

# Baumbasierte Strukturen

# Baumbasierte Struktur / Organisation als Binärbaum

- Haufendateien oder sortierte Dateien → nützlich für statische Dateien
- Dateien organisiert als Binärbaum
  - Effizientes Einfügen und Löschen von Datensätzen
  - Benutzt binäre Suche
- Meist verwendete Indexstrukturen: B-, B<sup>+</sup>- und B<sup>\*</sup>-Bäume
- Speicherstruktur für eine Datei organisiert als Binärbaum:
  - Eine Menge von Knoten; Zeiger zu der Wurzel
  - Eine Liste von freien Knoten

# Organisation als Binärbaum

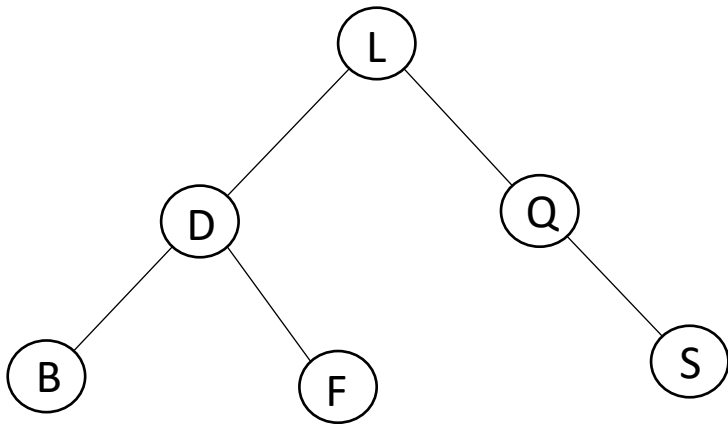
- Ein Eintrag in einem Knoten:

$S_i$	$D_i$	$V_{i-1}$	$V_i$
-------	-------	-----------	-------

- $S_i$  – Suchschlüssel
- $D_i$  – Datensatz oder Datensatzindetifikator
- $V_{i-1}$  – Verweis auf Knoten, die kleinere Schlüsselwerte enthalten
- $V_i$  – Verweis auf Knoten, die größere Schlüsselwerte enthalten

# Organisation als Binärbaum

- Wurzel – Zeiger zu dem Wurzel
- Frei – Zeiger zu dem Listenkopf der freien Knoten
- Binärbaum:



Wurzel = 1 →

Frei = 3 →

1	L	Daten <sub>L</sub>	2	4
2	D	Daten <sub>D</sub>	8	7
3			-6	NULL
4	Q	Daten <sub>Q</sub>	NULL	5
5	S	Daten <sub>S</sub>	NULL	NULL
6			-9	NULL
7	F	Daten <sub>F</sub>	NULL	NULL
8	B	Daten <sub>B</sub>	NULL	NULL
9			NULL	NULL

# Einfügen und Löschen von Datensätzen

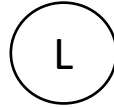
- Einen Datensatz einfügen:
  - Finde die Position wo der neue Datensatz eingefügt werden sollten
  - Speichere den Datensatz in einem freien Knoten
  - Füge die Zeiger für den neuen Knoten ein
- Einen Datensatz löschen
  - Suche den Datensatz
  - 3 Fälle:
    - Keine Kinder → Elterns Zeiger = NULL
    - 1 Kind → verbinde das Kind zu dem Elternteil
    - 2 Kinder → ersetze den Knoten mit dem nächsten Nachbarwert
  - Füge den Knoten, der frei geworden ist, zu der Liste mit freien Knoten ein

# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C

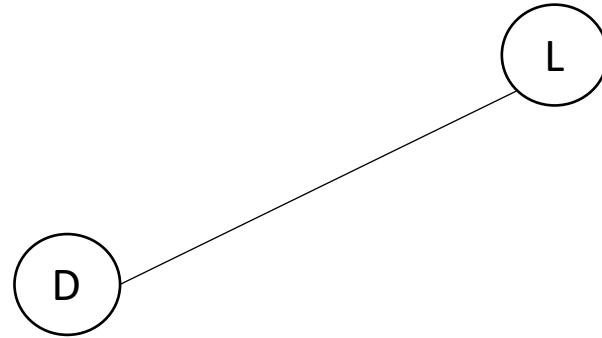
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



# Einfüge Anomalie in dem Binärbaum

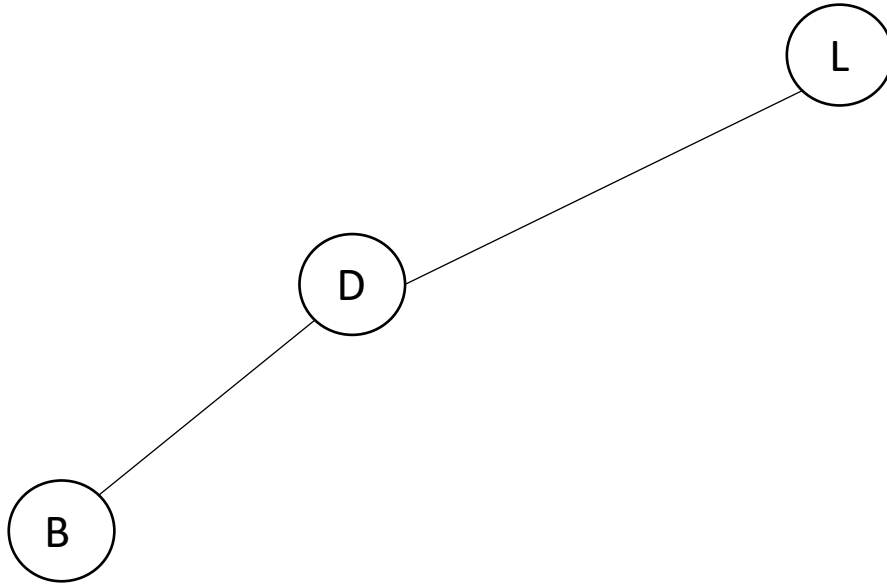
- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C





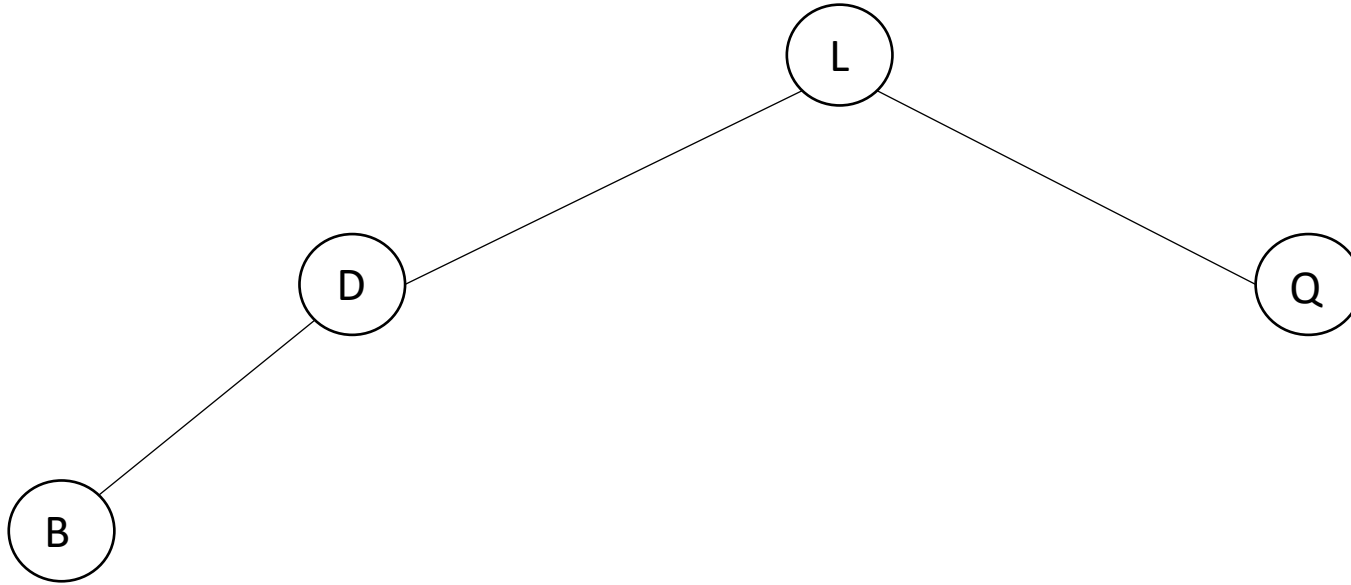
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



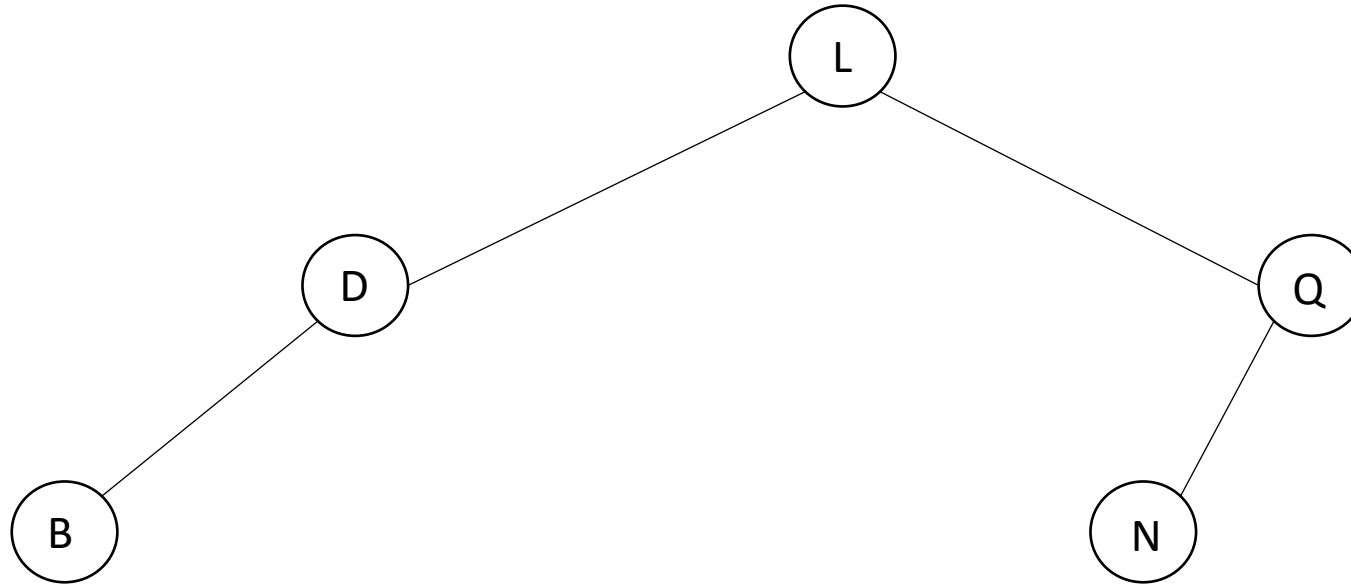
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



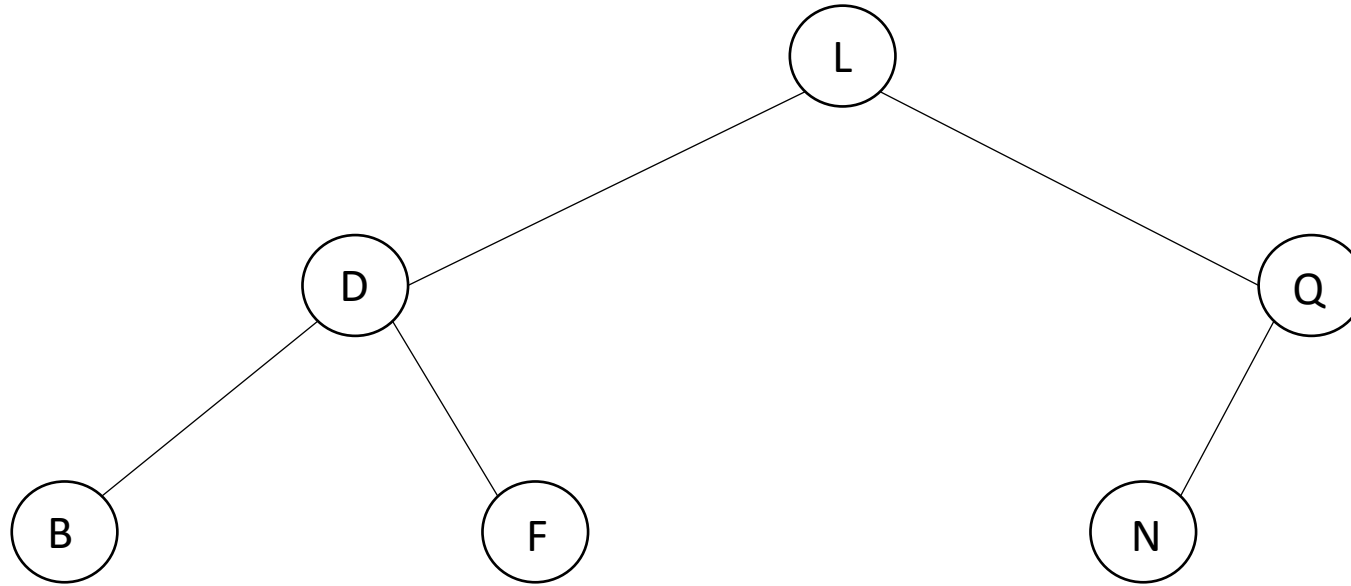
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



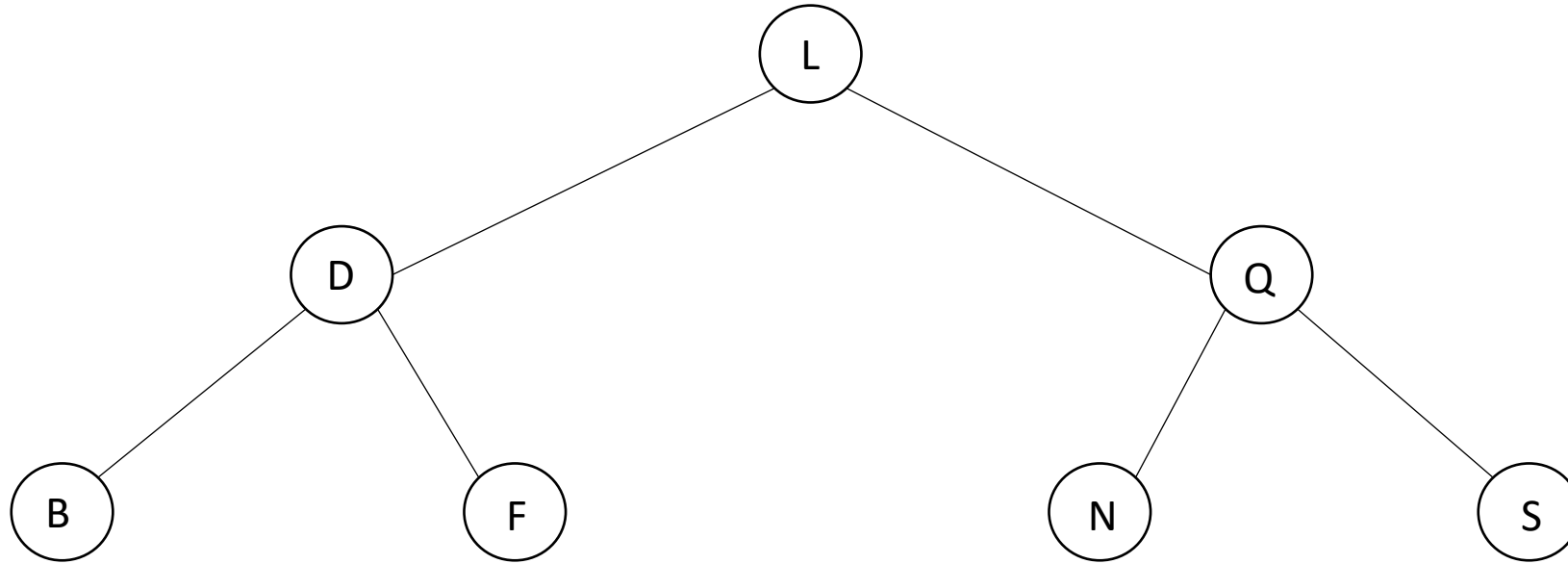
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



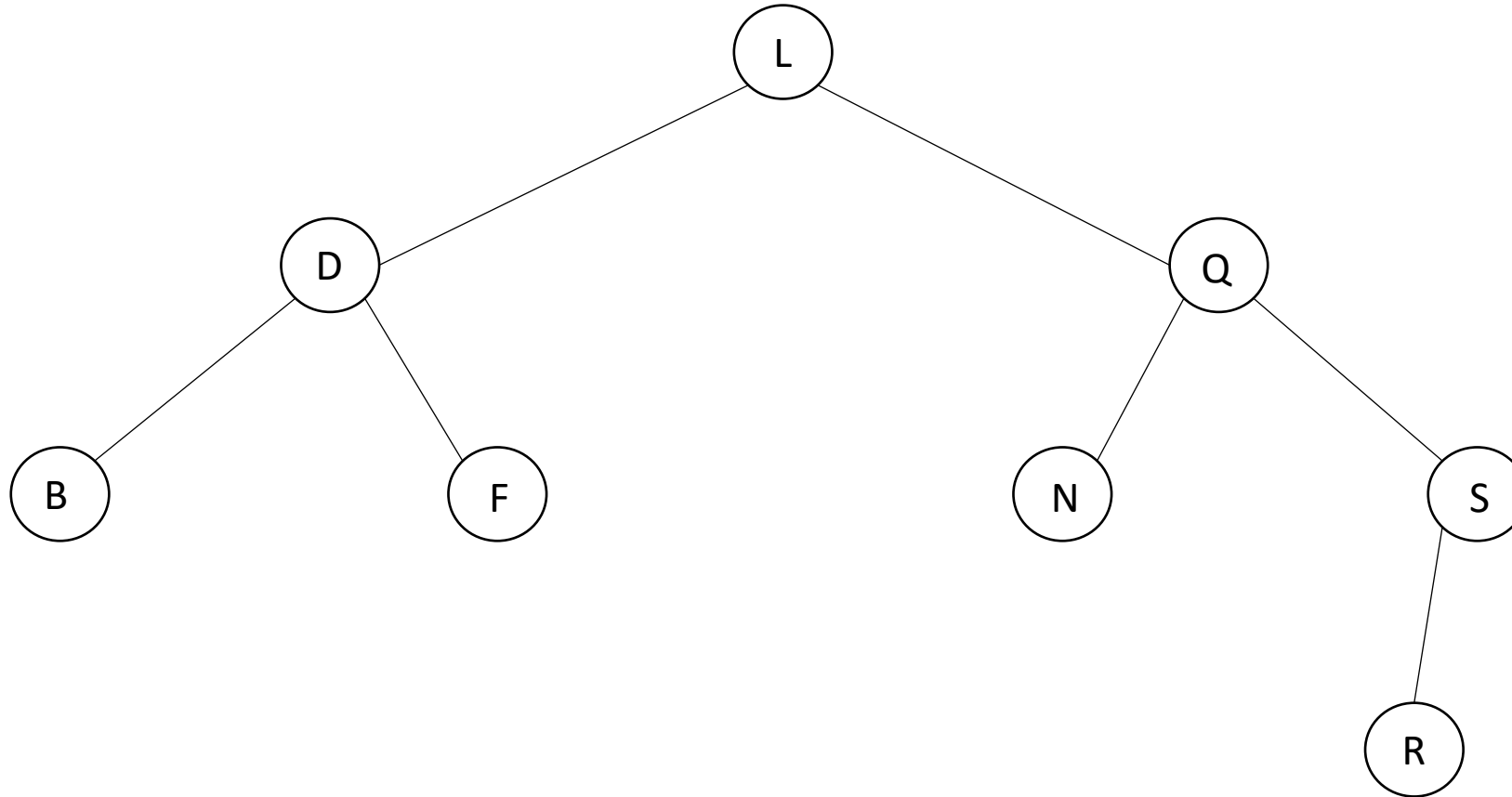
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



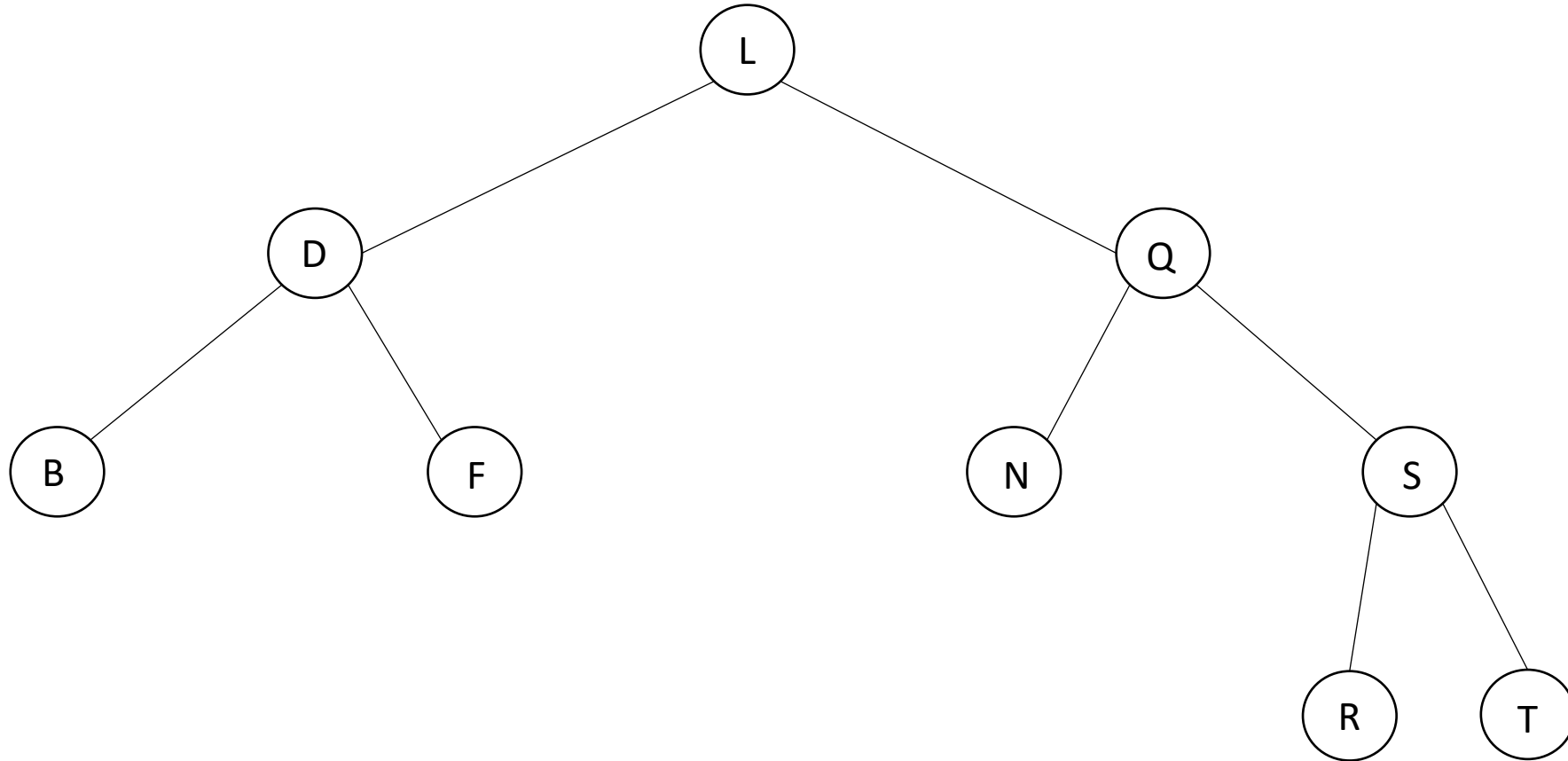
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



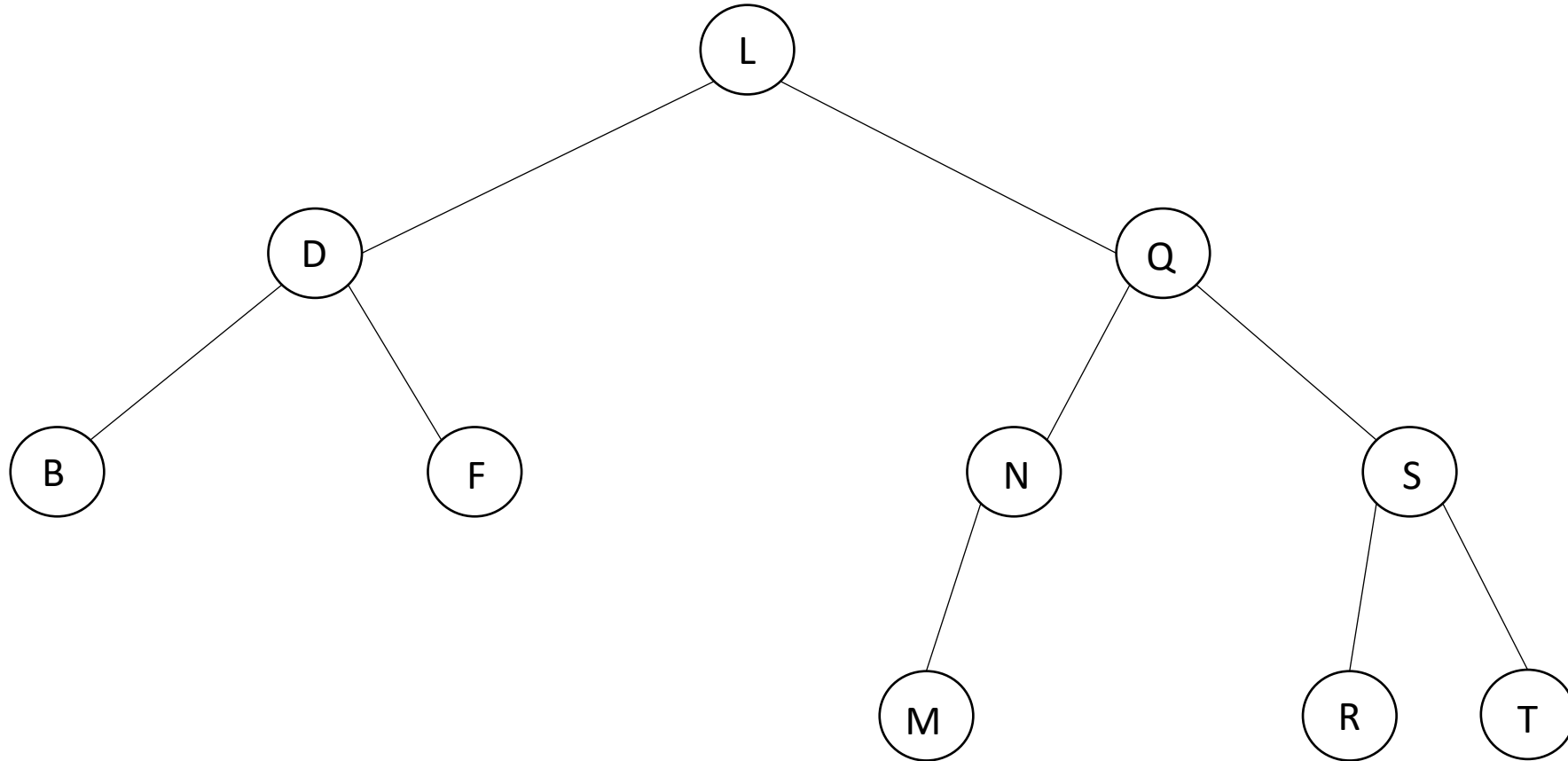
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



# Einfüge Anomalie in dem Binärbaum

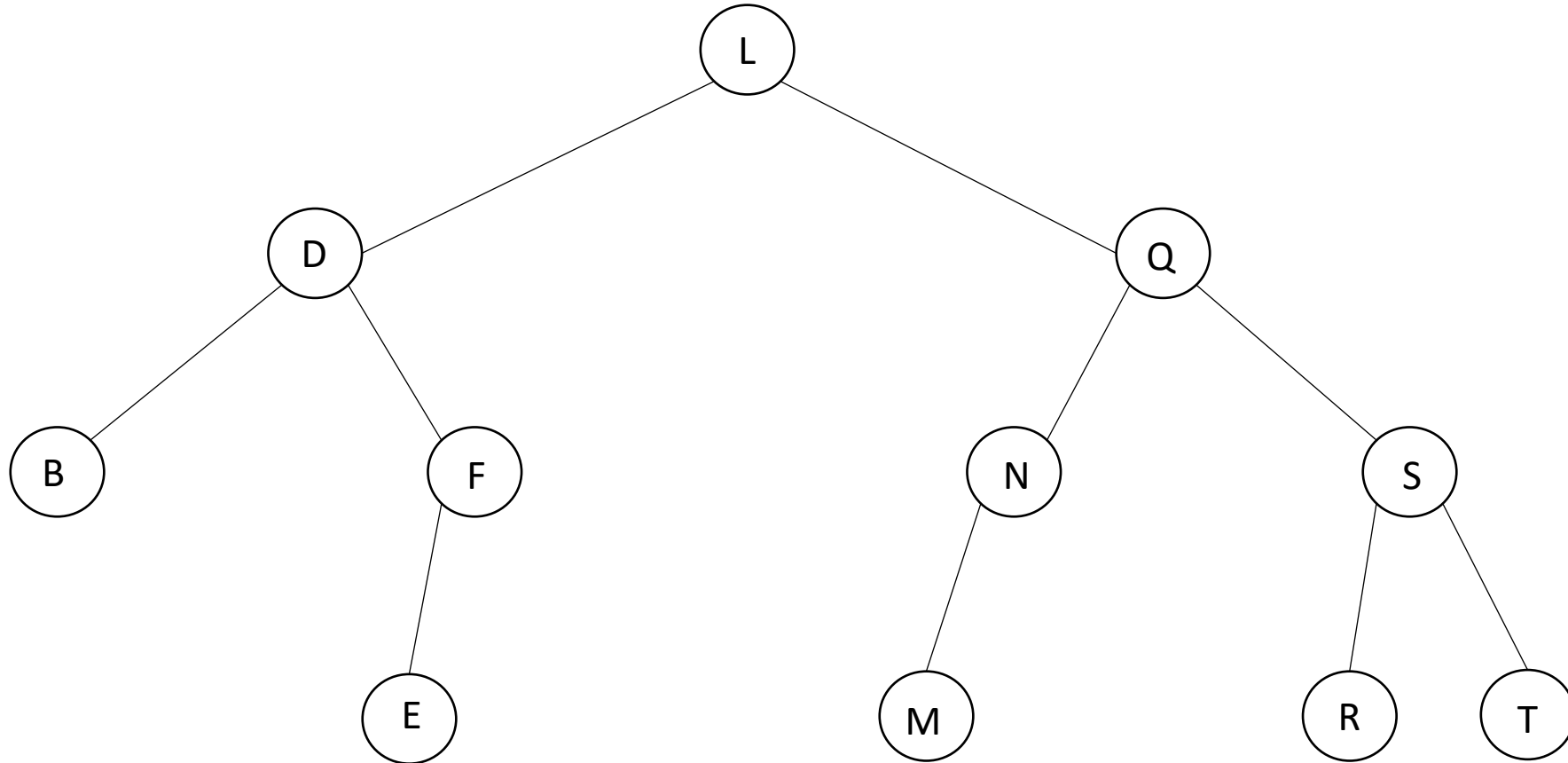
- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C





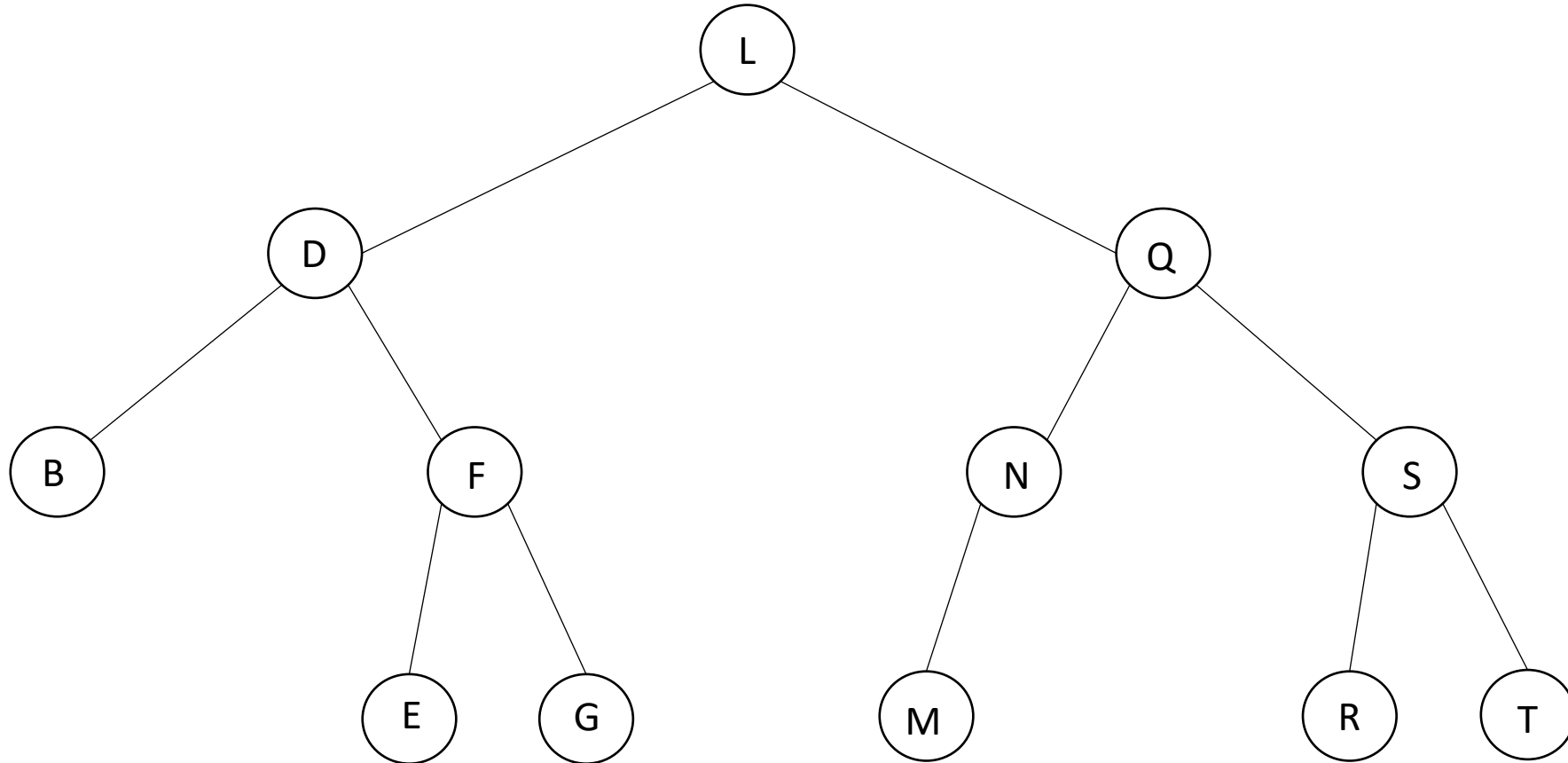
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



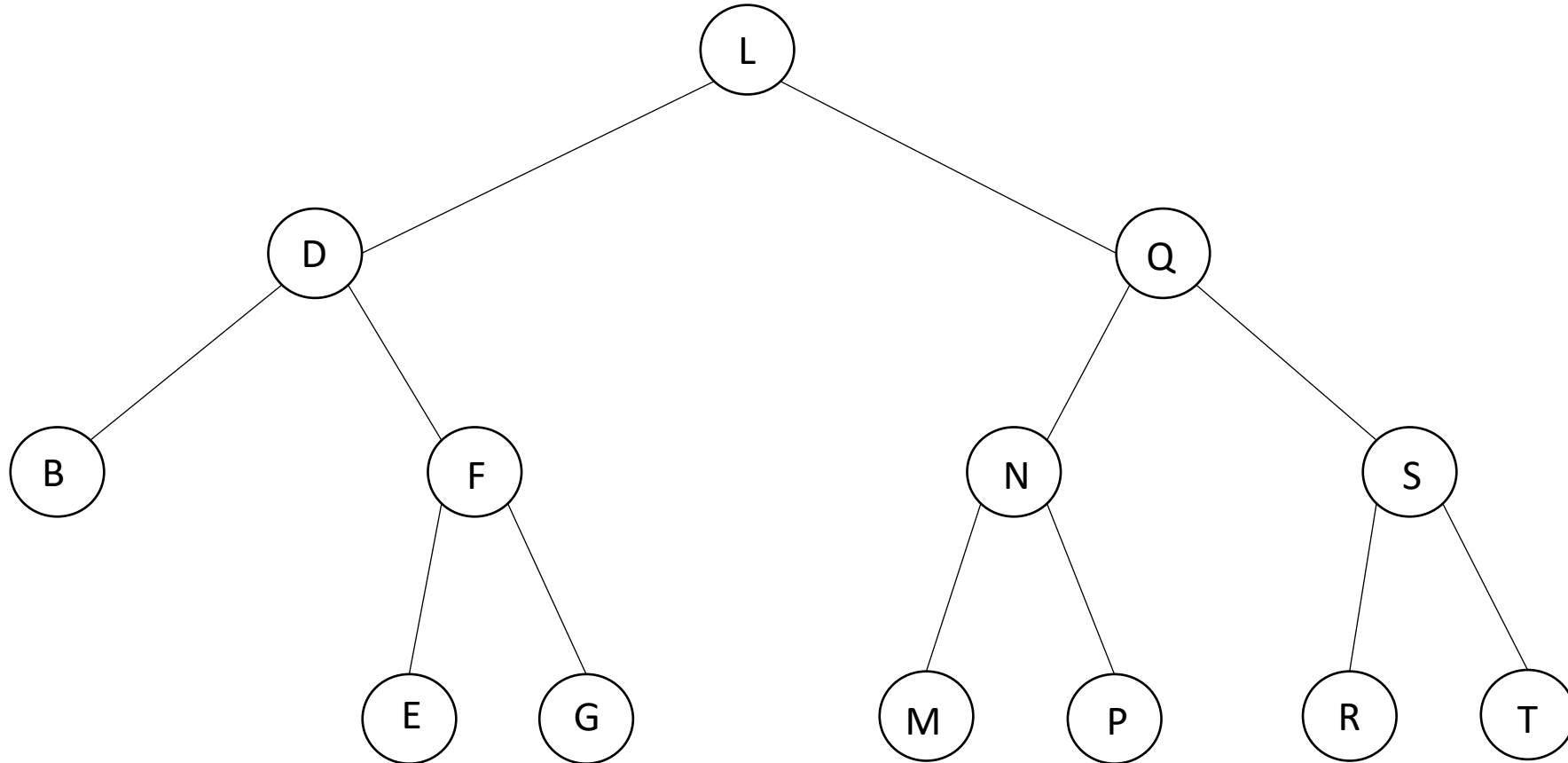
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



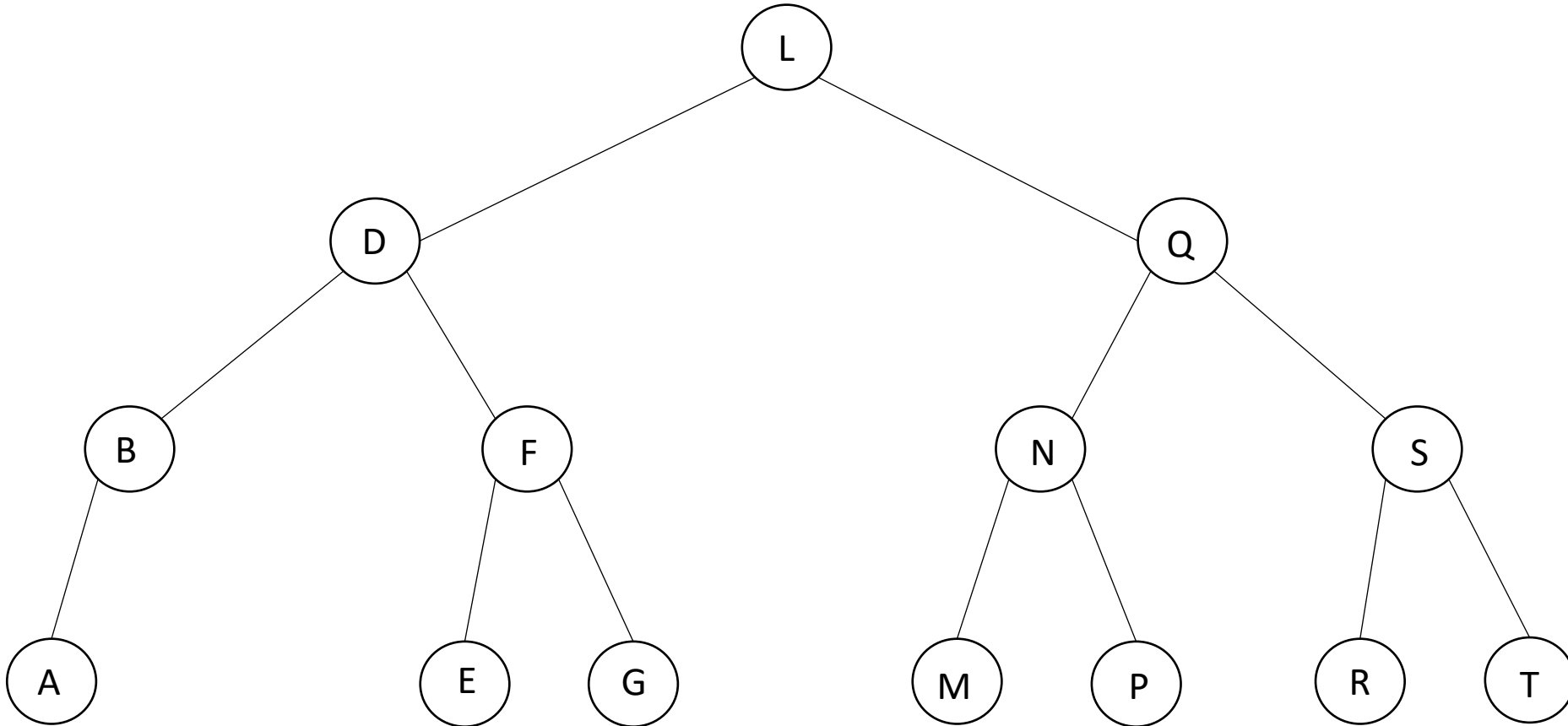
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



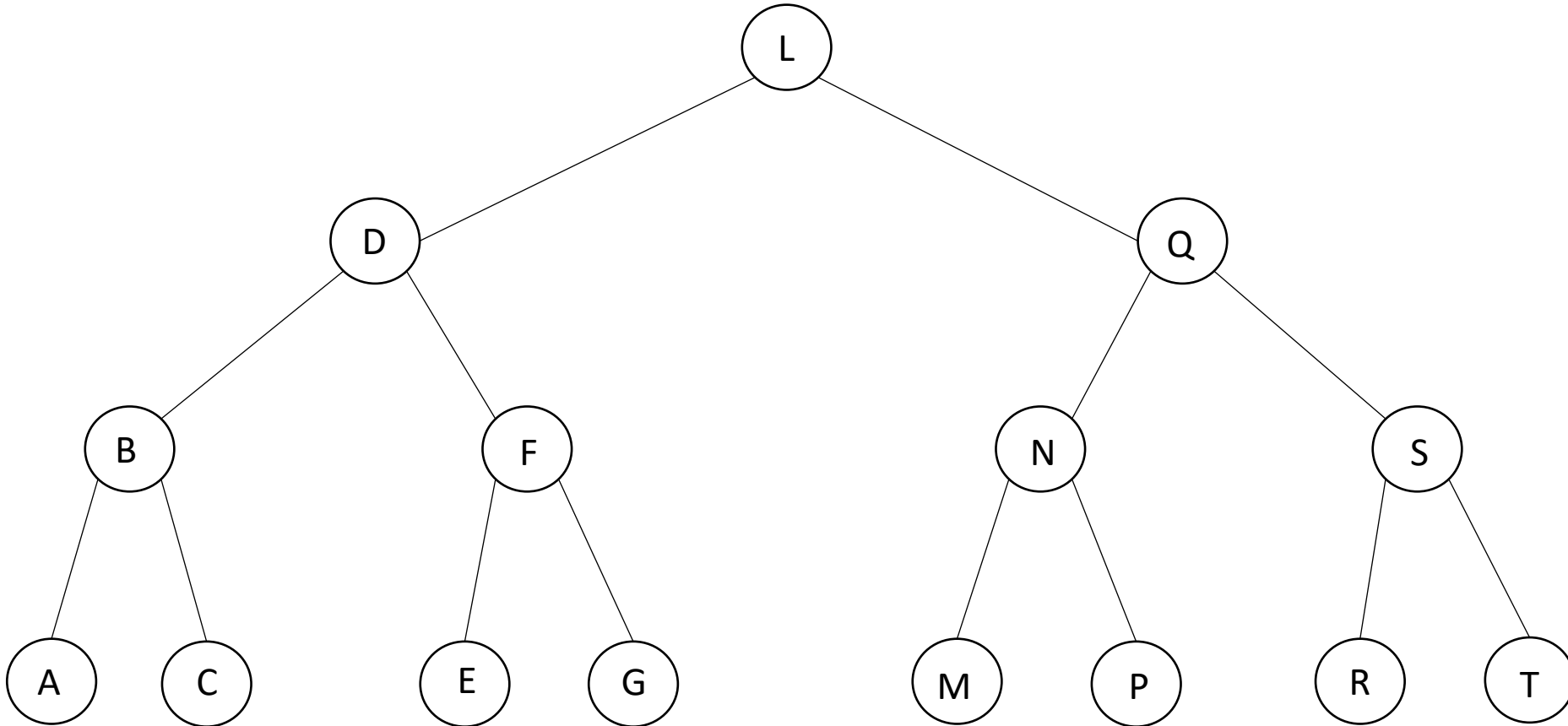
# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



# Einfüge Anomalie in dem Binärbaum

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C

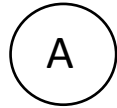


# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L

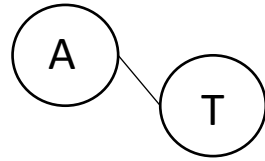
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



# Einfüge Anomalie in dem Binärbaum

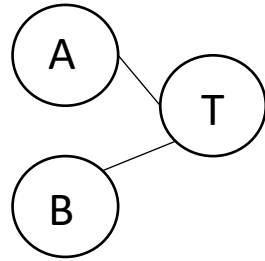
- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L





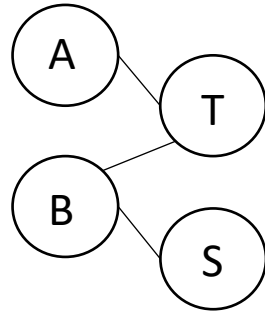
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



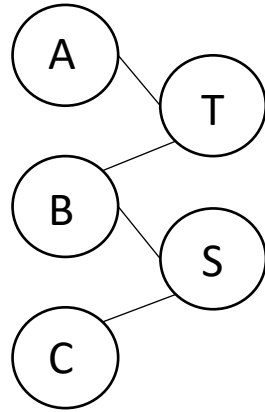
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



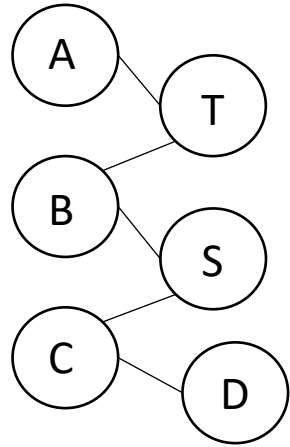
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



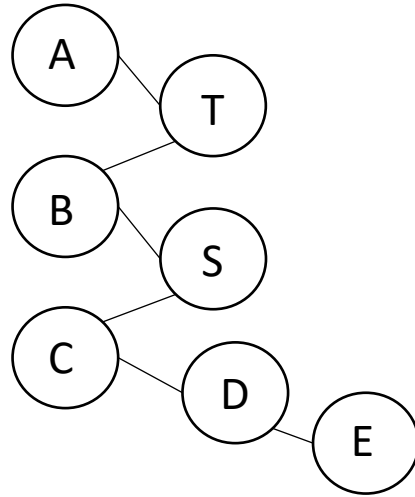
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



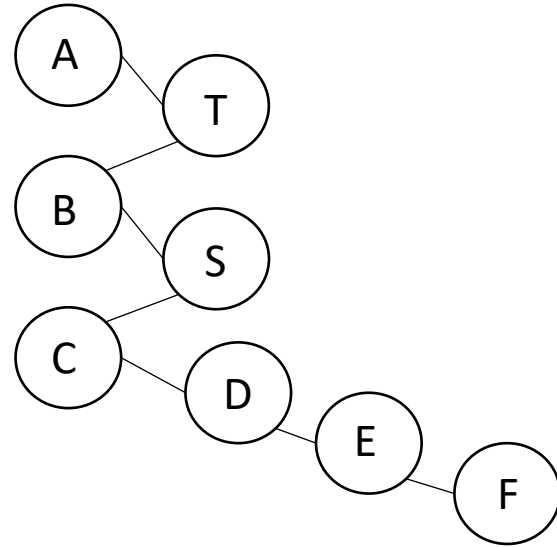
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



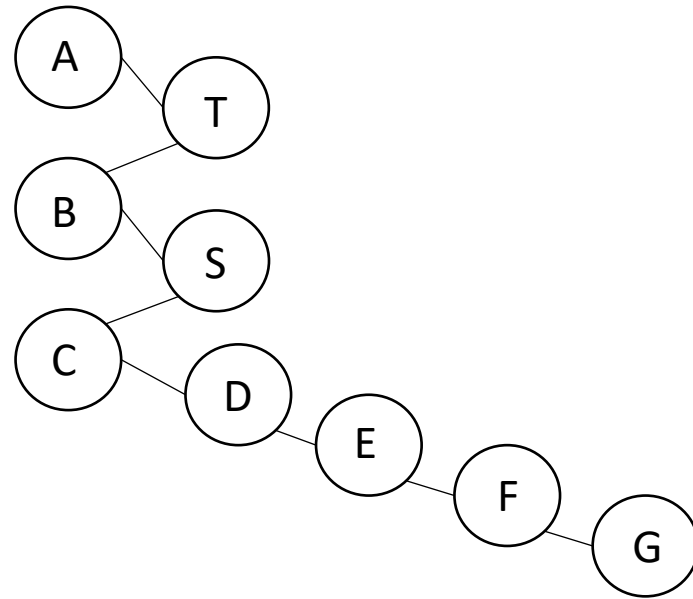
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



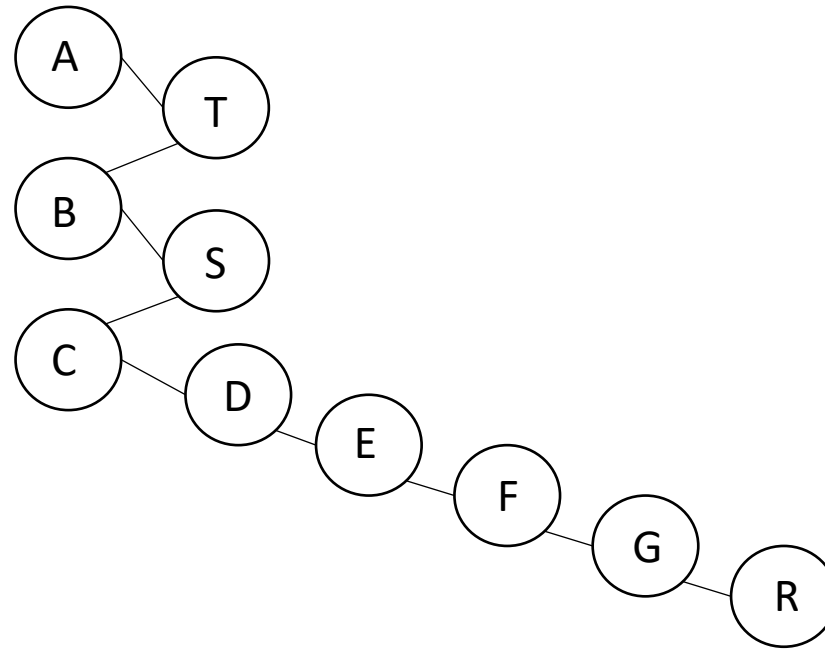
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



# Einfüge Anomalie in dem Binärbaum

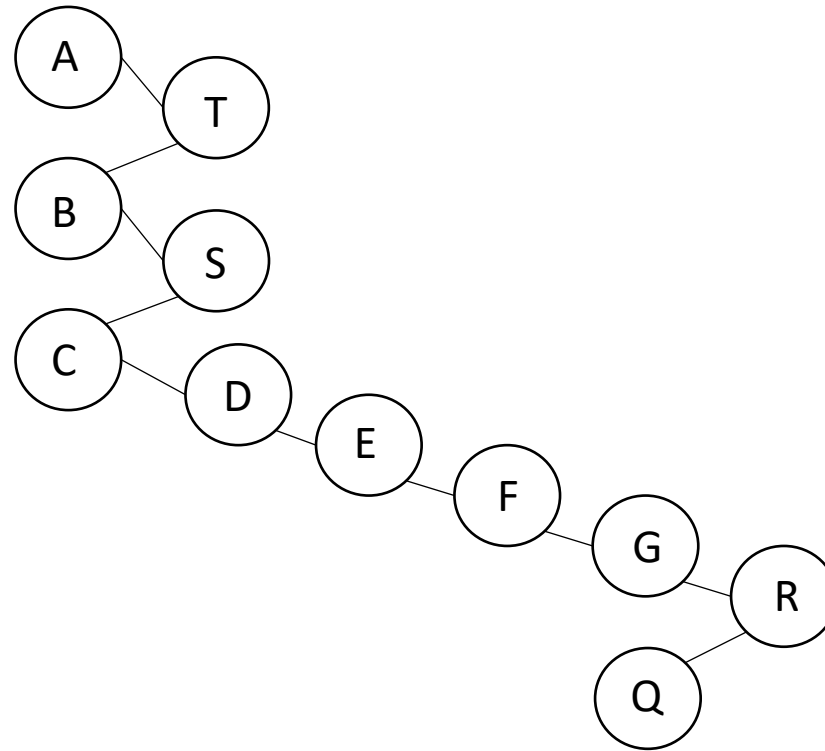
- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L





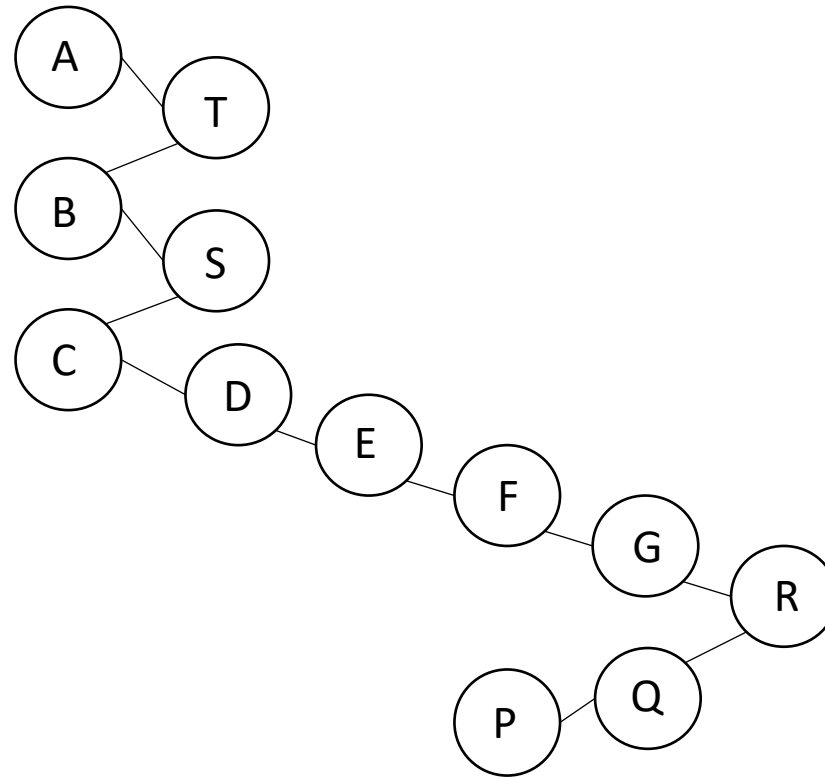
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



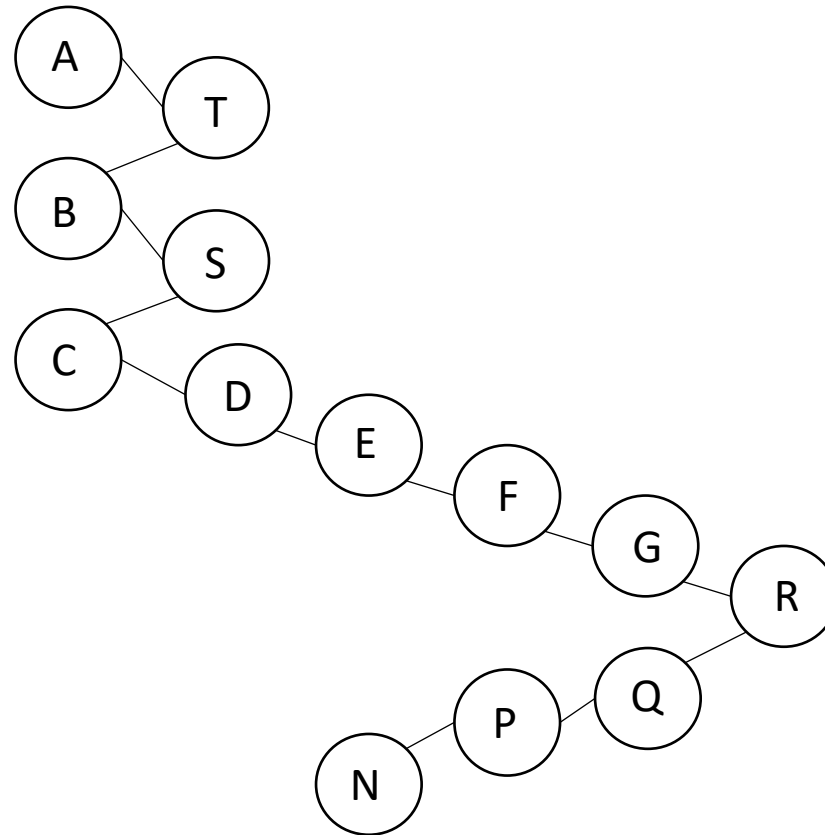
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



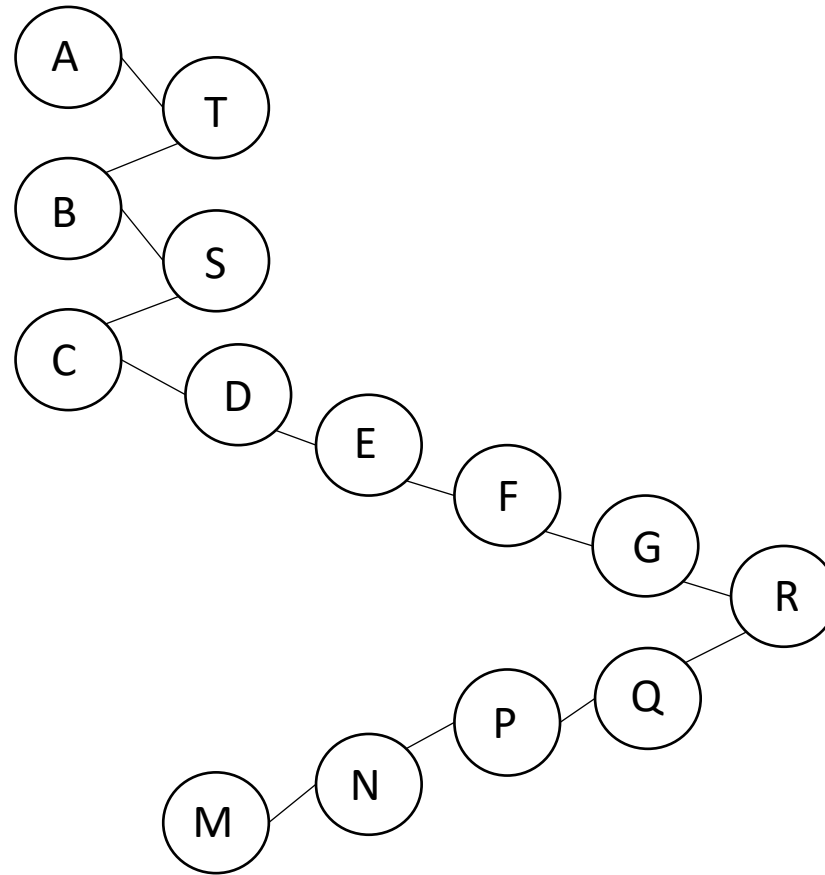
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



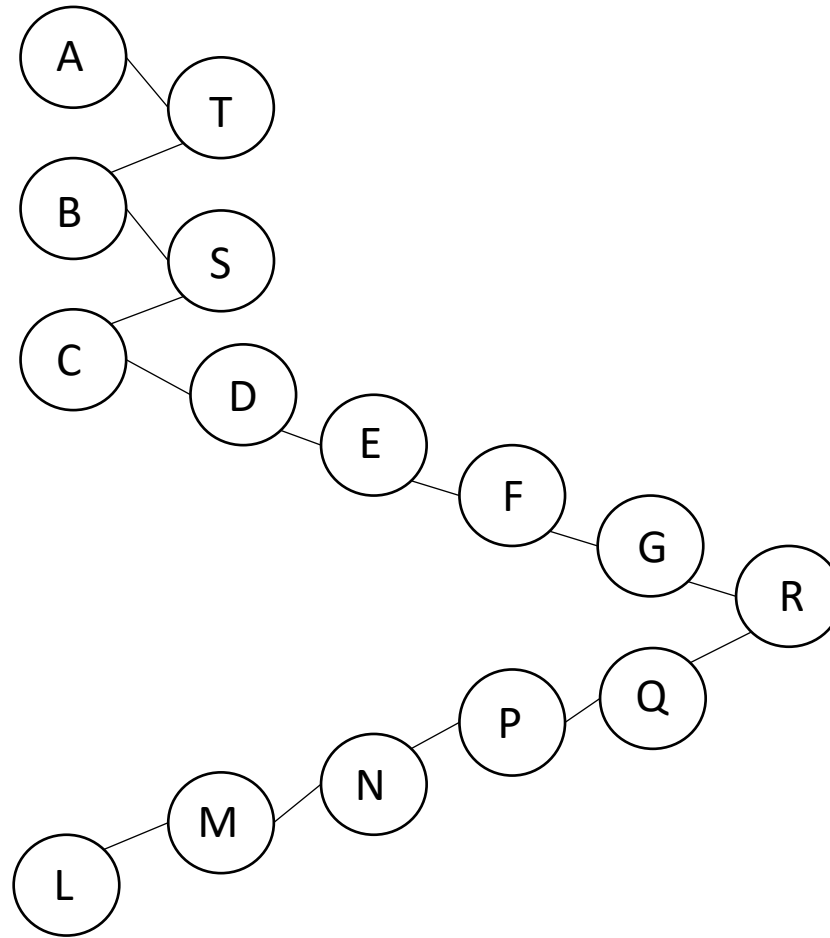
# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



# Einfüge Anomalie in dem Binärbaum

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



# Optimale vs. Balancierte Binärbäume

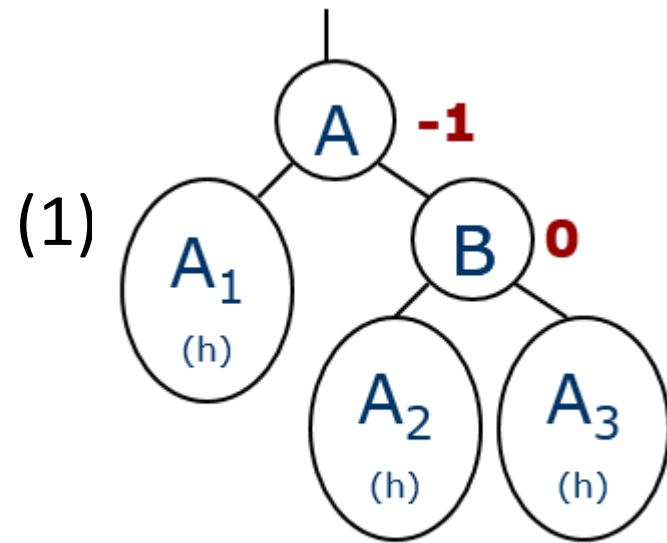
- Optimale Binärbäume:
  - Die Blätter befinden sich auf höchstens Level 2
  - Erhaltung ist aufwendig
- Balancierte Binärbäume:
  - für einen Knoten können die Höhen der zwei Teilbäume mit höchstens 1 voneinander abweichen (Höhe – Länge des längsten Pfades von der Wurzel zu den Blättern)
  - Wenigere Operationen um den Binärbaum balanciert zu behalten
  - Es gibt 6 Fälle wenn der Baum unbalanciert wird nach einem Einfügen

# Erhaltung der Balancierten Binärbäumen

(1)

(2)

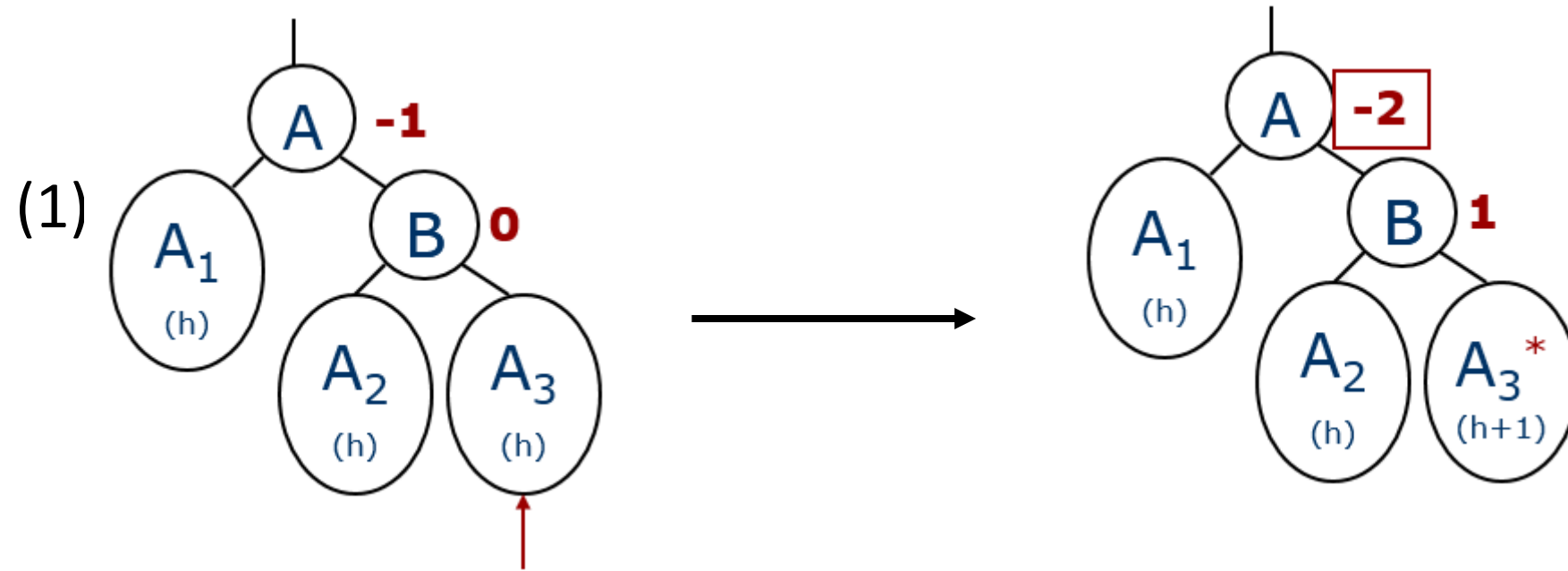
# Erhaltung der Balancierten Binärbäumen



(2)

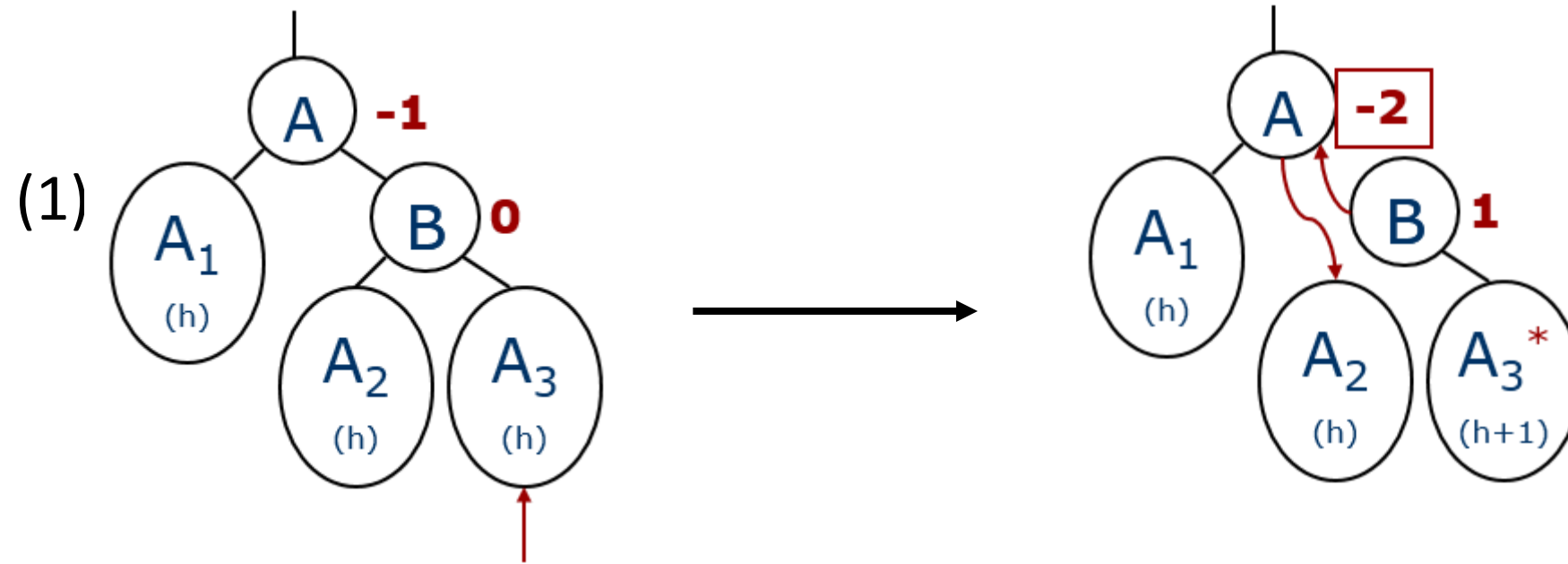


# Erhaltung der Balancierten Binärbäumen



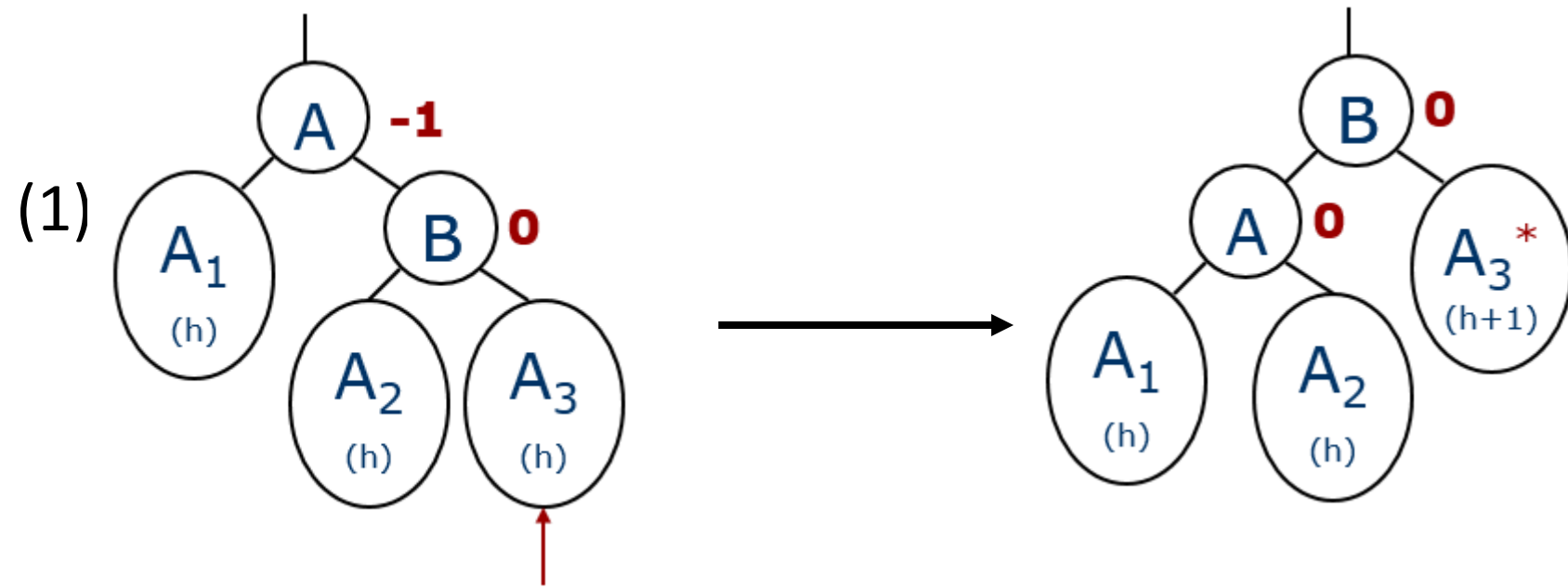
(2)

# Erhaltung der Balancierten Binärbäumen



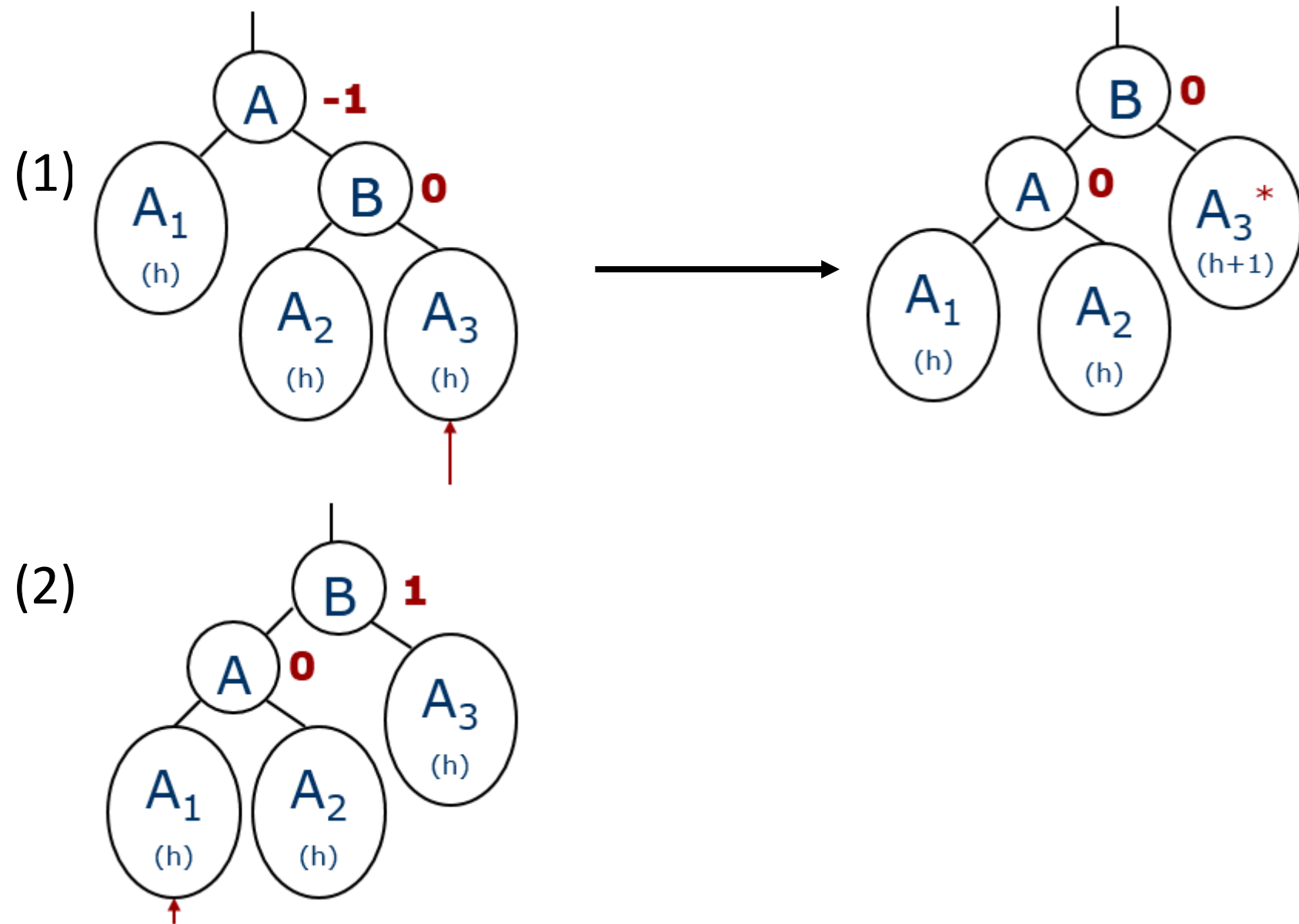
(2)

# Erhaltung der Balancierten Binärbäumen

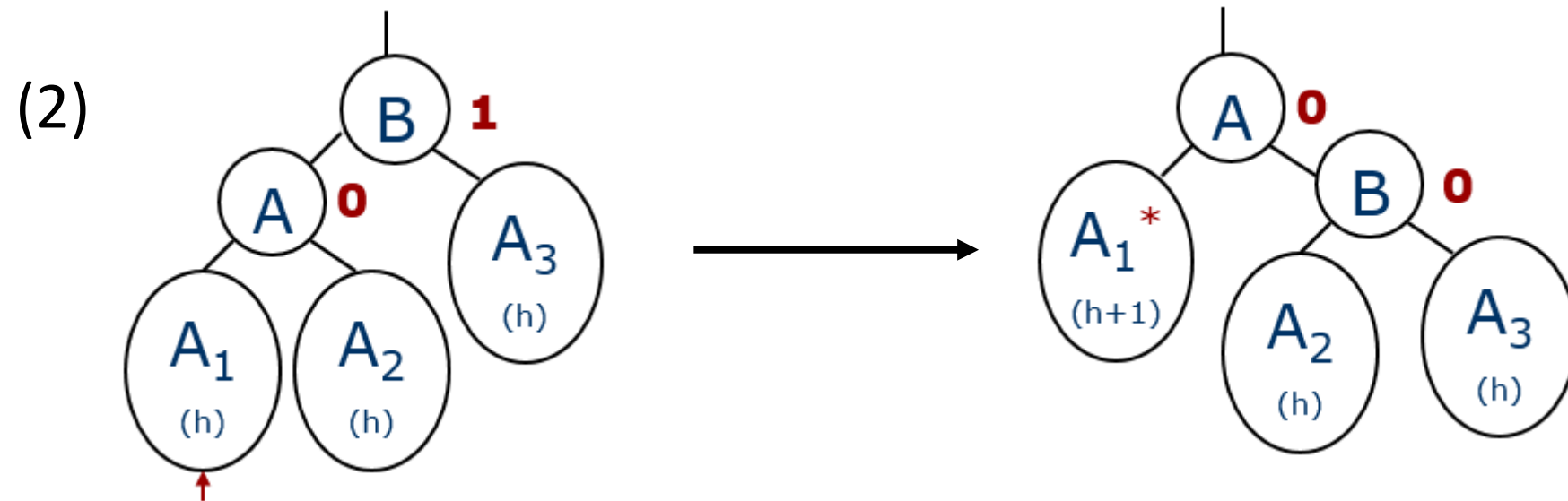
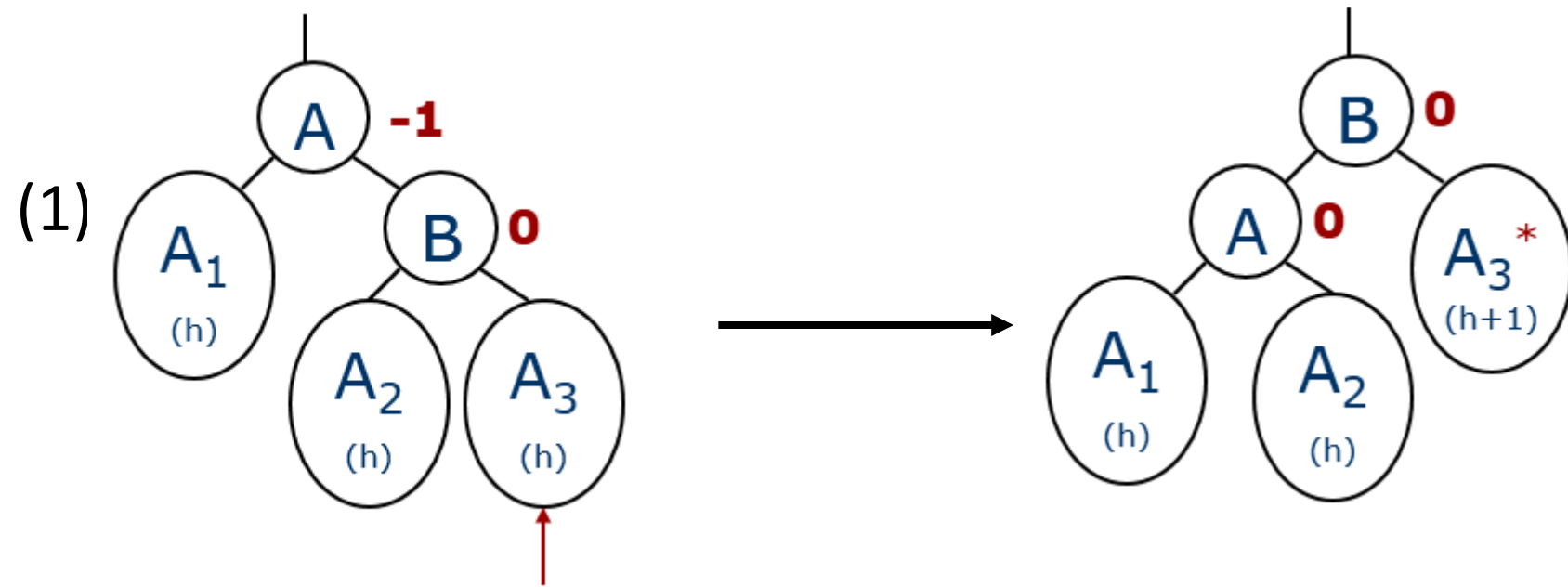


(2)

# Erhaltung der Balancierten Binärbäumen



# Erhaltung der Balancierten Binärbäumen



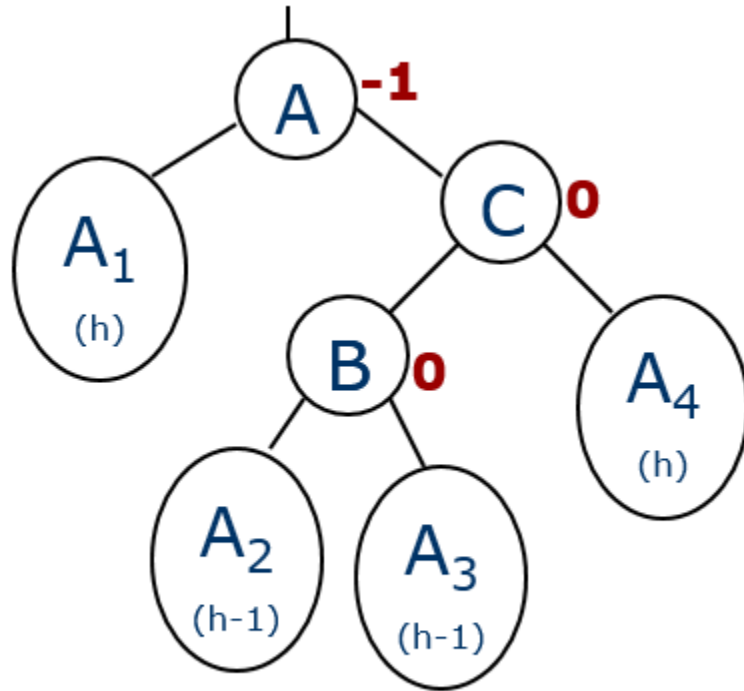
# Erhaltung der Balancierten Binärbäumen

(3)

(4)

# Erhaltung der Balancierten Binärbäumen

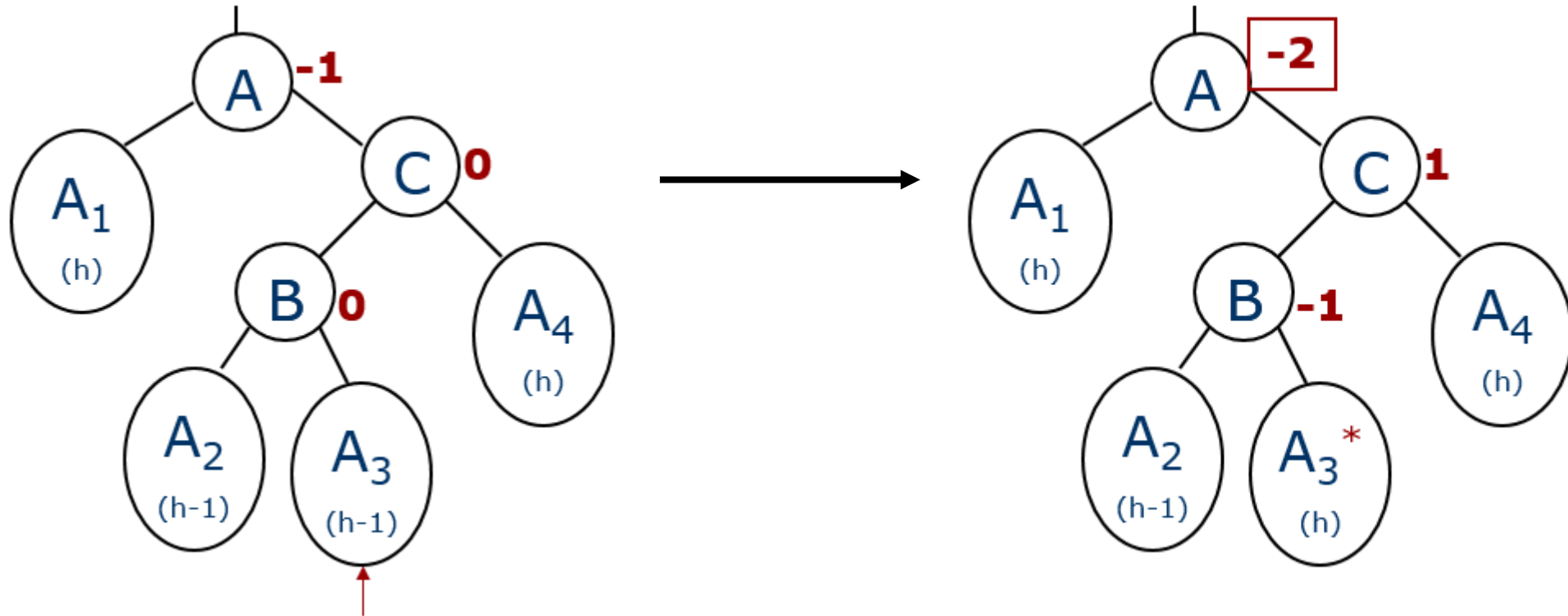
(3)



(4)

# Erhaltung der Balancierten Binärbäumen

(3)

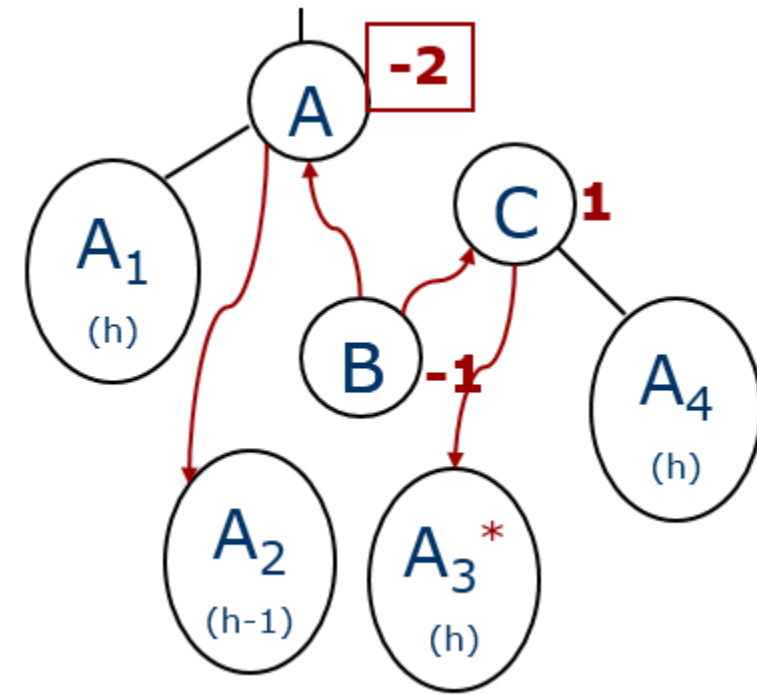
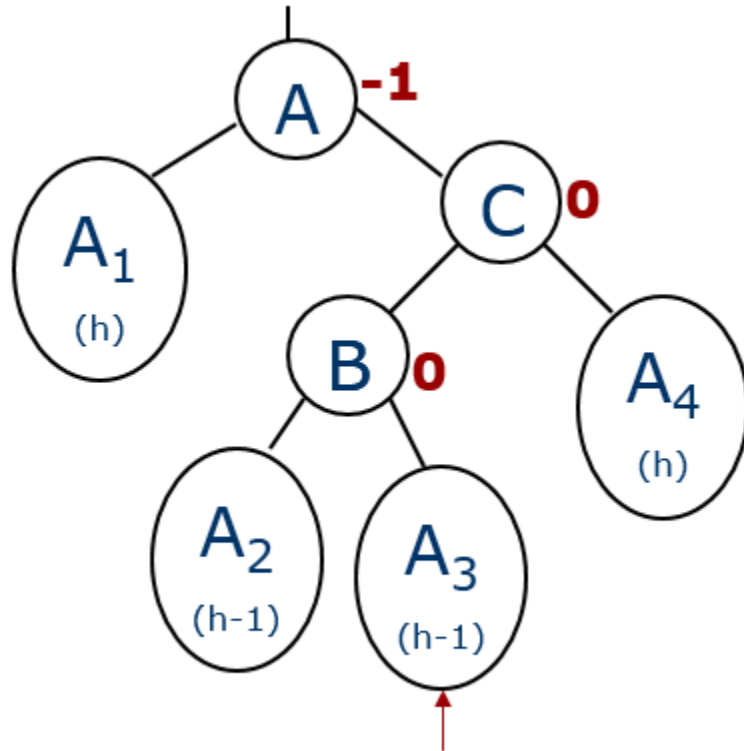


(4)



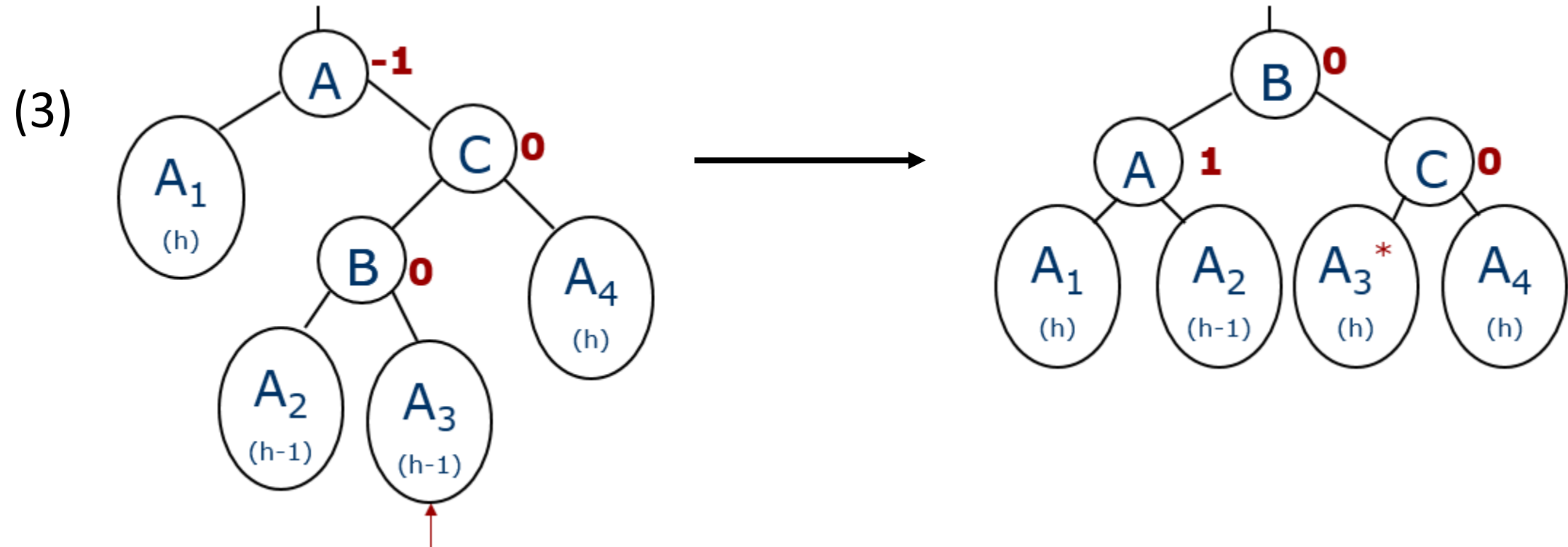
# Erhaltung der Balancierten Binärbäumen

(3)



(4)

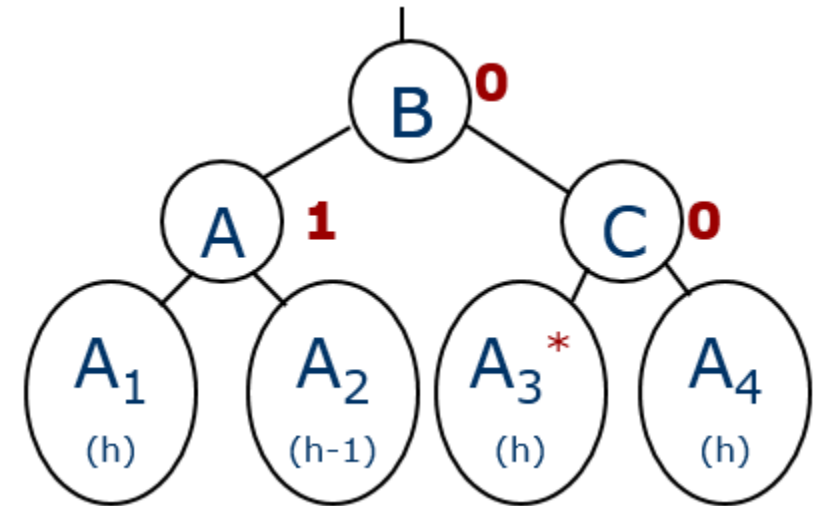
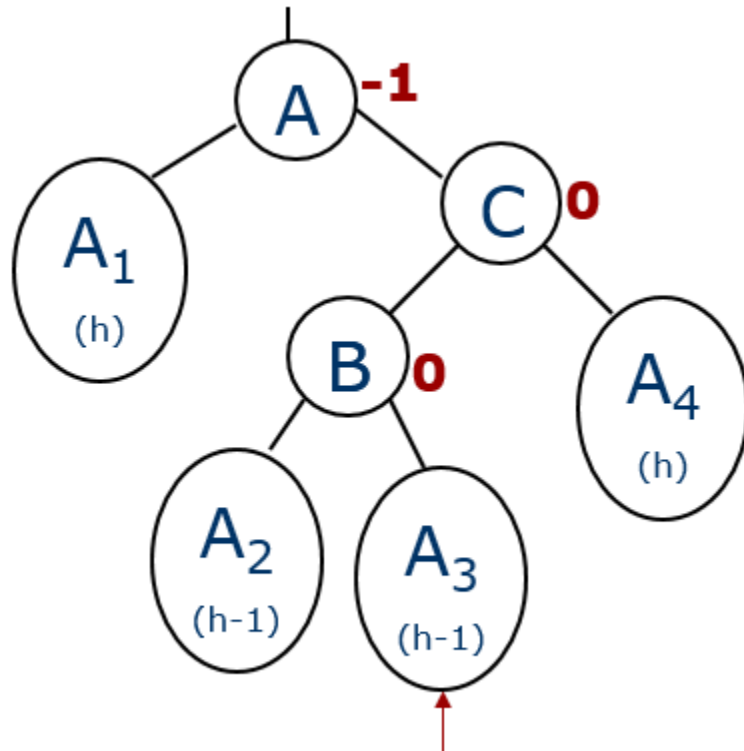
# Erhaltung der Balancierten Binärbäumen



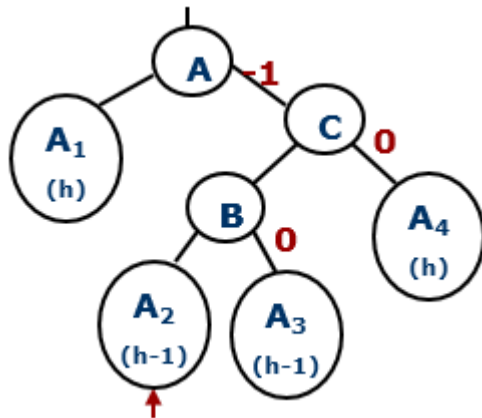
(4)

# Erhaltung der Balancierten Binärbäumen

(3)

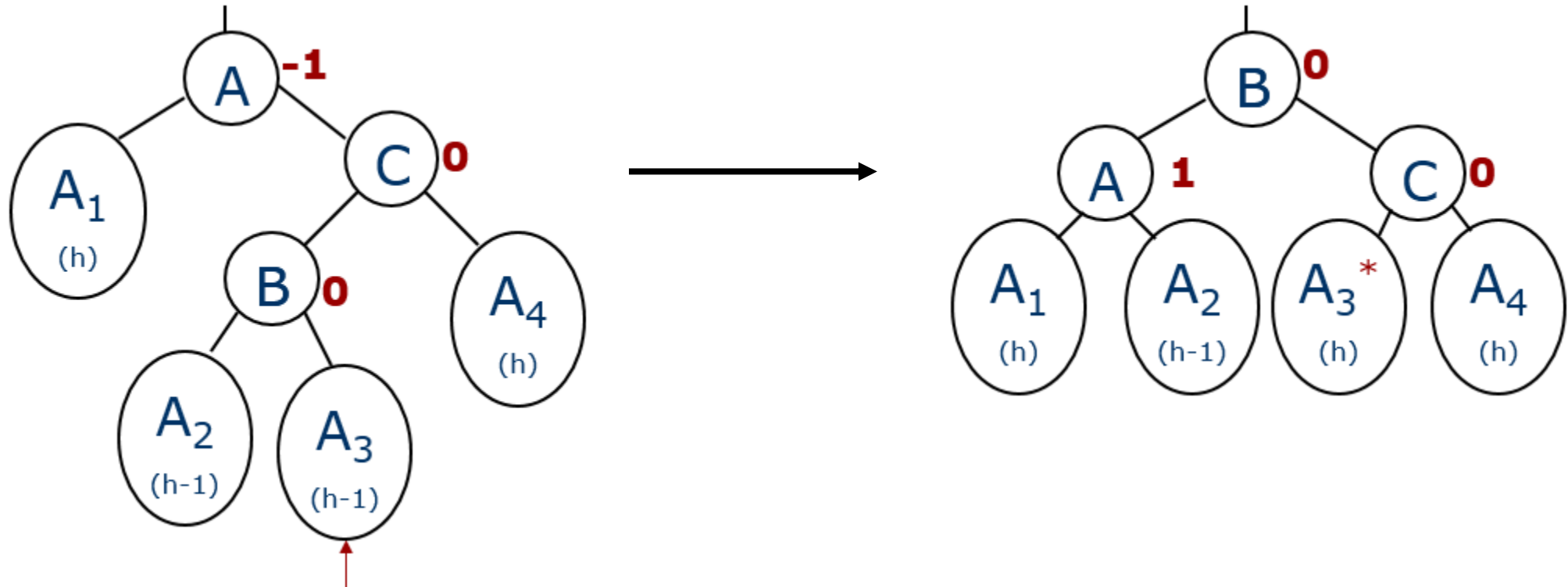


(4)

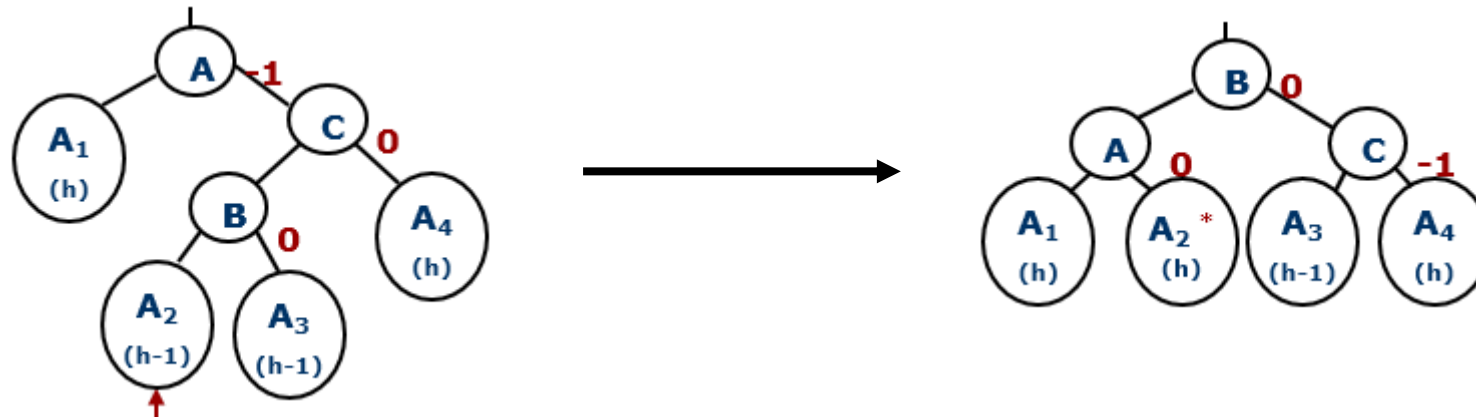


# Erhaltung der Balancierten Binärbäumen

(3)



(4)



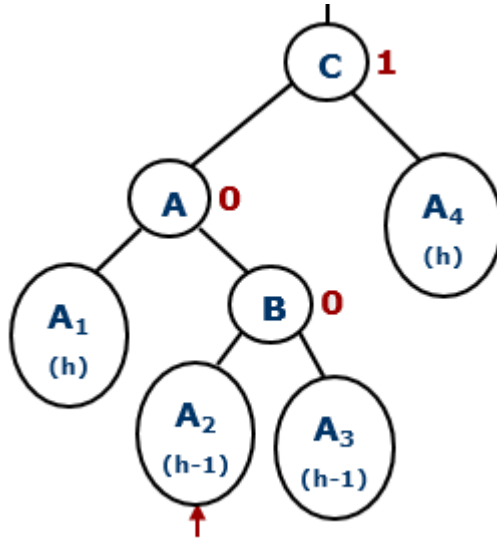
# Erhaltung der Balancierten Binärbäumen

(5)

(6)

# Erhaltung der Balancierten Binärbäumen

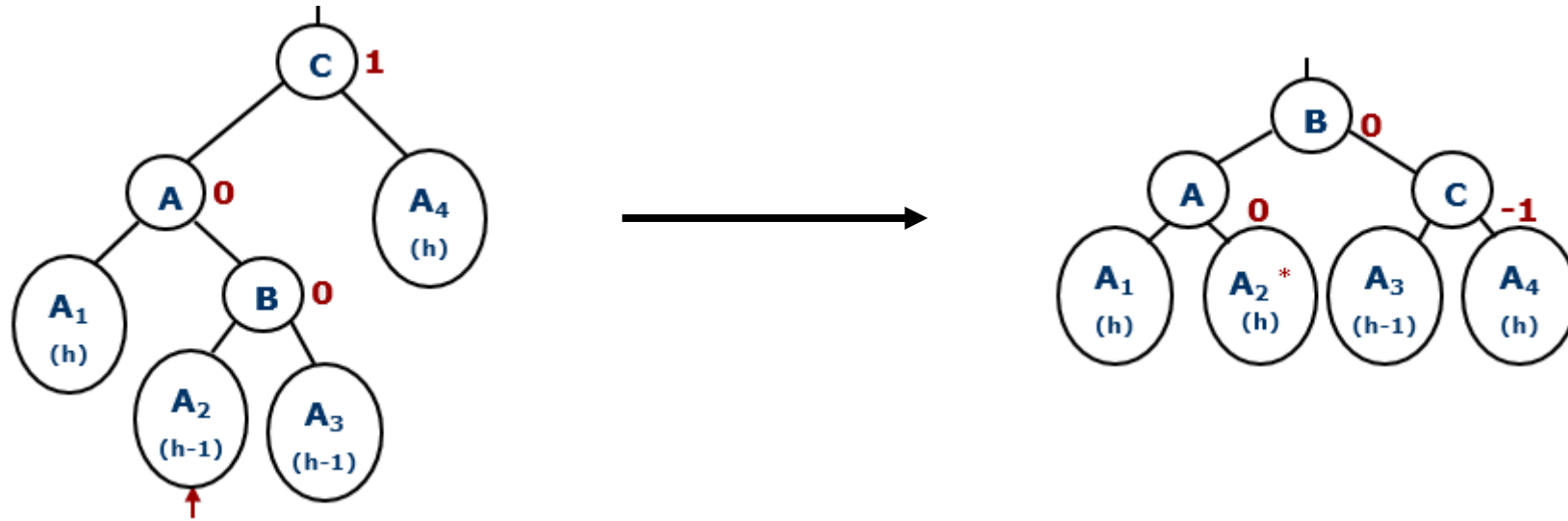
(5)



(6)

# Erhaltung der Balancierten Binärbäumen

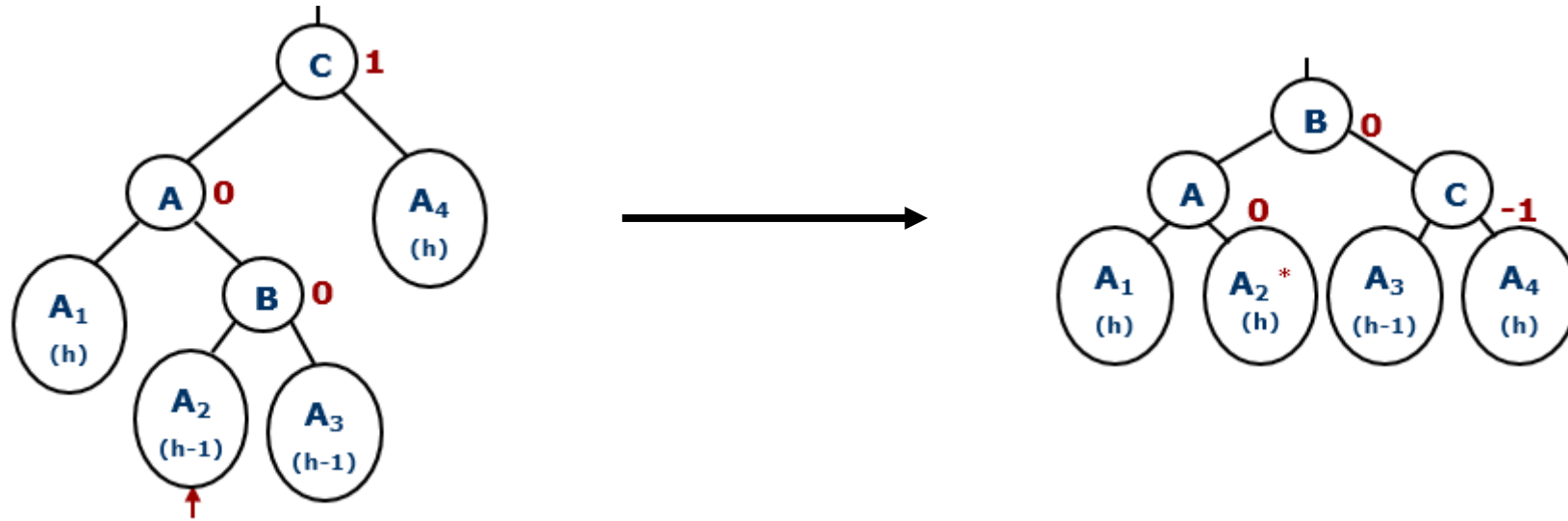
(5)



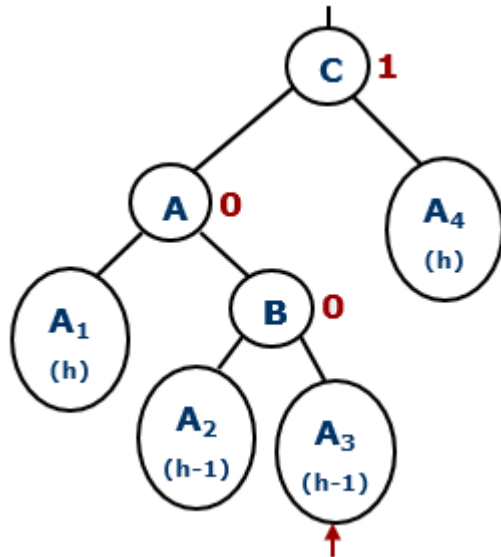
(6)

# Erhaltung der Balancierten Binärbäumen

(5)



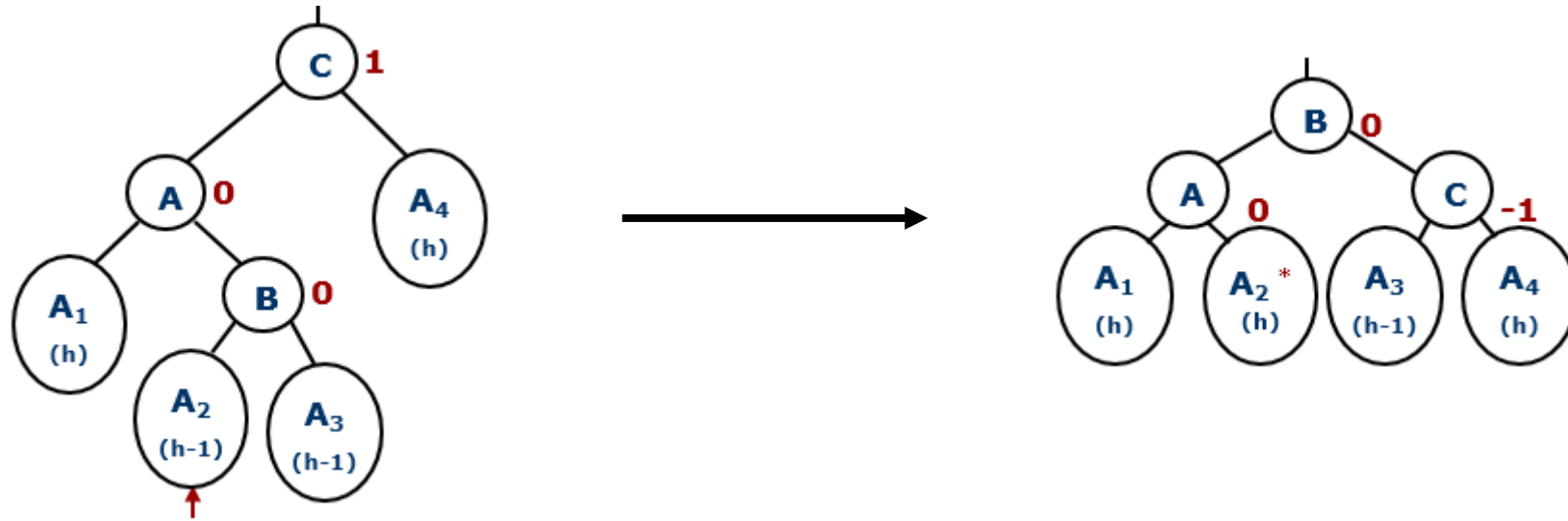
(6)



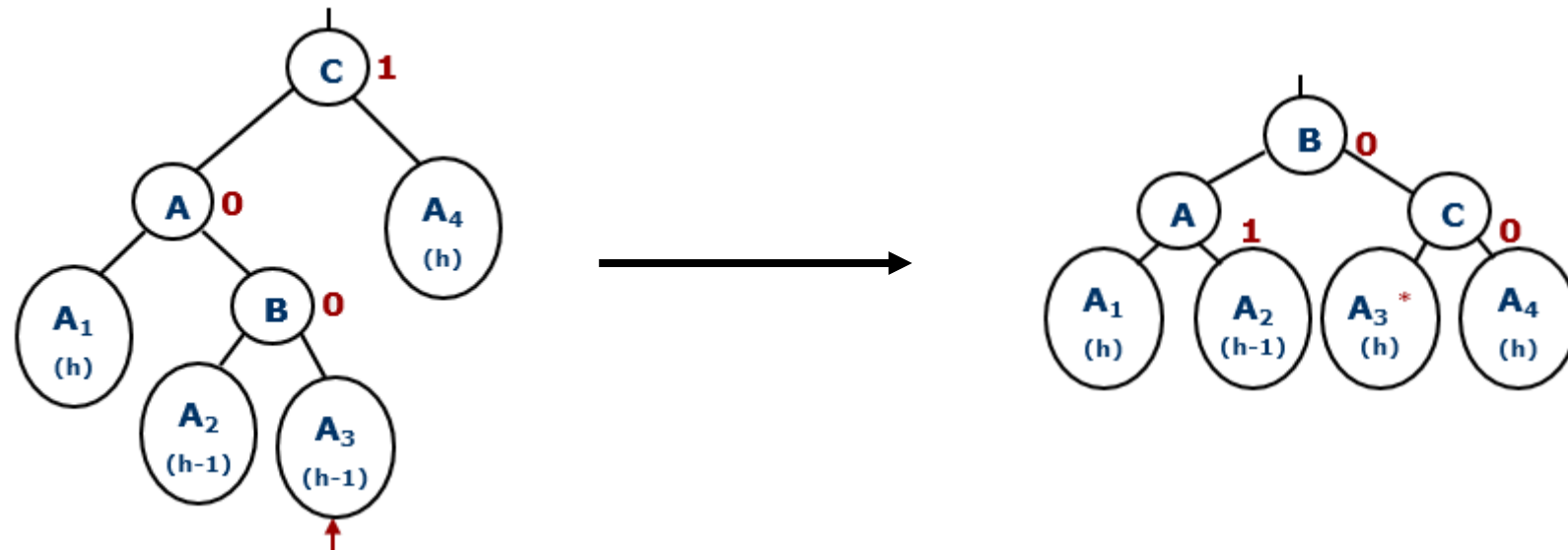


# Erhaltung der Balancierten Binärbäumen

(5)

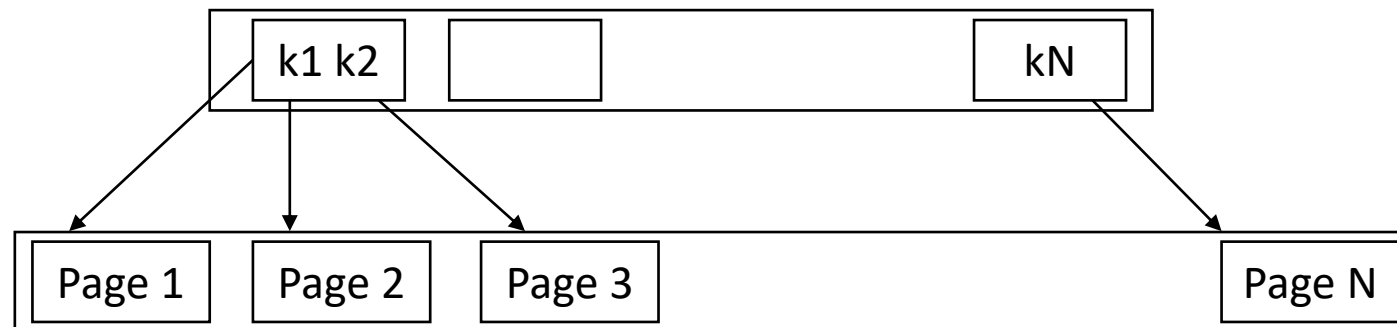


(6)



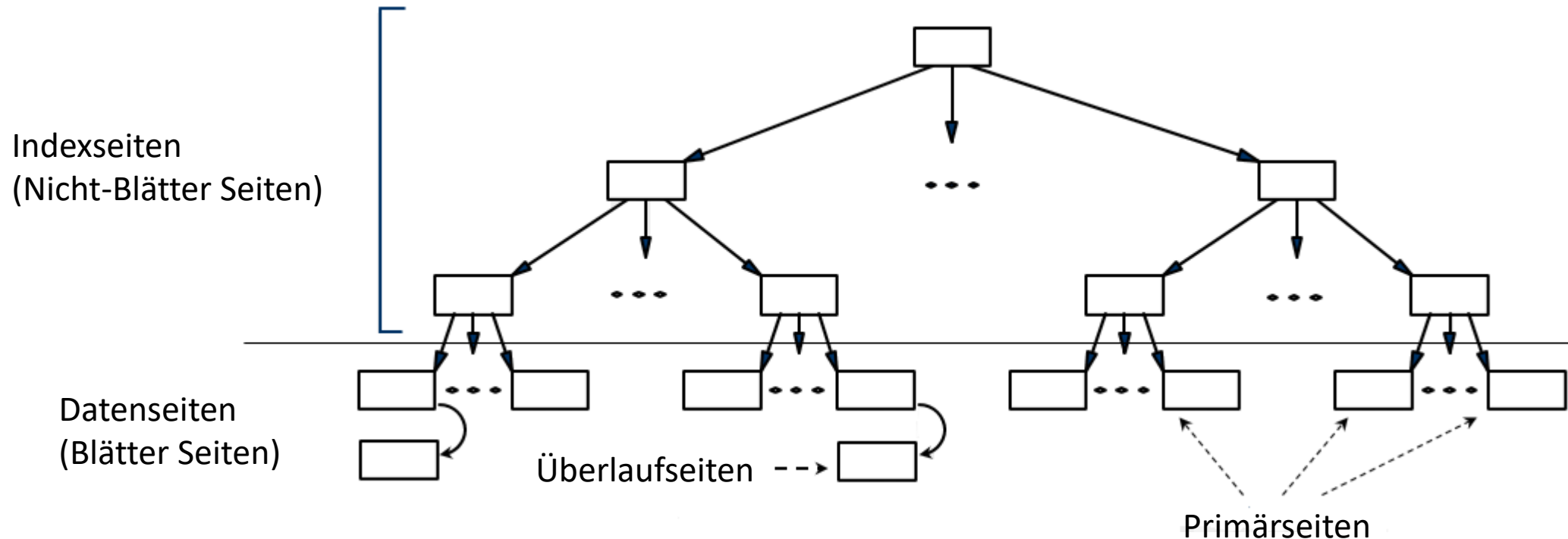
# Bereichsanfragen

- Finde alle Studenten mit Note > 8
  - Wenn die Daten sortiert sind, dann Binärsuche um der erste Student zu finden und dann Scannen, um die anderen zu finden
  - Aber: Binärsuche kann teuer sein
- Einfache Idee:
  - Index erstellen → ISAM (Index-Sequential Access Method)
  - Binärsuche auf dem Index (kleiner als die Datei selber)



# ISAM (Index-Sequential Access Method)

- Eine sehr einfache und manchmal auch sehr effektive Indexstruktur
- Wie beim Nachschlagen in einem Wörterbuch
- Die Indexdatei kann immer noch zu groß sein → wir können die Prozedur wiederholen (Index zu dem Index, usw. )



# ISAM

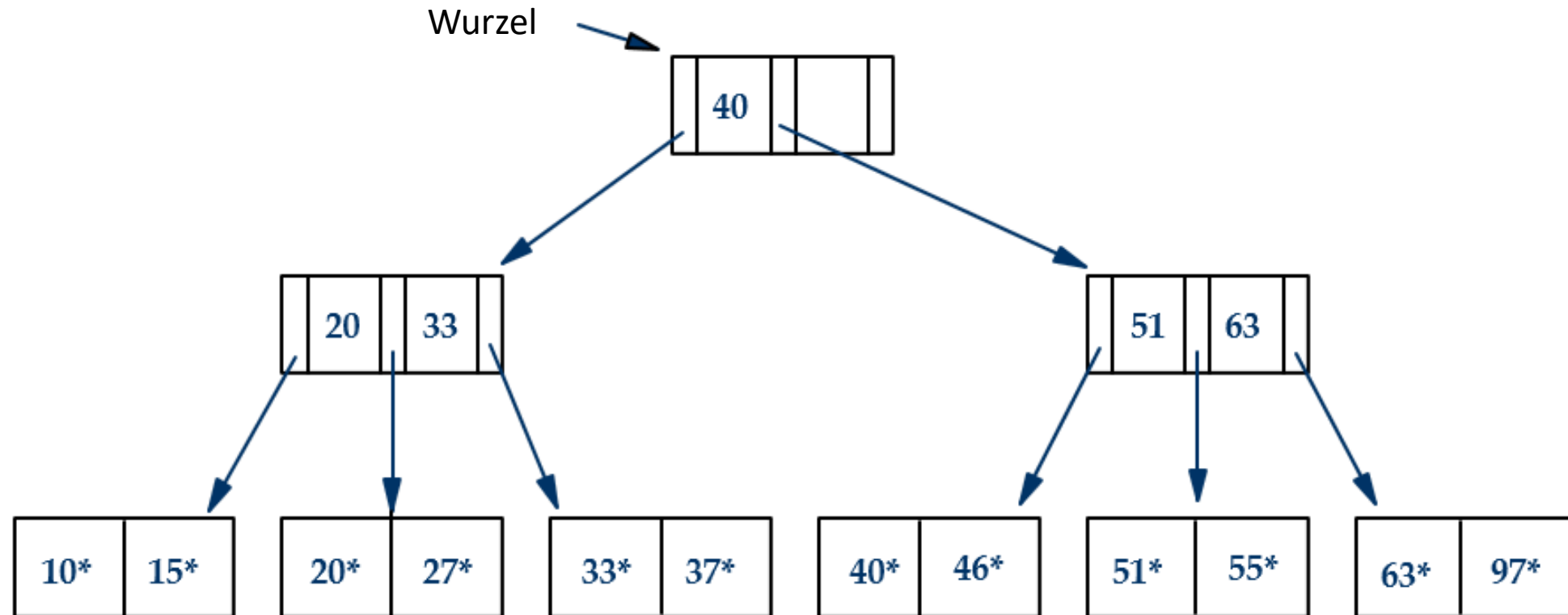
- Sowohl der Index als auch die Datensätze in den Datenseiten werden nach dem Suchschlüssel geordnet abgespeichert
- Der Index befindet sich auf Seiten, die sequentiell hintereinander auf dem Sekundärspeicher abgelegt sind
- Die Erstellung der Dateien:
  - Erst werden die Datenseiten erstellt und sequentiell abgespeichert
  - Dann die Indexseiten
  - Dann wird Speicherplatz für Überlaufseiten reserviert
- Indexeinträge der Form <Suchschlüsselwert, SeitenID>
- Statische Baumstruktur: Einfügen und Löschen von Datensätzen beeinflusst nur die Blätterseiten

# ISAM

- Suche eines Schlüssels:
  - beginne bei der Wurzel
  - Vergleiche den Suchschlüssel mit den Suchschlüsseln aus den Indexeinträgen und verfolge die Verweise, um eine Blattseite zu erreichen
  - Von dieser Datenseite an kann man wegen der Sortierung alle weiteren Datenseiten lesen, bis der gewünschte Datensatz gefunden wurde
  - Kosten:  $\log_F N$  , wobei  $F$  = Anzahl von Indexseiten,  $N$  = Anzahl von Datenseiten
- Einfügen: finde die Datenseite, wo der neue Datensatz gehören sollte und, wenn es freier Platz gibt, füge den Datensatz ein
  - Falls es kein freier Platz in der Datenseite gibt → Überlaufseite
- Löschen: finde die Datenseite und lösche den Datensatz
  - Falls eine Überlaufseite frei wird → Speicherplatz freigeben

# ISAM Baum - Beispiel

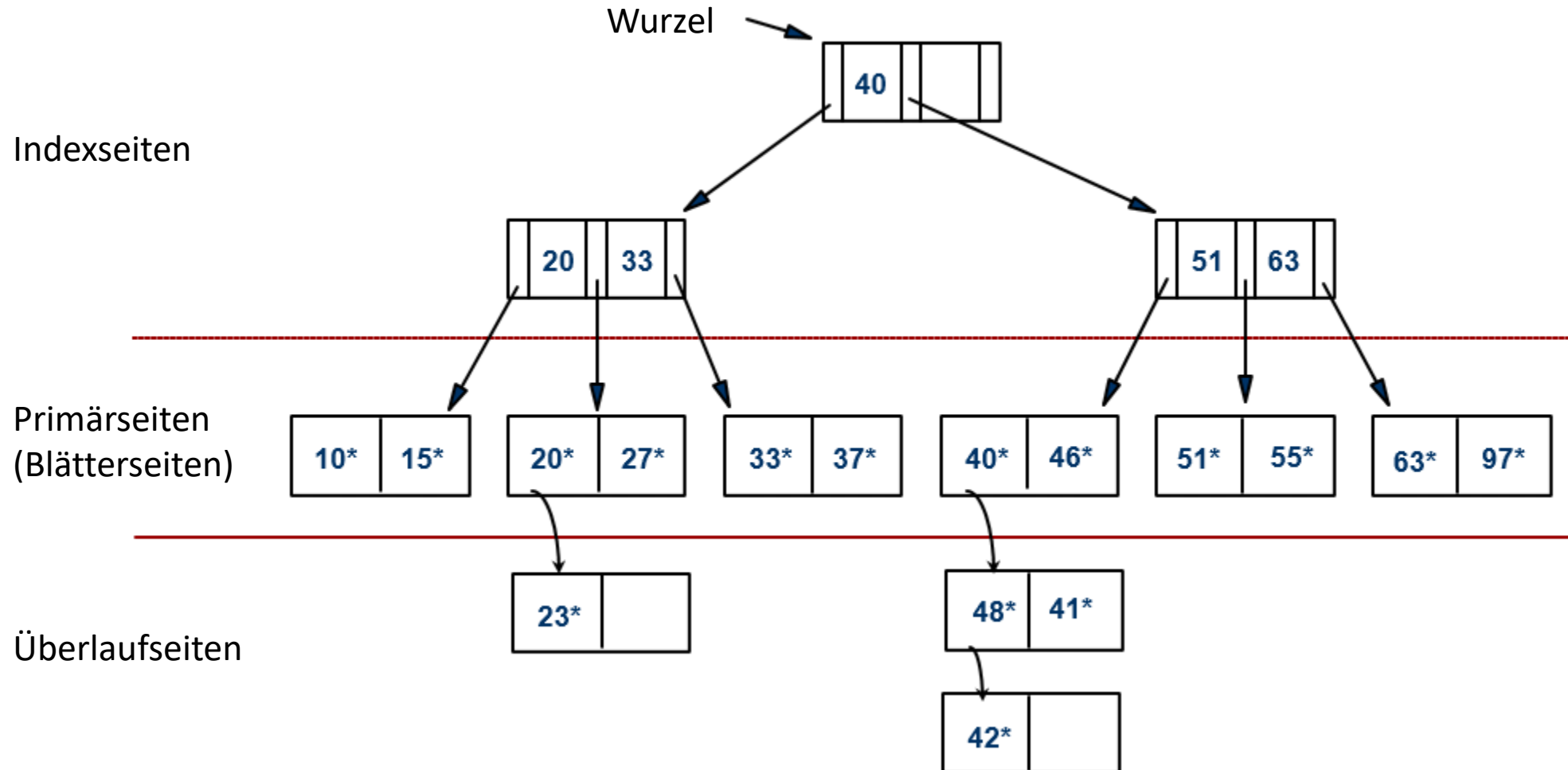
- In jedem Knoten passen zwei Einträge



- Füge folgende Werte ein: 23\*, 48\*, 41\*, 42\*

# ISAM Baum - Beispiel

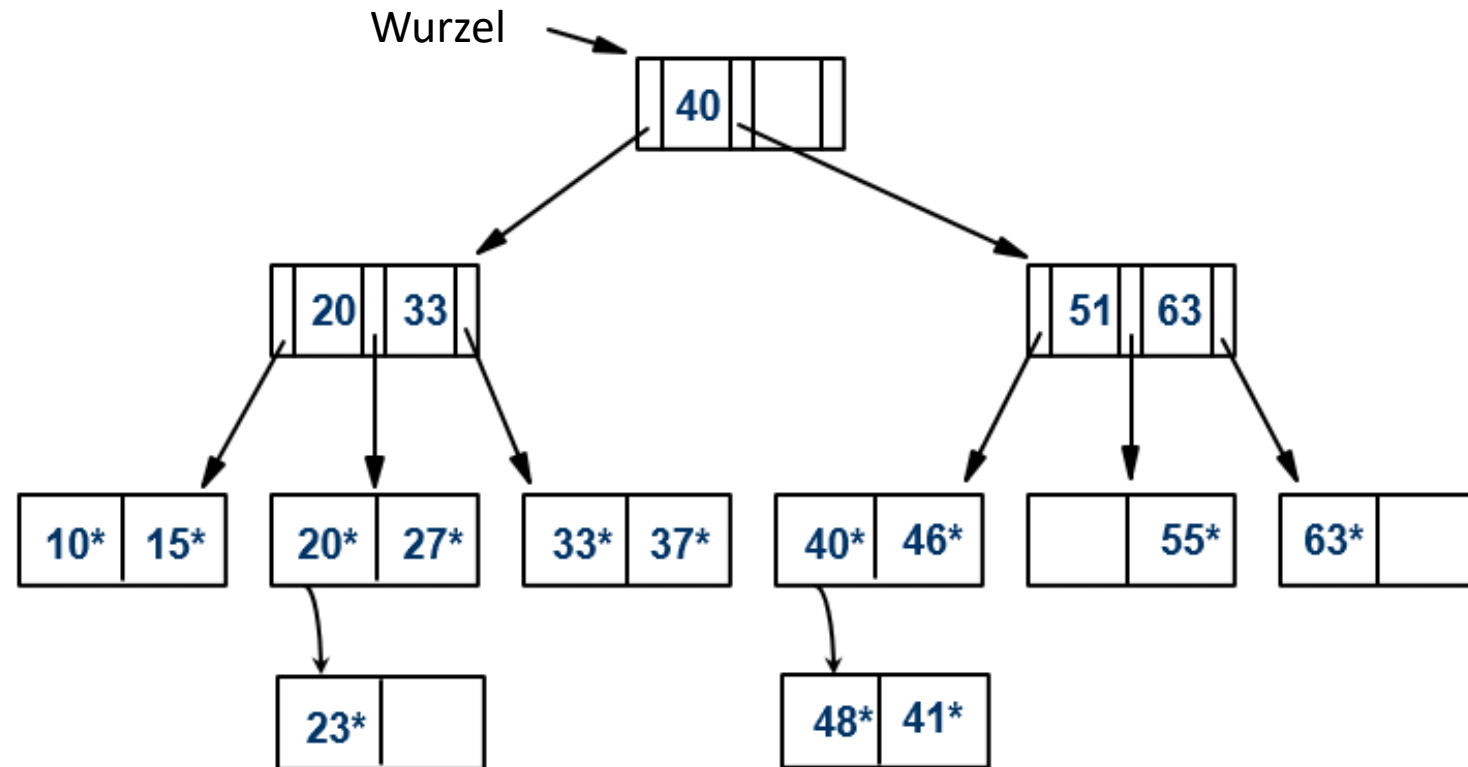
- Nach dem Einfügen von  $23^*$ ,  $48^*$ ,  $41^*$ ,  $42^*$



- Lösche:  $42^*$ ,  $51^*$ ,  $97^*$

# ISAM Baum - Beispiel

- Nach dem Löschen von 42\*, 51\*, 97\*
- Bemerkung. Nach dem Löschen von 51\* kommt dieser Wert in den Indexseiten immer noch vor (obwohl er von den Datenseiten gelöscht wurde)



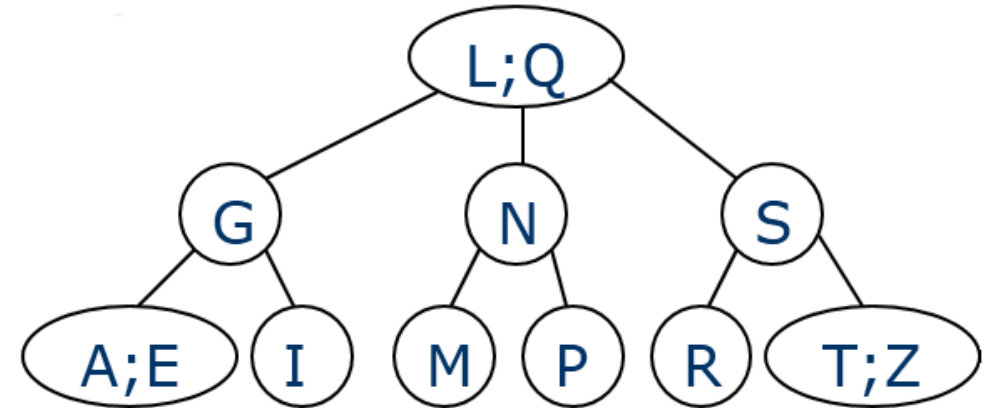


# Vor- und Nachteile von ISAM

- Es kann unbalanciert werden nach vielen Einfügen oder Löschen von Datensätzen → ungleichförmiger Suchzeit
- Datensätze in den Überlaufseiten sind meistens nicht sortiert
- Effizientes Einfügen und Löschen (keine Änderungen in den Knoten des Baumes außer den Blättern, keine Baum Balancierung)
- Effizienter gleichzeitiger Zugriff – die Knoten des Baumes werden nicht gesperrt
- Geeignet für Dateien, die nicht viel geändert werden

# B-Bäume

- Am meisten verwendete Indexstruktur
- „B“ kommt von „balanced“ oder „broad“
- B-Baum – sortierter Baum
- Normale Binärbäume wurden als Suchstruktur für den Hauptspeicher konzipiert und eignen sich nicht für den Sekundärspeicher
- Für den Hintergrundpeicher → Mehrwegbäume, deren Knotengrößen auf die Seitenkapazität abgestimmt werden
- Ein Knoten des Baumes entspricht einer Seite des Sekundärspeichers



# Die Struktur eines Knotens mit m Einträge

$V_0$	$S_1$	$V_1$	$S_2$	$V_2$	...	$V_{m-1}$	$S_m$	$V_m$
-------	-------	-------	-------	-------	-----	-----------	-------	-------

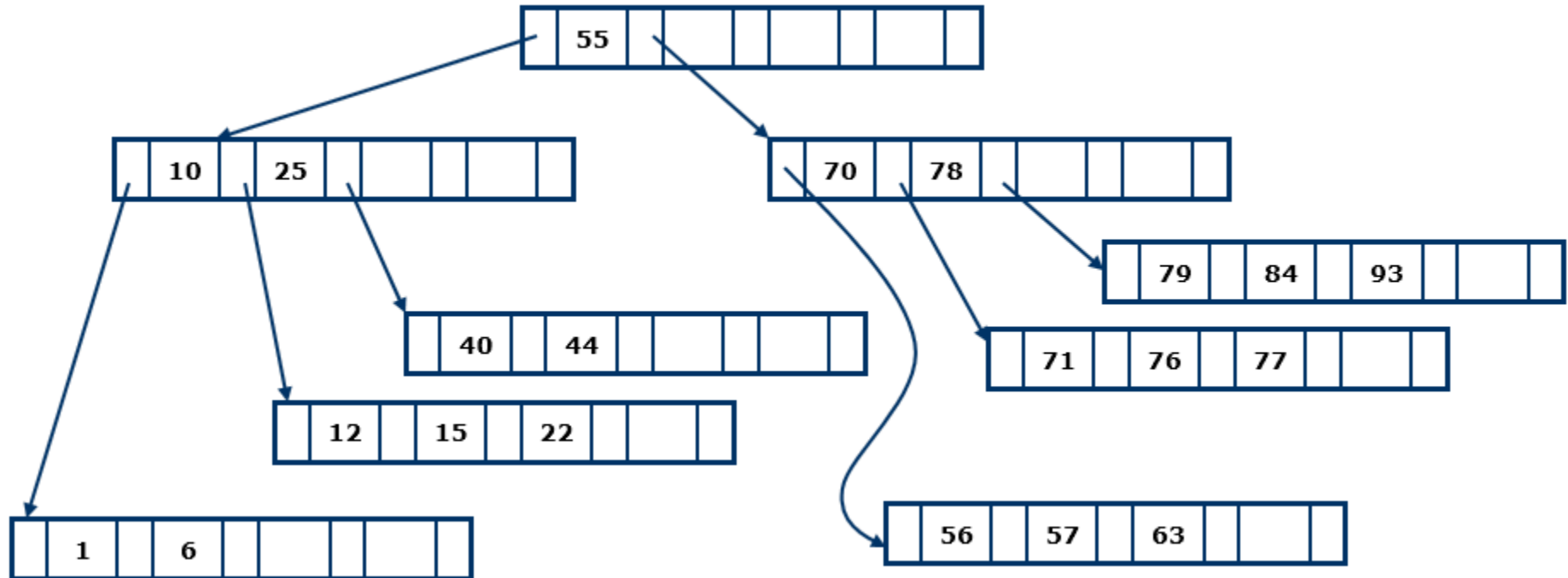
- $S_i$  – Suchschlüsselwerte
- $V_i$  – Zeiger zu einem Teilbaum
- $S_1 < S_2 < \dots < S_m$

# B-Baum mit Ordnung $k$ - Eigenschaften

- Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge
- Jeder Knoten außer der Wurzel hat mindestens  $k$  und höchstens  $2 \cdot k$  Einträge
- Die Wurzel hat zwischen einem und  $2 \cdot k$  Einträgen
- Die Einträge werden in allen Knoten sortiert behalten
- Alle Knoten mit  $n$  Einträgen, außer den Blättern, haben  $n+1$  Kinder
- Seien  $S_1, S_2, \dots, S_n$  die Schlüssel eines Knotens mit  $n+1$  Kindern; die Verweise auf diese Kinder:  $V_0, V_1, \dots, V_n$ 
  - $V_0$  – weist auf den Teilbaum mit Schlüssel kleiner als  $S_1$
  - $V_i$  ( $i = 1, \dots, n-1$ ) weist auf den Teilbaum, dessen Schlüssel zwischen  $S_i$  und  $S_{i+1}$  liegen
  - $V_n$  weist auf den Teilbaum mit Schlüssel größer als  $S_n$
- In den Blattknoten sind die Zeiger nicht definiert

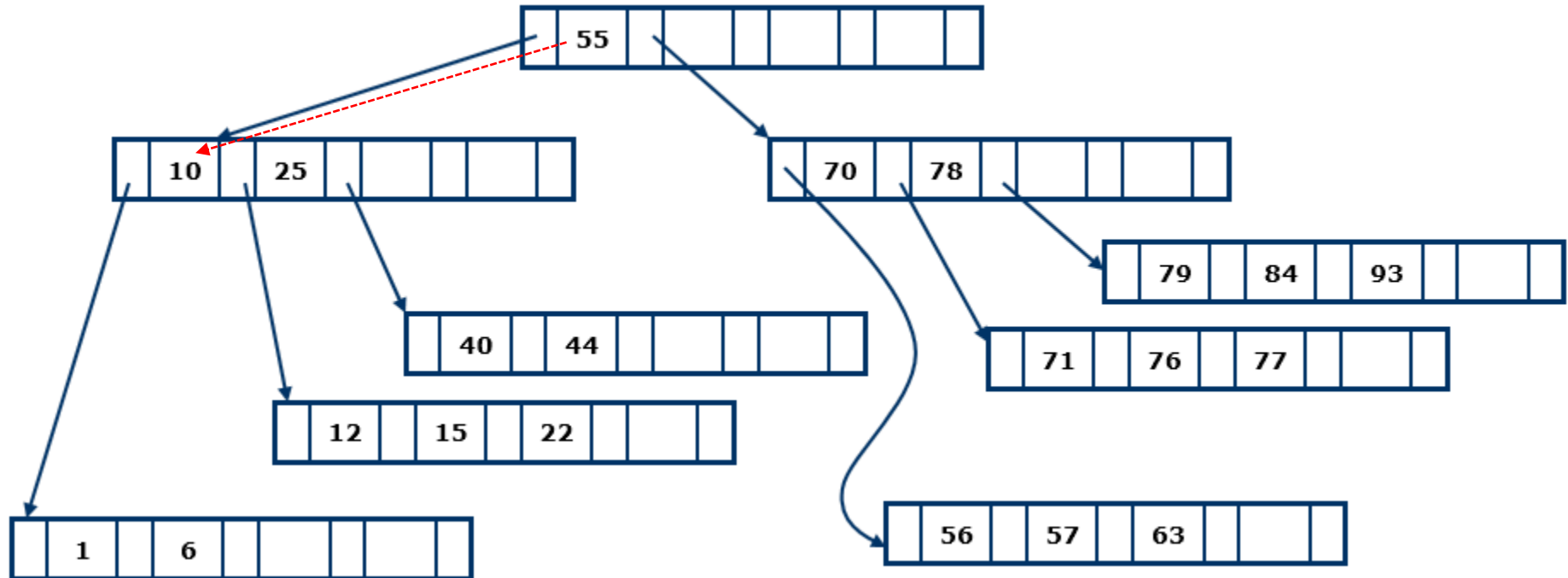
# Suchen in einem B-Baum

- In jedem Knoten muss man für den Pfad zwischen zwei Zeiger wählen
- Wir suchen „15“:



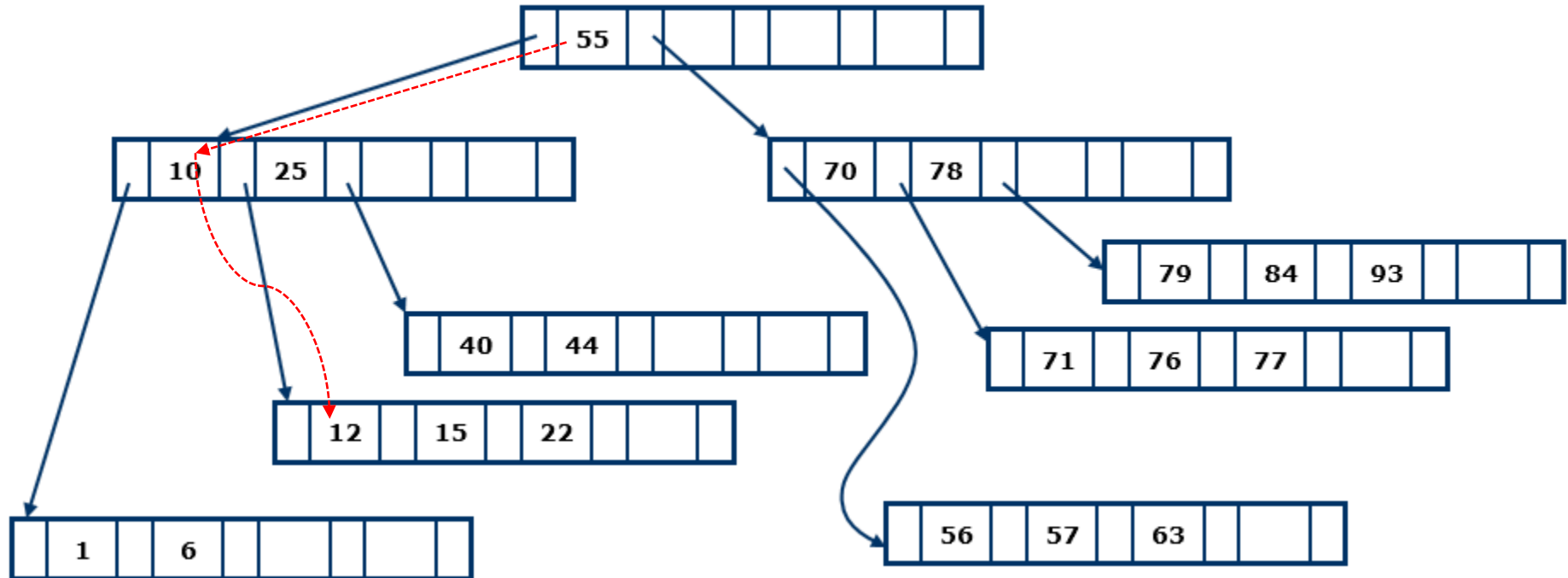
# Suchen in einem B-Baum

- In jedem Knoten muss man für den Pfad zwischen zwei Zeiger wählen
- Wir suchen „15“:



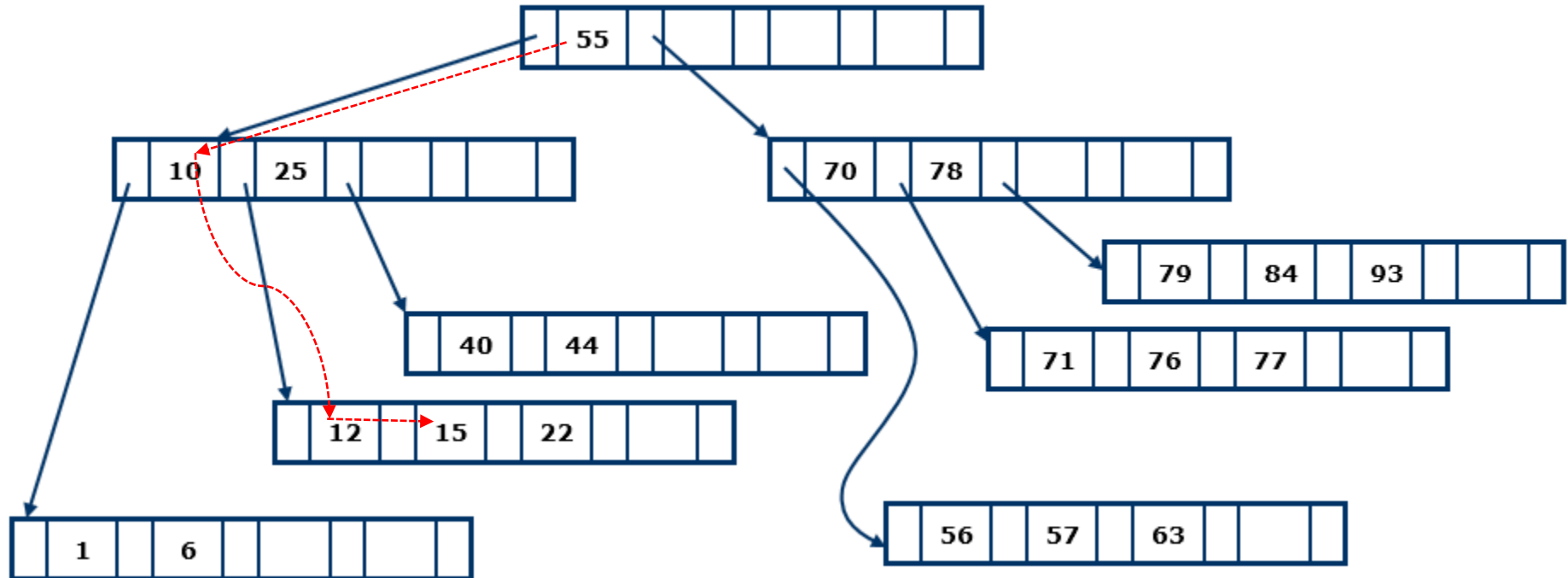
# Suchen in einem B-Baum

- In jedem Knoten muss man für den Pfad zwischen zwei Zeiger wählen
- Wir suchen „15“:



# Suchen in einem B-Baum

- In jedem Knoten muss man für den Pfad zwischen zwei Zeiger wählen
- Wir suchen „15“:



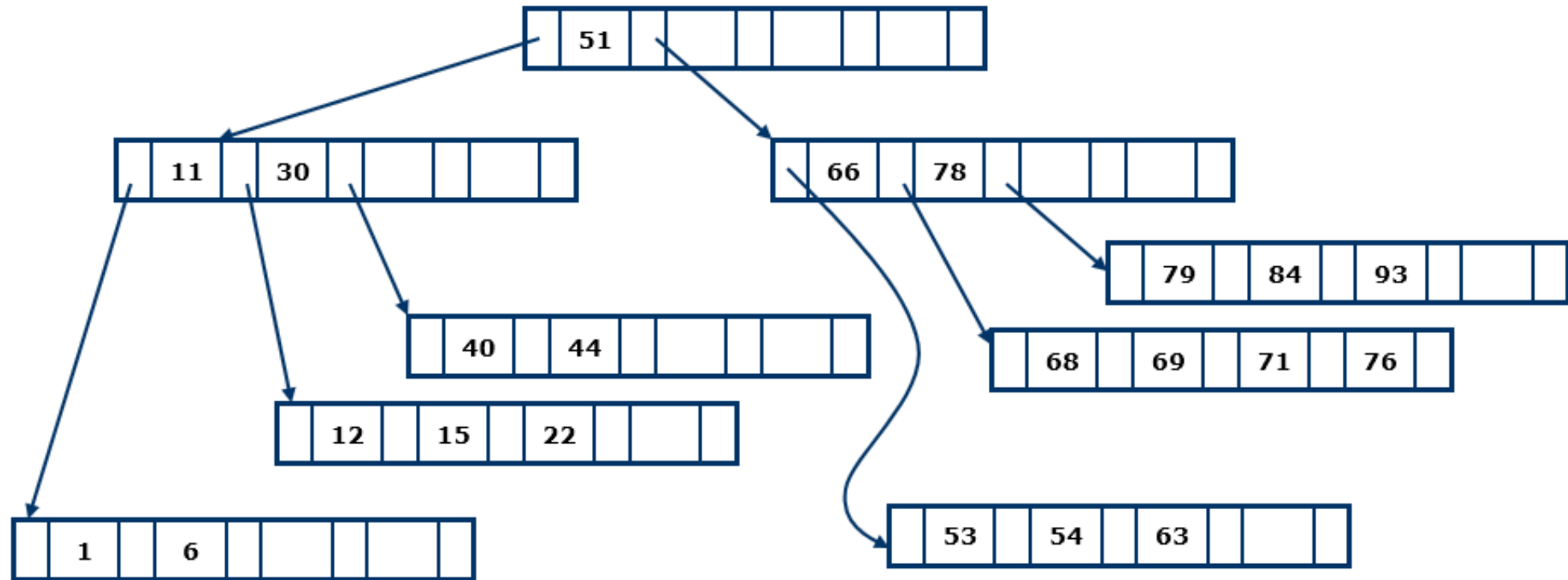


# Einfügen in einem B-Baum

- Algorithmus für den Einfügevorgang:
  - Führe eine Suche nach dem Schlüssel, um die Einfügestelle zu finden
  - Füge den Schlüssel dort ein
  - Ist der Knoten überfüllt, teile ihn:
    - Erzeuge einen neuen Knoten und belege ihn mit den Einträgen des überfüllten Knotens, deren Schlüssel größer ist als der des mittleren Eintrags
    - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein
    - Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten
  - Ist der Vaterknoten jetzt überfüllt?
    - Handelt es sich um die Wurzel, so lege eine neue Wurzel an
    - Wiederhole Schritt 3 mit dem Vaterknoten

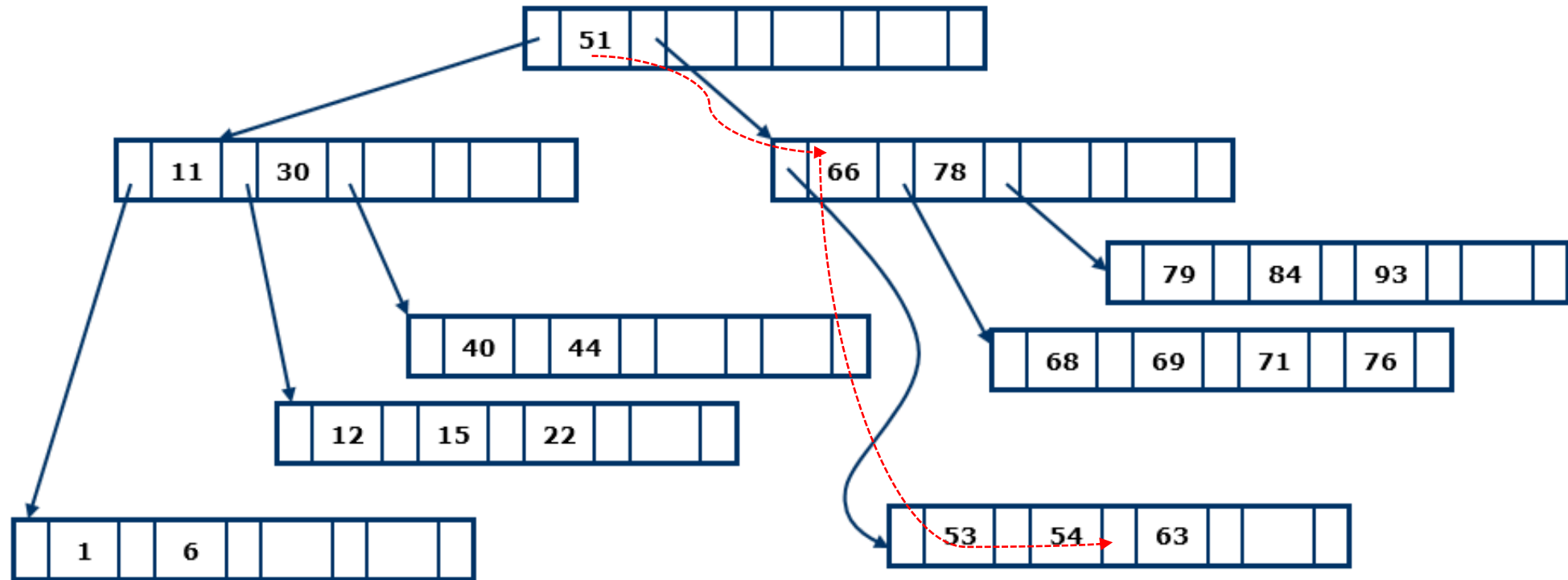
# Einfügen in einem B-Baum

- Füge den Schlüssel „57“ ein



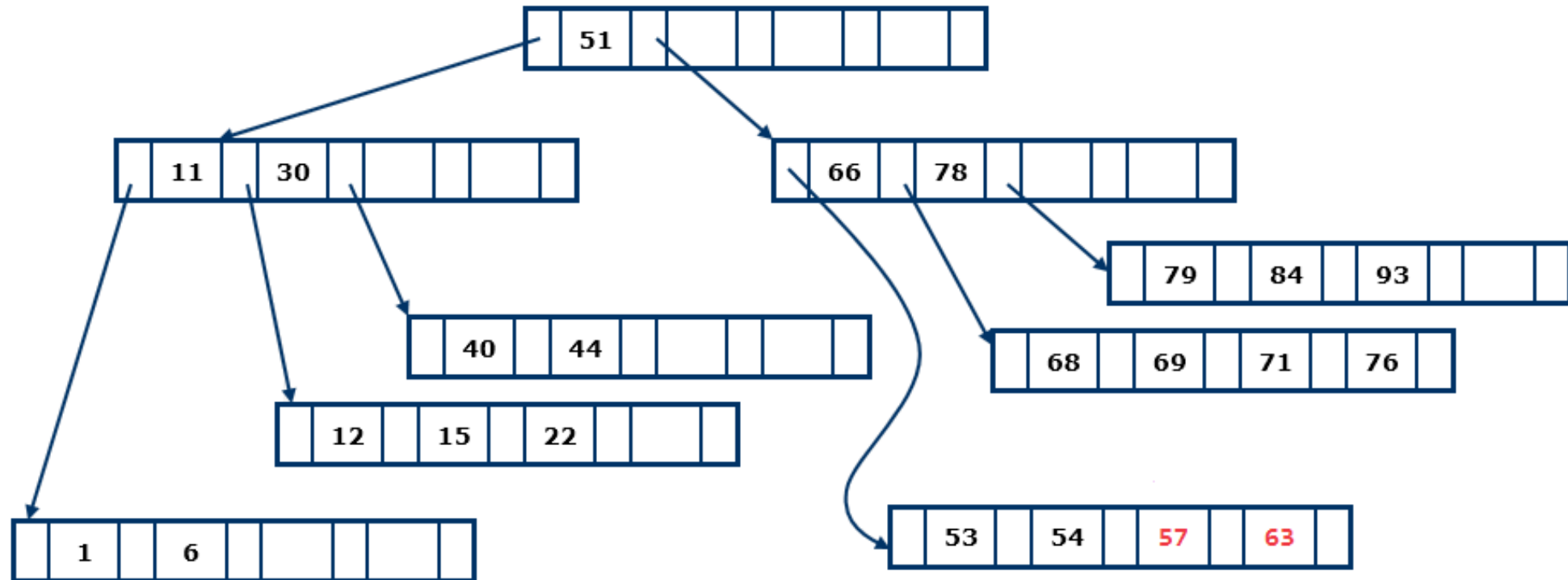
# Einfügen in einem B-Baum

- Füge den Schlüssel „57“ ein



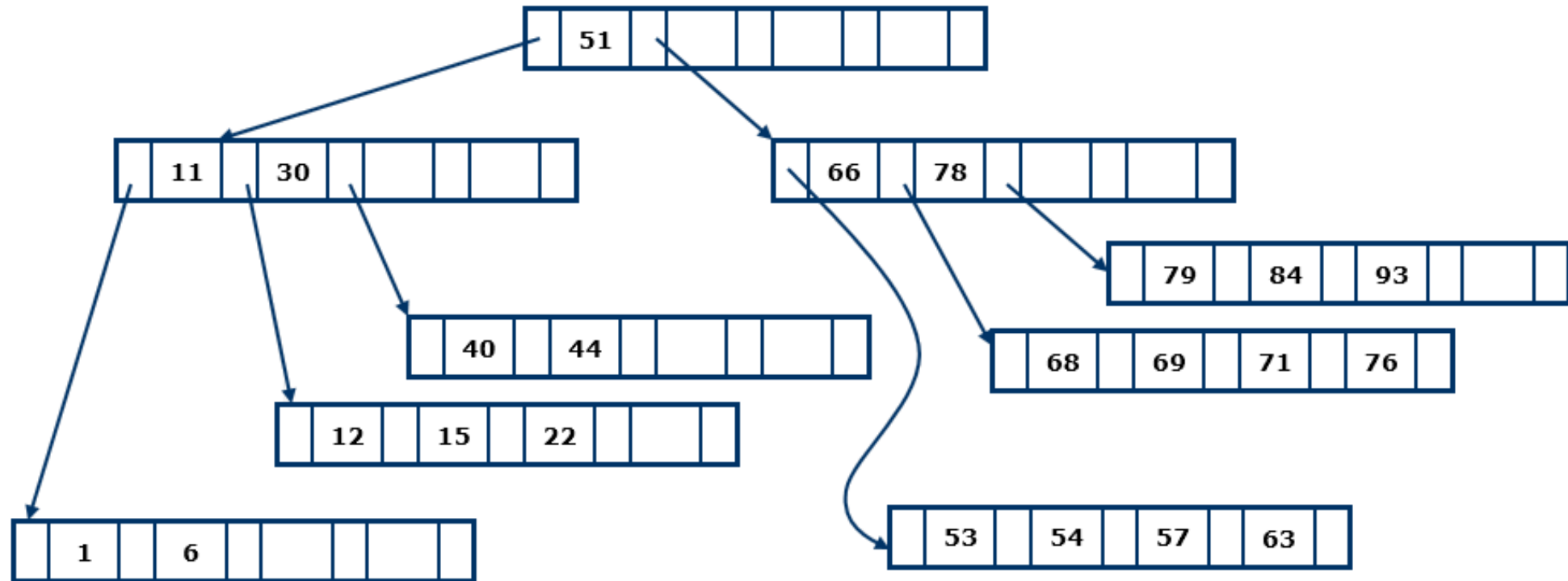
# Einfügen in einem B-Baum

- Füge den Schlüssel „57“ ein



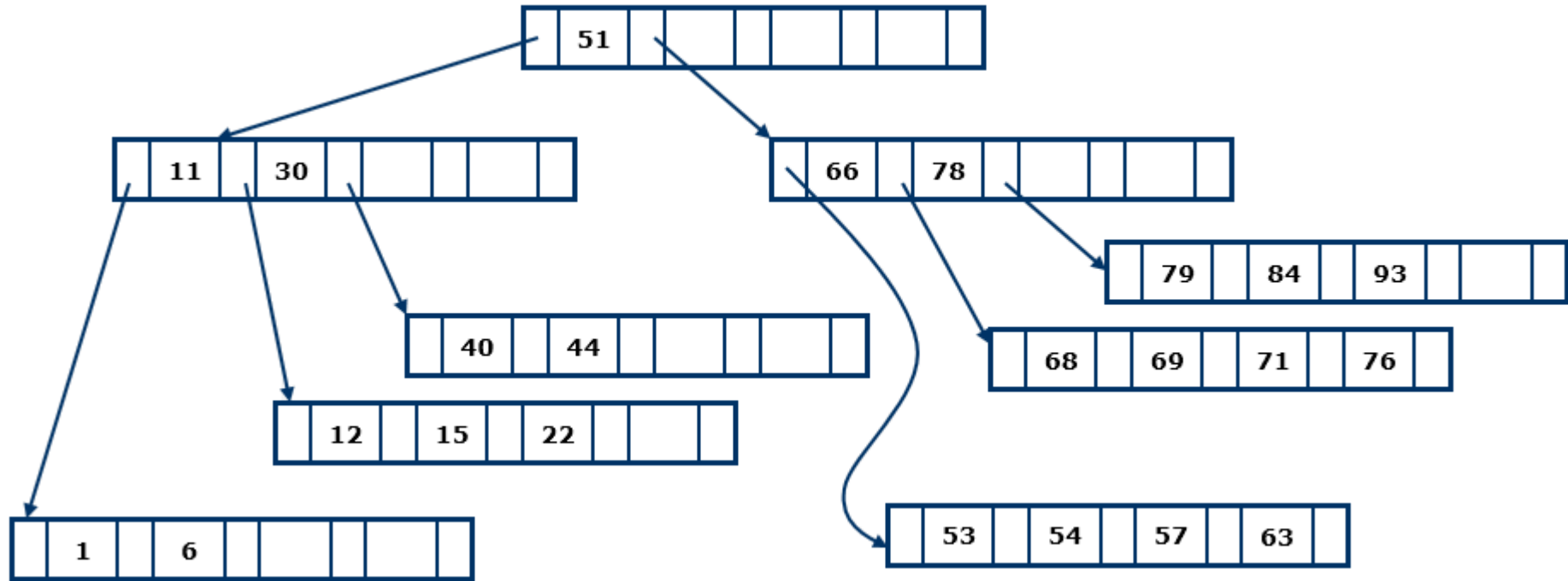
# Einfügen in einem B-Baum

- Füge den Schlüssel „57“ ein



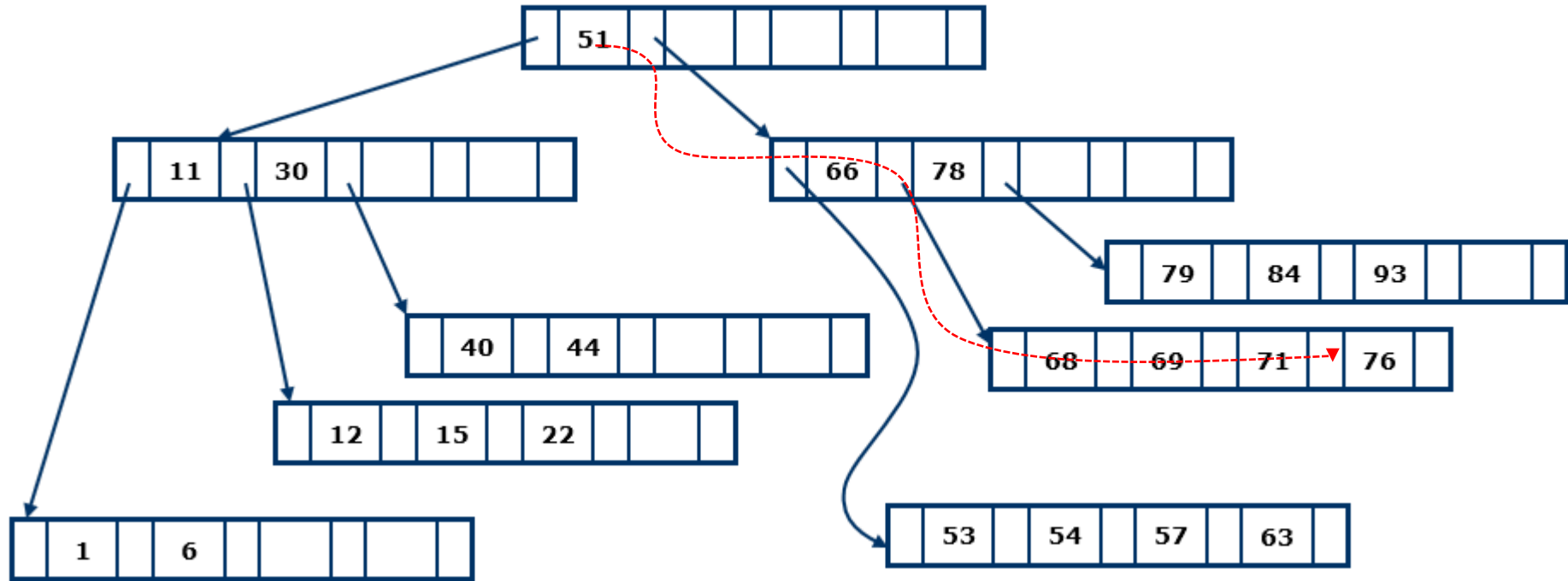
# Einfügen in einem B-Baum

- Füge den Schlüssel „72“ ein



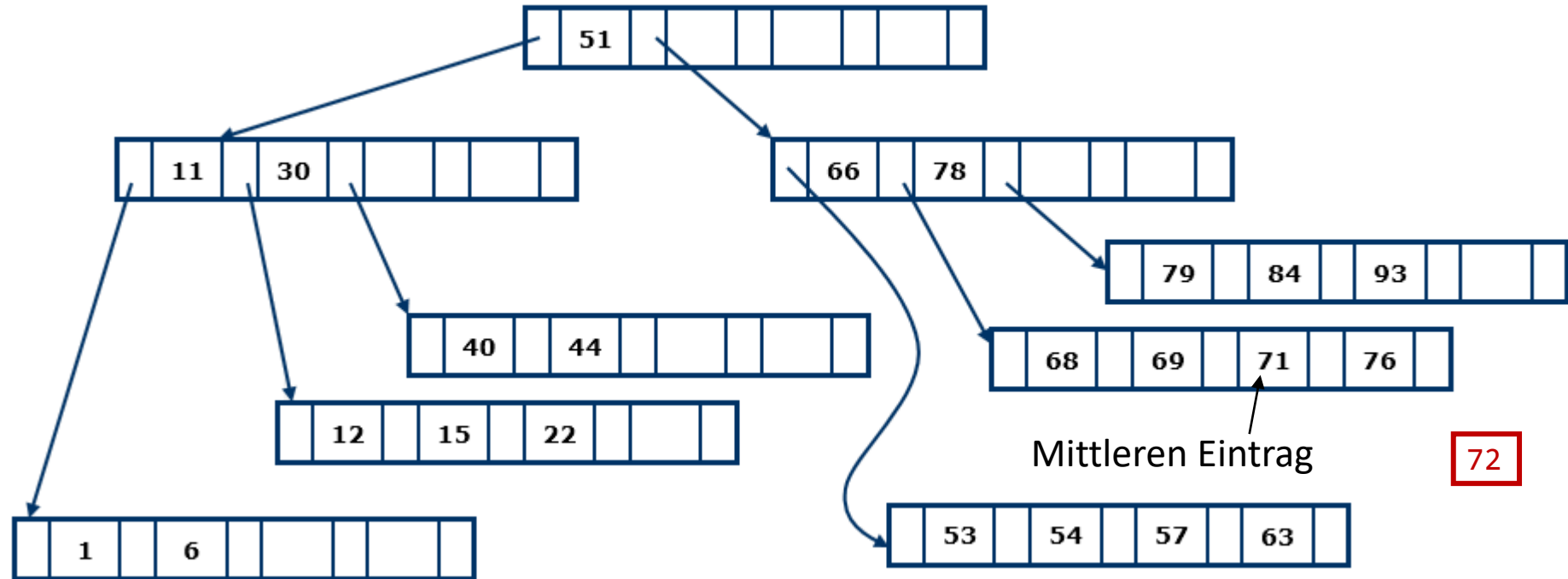
# Einfügen in einem B-Baum

- Füge den Schlüssel „72“ ein



# Einfügen in einem B-Baum

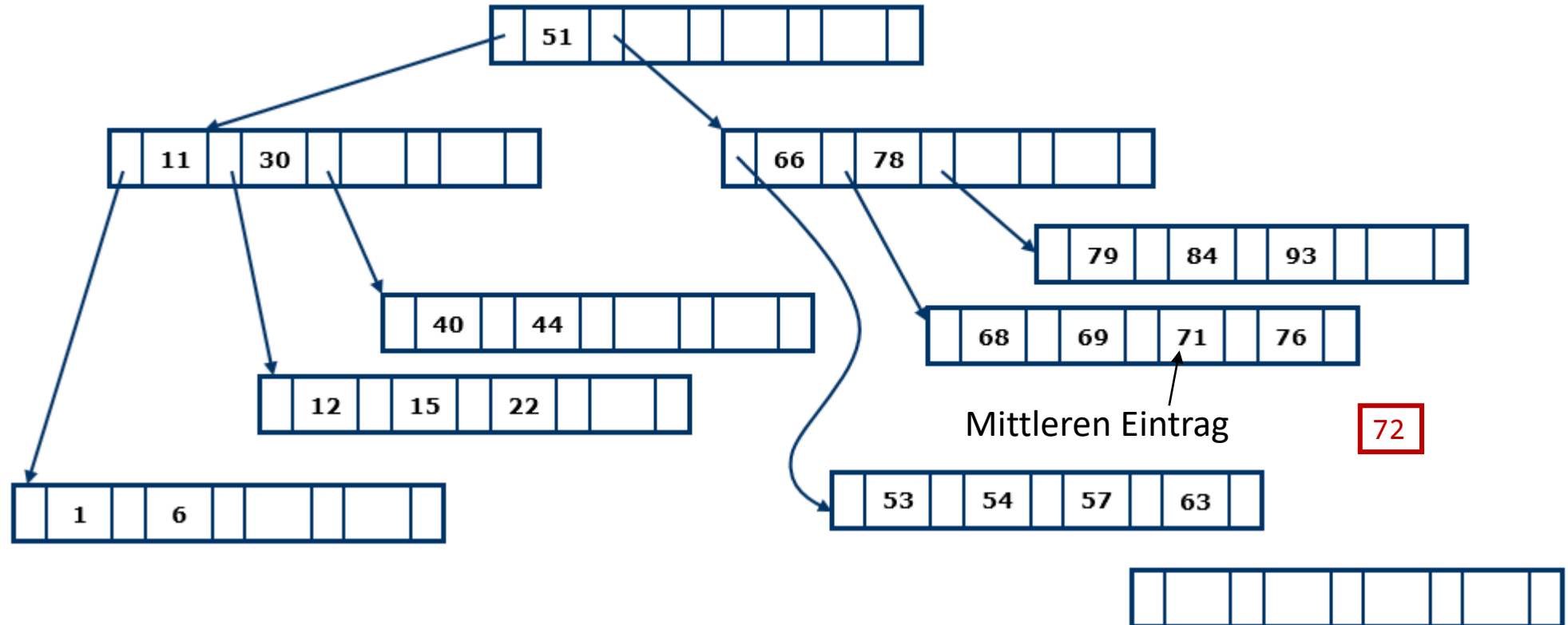
- Füge den Schlüssel „72“ ein





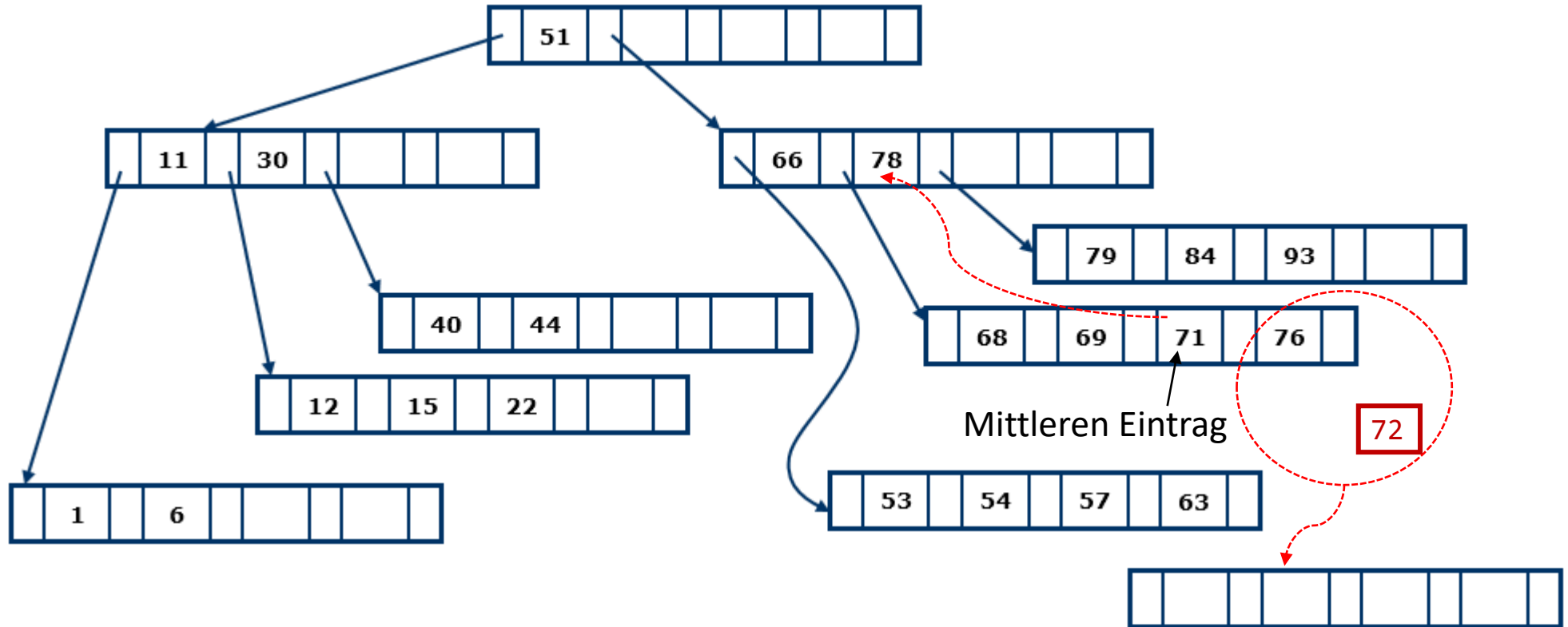
# Einfügen in einem B-Baum

- Füge den Schlüssel „72“ ein



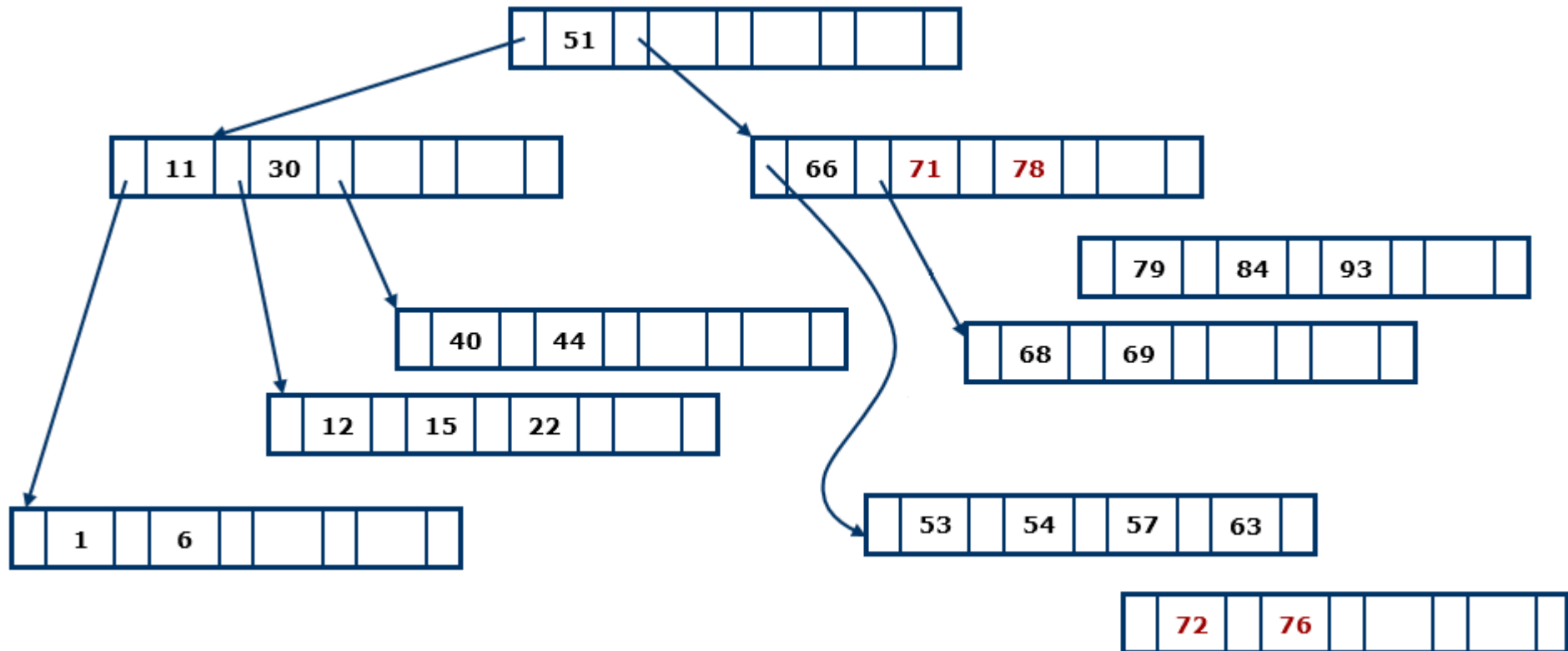
# Einfügen in einem B-Baum

- Füge den Schlüssel „72“ ein



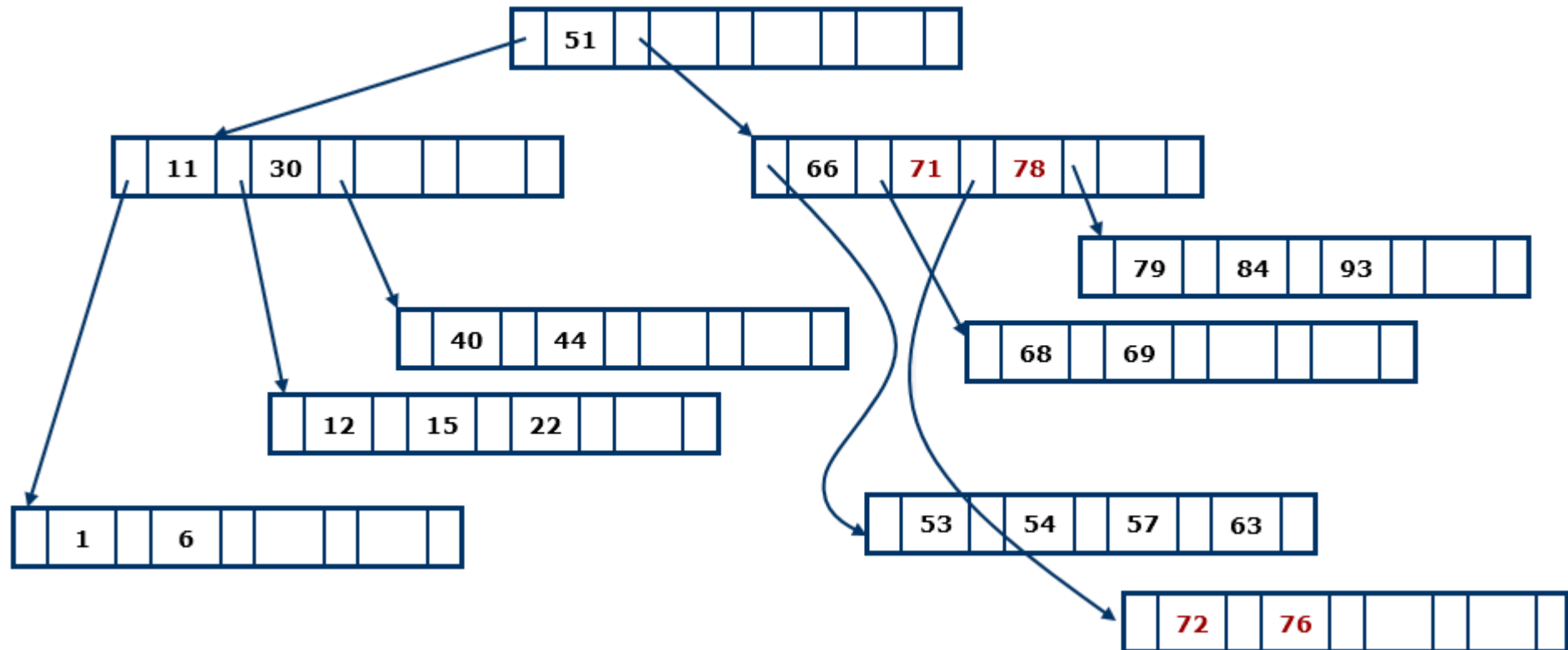
# Einfügen in einem B-Baum

- Füge den Schlüssel „72“ ein



# Einfügen in einem B-Baum

- Füge den Schlüssel „72“ ein



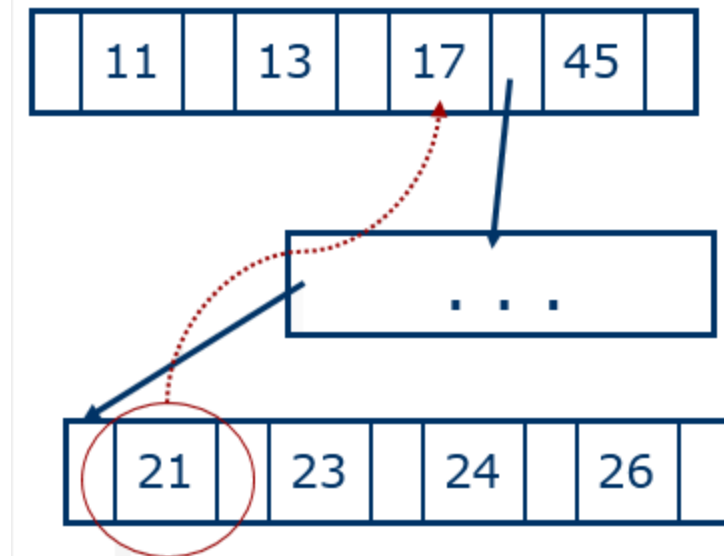
# Löschen in einem B-Baum mit Ordnung $k$

- Schritte beim Löschen:
  - Finde den Knoten, der den gewünschten Schlüssel enthält
  - Wenn der Knoten kein Blatt ist, dann kann man ein Schlüssel aus einem Kind anstatt dem gelöschten Schlüssel kopieren
  - Falls es einen Unterlauf gibt in dem Knoten  $p$ :
    - Balance – wenn einer der Nachbar genügend Suchschlüssel hat ( $> k$ ), wird seine Suchschlüsselreihe mit der von  $p$  ausgeglichen
    - Merge –  $p$  wird mit dem Nachbarn zu einem einzigen Knoten verschmolzen

# Algorithmus für Löschen in dem B-Baum mit Ordnung $k$

## 1. Finde den Schlüssel, der gelöscht werden muss

- Wenn sich der Schlüssel in einem inneren Knoten (Nichblattknoten) befindet, dann:
  - Ersetze den Schlüssel mit seinem Nachfolger/ größeren Nachbar (der Schlüssel am weitesten links aus dem Blatt am weitesten links aus dem rechten Teilbaum)

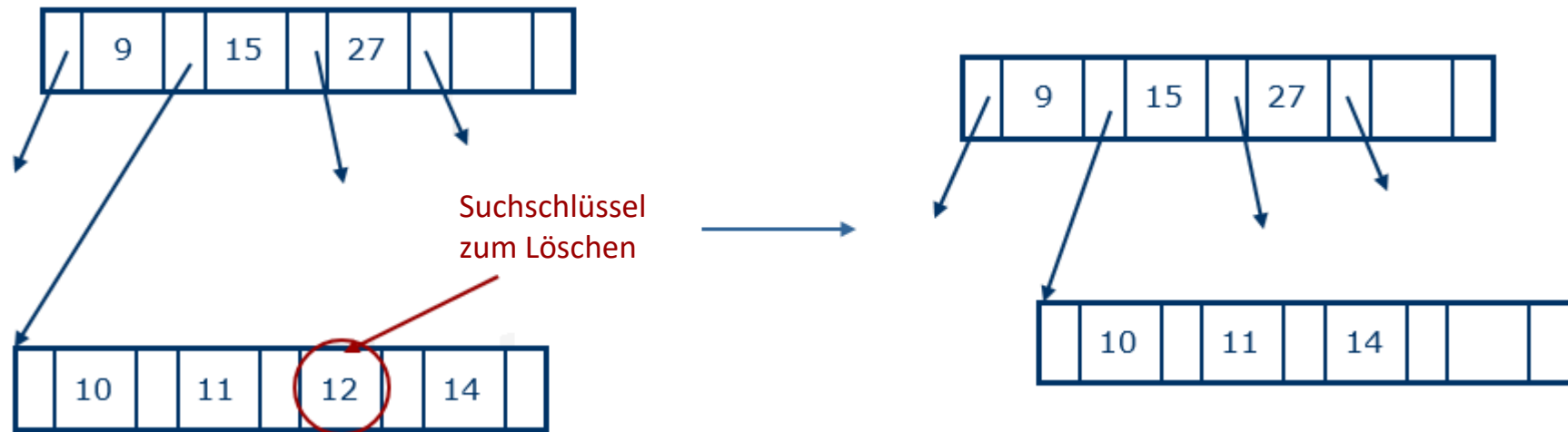


# Algorithmus für Löschen in dem B-Baum mit Ordnung k

2. Wiederhole diesen Schritt bis man zu Fall A oder Fall B/C gelangt

A. Wenn sich der Suchschlüssel, den wir grade löschen wollen, in der Wurzel befindet **oder** die Anzahl der gebliebenen Schlüssel  $\geq k$  ist:

- Lösche den gewünschte Suchschlüssel
- Die Zeiger zu den Suchschlüssel in dem Knoten müssen neu angeordnet werden
- Algorithmus wird beendet

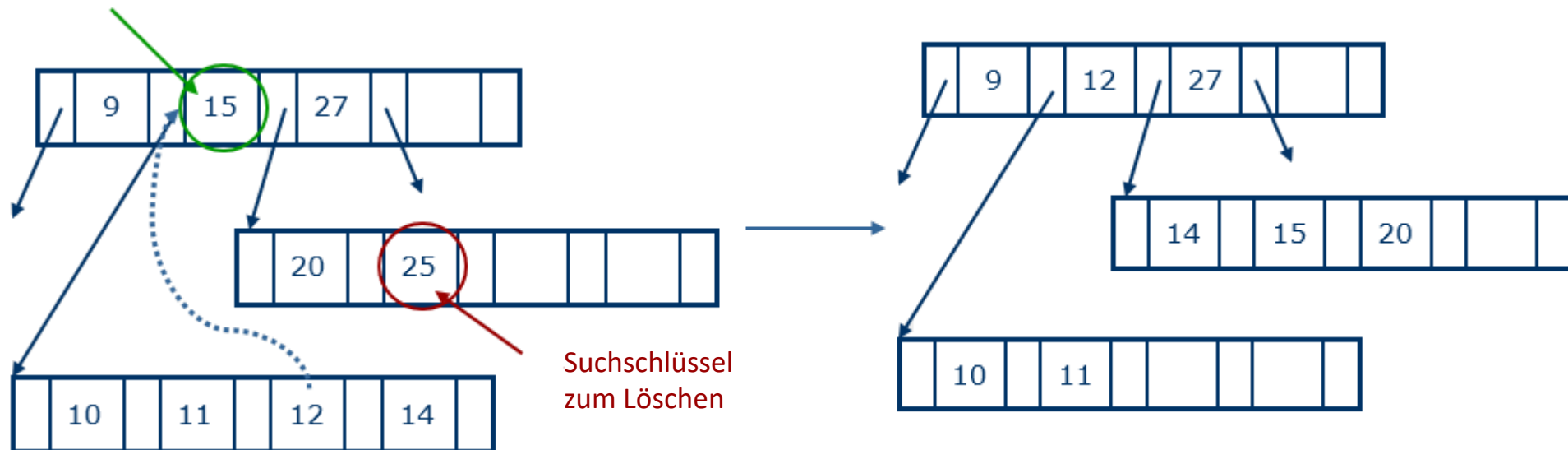


# Algorithmus für Löschen in dem B-Baum mit Ordnung k

B. Wenn die Anzahl der gebliebenen Suchschlüssel  $< k$  ist und ein Nachbarknoten  $> k$  Suchschlüssel enthält  $\rightarrow$  *Balance*

- Die Schlüssel der zwei Knoten und der Separatorschlüssel aus dem Elternteil werden gleichmäßig umverteilt
- Wähle den mittleren Schlüssel um den Separatorschlüssel in dem Elternteil (übergeordnete Knoten) zu ersetzen
- Algorithmus wird beendet

Separatorschlüssel



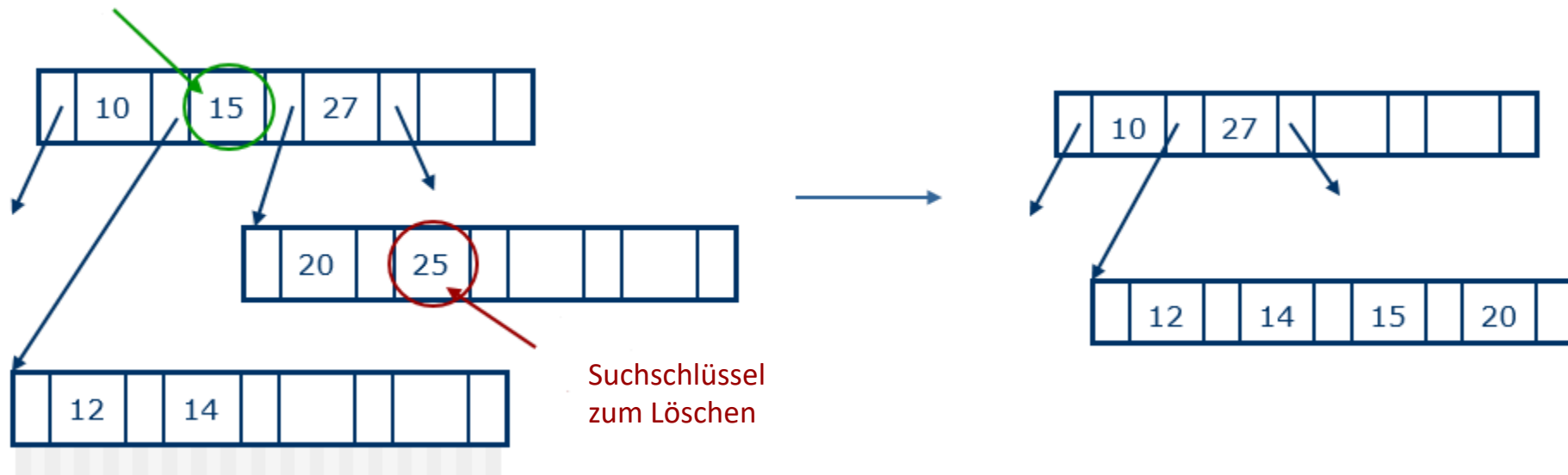


# Algorithmus für Löschen in dem B-Baum mit Ordnung k

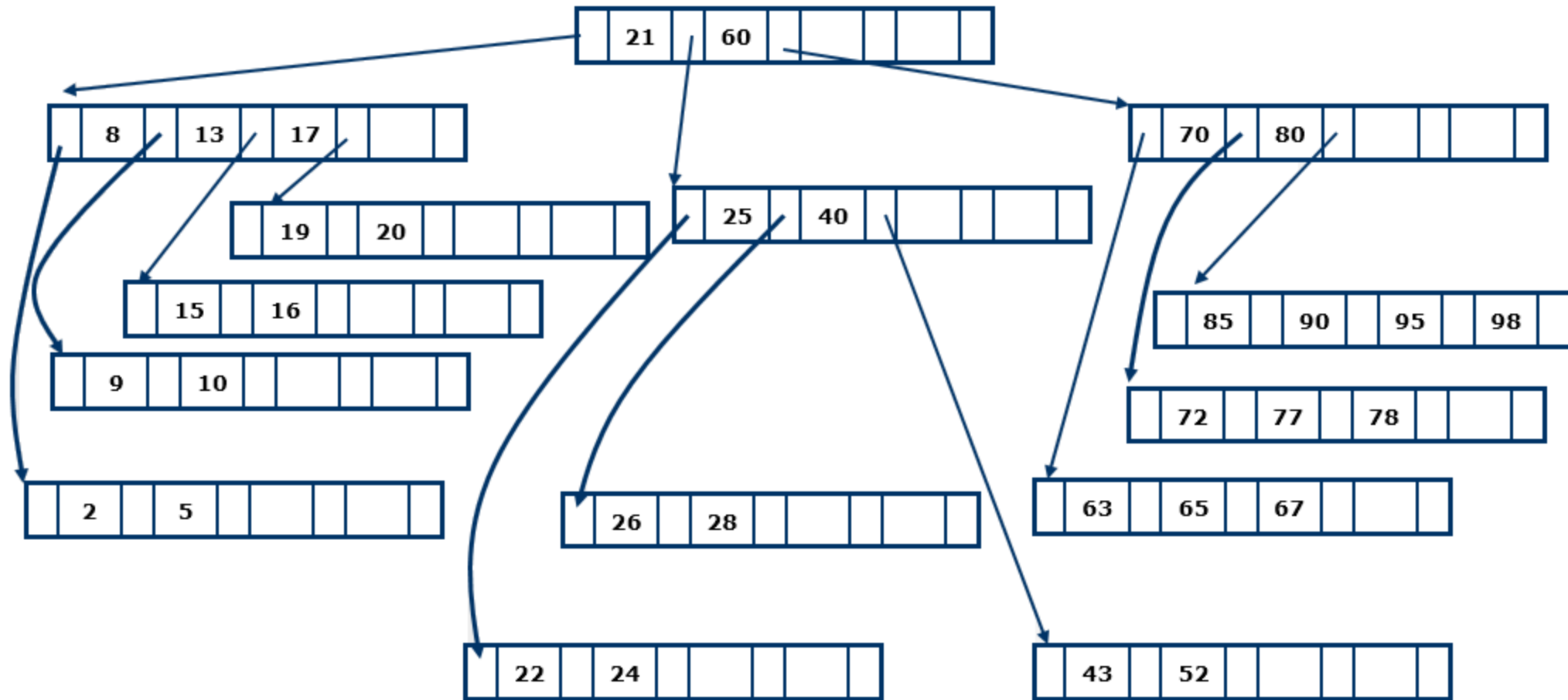
C. Wenn die Anzahl der gebliebenen Suchschlüsseln zusammen mit der Anzahl der Suchschlüsseln in jeder Nachbarknote  $< k$  ist  $\rightarrow$  *Merge*

- Verschmelze die zwei Knoten zusammen mit dem Separatorschlüssel
- Wiederhole Schritt 2. für den übergeordneten Knoten
- Wenn der übergeordnete Knoten der Wurzel ist und keine Schlüssel mehr enthält, dann wird der aktuelle Knoten zum Wurzel

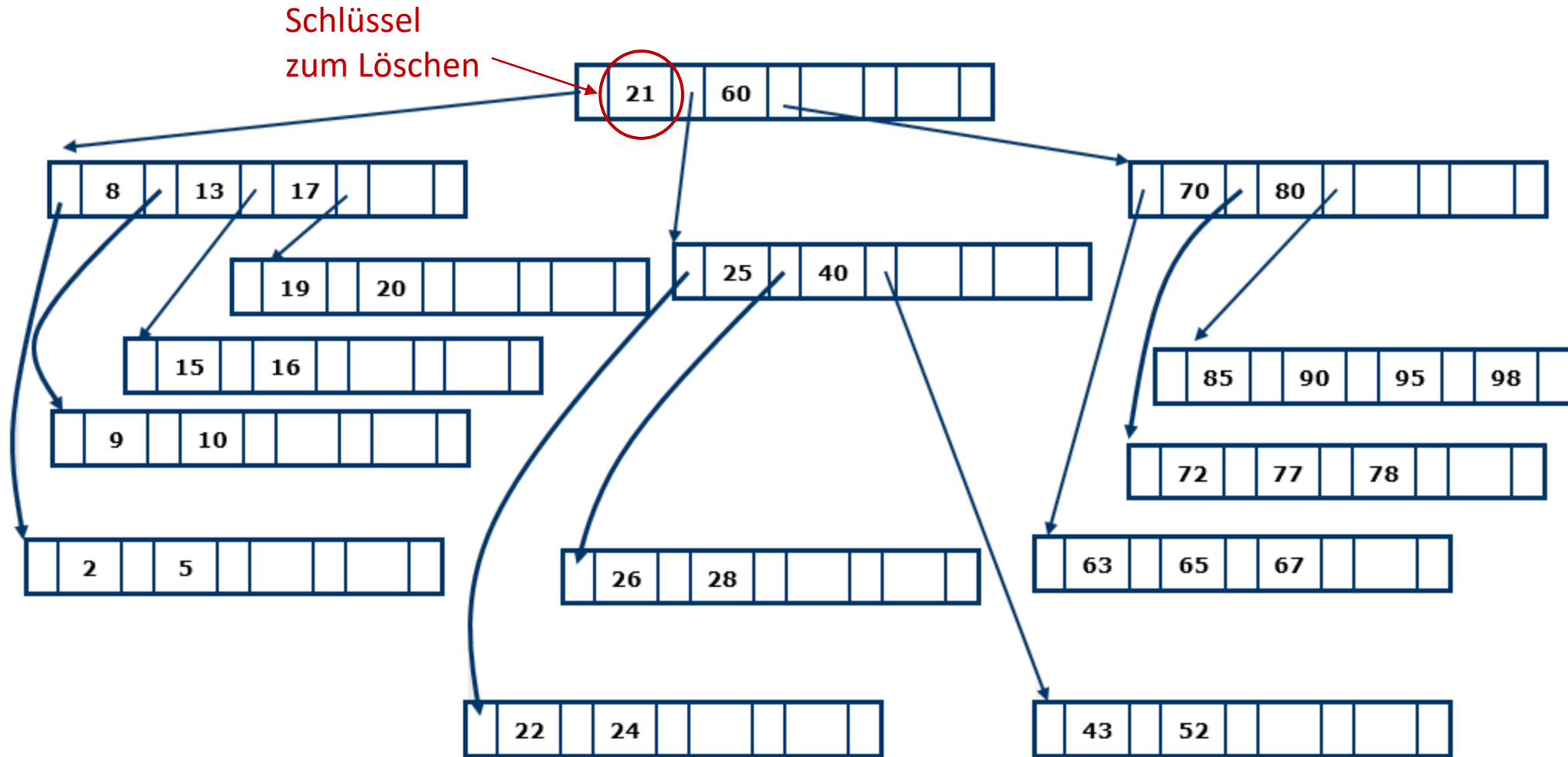
Separatorschlüssel



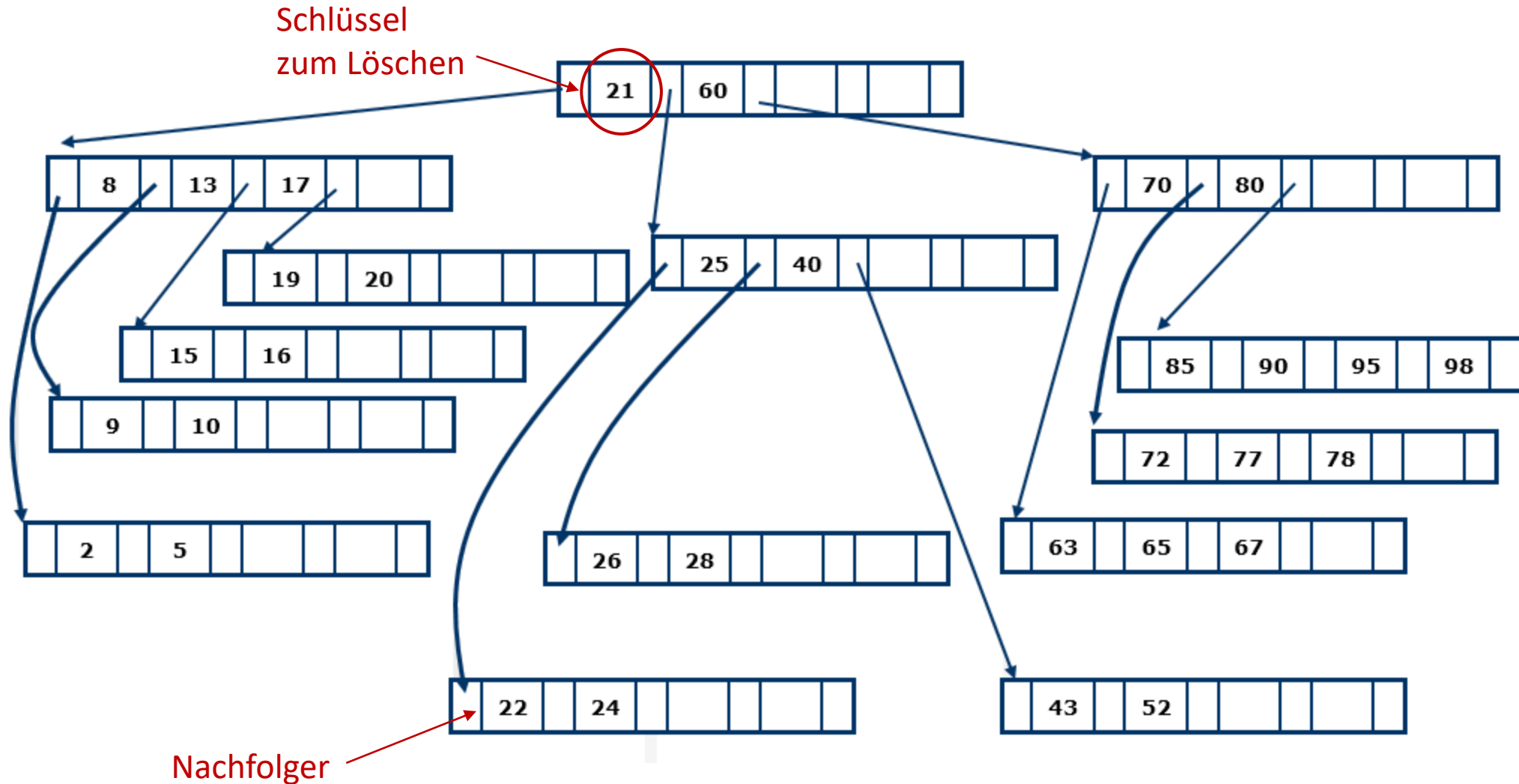
# Löschen in einem B-Baum



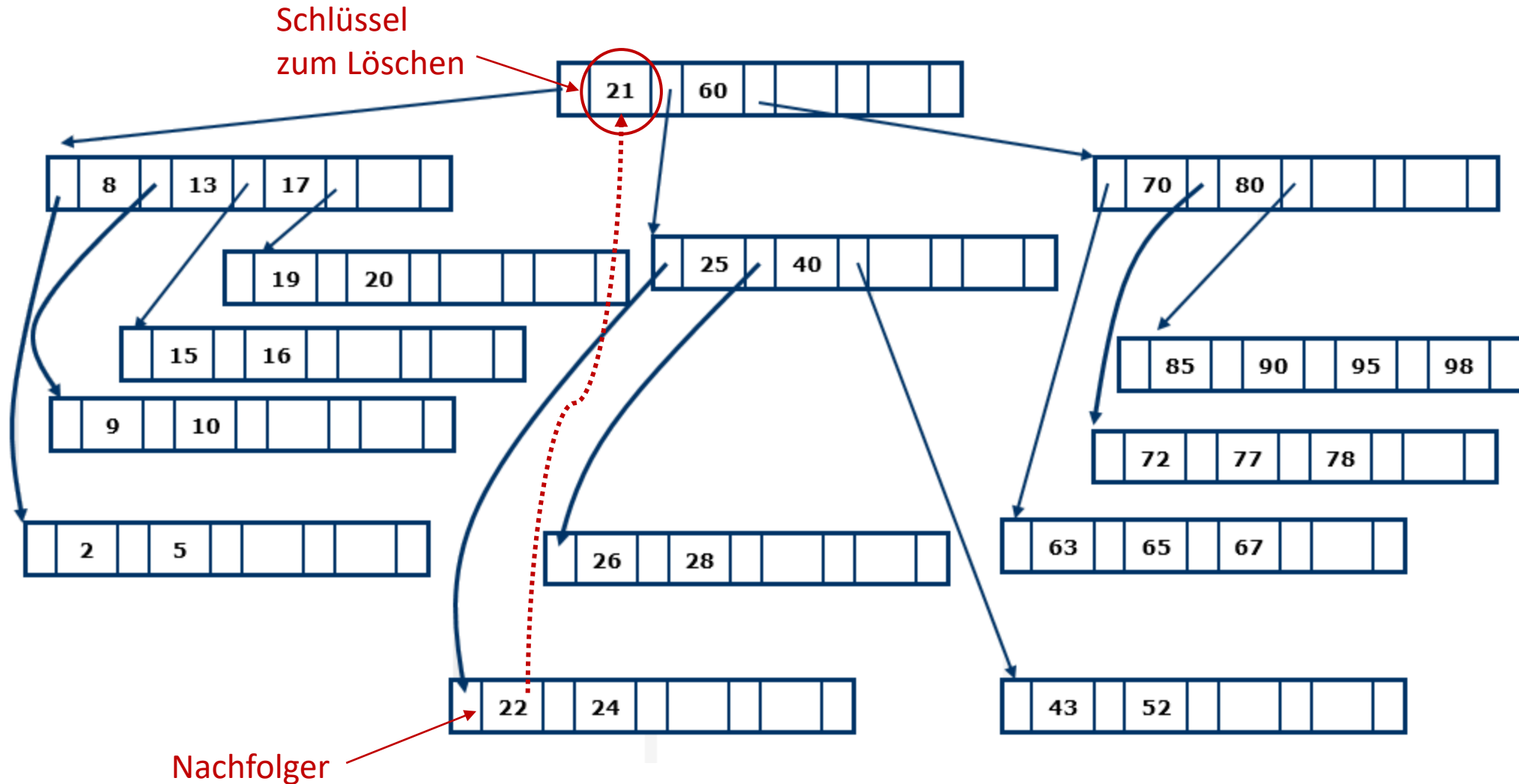
# Löschen in einem B-Baum



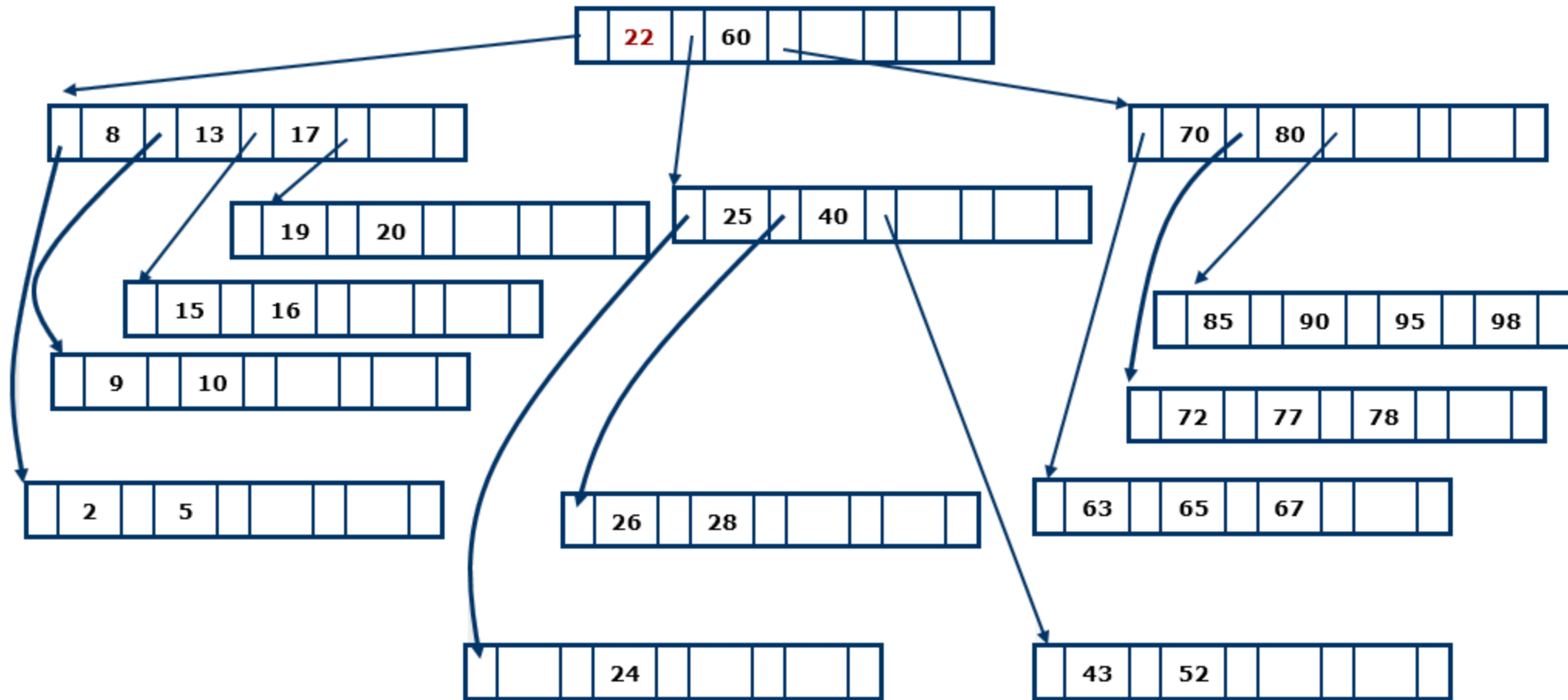
# Löschen in einem B-Baum



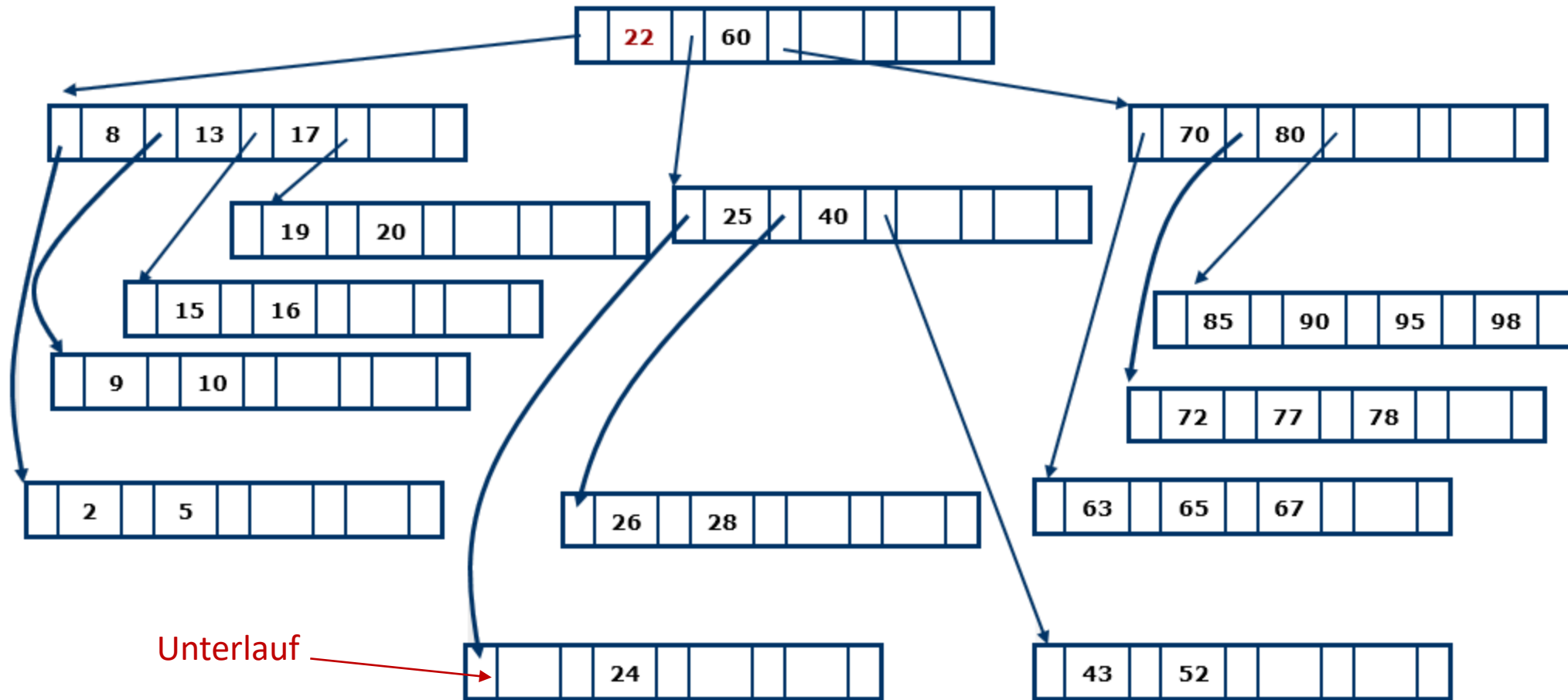
# Löschen in einem B-Baum



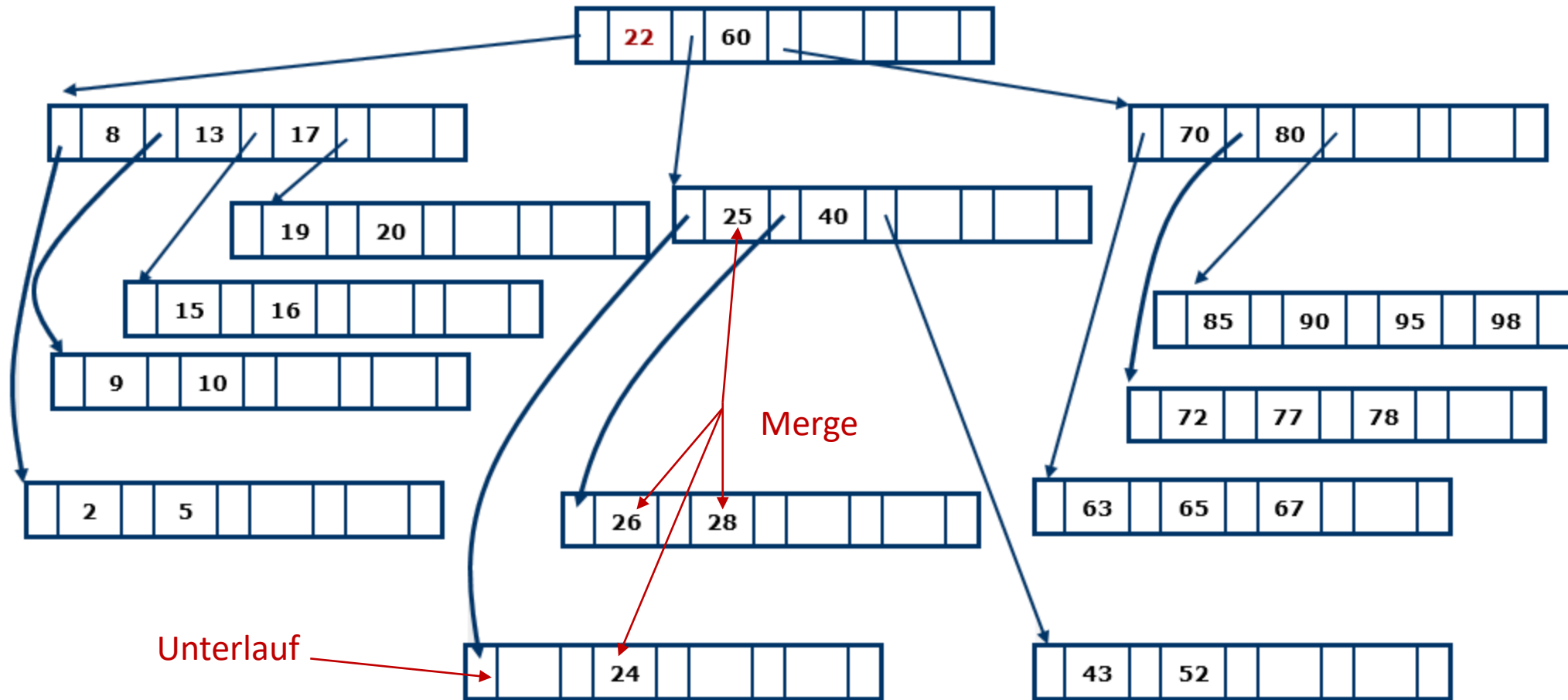
# Löschen in einem B-Baum



# Löschen in einem B-Baum

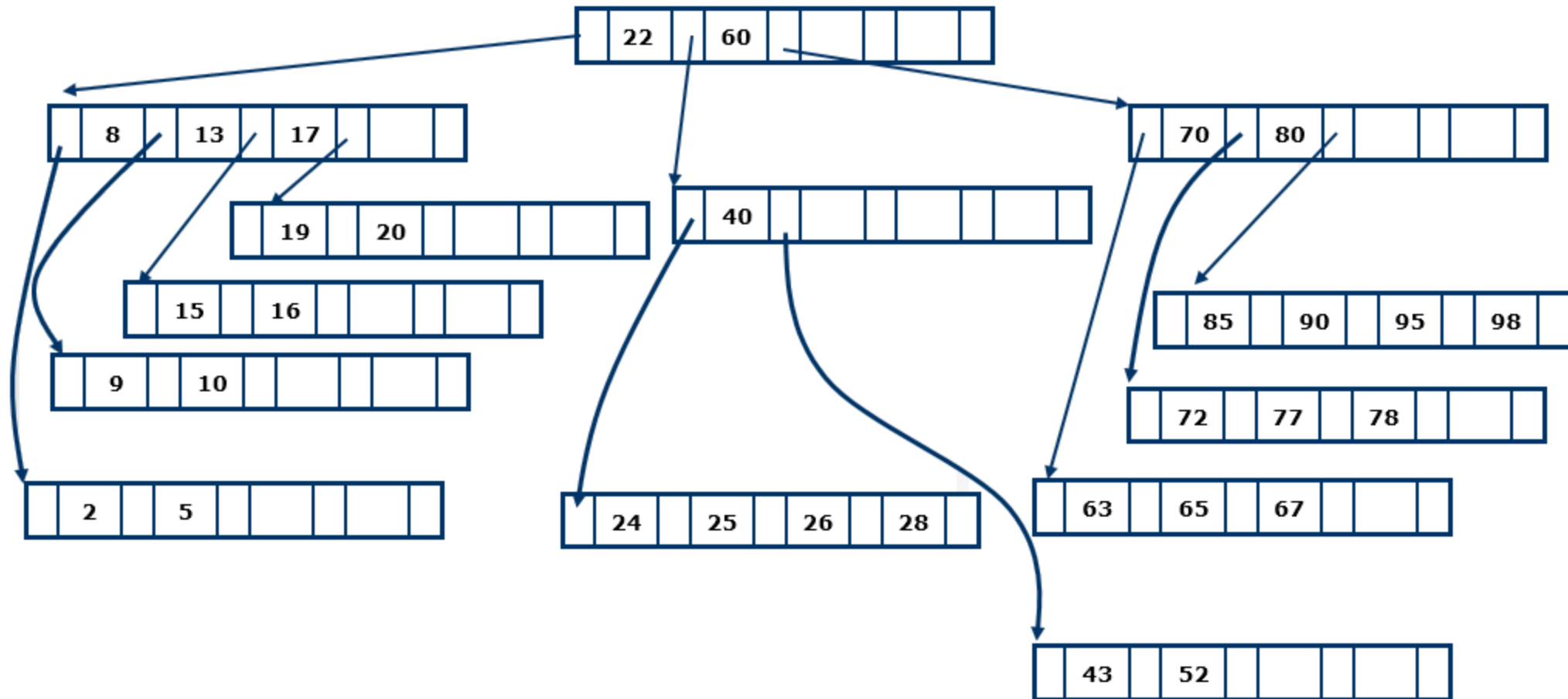


# Löschen in einem B-Baum

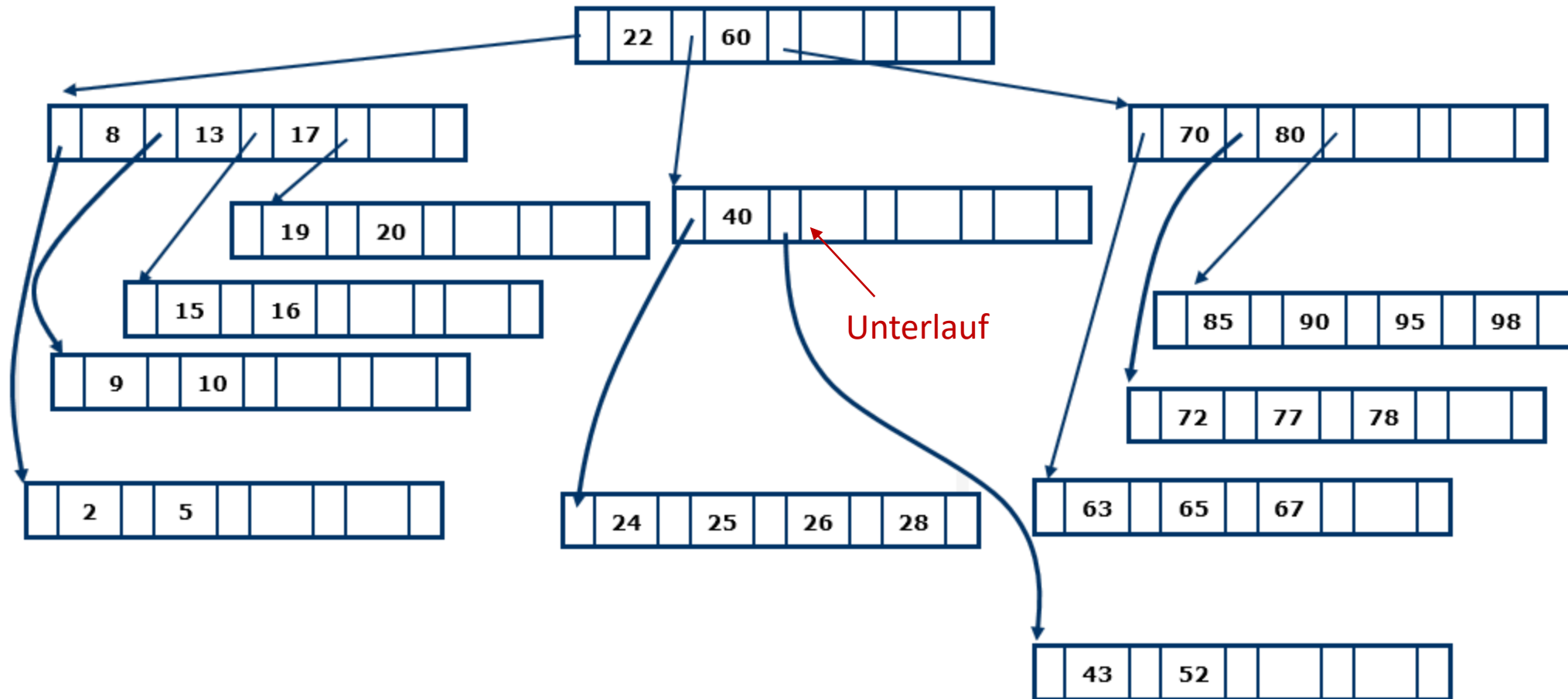




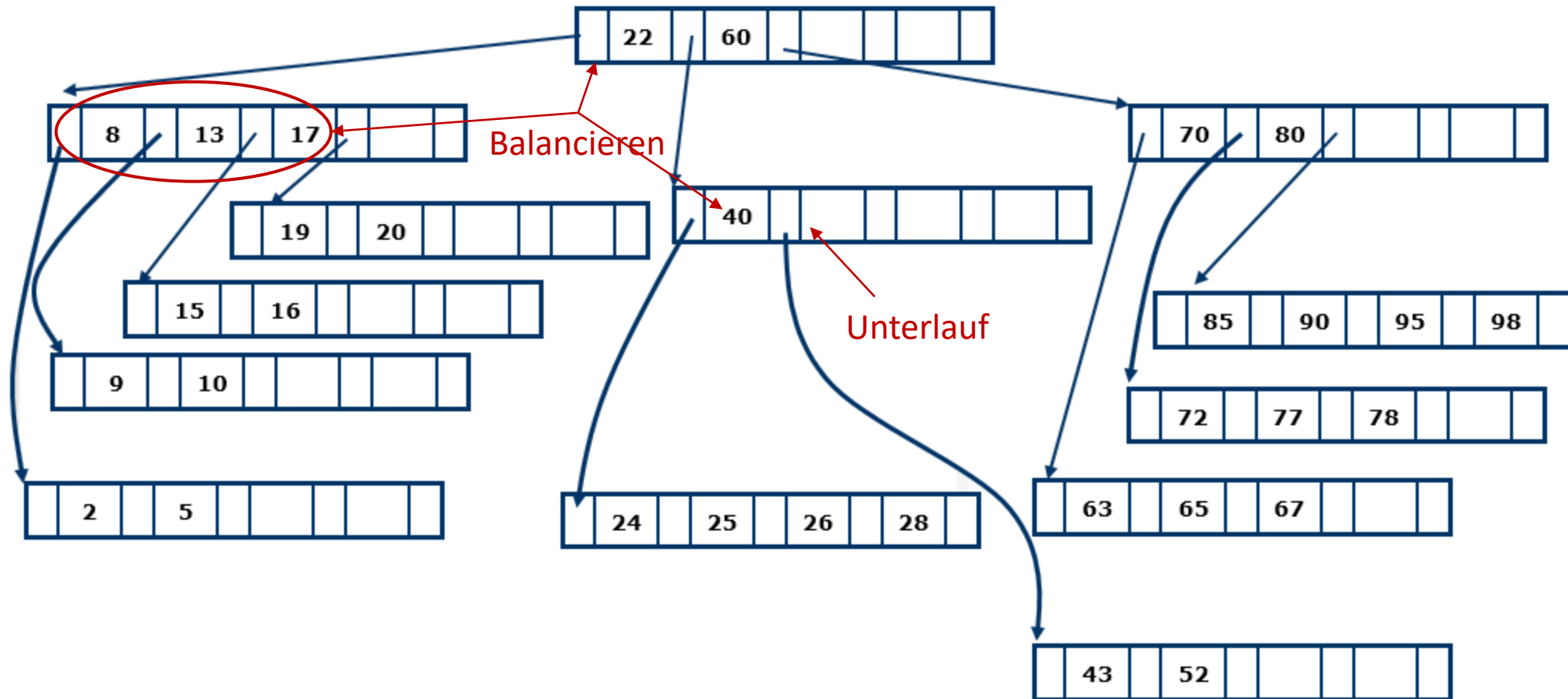
# Löschen in einem B-Baum



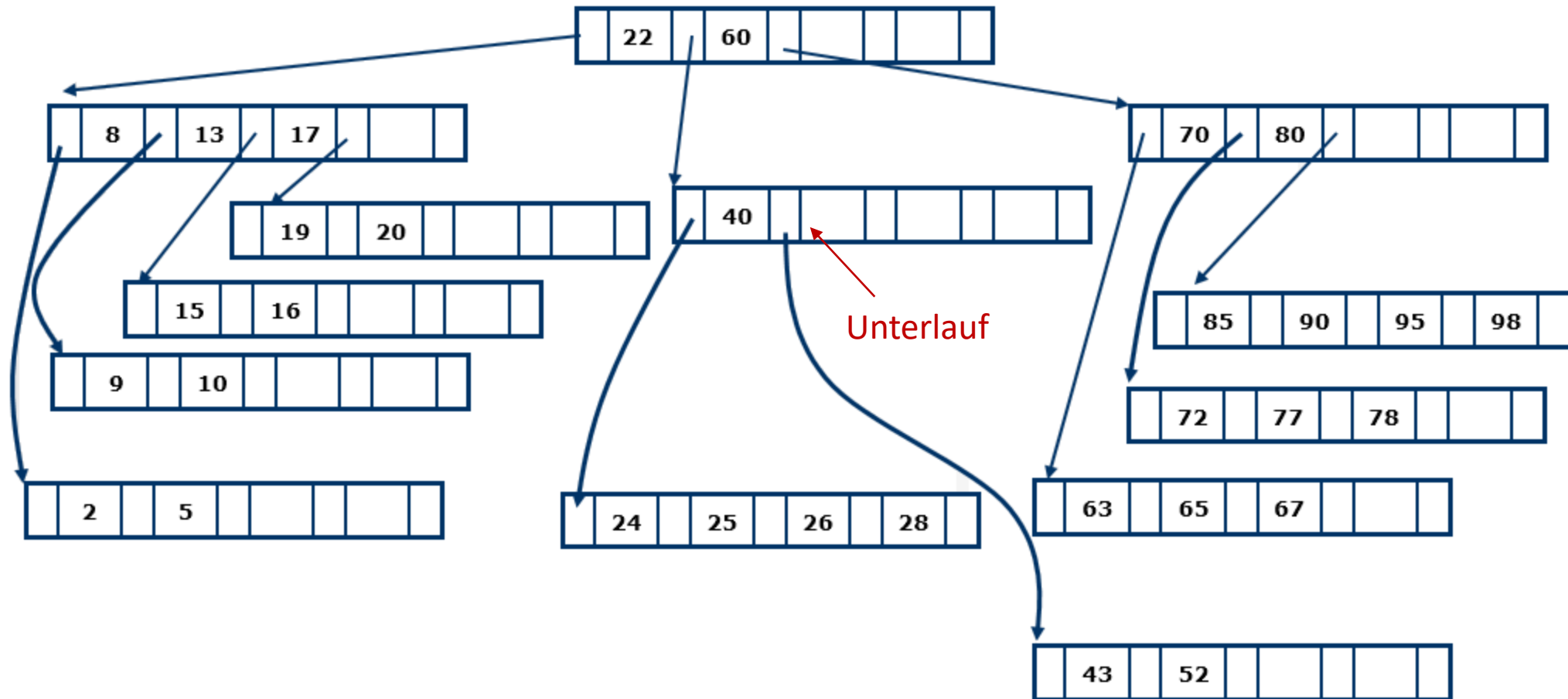
# Löschen in einem B-Baum



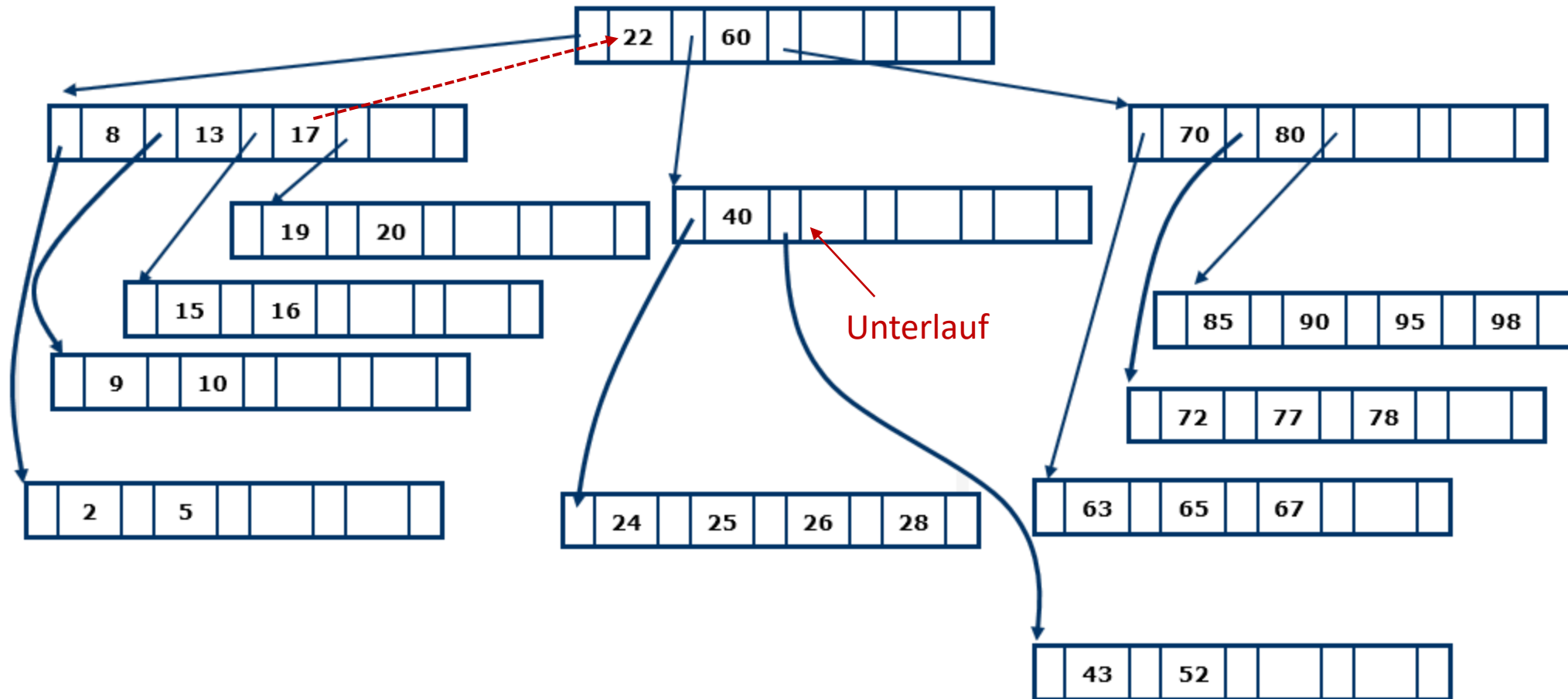
# Löschen in einem B-Baum



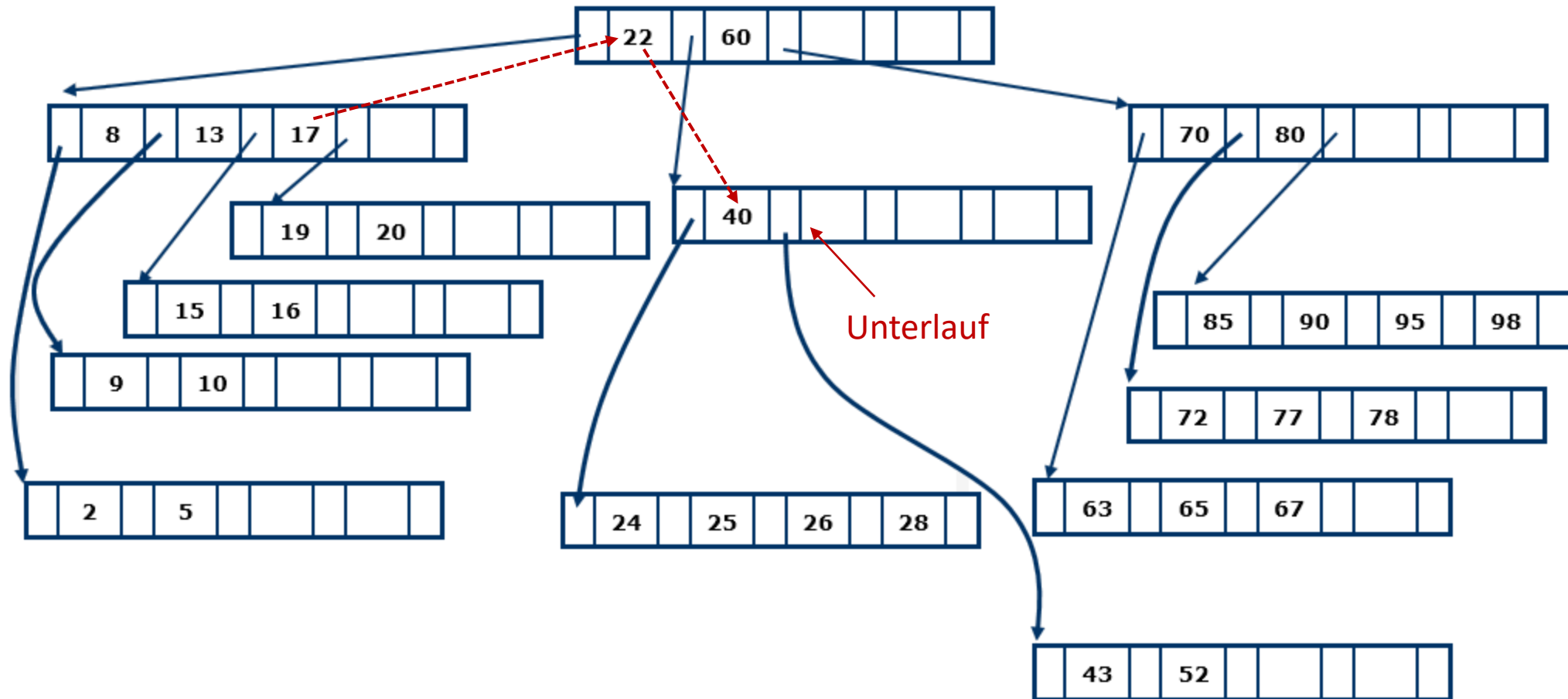
# Löschen in einem B-Baum



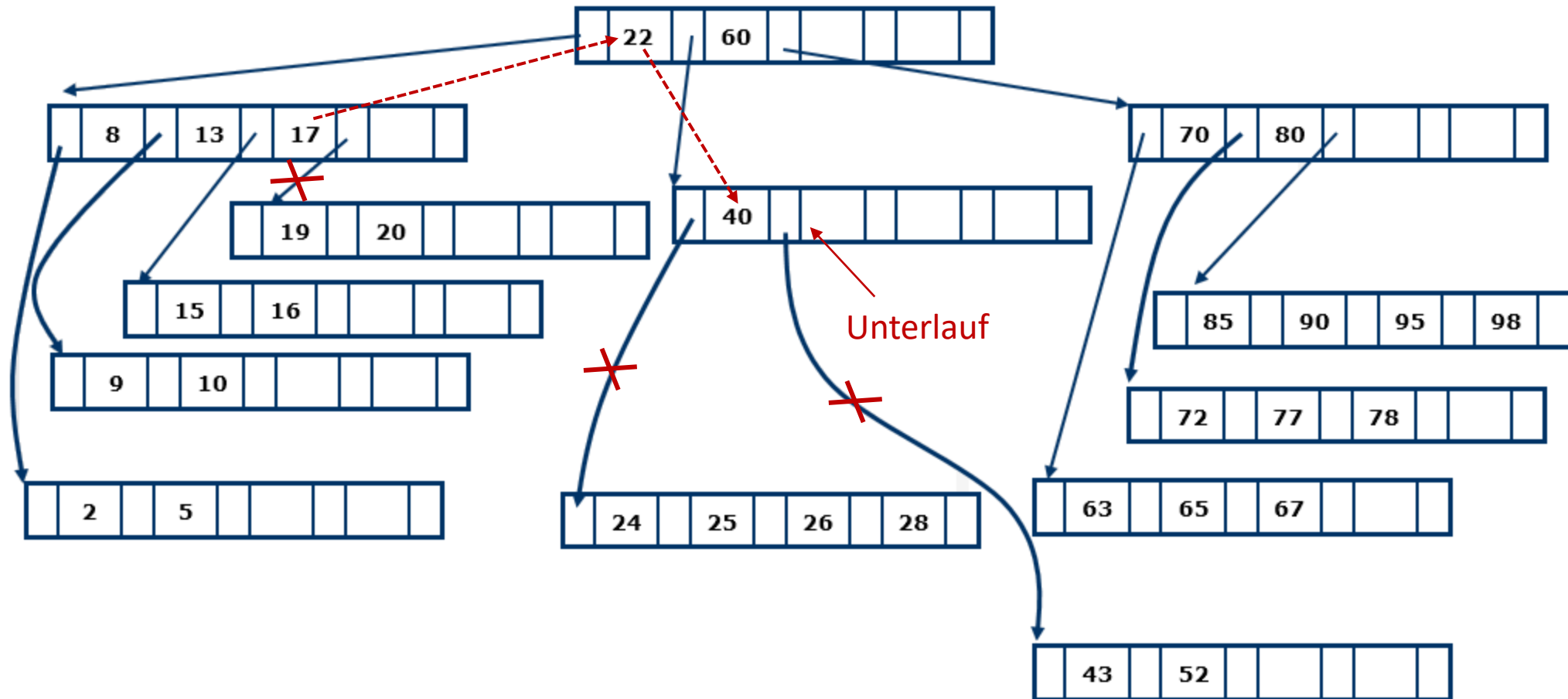
# Löschen in einem B-Baum



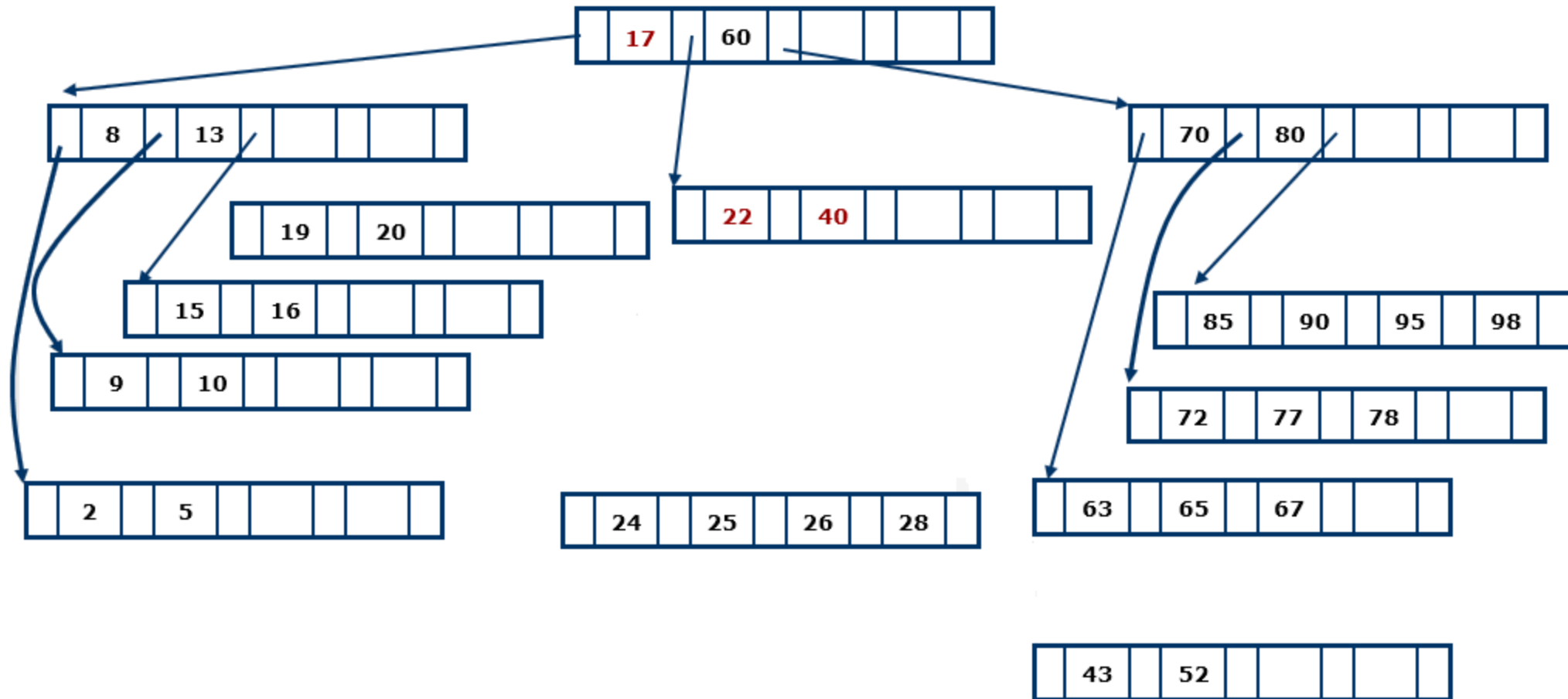
# Löschen in einem B-Baum



# Löschen in einem B-Baum

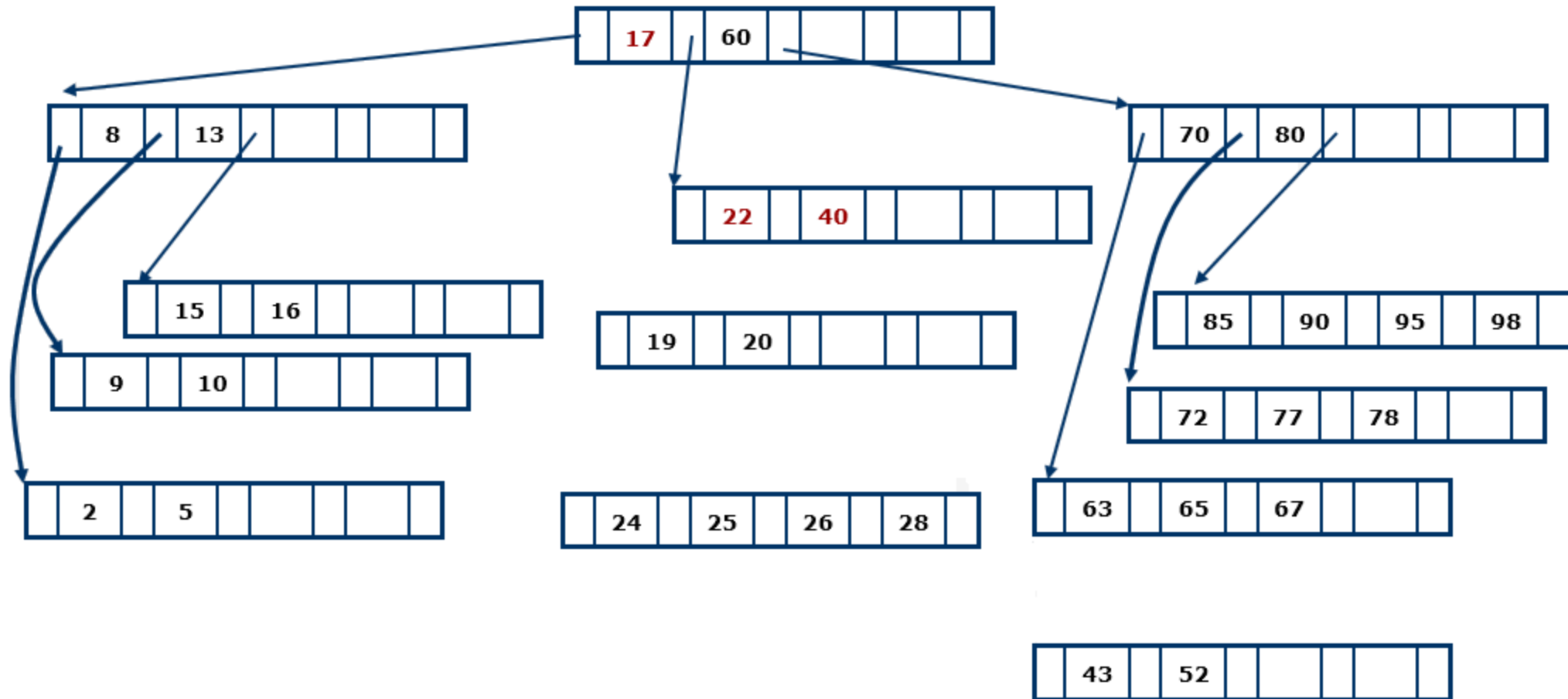


# Löschen in einem B-Baum

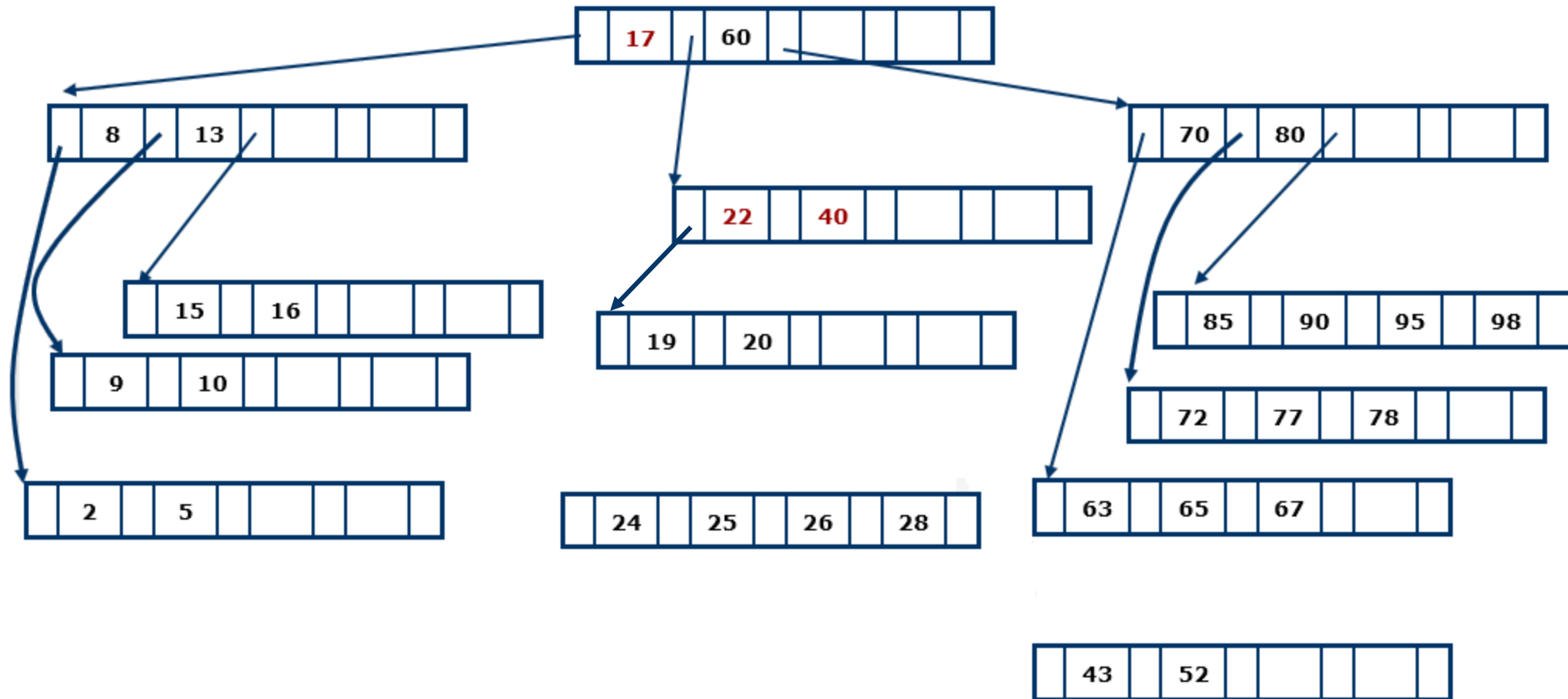




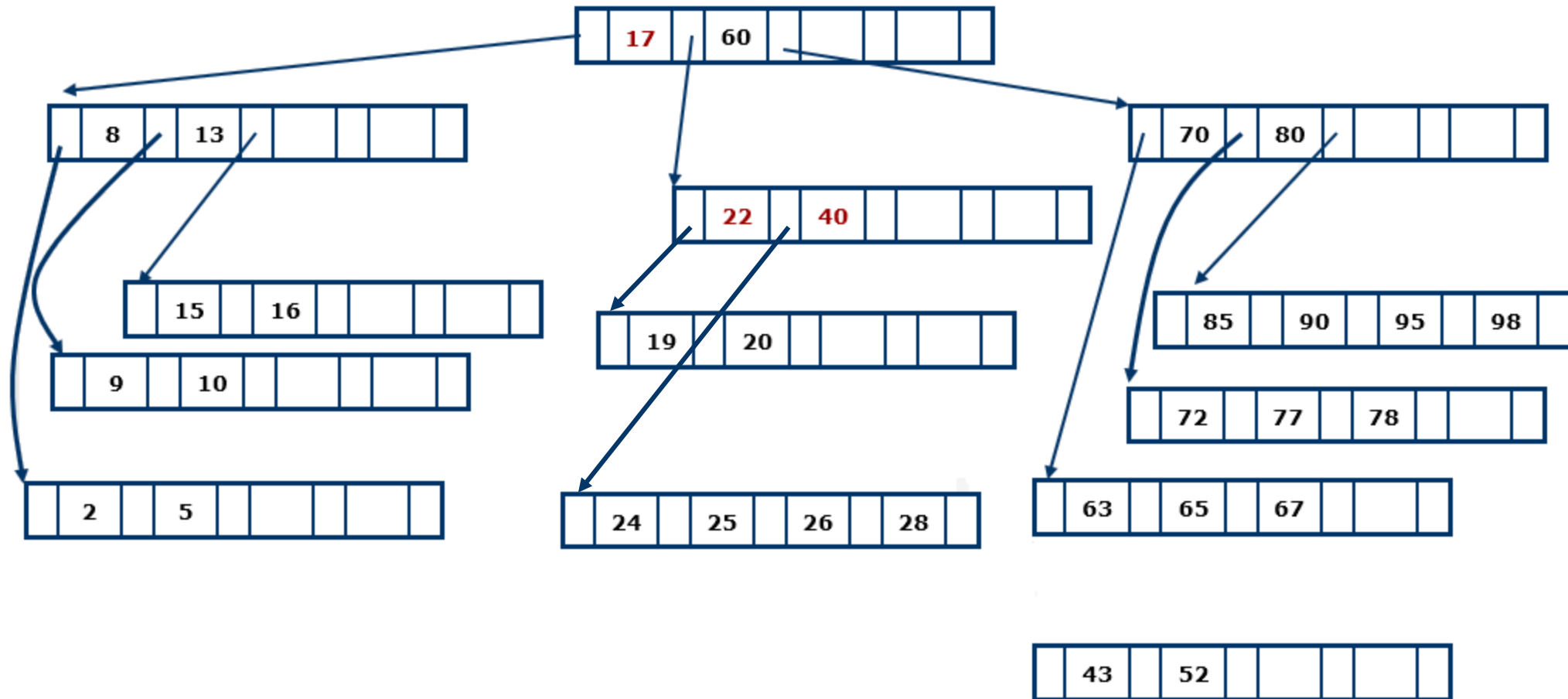
# Löschen in einem B-Baum



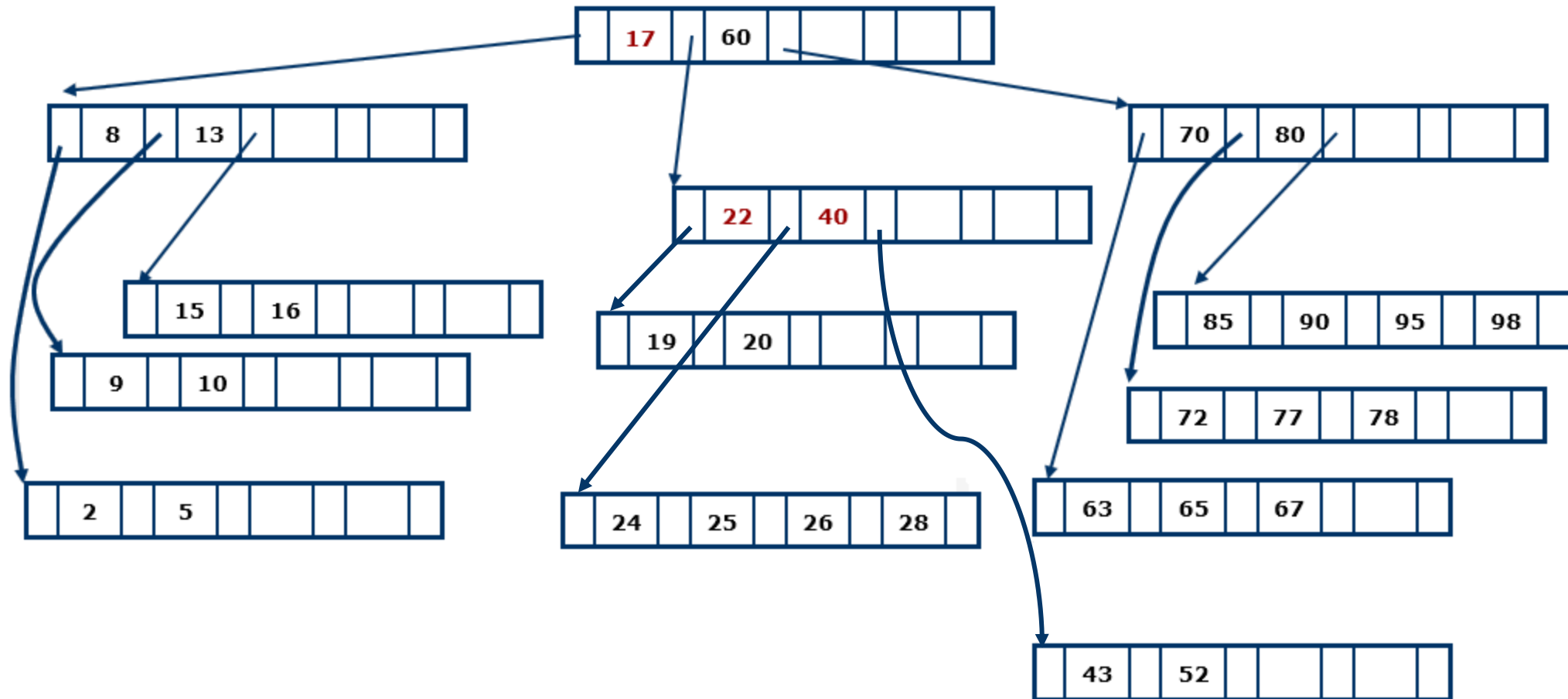
# Löschen in einem B-Baum



# Löschen in einem B-Baum

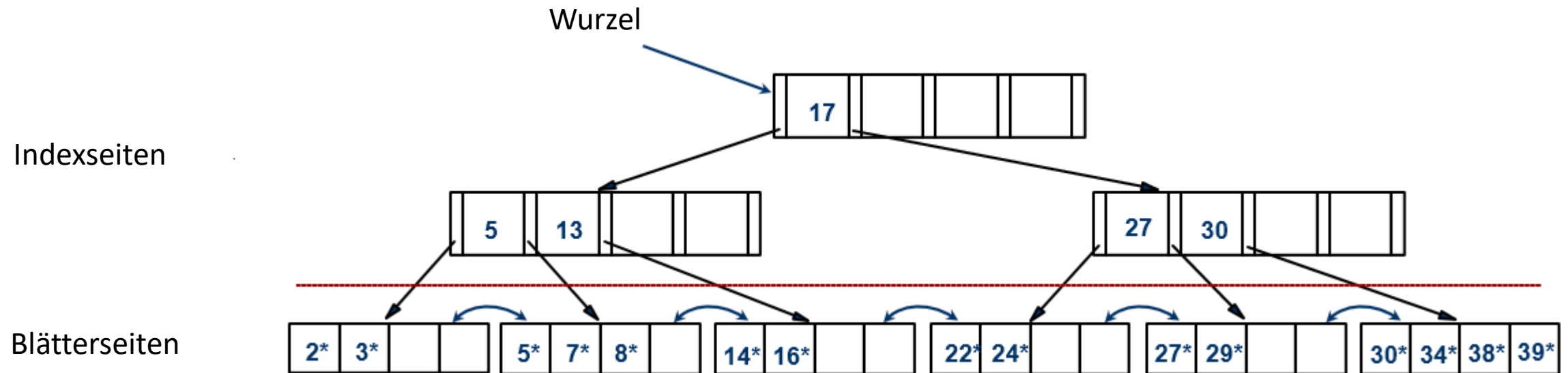


# Löschen in einem B-Baum



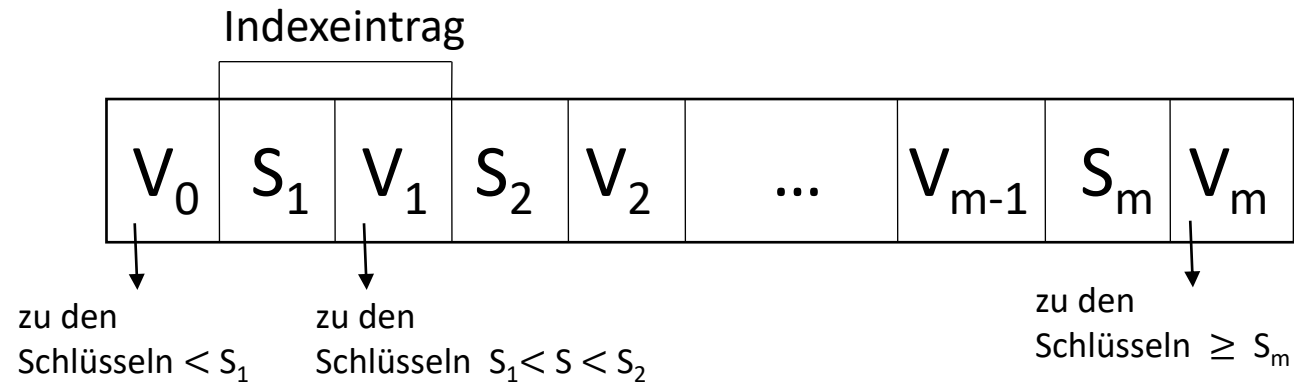
# B<sup>+</sup>-Bäume

- Eine Modifikation von B-Bäumen:
  - Alle Zeiger zu den Datensätzen befinden sich in den Blättern
  - Die Suchschlüssel der inneren Knoten dienen als Wegweiser oder Separatoren bei der Suche der Datensätzen
- Ein B<sup>+</sup>-Baum kann weniger Levels als der entsprechende B-Baum enthalten

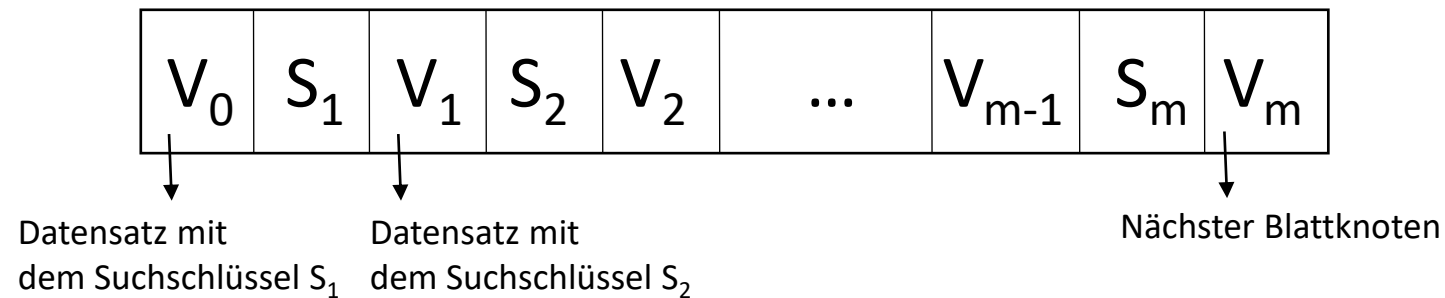


# Die Struktur der Knoten in einem B<sup>+</sup>-Baum

- Innere Knoten:



- Blätterknoten:



# B<sup>+</sup>-Bäume in der Praxis

- Typisches Verzweigungsgrad/Ordnung: 100, Typisches Füllgrad: 67%
  - Typisches Ausgangsgrad/Fan-out (Anzahl von Indexeinträge/ Indexseite) = 133
- Typische Größe:
  - Höhe 4:  $133^4 = 312,900,700$  Datensätze
  - Höhe 3:  $133^3 = 2,352,637$  Datensätze
- Oft können diese in den Top-Levels des Buffer Pools sein:
  - Level 1: 1 Seite = 8 KBytes
  - Level 2: 133 Seiten = 1 MByte
  - Level 3: 17,689 Seiten = 133 MBytes

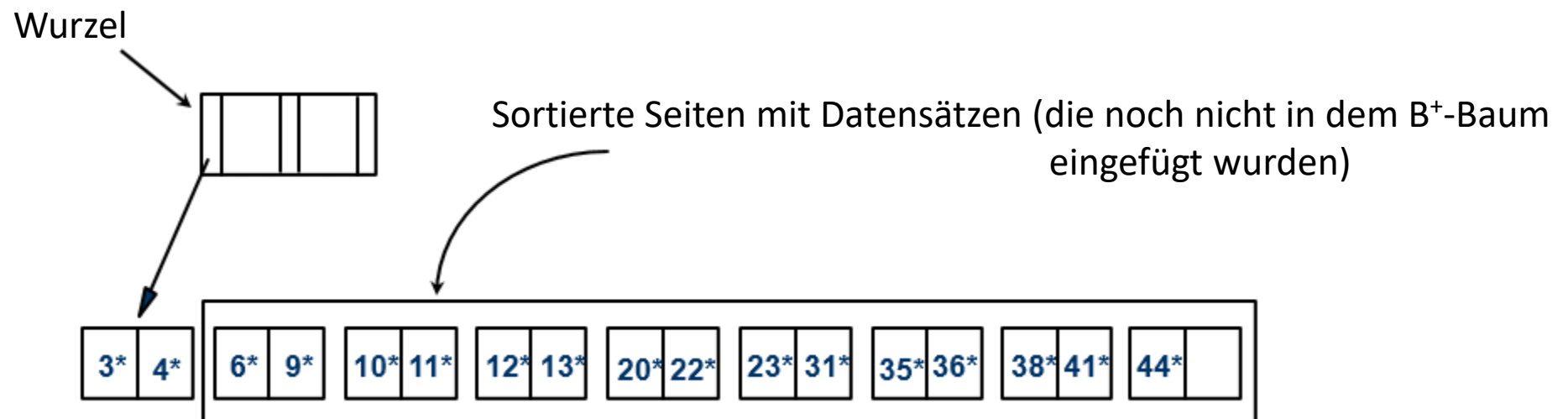
# Vor- und Nachteile der B<sup>+</sup>-Bäume

- Der Index bleibt balanciert → gleichformige Suchzeit
- Selten mehr als 3-5 Levels → man kann den Datensatz in 2-3 I/O Operationen finden
- Am meisten benutzt für Indexe in DBMS wegen der Flexibilität (auch am meisten optimierte in DBMS)
- B<sup>+</sup>-Bäume können für folgende Indexe benutzt werden:
  - geclustert, dünner Index, wenn die Daten sortiert sind
  - nicht-geclustert, dichter Index, wenn die Daten nicht sortiert sind



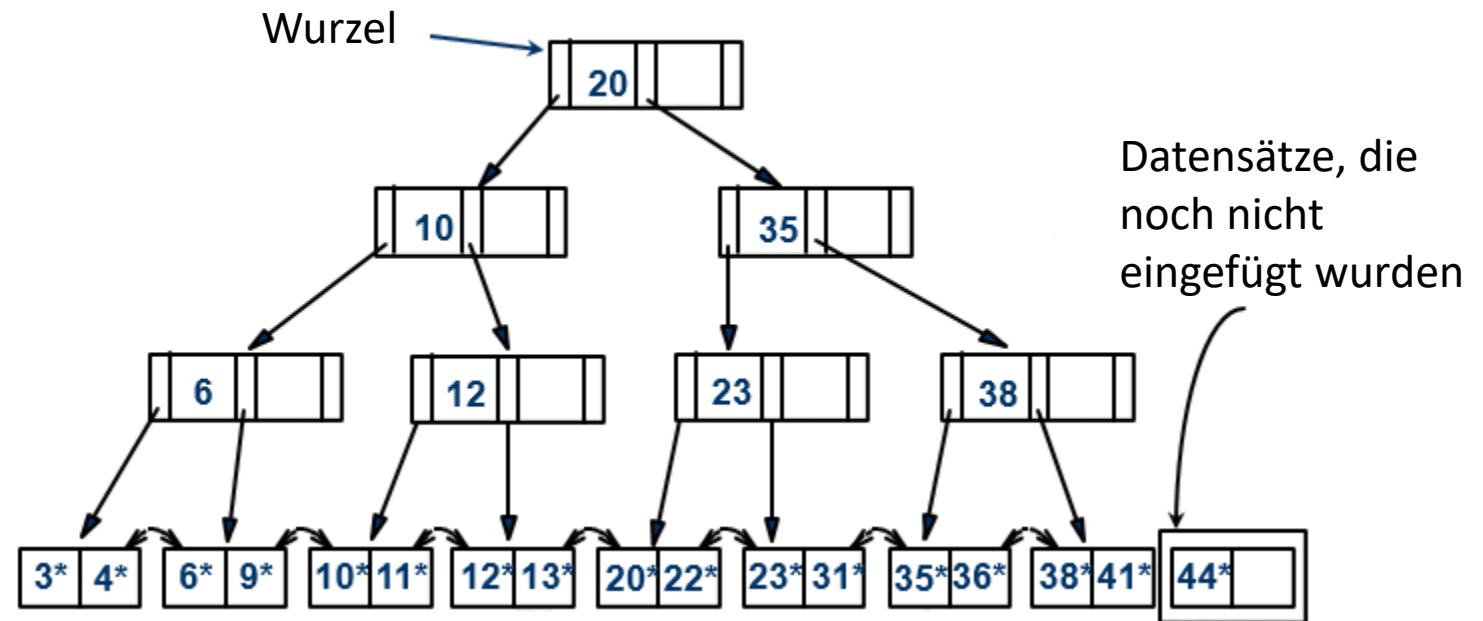
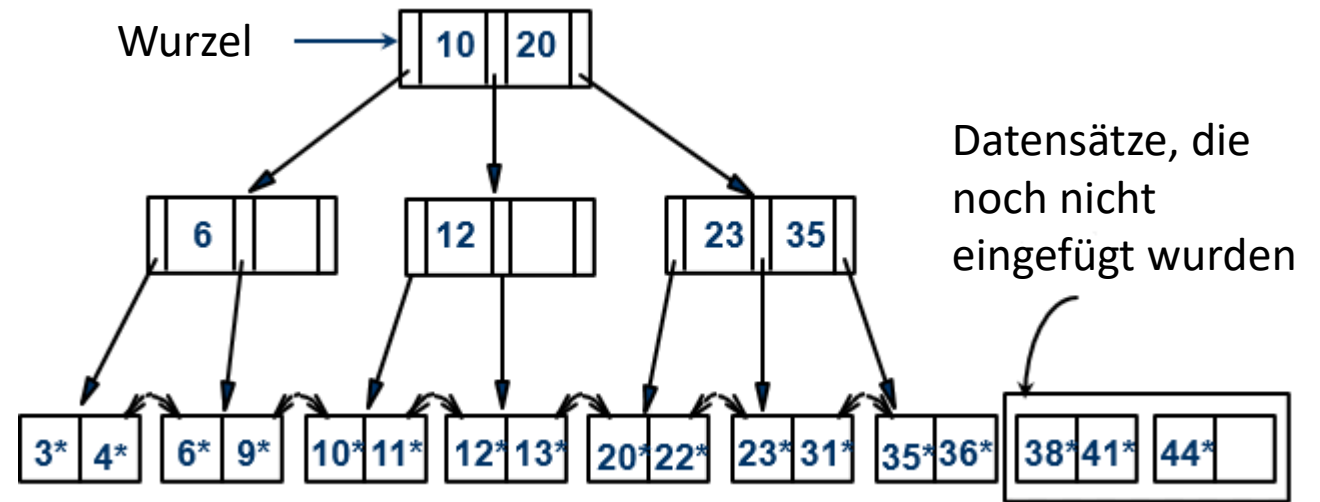
# Bulk Einfügen in einem B<sup>+</sup>- Baum

- Wenn wir einen B<sup>+</sup> - Baum auf einen Datenfeld für eine große Datenmenge erstellen wollen, dann ist es nicht effizient jeden Datensatz einzeln einzufügen → **Bulk Einfügen** ist viel effizienter
- Initialisierung:
  - Sortiere alle Datensätze
  - Erstelle eine neue Wurzelseite, die ein Zeiger zu der ersten Blattseite enthält



# Bulk Einfügen

- Indexeinträge die auf Blätterseiten verweisen werden immer in der am weitesten rechts und der Blätterseite übergeordnete Indexseite eingefügt
- Wenn diese voll ist, dann wird sie geteilt
- Die Behandlung des Überlaufs kann sich bis zu der Wurzel fortpflanzen und die Höhe des Baumes um eins erhöhen



# Bulk Einfügen vs. Einzelne Einfügeoperationen

- 1. Möglichkeit: mehrere einzelne Einfügeoperationen (ein einziger Datensatz)
  - Langsam
  - Die Blätter werden nicht sequentiell abgespeichert
- 2. Möglichkeit: Bulk Einfügen
  - Hat Vorteile für Mehrbenutzersynchronisation (wenigere Sperrungen)
  - Wenigere I/O Operationen um den B<sup>+</sup>- Baum zu erstellen
  - Blätter werden sequentiell abgespeichert
  - Man kann den Füllgrad beeinflussen

# Präfix B<sup>+</sup>-Bäume

- Die Größe der Separatoreinträge  $\Rightarrow$  bestimmt die Anzahl der Indexeinträge in einer Seite  $\Rightarrow$  bestimmt die Höhe des B<sup>+</sup>-Baumes
- Um die Höhe des B<sup>+</sup>-Baumes zu minimieren und den Ausgangsgrad/fan-out zu maximieren müssen möglichst viele Separatoren auf eine Seite passen
- Die Suchschlüssel haben nur die Funktion eines Separators, die eine Suche zu den Blattknoten führt  $\rightarrow$  können meistens komprimiert werden (ein Präfix ist ausreichend)
- Einfügen und Löschen müssen entsprechend geändert werden

# Präfix B<sup>+</sup>-Bäume - Beispiel

- z.B. Benachbarte Indexeinträge mit den Suchschlüsseln:
  - *Dan Yogurt, David Smith, Demy Moore, Davey Jones*
- Wir kürzen diese ab: *David Smith* → *Dav.*, *Davey Jones* → *Dave.*, ...
- Wie vergleichen wir jetzt *David Smith* mit *Davey Jones*?
- Ist jetzt „*Dav.*“ < „*Dave.*“?
- Wenn man eine Schlüsselkompression auswählt, dann muss die Ordnung zwischen den Suchschlüsseln aufbewahrt werden

# Verzweigungsgrad/Ordnung des B<sup>+</sup>-Baumes in der Praxis

- In der Praxis ersetzt man das Konzept von Verzweigungsgrad/Ordnung mit **Füllgrad** (wenigstens halb voll)
- Die Indexseiten können viel mehrere Einträge enthalten als die Blätterseiten
- Wenn die Länge der Suchschlüssel und der Datensätze variable ist, dann können Seiten unterschiedliche Anzahl von Einträgen enthalten
- Auch wenn die Länge nicht variable ist, kann es sein dass zwei Datensätze denselben Suchschlüsselwert haben → mit Alternative (3) wird die Länge der Indexeinträge variable

# Zusammenfassung

- Baum-strukturierte Indexe sind sehr effizient für Bereichsanfragen, aber auch gut für Gleichheitsanfragen
- ISAM (Index-Sequential Access Method) ist eine statische Struktur
  - Nur die Blätterseiten ändern sich, man braucht Überlaufsseiten
  - Überlaufsseiten können die Effizienz erniedrigen, außer wenn sich die Daten nicht oft ändern
- B<sup>+</sup>-Baum ist eine dynamische Struktur
  - Nach einem Einfügen oder Löschen bleibt der Baum balanciert
  - Großer Ausgangsgrad/fan-out → oft ist die Tiefe nicht mehr als 3 oder 4
  - Fast immer besser als die Datei sortiert zu behalten
  - Typischerweise Füllgrad 67%
  - Meistens besser als ISAM

# Zusammenfassung

- Präfix B<sup>+</sup>-Baum
  - Schlüsselkompression vergrößert den Ausgangsgrad/fan-out und verkleinert die Höhe
- Bulk-Einfügen kann viel schneller sein als wiederholte einzelne Einfüge-Operationen bei dem Erstellen eines B<sup>+</sup>-Baumes
- B<sup>+</sup>-Bäume sind am meisten benutzte Indexe in DBMS wegen der Flexibilität und der hohen Optimierung