

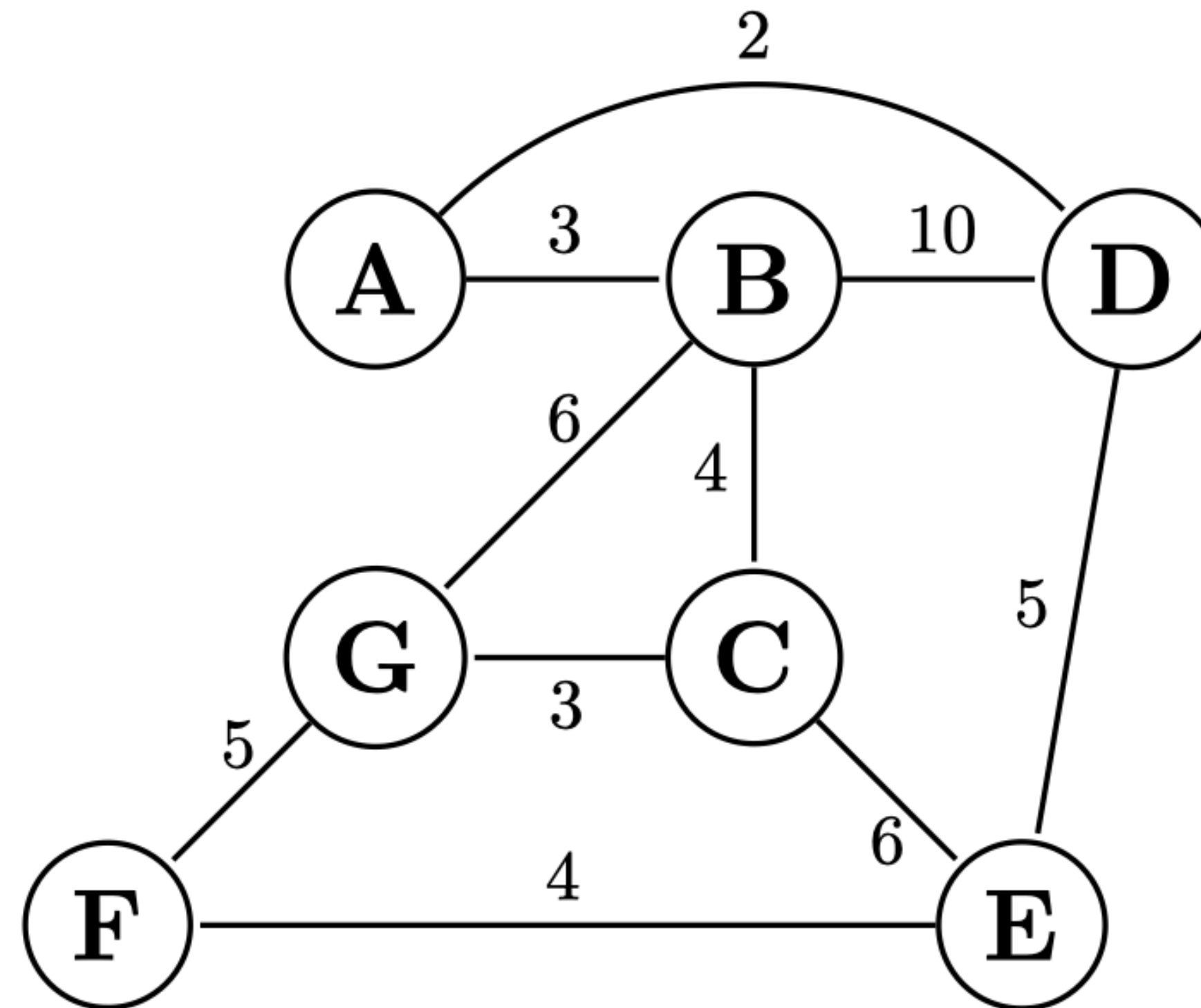
Algorithms and Probability

Week 4

Peergrading

Aufgabe 1 – *Touren*

Betrachten Sie den folgenden Graphen G :



Die Zahl neben jeder Kante e bezeichnet das Gewicht $\ell(e)$ dieser Kante. Das Gewicht eines Weges $W = v_1, v_2, \dots, v_k$ ist definiert als $\ell(W) = \ell(\{v_1, v_2\}) + \ell(\{v_2, v_3\}) + \dots + \ell(\{v_{k-1}, v_k\})$.

Aufgabe 1 – *Touren*

- (a) Finden Sie einen minimalen Spannbaum T von G . Begründen Sie Ihre Schritte/Rechnungen genau.

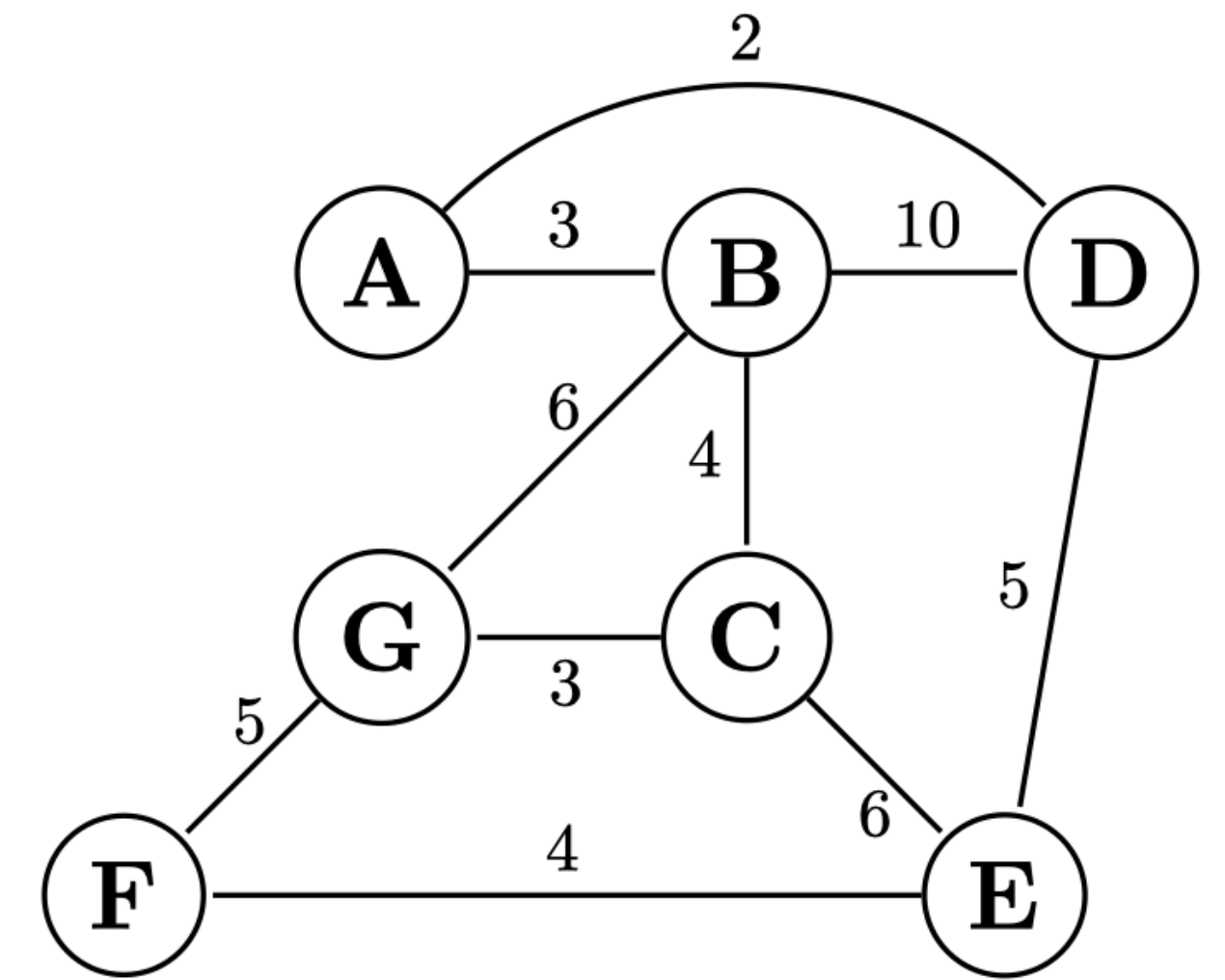
Kruskal to find MST T in G .

Sort edges by weight (ascending):

$\{A,D\}, \{A,B\}, \{C,G\}, \{B,C\}, \{E,F\}, \{F,G\}, \{D,E\}, \{B,G\}, \{B,D\}$

Pick edge $e = \{u, v\}$ if no $u - v$ path present in T .

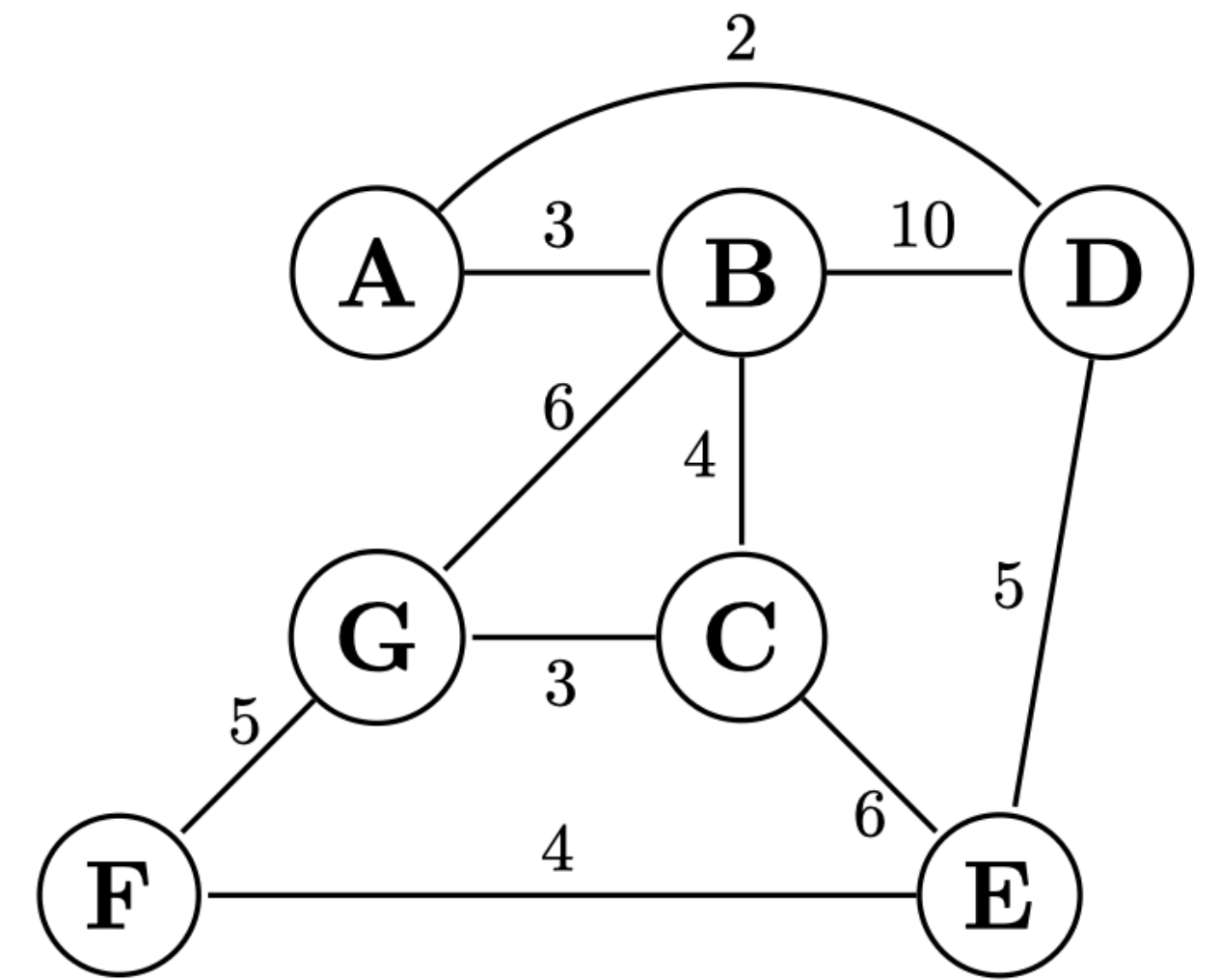
We get: $\{A,D\}, \{A,B\}, \{C,G\}, \{B,C\}, \{E,F\}, \{F,G\}$



Aufgabe 1 – *Touren*

(b) Bezeichne C das Gewicht von T . Verwenden Sie T um einen Zyklus Z zu finden, der in A beginnt und endet, jeden Knoten von G mindestens einmal besucht und Gewicht höchstens $2C$ hat.

- We double all edges in T to get T' .
- Every vertex in T' has degree $2 \cdot \deg_T(v)$ and is thus even.
- Satz 1.31 $\Rightarrow T'$ has a Eulertour, that traverses each edge exactly once and thus every vertex at least once. It has Weight $2 \cdot \sum_{e \in T} l(e) = 2 \cdot C$ as desired.



Aufgabe 1 – *Touren*

- (c) Beweisen Sie, dass jeder Zyklus Z , der jeden Knoten mindestens einmal besucht, mindestens Gewicht C hat.

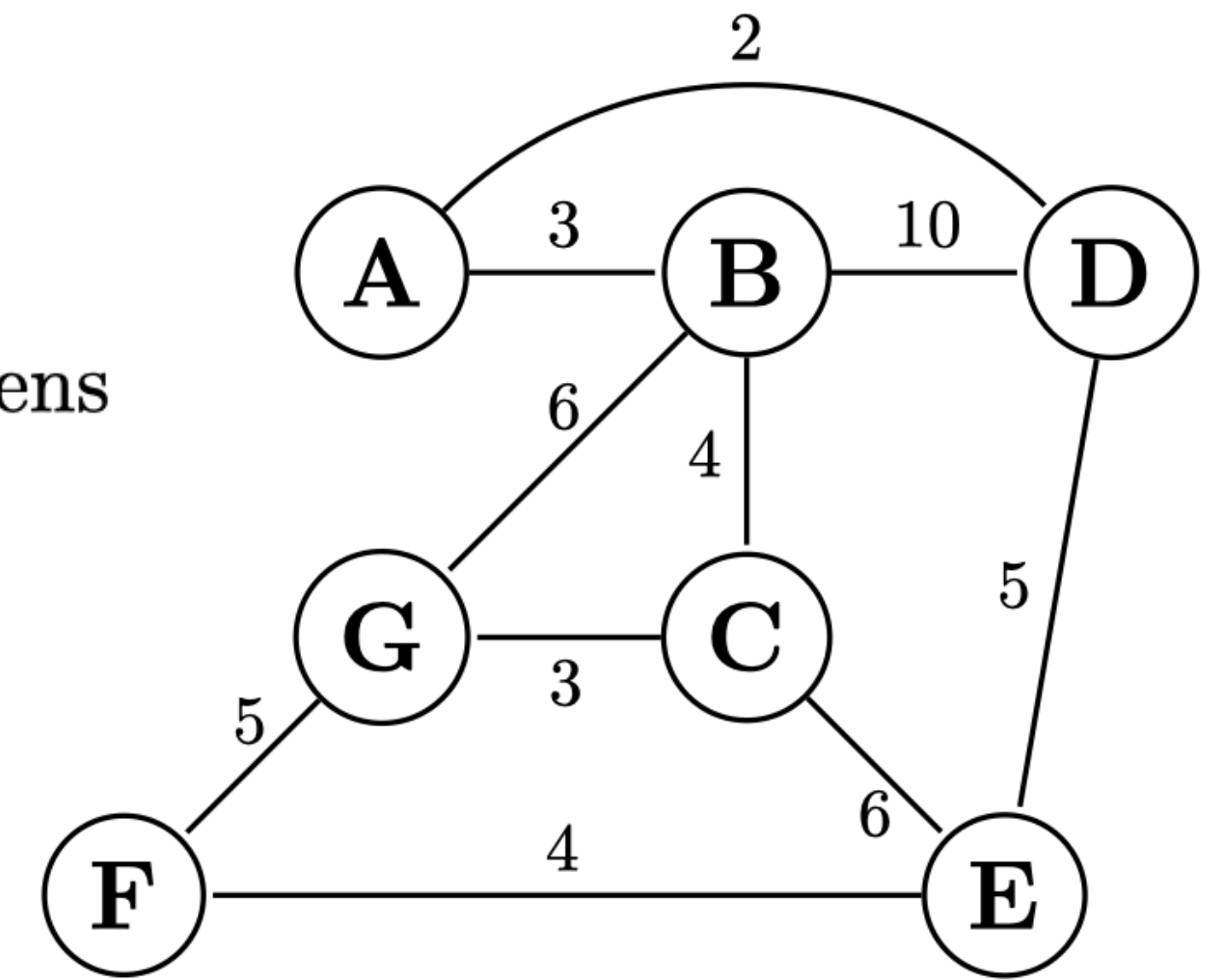
Let Z be some arbitrary Zyklus that fulfills the properties and let E_Z be the set of edges in Z (no duplicates).

As long as there exists a cycle with edges in E_Z remove one edge from that said cycle.

$E'_Z \subseteq E_Z$ are the remaining edges.

$G' = (V, E'_Z)$ is connected and spanning, since removing an edge from a cycle doesn't disconnect. G' is a spanning tree.

We have $C \leq \sum_{e \in E'_Z} l(e) \leq \sum_{e \in E_Z} l(e) = l(Z)$, since $C = l(T)$ (MST!).



Theory Recap

3/2-approximation-algorithm for metric TSP

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

1. Determine MST T in G .
2. Determine perfect matching of minimum weight M on the vertices of odd degree in T . Add matching M to T , i.e. let $T' = T \cup M$ (possibly multigraph).
3. Find Eulerian circuit E in T' .
4. Shorten E .

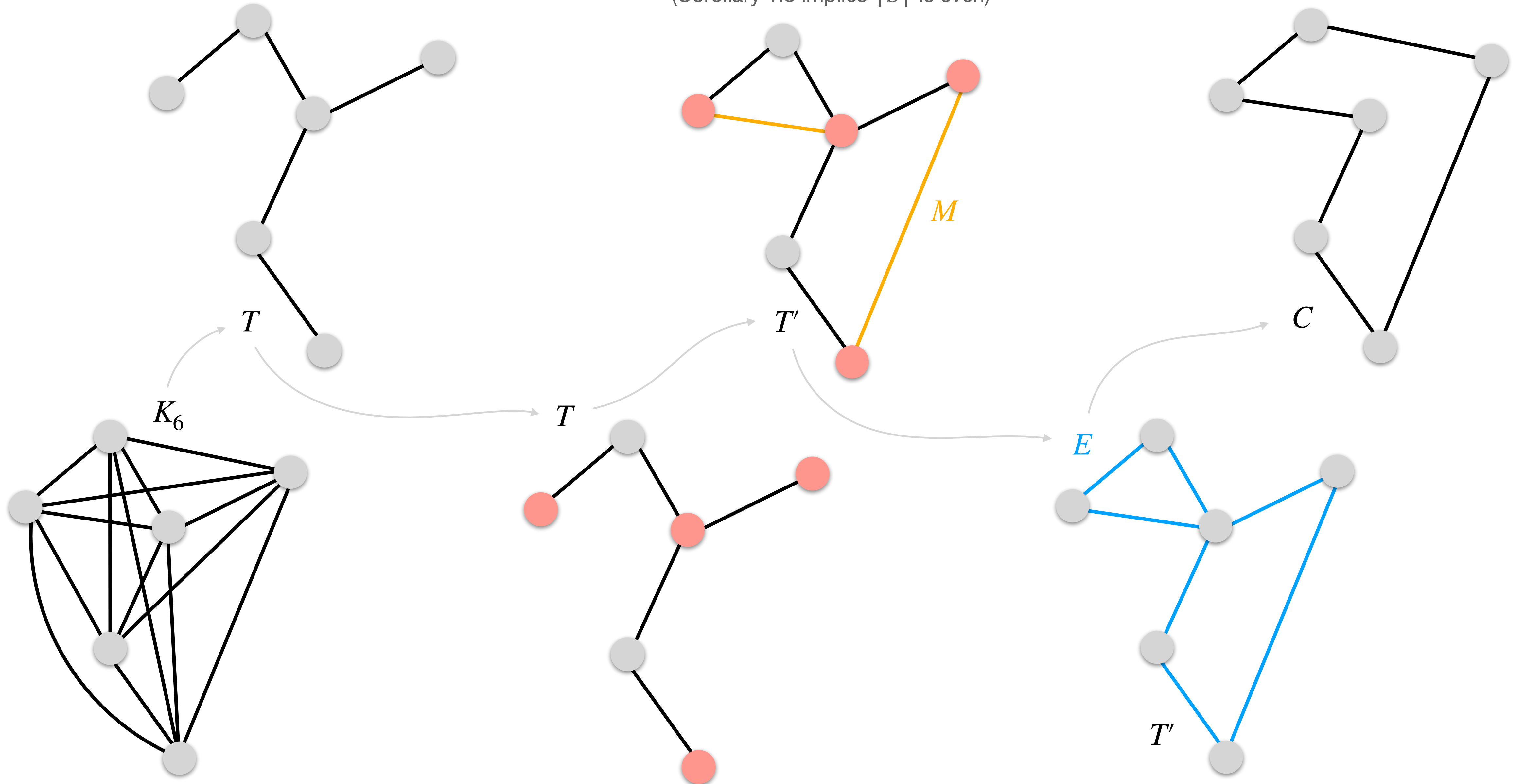


Step 2. is different from the 2-approximation.

T is a MST in K_6 .

M is a perfect matching on the vertices of S .
(Corollary 1.3 implies $|S|$ is even)

C is a Hamiltonian cycle s.t. $l(C) \leq 3/2 \cdot l(C_{\text{opt}})$.

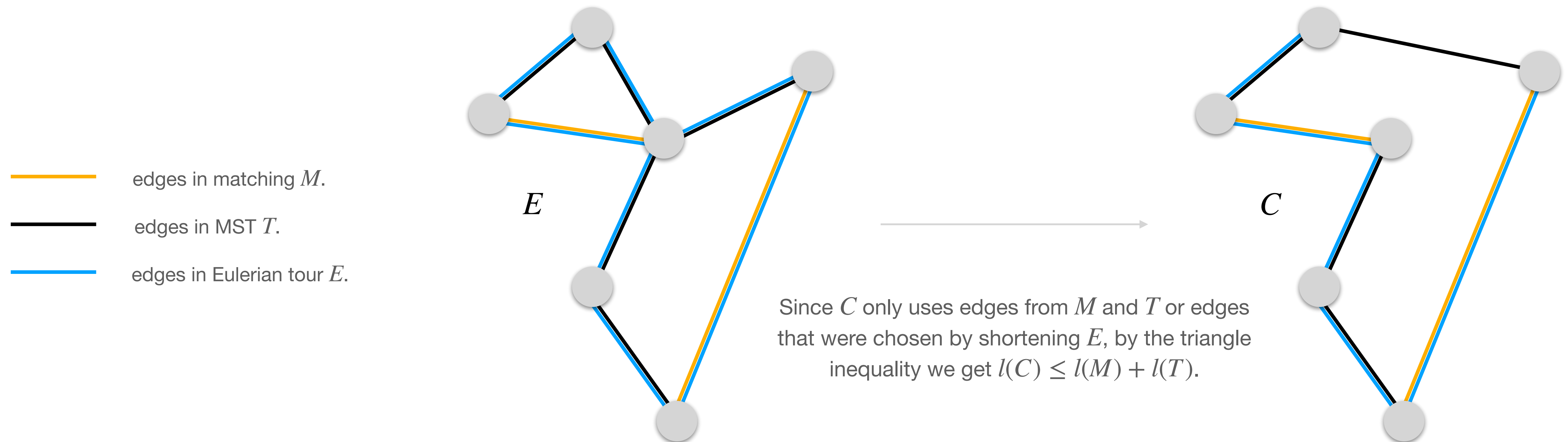


S contains the vertices of odd degree in T .

E is a Eulerian tour in K_6 .
(All vertices in T' have even degree)

Notice that $l(C) \leq l(M) + l(T)$. We know $l(T) \leq l(C_{\text{opt}})$.

If we show $l(M) \leq 1/2 \cdot l(C_{\text{opt}})$ then $l(C) \leq 3/2 \cdot l(C_{\text{opt}})$ as desired.

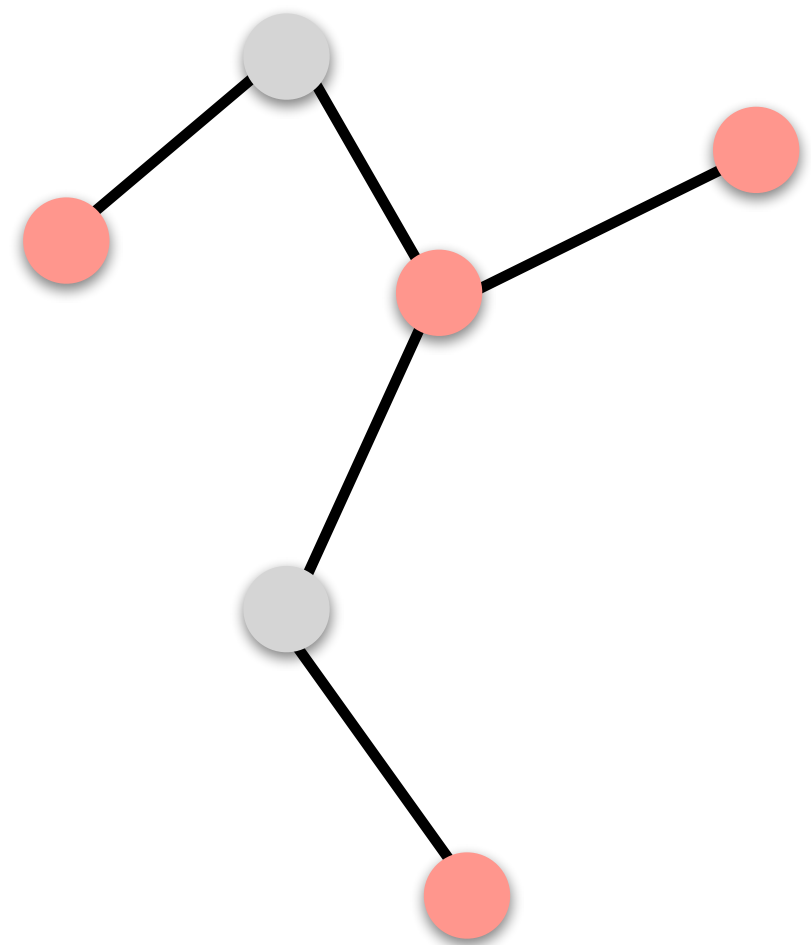


The vertices of odd degree (of T) in S partition C_{opt} in $|S|$ paths.

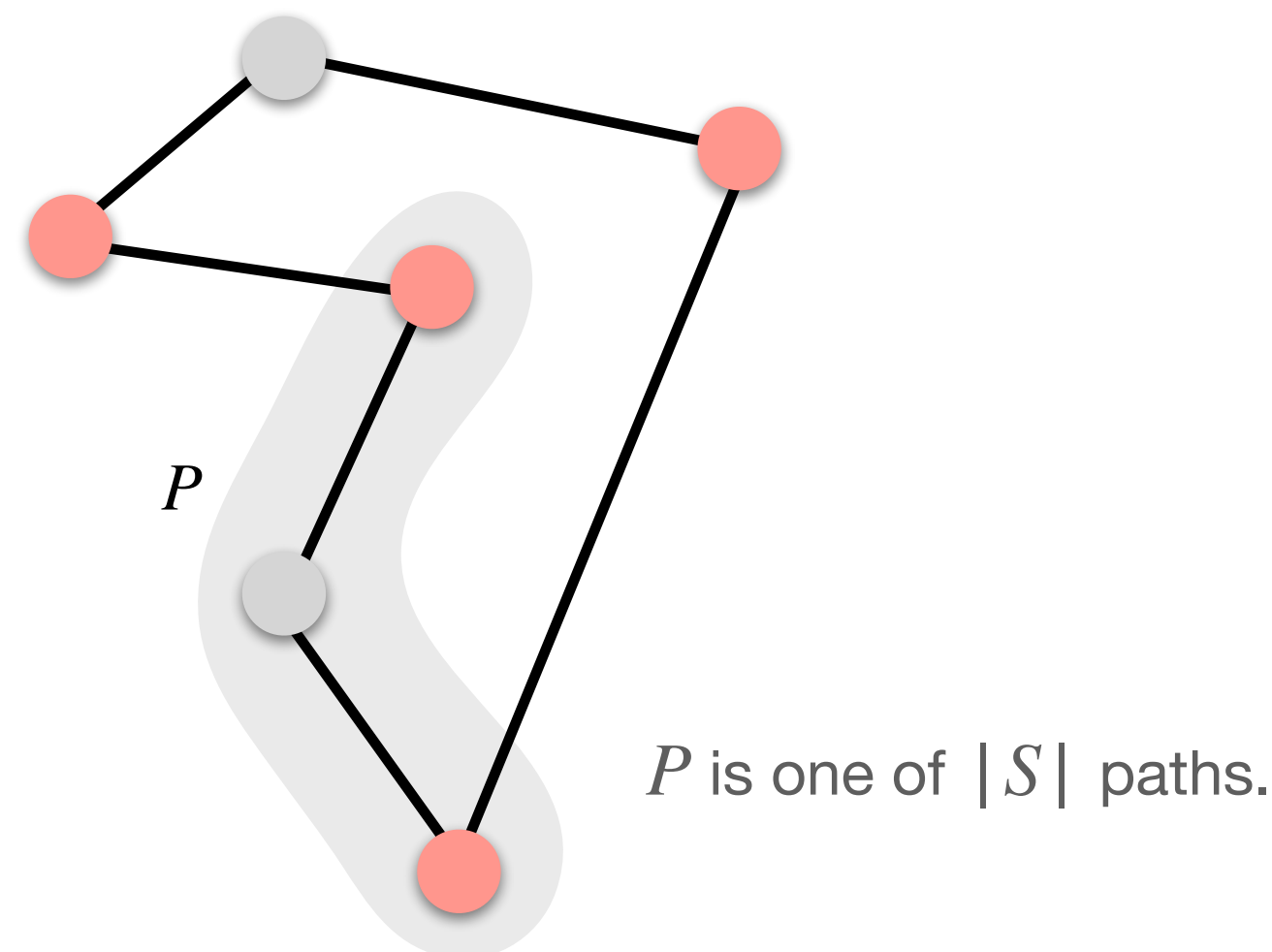
Shorten each path to one edge to get C_S with $l(C_S) \leq l(C_{\text{opt}})$ (triangle ineq.).

$C_S = M_1 \cup M_2$ (each matching is of size $|S|/2$), one must have $\leq 1/2 \cdot l(C_S)$.

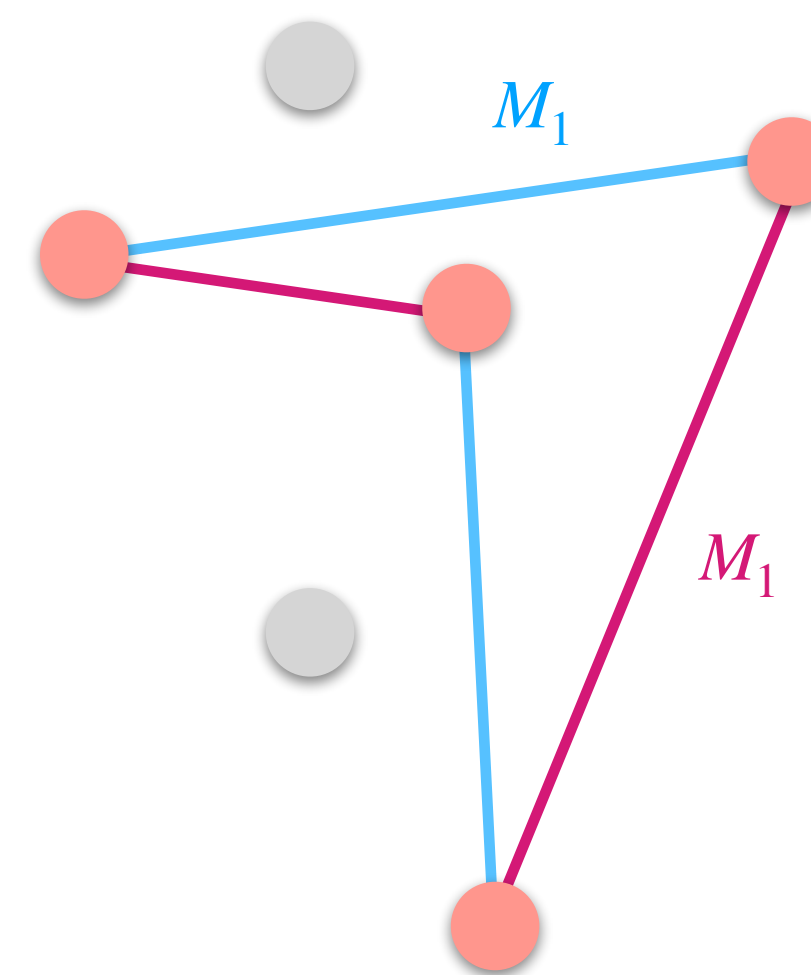
By definition of M this yields $l(M) \leq 1/2 \cdot l(C_S) \leq 1/2 \cdot l(C_{\text{opt}})$.



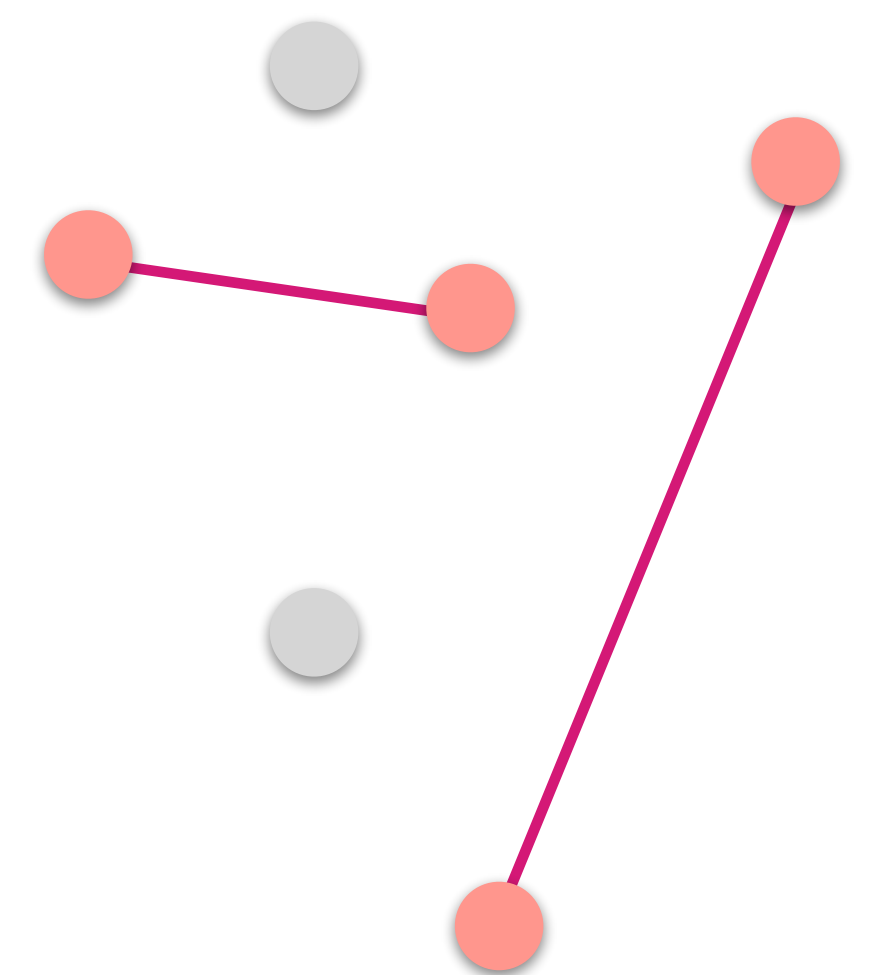
4 vertices have odd degree.



C_{opt} contains every $v \in V$,
therefore also the vertices in S .



$|S|$ shortened paths form
 $C_S = M_1 \cup M_2$.

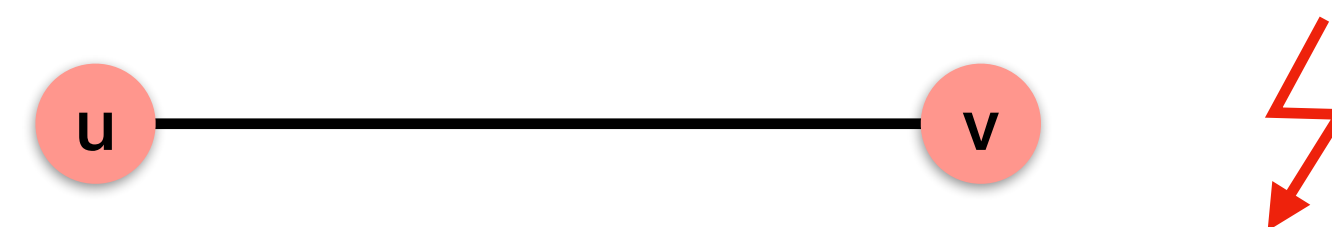
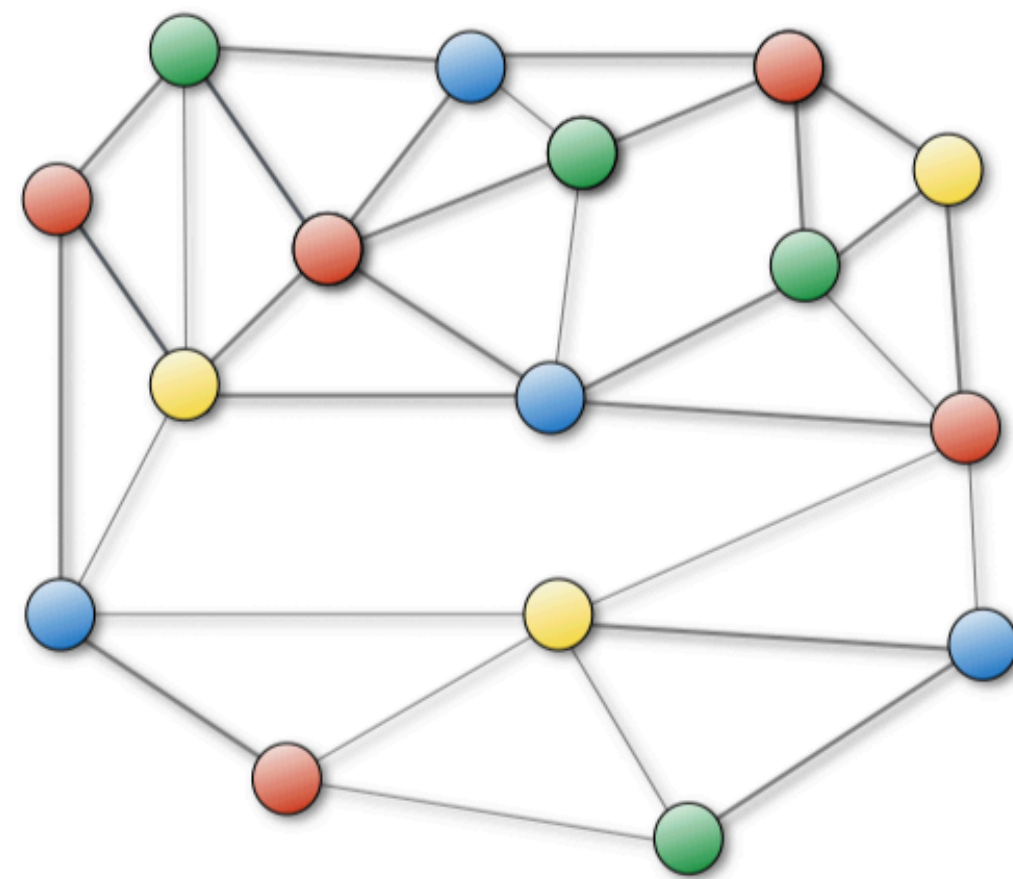


One matching must be the
 $\leq 1/2 \cdot l(C_S)$ since
 $l(M_1) + l(M_2) \leq l(C_S)$

Coloring

Eine (Knoten-)Färbung eines Graphen $G = (V, E)$ mit k Farben ist eine Abbildung $c : V \rightarrow [k]$, so dass gilt

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



Two adjacent nodes cannot be colored in the same color.

Coloring

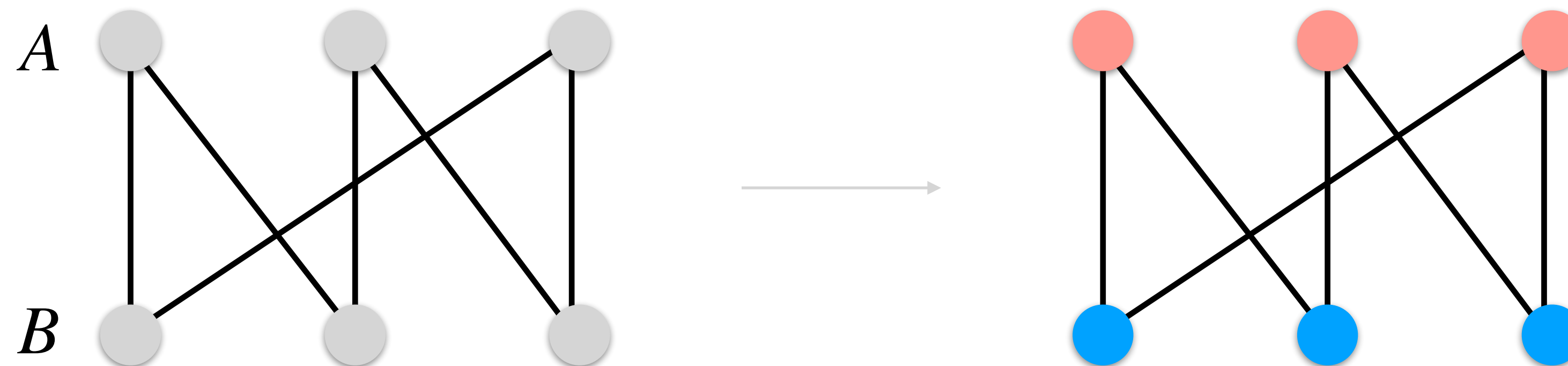
Die **chromatische Zahl** $\chi(G)$ ist die minimale Anzahl Farben, die für eine Knotenfärbung von G benötigt wird.

Äquivalente Formulierung: $\chi(G) \leq k$ gdw. G **k -partit**

Coloring

Graphen mit chromatischer Zahl k nennt man auch k -*partit* (engl. k -*partite*). Die Motivation für diese Namensgebung sollte klar sein: Ein Graph $G = (V, E)$ ist genau dann k -partit, wenn man seine Knotenmenge V so in k Mengen V_1, \dots, V_k partitionieren kann, dass alle Kanten Knoten aus verschiedenen Mengen verbinden. Besonders wichtig ist der Fall $k = 2$.

Example: 2-partite (bipartite)



A vertex from A can't be connected to a vertex in B .
A vertex of color c_1 can't be connected to a vertex of color c_2 .

Coloring

Satz: Für jedes $k \geq 3$ ist das Problem

„Gegeben ein Graph $G = (V, E)$, gilt $\chi(G) \leq k$?“

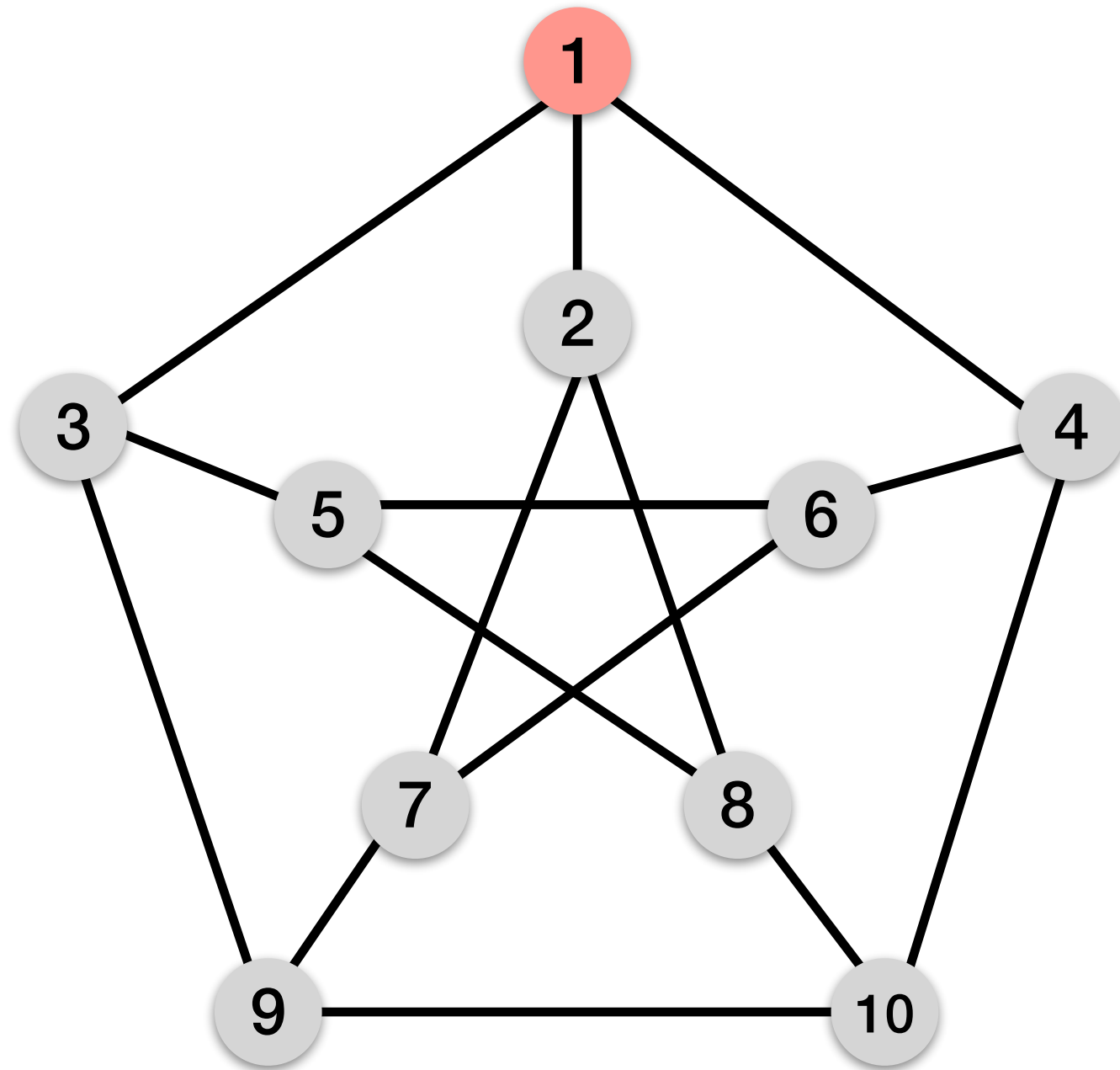
NP-vollständig.

Greedy coloring

GREEDY-FÄRBUNG (G)

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

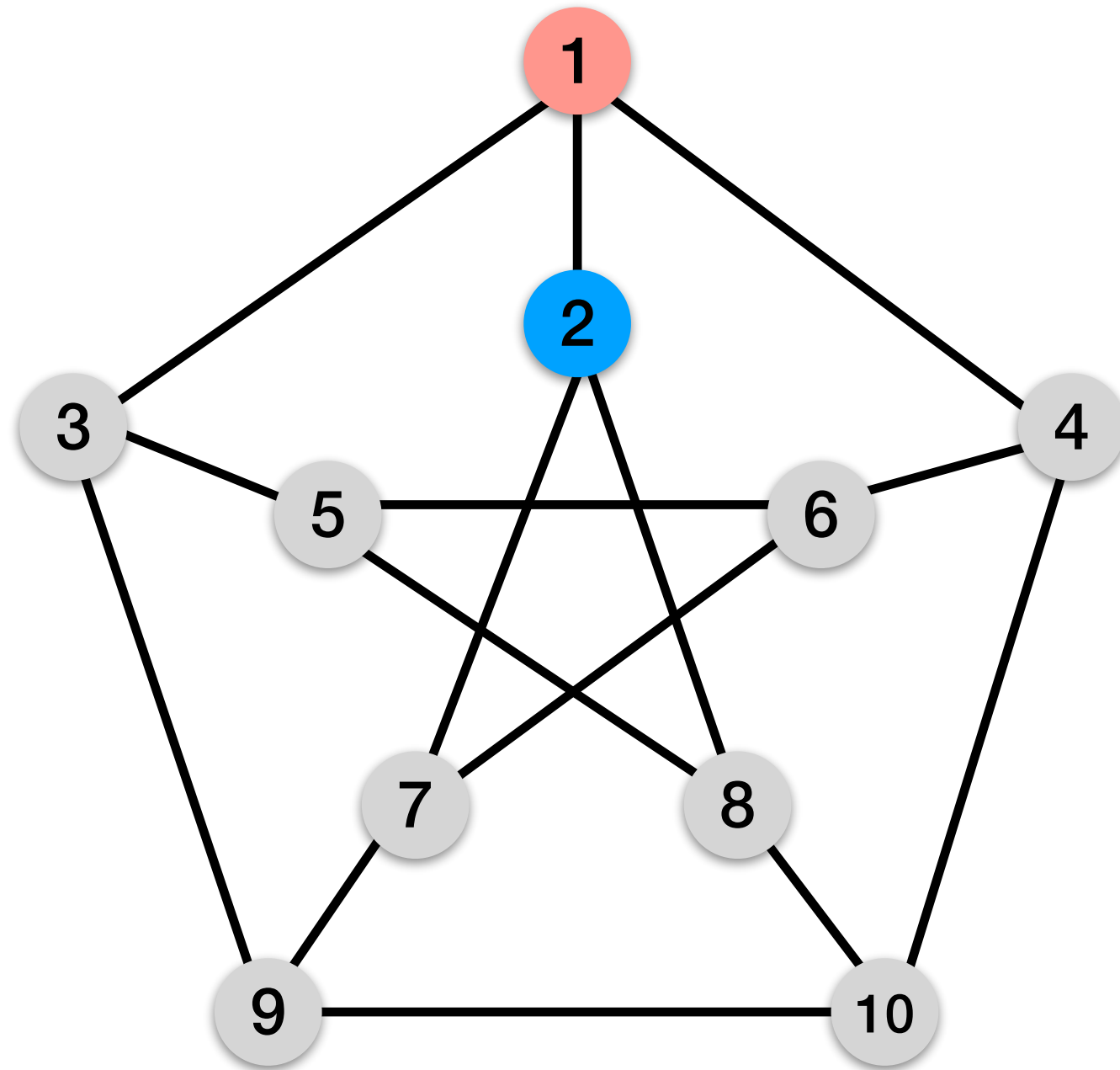
- 1: wähle eine beliebige Reihenfolge der Knoten: $V = \{v_1, \dots, v_n\}$
- 2: $c[v_1] \leftarrow 1$
- 3: **for** $i = 2$ **to** n **do**
- 4: $c[v_i] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ für alle } u \in N(v_i) \cap \{v_1, \dots, v_{i-1}\}\}$



$$c[\text{2}] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ for all } u \in \underbrace{N(v_2) \cap \{v_1\}}_{\{\text{1}\}}\}$$

→ 2

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



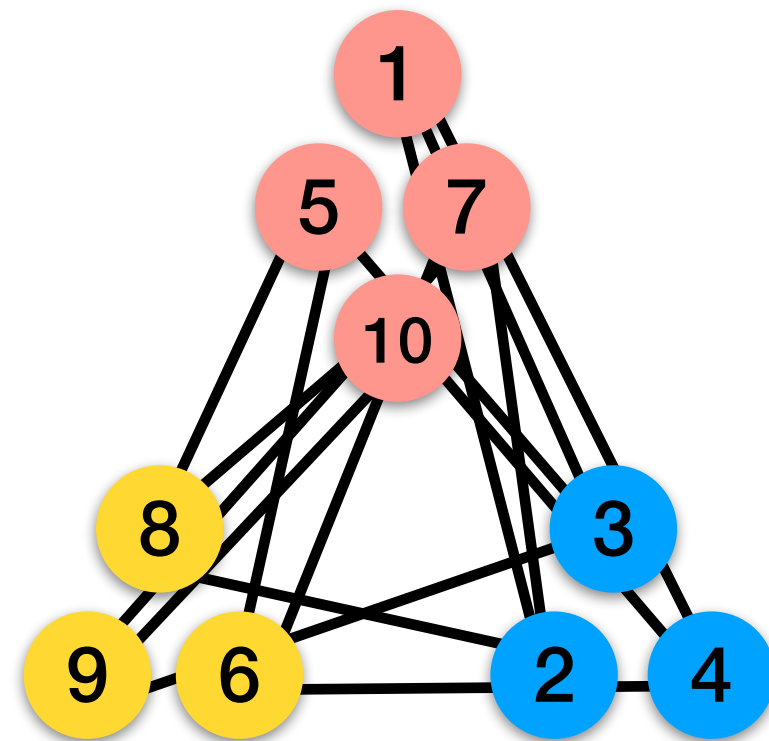
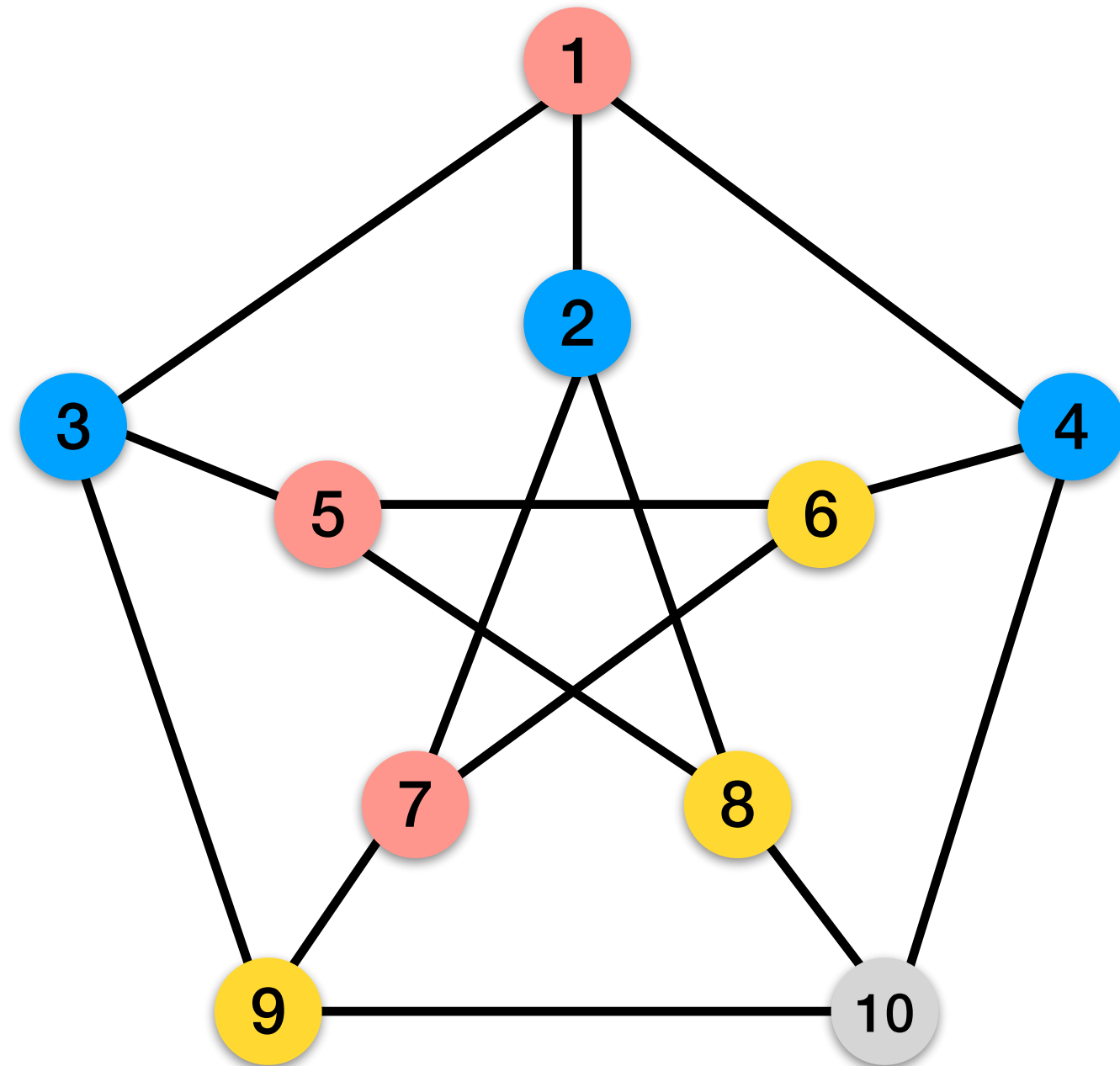
$$c[2] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ for all } u \in \underbrace{N(v_2) \cap \{v_1\}}_{\{1\}}\}$$

→ 2

$$c[3] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ for all } u \in \underbrace{N(v_3) \cap \{v_1\}}_{\{1\}}\}$$

→ 3

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



$$\begin{aligned}
 &c[2] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ for all } u \in \underbrace{N(v_2) \cap \{v_1\}}_{\{1\}}\} \\
 &\rightarrow 2 \\
 &c[3] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ for all } u \in \underbrace{N(v_3) \cap \{v_1\}}_{\{1\}}\} \\
 &\rightarrow 3 \\
 &\vdots \\
 &c[10] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ for all } u \in \underbrace{N(v_{10}) \cap \{v_1, \dots, v_9\}}_{\{4, 8, 9\}}\} \\
 &\rightarrow 10
 \end{aligned}$$

Satz 1.60. 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 $O(|E|)$.

Where $\Delta(G) := \max_{v \in V} \deg(v)$ denotes the highest degree of any vertex in V .

Satz 1.60. 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 $O(|E|)$.

Proof

Worst case?

All neighbors of v already colored, then $\deg(v) + 1$ colors needed. Hence at most $\Delta(G) + 1$ colors are needed.

Runtime: array $A[\deg(v) + 1] = [\text{false}, \dots, \text{false}]$. Go over neighbors n_i of v and set $A[c(n_i)] = \text{true}$. Go over A and set color to index of **first false entry**.

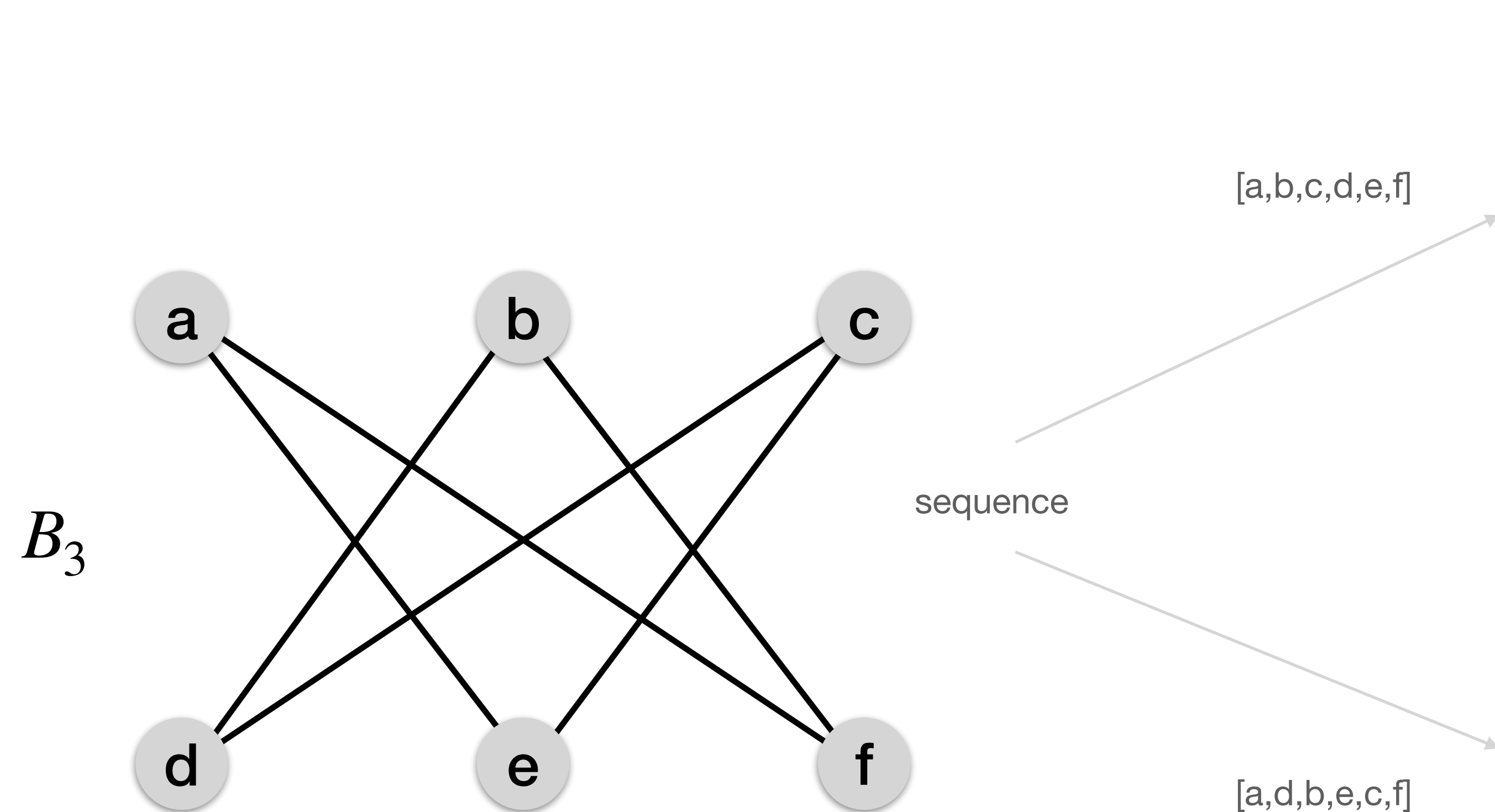
Coloring v takes $O(\deg(v))$: the algorithm runs in $O(\sum_{v \in V} \deg(v)) = O(|E|)$.

↑
Theorem 1.2.

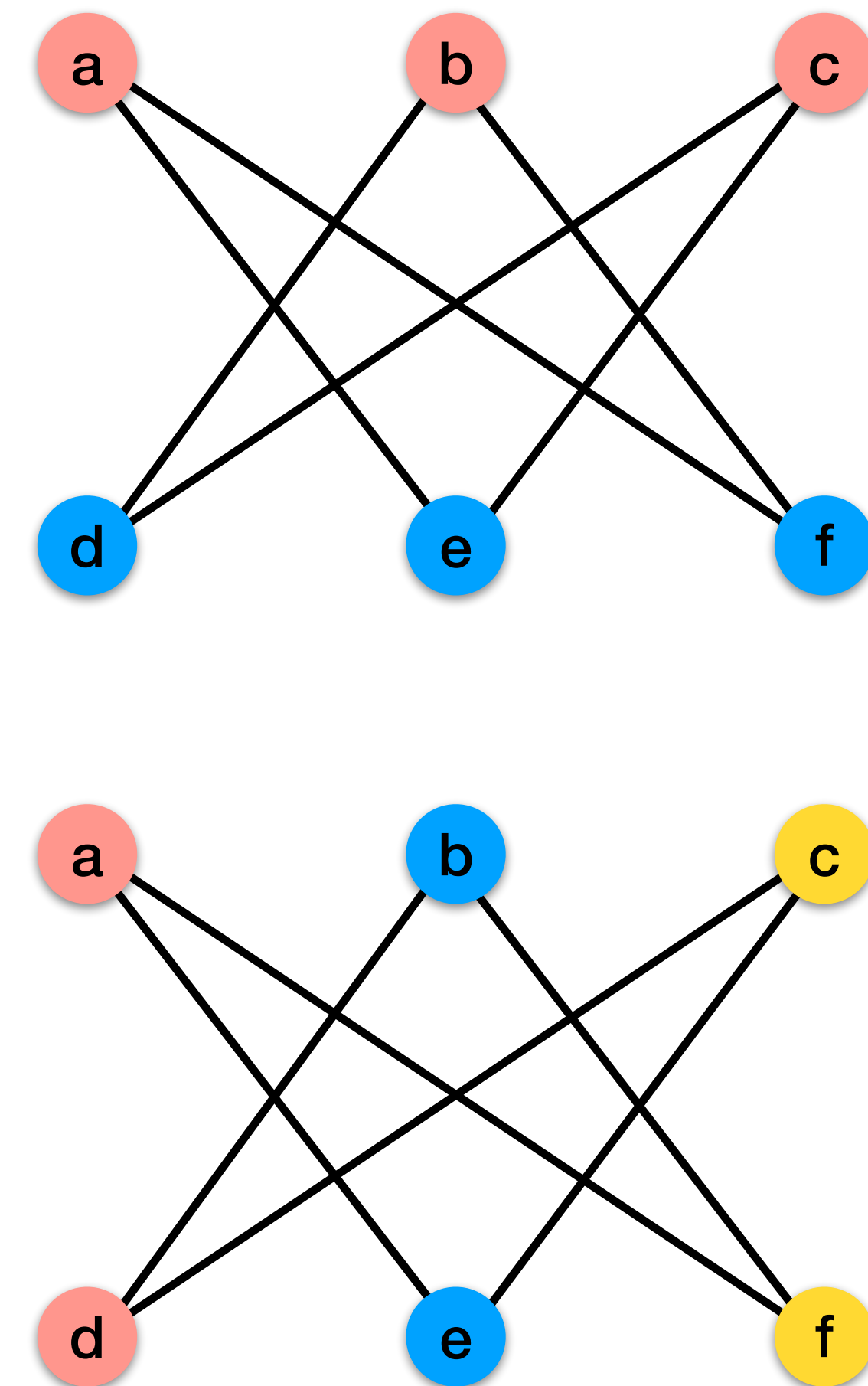
Remarks

- How many colors the greedy algorithm uses depends on the sequence of vertices.
- There always exists a sequence such that the algorithm needs $\chi(G)$ colors.

Beispiel 1.61. Betrachten wir den Graphen B_n mit $2n$ Knoten, der aus dem vollständigen bipartiten Graphen $K_{n,n}$ entsteht, indem man die Kanten zwischen gegenüberliegenden Knoten entfernt. Da der Graph B_n bipartit ist, könnte er eigentlich mit zwei Farben gefärbt werden; es ist aber nicht schwer einzusehen, dass es auch eine Reihenfolge der Knoten gibt, für die der Greedy-Algorithmus n Farben benötigt (Übung!).



Exercise



GREEDY-FÄRBUNG (G)

4: $c[v_i] \leftarrow \min \{k \in \mathbb{N} \mid k \neq c(u) \text{ für alle } u \in N(v_i) \cap \{v_1, \dots, v_{i-1}\}\}$

Beobachtung:

Gilt für die (gewählte) Reihenfolge $|N(v_i) \cap \{v_1, \dots, v_{i-1}\}| \leq k \quad \forall 2 \leq i \leq n$,
dann benötigt der Greedy-Algorithmus höchstens **k+1** viele Farben.

We want to minimize $|N(v_i) \cap \{v_1, \dots, v_{i-1}\}| \forall 2 \leq i \leq n$.

The bigger i gets, the bigger $|\{v_1, \dots, v_{i-1}\}|$ becomes.

Have v_i where $|N(v_i)|$ is small, **at the end** of the sequence.

Heuristik:

v_n := Knoten vom kleinsten Grad. Lösche v_n .

v_{n-1} := Knoten vom kleinsten Grad im Restgraph. Lösche v_{n-1} .

Iteriere.

Coloring

Heuristik:

v_n := Knoten vom kleinsten Grad. Lösche v_n .

v_{n-1} := Knoten vom kleinsten Grad im Restgraph. Lösche v_{n-1} .

Iteriere.

Falls $G=(V,E)$ erfüllt:

In jedem Subgraphen gibt es einen Knoten mit $\text{Grad} \leq k$

⇒ Heuristik liefert Reihenfolge v_1, \dots, v_n für die der Greedy-Algorithmus höchstens $k+1$ Farben benötigt

Coloring

Heuristik:

v_n := Knoten vom kleinsten Grad. Lösche v_n .

v_{n-1} := Knoten vom kleinsten Grad im Restgraph. Lösche v_{n-1} .
Iteriere.

Korollar:

$G=(V,E)$ zshgd. und es gibt $v \in V$ mit $\deg(v) < \Delta(G)$

⇒ Heuristik (oder Breiten-/Tiefensuche) liefert Reihenfolge,
für die der Greedy-Algorithmus höchstens $\Delta(G)$ Farben benötigt

Heuristik:

v_n := Knoten vom kleinsten Grad. Lösche v_n .

v_{n-1} := Knoten vom kleinsten Grad im Restgraph. Lösche v_{n-1} .
Iteriere.

Korollar:

$G=(V,E)$ zshgd. und es gibt $v \in V$ mit $\deg(v) < \Delta(G)$

⇒ Heuristik (oder Breiten-/Tiefensuche) liefert Reihenfolge,
für die der Greedy-Algorithmus höchstens $\Delta(G)$ Farben benötigt

Proof

Let $v \in V$ s.t. $\deg(v) < \Delta(G)$.


Start BFS/DFS in v .

The sequence for the greedy algorithm is the **reverse order** of how we traversed the vertices in G (i.e. $v = v_n$).

Then every v_i , $i < n$ has at least one uncolored neighbor v_j where $j > i$ and thus at most $\Delta(G) - 1$ **colored** neighbors.

We started in v with $\deg(v) < \Delta(G)$, then $v_n = v$ and v_n has at most $\Delta(G) - 1$ colored neighbors. Greedy algorithm needs at most $\Delta(G)$ colors.

Recommendation



Beispiel 1.62. Sei $G = (V, E)$ ein zusammenhängender Graph mit Maximalgrad $\Delta(G)$. Weiter nehmen wir an, dass es einen Knoten $v \in V$ gibt mit $\deg(v) < \Delta(G)$. Wenn wir jetzt eine Breiten- oder Tiefensuche in v starten und die Knoten in *umgekehrter* Reihenfolge nummerieren, wie sie vom Algorithmus durchlaufen werden (der Knoten v ist also der Knoten v_n), 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_n hat nach Wahl ebenfalls nur $\Delta(G) - 1$ gefärbte Nachbarn. Der Greedy-Algorithmus benötigt daher für diese Reihenfolge der Knoten höchstens $\Delta(G)$ Farben.

Beispiel 1.63. Sei $G = (V, E)$ ein zusammenhängender k -regulärer Graph, in dem es mindestens einen Artikulationsknoten gibt. Wir wissen bereits aus Abschnitt 1.4.1, dass wir mit einer modifizierten Tiefensuche in Zeit $O(|E|)$ einen solchen Artikulationsknoten v bestimmen können. Seien V_1, \dots, V_s die Knotenmengen der Zusammenhangskomponenten von $G - v$, wobei $s \geq 2$. Dann erfüllen alle Graphen $G_i := G[V_i \cup \{v\}]$, $1 \leq i \leq s$, die Annahme von Beispiel 1.62 – und können daher jeweils mit k Farben gefärbt werden. Durch eventuelles Vertauschen von Farben können wir zudem sicherstellen, dass in allen Graphen G_1, \dots, G_s der Knoten v die gleiche Farbe bekommen hat. Die Färbungen der Graphen G_i ergeben daher zusammen eine k -Färbung des Graphen G .

Coloring

Satz 1.67. Jeden 3-färbbaren Graphen $G = (V, E)$ kann man in Zeit $O(|E|)$ mit $O(\sqrt{|V|})$ Farben färben.

Satz 1.67. Jeden 3-färbbaren Graphen $G = (V, E)$ kann man in Zeit $O(|E|)$ mit $O(\sqrt{|V|})$ Farben färben.

Proof

Observation: for all $v \in V$, the induced subgraph $G[N(v)]$ can be colored using two colors and is therefore bipartite. We can color $G[N(v)]$ using BFS.

Idea: color vertices with *large degree* and their neighbors with 3 colors. If there is no vertex with *large degree* anymore, color the rest.

What is considered *large degree*? We will see: $\sqrt{|V|}$.

Satz 1.67. Jeden 3-färbbaren Graphen $G = (V, E)$ kann man in Zeit $O(|E|)$ mit $O(\sqrt{|V|})$ Farben färben.

Proof

While es gibt Knoten v , der $\geq \sqrt{|V|}$ ungefärbte Nachbarn hat:

Färbe v mit neuer Farbe und seine Nachbarn mit 2 weiteren neuen Farben.

Lösche alle gefärbten Knoten. Der Restgraph hat Maximalgrad $\Delta < \sqrt{|V|}$.

Each iteration we color **at least** $\sqrt{|V|}$ vertices, thus after **at most** $\sqrt{|V|}$ iterations we stop (since then $\sqrt{|V|} \cdot \sqrt{|V|} = |V|$ vertices are colored).

Each iteration takes 3 new colors: **at most** $3 \cdot \sqrt{|V|}$ colors.

Color the rest (after deletion) with Greedy: at most $\Delta + 1 \leq \sqrt{|V|}$ colors.

In total: $O(\sqrt{|V|})$ colors used.