Kompressionsalgorithmen: Deflate und bzip2

Seminar „Effiziente Programmierung“

Lennart Uhrmacher
10.01.2019

Was ist Deflate?

Deflate

- Algorithmus zur verlustlosen Datenkompression
- Hardwareunabhängig, patentfrei
- Ursprünglich für das ZIP-Format entwickelt
- Es gibt einige Implementation des Deflate-Algorithmus:
  - ZIP
  - GZIP
  - Zopfli

Deflate-Algorithmus

Deflate-Algorithmus

1. Datei wird in Blocks aufgeteilt (häufig 32 KB)

Deflate-Algorithmus

1. Datei wird in Blocks aufgeteilt (häufig 32 KB)
2. LZ77-Algorithmus wird angewandt, um doppelte Zeichenketten zu eliminieren

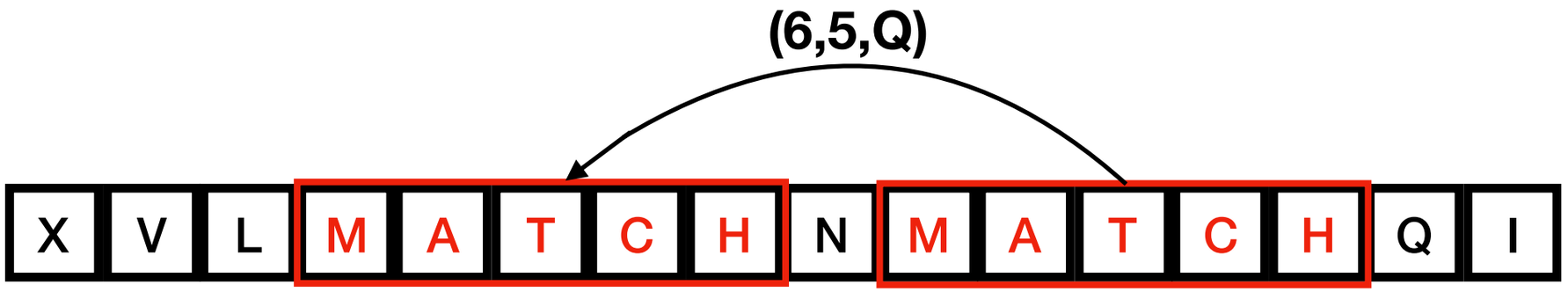
Deflate-Algorithmus

1. Datei wird in Blocks aufgeteilt (häufig 32 KB)
2. LZ77-Algorithmus wird angewandt, um doppelte Zeichenketten zu eliminieren
3. Huffman-Codierung zur Bitreduktion

LZ77

- Wenn Wörter mehrfach vorkommen, speichert man sie nicht erneut, sondern nur eine Referenz auf das erste vorgekommene Wort

- Deduplikation



Huffman-Kodierung

- Wort: ABCD

Huffman-Kodierung

- Wort: ABCD
- UTF-8:
  - 01000001 01000010 01000011 01000100
  - Mindestens 8 Bit pro Zeichen

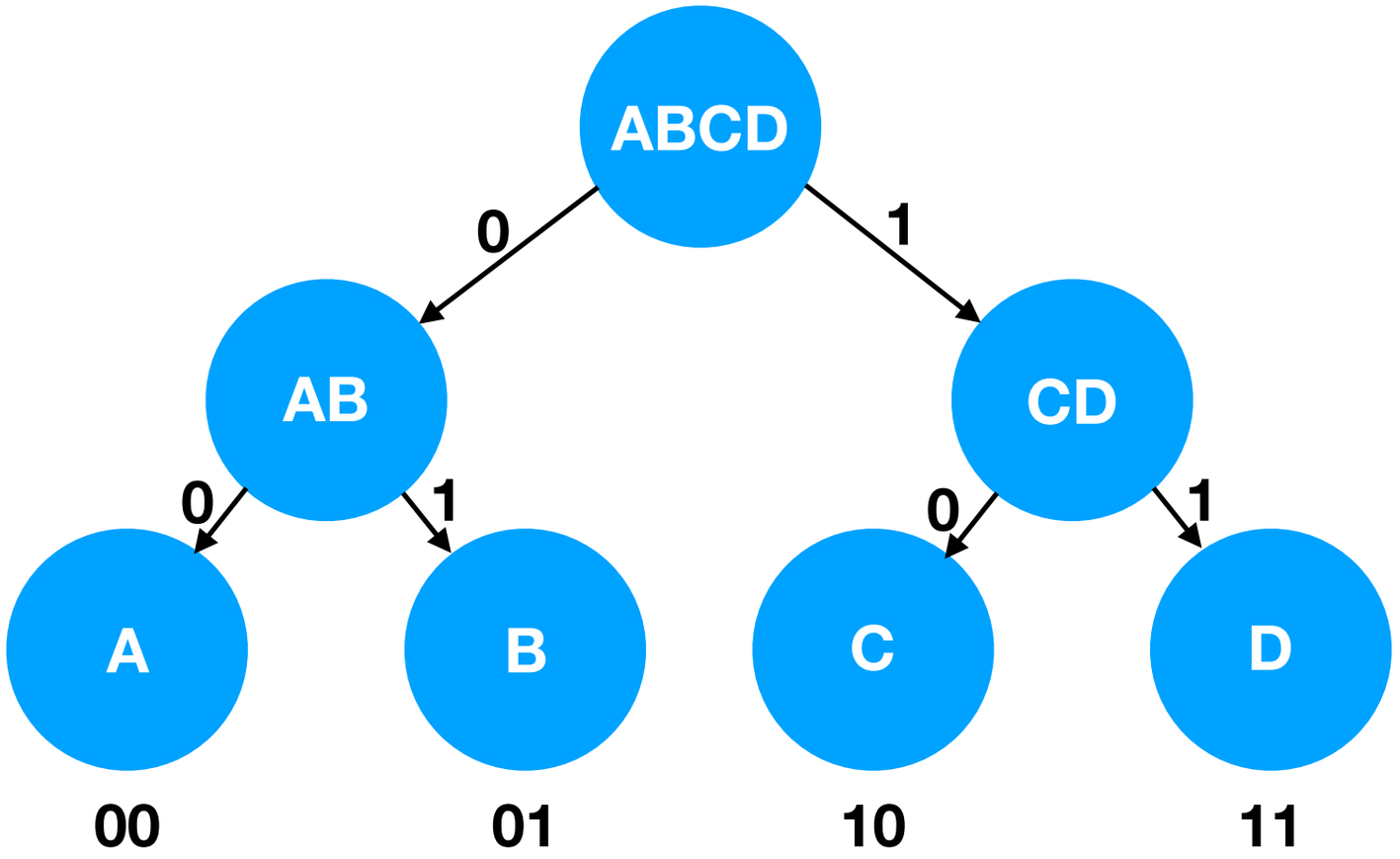
Huffman-Kodierung

- Wort: ABCD
- UTF-8:
  - 01000001 01000010 01000011 01000100
  - Mindestens 8 Bit pro Zeichen **Überflüssig!**

Huffman-Kodierung

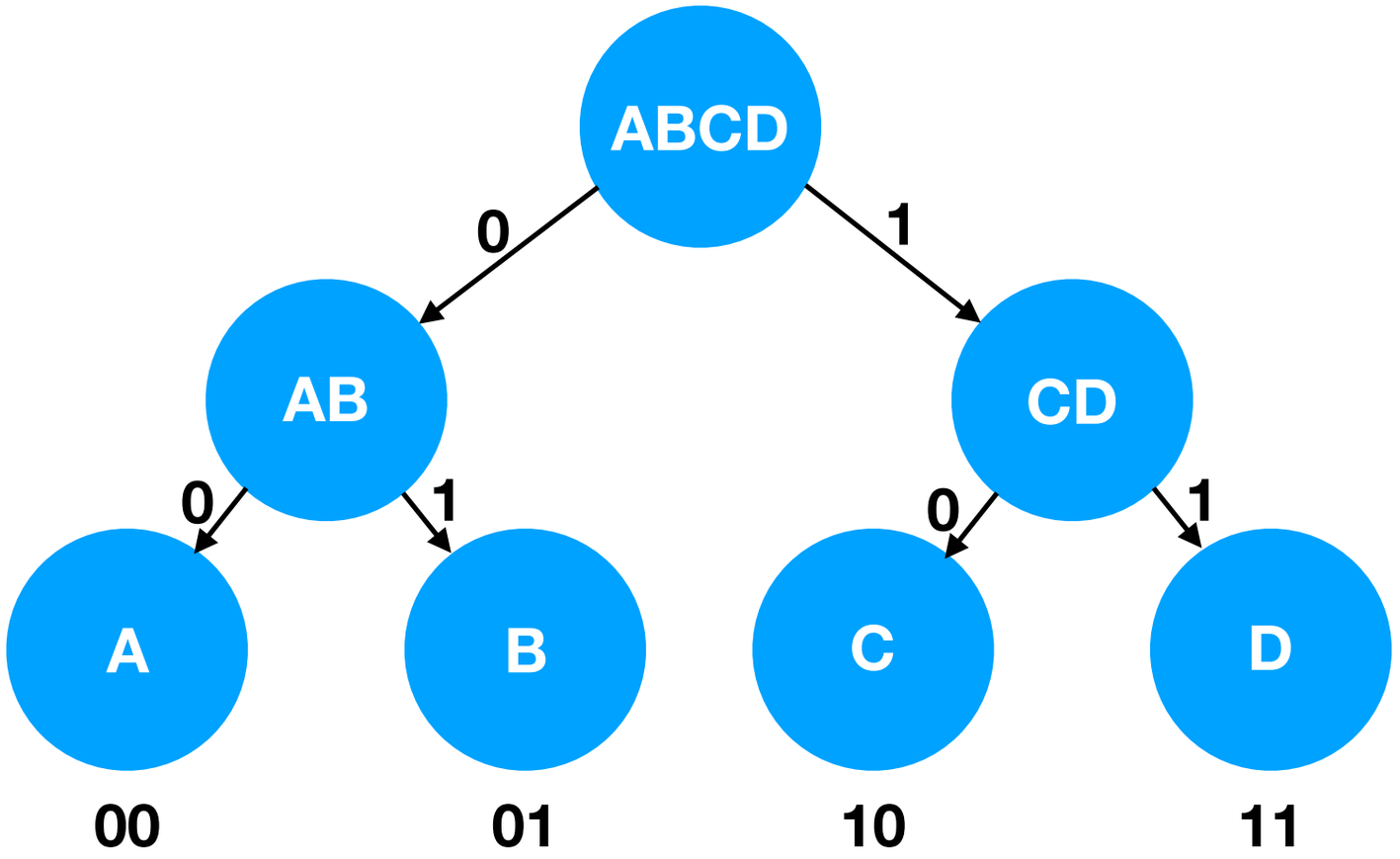
- Wort: ABCD
- UTF-8:
  - 01000001 01000010 01000011 01000100
    - Mindestens 8 Bit pro Zeichen **Überflüssig!**
- Besser:
  - 00 01 10 11
  - 2 Bit pro Zeichen

Huffman-Kodierung



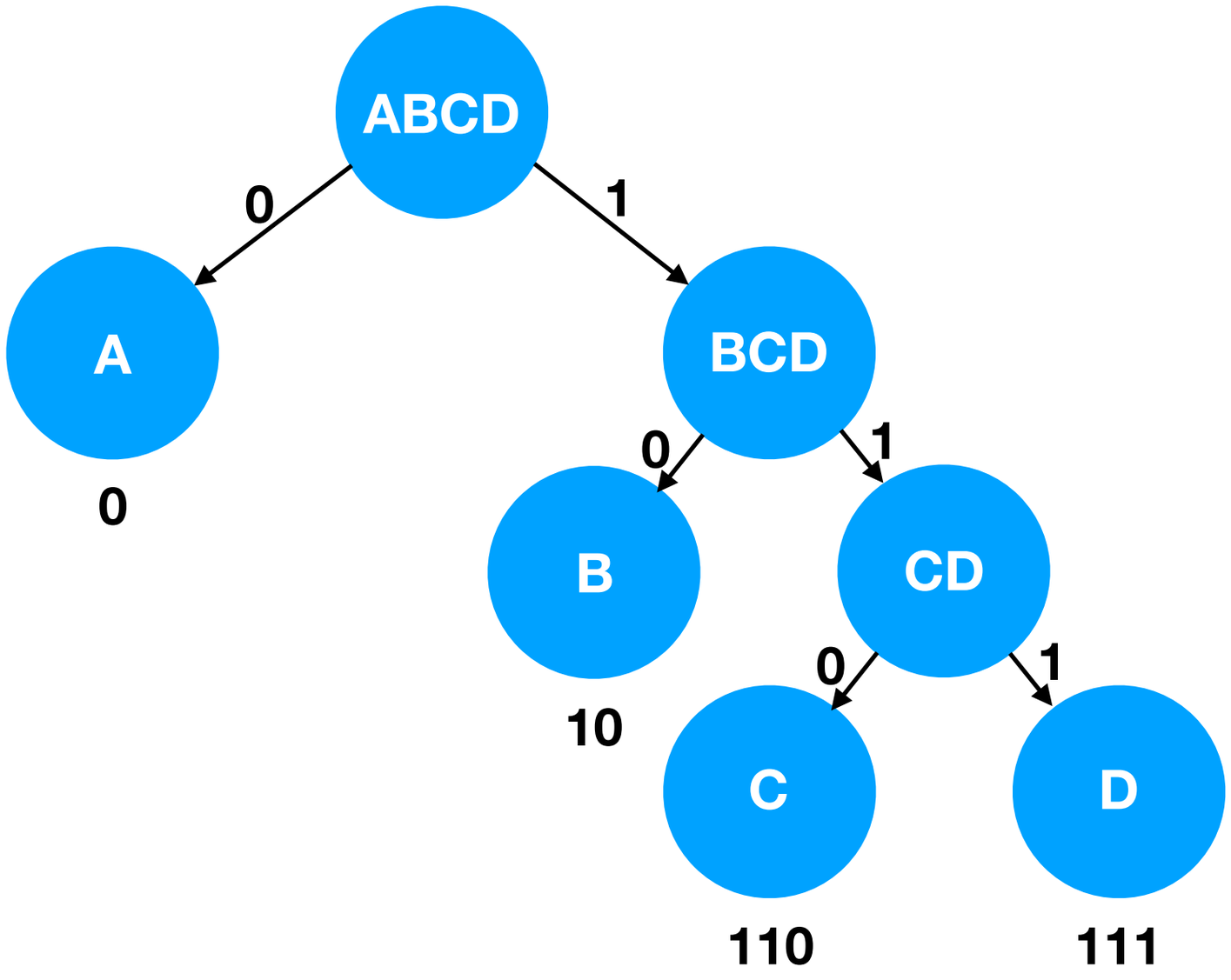
- ABCD = 00 01 10 11
- 2 Bit pro Zeichen

Huffman-Kodierung



- AAAAAABBCD = 00 00 00 00 00 00 01 01 10 11
- 2 Bit pro Zeichen

Huffman-Kodierung



- AAAAAABBCD = 0 0 0 0 0 0 10 10 110 111
- 1,6 Bit pro Zeichen

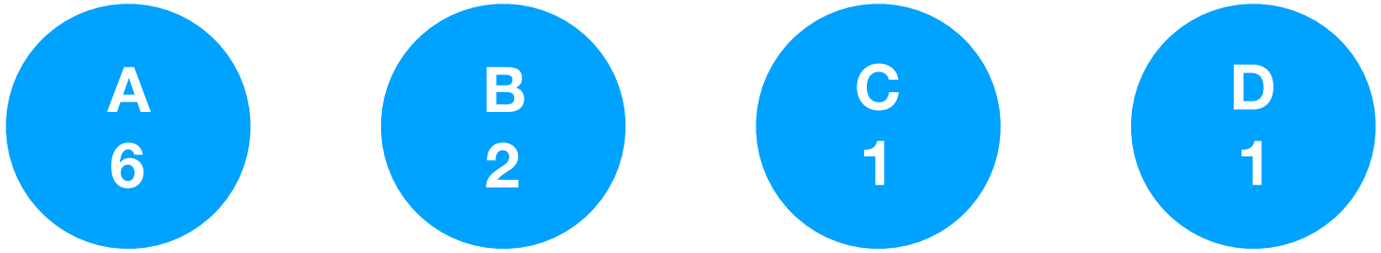
Huffman-Algorithmus

1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.

Wort: AAAAAABBCD

Huffman-Algorithmus

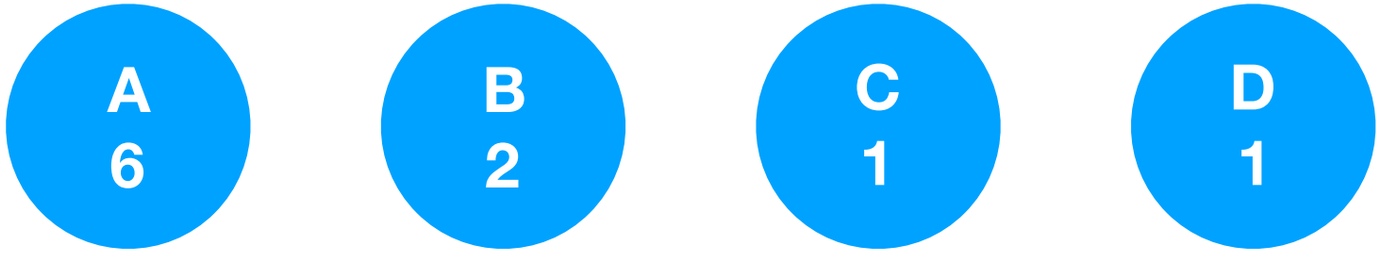
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.



Wort: AAAAAABBCD

Huffman-Algorithmus

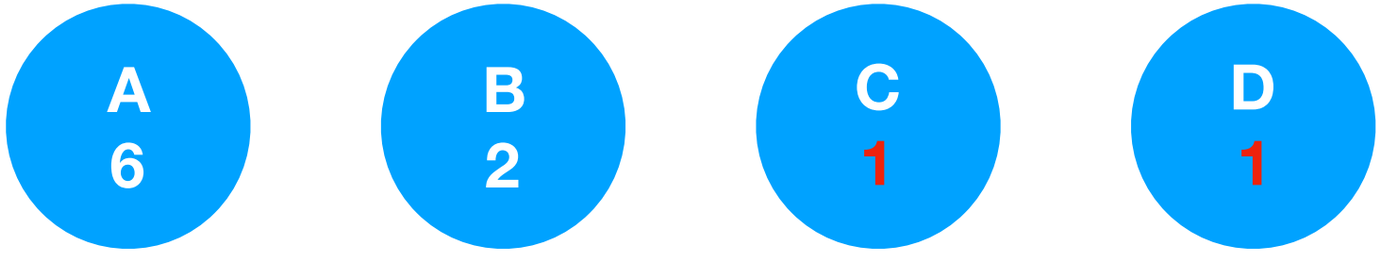
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.



Wort: AAAAAABBCD

Huffman-Algorithmus

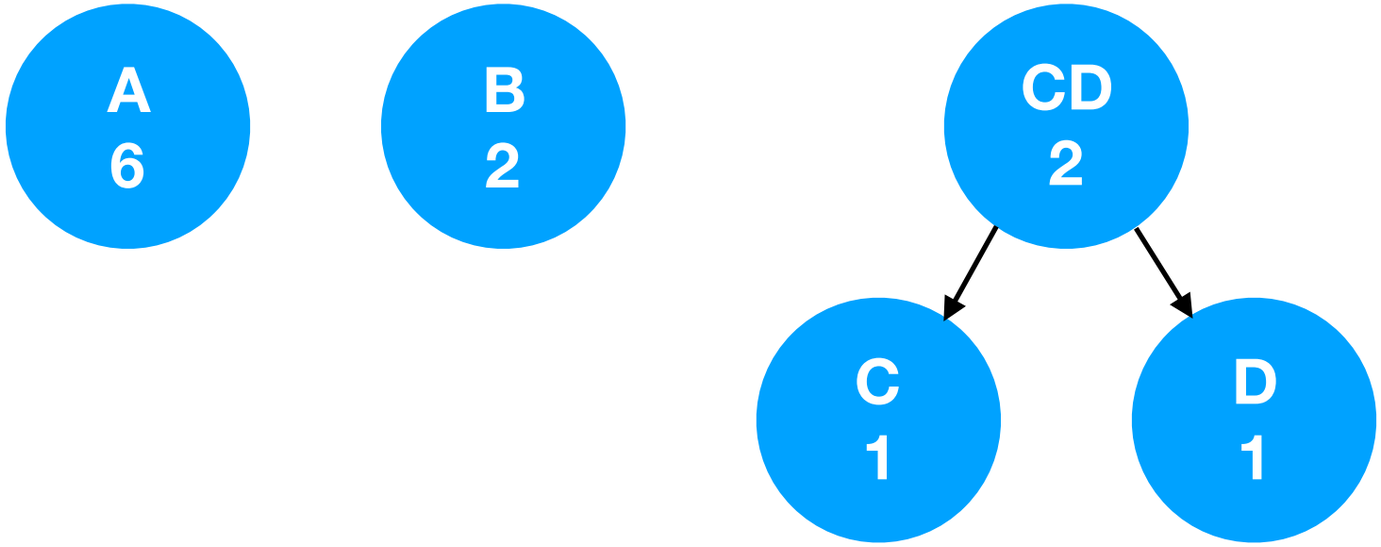
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.



Wort: AAAAAABBCD

Huffman-Algorithmus

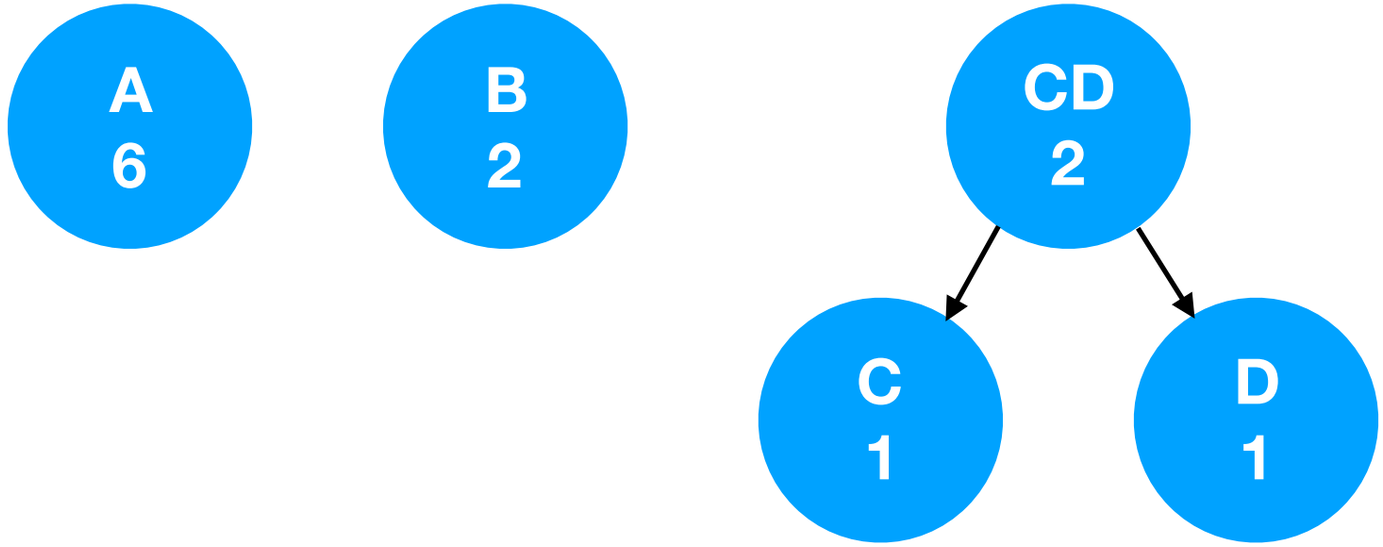
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.



Wort: AAAAAABBCD

Huffman-Algorithmus

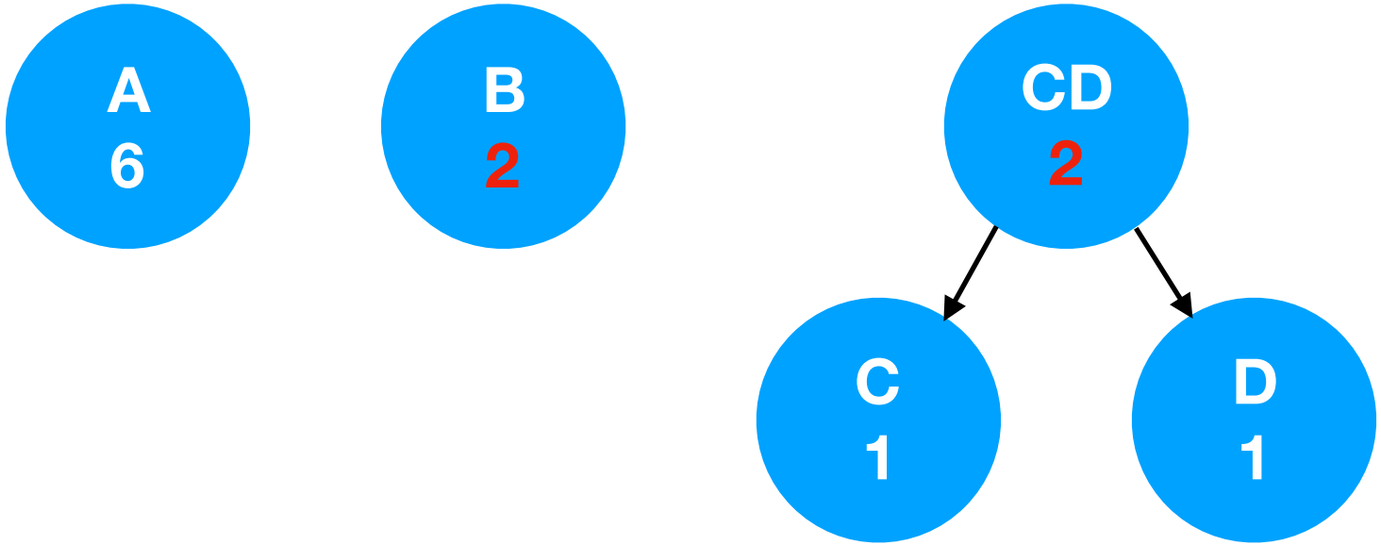
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.
3. Wiederholen, bis nur noch ein Baum übrig ist.



Wort: AAAAAABBCD

Huffman-Algorithmus

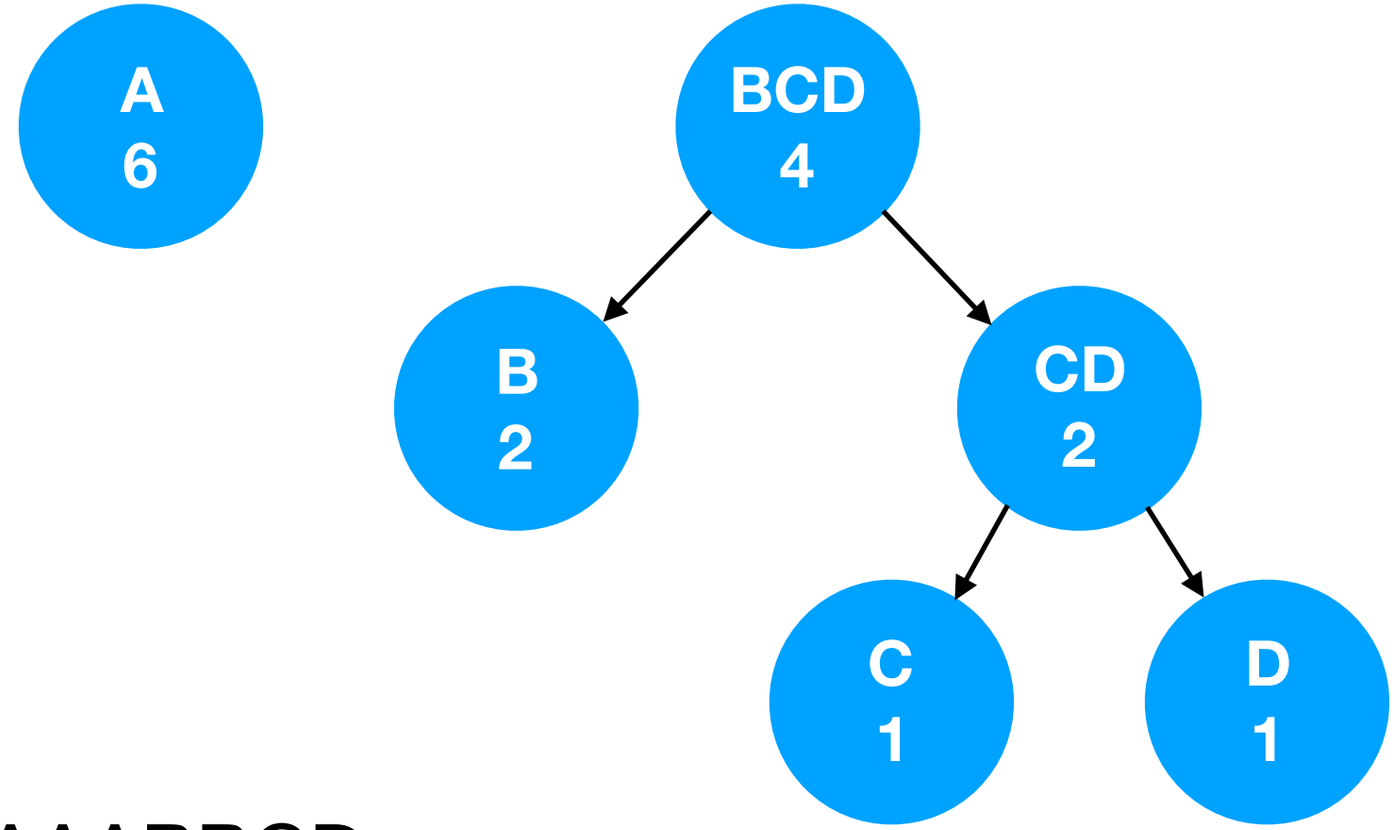
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.
3. Wiederholen, bis nur noch ein Baum übrig ist.



Wort: AAAAAABBCD

Huffman-Algorithmus

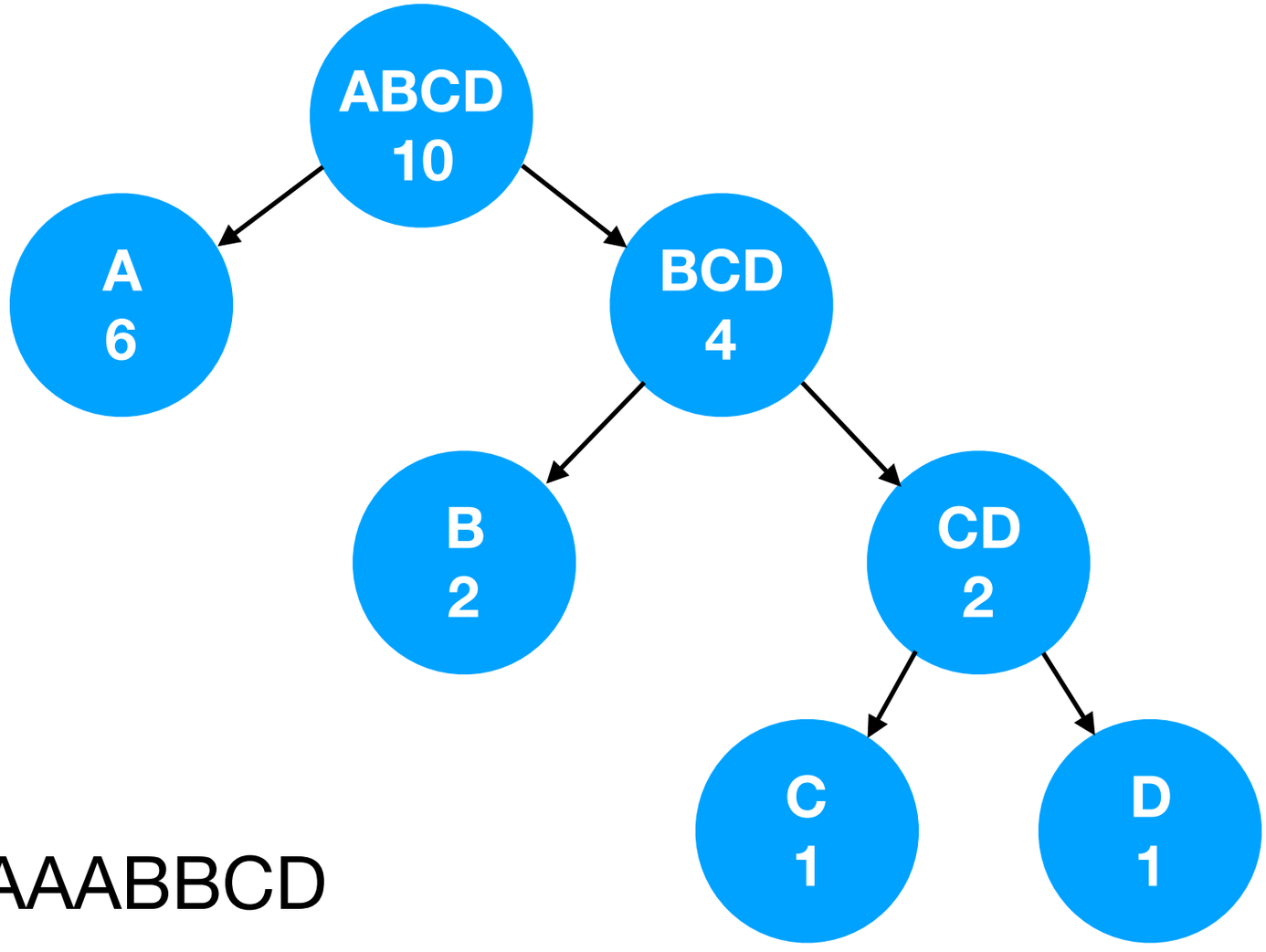
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.
3. Wiederholen, bis nur noch ein Baum übrig ist.



Wort: AAAAAABBCD

Huffman-Algorithmus

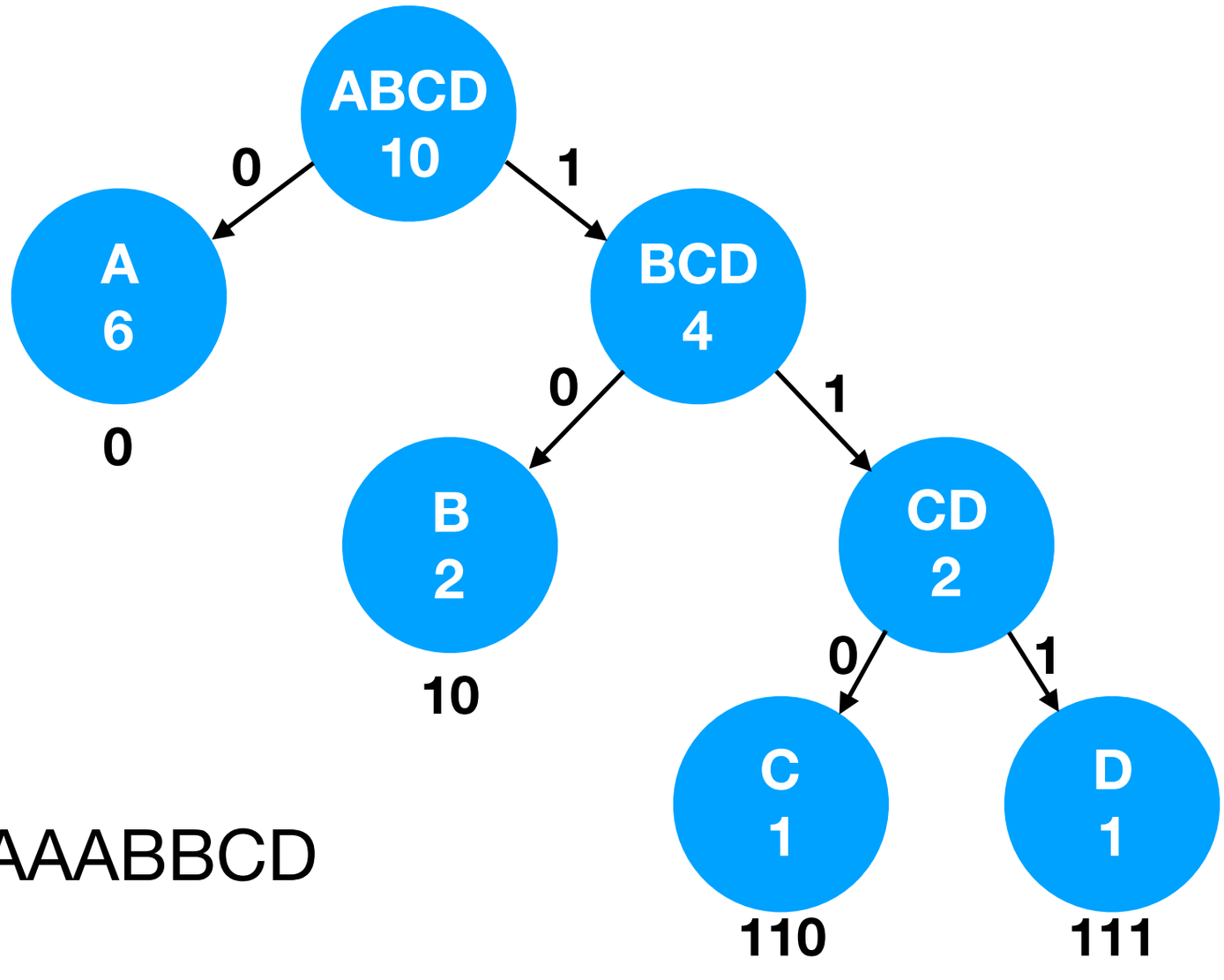
1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.
3. Wiederholen, bis nur noch ein Baum übrig ist.



Wort: AAAAAABBCD

Huffman-Algorithmus

1. Erstelle einen Wald mit Bäumen für jedes Zeichen. Jeder Baum enthält nur einen Knoten.
2. Kombiniere die beiden Bäume mit der geringsten Häufigkeit zu einen neuen Baum.
3. Wiederholen, bis nur noch ein Baum übrig ist.



Wort: AAAAAABBCD

Kompressionsalgorithmen

ZIP

- Dateiformat
- 1989 entwickelt
- Erlaubt die Benutzung von mehreren Kompressionsalgorithmen
- Davon ist Deflate der Gebräuchlichste

GZIP

- Software, die Deflate-Dateien erzeugt
- 1992 entwickelt
- Nutzt ähnliche Implementation des Deflate-Algorithmus wie ZIP

Zopfli

- Implementation des Deflate-Algorithmus
- 2012 vom schweizerischem Google-Team entwickelt
- Sehr hohe Kompressionsrate
- Kompression 80-mal langsamer als GZip
- Dekompression bleibt jedoch schnell
- Geeignet für Dateien, die einmalig komprimiert werden

Brotli

- Weitere Entwicklung des Google Teams in 2014
- Kodiert nicht ins Deflate-Format, sondern in ein Eigenes
- Nutzt jedoch ebenfalls LZ77 und Huffman
- Zusätzlich nutzt Brotli ein vordefiniertes Wörterbuch mit 13000 häufig auftretenden Strings

Brotli vs. Deflate

Algorithmus	Kompressionsrate	Kompressionsge-schwindigkeit (MB/s)	Dekompressionsge-schwindigkeit (MB/s)
Brotli 1	3.381	**98.3**	334.0
Brotli 9	3.965	17.0	**354.5**
Brotli 11	**4.347**	0.5	289.5
Deflate 1	2.913	93.5	323.0
Deflate 9	3.371	15.5	347.3

Table 5.3: Performance of Brotli and Deflate and bzip2 [AKSV15]

Gipfeli

- Noch eine Entwicklung des Google Teams
- Kodiert nicht ins Deflate-Format, sondern in ein Eigenes
- Nutzt LZ77, aber kein Huffman
- High-Speed-Kompression

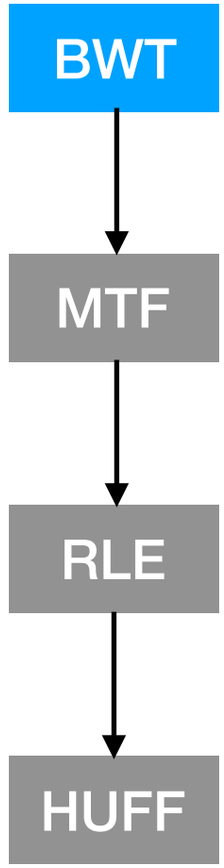
Kompression mit bzip2

bzip2

- Eigener Kompressionsalgorithmus, kein Deflate
- Kombination von verschiedenen Methoden:
  1. Burrows-Wheeler-Transformation
  2. Move-To-Front-Transformation
  3. Run-Length-Encoding
  4. Huffman-Codierung

Burrows-Wheeler-Transformation

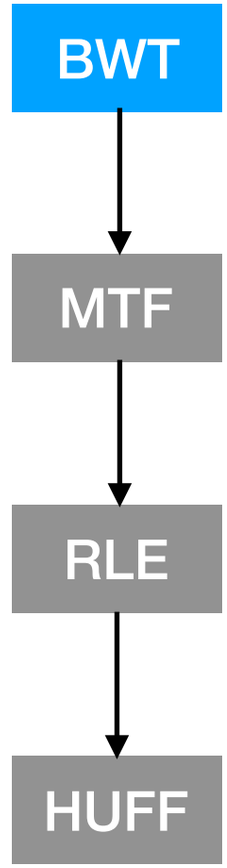
- Eingabe: Wortsequenz, z.B. ANANAS#
- Erzeugt eine Permutation der Eingabe
- Hohe Wahrscheinlichkeit, dass gleiche Zeichen nacheinander auftreten
- Kein Kompressionsalgorithmus!



Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

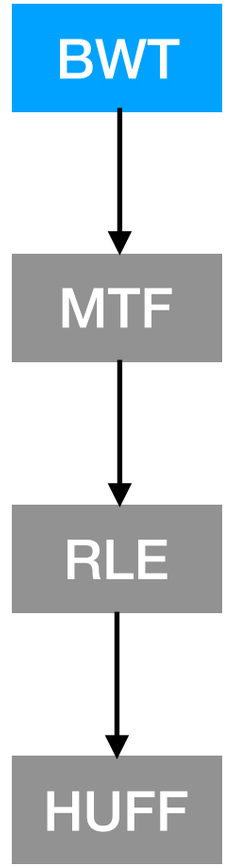
1. Rotieren	2. Sortieren	3. Ergebnis



Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

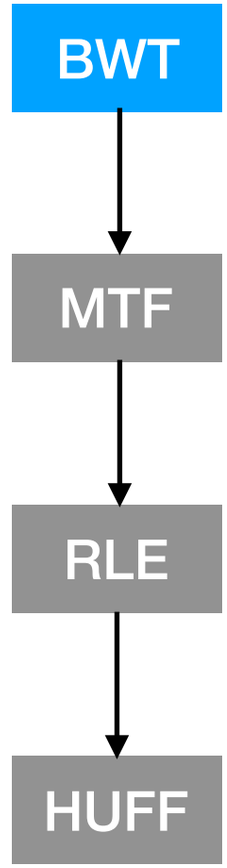
1. Rotieren	2. Sortieren	3. Ergebnis
ANANAS#		
NANAS#A		
ANAS#AN		
NAS#ANA		
AS#ANAN		
S#ANANA		
#ANANAS		



Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

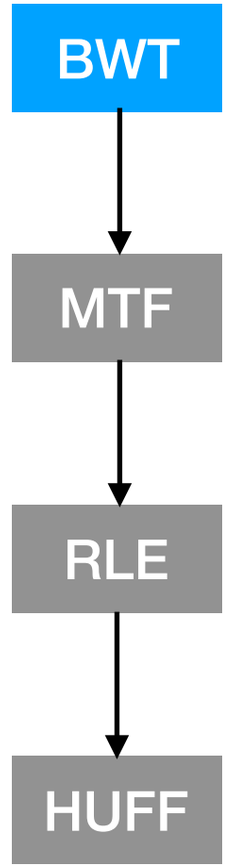
1. Rotieren	2. Sortieren	3. Ergebnis
ANANAS#	#ANANAS	
NANAS#A	ANANAS#	
ANAS#AN	ANAS#AN	
NAS#ANA	AS#ANAN	
AS#ANAN	NANAS#A	
S#ANANA	NAS#ANA	
#ANANAS	S#ANANA	



Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

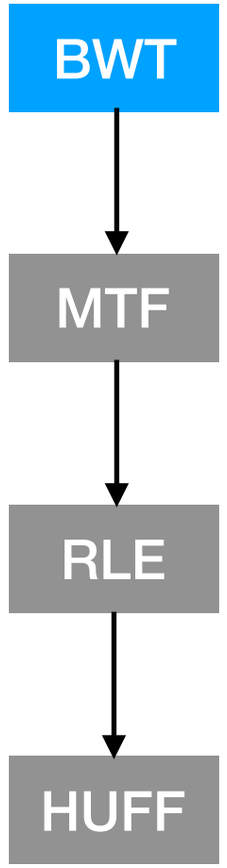
1. Rotieren	2. Sortieren	3. Ergebnis
ANANAS#	#ANANAS	
NANAS#A	ANANAS#	
ANAS#AN	ANAS#AN	
NAS#ANA	AS#ANAN	
AS#ANAN	NANAS#A	
S#ANANA	NAS#ANA	
#ANANAS	S#ANANA	



Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

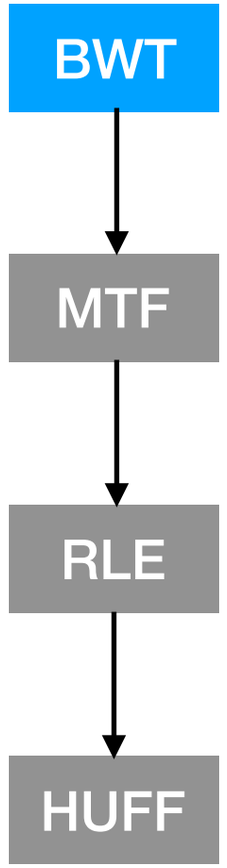
1. Rotieren	2. Sortieren	3. Ergebnis
ANANAS#	#ANANAS	S
NANAS#A	ANANAS#	#
ANAS#AN	ANAS#AN	N
NAS#ANA	AS#ANAN	N
AS#ANAN	NANAS#A	A
S#ANANA	NAS#ANA	A
#ANANAS	S#ANANA	A



Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

1. Rotieren	2. Sortieren	3. Ergebnis
ANANAS#	#ANANAS	S
NANAS#A	ANANAS#	#
ANAS#AN	ANAS#AN	N
NAS#ANA	AS#ANAN	N
AS#ANAN	NANAS#A	A
S#ANANA	NAS#ANA	A
#ANANAS	S#ANANA	A

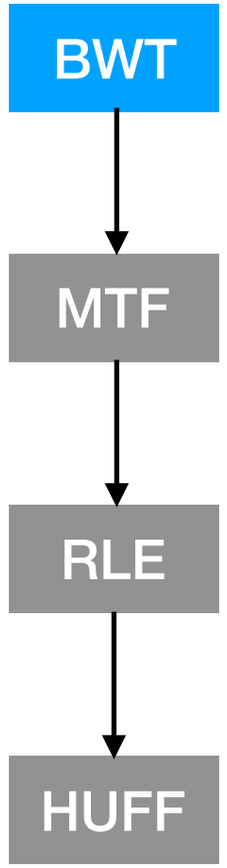


- Ausgabe: S#NNAAA

Burrows-Wheeler-Transformation

- Eingabe: ANANAS#

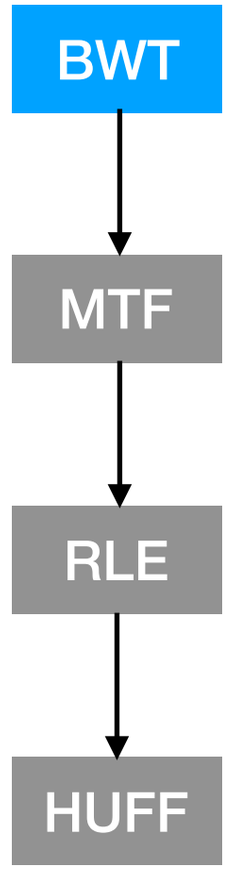
1. Rotieren	2. Sortieren	3. Ergebnis
ANANAS#	#ANANAS	S
NANAS#A	ANANAS#	#
ANAS#AN	ANAS#AN	N
NAS#ANA	AS#ANAN	N
AS#ANAN	NANAS#A	A
S#ANANA	NAS#ANA	A
#ANANAS	S#ANANA	A



- Ausgabe: S#NNAAA, 1

Burrows-Wheeler-Transformation

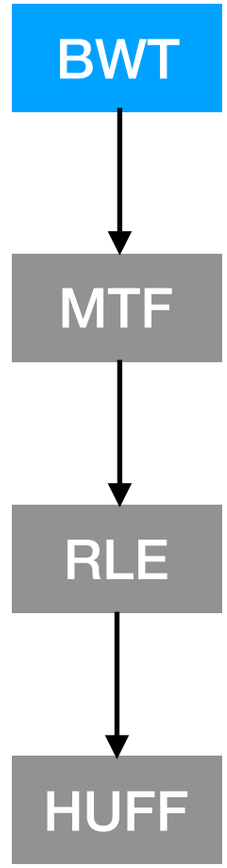
- Bei längeren Wörtern ist der Effekt deutlicher:
- Eingabe: ANANAS#ANANAS#ANANAS#
- Ausgabe: SSS###NNNNNNAAAAAAAAA, 3



BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

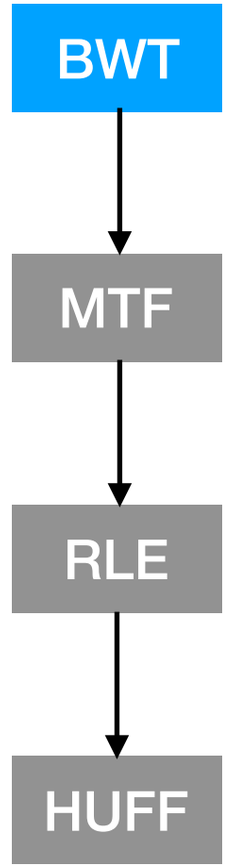
1. Rotieren	2. Sortieren	3. Ergebnis
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?



BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

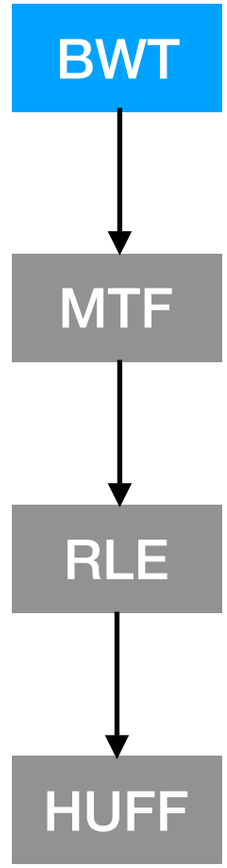
1. Rotieren	2. Sortieren	3. Ergebnis
?	?	S
?	?	#
?	?	N
?	?	N
?	?	A
?	?	A
?	?	A



BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

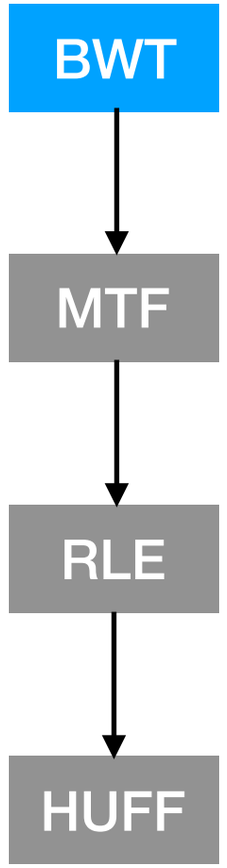
1. Rotieren	2. Sortieren	3. Ergebnis
?	??????S	S
?	??????#	#
?	??????N	N
?	??????N	N
?	??????A	A
?	??????A	A
?	??????A	A



BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

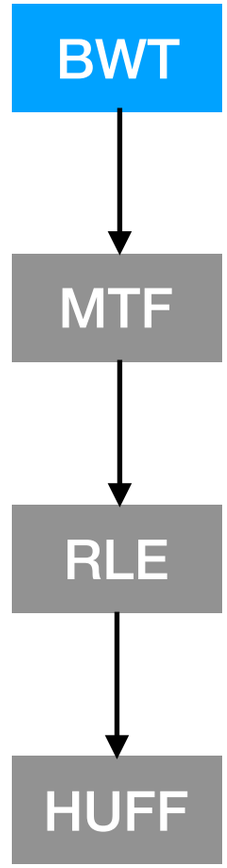
1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A



BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

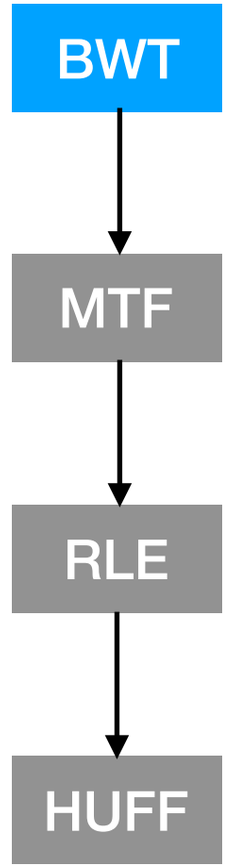
1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A



BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A

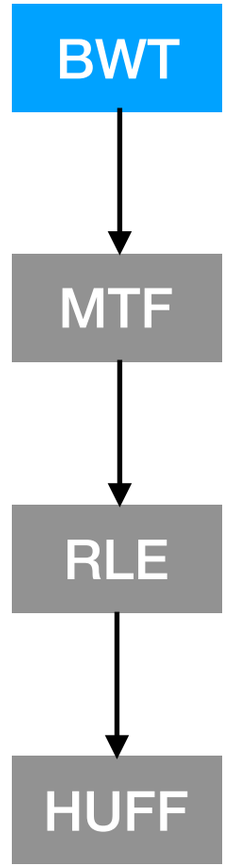


- Ausgabe: A

BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A

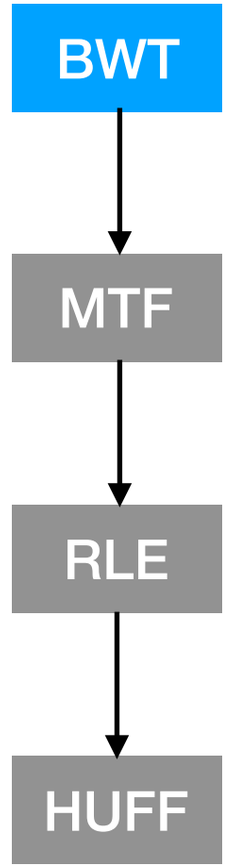


- Ausgabe: A

BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A

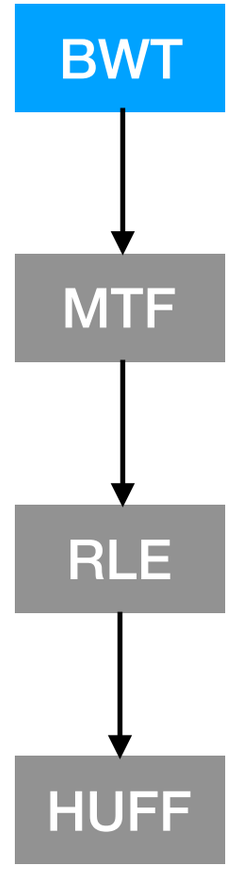


- Ausgabe: AN

BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A

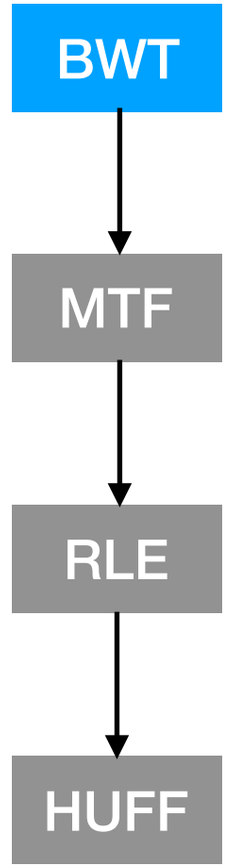


- Ausgabe: AN

BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A

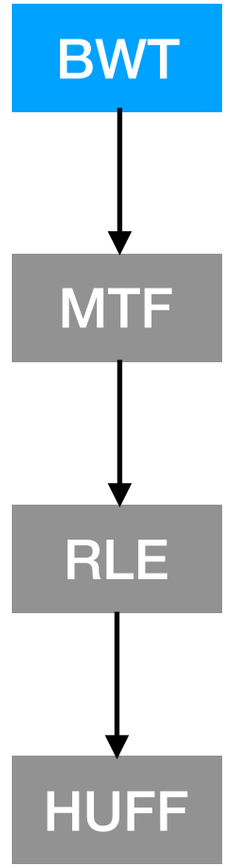


- Ausgabe: ANA

BWT-Rücktransformation

- Eingabe: S#NNAAA, 1

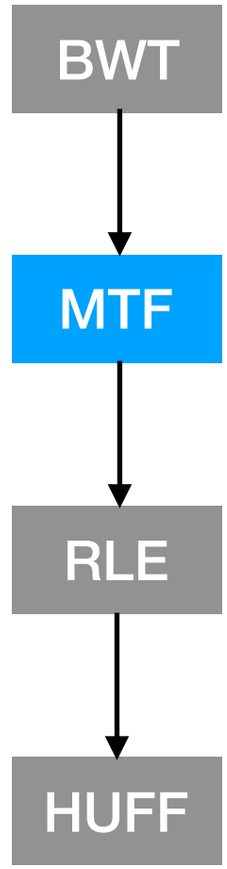
1. Rotieren	2. Sortieren	3. Ergebnis
?	#?????S	S
?	A?????#	#
?	A?????N	N
?	A?????N	N
?	N?????A	A
?	N?????A	A
?	S?????A	A



- Ausgabe: ANANAS#

Move-To-Front-Transformation

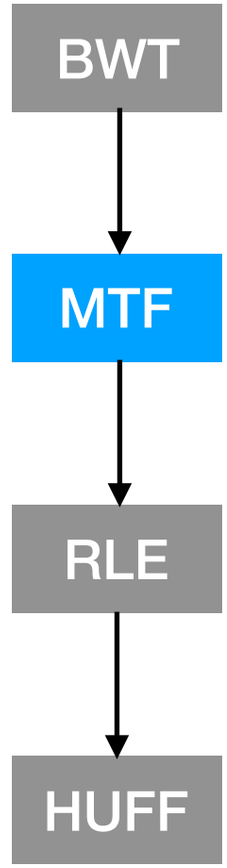
- Eingabe:
  - Wortsequenz
  - Alphabet dieser Wortsequenz
- Erzeugt eine Permutation der Eingabe
- Nacheinander auftretende Zeichen in der Eingabe resultieren in vielen 0en in der Ausgabe
- Auch hierbei handelt es sich noch nicht um eine Kompression!



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

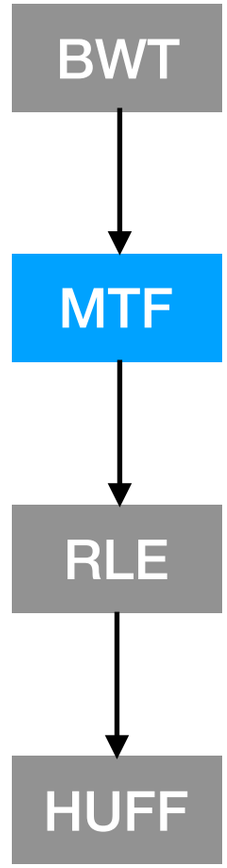
Input	Alphabet	Output	Alphabet’



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

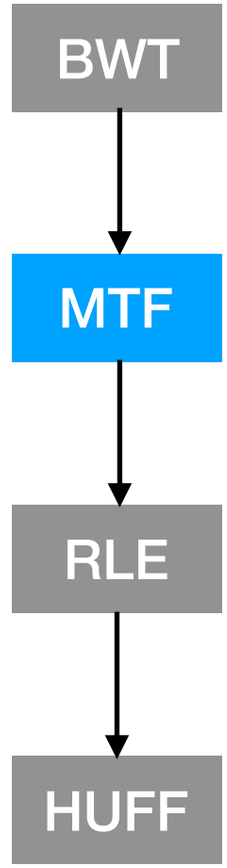
Input	Alphabet	Output	Alphabet’
S			
#			
N			
N			
A			
A			
A			



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

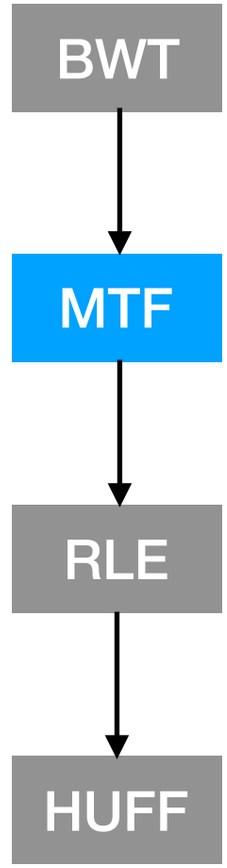
Input	Alphabet	Output	Alphabet’
S	#ANS		
#			
N			
N			
A			
A			
A			



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

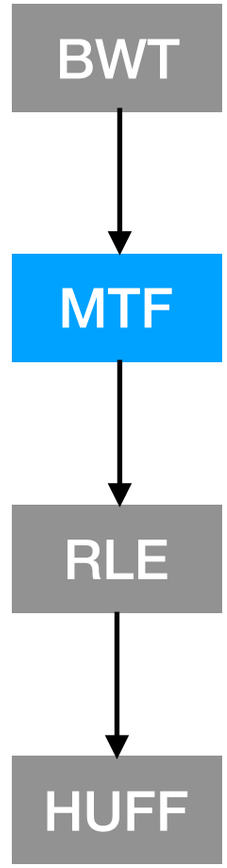
Input	Alphabet	Output	Alphabet’
S	#ANS	3	
#			
N			
N			
A			
A			
A			



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

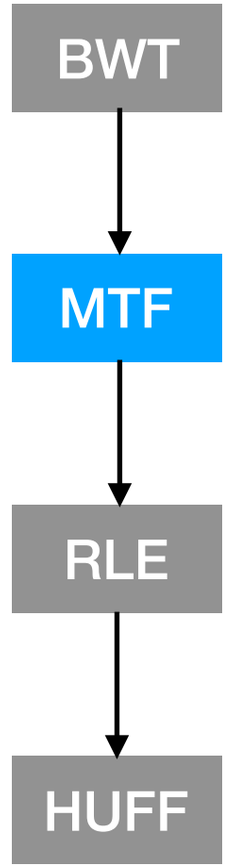
Input	Alphabet	Output	Alphabet’
S	#ANS	3	S#AN
#			
N			
N			
A			
A			
A			



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

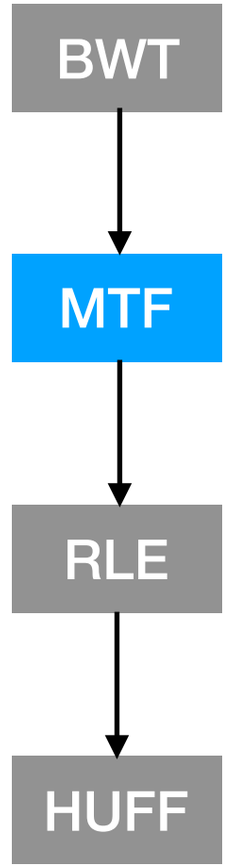
Input	Alphabet	Output	Alphabet’
S	#ANS	3	S#AN
#	S#AN		
N			
N			
A			
A			
A			



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

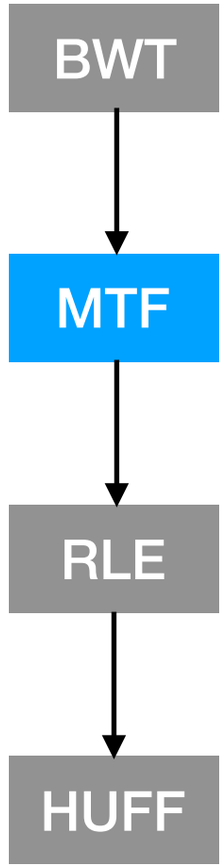
Input	Alphabet	Output	Alphabet’
S	#ANS	3	S#AN
#	S#AN	1	#SAN
N	#SAN		
N			
A			
A			
A			



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

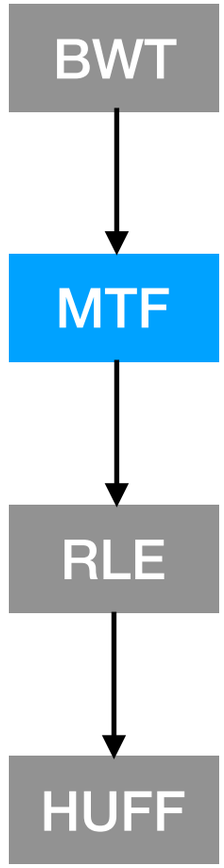
Input	Alphabet	Output	Alphabet’
S	#ANS	3	S#AN
#	S#AN	1	#SAN
N	#SAN	3	N#SA
N	N#SA	0	N#SA
A	N#SA	3	AN#S
A	AN#S	0	AN#S
A	AN#S	0	AN#S



Move-To-Front-Transformation

- Eingabe: S#NNAAA, #ANS

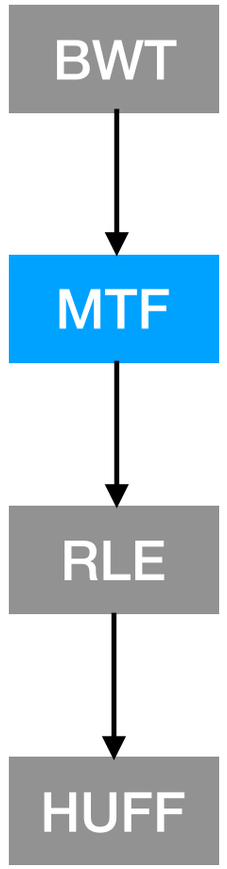
Input	Alphabet	Output	Alphabet’
S	#ANS	3	S#AN
#	S#AN	1	#SAN
N	#SAN	3	N#SA
N	N#SA	0	N#SA
A	N#SA	3	AN#S
A	AN#S	0	AN#S
A	AN#S	0	AN#S



- Ausgabe: 3130300, AN#S

Move-To-Front-Transformation

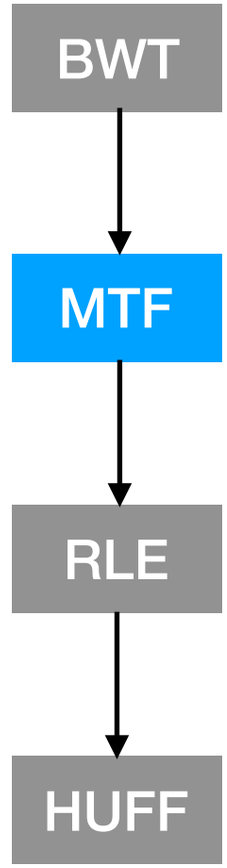
- Auch hier ist bei längeren Wörtern der Effekt deutlicher:
- Eingabe: SSS###NNNNNNAAAAAAAAA
- Ausgabe: 300100300000300000000, AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

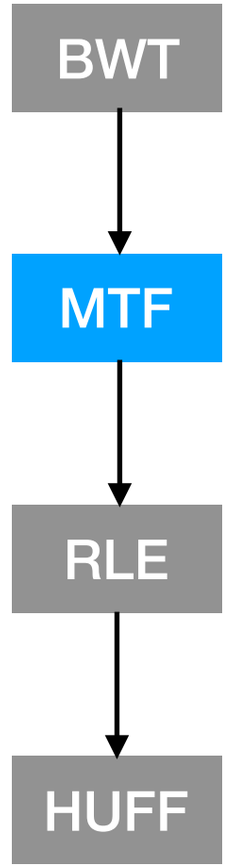
Output	Alphabet	Input	Alphabet’



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

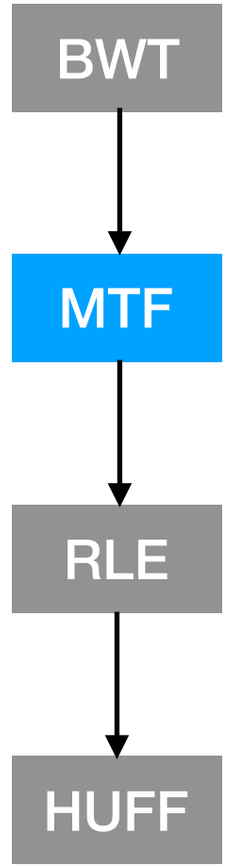
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	
		3	
		0	
		0	



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

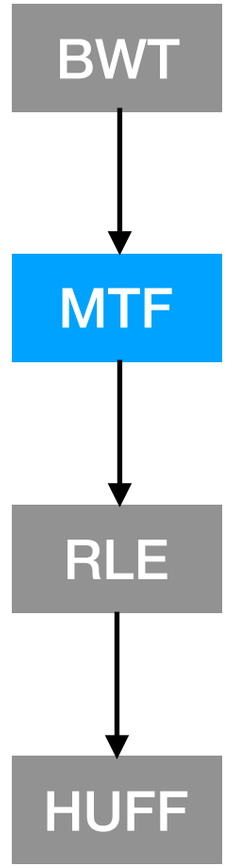
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	
		3	
		0	
		0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

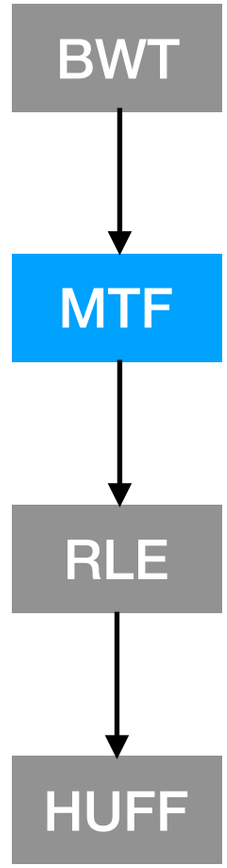
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	
		3	
		0	
A		0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

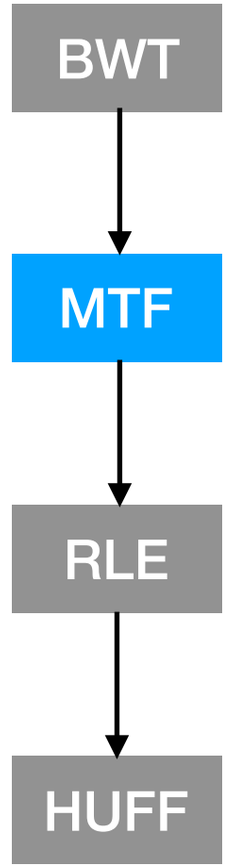
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	
		3	
		0	
A	AN#S	0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

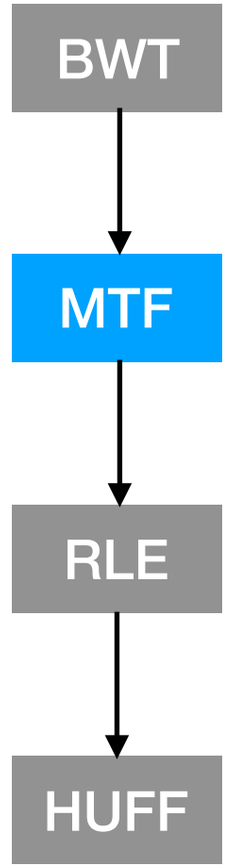
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	
		3	
		0	AN#S
A	AN#S	0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

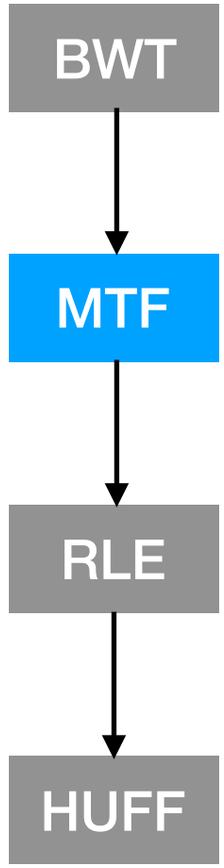
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	
		3	AN#S
A	AN#S	0	AN#S
A	AN#S	0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

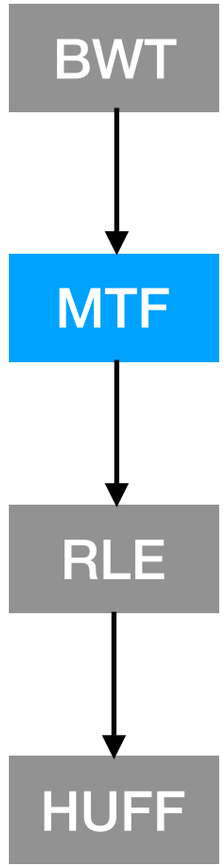
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	
		0	N#SA
A	N#SA	3	AN#S
A	AN#S	0	AN#S
A	AN#S	0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

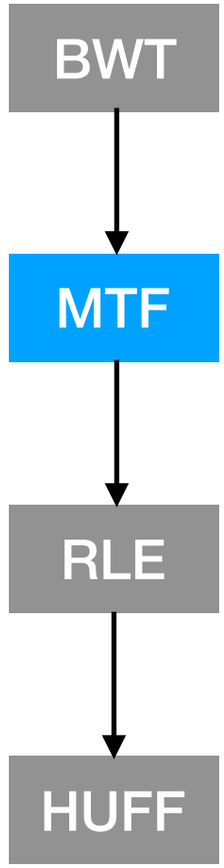
Output	Alphabet	Input	Alphabet’
		3	
		1	
		3	N#SA
N	N#SA	0	N#SA
A	N#SA	3	AN#S
A	AN#S	0	AN#S
A	AN#S	0	AN#S



MTF-Rücktransformation

- Eingabe: 3130300, AN#S

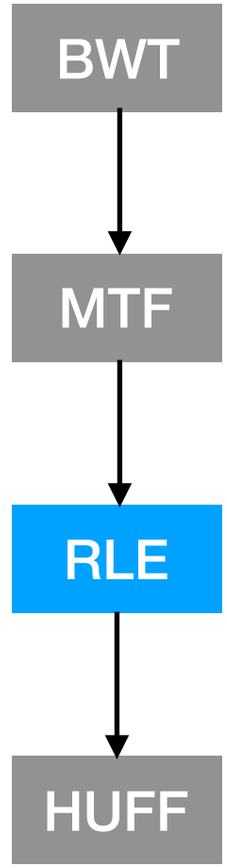
Output	Alphabet	Input	Alphabet’
S	#ANS	3	S#AN
#	S#AN	1	#SAN
N	#SAN	3	N#SA
N	N#SA	0	N#SA
A	N#SA	3	AN#S
A	AN#S	0	AN#S
A	AN#S	0	AN#S



- Ausgabe: S#NNAAA

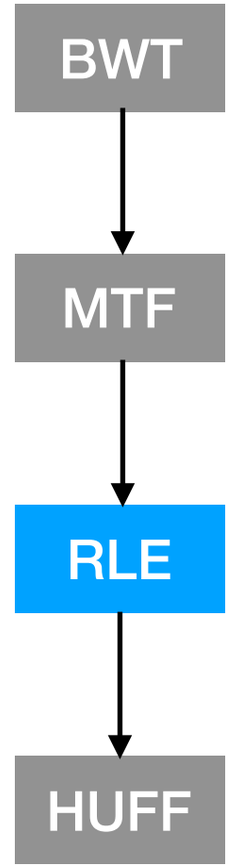
Run-Length-Encoding

- Gleiche aufeinanderfolgende Zeichen werden abgekürzt:
- z.B. AAAAAABBCD -> A6B2C1D1
- Deduplikation



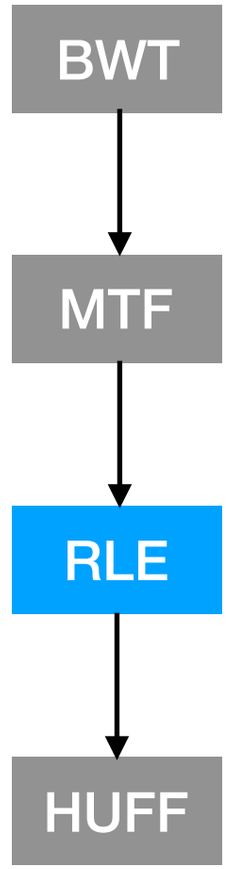
Run-Length-Encoding

- Eingabe: 300100300000300000000



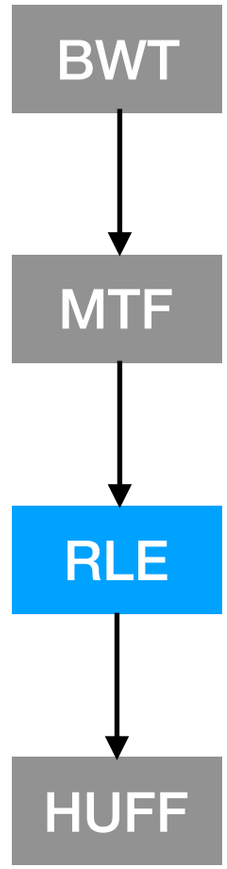
Run-Length-Encoding

- Eingabe: 3 00 1 00 3 00000 3 00000000



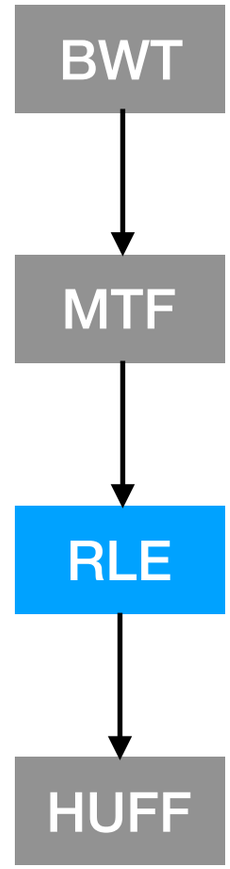
Run-Length-Encoding

- Eingabe: 3 00 1 00 3 00000 3 00000000
- Ausgabe: 31 02 11 02 31 05 31 08



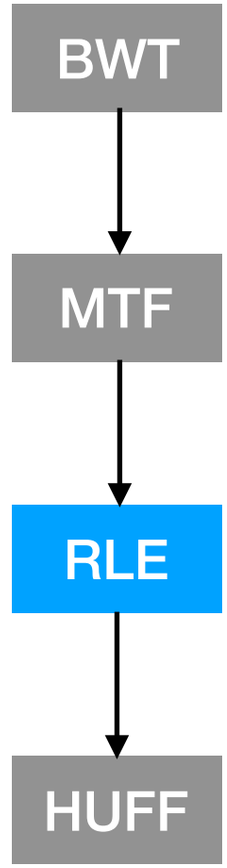
Run-Length-Encoding

- Eingabe: 3 00 1 00 3 00000 3 00000000
- Ausgabe: 3102110231053108



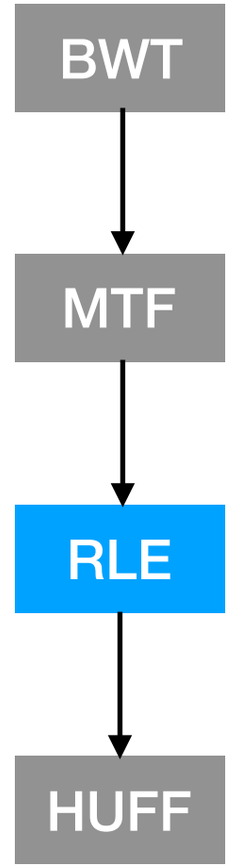
RLE-Dekodierung

- Eingabe: 31021102310531O8



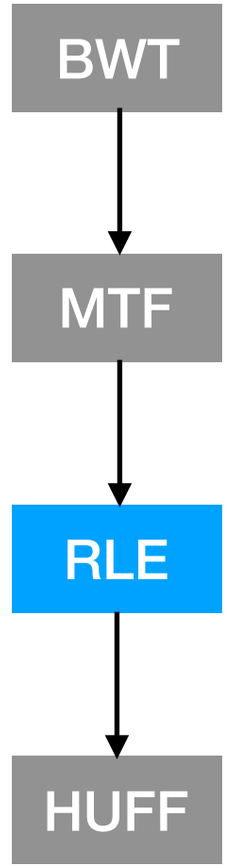
RLE-Dekodierung

- Eingabe: 31 02 11 02 31 05 31 08



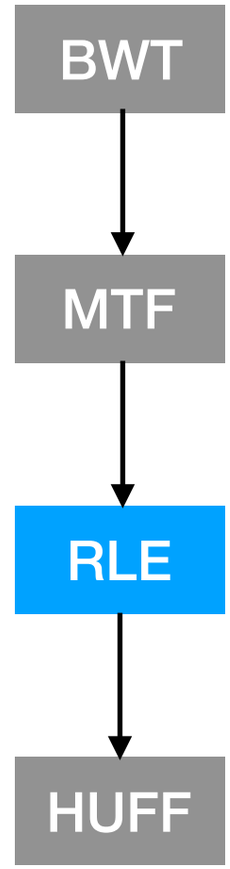
RLE-Dekodierung

- Eingabe: 31 02 11 02 31 05 31 08
- Ausgabe: 3 00 1 00 3 00000 3 00000000



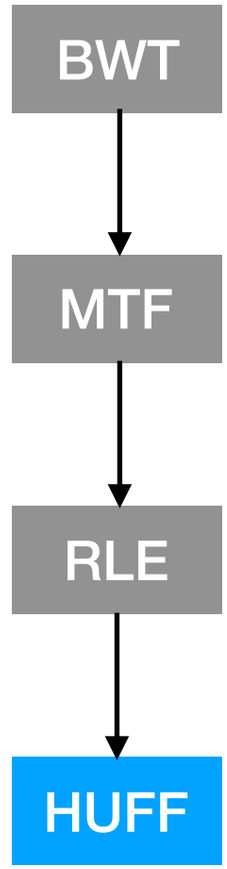
RLE-Dekodierung

- Eingabe: 31 02 11 02 31 05 31 08
- Ausgabe: 300100300000300000000



Huffman-Kodierung

- Eingabe: 31021102310531O8



Huffman-Kodierung

- Eingabe: 31021102310531O8

0 : 4 mal

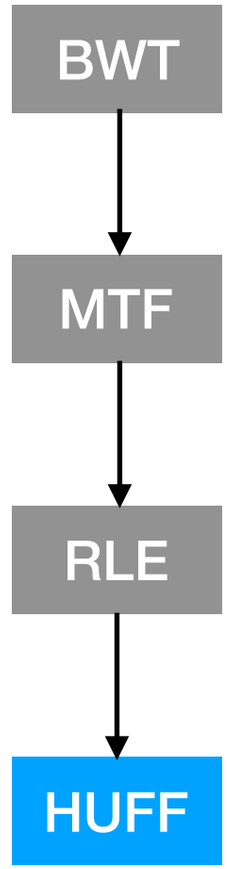
1 : 5 mal

2 : 2 mal

3 : 3 mal

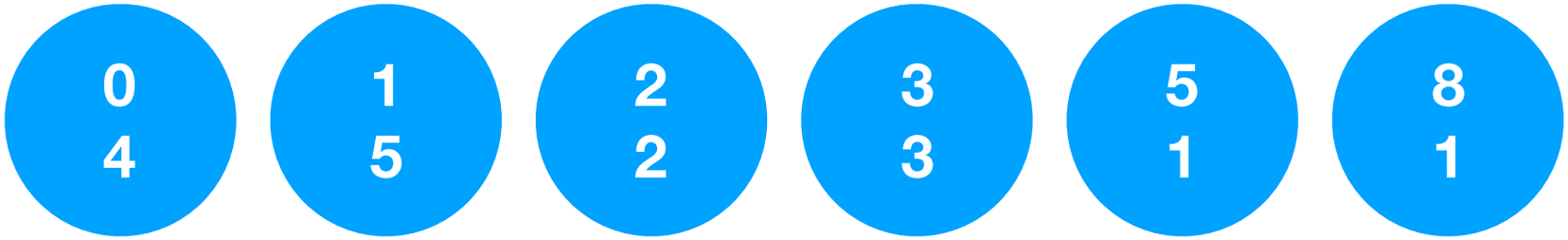
5 : 1 mal

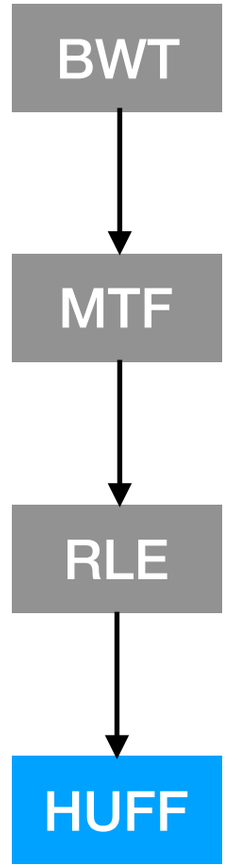
8 : 1 mal



Huffman-Kodierung

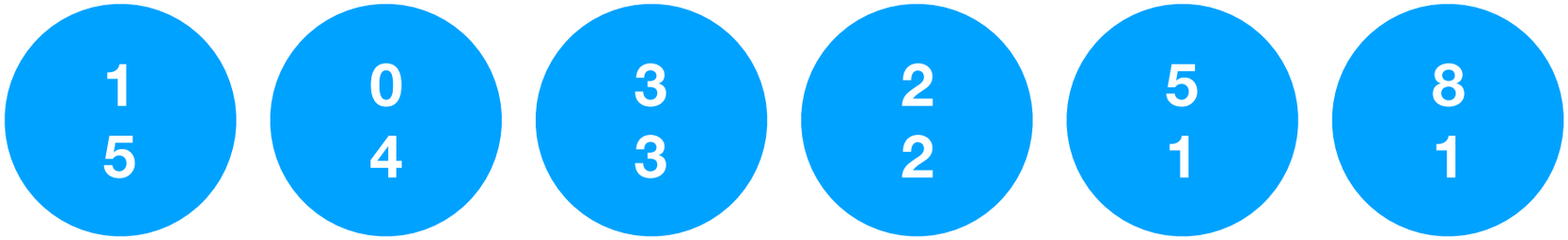
- Eingabe: 31021102310531O8

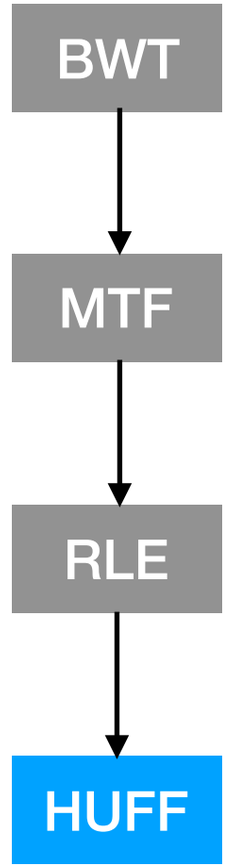




Huffman-Kodierung

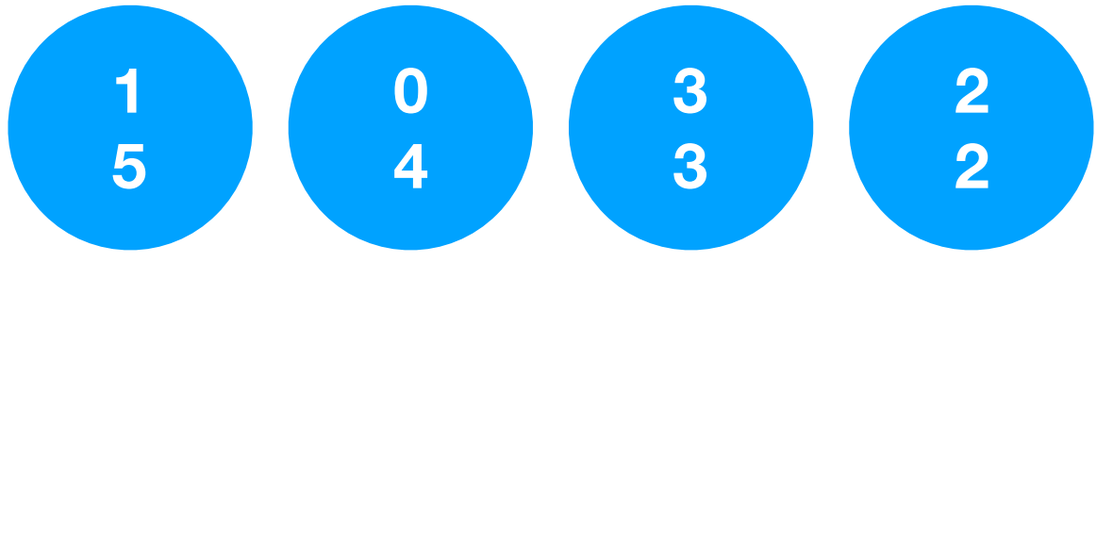
- Eingabe: 31021102310531O8

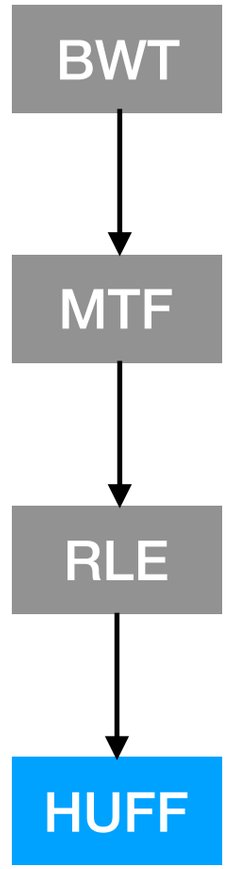




Huffman-Kodierung

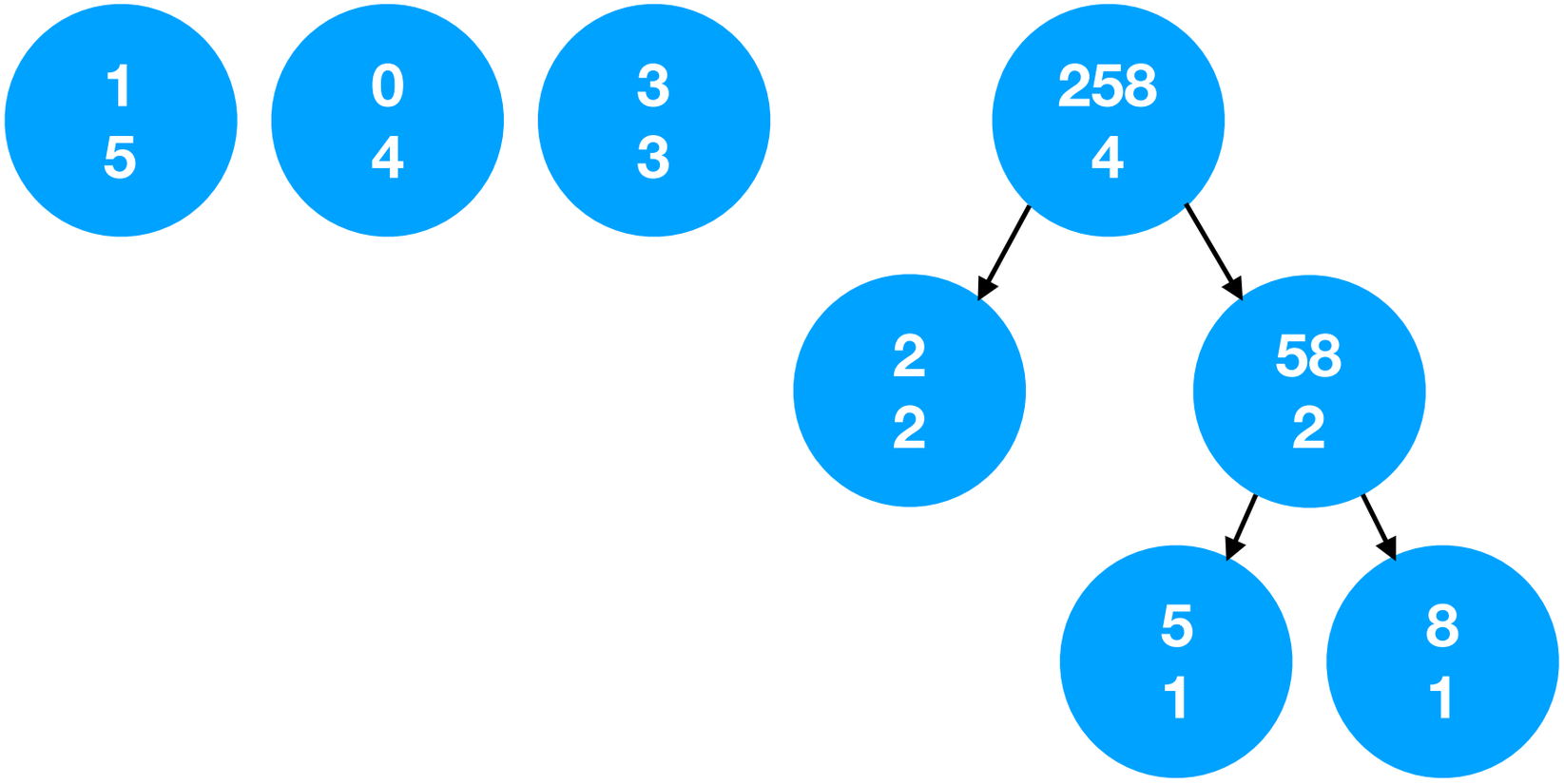
- Eingabe: 31021102310531O8

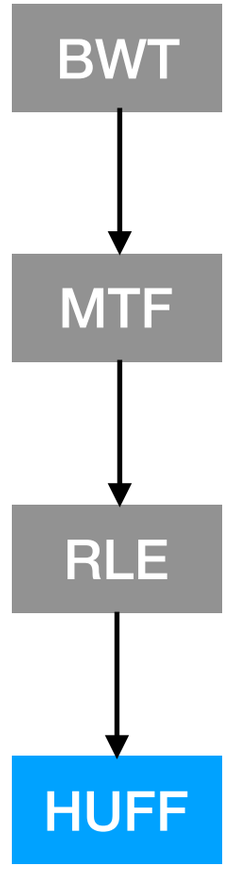




Huffman-Kodierung

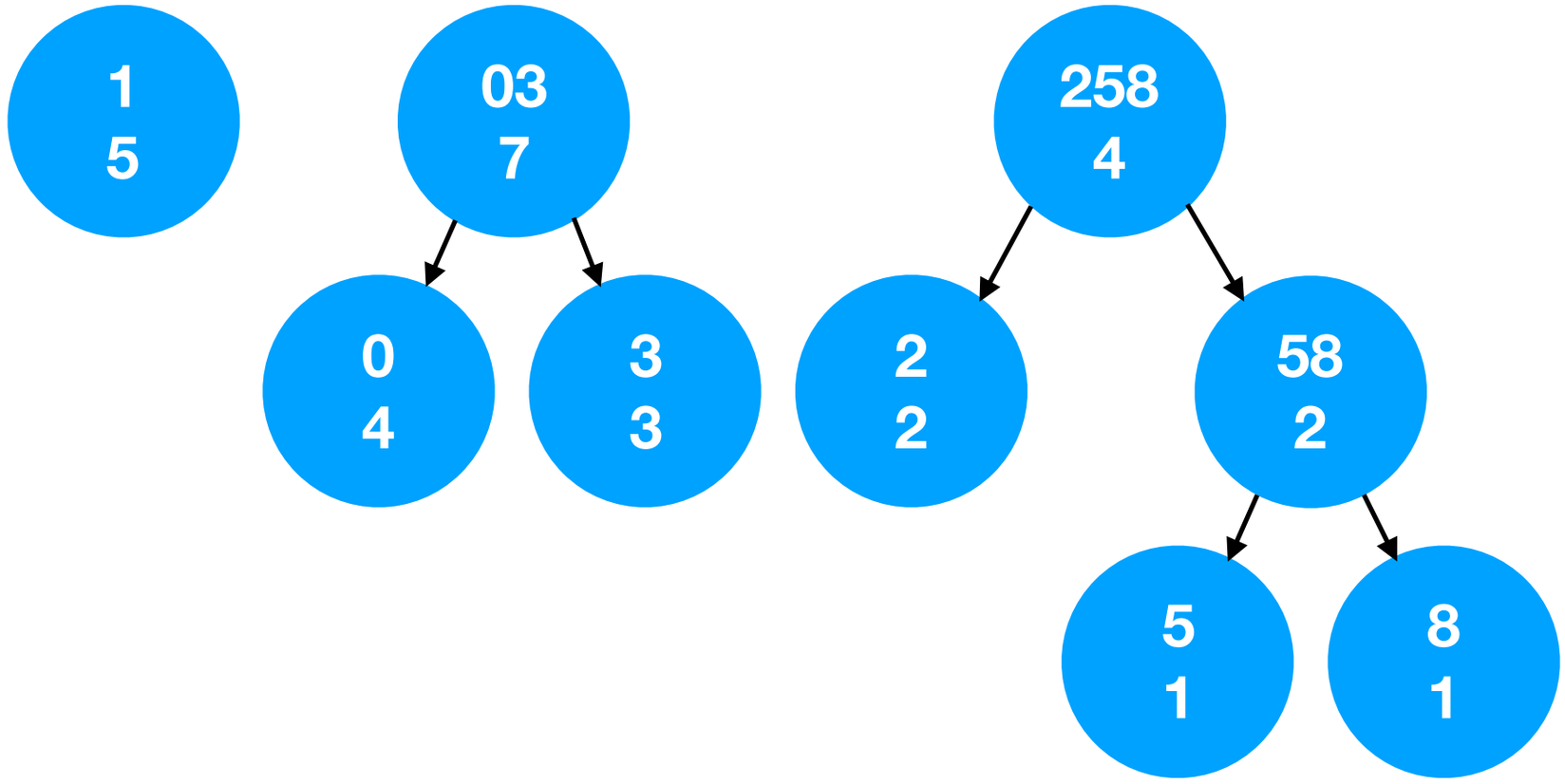
- Eingabe: 31021102310531O8

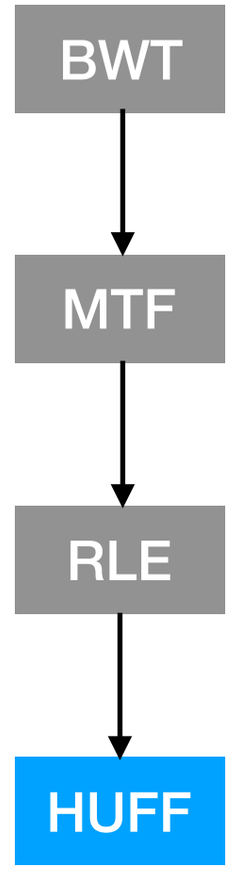




Huffman-Kodierung

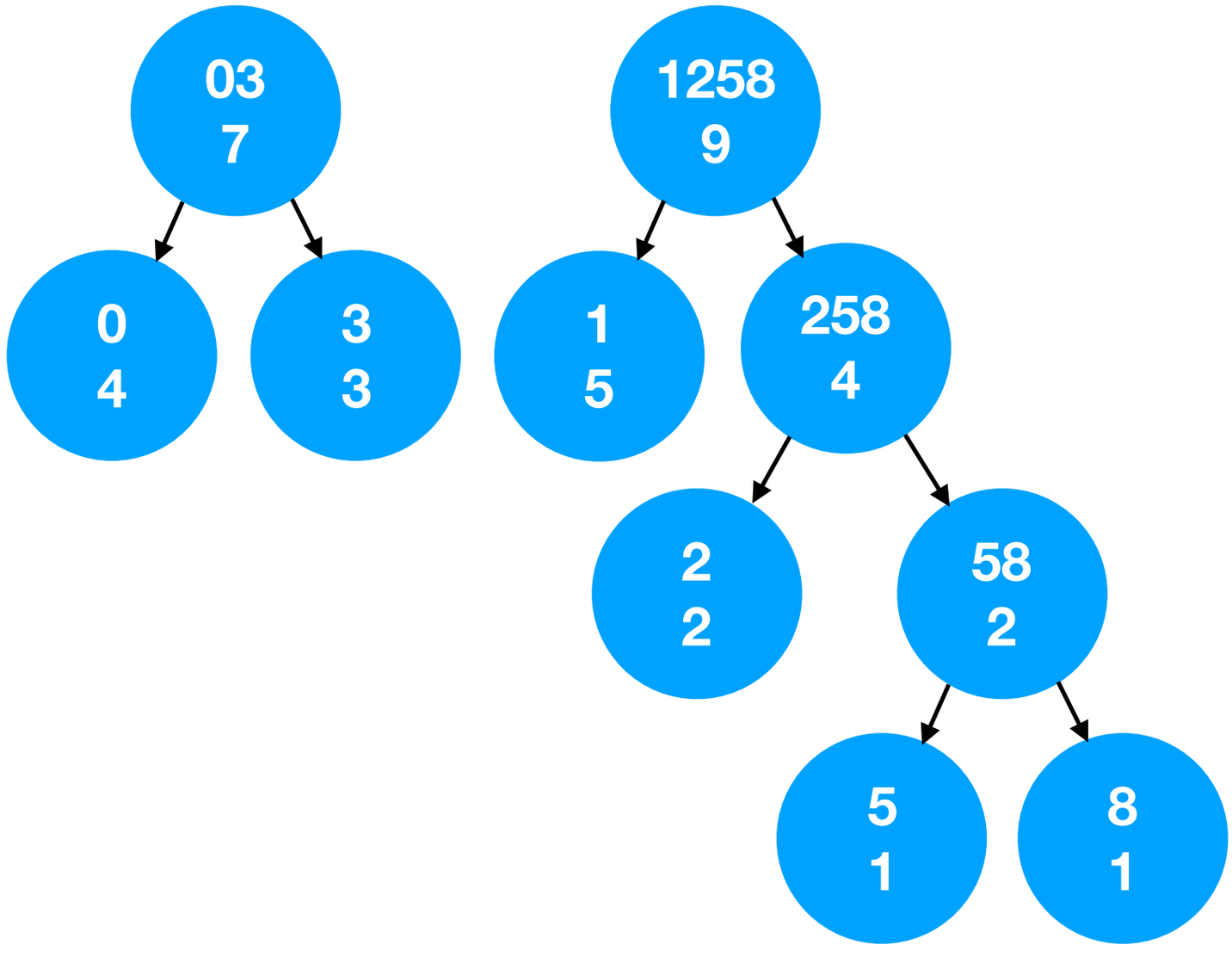
- Eingabe: 31021102310531O8

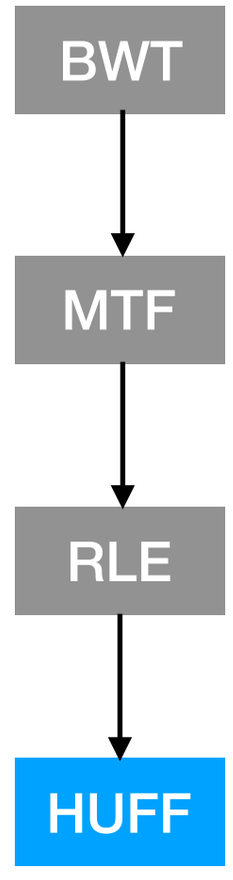




Huffman-Kodierung

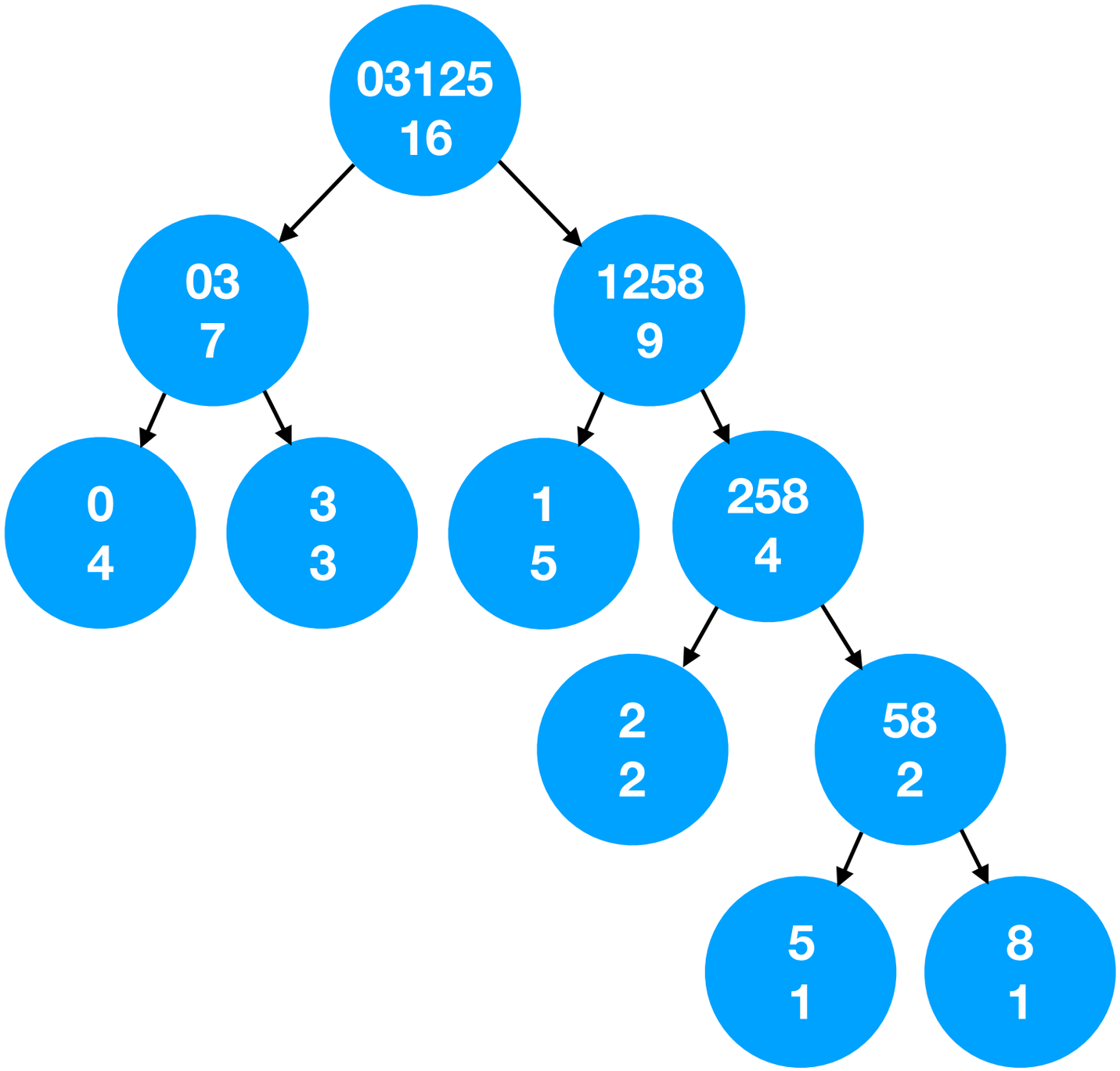
- Eingabe: 31021102310531O8

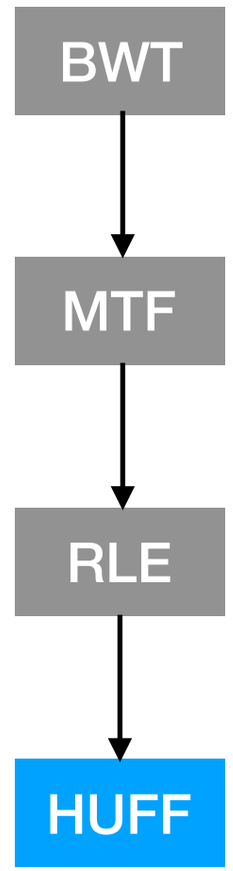




Huffman-Kodierung

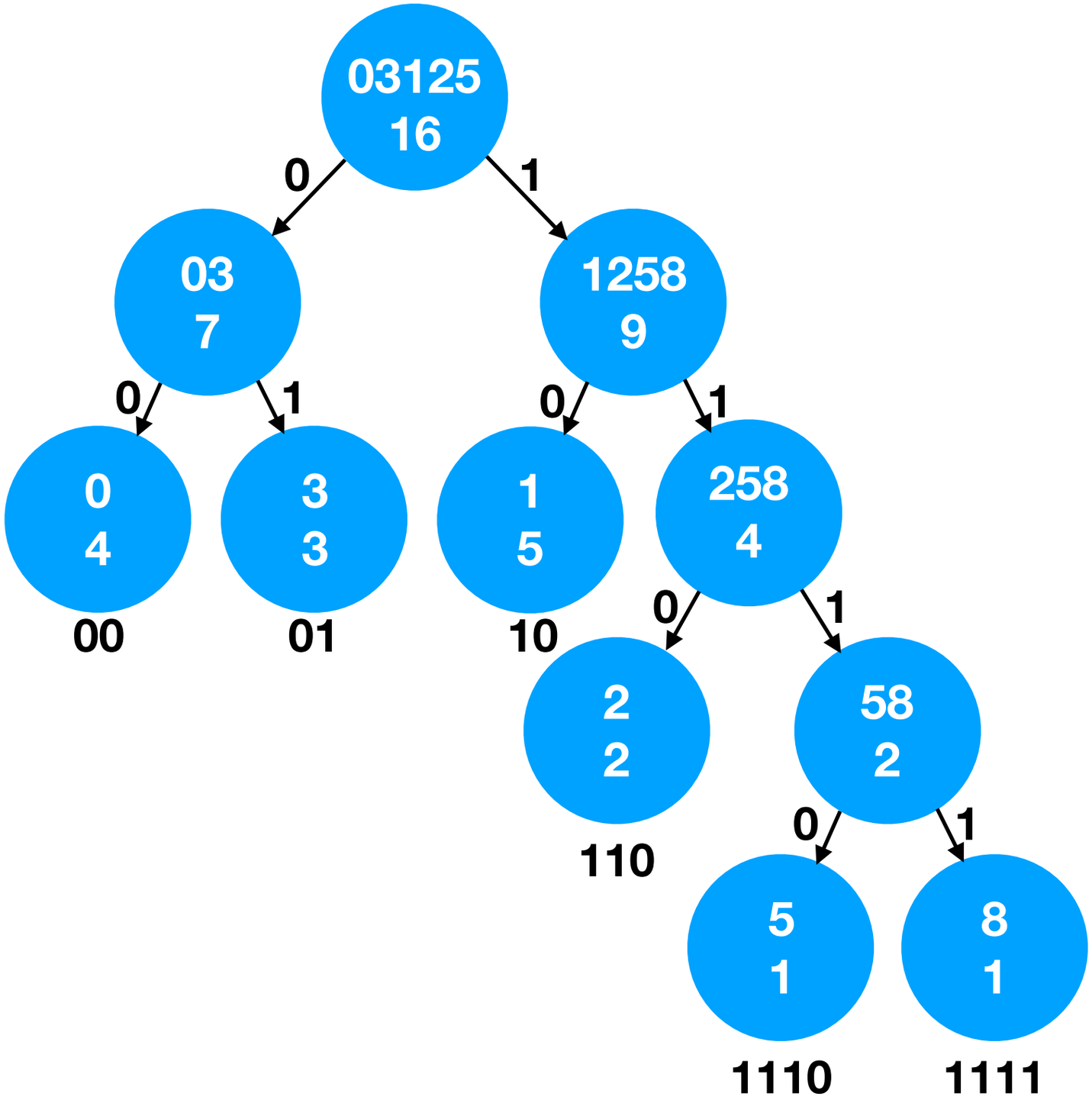
- Eingabe: 31021102310531O8

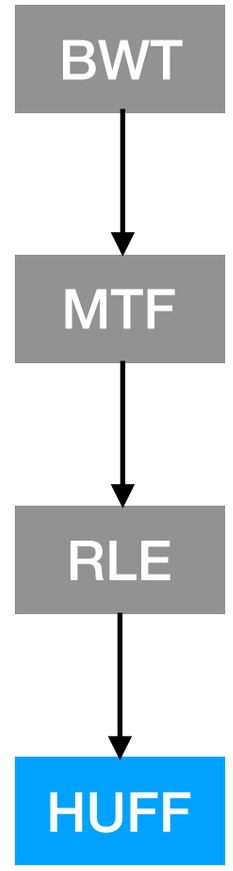




Huffman-Kodierung

- Eingabe: 31021102310531O8





Huffman-Kodierung

- Eingabe: 31021102310531O8

0 = 00

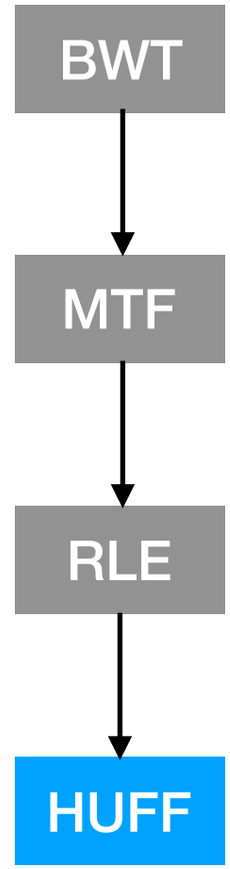
1 = 10

2 = 110

3 = 01

5 = 1110

8 = 1111



Huffman-Kodierung

- Eingabe: **3**1021102310531O8

0 = 00

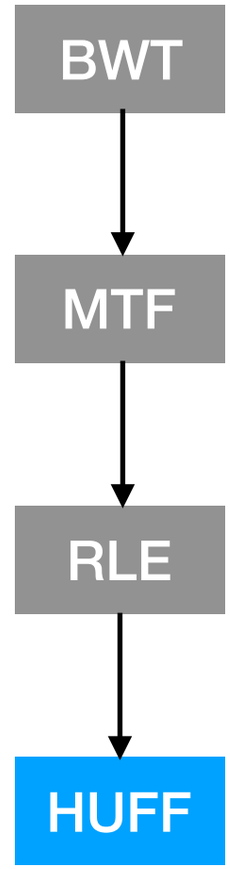
1 = 10

2 = 110

3 = 01

5 = 1110

8 = 1111



Huffman-Kodierung

- Eingabe: **3**1021102310531O8

0 = 00

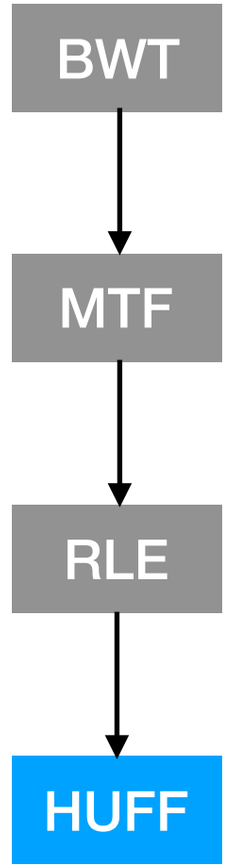
1 = 10

2 = 110

3 = 01

5 = 1110

8 = 1111



Huffman-Kodierung

- Eingabe: **3**1021102310531O8

0 = 00

1 = 10

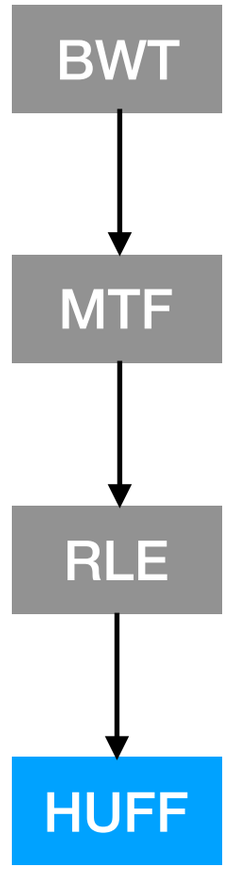
2 = 110

3 = 01

5 = 1110

8 = 1111

- Ausgabe: 01



Huffman-Kodierung

- Eingabe: 31021102310531O8

0 = 00

1 = 10

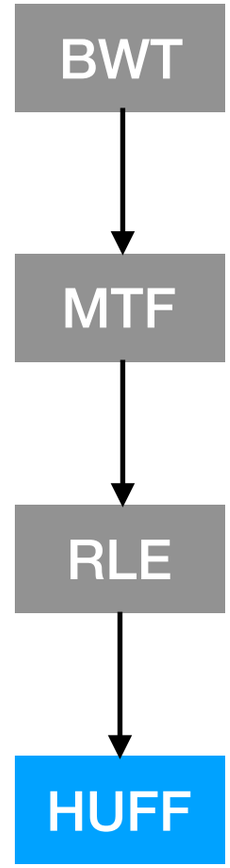
2 = 110

3 = 01

5 = 1110

8 = 1111

- Ausgabe: 01



Huffman-Kodierung

- Eingabe: 3**1**02110231053108

0 = 00

1 = 10

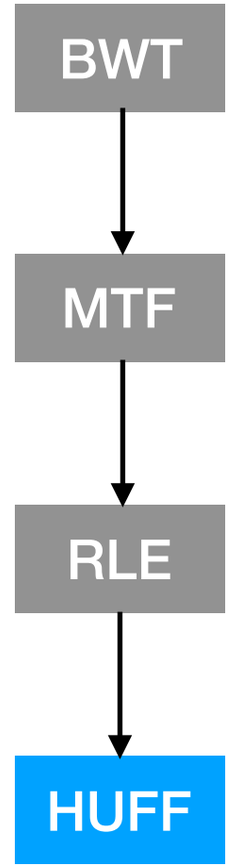
2 = 110

3 = 01

5 = 1110

8 = 1111

- Ausgabe: 01



Huffman-Kodierung

- Eingabe: 3**1**02110231053108

0 = 00

1 = 10

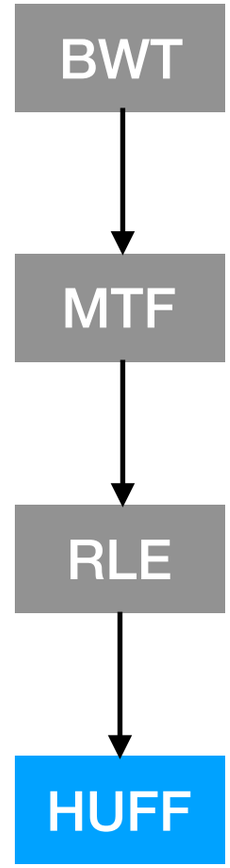
2 = 110

3 = 01

5 = 1110

8 = 1111

- Ausgabe: 0110



Huffman-Kodierung

- Eingabe: 31021102310531O8

0 = 00

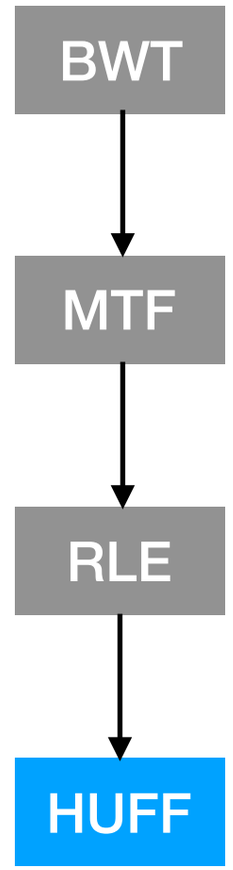
1 = 10

2 = 110

3 = 01

5 = 1110

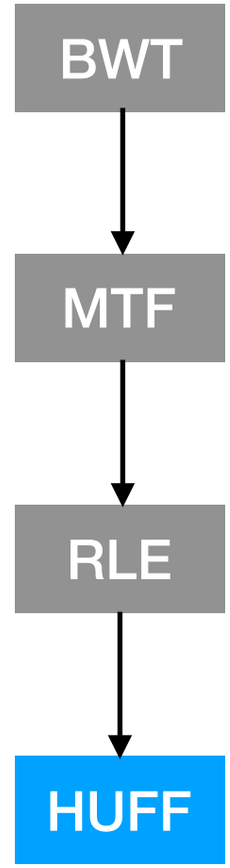
8 = 1111



- Ausgabe: 01100011010100011001100011100110001111

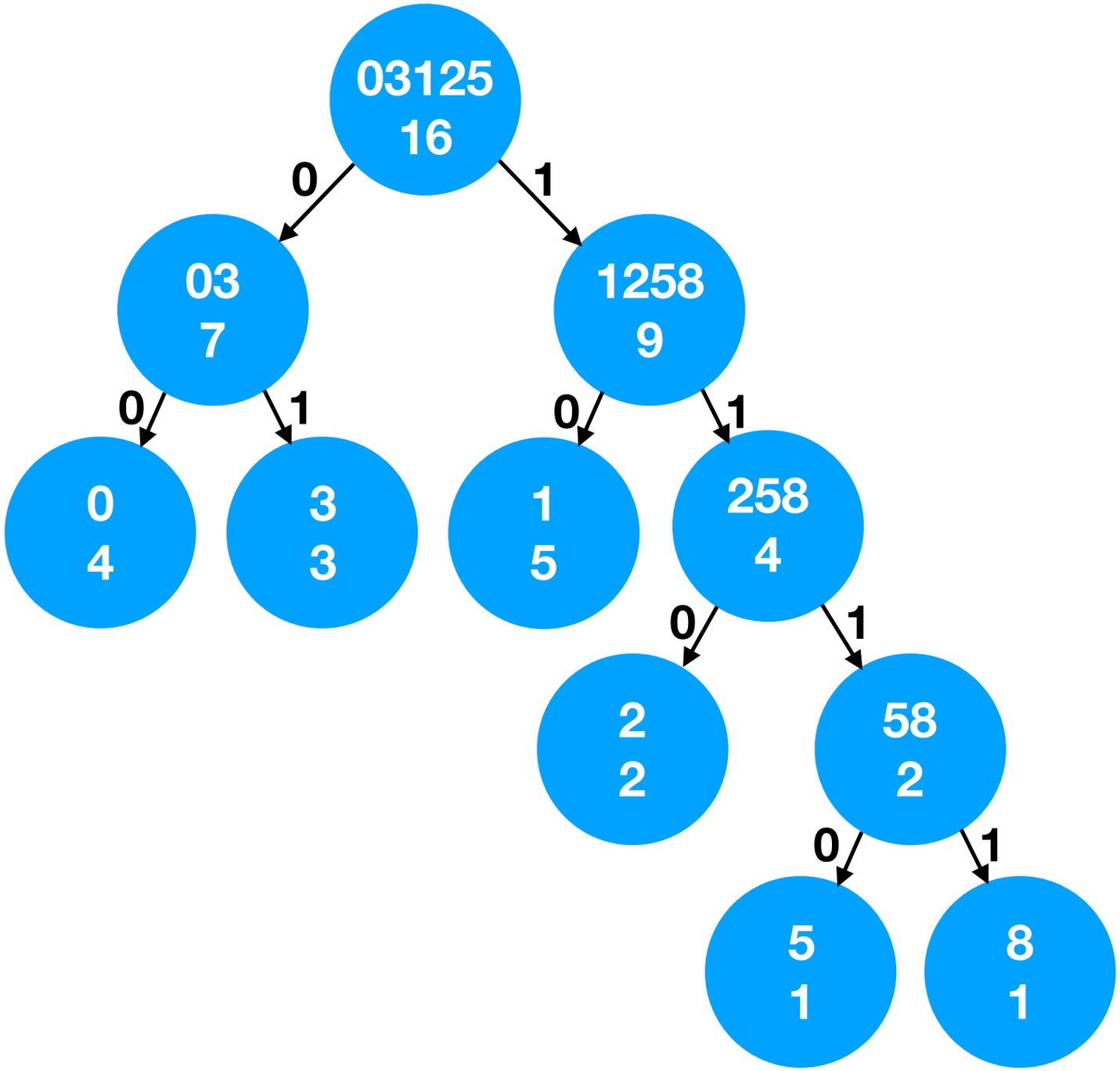
Huffman-Kodierung

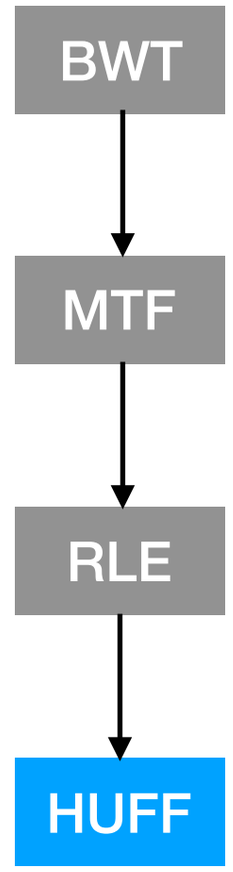
- Auch der Baum muss gespeichert werden



Huffman-Kodierung

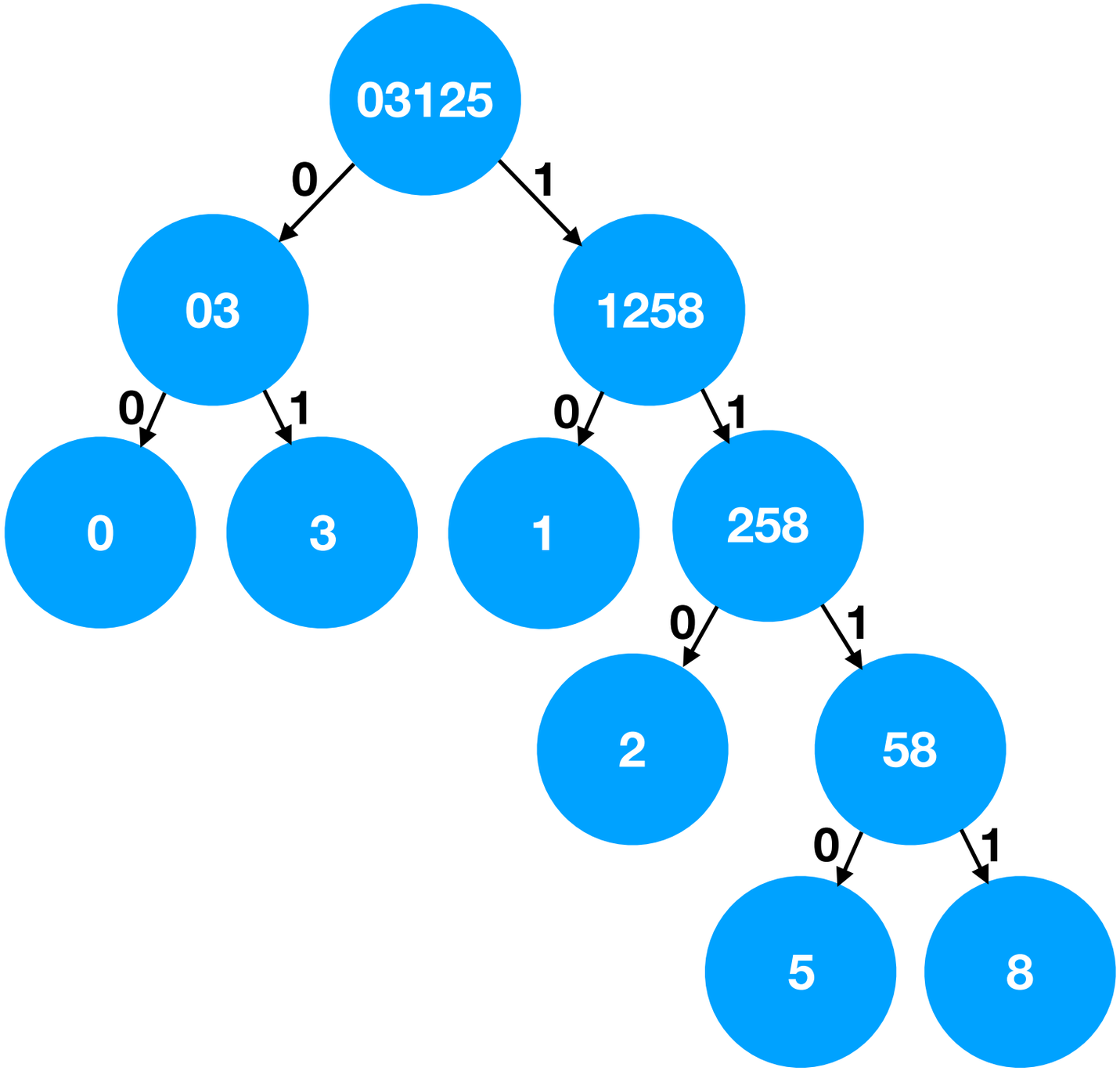
- Auch der Baum muss gespeichert werden

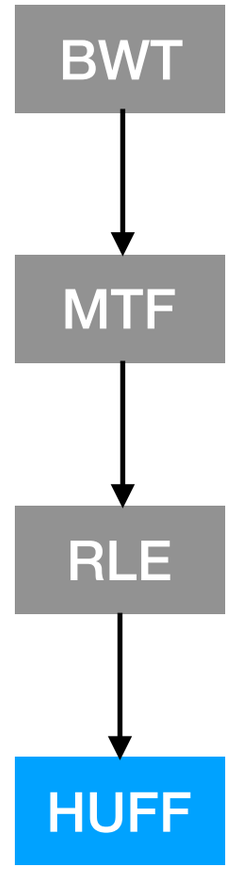




Huffman-Kodierung

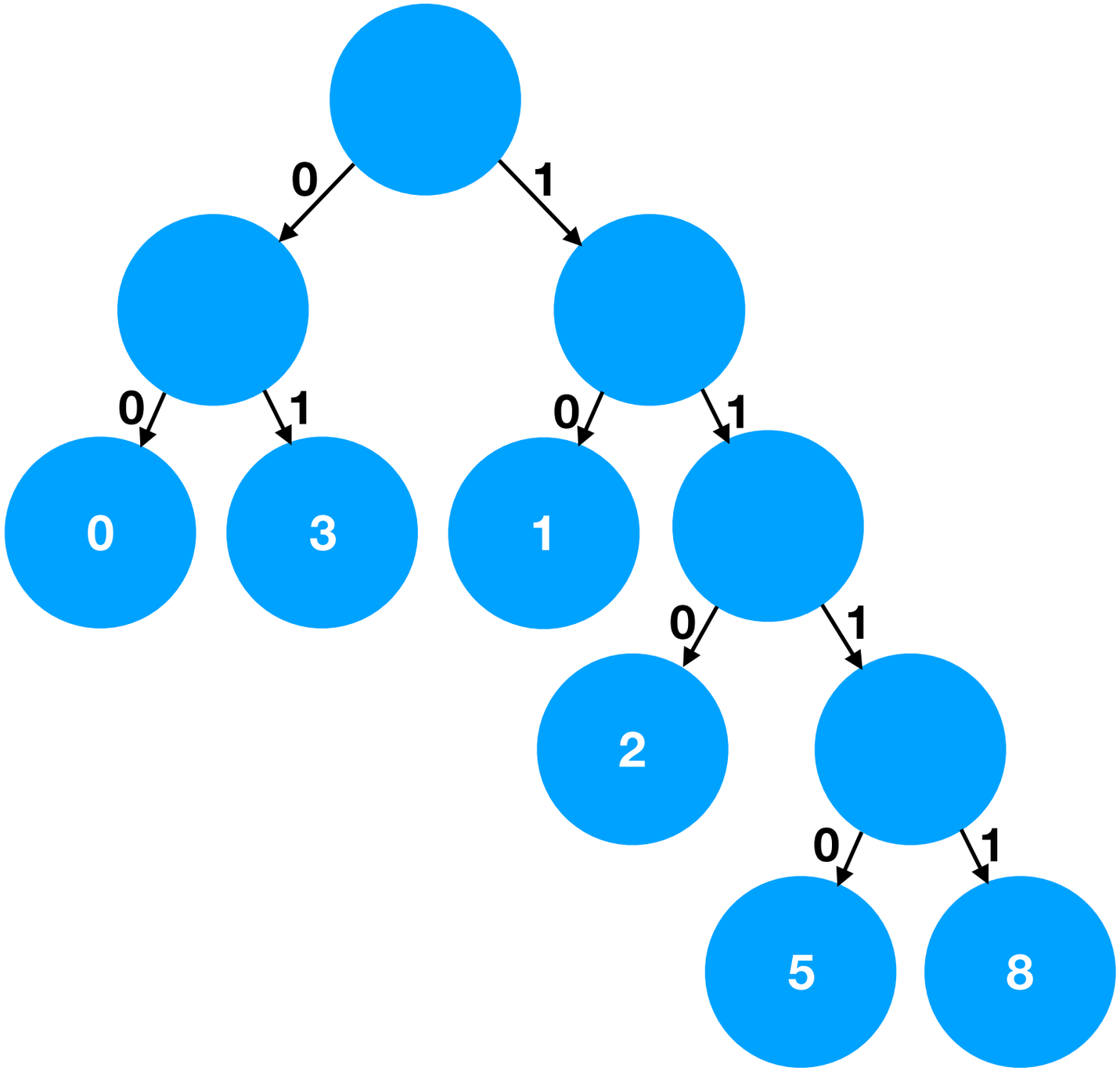
- Auch der Baum muss gespeichert werden

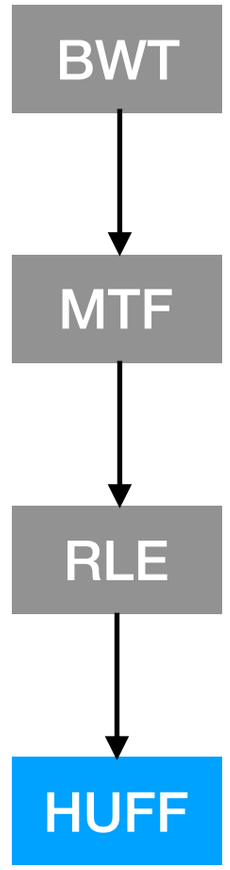




Huffman-Kodierung

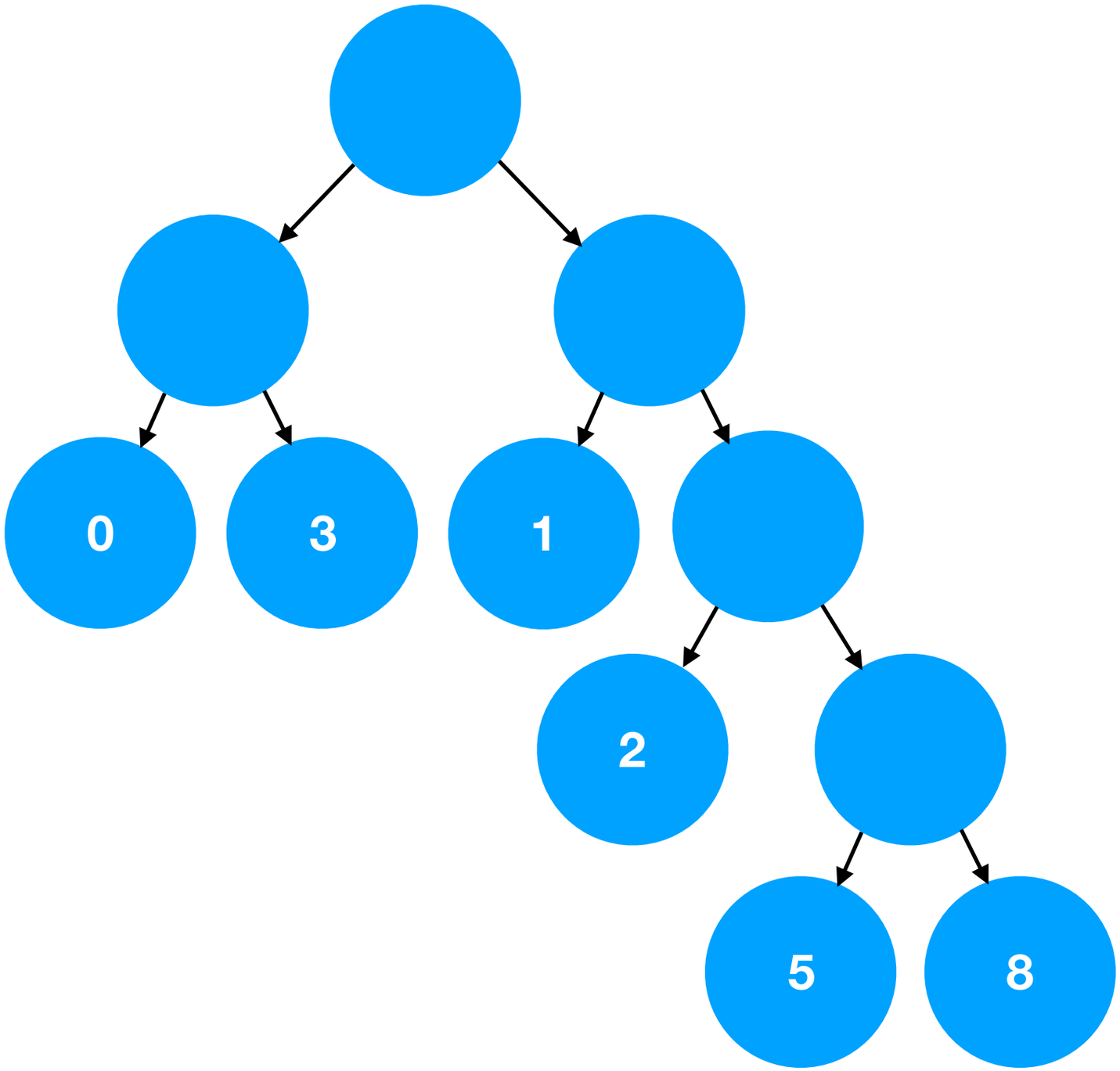
- Auch der Baum muss gespeichert werden

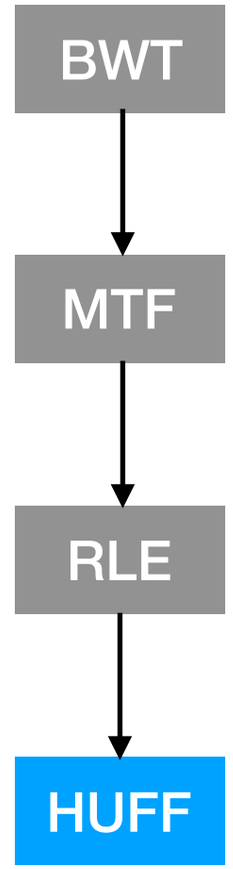




Huffman-Kodierung

- Auch der Baum muss gespeichert werden

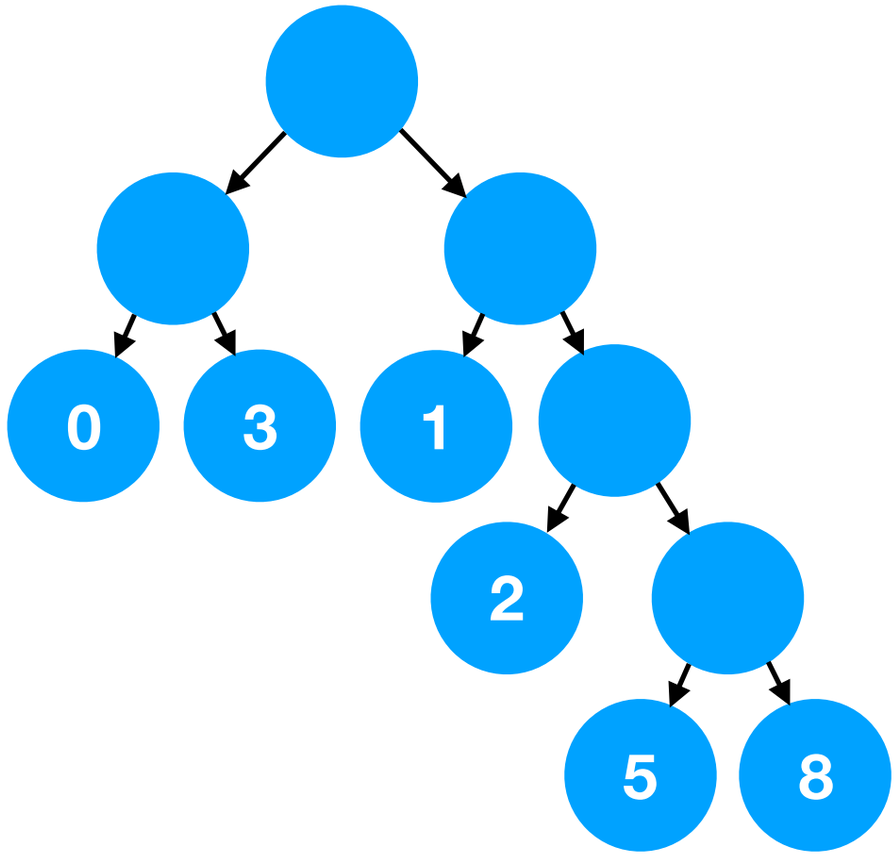


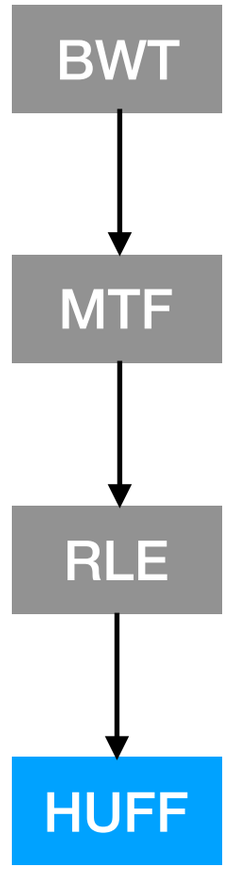


Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert

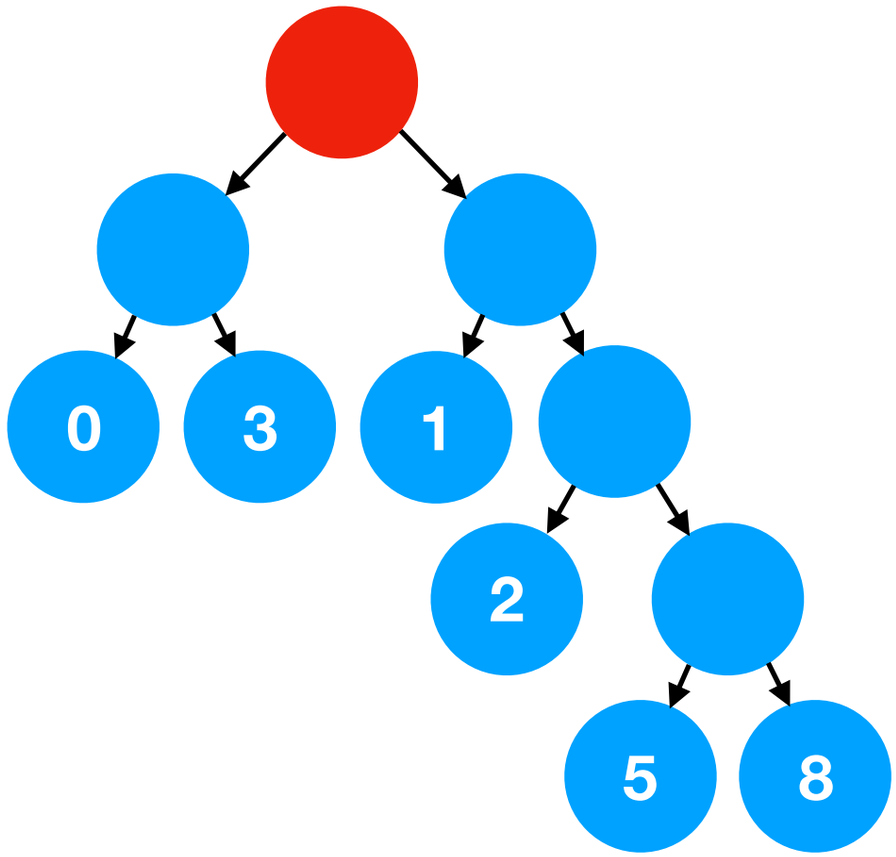


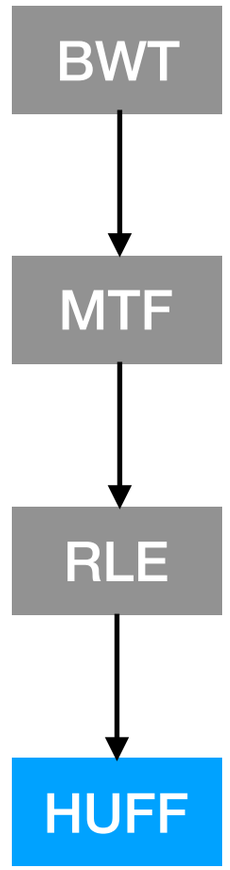


Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



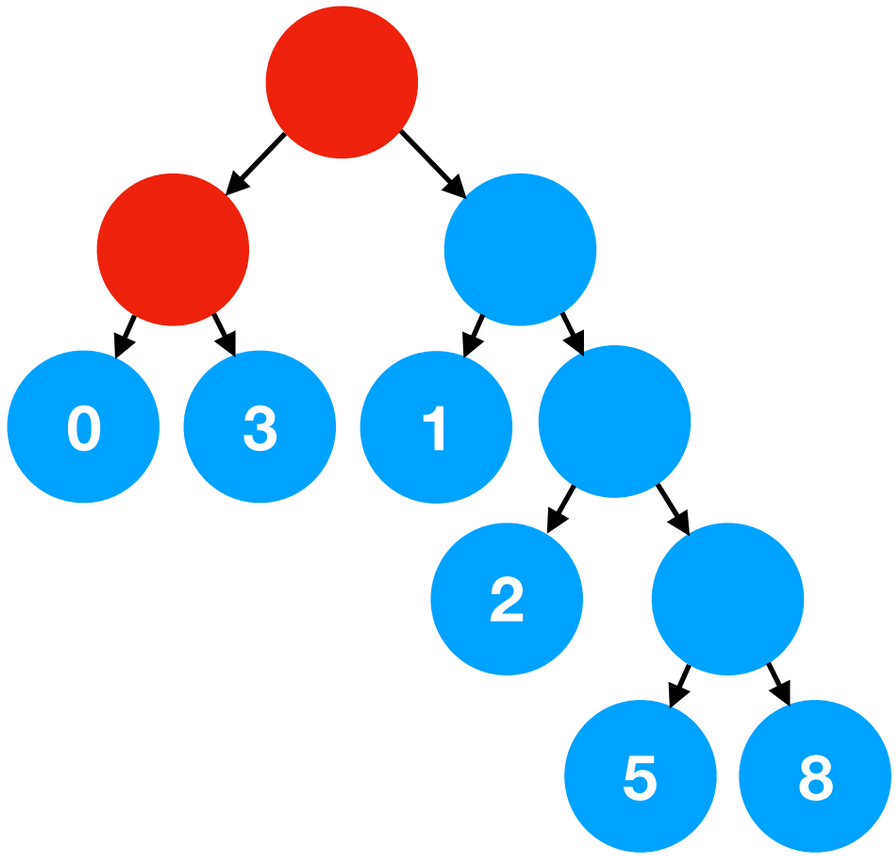


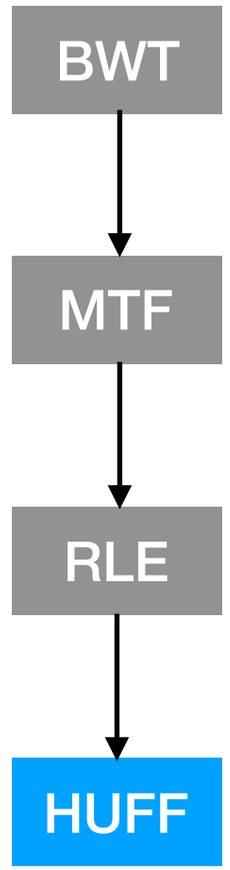
Code: 0

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



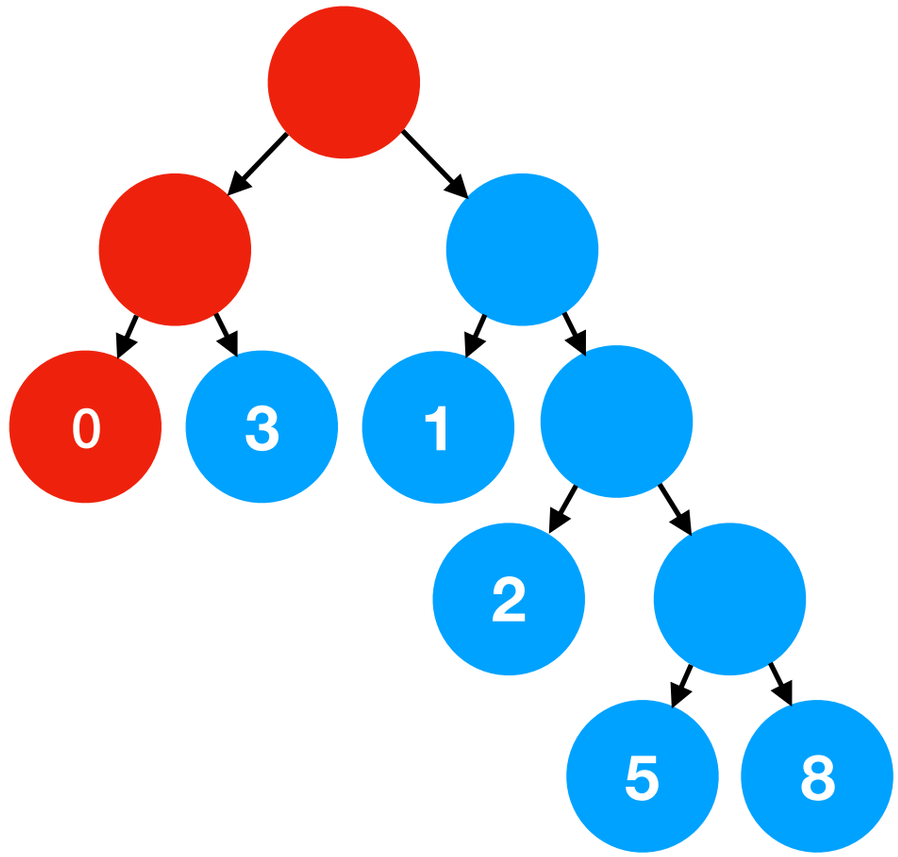


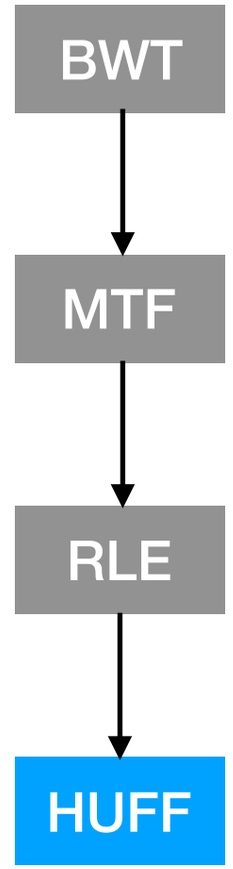
Code: 0 0

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



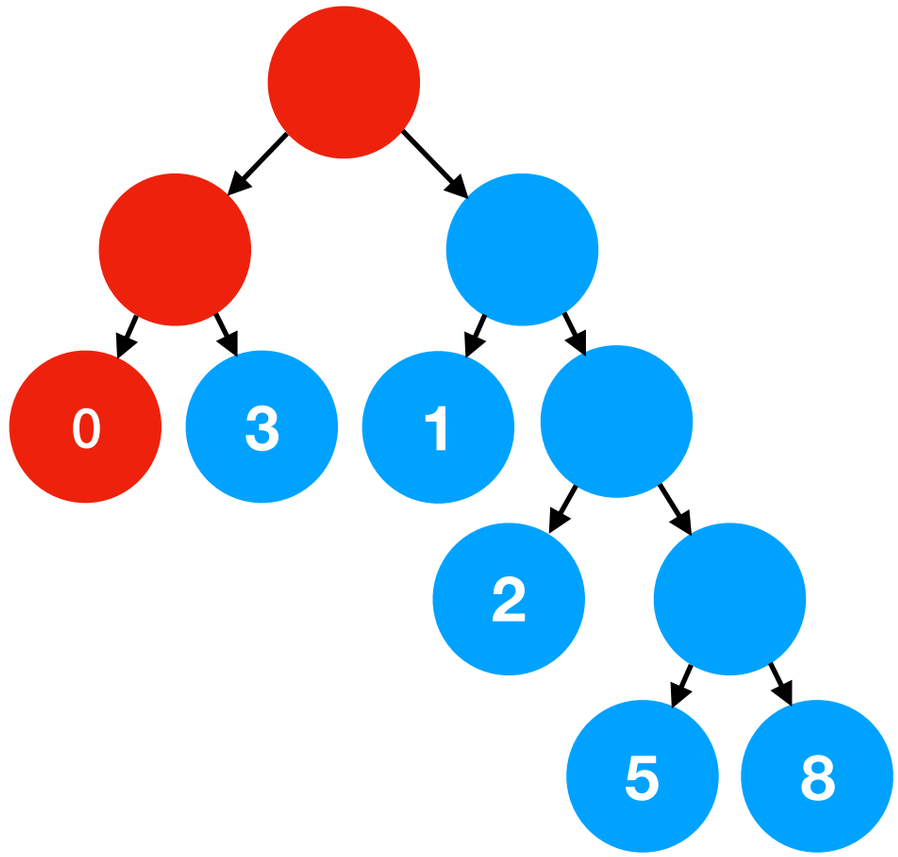


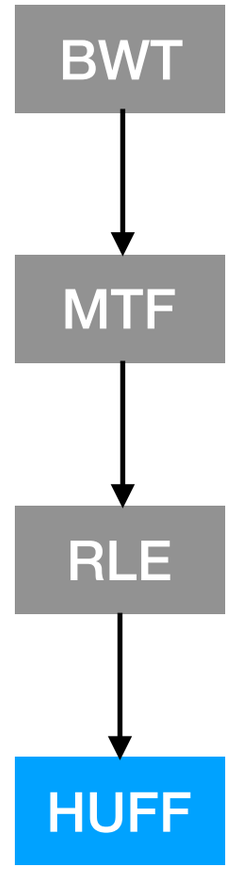
Code: 0 0 1

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



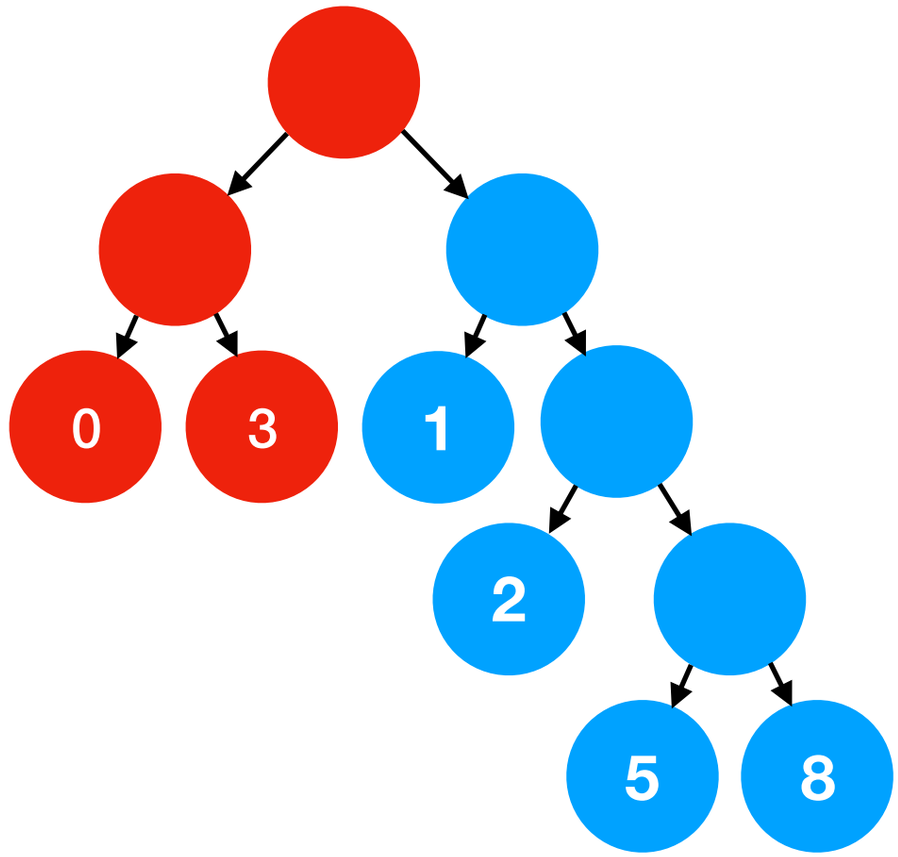


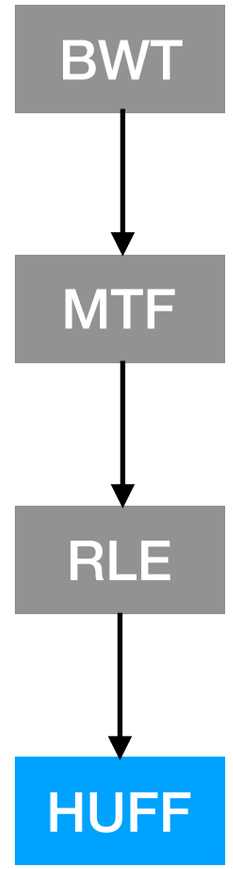
Code: 0 0 100000000

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



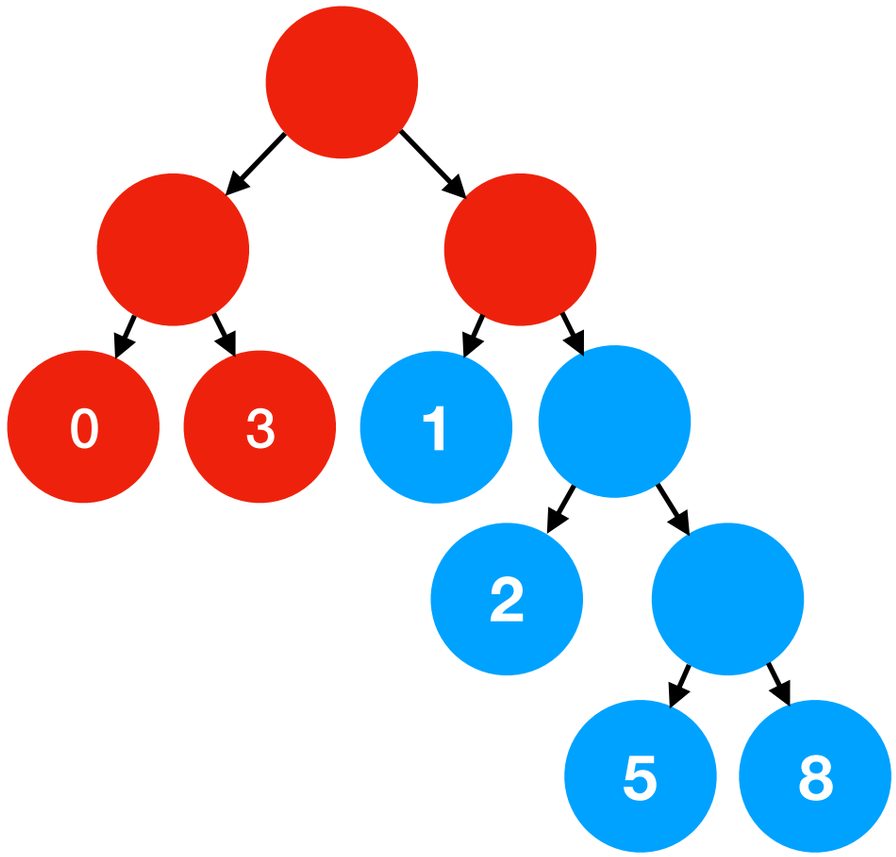


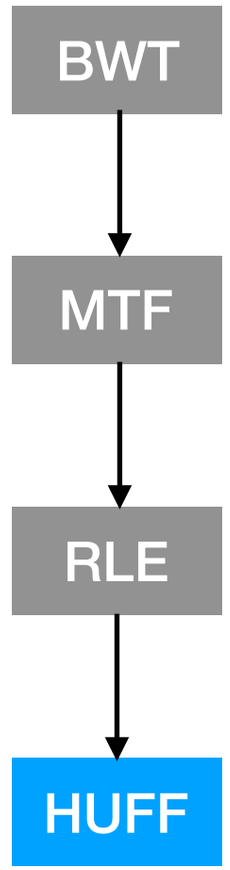
Code: 0 0 100000000 100000011

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



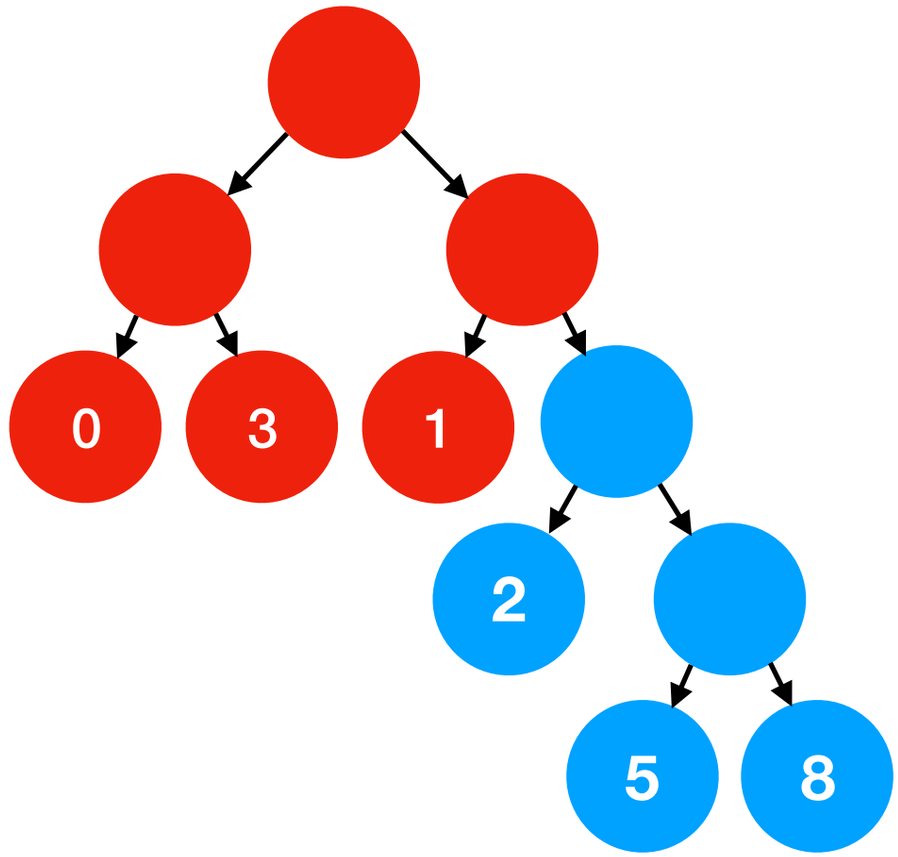


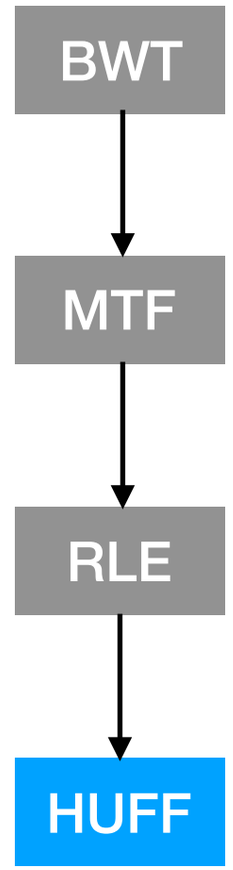
Code: 0 0 100000000 100000011 0

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert



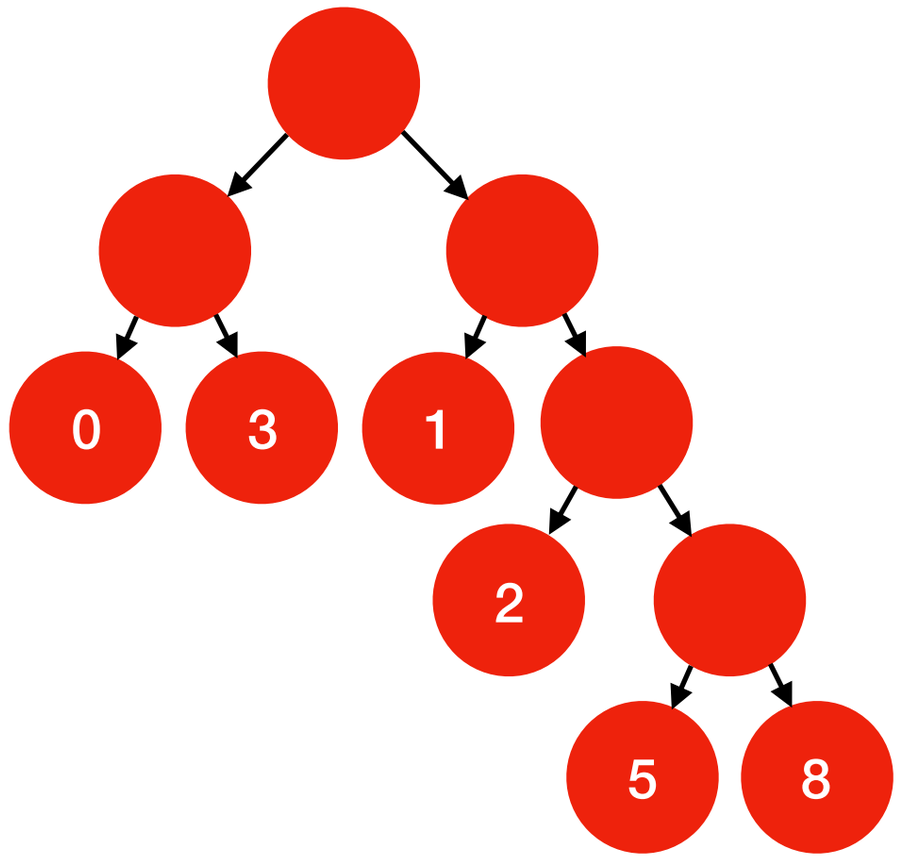


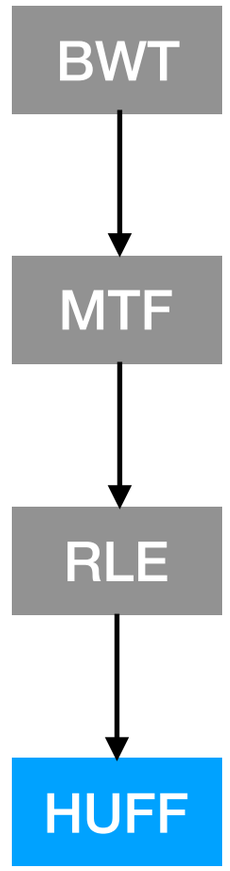
Code: 0 0 100000000 100000011 0 100000001

Huffman-Kodierung

Nicht-Blatt: 0 und beide Kinder

Blatt: 1 und der Wert





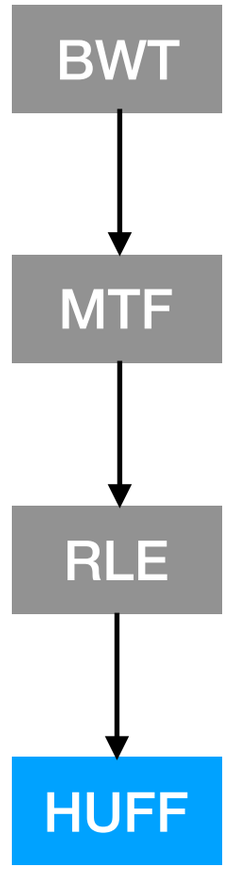
Code: 0 0 100000000 100000011 0 100000001 0 100000010 0 100000101 100001000

Huffman-Kodierung

- Ausgabe:

00100000000100000011010000000101000000
10010000010110000100 0 (Baum)

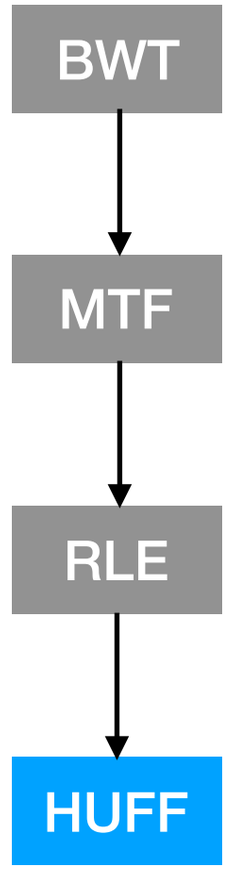
01100011010100011001100011100110001111
(Codierung)



Huffman-Dekodierung

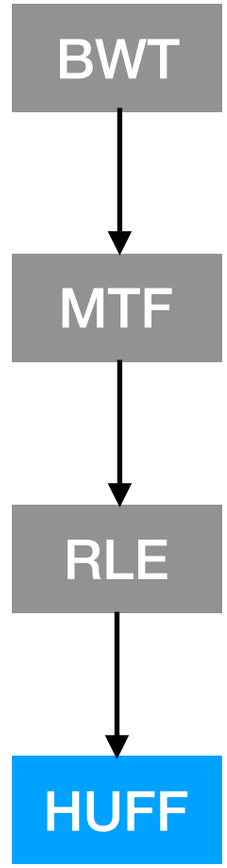
- Eingabe:

00100000000100000011010000000101000000
10010000010110000100001100011010100011
00110001110011000111

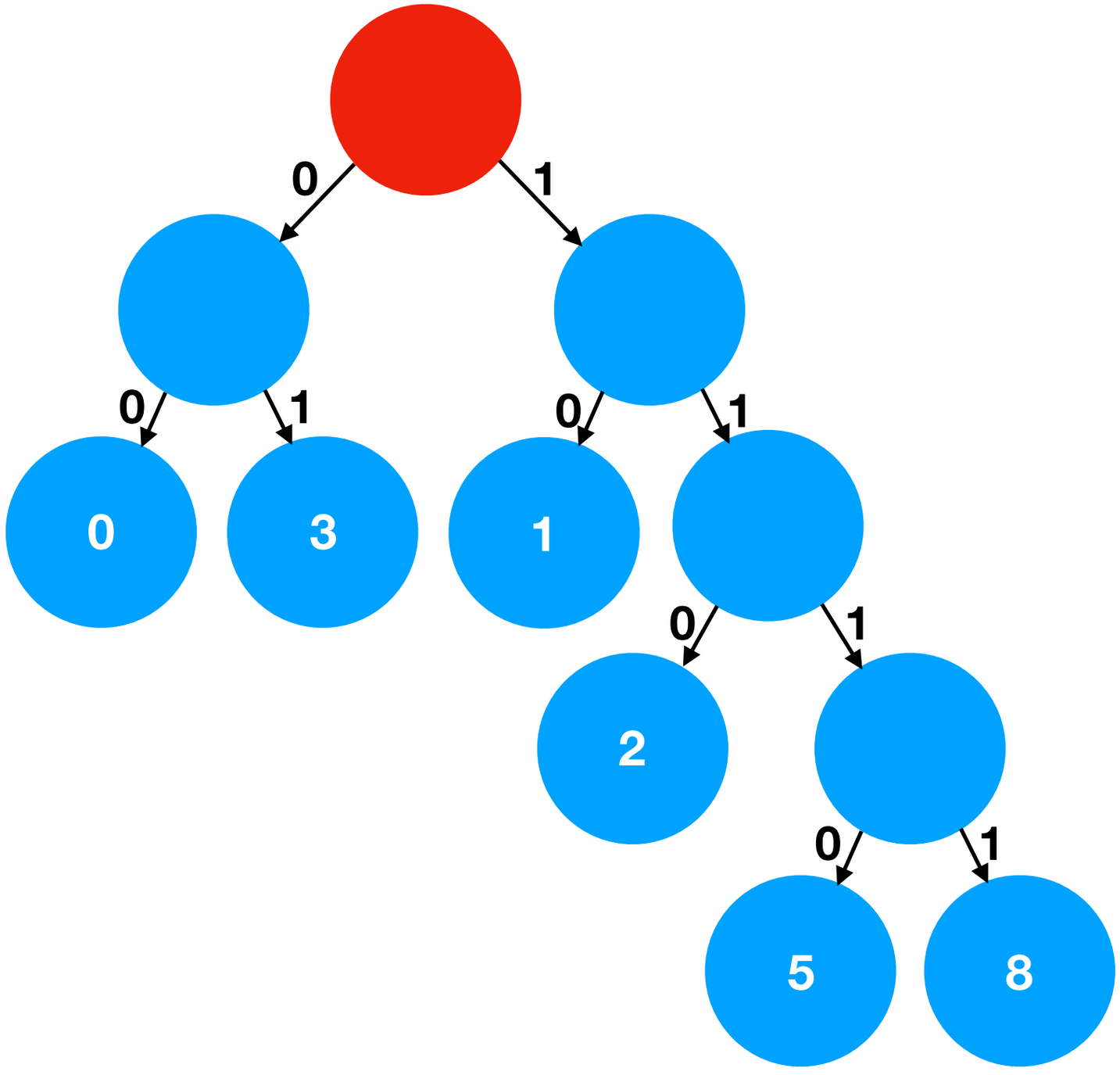


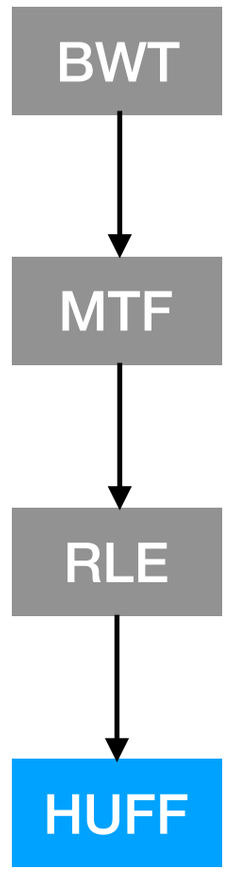
Huffman-Dekodierung

1. Baum aus der Datei auslesen

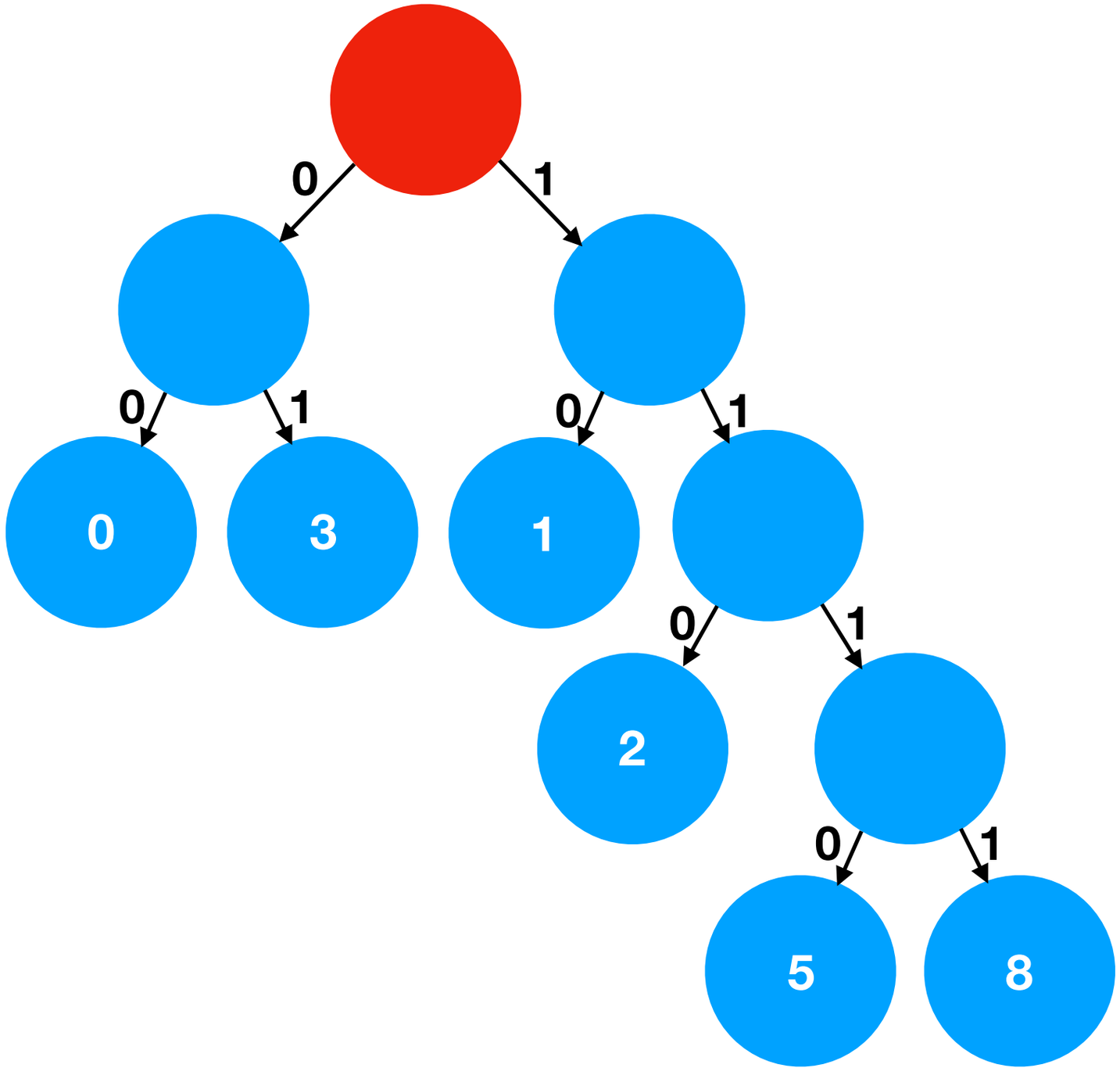


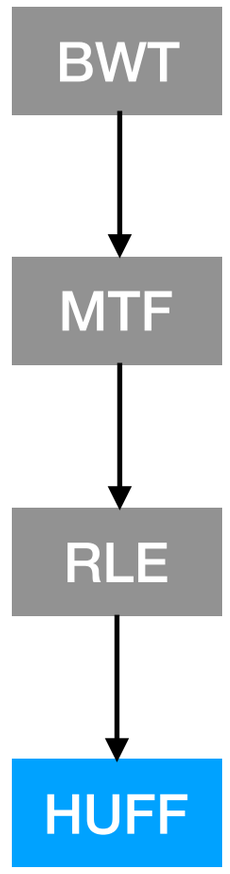
Huffman-Dekodierung





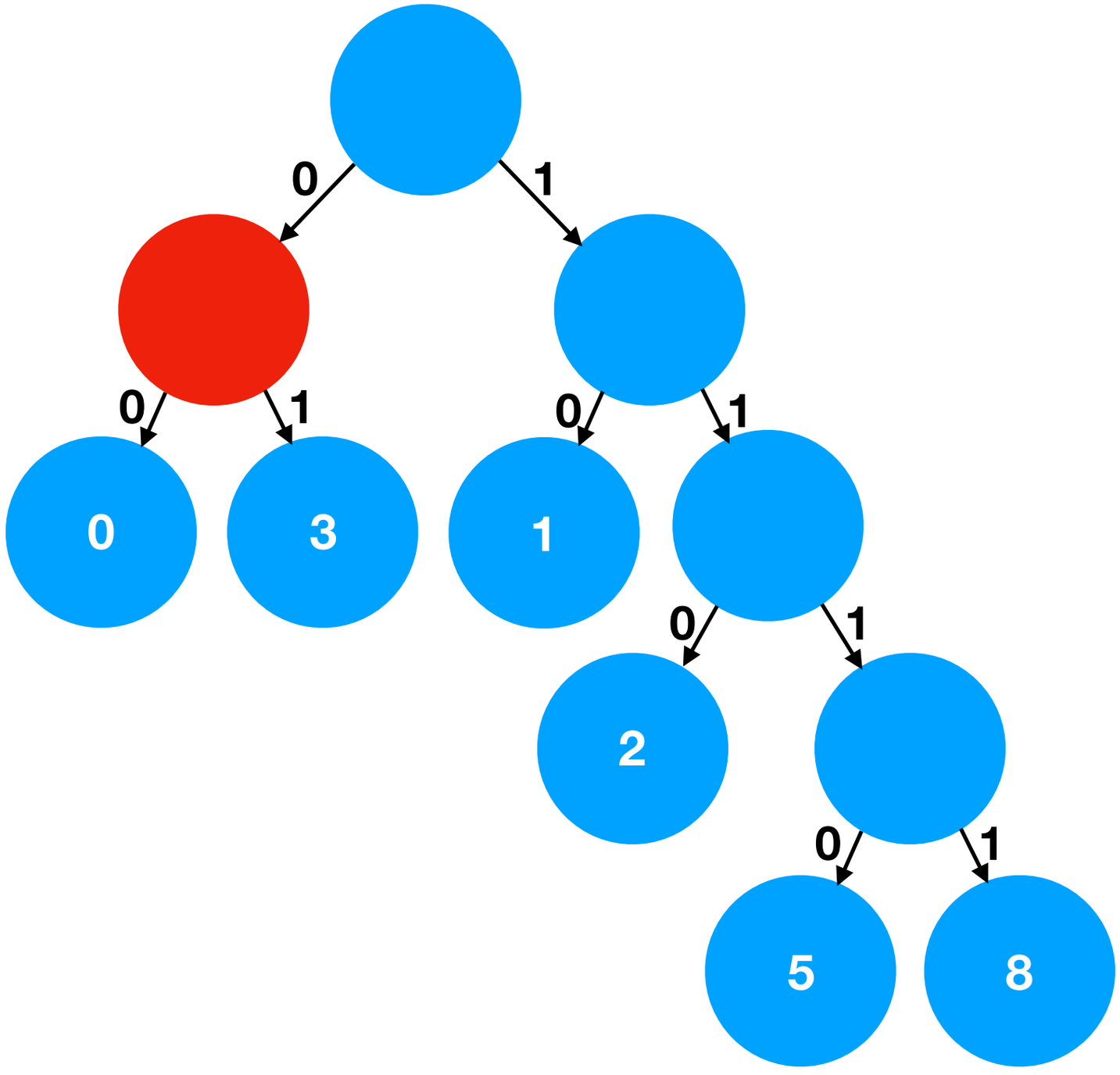
Huffman-Dekodierung

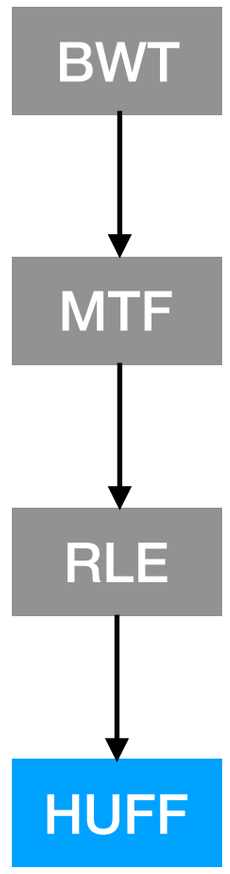




- Code: 01100011010100011001100011100110001111

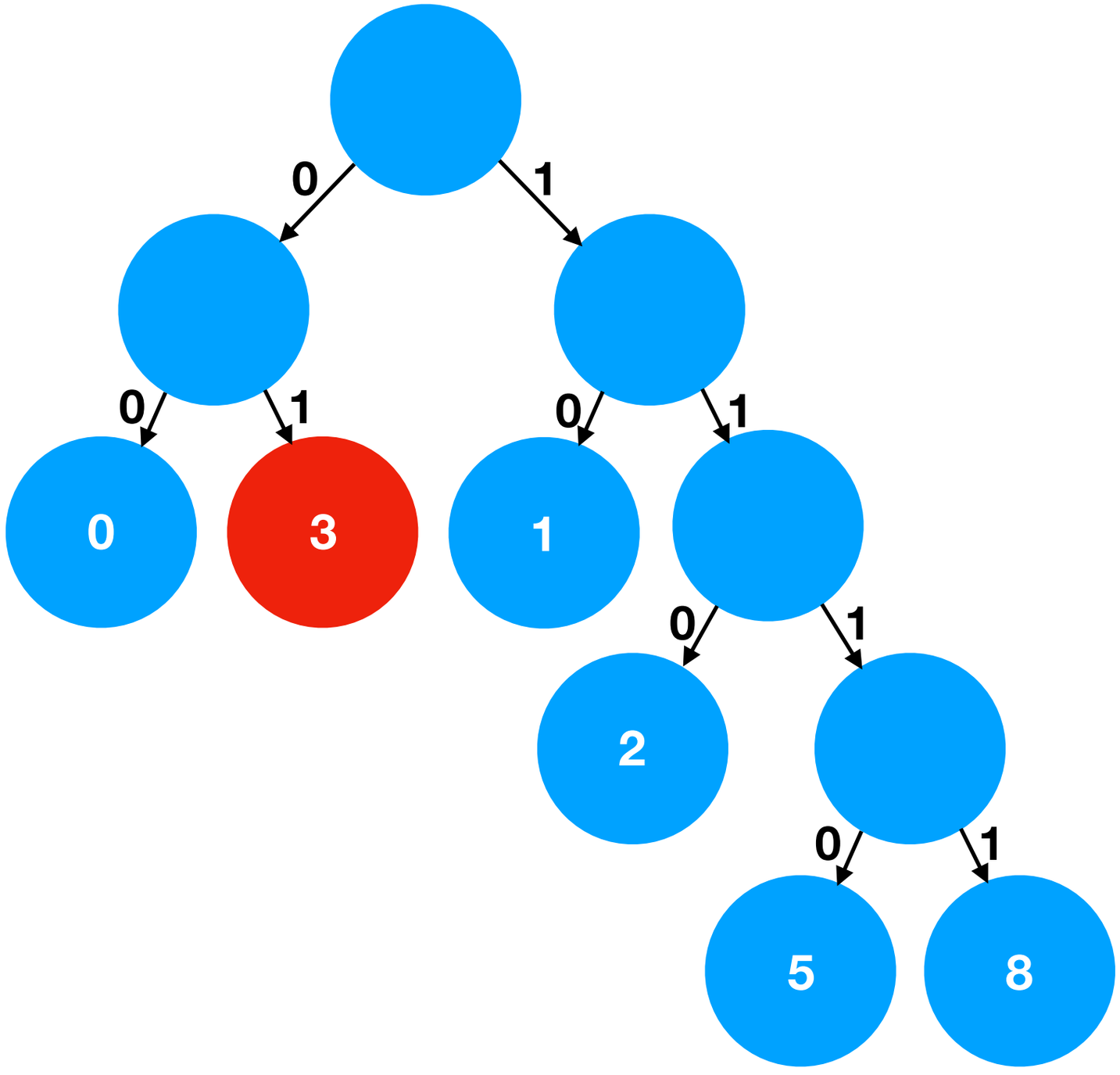
Huffman-Dekodierung

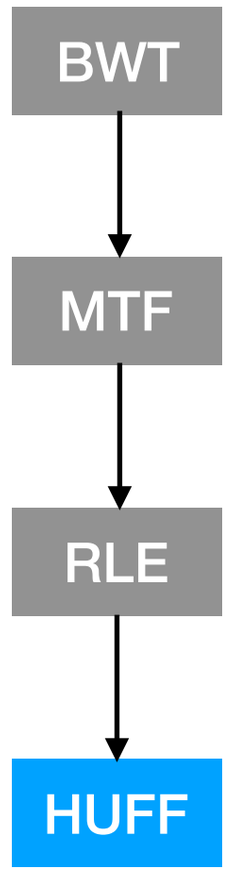




- Code: **0**1100011010100011001100011100110001111

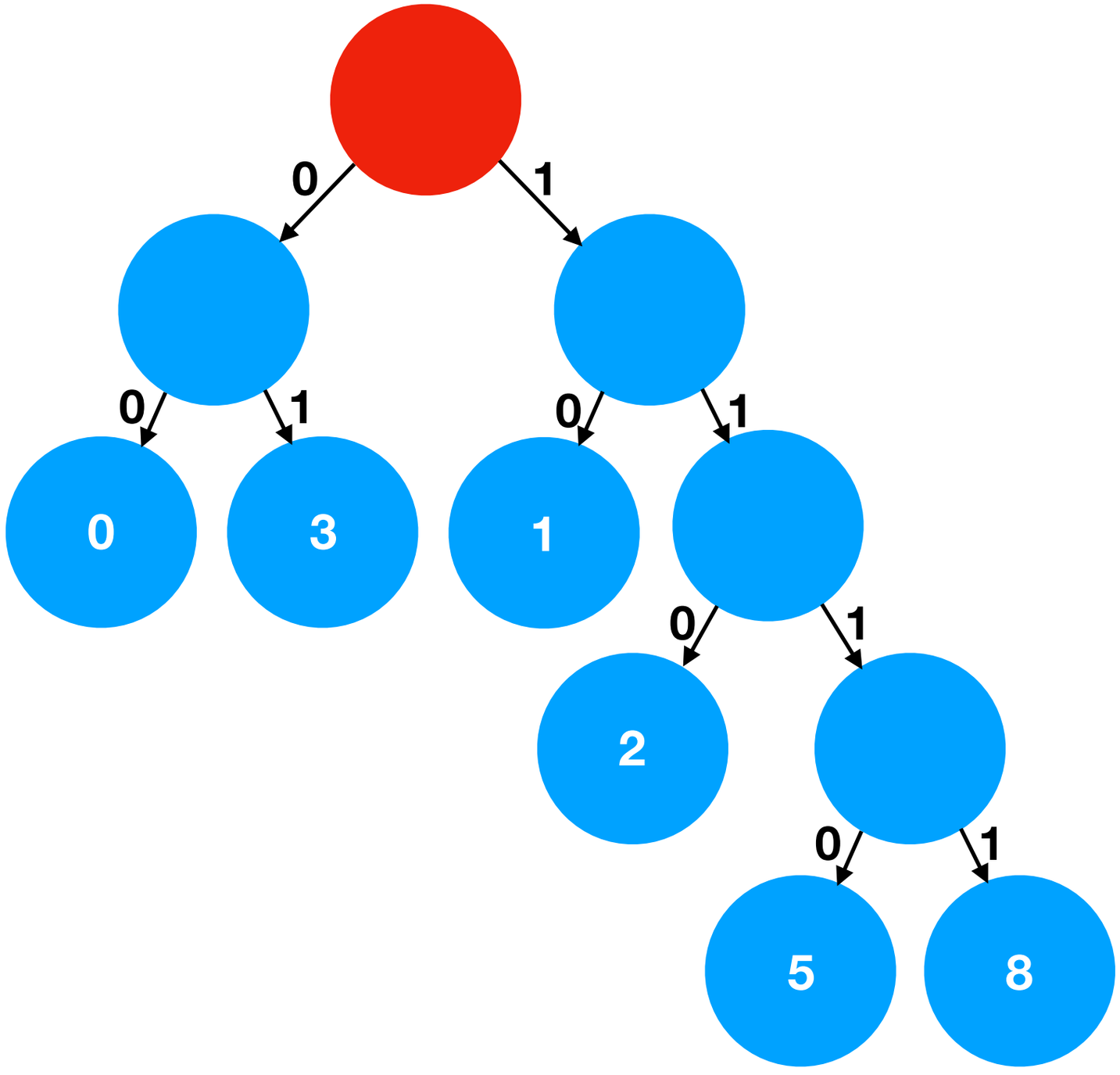
Huffman-Dekodierung

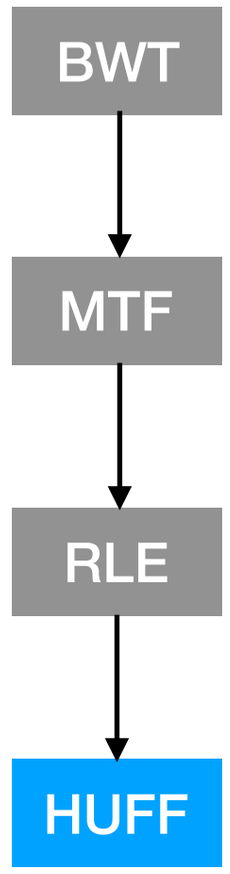




- Code: **01**1000110101000110011000111001100001111

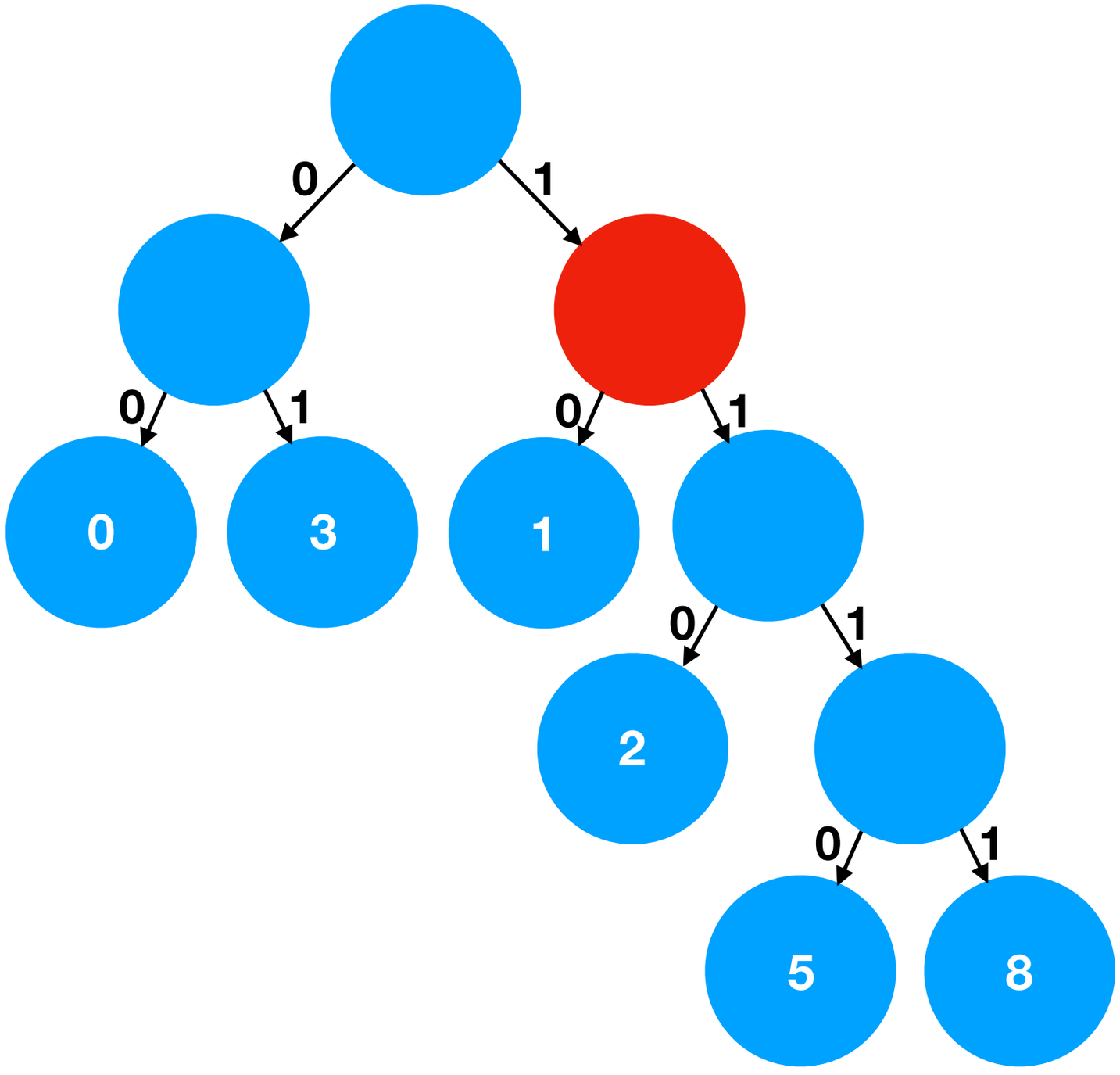
Huffman-Dekodierung

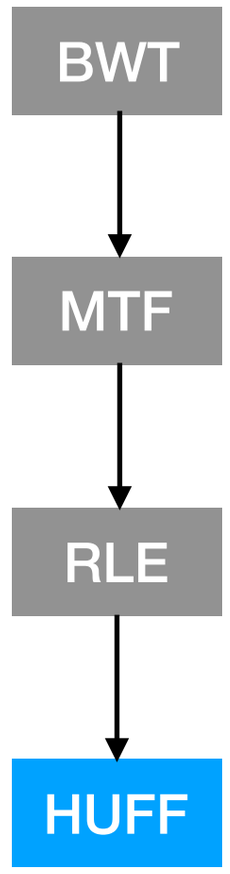




- Code: **01**1000110101000110011000111001100001111
- Ausgabe: 3

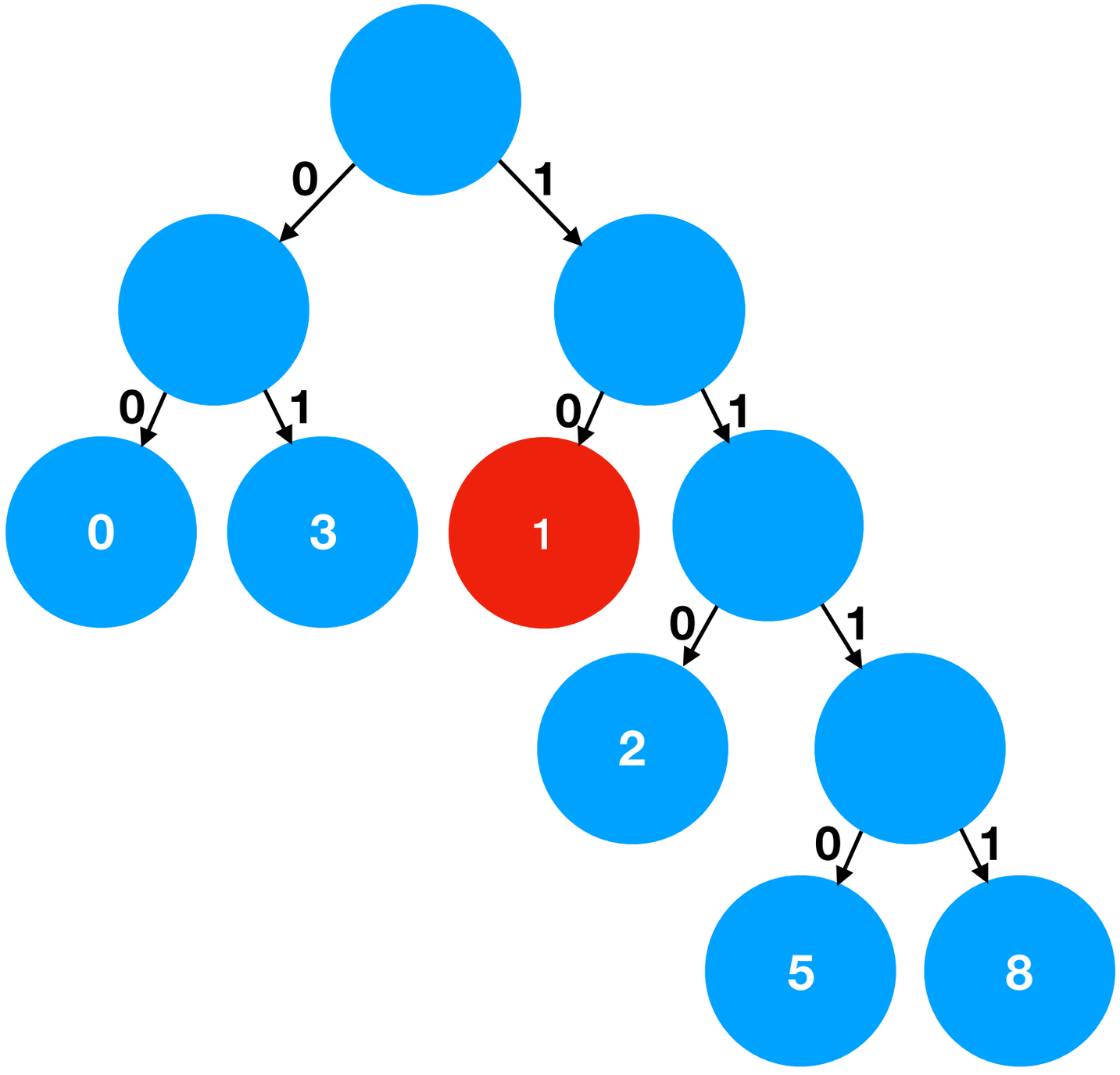
Huffman-Dekodierung

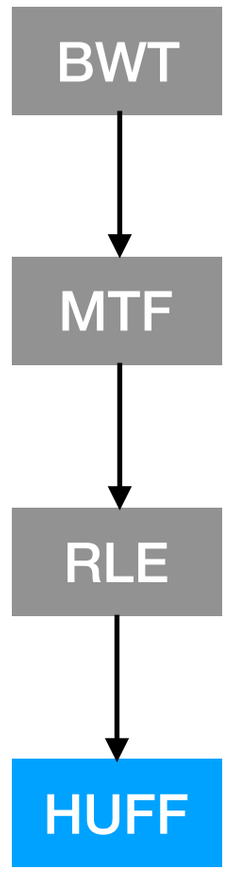




- Code: **011**0001101010001100110001110011000 1111
- Ausgabe: 3

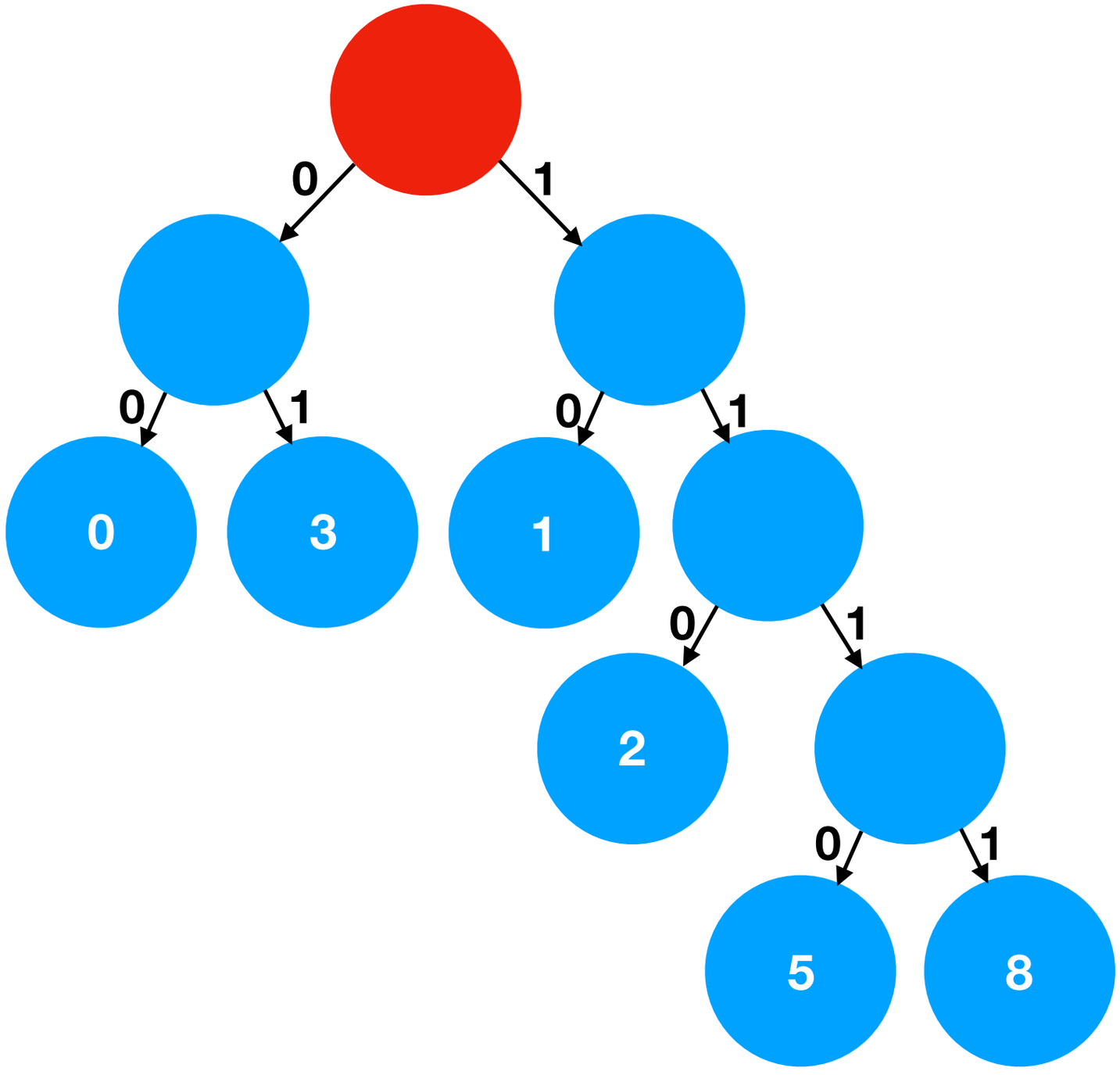
Huffman-Dekodierung

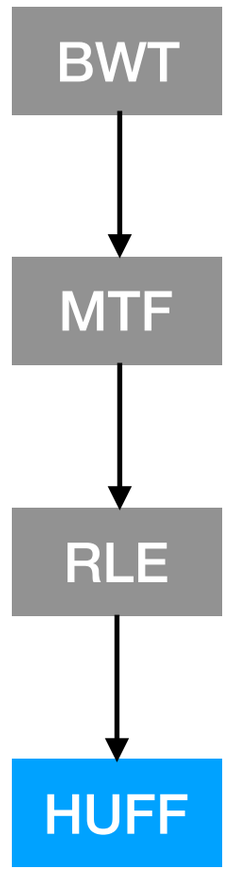




- Code: **0110**0011010100011001100011100110001111
- Ausgabe: 3

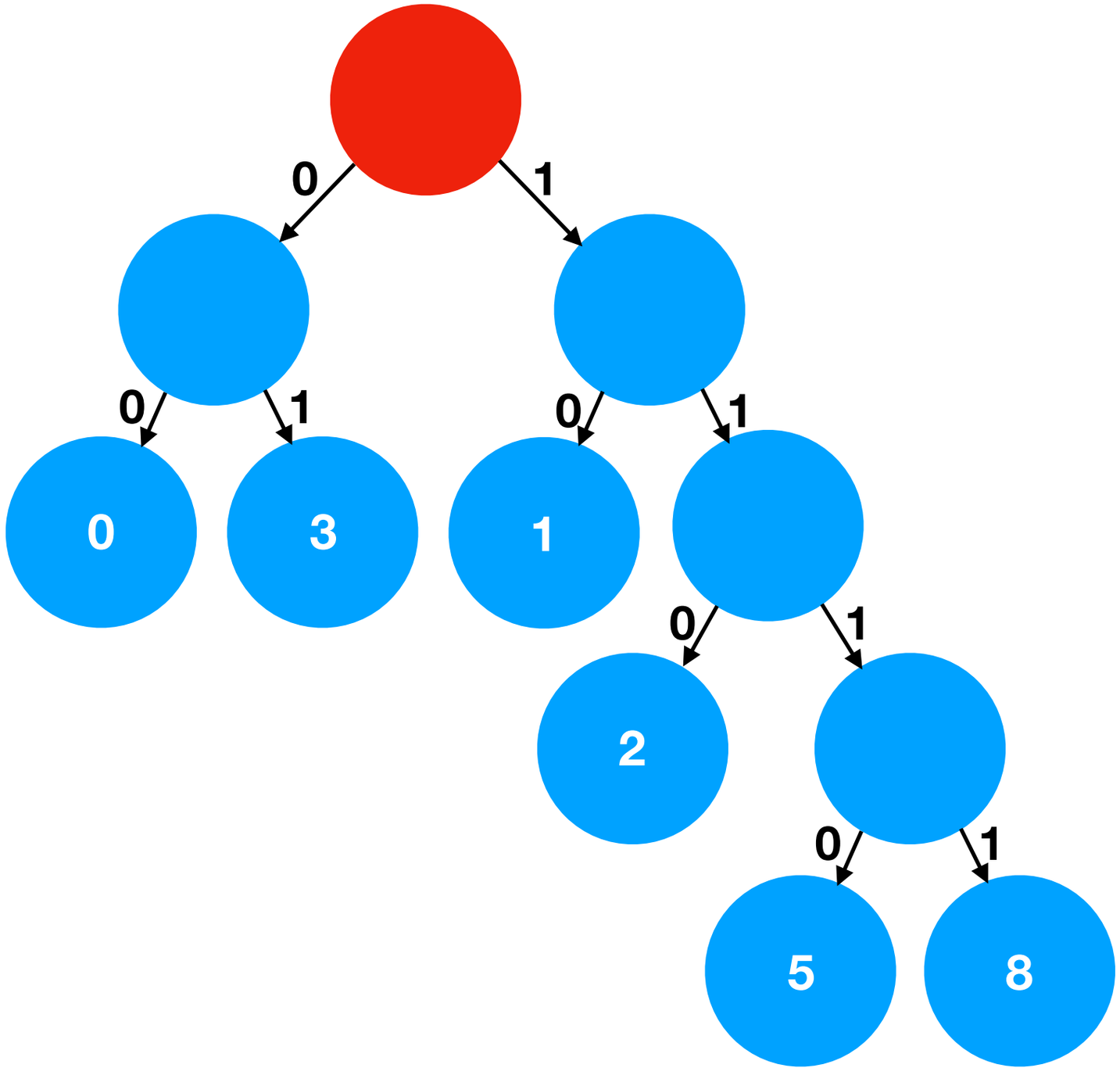
Huffman-Dekodierung

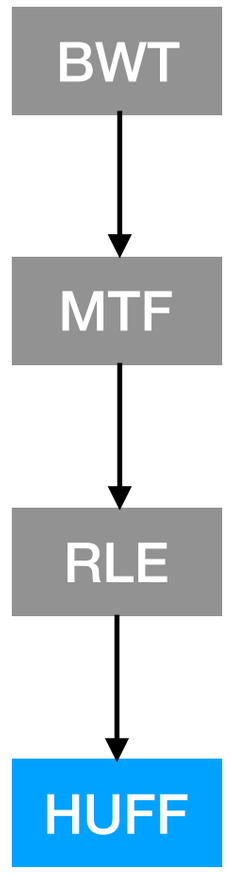




- Code: **0110**0011010100011001100011100110001111
- Ausgabe: 31

Huffman-Dekodierung





- Code: **01100011010100011001100011100110001111**
- Ausgabe: 3102110231053108

Zusammenfassung

- Deflate ist ein weit verbreitetes Verfahren um Daten zu komprimieren
- Google macht’s am besten
- Das Kompressionsverfahren bzip2 kombiniert mehrere Teilschritte die in Zusammenarbeit zu Erfolg führen

Vielen Dank für eure Aufmerksamkeit!

Literaturverzeichnis

- [AKSV15]: Jyrki Alakuijala, Evgenii Kliuchnikov, Zoltan Szabadka, and Lode Vande- venne. Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms. Technical report, Google, Inc., September 2015.
- [Ala12]: Jyrki Alakuijala. Brotli Compressed Data Format. https://datatracker. ietf.org/doc/rfc7932/, 2015. ((letzter Zugriff: 05-01-2018).
- [Ble01]: Guy E Blelloch. Introduction to data compression. Computer Science Department, Carnegie Mellon University, 2001.
- [BW94]: M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.
- [bwt18]: Burrows-Wheeler-Transformation. https://de.wikipedia.org/wiki/Burrows-Wheeler-Transformation, 2018. (letzter Zugriff: 05-01-2018).
- [Deu96]: L Peter Deutsch. DEFLATE compressed data format specification version 1.3. https://tools.ietf.org/html/rfc1951, 1996. (letzter Zugriff: 05-01-2018).
- [DFK]: Bjoern Deppe, Frank Fuest, Ingo Kaulbach . Verlustfreie Datenkompression mit Hilfe der Burrows Wheeler Transformation
- [Gil04]: Jeff Gilchrist. Parallel data compression with bzip2. In Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems, volume 16, pages 559–564, 2004.
- [huff18]: Huffman-Kodierung. https://de.wikipedia.org/wiki/Huffman-Kodierung, 2018. (letzter Zugriff: 05-01-2018).
- [LA12]: R. Lenhardt and J. Alakuijala. Gipfeli - High Speed Compression Algorithm. In *Data Compression Conference (DCC), 2012*, pages 109–118, April 2012.
- [mov18]: Move-to-front. https://de.wikipedia.org/wiki/Move_to_front, 2018. (letzter Zugriff: 05-01-2018).