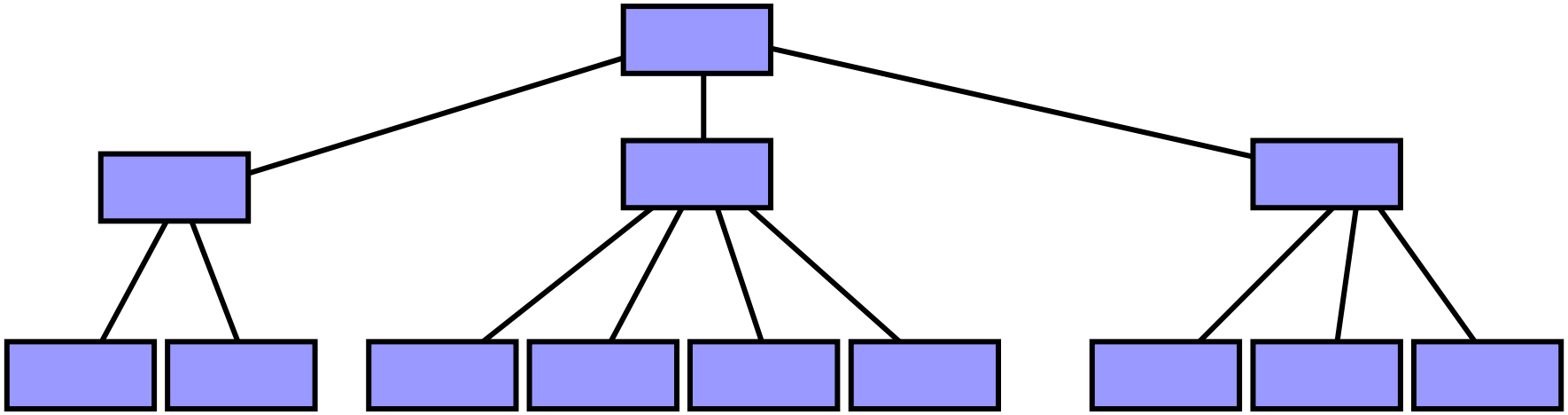
Datenbanksysteme SS 2011

noch Kapitel 4:
Physikalische Datenorganisation
Vorlesung vom 02.05.2011

Oliver Vornberger

Institut für Informatik
Universität Osnabrück

B*-Baum



- Jeder Weg von der Wurzel zu einem Blatt hat dieselbe Länge.
- Jeder Knoten außer der Wurzel und den Blättern hat mindestens *k* Nachfolger.
- Jeder Knoten hat höchstens 2 • *k* Nachfolger.
- Die Wurzel hat keinen oder mindestens 2 Nachfolger.

B*-Baum-Adressierung

Ein Knoten wird auf einem Block gespeichert

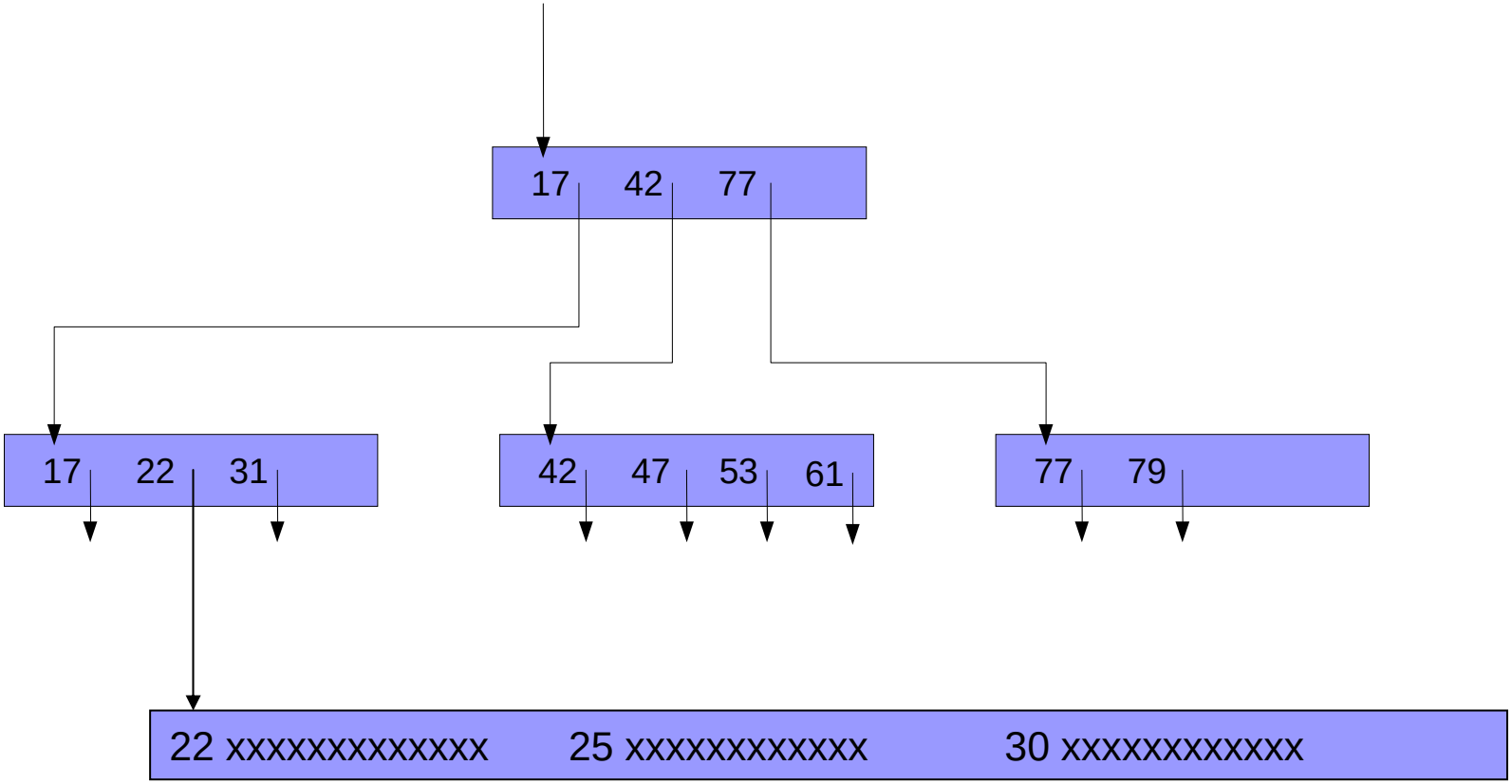
Ein Knoten mit j Nachfolgern ($j \leq 2 \bullet k$) speichert j Paare von Schlüsseln und Adressen (s_1, a_1), . . . , (s_j, a_j).

Es gilt $s_1 \leq s_2 \leq . . . \leq s_j$.

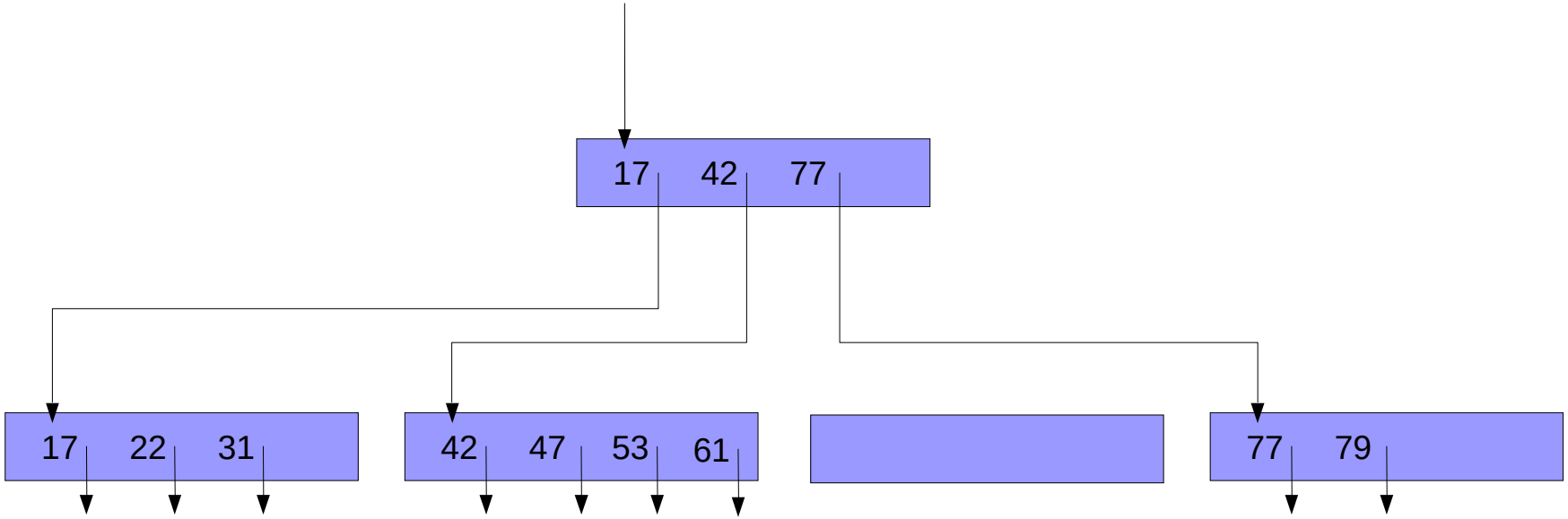
Eine Adresse in einem Blattknoten führt zum Datenblock mit den restlichen Informationen zum zugehörigen Schlüssel

Eine Adresse in einem anderen Knoten führt zu einem Baumknoten

B*-Baum: Lookup



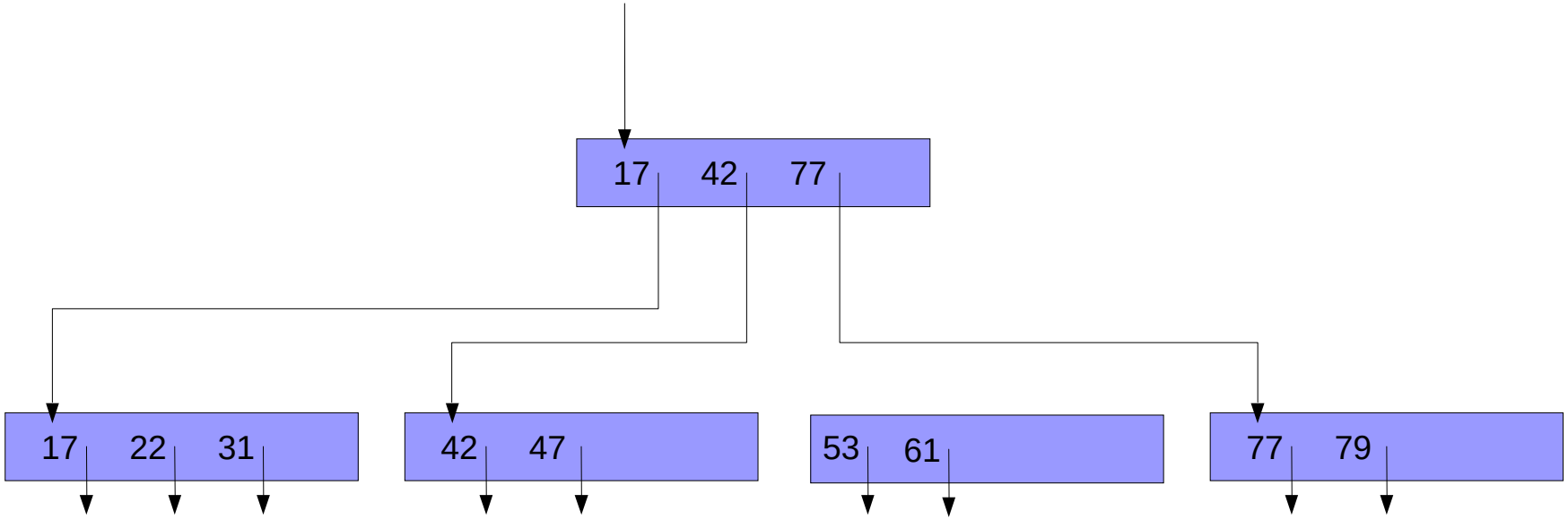
B*Baum: Insert



eingefügt werden soll: 45 Block füllen

Block anfordern

B*Baum: Insert



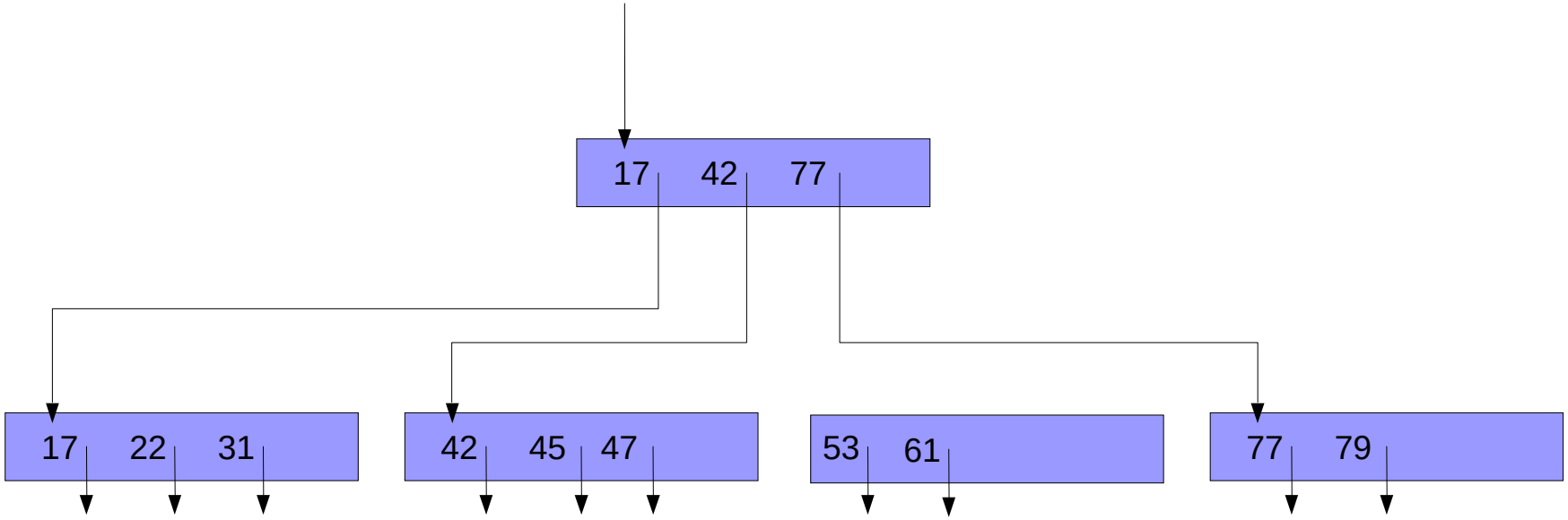
eingefügt werden soll: 45

Block füllen

Block anfordern

Element einordnen

B*Baum: Insert



eingefügt werden soll: 45

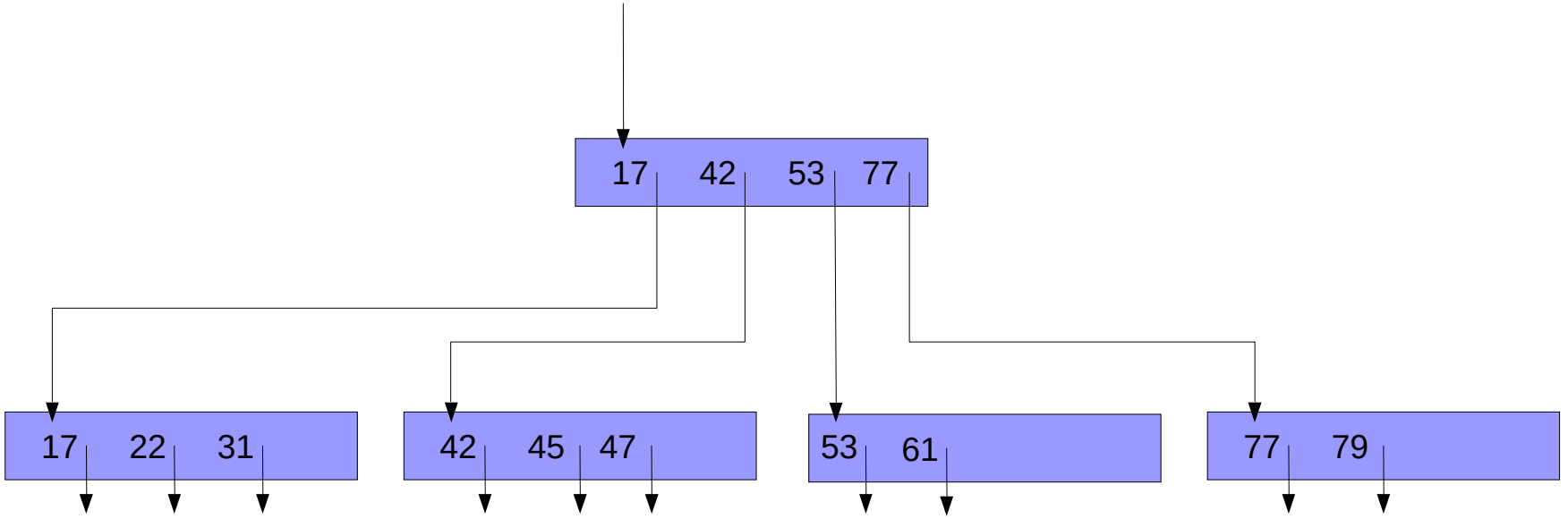
Block füllen

Vorgänger korrigieren

Block anfordern

Element einordnen

B*Baum: Insert



eingefügt werden soll: 45

Block füllen

Vorgänger korrigieren

Block anfordern

Element einordnen

Umrühren - Fertig !

Sequenz für B*-Baum mit k=2

27 55 12 94 37 88 72 39 25 88 74 58 64

27

27

27 55

27

27 55

```
27 55
```

```
27  55  12
```

```
27 55
```

27 55 12

12 27 55

27 55 12 94

12 27 55

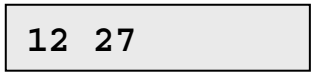
27 55 12 94

12 27 55 94

27 55 12 94 37

12 27 55 94

27 55 12 94 37

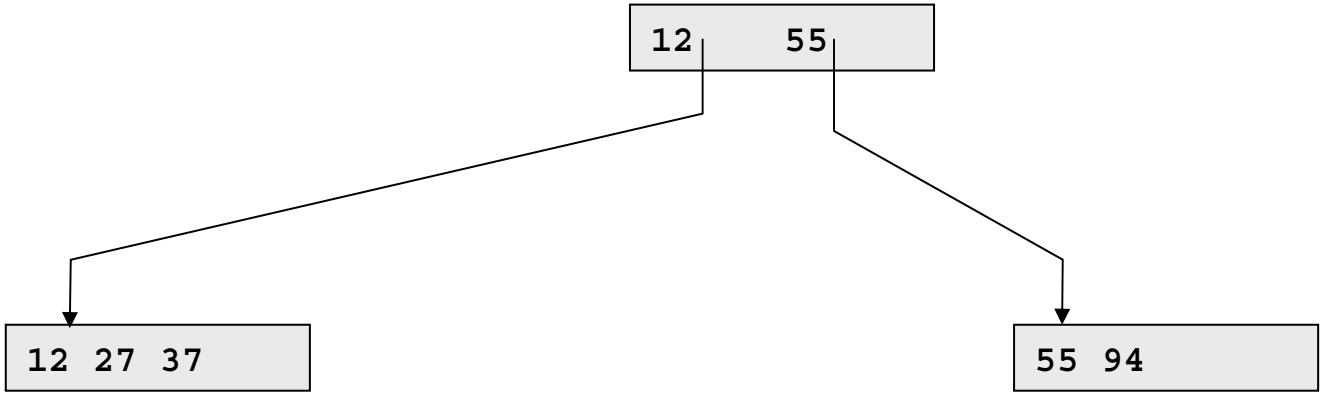


27 55 12 94 37

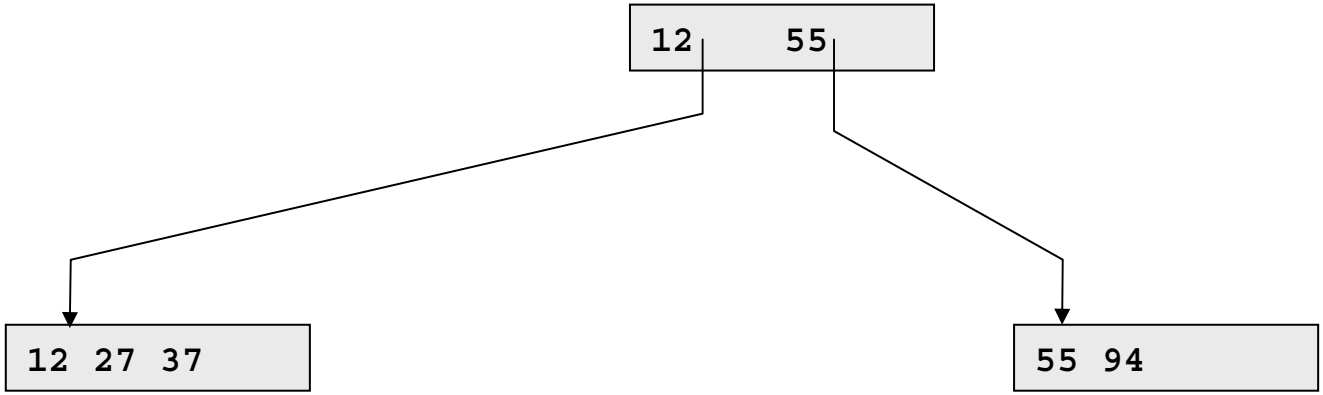
12 27 37

55 94

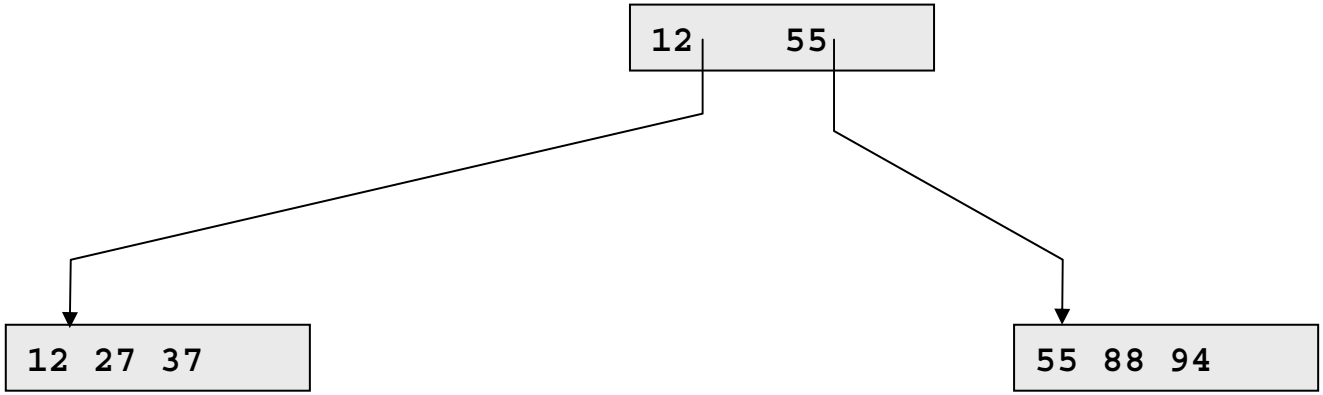
27 55 12 94 37



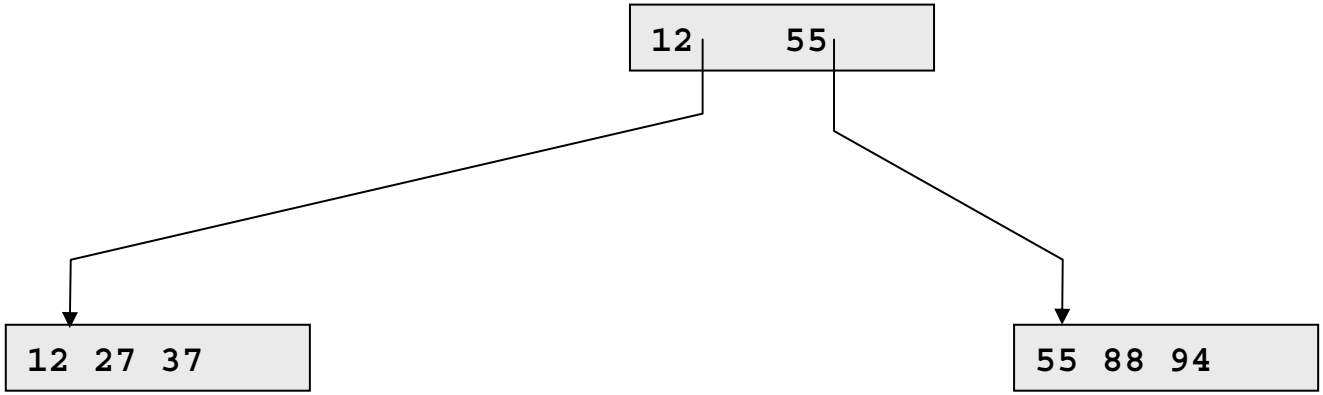
27 55 12 94 37 88



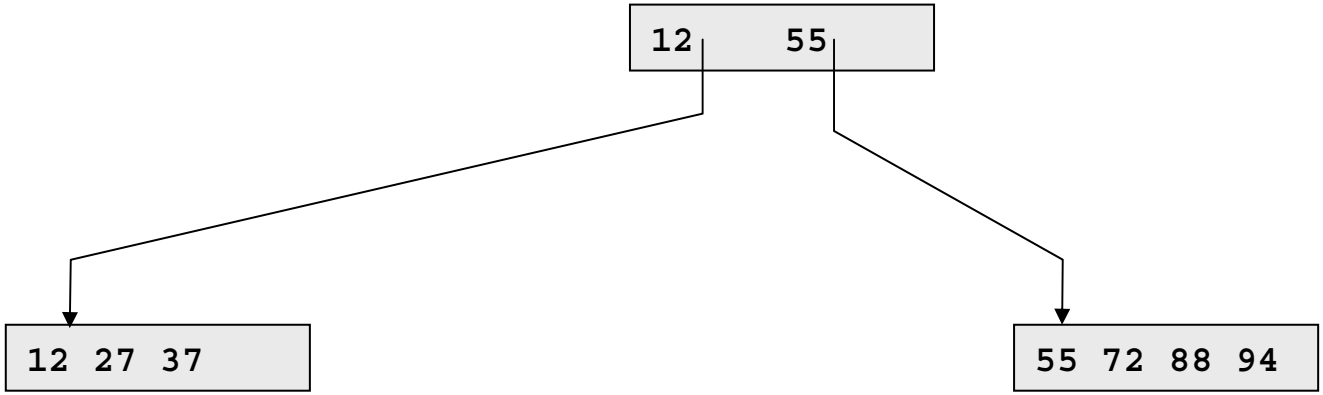
27 55 12 94 37 88



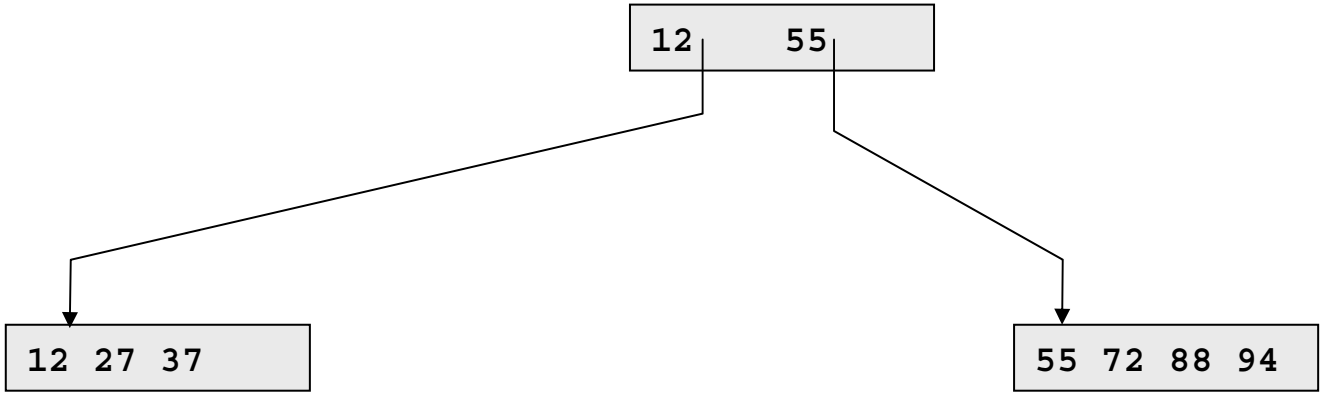
27 55 12 94 37 88 72



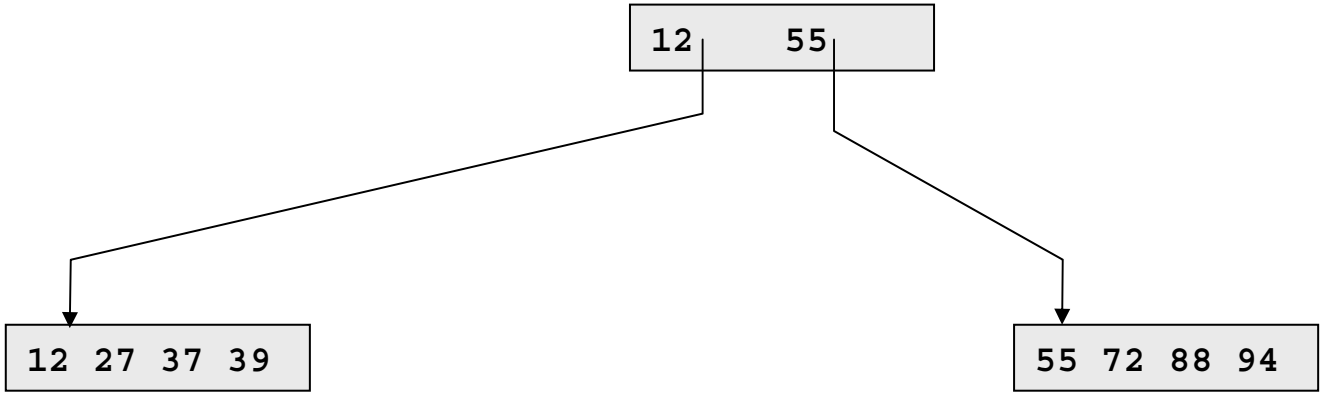
27 55 12 94 37 88 72



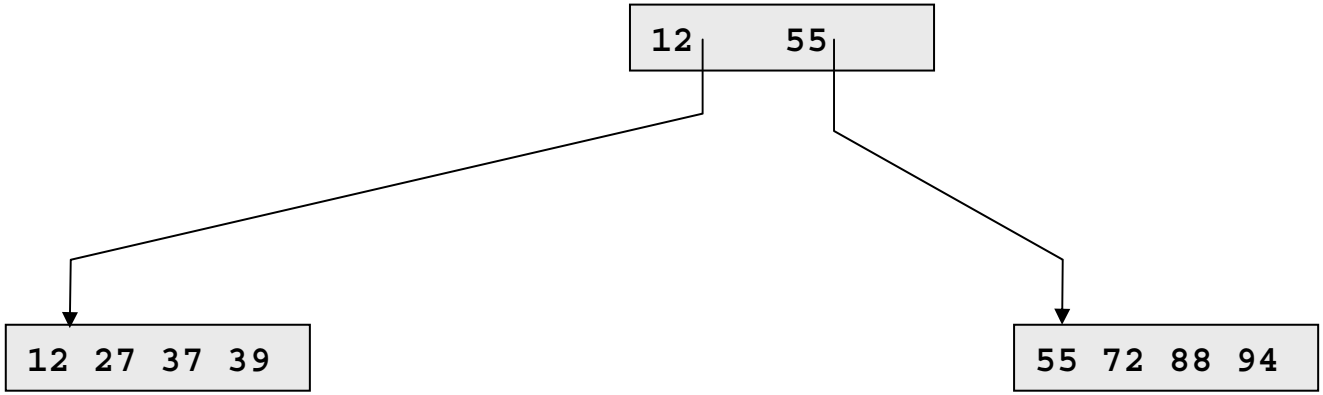
27 55 12 94 37 88 72 39



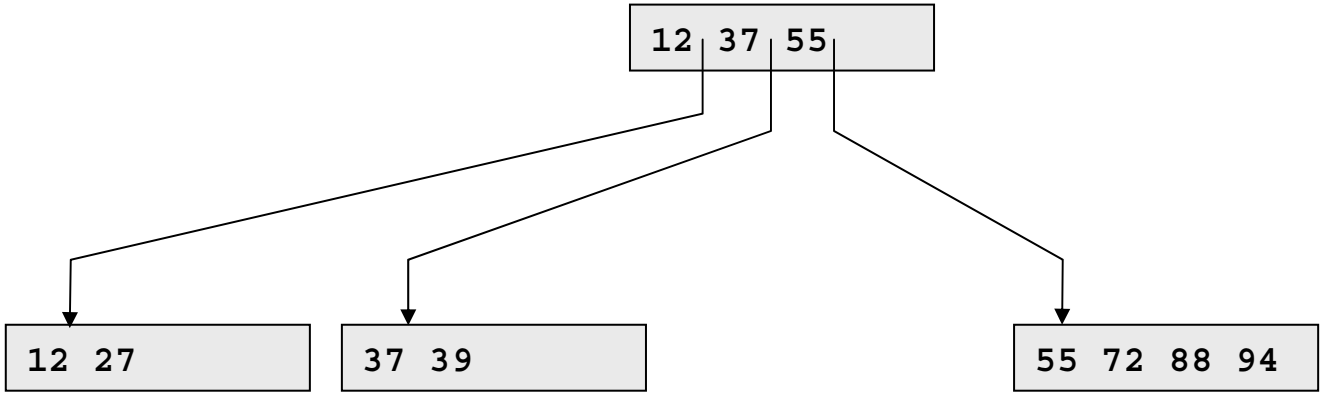
27 55 12 94 37 88 72 39



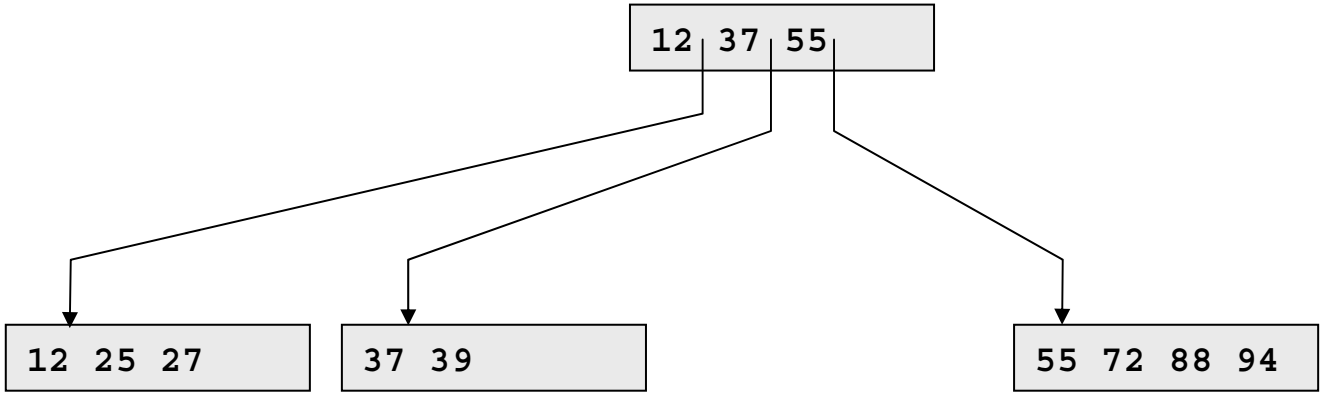
27 55 12 94 37 88 72 39 25



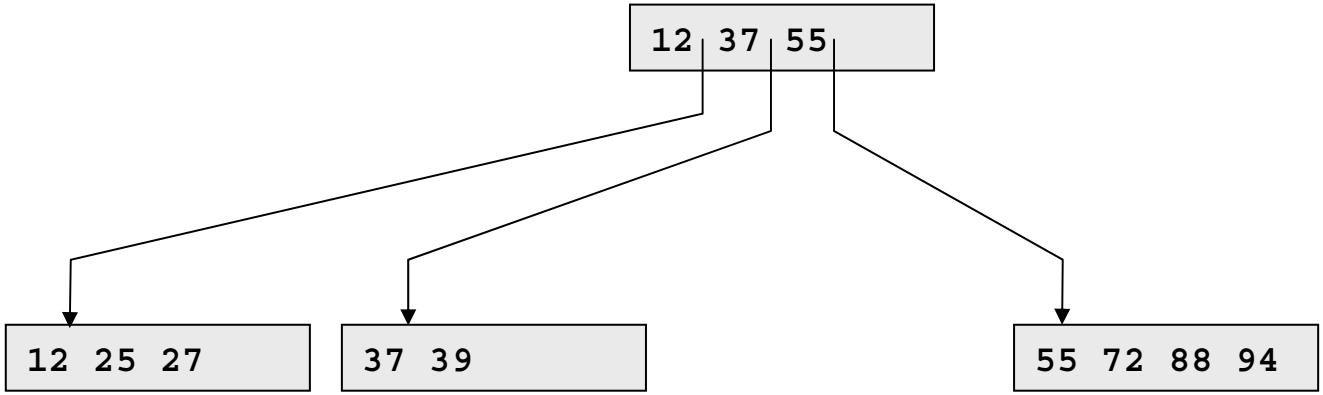
27 55 12 94 37 88 72 39 25



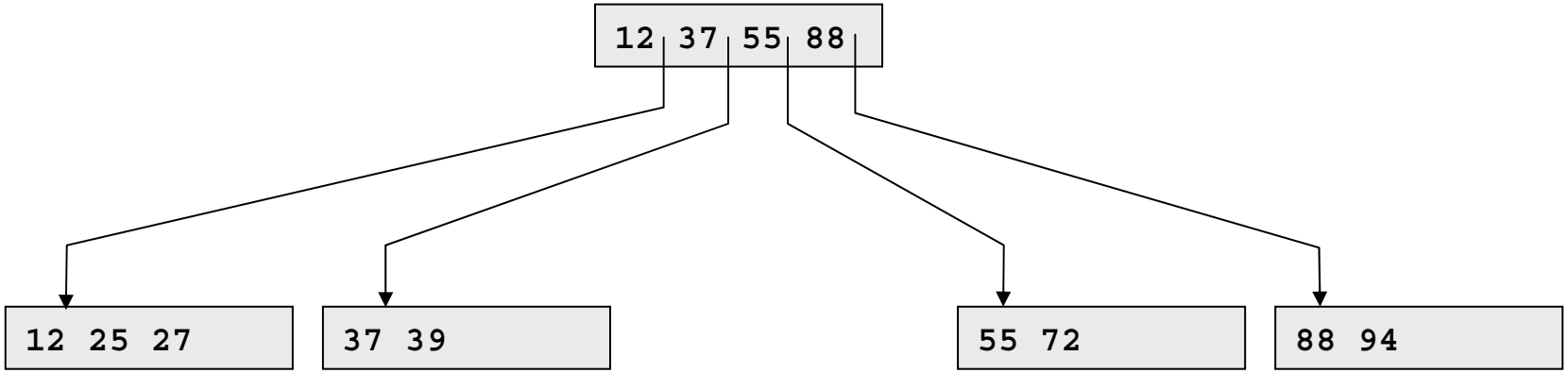
27 55 12 94 37 88 72 39 25



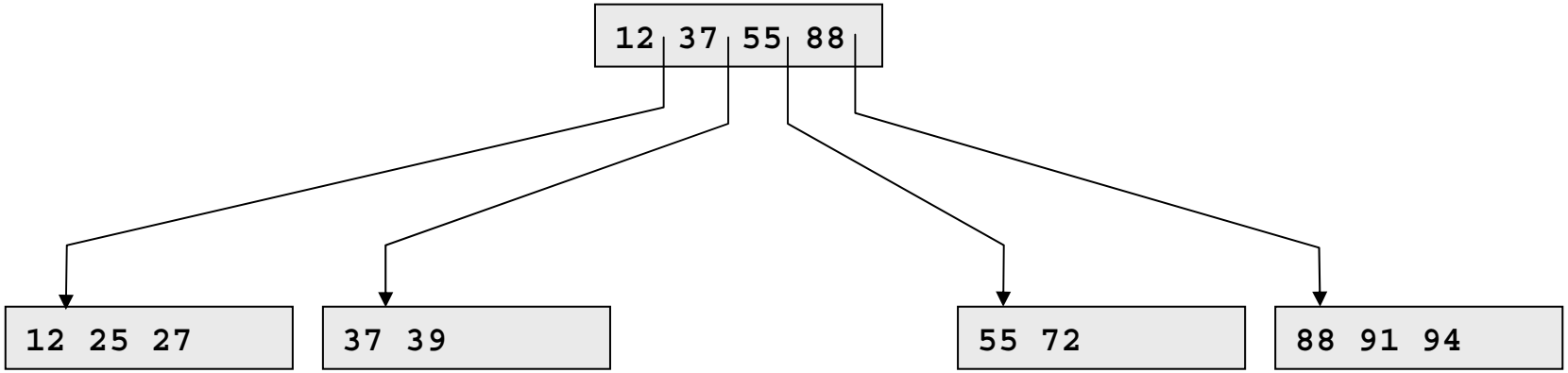
27 55 12 94 37 88 72 39 25 91



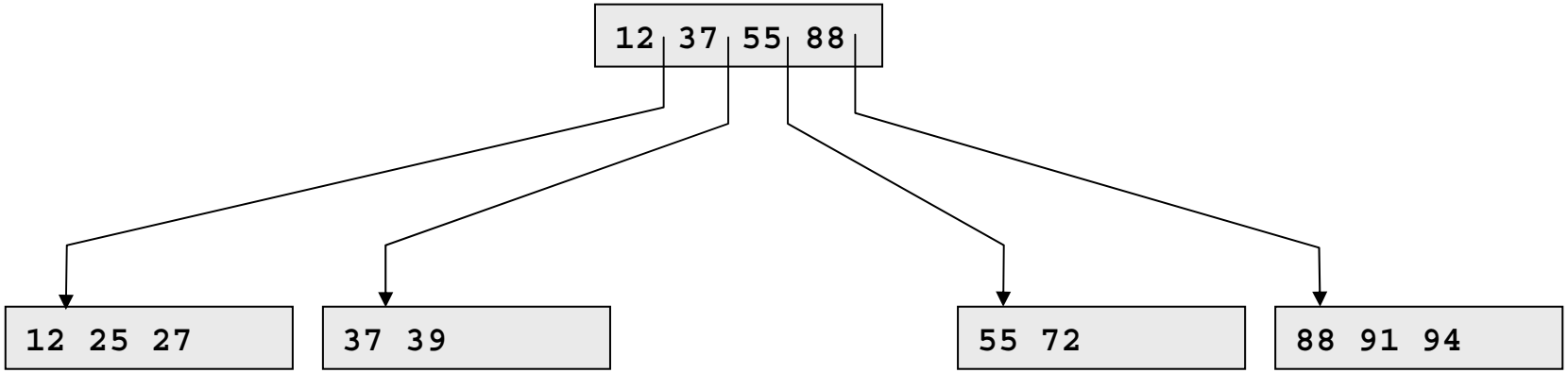
27 55 12 94 37 88 72 39 25 91



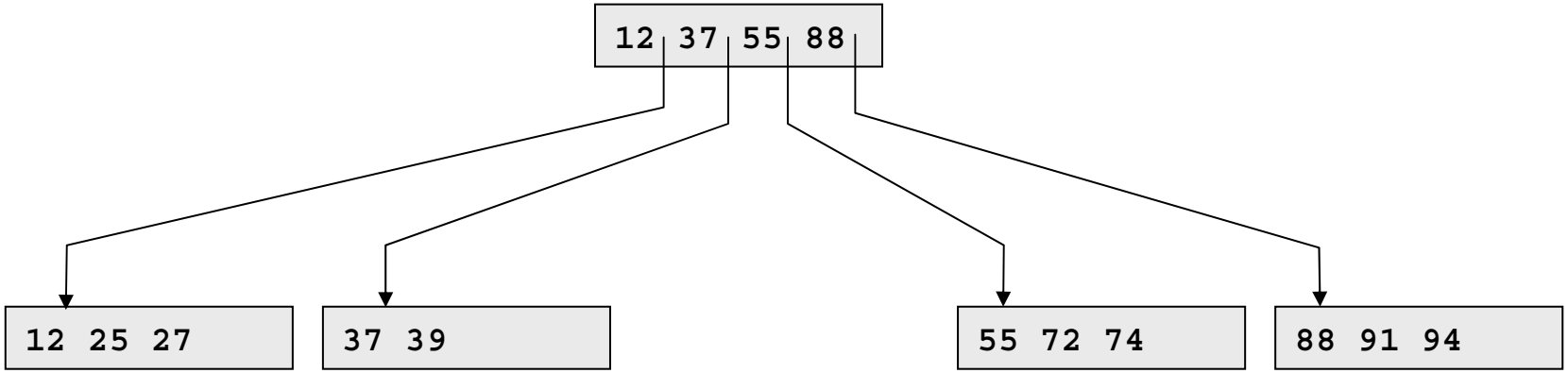
27 55 12 94 37 88 72 39 25 91



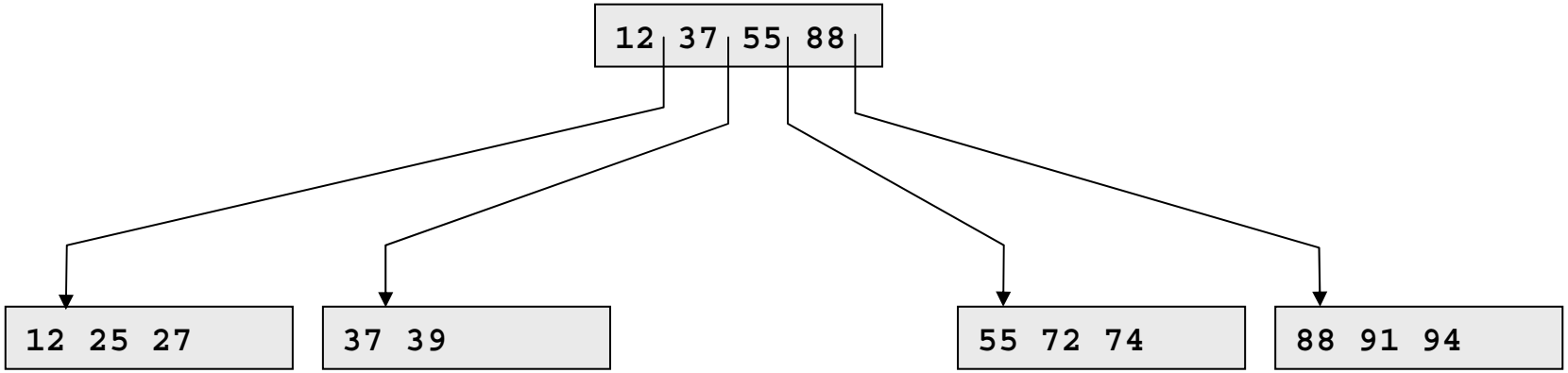
27 55 12 94 37 88 72 39 25 91 74



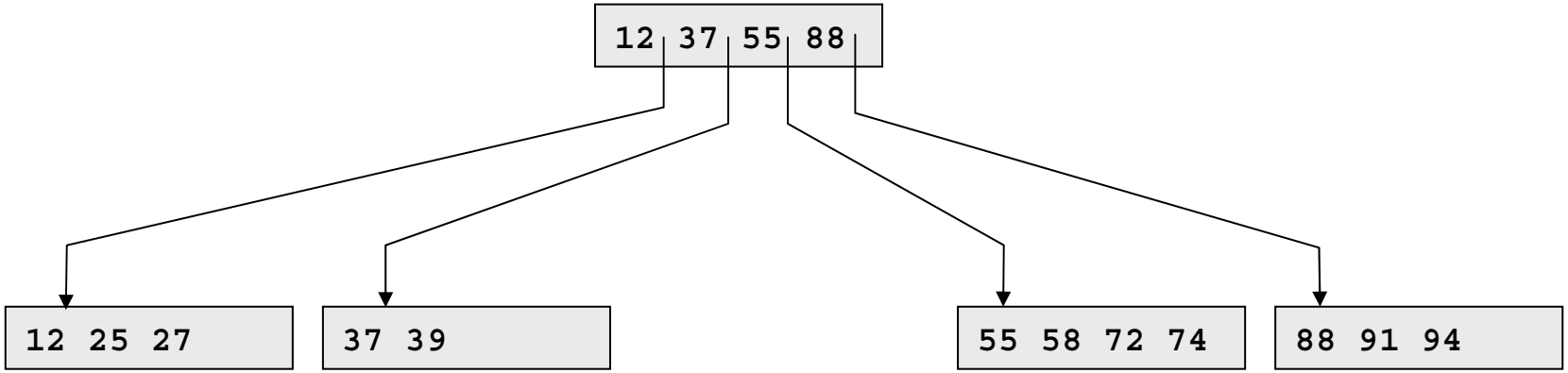
27 55 12 94 37 88 72 39 25 91 74



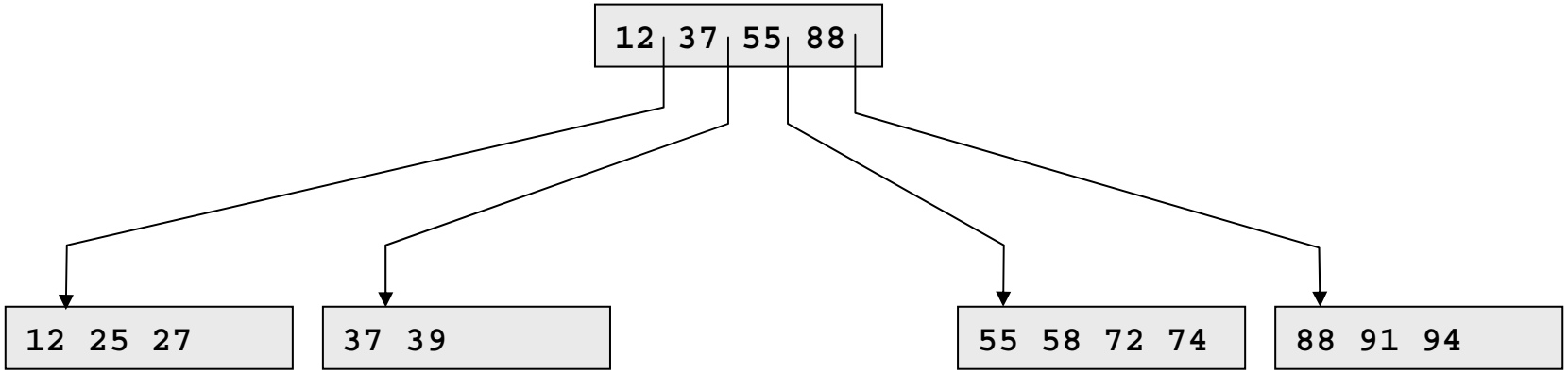
27 55 12 94 37 88 72 39 25 88 74 58



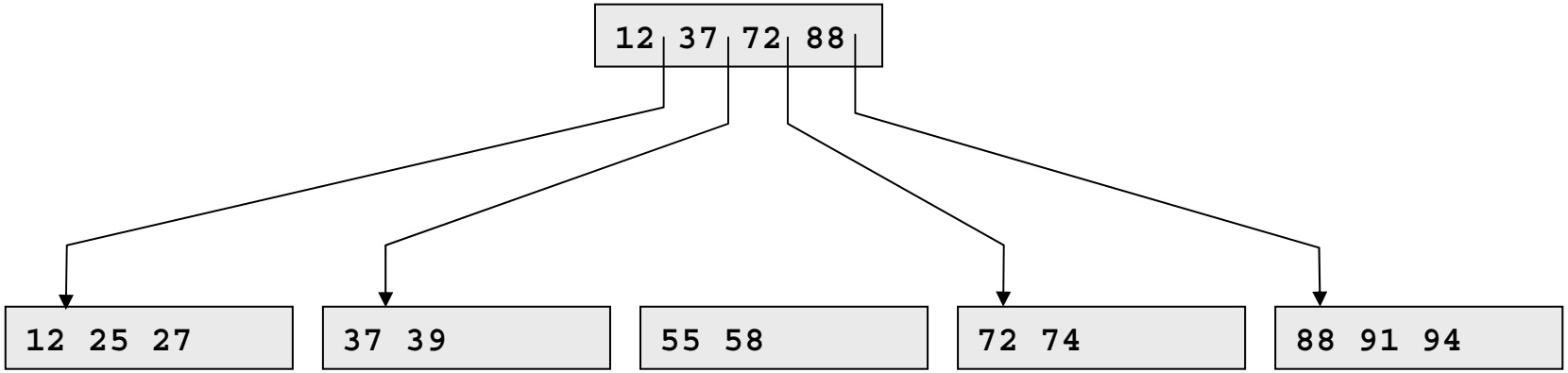
27 55 12 94 37 88 72 39 25 88 74 58



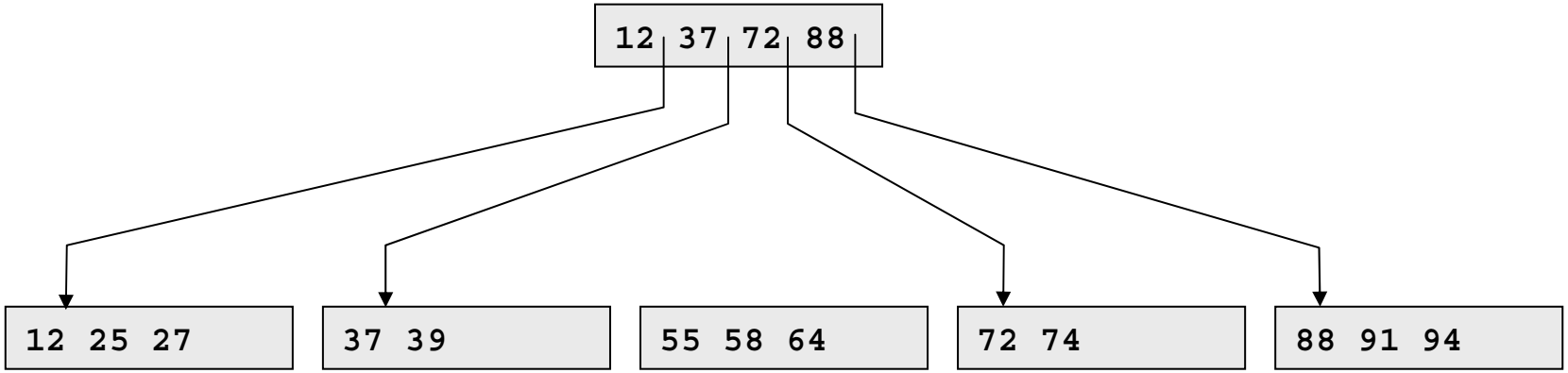
27 55 12 94 37 88 72 39 25 88 74 58 64



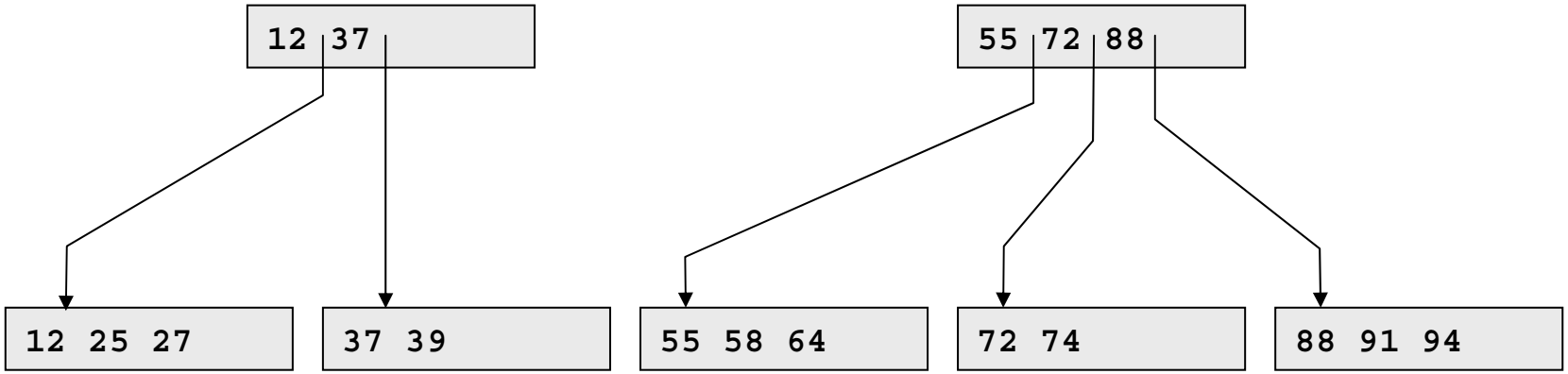
27 55 12 94 37 88 72 39 25 88 74 58 64



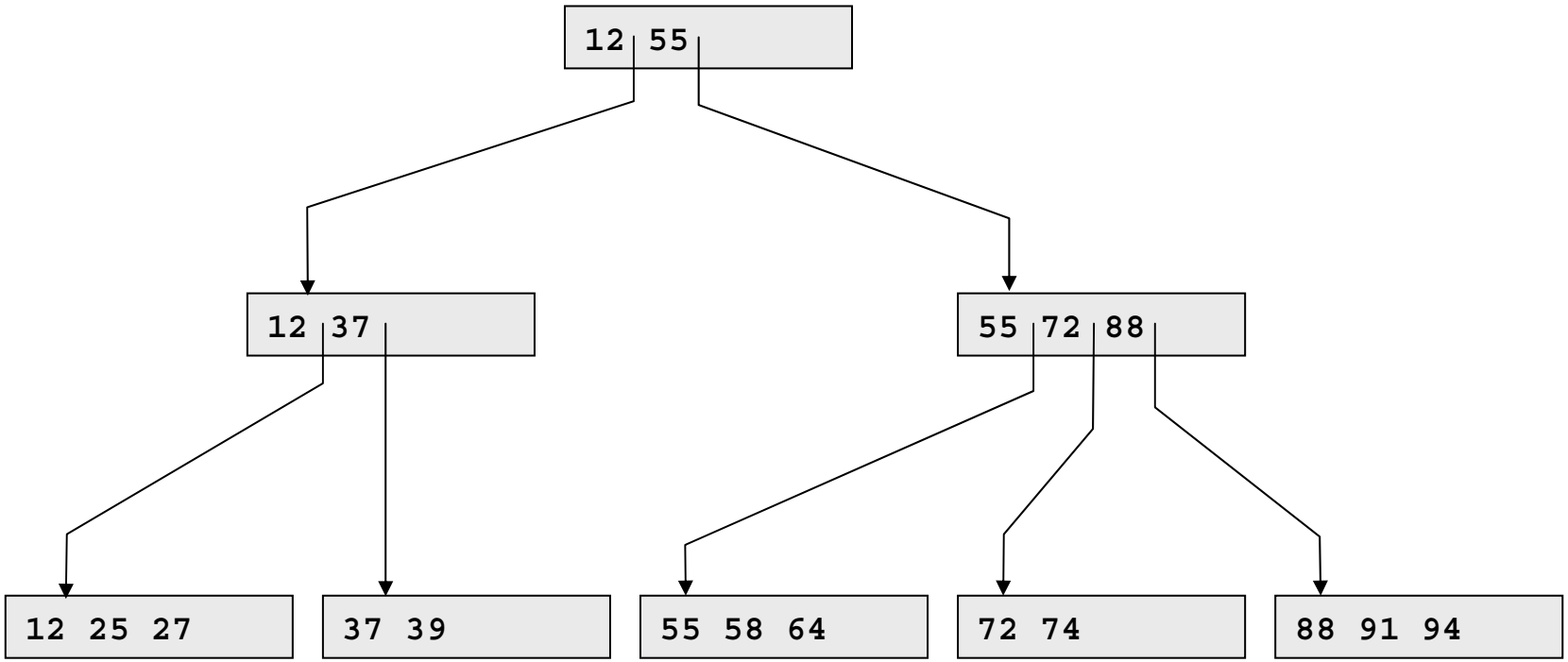
27 55 12 94 37 88 72 39 25 88 74 58 64



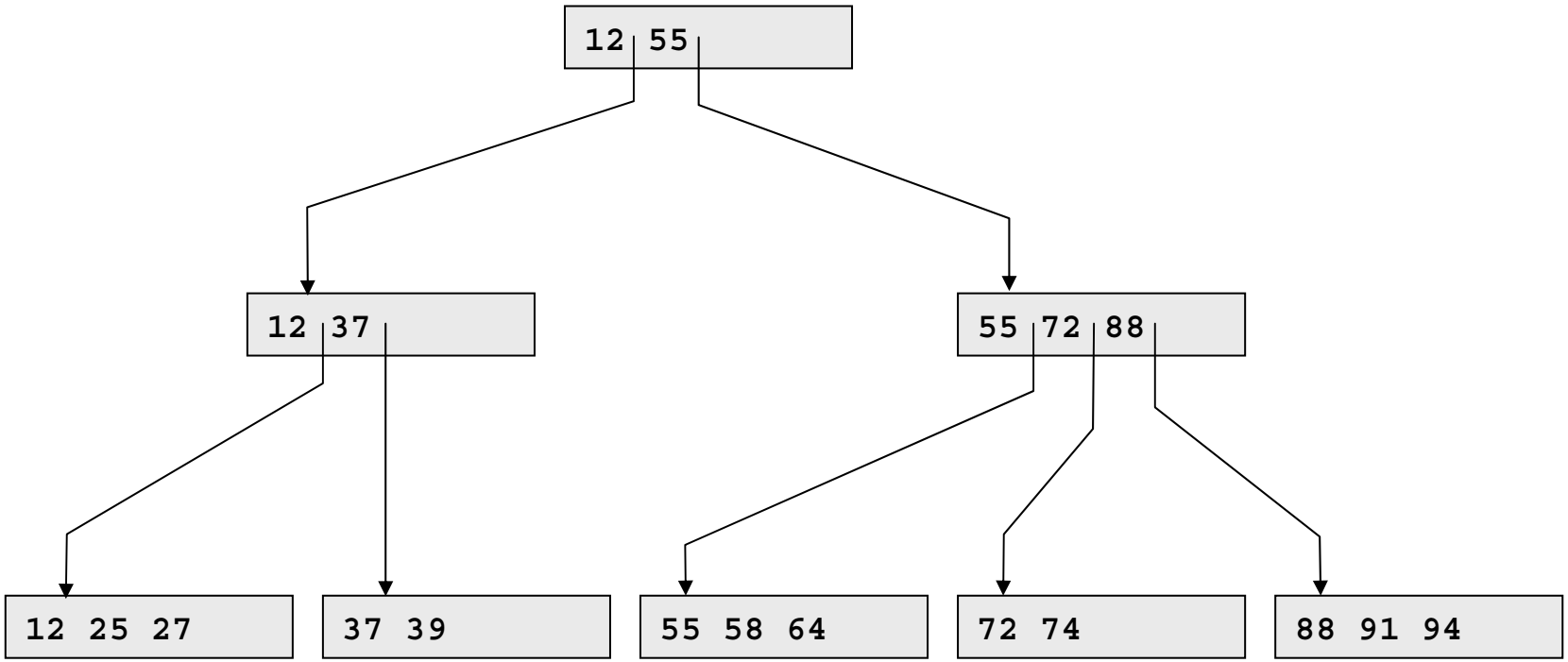
27 55 12 94 37 88 72 39 25 88 74 58 64



27 55 12 94 37 88 72 39 25 88 74 58 64

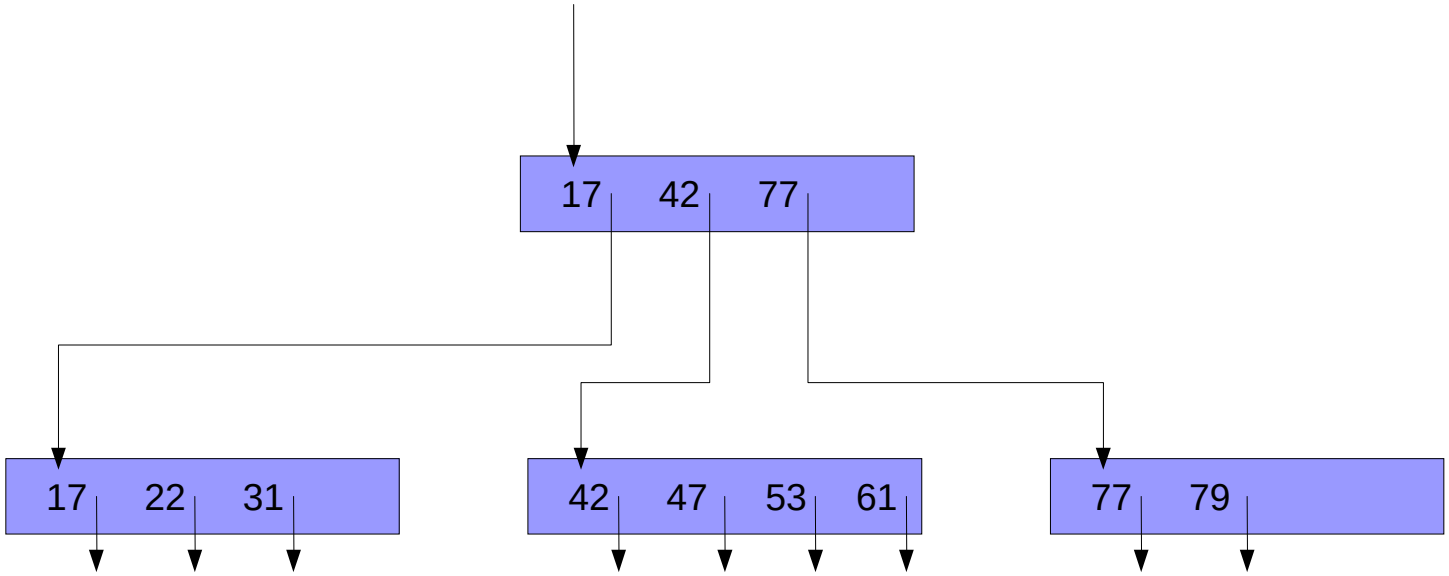


27 55 12 94 37 88 72 39 25 88 74 58 64



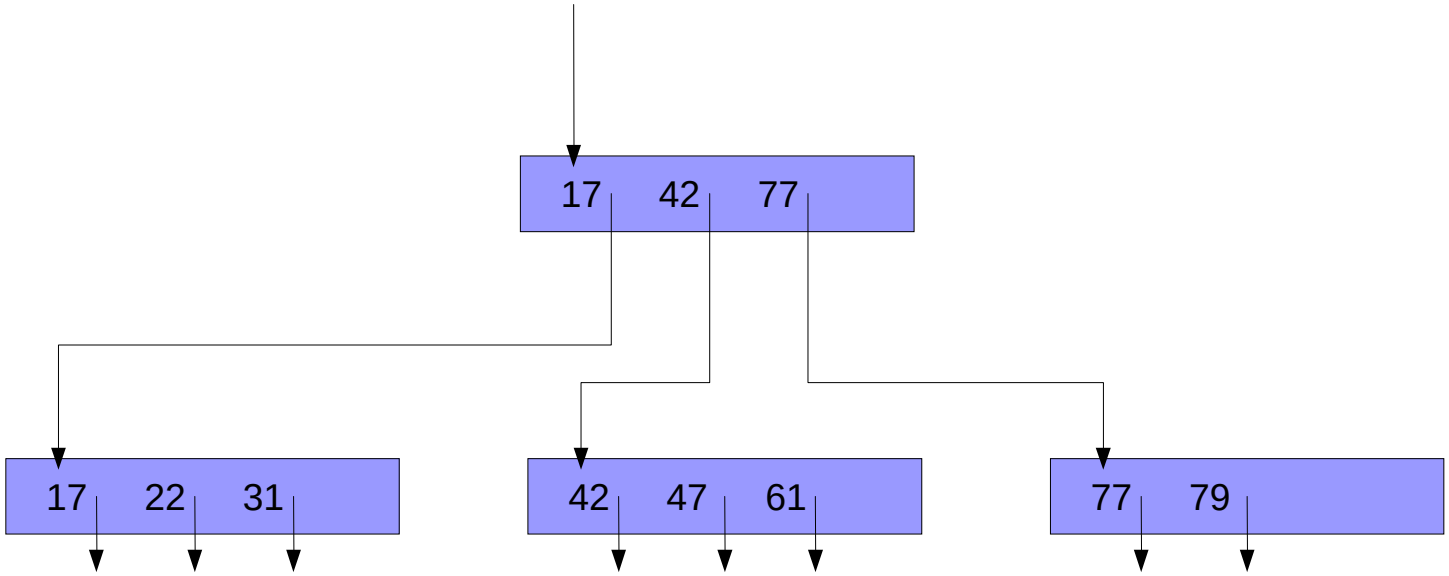
Sequenz eingefügt

B*Baum: Delete



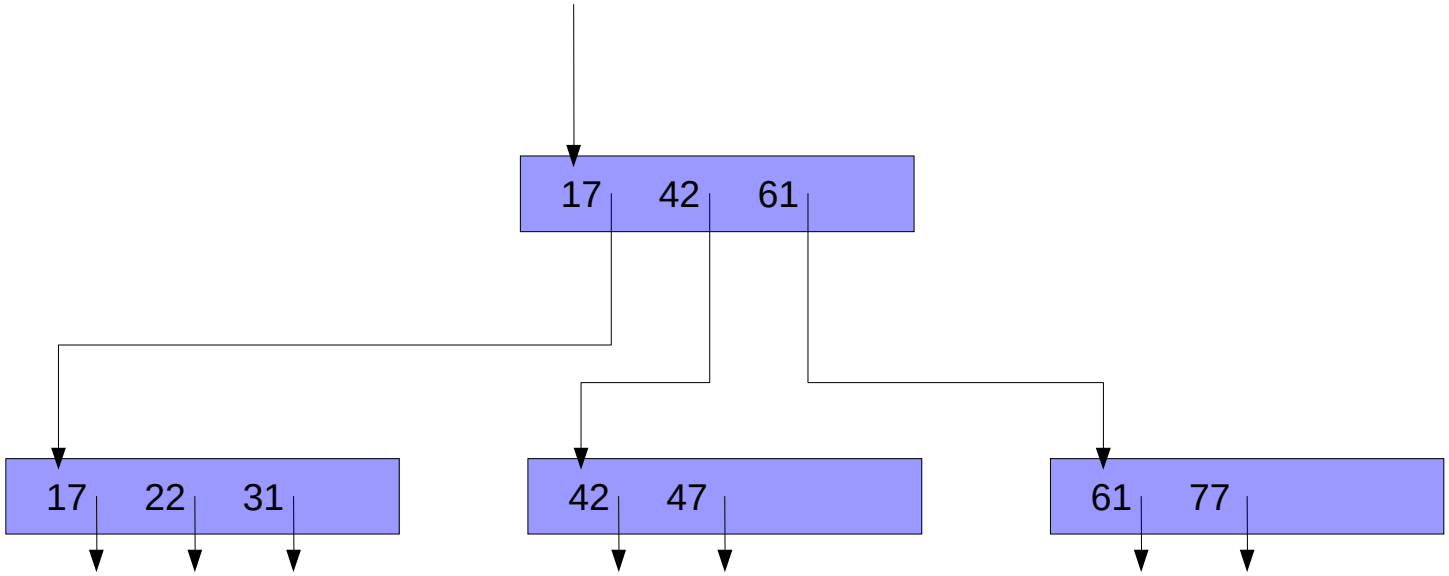
Entferne 53

B*Baum: Delete



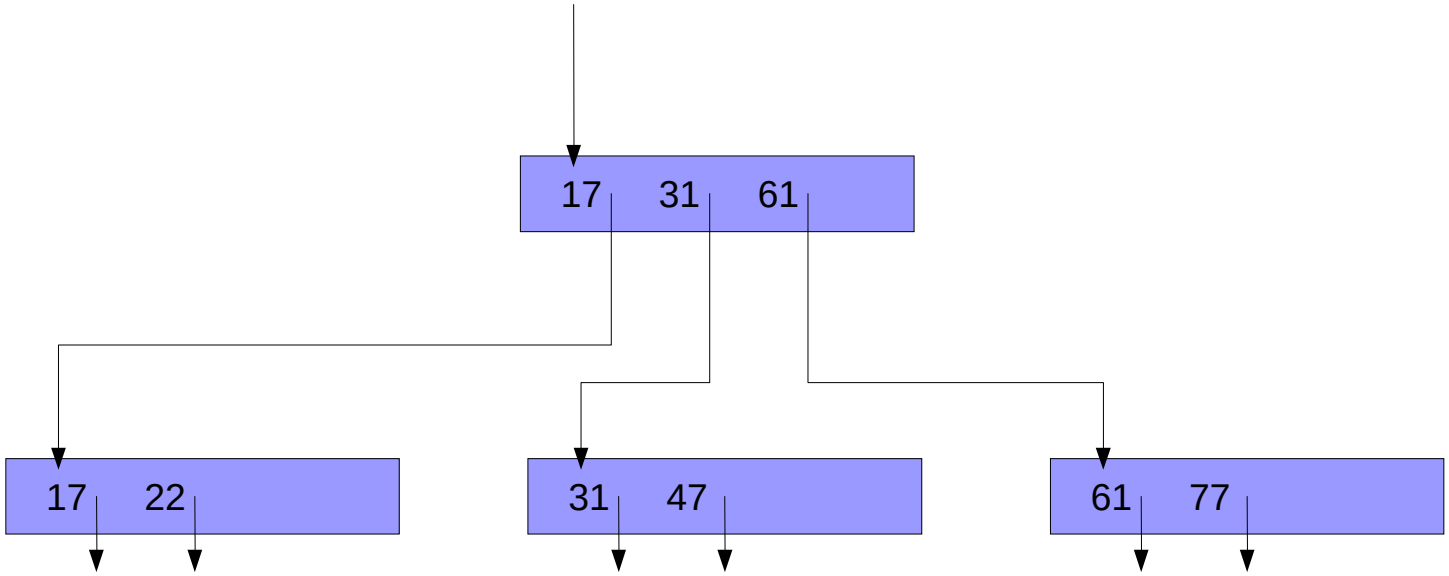
Entferne 79

B*Baum: Delete



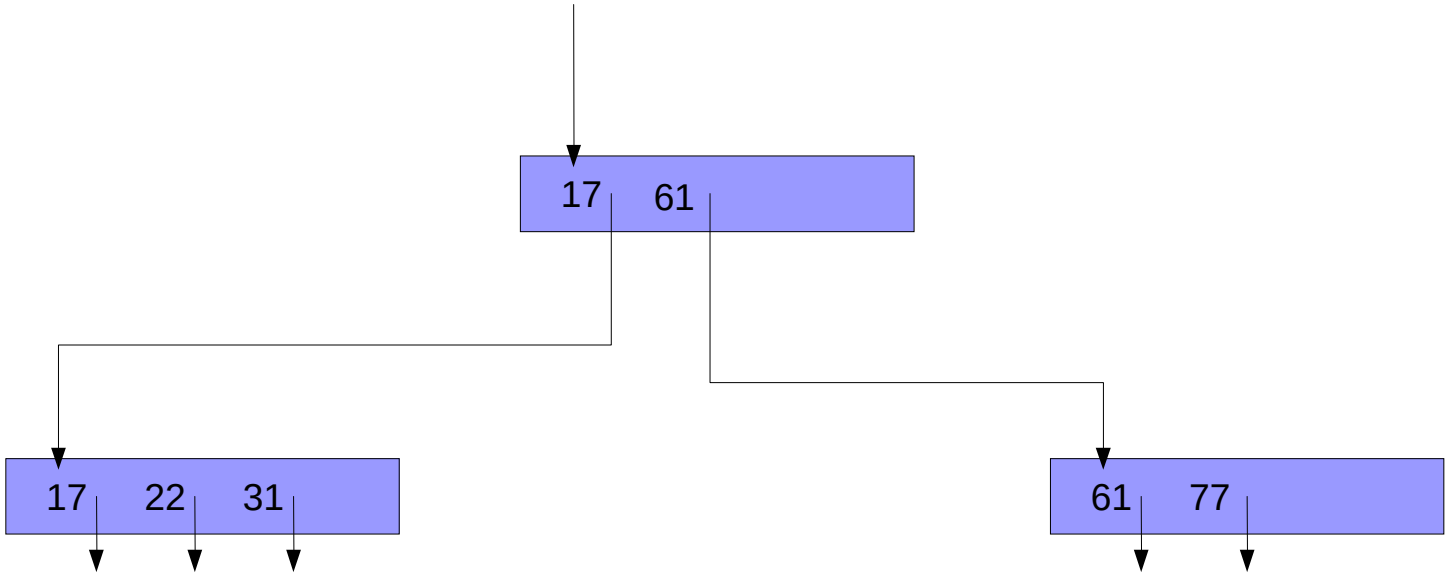
Entferne 42

B*Baum: Delete



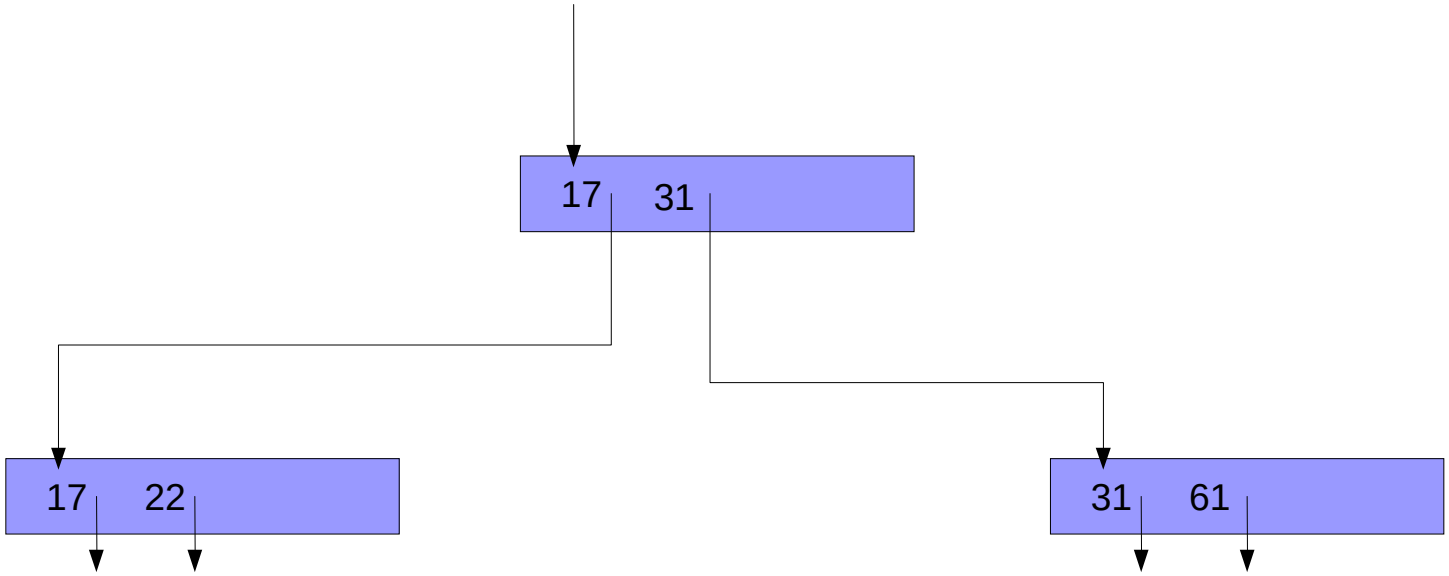
Entferne 47

B*Baum: Delete



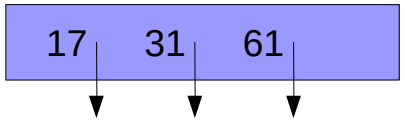
Entferne 77

B*Baum: Delete



Entferne 22

B*Baum: Delete



Fragen zum B*Baum

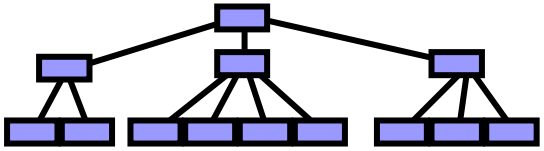
Wie groß ist k ?
Blockgröße / (Schlüssel / Adresspaar-Größe) =
1024 / (15+4) / 2 = 26

Wieviel Söhne hat eine zu 50 % gefüllte Wurzel ?
26

Wieviel Söhne hat ein zu 75 % gefüllter Knoten ?
39

Wieviel zu 75 % gefüllte Datenblöcke sind erforderlich ?
300.000 / 7,5 ≈ 40.000

Platzbedarf B*Baum



Wieviel Blöcke belegt der B*Baum ?

Höhe	Knoten	Zeiger aus Knoten
0	1	26
1	26	26 * 39 = 1.014
2	26*39	26*39*39 = 39.546

$\Rightarrow$ drei Ebenen reichen aus

$\Rightarrow$ Platzbedarf = 1 + 26 + 1.014 + 39.546 $\approx$ 40.000 Blöcke

Wieviel Blockzugriffe sind erforderlich ?

4

Hashing versus B*Baum

Welcher Platzbedarf entsteht beim Hashing, wenn dieselbe Zugriffszeit erreicht werden soll wie beim B*Baum?

4 Blockzugriffe = 1 Directory + 3 Datenblöcke

⇒ Buckets bestehen im Mittel aus 5 Blöcken.

⇒ von 5 Blöcken sind 4 voll und der letzte halb voll.

⇒ 4,5 * 10 = 45 Records pro Bucket

⇒ 300.000 / 45 = 6666 Buckets erforderlich

⇒ 6666 / (1024 / 4) = 26 Directory-Blöcke

⇒ Platzbedarf = 26 + 5 * 6.666 = 33.356

B*Baum versus Hashing

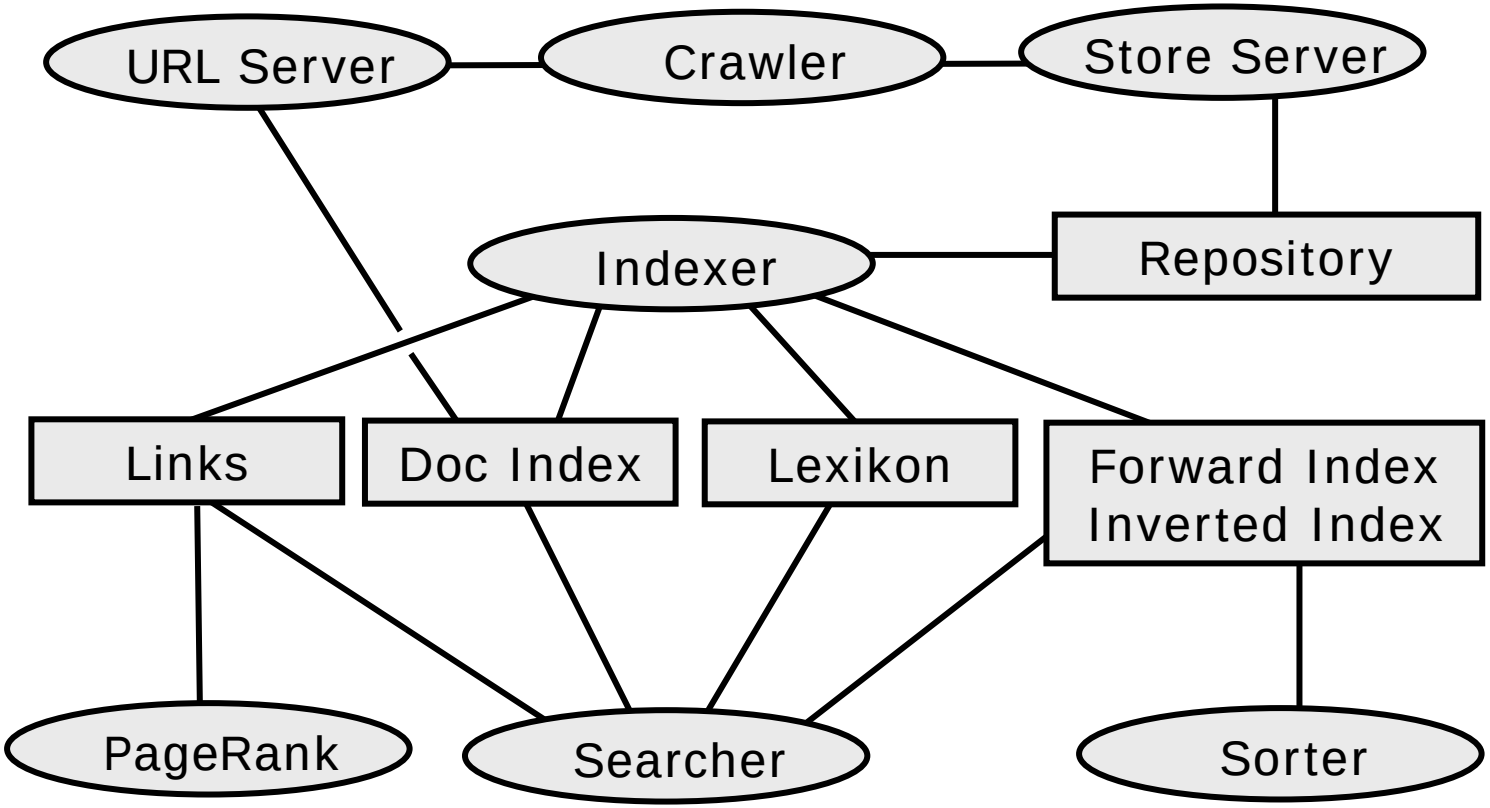
	B*Baum	Hashing
Vorteile	Dynamisch Sortierung möglich	Schnell platzsparend
Nachteile	Speicheroverhead kompliziert	keine Sortierung Neuorganisation

Google

- 1998 gegründet von Sergey Brin + Larry Page
- 25.000 Mitarbeiter
- 100.000.000.000 $ Marktwert
- Oberfläche in 100 Sprachen
- 1.000.000 Server
- 1.000.000.000.000 Webseiten im Cache
- 1.000.000.000.000.000 Bytes Plattenplatz
- 3.000.000.000 Queries pro Tag
- 30.000 Queries pro Sekunde
- 14.000.000 Wörter im Lexikon

Google Classic

Systemarchitektur



Repository

- komplettes HTML
- komprimiert mit zlib (1:3)

docID, docLength, docURL, docContent

Lexikon

- 14 Millionen Einträge
- jedes Wort gehasht auf **wordID**

wordID	**#docs**	**pointer**

zeigt auf erste Seite
im Inverted Index
mit **docIDs**,
relevant für **wordID**

HIT

- jeweils kodiert in 2 Bytes:

  Bit 00: capitalization

  Bit 01-03: font size

  Bit 04-15: position

- Plain Hit: innerhalb von HTML

- Fancy Hit: innerhalb von
  - URL
  - title
  - anchor text
  - meta tag

Forward Index

erzeugt vom Indexer aus Repository

```
docID wordID  #hits  hit hit hit hit hit
      wordID  #hits  hit hit hit
      wordID  #hits  hit hit hit hit

docID wordID  #hits  hit hit hit
      wordID  #hits  hit hit
      wordID  #hits  hit hit hit hit
      wordID  #hits  hit hit hit
```

Inverted Index

erzeugt vom Sorter aus Forward Index

```
wordID docID  #hits  hit hit hit hit hit
       docID  #hits  hit hit hit
       docID  #hits  hit hit hit hit
       docID  #hits  hit hit hit hit


wordID docID  #hits  hit hit hit hit
       docID  #hits  hit hit hit hit hit
       docID  #hits  hit hit hit hit
```

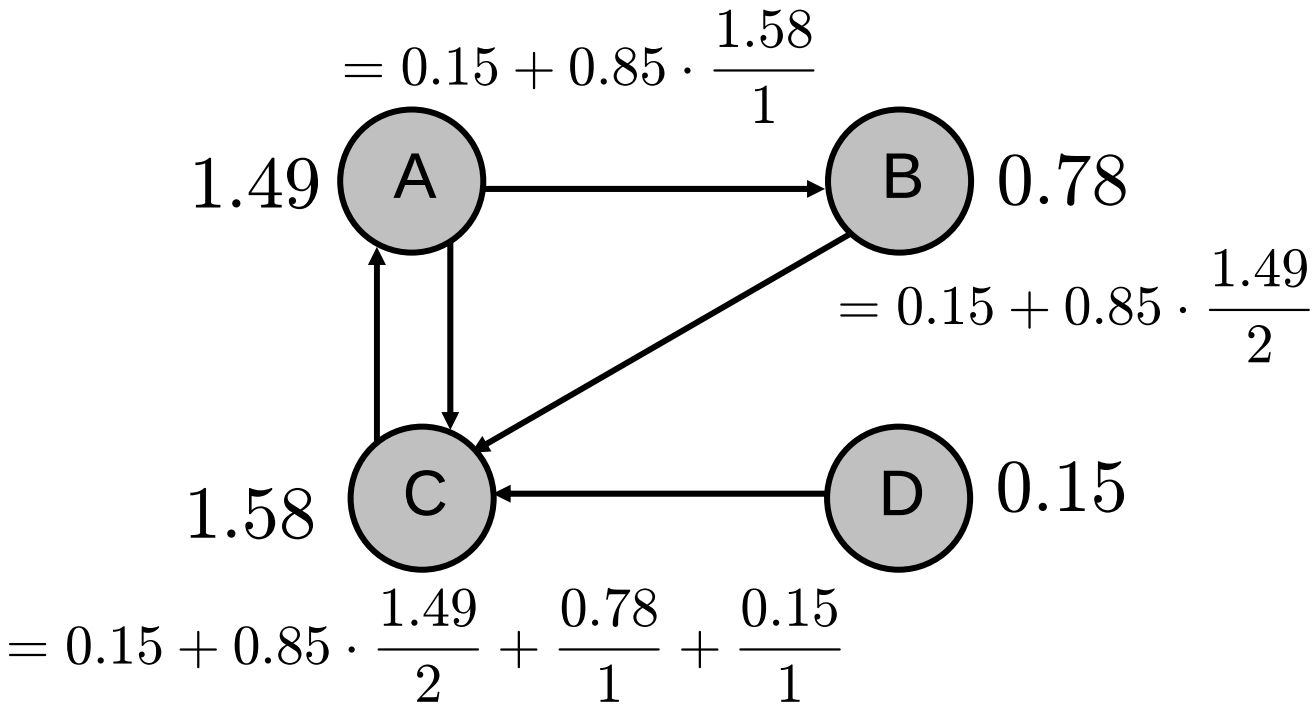
PageRank: Definition

Seite *T* habe *C(T)* ausgehende Links
Seiten T_1, T_2, ..., T_n zeigen auf Seite *A*
Gegeben sei Dämpfungsfaktor $0 \leq d \leq 1$

$$PR(A) := (1-d) + d \cdot \sum_{i=1}^{n} \frac{PR(T_i)}{C(T_i)}$$

$\Rightarrow$ Gleichungssystem, iterativ lösbar

PageRank: Beispiel



Dämpfungsfaktor d = *0.85*

Single Word Query

Betrachte Hit-Liste für Dokument d bzgl. Word w

Hit-Type	[title, anchor, URL, plain large, plain small, ...]
Hit-Weight	[, , , , , ...]
Hit-Count	[, , , , , ...]

Weight-Score (d,w) := Hit-Weight * Hit-Count

PageRank(d) := gemäß Gleichungssystem

Final-Score (d,w) := Weight-Score (d,w) $\oplus$ PageRank (d)

Multi Word Query

- 10 Proximity Klassen
  (von "benachbart" bis "weit entfernt")

- Proximity-Score belohnt nah beieinanderliegende Suchwörter

- Weight-Score belohnt Treffertyp und Häufigkeit

- Final-Score $(d, w_1, w_2, \ldots, w_n)$:=

  Weight-Score $(d, w_1, w_2, \ldots, w_n) \oplus$
  Proximity-Score $(d, w_1, w_2, \ldots, w_n) \oplus$
  PageRank (d)

http://www.google.de