Effiziente Algorithmen (SS2022)

Chapter 3
Matchings

Walter Unger

Lehrstuhl für Informatik 1

06.05.2022 17:13:43

Contents I

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.
  - Bestimme größtes Matching mit Kantengewichten.

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.
  - Bestimme größtes Matching mit Kantengewichten.
  - Bestimme günstigstes Matching (d.h. Kosten auf den Kanten) unter allen maximum Matchings.

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.
  - Bestimme größtes Matching mit Kantengewichten.
  - Bestimme günstigstes Matching (d.h. Kosten auf den Kanten) unter allen maximum Matchings.
- Im Folgenden: $G = (V, E)$ zusammenhängend, $|V| = n$, $|E| = m$

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.
  - Bestimme größtes Matching mit Kantengewichten.
  - Bestimme günstigstes Matching (d.h. Kosten auf den Kanten) unter allen maximum Matchings.
- Im Folgenden: $G = (V, E)$ zusammenhängend, $|V| = n$, $|E| = m$
- Im Folgenden: $G = (V, W, E)$ bipartit und zusammenhängend, $|V \cup W| = n$, $|E| = m$

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.
  - Bestimme größtes Matching mit Kantengewichten.
  - Bestimme günstigstes Matching (d.h. Kosten auf den Kanten) unter allen maximum Matchings.
- Im Folgenden: $G = (V, E)$ zusammenhängend, $|V| = n$, $|E| = m$
- Im Folgenden: $G = (V, W, E)$ bipartit und zusammenhängend, $|V \cup W| = n$, $|E| = m$

Definitionen

$\delta_G(v) = |N_G(v)|$ $N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definitionen

$\delta_G(v) = |N_G(v)|$ $N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definitionen

$\delta_G(v) = |N_G(v)|$ $N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definition ((inklusions-)maximales Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximales Matching:

$$\forall M' : M \subsetneq M' \subset E : M' \text{ ist kein Matching.}$$

Definitionen

$\delta_G(v) = |N_G(v)|\ N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definition ((inklusions-)maximales Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximales Matching:

$$\forall M' : M \subsetneq M' \subset E : M' \text{ ist kein Matching.}$$

Definitionen

$\delta_G(v) = |N_G(v)|$ $N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definition ((inklusions-)maximales Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximales Matching:

$$\forall M' : M \subsetneq M' \subset E : M' \text{ ist kein Matching.}$$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximum Matching:

$$\forall M' \subset E : |M| < |M'| : M' \text{ ist kein Matching.}$$

Definitionen

$\delta_G(v) = |N_G(v)|$ $N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definition ((inklusions-)maximales Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximales Matching:

$$\forall M' : M \subsetneq M' \subset E : M' \text{ ist kein Matching.}$$

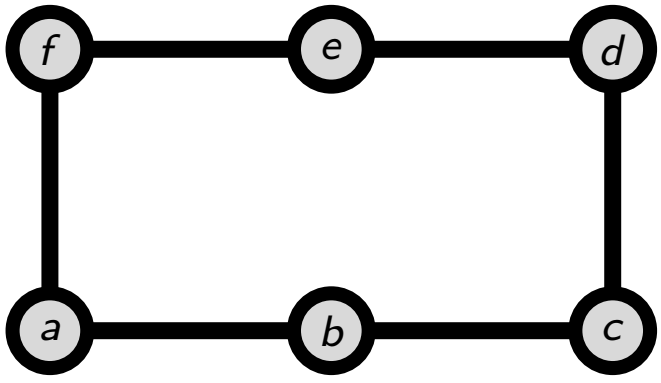
Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximum Matching:

$$\forall M' \subset E : |M| < |M'| : M' \text{ ist kein Matching.}$$

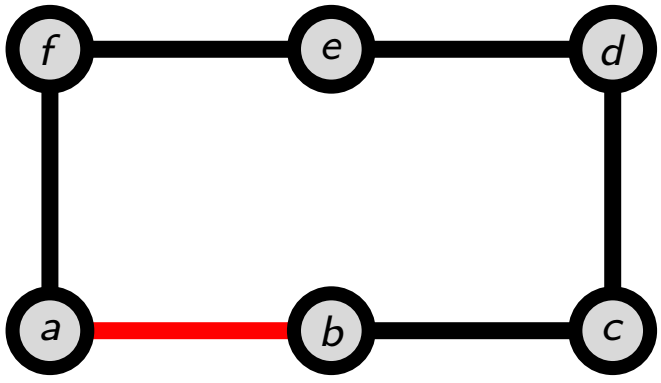
Beispiel

- Betrachte Graph:



Beispiel

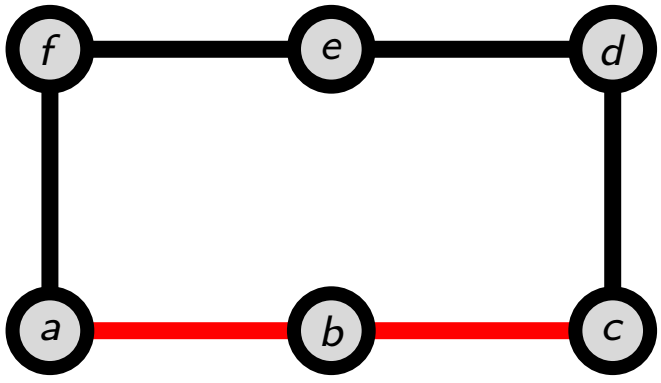
- Betrachte Graph:



- $\{\{a, b\}\}$ Matching

Beispiel

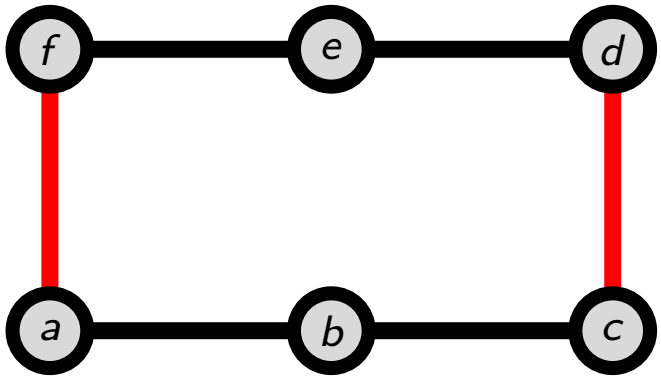
- Betrachte Graph:



- $\{\{a, b\}\}$ Matching
- $\{\{a, b\}, \{b, c\}\}$ kein Matching

Beispiel

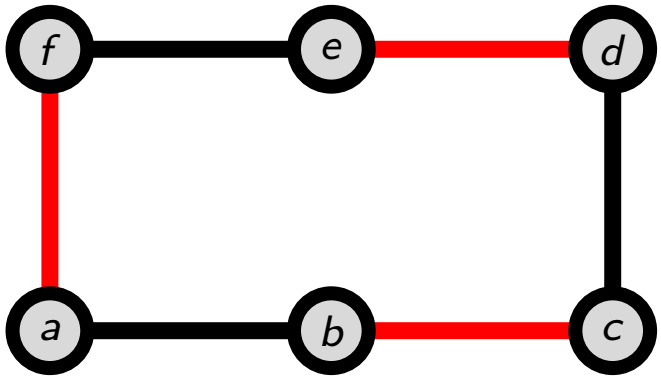
- Betrachte Graph:



- $\{\{a, b\}\}$ Matching
- $\{\{a, b\}, \{b, c\}\}$ kein Matching
- $\{\{a, f\}, \{c, d\}\}$ maximales Matching

Beispiel

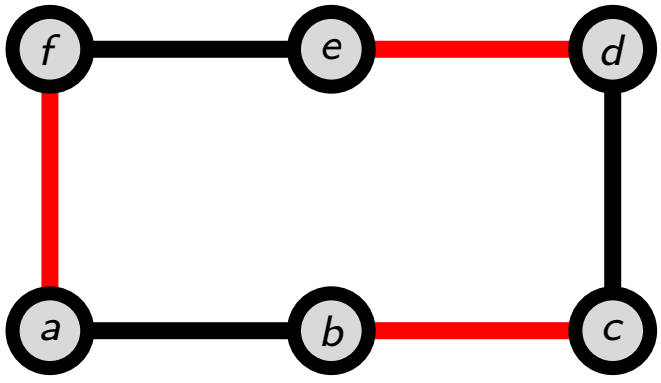
- Betrachte Graph:



- $\{\{a, b\}\}$ Matching
- $\{\{a, b\}, \{b, c\}\}$ kein Matching
- $\{\{a, f\}, \{c, d\}\}$ maximales Matching
- $\{\{a, f\}, \{b, c\}, \{e, d\}\}$ maximum Matching

Beispiel

- Betrachte Graph:



- $\{\{a, b\}\}$ Matching
- $\{\{a, b\}, \{b, c\}\}$ kein Matching
- $\{\{a, f\}, \{c, d\}\}$ maximales Matching
- $\{\{a, f\}, \{b, c\}, \{e, d\}\}$ maximum Matching

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:
   1. Wähle $e \in E$.

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:
   1. Wähle $e \in E$.
   2. Setze $M = M \cup \{e\}$

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:
   1. Wähle $e \in E$.
   2. Setze $M = M \cup \{e\}$
   3. Setze $E = \{e' \in E \mid e' \cap e = \emptyset\}$

Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:
   1. Wähle $e \in E$.
   2. Setze $M = M \cup \{e\}$
   3. Setze $E = \{e' \in E \mid e' \cap e = \emptyset\}$
4. Ausgabe: M.

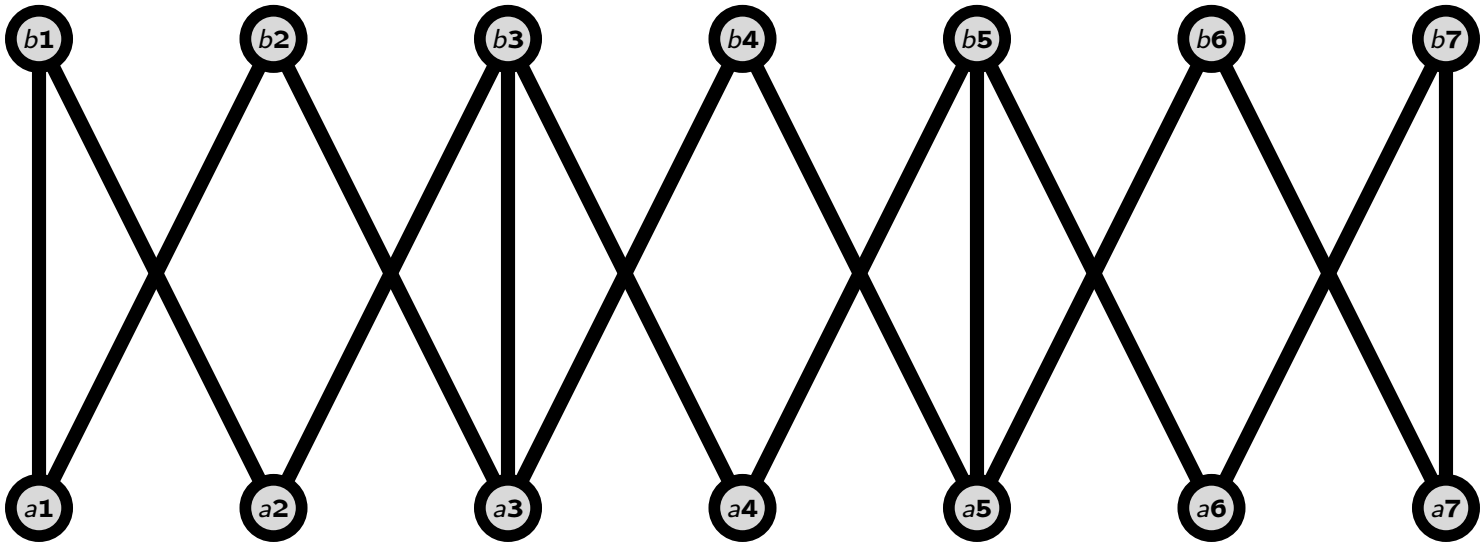
Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:
   1. Wähle $e \in E$.
   2. Setze $M = M \cup \{e\}$
   3. Setze $E = \{e' \in E \mid e' \cap e = \emptyset\}$
4. Ausgabe: M.

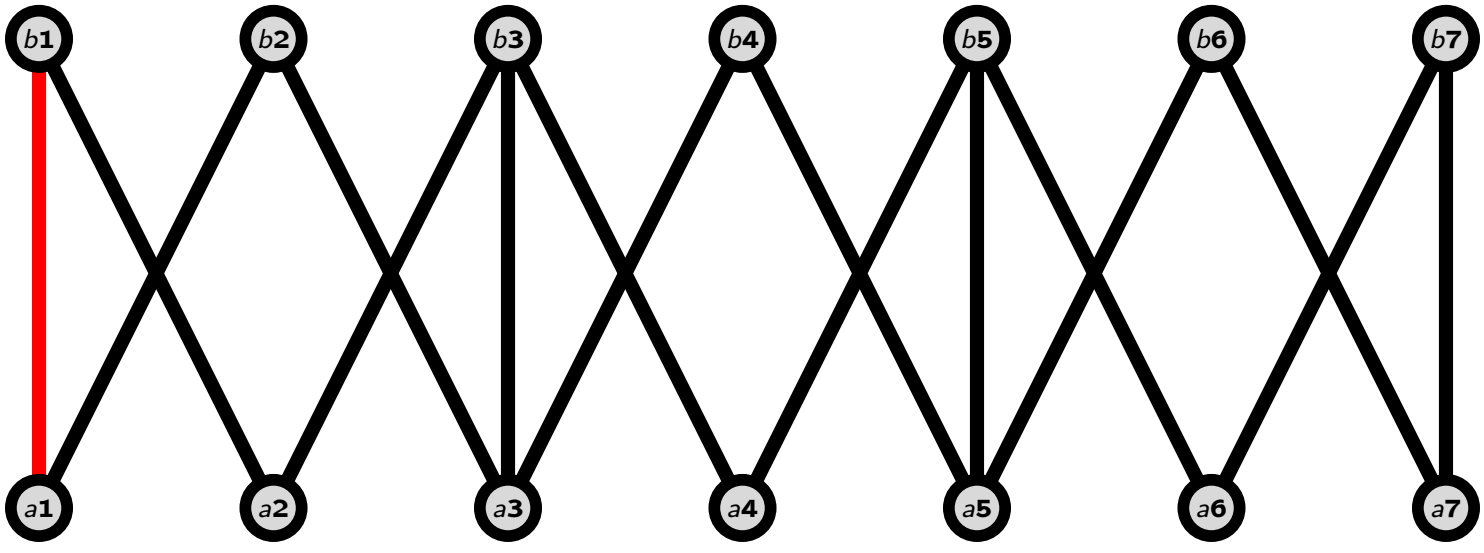
Beispiel

Idee: Greedy Algorithmus:



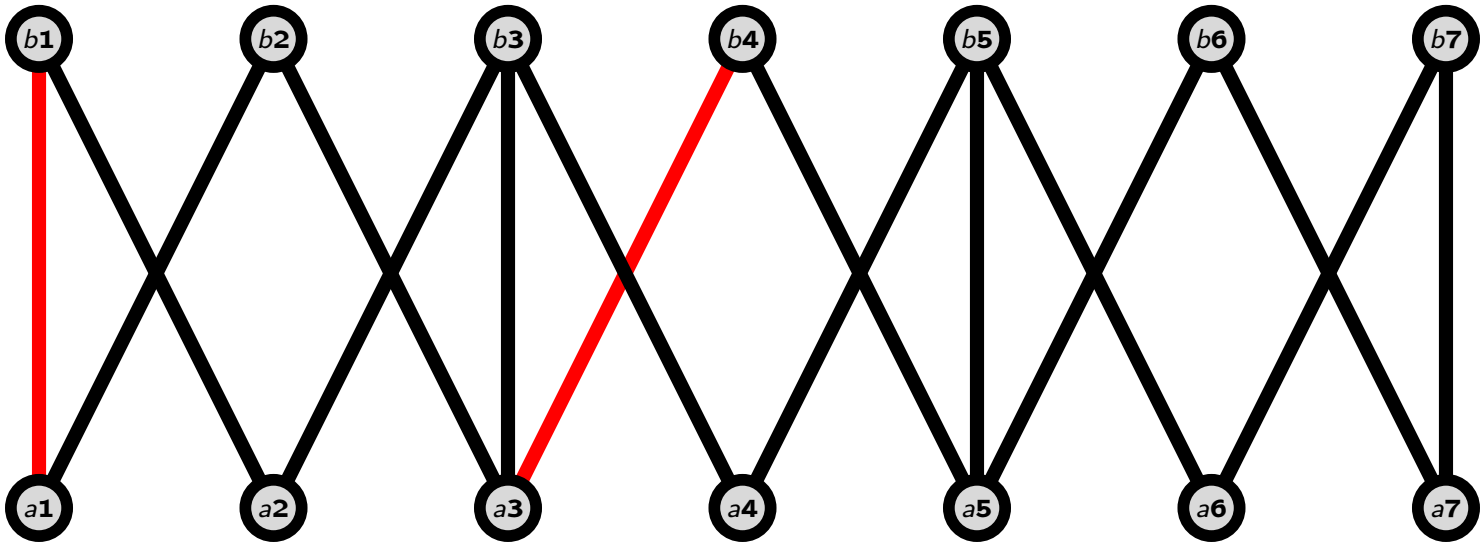
Beispiel

Idee: Greedy Algorithmus:



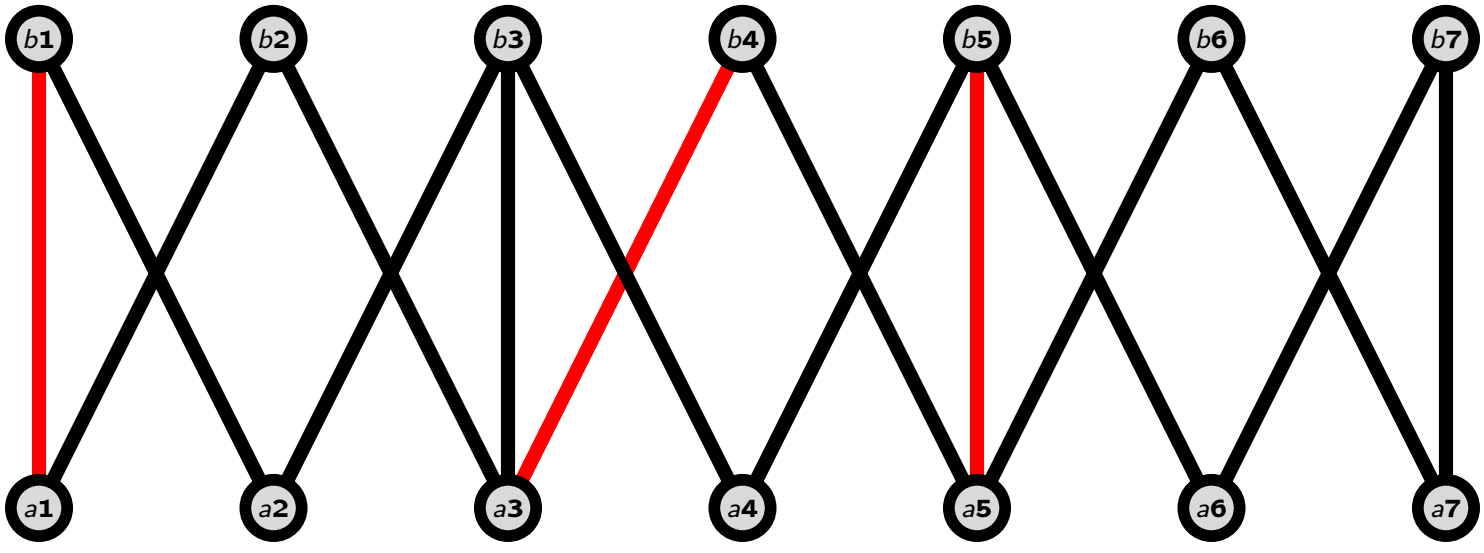
Beispiel

Idee: Greedy Algorithmus:



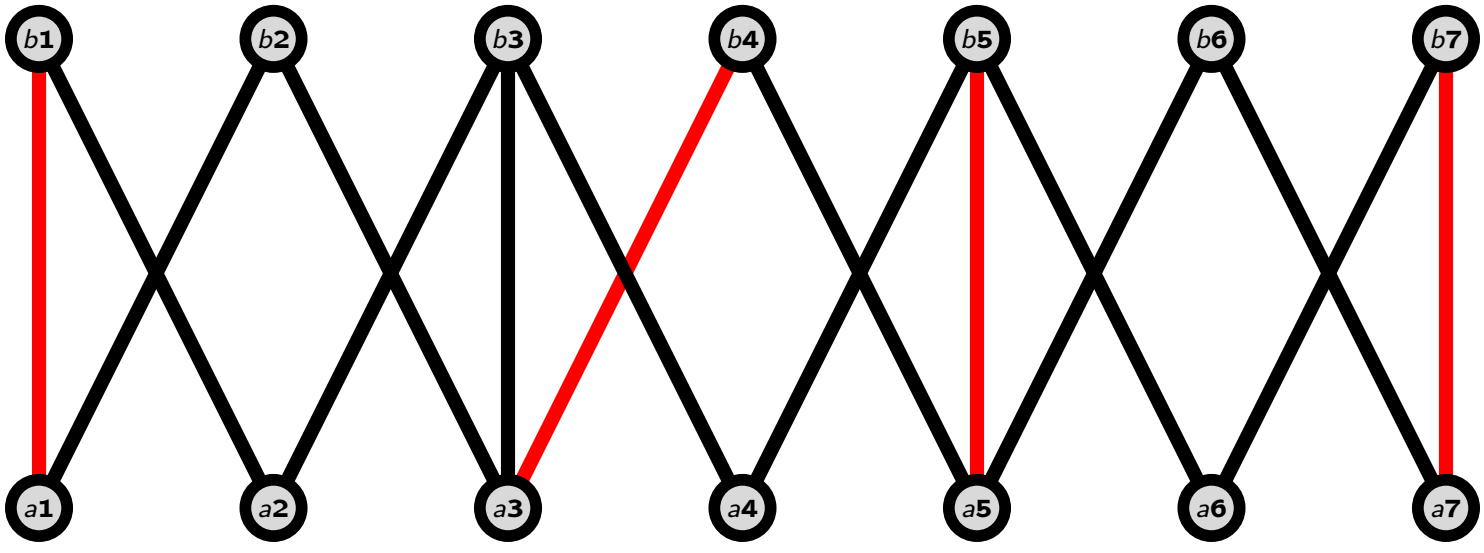
Beispiel

Idee: Greedy Algorithmus:



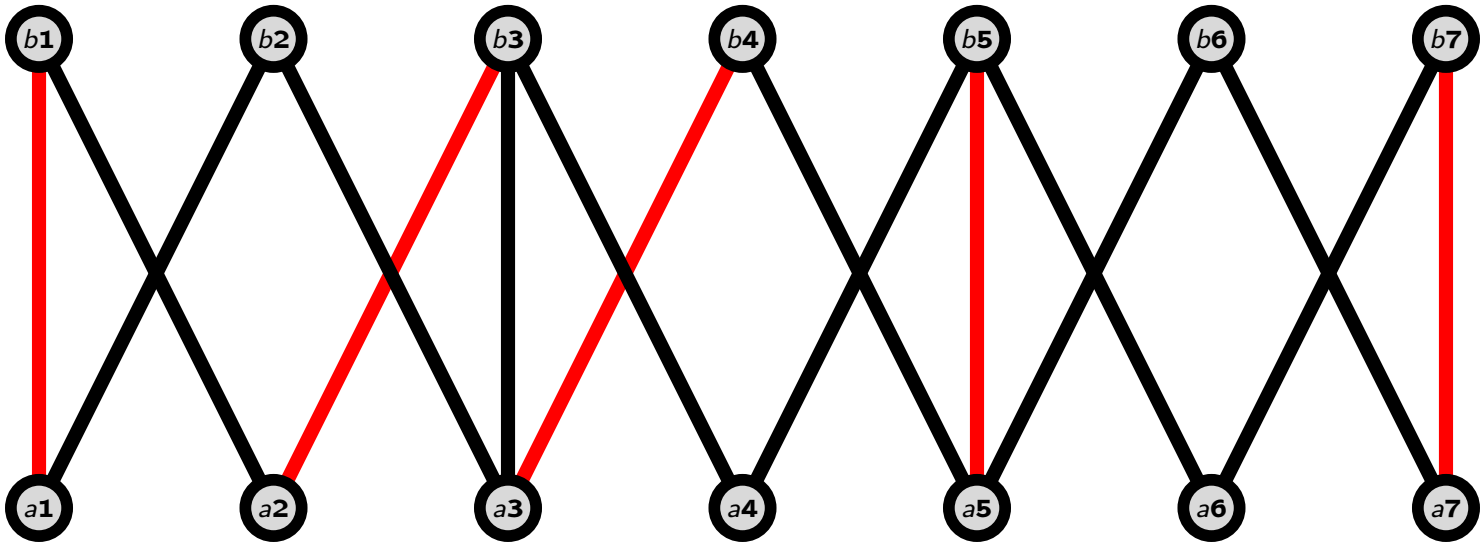
Beispiel

Idee: Greedy Algorithmus:



Beispiel

Idee: Greedy Algorithmus:



Probleme

Definition (Bipartites Matchingproblem)

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Definition (Bipartites kostenminimales Matchingproblem)

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Definition (Bipartites kostenminimales Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph und Kostenfunktion $l : E \mapsto \mathbb{N}$.

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Definition (Bipartites kostenminimales Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph und Kostenfunktion $l : E \mapsto \mathbb{N}$.

Gesucht: Kostenminimales maximum Matching auf G.

Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Definition (Bipartites kostenminimales Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph und Kostenfunktion $l : E \mapsto \mathbb{N}$.

Gesucht: Kostenminimales maximum Matching auf G.

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.
- Modelliere das durch Fluss, erweitere Graph:

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.
- Modelliere das durch Fluss, erweitere Graph:
  - Füge Quelle s ein und verbinde s zu jedem Knoten aus V.

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.
- Modelliere das durch Fluss, erweitere Graph:
  - Füge Quelle s ein und verbinde s zu jedem Knoten aus V.
  - Füge Senke t ein und verbinde jeden Knoten aus W zu t.

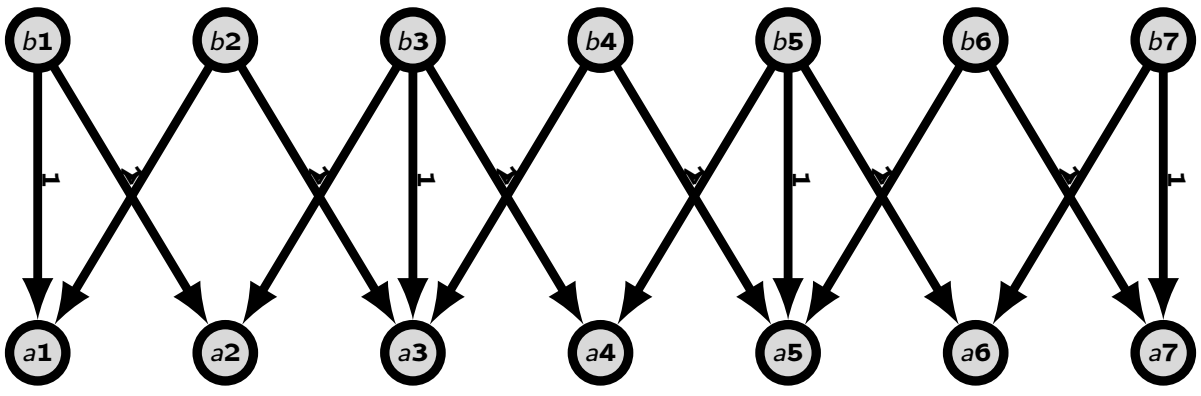
Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.
- Modelliere das durch Fluss, erweitere Graph:
  - Füge Quelle s ein und verbinde s zu jedem Knoten aus V.
  - Füge Senke t ein und verbinde jeden Knoten aus W zu t.
  - Diese neuen Kanten können eine Einheit transportieren.

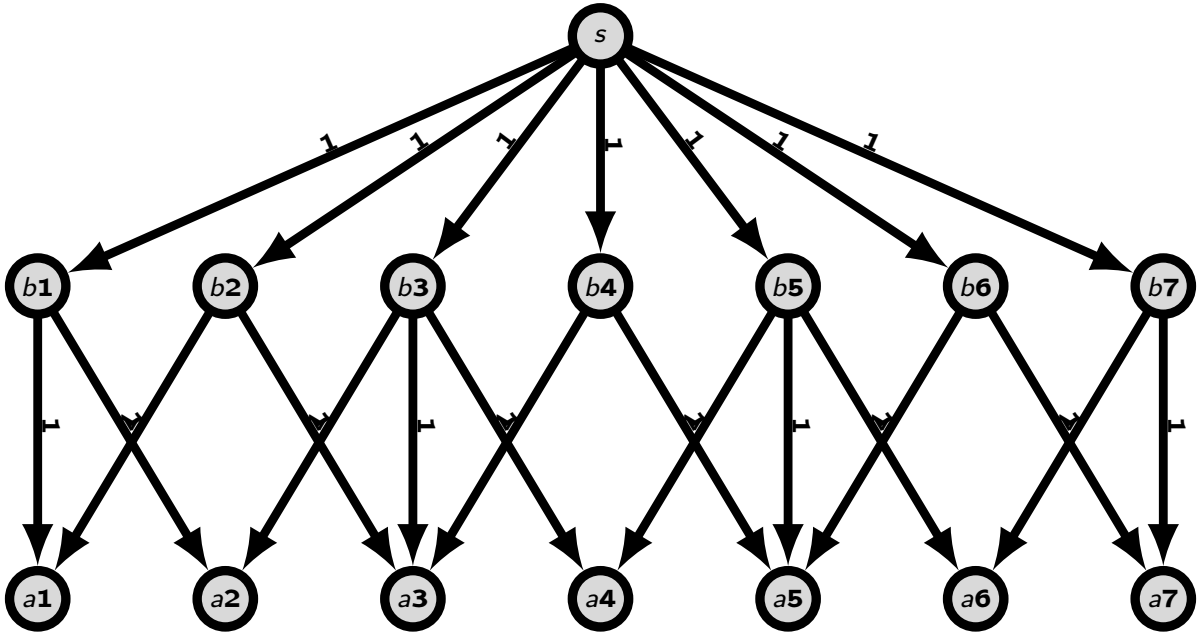
Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.
- Modelliere das durch Fluss, erweitere Graph:
  - Füge Quelle s ein und verbinde s zu jedem Knoten aus V.
  - Füge Senke t ein und verbinde jeden Knoten aus W zu t.
  - Diese neuen Kanten können eine Einheit transportieren.

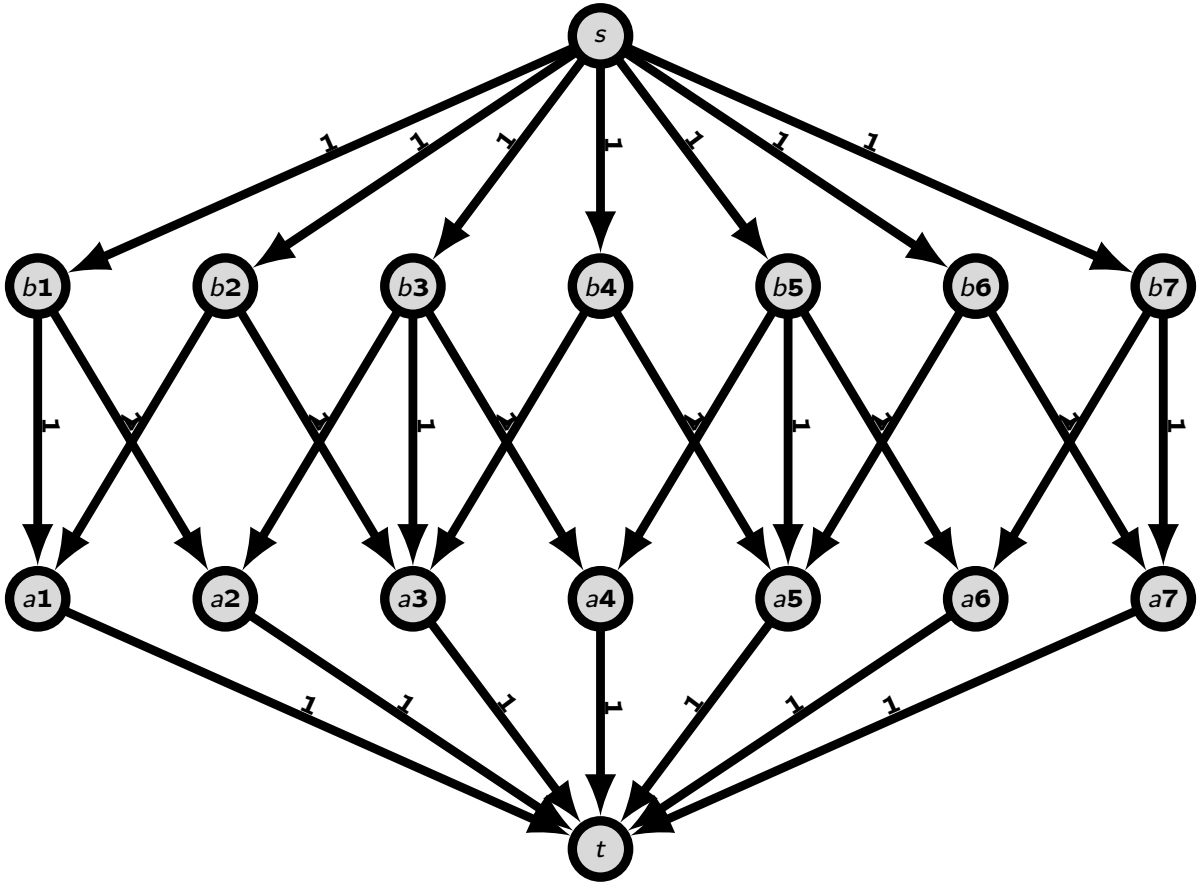
Bipartites Matching und Flüsse



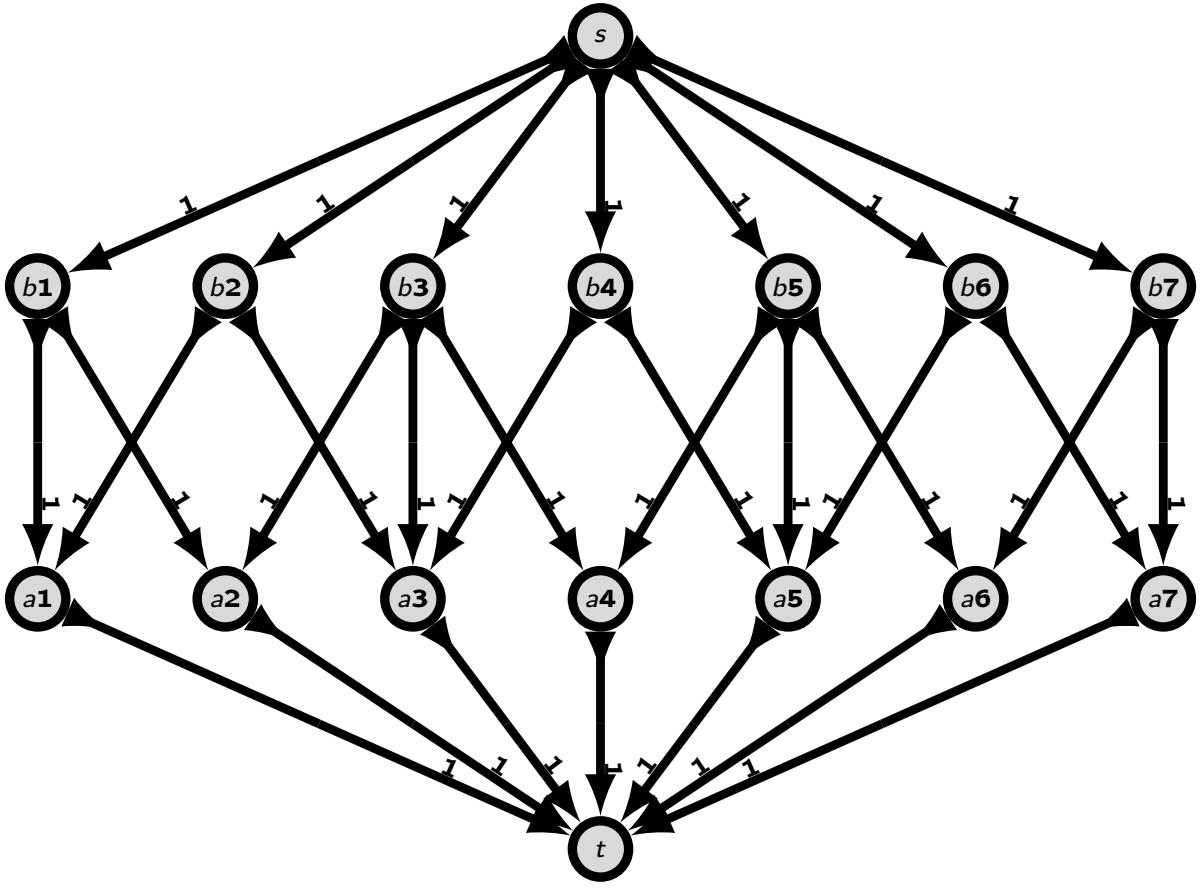
Bipartites Matching und Flüsse



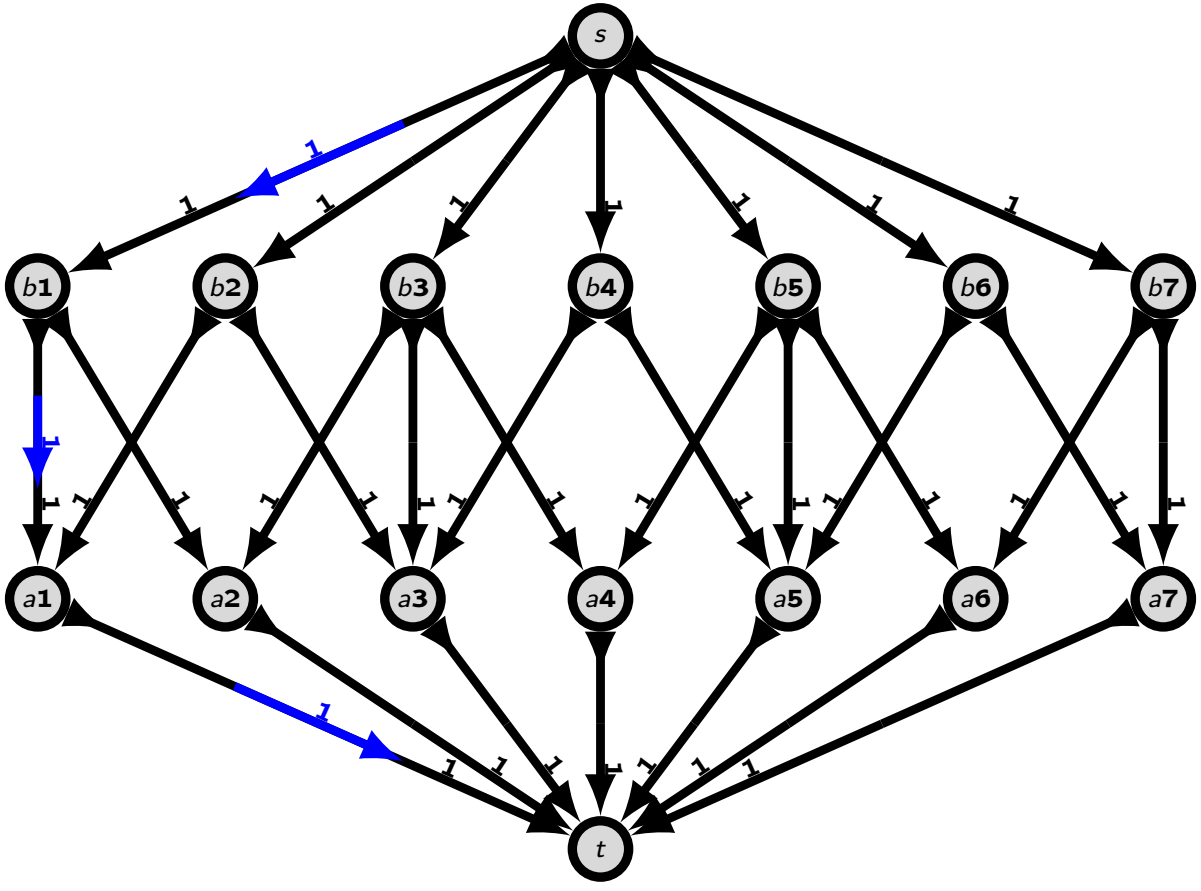
Bipartites Matching und Flüsse



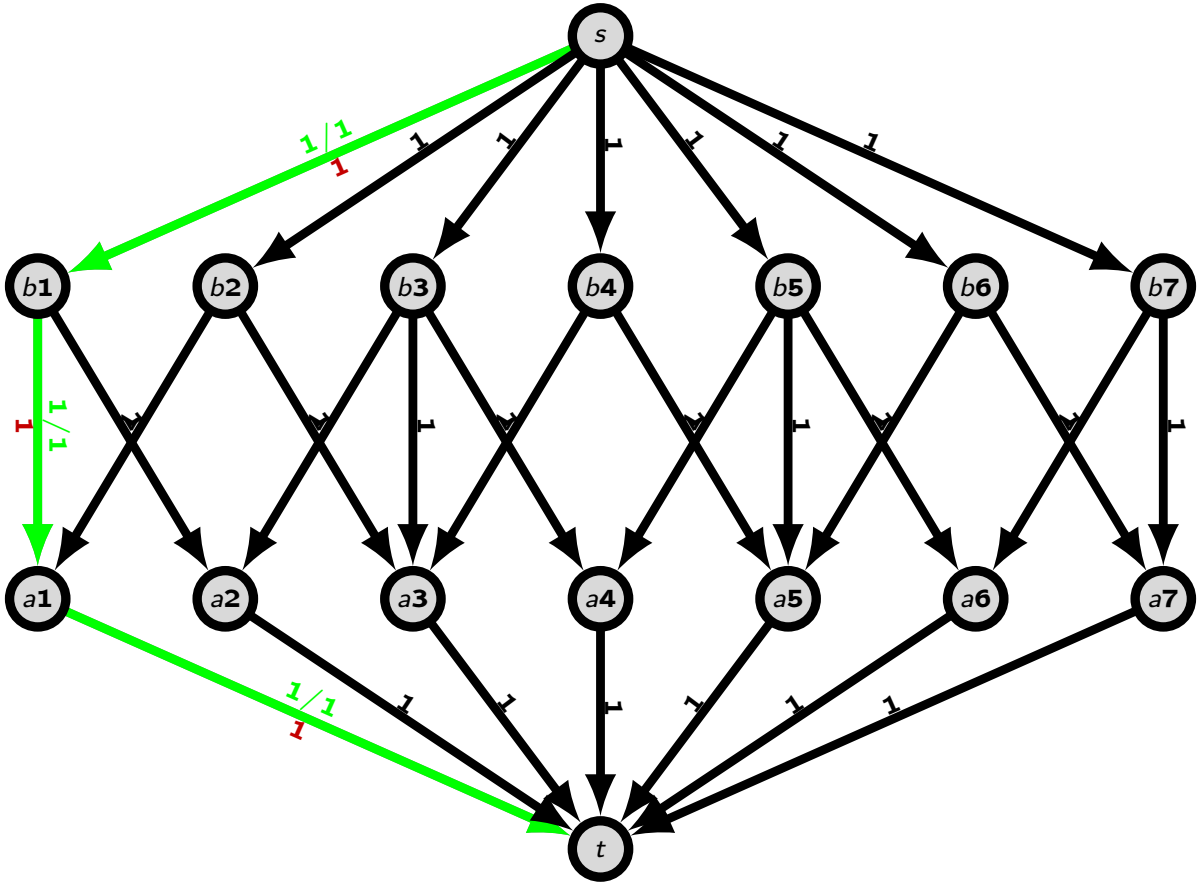
Bipartites Matching und Flüsse



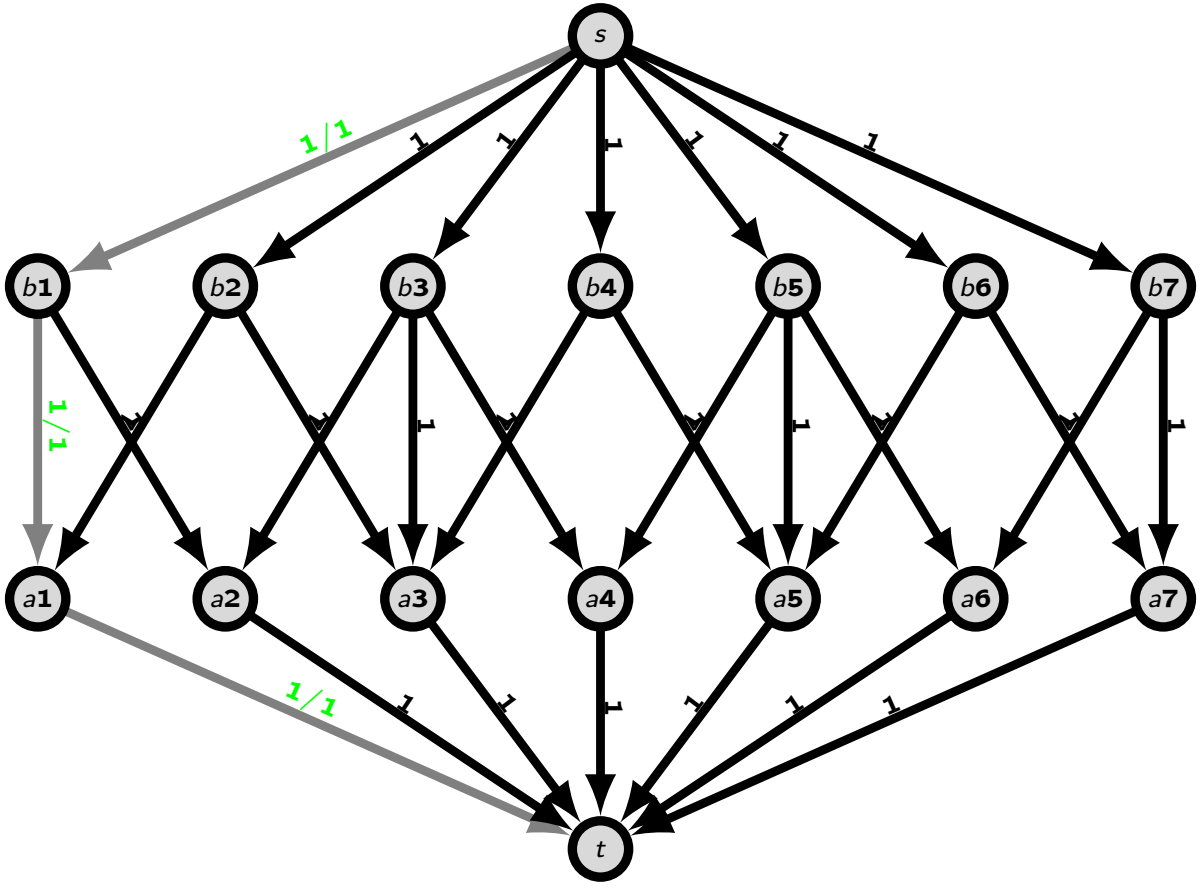
Bipartites Matching und Flüsse



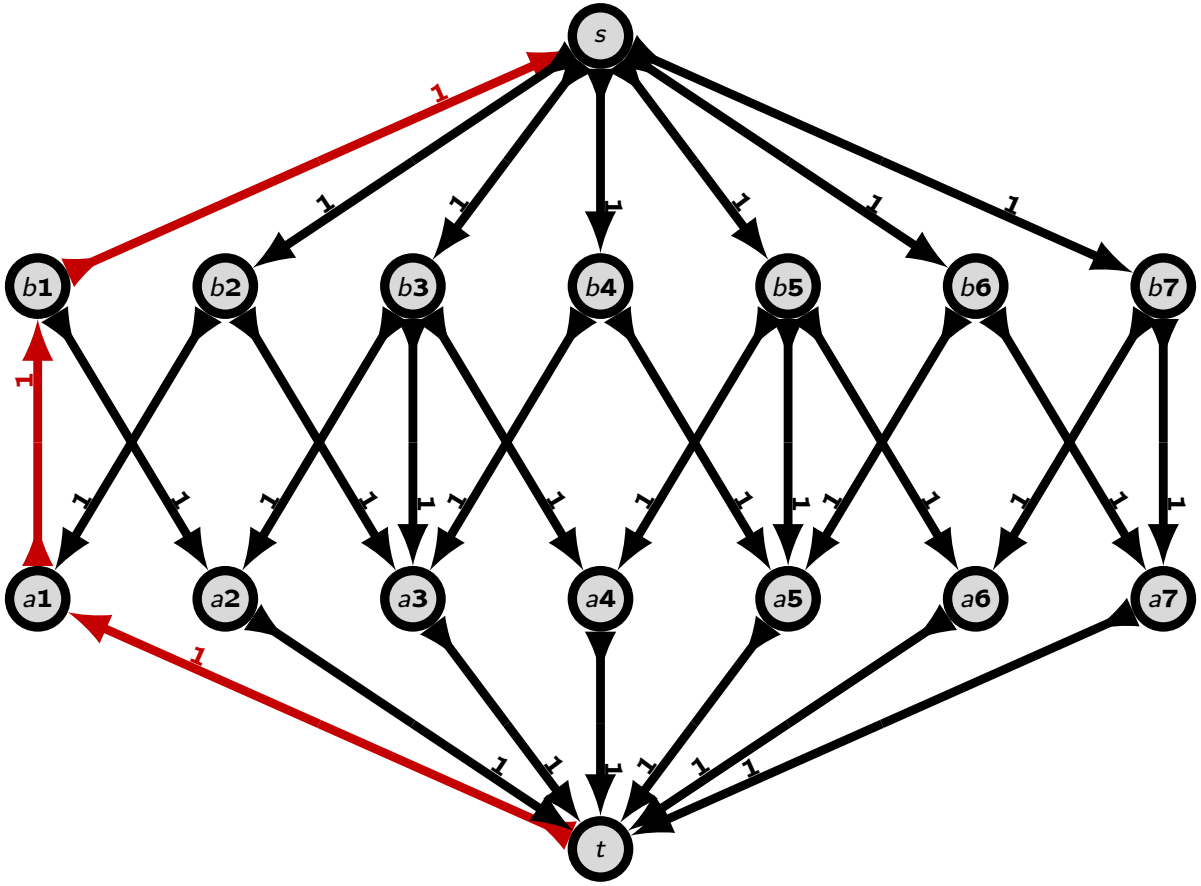
Bipartites Matching und Flüsse



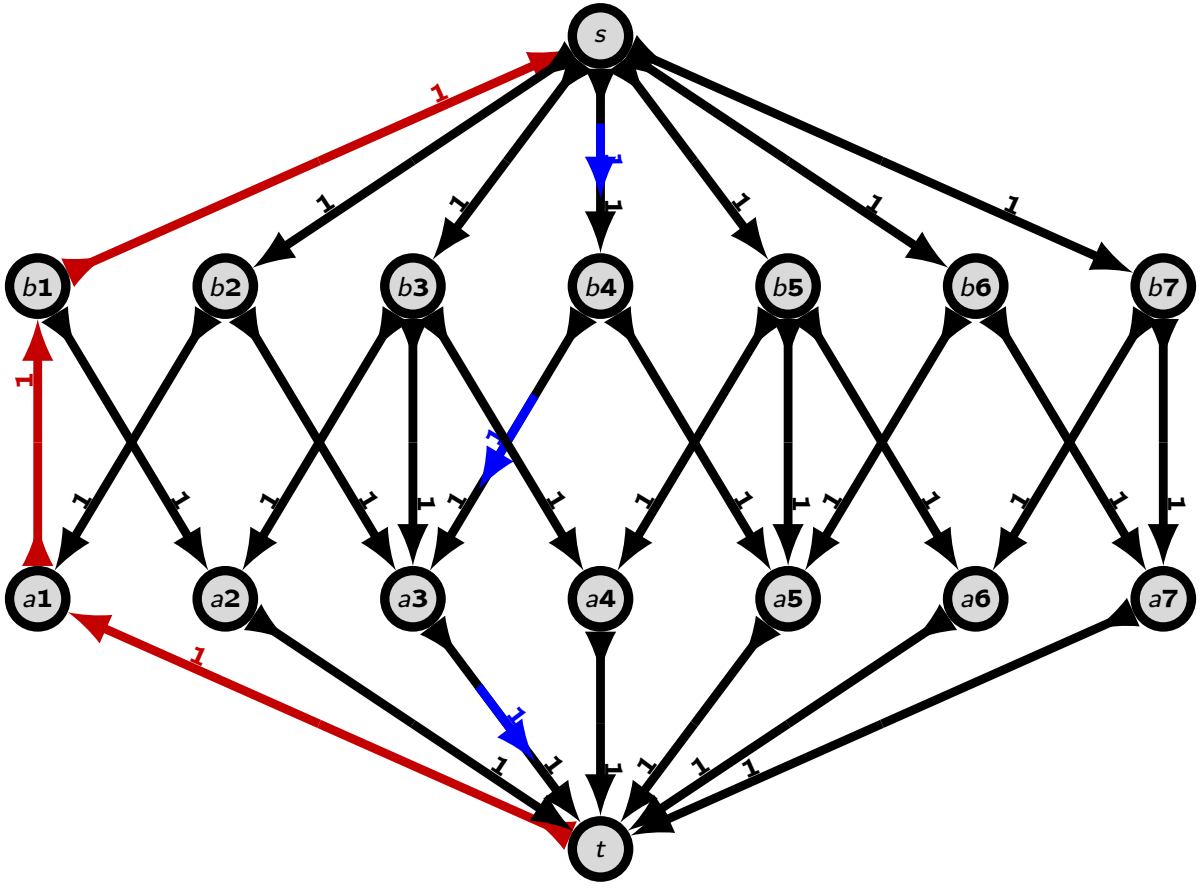
Bipartites Matching und Flüsse



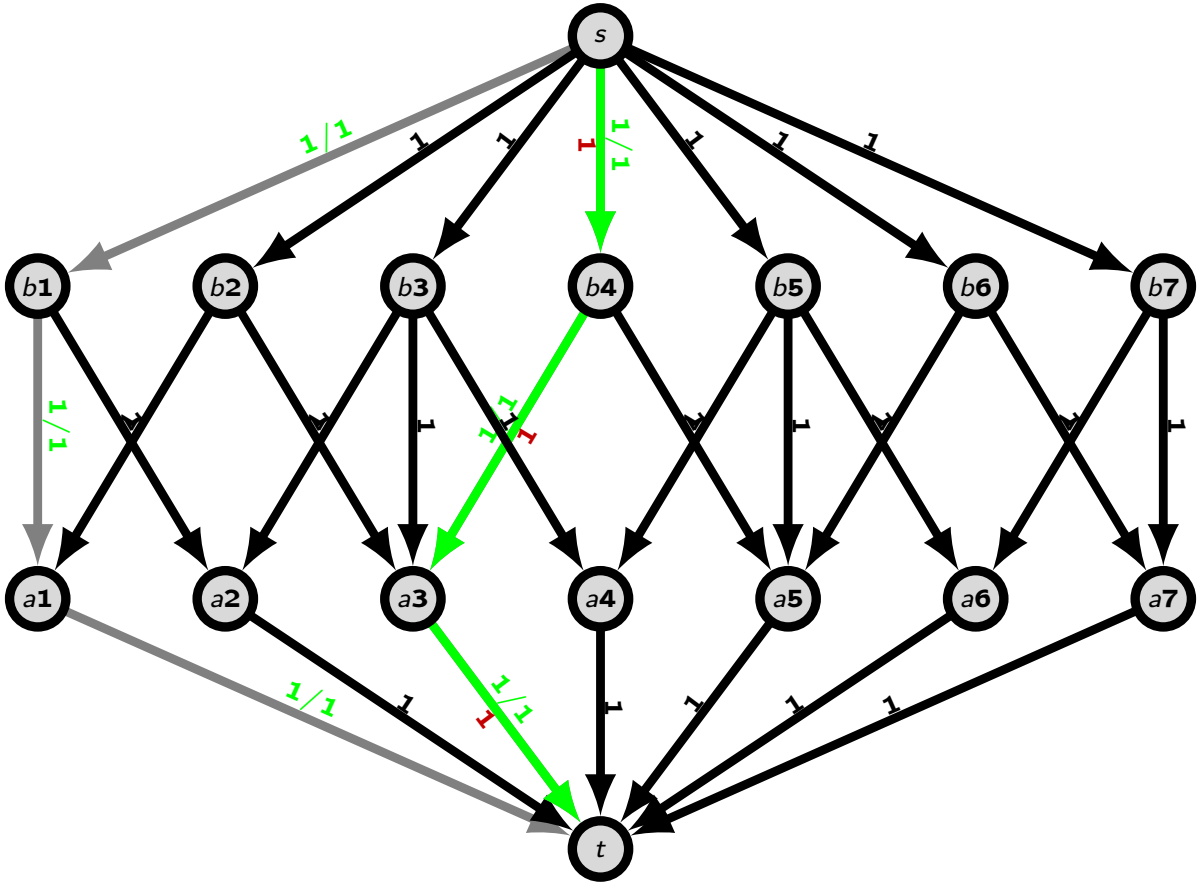
Bipartites Matching und Flüsse



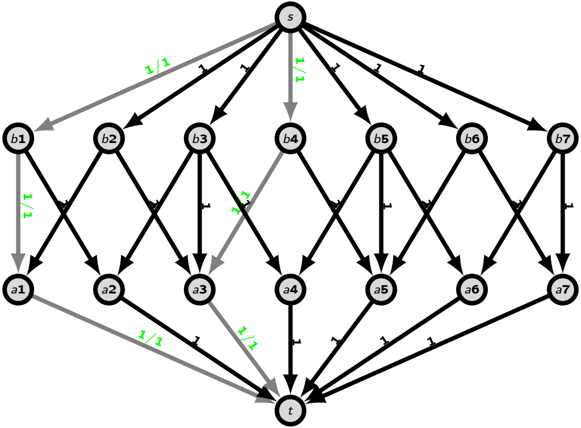
Bipartites Matching und Flüsse



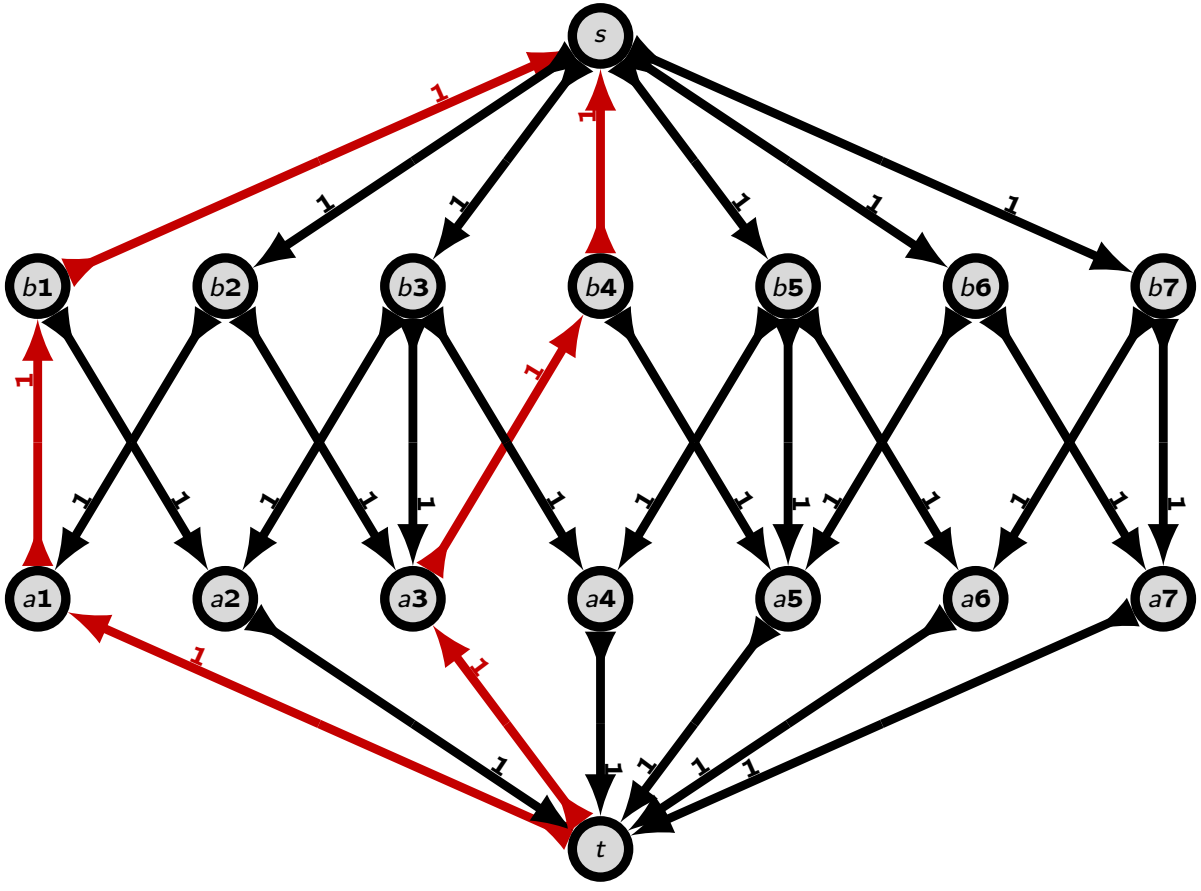
Bipartites Matching und Flüsse



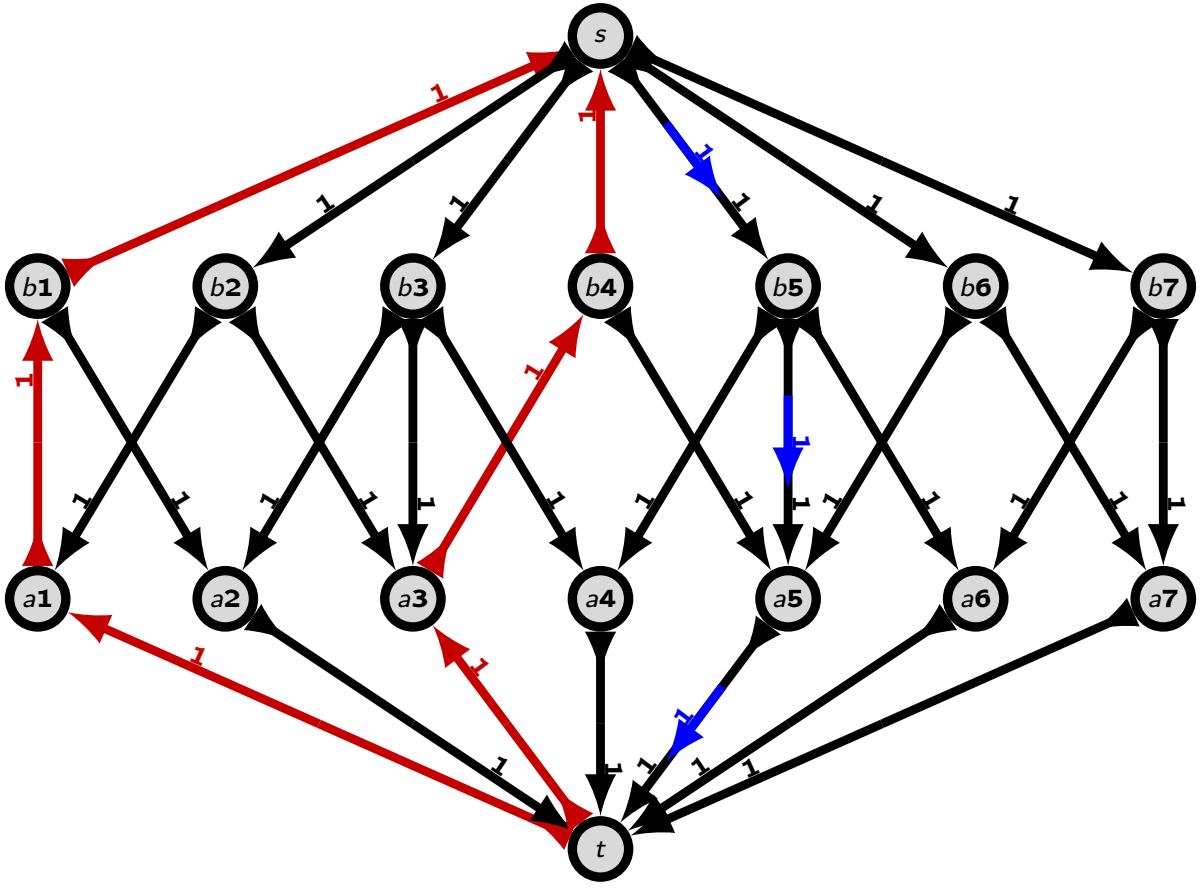
Bipartites Matching und Flüsse



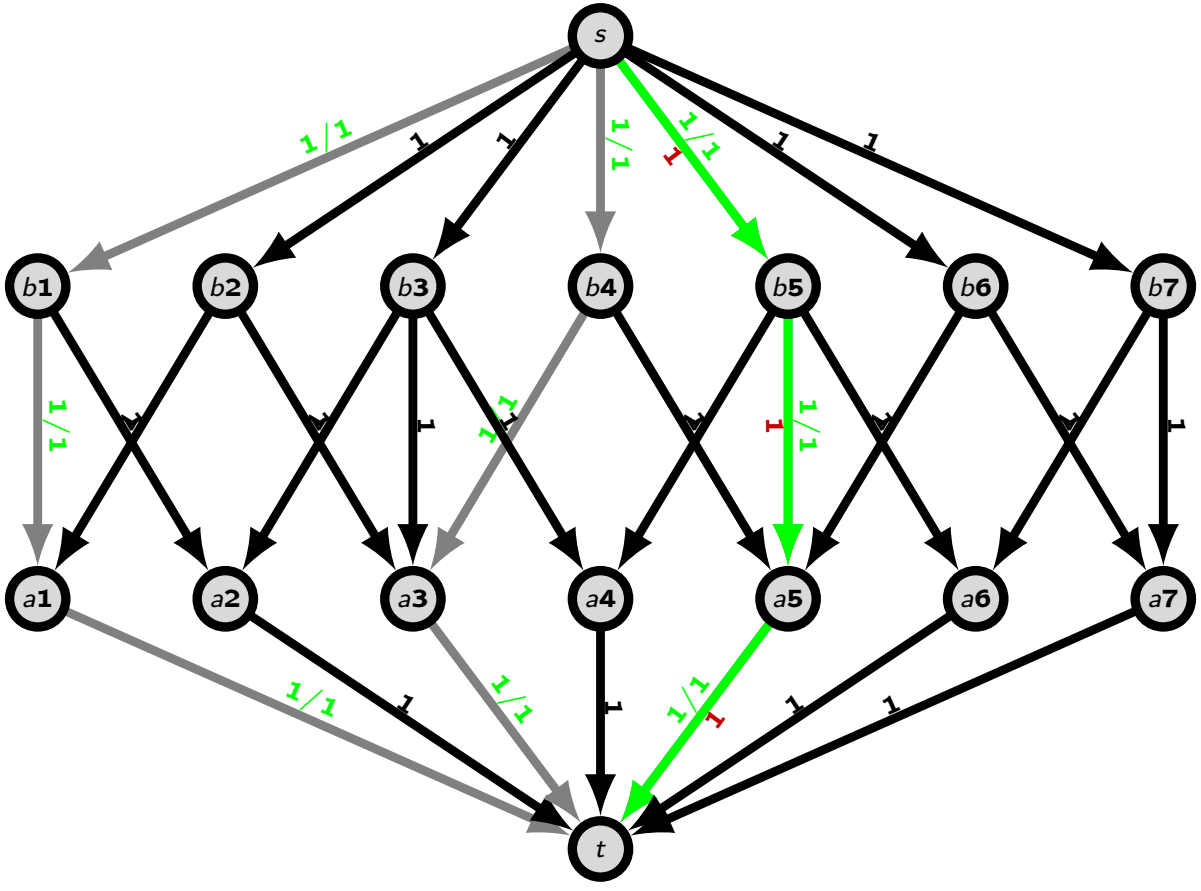
Bipartites Matching und Flüsse



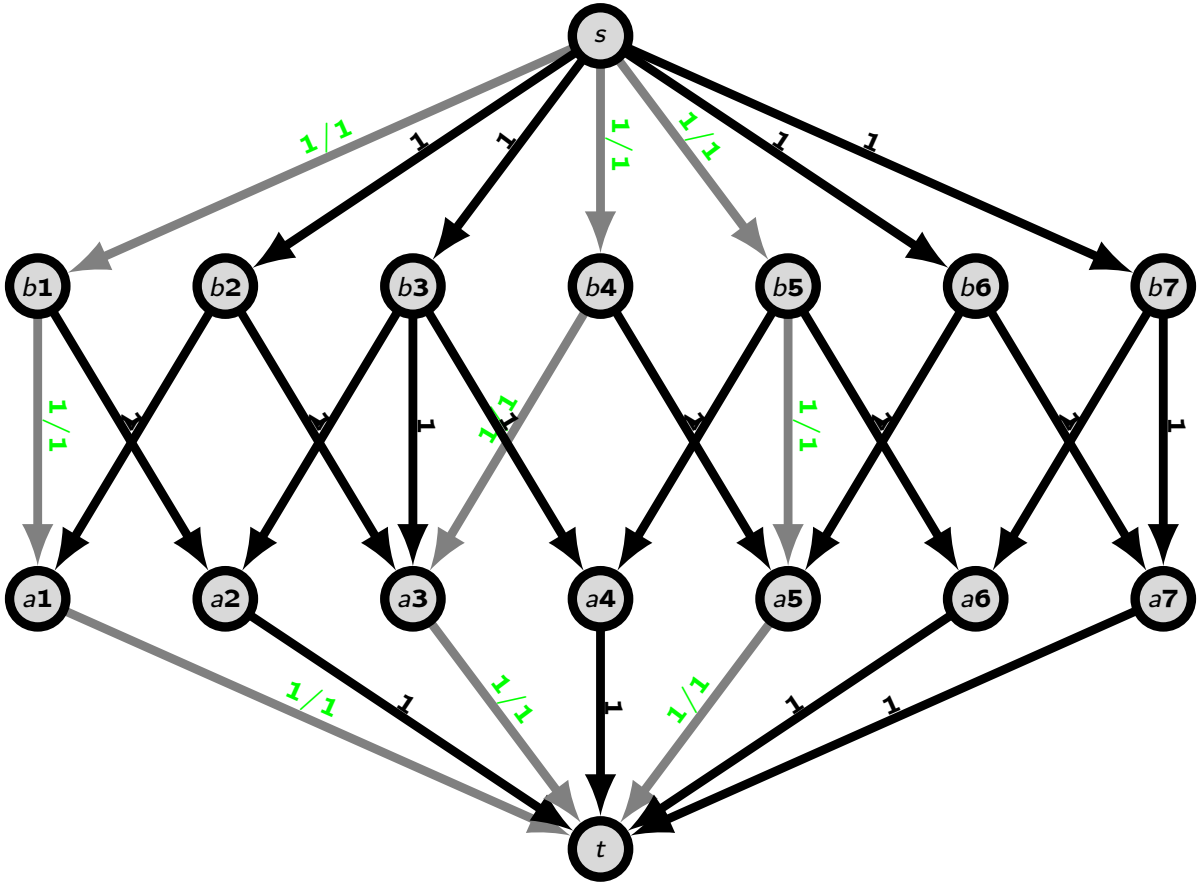
Bipartites Matching und Flüsse



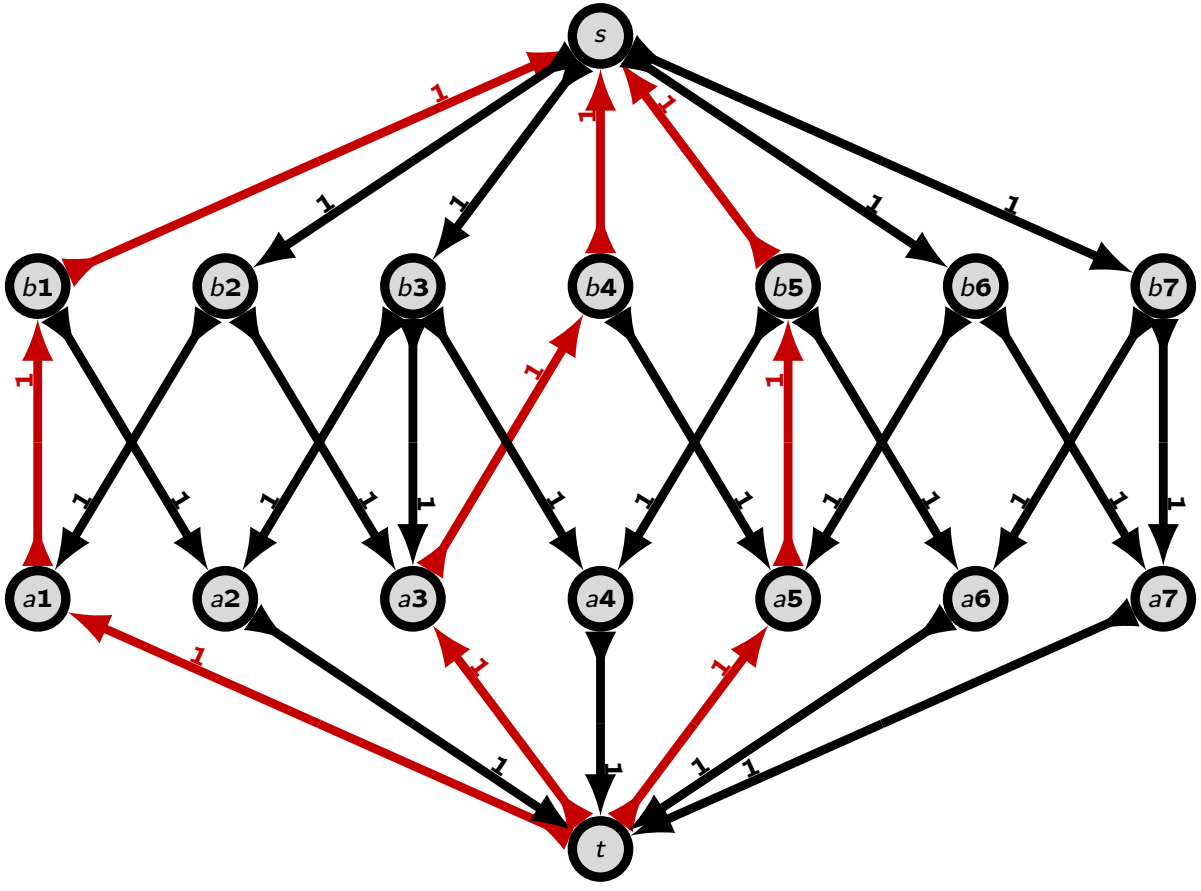
Bipartites Matching und Flüsse



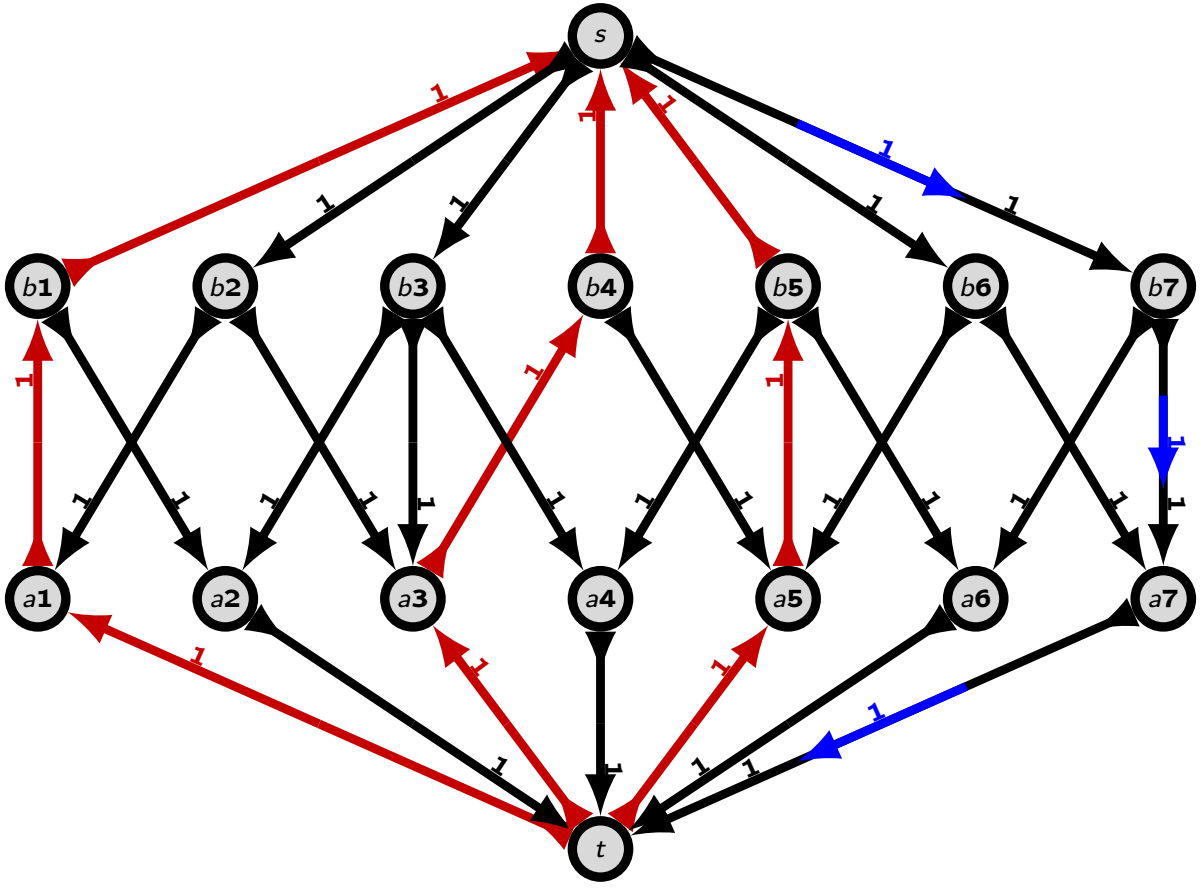
Bipartites Matching und Flüsse



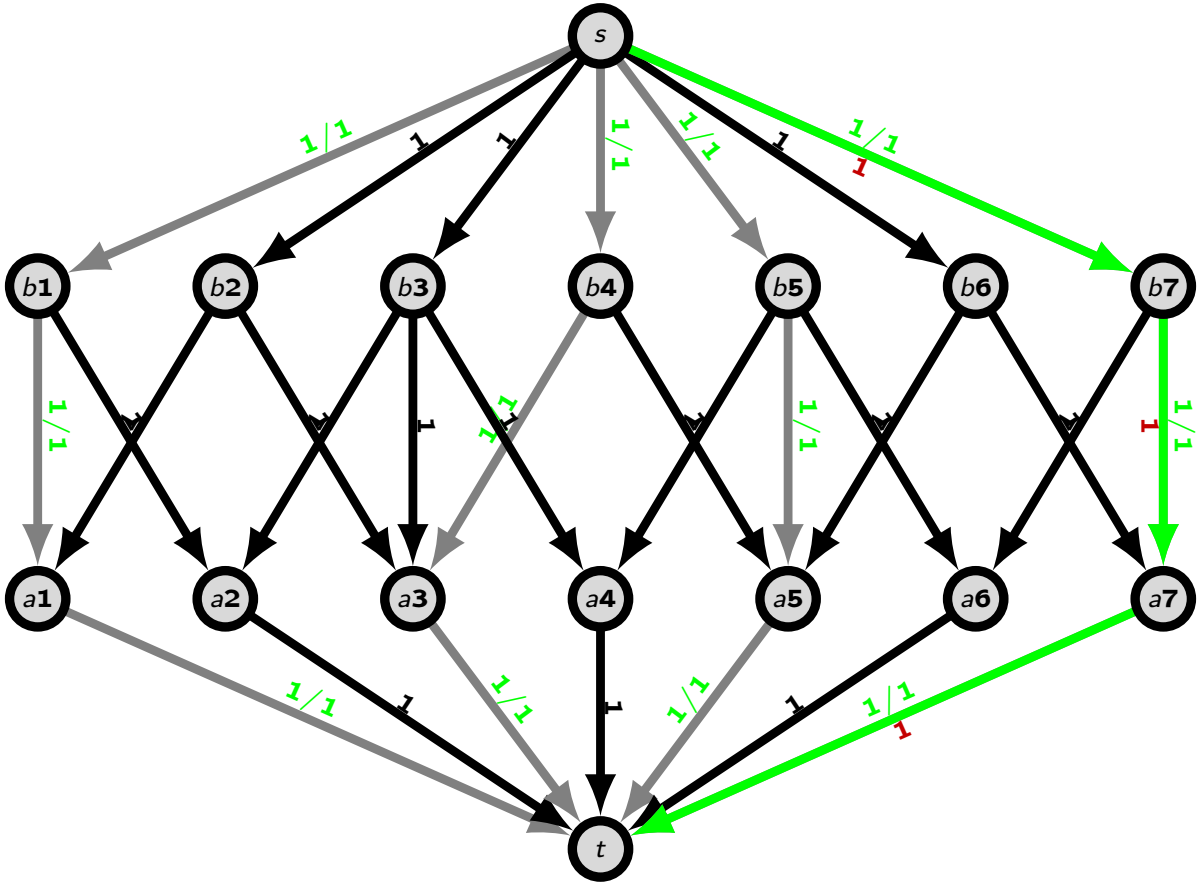
Bipartites Matching und Flüsse



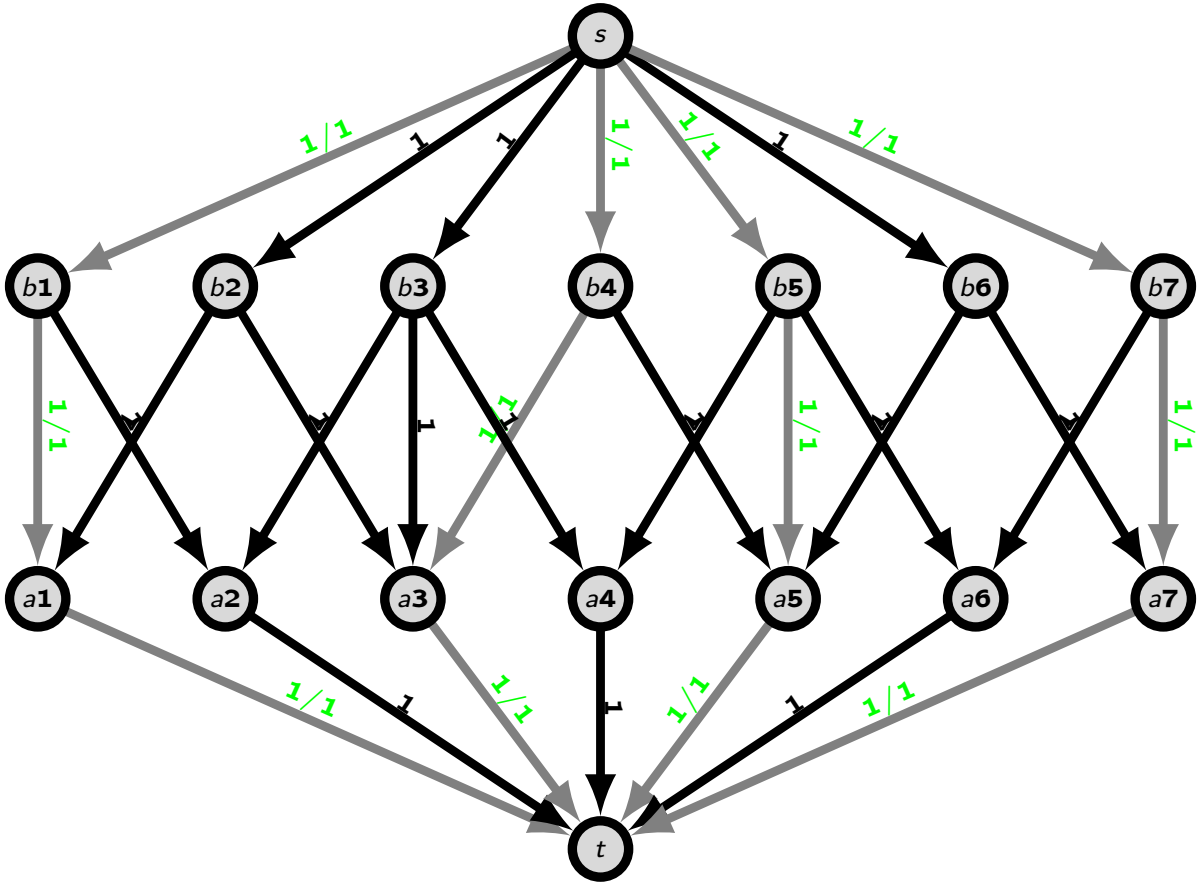
Bipartites Matching und Flüsse



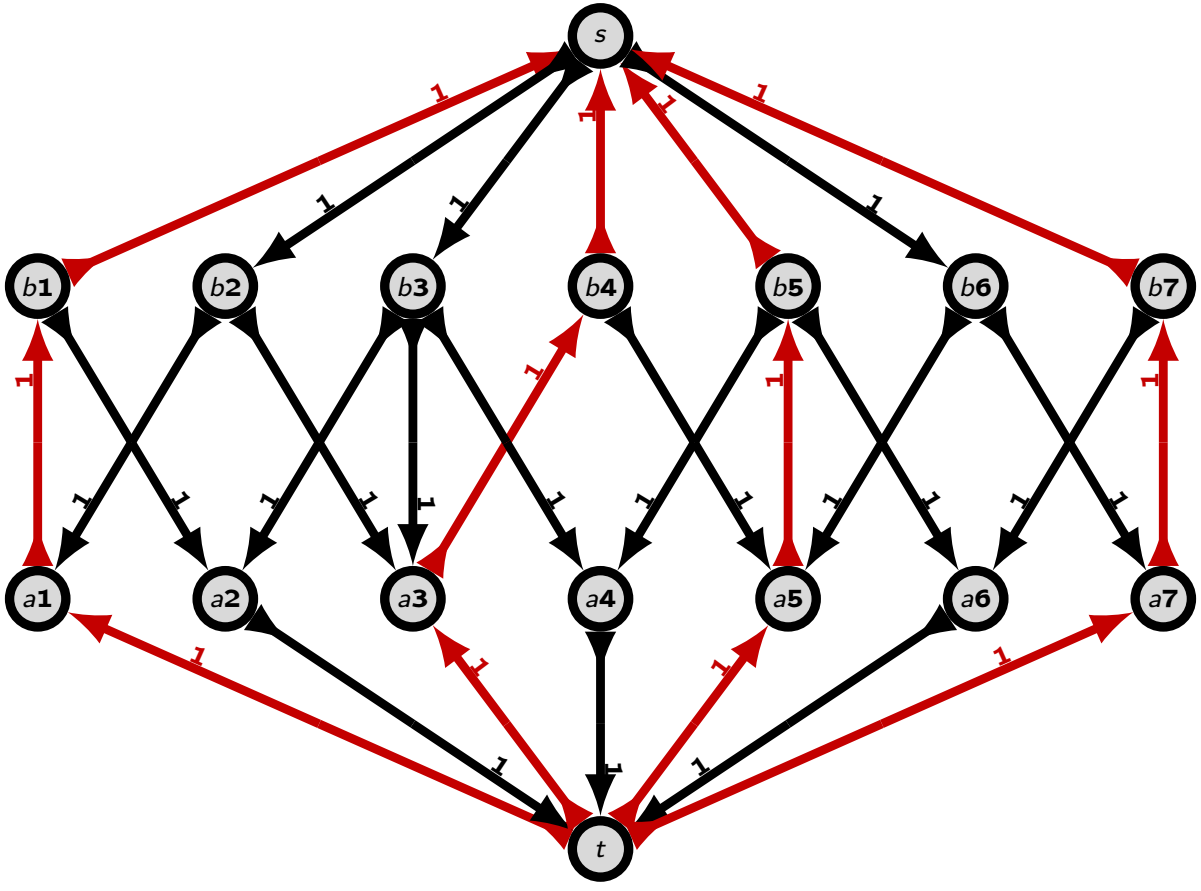
Bipartites Matching und Flüsse



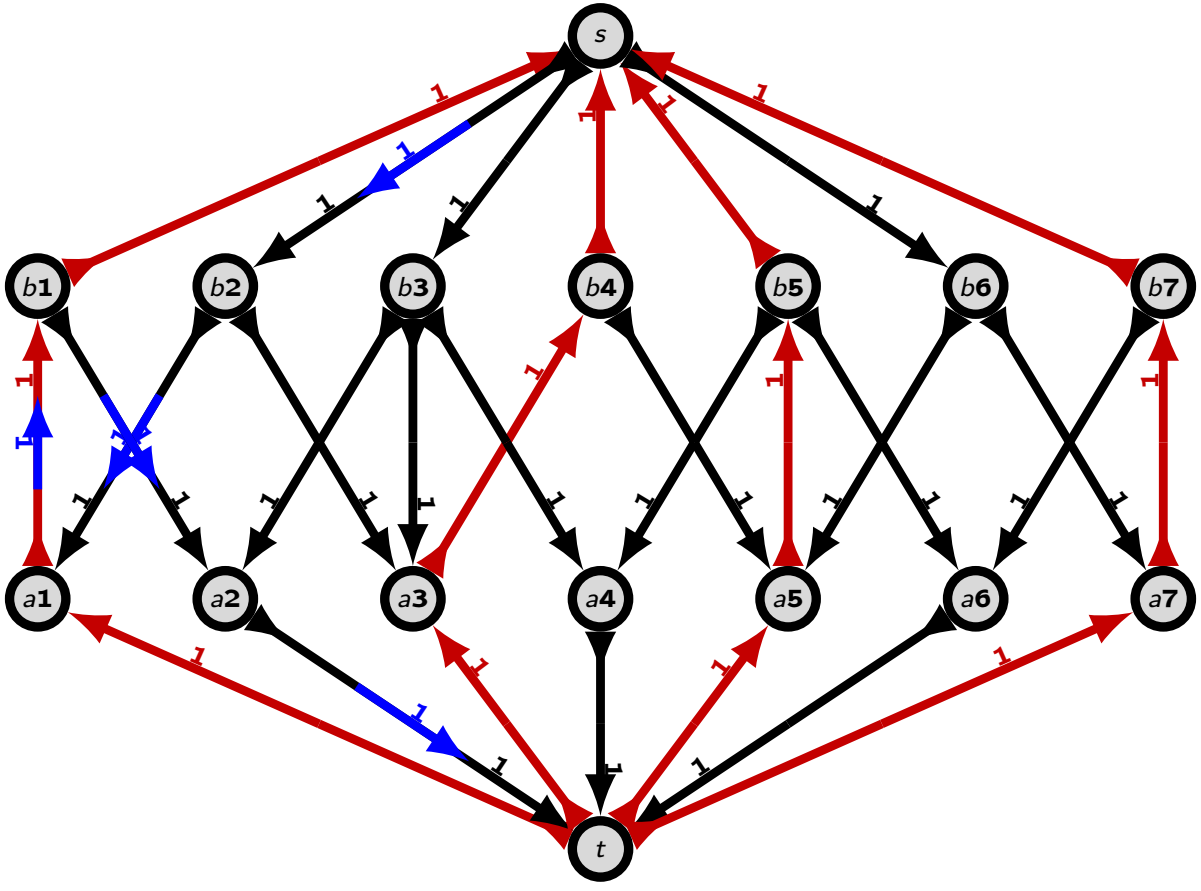
Bipartites Matching und Flüsse



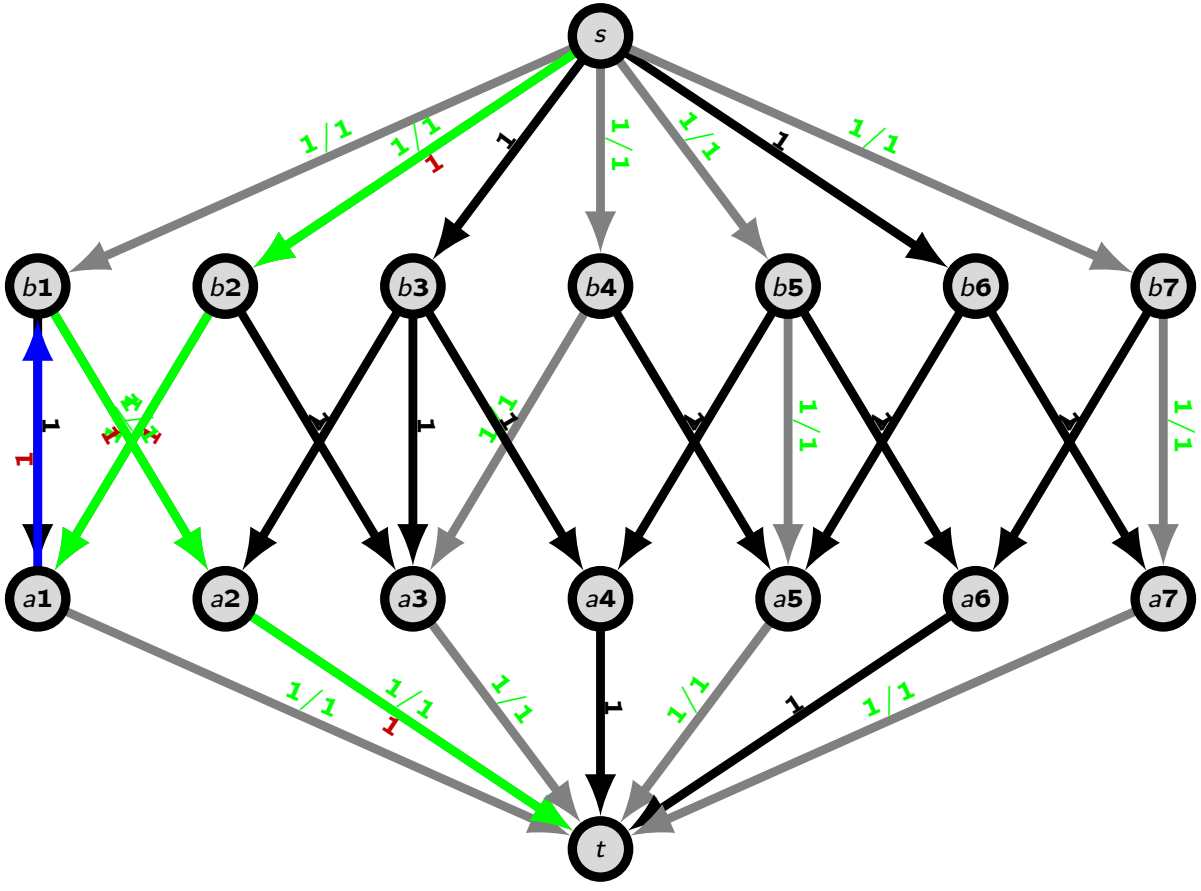
Bipartites Matching und Flüsse



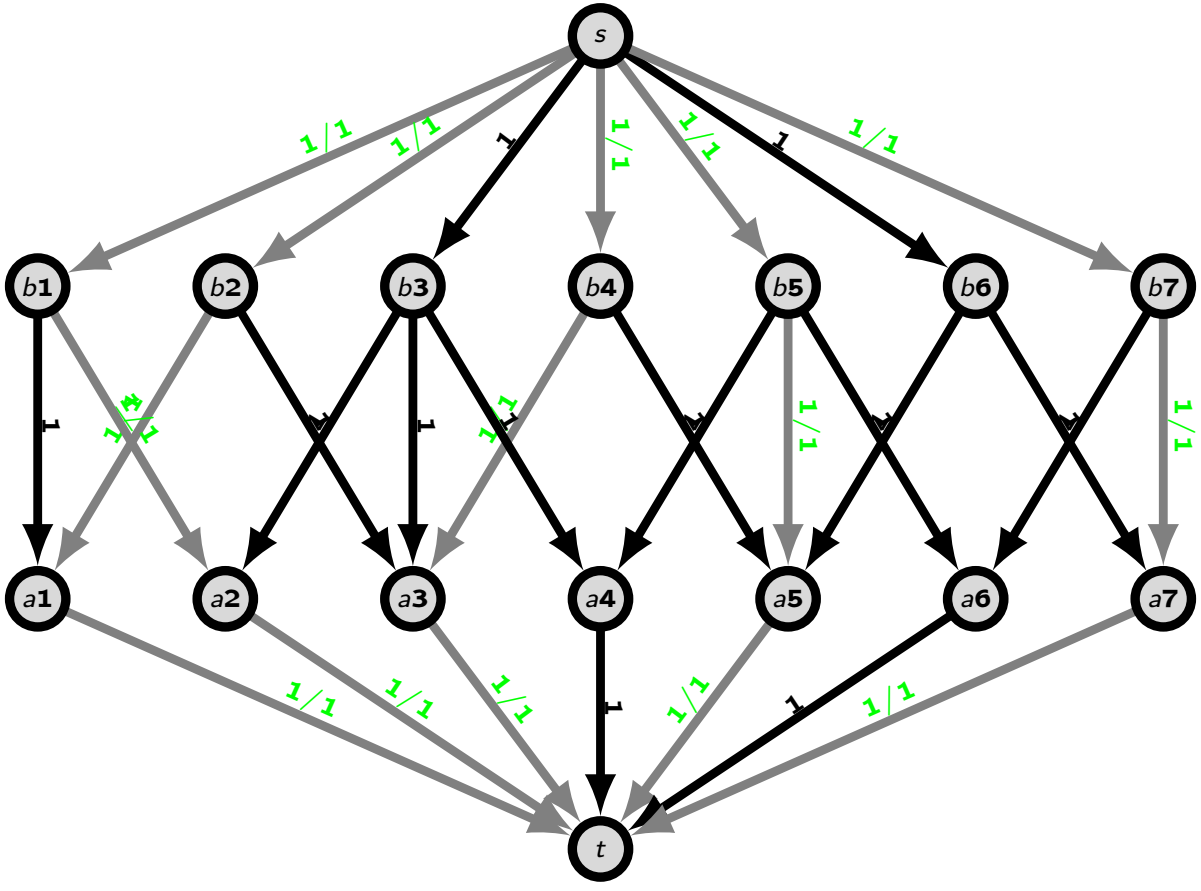
Bipartites Matching und Flüsse



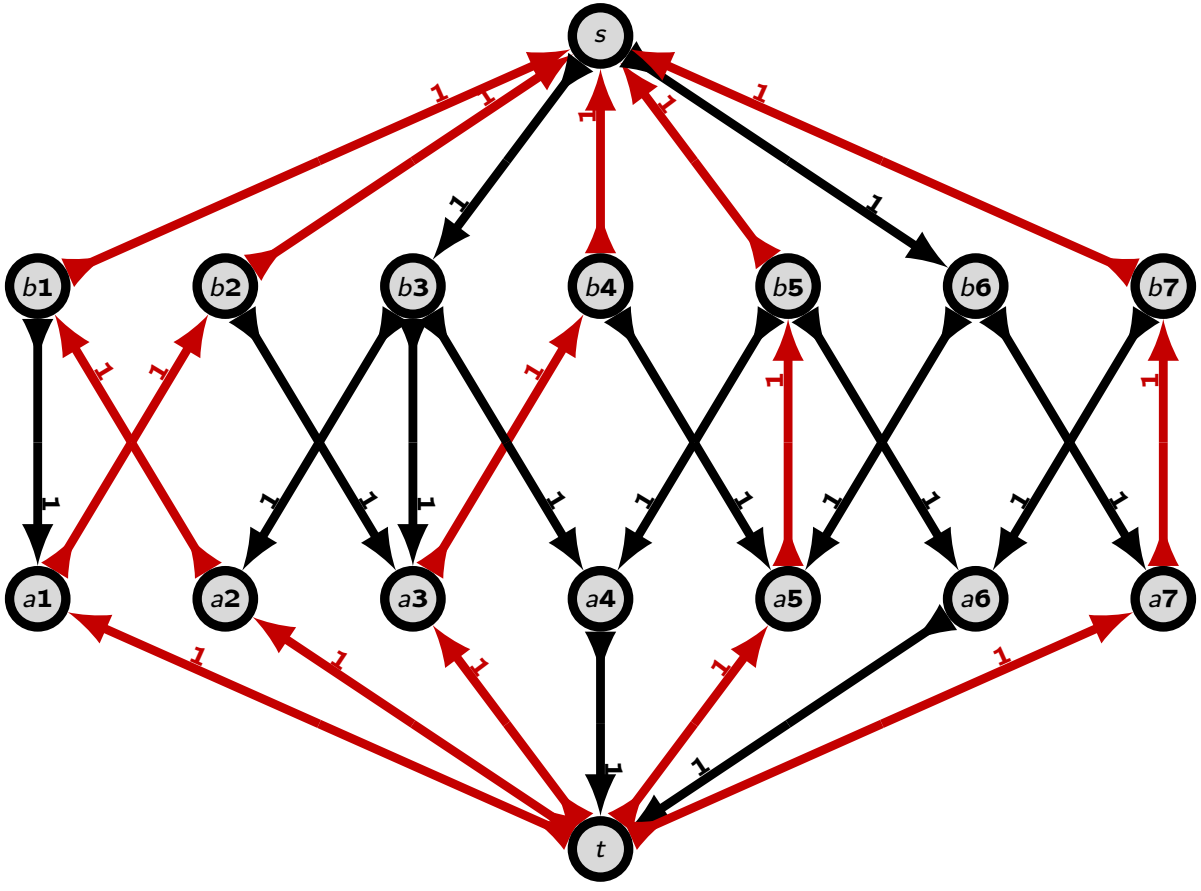
Bipartites Matching und Flüsse



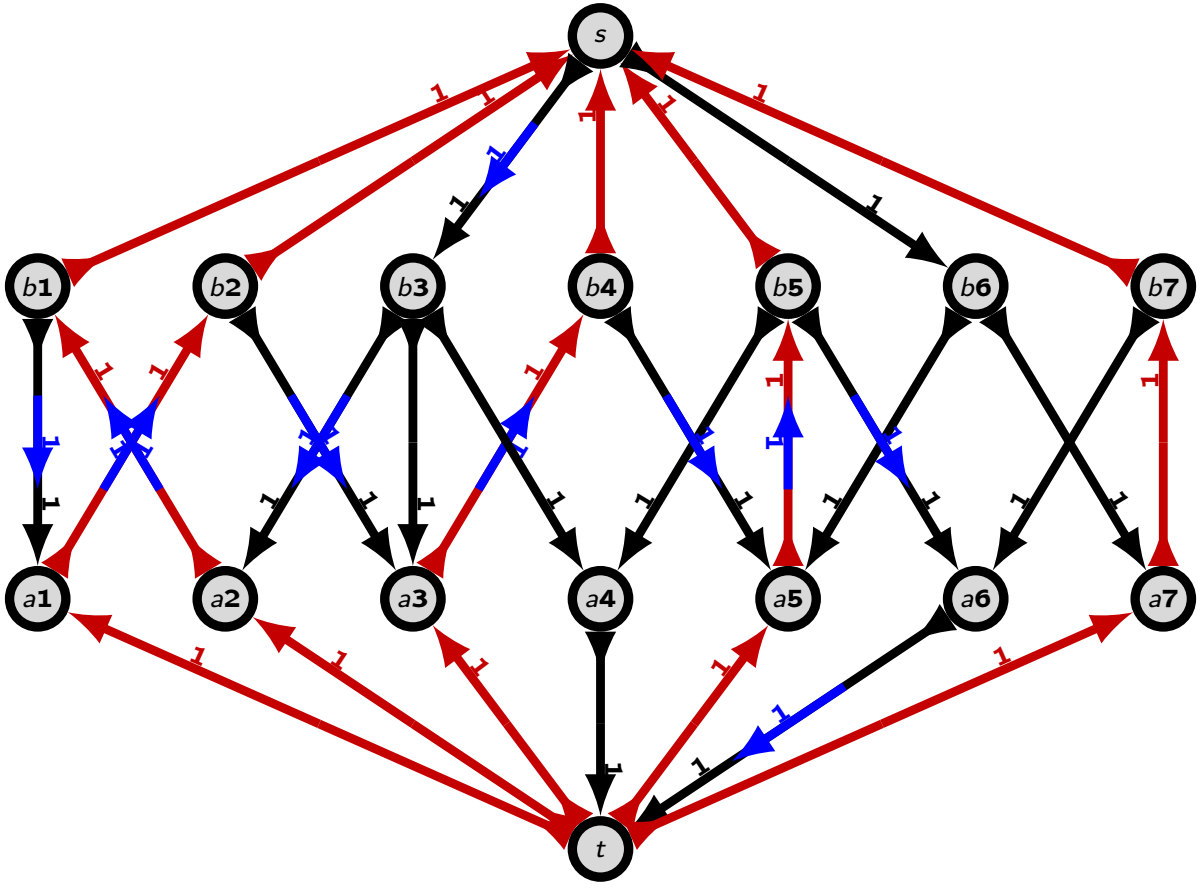
Bipartites Matching und Flüsse



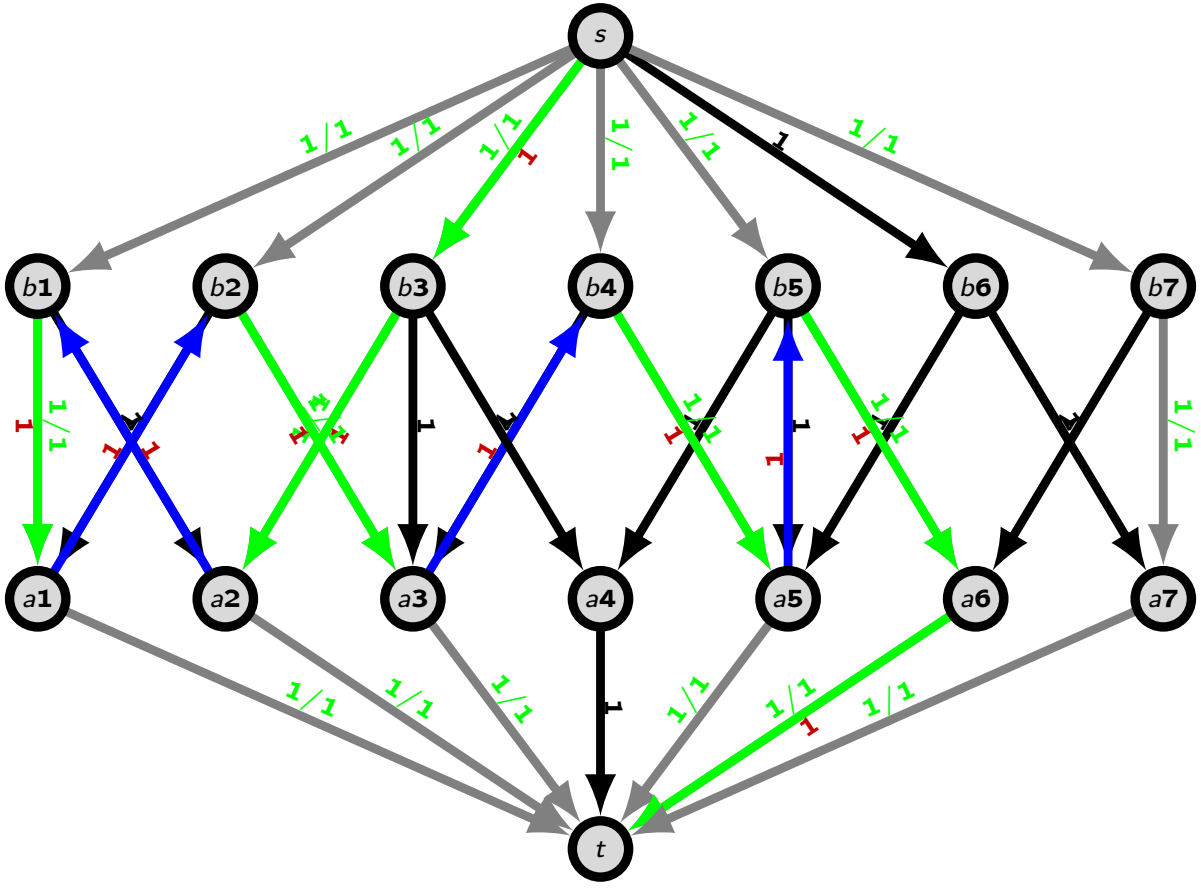
Bipartites Matching und Flüsse



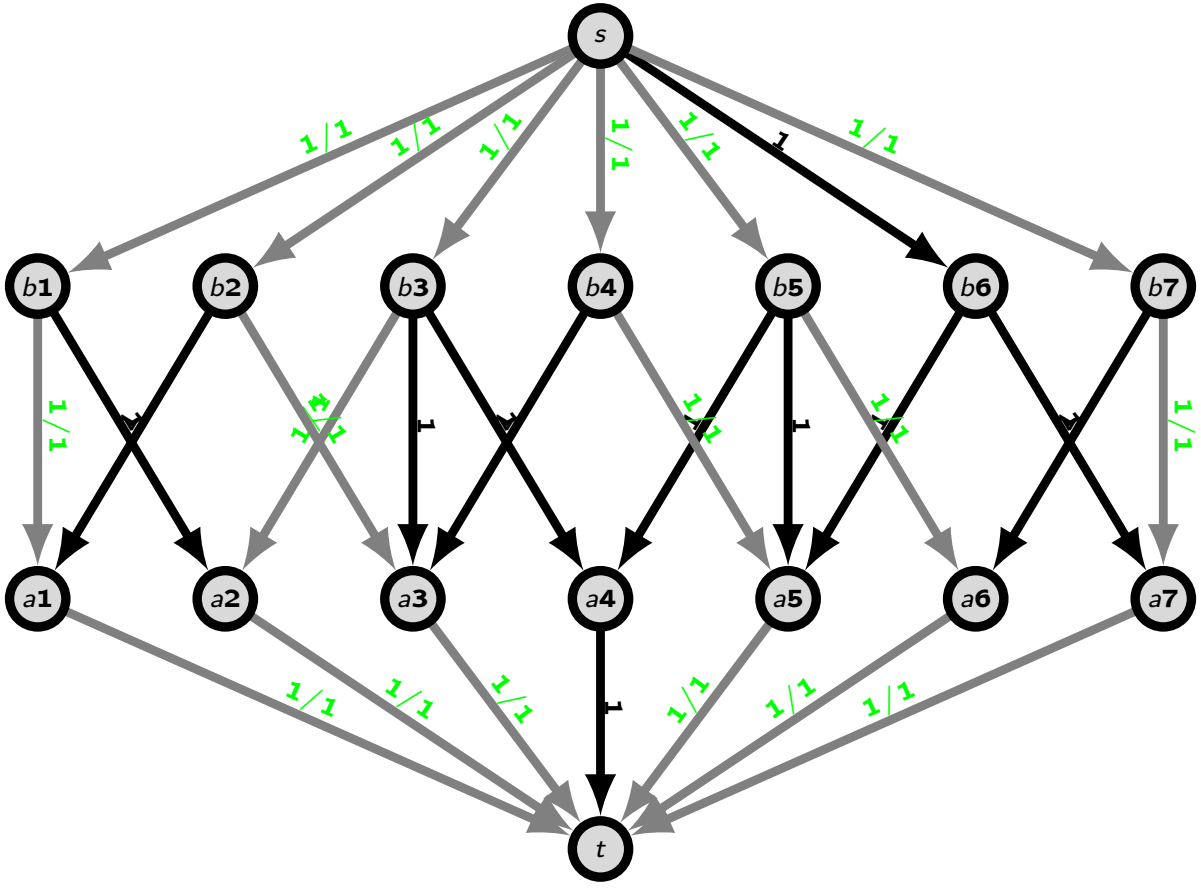
Bipartites Matching und Flüsse



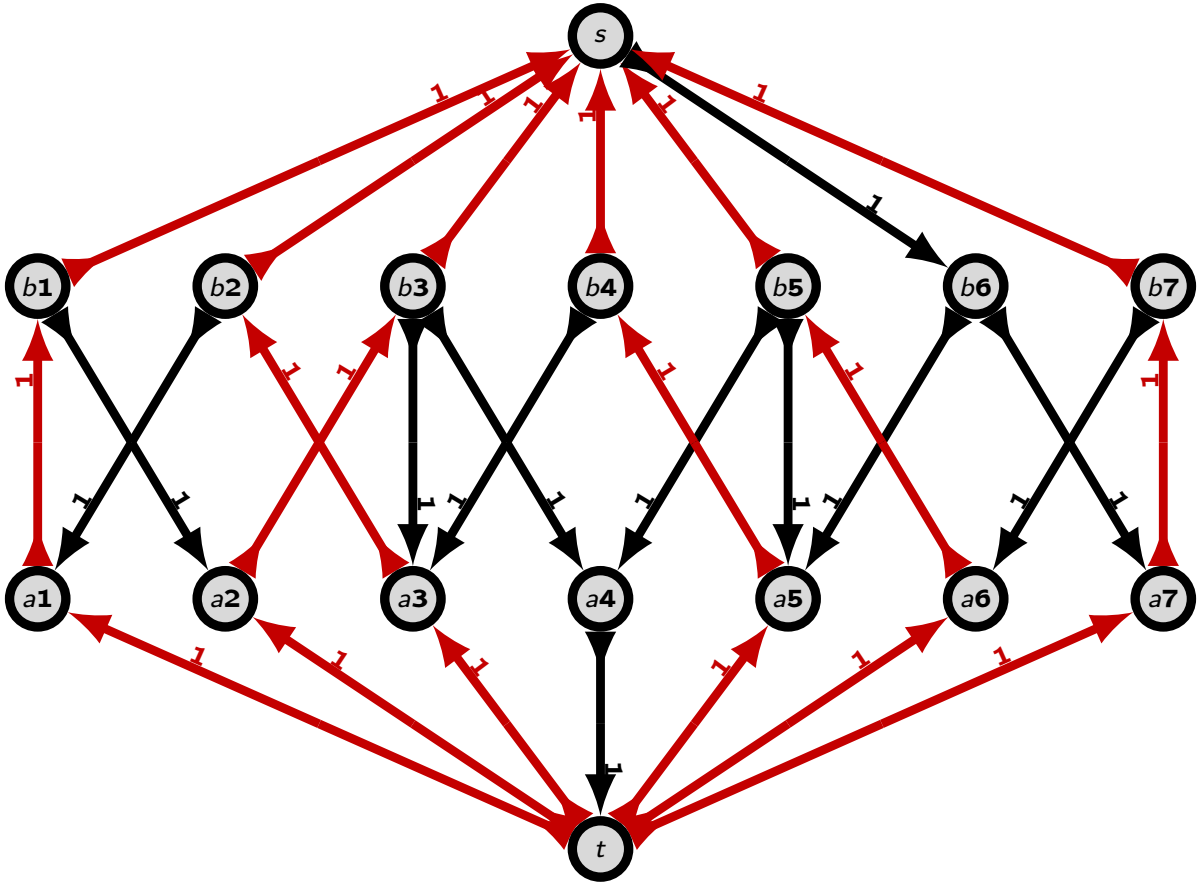
Bipartites Matching und Flüsse



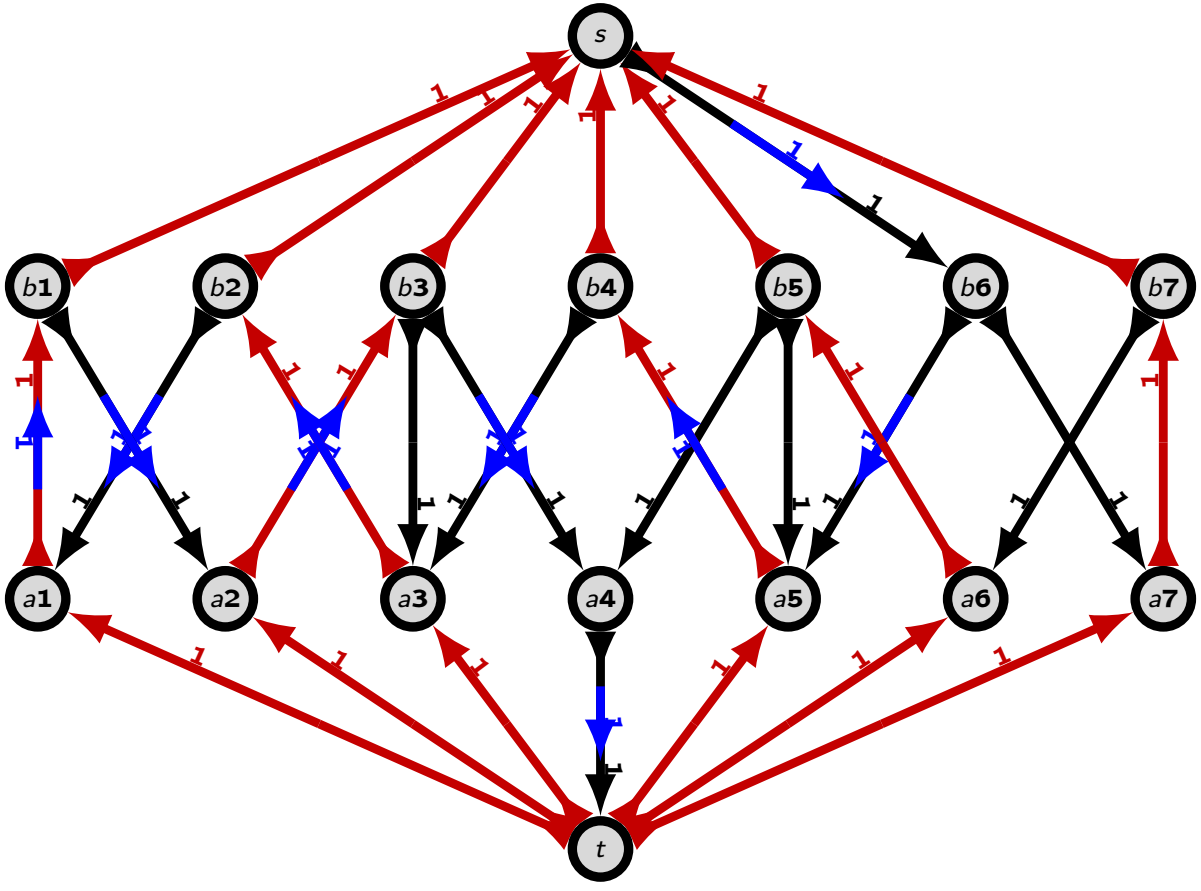
Bipartites Matching und Flüsse



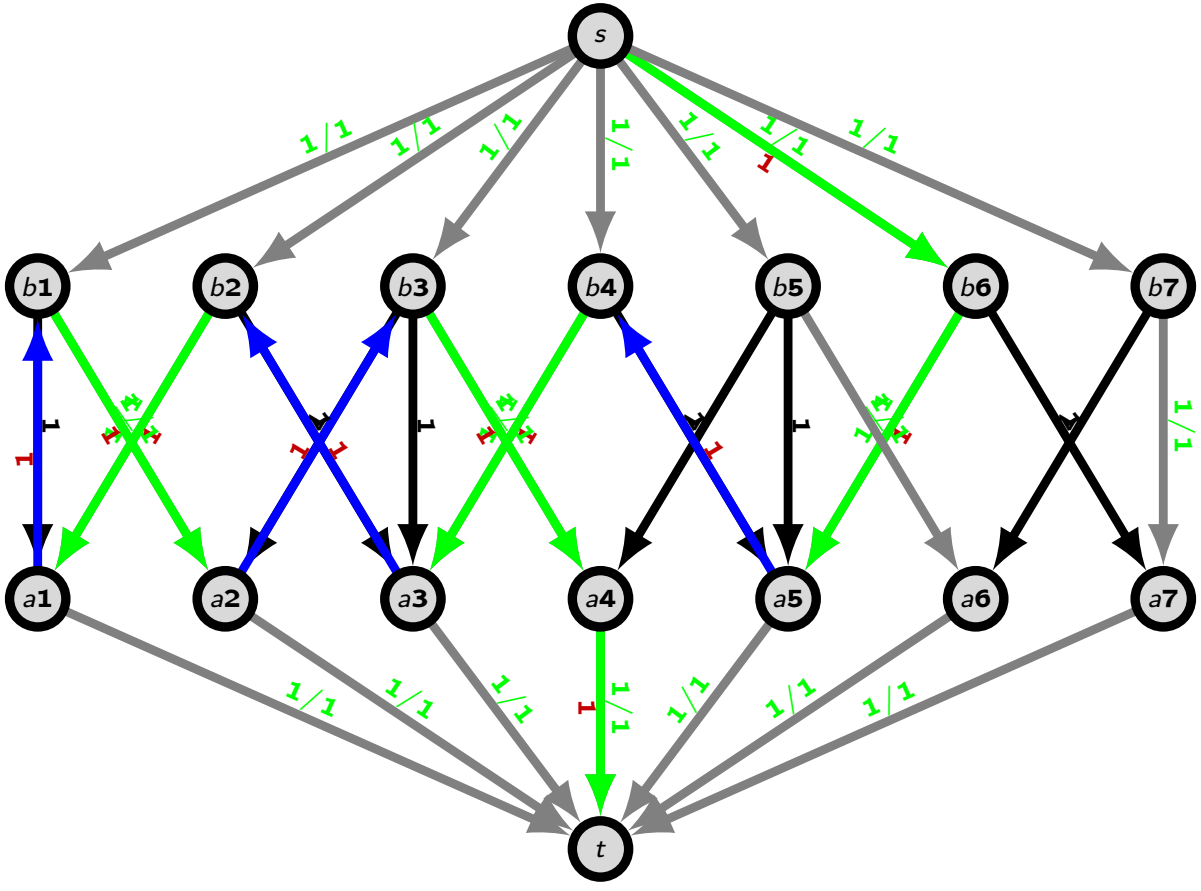
Bipartites Matching und Flüsse



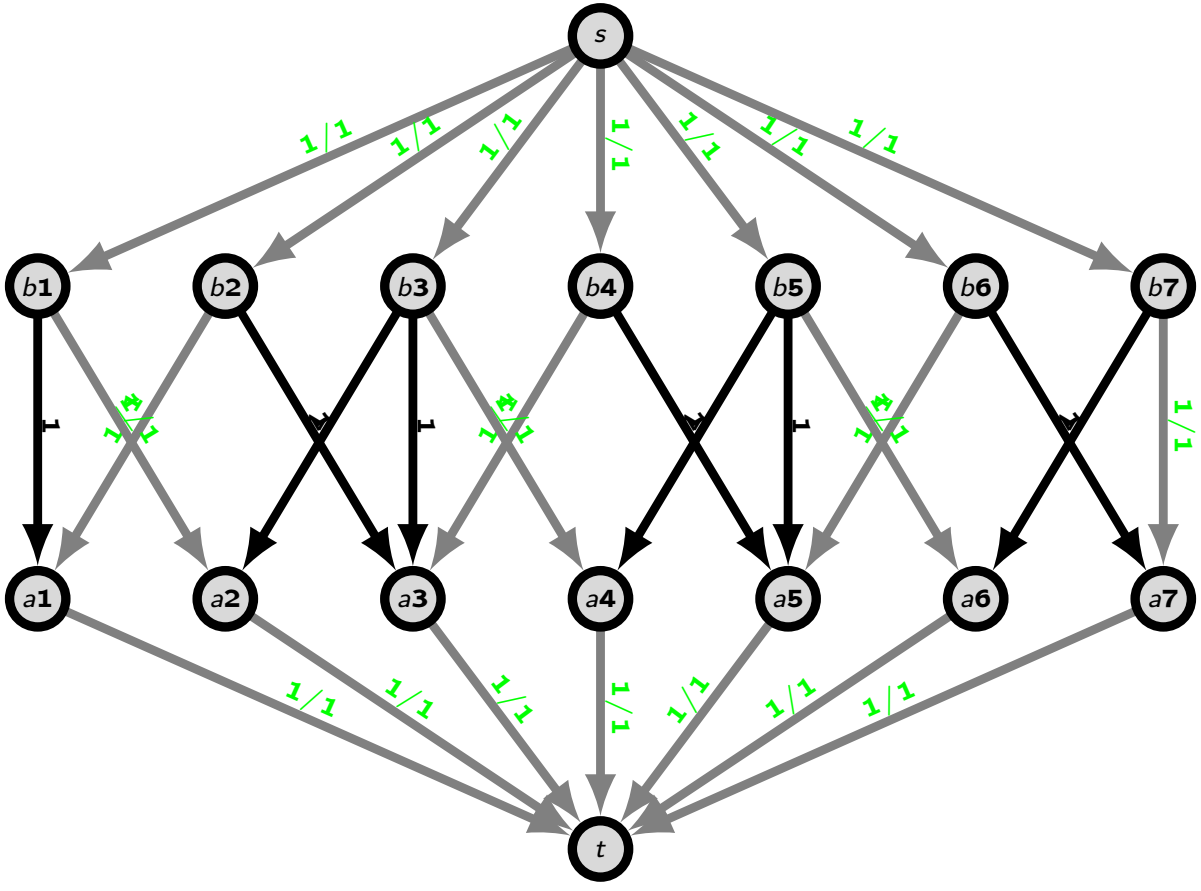
Bipartites Matching und Flüsse



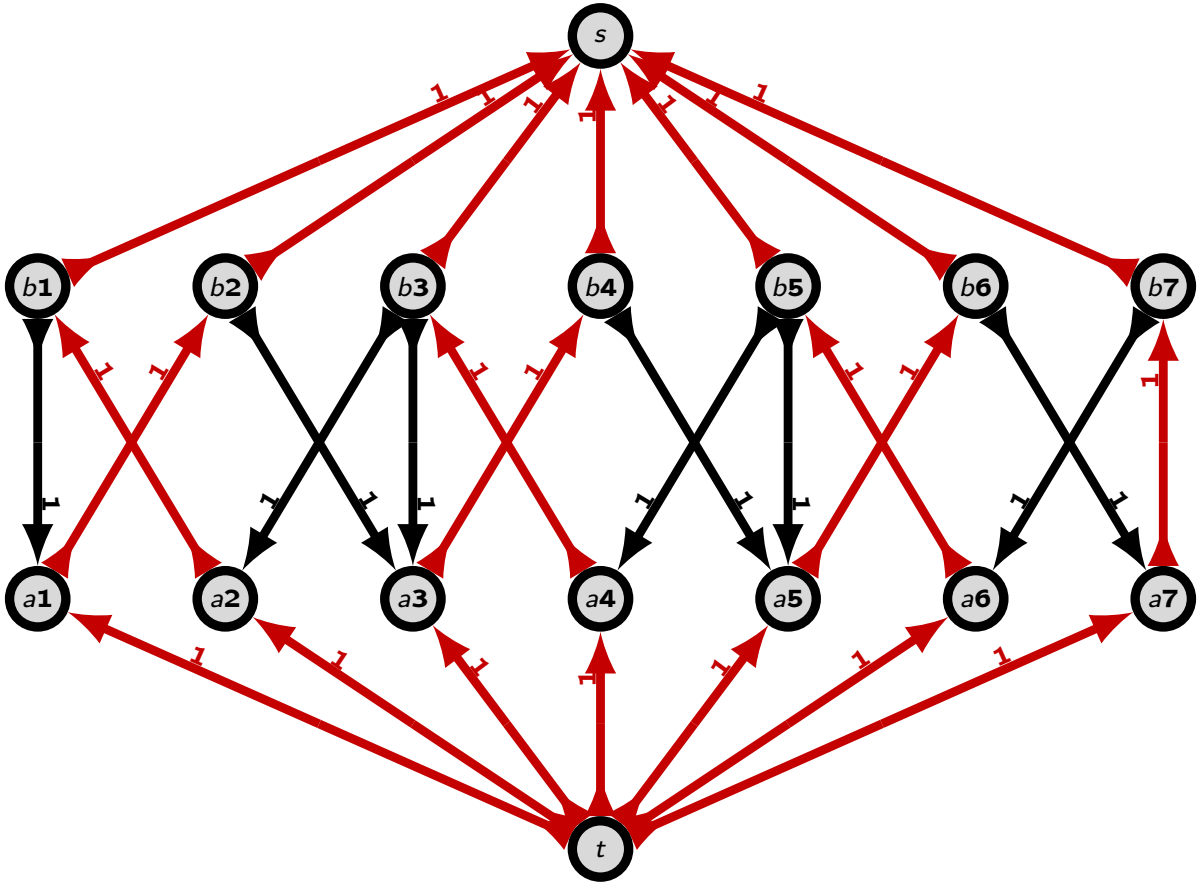
Bipartites Matching und Flüsse



Bipartites Matching und Flüsse



Bipartites Matching und Flüsse



Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem kann in Zeit $O(n + m)$ auf das Flussproblem transformiert werden.
G hat Matching der Größe α genau dann, wenn $w(G') = \alpha$.

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem kann in Zeit $O(n + m)$ auf das Flussproblem transformiert werden.
G hat Matching der Größe α genau dann, wenn $w(G') = \alpha$.

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem kann in Zeit $O(n + m)$ auf das Flussproblem transformiert werden.
G hat Matching der Größe α genau dann, wenn $w(G') = \alpha$.

Theorem

Das maximum Matching Problem ist auf bipartiten Graphen in Zeit $O(n^3)$ lösbar.

Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem kann in Zeit $O(n + m)$ auf das Flussproblem transformiert werden.
G hat Matching der Größe α genau dann, wenn $w(G') = \alpha$.

Theorem

Das maximum Matching Problem ist auf bipartiten Graphen in Zeit $O(n^3)$ lösbar.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem mit Kosten kann in Zeit $O(n + m)$ auf das Flussproblem mit Kosten transformiert werden.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem mit Kosten kann in Zeit $O(n + m)$ auf das Flussproblem mit Kosten transformiert werden.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem mit Kosten kann in Zeit $O(n + m)$ auf das Flussproblem mit Kosten transformiert werden.

Theorem

G hat Matching der Größe α mit Kosten β genau dann, wenn $w(G') = \alpha$ und die Kosten des Flusses sind β.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem mit Kosten kann in Zeit $O(n + m)$ auf das Flussproblem mit Kosten transformiert werden.

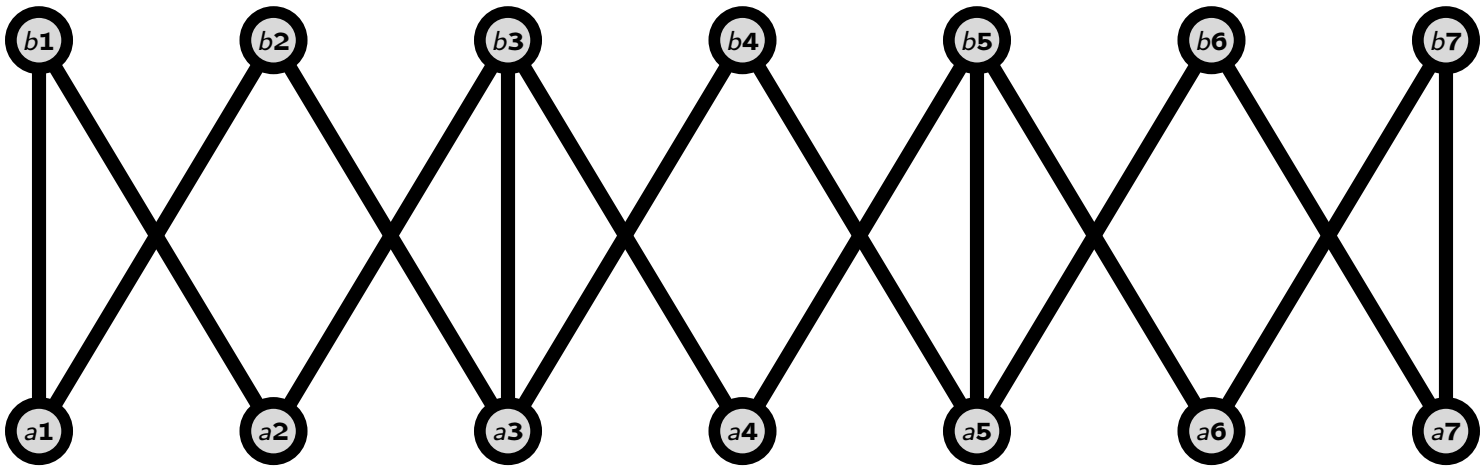
Theorem

G hat Matching der Größe α mit Kosten β genau dann, wenn $w(G') = \alpha$ und die Kosten des Flusses sind β.

Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

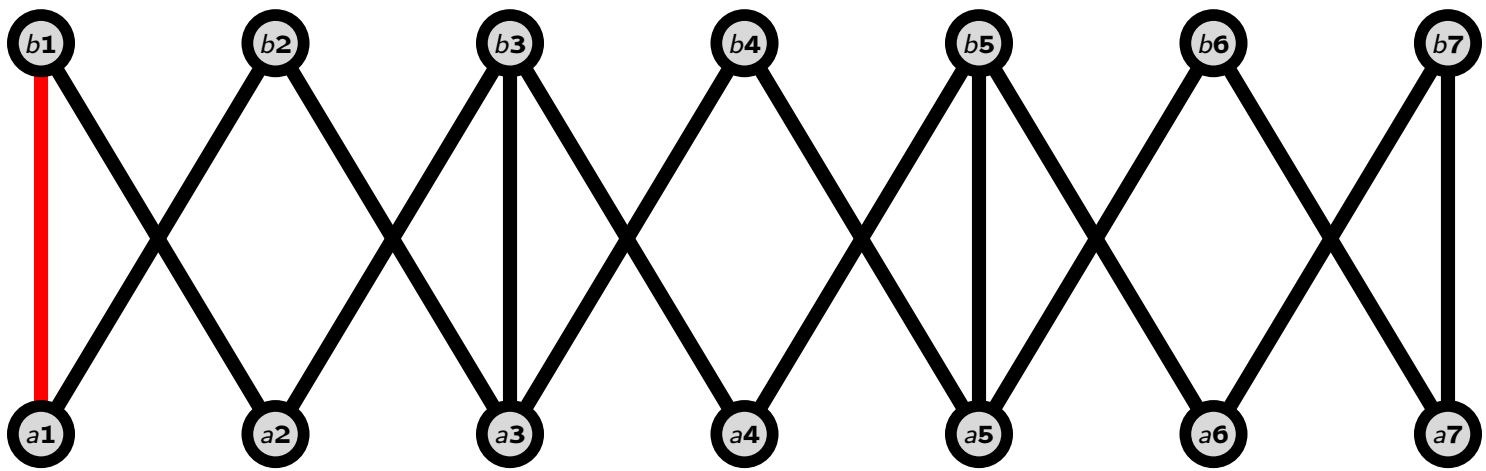
- Ein Fluss über eine Kante entspricht einer Matchingkante.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

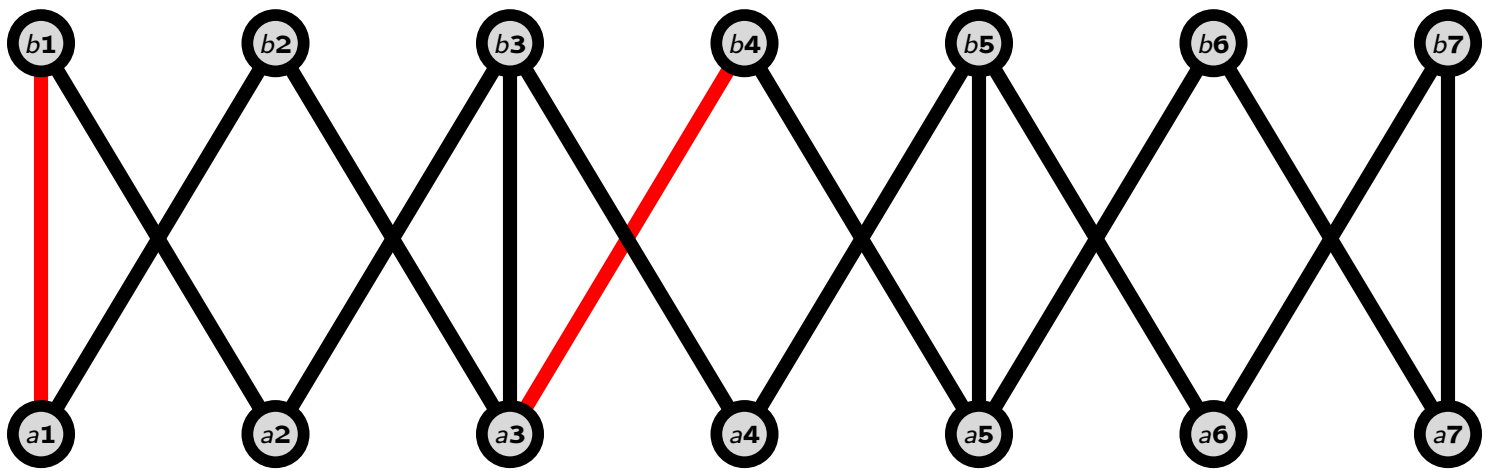
- Ein Fluss über eine Kante entspricht einer Matchingkante.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

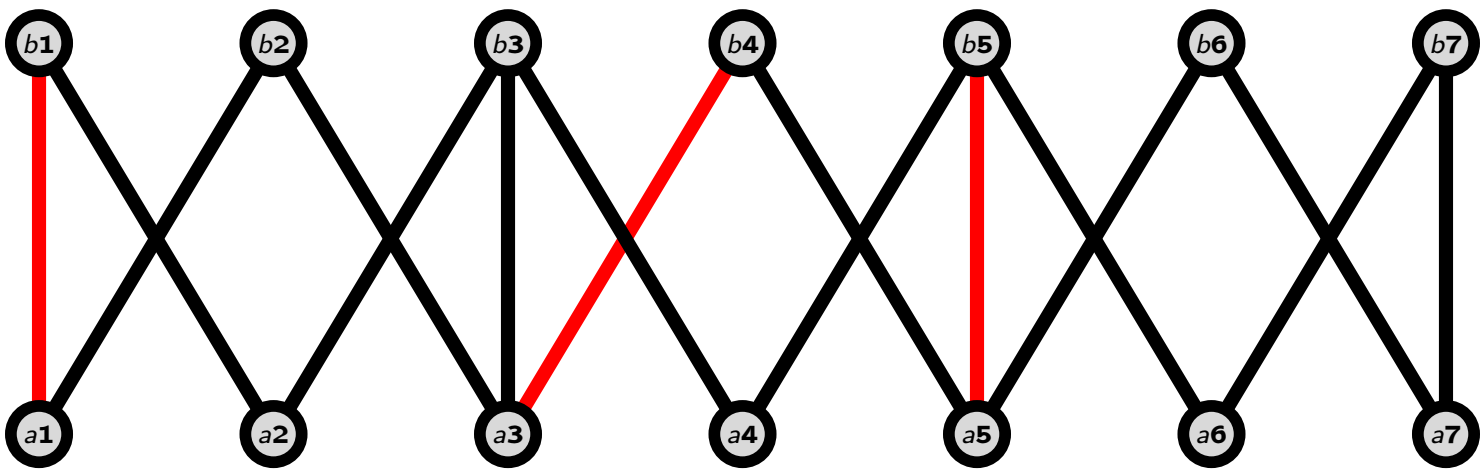
- Ein Fluss über eine Kante entspricht einer Matchingkante.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

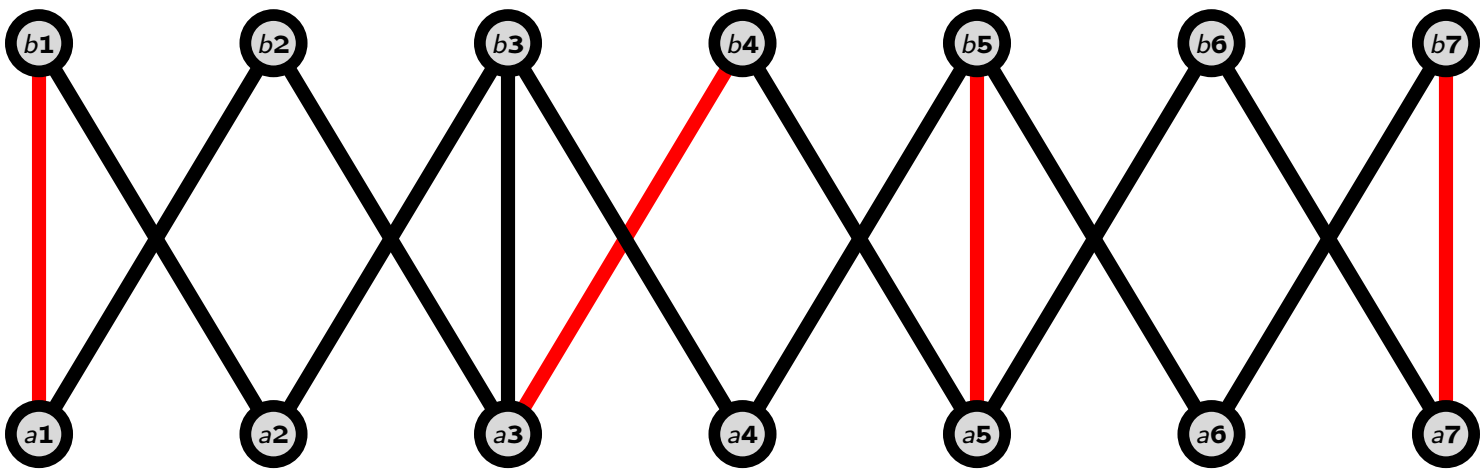
- Ein Fluss über eine Kante entspricht einer Matchingkante.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

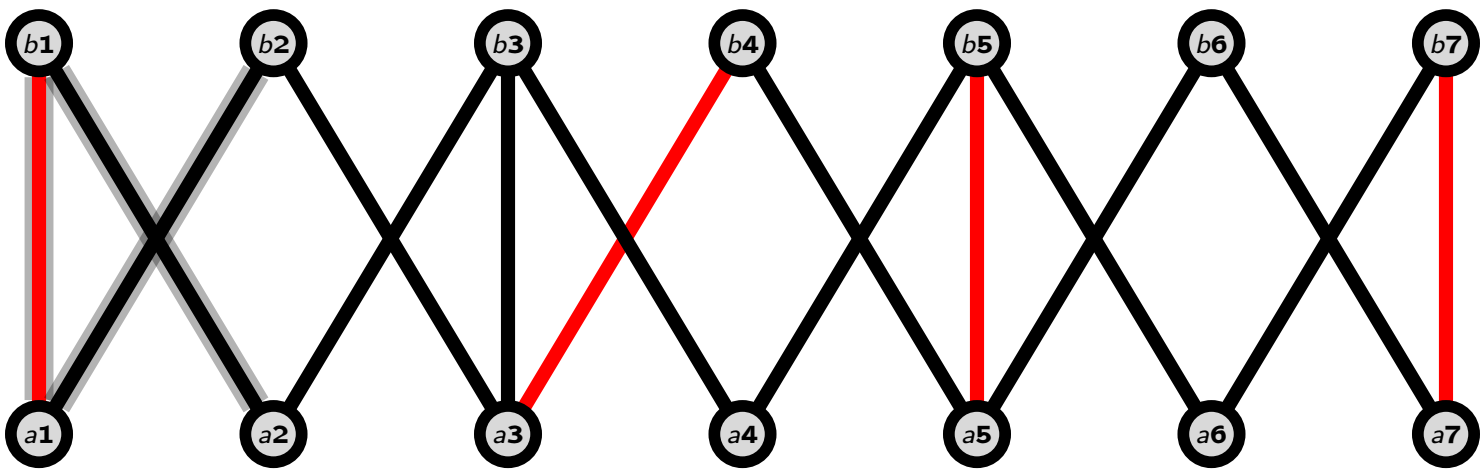
- Ein Fluss über eine Kante entspricht einer Matchingkante.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

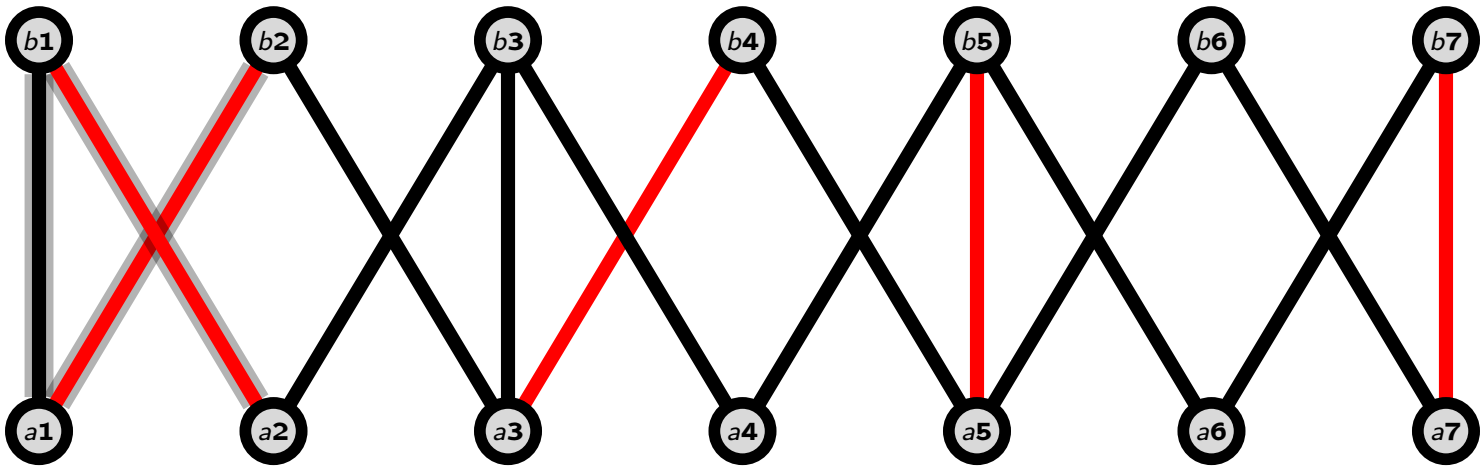
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

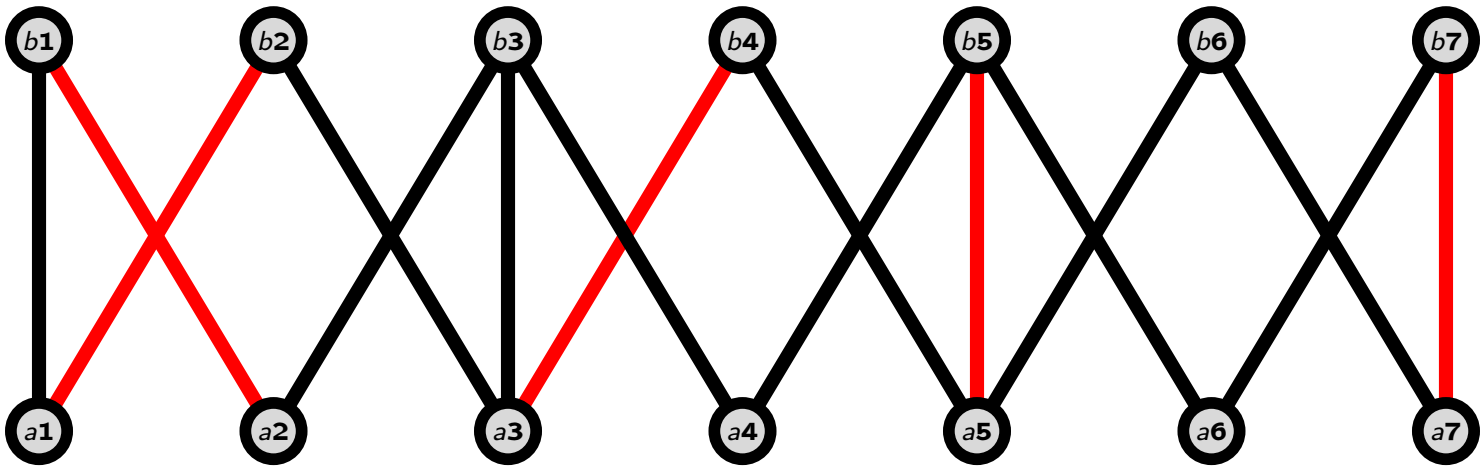
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

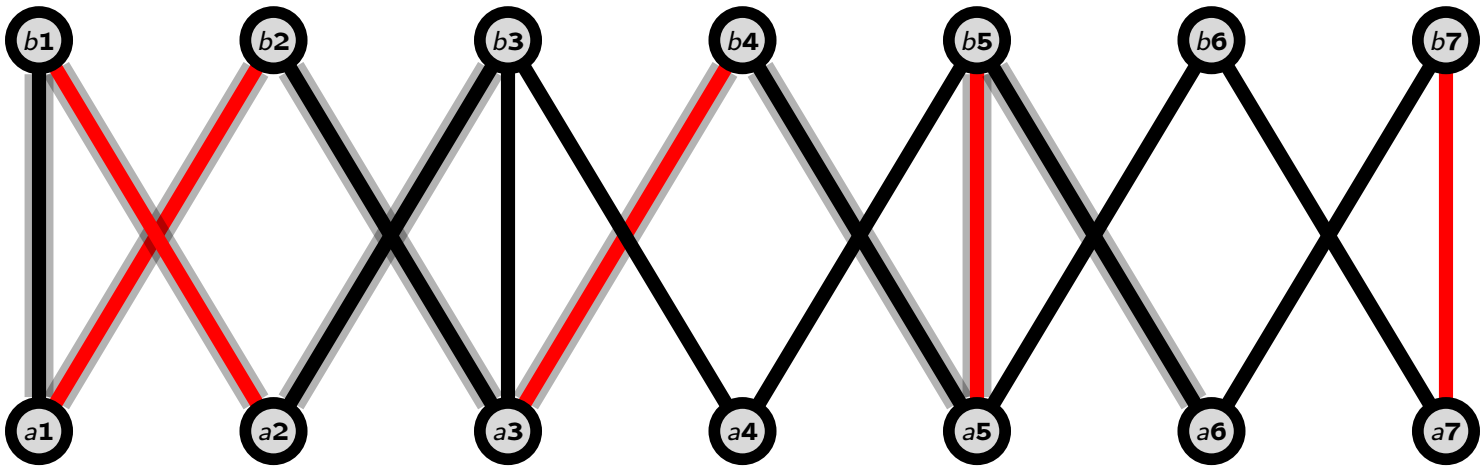
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

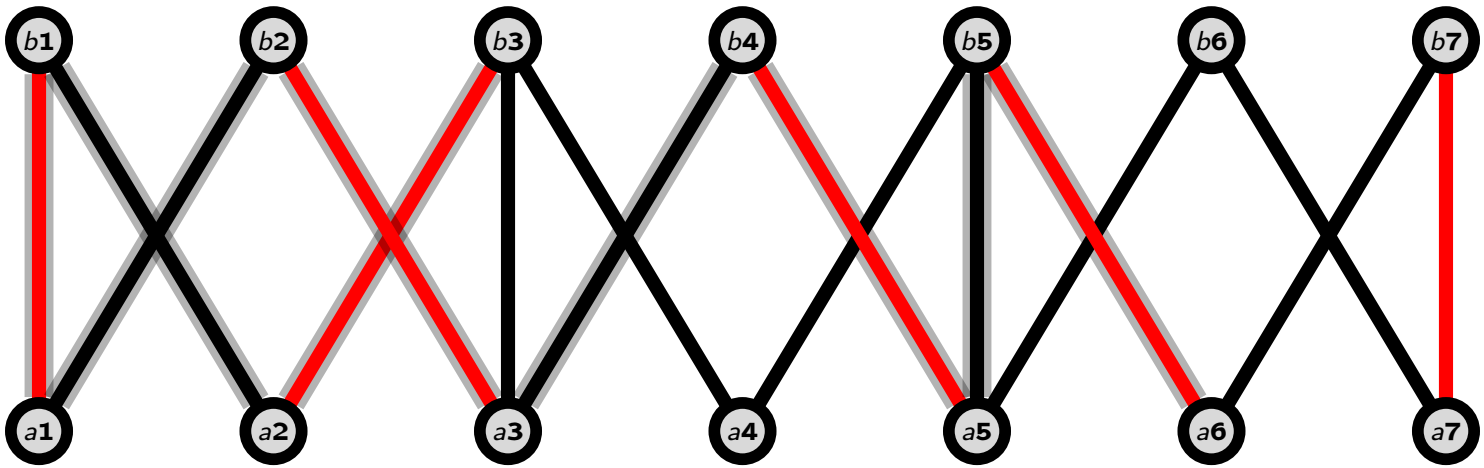
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

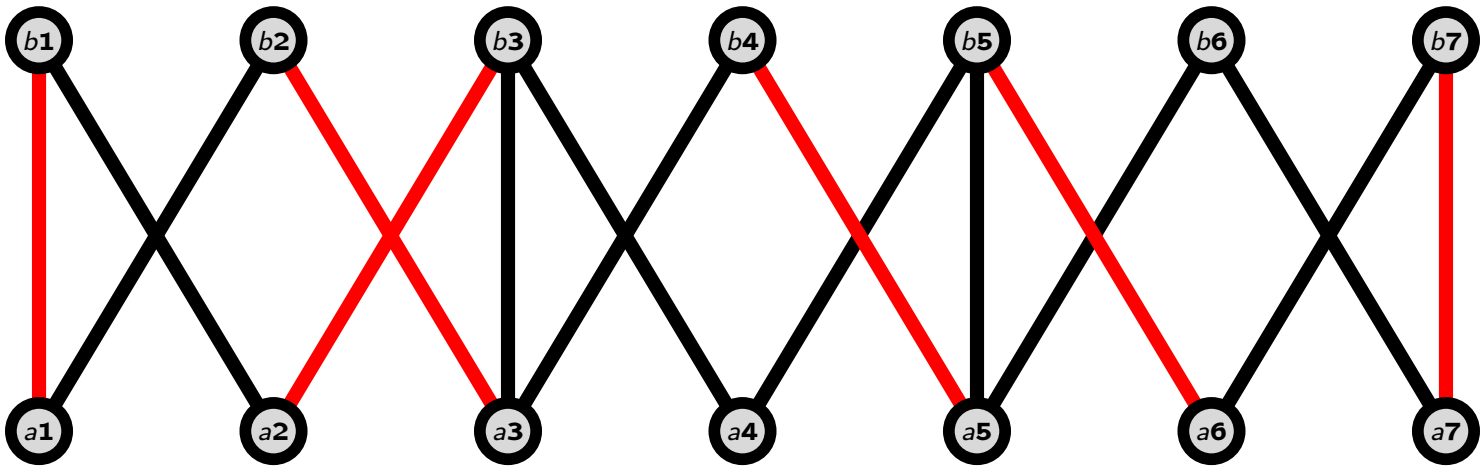
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

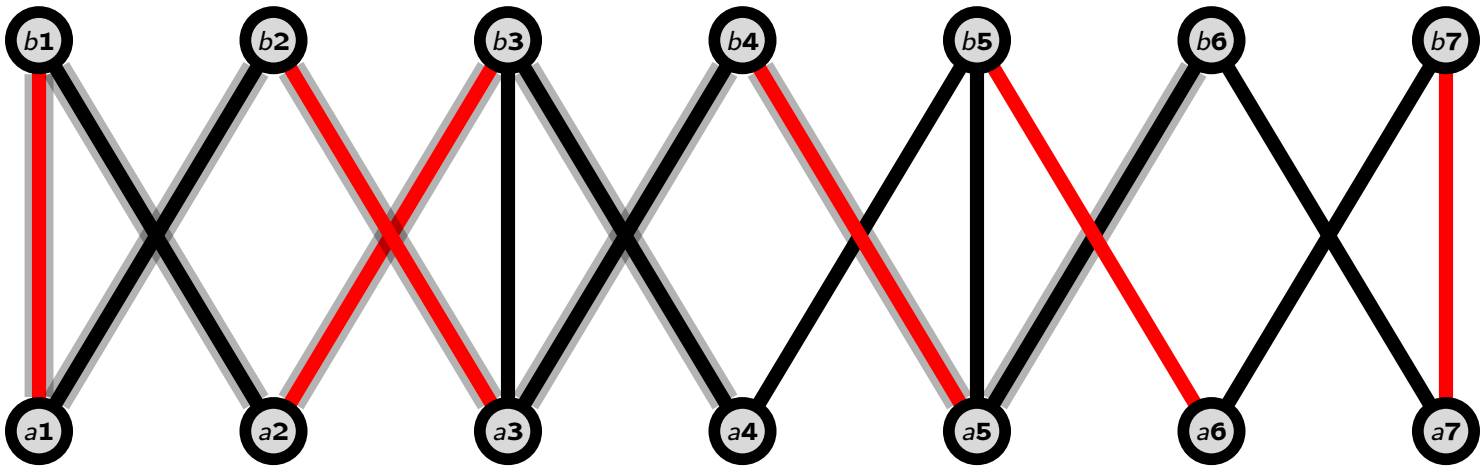
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

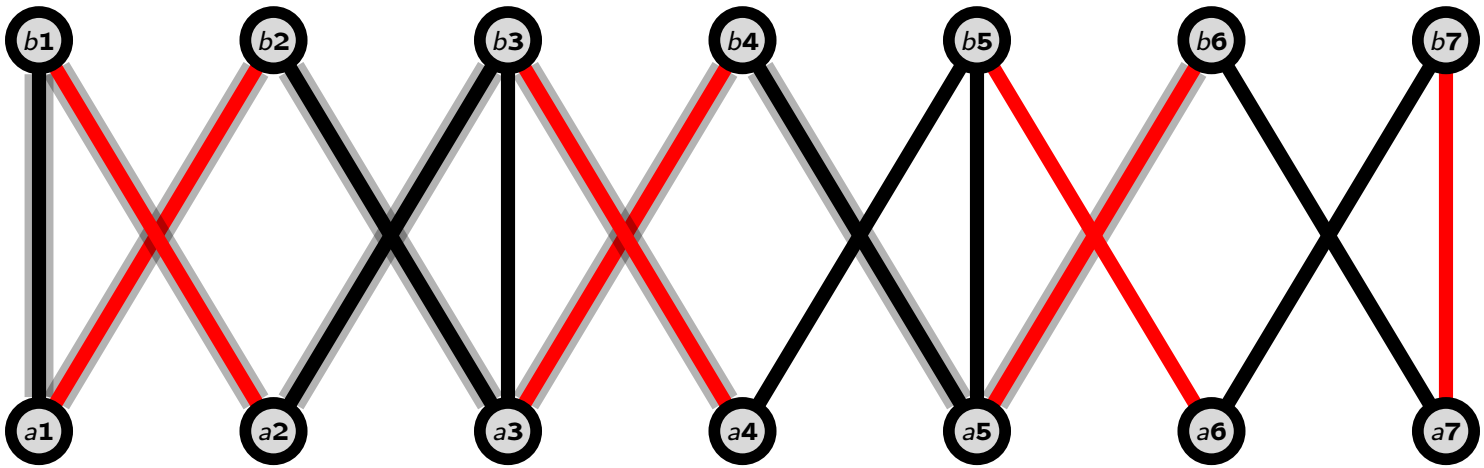
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

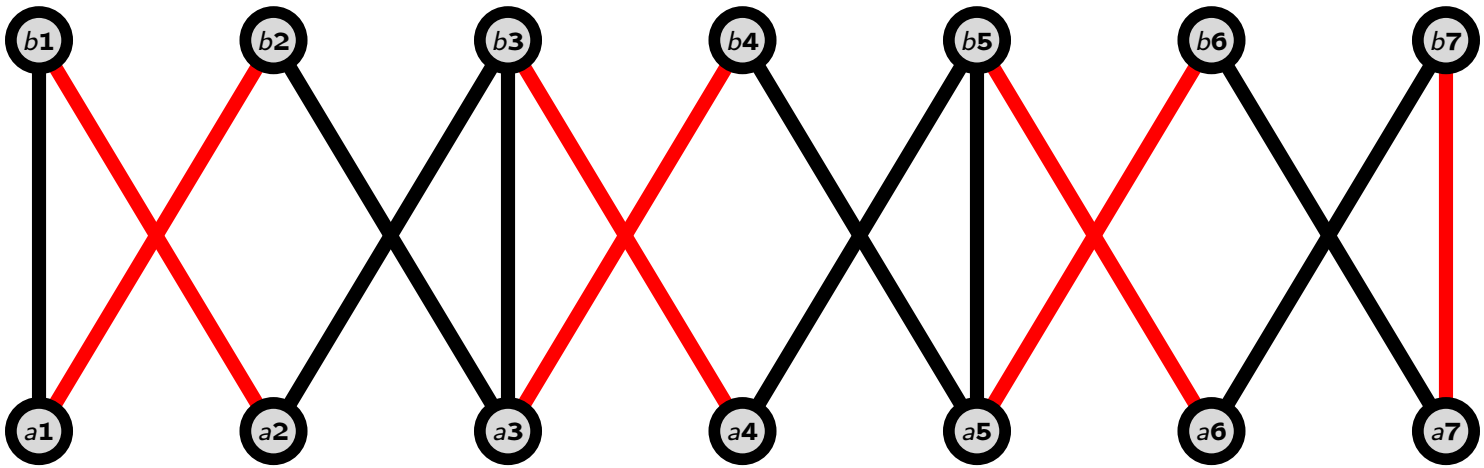
- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme “erweiternden Pfad”:

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme “erweiternden Pfad”:
  - Der Pfad startet an einem “freien” Knoten.

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme “erweiternden Pfad”:
  - Der Pfad startet an einem “freien” Knoten.
  - Der Pfad startet mit einer Vorwärtskante.

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme “erweiternden Pfad”:
  - Der Pfad startet an einem “freien” Knoten.
  - Der Pfad startet mit einer Vorwärtskante.
  - Der Pfad geht alternierend über Vorwärts- und Rückwärtskanten.

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme "erweiternden Pfad":
  - Der Pfad startet an einem "freien" Knoten.
  - Der Pfad startet mit einer Vorwärtskante.
  - Der Pfad geht alternierend über Vorwärts- und Rückwärtskanten.
  - Der Pfad endet mit einer Vorwärtskante.

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme "erweiternden Pfad":
  - Der Pfad startet an einem "freien" Knoten.
  - Der Pfad startet mit einer Vorwärtskante.
  - Der Pfad geht alternierend über Vorwärts- und Rückwärtskanten.
  - Der Pfad endet mit einer Vorwärtskante.
  - Der Pfad endet an einem "freien" Knoten.

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme "erweiternden Pfad":
  - Der Pfad startet an einem "freien" Knoten.
  - Der Pfad startet mit einer Vorwärtskante.
  - Der Pfad geht alternierend über Vorwärts- und Rückwärtskanten.
  - Der Pfad endet mit einer Vorwärtskante.
  - Der Pfad endet an einem "freien" Knoten.
- Damit können wir einen weiteren Algorithmus angeben.

Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme "erweiternden Pfad":
  - Der Pfad startet an einem "freien" Knoten.
  - Der Pfad startet mit einer Vorwärtskante.
  - Der Pfad geht alternierend über Vorwärts- und Rückwärtskanten.
  - Der Pfad endet mit einer Vorwärtskante.
  - Der Pfad endet an einem "freien" Knoten.
- Damit können wir einen weiteren Algorithmus angeben.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

- Ein alterniernder Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

- Ein alterniernder Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

- Ein alterniernder Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.
- Damit arbeitet der Algorithmus wie folgt:

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

- Ein alterniernder Pfad
  $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.
- Damit arbeitet der Algorithmus wie folgt:
  1. Setze $M = \emptyset$.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

- Ein alterniernder Pfad
  $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.
- Damit arbeitet der Algorithmus wie folgt:
  1. Setze $M = \emptyset$.
  2. Solange es erweiternden Pfad P gibt, mache:

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

- Ein alterniernder Pfad
  $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.
- Damit arbeitet der Algorithmus wie folgt:
  1. Setze $M = \emptyset$.
  2. Solange es erweiternden Pfad P gibt, mache:
     1. Erweitere M, d.h. $M = M \oplus E(P)$.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Graph und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

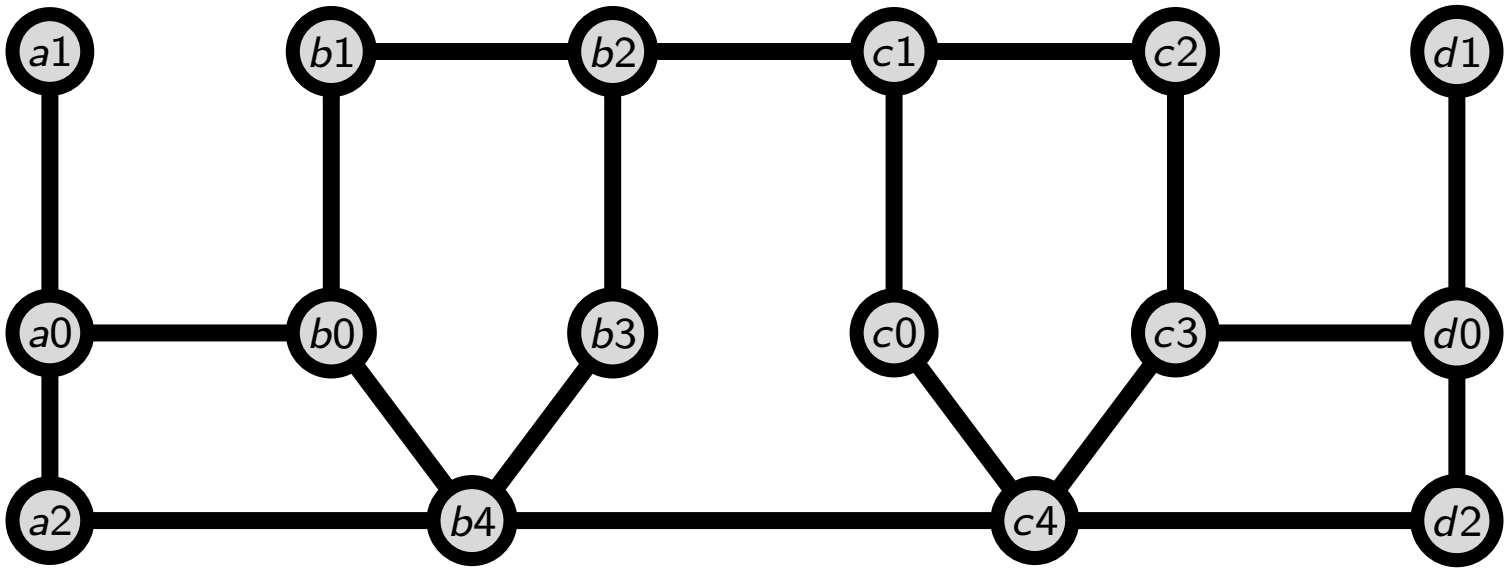
- Ein alterniernder Pfad
  $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.
- Damit arbeitet der Algorithmus wie folgt:
  1. Setze $M = \emptyset$.
  2. Solange es erweiternden Pfad P gibt, mache:
     1. Erweitere M, d.h. $M = M \oplus E(P)$.

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

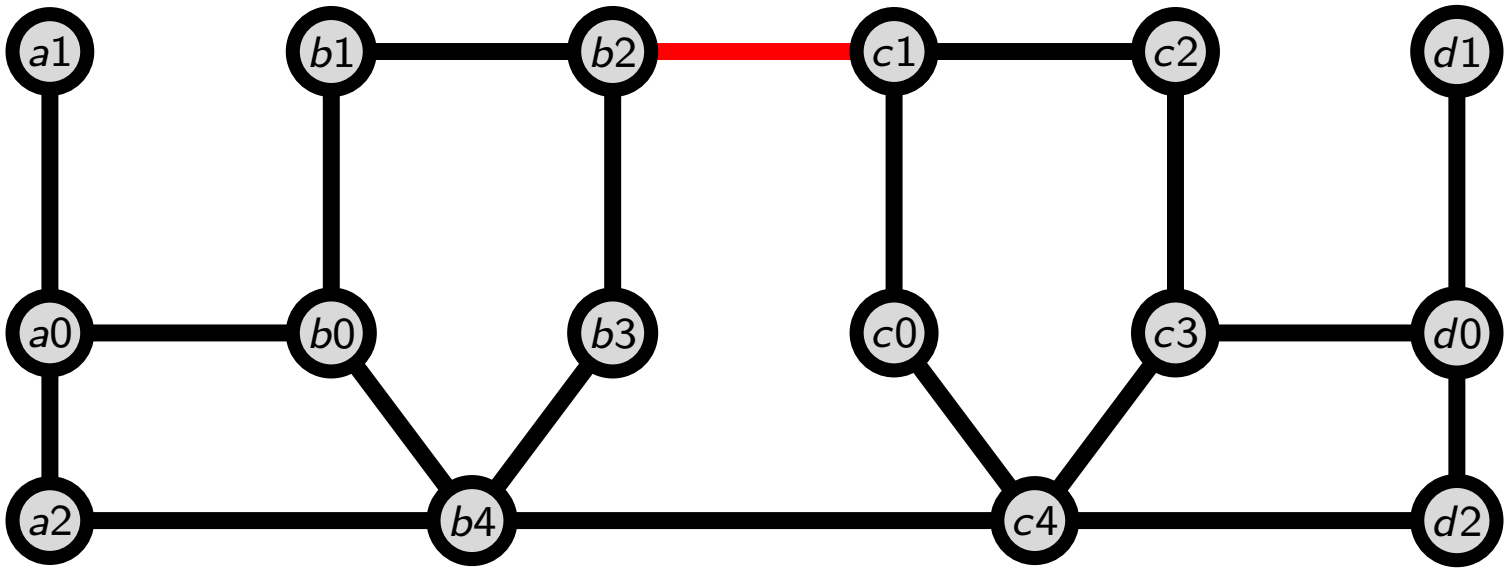
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

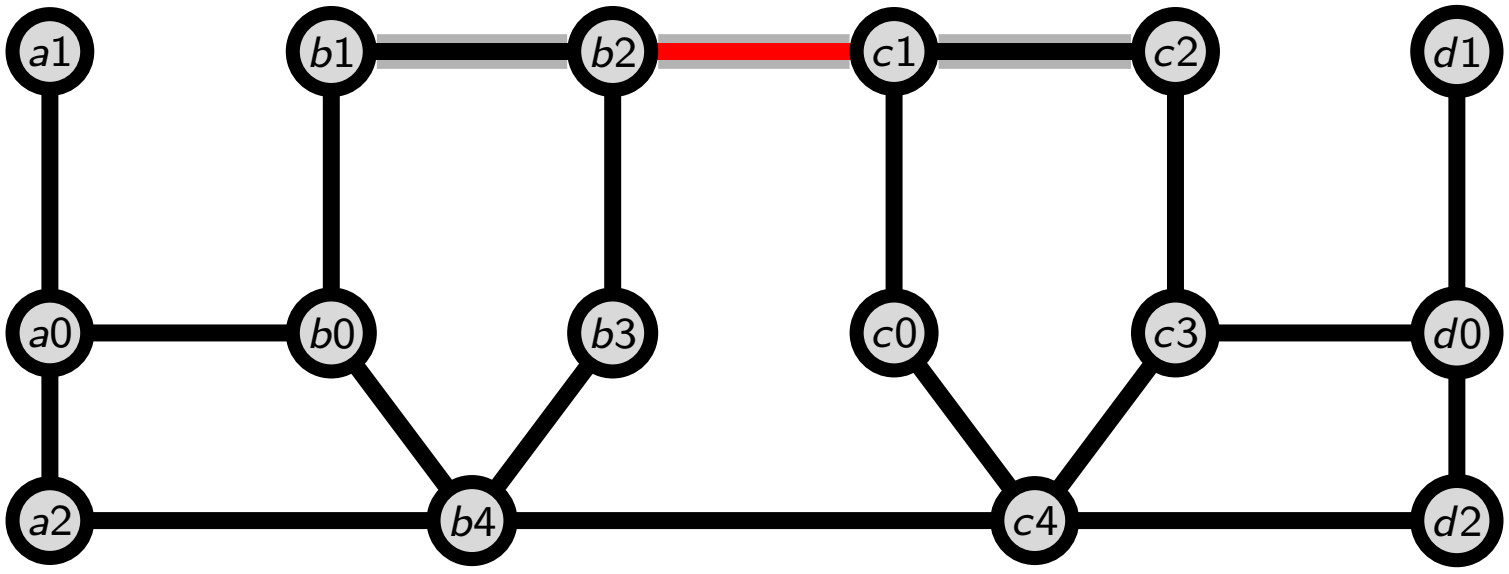
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

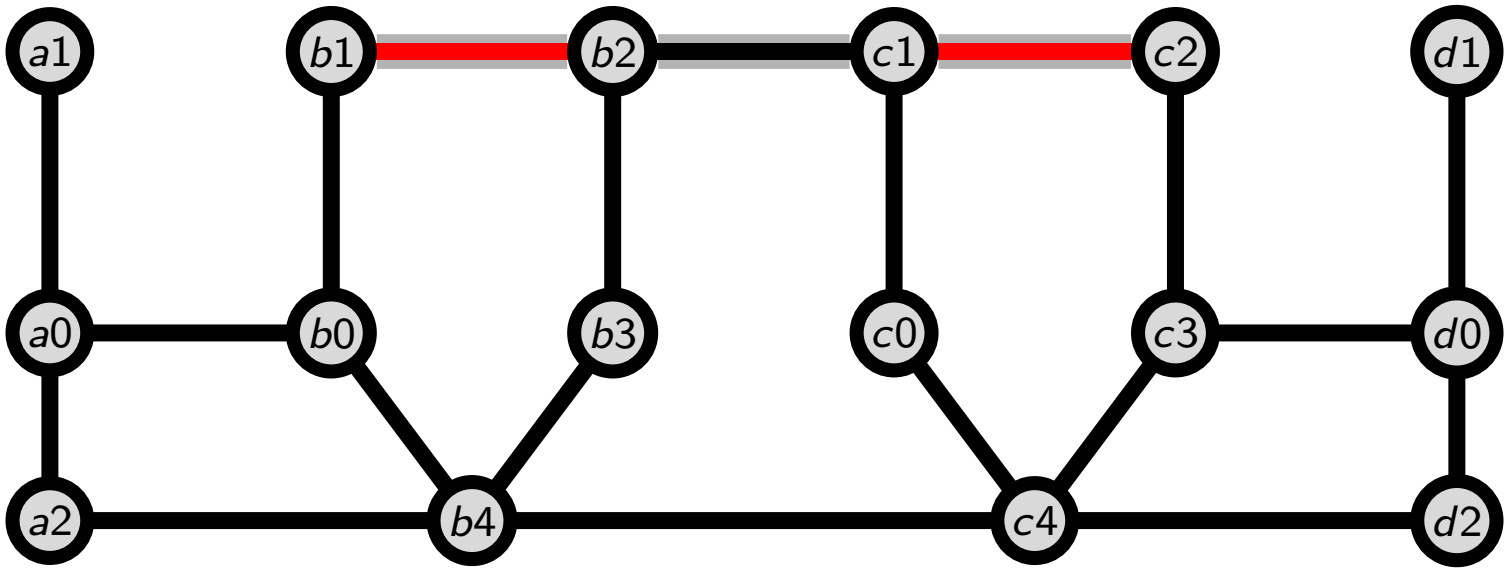
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

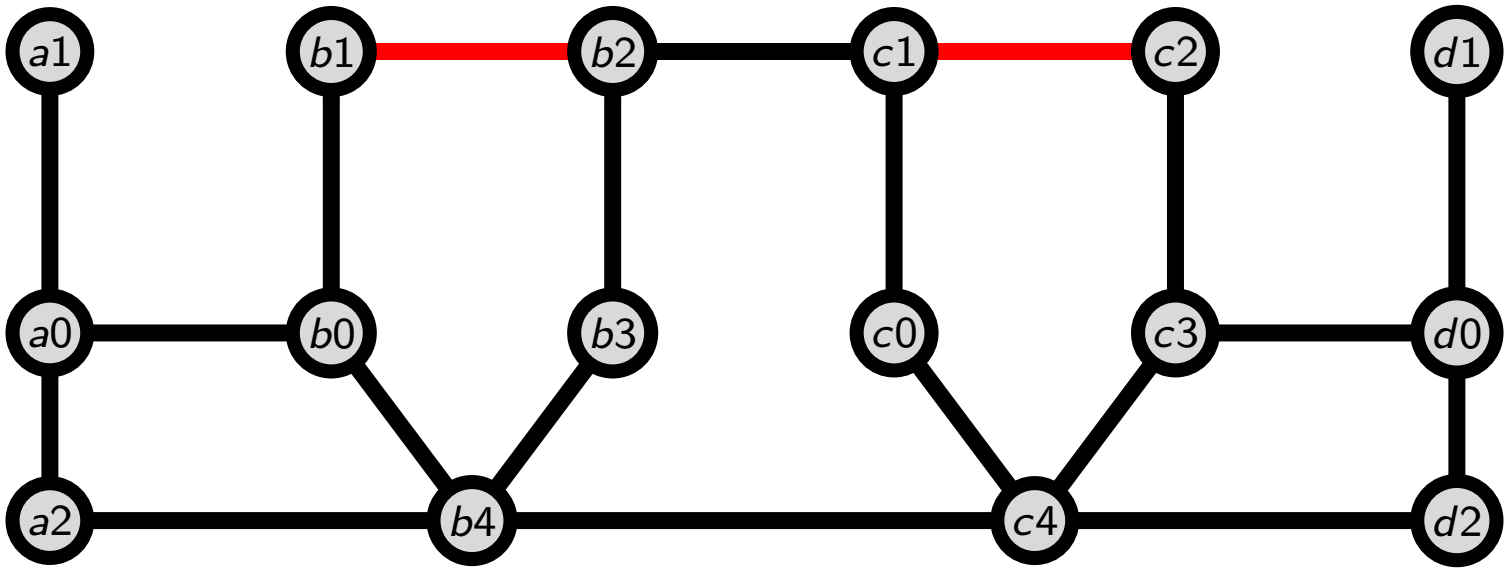
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

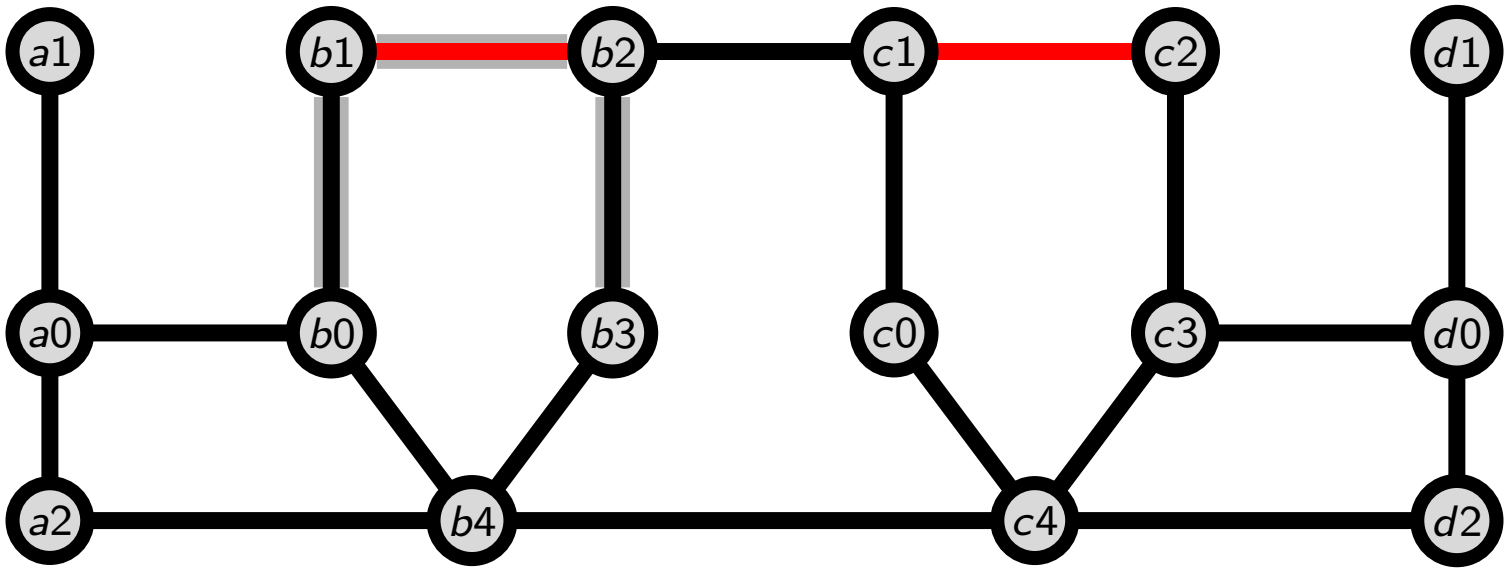
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

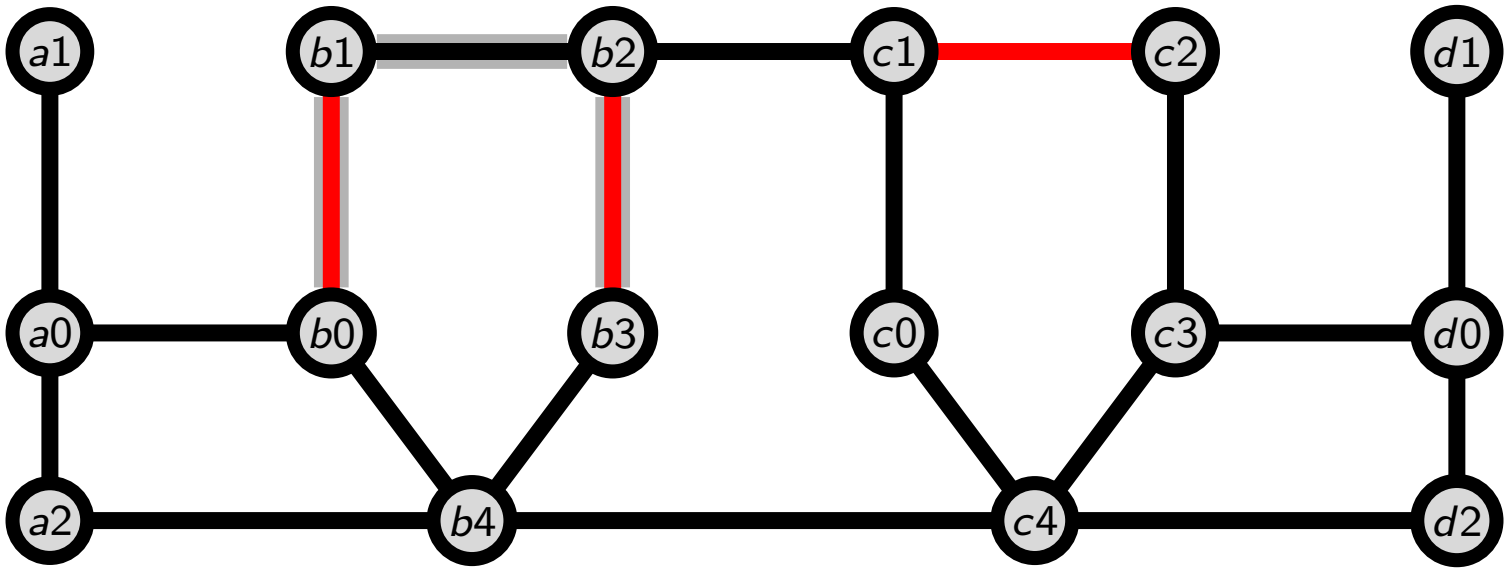
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

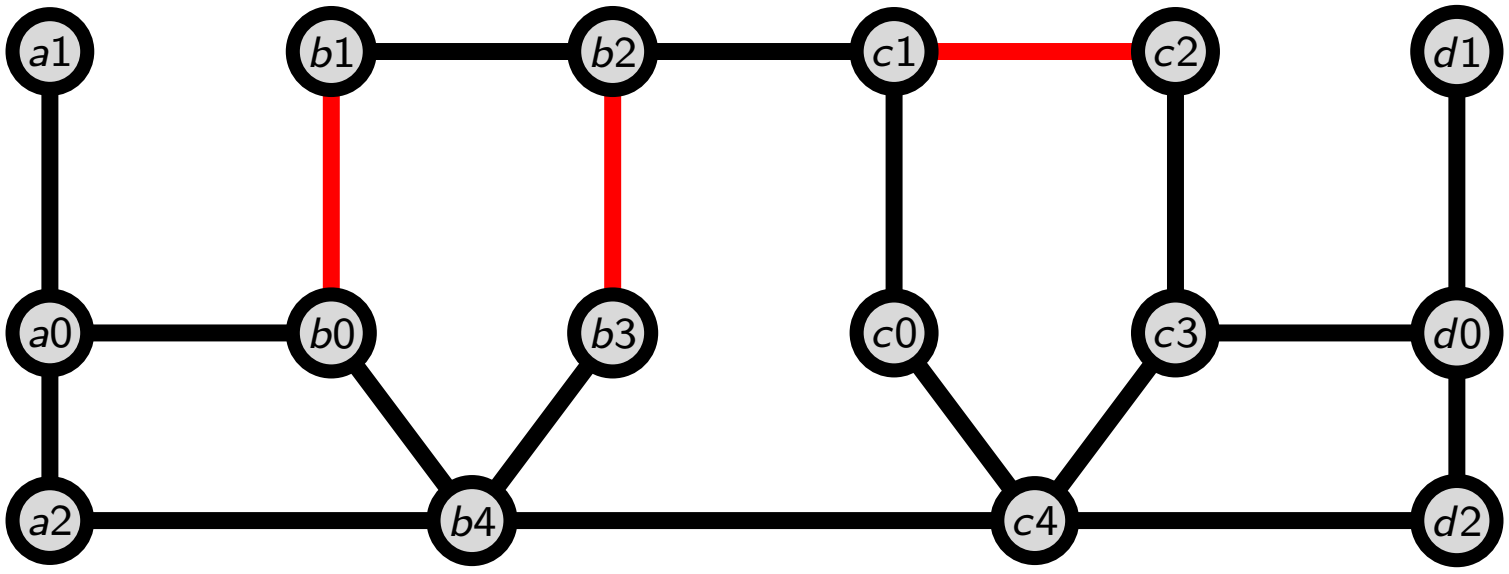
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

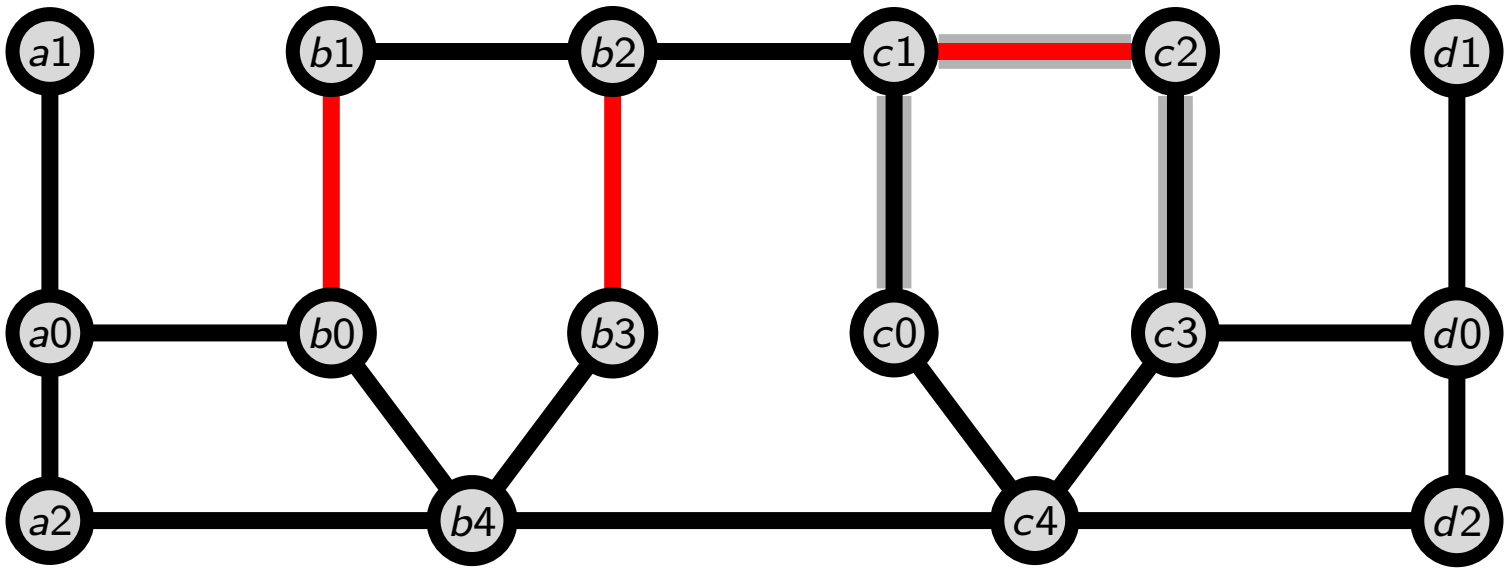
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

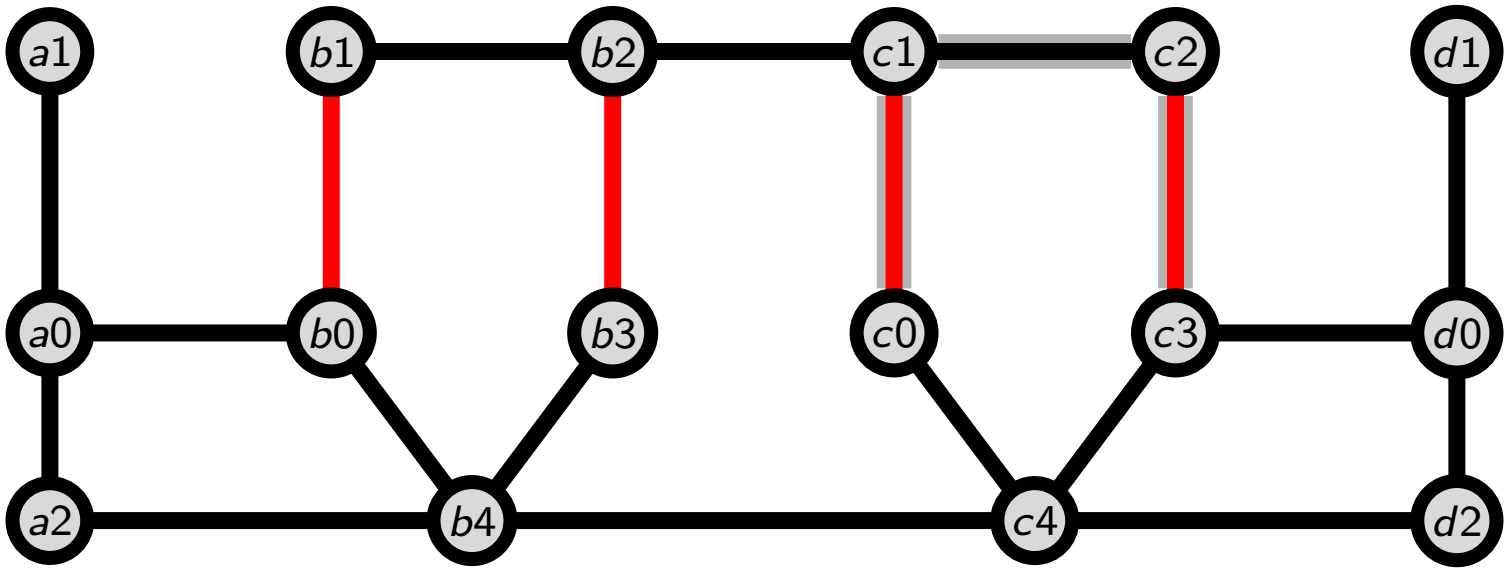
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

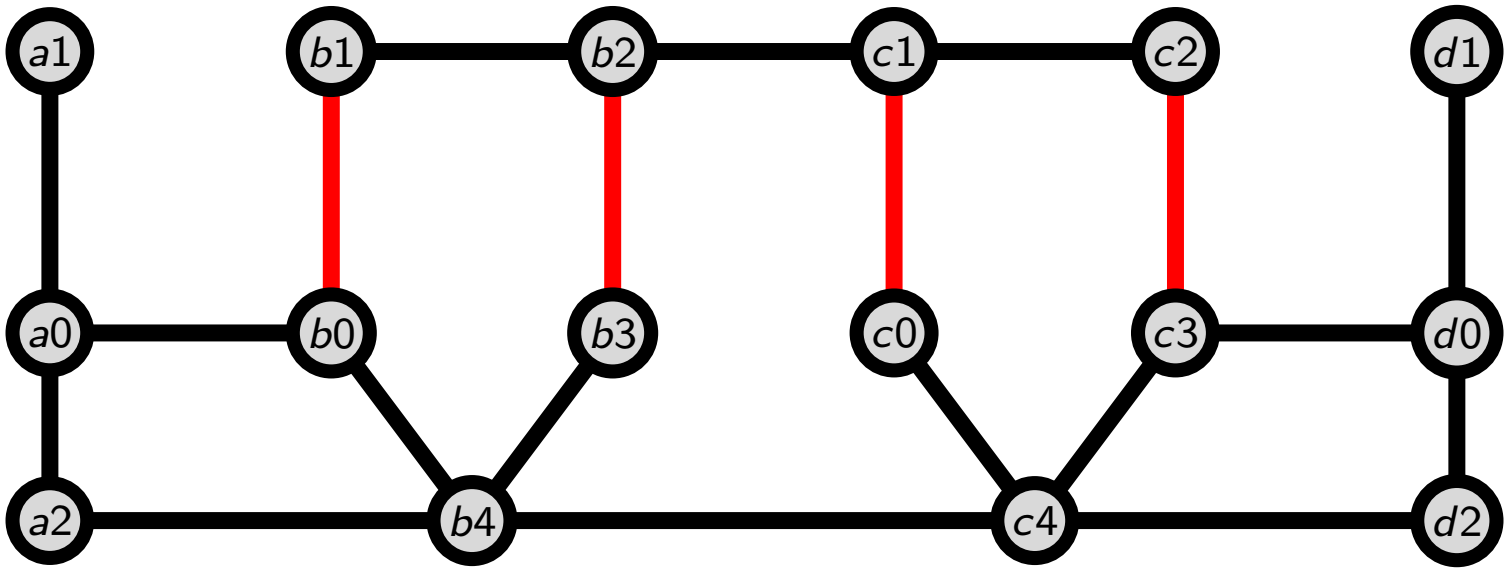
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

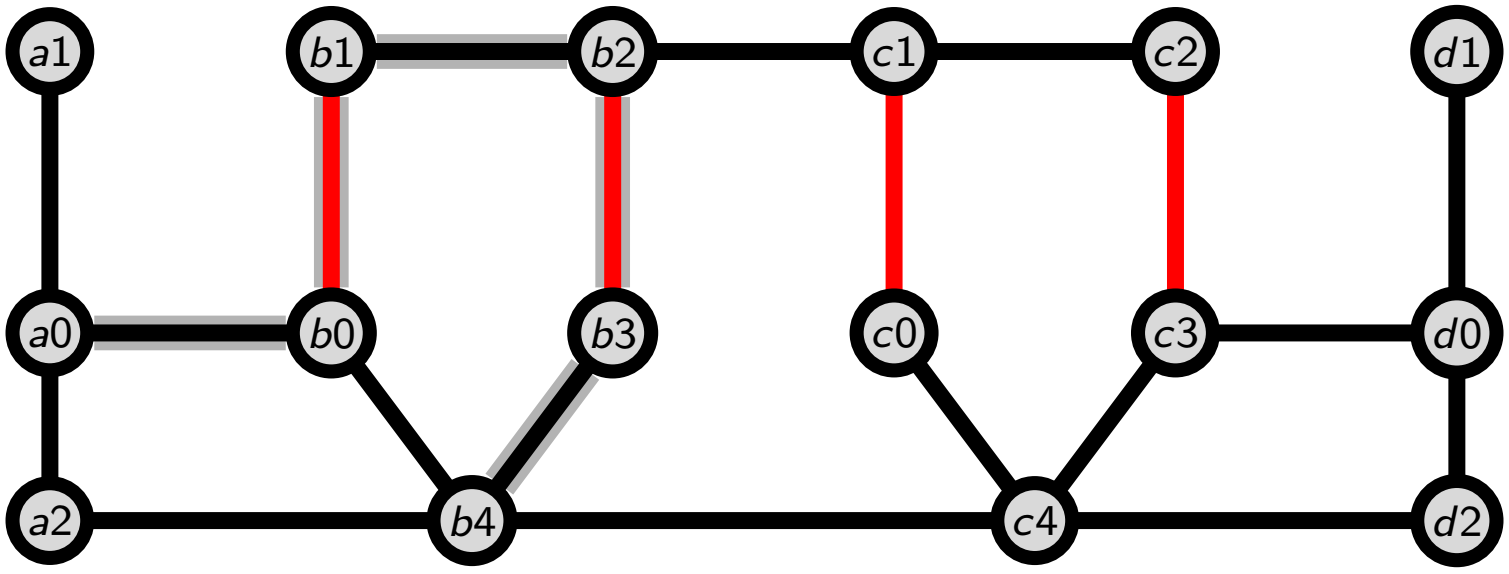
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

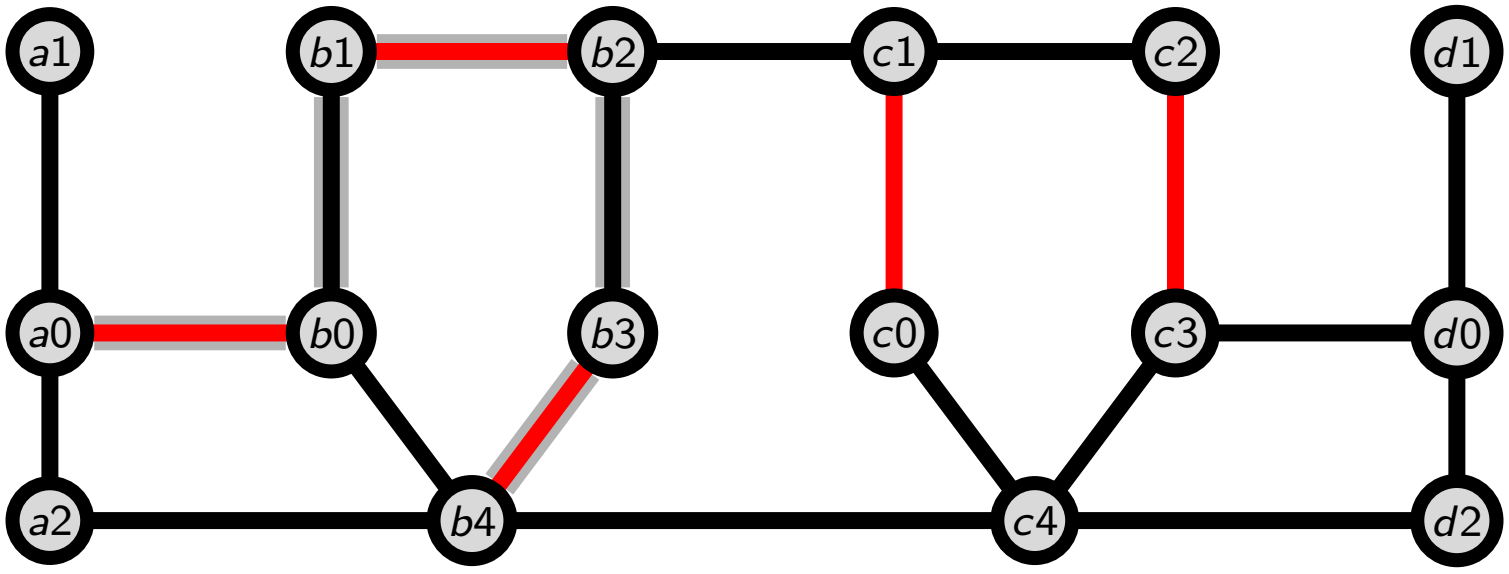
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

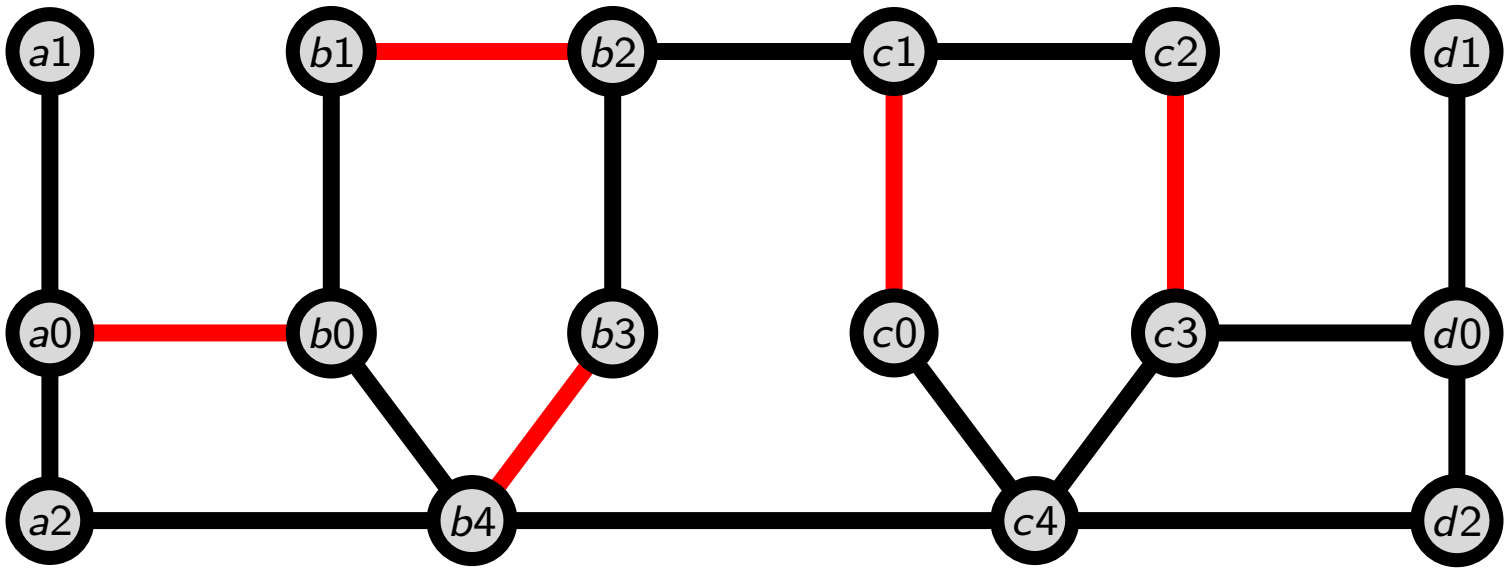
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

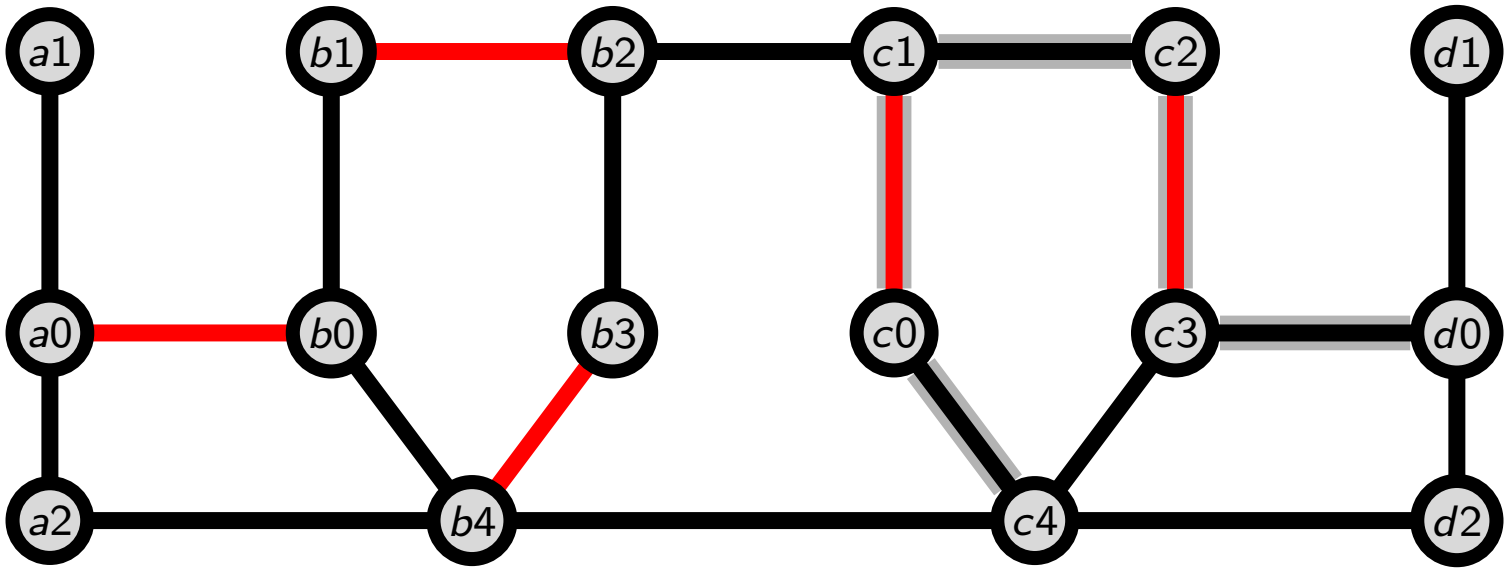
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

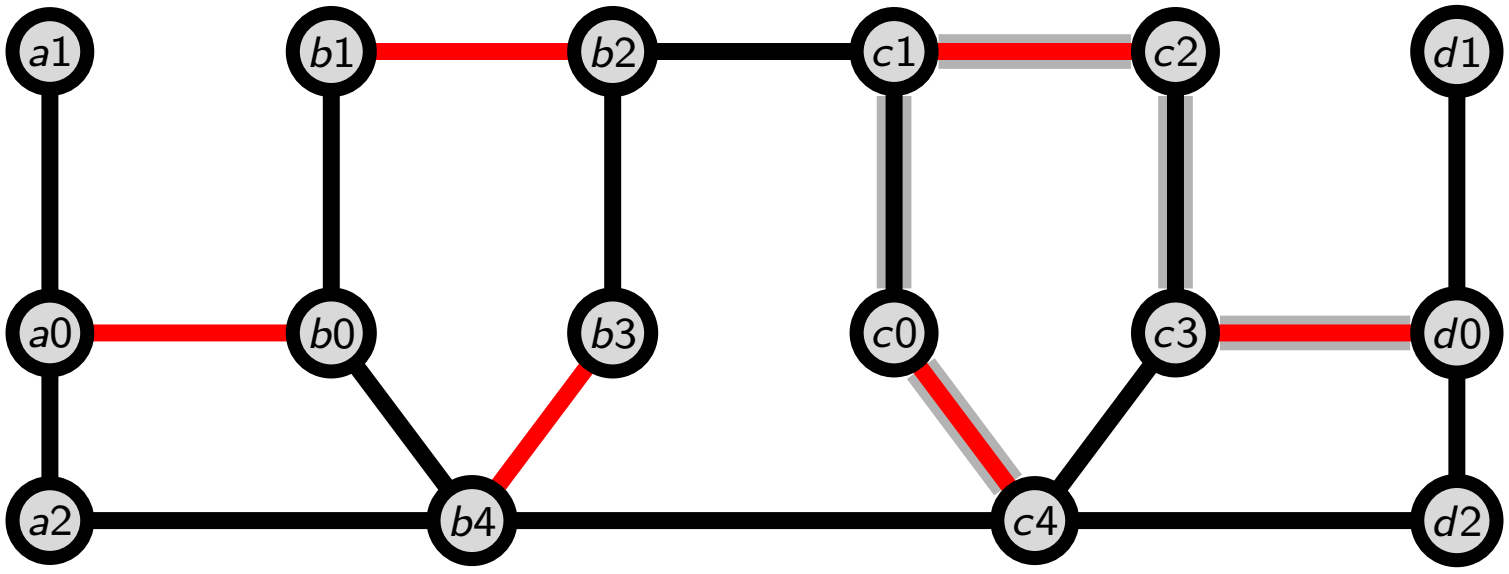
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

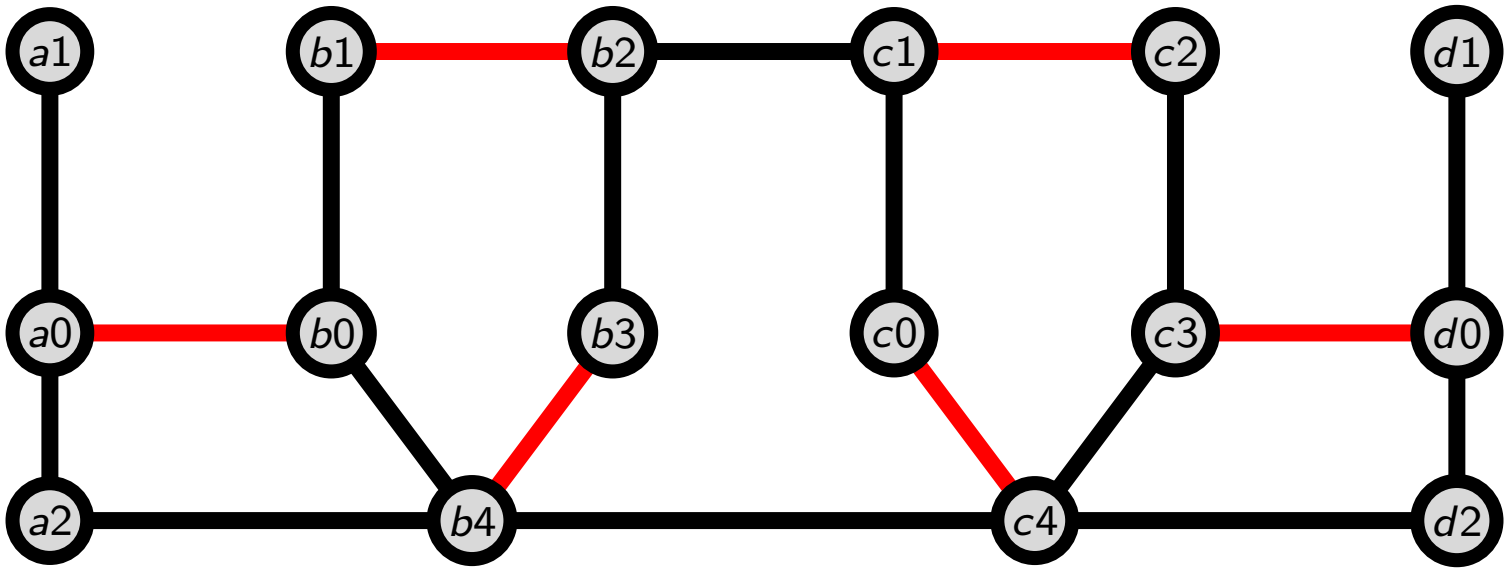
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

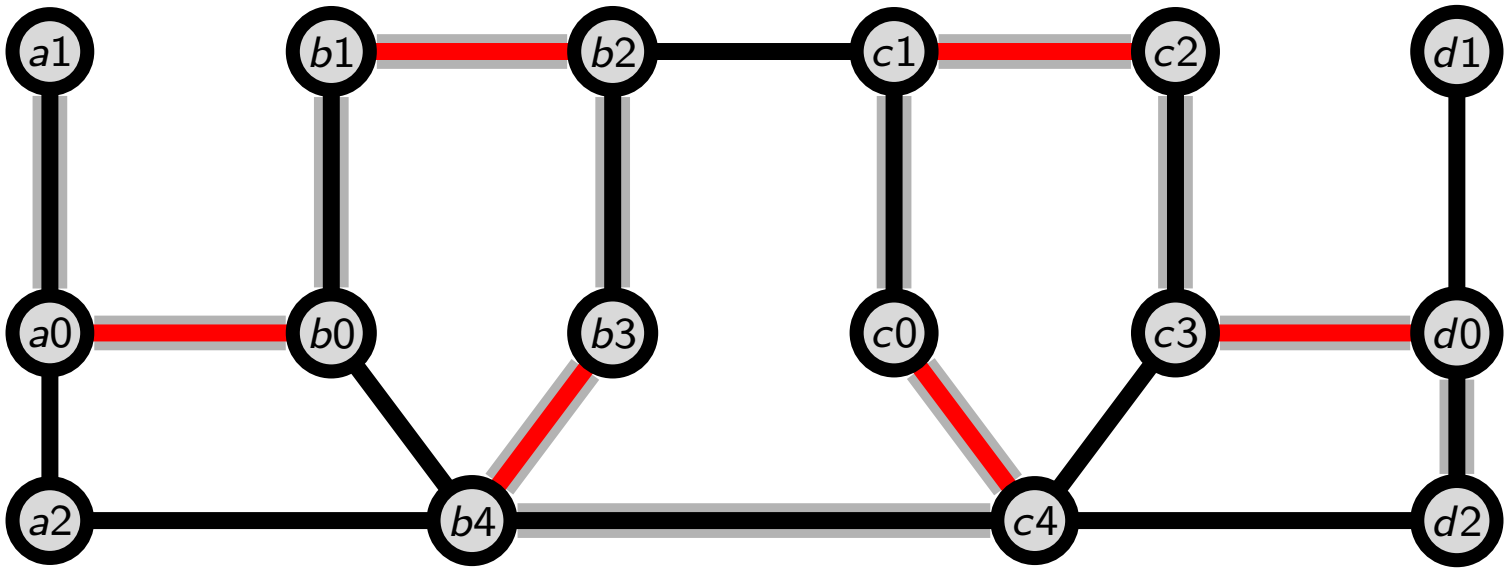
Versuche verbessernde Pfade auf allgemeinen Graphen:



Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:

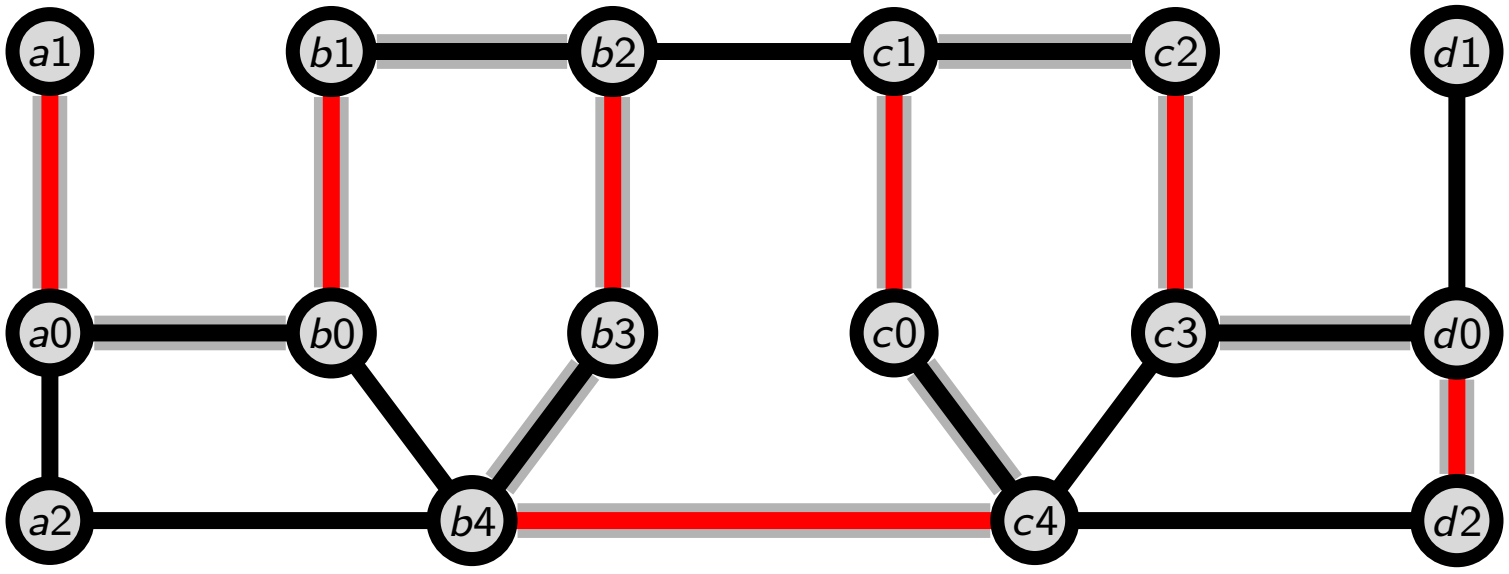


Ungerade Kreise können Probleme machen

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:

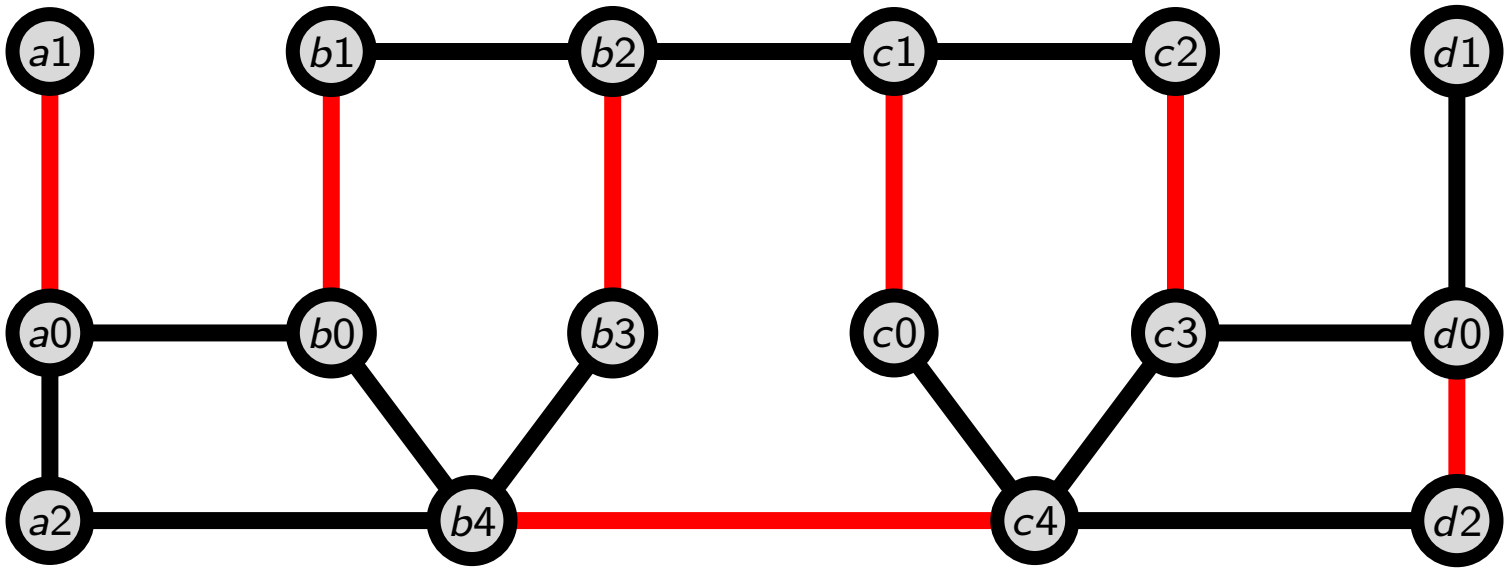


Ungerade Kreise können Probleme machen

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:

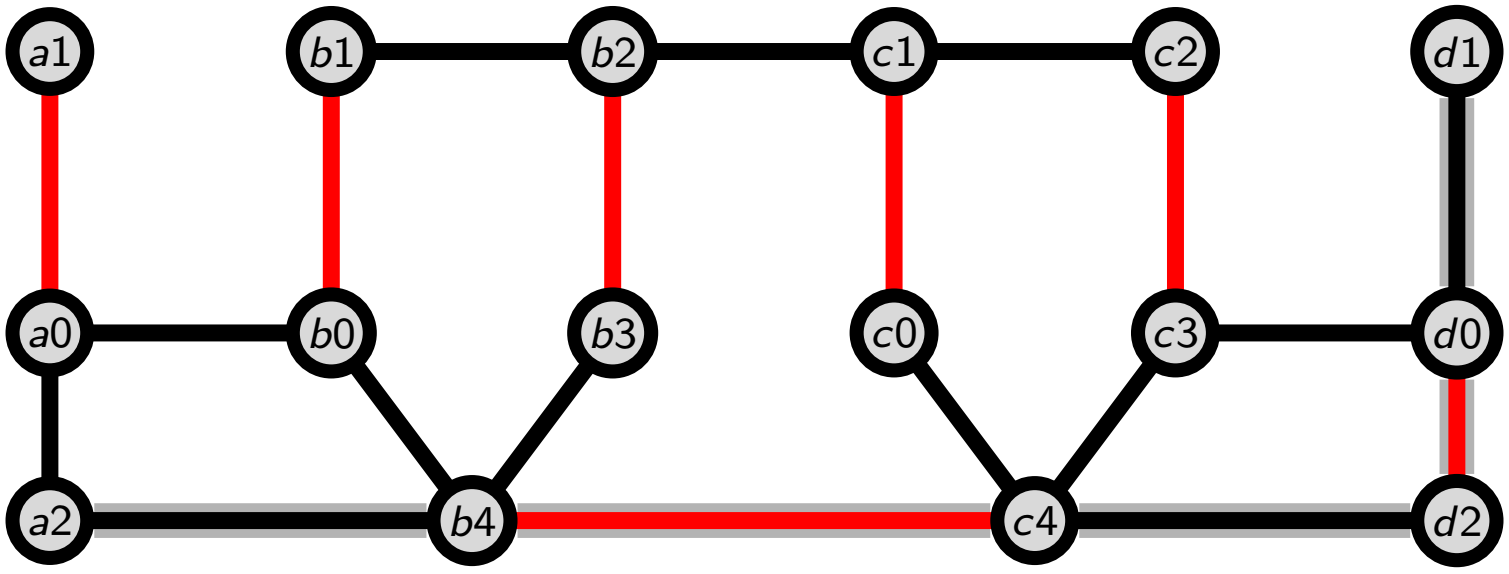


Ungerade Kreise können Probleme machen

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:

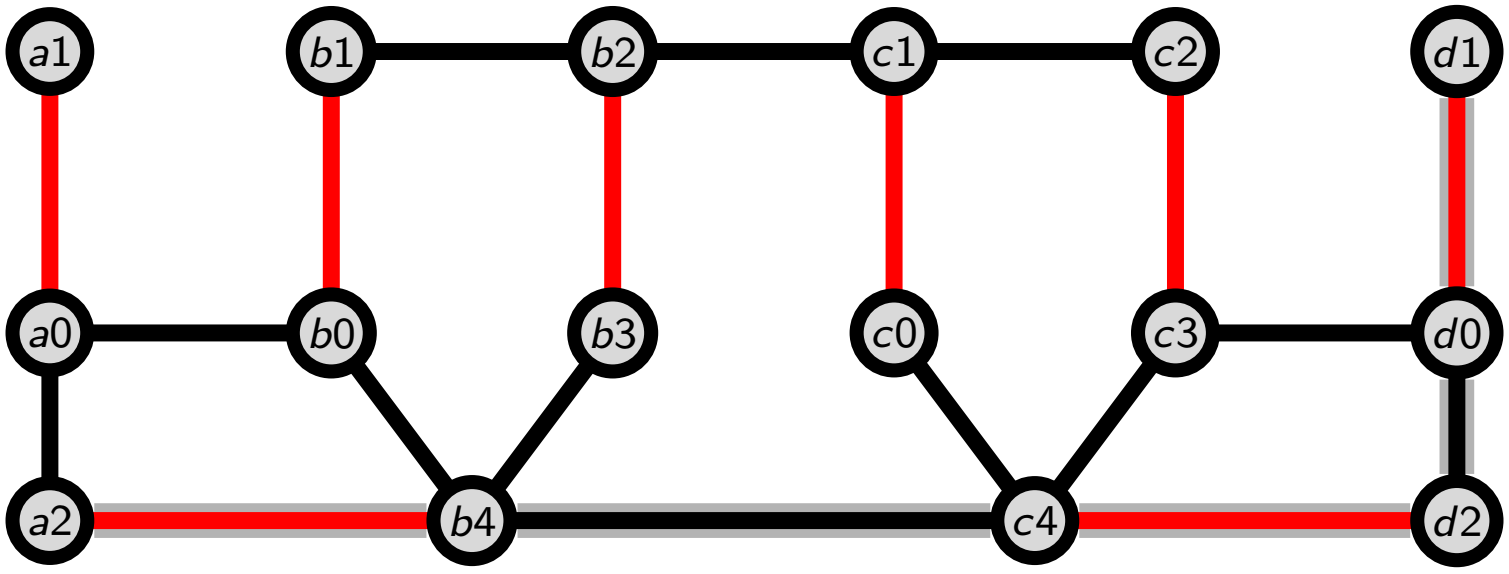


Ungerade Kreise können Probleme machen

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:

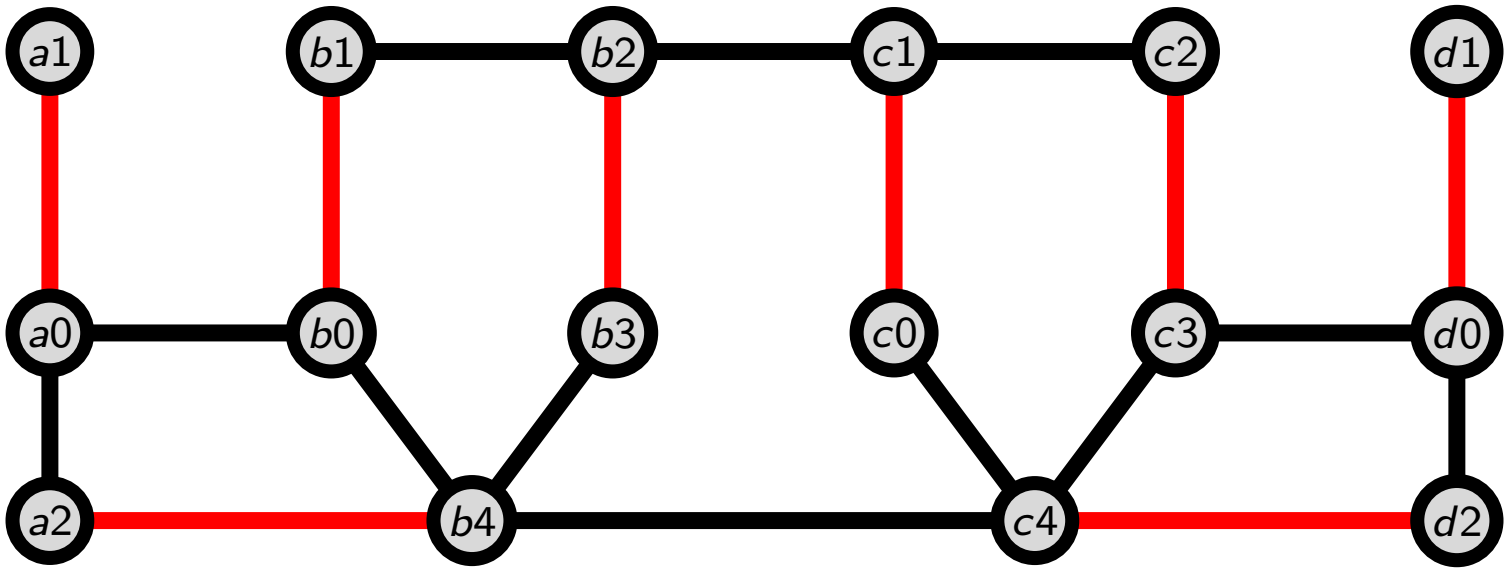


Ungerade Kreise können Probleme machen

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:



Ungerade Kreise können Probleme machen

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.
- Sei v der Startknoten von P, dann gilt $\delta_{G'}(v) = 0$ und $\delta_{G''}(v) = 1$.

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.
- Sei v der Startknoten von P, dann gilt $\delta_{G'}(v) = 0$ und $\delta_{G''}(v) = 1$.
- Sei w der Endknoten von P, dann gilt $\delta_{G'}(w) = 0$ und $\delta_{G''}(w) = 1$.

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.
- Sei v der Startknoten von P, dann gilt $\delta_{G'}(v) = 0$ und $\delta_{G''}(v) = 1$.
- Sei w der Endknoten von P, dann gilt $\delta_{G'}(w) = 0$ und $\delta_{G''}(w) = 1$.
- Sei u ein Zwischenknoten auf P, dann gilt $\delta_{G'}(u) = \delta_{G''}(u) = 1$.

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.
- Sei v der Startknoten von P, dann gilt $\delta_{G'}(v) = 0$ und $\delta_{G''}(v) = 1$.
- Sei w der Endknoten von P, dann gilt $\delta_{G'}(w) = 0$ und $\delta_{G''}(w) = 1$.
- Sei u ein Zwischenknoten auf P, dann gilt $\delta_{G'}(u) = \delta_{G''}(u) = 1$.
- Es wird eine Kante mehr hinzugefügt als entfernt.

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.
- Sei v der Startknoten von P, dann gilt $\delta_{G'}(v) = 0$ und $\delta_{G''}(v) = 1$.
- Sei w der Endknoten von P, dann gilt $\delta_{G'}(w) = 0$ und $\delta_{G''}(w) = 1$.
- Sei u ein Zwischenknoten auf P, dann gilt $\delta_{G'}(u) = \delta_{G''}(u) = 1$.
- Es wird eine Kante mehr hinzugefügt als entfernt.

Aussagen zu verbessernden Pfaden

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

Aussagen zu verbessernden Pfaden

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.

Aussagen zu verbessernden Pfaden

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.

Aussagen zu verbessernden Pfaden

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder
  - G_i ist ein Kreis gerader Länge oder

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder
  - G_i ist ein Kreis gerader Länge oder
  - G_i ist ein Pfad gerader Länge oder

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder
  - G_i ist ein Kreis gerader Länge oder
  - G_i ist ein Pfad gerader Länge oder
  - G_i ist ein Pfad ungerader Länge.

Aussagen zu verbessernden Pfaden

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder
  - G_i ist ein Kreis gerader Länge oder
  - G_i ist ein Pfad gerader Länge oder
  - G_i ist ein Pfad ungerader Länge.
- Die Kanten in G_i stammen alternierend aus M und N.

Aussagen zu verbessernden Pfaden

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder
  - G_i ist ein Kreis gerader Länge oder
  - G_i ist ein Pfad gerader Länge oder
  - G_i ist ein Pfad ungerader Länge.
- Die Kanten in G_i stammen alternierend aus M und N.

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.

Fortsetzung

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.
- Zusammengefasst gibt es mindestens $|N| - |M|$ verbessernde Pfade.

Fortsetzung

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.
- Zusammengefasst gibt es mindestens $|N| - |M|$ verbessernde Pfade.
- Nach Definition der Komponenten sind diese knotendisjunkt.

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.
- Zusammengefasst gibt es mindestens $|N| - |M|$ verbessernde Pfade.
- Nach Definition der Komponenten sind diese knotendisjunkt.

Fortsetzung

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.
- Zusammengefasst gibt es mindestens $|N| - |M|$ verbessernde Pfade.
- Nach Definition der Komponenten sind diese knotendisjunkt.

Lemma

Sei $G = (V, E)$ und M Matching in G.
M ist maximum Matching genau dann, wenn keinen verbessernden Pfad bezüglich M gibt.

Fortsetzung

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.
- Zusammengefasst gibt es mindestens $|N| - |M|$ verbessernde Pfade.
- Nach Definition der Komponenten sind diese knotendisjunkt.

Lemma

Sei $G = (V, E)$ und M Matching in G.
M ist maximum Matching genau dann, wenn keinen verbessernden Pfad bezüglich M gibt.

Algorithmus

1. Gegeben $G = (V, W, E)$

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

Theorem

Der obige Algorithmus hat eine Laufzeit von $O(n \cdot m)$.

Beweis:

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

Theorem

Der obige Algorithmus hat eine Laufzeit von $O(n \cdot m)$.

Beweis:

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

Theorem

Der obige Algorithmus hat eine Laufzeit von $O(n \cdot m)$.

Beweis:

- Wegen $|M| \leqslant \lfloor n/2 \rfloor$ gibt es höchstens $\lfloor n/2 \rfloor$ Schleifendurchläufe.

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

Theorem

Der obige Algorithmus hat eine Laufzeit von $O(n \cdot m)$.

Beweis:

- Wegen $|M| \leqslant \lfloor n/2 \rfloor$ gibt es höchstens $\lfloor n/2 \rfloor$ Schleifendurchläufe.
- Jede Schleife kann in Zeit $O(m)$ ausgeführt werden.

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

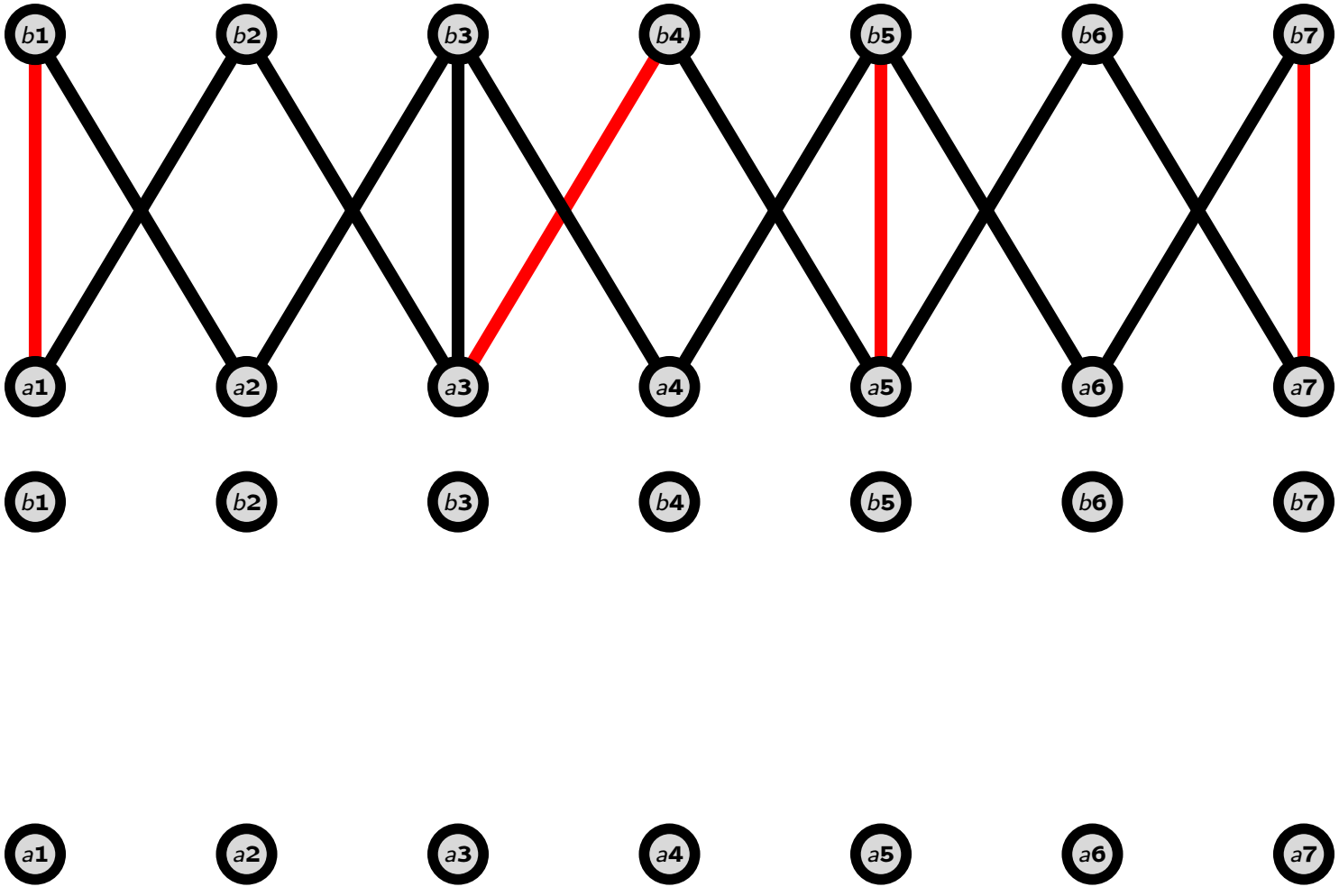
Theorem

Der obige Algorithmus hat eine Laufzeit von $O(n \cdot m)$.

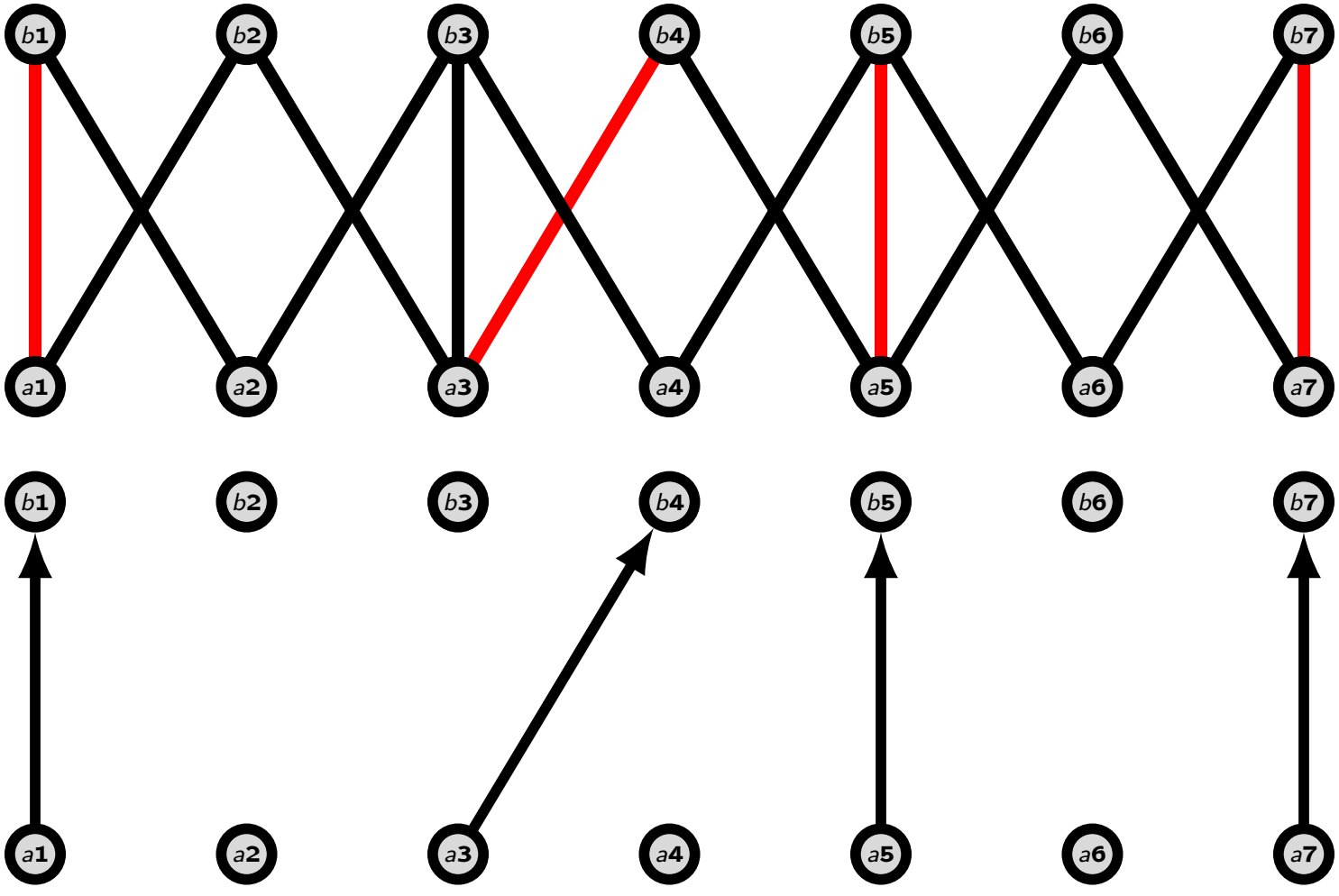
Beweis:

- Wegen $|M| \leqslant \lfloor n/2 \rfloor$ gibt es höchstens $\lfloor n/2 \rfloor$ Schleifendurchläufe.
- Jede Schleife kann in Zeit $O(m)$ ausgeführt werden.

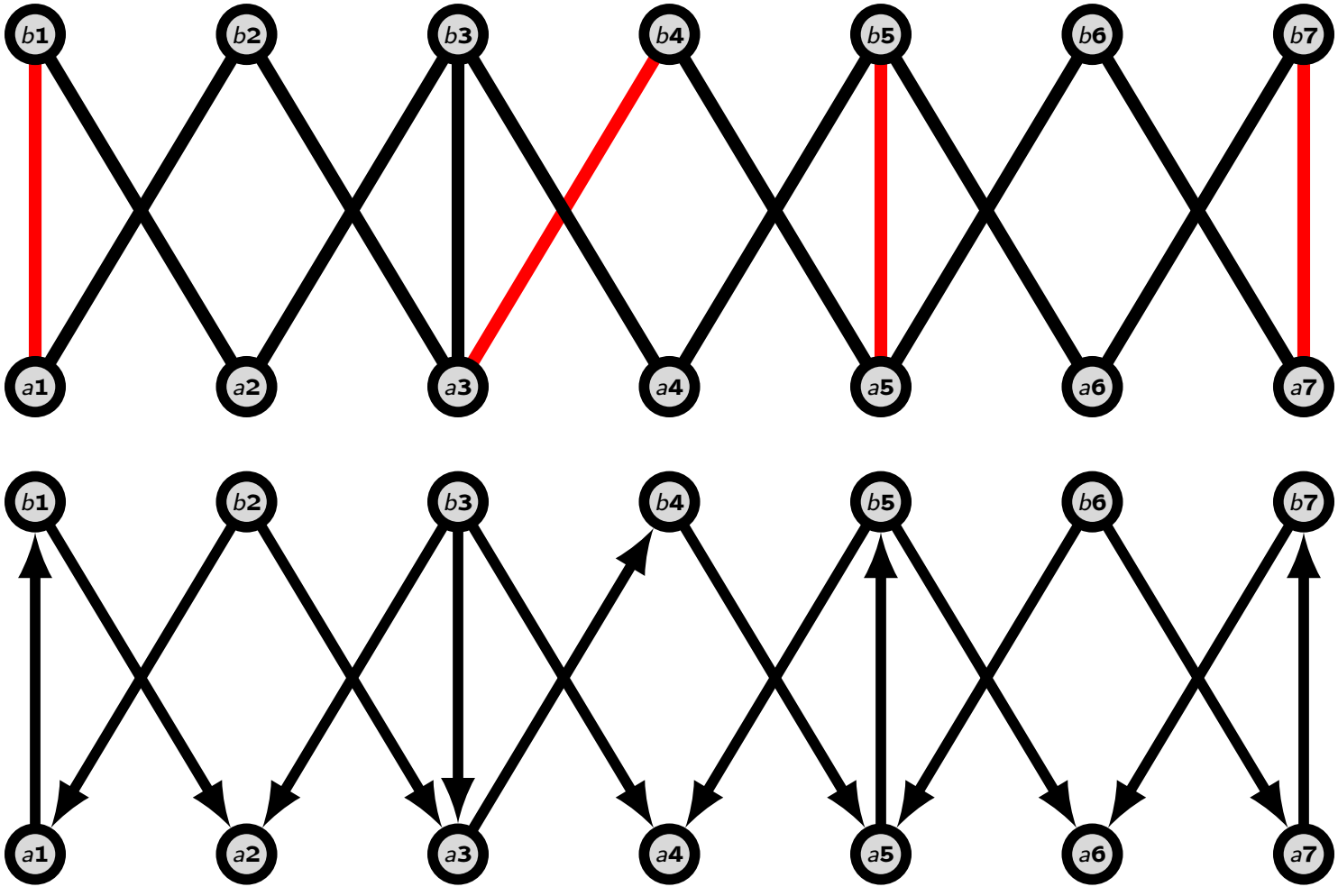
Beispiel (Schleifendurchlauf)



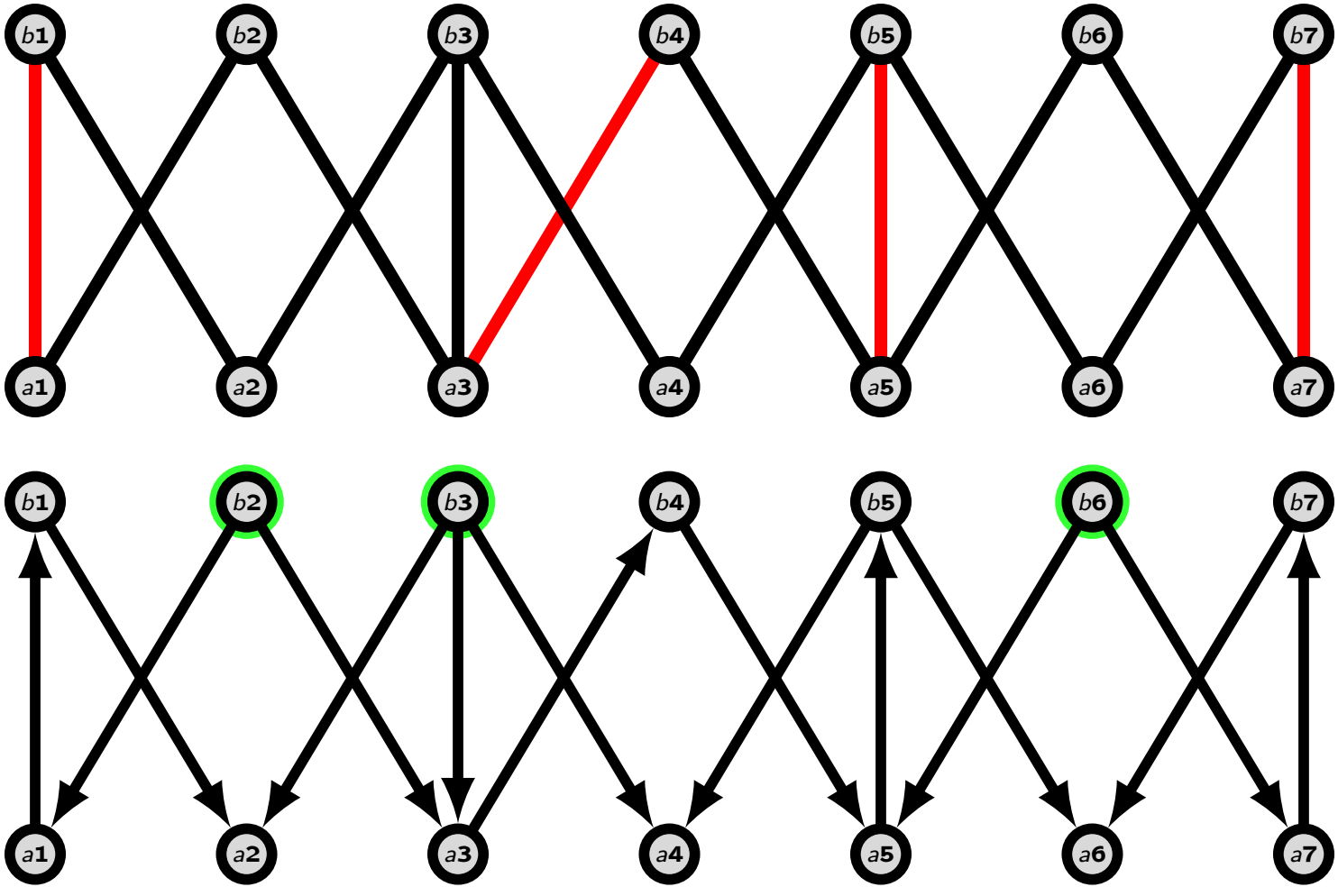
Beispiel (Schleifendurchlauf)



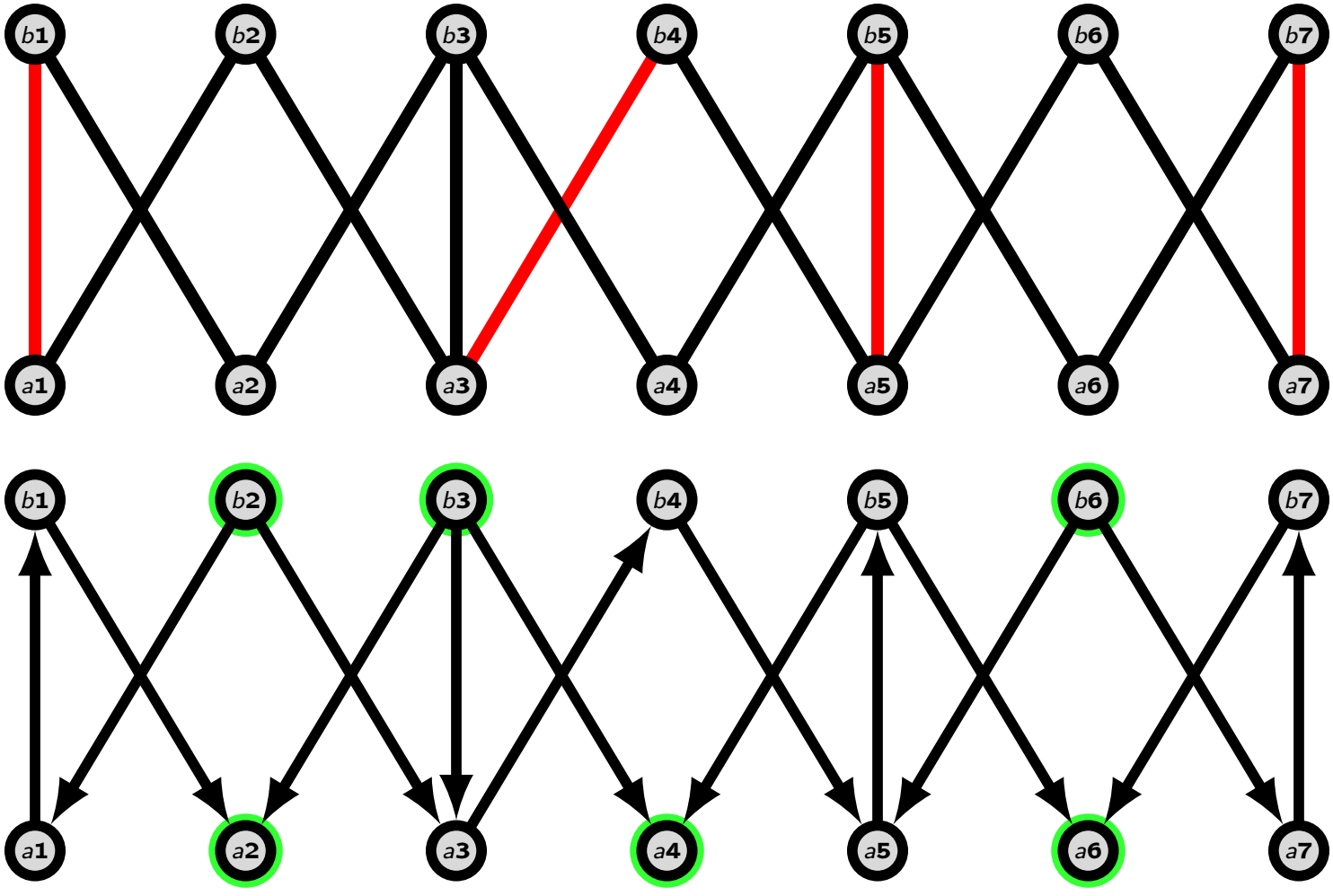
Beispiel (Schleifendurchlauf)



Beispiel (Schleifendurchlauf)



Beispiel (Schleifendurchlauf)



Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  - Von einem freien Knoten in V zu

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  - Von einem freien Knoten in V zu
  - einem freien Knoten in W.

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  - Von einem freien Knoten in V zu
  - einem freien Knoten in W.
- Damit ergibt sich eine Laufzeit von $O(m)$ durch Tiefensuche.

Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  - Von einem freien Knoten in V zu
  - einem freien Knoten in W.
- Damit ergibt sich eine Laufzeit von $O(m)$ durch Tiefensuche.

Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.

Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.

Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.

Idee

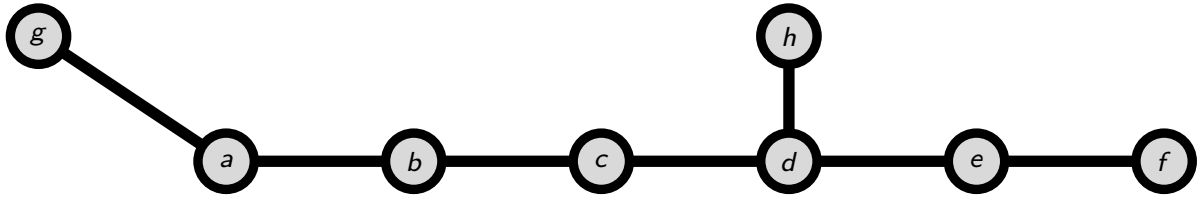
- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.

Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.

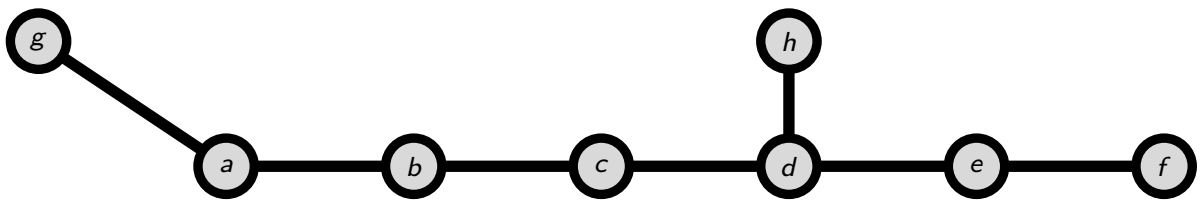
Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



Idee

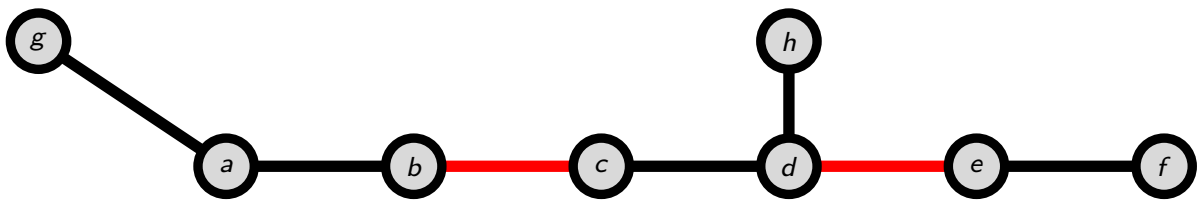
- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$

Idee

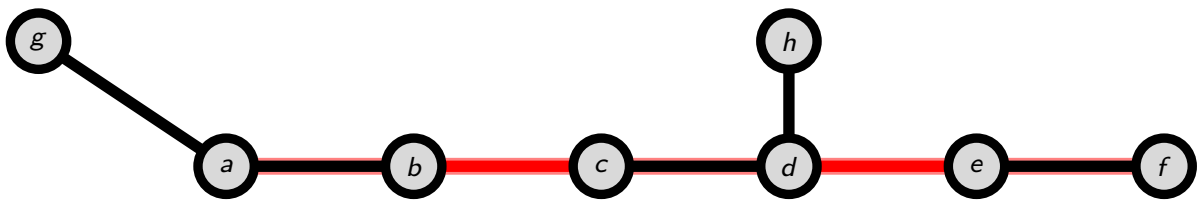
- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$
  - Erster kürzester verbessernder Pfad: a, b, c, d, e, f

Idee

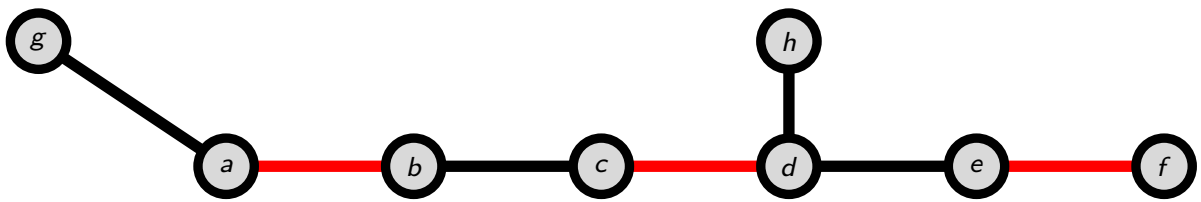
- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$
  - Erster kürzester verbessernder Pfad: a, b, c, d, e, f

Idee

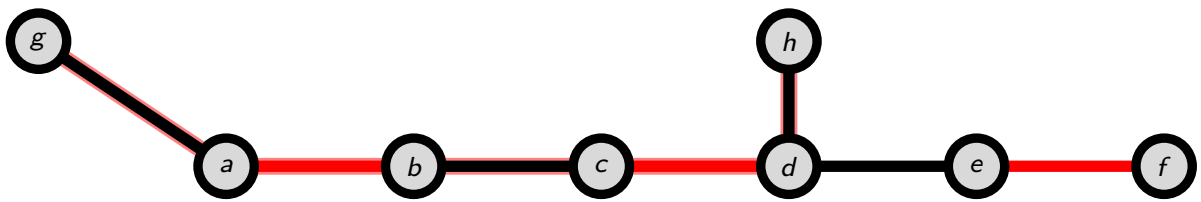
- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$
  - Erster kürzester verbessernder Pfad: a, b, c, d, e, f

Idee

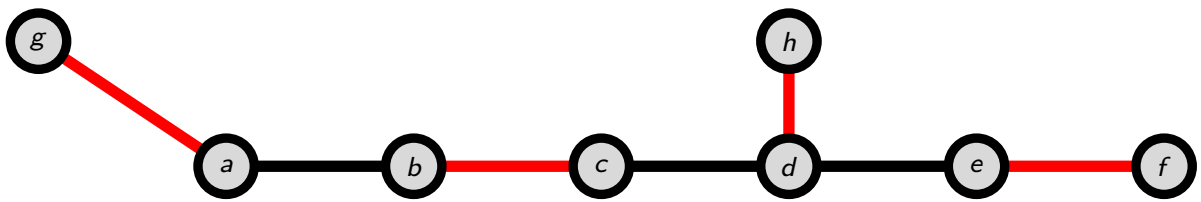
- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$
  - Erster kürzester verbessernder Pfad: a, b, c, d, e, f
  - Zweiter kürzester verbessernder Pfad: g, a, b, c, d, h

Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$
  - Erster kürzester verbessernder Pfad: a, b, c, d, e, f
  - Zweiter kürzester verbessernder Pfad: g, a, b, c, d, h
  - Widerspruch, da (g, a) noch kürzerer verbessernder Pfad.

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$M \oplus N \quad = \quad M \oplus ((M \oplus P) \oplus P')$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$M \oplus N \quad = \quad M \oplus ((M \oplus P) \oplus P')$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$\begin{aligned} M \oplus N &= M \oplus ((M \oplus P) \oplus P') \\ &= (M \oplus M) \oplus (P \oplus P') \end{aligned}$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$\begin{aligned} M \oplus N &= M \oplus ((M \oplus P) \oplus P') \\ &= (M \oplus M) \oplus (P \oplus P') \\ &= P \oplus P' \end{aligned}$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$\begin{aligned} M \oplus N &= M \oplus ((M \oplus P) \oplus P') \\ &= (M \oplus M) \oplus (P \oplus P') \\ &= P \oplus P' \end{aligned}$$

Damit gilt: $|P \oplus P'| \geqslant 2 \cdot |P|$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$\begin{array}{rrcl} & M \oplus N & = & M \oplus ((M \oplus P) \oplus P') \\ & & = & (M \oplus M) \oplus (P \oplus P') \\ & & = & P \oplus P' \\ \text{Damit gilt:} & |P \oplus P'| & \geqslant & 2 \cdot |P| \\ \text{Betrachte:} & |P \oplus P'| & = & |P \cup P'| - |P \cap P'| \end{array}$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$
\begin{array}{rrcl}
 & M \oplus N & = & M \oplus ((M \oplus P) \oplus P') \\
 & & = & (M \oplus M) \oplus (P \oplus P') \\
 & & = & P \oplus P' \\
\text{Damit gilt:} & |P \oplus P'| & \geqslant & 2 \cdot |P| \\
\text{Betrachte:} & |P \oplus P'| & = & |P \cup P'| - |P \cap P'| \\
 & & \leqslant & |P| + |P'| - |P \cap P'|
\end{array}
$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$
\begin{array}{rrcl}
 & M \oplus N & = & M \oplus ((M \oplus P) \oplus P') \\
 & & = & (M \oplus M) \oplus (P \oplus P') \\
 & & = & P \oplus P' \\
\text{Damit gilt:} & |P \oplus P'| & \geqslant & 2 \cdot |P| \\
\text{Betrachte:} & |P \oplus P'| & = & |P \cup P'| - |P \cap P'| \\
 & & \leqslant & |P| + |P'| - |P \cap P'| \\
\text{Damit gibt:} & 2 \cdot |P| & \leqslant & |P| + |P'| - |P \cap P'|
\end{array}
$$

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$
\begin{array}{rrcl}
 & M \oplus N & = & M \oplus ((M \oplus P) \oplus P') \\
 & & = & (M \oplus M) \oplus (P \oplus P') \\
 & & = & P \oplus P' \\
\text{Damit gilt:} & |P \oplus P'| & \geqslant & 2 \cdot |P| \\
\text{Betrachte:} & |P \oplus P'| & = & |P \cup P'| - |P \cap P'| \\
 & & \leqslant & |P| + |P'| - |P \cap P'| \\
\text{Damit gibt:} & 2 \cdot |P| & \leqslant & |P| + |P'| - |P \cap P'| \\
 & |P| + |P \cap P'| & \leqslant & |P'|
\end{array}
$$

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$
2. $|P_i| = |P_j| \Longrightarrow P_i$ *und* P_j *sind knotendisjunkt.*

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$
2. $|P_i| = |P_j| \Longrightarrow P_i$ *und* P_j *sind knotendisjunkt.*

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$
2. $|P_i| = |P_j| \Longrightarrow P_i$ *und* P_j *sind knotendisjunkt.*

Beweis

1. $|P_i| \leqslant |P_{i+1}|$ folgt aus Lemma "kurzer Weg".

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$
2. $|P_i| = |P_j| \Longrightarrow P_i$ *und* P_j *sind knotendisjunkt.*

Beweis

1. $|P_i| \leqslant |P_{i+1}|$ folgt aus Lemma "kurzer Weg".
2. Folgt per Widerspruch:

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$
2. $|P_i| = |P_j| \Longrightarrow P_i$ *und* P_j *sind knotendisjunkt.*

Beweis

1. $|P_i| \leqslant |P_{i+1}|$ folgt aus Lemma "kurzer Weg".
2. Folgt per Widerspruch:

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.
- Sei $v \in V(P_i) \cap V(P_j)$, genau die Matchingkante an v ist in $E(P_i) \cap E(P_j)$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.
- Sei $v \in V(P_i) \cap V(P_j)$, genau die Matchingkante an v ist in $E(P_i) \cap E(P_j)$.
- $|E(P_i) \cap E(P_j)| \geqslant 1$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.
- Sei $v \in V(P_i) \cap V(P_j)$, genau die Matchingkante an v ist in $E(P_i) \cap E(P_j)$.
- $|E(P_i) \cap E(P_j)| \geqslant 1$.
- Aus Lemma "kurzer Weg" folgt: $|P_j| \geqslant |P_i| + |P_i \cap P_j| \geqslant |P_i| + 1$.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.
- Sei $v \in V(P_i) \cap V(P_j)$, genau die Matchingkante an v ist in $E(P_i) \cap E(P_j)$.
- $|E(P_i) \cap E(P_j)| \geqslant 1$.
- Aus Lemma "kurzer Weg" folgt: $|P_j| \geqslant |P_i| + |P_i \cap P_j| \geqslant |P_i| + 1$.
- Widerspruch.

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.
- Sei $v \in V(P_i) \cap V(P_j)$, genau die Matchingkante an v ist in $E(P_i) \cap E(P_j)$.
- $|E(P_i) \cap E(P_j)| \geqslant 1$.
- Aus Lemma "kurzer Weg" folgt: $|P_j| \geqslant |P_i| + |P_i \cap P_j| \geqslant |P_i| + 1$.
- Widerspruch.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$
3. Solange es verbessernde Pfade gibt, mache:

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$
3. Solange es verbessernde Pfade gibt, mache:
   1. Bestimme l als die Länge des kürzesten verbessernden Pfads.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$
3. Solange es verbessernde Pfade gibt, mache:
   1. Bestimme l als die Länge des kürzesten verbessernden Pfads.
   2. Bestimme eine inklusions-maximale Menge von verbessernden Pfaden $P_1, P_2, \ldots, P_i$ der Länge l.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$
3. Solange es verbessernde Pfade gibt, mache:
   1. Bestimme l als die Länge des kürzesten verbessernden Pfads.
   2. Bestimme eine inklusions-maximale Menge von verbessernden Pfaden $P_1, P_2, \ldots, P_i$ der Länge l.
   3. $M = M \oplus P_1 \oplus P_2 \oplus P_3 \oplus \ldots \oplus P_i$.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$
3. Solange es verbessernde Pfade gibt, mache:
   1. Bestimme l als die Länge des kürzesten verbessernden Pfads.
   2. Bestimme eine inklusions-maximale Menge von verbessernden Pfaden $P_1, P_2, \ldots, P_i$ der Länge l.
   3. $M = M \oplus P_1 \oplus P_2 \oplus P_3 \oplus \ldots \oplus P_i$.

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$ und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$
und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

- Es gibt höchstens $s - |M|$ viele M-verbessernde Pfade.

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$
und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

- Es gibt höchstens $s - |M|$ viele M-verbessernde Pfade.
- Diese Pfade enthalten zusammen höchstens $|M|$ Kanten aus M.

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$
und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

- Es gibt höchstens $s - |M|$ viele M-verbessernde Pfade.
- Diese Pfade enthalten zusammen höchstens $|M|$ Kanten aus M.
- Damit gibt es einen Pfad, der höchstens $\lfloor \frac{|M|}{s-|M|} \rfloor$ Kanten aus M enthält. D.h. ein kürzester Pfad enthällt micht mehr als die Durchschnittsanzahl von Kanten aus M.

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$
und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

- Es gibt höchstens $s - |M|$ viele M-verbessernde Pfade.
- Diese Pfade enthalten zusammen höchstens $|M|$ Kanten aus M.
- Damit gibt es einen Pfad, der höchstens $\lfloor \frac{|M|}{s-|M|} \rfloor$ Kanten aus M enthält. D.h. ein kürzester Pfad enthällt micht mehr als die Durchschnittsanzahl von Kanten aus M.
- Dieser Pfad hat eine Länge von:

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$
und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

- Es gibt höchstens $s - |M|$ viele M-verbessernde Pfade.
- Diese Pfade enthalten zusammen höchstens $|M|$ Kanten aus M.
- Damit gibt es einen Pfad, der höchstens $\lfloor \frac{|M|}{s-|M|} \rfloor$ Kanten aus M enthält. D.h. ein kürzester Pfad enthällt micht mehr als die Durchschnittsanzahl von Kanten aus M.
- Dieser Pfad hat eine Länge von:

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.

Anzahl der Schleifendurchläufe

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

Anzahl der Schleifendurchläufe

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.
    - Da l in Zweierschritten wächst, gibt es höchsten $\sqrt{s}$ Iterationen in der ersten Phase.

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.
    - Da l in Zweierschritten wächst, gibt es höchsten $\sqrt{s}$ Iterationen in der ersten Phase.
  - Phase 2 aktiv, falls $|M| > \lfloor s - \sqrt{s} \rfloor$ gilt.

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.
    - Da l in Zweierschritten wächst, gibt es höchsten $\sqrt{s}$ Iterationen in der ersten Phase.
  - Phase 2 aktiv, falls $|M| > \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Es fehlen $\sqrt{s}$ viele Kanten.

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.
    - Da l in Zweierschritten wächst, gibt es höchsten $\sqrt{s}$ Iterationen in der ersten Phase.
  - Phase 2 aktiv, falls $|M| > \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Es fehlen $\sqrt{s}$ viele Kanten.
    - Die Phase endet nach maximal $\sqrt{s}$ Iterationen.

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.
    - Da l in Zweierschritten wächst, gibt es höchsten $\sqrt{s}$ Iterationen in der ersten Phase.
  - Phase 2 aktiv, falls $|M| > \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Es fehlen $\sqrt{s}$ viele Kanten.
    - Die Phase endet nach maximal $\sqrt{s}$ Iterationen.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis (Idee wird auch beim gewichteten Matching benutzt.)

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  Von den freien Knoten in V zu den freien Knoten in W.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  Von den freien Knoten in V zu den freien Knoten in W.
- Füge Quelle und Senke hinzu, damit entsteht G''.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G': Von den freien Knoten in V zu den freien Knoten in W.
- Füge Quelle und Senke hinzu, damit entsteht G''.
- Bestimme durch Breitensuche alle Kanten, die nicht auf einem kürzesten Pfad liegen.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G': Von den freien Knoten in V zu den freien Knoten in W.
- Füge Quelle und Senke hinzu, damit entsteht G''.
- Bestimme durch Breitensuche alle Kanten, die nicht auf einem kürzesten Pfad liegen.
- Damit entsteht ein Schichtennetzwerk G'''.

Laufzeit

Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G': Von den freien Knoten in V zu den freien Knoten in W.
- Füge Quelle und Senke hinzu, damit entsteht G''.
- Bestimme durch Breitensuche alle Kanten, die nicht auf einem kürzesten Pfad liegen.
- Damit entsteht ein Schichtennetzwerk G'''.
- Damit ergibt sich eine Laufzeit von $O(m)$ durch Tiefensuche auf G'''.

Laufzeit

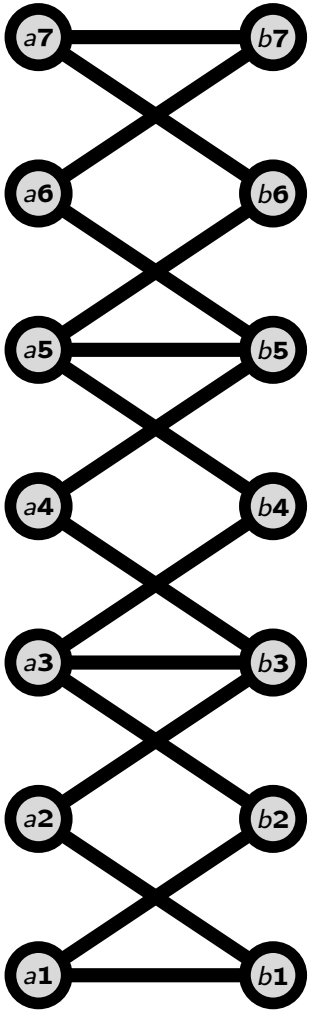
Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

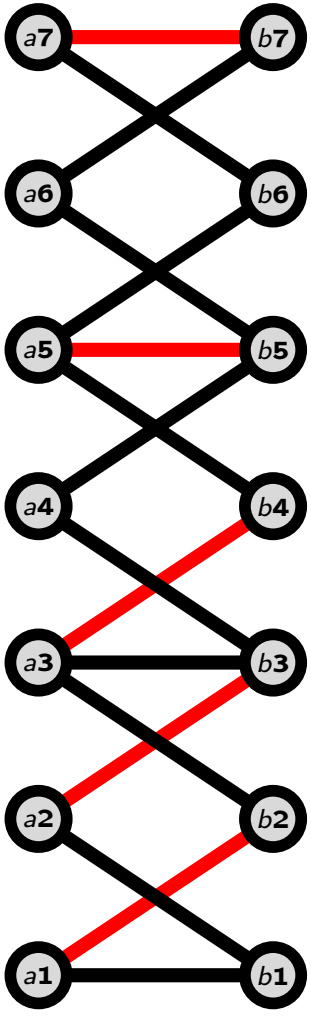
Beweis

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G': Von den freien Knoten in V zu den freien Knoten in W.
- Füge Quelle und Senke hinzu, damit entsteht G''.
- Bestimme durch Breitensuche alle Kanten, die nicht auf einem kürzesten Pfad liegen.
- Damit entsteht ein Schichtennetzwerk G'''.
- Damit ergibt sich eine Laufzeit von $O(m)$ durch Tiefensuche auf G'''.

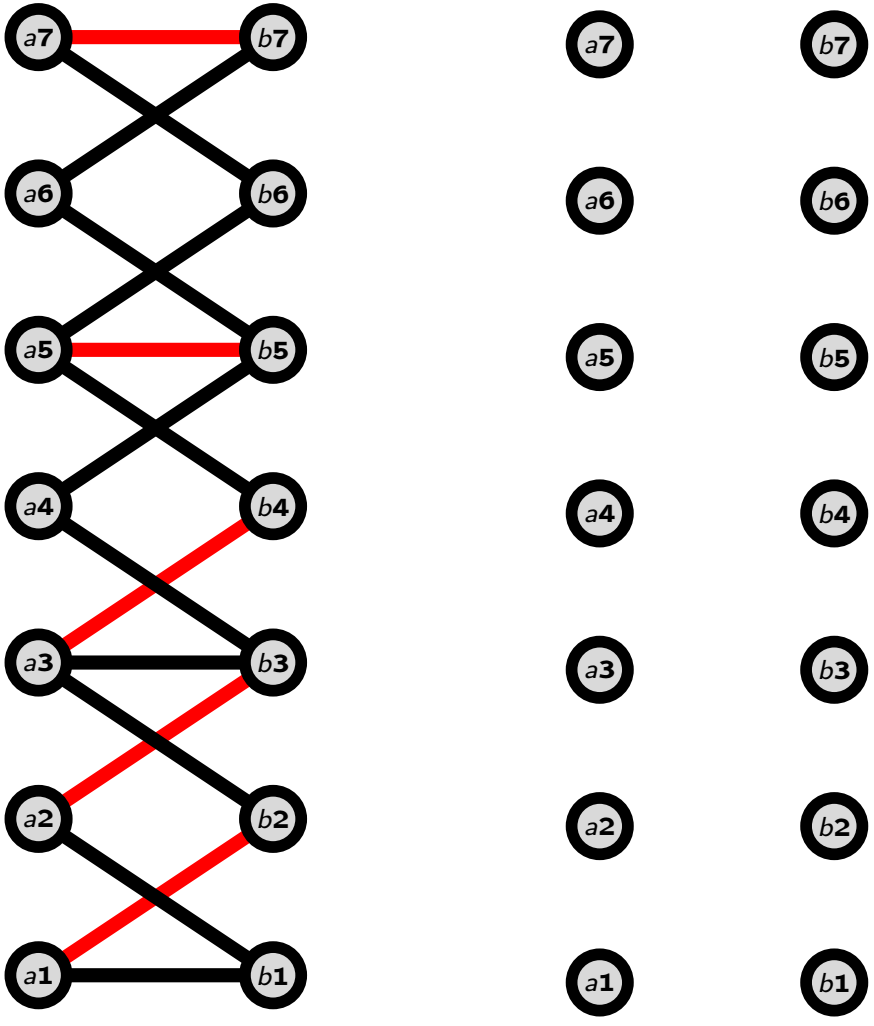
Beispiel (Bestimme G')



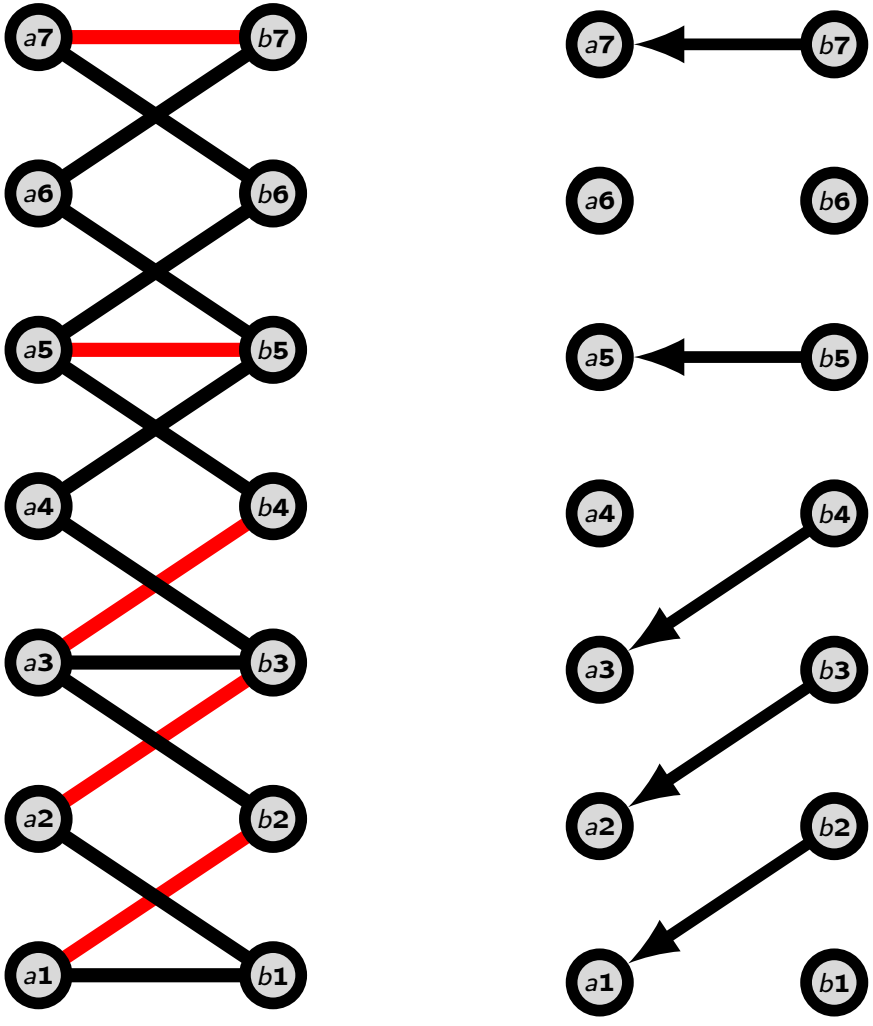
Beispiel (Bestimme G')



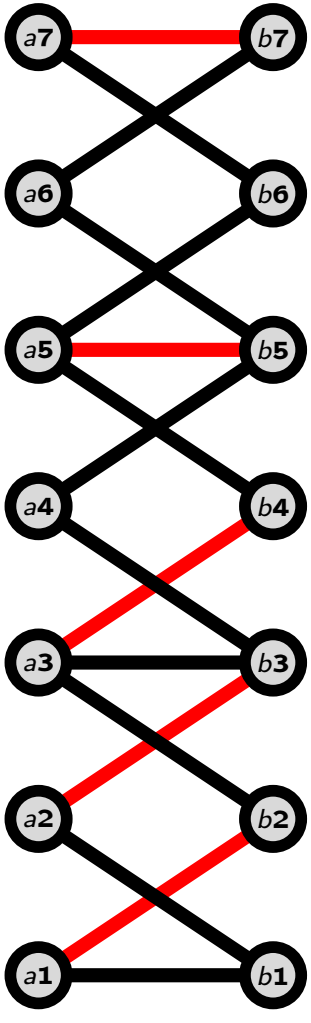
Beispiel (Bestimme G')

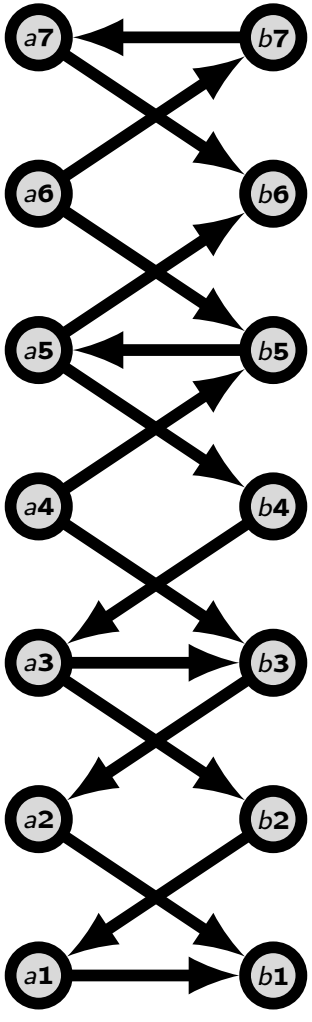


Beispiel (Bestimme G')

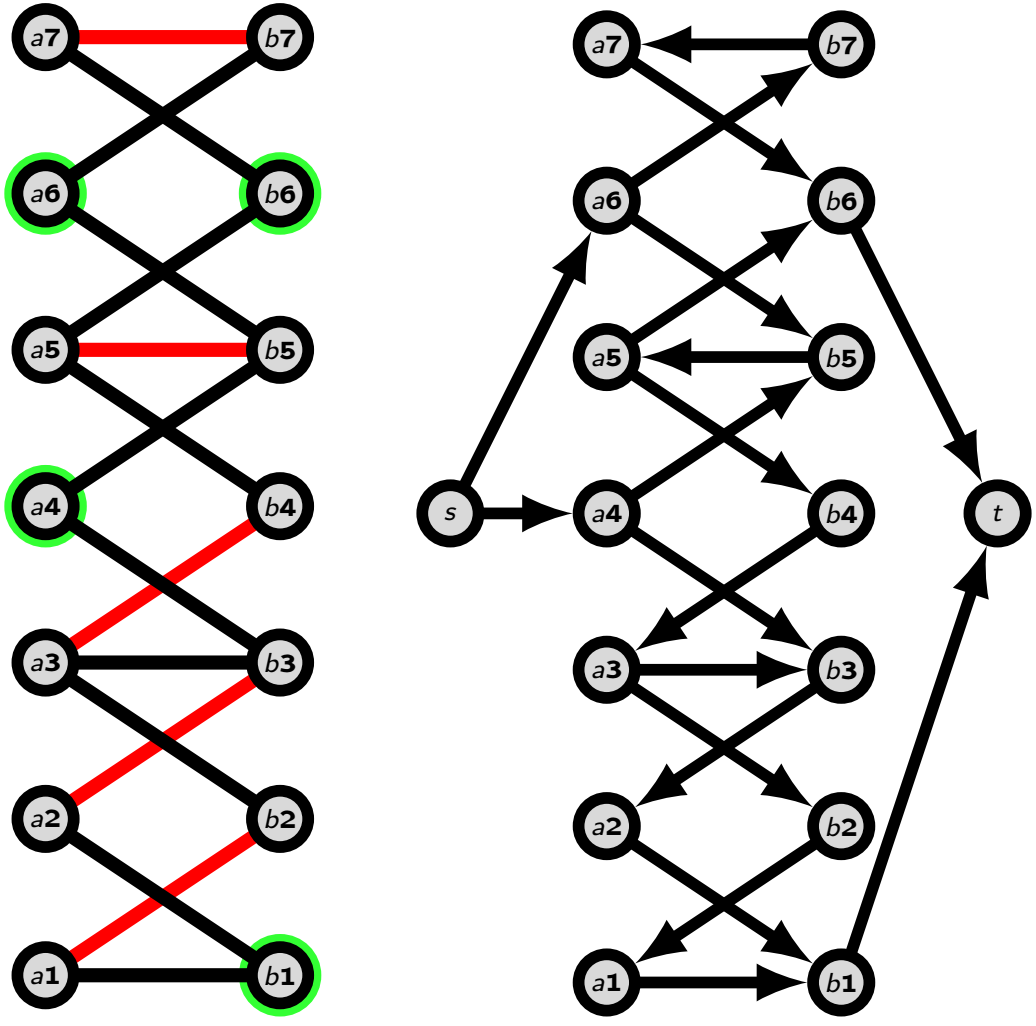


Beispiel (Bestimme G')

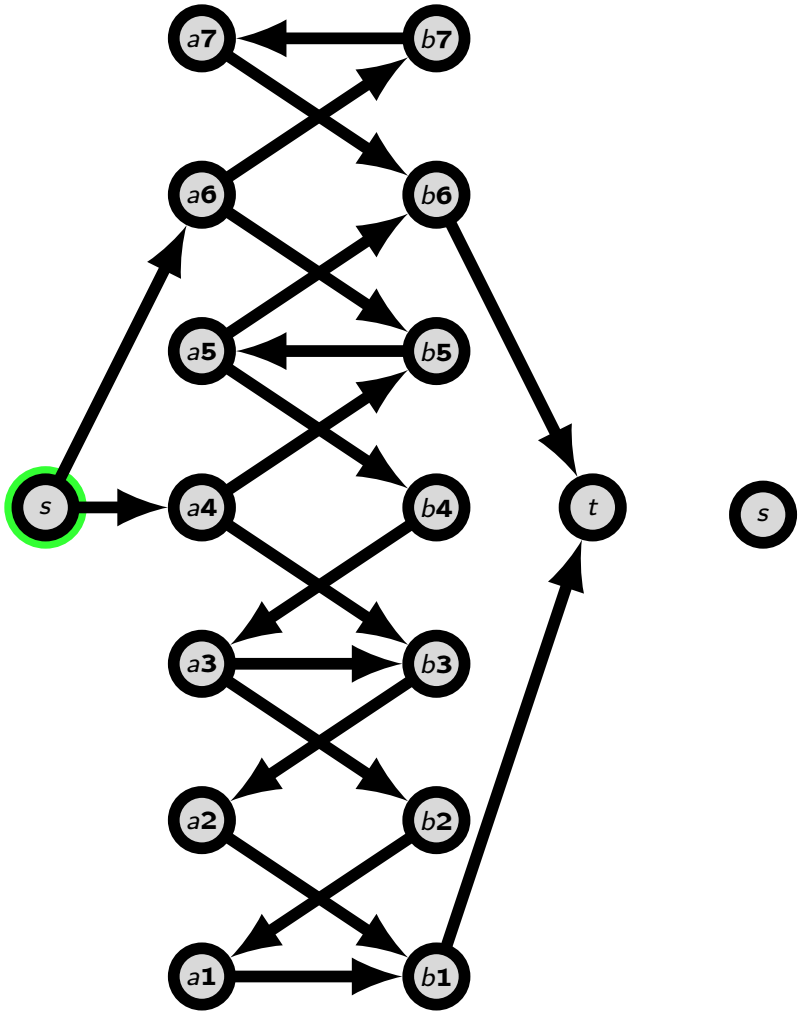




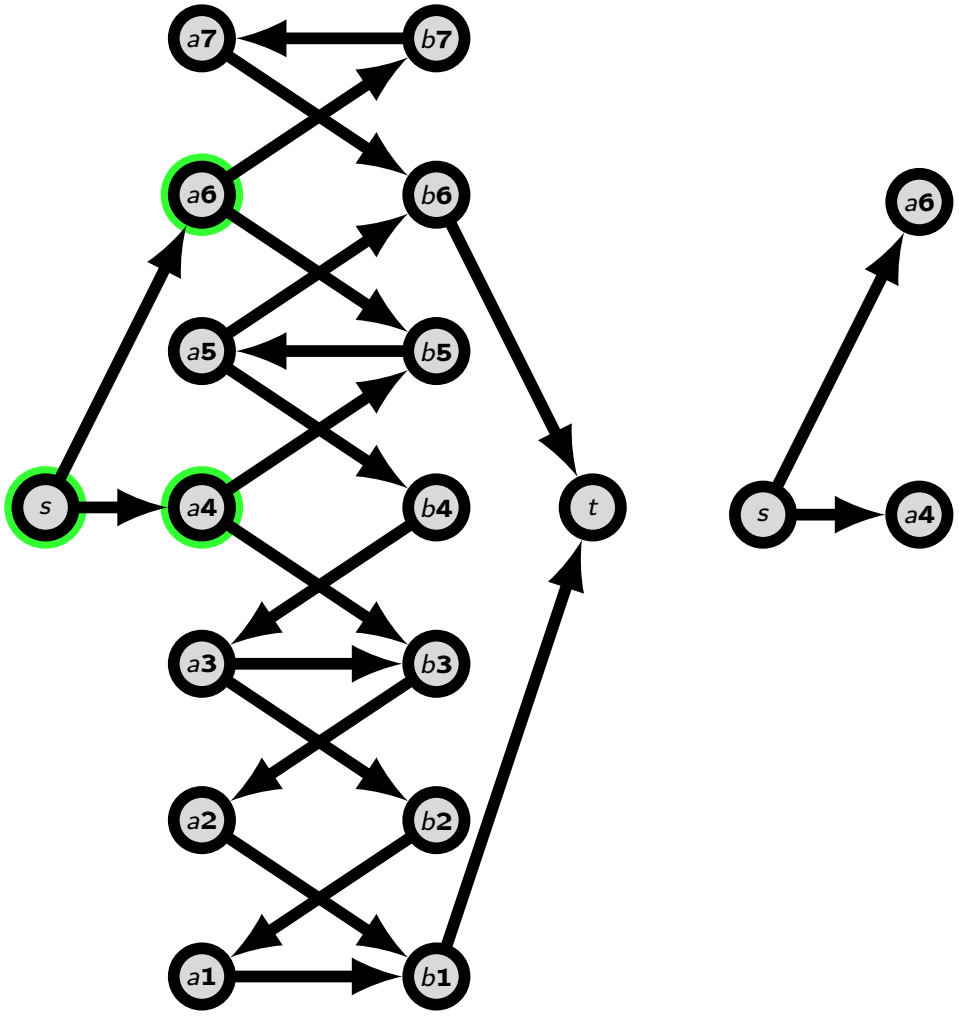
Beispiel (Bestimme G')



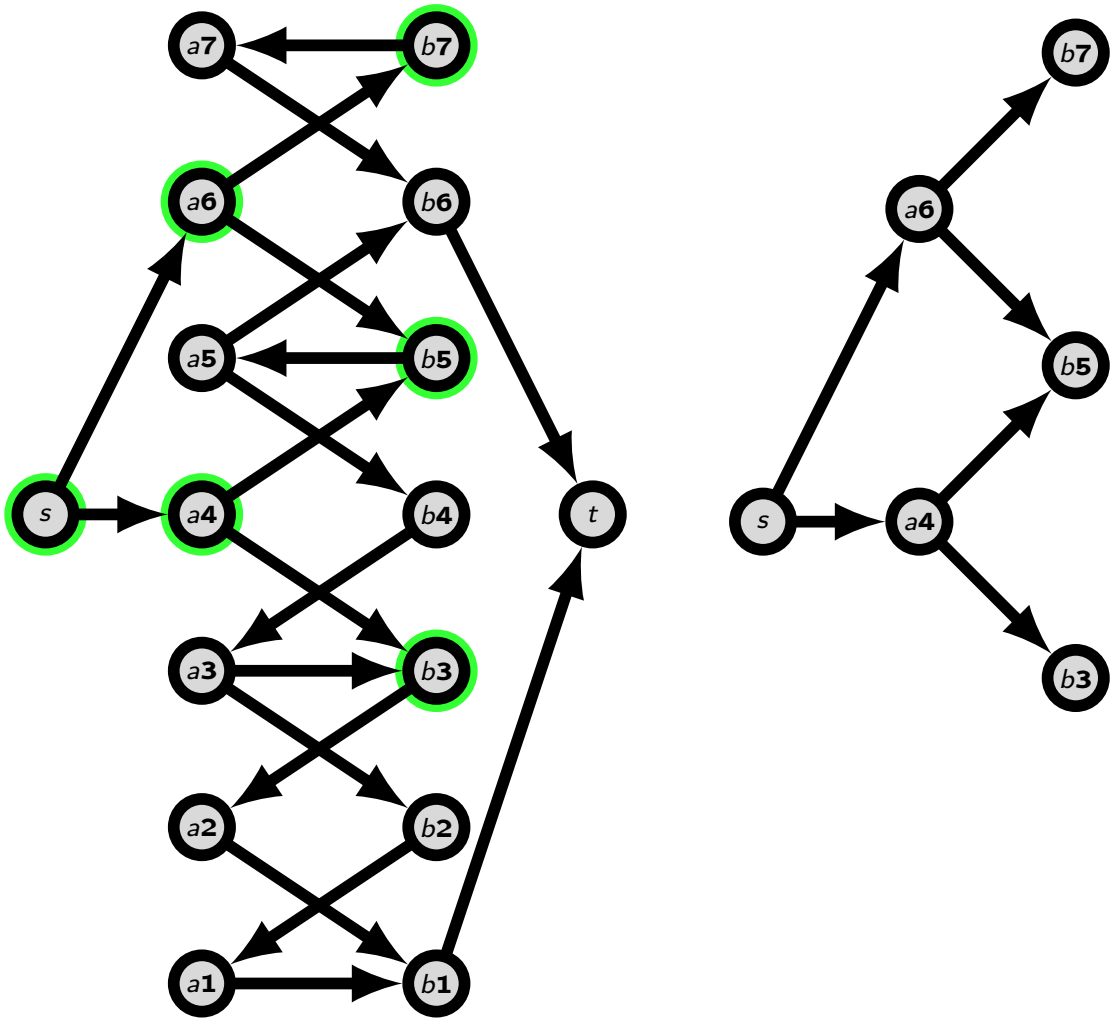
Beispiel (Breitensuche)



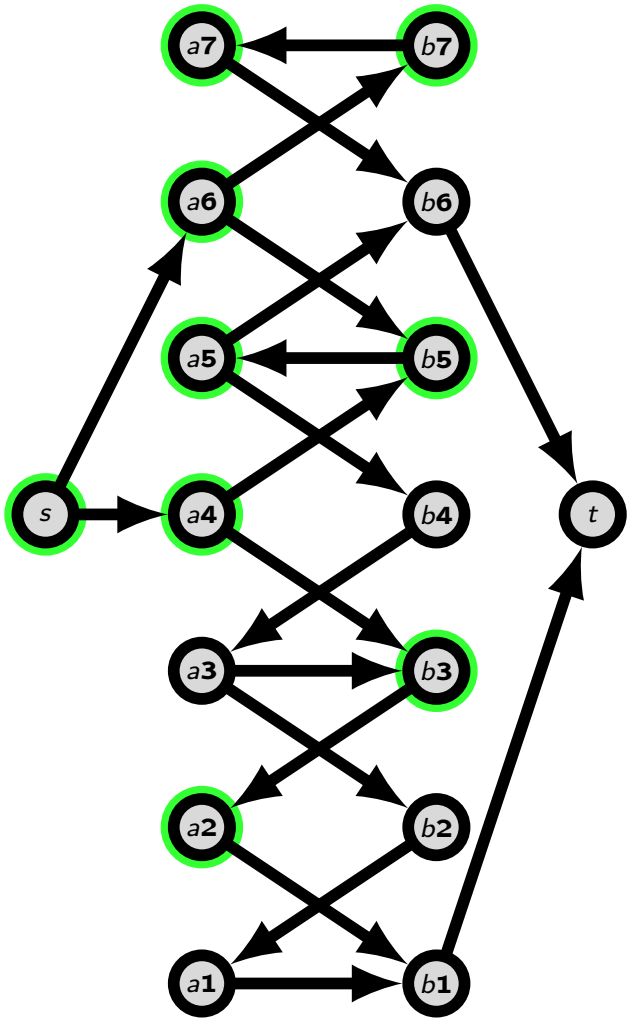
Beispiel (Breitensuche)

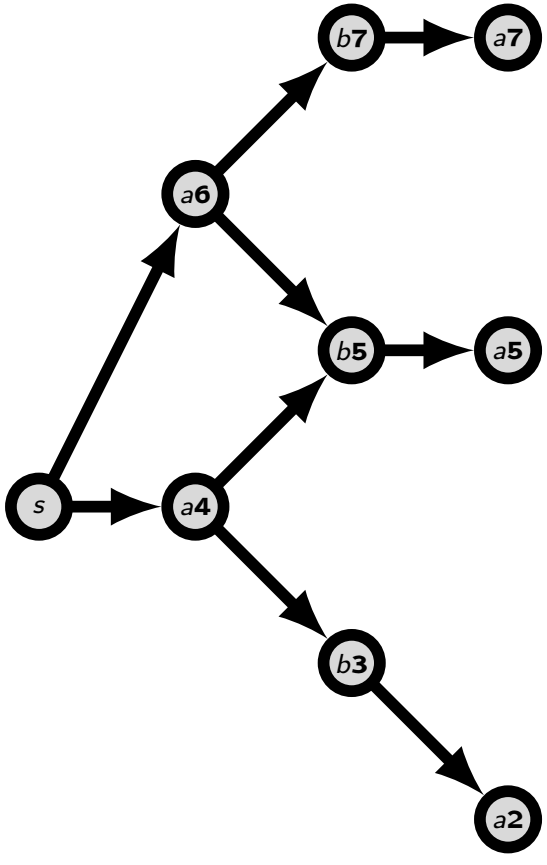


Beispiel (Breitensuche)

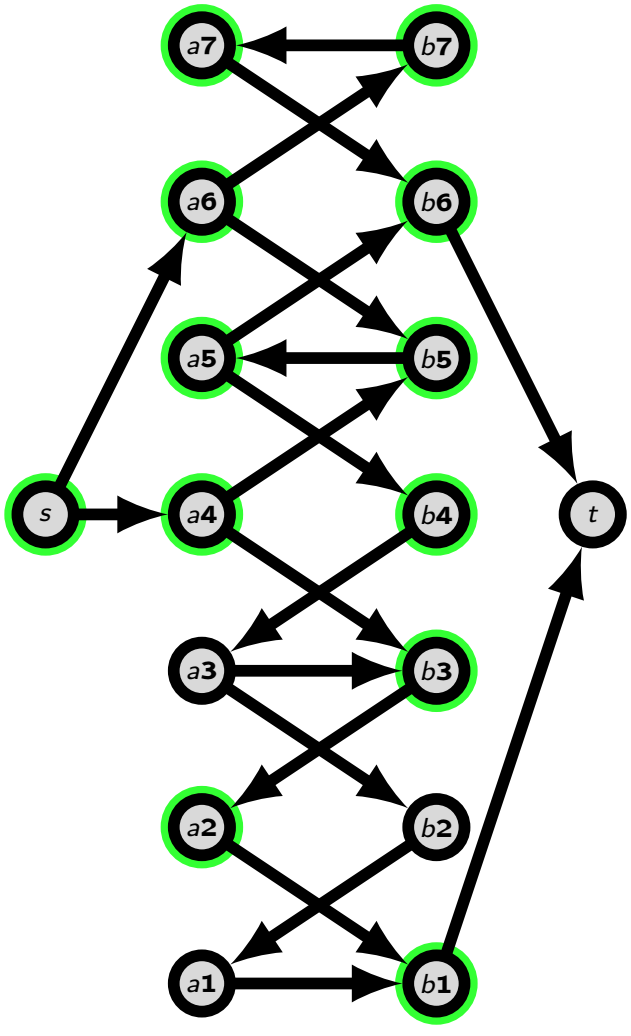


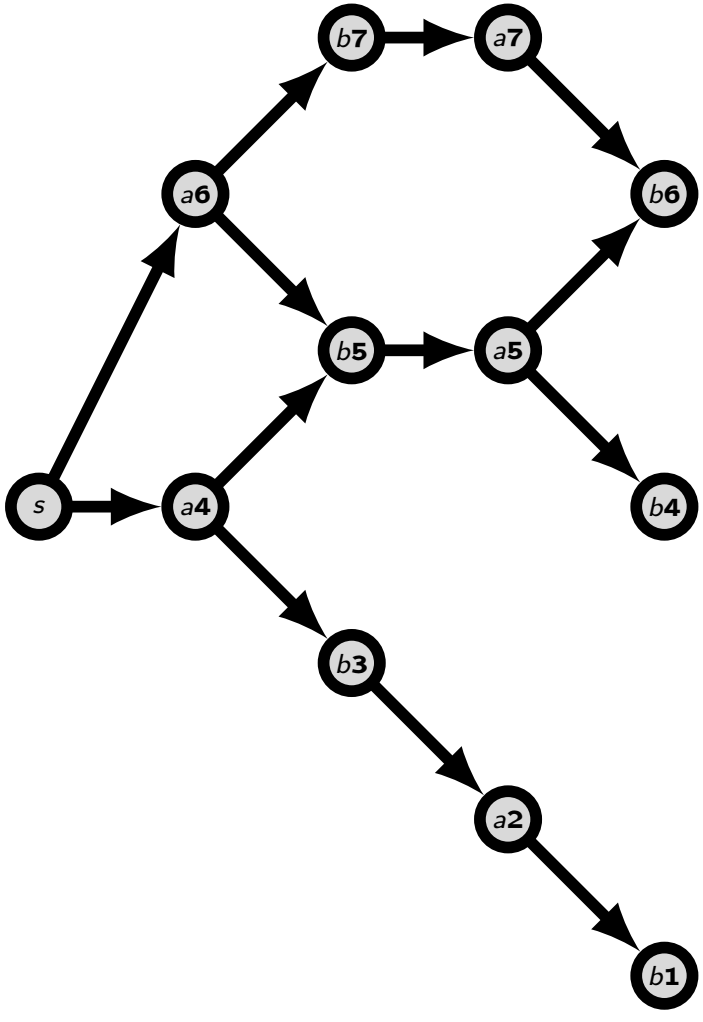
Beispiel (Breitensuche)



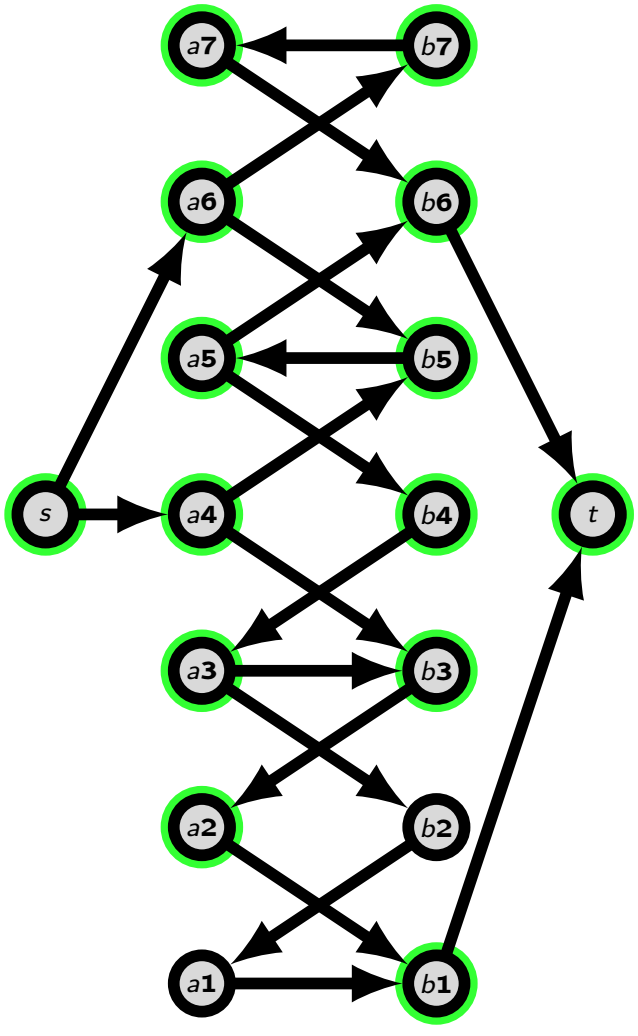


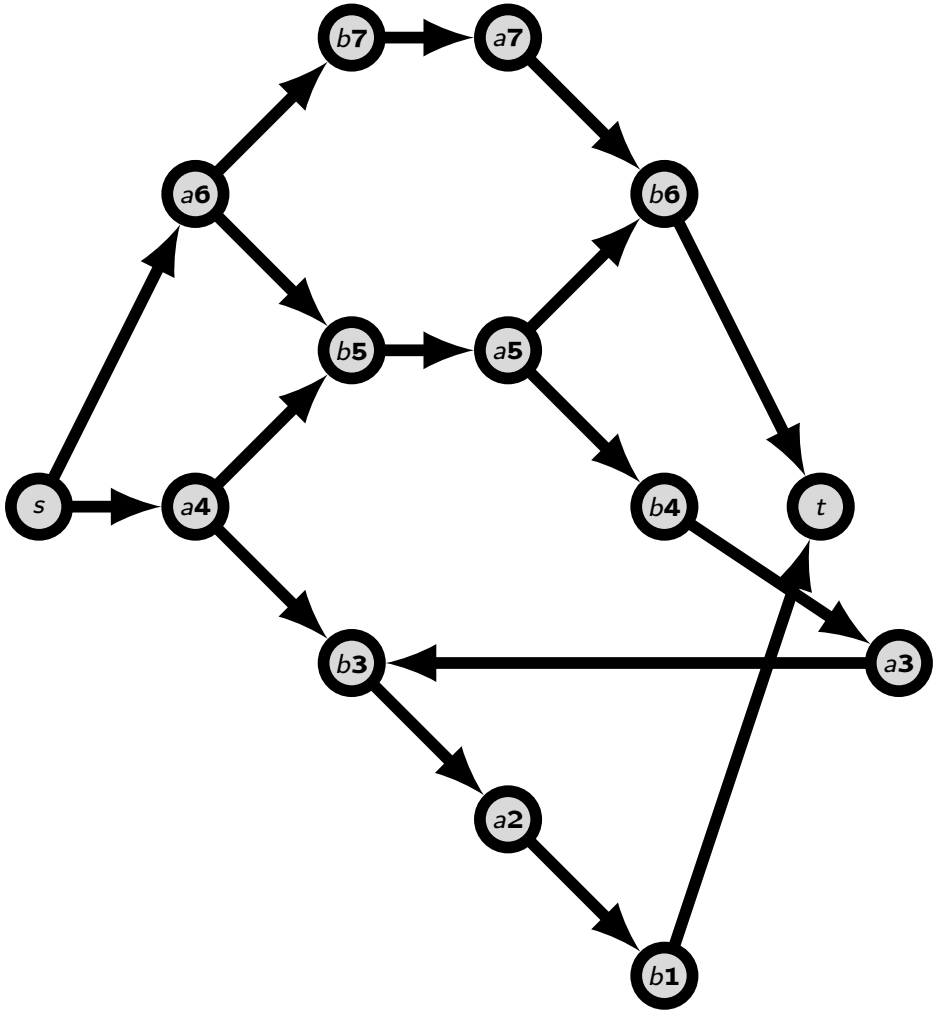
Beispiel (Breitensuche)



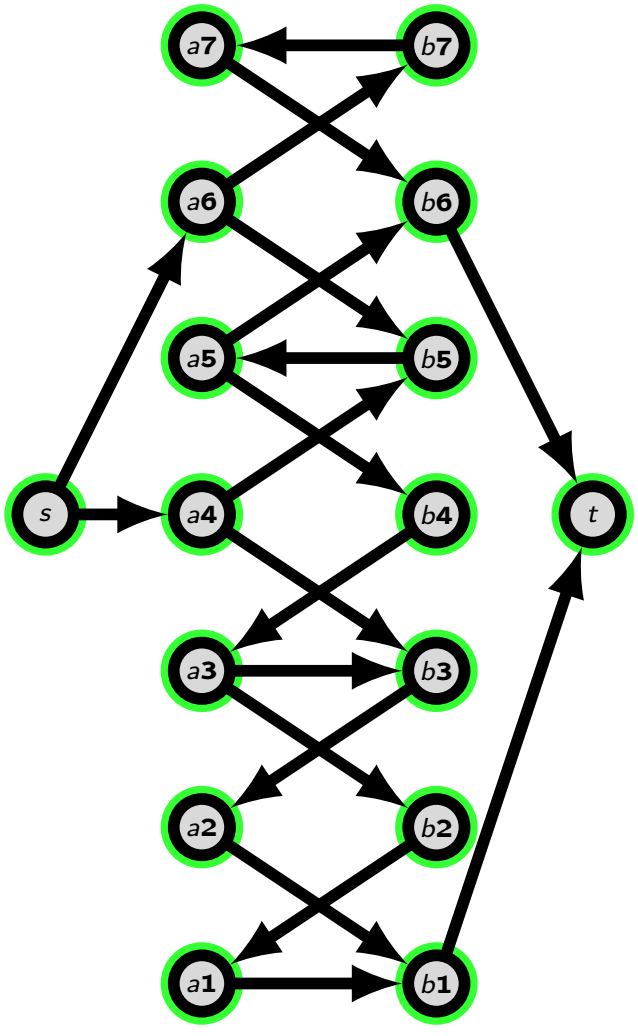


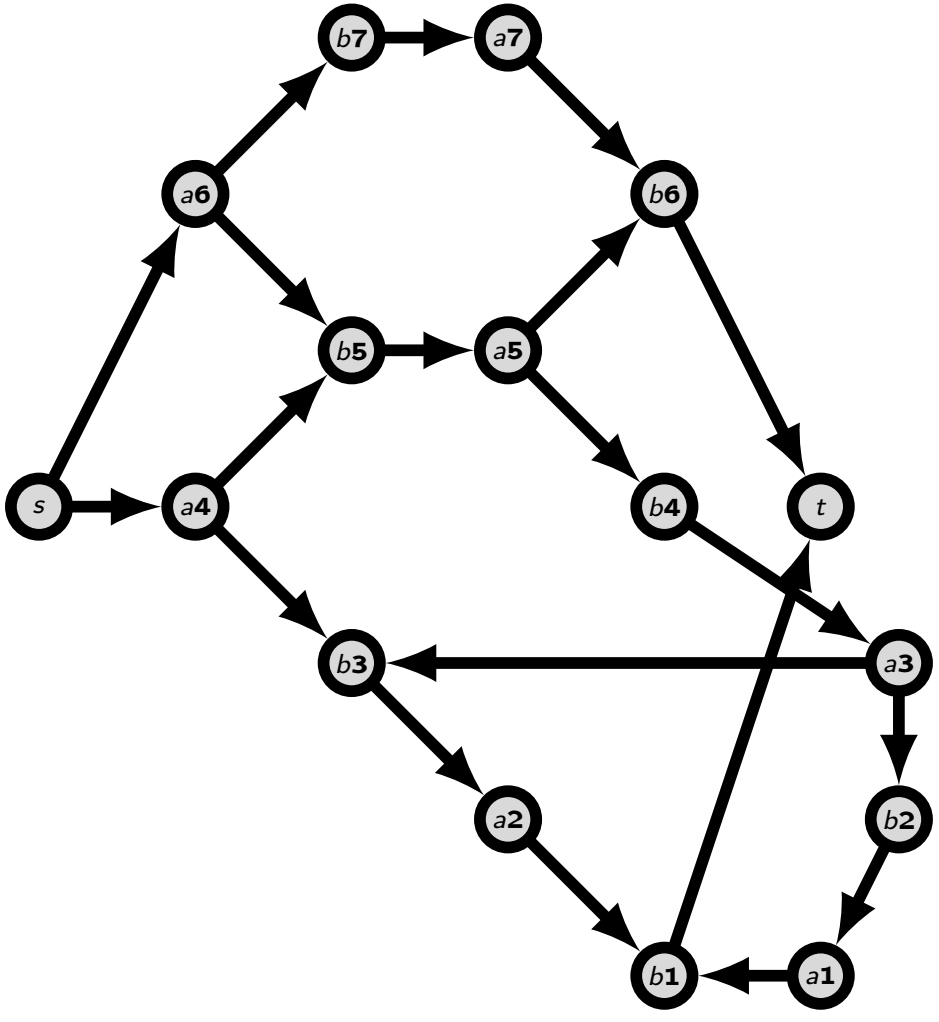
Beispiel (Breitensuche)



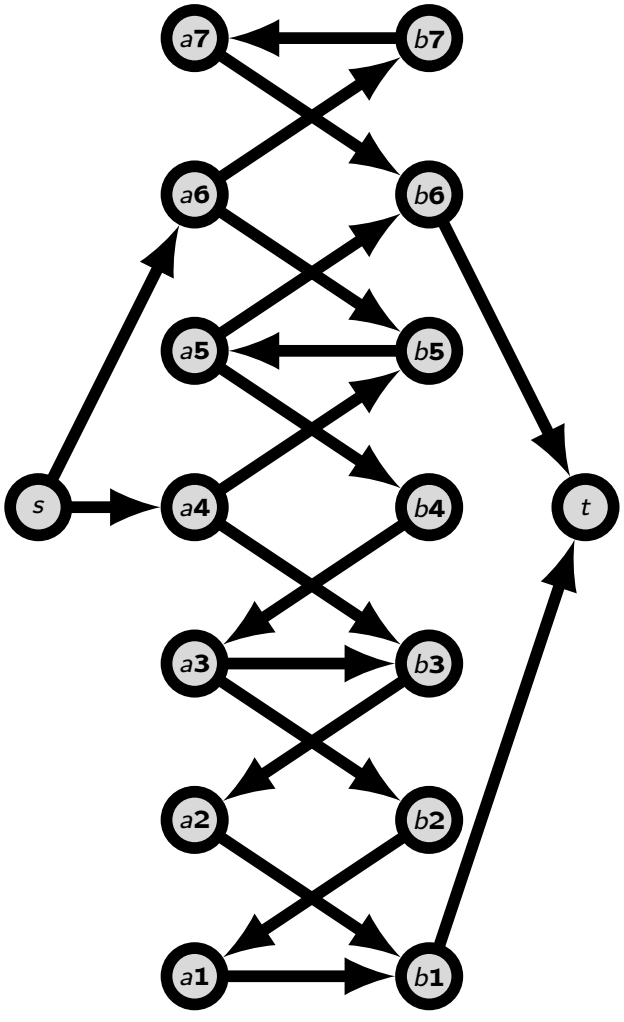


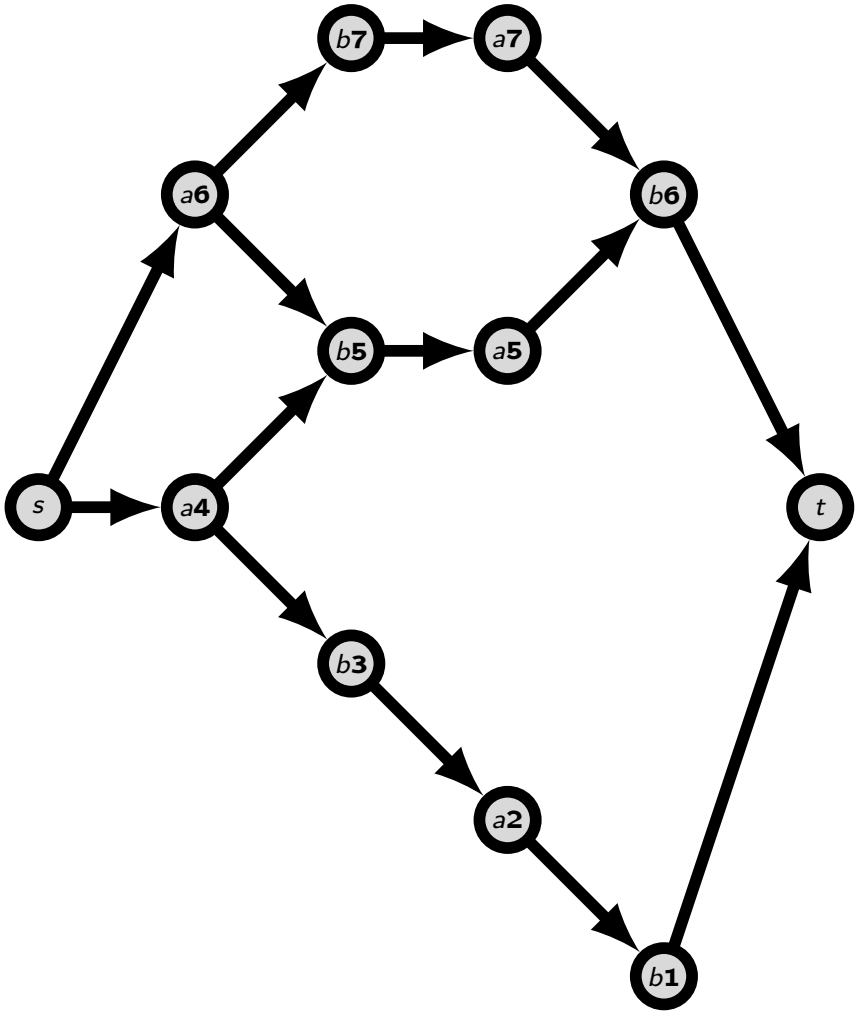
Beispiel (Breitensuche)



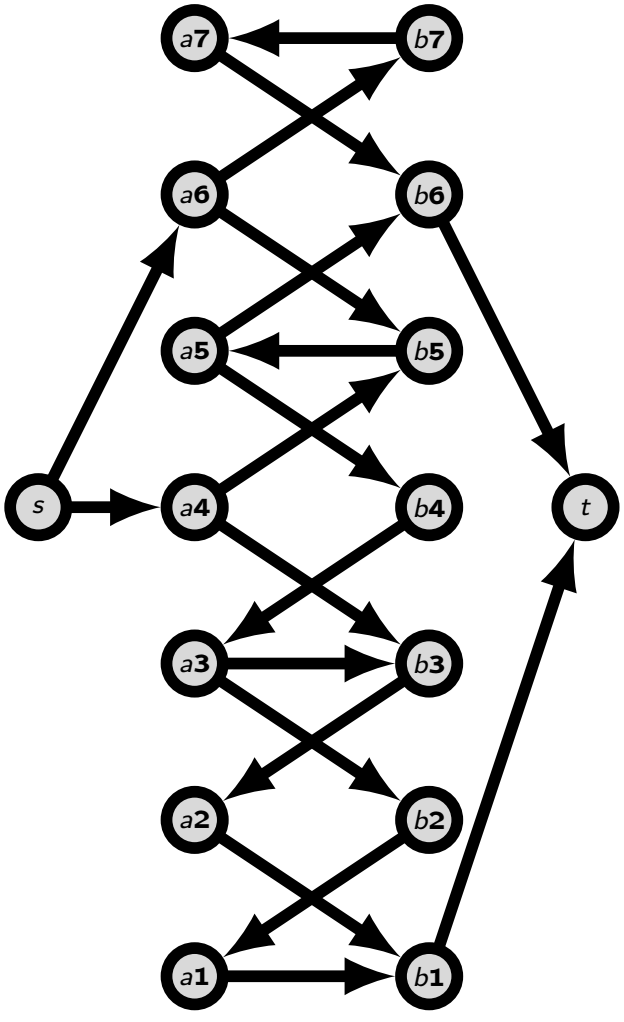


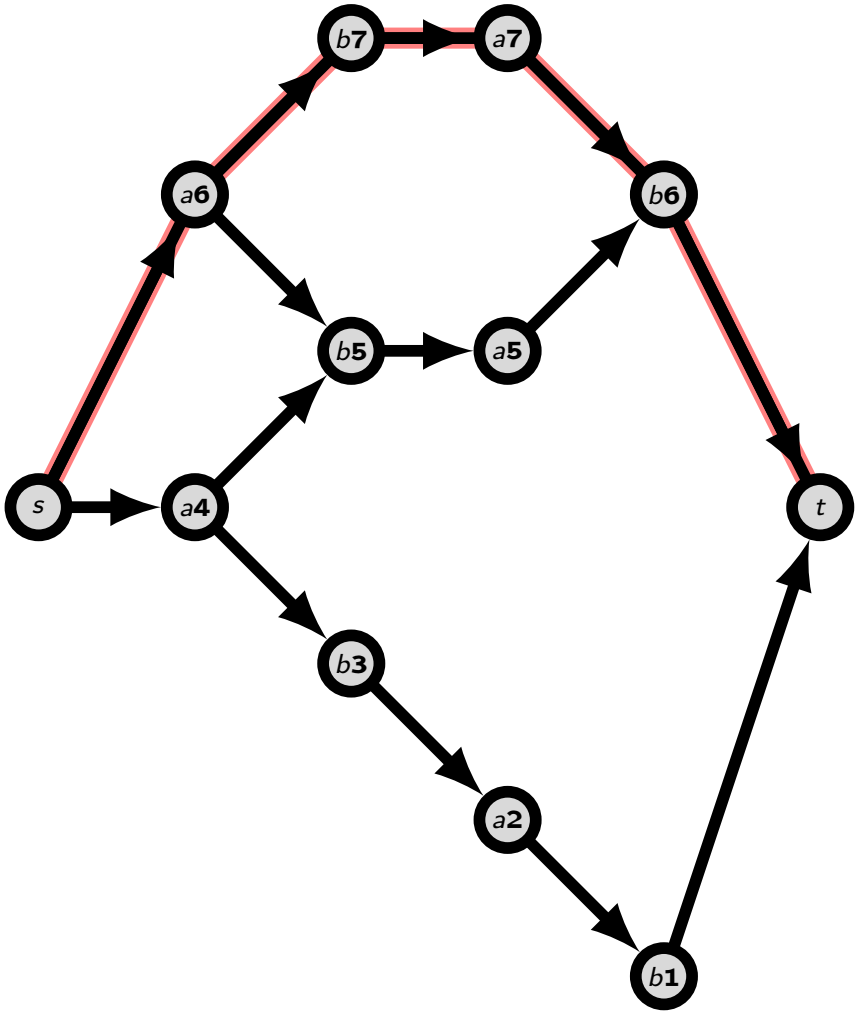
Beispiel (Tiefensuche)



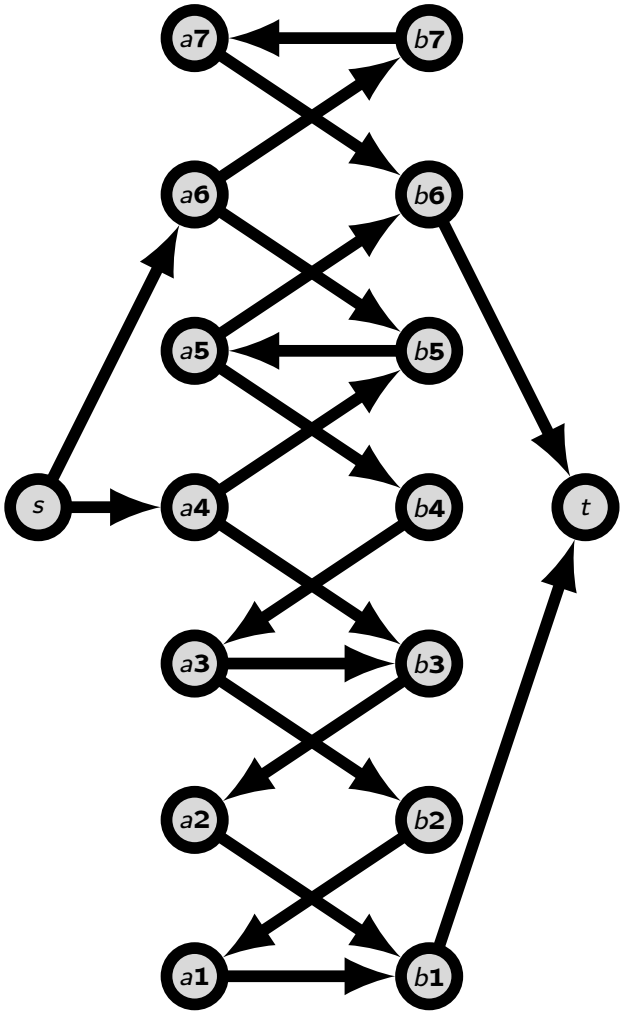


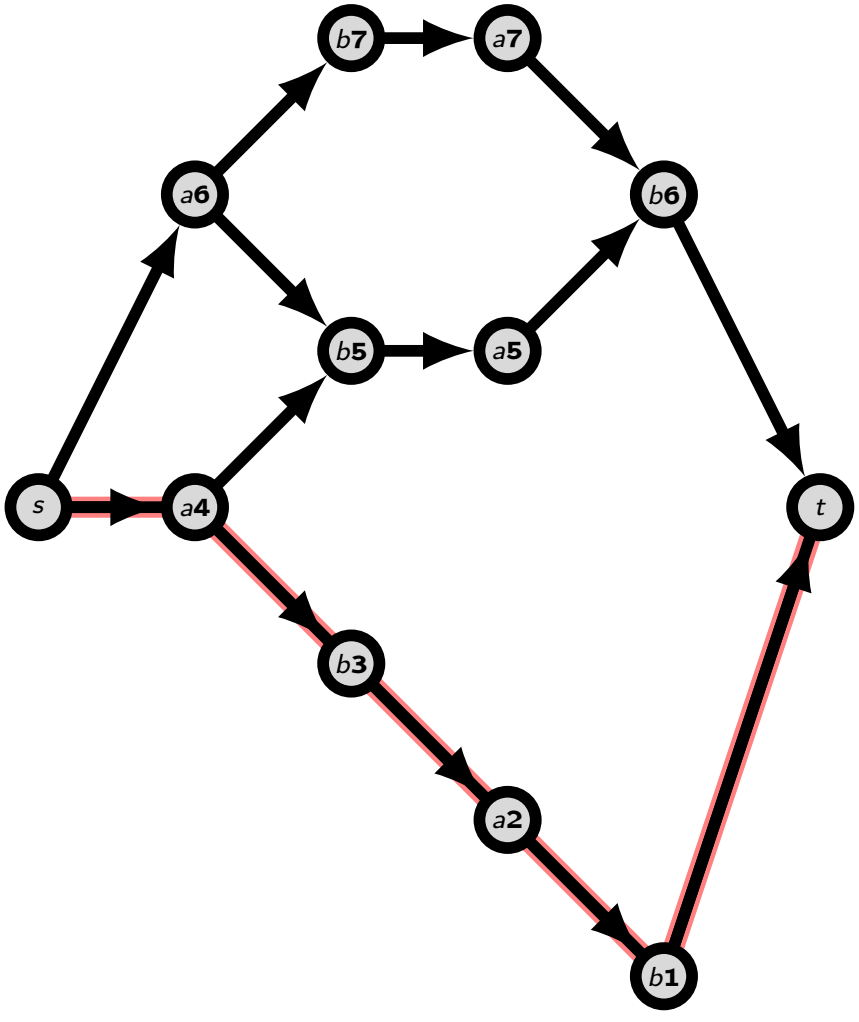
Beispiel (Tiefensuche)



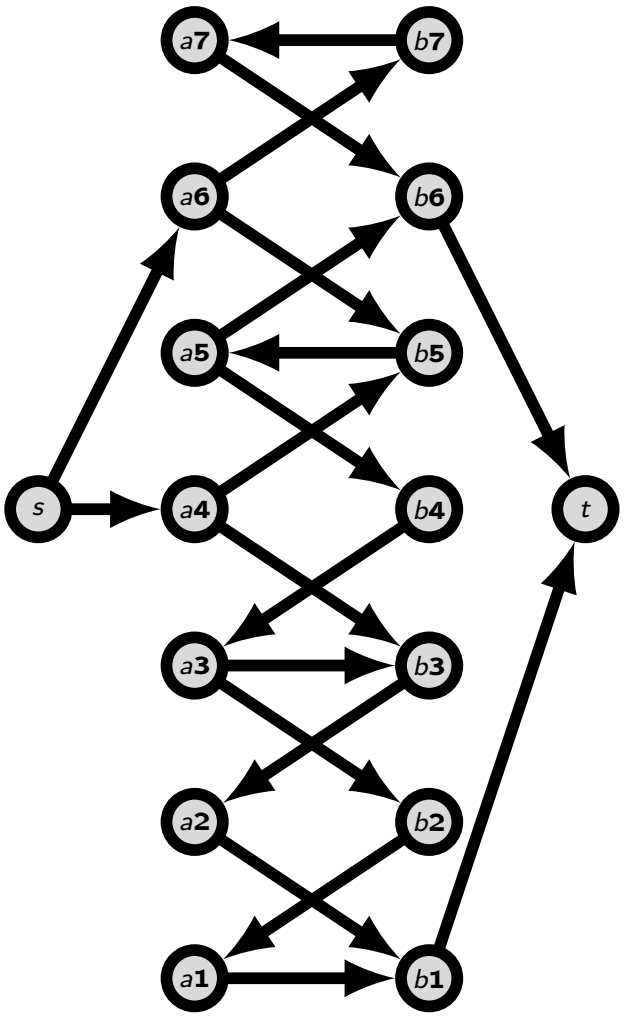


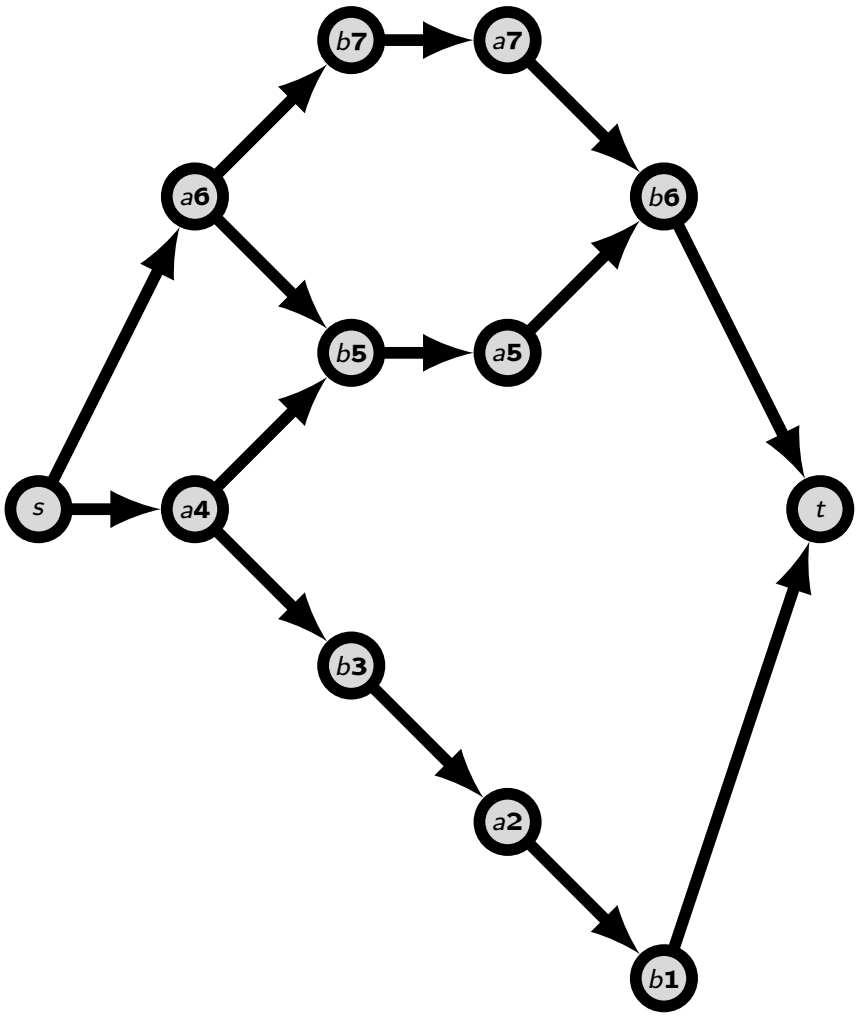
Beispiel (Tiefensuche)



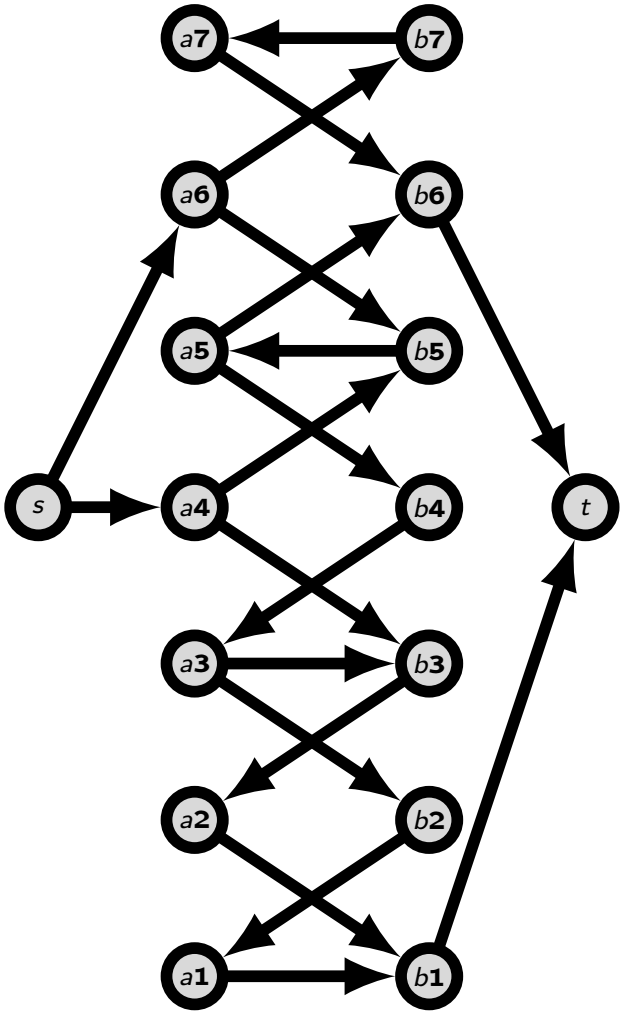


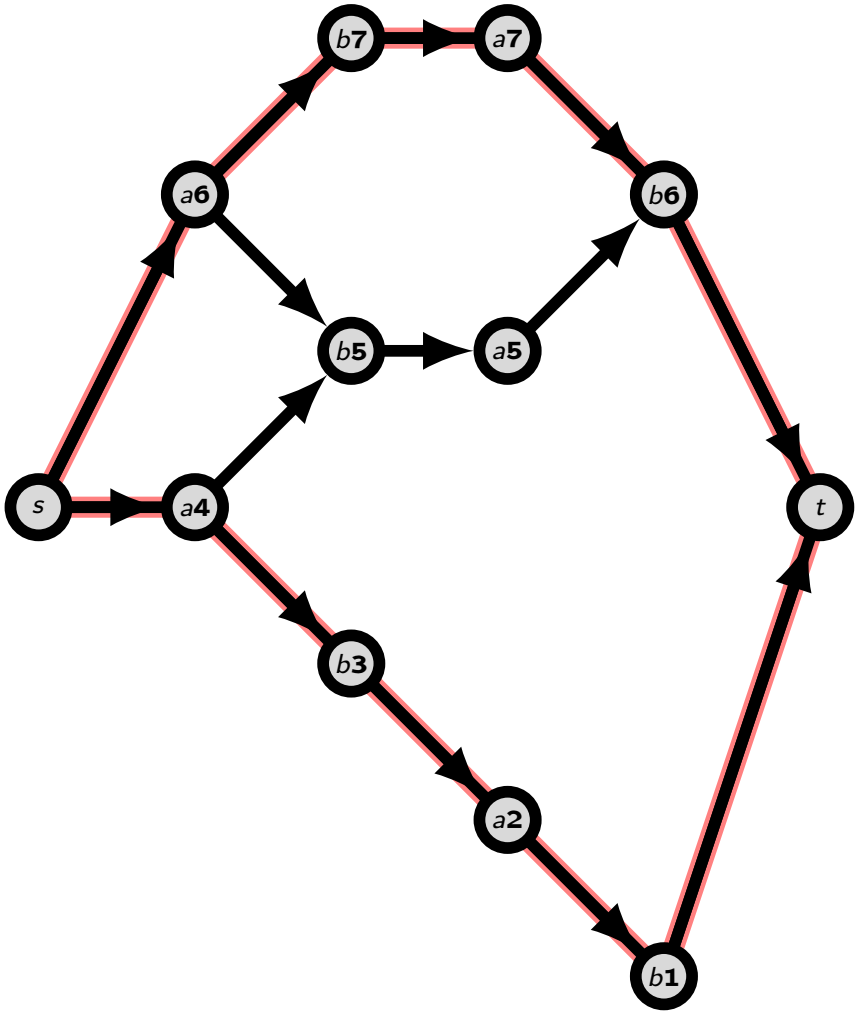
Beispiel (Verbessernde Pfade)



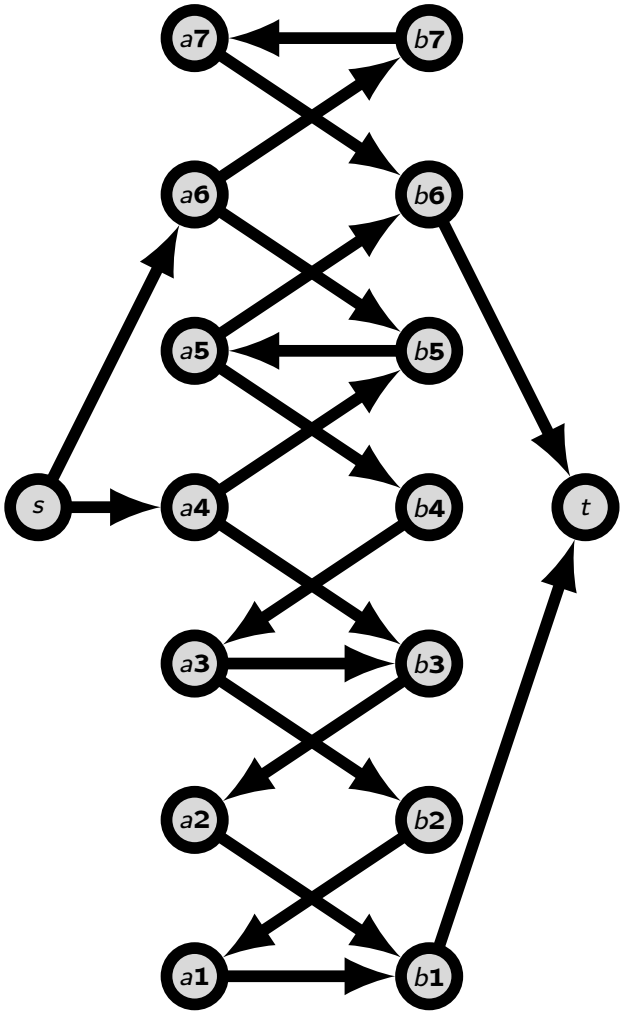


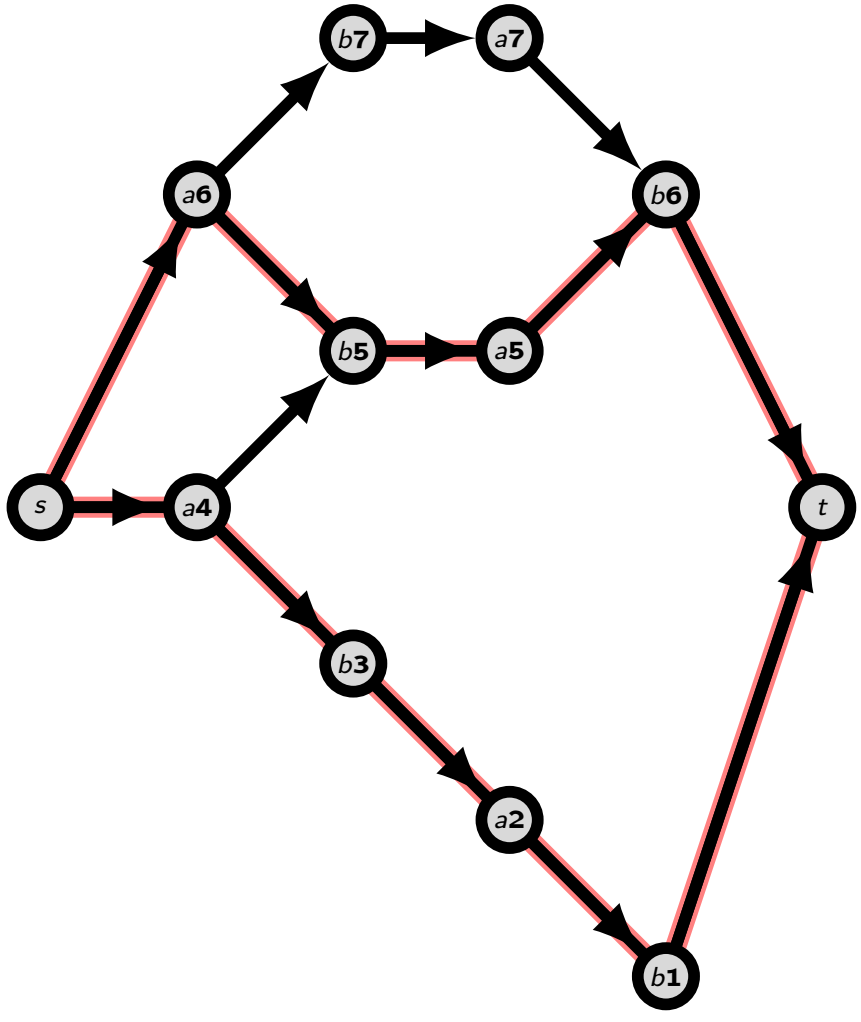
Beispiel (Verbessernde Pfade)



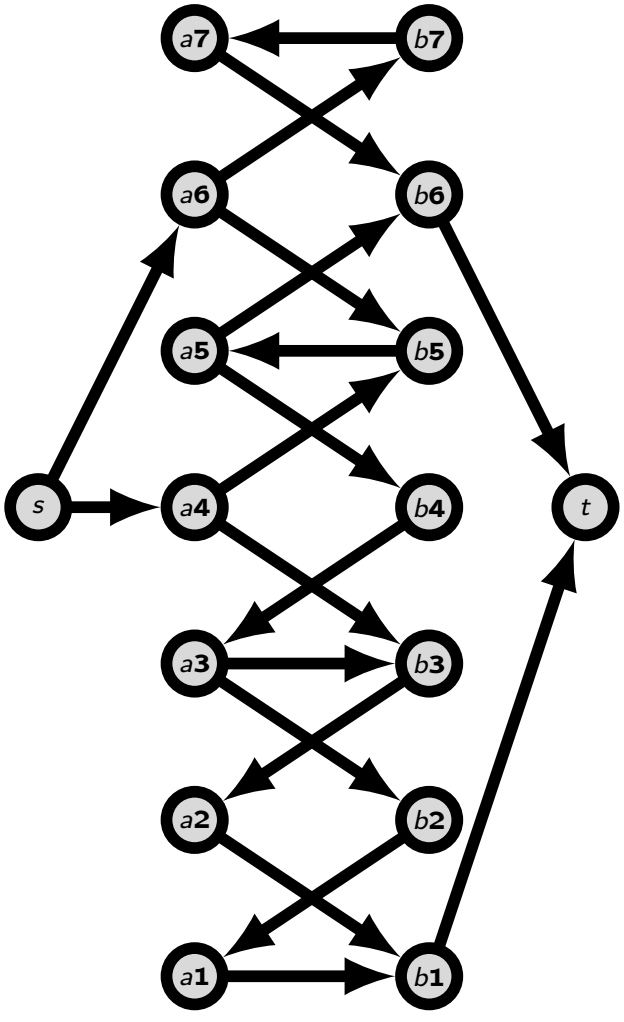


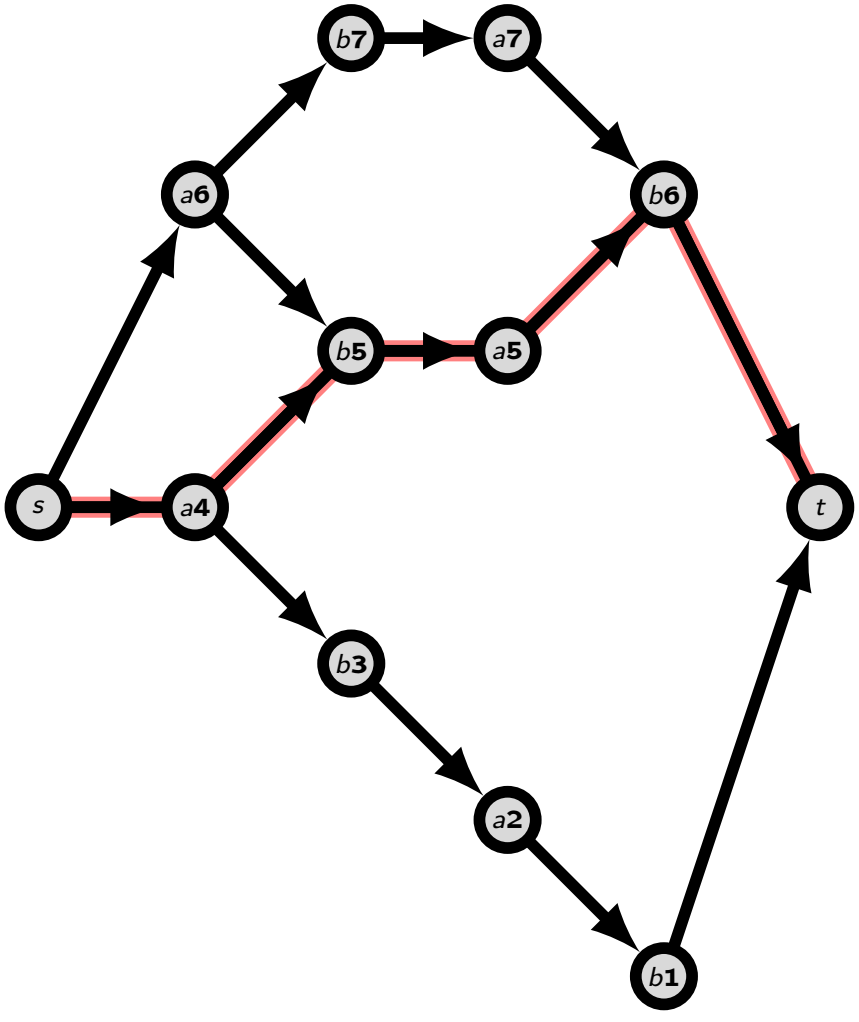
Beispiel (Verbessernde Pfade)



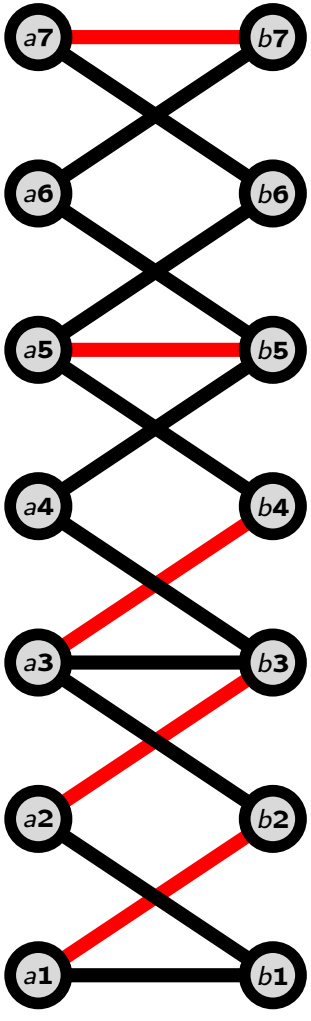


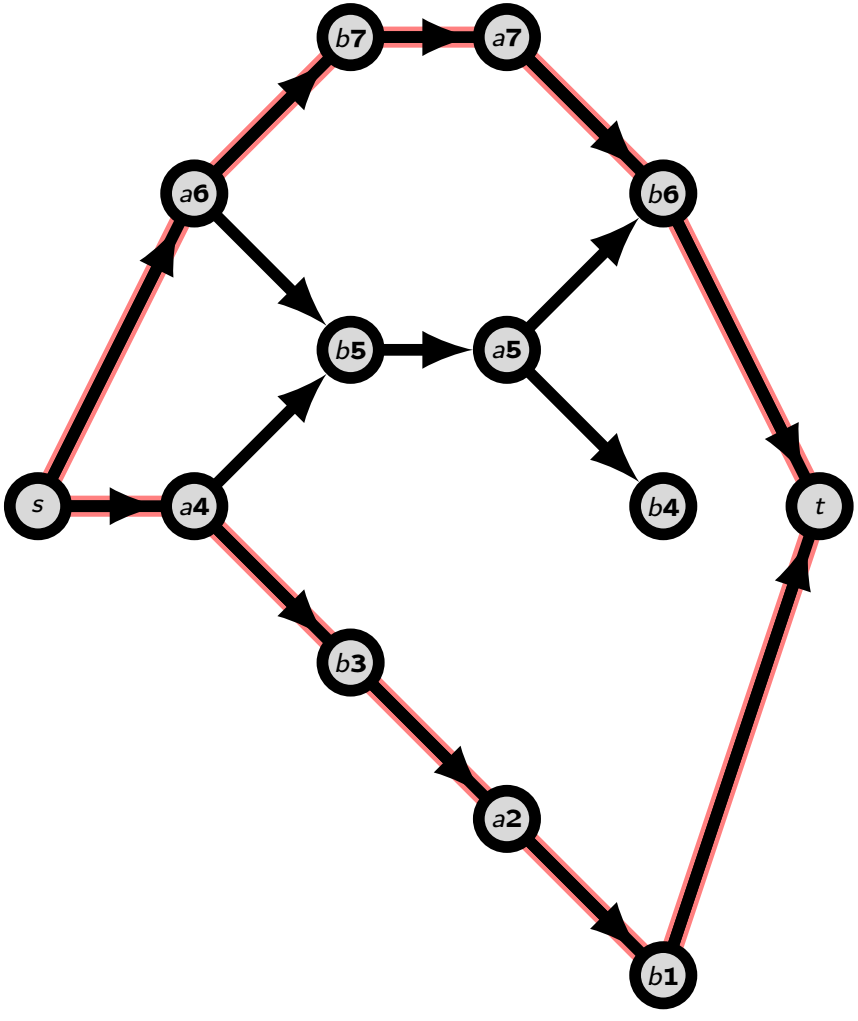
Beispiel (Verbessernde Pfade)



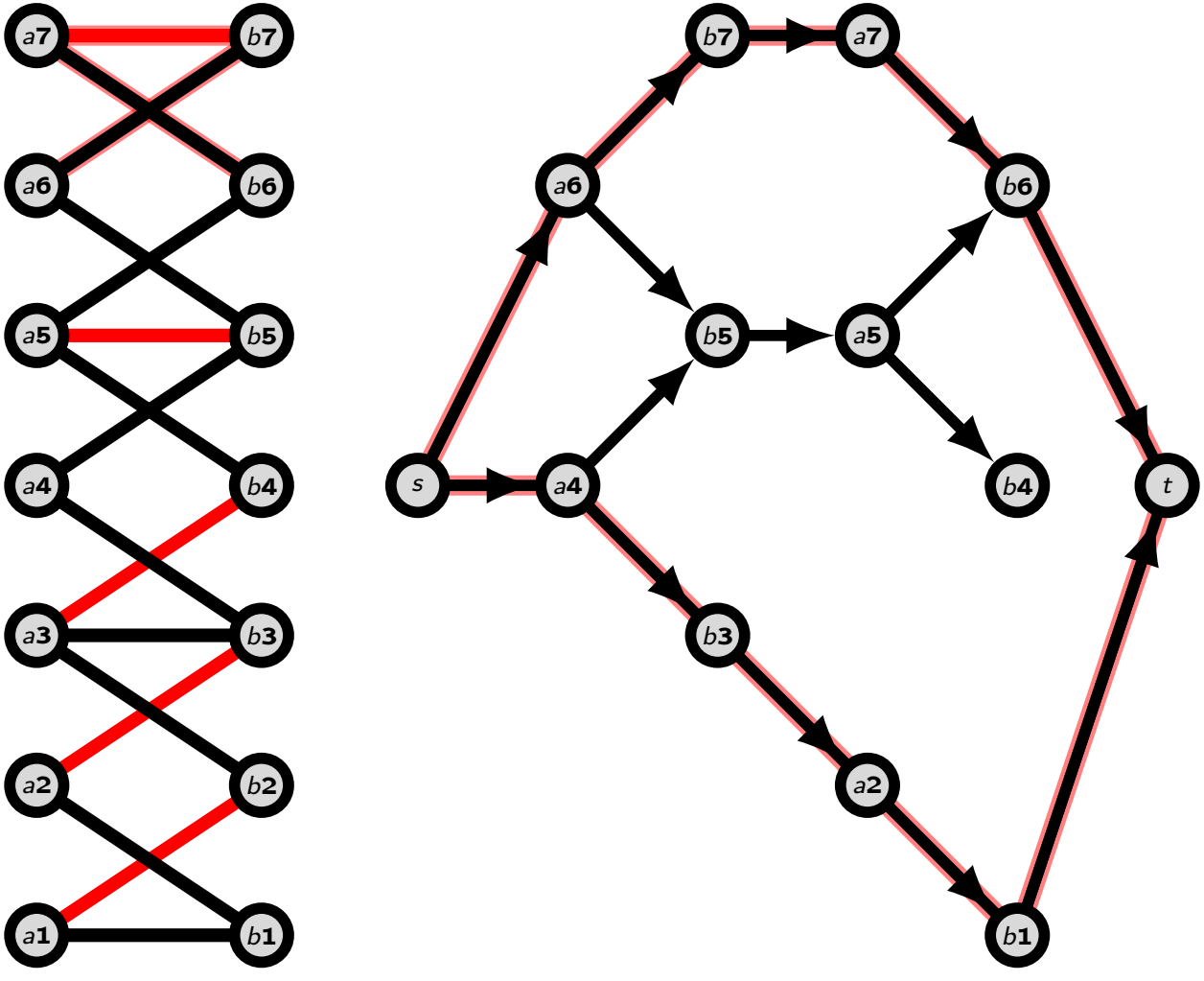


Beispiel

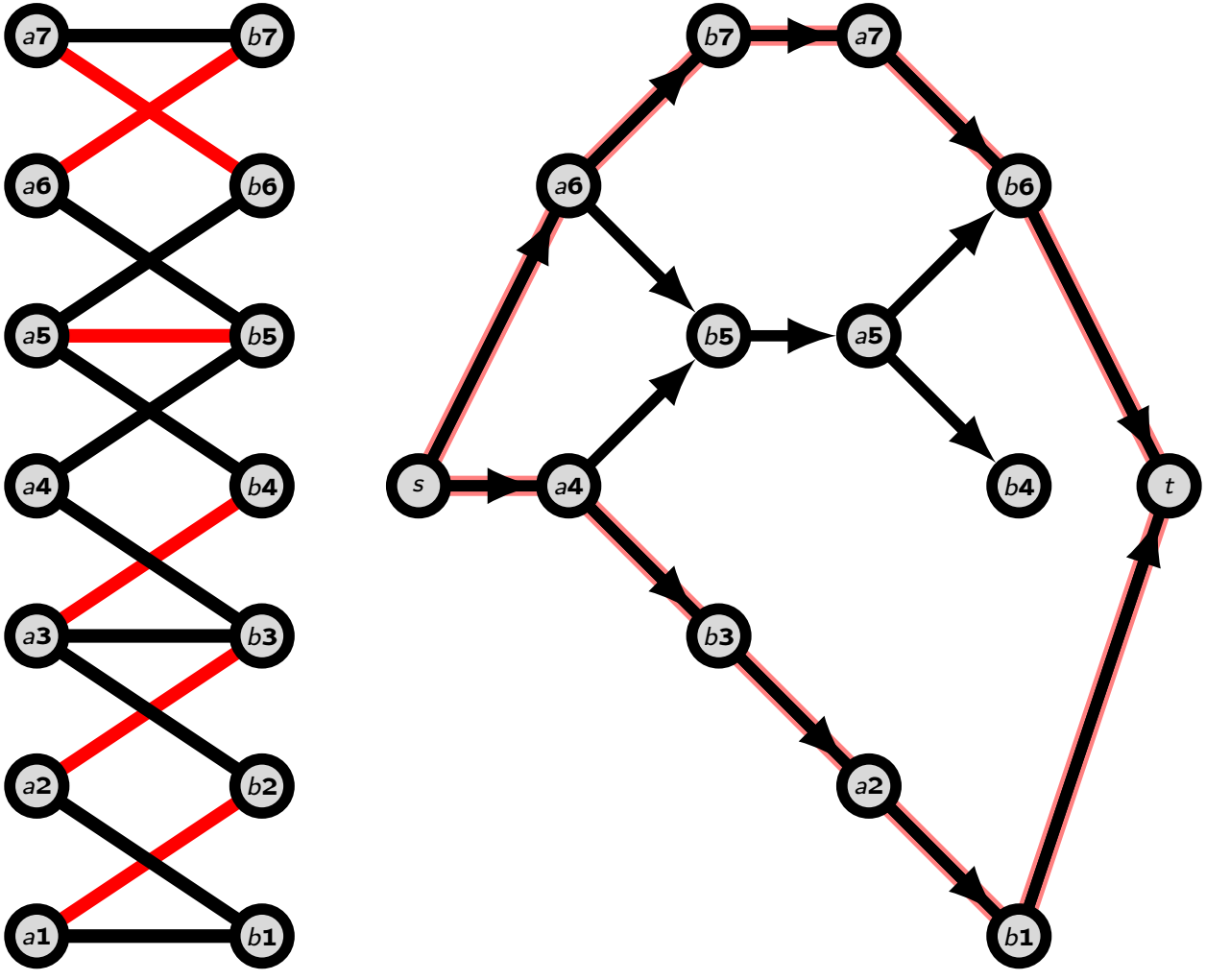

a7
b7
a6
b6
a5
b5
a4
b4
a3
b3
a2
b2
a1
b1


b7
a7
a6
b6
b5
a5
s
a4
b4
t
b3
a2
b1

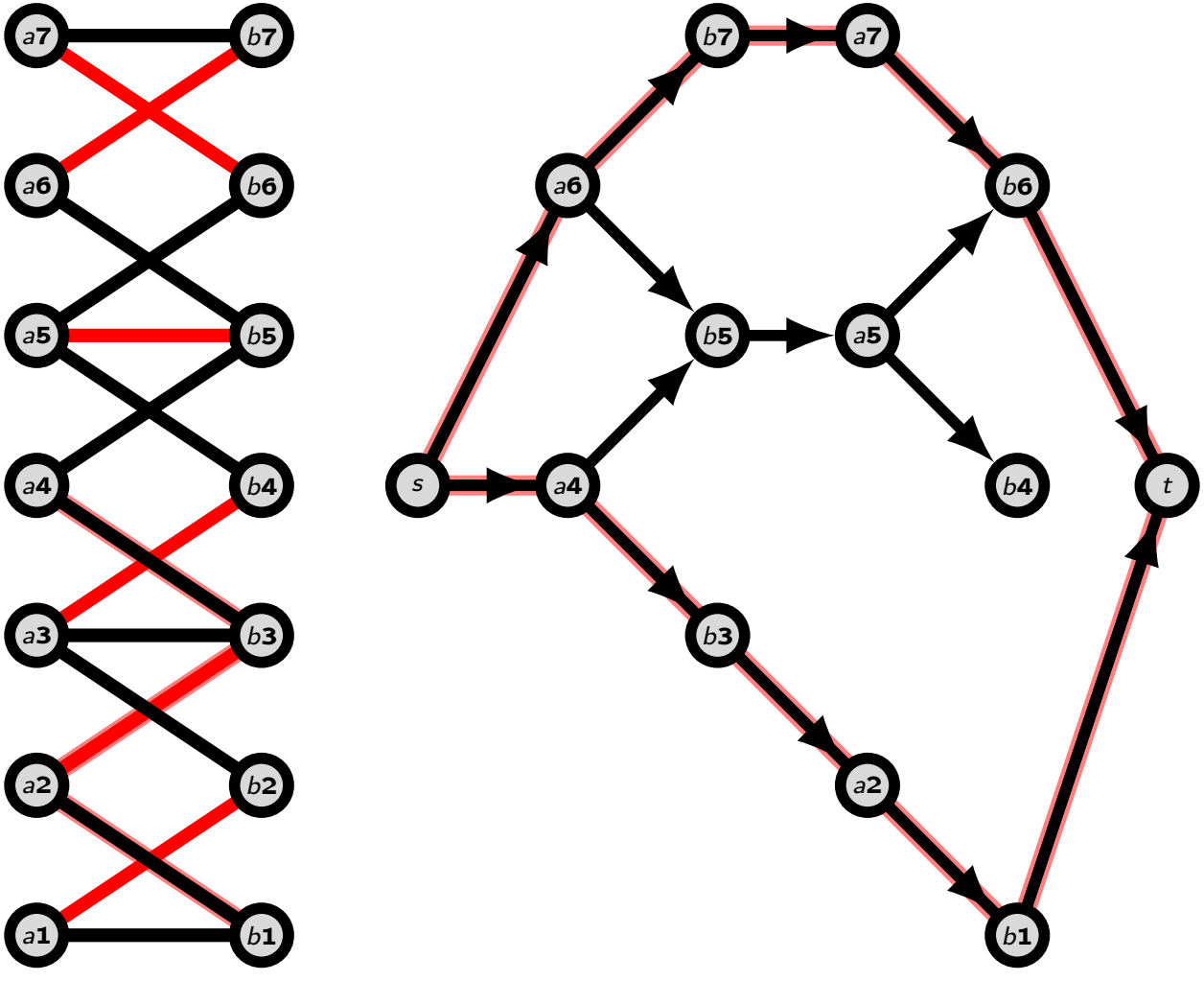
Beispiel



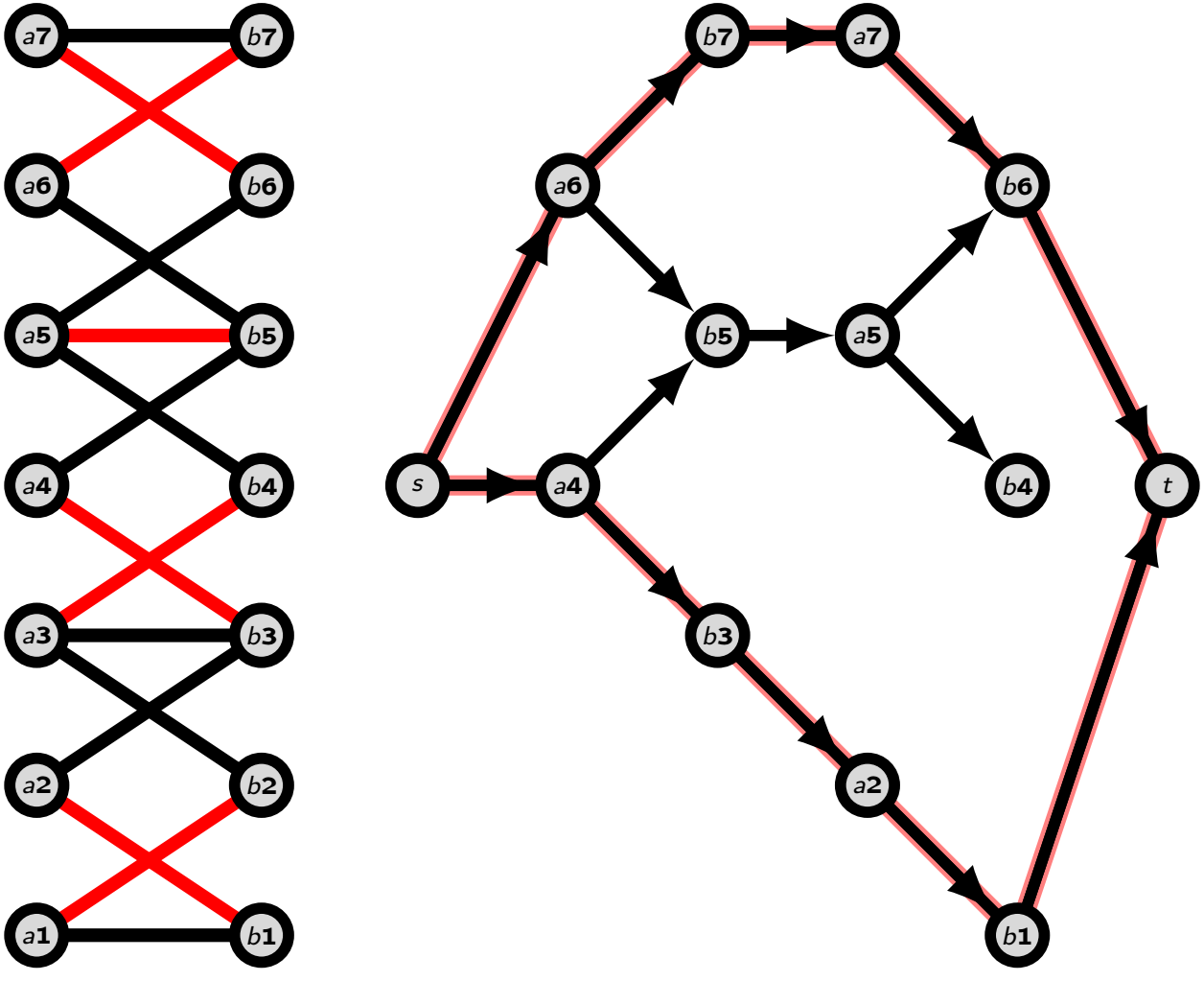
Beispiel


a7
b7
a6
b6
a5
b5
a4
b4
a3
b3
a2
b2
a1
b1
b7
a7
a6
b6
b5
a5
s
a4
b4
t
b3
a2
b1

Beispiel


a7
b7
a6
b6
a5
b5
a4
b4
a3
b3
a2
b2
a1
b1
b7
a7
a6
b6
b5
a5
s
a4
b4
t
b3
a2
b1

Beispiel


a7
b7
a6
b6
a5
b5
a4
b4
a3
b3
a2
b2
a1
b1
b7
a7
a6
b6
b5
a5
s
a4
b4
t
b3
a2
b1

Laufzeit

Theorem

Das Maximum Matchingproblem auf bipartiten Graphen kann in Zeit $O(m\sqrt{n})$ gelöst werden.

Laufzeit

Theorem

Das Maximum Matchingproblem auf bipartiten Graphen kann in Zeit $O(m\sqrt{n})$ gelöst werden.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$.
$M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$. $M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$.
$M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching.} : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$. $M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche.
- Hier werden nun Wege maximalen Gewichts gesucht.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$. $M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche.
- Hier werden nun Wege maximalen Gewichts gesucht.
  - Sei $G = (V, W, E)$ bipartiter Graph und M Matching.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$.
$M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche.
- Hier werden nun Wege maximalen Gewichts gesucht.
  - Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
  - Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
    $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
    $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
    $\forall e \in E' : g_M(e) = g(e)$ und
    $\forall e \in E'' : g_M(e) = -g(e)$.

Matching mit Kosten

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$.
$M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche.
- Hier werden nun Wege maximalen Gewichts gesucht.
  - Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
  - Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
    $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
    $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
    $\forall e \in E' : g_M(e) = g(e)$ und
    $\forall e \in E'' : g_M(e) = -g(e)$.
  - Füge Quelle und Senke hinzu, damit entsteht G'.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$.
$M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche.
- Hier werden nun Wege maximalen Gewichts gesucht.
  - Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
  - Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
    $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
    $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
    $\forall e \in E' : g_M(e) = g(e)$ und
    $\forall e \in E'' : g_M(e) = -g(e)$.
  - Füge Quelle und Senke hinzu, damit entsteht G'.

Idee

$g(E) = \sum_{e \in E} g(e)$

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.

Idee

$g(E) = \sum_{e \in E} g(e)$

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.

Idee

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.
- Damit wird ein ungerichteter M-verbessender Pfad P bestimmt.

Idee

$g(E) = \sum_{e \in E} g(e)$

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.
- Damit wird ein ungerichteter M-verbessender Pfad P bestimmt.
- Gewicht von P: $g_M(P) = \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e)$.

Idee

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.
- Damit wird ein ungerichteter M-verbessender Pfad P bestimmt.
- Gewicht von P: $g_M(P) = \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e)$.
- Damit haben wir:

$$g(M \oplus P) = \sum_{e \in M} g(e) + \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e) = g(M) + g_M(P)$$

Idee

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.
- Damit wird ein ungerichteter M-verbessender Pfad P bestimmt.
- Gewicht von P: $g_M(P) = \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e)$.
- Damit haben wir:

$$g(M \oplus P) = \sum_{e \in M} g(e) + \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e) = g(M) + g_M(P)$$

- Damit ergibt sich der folgende Algorithmus:

Idee

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.
- Damit wird ein ungerichteter M-verbessender Pfad P bestimmt.
- Gewicht von P: $g_M(P) = \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e)$.
- Damit haben wir:

$$g(M \oplus P) = \sum_{e \in M} g(e) + \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e) = g(M) + g_M(P)$$

- Damit ergibt sich der folgende Algorithmus:

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$
   4. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$
   4. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.
4. Ausgabe: M_{opt}.

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$
   4. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.
4. Ausgabe: M_{opt}.

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$
   4. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.
4. Ausgabe: M_{opt}.

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n^2 \cdot m)$.

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$
   4. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.
4. Ausgabe: M_{opt}.

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n^2 \cdot m)$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.

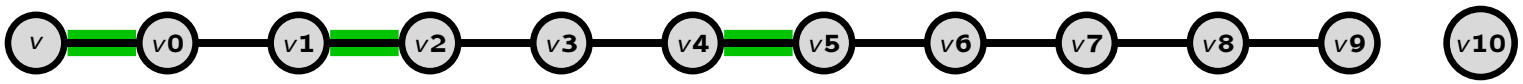
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



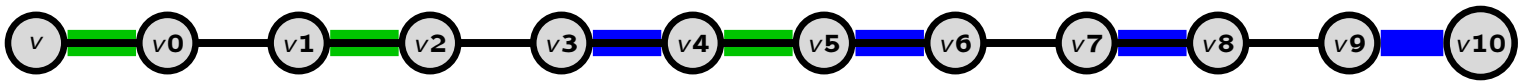
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.

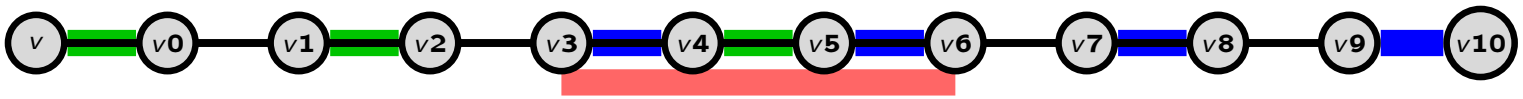
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.

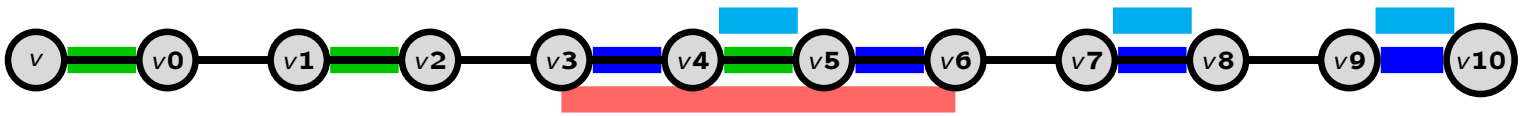
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.

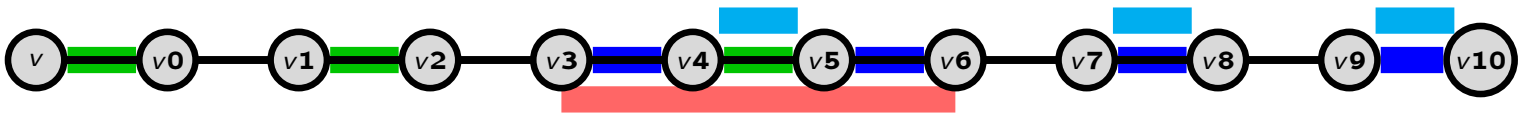
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.

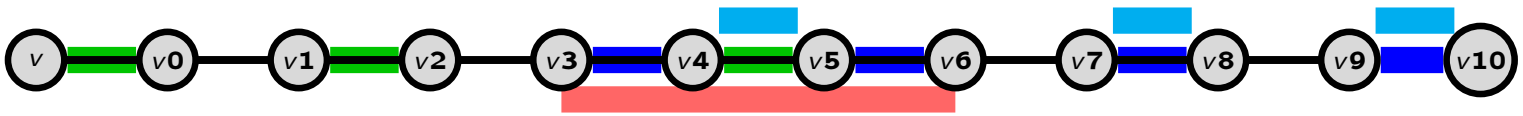
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i+1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



- Sei M' ein Matching mit $|M'| = i+1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i+1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i+1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.
- Per Induktion gilt: $g(M_i) \geqslant g(M) = g(M') - g_M(P)$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i+1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i+1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.
- Per Induktion gilt: $g(M_i) \geqslant g(M) = g(M') - g_M(P)$.
- Und wir erhalten $g_M(P) = g_{M_i}(P) \leqslant g_{M_i}(P')$,
  wobei P' der Pfad ist, den der Algorithmus bestimmte.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i+1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i+1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.
- Per Induktion gilt: $g(M_i) \geqslant g(M) = g(M') - g_M(P)$.
- Und wir erhalten $g_M(P) = g_{M_i}(P) \leqslant g_{M_i}(P')$, wobei P' der Pfad ist, den der Algorithmus bestimmte.
- Damit: $g(M') = g(M) + g_M(P) \leqslant g(M_i) + g_{M_i}(P') = g(M_{i+1})$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i+1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i+1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.
- Per Induktion gilt: $g(M_i) \geqslant g(M) = g(M') - g_M(P)$.
- Und wir erhalten $g_M(P) = g_{M_i}(P) \leqslant g_{M_i}(P')$,
  wobei P' der Pfad ist, den der Algorithmus bestimmte.
- Damit: $g(M') = g(M) + g_M(P) \leqslant g(M_i) + g_{M_i}(P') = g(M_{i+1})$.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.
  - Positive Kreise bezüglich g_M widersprächen der Optimalität von M_i:

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.
  - Positive Kreise bezüglich g_M widersprächen der Optimalität von M_i:
- Damit kann nun mit dem Bellmann-Ford-Algorithmus von einer Quelle aus gearbeitet werden.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.
  - Positive Kreise bezüglich g_M widersprächen der Optimalität von M_i:
- Damit kann nun mit dem Bellmann-Ford-Algorithmus von einer Quelle aus gearbeitet werden.
- Dieser hat Laufzeit $O(nm)$.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.
  - Positive Kreise bezüglich g_M widersprächen der Optimalität von M_i:
- Damit kann nun mit dem Bellmann-Ford-Algorithmus von einer Quelle aus gearbeitet werden.
- Dieser hat Laufzeit $O(nm)$.
- Damit ist die Gesamtlaufzeit $O(n^2 m)$.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.
  - Positive Kreise bezüglich g_M widersprächen der Optimalität von M_i:
- Damit kann nun mit dem Bellmann-Ford-Algorithmus von einer Quelle aus gearbeitet werden.
- Dieser hat Laufzeit $O(nm)$.
- Damit ist die Gesamtlaufzeit $O(n^2 m)$.

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).
- Transformiere dazu die Längen, so dass keine negativen Werte auftreten.

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).
- Transformiere dazu die Längen, so dass keine negativen Werte auftreten.
- Beachte: wir haben keine negativen Kreise.

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).
- Transformiere dazu die Längen, so dass keine negativen Werte auftreten.
- Beachte: wir haben keine negativen Kreise.
- Verwende dazu eine Potentialfunkion $p : V \mapsto \mathbb{Z}$ mit:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v)$$

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).
- Transformiere dazu die Längen, so dass keine negativen Werte auftreten.
- Beachte: wir haben keine negativen Kreise.
- Verwende dazu eine Potentialfunkion $p : V \mapsto \mathbb{Z}$ mit:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v)$$

- Die Berechnung so einer Potentialfunktion p im Rahmen der Suche nach dem Matching wird die Ungarische Methode genannt.

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).
- Transformiere dazu die Längen, so dass keine negativen Werte auftreten.
- Beachte: wir haben keine negativen Kreise.
- Verwende dazu eine Potentialfunkion $p : V \mapsto \mathbb{Z}$ mit:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v)$$

- Die Berechnung so einer Potentialfunktion p im Rahmen der Suche nach dem Matching wird die Ungarische Methode genannt.

Transformation 1

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

Transformation 1

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

Transformation 1

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

$$\text{dist}_{l'}(a, b) \quad = \quad \sum_{e \in P} l'(e)$$

Transformation 1

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

$$\begin{aligned} \text{dist}_{l'}(a, b) &= \sum_{e \in P} l'(e) \\ &= \sum_{(v,w)=e \in P} l(e) - p(w) + p(v) \end{aligned}$$

Transformation 1

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

$$\begin{aligned} \text{dist}_{l'}(a, b) &= \sum_{e \in P} l'(e) \\ &= \sum_{(v,w)=e \in P} l(e) - p(w) + p(v) \\ &= p(a) - p(b) + \sum_{e \in P} l(e) \end{aligned}$$

Transformation 1

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

$$\begin{aligned} \text{dist}_{l'}(a, b) &= \sum_{e \in P} l'(e) \\ &= \sum_{(v,w)=e \in P} l(e) - p(w) + p(v) \\ &= p(a) - p(b) + \sum_{e \in P} l(e) \end{aligned}$$

Damit können die kürzesten Wege auch mit diesen neuen Kantengewichten bestimmt werden.

Transformation 2

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.
- Und damit:

$$l'(e) \quad = \quad l(e) - p(w) + p(v)$$

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.
- Und damit:

$$l'(e) \quad = \quad l(e) - p(w) + p(v)$$

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.
- Und damit:

$$\begin{aligned} l'(e) &= l(e) - p(w) + p(v) \\ &= l(e) - \text{dist}_l(q, w) + \text{dist}_l(q, v) \end{aligned}$$

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.
- Und damit:

$$\begin{array}{rcl} l'(e) & = & l(e) - p(w) + p(v) \\ & = & l(e) - \text{dist}_l(q, w) + \text{dist}_l(q, v) \\ & \geqslant & l(e) - (\text{dist}_l(q, v) + l(e)) + \text{dist}_l(q, v) \quad = \quad 0 \end{array}$$

Transformation 2

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.
- Und damit:

$$\begin{array}{rcl} l'(e) & = & l(e) - p(w) + p(v) \\ & = & l(e) - \text{dist}_l(q, w) + \text{dist}_l(q, v) \\ & \geqslant & l(e) - (\text{dist}_l(q, v) + l(e)) + \text{dist}_l(q, v) \quad = \quad 0 \end{array}$$

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.
- Sei P ein kürzester Weg von q nach v.

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.
- Sei P ein kürzester Weg von q nach v.
- Dieser hat im Vergleich zu den ursprünglichen Kosten den aditiven Term $p(q) - p(v)$.

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.
- Sei P ein kürzester Weg von q nach v.
- Dieser hat im Vergleich zu den ursprünglichen Kosten den aditiven Term $p(q) - p(v)$.
- Damit $p(q) - p(v) = \text{dist}_l(q, q) - \text{dist}_l(q, v) = -\text{dist}_l(q, v)$.

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.
- Sei P ein kürzester Weg von q nach v.
- Dieser hat im Vergleich zu den ursprünglichen Kosten den aditiven Term $p(q) - p(v)$.
- Damit $p(q) - p(v) = \text{dist}_l(q, q) - \text{dist}_l(q, v) = -\text{dist}_l(q, v)$.
- Und weiter. $\text{dist}_{l'}(q, v) = \text{dist}_l(q, v) - \text{dist}_l(q, v) = 0$.

Transformation 3

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.
- Sei P ein kürzester Weg von q nach v.
- Dieser hat im Vergleich zu den ursprünglichen Kosten den aditiven Term $p(q) - p(v)$.
- Damit $p(q) - p(v) = \text{dist}_l(q, q) - \text{dist}_l(q, v) = -\text{dist}_l(q, v)$.
- Und weiter. $\text{dist}_{l'}(q, v) = \text{dist}_l(q, v) - \text{dist}_l(q, v) = 0$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.
   4. Setze $l'(e) = l_M(e) + p(v) - p(w)$ für alle Kanten $e = (v, w)$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.
   4. Setze $l'(e) = l_M(e) + p(v) - p(w)$ für alle Kanten $e = (v, w)$.
   5. Falls es Pfad P mit $l'(P) = 0$ von q zu einem Knoten aus $W \setminus V(M)$ gibt, so setze $M = M \oplus E(P)$

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.
   4. Setze $l'(e) = l_M(e) + p(v) - p(w)$ für alle Kanten $e = (v, w)$.
   5. Falls es Pfad P mit $l'(P) = 0$ von q zu einem Knoten aus $W \setminus V(M)$ gibt, so setze $M = M \oplus E(P)$
   6. Falls es keinen solchen Pfad P gibt, so gebe M_{opt} aus und terminiere.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.
   4. Setze $l'(e) = l_M(e) + p(v) - p(w)$ für alle Kanten $e = (v, w)$.
   5. Falls es Pfad P mit $l'(P) = 0$ von q zu einem Knoten aus $W \setminus V(M)$ gibt, so setze $M = M \oplus E(P)$
   6. Falls es keinen solchen Pfad P gibt, so gebe M_{opt} aus und terminiere.
   7. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.
   4. Setze $l'(e) = l_M(e) + p(v) - p(w)$ für alle Kanten $e = (v, w)$.
   5. Falls es Pfad P mit $l'(P) = 0$ von q zu einem Knoten aus $W \setminus V(M)$ gibt, so setze $M = M \oplus E(P)$
   6. Falls es keinen solchen Pfad P gibt, so gebe M_{opt} aus und terminiere.
   7. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.

Laufzeit

$l(e) = -g_M(e)$

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

Laufzeit

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.

Laufzeit

$l(e) = -g_M(e)$

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.

Laufzeit

$l(e) = -g_M(e)$

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.

Laufzeit

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.
- Für die gedreht Kante $e' = (w, v)$ gilt:

$$l'_{M_{i+1}}(e') = l_{M_{i+1}}(e') - p(w) + p(v) = -l_{M_i}(e') - p(w) + p(v) = -l'_{M_i}(e) = 0.$$

Laufzeit

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.
- Für die gedreht Kante $e' = (w, v)$ gilt:

$$l'_{M_{i+1}}(e') = l_{M_{i+1}}(e') - p(w) + p(v) = -l_{M_i}(e') - p(w) + p(v) = -l'_{M_i}(e) = 0.$$

- Damit können wir im Schritt 4.5 den Dijkstra-Algorithmus anwenden.

Laufzeit

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.
- Für die gedreht Kante $e' = (w, v)$ gilt:

$$l'_{M_{i+1}}(e') = l_{M_{i+1}}(e') - p(w) + p(v) = -l_{M_i}(e') - p(w) + p(v) = -l'_{M_i}(e) = 0.$$

- Damit können wir im Schritt 4.5 den Dijkstra-Algorithmus anwenden.
- Weiterhin können wir die Potentiale in Schritt 4.3 mit Breitesuche bestimmen.

Laufzeit

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.
- Für die gedreht Kante $e' = (w, v)$ gilt:

$$l'_{M_{i+1}}(e') = l_{M_{i+1}}(e') - p(w) + p(v) = -l_{M_i}(e') - p(w) + p(v) = -l'_{M_i}(e) = 0.$$

- Damit können wir im Schritt 4.5 den Dijkstra-Algorithmus anwenden.
- Weiterhin können wir die Potentiale in Schritt 4.3 mit Breitesuche bestimmen.
- Damit $O(n)$ Iterationen mit Laufzeit $O(m + n \log n)$ plus $O(n + m)$.

Laufzeit

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.
- Für die gedreht Kante $e' = (w, v)$ gilt:

$$l'_{M_{i+1}}(e') = l_{M_{i+1}}(e') - p(w) + p(v) = -l_{M_i}(e') - p(w) + p(v) = -l'_{M_i}(e) = 0.$$

- Damit können wir im Schritt 4.5 den Dijkstra-Algorithmus anwenden.
- Weiterhin können wir die Potentiale in Schritt 4.3 mit Breitesuche bestimmen.
- Damit $O(n)$ Iterationen mit Laufzeit $O(m + n \log n)$ plus $O(n + m)$.

Einleitung

$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?

Einleitung

$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.

Einleitung

$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.
- Ausblick und Vorgehen:

Einleitung

$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.
- Ausblick und Vorgehen:
  - Untersuche die möglichen Situationen.

Einleitung

$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.
- Ausblick und Vorgehen:
  - Untersuche die möglichen Situationen.
  - Erkenne die Situationen, die kritisch für den Algorithmus sind.

Einleitung

$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.
- Ausblick und Vorgehen:
  - Untersuche die möglichen Situationen.
  - Erkenne die Situationen, die kritisch für den Algorithmus sind.
  - Passe Algorithmus für diese Situationen an.

Einleitung

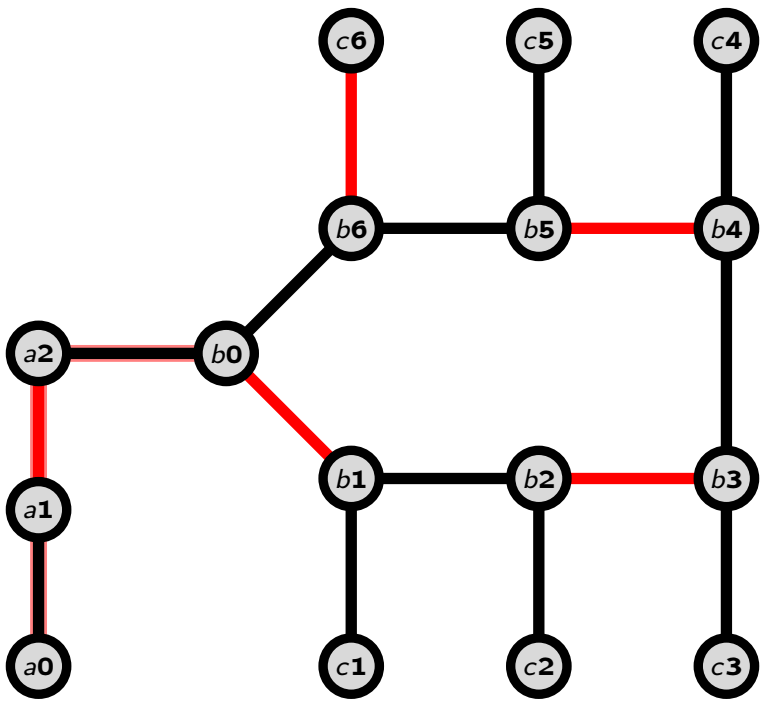
$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.
- Ausblick und Vorgehen:
  - Untersuche die möglichen Situationen.
  - Erkenne die Situationen, die kritisch für den Algorithmus sind.
  - Passe Algorithmus für diese Situationen an.

Erster unkritischer Fall

$l(e) = -g_M(e)$

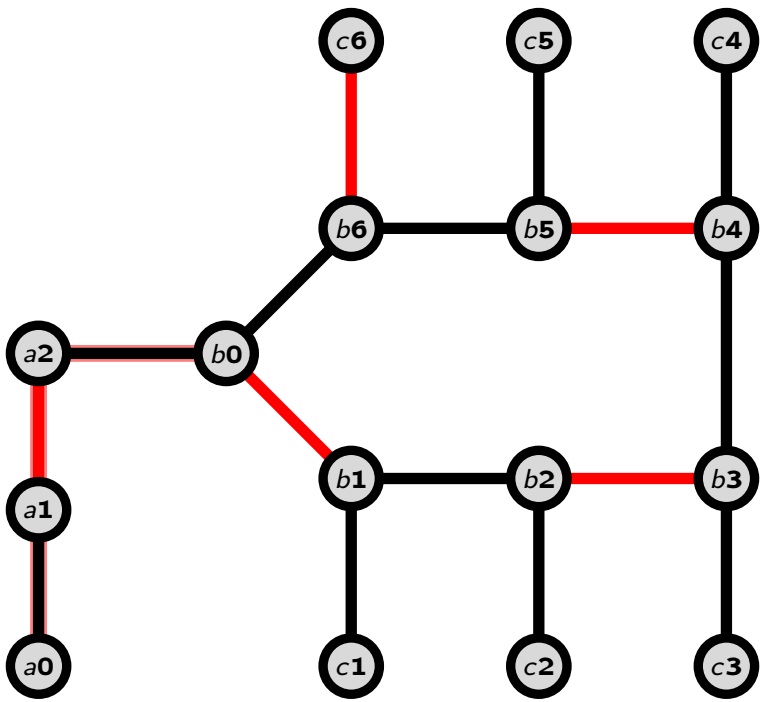
- Die Suche startet an a_0.



Erster unkritischer Fall

$l(e) = -g_M(e)$

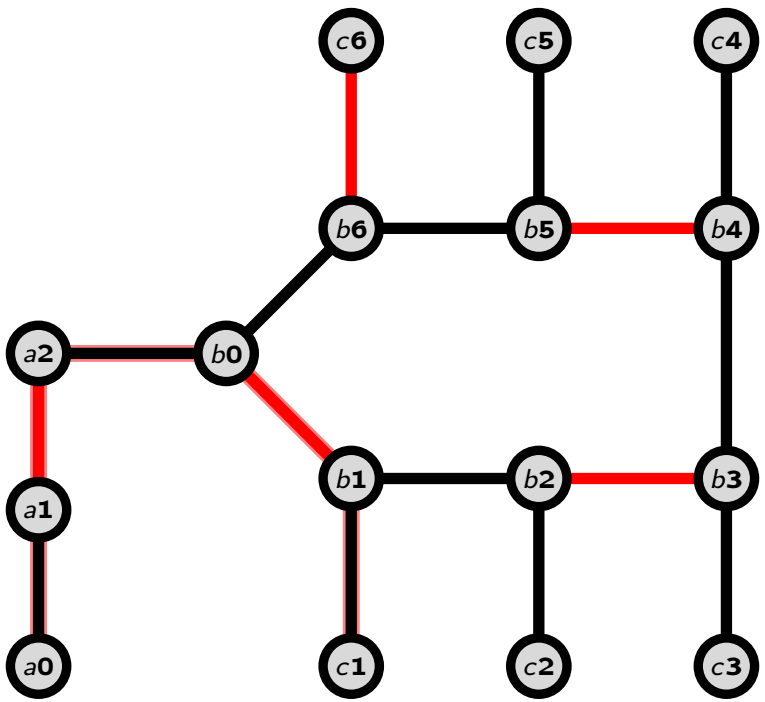
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.



Erster unkritischer Fall

$l(e) = -g_M(e)$

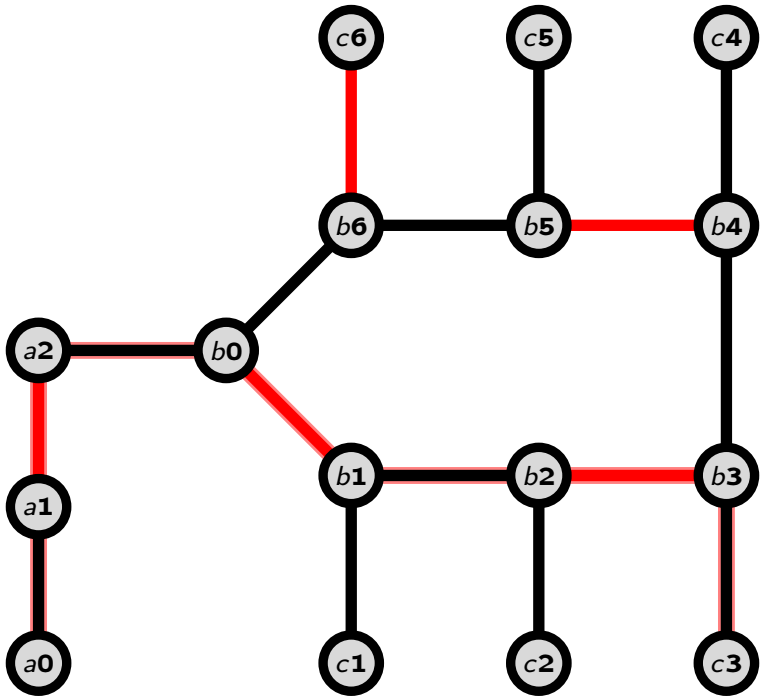
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Wegen $\{b_0, b_1\} \in M$ wird der Kreis in eindeutiger Richtung durchlaufen.



Erster unkritischer Fall

$l(e) = -g_M(e)$

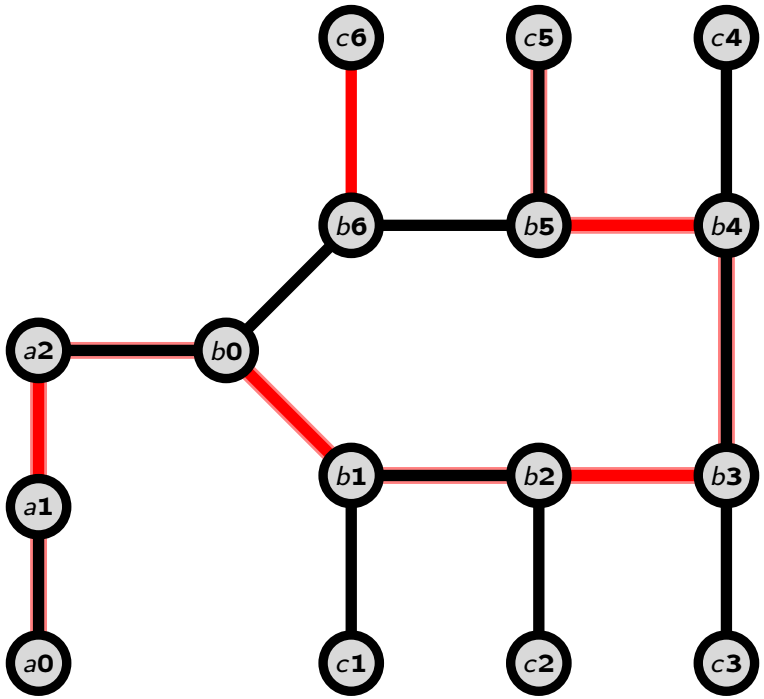
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Wegen $\{b_0, b_1\} \in M$ wird der Kreis in eindeutiger Richtung durchlaufen.
- Damit ist die Kante $\{b_0, b_6\}$ nicht mehr relevant.



Erster unkritischer Fall

$l(e) = -g_M(e)$

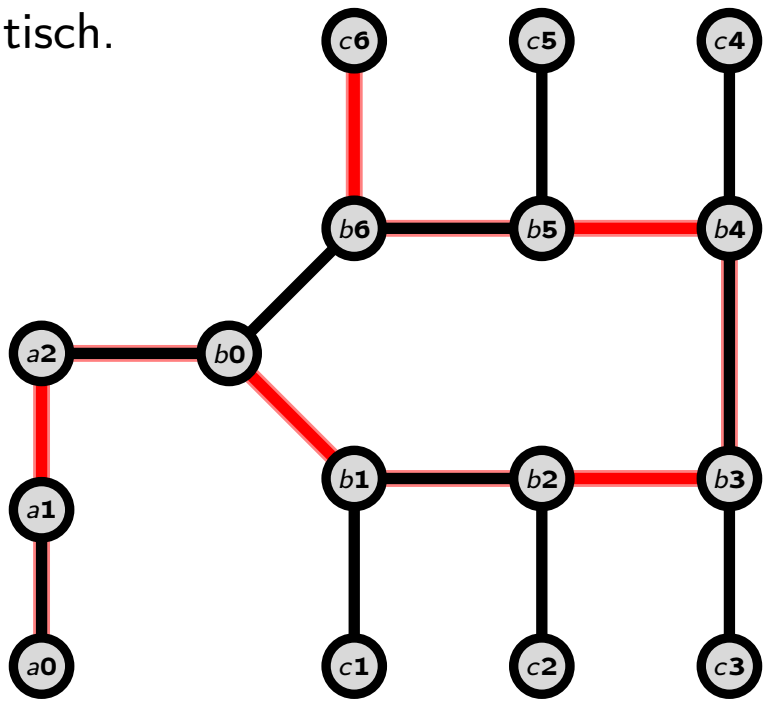
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Wegen $\{b_0, b_1\} \in M$ wird der Kreis in eindeutiger Richtung durchlaufen.
- Damit ist die Kante $\{b_0, b_6\}$ nicht mehr relevant.
- Die Suche sieht keinen Kreis.



Erster unkritischer Fall

$l(e) = -g_M(e)$

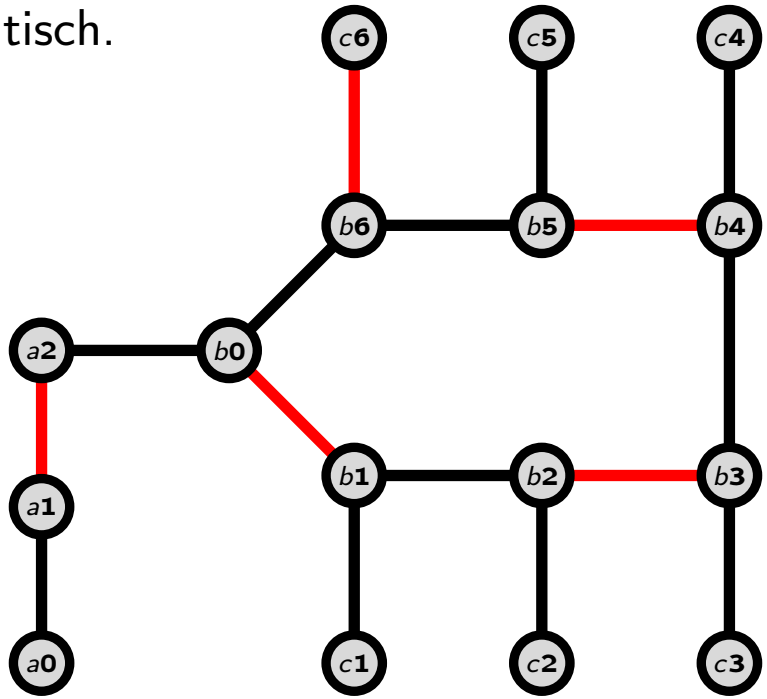
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Wegen $\{b_0, b_1\} \in M$ wird der Kreis in eindeutiger Richtung durchlaufen.
- Damit ist die Kante $\{b_0, b_6\}$ nicht mehr relevant.
- Die Suche sieht keinen Kreis.
- Der Fall ist unkritisch.



Erster unkritischer Fall

$l(e) = -g_M(e)$

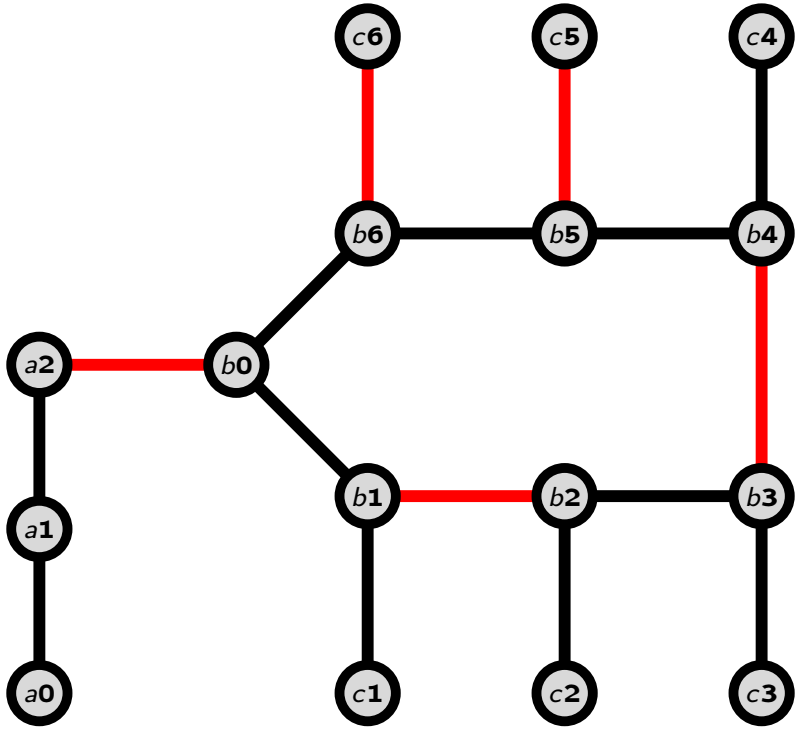
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Wegen $\{b_0, b_1\} \in M$ wird der Kreis in eindeutiger Richtung durchlaufen.
- Damit ist die Kante $\{b_0, b_6\}$ nicht mehr relevant.
- Die Suche sieht keinen Kreis.
- Der Fall ist unkritisch.



Zweiter unkritischer Fall

$l(e) = -g_M(e)$

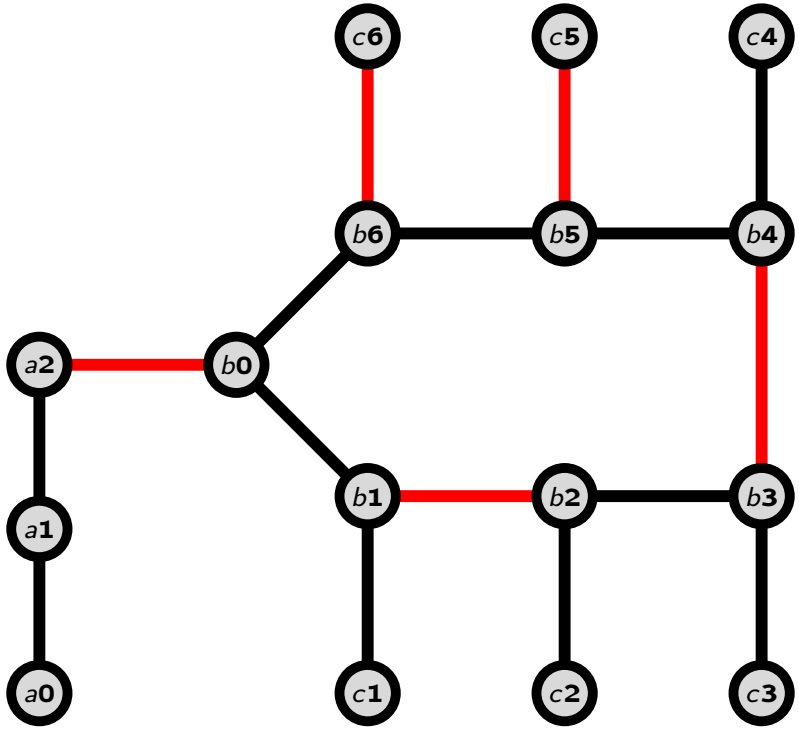
- Die Suche startet an a_1.



Zweiter unkritischer Fall

$l(e) = -g_M(e)$

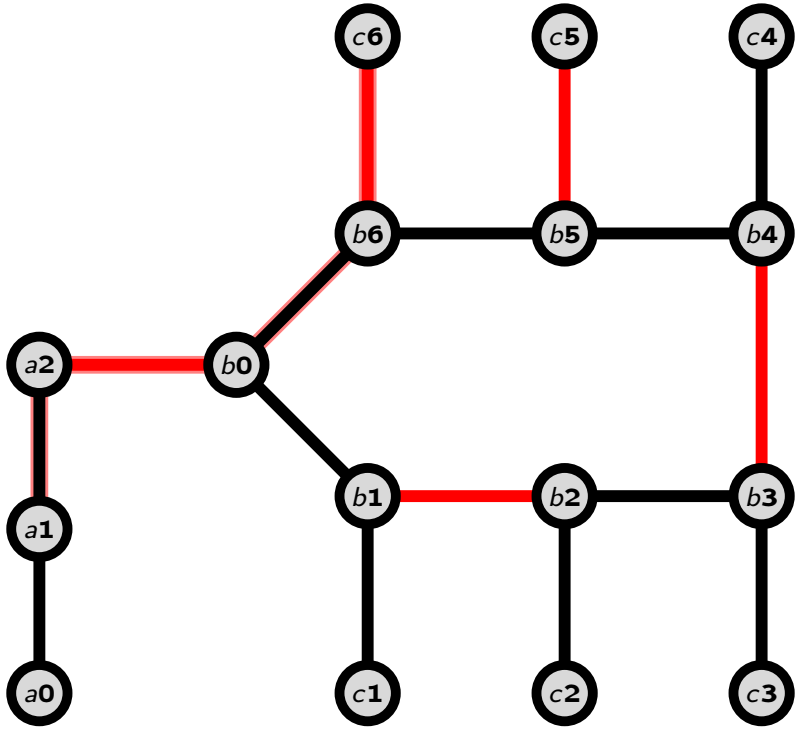
- Die Suche startet an a_1.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.



Zweiter unkritischer Fall

$l(e) = -g_M(e)$

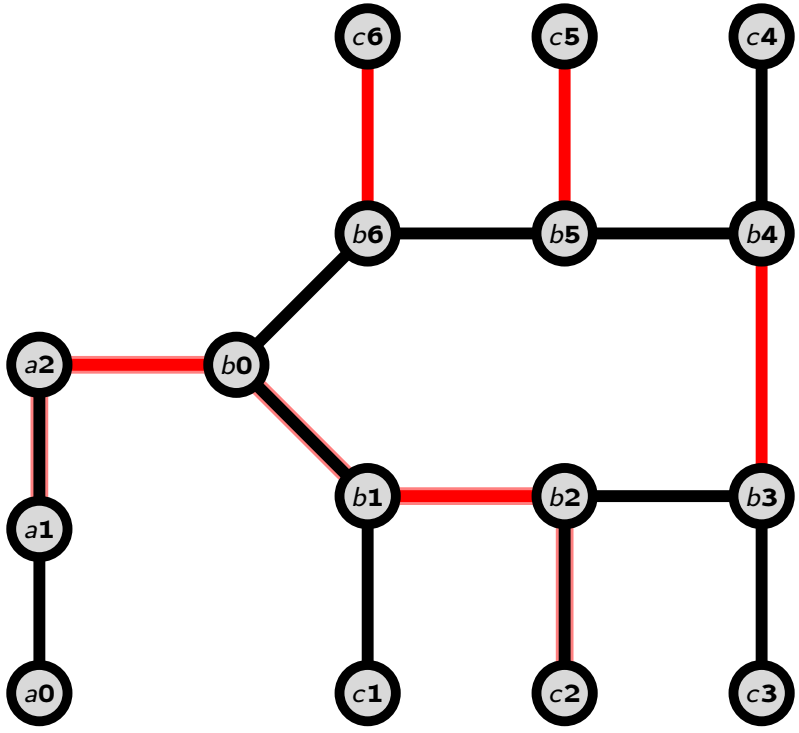
- Die Suche startet an a_1.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Es gibt zwei Möglichkeiten, in den Kreis zu laufen.



Zweiter unkritischer Fall

$l(e) = -g_M(e)$

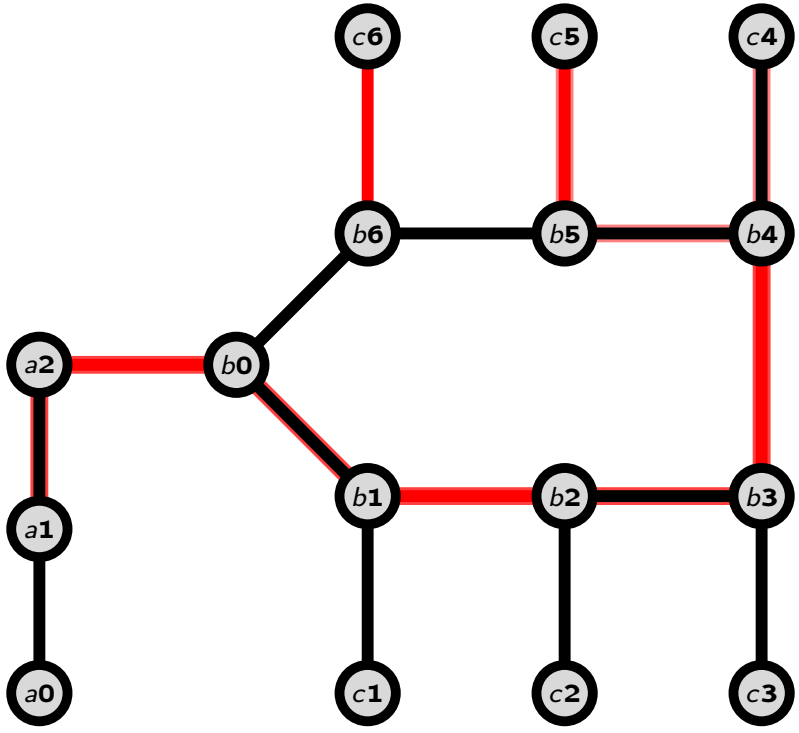
- Die Suche startet an a_1.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Es gibt zwei Möglichkeiten, in den Kreis zu laufen.



Zweiter unkritischer Fall

$l(e) = -g_M(e)$

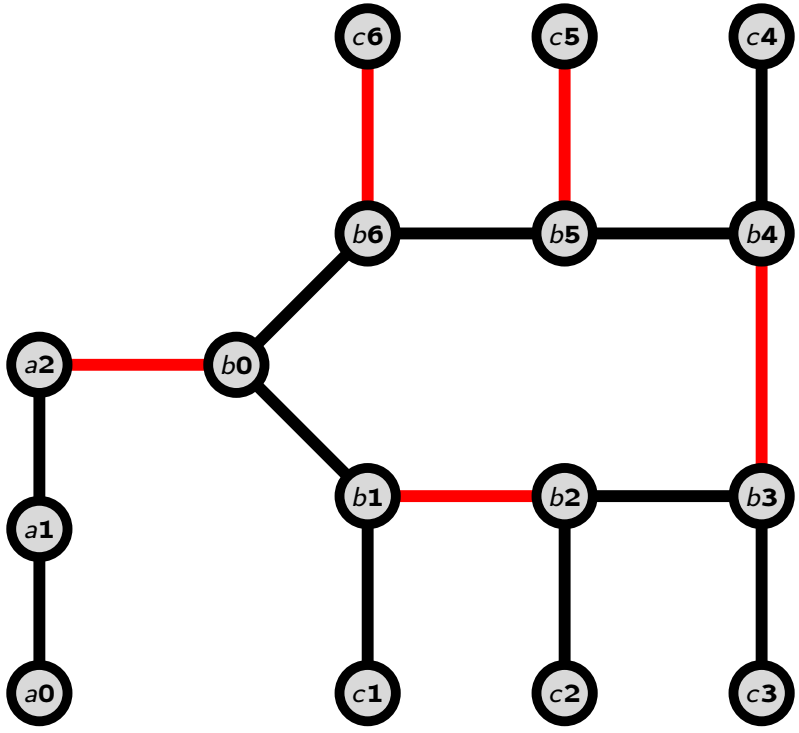
- Die Suche startet an a_1.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Es gibt zwei Möglichkeiten, in den Kreis zu laufen.
- Diese sind unabhängig, da b_5 frei in $G|C$ ($\{b_5, b_6\}$ nicht relevant)



Zweiter unkritischer Fall

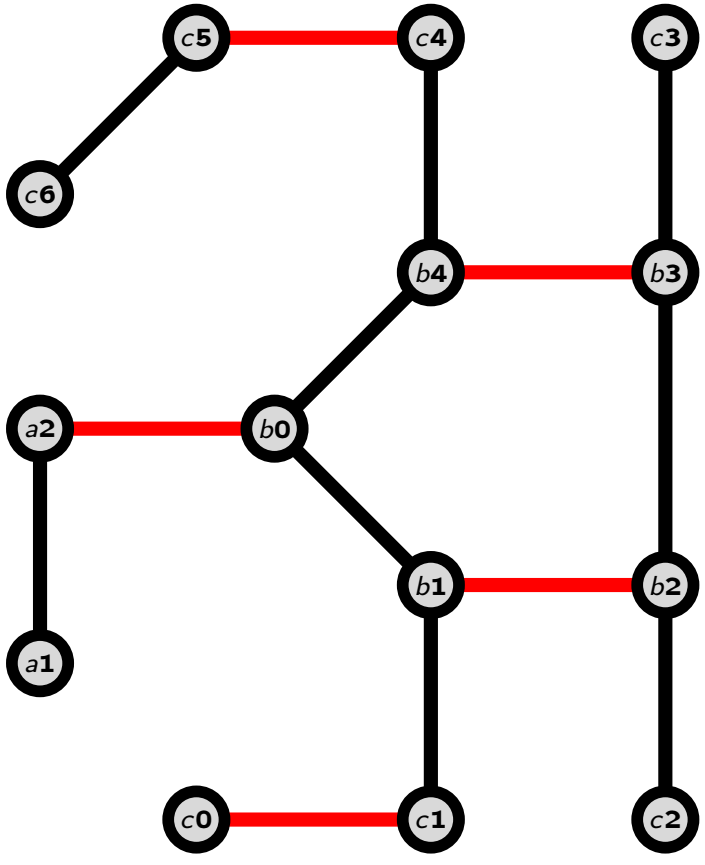
$l(e) = -g_M(e)$

- Die Suche startet an a_1.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Es gibt zwei Möglichkeiten, in den Kreis zu laufen.
- Diese sind unabhängig, da b_5 frei in $G|C$ ($\{b_5, b_6\}$ nicht relevant)
- Der Fall ist unkritisch.



Beispiel

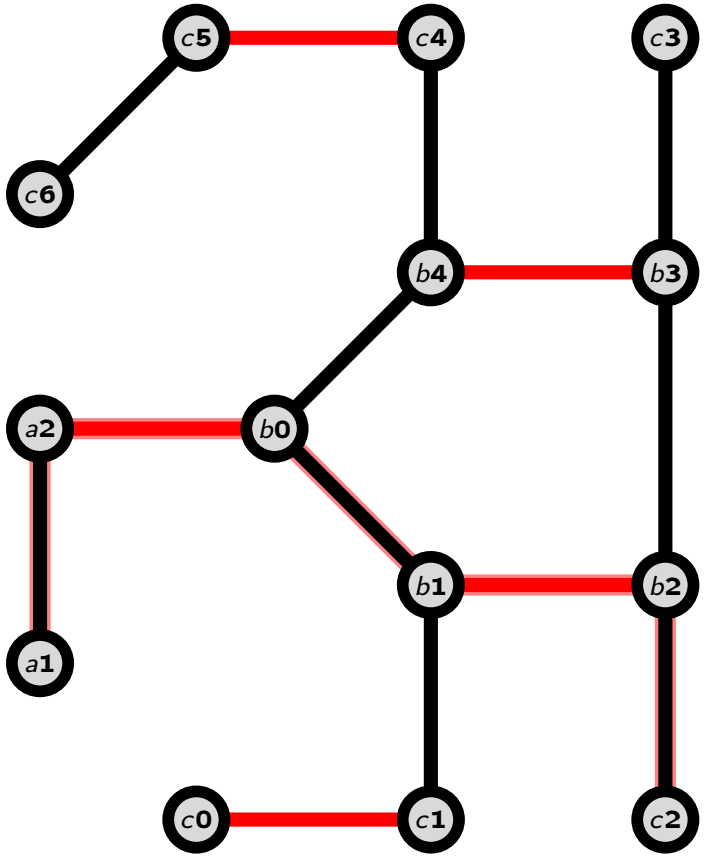
$l(e) = -g_M(e)$



Beispiel

$l(e) = -g_M(e)$

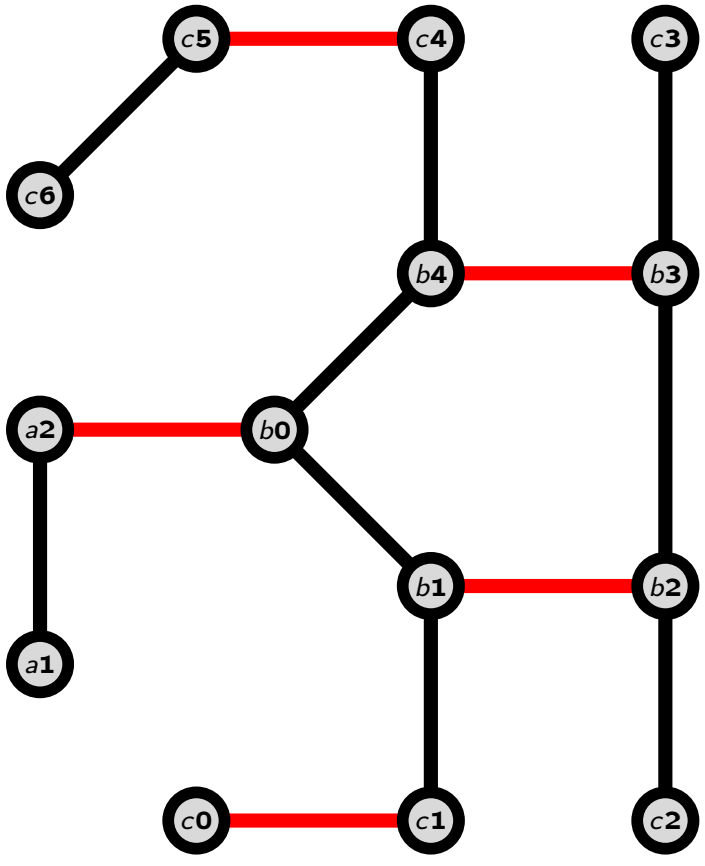
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2



Beispiel

$l(e) = -g_M(e)$

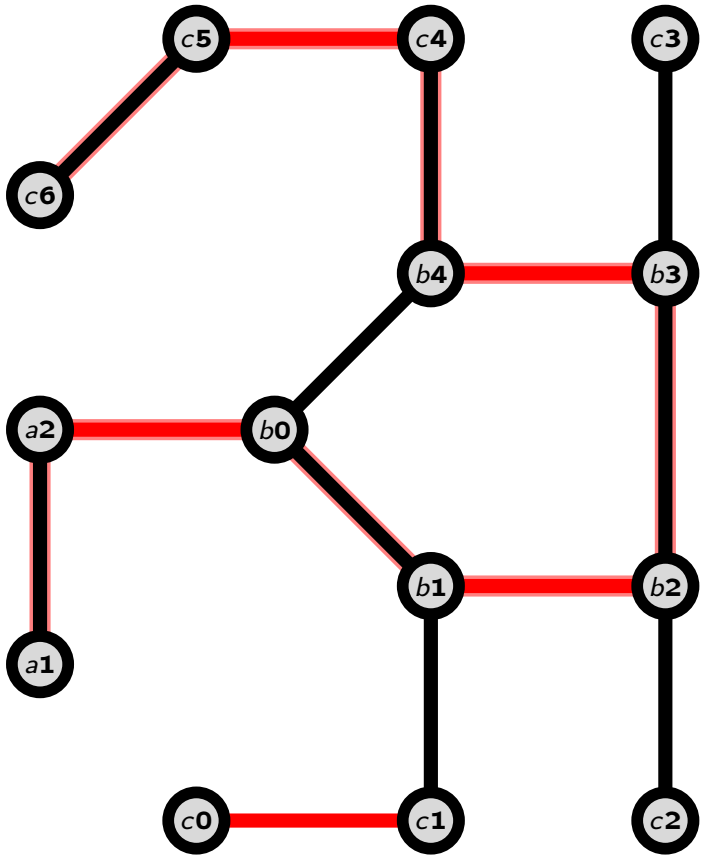
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2



Beispiel

$l(e) = -g_M(e)$

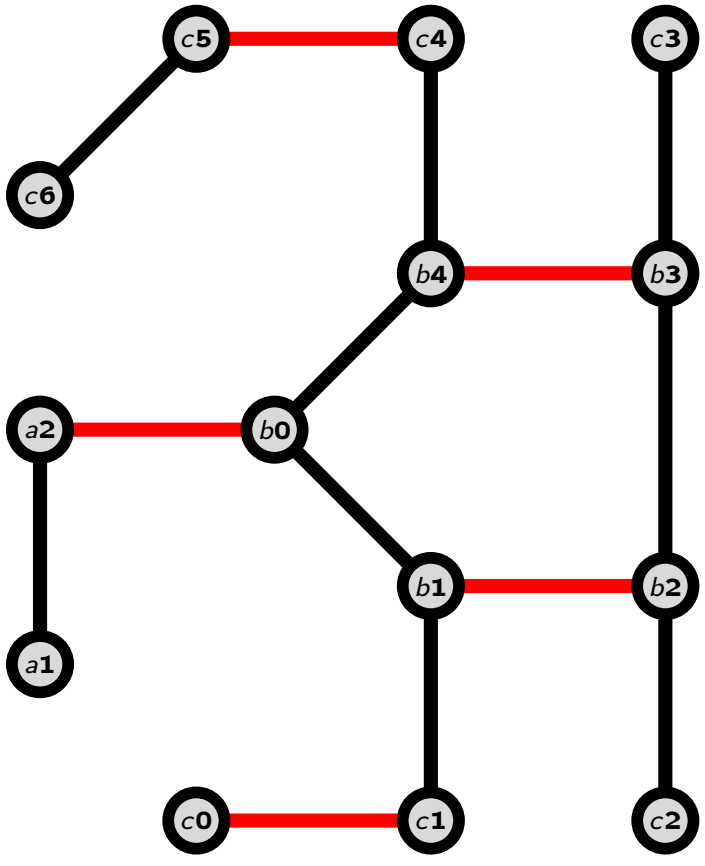
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6



Beispiel

$l(e) = -g_M(e)$

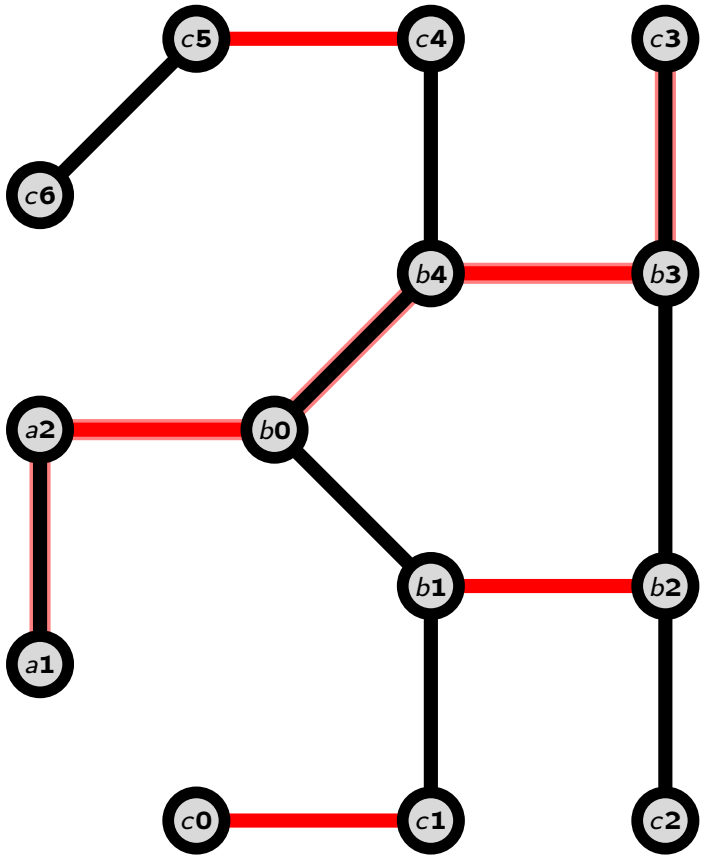
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6



Beispiel

$l(e) = -g_M(e)$

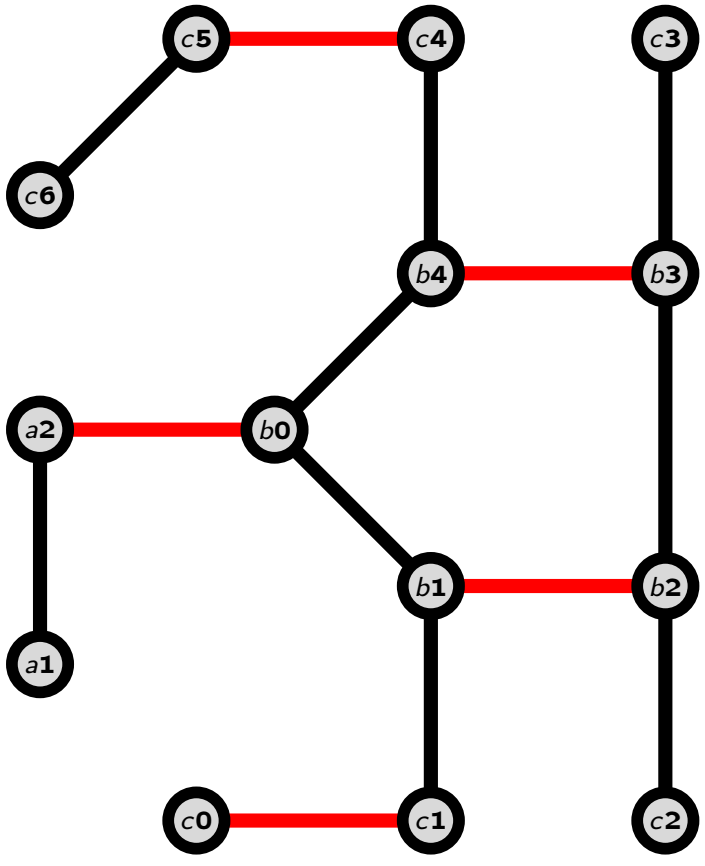
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3



Beispiel

$l(e) = -g_M(e)$

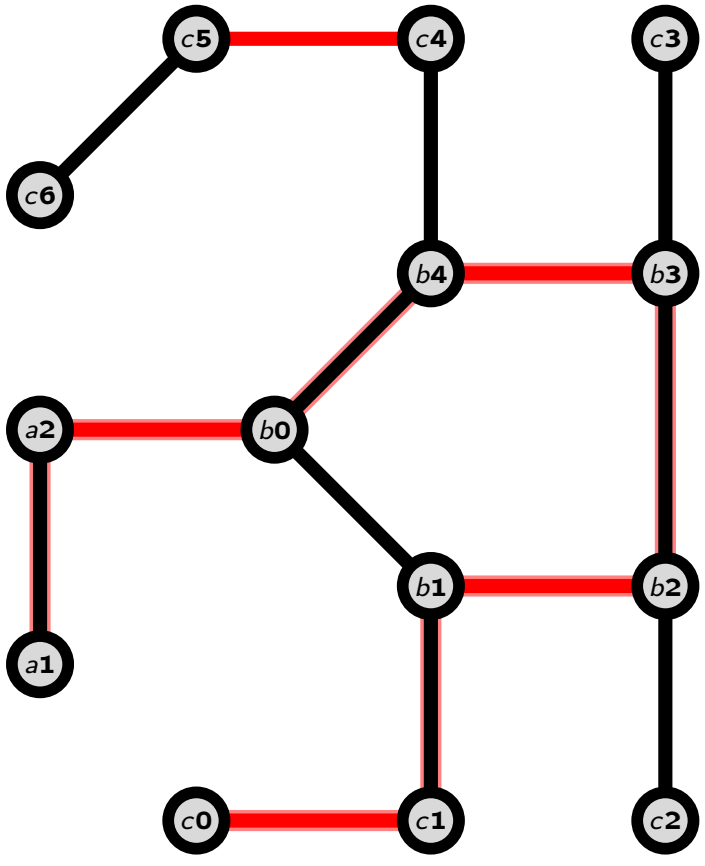
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3



Beispiel

$l(e) = -g_M(e)$

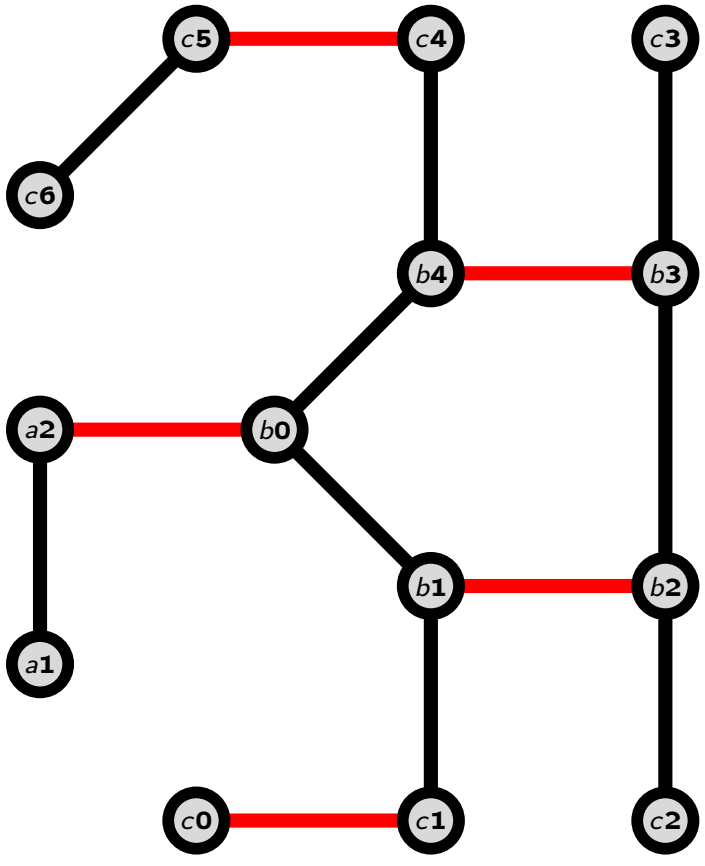
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0



Beispiel

$l(e) = -g_M(e)$

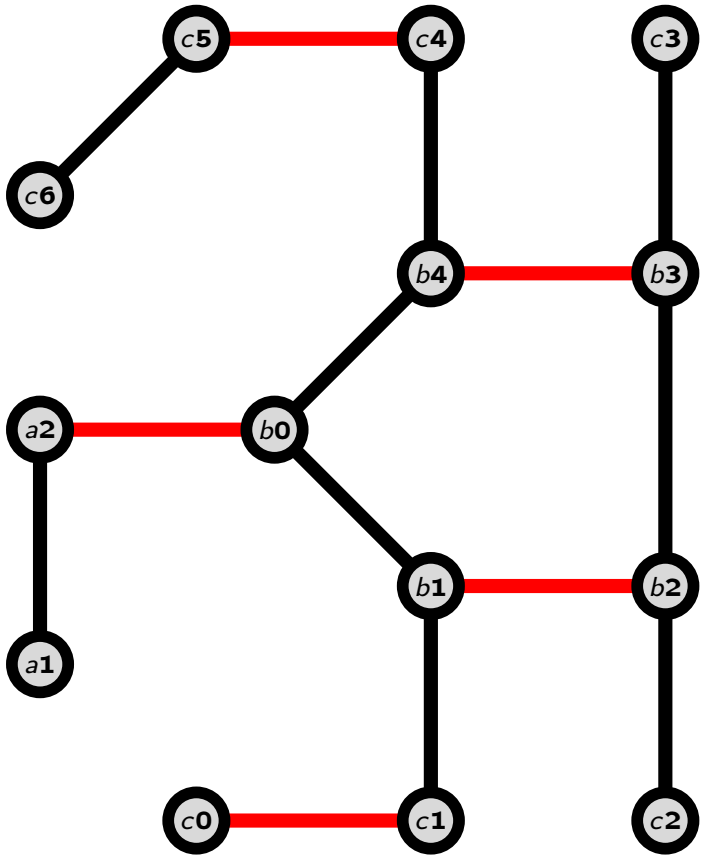
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0



Beispiel

$l(e) = -g_M(e)$

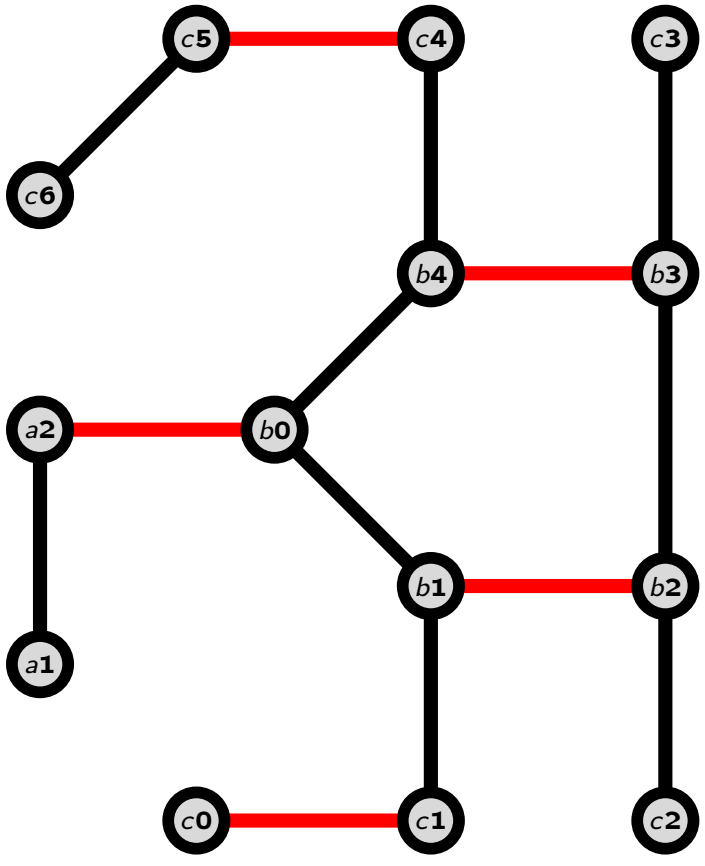
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.



Beispiel

$l(e) = -g_M(e)$

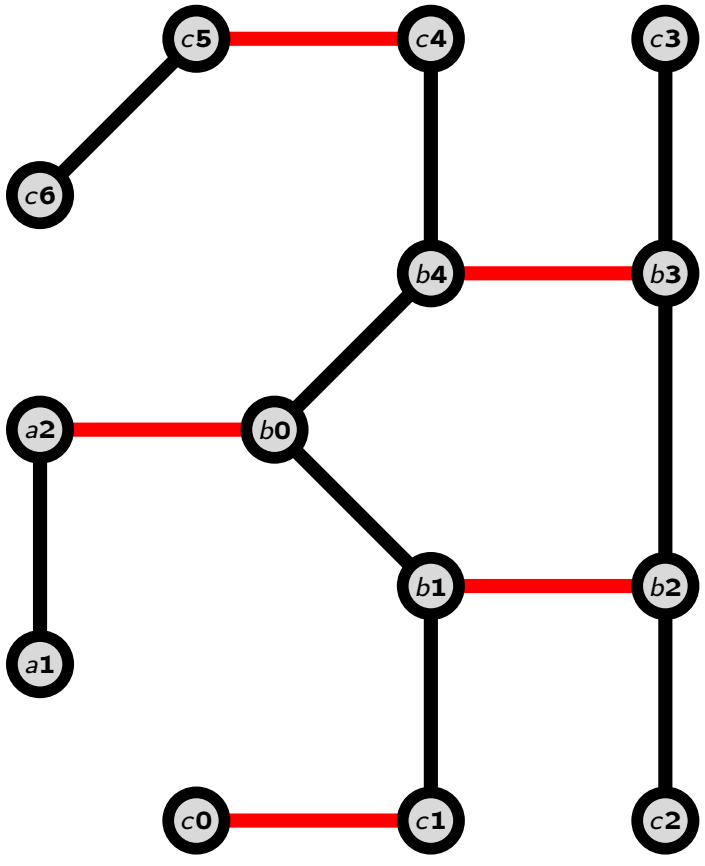
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.



Beispiel

$l(e) = -g_M(e)$

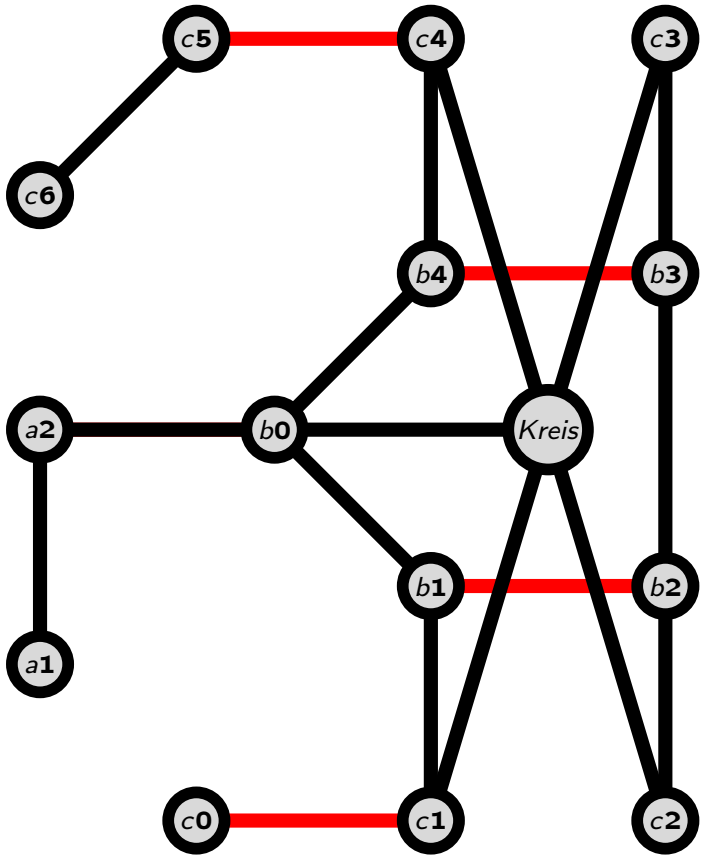
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Beispiel

$l(e) = -g_M(e)$

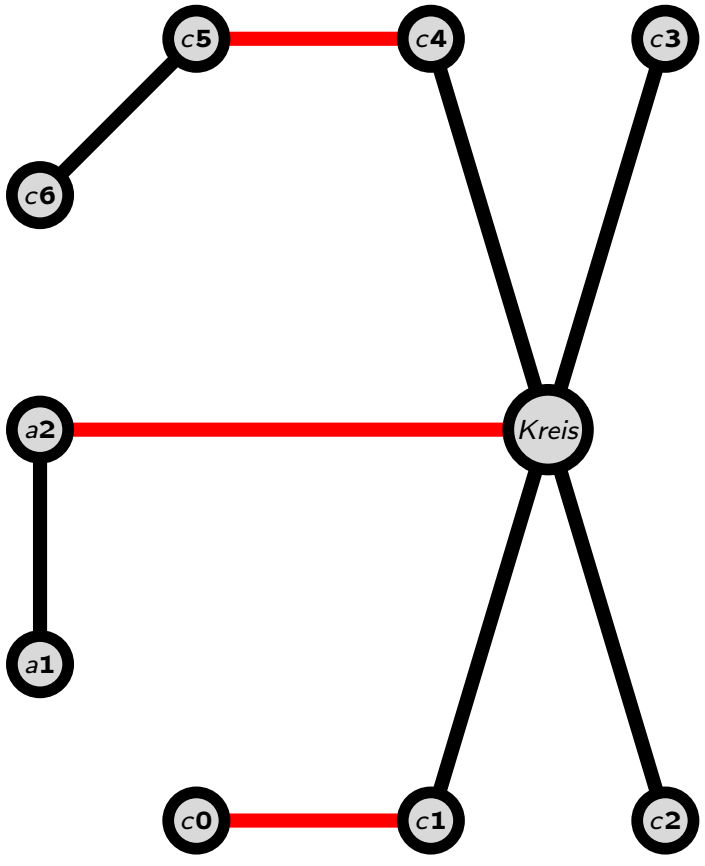
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Beispiel

$l(e) = -g_M(e)$

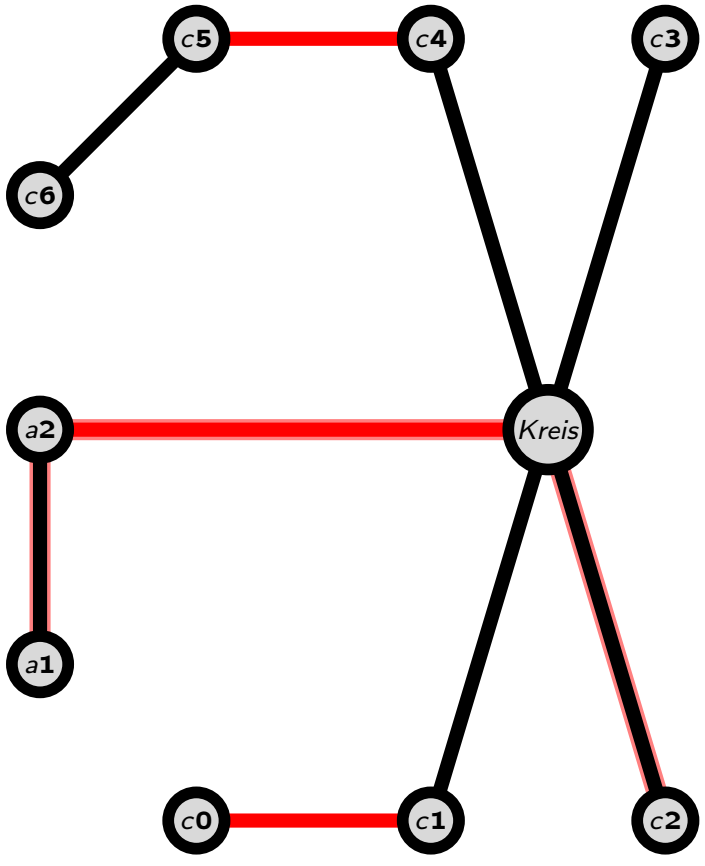
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Beispiel

$l(e) = -g_M(e)$

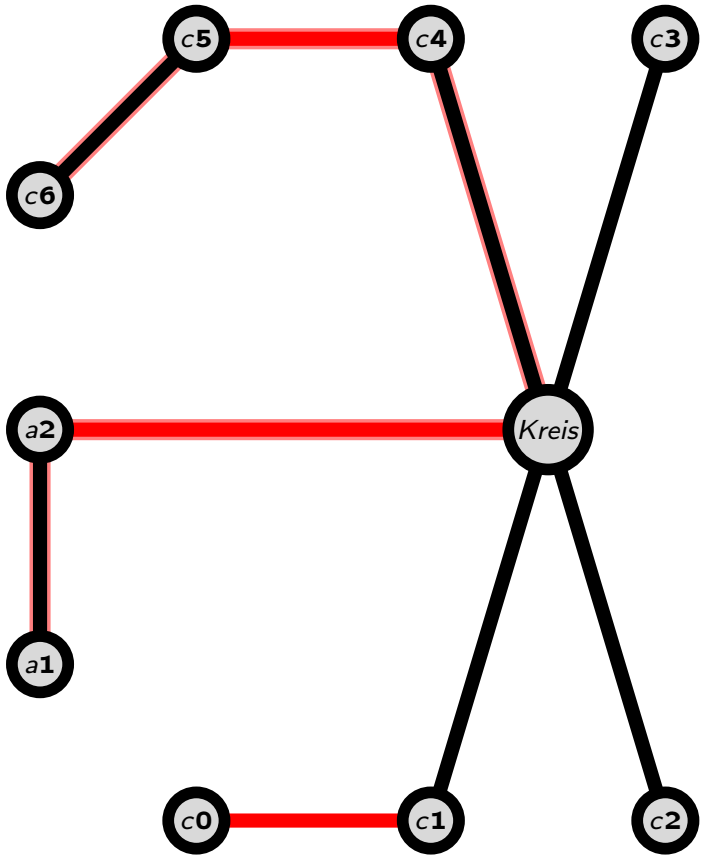
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Beispiel

$l(e) = -g_M(e)$

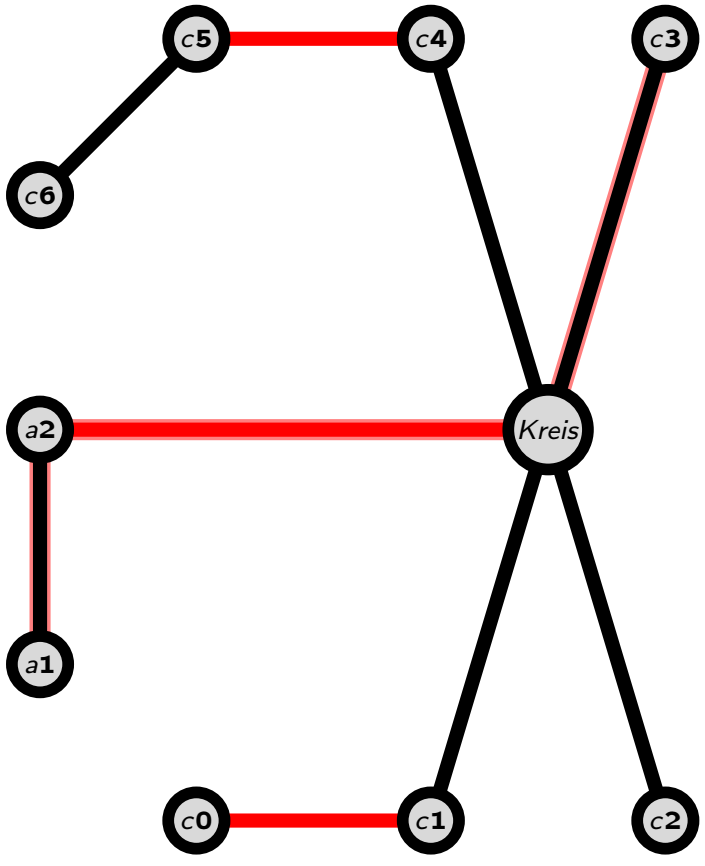
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Beispiel

$l(e) = -g_M(e)$

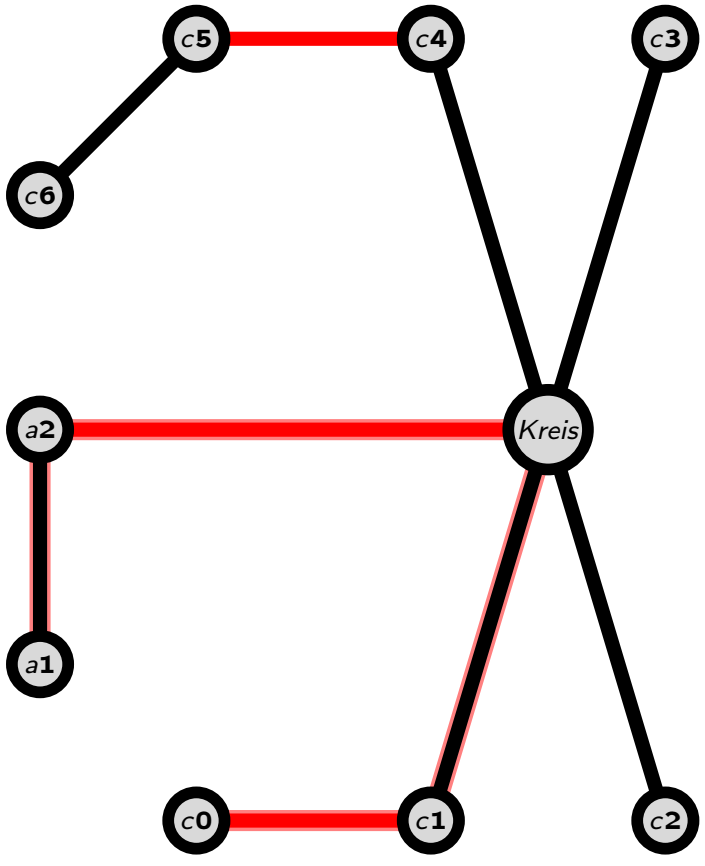
- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Beispiel

$l(e) = -g_M(e)$

- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

v_0 ist der Startknoten der Blüte C.
Definiere $S(G, C) = (V', E' \cup E'')$ mit:

- $V' = (V \setminus C) \dot\cup \{c\}$

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

v_0 ist der Startknoten der Blüte C.
Definiere $S(G, C) = (V', E' \cup E'')$ mit:

- $V' = (V \setminus C) \dot\cup \{c\}$
- $E' = \{\{v, w\} \mid \{v, w\} \in E \wedge v, w \in V'\}$

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

v_0 ist der Startknoten der Blüte C.
Definiere $S(G, C) = (V', E' \cup E'')$ mit:

- $V' = (V \setminus C) \dot\cup \{c\}$
- $E' = \{\{v, w\} \mid \{v, w\} \in E \wedge v, w \in V'\}$
- $E'' = \{\{v, c\} \mid \exists w \in C : \{v, w\} \in E\}$

Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

v_0 ist der Startknoten der Blüte C.
Definiere $S(G, C) = (V', E' \cup E'')$ mit:

- $V' = (V \setminus C) \dot\cup \{c\}$
- $E' = \{\{v, w\} \mid \{v, w\} \in E \wedge v, w \in V'\}$
- $E'' = \{\{v, c\} \mid \exists w \in C : \{v, w\} \in E\}$

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.
2. Setze: $G' = (V, E')$ mit $E' = \{(v, w) \mid \{v, w\} \in E\}$.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.
2. Setze: $G' = (V, E')$ mit $E' = \{(v, w) \mid \{v, w\} \in E\}$.
3. Für alle $\{v, w\} \in M$ setze: $P(v) = w$ und $P(w) = v$.

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.
2. Setze: $G' = (V, E')$ mit $E' = \{(v, w) \mid \{v, w\} \in E\}$.
3. Für alle $\{v, w\} \in M$ setze: $P(v) = w$ und $P(w) = v$.
4. Setze: $Z(v) =$ even für alle $v \in V$ mit $v \cap \cup_{e \in E} e \neq \emptyset$

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.
2. Setze: $G' = (V, E')$ mit $E' = \{(v, w) \mid \{v, w\} \in E\}$.
3. Für alle $\{v, w\} \in M$ setze: $P(v) = w$ und $P(w) = v$.
4. Setze: $Z(v) =$ even für alle $v \in V$ mit $v \cap \cup_{e \in E} e \neq \emptyset$
5. Setze: $Z(v) =$ away für alle $v \in V$ mit $v \cap \cup_{e \in E} e = \emptyset$

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.
2. Setze: $G' = (V, E')$ mit $E' = \{(v, w) \mid \{v, w\} \in E\}$.
3. Für alle $\{v, w\} \in M$ setze: $P(v) = w$ und $P(w) = v$.
4. Setze: $Z(v) =$ even für alle $v \in V$ mit $v \cap \cup_{e \in E} e \neq \emptyset$
5. Setze: $Z(v) =$ away für alle $v \in V$ mit $v \cap \cup_{e \in E} e = \emptyset$

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).

- Fall 1:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.

- Fall 1:
- Fall 2:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.

- Fall 1:
- Fall 2:
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.

- Fall 1:
- Fall 2:
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
- Fall 2:
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
- Fall 2:
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
  2. Schrumpfe Blüte $v, v', \ldots, v'', \ldots, v$.
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
  2. Schrumpfe Blüte $v, v', \ldots, v'', \ldots, v$.
  3. Passe dabei Daten p an.
- Fall 3:

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
  2. Schrumpfe Blüte $v, v', \ldots, v'', \ldots, v$.
  3. Passe dabei Daten p an.
- Fall 3:
  1. Verbinde v und v'.

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
  2. Schrumpfe Blüte $v, v', \ldots, v'', \ldots, v$.
  3. Passe dabei Daten p an.
- Fall 3:
  1. Verbinde v und v'.
  2. Verbessernder Pfad ist gefunden.

Algorithmus (Edmonds)

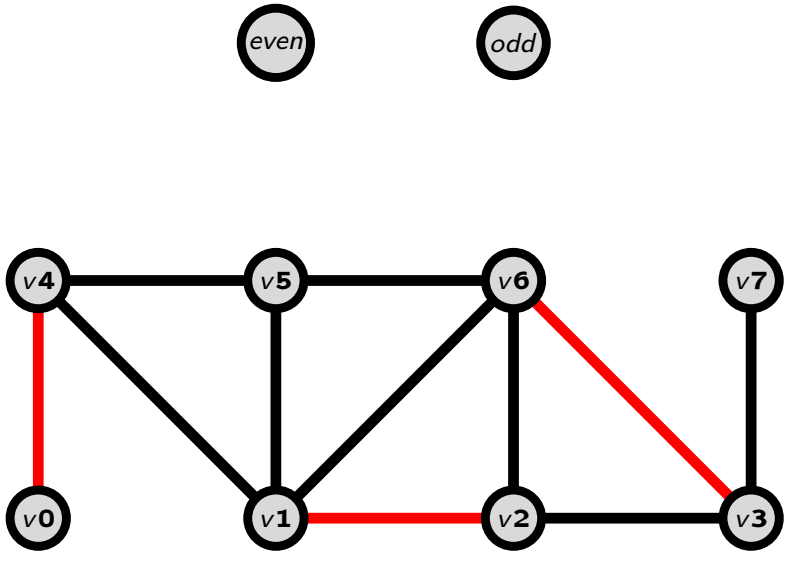
$l(e) = -g_M(e)$

1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
  2. Schrumpfe Blüte $v, v', \ldots, v'', \ldots, v$.
  3. Passe dabei Daten p an.
- Fall 3:
  1. Verbinde v und v'.
  2. Verbessernder Pfad ist gefunden.

Beispiel

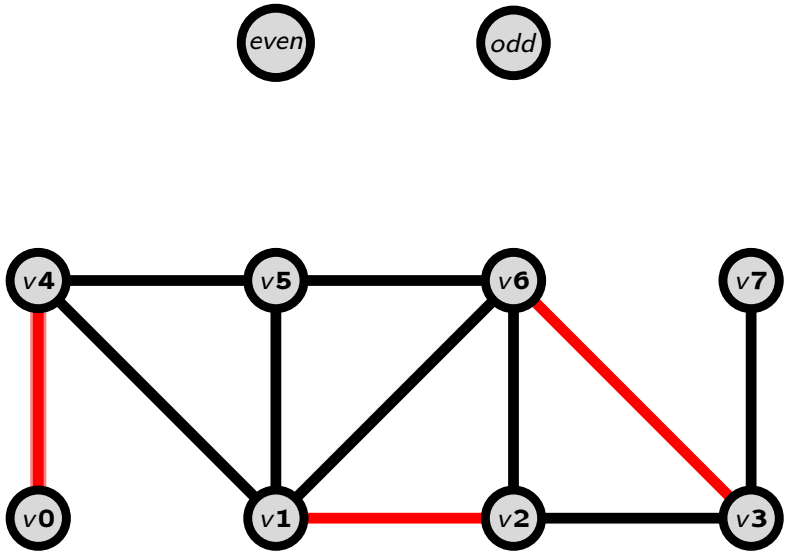
$l(e) = -g_M(e)$



Beispiel

$l(e) = -g_M(e)$

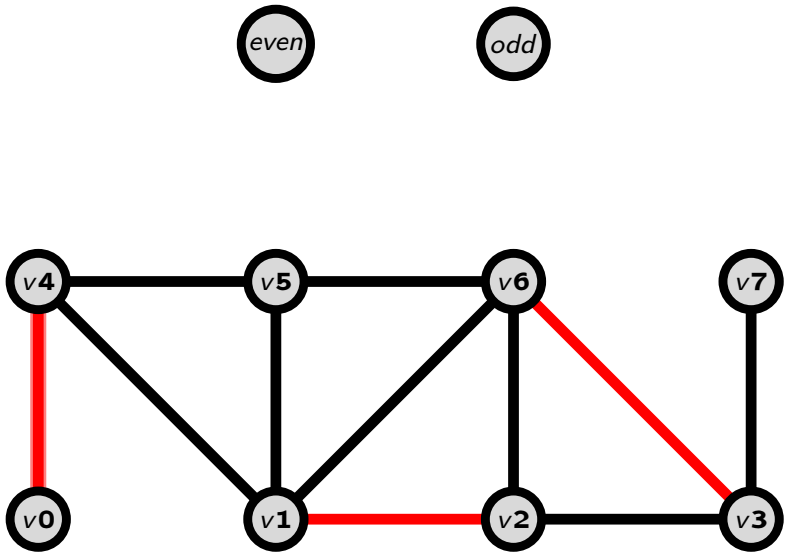
- Wähle Kante v_0, v_4.



Beispiel

$l(e) = -g_M(e)$

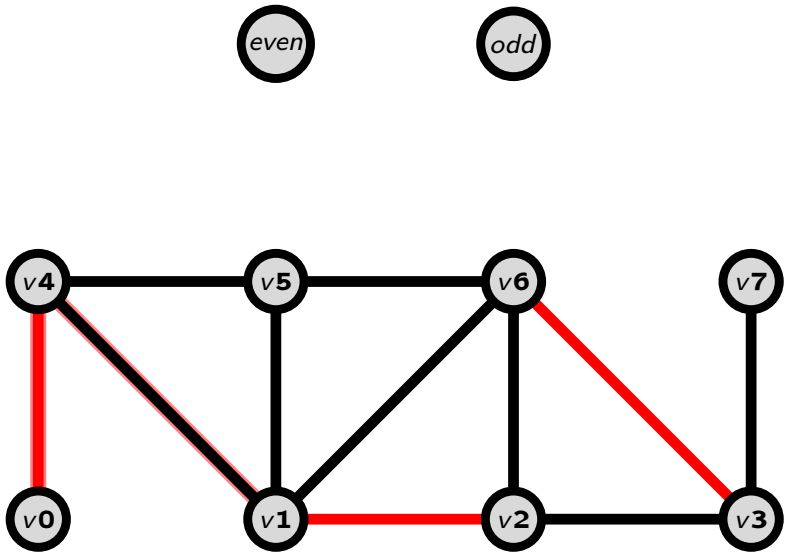
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4



Beispiel

$l(e) = -g_M(e)$

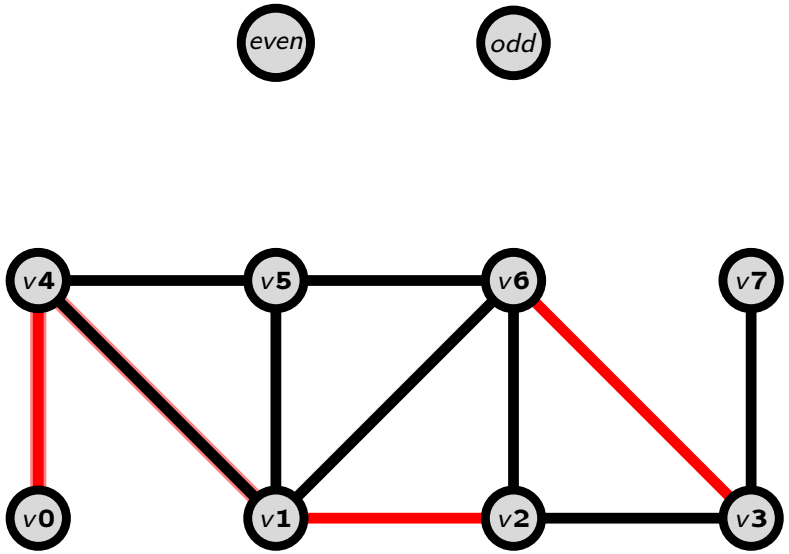
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.



Beispiel

$l(e) = -g_M(e)$

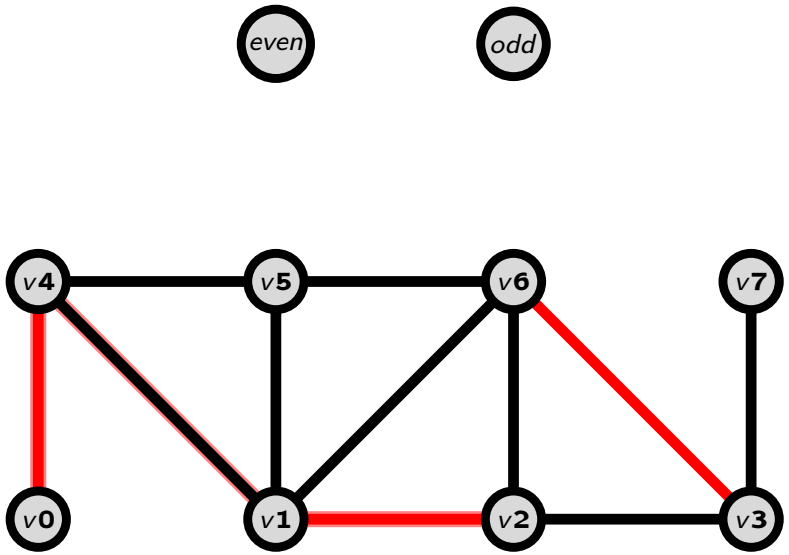
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1



Beispiel

$l(e) = -g_M(e)$

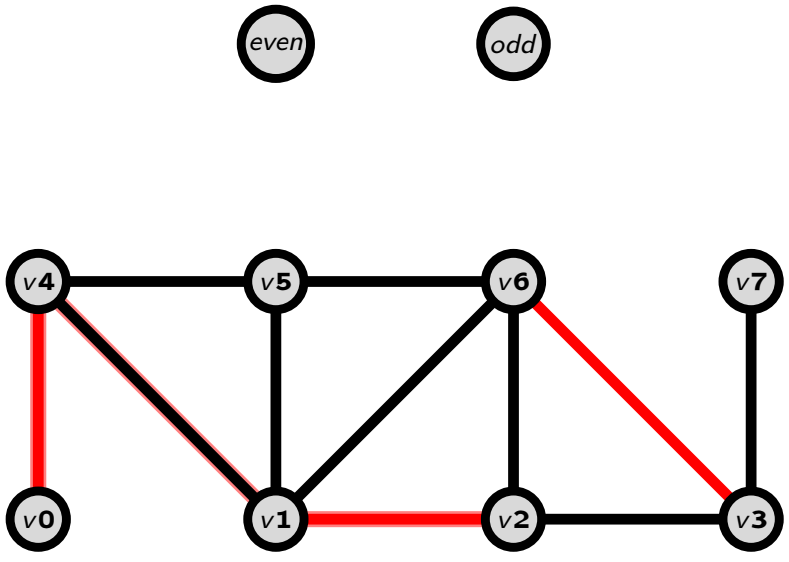
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.



Beispiel

$l(e) = -g_M(e)$

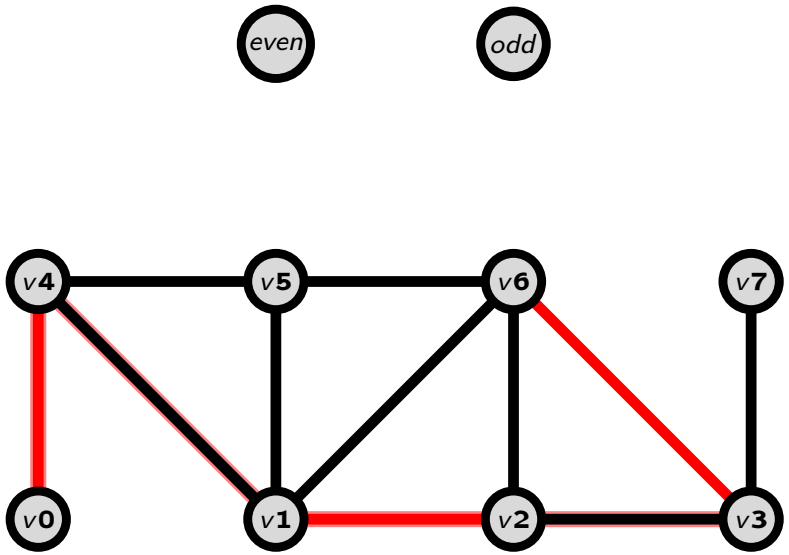
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2



Beispiel

$l(e) = -g_M(e)$

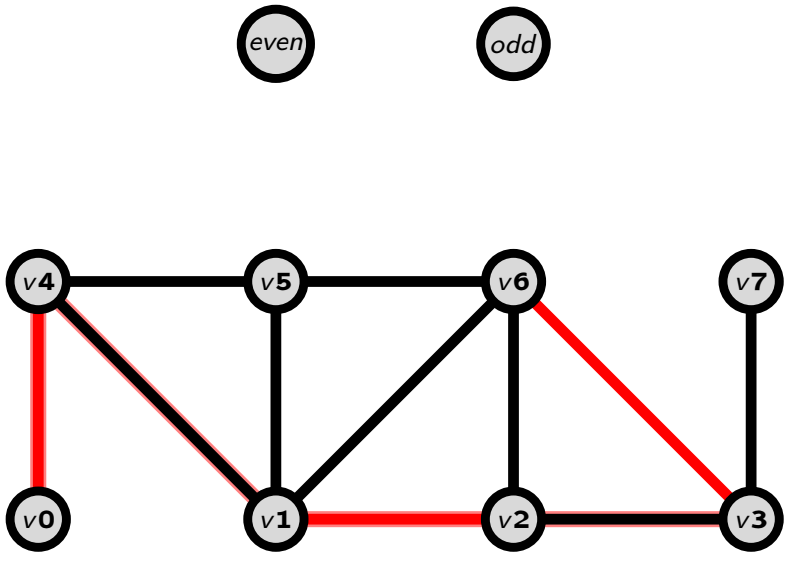
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.



Beispiel

$l(e) = -g_M(e)$

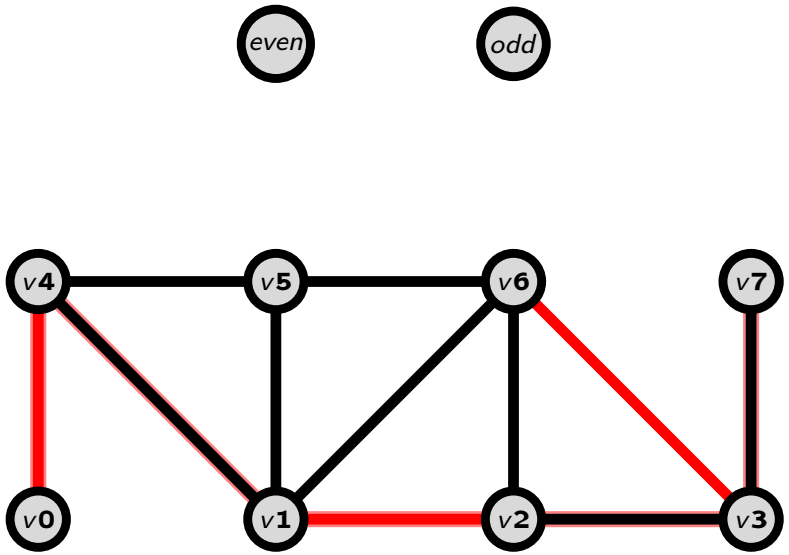
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3



Beispiel

$l(e) = -g_M(e)$

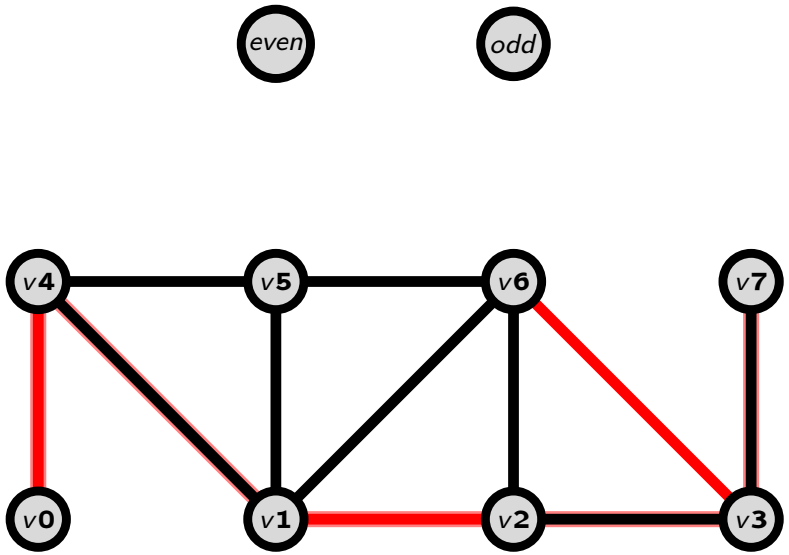
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3
- Wähle Kante v_3, v_7.



Beispiel

$l(e) = -g_M(e)$

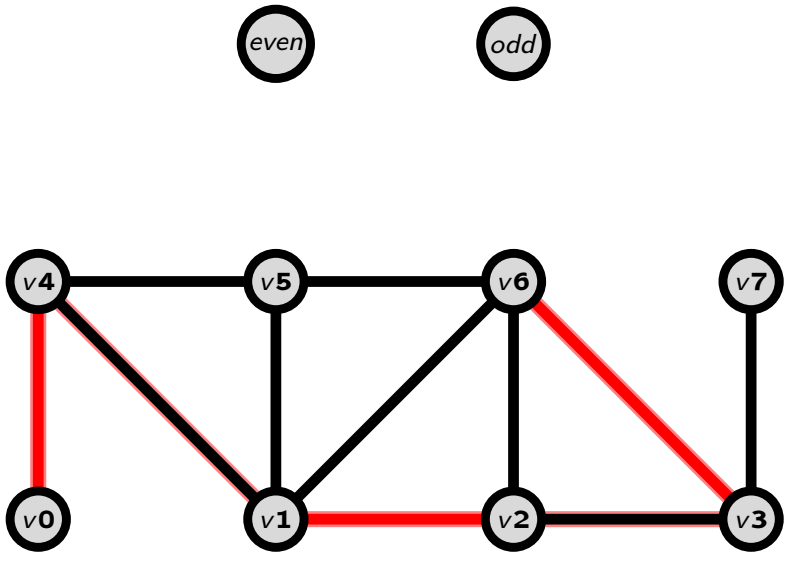
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3
- Wähle Kante v_3, v_7.
- Fall 0, v_3, v_7 ist betrachtet worden.



Beispiel

$l(e) = -g_M(e)$

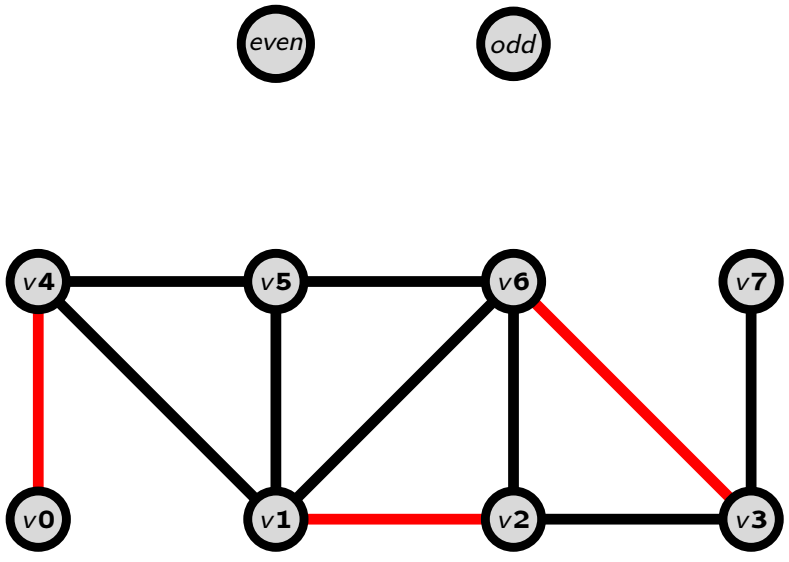
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3
- Wähle Kante v_3, v_7.
- Fall 0, v_3, v_7 ist betrachtet worden.
- Wähle Kante v_3, v_6.



Beispiel

$l(e) = -g_M(e)$

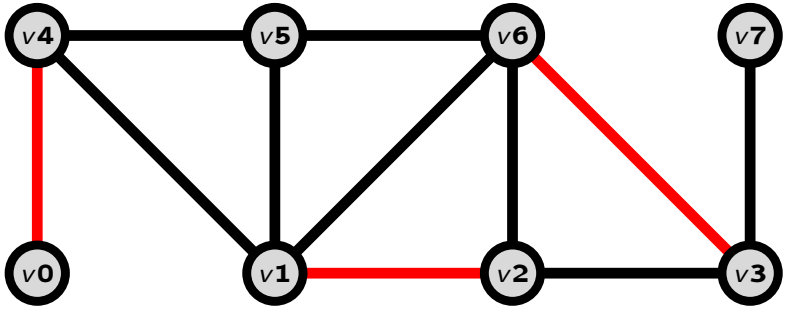
- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3
- Wähle Kante v_3, v_7.
- Fall 0, v_3, v_7 ist betrachtet worden.
- Wähle Kante v_3, v_6.
- Fall 2, es wird geschrumpft.

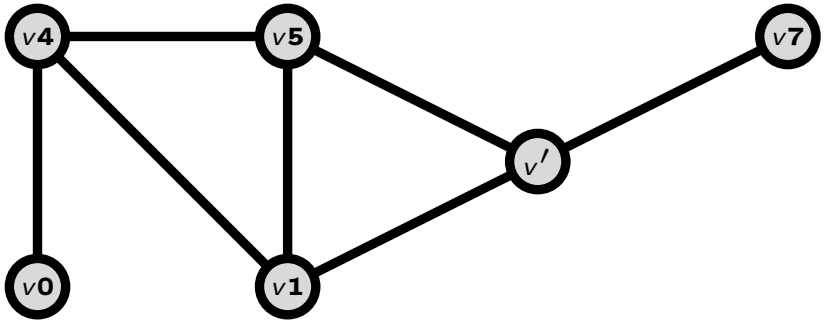


Beispiel

$l(e) = -g_M(e)$

- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3
- Wähle Kante v_3, v_7.
- Fall 0, v_3, v_7 ist betrachtet worden.
- Wähle Kante v_3, v_6.
- Fall 2, es wird geschrumpft.





Ergebnisse

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

Ergebnisse

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

- Kombiniere obigen Algorithmus mit der Suche nach kurzen verbessernden Pfaden.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

- Kombiniere obigen Algorithmus mit der Suche nach kurzen verbessernden Pfaden.
- D.h. adaptiere obigen Algorithmus in eine Breitensuche.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

- Kombiniere obigen Algorithmus mit der Suche nach kurzen verbessernden Pfaden.
- D.h. adaptiere obigen Algorithmus in eine Breitensuche.
- Verwalte dabei die Pfadlängen korrekt, d.h. beachte die Länge in den Blüten mit.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

- Kombiniere obigen Algorithmus mit der Suche nach kurzen verbessernden Pfaden.
- D.h. adaptiere obigen Algorithmus in eine Breitensuche.
- Verwalte dabei die Pfadlängen korrekt, d.h. beachte die Länge in den Blüten mit.
- Die Pfadlänge in den Blüten ist abhängig von den Kanten, die wir zum Betreten und Verlassen nutzen.

Ergebnisse

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

- Kombiniere obigen Algorithmus mit der Suche nach kurzen verbessernden Pfaden.
- D.h. adaptiere obigen Algorithmus in eine Breitensuche.
- Verwalte dabei die Pfadlängen korrekt, d.h. beachte die Länge in den Blüten mit.
- Die Pfadlänge in den Blüten ist abhängig von den Kanten, die wir zum Betreten und Verlassen nutzen.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

Beweisidee:

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

Beweisidee:

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

Beweisidee:

- Suche verbessernde Pfade und

Ergebnisse

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

Beweisidee:

- Suche verbessernde Pfade und
- Suche kostenverbessernde alternierende Kreise.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

Beweisidee:

- Suche verbessernde Pfade und
- Suche kostenverbessernde alternierende Kreise.
- Vorgehen wie bei kostenminimalen Flüssen.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

Beweisidee:

- Suche verbessernde Pfade und
- Suche kostenverbessernde alternierende Kreise.
- Vorgehen wie bei kostenminimalen Flüssen.

Literatur

- Cormen, Leiserson, Rives: Introduction to Algorithms, First Edition, MIT Press, 1990.

Literatur

- Cormen, Leiserson, Rives: Introduction to Algorithms, First Edition, MIT Press, 1990.
- Cormen, Leiserson, Rives: Introduction to Algorithms, Second Edition, MIT Press, 2001.

Literatur

- Cormen, Leiserson, Rives: Introduction to Algorithms, First Edition, MIT Press, 1990.
- Cormen, Leiserson, Rives: Introduction to Algorithms, Second Edition, MIT Press, 2001.
- Ottmann, Widmayer: Algorithmen und Datenstrukturen. BI-Wiss.-Verl. 1990.

Literatur

- Cormen, Leiserson, Rives: Introduction to Algorithms, First Edition, MIT Press, 1990.
- Cormen, Leiserson, Rives: Introduction to Algorithms, Second Edition, MIT Press, 2001.
- Ottmann, Widmayer: Algorithmen und Datenstrukturen. BI-Wiss.-Verl. 1990.