Lineare Liste

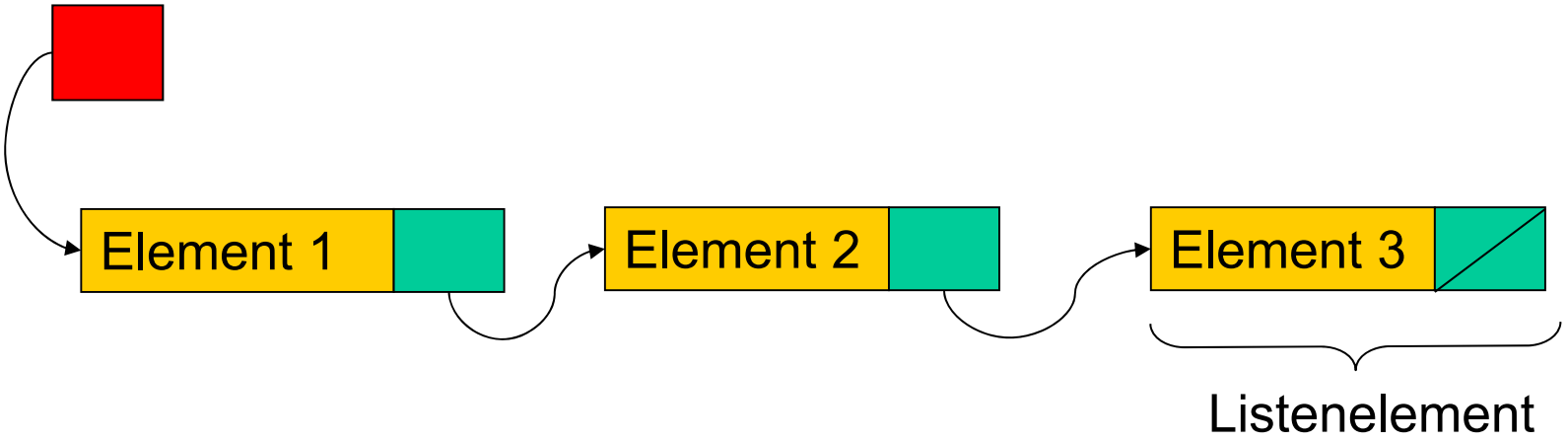
Zeigertechnik zum dynamischen Herstellen und Lösen von Verbindungen zwischen Datenelementen

Einsatz des Verbundtyps struct {...} in Kombination mit Zeigern.

Da das Element selbst eine Struktur ist, muss der Zeiger auf das nächste Element auch vom gleichen Strukturtyp sein, z.B.

```
struct list_element
   { float f;
     /* weitere Elemente */
     struct list_element *next;    /* Zeiger auf Nachfolger-Element */
   };
```

Lineare Liste




```
struct list_element
{ float f; char position[STRLEN];
  struct list_element *next;
};
typedef  struct list_element  list_elem_t;
…
list_elem_t *anker = NULL;
```

Lineare Liste - Definition

Eine Liste ist eine verkettete Folge von Elementen,
die aus Standarddatentypen zusammengesetzt sind und für die gilt:

1. Es gibt genau ein Listenelement, das keinen Vorgänger hat und Listenanfang heißt. Auf dieses Element zeigt der Listenanker.
2. Es gibt genau ein Listenelement, das keinen Nachfolger hat und Listenende heißt.
3. Die übrigen Listenelemente haben genau einen Vorgänger und genau einen Nachfolger.
4. Alle Listenelemente sind vom Listenanker aus durch Nachfolgerbildung in endlich vielen Schritten erreichbar.

Operationen mit Listen

Als typische Operationen mit Listen gelten:

- Erzeugen und Einketten eines Listenelements
- Traversieren einer Liste
- Ausketten und Löschen eines Listenelements
- Suchen eines Listenelements

Weitere:

Anhängen einer Liste an eine Liste

…

Erzeugen und Einketten eines neuen Elements

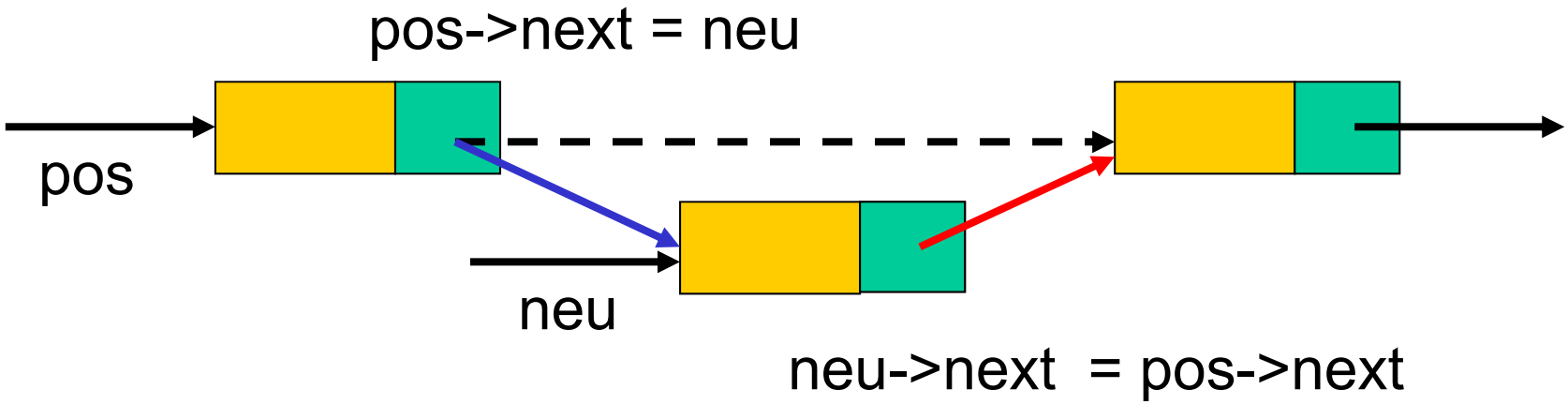
```
list_elem_t *create(list_elem_t x)
{  list_elem_t *neu;
   neu= (list_elem_t*) malloc(sizeof list_elem_t);
   *neu=x;
   return neu;
}
```

Aufruf:

```
list_elem_t daten = {23.5, “Waldhausen“,NULL};

list_elem_t *ne = create(daten);

…

insert(pos, ne);
```

Einketten eines Listenelements

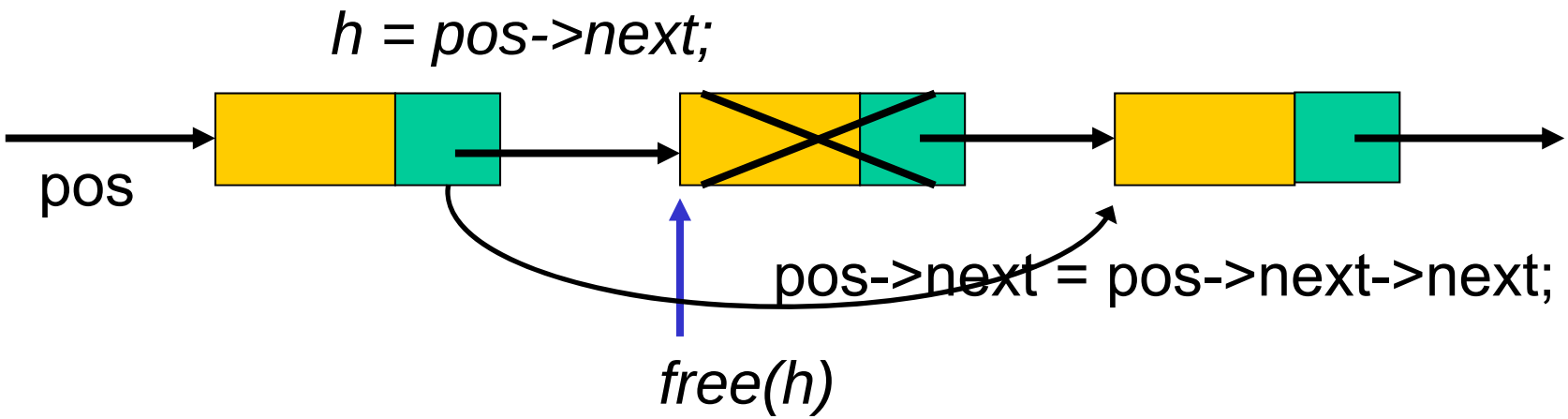
```
void insert(list_elem_t *pos, list_elem_t *neu)
{  /* pos zeigt auf das Listenelement, hinter dem das
      Listenelement neu eingekettet werden soll */
   neu->next=pos->next;
   pos->next = neu;
}
```



Noch nicht berücksichtigt: Einfügen in leere Liste und Einfügen an erster Posititon

Ausketten und Löschen eines Listenelements

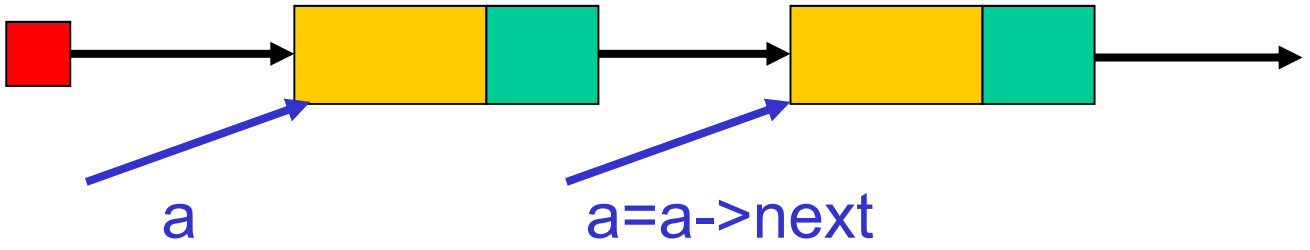
```
void dequeue(list_elem_t *pos)
{  /* pos zeigt auf Element vor dem auszukettenden Element */
   list_elem_t *h;
   h=pos->next;
   pos->next=(pos->next)->next;
   free(h);
}
```



Gesonderte Behandlung erforderlich, wenn erstes Element auszuketten ist und wenn einziges (letztes) Element ausgekettet wird.

Suchen eines Listenelements

```
list_elem_t *find(list_elem_t *anker, char *suchpos)
{ list_elem_t *a;
  a=anker;
  while (a!=NULL)
  {
    if (strcmp(a->posítion, suchpos)==0) return a;
                                  /*Element a gefunden*/
    a=a->next;
  }
  return NULL;                          /* nichts gefunden */
}
```



Traversieren einer Liste

```
void traverse( list_elem_t *anker, void (*f)(list_elem_t *e) )
{ list_elem_t *a;
  a=anker;
  while (a!=NULL)
  {
    /* mache etwas mit dem aktuellen Element */
    (*f)(a);   /* hier können auch direkt auszuf. Anweisungen stehen */
    a=a->next;
  }
}
```

Nichtlineare dynamische Datenstrukturen

Eine nichtlinear verkettete Menge von Elementen,
die aus Standarddatentypen zusammengesetzt sind, nennt man nichtlineare dynamische Datenstruktur.

Solche Strukturen sind vor allem:

- Bäume (Binärbäume, allg. Bäume)
  typisch für diese Strukturen ist die Anordnung als Hierarchie, d.h. zwischen den Elementen besteht noch eine Halbordnungsrelation
- Graphen
  netzartige Strukturen als allgemeinster Fall.

Nichtlineare dynamische Datenstrukturen

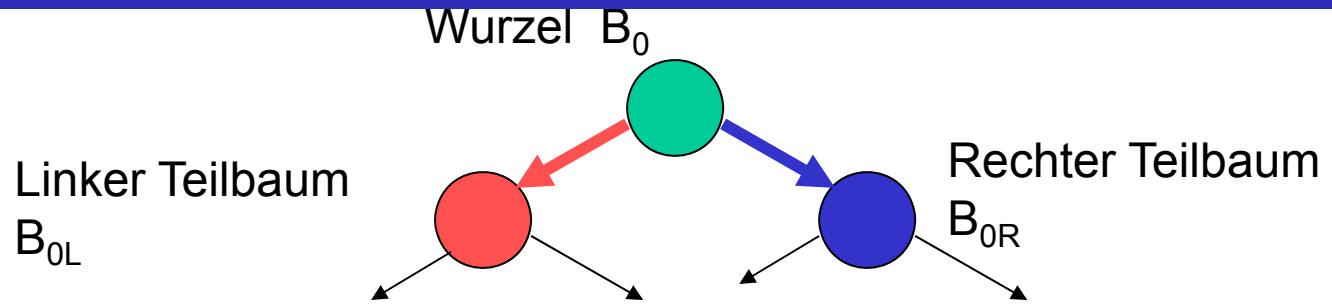
Viele Anwendungsalgorithmen basieren auf solchen nichtlinearen Strukturen.

Bei netzartigen Graphstrukturen müssen diese Algorithmen mit erschöpfendem Durchsuchen aller Möglichkeiten arbeiten. Das erfordert programmseitig spezielle Hilfen (globale stacks).

Bei Hierarchien (Bäume) reicht die einfache Rekursivität aus.
Bei vielen Sonderfällen in Hierarchien nutzt man die s.g. Teile-und-Herrsche-Algorithmen. Diese Algorithmen teilen den aktuellen Bearbeitungsraum in Teilbearbeitungsräume und wenden dann den gleichen Algorithmus rekursiv auf die Teile an (herrschen), solange bis eine weitere Teilung nicht mehr sinnvoll ist.

Beispiel: Binärbäume

Binärer Baum und Rekursion (1)



Binärbaum über Knotenmenge V

$B_0 = (W_0, B_{0L}, B_{0R}) \cup (W_0, \varnothing, \varnothing)$

$B_x = (W_x, B_{xL}, B_{xR}) \cup (W_x, \varnothing, \varnothing)$

mit $W_0, B_0, W_x, B_x \in V$ und $\varnothing$ als leeres Element

Indizes werden durch Aneinandereihung gebildet, z.B.

x=0L → xL = 0LL, x=0R → xL = 0RL usw.

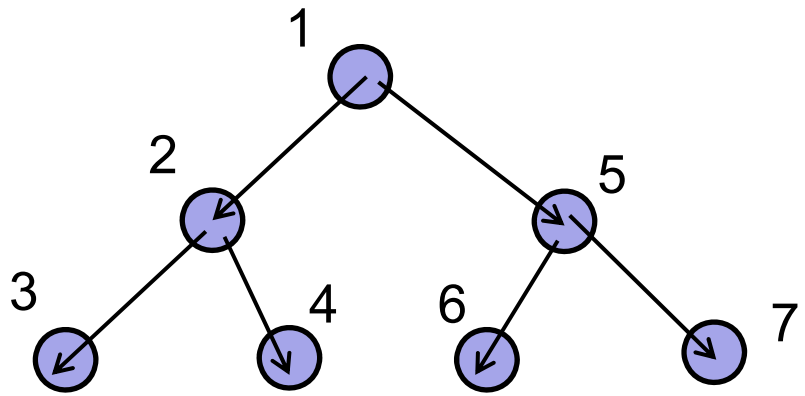
Ein Baum ist entweder ein einzelner Knoten oder ein als Wurzel dienender Knoten, der mit einer Menge von Bäumen verbunden ist. (beim Binärbaum mit zwei Teilbäumen verbunden)

Binärer Baum und Rekursion (2)

Verschiedene Strategien zum Traversieren des Baums

Preorder:

1. Besuche die Wurzel des Baumes
2. Besuche den linken Teilbaum
3. Besuche den rechten Teilbaum

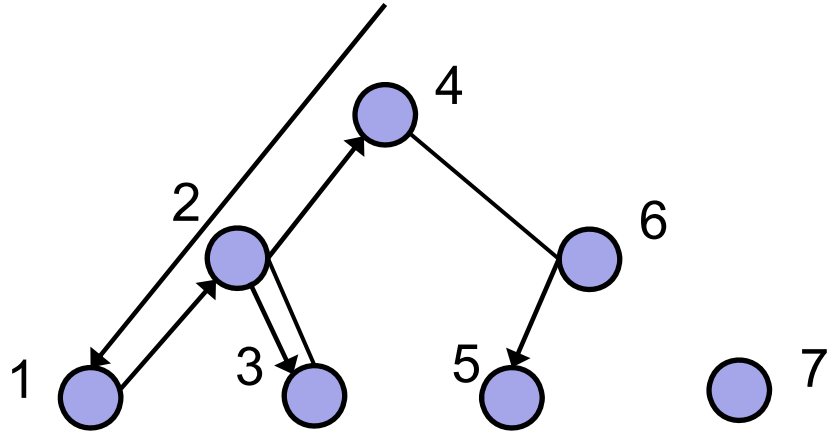


Binärer Baum und Rekursion (3)

Verschiedene Strategien zum Traversieren des Baums

Inorder (Symmetrische Strategie):

1. Besuche den linken Teilbaum
2. Besuche die Wurzel
3. Besuche den rechten Teilbaum

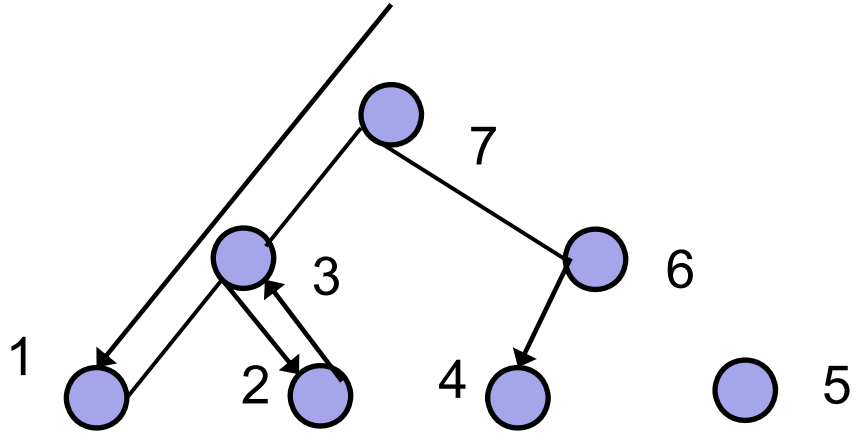


Binärer Baum und Rekursion (4)

Verschiedene Strategien zum Traversieren des Baums

Postorder:

1. Besuche den linken Teilbaum
2. Besuche den rechten Teilbaum
3. Besuche die Wurzel



Binärer Baum und Rekursion (5)

Strategien zum Traversieren des Baums (Fortsetzung)

Alle bisherigen Verfahren besuchen entweder tiefe Knoten oder links stehende Knoten zuerst.

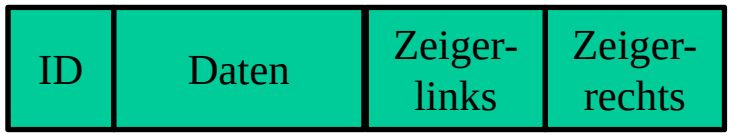
Bei Suchbäumen werden Lösungen u.U. erst spät gefunden.

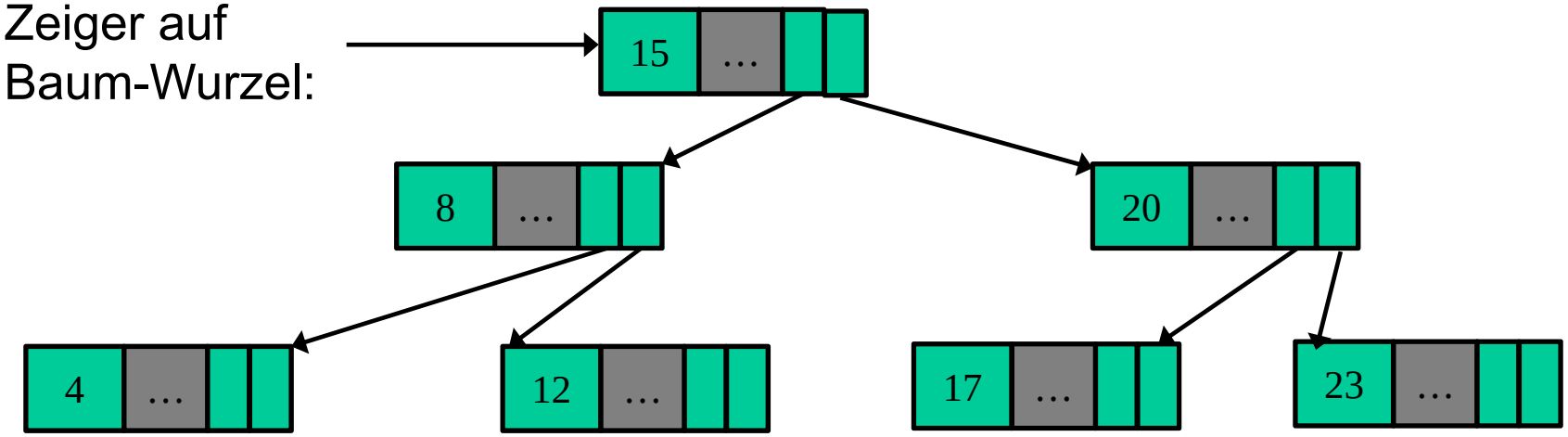
Level-Order-Traversierung:

- Besuche die Knoten “von links nach rechts“ innerhalb einer Ebene, danach die jeweils tiefere Ebene.
- Diese Reihenfolge wird nicht durch Zeiger in innerhalb der Baumstruktur unterstützt
- Diese Reihenfolge wird auch nicht durch Rekursion unterstützt

Binärer Baum mit sortierten Daten (1)

Baum-Elemente:





NULL-Zeiger, wenn Nachfolge-Elemente nicht vorhanden:

Binärer Baum mit sortierten Daten (2)

Suchen eines Elements mit ID=x im sortierten Binärbaum:

Knoten = Wurzel
Aufsuchen Knoten:

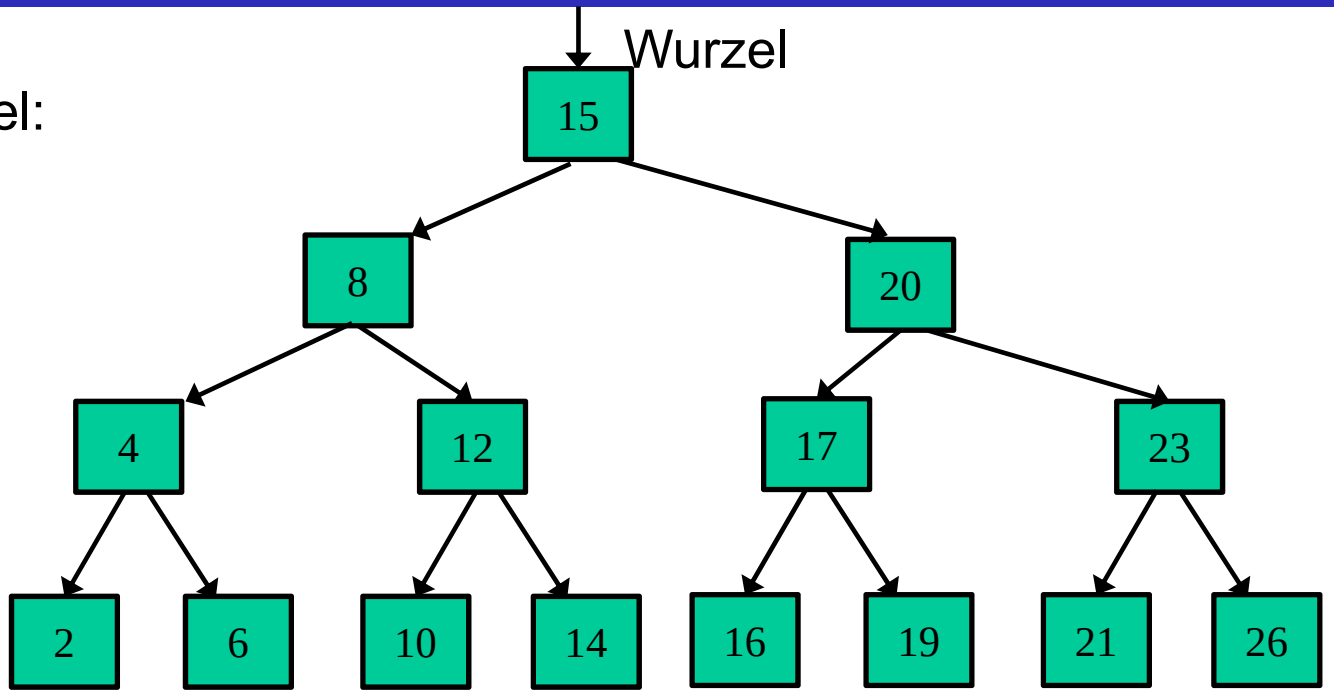
- falls ID==x dann gefunden, Ende
- falls x<ID: Verfolge Zeiger-links
- falls x>ID: Verfolge Zeiger-rechts

Nach „Verfolge“ wird der jeweilige Knoten nach o.g. Regel besucht, solange bis

- Knoten mit ID gefunden
- oder ein Verfolgen auf den NULL-Zeiger trifft. Dann ist das gesuchte Element im Baum nicht vorhanden.

Bewertung Binärbaum

Beispiel:



	im Bild	allgemein	Berechnung
Anzahl Ebenen	4	e	Konstruktionsparameter
Anzahl Elemente	15	$n = 2^e-1$	
Schritte zum Finden eines Elements	4 (inkl. Zugriff auf Wurzel)	s=e	$s = \lceil \log_2 n \rceil$