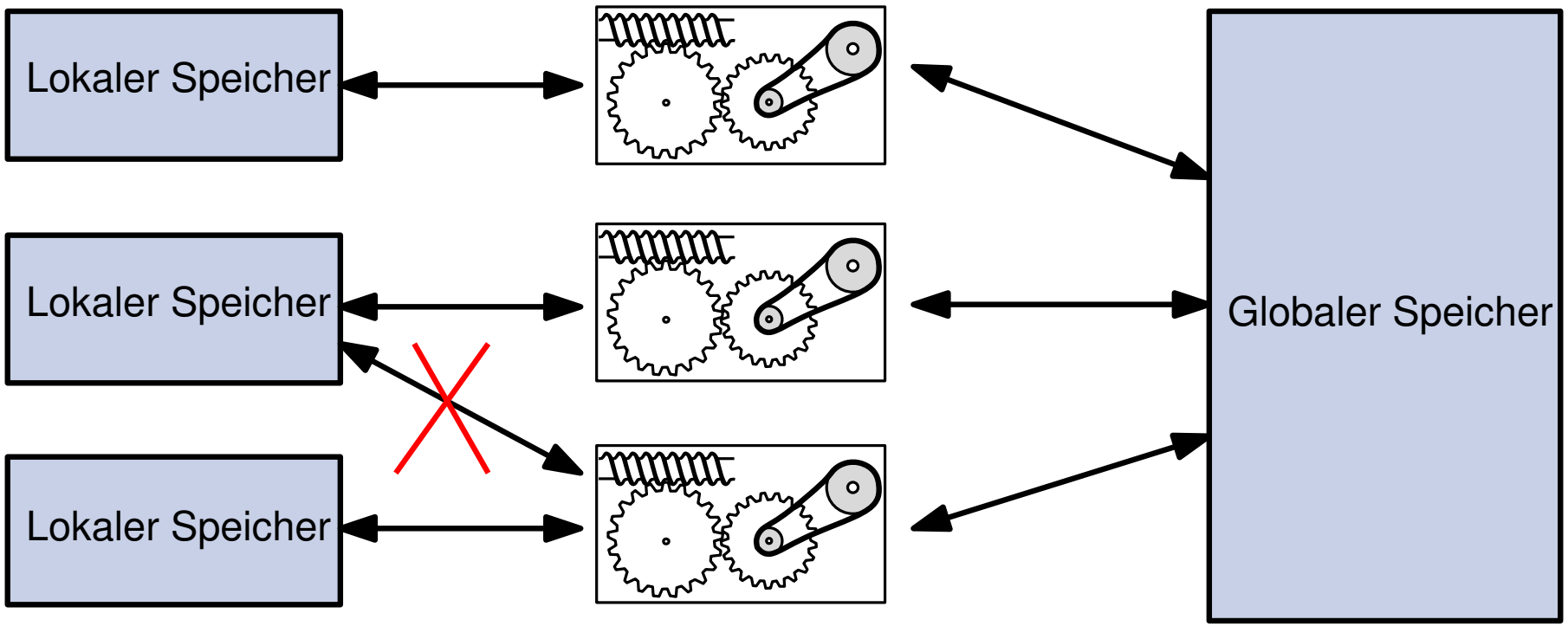
Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 06.02.2014

Parallele Algorithmen

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

PRAM Modell



- Unbegrenzte Prozessorenanzahl.
- Unbegrenzter globaler Speicher, auf den alle Prozessoren zugreifen können.
- Satz an Operationen, die jeder Prozessor einzeln ausführen kann.
- Jeder Prozessor hat einen eigenen lokalen unbegrenzten Speicher.

PRAM Modell

Problem: Was, wenn mehrere Prozessoren gleichzeitig die gleiche Speicherstelle im globalen Speicher lesen, bzw. beschreiben wollen?

gleichzeitiges Lesen erlaubt CR (*concurrent read*)	gleichzeitiges Lesen verboten ER (*exclusive read*)
gleichzeitiges Schreiben erlaubt CW (*concurrent write*)	gleichzeitiges Schreiben verboten EW (*exclusive write*)

Im folgenden Beschränkung auf CREW-PRAM-Modell.

Komplexität von parallelen Algorithmen

- Betrachte für Laufzeitanalyse immer den schlimmsten Fall.
- Berechnungsschritt = N Operationen, die gleichzeitig von N Prozessoren ausgeführt werden.

Definition 9.1 Die Laufzeit $T_{\mathcal{A}}(n)$ eines parallelen Algorithmus $\mathcal{A}$ ist

$$T_{\mathcal{A}}(n) := \max_{\substack{I \text{ Problembeispiel} \\ \text{der Größe } n}} \{\text{Anzahl der Berechnungsschritte von } \mathcal{A} \text{ bei Eingabe } I\}$$

Bemerkung:

1. Berechnung beginnt, wenn der erste Prozessor aktiv wird.
2. Berechnung endet, wenn der letzte Prozessor inaktiv geworden ist.
3. Prozessoren arbeiten synchron.

Definition 9.3 Die Prozessorenanzahl $P_{\mathcal{A}}(n)$ eines parallelen Algorithmus $\mathcal{A}$ ist

$$P_{\mathcal{A}}(n) := \max_{\substack{I \text{ Problembeispiel} \\ \text{der Größe } n}} \left\{ \begin{array}{l} \text{Anzahl an Prozessoren, die während des Ablaufs von } \mathcal{A} \\ \text{bei Eingabe } I \text{ gleichzeitig aktiv sind} \end{array} \right\}$$

Komplexität von parallelen Algorithmen

Vergleich von sequentiellen und parallelen Algorithmen für ein Problem:

$$\text{speed-up}(\mathcal{A}) := \frac{\text{worst-case Laufzeit des schnellsten bekannten sequentiellen Algorithmus}}{\text{worst-case Laufzeit des parallelen Algorithmus } \mathcal{A}}$$

Kombiniere Prozessorenanzahl und Laufzeit zu *Kosten* $C_{\mathcal{A}}$:

$$C_{\mathcal{A}}(n) := T_{\mathcal{A}}(n) \cdot P_{\mathcal{A}}(n)$$

kostenoptimal = asymptotisches Wachstum von $C_{\mathcal{A}}(n)$ und die schärfste asymptotische untere Schranke für die Laufzeit eines sequentiellen Algorithmus sind gleich.

Wenn untere Schranke nicht bekannt, so betrachte Effizienz:

$$E_{\mathcal{A}}(n) := \frac{\text{worst-case Laufzeit des schnellsten bekannten sequentiellen Algorithmus}}{\text{Kosten des parallelen Algorithmus } \mathcal{A}}$$

Komplexitätsklassen

Definition 9.4: Nick's Class nach Nicholas Pippinger
Die Klasse $\mathcal{NC}$ ist die Klasse der Probleme, die durch einen parallelen Algorithmus $\mathcal{A}$ mit polylogarithmischer Laufzeit und polynomieller Prozessorenzahl gelöst werden können, d.h. $T_{\mathcal{A}}(n) \in \mathcal{O}((\log n)^{k_1})$ mit k_1 Konstante, und $P_{\mathcal{A}}(n) \in \mathcal{O}(n^{k_2})$ mit k_2 Konstante.

Definition 9.5: Steve's Class nach Stephan Cook
Die Klasse $\mathcal{SC}$ ist die Klasse der Probleme, die durch einen sequentiellen Algorithmus mit polylogarithmischem Speicherplatzbedarf und polynomieller Laufzeit gelöst werden können.

Berechnung von Summen

Karlsruher Institut für Technologie

$\texttt{Summe}(a_1, \ldots, a_n)$
Eingabe: n Werte $a_1, \ldots, a_n$, o.B.d.A. sei $n = 2^m$.
Ausgabe: $\sum_{i=1}^{n} a_i$

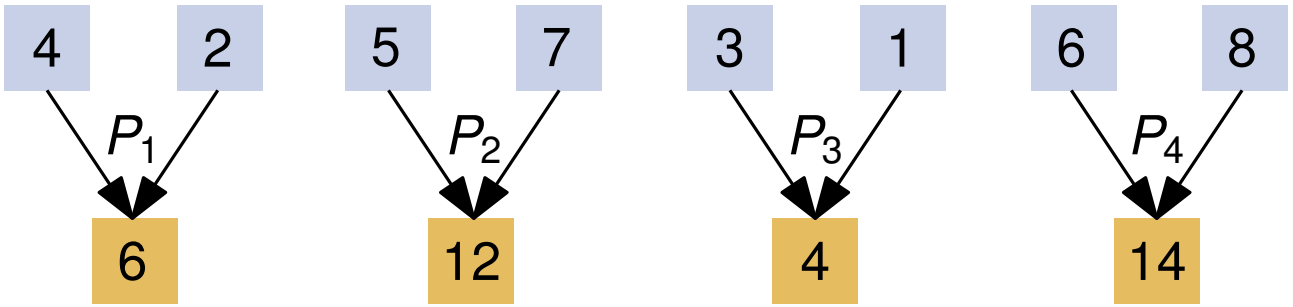
für $i = 1$ **bis** m **tue**
  Für alle j : $1 \leq j \leq \frac{n}{2^i}$ **führe parallel aus**
    Prozessor P_j berechnet $a_j := a_{2j-1} + a_{2j}$.
Gib a_1 aus.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Berechnung von Summen

Summe($a_1, \ldots, a_n$)
Eingabe: n Werte $a_1, \ldots, a_n$, o.B.d.A. sei $n = 2^m$.
Ausgabe: $\sum_{i=1}^{n} a_i$

für $i = 1$ **bis** m **tue**
  Für alle j : $1 \leq j \leq \frac{n}{2^i}$ **führe parallel aus**
    Prozessor P_j berechnet $a_j := a_{2j-1} + a_{2j}$.
Gib a_1 aus.

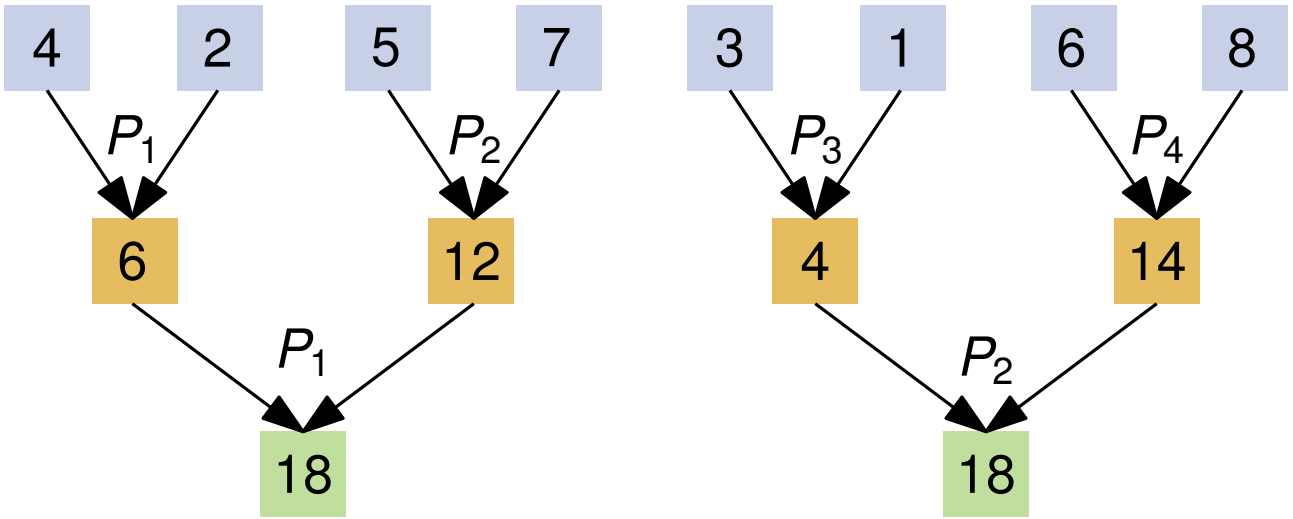


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Berechnung von Summen

Summe($a_1, \ldots, a_n$)
Eingabe: n Werte $a_1, \ldots, a_n$, o.B.d.A. sei $n = 2^m$.
Ausgabe: $\sum_{i=1}^{n} a_i$

für $i = 1$ **bis** m **tue**
  Für alle j : $1 \leq j \leq \frac{n}{2^i}$ **führe parallel aus**
    Prozessor P_j berechnet $a_j := a_{2j-1} + a_{2j}$.
Gib a_1 aus.

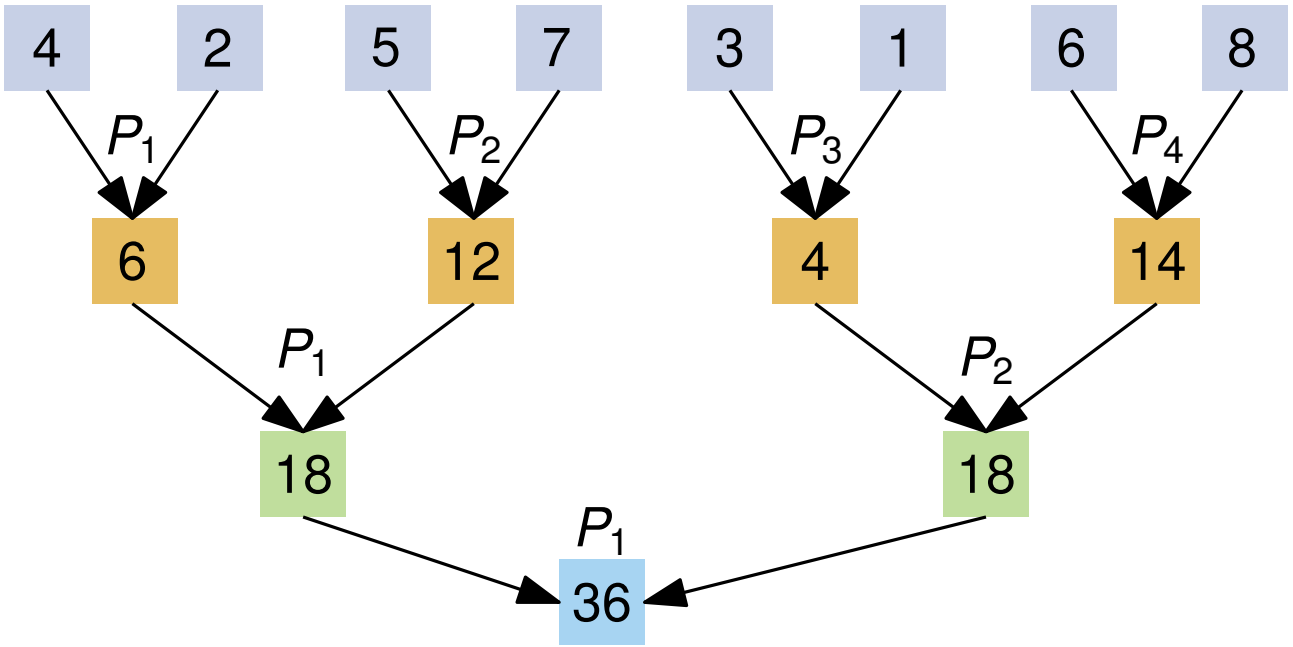


Berechnung von Summen

Summe($a_1, \ldots, a_n$)
Eingabe: n Werte $a_1, \ldots, a_n$, o.B.d.A. sei $n = 2^m$.
Ausgabe: $\sum_{i=1}^{n} a_i$

für $i = 1$ **bis** m **tue**
  Für alle j : $1 \leq j \leq \frac{n}{2^i}$ **führe parallel aus**
    Prozessor P_j berechnet $a_j := a_{2j-1} + a_{2j}$.
Gib a_1 aus.

- $P_{\mathcal{A}}(n) = \frac{n}{2}$, $T_{\mathcal{A}}(n) = \mathcal{O}(\log n)$
- Somit gilt: $C_{\mathcal{A}}(n) \in \mathcal{O}(n \log n)$
- **$\mathcal{A}$ ist nicht kostenoptimal.**



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Berechnung von Summen

Summe($a_1, \ldots, a_n$)
Eingabe: n Werte $a_1, \ldots, a_n$, o.B.d.A. sei $n = 2^m$.
Ausgabe: $\sum_{i=1}^{n} a_i$

für $i = 1$ **bis** m **tue**
  Für alle j : $1 \leq j \leq \frac{n}{2^i}$ **führe parallel aus**
    Prozessor P_j berechnet $a_j := a_{2j-1} + a_{2j}$.
Gib a_1 aus.

Verbesserung:

- Verwende $\lceil \frac{n}{\log n} \rceil$ anstatt n Prozessoren.
- Die $\frac{n}{2}$ Operationen aus dem ersten Schritt werden nun von $\lceil \frac{n}{\log n} \rceil$ Prozessoren in höchstens $\frac{\log n}{2}$ parallelen Berechnungsschritten ausgeführt.
- Im i-ten Schritt werden $\frac{n}{2^i}$ Operationen in höchstens $\frac{\log n}{2^i}$ Schritten ausgeführt.
- Es ergibt sich:

$$T_{\mathcal{A}}(n) \leq c \cdot \sum_{i=1}^{\log n} \frac{\log n}{2^i} = c \cdot \log n \underbrace{\sum_{i=1}^{\log n} \frac{1}{2^i}}_{\leq 1} \in \mathcal{O}(\log n).$$

- Wegen $P_{\mathcal{A}}(n) = \lceil \frac{n}{logn} \rceil$ gilt $C_{\mathcal{A}} \in O(n)$, **was kostenoptimal ist.**

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Berechnung von Summen

Karlsruher Institut für Technologie

$\texttt{Summe}(a_1, \ldots, a_n)$
Eingabe: n Werte $a_1, \ldots, a_n$, o.B.d.A. sei $n = 2^m$.
Ausgabe: $\sum_{i=1}^{n} a_i$

für $i = 1$ **bis** m **tue**
  Für alle j : $1 \leq j \leq \frac{n}{2^i}$ **führe parallel aus**
    Prozessor P_j berechnet $a_j := a_{2j-1} + a_{2j}$.
Gib a_1 aus.

Das selbe Verfahren funktioniert auch auf anderen binären Operationen, die n-fach angewendet werden. Beispiele: Minimum, Maximum, logisches Oder.

Bemerkung: CREW-PRAM (oder ähnliche Modelle) und ein Modell, bei dem zu Beginn nur ein Prozessor aktiv ist und alle anderen erst „aufgeweckt“ werden, unterscheiden sich in der Laufzeit im wesentlichen nur um einen Faktor $\log N$ (bei N Prozessoren, die gleichzeitig lesen wollen, bzw. aktiviert werden sollen).

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

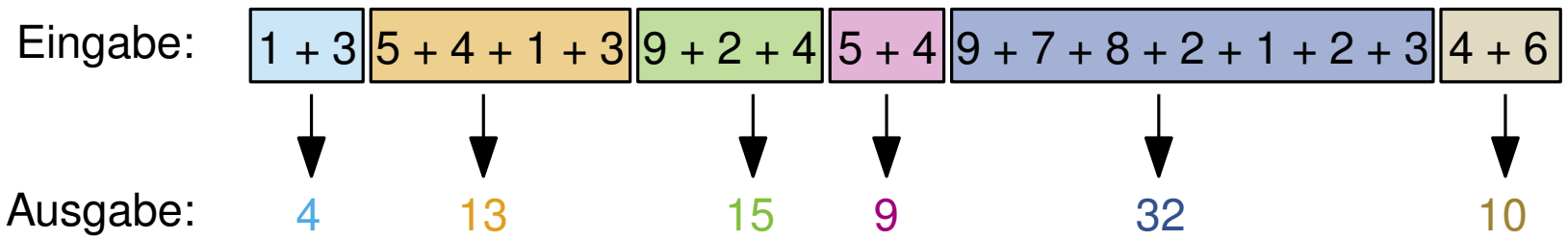
gegeben: n Werte, die in p Gruppen aufgeteilt sind.

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Ziel: Verwende K Prozessoren, sodass folgende Laufzeit verwendet wird.

$$t(n) \in \begin{cases} O(\lceil \frac{n}{K} \rceil + \log K), & \text{falls } n > K \\ O(\log n), & \text{sonst.} \end{cases}$$

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind.

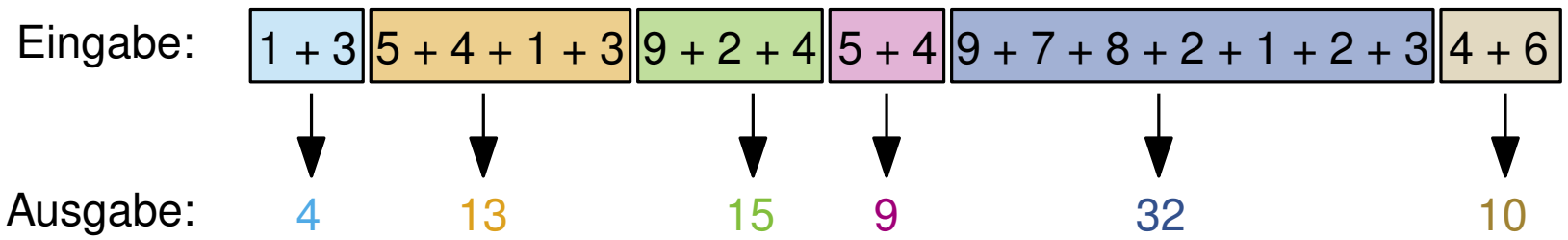
gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Ziel: Verwende K Prozessoren, sodass folgende Laufzeit verwendet wird.

$$t(n) \in \begin{cases} O(\lceil \frac{n}{K} \rceil + \log K), & \text{falls } n > K \\ O(\log n), & \text{sonst.} \end{cases}$$

→ Jeder Wert erhält einen Prozessor.
Löse wie auf vorherigen Folien.

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

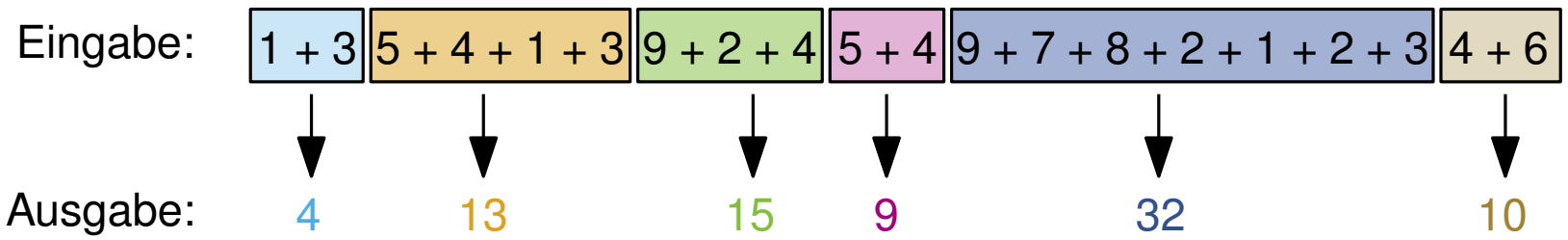
Ziel: Verwende K Prozessoren, sodass folgende Laufzeit verwendet wird.

$$t(n) \in \begin{cases} O(\lceil \frac{n}{K} \rceil + \log K), & \text{falls } n > K \\ O(\log n), & \text{sonst.} \end{cases}$$

→ Wird im folgenden gezeigt.

→ **Annahme:** $n > K$

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

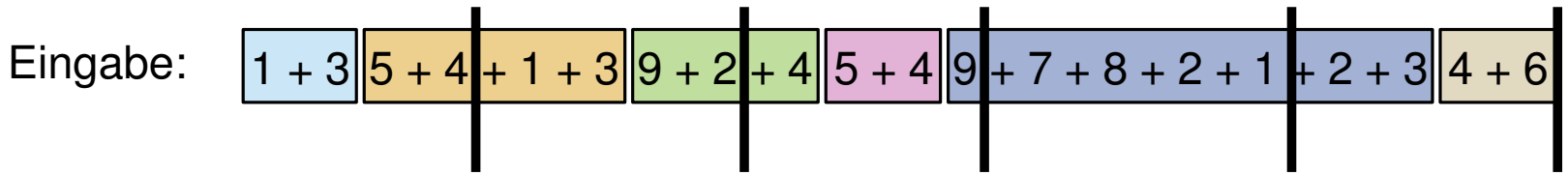
Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Teile die Werte in K Abschnitte auf. Jeder Abschnitt erhält einen Prozessor.

Beispiel:



Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

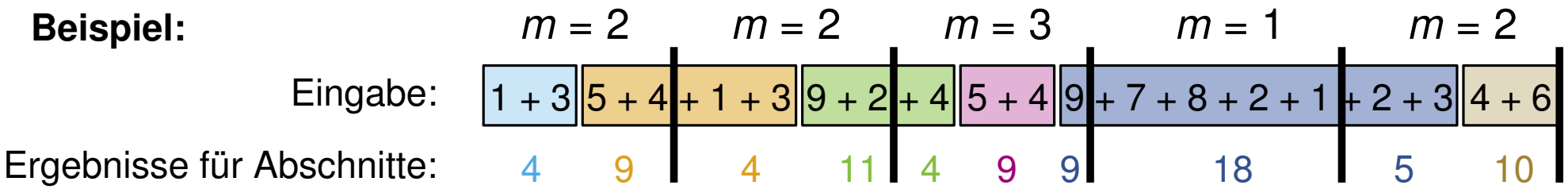
gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Teile die Werte in K Abschnitte auf. Jeder Abschnitt erhält einen Prozessor.

Betrachte einen Abschnitt:

1. Die Werte gehören zu m verschiedenen Gruppen.
2. Die Binäroperationen einer Gruppe können mit einem Prozessor in $\lceil \frac{n}{K} \rceil - 1$ Schritten berechnet werden.

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

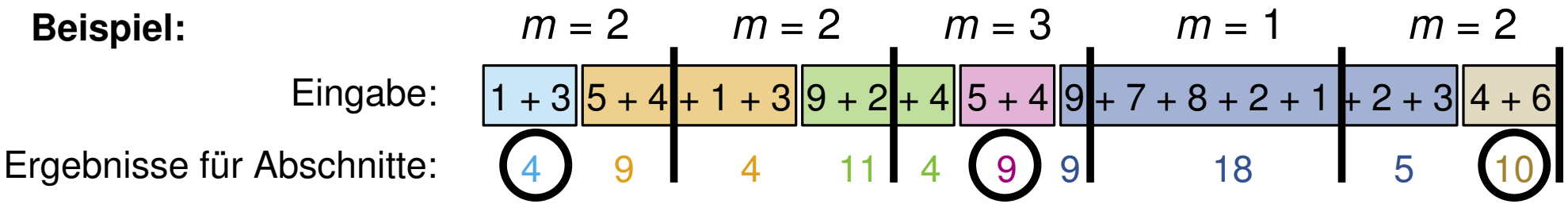
Teile die Werte in K Abschnitte auf. Jeder Abschnitt erhält einen Prozessor.

Betrachte einen Abschnitt:

1. Die Werte gehören zu m verschiedenen Gruppen.
2. Die Binäroperationen einer Gruppe können mit einem Prozessor in $\lceil \frac{n}{K} \rceil - 1$ Schritten berechnet werden.

→ Gruppen die vollständig in einem Abschnitt liegen, sind damit berechnet.

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Teile die Werte in K Abschnitte auf. Jeder Abschnitt erhält einen Prozessor.

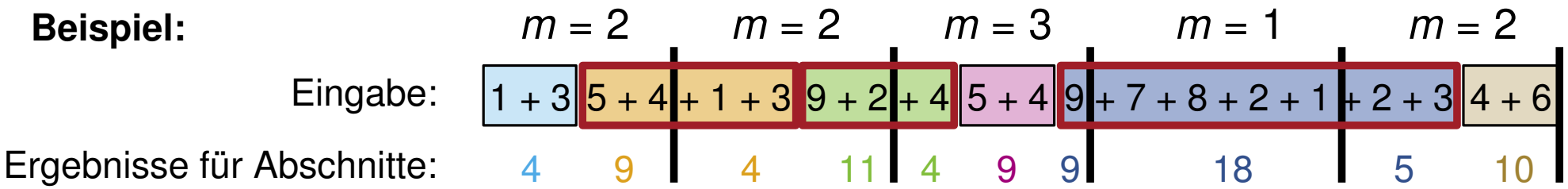
Betrachte einen Abschnitt:

1. Die Werte gehören zu m verschiedenen Gruppen.
2. Die Binäroperationen einer Gruppe können mit einem Prozessor in $\lceil \frac{n}{K} \rceil - 1$ Schritten berechnet werden.

→ Gruppen die vollständig in einem Abschnitt liegen, sind damit berechnet.

→ Für jeden Abschnitt gibt es max. zwei *unvollständige* Ergebnisse.

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

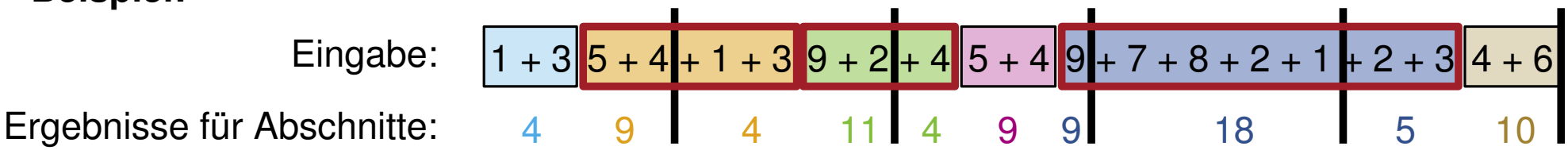
gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Es gilt:

$$\sum_{i=1}^{p} n_i \leq 2K - 2$$

mit n_i = Anzahl der noch zusammenfassenden Teilergebnisse.

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

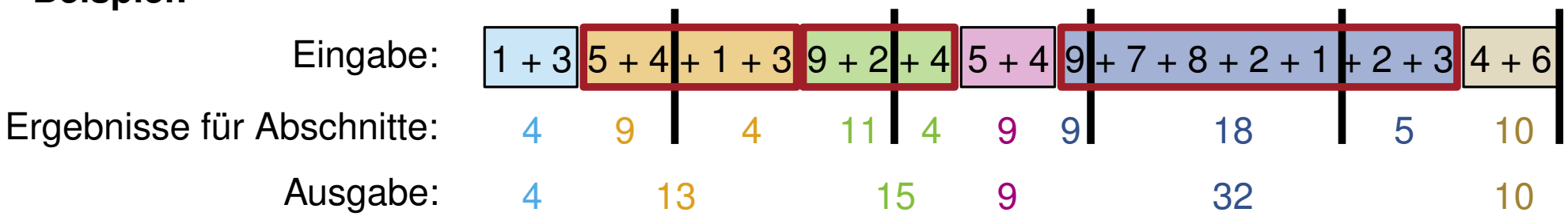
Es gilt: $\sum_{i=1}^{p} n_i \leq 2K - 2$ mit n_i = Anzahl der noch zusammenfassenden Teilergebnisse.

Ordne jeder Gruppe $\lfloor \frac{n_i}{2} \rfloor$ Prozessoren zu.

→ Berechne endgültiges Ergebnisse für G_i in $\log n_i$ Schritten.

→ wegen $n_i \leq K$ sind max. $\log K$ Schritte insgesamt nötig.

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Binäroperationen einer partitionierten Menge

Problem: *Binäroperationen auf einer partitionierten Menge (BPM)*

gegeben: n Werte, die in p Gruppen aufgeteilt sind. K Prozessoren mit $K < n$

gesucht: p Werte, die sich als Binäroperationen (Summe, Minimum, etc.) der Werte der p Gruppen ergeben.

Es gilt: $\sum_{i=1}^{p} n_i \leq 2K - 2$ mit n_i = Anzahl der noch zusammenfassenden Teilergebnisse.

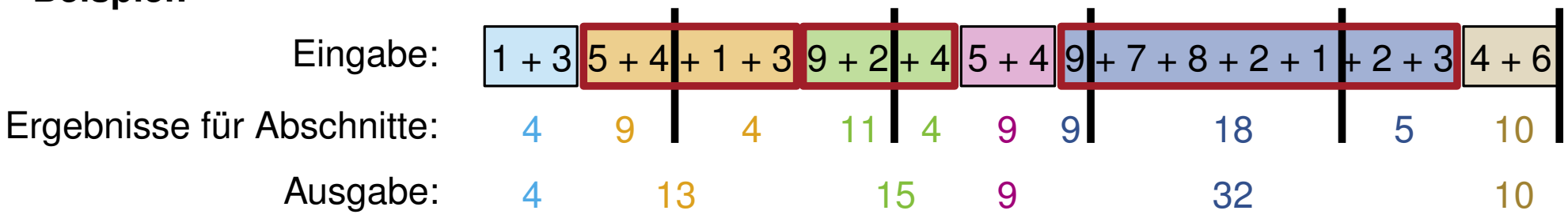
Ordne jeder Gruppe $\lfloor \frac{n_i}{2} \rfloor$ Prozessoren zu.

→ Berechne endgültiges Ergebnisse für G_i in $\log n_i$ Schritten.

→ wegen $n_i \leq K$ sind max. $\log K$ Schritte insgesamt nötig.

Anzahl Prozessoren reicht aus, da $\sum_{i=1}^{p} \lfloor \frac{n_i}{2} \rfloor \leq \frac{2K-2}{2} < K$

Beispiel:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

Problemstellung
gegeben: n Werte $a_n, \ldots, a_{2n-1}$.
gesucht: Präfixsumme $A_k = \sum_{i=n}^{k} a_i$ für jedes k mit $n \leq k \leq 2n-1$.

Verfahren besteht aus zwei Phasen:

1. Phase: Berechne $S :=$ SUMME$(a_n, \ldots, a_{2n-1})$.

2. Phase: Berechne aus S und den Zwischenergebnissen von SUMME$(a_n, \ldots, a_{2n-1})$ mithilfe von gegeigneten Differenzen die Präfixsummen.

Aus technischen Gründen: Indizierung mit $a_n, a_{n+1}, \ldots, a_{2n-1}$ und o.B.d.A. $n = 2^m$ für $m \in \mathbb{N}$.

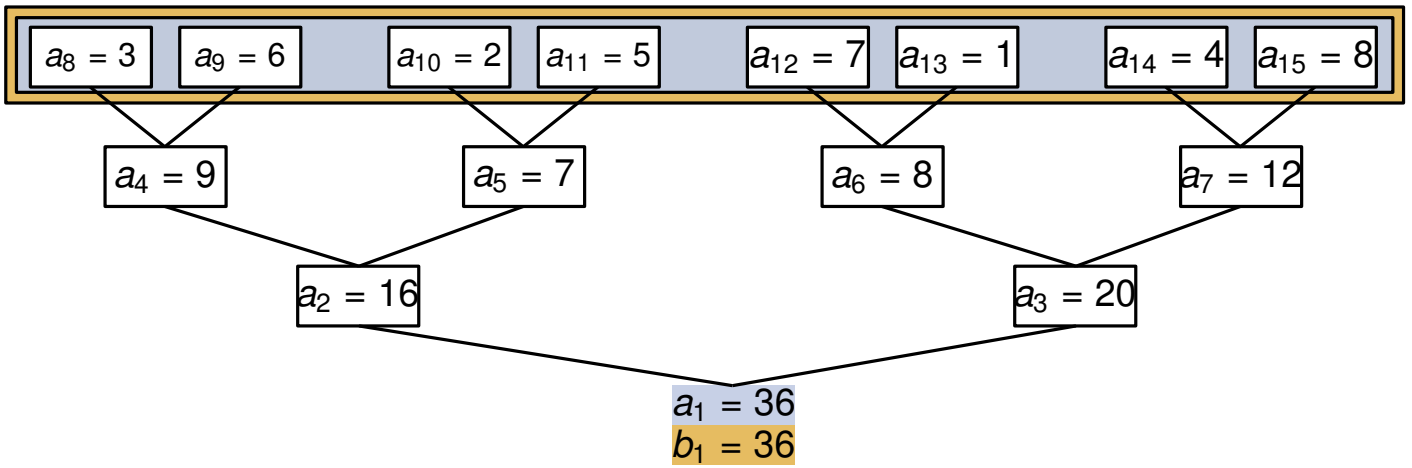
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

1. Phase: Berechne $S := \text{SUMME}(a_n, \ldots, a_{2n-1})$.

für $j = m - 1$ **bis** 0 **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    $a_i := a_{2i} + a_{2i+1}$
  $b_1 := a_1$

Erinnerung:
$n = 2^m$ für $m \in \mathbb{N}$

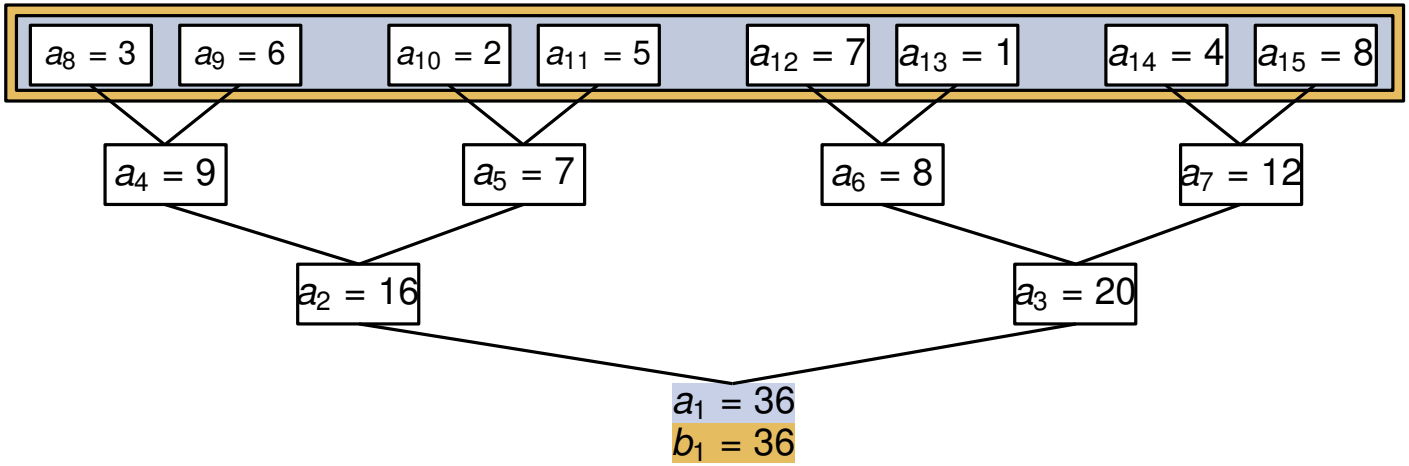


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

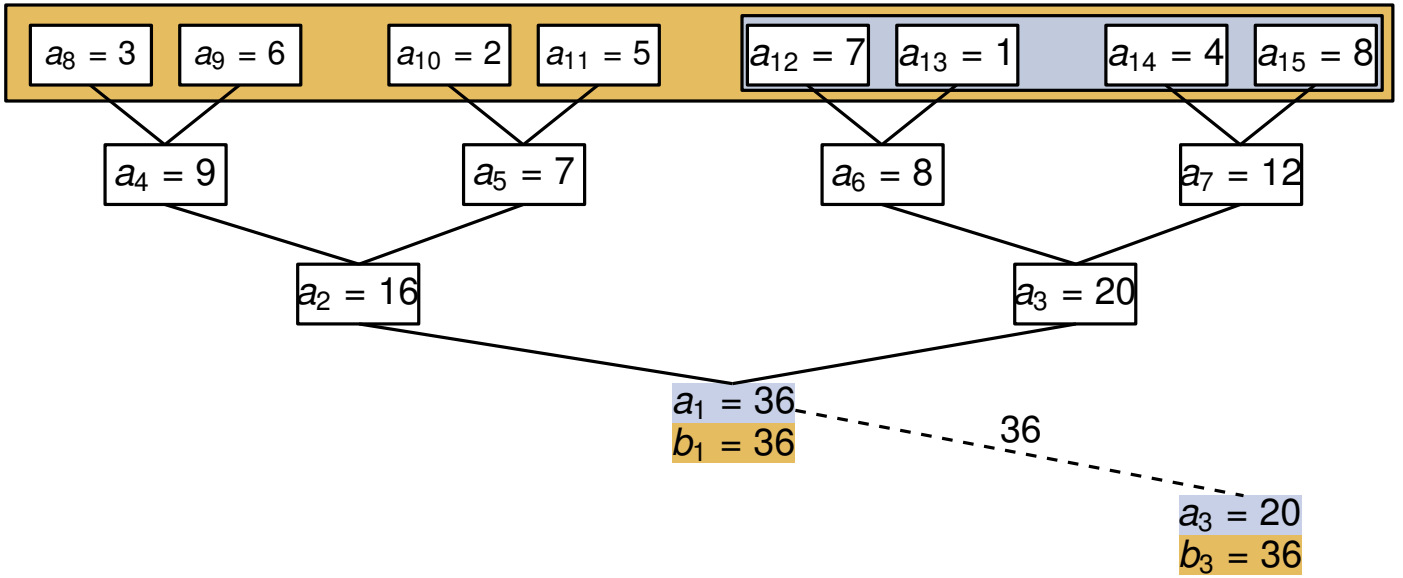


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

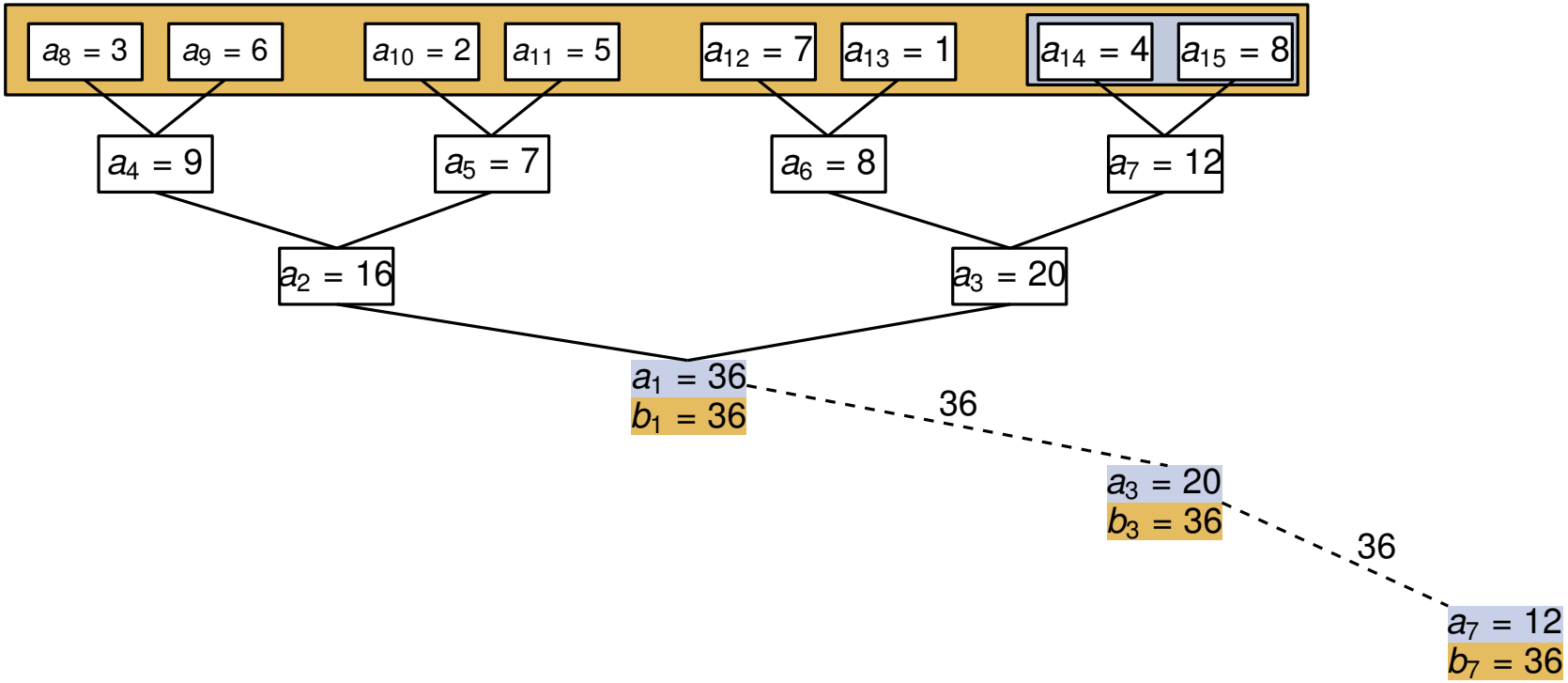
für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$



Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

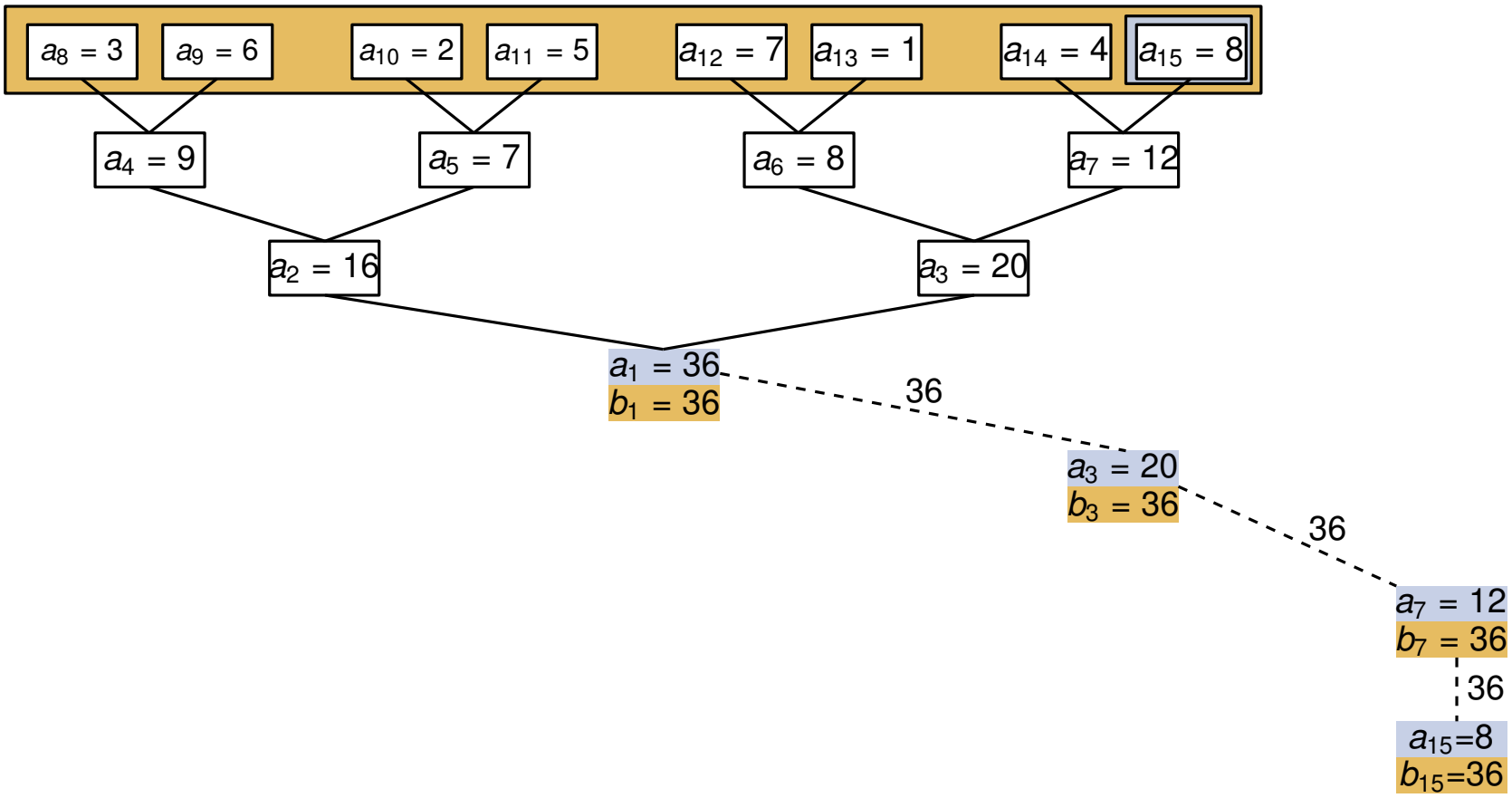
für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$



Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

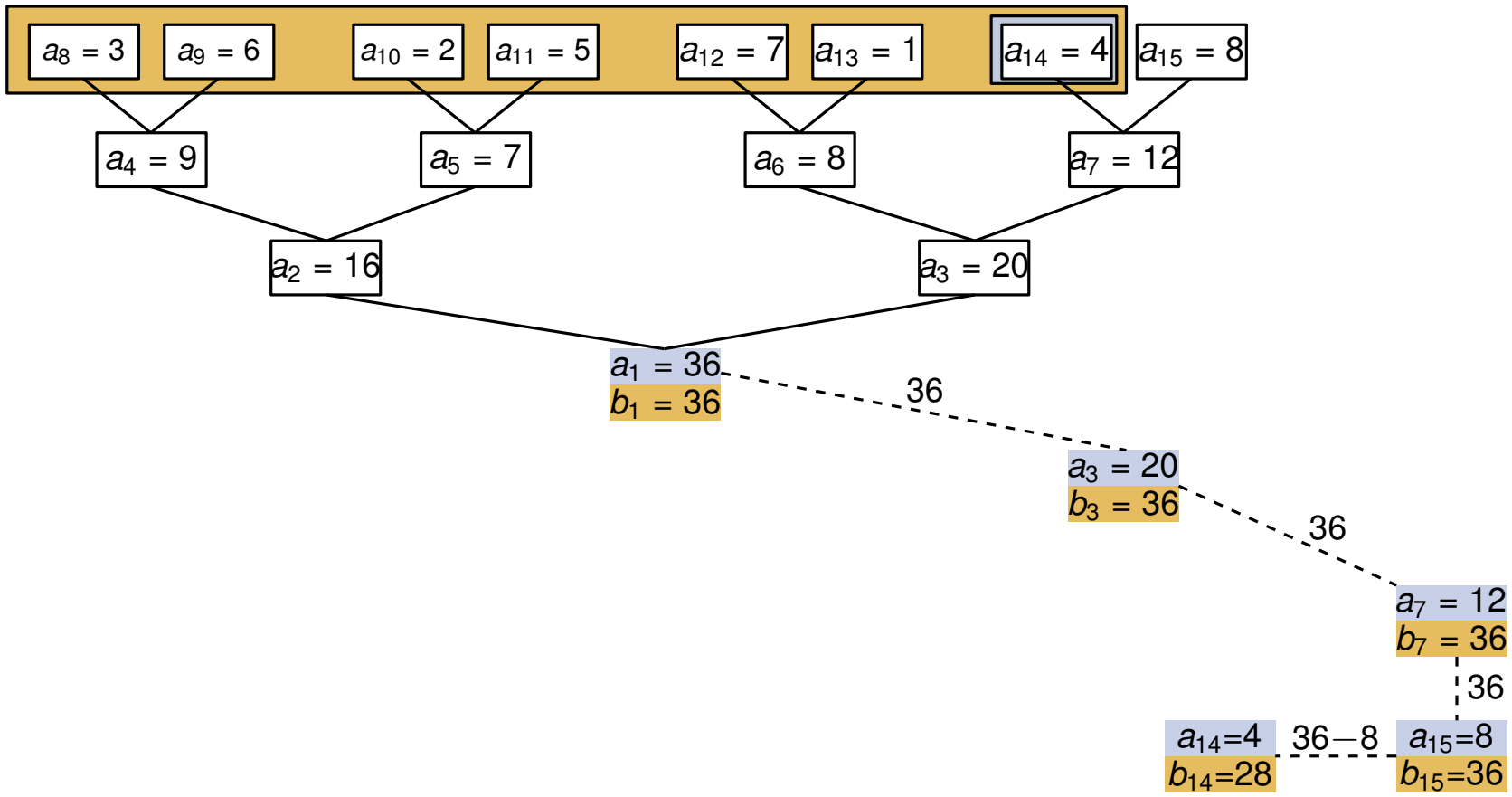


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

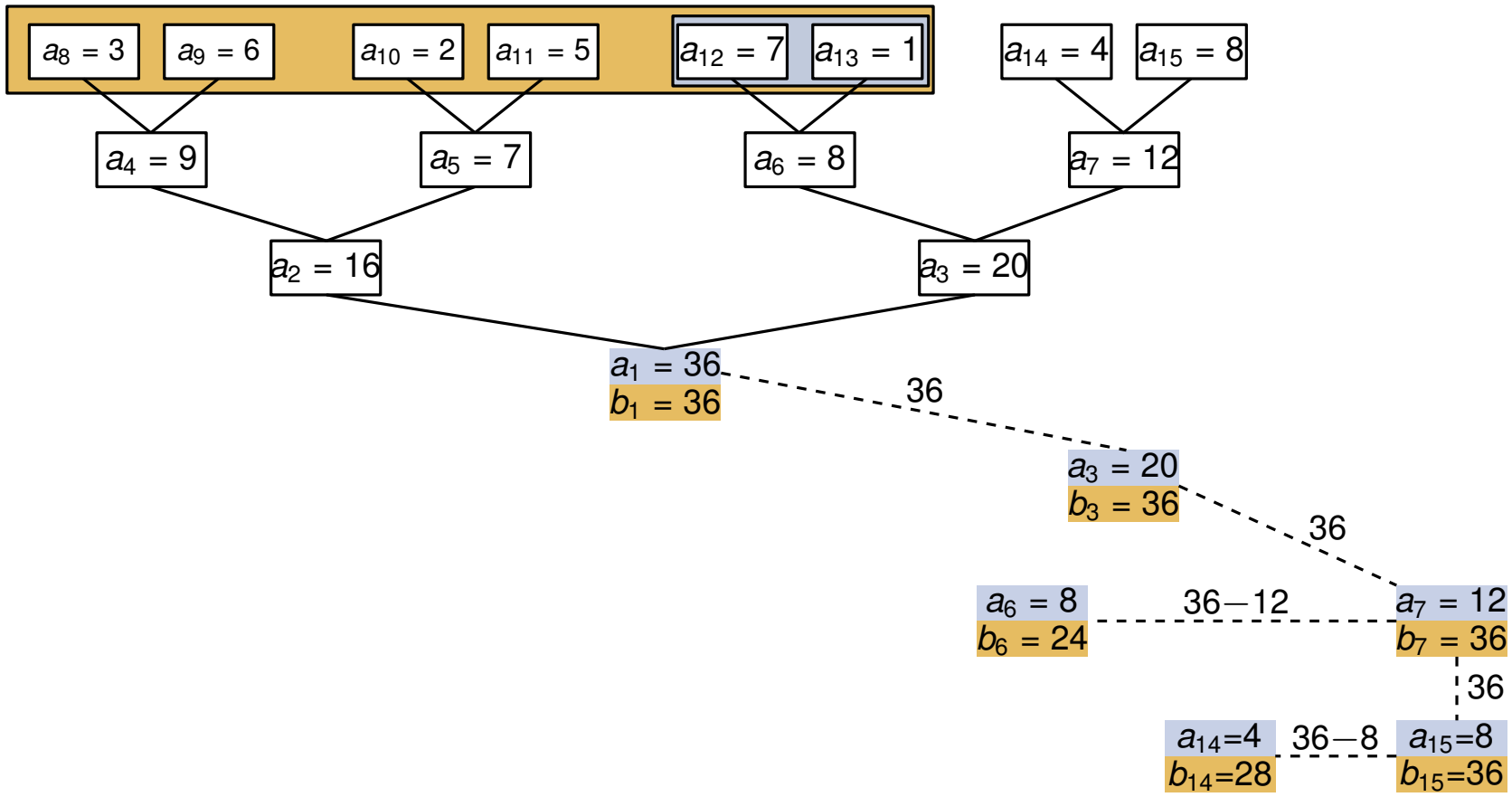


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

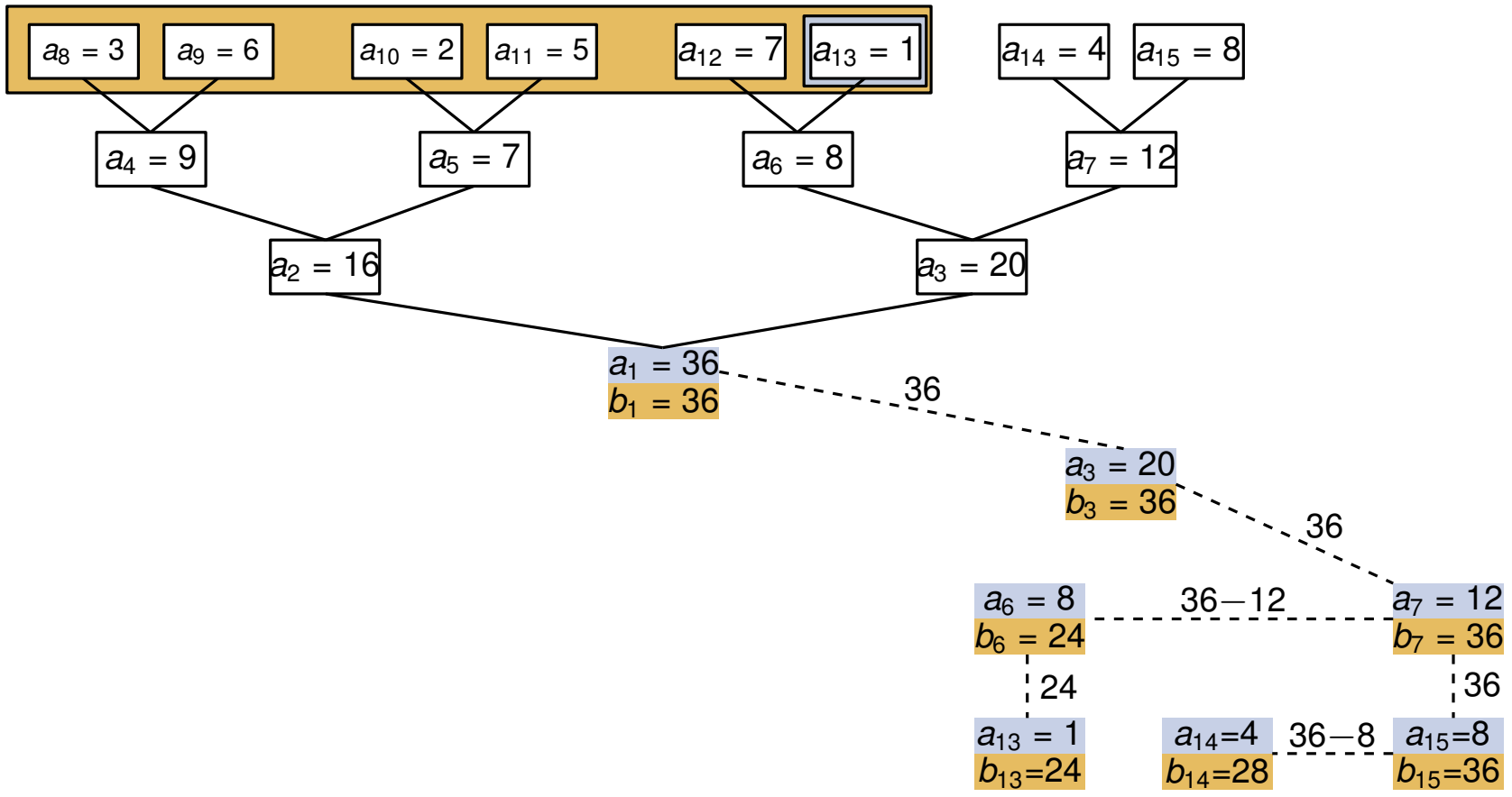


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

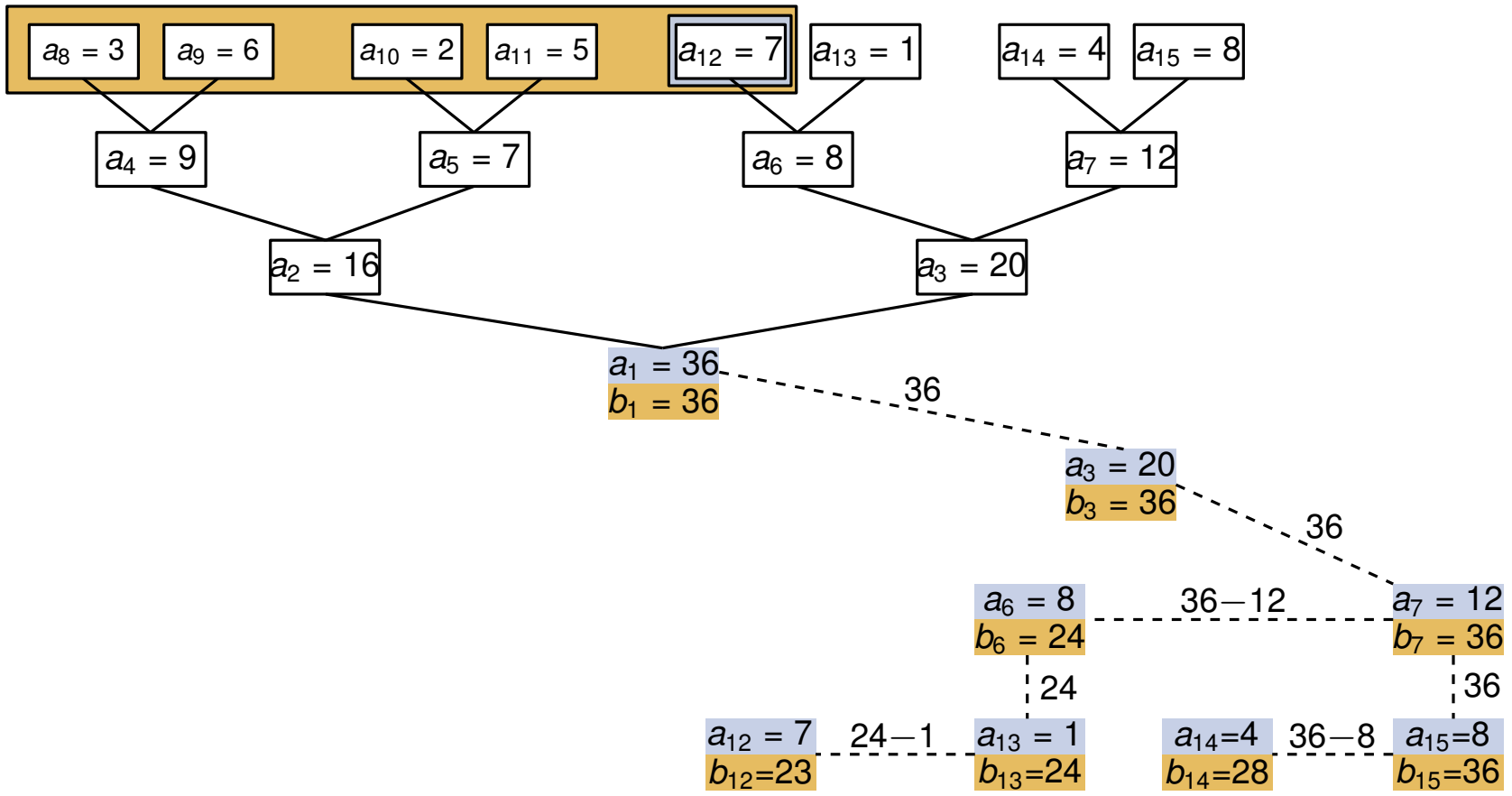


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

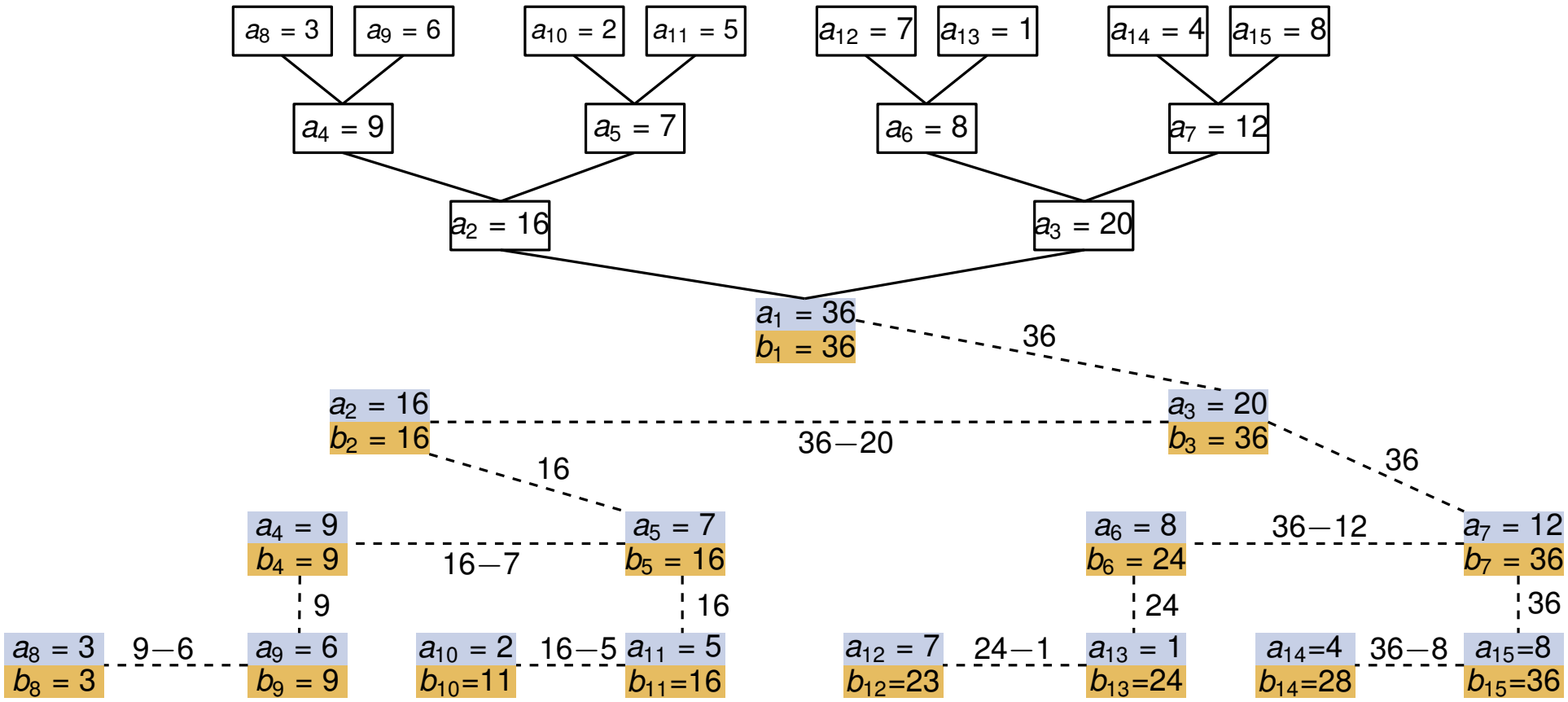


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : \ 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$

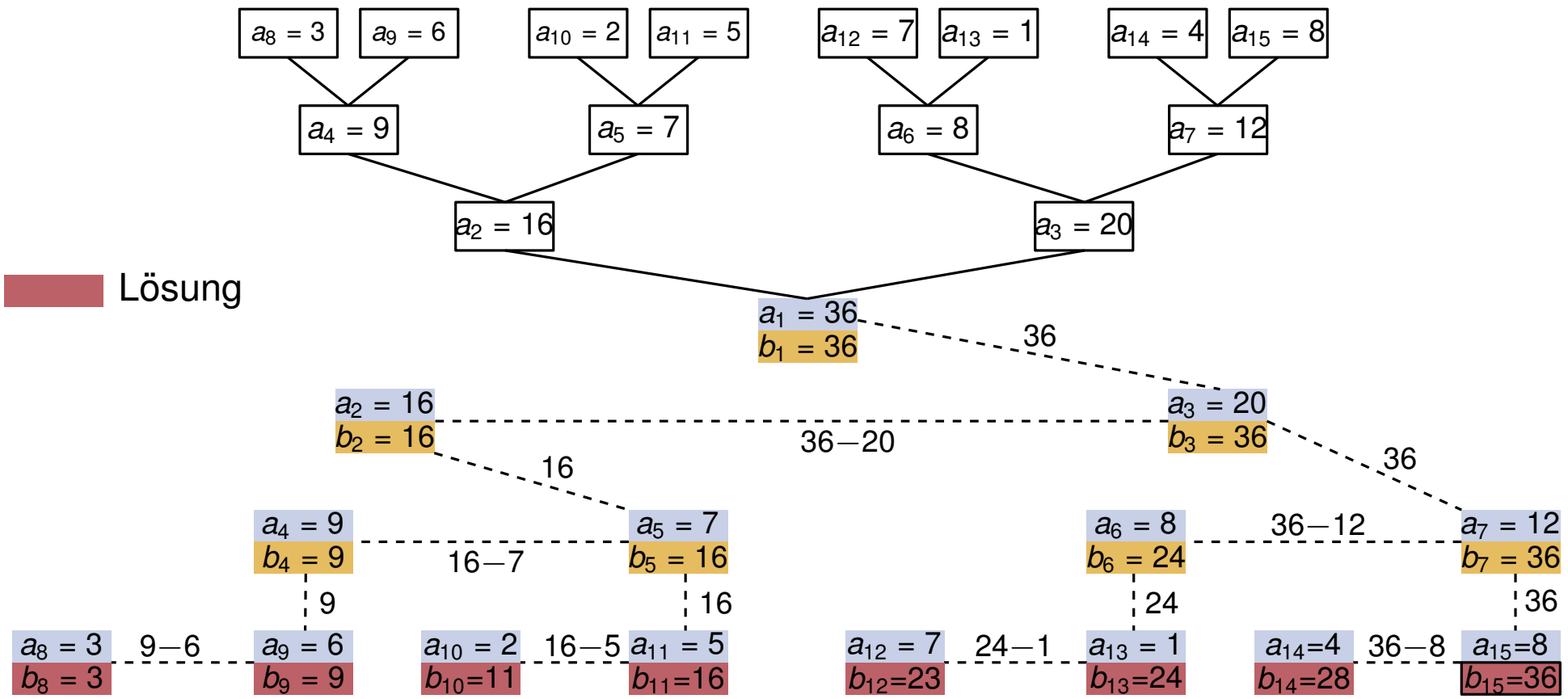


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{(i-1)/2}$
    sonst $b_i := b_{i/2} - a_{i+1}$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Präfixsumme

1. Phase: Berechne $S := \text{SUMME}(a_n, \ldots, a_{2n-1})$.

für $j = m - 1$ **bis** 0 **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    $a_i := a_{2i} + a_{2i+1}$
  $b_1 := a_1$

2. Phase: Berechne aus S und den Zwischenergebnissen die Präfixsummen.

für $j = 1$ **bis** m **tue**
  Für alle $i : 2^j \leq i \leq 2^{j+1} - 1$ **führe parallel aus**
    wenn i *ungerade* **dann** $b_i := b_{\frac{i-1}{2}}$
    sonst $b_i := b_{\frac{i}{2}} - a_{i+1}$

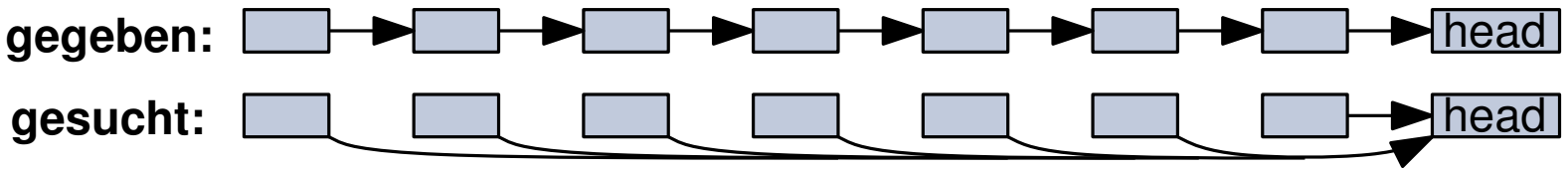
Komplexität:

- Laufzeit liegt in $O(\log n)$.
- Anzahl Prozessoren liegt in $O(n)$, da maximal $\frac{n}{2}$ Prozessoren gleichzeitig aktiv sind.
- Durch Rescheduling kann Prozessorenanzahl auf $O(\frac{n}{\log n})$ bei Laufzeit $O(\log n)$ reduziert werden.

$\Rightarrow$ Algorithmus ist kostenoptimal.

List Ranking

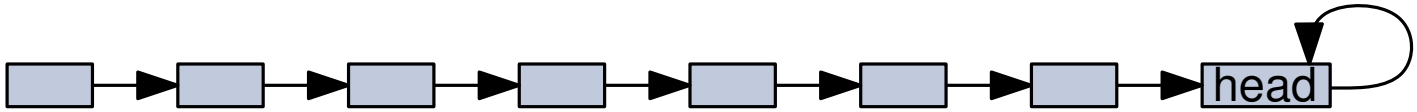
Problemstellung
gegeben: Einfach verkettete Liste L.
gesucht: Jedes Element in Liste soll auf den Kopf von L zeigen.



Vorbedingung: Array NEXT[] enthält Nachfolger NEXT[i] für jedes Element $1 \leq i \leq n$ (NEXT[head]=head).

Nachbedingung: Array NEXT[] enthält *head* für jedes Element $1 \leq i \leq n$.

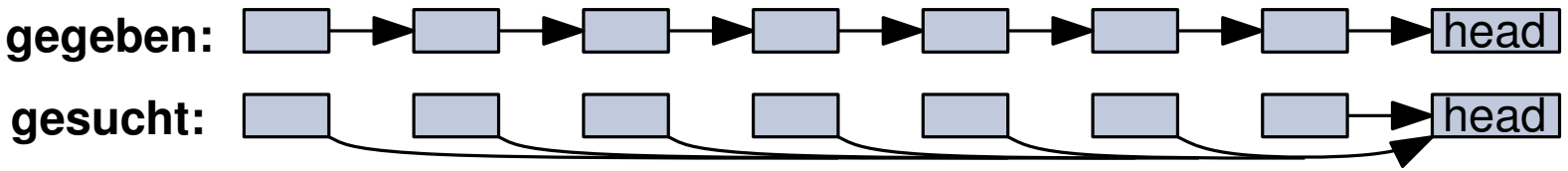
für $j = 1$ **bis** $\lceil \log n \rceil$ **tue**
  Für alle i : $1 \leq i \leq n$ **führe parallel aus**
    NEXT[i] := NEXT[NEXT[i]]



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

List Ranking

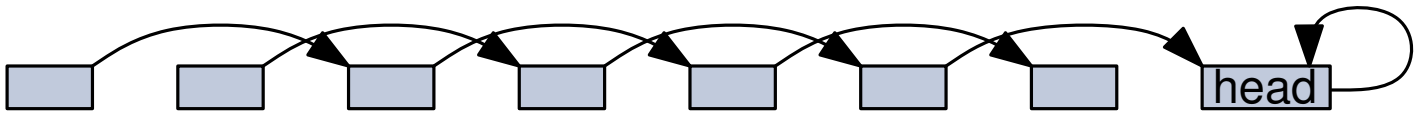
Problemstellung
gegeben: Einfach verkettete Liste L.
gesucht: Jedes Element in Liste soll auf den Kopf von L zeigen.



Vorbedingung: Array NEXT[] enthält Nachfolger NEXT[i] für jedes Element $1 \leq i \leq n$ (NEXT[head]=head).

Nachbedingung: Array NEXT[] enthält *head* für jedes Element $1 \leq i \leq n$.

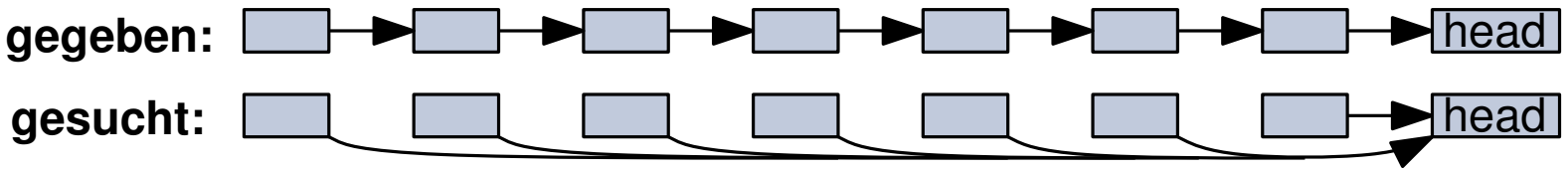
für $j = 1$ **bis** $\lceil \log n \rceil$ **tue**
  Für alle i : $1 \leq i \leq n$ **führe parallel aus**
    NEXT[i] := NEXT[NEXT[i]]



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

List Ranking

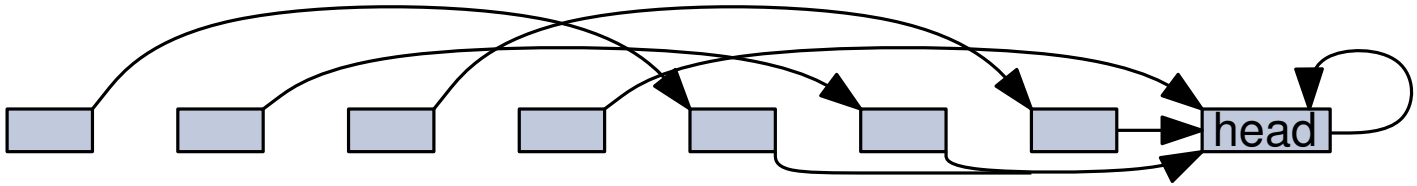
Problemstellung
gegeben: Einfach verkettete Liste L.
gesucht: Jedes Element in Liste soll auf den Kopf von L zeigen.



Vorbedingung: Array NEXT[] enthält Nachfolger NEXT[i] für jedes Element $1 \leq i \leq n$ (NEXT[head]=head).

Nachbedingung: Array NEXT[] enthält *head* für jedes Element $1 \leq i \leq n$.

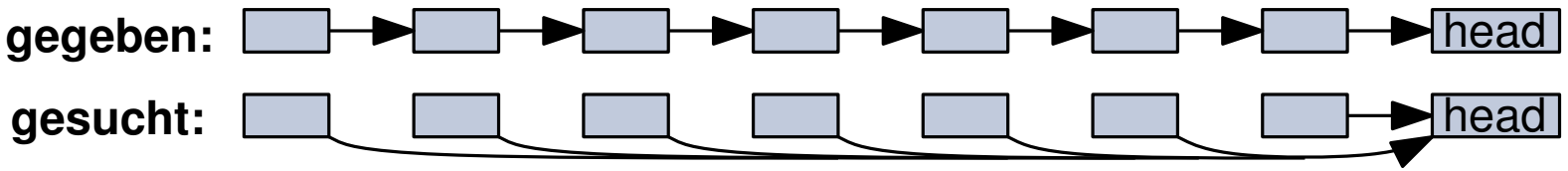
für $j = 1$ **bis** $\lceil \log n \rceil$ **tue**
  Für alle i : $1 \leq i \leq n$ **führe parallel aus**
    NEXT[i] := NEXT[NEXT[i]]



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

List Ranking

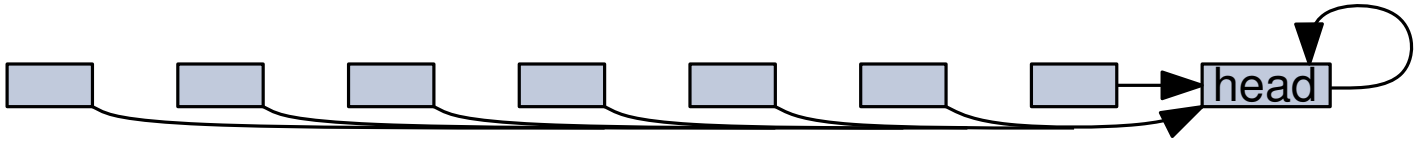
Problemstellung
gegeben: Einfach verkettete Liste L.
gesucht: Jedes Element in Liste soll auf den Kopf von L zeigen.



Vorbedingung: Array NEXT[] enthält Nachfolger NEXT[i] für jedes Element $1 \leq i \leq n$ (NEXT[head]=head).

Nachbedingung: Array NEXT[] enthält *head* für jedes Element $1 \leq i \leq n$.

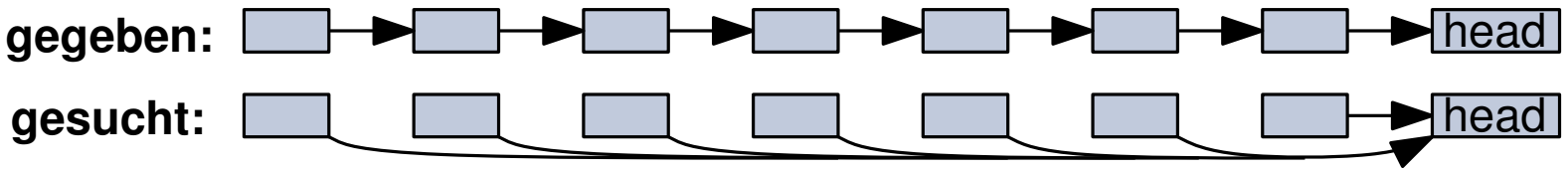
für $j = 1$ **bis** $\lceil \log n \rceil$ **tue**
  Für alle i : $1 \leq i \leq n$ **führe parallel aus**
    NEXT[i] := NEXT[NEXT[i]]



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

List Ranking

Problemstellung
gegeben: Einfach verkettete Liste L.
gesucht: Jedes Element in Liste soll auf den Kopf von L zeigen.



Vorbedingung: Array NEXT[] enthält Nachfolger NEXT[i] für jedes Element $1 \leq i \leq n$ (NEXT[head]=head).

Nachbedingung: Array NEXT[] enthält *head* für jedes Element $1 \leq i \leq n$.

für $j = 1$ **bis** $\lceil \log n \rceil$ **tue**
  Für alle i : $1 \leq i \leq n$ **führe parallel aus**
    NEXT[i] := NEXT[NEXT[i]]

- LISTRANKING kann in $O(\log n)$ Zeit mithilfe von n Prozessoren gelöst werden.
- Verfahren funktioniert auch auf Wurzelbäumen, in denen jeder Knoten nur seinen direkten Vorgänger kennt.

  → $O(\log h)$ Laufzeit und n Prozessoren.

  h = Baumhöhe n = Anzahl Knoten

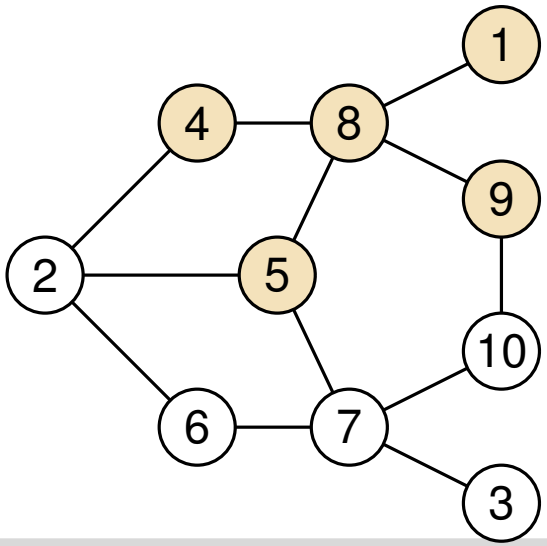
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Problemstellung
gegeben: Graph $G = (V, E)$, $V = \{1, \ldots, n\}$
gesucht: Zusammenhangskomponenten von G.

Ablauf:

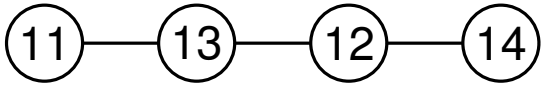
1. **Initialisierung:** Weise jedem Knoten seine eigene Komponente $K[i]$ zu.
2. Vereinige zusammenhängende Komponenten zu Superknoten $\rightarrow$ neuer Graph.
3. Wiederhole Schritt 2 und 3 $\lceil \log n \rceil$: Betrachte hierzu Graph der vorherigen Runde.



Notation:

Sei im folgenden $R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Beispiel: $R(8) = \{4, 5, 1, 8, 9\}$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

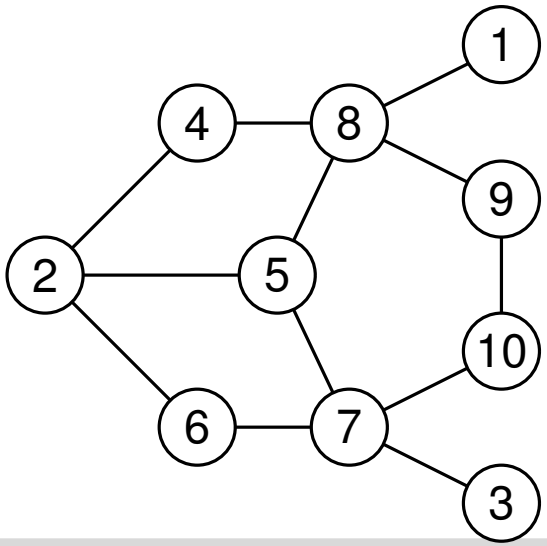
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

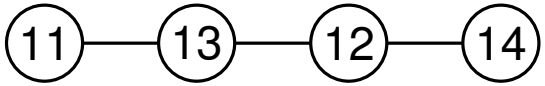
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.

= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$





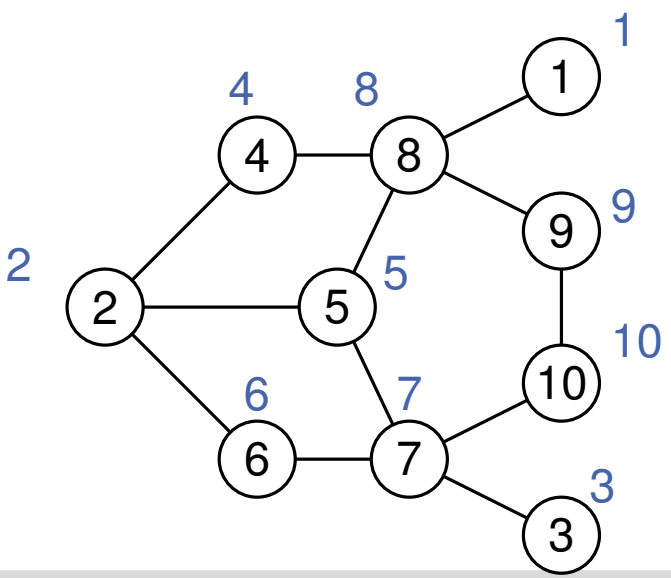
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

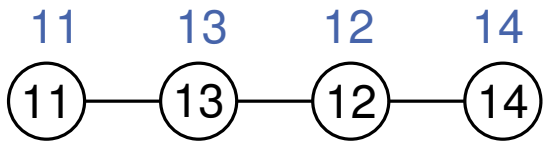
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$



Legende:
K[i]

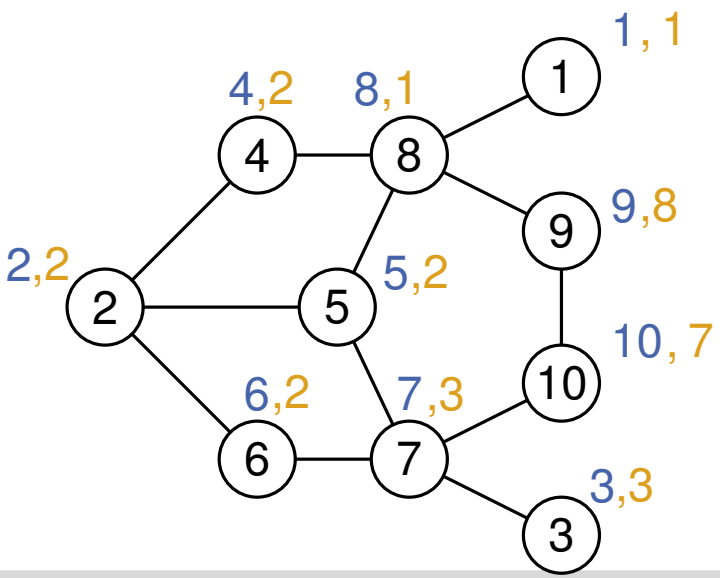
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

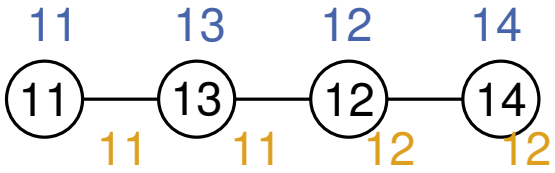
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$



Legende:
K[i] N[i]

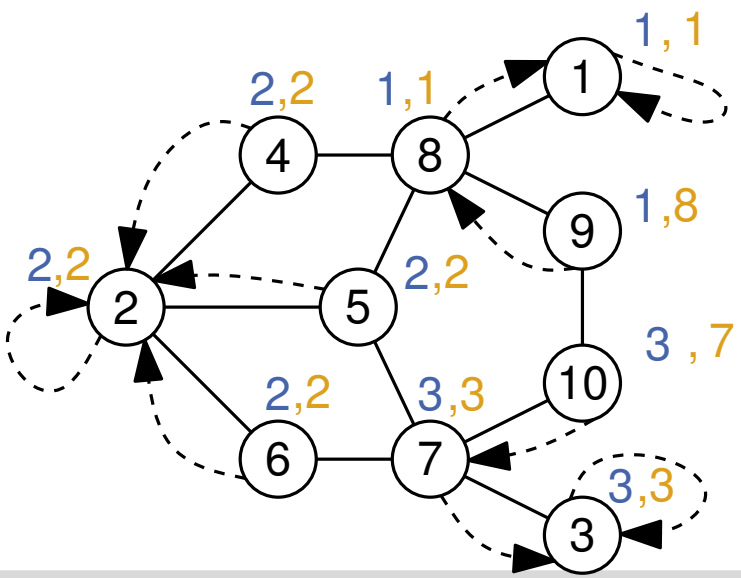
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

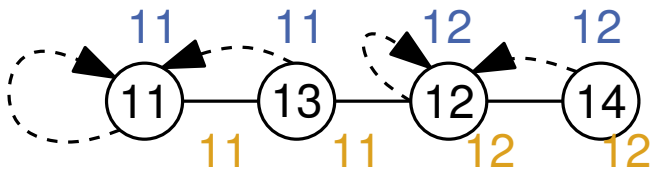
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$



Legende:
K[i] N[i]

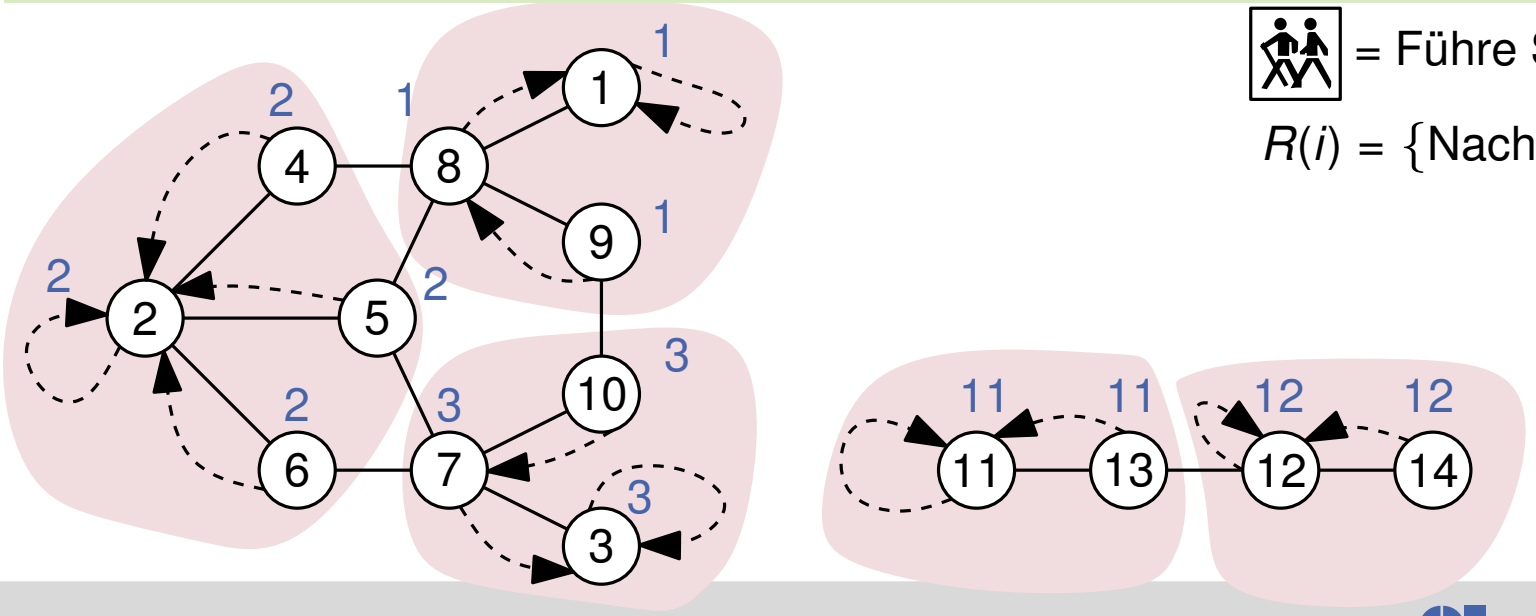
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Legende:
K[i] N[i]

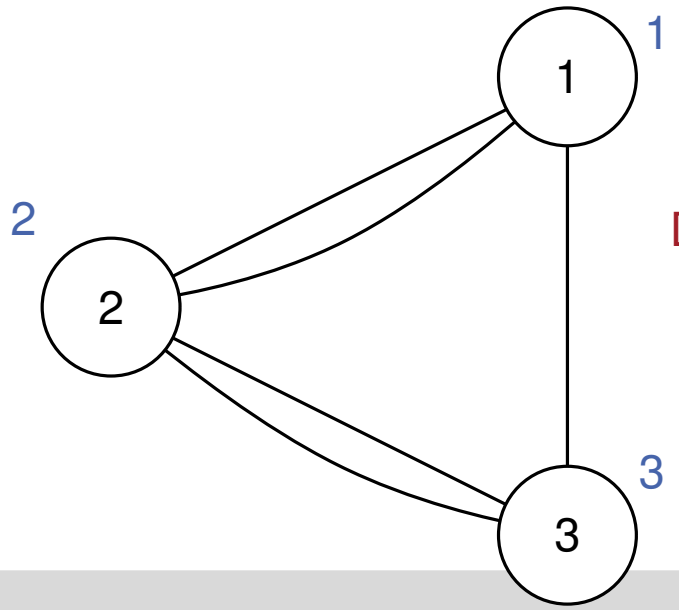
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

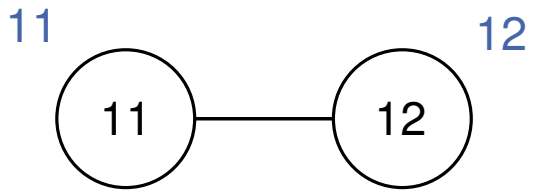
1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Doppelkanten entfernen!



Legende:
K[i] N[i]

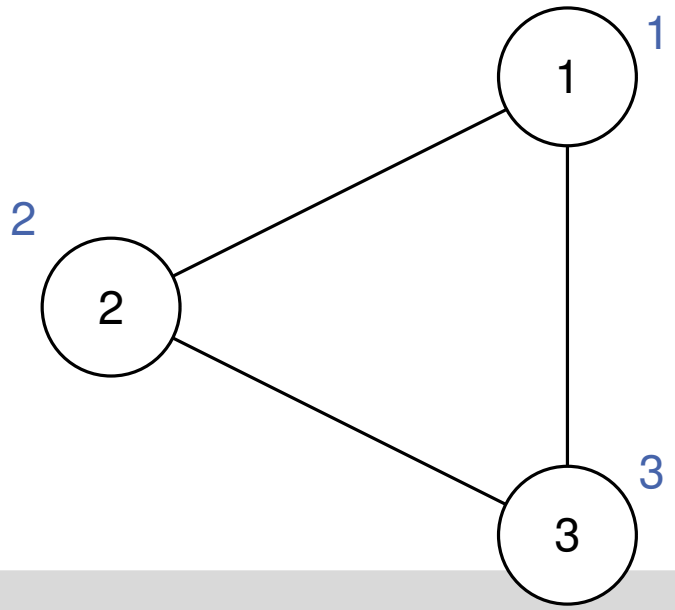
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

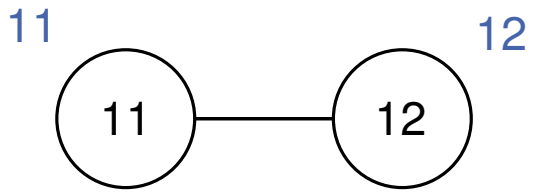
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$



Legende:
K[i] N[i]

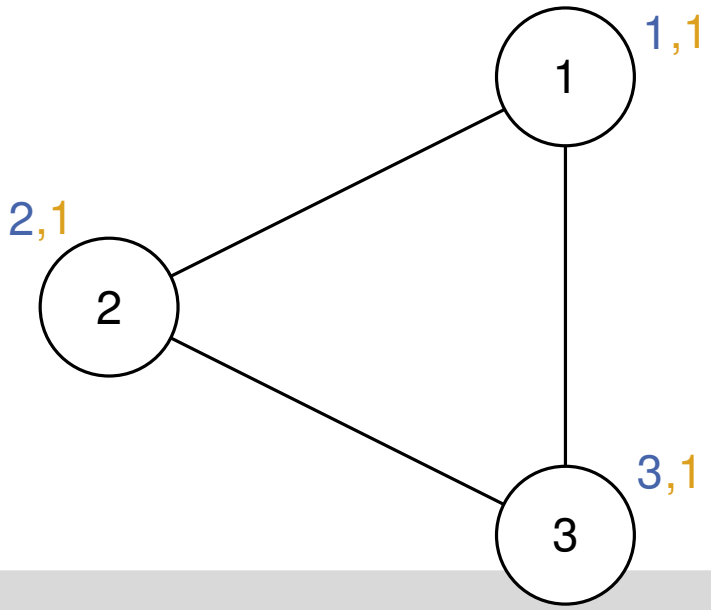
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

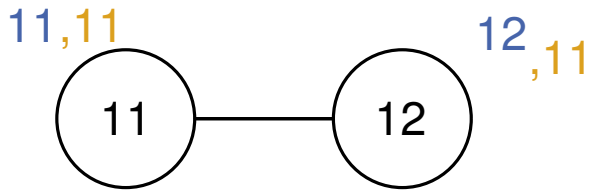
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$



Legende:
K[i] N[i]

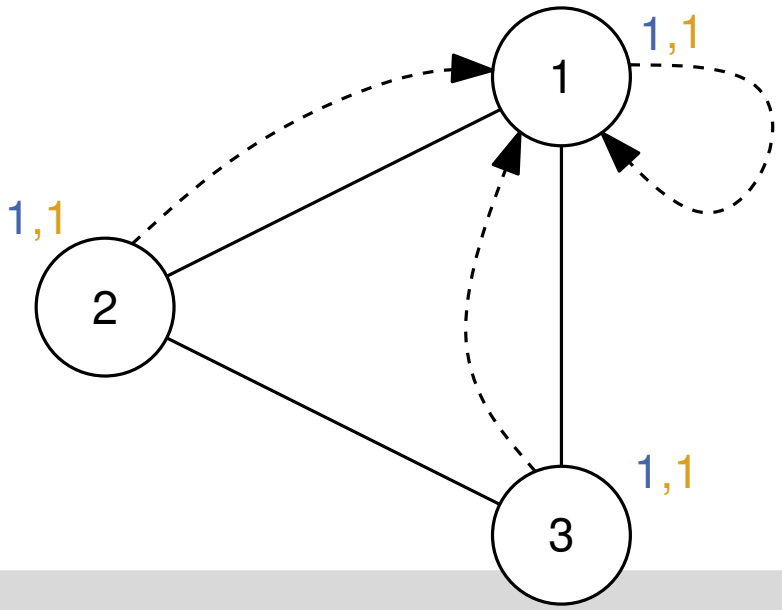
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

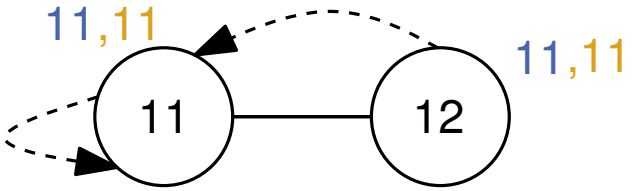
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$



Legende:
K[i] N[i]

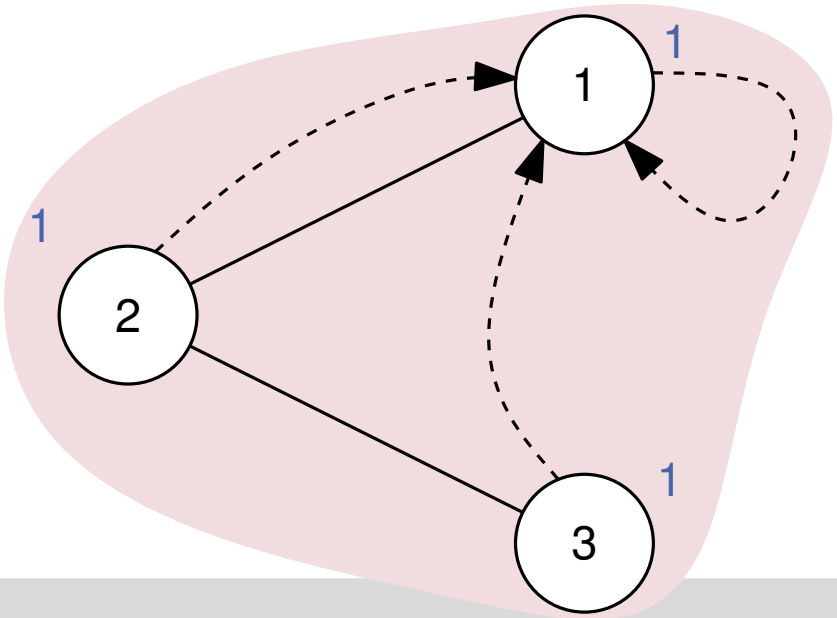
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

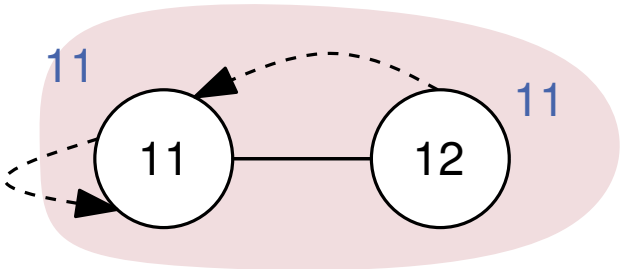
Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Legende:
K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.

= Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Legende:
K[i] N[i]

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.

   → $K[i]$ ist Zeiger auf i oder Nachbarn von i.

   → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.

   → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.

   → Arbeite auf Superknoten weiter.

Laufzeit:

Initialisierung: $O(1)$ Laufzeit $O(n)$ Prozessoren

 = Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.

→ $K[i]$ ist Zeiger auf i oder Nachbarn von i.

→ Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.

→ Superknoten erhält Komponente der Wurzel des entsprechenden Baums.

→ Arbeite auf Superknoten weiter.

Laufzeit:

 = Führe Schritt parallel aus.

Initialisierung: $O(1)$ Laufzeit $O(n)$ Prozessoren

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Schritt 1: Betrachte Knoten parallel. $O(n)$ Prozessoren

↳ Minimumsbildung auf $R(i)$ $O(\log n)$ Laufzeit $O(n)$ Prozessoren

→ $O(\log n)$ Laufzeit $O(n^2)$ Prozessoren

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.

→ $K[i]$ ist Zeiger auf i oder Nachbarn von i.

→ Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.

→ Superknoten erhält Komponente der Wurzel des entsprechenden Baums.

→ Arbeite auf Superknoten weiter.

Laufzeit:

Initialisierung:	$O(1)$ Laufzeit	$O(n)$ Prozessoren
Schritt 1:	$O(\log n)$ Laufzeit	$O(n^2)$ Prozessoren
Schritt 2:	$O(1)$ Laufzeit	$O(n)$ Prozessoren

→ Betrachte Knoten parallel.

 = Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.

→ $K[i]$ ist Zeiger auf i oder Nachbarn von i.

→ Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.

→ Superknoten erhält Komponente der Wurzel des entsprechenden Baums.

→ Arbeite auf Superknoten weiter.

Laufzeit:

Initialisierung:	$O(1)$ Laufzeit	$O(n)$ Prozessoren
Schritt 1:	$O(\log n)$ Laufzeit	$O(n^2)$ Prozessoren
Schritt 2:	$O(1)$ Laufzeit	$O(n)$ Prozessoren
Schritt 3:	$O(\log n)$ Laufzeit	$O(n)$ Prozessoren

→ Anwendung von List-Ranking.

 = Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Zusammenhang

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.

→ $K[i]$ ist Zeiger auf i oder Nachbarn von i.

→ Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.

→ Superknoten erhält Komponente der Wurzel des entsprechenden Baums.

→ Arbeite auf Superknoten weiter.

Laufzeit:

 = Führe Schritt parallel aus.

$R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Initialisierung:	$O(1)$ Laufzeit	$O(n)$ Prozessoren
Schritt 1:	$O(\log n)$ Laufzeit	$O(n^2)$ Prozessoren
Schritt 2:	$O(1)$ Laufzeit	$O(n)$ Prozessoren
Schritt 3:	$O(\log n)$ Laufzeit	$O(n)$ Prozessoren

} $O(\log n)$ Ausführungen

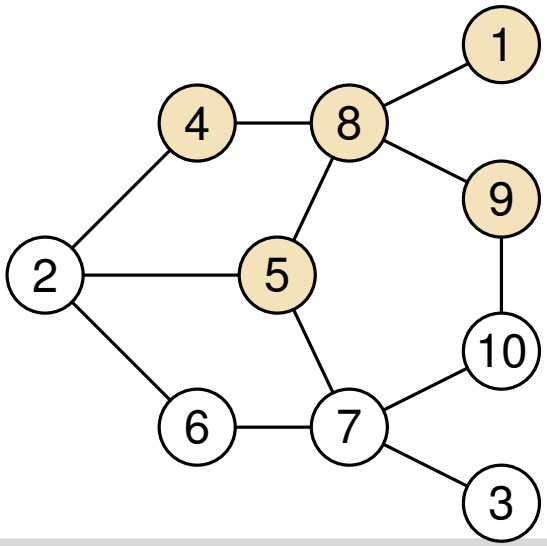
$O(n^2 \log^2 n)$ Kosten $\xrightarrow[\left\lceil \frac{n^2}{\log n} \right\rceil \text{ Prozessoren}]{\text{Rescheduling}}$ $O(n^2 \log n)$ Kosten (nicht kosten-optimal)

Spannbaum

Problemstellung
gegeben: Zusammenhängender Graph $G = (V, E)$, $V = \{1, \ldots, n\}$
gesucht: Spannbaum von G.

Ablauf: Anpassung des Algorithmus für Zusammenhangskomponenten.

1. **Initialisierung:** Weise jedem Knoten seine eigene Komponente $K[i]$ zu.
   Starte mit leerem Baum T.
2. Vereinige zusammenhängende Komponenten zu Superknoten $\rightarrow$ neuer Graph.
   ⟶ Füge *Zeiger-Kanten* zu T hinzu.
3. Wiederhole Schritt 2 und 3 $\lceil \log n \rceil$: Betrachte hierzu Graph der vorherigen Runde.



Notation:
Sei im folgenden $R(i) = \{\text{Nachbarn von } i\} \cup \{i\}$

Beispiel: $R(8) = \{4, 5, 8, 1, 9\}$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

 = Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.

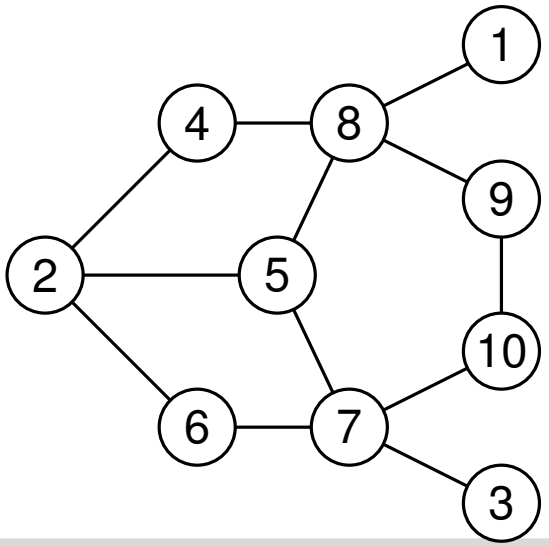
   → $K[i]$ ist Zeiger auf i oder Nachbarn von i.

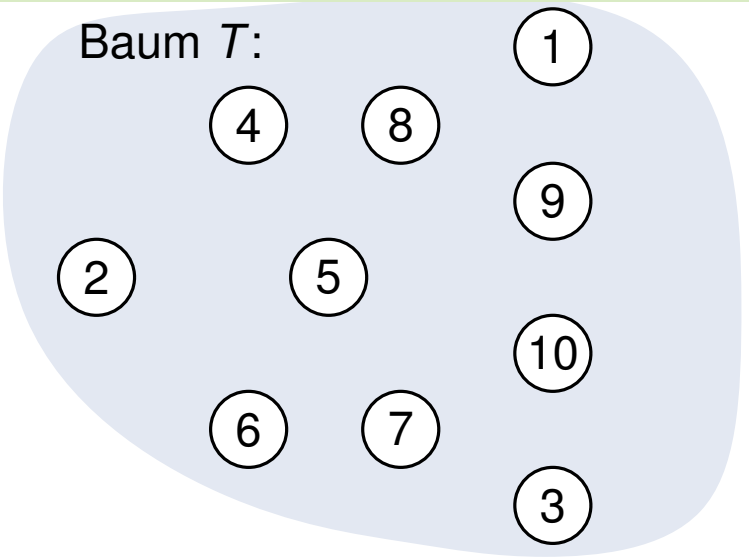
   → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.

   → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.

   → Arbeite auf Superknoten weiter.





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

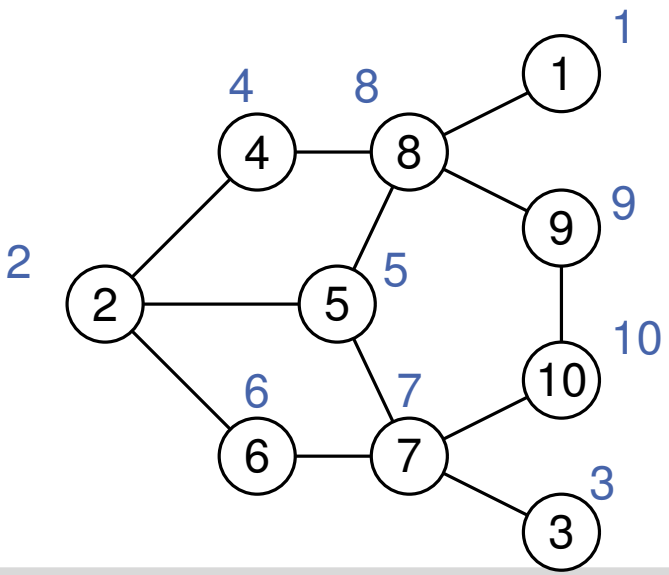
Spannbaum

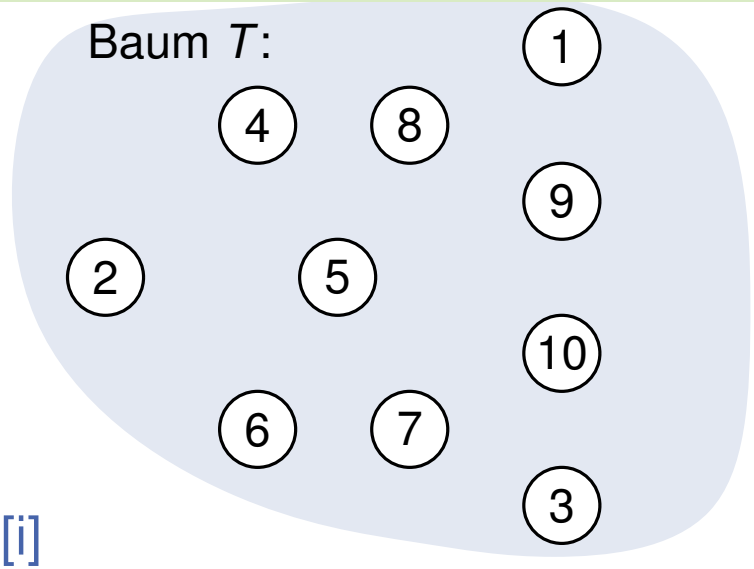
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

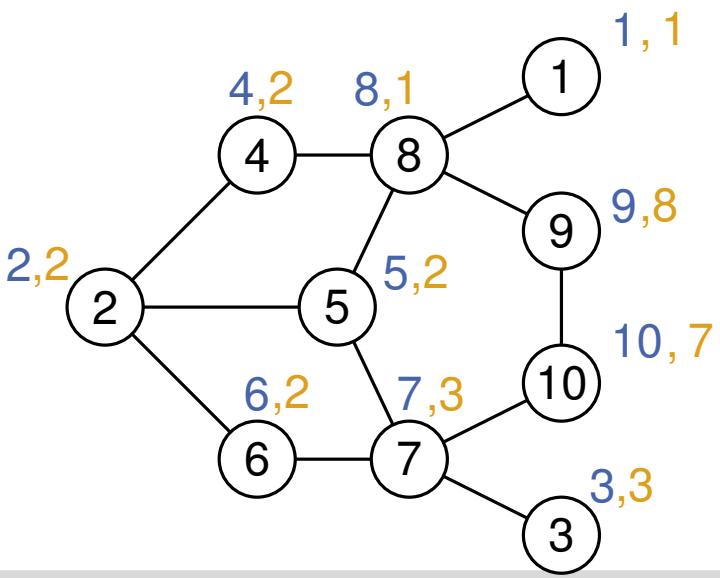
Spannbaum

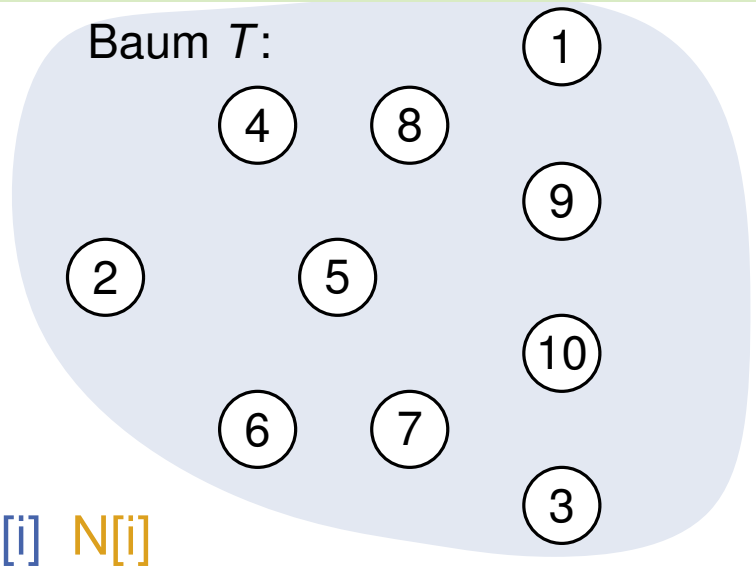
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

= Führe Schritt parallel aus.

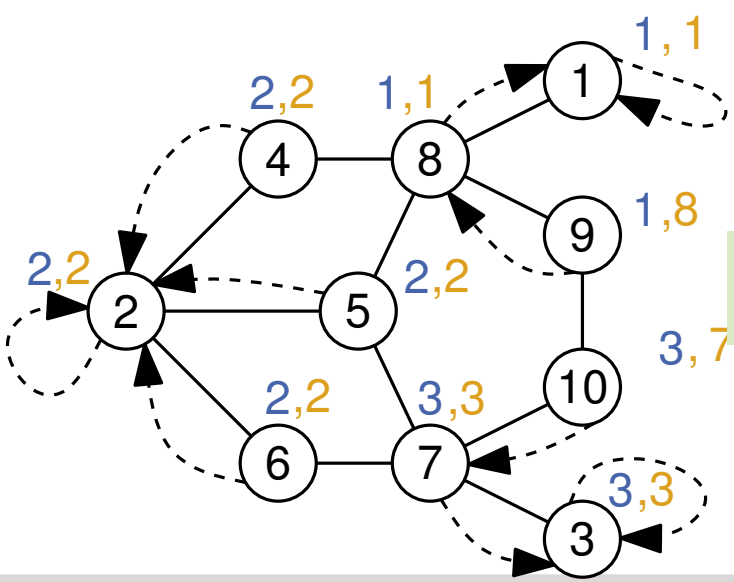
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

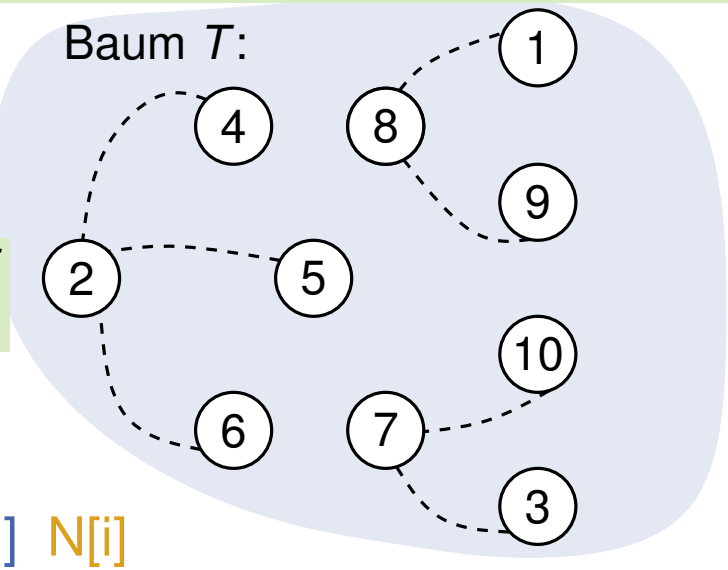
1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



Nehme *Zeiger* in T auf.



Legende: K[i] N[i]

Spannbaum

= Führe Schritt parallel aus.

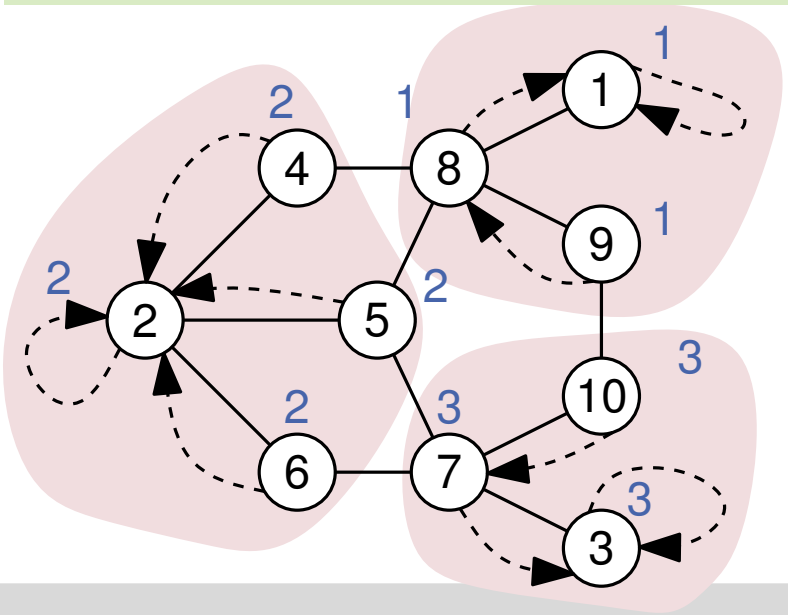
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

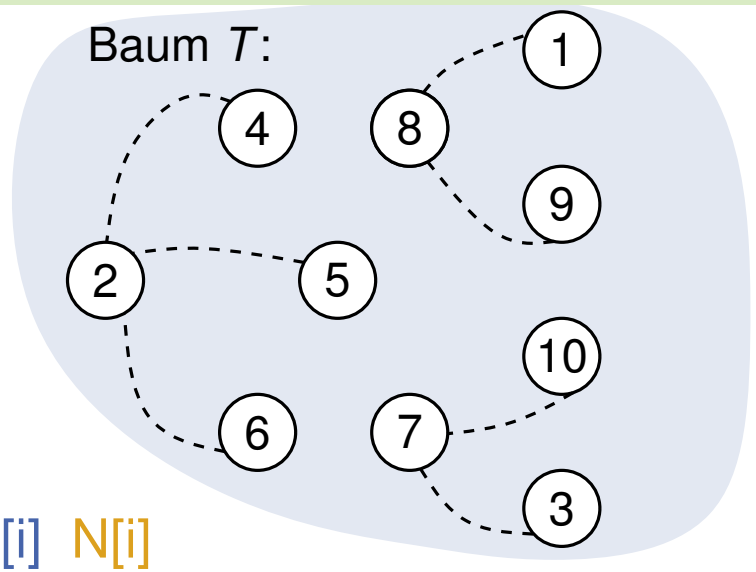
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

= Führe Schritt parallel aus.

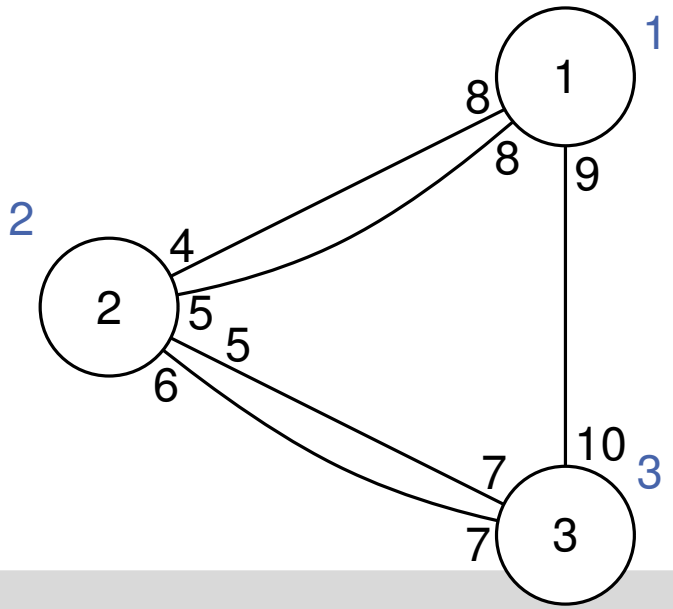
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

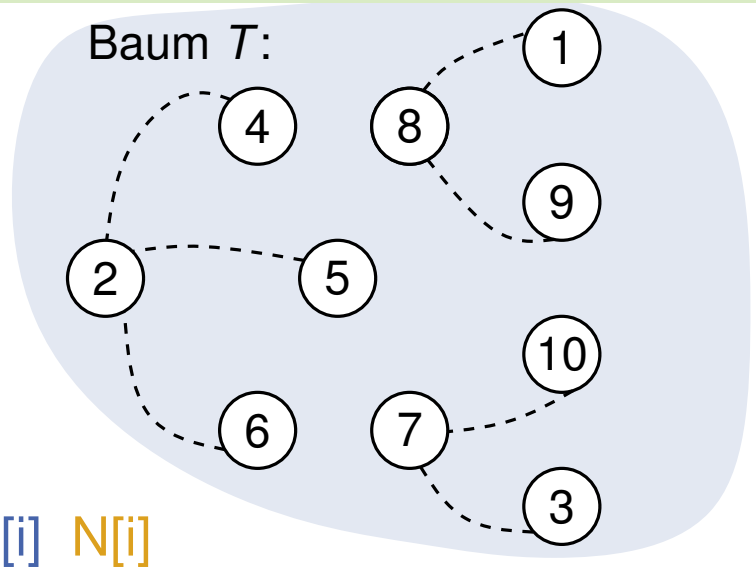
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

= Führe Schritt parallel aus.

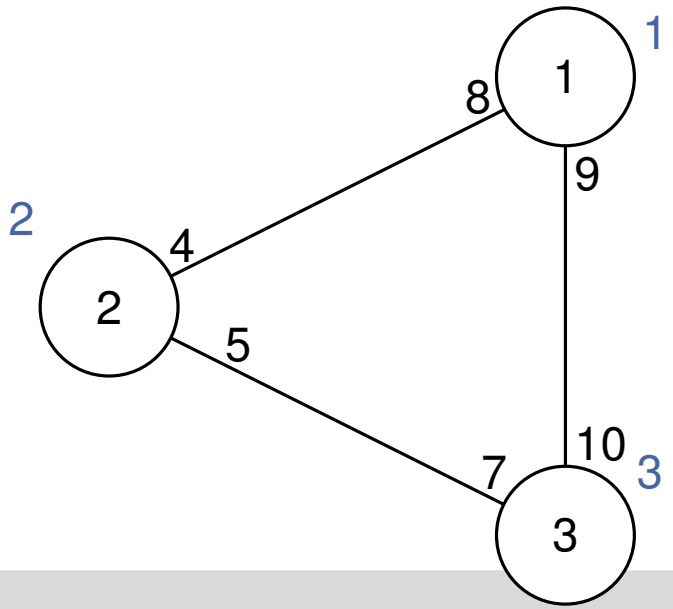
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

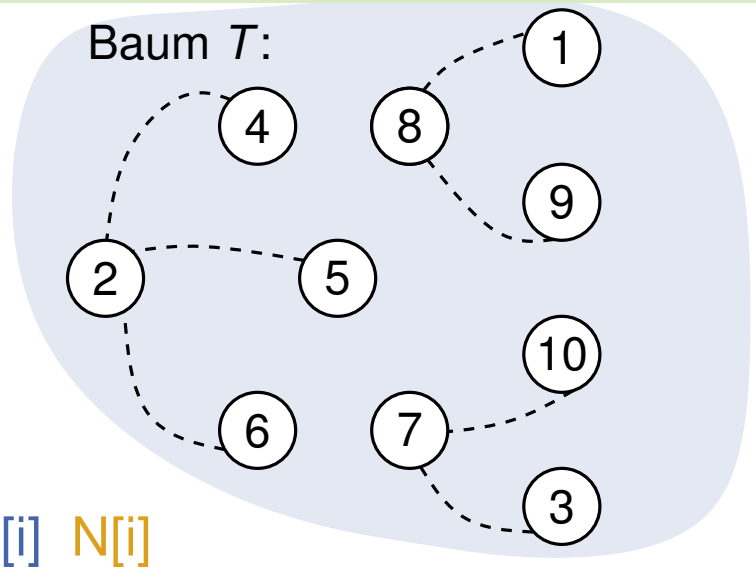
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

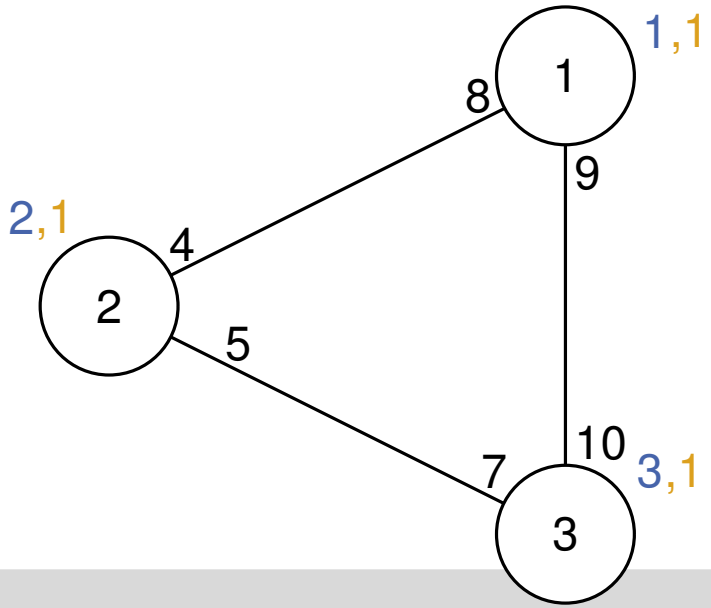
Spannbaum

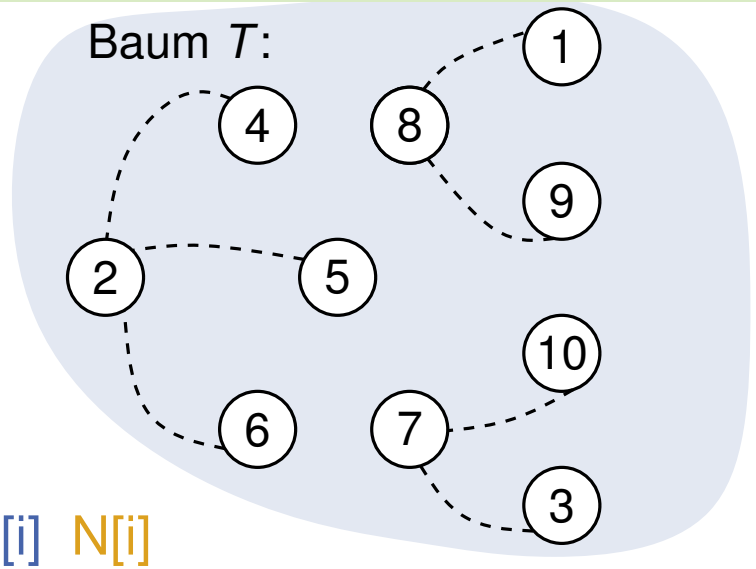
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

= Führe Schritt parallel aus.

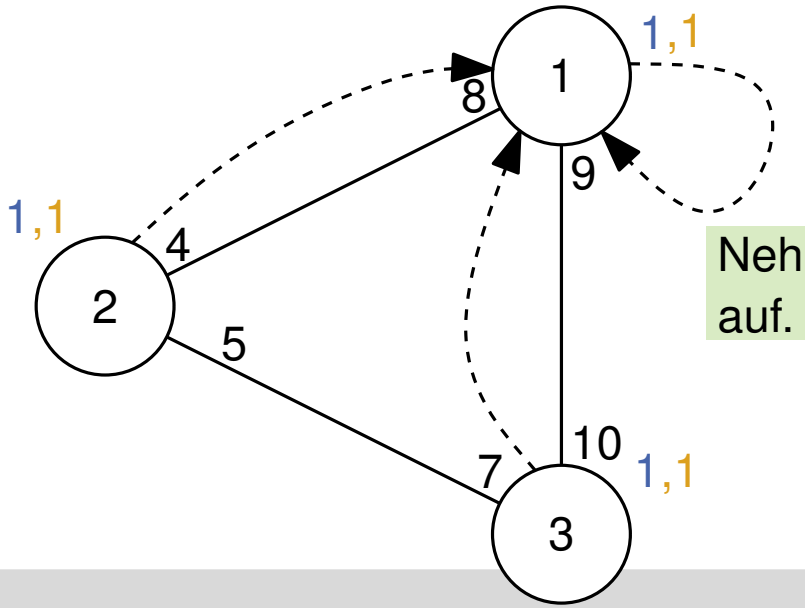
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

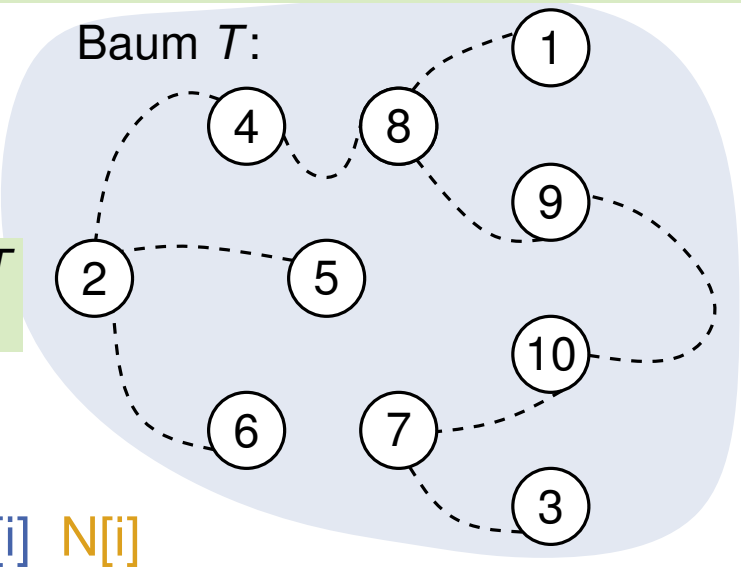
1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



Nehme *Zeiger* in T auf.



Legende: K[i] N[i]

Spannbaum

= Führe Schritt parallel aus.

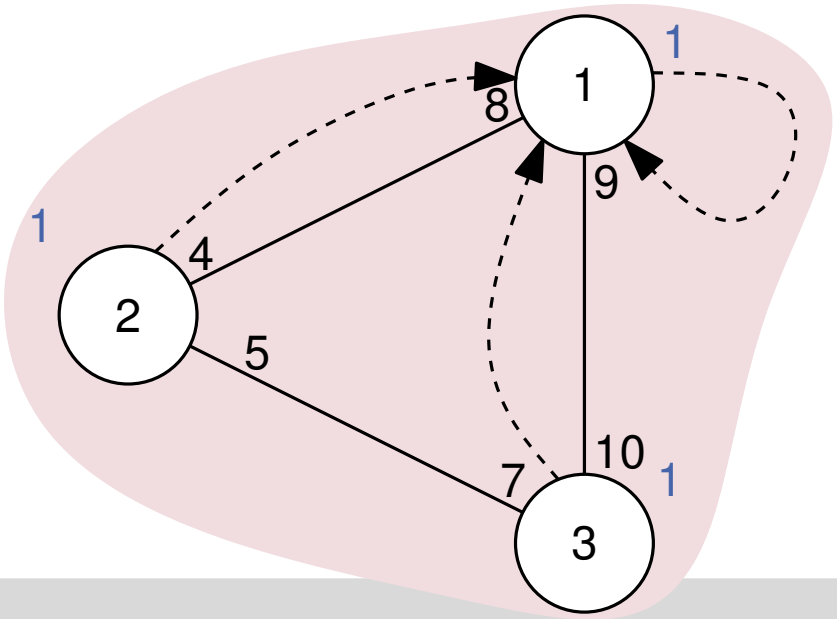
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

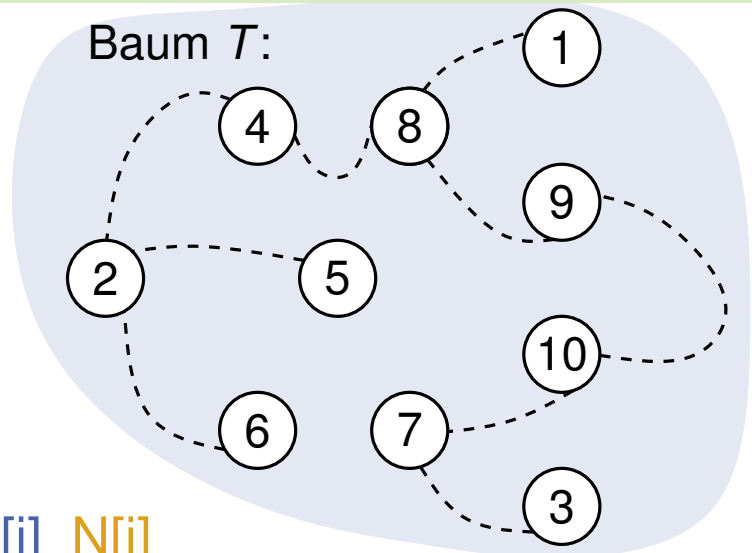
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

 = Führe Schritt parallel aus.

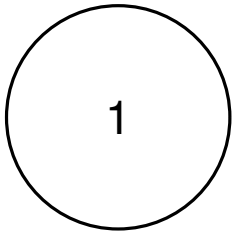
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu. 

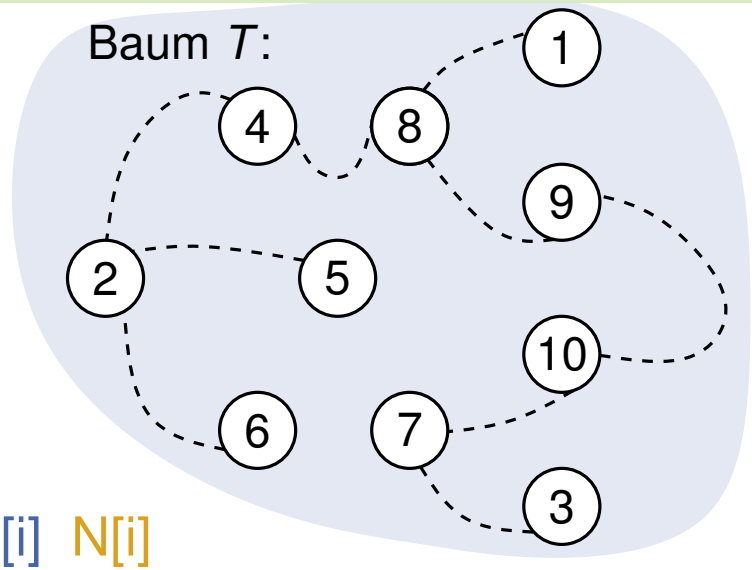
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Spannbaum

= Führe Schritt parallel aus.

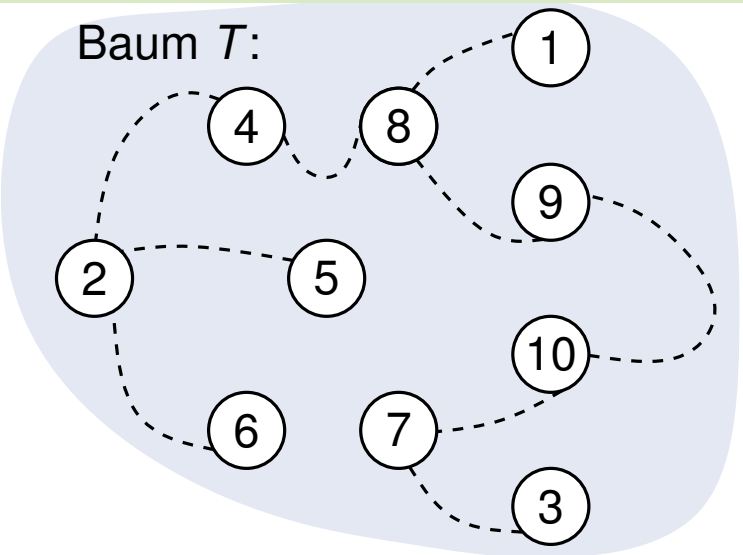
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme Knoten $N[i] \in R(i)$ mit kleinster Komponente.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.

Kosten bleiben unverändert:

$$O(n^2 \log n)$$



Minimaler Spannbaum

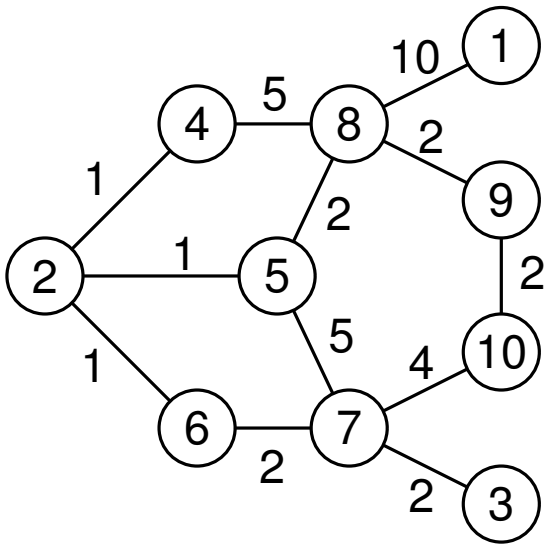
Problemstellung
gegeben: Zusammenhängender Graph $G = (V, E)$, $V = \{1, \ldots, n\}$ und $c\colon E \to \mathbb{R}^+$
gesucht: Minimaler Spannbaum von G.

Anpassung des Algorithmus für Berechnung eines Spannbaums:

Aus technischen Gründen setze $c(i, i) = \min\{c(i, k) \mid k \text{ ist Nachbar von } i\}$ für alle $i \in V$

Kleinster Nachbar von i = Knoten $j \in R(i)$, sodass

1. $c(i, j) \leq \min\{c(i, k) \mid k \in R(i)\}$, und
2. $j \leq k$ für alle $k \in R(i)$ mit $c(i, k) = c(i, j)$.



→ j ist Knoten, der mit i über Kante minimalen Gewichts verbunden ist.

→ Falls es mehrere solche Knoten gibt, wähle den mit kleinster Komponente.

Beispiel:

Kleinster Nachbar für Knoten 8 ist Knoten 5

Kleinster Nachbar für Knoten 2 ist Knoten 2

Ohne Einschränkung: G besitzt keine Schleifen.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Minimaler Spannbaum

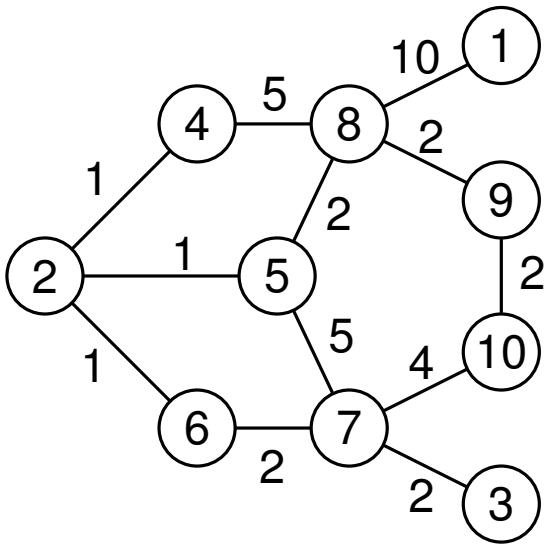
Problemstellung
gegeben: Zusammenhängender Graph $G = (V, E)$, $V = \{1, \ldots, n\}$ und $c\colon E \to \mathbb{R}^+$
gesucht: Minimaler Spannbaum von G.

Anpassung des Algorithmus für Berechnung eines Spannbaums:

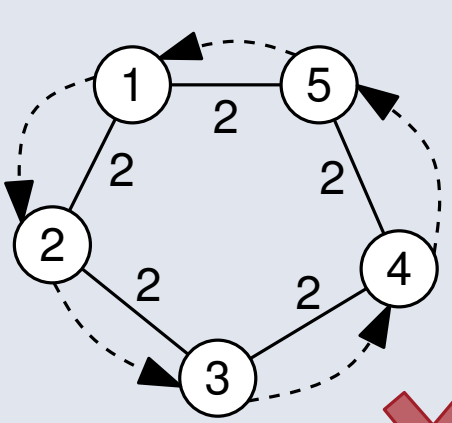
Aus technischen Gründen setze $c(i, i) = \min\{c(i, k) \mid k \text{ ist Nachbar von } i\}$ für alle $i \in V$

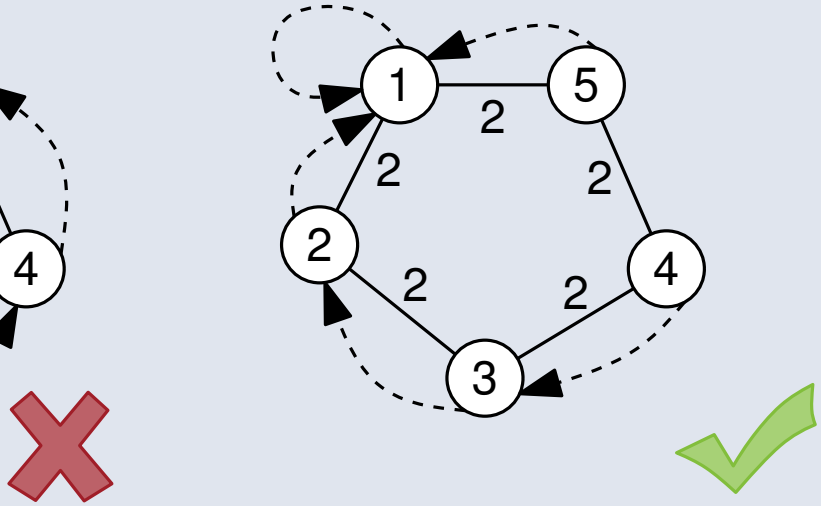
Kleinster Nachbar von i = Knoten $j \in R(i)$, sodass

1. $c(i, j) \leq \min\{c(i, k) \mid k \in R(i)\}$, und
2. $j \leq k$ für alle $k \in R(i)$ mit $c(i, k) = c(i, j)$.



2. Bedingung verhindert zyklische Ketten von kleinsten Nachbarn.





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

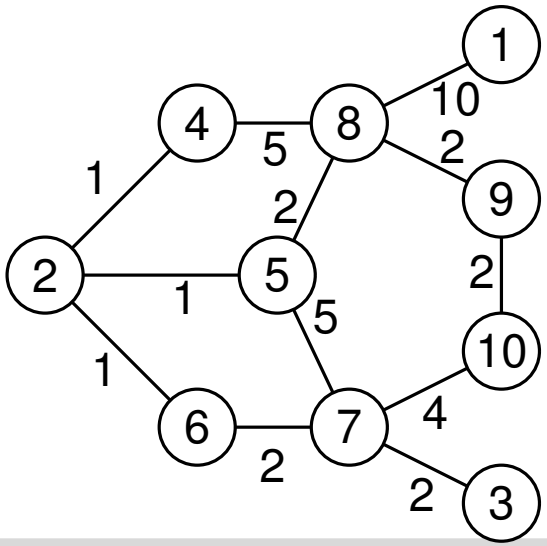
Minimaler Spannbaum

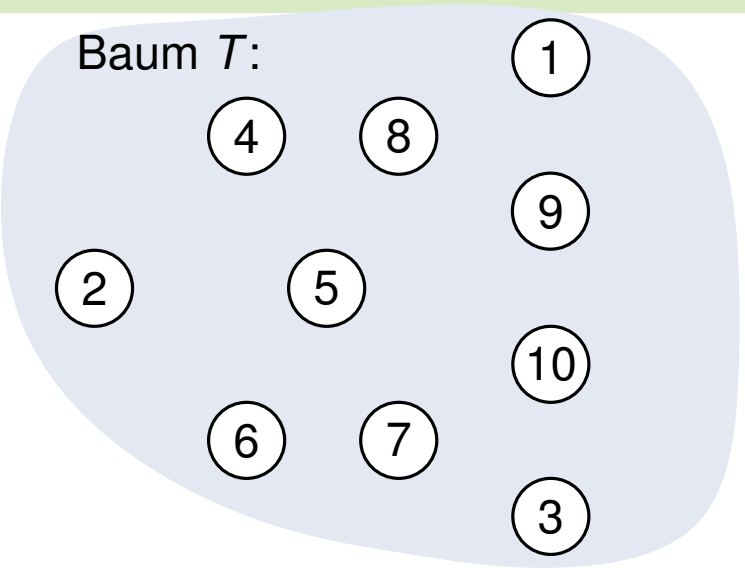
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

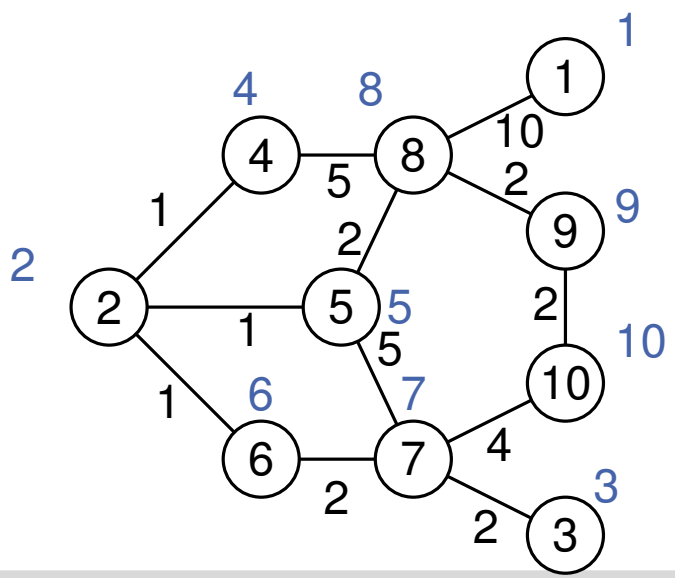
Minimaler Spannbaum

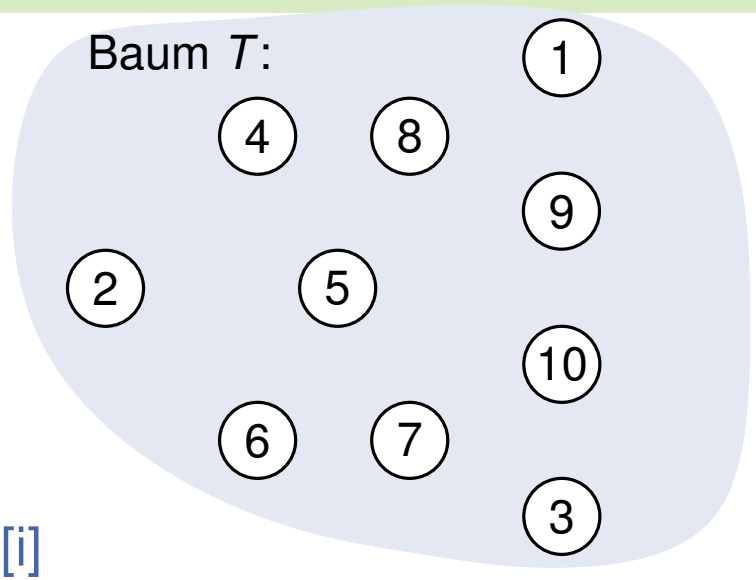
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i]

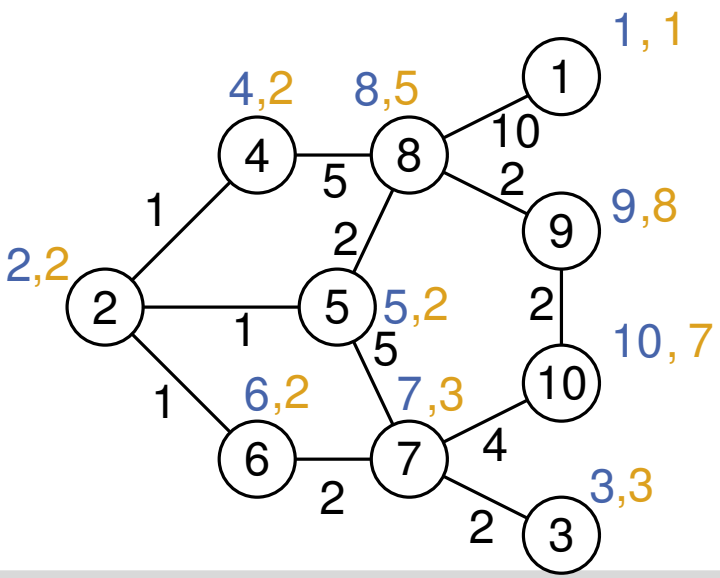
Minimaler Spannbaum

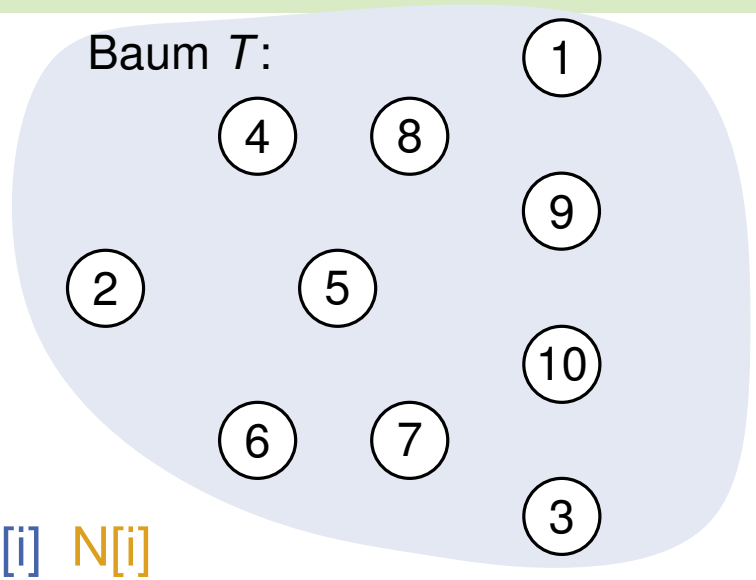
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Minimaler Spannbaum

= Führe Schritt parallel aus.

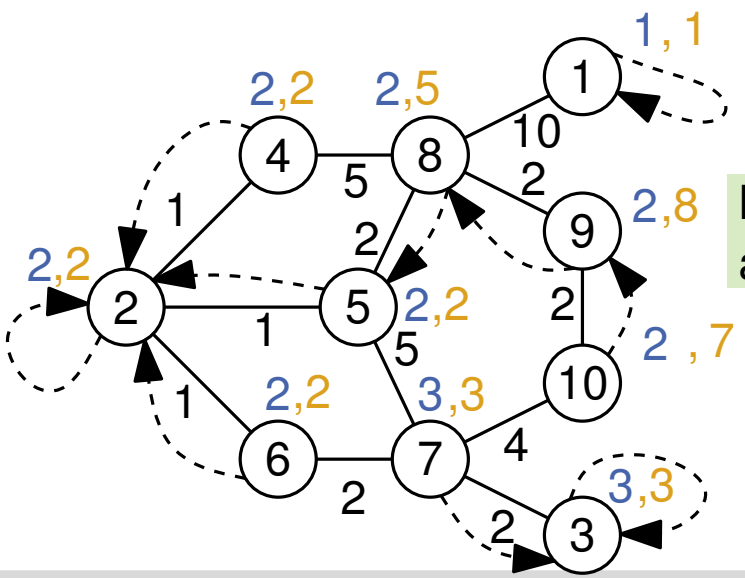
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

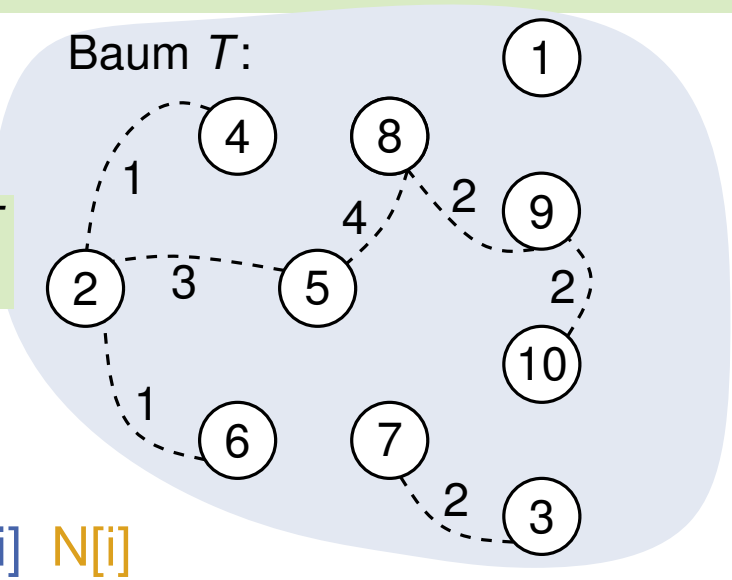
1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



Nehme *Zeiger* in T auf.



Legende: K[i] N[i]

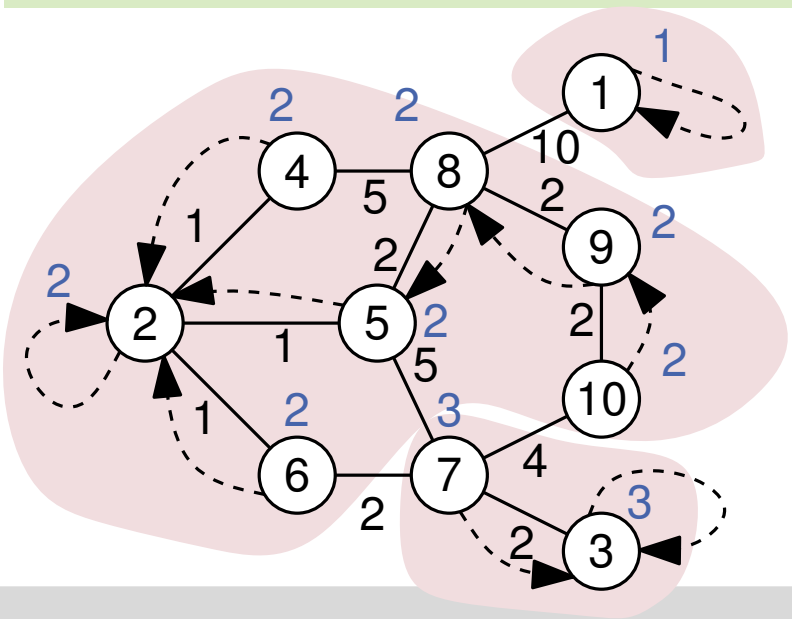
Minimaler Spannbaum

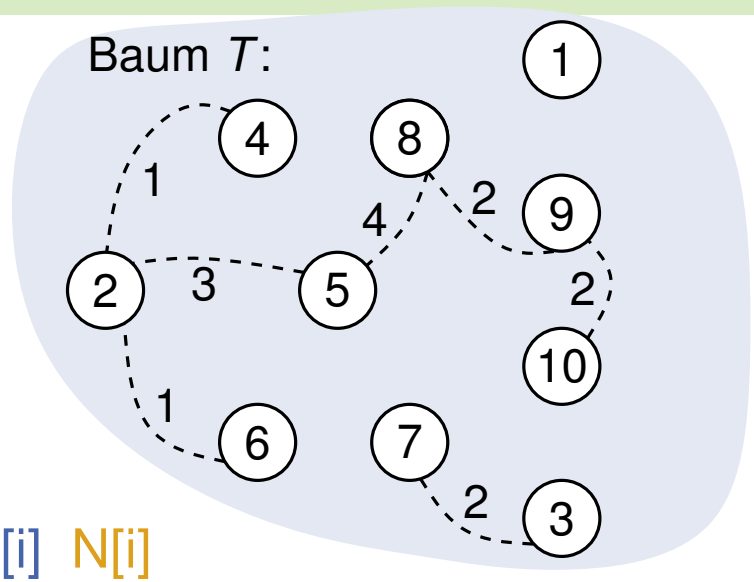
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

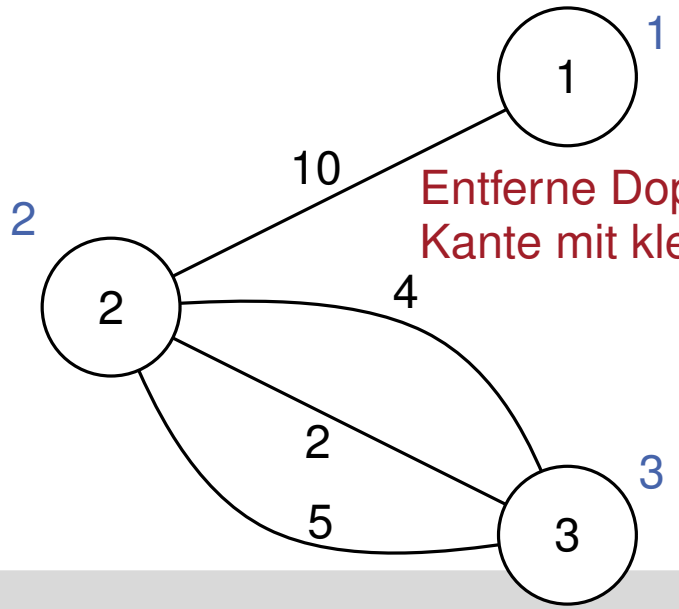
Minimaler Spannbaum

= Führe Schritt parallel aus.

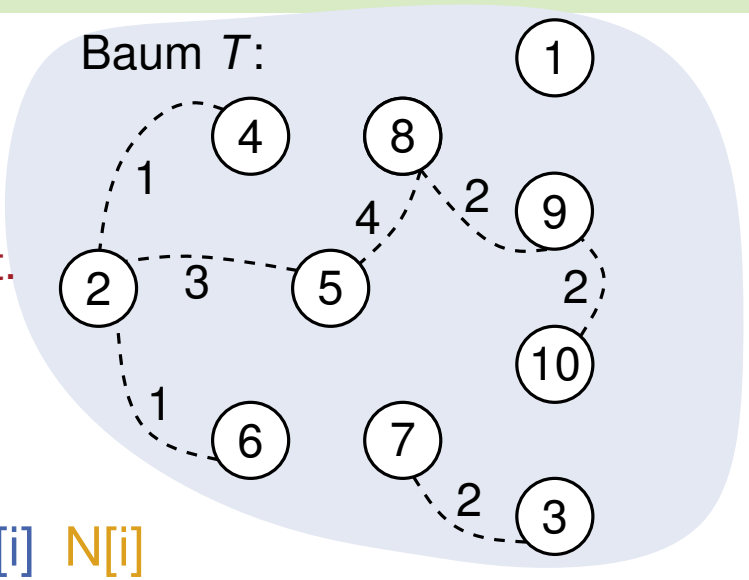
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



Entferne Doppelkanten!
Kante mit kleinstem Gewicht bleibt.



Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Minimaler Spannbaum

= Führe Schritt parallel aus.

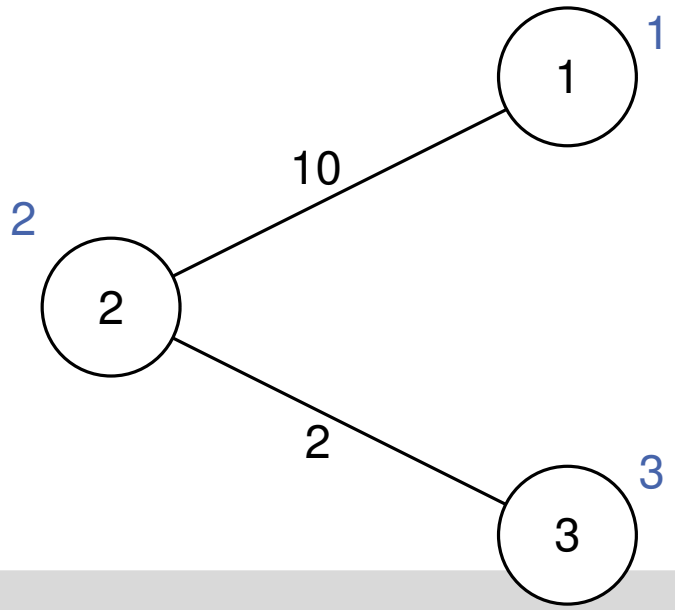
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

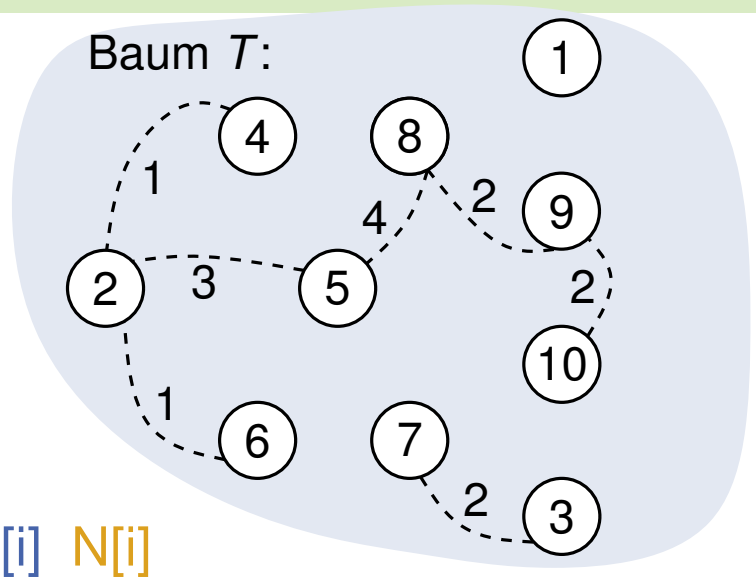
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

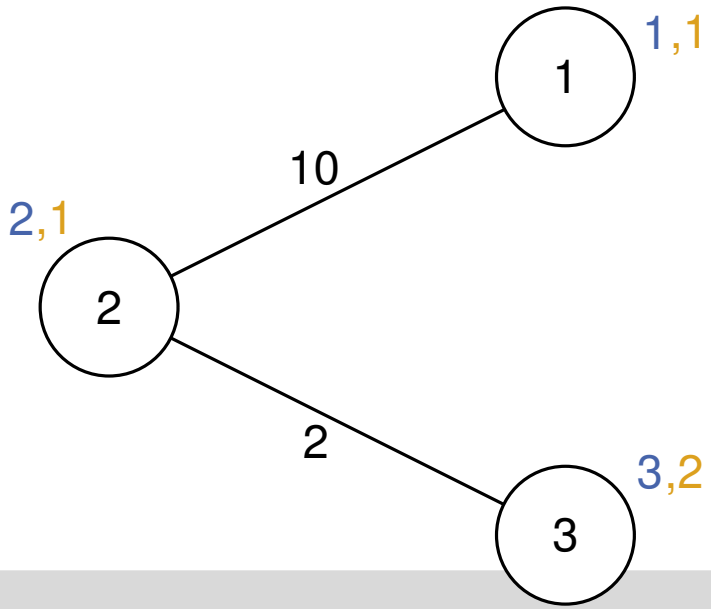
Minimaler Spannbaum

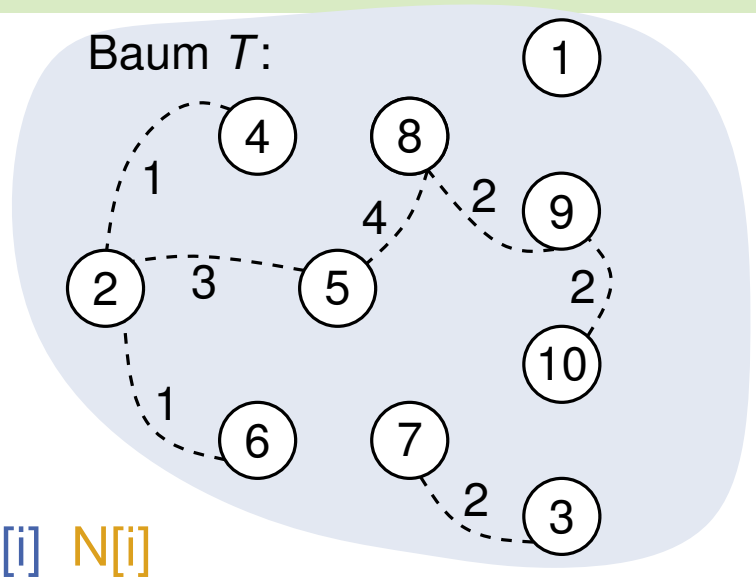
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Minimaler Spannbaum

= Führe Schritt parallel aus.

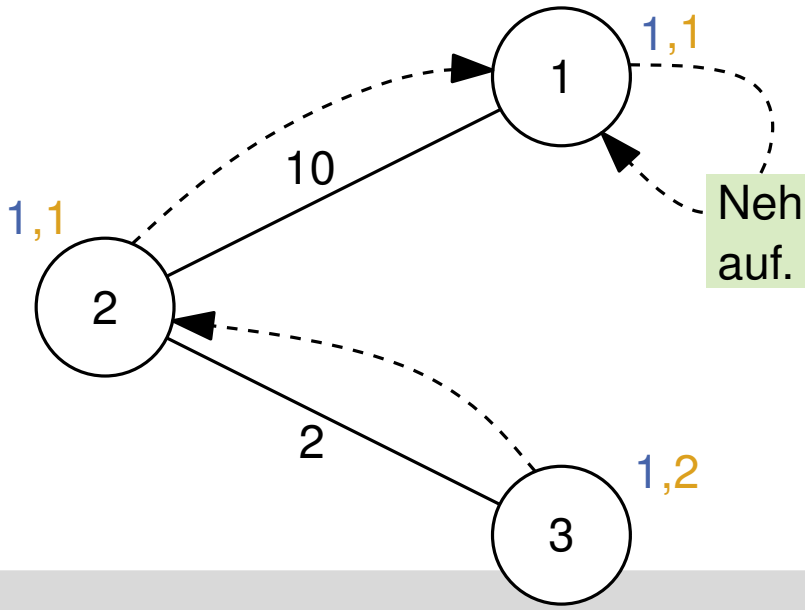
Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

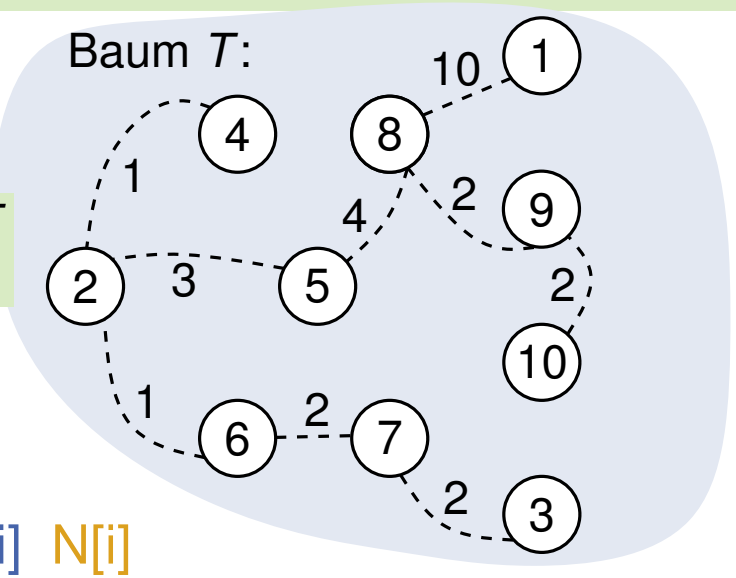
1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.

2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.

3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.



Nehme *Zeiger* in T auf.



Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

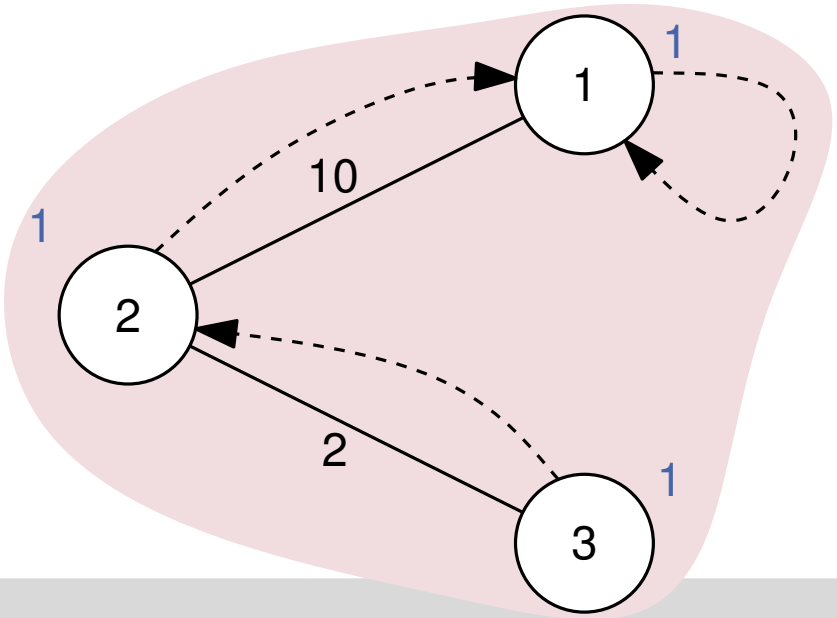
Minimaler Spannbaum

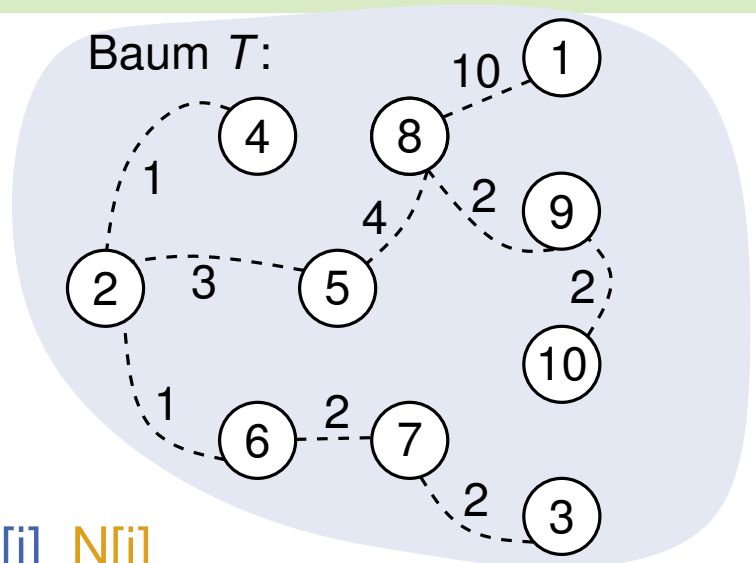
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

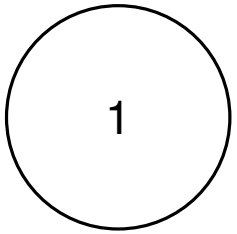
Minimaler Spannbaum

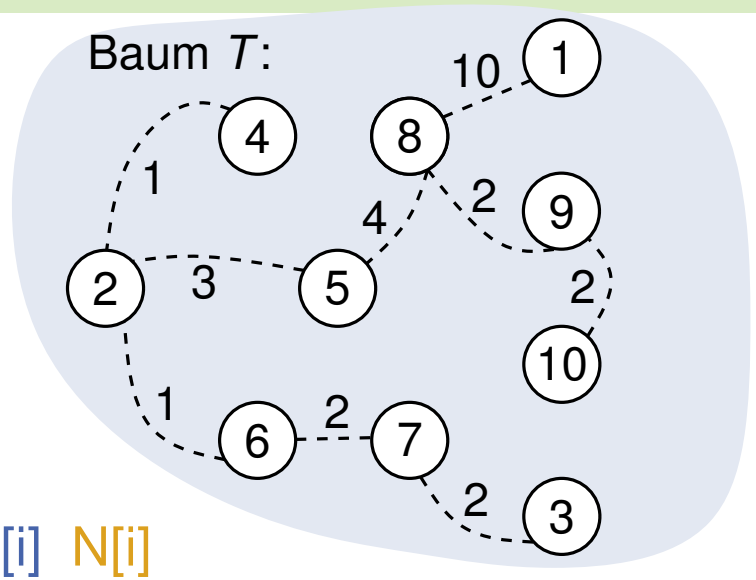
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.





Legende: K[i] N[i]

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Minimaler Spannbaum

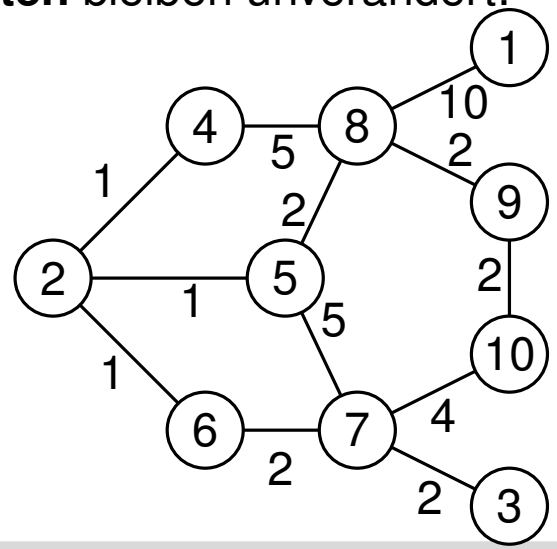
= Führe Schritt parallel aus.

Initialisierung: Weise jedem Knoten parallel eigene Komponente zu.

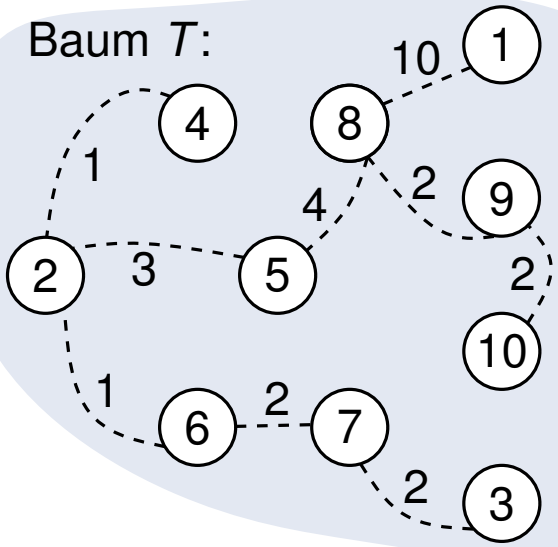
Führe folgende Schritte $\log\lceil n \rceil$-mal aus:

1. Für jeden Knoten $i \in V$ bestimme kleinsten Nachbarn $N[i]$ von i.
2. Für jeden Knoten $i \in V$ setze dessen Komponente $K[i]$ auf $N[i]$.
   - → $K[i]$ ist Zeiger auf i oder Nachbarn von i.
   - → Nach Wahl von $K[i]$ induzieren die $K[i]$'s Wurzelbäume.
3. Verwende List-Ranking, um Wurzelbäume in Superknoten zusammenzufassen.
   - → Superknoten erhält Komponente der Wurzel des entsprechenden Baums.
   - → Arbeite auf Superknoten weiter.

Kosten bleiben unverändert: $O(n^2 \log n)$



Baum T:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Korrektheit

Lemma: Der Algorithmus berechnet einen minimalen Spannbaum auf G.

Beweis: T ist am Ende offensichtlich ein Spannbaum.

Zeige: In jeder Iteration werden nur Kanten zu T hinzugefügt, die zu einem minimalen Spannbaum gehören.

Korrektheit

Lemma: Der Algorithmus berechnet einen minimalen Spannbaum auf G.

Beweis: T ist am Ende offensichtlich ein Spannbaum.

Zeige: In jeder Iteration werden nur Kanten zu T hinzugefügt, die zu einem minimalen Spannbaum gehören.

Betrachte erste Iteration.

Nehme an, dass für Knoten i die Kante (i, j) zu T hinzugefügt wird.

Annahme: (i, j) gehört nicht zu einem minimalen Spannbaum T'.

Korrektheit

Lemma: Der Algorithmus berechnet einen minimalen Spannbaum auf G.

Beweis: T ist am Ende offensichtlich ein Spannbaum.

Zeige: In jeder Iteration werden nur Kanten zu T hinzugefügt, die zu einem minimalen Spannbaum gehören.

Betrachte erste Iteration.

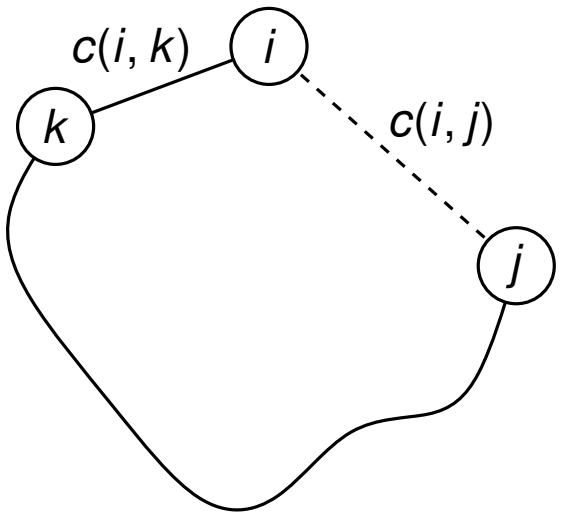
Nehme an, dass für Knoten i die Kante (i, j) zu T hinzugefügt wird.

Annahme: (i, j) gehört nicht zu einem minimalen Spannbaum T'.

Es gibt genau einen Pfad P von i nach j in T'.

P und (i, j) bilden einen Kreis C.

C enthält weitere Kante $\{i, k\}$ mit $i \neq j$.



Korrektheit

Lemma: Der Algorithmus berechnet einen minimalen Spannbaum auf G.

Beweis: T ist am Ende offensichtlich ein Spannbaum.

Zeige: In jeder Iteration werden nur Kanten zu T hinzugefügt, die zu einem minimalen Spannbaum gehören.

Betrachte erste Iteration.

Nehme an, dass für Knoten i die Kante (i, j) zu T hinzugefügt wird.

Annahme: (i, j) gehört nicht zu einem minimalen Spannbaum T'.

Es gibt genau einen Pfad P von i nach j in T'.

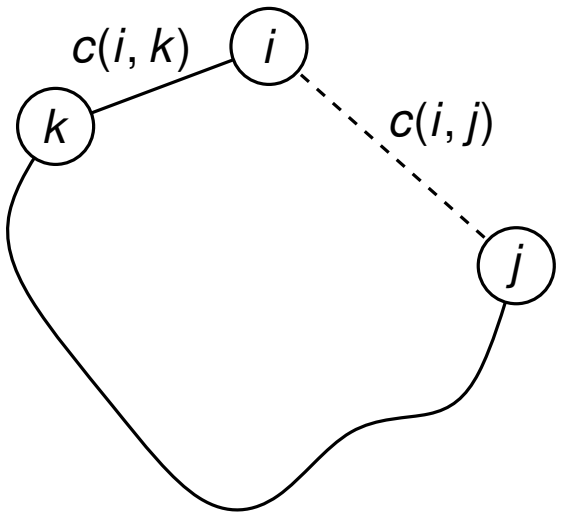
P und (i, j) bilden einen Kreis C.

C enthält weitere Kante $\{i, k\}$ mit $i \neq j$.

Nach Definition gilt $c(i, j) \leq c(i, k)$

$T'' = T' - \{i, k\} + \{i, j\}$ ist minimaler Spannbaum.

Widerspruch zur Annahme.



Laufzeit

Verwende Adjazenzmatrix für Implementierung.

Lemma: Die Gesamtlaufzeit (über alle Phasen) der Neuberechnung der Adjazenzmatrix beträgt

$$O\left(\frac{n}{K} + \log n \cdot \log K\right)$$

Zeit, wenn $n \cdot K$ Prozessoren verwendet werden.

(Ohne Beweis.)

Für $\frac{n^2}{\log^2 n}$ Prozessoren ergibt sich damit eine Gesamtlaufzeit von $O(\log^2 n)$ für die Bestimmung eines MST. Damit betragen die Kosten $O(n^2)$:

- bzgl. sequentiellen Algorithmen für MST, die auf Adjazenzmatrix des Graphen arbeiten, ist dies kostenoptimal.
- bzgl. beliebigen sequentiellen Algorithmen für MST ist dies nicht kostenoptimal.