Algorithms and Probability

Week 3

2025/03/06 — Georg Hasebe

Traveling salesman problem (TSP)

Given a weighted, complete graph K_n, we want to determine the Hamiltonian cycle of **smallest total weight** in K_n.

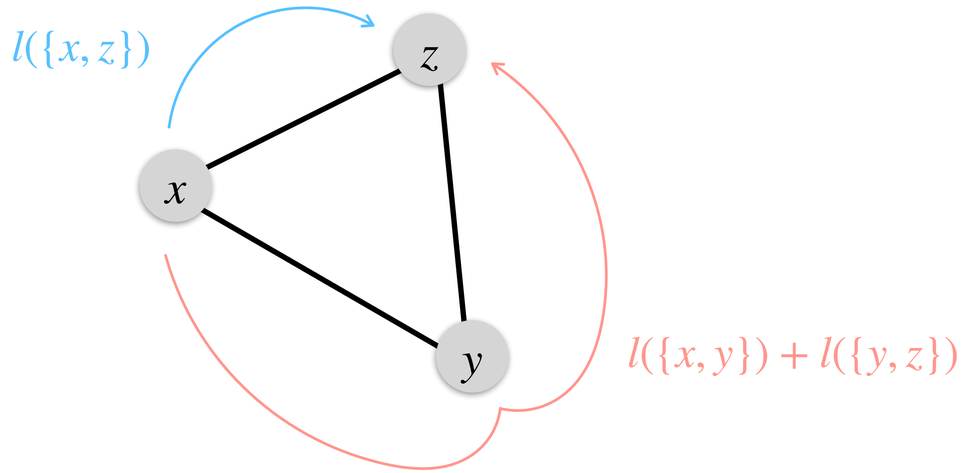
More formally, if l is the edge weight function of K_n, we look for a Hamiltonian cycle C such that

$$\sum_{e \in C} l(e) = min\{ \sum_{e \in C'} \mid C' \text{ is a Hamiltonian cycle in } K_n \}.$$

Metric TSP

Same as TSP, except the edge weight function l has to have the following property (triangle inequality)

$$l(\{x,z\}) \leq l(\{x,y\}) + l(\{y,z\}) \quad \text{for all } x,y,z \in [n].$$



traveling the direct route $\{x,z\}$ is cheaper than $\{x,y\} \to \{y,z\}$.

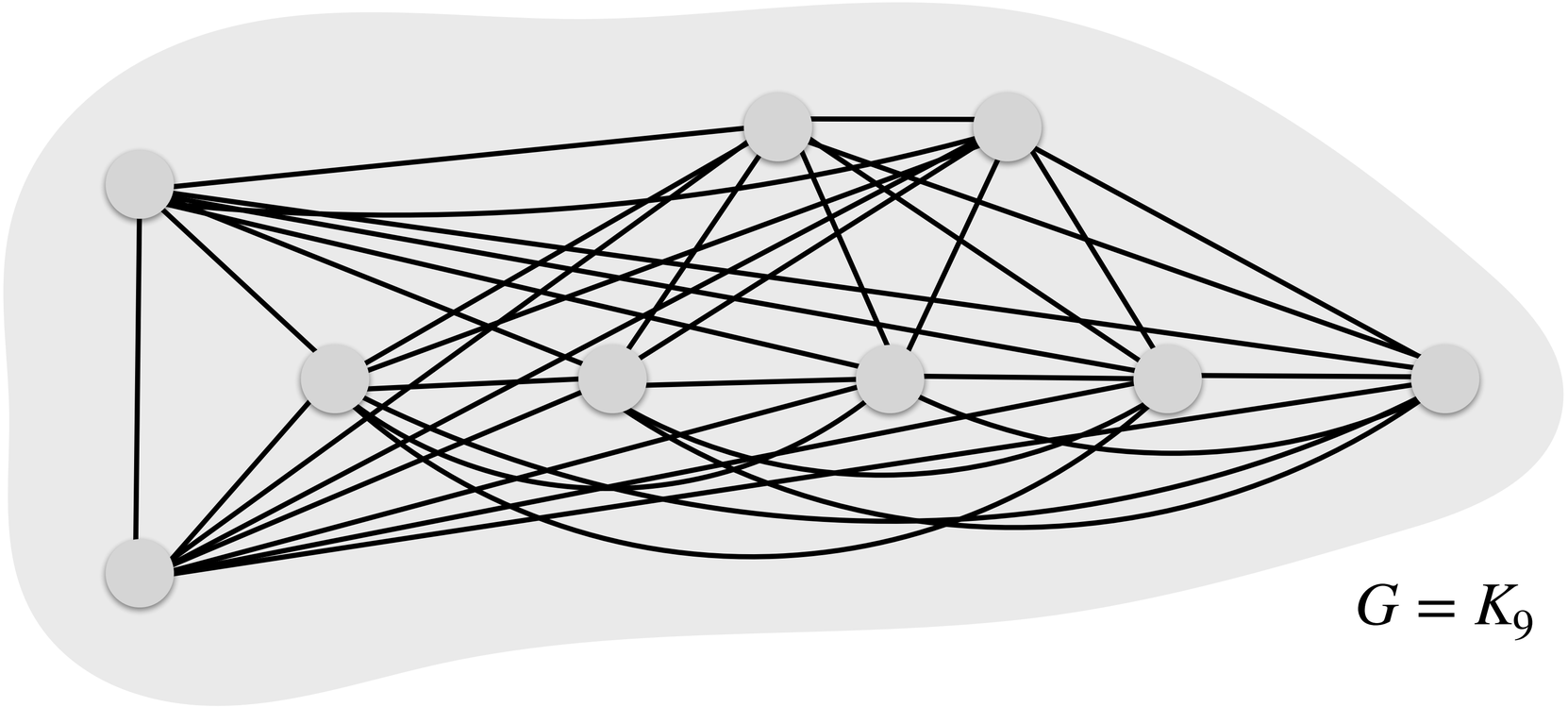
2-approximation-algorithm for metric TSP

Input: complete graph G, metric edge weight function l.

1. Determine MST T in G.
2. Double all edges in T, call it T' (a multigraph).
3. Find Eulerian circuit E in T'.
4. Shorten E (we will see what this means).

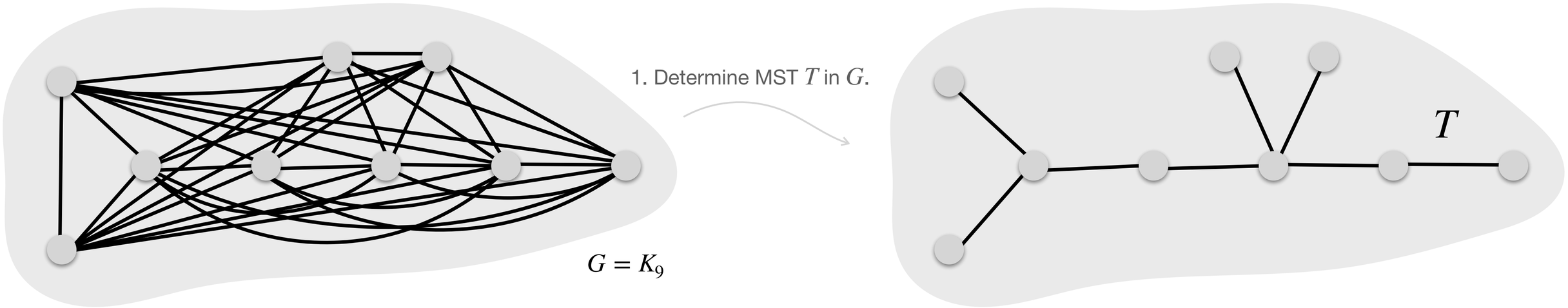
2-approximation-algorithm for metric TSP

Input: complete graph G, metric edge weight function l.



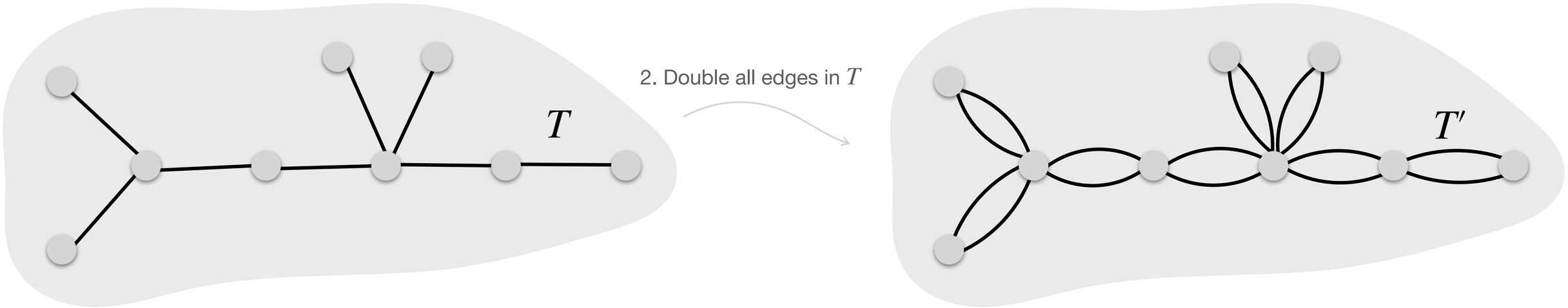
2-approximation-algorithm for metric TSP

Input: complete graph G, metric edge weight function l.



2-approximation-algorithm for metric TSP

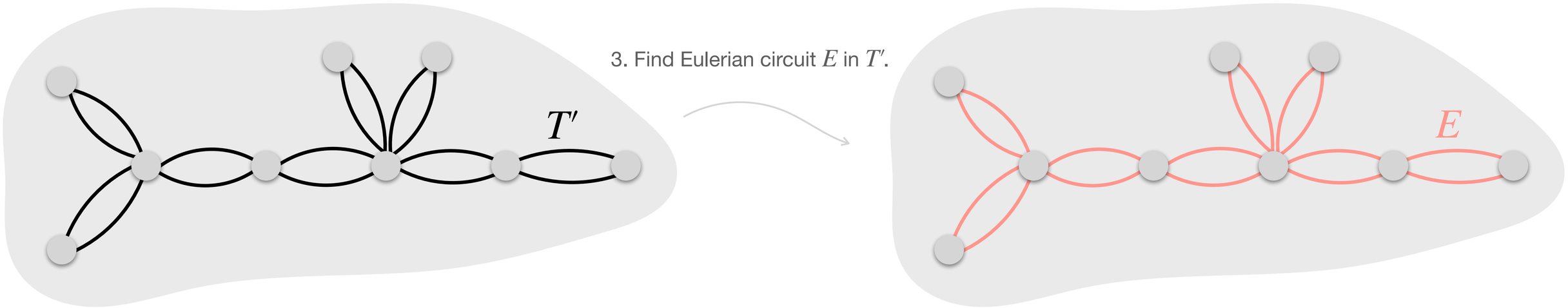
Input: complete graph G, metric edge weight function l.



(*) Note that T' is a multigraph.

2-approximation-algorithm for metric TSP

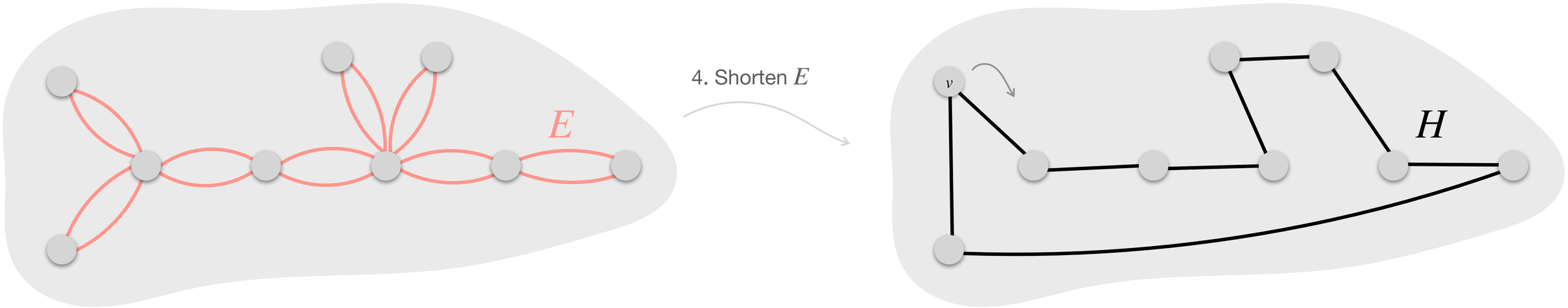
Input: complete graph G, metric edge weight function l.



(*) possible since every degree is even by construction of T'.

2-approximation-algorithm for metric TSP

Input: complete graph G, metric edge weight function l.

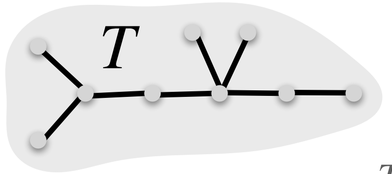


We shorten E by walking along it and skipping vertices we already walked over. In this example we start at v and walk to the right.

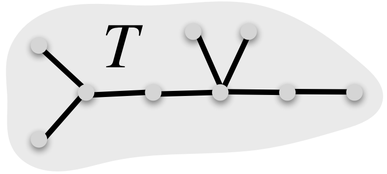
(*) here we use the fact that the edge weight function fulfills the triangle inequality.
(**) remember that G is a complete graph, thus making it possible to walk over any edge we get by shortening.

Analysis

Let OPT be the cheapest Hamiltonian cycle. Then $\text{OPT} \setminus \{e\}$ is a spanning tree.

cost() $\leq$ cost(OPT$\setminus\{e\}$) $\leq$ cost(OPT)

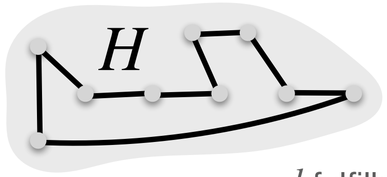
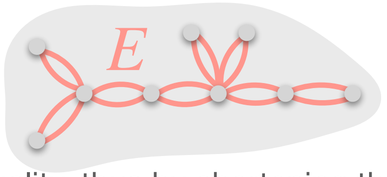
T is a MST

cost() $= 2 \cdot$ cost() $\leq 2 \cdot$ cost(OPT)

construction of T'

cost() = cost() $\leq 2 \cdot$ cost(OPT)

Eulerian circuit E goes over all edges in T'.

cost() $\leq$ cost() $\leq 2 \cdot$ cost(OPT)

l fulfills the triangle inequality, thus by shortening the cost can only get shorter.

Runtime

1. Determine MST T in $G \implies O(n^2)$

   See theorem 1.19. Remember that G is complete, thus $m = O(n^2)$.

2. Double all edges in T, call it T' (a multigraph) $\implies O(m)$

3. Find Eulerian circuit E in $T' \implies O(n)$

   Theorem 1.31. (b) we can find a Eulerian circuit in $O(m)$. Since T' has $2(n-1)$ edges we have $O(n)$.

4. Shorten E (we will see what this means) $\implies O(n)$

Result

Satz 1.43. Für das METRISCHE TRAVELLING SALESMAN PROBLEM gibt es einen 2-Approximationsalgorithmus mit Laufzeit $O(n^2)$.

Matchings

Eine Kantenmenge M ⊆ E heisst **Matching** in einem Graphen G = (V, E), falls kein Knoten des Graphen zu mehr als einer Kante aus M inzident ist.

Each vertex appears in at most one edge, i.e. $e \cap f = \emptyset$ for all $e, f \in M, e \neq f$

Ein Knoten v wird von M **überdeckt**, falls es eine Kante e ∈ M gibt, die v enthält.

Ein Matching M heisst **perfektes Matching**, wenn jeder Knoten durch genau eine Kante aus M überdeckt wird, oder, anders ausgedrückt, wenn |M| = |V|/2.

Matchings

english: maximal matching

Ein Matching $M \subseteq E$ ist ein **inklusionsmaximales Matching**, wenn es kein Matching M' gibt mit $M \subseteq M'$ und $|M'| > |M|$.

english: maximum matching

Ein Matching $M \subseteq E$ ist ein **(kardinalitäts-)maximales Matching**, wenn es kein Matching M' gibt mit $|M'| > |M|$.

inklusionsmaximales Matching
(aber nicht kardinalitätsmaximal)

english: maximal matching

kardinalitätsmaximales Matching
(auch inklusionsmaximal!)

english: maximum matching

DiskMath Recap

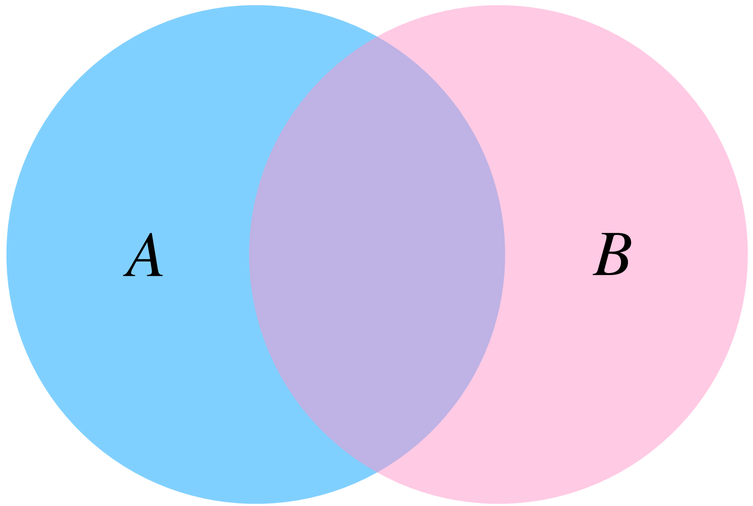
Exclusive or/Exclusive disjunction/Exor $\oplus$:

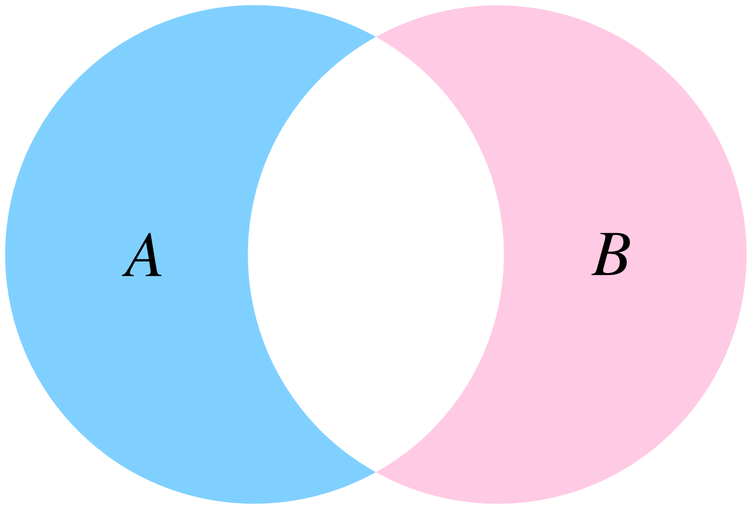
A	B	$A \oplus B$
False	False	False
False	True	True
True	False	True
True	True	False

True whenever A and B differ.

DiskMath Recap

$\bigoplus$ for sets:





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

$x \in A \oplus B$ if in A or B but not both.

Matchings

Satz: Ist M_{inc} ein inklusionsmaximales Matching und M_{max} ein kardinalitäts-maximales Matching, so gilt

$$|M_{inc}| \geq |M_{max}| \ / \ 2.$$

Hint: What can we say about edges in $M_{max} \oplus M_{inc}$?

Proof on my website.

(*) Remark

This proof also holds for two inklusionsmaximale matchings. This means $|M_{\text{max}}| \geq |M_{\text{inc}}|/2$ also holds, since every kardinalitätsmaximale matching is also inklusionsmaximal.

Moreover, this shows that any inklusionsmaximale matching is a 2-approximation of a kardinalitätsmaximale matching and also a 2-approximation of a minimum inklusionsmaximale matching.

(*) Not part of the lecture; source

Greedy Matching

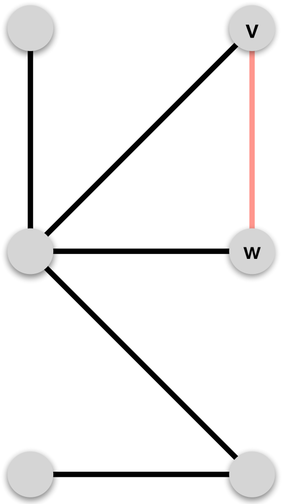
Input: Graph $G = (V, E)$

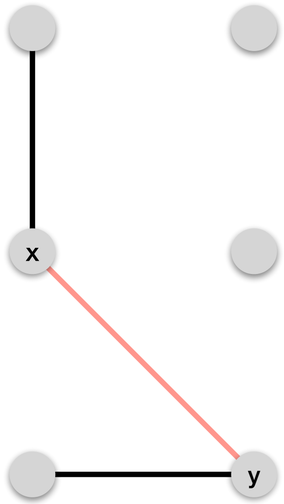
Initialize empty matching M

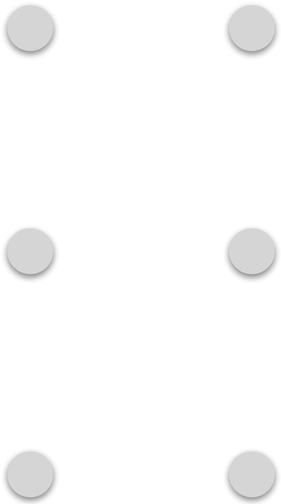
Repeat until E is empty:

1. Pick arbitrary edge e in E
2. Add e to matching M
3. Remove e and all incident edges from G

G

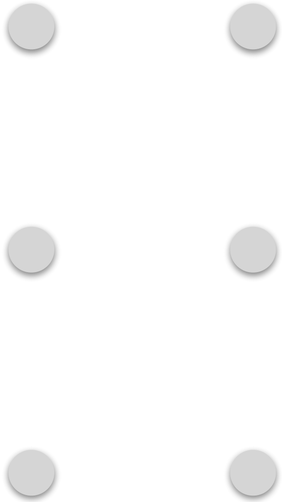


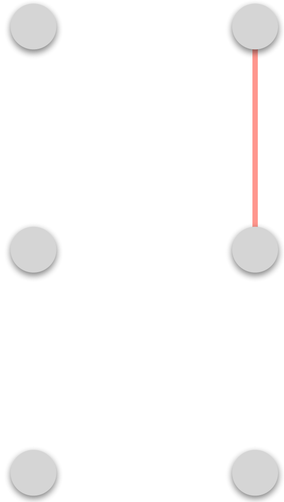


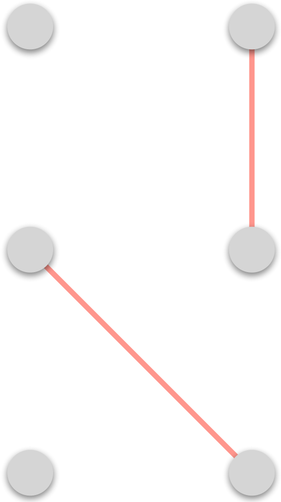


E empty; stop

M







Note that the matching M consists only of edges, i.e. $M \subseteq E$. The vertices were only drawn for better visualization.

Pseudocode

GREEDY-MATCHING (G)

```
1: M ← ∅
2: while E ≠ ∅ do
3:     wähle eine beliebige Kante e ∈ E
4:     M ← M ∪ {e}
5:     lösche e und alle inzidenten Kanten in G
```

Result

Satz 1.47. Der Algorithmus GREEDY-MATCHING bestimmt in Zeit $O(|E|)$ ein inklusionsmaximales Matching M_{Greedy} für das gilt:

$$|M_{Greedy}| \geq \frac{1}{2}|M_{max}|,$$

wobei M_{max} ein kardinalitätsmaximales Matching sei.

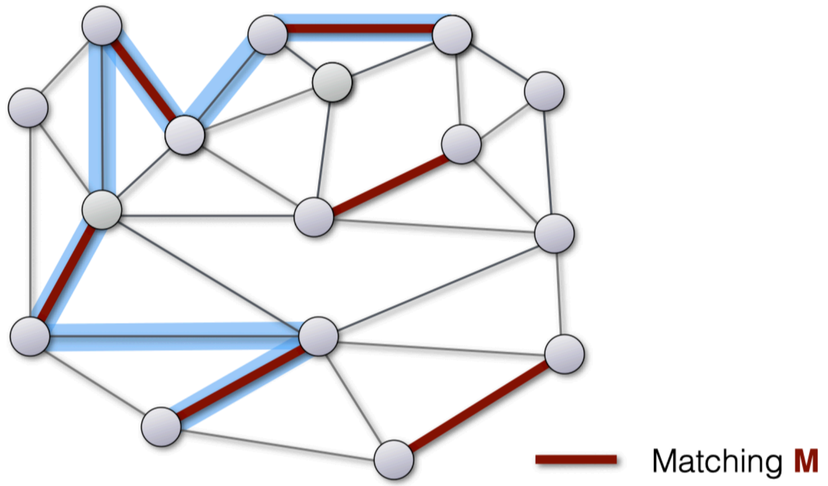
M_{Greedy} is a matching since after picking an edge $\{v, w\}$ we delete all edges incident to v and w. Thus for all $e, f \in M_{\text{Greedy}}$ we have $e \cap f = \emptyset$.

M_{Greedy} is inklusionsmaximal by construction of the algorithm: it only stops after E is empty, meaning there cannot exist a matching M' such that $M_{\text{Greedy}} \subseteq M'$ and $|M'| > M_{\text{Greedy}}$.

Now we apply the theorem from before: we get $|M_{\text{Greedy}}| \geq \frac{1}{2}|M_{\max}|$.

Augmenting Paths

Ein **M-augmentierender Pfad** ist ein Pfad, der abwechselnd Kanten aus M und nicht aus **M** enthält und der in von **M** nicht überdeckten Knoten beginnt und endet.



Augmentation means “Vergrösserung” in German, so augmenting paths are “Vergrösserungspfade”.
I think I made this up, but maybe it helps remembering.

Is there always an augmenting path?

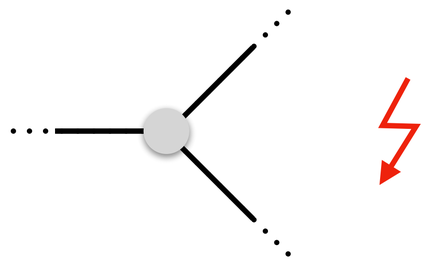
Satz 1.48 (Satz von Berge). Ist M ein Matching in einem Graphen $G = (V, E)$, das nicht kardinalitätsmaximal ist, so existiert ein augmentierender Pfad zu M.

Satz 1.48 (Satz von Berge). Ist M ein Matching in einem Graphen $G = (V, E)$, das nicht kardinalitätsmaximal ist, so existiert ein augmentierender Pfad zu M.

Proof

Let M_1, M_2 be matchings in $G = (V, E)$ such that $|M_1| < |M_2|$. Then M_1 can't be kardinalitätsmaximal.

We inspect the graph $G_M = (V, M_1 \oplus M_2)$. Each vertex in G_M has degree at most 2.



This situation could never occur as M_1 and M_2 are matchings.

Satz 1.48 (Satz von Berge). Ist M ein Matching in einem Graphen G = (V, E), das nicht kardinalitätsmaximal ist, so existiert ein augmentierender Pfad zu M.

Proof

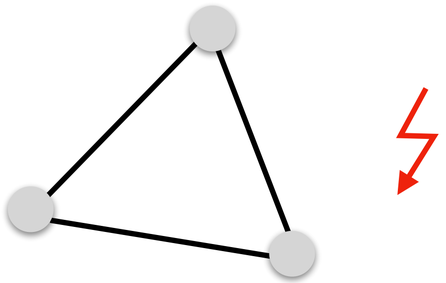
M_1, M_2 matchings in $G = (V, E)$, $|M_1| < |M_2|$. (M_1 not kardinalitätsmaximal)

$G_M = (V, M_1 \oplus M_2)$. Each vertex in G_M has degree at most 2.

The connected components of G_M are paths or cycles.

Follows from the degree being at most 2.

All the cycles in G_M are of even length.



This situation could never occur as M_1 and M_2 are matchings.

Satz 1.48 (Satz von Berge). Ist M ein Matching in einem Graphen G = (V, E), das nicht kardinalitätsmaximal ist, so existiert ein augmentierender Pfad zu M.

Proof

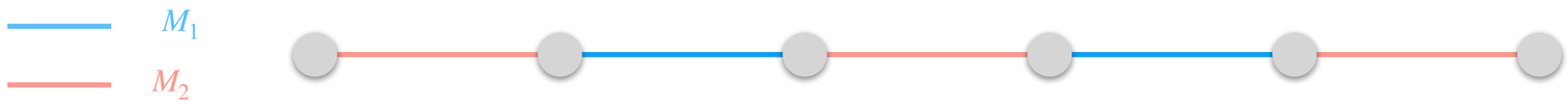
M_1, M_2 matchings in $G = (V, E), |M_1| < |M_2|$. ($M_1$ not kardinalitätsmaximal)

$G_M = (V, M_1 \oplus M_2)$. Each vertex in G_M has degree at most 2.

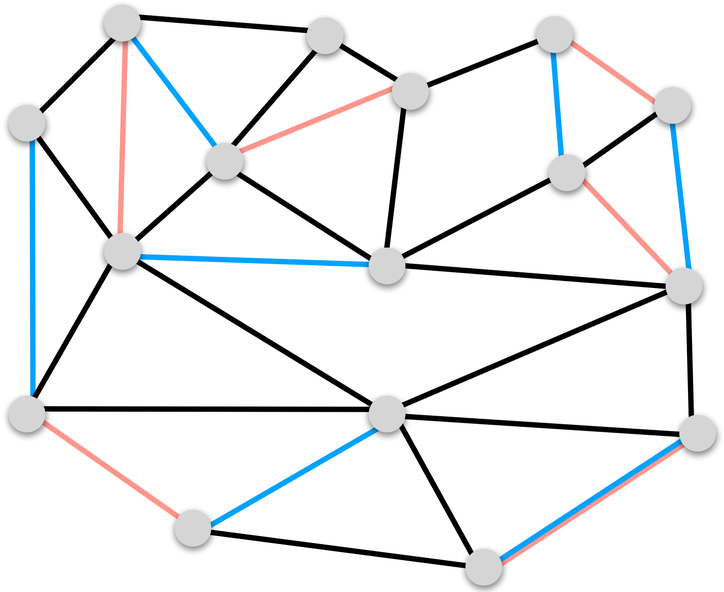
The connected components of of G_M are paths or cycles.

All the cycles in G_M are of even length. All paths and cycles of even length contain the same amount of edges from M_1 and M_2.

Since $|M_1| < |M_2|$, there exists a path P of odd length that contains more edges from M_2. P is an augmenting path.

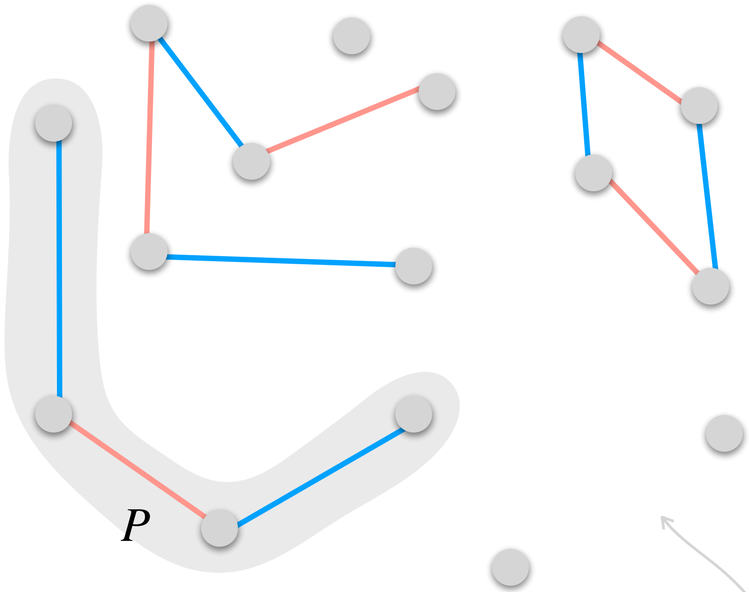


We can use this path to make M_1 bigger.

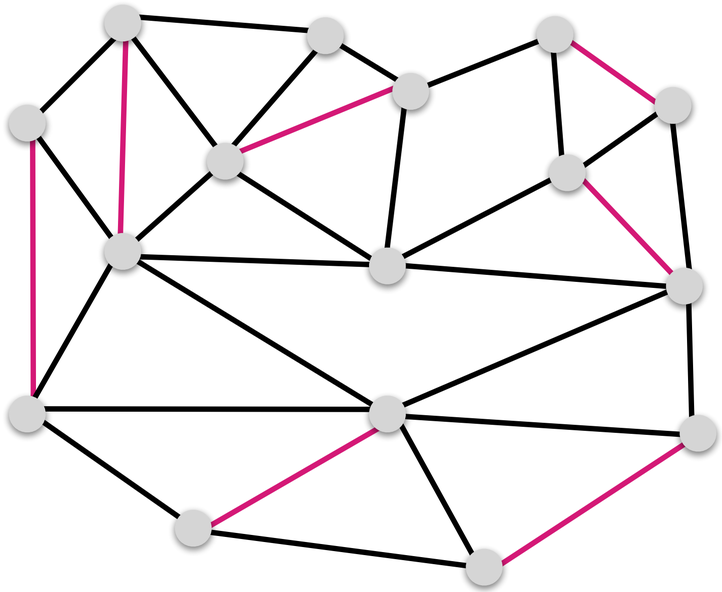


$G = (V, E)$

$G_M = (V, M_1 \oplus M_2)$



This edge was in both M_1 and M_2, meaning it can't be in $M_1 \oplus M_2$.



$M' = M_2 \oplus P$

Using the augmenting path P we increased the size of M_1.

M_1

M_2

Idea

Using the concept of an augmenting path, we can increase the number of edges in a matching until it is kardinalitätsmaximal.

Hopcroft-Karp algorithm

This algorithm relies on the concept of an augmenting path.

By repeatedly finding augmenting paths we can increase the number of edges in our matching until it is kardinalitätsmaximal.

But how do we find augmenting paths?

BFS for finding augmenting paths

Input: bipartite graph $G = (A \uplus B, E)$, matching M

We use BFS to go from A to B and back, while alternating between edges in M and $E \backslash M$.

Pseudocode

MAXIMAL_MATCHING $(G = (A \oplus B, E))$ (Hopcroft und Karp)

1: $M := \{e\}$ für irgendeine Kante $e \in E$.
2: **while** es gibt noch augmentierende Pfade **do**
3: $k :=$ Länge eines kürzesten augmentierenden Pfades
4: Finde eine inklusionsmaximale Menge S von paarweise disjunkten augmentierenden Pfaden der Länge k.
5: **for all** P aus S **do**
6: $M := M \oplus P$. // *augmentiere entlang der Pfade aus S*
7: **return** M

Pseudocode

AUGMENTING_PATH $(G = (A \uplus B, E), M)$

1: $L_0 := \{\text{unüberdeckte Knoten in } A\}$
2: Markiere alle Knoten aus L_0 als besucht.
3: **if** $L_0 = \emptyset$ **then**
4: **return** M ist maximal
5: **for all** $i = 1$ **to** n **do**
6: **if** i ungerade **then**
7: $L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } E \setminus M\}$
8: **else**
9: $L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } M\}$
10: Markiere alle Knoten aus L_i als besucht.
11: **if** L_i enthält unüberdeckten Knoten v **then**
12: Finde Pfad P von L_0 nach v durch backtracking
13: **return** P // *terminiert Algorithmus*
14: **return** M ist schon maximal

$M := \{e\}$ für irgendeine Kante $e \in E$.

if i ungerade **then**

$L_i := \{$unbesuchte Nachbarn von L_{i-1} via Kanten in $E \setminus M\}$

else

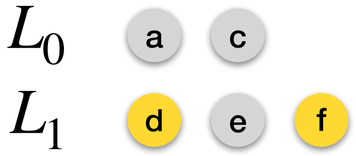
$L_i := \{$unbesuchte Nachbarn von L_{i-1} via Kanten in $M\}$

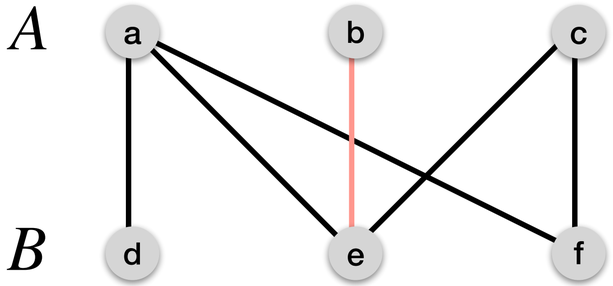
$L_0 := \{$unüberdeckte Knoten in $A\}$

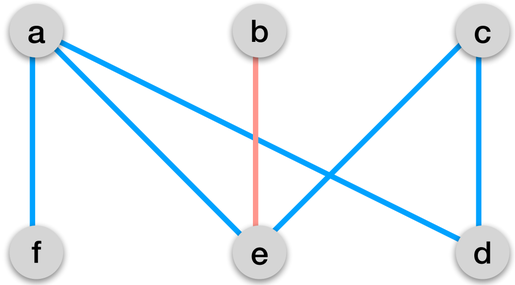
if L_i enthält unüberdeckten Knoten v **then**

Finde Pfad P von L_0 nach v durch backtracking

return P // *terminiert Algorithmus*







Is augmenting path P, return P.

$M := \{e\}$ für irgendeine Kante $e \in E$.

if i ungerade **then**

$L_i := \{$unbesuchte Nachbarn von L_{i-1} via Kanten in $E \setminus M\}$

else

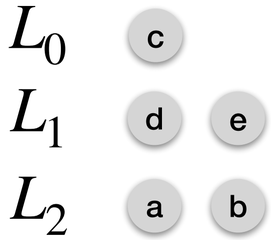
$L_i := \{$unbesuchte Nachbarn von L_{i-1} via Kanten in $M\}$

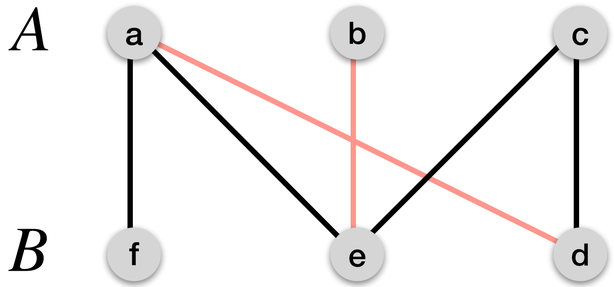
$L_0 := \{$unüberdeckte Knoten in $A\}$

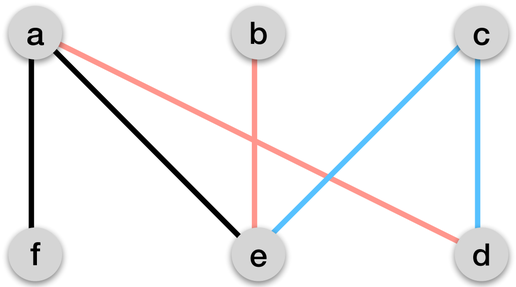
if L_i enthält unüberdeckten Knoten v **then**

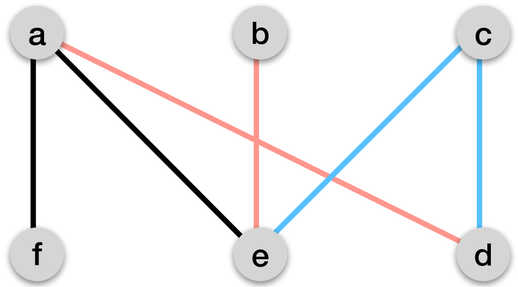
Finde Pfad P von L_0 nach v durch backtracking

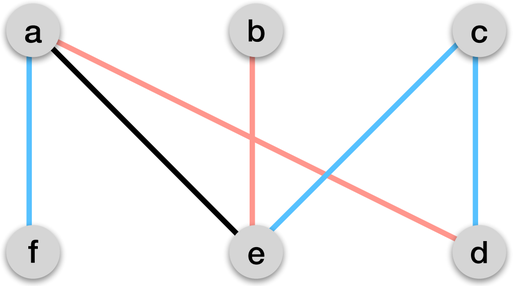
return P // *terminiert Algorithmus*











Is augmenting path P, return P.

$M := \{e\}$ für irgendeine Kante $e \in E$.

$L_0 := \{\text{unüberdeckte Knoten in } A\}$

if i ungerade **then**

$L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } E \setminus M\}$

else

$L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } M\}$

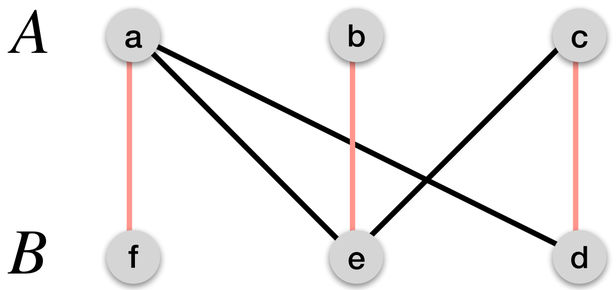
if L_i enthält unüberdeckten Knoten v **then**

Finde Pfad P von L_0 nach v durch backtracking

return P // *terminiert Algorithmus*

while es gibt noch augmentierende Pfade **do**

$L_0 = \varnothing$



No augmenting paths left. By theorem 1.48 this means that M is a maximum matching.

Result

Satz 1.49. Der Algorithmus von Hopcroft und Karp durchläuft die while-Schleife nur $O(\sqrt{|V|})$ Mal. Er berechnet daher ein maximales Matching in einem bipartiten Graphen in Zeit $O(\sqrt{|V|} \cdot (|V| + |E|))$.