

# Algorithmen und Datenstrukturen (für ET/IT)

Sommersemester 2017

Dr. Stefanie Demirci

Computer Aided Medical Procedures  
Technische Universität München



- 7 Fortgeschrittene Datenstrukturen
  - Graphen
  - Bäume

2

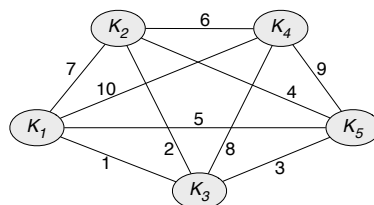
## Übersicht: Graphen

Klassen von Graphen:

- ungerichteter Graph
- gerichteter Graph

Zusätzliche Eigenschaften:

- gewichteter Graph



## Definition: Ungerichteter Graph

### Definition: Ungerichteter Graph

Ein ungerichteter Graph ist ein Paar  $G = (V, E)$  mit

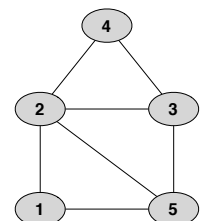
- $V$  endliche Menge der **Knoten**
- $E \subseteq \{\{u, v\} : u, v \in V\}$  Menge der **Kanten**

• auf Englisch:

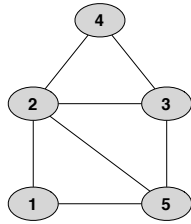
- Knoten = **vertices**
- Kanten = **edges**

• es ist  $\{u, v\} = \{v, u\}$ , d.h. Richtung der Kante spielt keine Rolle

Beispiel:



## Ungerichteter Graph: Beispiel



- Graph  $G_u = (V_u, E_u)$
- Knoten  $V_u = \{1, 2, 3, 4, 5\}$
- Kanten  
 $E_u = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}\}$

5

## Definition: Gerichteter Graph

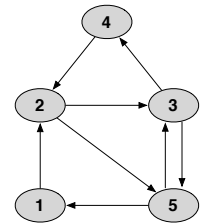
### Definition: Gerichteter Graph

Ein **gerichteter Graph** ist ein Paar  $G = (V, E)$  mit

- $V$  endliche Menge der **Knoten**
- $E \subseteq V \times V$  Menge der **Kanten**

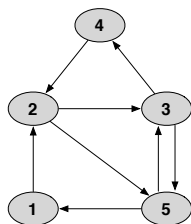
- $E \subseteq \{(u, v) : u, v \in V\} = V \times V$
- es ist  $(u, v) \neq (v, u)$ , d.h. Richtung der Kante spielt eine Rolle
- hier sind Schleifen möglich, d.h. Kanten der Form  $(u, u)$  für  $u \in V$

Beispiel:



6

## Gerichteter Graph: Beispiel



- Graph  $G_g = (V_g, E_g)$
- Knoten  $V_g = \{1, 2, 3, 4, 5\}$
- Kanten  
 $E_g = \{(1, 2), (2, 3), (2, 5), (3, 4), (3, 5), (4, 2), (5, 1), (5, 3)\}$

7

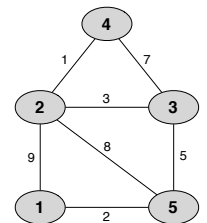
## Definition: Gewichteter Graph

### Definition: Gewichteter Graph

Ein **gewichteter Graph** ist ein Graph  $G = (V, E)$  mit einer Gewichtsfunktion  $w : E \rightarrow \mathbb{R}$ .

- der Graph  $G$  kann gerichtet oder ungerichtet sein
- je nach Anwendung kann ein verschiedener Wertebereich für die Funktion  $w$  gewählt werden
  - z.B.  $\mathbb{R}$  oder  $\mathbb{N}_0$

Beispiel:



8

## Eigenschaften von Graphen I

Sei  $G = (V, E)$  ein Graph (gerichtet oder ungerichtet).

- Ist  $(u, v) \in E$  bzw.  $\{u, v\} \in E$  für  $u, v \in V$ , so heißt  $v$  **adjazent** zu  $u$ .
- Sei  $G$  gerichteter Graph, sei  $v \in V$  ein Knoten.
  - die Anzahl der **eintretenden** Kanten in  $v$  heißt **Eingangsgrad** (englisch: indegree) von  $v$ ,

$$\text{indeg}(v) = |\{v' : (v', v) \in E\}|$$

- die Anzahl der **austretenden** Kanten von  $v$  heißt **Ausgangsgrad** (englisch: outdegree) von  $v$ ,

$$\text{outdeg}(v) = |\{v' : (v, v') \in E\}|$$

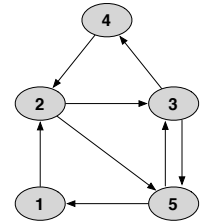
- Sei  $G$  ungerichteter Graph, sei  $v \in V$  ein Knoten.
  - die Anzahl der eintretenden bzw. austretenden Kanten von  $v$  heißt **Grad** (englisch: degree) von  $v$  oder kurz  $\text{deg}(v)$ .

9

## Eigenschaften von Graphen: Beispiel

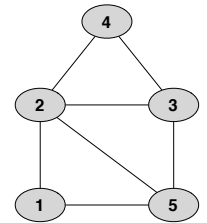
Beispiel gerichteter Graph:

- Knoten 2 ist adjazent zu Knoten 1
- Knoten 1 ist adjazent zu Knoten 5
- $\text{outdeg}(2) = 2$ ,  $\text{indeg}(2) = 2$
- $\text{indeg}(1) = 1$ ,  $\text{outdeg}(1) = 1$



Beispiel ungerichteter Graph:

- Knoten 4 ist adjazent zu Knoten 2
- Knoten 2 ist adjazent zu Knoten 4
- $\text{deg}(2) = 4$
- $\text{deg}(1) = 2$



10

## Eigenschaften von Graphen II

Sei  $G = (V, E)$  ein Graph (gerichtet oder ungerichtet).

- Seien  $v, v' \in V$ . Ein **Pfad** von  $v$  nach  $v'$  ist eine Folge von Knoten  $(v_0, v_1, \dots, v_k) \subset V$  mit
  - $v_0 = v$ ,  $v_k = v'$
  - $(v_i, v_{i+1}) \in E$  bzw.  $\{v_i, v_{i+1}\} \in E$  für  $i = 0, \dots, k-1$ $k$  heißt **Länge** des Pfades.
- Ein Pfad heißt **einfach**, falls alle Knoten des Pfades paarweise verschieden sind.

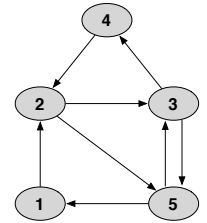
Gibt es einen Pfad von  $u$  nach  $v$ , so heißt  $v$  **erreichbar** von  $u$ .

11

## Eigenschaften von Graphen II: Beispiel

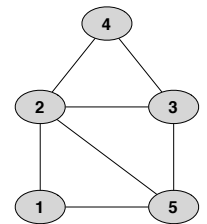
Beispiel gerichteter Graph:

- $(1, 2, 5)$  ist ein einfacher Pfad der Länge 2
- $(2, 3, 4, 2)$  ist ein Pfad der Länge 3, aber nicht einfach
- 5 ist erreichbar von 1



Beispiel ungerichteter Graph:

- $(5, 2, 4, 3, 2)$  ist ein Pfad der Länge 4, aber nicht einfach
- $(1, 2, 3, 4)$  ist ein einfacher Pfad der Länge 3
- 3 ist erreichbar von 1



12

## Eigenschaften von Graphen III

Sei  $G = (V, E)$  gerichteter Graph.

- Ein Pfad  $(v_0, \dots, v_k)$  heißt **Zyklus**, falls
  - $v_0 = v_k$  und
  - der Pfad mindestens eine Kante enthält
- Ein Zyklus  $(v_0, \dots, v_k)$  heißt **einfach**, falls  $v_1, \dots, v_k$  paarweise verschieden sind.

Sei  $G = (V, E)$  ungerichteter Graph.

- Ein Pfad  $(v_0, \dots, v_k)$  heißt **Zyklus**, falls
  - $v_0 = v_k$
  - $k \geq 3$
  - $v_1, \dots, v_k$  paarweise verschieden

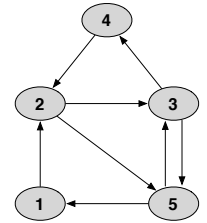
Ein Graph ohne Zyklen heißt **azyklisch**.

13

## Eigenschaften von Graphen III: Beispiel

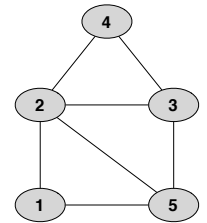
Beispiel gerichteter Graph:

- $(1, 2, 5, 3, 5, 1)$  ist ein Zyklus, aber nicht einfach
- $(1, 2, 5, 1)$  ist ein einfacher Zyklus



Beispiel ungerichteter Graph:

- $(1, 2, 5, 1)$  ist ein Zyklus
- $(2, 3, 4, 2)$  ist ein Zyklus



14

## Eigenschaften von Graphen IV

Sei  $G = (V, E)$  gerichteter Graph.

- $G$  heißt **stark zusammenhängend**, falls jeder Knoten von jedem anderen Knoten aus erreichbar ist.
- Eine **starke Zusammenhangskomponente** von  $G$  ist ein maximaler zusammenhängender Untergraph von  $G$ .
  - alternativ: Äquivalenzklassen der Knoten bezüglich Relation "gegenseitig erreichbar"

Sei  $G = (V, E)$  ungerichteter Graph.

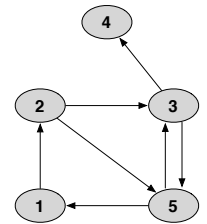
- $G$  heißt **zusammenhängend**, falls jeder Knoten von jedem anderen Knoten aus erreichbar ist.
- Eine **Zusammenhangskomponente** von  $G$  ist ein maximaler zusammenhängender Untergraph von  $G$ .
  - alternativ: Äquivalenzklassen der Knoten bezüglich Relation "erreichbar von"

15

## Eigenschaften von Graphen IV: Beispiel

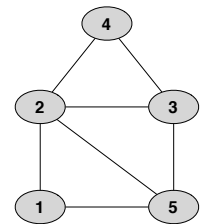
Beispiel gerichteter Graph:

- Graph ist **nicht** stark zusammenhängend (z.B. 3 nicht erreichbar von 4)
- starke Zusammenhangskomponenten:  $\{1, 2, 3, 5\}$  und  $\{4\}$



Beispiel ungerichteter Graph:

- Graph ist zusammenhängend
- nur eine Zusammenhangskomponente:  $\{1, 2, 3, 4, 5\}$



16

## Darstellung von Graphen: Adjazenzmatrizen

### Adjazenzmatrix

Sei  $G = (V, E)$  mit  $V = \{v_1, \dots, v_n\}$ . Die Adjazenzmatrix von  $G$  speichert die vorhandenen Kanten in einer  $n \times n$  Matrix  $A \in \mathbb{R}^{n \times n}$  mit

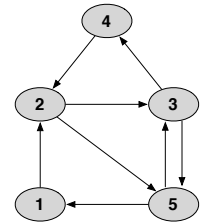
- $A(i, j) = 1$  falls Kante von Knoten  $v_i$  zu  $v_j$  existiert
- $A(i, j) = 0$  falls keine Kante von Knoten  $v_i$  zu  $v_j$  existiert für  $i, j \in \{1, \dots, n\}$ .

17

## Adjazenzmatrizen: Beispiele

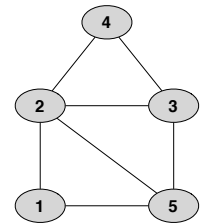
Beispiel gerichteter Graph:

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



Beispiel ungerichteter Graph:

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



18

## Adjazenzmatrizen: Eigenschaften

Eigenschaften von Adjazenzmatrizen zu Graph  $G = (V, E)$

- sinnvoll wenn der Graph nahezu **vollständig** ist (d.h. fast alle möglichen Kanten tatsächlich in  $E$  liegen)
- Speicherkomplexität:  $O(|V|^2)$
- bei **ungerichteten** Graphen ist die Adjazenzmatrix **symmetrisch**
- bei **gewichteten** Graphen kann man statt der 1 in der Matrix das Gewicht der Kante eintragen

19

## Darstellung von Graphen: Adjazenzlisten

### Adjazenzliste

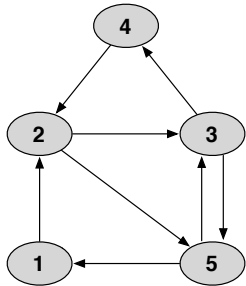
Sei  $G = (V, E)$  gerichteter Graph. Eine Adjazenzliste von  $G$  sind  $|V| + 1$  verkettete Listen, so daß

- die erste Liste alle Knoten enthält
- für jeden Knoten  $v$  eine Liste angelegt wird mit allen Knoten, die durch eine von  $v$  austretende Kante zu erreichen sind

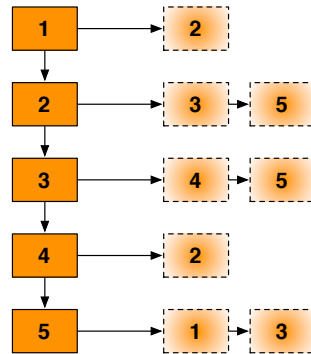
20

## Adjazenzliste: Beispiel

Graph:



Adjazenzliste:



21

## Adjazenzliste: Eigenschaften

Eigenschaften von **Adjazenzlisten** zu Graph  $G = (V, E)$

- sinnvoll bei dünn besetzten Graphen mit wenigen Kanten
- Speicherkomplexität:  $O(|V| + |E|)$
- bei **ungerichteten** Graphen gleiches Verfahren
  - allerdings muß jede Kante zweimal gespeichert werden
- bei **gewichteten** Graphen kann man die Gewichte mit in den verketteten Listen der jeweiligen Knoten speichern

22

## Komplexität der Darstellungen

Sei  $G = (V, E)$  Graph.

Operation	Adjazenzmatrix	Adjazenzliste
Kante einfügen	$O(1)$	$O( V )$
Kante löschen	$O(1)$	$O( V )$
Knoten einfügen	$O( V ^2)$	$O(1)$
Knoten löschen	$O( V ^2)$	$O( V  +  E )$

- falls Größe im Vorhinein bekannt, kann Knoten löschen/einfügen bei Adjazenzmatrix effizienter implementiert werden
- auch die Adjazenzlisten lassen sich effizienter implementieren, z.B. auf Kosten von Speicher (sequentielle Liste statt verkettete Liste)
- Löschen von Knoten ist immer aufwendig, da auch alle Kanten von/zu diesem Knoten gelöscht werden müssen

23

## Algorithmen auf Graphen

**Ausblick** auf Algorithmen auf Graphen:

- Traversierung (Durchlaufen) von allen Knoten
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
- kürzester Pfad zwischen Knoten in Graphen
- minimaler Spannbaum (minimum spanning tree, MST)

→ siehe Kapitel 9!



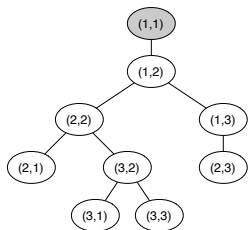
24

- 7 Fortgeschrittene Datenstrukturen
  - Graphen
  - Bäume

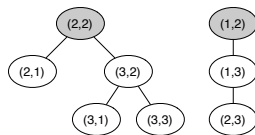
## Definition Wald/Baum

### Definition: Wald und Baum

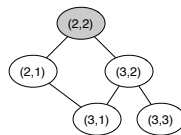
- Ein azyklischer ungerichteter Graph heißt auch **Wald**.
- Ein zusammenhängender, azyklischer ungerichteter Graph heißt auch **Baum**.



Baum



Wald



kein Baum!

# Bäume

Bäume sind alltägliches Mittel zur Strukturierung:

- Stammbaum
- Hierarchie in Unternehmen
- Systematik in der Biologie
- etc.



In Informatik:

- Bäume sind spezielle Graphen
- Wurzel oben!



## Eigenschaften von Bäumen

Sei  $G = (V, E)$  ein Baum.

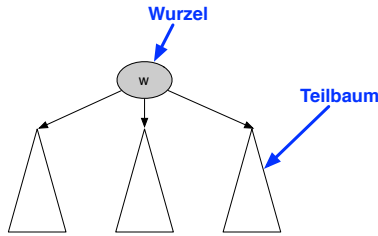
- jedes Paar von Knoten  $u, v \in V$  ist durch einen **einzigsten Pfad** verbunden
- $G$  ist **zusammenhängend**, aber wenn eine Kante aus  $E$  **entfernt** wird, ist  $G$  nicht mehr zusammenhängend
- $G$  ist **azyklisch**, aber wenn eine Kante zu  $E$  **hinzugefügt** wird, ist  $G$  nicht mehr azyklisch
- es ist  $|E| = |V| - 1$



## Wurzel von Bäumen

Sei  $G = (V, E)$  ein Baum.

- genau ein Knoten  $w \in V$  wird als **Wurzel** ausgezeichnet
- entfernt man  $w$  erhält man einen Wald von **Teilbäumen**



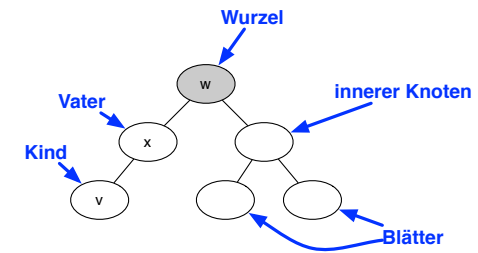
**Hinweis:** manchmal wird zwischen "freiem" und "gewurzeltem" Baum unterschieden!

29

## Weitere Begriffe bei Bäumen I

Sei  $G = (V, E)$  ein Baum mit Wurzel  $w \in V$ .

- jeder Knoten  $v \in V$  mit  $v \neq w$  ist mit genau einer Kante mit seinem **Vaterknoten**  $x \in V$  (oder: Vorgänger) verbunden
- $v$  wird dann als **Kind** (oder: Sohn, Nachfolger) von  $x \in V$  bezeichnet
- ein Knoten ohne Kinder heißt **Blatt**, alle anderen Knoten heißen **innere Knoten**

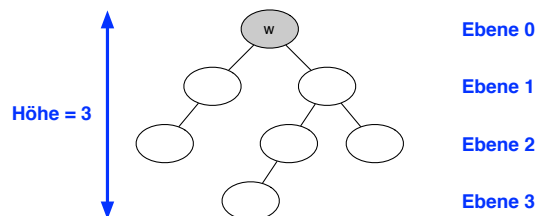


30

## Weitere Begriffe bei Bäumen II

Sei  $G = (V, E)$  ein Baum mit Wurzel  $w \in V$ .

- Anzahl der Kinder von Knoten  $x \in V$  heißt auch **Grad** von  $x$ 
  - Achtung: der Grad in dem ungerichteten Graph  $G$  ist anders definiert!
- Länge des Pfades von Wurzel  $w$  zu Knoten  $x \in V$  heißt **Tiefe** von  $x$
- alle Knoten gleicher Tiefe bilden eine **Ebene** des Baumes  $G$
- die **Höhe** des Baumes  $G$  ist die maximale Tiefe der Knoten von  $G$ 
  - Achtung: manchmal ist Höhe auch als Anzahl der Ebenen definiert



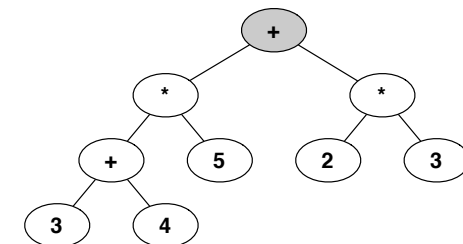
31

## Bäume: Beispiele

- **arithmetischer Ausdruck**

$$(3 + 4) * 5 + 2 * 3$$

repräsentiert als Baum:



- **hierarchisches Dateisystem**
  - Windows z.B. "C:\\"
  - Unix "/"
- **Suchbaum** → Kapitel 8

32



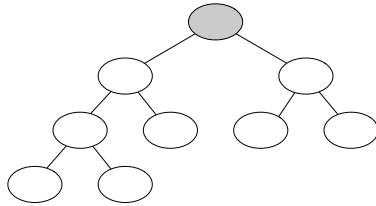
## Besondere Bäume

Sei  $G = (V, E)$  ein Baum mit Wurzel  $w \in V$ .

- sind die Kinder jedes Knotens in bestimmter Reihenfolge angeordnet, heißt  $G$  **geordneter Baum**
- ist die Anzahl  $n$  der Kinder jedes Knotens vorgegeben, heißt  $G$   **$n$ -ärer Baum**

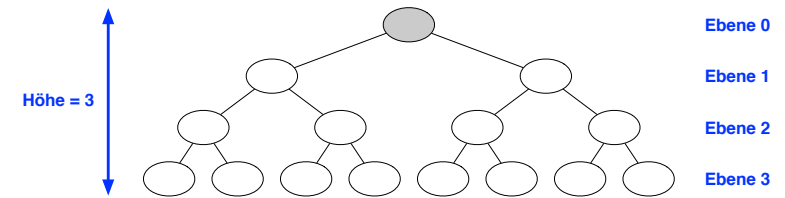
Wichtiger Spezialfall:

- ist  $G$  geordnet und hat jeder Knoten maximal zwei Kinder, heißt  $G$  **Binärbaum**



33

## Beispiel: Binärbaum



Binärbaum mit Höhe 3, 8 Blättern und 7 inneren Knoten.

- Binärbaum heißt **vollständig**, wenn jede Ebene die maximale Anzahl an Knoten enthält
- ein vollständiger Binärbaum der Höhe  $k$  hat  $2^{k+1} - 1$  Knoten, davon  $2^k$  Blätter

34

## Darstellung von Bäumen: als Graph

Bäume sind Graphen  $\rightarrow$  Darstellung als

- Adjazenzmatrix
- Adjazenzliste

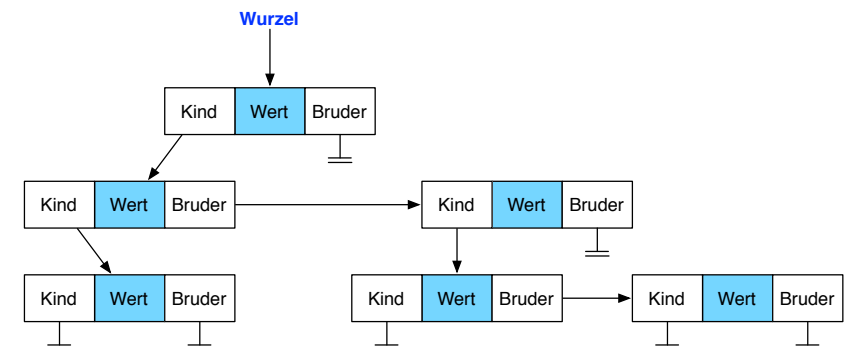
$\rightarrow$  leider meist **nicht effizient** (sowohl Laufzeit als auch Speicher)



35

## Darstellung von Bäumen: verkettete Liste

Darstellung mit angepassten verketteten Listen:

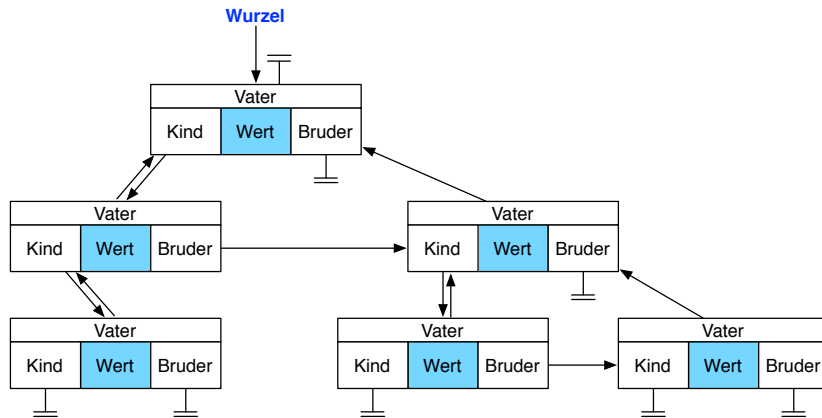


**Nachteil:** nur Navigation "nach unten" möglich

36

## Darstellung von Bäumen: doppelt verkettete Liste

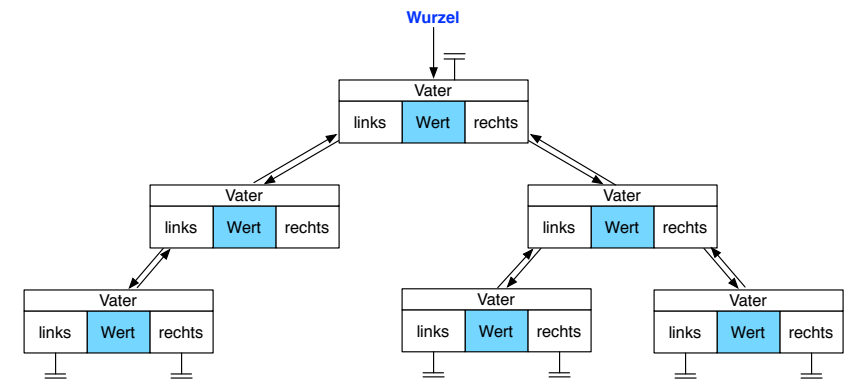
Darstellung mit angepassten doppelt verketteten Listen:



37

## Darstellung von Binärbäumen

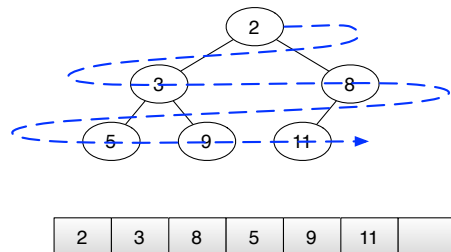
Darstellung direkt mit linkem/rechtem Kind (doppelt verkettet):



38

## Binärbaum als sequentielle Liste I

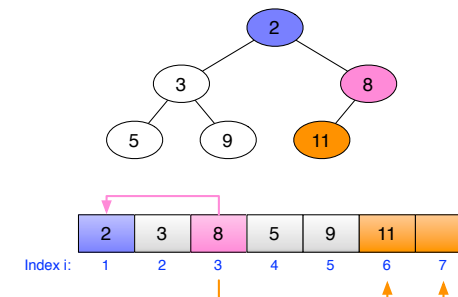
- vollständiger Binärbaum Höhe  $k$  hat  $2^{k+1} - 1$  Knoten
  - speichere Knoten von oben nach unten, von links nach rechts in sequentieller Liste (Array)
  - maximale Grösse von Array:  $2^{k+1} - 1$
- Beispiel fast vollständiger Binärbaum:



39

## Binärbaum als sequentielle Liste II

- **Wurzel:** an Position 1
  - **Knoten an Position  $i$ :**
    - **Vater-Knoten** an Position  $\lfloor i/2 \rfloor$
    - **linkes Kind** an Position  $2i$ ;
    - **rechtes Kind** an Position  $2i + 1$
- Pseudocode:**
- **vater( $i$ ):** return  $\lfloor i/2 \rfloor$ ;
  - **links( $i$ ):** return  $2i$ ;
  - **rechts( $i$ ):** return  $2i + 1$ ;



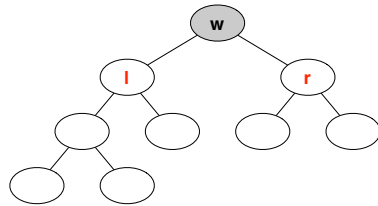
40

## Traversierung von Binärbäumen

Sei  $G = (V, E)$  Binärbaum.

In welcher Reihenfolge durchläuft man  $G$ ?

- Wurzel zuerst
- danach linker oder rechter Kind-Knoten  $l$  bzw.  $r$ ?
- falls  $l$ : danach Kind-Knoten von  $l$  oder zuerst  $r$ ?
- falls  $r$ : danach Kind-Knoten von  $r$  oder zuerst  $l$ ?



→ falls zuerst in die Tiefe: Depth-first search (DFS)

→ falls zuerst in die Breite: Breadth-first search (BFS)

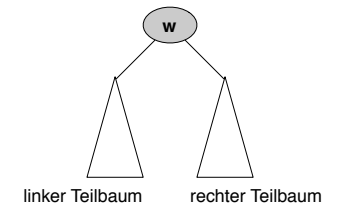
41

## DFS Binärbaum

Sei  $G = (V, E)$  Binärbaum.

Tiefensuche (Depth-first search, DFS) gibt es in 3 Varianten:

- 1 Pre-order Reihenfolge
  - besuche Wurzel
  - durchlaufe linken Teilbaum
  - durchlaufe rechten Teilbaum
- 2 In-order Reihenfolge
  - durchlaufe linken Teilbaum
  - besuche Wurzel
  - durchlaufe rechten Teilbaum
- 3 Post-order Reihenfolge
  - durchlaufe linken Teilbaum
  - durchlaufe rechten Teilbaum
  - besuche Wurzel



42

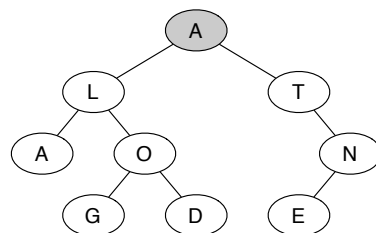
## Pre-order Traversierung

Pre-order Reihenfolge:

- besuche Wurzel
- durchlaufe linken Teilbaum
- durchlaufe rechten Teilbaum

Beispiel:

A, L, A, O, G, D, T, N, E



43

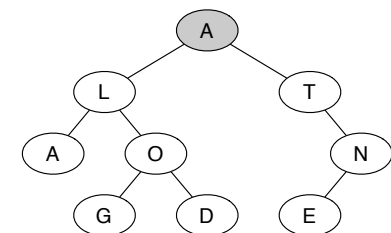
## In-order Traversierung

In-order Reihenfolge:

- durchlaufe linken Teilbaum
- besuche Wurzel
- durchlaufe rechten Teilbaum

Beispiel:

A, L, G, O, D, A, T, E, N



44

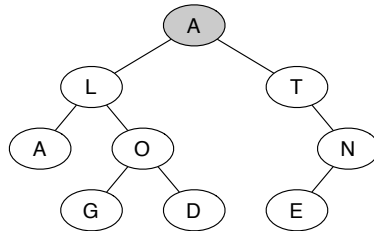
## Post-order Traversierung

Post-order Reihenfolge:

- durchlaufe linken Teilbaum
- durchlaufe rechten Teilbaum
- besuche Wurzel

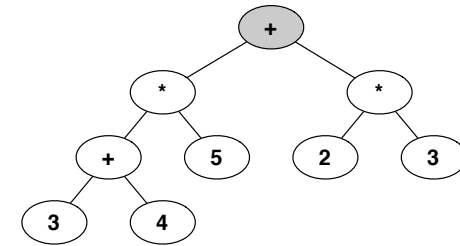
Beispiel:

A, G, D, O, L, E, N, T, A



45

## Beispiel: Arithmetischer Term



Traversierung:

- Pre-order: + \* + 3 4 5 \* 2 3
- In-order: 3 + 4 \* 5 + 2 \* 3
- Post-order: 3 4 + 5 \* 2 3 \* +

46

## Implementierung DFS Traversierung

Datenstruktur:



- Algorithmus **preorder**:  
**preorder**(knoten):  
 if (knoten == null) return;  
**besuche**(knoten);  
**preorder**(knoten.links);  
**preorder**(knoten.rechts);
- Algorithmus **inorder**:  
**inorder**(knoten):  
 if (knoten == null) return;  
**inorder**(knoten.links);  
**besuche**(knoten);  
**inorder**(knoten.rechts);
- Algorithmus **postorder**:  
**postorder**(knoten):  
 if (knoten == null) return;  
**postorder**(knoten.links);  
**postorder**(knoten.rechts);  
**besuche**(knoten);
- rekursive Algorithmen
- auf Call-Stack basiert

47

## Implementierung DFS Traversierung ohne Rekursion

- Datenstruktur:



- Hilfsmittel: Stack von Knoten "knotenStack"

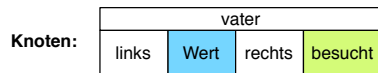
```

postorderIterativ(wurzelKnoten):
  knotenStack.push(wurzelKnoten);
  while ( !knotenStack.isEmpty() ) {
    knoten = knotenStack.top();
    if ( (knoten.links != null) && (!knoten.links.besucht) )
      knotenStack.push(knoten.links);
    else if ( (knoten.rechts != null) && (!knoten.rechts.besucht) )
      knotenStack.push(knoten.rechts);
    else {
      besuche(knoten);
      knoten.besucht = true;
      knotenStack.pop();
    }
  }
  
```

48

## Implementierung ohne Rekursion und ohne Stack

- Datenstruktur:



**postorderIterativeOhneStack**(wurzelKnoten):

```

knoten = wurzelKnoten;
while (true) {
  if ( (knoten.links != null) && (!knoten.links.besucht) )
    knoten = knoten.links;
  else if ( (knoten.rechts != null) && (!knoten.rechts.besucht) )
    knoten = knoten.rechts;
  else {
    besuche(knoten);
    knoten.besucht = true;
    if (knoten.vater == null) break;
    else knoten = knoten.vater;
  }
}

```

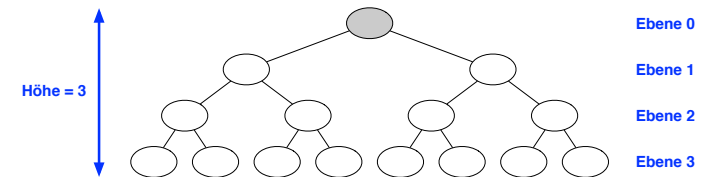
49

## BFS Binärbaum

Sei  $G = (V, E)$  Binärbaum.

Breitensuche (Breadth-first search, BFS):

- besuche Wurzel
- für alle Ebenen von 1 bis Höhe
  - besuche alle Knoten aktueller Ebene



50

## Implementierung BFS Traversierung

- Datenstruktur:



- Hilfsmittel: Queue von Knoten "knotenQueue"

**breitensuche**(wurzelKnoten):

```

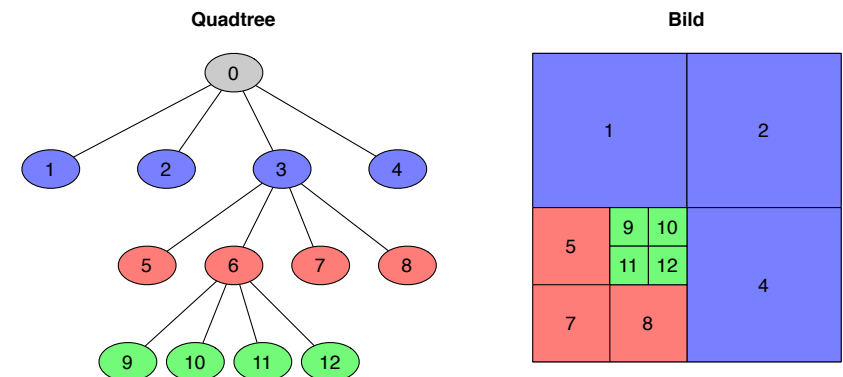
knotenQueue = leer;
knotenQueue.enqueue(wurzelKnoten);
while ( !knotenQueue.isEmpty() ) {
  knoten = knotenQueue.dequeue();
  besuche(knoten);
  if (knoten.links != null)
    knotenQueue.enqueue(knoten.links);
  if (knoten.rechts != null)
    knotenQueue.enqueue(knoten.rechts);
}

```

51

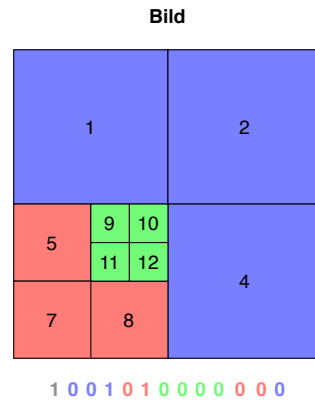
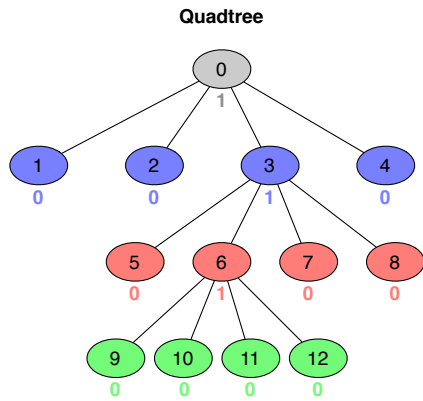
## Anwendung: Quadtree I

- 4-närer Baum heißt Quadtree



52

- Codierung des Baumes mit Binärziffern



- 7 Fortgeschrittene Datenstrukturen
  - Graphen
  - Bäume