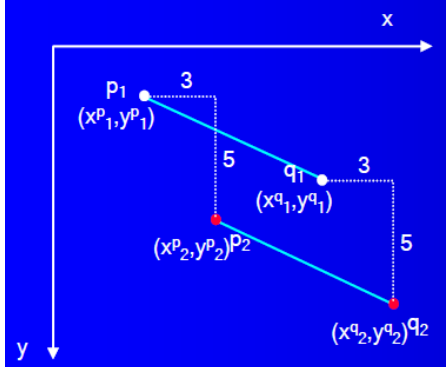
Idee: Mischen der einzelnen Farbanteile

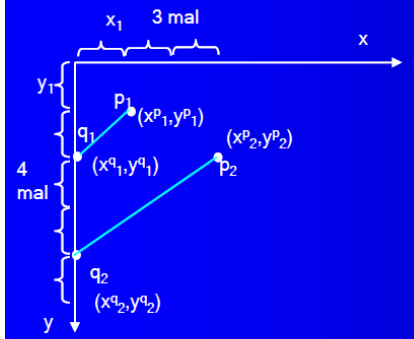
```
int singleShuffle(int i1_part, int i2_part, int p) {
    return i1_part + (i2_part - i1_part) * p / 100;
}

int colorShuffle(int i1, int i2, int p) {
  int red   = singleShuffle((i1 >> 16) & 255,(i2 >> 16) & 255,p);
  int green = singleShuffle((i1 >> 8)  & 255,(i2 >> 8)  & 255,p);
  int blue  = singleShuffle((i1)       & 255,(i2)       & 255,p);
  return (255 << 24) | (red << 16) | (green << 8) | blue;
}
```



- Die Punkte p_1 und q_1 werden bei dieser Translation auf die Punkte p_2 und q_2 abgebildet
- Die zugehörigen Linien haben ihre ***Form nicht geändert***
- Sie haben lediglich ihre ***Lage im Raum*** (2-Dim.) ***geändert***

Skalierung (Fort.)

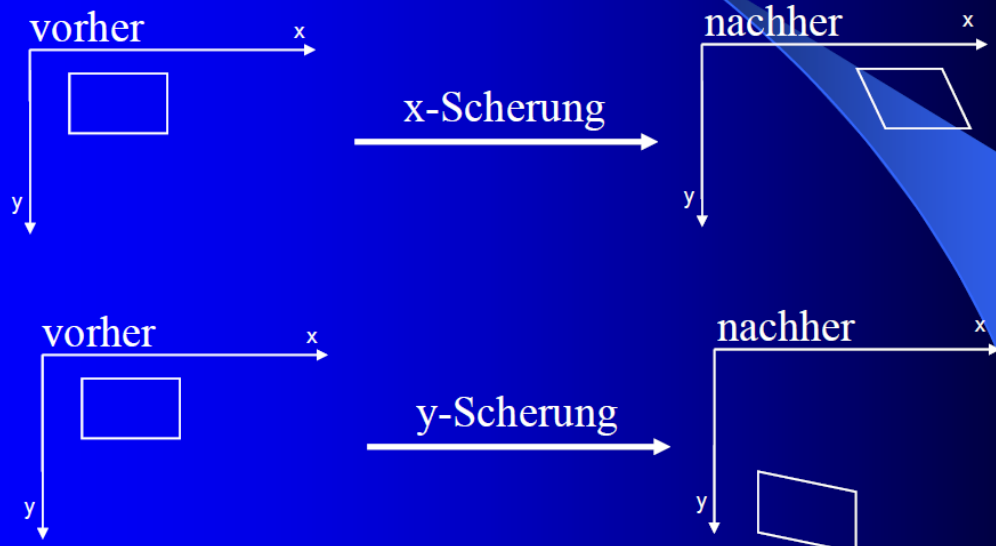
- anders als bei der Translation verändern sich die Punkte unterschiedlich in Abhängigkeit von ihrem Abstand zum Ursprung
- somit kann eine Vergrößerung bzw. Verkleinerung des Bildes erzielt werden



- Skalierung der Punktes p_1 und q_1 um die Werte 3 und 2
- das Ergebnis ist wieder eine Linie, die aber länger ist als die ursprüngliche Linie und eine andere Steigung hat
- durch Skalierungswerte < 1 kann eine Verkleinerung durchgeführt werden

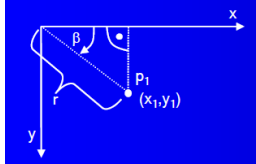
Scherung (Fort.)

- eine Scherung bewirkt eine Verzerrung in x- bzw. y-Richtung
- Beispiel:



Rotation (Fort.)

- für die Berechnung der Rotation muss man sich die Koordinaten des Punkts als eine Gleichung aus dem
  - Abstand zum Koordinatenursprung und
  - dem Winkel zwischen der Verbindung des Punktes und dem Koordinatenursprung und der y-Koordinate vorstellen



- es gilt:
  - $\sin(\beta) = y_1 / r$
  - $\cos(\beta) = x_1 / r$
- $\Rightarrow$
  - $x_1 = r \times \cos(\beta)$
  - $y_1 = r \times \sin(\beta)$

Rotation (Fort.)

mit der Definition von (x_1,y_1)

$x_1 = r \times \cos(\beta)$

$y_1 = r \times \sin(\beta)$

kann

$$x_2 = \overbrace{r\times\cos(\beta)}^{x_1}\times\cos(\alpha)-\overbrace{r\times\sin(\beta)}^{y_1}\times\sin(\alpha)$$

$$y_2 = \underbrace{r\times\sin(\beta)}_{y_1}\times\cos(\alpha)+\underbrace{r\times\cos(\beta)}_{x_1}\times\sin(\alpha)$$

wie folgt vereinfacht werden:

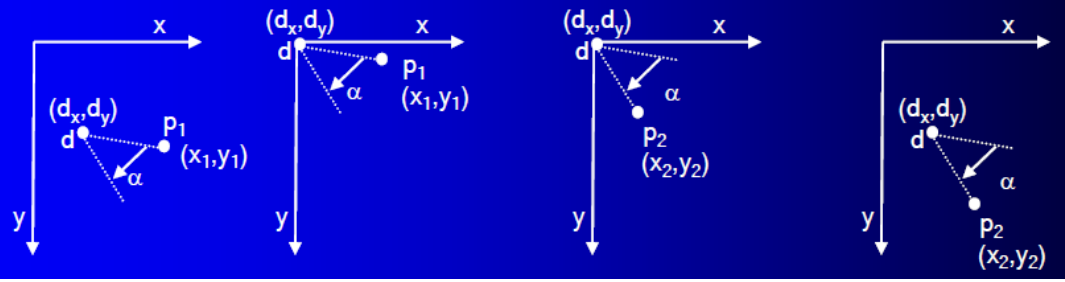
$$x_2 = x_1\times\cos(\alpha)-y_1\times\sin(\alpha)$$
$$y_2 = y_1\times\cos(\alpha)+x_1\times\sin(\alpha)$$

Gleichung für die Rotation des Punktes (x_1,y_1) um den Winkel α

Rotation (Fort.)

- die Rotation wird immer um den Koordinatenursprung durchgeführt
- Problem: was muss gemacht werden, wenn der Punkt p_1 mit (x_1,y_2) nicht um $(0,0)$ sondern um (d_x,d_y) gedreht werden soll?
- Antwort:
  1. verschiebe Punkt p_1 um $(-d_x,-d_y)$ (Translation)
  2. rotiere Punkt um den Ursprung
  3. verschiebe neuen Punkt um (d_x,d_y) (Translation)

Ausgangslage: 1. 2. 3.



VL 1 - VL 3

Matrizenmultiplikation

Zwei Matrizen können multipliziert werden, wenn die Spaltenanzahl der linken mit der Zeilenanzahl der rechten Matrix übereinstimmt.

Rechenbeispiel:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \cdot \begin{pmatrix} 6 & -1 \\ 3 & 2 \\ 0 & -3 \end{pmatrix} = \begin{pmatrix} 1\cdot 6+2\cdot 3+3\cdot 0 & 1\cdot(-1)+2\cdot 2+3\cdot(-3) \\ 4\cdot 6+5\cdot 3+6\cdot 0 & 4\cdot(-1)+5\cdot 2+6\cdot(-3) \end{pmatrix} = \begin{pmatrix} 12 & -6 \\ 39 & -12 \end{pmatrix}$$

- für die x-Scherung gilt:
  - $x_2 = x_1 + y_1 * ShX$
  - $y_2 = y_1$
- wird der Punkt (x_1,y_2) als Spaltenvektor interpretiert, so kann die x-Scherung als folgende 2×2 Matrix verstanden werden

$$\begin{vmatrix} 1 & ShX \\ 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} = \begin{vmatrix} x_1+ShX \times y_1 \\ y_1 \end{vmatrix}$$

- für die y-Scherung gibt entsprechend:

$$\begin{vmatrix} 1 & 0 \\ ShY & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} = \begin{vmatrix} x_1 \\ ShY \times x_1+y_1 \end{vmatrix}$$

Matrizen: Beispiel (Fort.)

- soll nun erst eine x-Scherung und dann eine y-Scherung durchgeführt werden, gilt folgendes:

$$\begin{vmatrix} 1 & 0 \\ ShY & 1 \end{vmatrix} \times \begin{vmatrix} 1 & ShX \\ 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix}$$

y-Scherung x-Scherung

WICHTIG: man beachte die Leseweise von rechts nach links

- da für Matrizen das Assoziativgesetz gilt, können auch erst die beiden Matrizen multipliziert werden:

$$\begin{vmatrix} 1 & 0 \\ ShY & 1 \end{vmatrix} \times \begin{vmatrix} 1 & ShX \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & ShX \\ ShY & ShY*ShX+1 \end{vmatrix}$$

- dies hat den Vorteil, dass mehrere Punkte immer ***nur mit einer statt*** mit ***zwei*** Matrizen multipliziert werden müssen

Transformations-Matrizen

- Translation:

$$\begin{vmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1+d_x \\ y_1+d_y \\ 1 \end{vmatrix}$$

erweiteter Koordinatenvektor

- Rotation:

$$\begin{vmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1\times\cos(\alpha)-y_1\times\sin(\alpha) \\ x_1\times\sin(\alpha)+y_1\times\cos(\alpha) \\ 1 \end{vmatrix}$$

Transformations-Matrizen

- Translation:

$$\begin{vmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1+d_x \\ y_1+d_y \\ 1 \end{vmatrix}$$

erweiteter Koordinatenvektor

- Rotation:

$$\begin{vmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1\times\cos(\alpha)-y_1\times\sin(\alpha) \\ x_1\times\sin(\alpha)+y_1\times\cos(\alpha) \\ 1 \end{vmatrix}$$

Transformations-Matrizen (Fort.)

- Skalierung:

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1\times S_x \\ y_1\times S_y \\ 1 \end{vmatrix}$$

- x-Scherung:

$$\begin{vmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1+y_1\times Sh_x \\ y_1 \\ 1 \end{vmatrix}$$

- y-Scherung:

$$\begin{vmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1 \\ x_1\times Sh_y+y_1 \\ 1 \end{vmatrix}$$

VL 1 - VL 3

Letztes Problem: Lösung

- Lösung: nicht von der Ursprungskoordinate losrechnet, sondern von Zielkoordinate fragen, welche Ursprungskoordinate auf diese abgebildet wird
- mathematisch ist das leicht geschrieben: statt

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = \begin{vmatrix} x_1 \times S_x \\ y_1 \times S_y \\ 1 \end{vmatrix}$$

- rechnen wir:

die Zielkoordinate wird durch die Transformationsmatrix geteilt

$$\begin{vmatrix} x_u \\ y_u \\ 1 \end{vmatrix} = \frac{\begin{vmatrix} x_z \\ y_z \\ 1 \end{vmatrix}}{\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}}$$

????? durch Matrix teilen ????

Letztes Problem: Lösung (Fort.)

- teilen bedeutet: mit dem multiplikativen Inversen multiplizieren, d.h. zu einer Matrix m wird die Matrix m^{-1} gesucht, so dass gilt:

$$m \times m^{-1} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- im Normalfall existieren diese multiplikativen Inversen von Matrizen nicht, jedoch in diesem Fall sind sie sehr einfach:
- Translation: nicht H und V sondern -H und -V
- Rotation: nicht α sondern $-\alpha$
- Skalierung: nicht S_x und S_y sondern $1/S_x$ und $1/S_y$
- usw.

- mit den inversen Matrizen kann jetzt eine Applikation erstellt werden
- soll ein Startbild S durch eine Transformationsmatrix m in ein Zielbild Z transformiert werden, wird folgendes gemacht:
  1. für alle Bildpunkte p_z in Z berechne $m^{-1} \times p_z$
  2. das Ergebnis beschreibt eine Koordinate p_s in S
  3. übertrage den Bildwert von p_s aus S in Z an den Punkt p_z

VL 8 – Approximation

Approximation: 1-dimensional (Beispiel)

M = {1, 4, 17, 23, -34, -2003, 1024, 6, 7}

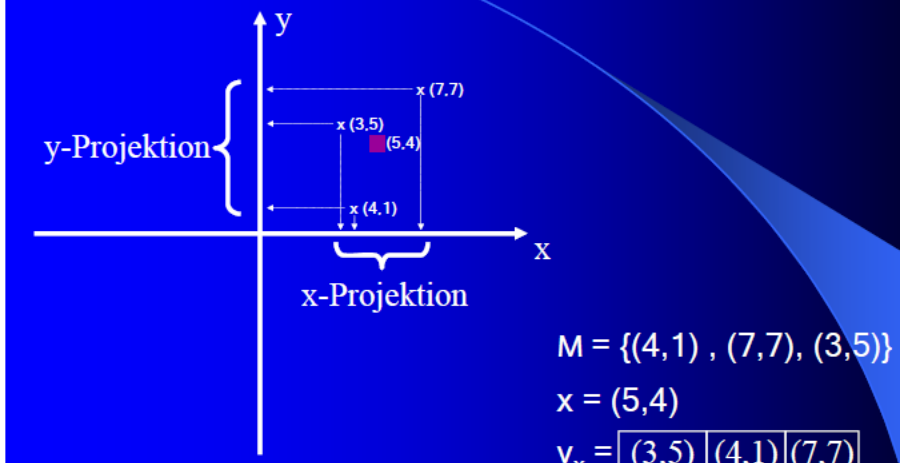
x = 9

v =

-2003	-34	1	4	6	7	17	23	1024
0	1	2	3	4	5	6	7	8

- Ergebnis der binären Suche: i = 5
- da v[5] = 7 < x = 9 ist, wird x zusätzlich mit v[6] verglichen
- Ergebnis des Vergleichs: v[5] = 7 hat den kleinsten Abstand zu 9 von allen Zahlen aus M

Approximation: 2-dimensional: Beispiel



M = {(4,1) , (7,7), (3,5)}

x = (5,4)

v_x =

(3,5)	(4,1)	(7,7)
0	1	2

v_y =

(4,1)	(3,5)	(7,7)
0	1	2

- Suchen von (5,4)

Binärsuche bzgl. 5 (x-Wert) endet hier

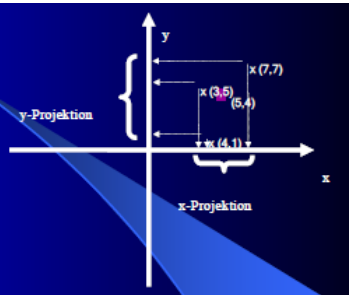
v_x =

(3,5)	(4,1)	(7,7)
0	1	2

v_y =

(4,1)	(3,5)	(7,7)
0	1	2

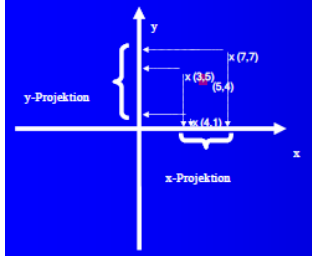
Binärsuche bzgl. 4 (y-Wert) endet hier



es sind 4 Vergleiche notwendig:

- 2 bzgl. der x-Projektion (5,4) mit (4,1) und (7,7)
- 2 bzgl. der y-Projektion (5,4) mit (4,1) und (3,5)

Approximation: 2-dimensional: Beispiel (Forts.)



bei den 4 Vergleichen werden die Abstände der Punkte zueinander berechnet (Satz des Pythagoras):

- $|(5,4) - (4,1)| = \sqrt{(1+9)} \approx 3{,}16$
- $|(5,4) - (7,7)| = \sqrt{(4+9)} \approx 3{,}60$
- $|(5,4) - (3,5)| = \sqrt{(4+1)} \approx 2{,}23$

die erste Vergleichsrunde hat ergeben, dass maximal in einem Abstand von 2,23 gesucht werden muss

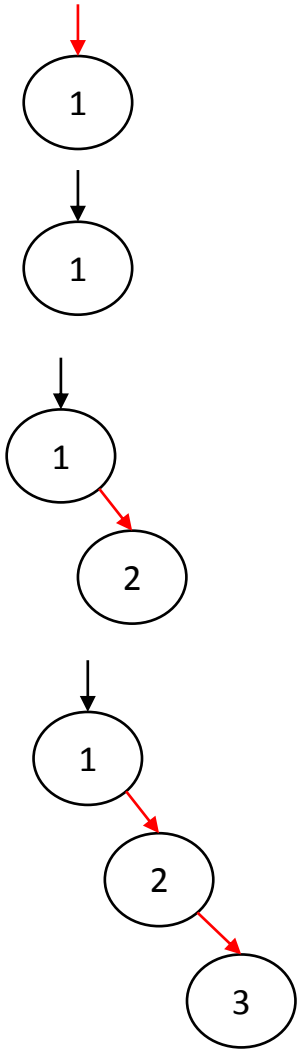
⇒ es müssen maximal bzgl. der **x-Projektion** die Werte zwischen 5-2,23 = 2,77 und 5+2,23 = 7,23 betrachtet werden

⇒ es müssen maximal bzgl. der **y-Projektion** die Werte zwischen 4-2,23 = 1,77 und 4+2,23 = 6,23 betrachtet werden

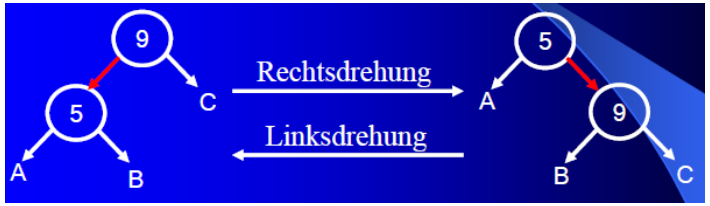
VL 9 - 11 – Top 2,3,4 und Rot Schwarz Baum

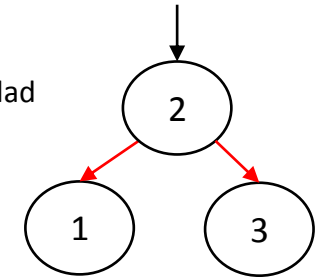
Regeln beim Einfügen – Bsp. 1,2,3,4,5,6,7,8

1. Einfügen der 1
2. M_Root == null
    1. Neuen Knoten einfügen (hat immer Rote Kante)
    2. SPLIT (wenn es einen Vater gibt und er Rot ist)
        1. False
    3. M_Root am ende immer auf Schwarz setzten
3. Einfügen der 2
4. M_Root nicht null
5. Solange Knoten nicht null
    1. 4er Knoten?
        1. False
    2. Vergleiche Knoten und steige ab
6. Knoten 2 rechts einfügen mit roter Kante
    1. SPLIT → false
7. Einfügen der 3
8. Keine 4er Knoten
9. Einfügen Rechts bei Knoten 2
    1. SPLIT = true → gibt vater mit roter Kante (2)
        1. Wenn die roten Kanten gleiche Ausrichtung haben rotiere den Großvater
        2. Wenn die roten Kanten nicht die gleiche Ausrichtung haben, rotiere Vater dann Großvater
        3. Kanten haben gleiche Ausrichtung
            1. Rotate(GROßVATER)
                1. Dad = 1 = schwarz
                2. Son = 2 = rot
                3. Soncolor = son
                4. Son kriegt farbe von vater
                5. Vater kriegt farbe von son
                6. Dad = 1 = rot
                7. Son = 2 = schwarz

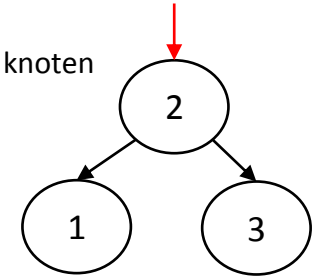


Wenn dad links = son dann dad links = son rechts und son rechts = dad
Ansonsten dad rechts = son links und Son links = dad

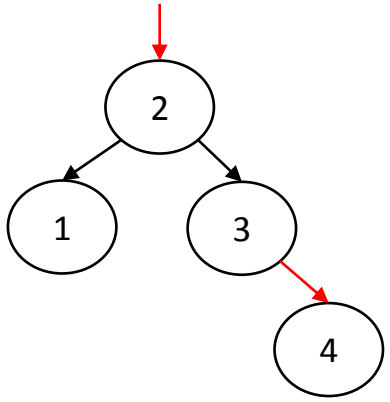




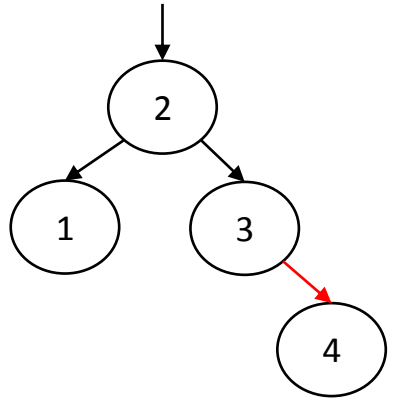
Einfügen der 4
Knoten 2 ist ein 4er knoten
Konvertieren



SPLIT false
Knoten rechts bei Knoten (3) einfügen



Split false
M_Root = schwarz



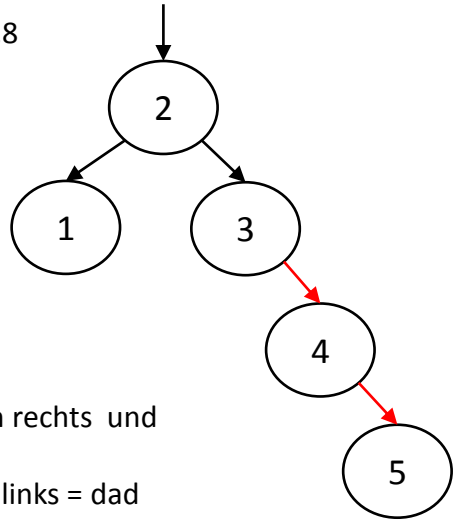
VL – Rot Schwarz Baum

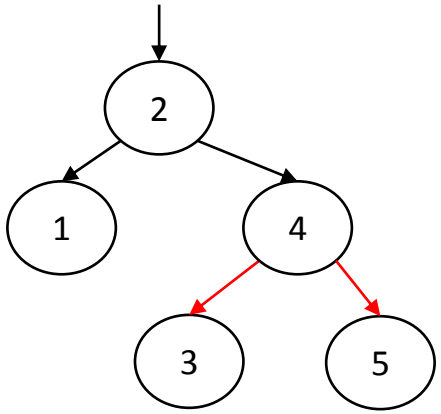
Regeln beim Einfügen – Bsp. 1,2,3,4,5,6,7,8

1. Einfügen der 5
2. Rechts neben 4 einfügen
3. SPLIT true
    1. Gleiche Ausrichtung
    2. Rotate(Großvater)
    3. Dad = 3
    4. Son = 4
    5. Tausche farben

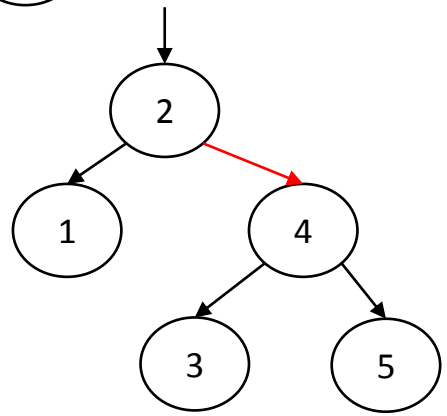
Wenn dad links = son dann dad links = son rechts und son rechts = dad

Ansonsten dad rechts = son links und Son links = dad

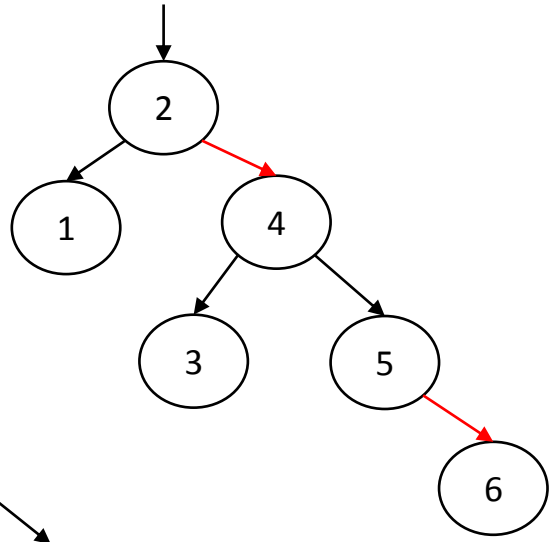




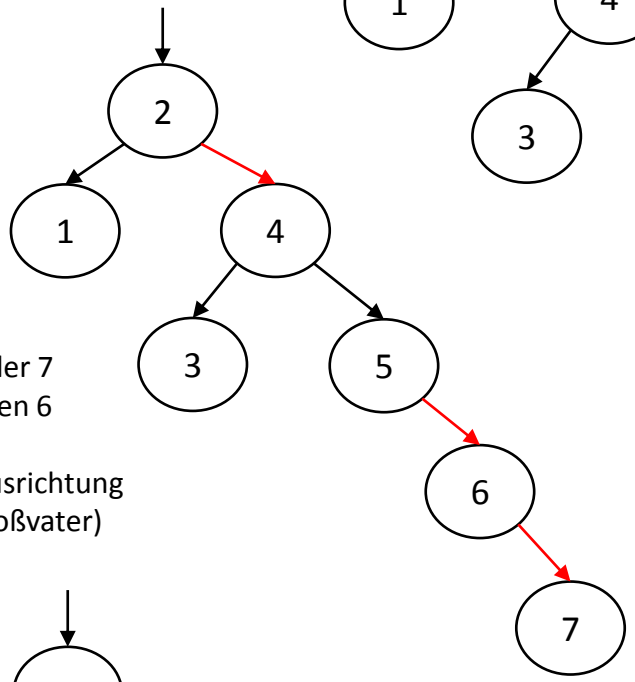
1. Einfügen der 6
2. Knoten 4 is 4er knoten
3. Konvertiere
4. SPLIT false da der Vater keine Rote Kante hat

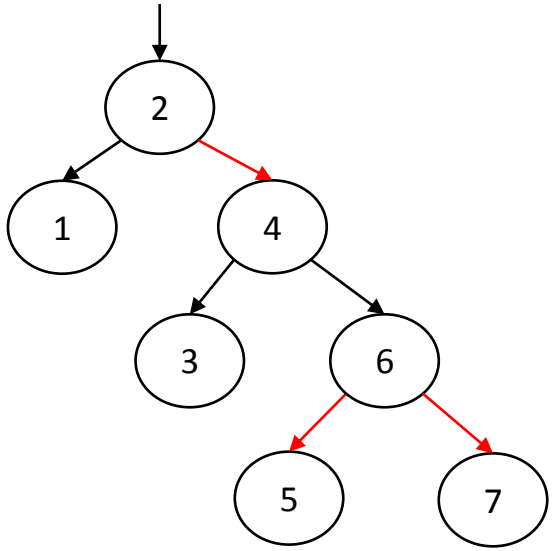


1. Rechts neben Knoten 5 einfügen
2. SPLIT false



1. Einfügen der 7
2. Recht neben 6
3. SPLIT true
4. Gleiche ausrichtung
5. Rotate(Großvater)

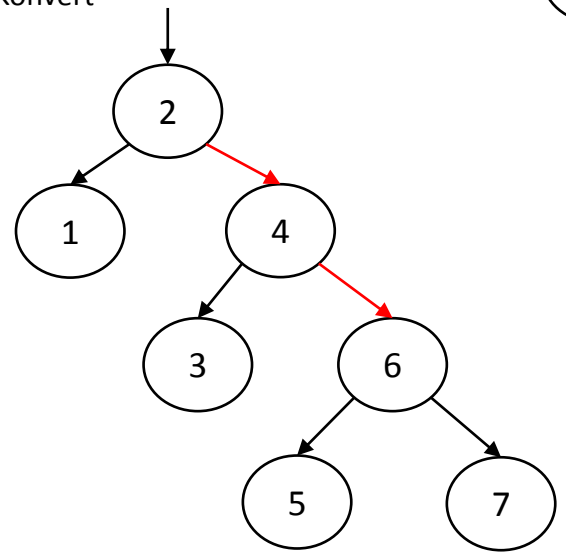


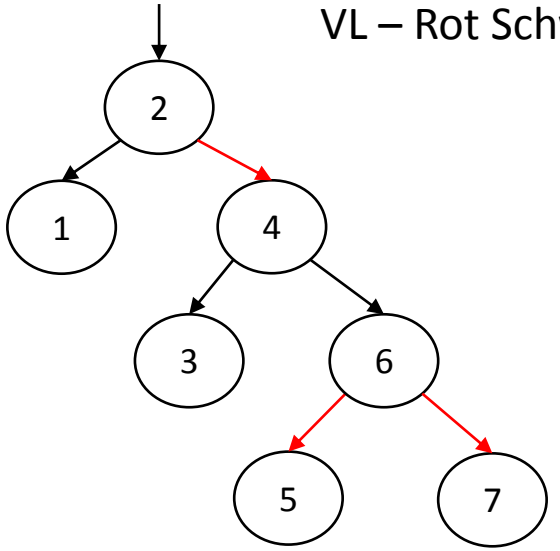


VL – Rot Schwarz Baum

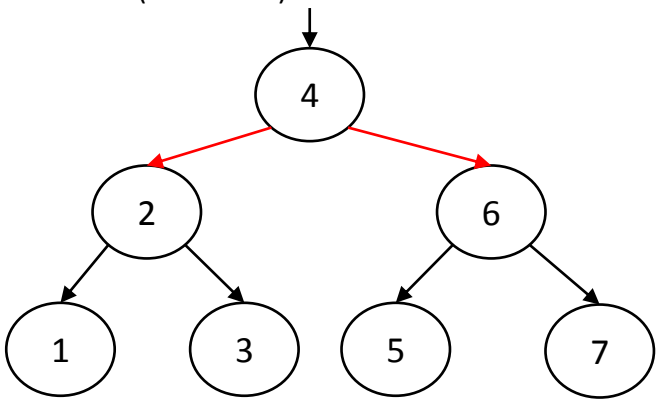
Regeln beim Einfügen – Bsp. 1,2,3,4,5,6,7,8

1. Einfügen der 8
2. Knoten 6 ist 4er Knoten
3. Konvert

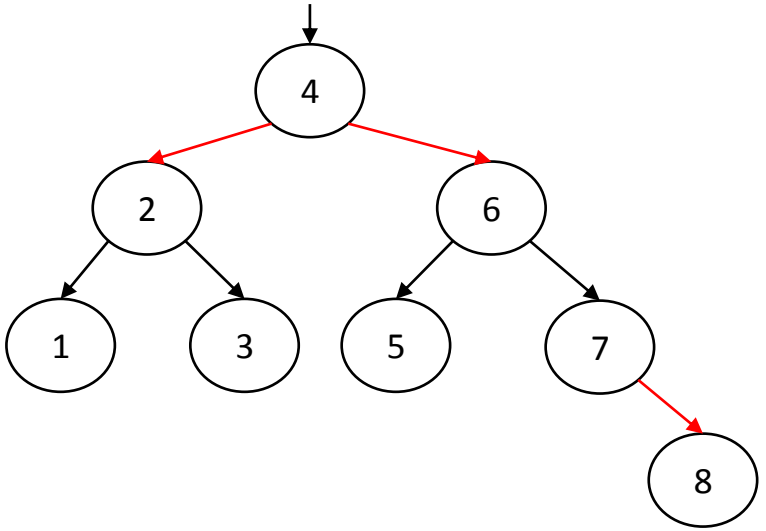




1. SPLIT true
2. Gleiche ausrichtung
3. Rotate (Großvater)



1. 8 rechts bei Knoten 7 einfügen
2. SPLIT false

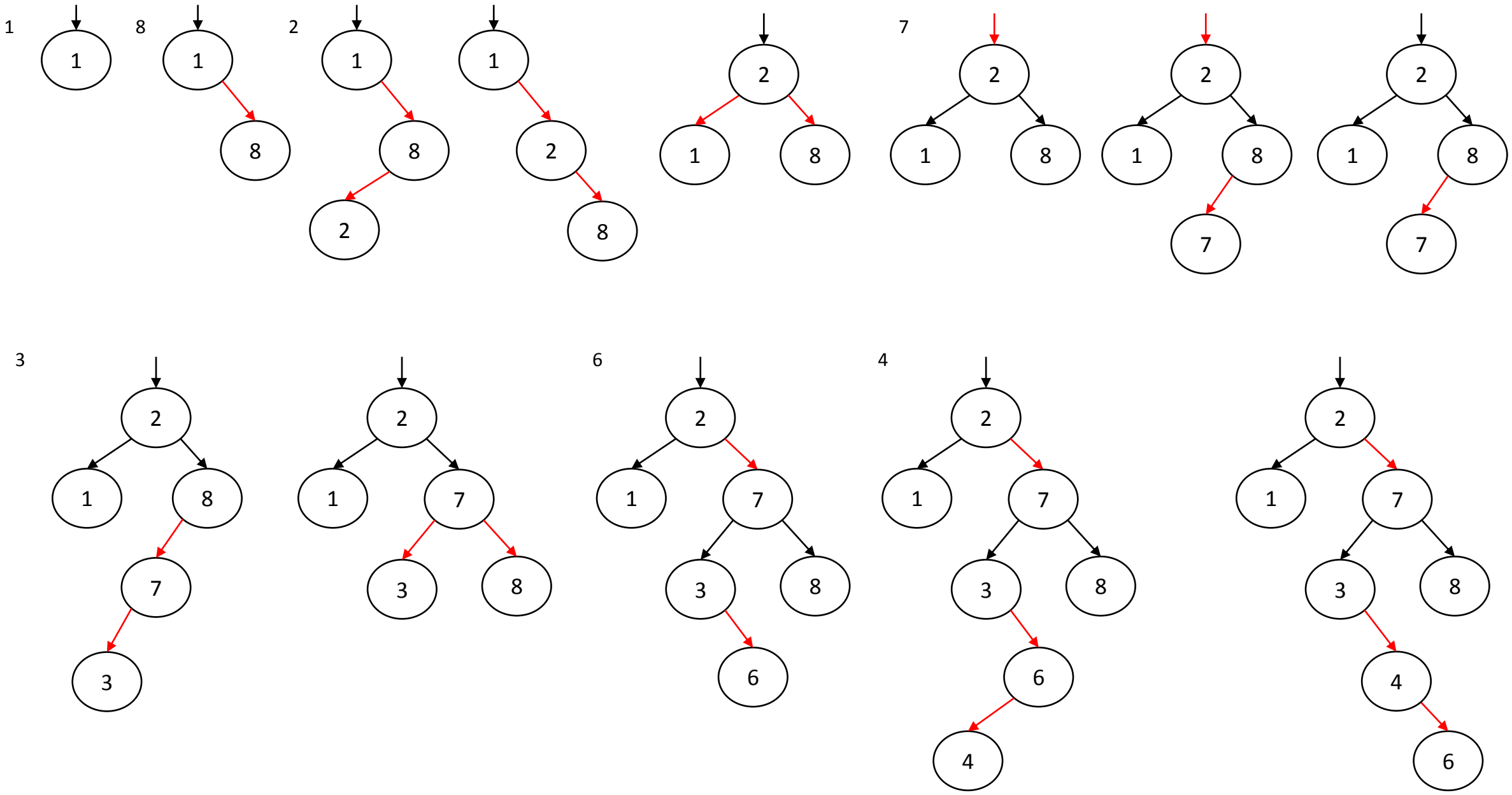


Regeln beim Einfügen

1. Beim Runterlaufen auf 4er Knoten prüfen
    1. Wenn 4er Knoten dann Konvertieren und SPLIT ausführen
    2. SPLIT prüft ob es einen Vater mit roter Kante gibt
    3. Wenn es einen Vater gibt dann Rotieren
    4. Bei Gleicher ausrichtung rotate(großvater)
    5. Bei ungleicher ausrichtung rotate(Vater) dann rotate(großvater)
2. KNOTEN SETZEN
3. SPLIT
4. M_Root = schwarz

VL – Rot Schwarz Baum

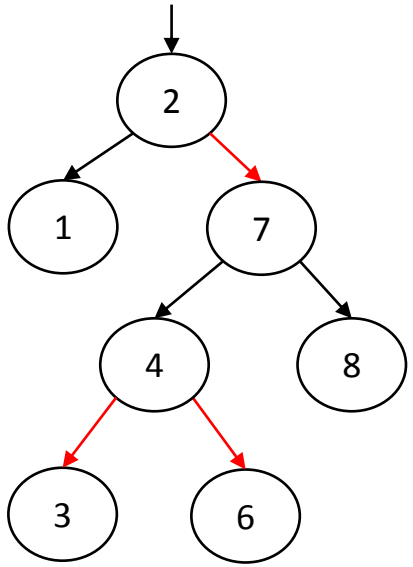
Bsp. 1, 8, 2, 7, 3, 6, 4, 5



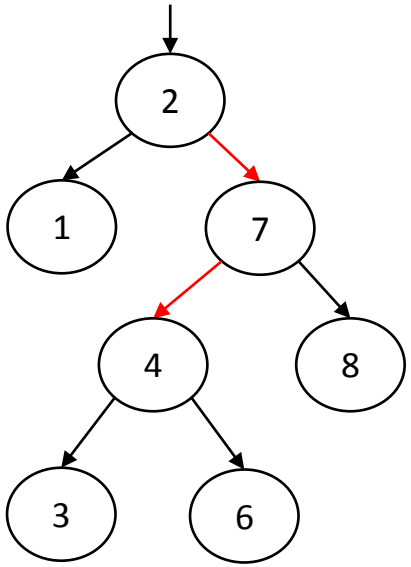
VL – Rot Schwarz Baum

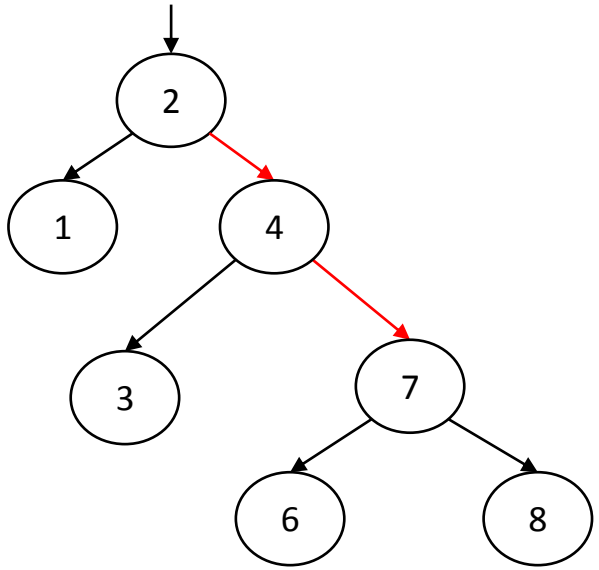
Bsp. 1, 8, 2, 7, 3, 6, 4, 5

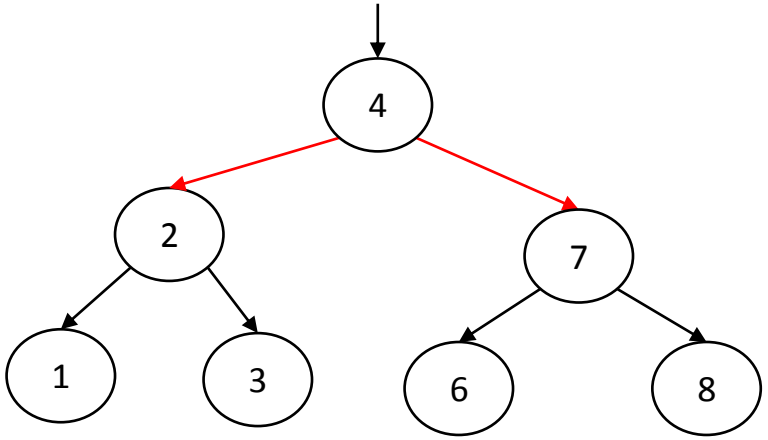
4

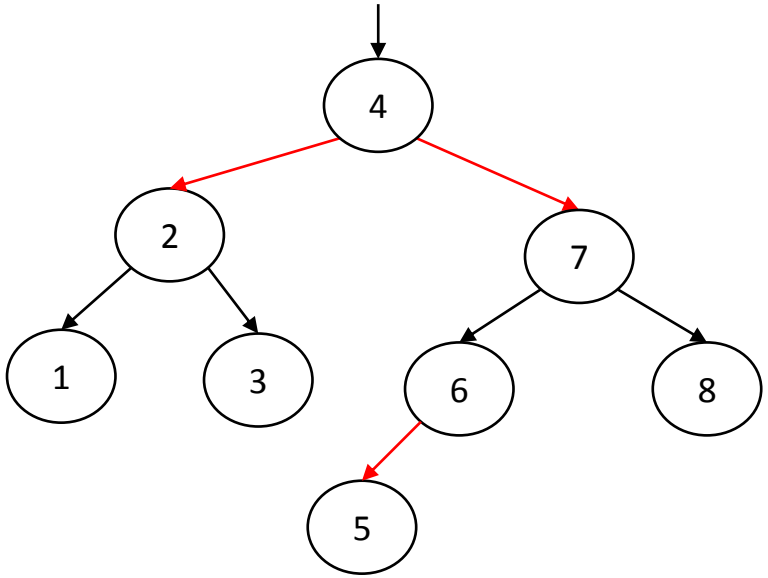


5





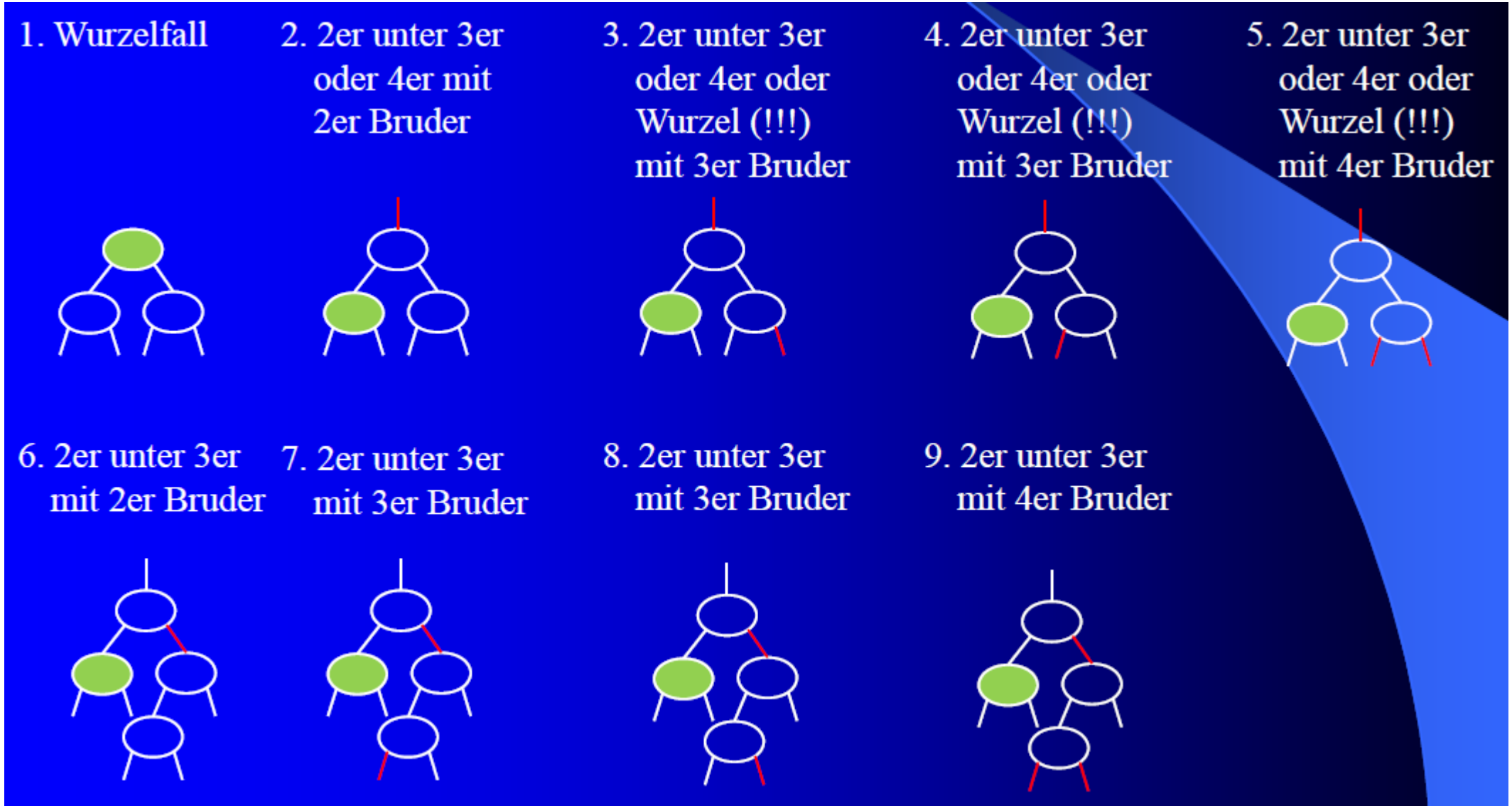




VL – Rot Schwarz Baum

Regeln beim Löschen

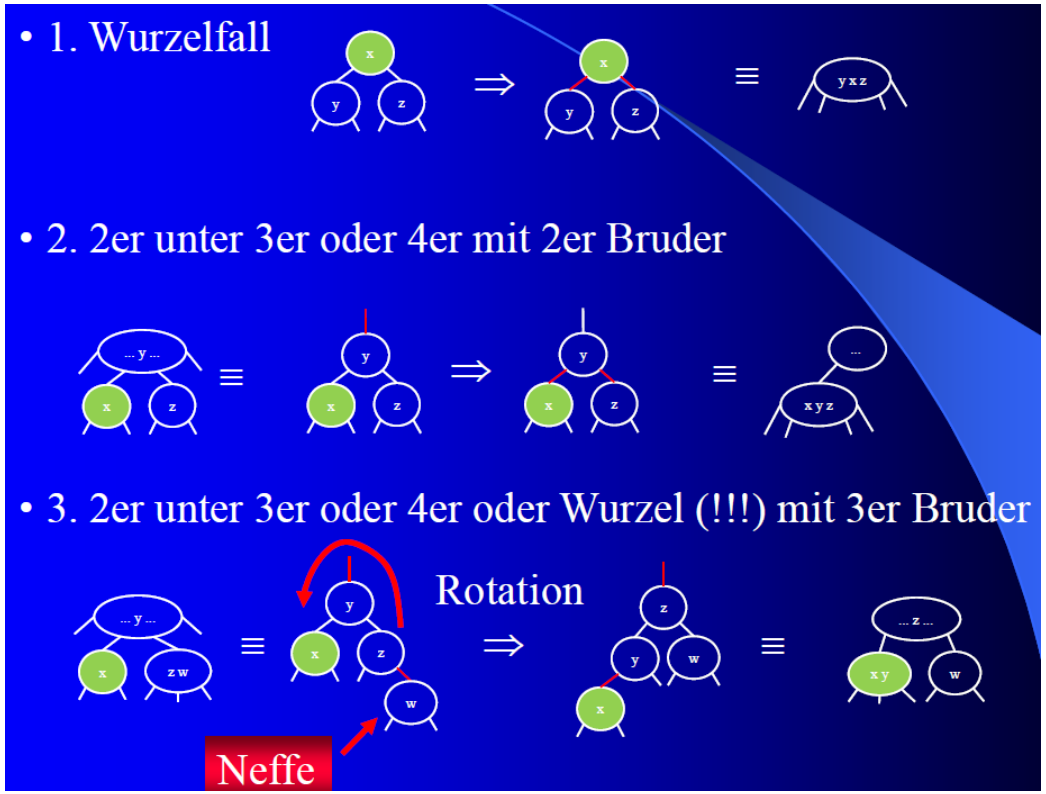
Es gibt 9 Fälle:

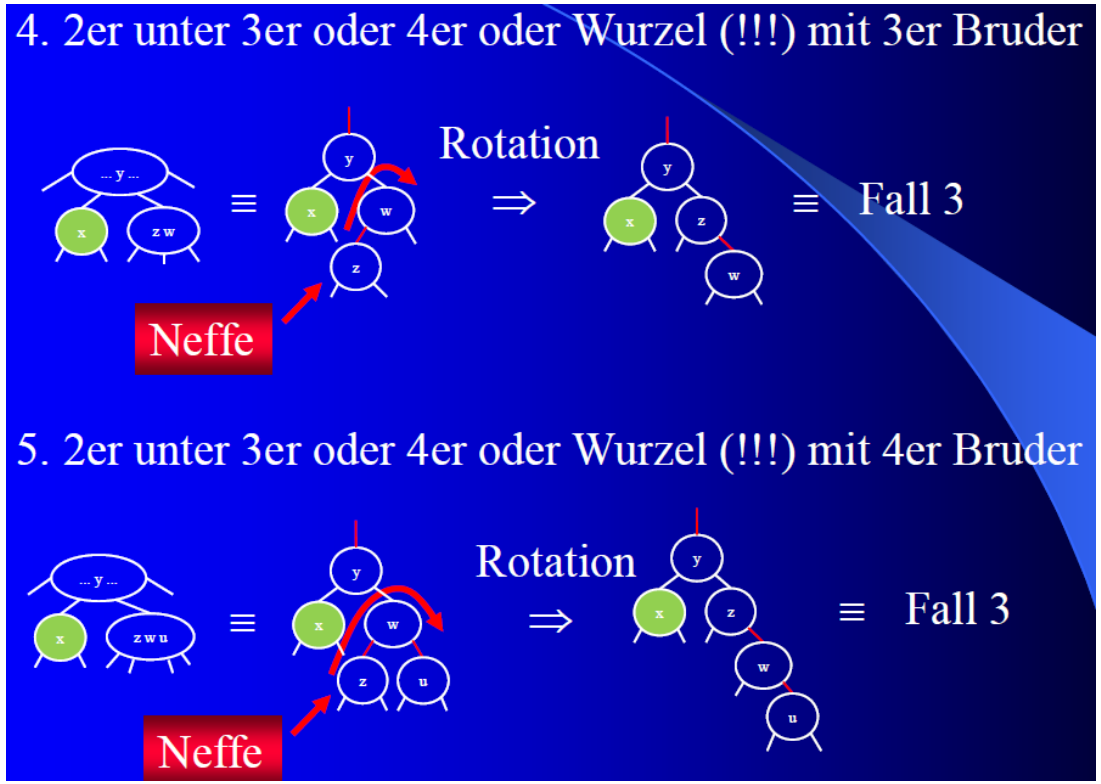


VL – Rot Schwarz Baum

Regeln beim Löschen

Es gibt 9 Fälle:

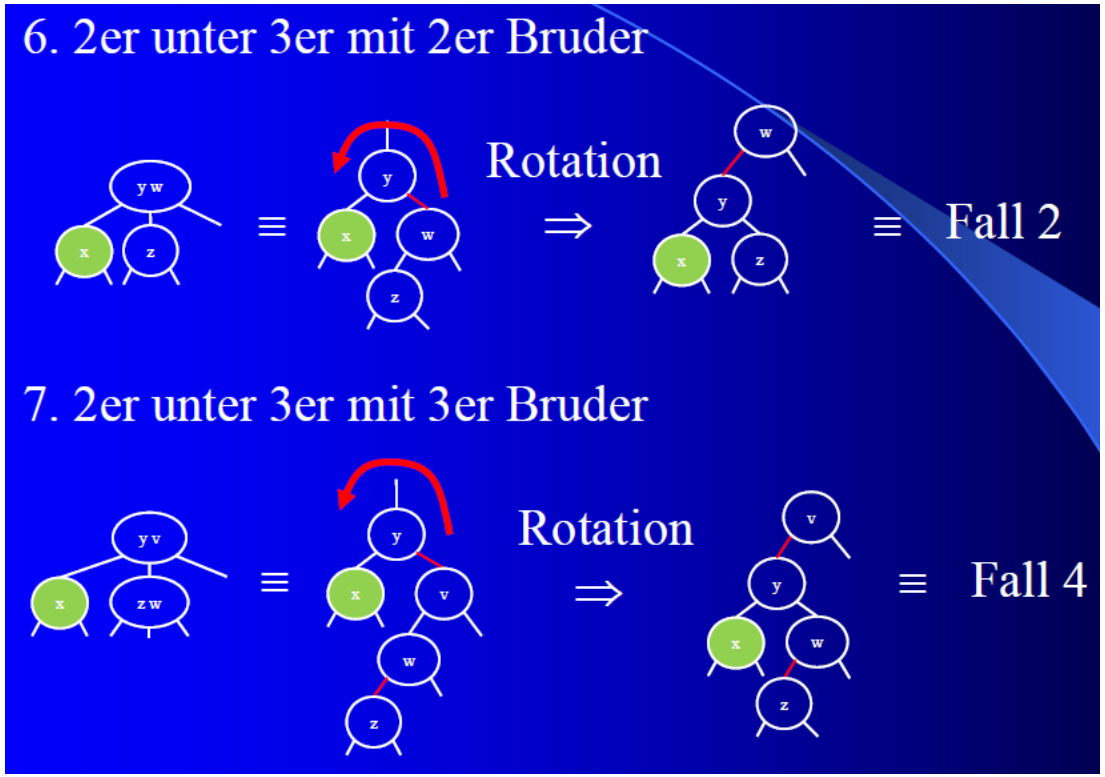


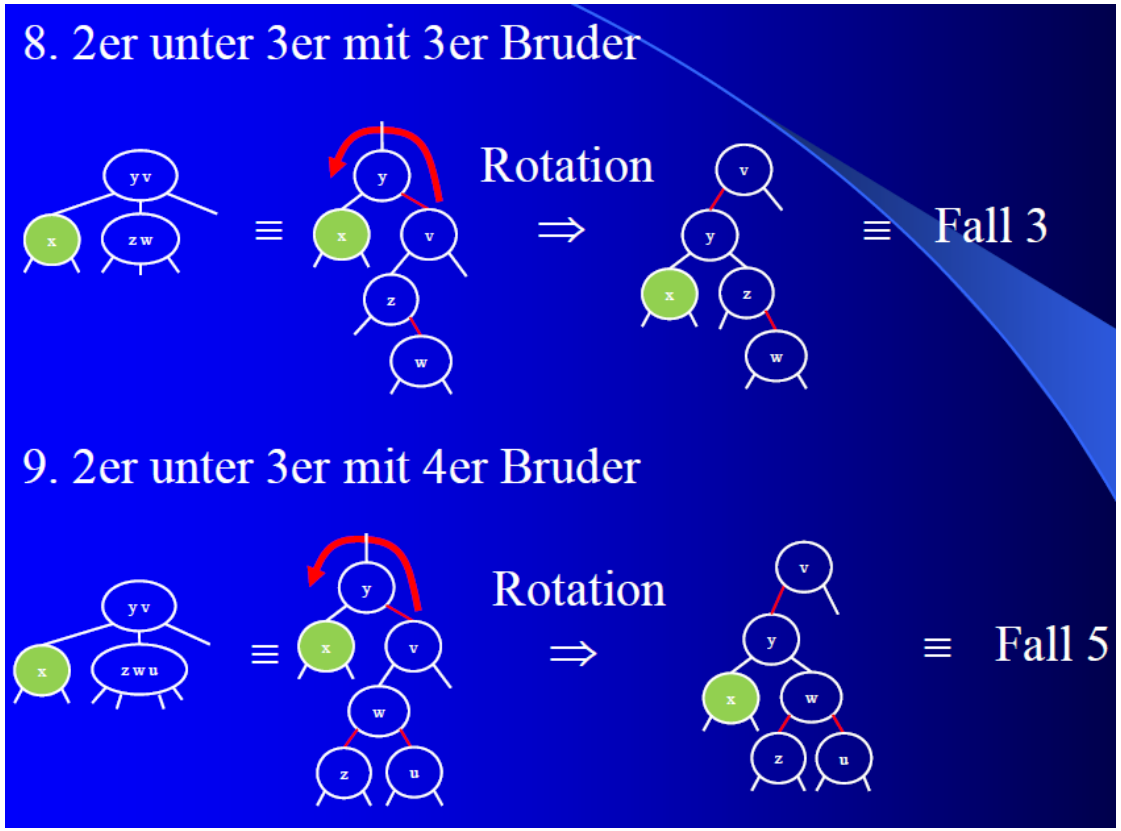


VL – Rot Schwarz Baum

Regeln beim Löschen

Es gibt 9 Fälle:





Regeln beim Löschen

1. Beim Runterlaufen auf 2er Knoten prüfen JOIN()
    1. If(FALL1 Wurzelfall) – Fall 1 = Kein Vater, beide Nachfolger 2er Knoten
        1. Beide Nachfolger = ROT
    2. Else IF(Gibt es einen Vater)
        1. If(Bruder == ROT) FALL 6-9
            1. Rotiere(Bruder um Vater)

        2. If(Bruder == 2er Knoten) FALL 2

        3. Else{
            1. If(gibt es einen Neffen und ist er ROT) FALL 4-5
                1. Rotiere(Neffe um Bruder)

            2. Fall 3
                1. Rotiere Bruder um Vater

VL – Rot Schwarz Baum

Regeln beim Löschen

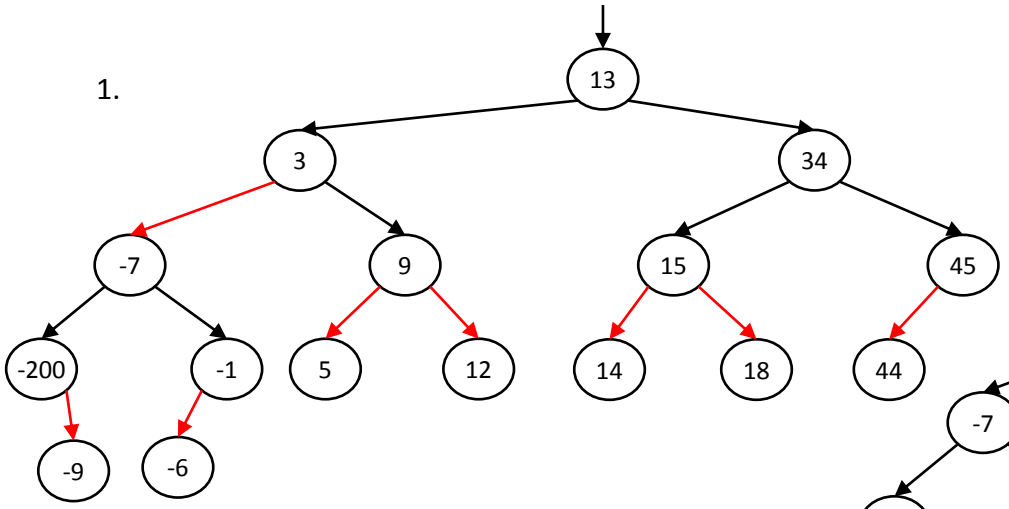
Bsp.: 13, -7, 45, 12, -1

1. Löschen von 13

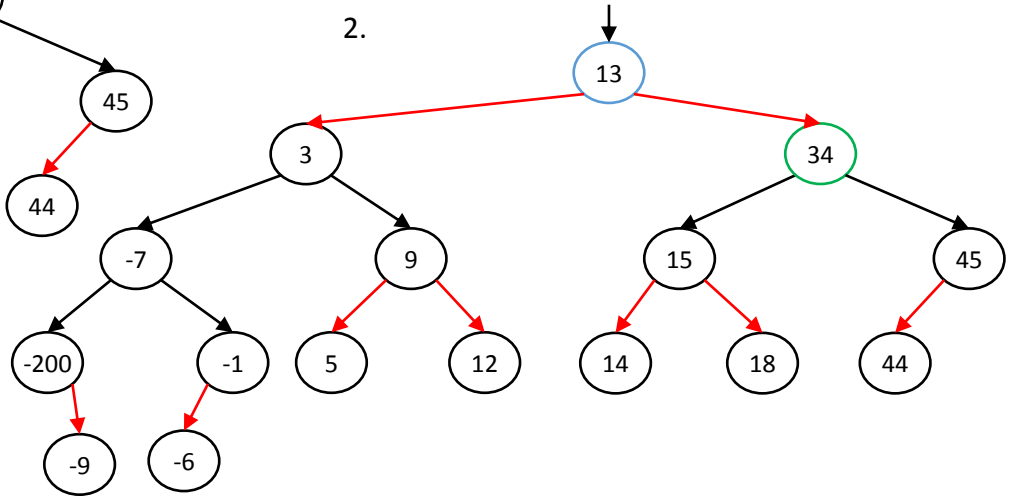
REGELN:

1. JOIN
    1. If 2erKnoten
    2. If Wurzelfall
        1. LEFT und RIGHT = ROT
    3. Elseif Vater != null
        1. If BRUDER ist ROT
            1. FALL 6-9 BRUDER und Vater Rotieren, Nefen im letzten Schritt nicht abfragen!!!
        2. If BRUDER ist 2er Knote
            1. FALL 2
        3. Else
            1. if Nefe = ROT
                1. Rotiere(Nefe, Vater von Nefe)
            2. Rotiere Bruder und Vater
2. Rotation
    1. If Son = ROT?
        1. Son LEFT und RIGHT auf Schwarz setzen
        2. Vater auf Schwarz setzen
        3. Vater LEFT und RIGHT auf ROT setzen
    2. Else
        1. Vater und sohn tauschen Farben (wie beim einfügen)
    3. Rotation

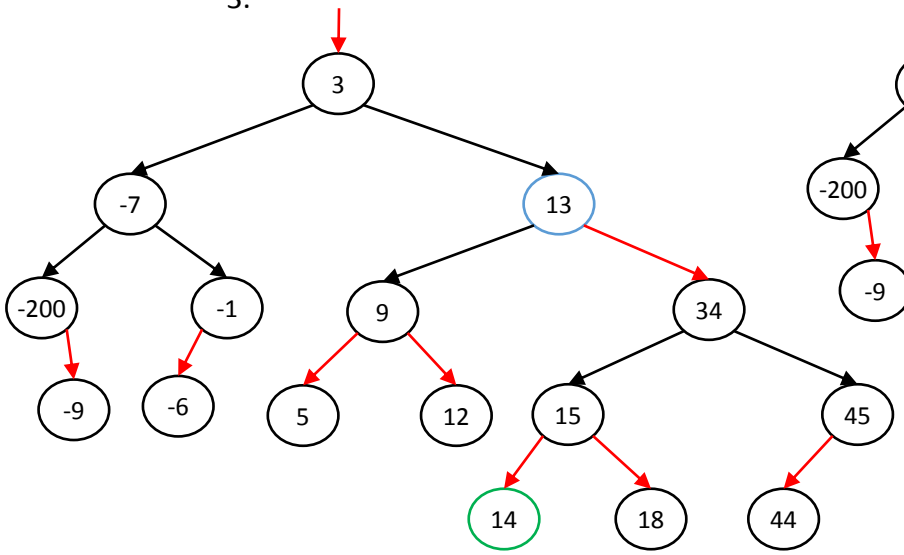
1.



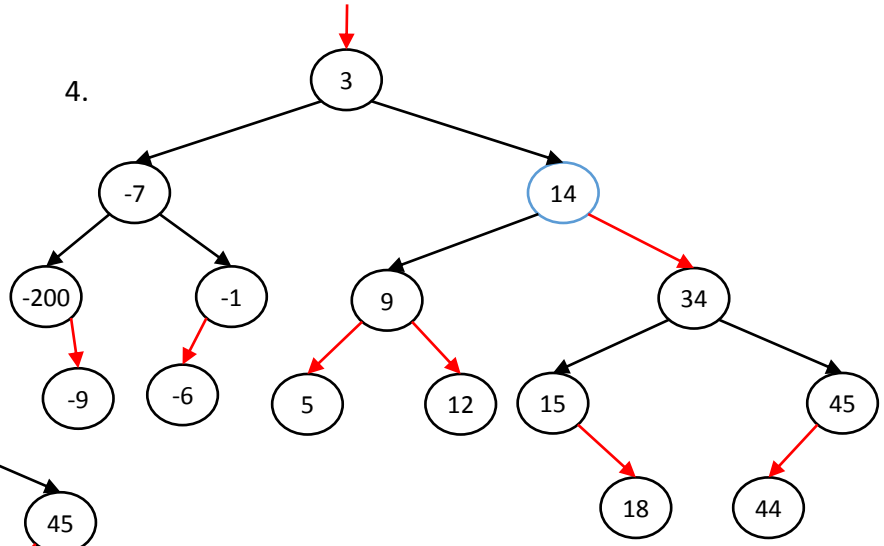
2.



3.



4.



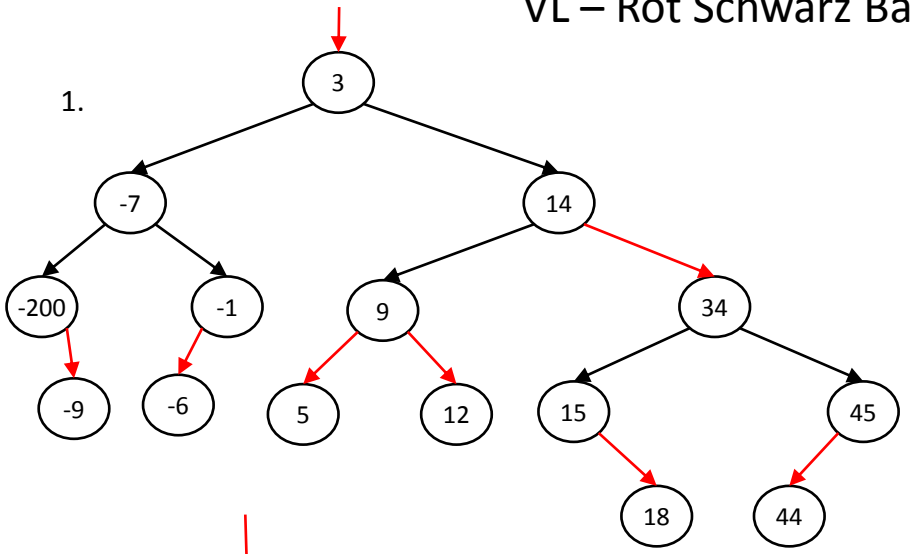
VL – Rot Schwarz Baum

Regeln beim Löschen

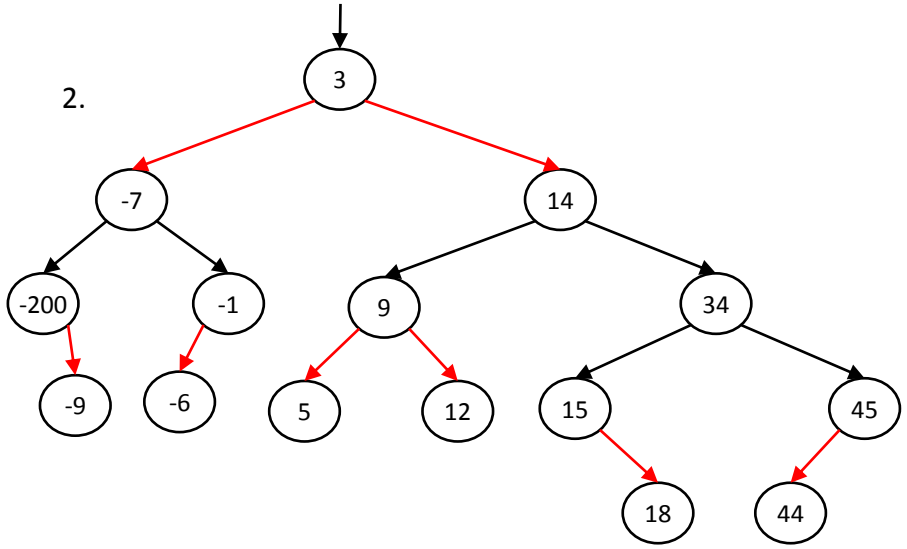
Bsp.: 13, -7, 45, 12, -1

1. Löschen von -7

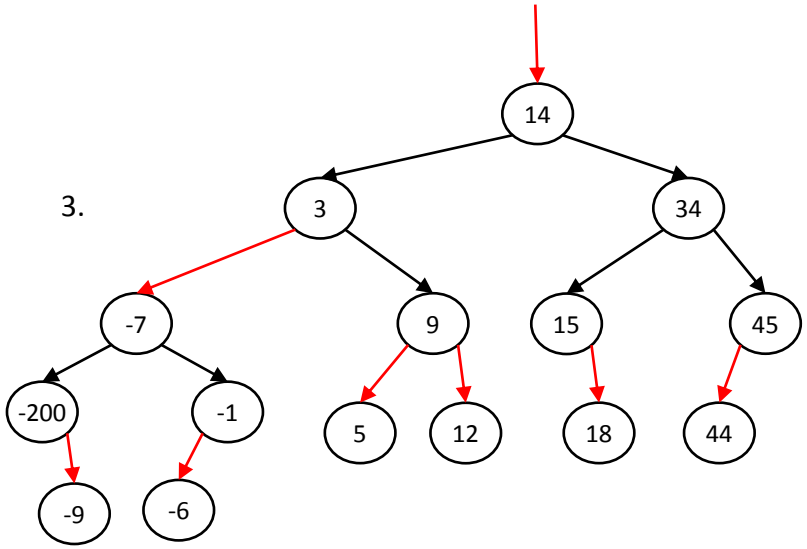
1.



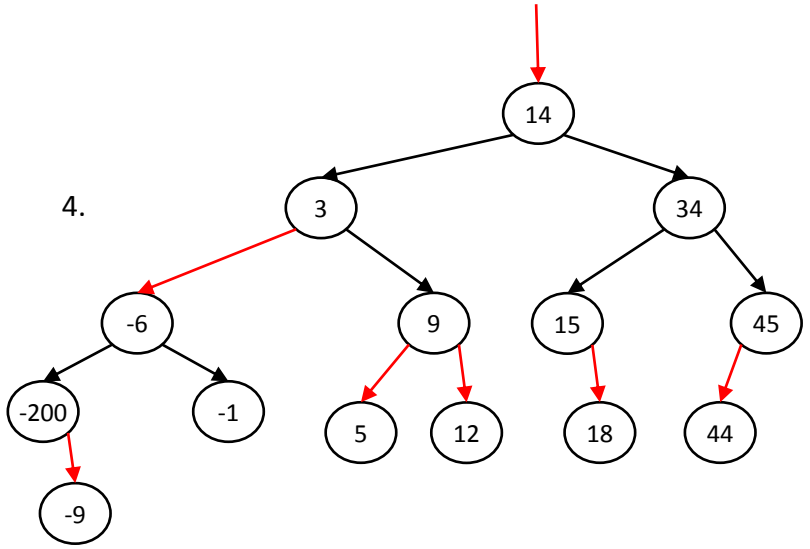
2.



3.



4.



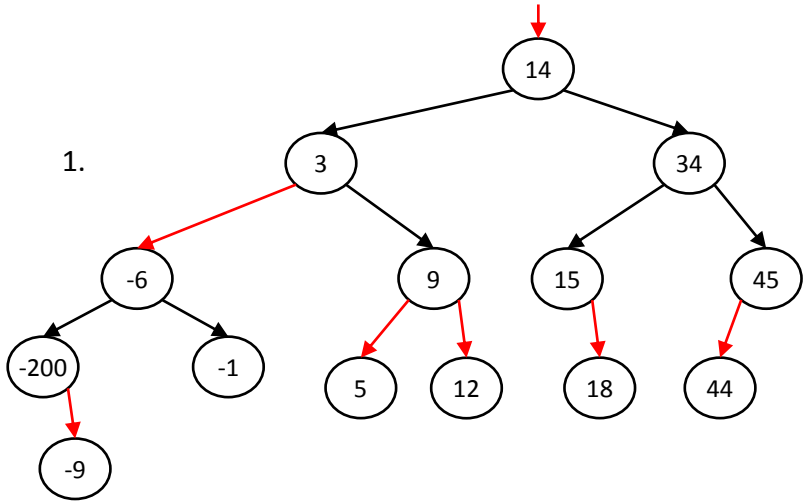
VL – Rot Schwarz Baum

Regeln beim Löschen

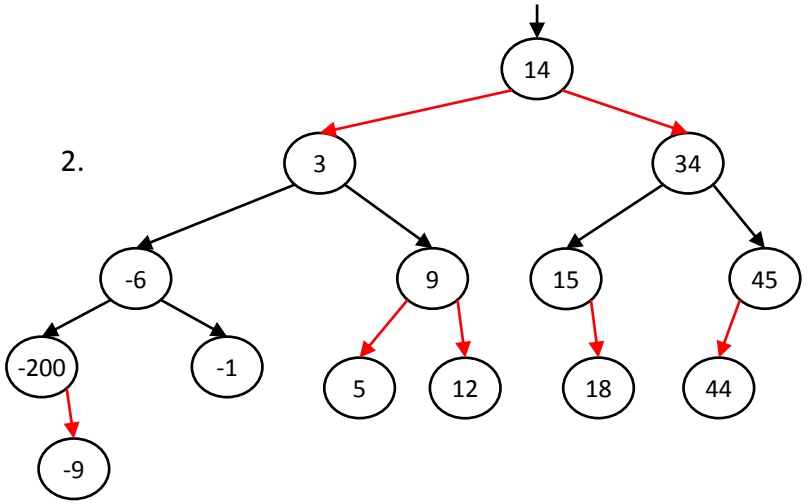
Bsp.: 13, -7, 45, 12, -1

1. Löschen von 45

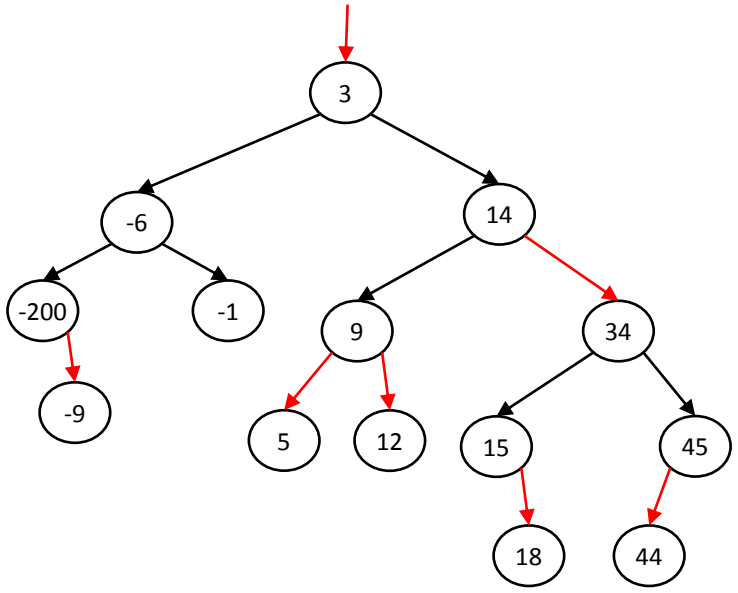
1.



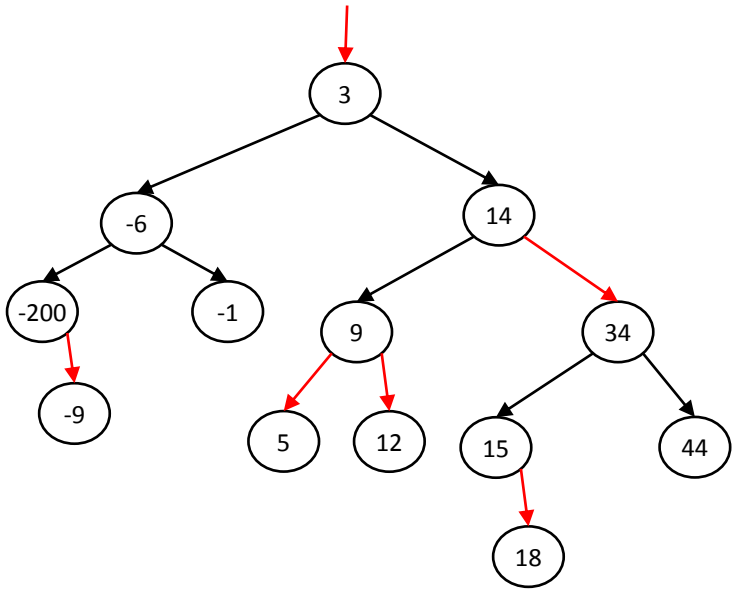
2.



3.



4.



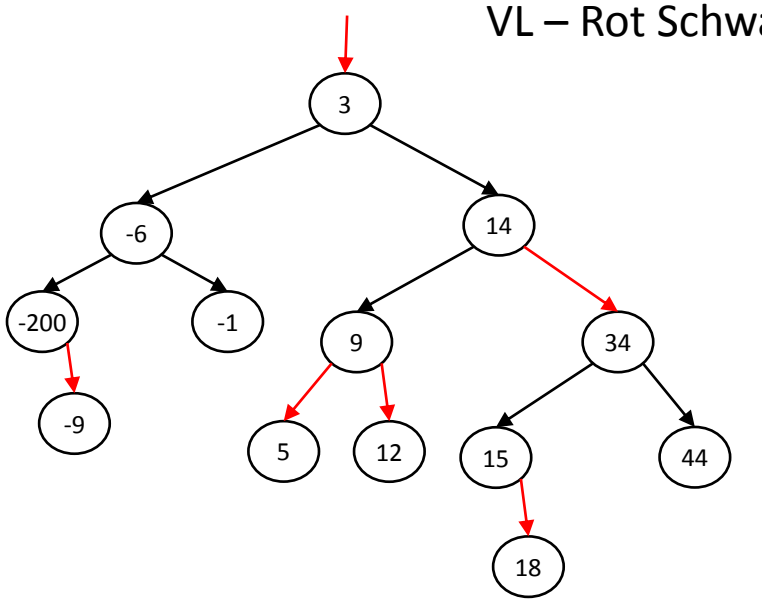
VL – Rot Schwarz Baum

Regeln beim Löschen

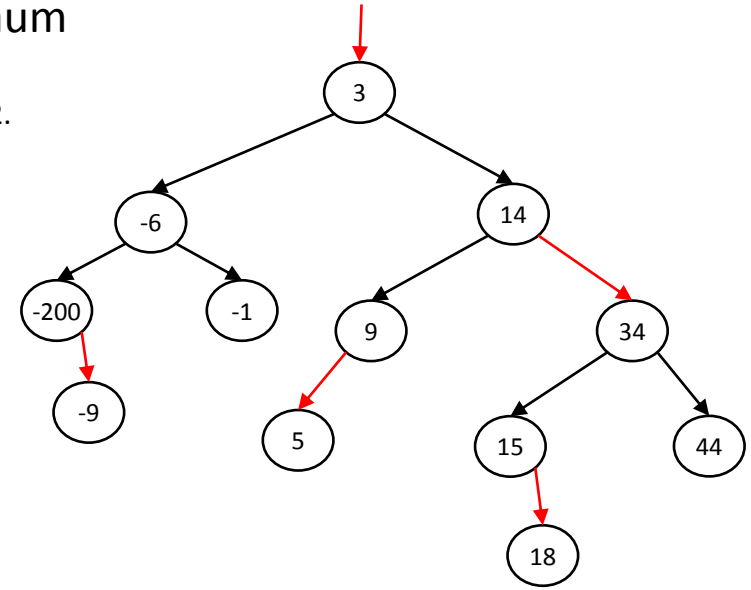
Bsp.: 13, -7, 45, 12, -1

1. Löschen von 12

1.



2.



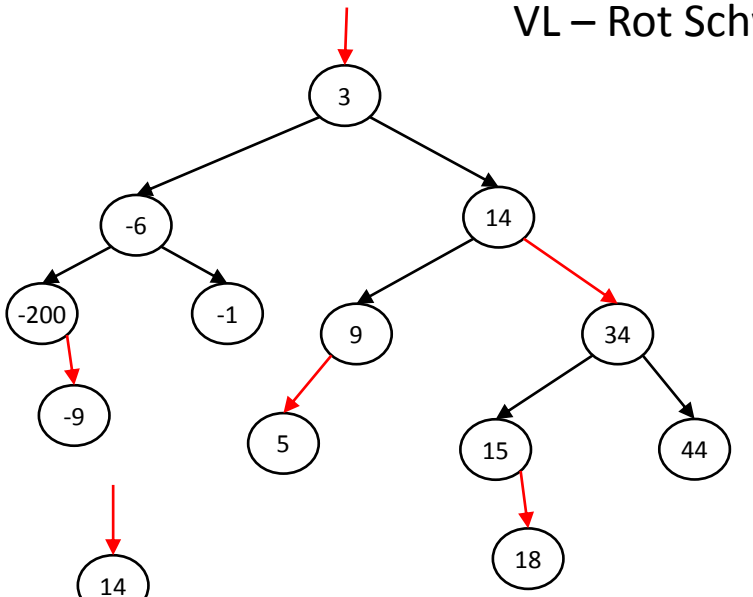
VL – Rot Schwarz Baum

Regeln beim Löschen

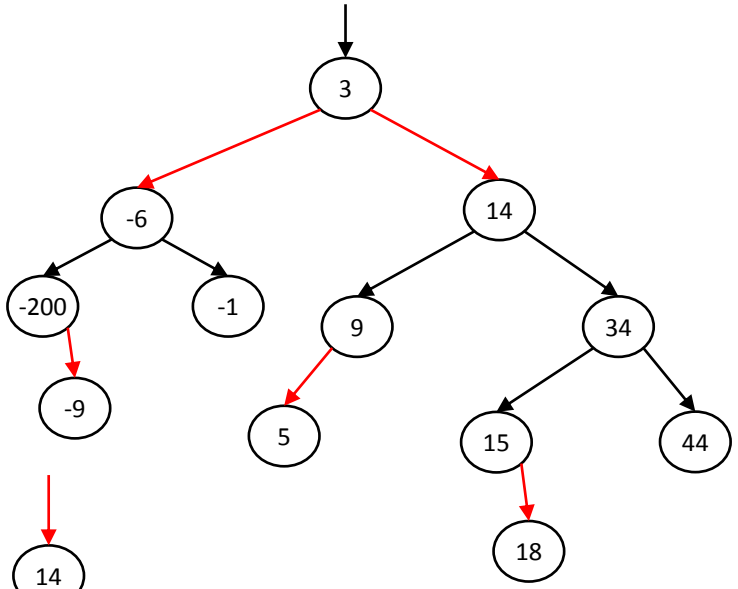
Bsp.: 13, -7, 45, 12, -1

1. Löschen von -1

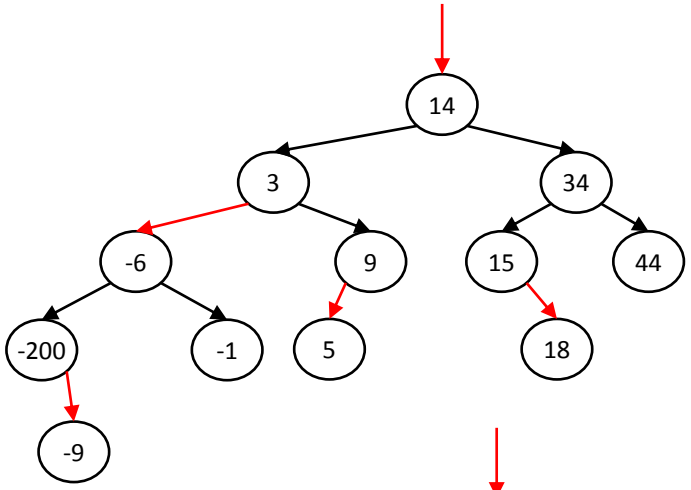
1.



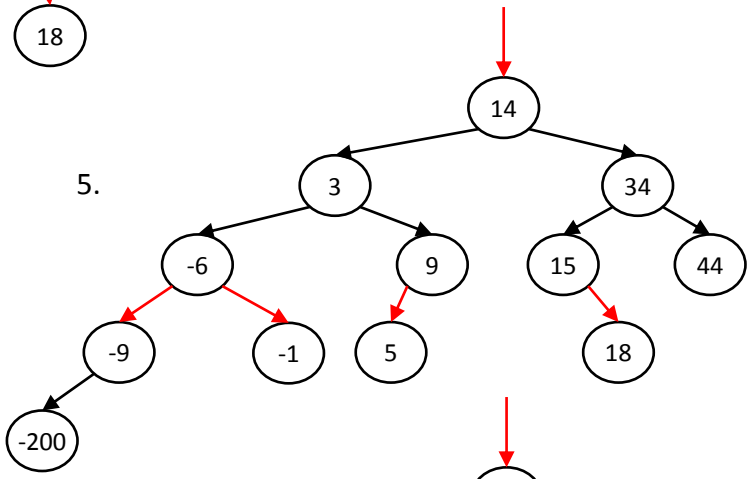
2.



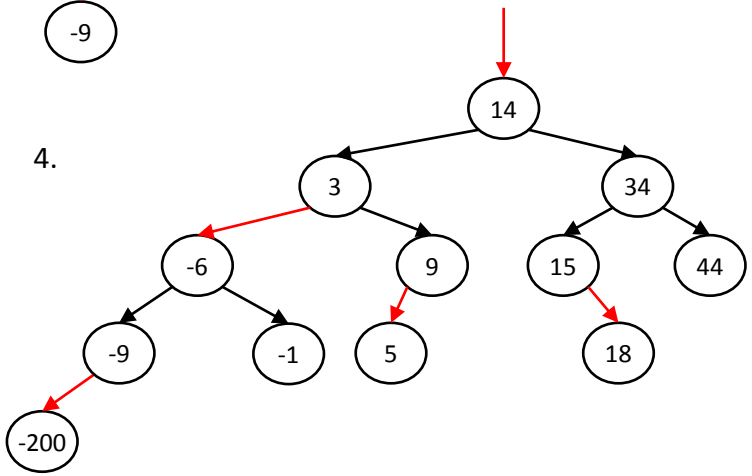
3.



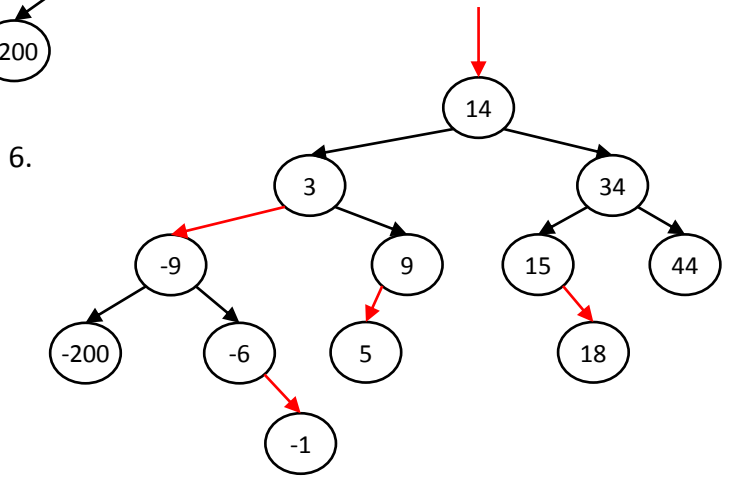
5.



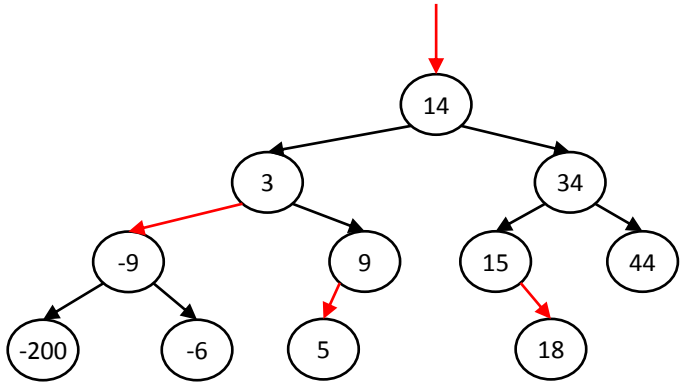
4.



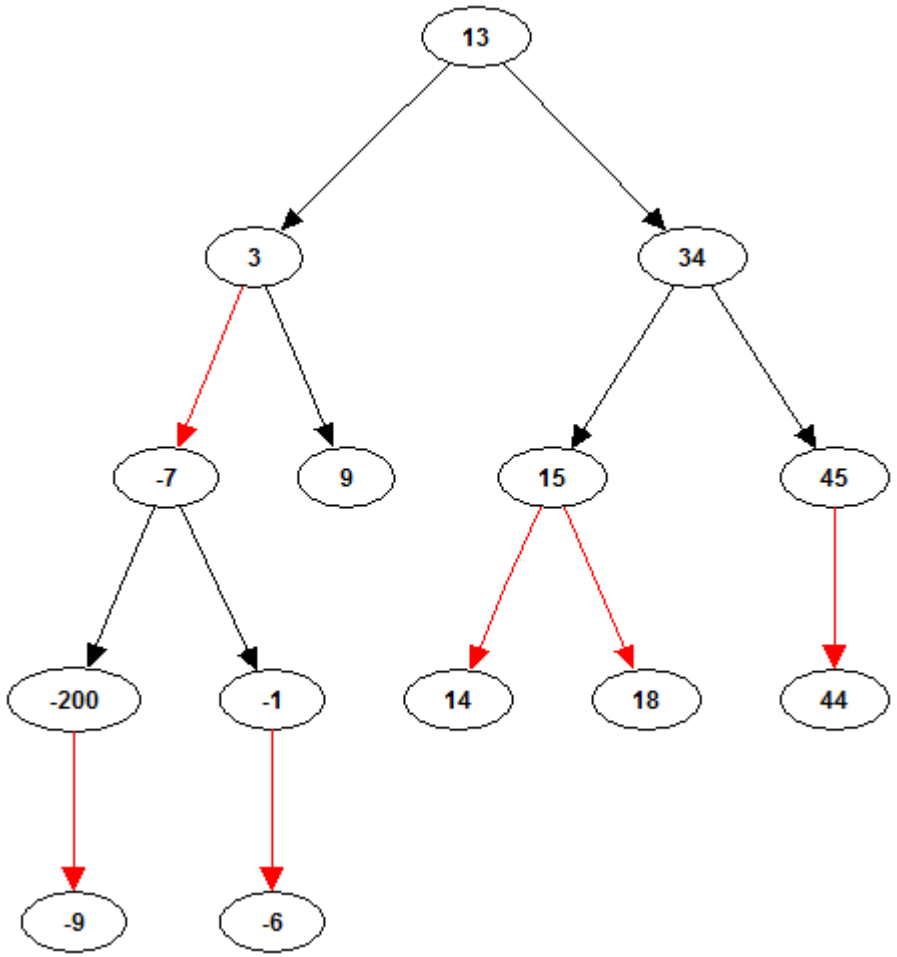
6.



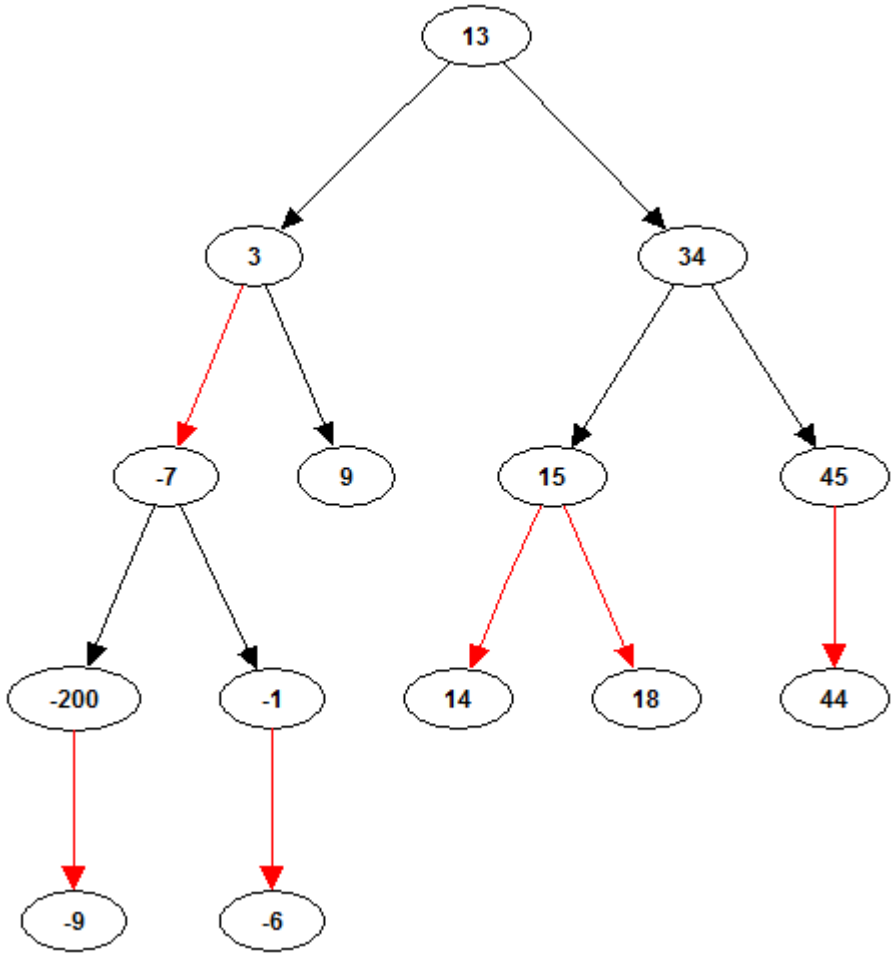
6.

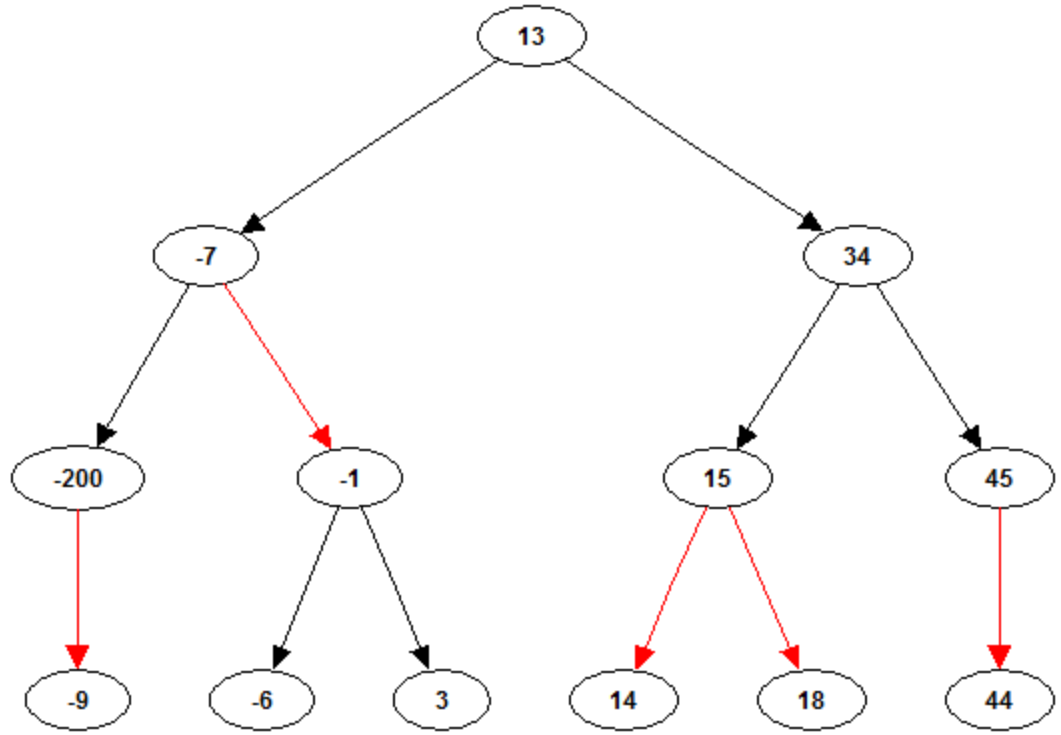


9 LÖSCHEN

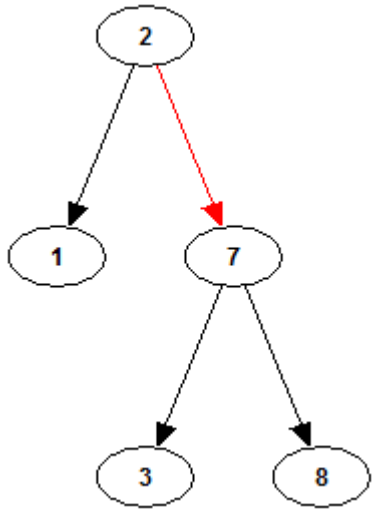


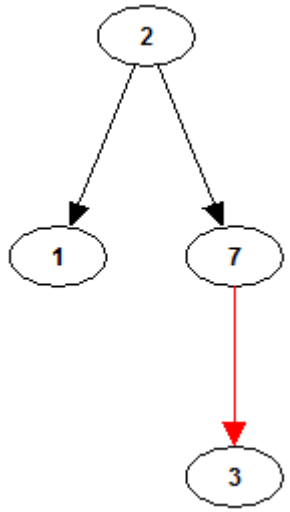
9 LÖSCHEN





8 LÖSCHEN





VL 12 – Patricia Tree

Gegeben sei die Buchstabenreihenfolge „qwert“. Zeichnen Sie den Patricia Tree, nachdem die Buchstaben in der vorgegeben Reihenfolge eingefügt worden sind. Die binären Werte für Buchstaben lauten:

q = 10001
w = 10111
e = 00101
r = 10010
t = 10100

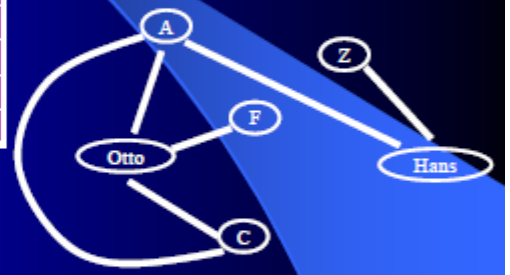
Adjanzenzmatrizen

Adjanzenzlisten

Implementierung

- Implementierung mittels ***Adjazenzmatrizen*** oder ***Adjazenzlisten***
- Adjazenzlisten verwendet man bei ***lichten*** Graphen (Anzahl der Kanten relativ klein)
- Adjazenzmatrizen verwendet man bei ***dichten*** Graphen (Anzahl der Kanten relativ hoch)

	0	1	2	3	4	5
0		x		x	x	
1	x		x	x		
2		x				
3	x	x				
4	x					x
5					x	



- Adjazenzmatrizen (ungewichteter Graph):
  - Nummerierung der Knoten von 0 bis |V|-1
  - 2-dimensionales boolesches Feld ***m*** mit m[i,j]=true $\Leftrightarrow (v_i,v_j) \in E$
- Adjazenzmatrizen (gewichteter Graph):
  - Nummerierung der Knoten von 0 bis |V|-1
  - 2-dimensionales Float (o.ä.) Feld ***m*** mit m[i,j]=f $\Leftrightarrow (v_i,v_j,f) \in E$

VL 15 – Huffmann