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. **Exercise: Perfect Bipartite Matching**
6. **Färbungen**

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: tree

RECAP: 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.

RECAP: Aug. Pfade: Algorithmus

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

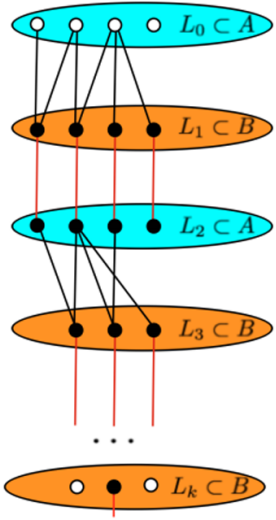
```
Start with M = ∅
REPEAT:
    Search for augmenting path P
    if no such path exists then return M
    else M = M ⊕ P
```

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)

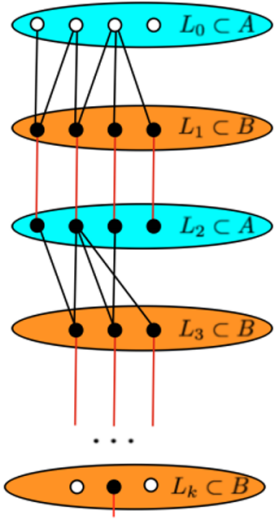
RECAP: Aug. Pfade: Algorithmus

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.



RECAP: 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.



RECAP: 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.

RECAP: Pseudocode

```
AUGMENTING_PATH (G = (A ⊎ B, E), M)
L_0 := {unüberdeckte Knoten in A}
Markiere alle Knoten aus L_0 als besucht.
if L_0 = ∅ then
    return M ist maximal
for all i = 1 to n do
    if i ungerade then
        L_i := {unbesuchte Nachbarn von L_{i-1} via Kanten in E \ M}
    else
        L_i := {unbesuchte Nachbarn von L_{i-1} via Kanten in M}
    Markiere alle Knoten aus L_i als besucht.
    if L_i enthält unüberdeckten Knoten v then
        Finde Pfad P von L_0 nach v durch backtracking
        return P // terminiert Algorithmus
return M ist schon maximal
```

RECAP: 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 $O(|V||E|)$.

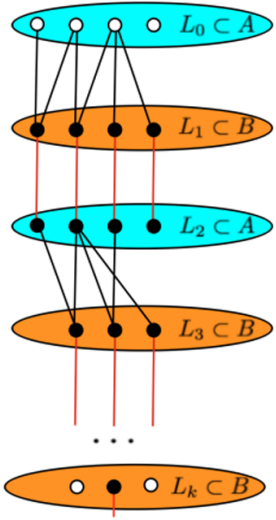
Hopcroft und Karp hatten eine bessere Idee.

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.
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.

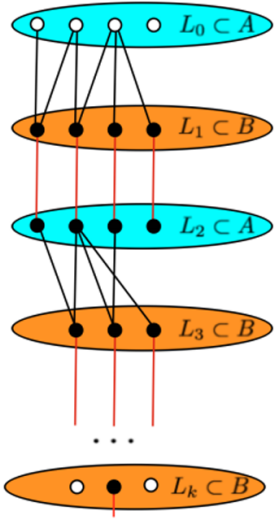
Wie? DFS!



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.
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:

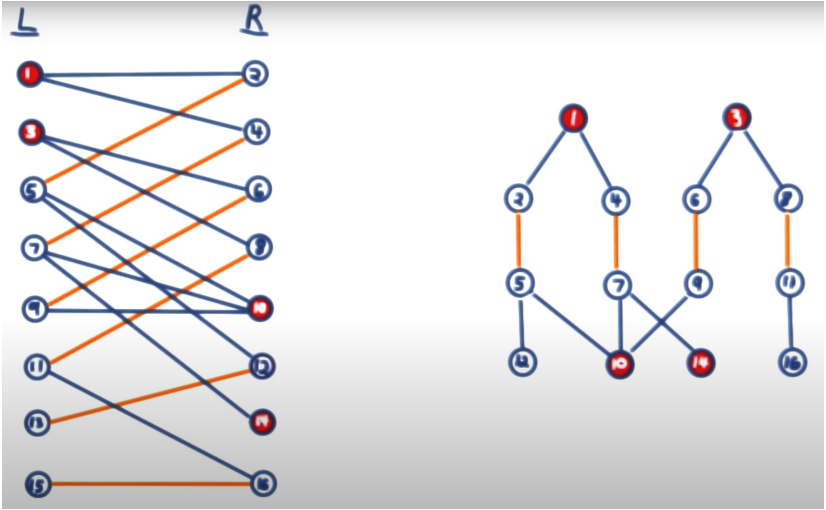
1. Alle Pfade in S haben dieselbe **minimale Länge** k (k ist also die Länge eines kürzesten augmentierenden Pfades)
2. Alle Pfade in S sind **paarweise disjunkt**.
3. S ist **inklusionsmaximal** mit dieser Eigenschaft, d.h. es lässt sich kein weiterer augmentierender Pfad der Länge k zu S hinzufügen,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

Hopcroft-Karp: Pseudocode

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

```
M := {e} für irgendeine Kante e ∈ E.
while es gibt noch augmentierende Pfade do
    k := Länge eines kürzesten augmentierenden Pfades
    Finde eine inklusionsmaximale Menge S von paarweise disjunkten
augmentierenden Pfaden der Länge k.
    for all P aus S do
        M := M ⊕ P. // augmentiere entlang der Pfade aus S
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.

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. $|P \cap P'| = 0$ für alle Pfade P in S $\rightarrow |P'| \geq k+1 \overset{\text{aug pfad len ungerade}}{\rightarrow} |P| \geq k+2$
2. $|P \cap P'| \geq 1$ für einen Pfaden P in S $\rightarrow$ Lemma 1

[0] $k :=$ Länge des kürzesten aug. Pfades, $S :=$ inklusionsmax. Menge der kürzesten aug. Pfade in einem Iter

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 mindestens $|M'| - |M|$ 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:

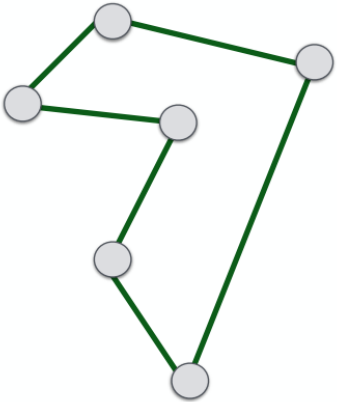
$$|M_{max}| \leq |M| + \frac{|V|}{k+1} \leq |M| + \sqrt{|V|} \text{ also } |M| \geq |M_{max}| - \sqrt{|V|}$$

- 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 $2 \cdot \sqrt{V}$ Durchläufe. Jeder Durchlauf geht in $O(|V| + |E|)$ (DFS). Insgesamt $O(\sqrt{|V|}(|V| + |E|))$ $\square$

Back to TSP

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

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*

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

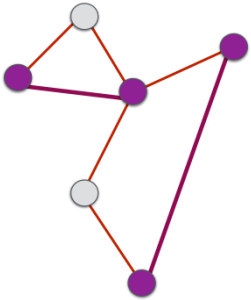
Minimales perfektes Mathcing

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 $O(n^3)$.

Exercise: Perfect Bipartite Matching

Let $G = (A \dot{\cup} B, E)$ be a bipartite k-regular graph (i.e. every vertex has degree exactly k).

(a) Show that G contains a perfect matching.

(b) Show that G contains k pairwise disjoint perfect matchings.
Hint: use (a)

[0] *Hint 2: use Hall's Theorem*

2^k-regulaerer bipartiter Graph

Satz 1.54. Ist $G = (V, E)$ ein 2^k-regulärer bipartiter Graph, so kann man in Zeit $O(|E|)$ ein perfektes Matching bestimmen.

Färbung

Färbung

Färbung: Definition

(Knoten-) Färbung

Eine Knotenfärbung eines Graphen $G = (V, E)$ mit k Farben ist eine Abbildung $c : V \to [k]$, sodass gilt

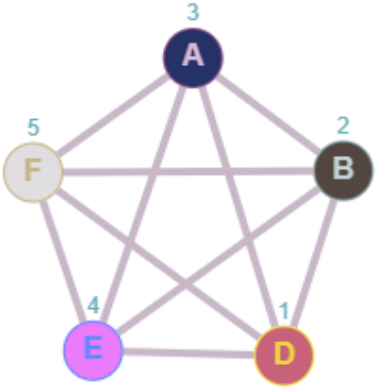
$$c(u) \neq c(v) \text{ für alle Kanten } \{u, v\} \in E.$$

Die *chromatische Zahl* $\chi(G)$ ist die minimale Anzahl Farben, die für eine Knotenfärbung von G benötigt wird.

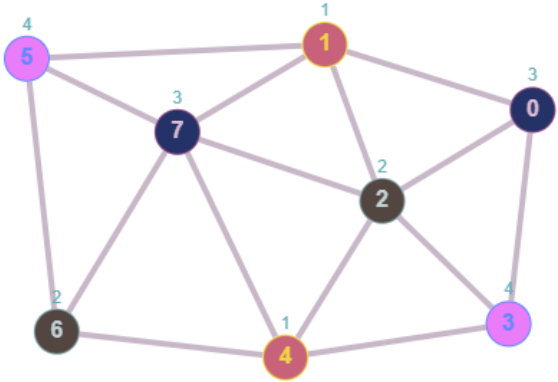
Chromatische Zahl $\chi(G) = k \iff$ G ist k-partit.

[0] *Engl. vertex coloring, chromatic number*

Färbung: Beispiele



Vollständig: $\chi(K_n) = n$



G planar: $\chi(G) \leq 5$

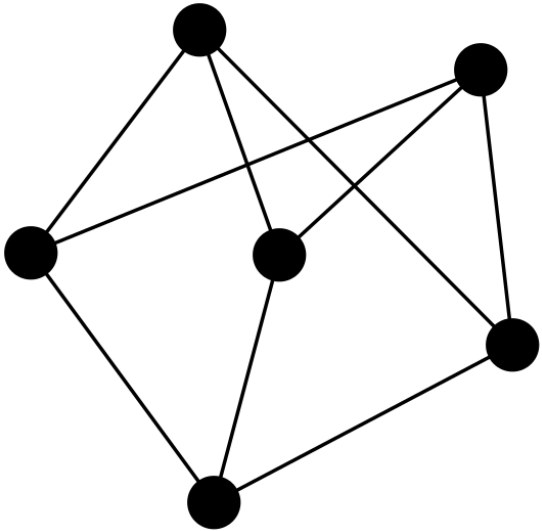
Bipartit Bedingung

Bipartit Bedingung (Satz 1.58.)

Ein Graph $G = (V, E)$ ist genau dann bipartit, wenn er keinen Kreis ungerader länge als Teilgraphen enthält.

- Kreise ungerader Länge haben chromatische Zahl 3.
- Bipartite Graphen sind 2 färbbar per Definition. Also können keine Kreise ungerader Länge enthalten.
- Die Graphen, die keinen Kreis ungerader Länge enthalten sind bipartit: Starte BFS mit einem beliebigen Knoten. Farbe Knoten nach Parität ihrer Distanz zum Startknoten.

Ist dieser Graph bipartit?



Vierfarbensatz

Vierfarbensatz

Jede Landkarte lässt sich mit vier Farben färben.

Man nimmt an ...

- ... dass das Gebiet eines heden Landes zusammenhngend ist
- ... und dass Länder, die nur in einem einzigen Punkt aneinanderstossen, gleich gefärbt werden dürfen.

Beweis: Theoretisch + Computerprogramm

Greedy Färbung

Go greedy! Wähle beliebige Reihenfolge, für jeden Knoten benutze die *kleinste* Farbe, die unter Nachbarknoten noch nicht aufgetaucht ist.

GREEDY-FÄRBUNG (G)

1: wähle eine beliebige Reihenfolge der Knoten: $V = \{v_1, \ldots, v_n\}$
2: $c[v_1] \leftarrow 1$
3: **for** $i = 2$ **to** n **do**
4: $\quad c[v_i] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ für alle } u \in N(v_i) \cap \{v_1, \ldots, v_{i-1}\}\}$

Greedy: Schlimmster Fall

Algorithmus bestimmt zulässige Färbung per konstruktion. Wie viele Farben werden benutzt?

Schlimmster Fall für einen Knoten v_i: Wenn die Nachbarknoten von v_i schon mit Farben $1, \ldots, deg(v_i)$ gefärbt sind. Dann brauchen wir noch eine Farbe für v_i

Schlimmster Fall für den ganzen Graphen: Falls v_i maximalen Grad hat.

Also braucht der Algorithmus

$$\Delta(G) + 1$$

Farben im schlimmsten Fall wobei

$$\Delta(G) := \text{maximaler Grad eines Knotens in G}$$

Greedy Färbung

Greedy Färbung

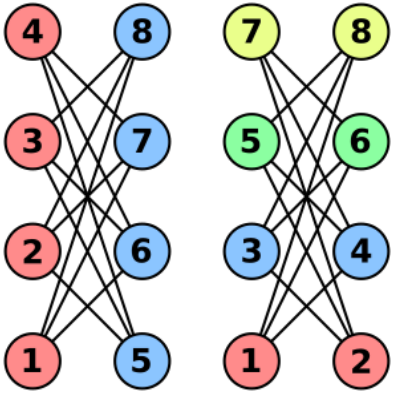
Sei G ein zusammenhängender Graph. Für die Anzahl Farben $C(G)$, die der Algorithmus Greedy-Färbung benötigt, um die Knoten des Graphen G zu färben, gilt

$$\chi(G) \leq C(G) \leq \Delta(G) + 1$$

Ist der Graph als Adjazenzliste gespeichert, findet der Algorithmus die Färbung in Zeit $O(|E|)$.

- Es gibt immer eine Reihenfolge, bei der der Greedy-Algorithmus mit $\chi(G)$ Farben auskommt.
- Anzahl Farben, die Greedy-Algorithmus braucht, um G zu färben, hängt von der gewählten Reihenfolge der Knoten ab.

Greedy Färbung: Beispiel



Ein bipartiter Graph ist 2-färbbar, jedoch kann Greedy-Algorithmus im schlimmsten Fall deutlich mehr Farben benutzen.

Merkwürdige Beispiele: Bsp 1.62.

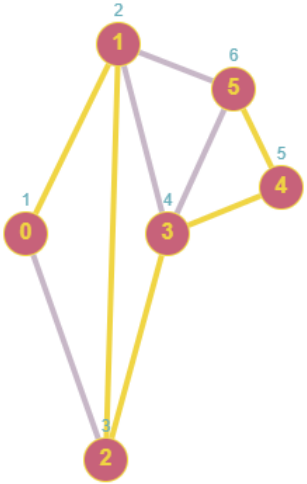
Sei $G = (V, E)$ wobei es einen Knoten $v \in V$ gibt, mit $deg(v) < \Delta(G)$.

- starte DFS (od. BFS) bei v.
- nummeriere Knoten in umgekehrter Reihenfolge, wie sie vom Algorithmus durchlaufen werden

So hat jeder Knoten v_i mit $i < n$ mindestens einen Nachbarn v_j mit $j > i$ und daher höchstens $\Delta(G) - 1$ gefärbte Nachbarn.

Der Knoten v hat auch meistens $\Delta(G) - 1$ (gefärbte) Nachbarn.

Greedy-Algorithmus kann für diese Reihenfolge höchstens $\Delta(G)$ Farben benutzen.

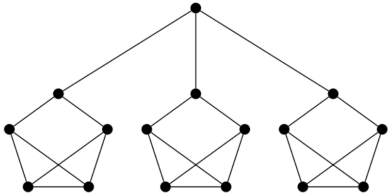


Merkwürdige Beispiele: Bsp 1.63

Sei $G = (V, E)$ ein zusammenhängender k-regulärer Graph, in dem es mindestens einen Artikulationsknoten gibt.

- finde Art.knoten v in $O(|E|)$
- Seien Zusammenhangskomponenten von $G - v$: $V_1 \dots V_s$
- Die Graphen $G_i = G[V_i \cup \{v\}]$ erfüllen die Annahme von *Beispiel 1.62.*
- Färbe alle solche Subgraphen jeweils mit k Farben
- Stelle sicher dass in allen G_i v die gleiche Farbe hat.

Am Ende haben wir eine k-Färbung.



Bsp: 3-regulär

Satz von Brooks

Satz von Brooks

Ist $G = (V, E)$ ein zusammenhängender Graph, der weder vollständig ist noch ein ungerader Kreis ist, also $G \neq K_n$ und $G \neq C_{2n+1}$, so gilt

$$\chi(G) \leq \Delta(G)$$

und es gibt einen Algorithmus, der die Knoten des Graphen in Zeit $O(|E|)$ mit $\Delta(G)$ Farben färbt.

Beweis: Benutze Beispiele oben + Fallunterscheidung

Ein Trick: Subgraphen

Satz 1.65.

Ist $G = (V, E)$ ein Graph und $k \in \mathbb{N}$ eine natürliche Zahl mit der Eigenschaft, dass jeder induzierte Subgraph von G einen Knoten mit Grad höchstens k enthält, so gilt $\chi(G) \leq k + 1$ und eine $(k+1)-$ Färbung lässt sich in Zeit $O(|E|)$ finden.

- Das ist unsere Heuristik aus der Vorlesung.
- Nach Annahme gibt es einen Knoten v im Graphen G, der Grad höchstens k hat. Wir beginnen mit diesem Knoten.
- Dann gibt es auch im Graphen $G - v$ einen Knoten der Grad höchstens k hat.
- So nummerieren wir die Knoten. Dann beutzen wir die umgekehrte Reihenfolge.
- Da in dieser Reihenfolge jeder Knoten höchstens k gefärbte Nachbar haben, wenn betrachtet von Greedy-Algorithmus, können wir den Graphen insgesamt mit $k + 1$ Farben färben.

The End

♠