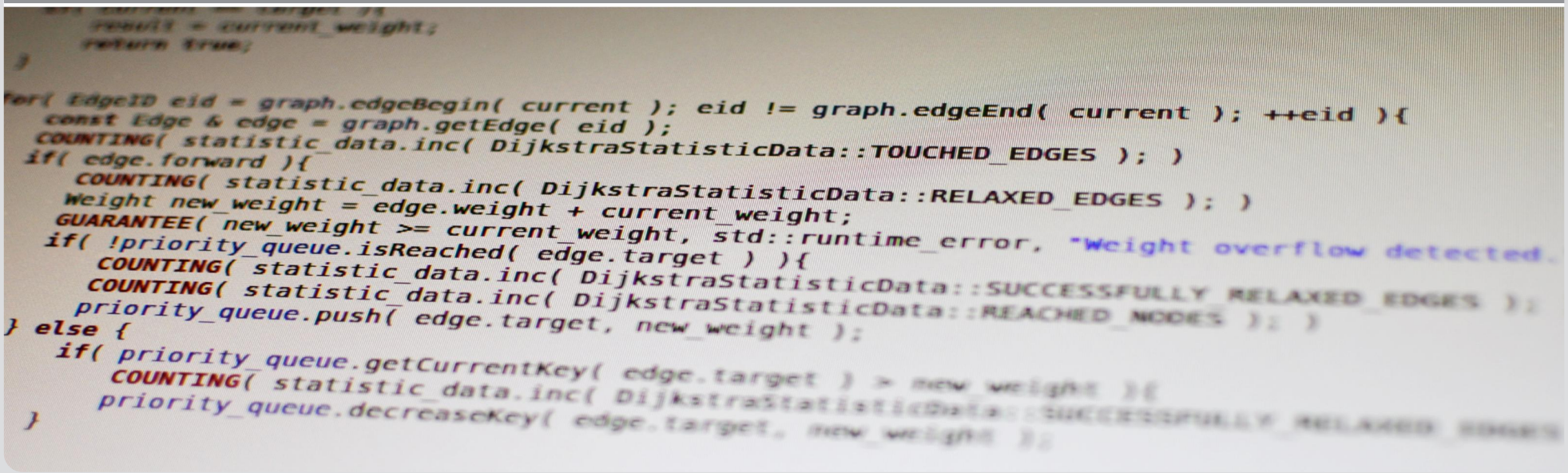
Karlsruher Institut für Technologie

Übung 8 – Algorithmen II

Sebastian Lamm, Demian Hespe – *lamm@kit.edu,hespe@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS18.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

Übungsinhalt

- parametrisierte Algorithmen
  (schwierige Probleme in Spezialfällen exakt lösen)
- Parallelverarbeitung
  - Modelle
  - Verbindungsstrukturen
  - Anwendungen
    - Präfixsumme
    - Paralleles Sortieren
    - Effizienz

Ergänzendes Material

- parametrisierte Algorithmen: Rekurrenzrelationen
- parametrisierte Algorithmen: Vertex Cluster Deletion

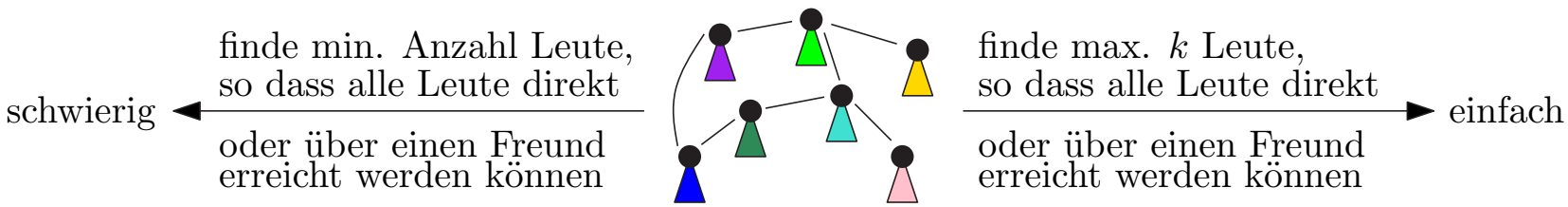
Parametrisierte Algorithmen

***fixed parameter tractable* (FPT)**

Warum Probleme parametrisieren?

- es gibt “schwierige” Probleme (z.B. Minimum Independent Set)
  - → allgemeine Instanzen haben zu lange Berechnungszeit

- Kann man Spezialfälle eventuell effizient berechnen?
  - → Identifizierung zusätzlicher Parameter k der Problemstellung
  - → falls “Komplexität” in diesen Parametern k steckt, effiziente Lösungen für $k = const.$!



Parametrisierte Algorithmen

Definition

- Ein Problem heißt fixed parameter tractable, wenn es eine Laufzeit

$$T(n,k) = \mathcal{O}(f(k) \cdot \mathrm{p}(n))$$

hat, mit $f(\cdot)$ berechenbar, $p(\cdot)$ Polynom.

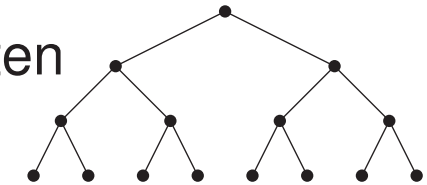
($f(\cdot)$ darf nicht von n abhängen und $p(\cdot)$ nicht von k; häufig Entscheidungsprobleme)

Parametrisierte Algorithmen

Techniken

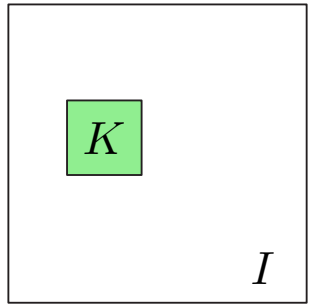
Tiefenbeschränkte Suche

- erschöpfendes Aufzählen und Testen aller Möglichkeiten
  $\rightarrow$ mit geeignetem Suchbaum beschränkter Tiefe
  (k gibt Hinweis, wie man in die Tiefe gehen muss)



Kernbildung

- Probleminstanz auf (schwierigen) Problemkern reduzieren
- Problemkern mit anderer Technik lösen



Weitere Techniken

- Farbkodierung, induktive Kompression, Baumzerlegung, ...

Parametrisierte Algorithmen

Schiebepuzzle

Problemstellung

- gegeben $n \times n$ Schiebepuzzle, $k \in \mathbb{N}$
- entscheide, ob das Puzzle in $\leq k$ Zügen gelöst werden kann
  - das Puzzle ist gelöst, wenn die Teile sortiert sind
  - Loch wird pro Zug eine Position horizontal oder vertikal verschoben

Algorithmus A

- es gibt ≤ 4 Möglichkeiten in jedem Zug, k Züge
  - baue Suchbaum (Höhe k, Verzweigungsgrad ≤ 4)
    $\rightarrow$ Baumgröße $\mathcal{O}(4^k)$
  - teste jeden Knoten auf korrekte Lösung
    $\rightarrow$ Aufwand $\mathcal{O}(n^2) \in \mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtaufwand: $\mathcal{O}(4^k n^2) \Rightarrow$ FPT

$(T(n,k) = 4T(n,k-1) + poly(n))$

24	8	13	12	20
11	2		17	21
7	15	14	19	5
6	10	3	9	1
4	23	11	18	22

Parametrisierte Algorithmen

Schiebepuzzle

Problemstellung

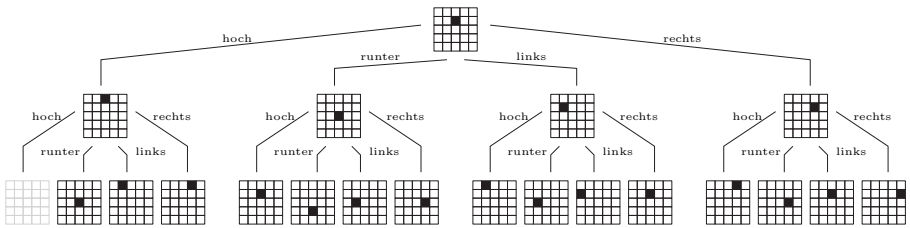
- gegeben $n \times n$ Schiebepuzzle, $k \in \mathbb{N}$
- entscheide, ob das Puzzle in $\leq k$ Zügen gelöst werden kann
  - das Puzzle ist gelöst, wenn die Teile sortiert sind
  - Loch wird pro Zug eine Position horizontal oder vertikal verschoben

Algorithmus A

- es gibt ≤ 4 Möglichkeiten in jedem Zug, k Züge
  - baue Suchbaum (Höhe k, Verzweigungsgrad ≤ 4)
    $\rightarrow$ Baumgröße $\mathcal{O}(4^k)$
  - teste jeden Knoten auf korrekte Lösung
    $\rightarrow$ Aufwand $\mathcal{O}(n^2) \in \mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtaufwand: $\mathcal{O}(4^k n^2) \Rightarrow$ FPT

$(T(n,k) = 4T(n,k-1) + poly(n))$

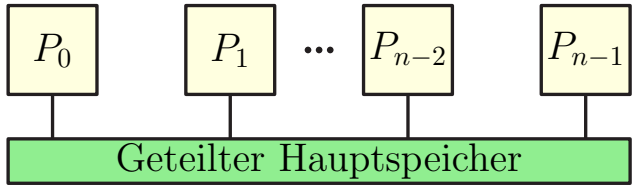


Parallelverarbeitung

Modelle

PRAM

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Verteilter gemeinsamer Speicher

(symmetrisch) gemeinsamer Speicher

B(ulk)S(ynchronous)P(arallel)

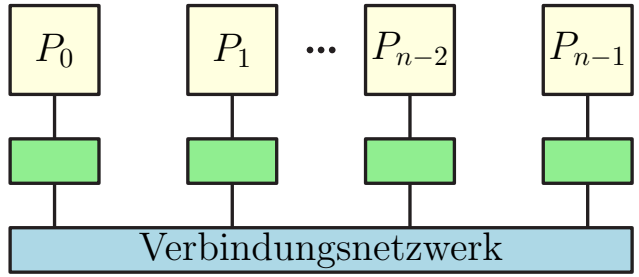
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Verteilter gemeinsamer Speicher

(symmetrisch) gemeinsamer Speicher

B(ulk)S(ynchronous)P(arallel)

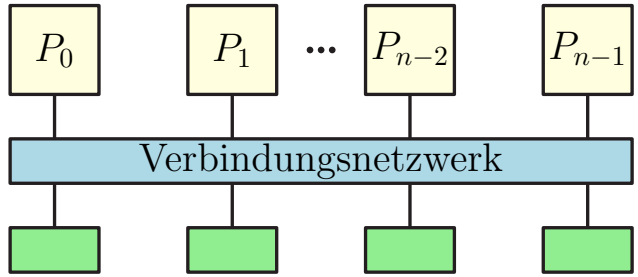
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Verteilter gemeinsamer Speicher

(symmetrisch) gemeinsamer Speicher

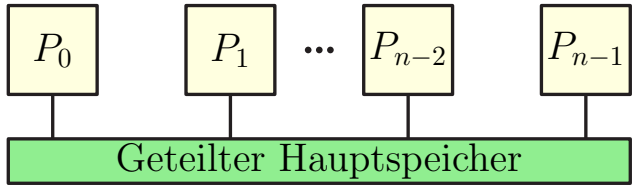
B(ulk)S(ynchronous)P(arallel)

- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

PRAM

Speicherkonflikte

PRAM



- klassifiziert nach Lese- (*read*) und Schreibzugriff (*write*)
- betrachte gleichzeitigen (*concurrent*) oder exklusiven Zugriff (*exclusive*)
  - EREW
  - ERCW (schwachsinn)
  - CREW
  - CRCW
    - **common**: alle müssen den gleichen Wert schreiben
    - **arbitrary**: Wert eines zufälligen Prozessors
    - **priority**: Wert mit kleinster Prozessor-ID
    - **combine**: Aggregation der Werte (z.B. Summe)

Verbindungsnetzwerke

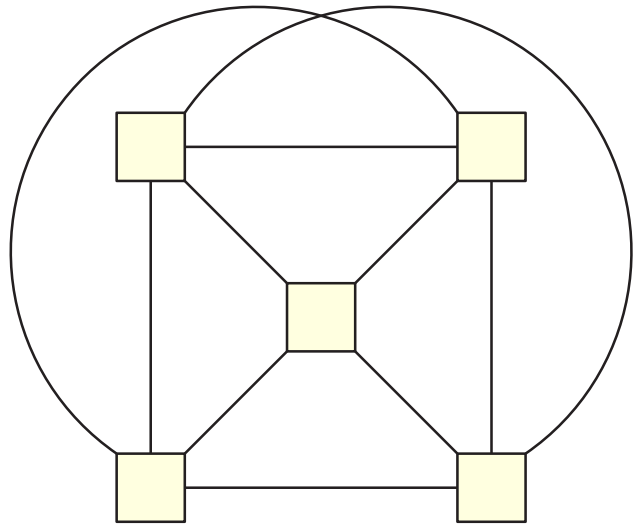
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex
  - Telefon
  - Duplex

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

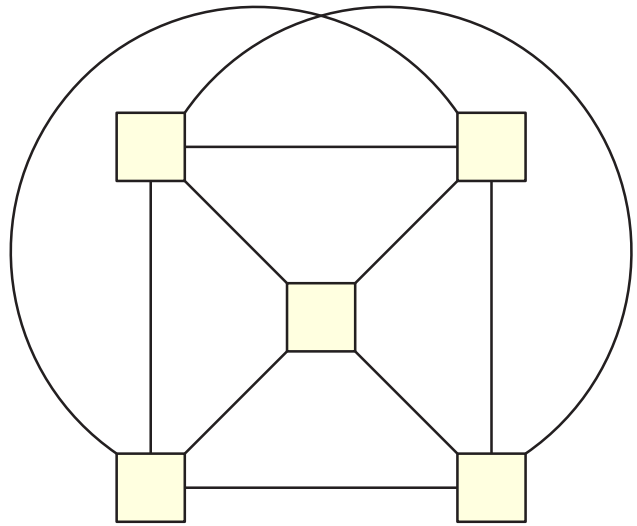
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

$\bullet$
0

Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

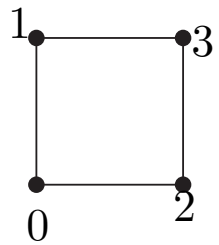
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

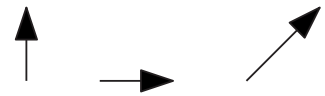
- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

Struktur

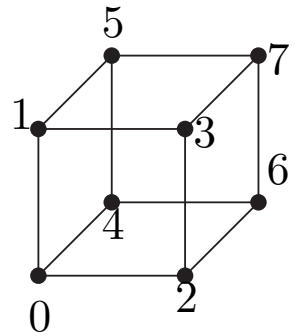
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

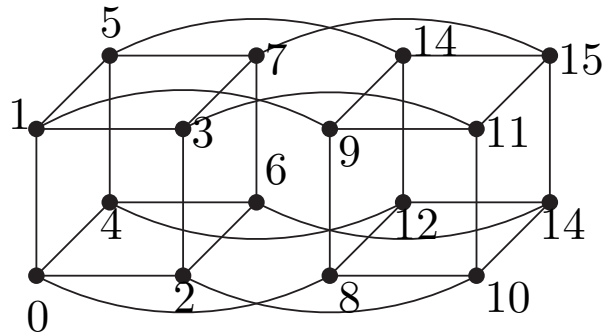
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

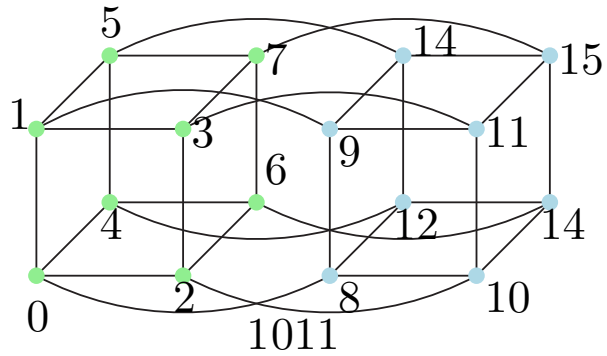
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

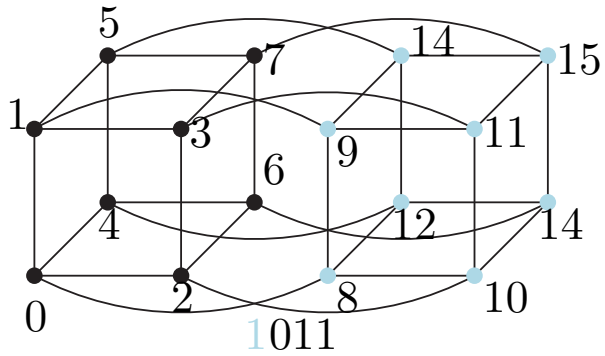
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

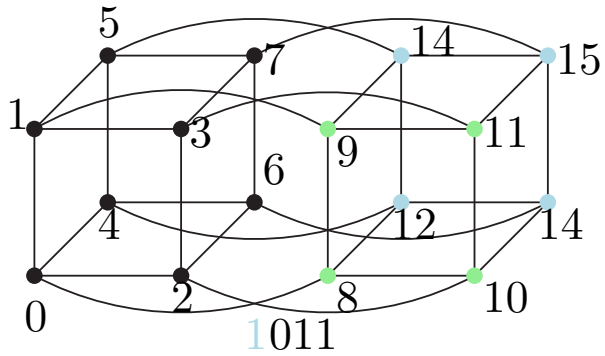
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

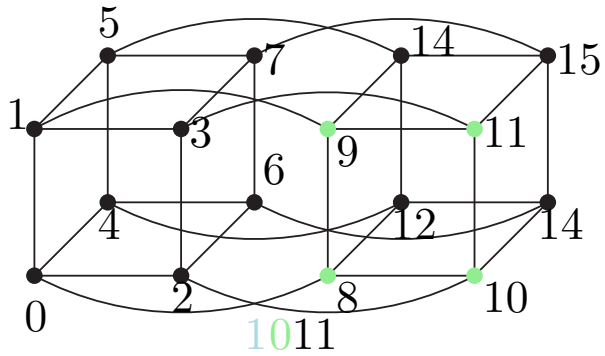
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

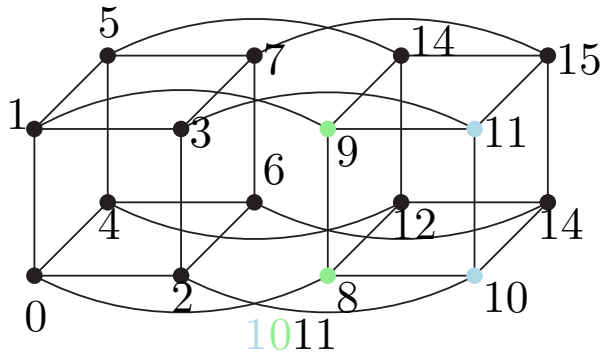
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

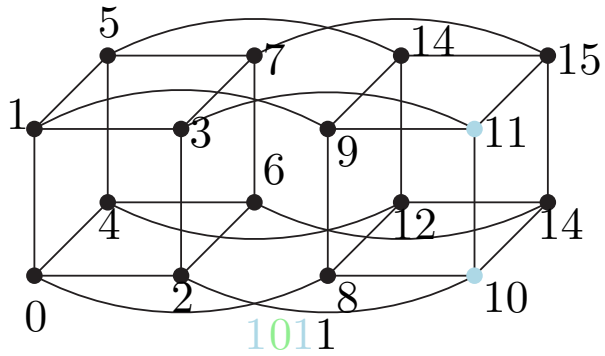
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

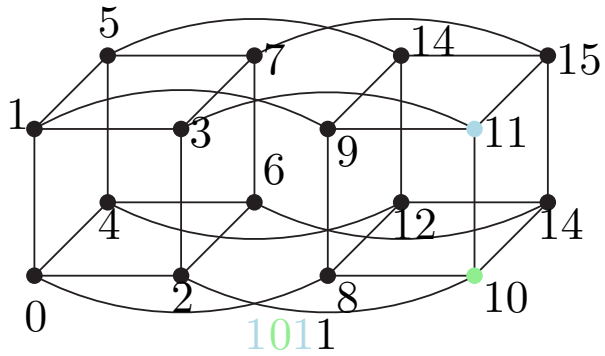
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

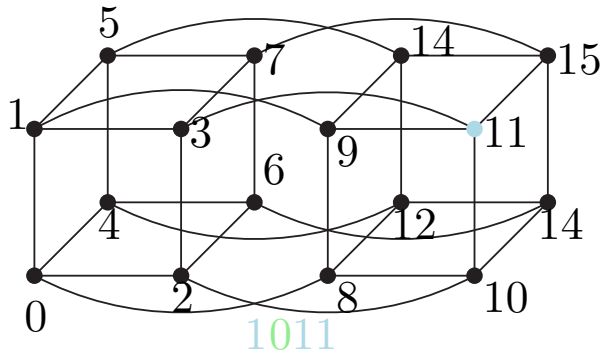
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Anwendungen

PRAM Präfixsumme

- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

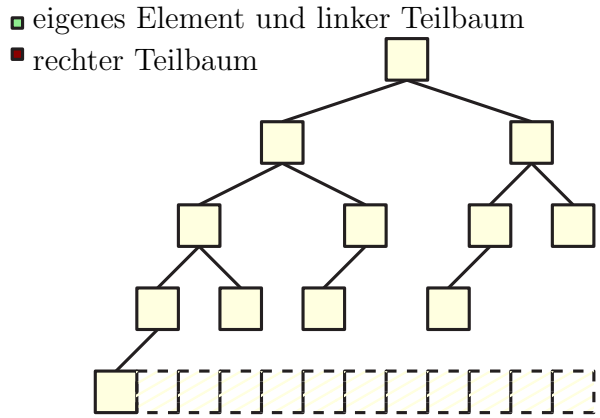
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

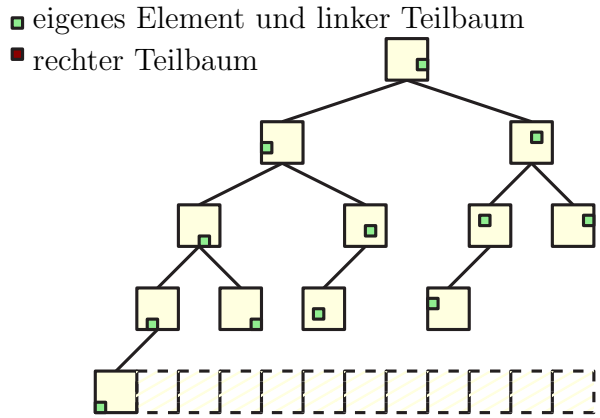
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

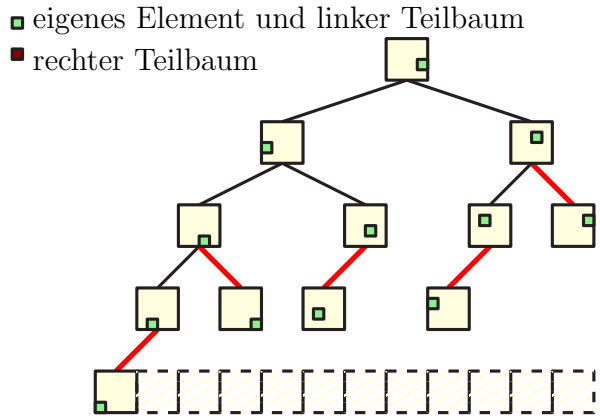
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

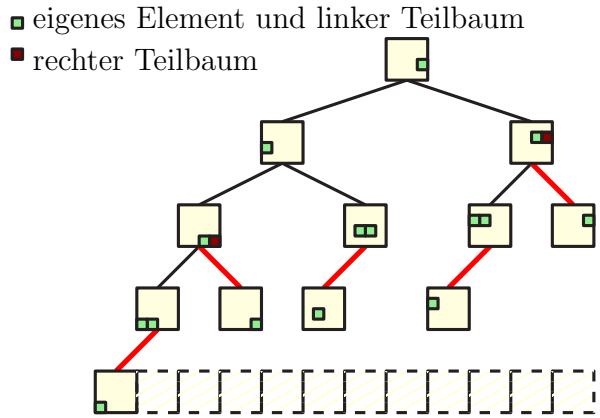
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

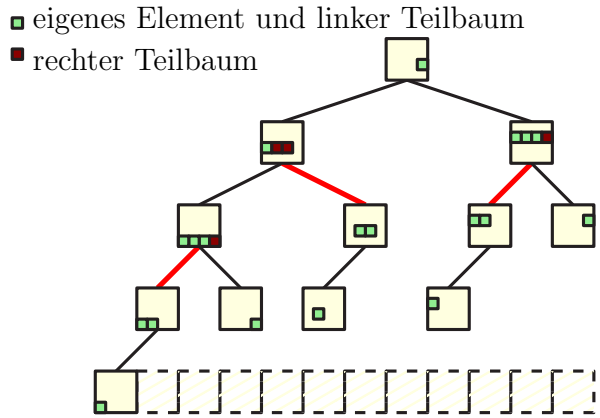
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

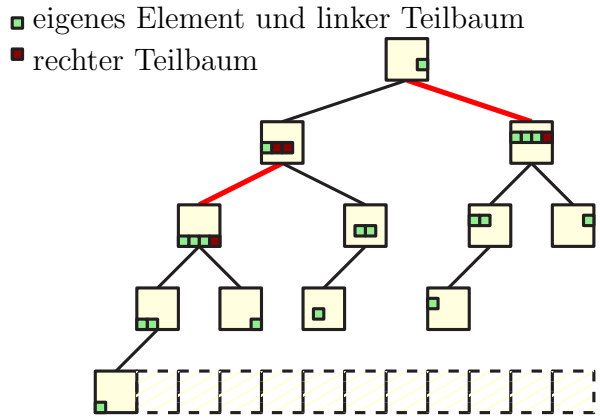
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

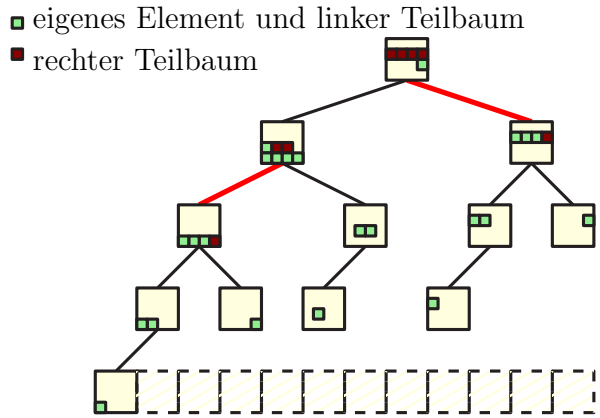
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

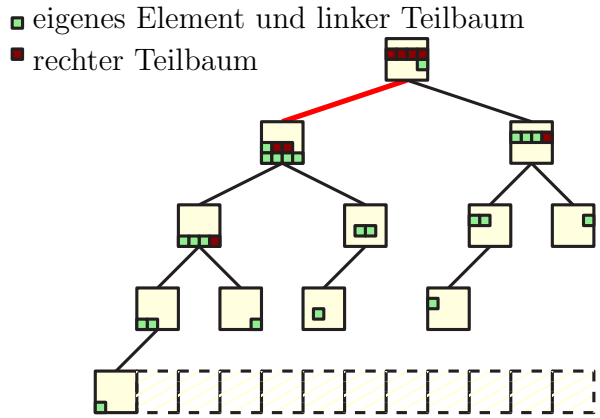
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

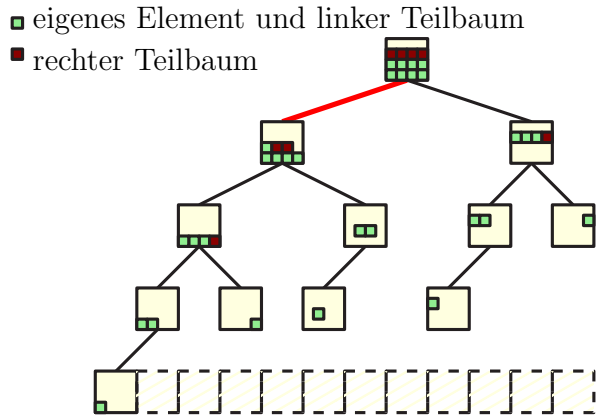
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

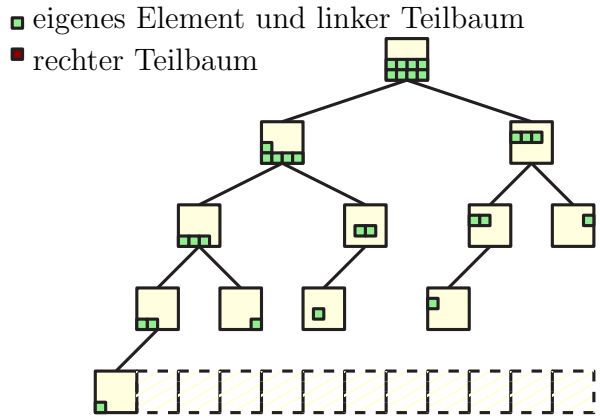
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

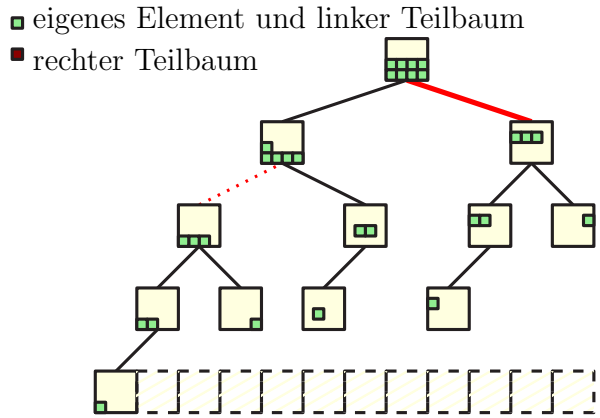
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

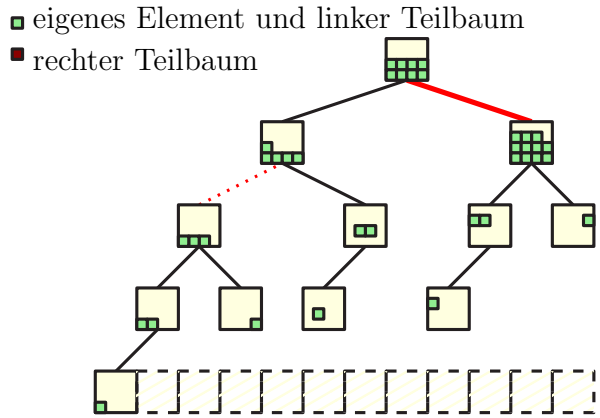
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

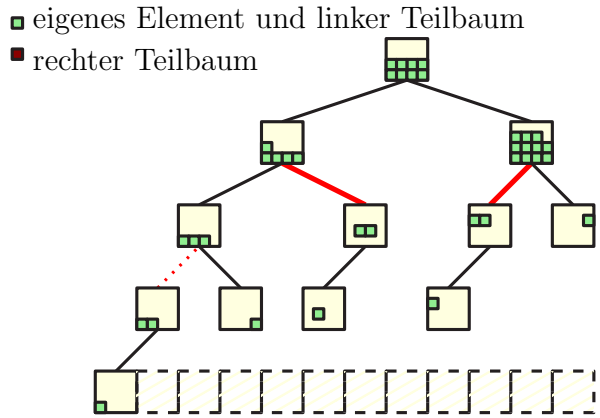
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

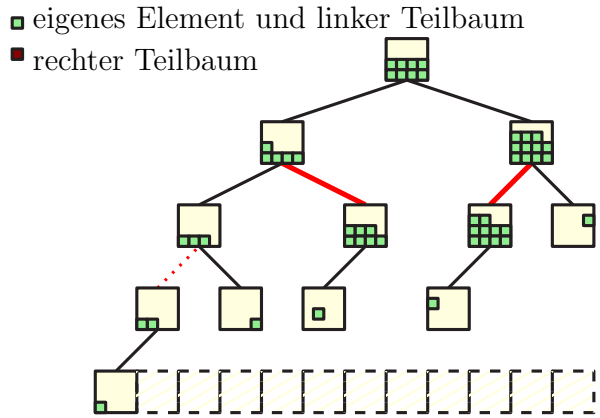
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

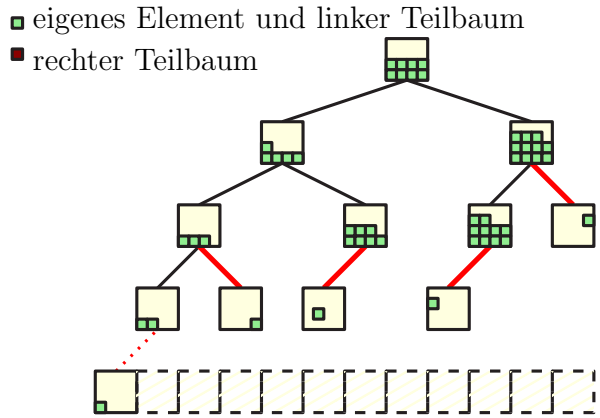
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

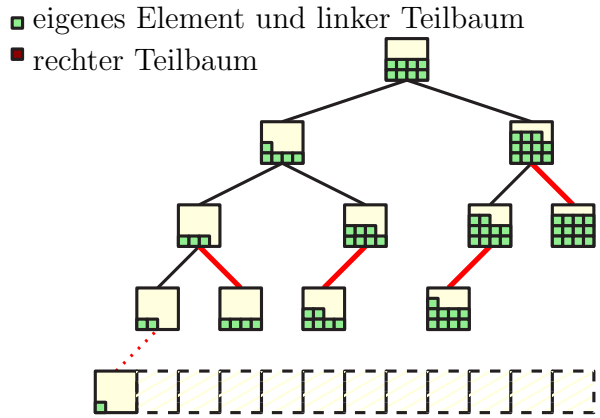
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

PRAM Präfixsumme



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier Beispiel Präfixsumme (Fibonacci-Baum)
- zweiphasig, aufwärts und abwärts

Aufwärtsphase

- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

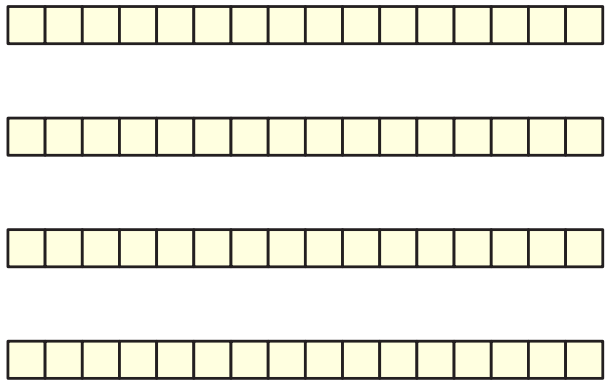
- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

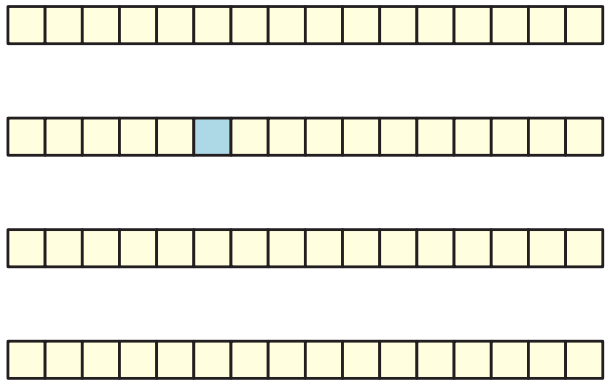


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

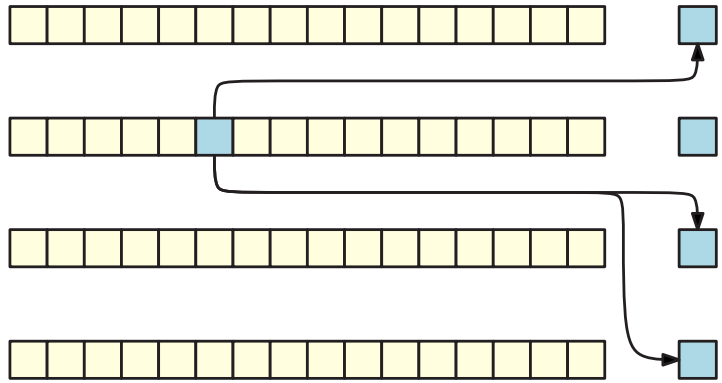


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

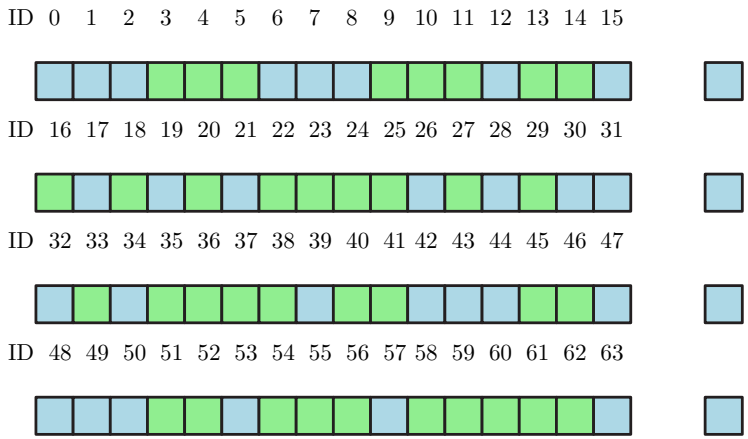


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

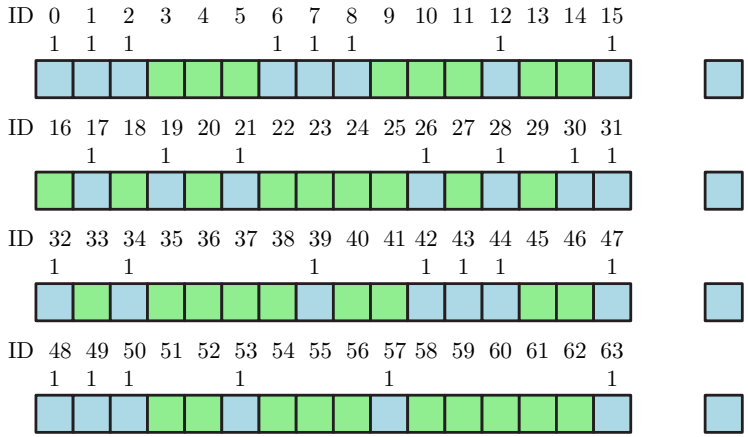


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

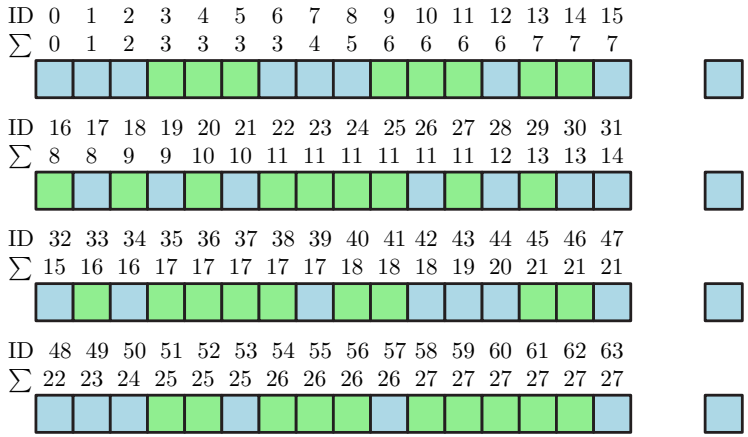


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

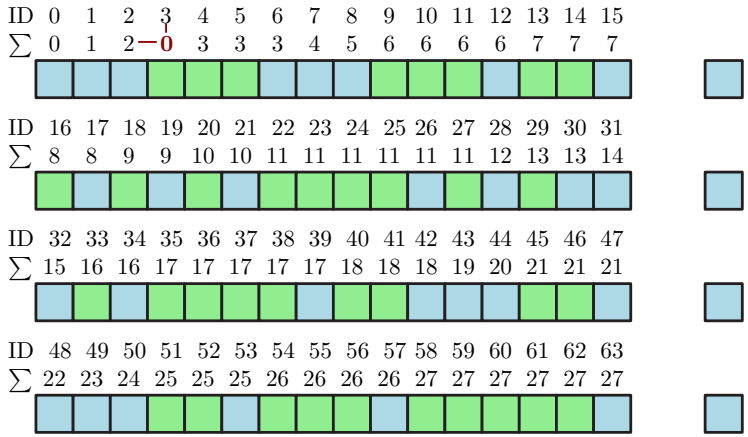


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

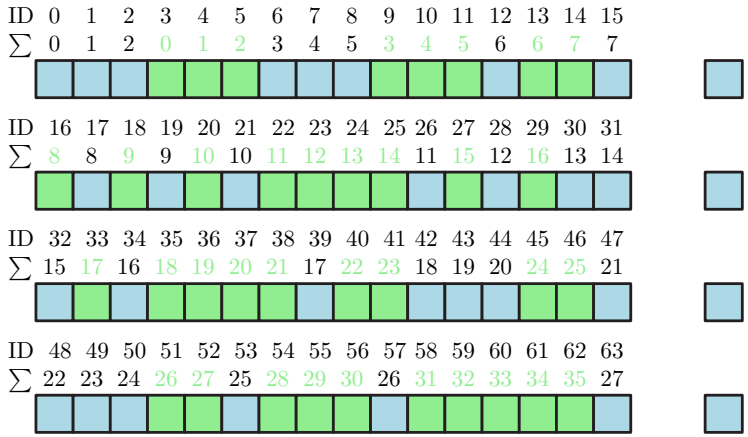


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

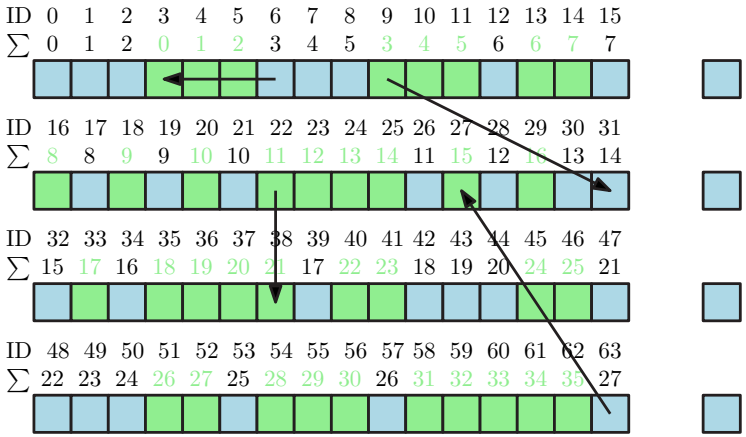


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

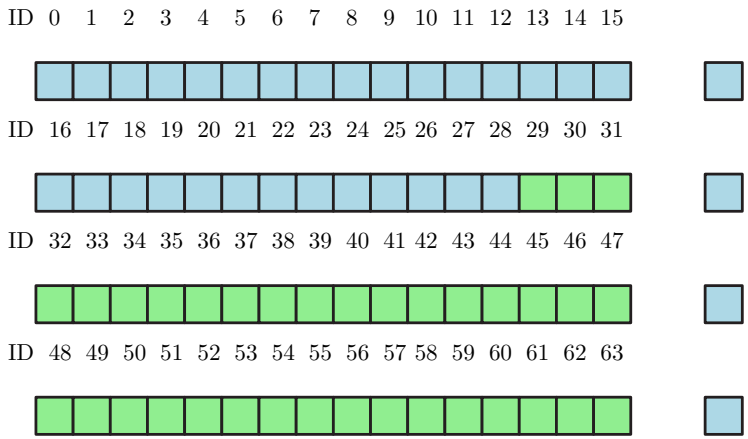


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

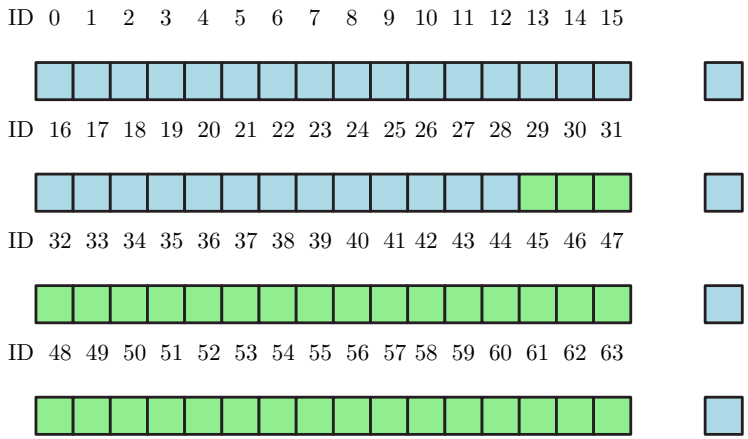


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

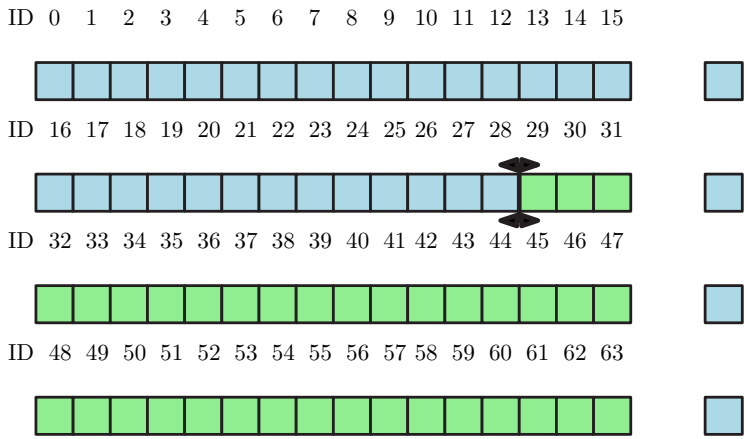


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

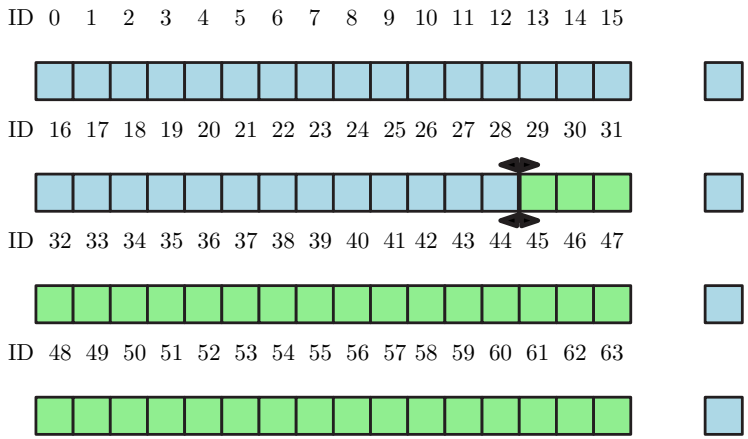
1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion



Anwendungen

Paralleler Quicksort

Rekursives Verfahren



1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten (Problematisch bei unbalancierter Verteilung)
7. parallele Rekursion

Parallele Programmierung

Ein Einstieg

Einstieg in parallele Programmierung?

- openMP
  - `www.openmp.org`
  - enthalten im GCC Compiler
  - Parallelität über Preprozessorflags (#pragma omp parallel)
- Boost Threads
  - `www.boost.org`
  - Bibliothek, eigenes Management
- Grafikkarten (für die, die es anspruchsvoller mögen)
  - `http://developer.nvidia.com/getting-started-parallel-computing`
  - cudafähige Grafikkarte nötig
  - Nebeneffekte durch ungewohnte Grafikkartenhardware

Ende!

Feierabend!

Parametrisierte Algorithmen

Rekurrenzrelationen – Zusammenfassung

$$T(n,k) = a_1 T(n, k-k_1) + a_2 T(n, k-k_2) + \ldots + a_j T(n, k-k_j) \; \{+f(n)\}$$

$(k_i \leq k_j, i < j)$

- Verzweigungsvektor ($a_i = 1$, ohne $f(n)$)
  - $\langle k_1, k_2, \ldots, k_j \rangle$

  $\Rightarrow$ tabelliert, Basis α aus Tabelle ablesen

  $\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(\alpha^k)$ ($\hat{=}$ Größe des Ausführungsbaums)

- erzeugendes Polynom (a_i beliebig, ohne $f(n)$)
  - Ansatz: $T(n,k) = x^k$

  $\Rightarrow$ $x^k - a_1 x^{k-k_1} - a_2 x^{k-k_2} - \ldots - a_j x^{k-k_j} = 0$ (eingesetzt)

  $\Rightarrow$ bestimme *betragsmäßig* größte Nullstelle α (mit Vielfachheit m)

  $\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(k^{m-1}\alpha^k)$

- inhomogene Rekurrenzrelation ($f(n) \hat{=}$ Arbeit in jedem Knoten)
  - Löse homogene Rekurrenz (ohne $f(n)$) $\rightarrow T'(n,k) = r(k)$

  $\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(r(k) \cdot f(n))$

Parametrisierte Algorithmen

Rekurrenzrelationen – Zusammenfassung

$$T(n,k) = a_1 T(n, k-k_1) + a_2 T(n, k-k_2) + \ldots + a_j T(n, k-k_j) \; \{+f(n)\}$$

$(k_i \leq k_j, i < j)$

- Verzweigungsvektor ($a_i = 1$, ohne $f(n)$)
  - $\langle k_1, k_2, \ldots, k_j \rangle$

  $\Rightarrow$ tabelliert, Basis α aus Tabelle ablesen

  $\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(\alpha^k)$ ($\hat{=}$ Größe des Ausführungsbaums)

- erzeugendes Polynom (a_i beliebig, ohne $f(n)$)
  - Ansatz: $T(n,k) = x^k$

  $\Rightarrow$ $x^k - a_1 x^{k-k_1} - a_2 x^{k-k_2} - \ldots - a_j x^{k-k_j} = 0$ (eingesetzt)

  $\Rightarrow$ bestimme *betragsmäßig* größte Nullstelle α (mit Vielfachheit m)

  $\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(k^{m-1}\alpha^k)$

- inhomogene Rekurrenzrelation ($f(n) \hat{=}$ Arbeit in jedem Knoten)
  - Löse homogene Rekurrenz (ohne $f(n)$) $\rightarrow T'(n,k) = r(k)$

  $\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(r(k) \cdot f(n))$

Parametrisierte Algorithmen

Rekurrenzrelationen – Verzweigungsvektor

$$T(n,k) = a_1 T(n, k-k_1) + a_2 T(n, k-k_2) + \ldots + a_j T(n, k-k_j)$$

$(k_i \leq k_j, i < j)$

Verzweigungsvektor $(a_i = 1)$

- $\langle k_1, k_2, \ldots, k_j \rangle$

$\Rightarrow$ tabelliert, Basis α aus Tabelle ablesen

$\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(\alpha^k)$ ($\hat{=}$ Größe des Ausführungsbaums)

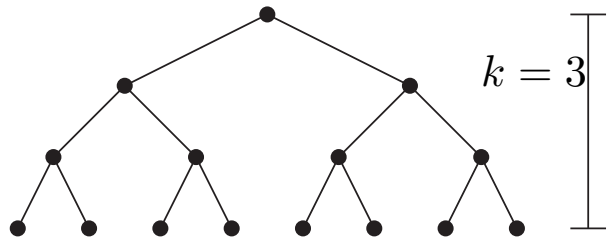
Beispiel

- $T(n,k) = T(n,k-1) + T(n,k-1)$

$\Rightarrow$ Verzweigungsvektor $\langle 1,1 \rangle$

$\Rightarrow$ nach Tabelle $\alpha = 2$

$\Rightarrow$ $T(n,k) = \mathcal{O}(2^k)$



Verzweigungsgrad $\alpha = 2$

Parametrisierte Algorithmen

Rekurrenzrelationen – erzeugendes Polynom

$$T(n,k) = a_1 T(n, k-k_1) + a_2 T(n, k-k_2) + \ldots + a_j T(n, k-k_j)$$

$(k_i \leq k_j, i < j)$

erzeugendes Polynom (a_i beliebig)

- Ansatz: $T(n,k) = x^k$

$\Rightarrow$ $x^k - a_1 x^{k-k_1} - a_2 x^{k-k_2} - \ldots - a_j x^{k-k_j} = 0$ (eingesetzt)

$\Rightarrow$ bestimme *betragsmäßig* größte Nullstelle α (mit Vielfachheit m)

$\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(k^{m-1}\alpha^k)$

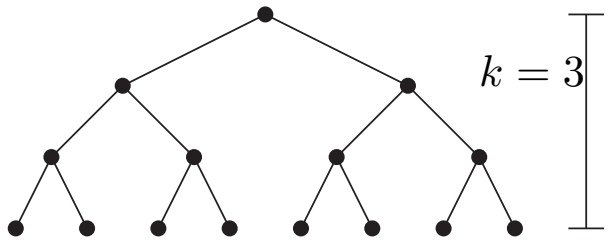
Beispiel

- $T(n,k) = T(n,k-1) + T(n,k-1)$

$\Rightarrow$ $x^k = x^{k-1} + x^{k-1} \Leftrightarrow x^{k-1}(x-2) = 0$

$\Rightarrow$ größte Nullstelle: $x = 2$

$\Rightarrow$ $T(n,k) = \mathcal{O}(2^k)$



Verzweigungsgrad $\alpha = 2$

Parametrisierte Algorithmen

Rekurrenzrelationen – Inhomogenität

$$T(n,k) = a_1 T(n, k-k_1) + a_2 T(n, k-k_2) + \ldots + a_j T(n, k-k_j) + f(n)$$

$(k_i \leq k_j, i < j)$

inhomogene Rekurrenzrelation ($f(n) \mathrel{\hat{=}}$ Arbeit in jedem Knoten)

- Löse homogene Rekurrenz (ohne $f(n)$) $\rightarrow T'(n,k) = r(k)$

$\Rightarrow$ Lösung $T(n,k) = \mathcal{O}(r(k) \cdot f(n))$

Beispiel

- $T(n,k) = T(n,k-1) + T(n,k-1) + n$

$\Rightarrow$ homogene Lösung $T'(n,k) = 2^k$

$\Rightarrow$ $T(n,k) = \mathcal{O}(2^k \cdot n)$

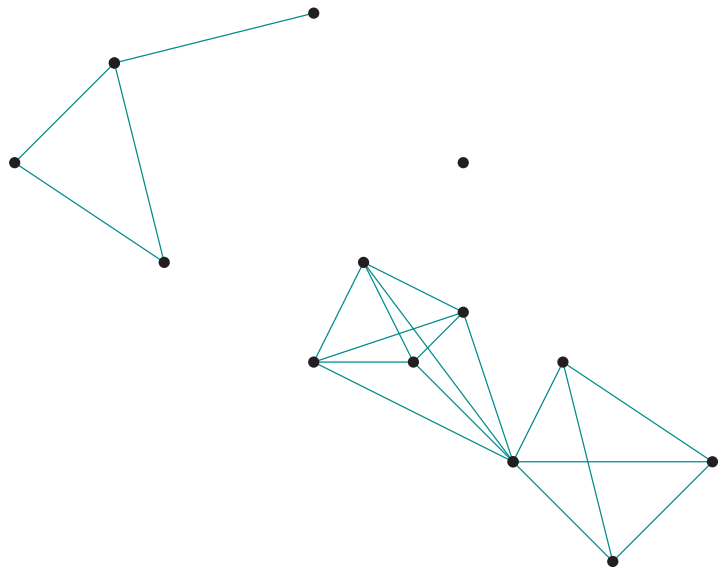
- Test: $\mathcal{O}(2^k \cdot n) = \mathcal{O}(2^{k-1} \cdot n + 2^{k-1} \cdot n + n)$ (Lösung eingesetzt)
  $\Rightarrow \mathcal{O}(2^k) = \mathcal{O}(2^{k-1} + 2^{k-1} + 1)$ (kürzen von n)
  $\Rightarrow \mathcal{O}(2^k) = \mathcal{O}(2^{k-1} + 2^{k-1})$ ($\mathcal{O}$ Kalkül ausnützen)

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

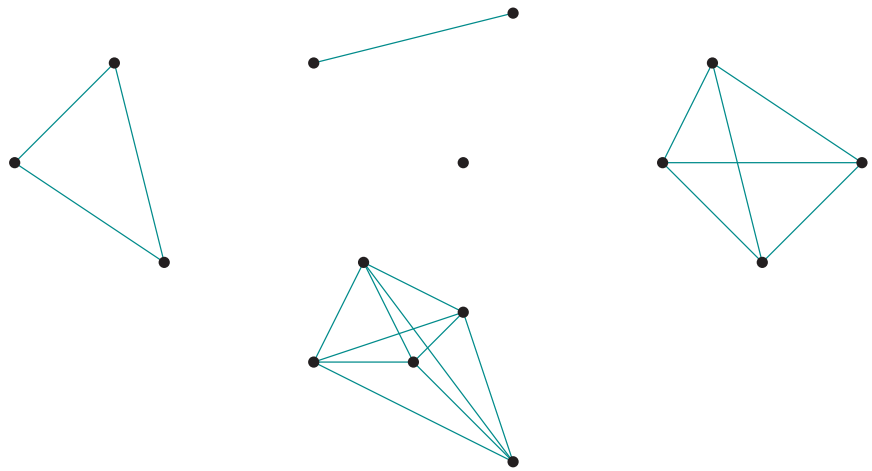
- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

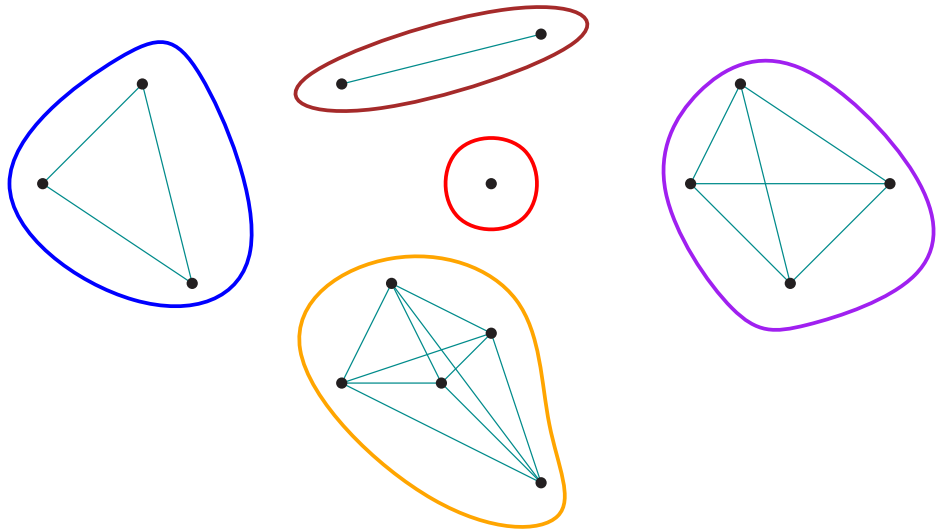
- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

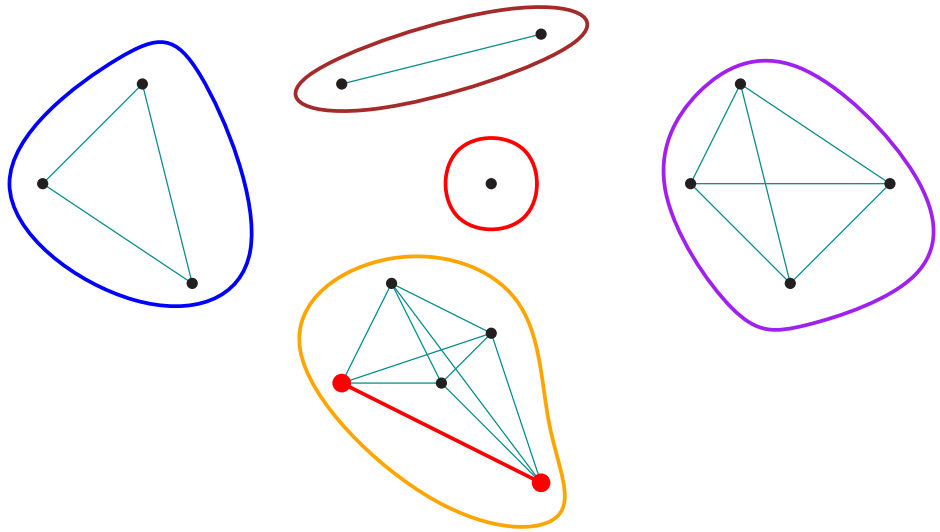
- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

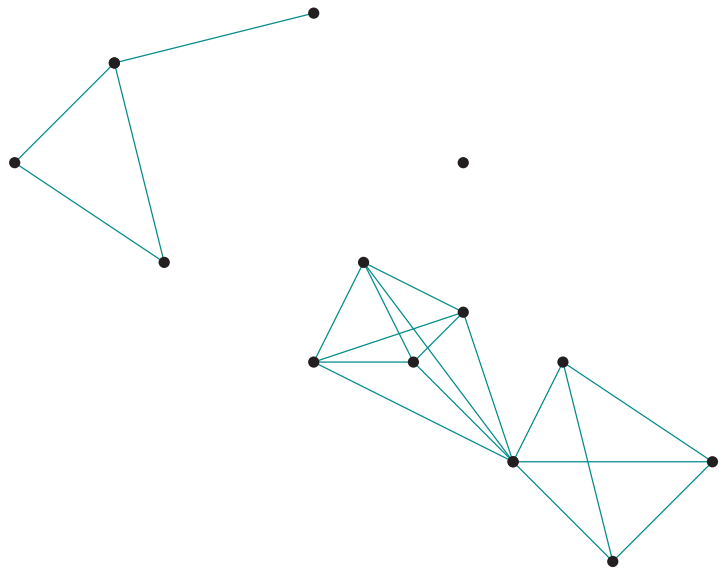
- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

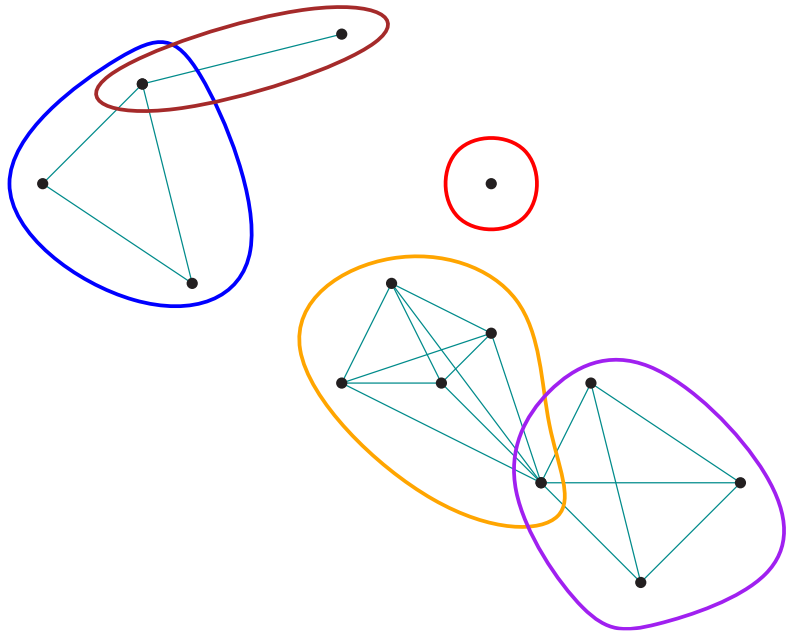
- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

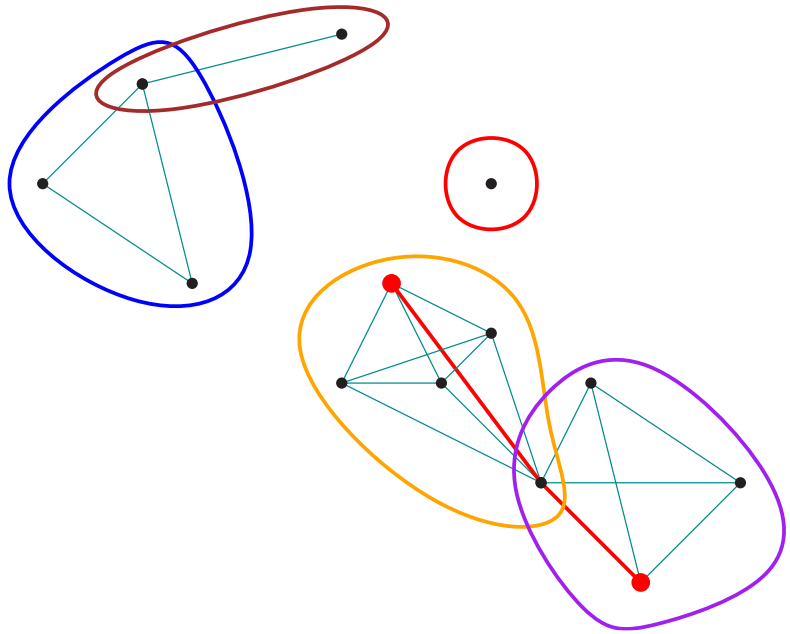
- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

Problemstellung

- gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$
- existiert $S \subset V$, $|S| \leq k$, so dass $(V \setminus S, E)$ einen Clustergraph ist?
  (Clustergraph $\hat{=}$ Graph bestehend aus isolierten Cliquen)



Beobachtung

- Graph ist Clustergraph gdw. alle kürzesten Pfade 2 Knoten besitzen

Parametrisierte Algorithmen

Vertex Cluster Deletion

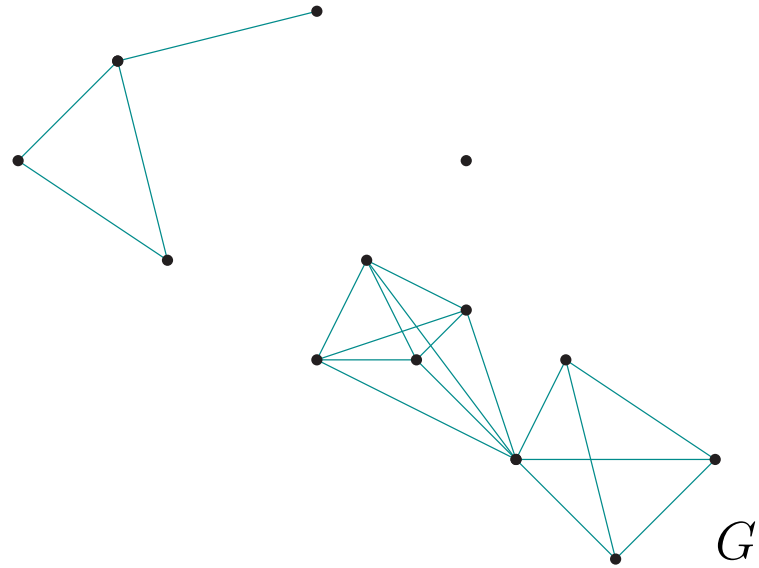
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
($T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3)$)



Parametrisierte Algorithmen

Vertex Cluster Deletion

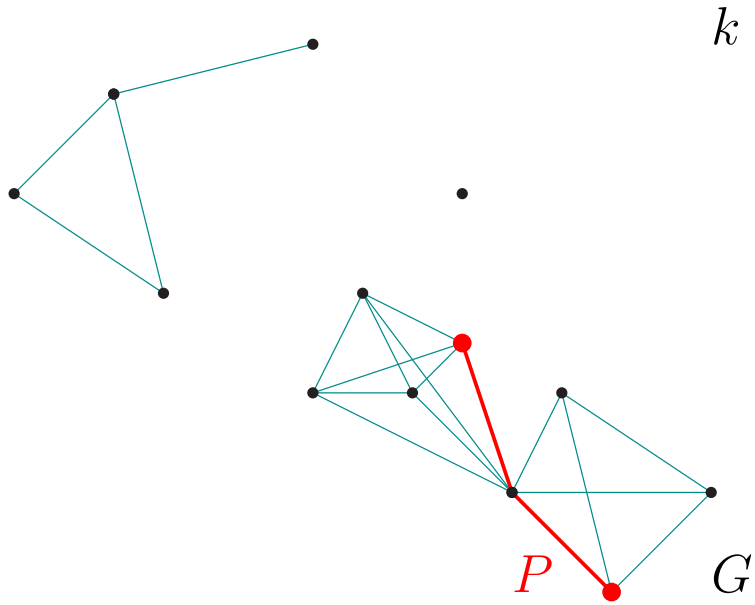
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

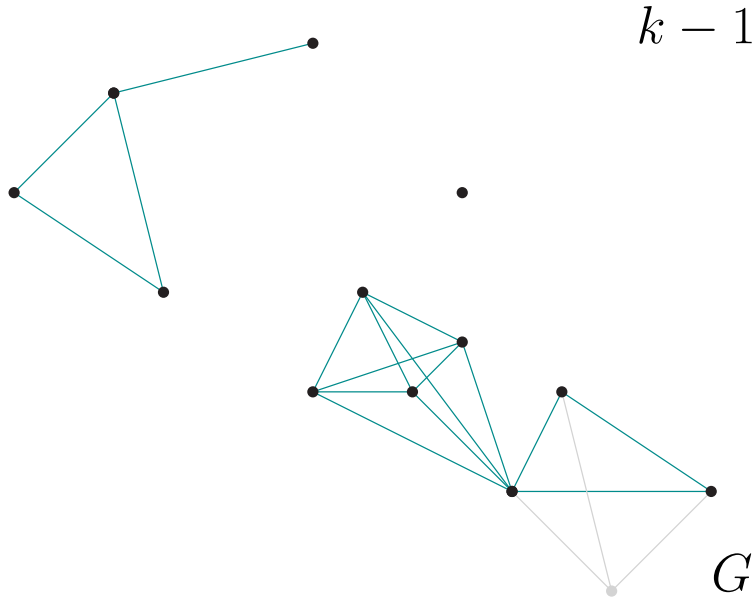
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

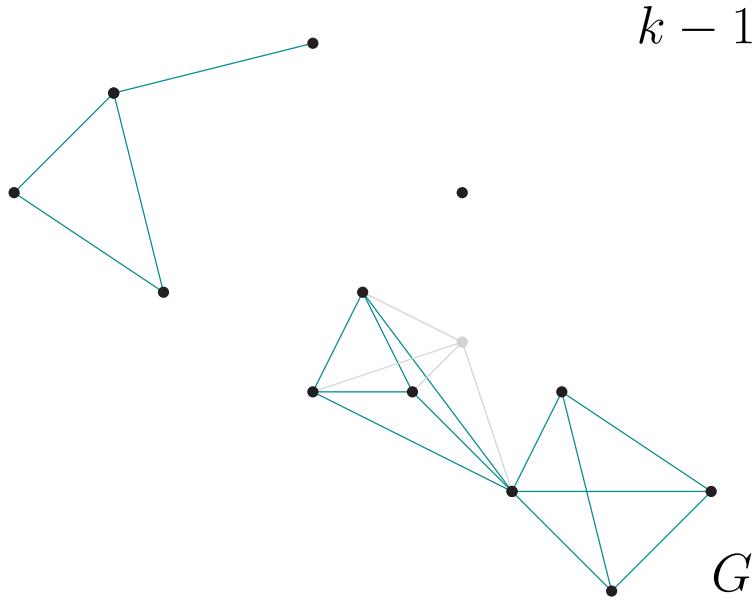
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

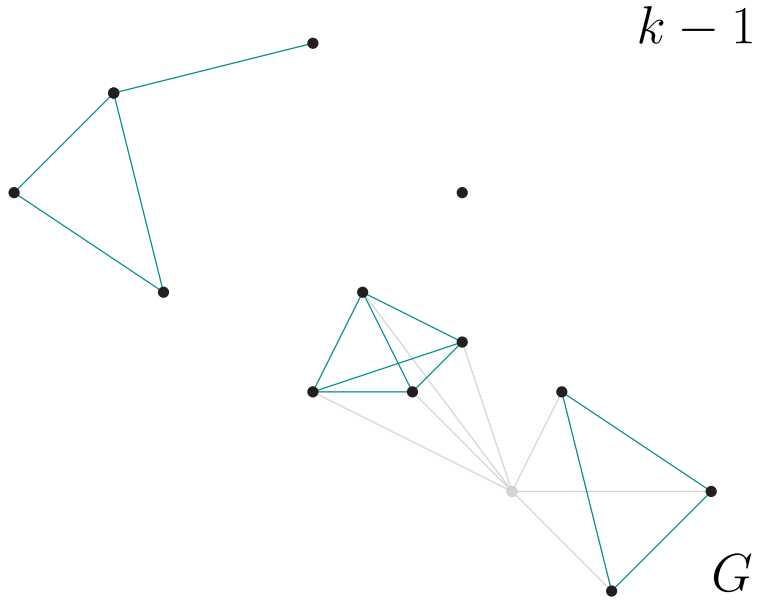
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

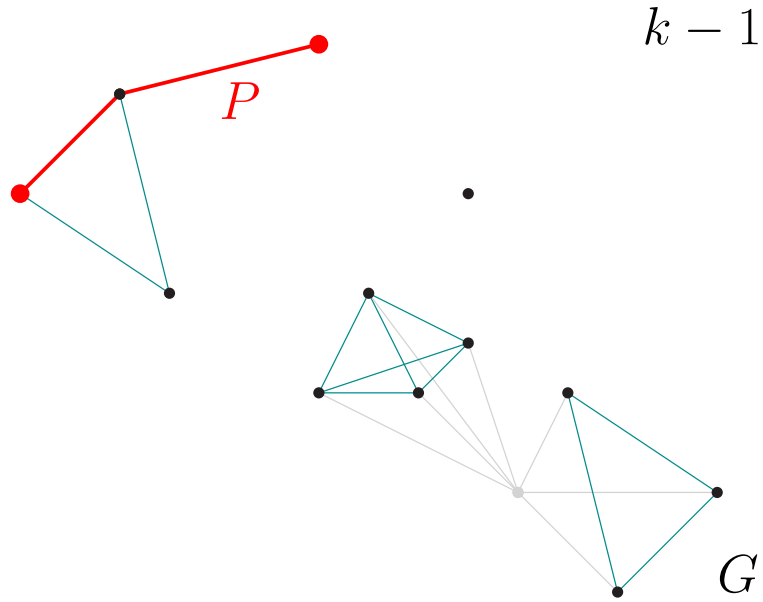
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

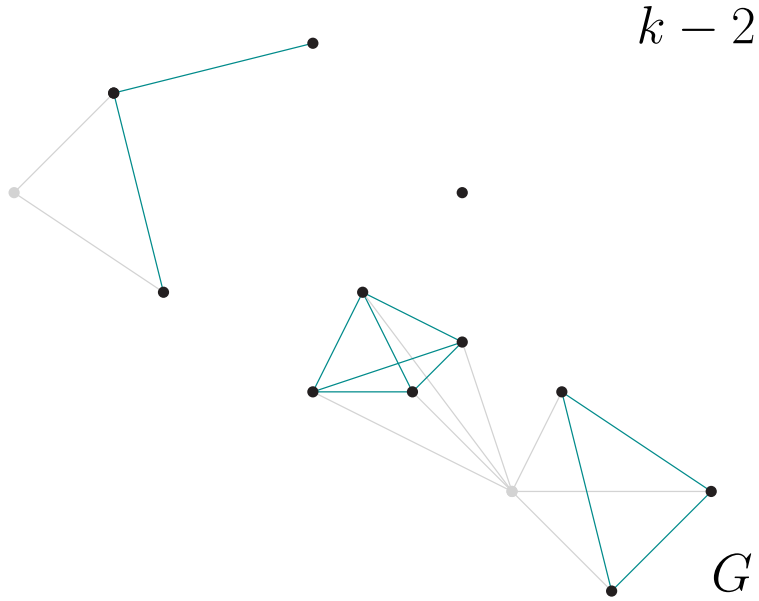
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

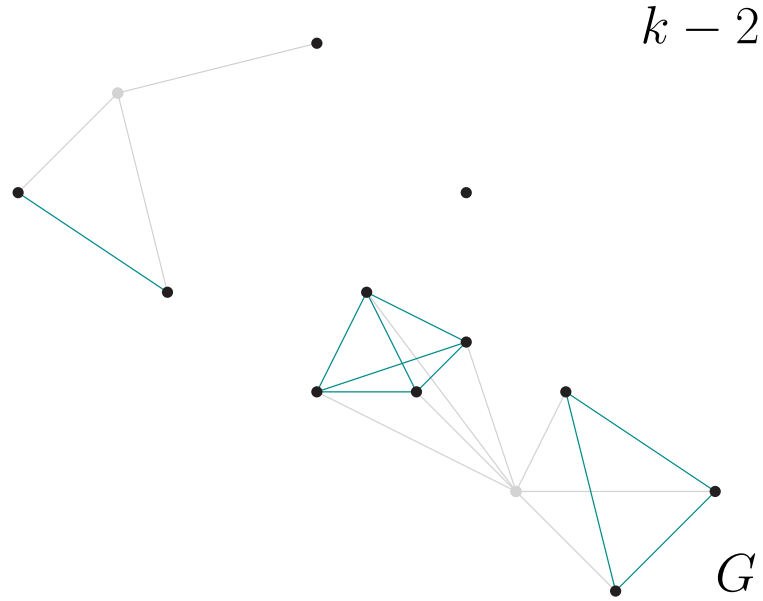
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

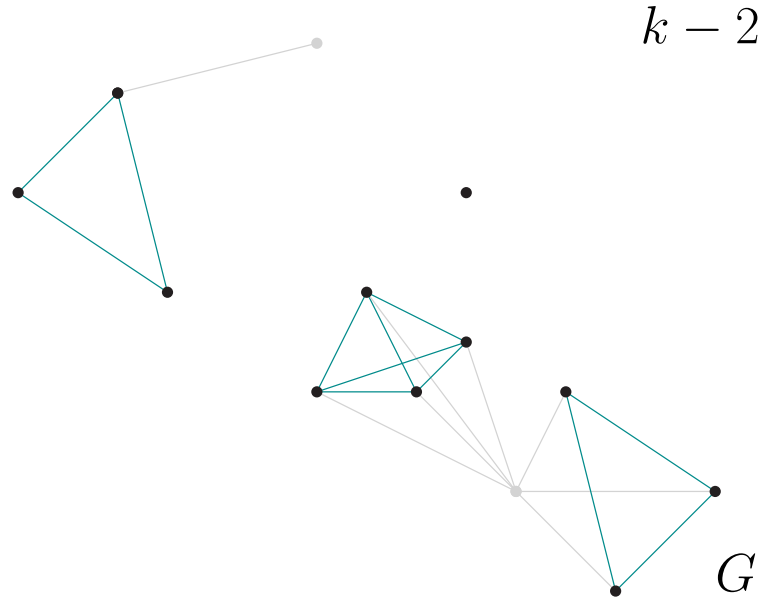
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

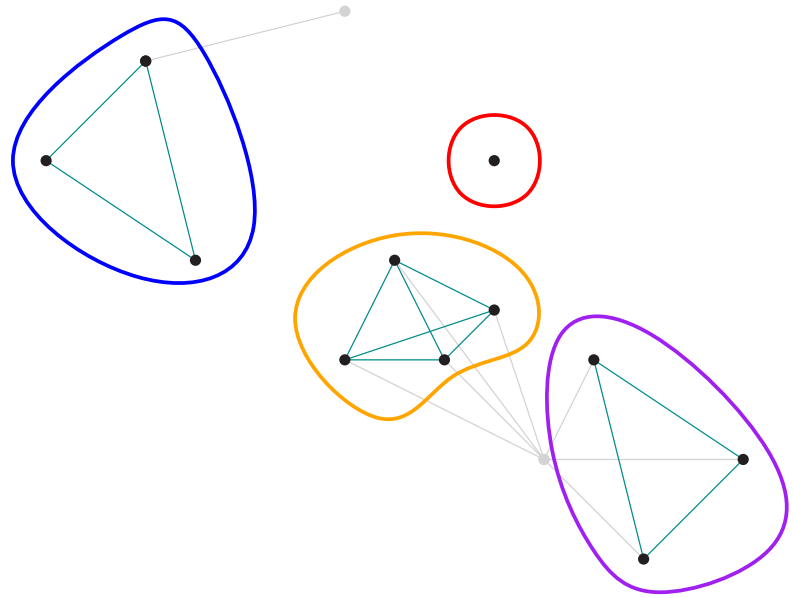
Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$



Parametrisierte Algorithmen

Vertex Cluster Deletion

Algorithmus

- Für $k < 0$, gebe NEIN zurück
- Suche in $\mathcal{O}(n^3)$ nach kürzestem Pfad P mit 3 Knoten
  - existiert keiner, gebe JA zurück
  - sonst, verzweige $3\times$ durch Löschen je eines Knotens von P aus G, wiederhole rekursiv mit $k \to k-1$

Laufzeit

- Baumgröße $\mathcal{O}(3^k)$
  (Suchbaum mit Höhe k, Verzweigungsgrad ≤ 3)
- Arbeit pro Knoten $\mathcal{O}(poly(n))$

$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(3^k poly(n)) \Rightarrow$ FPT
$(T(n,k) = 3T(n,k-1) + \mathcal{O}(n^3))$