Algorithmentechnik — Übung 3

http://i11www.ira.uka.de/teaching/WS_0607/algotech

Daniel Delling (delling@ira.uka.de)

WS 0607

Änderung auf Übungsblatt 3

Aufgabe 4:

- Verwenden Sie in Phase i den Knoten als Startknoten, **der Knoten i des Originalgraphen enthält.**

Sonstige Hinweise:

- Aufgabe 1:
  - Aufgabenstellung **genau** lesen (Gewichtsfunktion!)
  - Kontraktion anders als bei Stoer-Wagner
- Aufgabe 3: Gegenbeispiel bei nicht korrekten Algorithmen

Änderung auf Übungsblatt 3

Aufgabe 4:

- Verwenden Sie in Phase *i* den Knoten als Startknoten, **der Knoten *i* des Originalgraphen enthält.**

Sonstige Hinweise:

- Aufgabe 1:
  - Aufgabenstellung **genau** lesen (Gewichtsfunktion!)
  - Kontraktion anders als bei Stoer-Wagner
- Aufgabe 3: Gegenbeispiel bei nicht korrekten Algorithmen

Änderung auf Übungsblatt 3

Aufgabe 4:

- Verwenden Sie in Phase *i* den Knoten als Startknoten, **der Knoten *i* des Originalgraphen enthält.**

Sonstige Hinweise:

- Aufgabe 1:
  - Aufgabenstellung **genau** lesen (Gewichtsfunktion!)
  - Kontraktion anders als bei Stoer-Wagner
- Aufgabe 3: Gegenbeispiel bei nicht korrekten Algorithmen

Allgemeines zu den Übungsblättern

- Ordentlich schreiben!
- Pseudocode!

Allgemeines zu den Übungsblättern

- Ordentlich schreiben!
- **Pseudo**code!

Aufgabe 2

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in lokale Teilnetzgemeinschaften

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in „lokale“ Teilmengen, sodass

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in „lokale“ Teilmengen, sodass
  - Zu Beginn bildet jedes Person eine eigene, einelementige Teilmenge ($V[\ell] = W[\ell]$)

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in „lokale“ Teilmengen, sodass
  - Zu Beginn bildet jedes Person eine eigene, einelementige Teilmenge $C_i, i = 1, \ldots, n$.
  - Verschmelze iterativ immer die beiden Teilmengen mit geringster Entfernung.
  - Abbruch, falls nur noch eine Teilmenge vorhanden ist oder jedes Paar von Teilmengen mehr Abstand als $D_{\max}$ hat.

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in „lokale“ Teilmengen, sodass
  - Zu Beginn bildet jedes Person eine eigene, einelementige Teilmenge $C_i, i = 1, \ldots, n$.
  - Verschmelze iterativ immer die beiden Teilmengen mit geringster Entfernung.
  - Abbruch, falls nur noch eine Teilmenge vorhanden ist oder jedes Paar von Teilmengen mehr Abstand als $D_{\max}$ hat.

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in „lokale“ Teilmengen, sodass
  - Zu Beginn bildet jedes Person eine eigene, einelementige Teilmenge $C_i, i = 1, \ldots, n$.
  - Verschmelze iterativ immer die beiden Teilmengen mit geringster Entfernung.
  - Abbruch, falls nur noch eine Teilmenge vorhanden ist oder jedes Paar von Teilmengen mehr Abstand als $D_{\max}$ hat.

Nachbarschaftstratsch

Gegeben:

- Menge M von n verschiedenen Personen
- symmetrische *Entfernung* $d : M \times M \to \mathbb{R}$.
- Array $D = D[1], \ldots, D[n^2]$ mit Entfernungen aller Personenpaare. **aufsteigend sortiert**
- Arrays V, W mit $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$

Aufgabe:

- Partitioniere Personen in „lokale“ Teilmengen, sodass
  - Zu Beginn bildet jedes Person eine eigene, einelementige Teilmenge $C_i, i = 1, \ldots, n$.
  - Verschmelze iterativ immer die beiden Teilmengen mit geringster Entfernung.
  - Abbruch, falls nur noch eine Teilmenge vorhanden ist oder jedes Paar von Teilmengen mehr Abstand als $D_{\max}$ hat.

Algorithmus 1 : Tratschreichweite

Eingabe : Menge M, Array D, Wert $D_{\max}$, Array V, Array W
Seiteneffekte : Datenstruktur, welche die Partition verwaltet

```
1  n ← |M|
2  Für v ∈ M
3  |  Makeset(v)
4  l ← 1
5  solange l ≤ n² und D[l] ≤ D_max tue
6  |  v ← V[l]
7  |  w ← W[l]
8  |  p ← Find(v)
9  |  q ← Find(w)
10 |  Wenn p ≠ q
11 |  |  Union(p, q)
12 |  l ← l + 1
```

Laufzeitanalyse

im Worst-case

- n Makeset
- $2n^2$ Find
- n Union
- Laufzeit von Find dominiert Makeset und Union

$\Rightarrow O(n^2 \cdot G(n))$

ähnlich zu Klausuraufgabe WS05/06

Laufzeitanalyse

im Worst-case

- n Makeset
- $2n^2$ Find
- n Union
- Laufzeit von Find dominiert Makeset und Union

$\Rightarrow O(n^2 \cdot G(n))$

ähnlich zu Klausuraufgabe WS05/06

Laufzeitanalyse

im Worst-case

- n Makeset
- $2n^2$ Find
- n Union
- Laufzeit von Find dominiert Makeset und Union

$\Rightarrow O(n^2 \cdot G(n))$

ähnlich zu Klausuraufgabe WS05/06

Laufzeitanalyse

im Worst-case

- n Makeset
- $2n^2$ Find
- n Union
- Laufzeit von Find dominiert Makeset und Union

$\Rightarrow$ $O(n^2 \cdot G(n))$

ähnlich zu Klausuraufgabe WS05/06

Laufzeitanalyse

im Worst-case

- n Makeset
- $2n^2$ Find
- n Union
- Laufzeit von Find dominiert Makeset und Union

$\Rightarrow$ $O(n^2 \cdot G(n))$

ähnlich zu Klausuraufgabe WS05/06

Laufzeitanalyse

im Worst-case

- n Makeset
- $2n^2$ Find
- n Union
- Laufzeit von Find dominiert Makeset und Union

$\Rightarrow$ $O(n^2 \cdot G(n))$

ähnlich zu Klausuraufgabe WS05/06

Anmerkungen

Nicht vergessen!

- Makeset(n)
- Find(u) vor einem Union! In Vorlesung anders als im Cormen!

Anmerkungen

Nicht vergessen!

- Makeset(n)
- Find(u) vor einem Union! In Vorlesung anders als im Cormen!

Aufgabe 3

Alternative zu Heaps

Gegeben:

- maximal n Elemente (mit Werten) mit maximal k verschiedenen Schlüsseln

Gesucht:

- Datenstruktur der Größe $O(n)$ mit folgenden Operationen:
  - Insert(Schlüssel, Wert) in $O(1)$

Alternative zu Heaps

Gegeben:

- maximal n Elemente (mit Werten) mit maximal k verschiedenen Schlüsseln

Gesucht:

- Datenstruktur der Größe $O(n)$ mit folgenden Operationen:
  - Insert(Schlüssel, Wert) in $O(1)$
  - FindMay() in $O(k)$
  - Remove(Pointer auf Element in Datenstruktur) in $O(1)$

Alternative zu Heaps

Gegeben:

- maximal n Elemente (mit Werten) mit maximal k verschiedenen Schlüsseln

Gesucht:

- Datenstruktur der Größe $O(n)$ mit folgenden Operationen:
  - Insert(Schlüssel, Wert) in $O(1)$
  - FindMay() in $O(k)$
  - Remove(Pointer auf Element in Datenstruktur) in $O(1)$

Alternative zu Heaps

Gegeben:

- maximal n Elemente (mit Werten) mit maximal k verschiedenen Schlüsseln

Gesucht:

- Datenstruktur der Größe $O(n)$ mit folgenden Operationen:
  - Insert(Schlüssel, Wert) in $O(1)$
  - FindMay() in $O(k)$
  - Remove(Pointer auf Element in Datenstruktur) in $O(1)$

Alternative zu Heaps

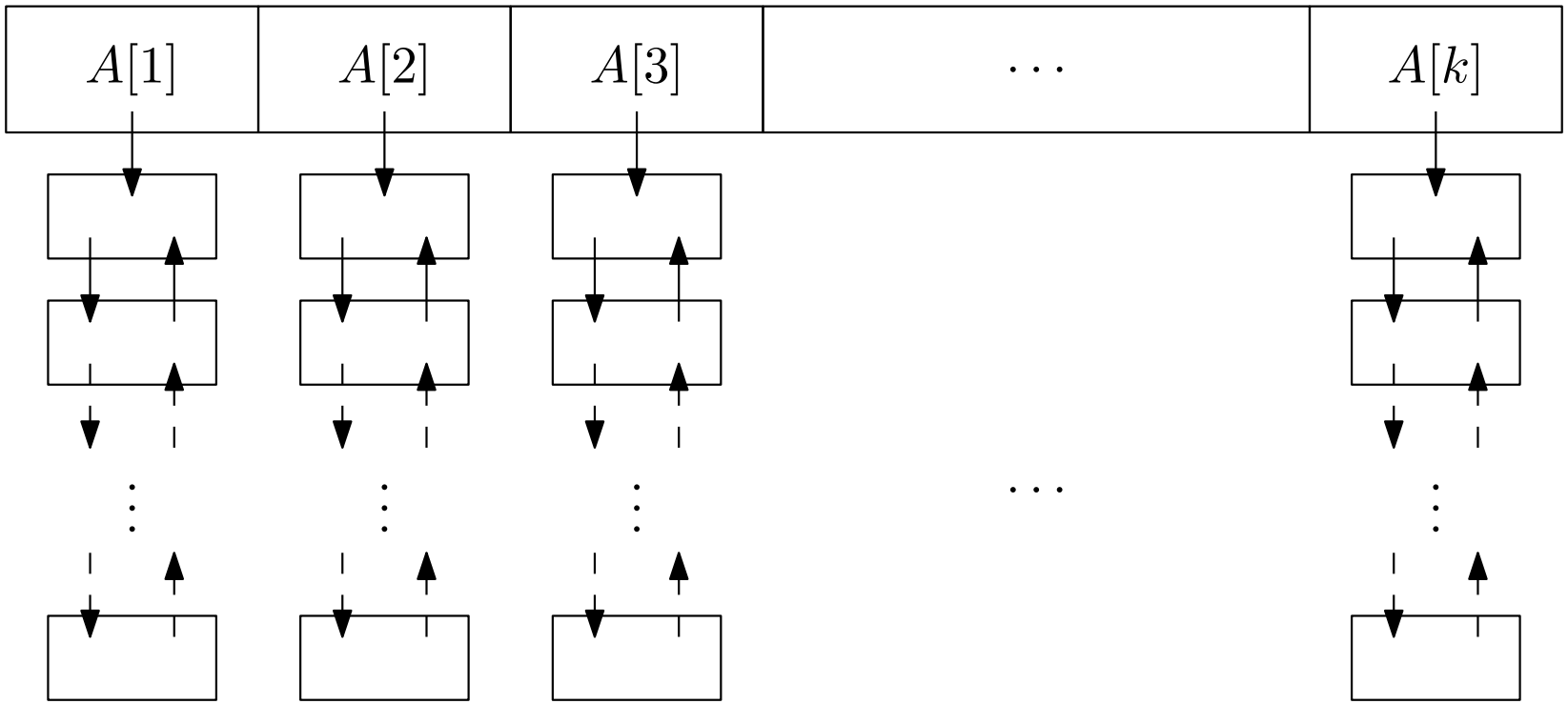
Gegeben:

- maximal n Elemente (mit Werten) mit maximal k verschiedenen Schlüsseln

Gesucht:

- Datenstruktur der Größe $O(n)$ mit folgenden Operationen:
  - Insert(Schlüssel, Wert) in $O(1)$
  - FindMay() in $O(k)$
  - Remove(Pointer auf Element in Datenstruktur) in $O(1)$

Array mit doppelt verketteter Liste



Insert

Algorithmus 2 : Insert(i, v)

Eingabe : Zu speichender Wert v mit Priorität i
Seiteneffekte : Der Wert v befindet sich in der Datenstruktur

1 Sei $A[1, \ldots, k]$ das Array
2 $L \leftarrow A[i]$
3 Add(L, v)

Laufzeit: $O(1)$

Insert

Algorithmus 3 : Insert(i, v)

Eingabe : Zu speichender Wert v mit Priorität i
Seiteneffekte : Der Wert v befindet sich in der Datenstruktur

1 Sei $A[1, \ldots, k]$ das Array
2 $L \leftarrow A[i]$
3 Add(L, v)

Laufzeit: $O(1)$

Remove

Algorithmus 4 : Delete(v)

Eingabe : Zu löschender Wert v
Seiteneffekte : v wird aus der Datenstruktur gelöscht

1 $Nach[Vor[v]] \leftarrow Nach[v]$
2 $Vor[Nach[v]] \leftarrow Vor[v]$

Laufzeit: $O(1)$

Remove

Algorithmus 5 : Delete(v)

Eingabe : Zu löschender Wert v
Seiteneffekte : v wird aus der Datenstruktur gelöscht

1 $Nach[Vor[v]] \leftarrow Nach[v]$
2 $Vor[Nach[v]] \leftarrow Vor[v]$

Laufzeit: $O(1)$

FindMax

Algorithmus 6 : FindMax

Ausgabe : Wert v mit höchster Priorität

```
1 Für i ← k, ..., 1
2     L ← A[i]
3     Wenn L ≠ ∅
4         return First(L)
```

Laufzeit: $O(k)$ (Durchsuchen des Arrays)

FindMax

Algorithmus 7 : FindMax

Ausgabe : Wert v mit höchster Priorität

```
1 Für i ← k, ..., 1
2     L ← A[i]
3     Wenn L ≠ ∅
4         return First(L)
```

Laufzeit: $O(k)$ (Durchsuchen des Arrays)

Aufgabe 4

d-Heaps

Gesucht:

- Verallgemeinerung des bekannten Heaps.
- jeder innere Knoten kann bis zu d Nachfolger haben
- normaler Heap: 2-Heap

d-Heaps

Gesucht:

- Verallgemeinerung des bekannten Heaps.
- jeder innere Knoten kann bis zu d Nachfolger haben
- normaler Heap: 2-Heap

d-Heaps

Gesucht:

- Verallgemeinerung des bekannten Heaps.
- jeder innere Knoten kann bis zu d Nachfolger haben
- normaler Heap: 2-Heap

Baumdarstellung eines 3-Heaps

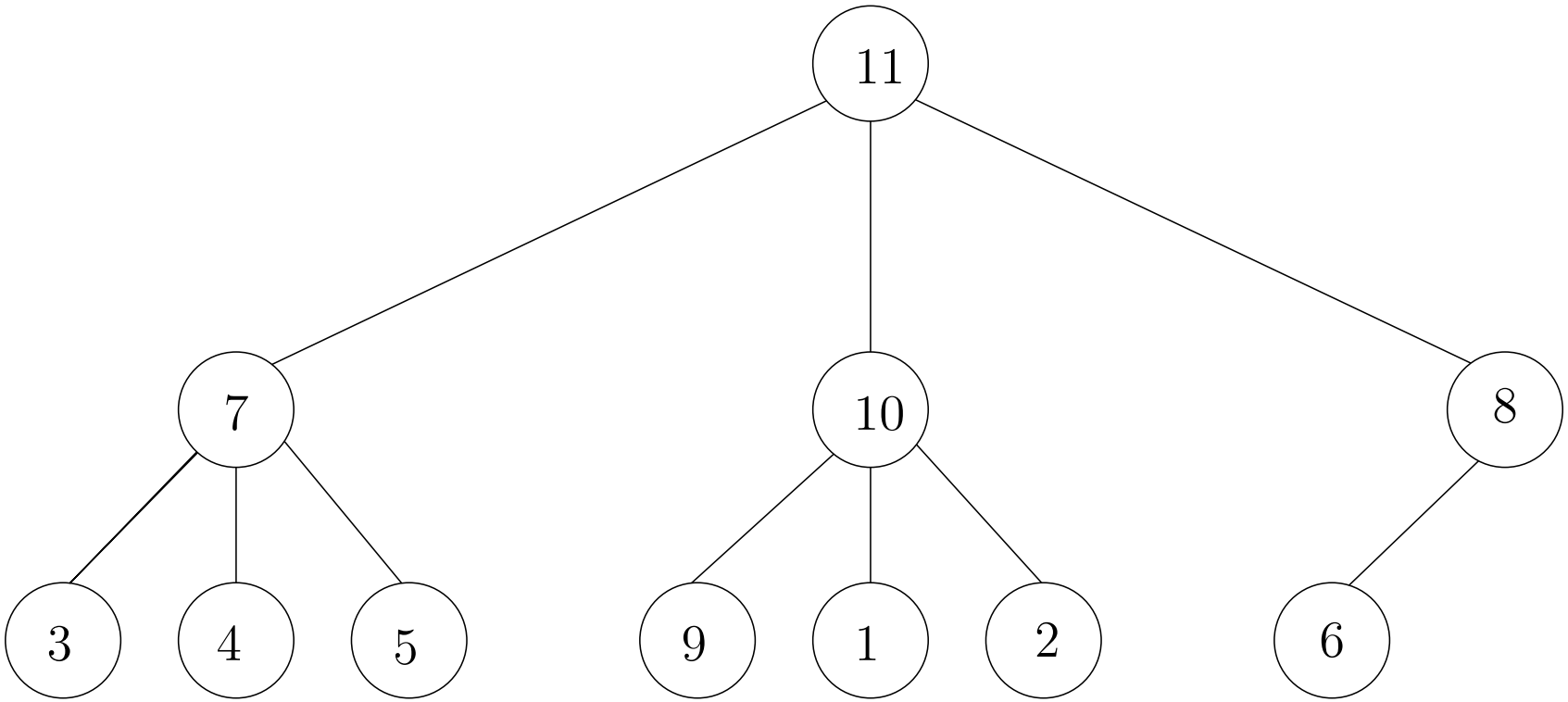


Abbildung: Baumdarstellung eines 3-Heaps

Arraydarstellung eines 3-Heaps

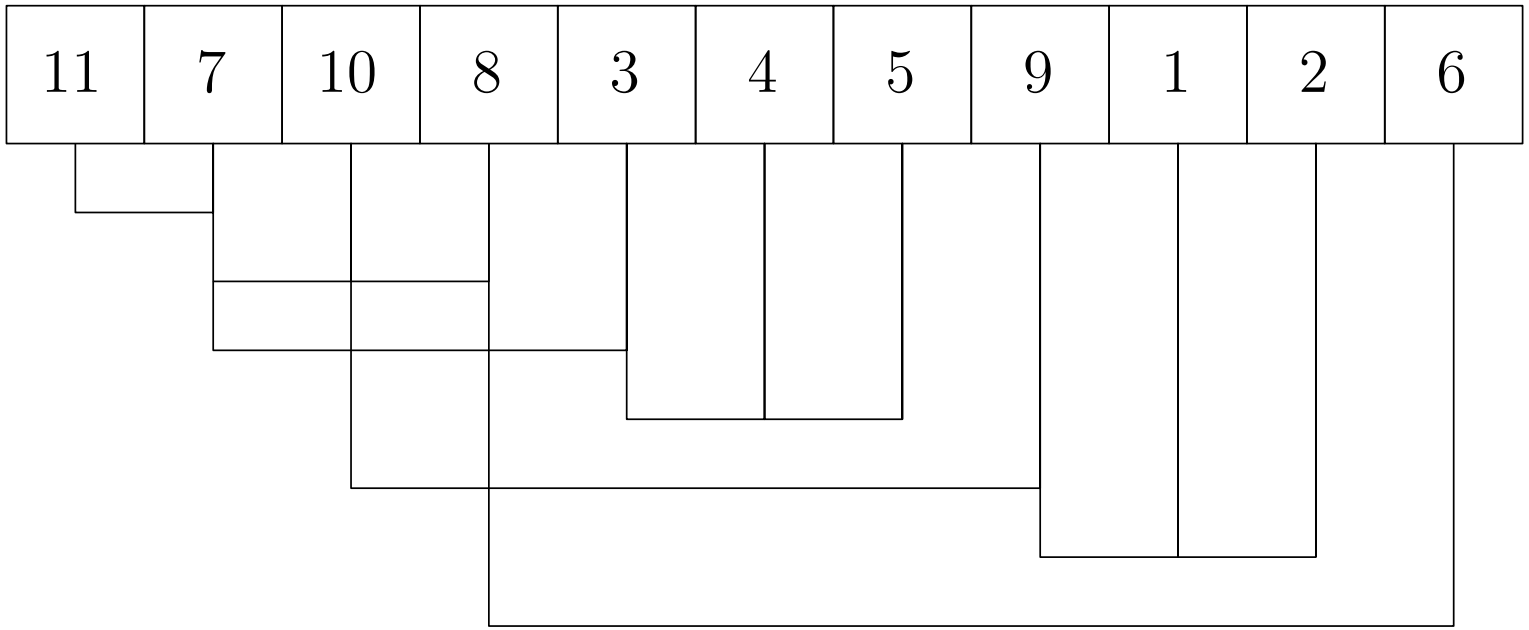


Abbildung: Arraydarstellung eines 3-Heaps

Arraydarstellung, formal

- analog zu 2-Heap
- Indizes des Arrays entsprechen einer Indizierung der Knoten:
  - im Baum von oben (Wurzel) nach unten
  - in jedem Level von links nach rechts

$$\text{d-Succ}(i,j) = d*(i-1)+1+j$$

$$\text{d-Pred}(i) = \left\lceil \frac{i-1}{d} \right\rceil$$

Arraydarstellung, formal

- analog zu 2-Heap
- Indizes des Arrays entsprechen einer Indizierung der Knoten:
  - im Baum von oben (Wurzel) nach unten
  - in jedem Level von links nach rechts

$$\text{d-Succ}(i,j) = d*(i-1)+1+j$$

$$\text{d-Pred}(i) = \left\lceil \frac{i-1}{d} \right\rceil$$

Arraydarstellung, formal

- analog zu 2-Heap
- Indizes des Arrays entsprechen einer Indizierung der Knoten:
  - im Baum von oben (Wurzel) nach unten
  - in jedem Level von links nach rechts

$$\text{d-Succ}(i,j) = d*(i-1)+1+j$$

$$\text{d-Pred}(i) = \left\lceil \frac{i-1}{d} \right\rceil$$

Arraydarstellung, formal

- analog zu 2-Heap
- Indizes des Arrays entsprechen einer Indizierung der Knoten:
  - im Baum von oben (Wurzel) nach unten
  - in jedem Level von links nach rechts

$$\begin{aligned} \text{d-Succ}(i,j) &= d*(i-1)+1+j \\ \text{d-Pred}(i) &= \left\lceil \frac{i-1}{d} \right\rceil \end{aligned}$$

d-Succ(i,j)

Sei ℓ die Tiefe in der Knoten i liegt (wobei $\ell_{\text{Wurzel}} = 0$). Dann gilt:

$$\text{Knoten im vollst. Baum bis Tiefe } \ell - 1 : \sum_{i=0}^{\ell} d^i = \frac{d^\ell - 1}{d - 1}$$

$$\text{Knoten vor } i \text{ innerhalb Tiefe } \ell \ : i - \frac{d^\ell - 1}{d - 1} - 1$$

$$\text{Knoten vor } d\text{−Succ(i, 1) innerhalb Tiefe } \ell + 1 \ : d(i - \frac{d^\ell - 1}{d - 1} - 1)$$

$$\text{Knoten im vollst. Baum bis Tiefe } \ell : \frac{d^{\ell+1} - 1}{d - 1}$$

d-Succ(i,j)

Sei ℓ die Tiefe in der Knoten i liegt (wobei $\ell_{\text{Wurzel}} = 0$). Dann gilt:

$$\text{Knoten im vollst. Baum bis Tiefe } \ell - 1 : \sum_{i=0}^{\ell} d^i = \frac{d^\ell - 1}{d - 1}$$

$$\text{Knoten vor } i \text{ innerhalb Tiefe } \ell \ : i - \frac{d^\ell - 1}{d - 1} - 1$$

$$\text{Knoten vor } d\text{−Succ(i, 1) innerhalb Tiefe } \ell + 1 \ : d(i - \frac{d^\ell - 1}{d - 1} - 1)$$

$$\text{Knoten im vollst. Baum bis Tiefe } \ell : \frac{d^{\ell+1} - 1}{d - 1}$$

d-Succ(i,j)

Sei ℓ die Tiefe in der Knoten i liegt (wobei $\ell_{\text{Wurzel}} = 0$). Dann gilt:

$$\text{Knoten im vollst. Baum bis Tiefe } \ell - 1 : \sum_{i=0}^{\ell} d^i = \frac{d^\ell - 1}{d - 1}$$

$$\text{Knoten vor } i \text{ innerhalb Tiefe } \ell \ : i - \frac{d^\ell - 1}{d - 1} - 1$$

$$\text{Knoten vor } d\text{−Succ(i, 1) innerhalb Tiefe } \ell + 1 \ : d(i - \frac{d^\ell - 1}{d - 1} - 1)$$

$$\text{Knoten im vollst. Baum bis Tiefe } \ell : \frac{d^{\ell+1} - 1}{d - 1}$$

d-Succ(i,j)

Sei ℓ die Tiefe in der Knoten i liegt (wobei $\ell_{\text{Wurzel}} = 0$). Dann gilt:

$$\text{Knoten im vollst. Baum bis Tiefe } \ell - 1 : \sum_{i=0}^{\ell} d^i = \frac{d^\ell - 1}{d - 1}$$

$$\text{Knoten vor } i \text{ innerhalb Tiefe } \ell \ : i - \frac{d^\ell - 1}{d - 1} - 1$$

$$\text{Knoten vor } d\text{−Succ(i, 1) innerhalb Tiefe } \ell + 1 \ : d(i - \frac{d^\ell - 1}{d - 1} - 1)$$

$$\text{Knoten im vollst. Baum bis Tiefe } \ell : \frac{d^{\ell+1} - 1}{d - 1}$$

d-Succ(i,j)

Sei ℓ die Tiefe in der Knoten i liegt (wobei $\ell_{\text{Wurzel}} = 0$). Dann gilt:

$$\text{Knoten im vollst. Baum bis Tiefe } \ell - 1 : \sum_{i=0}^{\ell} d^i = \frac{d^\ell - 1}{d - 1}$$

$$\text{Knoten vor } i \text{ innerhalb Tiefe } \ell \ : i - \frac{d^\ell - 1}{d - 1} - 1$$

$$\text{Knoten vor } d\text{−Succ(i, 1) innerhalb Tiefe } \ell + 1 \ : d(i - \frac{d^\ell - 1}{d - 1} - 1)$$

$$\text{Knoten im vollst. Baum bis Tiefe } \ell : \frac{d^{\ell+1} - 1}{d - 1}$$

d-Succ(i,j)

$$
\begin{aligned}
d\text{−Succ(i, j)} &= d(i - \frac{d^{\ell} - 1}{d-1} - 1) + \frac{d^{\ell+1} - 1}{d-1} + j \\
&= d \cdot i - \frac{d^{\ell+1}}{d-1} + \frac{d}{d-1} - d \cdot 1 + \frac{d^{\ell+1}}{d-1} - \frac{1}{d-1} + j \\
&= d \cdot (i-1) + 1 + j
\end{aligned}
$$

d-Pred(i)

Gesucht: $d-\mathrm{Pred(k)}$ im d-Heap

aus der Formel für $d-\mathrm{Succ(i,j)}$ ergibt sich:

$$k = d \cdot (i-1) + 1 + j$$

$$\Rightarrow \quad i = \frac{k-1-j}{d} + 1$$

$$= \frac{k-1}{d} + 1 - \underbrace{\frac{j}{d}}_{>0 \text{ und } \leq 1} \in \mathbb{N}$$

$$\Rightarrow \quad i + (0;1] = \frac{k-1-j}{d} + 1 \in \mathbb{N}$$

$$\Rightarrow \quad = \left\lceil \frac{k-1}{d} \right\rceil$$

d-Pred(i)

Gesucht: $d-\mathrm{Pred(k)}$ im d-Heap

aus der Formel für $d-\mathrm{Succ(i,j)}$ ergibt sich:

$$k = d \cdot (i-1) + 1 + j$$

$$\Rightarrow \quad i = \frac{k-1-j}{d} + 1$$

$$= \frac{k-1}{d} + 1 - \underbrace{\frac{j}{d}}_{>0 \text{ und } \leq 1} \quad \in \mathbb{N}$$

$$\Rightarrow \quad i + (0;1] = \frac{k-1-j}{d} + 1 \in \mathbb{N}$$

$$\Rightarrow \quad = \left\lceil \frac{k-1}{d} \right\rceil$$

d-Pred(i)

Gesucht: $d-\mathrm{Pred(k)}$ im d-Heap

aus der Formel für $d-\mathrm{Succ(i,j)}$ ergibt sich:

$$k = d \cdot (i-1) + 1 + j$$

$$\Rightarrow \quad i = \frac{k-1-j}{d} + 1$$

$$= \frac{k-1}{d} + 1 - \underbrace{\frac{j}{d}}_{>0 \text{ und } \leq 1} \quad \in \mathbb{N}$$

$$\Rightarrow \quad i + (0;1] = \frac{k-1-j}{d} + 1 \in \mathbb{N}$$

$$\Rightarrow \quad = \left\lceil \frac{k-1}{d} \right\rceil$$

d-Pred(i)

Gesucht: $d-\mathrm{Pred(k)}$ im d-Heap

aus der Formel für $d-\mathrm{Succ(i,j)}$ ergibt sich:

$$k = d \cdot (i-1) + 1 + j$$

$$\Rightarrow \quad i = \frac{k-1-j}{d} + 1$$

$$= \frac{k-1}{d} + 1 - \underbrace{\frac{j}{d}}_{>0 \text{ und } \leq 1} \quad \in \mathbb{N}$$

$$\Rightarrow \quad i + (0;1] = \frac{k-1-j}{d} + 1 \in \mathbb{N}$$

$$\Rightarrow \quad = \left\lceil \frac{k-1}{d} \right\rceil$$

d-Pred(i)

Gesucht: $d-\mathrm{Pred(k)}$ im d-Heap

aus der Formel für $d-\mathrm{Succ(i,j)}$ ergibt sich:

$$k = d \cdot (i-1) + 1 + j$$

$$\Rightarrow \quad i = \frac{k-1-j}{d} + 1$$

$$= \frac{k-1}{d} + 1 - \underbrace{\frac{j}{d}}_{>0 \text{ und } \leq 1} \quad \in \mathbb{N}$$

$$\Rightarrow \quad i + (0;1] = \frac{k-1-j}{d} + 1 \in \mathbb{N}$$

$$\Rightarrow \quad = \left\lceil \frac{k-1}{d} \right\rceil$$

d-Sift-Up(A, i)

analog zu Sift-Up aus der Vorlesung

Algorithmus 8 : d-Sift-Up(A, i)

Eingabe : Baum als Array A, d-Heap-Eigenschaft erfüllt, bis auf evtl. in i

Ausgabe : Das Array A als d-Heap

```
1 ℓ ← i
2 solange d-Pred(ℓ) > 0 und A[ℓ] > A[d-Pred(ℓ)] tue
3     Vertausche A[ℓ] und A[d-Pred(ℓ)]
4     ℓ ← d-Pred(ℓ)
```

Worst-case d-Sift-Up(A,i)

- Wert eines Blattes ändert sich
- tauschen entlang des Pfades zur Wurzel

$\Rightarrow$ $O(log_d\, n)$ Tauschoperationen

Worst-case d-Sift-Up(A,i)

- Wert eines Blattes ändert sich
- tauschen entlang des Pfades zur Wurzel

$\Rightarrow$ $O(log_d n)$ Tauschoperationen

Worst-case d-Sift-Up(A,i)

- Wert eines Blattes ändert sich
- tauschen entlang des Pfades zur Wurzel

$\Rightarrow$ $O(log_d\, n)$ Tauschoperationen

Aufgabe 1

Least Common Ancestor

Algorithmus 9 : LCA(u)

```
1  Makeset(u)
2  Vorfahr[Find(u)] ← u
3  Für jedes Kind v von u in T
4      LCA(v)
5      Union(u,v)
6      Vorfahr[Find(u)] ← u
7  farbe[u] ← SCHWARZ
8  Für jeden Knoten v mit {u, v} ∈ P
9      Wenn farbe[v] = SCHWARZ
10         print 'Der letzte gemeinsame Vorfahr von' u 'und' v 'ist'
           Vorfahr[Find(v)]
```

Teilaufgabe a)

Beh.: Zeile 10 wird nur einmal für jedes Paar aufgerufen.

- rekursive LCA Aufrufe in Preorder auf T

$\Rightarrow$ Schwarzfärbung pro Knoten genau einmal, in Postorder

- Sei $\{u, v\} \in P$. Wenn Zeile 10 in LCA(u) ausgeführt wird:
  - v vor u in Postorder
  - bei LCA(v) ist v noch nicht schwarz gefärbt

$\Rightarrow$ Behauptung

Teilaufgabe a)

Beh.: Zeile 10 wird nur einmal für jedes Paar aufgerufen.

- rekursive LCA Aufrufe in Preorder auf T

$\Rightarrow$ Schwarzfärbung pro Knoten genau einmal, in Postorder

- Sei $\{u, v\} \in P$. Wenn Zeile 10 in LCA(u) ausgeführt wird:
  - v vor u in Postorder
  - $\Rightarrow$ bei LCA(v) ist u noch nicht schwarz gefärbt

$\Rightarrow$ Behauptung

Teilaufgabe a)

Beh.: Zeile 10 wird nur einmal für jedes Paar aufgerufen.

- rekursive LCA Aufrufe in Preorder auf T

$\Rightarrow$ Schwarzfärbung pro Knoten genau einmal, in Postorder

- Sei $\{u, v\} \in P$. Wenn Zeile 10 in LCA(u) ausgeführt wird:
  - v vor u in Postorder

  $\Rightarrow$ bei LCA(v) ist u noch nicht schwarz gefärbt

$\Rightarrow$ Behauptung

Teilaufgabe a)

Beh.: Zeile 10 wird nur einmal für jedes Paar aufgerufen.

- rekursive LCA Aufrufe in Preorder auf T

$\Rightarrow$ Schwarzfärbung pro Knoten genau einmal, in Postorder

- Sei $\{u, v\} \in P$. Wenn Zeile 10 in LCA(u) ausgeführt wird:
  - v vor u in Postorder

  $\Rightarrow$ bei LCA(v) ist u noch nicht schwarz gefärbt

$\Rightarrow$ Behauptung

Teilaufgabe a)

Beh.: Zeile 10 wird nur einmal für jedes Paar aufgerufen.

- rekursive LCA Aufrufe in Preorder auf T

$\Rightarrow$ Schwarzfärbung pro Knoten genau einmal, in Postorder

- Sei $\{u, v\} \in P$. Wenn Zeile 10 in LCA(u) ausgeführt wird:
  - v vor u in Postorder

  $\Rightarrow$ bei LCA(v) ist u noch nicht schwarz gefärbt

$\Rightarrow$ Behauptung

Teilaufgabe a)

Beh.: Zeile 10 wird nur einmal für jedes Paar aufgerufen.

- rekursive LCA Aufrufe in Preorder auf T

$\Rightarrow$ Schwarzfärbung pro Knoten genau einmal, in Postorder

- Sei $\{u, v\} \in P$. Wenn Zeile 10 in LCA(u) ausgeführt wird:
  - v vor u in Postorder

  $\Rightarrow$ bei LCA(v) ist u noch nicht schwarz gefärbt

$\Rightarrow$ Behauptung

Teilaufgabe b)

Hilfsbehauptung:
Nach Ausführung von Zeile 7 in LCA(u) beinhaltet die Menge Find(u) alle Knoten die im Unterbaum $T(u)$ unterhalb von v liegen. Beweis per Induktion über die Höhe $h(v)$ von v.

- I.Anfang: Sei $h(v) = 0 \Rightarrow v$ ist Blatt $\Rightarrow$ Beh. gilt
- I.Annahme: Beh. gelte für alle Knoten u mit Höhe H
- I.Schluss: Sei $h(v) = H + 1$. Für alle Kinder w_i von v gilt:

Teilaufgabe b)

Hilfsbehauptung:
Nach Ausführung von Zeile 7 in LCA(u) beinhaltet die Menge Find(u) alle Knoten die im Unterbaum $T(u)$ unterhalb von v liegen. Beweis per Induktion über die Höhe $h(v)$ von v.

- I.Anfang: Sei $h(v) = 0 \Rightarrow v$ ist Blatt $\Rightarrow$ Beh. gilt
- I.Annahme: Beh. gelte für alle Knoten u mit Höhe H
- I.Schluss: Sei $h(v) = H + 1$. Für alle Kinder w_i von v gilt:
  - Während dem Aufruf LCA(v) werden die Mengen Find(w_i) mit der Menge u vereint. (Zeile 5)
  - am Ende von LCA(v) die Behauptung.

Teilaufgabe b)

Hilfsbehauptung:
Nach Ausführung von Zeile 7 in LCA(u) beinhaltet die Menge Find(u) alle Knoten die im Unterbaum $T(u)$ unterhalb von v liegen. Beweis per Induktion über die Höhe $h(v)$ von v.

- I.Anfang: Sei $h(v) = 0 \Rightarrow v$ ist Blatt $\Rightarrow$ Beh. gilt
- I.Annahme: Beh. gelte für alle Knoten u mit Höhe H
- I.Schluss: Sei $h(v) = H + 1$. Für alle Kinder w_i von v gilt:
  - Während dem Aufruf LCA(v) werden die Mengen Find(w_i) mit der Menge u vereint (Zeile 5)

  $\Rightarrow$ am Ende von LCA(v) die Behauptung

Teilaufgabe b)

Hilfsbehauptung:
Nach Ausführung von Zeile 7 in LCA(u) beinhaltet die Menge Find(u) alle Knoten die im Unterbaum $T(u)$ unterhalb von v liegen. Beweis per Induktion über die Höhe $h(v)$ von v.

- I.Anfang: Sei $h(v) = 0 \Rightarrow v$ ist Blatt $\Rightarrow$ Beh. gilt
- I.Annahme: Beh. gelte für alle Knoten u mit Höhe H
- I.Schluss: Sei $h(v) = H + 1$. Für alle Kinder w_i von v gilt:
  - Während dem Aufruf LCA(v) werden die Mengen Find(w_i) mit der Menge u vereint (Zeile 5)

  $\Rightarrow$ am Ende von LCA(v) die Behauptung

Teilaufgabe b)

Hilfsbehauptung:
Nach Ausführung von Zeile 7 in LCA(u) beinhaltet die Menge Find(u) alle Knoten die im Unterbaum $T(u)$ unterhalb von v liegen. Beweis per Induktion über die Höhe $h(v)$ von v.

- I.Anfang: Sei $h(v) = 0 \Rightarrow v$ ist Blatt $\Rightarrow$ Beh. gilt
- I.Annahme: Beh. gelte für alle Knoten u mit Höhe H
- I.Schluss: Sei $h(v) = H + 1$. Für alle Kinder w_i von v gilt:
  - Während dem Aufruf LCA(v) werden die Mengen Find(w_i) mit der Menge u vereint (Zeile 5)

$\Rightarrow$ am Ende von LCA(v) die Behauptung

Teilaufgabe b)

Hilfsbehauptung:
Nach Ausführung von Zeile 7 in LCA(u) beinhaltet die Menge Find(u) alle Knoten die im Unterbaum $T(u)$ unterhalb von v liegen. Beweis per Induktion über die Höhe $h(v)$ von v.

- I.Anfang: Sei $h(v) = 0 \Rightarrow v$ ist Blatt $\Rightarrow$ Beh. gilt
- I.Annahme: Beh. gelte für alle Knoten u mit Höhe H
- I.Schluss: Sei $h(v) = H + 1$. Für alle Kinder w_i von v gilt:
  - Während dem Aufruf LCA(v) werden die Mengen Find(w_i) mit der Menge u vereint (Zeile 5)

  $\Rightarrow$ am Ende von LCA(v) die Behauptung

Teilaufgabe b)

Behauptung:
Anzahl Mengen in Datenstruktur zur Zeit des Aufrufs LCA(v) ist Tiefe von v in T

- $D(u)$: Anzahl Mengen in Datenstruktur beim Aufruf LCA(u).
- rekursive Aufrufe von LCA(u) erfolgen in Preorder.
- Induktion über diese Folge:
  - I.Anfang: Für LCA($wurzel[T]$) gilt die Behauptung offenbar.
  - I.Annahme: $D(u) =$ Tiefe(u) für alle Knoten u *vor* Knoten v (bzgl. Preorder).

Teilaufgabe b)

Behauptung:
Anzahl Mengen in Datenstruktur zur Zeit des Aufrufs LCA(v) ist Tiefe von v in T

- $D(u)$: Anzahl Mengen in Datenstruktur beim Aufruf LCA(u).
- rekursive Aufrufe von LCA(u) erfolgen in Preorder.
- Induktion über diese Folge:
  - I.Anfang: Für LCA($wurzel[T]$) gilt die Behauptung offenbar.
  - I.Annahme: $D(u) =$ Tiefe(u) für alle Knoten u *vor* Knoten v (bzgl. Preorder).

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen I.Inbehauptung enthält die Menge $\text{Find}(w_i)$ den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird $\text{Find}(w_i)$ mit $\text{Find}(\text{Vater}(v))$ vereint
- $D(v) = D(\text{Vater}(v)) + 1$

Beh. gilt

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen Hilfsbehauptung enthält die Menge Find(w_i) den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird Find(w_i) mit Find(Vater(v)) vereint

$\Rightarrow$ $D(v) = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen Hilfsbehauptung enthält die Menge Find(w_i) den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird Find(w_i) mit Find(Vater(v)) vereint

$\Rightarrow$ $D(v) = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen Hilfsbehauptung enthält die Menge Find(w_i) den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird Find(w_i) mit Find(Vater(v)) vereint

$\Rightarrow$ $D(v) = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen Hilfsbehauptung enthält die Menge Find(w_i) den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird Find(w_i) mit Find(Vater(v)) vereint

$\Rightarrow$ $D(v) = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen Hilfsbehauptung enthält die Menge Find(w_i) den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird Find(w_i) mit Find(Vater(v)) vereint

$\Rightarrow$ $D(v) = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Induktions-Schluss

I.Schluss: Beim Aufruf von LCA(v) gilt
Tiefe(v) = Tiefe(Vater(v)) + 1.
Fallunterscheidung:

Fall 1 Kein Geschwister von v liegt *vor* v:

- $D(v) = D(\text{Vater}(v)) + |\{\{\text{Vater}(v)\}\}| = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Fall 2 Seien $\{w_1, \ldots, w_k\}$ Geschwister von v die *vor* v liegen.

- $\forall w_i$: wegen Hilfsbehauptung enthält die Menge Find(w_i) den Unterbaum von w_i
- in Zeile 5 von LCA(v) wird Find(w_i) mit Find(Vater(v)) vereint

$\Rightarrow$ $D(v) = D(\text{Vater}(v)) + 1$

$\Rightarrow$ Beh. gilt

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

- Vorfahr[Find(w)] ist ein Vorfahr von w, falls definiert.

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

- Vorfahr[Find(w)] ist ein Vofahr von w, falls definiert
- für u und v am Ende von LCA(u) gewährleistet

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

- Vorfahr[Find(w)] ist ein Vofahr von w, falls definiert
- für u und v am Ende von LCA(u) gewährleistet

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

- Vorfahr[Find(w)] ist ein Vofahr von w, falls definiert
- für u und v am Ende von LCA(u) gewährleistet

Korrektheit (Teilaufgabe c)

Beh.: Für jedes Paar $\{u, v\} \in P$ gibt LCA den korrekten letzten gemeinsamen Vorfahren aus

O.B.d.A.: v vor u in Postorder.

- v ist bei Aufruf von Zeile 8 in LCA(u) bereits schwarz.
- Algorithmus gibt Vorfahr[Find(v)] aus
- korrekter LCA?

Beobachtung: Durch Zeile 5,6 gilt für jeden Knoten w:

- Vorfahr[Find(w)] ist ein Vofahr von w, falls definiert
- für u und v am Ende von LCA(u) gewährleistet

Korrektheit (Fall 1)

Fall 1: v sei im Unterbaum von u.

- $v \in \text{Find}(u)$ wegen Beobachtung

$\Rightarrow \text{Find}(v) = \text{Find}(u)$

- Außerdem: $\text{Vorfahr}[\text{Find}(u)] = u$ wegen Zeile 6 in $\text{LCA}(u)$

$\Rightarrow \text{Vorfahr}[\text{Find}(v)] = \text{Vorfahr}[\text{Find}(u)] = u$

Korrektheit (Fall 1)

Fall 1: v sei im Unterbaum von u.

- $v \in \text{Find}(u)$ wegen Beobachtung

$\Rightarrow \text{Find}(v) = \text{Find}(u)$

- Außerdem: $\text{Vorfahr}[\text{Find}(u)] = u$ wegen Zeile 6 in $\text{LCA}(u)$

$\Rightarrow \text{Vorfahr}[\text{Find}(v)] = \text{Vorfahr}[\text{Find}(u)] = u$

Korrektheit (Fall 1)

Fall 1: v sei im Unterbaum von u.

- $v \in \text{Find}(u)$ wegen Beobachtung

$\Rightarrow \text{Find}(v) = \text{Find}(u)$

- Außerdem: $\text{Vorfahr}[\text{Find}(u)] = u$ wegen Zeile 6 in $\text{LCA}(u)$

$\Rightarrow \text{Vorfahr}[\text{Find}(v)] = \text{Vorfahr}[\text{Find}(u)] = u$

Korrektheit (Fall 1)

Fall 1: v sei im Unterbaum von u.

- $v \in \text{Find}(u)$ wegen Beobachtung

$\Rightarrow \text{Find}(v) = \text{Find}(u)$

- Außerdem: $\text{Vorfahr}[\text{Find}(u)] = u$ wegen Zeile 6 in $\text{LCA}(u)$

$\Rightarrow \text{Vorfahr}[\text{Find}(v)] = \text{Vorfahr}[\text{Find}(u)] = u$

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π
  - $\Rightarrow$ x ist kein Vorfahr von v
  - $\Rightarrow$ Widerspruch

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π
  - $\Rightarrow$ x ist kein Vorfahr von v
  - $\Rightarrow$ Widerspruch

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π
  - $\Rightarrow$ x ist kein Vorfahr von v
  - $\Rightarrow$ Widerspruch

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π
  - $\Rightarrow$ x ist kein Vorfahr von v
  - $\Rightarrow$ Widerspruch

Korrektheit (Fall 2)

Fall 2: v sei nicht im Unterbaum von u.

Sei q der korrekte LCA und Π der Pfad von u zur Wurzel, $q \in \Pi$.
Z.z.: Vorfahr[Find(v)] $= q$

- Angenommen Vorfahr[Find(v)] $= x$ ist echter Vorfahr von q
  - $\Rightarrow$ u und v sind im selben Unterbaum von x
  - $\Rightarrow$ wegen Vorfahr[Find(v)] $= x$ wurde in LCA(x) Zeile 5 bereits für den Unterbaum von x, der v und u enthält, durchgeführt
  - $\Rightarrow$ LCA(u) ist schon abgearbeitet
  - $\Rightarrow$ Widerspruch
- Angenommen Vorfahr[Find(v)] $= x$ ist echter Nachkomme von q und Element von Π
  - $\Rightarrow$ x ist kein Vorfahr von v
  - $\Rightarrow$ Widerspruch

Least Common Ancestor

Algorithmus 10 : LCA(u)

```
1  Makeset(u)
2  Vorfahr[Find(u)] ← u
3  Für jedes Kind v von u in T
4      LCA(v)
5      Union(u,v)
6      Vorfahr[Find(u)] ← u
7  farbe[u] ← SCHWARZ
8  Für jeden Knoten v mit {u, v} ∈ P
9      Wenn farbe[v] = SCHWARZ
10         print 'Der letzte gemeinsame Vorfahr von' u 'und' v 'ist'
           Vorfahr[Find(v)]
```

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow O(n^2 G(n))$.

... Es geht auch in $O(n^2)$. Wie?

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow O(n^2 G(n))$.

... Es geht auch in $O(n^2)$. Wie?

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow O(n^2 G(n))$.

... Es geht auch in $O(n^2)$. Wie?

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow O(n^2 G(n))$.

Es geht auch in $O(n^2)$. Wie?

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow O(n^2 G(n))$.

Es geht auch in $O(n^2)$. Wie?

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow O(n^2 G(n))$.

Es geht auch in $O(n^2)$. Wie?

Laufzeitanalyse (Teilaufgabe d)

ganzheitliche amortisierte Analyse über alle Rekursionen

- Zeilen 2 und 7 werden n mal ausgeführt $\Rightarrow O(n)$
- Zeilen 1,5,6 sind $\Theta(n)$ Operation aus Makeset, Union, Find $\Rightarrow O(nG(n))$
- Falls $|P| \in \Omega(n^2)$, werden Zeile 9 und 10 $\Omega(n^2)$ mal ausgeführt $\Rightarrow n^2$ Find Operationen

$\Rightarrow$ Laufzeit also majorisiert von n^2 Operationen vom Typ Makeset, Union, Find.

$\Rightarrow$ $O(n^2 G(n))$.

Es geht auch in $O(n^2)$. Wie?

Stoer-Wagner

Warum reichen $n-1$ Schritte für die Bestimmung eines MinCuts?

Ankündigungen

Termine

- Nächste Vorlesung: Dienstag, 5. Dezember, 15:45 Uhr
- Übungsblatt Abgabe: Mittwoch, 6. Dezember 15:30 Uhr
- Nächstes Übungsblatt: Dienstag, 12. Dezember, 15.45 Uhr
- Nächste Übung: **Dienstag, 17. Dezember**, 15.45 Uhr