

# Digraphen, DAGs und Wurzelbäume

## Eingangs- und Ausgangsgrad

Bei einer gerichteten Kante  $e = (u, v) \in E$  heißt  $u$  *Startknoten* von  $e$ ;  $v$  heißt *Endknoten* von  $e$ ;  $e$  heißt eine aus  $u$  *austretende* und in  $v$  *mündende* Kante.

**Eingangsgrad:**  $\text{indeg}(v)$  = Anzahl der in  $v$  mündenden Kanten.

**Ausgangsgrad:**  $\text{outdeg}(v)$  = Anzahl der aus  $v$  austretenden Kanten.

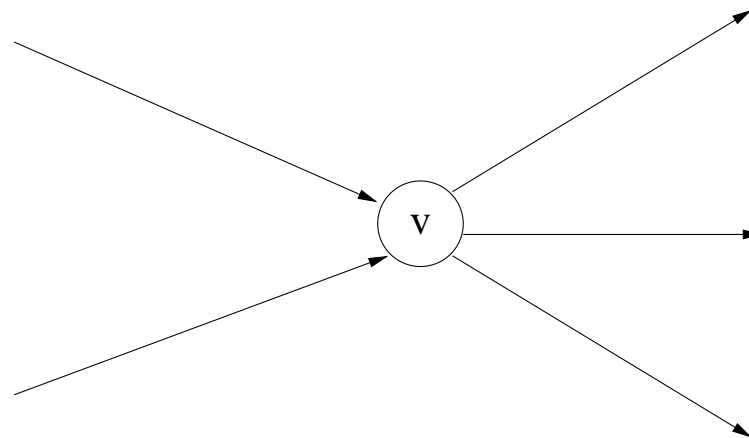
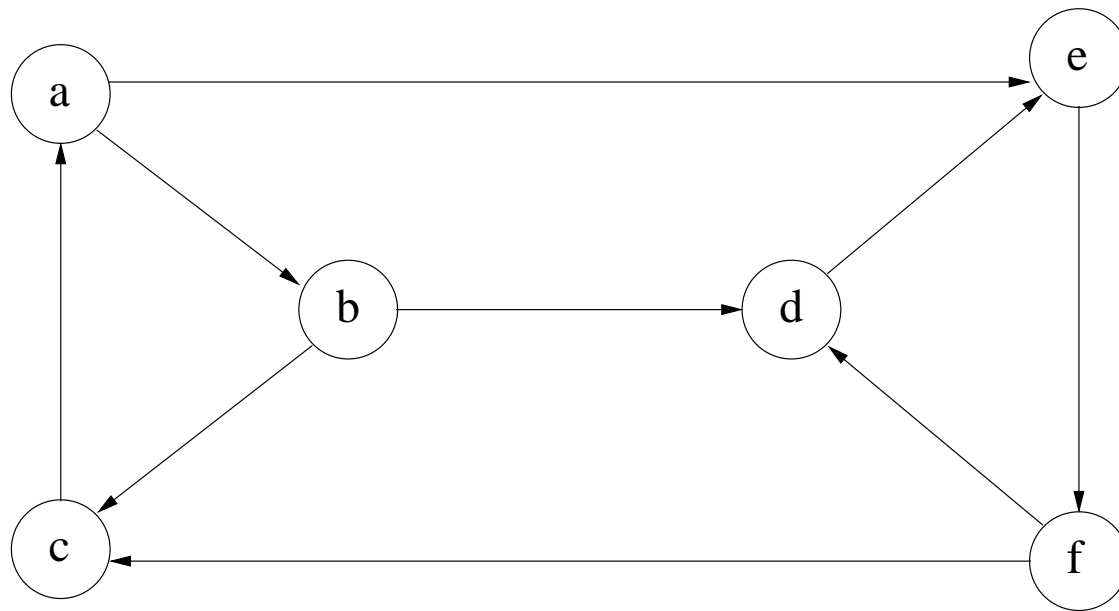


Abbildung 1: Ein Knoten  $v$  mit  $\text{indeg}(v) = 2$  und  $\text{outdeg}(v) = 3$ .

## Gerichtete Wege, Pfade, Kreise

Analog definiert wie bei Graphen außer dass Kanten nur entlang ihrer Orientierung durchlaufen werden dürfen (Einbahnstraßen!).



Weg: b c a b d e

Pfad: a b d e f

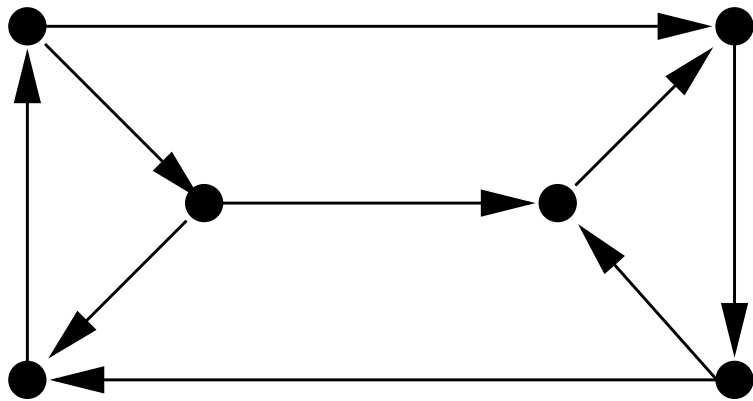
Kreis: b d e f c a

## Starke Komponenten

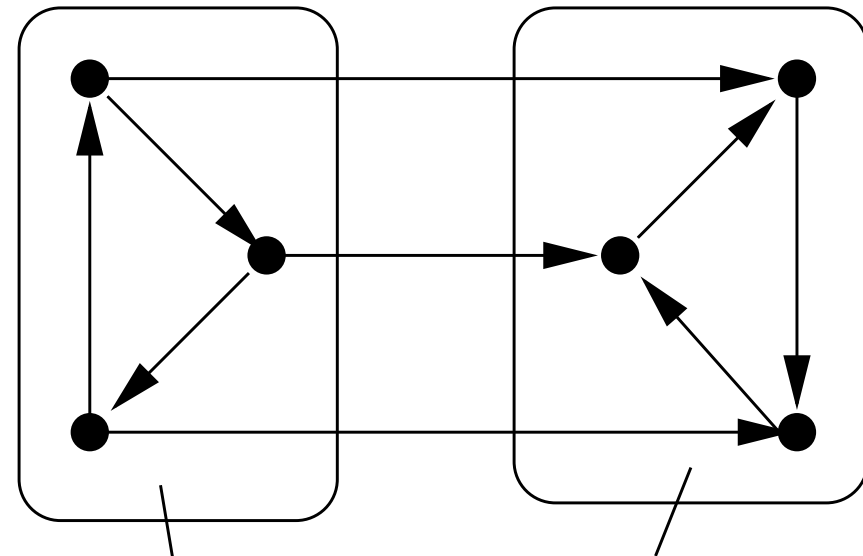
Zwei Knoten  $u, v$  eines Digraphen  $G = (V, E)$  heißen *stark zusammenhängend*, wenn es in  $G$  sowohl einen (gerichteten) Weg von  $u$  nach  $v$  als auch einen (gerichteten) Weg von  $v$  nach  $u$  gibt. In Zeichen:  $u \leftrightarrow v$ .

- Wenn es einen gerichteten Weg von  $u$  nach  $v$  gibt, dann gibt es auch einen gerichteten Pfad von  $u$  nach  $v$  (auf dem Weg liegende Kreise einfach rausschneiden!).
- „ $\leftrightarrow$ “ ist eine Äquivalenzrelation auf  $V$ .
- Die resultierenden Knoten-Äquivalenzklassen (bzw. die von ihnen aufgespannten Untergraphen) heißen *starke Komponenten*.
- Ein Digraph mit nur einer Komponente heißt *stark zusammenhängend*.

## Beispiel



(a)



Komponente 1

Komponente 2

(b)

Abbildung 2: (a) ein stark zusammenhängender Digraph (b) ein Digraph und seine starken Komponenten

## Transitive Hülle

- Ein von einem Punktpfad verschiedener Pfad heißt *echt*.
- Die *transitive Hülle*  $G^+ = (V, E^+)$  von  $G = (V, E)$  repräsentiert Pfadverbindungen in  $G$  durch Kanten:

$$E^+ := \{(u, v) \mid \text{Es gibt in } G \text{ einen echten Pfad von } u \text{ nach } v\}$$

- Die transitive Hülle ist mit dem Algorithmus von Warshall in  $O(n^3)$  Schritten berechenbar.

## Algorithmus von Warshall („high level“)

**Eingabe:** Adjazenzmatrix  $A$  von  $G = (V, E)$  mit  $V = [n]$ .

**Ausgabe:** Adjazenzmatrix  $A^+$  von  $G^+ = (V, E^+)$

**Methode:** Dynamisches Programmieren:

1. Berechne  $n$  Matrizen  $P_0, P_1, \dots, P_n$  mit folgender Interpretation:

$$P_k[i, j] = \begin{cases} 1 & \text{es gibt einen Pfad (Punktpfade ausgeschlossen)} \\ & \text{von } i \text{ nach } j \text{ mit Zwischenknoten aus } [k] \\ 0 & \text{sonst} \end{cases}$$

2. Gib  $A^+ := P_n$  als Adjazenzmatrix von  $G^+$  aus!

Bei  $k = 0$  sind also gar keine Zwischenknoten zugelassen und damit nur direkte Verbindungen durch Kanten erlaubt !

## Algorithmus von Warshall („low level“)

1. Initialisiere  $P$  mit  $P := A$  (entspricht der Matrix  $P_0$ ).
2. Die folgende 3-fach geschachtelte Laufanweisung berechnet  $P_k$  (mit Laufindex  $k = 1, \dots, n$ ):

**for**  $k := 1$  **to**  $n$  **do**

**for**  $i := 1$  **to**  $n$  **do**

**for**  $j := 1$  **to**  $n$  **do**

$$P[i, j] := P[i, j] \vee (P[i, k] \wedge P[k, j]) \quad (*)$$

**end-for**

**end-for**

**end-for;**

3. Gib  $A^+ := P$  aus.



## Algorithmus von Warshall (Korrektheitsnachweis)

Die mit (\*) markierte Programmzeile

$$P[i, j] := P[i, j] \vee (P[i, k] \wedge P[k, j])$$

berechnet  $P_k$  korrekt aus  $P_{k-1}$ , und zwar aus folgendem Grund:

- Es gibt einen Pfad von  $i$  nach  $j$  mit Zwischenknoten aus  $[k]$  **gdw** einer der folgenden beiden Fälle eintritt:
  - Es gibt einen Pfad mit Zwischenknoten aus  $[k-1]$  von  $i$  nach  $j$ .
  - Es gibt einen Pfad mit Zwischenknoten aus  $[k-1]$  sowohl von  $i$  nach  $k$  als auch von  $k$  nach  $j$ .

## Azyklische Digraphen, topologische Nummerierung

- Ein Digraph ohne (gerichtete) Kreise heißt *azyklischer Digraph*, auf Englisch auch „Directed Acyclic Graph“ oder einfach „DAG“ genannt.
- In einem DAG heißt  $u$  ein (*echter*) *Vorgänger* von  $v$  bzw.  $v$  ein (*echter*) *Nachfolger* von  $u$ , falls ein (echter) Pfad von  $u$  nach  $v$  existiert.
- Eine bijektive Abbildung  $N : V \rightarrow [n]$  heißt *topologische Nummerierung* der Knoten des DAG  $G = (V, E)$  **gdw**  $N(u) < N(v)$  für alle  $(u, v) \in E$  gilt. (Echte Vorgänger eines Knoten haben also stets kleinere Nummern als der Knoten selbst.)

## Beispiel

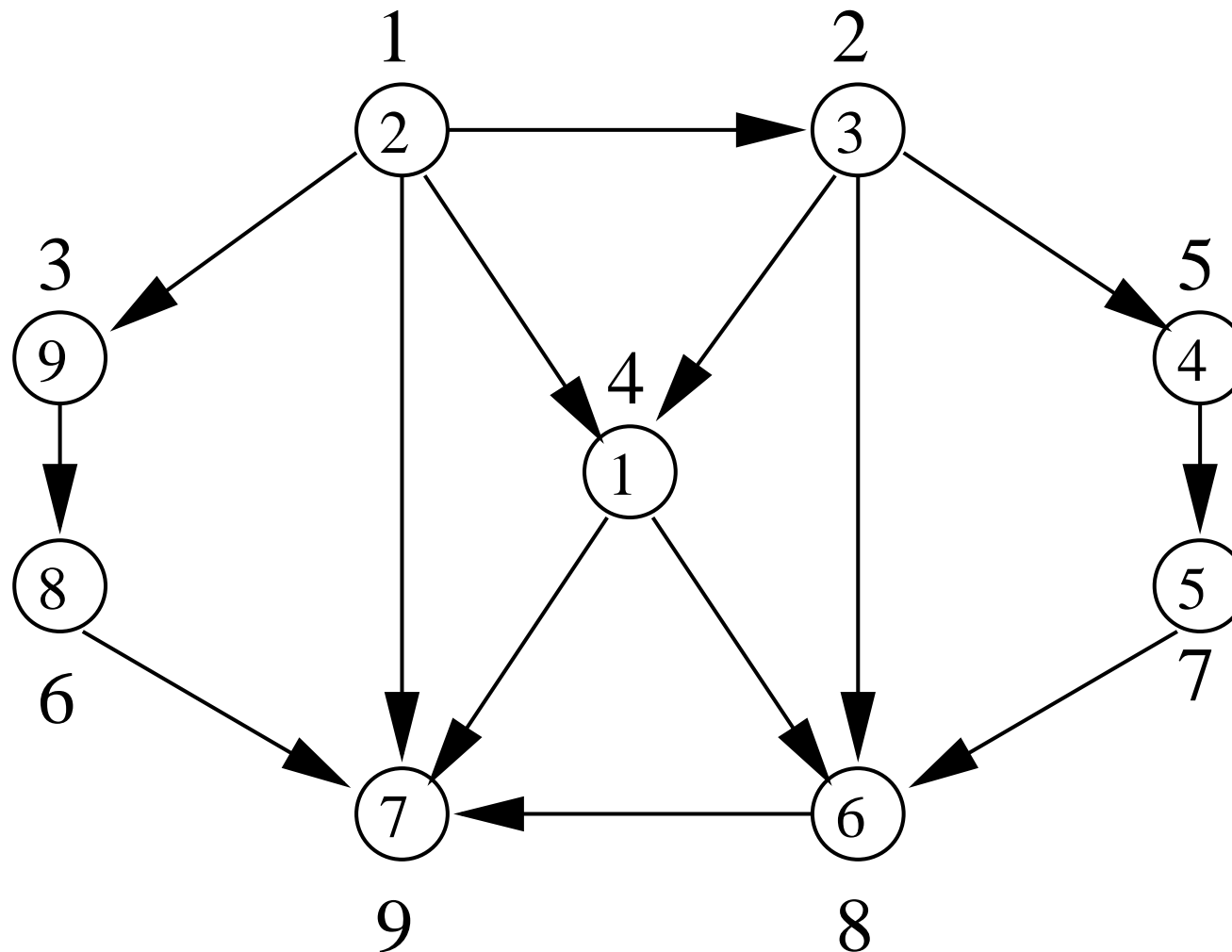
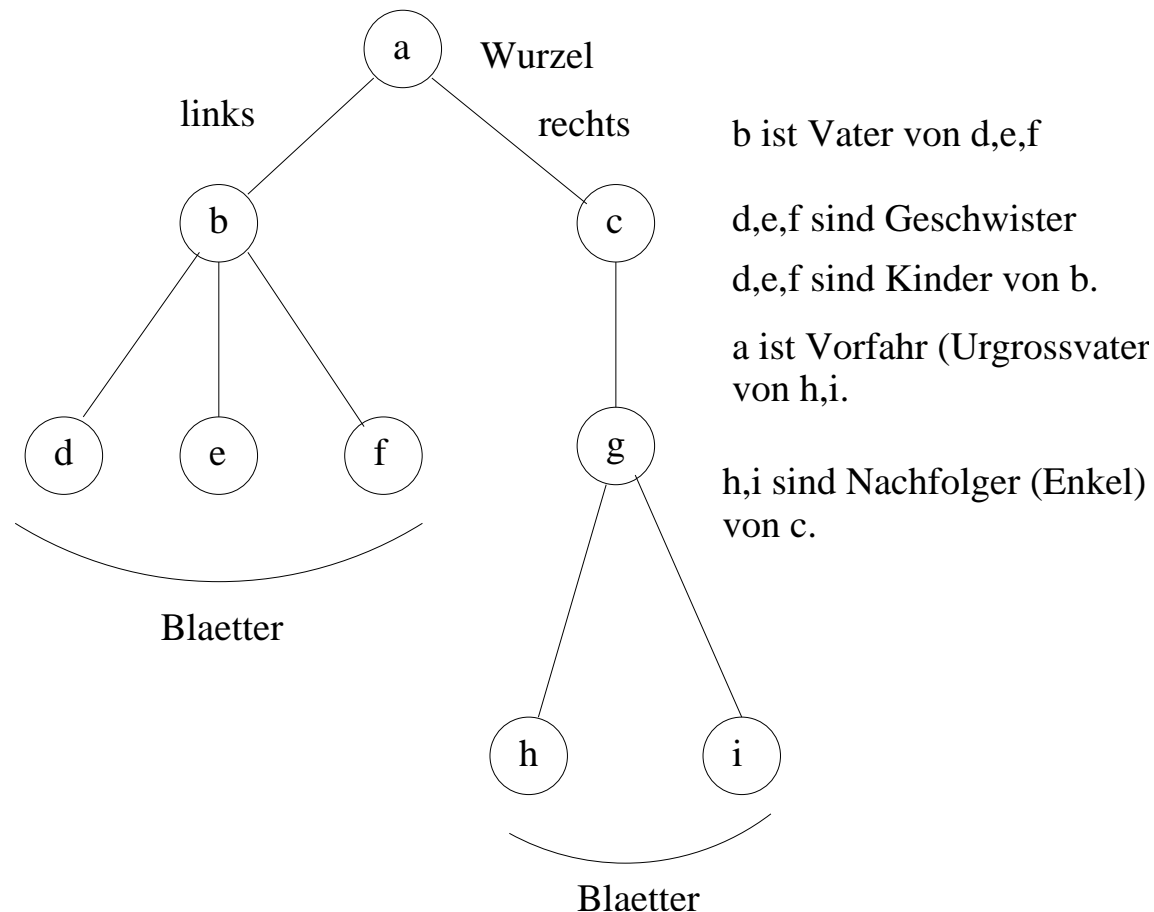


Abbildung 3: Ein DAG und eine topologische Nummerierung seiner Knoten.

## Wurzelbäume

Ein Wurzelbaum ist ein Baum  $T = (V, E)$ , bei dem ein Knoten als *Wurzel* ausgezeichnet ist. Graphische Visualisierung und Terminologie lehnen sich an „Familien-Stammbäume“ an:



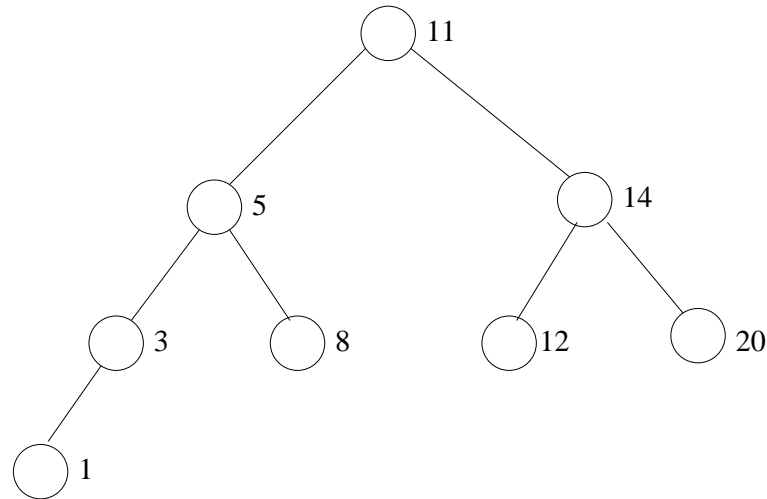
## Hierarchische Partitionierungen, Tiefe

Wurzelbäume eignen sich zur Visualisierungen von Hierarchien wie zum Beispiel:

- Verzeichnisse und Unterverzeichnisse auf einem PC
- Untergliederung einer Firma in Abteilungen, Gruppen, Labors.
- Untergliederung einer Fakultät in Fachbereiche, Institute, Lehrstühle.
- usw.

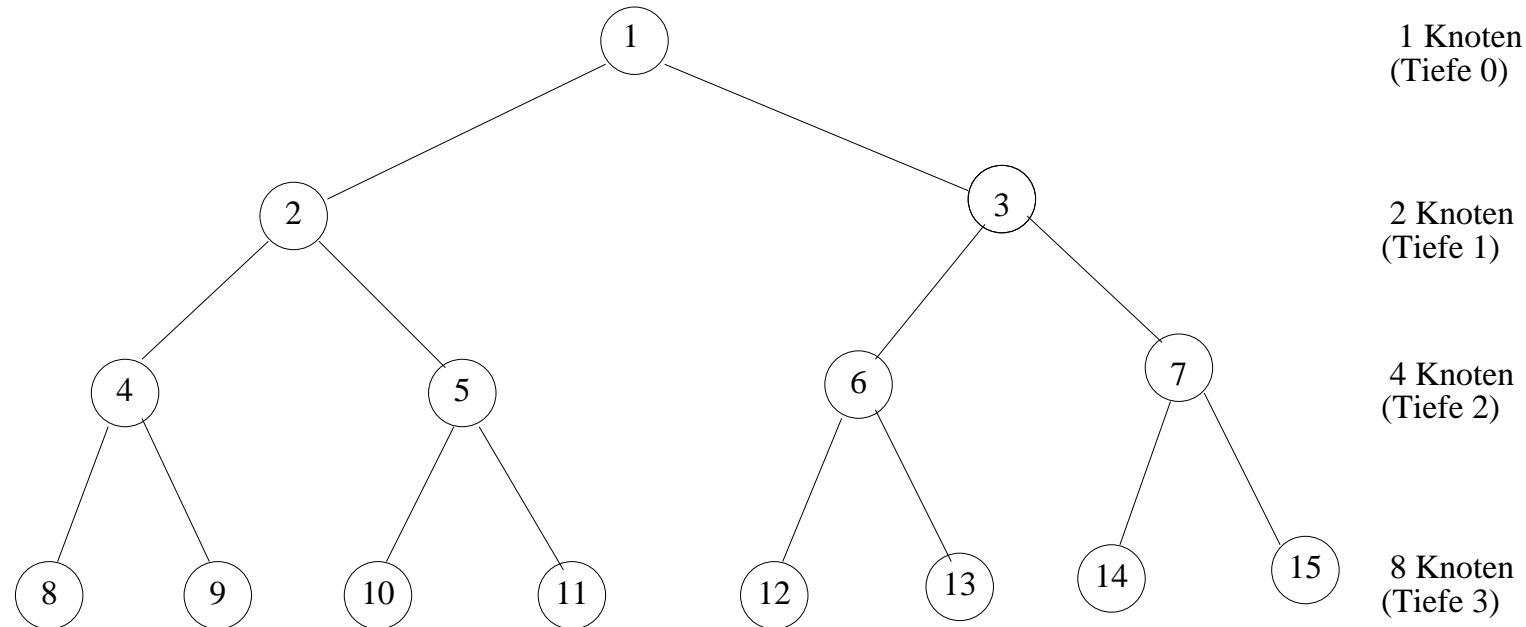
Die **Tiefe** eines Knotens  $v$  in einem Baum mit Wurzel  $r$  ist seine Entfernung zur Wurzel gemessen in der Anzahl der Kanten auf dem Pfad von  $v$  nach  $r$ .

## Binäre Suchbäume



- Der „Schlüssel“, der an einem Knoten gespeichert ist, ist stets größer als alle Schlüssel im linken Unterbaum und stets kleiner als alle Schlüssel im rechten Unterbaum.
- Dieses Organisationsprinzip eröffnet die Möglichkeit der (überaus effizienten) „Binärsuche“.

## Vollständige Binärbäume



- Knoten  $i$  hat die Kinder  $2i$  und  $2i + 1$  (daher implizite Darstellung in einem Array  $T[1 : n]$  möglich).
- Auf dem Level der „Tiefe“  $d$  gibt es  $2^d$  Knoten.
- Datenstruktur findet Anwendung beim Sortierverfahren „Heapsort“ (mehr dazu in der Vorlesung „Datenstrukturen“).