

Effizienter Planaritätstest

Vorlesung am 9.07.2014

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Satz

Gegebenen einen Graphen $G = (V, E)$ mit n Kanten und m Knoten, kann in $O(n + m)$ Zeit getestet werden, ob G planar ist.

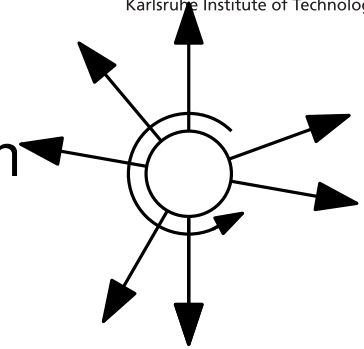
Ist dies der Fall, kann in derselben Zeit eine planare Einbettung berechnet werden. Anderenfalls kann in derselben Zeit K_5 oder $K_{3,3}$ als Minor gefunden werden.

Wir zeigen polynomielle Laufzeit.

Kantenordnungen und Einbettungen

Einbettung wird kodiert als Kantenordnung um jeden Knoten

↪ **Rotations-Schema**






Beschreibt ein gegebenes Rotationsschema eine planare Einbettung?

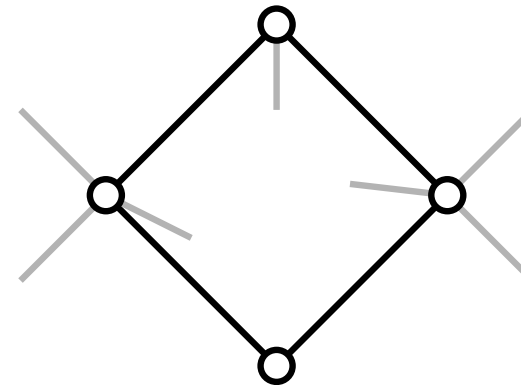
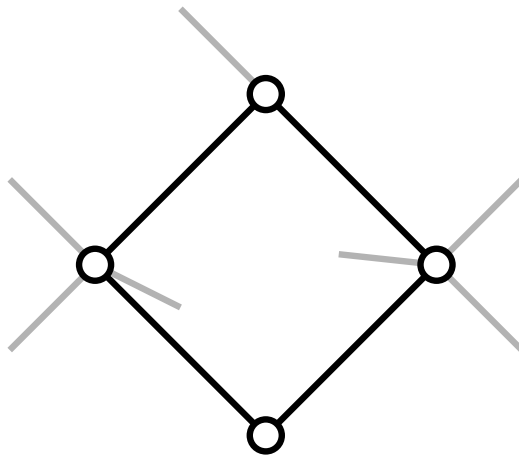
Wie kann man das testen?

Planaritätstest, 1. Ansatz

Füge Knoten schrittweise hinzu, drei Arten von Kanten:




- eingebettet 
- halb eingebettet 
- nicht eingebettet 

Idee: Speichere alle möglichen Einbettungen (halb) eingebetteten Kanten.

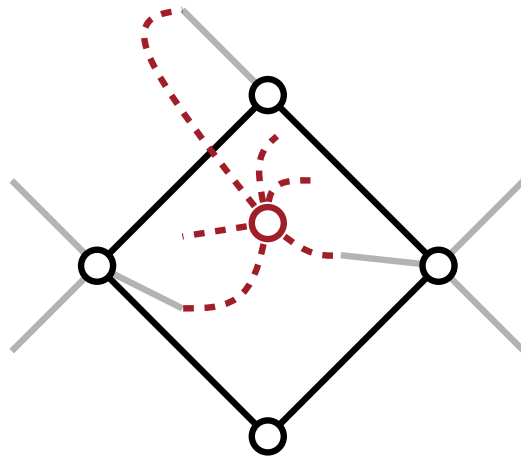


Planaritätstest, 1. Ansatz

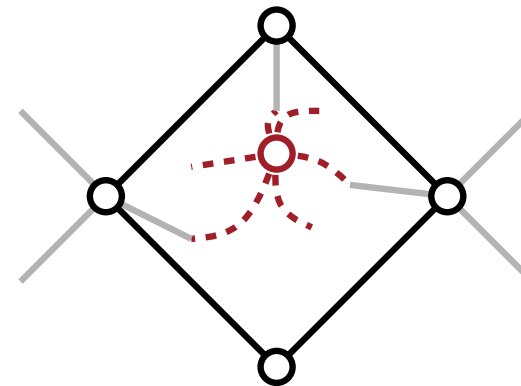
Füge Knoten schrittweise hinzu, drei Arten von Kanten:

- eingebettet 
- halb eingebettet 
- nicht eingebettet 

Idee: Speichere alle möglichen Einbettungen (halb) eingebetteten Kanten.






Erweiterung nicht möglich.



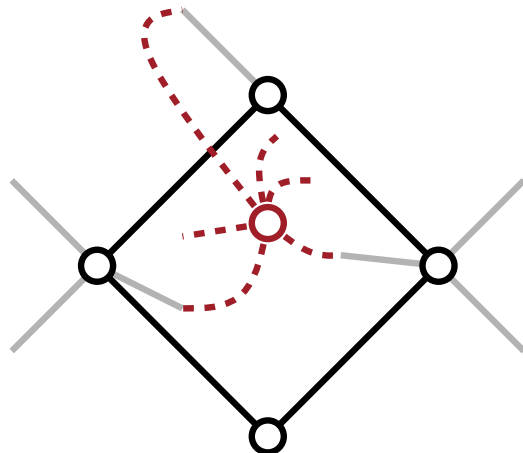
Erweiterung möglich, erzeuge alle Möglichkeiten.

Planaritätstest, 1. Ansatz

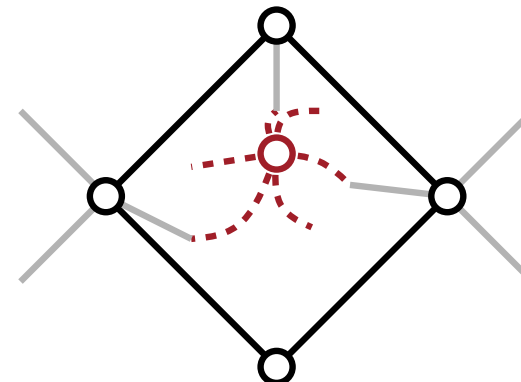
Füge Knoten schrittweise hinzu, drei Arten von Kanten:

- eingebettet 
- halb eingebettet 
- nicht eingebettet 

Idee: Speichere alle möglichen Einbettungen (halb) eingebetteten Kanten.



Erweiterung nicht möglich.



Erweiterung möglich, erzeuge alle Möglichkeiten.

⇒ Exponentielle Laufzeit & Speicher



Planaritätstest, weniger Möglichkeiten

Reduziere Anzahl Möglichkeiten: Wähle Einfüge-Reihenfolge so, dass nicht eingefügter Teilgraph zusammenhängend.

Geht das?

Was bringt das?

Planaritätstest, weniger Möglichkeiten

Reduziere Anzahl Möglichkeiten: Wähle Einfüge-Reihenfolge so, dass nicht eingefügter Teilgraph zusammenhängend.

Geht das?

Was bringt das?

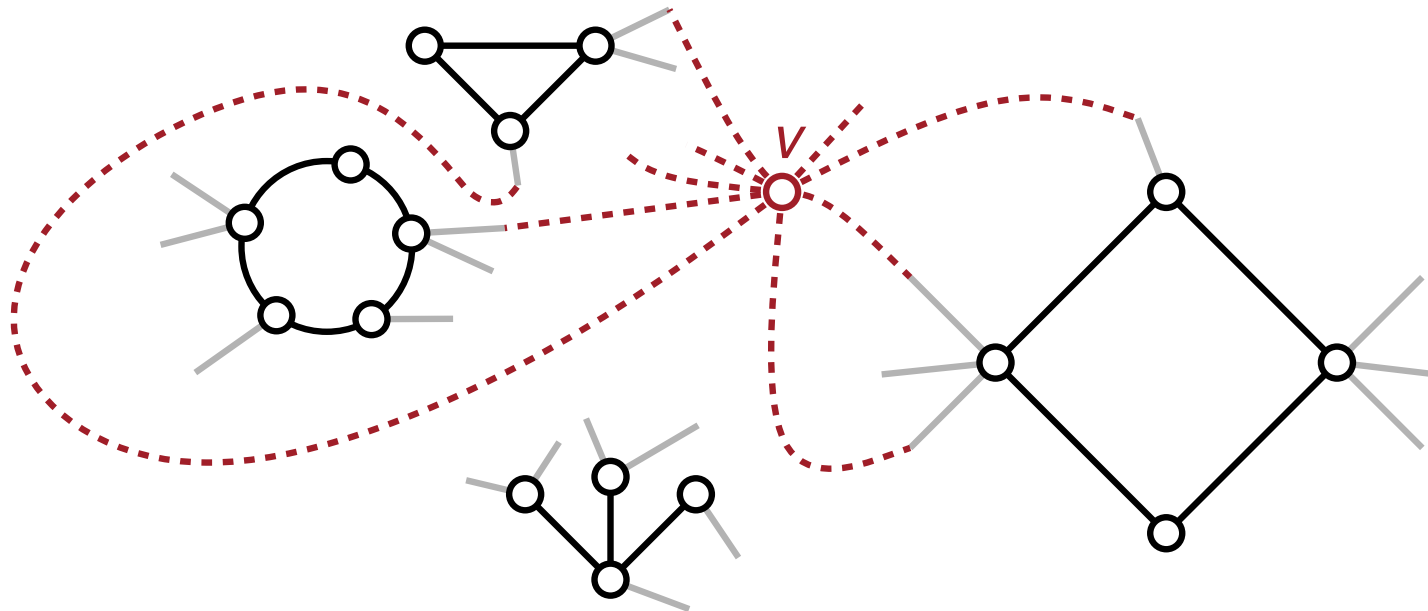
Planaritätstest, weniger Möglichkeiten

Reduziere Anzahl Möglichkeiten: Wähle Einfüge-Reihenfolge so, dass nicht eingefügter Teilgraph zusammenhängend.

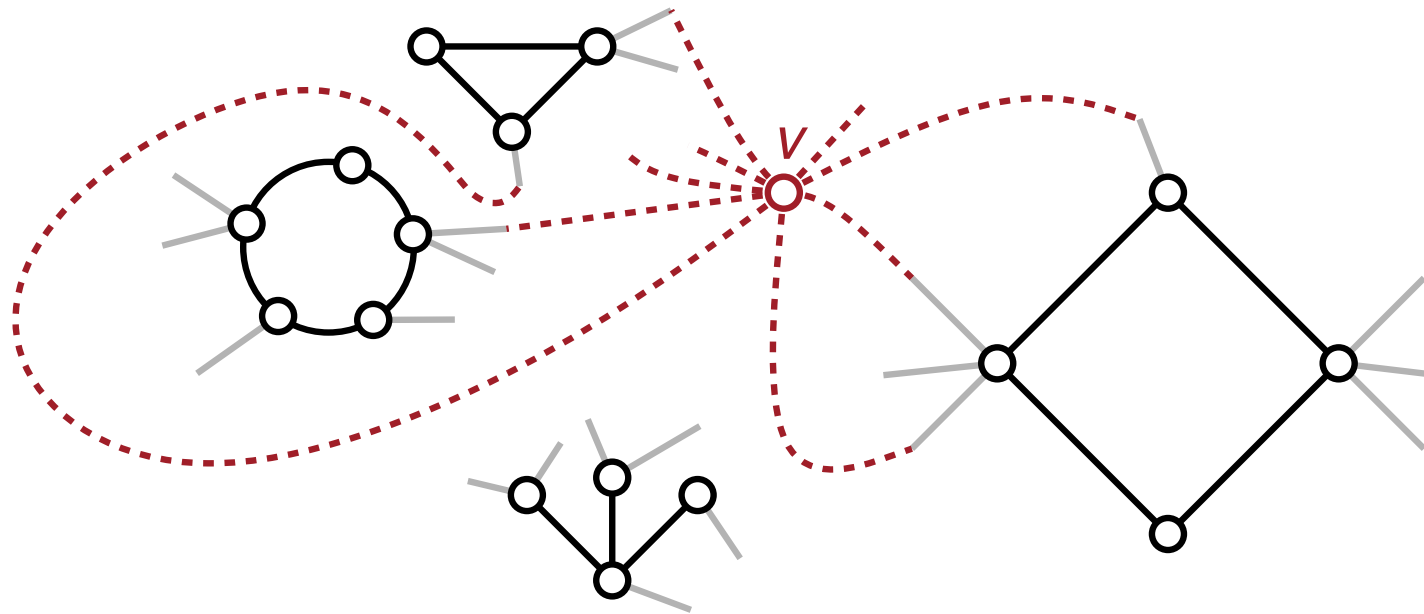
Geht das?

Was bringt das?

Ordnung von Blättern zur Wurzel bezüglich beliebigem Spannbaum.
Alle Halbkanten müssen in selbe (äußere) Facette eingebettet werden.

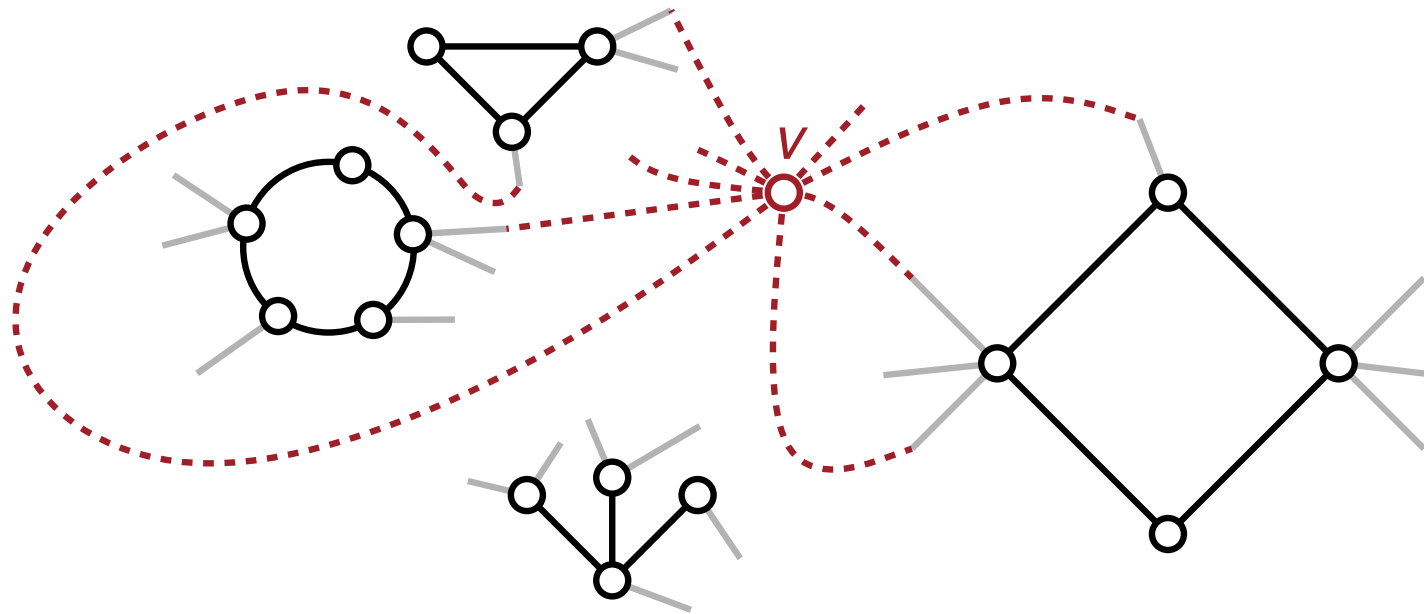


Planaritätstest, Knoten einfügen



1. Reduktion: halb-eingebettete Kanten inzident zu v , die zur selben Komponente gehören müssen konsekutiv sein.
2. Kombination: Lege Reihenfolge der Komponenten und Halbkanten inzident zu v fest.

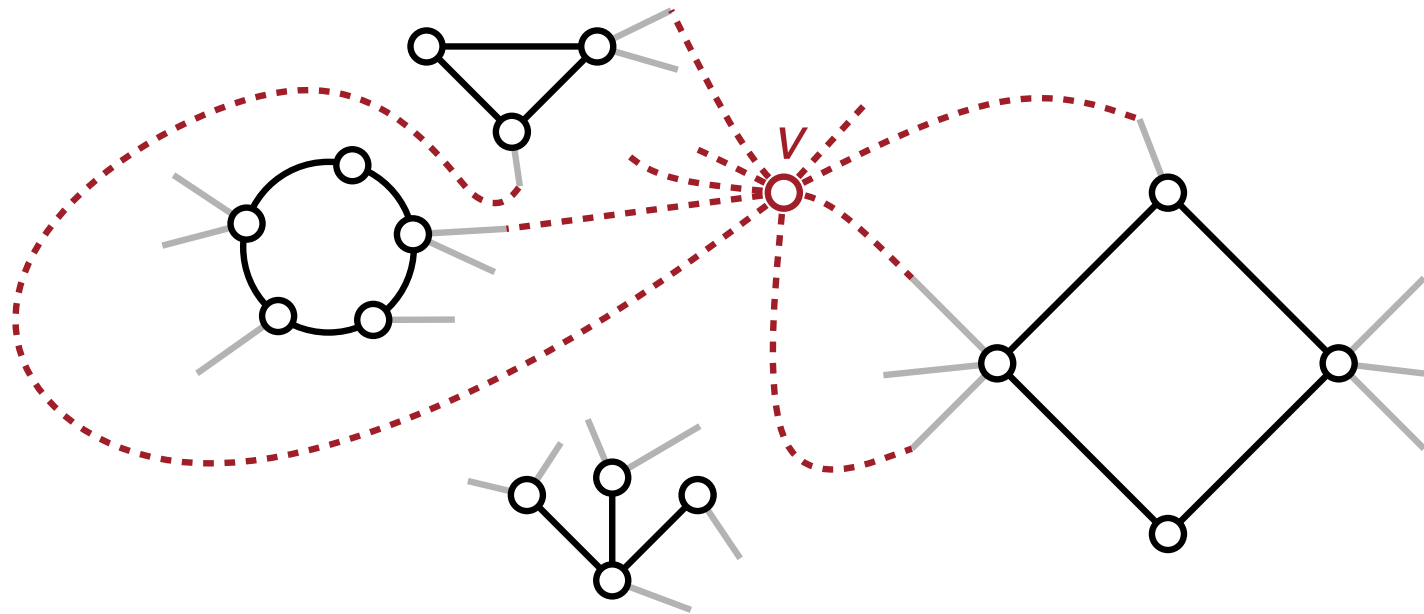
Planaritätstest, Knoten einfügen



1. Reduktion: halb-eingebettete Kanten inzident zu v , die zur selben Komponente gehören müssen konsekutiv sein.
2. Kombination: Lege Reihenfolge der Komponenten und Halbkanten inzident zu v fest.

Speichern aller Möglichkeiten noch immer zu teuer. 😞

Planaritätstest, Knoten einfügen



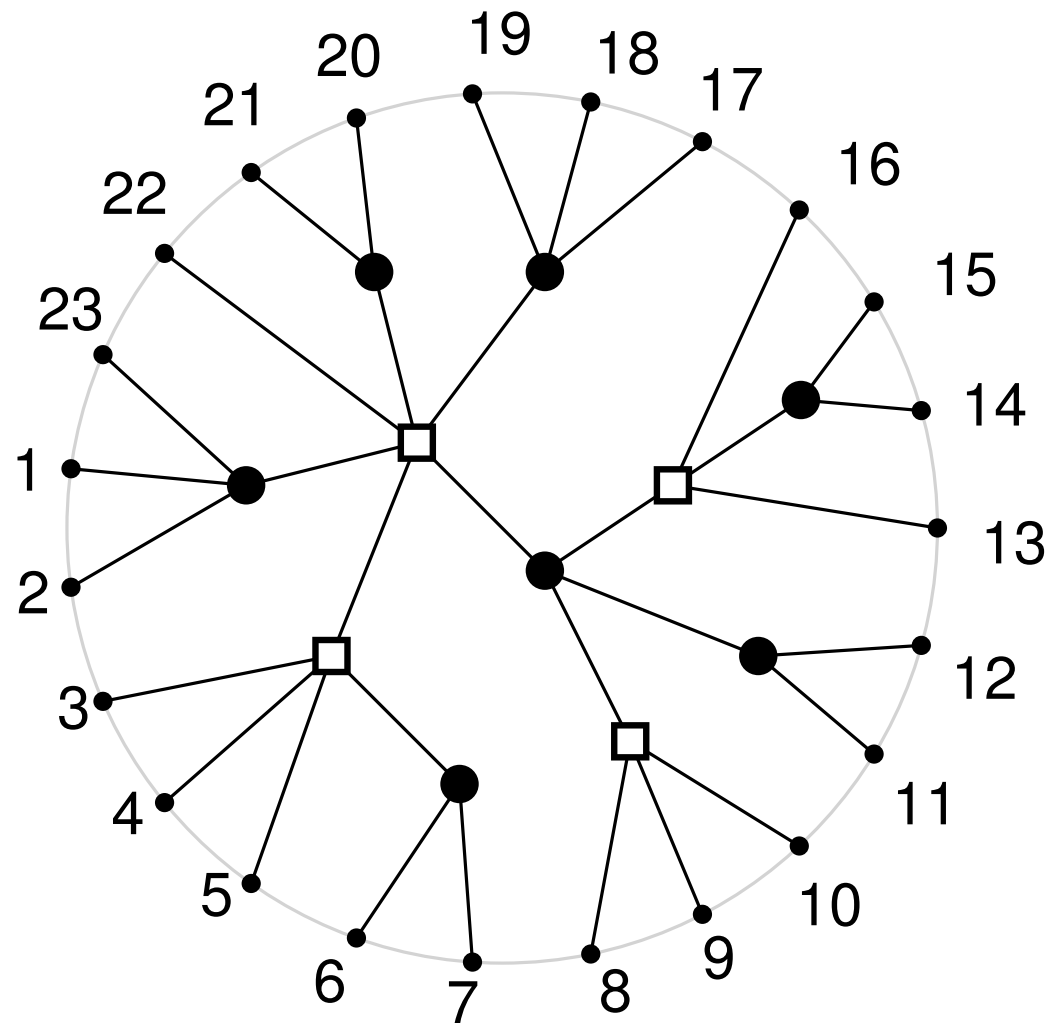
1. Reduktion: halb-eingebettete Kanten inzident zu v , die zur selben Komponente gehören müssen konsekutiv sein.
2. Kombination: Lege Reihenfolge der Komponenten und Halbkanten inzident zu v fest.

Speichern aller Möglichkeiten noch immer zu teuer. 😞

Konstruiere Datenstruktur, die solche Ordnungen kompakt repräsentiert. 🤔

PQ-Bäume

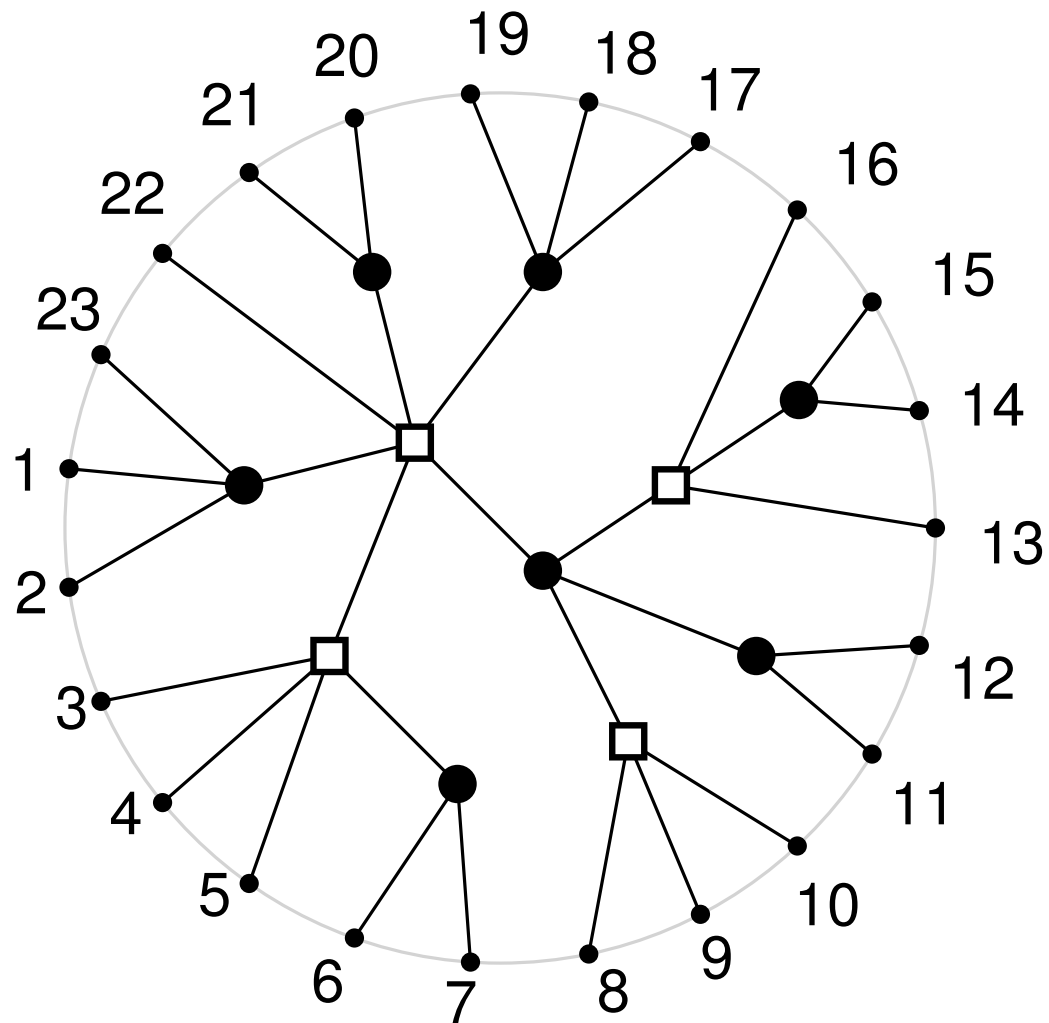
PQ-Baum



- P-Knoten
vertausche Kinder beliebig
- Q-Knoten
nur Spiegeln erlaubt

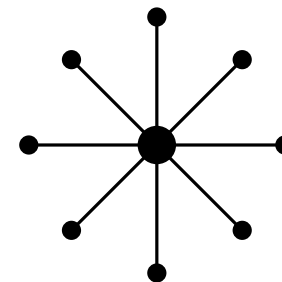
PQ-Baum repräsentiert mögliche zirkuläre Ordnungen der Blätter

PQ-Baum



● P-Knoten
vertausche Kinder beliebig

□ Q-Knoten
nur Spiegeln erlaubt

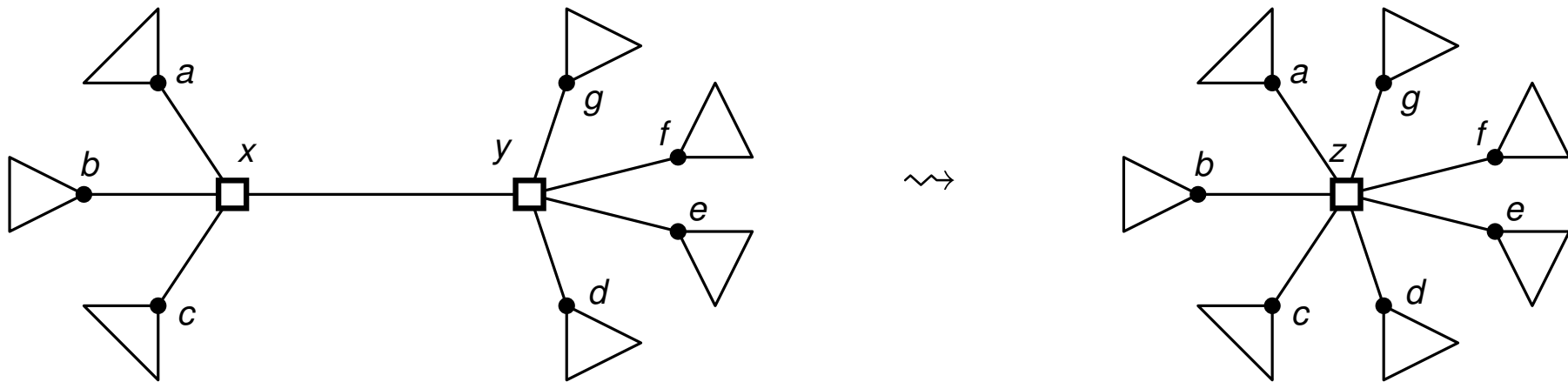


Einzelner P-Knoten repräsentiert alle möglichen
Permutationen seiner Blätter

PQ-Baum repräsentiert mögliche zirkuläre Ordnungen der Blätter

Ordnungserhaltende Kontraktion, Null-Baum

Ordnungserhaltende Kontraktion

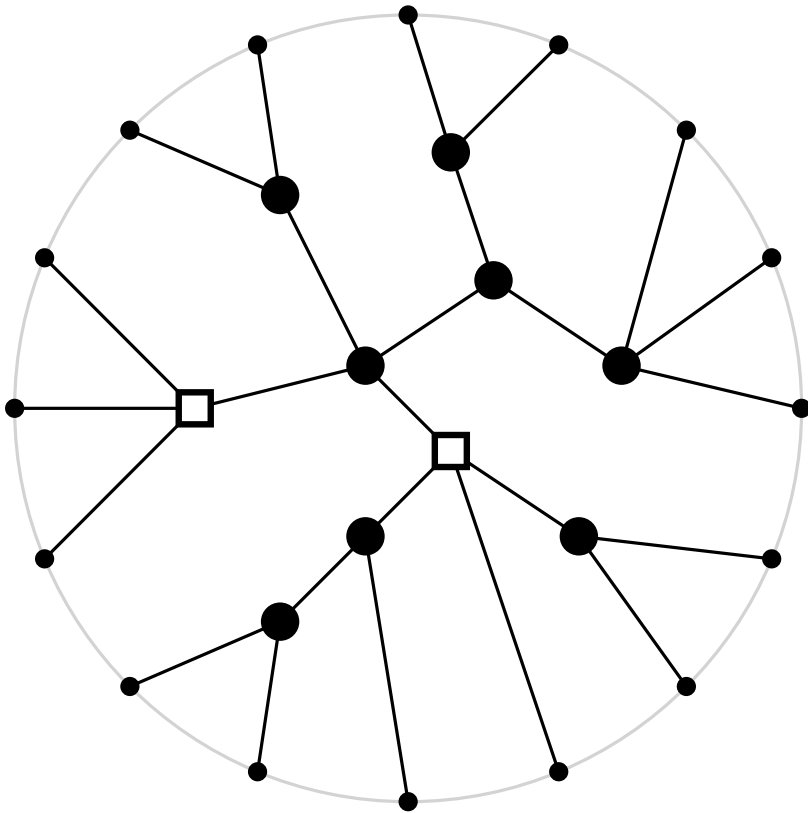


Achtung, dadurch gehen mögliche Ordnungen verloren.

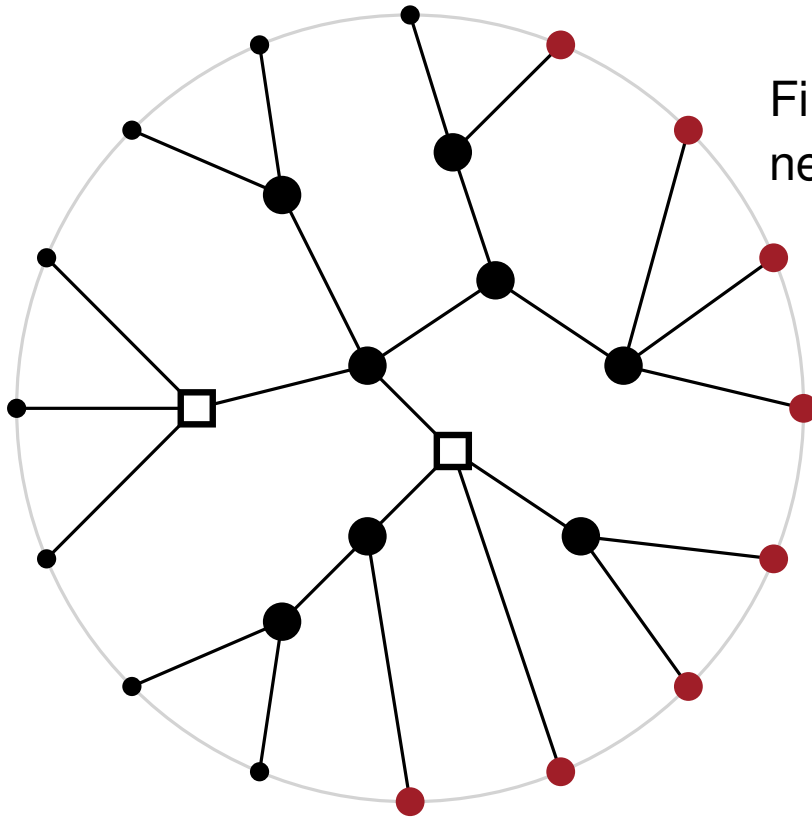
Null-Baum: Repräsentiert formal die leere Menge an Permutationen

Achtung: Null-Baum \neq leerer Baum (repräsentiert Permutationen der leeren Menge)

Konsekutivität von Teilmengen



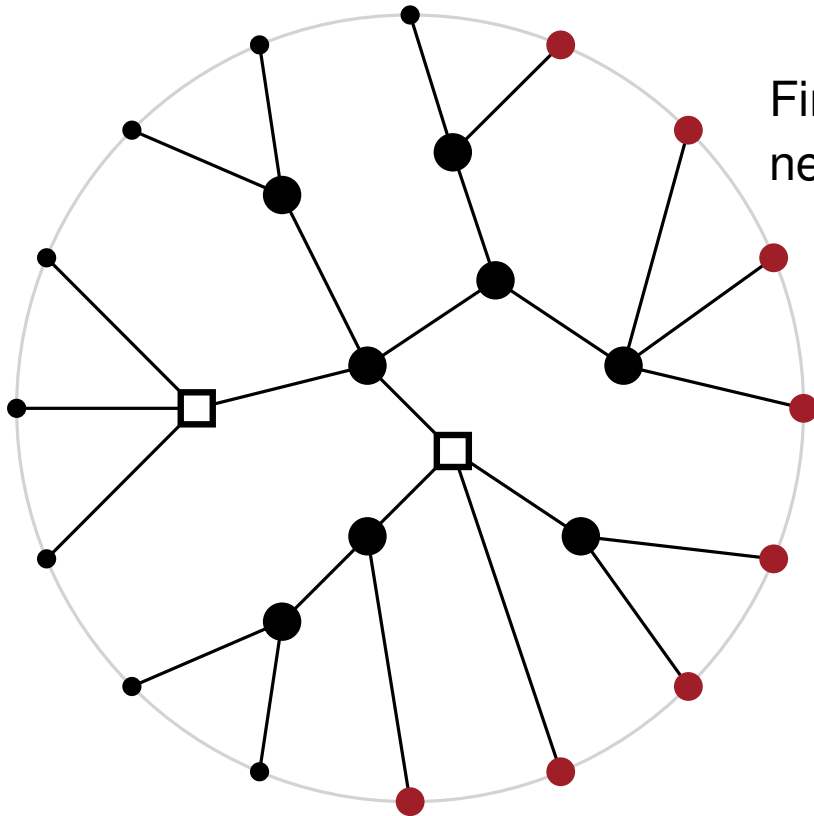
Konsekutivität von Teilmengen



Finde neuen PQ-Baum, der genau diejenigen Permutationen repräsentiert, die

- der vorliegende Baum repräsentiert
- in denen die **roten Blätter** konsekutiv sind

Konsekutivität von Teilmengen

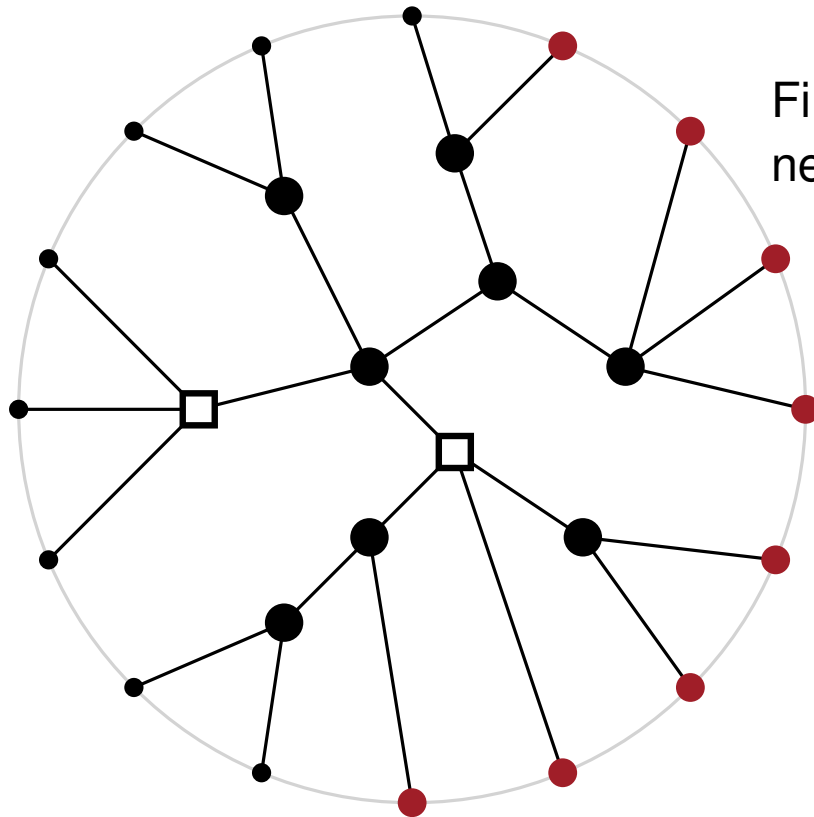


Finde neuen PQ-Baum, der genau diejenigen Permutationen repräsentiert, die

- der vorliegende Baum repräsentiert
- in denen die **roten Blätter** konsekutiv sind

Lassen sich die Blätter nicht so anordnen, dass die roten Knoten konsekutiv sind, so ist das Ergebnis der Null-Baum.

Konsekutivität von Teilmengen



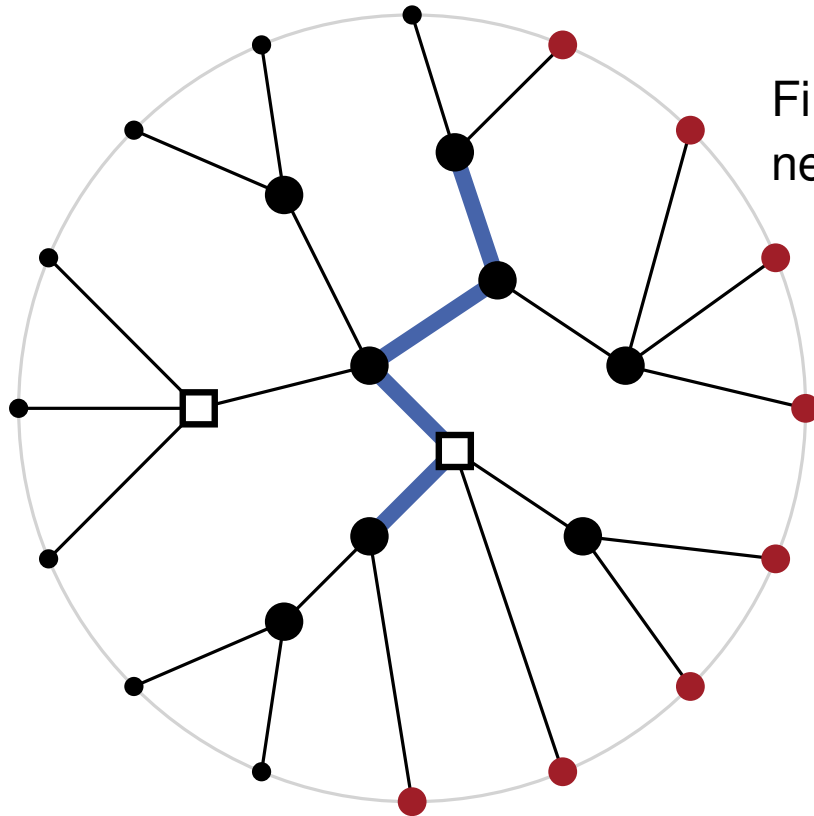
Finde neuen PQ-Baum, der genau diejenigen Permutationen repräsentiert, die

- der vorliegende Baum repräsentiert
- in denen die **roten Blätter** konsekutiv sind

Lassen sich die Blätter nicht so anordnen, dass die roten Knoten konsekutiv sind, so ist das Ergebnis der Null-Baum.

Kante ist **defekt** wenn beide Teilbäume sowohl rote und schwarze Blätter enthalten. \rightsquigarrow defekte Kanten ermöglichen verbotene Permutationen

Konsekutivität von Teilmengen



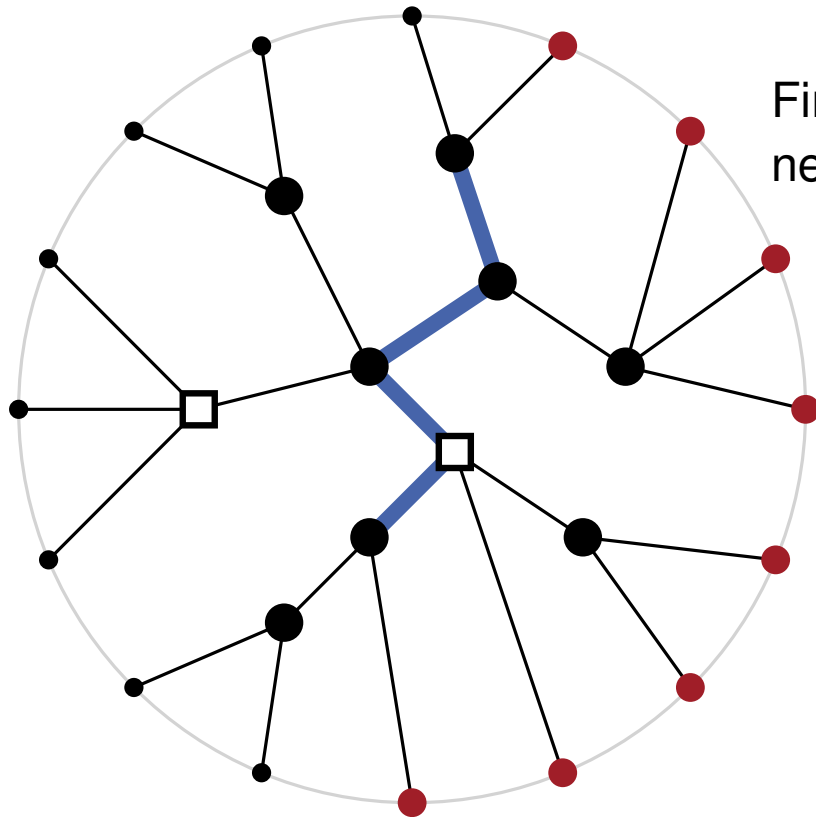
Finde neuen PQ-Baum, der genau diejenigen Permutationen repräsentiert, die

- der vorliegende Baum repräsentiert
- in denen die **roten Blätter** konsekutiv sind

Lassen sich die Blätter nicht so anordnen, dass die roten Knoten konsekutiv sind, so ist das Ergebnis der Null-Baum.

Kante ist **defekt** wenn beide Teilbäume sowohl rote und schwarze Blätter enthalten. \rightsquigarrow defekte Kanten ermöglichen verbotene Permutationen

Konsekutivität von Teilmengen



Finde neuen PQ-Baum, der genau diejenigen Permutationen repräsentiert, die

- der vorliegende Baum repräsentiert
- in denen die **roten Blätter** konsekutiv sind

Lassen sich die Blätter nicht so anordnen, dass die roten Knoten konsekutiv sind, so ist das Ergebnis der Null-Baum.

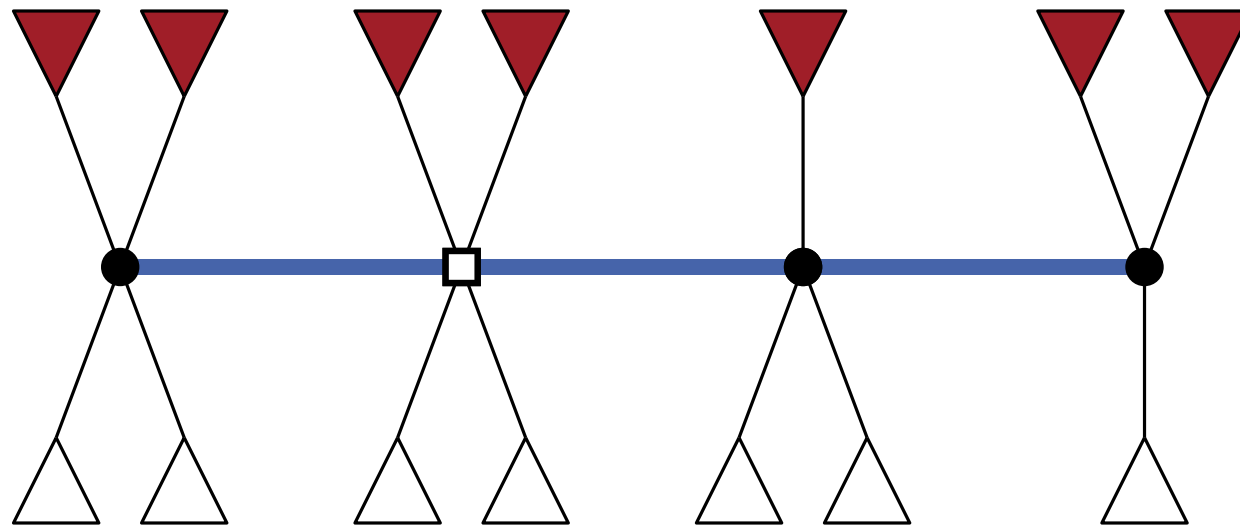
Kante ist **defekt** wenn beide Teilbäume sowohl rote und schwarze Blätter enthalten. \rightsquigarrow defekte Kanten ermöglichen verbotene Permutationen

Lemma:

Gibt es eine Anordnung des PQ-Baums in der die roten Blätter konsekutiv sind, so bilden die defekten Kanten einen Pfad.

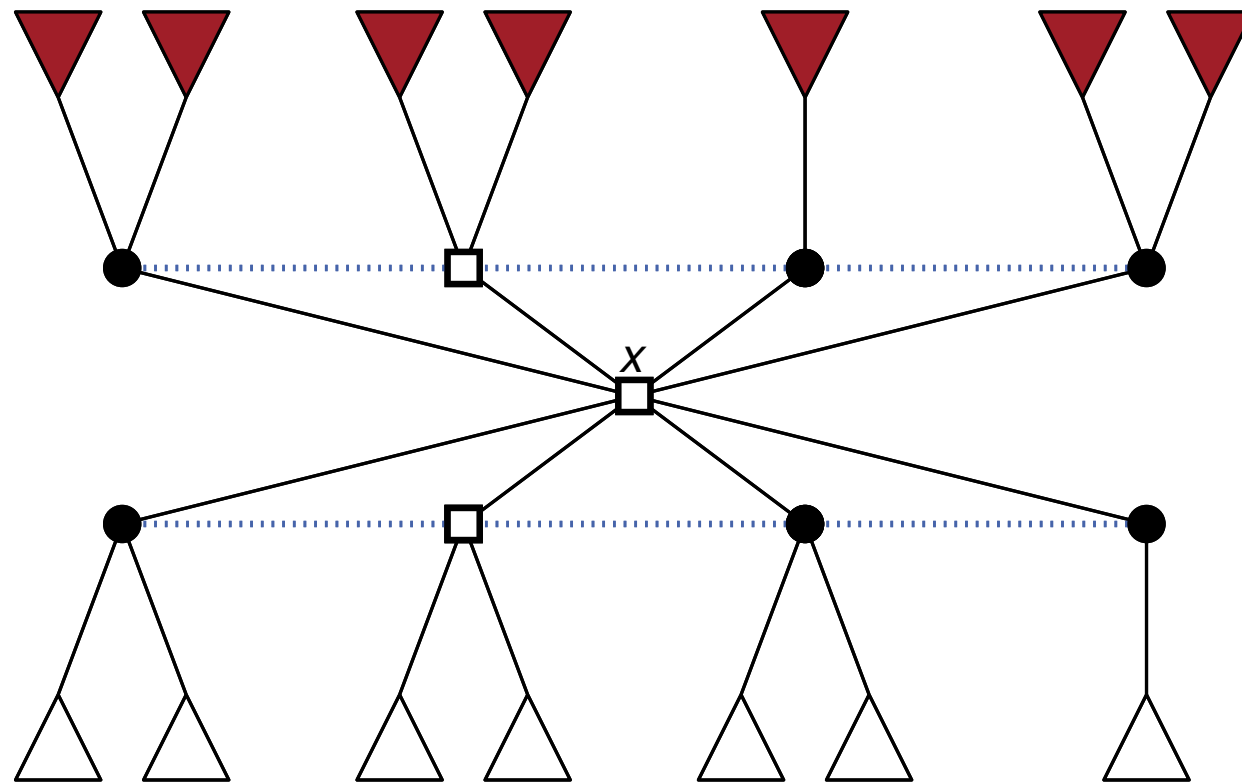
Update des PQ-Baums

1. Bestimme Pfad der defekten Kanten, ordne Baum so an, dass rote und schwarze Blätter auf verschiedenen Seiten liegen.



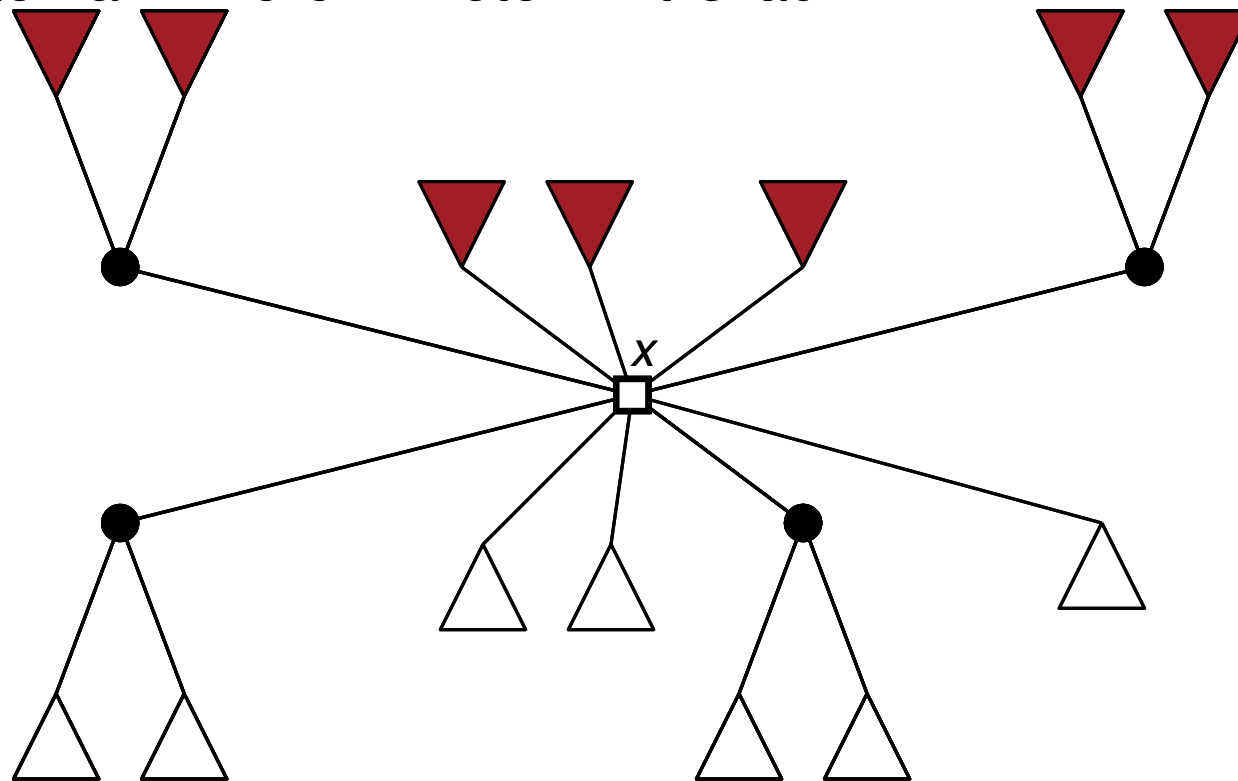
Update des PQ-Baums

1. Bestimme Pfad der defekten Kanten, ordne Baum so an, dass rote und schwarze Blätter auf verschiedenen Seiten liegen.
2. Splitte defekten Pfad (*Terminalpfad*), füge neuen Q-Knoten x ein.



Update des PQ-Baums

1. Bestimme Pfad der defekten Kanten, ordne Baum so an, dass rote und schwarze Blätter auf verschiedenen Seiten liegen.
2. Splitte defekten Pfad (*Terminalpfad*), füge neuen Q-Knoten x ein.
3. Kontrahiere Kanten von x zu anderen Q-Knoten ordnungserhaltend, kontrahiere Kanten an inneren Knoten mit Grad 2



Satz:

Der Update-Algorithmus ist korrekt.

Beweis:

- Struktur des resultierenden Baums unabhängig von Ordnung.
- Andererseits: jede Ordnung des resultierenden Baums wird vom Startbaum repräsentiert.

Satz:

Der Update-Algorithmus ist korrekt.

Beweis:

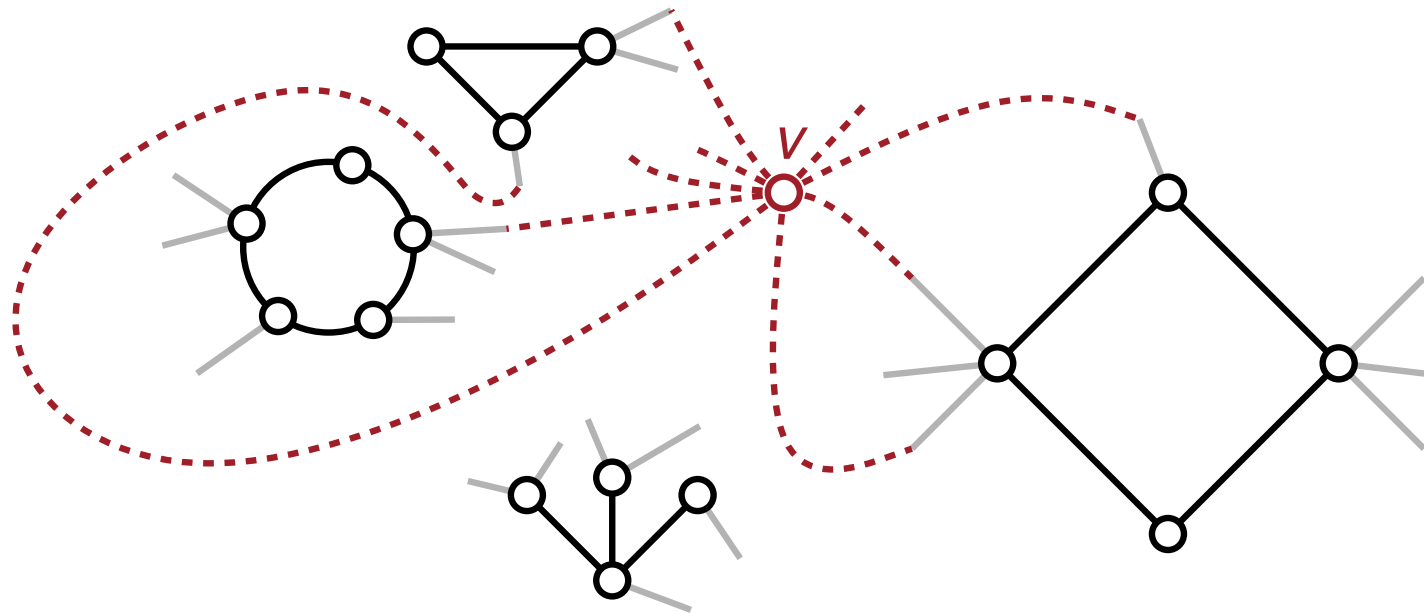
- Struktur des resultierenden Baums unabhängig von Ordnung.
- Andererseits: jede Ordnung des resultierenden Baums wird vom Startbaum repräsentiert.

Offensichtlich: Update implementierbar mit polynomieller Laufzeit.

Lineare Implementierung braucht mehr Arbeit.

Zurück zum Planaritätstest

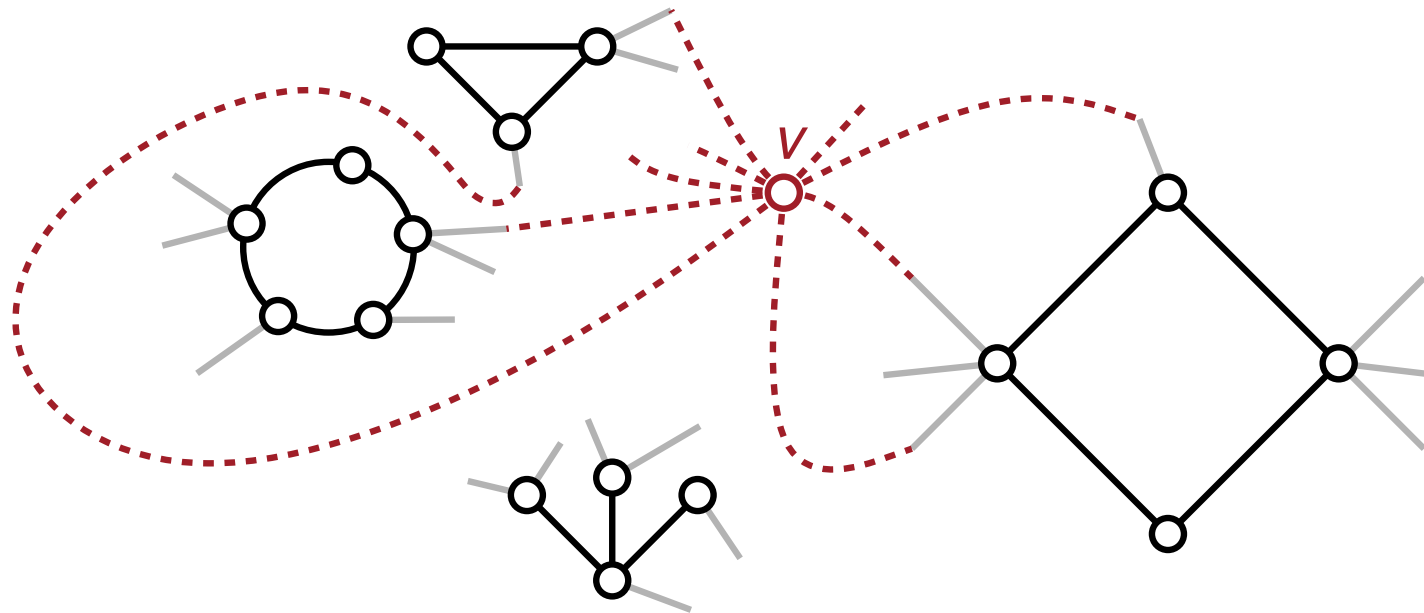
Planaritätstest, Knoten einfügen



1. Reduktion: halb-eingebettete Kanten inzident zu v , die zur selben Komponente gehören müssen konsekutiv sein.
2. Kombination: Lege Reihenfolge der Komponenten und Halbkanten inzident zu v fest.

Explizites Speichern aller Möglichkeiten noch immer zu teuer.

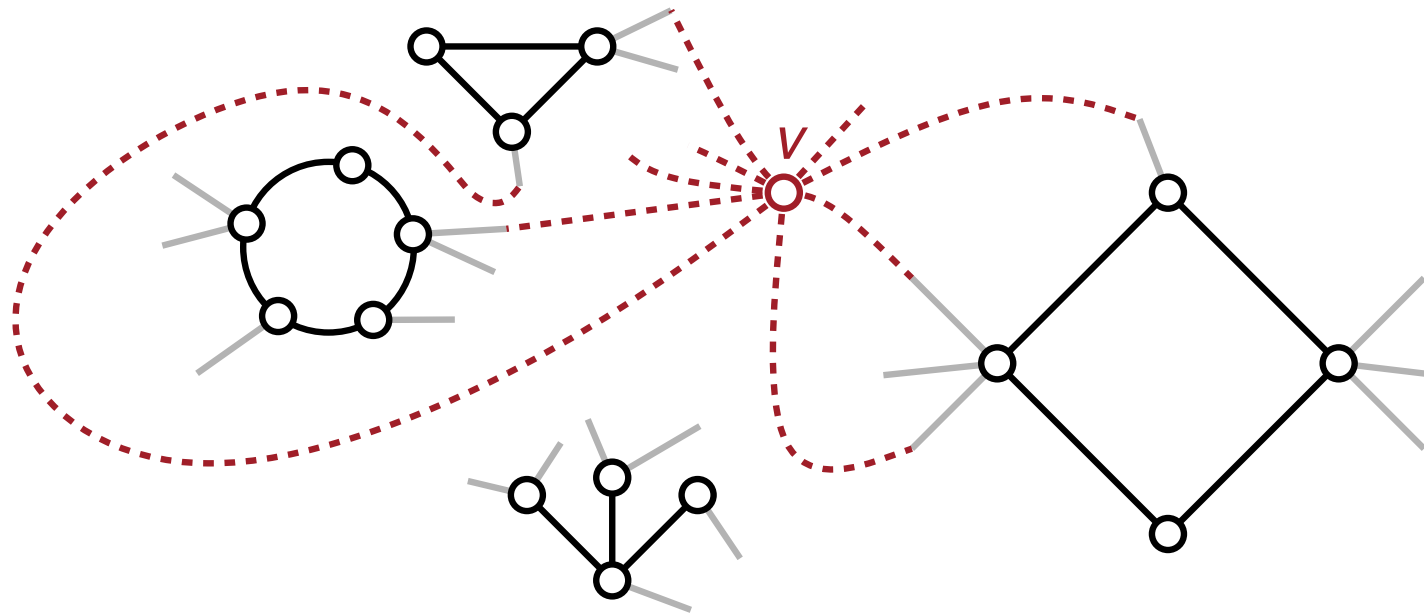
Planaritätstest, Knoten einfügen

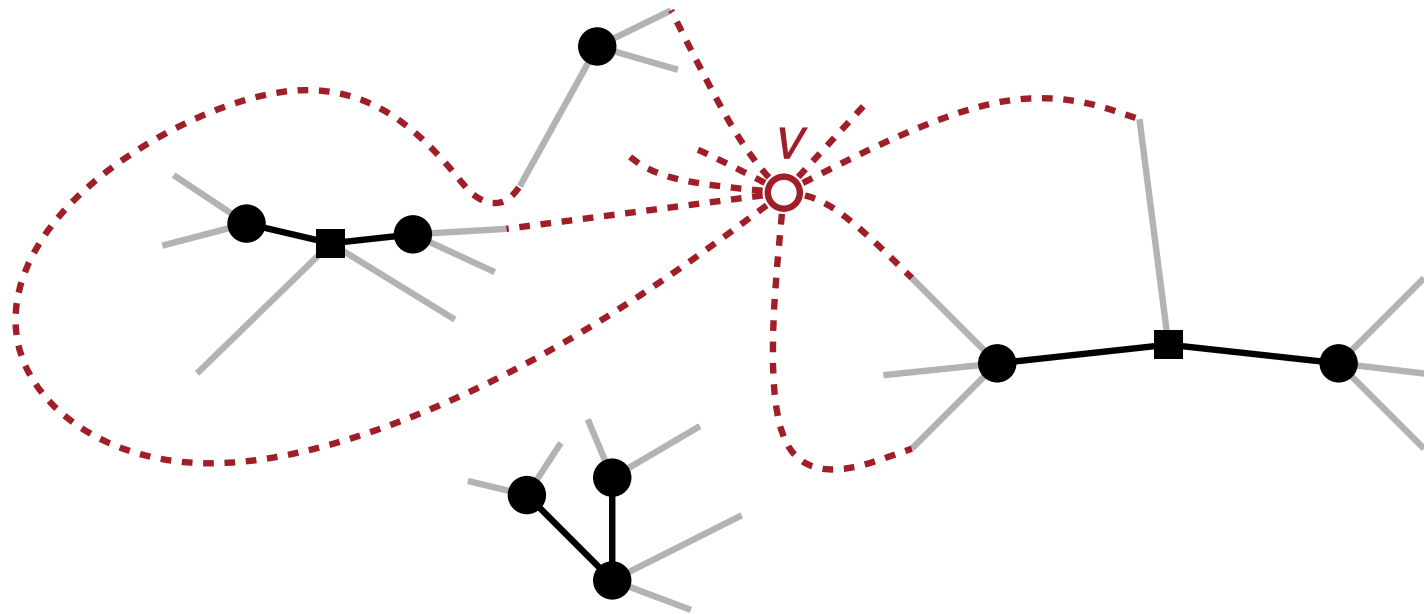


1. Reduktion: halb-eingebettete Kanten inzident zu v , die zur selben Komponente gehören müssen konsekutiv sein.
2. Kombination: Lege Reihenfolge der Komponenten und Halbkanten inzident zu v fest.

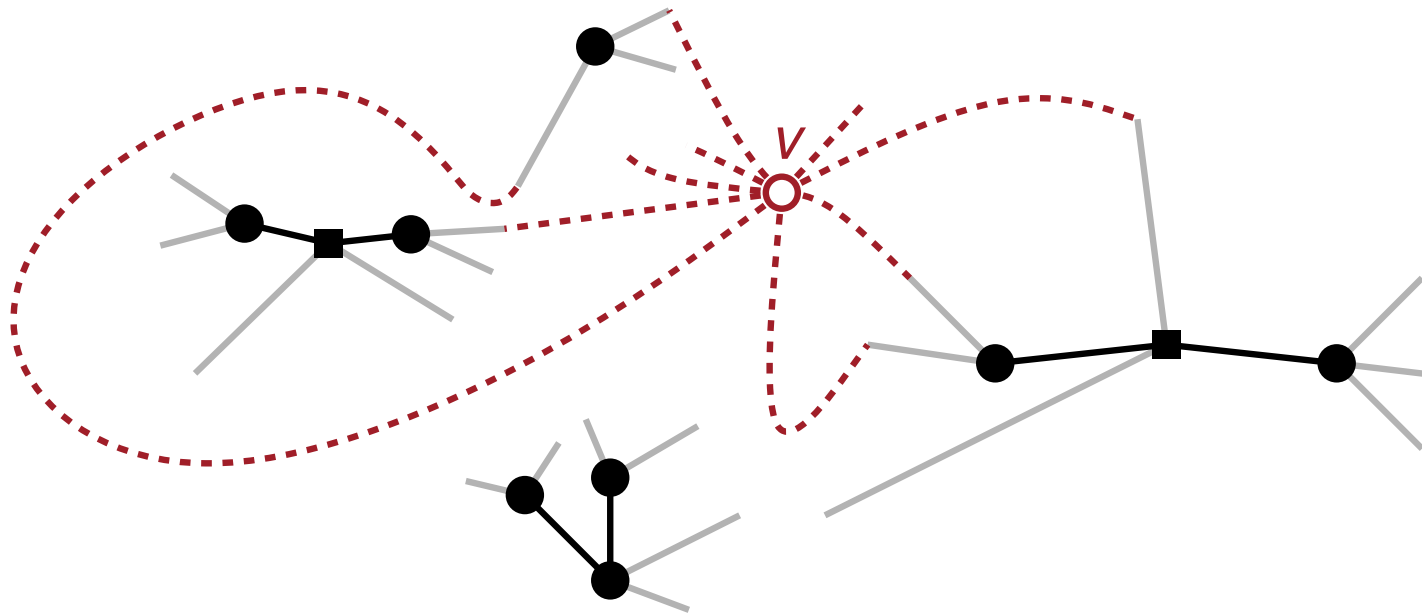
Explizites Speichern aller Möglichkeiten noch immer zu teuer.

↪ Verwende PQ-Baum



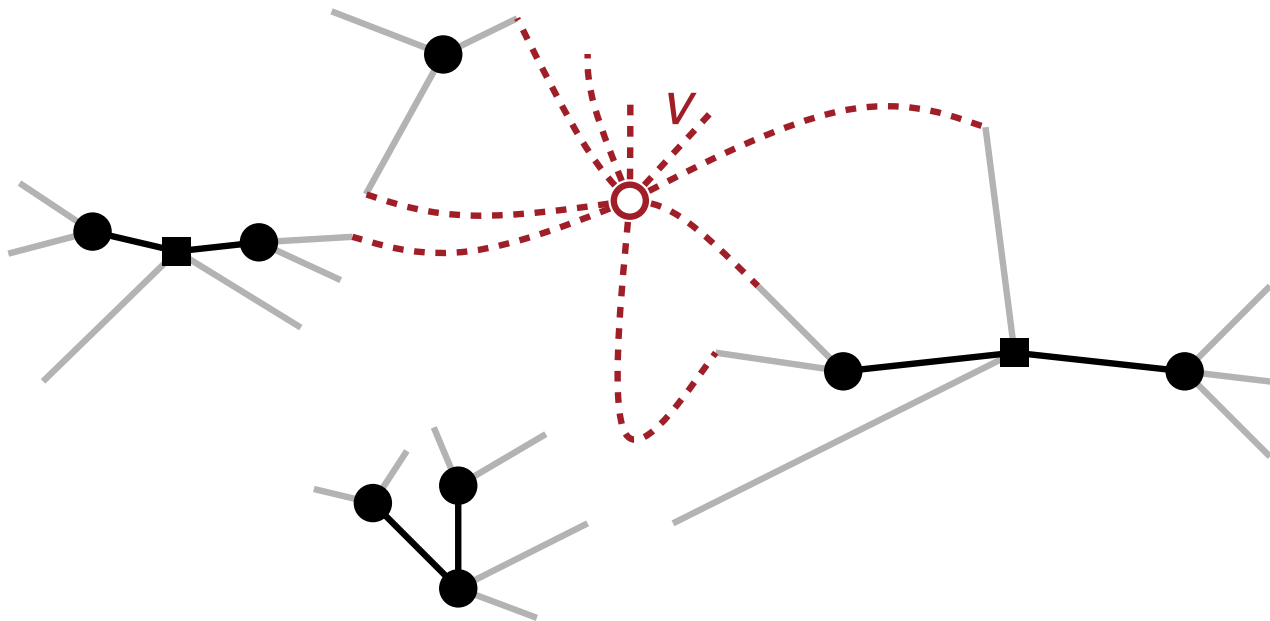


Verwalte PQ-Baum für jede Komponente



Verwalte PQ-Baum für jede Komponente

- Reduktion: mache Halbkanten zu v in jeder Komponente konsekutiv



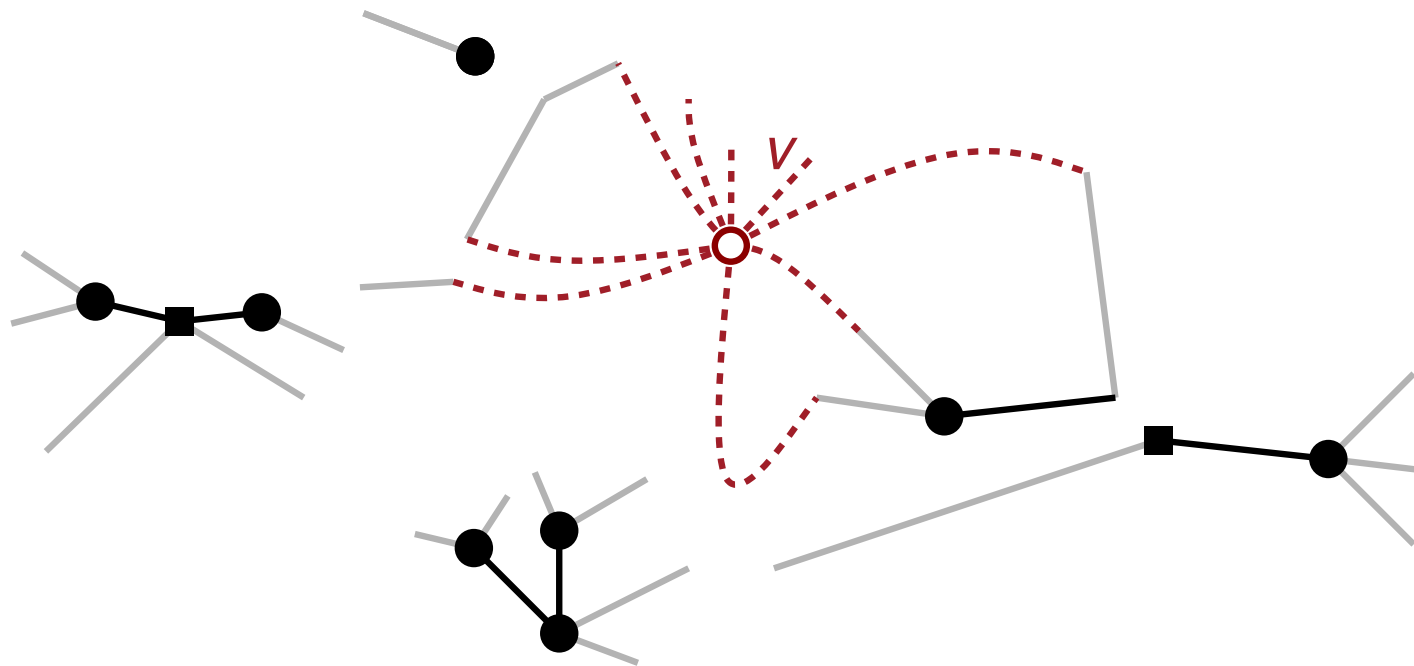
Verwalte PQ-Baum für jede Komponente

- Reduktion: mache Halbkanten zu v in jeder Komponente konsekutiv

- Kombination: verschmelze zu einem PQ-Baum

↪ neuer PQ-Baum für resultierende Zusammenhangskomponente

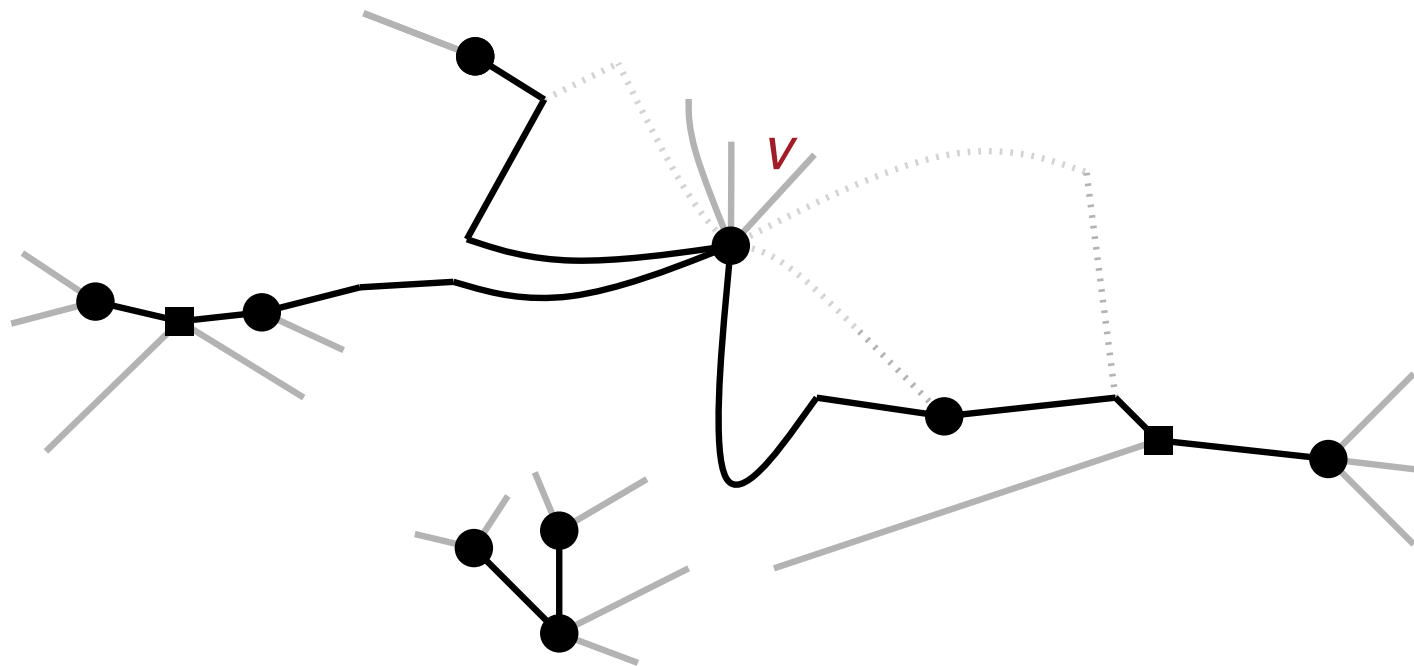
Kosten pro Knoten: amortisiert $O(\deg v)$



Verwalte PQ-Baum für jede Komponente

- Reduktion: mache Halbkanten zu v in jeder Komponente konsekutiv
 - Kombination: verschmelze zu einem PQ-Baum
- ↪ neuer PQ-Baum für resultierende Zusammenhangskomponente

Kosten pro Knoten: amortisiert $O(\deg v)$

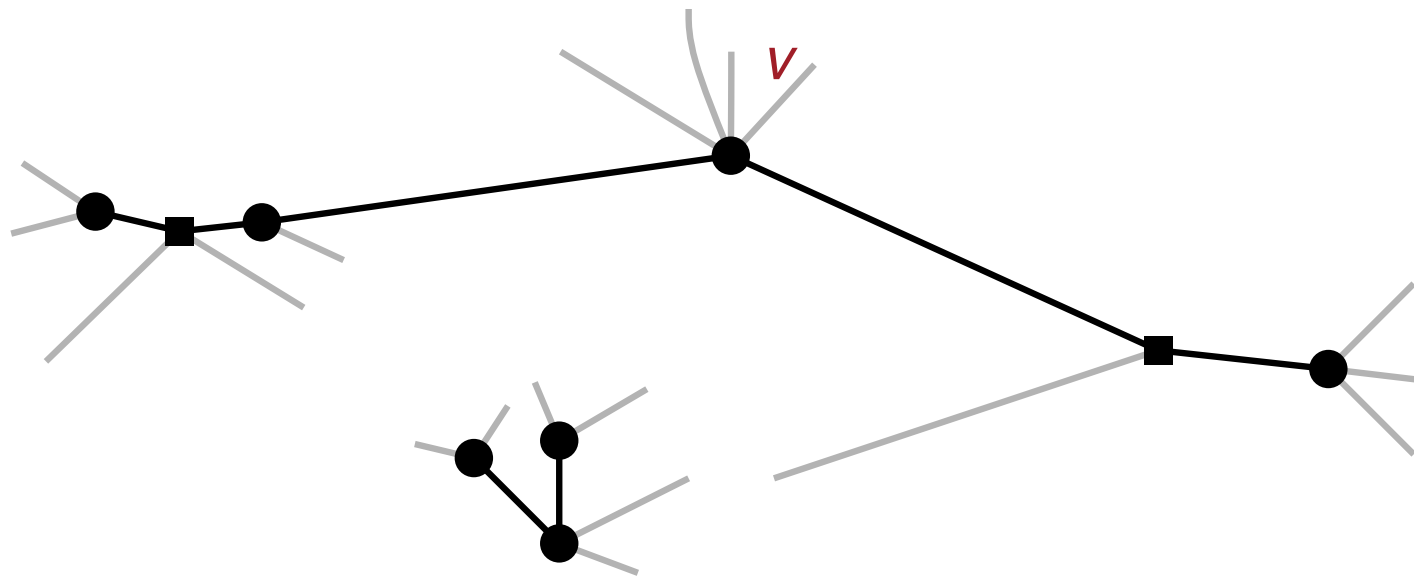


Verwalte PQ-Baum für jede Komponente

- Reduktion: mache Halbkanten zu v in jeder Komponente konsekutiv
- Kombination: verschmelze zu einem PQ-Baum

↪ neuer PQ-Baum für resultierende Zusammenhangskomponente

Kosten pro Knoten: amortisiert $O(\deg v)$



Verwalte PQ-Baum für jede Komponente

- Reduktion: mache Halbkanten zu v in jeder Komponente konsekutiv
 - Kombination: verschmelze zu einem PQ-Baum
- ↪ neuer PQ-Baum für resultierende Zusammenhangskomponente

Kosten pro Knoten: amortisiert $O(\deg v)$

Graph ist genau dann planar, wenn keiner der Reduktionsschritte scheitert.

Einbettung kann gewonnen werden, indem Schritte nach und nach rückgängig gemacht werden. Dabei von PQ-Bäumen repräsentierte Ordnungen wählen/erweitern.

Graph ist genau dann planar, wenn keiner der Reduktionsschritte scheitert.

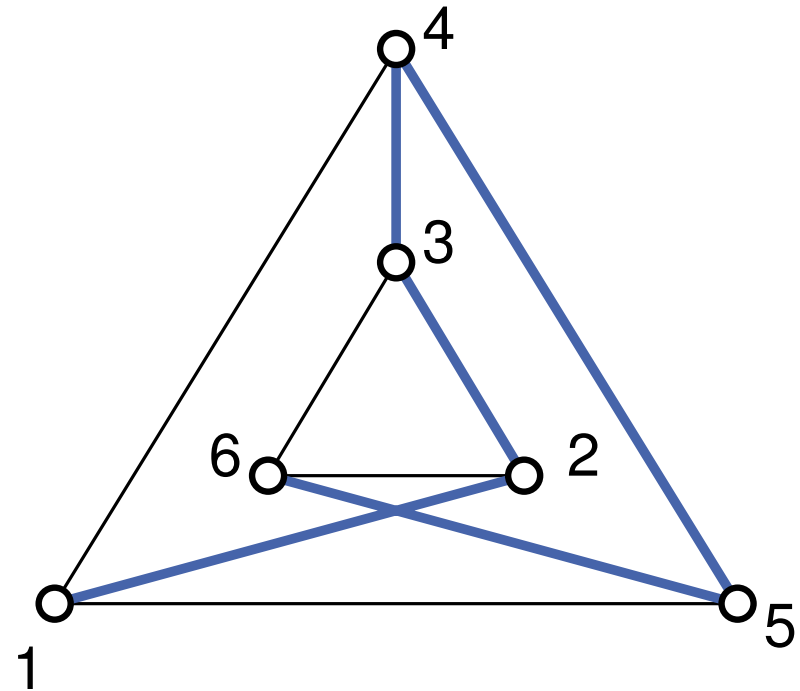
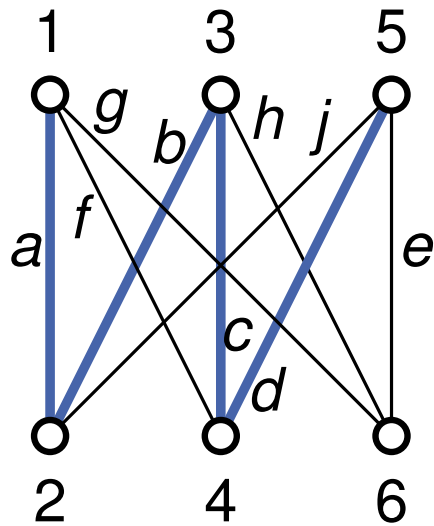
Einbettung kann gewonnen werden, indem Schritte nach und nach rückgängig gemacht werden. Dabei von PQ-Bäumen repräsentierte Ordnungen wählen/erweitern.

Vorsicht bei linearer Laufzeit: PQ-Bäume "umwurzeln" ist teuer.
Verschmelzen funktioniert nur effizient, wenn genau einer der Bäume seine Wurzel behält.

Zwei mögliche Ordnungen:

- s - t -Ordnung (zweifach zusammenhängende Graphen)
(vgl. Vorlesung Graphvisualisierung) [Lempel, Even, Cederbaum '67]
- Tiefensuche [Shih, Hsu '99, Boyer, Myrvold '04]

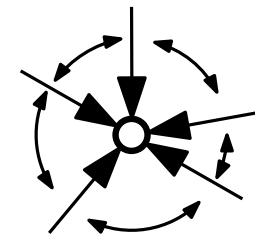
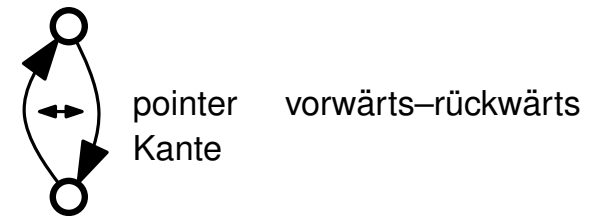
Beispiele



Anhang: lineare Laufzeit bei PQ-Bäumen

Implementierung mit linearer Laufzeit:

- Wähle Wurzel
- speichere jede Kante doppelt gerichtet
- speichere eingehende Kanten an jedem Knoten in doppelt verketteter Liste
- Flag an jeder Kante ob sie vom Parent kommt
- P-Knoten haben Pointer auf Parent



Achtung: Eltern von Q-Knoten sind aufwendig zu berechnen.

Berechnung des Terminalpfades

Lemma: Länge des Terminalpfades p Anzahl der konsekutiven Elemente k
Terminalpfad kann in $O(p + k)$ Zeit gefunden werden.

Berechnung des Terminalpfades

Lemma: ^{Länge des Terminalpfades} $O(p + k)$ ^{Anzahl der konsekutiven Elemente} Zeit gefunden werden.

Klassifiziere Knoten des PQ-Baums:

- Blätter sind **voll** wenn sie ein **rotes** Element repräsentieren
- Innere Knoten sind **voll** wenn alle bis auf einen Nachbarn voll sind.
- Knoten sind **teilweise voll** wenn einer ihrer Nachbarn voll ist.

$\rightsquigarrow O(k)$ Zeit bis alle Knoten gelabelt.

Berechnung des Terminalpfades

Lemma: $O(p + k)$ Zeit gefunden werden.
Länge des Terminalpfades p Anzahl der konsekutiven Elemente k

Klassifiziere Knoten des PQ-Baums:

- Blätter sind **voll** wenn sie ein **rotes** Element repräsentieren
- Innere Knoten sind **voll** wenn alle bis auf einen Nachbarn voll sind.
- Knoten sind **teilweise voll** wenn einer ihrer Nachbarn voll ist.

$\rightsquigarrow O(k)$ Zeit bis alle Knoten gelabelt.

Teilweise volle Knoten sind potentielle Endknoten des Terminalpfades.

- Starte potenziellen Pfad an jedem Terminalknoten, suche zu Eltern.
- Stoppe Erweiterung eines Pfades wenn anderer Pfad getroffen wird.
- Pfade bilden schließlich Baum
- Höchster Knoten, der teilweise voll ist, oder zwei benachbarte Suchpfade hat ist höchster Punkt des Terminalpfades.

Berechnung des Terminalpfades

Lemma: ^{Länge des Terminalpfades} $O(p + k)$ ^{Anzahl der konsekutiven Elemente} Zeit gefunden werden.

Klassifiziere Knoten des PQ-Baums:

- Blätter sind **voll** wenn sie ein **rotes** Element repräsentieren
- Innere Knoten sind **voll** wenn alle bis auf einen Nachbarn voll sind.
- Knoten sind **teilweise voll** wenn einer ihrer Nachbarn voll ist.

$\rightsquigarrow O(k)$ Zeit bis alle Knoten gelabelt.

Teilweise volle Knoten sind potentielle Endknoten des Terminalpfades.

- Starte potenziellen Pfad an jedem Terminalknoten, suche zu Eltern.
- Stoppe Erweiterung eines Pfades wenn anderer Pfad getroffen wird.
- Pfade bilden schließlich Baum
- Höchster Knoten, der teilweise voll ist, oder zwei benachbarte Suchpfade hat ist höchster Punkt des Terminalpfades.

Wie findet man schnell die Elternknoten?

Update-Schritt

Splitte Terminal-Pfad indem neue Knoten für volle Nachbarn erstellt werden.

Einzelner Split:

- Merke Nachbarn auf Terminalpfad, entferne diese Kanten $O(1)$
- Erstelle Kopie, übertrage volle Nachbarn $\frac{O(\# \text{Anzahl volle Nachbarn})}{O(p+k)}$

Zentraler Q-Knoten in $O(p)$ Zeit

Kontraktionen in $O(1)$

Gesamtlaufzeit

X Grundmenge

$\mathcal{U} = \{U_1, \dots, U_\ell\}$ Menge von Teilmengen von X .

Satz:

PQ-Baum, der Permutationen repräsentiert, in denen U_1, \dots, U_ℓ konsekutiv sind kann in amortisierter Zeit $O(|X| + |U_1| + \dots + |U_\ell|)$ berechnet werden.

Beweis: Betrachte Potentialfunktion $\phi(\mathcal{U}, i) = u_i + |Q_i| + \sum_{x \in P_i} (\deg(x) - 1)$.

- $u_i = \sum_{j>i} |U_j|$
- $Q_i = Q$ -Knoten im Baum nachdem U_1, \dots, U_i verarbeitet wurde.
- $P_i = P$ -Knoten im Baum ...

$$\phi(\mathcal{U}, 0) = \Theta(|X|)$$

Bezahle alle Operationen mit Φ

ϕ bleibt ≥ 0

\rightsquigarrow behauptete Laufzeit.