Karlsruher Institut für Technologie

Algorithmen I
Tutorium 33

Woche 8 | 15. Juni 2018

Daniel Jungkind (daniel.jungkind@student.kit.edu)

INSTITUT FÜR THEORETISCHE INFORMATIK

Inhalt

Sortierte Folgen

Graphen

Zu Blatt #7

Durchschnitt: 71 % der Punkte

- Heaps liegen **lückenlos** im Speicher!

Schwarzes Brett

Am **Mi, 20.06.** zum Algorithmen-Termin: **Probeklausur!**

- ✚ „Reale“ Klausurbedingungen
- ⇒ Hingehen lohnt sich!
  Die Probeklausur zählt wie zwei Blätter, hat also 20 Punkte.
  Gesamtpunktezahl aller Blätter wird erhöht!
  - Hilfsmittel: **Einseitig beschriebenes Cheatsheet** erlaubt (Tipp: Mit Druckbleistift kann man sehr klein schreiben! ☺)
- ✚ Cheatsheet **auch in der echten Klausur** nutzbar!

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben? **?**

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Ist das Array $(23, 17, 14, 6, 13, 10, 1, 5, 7, 12)$ ein Min-Heap? **?**

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Ist das Array $(23, 17, 14, 6, 13, 10, 1, 5, 7, 12)$ ein Min-Heap? **Nein.**

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Ist das Array $(23, 17, 14, 6, 13, 10, 1, 5, 7, 12)$ ein Min-Heap? **Nein.**

In einem Max-Heap befindet sich das kleinste Element immer im untersten Level. **?**

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Ist das Array $(23, 17, 14, 6, 13, 10, 1, 5, 7, 12)$ ein Min-Heap? **Nein.**

In einem Max-Heap befindet sich das kleinste Element immer im untersten Level. **Falsch.**

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Ist das Array $(23, 17, 14, 6, 13, 10, 1, 5, 7, 12)$ ein Min-Heap? **Nein.**

In einem Max-Heap befindet sich das kleinste Element immer im untersten Level. **Falsch.**

Ist ein aufsteigend sortiertes Array ein Min-Heap? **?**

Quiz

Wie viele Elemente kann ein Heap der Höhe h minimal/maximal haben?

Minimal 2^h, maximal $2^{h+1} - 1$.

Ist das Array $(23, 17, 14, 6, 13, 10, 1, 5, 7, 12)$ ein Min-Heap? **Nein.**

In einem Max-Heap befindet sich das kleinste Element immer im untersten Level. **Falsch.**

Ist ein aufsteigend sortiertes Array ein Min-Heap? **Ja.**

Sortierte Folgen

Die eierlegende Wollmilchdatenstruktur

Sortierte Folgen

Heap- und stichfest?

- **Ziel**: eine **dynamische** und **stets sortierte** Datenstruktur
- Operationen:
  Einfügen,
  Entfernen,
  Finden des nächstkleineren/größeren Elements
  $\Rightarrow$ *so schnell wie möglich*
- **Idee**: Binärer Heap – sieht sortiert aus, ist es aber **nicht**!

Sortierte Folgen

Heap- und stichfest?

- **Ziel**: eine **dynamische** und **stets sortierte** Datenstruktur
- **Operationen**:
  Einfügen,
  Entfernen,
  Finden des nächstkleineren/größeren Elements
  $\Rightarrow$ *so schnell wie möglich*
- **Idee**: Binärer Heap – sieht sortiert aus, ist es aber **nicht**!

Sortierte Folgen

Karlsruher Institut für Technologie

Heap- und stichfest?

- **Ziel**: eine **dynamische** und **stets sortierte** Datenstruktur
- **Operationen**:
  Einfügen,
  Entfernen,
  Finden des nächstkleineren/größeren Elements
  $\Rightarrow$ *so schnell wie möglich*
- **Idee**: Binärer Heap – sieht sortiert aus, ist es aber **nicht**!

Sortierte Folgen

Einfach sortierter Binärbaum

- **Vorschlag**: Binärbaum mit strengerer Ordnung:
  $\forall v \in V:\ \text{LeftChild}(v) \leqslant v < \text{RightChild}(v)$
- Intuitiv: Laufzeiten in $O(\log n)$ mittels **binärer Suche**

$\Longrightarrow$ **Worst-Case**: Füge aufsteigende Folge ein

$\Rightarrow$ Lange Kette entsteht („Baum **unbalanciert**"),
Laufzeiten in $O(n)$ ☹

$\Rightarrow$ I. A. eher ungeeignet

$\Rightarrow$ Wollen **balancierten** Baum (alle Blätter haben gleiche Tiefe)

Sortierte Folgen

Einfach sortierter Binärbaum

- **Vorschlag**: Binärbaum mit strengerer Ordnung:
  $\forall v \in V:\ \text{LeftChild}(v) \leqslant v < \text{RightChild}(v)$
- Intuitiv: Laufzeiten in $O(\log n)$ mittels **binärer Suche**

$\hookrightarrow$ **Worst-Case:** Füge aufsteigende Folge ein

$\Rightarrow$ Lange Kette entsteht („Baum **unbalanciert**"),
Laufzeiten in $O(n)$ ☹

$\Rightarrow$ I. A. eher ungeeignet

$\Rightarrow$ Wollen **balancierten** Baum (alle Blätter haben gleiche Tiefe)

Sortierte Folgen

Einfach sortierter Binärbaum

- **Vorschlag**: Binärbaum mit strengerer Ordnung:
  $\forall v \in V:\ \text{LeftChild}(v) \leqslant v < \text{RightChild}(v)$
- Intuitiv: Laufzeiten in $O(\log n)$ mittels **binärer Suche**
- **Worst-Case**: Füge aufsteigende Folge ein
  $\Rightarrow$ Lange Kette entsteht („Baum **unbalanciert**“),
  Laufzeiten in $O(n)$ ☹
  $\Rightarrow$ I. A. eher ungeeignet
  $\Rightarrow$ Wollen **balancierten** Baum (alle Blätter haben gleiche Tiefe)

Sortierte Folgen

Einfach sortierter Binärbaum

- **Vorschlag**: Binärbaum mit strengerer Ordnung:
  $\forall v \in V : \ \text{LeftChild}(v) \leqslant v < \text{RightChild}(v)$
- Intuitiv: Laufzeiten in $O(\log n)$ mittels **binärer Suche**
- **Worst-Case**: Füge aufsteigende Folge ein
  $\Rightarrow$ Lange Kette entsteht („Baum **unbalanciert**“),
  Laufzeiten in $O(n)$ ☹
  $\Rightarrow$ I. A. eher ungeeignet
  $\Rightarrow$ Wollen **balancierten** Baum (alle Blätter haben gleiche Tiefe)

Sortierte Folgen

Einfach sortierter Binärbaum

- **Vorschlag**: Binärbaum mit strengerer Ordnung:
  $\forall v \in V : \ \text{LeftChild}(v) \leqslant v < \text{RightChild}(v)$
- Intuitiv: Laufzeiten in $O(\log n)$ mittels **binärer Suche**
- **Worst-Case**: Füge aufsteigende Folge ein
  $\Rightarrow$ Lange Kette entsteht („Baum **unbalanciert**“),
  Laufzeiten in $O(n)$ ☹
  $\Rightarrow$ I. A. eher ungeeignet

$\Rightarrow$ Wollen **balancierten** Baum (alle Blätter haben gleiche Tiefe)

Sortierte Folgen

(a, b)-Bäume

- **Besser**: Baum mit **flexiblem** Knotengrad
  $\Rightarrow$ Anzahl **Kinder** zwischen $a \ldots b$
  Ausnahme: Wurzel kann weniger haben
- Dafür sinnvoll: $a \geqslant 2$ und $b \geqslant 2a - 1$
- Jeder **Knoten** hat ein **Navigations-Array**:
  Einträge mit (k : Key, T_k : Subtree):
  T_k führt nur zu Elementen $e \leqslant k$
  Letzter Eintrag: kein Key k, führt zu Elementen $e >$ letztes k
- **Blätter**: Eigentliche Elemente/Daten als **verkettete Liste**
- Zur Vermeidung von Sonderfällen:
  „Dummy-Wert" ∞ ganz am Ende

Sortierte Folgen

(a, b)-Bäume

- **Besser**: Baum mit **flexiblem** Knotengrad
  $\Rightarrow$ Anzahl **Kinder** zwischen $a \ldots b$
  Ausnahme: Wurzel kann weniger haben
- Dafür sinnvoll: $a \geqslant 2$ und $b \geqslant 2a - 1$
- Jeder **Knoten** hat ein **Navigations-Array**:
  Einträge mit (k : Key, T_k : Subtree):
  T_k führt nur zu Elementen $e \leqslant k$
  Letzter Eintrag: kein Key k, führt zu Elementen $e >$ letztes k
- **Blätter**: Eigentliche Elemente/Daten als verkettete Liste
- Zur Vermeidung von Sonderfällen:
  „Dummy-Wert" ∞ ganz am Ende

Sortierte Folgen

(a, b)-Bäume

- **Besser**: Baum mit **flexiblem** Knotengrad
  $\Rightarrow$ Anzahl **Kinder** zwischen $a \ldots b$
  Ausnahme: Wurzel kann weniger haben
- Dafür sinnvoll: $a \geqslant 2$ und $b \geqslant 2a - 1$
- Jeder **Knoten** hat ein **Navigations-Array**:
  Einträge mit (k : Key, T_k : Subtree):
  T_k führt nur zu Elementen $e \leqslant k$
  Letzter Eintrag: kein Key k, führt zu Elementen $e >$ letztes k
- **Blätter**: Eigentliche Elemente/Daten als **verkettete Liste**
- Zur Vermeidung von Sonderfällen:
  „Dummy-Wert" ∞ ganz am Ende

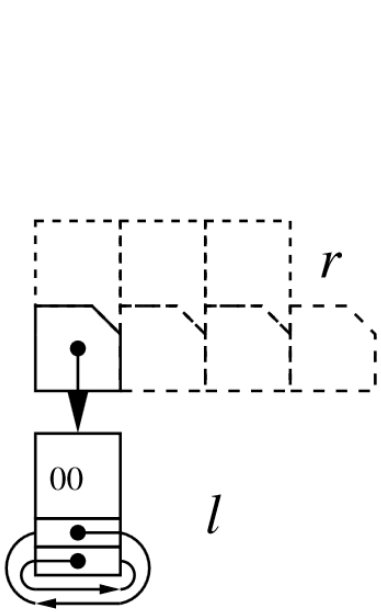
Sortierte Folgen

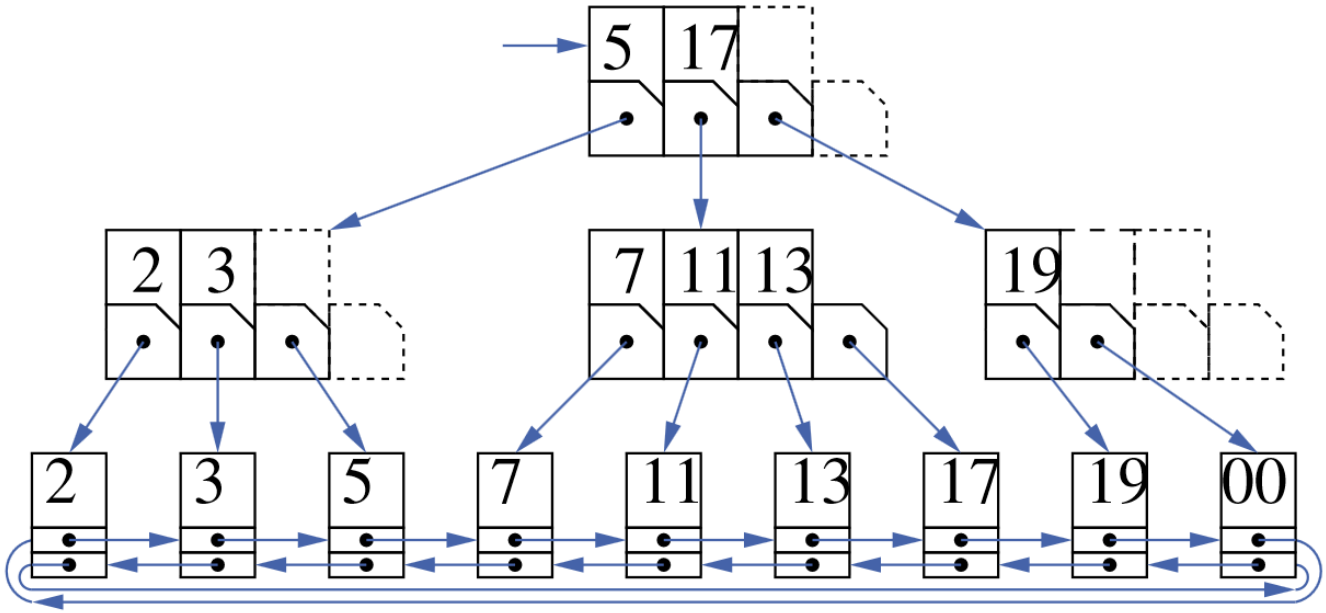
(a, b)-Bäume

- **Besser**: Baum mit **flexiblem** Knotengrad
  $\Rightarrow$ Anzahl **Kinder** zwischen $a \ldots b$
  Ausnahme: Wurzel kann weniger haben
- Dafür sinnvoll: $a \geqslant 2$ und $b \geqslant 2a - 1$
- Jeder **Knoten** hat ein **Navigations-Array**:
  Einträge mit (k : Key, T_k : Subtree):
  T_k führt nur zu Elementen $e \leqslant k$
  Letzter Eintrag: kein Key k, führt zu Elementen $e >$ letztes k
- **Blätter**: Eigentliche Elemente/Daten als **verkettete Liste**
- Zur Vermeidung von Sonderfällen:
  „Dummy-Wert“ ∞ ganz am Ende

Sortierte Folgen – (a, b)-Bäume

Beispiel: (2, 4)-Baum („00“ steht in VL für ∞)





Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei Wurzel
Suche Element j im Navigationsarray, wobei
$j := \min\{j \mid e \leqslant j\}$
Blatt-Ebene erreicht? $\Rightarrow$ return j
Sonst Wiederhole auf Subtree von j oder ganz rechtem Link falls $\nexists\, j$

- Laufzeit in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei **Wurzel**

Suche Element j im Navigationsarray, wobei
$j := \min\{j \mid e \leqslant j\}$
Blatt-Ebene erreicht? $\Rightarrow$ return j
Sonst Wiederhole auf Subtree von j oder ganz rechtem Link falls $\nexists\, j$

- Laufzeit in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei **Wurzel**
Suche Element j im Navigationsarray, wobei
$j := \min \{ j \mid e \leqslant j \}$
Blatt-Ebene erreicht? $\Rightarrow$ return j
Sonst Wiederhole auf Subtree von j oder ganz rechtem Link falls $\nexists\, j$

- Laufzeit in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei **Wurzel**
Suche Element j im Navigationsarray, wobei
$j := \min \{ j \mid e \leqslant j \}$
Blatt-Ebene erreicht? $\Rightarrow$ **return** j
Sonst **Wiederhole** auf Subtree von j oder ganz rechtem Link falls $\nexists j$

- **Laufzeit** in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei **Wurzel**
Suche Element j im Navigationsarray, wobei
$j := \min \{ j \mid e \leqslant j \}$
Blatt-Ebene erreicht? $\Rightarrow$ **return** j
Sonst **Wiederhole** auf Subtree von j oder ganz rechtem Link falls $\nexists j$

- **Laufzeit** in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei **Wurzel**
Suche Element j im Navigationsarray, wobei
$j := \min \{ j \mid e \leqslant j \}$
Blatt-Ebene erreicht? $\Rightarrow$ **return** j
Sonst **Wiederhole** auf Subtree von j oder ganz rechtem Link falls $\nexists\, j$

- **Laufzeit** in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

- **Geg.**: Wert e
  Ges.: (Nächstgrößeres) Element $z \geqslant e$

$\Rightarrow$ Starte bei **Wurzel**
Suche Element j im Navigationsarray, wobei
$j := \min \{ j \mid e \leqslant j \}$
Blatt-Ebene erreicht? $\Rightarrow$ **return** j
Sonst **Wiederhole** auf Subtree von j oder ganz rechtem Link falls $\nexists j$

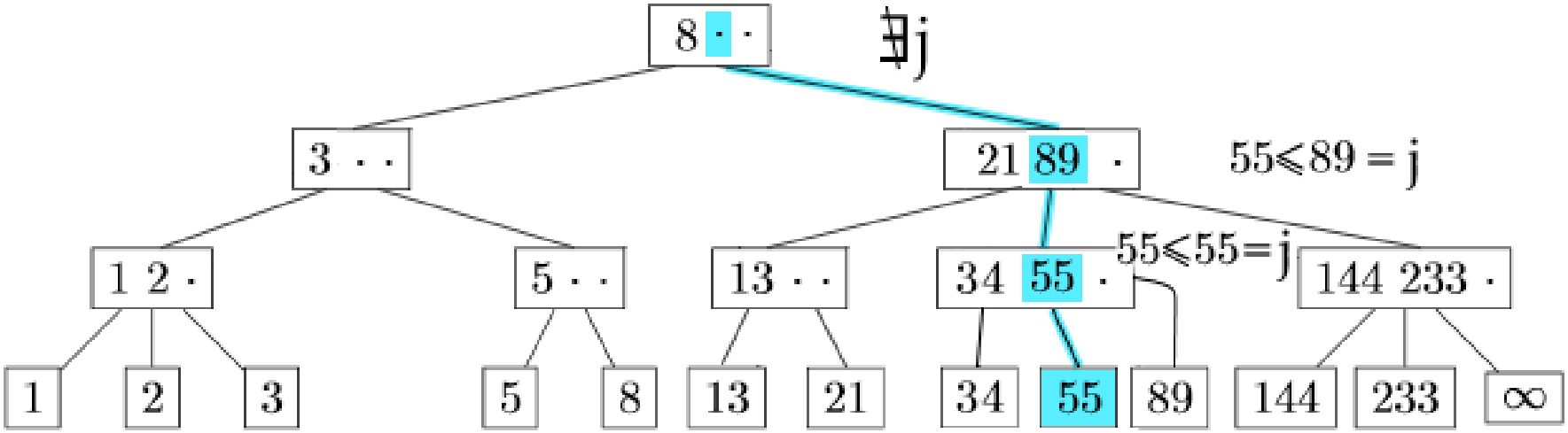
- **Laufzeit** in $O(b \cdot \text{Höhe}) = O(b \cdot \log_a n)$

- Finden von *nächstkleinerem* Element:
  Finde nächstgrößeres;
  Falls $j \neq e$: Nehme Vorgänger von j in verketteter Liste

Sortierte Folgen – (a, b)-Bäume

Finden von (nächstgrößeren (-kleineren)) Elementen

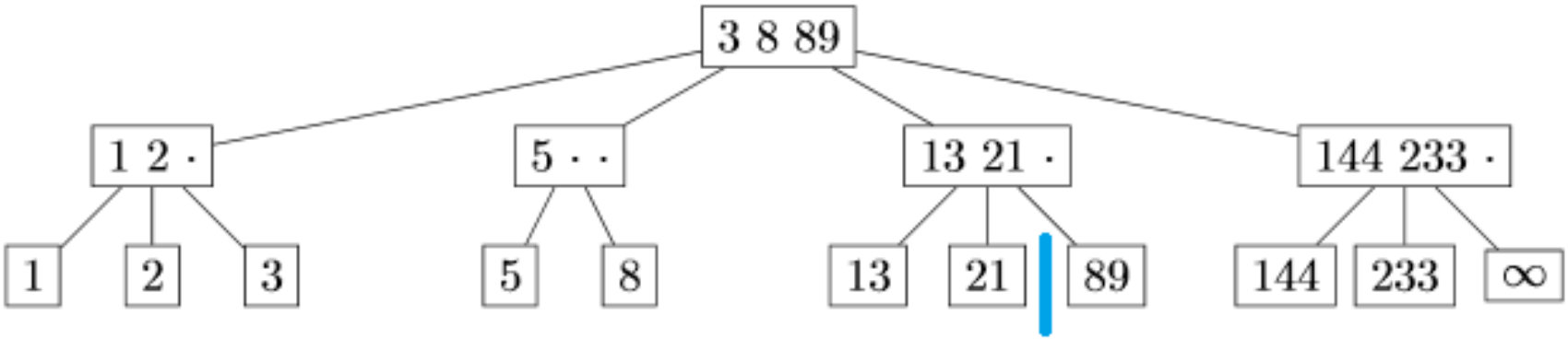
Beispiel: Suche nach 55:



Sortierte Folgen – (a, b)-Bäume

Einfügen von Elementen

1. Finde Einfügestelle (wie beim Suchen)

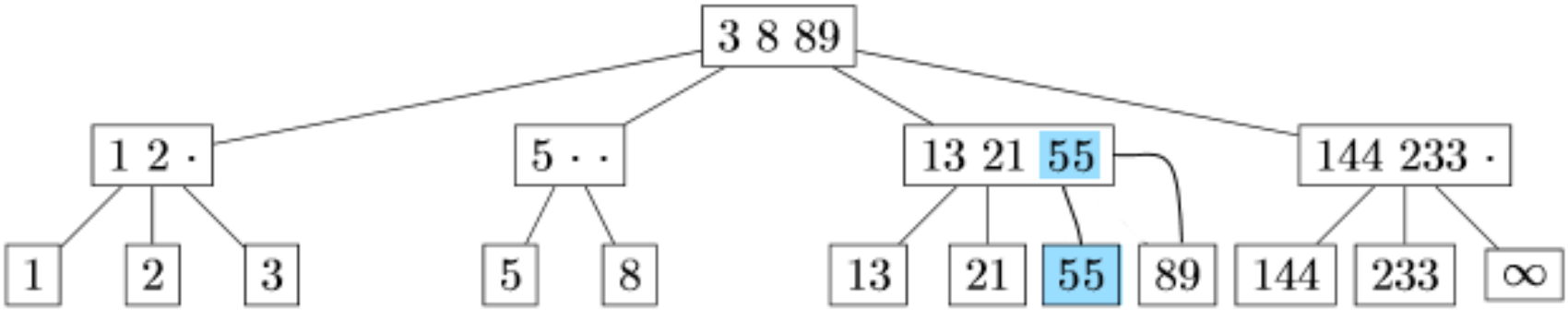


Sortierte Folgen – (a, b)-Bäume

Einfügen von Elementen

1. Finde Einfügestelle (wie beim Suchen)

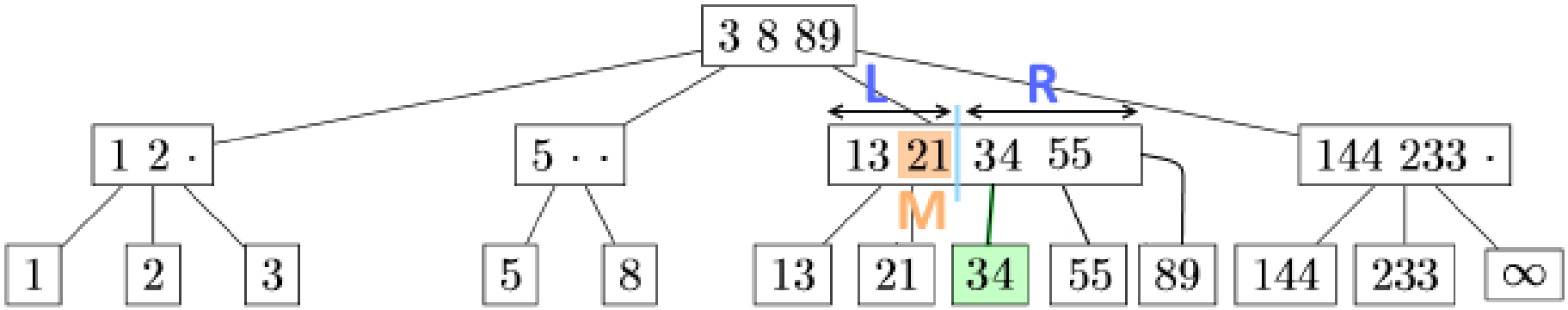
2a. **Fall 1**: Platz im Navigationsarray frei?
   $\Rightarrow$ Einfügen, im Nav-Array verlinken, fertig! ☺
   (falls neues Maximum: **Verlinkung anpassen**!)



Sortierte Folgen – (a, b)-Bäume

Einfügen von Elementen (Forts.)

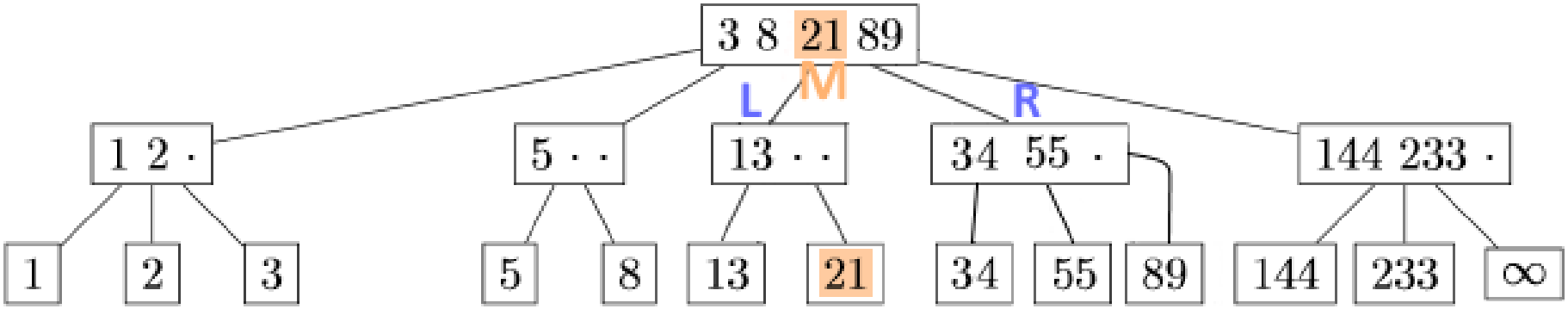
2b. **Fall 2**: Kein Platz im Nav-Array frei? $\Rightarrow$ *„split“*
D.h. Element einfügen, Knoten **halbieren**:
Linker Teil *L* (enthält **Mittelelement** *M*), **Rechter** Teil *R*



Sortierte Folgen – (a, b)-Bäume

Einfügen von Elementen (Forts.)

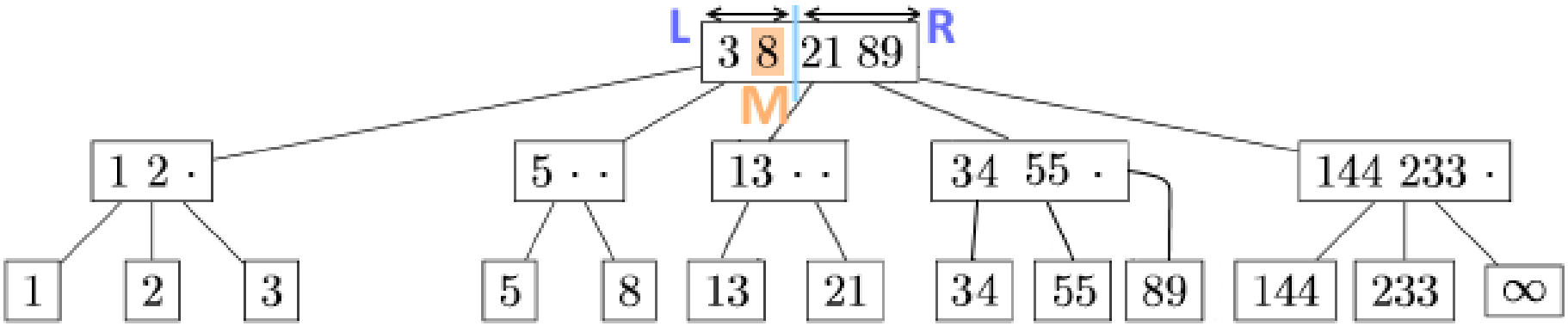
3. Füge *M* in Vorgänger ein, hänge *L* als Subtree daran; *R* hängt schon im Vorgänger



Sortierte Folgen – (a, b)-Bäume

Einfügen von Elementen (Forts.)

3. Füge *M* in Vorgänger ein, hänge *L* als Subtree daran; *R* hängt schon im Vorgänger

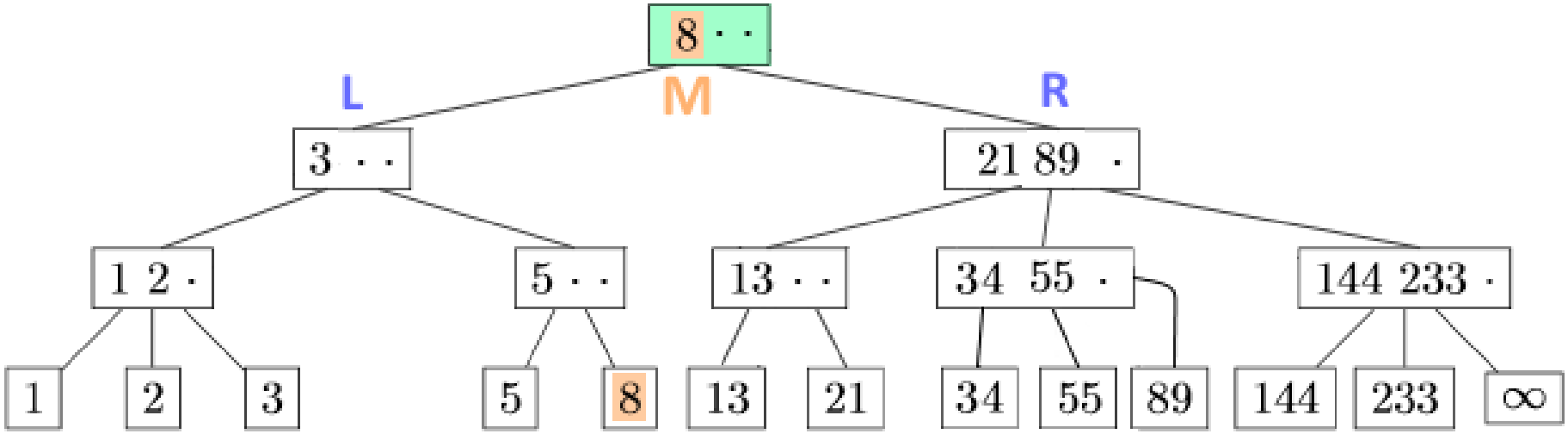


4. Vorgänger **voll**? $\Rightarrow$ **Recurse** from step 2b.

Sortierte Folgen – (a, b)-Bäume

Einfügen von Elementen (Forts.)

3. Füge *M* in Vorgänger ein, hänge *L* als Subtree daran; *R* hängt schon im Vorgänger

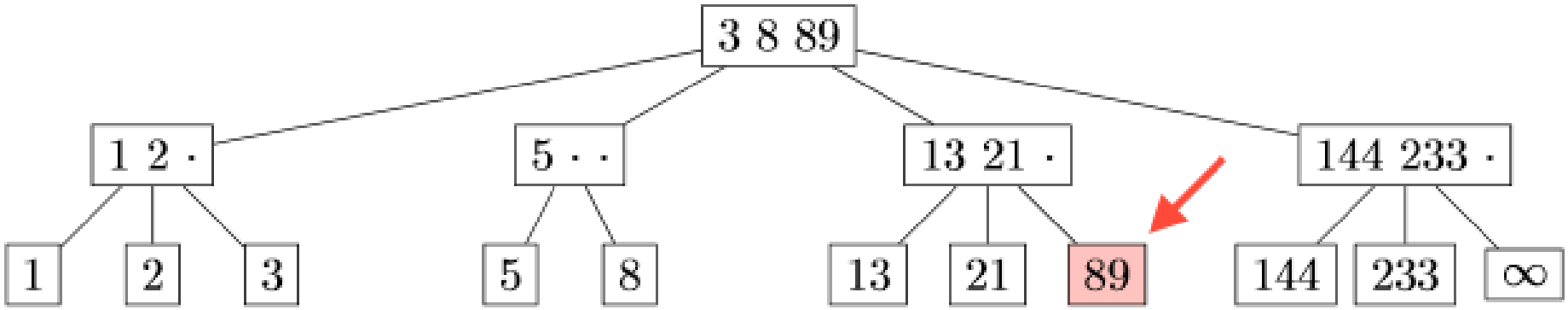


4. Vorgänger **voll**? $\Rightarrow$ **Recurse** from step 2b.
   Endet ggf. mit Anlegen einer *neuen Wurzel*

Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

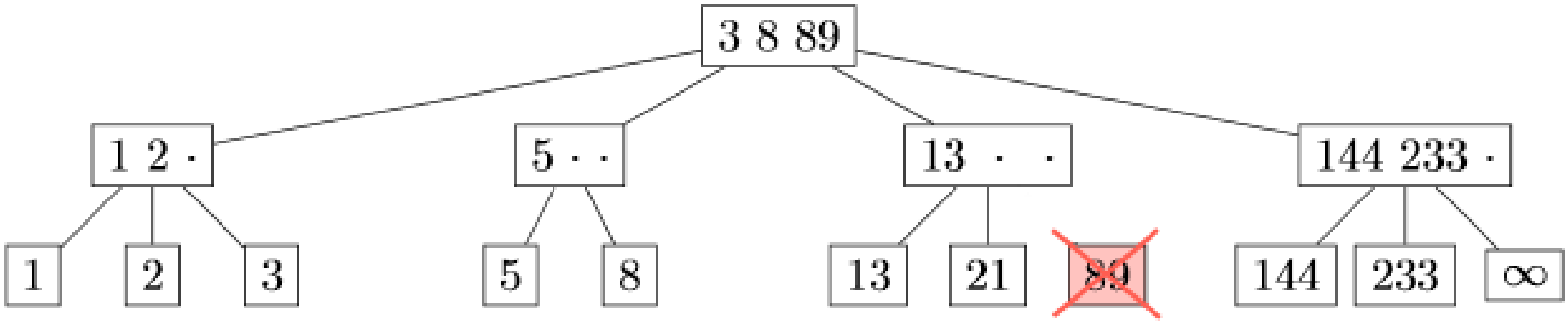
1. Einfach: Finden



Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

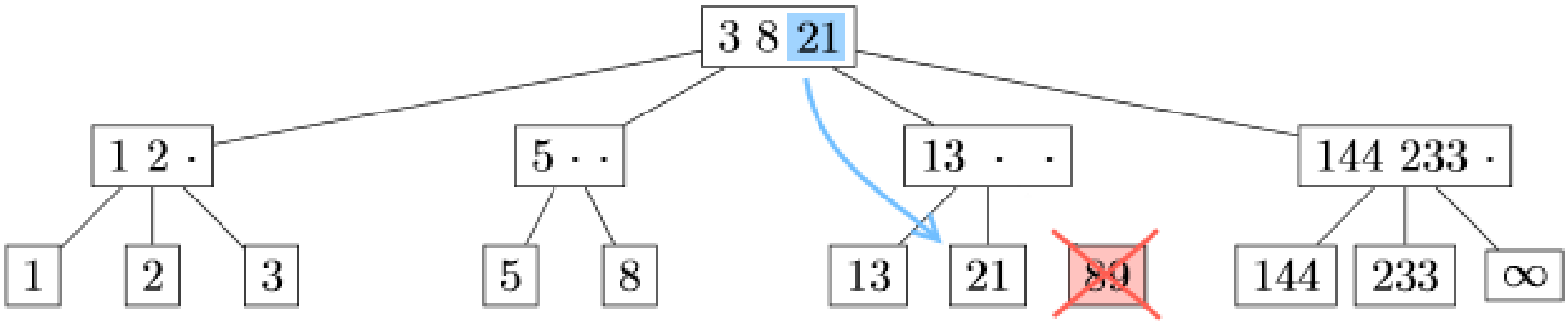
1. Einfach: Finden und Entfernen.



Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

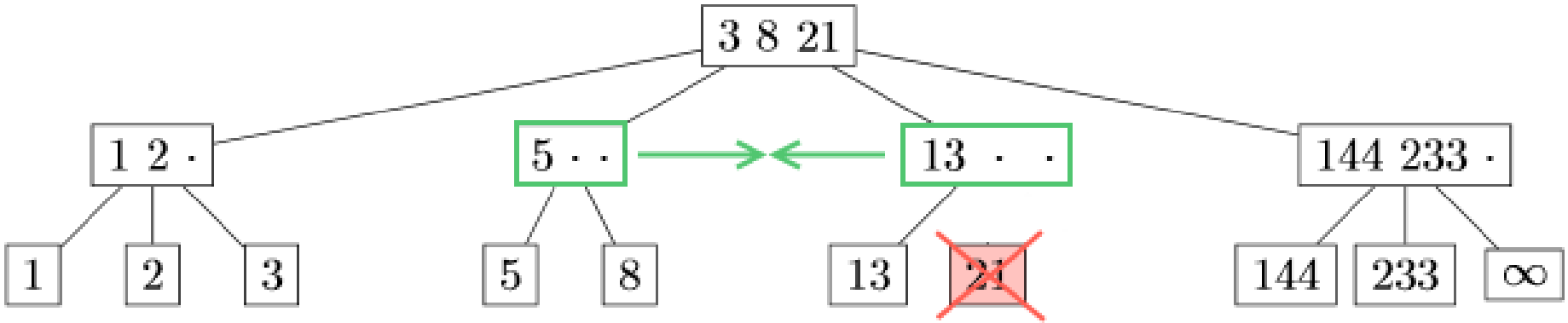
1. Einfach: Finden und Entfernen.
   Knotenmaximum wurde entfernt?
   $\Rightarrow$ **Aktualisiere Verlinkung auf neues Maximum!**



Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

2. Knoten jetzt **zu klein?**
   2a. **Fall 1**: ...und ∃ Nachbar, der leer genug?
   ⇒ *„fuse“*: Knoten zusammenfügen

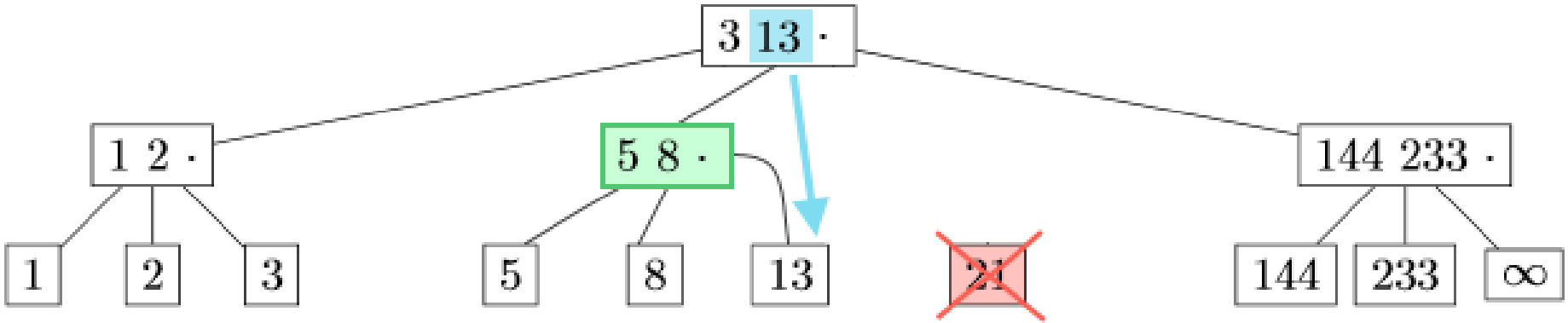


Vorgänger jetzt zu klein? ⇒ Recurse from step 2.

Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

2. Knoten jetzt **zu klein?**
   2a. **Fall 1**: ...und ∃ Nachbar, der leer genug?
   ⇒ *„fuse“*: Knoten zusammenfügen
   ...und **Verlinkung anpassen!**

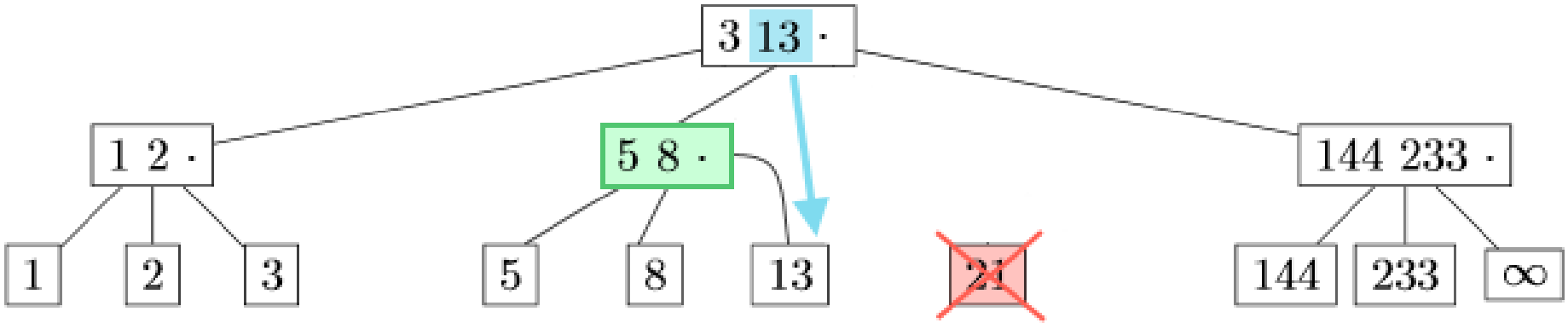


Vorgänger jetzt zu klein? ⇒ Recurse from step 2.

Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

2. Knoten jetzt **zu klein?**
   2a. **Fall 1**: ...und ∃ Nachbar, der leer genug?
   ⇒ *„fuse“*: Knoten zusammenfügen
   ...und **Verlinkung anpassen!**



Vorgänger jetzt **zu klein**? ⇒ **Recurse** from step 2.

Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

2. Knoten jetzt **zu klein?**
   2b. **Fall 2**: Ansonsten: $\exists$ Nachbar, der voll genug
   $\Rightarrow$ „*balance*“: Klaue Elemente vom fetten Nachbarn
   (von links: maximale, von rechts: minimale Elemente)
   ...und **Verlinkung anpassen!**

Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

2. Knoten jetzt **zu klein?**

   2b. **Fall 2**: Ansonsten: $\exists$ Nachbar, der voll genug

   $\Rightarrow$ *„balance“*: Klaue Elemente vom fetten Nachbarn

   (von links: maximale, von rechts: minimale Elemente)

   ...und Verlinkung anpassen!

Sortierte Folgen – (a, b)-Bäume

Entfernen von Elementen

2. Knoten jetzt **zu klein?**

   2b. **Fall 2**: Ansonsten: $\exists$ Nachbar, der voll genug

   $\Rightarrow$ *„balance“*: Klaue Elemente vom fetten Nachbarn

   (von links: maximale, von rechts: minimale Elemente)

   ...und **Verlinkung anpassen!**

Sortierte Folgen – (a, b)-Bäume

Laufzeiten:

$$\left.\begin{array}{l} \textit{locate} \\ \textit{insert} \\ \textit{remove} \end{array}\right\} \text{in } O(b \cdot \text{Höhe}) = O(b \cdot \log_a n) \quad \text{(für konst. } a, b\text{: } O(\log n)\text{).}$$

GRAPHEN

Der Plural, nicht der Kohlenstoff

Graphen

Wir erinnern uns ...

- Graph $G = (V, E)$ mit **Knoten**menge $V \neq \varnothing$ und **Kanten**menge E
  - *Gerichteter* Graph: $E \subseteq V \times V$
  - *Ungerichteter* Graph: $E \subseteq \left\{ \{u, v\} \;\middle|\; u, v \in V \right\}$
  - Umwandlung *ungerichtet* $\rightsquigarrow$ *gerichtet* trivial
    $\Rightarrow$ Im Folgenden stets *gerichtete* Graphen
  - $n := |V|$
  - $m := |E|$
  - Betrachten üblicherweise $V = \{1 \ldots n\}$

Graphen

Wir erinnern uns ...

- Graph $G = (V, E)$ mit **Knoten**menge $V \neq \varnothing$ und **Kanten**menge E
- *Gerichteter* Graph: $E \subseteq V \times V$
- *Ungerichteter* Graph: $E \subseteq \left\{ \{u, v\} \;\middle|\; u, v \in V \right\}$
- Umwandlung *ungerichtet* $\rightsquigarrow$ *gerichtet* trivial
  $\Rightarrow$ Im Folgenden stets *gerichtete* Graphen
- $n := |V|$
- $m := |E|$
- Betrachten üblicherweise $V = \{1 \ldots n\}$

Graphen

Wir erinnern uns ...

- Graph $G = (V, E)$ mit **Knoten**menge $V \neq \varnothing$ und **Kanten**menge E
- *Gerichteter* Graph: $E \subseteq V \times V$
- *Ungerichteter* Graph: $E \subseteq \left\{ \{u, v\} \;\middle|\; u, v \in V \right\}$
- Umwandlung *ungerichtet* $\rightsquigarrow$ *gerichtet* trivial
  $\Rightarrow$ Im Folgenden stets *gerichtete* Graphen
- $n := |V|$
- $m := |E|$
- Betrachten üblicherweise $V = \{1 \ldots n\}$

Graphen

Wir erinnern uns ...

- Graph $G = (V, E)$ mit **Knoten**menge $V \neq \varnothing$ und **Kanten**menge E
- *Gerichteter* Graph: $E \subseteq V \times V$
- *Ungerichteter* Graph: $E \subseteq \left\{ \{u, v\} \;\middle|\; u, v \in V \right\}$
- Umwandlung *ungerichtet* $\rightsquigarrow$ *gerichtet* trivial
  $\Rightarrow$ Im Folgenden stets *gerichtete* Graphen
- $n := |V|$
- $m := |E|$
- Betrachten üblicherweise $V = \{1 \ldots n\}$

Graphen

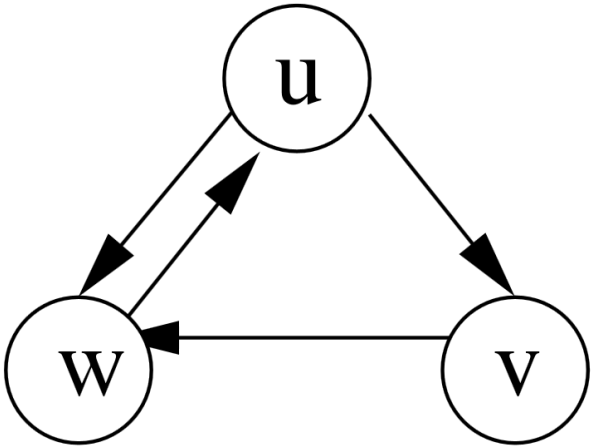
Wir erinnern uns ...

- Graph $G = (V, E)$ mit **Knoten**menge $V \neq \varnothing$ und **Kanten**menge E
- *Gerichteter* Graph: $E \subseteq V \times V$
- *Ungerichteter* Graph: $E \subseteq \left\{ \{u, v\} \;\middle|\; u, v \in V \right\}$
- Umwandlung *ungerichtet* $\rightsquigarrow$ *gerichtet* trivial
  $\Rightarrow$ Im Folgenden stets *gerichtete* Graphen
- $n := |V|$
- $m := |E|$
- Betrachten üblicherweise $V = \{1 \ldots n\}$

Repräsentationen von Graphen

Kantenfolge

- (Zusammenhängender) Graph eindeutig definiert durch **Menge aller Kanten** (Reihenfolge egal)
  - Knoten v existiert in G
    $\Leftrightarrow \exists\ (v, x)$ oder $(x, v) \in$ *Kantenliste* (x beliebig)
  - **Kompakt** $\Rightarrow$ Gut handhabbar (im Speicher oder bei I/O)
  - Einzige effiziente Operation: **Durchlaufen** aller Kanten



$$\Leftrightarrow \langle (u,v),(v,w),(w,u),(u,w) \rangle$$

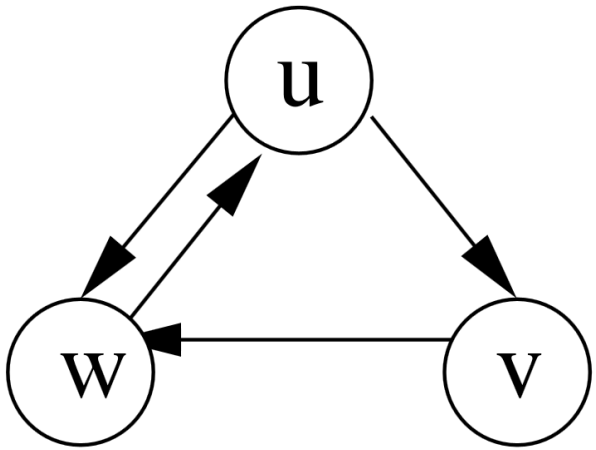
Repräsentationen von Graphen

Kantenfolge

- (Zusammenhängender) Graph eindeutig definiert durch **Menge aller Kanten** (Reihenfolge egal)
- Knoten v existiert in G
  $\Leftrightarrow \exists\ (v, x)$ oder $(x, v) \in$ *Kantenliste* (x beliebig)

Kompakt $\Rightarrow$ Gut handhabbar (im Speicher oder bei I/O)

$\Rightarrow$ Einzige effiziente Operation: **Durchlaufen** aller Kanten



$$\Leftrightarrow \langle (u, v), (v, w), (w, u), (u, w) \rangle$$

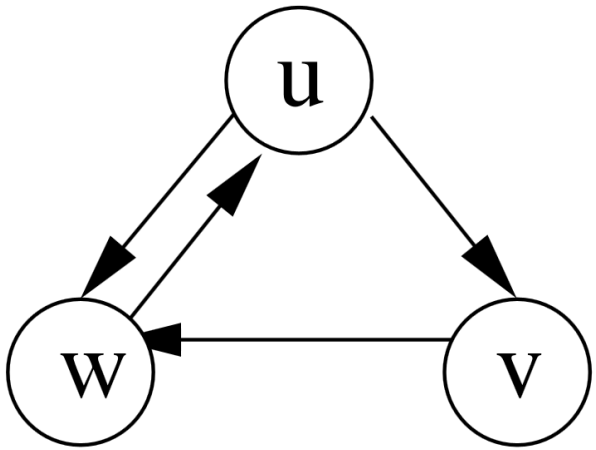
Repräsentationen von Graphen

Kantenfolge

- (Zusammenhängender) Graph eindeutig definiert durch **Menge aller Kanten** (Reihenfolge egal)
- Knoten v existiert in G
  $\Leftrightarrow \exists\,(v, x)$ oder $(x, v) \in$ *Kantenliste* (x beliebig)

✚ **Kompakt** $\Rightarrow$ Gut handhabbar (im Speicher oder bei I/O)

⟹ Einzige effiziente Operation: **Durchlaufen** aller Kanten



$$\Leftrightarrow \langle (u, v), (v, w), (w, u), (u, w) \rangle$$

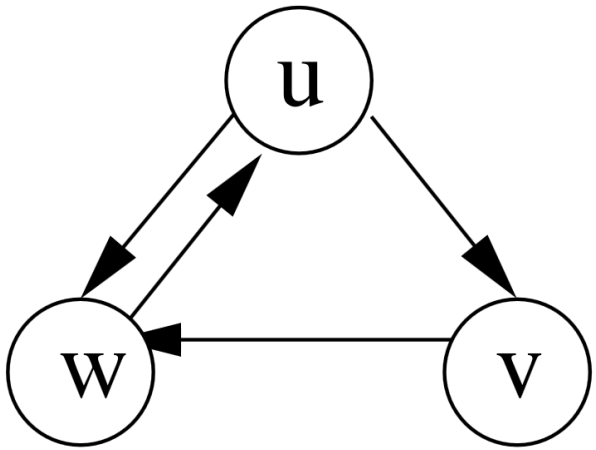
Repräsentationen von Graphen

Kantenfolge

- (Zusammenhängender) Graph eindeutig definiert durch **Menge aller Kanten** (Reihenfolge egal)
- Knoten v existiert in G
  $\Leftrightarrow \exists\,(v,x)$ oder $(x,v) \in$ *Kantenliste* (x beliebig)

✚ **Kompakt** $\Rightarrow$ Gut handhabbar (im Speicher oder bei I/O)

▬ Einzige effiziente Operation: **Durchlaufen** aller Kanten

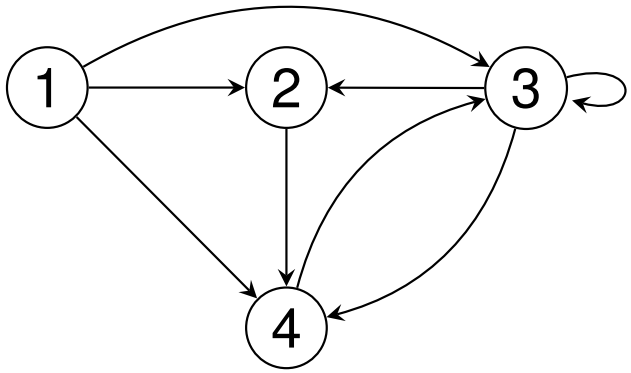


$$\Leftrightarrow \langle (u,v),(v,w),(w,u),(u,w) \rangle$$

Repräsentationen von Graphen

Adjazenzmatrix

Verwende Matrix $A \in \{0,1\}^{n \times n}$ mit $a_{ij} = 1 \Leftrightarrow (i,j) \in E$



$$A = \begin{array}{c} \\ \text{Von } 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} \text{Nach } 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D
- ➖ platzineffizient bei dünnbesetzten Graphen
  (also durchschn. Knotengrad $\ll n$)
- ➖ **langsame** Navigation
- ➖ LA **kotz**

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D

- ▭ platzineffizient bei dünnbesetzten Graphen
  (also durchschn. Knotengrad $\ll n$)
- ▭ **langsame** Navigation
- ▭ LA **kotz**

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D
- ➖ platzineffizient bei dünnbesetzten Graphen (also durchschn. Knotengrad $\ll n$)
- ➖ langsame Navigation
- ➖ LA *"kotz"*

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D

- ➖ platzineffizient bei dünnbesetzten Graphen
  (also durchschn. Knotengrad $\ll n$)
- ➖ langsame Navigation
- ➖ LA *"kotz"*

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D

- ▬ platz**in**effizient bei dünnbesetzten Graphen
  (also durchschn. Knotengrad $\ll n$)
- ▬ langsame Navigation
- ▬ LA *"kotz"*

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D

- ▬ platz**in**effizient bei dünnbesetzten Graphen
  (also durchschn. Knotengrad $\ll n$)
- ▬ **langsame** Navigation
- ▬ LA **kotz**

Repräsentationen von Graphen

Adjazenzmatrix

- ✚ recht **platzeffizient**, falls Graph dicht
- ✚ **Einfügen**, **Löschen** und **Testen** von Kanten in $O(1)$ und simpel
- ✚ $\{0, 1\} \rightsquigarrow \mathbb{R}$ erweitern für **Kantengewichte**
- ✚ LA, yay! :D

- ▬ platz**in**effizient bei dünnbesetzten Graphen
  (also durchschn. Knotengrad $\ll n$)
- ▬ **langsame** Navigation
- ▬ LA **kotz**

Repräsentationen von Graphen

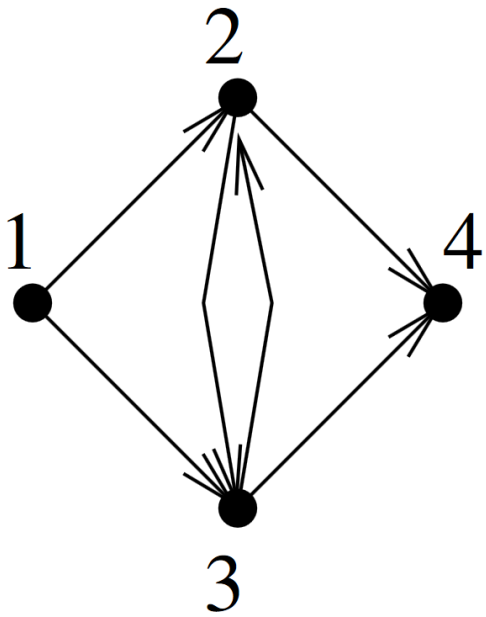
Adjazenzfeld (aka Adjazenzarray)

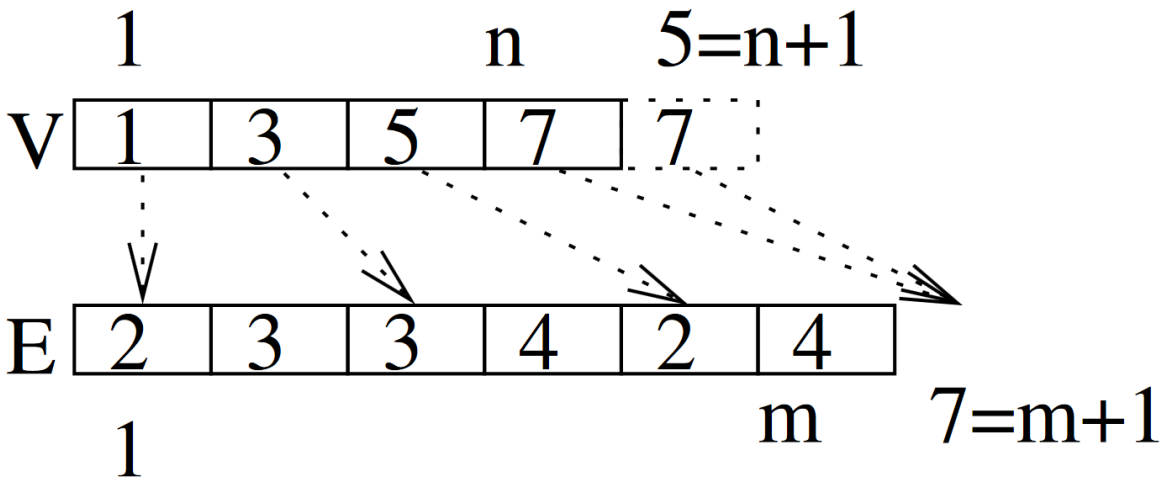
- Definiere V : **array**$[1 \ldots n+1]$ **of** $\{1 \ldots m+1\}$ und
  E : **array**$[1 \ldots m]$ **of** $\{1 \ldots n\}$
- Von v erreichbare Knoten: $\{\, E[i] \mid V[v] \leqslant i < V[v+1] \,\}$
- „*Dummy-Eintrag*“:
  $V[n+1] := m+1$, damit oben $v == n$ nicht knallt

Repräsentationen von Graphen

Adjazenzfeld (aka Adjazenzarray)

- Definiere V : **array**$[1 \ldots n+1]$ **of** $\{1 \ldots m+1\}$ und
  E : **array**$[1 \ldots m]$ **of** $\{1 \ldots n\}$
- Von v erreichbare Knoten: $\{\, E[i] \mid V[v] \leqslant i < V[v+1] \,\}$
- „Dummy-Eintrag“:
  $V[n+1] := m+1$, damit oben $v == n$ nicht knallt

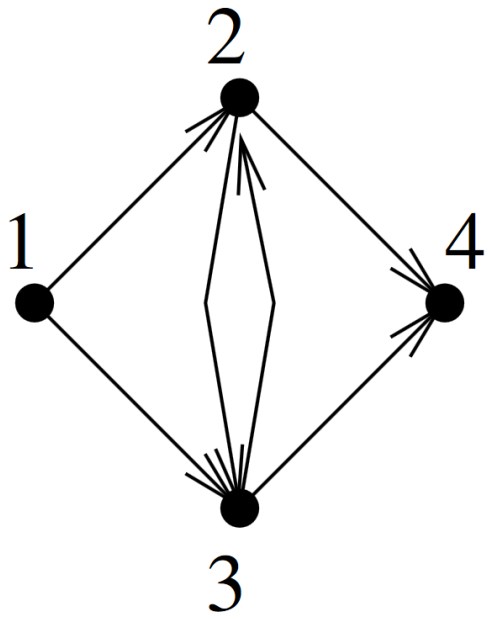


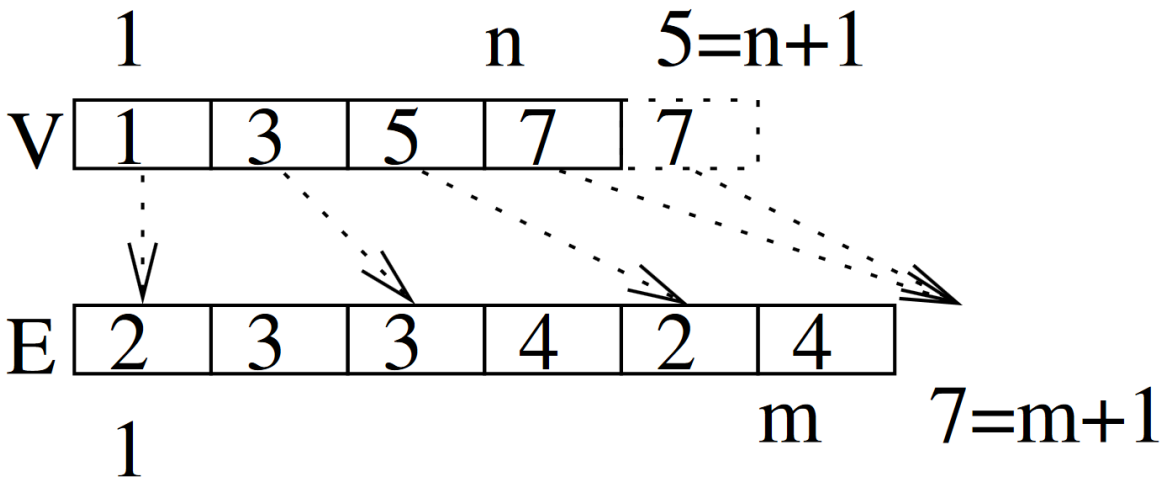


Repräsentationen von Graphen

Adjazenzfeld (aka Adjazenzarray)

- Definiere V : **array**$[1 \ldots n+1]$ **of** $\{1 \ldots m+1\}$ und
  E : **array**$[1 \ldots m]$ **of** $\{1 \ldots n\}$
- Von v erreichbare Knoten: $\{\, E[i] \mid V[v] \leqslant i < V[v+1] \,\}$
- *„Dummy-Eintrag“*:
  $V[n+1] := m+1$, damit oben $v = n$ nicht knallt





Repräsentationen von Graphen

Adjazenzfeld (aka Adjazenzarray)

- ✚ **Navigation** gut möglich
- ✚ **Zusatzinfos** (z. B. Kantengewichte) durch weitere Arrays leicht aufrüstbar
- ✚ **Cachefreundlich**
- ✚ Nachrüstbar: Kanten **löschen**, **rückwärts** laufen
- ➖ Hinzufügen von Kanten scheiße (#ArraysHalt...)

Repräsentationen von Graphen

Adjazenzfeld (aka Adjazenzarray)

- ✚ **Navigation** gut möglich
- ✚ **Zusatzinfos** (z. B. Kantengewichte) durch weitere Arrays leicht aufrüstbar
- ✚ **Cachefreundlich**
- ✚ Nachrüstbar: Kanten **löschen**, **rückwärts** laufen
- ➖ Hinzufügen von Kanten scheiße (#ArraysHalt...)

Repräsentationen von Graphen

Adjazenzfeld (aka Adjazenzarray)

- ✚ **Navigation** gut möglich
- ✚ **Zusatzinfos** (z. B. Kantengewichte) durch weitere Arrays leicht aufrüstbar
- ✚ **Cachefreundlich**
- ✚ Nachrüstbar: Kanten löschen, **rückwärts** laufen
- ➖ Hinzufügen von Kanten scheiße (#ArraysHalt...)

Repräsentationen von Graphen

Adjazenzfeld (aka Adjazenzarray)

- ✚ **Navigation** gut möglich
- ✚ **Zusatzinfos** (z. B. Kantengewichte) durch weitere Arrays leicht aufrüstbar
- ✚ **Cachefreundlich**
- ✚ Nachrüstbar: Kanten **löschen**, **rückwärts** laufen
- ═ Hinzufügen von Kanten scheiße (#ArraysHalt...)

Repräsentationen von Graphen

Karlsruher Institut für Technologie

Adjazenzfeld (aka Adjazenzarray)

- ✚ **Navigation** gut möglich
- ✚ **Zusatzinfos** (z. B. Kantengewichte) durch weitere Arrays leicht aufrüstbar
- ✚ **Cachefreundlich**
- ✚ Nachrüstbar: Kanten **löschen**, **rückwärts** laufen
- ➖ Hinzufügen von Kanten scheiße (#ArraysHalt...)

Repräsentationen von Graphen

Adjazenzliste

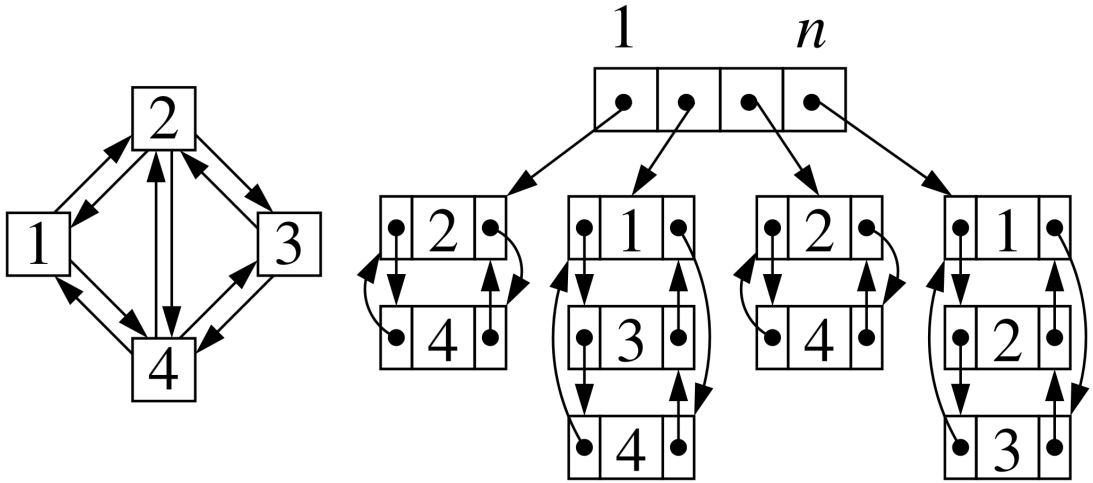
- Verwende **array** $A[1 \ldots n]$ von verketteten Listen
  - $A[v]$: Liste aller von $v \in V$ aus erreichbaren Knoten

⇒ Alle Features vom Adjazenzfeld

⇒ ...und noch mehr: **Einfügen, Löschen** von Kanten

⇒ Benötigt mehr **Platz** (für Zeiger)

⇒ **Cachefeindlicher**

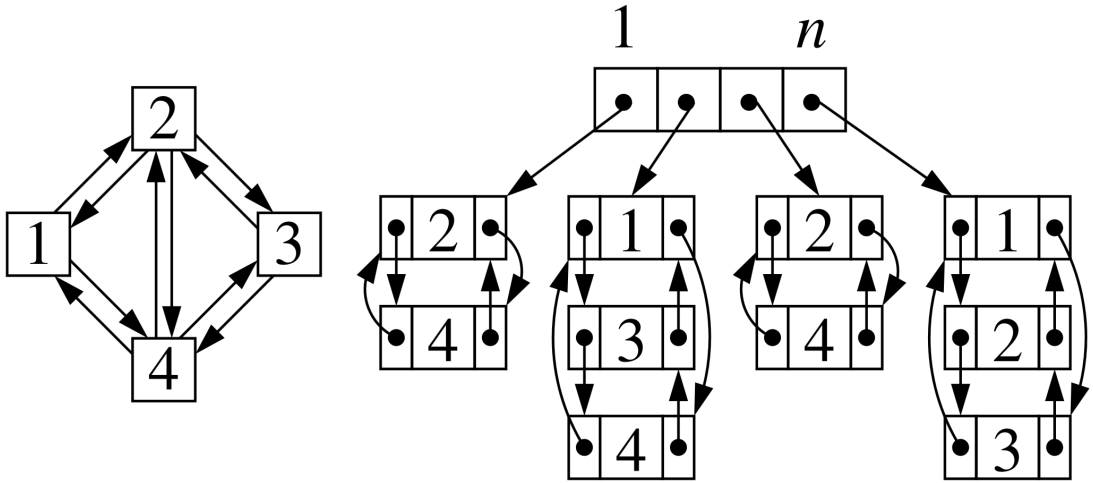


Repräsentationen von Graphen

Adjazenzliste

- Verwende **array** $A[1 \ldots n]$ von verketteten Listen
- $A[v]$: Liste aller von $v \in V$ aus **erreichbaren Knoten**

- Alle Features vom Adjazenzfeld
- ...und noch mehr: **Einfügen, Löschen** von Kanten
- Benötigt mehr **Platz** (für Zeiger)
- **Cachefeindlicher**



Repräsentationen von Graphen

Adjazenzliste

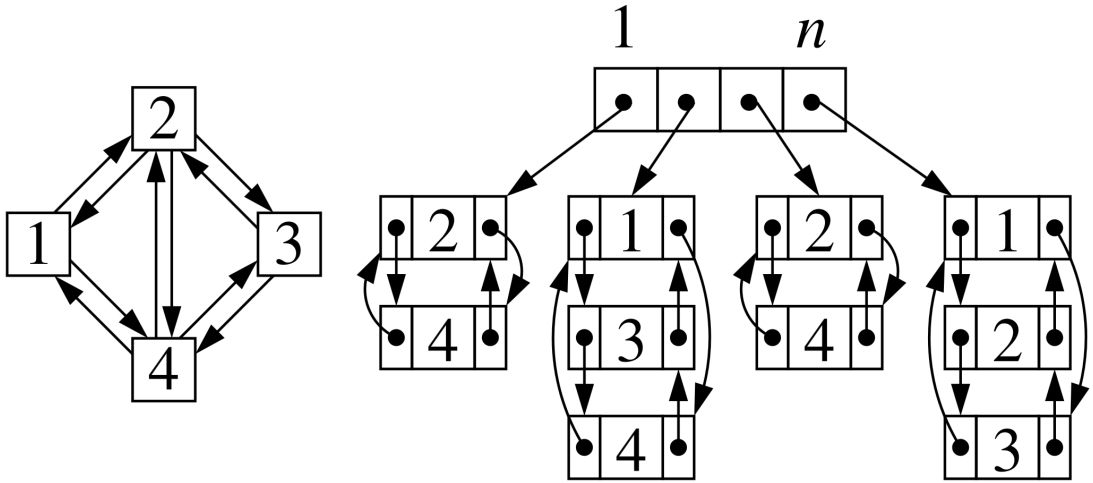
- Verwende **array** $A[1 \ldots n]$ von verketteten Listen
- $A[v]$: Liste aller von $v \in V$ aus **erreichbaren Knoten**

✚ Alle Features vom Adjazenzfeld

✚ ...und noch mehr: **Einfügen, Löschen** von Kanten

⟹ Benötigt mehr **Platz** (für Zeiger)

⟹ **Cachefeindlicher**

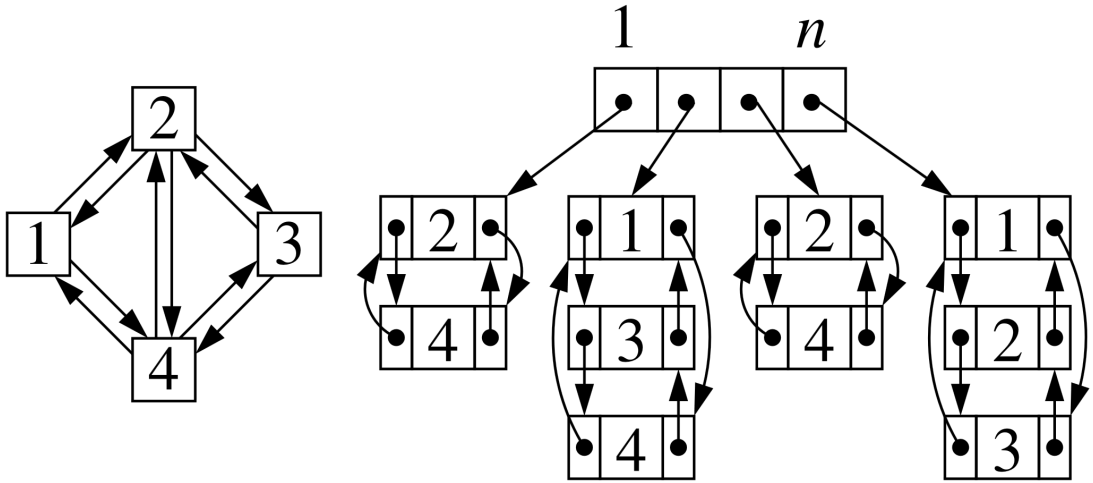


Repräsentationen von Graphen

Adjazenzliste

- Verwende **array** $A[1...n]$ von verketteten Listen
- $A[v]$: Liste aller von $v \in V$ aus **erreichbaren Knoten**

- ✚ Alle Features vom Adjazenzfeld
- ✚ ...und noch mehr: **Einfügen**, **Löschen** von Kanten
- ⟹ Benötigt mehr Platz (für Zeiger)
- ⟹ Cachefeindlicher

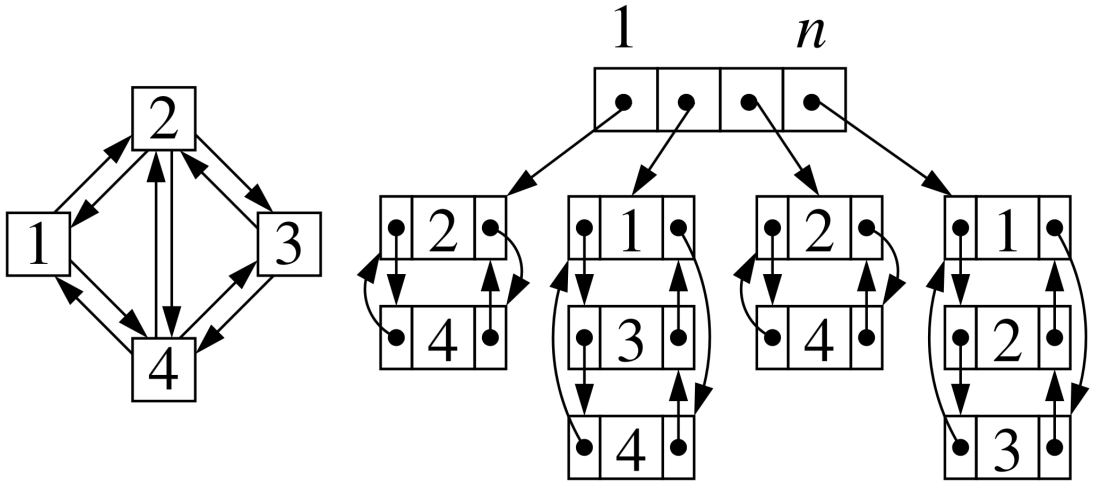


Repräsentationen von Graphen

Adjazenzliste

- Verwende **array** $A[1...n]$ von verketteten Listen
- $A[v]$: Liste aller von $v \in V$ aus **erreichbaren Knoten**

- ✚ Alle Features vom Adjazenzfeld
- ✚ ...und noch mehr: **Einfügen**, **Löschen** von Kanten
- ▬ Benötigt mehr **Platz** (für Zeiger)
- ▬ Cachefeindlicher

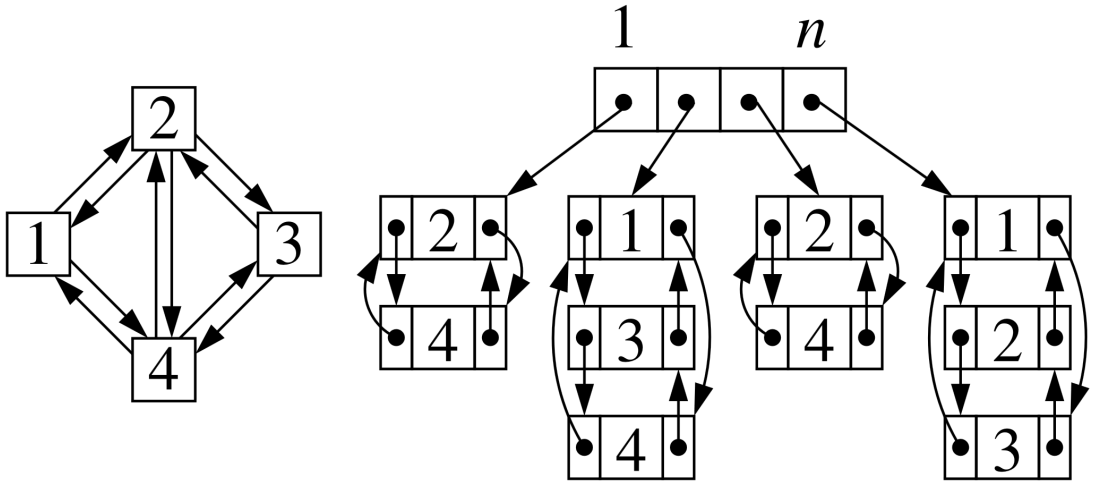


Repräsentationen von Graphen

Adjazenzliste

- Verwende **array** $A[1...n]$ von verketteten Listen
- $A[v]$: Liste aller von $v \in V$ aus **erreichbaren Knoten**

- ✚ Alle Features vom Adjazenzfeld
- ✚ ...und noch mehr: **Einfügen**, **Löschen** von Kanten
- ▬ Benötigt mehr **Platz** (für Zeiger)
- ▬ **Cachefeindlicher**



Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

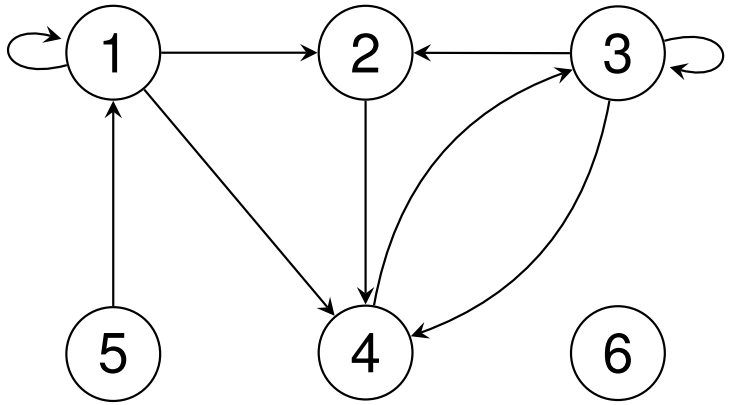
Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 1: Lesen in der Matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



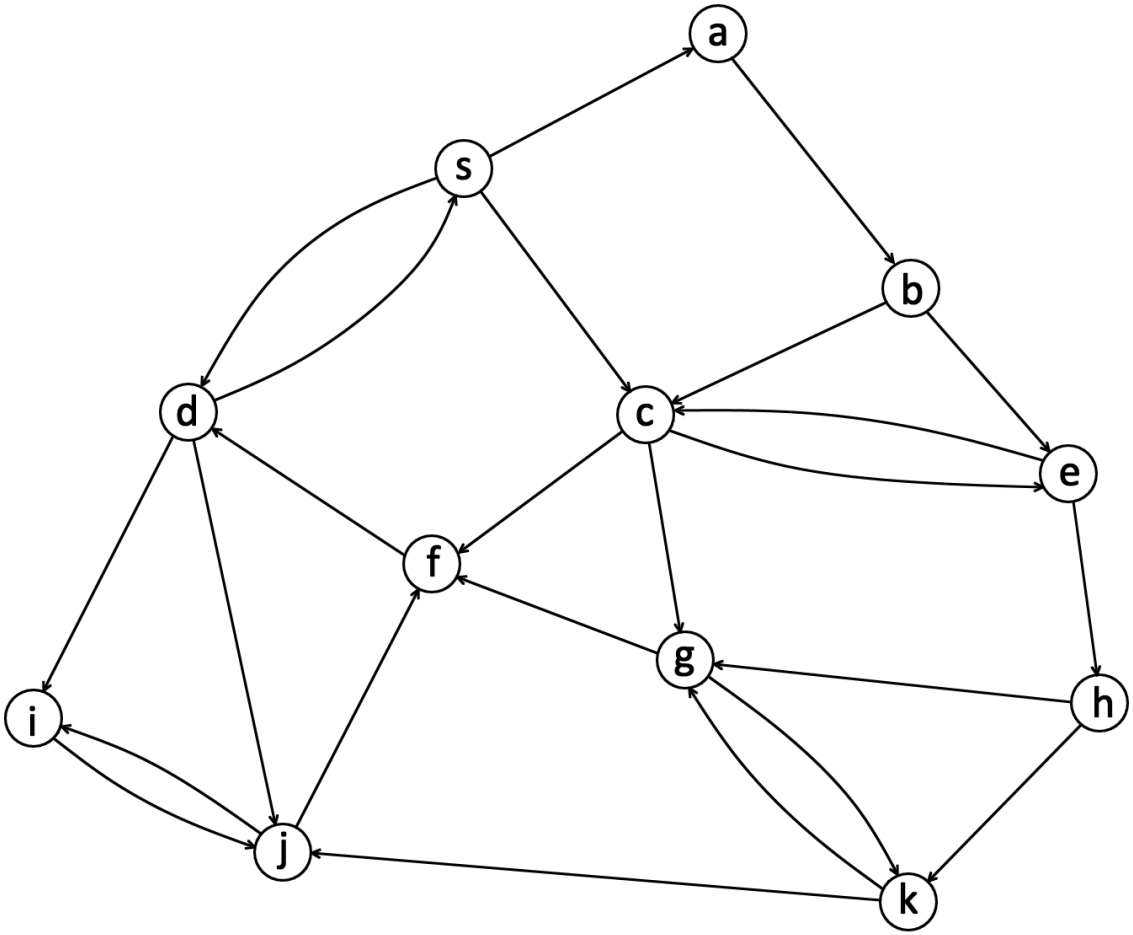
Was kann man an der Adjazenzmatrix ablesen?

- Gerichtet oder ungerichtet? $\Rightarrow$ gerichtet (weil A nicht symm.)
- Schlingen? $\Rightarrow$ Auf der Diagonalen von A: Knoten 1, 3
- Zusammenhängend? $\Rightarrow$ Nein (6 ist isoliert).
- Zeichnet den Graphen und stellt ihn als Adjazenzfeld und Kantenfolge dar (alphabetisch geordnet).

Repräsentationen von Graphen

Aufgabe 2: Malen nach Zahlen

Stellt diesen Graphen als Adjazenzfeld, Adjazenzmatrix und Kantenfolge dar (alphabetisch geordnet).



Graphenbeweise

Aufgabe 3: I Wanna Ride My Acycle!

Es sei $G = (V, E)$ ein gerichteter azyklischer Graph (DAG) mit endlich vielen und mindestens einem Knoten. Zeige, dass G mindestens einen Knoten mit Eingangsgrad 0 besitzt.

Lösung zu Aufgabe 3

Angenommen, jeder Knoten hat Eingangsgrad $\geqslant 1$. (= Gegenteil.)
Nehme irgendeinen Knoten v. Auf diesen zeigt also garantiert ne Kante.
Laufe sie **rückwärts** $\rightsquigarrow$ neuer Knoten. **Wiederhole** beliebig oft (das geht dank Annahme!).
Also geht das öfter, als G Knoten hat $\Rightarrow$ Irgendwann ein Knoten 2x besucht $\Rightarrow$ Wir laufen im Kreis $\Rightarrow \lightning$ G kreisfrei.

Graphenbeweise

Aufgabe 3: I Wanna Ride My Acycle!

Es sei $G = (V, E)$ ein gerichteter azyklischer Graph (DAG) mit endlich vielen und mindestens einem Knoten. Zeige, dass G mindestens einen Knoten mit Eingangsgrad 0 besitzt.

Lösung zu Aufgabe 3

Angenommen, jeder Knoten hat Eingangsgrad $\geqslant 1$. ($=$ Gegenteil.)
Nehme irgendeinen Knoten v. Auf diesen zeigt also garantiert ne **Kante**.
Laufe sie **rückwärts** $\rightsquigarrow$ neuer Knoten. **Wiederhole** beliebig oft (das geht dank Annahme!).
Also geht das öfter, als G Knoten hat $\Rightarrow$ Irgendwann ein Knoten 2x besucht $\Rightarrow$ Wir laufen im Kreis $\Rightarrow$  G kreisfrei.