon a_k \leq_G b_k$.
- Bsp.: $\{p_1, p_3, p_6\} \leq_G \{p_1, p_4, p_6\}$.
- Eine zulässige Menge A heißt minimal, falls $A \leq_G B$ für alle zulässigen Mengen B mit $|A| = |B|$.

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 *mit* $k \in \mathbb{N}$ *Pfaden, so ist die Menge* G *der in gieriger Ordnung* $\leq_G$ *kleinsten* k *Pfade, die das gierige Verfahren berechnet, eine minimale Menge.*

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

- Transformiere Q_0 in k Schritten in G und erhalte:
  Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

- Transformiere Q_0 in k Schritten in G und erhalte: Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.
- I.S.: p sei i-ter Pfad von G

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

- Transformiere Q_0 in k Schritten in G und erhalte:
  Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.
- I.S.: p sei i-ter Pfad von G mit $p \notin Q_{i-1}$.

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

- Transformiere Q_0 in k Schritten in G und erhalte:
  Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.
- I.S.: p sei i-ter Pfad von G mit $p \notin Q_{i-1}$.
  q sei Pfad aus Q_{i-1} mit $q >_G p$ und kleinstem Startknoten.

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

- Transformiere Q_0 in k Schritten in G und erhalte:
  Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.
- I.S.: p sei i-ter Pfad von G mit $p \notin Q_{i-1}$.
  q sei Pfad aus Q_{i-1} mit $q >_G p$ und kleinstem Startknoten.
  Erhalte Q_i durch Ersetzen von q durch p in Q_{i-1}.

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

- Transformiere Q_0 in k Schritten in G und erhalte:
  Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.
- I.S.: p sei i-ter Pfad von G mit $p \notin Q_{i-1}$.
  q sei Pfad aus Q_{i-1} mit $q >_G p$ und kleinstem Startknoten.
  Erhalte Q_i durch Ersetzen von q durch p in Q_{i-1}.
  Dann $Q_i \leq_G Q_{i-1}$.

Korrektheit des gierigen Verfahrens – II

Lemma (Optimalität des gierigen Verfahrens)

Existiert eine zulässige Teilmenge Q_0 mit $k \in \mathbb{N}$ Pfaden, so ist die Menge G der in gieriger Ordnung $\leq_G$ kleinsten k Pfade, die das gierige Verfahren berechnet, eine minimale Menge.

Beweisskizze:

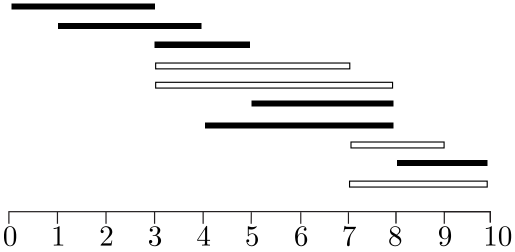
- Transformiere Q_0 in k Schritten in G und erhalte: Q_i zulässig, $Q_{i+1} \leq_G Q_i$ und Q_i stimmt auf ersten i Pfaden mit G überein.
- I.S.: p sei i-ter Pfad von G mit $p \notin Q_{i-1}$. q sei Pfad aus Q_{i-1} mit $q >_G p$ und kleinstem Startknoten. Erhalte Q_i durch Ersetzen von q durch p in Q_{i-1}. Dann $Q_i \leq_G Q_{i-1}$. Mit I.V.: Q_i zulässig, da keine Kantenkapazität verletzt wird.

Algorithmus für identische Kapazitäten

- Feste Kapazität $C \in \mathbb{N}$ für alle Kanten.

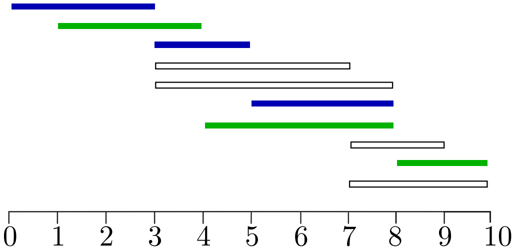
Algorithmus für identische Kapazitäten

- Feste Kapazität $C \in \mathbb{N}$ für alle Kanten.
- Beispiel mit $C = 2$:



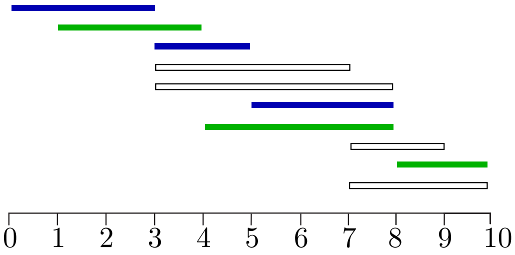
Algorithmus für identische Kapazitäten

- Feste Kapazität $C \in \mathbb{N}$ für alle Kanten.
- Beispiel mit $C = 2$:



Algorithmus für identische Kapazitäten

- Feste Kapazität $C \in \mathbb{N}$ für alle Kanten.
- Beispiel mit $C = 2$:



- Call-Control-Problem entspricht maximaler C-Färbung.

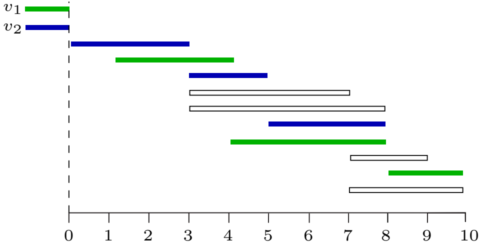
Algorithmus für C-Färbung

Algorithmus für C-Färbung

- Füge virtuelle Pfade $v_1, \ldots, v_C$, je unterschiedlich gefärbt in einer der C Farben, vor allen Pfaden ein.

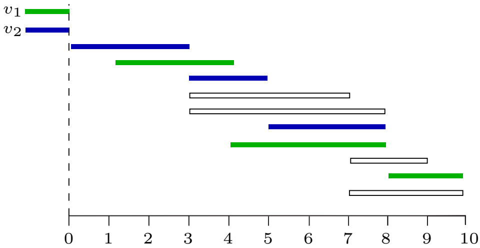
Algorithmus für C-Färbung

- Füge virtuelle Pfade $v_1, \ldots, v_C$, je unterschiedlich gefärbt in einer der C Farben, vor allen Pfaden ein.
- Speichere zu jeder Farbe c den *aktuellen Anführer von* c (den in $\leq_G$ größten c-gefärbten Pfad). Zu Beginn: Der zugehörige virtuelle Pfad.



Algorithmus für C-Färbung

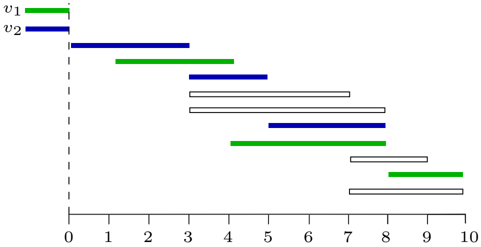
- Füge virtuelle Pfade $v_1, \ldots, v_C$, je unterschiedlich gefärbt in einer der C Farben, vor allen Pfaden ein.
- Speichere zu jeder Farbe c den *aktuellen Anführer von* c (den in $\leq_G$ größten c-gefärbten Pfad). Zu Beginn: Der zugehörige virtuelle Pfad.



- Suche bei Bearbeitung von Pfad p den *optimalen Anführer von* p, d.h. den aktuell größten Anführer, der sich nicht mit p überschneidet.

Algorithmus für C-Färbung

- Füge virtuelle Pfade $v_1, \ldots, v_C$, je unterschiedlich gefärbt in einer der C Farben, vor allen Pfaden ein.
- Speichere zu jeder Farbe c den *aktuellen Anführer von* c (den in $\leq_G$ größten c-gefärbten Pfad). Zu Beginn: Der zugehörige virtuelle Pfad.



- Suche bei Bearbeitung von Pfad p den *optimalen Anführer von* p, d.h. den aktuell größten Anführer, der sich nicht mit p überschneidet.
- Ist das möglich in gesamt-linearer Zeit?

C-Färbung – Union-Find-Algorithmus

C-Färbung – Union-Find-Algorithmus

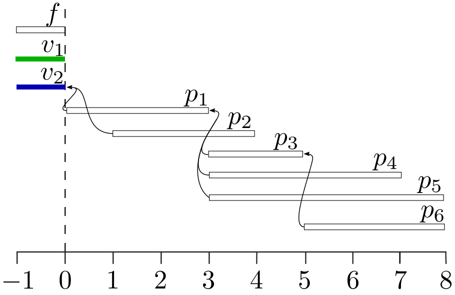
- Füge weiteren Pfad f, den fiktiven Anführer, als ersten Pfad ein.

C-Färbung – Union-Find-Algorithmus

- Füge weiteren Pfad f, den fiktiven Anführer, als ersten Pfad ein.
- Ermittle für jeden Pfad p seinen *bevorzugten Anführer*, d.h. den in $\leq_G$ größten Pfad, dessen Zielknoten nicht nach dem Anfangsknoten s_p kommt.

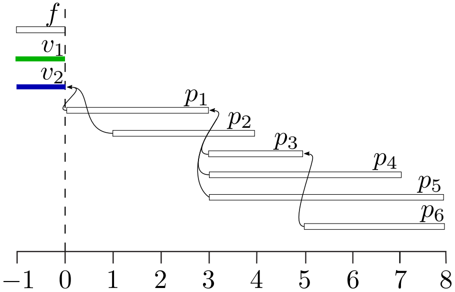
C-Färbung – Union-Find-Algorithmus

- Füge weiteren Pfad f, den fiktiven Anführer, als ersten Pfad ein.
- Ermittle für jeden Pfad p seinen *bevorzugten Anführer*, d.h. den in $\leq_G$ größten Pfad, dessen Zielknoten nicht nach dem Anfangsknoten s_p kommt.



C-Färbung – Union-Find-Algorithmus

- Füge weiteren Pfad f, den fiktiven Anführer, als ersten Pfad ein.
- Ermittle für jeden Pfad p seinen *bevorzugten Anführer*, d.h. den in $\leq_G$ größten Pfad, dessen Zielknoten nicht nach dem Anfangsknoten s_p kommt.



- Erstelle Union-Find-Instanz mit jedem Pfad in eigener Gruppe.

C-Färbung – Union-Find-Algorithmus

- Invarianten:

C-Färbung – Union-Find-Algorithmus

- Invarianten:
  - Repräsent einer Gruppe ist in $\leq_G$ kleinster Pfad der Gruppe.

C-Färbung – Union-Find-Algorithmus

- Invarianten:
  - Repräsent einer Gruppe ist in $\leq_G$ kleinster Pfad der Gruppe.
  - Gruppen enthalten nur in $\leq_G$ aufeinanderfolgende Pfade

C-Färbung – Union-Find-Algorithmus

- Invarianten:
  - Repräsent einer Gruppe ist in $\leq_G$ kleinster Pfad der Gruppe.
  - Gruppen enthalten nur in $\leq_G$ aufeinanderfolgende Pfade
  - Nicht verarbeitete Pfade sind in Einzelgruppen; Repräsentant einer verarbeiteten Gruppe ist Anführer einer Farbe oder der fiktive Anführer.

C-Färbung – Union-Find-Algorithmus

- Invarianten:
  - Repräsent einer Gruppe ist in $\leq_G$ kleinster Pfad der Gruppe.
  - Gruppen enthalten nur in $\leq_G$ aufeinanderfolgende Pfade
  - Nicht verarbeitete Pfade sind in Einzelgruppen; Repräsentant einer verarbeiteten Gruppe ist Anführer einer Farbe oder der fiktive Anführer.
- Bei Bearbeitung von Pfad p mit bevorzugtem Anführer q:

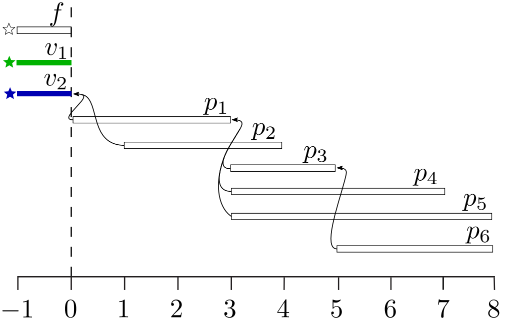
C-Färbung – Union-Find-Algorithmus

- Invarianten:
  - Repräsent einer Gruppe ist in $\leq_G$ kleinster Pfad der Gruppe.
  - Gruppen enthalten nur in $\leq_G$ aufeinanderfolgende Pfade
  - Nicht verarbeitete Pfade sind in Einzelgruppen; Repräsentant einer verarbeiteten Gruppe ist Anführer einer Farbe oder der fiktive Anführer.
- Bei Bearbeitung von Pfad p mit bevorzugtem Anführer q:
  - *find*$(q) = f$: Verwerfe p und vereinige die Gruppe von p mit der des Vorgängers von p.

C-Färbung – Union-Find-Algorithmus

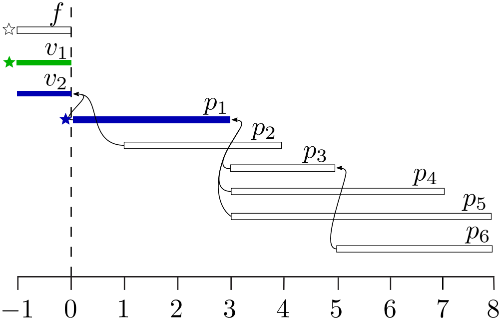
- Invarianten:
  - Repräsent einer Gruppe ist in $\leq_G$ kleinster Pfad der Gruppe.
  - Gruppen enthalten nur in $\leq_G$ aufeinanderfolgende Pfade
  - Nicht verarbeitete Pfade sind in Einzelgruppen; Repräsentant einer verarbeiteten Gruppe ist Anführer einer Farbe oder der fiktive Anführer.
- Bei Bearbeitung von Pfad p mit bevorzugtem Anführer q:
  - *find*$(q) = f$: Verwerfe p und vereinige die Gruppe von p mit der des Vorgängers von p.
  - *find*(q) ist c-gefärbter Anführer: Akzeptiere p, färbe p in c und vereinige die Gruppe von *find*(q) mit der des Vorgängers von *find*(q).

C-Färbung – Union-Find-Algorithmus am Beispiel



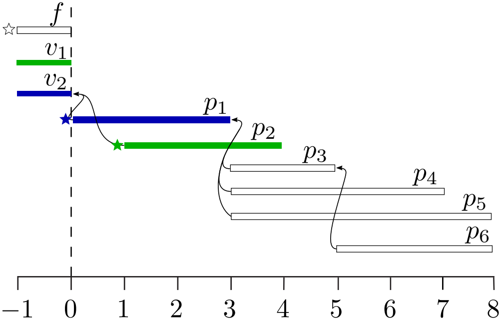
initial: f | v_1 | v_2 | p_1 | p_2 | p_3 | p_4 | p_5 | p_6

C-Färbung – Union-Find-Algorithmus am Beispiel



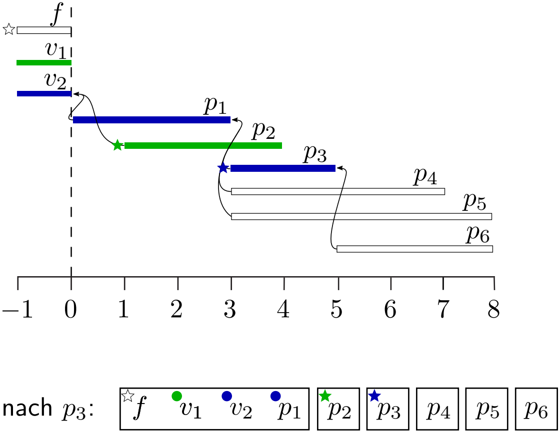
nach p_1:

C-Färbung – Union-Find-Algorithmus am Beispiel

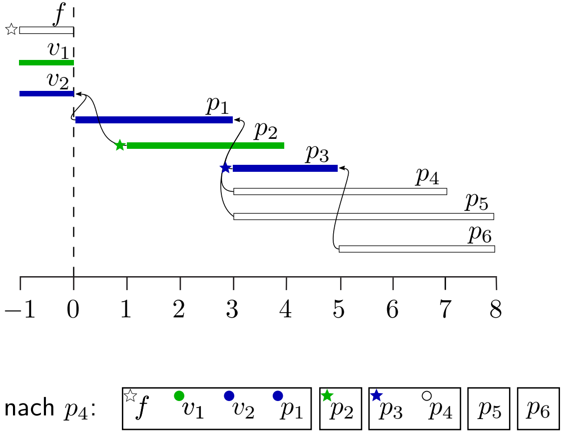


nach p_2: f v_1 v_2 | p_1 | p_2 | p_3 | p_4 | p_5 | p_6

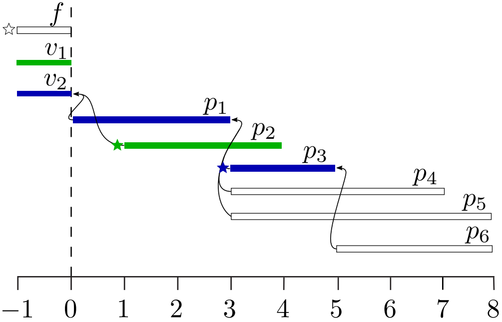
C-Färbung – Union-Find-Algorithmus am Beispiel



C-Färbung – Union-Find-Algorithmus am Beispiel

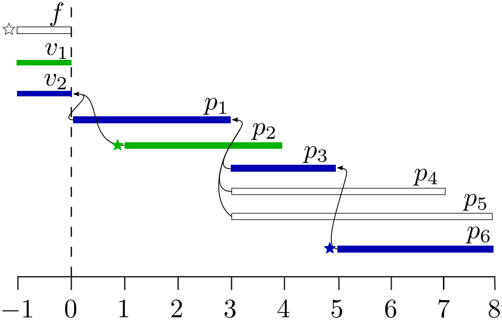


C-Färbung – Union-Find-Algorithmus am Beispiel



nach p_5: $\boxed{f \quad v_1 \quad v_2 \quad p_1}$ $\boxed{p_2}$ $\boxed{p_3 \quad p_4 \quad p_5}$ $\boxed{p_6}$

C-Färbung – Union-Find-Algorithmus am Beispiel



nach p_6: $\boxed{f \quad v_1 \quad v_2 \quad p_1}$ $\boxed{p_2 \quad p_3 \quad p_4 \quad p_5}$ $\boxed{p_6}$

C-Färbung – Union-Find-Algorithmus

C-Färbung – Union-Find-Algorithmus

- Wir benötigen m find- und union-Aufrufe.

C-Färbung – Union-Find-Algorithmus

- Wir benötigen m find- und union-Aufrufe.
- Alle Vereinigungen geschehen entlang einer Kette.

C-Färbung – Union-Find-Algorithmus

- Wir benötigen m find- und union-Aufrufe.
- Alle Vereinigungen geschehen entlang einer Kette.
- Mit Static-Tree-Set-Union benötigen wir $\mathcal{O}(m)$ Zeit dafür (Gabow und Tarjan in [3]).

C-Färbung – Union-Find-Algorithmus

- Wir benötigen m find- und union-Aufrufe.
- Alle Vereinigungen geschehen entlang einer Kette.
- Mit Static-Tree-Set-Union benötigen wir $\mathcal{O}(m)$ Zeit dafür (Gabow und Tarjan in [3]).
- Das Call-Control-Problem mit identischen Kapazitäten ist in $\mathcal{O}(m)$ Zeit optimal lösbar, wenn die Menge der Pfade bereits in gieriger Ordnung sortiert ist.

C-Färbung – Union-Find-Algorithmus

- Wir benötigen m find- und union-Aufrufe.
- Alle Vereinigungen geschehen entlang einer Kette.
- Mit Static-Tree-Set-Union benötigen wir $\mathcal{O}(m)$ Zeit dafür (Gabow und Tarjan in [3]).
- Das Call-Control-Problem mit identischen Kapazitäten ist in $\mathcal{O}(m)$ Zeit optimal lösbar, wenn die Menge der Pfade bereits in gieriger Ordnung sortiert ist.
- Genauere Analyse des Verfahrens durch Carlisle und Lloyd in [2].

Anpassen für willkürliche Kapazitäten

Anpassen für willkürliche Kapazitäten

- Betrachten willkürliche Kapazitäten $c : E \to \mathbb{N}$.

Anpassen für willkürliche Kapazitäten

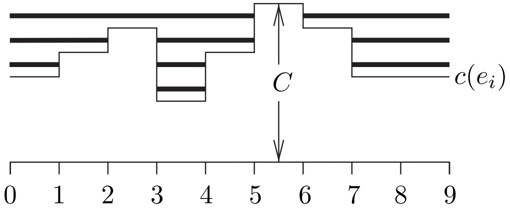
- Betrachten willkürliche Kapazitäten $c : E \to \mathbb{N}$.
- Anpassung der Idee für identische Kapazitäten:

Anpassen für willkürliche Kapazitäten

- Betrachten willkürliche Kapazitäten $c : E \to \mathbb{N}$.
- Anpassung der Idee für identische Kapazitäten:
  - Setze $C := \max_{e \in E} c(e)$ als neue Kapazität jeder Kante.

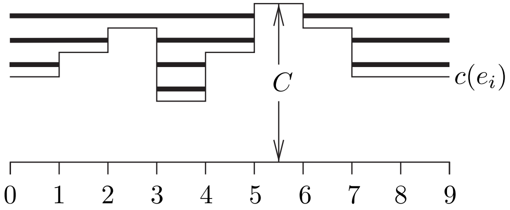
Anpassen für willkürliche Kapazitäten

- Betrachten willkürliche Kapazitäten $c : E \to \mathbb{N}$.
- Anpassung der Idee für identische Kapazitäten:
  - Setze $C := \max_{e \in E} c(e)$ als neue Kapazität jeder Kante.
  - Füge an Kanten mit überflüssigen Kapazitäten Platzhalterpfade ein:



Anpassen für willkürliche Kapazitäten

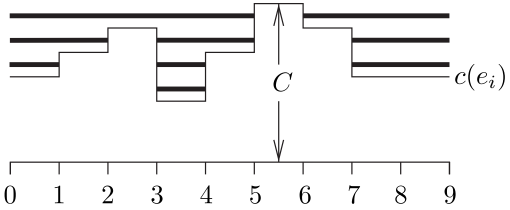
- Betrachten willkürliche Kapazitäten $c : E \to \mathbb{N}$.
- Anpassung der Idee für identische Kapazitäten:
  - Setze $C := \max_{e \in E} c(e)$ als neue Kapazität jeder Kante.
  - Füge an Kanten mit überflüssigen Kapazitäten Platzhalterpfade ein:



  - Sorge dafür, dass alle Platzhalterpfade akzeptiert werden.

Anpassen für willkürliche Kapazitäten

- Betrachten willkürliche Kapazitäten $c : E \to \mathbb{N}$.
- Anpassung der Idee für identische Kapazitäten:
  - Setze $C := \max_{e \in E} c(e)$ als neue Kapazität jeder Kante.
  - Füge an Kanten mit überflüssigen Kapazitäten Platzhalterpfade ein:

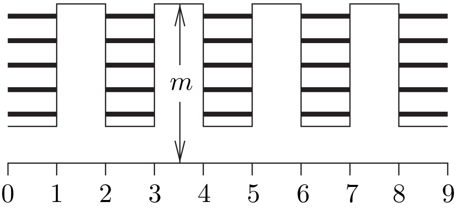


  - Sorge dafür, dass alle Platzhalterpfade akzeptiert werden.
- Probleme: Anzahl Platzhalter? Wie akzeptieren wir alle Platzhalter?

Anzahl der Platzhalterpfade

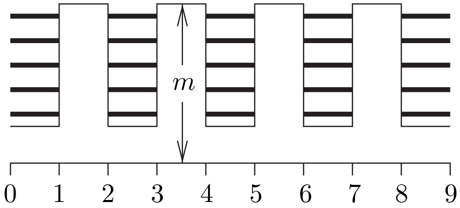
Anzahl der Platzhalterpfade

- Für bestimmte Kettennetzwerke kann es passieren, dass wir $\Omega(n \cdot m)$ Platzhalter einfügen:



Anzahl der Platzhalterpfade

- Für bestimmte Kettennetzwerke kann es passieren, dass wir $\Omega(n \cdot m)$ Platzhalter einfügen:



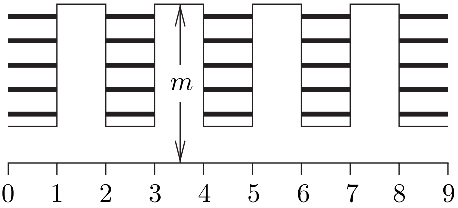
- Flache die Kapazitäten ab mit

$$c'(e_i) = \begin{cases} \min(c(e_0), n_0) & \text{für } i = 0 \\ \min(c(e_i), c'(e_{i-1}) + n_i) & \text{für } i \geq 1 \end{cases}$$

wobei n_i die Anzahl der Pfade in P mit Anfangsknoten i ist.

Anzahl der Platzhalterpfade

- Für bestimmte Kettennetzwerke kann es passieren, dass wir $\Omega(n \cdot m)$ Platzhalter einfügen:



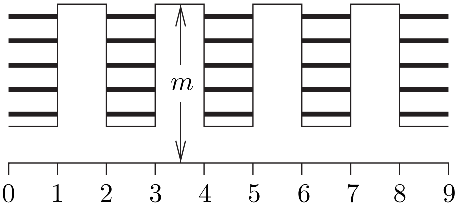
- Flache die Kapazitäten ab mit

$$c'(e_i) = \begin{cases} \min(c(e_0), n_0) & \text{für } i = 0 \\ \min(c(e_i), c'(e_{i-1}) + n_i) & \text{für } i \geq 1 \end{cases}$$

  wobei n_i die Anzahl der Pfade in P mit Anfangsknoten i ist.
- Damit werden nur $\mathcal{O}(m)$ Platzhalter generiert.

Anzahl der Platzhalterpfade

- Für bestimmte Kettennetzwerke kann es passieren, dass wir $\Omega(n \cdot m)$ Platzhalter einfügen:



- Flache die Kapazitäten ab mit

$$c'(e_i) = \begin{cases} \min(c(e_0), n_0) & \text{für } i = 0 \\ \min(c(e_i), c'(e_{i-1}) + n_i) & \text{für } i \geq 1 \end{cases}$$

  wobei n_i die Anzahl der Pfade in P mit Anfangsknoten i ist.
- Damit werden nur $\mathcal{O}(m)$ Platzhalter generiert.
- Anpassen der Kapazitäten und Auffüllen mit Platzhaltern in $\mathcal{O}(n+m)$ Zeit möglich.

Akzeptieren der Platzhalter

Akzeptieren der Platzhalter

Ähnliches Vorgehen wie zuvor, versuche nun aber Platzhalterpfade möglichst früh zu bearbeiten:

Akzeptieren der Platzhalter

Ähnliches Vorgehen wie zuvor, versuche nun aber Platzhalterpfade möglichst früh zu bearbeiten:

- Erstelle eine Liste L der Endknoten aller Pfade (zu jedem Eintrag der Liste speichere eine Referenz auf zugehörigen Pfad):

Akzeptieren der Platzhalter

Ähnliches Vorgehen wie zuvor, versuche nun aber Platzhalterpfade möglichst früh zu bearbeiten:

- Erstelle eine Liste L der Endknoten aller Pfade (zu jedem Eintrag der Liste speichere eine Referenz auf zugehörigen Pfad):
  - Nach Endknoten aufsteigend sortiert
  - Bei gleichem Endknoten sollen Anfangsknoten vor Zielknoten geordnet werden

Akzeptieren der Platzhalter

Ähnliches Vorgehen wie zuvor, versuche nun aber Platzhalterpfade möglichst früh zu bearbeiten:

- Erstelle eine Liste L der Endknoten aller Pfade (zu jedem Eintrag der Liste speichere eine Referenz auf zugehörigen Pfad):
  - Nach Endknoten aufsteigend sortiert
  - Bei gleichem Endknoten sollen Anfangsknoten vor Zielknoten geordnet werden
- Erhalte $\leq_G$ durch Ersetzen von Zielknoten der Liste durch die zug. Pfade.

Akzeptieren der Platzhalter

Ähnliches Vorgehen wie zuvor, versuche nun aber Platzhalterpfade möglichst früh zu bearbeiten:

- Erstelle eine Liste L der Endknoten aller Pfade (zu jedem Eintrag der Liste speichere eine Referenz auf zugehörigen Pfad):
  - Nach Endknoten aufsteigend sortiert
  - Bei gleichem Endknoten sollen Anfangsknoten vor Zielknoten geordnet werden
- Erhalte $\leq_G$ durch Ersetzen von Zielknoten der Liste durch die zug. Pfade.
- Füge wieder C virtuelle Pfade sowie den fiktiven Anführer vor den anderen Pfaden ein und ordne bevorzugte Anführer zu.

Akzeptieren der Platzhalter

Ähnliches Vorgehen wie zuvor, versuche nun aber Platzhalterpfade möglichst früh zu bearbeiten:

- Erstelle eine Liste L der Endknoten aller Pfade (zu jedem Eintrag der Liste speichere eine Referenz auf zugehörigen Pfad):
  - Nach Endknoten aufsteigend sortiert
  - Bei gleichem Endknoten sollen Anfangsknoten vor Zielknoten geordnet werden
- Erhalte $\leq_G$ durch Ersetzen von Zielknoten der Liste durch die zug. Pfade.
- Füge wieder C virtuelle Pfade sowie den fiktiven Anführer vor den anderen Pfaden ein und ordne bevorzugte Anführer zu.
- Statt die Pfade in der Reihenfolge von $\leq_G$ zu bearbeiten wird nun die Liste L durchlaufen und
  - Platzhalterpfade bei Antreffen ihres Anfangsknotens
  - Originalpfade bei Antreffen ihres Zielknotens

  wie im vorherigen Algorithmus verarbeitet.

Korrektheit des Algorithmus I

Lemma (Korrektheit des Algorithmus)

Der beschriebene Algorithmus U ist eine korrekte Implementierung des gierigen Verfahrens G für willkürliche Kapazitäten.

Beweisskizze:

Korrektheit des Algorithmus I

Lemma (Korrektheit des Algorithmus)

Der beschriebene Algorithmus U ist eine korrekte Implementierung des gierigen Verfahrens G für willkürliche Kapazitäten.

Beweisskizze:

- U berechnet wieder C-Färbung der Pfade mit Platzhalter.

Korrektheit des Algorithmus I

Lemma (Korrektheit des Algorithmus)

Der beschriebene Algorithmus U ist eine korrekte Implementierung des gierigen Verfahrens G für willkürliche Kapazitäten.

Beweisskizze:

- U berechnet wieder C-Färbung der Pfade mit Platzhalter.
- U akzeptiert alle Platzhalterpfade (U berechnet insb. eine zulässige Menge).

Korrektheit des Algorithmus I

Lemma (Korrektheit des Algorithmus)

Der beschriebene Algorithmus U ist eine korrekte Implementierung des gierigen Verfahrens G für willkürliche Kapazitäten.

Beweisskizze:

- U berechnet wieder C-Färbung der Pfade mit Platzhalter.
- U akzeptiert alle Platzhalterpfade (U berechnet insb. eine zulässige Menge).
- U akzeptiert alle Pfade, die das gierige Verfahren akzeptiert.

Korrektheit des Algorithmus II

U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

Korrektheit des Algorithmus II

U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

- p erster Originalpfad mit unterschiedlicher Entscheidung.

Korrektheit des Algorithmus II

U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

- p erster Originalpfad mit unterschiedlicher Entscheidung.
  - A sei die Menge der (gemeinsam) akzeptierten Pfade vor p.

Korrektheit des Algorithmus II

U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

- p erster Originalpfad mit unterschiedlicher Entscheidung.
  - A sei die Menge der (gemeinsam) akzeptierten Pfade vor p.
  - Dann: U verwirft p, G akzeptiert p. Insb. $A \cup \{p\}$ zulässig.

Korrektheit des Algorithmus II

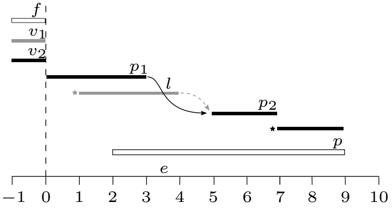
U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

- p erster Originalpfad mit unterschiedlicher Entscheidung.
  - A sei die Menge der (gemeinsam) akzeptierten Pfade vor p.
  - Dann: U verwirft p, G akzeptiert p. Insb. $A \cup \{p\}$ zulässig.
- Sei l kleinster Anführer einer Farbe zur Zeit der Bearbeitung von p, e die letzte Kante von l.

Korrektheit des Algorithmus II

U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

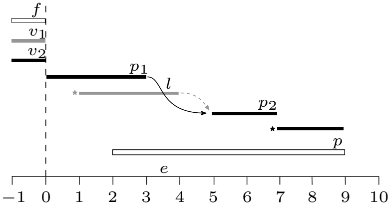
- p erster Originalpfad mit unterschiedlicher Entscheidung.
  - A sei die Menge der (gemeinsam) akzeptierten Pfade vor p.
  - Dann: U verwirft p, G akzeptiert p. Insb. $A \cup \{p\}$ zulässig.
- Sei l kleinster Anführer einer Farbe zur Zeit der Bearbeitung von p, e die letzte Kante von l.
- $\mathbb{A}$: $\exists c$ Farbe, sodass kein c-gefärbter Pfad e enthält.
  - Es ex. c-gefärbte Pfade links und rechts von e.
  - p_1, p_2 seien solche möglichst nah an e.
  - Widerspruch: l besserer Anführer von p_2 als p_1.



Korrektheit des Algorithmus II

U akzeptiert alle Pfade, die das gierige Verfahren G akzeptiert.
Über Widerspruch:

- p erster Originalpfad mit unterschiedlicher Entscheidung.
  - A sei die Menge der (gemeinsam) akzeptierten Pfade vor p.
  - Dann: U verwirft p, G akzeptiert p. Insb. $A \cup \{p\}$ zulässig.
- Sei l kleinster Anführer einer Farbe zur Zeit der Bearbeitung von p, e die letzte Kante von l.
- $\mathbb{A}$: $\exists c$ Farbe, sodass kein c-gefärbter Pfad e enthält.
  - Es ex. c-gefärbte Pfade links und rechts von e.
  - p_1, p_2 seien solche möglichst nah an e.
  - Widerspruch: l besserer Anführer von p_2 als p_1.



- Also gibt es (zusätzlich zu p) C weitere von G akzeptierte Pfade, die e beinhalten.

Rekapitulation

Was haben wir erreicht?

Rekapitulation

Was haben wir erreicht?

- Das gierige Verfahren löst das Call-Control-Problem für willkürliche Kapazitäten optimal.

Rekapitulation

Was haben wir erreicht?

- Das gierige Verfahren löst das Call-Control-Problem für willkürliche Kapazitäten optimal.
- Mit einer speziellen Union-Find-Struktur finden wir eine Implementierung für identische Kapazitäten mit Laufzeit $\mathcal{O}(m)$.

Rekapitulation

Was haben wir erreicht?

- Das gierige Verfahren löst das Call-Control-Problem für willkürliche Kapazitäten optimal.
- Mit einer speziellen Union-Find-Struktur finden wir eine Implementierung für identische Kapazitäten mit Laufzeit $\mathcal{O}(m)$.
- Wir verwenden dann eine angepasste Version mit Platzhalterpfaden, um willkürliche Kapazitäten zu erlauben, und erhalten:

Rekapitulation

Was haben wir erreicht?

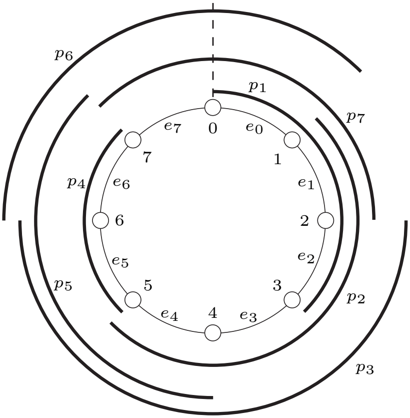
- Das gierige Verfahren löst das Call-Control-Problem für willkürliche Kapazitäten optimal.
- Mit einer speziellen Union-Find-Struktur finden wir eine Implementierung für identische Kapazitäten mit Laufzeit $\mathcal{O}(m)$.
- Wir verwenden dann eine angepasste Version mit Platzhalterpfaden, um willkürliche Kapazitäten zu erlauben, und erhalten:

Theorem

Das gierige Verfahren berechnet eine optimale Lösung für Call-Control in Ketten mit willkürlichen Kapazitäten und kann in einer Laufzeit von $\mathcal{O}(n + m)$ implementiert werden.

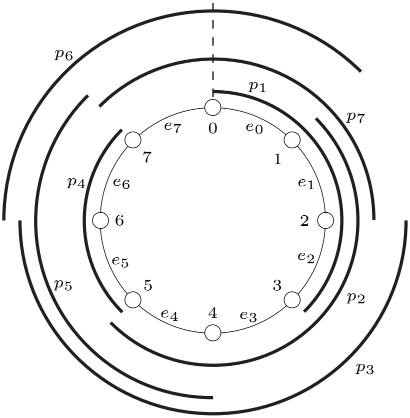
Call-Control in Ringen

Call-Control in Ringen



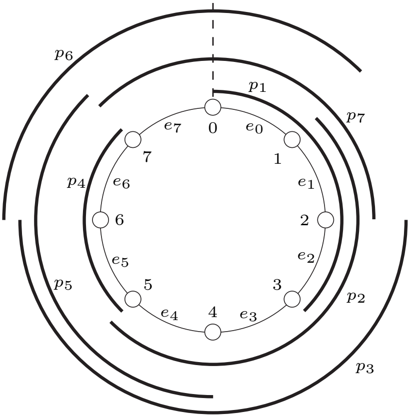
Call-Control in Ringen

- Teile Pfade P auf in die Pfade P_1, die 0 nicht als inneren Knoten haben, und die Pfade P_2, die 0 als inneren Knoten haben.



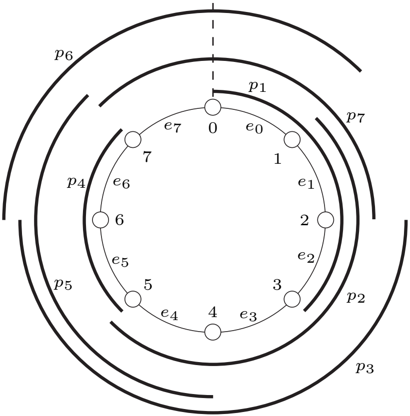
Call-Control in Ringen

- Teile Pfade P auf in die Pfade P_1, die 0 nicht als inneren Knoten haben, und die Pfade P_2, die 0 als inneren Knoten haben.
- Hänge zwei Kopien der Kanten $e_0, \ldots, e_n$ aneinander und erhalte mit zugehörigen Knoten $0, \ldots, 2n-1$ eine Kette.



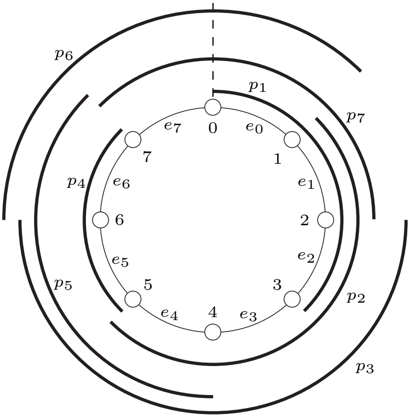
Call-Control in Ringen

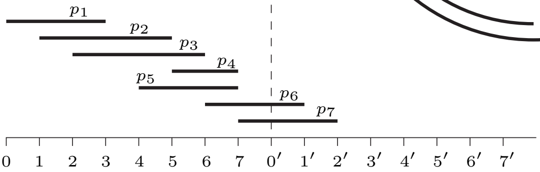
- Teile Pfade P auf in die Pfade P_1, die 0 nicht als inneren Knoten haben, und die Pfade P_2, die 0 als inneren Knoten haben.
- Hänge zwei Kopien der Kanten $e_0, \ldots, e_n$ aneinander und erhalte mit zugehörigen Knoten $0, \ldots, 2n-1$ eine Kette.
- Wir schreiben statt $n, \ldots, 2n-1$: $0', \ldots, (n-1)'$.



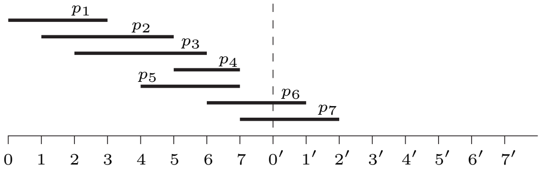
Call-Control in Ringen

- Teile Pfade P auf in die Pfade P_1, die 0 nicht als inneren Knoten haben, und die Pfade P_2, die 0 als inneren Knoten haben.
- Hänge zwei Kopien der Kanten $e_0, \ldots, e_n$ aneinander und erhalte mit zugehörigen Knoten $0, \ldots, 2n-1$ eine Kette.
- Wir schreiben statt $n, \ldots, 2n-1$: $0', \ldots, (n-1)'$.



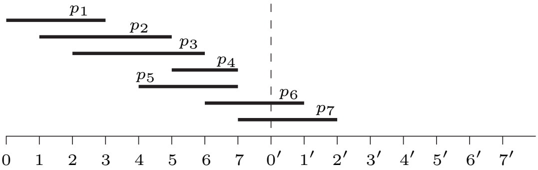


Belastung und Profil



Belastung und Profil

Mit $Q \subseteq P$ ist die *Belastung* $L_1(Q, e_i)$ die Anzahl Pfade in Q, die die erste Kopie der Kante e_i enthalten. Analog dazu: $L_2(Q, e_i)$.

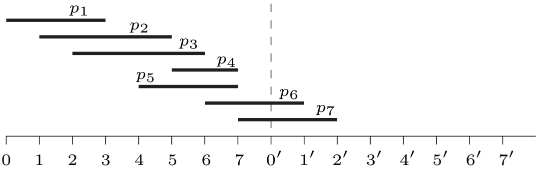


Belastung und Profil

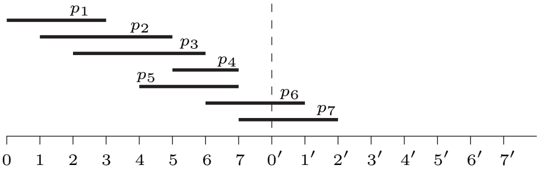
Mit $Q \subseteq P$ ist die *Belastung* $L_1(Q, e_i)$ die Anzahl Pfade in Q, die die erste Kopie der Kante e_i enthalten. Analog dazu: $L_2(Q, e_i)$.

Definition (Profil)

Mit $Q \subseteq P$ heißt die monoton fallende Folge $\pi_Q := (L_2(Q, e_0), \ldots, L_2(Q, e_{n-1}))$ das Profil von Q.
Außerdem $\pi_Q(e_i) := L_2(Q, e_i)$ und für zwei Profile π, π' schreiben wir $\pi \leq \pi'$, falls $\pi(e_i) \leq \pi'(e_i)$ für alle Kanten e_i.



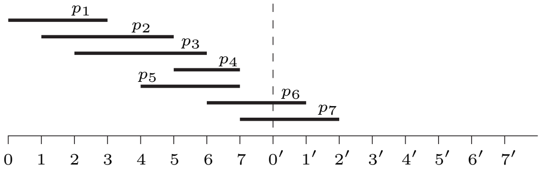
Kettenzulässigkeit



Kettenzulässigkeit

Definition (Kettenzulässig)

Eine Menge $Q \subseteq P$ heißt kettenzulässig, falls $L_1(Q, e) \leq c(e)$ für alle Kanten e. Gilt weiterhin $L_1(Q, e) + \pi(e) \leq c(e)$, so heißt Q kettenzulässig zum Startprofil π.



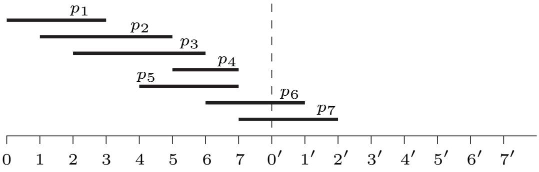
Kettenzulässigkeit

Definition (Kettenzulässig)

Eine Menge $Q \subseteq P$ heißt kettenzulässig, falls $L_1(Q, e) \leq c(e)$ für alle Kanten e. Gilt weiterhin $L_1(Q, e) + \pi(e) \leq c(e)$, so heißt Q kettenzulässig zum Startprofil π.

Insbesondere:

Q zulässig im Ring $\iff$ Q kettenzulässig zum Startprofil π_Q.



Der Algorithmus für Ringe

Der Algorithmus für Ringe

- Der Algorithmus ist nun wie folgt aufgebaut:

Der Algorithmus für Ringe

- Der Algorithmus ist nun wie folgt aufgebaut:
- Suche mit binärer Suche nach größtem $k \in \{0, \dots, m\}$, für das wir eine (im Ring) zulässige Menge Q mit $|Q| = k$ finden können.

Der Algorithmus für Ringe

- Der Algorithmus ist nun wie folgt aufgebaut:
- Suche mit binärer Suche nach größtem $k \in \{0, \ldots, m\}$, für das wir eine (im Ring) zulässige Menge Q mit $|Q| = k$ finden können.
- Das zugehörige Q ist dann optimale Lösung des Call-Control-Problems.

Prozedur: Finde Q zulässig mit $|Q| = k$

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \dots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.

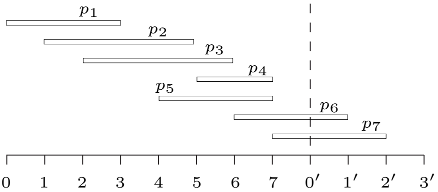
Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i + 1)$-te Runde.

Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i+1)$-te Runde.

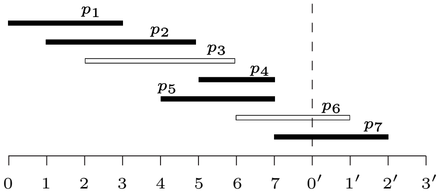
Beispiel für $k = 5$ und $c(e) = 2$ für alle $e \in E$:



Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i+1)$-te Runde.

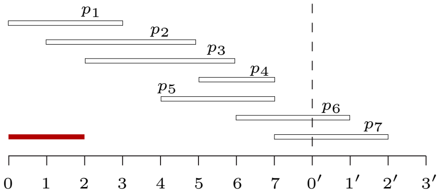
Beispiel für $k = 5$ und $c(e) = 2$ für alle $e \in E$:



Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i+1)$-te Runde.

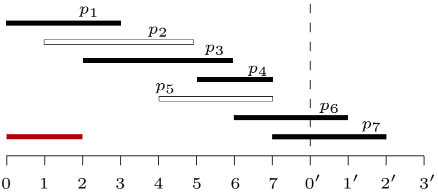
Beispiel für $k = 5$ und $c(e) = 2$ für alle $e \in E$:



Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \dots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i+1)$-te Runde.

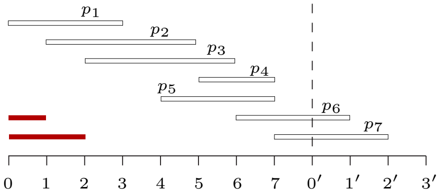
Beispiel für $k = 5$ und $c(e) = 2$ für alle $e \in E$:



Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i + 1)$-te Runde.

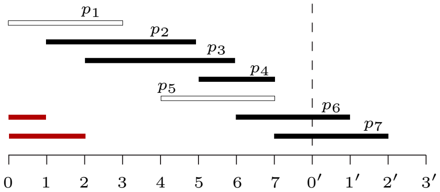
Beispiel für $k = 5$ und $c(e) = 2$ für alle $e \in E$:



Prozedur: Finde Q zulässig mit $|Q| = k$

- Starten mit $\pi_0 = (0, \ldots, 0)$ und $i = 1$.
- In der i-ten Runde:
  - Initialisiere Kapazitäten beider Kopien jeder Kante e mit $c(e)$, in der ersten Kopie um $\pi_{i-1}(e)$ reduziert.
  - Wende darauf gieriges Verfahren an und erhalte $G \subseteq P$.
  - Ist $|G| < k$, gibt die Prozedur zurück, dass keine k-elementige zulässige Menge existiert.
  - Sonst sei G_i die Menge der in $\leq_G$ kleinsten k Pfade von G und $\pi_i := \pi_{G_i}$.
  - Ist $\pi_i = \pi_{i-1}$, gibt die Prozedur G_i als zulässige Menge zurück.
  - Sonst gehe in $(i+1)$-te Runde.

Beispiel für $k = 5$ und $c(e) = 2$ für alle $e \in E$:



Korrektheit der Prozedur

Beobachtungen:

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Beweisskizze für die Korrektheit der Prozedur:

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Beweisskizze für die Korrektheit der Prozedur:

- Die Folge der π_i ist monoton wachsend.

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Beweisskizze für die Korrektheit der Prozedur:

- Die Folge der π_i ist monoton wachsend.
- Existiert eine zulässige Lösung Q^* mit k Pfaden, dann $\pi_i \leq \pi_{Q^*}$ für alle i.

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Beweisskizze für die Korrektheit der Prozedur:

- Die Folge der π_i ist monoton wachsend.
- Existiert eine zulässige Lösung Q^* mit k Pfaden, dann $\pi_i \leq \pi_{Q^*}$ für alle i.
- Die Prozedur macht höchstens $n \cdot c(e_0)$ Runden.

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Beweisskizze für die Korrektheit der Prozedur:

- Die Folge der π_i ist monoton wachsend.
- Existiert eine zulässige Lösung Q^* mit k Pfaden, dann $\pi_i \leq \pi_{Q^*}$ für alle i.
- Die Prozedur macht höchstens $n \cdot c(e_0)$ Runden.
  - Die Folge der Profile kann höchstens $\sum_{j=0}^{n-1} \pi_{Q^*}(e_j)$ mal echt wachsen.

Korrektheit der Prozedur

Beobachtungen:

- Existiert ein i mit $\pi_i = \pi_{i+1}$, so ist G_{i+1} eine gesuchte zulässige Menge, die von der Prozedur auch zurückgegeben wird.
- Das ist die einzige Möglichkeit, dass eine Menge zurückgegeben wird.

Beweisskizze für die Korrektheit der Prozedur:

- Die Folge der π_i ist monoton wachsend.
- Existiert eine zulässige Lösung Q^* mit k Pfaden, dann $\pi_i \leq \pi_{Q^*}$ für alle i.
- Die Prozedur macht höchstens $n \cdot c(e_0)$ Runden.
  - Die Folge der Profile kann höchstens $\sum_{j=0}^{n-1} \pi_{Q^*}(e_j)$ mal echt wachsen.
  - $\sum_{j=0}^{n-1} \pi_{Q^*}(e_j) \leq n \cdot \pi_{Q^*}(e_0) \leq n \cdot (e_0)$

Monotones Wachstum der Profile $(\pi_i)_i$

Zeige $\pi_i \leq \pi_{i+1}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

Monotones Wachstum der Profile $(\pi_i)_i$

Zeige $\pi_i \leq \pi_{i+1}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_{i-1} \leq \pi_i$.

Monotones Wachstum der Profile $(\pi_i)_i$

Zeige $\pi_i \leq \pi_{i+1}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_{i-1} \leq \pi_i$.
- G_i ist minimale zulässige Menge auf der Kette mit den durch π_{i-1} reduzierten Kapazitäten, da sie mit dem gierigen Verfahren berechnet wurde.

Monotones Wachstum der Profile $(\pi_i)_i$

Zeige $\pi_i \leq \pi_{i+1}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_{i-1} \leq \pi_i$.
- G_i ist minimale zulässige Menge auf der Kette mit den durch π_{i-1} reduzierten Kapazitäten, da sie mit dem gierigen Verfahren berechnet wurde.
- Da G_i zulässig zum Startprofil π_{i-1} und $\pi_{i-1} \leq \pi_i$, ist G_i auch zulässig zum Startprofil π_i.

Monotones Wachstum der Profile $(\pi_i)_i$

Zeige $\pi_i \leq \pi_{i+1}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_{i-1} \leq \pi_i$.
- G_i ist minimale zulässige Menge auf der Kette mit den durch π_{i-1} reduzierten Kapazitäten, da sie mit dem gierigen Verfahren berechnet wurde.
- Da G_i zulässig zum Startprofil π_{i-1} und $\pi_{i-1} \leq \pi_i$, ist G_i auch zulässig zum Startprofil π_i.
- Da G_i minimal, ist also $G_i \leq_G G_{i+1}$ und damit $\pi_i \leq \pi_{i+1}$.

Obere Grenze der Profile $(\pi_i)_i$

Sei Q^* zulässige Lösung mit k Pfaden. Zeige $\pi_i \leq \pi_{Q^*}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

Obere Grenze der Profile $(\pi_i)_i$

Sei Q^* zulässige Lösung mit k Pfaden. Zeige $\pi_i \leq \pi_{Q^*}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_i \leq \pi_{Q^*}$.

Obere Grenze der Profile $(\pi_i)_i$

Sei Q^* zulässige Lösung mit k Pfaden. Zeige $\pi_i \leq \pi_{Q^*}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_i \leq \pi_{Q^*}$.
- Da Q^* kettenzulässig zu π_{Q^*} ist mit I.V. Q^* auch kettenzulässig zu π_i.

Obere Grenze der Profile $(\pi_i)_i$

Sei Q^* zulässige Lösung mit k Pfaden. Zeige $\pi_i \leq \pi_{Q^*}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_i \leq \pi_{Q^*}$.
- Da Q^* kettenzulässig zu π_{Q^*} ist mit I.V. Q^* auch kettenzulässig zu π_i.
- G_{i+1} ist minimale kettenzulässige Menge zu π_i, da sie mit dem gierigen Verfahren berechnet wurde.

Obere Grenze der Profile $(\pi_i)_i$

Sei Q^* zulässige Lösung mit k Pfaden. Zeige $\pi_i \leq \pi_{Q^*}$ per Induktion (für $i = 0$ klar, da $\pi_0 = 0$).

- Es gelte $\pi_i \leq \pi_{Q^*}$.
- Da Q^* kettenzulässig zu π_{Q^*} ist mit I.V. Q^* auch kettenzulässig zu π_i.
- G_{i+1} ist minimale kettenzulässige Menge zu π_i, da sie mit dem gierigen Verfahren berechnet wurde.
- Also gilt $G_{i+1} \leq_G Q^*$, insbesondere $\pi_{i+1} \leq \pi_{Q^*}$.

Resultat

Wir fassen unser Ergebnis zusammen:

Resultat

Wir fassen unser Ergebnis zusammen:

- Wir benötigen also pro Runde $\mathcal{O}(n+m) = \mathcal{O}(m)$ Zeit.

Resultat

Wir fassen unser Ergebnis zusammen:

- Wir benötigen also pro Runde $\mathcal{O}(n+m) = \mathcal{O}(m)$ Zeit.
- Davon gibt es maximal $n \cdot c_{\min}$.

Resultat

Wir fassen unser Ergebnis zusammen:

- Wir benötigen also pro Runde $\mathcal{O}(n+m) = \mathcal{O}(m)$ Zeit.
- Davon gibt es maximal $n \cdot c_{\min}$.
- Mit binärer Suche wird die Prozedur $\mathcal{O}(\log m)$ mal aufgerufen.

Wir erhalten:

Resultat

Wir fassen unser Ergebnis zusammen:

- Wir benötigen also pro Runde $\mathcal{O}(n+m) = \mathcal{O}(m)$ Zeit.
- Davon gibt es maximal $n \cdot c_{\min}$.
- Mit binärer Suche wird die Prozedur $\mathcal{O}(\log m)$ mal aufgerufen.

Wir erhalten:

Theorem

Das Call-Control-Problem in Ringen kann in $\mathcal{O}(m \cdot n \cdot c_{\min} \cdot \log m)$ Zeit gelöst werden, wobei n die Anzahl der Knoten, m die Anzahl der Pfade und $c_{\min}$ die kleinste Kantenkapazität ist.

Literatur I

[1] Udo Adamy, Christoph Ambühl, R. Sai Anand, and Thomas Erlebach.
Call control in rings.
Algorithmica, 47:217–238, 2007.

[2] Martin C. Carlisle and Errol. L. Lloyd.
On the k-coloring of intervals.
Discrete Applied Mathematics, 59:225–235, 1995.

[3] Harold N. Gabow and Robert Endre Tarjan.
A linear-time algorithm for a special case of disjoint set union.
Journal of Computer and System Sciences, 30:209–221, 1985.

Literatur II

[4] Michael R. Garey, David S. Johnson, Gary L. Miller, and Christos H. Papadimitriou.
The complexity of coloring circular arcs and chords.
SIAM Journal on Algebraic and Discrete Methods, 1:216–227, 1980.

[5] Dorit S. Hochbaum and Asaf Levin.
Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms.
Discrete Optimization, 3(4):327–340, 2006.