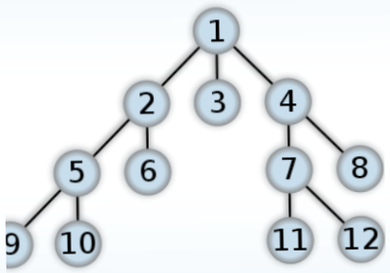
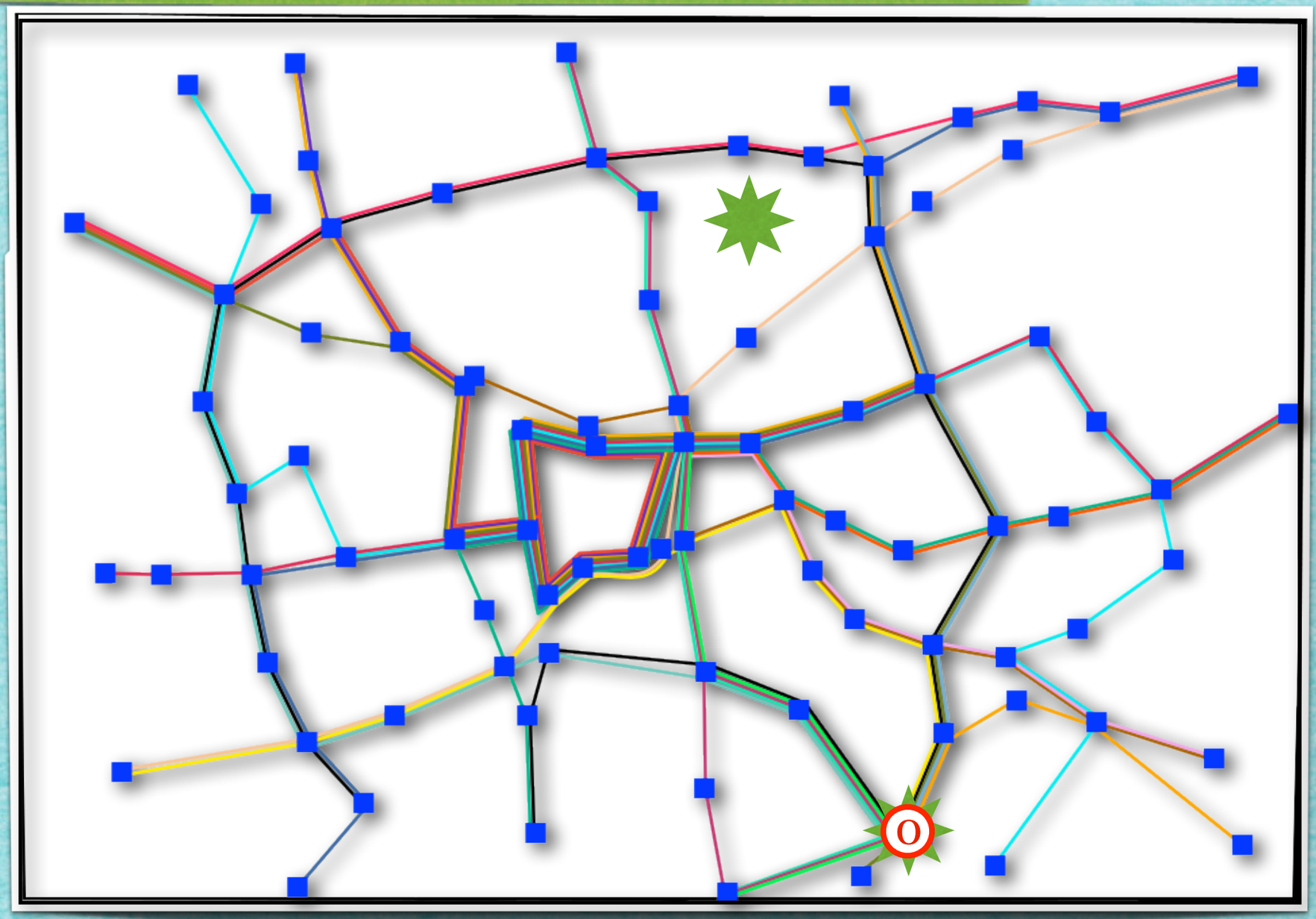


Kapitel 3.9: Eigenschaften von DFS und BFS

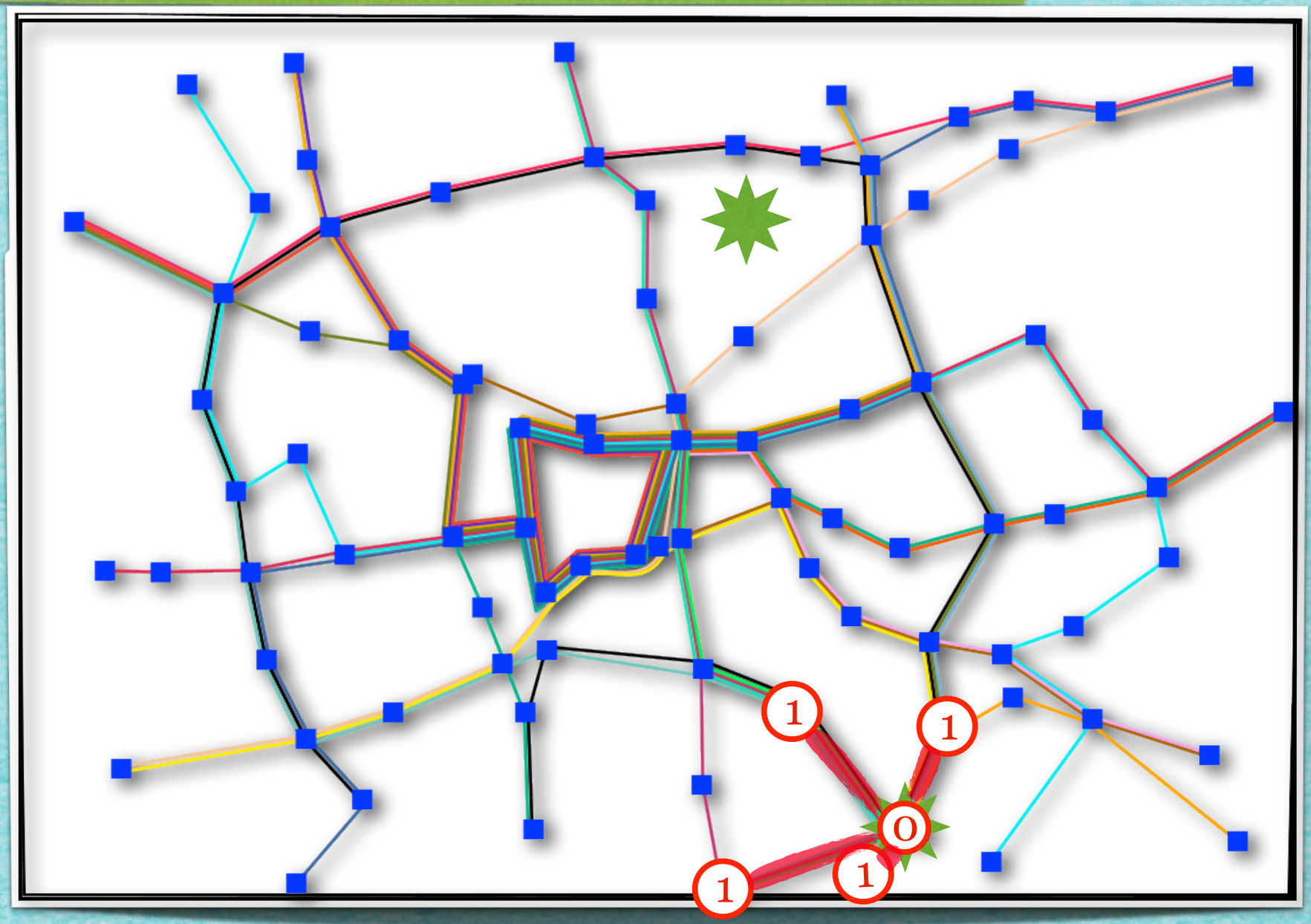
Algorithmen und Datenstrukturen
WS 2024/25

Prof. Dr. Sándor Fekete

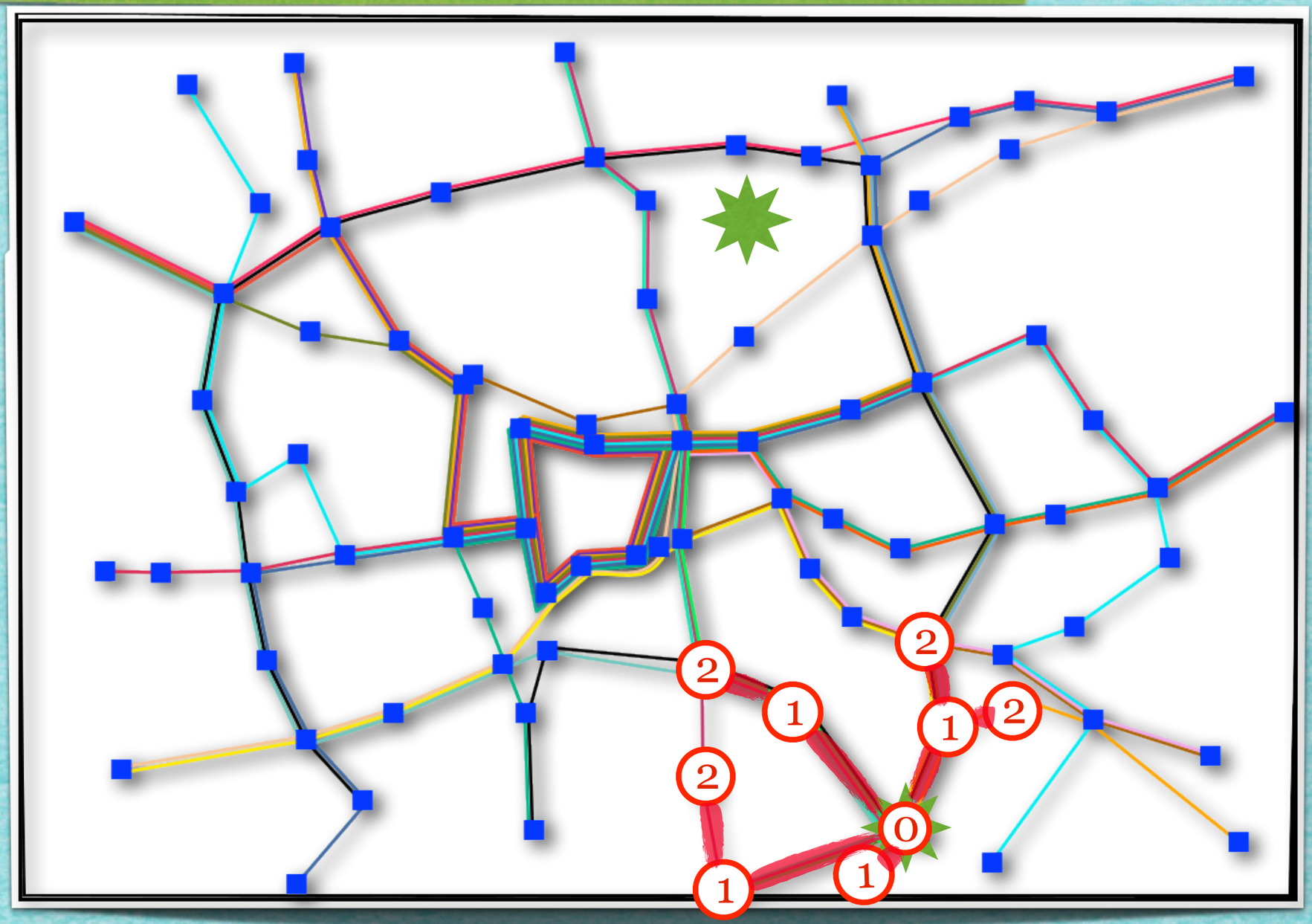
Wellenreiten in Graphen



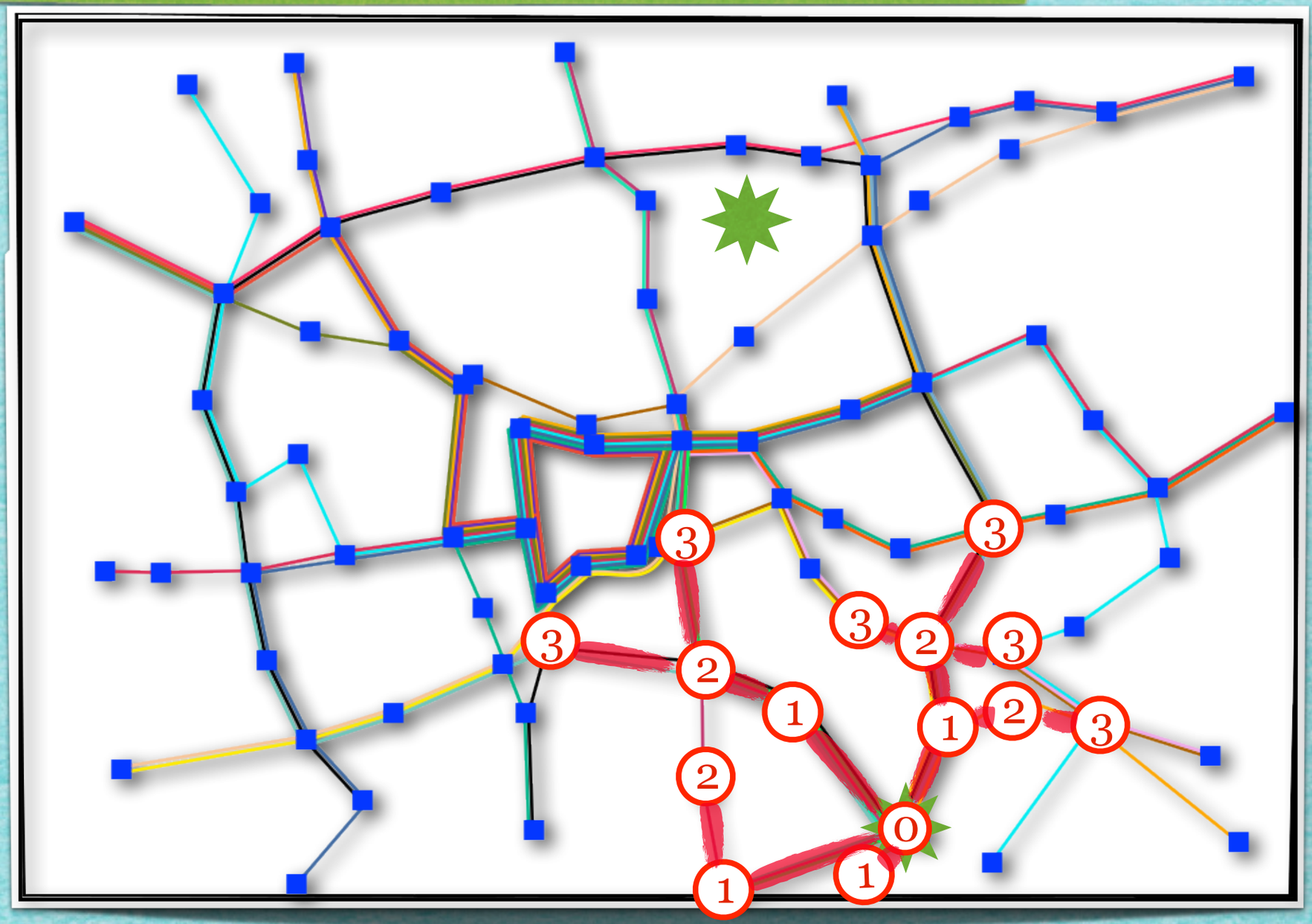
Wellenreiten in Graphen



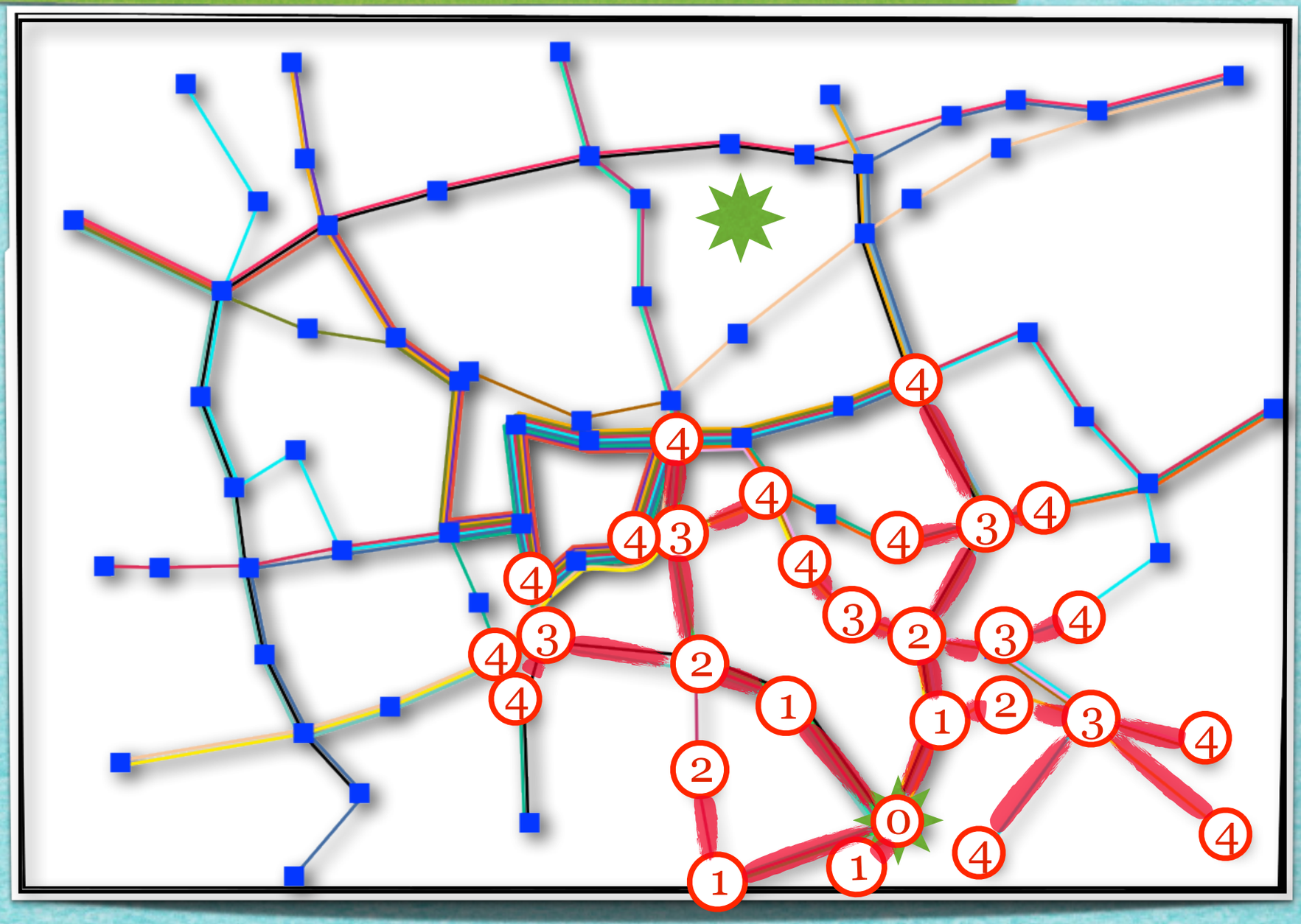
Wellenreiten in Graphen



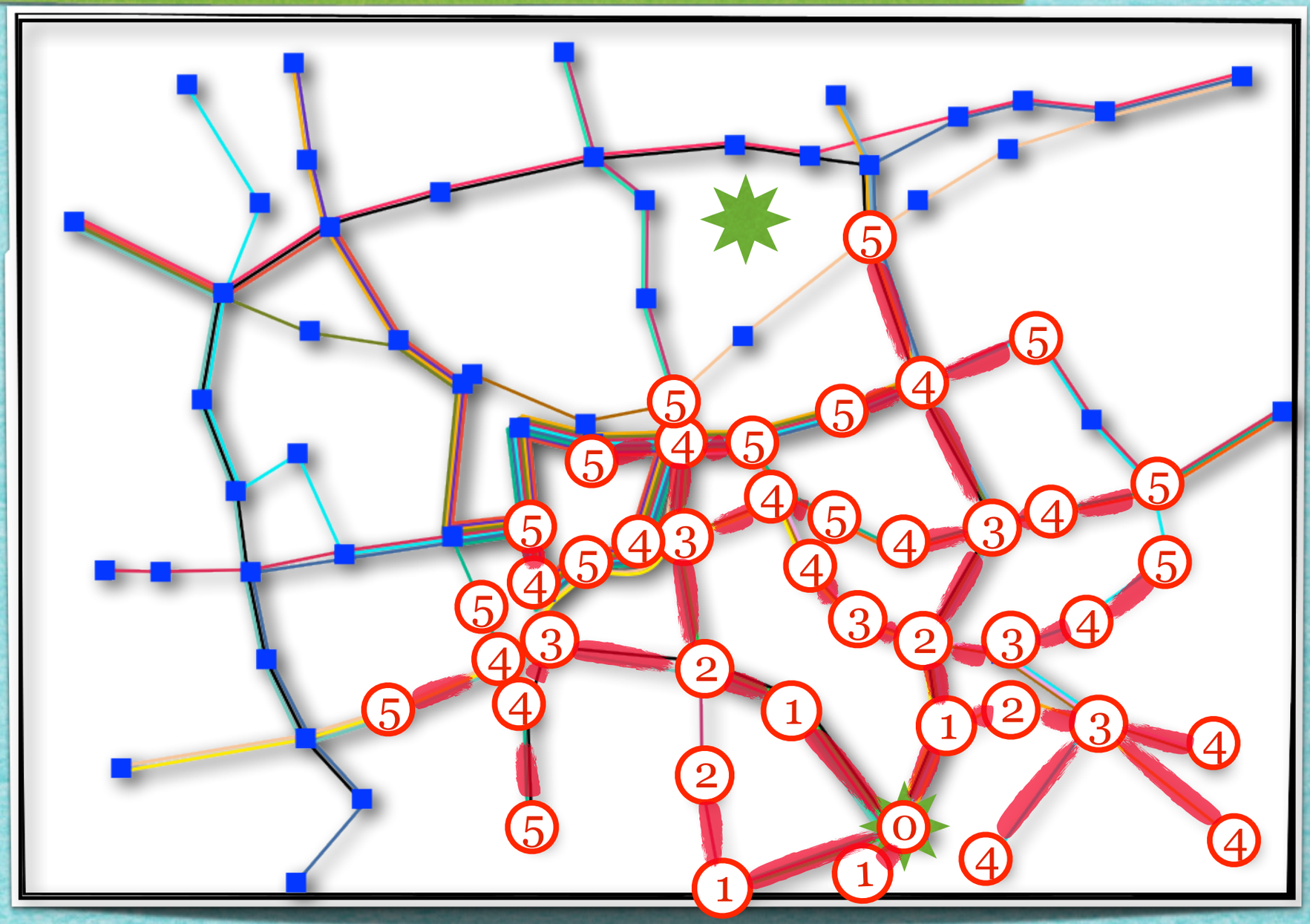
Wellenreiten in Graphen



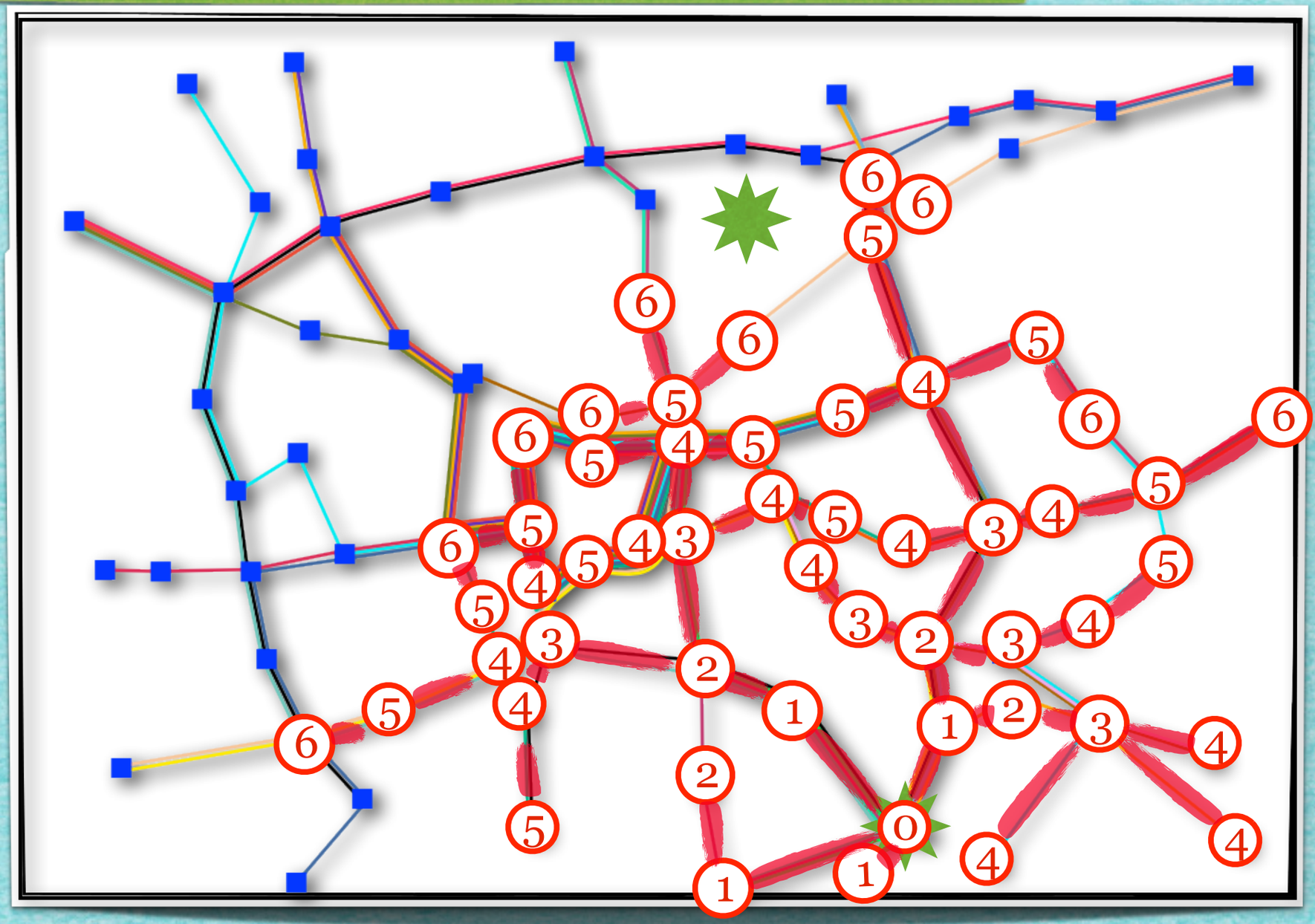
Wellenreiten in Graphen



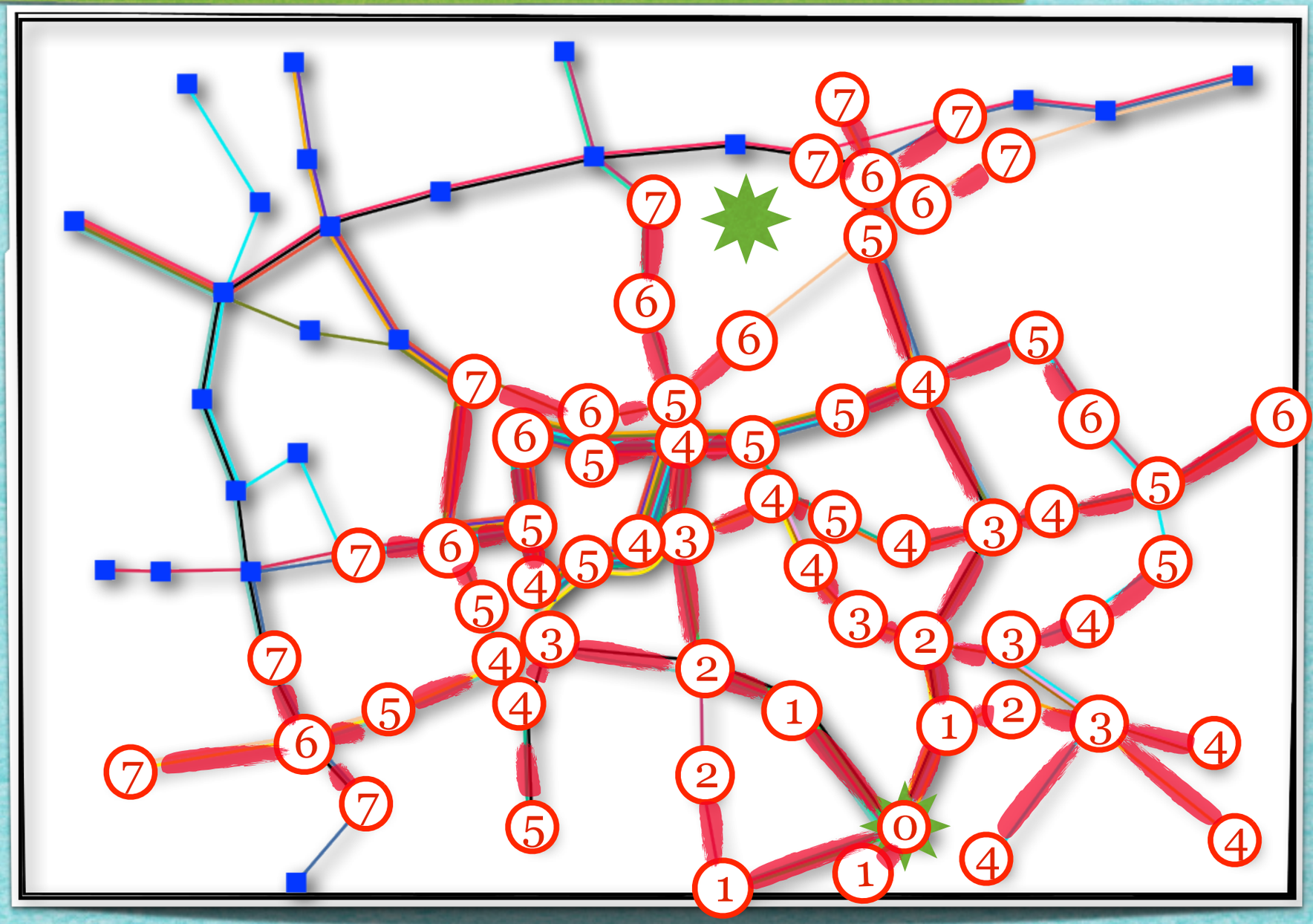
Wellenreiten in Graphen



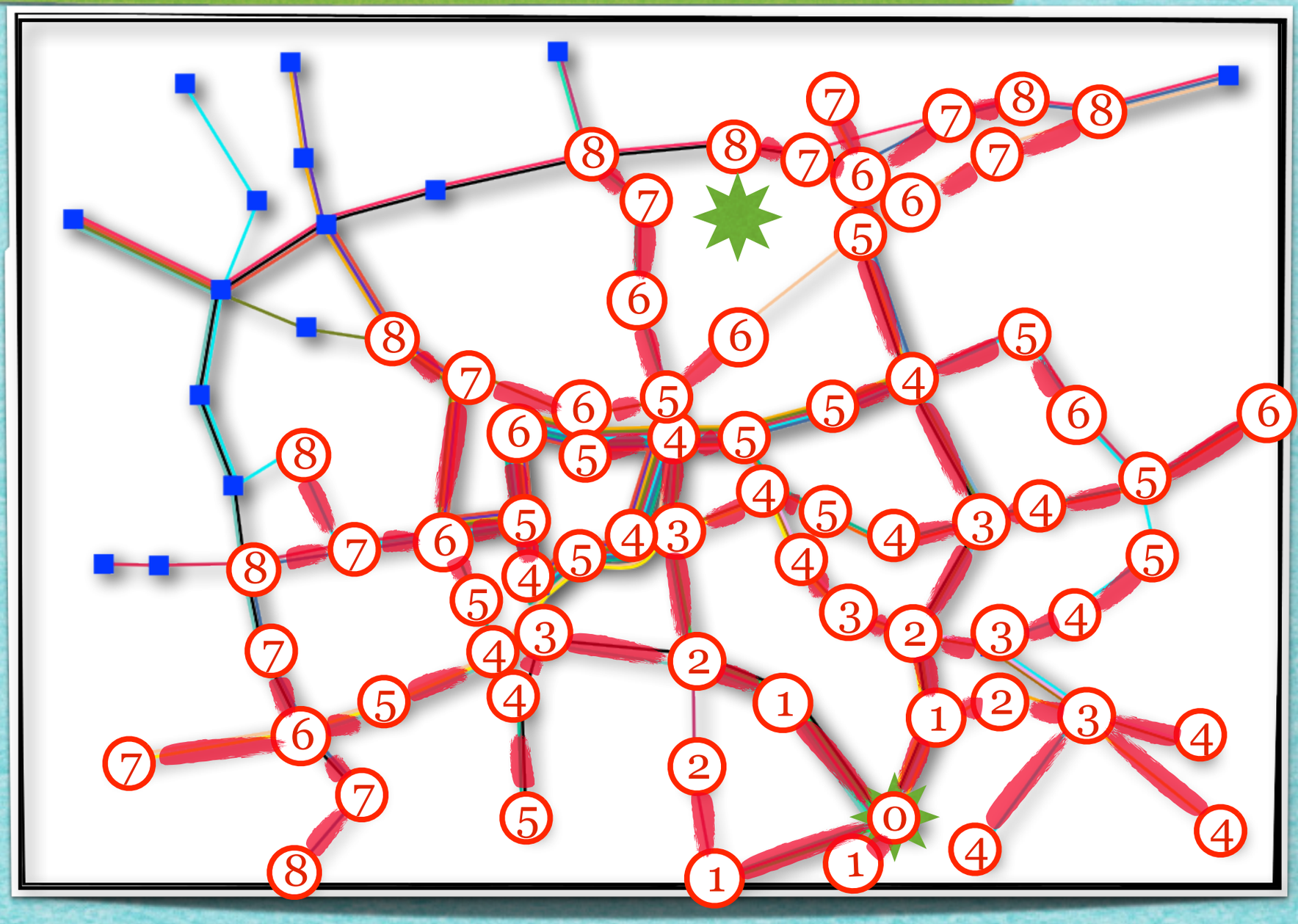
Wellenreiten in Graphen



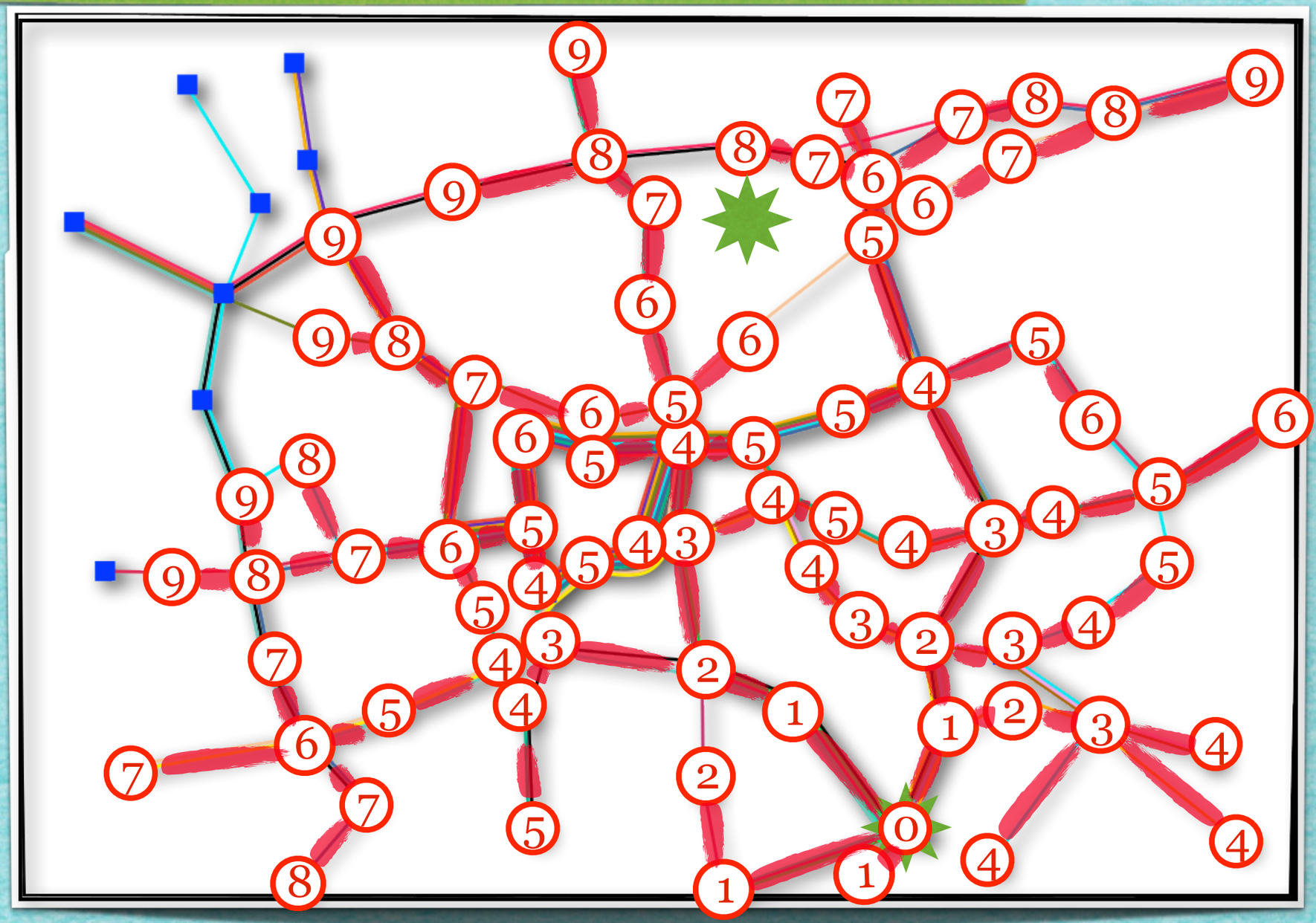
Wellenreiten in Graphen



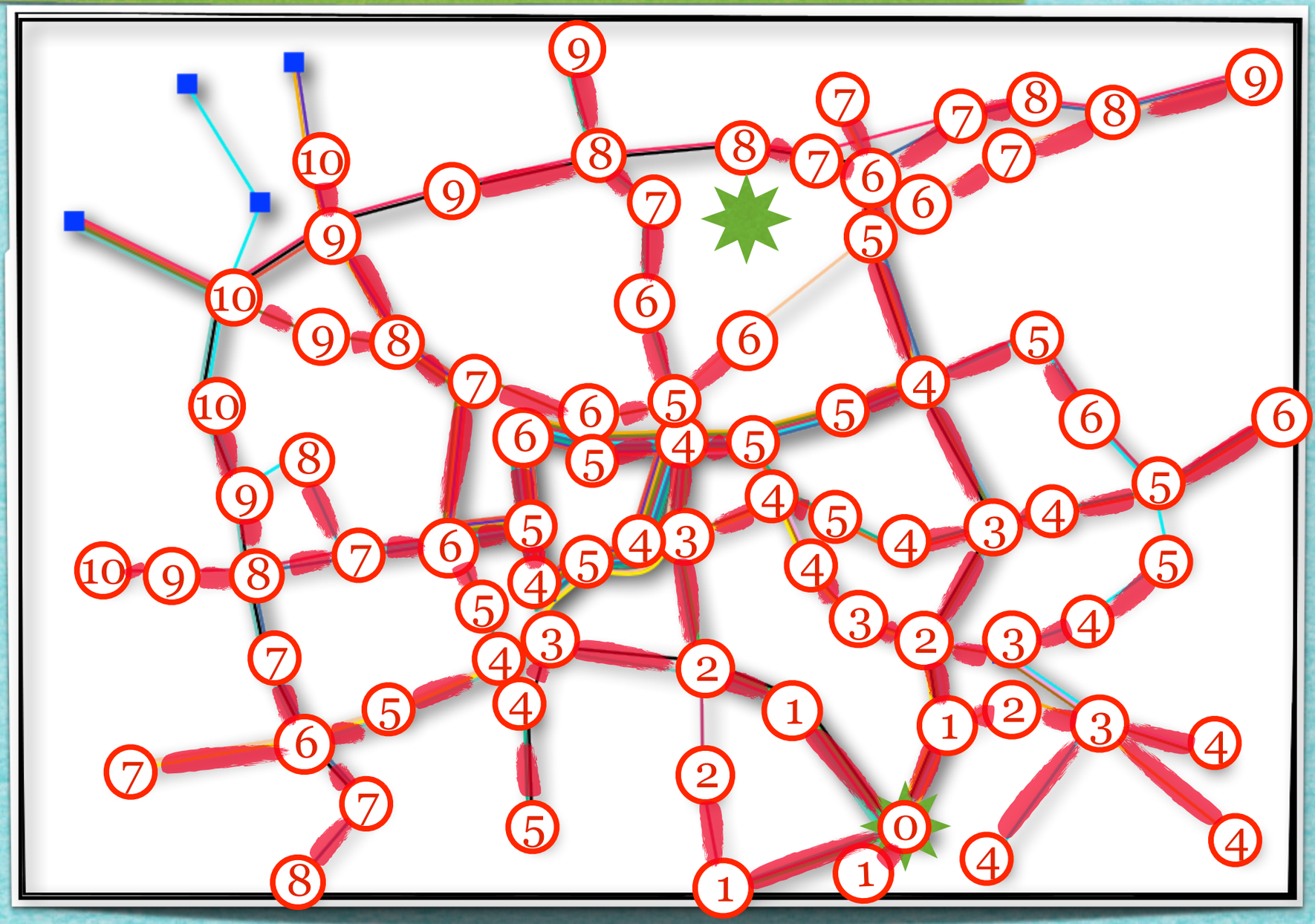
Wellenreiten in Graphen



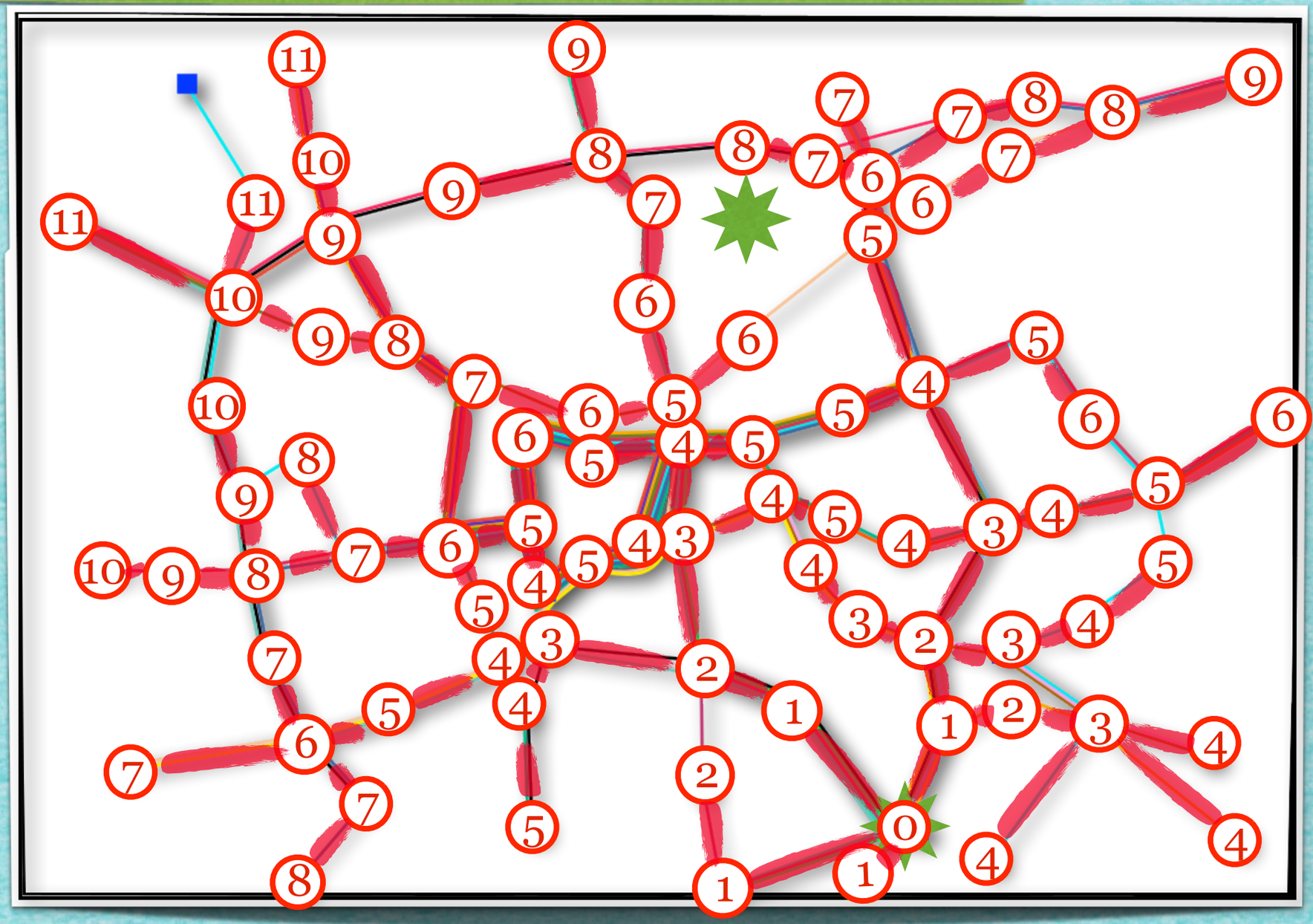
Wellenreiten in Graphen



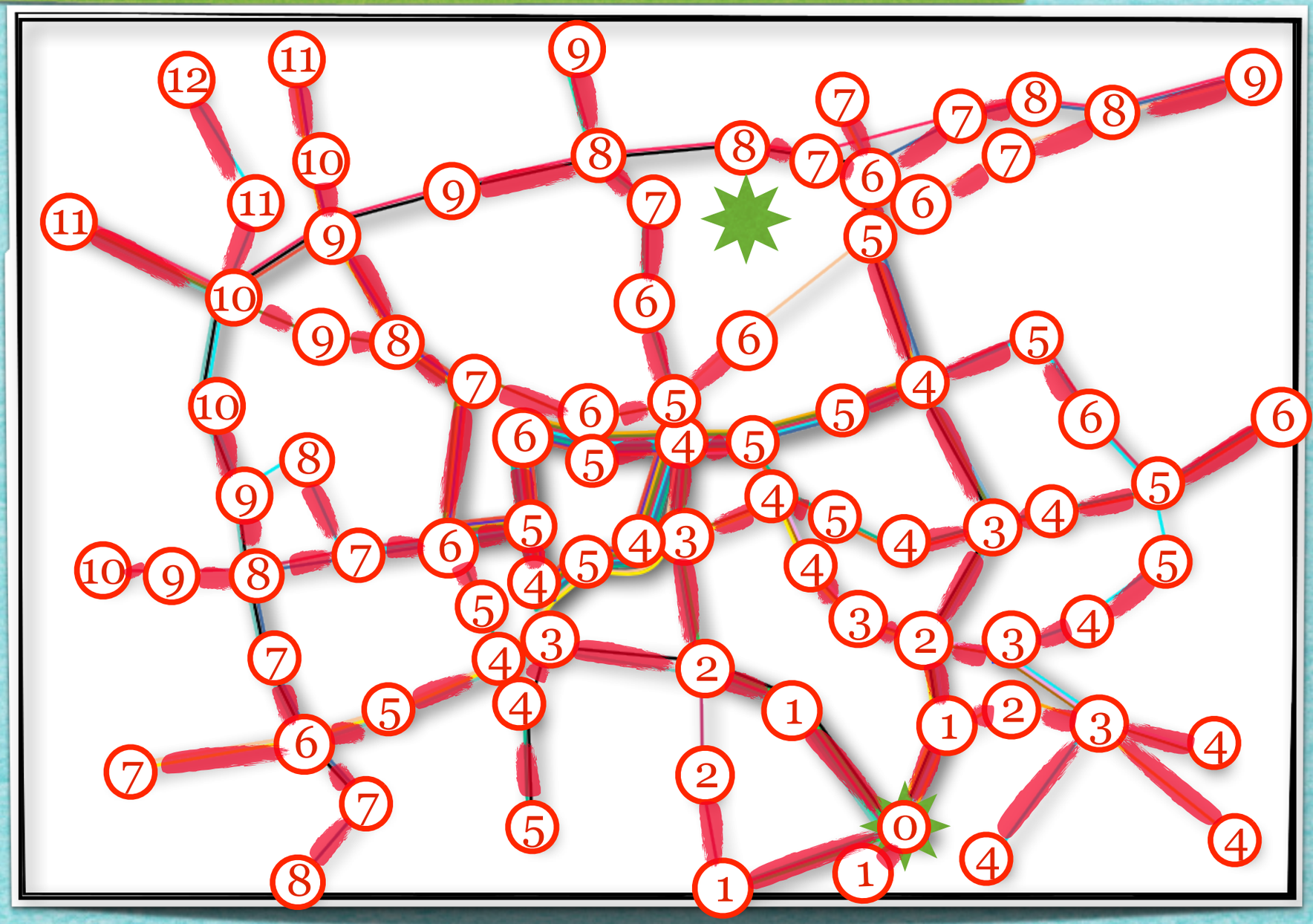
Wellenreiten in Graphen



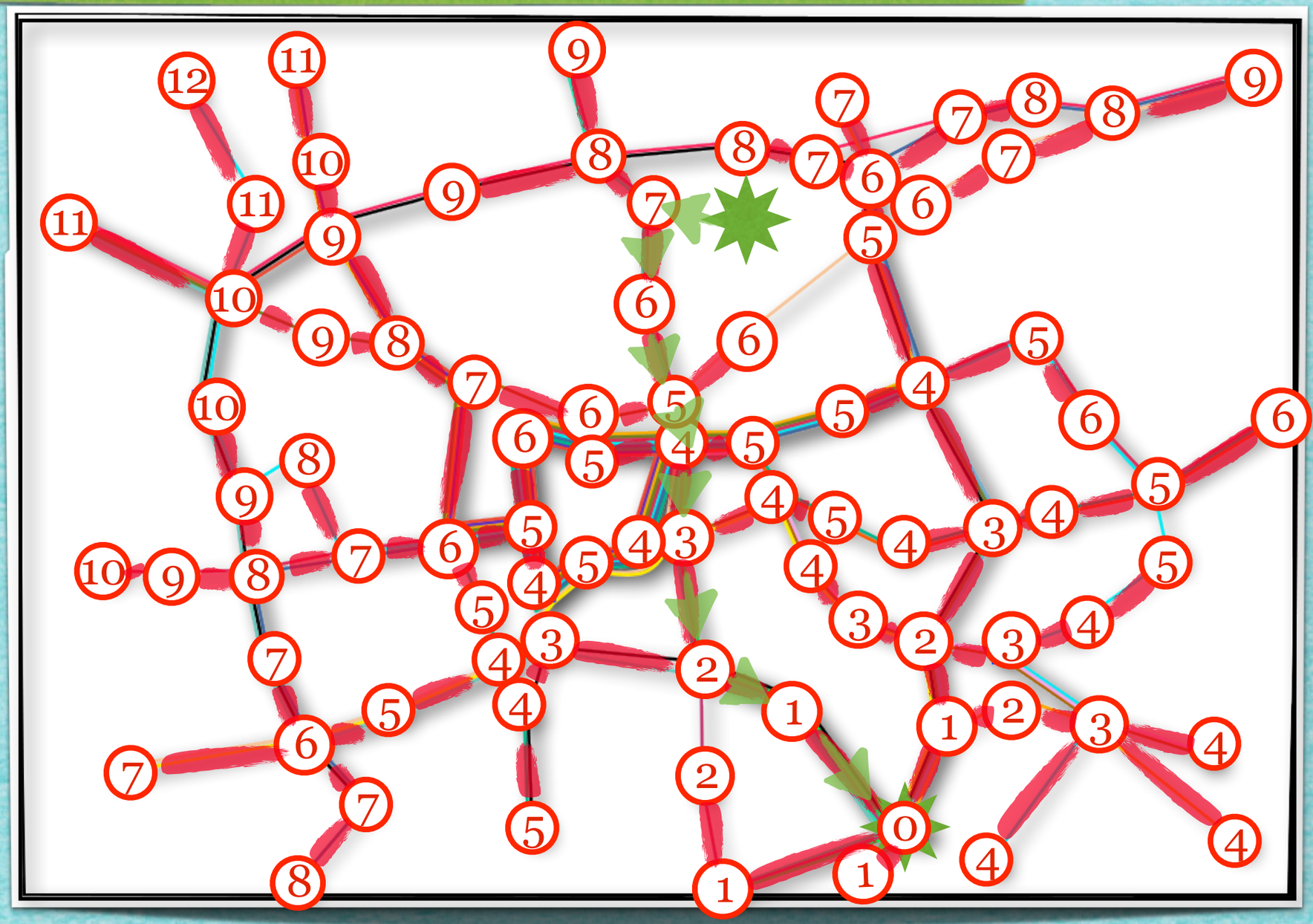
Wellenreiten in Graphen



Wellenreiten in Graphen



Wellenreiten in Graphen



Breitensuche

Algorithmus 3.17

INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge $Y \subseteq V$, die von s aus erreichbar ist,

für jeden Knoten $v \in Y$ die Länge l(v) eines kürzesten s-v-Weges,

Kantenmenge $T \subseteq E$, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅, l(s):=0
2. WHILE (R≠∅) DO {
   - 2.1. wähle Element $v \in R$
   - 2.2. IF (es gibt kein w V\Y mit $e=\{v,w\} \in E$) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1. wähle ein $w \in V\backslash R$ mit $e=\{v,w\} \in E$;
     - 2.3.2. setze R := R u {w}, Y := Y u {w}, T := T u {e};
     - 2.3.3. setze l(w):=l(v)+1
     - }

}

Satz 3.18

(1) Das Verfahren 3.17 ist endlich.

(2) Die Laufzeit ist O(n+m).

(3) Am Ende ist für jeden erreichbaren Knoten $v \in Y$ die Länge eines kürzesten Weges von s nach v im Baum (Y,T) durch l(v) gegeben.

(4) Am Ende ist für jeden erreichbaren Knoten $v \in Y$ die Länge eines kürzesten Weges von s nach v im Graphen (V,E) durch l(v) gegeben.

Beweis:

(1) **Wie für Algorithmus 3.7 gelten alle Eigenschaften. zusätzlich ist für jeden Knoten $v \in Y$ per Induktion, der Wert l(v) tatsächlich definiert.**

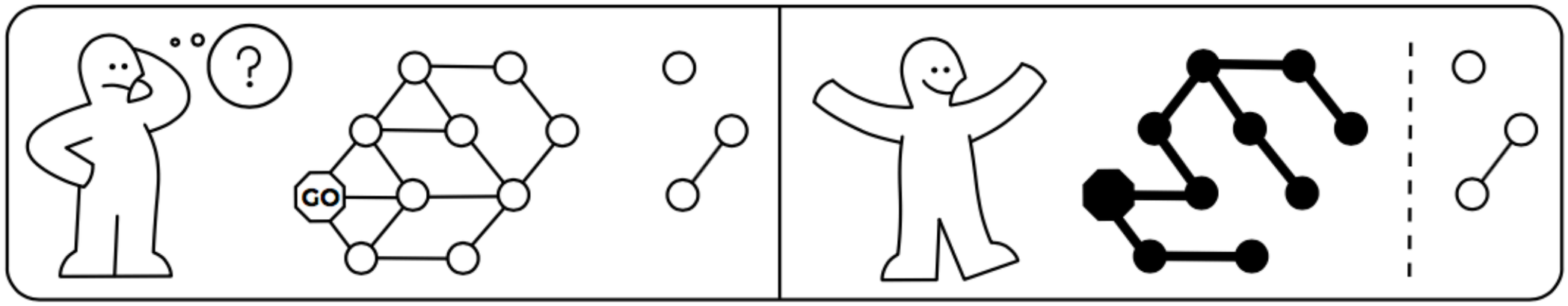
(2) **Die Laufzeit bleibt von Algorithmus 3.7 erhalten.**

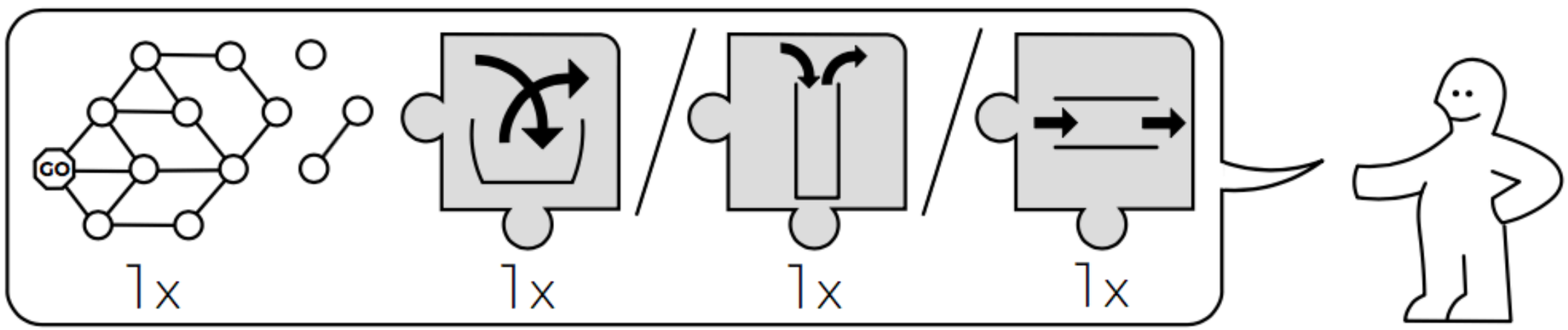
Mehr Details!

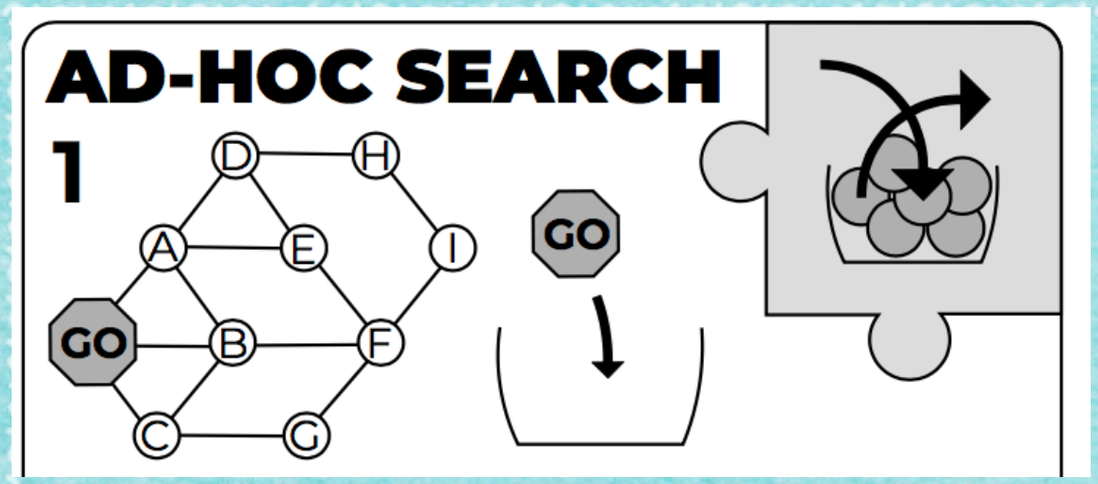
s.fekete@tu-bs.de

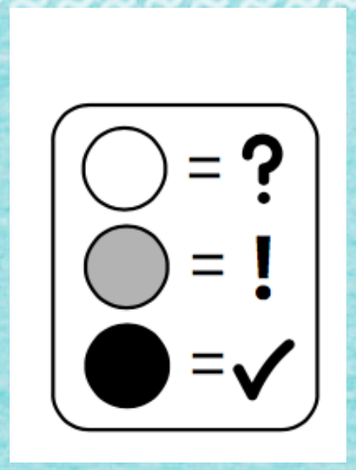
GRÅPH SCÄN

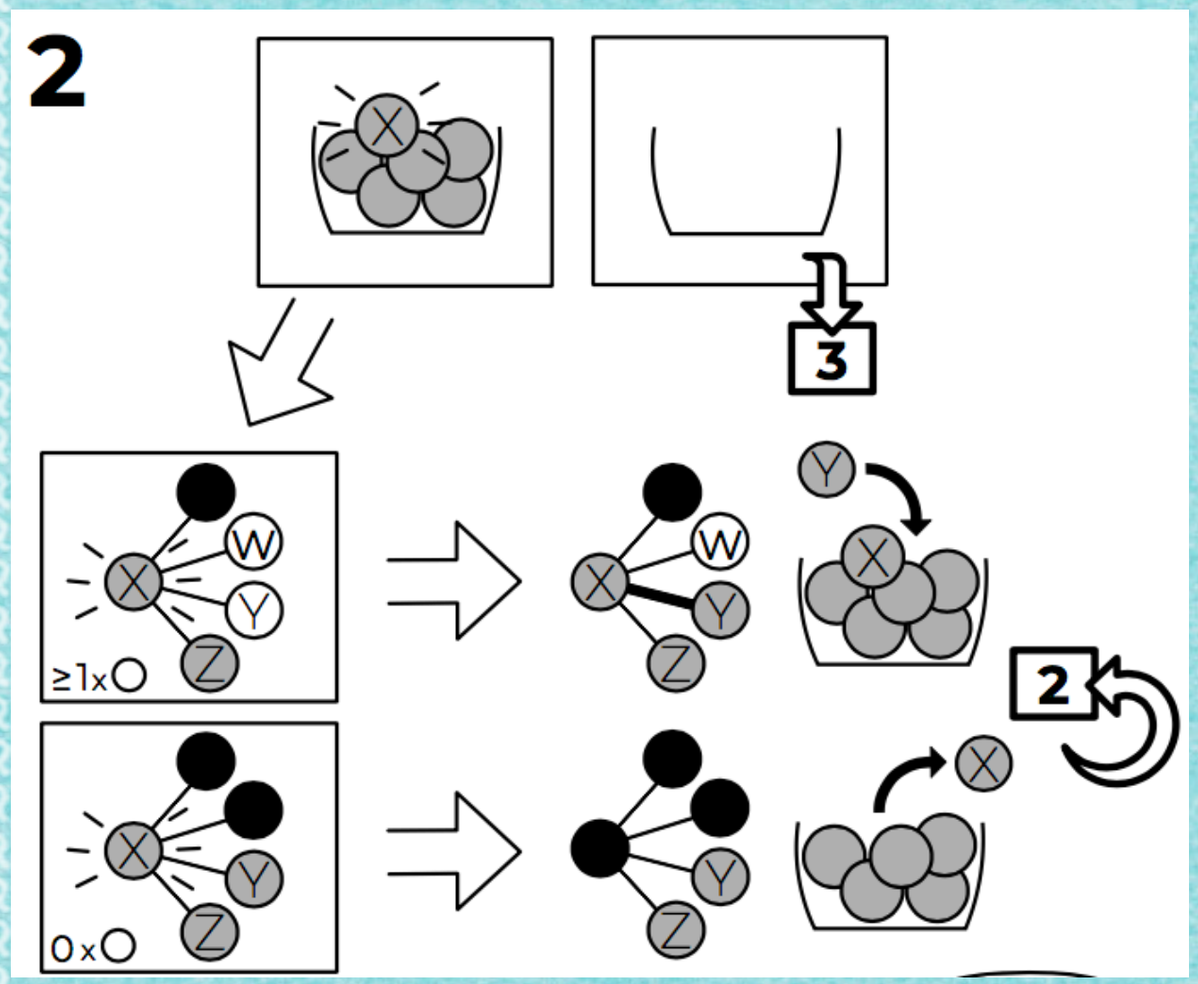
idea-instructions.com/graph-scan/
v1.0, CC by-nc-sa 4.0

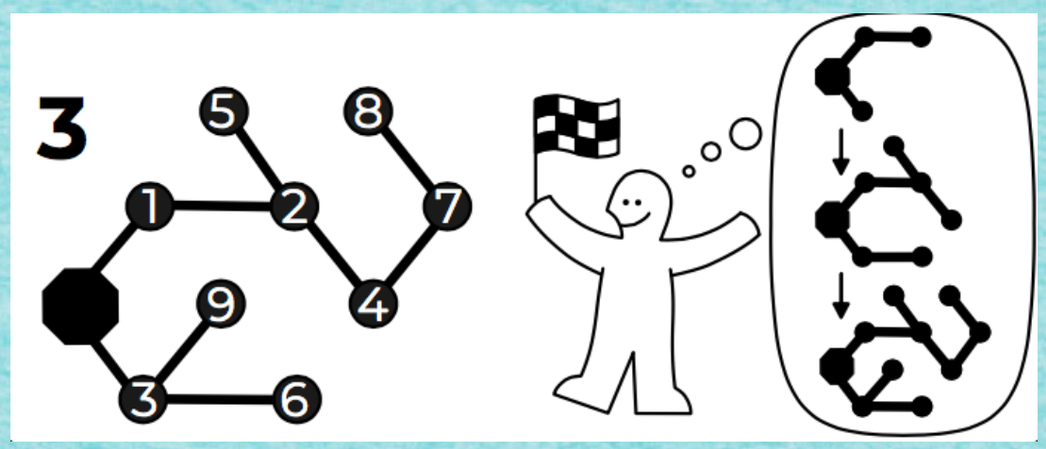




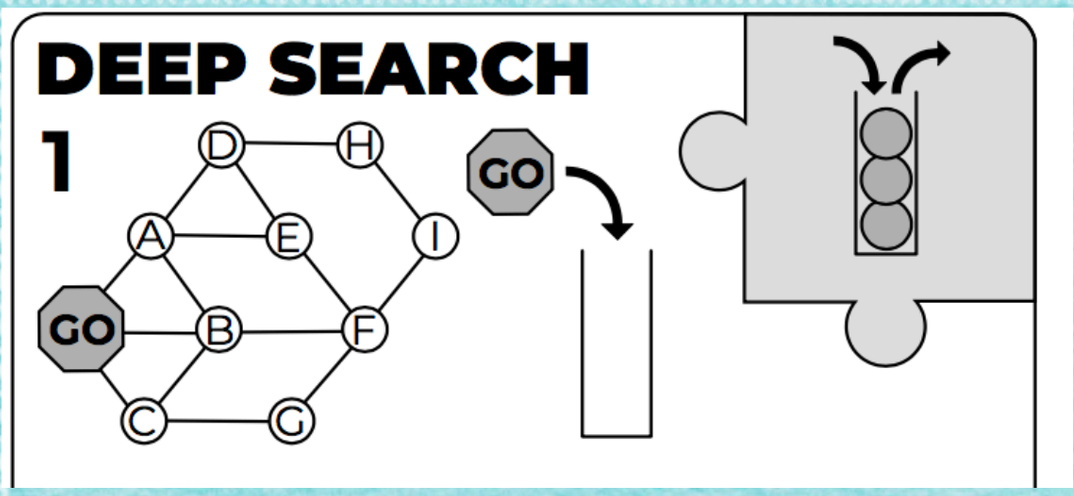

AD-HOC SEARCH
1
D
H
A
E
I
GO
B
F
C
G
GO

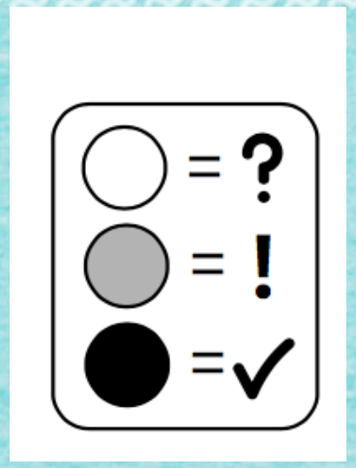

= ?
= !
=

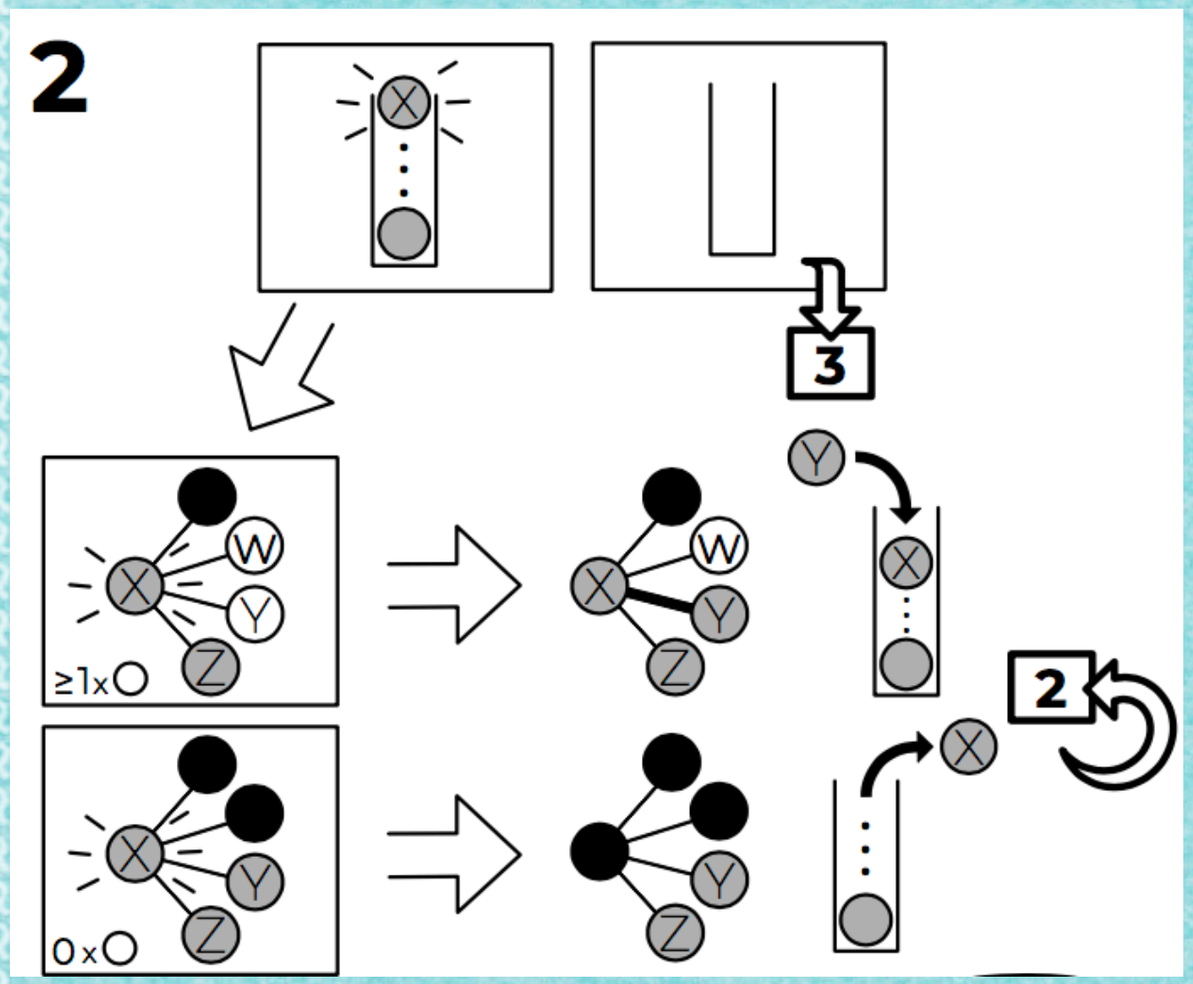

2
X
3
W
X
Y
Z
≥1xO
0xO
2

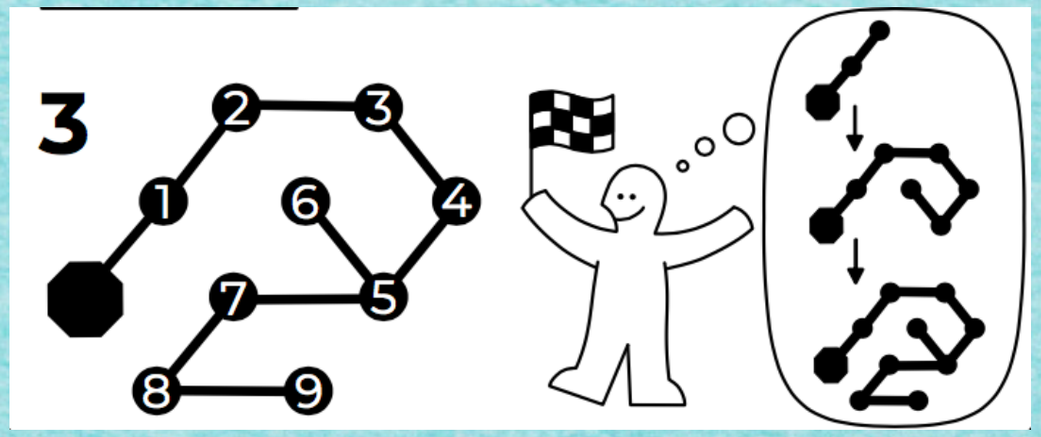

3
5
8
1
2
7
9
4
3
6

DEEP SEARCH





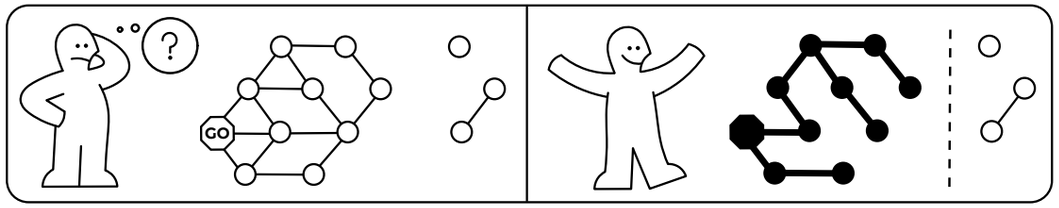


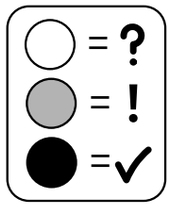


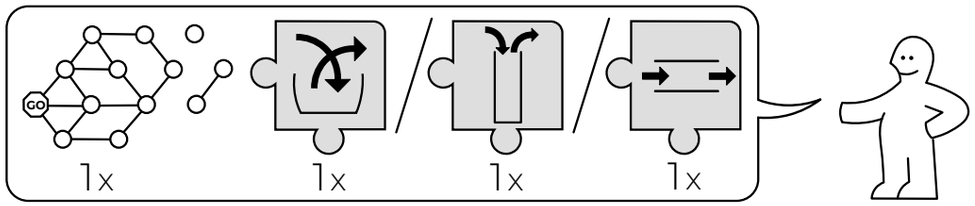
GRÅPH SKÄN

idea-instructions.com/graph-scan/
v1.3, CC by-nc-sa 4.0

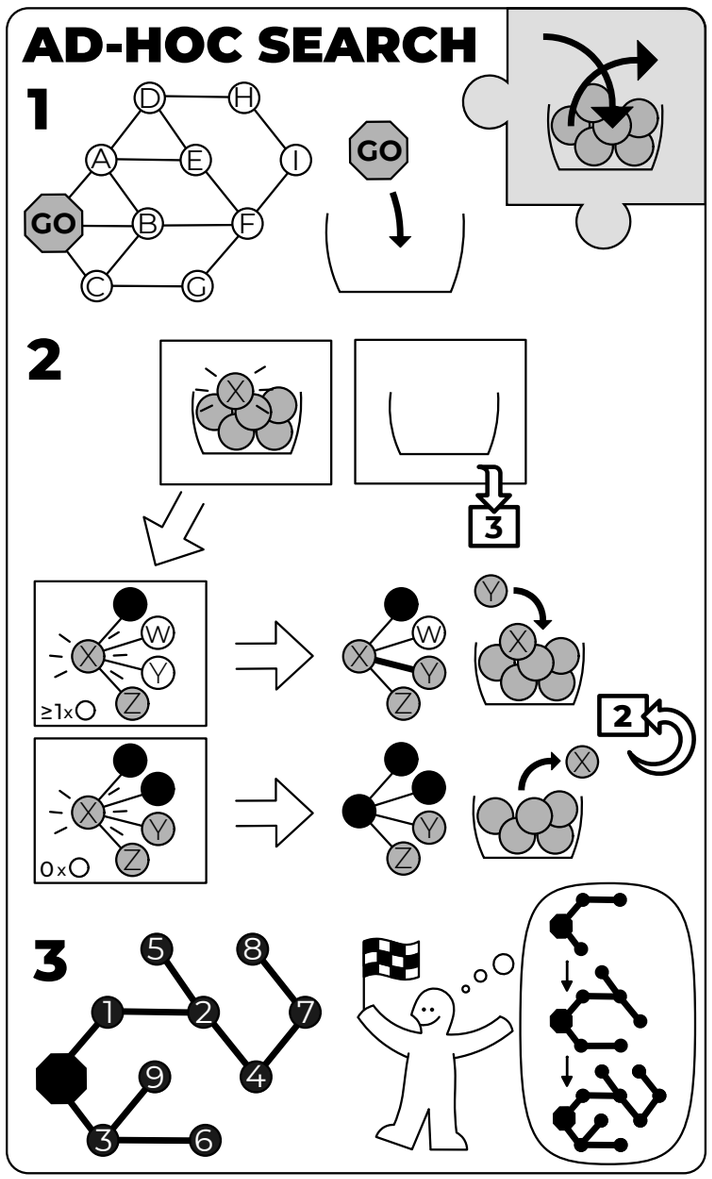
IDEA



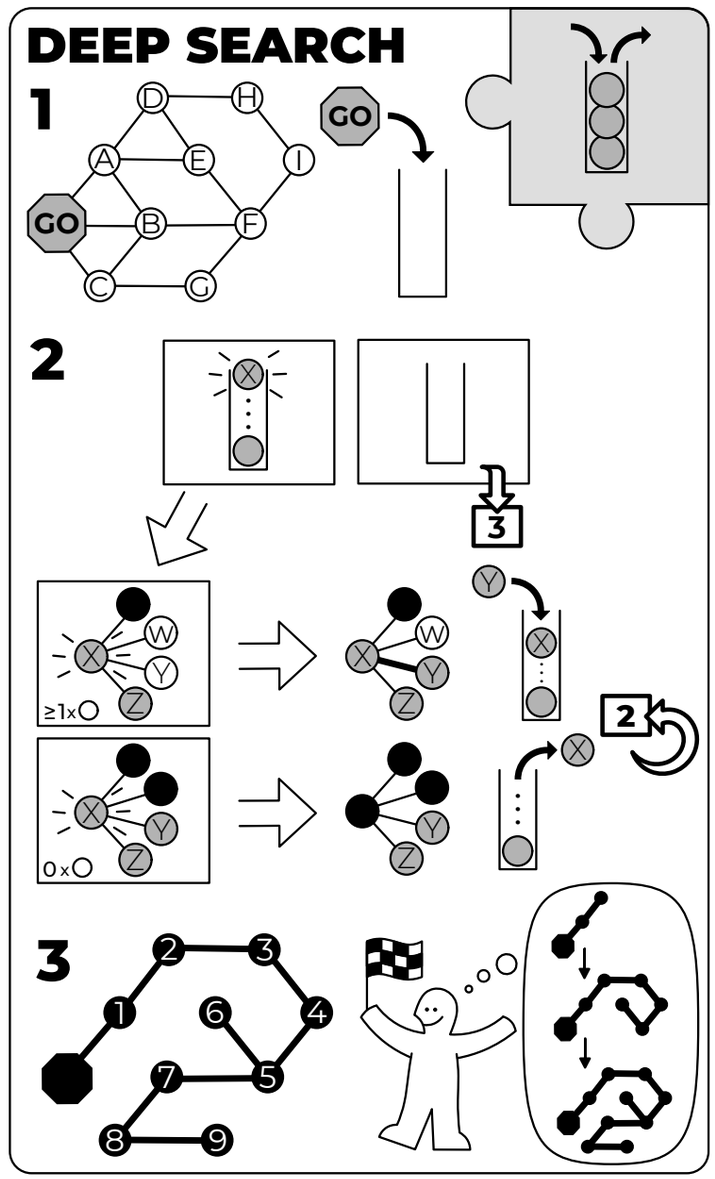




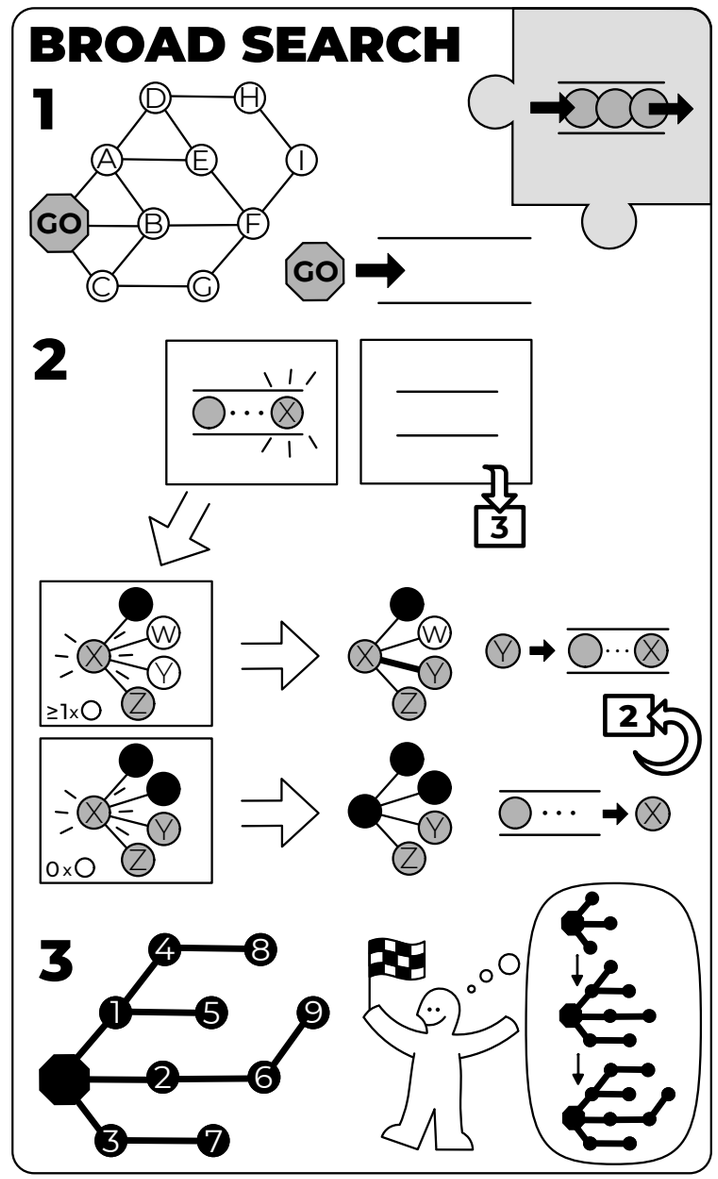
AD-HOC SEARCH



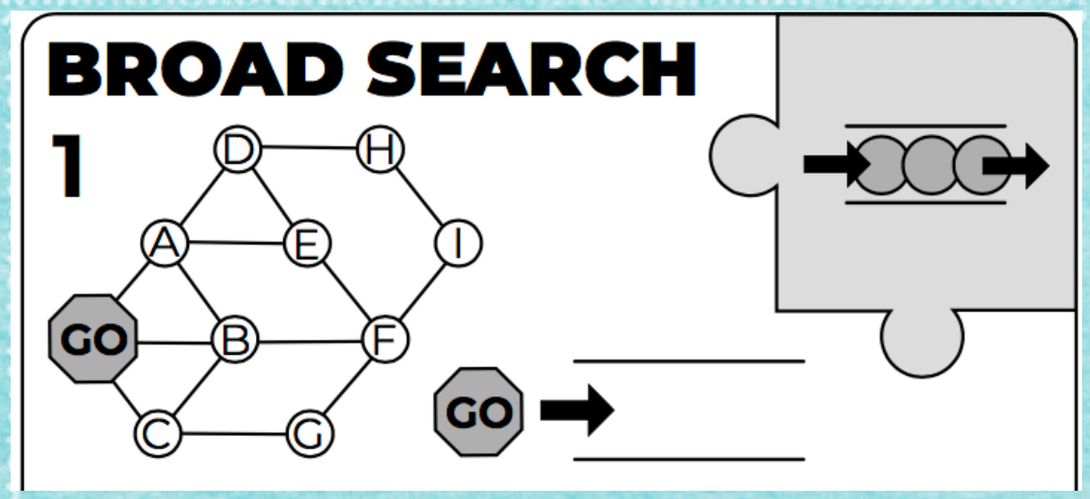
DEEP SEARCH

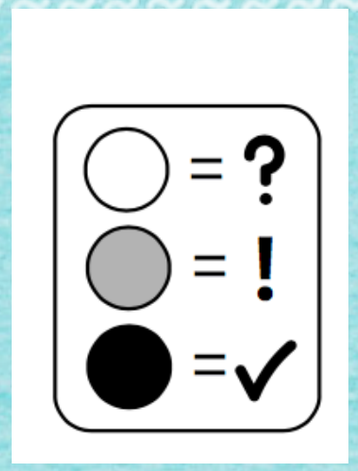


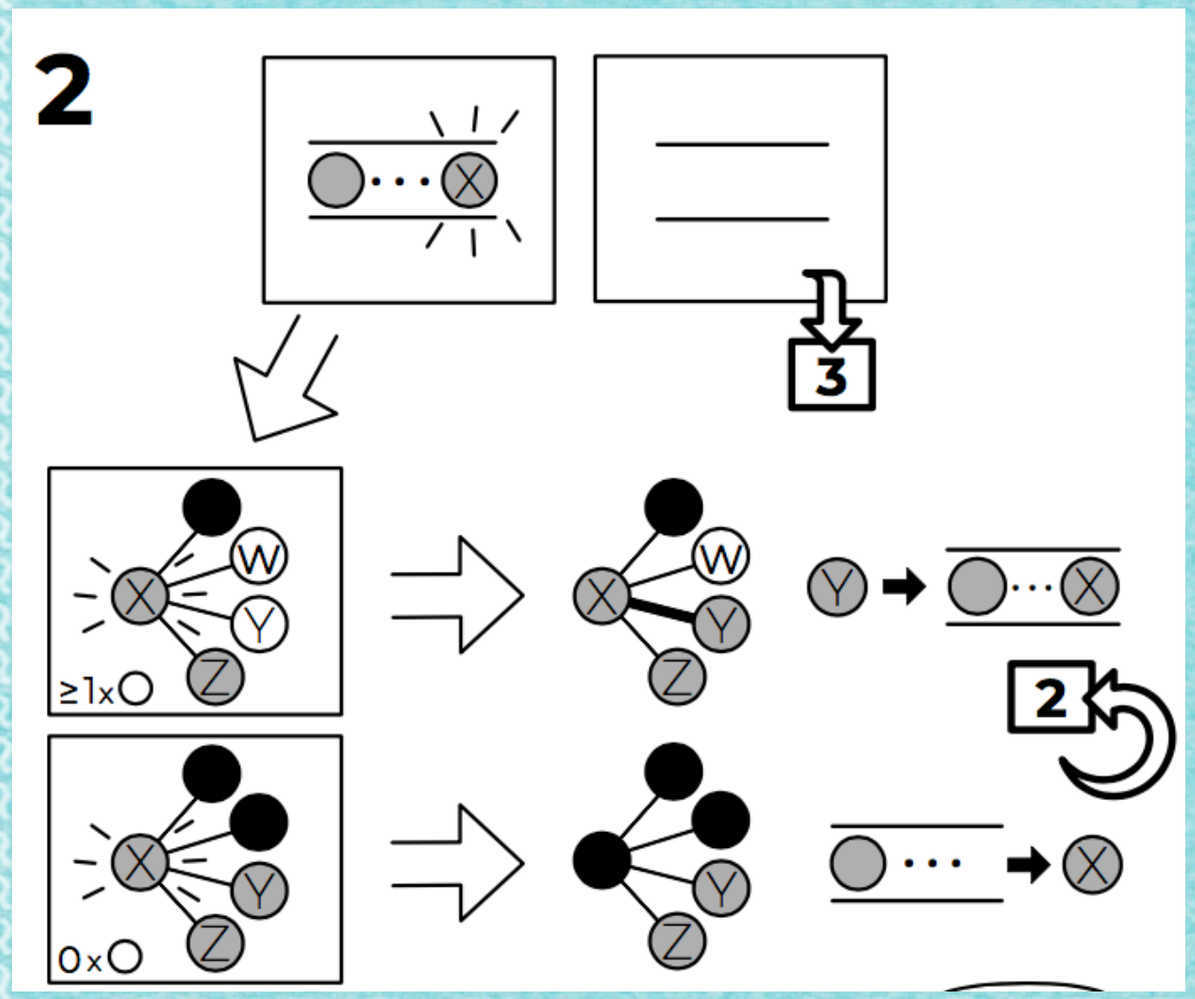
BROAD SEARCH

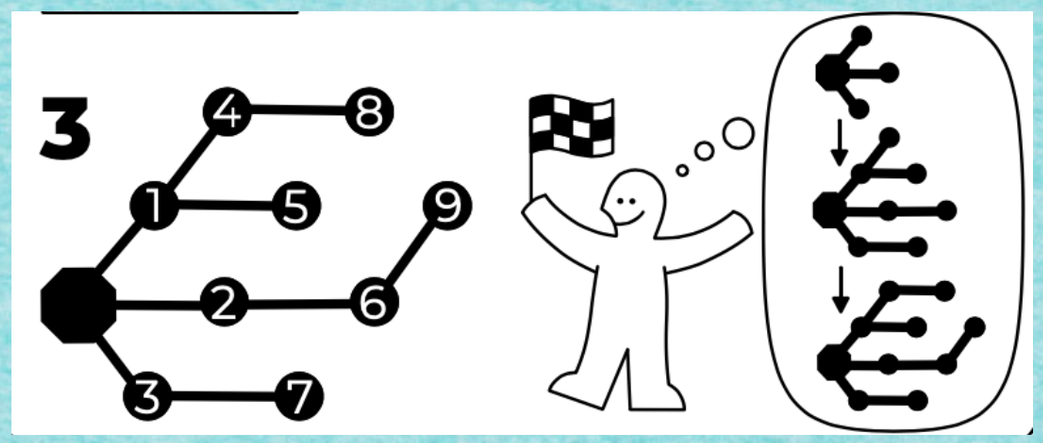


BROAD SEARCH





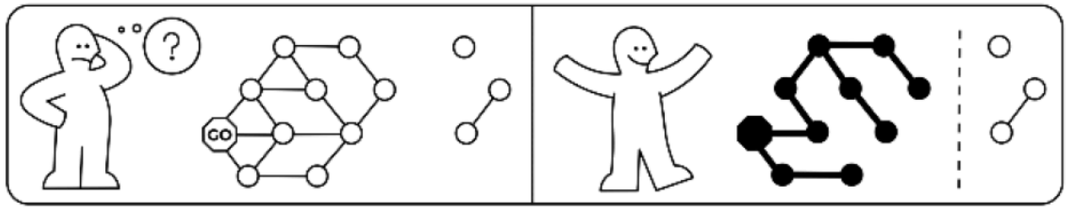


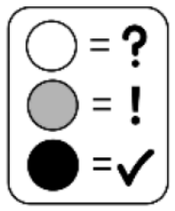


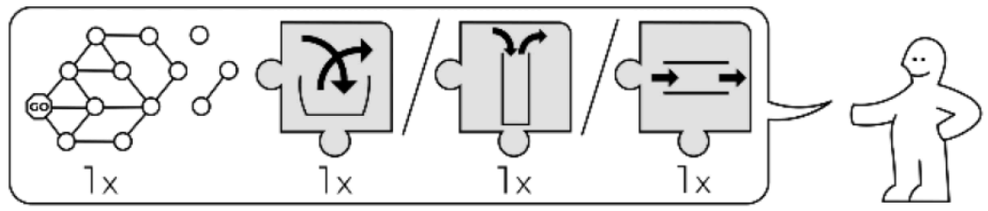
GRÅPH SKÄN

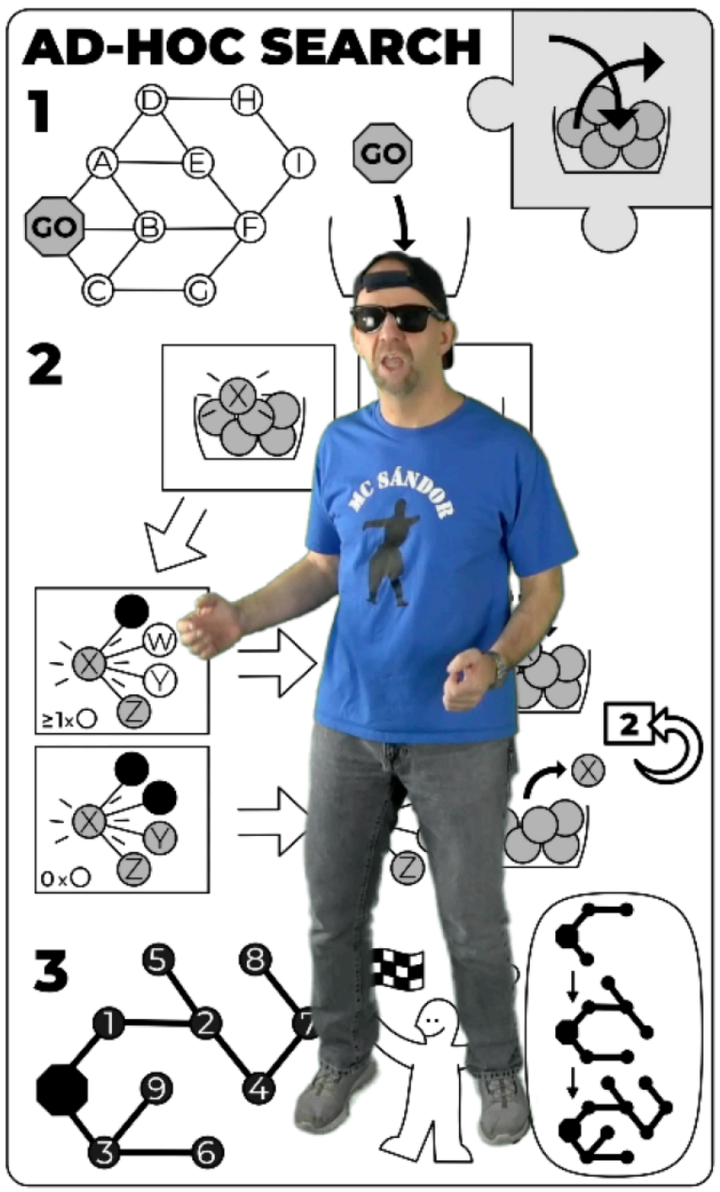
idea-instructions.com/graph-scan/
v1.3, CC by-nc-sa 4.0

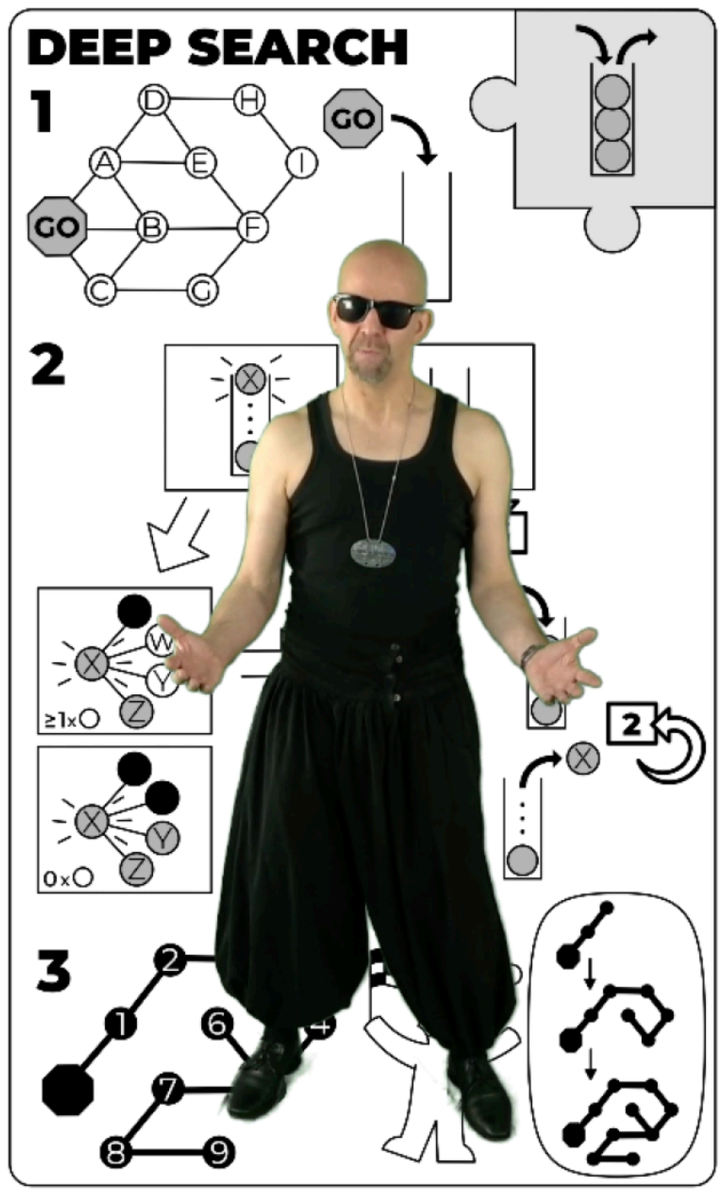
IDEA

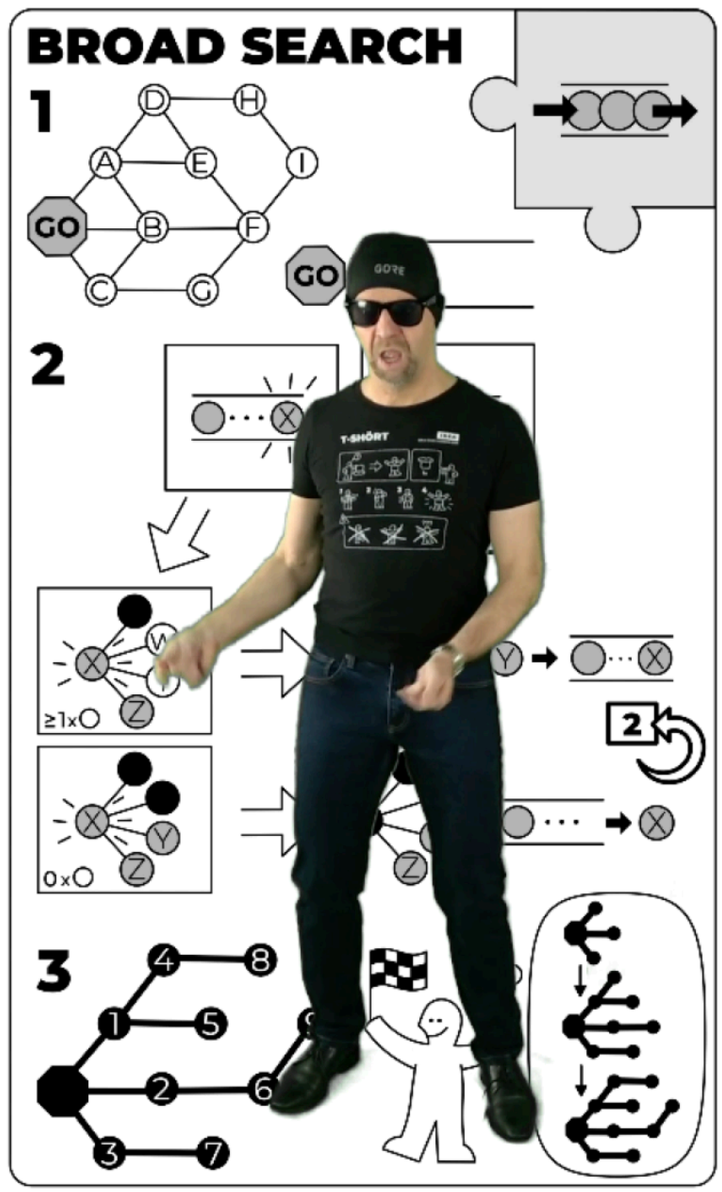






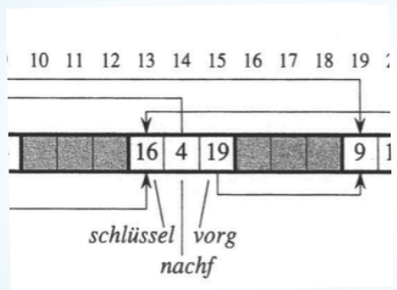


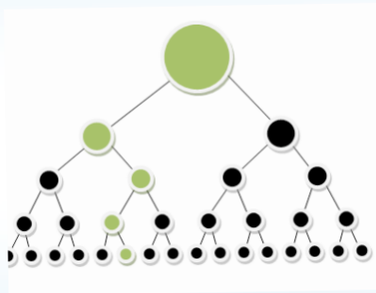




Kapitelende!

s.fekete@tu-bs.de






Kapitel 4: Dynamische Datenstrukturen

Algorithmen und Datenstrukturen
WS 2024/25

Prof. Dr. Sándor Fekete

Wie verwalten wir dynamische Mengen von Objekten?

Waschkorb

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

Im Folgenden:

S Menge von Objekten

k Wert eines Elements (“Schlüssel”)

x Zeiger auf Element

NIL spezieller, “leerer” Zeiger

<u>SEARCH(S,k):</u> **“Suche in S nach k”**

Durchsuche die Menge S nach einem Element von Wert k.

Ausgabe: Zeiger x, falls x existent
NIL, falls kein Element Wert k hat.

4.1 Grundoperationen

<u>INSERT(S,x):</u> **“Füge x in S ein”**

Erweitere S um das Element, das unter der Adresse x steht.

4.1 Grundoperationen

DELETE(S,x): **“Entferne x aus S”**

Lösche das unter der Adresse x stehende Element aus der Menge S.

4.1 Grundoperationen

MINIMUM(S): **“Suche das Minimum in S”**

Finde in S ein Element von kleinstem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

Ausgabe: Zeiger x auf solch ein Element

MAXIMUM(S): **"Suche das Maximum in S"**

Finde in S ein Element von größtem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

Ausgabe: Zeiger x auf solch ein Element

PREDECESSOR(S,x):
“Finde das nächstkleinere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstkleinerem Wert in S.

Ausgabe: Zeiger y auf Element
NIL, falls x Minimum von S angibt

<u>SUCCESSOR(S,x):</u>
“Finde das nächstgrößere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstgrößerem Wert in S.

Ausgabe: Zeiger y auf Element
NIL, falls x Maximum von S angibt

4.1 Grundoperationen

Wie nimmt man das vor?

Wie lange dauert das,
in Abhängigkeit von der Größe von S?

Unsortierte Unterlagen:
Immer alles durchgehen, also: **O(n)**

Sortierte Unterlagen: Geht schneller!

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

  Immer gleich schnell, egal wie groß S ist.

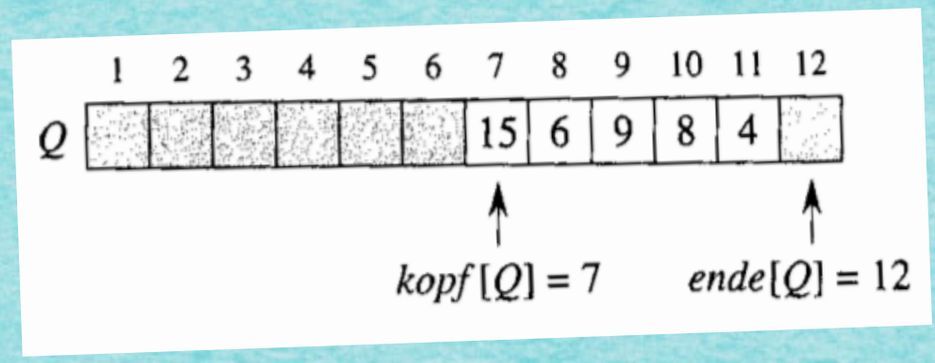
Schnell:

- ***O(log n):*** ***logarithmische Zeit***

  Wiederholtes Halbieren

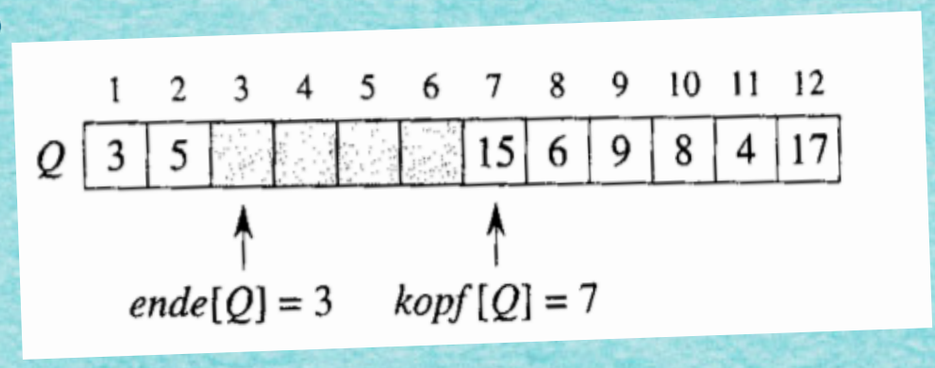
4.2 Stapel und Warteschlange

OUT

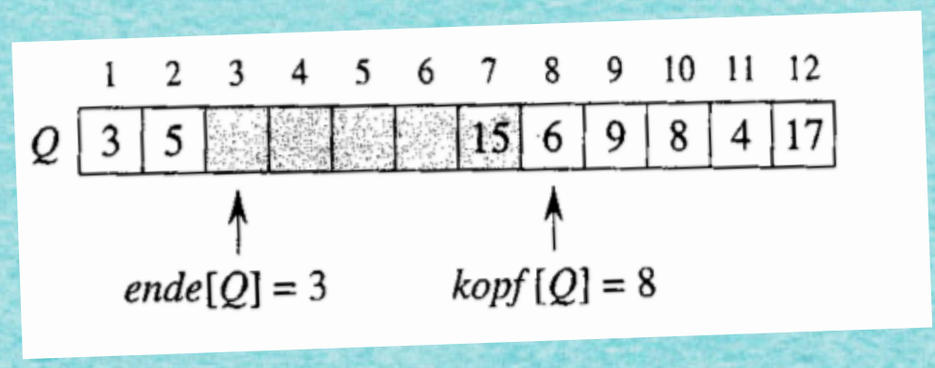


IN

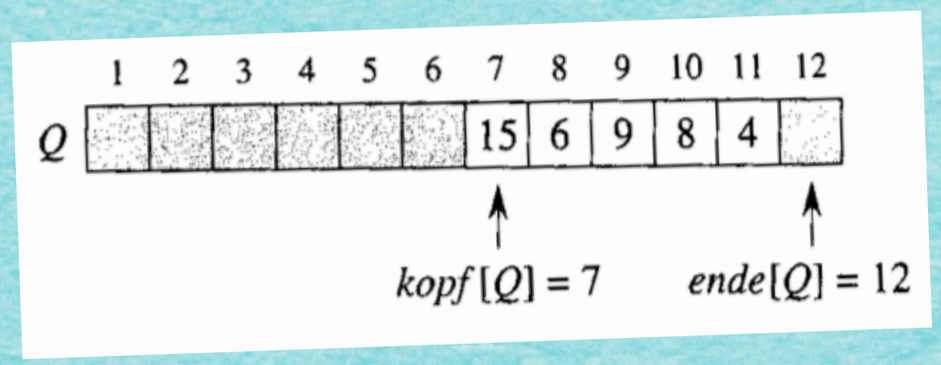
Enqueue: 17, 3, 5

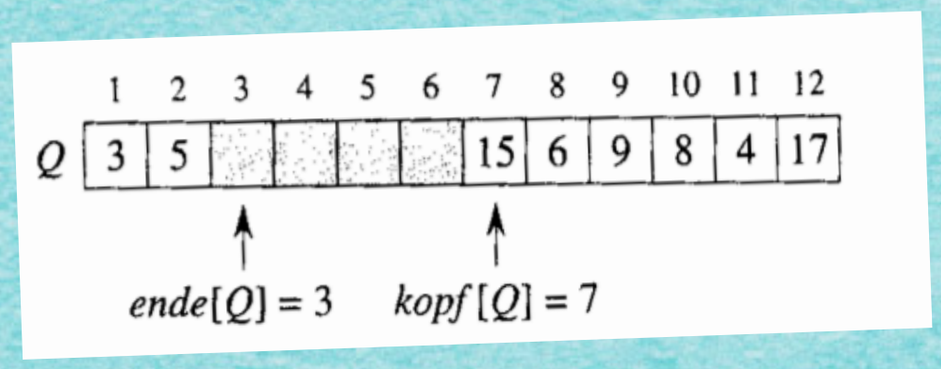


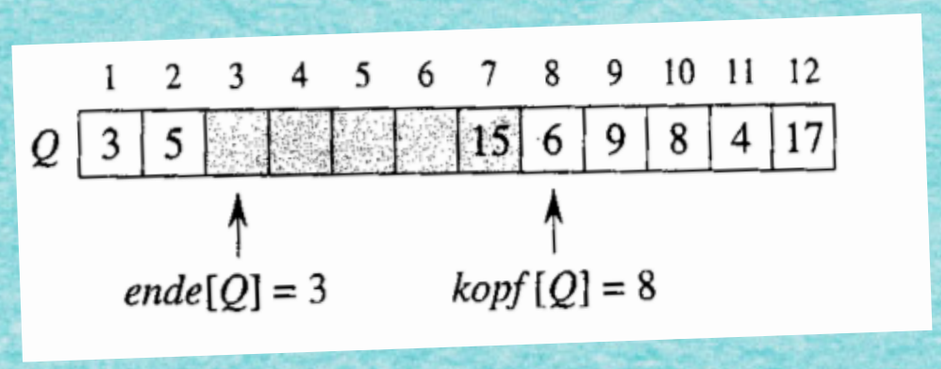
Dequeue:



Warteschlange auf Array umgesetzt



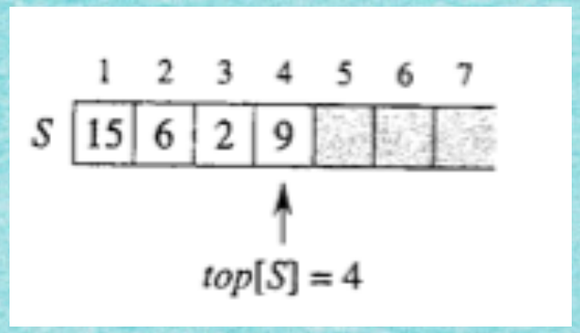


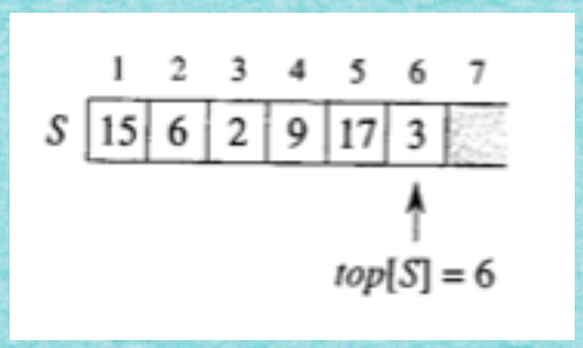


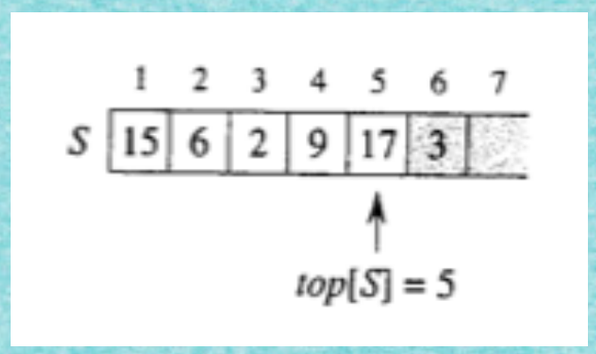

```
ENQUEUE(Q, x)
1  Q[ende[Q]] ← x
2  if ende[Q] = länge[Q]
3     then ende[Q] ← 1
4     else ende[Q] ← ende[Q] + 1
```

```
DEQUEUE(Q)
1  x ← Q[kopf[Q]]
2  if kopf[Q] = länge[Q]
3     then kopf[Q] ← 1
4     else kopf[Q] ← kopf[Q] + 1
5  return x
```

Stack
auf Array umgesetzt

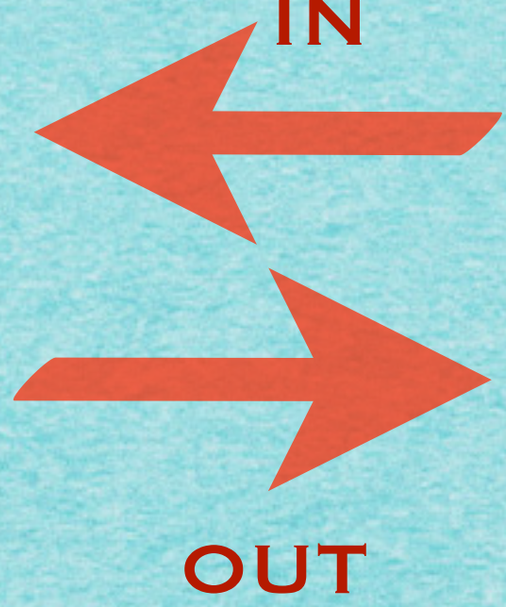




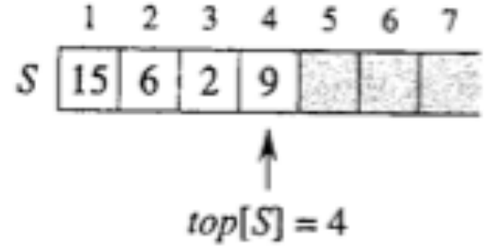


Push: 17, 3

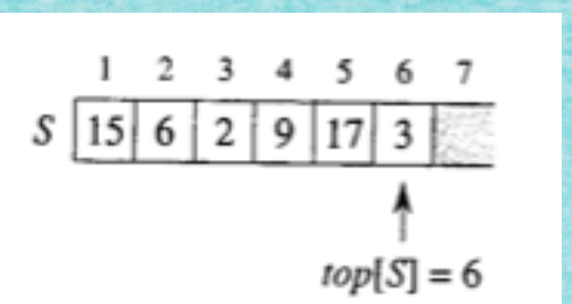
Pop



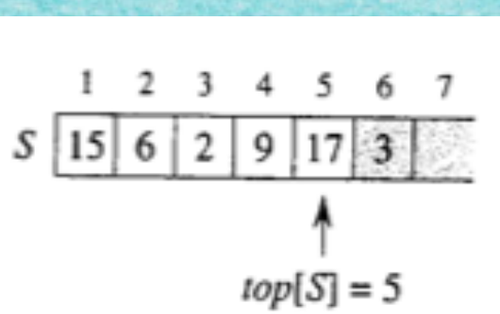
Stack auf Array umgesetzt



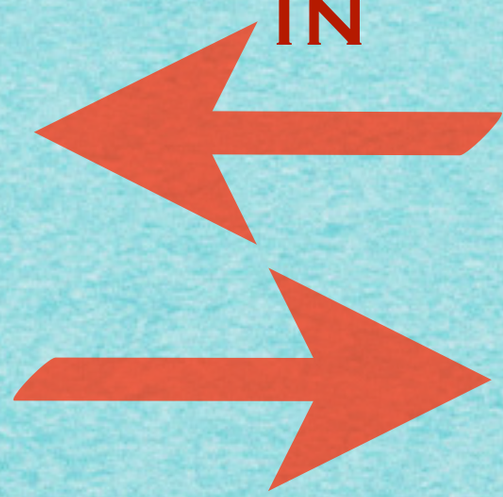

```
STACK-EMPTY(S)
1  if top[S] = 0
2       then return WAHR
3       else  return FALSCH
```




```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```




```
POP(S)
1  if STACK-EMPTY(S)
2       then error "Unterlauf"
3       else  top[S] ← top[S] − 1
4             return S[top[S] + 1]
```

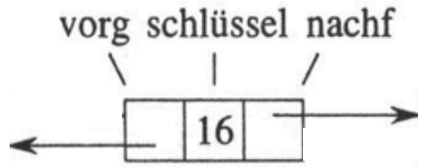


4.3 Verkettete Listen

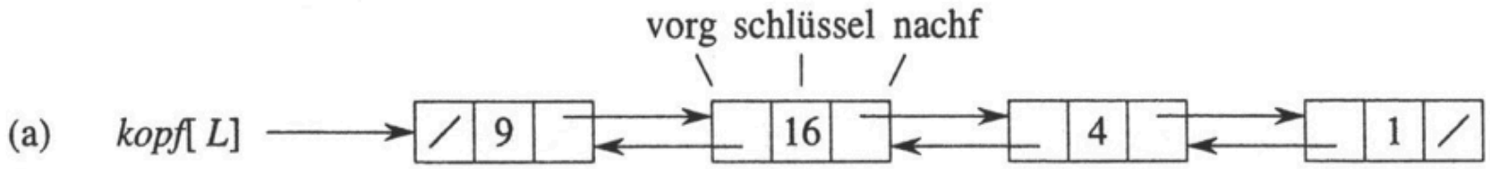
Idee:

Ordne Objekte nicht explizit in aufeinanderfolgenden Speicherzellen an, sondern gib jeweils Vorgänger und Nachfolger an.

Struktur einer doppelt verketteten Liste

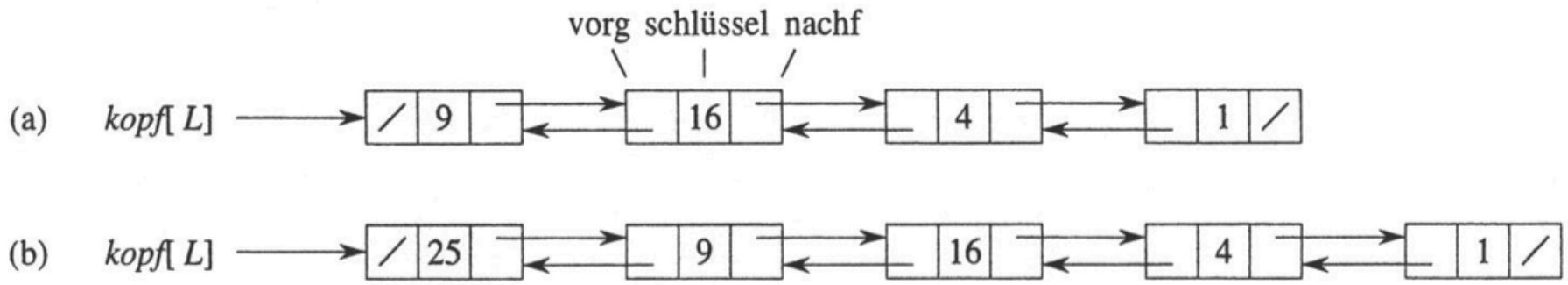


Struktur einer doppelt verketteten Liste



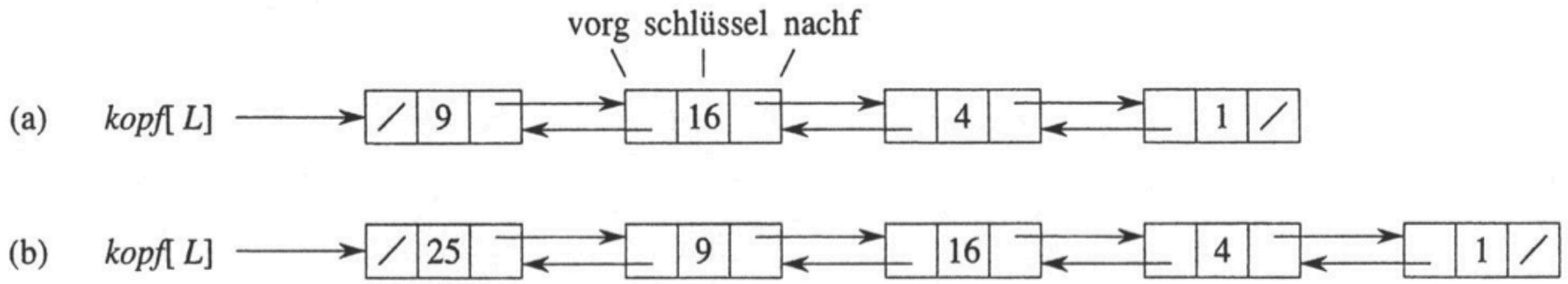
- **Füge vorne das Element mit Schlüssel 25 ein.**

Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

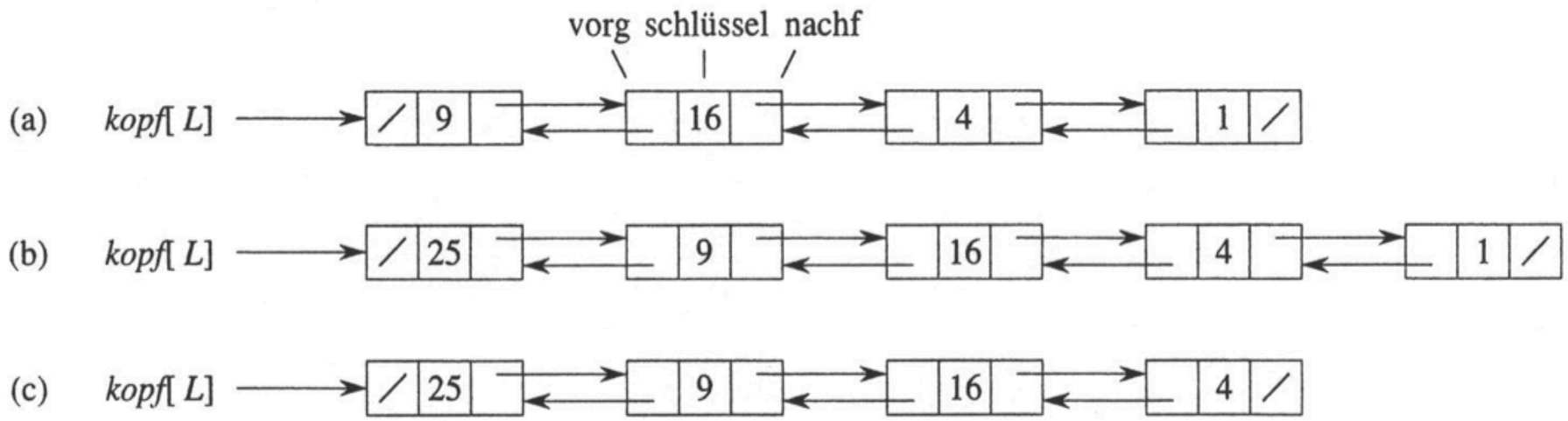
Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

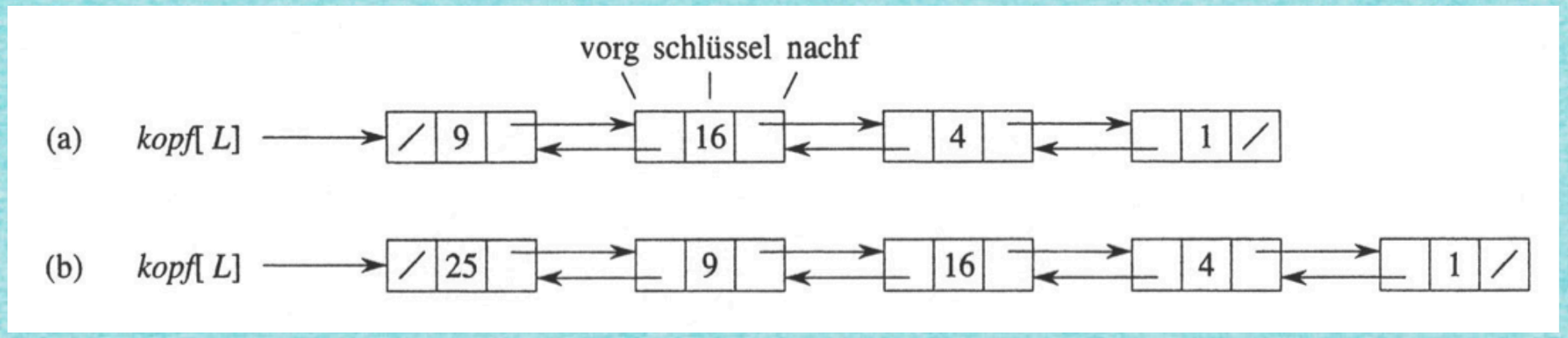
Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

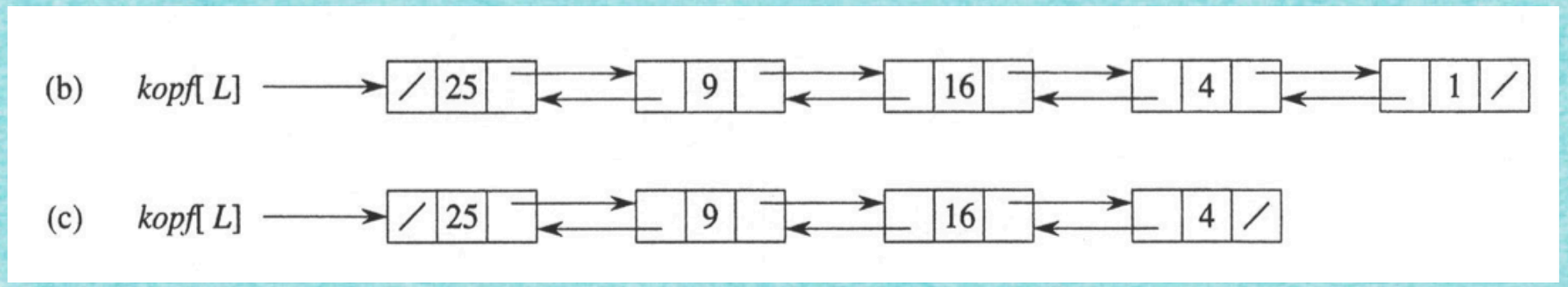
Einfügen in eine doppelt verkettete Liste




```
LIST-INSERT(L, x)
1  nachf[x] ← kopf[L]
2  if kopf[L] ≠ NIL
3      then vorg[kopf[L]] ← x
4  kopf[L] ← x
5  vorg[x] ← NIL
```

<u>Laufzeit:</u> O(1)

Löschen aus einer doppelt verketteten Liste



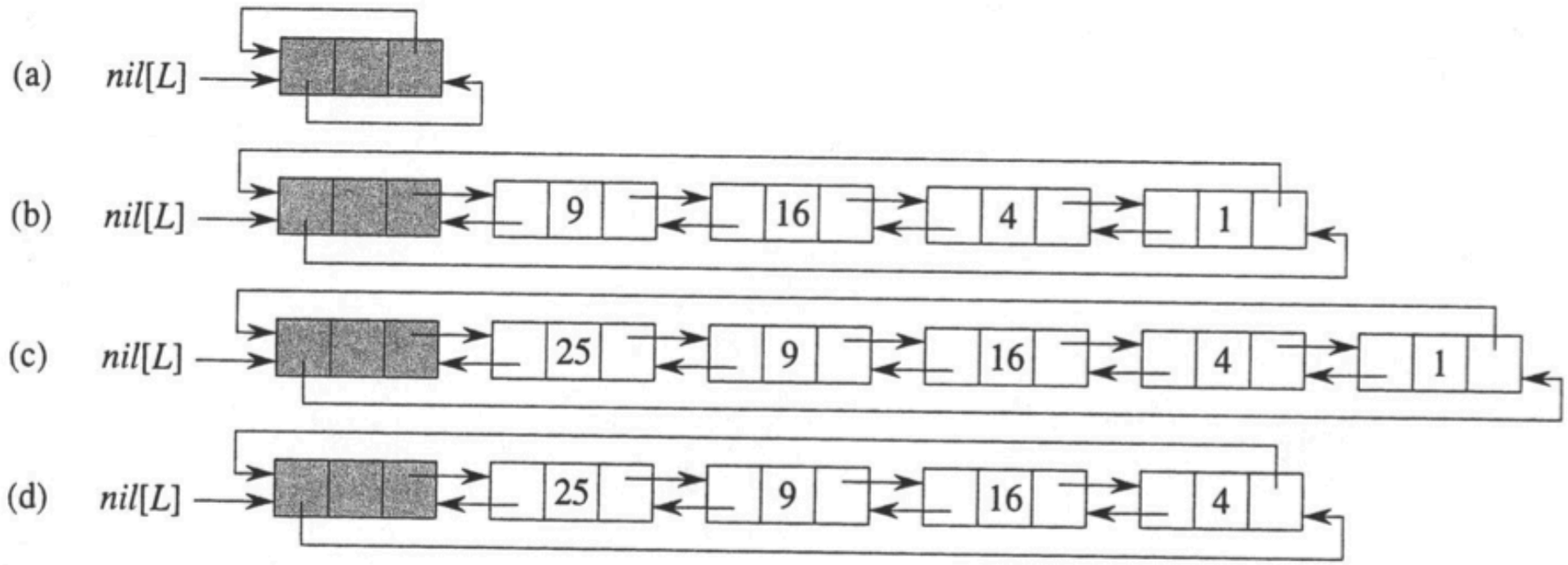
Laufzeit: O(n)

```
LIST-SEARCH(L, k)
1  x ← kopf[L]
2  while x ≠ NIL und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

Laufzeit: O(1)

```
LIST-DELETE(L, x)
1  if vorg[x] ≠ NIL
2     then nachf[vorg[x]] ← nachf[x]
3     else kopf[L] ← nachf[x]
4  if nachf[x] ≠ NIL
5     then vorg[nachf[x]] ← vorg[x]
```

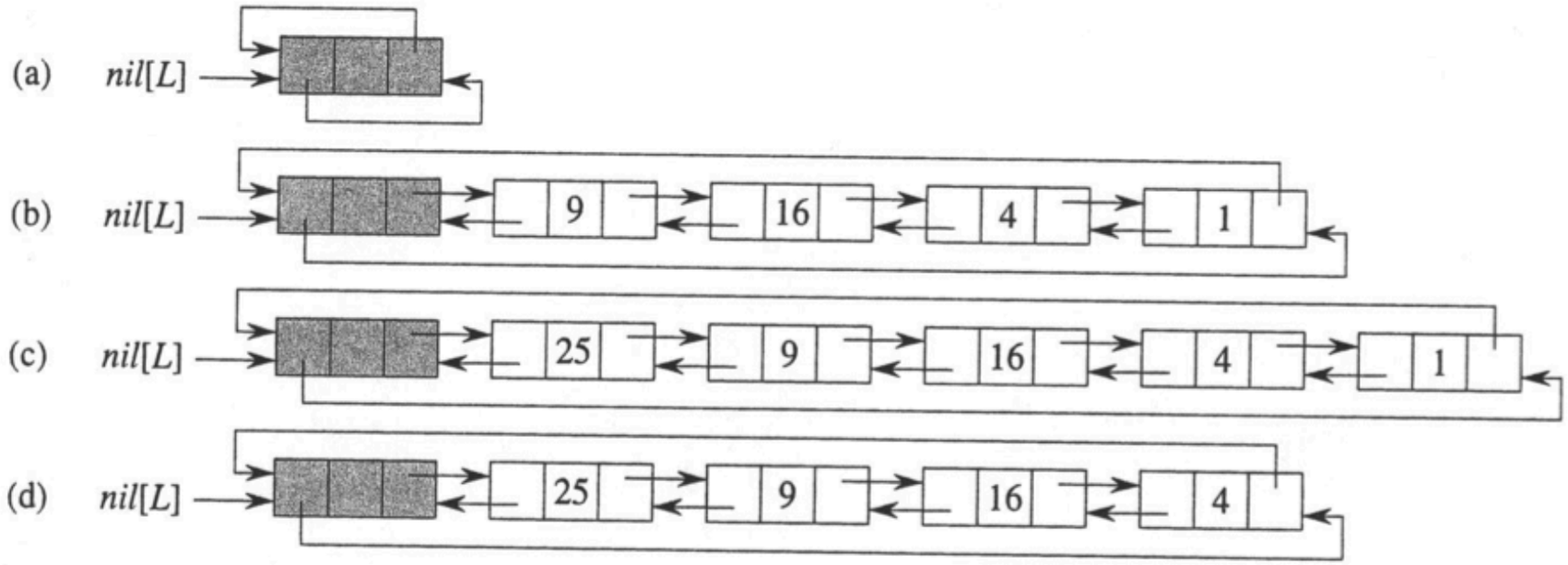
Alternative: Zyklische Struktur mit "Wächter" nil[L]



LIST-INSERT′(L, x)

1 $nachf[x] \leftarrow nachf[nil[L]]$
2 $vorg[nachf[nil[L]]] \leftarrow x$
3 $nachf[nil[L]] \leftarrow x$
4 $vorg[x] \leftarrow nil[L]$

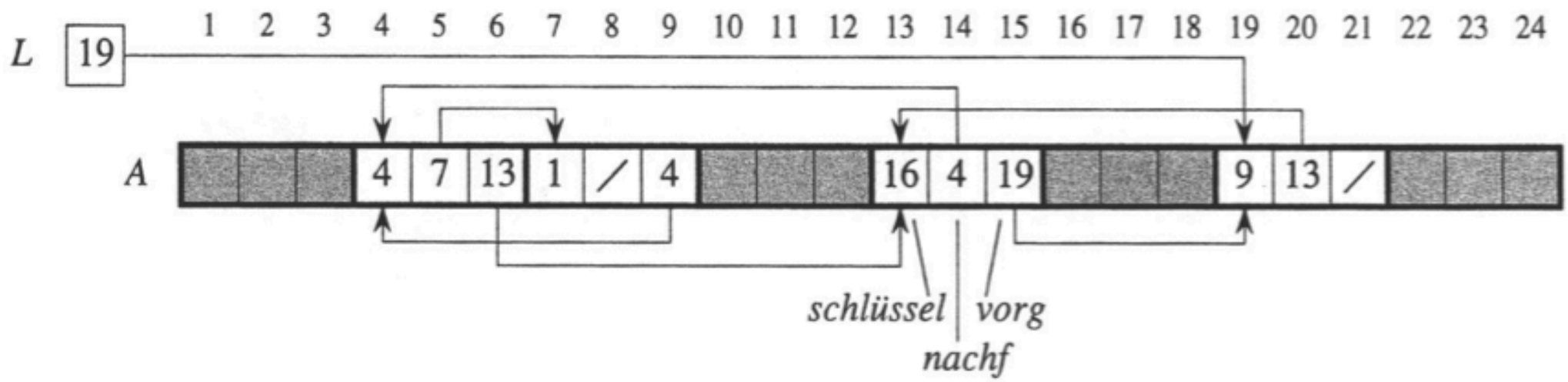
Alternative: Zyklische Struktur mit Wächter "nil[L]"

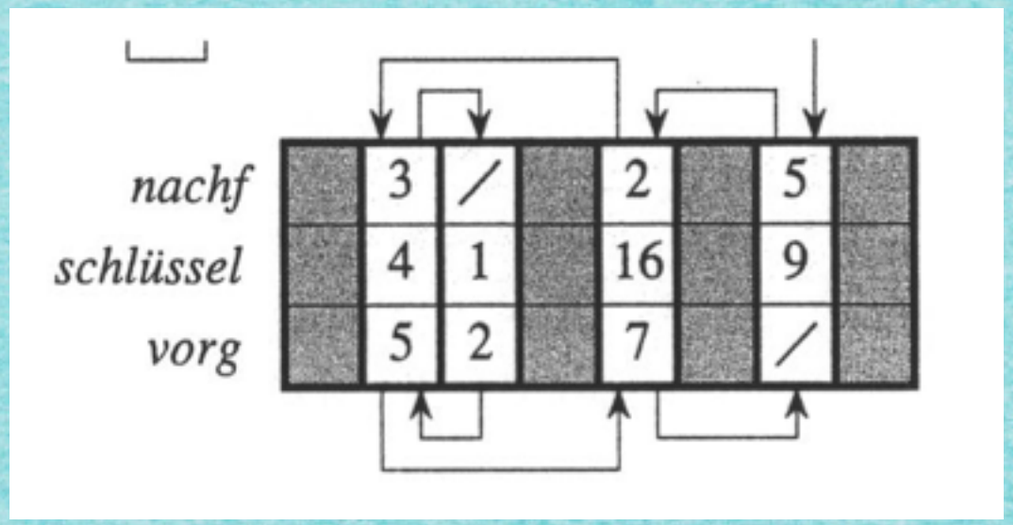



```
LIST-SEARCH'(L, k)
1  x ← nachf[nil[L]]
2  while x ≠ nil[L] und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

```
LIST-DELETE'(L, x)
1  nachf[vorg[x]] ← nachf[x]
2  vorg[nachf[x]] ← vorg[x]
```

Speicherung kann irgendwo erfolgen!





4.4 Binäre Suche

<u>Aufgabenstellung:</u>

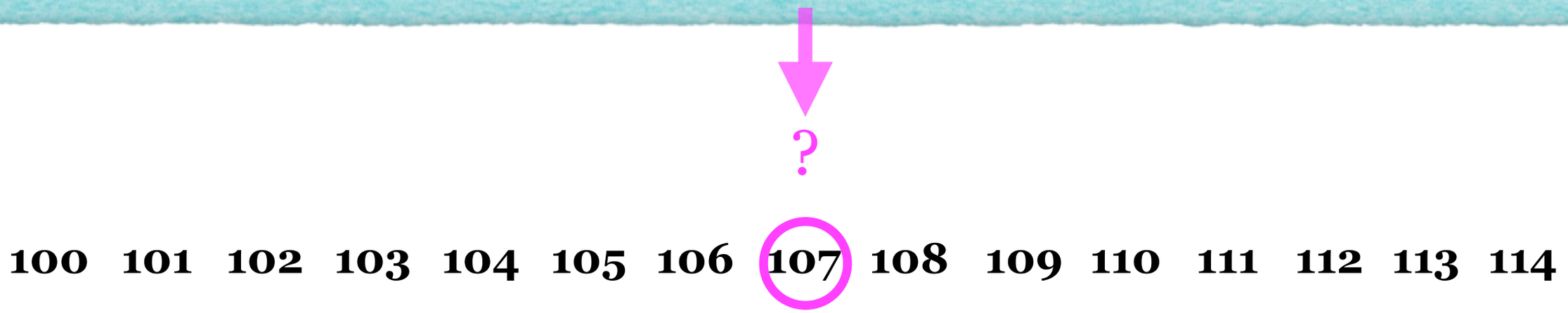
- ***Rate eine Zahl zwischen 100 und 114!***

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

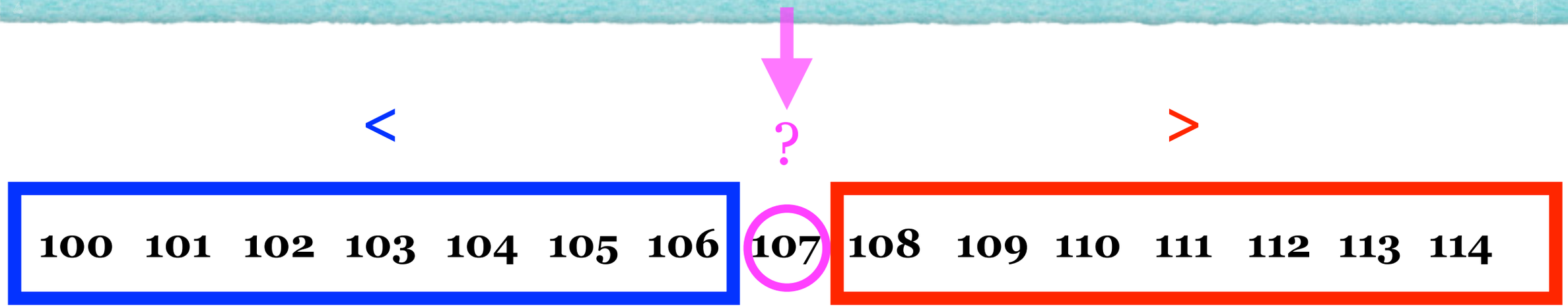
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

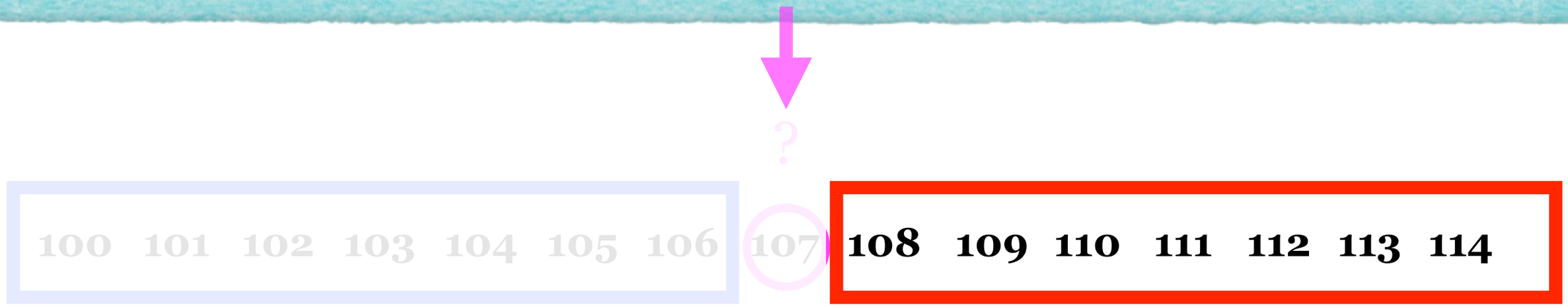
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

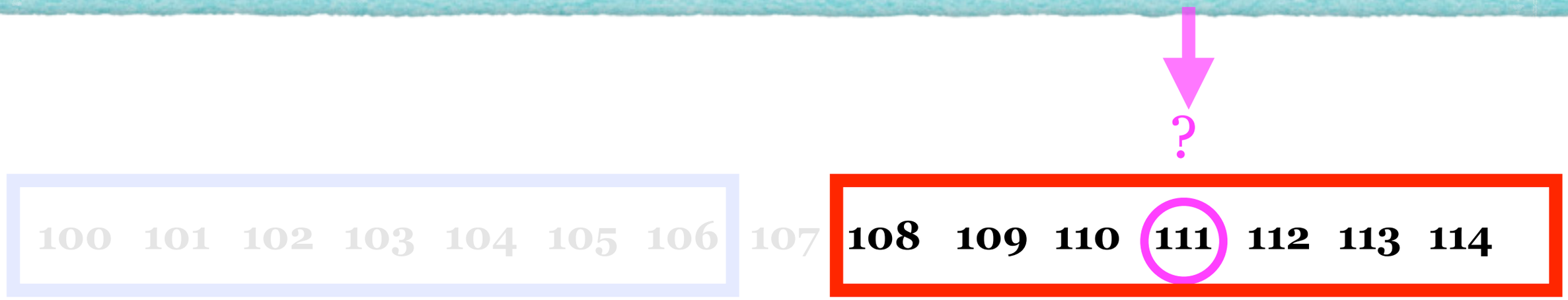
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

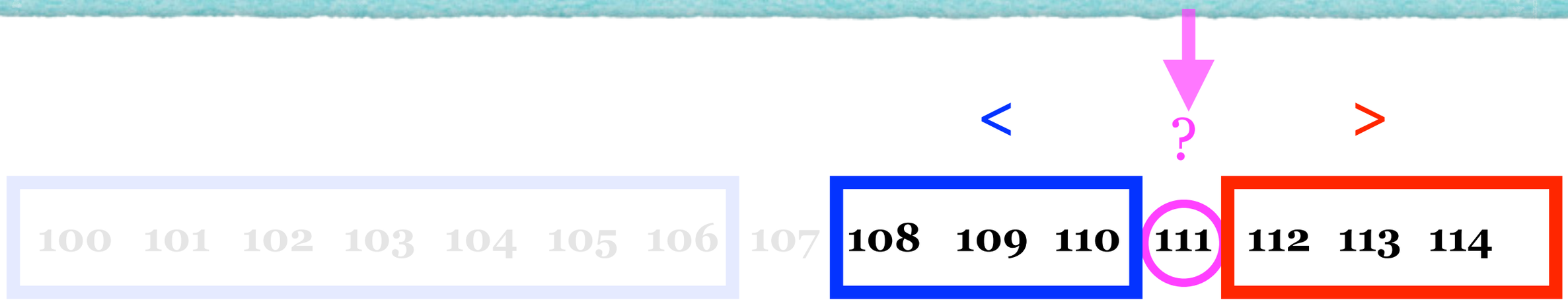
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

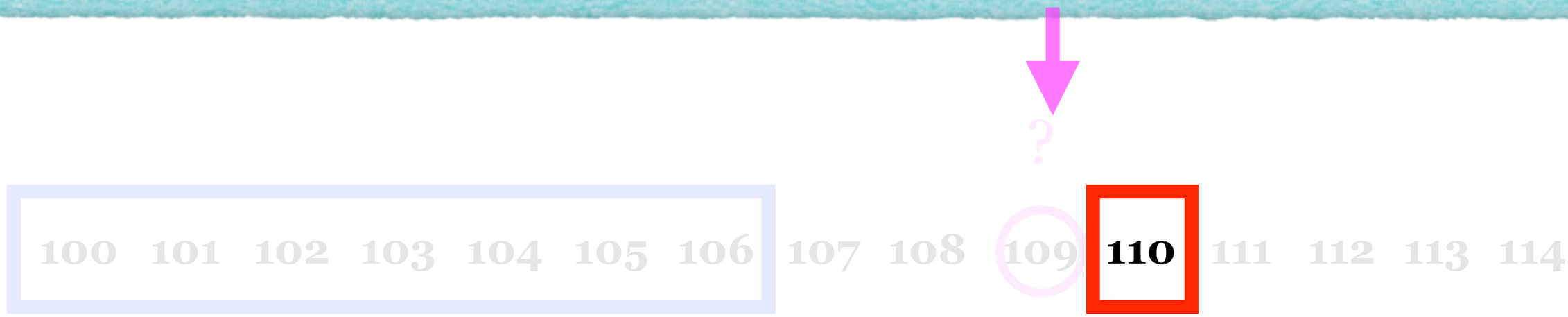
<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

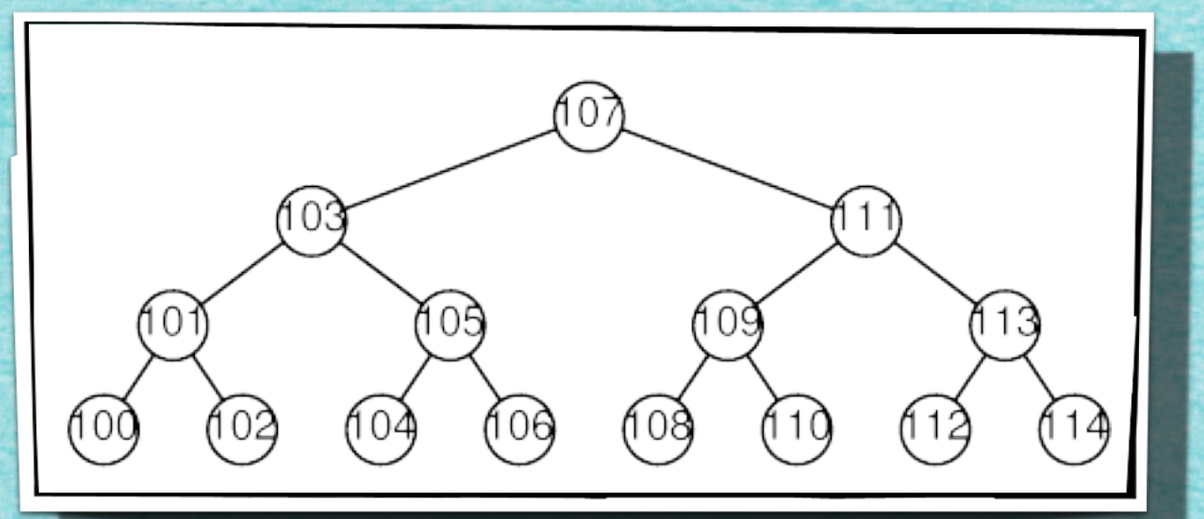
<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

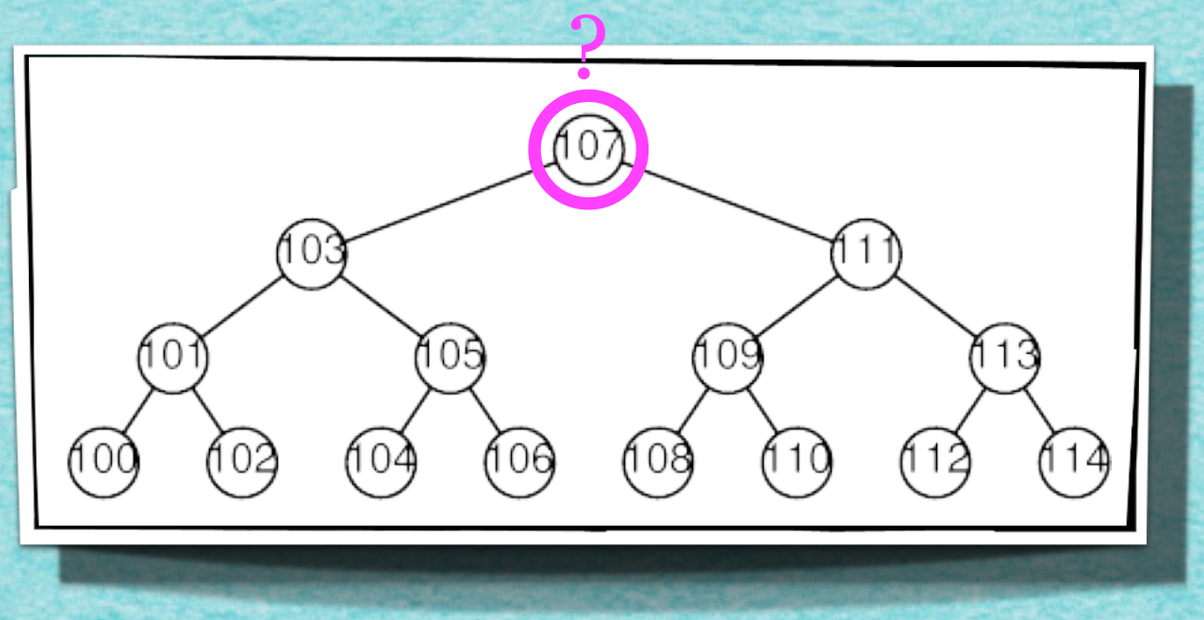
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

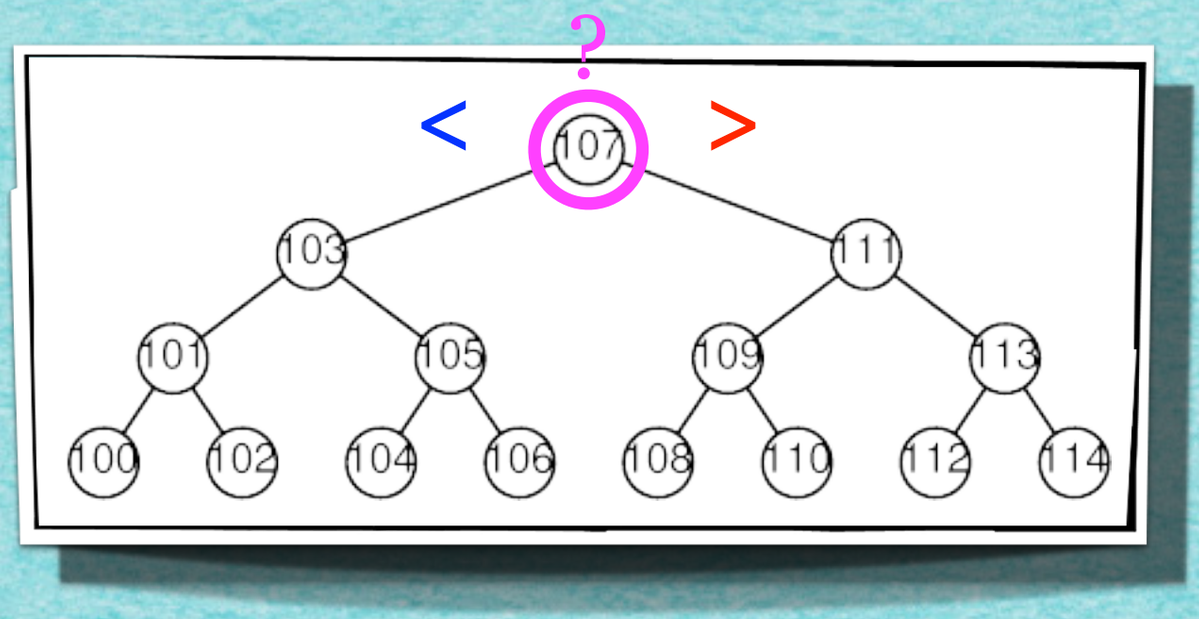
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

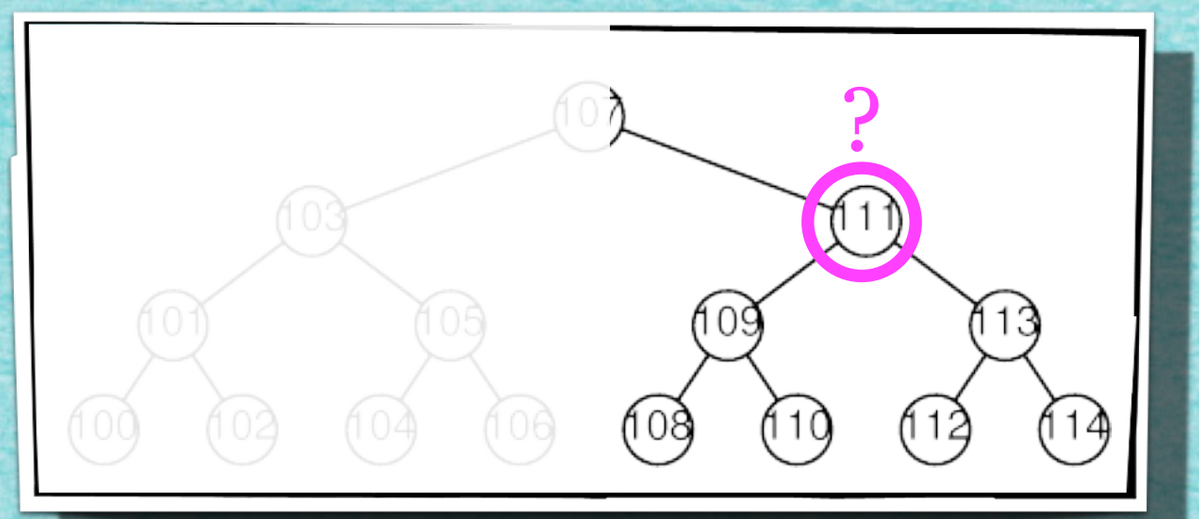
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

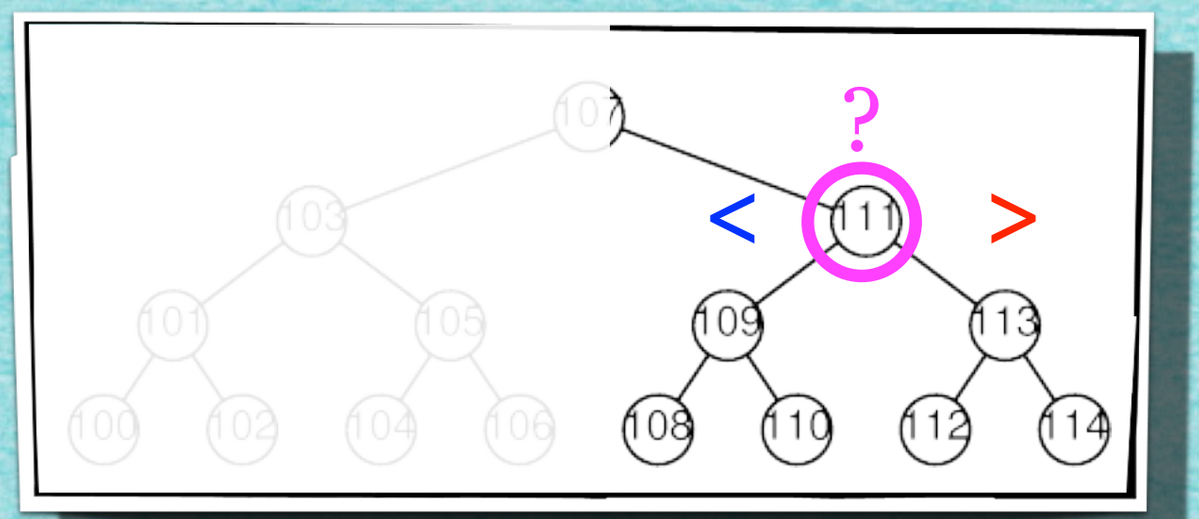
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

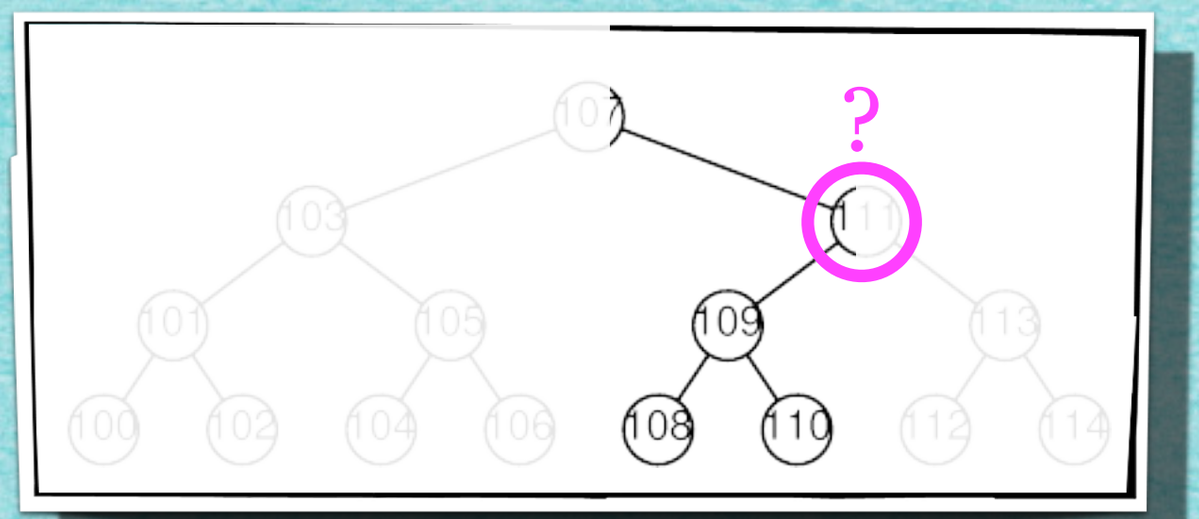
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

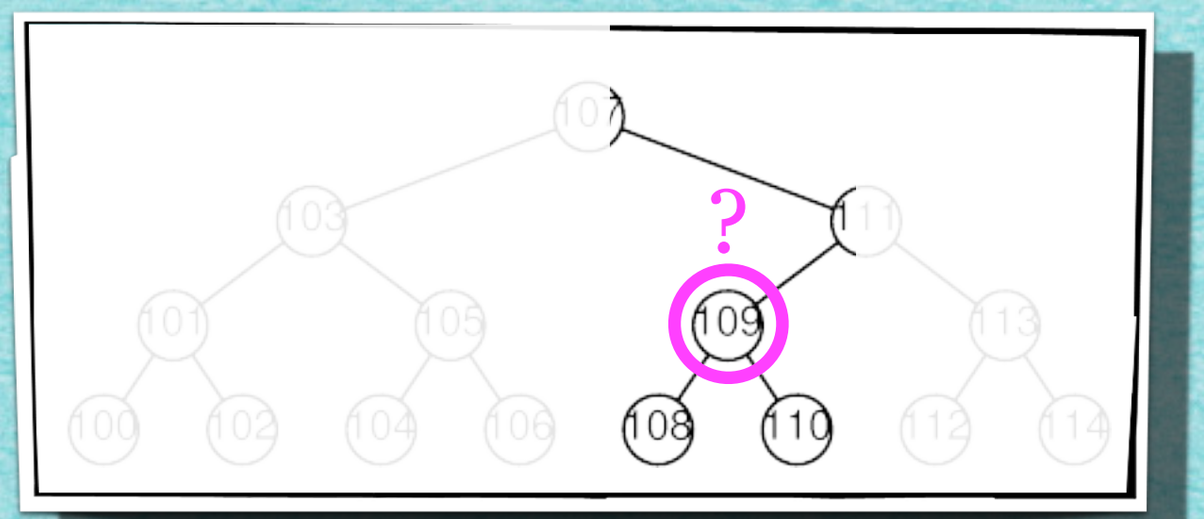
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

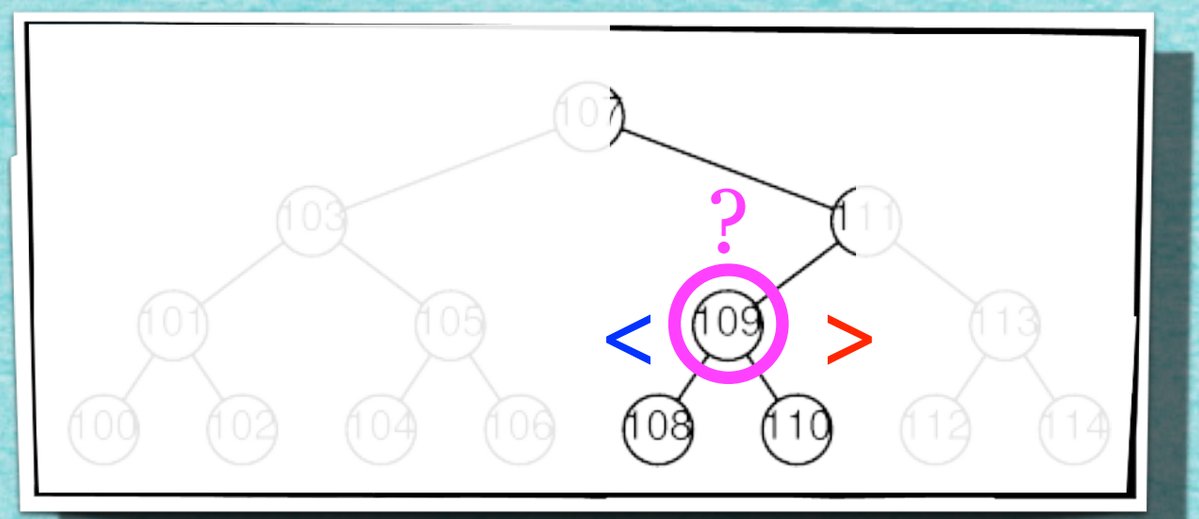
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

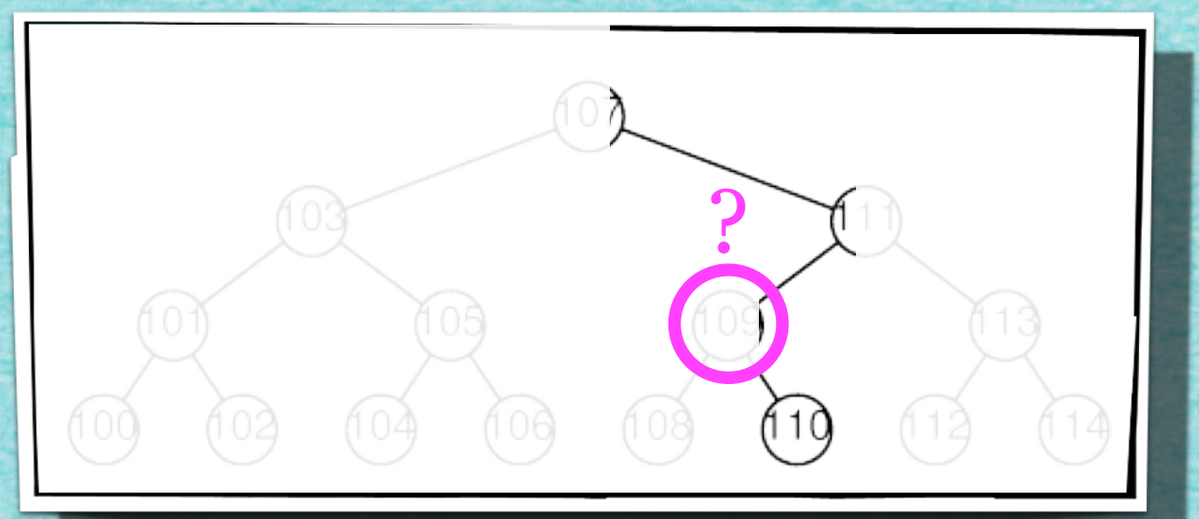
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

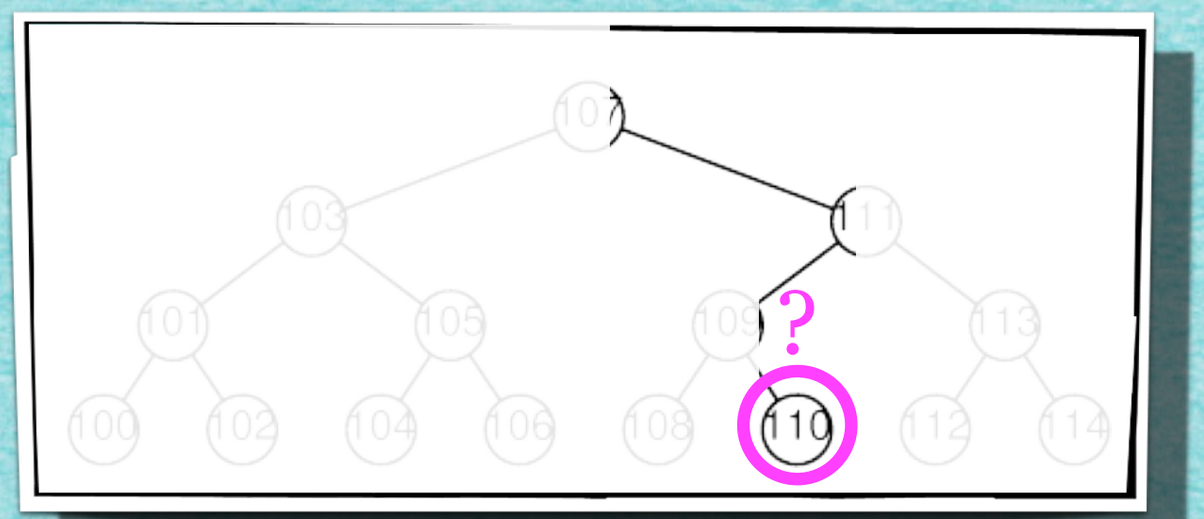
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {
    1.1. MITTE:= $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$
    1.2. IF (S[MITTE]=WERT) THEN
        1.2.1. RETURN MITTE
    1.3. ELSEIF (S[MITTE]>WERT) THEN
        1.3.1. RECHTS:=MITTE-1
    1.4. ELSEIF
        1.4.1. LINKS:=MITTE+1
    }
2. RETURN "WERT nicht gefunden!"

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

		LINKS											RECHTS	
i		1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]		1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

		LINKS					MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
								MITTE	LINKS						RECHTS
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96	

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS			MITTE			RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS			MITTE			RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS		RECHTS	MITTE			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

RETURN 9

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS												RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS					MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS					RECHTS	MITTE							
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

LINKS

MITTE

WERT=7

RECHTS

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

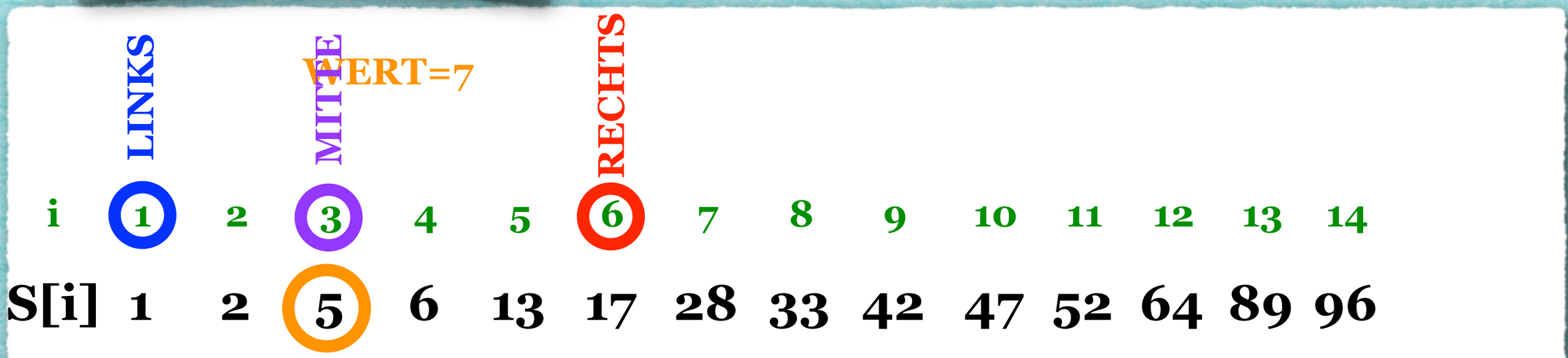
```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***



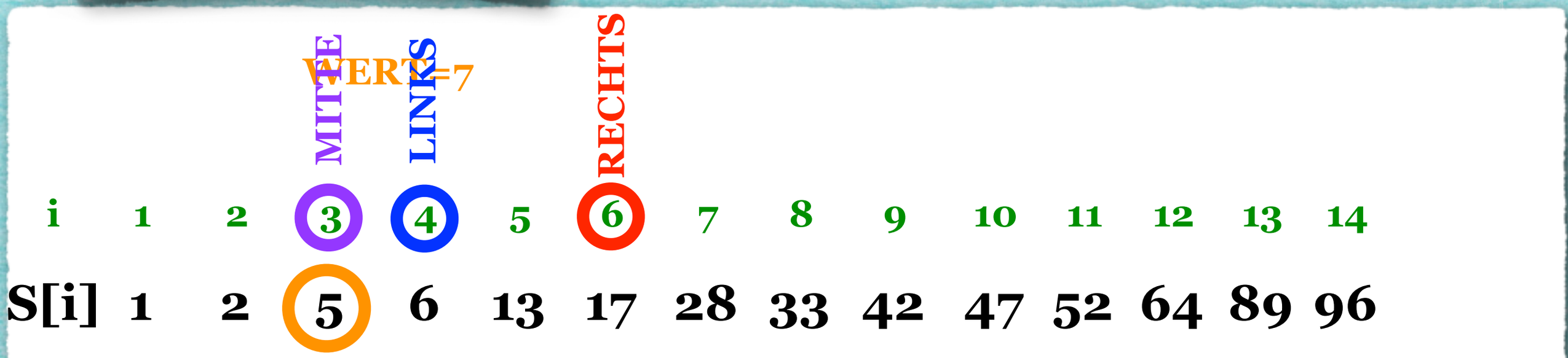
```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***



```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS	MITTE	RECHTS									
i	4	5	6	7	8	9	10	11	12	13	14	
S[i]	6	13	17	28	33	42	47	52	64	89	96	

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS	MITTE	RECHTS										
i	4	5	6	1	2	3	7	8	9	10	11	12	13	14
S[i]	6	13	17	1	2	5	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

i	1	2	3	4 (RECHTS)	5 (MITTE)	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

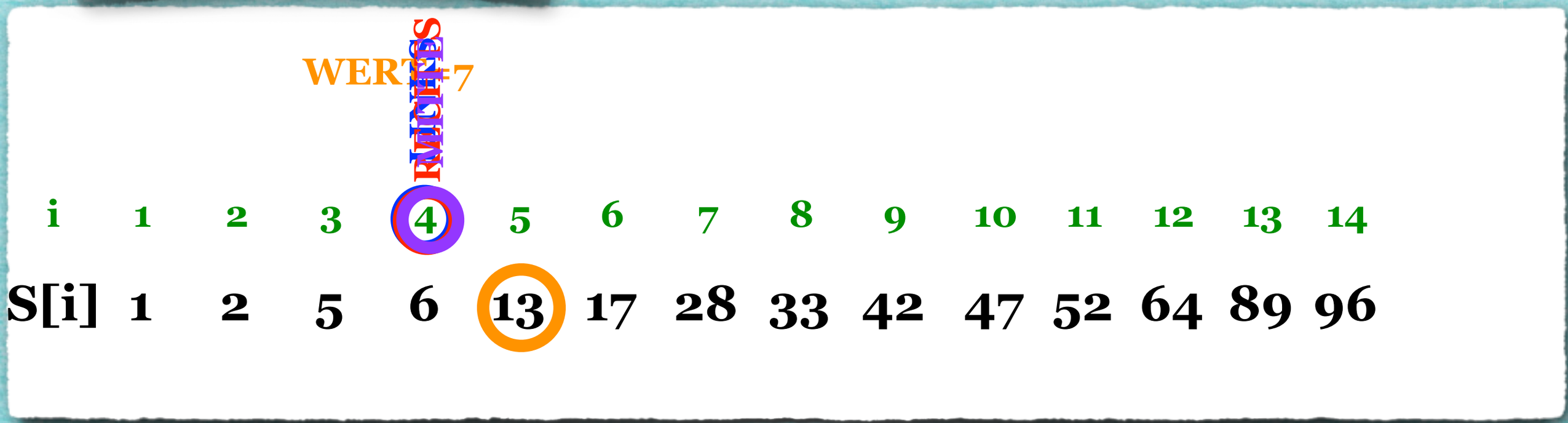
```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***



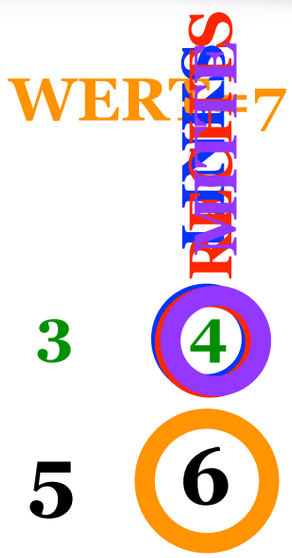
```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

RECHTES

LINKS

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	RECHTES				LINKS									
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

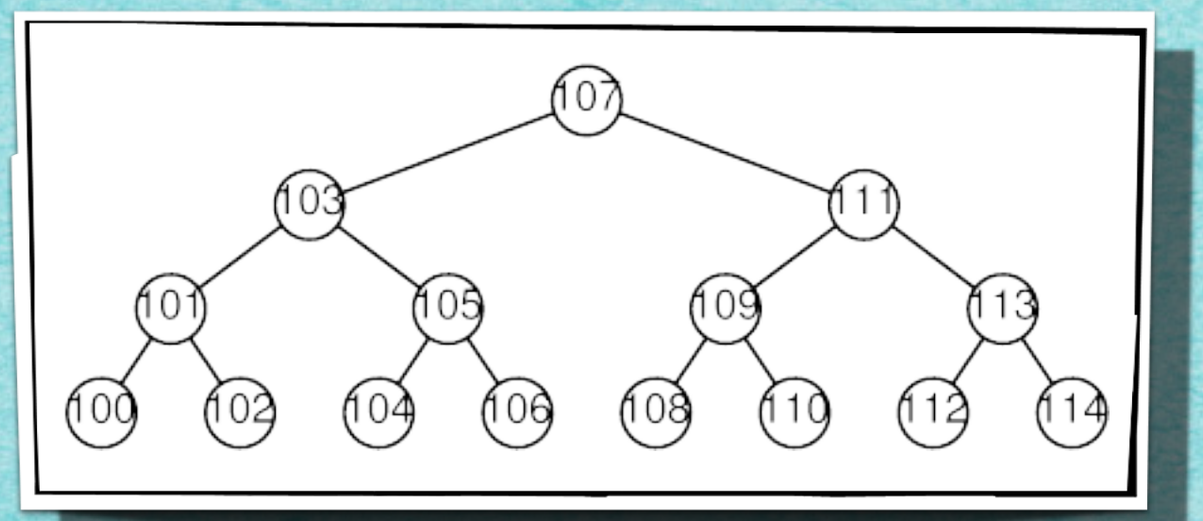
WERT nicht gefunden!

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten (für RECHTS>LINKS).

Beweis:

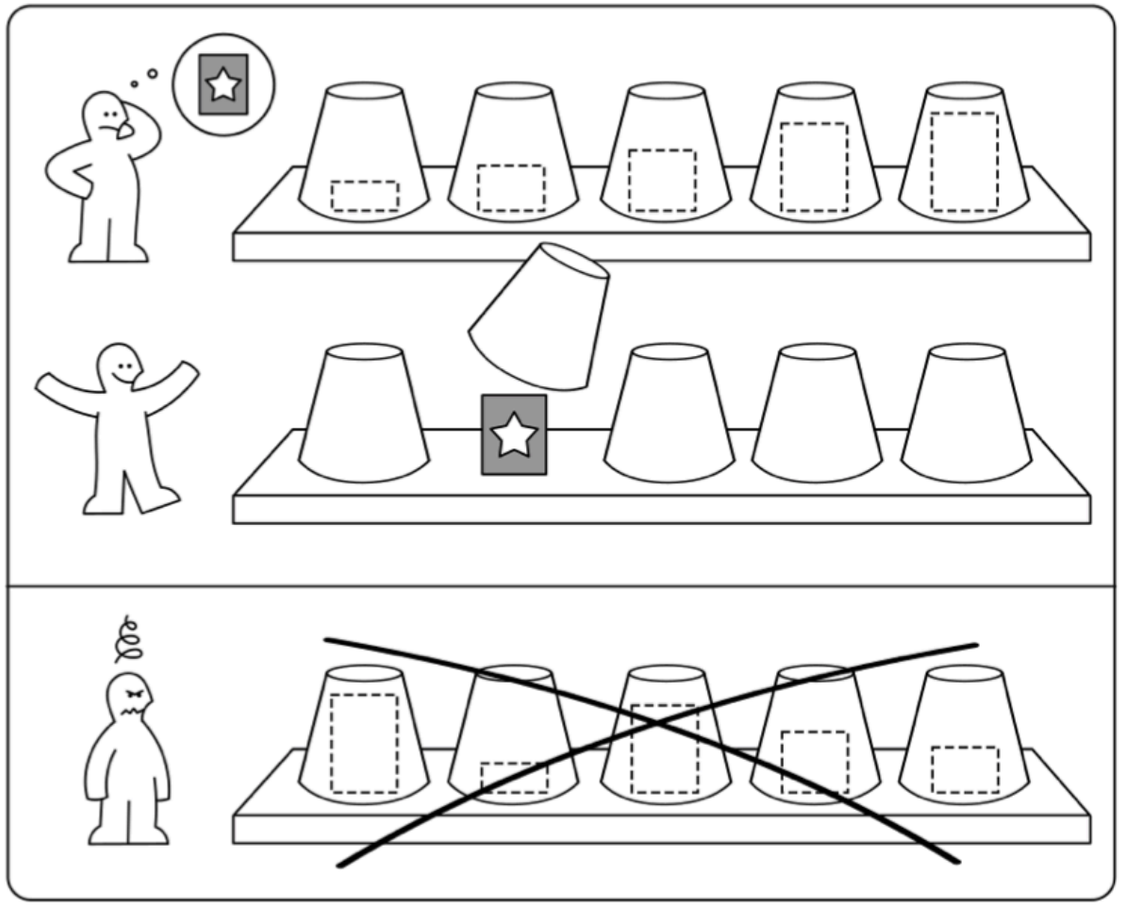
Selbst!



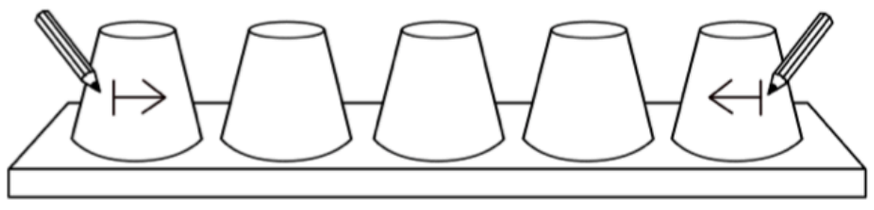
BINÄRY SEARCH

idea-instructions.com/binary-search/
v1.1, CC by-nc-sa 4.0

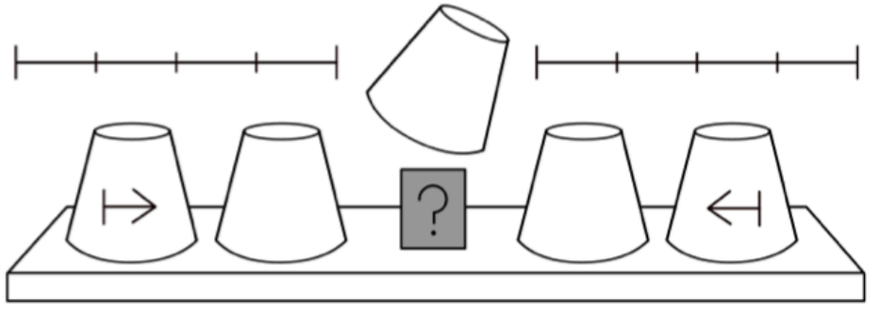
IDEA



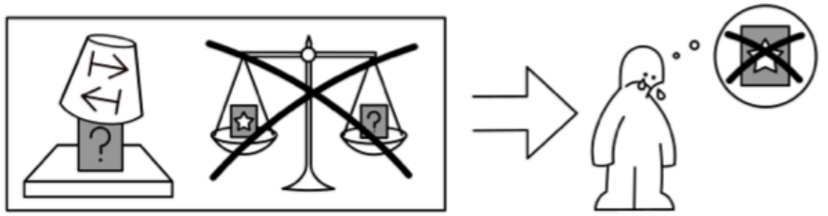
1



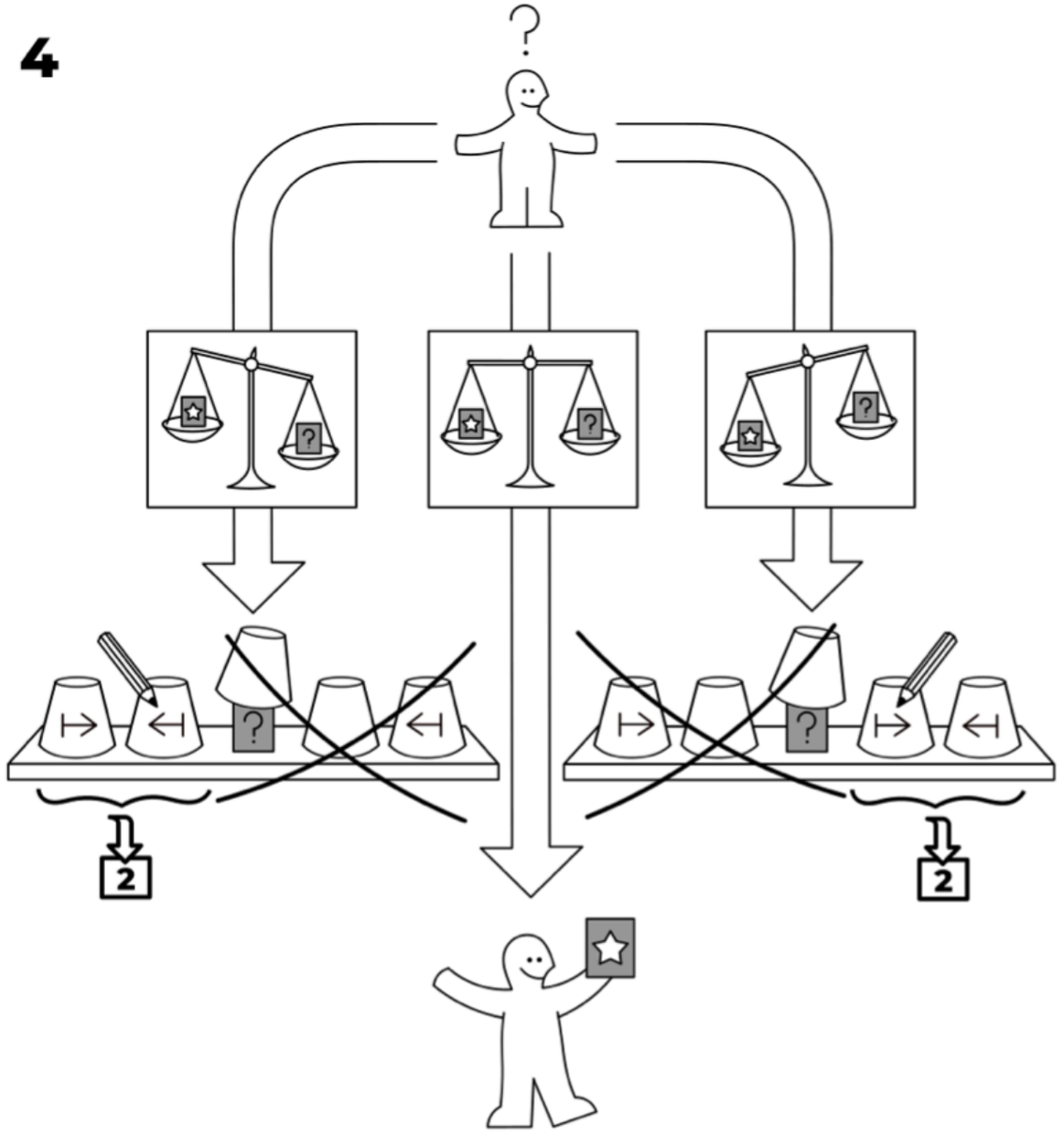
2



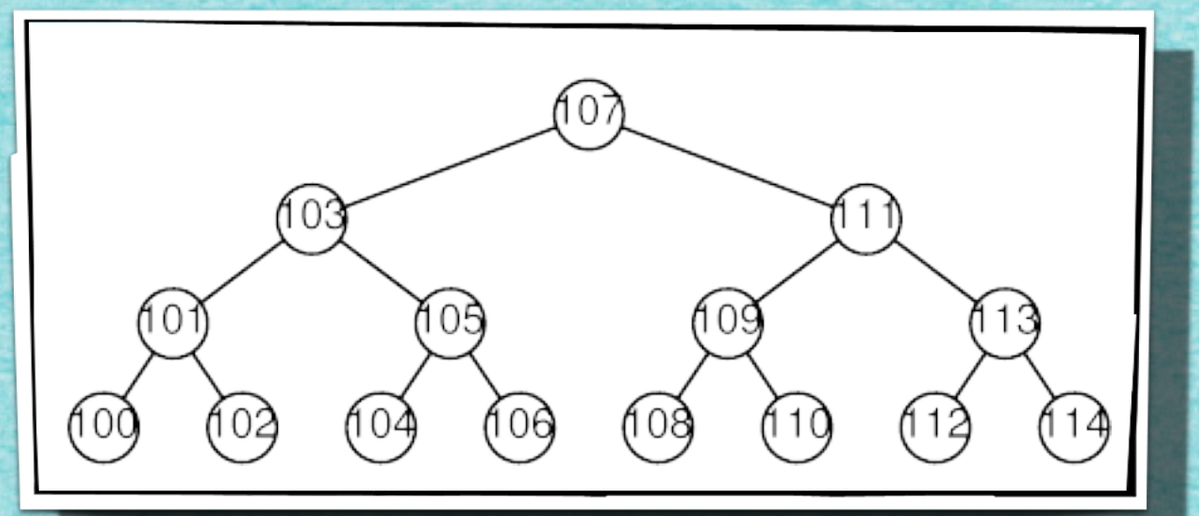
3



4



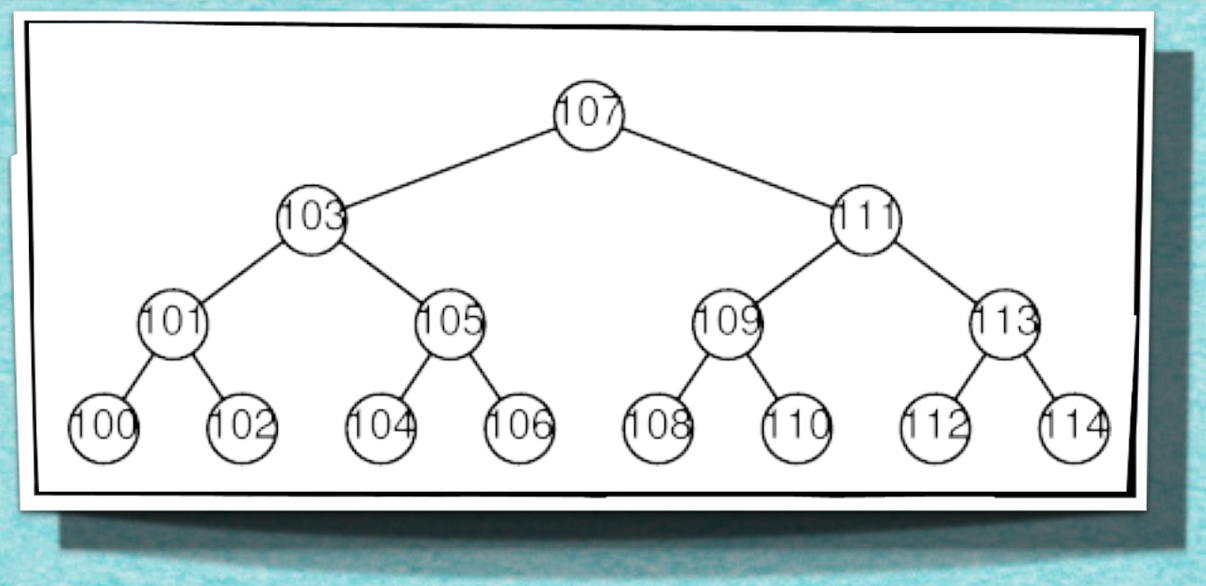
4.5 Binäre Suchbäume



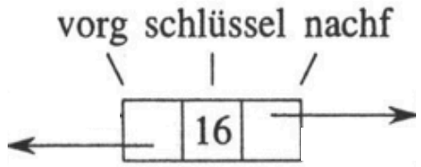
4.5 Binäre Suchbäume

Ideen:

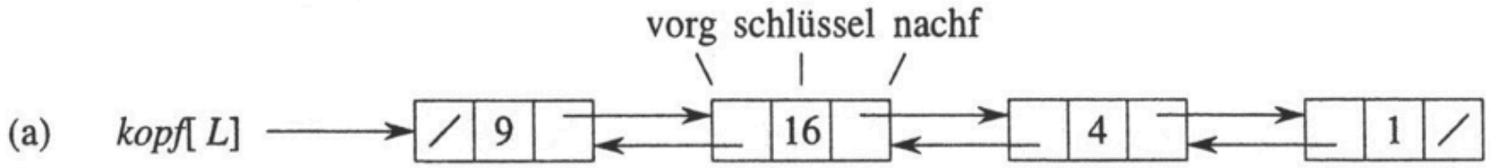
- ***Strukturiere Daten wie im möglichen Ablauf einer binären Suche!***
- ***Erziele logarithmische Zeiten!***



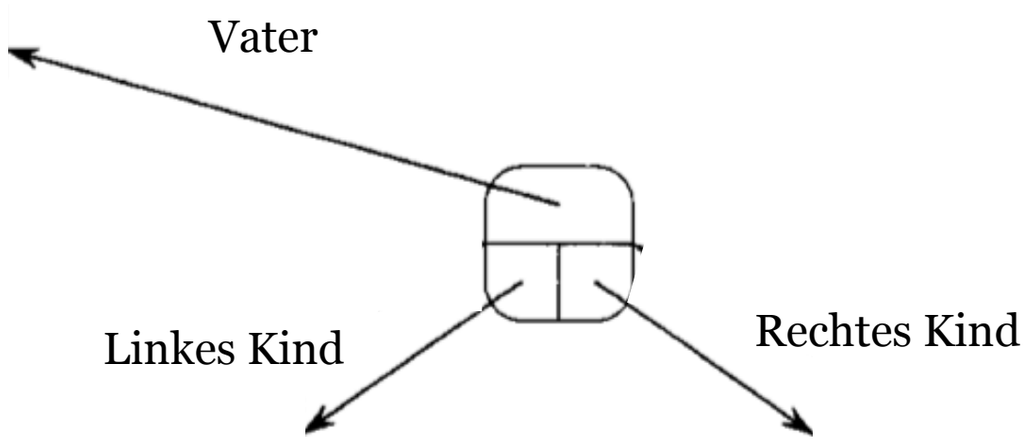
Struktur einer doppelt verketteten Liste



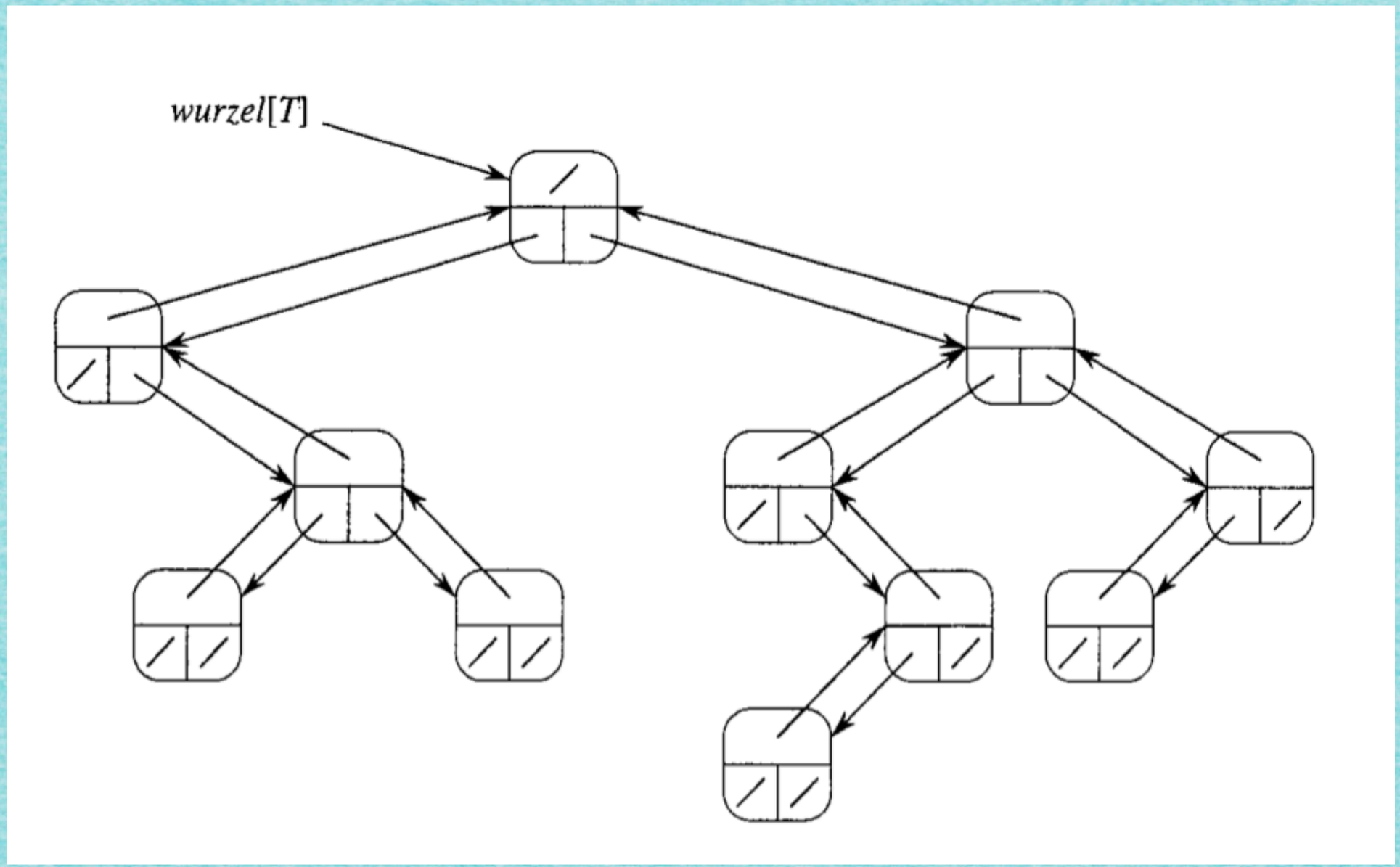
Struktur einer doppelt verketteten Liste



Binärer Suchbaum



Binärer Suchbaum



Außerdem wichtig: Struktur der Schlüsselwerte!

Mehr demnächst!

s.fekete@tu-bs.de