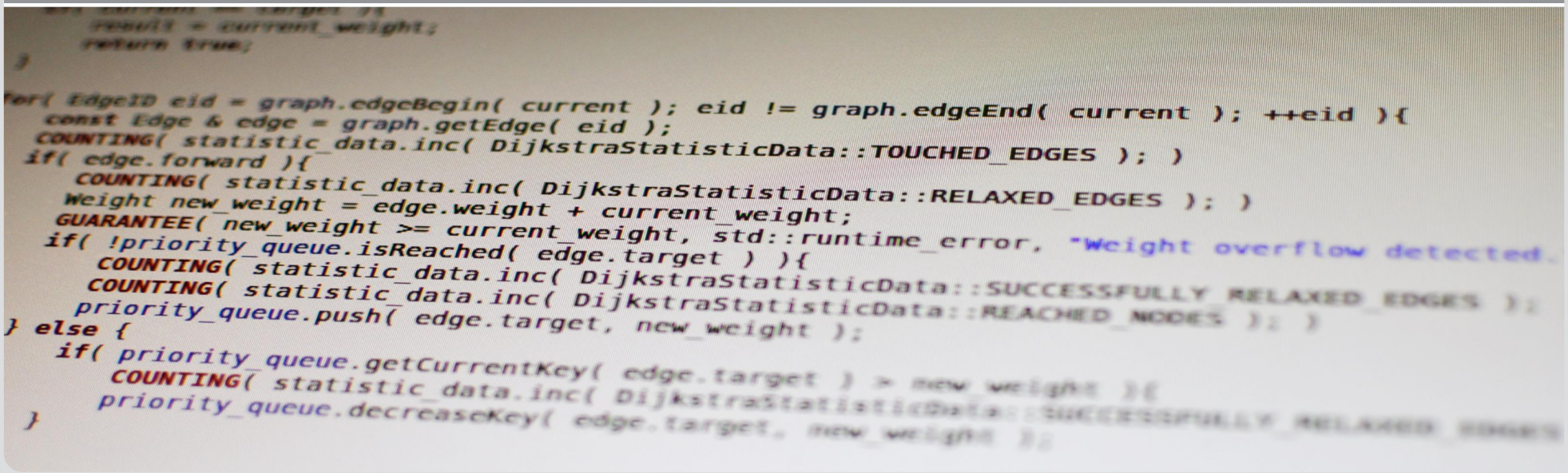
Karlsruher Institut für Technologie

Übung 7 – Algorithmen II

Tobias Heuer, Sebastian Lamm – *tobias.heuer@kit.edu, lamm@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS19.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

Karlsruher Institut für Technologie

Übungsinhalt

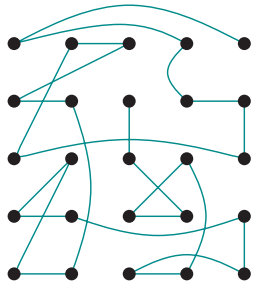
- Approximationsalgorithmen
  (schwierige Probleme gut abschätzen)
  - Grundlagen (Gütemaß, Klassen)
  - Minimum Metric TSP
- Parametrisierte Algorithmen
  (schwierige Probleme in Spezialfällen exakt lösen)
  - *fixed parameter tractable*
  - Beispiel: *Schiebepuzzle*
- Parallelverarbeitung
  - Modelle
  - Verbindungsstrukturen
  - Anwendungen
    - Präfixsumme
    - Paralleles Sortieren
    - Effizienz

Approximationsalgorithmen

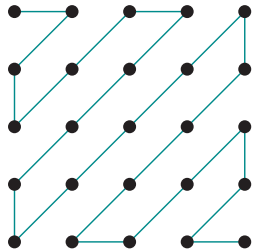
Grundlagen

Warum Lösungen abschätzen?

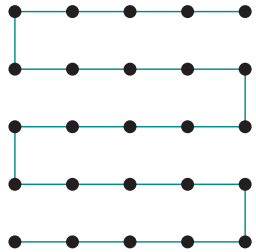
- es gibt “schwierige” Probleme (z.B. TSP mit exponentieller Laufzeit)
  - → exakte Berechnung nicht möglich zu unseren Lebzeiten
- vernünftige Näherungen effizient berechnen
  - → exakte/optimale Ergebnisse nicht immer wichtig
  - → “gute” Lösungen genügen oft
  - → aber Abstand zur korrekten Lösung wissenswert



Weglänge “lang”



Weglänge $8 + 16\sqrt{2}$



Weglänge 24

Approximationsalgorithmen

Gütemaß

- Ein Algorithmus *ALG* hat einen Approximationsfaktor ρ, wenn gilt

$$\frac{w\left(ALG(I)\right)}{w\left(OPT(I)\right)} \leq \rho$$

f.a. Probleminstanzen I, OPT optimale Lösung, w Bewertungsfunktion

(für Minimierungsprobleme $\Rightarrow \rho > 1$, für Maximierungsprobleme $\Rightarrow \rho < 1$)

Beispiel:

- *ALG* schätzt Distanz von Strecke x auf nächste Zweierpotenz $2^{\lceil \log |x| \rceil}$
- *OPT* bestimmt korrekte Distanz $|x|$

$$\Rightarrow \quad \frac{w(ALG)}{w(OPT)} = \frac{2^{\lceil \log |x| \rceil}}{|x|} \leq \frac{2^{\log |x|+1}}{|x|} = \frac{2|x|}{|x|} = 2 = \rho$$

(Zahlenbeispiel: $|x| = 2^{10} + 1 \quad \rightarrow \quad \frac{2^{11}}{2^{10}+1} = 2 - \frac{2}{2^{10}+1} \approx 2$)

Approximationsalgorithmen

Klassen

Approximationsprobleme klassifzierbar durch

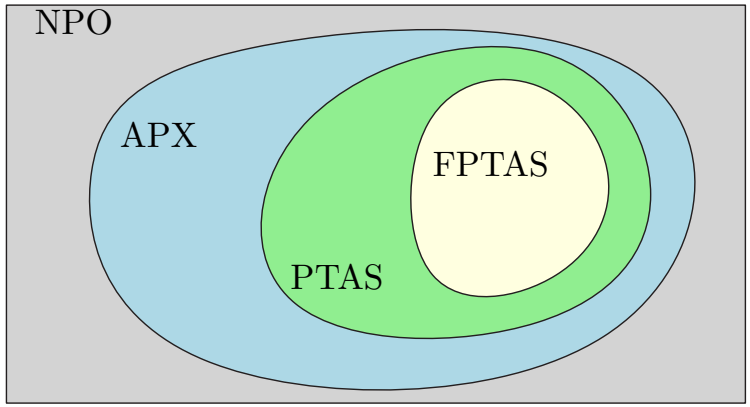
- Laufzeit $T(n, \varepsilon)$
- Approximationsfaktor $\rho(n, \varepsilon)$

Klassen an Approximationsproblemen

APX (*approximable*)
$\rho = const.$, T polynomiell in n

PTAS (*polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n

FPTAS (*fully polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n, $\frac{1}{\varepsilon}$



Beispiele:

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 2$

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 1 + \varepsilon$

$\Rightarrow$ $T(n, \varepsilon) = \frac{1}{\varepsilon} n$, $\rho(n, \varepsilon) = 1 + 2\varepsilon$

Approximationsalgorithmen

Klassen

Approximationsprobleme klassifzierbar durch

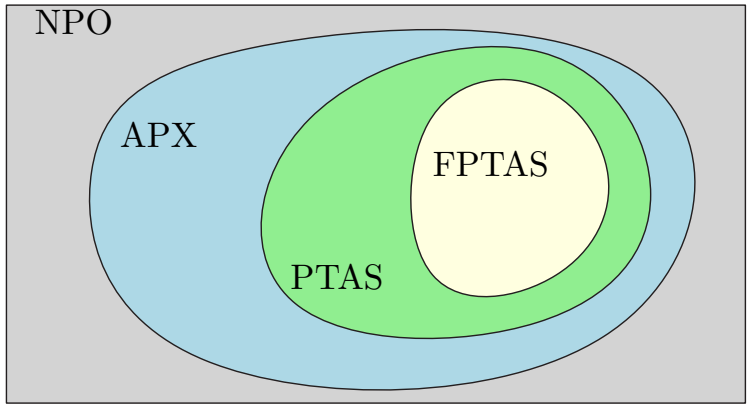
- Laufzeit $T(n, \varepsilon)$
- Approximationsfaktor $\rho(n, \varepsilon)$

Klassen an Approximationsproblemen

APX (*approximable*)
$\rho = const.$, T polynomiell in n

PTAS (*polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n

FPTAS (*fully polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n, $\frac{1}{\varepsilon}$



Beispiele:

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 2$ (APX)

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 1 + \varepsilon$

$\Rightarrow$ $T(n, \varepsilon) = \frac{1}{\varepsilon} n$, $\rho(n, \varepsilon) = 1 + 2\varepsilon$

Approximationsalgorithmen

Klassen

Approximationsprobleme klassifzierbar durch

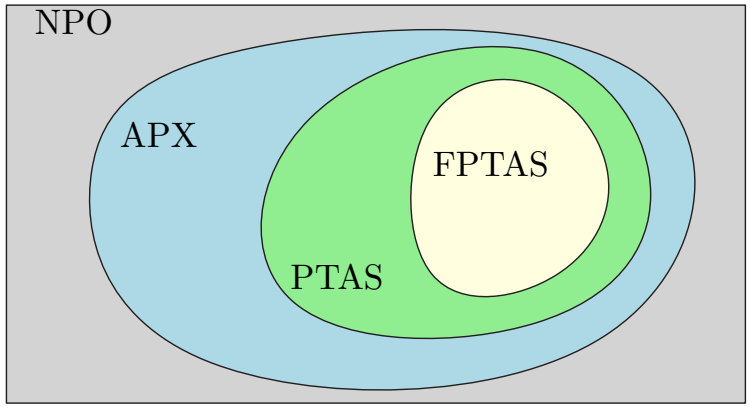
- Laufzeit $T(n, \varepsilon)$
- Approximationsfaktor $\rho(n, \varepsilon)$

Klassen an Approximationsproblemen

APX (*approximable*)
$\rho = const.$, T polynomiell in n

PTAS (*polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n

FPTAS (*fully polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n, $\frac{1}{\varepsilon}$



Beispiele:

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 2$ (APX)

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 1 + \varepsilon$ (PTAS)

$\Rightarrow$ $T(n, \varepsilon) = \frac{1}{\varepsilon} n$, $\rho(n, \varepsilon) = 1 + 2\varepsilon$

Approximationsalgorithmen

Klassen

Approximationsprobleme klassifzierbar durch

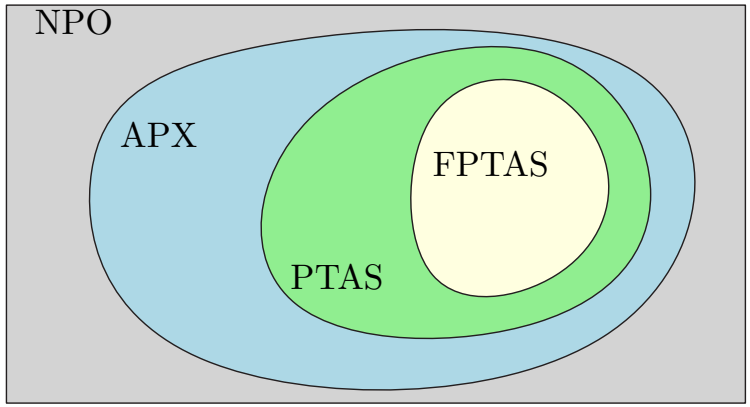
- Laufzeit $T(n, \varepsilon)$
- Approximationsfaktor $\rho(n, \varepsilon)$

Klassen an Approximationsproblemen

APX (*approximable*)
$\rho = const.$, T polynomiell in n

PTAS (*polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n

FPTAS (*fully polynomial time approximation scheme*)
$\rho = 1 \pm \varepsilon$, bel. $\varepsilon \in (0, 1)$, T poly. in n, $\frac{1}{\varepsilon}$



Beispiele:

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 2$ (APX)

$\Rightarrow$ $T(n, \varepsilon) = n^{\frac{1}{\varepsilon}}$, $\rho(n, \varepsilon) = 1 + \varepsilon$ (PTAS)

$\Rightarrow$ $T(n, \varepsilon) = \frac{1}{\varepsilon} n$, $\rho(n, \varepsilon) = 1 + 2\varepsilon$ (FPTAS)

Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Problemstellung

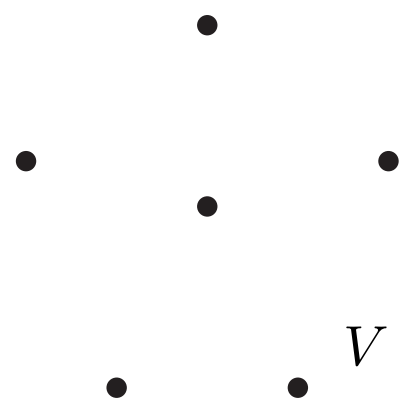
- Gegeben eine Menge an Punkten V in der Ebene
- Vollständiger Graph
- Kantengewichte erfüllen Dreiecks-Ungleichung
- Finde Kreis minimaler Länge der alle Punkte abläuft

Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

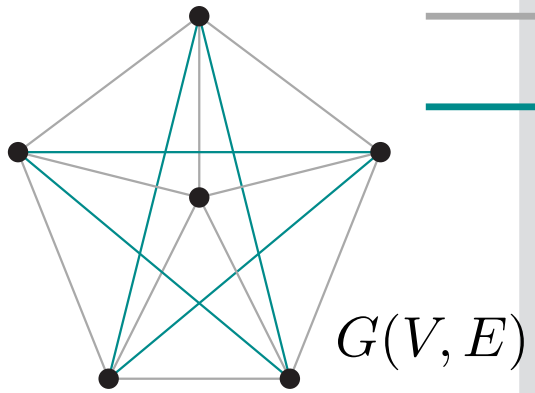


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

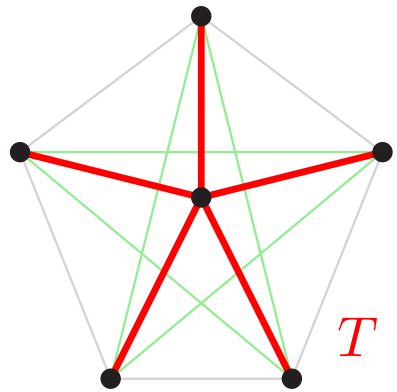


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

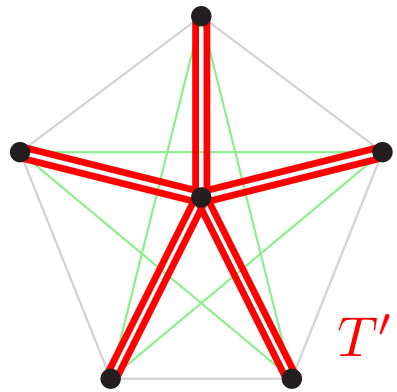


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

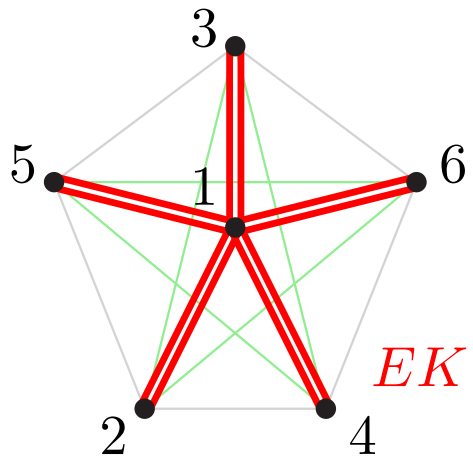


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

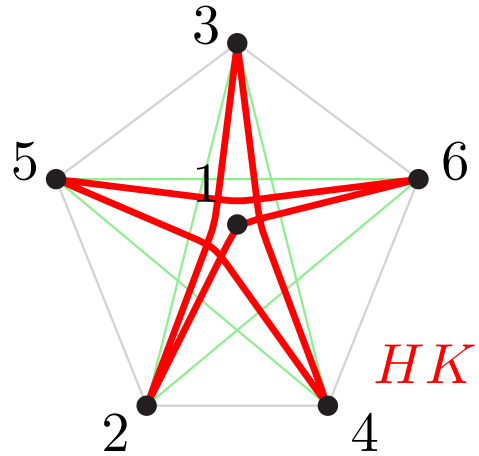


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

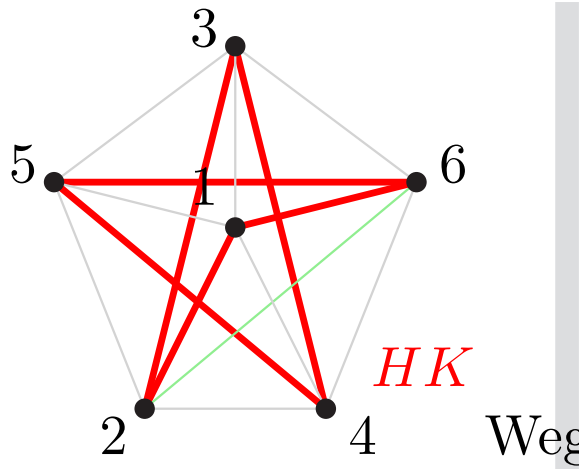


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

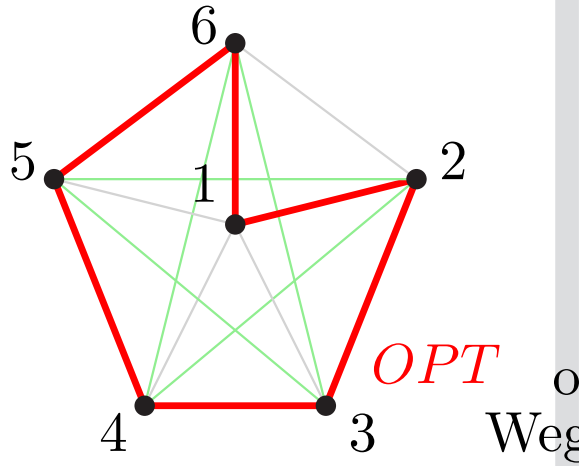


Approximationsalgorithmen

Minimum Metric TSP (NP-hart)

Wiederholung 2-Approximation (Algorithmus)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T $(w(T) \leq w(OPT))$
- verdopple Kanten von $T \rightarrow T'$ $(w(T') \leq 2w(OPT))$
- bestimme Eulerkreis EK auf T' $(w(EK) = w(T'))$
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) \leq 2w(OPT))$

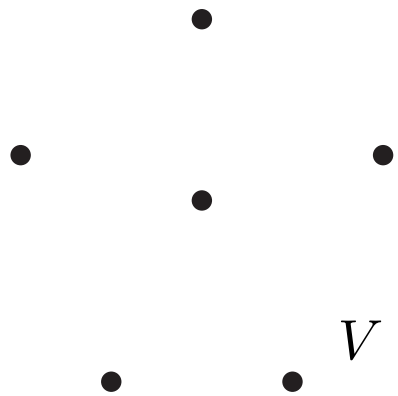


Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



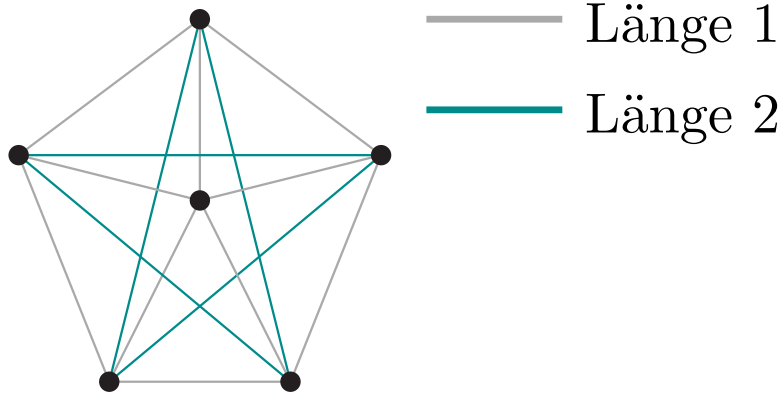
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



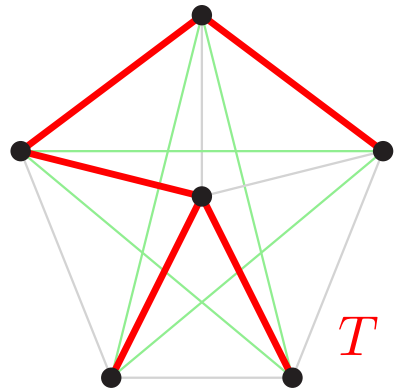
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



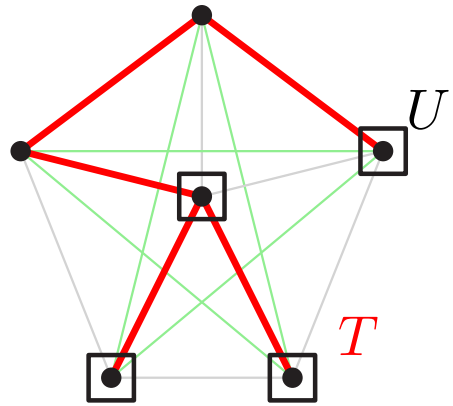
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



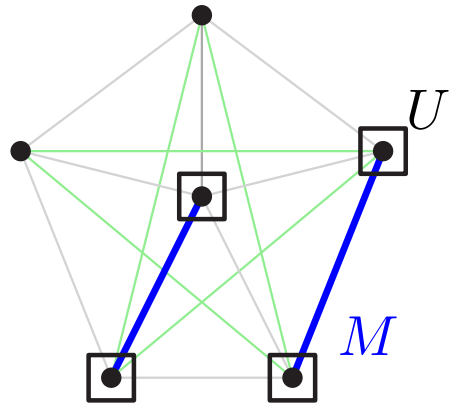
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



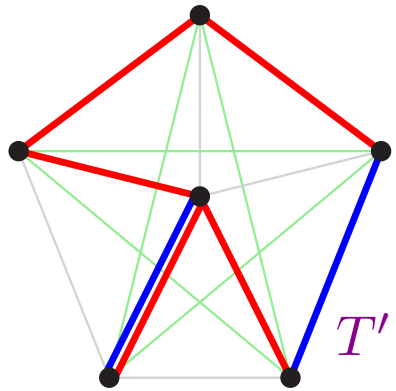
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



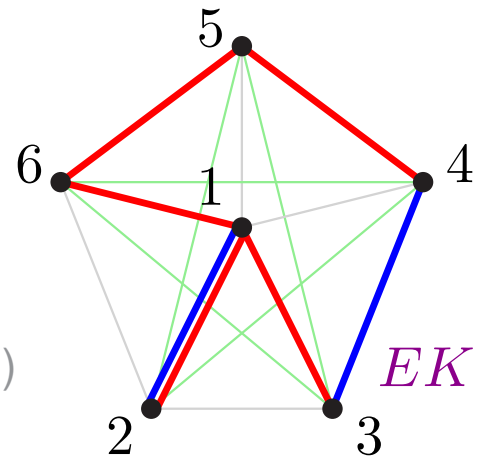
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



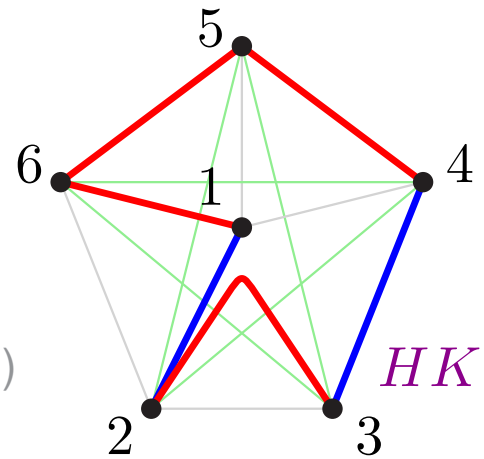
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



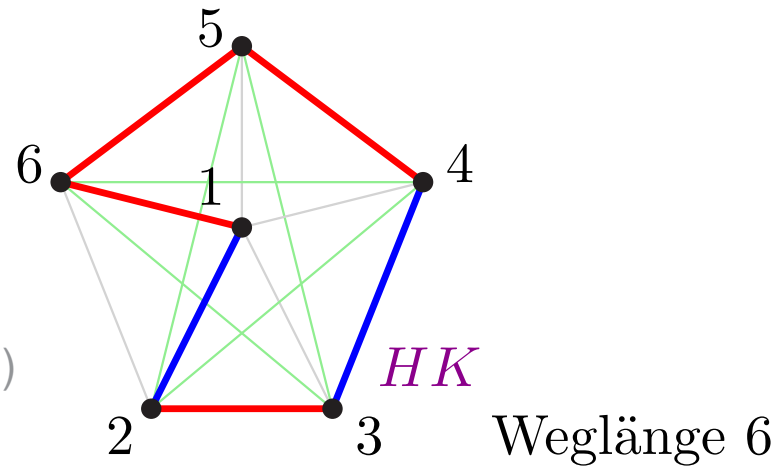
(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Algorithmus) (Christofides-Heuristik)

- Graph $G = (V, E)$ (vollständig, ungerichtet)
- bestimme MST T
- bestimme Knoten U mit ungeradem Grad in T
- finde *minimales perfektes Matching* M auf (U, E)
  (alle Knoten gematcht, Summe der Gewichte der Matchingkanten minimal)
- füge Kanten M zu $T \rightarrow T'$
  $(w(T') = w(T) + w(M) \leq w(OPT) + w(M))$
- bestimme Eulerkreis EK auf T'
  (alle Knoten haben geraden Grad)
- wandle EK zu Hamiltonkreis HK
  $(w(HK) \leq w(EK) = w(T') \leq w(OPT) + w(M))$



(Laufzeit durch Matching dominiert, $\mathcal{O}(n^3)$)

Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

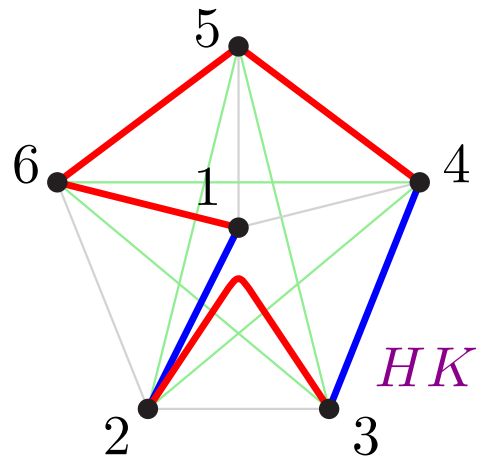
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

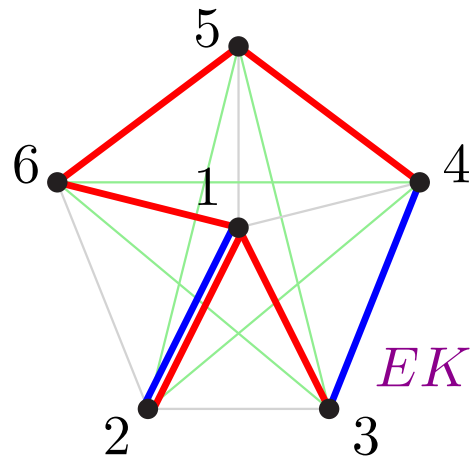
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

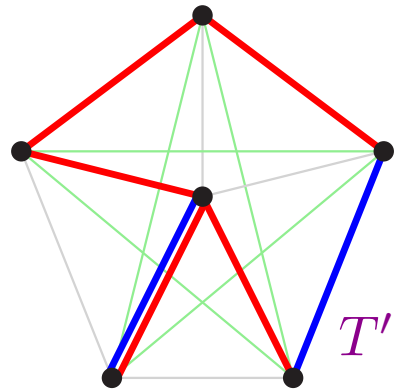
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

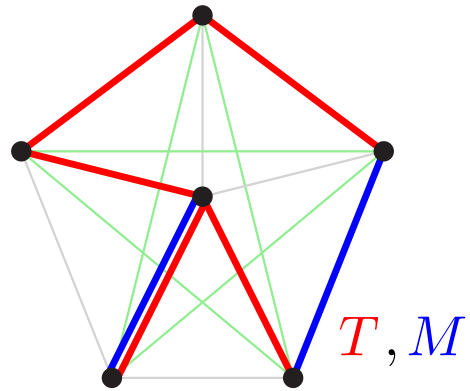
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

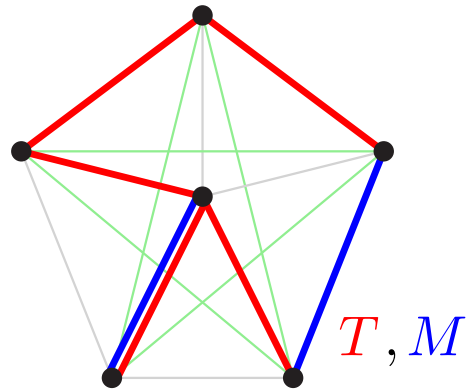
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

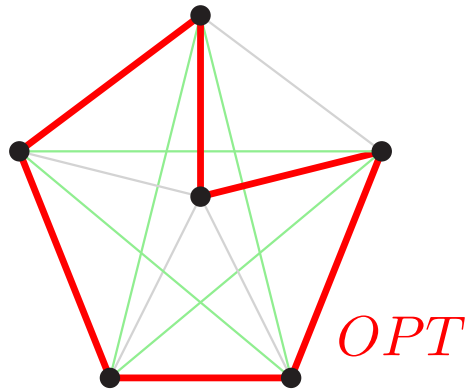
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

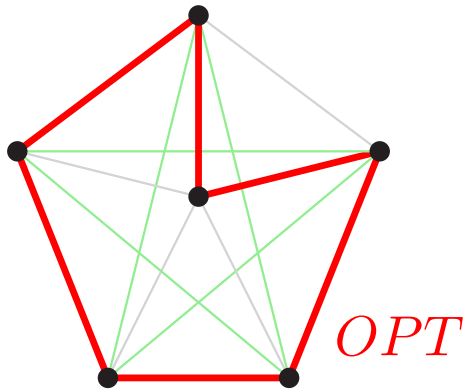
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

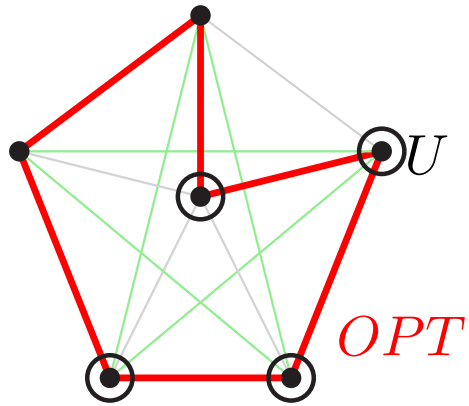
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

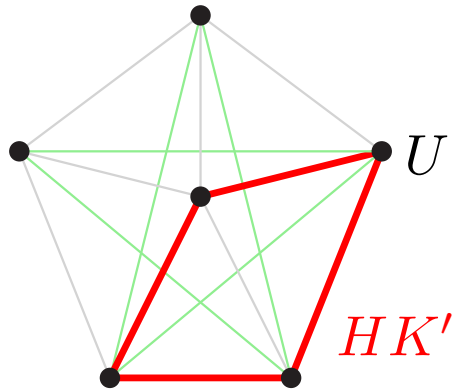
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

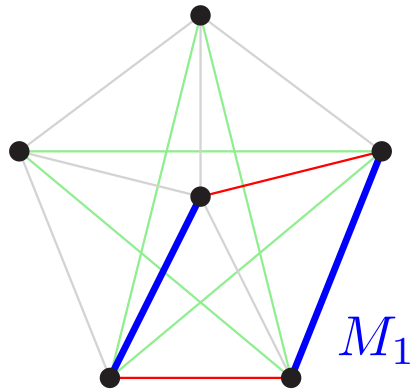
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

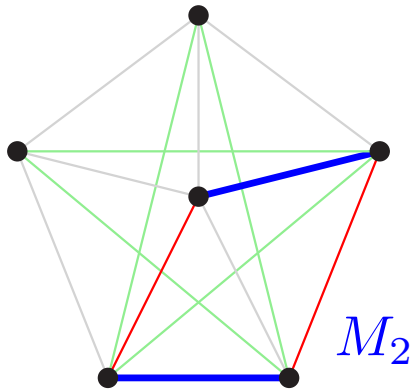
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

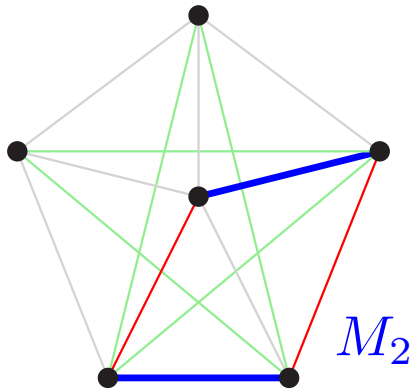
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

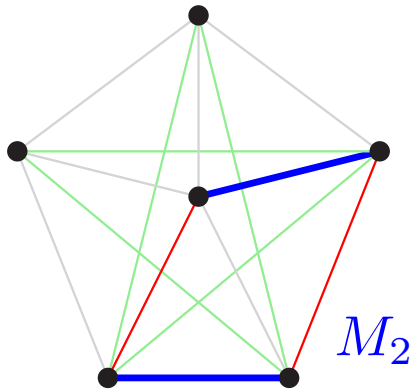
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

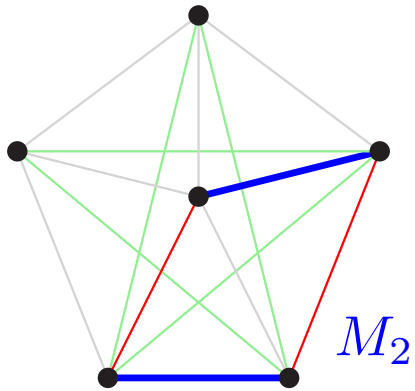
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



Approximationsalgorithmen

Minimum Metric TSP

3/2-Approximation (Beweis) (Christofides-Heuristik)

- Abschätzungen: (analog zu vorherigem Beweis)

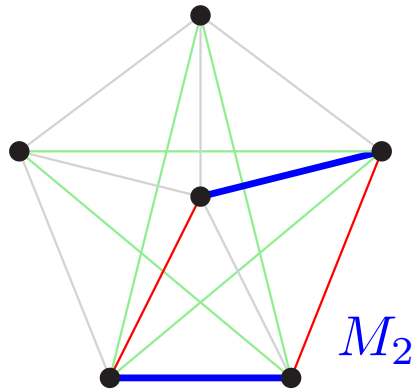
  $\Rightarrow\ w(HK) \leq w(EK) = w(T') = w(T) + w(M) \leq w(OPT) + w(M)$

- Bestimmung von $w(M)$:
  - sei HK' Hamiltonkreis auf U (U: Knoten mit ungeradem Grad in T)
    (erzeugt durch Überspringen aller Knoten $V \setminus U$ in OPT)
  - definiere alternierende perfekte Matchings M_1, M_2 auf HK'
    (existiert, da $|U|$ gerade, $HK' = M_1 \cup M_2$)

  $\Rightarrow\ w(M) \leq w(M_1), w(M) \leq w(M_2)$ (M min. Matching!)

  $\Rightarrow\ 2 \cdot w(M) \leq w(M_1) + w(M_2) = w(HK') \leq w(OPT)$
  (Überspringen gerader Knoten und Dreiecks-Ungleichung)

$\Rightarrow\ w(HK) \leq \frac{3}{2} w(OPT)$



- Existiert immer perfektes Matching M?
  (Zeige, dass $|U|$ immer gerade ist!)

Approximationsalgorithmen

Zusammenfassung

Wissenswert (nicht erschöpfend!)

- Approximationsfaktor ρ
- APX, PTAS, FPTAS, pseudopolynomiell
- es gibt nicht gut approximierbare Probleme (z.B. minimum TSP)

Typische Fragestellungen

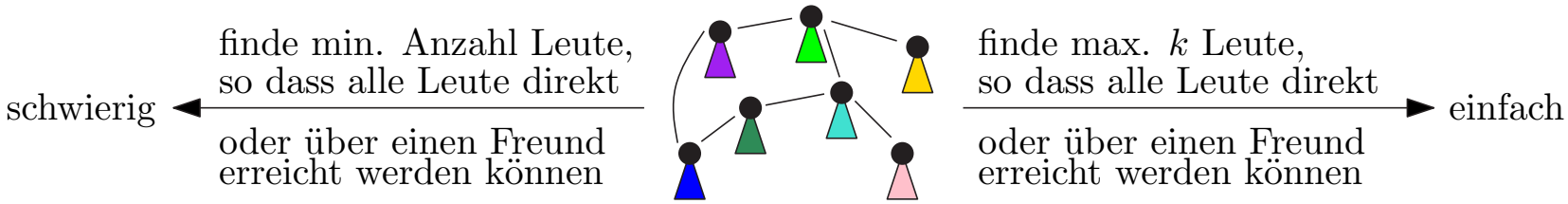
- Zu welcher Klasse gehört Algorithmus X mit $T(n, \varepsilon)$, $\rho(n, \varepsilon)$?
- Zeigen/Widerlegen Sie Approximationsfaktor ρ für Algorithmus X.
- (Bestimmen Sie einen Algorithmus mit Approximationsfaktor ρ)

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.

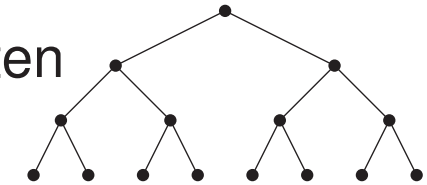
($f(\cdot)$ darf nicht von n abhängen und $p(\cdot)$ nicht von k; häufig Entscheidungsprobleme)

Parametrisierte Algorithmen

Techniken

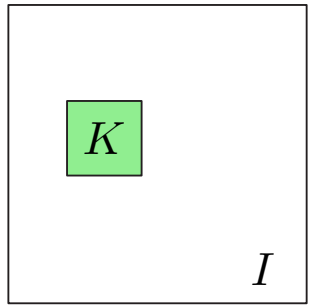
Tiefenbeschränkte Suche

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



Kernbildung

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



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

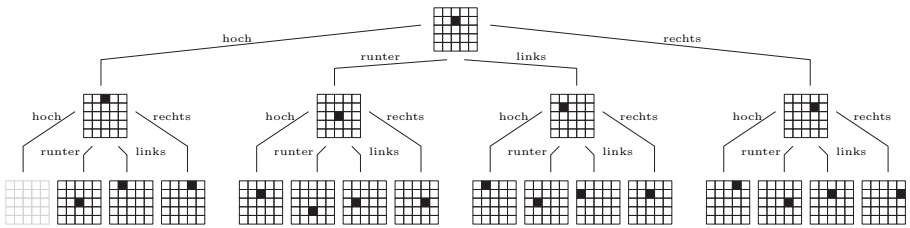
- gegeben $n \times n$ Schiebepuzzle, $k \in \mathbb{N}$
- entscheide, ob das Puzzle in $\leq k$ Zügen gelöst werden kann
  - das Puzzle ist gelöst, wenn die Teile sortiert sind
  - Loch wird pro Zug eine Position horizontal oder vertikal verschoben

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

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

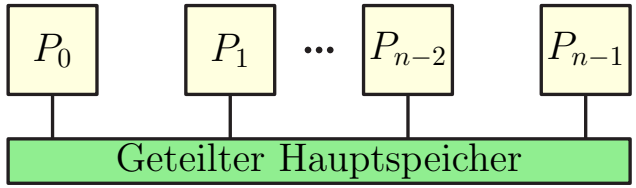


Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

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

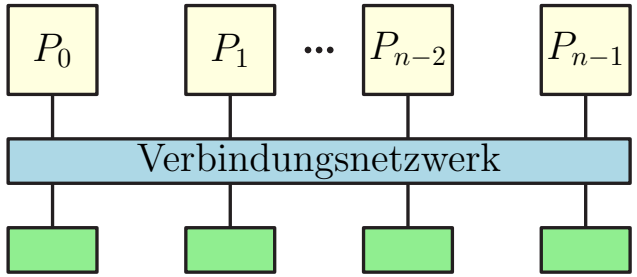
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

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

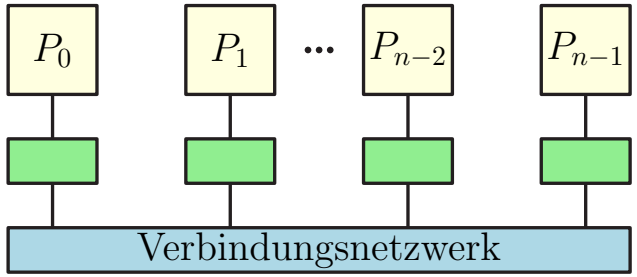
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

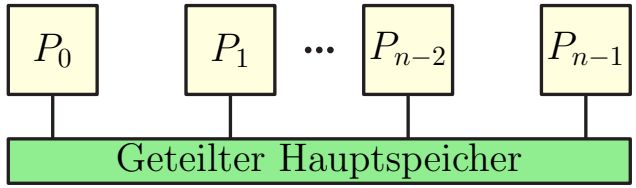
B(ulk)**S**(ynchronous)**P**(arallel)

- 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

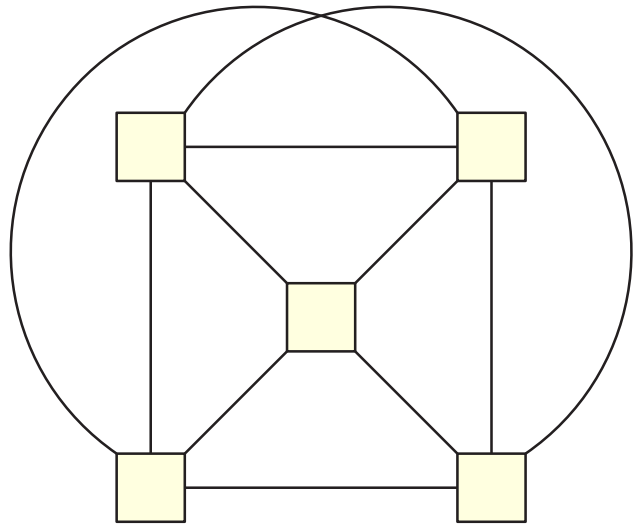
Struktur

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

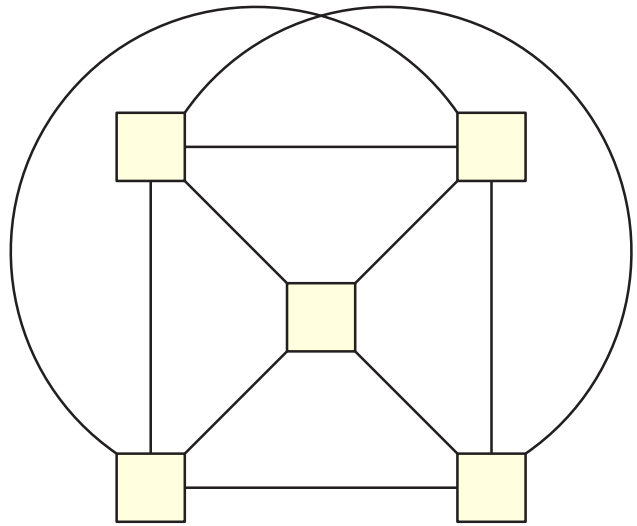
- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

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
  $***1** \leftrightarrow ***0**$

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

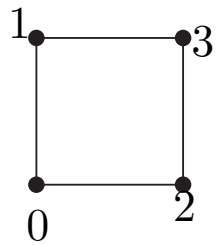
Struktur

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

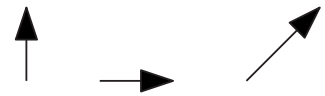
- Kostenmaß Kommunikation ($T_{comm} = T_{start} + l \cdot T_{byte}$)

Verbindungsnetzwerke

Struktur

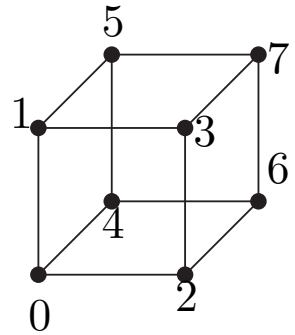
Vollverkabelt

- 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

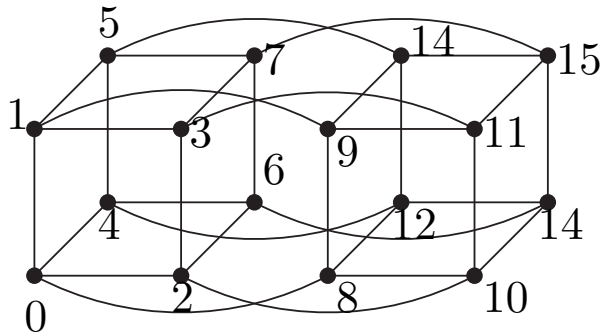
Struktur

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

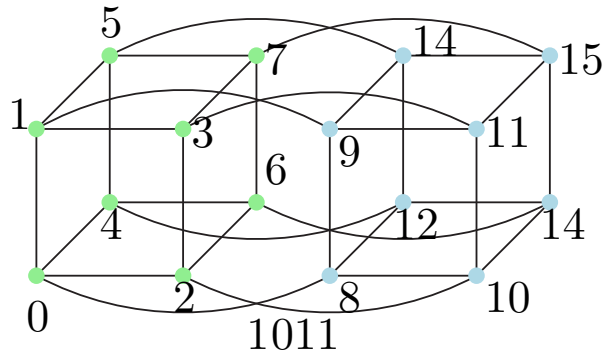
Struktur

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

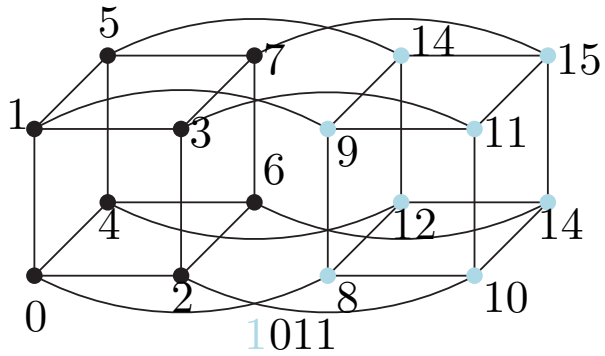
Struktur

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

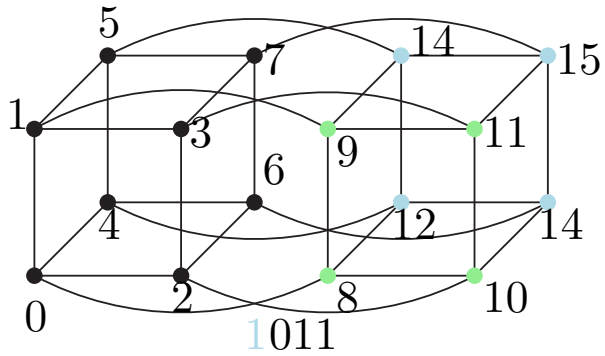
Struktur

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

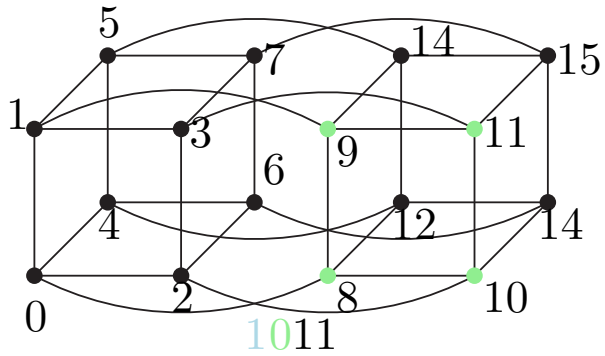
Struktur

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

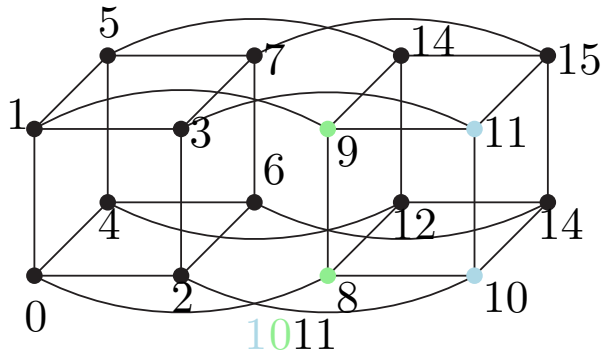
Struktur

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

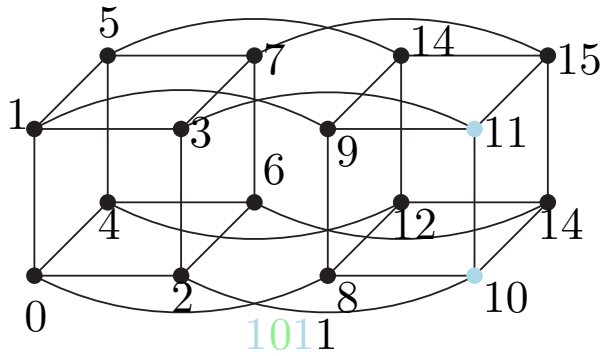
Struktur

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

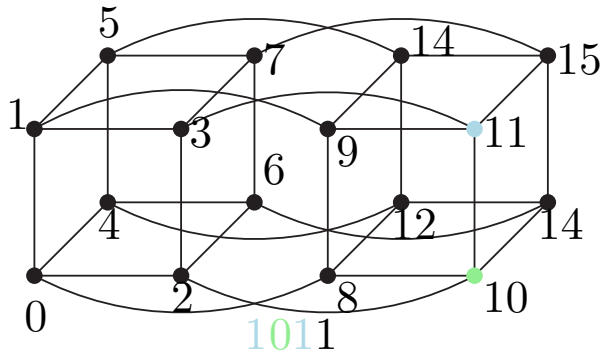
Struktur

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

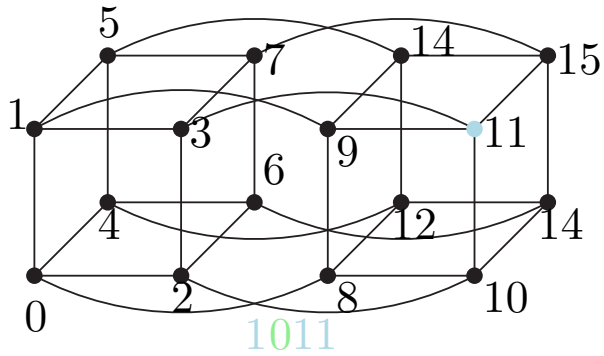
Struktur

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
  $***1** \leftrightarrow ***0**$

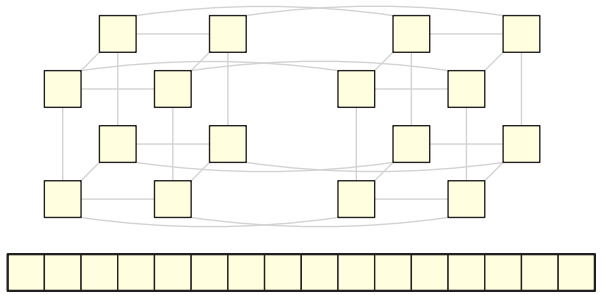


Kosten

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

Anwendungen

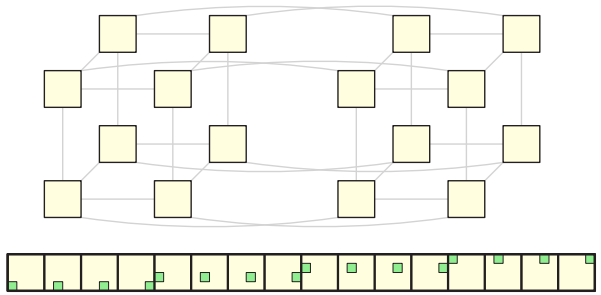
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

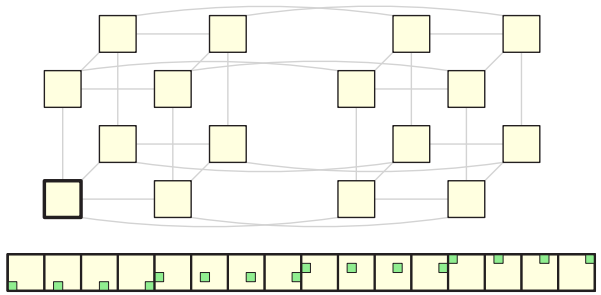
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

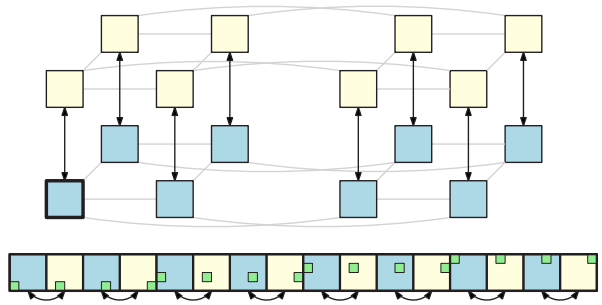
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

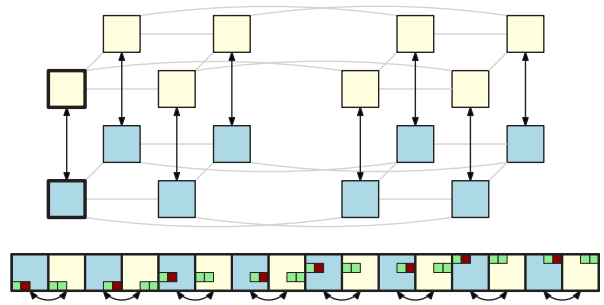
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

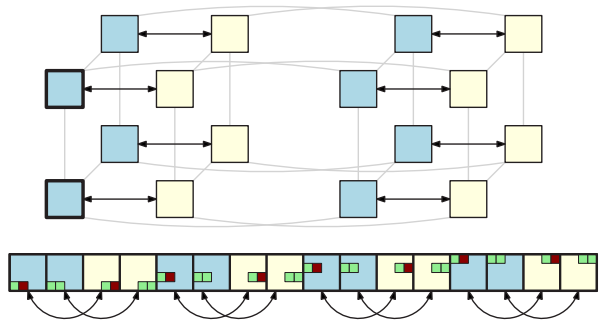
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

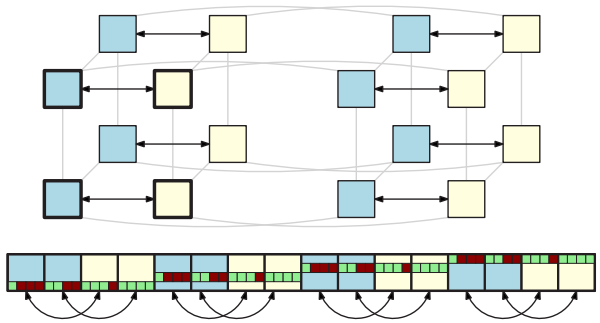
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID

- sonst: erhaltene Daten gehören zu CPUs größerer ID

- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

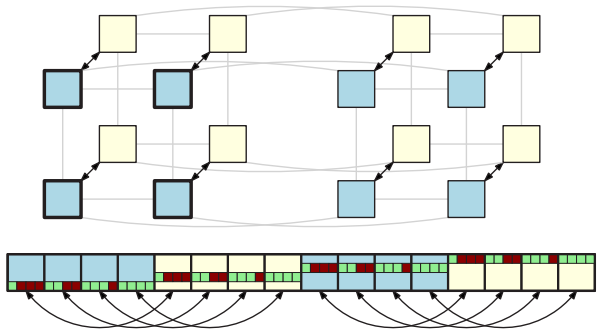
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID

- sonst: erhaltene Daten gehören zu CPUs größerer ID

- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

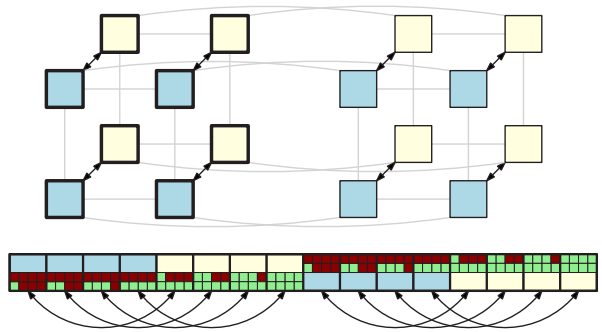
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

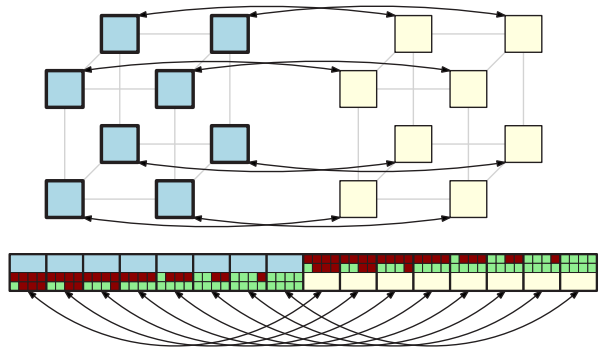
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

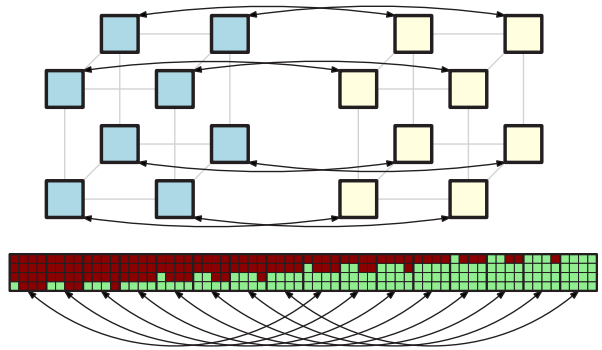
Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

Präfixsumme - Hypercube



- jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- ist in Schritt k das eigene k-te Bit 1 so gilt:
  Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID
- sonst: erhaltene Daten gehören zu CPUs größerer ID
- $T(n,p) = (T_{start} + n \cdot T_{byte}) \cdot \log p$

Anwendungen

PRAM Präfixsumme

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

Aufwärtsphase

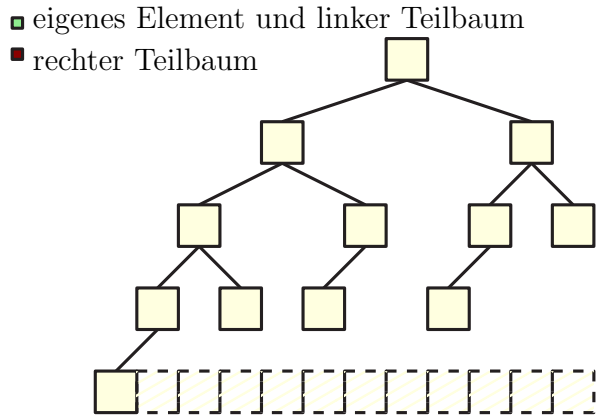
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

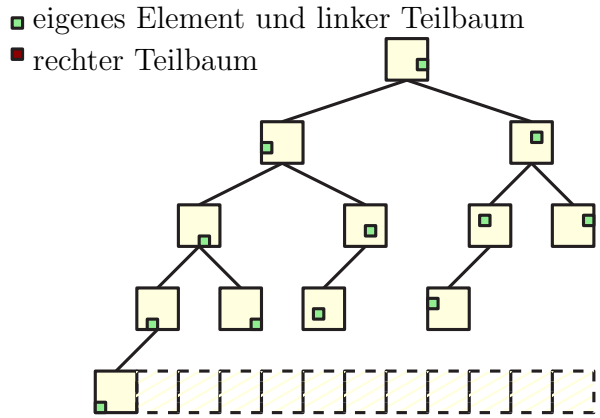
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

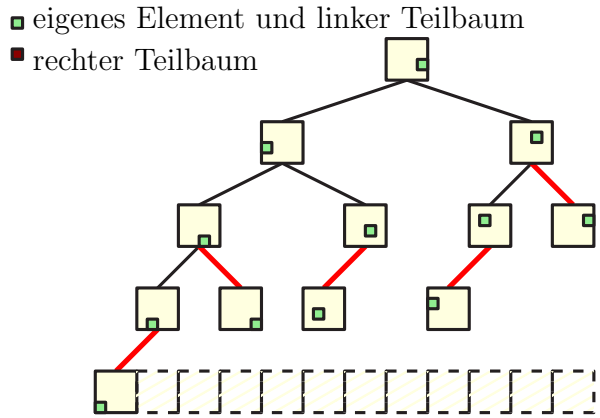
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

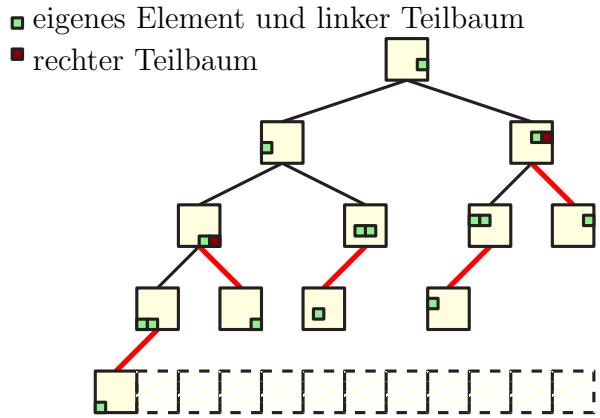
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

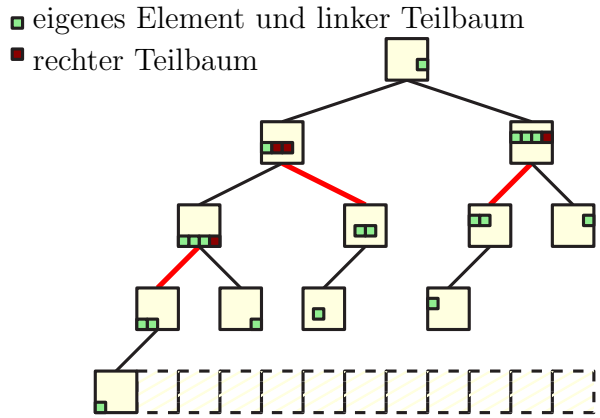
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

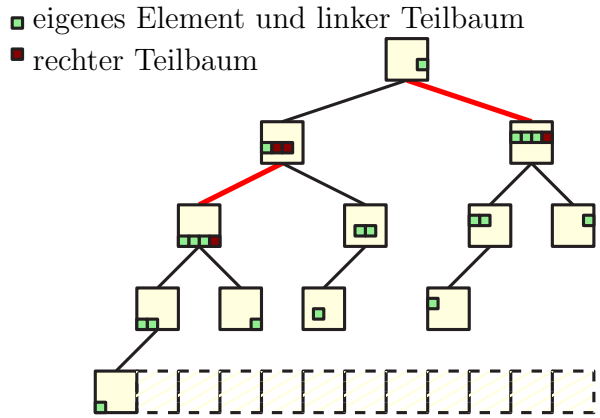
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

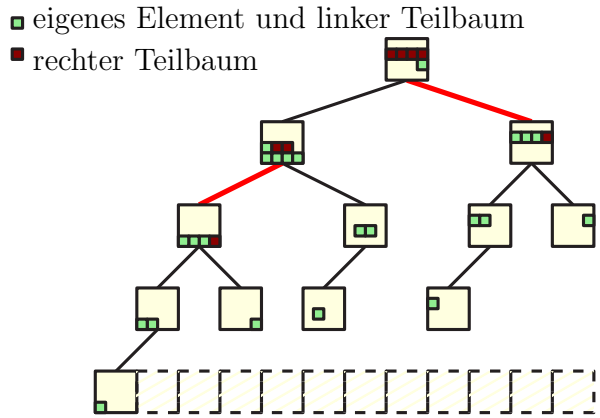
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

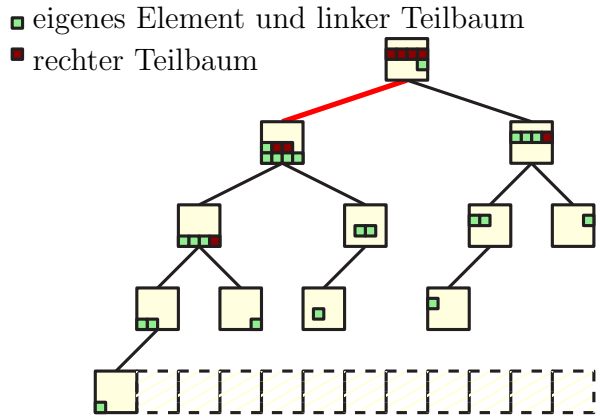
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

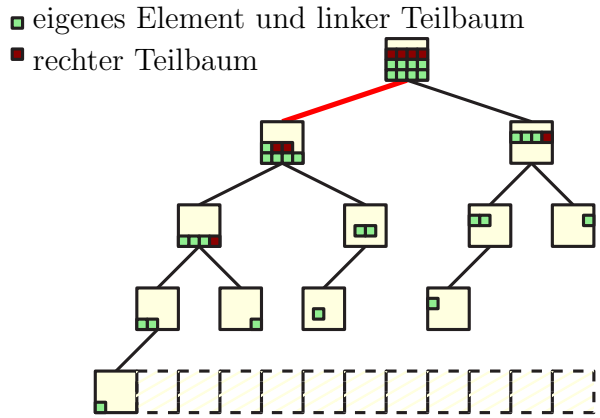
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

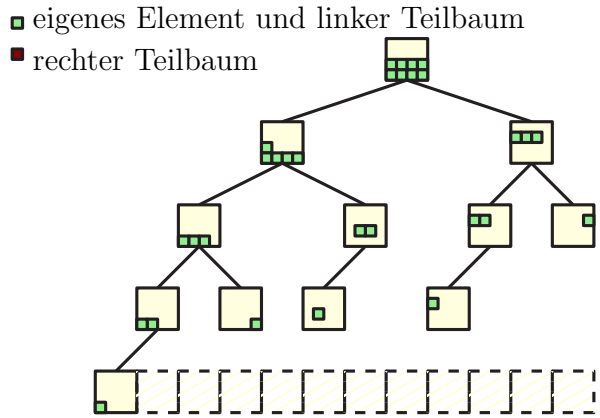
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

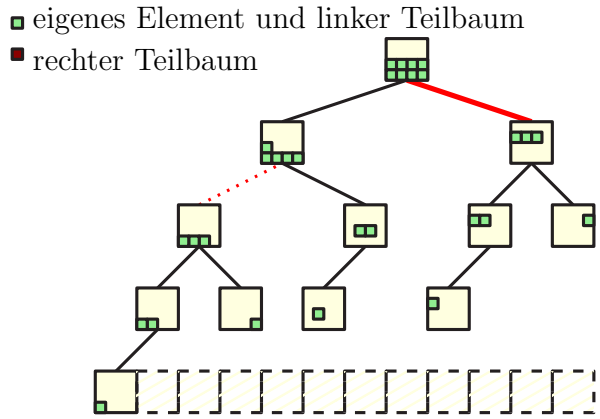
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

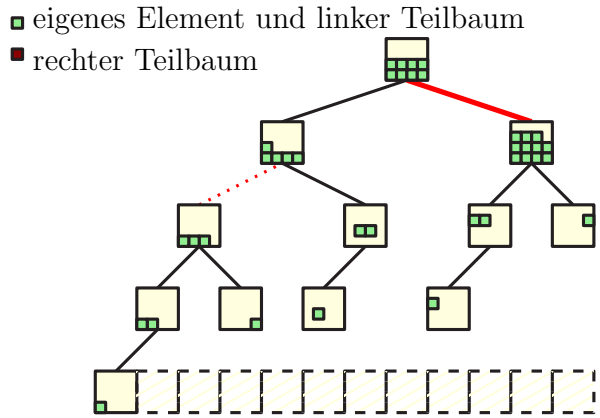
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

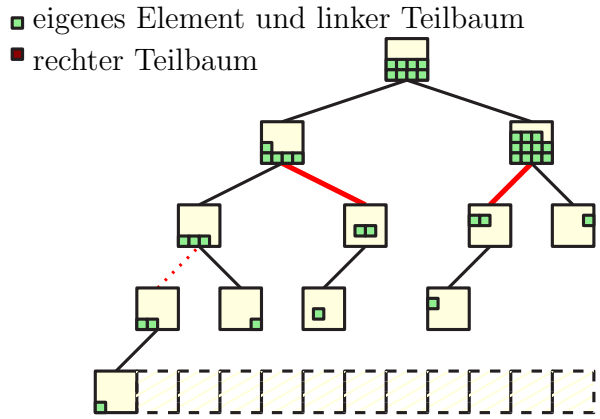
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

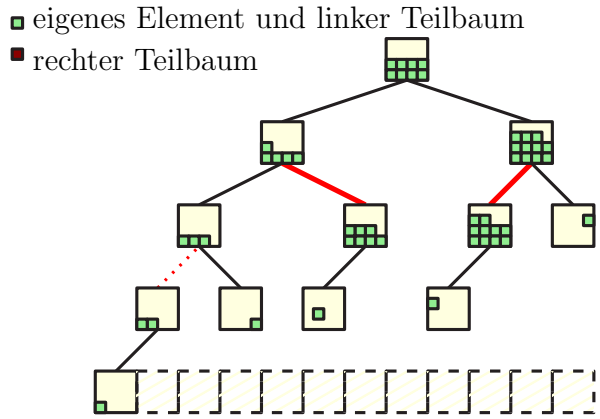
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

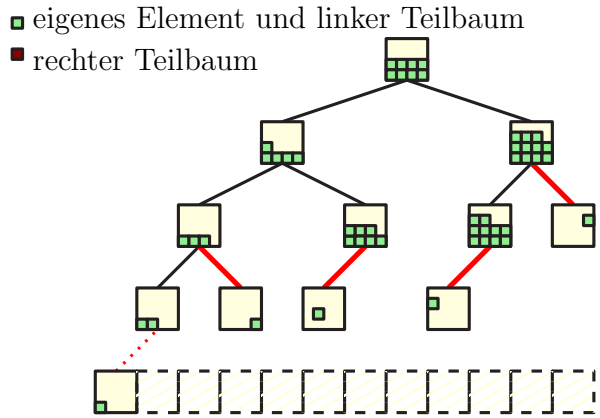
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

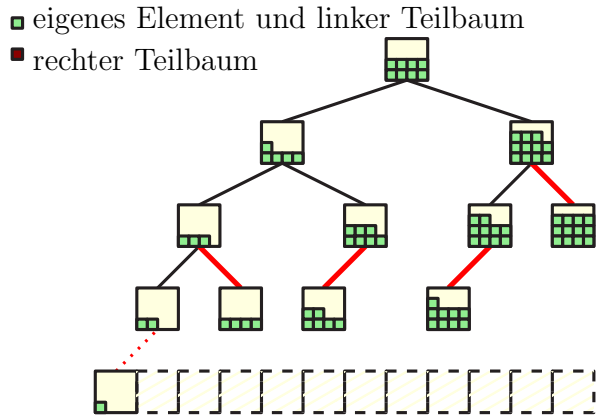
- Prozessoren aggregieren Daten aus Teilbäumen
- speichere Summe kleinerer (linker Teilbaum) und größerer (rechter Teilaum) Elemente (getrennt voneinander)
- leite Summe aller Elemente an Vorgängerknoten

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

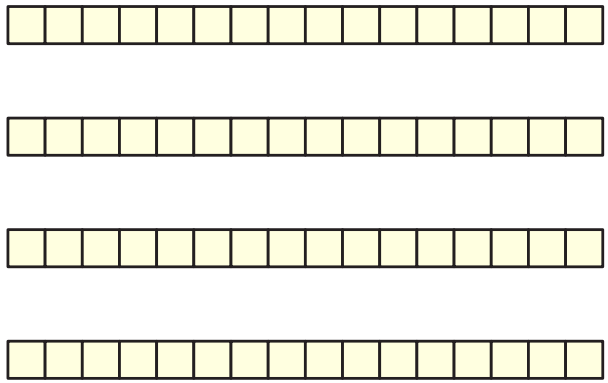
- gesammelte Daten werden verteilt
- bisher in Abwärtsphase empfangene Daten; eigene Daten und Daten des linken Teilbaums nur nach rechts

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

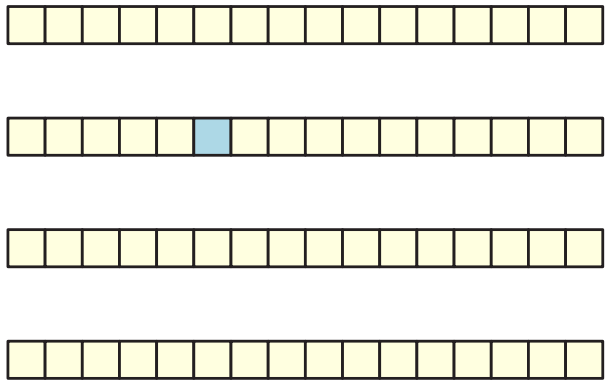


Anwendungen

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

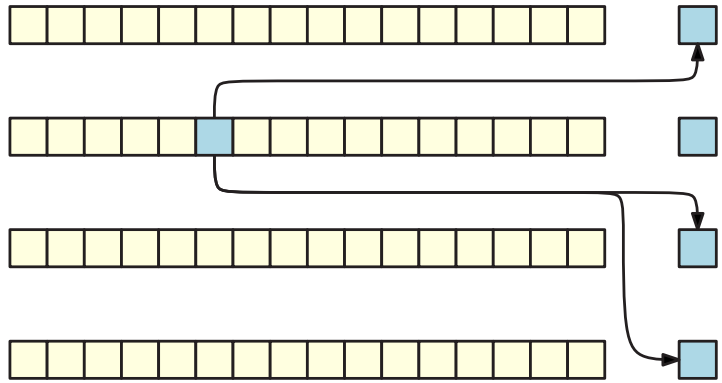


Anwendungen

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

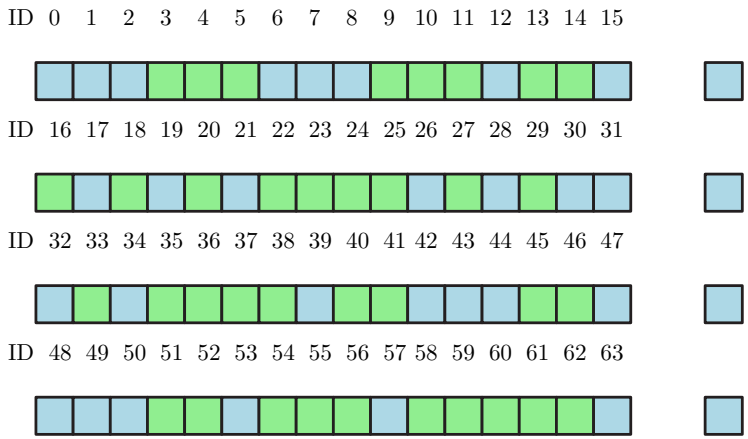


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   → 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

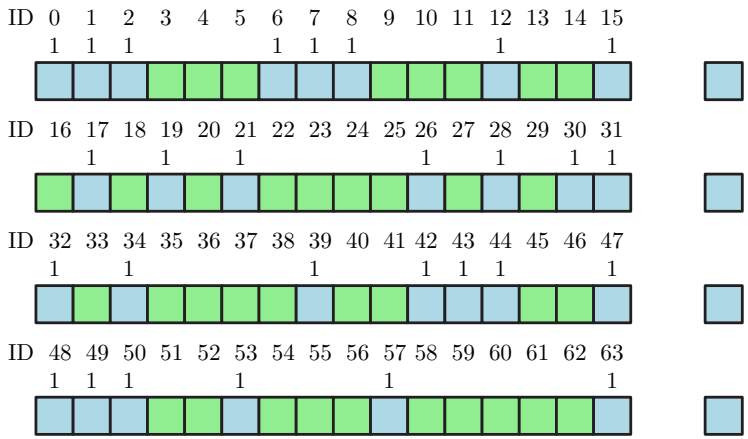


Anwendungen

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

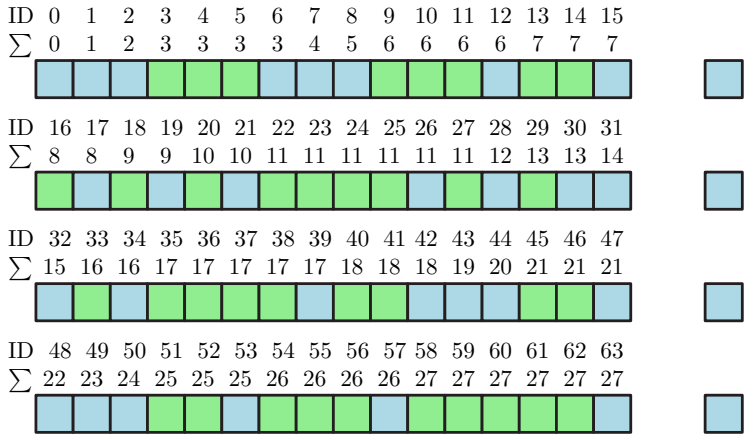


Anwendungen

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

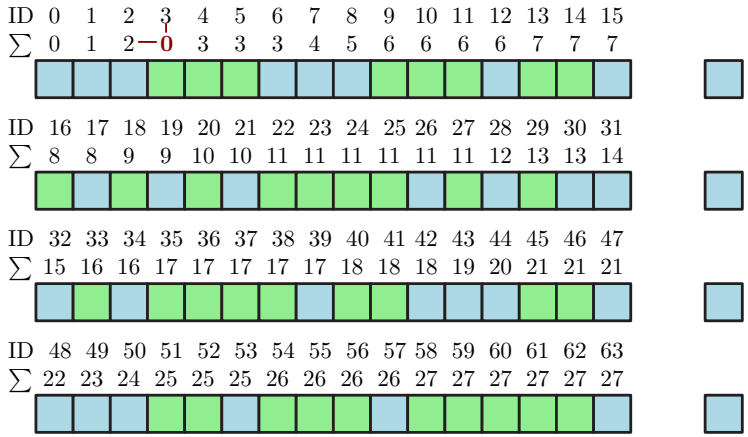


Anwendungen

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

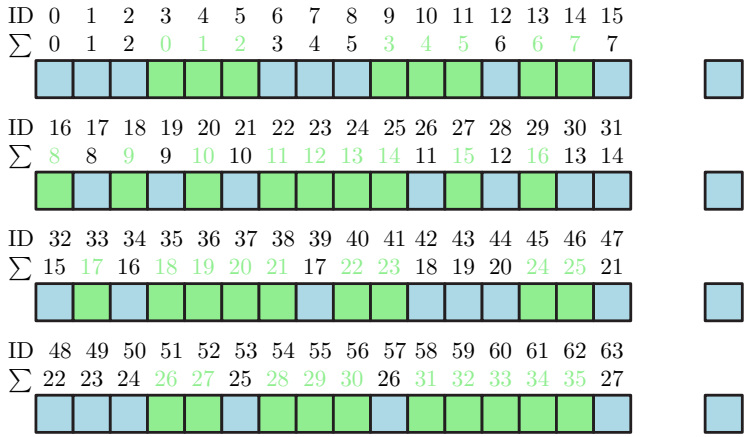


Anwendungen

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

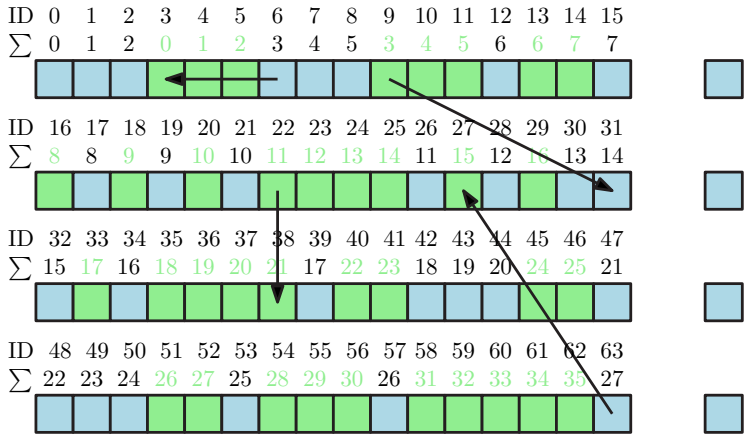


Anwendungen

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

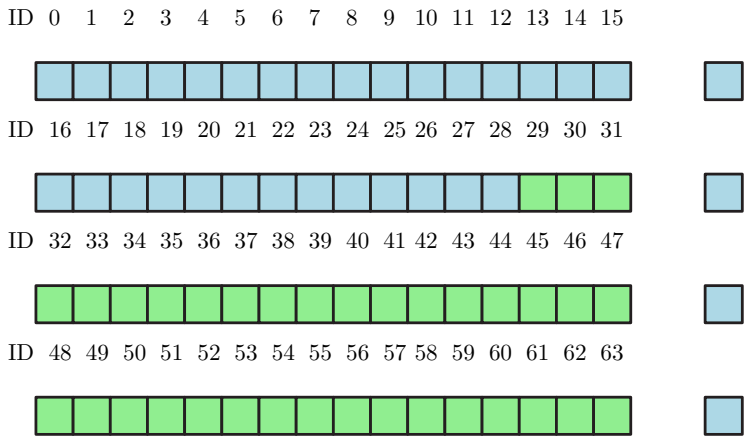


Anwendungen

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

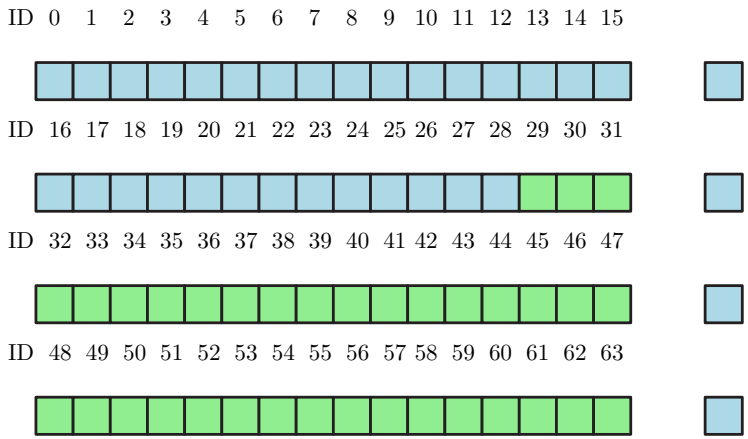


Anwendungen

Paralleler Quicksort

Rekursives Verfahren

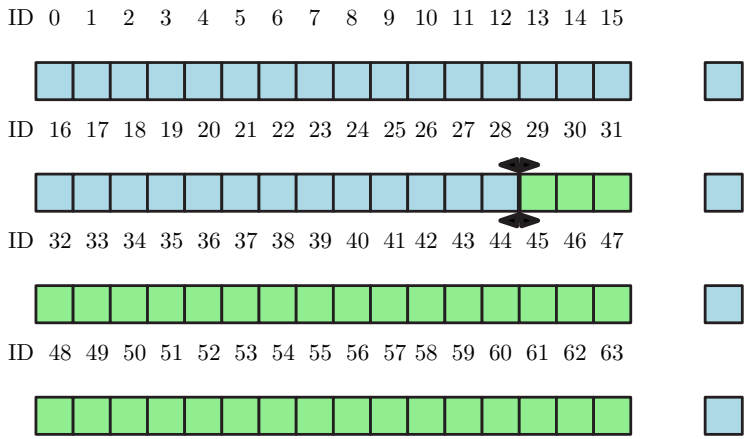
1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position 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

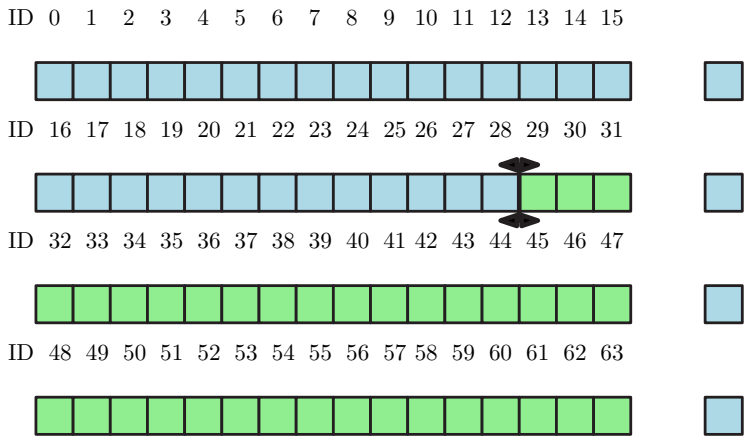


1. ein PE stellt Pivot (zufällig)
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position 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)
- Intel Thread Building Block Library (TBB)
  - `https://software.intel.com/en-us/tbb`
  - mehr objektorientierter als *OpenMP*
  - enthält konkurrente Datenstrukturen und viele parallele Primitive
    - Konkurrente *Queues*, *Arrays*, ...
    - Parallel *Sort*, *For*, *While*, ...
    - Komplexe *Dataflow*-Graphen
    - Geschachtelter und rekursiver Parallelismus
- Message Passing Interface (MPI)
  - `https://www.mcs.anl.gov/research/projects/mpi/`
  - Standard für verteilte Programmierung
  - implementiert die gängisten Kommunikationsprimitiven (`C`-Style Interface)

Ende!

Feierabend!