Seminarvortrag - Bakers Approximationsschema für planare Graphen

Philip Whittington

RWTH Aachen

29.06.2018

Inhaltsverzeichnis

Motivation

- NP-vollständige Probleme auf planaren Graphen eignen sich für viele Anwendungsfälle
- Laufzeit ist problematisch!

Motivation

- NP-vollständige Probleme auf planaren Graphen eignen sich für viele Anwendungsfälle
- Laufzeit ist problematisch!
- Lösungen lassen sich approximieren

Motivation

- NP-vollständige Probleme auf planaren Graphen eignen sich für viele Anwendungsfälle
- Laufzeit ist problematisch!
- Lösungen lassen sich approximieren
- oder auf speziellen planaren Graphen mit besserer Laufzeit lösen

Motivation

- NP-vollständige Probleme auf planaren Graphen eignen sich für viele Anwendungsfälle
- Laufzeit ist problematisch!
- Lösungen lassen sich approximieren
- oder auf speziellen planaren Graphen mit besserer Laufzeit lösen
- Wir betrachten das Approximationsschema von Brenda S. Baker, hauptsächlich für Maximum Independent Set

Ziele

- Approximationsalgorithmus für planare Graphen mit
  - Approximationsfaktor $\frac{k}{k+1}$
  - Laufzeit $\mathcal{O}(8^k kn)$ für frei wählbares $k \in \mathbb{N}$

Ziele

- Approximationsalgorithmus für planare Graphen mit
  - Approximationsfaktor $\frac{k}{k+1}$
  - Laufzeit $\mathcal{O}(8^k kn)$ für frei wählbares $k \in \mathbb{N}$
- Exakten Algorithmus für k-außerplanare Graphen mit
  - Laufzeit $\mathcal{O}(8^k n)$,
  - also linearer Laufzeit für festes k

Ziele

- Approximationsalgorithmus für planare Graphen mit
  - Approximationsfaktor $\frac{k}{k+1}$
  - Laufzeit $\mathcal{O}(8^k kn)$ für frei wählbares $k \in \mathbb{N}$
- Exakten Algorithmus für k-außerplanare Graphen mit
  - Laufzeit $\mathcal{O}(8^k n)$,
  - also linearer Laufzeit für festes k
- Ausblick auf Anpassung für andere NP-vollständige Probleme

Definition: k-außerplanare Graphen

- Knoten, die auf dem äußeren Face liegen, sind außerplanar und werden Stufe-1-Knoten genannt

Definition: k-außerplanare Graphen

- Knoten, die auf dem äußeren Face liegen, sind außerplanar und werden Stufe-1-Knoten genannt
- Ein Kreis aus Stufe-i-Knoten ist ein Stufe-i-Face, falls er ein inneres Face im Subgraph aller Stufe-i-Knoten induziert

Definition: k-außerplanare Graphen

- Knoten, die auf dem äußeren Face liegen, sind außerplanar und werden Stufe-1-Knoten genannt
- Ein Kreis aus Stufe-i-Knoten ist ein Stufe-i-Face, falls er ein inneres Face im Subgraph aller Stufe-i-Knoten induziert
- Sei G_f der Subgraph aus allen Knoten, die innerhalb des Stufe-i-Faces f liegen, dann sind die Knoten auf dem äußeren Face von G_f Stufe-$(i+1)$-Knoten

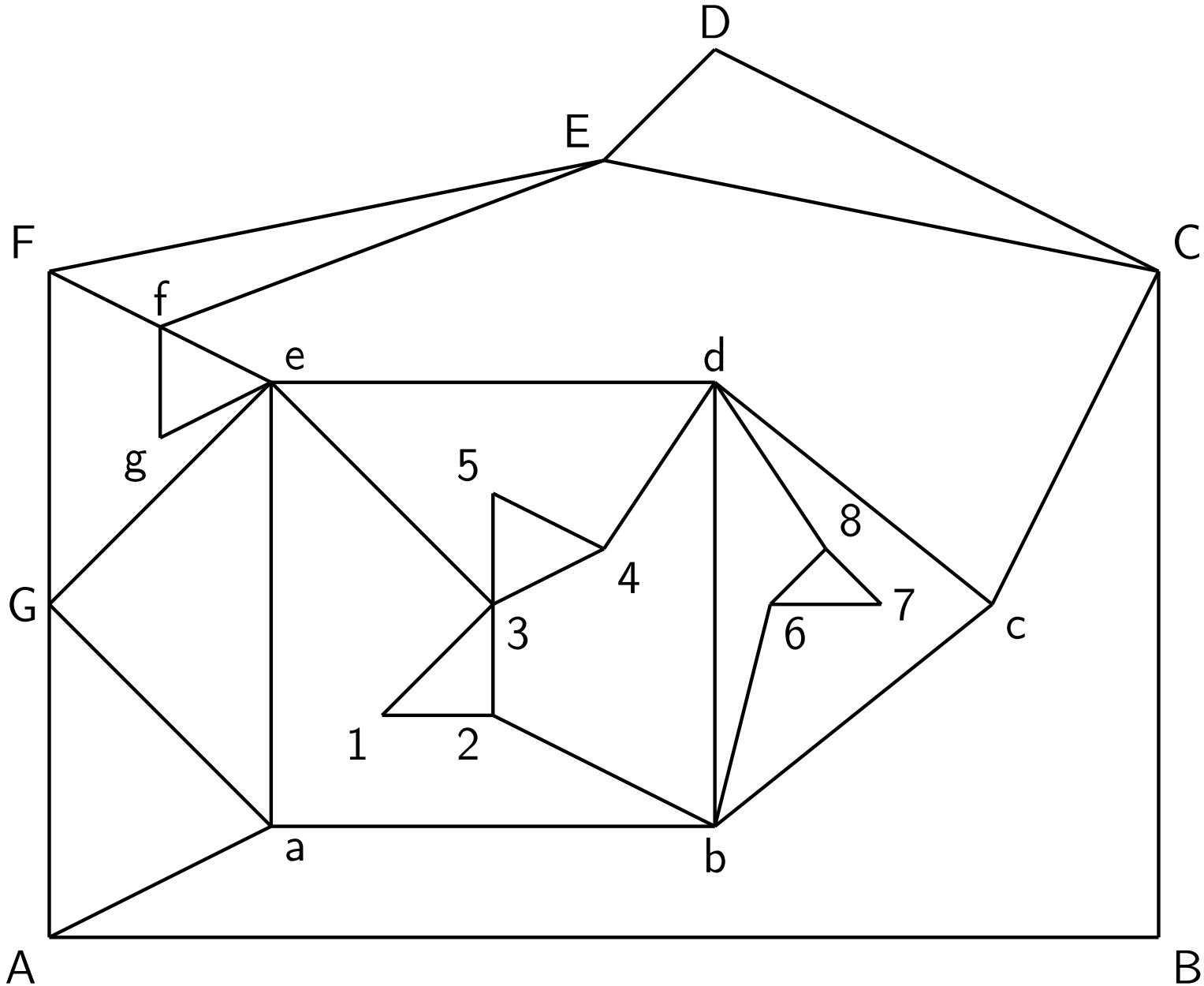
Definition: k-außerplanare Graphen

- Knoten, die auf dem äußeren Face liegen, sind außerplanar und werden Stufe-1-Knoten genannt
- Ein Kreis aus Stufe-i-Knoten ist ein Stufe-i-Face, falls er ein inneres Face im Subgraph aller Stufe-i-Knoten induziert
- Sei G_f der Subgraph aus allen Knoten, die innerhalb des Stufe-i-Faces f liegen, dann sind die Knoten auf dem äußeren Face von G_f Stufe-$(i+1)$-Knoten
- Eine planare Einbettung ist k-stufig, wenn alle Knoten höchstens Stufe-k-Knoten sind

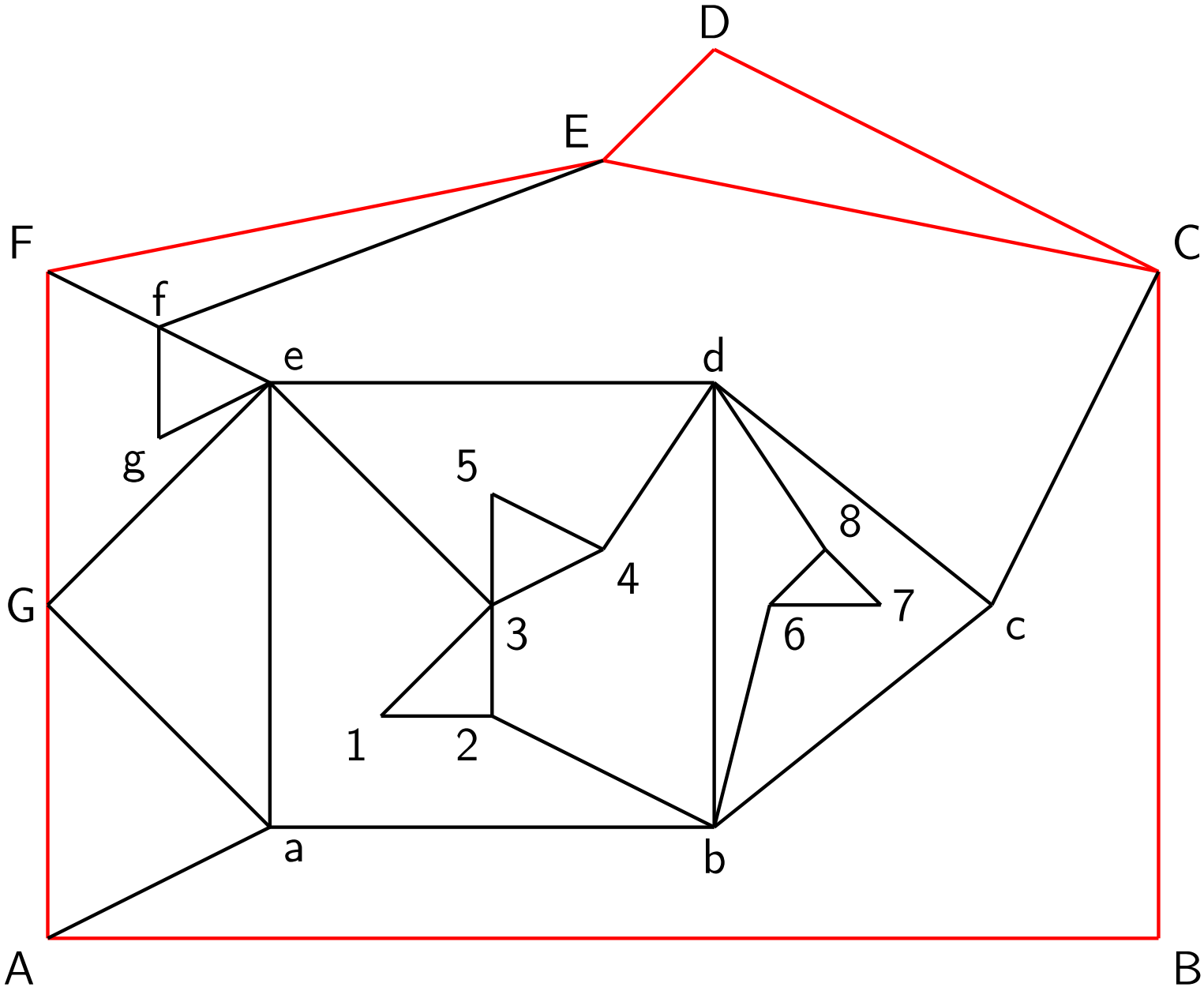
Definition: k-außerplanare Graphen

- Knoten, die auf dem äußeren Face liegen, sind außerplanar und werden Stufe-1-Knoten genannt
- Ein Kreis aus Stufe-i-Knoten ist ein Stufe-i-Face, falls er ein inneres Face im Subgraph aller Stufe-i-Knoten induziert
- Sei G_f der Subgraph aus allen Knoten, die innerhalb des Stufe-i-Faces f liegen, dann sind die Knoten auf dem äußeren Face von G_f Stufe-$(i+1)$-Knoten
- Eine planare Einbettung ist k-stufig, wenn alle Knoten höchstens Stufe-k-Knoten sind
- Ein Graph ist k-stufig, wenn er eine k-stufige planare Einbettung besitzt

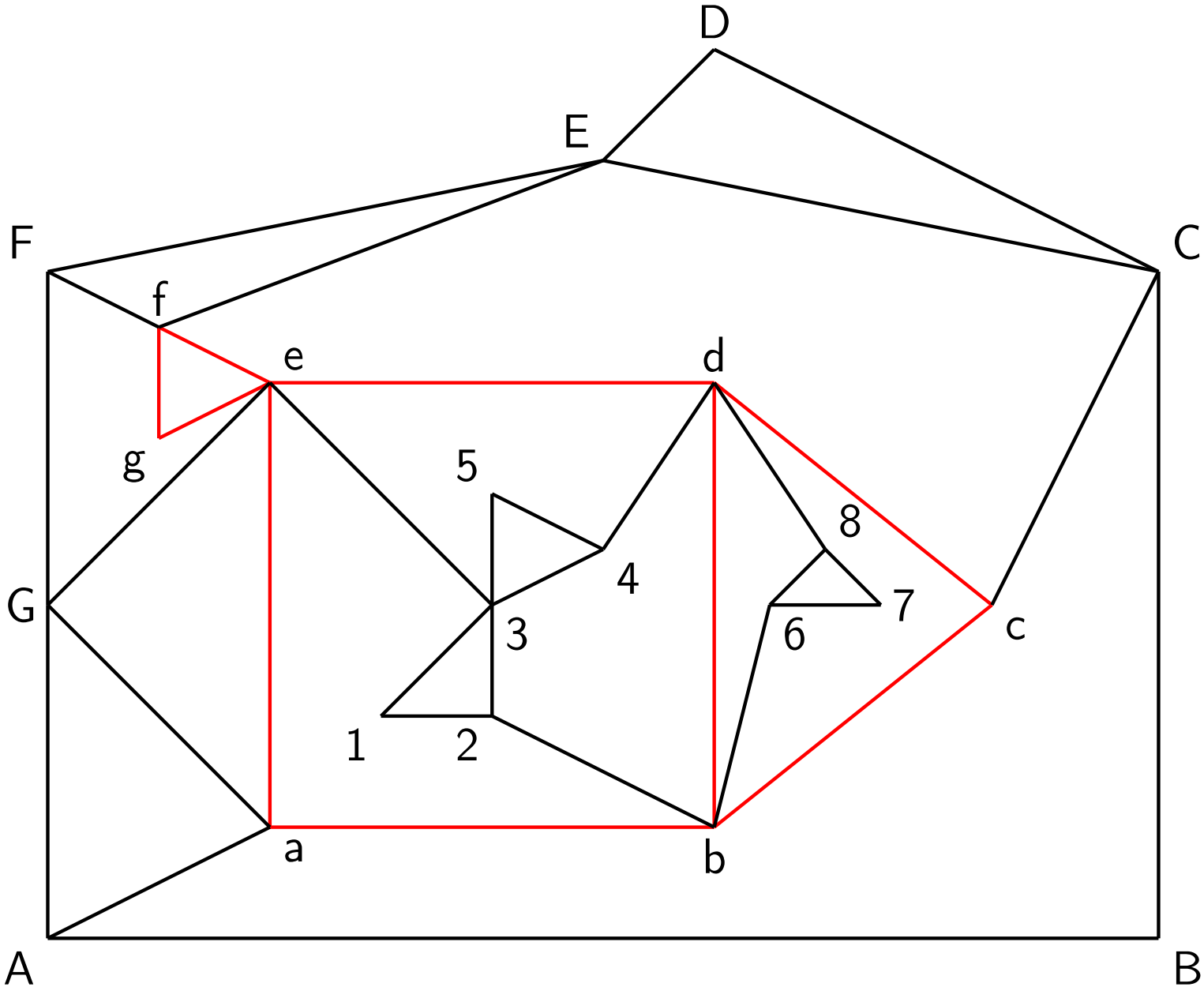
Beispiel: k-außerplanare Graphen



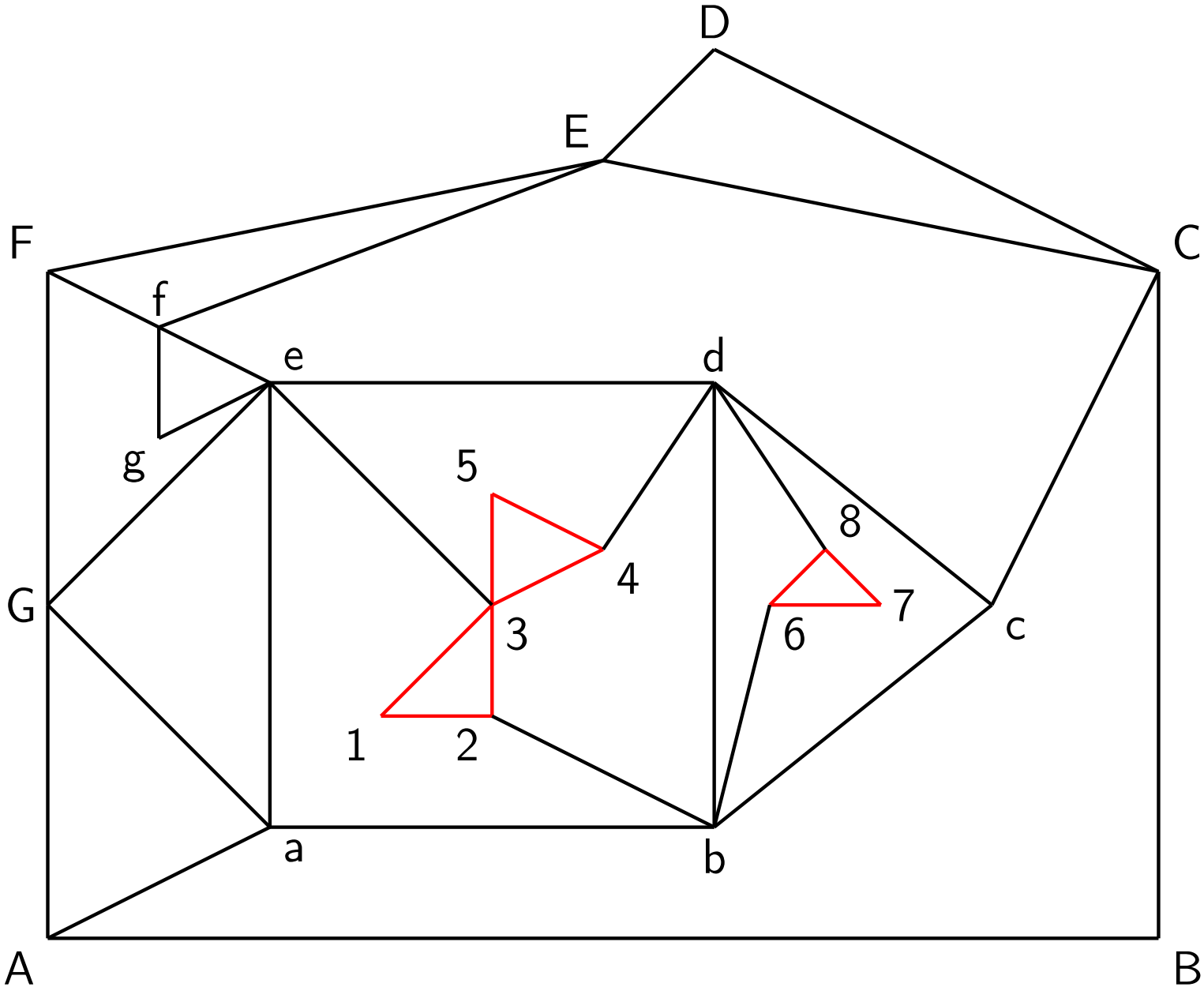
Beispiel: k-außerplanare Graphen



Beispiel: k-außerplanare Graphen



Beispiel: k-außerplanare Graphen



Theorem: Exakter Algorithmus auf k-außerplanaren Graphen

Theorem

Sei G ein Graph und $k > 0$. Gegeben eine k-außerplanare Einbettung von G, kann in $\mathcal{O}(8^k n)$ eine optimale Lösung für Maximum Independent Set ermittelt werden.

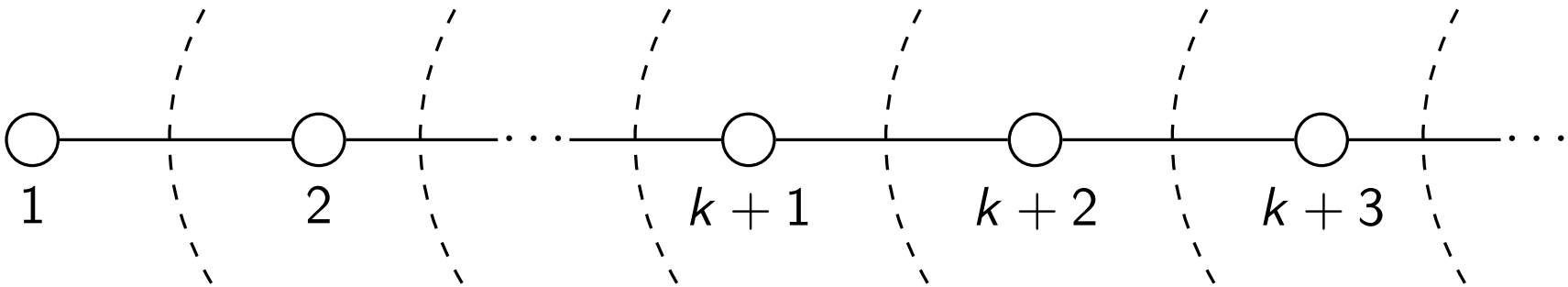
Theorem: Exakter Algorithmus auf k-außerplanaren Graphen

Theorem

Sei G ein Graph und $k > 0$. Gegeben eine k-außerplanare Einbettung von G, kann in $\mathcal{O}(8^k n)$ eine optimale Lösung für Maximum Independent Set ermittelt werden.

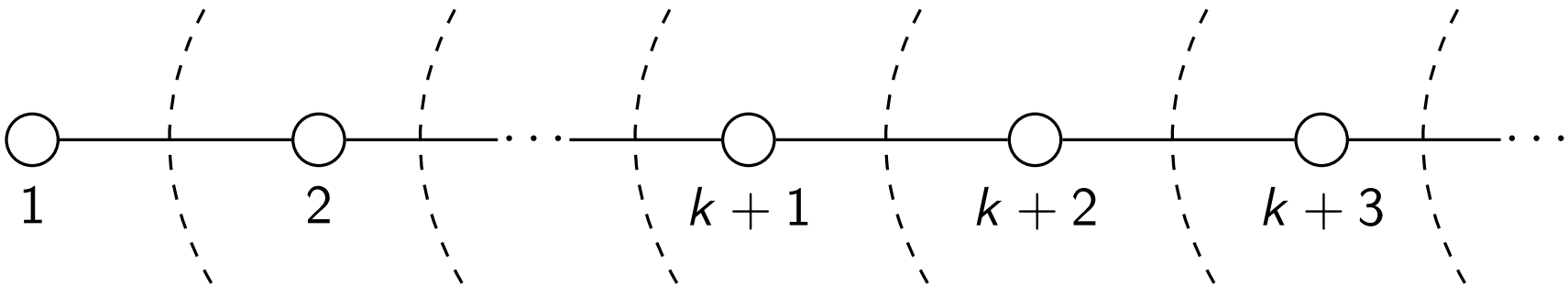
$\Longrightarrow$ Für festes k kann nach diesem Theorem in Linearzeit eine optimale Lösung für Maximum Independent Set auf k-außerplanaren Graphen gefunden werden.

Unterteilung des Graphen



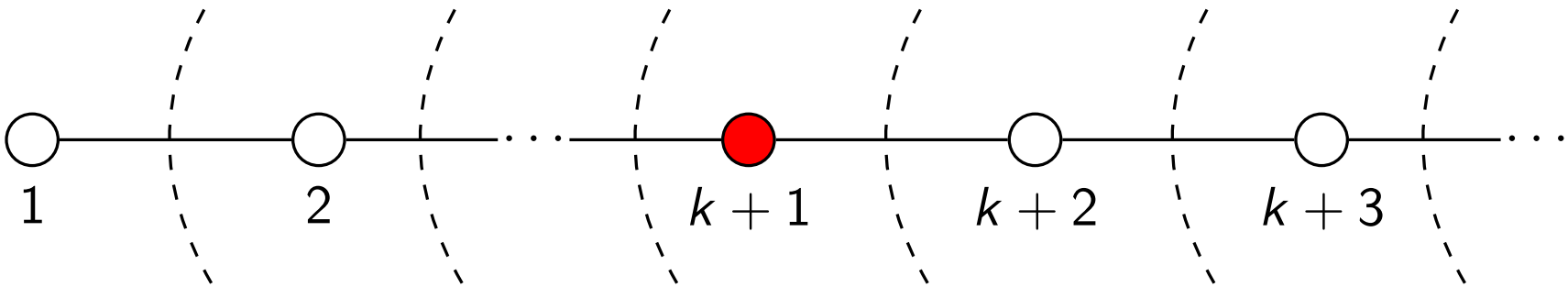
- Im Allgemeinen besitzen Graphen mehr als k Stufen.

Unterteilung des Graphen



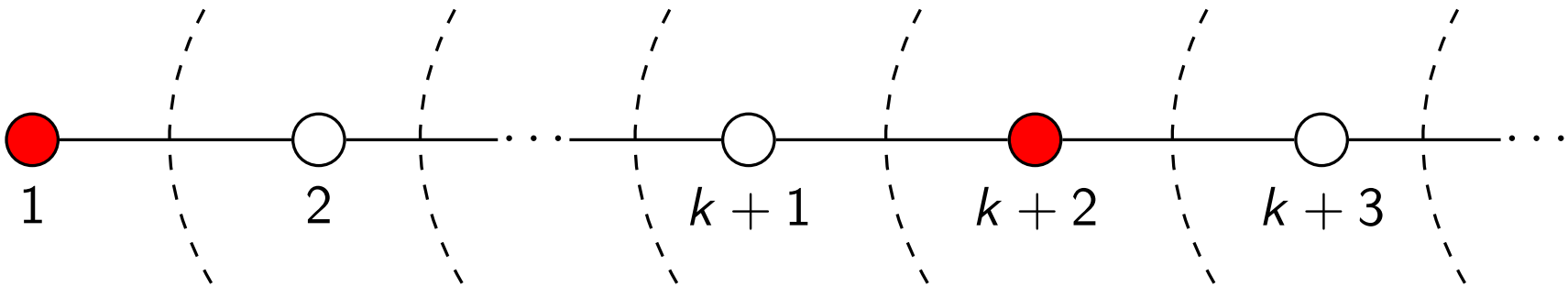
- Im Allgemeinen besitzen Graphen mehr als k Stufen.
- Wir unterteilen solche Graphen in $k+1$ Knotenmengen:
- Zwei Knoten sind in der gleichen Menge, wenn ihre Stufen modulo $k+1$ gleich sind

Unterteilung des Graphen



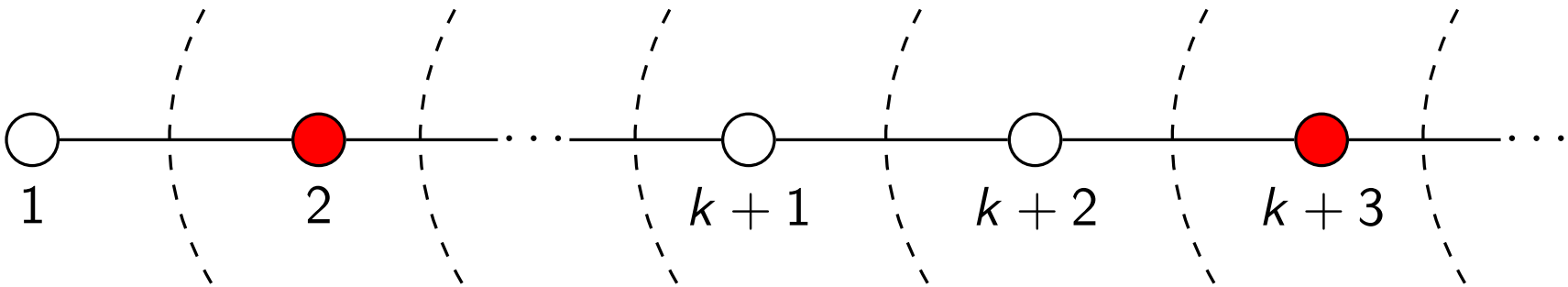
- Im Allgemeinen besitzen Graphen mehr als k Stufen.
- Wir unterteilen solche Graphen in $k+1$ Knotenmengen:
- Zwei Knoten sind in der gleichen Menge, wenn ihre Stufen modulo $k+1$ gleich sind

Unterteilung des Graphen



- Im Allgemeinen besitzen Graphen mehr als k Stufen.
- Wir unterteilen solche Graphen in $k+1$ Knotenmengen:
- Zwei Knoten sind in der gleichen Menge, wenn ihre Stufen modulo $k+1$ gleich sind

Unterteilung des Graphen



- Im Allgemeinen besitzen Graphen mehr als k Stufen.
- Wir unterteilen solche Graphen in $k+1$ Knotenmengen:
- Zwei Knoten sind in der gleichen Menge, wenn ihre Stufen modulo $k+1$ gleich sind

Approximationsalgorithmus

- Iteriere über all diese Mengen, indem man die jeweilige Menge für diese Iteration aus dem Graphen löscht, wodurch der Graph in Zusammenhangskomponenten zerfällt

Approximationsalgorithmus

- Iteriere über all diese Mengen, indem man die jeweilige Menge für diese Iteration aus dem Graphen löscht, wodurch der Graph in Zusammenhangskomponenten zerfällt
- Wende den exakten Algorithmus auf die Subgraphen an, die nun k-außerplanar sind und addiere die Lösungswerte
- Übernimm das beste Ergebnis aller Iterationen

Approximationsalgorithmus

- Iteriere über all diese Mengen, indem man die jeweilige Menge für diese Iteration aus dem Graphen löscht, wodurch der Graph in Zusammenhangskomponenten zerfällt
- Wende den exakten Algorithmus auf die Subgraphen an, die nun k-außerplanar sind und addiere die Lösungswerte
- Übernimm das beste Ergebnis aller Iterationen
- Betrachte die optimale Lösung S_{opt}
- Dann enthält eine Menge maximal $\frac{S_{opt}}{k+1}$ Knoten aus S_{opt}, somit ist die in dieser Iteration ermittelte Lösung $\frac{k}{k+1}$-optimal

Theorem: Approximationsalgorithmus

Theorem

Sei G ein planarer Graph und $k > 0$ fest. In $\mathcal{O}(8^k kn)$ kann eine Lösung für Maximum Independent Set mit Approximationsfaktor $\frac{k}{k+1}$ auf G gefunden werden.

Theorem: Approximationsalgorithmus

Theorem

Sei G ein planarer Graph und $k > 0$ fest. In $\mathcal{O}(8^k kn)$ kann eine Lösung für Maximum Independent Set mit Approximationsfaktor $\frac{k}{k+1}$ auf G gefunden werden.

$\Longrightarrow$ Für $k \in \mathcal{O}(\log(n))$ kann nach diesem Theorem in Polynomialzeit eine $\frac{k}{k+1}$-optimale Lösung für Maximum Independent Set auf planaren Graphen gefunden werden.

Konstruktion von $\overline{G}$

Wir konstruieren den Graphen $\overline{G}$, der einen Rundgang auf G repräsentiert.

1. Ersetze Brücken durch zwei Kanten

Konstruktion von $\overline{G}$

Wir konstruieren den Graphen $\overline{G}$, der einen Rundgang auf G repräsentiert.

1. Ersetze Brücken durch zwei Kanten
2. Erzeuge Knoten für jedes innere Face und jede äußere Kante

Konstruktion von $\overline{G}$

Wir konstruieren den Graphen $\overline{G}$, der einen Rundgang auf G repräsentiert.

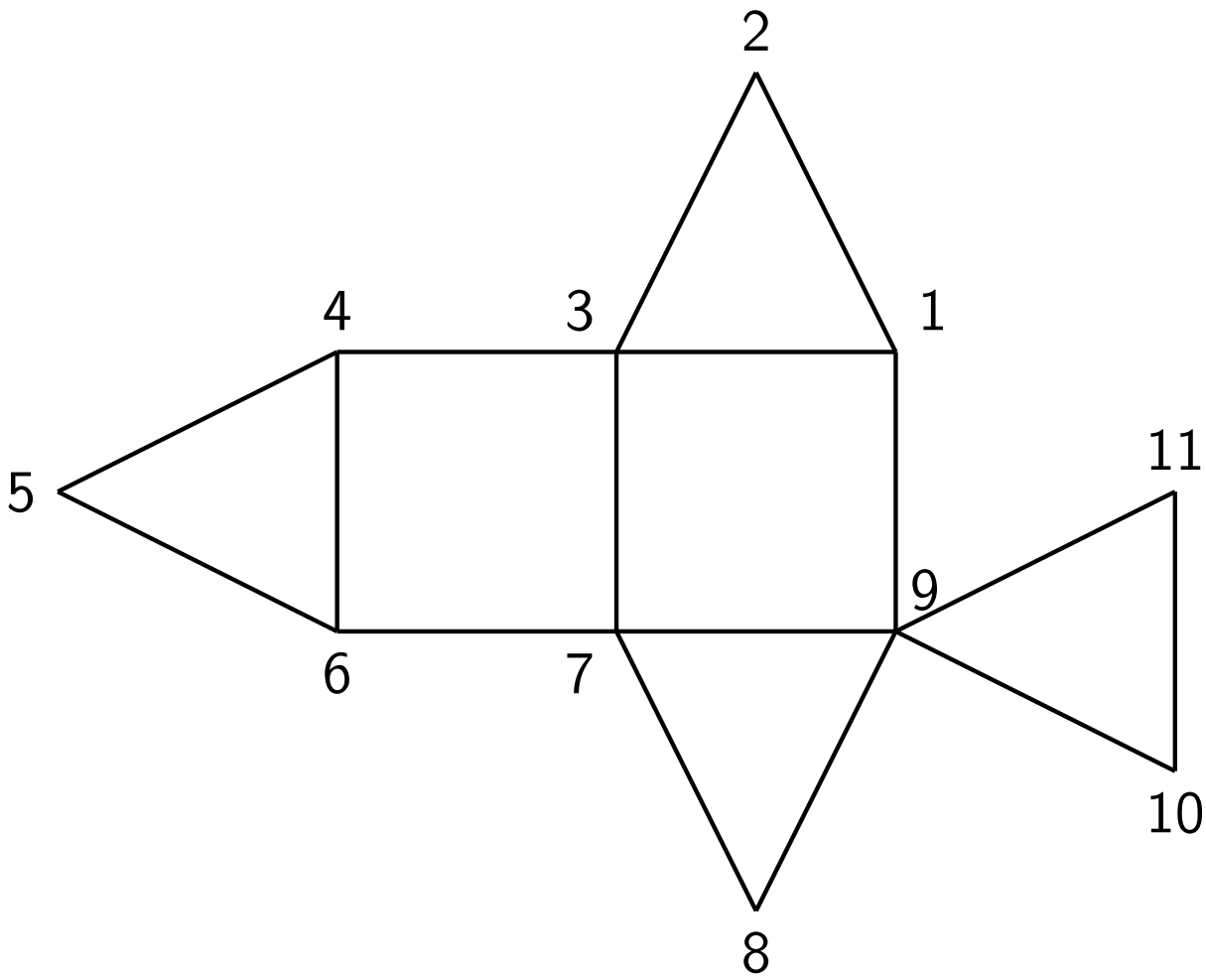
1. Ersetze Brücken durch zwei Kanten
2. Erzeuge Knoten für jedes innere Face und jede äußere Kante
3. Verbinde Knoten von Kanten und ihrem Face sowie adjazenten Faces

Konstruktion von $\overline{G}$

Wir konstruieren den Graphen $\overline{G}$, der einen Rundgang auf G repräsentiert.

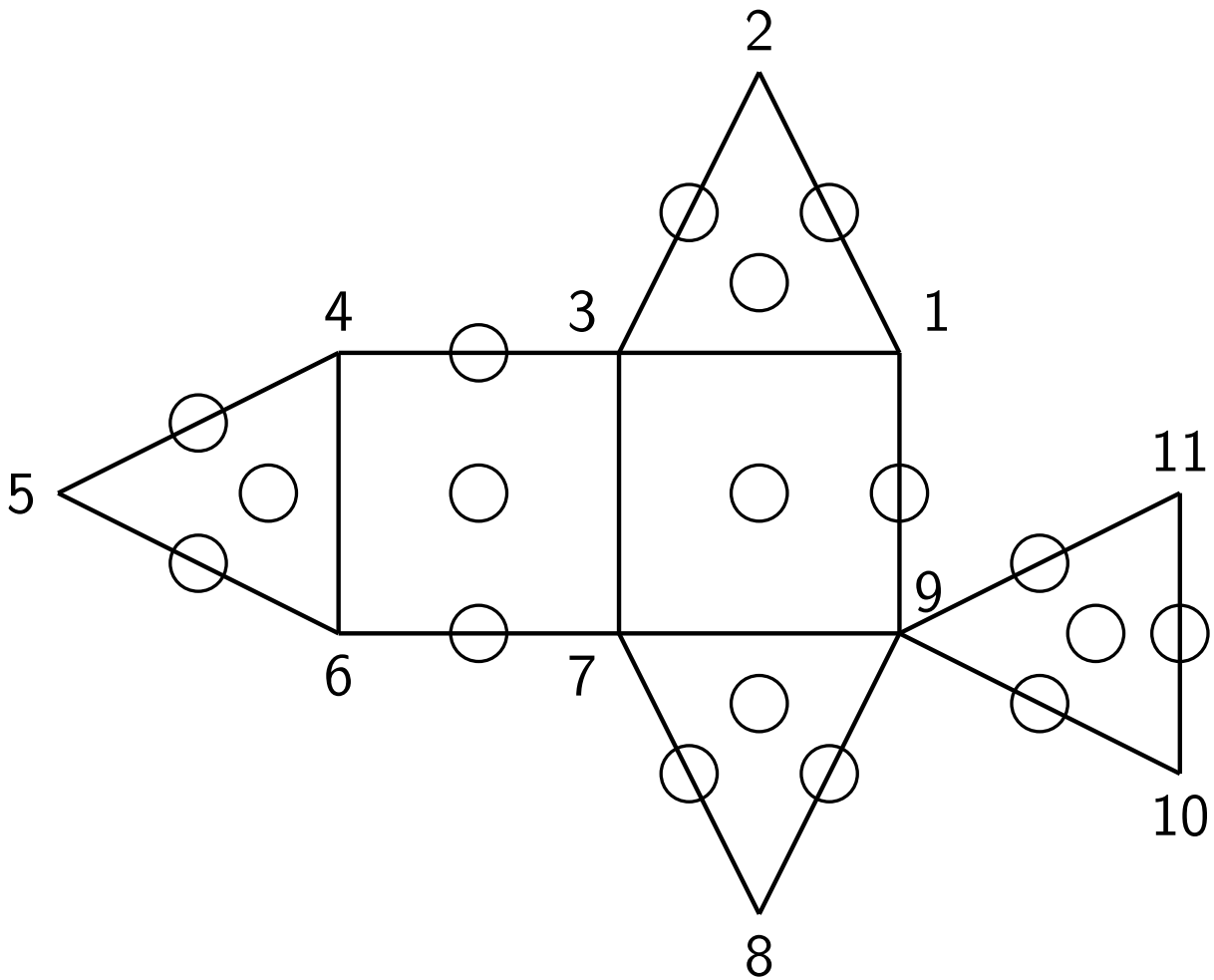
1. Ersetze Brücken durch zwei Kanten
2. Erzeuge Knoten für jedes innere Face und jede äußere Kante
3. Verbinde Knoten von Kanten und ihrem Face sowie adjazenten Faces
4. Spezialfall Schnittpunkte: Verbinde die Knoten zweier Faces, die sich am Schnittpunkt treffen

Beispiel: $\overline{G}$ in G



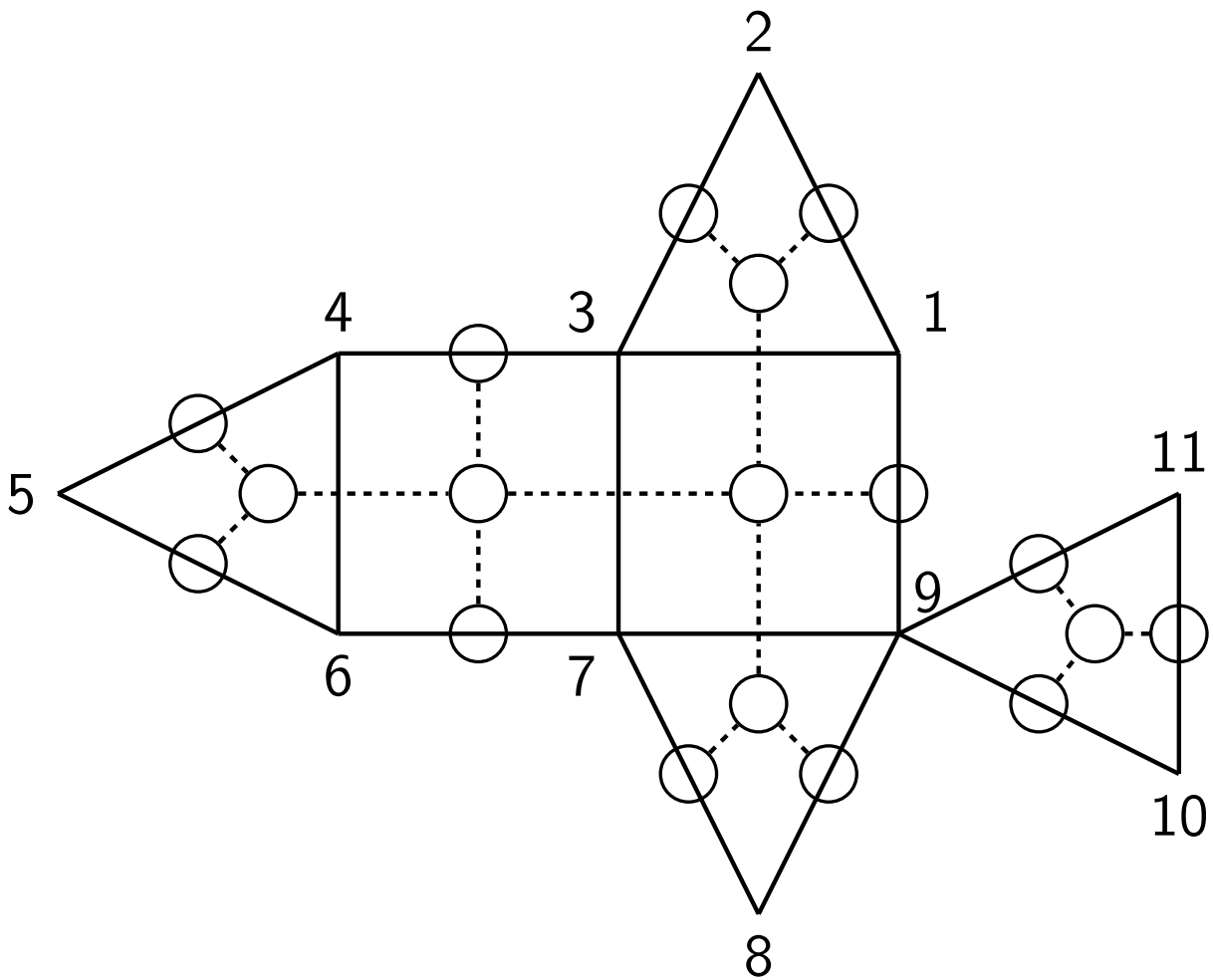
Beispiel: $\overline{G}$ in G

2. Erzeuge Knoten für jedes innere Face und jede äußere Kante



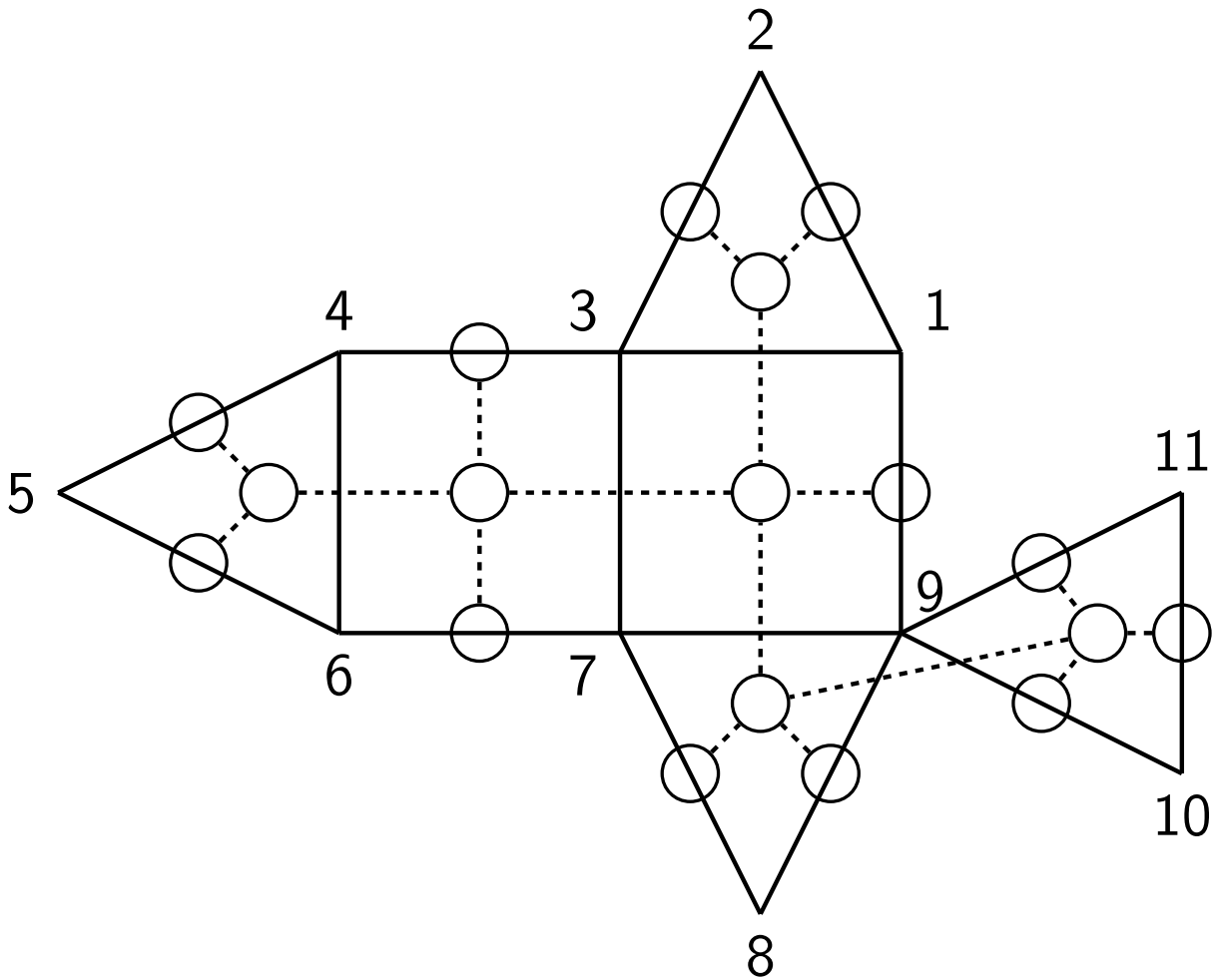
Beispiel: $\overline{G}$ in G

3. Verbinde Knoten von Kanten und ihrem Face sowie adjazenten Faces

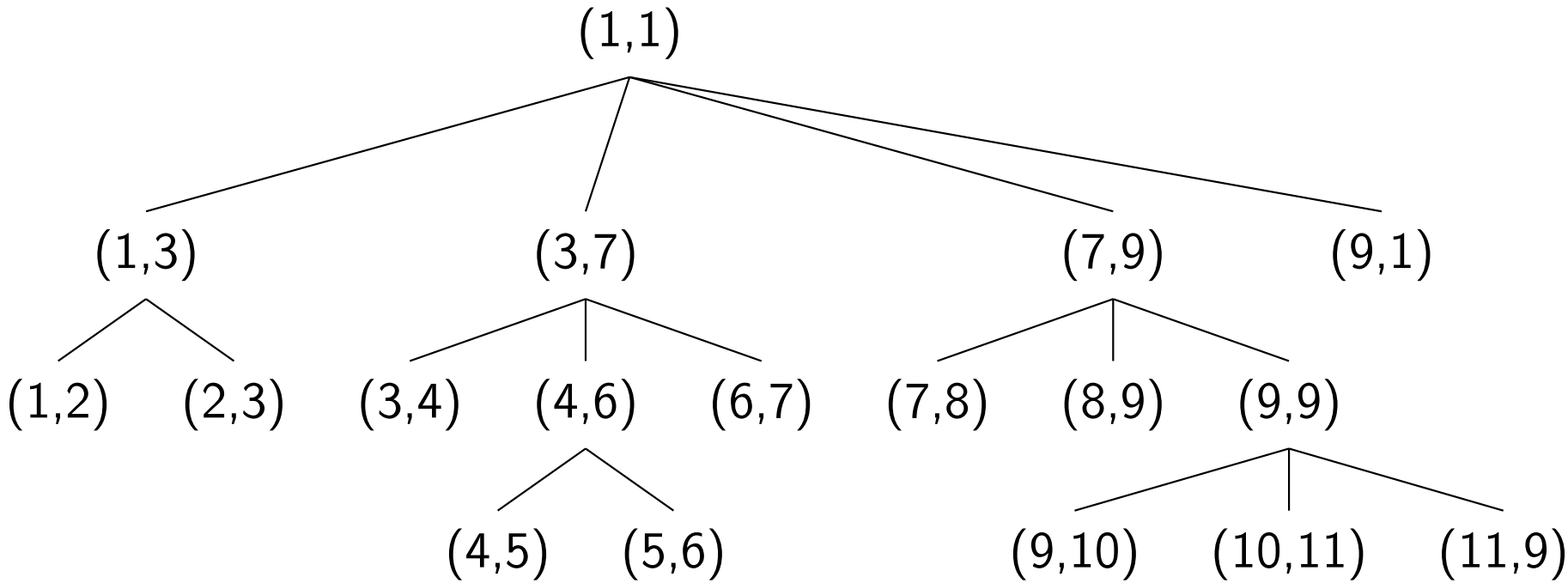


Beispiel: $\overline{G}$ in G

4. Spezialfall Schnittpunkte: Verbinde die Knoten zweier Faces, die sich am Schnittpunkt treffen



Beispiel: $\overline{G}$ separat



Dynamisches Programm auf $\overline{G}$

- Aufruf auf Baumknoten (x, y), Rückgabe: vierelementiger Vektor
- Startaufruf auf $(1, 1)$, Lösung ist Maximum der Vektoreinträge

Dynamisches Programm auf $\overline{G}$

- Aufruf auf Baumknoten (x, y), Rückgabe: vierelementiger Vektor
- Startaufruf auf $(1, 1)$, Lösung ist Maximum der Vektoreinträge
- Vektoreinträge kodieren den Wert der Lösung jenachdem, ob x und/oder y im Maximum Independent Set sind

Dynamisches Programm auf $\overline{G}$

- Aufruf auf Baumknoten (x, y), Rückgabe: vierelementiger Vektor
- Startaufruf auf $(1, 1)$, Lösung ist Maximum der Vektoreinträge
- Vektoreinträge kodieren den Wert der Lösung jenachdem, ob x und/oder y im Maximum Independent Set sind
- Es gibt immer undefinierte Bits (unterschiedlich bei $x = y$)!

Programmcode

```
1  table(v)
2    if v is a level 1 leaf with label (x,y)
3      return a table representing (x,y)
4    else T=table(u), where u is the leftmost child of v
5      for each other child c of v from left to right
6        T= merge(T,table(c))
7      return adjust(T)
```

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn diese kompatibel sind
- Kompatibel: die zweite Tabelle gehört zum Knoten (z, w), enthält daher einen fertigen Vektor für (z, w) und die erste Tabelle enthält einen fertigen Vektor für (x, z)

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn diese kompatibel sind
- Kompatibel: die zweite Tabelle gehört zum Knoten (z, w), enthält daher einen fertigen Vektor für (z, w) und die erste Tabelle enthält einen fertigen Vektor für (x, z)
- Erzeuge dann einen Vektor (x, w) mit folgender Formel:

$$V(T) + V(\mathit{table}(c)) - b_2$$

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn diese kompatibel sind
- Kompatibel: die zweite Tabelle gehört zum Knoten (z, w), enthält daher einen fertigen Vektor für (z, w) und die erste Tabelle enthält einen fertigen Vektor für (x, z)
- Erzeuge dann einen Vektor (x, w) mit folgender Formel:

$$V(T) + V(table(c)) - b_2$$

adjust:

- Passt den Vektor bezüglich undefinierter Zustände an, da am Ende der zuletzt berechnete Vektor dem aktuellen Face-Knoten entspricht, welcher wiederum eine Kante oder ein Knoten ist

Verbinden der Bäume

- Wir können die bisher gesehenen Fortschritte auch für k-stufige Graphen nutzen!
- Der Baum für die äußere Stufe wird wie gewohnt berechnet, ansonsten werden Wurzel und linkes Kind speziell gewählt

Verbinden der Bäume

- Wir können die bisher gesehenen Fortschritte auch für k-stufige Graphen nutzen!
- Der Baum für die äußere Stufe wird wie gewohnt berechnet, ansonsten werden Wurzel und linkes Kind speziell gewählt
- Sei C eine Stufe-i-Komponente, eingeschlossen von einem Face f, welches durch den Baumknoten (x, y) repräsentiert wird

Verbinden der Bäume

- Wir können die bisher gesehenen Fortschritte auch für k-stufige Graphen nutzen!
- Der Baum für die äußere Stufe wird wie gewohnt berechnet, ansonsten werden Wurzel und linkes Kind speziell gewählt
- Sei C eine Stufe-i-Komponente, eingeschlossen von einem Face f, welches durch den Baumknoten (x, y) repräsentiert wird
- Erzeuge (in Linearzeit) eine Triangulierung zwischen C und f

Verbinden der Bäume

- Wir können die bisher gesehenen Fortschritte auch für k-stufige Graphen nutzen!
- Der Baum für die äußere Stufe wird wie gewohnt berechnet, ansonsten werden Wurzel und linkes Kind speziell gewählt
- Sei C eine Stufe-i-Komponente, eingeschlossen von einem Face f, welches durch den Baumknoten (x, y) repräsentiert wird
- Erzeuge (in Linearzeit) eine Triangulierung zwischen C und f
- Für $x \neq y$ sei z, z die Wurzel von C, wobei z der einzige Knoten aus C adjazent zu x und y ist
- Für $x = y$ sei z ein beliebiger, zu x adjazenter Knoten aus C

Verbinden der Bäume

- Wir können die bisher gesehenen Fortschritte auch für k-stufige Graphen nutzen!
- Der Baum für die äußere Stufe wird wie gewohnt berechnet, ansonsten werden Wurzel und linkes Kind speziell gewählt
- Sei C eine Stufe-i-Komponente, eingeschlossen von einem Face f, welches durch den Baumknoten (x, y) repräsentiert wird
- Erzeuge (in Linearzeit) eine Triangulierung zwischen C und f
- Für $x \neq y$ sei z, z die Wurzel von C, wobei z der einzige Knoten aus C adjazent zu x und y ist
- Für $x = y$ sei z ein beliebiger, zu x adjazenter Knoten aus C
- Falls C aus mehr als einem Knoten, also z, besteht, wird u als linkes Kind gewählt, wobei (z, u) die erste Kante links von (z, x) ist

Slices

- Wie vorher wollen wir Teillösungen mittels dynamischer Programmierung verbinden!

Slices

- Wie vorher wollen wir Teillösungen mittels dynamischer Programmierung verbinden!
- Keine klare Aufteilung des Graphen gegeben $\Longrightarrow$ Aufteilung muss konstruiert werden!

Slices

- Wie vorher wollen wir Teillösungen mittels dynamischer Programmierung verbinden!
- Keine klare Aufteilung des Graphen gegeben $\Longrightarrow$ Aufteilung muss konstruiert werden!
- Konstruktion von Slices, eine für jeden Baumknoten,
- und für jede Slice einen Vektor, der den Wert der optimalen Lösung kodiert
- Zu jeder Slice $2i$ Boundaries, also 2 pro Stufe

Informelle Definition von Slices

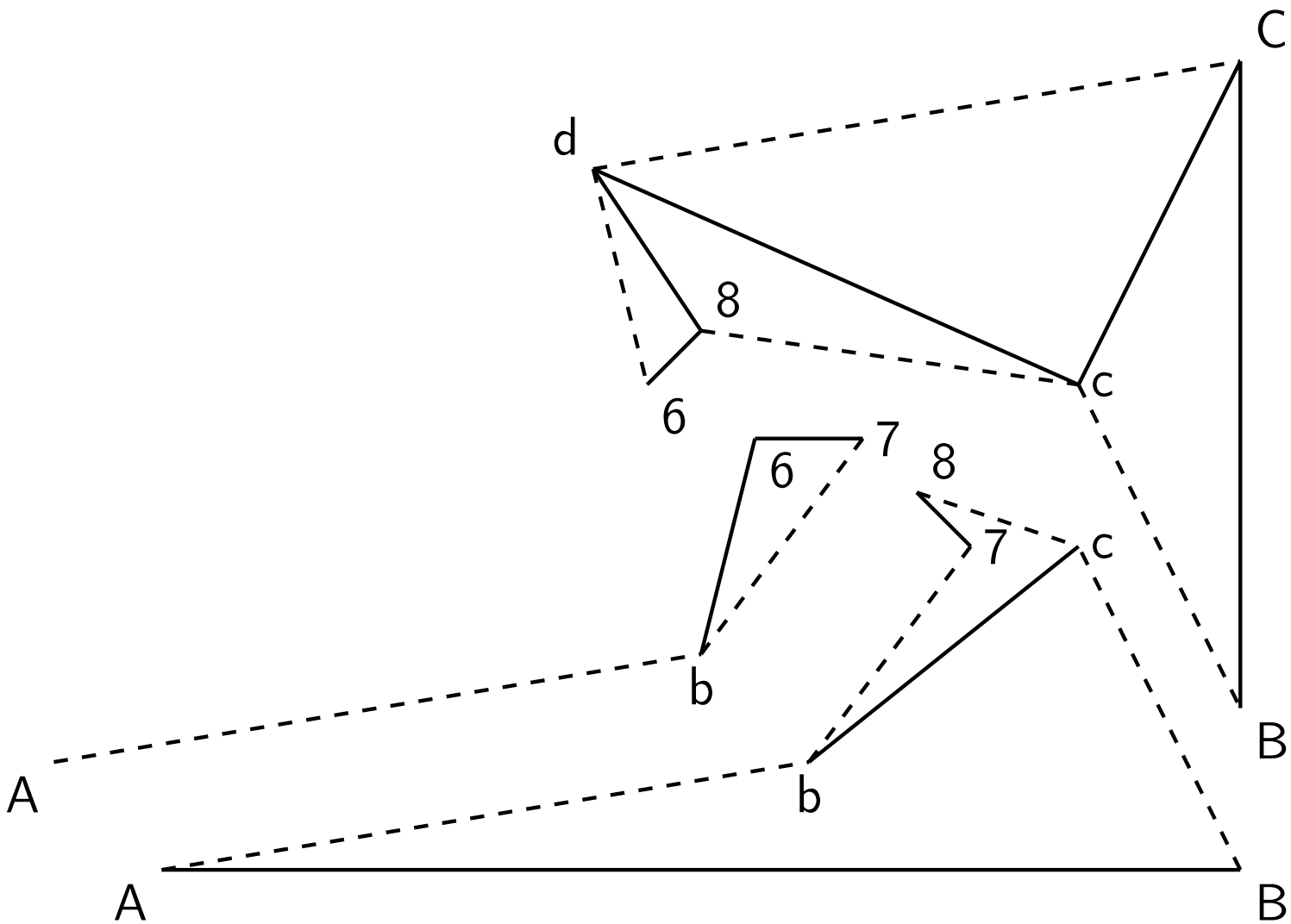
Sei v ein Baumknoten (x, y)

Informelle Definition von Slices

Sei v ein Baumknoten (x, y)

Stufe-i-Face ohne eingeschlossene Knoten	Vereinigung der Slices der Kinder von v und (x, y)
Stufe-i-Face mit eingeschlossenem Komponenten C	Slice der Wurzel des Baumes für C und (x, y)
Stufe-1-Blatt	(x, y)
Stufe-i-Blatt für $i > 1$	(x, y), alle Kanten von x und y zu anderen Stufe-i-Knoten und die Slices von geeigneten Stufe-i-Bäumen, abhängig von den Boundaries

Beispiel



Trennpunkte

- Betrachte wieder die Triangulierung zwischen C und f

Trennpunkte

- Betrachte wieder die Triangulierung zwischen C und f
- Für jede zwei Kanten $(x_1, x_2), (x_2, x_3)$ auf einem Rundgang gegen den Uhrzeigersinn auf den äußeren Kanten von C existiert ein Knoten y in f, sodass die Kanten $(x_2, x_1), (x_2, y), (x_2, x_3)$ gegen den Uhrzeigersinn um x_2 herum verlaufen

Trennpunkte

- Betrachte wieder die Triangulierung zwischen C und f
- Für jede zwei Kanten $(x_1, x_2), (x_2, x_3)$ auf einem Rundgang gegen den Uhrzeigersinn auf den äußeren Kanten von C existiert ein Knoten y in f, sodass die Kanten $(x_2, x_1), (x_2, y), (x_2, x_3)$ gegen den Uhrzeigersinn um x_2 herum verlaufen
- Diesen Knoten y nennen wir einen Trennpunkt für $(x_1, x_2), (x_2, x_3)$

Regeln für Boundaries zwischen Stufen

- Sei C eine durch das Face f eingeschlossene Komponente

Regeln für Boundaries zwischen Stufen

- Sei C eine durch das Face f eingeschlossene Komponente
- $LB(v)$ und $RB(v)$ seien Funktionen, die die Knoten aus C auf die r Kinder des Baumknotens von f abbilden

Regeln für Boundaries zwischen Stufen

- Sei C eine durch das Face f eingeschlossene Komponente
- $LB(v)$ und $RB(v)$ seien Funktionen, die die Knoten aus C auf die r Kinder des Baumknotens von f abbilden
- $LB(v_1) = 1$ und $RB(v_t) = r + 1$, wobei v_1 bzw. v_t das linkeste bzw. rechteste Blatt von $\overline{C}$ ist

Regeln für Boundaries zwischen Stufen

- Sei C eine durch das Face f eingeschlossene Komponente
- $LB(v)$ und $RB(v)$ seien Funktionen, die die Knoten aus C auf die r Kinder des Baumknotens von f abbilden
- $LB(v_1) = 1$ und $RB(v_t) = r + 1$, wobei v_1 bzw. v_t das linkeste bzw. rechteste Blatt von $\overline{C}$ ist
- $LB(v_j)$ entspricht der kleinsten Nummer $\geq LB(v_{j-1})$ eines Knotens aus f, welcher einen Trennpunkt für (v_{j-1}, v_j) und (v_j, v_{j+1}) darstellt

Regeln für Boundaries zwischen Stufen

- Sei C eine durch das Face f eingeschlossene Komponente
- $LB(v)$ und $RB(v)$ seien Funktionen, die die Knoten aus C auf die r Kinder des Baumknotens von f abbilden
- $LB(v_1) = 1$ und $RB(v_t) = r + 1$, wobei v_1 bzw. v_t das linkeste bzw. rechteste Blatt von $\overline{C}$ ist
- $LB(v_j)$ entspricht der kleinsten Nummer $\geq LB(v_{j-1})$ eines Knotens aus f, welcher einen Trennpunkt für (v_{j-1}, v_j) und (v_j, v_{j+1}) darstellt
- $RB(v_j) = LB(v_{j+1})$

Regeln für Boundaries zwischen Stufen

- Sei C eine durch das Face f eingeschlossene Komponente
- $LB(v)$ und $RB(v)$ seien Funktionen, die die Knoten aus C auf die r Kinder des Baumknotens von f abbilden
- $LB(v_1) = 1$ und $RB(v_t) = r + 1$, wobei v_1 bzw. v_t das linkeste bzw. rechteste Blatt von $\overline{C}$ ist
- $LB(v_j)$ entspricht der kleinsten Nummer $\geq LB(v_{j-1})$ eines Knotens aus f, welcher einen Trennpunkt für (v_{j-1}, v_j) und (v_j, v_{j+1}) darstellt
- $RB(v_j) = LB(v_{j+1})$
- Die inneren Knoten übernehmen die LB ihres linken und die RB ihres rechten Blattes

Formale Definition für Stufe-i-Blätter

- Sei v ein Stufe-i-Blatt, $i > 1$
- Sei f das v umschließende Face, f habe Kinder u_j, $1 \leq j \leq t$ und u_j repräsentiert (z_j, z_{j+1})

Formale Definition für Stufe-i-Blätter

- Sei v ein Stufe-i-Blatt, $i > 1$
- Sei f das v umschließende Face, f habe Kinder u_j, $1 \leq j \leq t$ und u_j repräsentiert (z_j, z_{j+1})
- Falls $LB(v) \neq RB(v)$, dann besteht die Slice für v aus (x, y), allen Kanten von x oder y zu z_j und den Slices aller u_j für alle $LB(v) \leq z_j \leq RB(v)$

Formale Definition für Stufe-i-Blätter

- Sei v ein Stufe-i-Blatt, $i > 1$
- Sei f das v umschließende Face, f habe Kinder u_j, $1 \leq j \leq t$ und u_j repräsentiert (z_j, z_{j+1})
- Falls $LB(v) \neq RB(v)$, dann besteht die Slice für v aus (x, y), allen Kanten von x oder y zu z_j und den Slices aller u_j für alle $LB(v) \leq z_j \leq RB(v)$
- Falls $LB(v) = RB(v) = m$, dann besteht die Slice für v aus (x, y), allen Kanten von x oder y zu z_m und der linken Boundary der Slice für u_m sowie allen Kanten zwischen Knoten dieser Boundary

Ermitteln der optimalen Lösung

- Wir haben gesehen, wie der Graph in Slices aufgeteilt wird
- Für jede Slice muss nun der Wert der optimalen Lösung ermittelt werden

Ermitteln der optimalen Lösung

- Wir haben gesehen, wie der Graph in Slices aufgeteilt wird
- Für jede Slice muss nun der Wert der optimalen Lösung ermittelt werden
- Jede Slice hat $2i$ Boundaries $\Longrightarrow$ 2^{2i} mögliche Kombinationen

Ermitteln der optimalen Lösung

- Wir haben gesehen, wie der Graph in Slices aufgeteilt wird
- Für jede Slice muss nun der Wert der optimalen Lösung ermittelt werden
- Jede Slice hat $2i$ Boundaries $\implies$ 2^{2i} mögliche Kombinationen
- Wieder werden Vektoren für Slices, welche eine Boundary teilen, zusammengefügt
- Die Slices werden also implizit berechnet, hauptsächlich werden die Vektoren betrachtet

Das dynamische Programm

- Aufruf auf Wurzel des Stufe-1-Baumes
- Maximum des Ergebnisvektors ist Wert optimaler Lösung

Das dynamische Programm

- Aufruf auf Wurzel des Stufe-1-Baumes
- Maximum des Ergebnisvektors ist Wert optimaler Lösung
- Rekursiver Aufruf auf allen Baumknoten
- Verfahren ist unterschiedlich je nach Typ des Baumknotens

Das dynamische Programm

- Aufruf auf Wurzel des Stufe-1-Baumes
- Maximum des Ergebnisvektors ist Wert optimaler Lösung
- Rekursiver Aufruf auf allen Baumknoten
- Verfahren ist unterschiedlich je nach Typ des Baumknotens

Stufe-i-Face ohne eingeschlossene Knoten	*merge* der Tabellen aller Kinder, gefolgt von *adjust*
Stufe-i-Face mit eingeschlossenem Komponenten C	Rekursiver Aufruf auf der Wurzel von $\overline{C}$, dann *contract*, um richtige Anzahl Boundaries zu erhalten, und *adjust*
Stufe-1-Blatt	Tabelle für (x, y)

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn ihre Slices eine gemeinsame Boundary haben
- Nehme also Slices mit Boundary $\overline{u}, \overline{z}$ und $\overline{z}, \overline{v}$

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn ihre Slices eine gemeinsame Boundary haben
- Nehme also Slices mit Boundary $\overline{u}, \overline{z}$ und $\overline{z}, \overline{v}$
- Für jede mögliche Boundary $\overline{u}, \overline{v}$ wird das Maximum über allen $\overline{z}$ für $wert(\overline{u}, \overline{z})) + wert(\overline{z}, \overline{v})) - |\overline{z}|^1$ genommen

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn ihre Slices eine gemeinsame Boundary haben
- Nehme also Slices mit Boundary $\overline{u}, \overline{z}$ und $\overline{z}, \overline{v}$
- Für jede mögliche Boundary $\overline{u}, \overline{v}$ wird das Maximum über allen $\overline{z}$ für $\mathit{wert}(\overline{u}, \overline{z})) + \mathit{wert}(\overline{z}, \overline{v})) - |\overline{z}|^1$ genommen
- Laufzeit in $\mathcal{O}(8^k)$, da auf Stufe k für 2^{2k} verschiedene Boundaries $\overline{u}, \overline{v}$ das Maximum über 2^k verschiedenen Möglichkeiten für $\overline{z}$ genommen wird

merge und *adjust*

merge:

- Fügt zwei Tabellen zusammen, wenn ihre Slices eine gemeinsame Boundary haben
- Nehme also Slices mit Boundary $\overline{u}, \overline{z}$ und $\overline{z}, \overline{v}$
- Für jede mögliche Boundary $\overline{u}, \overline{v}$ wird das Maximum über allen $\overline{z}$ für $\mathit{wert}(\overline{u}, \overline{z})) + \mathit{wert}(\overline{z}, \overline{v})) - |\overline{z}|^1$ genommen
- Laufzeit in $\mathcal{O}(8^k)$, da auf Stufe k für 2^{2k} verschiedene Boundaries $\overline{u}, \overline{v}$ das Maximum über 2^k verschiedenen Möglichkeiten für $\overline{z}$ genommen wird

adjust:

- Betrachtet die beiden Knoten von höchster Stufe in den Boundaries der durch T repräsentierten Slice und setzt dementsprechend Werte aus T undefiniert

contract

- Neue Operation, um Boundaries ohne *merge* zu verkürzen

contract

- Neue Operation, um Boundaries ohne *merge* zu verkürzen
- Benötigt für den Übergang zwischen Stufe-i-Graphen C und seinem einschließenden Face f

contract

- Neue Operation, um Boundaries ohne *merge* zu verkürzen
- Benötigt für den Übergang zwischen Stufe-i-Graphen C und seinem einschließenden Face f
- Sei (z, z) die Wurzel von $\overline{C}$ und damit der Teil der Boundary, der entfernt werden soll

contract

- Neue Operation, um Boundaries ohne *merge* zu verkürzen
- Benötigt für den Übergang zwischen Stufe-i-Graphen C und seinem einschließenden Face f
- Sei (z, z) die Wurzel von $\overline{C}$ und damit der Teil der Boundary, der entfernt werden soll
- Für jede Kombination der restlichen Boundary-Knoten hat die aktuelle Tabelle zwei Werte, jenachdem ob z Teil des Maximum Independent Sets ist

contract

- Neue Operation, um Boundaries ohne *merge* zu verkürzen
- Benötigt für den Übergang zwischen Stufe-i-Graphen C und seinem einschließenden Face f
- Sei (z, z) die Wurzel von $\overline{C}$ und damit der Teil der Boundary, der entfernt werden soll
- Für jede Kombination der restlichen Boundary-Knoten hat die aktuelle Tabelle zwei Werte, jenachdem ob z Teil des Maximum Independent Sets ist
- Nehme das Maximum dieser Werte als neuen Eintrag für diese Kombination an Boundary-Knoten

Programm für Stufe-i-Blätter

- Ermittle $LB(v)$ und $RB(v)$ und damit, welche Kinder des Baumknotens des einschließenden Faces mit in die Slice aufgenommen werden müssen

Programm für Stufe-i-Blätter

- Ermittle $LB(v)$ und $RB(v)$ und damit, welche Kinder des Baumknotens des einschließenden Faces mit in die Slice aufgenommen werden müssen
- Erzeuge einen Startgraphen per *create*, welcher nur (x, y) und einen Pfad durch die Stufen, also i Boundaries enthält

Programm für Stufe-i-Blätter

- Ermittle $LB(v)$ und $RB(v)$ und damit, welche Kinder des Baumknotens des einschließenden Faces mit in die Slice aufgenommen werden müssen
- Erzeuge einen Startgraphen per *create*, welcher nur (x, y) und einen Pfad durch die Stufen, also i Boundaries enthält
- Erweitere mithilfe von *extend* die Kinder links dieses Pfades um x und *merge* diese
- Analog: Erweitere mithilfe von *extend* die Kinder rechts dieses Pfades um y und *merge* diese

create

create:

- Wird für von einem Face f eingeschlossene Blätter aufgerufen
- Schafft den Zusammenhang zwischen den Faces der unterliegenden Stufe

create

create:

- Wird für von einem Face f eingeschlossene Blätter aufgerufen
- Schafft den Zusammenhang zwischen den Faces der unterliegenden Stufe
- Erzeugt eine Tabelle für den Graphen bestehend aus (x, y) und den Boundaries eines Kindes des Knotens für f

create

create:

- Wird für von einem Face f eingeschlossene Blätter aufgerufen
- Schafft den Zusammenhang zwischen den Faces der unterliegenden Stufe
- Erzeugt eine Tabelle für den Graphen bestehend aus (x, y) und den Boundaries eines Kindes des Knotens für f
- Wahl des Knotens ist abhängig von $LB(v)$ bzw. $RB(v)$ und davon, ob y adjazent zu einem der Kinder des Knotens von f ist

create

create:

- Wird für von einem Face f eingeschlossene Blätter aufgerufen
- Schafft den Zusammenhang zwischen den Faces der unterliegenden Stufe
- Erzeugt eine Tabelle für den Graphen bestehend aus (x, y) und den Boundaries eines Kindes des Knotens für f
- Wahl des Knotens ist abhängig von $LB(v)$ bzw. $RB(v)$ und davon, ob y adjazent zu einem der Kinder des Knotens von f ist
- Dadurch wird sichergestellt, dass die Boundary über einen Schnittpunkt verläuft und keine Kante die Boundary kreuzt

extend

extend:

- Fügt einen Knoten x zu einer Tabelle T hinzu

extend

extend:

- Fügt einen Knoten x zu einer Tabelle T hinzu
- x und die Boundary der Slice bezüglich T ergeben die neue Boundary

extend

extend:

- Fügt einen Knoten x zu einer Tabelle T hinzu
- x und die Boundary der Slice bezüglich T ergeben die neue Boundary
- Für jeden Eintrag in T, erzeuge einen gleichwertigen Eintrag, der dafür steht, dass x nicht Teil des Sets ist

extend

extend:

- Fügt einen Knoten x zu einer Tabelle T hinzu
- x und die Boundary der Slice bezüglich T ergeben die neue Boundary
- Für jeden Eintrag in T, erzeuge einen gleichwertigen Eintrag, der dafür steht, dass x nicht Teil des Sets ist
- Außerdem erzeuge einen Eintrag für x Teil des Sets mit einem um 1 inkrementierten oder undefinierten Wert, jenachdem ob x zu einem Boundary-Knoten adjazent ist

Laufzeit

- Konstruktion der Bäume und Berechnung der Boundaries geschieht in linearer Zeit

Laufzeit

- Konstruktion der Bäume und Berechnung der Boundaries geschieht in linearer Zeit
- Jeder Baumknoten wird maximal einmal rekursiv aufgerufen, und die Anzahl Baumknoten entspricht der Anzahl Kanten im Ursprungsgraphen
- Anzahl Kanten bei planaren Graphen linear in der Anzahl der Knoten

  $\Longrightarrow$ linear viele Aufrufe!

Laufzeit

- Konstruktion der Bäume und Berechnung der Boundaries geschieht in linearer Zeit
- Jeder Baumknoten wird maximal einmal rekursiv aufgerufen, und die Anzahl Baumknoten entspricht der Anzahl Kanten im Ursprungsgraphen
- Anzahl Kanten bei planaren Graphen linear in der Anzahl der Knoten

  $\Longrightarrow$ linear viele Aufrufe!
- *merge* dominiert mit Laufzeit $\mathcal{O}(8^k)$

Laufzeit

- Konstruktion der Bäume und Berechnung der Boundaries geschieht in linearer Zeit
- Jeder Baumknoten wird maximal einmal rekursiv aufgerufen, und die Anzahl Baumknoten entspricht der Anzahl Kanten im Ursprungsgraphen
- Anzahl Kanten bei planaren Graphen linear in der Anzahl der Knoten

  $\Longrightarrow$ linear viele Aufrufe!
- *merge* dominiert mit Laufzeit $\mathcal{O}(8^k)$
- Alle Operationen werden maximal einmal pro rekursivem Aufruf ausgeführt!

Laufzeit

- Konstruktion der Bäume und Berechnung der Boundaries geschieht in linearer Zeit
- Jeder Baumknoten wird maximal einmal rekursiv aufgerufen, und die Anzahl Baumknoten entspricht der Anzahl Kanten im Ursprungsgraphen
- Anzahl Kanten bei planaren Graphen linear in der Anzahl der Knoten
  $\Longrightarrow$ linear viele Aufrufe!
- *merge* dominiert mit Laufzeit $\mathcal{O}(8^k)$
- Alle Operationen werden maximal einmal pro rekursivem Aufruf ausgeführt!

$\Longrightarrow$ Laufzeit des gesamten Algorithmus: $\mathcal{O}(8^k n)$!

Approximierbare Probleme

Minimum Vertex Cover:

- Minimierungs- statt Maximierungsproblem

Approximierbare Probleme

Minimum Vertex Cover:

- Minimierungs- statt Maximierungsproblem
- Unterschied nur in der Buchhaltung

Approximierbare Probleme

Minimum Vertex Cover:

- Minimierungs- statt Maximierungsproblem
- Unterschied nur in der Buchhaltung
- Vektoreinträge enthalten minimal benötige Anzahl Knoten

Approximierbare Probleme

Minimum Vertex Cover:

- Minimierungs- statt Maximierungsproblem
- Unterschied nur in der Buchhaltung
- Vektoreinträge enthalten minimal benötige Anzahl Knoten

Minimum Edge Dominating Set:

- Vektoreinträge enthalten mimimal benötigte Anzahl Kanten

Approximierbare Probleme

Minimum Vertex Cover:

- Minimierungs- statt Maximierungsproblem
- Unterschied nur in der Buchhaltung
- Vektoreinträge enthalten minimal benötige Anzahl Knoten

Minimum Edge Dominating Set:

- Vektoreinträge enthalten mimimal benötigte Anzahl Kanten
- Abhängig davon, welche Boundary-Knoten Endpunkte von Kanten des Dominating Set sind

Approximierbare Probleme

Minimum Vertex Cover:

- Minimierungs- statt Maximierungsproblem
- Unterschied nur in der Buchhaltung
- Vektoreinträge enthalten minimal benötige Anzahl Knoten

Minimum Edge Dominating Set:

- Vektoreinträge enthalten mimimal benötigte Anzahl Kanten
- Abhängig davon, welche Boundary-Knoten Endpunkte von Kanten des Dominating Set sind

Der Approximationsalgorithmus und die Aufteilung in Slices funktioniert hier komplett gleich!

3-Coloring

- Keine Approximation möglich, da Knoten nicht vernachlässigt werden können!

3-Coloring

- Keine Approximation möglich, da Knoten nicht vernachlässigt werden können!
- Auch hier: Unterschied fast nur in der Buchhaltung

3-Coloring

- Keine Approximation möglich, da Knoten nicht vernachlässigt werden können!
- Auch hier: Unterschied fast nur in der Buchhaltung
- Farbe der Boundary-Knoten wird gespeichert, daher 3^{2i} Einträge pro Tabelle

3-Coloring

- Keine Approximation möglich, da Knoten nicht vernachlässigt werden können!
- Auch hier: Unterschied fast nur in der Buchhaltung
- Farbe der Boundary-Knoten wird gespeichert, daher 3^{2i} Einträge pro Tabelle
- *merge* hat dementsprechend eine Laufzeit von $\mathcal{O}(3^{3k})$