Algorithms and Probability

Week 4

G08 - mkilic

13.III.2025

Overview

1. **Minitest**
2. **Augmentierende Pfade**
3. **Hopcroft & Karp**
4. **TSP $\frac{3}{2}$-Approximationsalgorithmus**
5. **Exercises**

Roadmap

1. Graphentheorie
   - Zusammenhang
   - Kreise
   - Matchings
   - Färbungen
2. W'keitstheorie
   - Bedingte W'keit
   - Unabhängigkeit
   - (mehrere) Zufallsvariablen
   - Diskrete Verteilungen
   - Abschätzen von W'keiten
   - Randomisierte Algorithmen
3. Algorithmen
   - Lange-Bunte Pfade
   - MaxFlow
   - MinCut
   - Kleinster umschliessender Kreis
   - Konvexe Hülle

Minitest

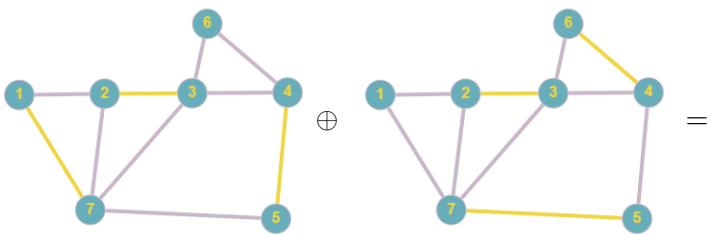
Password: matching

Exor von zwei Matchings

Exklusives Oder

Exklusives Oder von zwei Matchings (oder im Allgemeinen von zwei Kantenmengen) M_1 und M_2 ist:

$$M_1 \oplus M_2 = (M_1 \cup M_2) \setminus (M_1 \cap M_2)$$

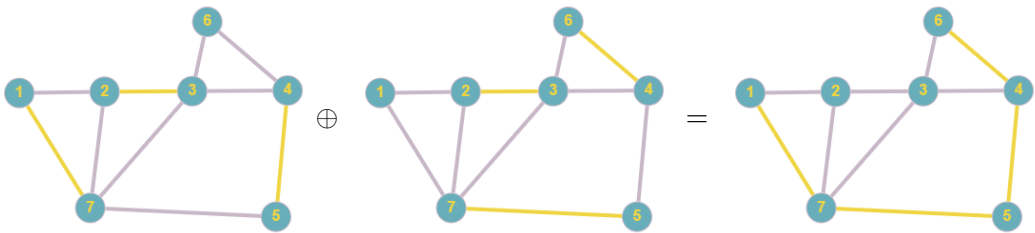


Exor von zwei Matchings

Exklusives Oder

Exklusives Oder von zwei Matchings (oder im Allgemeinen von zwei Kantenmengen) M_1 und M_2 ist:

$$M_1 \oplus M_2 = (M_1 \cup M_2) \setminus (M_1 \cap M_2)$$



Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.

Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.
2. Wegen *1.* können die ZHKe von G alles Pfade und/oder Kreise sein wobei alle Kreise gerade Länge haben müssen (die Kanten sind abwechselnd aus M_1 und M_2 = bipartit Kreis)

Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.
2. Wegen *1.* können die ZHKe von G alles Pfade und/oder Kreise sein wobei alle Kreise gerade Länge haben müssen (die Kanten sind abwechselnd aus M_1 und M_2 = bipartit Kreis)
3. Jede Komponente, die ein Kreis oder ein Pfad mit gerader Länge ist, enthält jeweils gleich viele Kanten aus M_1 und M_2

Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.
2. Wegen *1.* können die ZHKe von G alles Pfade und/oder Kreise sein wobei alle Kreise gerade Länge haben müssen (die Kanten sind abwechselnd aus M_1 und M_2 = bipartit Kreis)
3. Jede Komponente, die ein Kreis oder ein Pfad mit gerader Länge ist, enthält jeweils gleich viele Kanten aus M_1 und M_2

Sei jetzt o.B.d.A. $|M_1| < |M_2|$

Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.
2. Wegen *1.* können die ZHKe von G alles Pfade und/oder Kreise sein wobei alle Kreise gerade Länge haben müssen (die Kanten sind abwechselnd aus M_1 und M_2 = bipartit Kreis)
3. Jede Komponente, die ein Kreis oder ein Pfad mit gerader Länge ist, enthält jeweils gleich viele Kanten aus M_1 und M_2

Sei jetzt o.B.d.A. $|M_1| < |M_2|$

4. Wegen *3.*, muss es ZHKe geben, die Pfade P_i sind, die mehr Kanten aus M_2 als aus M_1 enthalten.

Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.
2. Wegen *1.* können die ZHKe von G alles Pfade und/oder Kreise sein wobei alle Kreise gerade Länge haben müssen (die Kanten sind abwechselnd aus M_1 und M_2 = bipartit Kreis)
3. Jede Komponente, die ein Kreis oder ein Pfad mit gerader Länge ist, enthält jeweils gleich viele Kanten aus M_1 und M_2

Sei jetzt o.B.d.A. $|M_1| < |M_2|$

4. Wegen *3.*, muss es ZHKe geben, die Pfade P_i sind, die mehr Kanten aus M_2 als aus M_1 enthalten.
5. Da sich Kanten aus M_1 und M_2 abwechseln, kann ein solcher Pfad höchstens eine Kante mehr aus M_2 enthalten (Beide äussere Kanten von P gehören zu M_2).

Augmentierende Pfade (Augmenting Paths)

Im Folgenden seien M_1 und M_2 zwei beliebige Matchings in $G = (V, E)$.

1. Jeder Knoten im Graphen $G_M = (V, M_1 \oplus M_2)$ hat Grad höchstens 2.
2. Wegen *1.* können die ZHKe von G alles Pfade und/oder Kreise sein wobei alle Kreise gerade Länge haben müssen (die Kanten sind abwechselnd aus M_1 und M_2 = bipartit Kreis)
3. Jede Komponente, die ein Kreis oder ein Pfad mit gerader Länge ist, enthält jeweils gleich viele Kanten aus M_1 und M_2

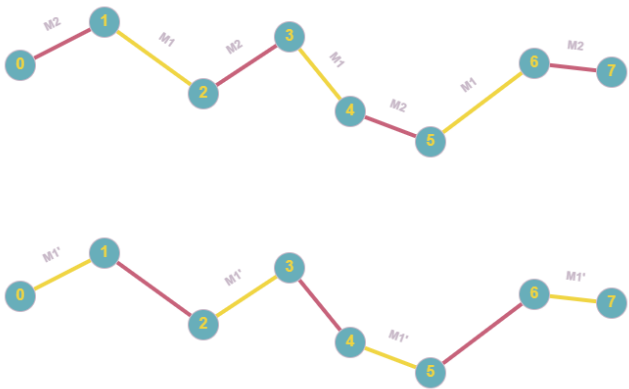
Sei jetzt o.B.d.A. $|M_1| < |M_2|$

4. Wegen *3.*, muss es ZHKe geben, die Pfade P_i sind, die mehr Kanten aus M_2 als aus M_1 enthalten.
5. Da sich Kanten aus M_1 und M_2 abwechseln, kann ein solcher Pfad höchstens eine Kante mehr aus M_2 enthalten (Beide äussere Kanten von P gehören zu M_2).
6. Aus *4.* + *5.* folgt: $|M_2| = |M_1| + k \Rightarrow$ es muss mindestens k solche (wie in *4.*) Pfade geben.

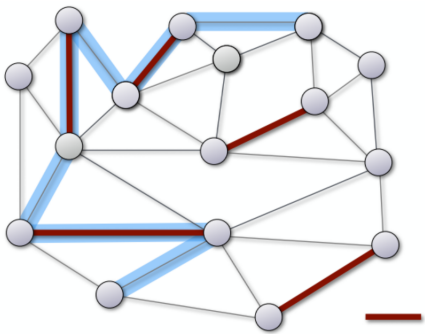
Augmentierende Pfade

Jetzt können wir aus dem Matching M_1 ein neues Matching M_1' erhalten, das eine Kante mehr enthält. Wie?

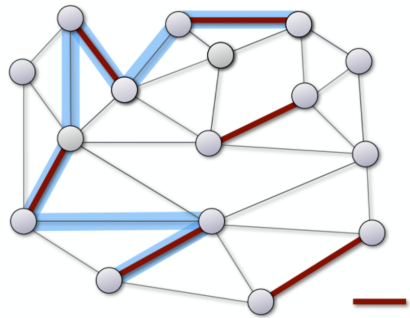
Wir **augmentieren**: Setze $M_1' := M_1 \oplus P$.



Augmentierende Pfade



augmentieren
$\rightarrow$



$M_1' > M_1$ Also haben wir jetzt ein grösseres Matching!

[0] Hier M_1 : rote Kanten, P : blaue Kanten

Augmentierende Pfade

M-augmentierender Pfad

Formal ist ein M-augmentierender Pfad definiert durch: Die beiden Endknoten von P werden durch M nicht überdeckt und P besteht abwechselnd aus Kanten, die nicht zu M gehören, und Kanten aus M.

Augmentierende Pfade

M-augmentierender Pfad

Formal ist ein M-augmentierender Pfad definiert durch: Die beiden Endknoten von P werden durch M nicht überdeckt und P besteht abwechselnd aus Kanten, die nicht zu M gehören, und Kanten aus M.

Satz von Berge

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

Augmentierende Pfade

M-augmentierender Pfad

Formal ist ein M-augmentierender Pfad definiert durch: Die beiden Endknoten von P werden durch M nicht überdeckt und P besteht abwechselnd aus Kanten, die nicht zu M gehören, und Kanten aus M.

Satz von Berge

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

- Eine einzelne Kante, die keine Endpunkte in M hat, ist ein M-augmentierender Pfad der Länge 1.

Aug. Pfade: Algorithmus

Mit diesen Ideen kann man einen ersten Algorithmus konstruieren, der **kardinalitätsmaximale Matching** bestimmt.

```
1        Start with M = ∅
2        REPEAT:
3            Search for augmenting path P
4            if no such path exists then return M
5            else M = M ⊕ P
6
```

Aug. Pfade: Algorithmus

Mit diesen Ideen kann man einen ersten Algorithmus konstruieren, der **kardinalitätsmaximale Matching** bestimmt.

```
1        Start with M = ∅
2        REPEAT:
3            Search for augmenting path P
4            if no such path exists then return M
5            else M = M ⊕ P
6
```

Spätestens nach $\frac{|V|}{2} - 1$ *solcher Schritten ist das Matching dann maximal, denn ein Matching kann nicht mehr als* $\frac{|V|}{2}$ *viele Kanten enthalten.*

Aug. Pfade: Algorithmus

Mit diesen Ideen kann man einen ersten Algorithmus konstruieren, der **kardinalitätsmaximale Matching** bestimmt.

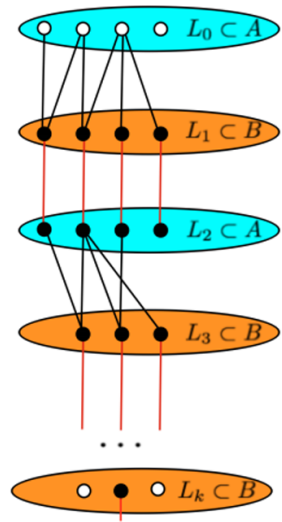
```
1        Start with M = ∅
2        REPEAT:
3            Search for augmenting path P
4            if no such path exists then return M
5            else M = M ⊕ P
6
```

> *Spätestens nach* $\frac{|V|}{2} - 1$ *solcher Schritten ist das Matching dann maximal, denn ein Matching kann nicht mehr als* $\frac{|V|}{2}$ *viele Kanten enthalten.*

Wie suchen wir augmentierende Pfade effizient? BFS! (in bipartiten Graphen)

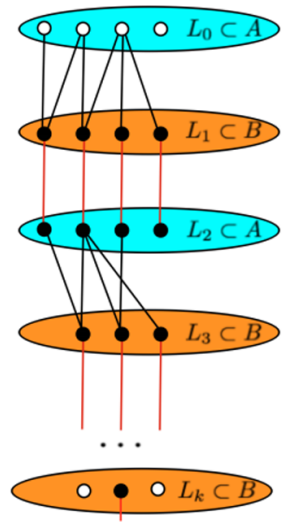
Aug. Pfade: Algorithmus

1. Ein augmentierender Pfad muss immer ungerade Länge haben, denn er hat eine Kante aus $E \setminus M$ mehr als Kanten in M.



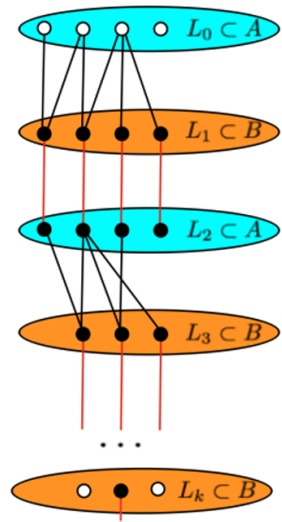
Aug. Pfade: Algorithmus

1. Ein augmentierender Pfad muss immer ungerade Länge haben, denn er hat eine Kante aus $E \setminus M$ mehr als Kanten in M.
2. Deshalb für alle augmentierenden Pfaden gilt: Eine Endpunkt in A eine Endpunkt in B



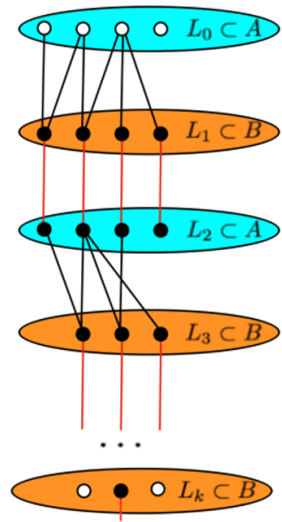
Aug. Pfade: Algorithmus

1. Ein augmentierender Pfad muss immer ungerade Länge haben, denn er hat eine Kante aus $E \setminus M$ mehr als Kanten in M.
2. Deshalb für alle augmentierenden Pfaden gilt: Eine Endpunkt in A eine Endpunkt in B
3. Wegen *(2.)* reicht es mit unüberdeckten Knoten in A anzufangen. Alle augmentierende Pfade haben einen Endpunkt in der Menge von unüberdeckten Knoten in A. Keine wird übersehen.



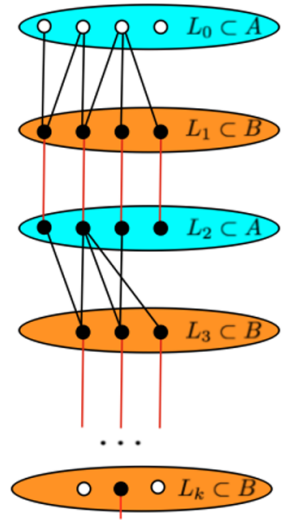
Aug. Pfade: Algorithmus

4. Da wir mit unüberdeckten Knoten anfangen benutzen wir Kanten aus $E \setminus M$ um von L_0 zu L_1 zu übergehen.



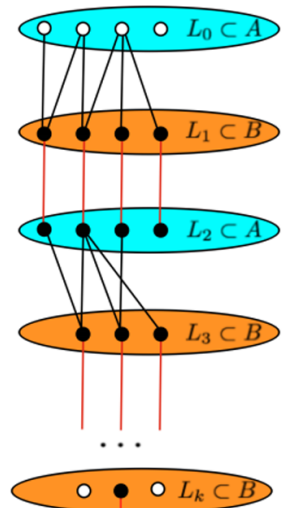
Aug. Pfade: Algorithmus

4. Da wir mit unüberdeckten Knoten anfangen benutzen wir Kanten aus $E \setminus M$ um von L_0 zu L_1 zu übergehen.
5. Weil wir beim Schritt *(4.)* $E \setminus M$ Kanten benutzt haben, landen wir auf ein Knoten aus B, von denen wir mit einer Matching Kante weitergehen können.



Aug. Pfade: Algorithmus

4. Da wir mit unüberdeckten Knoten anfangen benutzen wir Kanten aus $E \setminus M$ um von L_0 zu L_1 zu übergehen.
5. Weil wir beim Schritt *(4.)* $E \setminus M$ Kanten benutzt haben, landen wir auf ein Knoten aus B, von denen wir mit einer Matching Kante weitergehen können.
6. Dann benutzen wir für jeden Knoten in L_1 die zu einem Knoten in der Menge von unüberdeckten Knoten in A benachbart sind, die für jeden Knoten eindeutig bestimmten Matching Kanten aus M die uns von überdeckten Knoten in B zu überdeckten Knoten in A führen.



Aug. Pfade: Algorithmus

- Knoten aus B erreichen wir daher grundsätzlich über Kanten aus $E \setminus M$, und Knoten aus A über die (eindeutig bestimmte) inzidente Kante aus M.

Aug. Pfade: Algorithmus

- Knoten aus B erreichen wir daher grundsätzlich über Kanten aus $E \setminus M$, und Knoten aus A über die (eindeutig bestimmte) inzidente Kante aus M.
- Insbesondere liegen alle unüberdeckten Knoten, die der Algorithmus besucht, im ersten und letzten Layer.

Aug. Pfade: Algorithmus

- Knoten aus B erreichen wir daher grundsätzlich über Kanten aus $E \setminus M$, und Knoten aus A über die (eindeutig bestimmte) inzidente Kante aus M.
- Insbesondere liegen alle unüberdeckten Knoten, die der Algorithmus besucht, im ersten und letzten Layer.
- Wir stoppen falls L_i unüberdeckte Knoten enthält. Weil das würde heissen, dass wir einen unüberdeckten Knoten aus B entdeckt haben, da wir alle unüberdeckten Knoten aus A in L_0 hinzugefügt haben.

Pseudocode

```
AUGMENTING_PATH (G = (A ⊎ B, E), M)
 1: L_0 := {unüberdeckte Knoten in A}
 2: Markiere alle Knoten aus L_0 als besucht.
 3: if L_0 = ∅ then
 4:     return M ist maximal
 5: for all i = 1 to n do
 6:     if i ungerade then
 7:         L_i := {unbesuchte Nachbarn von L_{i-1} via Kanten in E \ M}
 8:     else
 9:         L_i := {unbesuchte Nachbarn von L_{i-1} 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
```

Laufzeit

Laufzeitanalyse

Mit der modifizierten Breitensuche kann man einen augmentierenden Pfad in Zeit $O(|E|)$ finden. (falls er existiert)

Wir müssen höchstens $|V|/2$ oft augmentieren, denn es gilt immer $|M_{max}| < |V|/2$.

Insgesamt erhalten wir einen Algorithmus mit Laufzeit $O(|V||E|)$.

Laufzeit

Laufzeitanalyse

Mit der modifizierten Breitensuche kann man einen augmentierenden Pfad in Zeit $O(|E|)$ finden. (falls er existiert)

Wir müssen höchstens $|V|/2$ oft augmentieren, denn es gilt immer $|M_{max}| < |V|/2$.

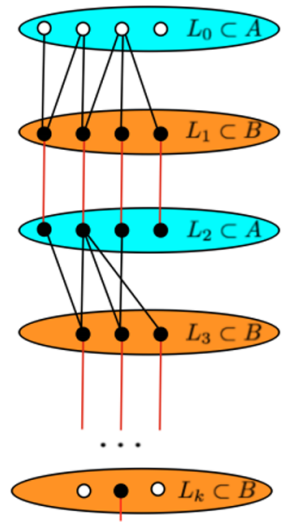
Insgesamt erhalten wir einen Algorithmus mit Laufzeit $O(|V||E|)$.

Hopcroft und Karp hatten eine bessere Idee.

Hopcroft-Karp: Algorithmus

Idee ist sehr ähnlich, nur mit einem kleinen Unterschied. Wir augmentieren entlang eine inklusionsmaximale Menge kürzester augmentierenden Pfaden.

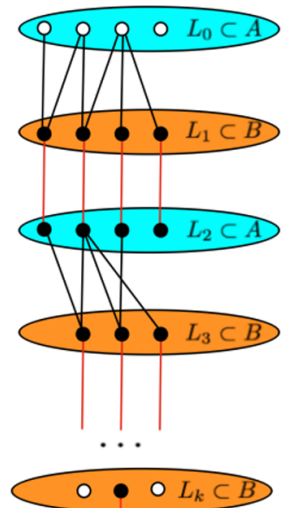
1. Nehme an, dass es auf Layer k mehrere unüberdeckte Knoten liegen.



Hopcroft-Karp: Algorithmus

Idee ist sehr ähnlich, nur mit einem kleinen Unterschied. Wir augmentieren entlang eine inklusionsmaximale Menge kürzester augmentierenden Pfaden.

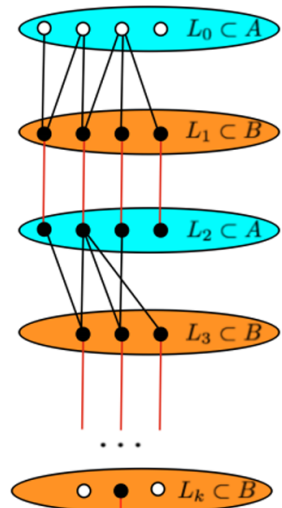
1. Nehme an, dass es auf Layer k mehrere unüberdeckte Knoten liegen.
2. Wähle einen solchen Knoten v_1 aus L_k aus, finde augmentierenden Pfad P_1 von L_0 zu v_1 und **lösche** P_1 aus der Layerstruktur.



Hopcroft-Karp: Algorithmus

Idee ist sehr ähnlich, nur mit einem kleinen Unterschied. Wir augmentieren entlang eine inklusionsmaximale Menge kürzester augmentierenden Pfaden.

1. Nehme an, dass es auf Layer k mehrere unüberdeckte Knoten liegen.
2. Wähle einen solchen Knoten v_1 aus L_k aus, finde augmentierenden Pfad P_1 von L_0 zu v_1 und **lösche** P_1 aus der Layerstruktur.
3. Wähle weiteren unüberdeckten Knoten v_2 aus L_k, und schaue falls es noch ein aug. Pfad P_2 gibt zu diesem Knoten.

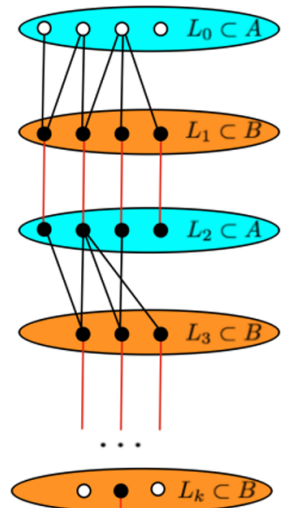


Hopcroft-Karp: Algorithmus

Idee ist sehr ähnlich, nur mit einem kleinen Unterschied. Wir augmentieren entlang eine inklusionsmaximale Menge kürzester augmentierenden Pfaden.

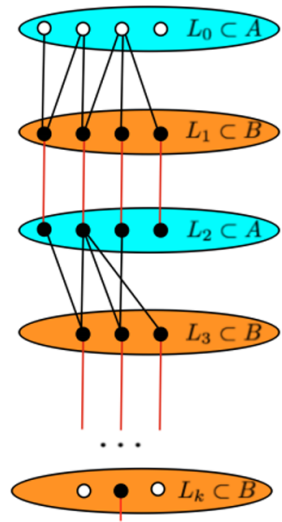
1. Nehme an, dass es auf Layer k mehrere unüberdeckte Knoten liegen.
2. Wähle einen solchen Knoten v_1 aus L_k aus, finde augmentierenden Pfad P_1 von L_0 zu v_1 und **lösche** P_1 aus der Layerstruktur.
3. Wähle weiteren unüberdeckten Knoten v_2 aus L_k, und schaue falls es noch ein aug. Pfad P_2 gibt zu diesem Knoten.

Wie? DFS!



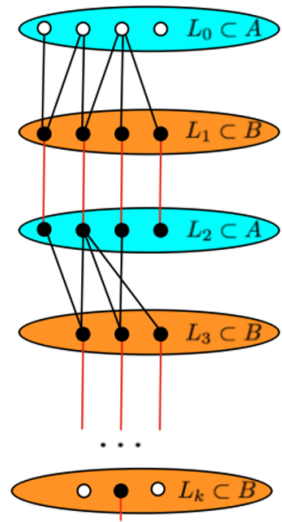
Hopcroft-Karp: Algorithmus

3. Wähle weiteren unüberdeckten Knoten v_2 aus L_k, und schaue falls es noch ein aug. Pfad P_2 gibt zu diesem Knoten mit DFS.



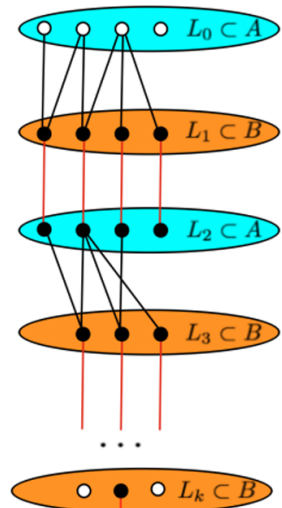
Hopcroft-Karp: Algorithmus

3. Wähle weiteren unüberdeckten Knoten v_2 aus L_k, und schaue falls es noch ein aug. Pfad P_2 gibt zu diesem Knoten mit DFS.
4. Falls DFS erfolgreich zu L_0 erreicht, heisst das, dass wir noch einen augmentierenden Pfad P_2 gefunden haben, der disjunkt zu P_1 ist.



Hopcroft-Karp: Algorithmus

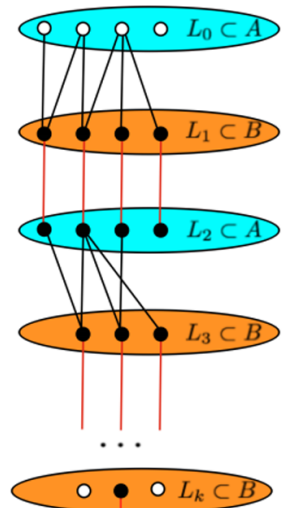
3. Wähle weiteren unüberdeckten Knoten v_2 aus L_k, und schaue falls es noch ein aug. Pfad P_2 gibt zu diesem Knoten mit DFS.
4. Falls DFS erfolgreich zu L_0 erreicht, heisst das, dass wir noch einen augmentierenden Pfad P_2 gefunden haben, der disjunkt zu P_1 ist.
5. Dann lösche alle Knoten auf P_2 und wiederhole.



Hopcroft-Karp: Algorithmus

3. Wähle weiteren unüberdeckten Knoten v_2 aus L_k, und schaue falls es noch ein aug. Pfad P_2 gibt zu diesem Knoten mit DFS.
4. Falls DFS erfolgreich zu L_0 erreicht, heisst das, dass wir noch einen augmentierenden Pfad P_2 gefunden haben, der disjunkt zu P_1 ist.
5. Dann lösche alle Knoten auf P_2 und wiederhole.

Das machen wir für jeden unüberdeckten Knoten auf Layer L_k. Da wir nach jedem Besuch jede Kante und jeden Knoten löschen, ist die Laufzeit dieses Prozesses für ein Layer gleich $O(|V| + |E|)$



Hopcroft-Karp: Algorithmus

Sei $S :=$ die Menge aus augmentierenden Pfaden bei einem Schritt von Hopcroft-Karp Algo. Zusammenfassend gelten:

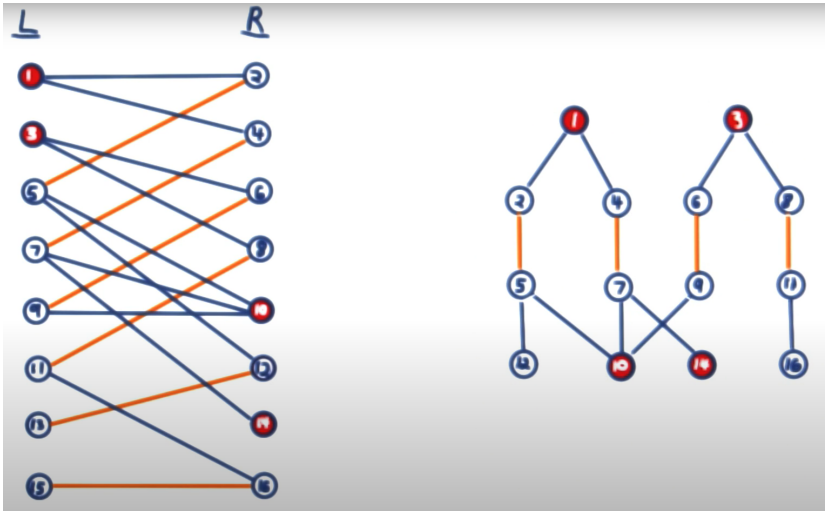
1. Alle Pfade in S haben dieselbe **minimale Länge** k (k ist also die Länge eines kürzesten augmentierenden Pfades)
2. Alle Pfade in S sind **paarweise disjunkt**.
3. S ist **inklusionsmaximal** mit dieser Eigenschaft, d.h. es lässt sich kein weiterer augmentierender Pfad der Länge k zu S hinzufügen, ohne die zweite Bedingung zu verletzen.

Hopcroft-Karp: 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

Hopcroft-Karp: Beispiel Ausführung



Link to YouTube video: https://www.youtube.com/watch?v=n7r4Dp6cVg8

Hopcroft-Karp: Laufzeit

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 $O(\sqrt{|V|} \cdot (|V| + |E|))$.

Hopcroft-Karp: Laufzeit

Diese Laufzeit folgt aus drei Lemmas.

Lemma 1

Ist M ein Matching, P ein kürzester augmentierender Pfad von M, und P' ein augmentierender Pfad für $M \oplus P$, so gilt:

$$|P'| \geq |P| + 2|P \cap P'|$$

Intuition: Augmentieren wir M sukzessive mit *kürzesten* augmentierenden Pfaden, so können die Längen dieser Pfade nicht abnehmen.
Beweis: Auf der Tafel

Hopcroft-Karp: Laufzeit

Lemma 2

Mit jedem Durchlaufen der while-Schleife erhöht sich k um mindestens 2.

Fallunterscheidung: Sei P ein augmentierender Pfad bei der i-ten Iteration und P' ein augmentierender Pfad bei der $i+1$-ten Iteration.

1. $|P \cap P'| = 0$
2. $|P \cap P'| \geq 1$

Beweis: Auf der Tafel

[0] Zur Erinnerung: $k :=$ Länge des kürzesten aug. Pfades

Hopcroft-Karp: Laufzeit

Lemma 3

Sei M ein Matching, für das der kürzeste augmentierende Pfad Länge k hat, und M' ein beliebiges anderes Matching. Dann gilt

$$|M'| \leq |M| + \frac{|V|}{k+1}$$

Beweis:

Es gibt $|M'| - |M|$ disjunkte augmentierende Pfade für M

Auf jedem solchen Pfad liegen mindestens $k+1$Knoten, da mind. aug. Pfad Länge $= k$

Insgesamt ergibt das $(|M'| - |M|) \cdot (k+1)$ viele Knoten

In dem Graphen gibt es $|V|$ Knoten, also gilt$(|M'| - |M|) \cdot (k+1) \leq |V|$

Aussage folgt durch Umformung. $\square$

Hopcroft-Karp: Laufzeit

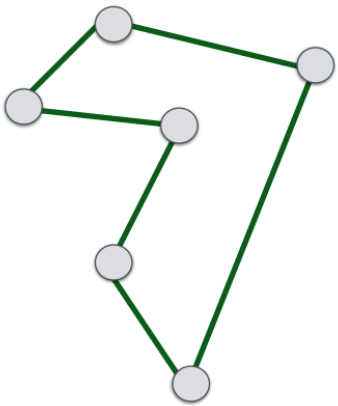
Jetzt benutzen wir diese Lemmas um auf die Laufzeit von Hopcroft-Karp zu kommen.

- *2)* besagt, dass nach den ersten $\lceil \sqrt{V}/2 \rceil$ Durchläufen der while-Schleife $k \geq \sqrt{|V|}$ gilt.
- Jetzt besagt *3)*, dass ein maximales Matching M_{max} erfüllt:

$$|M_{max}| \leq |M| + \frac{|V|}{k+1} \leq |M| + \sqrt{|V|} \text{ also } |M| \geq |M_{max}| - \sqrt{|V|}$$

- Mit jedem weiteren Durchlauf der while-Schleife erhöht sich aber die Grösse von M um mindestens 1. Nach spätestens $\sqrt{|V|}$ weiteren Durchläufen erreicht $|M|$ also die Grösse $|M_{max}|$ und der Algorithmus terminiert.
- Das macht $2|\sqrt{V}$ Durchläufe. Jeder Durchlauf geht in $O(|V| + |E|)$ (DFS). Insgesamt $O(\sqrt{|V|}(|V| + |E|))$ $\square$

TSP 2-Approximationsalgorithmus: Erinnerung



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**) *es gilt*

$$\ell(C) \leq \ell(W) \underset{\Delta ungl.}{=} 2\ell(T) \leq 2opt(K_n, \ell)$$

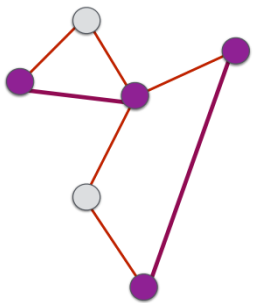
Minimales perfektes Mathcing

Satz 1.50

Ist n gerade und $\ell : \binom{[n]}{2} \to \mathbb{N}_0$ eine Gewichtsfunktion des vollständigen Graphen K_n, so kann man in Zeit $O(n^3)$ ein minimales perfektes Matching finden, also ein perfektes Matching M mit

$$\sum_{e \in M} \ell(e) = \min \left\{ \sum_{e \in M'} \ell(e) \mid M' \text{ perfektes Matching in } K_n \right\}.$$

TSP $\frac{3}{2}$-Approximationsalgorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. Sei $S :=$ Knoten mit ungeradem Grad in T $|S|$ ist gerade (Handschlaglemma) Also gibt es in K_n ein perfektes Matching für Knoten in S. Bestimme **minimales perfektes Matching** M für S: $\ell(M) \leq \frac{1}{2} opt(K_n, \ell)$
3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = \ell(M) + \ell(T) \leq \frac{3}{2} opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**) *es gilt*

$$\ell(C) \underset{\Delta ungl.}{\leq} \ell(W) = \ell(M) + \ell(T) \leq \frac{3}{2} opt(K_n, \ell)$$

Beweis von 2. Schritt

Gut aber warum gilt: $\ell(M) \leq \frac{1}{2} opt(K_n, \ell)$?

- Betrachte Hamiltonkreis C_{opt} der Länge $\ell(C_{opt}) = opt(K_n, \ell)$
- Die Knoten aus S zerlegen den Kreis C_{opt} in S viele Pfade.
- Reduziere diese Pfade zu einer Kante, ohne die Länge von C_{opt} zu erhöhen ($\Delta-$ungleichung)
- Jetzt haben wir einen anderen Kreis C_S der aus Knoten aus S entsteht mit $C_S \leq C_{opt}$.
- C_S kann man als Vereinigung von zwei perfekten Matchings schreiben: $C_S = M_1 \cap M_2$
- Einer dieser Matchings muss Länge $\leq \frac{1}{2}\ell(C_S)$ haben, sonst ergibt die Summe $\ell(M_1) + \ell(M_2)$grösser als C_S
- Da wir ein minimales Matching finden bei Schritt 2, wählen wir dieses kürzere Matching. Also gilt: $\ell(M) \leq \frac{1}{2}\ell(C_S) \leq \frac{1}{2}\ell(C_{opt}) = \frac{1}{2} opt(K_n, \ell)$ $\square$

TSP $\frac{3}{2}$-Approximationsalgorithmus: Fazit

Satz 1.51

Für das Metrische Travelling Salesman Problem gibt es einen $\frac{3}{2}$-Approximationsalgorithmus mit Laufzeit $O(n^3)$.

Exercises

Quiz auf der Webseite!

The End

♠